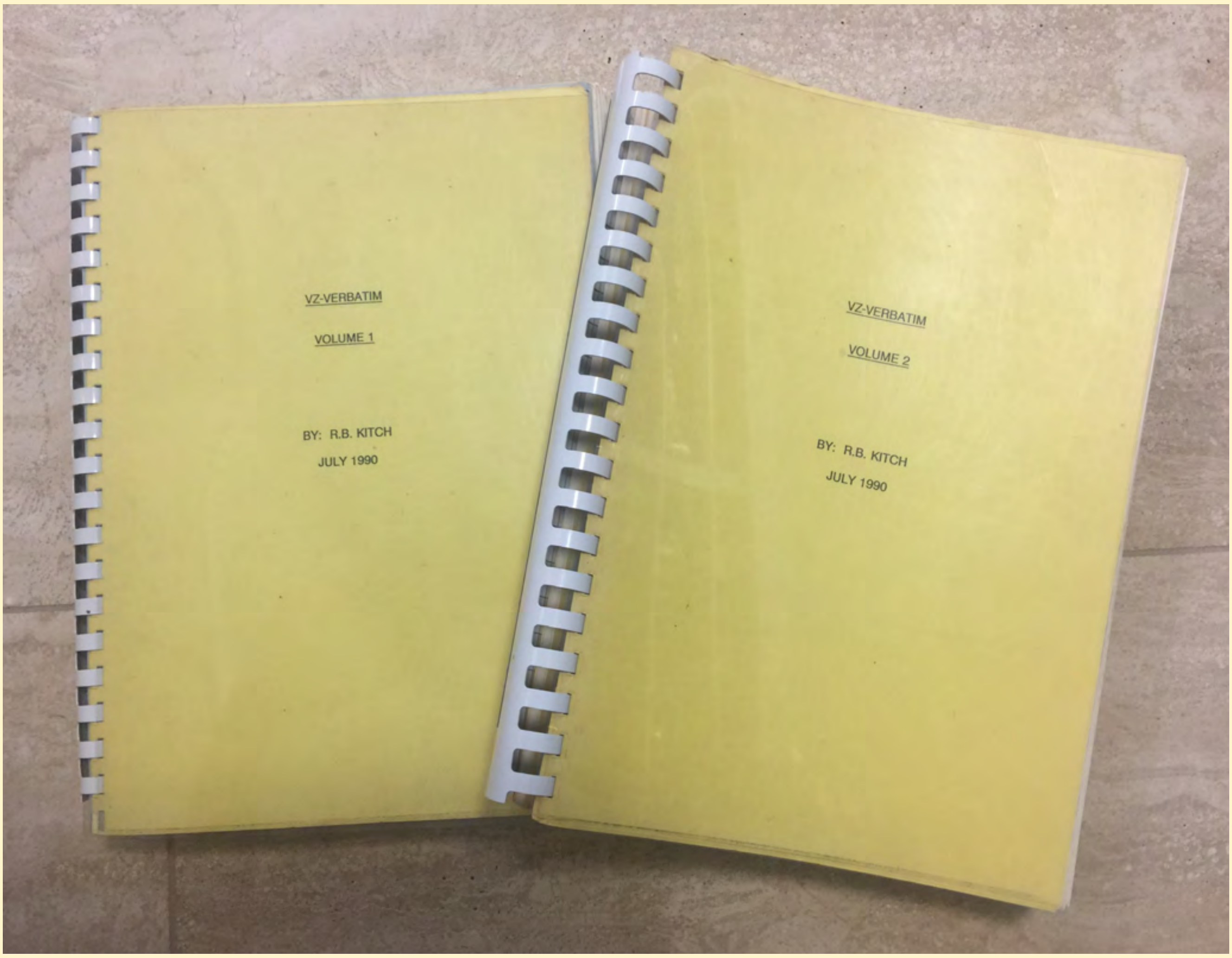
Bob Kitch 1990

VZ–VERBATIM

(A Collection of Magazine and
Technical Articles for
VZ Computers 1981 to 1990)

Volume 1 Software
Utilities, Games & Business



Compiled by Bob Kitch 1981 to 1990
Scanned March 2021
Brisbane Australia for All VZ Users
E: rbkitch@hotmail.com
M: +61 (0) 400 083 465

COMPILERS GUIDE FOR VZ USERS

Bob Kitch
Brisbane March 2021

***VZ-Verbatim** is a research resource for the DSE VZ00 and VZ300 micro-computers marketed in Australasia during the 1980's - in the pre-PC and post-TRS80/System 80 eras. Many young (and old) computer users cut their digital teeth on these Z80-based machines. A number of VZ User Groups also sprang up, held meetings and produced Newsletters. There was a huge thirst for knowledge, enthusiasm, learning, coding and general learning about "things digital" centred upon the VZ.*

All of the information in this compilation is long out-of-print and quite difficult to obtain. It may not be sold or recompiled into any other format without my express permission. Note the highly practical electronic and computing information that was offered in technical magazines of this era.

An information companion to VZ-Verbatim is the "Bob Kitch's VZ Scrap Book" that contains thirty technical contributions I made to magazines and various User Groups Newsletters during 1985 to 1990. Approximately 25 BASIC and ASSEMBLER ASCII listings are provided in that Directory. These articles were about learning and encouraging VZ Users to develop digital skills and interests.

VZ-Verbatim was a last-Century response to an information demand to encourage a new generation of digital enthusiasts in the pre-WWW era. It was compiled during 1985 to 1990 but with articles going back to 1981. The original format was as loose A4 Master Sheets wherein specific photocopies could be returned by snail mail to interested and puzzled VZ Users. As interest in 8-bit computers waned in early 1990's, a lone copy of VZ-Verbatim (as two volumes) was made (pictured on cover). It is in the last month these volumes have come to hand, been scanned at 400dpi and converted to pdf's.

As a late incarnation of the 8-bit microcomputer era, the Video Technology/DSE VZ200/300 was highly influential in homes throughout Australasia and under other names elsewhere in the World. A fair level of interest remains amongst enthusiasts in Vintage Computer Groups and Emulators Users. A number of now middle-aged men, were young enthusiasts that learned about computing in the 1980's and still use the VZ for largely nostalgic reasons. I note that a remarkable number of these young enthusiasts are now employed in the IT industry. These enthusiasts are instrumental in maintaining Z80 emulators and hardware, have added more convenient I/O peripherals than the contemporary cassette and floppies and have added memory capabilities beyond 64K. Tape and disk software has been converted to more durable digital formats.

Preserving and providing ready access the "Lump" of VZ technical information, software images, operating hardware and emulators is regarded as a priority. This compilation is part of that "VZ Lump".

Bob Kitch
Brisbane, Queensland, Australia
E: rbkitch@hotmail.com
M: +61 (0) 400 083 465

Structure of Volumes

Following on the blue pages is a complete listing of all articles contained within Volumes 1 and 2.

This is shown in the original list format that was frequently updated and circulated to VZ Users.

Pages 12 to 14 of that list is included for completeness. These pages are a list of books on BASIC, Assembler and the Z80. Most of these are available on-line as e-books in pdf format.

The yellow pages detail the various sections within the volumes.

Volume 1 contains software articles categorised as

Utilities
Games
Business

Volume 2 contains

Hardware Peripherals
Software Reviews
Software Advertisements
Hardware Reviews
General Programming
DSE Technical Bulletins.

These volumes were derived from 400dpi scans of second generation photocopies of the original bound articles and were delivered in Adobe Acrobat pdf format.

Using Adobe Acrobat Pro 2017 each page was edited and enhanced involving

- character recognition to provide editable text and images
- text and images de-skewed
- font replaced with document fonts for sharpening

VZ-VERBATIM

VOLUME 1

BY: R.B. KITCH

JULY 1990

LISTING OF VZ200/300 MAGAZINE ARTICLES

AS AT 31 JULY 1990

Since its introduction in early 1983, over three hundred articles on the VZ-200 and 300 have appeared in the magazines. Some articles review the hardware and others describe peripherals. Some excellent games have been published and a very useful set of utility routines has emerged.

This bibliography for the VZ computer is a must for the serious VZ-User.

Compiled by:-

R.B. KITCH, 7 Eurella St., Kenmore, Qld. 4069. Phone: (07)378-3745.
PLEASE ADVISE OF ANY ADDITIONAL ARTICLES ..or.. CHANGES, ALTERATIONS OR BUGS IN LISTINGS to assist other Users.

The numbers in brackets are the number of sheets in each article. A dash (-) indicates that the article is on the same sheet as the item above.

If Users wish to obtain copies of the articles referred to in this bibliography, they may -

i) contact me for copies ..or..
ii) buy back copies of the magazine from the distributor ..or..
iii) borrow from your local library.

I can supply copies FOR YOUR OWN USE ONLY at 20c. per sheet. Kindly add postage to your request as follows:

No. of Sheets	Qld.	Interstate (Surface)
1 - 3	$0.41	$0.41
4 - 18	$0.95	$1.10
19 - 90	$1.90	$2.50
> 90	expensive!	

UTILITIES

Oct.	83	APC	52,4	BASIC program conversion. (Surya)	(2)
Jan.	84	APC	20-21	Beginners tips. (White)	(-)
Nov.	83	APC	57,9	Program conversion Pt. 2 (Surya)	(2)
Nov.	83	APC	89-95	BASIC converter chart. (Surya)	(7)
Feb.	84	APC	140-1	Program conversion Pt. 2 (Surya)	(2)
Mar.	84	APC	42-3	Program conversion - Apple II (Surya)	(2)
Apr.	84	APC	71-2	Program conversion - TRS 80/System 80 (Surya)	(1)
May	84	APC	75-6	Program conversion - Atari (Surya)	(2)
Jun.	84	APC	67	Program conversion - Sinclair (Surya)	(1)
Jul.	84	APC	129-30	Program conversion - BBC (Surya)	(2)
Mar.	84	ETI	63	More functions for the VZ-200. (Olney)	(1)
Apr.	85	ETI	117	Notes and errata for Olney.	(-)
Jul.	84	BB	56	Some more routines. (Middlemiss)	(1)
Jul.	84	M80	3-4	VZED - three new functions.	(1)
Aug.	84	M80	2	VZ-200 output latch.	(1)
Aug.	84	M80	9,15,16	Memory peek VZED. (Carson)	(1)
Aug.	84	M80	3-4	Microsoft ROM BASIC Level I bug.	(1)
Apr.	85	APC	97	VZ-200 bug. (Tritscher)	(-)
Aug.	85	APC	31	VZ bug. (Tritscher)	(-)
Aug.	84	APC	94	VZ-200 moving message and trace. (Batterson)	(1)
Nov.	84	APC	76	Trace function. (Breffit)	(-)
Nov.	84	APC	125	VZ-200 correction. (Kelly)	(-)
Sep.	84	CI	19	VZ200 Input. (Woolf)	(1)
Sep.	84	BB	63	Poking extra functions. (Clark & Hill)	(1)
Oct.	84	ETI	135-7	Extending VZ-200 BASIC. (Olney)	(3)
Nov.	84	APC	125-6	TRON/TROFF function for VZ-200. (Thompson)	(1)
Nov.	84	APC	208-12	MON-200 machine code monitor. (Stamboulidas)	(5)
Nov.	84	PCG	55-56	Lprinter. (Quinn)	(2)
Nov.	84	PCG	suppl.	VZ-200 reverse video.	(1)
Dec.	84	BB	64	Enlarged characters. (Velde)	(1)
Feb.	85	APC	171	BASIC understanding. (Hobson)	(1)
Feb.	85	APC	20	VZ-200 into puberty - Olney's extended BASIC.	(1)
Feb.	85	ARA	19-26	Calculating grey line. (Baker)	(6)
Mar.	85	CI	12-14	Renumber. (Marsden)	(3)
Apr.	85	PCG	62-64	Find. (Stamboulidas)	(3)
Apr.	85	APC	19	Use of RND in dice and card games. (Holland)	(1)
Apr.	85	APC	103	VZ variable definition. (Stamboulidas)	(1)
Apr.	85	APC	95	Variable GO TO on VZ. (Olsen)	(1)
Jul.	85	APC	176	Correction to VZ variable GO TO.	(-)
May	85	APC	52-3	Lysco support for VZ-200. (Young)	(1)
May	85	ETI	99-101	VZ-200 hardware interrupt. (Olney)	(3)
May	85	APC	110	Background VZ. (Williams)	(1)
Aug.	85	APC	130	VZ-200 instant colour. (Willows)	(-)
Aug.	85	APC	130-3	Reversed REM. (Quinn)	(1)
Sep.	85	APC	145	Real-time clock. (Griffin)	(1)
Oct.	85	APC	218	APC benchmark BASIC programs.	(1)
Oct.	85	APC	147	VZ deletions. (Quinn)	(1)
Nov.	85	APC	189	VZ EDITOR/ASSEMBLER tips. (Lam)	(1)
Nov.	85	ETI	94-5	Olney's Level II BASIC for VZ200/300. (Rowe)	(2)

Jan.	86	APC	83,5	VZ user graphics.	(1)
Feb.	86	APC	127	Machine language calls.	(1)
Mar.	86	APC	chart	APC BASIC converter chart 1986.	(8)
Mar.	86	YC	103-5	VZ-200 cassette inlays. (Dutfield)	(3)
May	86	APH	54-55	VZ and photography. (Kohen)	(2)
Jun.	86	APC	209	VZ pause.	(1)
Aug.	86	ETI	86-89	VZ software mods. (CHIP-8 Editor) (Griffin)	(3)
Oct.	86	ETI	28-33	VZ CHIP-8 Interpreter. (Griffin)	(5)
Sep.	86	AEM	89-92	Screen handling on VZ. Part I. (Kitch)	(4)
Oct.	86	AEM	110-112	Screen handling on VZ. Part II. (Kitch)	(4)
Oct.	86	AEM	113,4,21	Reference list of VZ articles. (Kitch)	(2)
Oct.	86	ETI	47	Labeller. (Gallagher)	(1)
Oct.	86	ARA	38-42	Amateur radio logger. (Johnson)	(5)
Nov.	86	EA	35	Speaker enclosure calculator. (Allison)	(1)
Dec.	86	AEM	90-95	Memory mapping on VZ. (Kitch)	(6)
Mar.	87	AR	10-12	Feedline calculations. (Buhre)	(3)
Apr.	87	EA	100-101	Op amp noise. (Allison)	(2)
Apr.	87	ARA	20-24	Beam Headings. (Baker)	(5)
May	87	AEM	86-88	VZ Epson printer patch. (Taylor)	(3)
Jun.	87	AEM	74,75,79	VZ Epson printer patch Pt II.	(3)
Aug.	87	AEM	82-83	VZ expanded EPROM. (Meager)	(2)
-	88	BYC	88	Restore file. (Banks & Saunders)	(1)
-	88	-	-	B-file copier. (Buhre)	(1)
Feb.	88	ETI	70	String file name. (Hand)	(1)
Jul.	88	ETI	74	Disk directory dumper. (Tunny)	(1)
Oct.	88	ETI	124	CTRL-Break disabler. (Tunny)	(1)
Oct.	88	AEM	96-97	VZBUG. (Batger)	(2)
Nov.	88	ETI	120	Clock. (Tunny)	(2)
Feb.	89	ETI	118-119	DOS Hello (Tunny)	(1)
Feb.	89	ETI	119-120	Visisort (Sheppard)	(2)
Nov.	89	ETI	73	Restore (Rowe)	(1)
Nov.	89	ETI	73	Hex/dec conversion (Maunder)	(1)
Jan.	90	CBA	17-19	Beam headings (Baker)	(3)

GAMES

Nov/Dec	83	SYN	22-24	Projectile Plotting (Grosjean)	(2)
Dec.	83	APC	161-3	Missile Command. (Whitwell)	(2)
Feb.	84	BB	50-51	Caddy and Reaction Test. (Hartnell)	(2)
Jan.	84	YC	65	Graphic Sine Waves for VZ-200. (Nickasen)	(1)
Apr.	84	APC	178-80	Moon Lander. (Alley)	(2)
Jul.	84	APC	174-8	Blockout. (Pritchard)	(3)
Jul.	84	M80	7,22	Battleships. (Carson)	(1)
Jul.	84	M80	7,20,21	Junior Maths. (Carson)	(2)
Aug.	84	M80	9,16	Contest Log VZED. (Carson)	(1)
Aug.	84	M80	9,16,17	Dog Race VZED. (Carson)	(1)
Oct.	84	PCG	55-7	High Resolution Graphics Plotting. (Thompson)	(3)
Nov.	84	PCG	82	Tips for 'Ladder Challenge', 'Panik' and 'Asteroids'.	(1)
Jan.	85	PCG	54	POKE's to 'Ghost Hunter'.	(-)
-	85	BYC	146-7	Golf Simulation. (McCleary)	(2)
Mar.	86	CFG	4-5	Golf Simulation. (McCleary)	(-)
-	85	BYC	147	Knight's Cross. (Lucas)	(1)
Jan.	85	APC	129-31	Sketcher. (Leon)	(3)
Jan.	85	YC	88-89	Punch. (Rowe)	(2)
Jan.	85	PCG	44-48	Space Station Defender. (Shultz)	(5)
Feb.	85	CI	27-28	Lost. (Potter)	(2)
Mar.	85	YC	105-9	Decoy. (Rowe)	(2)
Mar.	85	CI	-	Mouse Maze. (Crandall)	(1)
Apr.	85	YC	160	Painter. (Daniel)	(1)
Apr.	85	PCG	65-7	Roadrace. (Thompson)	(3)
May	85	YC	106	Number Sequence. (Thompson)	(1)
May/Jun	85	PCG	63-7	Sketchpad. (Thompson)	(5)
Jun	85	YC	70	Morse Tutor program. (Heath)	(1)
Jan.	86	YC	150-1	Morse Tutor - again. (Heath)	(2)
Jul.	85	YC	81	Electric Tunnel. (Daniel)	(1)
Aug.	85	YC	114	Number Slide. (Daniel)	(1)
Oct.	85	PCG	47-52	Cube. (McMullan)	(6)
Oct.	85	YC	105-7	Yahtzee. (Thompson)	(3)
Mar.	86	APC	208-9	VZ Frog. (Alley)	(1)
May	86	ETI	93	Balloon Safari, The Drop and Flatten. (Sheppard)	(1)
Jul.	86	YC	75	Simon. (Proctor)	(1)
-	88	BYC	76	Drawing Program. (Winter)	(1)
-	88	BYC	77	Tea-pot Song. (Winter)	(1)
-	88	BYC	78	Ping Tennis. (Duncan)	(1)
-	88	BYC	79-82	Concentration. (Vella)	(4)
-	88	BYC	83	Super Snake Trapper. (Duncan)	(1)
-	88	BYC	84	Worm. (Thompson)	(1)
-	88	BYC	85	Dogfight. (Thompson)	(1)
-	88	BYC	86-87	Bezerk. (Banks & Saunders)	(2)
-	88	BYC	87	Argggh! (Banks & Saunders)	(1)
-	88	BYC	87	Encode/Decode. (Banks & Saunders)	(1)
-	88	BYC	88	Catch. (Banks & Saunders)	(1)
Apr.	88	ETI	65	U-foe. (Alderton)	(1)
Jul.	88	ETI	73	Disintegrator. (Stibbard)	(1)
Aug.	88	ETI	65	Star Fighter. (Roberts)	(1)
Nov.	88	ETI	121	Drawing Board. (Maunder)	(1)
May	89	ETI	87-88	Camel (Maunder)	(2)

BUSINESS

Aug.	84	APC	172-7	Database VZ-200. (Barker)	(6)
Oct.	84	APC	214	WP for VZ-200. (McQuillan)	(-)
Oct.	85	APC	82-3	Comment on Barker's and Quinn's DB. (Lukes)	(-)
Oct.	84	APC	126-30	Minicalc Spreadsheet. (Stamboulidas)	(5)
Dec.	84	APC	214	Correction to Minicalc.	(1)
May	85	APC	162-3	Micro Type(WP). (Browell)	(2)
Jul.	85	APC	164-6	Database. (Quinn)	(2)
Feb.	88	ETI	72	VZ Wordprocessor. (Tunny)	(1)

PERIPHERALS

Feb.	84	EA	131-2	Real-world interface.	(1)
Aug.	84	EA	65	Improved graphics on VZ-200. (Dimond)	(1)
Aug.	84	PCG	83	I/O card for VZ-200. (ad)	(1)
Oct.	84	APC	214	Serial help request. (Pope)	(1)
Dec.	84	APC	36	Add-ons for VZ-200. (Bleckendorf)	(-)
Oct.	85	YC	140	VZ200/300 Modem. (ad)	(-)
Nov.	84	BI	3,4	RTTY with VZ200. (Keatinge)	(2)
Nov.	84	ETI	106-12	A 'Glass-Teletype' using the VZ-200 Pt I	(7)
Dec.	84	ETI	93-7	" " " " Pt II	(5)
Aug.	85	ETI	72-8	VZ-200 terminal.	(7)
Jun.	86	EA	106	VZ serial terminal. (ad DSE kit K6317)	(-)
				Assembler listing of RS-232 ROM software	(13)
Sep.	85	AR	10-11	Another RTTY. (Butler)	(2)
Jan.	86	AR	19-20	Morse on RTTY. (Butler)	(2)
Feb.	86	ETI	72-4	Modifying VZ-200 16K memory expansion. (Olney)	(3)
Mar.	86	ETI	48	Talking VZ-200. (Bennets)	(1)
Jul.	86	ETI	55-60	Super II VZ-200 hardware modifications. (Sorrell)	(6)
Oct.	86	ETI	14	Errata for Super II.	(-)
Jan.	87	EA	60	EPROM programmer modification. (Buhre)	(1)
Feb.	87	AR	16-17	Morse Interface. (Forster)	(2)
May	87	EA	51	16K Memory Expansion VZ300. (Kosovich)	(3)
Jan.	88	EA	174	VZ-300 expansion problem.	(-)
Aug.	88	EA	138	VZ-300 expansion.	(-)
May	89	EA	124-125	RAM Expansion - Discussion (Sorrell)	(-)
Oct.	88	EA	140	Circuit idea.	(-)
Jun.	87	EA	129	Errata Memory Expansion.	(-)
Jun.	87	AEM	8	VZ software. (Thompson)	(1)
Apr.	88	AR	11-15	Memory expansion for V2200/300	(5)
Apr.	88	AEM	57-63	Ultra-graphics adaptor. (Sorrell)	(8)
Jun.	88	AEM	7	Correction.	(-)
Jul.	88	AEM	7	Correction.	(-)
May	88	ETI	70	VZ amp. (Merrifield)	(1)
Apr.	89	ETI	96	Better VZ amp. (Hobson)	(-)
May	88	ETI	82-86	VZ300 EPROM programmer. (Nacinovich)	(5)
Jun.	88	ETI	86-89	" " " "	(4)
				BASIC listing of software	(5)
Jul.	88	ETI	88-92	VZ300 data logger. (Sutton)	(5)

COMMERCIAL SOFTWARE REVIEWS

Mar.	84	APC	190-1	Review of DSE 'Matchbox', 'Biorhythms', 'Circus' and 'Poker'. (Davies)	(2)
Aug.	84	PCG	46-47	Review of DSE 'Panik' and 'Ladder Challenge'.	(1)
Oct.	84	PCG	90-91	Review of DSE 'Knights and Dragons', 'Ghost Hunter', 'Othello', and 'Invaders'.	(2)
Nov.	84	PCG	90-96	Review of LYSCO 'Cub Scout' and DSE 'Dracula's Castle'.	(1)
Jan.	85	PCG	65	Review of DSE 'Air Traffic Controller' and 'Tennis'.	(1)
Feb.	85	PCG	76	Review of DSE 'Defence Penetrator' and 'Star Blaster'.	(1)
Mar.	85	PCG	76-77	Review of DSE 'Planet Patrol' and 'Learjet'.	(1)
Apr.	85	PCG	94-99	Review of DSE 'Asteroids', Super Snake' and 'Lunar Lander'.	(1)
Apr.	85	ETI	103	Logbook and Morse on VZ-200.	(1)
Oct.	85	PCG	68-9	Review of DSE 'Duel'.	(1)
Nov.	85	PCG	70-1	Review of DSE 'Attack of the Killer Tomatoes'.	(1)
Nov.	85	CLC	31	Review of educational software.	(1)

SOFTWARE ADVERTISEMENTS

A 15 page compilation of ads. for a variety of software, services, User groups etc. (12)

HARDWARE REVIEWS

Apr.	83	YCU	56-59	Texet TX-8000. (Bennett)	(3)
Apr.	83	APC	58-66	VZ-200. (Hartnell)	(5)
Apr.	83	CC	38-43	Review of VZ-200.	(3)
May	83	CC	26-30	Video Technology VZ-200 PC. (Ahl)	(3)
Jun.	83	EA	137	New low-cost computer - VZ-200.	(1)
Jun.	83	ETI	30	Dick Smith colour computer.	(1)
Jun.	83	YC	6	DSE VZ-200.	(-)
Aug.	84	PCG	12	VZ-200.	(-)
Jul.	83	ETI	32-7	DSE's personal colour computer. (Harrison)	(3)
Jul.	83	EA	130-3	The VZ-200: colour, graphics and sound. (Vernon)	(4)
Jul.	83	PCN	16	Timing the Laser's phazer. (Stokes)	(1)
Sep.	83	WM	40	Laser.	(-)
Sep.	83	BB	18-20	Dick Smith VZ200: good value. (Fullerton)	(3)
Aug.	83	YC	20-33	Cash and Carry Computers. (Bell)	(9)
Sep.	83	CC	202-4	Review of VZ-200 and PP40.	(1)
Oct.	83	APC	77-8	VZ-200.	(1)
Oct.	83	WM	135	Texet TX8000.	(1)
Oct.	83	CT	12	The Laser 200.	(-)
Dec.	83	CT	11	Laser 200.	(-)
Nov.	83	CT	37-40	A look at the Laser. (Green)	(4)
Nov.	83	WM	42-108	The Laser - a shot in the dark.	(3)
Nov/Dec	83	SYN	17-22	VZ-200. (Ahl)	(2)
Feb.	84	CC	218-21	Laser PP40 Printer/Plotter.	(2)
Spring	84	MC	52-4	Laser 200. (Green)	(3)
Jun.	84	EA	12-9	Buying your first computer. (Vernon)	(6)
Aug.	84	EA	30-3	An important role for small computers. (Williams)	(4)
Oct.	84	PCG	82-87	Home micro supertest. Pt. 3 (Bollington)	(5)
Nov.	84	PCG	14-19	Home micro supertest. Pt. 4 (Bollington)	(4)
Nov.	84	EA	78-80	VZ-200 as a WP (DSE E&F tape WP). (Williams)	(2)
Dec.	84	CHC	28-31	Review of video games consoles.	(4)
Mar.	85	EA	31-33	Back to the VZ-200. (Williams)	(1)
Jul.	85	ETI	102-6	Dick Smith's new VZ-300. (Rowe)	(5)
Aug.	85	EA	22-7	WP on the new VZ-300. (Williams)	(5)
Dec/Jan	86	PCG	11-15	How to buy a micro - VZ-300 compared.	(4)
Aug.	86	AHC	38-39	Computers for the Rest of Us. (Roberts)	(2)
Nov.	86	AHC	44	Letter. (Kennedy)	(-)
Dec.	87	YC	20-21	VZ-300. (Hartnell)	(2)
Dec.	87	YC	78	VZ-300	(1)

GENERAL PROGRAMMING

Apr.	81	ETI	87-93	Extra Z80 opcodes.		(4)
Jun.	81	ETI	97	More uncovering Z80. (Dennis)		(1)
Jul.	81	ETI	83	Z80 uncovered. (Garland)		(-)
				Z80 CPU reference card		(2)
Feb.	82	YC	64-66	Understanding Assembler (Bell)	Part I	(3)
Mar.	82	YC	74-77	(8080)	Part II	(4)
Apr.	82	YC	61-63	" " "	Part III	(3)
May	82	YC	60-62	" " "	Part IV	(3)
Jun.	82	YC	99-101	" " "	Part V	(3)
Jul.	82	YC	1-74	" " "	Part VI	(3)
Sep.	82	YC	57-59	" " "	Part VII	(3)
Nov.	82	YC	45-46	" " "	Part VIII	(2)
Dec.	82	YC	93-97	" " "	Part IX	(4)
Jan/Feb	83	YC	52-55	" " "	Part X	(4)
Mar.	83	YC	61-62	" " "	Part XI	(2)
Aug.	83	YC	62-68	" " "	Part XII	(6)
Oct.	83	YC	87-89	" " "	Part XIII	(2)
Nov.	83	YC	102-104	" " "	Part XIV	(3)
Feb.	84	YC	93-94	" " "	Part XV	(2)
Apr.	84	YC	123-126	" " "	Part XVI	(2)
Nov.	82	PE	1/1-1/5	PE Micro-file #1 - 8080 & 8085 (Coles)		(5)
Jan.	83	PE	3/1-3/5	PE Micro-file #3 - Z80. (Coles)		(5)
Mar.	84	APC	73-85	Teach yourself assembler	Pt. 1 (Overaa)	(6)
Apr.	84	APC	57-64	(8080, Z80, 6502)	Pt. 2 (Overaa)	(5)
May	84	APC	89-98	" "	Pt. 3 (Overaa)	(5)
Jun.	84	APC	53-60	" "	Pt. 4 (Overaa)	(5)
Jul.	84	APC	61-64	" "	Pt. 5 (Overaa)	(3)
Aug.	84	APC	110-116	" "	Pt. 6 (Overaa)	(5)
Sep.	84	APC	145-151	" "	Pt. 7 (Overaa)	(4)
Jan.	85	APC	122-124	Sort at input. (Ithell)		(1)
Feb.	85	APC	103-109	The basic art - algorithms, structures. (Liardet)		(4)
Mar.	85	APC	98-109	Pick a number - arithmetic. (Liardet)		(5)
Apr.	85	APC	79-87	It takes all sorts - sorting. (Liardet)		(5)
Oct.	85	APC	82	The Art of Programming - Progress. (Hjaltson)		(-)
Jun.	85	APC	170-171	Comment on binary search. (Lamich)		(1)
Jun.	85	APC	171-173	Comment on distribution sort. (Riordon)		(1)
Oct.	85	YC	107-8	Sorting out the sorts. (Jankowski)		(1)
Mar.	86	PE	17-18	Z80		(2)

AEM	Australian Electronics Monthly
AHC	Australian Home Computers
APC	Australian Personal Computer
APH	Australian Photography
AR	Amateur Radio
ARA	Amateur Radio Action
BB	Bits and Bytes (NZ)
BI	Break In (NZ)
BYC	Bumper Book of Programs by YC
CBA	CB Action
CC	Creative Computing (US)
CFG	Computer Fun and Games
CI	Computer Input (NZ)
CLC	Classroom Computing
CT	Computing Today (UK)
CHC	Choice
EA	Electronics Australia
ETI	Electronics Today International
M80	Micro-80
MC	Micro Choice (UK)
PCG	Personal Computer Games
PCN	Personal Computer News (UK)
PE	Practical Electronics (UK)
SYN	Sync (US)
WM	Which Micro (UK)
YC	Your Computer
YCU	Your Computer (UK)

FURTHER LITERATURE RELATING TO THE VZ200/300 COMPUTER

As an extension to my list of magazine articles, I have produced the following list of books (I have copies of all of the publications). The books relate to the VZ computer specifically, Microsoft BASIC Level II or the Z-80 microprocessors, as used in the VZ200/300. Additionally, I hold a lot of additional technical information, ROM listings, Users Group newsletters, software etc.

TECHNICAL BULLETINS FOR VZ COMPUTERS

# 88	Printing out System-80 screen graphics.	(2)
# 91	Programming the VZ-200 computer's joysticks.	(3)
# 92	Finding where variables are stored by the VZ-200's BASIC.	(3)
# 93	Problems with the X-7208 printer/plotter and Microsoft BASIC.	(1)
# 94	Using the X-3245 TP-40 printer/plotter with the VZ-200 & System-80.	(1)
# 98	Printing lower case and control characters on the VZ200/300.	(1)
#111	VZ-300 Mailing List tape to disk file conversions.	(1)
#114	Obtaining colour on the VZ300.	(1)
#116	Fixing the printer bug in the VZ Editor-Assembler.	(1)
	Letter on tapes and keyboard	(1)
	General hints on VZ	(1)
	Service Manual for printer interface	(7)
	Service Manual for disk drive controller	(12)

BOOKS ON VZ COMPUTERS

Henson, T.L.,	1983	"Introduction to Computing". DSE, 114 p.	(60)
Hartnell, T., & Predebon, N.,	1983	"Getting Started". DSE, 121 p.	(68)
Hartnell, T.,	1983	"Further Programming". DSE, 135 p.	(74)
Hartnell, T., & Pringle, G.,	1983	"The Giant Book of Games". DSE, 179 p.	(94)
-	1983	"First Book of Programs". DSE, 58 p.	(60)
-	1983	"Second Book of Programs". DSE, 57 p.	(60)
Rowe, J.,	1983	"VZ-200 Technical Reference Manual". DSE, 22 p.	(30)
-	1985	"VZ-300 Technical Manual". DSE, 39 p. (Available from DSE $14.95)	(65)
Hartnell, T.,	1986	"Programming the VZ300". DSE, 171 p. (Available from DSE $14.95)	
Hartnell, T.,	1986	"The Giant Book of Games for the VZ300". DSE, 278 p. (Available from DSE $19.95)	
Hartnell, T.,	1986	"The Amazing VZ300 Omnibus". DSE, 188 p. (Available from DSE $19.95)	
Wolf, G.,	1985	"ROM-listings fur Laser 110, 210, 310 und VZ200". Vogel-Buchverlag. 278 p.	
Wolf, G.,	1985	"Der BASIC-Interpreter in Laser 110, 210, 310 und VZ200". Vogel-Buchverlag. 152 p.	
Wolf, G.,	1985	"Das Laser-DOS fur Laser 110, 210, 310 und VZ200". Vogel-Buchverlag. 131 p.	
Sanyo,	1984	"Mein Laser Home-Computer, Tips and Tricks fur Einsteiger". Sanyo Video Vertrieb. 91 p.	
Sanyo,	1984	"Laser Home-Computer, Software-System Handbuch I". Sanyo Video Vertrieb. 114 p.	
D'Alton, J.,	1986	"Vprogrammez Hints and Hardware No. 1" 48 p.	
Schaper, P.,	1987	"Beginners Guide to the VZ 200/300 Editor Assembler" 57 p.	
Olney, S.	1987	"VZ 200/300 Assembly Language Programming Manual for Beginners". 140 p.	

BOOKS ON BASIC

Albrecht, R.L., Finkel, L., & Brown, J.R.,	1978	"BASIC". John Wiley, 2nd Edition. 325 p.
Albrecht, B., Inman, D., & Zamora, R.,	1980	"TRS-80 BASIC". John Wiley. 351 p.
Inman, D., Zamora, R., & Albrecht, B.,	1981 1981	"More TRS-80 BASIC". John Wiley. 280 p.
Lien, D.A.,	1982	"Learning TRS-80 BASIC". Compusoft. 528 p.
Gratzer, G.A. & Gratzer, T.G.,	1982	"Fast Basic - beyond TRS-80 BASIC". John Wiley. 278 p.
Rosenfelder, L.,	1981	"BASIC Faster and Better and other mysteries". IJG, California. 288 p.
Bardon, W.,	1985	"TRS-80 Computer Reference Handbook" Radio Shack 2nd edit.

BOOKS ON ASSEMBLER AND Z80

Carr, J.J.,	1980	"Z80 Users Manual". Reston Publishing Co., 326 p.
Weller, W.J.,	1978	"Practical Microcomputer Programming: the Z80". Northern Technology, 481 p.
Fernandez, J.N., & Ashley, R.	1981	"Introduction to 8080/8085 Assembly Language Programming". John Wiley, 303 p.
Miller, A.R.,	1981	"8080/Z80 Assembly Language - techniques for improved programming". John Wiley, 318 p.
Leventhal, L.A.,	1979	"Z80 Assembly Language Programming". Osborne/McGraw-Hill.
Leventhal, L.A., & Saville, W.	1983	"Z80 Assembly Language Subroutines". Osborne/McGraw-Hill, 497 p.
Nitschke, W.,	1985	"Advanced Z80 - Machine Code Programming". Interface Publications, 342 p.

Nichols, J.C., Nichols, E.A., & Rony, P.R.	1979	"Z-80 microprocessor programming and interfacing - Book 1". Howard W. Sams, 302 p.
Nichols, J.C., Nichols, E.A., & Rony, P.R.	1979	"Z-80 microprocessor programming and interfacing - Book 2". Howard W. Sams, 494 p.
Nichols, J.C., Nichols, E.A., & Musson, K.R.	1983	"Z-80 microprocessor advanced interfacing with applications in data communications". Howard W. Sams, 347 p.
Barden, W.,	1979	"TRS-80 Assembly-Language Programming". Radio Shack, 224 p.
Barden, W.,	1982	"More TRS-80 Assembly-Language Programming". Radio Shack, 430 p.
Farvour, J.L.		"Microsoft BASIC Decoded and other mysteries". IJG, California, 310 p.
Sargent, M., & Shoemaker, R.L.	1981	"Interfacing Z80 microcomputers to the real world". Addison Wesley, 288 p.
Ullman, J.,	1984	"Pocket Guide Assembly Language for the Z80". Pitman, 58 p.
Overea, P.A.,	1984	"Teach Yourself Assembler Z80". Century Communications, London, 236 p.
Barrow, D.,	1985	"Assembler Routines for the Z-80". Century Communications, London, 192 p.
Uffenbeck, J.,	1985	"Microcomputers and Microprocessors: the 8080, 8085 and Z80. Programming, Interfacing and Troubleshooting". Prentice Hall, 670 p.
Barden, W.,	1978	"The Z80 Microcomputer Handbook" Howard Sams, 304 p.
Goodwin, M.	1983	"Level II ROMS" Tab Books, 536 p.
Blattner, J., & Mumford, B.,	1980	"Inside Level II" Mumford Micro Systems, 65 p.
Barden, W.,	1982	"TRS-80 Assembly Language Subroutines" Prentice Hall, 232 p.
Toothill, A., & Barrow, D.,	1983	"Z80 Code for Humans" Granada, 152 p.

UTILITIES

Oct.	83	APC	52,4	BASIC program conversion. (Surya)	(2)
Jan.	84	APC	20-21	Beginners tips. (White)	(-)
Nov.	83	APC	57,9	Program conversion Pt. 2 (Surya)	(2)
Nov.	83	APC	89-95	BASIC converter chart. (Surya)	(7)
Feb.	84	APC	140-1	Program conversion Pt. 2 (Surya)	(2)
Mar.	84	APC	42-3	Program conversion - Apple II (Surya)	(2)
Apr.	84	APC	71-2	Program conversion - TRS 80/System 80 (Surya)	(1)
May	84	APC	75-6	Program conversion - Atari (Surya)	(2)
Jun.	84	APC	67	Program conversion - Sinclair (Surya)	(1)
Jul.	84	APC	129-30	Program conversion - BBC (Surya)	(2)
Mar.	84	ETI	63	More functions for the VZ-200. (Olney)	(1)
Apr.	85	ETI	117	Notes and errata for Olney.	(-)
Jul.	84	BB	56	Some more routines. (Middlemiss)	(1)
Jul.	84	M80	3-4	VZED - three new functions.	(1)
Aug.	84	M80	2	VZ-200 output latch.	(1)
Aug.	84	M80	9,15,16	Memory peek VZED. (Carson)	(1)
Aug.	84	M80	3-4	Microsoft ROM BASIC Level I bug.	(1)
Apr.	85	APC	97	VZ-200 bug. (Tritscher)	(-)
Aug.	85	APC	31	VZ bug. (Tritscher)	(-)
Aug.	84	APC	94	VZ-200 moving message and trace. (Batterson)	(1)
Nov.	84	APC	76	Trace function. (Breffit)	(-)
Nov.	84	APC	125	VZ-200 correction. (Kelly)	(-)
Sep.	84	CI	19	VZ200 Input. (Woolf)	(1)
Sep.	84	BB	63	Poking extra functions. (Clark & Hill)	(1)
Oct.	84	ETI	135-7	Extending VZ-200 BASIC. (Olney)	(3)
Nov.	84	APC	125-6	TRON/TROFF function for VZ-200. (Thompson)	(1)
Nov.	84	APC	208-12	MON-200 machine code monitor. (Stamboulidas)	(5)
Nov.	84	PCG	55-56	Lprinter. (Quinn)	(2)
Nov.	84	PCG	suppl.	VZ-200 reverse video.	(1)
Dec.	84	BB	64	Enlarged characters. (Velde)	(1)
Feb.	85	APC	171	BASIC understanding. (Hobson)	(1)
Feb.	85	APC	20	VZ-200 into puberty - Olney's extended BASIC.	(1)
Feb.	85	ARA	19-26	Calculating grey line. (Baker)	(6)
Mar.	85	CI	12-14	Renumber. (Marsden)	(3)
Apr.	85	PCG	62-64	Find. (Stamboulidas)	(3)
Apr.	85	APC	19	Use of RND in dice and card games. (Holland)	(1)
Apr.	85	APC	103	VZ variable definition. (Stamboulidas)	(1)
Apr.	85	APC	95	Variable GO TO on VZ. (Olsen)	(1)
Jul.	85	APC	176	Correction to VZ variable GO TO.	(-)
May	85	APC	52-3	Lysco support for VZ-200. (Young)	(1)
May	85	ETI	99-101	VZ-200 hardware interrupt. (Olney)	(3)
May	85	APC	110	Background VZ. (Williams)	(1)
Aug.	85	APC	130	VZ-200 instant colour. (Willows)	(-)
Aug.	85	APC	130-3	Reversed REM. (Quinn)	(1)
Sep.	85	APC	145	Real-time clock. (Griffin)	(1)
Oct.	85	APC	218	APC benchmark BASIC programs.	(1)
Oct.	85	APC	147	VZ deletions. (Quinn)	(1)
Nov.	85	APC	189	VZ EDITOR/ASSEMBLER tips. (Lam)	(1)
Nov.	85	ETI	94-5	Olney's Level II BASIC for VZ200/300. (Rowe)	(2)

Jan.	86	APC	83,5	VZ user graphics.	(1)
Feb.	86	APC	127	Machine language calls.	(1)
Mar.	86	APC	chart	APC BASIC converter chart 1986.	(8)
Mar.	86	YC	103-5	VZ-200 cassette inlays. (Dutfield)	(3)
May	86	APH	54-55	VZ and photography. (Kohen)	(2)
Jun.	86	APC	209	VZ pause.	(1)
Aug.	86	ETI	86-89	VZ software mods. (CHIP-8 Editor) (Griffin)	(3)
Oct.	86	ETI	28-33	VZ CHIP-8 Interpreter. (Griffin)	(5)
Sep.	86	AEM	89-92	Screen handling on VZ. Part I. (Kitch)	(4)
Oct.	86	AEM	110-112	Screen handling on VZ. Part II. (Kitch)	(4)
Oct.	86	AEM	113,4,21	Reference list of VZ articles. (Kitch)	(2)
Oct.	86	ETI	47	Labeller. (Gallagher)	(1)
Oct.	86	ARA	38-42	Amateur radio logger. (Johnson)	(5)
Nov.	86	EA	35	Speaker enclosure calculator. (Allison)	(1)
Dec.	86	AEM	90-95	Memory mapping on VZ. (Kitch)	(6)
Mar.	87	AR	10-12	Feedline calculations. (Buhre)	(3)
Apr.	87	EA	100-101	Op amp noise. (Allison)	(2)
Apr.	87	ARA	20-24	Beam Headings. (Baker)	(5)
May	87	AEM	86-88	VZ Epson printer patch. (Taylor)	(3)
Jun.	87	AEM	74,75,79	VZ Epson printer patch Pt II.	(3)
Aug.	87	AEM	82-83	VZ expanded EPROM. (Meager)	(2)
-	88	BYC	88	Restore file. (Banks & Saunders)	(1)
-	88	-	-	B-file copier. (Buhre)	(1)
Feb.	88	ETI	70	String file name. (Hand)	(1)
Jul.	88	ETI	74	Disk directory dumper. (Tunny)	(1)
Oct.	88	ETI	124	CTRL-Break disabler. (Tunny)	(1)
Oct.	88	AEM	96-97	VZBUG. (Batger)	(2)
Nov.	88	ETI	120	Clock. (Tunny)	(2)
Feb.	89	ETI	118-119	DOS Hello (Tunny)	(1)
Feb.	89	ETI	119-120	Visisort (Sheppard)	(2)
Nov.	89	ETI	73	Restore (Rowe)	(1)
Nov.	89	ETI	73	Hex/dec conversion (Maunder)	(1)
Jan.	90	CBA	17-19	Beam headings (Baker)	(3)

A BEGINNER'S GUIDE TO PROGRAM CONVERSION

This month Surya provides some direction for those trying to get to grips with program conversion. Next month, hours upon hours of blood, sweat and tears will come to fruition in the presentation of APC's Basic Program Converter Chart. It's a compilation of the Basic keywords of popular micros set out to enable equivalent words in your micro's dialect of Basic to be used in their place.

When you've just picked up your copy of *APC* and spotted a nice little cassette-based database for the TRS-80 it's very tempting to sit down in front of your VIC 20 and start tapping away, altering lines as you go and hoping that it will run when you've finished it. Unfortunately, while you can sometimes get away with this on very short programs, anything longer than twenty or thirty lines and you quickly find yourself in a mess. The first rule of program conversion is stop and think! This brief article is not a definitive guide to program conversion, bit it should give a few pointers to those relatively new to the game.

So where do you start? Well, first of all think about whether a conversion is really the best approach to the problem. Although modifying an existing listing may sound easier than writing the program from scratch, this is not always the case. In choosing between a conversion and a complete rewrite, there are a number of factors to be considered:

(a) The compatibility of the machines.

Some machines support very similar dialects of Basic: the TRS-80 and the System 80 for example. In a number of cases, the program may require only a few minor changes here and there to enable it to run on a similar machine. You may even find that no changes at all are needed.

Other machines, however, are almost entirely incompatible. Converting from a Commodore machine, for example, with its cursor-control statements embedded in the text, can be a real pain. Equally, converting from a powerful machine to a lesser beast may cause problems: a Basic with recursively-defined procedures (procedures within procedures) and REPEAT-UNTIL loops can be very difficult to rewrite efficiently for a machine which doesn't support a structured Basic.

Although converting from a simple machine to a more sophisticated one is generally easier than the other way around, you will be sacrificing the features for which you bought the machine. Any ZX81 listing will run on a Spectrum, but then what's the point of having a Spectrum?

(b) Sound and graphics.

However compatible machines may be in other respects, they usually bear not the slightest resemblance where sound control and graphics resolution are concerned. Where a program relies heavily on these features, therefore, rewriting the program from scratch would probably be easier than attempting to modify it.

(c) Machine-code, assembler, PEEKs and POKEs.

Any program relying heavily on machine-code or assembler, or where a significant amount of PEEKing and POKEing is done, will be extremely difficult — if not impossible — to modify for a different machine. Anyone who knows enough about low-level programming to do the job would almost certainly be able to write their own routines in a fraction of the time taken to convert someone else's.

(d) The structure of the program.

I must confess a sneaking sympathy for the view that 'all that matters is that it works'. When I'm writing ordinary day-to-day programs for use around the office or whatever, my programs are neither elegant not structured. Having publicly owned up to this fatal flaw in my otherwise perfect character, I am now going to sing the praises of structured or modular programming.

Structured programming is the art of assigning each component function of the program a routine of its own. Take the example of a simple database, there would be one routine to display the menu, another to accept input, another to sort data, yet another to output data to a printer, and so forth. Each routine, or module is entirely independent of any other, being called by a central 'control' module. You could, for example, remove the printout routine simply by deleting a solid chunk of code and deleting the option from the menu. The rest of the program would be totally unaffected.

A well-structured program is not only easy to read and edit, is also lends itself to modification for a different machine. If (say) the bar-chart section cannot be used on your machine because of the difference in screen-addressing, you can simply replace it with your own routine without necessitating all kinds of changes in other sections of the program.

If a program is very badly structured, it is often easier to write your program rather than wading through GOTOs, attempting to follow a logical path which jumps in and out of loops and so on, and altering one part of the program may have unforeseen effects in a completely different part.

(e) The program as a whole.

Does it do exactly what you'd like it to, or merely approximately what you want? There's little point in modifying an exciting program if you're then going to have to spend a lot more time on it in order to get it to do something else.

Do you understand the way the program works? If you don't, then not only are your chances of carrying out a successful modification pretty slim, but the program may not do what you thought it would even if you succeed!

By this stage, then, you should have decided whether you're going to modify the program as it stands, or write a completely new program of your own to do the job. If you decide on the latter, it doesn't necessarily put you right back at square one. The general structure of the program may provide a good starting-point, and you may also be able to incorporate some of the routines into your own program. Treat the original

Oct 83 4 (10) p 52 and 54 1 of 2.

A BEGINNER'S GUIDE TO PROGRAM CONVERSION

program as a source of ideas and techniques, but don't be limited by it.

Let's say you've decided on a conversion. I'll identify the sections likely to cause problems. PEEKs and POKEs are an obvious place to start. The author should have added REMark statements telling you what they do, and you need only figure out how to achieve the same effect on your own machine. If not, then you're into the business of getting hold of the host machine (that is, the machine the program was written for) and trying out anything you're not sure of.

Next to look for is the screen displays: mainly graphics and PRINT AT statements. These will probably have to be completely rewritten. Work out what is happening — what is being plotted and where messages appear on the screen. This can sometimes be tricky, particularly where those quaint Commodore control-codes are concerned (you may have gathered that I don't jump up and down about Commodore screen-handling). Bear in mind that you don't have to duplicate the original screen exactly — or even approximately — for menus and so on. Generally, the only time when you need to recreate the screen faithfully is during games where the graphics are vital. The difficulty of adapting such programs has already been mentioned.

By now, you will probably have come across several sections of code that appear totally alien to the version of Basic supported by your machine. In these cases you must work out exactly what is happening, when, where, why and how. Once you've done that (he says lightly), it should be a straightforward matter to replace the offending code with your own routine. This is when you find out just how structured the program really is. I once followed a series of about nine GOTOs, the final one ending on the line following the first one with nothing having happened in between. OK it's an extreme example, but there are some funny people about . . .

Anyway, next on the agenda is to go through the listing making note of anything which looks slightly, rather than totally, out of place in your machine's Basic. You'll find that most of the changes will be fairly obvious even if you've never seen some of the keywords before. Most people would guess that HOME is the same as CLS, for example. Next month, *APC* will publish its Basic Converter Chart (which has been no mean feat to produce) which should help you sort out the stranger idiosyncracies of some machines.

If you're converting to a less powerful Basic then you may have to work at simulating some of the more sophisticated features. FOR-NEXT loops come in very handy to simulate functions such as INSTR$, STRING$ and so on.

And this is the point where you start hammering away at the keyboard! Provided you've done all the above thoroughly, a combination of the *APC* Basic Converter Chart and good old-fashioned trial-and-error should see you through!

APC Oct 83 4(10) p. 52 and 54 2 of 2.

Beginner's tips

On reading the October issue of *APC*, I noticed that Surya made a very common error in his 'Beginner's guide to program conversion'. He states that '(repeat-until and while-endwhile) . . . are two forms of the same loop, one being the logical reverse of the other.'

There is one essential difference between while <cond> and repeat <block> <block> endwhile until not (<cond>)

The 'while' form checks the condition first. If it's false, then <block> is not executed even once. By contrast, the 'repeat' form causes at least one execution of <block>, even if the condition is initially false.

Wherever a 'repeat-until' is used, it may, if desired, be replaced by a 'while-endwhile' with inverted condition (although there are several cases where a 'repeat-until' is more natural — which is precisely why any decent structured language provides both constructs).

As practical examples of differences, consider the following two examples: first, a routine to throw a die until a six is thrown:

```
repeat
  DIE:=rnd(1 to 6)
  print 'You throw a', DIE
until DIE=6
```

This can be written as a somewhat convoluted 'while':

```
DIE:=0 (indeed, any number
that isn't six)
while DIE< >6
  DIE:=rnd (1 to 6)
  print 'You throw a' DIE
endwhile
```

(although no-one but an idiot would use this if they had repeat-until available).

Second, consider a routine to print a sequential file:

```
open FILE$
while not (eof)
  readline (A$)
  print A$
endwhile
close FILE$
```

(eof is a boolean (true or false) function indicating whether or not the End Of File marker has been encountered. Any attempt to read a line of text when eof is true will probably crash the routine). Using the Surya-style conversion, we obtain:

```
open FILE$
repeat
  readline (A$)
  print A$
until eof
close FILE$
```

Whereas the first form correctly detects, when the file is empty, that eof is true initially — and so immediately closes the file, the second form attempts to read a line of text from the empty file — thus crashing the program.

Therefore, to summarise, any repeat-until may be replaced by a while-endwhile — but with some loss of clarity, but the converse is not true — attempting to convert from a while-endwhile to a repeat-until does not usually work.

Duncan White

Yes, you are quite correct. When converting from a while-wend to a repeat-until loop it is sometimes necessary to insert manually a test which somewhat defeats the point of the loop! It is, however, usually possible to make the initial test before entering the loop, thus retaining some degree of structure. Thus:

```
OPEN FILE$:IF NOT OF
THEN PROC readfile ELSE
CLOSE FILE$ . . .
DEFPROC readfile
  REPEAT
  READLINE (A$)
  PRINT A$
  UNTIL EOF
  CLOSE FILE$
```

I would, however, agree wholeheartedly that a truly structured language should offer both constructs.

Surya

APC Jan 84 5(1) p. 20 & 21.

A BEGINNER'S GUIDE TO PROGRAM CONVERSION
PART 2: SIMULATING STATEMENTS

Last month Surya looked at the factors to consider when choosing between a program conversion and a complete rewrite. Here he assumes that a conversion is appropriate and analyses the procedure in detail.

The initial steps to be taken when converting a program from one dialect of Basic to another are much the same as when coding from scratch and just as much discipline is required. The starting point in either case is to have a clear understanding of what you're setting out to achieve. Make sure you can follow the logic of the program before you attempt to modify it. Spend a little time working out why the author has done things in that particular way. All this may seem unnecessary at first, but it's time well spent: the greater your understanding of the program, the easier the conversion will be.

Once you're satisfied that you have a clear overview of the program as a whole, you can look at each section in detail. Break the program down into its component subroutines. This is only possible with a reasonably structured program, but as mentioned last month, programs with poor or non-existent structuring are best left alone.

When examining each routine, take a special look at the variables. Determine which are global and which are local. Global variables are those used throughout the program. Typical global variables include scores in games, some counters, printer-settings and so on. Local variables are those whose values are used only within a given subroutine: once the routine has been exited, the values are no longer required and the variables may be used for a different purpose within another routine. Typical local variables are counters in FOR-NEXT loops and flags used to check validity of data.

The reason you need to distinguish between the two is that local variables may be freely changed or discarded as appropriate, but global variables need to be treated with a great deal of care — the program as a whole is dependent upon them. If you're lucky, the programmer will have gone to the trouble of listing all global variables in remarks at the beginning of the program, and used fixed local variables so that, for example, w is always a FOR-NEXT loop counter. Failing that, there are utility programs available that will locate variables for you.

Coding

(Note: in the examples given below, I am using A$ to represent any string variable and 100 onwards whenever line numbers are required. These choices are purely arbitrary and have no significance.)

During the process of converting a program from one machine to another, you will very often come across a keyword in the original program for which your machine has no equivalent. While experienced programmers will soon find a way round the problem, those a little newer to the game may find themselves stuck for a solution. What I have done below is to look at some of the common offending statements and methods of achieving the same effect using standard Microsoft. The keywords covered are not in any particular order.

INKEY$: This statement is an almost statutory presence in just about every Basic program ever written. This statement tells the computer to scan the keyboard to test for a key depression and place the result into a specified variable. The standard format is A$=INKEY$; the most common variations are A$=GET$ GET$=A$ and GET A$.

The statement takes one of two forms. On most machines, the processor will carry out a single sweep of the keyboard: if a key is pressed during this scan, the value of the key pressed will be placed into the variable A$. If no key is pressed, A$ will be null (empty). On some machines, however, the computer will carry out a continual series of sweeps until a key-press is detected. A few machines offer both forms.

A continuous scan using the former version of INKEY$ is straightforward: 100 A$=INKEY$:IF A$="" THEN GOTO100. The BBC, however, goes a step further in offering a timed keyboard scan in the form A$=INKEY$(time), where time is given in 100ths of a second. To simulate this using the standard INKEY$ statement, we use a FOR-NEXT loop thus: 100 FOR A=0 TO (value):A$=INKEY$:NEXT. The value of the variable will need to be adjusted to suit. Since different machines have different processing speeds, you'll have to experiment with different values to establish some kind of relationship between the value of the FOR-NEXT counter and real time.

Of course, the example given above would return the final key pressed if there were two or more key depressions during the scan period, but this is easily overcome:

```
100 FLAG=0:A$=""
110 FOR A=0 TO (value)
120 B$=INKEY$:IF NOT B$="" AND
    FLAG=0 THEN A$=B$:FLAG=1
130 NEXT
```

The value of the first key depression is now stored in A$. If no key was pressed, then A$ will be empty.

INSTR: This statement is used to search one string to find out whether it contains a second string. The format is INSTR(main string, sub-string) where the starting position of the sub-string is returned on a successful match and 0 is returned if the search fails. INSTR("APC","P") would return 2 while INSTR("APC","X") would return 0.

We might, for example, want to find out whether NAME$ contains the sub-string 'Rev.'. Using INSTR, we would do this like so:

```
100 IF NOT(INSTR(NAME$,"Rev.")
    =0) THEN PRINT NAME$;" is a
    priest"
```

To simulate this in standard Microsoft, we use MID$. In the above example, we would do so thus:

```
100 FLAG=0:FOR A=1 TO
    (LEN(NAME$)-4
110 IF MID$(NAME$,A,4)="Rev."
    THEN FLAG=1
120 NEXT
130 IF FLAG=1 THEN PRINT
    NAME$"is a priest"
```

Note that on an Atari, line 110 would read as follows:

```
110 IF NAME$(A,4)="Rev." THEN
    FLAG=1
```

and on a Sinclair machine, it would read:

```
110 IF NAME$(A TO A+4)="Rev."
    THEN FLAG=1
```

These differences are due to the non-standard forms of MID$ supported by these machines. The original example should work on all other dialects of Basic.

PROCEDURES AND FUNCTIONS: User-definable functions are supported in varying degrees of sophistication by a number of machines. Procedures and functions make programs infinitely neater and more readable, but they don't actually achieve anything which cannot be duplicated using ordinary sub-routines.

Some dialects of Basic will allow you to GOTO or GOSUB a variable which greatly aids readability — the Basic Converter Chart will tell you which machines do if you look under GOTO.

Nov 83 4(11) p 57 and 59
1 of 2.

A BEGINNER'S GUIDE TO PROGRAM CONVERSION

REPEAT-UNTIL and WHILE-WEND. These are two forms of the same control loop, one being the logical reverse of the other. WHILE-WEND checks that a given expression is true and then executes all statements up to the first WEND statement encountered. The computer then returns to the original condition to check whether it is still true. If the condition is false, the statement following the WEND statement is executed.

For example:

```
100 REM — Silly example
110 X=10
120 WHILE X>0
130 PRINT "The current value of X
      =";X;"."
140 X=X-1:WEND
150 REM – X is now zero and the WHILE
      test fails
```

In a WHILE-WEND loop, the loop is repeated while the test expression is true. A REPEAT-UNTIL loop works the other way around. All statements between REPEAT and UNTIL are executed until the test expression is true. Thus the above example would be written:

```
100 REM — Same sillv example
110 X=10
120 REPEAT
130 PRINT "The current value of X
      =";X;"."
140 X=X-1:UNTIL X=0
150 REM – X is now zero and the
      REPEAT test is satisfied
```

Converting from one structure to the other is thus straightforward. But the majority of present-day Basics offer neither of the above. To create the same effect, we have to use a statement that causes purists to gasp in horror and head straight for the reassurance of their micro: the GOTO.

Thus:

```
100 REM — Here we go again
110 X=10
120 PRINT "The current value of X
      =";X;"."
130 IF X>0 THEN X=X-1:GOTO120
140 REM – X is now zero and the test fails
```

While somewhat less elegant, the net result is the same. We can see that rewriting a WHILE-WEND or REPEAT-UNTIL structure is simply a matter of manually inserting the test (using IF-THEN) and pointer (GOTO).

**STRING$** is a statement which allows you to repeat a given sequence of characters. The format is STRING$(number of times to print string,string). If you wanted to print a line of asterisks across an 80-column screen, for example, you would state: STRING$(80,"*"). If your machine doesn't support this statement, then we fall back once again on the ever ready FOR-NEXT loop. Thus: FOR A=1 TO 80:PRINT"*";:NEXT, the string is simply duplicated, and the numeric argument placed in the FOR-NEXT loop.

TAB. This is supported by most machines.

Next month: Graphics and sound

END

APC Nov 83 4(11) p. 57 and 59
2 of 2.

Article republished in APC Feb 84.

BASIC CONVERTER CHART

One day, all computers will understand the same language (and read each others' disks and address the screen in the same way and . . .). To tide you through until this great day arrives, however, we set out to beg, steal or even buy eleven of the most popular home micros to produce this APC Basic Converter Chart.

Whether you're trying to convert that amazing Atari game to run on your Apple, have just spent the past three hours wondering why your new Commodore 64 micro doesn't seem to give the right answer to a FRE statement or simply want to write programs which can be easily converted to other micros, the APC Basic Converter Chart is here to help.

It isn't possible, of course, to cover every micro nor every command supported by each of the machines included — much as we'd like to. Also, since different micros have an annoying tendency to use the same keyword to perform slightly — or totally — different functions, converting from one machine to another will require some rewiting beyond simply changing the syntax. What this chart aims to do, however, is provide you with an at-a-glance syntax comparison using Microsoft Basic as the standard. The chart won't convert programs for you, but it should save you the trouble of wading through masses of manuals written by authors who have apparently not yet heard about alphabetical indexing.

Due to the limited amount of information we can squeeze into each box, it hasn't always been possible to indicate the full power of every command or statement. Most LIST statements, for example, allow you to list the whole program, list a specified line, list all lines within a given range, list all lines up to a specified line or list from a specified line. Fiddling around with brackets in an attempt to represent each of these possibilities would lead to a totally incomprehensible entry. It should be assumed, therefore, that we're dealing with the most common use of each statement here and that other uses may be available.

Something to be aware of is that identical syntax may have very different effects on different machines. SYSTEM on a TRS-80 will transfer program control to a machine language routine while in Microsoft Basic closes files prior to returning to the operating system.

You will notice that we haven't included anything on sound and graphics; with most of today's micros offering both high-resolution graphics and fairly sophisticated sound control, this area would require a chart of its own. APC will be looking at sound and colour in a later issue.

The abbreviations used in the chart are as follows:

addr = address, exp = expression,
sub = subscript, stmt = statement,
var = variable,
Square bracket [ ] indicates optional code.

1 of 7.

Nov 83 4(11) 89 - 95

BASIC RESERVED WO

STANDARD MICROSOFT	ABS	ASC	ATN	AUTO	CALL	CHAIN	CHR$	CLEAR	CLOSE	CONT	COS
	Gives absolute value of expression.	Returns ASCII value of first character of string.	Arctangent of expression.		Calls assembler language subroutine.	Call a new program & pass variables to it.	Gives one-char-string with ASCII code of exp.	Clear-selected vaariables.	Closes disk files —closes all files if no specification.	Continue program execution.	Cosine of expression.
MACHINE	ABS(exp)	ASC(string)	ATN(exp)	AUTO [lineno, val]	CALL var[(var, ver . . .)]	CHAIN "filename"	CHRS(exp)	CLEAR[exp. exp]		CONT	COS(exp)
APPLESOFT	ABS(exp)	ASC(string)	ATN(exp)		CALL addr	CHAIN "filename"	CHRS(exp)	CLEAR	CLOSE "filename"	CONT	COS(exp)
ATARI	ABS(exp)	ASC(string)	ATN(exp)			RUN"C:" NB: program must have been saved using SAVE "C"	CHRS	CLR	CLOSE [#fileno. fileno . . .]	CONT	COS(exp)
BBC MICRO	ABS(exp)	ASC(string)	ATN(exp)	AUTO [lineno. val]	CALL addr [.var,var . . .]	CHAIN "filename"	CHRS(exp)	CLEAR	CLOSE #fileno Note:CLOSE #0 to close all files		COS(exp)
COMMODORE 64	ABS(exp)	ASC(string)	ATN(exp)		SYS(addr)		CHRS(exp)	CLR(exp)	CLOSE fileno	CONT	COS(exp)
MICROBEE	ABS(real-exp)	ASC(string)	ATAN(real-exp)	AUTO (lineno. val]			CHR(integer-exp)	STRS(int-exp) Note: set limits for string memory		CONT	COS(real-ex
PET	ABS(exp)	ASC(string)	ATN(exp)		SYS(addr)		CHRS(exp)	CLR	CLOSE fileno	CONT	COS(exp)
TRS-80/SYSTEM 80	ABS(exp)	ASC(string)	ATN(exp)	AUTO [lineno. val]			CHRS(exp)	CLEAR(exp) Note: Clears string space if exp given	[depends on OS: consult OS manual]	CONT	COS(exp)
VIC-20	ABS(exp)	ASC(string)	ATN(exp)		SYS addr		CHRS(exp)	CLR	CLOSE #fileno	CONT	COS(exp)
VZ200	ABS(exp)	ASC(string)	ATN(exp)				CHRS(exp)	CLEAR[exp] N clears string space		CONT	COS(exp)
ZX81	ABS(exp)	CODE(string) Note: ZX81 does not use ASCII code	ATN(exp)		LET var = USR (addr) Note: equivalent statement		CHRS(exp) Note: ZX81 does not use ASCII code	CLEAR	N/A — ZX81 does not support file-handling	CONT	COS(exp)
ZX SPECTRUM	ABS(exp)	CODE(string)	ATN(exp)		LET var = USR (addr) Note: roughly equivalent		CHRS(exp)	CLEAR	Consult Microdrive manual	CONTINUE	COS(exp)

Nov 83 4(11) 89-95 2 of 7.

RDS & FORMATS

DATA	DEF	DELETE	DIM	EDIT	END	EXP	FOR	FRE	GET	GOSUB	GOTO	IF/THEN/ELSE
Lists data to be used in a READ statement.	Define arithmetic string function.	Delete specified program lines.	Allocates space for arrays. specifies max subscript values.	Edit a program line.	Stop program & return to BASIC.	Raises to power of expression.	Used with NEXT to repeat a sequence of lines.	Returns remaining memory space.	Read a record from disk or tape	Branch to a Basic subroutine.	Branch to a specified line number.	If exp is true stmt is executed. If not ELSE or following line is executed.
DATA const [.const . . . }	DEF FNvar [(var,var . . .)] =exp	DELETE lineno [,lineno]	DIM var(sub), [,var(sub), . . .]	EDIT lineno	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	FRE(exp)	file GET [#] file-no [,record no]	GOSUB lineno	GOTO lineno	IF exp THEN stmt [ELSE stmt]
DATA CONST [,const . . .]	DEF FNvar (var) = exp	DEL lineno, lineno	DIM var(sub) [,var(sub) . . .]	[screen editing using CTRL keys]	END	EXP(exp)	FOR var = exp TO exp	FRE(exp)Note: exp is a dummy variable	INPUT var [,var . . .] NB: Get var(s) from current input device	GOSUB lineno/ var/exp	GOTO lineno	If exp THEN stmt Note: no ELSE
DATA const [,const . . .]			DIM [or COM] var (sub) [,var (sub) . . .] NB:dim'sion ALL strings	[cursor editing]	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	FRE(exp) Note: exp is a dummy variable	GET # fileno, record	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	If exp THEN stmt Note: no ELSE
DATA const [,const . . .]	DEF FNvar [(var. var)] = exp	DELETE lineno. lineno	DIM Var(sub) [,var(sub) . . .]	[cursor editing]	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	HIMEM-TOP	INPUT # fileno. record [,record . . .]	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	IF exp THEN stmt [ELSE stmt]
DATA const [,const . . .]	DEF FNvar =exp		DIM var(sub) [,var(sub) . . .]	[cursor editing]	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	FRE(exp) Note: exp is a dummy variable	GET #fileno, record [,record . . .]	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
DATA expr (,exp ("exp"))	FNn = exp	DELETE lineno. (lineno.)	DIM var(sub) (,var(sub))	EDIT (lineno.)	END	EXP (real=exp)	FOR var=exp TO exp (STEP exp)	FRE(0) mem. space FRE($) str. space		GOSUB NB: sq. br. significant	GOTO lineno	IF exp THEN stmt (ELSE stmt)
DATA const [,const . . .]	DEF FNvar (var) = exp)		DIM var(sub) [,var(sub) . . .)	[cursor editing]	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	FRE(exp) [TRS-80] is a dummy variable	INPUT # fileno. record [,record . . .]	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
DATA const [,const]	Various DEF statements available but none equivalent	DELETE lineno-lineno	DIM var(sub) [,var(sub) . . .]	EDIT lineno	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	FRE(exp) [TRS-80] or MEM [System 80]	INPUT # fileno. record [,record . . .]	GOSUB lineno	GOTO lineno	IF exp THEN stmt [ELSE stmt]
DATA const [,const . . .]	DEF FN(var) -exp		DIM var(sub) [,var(sub) . . .]	[cursor editing]	END	EXP(exp)	FOR var = exp TO exp [STEP exp]	FRE(exp) Note: exp is a dummy variable	GET # fileno, record	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
DATA const [,const . . .]			DIM var(sub) [,var(sub) . . .]		END	EXP(exp)	FOR var=exp TO exp [STEP exp]		INPUT # file-name var[,var . . .] NB: Gets record from tape	GOSUB lineno	GOTO LINENO	IF exp THEN stmt lineno [ELSE stmt /lineno]
			DIM var(sub)	EDIT Note: use cursor to select line		EXP(exp)	FOR var = exp TO exp [STEP exp]			GOSUB LINENO var/exp	GOTO LINENO var/exp	IF exp THEN stmt Note: no ELSE
DATA const [,const . . .]	DEF FNvar [(var.var . . .)] - exp		DIM var(sub)	EDIT (lineno) Note: cursor line by default		EXP(exp)	FOR var = exp TO exp [STEP exp]		Consult Microdrive manual	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	IF exp THEN stmt Note: no ELSE

Nov 83 4(11) 89-95
3 of 7.

BASIC RESERVED WO

STANDARD MICROSOFT	INKEY$ Returns character typed at keyboard or null if no character typed.	INPUT Read data from terminal.	INT Evaluates expression for largest integer contained.	LEFT$ Returns specified no. of characters starting at beginning of string.	LEN Gives decimal length of string.	LET Gives a value to a variable.	LIST List specified program lines at terminal.	LLIST List specified program lines at printer.	LOAD Load a program file into memory.	LOG Natural logarithm of expression.	MID$ Gives specified of characters to right of start position in string
MACHINE	INKEYS	INPUT [STRING;] var [,var . . .]	INT(exp)	LEFTS(string, length)	LEN (string)	[LET] var=exp	list [lineno, lineno]	LLIST [lineno, lineno]	LOAD ["filename"]	LOG(exp)	MIDS(string, [,length])
APPLESOFT	GET VAR	INPUT[STRING,] VAR [,VAR . . .]	INT(exp)	LEFTS (string) LENGTH)	LEN(string)	[LET] var = exp	LIST [Lineno, lineno] Note '—' may be used in place of ','	[depends on interface arrangement—usually LIST"P]	LOAD FILENAME	LOG(exp)	MIDS(string, start[,length])
ATARI		INPUT [exp] var [,var . . .] or INPUT [exp] string-var	INT(exp)	string (start, length)	LEN(string)	[LET] var=exp	LIST [lineno, lineno]	LIST "P"	CLOAD ["file-name"][cass] or LOAD "Dn:file-name" (disk)	LOG(exp)	string(start [,length])
BBC MICRO	GET var (unlimited time) or INKEYS (time) Note: 100ths sec.	INPUT [string [,]] var [,var . . .]	INT(exp)	LEFTS(string, length)	LEN(string)	[LET] var=exp	LIST [lineno-lineno]	CTRL-B then LIST [lineno-lineno]	LOAD "filename" Note "*DISK" or "*TAPE" to select device	LN(exp) NB: LOG(exp) gives common rather than natural log	MIDS(string, start[,length])
COMMODORE 64	GET var	INPUT [string;] var [,var . . .]	INT(exp)	LEFTS(string, length)	LEN(string)	[LET] var = exp	LIST [lineno-lineno]	OPEN 4,4: CMD4: LIST [lineno-lineno]	LOAD ["file-name"] [cass] or LOAD "filename" 8 [disk]	LOG(exp)	MIDS(string, start[,length])
MICROBEE	KEY	INPUT (string) var (,var)	INT(real-exp)	var(;1, length)	LEN(string)	(LET) var=exp LET obligatory after THEN and ELSE	LIST (lineno. (,lineno)) forceloads	LLIST (lineno. (,lineno))	LOAD (U) (?) ("filename") LOAD U	LOG(real - exp)	var(;n,n+m-1), -n-start charac m-length
PET	GET var	INPUT [STRING, var [,var . . .]	INT(exp)	LEFTS (string, length)	LEN(string)	[LET] var = exp	LIST [Lineno-lineno]	OPEN 4.4: CMD4: LIST [lineno-lineno]	LOAD["file-name"] [cass] or LOAD "filename", 8 [disk]	LOG(exp)	MIDS(string, start[,length])
TRS-80/SYSTEM 80	INKEYS	INPUT [string;] var [,var . . .]	INT(exp)	LEFTS(string, length)	LEN string)	[LET] var = exp	LIST [lineno-lineno]	LLIST [Lineno-lineno]	CLOAD "[file-name]" [cass] or LOAD "filename" [disk floppy tape]	LOG(exp)	MIDS(string, start, length])
VIC-20	GET var	INPUT [string;] var [,var . . .]	INT(exp)	LEFTS(string, length)	LEN(string)	[LET] var = exp	LIST [lineno-lineno]	OPEN3,4:CMD 3: LIST [lineno-lineno]	LOAD ["file-name"][cass] or LOAD "filename", 8 [disk]	LOG(exp)	MIDS(string, start[,length])
VZ200	INKEYS	INPUT[string;] var [,var . . .]	INT(exp)	LEFTS (string, length)	LEN (string)	[LET] var = exp	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD ["file-name"]	LOG (exp)	MIDS (string, start, [,len])
ZX81	INKEYS	INPUT var	INT(exp)	string(TO finish)	LEN(string)	LET var = exp	LIST [lineno]	LLIST [lineno]	LOAD "["filename"]	LN(exp)	string(start TO finish)
ZX SPECTRUM	INKEYS	INPUT [string;] var	INT(exp)	string (TO finish)	LEN(string)	LET var = exp	LIST [lineno] Note: will fill screen then ask SCROLL?	LLIST [lineno]	LOAD "filename" [cass] Note: Microdrive command for disk	LN(exp)	string(start TO finish)

Nov 84 4(11) 89-95
4 of 7.

RDS & FORMATS

NAME	NEW	NEXT	ON ERROR	ON/GOSUB	ON/GOTO	OPEN	OUT	PEEK	POKE	PRINT	RANDOMIZE	READ
Rename a file.	Delete current program & data from memory.	End of FOR/NEXT loop.	Error trap subroutine.	GOTO lineno specified by evaluation of expression.	GOTO lineno specified by evaluation of expression.	Open disk file.	Put specified byte to specified output port.	Read byte from specified memory location.	Put specified byte to specified -- memory address.	Write data to disk file.	Reset random number generator.	Read from data statements into specified variables.
NAME "filename" AS "filename"	NEW	NEXT var [,var . . .]	ON ERROR GOTO lineno	On exp GOSUB lineno [,lineno . . .]	On exp GOTO lineno [,lineno . . .]	OPEN mode [#] fileno "filename"	OUT port,byte	PEEK (addr)	POKE addr,byte	PRINT [[#] fileno][exp] [,exp . . .]	RANDOMIZE [exp]	READ var [,var . . .]
RENAME oldname, newname	NEW	NEXT [var, var . . .]	ONERR GOTO lineno	On exp GOSUB lineno [,lineno . . .]	On exp GOTO lineno [,lineno . . .]	OPEN filename		PEEK(addr)	POKE addr,byte	PRINT exp [,exp . . .] NB: prints to current output device		READ var [,var . . .]
	NEW	NEXT var	TRAP lineno/ var/exp	ON exp GOSUB lineno [,lineno . . .]	ON EXP GOTO lineno [,lineno . . .]	OPEN #fileno, mode control code, filename	[not equivalent]	PEEK(addr)	POKE addr,byte	PRINT #fileno, record [,record . . .]	RND(-exp)	READ var [,var . . .]
	NEW Note: under cert. circum. may be recovered using OLD	NEXT [var, var . . .]	ON ERROR stmt	ON exp/var GOSUB lineno [,lineno . . .]	ON exp/var GOSUB lineno [,lineno . . .]	fileno-OPENIN [to read] or fileno-OPENOUT [to write]		?addr NB: '?' does NOT mean 'print' in BBC Basic	?addr,byte	PRINT #filename, record [,record . . .]	RND(-exp)	READ var [,var . . .]
OPEN 1,B,15, "RO: filename-filename"[disk only]	NEW	NEXT [var, var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN # exp, fileno, mode, "filename"		PEEK(addr) BYTE	POKE ADOR,	PRINT #fileno, record [,record . . .]	RND(-TI)	READ var [,var . . .]
	NEW	NEXT var NEXT *var lineno. -exits loop before completion	ON ERROR GOTO lineno.	ON exp GOSUB ((exp(,exp))]lineno (,([exp . . .	ON exp GOTO lineno (,lineno.)		OUT port,byte	PEEK(address)	POKE address, byte	PRINT list		READ ((lineno.)) var(,var)
RENAME [fileno,] "oldname" TO "newname"	NEW	NEXT [var,var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN #exp, mode, "filename" fileno, mode		PEEK(addr)	POKE addr,byte	PRINT #fileno, record [,record . . .]	RND(-TI)	READ VAR [,var . . .]
[depends on OS; consult OS manual]	NEW	NEXT [var,var . . .]	ON ERROR GOTO lineno	ON exp GOSUB lineno [,lineno . . .]	ON EXP GOTO lineno [,lineno . . .]	[depends on OS; consult OS manual.]	OUT Port,byte	PEEK(addr)	POKE addr,byte	PRINT #-fileno, record [,record . . .] [cass]	RANDOM	READ var [,var . . .]
	NEW	NEXT [var,var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN exp,fileno, mode, "filename"		PEEK(addr)	POKE addr,byte	PRINT #fileno, record [,record . . .]	RMD(-TI)	READ var [,var . . .]
	NEW	NEXT[var]					OUT port,byte	PEEK(addr)	POKE addr,byte	PRINT#"filename", exp[,exp . . .] NB prints to tape		READ var[,var . . .]
	NEW	NEXT var						PEEK(addr)	POKE addr,byte	—	RAND(exp)	
	NEW	NEXTvar				Consult Microdrive manual	OUT post,byte	PEEK(addr)	POKE addr,byte	Consult Microdrive manual	RAND(exp)	READ var [, var . . .]

Nov 84 4(11) 89-95
5 of 7.

BASIC RESERVED WO

STANDARD MICROSOFT MACHINE	REM Used to insert comments on a program which the computer ignores.	RENUM Change program line numbers.	RESTORE Resets pointer to facilitate re-reading of DATA statements.	RESUME Return from ON ERROR subroutine to stmt that caused error.	RETURN Return from subroutine to statement following last GOSUB executed.	RIGHT$ Returns specified no. of characters starting at end of string.	RND Generates a random number.	RUN Execute a program.	SAVE Save a program either onto disk tape.	SGN Returns 1 if exp >0 0 if exp = 0 -1 if exp <0	SIN Sine of expression in Radians.
	REM text	RENUM [lineno, val]	RESTORE	RESUME	RETURN	RIGHTS(string, length)	RND(exp)	RUN [lineno]	SAVE filename	SGN(exp)	SIN(exp)
APPLESOFT	REM text		RESTORE	RESUME	RETURN	RIGHTS(string, length)	RND(exp) Note: exp is a dummy variable	RUN [lineno]	SAVE filename, fileno	SGN(exp)	SIN(exp)
ATARI	REM text		RESTORE [lineno]		RETURN	string(start) NB not strictly equivalent	RND(exp) Note: exp is a dummy variable	RUN	CSAVE "filename" [cass] or SAVE "filenc:filename" [disk]	SGN(exp)	SIN(exp)
BBC MICRO	REM text	RENUMBER [start] [interval]	RESTORE(exp)		RETURN	RIGHTS(string, length)	RND(exp)	RUN	SAVE "filename" Note: see note under LOAD	SGN(exp)	SIN(exp)
COMMODORE 64	REM text		RESTORE		RETURN	RIGHTS(string, length)	RND(exp)	RUN [lineno]	SAVE ["filename"][cass] or SAVE "filename", 8 [disk]	SGN(exp)	SIN(exp)
MICROBEE	REM text	RENUM (new-start (,increment (,start-line (,finish-line))))	RESTORE (lineno.)		RETURN	var(;LEN(var)-n-1) -n - number of characters required	RND	RUN	SAVE "filename" - 300 bpi SAVEF "filename" -1200 bpi	SGN(real-exp)	SIN(real-exp)
PET	REM text		RESTORE		RETURN	RIGHTS(string, length)	RND(exp)	RUN	Save("filename") [cass] or SAVE "[filenc:]filename", 8 [disk]	SGN(exp)	SIN(exp)
TRS-80/SYSTEM 80	REM text	RENUM Start interval Note: System 80 only	RESTORE	RESUME [lineno]	RETURN	RIGHTS(string, length)	RND(exp)	RUN [Lineno]	CSAVE "filename" [cass] or SAVE "filename" [disk floppy tape]	SGN(exp)	SIN(exp)
VIC-20	REM text		RESTORE		RETURN	RIGHTS(string, length)	RND(exp) Note: exp is a dummy	RUN [LINENO]	SAVE["filename", control code][cass] or SAVE "filename", 8 [disk]	SGN(exp)	SIN(exp
VZ200	REM text		RESTORE		RETURN	RIGHTS(string, exp)	RND(exp) NB Nonstandard — see VZ200 manual P58	RUN[lineno]	CSAVE["filename"]	SGN(exp)	SIN(exp)
ZX81	REM text				RETURN	string(start TO)	RND	RUN [lineno/ var/exp	SAVE "filename"	SGN(exp)	SIN(exp)
ZX SPECTRUM	REM text		RESTORE [lineno/ exp]		RETURN	string(start TO)	RND	RUN [lineno/ var/exp]	SAVE "filename" [cass] Note: Microdrive manual for disk	SGN(exp)	SIN(exp)

Nov 84 4(11) 89-95
6 of 7.

RDS & FORMATS

SQR Square root of expression.	STOP Stop program execution & return to command mode.	STRINGS Returns a string of specified length containing specified character.	STR$ Converts a numeric expression to a string.	SYSTEM Close files for return to operating system.	TAN Tangent of expression in radians.	TROFF Trace off.	TRON Trace on.	USR Calls an assembler language subroutine which returns one value.	VAL Gives numeric value of string of ASCII numbers.	WAIT Suspend program execution for specified time.	WHILE/WEND Execute statements in WHILE/WEND loop as long as exp is true	WIDTH Sets printer carriage/screen width.
R(exp)	STOP	STRINGS(length, string)	STRS(exp)	SYSTEM	TAN(exp)	TROFF	TRON	USR(parameter)	VAL(string)	WAIT port, mask [,select]	WHILE exp WEND	WIDTH(exp)
R(exp)	STOP		STRS(exp)		TAN(exp)	NOTRACE	TRACE	USR(parameter)	VAL(string)	WAIT ADDR. exp [,exp]		POKE 32, left margin: POKE 33, screen width
SQR(exp)	STOP		STRS(exp)	BYE NB: not equivalent				USR(parameter)	VAL(string			POKE 83, val [left margin]:POKE 83, val [right margin]
SOR(exp)	STOP	STRINGS(length, string)	STRS(exp)	*DISK NB: disk-handling done through Basic so not true eq.	TAN(exp)	TRACE OFF	TRACE ON	USR(parameter)	VAL(string)	[no WAIT stmt but see INKEYS]	REPEAT stmt UNTIL exp Note: reverse logic	WIDTH val Note: 0-'unlimited'
QR(exp)	STOP		STRS(exp)		TAN(exp)			USR(parameter)	VAL(string)	WAIT addr, exp,exp		
QR(real-exp)	STOP	PRINT [An m] -n = length of string; m = ASCII code of character	STR(exp)			TRACE OFF	TRACE ON	USR(address (, integer-exp))	VAL(string-exp)	PLAY 0, int (1(int(255; 1 - 1/8 second)		ZONE (integer-exp) 1 ≤ integer-exp ≤ 16
QR(exp)	STOP		STRS(exp)		TAN(exp)			USR(parameter)	VAL(string)	WAIT addr, exp,exp		
SQR(exp)	STOP	STRINGS(length, string)	STRS(exp)	SYSTEM plus code following prompt Note: not equivalent	TAN(exp)	TROFF	TRON	USR(parameter)	VAL(string)			
SQR(exp)	STOP		STRS(exp)		TAN(exp)			USR(parameter)	VAL(string)	WAIT addr, exp,exp		
QR(exp)	STOP		STRS(exp)		TAN(exp)			USR(parameter)	VAL(string)			
QR(exp)	STOP		STRS(exp)		TAN(exp)			USR(addr)	VAL(exp)	PAUSE exp Note: halts screen display only		
QR(exp)	STOP		STRS(exp)		TAN(exp)			USR addr	VAL(string)	PAUSE no. of frames (50/second)		

Nov 84 4(11) 89-95
7 of 7.

A BEGINNER'S GUIDE TO PROGRAM CONVERSION
PART 2:SIMULATING STATEMENTS

In the October issue of APC Surya looked at the factors to consider when choosing between a program conversion and a complete rewrite. In the November issue he followed that up with the Basic Converter Chart and now he continues the series on the conversion of one Basic dialect to another with the assumption that a conversion is appropriate and analyses the procedure in detail.
Next month Surya will continue with a look at graphics and sound conversion.

The initial steps to be taken when converting a program from one dialect of Basic to another are much the same as when coding from scratch and just as much discipline is required. The starting point in either case is to have a clear understanding of what you're setting out to achieve. Make sure you can follow the logic of the program before you attempt to modify it. Spend a little time working out why the author has done things in that particular way. All this may seem unnecessary at first, but it's time well spent: the greater your understanding of the program, the easier the conversion will be.

Once you're satisfied that you have a clear overview of the program as a whole, you can look at each section in detail. Break the program down into its component subroutines. This is only possible with a reasonably structured program, but as mentioned in the October issue, programs with poor or non-existent structuring are best left alone.

When examining each routine, take a special look at the variables. Determine which are global and which are local. Global variables are those used throughout the program. Typical global variables include scores in games, some counters, printer-settings and so on. Local variables are those whose values are used only within a given subroutine: once the routine has been exited, the values are no longer required and the variables may be used for a different purpose within another routine. Typical local variables are counters in FOR-NEXT loops and flags used to check validity of data.

The reason you need to distinguish between the two is that local variables may be freely changed or discarded as appropriate, but global variables need to be treated with a great deal of care — the program as a whole is dependent upon them. If you're lucky, the programmer will have gone to the trouble of listing all global variables in remarks at the beginning of the program, and used fixed local variables so that, for example, w is always a FOR-NEXT loop counter. Failing that, there are utility programs available that will locate variables for you.

Coding

(Note: in the examples given below, I am using A$ to represent any string variable and 100 onwards whenever line numbers are required. These choices are purely arbitrary and have no significance.)

During the process of converting a program from one machine to another, you will very often come across a keyword in the original program for which your machine has no equivalent. While experienced programmers will soon find a way round the problem, those a little newer to the game may find themselves stuck for a solution. What I have done below is to look at some of the common offending statements and methods of achieving the same effect using standard Microsoft. The keywords covered are not in any particular order.

INKEY$: This statement is an almost statutory presence in just about every Basic program ever written. This statement tells the computer to scan the keyboard to test for a key depression and place the result into a specified variable. The standard format is A$=INKEY$; the most common variations are A$=GET$, GET$=A$ and GET A$.

The statement takes one of two forms. On most machines, the processor will carry out a single sweep of the keyboard: if a key is pressed during this scan, the value of the key pressed will be placed into the variable A$. If no key is pressed, A$ will be null (empty). On some machines, however, the computer will carry out a continual series of sweeps until a key-press is detected. A few machines offer both forms.

A continuous scan using the former version of inkey$ is straightforward: 100 A$=INKEY$:IF A$="" THEN GOTO100. The BBC, however, goes a step further in offering a timed keyboard scan in the form A$=INKEY$(time), where time is given in 100ths of a second. To simulate this using the standard INKEY$ statement, we use a FOR-NEXT loop thus: 100 FOR A=0 TO (value):A$=INKEY$:NEXT. The value of the variable will need to be adjusted to suit. Since different machines have different processing speeds, you'll have to experiment with different values to establish some kind of relationship between the value of the FOR-NEXT counter and real time.

Of course, the example given above would return the final key pressed if there were two or more key depressions during the scan period, but this is easily overcome:

```
100 FLAG=0:A$=""
110 FOR A=0 TO (value)
120 B$=INKEY$:IF NOT B$="" AND
      FLAG=0 THEN A$=B$:FLAG=1
130 NEXT
```

The value of the first key depression is now stored in A$. If no key was pressed, then A$ will be empty.

INSTR: This statement is used to search one string to find out whether it contains a second string. The format is INSTR(main string, sub-string) where the starting position of the sub-string is returned on a successful match and 0 is returned if the search fails. INSTR("APC","C") would return 2 while INSTR("APC","X") would return 0.

We might, for example, want to find out whether NAME$ contains the sub-string 'Rev.'. Using INSTR, we would do this like so:

```
100 IF  NOT(INSTR(NAME$,"Rev.")
      =0) THEN PRINT NAME$;" is a
      vicar."
```

To simulate this in standard Microsoft, we use MID$. In the above example, we would do so thus:

```
100 FLAG=0:FOR     A=1      TO
      (LEN(NAME$)-4)
110 IF    MID$(NAME$,A,4)="Rev."
      THEN FLAG=1
120 NEXT
130 IF   FLAG=1   THEN   PRINT
      NAMES:"is a priest."
```

Note that on an Atari, line 110 would read as follows:

```
110 IF  NAME$(A,4)="Rev."  THEN
      FLAG=1
```

Feb 84 5(2) p 140-141   1 of 2.

and on a Sinclair machine, it would read:

```
110 IF NAME$(A TO A+4)="Rev."
    THEN FLAG=1
```

These differences are due to the non-standard forms of MID$ supported by these machines. The original example should work on all other dialects of Basic.

PROCEDURES AND FUNCTIONS: User-definable functions are supported in varying degrees of sophistication by a number of machines, but you are most likely to come across the extended use of procedures and functions in BBC programs. Procedures and functions make programs infinitely neater and more readable, but they don't actually achieve anything which cannot be duplicated using ordinary sub-routines.

Some dialects of Basic will allow you to GOTO or GOSUB a variable which greatly aids readability — the Basic Converter Chart will tell you which machines do if you look under GOTO.

Sharp Basic SP-5025 has a number of weaknesses which are discussed in the article 'Sharp Logic' in the September issue.

REPEAT-UNTIL and WHILE-WEND. These are two forms of the same control loop, one being the logical reverse of the other. WHILE-WEND checks that a given expression is true and then executes all statements up to the first WEND statement encountered. The computer then returns to the original condition to check whether it is still true. If the condition is false, the statement following the WEND statement is executed.

For example:

```
100 REM — Silly example
110 X=10
120 WHILE X>0
130 PRINT "The current value of X
    =";X;"."
140 X=X-1:WEND
150 REM – X is now zero and the WHILE
    test fails
```

In a WHILE-WEND loop, the loop is repeated while the test expression is true. A REPEAT-UNTIL loop works the other way around. All statements between REPEAT and UNTIL are executed until the test expression is true. Thus the above example would be written:

```
100 REM — Same silly example
110 X=10
120 REPEAT
130 PRINT "The current value of X
    =";X;"."
140 X=X-1:UNTIL X=0
150 REM – X is now zero and the
    REPEAT test is satisfied
```

Converting from one structure to the other is thus straightforward. But the majority of present-day Basics offer neither of the above. To create the same effect, we have to use a statement that causes purists to gasp in horror : the GOTO.

Thus:

```
100 REM — Here we go again
110 X=10
120 PRINT "The current value of X
    =";X;"."
130 IF X>0 THEN X=X-1:GOTO120
140 REM – X is now zero and the test fails
```

While somewhat less elegant, the net result is the same. We can see that rewriting a WHILE-WEND or REPEAT-UNTIL structure is simply a matter of manually inserting the test (using IF-THEN) and pointer (GOTO).

**STRING$** is a statement which allows you to repeat a given sequence of characters. The format is STRING$(number of times to print string,string). If you wanted to print a line of asterisks across an 80-column screen, for example, you would state: STRING$(80,"*"). If your machine doesn't support this statement, then we fall back once again on the ever ready FOR-NEXT loop. Thus: FOR A=1 TO 80:PRINT"*";:NEXT, the string is simply duplicated, and the numeric argument placed in the FOR-NEXT loop.

TAB. This is supported by most machines, except that on the BBC micro the TAB function is performed by SPC while TAB prints in predetermined screen fields.

APC. Feb 84 5(2) p 140-141 2 of 2.

This article was published in APC Nov 83.
However this version appears to be more complete
e.g. see TAB command.

A BEGINNER'S GUIDE TO PROGRAM CONVERSION
PART 3:APPLE II GRAPHICS

Surya begins the graphics supplement to the APC Basic Converter Chart with a look at the Apple II.

Applesoft supports no less than four forms of tab statement: SPC, TAB, HTAB and VTAB. SPC (x) prints x spaces. So, SPC(10);"Hello" would move the cursor ten columns forward and then print 'Hello'. TAB (x) moves the cursor to column x. If x is less than the current cursor column, then the statement is ignored. Thus SPC moves the cursor relative to its current position, wrapping around lines as necessary, whereas TAB moves to the absolute screen column specified.

HTAB (Horizontal TAB) is similar to TAB, but can move left as well as right. HTAB (x) moves the cursor to column regardless of the cursor's current position. VTAB (Vertical TAB) is used to position the cursor vertically. VTAB (x) moves the cursor to line x leaving its column position unchanged.

As an example:

```
100 REM: Tabulating on an Apple II
110 HOME: REM clear screen, position
        cursor top-left.
120 PRINT TAB(10); "Line 1, column
        10"
130 PRINT "Line 2, column 0";
SPC(5); "column 22"; HTAB(16);
"and";
140 REM Above line would appear on
        screen as Line 2, column 0 and
        column 22
150 PRINT VTAB(12); HTAB(19); "*":
        REM centre of 40-column screen
160 END
```

To find the current cursor position, the POS (POSition) statement is used. POS (x) returns the current cursor column. The expression x is a dummy value (that is, the value has no effect) but must be a valid expression which Applesoft can evaluate.

INVERSE switches on the inverse video attribute, and is cancelled by the NORMAL statement. So:

```
100 HOME
110 INVERSE
120 PRINT "This will be printed in
        inverse"
130 NORMAL
140 PRINT "This will be printed
        normally"
150 END
```

FLASH works in a similar fashion to INVERSE, switching on the flashing attribute:

```
100 HOME
110 FLASH
120 PRINT "This text will flash"
130 NORMAL
140 PRINT "And this text won't!"
150 END
```

Finally, the SPEED statement allows the user to control the speed at which text is displayed on the screen. By default, the Apple prints text to the screen as fast as it can, but other speeds can be selected. Slow speeds (<100) are useful for displaying instructions and so on, where the display speed is set to the average reading speed.

The statement takes the form SPEED=x, where x is an expression between 0 (slowest) and 255 (default):

```
100 HOME: SPEED=0
110 PRINT "This will be printed very
        slowly . . . "
120 SPEED=255
130 PRINT "And this will be printed at
        the normal speed"
140 END
```

The easiest way to simulate slow printing on other machines is to place the With many things in the microcomputing world, there are agreed standards. The ASCII code for communications; the RS232, Centronics and IEEE for interfacing; the 5.25in disk and so forth. But when it comes to graphics it seems that manufacturers and designers don't know the meaning of the word 'standard'. The reason for this is simple. In the time it would take to debate, argue, redesign and eventually implement a set of standards, the graphics capabilities of the machines being developed would have increased beyond all recognition, rendering the standards useless.

Different machines not only use different screen resolutions, but the range of graphics-handling statements supported varies from simple SET, RESET and POINT to a whole array of sophisticated features like drawing circles and filling-in shapes. All this is a rather roundabout way of saying that it is not possible to cover the subject of graphics in the form of a quick-reference chart as with the *APC* Basic Converter Chart. (See November '83, *APC.*)

What I have set out to do in this series of articles is to give you enough information about the graphics-handling of each machine covered by the chart to enable you to work out what is happening in a listing.

Incidentally, as a general tip when converting graphics, I recommend mapping out a picture of the graphics screen of the machine from which you are converting on square-ruled paper, marking on it rough values. Next, place a piece of tracing paper over this grid and follow the listing through, sketching in lines and text. You can then place this tracing paper over a map of your own screen to see roughly what values you will need to use.

The complexity of micros' graphics often make program listings for one machine all but incomprehensible to the owners of other computers. There are a lot of well written listings in *APC* for a variety of machines which readers would no doubt like to get up and running on their own micros. For this reason it is worthwhile going into the subject of graphics in a fair amount of detail.

The Apple Family

The Apple II has three variations: the Apple II, the Apple II+ and the Apple IIe. All three support Applesoft Basic and therefore use the same graphics handling statements.

Applesoft supports three screen modes — text, low-resolution graphics and high-resolution graphics. These are called by the statements TEXT, GR and HGR respectively.

Text

The normal text screen comprises 24 lines by 40 columns. An 80-column screen is available by installing an optional circuit board; and *APC* programs written for an 80-column machine will have this clearly stated in the accompanying notes.

Text mode has ten statements which may be used to format text output on the screen:

HOME clears the screen and positions the cursor at the top-left corner. On most machines, this is achieved by the statement CLS.

Mar 84 5(3) 42-43 1 of 2.

text into DATA statements and use a FOR-NEXT loop to print one character at a time. A delay loop is used after each character is printed to achieve the reduced speed:

```
100 REM: This solution is designed to
     be portable, not elegant!
110 FOR a=1 TO 3: REM number of
     data statements to read
120   READa$: REM read line of text to
       be printed
130   FOR b=1 to LEN(a$)
140     PRINT MID$(a$,b,1);:  REM
         print one character of a$
150     FOR c=1 to 12: NEXT: REM
         empty loop to cause delay
160     REM adjust value of above
         loop to vary speed
170   NEXT b: REM repeat for next
       character in data statement
180   PRINT: REM move cursor onto
       next line
190 NEXT a: REM repeat for next data
     statement
200 DATA This text will be printed
     slowly
210 DATA So will this
220 DATA And this
230 END
```

Low resolution graphics (GR)

The low-resolution screen on the Apple is addressed as 40 columns by 48 rows. Sixteen colours are available. The bottom four lines (8 rows) are normally reserved for text, but the oft-used POKE-16302,0 makes these available for graphics use. (The CALL-1998 statement which usually follows the above POKE simply sets the extra rows to black.

Once in GR mode, there are five graphics statements available:
COLOR=x sets the foreground colour, where x is in the range 0-15 and is defined:

0	black
1	magenta
2	dark blue
3	violet
4	dark green
5	grey
6	medium blue
7	light blue
8	brown
9	orange
10	a different shade of grey!
11	pink
12	green
13	yellow
14	aqua
15	white

Although I just said that x must be in the range 0-15, it is possible to use any value up to 255. But since 16 is equivalent to 0, 17 to 1 and so on, this fact is not spectacularly useful.

PLOTx,y is used to light up the specified block in the current foreground colour, where x is the column and y is the row. In GR mode, the origin (0,0) is top-left.

HLIN x1,x2 AT y is used to draw a Horizontal LINe in the current foreground colour from (x1,y) to (x2,y) where x1 and x2 are different column numbers and y is the row.

VLINy1,y2 ATx — of course — draws a Vertical LINe from (x,y1) to (x,y2) where x is the column number and y1 and y2 are different rows.

SCRN (x,y) returns the code of the colour at position (x,y). On most machines, this is achieved using a POINT (x,y) statement.

Next month: Apple II high resolution graphics and sound, and the TRS-80/ System 80.

APC Mar 84 5(3) p 42-43.

As promised last month, we continue the Apple II guide with high-res graphics and sound.

High resolution graphics (HGR)

The HGR screen is addressed as 280 columns by 192 rows with six colours available. The Apple reserves enough memory for two high resolution screens, these being called by HGR and HGR2 respectively. Four text lines are again reserved by default and can be made available for graphics use by the statement POKE — 16302,0 and reset to text by POKE — 16301,0.

Two POKEs which you are likely to find in Apple programs using the HGR mode are:
POKE — 16300,0 to switch from HGR2 .back to HGR
— 16303,0 to switch from graphics to text retaining text-windows and cursor position.

In HGR mode, there are two main graphics statements: HCOLOR and HPLOT. HCOLOR=x sets the foreground colour to x, defined as:

0	black
1	green
2	violet
3	white
4	black
5	orange
6	blue
7	white

Although there are eight codes, two are redundant (4 and 7), leaving six effective colours.

HPLOT is an easy-to-follow statement operating in a similar way to most machines DRAW statements:

HPLOT x,y lights point (x,y) in the current colour.
HPLOTx1,y1 TOx2,y2 draws a line from (x1,y1) to (x2,y2). Cordinates can be 'chained', so that the following HPLOT statement:
HPLOT 0,0 TO 279,0 TO 279,191 TO 0.191 TO 0.0
draws a rectangle around the edge of the screen. Most basics don't allow this type of chaining, so you'd have to split up each pair of coordinates and DRAW, SET or PLOT each line separately.

HPLOT TO x,y draws a line from the current cursor position to coordinate (x,y); it carries on from where it last left off.

There are seven other graphics statements in Applesoft HGR mode: DRAW, XDRAW, SCALE, ROT, SHLOAD, BSAVE and BLOAD. These statements concern a feature known as shape tables. Shape tables are too complex to go into in the space available here and, in any case, the information wouldn't be much use to owners of other machines since you will find them all but impossible to duplicate.

Shape tables are a form of sprite, a kind of sophisticated use-definable character. Created by POKEing values into memory, shape tables may be saved to tape or disk for later loading. The scale and orientation of the resultant shapes can be manipulated using the statements mentioned above. Anyhow, unless you are very familiar with both Applesoft and the machine you are tran slating to, any program making liberal use of DRAW, XDRAW, SCALE, ROT, SHLOAD, BSAVE or BLOAD should be left well alone.

Sound

There are only two ways to produce sound on Apple: PRINT CHR$(7) and POKEing memory location — 16336. PRINT CHR$(7) produces a short beep, as with most machines. Producing anything interesting from the noises emitted by POKEing — 16336 is a decidedly frustrating and not over-fruitful task, so this POKE may be safely omitted when converting to other machines.

APC Apr 84 5(4) p 72. 2of2.

A BEGINNER'S GUIDE TO PROGRAM CONVERSION

TRS-80/System 80

Surya continues his analysis of each machine on the APC Converter Chart (see November 1983 issue). High and low resolution graphics and sound capabilities for the System 80 and TRS-80 Model 100 are featured this month, plus the final part of the Apple II conversion.

The TRS-80 has limited graphics facilities; not surprising when you look at how long the machine has been around. The graphics resolution is 64 x 48, the origin (0,0) being at the top left-hand corner of the screen. Thus:

```
100  Rem: A totally pointless program
110  CLS: Y=0
120  FOR X=0 TO 63 STEP 1.3
130        Y=Y+1
140        SET ((INT(X)),Y)
150  NEXT X
```

draws a line diagonally across the screen.

The graphics statements are SET, RESET and POINT. SET(x,y) lights the block at coordinate (x,y). RESET switches it off again. POINT(x,y) tests the specified point, returning — 1 if it is lit and 0 if it is not.

The TRS-80 also supports a PRINT @ statement. This allows text to be placed at a specified location on the screen. For the purposes of the PRINT @ statement, the top row of the screen is numbered from zero at the left-hand side to 63 at the right. The next line is numbered 64 to 127, and so on to the bottom line, 960 to 1023. To print at the bottom line, for example, you simply PRINT @ 960, thus:

```
100 PRINT @ 960, "This is printed on the bottom line";
```

The semi-colon at the end of the PRINT statement supresses the line feed which would otherwise scroll the screen upwards.

The TRS-80 does not support sound as standard.

The System 80

The System 80 is an oriental imitation of the American TRS-80. Unlike most imitations, however, the System 80 is every bit as good as the original. The TRS-80 is slightly fussier about syntax that the System 80, but the two are all but identical. Most Basic programs are interchangeable. In *APC's* Programs section, the label TRS-80/System 80 is used to describe programs written on either machine.

The TRS-80 Model 100

The TRS-80 Model 100 is Tandy's portable micro. The graphics resolution is 239 x 63, and the graphics commands are PSET, PRESET and LINE. PSET and PRESET are exact equivalents of SET and RESET. Considering that LCD screens are not noted for wonderful graphics, the LINE statement is surprisingly powerful.

The format of the statement is LINE (x1,y1)—(x2,y2), a, BF. The statement draws a line from the first coordinates to the second. If a=1, the line is PSET; if 0, it is PRESET. The additions B and F are optional. If B is included, than a B)ox will be drawn with (x1,y1) as one corner and (x2,y2) as the other. If the F is included, the box will be F)illed — either PSET or PRESET, depending on the value of a.

The model 100 also supports sound (of the beep variety). BEEP beeps. 'SOUND pitch,length' plays the specified note and is similar to most sound statements.

APC Apr 84 5(4) p 71-72.

(p. 72 contained hi-res graphics for Apple II and is included with that article.)

A BEGINNER'S GUIDE TO PROGRAM CONVERSION

Atari

This month, Surya continues his analysis of each machine on the APC Convertor Chart with a look at graphics and sound capabilities on the Atari microcomputers.

The Atari is available in Australia in three forms: the 400, 800 and most recently, the 600XL. The three models are upward-compatible, and all have the same graphics capabilities.

The Atari supports nine different screen modes, numbered 0 to 8. Of these, the first three are text modes, the rest graphics. A summary of the modes is given in Fig 1.

The statement GRAPHICS x is used to select the desired mode, Mode 0 is the default.

Mode	Type	Resolution Full screen	Split screen	Colours	RAM required
0	Text	40x24	Not available	2	993
1	Text	20x24	20x20	5	513
2	Text	20x12	20x10	5	261
3	Graphics	40x24	40x20	4	273
4	Graphics	80x48	80x40	2	537
5	Graphics	80x48	80x40	4	1017
6	Graphics	160x96	160x80	2	2025
7	Graphics	160x96	160x90	4	3945
8	Graphics	320x192	320x160	1	7900

Fig 1. Atari screen modes

In Fig 1, I refer to full screen and split screen. Normally, in a graphics mode, the bottom lines of the screen are reserved for text. By adding 16 to the mode, this text window can be converted to graphics use. Thus, GRAPHICS 2+16, or GRAPHICS 18, selects mode 2 without a text window. In a graphics mode, PRINT prints to the text window, while PRINT#6, prints to the graphics area.

The Atari has a 'palette' of 16 colours, these being known as hues. The hues are numbered from 0 to 15:

0	grey	8	blue
1	gold	9	light-blue
2	orange	10	turquoise
3	red-orange	11	green-blue
4	pink	12	green
5	purple	13	yellow-green
6	purple-blue	14	orange-green
7	blue	15	light orange

Colour register

A maximum of five colours may be displayed at any one time, and this only in modes 1 and 2. Therefore, Atari gives us a 'working palette' of five colours from which to choose; these are known as the colour registers. The colour register defaults are shown in Fig 2.

Colour register	Default hue number	Physical colour
0	2	Orange
1	12	Green
2	9	Dark blue
3	4	Pink-red
4	0	Black

Fig 2. Colour register defaults

To select one of these colours, the COLOUR statement is used. Thus COLOUR 0 will select orange as the current foreground colour. Colour settings apply only to graphics modes.

The default colour registers can be reset using the SETCOLOUR statement. SETCOLOUR takes the format: SETCOLOUR colour register to be reset, hue colour number and intensity. The intensity is an even number between 0 and 14: the higher the number the brighter the colour, so SETCOLOUR 1,4,5 sets colour register 1 to a moderately bright (5) and pink (4) from its default of bright green (1). Very bright colours (12 and 14) appear almost pure white.

All characters on the Atari are printed in upper case by default. The statement POKE 756,226 switches to lower case; POKE 756,224 goes back to upper case.

Once the business of selecting graphics modes and colours has been sorted out, there are then seven graphics statements supported: DRAWTO, PLOT, LOCATE, POSITION, PUT, GET, X10.

DRAWTO x,y draws a line in the current foreground colour from the last point visited to the specified coordinate. (0,0) is at the top left of the screen.

PLOT x,y plots a single point in the current foreground colour at the specified coordinate.

LOCATE x,y,var is similar to the Microsoft Basic POINT statement: it returns the colour of the specified coordinate. In the text modes, it returns a number between 0 and 255 indicating the ASCII code of the character plotted there, and places it into the specified variable.

POSITION x,y positions the graphics cursor at the specified coordinate without affecting the display.

PUT #6,z places the CHR$ of the specified ASCII code (z) at the current graphics cursor position in modes 0 through 2. In the graphics modes (3-8),

May 84 5(5): 75-76
1 of 2.

it plots the colour register (z) at the current graphics cursor position.

GET #6, var returns the ASCII code (text modes) or colour register (graphics modes) of the specified coordinate, placing it into the specified variable.

Note that PUT# and GET# statements only refer to the screen where the specified stream is 6; other values refer to other devices.

X10 18,#6,0,0,"S:" is a specialised use of the X10 (general-purpose input/output statement). It is used to paint a predefined area with a predefined colour. To use the statement, the bottom right-hand corner of the area to be filled is PLOTted. Next, a DRAWTO the top right-hand corner is executed. Thirdly, the cursor is POSITIONed at the bottom left-hand corner, and address 765 is POKEd with the colour register of the desired colour. Finally, the X10 18,#6,0,0,"S:"§ is executed.

How is the text colour set in modes 0 through 2? Why this can't be something as straightforward as COLOUR x, I don't know. The method of achieving this modest task is very strange and absurdly complex, involving referral to two separate tables and not a little arithmetic. It involves setting SETCOLOUR to some unlikely-looking value, but my advice is just choose a text colour which looks pretty on the machine you're converting to.

Sound

Sound is handled with a statement called (wait for it) SOUND. SOUND has four parameters which, for want of anything more original, we'll call a,b,c and d.

Parameter a specifies the voice (channel) in the range 0-3; b is the pitch (0-255); c the distortion (0-14, 10 giving a pure note, any other channel being filtered through one of the 13 fixed envelopes); d is the volume, from 1 (barely audible) to 15 (audible).

Middle C is pitch 121, each semi-tone is either 6 or 7 steps.

APC May 84 5(5), 75-76

2 of 2.

Sinclair

Surya continues his look at graphics and sound on each of the machines included on the APC Basic Converter Chart (see November issue). This month, the Sinclair ZX81 and Spectrum.

Sinclair ZX81

The ZX81 produces black graphics on a white background. The graphics resolution is 64 x 44, the origin (0,0) being the bottom left-hand corner of the screen. Two graphics statements are supported: PLOT and UNPLOT.

PLOT x,y switches on (ie lights up) coordinate (x,y). UNPLOT x,y switches off the specified coordinate. Drawing lines is achieved using FOR-NEXT loops, thus:

```
100  FOR X=0 TO 63
110    PLOT X,0
120    PLOT X,43
130  NEXT X
140  FOR Y=0 TO 43
150    PLOT 0,Y
160    PLOT 63,Y
170  NEXT Y
```

would draw a box around the edge of the screen.

The ZX81 also supports a PRINT AT function (PRINT @, on most machines). The PRINT AT screen comprises a 32 x 22 grid with the origin — just to confuse — as the top left-hand corner. To print 'HELLO' in the middle of the screen, you would enter PRINT AT 11,13;"HELLO".

The ZX81 reserves the bottom two lines of the screen for input prompts, error messages, and so on; these lines are not accessible when programming in Basic, and so are not assigned coordinates.

Sound is not supported.

Sinclair Spectrum

Graphics:

The Spectrum is available with either 16k or 48k RAM, but there are no other differences between the two models.

The Spectrum supports eight foreground and eight background colours. The single graphics resolution is 256 x 176, but there are limitations when using colour. The graphics statements are as follows:

PLOT — PLOT x,y lights coordinate (x,y) in the current foreground colour.

DRAW — DRAW x,y [,a] draws a line from the last coordinate visited (using PLOT, DRAW or CIRCLE) to a point x coordinates to the right and y coordinates up. The values of x and y may be either positive or negative, and may be expressions and/or variables as well as literal numbers.

The value 'a' is optional, and instructs the computer to draw a curved, rather than straight, line. This value specifies the number of radians the line must turn through as it draws; if a is positive, the line will curve to the right, if negative to the left. As a rough guide when reading listings, if a = 2*pi, a complete circle will be drawn, a=pi then a semi-circle is drawn, etc.

CIRCLE — The Spectrum has a built-in function to draw circles. This is considerably faster than using DRAW, but less accurate, which is why you find the DRAW method used in some listings. To draw a circle, you state CIRCLE x,y,r where (x,y) are the coordinates of the centre of the circle and r is the radius.

CIRCLE also appears to contain a slight bug. After drawing the circle, the statement leaves the graphics cursor in — as the manual puts it — 'a rather indeterminate place'. For this reason, you will normally find a PLOT statement immediately following a CIRCLE. This is simply to put the graphics cursor in a known position rather than being a part of the display routine as such.

PAPER & INK — A wonderfully sensible idea; PAPER being used to set the backgound colour and INK the foreground colour.The format is the same in both cases, PAPER (or INK) z where z is the colour as defined below:

0 —	black
1 —	blue
2 —	red
3 —	magenta
4 —	green
5 —	cyan
6 —	yellow
7 —	white

BRIGHT — Sets the brightness of the colours. BRIGHT 0 being normal, BRIGHT 1 being extra bright.

FLASH — Flashes foreground colour. 1 = on, 0 = off.

INVERSE — Reverses INK and PAPER. 1 = on, 0 = off.

OVER — Allows overprinting. Normally, if you print (say) a letter 'X' and then an addition sign at the same position, the second character will obliterate the first. OVER allows the old character to remain visible, so that the above example would produce something like an asterisk (*). 1 = on, 0 = off. The only way to recreate this on other machines is to work out what the combined character would look like and see if your character set supports something similar. If your machine has the facility to support user-definable characters, then this is, of course, another way around the problem.

BORDER — The Spectrum has a border around the screen which the user cannot access for screen displays using Basic, but its colour can be reset using BORDER z, where z is as for PAPER and INK. BORDER has no equivalent on most machines and can be safely ignored when converting from a Spectrum listing.

Note that colour 8 can be used with PAPER, INK, BRIGHT and FLASH to set the respective attributes to 'transparent'. Colour 9 can be used with PAPER and INK to select automatically maximum contrast, thus each is set to white if the other is a dark colour and black if the other is a light colour. This would have to be done 'manually' on most machines.

When describing the resolution of the graphics screen, I mentioned a limitation when using colour. Plotting a particular attribute (colour, inverse, flashing, and so on) affects the whole of the character position, rather than just the pixel in question. Thus, you cannot have a steady blue line right next to a flashing green one, though you can have two lines sporting identical attributes running alongside each other.

The final graphics-related statement supported on the Spectrum is SCREEN$. This is a very useful feature which allows you to save the contents of the screen memory on tape. This can subsequently be loaded from tape in order to recreate the display. The format is SAVE "filename" SCREEN$ to save, and LOAD "filename" SCREEN$ to load. This is most commonly used to load title screens for display while the main program is loaded.

Sound:

Sound on the Spectrum is controlled using the BEEP statement, the onomatopeiac word BEEP being a pretty accurate description of the sound quality. The format is SOUND duration, pitch.

Duration is in seconds and pitch is in semitones: 0 is middle C, negative numbers are lower, positive numbers higher. Each octave, of course, spans 12 semitones.

Jun 84 5(6) p 67.

PROGRAM CONVERSION

BBC

This month Surya turns his attention to the BBC in his continuing series on graphics and sound on each of the machines included in the APC Basic Converter Chart (see November issue). Find out how to convert BBC listings to work on your micro.

The complexity of the BBC's graphics often make its listings all but incomprehensible to owners of other machines. But there are a lot of well-written BBC listings around which the aforementioned owners would no doubt like to get up and running on their own machines. For this reason, I think it worthwhile to go into the subject in a fair amount of detail.

The BBC comes in one of two models: the 'A' and 'B'. The only difference between the two as far as graphics is concerned is that the model B offers eight screen resolutions, or 'modes', while the A offers only four.

The BBC has very powerful graphics-handling capabilities. This is useful if you own one, but makes life difficult for anyone trying to convert BBC graphics routines. Let's start with the business of modes. The model B can support eight different screen resolutions, while the model A supports modes 4, 5, 6 and 7 only. A brief summary of the modes follows:

0 — 80x32 text, 640x256 graphics, 2 colours
1 — 40x32 text, 320x256 graphics, 4 colours
2 — 20x32 text, 160x256 graphics, 16 colours
3 — 80x25 text, 2 colours, text only
4 — 40x32 text, 320x256 graphics, 2 colours
5 — 20x32 text, 160x256 graphics, 4 colours
6 — 40x25 text, 2 colours, text only
7 — 40x25 text, teletext mode (see later)

Mode x, where x is in the range 0 to 7, clears the screen and places you into the appropriate mode. This can be done as either a command or statement.

Once in a given mode, the graphics statements are as follows:

CLG —clears the graphics screen
CLS —clears the text screen
MOVE x,y —move the graphics cursor to point x,y
DRAW x,y —draw a line from the current cursor position to point x,y in the current foreground colour
COLOUR x —set the colour to be used for all subsequent printing of text, where x is an integer in range 0 to 15 to set foreground colour, 128 to 143 to set background colour. Note that the colour values are dependent upon the current mode: colour 2, for example, is yellow in a four-colour mode but green in mode 2 (the 16-colour mode). For an explanation of the colour codes, see later.
GCOL w,x —sets the colour to be used for all subsequent graphics operations, where x is the colour and w is the logical operation defined as:

0 — use the specified colour
1 — OR the specified colour with any colour already present
2 — AND the specified colour with any colour already present
3 — XOR (eXclusive OR) the specified colour with that already present
4 — invert (that is, change to the logical opposite) the colour already present

Note that x is as for COLOUR.

PLOT —more powerful version of draw: see later for further details

To set the text or graphics colour, numbered codes are used. These codes, as has been mentioned, are dependent upon the current mode. These codes can be reset (see VDU later — virtually everything you say about BBC graphics needs to be qualified in some way), but default values are:

Two-colour modes (0,3,4 and 6):

Black	—0 foreground, 128 background
White	—1 foreground, 129 background

Four-colour modes (1 and 5):

Black	—0 foreground, 128 background
Red	—1 foreground, 129 background
Yellow	—2 foreground, 130 background
White	—3 foreground, 131 background

Sixteen-colour mode (2):

Black	—0 foreground, 128 background
Red	—1 foreground, 129 background
Green	—2 foreground, 130 background
Yellow	—3 foreground, 131 background
Blue	—4 foreground, 132 background
Magenta	—5 foreground, 133 background
Cyan	—6 foreground, 134 background
White	—7 foreground, 135 background

Flashing colours:

Black/White	—8 foreground, 136 background
Red/cyan	—9 foreground, 137 background
Green/magenta	—10 foreground, 138 background
Yellow/blue	—11 foreground, 139 background
Blue/yellow	—12 foreground, 140 background
Magenta/green	—13 foreground, 141 background
Cyan/red	—14 foreground, 142 background
White/black	—15 foreground, 143 background

The last four colours incidentally, are not a typesetting error but merely one of the BBC's little idiosyncrasies.

To recap, first of all a mode is selected. This determines the resolution and the

1 of 2.

Jul 84 5(7) 129 -130

number of colours available. Then the screen may be cleared (using CLG and CLS), and the text colour (COLOUR x) and graphics colour (GCOLx) set. The graphics statements available are MOVE, DRAW and PLOT.

PLOT:

Whichever mode has been selected, the screen is addressed as a virtual screen 1280 x 1024 pixels. The origin (0,0) is at the bottom left-hand corner of the screen though this — like most things on the BBC — can be repositioned if desired. As desribed earlier, DRAW x,y draws a line in the current foreground colour to the specified coordinates. MOVE x,y moves to the specified coordinates without drawing (OK — for the purists — it draws a line in the current background colour (s)). PLOT is a more sophisticated form of DRAW and uses three parameters which we'll call k, x and y since the manual does.

Parameters x and y are straightforward, these being the coordinates used. The parameter k determines the manner in which the line is plotted as follows:

0 — move (ie, draw in background colour (s)) relative to present position
1 — draw (in foreground colour) relative to present position
2 — as 1, above, but in logical inverse colour
3 — as 1, above, but in background colour. This differs from 0 in that the background colour will overwrite any foreground colour present
4 — move to position (x,y)
5 — draw line to position (x,y) in current foreground colour
6 — as 5, but in logical inverse colour
7 — as 6, but in current background colour

Note that 0-3 plot x points in the x-axis and y points in the y-axis; that is, the plot is relative. 4-7 move to the screen coordinate (x,y); that is, the plot is absolute.

Higher values of k may be used to achieve other effects. The ones which are currently implemented are:

8-15 — as 0-7 but with the last point in the line omitted
16-23 — as 0-7 but using a dotted line
24-31 — as 0-7 but using a dotted line and with the last point in the line omitted
64-71 — as 0-7 but plotting only the last point of the line
80-87 — as 0-7 but use the last two points visited to plot and fill a solid triangle

You can see from the above that PLOT 4 is the same as MOVE and PLOT 5 is the same as DRAW.

There are also 33 'VDU codes', a number of which are related to graphics. These appear in listings as VDUx, where the most commonly used values of x are:

5 — join text and graphics cursors to enable text and graphics to be printed at the present graphics cursor position. This is disabled using VDU 4
19 — a very common VDU code used to redefine logical colours. For example, colour 1 is normally white in two-colour modes, but the programmer may wish to change it to a different colour. Thus VDU 19 allows access to colours not normally available in a given mode. The statement takes the form VDU 19, logical colour code, new colour code, 0,0,0 OR VDU 19, logical colour code, new colour code;0;. Thus in mode 0, VDU 19,1,3;0; would redefine white to appear as yellow. VDU 20 resets all colour codes to their default values.
23 — define a user-defined character. It uses the same binary-based system as most other machines, the form being VDU 23, ASCII code of the character to be defined, followed by the eight codes separated by commas.
24 — define a graphics window, that is an area of the screen outside of which no graphics may appear. The form taken is VDU 24,lower x coordinate;lower y coordinate;upper x coordinate;upper y coordinate;. Thus VDU 24,100;200;300;400; would define a graphics window with coordinate (100,200) as the bottom left-hand corner and (300,400) as the top right-hand corner. This is reset by VDU 26.
28 — define a text window. This works as for VDU 24, only commas are used instead of semi-colons and no trailing punctuation mark is required. The text screen is 39x31 characters by default. VDU 26 resets default.

And that covers the graphics handling. Now for sound.

Sound

The BBC has two sound statements, SOUND and ENVELOPE. The SOUND statement is relatively straightforward, ENVELOPE is so specific to the BBC that it would be of little use to spend the not inconsiderable amount of time necessary to explain it. Even if you could work out roughly what sort of sound was being created, you would have no way of effectively simulating it on another machine. What ENVELOPE does is to define the shape of the sound generated by the SOUND statement, so you may not be able to recreate the sound faithfully.

The format is SOUND channel, volume, pitch, duration where:

* Channel is in the range 0-3, channel 0 producing 'white noise' and used to create special effects.
* Volume is in the range 0 to −15 with 0 silent (useful) and −15 the loudest.
* Pitch ranges from 0 to 255, covering some five-and-a-bit octaves.
* Duration is in the range −1 to 254. −1 means 'continue until stopped' (either by pressing escape or by sending another note to the same channel), positive values are in twentieths of a second.

Sending two or more notes to the same channel at the same time produces a chord. Where channel 0 is used, the type of white noise produced depends upon pitch, the BBC manual summarising the effects as follows:

0 — high-frequency periodic noise
1 — medium-frequency periodic noise
2 — low-frequency periodic noise
3 — periodic noise, frequency determined by pitch setting of channel 1
4 — high-frequency white noise
5 — medium-frequency white noise
6 — low-frequency white noise
7 — white noise, frequency determined by pitch setting of channel 1

And that's the BBC micro! You do need to remember that without the equivalent of the ENVELOPE statement, you will not be able to achieve the kind of complex sound effects used in some BBC programs. Sound effects are generally the frills rather than the meat of a program, and while good sound effects can very much improve a program, they can usually be simplified without losing the effectiveness of a program.

END

Jul 84 5(7) 129-130 2 of 2.

NOTES AND ERRATA 1984

✻ **More functions for the VZ200, March '84:** There is an error in the second column, just above the listing of the short BASIC program. It should be . . . (Can be done directly by POKE 30945,175.)

ETI April 1985 — 117

? POKE 31003, 175
or
193 ?

More functions for the VZ200

This article details how you can simply add automatic line numbering and TRON and TROFF trace functions to the Dick Smith VZ200 colour computer.

Steve Olney

HERE is a simple way to add *automatic line numbering* and *trace* functions to VZ200 BASIC. Automatic line numbering should be self-explanatory. However, the trace function may need some explanation. When attempting to debug a BASIC program, it is sometimes useful to see exactly what sequence of instructions the computer is interpreting. This is the function of the trace command. It prints out on the video the sequence of line numbers the computer (the interpreter) is stepping through when executing a program. This allows you to make sure the program is doing what you intended it to do. (Especially useful in the case of conditional GOTO's or GOSUB's).

As adding the trace functions (TRON and TROFF) is the simplest task, I will deal with that first.

Before running your program, type in POKE 31003,175 from the immediate mode (no line numbers *That's it!* This is equivalent to typing in 'TRON'. Now when you run your program, each time a new line is selected to be interpreted (or the same line number repeated) it will be printed on the video. To disable this function just type POKE 31003,0 from the command level. This simulates using the 'TROFF' command.

A drawback with this method is that you might only want to debug a small section of the program and so have to contend with sorting out that small section from the rest of the displayed line numbers. This can be simply overcome by adding POKE 31003,175 into your program with a line number which places it in the program just before where you want to start the trace. Then add POKE 31003,0 with a line number which places it where you want the trace to stop.

Auto

Now to deal with the slightly more complex 'AUTO' function. This function, when enabled, saves you the trouble of typing sequential line numbers when entering a program. This very useful function will automatically display the next line number when you hit 'RETURN' at the end of a line of program.

To do this you need to supply the starting line number and the increment between lines. Next you need to set a flag which tells the BASIC that the 'AUTO' function is enabled. (This must be done last or you will go into the 'AUTO' mode before you have supplied the starting line and increment.)

The starting line number must be POKEd into locations 30946 and 30947, and the line increment POKEd into locations 30948 and 30949. These have to be in two-byte form with the least significant bit (LSB) going into the first location of each pair, and the most significant bit (MSB) going into the second.

For the line increment this is no problem as long as you keep the increment below 255. (Most increments would normally be less than 100). Just POKE 30948, 'increment' and then POKE 30949,0 where 'increment' is less than, or equal to 255. (I usually use 10 or 20 as the increment.)

Of course, the line number would most likely be above 255, so you must convert your starting line number into two bytes where:

LINE NO. = (MSB * 256) + LSB
and where we:
POKE 30946, LSB and POKE 30947, MSB

Example: For a starting line number of 2000
  MSB = INT(2000/256) = 7
  LSB = LINE NO.—(MSB*256) = 208
So we must:
  POKE 30946, 208
  POKE 30947, 7

For those not content with trouble of calculating this every time the 'AUTO' mode is entered, I have written a short program to do this as well as to enable the 'AUTO' function itself. (Can be done directly by ✻POKE 30934,175.) Use line numbers which will put it well out of the way of any main program you are entering.

```
 0 CLS
10 INPUT"STARTING LINE NO. ";S
20 INPUT"INCREMENT ";I
30 MS=INT(S/256): LS=S-MS*256
40 POKE 30946,LS: POKE 30947,MS
50 MI=INT(I/256): LI=I-MI*256
60 POKE 30948,LI: POKE 30949,MI
70 POKE 30945,175
80 END
```

For convenience, type this small program in starting from line number 0. This will enable quick access by just typing 'RUN' and then 'RETURN'. However to run your program, you will now need to type, 'RUN xxxx;, where 'xxxx' is the first line number of your program.

To exit from the 'AUTO' mode, type 'CTRL' and 'BREAK' simultaneously exactly the same way you exit or interrupt a BASIC program. Incidentally, BASIC will automatically exit from the 'AUTO' mode when the new line number would have been greater than 65529. (The maximum line number allowed in this BASIC.)

(2¹⁶ − 7)

A useful feature of this 'AUTO' function is that, if you specify line numbers which include previously entered lines, then not only is the line number displayed but also the statements previously entered.

The cursor is conveniently positioned at the end of the line ready for any additions to that line. This can be used as a convenient editing feature. For example, let us suppose you have entered your program and now wish to go through and make corrections. Enter the first line number of the program to be corrected and the appropriate line increment for that program. You can now single step through your listing and make corrections as you wish! Unfortunately, there is no simple way of decrementing the line number. (other than manually POKEing in location 30946).

Why So Simple?

How was I able to add these two functions so easily? Well, on close scrutiny of the VZ200 BASIC in ROM, I discovered that it was fundamentally similar to Level II TRS-80 BASIC. By finding the equivalent control areas in RAM for the VZ200 BASIC, and by experimentation, I was able to get the functions working.

Apparently, the machine code for the execution of the 'AUTO', 'TRON' and 'TROFF' functions is still present in the VZ200 BASIC ROM, but the interpreter has been altered so as not to recognise the commands as valid in an input text string.

Why the machine code would be present in the BASIC ROM but not enabled is a bit strange. Perhaps some functions were dropped in order to implement all functions provided on the multi-function keys.

A word of warning! Like all situations where you are patching software (especially when written by someone else), beware of yet-undiscovered gremlins. I take no responsibility for any havoc wreaked by same!

A more elegant and flexible approach would be to intercept the text interpreter and make it recognise the 'AUTO' and trace commands from the immediate command level, and perhaps add a line renumbering command. But that's another story! ●

Some more routines

By Philip Middlemiss

In the Dick Smith VZ200 there are a number of new routines which can be used by the use of simple BASIC commands. These routines are:

1: Defint x (defines variables listed as integers).

2: Defdbl x (defines variables listed as double precision).

3: Auto: (auto line numbers).

4: Print Mem: (prints the memory available).

5: On x GOTO line1, line2, etc.

6: Delete (deletes a block of BASIC program).

All of these routines must be used with a line number, and under most circumstances should be typed before any other program lines are typed in.

If a program is already in the computer and you want to add one of the above routines then put the line right at the beginning of the program with a GOSUB or GOTO routine in the line where you want the routine to be used. (See example A).

When the routine is put into the computer you must use a line number lower than any existing line number already in the computer.

When you LIST your program you will see the line number only, with nothing after it, so editing this line is not possible. The reason that the line is blank is that in the VZ200 ROM there are no BASIC words for these routines.

Here are the instructions for each routine. Don't type that which is enclosed in ( ).

(DEFINT X) (X can be A,B,C, etc or A-L etc).

```
10 PRINT A,B
POKE 31469,153
```

Then type rest of program.

(DEFDBL X)

```
10 PRINT A,B,C
POKE 31469,155
```

Then rest of program.

When these variables are found in your program they will automatically be used as integers or double precision as programmed.)

(AUTO) (to generate AUTO line numbers 10-20-30-40, etc).

```
1 PRINT
POKE 31469,183
RUN
```

(To generate AUTO line numbers starting at, say, 500 with steps of 20).

```
1 PRINT 500,20
POKE 31469,183
RUN
```

(The first number is the start number, the second is the step between numbers).

When AUTO is finished with remove line 1.

(PRINT MEM)

```
10PRINT X
POKE 31470,200
RUN
```

(Also see example A.)

(ON X GOTO (OR GOSUB) 100,200,300)

```
10 POKE 31469,161
```

(For use see example B.)

(DELETE)

(After a program has been loaded and is working you sometimes need to remove a block of program that is no longer needed or needs to be replaced.)

```
1 PRINT 150-300
POKE 31469,182
RUN
```

(In this example lines 150 to 300 will be deleted.)

Example A

When the routine is required in the middle of a program use as this example.

```
2 PRINT X:RETURN
POKE 31470,200
1 GOTO 10
10 (rest of program)
```

When memory available is required in the program use: Line no GOSUB 2.

Example B

In the ON X GOTO routine, when X =1 the program will branch to the first line No., and if X =2 then the program will branch to the second line No., etc. Here is how it can be used.

```
70 PRINT X GOTO 100,200,300,400
POKE 31469,161
10INPUT "ENTER TWO NUMBERS",a,b,
20PRINT"ENTER 1 TO ADD"
30PRINT"      2 TO SUBTRACT"
40PRINT"      3 TO MULTIPLY"
50PRINT"      4 TO DIVIDE"
60INPUT X
80 (continue with rest of program)
```

Other words decide where this line is to be, give it the correct line number. But type it in first followed immediately by its POKE statement. You could also use it as example A. You could do it this way:

Type IN PROGRAM B, replace line 70 with GOTO 2 and then add:

```
2 PRINT X GOTO 100,200,300,400
POKE 31469,161
1 GOTO 10
```

If two or more of these routines are required type as below:

```
5 PRINT A,B,C
POKE 31469,153
4 PRINT X,Y,Z
POKE 31469,155
```

This will make A,B,C variables integers and X,Y,Z variables double precision. These lines can be typed in after the program is loaded as long as line numbers lower than five have not been used.

V-ZED — THREE NEW FUNCTIONS

This is a regular feature to assist VZ 200 users to come to understand more about their computers and to learn a few tricks which are not necessarily covered by the manuals. We welcome contributions from Readers who have discovered new features of the machine or interesting techniques which they would like to share with their fellow VZ-200 users.

The BASIC Interpreter in the VZ 200 was written by MICROSOFT, the company which developed the first BASIC Interpreter for a microcomputer way back in the mid 70's and which probably supplies over 80% of all BASIC Interpreters in use today. Not surprisingly, when a new computer such as the VZ comes along, MICROSOFT takes its standard BASIC Interpreter and modifies it to suit the new hardware and the particular features which the manufacturer would like included. From the user's point of view there are both advantages and disadvantages to this approach. The main disadvantage is that the resulting code can become very untidy with patches on patches right throughout the ROM. The outcome often being inefficient use of space and slower execution. On the positive side however, there are likely to be routines still left in from other Interpreters which are not intended to be available in the VZ but, with a little fiddling can be used. To the average computer user, the thrill of making your computer do something which the manufacturer never intended, is worth any of the disadvantages. The purpose of this article is to start you off with three hidden functions. Once you start experimenting in this area you will no doubt find others. Please write in and let us know about them so that we may all share in them.

The MICROSOFT BASIC interpreter as implemented in the Tandy TRS-80 Model 1 occupied 12 Kbytes of ROM. Although we do not know for sure, it is likely that this implementation started a new family of BASIC Interpreters of which the VZ's is a derivative. Certainly there seems to be no surplus code in the Tandy Interpreter although the Model 3 version shows evidence of having been extensively patched and hacked around. The Interpreter in the VZ has a number of additional features over and above those available in the Tandy. In particular, the support for higher screen resolution, colour and full screen editing obviously requires extra code. Even though this interpreter now occupies 16K of ROM it became necessary to leave out some of the features which had been in the TRS-80 version. In particular, the AUTO TRACE function and the free memory indicator have gone whilst there is no facility to turn off the sound, should you wish to do so. However, the essential routines to do all these things remain locked away in the ROM and can be accessed with a bit of judicious POKEing.

AUTO LINE NUMBERING

The Interpreter contains an AUTO line numbering routine which when activated, automatically prints the next line number on the screen to speed up the entry of BASIC programs. It is possible to specify the starting line number and the increment between line numbers. For example, you may wish to start entering lines commencing with line 100 with an increment of 10 so that the second line would be 110 the third 120 etc. The AUTO routine operates every time you press the RETURN key from the COMMAND mode. It looks at address 30945. If that address contains a zero then AUTO numbering is off and the computer behaves normally. However, if that value is 1, the AUTO routine looks at addresses 30946 and 30947 to find the value of the starting line number then at addresses 30948 and 30949 for the increment between line numbers. The next line number is then automatically displayed on the screen. The only part of the AUTO routines missing is the ability to recognise the AUTO command itself. However, if you POKE the appropriate values into the memory addresses above, you will be able to use this facility.

To set the starting line number, POKE the decimal equivalent of its Least Significant Byte (LSB) into address 30946 and the decimal equivalent of its Most Significant Byte (MSB) into 30947. Similarly, to set the line increment, POKE its LSB into 30948 and its MSB into 30949. It is likely that this is double Dutch to relatively new users of the VZ so we have illustrated the techniques with the program below. If you wish to know more about the subject of POKEing etc. you will find a good article in Volume 4, issue 4/5.

We suggest you enter this routine, make sure it works satisfactorily then CSAVE it under the name AUTO or similar. You can then load it in whenever you are doing program development. We have used high line numbers to keep it out of the way of your own programs. To start it operating, type RUN 60,000. Incidentally, you terminate AUTO line numbering by pressing the BREAK key.

TURNING OFF THE BEEPING KEYBOARD

Now that you have AUTO line numbering, you will probably want to sit up all night entering programs. Only trouble is, the beeping of the keys is likely to keep the rest of the family awake.

No problem:

POKE 30779, 0 disables the key beep whilst

POKE 30779, 1 turns it on again.

You may enter this straight from the keyboard or include it as a line in your program.

Incidentally, this memory address appears to carry out some other functions, depending on the bit that is set. We did a little experimenting and found that bit 0 turns on and off the beep as expected i.e. an even value POKEd into address 30779 turns off the beep whilst an odd number turns it on i.e. 0, 2, 4, 6, 8 etc. turn it off, 1, 3, 5, 7, 9 etc. turn it on. Bits 1 and 2 have no special effect but bit 3 clears the screen and positions the cursor at the bottom left hand corner. This bit also causes an audible click from somewhere inside the computer probably from the piezo electric speaker. Bit 4 changes the background colour from green to orange. As far as we could tell bits 5, 6 and 7 had no effect.

FREE SPACE

Probably the most useful POKE for a programmer would be a way of finding out how much string space is available or how much memory you have left to cram in those last few lines before being told by the machine that you are Out of Memory.

Try the following.

```
POKE 30862,212:
POKE 30863,39:
PRINT USR(X) 'FREE MEMORY
   OR
PRINT USR(X$) 'FREE STRING
SPACE
```

PROGRAM LISTING 1

```
60000 REM SET STARTING LINE NO                  FOR THE AUTO ROUTINE
60010 INPUT"STARTING LINE NUMBER";SL
60020 POKE 30946,(SL-256*INT(SL/256))
60030 POKE 30947,INT(SL/256)
60350 REM SET THE INCREMENT                     BETWEEN LINE NUMBERS
60360 INPUT"INCREMENT BETWEEN LINE NOS";IN
60070 POKE 30948,(IN-256*INT(IN/256))
60090 POKE 30949,INT(IN/256)
60100 REM SWITCH ON THE AUTO                   LINE NUMBERING ROUTINE
60110 POKE30945,1
```

Micro-80 Jul 84 4(7) : p 3-4.

V-ZED

Last Issue we explained how to obtain three new functions from the VZ200, including a POKE which turns off the beeping keyboard. Reader Ken Hicks became concerned that this latter recommendation might actually cause some damage to the innards of the computer and possibly to the speaker itself, he writes:

I read with some interest your piece on the new functions for the V-ZED.

It was on the strength of your supporting this machine that I bought one for my young son. To date I have had no joy with the darn thing — it has twice been returned for service, and I have not yet received it or a replacement.

I purchased a copy of the Technical Reference Manual with the unit, so while waiting for the unit to turn up again, I have read the manual from cover to cover, which probably is not a bad idea, but which I almost certainly would not have done under normal circumstances. This Manual gives full circuit diagrams and reveals the very much simplified address decoding. There is also some very useful information on the System pointers, memory mapping, and particularly the details of graphics.

The addresses of a few routines in ROM are given, which will be familiar to ML programmers who use the old Microsoft ROM. For example, 28A7H and 01C9H are still message output and clear screen routines.

Evidently the writer of your article has not studied his TR Manual, as it gives details of the function of an output latch which effectively occupies all locations from 6800 to 6FFF inclusive. This is a write-only latch which services the cassette output, speaker, and video display controller. This latch is copied at 783B (30779), and its bit allocation is:

> *Bits 0 & 5 drive the speaker. They are normally toggled alternatively in a push-pull fashion to produce a tone. Holding one bit at '0' would therefore hold the speaker diaphragm 'pushed', while holding the other bit at '0' would keep it 'pulled', with an audible click as it went from one state to the other.*
>
> *Bits 1 & 2 generate the cassette output signal. Fiddling with these could corrupt a tape if the cassette were in the RECORD position!*
>
> *Bit 3 controls the VDC display mode. An '0' here sets MODE (0), while a '1' causes the VDC to operate in MODE (1). This effect is via the video controller chip.*
>
> *Bit 4 controls the background colour. It it is '0' then the background will be green, while if it is '1' the background will be orange if in MODE (0) and buff if in MODE (1). Thus, its effect depends on bit 3.*

The BEEP routine is at 3450H. Calling this address will produce a BEEP, but some disassembly around this area would be necessary (or perhaps around the keyboard scanning area — from 2EF4H) to find out how to silence the BEEP. It is possible that the brute force method suggested by your correspondent could damage the speaker or a chip by passing a current continuously, which is apparently what happens when '0' is POKED into 30779. I don't want to disparage your correspondent, but this just could be one instance where it is possible to cause physical damage to a computer via the keyboard!

Thank you Ken. There are two minor errors in your analysis of the situation of which one is significant to this discussion. Firstly, to correct a point of fact, bit 5 of the output latch is always held high whilst bit 0 is toggled from high to low to produce sound from the speaker. Of far more significance than that, however, is the nature of the "Speaker" itself. It is a piezo electric device. i.e. it consists of a crystalline substance with two metallised plates, one connected to bit 5 the other to bit 0. When there is a voltage difference between these two plates, the crystal actually changes shape, thus displacing the air surrounding it causing a "Click" to be heard (if the differential voltage has been applied rapidly enough). The BEEP routine you mention at 3450H alternatively sets and resets bit 0 thus applying a continually varying voltage across the crystal causing it to change shape rapidly and emit an audible tone. During this process very little energy is disippated since the piezo electric device appears electrically like a capacitor being alternatively changed and discharged. This device will not be damaged by applying a constant potential across it which is within its operating range. Nor will any IC be called on to carry excessive currents. In short, the POKE's recommended will not cause any harm to the computer. Nevertheless, thank you for raising this interesting subject. We would welcome similar contributions from our other readers.

Micro-80 4(8) p 2.
1984

```
1 REM   *** MEMORY PEEK ***
2 REM       FOR  VZ 200
3 REM       BY R.CARSON
4 REM   ******************
5 CLS
6 PRINT"           MEMORY PEEK           ";
7 PRINT"                                 ";
8 PRINT"PRESS <<SPACE>> TO PRINT HEX NO.";'TO SLOW DOWN PRINTING
9 PRINT"               TO SLOW DOWN      ";
10 PRINT"                                 ";
11 PRINT"PRESS<<:>>  FOR  NEW ADDRESS";
12 PRINT"                                 ";
20 INPUT"MEMORY LOCATION DECIMAL=";X1
22 PRINT"ADDR HEX     DEC  Z80ADEC CHR ASC";
23 FORD=0TO499:NEXTD
24 GOTO20000
25 X=ABS(X1)+ABS(A1)
26 IFX>65535THENGOTO20100
30 A2=X/4096:B2=A2-INT(A2):C2=INT(A2-B2):Z=65
40 FORY=10TO15
50 IFC2=YTHENQ$=CHR$(Z):GOTO80
60 Z=Z+1:NEXT
80 D2=B2*4096:E2=D2/256:F2=E2-INT(E2):G=INT(E2-F2):Z=65
90 FORY=10TO15
100 IFG=YTHENR$=CHR$(Z):GOTO130
110 Z=Z+1:NEXT
130 H=F2*256:I=H/16:J=I-INT(I):K=INT(I-J):Z=65
140 FORY=10TO15
150 IFK=YTHENS$=CHR$(Z):GOTO180
160 Z=Z+1:NEXT
180 L=J*16:M=L-INT(L):P=INT(L-M):Z=65
190 FORY=10TO15
200 IFP=YTHENT$=CHR$(Z):GOTO230
210 Z=Z+1:NEXT
230 IFC2>9THEN240ELSE250
240 PRINTTAB(2)Q$;:GOTO260
250 PRINTC2;
260 IFG>9THEN270ELSE280
270 PRINTTAB(4)R$;:GOTO290
280 PRINTG;
290 IFK>9THEN300ELSE310
300 PRINTTAB(6)S$;:GOTO320
310 PRINTK;
320 IFP>9THEN330ELSE340
330 PRINTTAB(8)T$;:GOTO350
340 PRINTP;
350 GOTO5055
5030 FORA1=0TO65535
5032 X2=A1+X1
5035 IFX2>65535THENGOTO20100
5037 IFX2>32767THENX2=X2-65536
5040 B1=PEEK(X2)
5045 L$=INKEY$:IFL$=" "THEN25
5047 GOTO5055
5052 PRINT"ADDR HEX  DECIMAL Z80DEC CHR ASC";
5053 FORD=0TO499:NEXTD
5055 PRINTTAB(12)X1+A1;
5060 PRINTTAB(20)X2;

5070 PRINTTAB(26)CHR$(B1);
5080 PRINTTAB(28)B1
5085 K$=INKEY$:IFK$=":"THEN20
5100 NEXTA1
20000 IFX1<-32768THENGOTO20100
20020 GOTO5030
20100 PRINT"    THIS IS TOP OF MEMORY    ";
20110 PRINT"  TO CONTINUE PRESS<Y> OR <N>  "
20115 K$=INKEY$
20116 I$=INKEY$:IFI$=""THEN20116
20117 IFI$="Y"CLS:GOTO20
20118 IFI$="N"CLS:END
20120 I$=INKEY$:IFI$<>"Y"ANDI$<>"N"THEN20116
```

MEMORY PEEK VZED

by Ron Carson

If you are interested in finding out what your VZ200 stores in its memory enter this program and have a look.

The program will display on the screen the information you need to know to run it and asks for a start address in decimal.

After going to the start location it will print the DECIMAL address, Z80 address, CHR at that address and ASCII code.

The program runs very quickly so to slow it down press the SPACE key. Pressing the SPACE key slows down the program and also prints the HEX address of each location on the screen.

If you want to change the memory location while the program is running press the (:) colon key and you will be asked for a new start address.

Micro-80

4(8) 1984

P. 9, 15 & 16.

VZ-200 BUG

To the VZ-200 hackers among us this short series of program statements crashes the VZ-200 (Version 2.0).

```
10 N=1 : INPUTS : FOR
   P=1 TO S : N=N*
   P/(P+1) : ? N; : NEXT :
   RUN
```

INPUT 23 twice and the second time round the machine goes crazy.

W Tritscher

P.S. If you pay me for the above, keep it and send it to the person who provides the ROM-patch routine.

APC Apr. 85 V.6(4): 97

VZ bug

I hope you haven't completed a review of the Dick Smith VZ-300 because it has a bug in the firmware (the same as the VZ-200).

If one RUNs, (then INPUTs 29), the following series of statements, the computer will crash.

```
10 N = 1 : INPUTS : FOR
   A = 1 TO S : N = N +
   1/(1 + A) : ? N; :
   NEXT : RUN
```

I first became aware of this fault at the 4th *APC* Show held at Centrepoint in Sydney earlier this year and informed Dick Smith. However, when I repeated the test on a new VZ-300 the results were the same. Dick Smith is therefore selling the VZ-300 with bugs.

W Tritscher

V. 6(8): Aug. 85.

GROUP ONE

This month we would like to bring your attention to some bugs in the Microsoft Basic interpreter as included in the Model I. Users of the CoCo and VZ200 might like to try and see if these bugs are also present in their computers.

Firstly, there is a problem with BASIC's handling of the "raise to the power" function. Enter the following program into your computer and 'RUN' it:—

```
10 FOR X = 1 TO 15
20 PRINT 2↑X
30 NEXT
```

The resultant printout will be as follows:—

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192.01
16384
32768
```

Whilst the above problem probably won't occur all that often, it is a good idea to be aware of it. The same applies to the following bug.

RND(X) can return a value of X + 1 when X is a power of 2. In cases where RND(0) is just under the value of one, when multiplied by X, the product is rounded and this is where the problem occurs. For instance, A = RND(16) can return a value for A of 17. To get around this, use the following:—

```
10  A = RND(16)  :  IF  A > 16
THEN 10
```

The next bug can be found if you try and use the expression PRINT VAL ("%") in your program. Whenever you have a % sing in a string to be converted by VAL you will get a syntax error. This bug also appears in the Model III ROM. To avoid this error in Disk Basic use the following routine:—

```
1000 I = INSTR(X$,"%")
1010 IF I THEN X = VAL
(LEFT$(X$,I - 1)) ELSE
X = VAL(X$)
```

Non-disk users should use the following:—

```
1000 FOR I = 1 TO LEN(X$)
1010  IF  MID$(X$,I,1) = "%"
THEN 1040
1020 NEXT I
1030 I = LEN(X$) + 1
1040 X = VAL(LEFT$(X$,I - 1))
```

This final bug also appears in all versions of the 'Level II' ROM. Enter the following program and 'RUN' it:—

```
10 INPUT A#
20 A# = INT(A#)
30 PRINT A#
```

If you were to enter – 56320 in answer to the prompt, the computer would come back with a result of – 56576. To explain, when taking the INT function of a double-precision number which is evenly divisible by 256 and is less than – 32768 one extra bit is turned on when processing the number which is subsequently reduced by 256, 512 or some other power of 256. To avoid this add the following filter to your program:—

```
100 A# = SGN(A#)
*INT(ABS(A#))
```

The first bug was mentioned originally in '80-US'. The rest of these bugs were first mentioned in 'The Alternate Source'.

Micro-80 4(8) Aug. 84 p 3-4.

VZ-200 trace

In the July edition of *APC*, J Williams suggested a method for printing a moving message across the bottom of the Comodore 64 screen. I modified this for the VZ-200:

```
 5 CLEAR 1000
10 A$="PUT MESSAGE
   HERE":REM LET A$ BE
   MESSAGE
15 PRINT@480," "
20 PRINT LEFT$(A$,31);
25 PRINT CHR$(27);:REM
   MOVES CURSOR UP
30 FOR I=1 TO 40:
   NEXT:REM: DELAY
35 A$=MID$(A$,2)+
   LEFT$ (A$,1):GOTO 25
```

A friend also told me of a tracing function for the VZ-200:

POKE 31003,175 starts trace function and prints line numbers

POKE 31003;0 disables this function.

The only problem is with MODE(1), the screen returns to MODE(0) to print line numbers and you don't get to see what is happening in high-res graphics.

Jay Batterson

APC 5(8) Aug. 84
p. 94.

Trace function

Jay Batterson's report on the trace function for the VZ-200 is interesting — it is the same for TRS-80 and System 80 computers bu what readers might find interesting is the way it is written in ROM viz:

```
1DF7     3E
1DF8     AF
1DF9     32
1DFA     1B
1DFB     41
1DFC     C9
TRON calls 1D7 and reads     LD A, 175
                             LD (16667(, A
                             RET
TROFF calls 1DF8 and reads  XOR A
                            LD (16667), A
                            RET
```

AR Breffit

APC 5(11) Nov. 84.
p 125

VZ-200 correction

In the August issue of *APC*, Jay Batterson submitted a short program for printing a moving message across the screen with a VZ-200. I tried this program and it didn't work. I was a bit disappointed that you had published it without testing it first, so I left it alone for a while.

Recently I had occasion to use my computer for a message on the screen, so I dug out the August issue and played around with the program until I found what was wrong with it.

So here is the same program with modifications to make it function:

```
 5 CLS
10 A$="YOUR MESSAGE"
20 PRINT @ 480,
   LEFT$(A$,31);
30 PRINT CHR$(28);
40 FOR I=1 TO 60:NEXT
50 A$=MID$(A$,2)
  +LEFT$(A$,1): GOTO 20
```

I know this one works.

J Kelly

APC 5(11) Nov. 84.
p. 125.

VZ200 Input

Data + Pyramids.

If you are using programs with DATA lines, why not use the VZ200 capability by a subroutine that will the new data to create revised data lines, as follows:

```
100 DATA 56
110 INPUT A
120 READ B
130 C=A+B
140 PRINT C
150 PRINT "100 DATA";C
```

Now CSAVE and the next time the program is used (once you have moved the cursor up to the last printed line and entered) the new data will be in the program.

With a FOR/NEXT loop, the theory can be applied to extensive programs. For example, you can use it to update top scores in games programs, or to update a budget program.

Gordon Woolf.

From Paul Vowles comes this program to produce, amazing pictures of 3D pyramids on your VZ200. Without doubt, this is one of the best programs we've seen so far for the VZ200 Colour Computer!

```
10 REMARKABLE PYRAMIDS
15 REM BY PAUL VOWLES
20 CLS:INPUT "PYRAMID HEIGHT";H
22 INPUT "LENGTH OF BASE";B
25 D=B/2
30 IF B<1 OR B>63 OR H<0 OR H>60 THEN 20
40 CLS:MODE(1):COLOR 6,1:REM CYAN
50 DL=(63-B)+(B/2.5)
55 DU=60-H:DM=63-B
57 DX=60-INT(H/2.5)
60 Y1=DU:X1=DL:Y2=60:X2=63+D:GOSUB 1000
65 DX=60-INT(H/2.5)
70 Y1=60:X1=DM:GOSUB  1000
80 Y1=DX:Y2=DX:GOSUB 1000
90 FOR Z=Y1 TO 60: SET(X1,Z)
95 SET (X2,Z):NEXT Z
100 X2=DL:Y1=60:Y2=DU:GOSUB 1000
110 Y1=DX:GOSUB 1000
120 X1=63+D:GOSUB 1000
130 COLOR 7,1
140 DN=63+B/2:DK=(63+B/2)-(B/2.5)
150 X2=DK:X1=DN:GOSUB 1000
160 X1=63-B:GOSUB 1000
170 Y1=60:GOSUB 1000
180 X1=DN:GOSUB 1000
190 FOR Z=1 TO 5000:NEXT Z
200 INPUT "AGAIN";A$
210 IF LEFT$(A$,1)="Y" THEN 20
220 END
1000 S=1:IF X1>X2 AND Y1>Y2 THEN S=-1
1010 SET(X1,Y1):SET (X2,Y2)
1015 Y=Y1:N=1:IF Y1=Y2 THEN A1=0:GOTO 1030
1020 A1=(X2-X1)/(Y2-Y1):IF S=-1 THEN A1=-A1
1030 FOR X=X1 TO X2 STEP S
1035 IF X<0 THEN X=0
1040 IF Y<0 THEN Y=0
1050 SET(X,Y):N=N+1
1060 IF A1<>0 THEN Y=Y1+N/A1
1079 NEXT X:RETURN
```

Cutting the margin

By L. Clarke & A.R. Hill

These hints may help you shorten a line which is marginally too long to type into the 64 character input buffer (ie, exceeds two lines on the screen).

The word, "PRINT" may be entered as a question mark (?) saving four character spaces. The word, "REM" or ":REM", may be replaced by an apostrophe ('), saving either two or three character spaces.

The computer will convert the (?) to the token for "PRINT" when it is stored in the memory, so that when the line is listed, it will appear as "PRINT". If the line then exceeds 64 characters on the screen, it will "wrap around" onto the next line, but will still function normally. Of course, the on screen editor uses the input buffer, and any attempt to edit a line exceeding 64 characters will result in the loss of all text after the 64th character displayed on the screen!

The following functions must be POKEd into an existing line in a BASIC program.

Example 1:

If the first line of a program is used (eg, line number 1), then the first memory location past the line number is 31469. This does not change regardless of the number of digits in the line number because all line numbers are stored in memory as a two byte code.

Example 2:

If you want use any of the following functions in the middle of a program — just type up to the place where you wish to insert the function, place a dummy character in that position, and press [RETURN].

Immediately (with no line number) type in the following

PRINT PEEK(30969) + 256 * PEEK(30970) −4.

This will give you the memory location of the last character you typed into the last program line (in this case the dummy character). Memorise this number (write it down!) then finish typing in the BASIC line, continuing immediately after the dummy character.

When you have finished typing in the line, LIST it and check it is correct, because once you have POKEd the function code into the memory location in which your dummy character is stored, you will not be able to edit that line!

You may now POKE the function code into the memorised location which holds the dummy character. If the memory address should exceed 32767, it is first necessary to subtract 65536 to reduce it to an integer for the POKE command to work.

It is assumed you have made no changes (insertions or deletions) to the program before the dummy character, because these would have changed its memory location.

DELETE
TRON
TROFF
MERGE
RENUM
DEF STR
ON GOTO
ON GOSUB
POS

Function	No	How to Use	Description
RANDOM	134	1# POKE31469,134	Makes RND( ) statement more random.
DEFINT	153	1#A,B POKE31469,153	Defines all variable starting with "A" or "8" as being integers.
DEFSNG	154	1#C,D POKE31469,154	Defines all variables starting with "C" or "D" as being single precision (6/7 digit floating).
DEFDBL	155	1#E,F POKE31469,155	Defines all variables starting with "E" or "F" as being double precision (16/17 digit floating).
ON	161	1# POKE31469,161	Used with ON GOTO, ON GOSUB or ON ERROR (see below).
ERROR	158	1#* POKE31469,161 POKE31470,158	Used as "ON ERROR GOTO line no".
RESUME	159	1# 1#100 1#100 NEXT POKE31469,159	After error, return to error point. After error, GOTO 100. After error, return to the line after the one producing the error.
DELETE	182	1#150-300 POKE31469,182	Deletes lines 150 to 300 inclusive. Both lines 150 & 300 must exist.
AUTO	183	1# POKE31479, 183:RUN	Automatically prints line numbers starting at 10, increment of 10.
AUTO	183	1#500, 20 POKE31469, 183:RUN POKE30945,175	Automatically prints line numbers starting at 500, increment of 20. AUTO will print any existing lines found. If the AUTO function was halted with[BREAK], it will now continue from that point.
VARPTR	192	1#(X) POKE31469,192	Used to locate the memory address of a variable.
STRING$	196	1PRINT#(12,45) POKE31470,196	Will print 12 asterisks "*" (maximum length of string = 256 characters).
MEM	200	1PRINT# POKE31470,200	Tells the amount of unused memory left.
FRE	218	1PRINT#(A) POKE31470,218	Tells the number of unused bytes left in memory.
FRE	218	1PRINT#(A$) POKE31470,218	Tells the number of unused bytes left in the reserved string space.
CINT	239	1#Z POKE31469,239	Removes all digits after the decimal point.
CSNG	240	1#Z POKE31469,240	Converts numeric variable from double to single precision.
CDBL	241	1#Z POKE31469,241	Converts numeric variable from single to double precision.
FIX	242	1A=#(N) POKE31471,242	Removes all digits to the right of the decimal point. Doesn't round down negative numbers.
ERL	194	1PRINT# POKE31470,194	Returns the line number from which program branched to error routine.
ERR	195	1PRINT# POKE31470,195	Returns a value related to the type of error which last occurred.

These functions may be performed either with or without a line number.

For TRON (Trace ON) just POKE 31003,175

For TROFF (Trace OFF) just POKE 31003,0

The audible "beep" produced when a key is pressed can be controlled.

For BEEP ON just POKE 30779,32

For BEEP OFF just POKE 30779,0

EXTENDING VZ 200 BASIC

Following on from a previous article ("More functions for the VZ200" — ETI March 1984) this article outlines a method of adding commands to the standard VZ200 BASIC.

Steve Olney

THE PREVIOUS article showed how to unlock several 'hidden' functions contained in the VZ200 BASIC ROM by entering the commands indirectly via a BASIC program itself. This approach meant that it was necessary to run the BASIC program each time the function was needed. This is very inconvenient and, as was hinted at in the previous article, a more elegant (and more convenient) approach would be to have the added functions accessed as if they were part of the original command set.

This article gives a method by which this can be done and gives a practical example by making the AUTO command part of the legal VZ200 BASIC command set.

The machine code necessary to achieve this is quite short because, as indicated in the previous article, the code which does the bulk of the work is already resident in the VZ200 BASIC ROM. It is only necessary to get the BASIC interpreter to recognise the auto line-numbering command (AUTO X, Y) as legal and then jump to the relevant code in ROM.

The method outlined here only applies to adding commands to the 'immediate execution mode'. (i.e: typing in commands without line numbers). It does not deal with commands that are to be used within programs.

How it works

Those who are only interested in the end result of adding the AUTO command to the legal commands can skip this section and go straight to the section dealing with entering the program. Those who are interested in how it works — read on!

The reason why it is possible to add commands to the standard VZ200 BASIC command set (thereby extending it) is that, in common with some other BASICs, at various points in the machine code in ROM, calls are made to locations in RAM. This makes it feasible to modify and/or extend the code at a later date. A common example is where a disk system is added later. An extended or enhanced BASIC can be implemented by downloading extra code off disk to the relevant called location. If all the code was executed in ROM then this could not be done.

In a non-disk system (such as the present VZ200) these called locations are usually initialised to '0C9H' (H means hex address of location), which is Z-80 machine code for Ret. So normally, when these RAM locations are jumped to via 'calls' from the BASIC ROM, execution returns immediately to the BASIC ROM via the 'Ret'.

Now, because the Ret's are in RAM, it is possible to change the Ret to a jump to extra code which will be executed before control is returned back to the BASIC ROM.

In the VZ200, all the calls from the BASIC ROM to RAM are to locations between 7952H and 79E2H. One of these exits will be used to add Auto X,Y to the legal command set.

The BASIC interpreter

Leaving the ROM exits for the moment, consider what happens when an 'immediate execution' command is entered. While the text is being typed in, the character codes for each key-press are being entered into a text buffer at around 79E8H. When Return is hit, the interpreter looks at what has been entered into the buffer. Scanning from left to right, it looks for 'reserved words' (words set aside for commands e.g: Print, List etc.). The BASIC ROM contains a list of these reserved words beginning at 1650H and ending at 1820H. This can be revealed by an ASCII dump of this block of memory (the first letter of each reserved word has 80H added to ASCII code which will result in garbage for that letter.)

The interpreter scans the text trying to find one or more of these reserved words. when one of these is found the reserved word text is replaced by a single byte or ▶

1 of 3.

'token' (80H to 0FBH). The token is the offset into the list where the reserved word is located and is used as an index into another table which contains the address of the machine code for that command.

If the text cannot be resolved into reserved words or text which belongs to the reserved words, then a Syntax error message is generated. The trick is to intercept control of the interpreter just after the reserved list has been scanned and add code to re-scan the text to see if it contains the new command Auto X,Y.

By good fortune (or good design), immediately after scanning has been done there is a call to RAM (to 79B2H). The Ret (0C9H) at 79B2H is changed to a jump to extra code which will re-scan the text buffer for Auto and if found, will replace the text with the relevant token.

Because only the reserved word list is disabled (by deleting Auto from it), once the Auto command text has been replaced by the correct token (0B7H), the following interpreter code will recognise the token and accept it as legal.

Entering the program

The machine code program is entered via a BASIC program (Listing 1) which POKEs the code into RAM from Data statements. The BASIC program locates the machine code to high memory after resetting the BASIC top-of-memory pointer to below where the code will be POKEd. By this, the machine code program is located out of the way of any BASIC program to be entered later. This action is independent of memory size.

The machine code listing is shown for reference only. All that is necessary is to enter the BASIC Program, save it on tape, and from then on just run it before you start entering your BASIC program. If all is well, control will be returned to the Ready level and, unless the machine code is overwritten by POKEs or the VZ200 is reset, the Auto command is now part of the immediate command set.

Auto command syntax

The form of the Auto command is 'AUTO X,Y' where X is the starting line number and Y is the increment beteen line numbers.

Entering AUTO X will give a starting line number of X and a default increment of 10, while entering AUTO, Y will give a default starting line number of 10 and an increment of Y. AUTO by itself will give both the line number and increment a default of 10.

To exit the Auto mode, hit 'CTRL BREAK'. Entering the Auto mode with line numbers of statements already entered can be a useful single step checking and editing feature (see previous article).

Adding other commands

This method can be used for 'unlocking' other commands 'hidden' in the VZ200 BASIC ROM. As shown in the previous article, the commands TRON and TROFF are also accessible. In the time since that article was submitted it has been found that the code for a delete command (DEL X-Y), with the same syntax as the LIST command, is also present in the VZ200 BASIC ROM.

The listing for a BASIC program that 'unlocks' the 'hidden' code for the AUTO, TRON, TROFF and DEL commands is available from the author. It is of the same form as the program described here.

What next?

The above four extra commands have proved to be very useful and have resulted in significant time-savings in writing BASIC code. Other useful commands would be REN (line re-numbering), MERGE (merging small sub-programs on tape into one program — difficult, because it appears that the VZ200 CLOAD always loads a BASIC program to the location in

2 of 3.

memory from which it was CSAVEd). DH and HD (allows decimal to hexa-decimal conversion, and vice-versa). These would be much more difficult to implement as there is no code present in the VZ200 BASIC ROM, so they will have to be written from scratch.

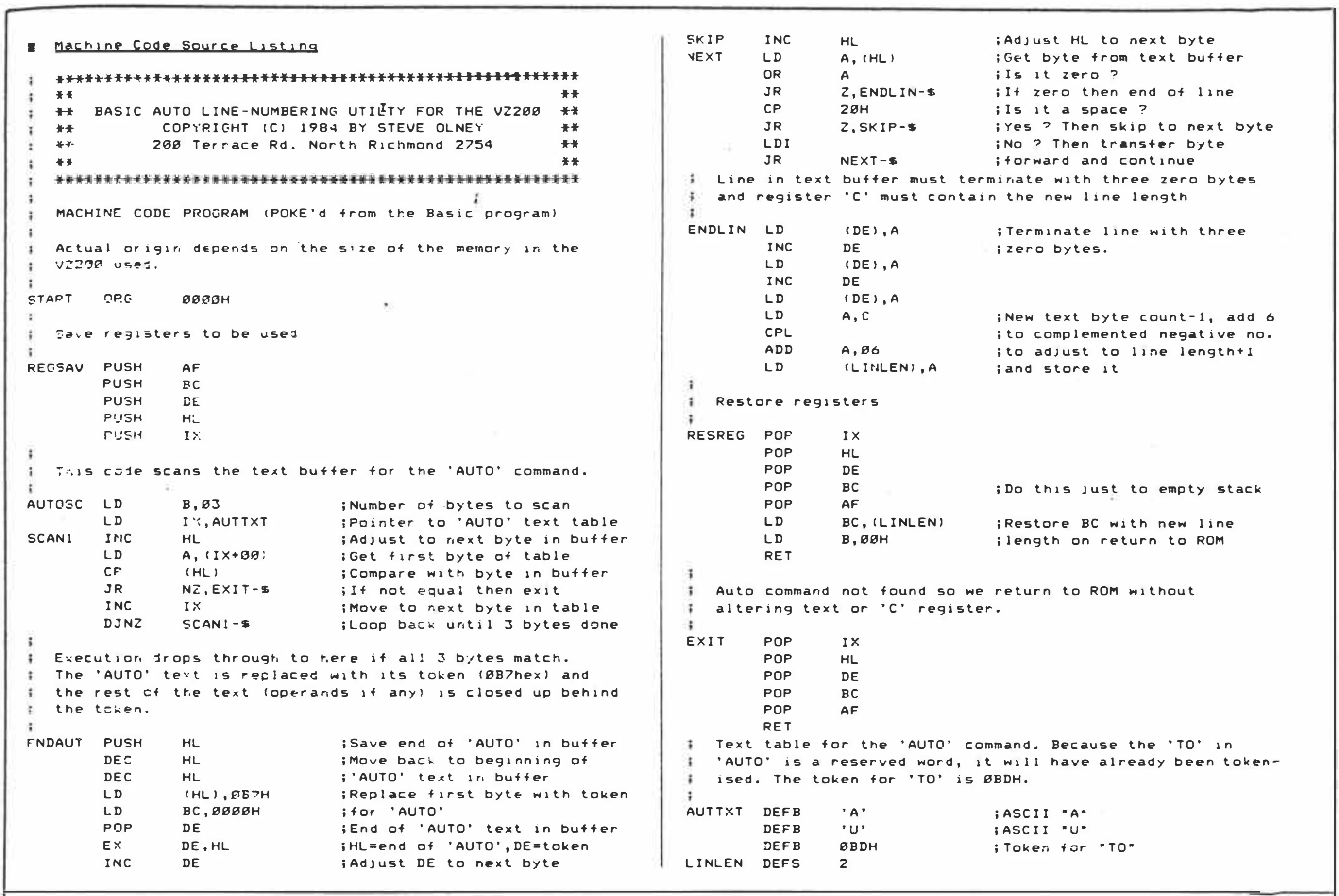
Cautions

Firstly, as this program uses code in the Version 2.0 BASIC ROM, users with other versions (if any) will have to check to see if the program works with their version.

Secondly, you may have already found that during normal program entry, occasionally the cursor will skip a line after you hit Return. This is of no real consequence — until now. Unfortunately the auto line-numbering code doesn't like this and responds by displaying the next line number as it should, but then positions the cursor at the beginning of the next line. Any BASIC statements or text entered on that line will be lost.

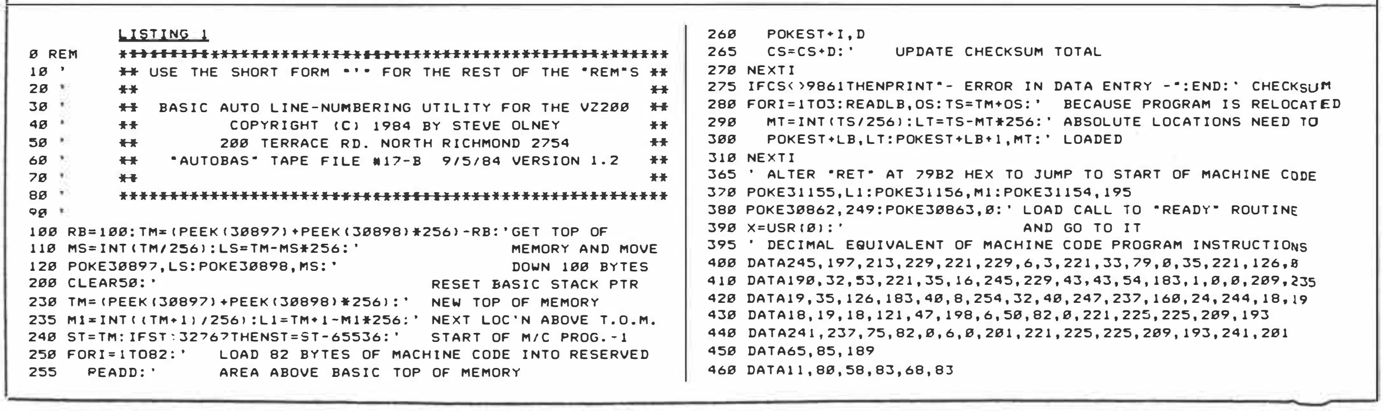
Each time Return is hit for a new line number, check to see that the cursor is on the same line as the new line number. If it isn't, hit Return again. This will skip to the next line number. Do this until the cursor is positioned on the same line as the new line number, then it is OK to enter statements. Unless you are fussy the missed line numbers should not be a problem. Of course, you can exit the auto mode (CTRL BREAK) and restart so as not to miss a line number.

A printed listing of a larger program to add the AUTO plus TRON, TROFF, DEL commands to the legal command set can be obtained for $5.00 from the author at: **200 Terrace Rd, North Richmond NSW 2754.** Remember YOUR address! (pref. SAE) ●



■ Machine Code Source Listing

```
;   *******************************************************
;   **                                                   **
;   **  BASIC AUTO LINE-NUMBERING UTILITY FOR THE VZ200  **
;   **         COPYRIGHT (C) 1984 BY STEVE OLNEY         **
;   **        200 Terrace Rd. North Richmond 2754        **
;   **                                                   **
;   *******************************************************
;
;   MACHINE CODE PROGRAM (POKE'd from the Basic program)
;
;   Actual origin depends on the size of the memory in the
;   VZ200 used.
;
START   ORG     0000H
;
;   Save registers to be used
;
REGSAV  PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
        PUSH    IX
;
;   This code scans the text buffer for the 'AUTO' command.
;
AUTOSC  LD      B,03                ;Number of bytes to scan
        LD      IX,AUTTXT           ;Pointer to 'AUTO' text table
SCAN1   INC     HL                  ;Adjust to next byte in buffer
        LD      A,(IX+00)           ;Get first byte of table
        CP      (HL)                ;Compare with byte in buffer
        JR      NZ,EXIT-$           ;If not equal then exit
        INC     IX                  ;Move to next byte in table
        DJNZ    SCAN1-$             ;Loop back until 3 bytes done
;
;   Execution drops through to here if all 3 bytes match.
;   The 'AUTO' text is replaced with its token (ØB7hex) and
;   the rest of the text (operands if any) is closed up behind
;   the token.
;
FNDAUT  PUSH    HL                  ;Save end of 'AUTO' in buffer
        DEC     HL                  ;Move back to beginning of
        DEC     HL                  ;'AUTO' text in buffer
        LD      (HL),ØB7H           ;Replace first byte with token
        LD      BC,ØØØØH            ;for 'AUTO'
        POP     DE                  ;End of 'AUTO' text in buffer
        EX      DE,HL               ;HL=end of 'AUTO',DE=token
        INC     DE                  ;Adjust DE to next byte
SKIP    INC     HL                  ;Adjust HL to next byte
NEXT    LD      A,(HL)              ;Get byte from text buffer
        OR      A                   ;Is it zero ?
        JR      Z,ENDLIN-$          ;If zero then end of line
        CP      2ØH                 ;Is it a space ?
        JR      Z,SKIP-$            ;Yes ? Then skip to next byte
        LDI                         ;No ? Then transfer byte
        JR      NEXT-$              ;forward and continue
;   Line in text buffer must terminate with three zero bytes
;   and register 'C' must contain the new line length
;
ENDLIN  LD      (DE),A              ;Terminate line with three
        INC     DE                  ;zero bytes.
        LD      (DE),A
        INC     DE
        LD      (DE),A
        LD      A,C                 ;New text byte count-1, add 6
        CPL                         ;to complemented negative no.
        ADD     A,Ø6                ;to adjust to line length+1
        LD      (LINLEN),A          ;and store it
;
;   Restore registers
;
RESREG  POP     IX
        POP     HL
        POP     DE
        POP     BC                  ;Do this just to empty stack
        POP     AF
        LD      BC,(LINLEN)         ;Restore BC with new line
        LD      B,ØØH               ;length on return to ROM
        RET
;
;   Auto command not found so we return to ROM without
;   altering text or 'C' register.
;
EXIT    POP     IX
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        RET
;   Text table for the 'AUTO' command. Because the 'TO' in
;   'AUTO' is a reserved word, it will have already been token-
;   ised. The token for 'TO' is ØBDH.
;
AUTTXT  DEFB    'A'                 ;ASCII "A"
        DEFB    'U'                 ;ASCII "U"
        DEFB    ØBDH                ;Token for "TO"
LINLEN  DEFS    2
```



LISTING 1

```
Ø REM     ********************************************************
1Ø '      ** USE THE SHORT FORM "'" FOR THE REST OF THE "REM"S **
2Ø '      **                                                   **
3Ø '      **  BASIC AUTO LINE-NUMBERING UTILITY FOR THE VZ2ØØ  **
4Ø '      **         COPYRIGHT (C) 1984 BY STEVE OLNEY         **
5Ø '      **        2ØØ TERRACE RD. NORTH RICHMOND 2754        **
6Ø '      **   "AUTOBAS" TAPE FILE #17-B  9/5/84 VERSION 1.2   **
7Ø '      **                                                   **
8Ø '      ********************************************************
9Ø '
1ØØ RB=1ØØ:TM=(PEEK(3Ø897)+PEEK(3Ø898)*256)-RB:'GET TOP OF
11Ø MS=INT(TM/256):LS=TM-MS*256:'                 MEMORY AND MOVE
12Ø POKE3Ø897,LS:POKE3Ø898,MS:'                   DOWN 1ØØ BYTES
2ØØ CLEAR5Ø:'                            RESET BASIC STACK PTR
23Ø TM=(PEEK(3Ø897)+PEEK(3Ø898)*256):'   NEW TOP OF MEMORY
235 M1=INT((TM+1)/256):L1=TM+1-M1*256:' NEXT LOC'N ABOVE T.O.M.
24Ø ST=TM:IFST>32767THENST=ST-65536:'   START OF M/C PROG.-1
25Ø FORI=1TO82:'   LOAD 82 BYTES OF MACHINE CODE INTO RESERVED
255   READD:'      AREA ABOVE BASIC TOP OF MEMORY
26Ø   POKEST+I,D
265   CS=CS+D:'    UPDATE CHECKSUM TOTAL
27Ø NEXTI
275 IFCS<>9861THENPRINT"- ERROR IN DATA ENTRY -":END:' CHECKSUM
28Ø FORI=1TO3:READLB,OS:TS=TM+OS:'  BECAUSE PROGRAM IS RELOCATED
29Ø   MT=INT(TS/256):LT=TS-MT*256:' ABSOLUTE LOCATIONS NEED TO
3ØØ   POKEST+LB,LT:POKEST+LB+1,MT:' LOADED
31Ø NEXTI
365 ' ALTER "RET" AT 79B2 HEX TO JUMP TO START OF MACHINE CODE
37Ø POKE31155,L1:POKE31156,M1:POKE31154,195
38Ø POKE3Ø862,249:POKE3Ø863,Ø:' LOAD CALL TO "READY" ROUTINE
39Ø X=USR(Ø):'                        AND GO TO IT
395 ' DECIMAL EQUIVALENT OF MACHINE CODE PROGRAM INSTRUCTIONS
4ØØ DATA245,197,213,229,221,229,6,3,221,33,79,Ø,35,221,126,Ø
41Ø DATA19Ø,32,53,221,35,16,245,229,43,43,54,183,1,Ø,Ø,2Ø9,235
42Ø DATA19,35,126,183,4Ø,8,254,32,4Ø,247,237,16Ø,24,244,18,19
43Ø DATA18,19,18,121,47,198,6,5Ø,82,Ø,221,225,225,2Ø9,193
44Ø DATA241,237,75,82,Ø,6,Ø,2Ø1,221,225,225,2Ø9,193,241,2Ø1
45Ø DATA65,85,189
46Ø DATA11,8Ø,58,83,68,83
```

3 of 3.

TRON/TROFF function for VZ-200

When debugging a Basic program, it is frequently useful to see exactly what sequence of instructions the computer is interpreting. This is the function of the TRON (Trace ON) command found in many versions of Basic.

This command is not, however, directly available to the VZ-200 user and must be executed by POKEing directly to the screen.

POKE 31003,173 enables the TRON command
POKE 31003,0 disables the command (enables TROFF)

The TRON function executes the program as in normal execution, but displays each line number within brackets as it is executed. This Trace is useful in following the program flow during debugging, especially in the case of conditional GOTOs or GOSUBs. Normal display data generated by PRINT or other commands will be interspersed with the Trace line numbers.

The POKE values can be entered directly from the command level and then RUNning the programs, or they can be incorporated within the body of the program (especially useful if only a section of the program requires debugging).

The use of the CTRL and BREAK keys can be used at any time to stop the display for scrutiny. Entering the CONTinue command will restart program execution.

I Thompson

A.P.C. 5(11) Nov. 84.
p. 125-6

MON-200

by Chris Stamboulidis

MON-200 is a machine code monitor program for both 8 and 24k VZ-200s, featuring relatively easy data entry, screen listing of memory, execution of routines and provision for dumping memory to a printer. Also included are utilities for decimal to hex conversion (and vice versa) as well as a block memory move facility. All input is in hexadecimal.

After CSAVEing and RUNning the program, you will have the following options available:

(E) Enter Data: data is entered eight bytes at a time in the format

'NNNN dd dd dd dd dd dd dd dd'

where NNNN is the location of the first byte to be entered, and dd represents a single byte. Hit RETURN after the last byte, and note that the spaces are essential for successful operation. Data entry is not accepted if you specify a ROM location (obviously), system RAM, program RAM or the location of the block move routine. After entering the first eight bytes, you may choose to repeat the procedure or, if entering data in sequential locations, simply hit RETURN when the input prompt appears; the next logical memory location is automatically calculated and printed for you. The entry format remains the same whichever method is used. To abort data entry, hit 'A', and to return to the option menu use '–', which is the universal return-to-menu key throughout the program.

(V) View Memory: after selecting the 'View' option you will be asked for starting and ending locations (which default to 0 and 65528/FFF8H respectively if none is specified). Again, the 'A' key may be used to abort.

(R) Run: in the execute mode you will be asked to confirm your intention by typing 'R'. After entering the starting location of your routine, and assuming there is provision for a RET to Basic, you will be returned to the main menu after execution.

(D) Decimal-Hex and (H) Hex-Decimal: simple to use, just enter the number to be converted and hit RETURN. Press RETURN to use again or '–' to exit.

(M) Move Memory: you will be asked to enter the source, destination and length of memory to be moved, and are returned to the main menu on completion. The code for the routine is POKEd into memory from 29200/7210H onwards, which is part of the video RAM used by the hi-res screen. This doesn't rule out the use of MODE (1) as the routine is POKEd into place when needed.

(P) Printout: if you require a hard copy, ensure that your printer is connected before power-up. The routine was written for the PP-40 Printer/Plotter, although any printer should do. Note that line 4030 sets the printer to 40-column mode and selects black ink. Simply replaceing this with the appropriate instructions for your printer. After providing the code to be dumped with a name, hitting RETURN will enter the View mode where operation is as described here.

(X) Exit: you will be asked to confirm that this is your intention — 'YES' is the only way out.

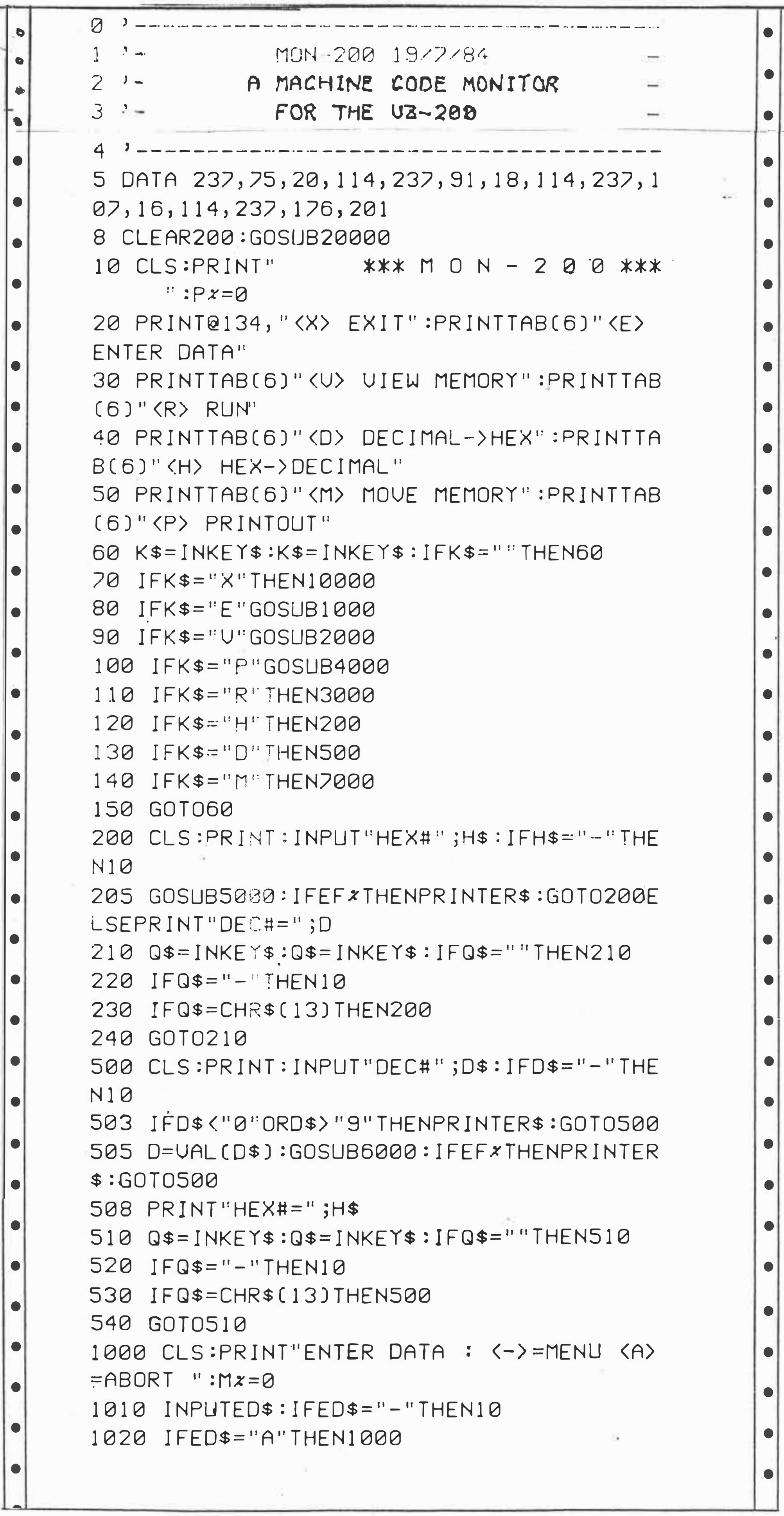
Note that the following should be typed in with inverse text:

- line 10 : everything within the quote mark
- lines 20-50 : the letters inside the greater/less than symbols.

A.P.C. Nov. '84 V5(11)

p 208 – 212

1 of 5.

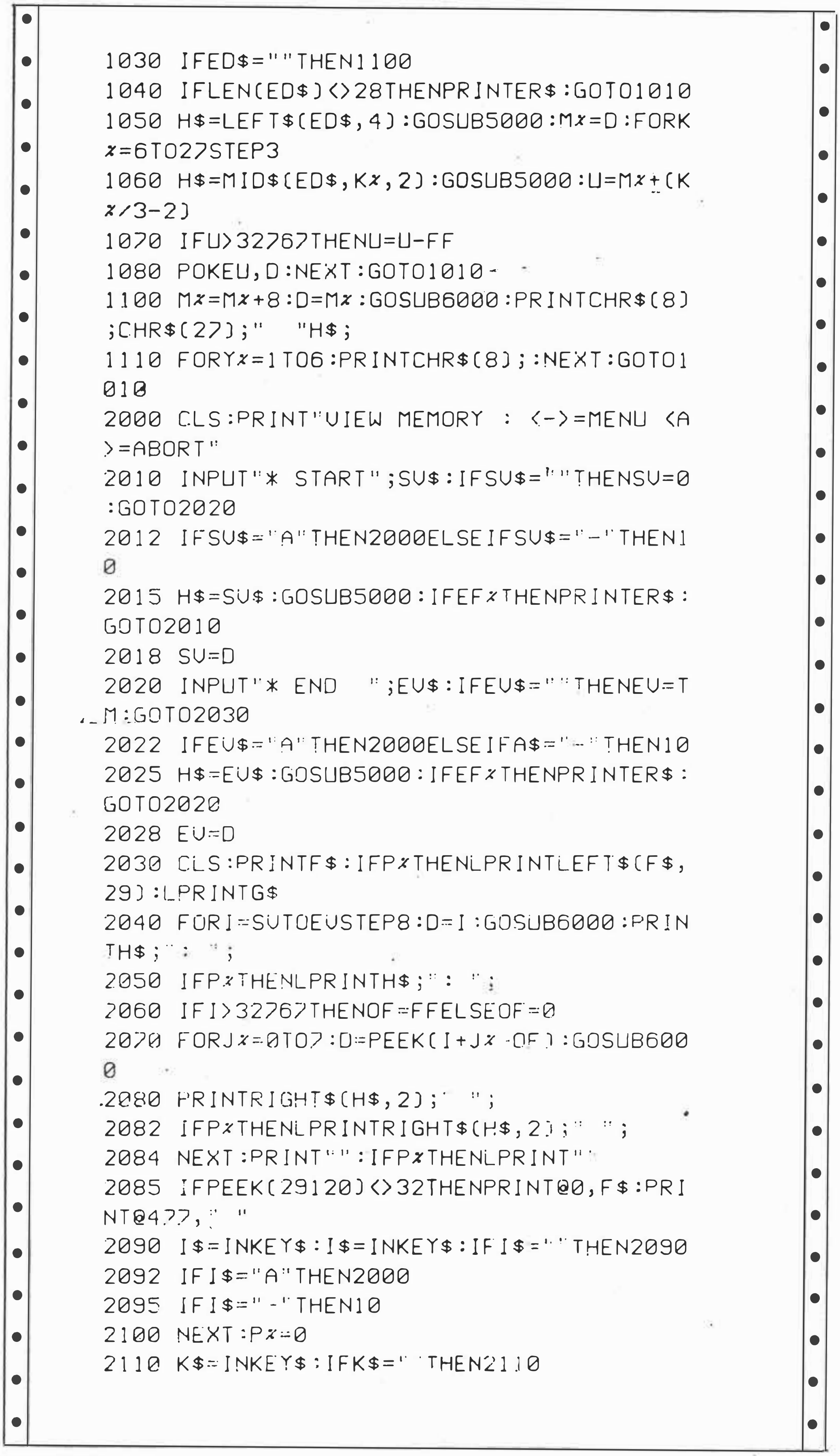


```
0 '--------------------------------------
1 '-          MON-200 19/7/84           -
2 '-      A MACHINE CODE MONITOR        -
3 '-          FOR THE VZ-200            -
4 '--------------------------------------
5 DATA 237,75,20,114,237,91,18,114,237,1
07,16,114,237,176,201
8 CLEAR200:GOSUB20000
10 CLS:PRINT"       *** M O N - 2 0 0 ***
    ":P%=0
20 PRINT@134,"<X> EXIT":PRINTTAB(6)"<E>
ENTER DATA"
30 PRINTTAB(6)"<V> VIEW MEMORY":PRINTTAB
(6)"<R> RUN"
40 PRINTTAB(6)"<D> DECIMAL->HEX":PRINTTA
B(6)"<H> HEX->DECIMAL"
50 PRINTTAB(6)"<M> MOVE MEMORY":PRINTTAB
(6)"<P> PRINTOUT"
60 K$=INKEY$:K$=INKEY$:IFK$=""THEN60
70 IFK$="X"THEN10000
80 IFK$="E"GOSUB1000
90 IFK$="V"GOSUB2000
100 IFK$="P"GOSUB4000
110 IFK$="R"THEN3000
120 IFK$="H"THEN200
130 IFK$="D"THEN500
140 IFK$="M"THEN7000
150 GOTO60
200 CLS:PRINT:INPUT"HEX#";H$:IFH$="-"THE
N10
205 GOSUB5000:IFEF%THENPRINTER$:GOTO200E
LSEPRINT"DEC#=";D
210 Q$=INKEY$:Q$=INKEY$:IFQ$=""THEN210
220 IFQ$="-"THEN10
230 IFQ$=CHR$(13)THEN200
240 GOTO210
500 CLS:PRINT:INPUT"DEC#";D$:IFD$="-"THE
N10
503 IFD$<"0"ORD$>"9"THENPRINTER$:GOTO500
505 D=VAL(D$):GOSUB6000:IFEF%THENPRINTER
$:GOTO500
508 PRINT"HEX#=";H$
510 Q$=INKEY$:Q$=INKEY$:IFQ$=""THEN510
520 IFQ$="-"THEN10
530 IFQ$=CHR$(13)THEN500
540 GOTO510
1000 CLS:PRINT"ENTER DATA : <->=MENU <A>
=ABORT ":M%=0
1010 INPUTED$:IFED$="-"THEN10
1020 IFED$="A"THEN1000
```

Nov. 84   V5(11)

P 208-212

2 of 5

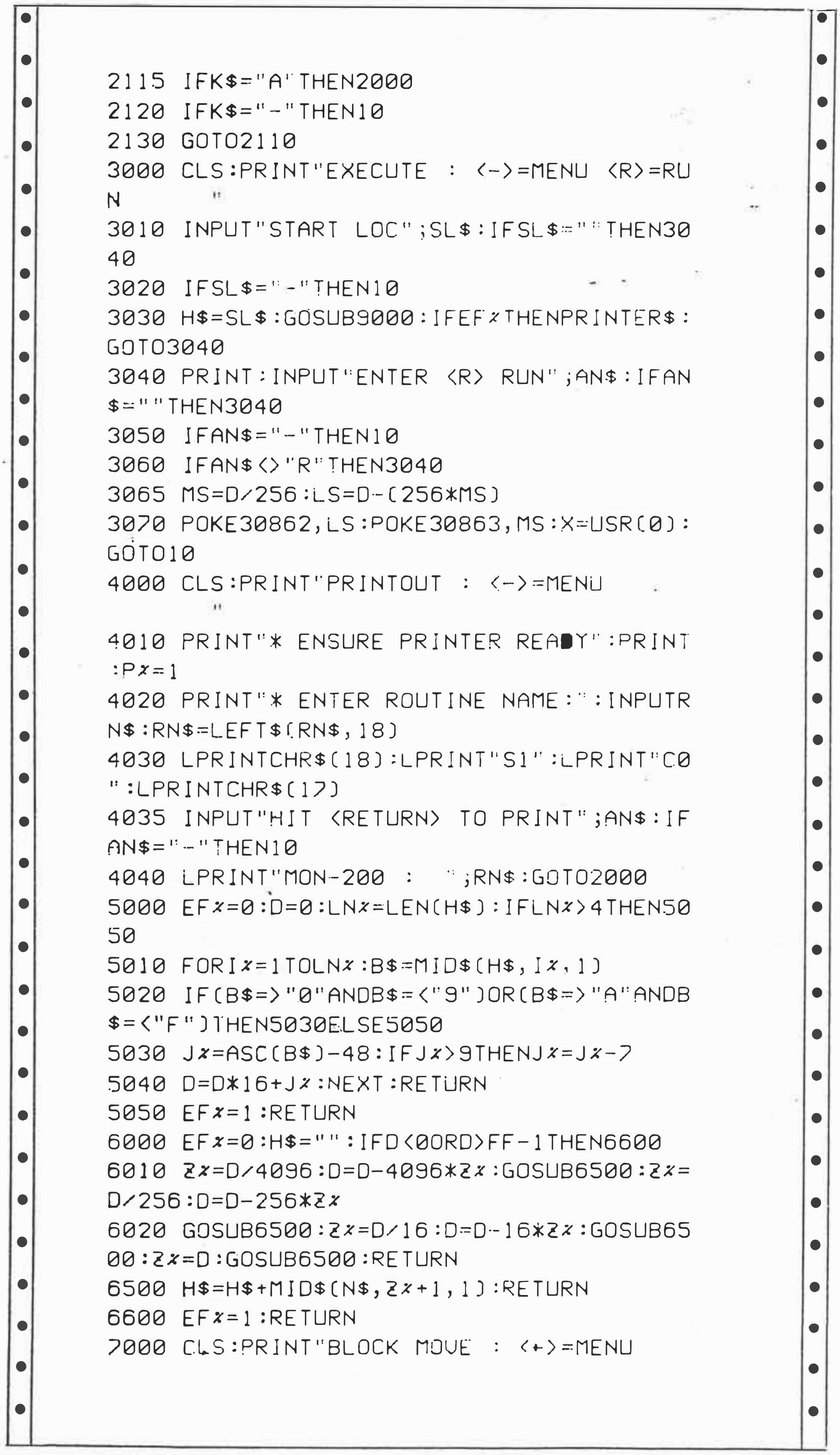


```
1030 IFED$=""THEN1100
1040 IFLEN(ED$)<>28THENPRINTER$:GOTO1010
1050 H$=LEFT$(ED$,4):GOSUB5000:M%=D:FORK
%=6TO27STEP3
1060 H$=MID$(ED$,K%,2):GOSUB5000:U=M%+(K
%/3-2)
1070 IFU>32767THENU=U-FF
1080 POKEU,D:NEXT:GOTO1010-
1100 M%=M%+8:D=M%:GOSUB6000:PRINTCHR$(8)
;CHR$(27);"   "H$;
1110 FORY%=1TO6:PRINTCHR$(8);:NEXT:GOTO1
010
2000 CLS:PRINT"VIEW MEMORY : <->=MENU <A
>=ABORT"
2010 INPUT"* START";SV$:IFSV$=""THENSV=0
:GOTO2020
2012 IFSV$="A"THEN2000ELSEIFSV$="-"THEN1
0
2015 H$=SV$:GOSUB5000:IFEF%THENPRINTER$:
GOTO2010
2018 SV=D
2020 INPUT"* END   ";EV$:IFEV$=""THENEV=T
M:GOTO2030
2022 IFEV$="A"THEN2000ELSEIFA$="-"THEN10
2025 H$=EV$:GOSUB5000:IFEF%THENPRINTER$:
GOTO2020
2028 EV=D
2030 CLS:PRINTF$:IFP%THENLPRINTLEFT$(F$,
29):LPRINTG$
2040 FORI=SVTOEVSTEP8:D=I:GOSUB6000:PRIN
TH$;": ";
2050 IFP%THENLPRINTH$;": ";
2060 IFI>32767THENOF=FFELSEOF=0
2070 FORJ%=0TO7:D=PEEK(I+J%-OF):GOSUB600
0
2080 PRINTRIGHT$(H$,2);" ";
2082 IFP%THENLPRINTRIGHT$(H$,2);" ";
2084 NEXT:PRINT"":IFP%THENLPRINT""
2085 IFPEEK(29120)<>32THENPRINT@0,F$:PRI
NT@477," "
2090 I$=INKEY$:I$=INKEY$:IFI$=""THEN2090
2092 IFI$="A"THEN2000
2095 IFI$="-"THEN10
2100 NEXT:P%=0
2110 K$=INKEY$:IFK$=""THEN2110
```

Nov. 84 V 5(11)

208-212

3 of 5.

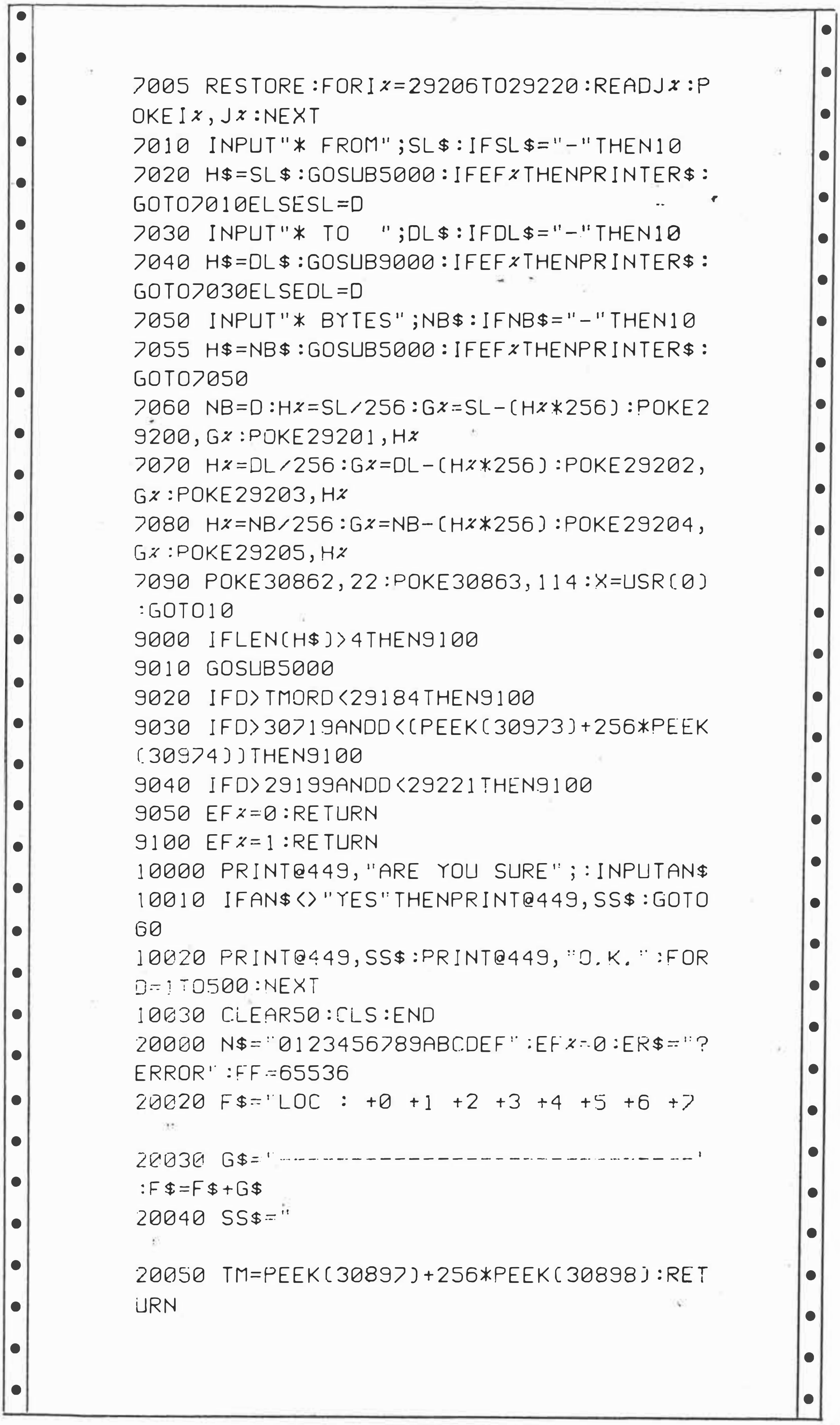


```
2115 IFK$="A"THEN2000
2120 IFK$="-"THEN10
2130 GOTO2110
3000 CLS:PRINT"EXECUTE : <->=MENU <R>=RU
N      "
3010 INPUT"START LOC";SL$:IFSL$=""THEN30
40
3020 IFSL$="-"THEN10
3030 H$=SL$:GOSUB9000:IFEF%THENPRINTER$:
GOTO3040
3040 PRINT:INPUT"ENTER <R> RUN";AN$:IFAN
$=""THEN3040
3050 IFAN$="-"THEN10
3060 IFAN$<>"R"THEN3040
3065 MS=D/256:LS=D-(256*MS)
3070 POKE30862,LS:POKE30863,MS:X=USR(0):
GOTO10
4000 CLS:PRINT"PRINTOUT : <->=MENU
      "

4010 PRINT"* ENSURE PRINTER READY":PRINT
:P%=1
4020 PRINT"* ENTER ROUTINE NAME:":INPUTR
N$:RN$=LEFT$(RN$,18)
4030 LPRINTCHR$(18):LPRINT"S1":LPRINT"C0
":LPRINTCHR$(17)
4035 INPUT"HIT <RETURN> TO PRINT";AN$:IF
AN$="-"THEN10
4040 LPRINT"MON-200 :   ";RN$:GOTO2000
5000 EF%=0:D=0:LN%=LEN(H$):IFLN%>4THEN50
50
5010 FORI%=1TOLN%:B$=MID$(H$,I%,1)
5020 IF(B$=>"0"ANDB$=<"9")OR(B$=>"A"ANDB
$=<"F")THEN5030ELSE5050
5030 J%=ASC(B$)-48:IFJ%>9THENJ%=J%-7
5040 D=D*16+J%:NEXT:RETURN
5050 EF%=1:RETURN
6000 EF%=0:H$="":IFD<0ORD>FF-1THEN6600
6010 Z%=D/4096:D=D-4096*Z%:GOSUB6500:Z%=
D/256:D=D-256*Z%
6020 GOSUB6500:Z%=D/16:D=D-16*Z%:GOSUB65
00:Z%=D:GOSUB6500:RETURN
6500 H$=H$+MID$(N$,Z%+1,1):RETURN
6600 EF%=1:RETURN
7000 CLS:PRINT"BLOCK MOVE : <->=MENU
```

Nov. 84 V 5(11).

208-212

4 of 5



```
7005 RESTORE:FORI%=29206TO29220:READJ%:P
OKEI%,J%:NEXT
7010 INPUT"* FROM";SL$:IFSL$="-"THEN10
7020 H$=SL$:GOSUB5000:IFEF%THENPRINTER$:
GOTO7010ELSESL=D
7030 INPUT"* TO  ";DL$:IFDL$="-"THEN10
7040 H$=DL$:GOSUB9000:IFEF%THENPRINTER$:
GOTO7030ELSEDL=D
7050 INPUT"* BYTES";NB$:IFNB$="-"THEN10
7055 H$=NB$:GOSUB5000:IFEF%THENPRINTER$:
GOTO7050
7060 NB=D:H%=SL/256:G%=SL-(H%*256):POKE2
9200,G%:POKE29201,H%
7070 H%=DL/256:G%=DL-(H%*256):POKE29202,
G%:POKE29203,H%
7080 H%=NB/256:G%=NB-(H%*256):POKE29204,
G%:POKE29205,H%
7090 POKE30862,22:POKE30863,114:X=USR(0)
:GOTO10
9000 IFLEN(H$)>4THEN9100
9010 GOSUB5000
9020 IFD>TMORD<29184THEN9100
9030 IFD>30719ANDD<(PEEK(30973)+256*PEEK
(30974))THEN9100
9040 IFD>29199ANDD<29221THEN9100
9050 EF%=0:RETURN
9100 EF%=1:RETURN
10000 PRINT@449,"ARE YOU SURE";:INPUTAN$
10010 IFAN$<>"YES"THENPRINT@449,SS$:GOTO
60
10020 PRINT@449,SS$:PRINT@449,"O.K.":FOR
D=1TO500:NEXT
10030 CLEAR50:CLS:END
20000 N$="0123456789ABCDEF":EF%=0:ER$="?
ERROR":FF=65536
20020 F$="LOC : +0 +1 +2 +3 +4 +5 +6 +7

20030 G$="------------------------------"
:F$=F$+G$
20040 SS$="

20050 TM=PEEK(30897)+256*PEEK(30898):RET
URN
```

Nov 84   V5(11).

208-212

5 of 5.

LPRINTER

By Robert Quinn

A PP40 printer program for the VZ-200, it allows you to use your VZ-200 as a typewriter, LPRINTING in upper case, lower case, normal or inverse print, and to LPRINT graphics.

Instructions

Switch on your PP40 printer plotter. RUN the program and a blinking cursor will appear on a black screen to indicate your start position. Type using any of the character keys on the keyboard by themselves or with the SHIFT key held down. The corresponding characters will print on the screen and LPRINT to your PP40 printer.

LPRINTER starts up in normal upper case mode. Press the CTRL key to shift to lower case LPRINTING; and press the CTRL key again to return to upper case LPRINTING.

Hold the SHIFT key and press the X key to shift to inverse printing and LPRINTING: inverse LPRINTING is distinguished from normal LPRINTING by underlining.

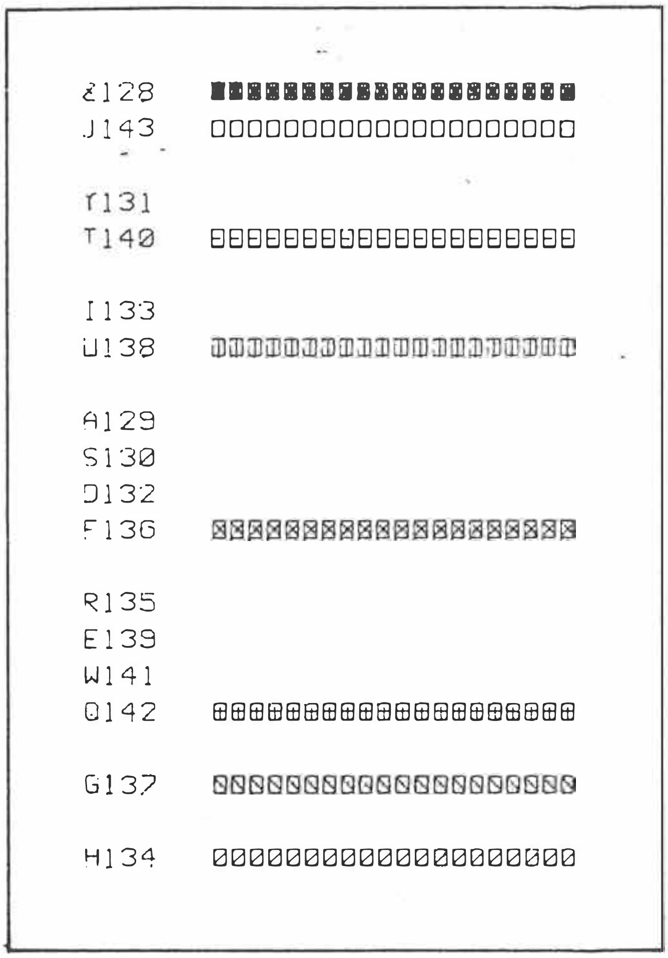
A carriage return will operate automatically to start a new line when the end of the line is reached, though the end of the LPRINTER line (40 characters) will not correspond with the end of the screen line (32 characters).

A carriage return can be accomplished any time by pressing the RETURN key.

Backspacing to the start of the LPRINTER line can be accomplished by holding the SHIFT key and pressing the B key. Everytime SHIFT and B are pressed the pen holder will move left one character. The screen cursor will backspace as well, but will erase characters it passes over.

The screen cursor will blink a hash sign when the 35th position on the cursor LPRINTER line is reached and a hi-lo warning buzz will sound to indicate that you are nearing the end of the LPRINTER line.

The VZ-200 supports sixteen graphic characters. LPRINTER LPRINTS graphic characters but does not uniquely define every one of the sixteen. In the categories that follow the letters designate the letter keys by which (with the SHIFT key held down) the corresponding screen graphic characters are accessed. The number following each letter is the ASCII code for the graphic character. Then follows a line of the LPRINTER graphic character that defines those screen graphic characters. You may wish to refine the definition of screen graphics so as to give each screen graphic character a unique LPRINTER character.



```
Z128
J143

Y131
T140

I133
U138

A129
S130
D132
F136

R135
E139
W141
Q142

G137

H134
```

A COPY subroutine is RUN from within the program by holding the SHIFT key and pressing the C key, producing a printout of the entire contents of the screen — normal, INVERSE and graphics.

With LPRINTER CLOADed but not RUNning the COPY subroutine can be used directly by entering the command GOSUB300 and pressing the RETURN key.

```
5 REM LPRINTER FOR VZ200  BY ROBERT QUIN
N
10 COLOR,1:SOUND0,2:CLS

20 FORR=1TO2STEP0:IFPEEK(26875)=249THENS
OUND20,1:P=NOTP
22 IFPEEK(26875)=243THENLPRINTCHR$(13);:
LPRINT:D=0:GOSUB300
25 IFPEEK(26877)=251THENK=NOTK:SOUND20,1
26 IFPEEK(26875)=250ANDD>0THENGOSUB200
27 IFC=20ANDD=35THENPRINT"#";CHR$(8);:
GOTO29
28 IFC=20THENPRINT"_";CHR$(8);
29 C=C+1:IFC=40THENC=1:PRINT" ";CHR$(8);
30 B$=INKEY$:A$=INKEY$:IFA$<>""THENSOUND
10,1:GOSUB50
40 A$="":NEXT
```

PCG. Nov 84 1(4) p55-56 1 of 2.

```
50 A=ASC(A$):B=A:IFP=-1ANDA>31ANDA<64THE
NB=B+132
60 IFP=-1ANDA>63ANDA<128THENB=B+128
65 IFK=-1ANDA>63ANDA<95THENA=A+32
70 IFA>127THENGOSUB110:GOTO90
80 LPRINTCHR$(A);
90 IFP=-1ANDA<127ANDA>31THENLPRINTCHR$(8
);CHR$(95);
95 IFB=13THENPRINT" ";CHR$(8);:D=-1
100 PRINTCHR$(B);:D=D+1:IFD=35THENSOUND3
1,2;20,1
102 IFD=41THEND=1
105 RETURN
110 IFA=133ORA=138THENLPRINTCHR$(85);CHR
$(8);CHR$(84);:RETURN
120 IFA=131ORA=140THENLPRINTCHR$(85);CHR
$(8);CHR$(69);:RETURN
130 IFA=137THENGOSUB190:LPRINTCHR$(92);:
RETURN
140 IFA=134THENGOSUB190:LPRINTCHR$(47);:
RETURN
150 IFA=143THENLPRINTCHR$(79);CHR$(8);CH
R$(85);:RETURN
160 IFA=128THENGOSUB190:LPRINTCHR$(42);C
HR$(8);CHR$(35);:RETURN
165 IFA=135ORA>138THENGOSUB190:LPRINTCHR
$(43);:RETURN
170 LPRINTCHR$(127);:RETURN

190 LPRINTCHR$(79);CHR$(8);CHR$(85);CHR$
(8);:RETURN


200 SOUND10,1:PRINT" ";CHR$(8);CHR$(8);:
LPRINTCHR$(8);
210 D=D-1:RETURN


300 FORT=28672TO29183:A=PEEK(T)
310 IFA<32THENLPRINTCHR$(A+64);ELSEIFA<6
4THENLPRINTCHR$(A);
320 IFA>63ANDA<96THENLPRINTCHR$(A);CHR$(
8);CHR$(95);
330 IFA>95ANDA<128THENLPRINTCHR$(A-64);C
HR$(8);CHR$(95);
340 IFA>127THENGOSUB370
350 D=D+1:IFD=32THEND=0:LPRINTCHR$(13);
360 NEXT:D=0:LPRINTCHR$(13);:LPRINT:RETU
RN

370 IFA>143THENA=A-16:GOTO370
380 GOSUB110:RETURN
```

PCG, Nov 84 1(4) p 55-56 2 of 2.

VZ-200

Reverse video

An interesting effect available on the VZ-200 is the ability to reverse the video display via a POKE command.

On turning on the VZ-200 (version 2.0), the text is shown as black on a light green background. COLOR,1 changes the display to black on an orange background.

POKEing 30744,1 reverses the image, giving light green text on a dark green background with COLOR,0 and orange text on a red background with COLOR,1.

POKEing 30744,0 reverts back to black text on a light background.

In summarising:

POKE 30744,0 : COLOR,0 — black text on light green.
POKE 30744,0 : COLOR,1 — black text on orange.
POKE 30744,1 : COLOR,0 — light green text on dark green.
POKE 30744,1 : COLOR,1 — orange on red.

Using a black and white TV set as monitor, the effect is shown as black text on white, or white text on black, respectively.

POKEing these values has no effect on the eight foreground colours in low resolution graphics MODE(0), only the background colours, nor do they have any effect in high resolution, MODE(1). They do, however have an effect on the block graphics on the upper case J and Z keys, the poles of these two block graphics being reversed when entering POKE 30744,0.

PCG. Nov 84 suppl.

VZ-200

Enlarged characters

By John ten Velde

This program allows the user to create a "notice board" containing a message in enlarged characters. It could be used for advertising purposes or as a teaching aid.

The program consists of three main parts: the character information section (lines 33 to 90); the input section which allows the user to enter a message of up to 5 lines of 15 characters (lines 100 to 290); and the outout section which displays the message on the screen.

In the character information section, each character is defined by a 27 code which represents the pixels turned on in a 7 x 9 pixel grid. The digits are made up of nine three groups, each group representing one of the character in binary form. The character information can be altered to produce characters chosen by the user required.

```
4 '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
5 '@@@ ENLARGED CHARACTERS @@@
6 '@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
7 REM
10 POKE 30744,1:CLEAR1500:DIMA$(90):CLS
20 DIMB(15,5)
33 A$(33)="008008008008008008008000008"'!
42 A$(42)="073073042028127028042073073"'*
48 A$(48)="028034065065065065065034028"'0
49 A$(49)="008024040008008008008008028"'1
50 A$(50)="028034065002004008016032127"'2
51 A$(51)="028034065002012002065034028"'3
52 A$(52)="004012020036127004004004004"'4
53 A$(53)="127064064064124002001002124"'5
54 A$(54)="028034065064092098065034028"'6
55 A$(55)="127001002004008016032064064"'7
56 A$(56)="028034065034028034065034028"'8
57 A$(57)="028034065035029001065034028"'9
58 A$(58)="000000028028000028028000000"':
59 A$(59)="000000028028000028028012024"';
65 A$(65)="008020034065127065065065065"'A
66 A$(66)="124066065066124066065066124"'B
67 A$(67)="028034065064064064065034028"'C
68 A$(68)="124066065065065065065066124"'D
69 A$(69)="127064064064124064064064127"'E
70 A$(70)="127064064064124064064064064"'F
71 A$(71)="028034065064071065065034028"'G
72 A$(72)="065065065065127065065065065"'H
73 A$(73)="028008008008008008008008028"'I
74 A$(74)="001001001001001001065034028"'J
75 A$(75)="065066068072080104068066065"'K
76 A$(76)="064064064064064064064064127"'L
77 A$(77)="065099085073065065065065065"'M
78 A$(78)="065097081073069067065065065"'N
79 A$(79)="028034065065065065065034028"'O
80 A$(80)="124066065066124064064064064"'P
81 A$(81)="028034065065065065069034029"'Q
82 A$(82)="124066065066124072078066065"'R
83 A$(83)="028034065032028002065034028"'S
84 A$(84)="127008008008008008008008008"'T
85 A$(85)="065065065065065065065034028"'U
86 A$(86)="065065065034034020020008008"'V
87 A$(87)="065065065065065073093119034"'W
88 A$(88)="065065034020008020034065065"'X
89 A$(89)="065065034020008008008008008"'Y
90 A$(90)="127001002004008016032064127"'Z
100 FOR X=28807 TO 28823 :POKE X,96:NEXT
120 FOR X=28839 TO 28967 STEP32:POKE X,96
130 NEXT
140 FOR X=28855 TO 28983 STEP32:POKE X,96
150 NEXT
160 FOR X=28999 TO 29015 :POKE X,96:NEXT
180 FOR Y=168 TO 296 STEP 32
190 FOR X=0 TO 14
200 A$=INKEY$:A$=INKEY$
210 IF A$="" THEN 200
220 PRINT@(Y+X),A$;
225 IF INKEY$<>""THEN225
230 NEXTX
240 IF INKEY$<>""THEN240 ELSE NEXT Y
250 FOR Y= 0 TO 4
260 FOR X= 0 TO 14
270 B(X,Y)=PEEK((901+Y)*32+8+X)
280 NEXT X
290 NEXT Y
300 MODE (1)
500 FOR Y=0 TO 4
510 FOR X=0 TO 14
520 B =B(X,Y)
550 IFB<32THENB=B+64
560 IFB= 32 THEN 660
570 B$=A$(B)
580 FOR Y0=0 TO 8
590 A=VAL(MID$(B$,(Y0+1)*3-2,3))
600 FOR N=6 TO 0 STEP -1
610 M=2^N
620 IFA>=M THENSET(X*8+6-N,Y*11+Y0):A=A-M
630 NEXT N
640 NEXT Y0
660 NEXT X
670 NEXTY
680 T$=INKEY$:T$=INKEY$
690 IF T$="" THEN 680
700 IF C=2 THEN C=3 ELSE C=2
710 COLOR C
720 GOTO500
```

Basic understanding

I have come to the conclusion that although people want more software written for their particular micro, nobody is prepared to give away any secrets, so that more up-and-coming programmers can have a better understanding of the way a certain problem is solved by a computer.

In a previous edition of *APC*, in the Communications section, there was a cry of despair from a VZ-200 user for a word processor type program for the VZ-200. On reading through the Programs section of a few *APC* issues, it is easy to see why nobody (novices) can write programs for the VZ-200. It appears that those who know the deep dark secrets of programming would like to keep these secrets to themselves.

All of the programs that I have seen in *APC* for the VZ-200, have had no comments (apart from those with the authors name etc) in them. It doesn't take long to add a few comments into a program just to let the reader know what the program is doing. For example the following code is from a Basic program:

```
210 CLS:PRINT"RECORD NUMBER:";NF%+1:PRINT
220 FORL1=1TONR%:PRINTRN$(L1,1);:INPUTRC$(L1,NF%+1)
230 IF(L1=1)AND(RC$(L1,NF%+1)="")THENRETURN
240 NEXT:NF%=NF%+1:IFNF%<50THEN200
250 PRINT"DATABASE FULL!!!":FORL1=1TO1000:NEXT:RETURN
400 CLS:INPUT"WHAT RECORD";NU%
410 IF(NU%>NF%)OR(NU%<0)THEN400
420 IFNU%=0THENRETURN
430 ......e.t.c.
```

Wouldn't it be a lot easier to see what the program is doing (apart from spending hours tracing through it) if it were presented in the following form:

```
198 REM  *************************************
199 REM  ***       ADDING A RECORD         ***
210 CLS:PRINT .......e.t.c.
260 REM  ***       END OF ADDITION         ***
261 REM  *************************************
399 REM  ***      CHANGING A RECORD        ***
400 CLS:INPUT .......e.t.c.
```

At least from there, the reader can see what the particular section of a program is doing; then if they want to go into any more detail, they can use their Basic reference manual. It also helps if there is a list of the variables (in REM statements), and what each variable is used for, at the beginning of the program. Another tip is to use variables that represent something. In the example, NU% is for NUmeric storage, NR% is for Number of Records, L1 is for a Loop (there are three of these in the program, L1 . . . L3), and RC$ stands for Record Contents.

Some readers may think this all a gross waste of time and effort, but if their little micros ever acquire the capability of running other high level languages (eg, Pascal, Cobol), they will see why this is a good practice to get in to.

There is no need to go overboard with the comments, but imagine a beginner in this wondrous field of computing, sitting there with his/her reference manual, and trying to figure what the heck is going on in the first lot of code or what part of the program it is. I have visions of a 12/13 year old in tears, ripping up the manual, pulling the plug on the computer and vowing never to use it again.

If we want this industry to grow, lets share the secrets around so that the up and coming youngsters have the opportunity of learning from things that we had to find out for ourselves.

S Hobson

Feb 85 6(2)

VZ-200 into puberty

Steve Olney has produced a machine code utility which "re-enables all 23 hidden commands resident in the VZ Basic ROM". Apparently this means VZ-200 will then have most of the Level II TRS-80 commands and a couple more. It'll set you back a moderate $15. Write to Steve Olney, 200 Terrace Road, North Richmond, 2754.

Feb 85 6(2)

LOW BAND DX

CALCULATING THE GREY LINE

By Greg Baker

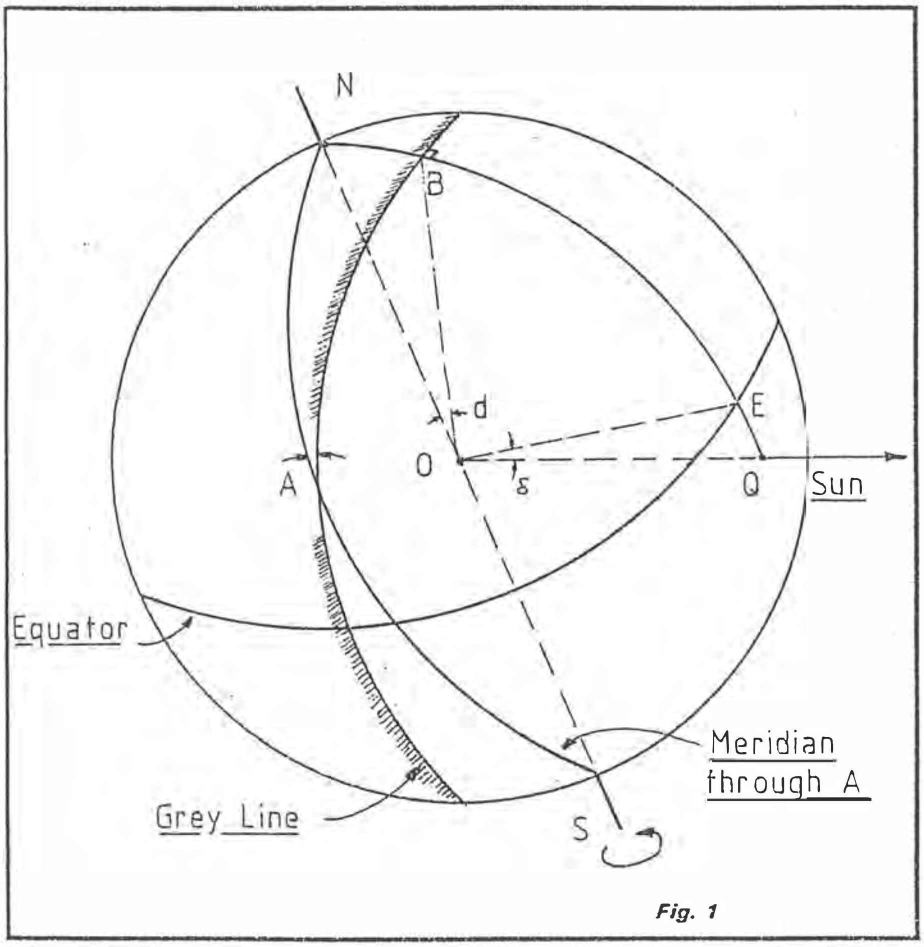


Fig. 1

As the earth spins on its axis, there is always one hemisphere in sunlight and one in shadow. The junction between these two hemispheres - day and night - is a great circle which is called the "grey line". A zone of undefined width along the grey line is called they "grey zone". The grey zone is of interest largely because here there is a fairly abrupt change in the ionosphere. For example, the D-layer disappears almost completely at sunset, bringing with its passing the rapid build-up of MF DX; the opposite is the case at sunrise.

There is also the well known property that efficient communication is possible between stations both lying in the grey zone. Thus it is of interest to amateurs to know where the grey zone is at any time and to exploit its properties where possible.

The easiest method of finding the grey line is to buy a radio globe of the world such s the "Grey Line Radio Globe" reviewed in Amateur Radio Action, Volume 5, Number 11, or to construct one from an ordinary school kids globe as described in Practical Wireless, March 1984.

A more difficult method, but a more accurate one, is to calculate the grey line. It is a relatively straight-forward matter to calculate for any date the bearings of the grey line as it passes any location. For this is a calculator or set of mathematical tables is sufficient. To calculate **when** the grey line passes a location — sunrise and sunset times — a moderately sophisticated calculator is still sufficient. However, in the long sequence of calculations involved, a home computer is not only quicker and easier, it is likely to be more accurate. Because of this, the latter part of the article is directed towards home computers, with a reference for further reading for those with calculators only.

BEARINGS OF THE GREY LINE

In Figure 1, NAS is the meridian through location A on the grey line. Q is the subsolar point, ie the place on the earth's surface which lies in a direct line between the centre of the earth (O) and the sun. Any great circle through Q intersects the grey line at right angles. QBN is that part of one of these great circles which passes through the north geographic pole.

What we want to find is angle A and from it the bearings of the grey line, X, and (180 + X) degrees. These will be the bearings at

1 of 6. ? Feb 85. 19-26.

sunrise; at sunset the bearings will be (360 — X) and (180 — X) degrees.

For spherical triangle NAB,

sin A/ sin NB = sin B/ sin NA.

NA and NB, although sides of the triangle, are expressed as the angles these sides subtend at the centre of the earth.

Noting that B = sin90 = 1, that NA is (90 -latitude A) if we set north latitudes positive and south latitudes negative, and setting NB = d, we get

sin A = sin d / sin (90 — lat A)
= sin d / cos (lat A)

Referring again to Figure 1, we can see that d is the same as QOE, which is called the sun's declination.

We know latitude. The only unknown is declination. VK2KII in ARA Volume 6, Number 9, page 33 gives a rough formula which is probably good enough for most purposes. That formula (modified) is d = —23.4 sin (0.9856 D) where D is the number of days after 21st September. The value 0.9856 D is a value in degrees, not radians. The declination, d, clearly ranges in value from —23.4 to 23.4 degrees.

Note that this is the negative of VK2KII's formula to ensure that the usual sign conventions apply, ie northern declinations positive and southern declinations negative. VK2KII has it the other way around.

Since cos (lat A) is positive regardless of the sign of the latitude, A takes on the sign of declination. That is, south declinations such as shown in Figure 1, produce negative values for A and north declinations produce positive values for A. To get the bearing X, the rule is to set X equal to (360 — A) degrees and subtract 360 if this exceeds 360 degrees. This leads to the two sunrise bearings X and (X + 180) degress and the two sunset bearings (360 — X) and (180 — X) degrees. If any of these exceeds 360 degrees, subtract 360 degrees.

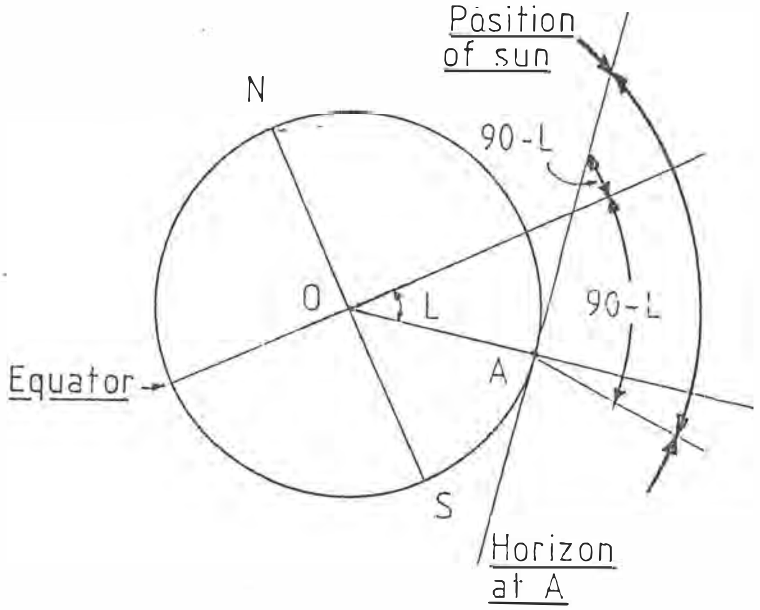


Figure 3

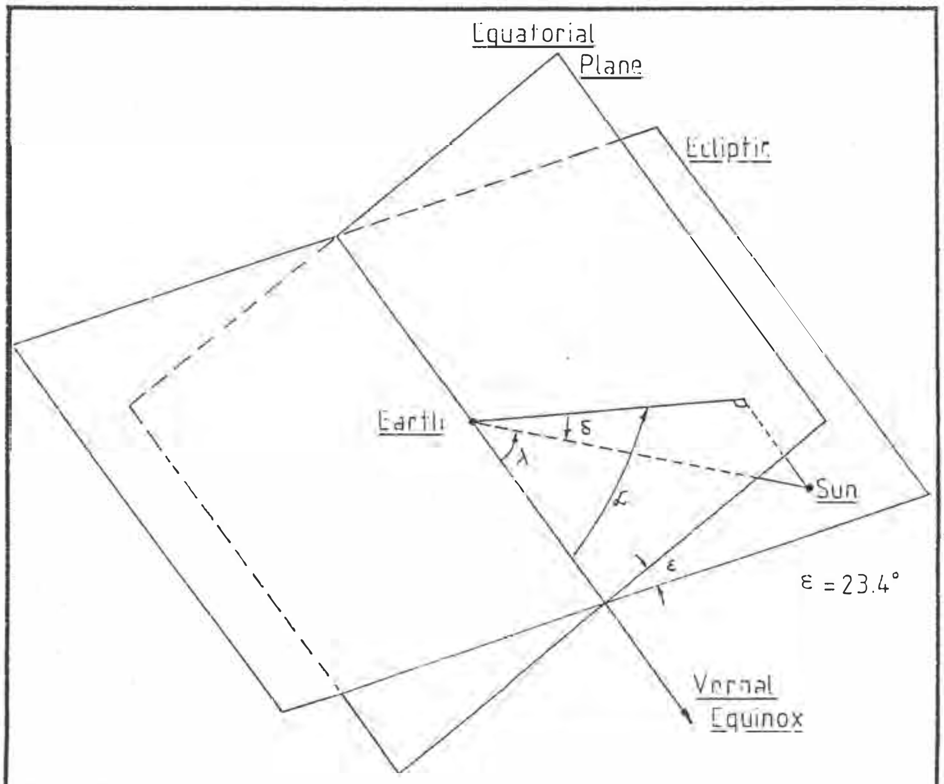


Figure 2

For example, on 3rd May, D = 224 and hence d = 15.3 degrees. At latitude —35 degrees, say, A = 18.7 degrees and X = 341.3 degrees. The sunrise bearings of the grey line are 341.3 and 161.3 degrees; the sunset bearings are 18.7 and 198.7 degrees

Sunrise and Sunset Times

It is possible to look at a daily newspaper for the times of sunrise and sunset. They don't vary much from day to day, so today's times are probably good enough for tomorrow. The information above is, in these circumstances, sufficient for day to day operation.

However, if you don't buy a daily newspaper, have poor library services, or you want to know, for example, when the grey line will pass the operators in your net across the Tasman, you may want to calculate sunrise and sunset times.

Program GREYLINE described and listed below carries out these calculations accurate to a few minutes. The rest of this section is a description of the procedure followed and can be omitted on first reading. Peter Duffet-

Smith in his excellent "Practical Astronomy with your Calculator", referenced in full below, has more detail and the interested reader is strongly recommended to get hold of a copy and read the relevant sections.

Sunrise and sunset times depend on where the sun is in relation to the earth and on the location of the point of observation. To find out where the sun is on any date, it is necessary to know how the position of the sun is described by astronomers.

There are two co-ordinate systems used: equatorial co-ordinates and ecliptic co-ordinates.

Equatorial co-ordinates are based on the equatorial plane which is the projection of the plane cutting the earth at the equator. Ecliptic co-ordinates are based on the ecliptic which is the plane in which the earth and sun move. Figure 2 shows both these planes which are at an angle of about 23.4 degrees to one another.

The planes meet in a line which passes through the earth. One direction along this line from earth is used as a reference direction for both co-ordinate systems. It is called the vernal equinox because the sun lies in this direction from earth on 21st March in the northern spring.

In each co-ordinate system the plane and reference direction are used in a manner analagous to the way the plane of the equator and the line from the earth's centre to the Greenwich meridian at the equator are used for our usual geographic co-ordinate system.

Ecliptic longitude (lambda) begins at 0 degrees at the vernal equinox and increases in an anti-clockwise direction in the ecliptic plane to 360 degrees back at the vernal equinox again. The ecliptic latitude (beta) begins at 0 degrees and increases to 90 degrees above and decreases to — 90 degrees below the ecliptic. The ecliptic latitude of the sun is zero always of course.

In equatorial co-ordinates the longitude, called right ascension (alpha) is based on the vernal equinox in a way exactly analagous to that for ecliptic longitude. The angle above or below the equatorial plane is the declination (delta) and is positive above the plane and negative below the plane.

Astronomers have tabulations of various data, including the position of the sun at various times. The position of the sun is given by its ecliptic longitude. Following Duffett-Smith I use the ecliptic longitude at the beginning of 1980 and from this calculate the sun's ecliptic longitude at any time thereafter as:

M + 360/PI.e.sinM + WG

where M = (360/365.2422). d + EG-WG

and D = the number of days since the beginning of 1980

EG = 278.83354 degrees. The ecliptic longitude at the start of 1980.

WG = 282.596403. The ecliptic longitude at perigee, the point where the sun and earth are closest.

e = 0.016718. The eccentricity of the sun-earth orbit.

Because the sun is moving relatively rapidly in relation to the earth, the program calculates the sun's position at the two midnights straddling the day of interest. It later uses these to get weighted average and hence more accurate sunrise and sunset times.

From the ecliptic co-ordinates, convert to equatorial co-ordinates thus:

Right ascension = tan^{-1} (sin lambda.cos EP/cos lambda)

Declination = sin^{-1} (sin EP.sin lambda)

where EP is the angle between the ecliptic and the equatorial plane (23.441884 degrees).

Declination gives (i) the bearings of the grey line — as shown above, (ii) whether the sun rises and sets, and (iii) for how long the sun remains above the horizon if it rises and sets. These last two can be seen by reference to Figure 3. Consider an observer (A) at south latitude L degrees. Here L is treated as the unsigned latitude, ie the absolute value of latitude. If the declination of the sun is more than (90-L) degrees north, the observer at A will never see it. If it has a declination of more than (90-L) degrees south, it will always be above the horizon. If the sun's declination lies in the range (90-L) degrees north to (90-L) degrees south, the sun rises and sets.

The length of time it is above the horizon will depend on the latitude of the observer and the declination of the sun relative to (90-L) north and (90-L) south. Algebraically this time is 2H hours where:

H = (cos^{-1} (-tan Latitude.tan delta))/ 15.

The other equatorial co-ordinate, the right ascension, leads to the precise period within the day that the sun is above the horizon. There are several steps. Right ascension gives local sidereal time (see below) of sunrise and sunset thus:

Rising time = 24 + alpha —H

Setting time = alpha + H.

To understand sidereal times, refer to Figure 4. On 21st March, the sun is at the vernal equinox to an observer on the meridian through V and it is noon. 23 hours 56 minutes later the vernal equinox is again over the meridian at V. One sidereal ("of the stars") day has passed. Four minutes later again, the sun is over the local meridian at V and one solar day has passed. It is noon again. Because the sidereal day is 23 hours 56 minutes long, sidereal noon falls four minutes earlier each day than the day before. There are thus approximately 366 sidereal days in the 365 solar day year and this is because the earth rotates 366 times in the course of one year not 365. A little experimentation with a couple of oranges or tennis balls will show this is the case.

The sidereal rising and setting times need to be converted into UTC thus:

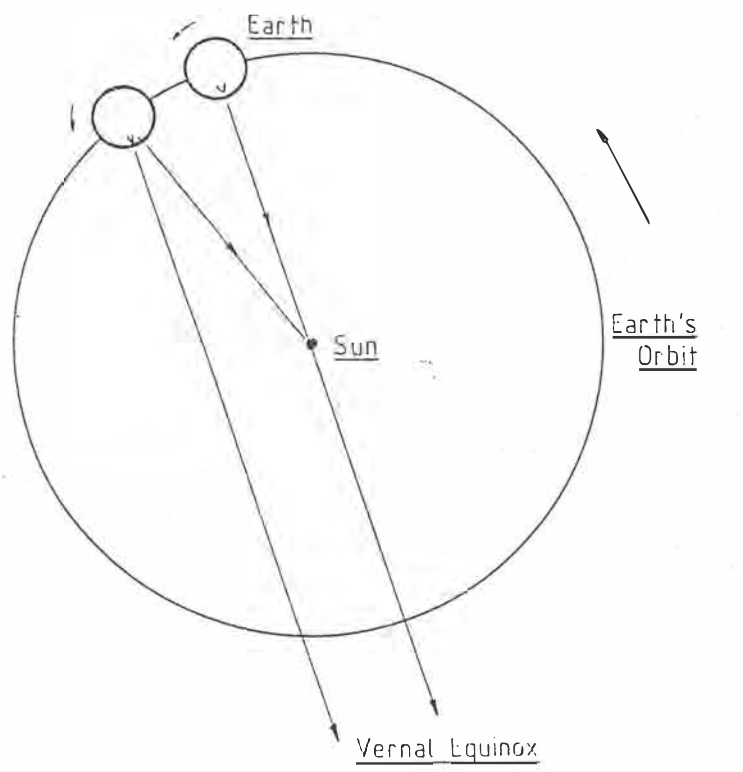


Fig. 4

3 of 6

LOW BAND DX

should be referred to the author. Please include a stamped self addressed envelope and as many details as possible of the problem. Suggestions for improvements and notification of errors will be gratefully received.

Arrays

B(I) The number of days from the beginning of 1980 to the beginning of (1979+1)

C(I,J) Right ascension (J = 1) and declination (J=2) of the sun at the two midnights (I = 1.2) straddling the day of interest.

E(I,J) Number of days in each month (I = 1 to 12) for ordinary (J = 1) and leap years (J = 2)

F(1,J) Bearings of rising (J = 1) and setting (J = 2) sun based on two midnights (I = 1,2) straddling day of interest.

L(I) Ecliptic longitude of the sun at the two midnights (I = 1,2).

Q(I,J) Latitude (I = 1) and longitude (I = 2) of QTH in degrees (J = 1) and minutes (J=2).
Later in program latitude in decimal form in Q(1,1), longitude in Q(2,1).

S(I,J) Local sidereal times of sunrise (J = 1) and sunset (J = 2) based on the two midnights (I = 1,2).

Greenwich sidereal, and UTC times.

Test Data

The following locations, dates, times and bearings may be useful as test data.

UTC = (t - longitude/15 - d.A + B). 0.99727

The expression in the brackets must be made to lie in the range 0 to 24 hours, by addition or subtraction of multiples of 24, before the multiplication takes place. In the equation, t is local sidereal time, d is the number of days since the beginning of 1980, A = 0.0657098, and B is a constant which is different for each year. The program uses B = 17.37 which is near enough for 1985 and 1986. At around 2400 UTC, this formula does not convert accurately. However, sunrise and sunset times in Oceania should not be affected.

The Program

The program asks for the location latitude and longitude and the date in which you are interested. Latitude and longitude need to be signed. North latitudes are positive; south latitudes are negative. West longitudes are negative; east longitudes are positive. Only sign the degrees, not the minutes. Illegal latitudes and longitudes are signalled and the user asked to re-input. The date is input as DD,MM,YY, eg 22nd April 1985 is 22.04,85 or 22,4,85. Dates must be in the range 1,1,-80 to 31,12,99.

Output form is shown in Figure 5. Sunrise and sunset times are accurate to within a few minutes.

The program runs in the un-enlarged VZ200. Only minor translation should be necessary for other machines. Problems

4 of 6.

	Canberra	Adelaide
Latitude	−35°17′	−34°56′
Longitude	149°13′	138°36′
Date	22.4.84	24.3.84
Sunrise (UTC)	2031	2050
Bearings	345.4° 165.4°	358.6° 178.6°
Sunset (UTC)	0728	0849

Bearings	194.6° 14.6°	181.4° 1.4°

References

The basic reference is "Practical Astronomy with Your Calculator", by Peter Duffett-Smith, 2nd Edition, Cambridge University Press, 1982; available in paperback. Begin with Section 45, Sunrise and Sunset. The analogue methods are in ARA Volume 5, Number 11, and Practical Wireless March 1984, and some simple sunrise and sunset calculations are in Ian VK2KII's article in ARA Volume 6, Number 9. The Shortwave Propagation Handbook also addresses the issue of propagation along the grey line (Section 6.8).

```
1Ø   DIM B(2Ø),C(2,2),E(12,2),F(2,2),L(2),
     Q(2,2),S(2,2),T(2)
2Ø   DIM S%(2),T%(2)
3Ø   FOR I=1 TO 2Ø
4Ø   READ B(I)
5Ø   NEXT
6Ø   DATA Ø,366,731,1Ø96,1461,1827,2192,2557,
     2922,3228
7Ø   DATA 3653,4Ø18,4383,4749,5114,5479,5845,
     621Ø,6375,694Ø
8Ø   FOR I=1 TO 12
9Ø   READ E(I,1)
1ØØ  E(I,2)=E(I,1)
11Ø  NEXT
12Ø  DATA 31,28,31,3Ø,31,3Ø,31,31,3Ø,31,3Ø,31
13Ø  E(2,2)=29
14Ø  EG=278.83354
15Ø  WG=282.5964Ø3
16Ø  PI=3.1415927
17Ø  EC=Ø.Ø16718
18Ø  DR=57.29578
19Ø  EP=23.441884
23Ø  FL=Ø
231  PRINT "LATITUDE?   (SIGNED) DEGS, MINS"
24Ø  INPUT Q(1,1),Q(1,2)
25Ø  PRINT "LONGITUDE?  (SIGNED) DEGS, MINS"
26Ø  INPUT Q(2,1),Q(2,2)
27Ø  FOR I=1 TO 2
28Ø  Z=9Ø+(I-1)*9Ø
29Ø  IF ABS(Q(I,1))<=Z THEN 33Ø
3ØØ  PRINT "ERROR IN LAT/LONG",Q(I,1),"DEG
     REES",Q(I,2),"MINUTES"
31Ø  PRINT "TRY AGAIN"
32Ø  GOTO 23Ø
33Ø  IF Q(I,2)<Ø OR Q(I,2)>=6Ø THEN 3ØØ
34Ø  Q(I,1)=Q(I,1)+SGN(Q(I,1))*Q(I,2)/6Ø
35Ø  NEXT
36Ø  PRINT "DATE?        DD,MM,YY"
37Ø  INPUT DD,MM,YY
38Ø  IF YY<8Ø OR YY>99 THEN 44Ø
39Ø  IF MM<1 OR MM>12 THEN 44Ø
4ØØ  LY=1
41Ø  Y=YY-INT(YY/4)*4
42Ø  IF Y=Ø THEN LY=2
43Ø  IF DD>=1 AND DD<=E(MM,LY) THEN 455
44Ø  PRINT "ILLEGAL DATE: TRY AGAIN"
45Ø  GOTO 36Ø
455  D=B(YY-79)+DD
46Ø  FOR I=1 TO MM-1
47Ø  D=D+E(I,LY)
48Ø  NEXT
49Ø  M=36Ø/365.2422*D+EG-WG
5ØØ  V=M+36Ø/PI*EC*SIN(M/DR)
51Ø  L(1)=V+WG
52Ø  IF L(1)>=Ø THEN 55Ø
53Ø  L(1)=L(1)+36Ø
54Ø  GOTO 52Ø
55Ø  IF L(1)<36Ø THEN 57Ø
555  L(1)=L(1)-36Ø
56Ø  GOTO 55Ø
57Ø  L(2)=L(1)+Ø.985647
59Ø  IF L(2)>=36Ø THEN L(2)=L(2)-36Ø
61Ø  FOR I=1 TO 2
62Ø  Y=SIN(L(I)/DR)*COS(EP/DR)
63Ø  X=COS(L(I)/DR)
64Ø  IF X<>Ø THEN 68Ø
65Ø  IF Y>Ø THEN C(I,1)=9Ø
66Ø  IF Y<Ø THEN C(I,1)=27Ø
67Ø  GOTO 77Ø
68Ø  IF Y<>Ø THEN 72Ø
69Ø  IF X>Ø THEN C(I,1)=Ø
7ØØ  IF X<Ø THEN C(I,1)=18Ø
71Ø  GOTO 77Ø
72Ø  C(I,1)=ATN(Y/X)*DR
73Ø  IF Y>Ø THEN 75Ø
74Ø  C(I,1)=C(I,1)+18Ø
75Ø  IF X*Y>Ø THEN 77Ø
76Ø  C(I,1)=C(I,1)+18Ø
77Ø  C(I,1)=C(I,1)/15
```

5 of 6

```
 775  ZZ=SIN(EP/DR)*SIN(L(I)/DR)
 776  GOSUB 139Ø
 777  C(I,2)=AS*DR
 79Ø  X=SIN(C(I,2)/DR)/COS(Q(1,1)/DR)
 8ØØ  IF X>-1 AND X<1 THEN 822
 81Ø  FL=1
 812  GOTO 1262
 822  ZZ=X
 824  GOSUB 137Ø
 83Ø  F(I,1)=AC*DR
 84Ø  F(I,2)=36Ø-F(I,1)
 85Ø  X=-TAN(Q(1,1)/DR)*TAN(C(I,2)/DR)
 86Ø  IF X<-1 OR X>1 THEN 81Ø
 862  ZZ=X
 864  GOSUB 137Ø
 87Ø  H=AC*DR/15
 88Ø  T(1)=24+C(I,1)-H
 89Ø  T(2)=C(I,1)+H
 9ØØ  FOR J=1 TO 2
 91Ø  IF T(J)>24 THEN T(J)=T(J)-24
 92Ø  S(I,J)=T(J)
 93Ø  NEXT J
 935  NEXT I
 94Ø  FOR J=1 TO 2
 95Ø  T(J)=24.Ø7*S(1,J)/(24.Ø7+S(1,J)-S(2,J))
 96Ø  NEXT
 97Ø  DE=(C(1,2)+C(2,2))/2
 972  ZZ=SIN(Q(1,1)/DR)/COS(DE/DR)
 974  GOSUB 137Ø
 98Ø  PS=AC*DR
 99Ø  X=Ø.8356Ø8
1ØØØ  ZZ=TAN(X/DR)/TAN(PS/DR)
1ØØ2  GOSUB 139Ø
1ØØ4  DA=AS*DR
1Ø1Ø  ZZ=SIN(X/DR)/SIN(PS/DR)
1Ø12  GOSUB 139Ø
1Ø14  Y=AS*DR
1Ø2Ø  DT=24Ø*Y/COS(DE/DR)/36ØØ
1Ø3Ø  FOR J=1 TO 2
1Ø4Ø  T(J)=T(J)+(-1)↑J*DT
1Ø5Ø  FOR I=1 TO 2
1Ø6Ø  F(I,J)=F(I,J)+(-1)↑J*DA
1Ø7Ø  NEXT
1Ø8Ø  T(J)=T(J)-Q(2,1)/15
115Ø  DX=D*Ø.Ø657Ø98-17.37
117Ø  T(J)=T(J)-DX
1181  IF T(J)>=Ø THEN 1184
1182  T(J)=T(J)+24
1183  GOTO 1181
1184  IF T(J)<24 THEN 119Ø
1185  T(J)=T(J)-24
1186  GOTO 1184
119Ø  T(J)=T(J)*Ø.99727
1192  S%(J)=INT(T(J))
1193  T%(J)=(T(J)-S%(J))*6Ø+Ø.5
12ØØ  NEXT J
121Ø  FOR J=1 TO 2
122Ø  F(1,J)=(F(1,J)+F(2,J))/2-9Ø
123Ø  F(2,J)=F(1,J)+18Ø
124Ø  IF F(1,J)<Ø THEN F(1,J)=36Ø+F(1,J)
125Ø  IF F(2,J)>36Ø THEN F(2,J)=F(2,J)-36Ø
126Ø  NEXT
1262  CLS
1265  PRINT@Ø"*****************","GREG BAKER,
      MONGARLOWE, 2622"
1266  PRINT "GREYLINE CALCULATOR RESULTS:",
      "*****************"
127Ø  PRINT "LATITUDE",Q(1,1),"LONGITUDE",
      Q(2,1)
1271  PRINT "DATE:",DD;".";MM;".";YY
1272  IF FL=1 THEN 129Ø
1273  PRINT "SUNRISE",S%(1);":";T%(1);"UTC",
      "BEARINGS:        ";
1274  PRINT USING "###.#";F(1,1);
1275  PRINT USING "######.#";F(2,1)
1276  PRINT "SUNSET",S%(2);":";T%(2);"UTC",
      "BEARINGS:        ";
1277  PRINT USING "###.#";F(1,2);
1278  PRINT USING "######.#";F(2,2)
128Ø  PRINT,,,"ANOTHER QTH OR DATE?","TYPE
      'Y' TO CONTINUE"
1282  INPUT Y$
1284  IF Y$<>"Y" THEN 136Ø ELSE 23Ø
129Ø  PRINT "SUN DOES NOT RISE OR SET"
1291  PRINT "HENCE THERE IS NO GREYLINE"
1292  GOTO 128Ø
136Ø  END
137Ø  AC=-ATN(ZZ/SQR(1-ZZ*ZZ))+PI/2
138Ø  RETURN
139Ø  AS=ATN(ZZ/SQR(1-ZZ*ZZ))
14ØØ  RETURN
```

6 of 6.

VZ-200 BASIC PROGRAM STORAGE & LINE RENUMBERING

GRAHAM MARSDEN

The VZ-200 does not have a RENUMBER command so trying to modify a program with insufficient vacant line numbers is not a welcome task. This program enables the line numbers of a program to be reset using any start number and increment providing they meet certain conditons.

In order to understand the operation of the program it is necessary to understand how a BASIC program listing and its line numbers are stored in memory.

Each line of program is formated as below:-

- The first two bytes of the sequence hold the address, in two byte form, of the first byte of the sequence for the next program line. i.e. the location holding the R above is in location P+256*Q
- The third and fourth bytes hold the line number.
  i.e. in this case the line number will be L+256*I
- Then the contents of the program line follow, terminating with a byte containing the value zero.

For example suppose the line:-

300 PRINT"!":GOTO400

was stored starting at address 38420. This would be the contents of locations 38420 - 38433

Note that characters (including line numbers used within a program line after GOTO or GOSUB) are stored as their ASC codes.

"Operators" like PRINT, GOTO, etc have their own single byte codes which represent the operation. The program looks for the codes for GOTO and GOSUB (amongst others- see explanation of program operation) in order to find the locations of line numbers within program lines. The codes for various operations can be determined by putting in a line using the operation in question, (ensure it has the lowest of all line numbers) and then type in -

```
FORZ = 31469 TO 31469+N:PRINT
PEEK(Z);:NEXTZ
```

where N is the number of memory positions that you wish to see codes for. 31469 is the position in memory of the first item of the first BASIC program line. i.e. the one immediately after the line number bytes. The BASIC Program listing normally starts at 31465 unless moved - but that is another story.

Having understood how a BASIC program is stored it is possible to make changes to it without having to edit it on screen.

One thing that can be done is to change all the line numbers so they follow a constant incremant.

Here is a program to do just that:-

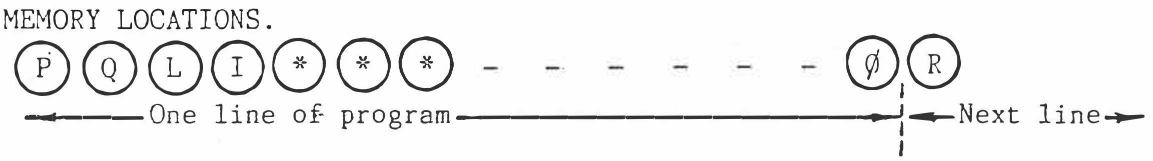
How to use this program:-

1) Type in and CSAVE as listed.
2) Before keying in your next program load the renumbering program from tape.
3) Key in your program with particular attention to the following.
   a) Line numbers used and called must be in the range 1-9999
   b) All line numbers or subroutines quoted within the text of a program line must be preceeded by GOTO or GOSUB and be right justified in a 4 space field. This means that 5 digit numbers if used will be seen only as the first four digits from the left and therefore will not be found as an existing number.

   i.e. IF ...THEN20ELSE325 must be entered as
   IF...THENGOTO 20ELSEGOTO 325
   (the line number 20 is preceeded by two spaces the number 325 by one, to create a 4 space field for the number - This allows say a two digit number to be reassigned as a three or four digit number)

4 space field for the number - This allows say a two digit number to be reassigned as a three or four digit number)

   c) Line 10010:-
   Dimmension N%() greater than the number of line numbers in the section of program to be renumbered - A generous guess will do unless you are short of memory.
   :- Set the value of variable S to create a "safe zone" which the renumbering program will not alter. Normally this value will be 10000 (the first line number of the renumbering program itself) or it may be less if you wish to create a "gap" in line numbers between two sections of program - say between a main operating section and another section containing subroutines or Data lines. Remember that nothing in the "safe zone" is altered to a GOTO or GOSUB calling a lower section renumbered line would have to be changed separately.
4) Always CSAVE BEFORE running this program. If for any reason the renumbering is not totally successful then what remains of your program will probably be useless as part will be renumbered and part will not - equivalent to a population explosion of bags.
5) Key in RUN10000
   (If the result is a BAD SUBSCRIPT ERROR IN 10090 then increase the size of N%() in line 10010 - reloading will not be necessary as nothing has been altered yet.)
6) Enter 1st line number and increment on prompt.
7) When the renumbering is complete the cursor will return and the computer will be in READY mode (The time to execute



38420	38421	38422	38423	38424	38425	38426	38427
34	15Ø	44	1	178	34	33	34
38434		300		Print	"	!	"

38428	38429	38430	38431	38432	38433	38434	38435
58	141	52	48	48	Ø		
:	Goto	4	0	0		Next Line	

1 of 3.

will be about 1½ seconds per program line)

8) Make any changes as indicated by messages printed during execution. (You can BREAK the execution to copy notes from the screen if it gets too full and then enter CONT to continue).

9) Probably a good idea to CSAVE again

10) Thoroughly test the renumbered section of the program - any problems - reload that saved at (4) and reRUN 10000 - it is not unknown for gremlins to be about for one renumbering run but absent for the next.

Program Operation

LINES 10000-10090

The line numbers are stored in array N%. Variable M is initialised to 31465 (BASIC Program start) and moved to the 1st byte of each program line by the calculation based on the values of P and Q the "start - position of next line" pointers. While M holds the decimal value of the memory positions the value of variable A is used in the PEEK'S. This is because PEEK and POKE will only work for the range -32768 A 32767 values for memory above 32767 must have 65536 subtracted before PEEK or POKE.
Line 10070 ends execution if P and Q both are zero - this occurs when the end-of-program byte sequence is found. (Two zero bytes then a 4 byte.

LINES 10100-10120

The first line number and increment are entered and edited so that they are both positive integers greater than zero. At 10120 a check is made as to whether the new numbering reach into the "safe zone" of line numbers.

LINE 10130

Reinitialise M back to program start and variable c to 1, C is the number of the line ie 1st 2nd 3rd etc.

LINES 10140-10180

Calculate new line number and POKE new line number to the appropriate bytes. M is now at the first byte of the line's storage sequence. The execution ends at 10160 if the "safe-zone" is found.

LINES 10190-10210

Calculate R the memory position of the start of the next line.

LINES 10220-10290

This section searches through the program line contents looking at the value in each byte:

0: - end of line - go to next line
34: - Quote marks in a PRINT or INPUT statement - ignore
136: - DATA - ignore whole line
147: - REM - ignore whole line
141: - GOTO or 145 GOSUB: - alter line number called.

LINES 10300-10380

M set to the units column position of the number field area. The string variable 0$ is loaded with the characters of the line number field, and variable 0 is given the value of the

```
10000 '"VZ-200  LINE RENUMBERING G.A MAR
SDEN 1984
10010 DIM N%(200):M=31465:C=1:S=10000'"F
IRST LINE OF SAFE ZONE"
10020 A=M:IFA>32767 THEN A=A-65536
10030 P=PEEK(A):A=M+1:IF A>32767 THEN A=
A-65536
10040 Q=PEEK(A):A=M+2:IFA>32767 THEN A=A
-65536
10050 L=PEEK(A):A=M+3:IF A>32767 THEN A=
A-65536
10060 I=PEEK(A)
10070 IF P=0ANDQ=0THEN PRINT "SAFE LINE
PAST END  ":END
10080 IFL+256*I>=STHEND=C-1:PRINTD;"LINE
S FOUND":GOTO10100
10090 N%(C)=L+256*I:M=P+256*Q:C=C+1:GOTO
10020
10100 INPUT"1ST LINE N0.";L$:F=INT(VAL(L
$)):IF F<1 THENGOTO10100
10110 INPUT"INCREMENT";I$:J=INT(VAL(I$))
:IFJ<1THENJ=1
10120 IF F+J*(C-1)>=STHEN PRINT" VALUES T
OO LARGE":GOTO10100
10130 M=31465:C=1
10140 A=M+2:IFA>32767THENA=A-65536
10150 B=M+3:IF B>32767A=A-65536
10160 IF PEEK(A)+256*PEEK(B)>=STHEN PRIN
T"FINISHED":END
10170 W=F+(C-1)*J
10180 POKEA,W-256*INT(W/256):POKEB,INT(W
/256)
10190 A=M:IFA>32767 THENA=A-65536
10200 B=M+1:IFB>32767 THENB=B-65536
10210 R=PEEK(A)+256*PEEK(B)
10220 M=M+4:T=1
10230 A=M:IF A>32767 THEN A=A-65536
10240 IF PEEK(A)=136OR PEEK(A)=147 THEN
M=R:C=C+1:GOTO10140
10250 IFPEEK(A)=145OR PEEK(A)=141 THEN G
OTO10300
10260 IF PEEK(A)=34THEN T=T*-1
10270 IF PEEK(A)=O THEN M=R:C=C+1:GOTO10
140
10280 M=M+1
10290 IFT<O THENGOTO10260ELSE GOTO10230
10300 M=M+4
10310 O$=""
10320 FORG=3TO0STEP-1
10330 A=M-G:IFA>32767THENA=A-65536
10340 O$=O$+CHR$(PEEK(A))
10350 NEXTG
```

2 of 3.

line number. Lastly a check to see if the line number called is within the safe zone.

LINES 10390-10470

The position of the old line no 0 is found in the arrayN%() H% is the position in the array that the value of 0 is compared with, in line 10470, and is initialised at the middle of the occupied area of the array. K% is initialised at just over¼ of the occupied length of the array. K% is reduced to just over ½ its value at each loop and H% is altered by adding or subtracing K% depending on which direction the search for the line number must go. The values of the last two array positions looked at are held in HL% and HP%. If a second look is taken at any array position then the conclusion is that the number does not exist and the error message at 10460 is printed and the search for another GOTO or GOSUB resumes at 10230. This search routine takes only 5 or 6 loops to find a number in an array of 70 line numbers and is therefore more efficient than just starting at the bottom and looking at each array position on the way up, which would take an average 35 loops to find a line number Of course it relies on the fact that the numbers are stored in numerical order. The tests at 10430 and 10440 are to see if the search has gone beyond the occupied range of the array and modify H% and K% accordingly. This routine is useful to look through any array of values providing they are in number order. (ascending or descending).

LINES 10480-10510

String variable NN$ is set to the characters of the new line number (including spaces) to be inserted in the 4 byte field of the program line. Line 10500 allows the start positions of subroutines to be recorded as the renumbering goes on. This line could be ommited and the start of subroutines marked in the program listing using REM" or "". The " allows the remarks to be put in inverse characters to make them stand out as the program zooms up the screen after LIST.

LINES 10520-10620

This segment ensures that the value of 0 is consistant with the number of spaces at the left of the 4 byte field that the number came from.

LINES 10630-10670

Character by character POKEing of the new line number to its position in the line format.

```
10360 O=VAL(O$)
10370 IFO>=STHEN PRINT"LINE";W;"(NEW):-"
;OELSE GOTO 10430
10380 PRINT"[TWELVE SPACES]---INSIDE SAF
E ZONE:M=M+1:GOTO10230
10390 HP%=0:HL%=0
10400 H%=1+D/2:K%=D/2
10410 K%(K%+1)/2:HP%=HL%:HL%=H%
10420 H%=H%+SGN(0-N%(H%))*K%
10430 IFN%{H%)=OTHENH%=H%-K%:K%=1
10440 IF H%<1THENH%=1:K%=1
10450 IFH%=HP%THEN GOTO10460ELSEGOTO1047
0
10460 PRINT"LINE";W;"(NEW):- ";O;"NOT FO
UND":M=M+1:GOTO10230
10470 IFN%(H%)<>OTHENGOTO10410
10480 NN$="[3 SPACES]"+STR$(F+(H%-1)*J)
10490 A=M-4:IFA>32767 THENA=A-65536
10500 IF PEEK(A)=145THEN PRINT"NEW SUBR@
 ";NN$"CALLED@";W
10510 NN$=RIGHT$(NN$,4)
10520 IFO>=1000THENGOTO10630
10530 A=M-3:IFA>32767THENA=A-65536
10540 IF PEEK(A)<>32THENGOTO10610
10550 IF O>=100THENGOTO10630
10560 A=M-2:IFA>32767THENA=A-65536
10570 IFPEEK(A)<>32THENGOTO10610
10580 IFO>=10THENGOTO10630
10590 A=M-1:IFA>32767THENA=A-65536
10600 IF PEEK(A)=32THENGOTO10630
10610 PRINT"LINE";W;"(NEW):FIELD ERROR";
O
10620 PRINT"[4 SPACES]---CHANGE TO NEW N
O.: ";NN$:GOTO10670
10630 FORG=1TO4
10640 A=M-4+G:IFA>32767THENA=A-65536
10650 POKEA,ASC(MID$(NN$,G,1))
10660 NEXTG
10670 M=M+1:GOTO10230
```

3 of 3.

FIND

By Chris St~~r~~amboulidis

see update by
Larry Taylor
LE V2 #

Find is a machine language routine which searches your Basic program for lines which contain a specified string up to 16 characters in length. The routine is quite short (only 117 bytes) and will work with any size VZ because it resides in an unused section of the communications region.

There are two methods of entering Find into your machine: if you have an Editor Assembler, simply type in Listing 1, set the origin to 7A28H/31272, assemble and dump the object code to tape under the name 'FIND.OBJ'. When you CLOAD or CRUN the tape, the routine will autorun and immediately return you to the 'READY' prompt.

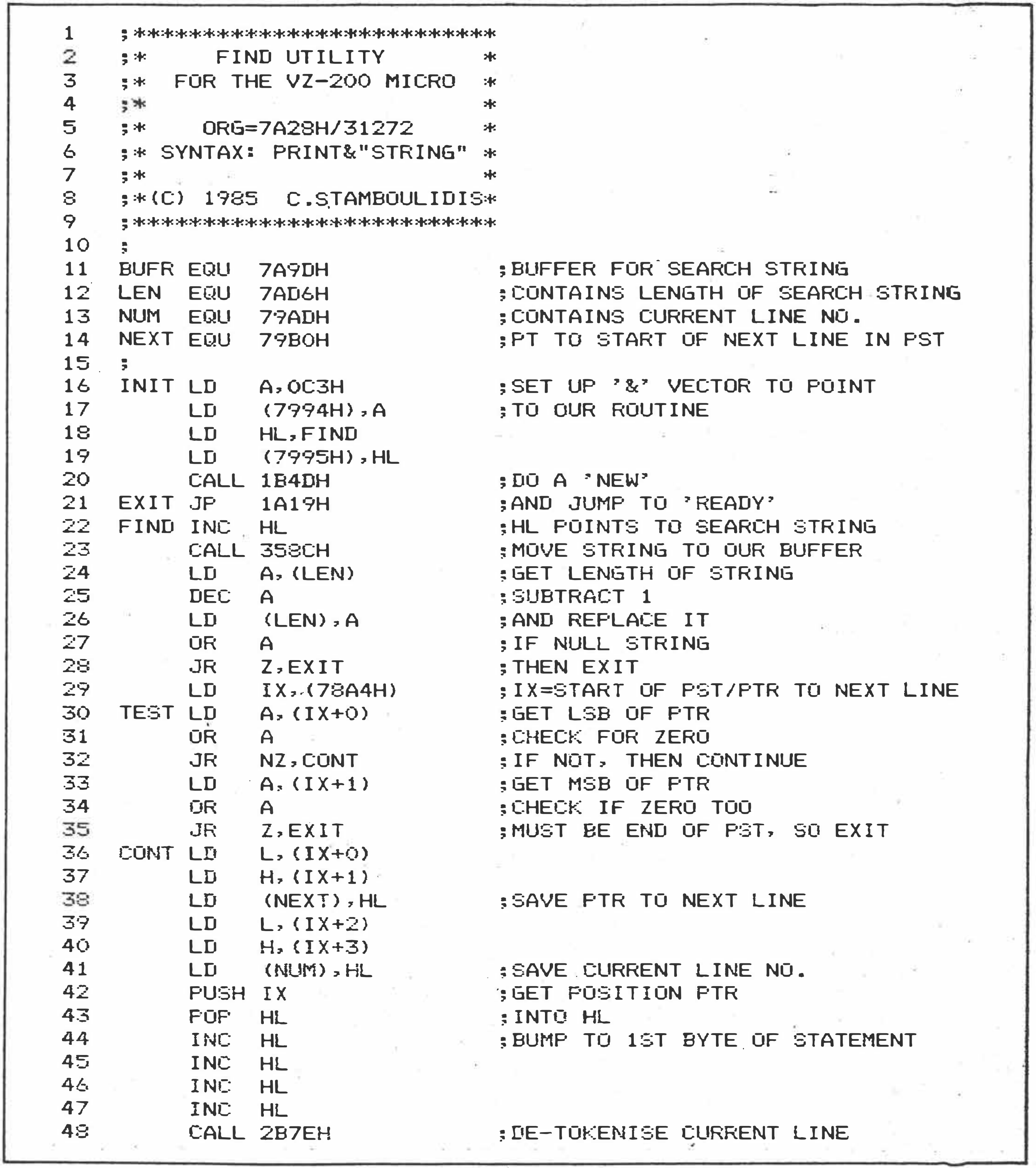
The other method is to type in Listing 2, which will POKE the machine code instructions into place for you and will do all the initialisation. In this case, make sure that you CSAVE a copy of 'FIND.BAS' before you try to RUN it. To save you typing it all in again if it crashes for any reason, such as a wrong number in the data statements. A checksum is used to make sure that all these numbers add up, but this doesn't prevent numbers being placed in the wrong order. When you RUN the loader, it should only take a couple of seconds to do its job and then return you to 'READY'. The Basic loader will have been NEWed and you're ready to go.

To use Find, simply enter the following as a direct command:

PRINT&"string"
or
?&"string"

with the string to search for in between the quotes. The line numbers of the lines which contain the search string will then be printed on the screen for you. Note that leading spaces in the search string are ignored and so the routine cannot search for spaces, eg PRINT&" " would be interpreted as a null string and would not be searched for.

PCG Apr 85 62-64
1 of 3.



```
;*************************
;*      FIND UTILITY       *
;*  FOR THE VZ-200 MICRO   *
;*                         *
;*     ORG=7A28H/31272     *
;* SYNTAX: PRINT&"STRING"  *
;*                         *
;*(C) 1985  C.STAMBOULIDIS*
;*************************
;
BUFR EQU  7A9DH             ;BUFFER FOR SEARCH STRING
LEN  EQU  7AD6H             ;CONTAINS LENGTH OF SEARCH STRING
NUM  EQU  79ADH             ;CONTAINS CURRENT LINE NO.
NEXT EQU  79B0H             ;PT TO START OF NEXT LINE IN PST
;
INIT LD   A,0C3H            ;SET UP '&' VECTOR TO POINT
     LD   (7994H),A         ;TO OUR ROUTINE
     LD   HL,FIND
     LD   (7995H),HL
     CALL 1B4DH             ;DO A 'NEW'
EXIT JP   1A19H             ;AND JUMP TO 'READY'
FIND INC  HL                ;HL POINTS TO SEARCH STRING
     CALL 358CH             ;MOVE STRING TO OUR BUFFER
     LD   A,(LEN)           ;GET LENGTH OF STRING
     DEC  A                 ;SUBTRACT 1
     LD   (LEN),A           ;AND REPLACE IT
     OR   A                 ;IF NULL STRING
     JR   Z,EXIT            ;THEN EXIT
     LD   IX,(78A4H)        ;IX=START OF PST/PTR TO NEXT LINE
TEST LD   A,(IX+0)          ;GET LSB OF PTR
     OR   A                 ;CHECK FOR ZERO
     JR   NZ,CONT           ;IF NOT, THEN CONTINUE
     LD   A,(IX+1)          ;GET MSB OF PTR
     OR   A                 ;CHECK IF ZERO TOO
     JR   Z,EXIT            ;MUST BE END OF PST, SO EXIT
CONT LD   L,(IX+0)
     LD   H,(IX+1)
     LD   (NEXT),HL         ;SAVE PTR TO NEXT LINE
     LD   L,(IX+2)
     LD   H,(IX+3)
     LD   (NUM),HL          ;SAVE CURRENT LINE NO.
     PUSH IX                ;GET POSITION PTR
     POP  HL                ;INTO HL
     INC  HL                ;BUMP TO 1ST BYTE OF STATEMENT
     INC  HL
     INC  HL
     INC  HL
     CALL 2B7EH             ;DE-TOKENISE CURRENT LINE
```

Apr 85   2 of 3

```
49         LD   DE,79E8H            ;DE= LOCATION OF EXPANDED LINE
50   PRE   LD   A,(LEN)             ;GET LENGTH OF SEARCH STRING
51         LD   B,A                 ;INTO B
52         LD   HL,BUFR-1           ;HL= BYTE BEFORE STRING BUFFER
53   SCAN  INC  HL                  ;PT TO NEXT BYTE IN STRING
54         LD   A,(HL)              ;CHECK IF END OF STRING
55         OR   A
56         JR   Z,EXIT              ;IF SO, THEN WE'RE DONE
57         LD   A,(DE)              ;DE= BYTE FROM STATEMENT LINE
58         OR   A                   ;CHECK FOR END OF LINE
59         JR   Z,MORE              ;IF SO, THEN PROCESS NEXT LINE
60         INC  DE                  ;DE= NEXT BYTE IN STATEMENT
61         CP   (HL)                ;CHECK IF SAME AS STRING BYTE,
62         JR   NZ,PRE              ;IF NOT, THEN TRY NEXT BYTE
63         DJNZ SCAN                ;CONTINUE UNTIL ALL BYTES FOUND
64         LD   A,20H               ;MUST BE ALL THERE, SO
65         CALL 33AH                ;PRINT A SPACE
66         LD   HL,(NUM)            ;AND PRINT THE
67         CALL OFAFH               ;CURRENT LINE NO.
68   MORE  LD   IX,(NEXT)           ;IX= PTR TO NEXT LINE IN PST
69         JR   TEST                ;BACK TO CHECK NEXT LINE


ERRORS : 00000
BYTES FREE :- 10288
```

LISTING 2

```
100 '**************************************************************
110 '*                        FIND.BAS                            *
120 '*               FIND UTILITY FOR THE VZ-200 MICRO            *
130 '*            ORG=7A28H/31272    SYNTAX: PRINT&"STRING"       *
140 '*    NB. STRING LENGTH MUST BE 16 CHARACTERS OR LESS         *
150 '*              (C) 1985  CHRIS STAMBOULIDIS                  *
160 '**************************************************************
170 '
180 POKE30862,40:POKE30863,122     'SET UP USR JUMP TO INITIALISE
190 FORI=31272TO31388:READJ:C=C+J:POKEI,J:NEXT    'SET UP ROUTINE
200 IFC<>13013PRINT"CHECKSUM ERROR":STOP     'ERROR IN DATA LINES
210 X=USR(0)                                  'GO INITIALISE ROUTINE
220 END
230 DATA 62,195,50,148,121,33,57,122,34,149,121,205,77,27
240 DATA 195,25,26,35,205,140,53,58,214,122,61,50,214,122
250 DATA 183,40,239,221,42,164,120,221,126,0,183,32,6,221
260 DATA 126,1,183,40,223,221,110,0,221,102,1,34,176,121
270 DATA 221,110,2,221,102,3,34,173,121,221,229,225,35,35
280 DATA 35,35,205,126,43,17,232,121,58,214,122,71,33,156
290 DATA 122,35,126,183,40,180,26,183,40,17,19,190,32,236
300 DATA 16,241,62,32,205,58,3,42,173,121,205,175,15,221
310 DATA 42,176,121,24,174
```

Apr 85   3 of 3.

Yahtzee dice loaded!

With reference to Tumbling Dice by Ron Roberts in the November issue of *APC* I became suspicious of its "fairness" when Yahtzees with ones or sixes seemed almost impossible. Testing the random number expression used [R=INT(RND(1)*5+1.3] I found the probability of getting a one or a six half the probability of getting either 2, 3, 4 or 5. The following program verifies this claim:

```
 10  DIM N(6)
 20  FOR I=1 TO 6 :
     N(I)=0 : NEXT I
 30  PRINT
 40  FOR I=1 TO 1000
 50  R=INT (5*RND(1)
     +1.5)
 60  N(R)=N(R) + 1
 70  NEXT I
 80  FOR T=1 TO 6
 90  PRINT T "---" N(T)
100  NEXT
```

May I suggest the more correct formula R=INT(6*RND(1)+1) for a fair game.

W Holland

APC. Apr 85 6(4): 19.

VZ VARIABLE DEFINITION

The statements DEFINT, DEFSNG, DEFDBL and DEFSTR are not implemented in VZ-200 Basic (although the code for these is in ROM). A way of simulating these statements, without having to write great chunks of assembler, is to make use of the Variable Declaration Table located between 30977 and 31002 (7901-791AH).

The VDT is 26 bytes in length, one for each letter of the alphabet. Each location contains a code defining the status of variables beginning with each letter:

2 — integer
3 — string
4 — single precision
8 — double precision

On power up and whenever a program is RUN, the whole of the VDT is initialised to single precision (ie, each location contains a 4).

The values in the VDT may be altered to define different variable types. For example, if you wanted to define all A to Z variables as integers, you would put the following code at the start of your program:

```
10  FOR I = 30977 TO
      31002 : POKE I,2 :
      NEXT
```

This is equivalent to the 'DEFINT A-Z' statement in Level II Microsoft Basic.

Alternatively, the following formula could be used to define individual variables:

```
10  POKE 30912 +
      ASC("Q"),3
```

(This would define Q as a string as in 'DEFSTR Q'.)

Note that Basic will not accept double precision variables as counters in FOR-NEXT loops. Also note that it is no longer necessary to use a suffix of '$' or '%' after a string or integer variable has been defined.

C Stamboulidis

APC Apr 85 6(4): 103.

VARIABLE VZ GOTO

The following routine eliminates those massive if then lists like:

IFA=10THEN100
IFA=20THEN110
IFA=30THEN120
etc.

After calling the routine, the variable 'GT' holds the value of the line to GOTO

To use, simply compute your line number to GOTO (or GOSUB) and having computed GT simply GOTO or GOSUB 2

F Olsen

```
0 GOTO1000
1 GOTO XXXX:' MUST LEAVE SPACE AND DO NOT ALTER FIRST TWO LINES
2 T$=STR$(GT)
3 T=LEN(T$):IFT<6THENT$=T$+CHR$(32)+T$:GOTO3
4 FORC=2TO6:POKE31478+C,ASC(MID$(T$,C,1)):NEXT:GOTO1
```

APC Apr 85 6(4):95.

The 'Variable VZ GOTO' in April *APC* does not work due to an error in line 3. Here is a revision that does:

```
0  GOTO1000
1  GOTO12345
2  T$=STR$(GT)
3  IFLEN(T$) <6THENT$=
   T$+" " :GOTO3
4  FORC=2TO6 :POKE
   31478+C,ASC(MID$
   (T$,C,1)
)  :NEXT :GOTO1
```

The GOTO in line 0 can be any four digit number. If you want to start your main program at a line numbered less than 1000, then use zeroes to make up the four digits. For example:

0 GOTO0058

To test the routine, enter these lines:

```
95  LIST-1000
1000  GT=95,GOTO2
```

and RUN.

For a variable list, which can be useful when debugging a program, simply change line 1 to:

1 LIST12345

APC Jul 85 6(7):176.

Hobson

Lonely hearts club

In reply to the letter "Basic Understanding" printed in the February edition of *APC*, I would like to commend S Hopson on the stand he has taken for the sharing of program knowledge. The computer which he uses as an example, the VZ-200, has been greatly disadvantaged by its marketing being limited to Australasia. This has meant that there are very few books and other publications for it. The programs printed in magazines such as *APC* are among the few sources available for programming knowledge for this and many other home computers.

It does seem a pity that more programmers do not comment on or explain the various routines used in their programs. However, computer novices should not despair. LYSCo print a newsletter for the VZ-200/300, the Amstrad CPC-464 and the Commodore 16 and Plus/4. In the newsletter we print a host of hints and tips sent in by its readers and programmers. Entire program listings are printed in some editions and we endeavour to answer questions asked by the readers. These letters are completely free to people on our mailing list. Anyone wishing to receive the newsletter should send a large stamped addressed envelope to LYSCo, PO Box 265, Bunbury, WA 6230 specifying the computer they own.

L Young

APC May 85 6(5)
p 52-53.

VZ200 VIDEO HARDWARE INTERRUPT

Steve Olney

This article details how to use the video hardware interrupt on the VZ200 and gives three simple examples of its usefulness.

THE HARDWARE INTERRUPT is a very useful feature of a computer's capability, with many different applications. The usefulness comes from the ability to 'interrupt' the normal flow of software execution, diverting the operation of the CPU by external means. The CPU can then be made to execute a separate, independent program before returning to the original program execution.

This description may sound like a GOSUB call to a subroutine in Basic, or a CALL to a subroutine in a machine code program, but there is an important difference. The difference is that the interrupt can occur asynchronously to the normal program execution (that is, it can occur at any time unrelated to the progress of normal program execution).

This capability is extremely useful when the computer has to serve some external device which can't wait for an action by the computer during normal program execution. Such devices range from a digital-to-analogue converter (which must sample data at strictly regular intervals), to a software clock counter which needs to be incremented by an external hardware clock pulse. By using a hardware interrupt these devices can be served almost immediately, in the time it takes the CPU to complete the current instruction.

The interrupt is called a hardware interrupt because there is a special pin on the CPU chip itself, which, when taken to ground potential (low or zero), initiates the interrupt sequence. This action is also performed by some external hardware device.

The VZ200 uses a Z80 CPU chip, which has three different responses to this interrupt signal depending on the interrupt mode set in the internal interrupt register (IR). Note that we are talking about the $\overline{\text{INT}}$ case, not the $\overline{\text{NMI}}$). For the VZ200 the interrupt register is set to interrupt mode 1 (by an IM1 instruction) during the initialization sequence.

The response to an interrupt in Interrupt Mode 1 is to complete the current instruction, save the program counter register (PCR) contents on the stack (allowing resumption of execution at that point upon returning from the interrupt) and then jump to location 0038 HEX. This could be viewed as a hardware version of the software RST 38 instruction.

The VZ200 video interrupt

Those of you who have access to a circuit diagram of the VZ200 will see that the interrupt pin (pin 16 $\overline{\text{INT}}$) of the Z80 CPU is connected to pin 37 ($\overline{\text{FS}}$) of the 6847 video controller chip. Reference to the 6847 data sheets shows that pin 37 of the 6847 chip is the video field sync output pin. This pin is pulled low by the 6847 chip during the vertical retrace period of the video output signal. That is, the field sync output pin goes low every 1/50 of a second (video frame rate of 50 per second) causing the Z80 CPU to be interrupted and diverted to location 0038 HEX every 20 ms.

Scrutiny of the machine code (in ROM) at location 0038 HEX reveals a JUMP instruction to location 2EB8 HEX. This jump is referred to as interrupt vector.

The machine code at 2EB8 HEX contains several CALLs to various locations before returning to the original program execution. I haven't looked at these in detail, but most likely they are concerned with cursor control and perhaps screen scrolling during listing.

In any case, the code in which we are interested is near the start of the code at 2EB8 HEX. The first CALL after saving affected registers is to location 787D HEX. There are two interesting points to note here. The first is that location 787D HEX is in RAM, and secondly, this is the memory location referred to in the VZ200 Technical Manual (under System pointers) as the "interrupt exit".

By PEEKing location 787D HEX (eg ▶

1 of 3

LISTING 1

```
HEX CODE        MNEMONIC
F5              PUSH AF           ; Save 'AF' register because we alter it
3E 2A           LD   A,2AH        ; Load 'A' register with code for '*'
32 1F 7Ø        LD   (7Ø1FH),A ; Put it in the top right-hand corner of screen
F1              POP AF            ; Restore 'AF' register
C9              RET               ; Return
```

LISTING 2

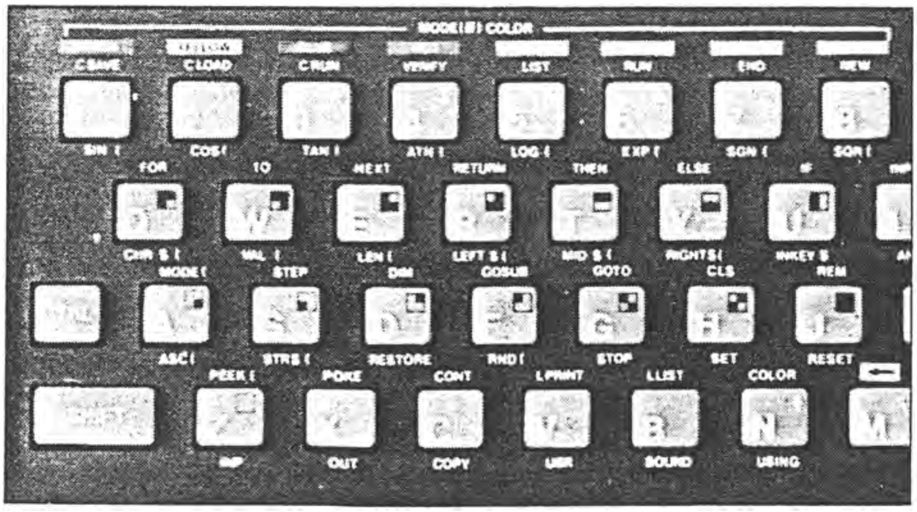
```
1ØØ  S= -32768 : F = S + 7 :' START AT 8ØØØ HEX
2ØØ  FOR I = S TO F        :' POKE THE 8-BYTE MACHINE CODE PROGRAM
3ØØ   READ D               :' INTO MEMORY STARTING AT 8ØØØ HEX
4ØØ   POKE I,D             :'
5ØØ  NEXT I                :'
6ØØ  POKE 3Ø846,ØØ         :' ENTER THE START ADDRESS OF THE MACHINE
7ØØ  POKE 3Ø847,128        :' CODE PROGRAM INTO INTERRUPT JUMP
8ØØ  POKE 3Ø845,195        :' EXIT AT 787D HEX.
9ØØ  DATA 245,62,42,5Ø,31,112,241,2Ø1:' 'DECIMAL EQUIVALENT OF HEX
```

LISTING 3

```
HEX CODE        MNEMONIC
F5              PUSH AF           ; save registers
C5              PUSH BC           ; we destroy
E5              PUSH HL           ;
3A 3B 78        LD   A,(783BH) ; load latch contents
Ø6 Ø8           LD   B,8          ; bit counter
21 18 7Ø        LD   HL,7Ø18H  ; start of screen display
17         LOOP RLA               ; rotate into carry and test
3Ø Ø7           JR   NC,ZERO   ;
36 31           LD  (HL),31H   ; output '1'
23              INC  HL           ; adjust to next display position
1Ø F8           DJNZ LOOP         ; go until all bits are done
18 Ø5           JR   EXIT         ; exit if done
36 3Ø      ZERO LD   (HL),3ØH  ; output 'Ø'
23              INC  HL           ; adjust to next screen position
1Ø F1           DJNZ LOOP         ; go until all bits are done
E1         EXIT POP  HL           ; exit
C1              POP BC            ;
F1              POP AF            ;
C9              RETURN            ;
```

LISTING 4

```
1ØØ  S= -32768 : F = S + 29 : 'START AT 8ØØØ HEX
2ØØ  FOR I = S TO F           :' POKE THE 8-BYTE MACHINE CODE PROGRAM
3ØØ   READ D                  :' INTO MEMORY STARTING AT 8ØØØ HEX
4ØØ   POKE I,D                :'
5ØØ  NEXT I                   :'
6ØØ  POKE 3Ø846,ØØ            :' ENTER THE START ADDRESS OF THE MACHINE
7ØØ  POKE 3Ø847,128           :' CODE PROGRAM INTO INTERRUPT JUMP
8ØØ  POKE 3Ø845,195           :' EXIT AT 787D HEX.
9ØØ  DATA 245,197,229,58,59,12Ø,6,8
1ØØØ DATA 33,24,112,23,48,7,54,49
11ØØ DATA 35,16,248,24,5,54,48,35
12ØØ DATA 16,241,225,193,241,2Ø1
```



PRINT PEEK[30845]) you should find it contains 201 DECIMAL (0C9 HEX) which is the Z80 RETurn instruction.

Using the video interrupt

Let's just back up to summarize what we've discussed so far. Every 20 ms the Z80 CPU is interrupted by the 6847 video controller chip. The interrupt mode (mode 1) causes the Z80 to jump to location 0038 HEX. From here execution jumps to 2EB8 HEX where a CALL to 787D HEX is encountered. Location 787D HEX (in RAM) contains a RET instruction and so execution returns immediately and continues until 2EDA HEX where a return from interrupt instruction (RETI) is found. Execution is now RETurned to the original program flow.

Now, because location 787D HEX is in RAM, we can change the RET instruction at that location to a JUMP to some other selected location. At this location we can insert our own interrupt servicing code.

Here is a very simple example to illustrate this procedure. Starting at location 3450 HEX in the Basic ROM is a subroutine which generates the 'beep' whenever you press a key. We can alter location 787D, 787E and 787F HEX to contain a JUMP to 3450 HEX to execute this 'beep' routine every time a video interrupt occurs (every 20 ms).

To do this we POKE the following machine code into memory starting at location 787D HEX:

Hex Code	**Mnemonic**
C3 50 34	JP 3450H

Note: Remember location 787D HEX is CALLed every 20 ms, so you must not alter the RET at this location until you have entered a valid jump address in the following two bytes. Otherwise the Z80 will jump to some indeterminate address depending on what random data was contained in 787E and 787F HEX.

The following strict order should be used:

POKE 30846,80 (POKE 50 HEX into location 787E HEX)
POKE 30847,52 (POKE 34 HEX into location 787F HEX)
POKE 30845,195 (POKE C3 HEX into location 787D HEX)

Type in the above commands via the immediate mode (without line numbers). The text within the brackets should *not* be typed in as it is for information only.

Once you have done this you should hear an almost continuous beep from the internal speaker. Notice that there is nothing which interferes with this beeping. Well, almost nothing, as will be explained a little later. However, you can enter a Basic program as normal (except for the distraction of the beeping) and even RUN or LIST it. In fact, you can do all the normal operations (ex-

2 of 3.

cept tape operations — see below) without affecting the beeping. This is because the interrupt has priority over other software execution. So we see it is possible to have a Basic program running in the 'foreground' with a separate machine language program running in the 'background' being executed at regular intervals.

To stop the beep all that is necessary is to change the JUMP instruction (0C3 HEX) at location 787D HEX back to a RET (0C9 HEX) by:

POKE 30845,201

Tape operations

As mentioned earlier, there is another action which will disable the 'beep'. During tape operations, interrupts are disabled to ensure that accurate timing delays in the tape function's machine code are not disturbed. So while you are CSAVEing, CRUNning or CLOADing data to or from tape the beeping will stop. However, once the operation is over the interrupts are enabled once again and the beeps return.

To enable the 'beep' again, enter —

POKE 30845,195

Note: Before typing the above, make sure that locations 787E and 787F HEX contain the correct jump address (3450 HEX)!

Non erasable video display

Next we'll look at an example which shows how the video interrupt can be used to put 'non-erasable' information on the video screen.

Normally, any information displayed on the screen can be overwritten, cleared or scrolled off the screen, either during program execution or in the immediate execution mode. By using the video interrupt you can display information which cannot be overwritten.

The machine language source code is shown in Listing 1.

Use the Basic program shown in Listing 2 to enter and then to enable the machine code program shown in Listing 1.

After you have entered Listing 2, CSAVE it before RUNning it. You should see an '*' in the top right-hand corner of the screen. Try to erase this by any means you like and you will find the best you can do is to erase it momentarily (in fact a maximum of approximately 20 ms, the time taken between successive interrupts). The only way to erase the '*' is to disable the interrupt itself, or to disable the machine code program by:

POKE 30845,201

which POKEs a RET instruction (0C9 HEX) back into location 787D HEX.

Real-time system pointer display

When programming in Basic a useful feature would be to see a constantly updated display of various system pointers (eg start of program, end of program, start of free space etc) to aid in keeping track of the progress of these parameters.

To illustrate this principle simply, we will display the contents of the output latch. A copy of the latch contents is maintained at location 783B HEX (307779 decimal). The latch controls the following:

BIT	FUNCTION	0	1
0	speaker O/P #1	see note below	
1	unused		
2	cassette O/P	toggles according to data O/P	
3	mode control	Mode 0	Mode 1
4	background colour	green	buff
5	speaker O/P #2	see note below	
6	unused		
7	unused		

Note: During a key press 'beep' or execution of the SOUND command, the software toggles bit 0 and bit 5. When it does this, it first looks at the state of each bit and then inverts that state. Normally each bit (0 and 5) are the complement of each other, and the inversion of both at the same time gives a 'push-pull' like drive signal to the speaker. However, if both bits were the same, there would be no differential change when they are inverted, and so no output. You can therefore disable the 'beep' and the SOUND command by looking at both bits and then POKEing a value into location 783B HEX (30779 decimal) which makes them equal. That is, if the contents of 783B HEX are even, then POKE back into 783B HEX a value equal to (contents + 1). Conversely, if the contents are odd, POKE back a value of (contents − 1).

Getting back to the latch display — to indicate the state of each bit, we will display a '0' or '1' for each bit in the top right-hand corner of the screen.

The machine language source code is shown in Listing 3.

The Basic program in Listing 4 will enter and enable the machine code program of Listing 3. Note that Listing 4 is similar to Listing 2, so if you have already entered Listing 2 you can modify it to Listing 4. Once again, enter the Basic program (Listing 4), and CSAVE it before RUNning it. You should see the contents of the output latch displayed in binary in the top right-hand corner of the screen, reading from left to right, starting with bit 7 across to bit 0. Change the background colour (COLOR,0 and COLOR,1) and note the change in bit 4 in the display.

Cursor position pointer

Edit line number 900 to:

900 DATA 245,197,229,58,166,120,6

ReRUN the program.

This will display the horizontal cursor position pointer (0-31) from location 78A6 HEX (30886 decimal). Use the left/right cursor position arrows to move the cursor and observe the display.

Basic program pointers

Now edit line number 900 to:

900 DATA 245,197,229,58,249,120,6

ReRUN the program again.

This will display the LSB (Least Significant Byte) of the 'end of Basic program' pointer. Try adding extra lines to the Basic program and note the change in the display. For example, add the line:

1500 REM TEST

Note down the binary value displayed and then edit line 1500 to:

1500 TEST

Compare the new display value with the previous value.

This exercise reveals that although the short form remark symbol (') occupies two *screen* spaces *less* than the long form REM command, it needs two *more program memory* spaces to store it than the long form!

What next?

These given examples are very simple ones designed to illustrate the basic principle of using the video interrupt and do not show the full potential of the technique. I have written two programs which utilize this technique in a more complex fashion. The first of these is a real-time clock which is controlled by the internal clock of the VZ200. This gives a digital readout display in the upper right-hand corner of the screen. The real-time clock is implemented entirely in software (no need for extra hardware or modifications).

The second program demonstrates a split-screen graphics mode with one part of the screen having text and lo-res graphics, with the remainder in hi-res graphics.

Other applications

These are but a few of the many possible uses of the video interrupt. Other applications include:

- arcade games — synchronizing movement with the video raster rate to give smooth action. Mixed hi-res graphics and text for scoring, simulating instrumentation etc;
- stopwatch — event timer or lap-scorer;
- frequency counter — using the internal VZ200 clock to give the timing gate period; and
- real-time control — using the VZ200 as a component in a control system, eg burglar alarm.

The list could go on, as anything which requires a reasonably accurate time-keeping function or synchronization with the video display, is a possible candidate. Which all goes to show that it's not always rude to interrupt! ●

3 of 3.

```
30862, 241      F1H
———— 3, 143     8FH.
     START add.   8FF1 H

-28687 ≡ 36849   ≡ 8FF1 }
-28674 = 36862   ≡ 8FFE } 14 bytes
```

Disassembled listing

```
21 00 70   LD HL, 7000H   ; #28672D Start video
11 01 70   LD ~~DE~~, 7001 H  ; #28673D Next
01 FF 07   LD BC, 07FFH   ; #2047D Size
36 55      LD (HL), 55 H  ; #85D Yellow
ED B0      LDIR           ; repeat until BC=0
C9         RET
```

VZ-200 instant colour

This short machine code routine will turn the screen the colour you have put in the data — instantly!!

To call the machine code routine type X=USR (0) where needed in your program.

To get different colours you change the underlined number in the data.

The numbers for the different colours are:

0=GREEN	170=BLUE
85=YELLOW	255=RED

A Willows

```
00010 FORI=-28687 TO -28674
00020 READA:POKEI,A
00030 NEXT
00040 DATA33.0.112,17,1,112,1
      ,255,7,54,85,237,176,201
00050 POKE30862,241:POKE30863,143
```

Page 130 Australian Personal Computer

V.6(8): Aug 85.

```
LDI assign (HL) to (DE) ]
      inc HL            ] Block
      inc DE            ] Move.
      dec. BC           ]
      Repeat until BC=0 ]
```

Used by BLOCKOUT game.

BACKGROUND VZ

One of the limitations of the VZ-200 is that it has only two background colours in each mode: green and orange in mode 0, buff and green in mode 1. This short machine code program fills the screen with any desired character in either mode 0 or 1, making any of the eight foreground colours available as a background.

To use the program just type in the listing, either at the start of another program or on its own, and CSAVE it. RUN the program and, to fill the screen, POKE the code for the desired character into location 28672 (start of screen address) and enter PRINT USR(0). In mode 1 and colour 0, 0 gives a green background, 85 gives yellow, 170 blue and 255 gives a red background. In mode 1, colour 1, buff = 0, cyan = 85, 170 = orange and 255 = magenta.

I Williams

Basic listing:

```
10 TM=PEEK(30897)+256*PEEK(30898)-20
20 POKE 30897,TM-INT(TM/256)*256:POKE
   30898,INT(TM/256)
30 TM=TM-1:A=TM-65536
40 FOR I=0 TO 15
50 READ D:POKE I+A,D
60 NEXT I
70 POKE 30862,TM-INT(TM/256)*256:POKE
   30863,INT(TM/256)
80 DATA 58,0,112,71,33,0,112,17,0,120,112,35,
   223,32,251,201
```

Reserve 20 at Tom routine
Reset Tom

should be Tm=Tm+1 (next addr)

Set USR( )

APC May 85 6(5) p 110.

Finds top of memory before loading program. ie. self loading.

Disassembled listing

```
3A 00 70    LD A, (7000H)   ; Start of video RAM.
71          LD B, A         ; save in B
21 00 70    LD HL,7000H     ; start video
11 00 78    LD DE, 7800H    ; end video
70          LD (HL), B.     ; store at (HL)
23          INC HL
DF          RST 18 H        ; <Restart 3>
20 FB       JR NZ, FBH      ; Jump if not zero.
C9          RET
```

?(RST 18) Compares two registers ??

ADDENDUM.
25 CLEAR 50 : Reset stack ptrs.
30 Tm = Tm+1 : A = Tm-65536.

Reversed REM

Labelling subroutines with REM statements that describe the functions of the subroutines is obviously helpful to the programmer who has trouble remembering what parts do what when designing a long program.

One way to make the subroutines stand out in the LISTing is to use inverse REM statements. But the VZ computer will not straightforwardly accept REM statements in inverse print — such REM lines are not entered into the LISTing when return key is pressed and the SYNTAX ERROR? MESSAGE displays.

This can be simply overcome by preceding an inverse REM statement with quotes.

120 REM"AN EXAMPLE

end quotes are not needed; the underlined characters are in inverse form — do not inverse the word REM!

Having suitably named our subroutines, wouldn't it be great if we could call those subroutines by name instead of GOSUB a line number?

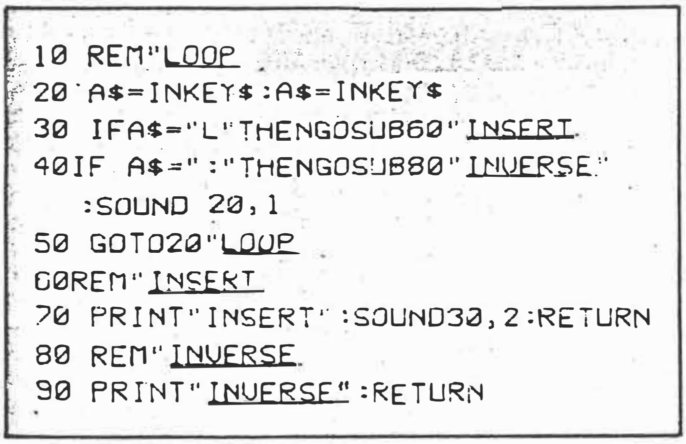
The VZ does not implement procedural calls, but we can simulate this desirable feature by placing the name we have given the subroutine immediately after the GOSUB number:

30 GOSUB120"AN EXAMPLE"

and because the name is in inverse form here also, it stands out clearly in the LISTing that this is a call on that particular subroutine. In the case of a GOSUB you must use end quotes also if any further statements follow the GOSUB on the same program line.

GOTO can be treated in the same way — simply give a REM name to the block of code you GOTO.

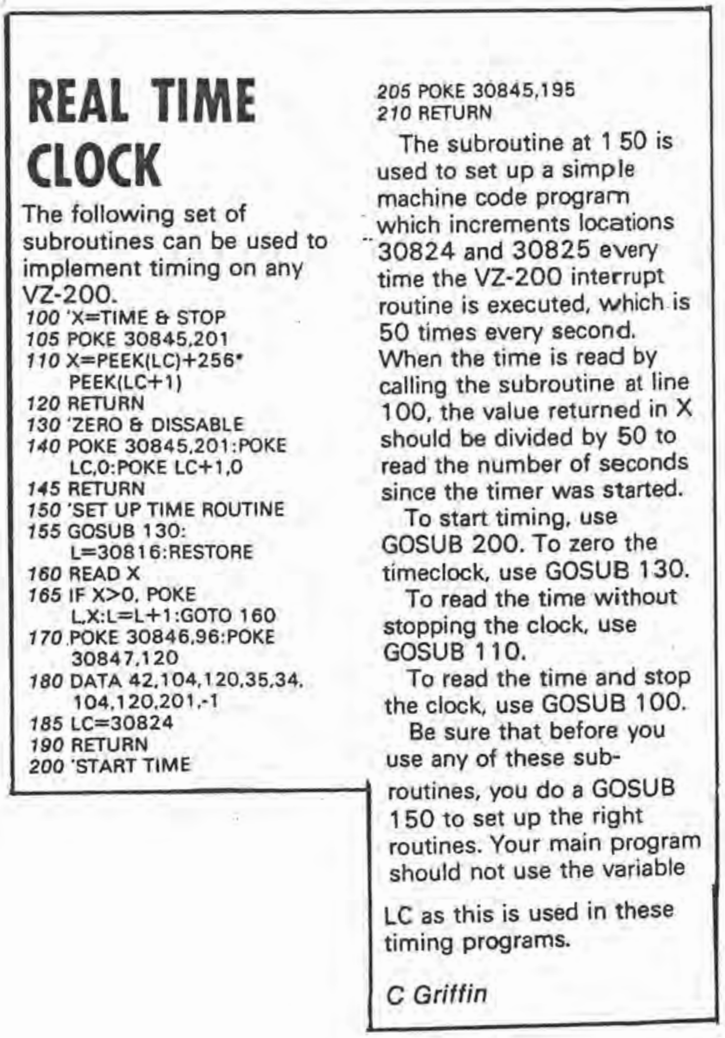
R Quinn



```
10 REM"LOOP
20 A$=INKEY$:A$=INKEY$
30 IFA$="L"THENGOSUB60"INSERT
40IF A$=":"THENGOSUB80"INVERSE"
   :SOUND 20,1
50 GOTO20"LOOP
60REM"INSERT
70 PRINT"INSERT":SOUND30,2:RETURN
80 REM"INVERSE
90 PRINT"INVERSE":RETURN
```

Sample listing

APC Aug. 85 p 130-3.
6(8)



REAL TIME CLOCK

The following set of subroutines can be used to implement timing on any VZ-200.

```
100 'X=TIME & STOP
105 POKE 30845,201
110 X=PEEK(LC)+256*
    PEEK(LC+1)
120 RETURN
130 'ZERO & DISSABLE
140 POKE 30845,201:POKE
    LC,0:POKE LC+1,0
145 RETURN
150 'SET UP TIME ROUTINE
155 GOSUB 130:
    L=30816:RESTORE
160 READ X
165 IF X>0, POKE
    L,X:L=L+1:GOTO 160
170 POKE 30846,96:POKE
    30847,120
180 DATA 42,104,120,35,34,
    104,120,201,-1
185 LC=30824
190 RETURN
200 'START TIME
205 POKE 30845,195
210 RETURN
```

The subroutine at 1 50 is used to set up a simple machine code program which increments locations 30824 and 30825 every time the VZ-200 interrupt routine is executed, which is 50 times every second. When the time is read by calling the subroutine at line 100, the value returned in X should be divided by 50 to read the number of seconds since the timer was started.

To start timing, use GOSUB 200. To zero the timeclock, use GOSUB 130.

To read the time without stopping the clock, use GOSUB 110.

To read the time and stop the clock, use GOSUB 100.

Be sure that before you use any of these subroutines, you do a GOSUB 150 to set up the right routines. Your main program should not use the variable LC as this is used in these timing programs.

C Griffin

V 6 (9) : Sep. 85.

Disassembled listing

```
2A 68 78     LD HL, (7868H)     ; fetch from 30824D
23           INC HL             ; add 1
22 68 78     LD (7868H), HL     ; store at 30824D.
C9           RET
```

BENCHMARKS

A list of Benchmarks used when evaluating micros is given below. An explanation can be found in the February '84 issue.

```
100 REM Benchmark 1
110 PRINT "S"
120 FOR K=1 TO 1000
130 NEXT K
140 PRINT "E"
150 END

100 REM Benchmark 2
110 PRINT "S"
120 K=0
130 K=K+1
140 IF K<1000 THEN 130
150 PRINT "E"
160 END

100 REM Benchmark 3
110 PRINT "S"
120 K=0
130 K=K+1
140 A=K/K*K+K−K
150 IF K<1000 THEN 130
160 PRINT "E"
170 END

100 REM Benchmark 4
110 PRINT "S"
120 K=0
130 K=K+1
140 A=K/2*3+4−5
150 K<1000 THEN 130
160 PRINT "E"
170 END

100 REM Benchmark 5
110 PRINT "S"
120 K=0
130 K=K+1
140 A=K/2*3+4−5
150 GOSUB 190
160 IF K<1000 THEN 130
170 PRINT "E"
180 END
190 RETURN

100 REM Benchmark 6
110 PRINT "S"
120 K=0
130 DIM M(5)
140 K=K+1
150 A=K/2*3+4−5
160 GOSUB220
170 FORL=1 TO 5
180 NEXTL
190 IF K<1000 THEN 140
200 PRINT "E"
210 END
220 RETURN

100 REM Benchmark 7
110 PRINT "S"
120 K=0
130 DIM M(5)
140 K=K+1
150 A=K/2*3+4−5
160 GOSUB 230
170 FOR L=1 TO 5
180 M(L)=A
190 NEXTL
200 If K<1000 THEN 140
210 PRINT "E"
220 END
230 RETURN

100 REM Benchmark 8
110 PRINT "S"
120 K=0
130 K=K+1
140 A=K^2
150 B=LOG(K)
160 C=SIN(K)
170 IF K<1000 THEN 130
180 PRINT "E"
190 END
```

V6(10): Oct.85.

VZ DELETIONS

The VZ-200 computer is a much more powerful machine than appears. Many of its facilities slumber because someone has made a marketing decision to restrict Basic access to certain facilities. Here is how one of them can be awakened.

DELETE is a Basic editing command that allows you to erase a block of Basic lines from a program in one go, instead of having to eliminate them one by one by entering each line number and pressing the return key.

Suppose, for example, you want to delete lines 250 to 530 from a program. Add this line to your program:

```
0 D250-530
```

Now enter the following commands and press the return key:

```
POKE31469,182:RUN
```

If you now list the program you will find the absence of all those lines you desire to be rid of. The content of line 0 will be invisible. Having accomplished your goal you can delete line 0 in the conventional way — enter 0 and press return.

0 D-x where x is an end line number will, when the above POKE is made and the program RUN, eliminate all lines from the first line in the program (which of course will be line 0:) to line x.

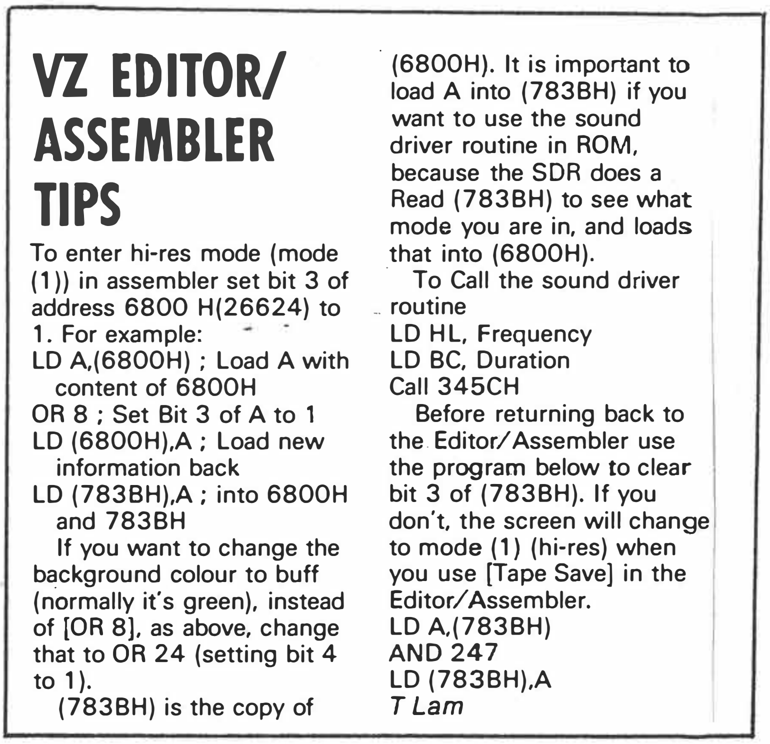
On another matter, try this line:

```
10 FORR=5TO485STEP32:
    PRINT@R,""; :INPUTA:
    PRINT@R+16, "A=";A:
    NEXT
```

What it shows is that PRINT@ and INPUT statements will not work together on odd numbered lines (counting down the screen 0,1,2,...,16). A numerical INPUT will always return 0; a string INPUT will return the null string. So take care when programming with these two statements.

R Quinn

V 6(10): Oct. 85.



VZ EDITOR/ ASSEMBLER TIPS

To enter hi-res mode (mode (1)) in assembler set bit 3 of address 6800 H(26624) to 1. For example:

```
LD A,(6800H) ; Load A with
   content of 6800H
OR 8 ; Set Bit 3 of A to 1
LD (6800H),A ; Load new
   information back
LD (783BH),A ; into 6800H
   and 783BH
```

If you want to change the background colour to buff (normally it's green), instead of [OR 8], as above, change that to OR 24 (setting bit 4 to 1).

(783BH) is the copy of (6800H). It is important to load A into (783BH) if you want to use the sound driver routine in ROM, because the SDR does a Read (783BH) to see what mode you are in, and loads that into (6800H).

To Call the sound driver routine

```
LD HL, Frequency
LD BC, Duration
Call 345CH
```

Before returning back to the Editor/Assembler use the program below to clear bit 3 of (783BH). If you don't, the screen will change to mode (1) (hi-res) when you use [Tape Save] in the Editor/Assembler.

```
LD A,(783BH)
AND 247
LD (783BH),A
```

T Lam

A.P.C. 6(11): Nov. 85.
p. 189.

LOW COST PROGRAM GIVES VZ200/300 FULL LEVEL II BASIC

Ever wished that your little VZ200 or VZ300 would run full Microsoft Level II BASIC instead of just a stripped-down version? You needn't wish any longer thanks to an enterprising local programmer.

Jim Rowe

REMEMBER STEVE OLNEY? If you're a VZ200 or VZ300 owner and BASIC programmer, you should. We've published at least three of his articles so far, mainly on resurrecting dormant functions and statement keywords in VZ BASIC. One was in the March '84 issue, another in October '84 and the last in May '85.

Steve's a very knowledgeable guy when it comes to the VZ200/300, in terms of both software and hardware. He's spent quite a lot of time burrowing into its little secrets, and probably knows as much about it as anyone in Australia.

I know that sounds a bit like paeaning in his pocket, but I've just been trying out the latest fruit of his labours. And this time it's not just an article showing you how to restore a few more missing functions to VZ BASIC. It's a machine language utility program that restores pretty well the whole blinking lot for you — instant Level II BASIC! Hence my little paean of praise.

Steve calls his new utility Extended BASIC Version 2.2, or 'EXBSV2.2' for short. It is available on either cassette tape or disk, to suit both basic and expanded VZ systems. It is also compatible with both the VZ200 and VZ300, and with the current Disk BASIC (V1.2 DOS).

You load EXBSV2.2 into your VZ before you load in anything else. It is only about 1600 bytes long (about 1.5K) and is fully self-locating, finding the top of available RAM and installing itself there. At the same time it lowers the BASIC 'top of RAM' pointer to prevent any other programs from being loaded over it.

As part of the installation it patches itself into ROM BASIC, in much the same way that Disk BASIC does, to become transparent to the user. All that you're aware of is that the RAM is now about 1.5K smaller than before — plus, of course, the fact that your trusty VZ now responds to no less than 25 new BASIC commands!

Of these 25 new commands, 23 are basically resurrected Level II commands that have been sleeping there all the time in the VZ's ROM, quietly waiting for EXBSV2.2 to sound the trumpet. They're listed in the table. The other two are extras — a bonus that Steve Olney has thrown in for good measure. And very handy thay are too: MERGE, to allow you to combine programs and routines, and RENUM to let you rationalise and tidy up a program whose line numbers have become a mess after a lot of editing and patching (or after using MERGE).

All of the 25 new commands are fully functional, and when used in a program can be LISTed — at least on any machine with EXBSV2.2 loaded. All but two of them will even RUN on a VZ which doesn't have EXBSV2.2 loaded! The two exceptions are ON and ERROR, which arise because of a conflict in token codes (normal VZs use the normal ERROR token for the added command SOUND).

Even here Steve Olney has provided an answer, for those who really do want the Level II programs they generate to be capable of running on plain-vanilla VZs (how helpful can the guy get?). He's done this by providing the listing of a short BASIC routine which you can MERGE into the top of your programs after they're finished and debugged. You then use it to convert your finished programs

When it has finished, you DELETE the routine itself (notice that?) and CSAVE the converted program. It won't LIST properly any more, but it will now RUN on a VZ without EXBSV2.2 installed. There's just one tiny catch: you can't use the construct 'IF <expression> THEN ERROR <n>' in any program that you want to convert in this fashion. You can only use ERROR in the 'ON ERROR GOTO' construct. Not a serious limitation, but worth remembering.

But back to EXBSV2.2 itself. Normally you'd expect to load this into your VZ every time you turn it on, which is easy enough and only takes a couple of seconds with the disk system. And with the utility installed, all of the new commands are at your disposal.

It's great to be able to use direct commands like DELETE, AUTO, TRON and TROFF, RENUM and MERGE. How did we ever get along without DELETE? It's so damn useful — not to say virtually essential when you want to scrub a whole range of program lines.

Then into the actual programming. It's really good to be able to use double-precision constants and variables again. Plus to be able to define variables as integer, single, double or string type using DEFINT, DEFSNG, DEFDBL and DEFSTR. It's also much neater to be able to use ON-GOTO and ON-GOSUB, instead of a flock of IF-THENs. Not to mention being able to use ERROR, ERR and ERL. It's nice to be able to use RESUME and RANDOM, too.

Of course there's also FIX, FRE, and MEM — plus familiar old mates like CINT, CSNG and CDBL, POS and STRING$ (handy in setting out screens, that one — I missed it). And of course the very versatile VARPTR. Wheee! Makes

1 of 2.

Feb 86. Olney sells a V2.3.
Apr 86 V2.5

TABLE 1. WHAT EXTENDED BASIC PROVIDES

System Commands:	
AUTO	automatic line numbering for program entry
DELETE	delete a line or group of lines
TRON	enable trace function (for debugging)
TROFF	disable trace function
MERGE	merge tape program with program in memory
RENUM	renumber program lines
BASIC Statements:	
DEFINT	define variable as an integer
DEFSNG	define variable as single precision
DEFDBL	define variable as double precision
DEFSTR	define variable as string type
ERR	error code
ERL	line in which error was deleted
ERROR	used to simulate an error condition
ON-GOTO	branch to one of several line numbers depending upon the value of an expression
ON-GOSUB	branch to one of several subroutines depending upon the value of an expression
RANDOM	reseed random number generator
RESUME	continue program execution after error handling
BASIC Functions:	
CINT	convert variable to an integer
CSNG	convert variable to single precision
CDBL	convert variable to double precision
FIX	return truncated integer part of a number
FRE	returns the amount of free memory remaining
MEM	returns the amount of free memory remaining
POS	returns the current screen cursor position
STRING$	returns a string of specified length
VARPTR	locates a variable in memory

you feel a bit like Uncle Scrooge let loose in the Mint (well almost).

All of the new commands and functions seem to work perfectly. I certainly couldn't find any bugs, anyway — if there are any, they're pretty well hidden. From a functional point of view, my VZ now behaves like any other Level II machine.

So thanks to EXBSV2.2, Steve Olney's little genie, you can now trundle out all those old TRS80/System80 programs and get them running on your trusty VZ. The graphics will need a few mods, of course, but the programs themselves will be fine.

And the cost of this magic ute? A mere $15 for the tape version, or $22 for the disk version. Both prices include packing and postage, and EXBSV2.2 comes complete with a set of driving instructions. You couldn't get much better value for money — obviously Steve Olney is not out to rip anyone off.

I've only got one complaint. Couldn't he have given it a name that's easier to pronounce and type, like 'Jeannie'? Try typing EXBSV2.2 all the way through a review, and you'll know what I mean!

Still, whatever he cares to call it, it's a utility that almost every VZ programmer is going to want. And at this stage you can only get it direct from Steve Olney at 200 Terrace Road, North Richmond, NSW 2754. I only hope that his local post office is prepared for the onslaught. ●

ETI Nov. 85 p. 95.
2 of 2.

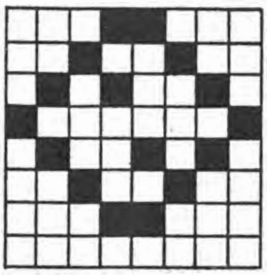
VZ USER GRAPHICS

```
1000 A=44800:B=65536
1010 READ C:IF C=1THEN
     1070ELSEPOKEA-B,C
     :A=A+1:GOTO1010
1020 DATA245,197,213,
     229,33,0,0,17,0,0
1030 DATA14,8,26,119,
     35,19,26,119,6,31
1040 DATA35,5,120,254,
     0,194,20,175,19,13
1050 DATA121,254,0,
     194,12,175,241,193
1060 DATA209,225,
     201,-1
1070 POKE30862,0:POKE
     30863,175:RETURN
```

This routine will provide any VZ programmers with the ability of creating their own definable high resolution characters in 8x8 pixels.

For example

```
00000011    11000000
00001100    00110000
00110011    00001100
11000000    00000011
00110000    11001100
00001100    00110000
00000011    11000000
00000000    00000000
```



Refer to the technical manual for more details on high resolution graphics.

To activate this routine, you simply poke the starting address of the code for your user definable graphic into the memory location 44808/9 and the screen position of your user definable graphic into the memory location 44805/6.

Sample Program

```
   5 GOSUB 1000
  10 FOR T=45000 TO
     45015
  20 READ S:POKE
     T-65536,S
  30 NEXT T
  40 POKE 44808-65536,
     200:POKE 44809-
     65536,175
  50 POKE 44805-65536,
     0:POKE 44806-
     65536,112
  60 MODE(1):X=USR(0)
  70 GOTO 70
1080 DATA 3,192,12,48,
     51,12,192,3,48
1090 DATA 204,12,48,3,
     192,0,0
```

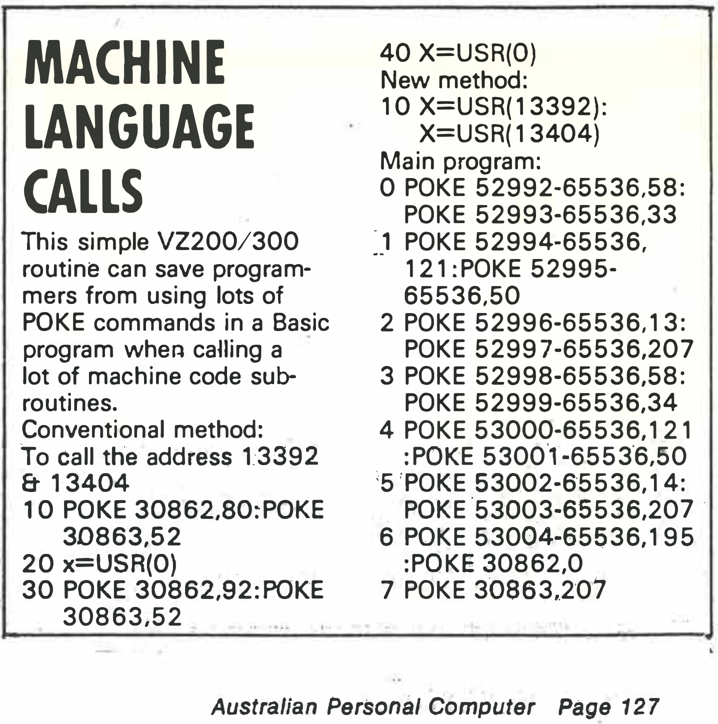
APC Jan 86, 7(1) : 83 & 85.

```
44800D ≡ AF00 H.            AFH ≡ 175D

AF00  F5          PUSH AF         ; Save reg.
 " 01 C5          PUSH BC         ;  to stack.
 " 02 D5          PUSH DE         ;
 " 03 E5          PUSH HL         ;
 " 04 21 00 70    LD HL, 7000H    ; screen posn.
 " 07 11 C8 AF    LD DE, AFC8 H   ; user def. graphics
 " 0A 0E 08       LD C, 08 H.     ; 8 bytes-pairs.
 " 0C 1A        ┌ LD A, (DE)      ; char. into A.
 " 0D 77        │ LD (HL), A      ; transfer to screen.
 " 0E 23        │ INC HL          ; next scrn loc.
 " 0F 13        │ INC DE          ; next char.
 " 10 1A        │ LD A, (DE)      ; next char. into A.
 " 11 77        └ LD (HL), A      ; transfer to screen
   (handle pairs of gr. chars.)
 " 12 06 1F     ┌ LD B, 1FH;  B = 31D. (LF)
 " 14 23        │ INC HL          ; next scrn pos.
 " 15 05        │ DEC B           ; B = B-1.
 " 16 78        ↓ LD A, B;          into A.
   (LF routine.)
```

```
AFC8 H ≡ 45000D;  C8H ≡ 200D;  AFH = 175D.

AF17 FE 00     ↑ CP 00 H        ; set flag.
 " 19 C2 14 AF └ JP NZ AF14 H   ; cont. bumping LF routine.
 " 1C 13         INC DE         ; next graphics char.
 " 1D 0D         DEC C          ; dec. pair counter.
 " 1E 79         LD A, C        ; into A.
 " 1F FE 00      CP 00 H        ; set flag.
 " 21 C2 0C AF   JP NZ AF0CH    ; get next pair.
 " 24 F1         POP AF  ┐
 " 25 C1         POP BC  │  * incorrect.
 " 26 D1         POP DE  │    order to
 " 27 E1         POP HL  ┘    restore from
                              stack.
 " 28 C9         RET
```

Program reads in 8×2 bytes from AFC8 H and puts it onto to top LH corner of screen. or other posn. as determined by AF05/6. H.

Not a particularly elegant program but has some developement potential for sprites.



MACHINE LANGUAGE CALLS

This simple VZ200/300 routine can save programmers from using lots of POKE commands in a Basic program when calling a lot of machine code subroutines.

Conventional method:
To call the address 13392 & 13404

```
10 POKE 30862,80:POKE
   30863,52
20 x=USR(0)
30 POKE 30862,92:POKE
   30863,52
40 X=USR(0)
```

New method:

```
10 X=USR(13392):
   X=USR(13404)
```

Main program:

```
0 POKE 52992-65536,58:
  POKE 52993-65536,33
1 POKE 52994-65536,
  121:POKE 52995-
  65536,50
2 POKE 52996-65536,13:
  POKE 52997-65536,207
3 POKE 52998-65536,58:
  POKE 52999-65536,34
4 POKE 53000-65536,121
  :POKE 53001-65536,50
5 POKE 53002-65536,14:
  POKE 53003-65536,207
6 POKE 53004-65536,195
  :POKE 30862,0
7 POKE 30863,207
```

Australian Personal Computer   Page 127

Feb 86 7(2)

This is a fairly elegant procedure if a number of calls to low level subroutines is required. (The coding suggested is dreadful - use a FOR-NEXT loop to load data.)

Background. to method.

The USR( ) command is capable of passing an argument to the subroutine being called. Usually a dummy (0) is passed. The argument is stored in 31009 & 31010 (LSB 7921 H & MSB 7922 H.) This method passes the start address of the required routine via this "Jump" routine. The RETurn in the subroutine called goes back to BASIC.

Conventional method.

13392 ≡ 3450 H   st. add. Rom generate beep

13404 ≡ 345C H.   st. add. Rom generate sound.

(These are two subs. in Rom used as examples.)

line 10 pokes 3450 H as 'jump address'.
line 30 pokes 345C H as 'jump address'.
both initiated by USR command in lines 20 and 40. Note dummy argument passed.

5/3/91

Better - use !!!

```
LD  HL, 7921H     21 21 79
JP  (HL)          E9
```

New method.

52992 - 65536 = -12544 ≡ CF00 H
53004 - 65536 = -12532 = CF0C H   } 13 bytes

(note that next two bytes are used also)
15 bytes in all.

Disassembled listing.

```
CF00   3A 21 79     LD A, (7921 H)
  03   32 0D CF     LD (CF0DH), A
  06   3A 22 79     LD A, (7922 H)
  09   32 0E CF     LD (CF0E H), A
CF0C   C3 nn nn     JP Annn
         (LSB)(MSB)
```

Thus the argument passed by the USR command is read from 7921/2 H and written into CF0D/E H. which is then jumped to.

This simplifies the main line program significantly.

From: Australian Personal Computer November 1983
Vol. 4 No. 11 p89-95
Revised: Australian Personal Computer March 1986
Scanned: Bob Kitch February 2021 at 400 dpi
Composited: MicroSoft Publisher 2016
Size: 600 x 820 mm

BASIC CONVERTER CHART '86

Australian Personal Computer
WALLCHART

Those rotten manufacturers still insist on making machines that won't talk to each other in the same language. Some enlightened people are having a go with MSX, but in the meantime and in response to overwhelming demand, here's the 1986 APC Converter Chart.

We've added seven new Basics, covering the latest machines, and revised and updated the chart. It isn't possible, of course, to cover every micro nor every command supported by each of the machines included. What this chart aims to do is to provide an at-a-glance syntax comparison using Microsoft Basic as a reference point. The chart won't convert programs for you but it will save you the trouble of getting hold of piles of manuals — and even when you've got them it's often the beginning, not the end of your worries.

To use the chart, first check that the keyword you want isn't in the box on the right. If it is, then you're lucky: it's one of the few that IS the same on every single machine featured here. Due to the limited amount of information we can squeeze into each box, it hasn't always been possible to indicate the full power of every statement. It should be assumed, therefore, that we're dealing with the most common uses of each statement, and that other uses may be available.

Something to watch out for: identical syntax may have different effects on different machines. Watch out especially for SYSTEM and RND.

You'll notice we haven't included anything on sound and graphics: that's too complicated for a quick reference chart, but we've covered the subject in a series of articles which will appear in APC for a range of machines.

SHARED INSTRUCTIONS

ABS	(exp)	
COS	(exp)	
END		NB not available on QL
FOR	var=exp	TO exp [STEP exp]
LEN	(string)	NB Space *must* be present for Memotech
LET	var=EXP	NB LET obligatory after THEN and ELSE on MicroBee
REM	text	
SIN	(exp)	
SQR	(exp)	
STOP		
TAN	(exp)	
VAL	(exp)	NB not available on QL

ABBREVIATIONS USED IN THIS CHART:

addr	= address
exp	= expression
parm(s)	= parameter(s)
stmt	= statement
var	= variable

Square brackets [] indicate optional code.

BASIC RESERVED WORDS & FORMATS

MACHINE / STANDARD MICROSOFT	ASC Returns ASCII value of first character of string.	ATN Arctangent of expression.	AUTO	CALL Calls assembler language sub-routine	CHAIN Call a new program & pass variables to it.	CHR$ Gives one-char string with ASCII code of exp.	CLEAR CLEAR all [or selected] variables.	CLOSE Closes disk files — closes all files if no specification.	CONT continue program execution	DATA Lists data to be used in a READ statement	DEF Define arithmetic string function.	DELETE Delete specified program lines.	DIM Allocates space for arrays, specifies max subscript values.	EDIT Edit a program line.	EXP Raises to power of expression.	FRE Returns remaining memory space.	GET Read a record from disk or tape file.	GOSUB Branch to a Basic subroutine.	GOTO Branch to a specified line number.	IF/THEN/ELSE If exp is true stmt is executed. If not ELSE or following line is executed.
STANDARD MICROSOFT	ASC (string)	ATN (exp)	AUTO [lineno, val]	CALL var[,var, var . . .]	CHAIN "filename"	CHRS (exp)	CLEAR [exp,exp]	CLOSE	CONT	DATA const [,const, . . .]	DEF FNvar [(var,var . . .)] =exp	DELETE lineno [,lineno]	DIM var(sub) [,var (sub), . . .]	EDIT lineno	EXP(exp)	FRE(exp)	GET [#] lineno [,record no] or INPUT#fileno, var [,var . . .] for sequential files	GOSUB lineno	GOTO lineno	If exp THEN [ELSE stmt]
AMSTRAD 464/664/6128	ASC (string)	ATN (exp)	AUTO [lineno, incl]	CALL addr [,parms]	CHAIN "filename" [,lineno,exp]	CHRS (exp)	CLEAR [all] ERASE [list of] var NB: clears and removes arrays	CLOSEIN [NB cassette input file] CLOSEOUT [NB cassette output file]	CONT	DATA const [,const, . . .]	DEF FN(var) [(var,var, . . .)]=exp	DELETE [line no-line no]	DIM [list of] var (dimension list)	EDIT lineno	EXP(exp)	FRE(exp) Note: exp is a dummy variable	LINE INPUT#, [;string][var, . . .]	GOSUBlineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
APPLE II	ASC (string)	ATN (exp)		CALL addr [var,var . . .]	CHAIN "filename"	CHRS (exp)	CLEAR	CLOSE [filename]	CONT	DATA const [,const, . . .]	DEF FNvar (var)=exp	DEL lineno, lineno	DIM var(sub) [,var(sub) . . .]	[screen editing using ESC key]	EXP(exp)	FRE(exp) Note: exp is a dummy variable	INPUT var [,var . . .] NB: Get var(s) from current input device	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
ATARI	ASC (string)	ATN (exp)		USR (addr [,var,var . . .])	RUN "C:" NB: program must have been saved using SAVE "C:"	CHRS (exp)	CLR	CLOSE [#fileno, var,var,filename]	CONT	DATA const [,const, . . .]			DIM var (sub) [,var (sub) . . .] NB: dimension ALL strings	[cursor editing]	EXP(exp)	FRE(exp) Note: exp is a dummy variable	GET #lineno, record	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	If exp THEN stmt Note: no ELSE
BBC	ASC (string)	ATN (exp)	AUTO [lineno, val]	CALL addr, [var][,var, . . .]	CHAIN "filename"	CHRS (exp)	CLEAR	CLOSE #fileno Note: CLOSE #0 to close all files	NB not available: use GOTO lineno	DATA const [,const, . . .]	DEF FNvar =exp	DELETE lineno [-lineno]	DIM var(sub) [,var(sub) . . .]	[cursor editing]	EXP(exp)	HIHEM-TOP Use PRINT	INPUT #lineno, record [,record . . .]	GOSUB lineno/ [var][exp]	GOTO lineno/ [var][exp]	If exp THEN stmt [ELSE stmt]
COMMODORE 64 & VIC 20	ASC (string)	ATN (exp)		SYS addr		CHRS (exp)	CLR	CLOSE #fileno	CONT	DATA const [,const, . . .]	DEF FNvar =exp		DIM var(sub) [(sub) . . .]	[cursor editing]	EXP(exp)	FRE(exp) Note: exp is a dummy variable	GET #fileno, var	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
IBM PC-BASIC A	ASC (string)	ATN (exp)	AUTO [lineno] [,inc]	CALL addr [var, . . .,var]	CHAIN filename	CHRS (exp)	CLEAR	CLOSE [#] [filename]	CONT	DATA const [,const, . . .]	DEF FNvar [(parms)]=exp	DELETE [lineno] [-lineno]	DIM var(sub) [,var(sub) . . .]	EDIT lineno	EXP(exp)	FRE(exp) Note: exp is a dummy variable	GET [#] filename [,rec no]	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
MEMOTECH MTX 512	ASC (string)	ATN (exp)	AUTO [lineno] [,inc]	USR (addr)		CHRS (exp)	CLEAR	DISC CLOSE # channel no	CONT	DATA const [,const, . . .]	DEF FNvar [(parms)]=exp		DIM var(sub) [,var(sub)] . . .	EDIT lineno	EXP(exp)		Disc INPUT # channelno	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
MICROBEE	ASC (string)	ATAN (real-exp)	AUTO [lineno, val]			CHR (integer-exp)	STRS (int-exp) Note: set limits for string memory		CONT	DATA expr (,exp ("exp"))	FNn=exp	DELETE lineno. lineno.)	DIM var(sub) [,var(sub) . . .]	EDIT (lineno.)	EXP(exp)	FRE(0) mem. space FRE($) str. space		GOSUB NB: sq. br. significant	GOTO lineno	If exp THEN stmt [ELSE stmt]
MSX BASIC	ASC (string)	ATN (exp)	AUTO [lineno ,inc]	USR (addr)		CHRS (exp)	CLEAR [var]	DISK basic only	CONT	DATA const [,const, . . .]	DEF FN(var) [(parms)]=exp	DELETE [lineno] -lineno]	DIM var(sub) [,var(sub) . . .]	[cursor editing]	EXP(exp)	FRE(exp) Note: exp is a dummy variable		GOSUB lineno	GOTO lineno	IF exp THEN stmt [ELSE stmt]
TANDY 100	ASC (string)	ATN (exp)		CALLadr [,param,param]		CHRS (exp)	CLEAR [(exp)] — Clears string space	CLOSE [fileno] if exp is given	CONT	DATA const [,const . . .]			DIM var(sub) [,var(sub) . . .]	EDIT lineno [-lineno]	EXP (exp)	FRE (exp)	INPUT #fileno, var [,var . . .]	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
TANDY COLOR	ASC (string)	ATN (exp)		EXEC addr		CHRS (exp)	CLEAR [(exp)] clears string space if exp is given	CLOSE #-fileno	CONT	DATA const [,const, . . .]	DEF FNvar (var)=exp	DELETE lineno-neno	DIM var(sub) [,var(sub) . . .]	EDIT lineno	EXP(exp)	MEM	INPUT #-fileno, record	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
SINCLAIR QL	CODE (str)	ATAN (exp)	AUTO [lineno] [,inc]	CALL addr [,parms]	M RUN "filename"	CHRS (exp)	CLEAR	CLOSE # channel	CONTINUE	DATA const [,const, . . .]	DEF FNvar [S/%]=exp: END DEF	DLINE lineno [TO]lineno]	DIM var(sub) [,var(sub) . . .]	EDIT lineno [,step]	EXP(exp)		INKEYS (#channel)	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	IF exp THEN stmt [ELSE stmt][END IF]
TRS-80 II/SYSTEM 80	ASC (string)	ATN (exp)	AUTO [lineno, val]			CHRS (exp)	CLEAR [(exp)] Note: Clears string space if exp given	[depends on OS; consult OS manual]	CONT	DATA const [,const, . . .]	Various DEF statements available but none equivalent	DELETE lineno -lineno]	DIM var(sub)	EDIT lineno	EXP(exp)	FRE(exp) [TRS-80] or MEM [System 80]	INPUT #-fileno, record [,record . . .]	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
VA-200	ASC (string)	ATN (exp)				CHRS (exp)	CLEAR [exp] N clears string space		CONT	DATA const [,const, . . .]			DIM var(sub) [,var(sub) . . .]		EXP(exp)		INPUT # file-name var[,var . . .] NB: Gets record from tape	GOSUB lineno	GOTO lineno	IF exp THEN stmt [ELSE stmt]
ZX SPECTRUM	CODE (string)	ATN (exp)		LET var=USR addr		CHRS (exp)	CLEAR [var]	CLOSE # channel no	CONT	DATA const [,const, . . .]	DEF FN(var) [(var,var . . .)]		DIM var(sub)	EDIT [lineno] Note: cursor line by default	EXP(exp)		Consult Microdrive manual	GOSUB lineno [exp]	GOTO lineno [exp]	IF exp THEN stmt Note: no ELSE

MACHINE / STANDARD MICROSOFT	INKEY$ Returns character typed at keyboard or null if no character used	INPUT Read data from terminal	INT Evaluates expression for largest integer contained.	LEFT$ Returns specified no. of characters starting at beginning of string.	LIST List specified program lines at terminal.	LLIST List specified program lines at printer.	LOAD Load a program file into memory.	LOG Natural logarithm of expression.	MID$ Gives specified no. of characters to the right of start position in string.	NAME Rename a file.	NEW Delete current program & data from memory.	NEXT End of FOR/NEXT loop.	ON ERROR Error trap subroutine.	ON/GOSUB GOTO lineno specified by evaluation of expression.	ON/GOTO GOTO lineno specified by evaluation of expression.	OPEN Open disk file.	OUT Put specified byte to specified output port.	PEEK Read byte from specified memory location.	POKE Put specified byte to specified memory address.	PRINT Write data to disk tape or terminal.
STANDARD MICROSOFT	INKEYS	INPUT [string:] var[,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno, lineno]	LLIST [lineno, lineno]	LOAD ["filename"]	LOG(exp)	MIDS(string,start [,length])	NAME "filename" AS "filename"	NEW	NEXT var [,var . . .]	ON ERROR GOTO lineno	On exp GOSUB lineno [,lineno . . .]	On exp GOTO lineno [,lineno . . .]	OPEN mode [#] fileno "filename"	OUT port, byte	PEEK(addr)	POKE addr,byte	PRINT [[#] fileno] [exp] [,exp . . .]
AMSTRAD 464/664/6128	INKEYS	INPUT [#no] [prompt] [var [list]]	INT (exp)	LEFTS (string, length)	LIST [lineno, lineno]	LIST [lineno, lineno] #8	LOAD ["filename"] [,addr]	LOG(exp) Note: LOG10(exp) gives Log base 10	MIDS(string, start, length)		NEW	NEXT [var][,var, . . .]	ON ERROR GOTO lineno	On exp GOSUB lineno [,lineno, . . .]	ON exp GOTO lineno [,lineno, . . .]	OPEN mode [#] fileno "filename"	OUT port, byte	PEEK(addr)	POKE addr,byte	PRINT [#]fileno] [exp][,exp . . .]
APPLE II	Get var	INPUT [string,] var [,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno, lineno] Note: '—' may be used in place of ','	PR # slotno: LIST [lineno,lineno]: PR # 0	LOAD filename	LOG(exp)	MIDS(string, start[,length])	RENAME oldname, newname	NEW	NEXT [var,var . . .]	ONERR GOTO lineno	On exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN filename, parm		PEEK(addr)	POKE addr,byte	PRINT exp [,exp . . .] NB: prints to current output device
ATARI		INPUT [string,] var [,var . . .]	INT (exp)	string (start, length)	LIST [lineno, lineno]	LIST [lineno, lineno]	CLOAD ["filename"] [cases] or LOAD "fileno-filename" [disk]	LOG(exp)	string(start [,length])		NEW	NEXT var	TRAP lineno/ var/exp	ON exp GOSUB lineno/var/exp [,lineno/var/exp . . .]	ON exp GOTO lineno/var/exp [,lineno/var/exp . . .]	OPEN #fileno, mode control code, filename	[not equivalent]	PEEK(addr)	POKE addr,byte	PRINT #device, exp[,exp]
BBC	Get var [unlimited] time] or INKEYS (time) Note: 100ths sec.	INPUT [string;] var [,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno, lineno]	CTRL-B then LIST [lineno-lineno]	LOAD "filename" Note '*DISK' or '*TAPE' to select device	LN(exp) NB: LOG(exp) gives common rather than natural log	MIDS(string, start[,length])		NEW Note: under cert. circum. may be recovered using OLD	NEXT [var][,var . . .]	ON ERROR stmt [OFF]	ON exp[var] GOSUB lineno [,lineno . . .]	ON exp[var] GOTO lineno [,lineno . . .]	fileno=OPENIN [to read] or fileno= OPENOUT [to write]		?addr NB: '?' does NOT mean 'print' in BBC Basic	?addr,byte	PRINT #fileno record [,record . . .]
COMMODORE 64 & VIC 20	GET var	INPUT [string:] var [,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno-lineno]	OPEN 4,4:CMD4: LIST [lineno-lineno] OPEN 3,4:CMD3: LIST [lineno-lineno]	LOAD ["filename"] [cass] or LOAD "filename",8 [disk]	LOG(exp)	MIDS(string, start[,length])	OPEN 1,8,15, "RO: filename= filename" [disk only] N/A	NEW	NEXT [var][,var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN #exp, fileno, mode, "filename"		PEEK(addr)	POKE addr,byte	PRINT #fileno record [,record . . .]
IBM PC-BASIC A	var $=INKEYS	INPUT [prompt] var [,var]	INT (exp)	LEFTS (string, length)	LIST [1st line] [-last line] [,filespec]	LIST [lineno, lineno]	LOAD filename [,R]	LOG(exp)	MIDS(string, start[,length])	NAME filename AS filename	NEW	NEXT [var,var, . . .]	ON ERROR GOTO lineno	ON [exp:COM: KEY:PEN:STRING] GOSUB lineno	ON exp GOTO lineno	OPEN filename [FOR Mode] AS [#] filename [LEN=rec]	OUT port,data	PEEK(addr)	POKE addr,byte	PRINT [exp][;]
MEMOTECH MTX 512	var $=INKEYS	INPUT [prompt] var [,var]	INT (exp)	LEFTS (string, length)	LIST [1st line] [,last line]	LIST [lineno, lineno]	LOAD "filename"	LN(exp)	MIDS(string, start[,length])	DISC REN string=string	NEW	NEXT var		ON exp GOSUB lineno	ON exp GOTO lineno	DISC OPEN # channel no, "filename", filetype, record length	OUT port,data	PEEK(addr)	POKE addr,byte	[DISC] PRINT [#channelno,] print list
MICROBEE	KEY	INPUT [string] var [,var]	INT (real-exp)	var(;1, length)	LIST [lineno [,lineno]] forceloads	LLIST (lineno. (,lineno))	LOAD (U) (?) ("filename") LOAD U	LOG(real-exp)	var(;n,n+m-1) -n-start character, m-length		NEW	NEXT var NEXT *var lineno. -exits loop before completion	ON ERROR GOTO lineno.	ON exp GOSUB ([exp(,exp)])lineno (,([exp . . .	ON exp GOTO lineno [,lineno . . .]		OUT port,byte	PEEK(addr)	POKE addr, byte	PRINT list of arguments
MSX BASIC	var $=INKEYS	INPUT [prompt] var [,var]	INT (exp)	LEFTS (string, length)	LIST [1st line] [-last line]	LLIST [lineno, lineno]	CLOAD filename	LOG(exp)	MIDS(string, start[,length])		NEW	NEXT [var][,var, . . .]	ON ERROR GOTO lineno	ON exp GOSUB lineno [,lineno, . . .]	ON exp GOTO lineno [,lineno, . . .]	OPEN "device: filename" for OUTPUT [for INPUT] AS#var	OUT port,data	PEEK(addr)	POKE addr,byte	PRINT [#fileno,] var[,var, . . .]
TANDY 100	INKEYS	INPUT [string;] var [,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno-lineno]	CLOAD lineno]	LOG (exp) ["filename"]	MIDS(string,	start[,length])	NEW		NEXT [var,var . . .]	ON ERROR GOTO lineno	On exp GOTO lineno [,lineno . . .]	OPEN "filename" lineno	OUT port,byte FOR (mode)	PEEK(addr)	POKE addr,byte	PRINT [#fileno]	[exp,exp . . .]
TANDY COLOR	INKEYS	INPUT [string;] var [,var]	INT (exp)	LEFTS (string, length)	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD ["filename"]	LOG(exp)	MIDS(string, start[,length])		NEW	NEXT [var][,var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN mode, #-fileno "filename"		PEEK(addr)	POKE addr,byte	PRINT #-fileno. exp[,exp . . .]
SINCLAIR QL	INKEYS (#channel ,time)	INPUT [# channel,] [prompt] [var [,var] . . .]	INT (exp)	string (TO finish)	LIST [#channel] 1st line [To last line]	[LIST #[channel] [lineno] [To lineno]	LOAD device [inc. filename]	LN(exp) Note: LOG10(exp) gives common rather than natural log	string(start TO finish)		NEW	NEXT var[,var]/ END FOR var[,var]	WHEN ERROR var: END WHEN Note: OS JS	ON var GOSUB lineno [,lineno]	ON var GOTO lineno [,lineno]	OPEN #channel, "filename"		PEEK [or W or L](addr)	POKE [or W or L](addr),byte	PRINT [#channel,] exp[,exp] . . .
TRS-80 II/SYSTEM 80	INKEYS	INPUT [string;] var [,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD "[filename]" [cass] or LOAD "filename" [disk/ floppy tape]	LOG(exp)	MIDS(string, start,length)	[depends on OS; consult OS manual]	NEW	NEXT [var][,var . . .]	ON ERROR GOTO lineno	ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	[depends on OS; consult OS manual.]	OUT port,byte	PEEK(addr)	POKE addr,byte	PRINT #fileno, record [,record . . .] [cass]
VZ-200	INKEYS	INPUT [string;] var [,var . . .]	INT (exp)	LEFTS (string, length)	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD ["file-name"]	LOG(exp)	MIDS(string, start,[,length])		NEW	NEXT [var]					OUT port,byte	PEEK(addr)	POKE addr,byte	PRINT #"filename", exp[,exp . . .] NB prints to tape
ZX SPECTRUM	INKEYS	INPUT [string:] var	INT (exp)	string (TO finish)	LIST [lineno] Note: will fill screen then ask SCROLL	LLIST [lineno]	LOAD "filename" [code start, length]	LN(exp)	string(start TO finish)		NEW	NEXT var				Consult Microdrive manual	OUT port,byte	PEEK(addr)	POKE addr,byte	Consult manual

MACHINE / STANDARD MICROSOFT	RANDOMIZE Reset random number generator.	READ Read from data statements into specified variables	RENUM Change program line numbers.	RESTORE Resets pointer to facilitate re-reading of DATA statements.	RESUME Return from ON ERROR sub-routine to stmt that caused error.	RETURN Return from sub-routine to statement following last GOSUB executed.	RIGHT$ Returns specified no. of characters starting at end of string.	RND Generates a random number.	RUN Execute a program.	SAVE Save a program either onto disk or tape.	SGN Returns 1 if exp>0 0 if exp=0 −1 if exp<0.	STRING$ Returns a string of specified length containing specified character.	STR$ Converts a numeric expression to a string.	SYSTEM Close files for return to operating system.	TROFF Trace off.	TRON Trace on.	USR Calls an assembler language sub-routine which returns one value.	WAIT Suspend program execution for specified time.	WHILE/END Execute statements in WHILE/WEND loop as long as exp is true	WIDTH Sets printer carriage/screen width.
STANDARD MICROSOFT	RANDOMIZE [exp]	READ var [,var . . .]	RENUM [lineno, val]	RESTORE	RESUME	RETURN	RIGHTS (string, length)	RND[exp]	RUN [lineno]	SAVE filename	SGN(exp)	STRINGS(length string)	STRS(exp)	SYSTEM	TROFF	TRON	USR(parameter)	WAIT port,mark [,select]	WHILE exp WEND	WIDTH(val)
AMSTRAD 464/664/6128	RANDOMIZE (exp)	READ var [,var . . .]	RENUM [new start no ,old start no ,inc]	RESTORE [lineno]	RESUME [line no] or RESUME NEXT	RETURN	RIGHTS (string, length)	RND[exp]	RUN ["filename"] [line no]	SAVE "filename" [, file type] [,binary parms]	SNG(exp)	STRINGS (length,string)	STRS(exp)		TROFF	TRON	CALL addr [,parms]	WAIT addr, mask [,inversion]	WHILE exp WEND	WIDTH exp
APPLE II		READ var [,var . . .]		RESTORE	RESUME	RETURN	RIGHTS(string, length)	RND(-exp)	RUN [lineno]	SAVE [filename [,binary parms]]	SGN(exp)		STRS(exp)		NOTRACE	TRACE	USR(parameter)	WAIT addr, exp [,exp]		POKE 32, left margin: POKE 33, screen width
ATARI	RND (-exp)	READ var [,var . . .]		RESTORE [lineno]		RETURN	string(start NB: not strictly equivalent	RND(exp) Note: exp is a dummy variable	RUN [lineno]	CSAVE "filename" [cass] or SAVE ["fileno:filename" [disk]	SGN(exp)		STRS(exp)	BYE NB: note equivalent			USR(addr,parameter [,parameter . . .])			POKE 82, val [left margin]: POKE 83, val [right margin]
BBC	RND (-time)	READ var [,var . . .]	RENUMBER [start lineno][,interval]	RESTORE [lineno]		RETURN	RIGHTS(string, length)	RND(exp)	RUN	SAVE "filename" Note: see note under LOAD	SGN(exp)	STRINGS(length, string)	STRS(exp)	*DISK NB: disk-handling done through Basic so not true eq.	TRACE OFF	TRACE ON	USR(parameter)	(no WAIT stmt but see INKEYS)	REPEAT stmt UNTIL exp Note: reverse logic	WIDTH val Note: 0='unlimited'
COMMODORE 64 & VIC 20	RND (-TI)	READ var [,var . . .]		RESTORE		RETURN	RIGHTS(string, length)	RND(exp) Note: exp is a dummy for VIC	RUN [lineno]	SAVE ["filename"] [cass] or SAVE "filename",8 [disk]	SNG(exp)		STRS(exp)				USR(parameter)	WAIT addr, exp[,exp]		
IBM PC-BASIC A	RANDOMIZE (exp)	READ var [,var, . . .]	RENUM [new start no] [,old start no] [,inc]	RESTORE [lineno]	RESUME	RETURN [lineno]	RIGHTS (exp, length)	RND(exp)	RUN [lineno]	SAVE "filename" [,A,P]	SGN(exp)	STRINGS (length,string)	STRS(exp)	SYSTEM	TROFF	TRON	USR(exp)	WAIT port, exp[,exp]	WHILE exp WEND	WIDTH exp
MEMOTECH MTX 512	RAND (exp)	READ var [,var, . . .]		RESTORE [lineno]		RETURN	RIGHTS (exp, length)	RND(exp)	RUN [lineno]	SAVE "filename"	SGN(exp)		STRS(exp)	BYE			USR(parameter)	PAUSE (delay)		
MICROBEE		READ [(lineno.)] var[,var]	RENUM (new-start (,increment (,start-line (,finish-line))))	RESTORE (lineno.)		RETURN	var(;LEN(var)-n-1) -n - number of characters required	RND	RUN	SAVE "filename" - 200 bpi SAVEF "filename" - 1200 bpi	SGN(real-exp)	PRINT [An m] -n=length of string; m=ASCII code of character	STR(exp)		TRACE OFF	TRACE ON	USR(address (, integer-exp))	PLAY 0, int (1(int(255; 1-1/8 second)		ZONE (integer-exp) 1 < integer-exp < 16
MSX BASIC	RND (-time)	READ var [,var, . . .]	RENUM [new start no] [,old start no] [,inc]	RESTORE [lineno]	RESUME [lineno] or RESUME TEXT	RETURN [lineno]	RIGHTS (STRING, length)	RND Note: X=dummy val	RUN [lineno]	CSAVE "filename"	SGN(exp)	STRINGS (length,string)	STRS(exp)		TROFF	TRON	USR(parameter)			WIDTH(exp)
TANDY 100	RND (-exp)	READ var [,var . . .]		RESTORE [lineno]	RESUME [lineno] or RESUME NEXT	RETURN	RIGHTS(string, length)	RND (exp)	RUN [lineno]	CSAVE ["filename"]	SNG (exp)	STRINGS (length,string)	STRS(exp)	CALL 0 — similar effect						
TANDY COLOR		READ var [,var . . .]	RENUM [lineno, start, interval]	RESTORE		RETURN	RIGHTS(string, length)	RND(exp)	RUN [lineno]	CSAVE "filename"	SGN(exp)	STRINGS(length, string)	STRS(string)		TROFF	TRON	USR(parameter)			
SINCLAIR QL	RANDOMISE [exp]	READ var [,var . . .]	RENUM [[old start no] [TO old end no;] [new start no], [inc]]	RESTORE [lineno]	RETRY	RETURN exp	stringname (first car to last char	RND [exp TO exp]	RUN [lineno]	SAVE "filename" [lineno [to lineno]] . . .	SGN(exp)	FILLS(string, length)	Note: conversion automatic on assignment	Note: disk handling done through Basic			See CALL (or use EXEC)	PAUSE [delay]	REPEAT name IF cond EXIT name END REPEAT name	WIDTH [#chanel,] exp
TRS-80 II/SYSTEM 80	RANDOM	READ var [,var . . .]	RENUM start, interval Note: System 80 only	RESTORE [lineno/ exp]	RESUME [lineno/exp]	RETURN	RIGHTS(string, length)	RND(exp)	RUN [lineno]	CSAVE "filename" [cass] or SAVE "filename" [disk/ floppy tape]	SGN(exp)	STRINGS(length, string)	STRS(exp)		TROFF	TRON	USR(parameter)			
VZ-200		READ var [,var . . .]		RESTORE		RETURN	RIGHTS(string, exp)	RND(exp) NB: Nonstandard — see VZ200 manual P58	RUN [lineno]	CSAVE ["filename"]	SGN(exp)		STRS(exp)				USR(parameter)			
ZX SPECTRUM	RAND (exp)	READ var [,var . . .]		RESTORE [lineno/ exp]		RETURN	string(TO start)	RND	Run [lineno/ var/exp]	SAVE "filename" [CODE start, length] [SCREENS]	SGN(exp)		STRS(exp)				USR addr	PAUSE no. of frames (50/second)		

BASIC CONVERTER

Australian Personal Computer
WALLCHART

Those rotten manufacturers still insist on making machines that won't talk to each other in the same language. Some enlightened people are having a go with MSX, but in the meantime and in response to overwhelming demand, here's the 1986 APC Converter Chart.

We've added seven new Basics, covering the latest machines, and revised and updated the chart. It isn't possible, of course, to cover every micro nor every command supported by each of the machines included. What this chart aims to do is to provide an at-a-glance syntax comparison using Microsoft Basic as a reference point. The chart won't convert programs for you but it will save you the trouble of getting hold of piles of manuals — and even when you've got them it's often the beginning, not the end of your worries.

To use the chart, first check that the keyword you want isn't in the box on the right. If it is, then you're lucky: it's one of the few that IS the same on every single machine featured here. Due to t limited amount of information we can squeeze into each box hasn't always been possible to indicate the full power of eve ment. It should be assumed, therefore, that we're dealing wi most common uses of each statement, and that other uses available.

Something to watch out for: identical syntax may have diffe effects on different machines. Watch out especially for SYST and RND.

You'll notice we haven't included anything on sound and gra that's too complicated for a quick reference chart, but we've the subject in a series of articles which will appear in APC a range of machines.

BASIC RESERVED WORD

STANDARD MICROSOFT / MACHINE	ASC	ATN	AUTO	CALL	CHAIN	CHR$	CLEAR	CLOSE	CONT
	Returns ASCII value of first character of string.	Arctangent of expression.		Calls assembler language sub-routine	Call a new program & pass variables to it.	Gives one-char string with ASCII code of exp.	CLEAR all [or selected] variables.	Closes disk files — closes all files if no specification.	continue program execution
STANDARD MICROSOFT	ASC (string)	ATN (exp)	AUTO [lineno, val]	CALL var[,var, var . . .]	CHAIN "filename"	CHR$ (exp)	CLEAR [exp,exp]	CLOSE	CONT
AMSTRAD 464/664/6128	ASC (string)	ATN (exp)	AUTO [lineno, incl]	CALL addr [,parms]	CHAIN "filename" [,lineno,exp]	CHR$ (exp)	CLEAR [all] ERASE [list of] var NB: clears and removes arrays	CLOSEIN [NB cassette input file] CLOSEOUT [NB cassette output file]	CONT
APPLE II	ASC (string)	ATN (exp)		CALL addr [var,var . . .]	CHAIN "filename"	CHR$ (exp)	CLEAR	CLOSE [filename]	CONT
ATARI	ASC (string)	ATN (exp)		USR (addr [,var,var . . .])	RUN "C:" NB: program must have been saved using SAVE "C:"	CHR$ (exp)	CLR	CLOSE [#fileno, var,var,filename]	CONT
BBC	ASC (string)	ATN (exp)	AUTO [lineno, val]	CALL addr, [var][,var, . . .]	CHAIN "filename"	CHR$ (exp)	CLEAR	CLOSE #fileno Note: CLOSE #0 to close all files	NB not available: use GOTO lineno
COMMODORE 64 & VIC 20	ASC (string)	ATN (exp)		SYS addr		CHR$ (exp)	CLR	CLOSE #fileno	CONT
IBM PC-BASIC A	ASC (string)	ATN (exp)	AUTO [lineno] [,inc]	CALL addr [var, . . .,var]	CHAIN filename	CHR$ (exp)	CLEAR	CLOSE [#] [filename]	CONT
MEMOTECH MTX 512	ASC (string)	ATN (exp)	AUTO [lineno] [,inc]	USR (addr)		CHR$ (exp)	CLEAR	DISC CLOSE # channel no	CONT
MICROBEE	ASC (string)	ATAN (real-exp)	AUTO [lineno, val]			CHR (integer-exp)	STR$ (int-exp) Note: set limits for string memory		CONT
MSX BASIC	ASC (string)	ATN (exp)	AUTO [lineno ,inc]	USR (addr)		CHR$ (exp)	CLEAR [var]	DISK basic only	CONT
TANDY 100	ASC (string)	ATN (exp)		CALLadr [,param,param]		CHR$ (exp)	CLEAR [(exp)] — Clears string space	CLOSE [fileno] if exp is given	CONT
TANDY COLOR	ASC (string)	ATN (exp)		EXEC addr		CHR$ (exp)	CLEAR [(exp)] clears string space if exp is given	CLOSE #-fileno	CONT
SINCLAIR QL	CODE (str)	ATAN (exp)	AUTO [lineno] [,inc]	CALL addr [,parms]	M RUN "filename"	CHR$ (exp)	CLEAR	CLOSE # channel	CONTINUE
TRS-80 II/SYSTEM 80	ASC (string)	ATN (exp)	AUTO [lineno, val]			CHR$ (exp)	CLEAR [(exp)] Note: Clears string space if exp given	[depends on OS; consult OS manual]	CONT
VA-200	ASC (string)	ATN (exp)				CHR$ (exp)	CLEAR [exp] N clears string space		CONT
ZX SPECTRUM	CODE (string)	ATN (exp)		LET var=USR addr		CHR$ (exp)	CLEAR [var]	CLOSE # channel no	CONT

STANDARD MICROSOFT / MACHINE	INKEY$	INPUT	INT	LEFT$	LIST	LLIST	LOAD	LOG	MID$
	Returns character typed at keyboard or null if no character used	Read data from terminal	Evaluates expression for largest integer contained.	Returns specified no. of characters starting at beginning of string.	List specified program lines at terminal.	List specified program lines at printer.	Load a program file into memory.	Natural logarithm of expression.	Gives specified n of characters to right of start position in string.
STANDARD MICROSOFT	INKEY$	INPUT [string:] var[,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno, lineno]	LLIST [lineno, lineno]	LOAD ["filename"]	LOG(exp)	MID$(string,start [,length])

TER CHART '86

atured here. Due to the
queeze into each box, it
the full power of every state-
hat we're dealing with the
and that other uses may be

ntax may have different
t especially for SYSTEM

ng on sound and graphics:
nce chart, but we've covered
will appear in APC for

SHARED INSTRUCTIONS

ABS	(exp)	
COS	(exp)	
END		NB not available on QL
FOR	var=exp	TO exp [STEP exp]
LEN	(string)	NB Space *must* be present for Memotech
LET	var=EXP	NB LET obligatory after THEN and ELSE on MicroBee
REM	text	
SIN	(exp)	
SQR	(exp)	
STOP		
TAN	(exp)	
VAL	(exp)	NB not available on QL

ABBREVIATIONS USED IN THIS CHART:

addr	= address
exp	= expression
parm(s)	= parameter(s)
stmt	= statement
var	= variable

Square brackets [] indicate optional code.

VORDS & FORMATS

E files all files ication.	CONT continue program execution	DATA Lists data to be used in a READ statement	DEF Define arithmetic string function.	DELETE Delete specified program lines.	DIM Allocates space for arrays, specifies max subscript values.	EDIT Edit a program line.	EXP Raises to power of expression.	FRE Returns remaining memory space.	GET Read a record from disk or tape file.	GOSUB Branch to a Basic subroutine.	GOTO Branch to a specified line number.	IF/THEN/ELSE If exp is true stmt is executed. If not ELSE or following line is executed.
	CONT	DATA const [,const, . . .]	DEF FNvar [(var,var . . .)] =exp	DELETE lineno [,lineno]	DIM var(sub) [,var (sub), . . .]	EDIT lineno	EXP(exp)	FRE(exp)	GET [#] lineno [,record no] or INPUT#fileno, var [,var . . .] for sequential files	GOSUB lineno	GOTO lineno	If exp THEN [ELSE stmt]
NB ut file] [NB tput file]	CONT	DATA const [,const, . . .]	DEF FN(var) [(var,var, . . .)]=exp	DELETE [line no-line no]	DIM [list of] var (dimension list)	EDIT lineno	EXP(exp)	FRE(exp) Note: exp is a dummy variable	LINE INPUT#, [;string][var, . . .]	GOSUBlineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
name)	CONT	DATA const [,const, . . .]	DEF FNvar (var)=exp	DEL lineno, lineno	DIM var(sub) [,var(sub) . . .]	[screen editing using ESC key]	EXP(exp)	FRE(exp) Note: exp is a dummy variable	INPUT var [,var . . .] NB: Get var(s) from current input device	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
fileno, ame]	CONT	DATA const [,const, . . .]			DIM var (sub) [,var (sub) . . .] NB: dimension ALL strings	[cursor editing]	EXP(exp)	FRE(exp) Note: exp is a dummy variable	GET #lineno, record	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	If exp THEN stmt Note: no ELSE
leno E #0 to es	NB not available: use GOTO lineno	DATA const [,const, . . .]	DEF FNvar =exp	DELETE lineno [-lineno]	DIM var(sub) [,var(sub) . . .]	[cursor editing]	EXP(exp)	HIHEM-TOP Use PRINT	INPUT #lineno, record [,record . . .]	GOSUB lineno/ [var][exp]	GOTO lineno/ [var][exp]	If exp THEN stmt [ELSE stmt]
leno	CONT	DATA const [,const, . . .]	DEF FNvar =exp		DIM var(sub) [(sub) . . .]	[cursor editing]	EXP(exp)	FRE(exp) Note: exp is a dummy variable	GET #fileno, var	GOSUB lineno	GOTO lineno	IF exp THEN stmt Note: no ELSE
]	CONT	DATA const [,const, . . .]	DEF FNvar [(parms)]=exp	DELETE [lineno] [-lineno]	DIM var(sub) [,var(sub) . . .]	EDIT lineno	EXP(exp)	FRE(exp) Note: exp is a dummy variable	GET [#] filename [,rec no]	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
E #	CONT	DATA const [,const, . . .]	DEF FNvar [(parms)]=exp		DIM var(sub) [,var(sub)] . . .	EDIT lineno	EXP(exp)		Disc INPUT # channelno	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
	CONT	DATA expr (,exp ("exp"))	FNn=exp	DELETE lineno. (lineno.)	DIM var(sub) [,var(sub) . . .]	EDIT (lineno.)	EXP(exp)	FRE(0) mem. space FRE($) str. space		GOSUB NB: sq. br. significant	GOTO lineno	If exp THEN stmt [ELSE stmt]
only	CONT	DATA const [,const, . . .]	DEF FN(var) [(parms)]=exp	DELETE [lineno] [-lineno]	DIM var(sub) [,var(sub) . . .]	[cursor editing]	EXP(exp)	FRE(exp) Note: exp is a dummy variable		GOSUB lineno	GOTO lineno	IF exp THEN stmt [ELSE stmt]
no] ven	CONT	DATA const [,const . . .]			DIM var(sub) [,var(sub) . . .]	EDIT lineno [-lineno]	EXP (exp)	FRE (exp)	INPUT #fileno, var [,var . . .]	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
fileno	CONT	DATA const [,const, . . .]	DEF FNvar (var)=exp	DELETE lineno-lineno	DIM var(sub) [,var(sub) . . .]	EDIT lineno	EXP(exp)	MEM	INPUT #-fileno, record	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
channel	CONTINUE	DATA const [,const, . . .]	DEF FNvar [S/%]=exp: END DEF	DLINE lineno [[TO]lineno]	DIM var(sub) [,var(sub) . . .]	EDIT lineno [,step]	EXP(exp)		INKEYS (#channel)	GOSUB lineno/ var/exp	GOTO lineno/ var/exp	IF exp THEN stmt [ELSE stmt][END IF]
OS; manual]	CONT	DATA const [,const, . . .]	Various DEF statements available but none equivalent	DELETE lineno [-lineno]	DIM var(sub)	EDIT lineno	EXP(exp)	FRE(exp) [TRS-80] or MEM [System 80]	INPUT #-fileno, record [,record . . .]	GOSUB lineno	GOTO lineno	If exp THEN stmt [ELSE stmt]
	CONT	DATA const [,const, . . .]			DIM var(sub) [,var(sub) . . .]		EXP(exp)		INPUT # filename var[,var . . .] NB: Gets record from tape	GOSUB lineno	GOTO lineno	IF exp THEN stmt [ELSE stmt]
	CONT	DATA const [,const, . . .]	DEF FN(var) [(var,var . . .)]		DIM var(sub)	EDIT [lineno] Note: cursor line by default	EXP(exp)		Consult Microdrive manual	GOSUB lineno [exp]	GOTO lineno [exp]	IF exp THEN stmt Note: no ELSE

rithm n.	MID$ Gives specified no. of characters to the right of start position in string.	NAME Rename a file.	NEW Delete current program & data from memory.	NEXT End of FOR/NEXT loop.	ON ERROR Error trap subroutine.	ON/GOSUB GOTO lineno specified by evaluation of expression.	ON/GOTO GOTO lineno specified by evaluation of expression.	OPEN Open disk file.	OUT Put specified byte to specified output port.	PEEK Read byte from specified memory location.	POKE Put specified byte to specified memory address.	PRINT Write data to disk tape or terminal.
	MIDS(string,start [,length])	NAME "filename" AS "filename"	NEW	NEXT var [,var . . .]	ON ERROR GOTO lineno	On exp GOSUB lineno [,lineno . . .]	On exp GOTO lineno [,lineno . . .]	OPEN mode [#] fileno "filename"	OUT port, byte	PEEK(addr)	POKE addr,byte	PRINT [[#] fileno] [exp] [,exp . . .]

MACHINE									
AMSTRAD 464/664/6128	INKEY$	INPUT [#no] [prompt] [var [list]]	INT (exp)	LEFT$ (string, length)	LIST [lineno, lineno]	LIST [lineno, lineno] #8	LOAD ["filename"] [,addr]	LOG(exp) Note: LOG10(exp) gives Log base 10	MID$(string, start, length)
APPLE II	Get var	INPUT [string,] var [,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno, lineno] Note: '__' may be used in place of ','	PR # slotno: LIST [lineno,lineno]: PR # 0	LOAD filename	LOG(exp)	MID$(string, start[,length])
ATARI		INPUT [string,] var [,var . . .]	INT (exp)	string (start, length)	LIST [lineno, lineno]	LIST [lineno, lineno]	CLOAD ["filename"] [cases] or LOAD "fileno-filename" [disk]	LOG(exp)	string(start [,length])
BBC	Get var [unlimited] time] or INKEY$ (time) Note: 100ths sec.	INPUT [string;] var [,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno, lineno]	CTRL-B then LIST [lineno-lineno]	LOAD "filename" Note '*DISK' or '*TAPE' to select device	LN(exp) NB: LOG(exp) gives common rather than natural log	MID$(string, start[,length])
COMMODORE 64 & VIC 20	GET var	INPUT [string:] var [,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno-lineno]	OPEN 4,4:CMD4: LIST [lineno-lineno] OPEN 3,4:CMD3: LIST [lineno-lineno]	LOAD ["filename"] [cass] or LOAD "filename",8 [disk]	LOG(exp)	MID$(string, start[,length])
IBM PC-BASIC A	var $=INKEY$	INPUT [prompt] var [,var]	INT (exp)	LEFT$ (string, length)	LIST [1st line] [-last line] [,filespec]	LIST [lineno, lineno]	LOAD filename [,R]	LOG(exp)	MID$(string, start[,length])
MEMOTECH MTX 512	var $=INKEY$	INPUT [prompt] var [,var]	INT (exp)	LEFT$ (string, length)	LIST [1st line] [,last line]	LIST [lineno, lineno]	LOAD "filename"	LN(exp)	MID$(string, start[,length])
MICROBEE	KEY	INPUT [string] var [,var]	INT (real-exp)	var(;1, length)	LIST [lineno [,lineno]] forceloads	LLIST (lineno. (,lineno))	LOAD (U) (?) ("filename") LOAD U	LOG(real-exp)	var(;n,n+m-1) -n-start character, m-length
MSX BASIC	var $=INKEY$	INPUT [prompt] var [,var]	INT (exp)	LEFT$ (string, length)	LIST [1st line] [-last line]	LLIST [lineno, lineno]	CLOAD filename	LOG(exp)	MID$(string, start[,length])
TANDY 100	INKEY$	INPUT [string;] var [,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno-lineno]	CLOAD lineno]	LOG (exp) ["filename"]	MID$(string,	start[,length])
TANDY COLOR	INKEY$	INPUT [string;] var [,var]	INT (exp)	LEFT$ (string, length)	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD ["filename"]	LOG(exp)	MID$(string, start[,length])
SINCLAIR QL	INKEY$ (#channel ,time)	INPUT [# channel,] [prompt] [var [,var] . . .]	INT (exp)	string (TO finish)	LIST [#channel] 1st line [To last line]	[LIST #[channel] [lineno] [To lineno]	LOAD device [inc. filename]	LN(exp) Note: LOG10(exp) gives common rather than natural log	string(start TO finish)
TRS-80 II/SYSTEM 80	INKEY$	INPUT [string;] var [,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD "[filename]" [cass] or LOAD "filename" [disk/ floppy tape]	LOG(exp)	MID$(string, start,length)
VZ-200	INKEY$	INPUT [string;] var [,var . . .]	INT (exp)	LEFT$ (string, length)	LIST [lineno-lineno]	LLIST [lineno-lineno]	CLOAD ["file-name"]	LOG(exp)	MID$(string, start,[,length])
ZX SPECTRUM	INKEY$	INPUT [string;] var	INT (exp)	string (TO finish)	LIST [lineno] Note: will fill screen then ask SCROLL	LLIST [lineno]	LOAD "filename" [code start, length]	LN(exp)	string(start TO finish)

MACHINE	**RANDOMIZE** Reset random number generator.	**READ** Read from data statements into specified variables	**RENUM** Change program line numbers.	**RESTORE** Resets pointer to facilitate re-reading of DATA statements.	**RESUME** Return from ON ERROR sub-routine to stmt that caused error.	**RETURN** Return from sub-routine to statement following last GOSUB executed.	**RIGHT$** Returns specified no. of characters starting at end of string.	**RND** Generates a random number.	**RUN** Execute a program.
STANDARD MICROSOFT	RANDOMIZE [exp]	READ var [,var . . .]	RENUM [lineno, val]	RESTORE	RESUME	RETURN	RIGHT$ (string, length)	RND[exp]	RUN [lineno]
AMSTRAD 464/664/6128	RANDOMIZE (exp)	READ var [,var . . .]	RENUM [new start no ,old start no ,inc]	RESTORE [lineno]	RESUME [line no] or RESUME NEXT	RETURN	RIGHT$ (string, length)	RND[exp]	RUN ["filename"] [line no]
APPLE II		READ var [,var . . .]		RESTORE	RESUME	RETURN	RIGHT$(string, length)	RND(-exp)	RUN [lineno]
ATARI	RND (-exp)	READ var [,var . . .]		RESTORE [lineno]		RETURN	string(start NB: not strictly equivalent	RND(exp) Note: exp is a dummy variable	RUN [lineno]
BBC	RND (-time)	READ var [,var . . .]	RENUMBER [start lineno][,interval]	RESTORE [lineno]		RETURN	RIGHT$(string, length)	RND(exp)	RUN
COMMODORE 64 & VIC 20	RND (-TI)	READ var [,var . . .]		RESTORE		RETURN	RIGHT$(string, length)	RND(exp) Note: exp is a dummy for VIC	RUN [lineno]
IBM PC-BASIC A	RANDOMIZE (exp)	READ var [,var, . . .]	RENUM [new start no] [,old start no] [,inc]	RESTORE [lineno]	RESUME	RETURN [lineno]	RIGHT$ (exp, length)	RND(exp)	RUN [lineno]
MEMOTECH MTX 512	RAND (exp)	READ var [,var, . . .]		RESTORE [lineno]		RETURN	RIGHT$ (exp, length)	RND(exp)	RUN [lineno]
MICROBEE		READ [(lineno.)] var[,var]	RENUM (new-start (,increment (,start-line (,finish-line))))	RESTORE (lineno.)		RETURN	var(;LEN(var)-n-1) -n - number of characters required	RND	RUN
MSX BASIC	RND (-time)	READ var [,var, . . .]	RENUM [new start no] [,old start no] [,inc]	RESTORE [lineno]	RESUME [lineno] or RESUME TEXT	RETURN [lineno]	RIGHT$ (STRING, length)	RND Note: X=dummy val	RUN [lineno]
TANDY 100	RND (-exp)	READ var [,var . . .]		RESTORE [lineno]	RESUME [lineno] or RESUME NEXT	RETURN	RIGHT$(string, length)	RND (exp)	RUN [lineno]
TANDY COLOR		READ var [,var . . .]	RENUM [lineno, start, interval]	RESTORE		RETURN	RIGHT$(string, length)	RND(exp)	RUN [lineno]
SINCLAIR QL	RANDOMISE [exp]	READ var [,var . . .]	RENUM [[old start no] [TO old end no;] [new start no], [inc]]	RESTORE [lineno]	RETRY	RETURN exp	stringname (first car to last char	RND [exp TO exp]	RUN [lineno]
TRS-80 II/SYSTEM 80	RANDOM	READ var [,var . . .]	RENUM start, interval Note: System 80 only	RESTORE [lineno/ exp]	RESUME [lineno/exp]	RETURN	RIGHT$(string, length)	RND(exp)	RUN [lineno]
VZ-200		READ var [,var . . .]		RESTORE		RETURN	RIGHT$(string, exp)	RND(exp) NB: Nonstandard — see VZ200 manual P58	RUN [lineno]
ZX SPECTRUM	RAND (exp)	READ var [,var . . .]		RESTORE [lineno/ exp]		RETURN	string(TO start)	RND	Run [lineno/ var/exp]

0(exp) ase 10	MIDS(string, start, length)		NEW	NEXT [var][,var, . . .]	ON ERROR GOTO lineno	On exp GOSUB lineno [,lineno, . . .]	ON exp GOTO lineno [,lineno, . . .]	OPEN mode [#] fileno "filename"	OUT port, byte	PEEK(addr)	POKE addr,byte	PRINT [#]fileno] [exp][,exp . . .]
	MIDS(string, start[,length])	RENAME oldname, newname	NEW	NEXT [var,var . . .]	ONERR GOTO lineno	On exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN filename, parm		PEEK(addr)	POKE addr,byte	PRINT exp [,exp . . .] NB: prints to current output device
	string(start [,length])		NEW	NEXT var	TRAP lineno/ var/exp	ON exp GOSUB lineno/var/exp [,lineno/var/exp . . .]	ON exp GOTO lineno/var/exp [,lineno/var/exp . . .]	OPEN #fileno, mode control code, filename	[not equivalent]	PEEK(addr)	POKE addr,byte	PRINT #device, exp[,exp]
: ives her log	MIDS(string, start[,length])		NEW Note: under cert. circum. may be recovered using OLD	NEXT [var][,var . . .]	ON ERROR stmt [OFF]	ON exp[var] GOSUB lineno [,lineno . . .]	ON exp[var] GOTO lineno [,lineno . . .]	fileno=OPENIN [to read] or fileno= OPENOUT [to write]		?addr NB: '?' does NOT mean 'print' in BBC Basic	?addr,byte	PRINT #fileno record [,record . . .]
	MIDS(string, start[,length])	OPEN 1,8,15, "RO: filename= filename" [disk only] N/A	NEW	NEXT [var][,var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN #exp, fileno, mode, "filename"		PEEK(addr)	POKE addr,byte	PRINT #fileno record [,record . . .]
	MIDS(string, start[,length])	NAME filename AS filename	NEW	NEXT [var,var, . . .]	ON ERROR GOTO lineno	ON [exp:COM: KEY:PEN:STRING] GOSUB lineno	ON exp GOTO lineno	OPEN filename [FOR Mode] AS [#] filename [LEN=rec)	OUT port,data	PEEK(addr)	POKE addr,byte	PRINT [exp][;]
	MIDS(string, start[,length])	DISC REN string=string	NEW	NEXT var		ON exp GOSUB lineno	ON exp GOTO lineno	DISC OPEN # channel no, "filename", filetype, record length	OUT port,data	PEEK(addr)	POKE addr,byte	[DISC] PRINT [#channelno,] print list
p)	var(;n,n+m-1) -n-start character, m-length		NEW	NEXT var NEXT *var lineno. -exits loop before completion	ON ERROR GOTO lineno.	ON exp GOSUB ([exp(,exp)])lineno (,([exp . . .	ON exp GOTO lineno [,lineno . . .]		OUT port,byte	PEEK(addr)	POKE addr, byte	PRINT list of arguments
	MIDS(string, start[,length])		NEW	NEXT [var][,var, . . .]	ON ERROR GOTO lineno	ON exp GOSUB lineno [,lineno, . . .]	ON exp GOTO lineno [,lineno, . . .]	OPEN "device: filename" for OUTPUT [for INPUT] AS#var	OUT port,data	PEEK(addr)	POKE addr,byte	PRINT [#fileno,] var[,var, . . .]
,	start[,length])	NEW		NEXT [var,var . . .]	ON ERROR GOTO lineno	On exp GOTO lineno [,lineno . . .]	OPEN "filename" lineno	OUT port,byte FOR (mode)	PEEK(addr)	POKE addr,byte	PRINT [#fileno]	[exp,exp . . .]
	MIDS(string, start[,length])		NEW	NEXT [var][,var . . .]		ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	OPEN mode, #- fileno "filename"		PEEK(addr)	POKE addr,byte	PRINT #-fileno. exp[,exp . . .]
0(exp) on rather log	string(start TO finish)		NEW	NEXT var[,var]/ END FOR var[,var]	WHEN ERROR var: END WHEN Note: OS JS	ON var GOSUB lineno [,lineno]	ON var GOTO lineno [,lineno]	OPEN #channel, "filename"		PEEK [or W or L](addr)	POKE [or W or L](addr),byte	PRINT [#channel,] exp[,exp] . . .
	MIDS(string, start,length)	[depends on OS; consult OS manual]	NEW	NEXT [var][,var . . .]	ON ERROR GOTO lineno	ON exp GOSUB lineno [,lineno . . .]	ON exp GOTO lineno [,lineno . . .]	[depends on OS; consult OS manual.]	OUT port,byte	PEEK(addr)	POKE addr,byte	PRINT #fileno, record [,record . . .] [cass]
	MIDS(string, start,[,length])		NEW	NEXT [var]					OUT port,byte	PEEK(addr)	POKE addr,byte	PRINT #"filename", exp[,exp . . .] NB prints to tape
	string(start TO finish)		NEW	NEXT var				Consult Microdrive manual	OUT port,byte	PEEK(addr)	POKE addr,byte	Consult manual

	RUN Execute a program.	**SAVE** Save a program either onto disk or tape.	**SGN** Returns 1 if exp>0 0 if exp=0 −1 if exp<0.	**STRINGS$** Returns a string of specified length containing specified character.	**STR$** Converts a numeric expression to a string.	**SYSTEM** Close files for return to operating system.	**TROFF** Trace off.	**TRON** Trace on.	**USR** Calls an assembler language subroutine which returns one value.	**WAIT** Suspend program execution for specified time.	**WHILE/END** Execute statements in WHILE/WEND loop as long as exp is true	**WIDTH** Sets printer carriage/screen width.
ber.												
	RUN [lineno]	SAVE filename	SGN(exp)	STRINGS(length string)	STRS(exp)	SYSTEM	TROFF	TRON	USR(parameter)	WAIT port,mark [,select]	WHILE exp WEND	WIDTH(val)
	RUN ["filename"] [line no]	SAVE "filename" [, file type] [,binary parms]	SNG(exp)	STRINGS (length,string)	STRS(exp)		TROFF	TRON	CALL addr [,parms]	WAIT addr, mask [,inversion]	WHILE exp WEND	WIDTH exp
	RUN [lineno]	SAVE [filename [,binary parms]]	SGN(exp)		STRS(exp)		NOTRACE	TRACE	USR(parameter)	WAIT addr, exp [,exp]		POKE 32, left margin: POKE 33, screen width
lote: mmy	RUN [lineno]	CSAVE "filename" [cass] or SAVE ["fileno:filename" [disk]	SGN(exp)		STRS(exp)	BYE NB: note equivalent			USR(addr,parameter [,parameter . . .])			POKE 82, val [left margin]: POKE 83, val [right margin]
	RUN	SAVE "filename" Note: see note under LOAD	SGN(exp)	STRINGS(length, string)	STRS(exp)	*DISK NB: disk-handling done through Basic so not true eq.	TRACE OFF	TRACE ON	USR(parameter)	(no WAIT stmt but see INKEYS]	REPEAT stmt UNTIL exp Note: reverse logic	WIDTH val Note: 0='unlimited'
s a VIC	RUN [lineno]	SAVE ["filename"] [cass] or SAVE "filename",8 [disk]	SNG(exp)		STRS(exp)				USR(parameter)	WAIT addr, exp[,exp]		
	RUN [lineno]	SAVE "filename" [,A,P]	SGN(exp)	STRINGS (length,string)	STRS(exp)	SYSTEM	TROFF	TRON	USR(exp)	WAIT port, exp[,exp]	WHILE exp WEND	WIDTH exp
	RUN [lineno]	SAVE "filename"	SGN(exp)		STRS(exp)	BYE			USR(parameter)	PAUSE (delay)		
	RUN	SAVE "filename" - 200 bpi SAVEF "filename" - 1200 bpi	SGN(real-exp)	PRINT [An m] -n=length of string; m=ASCII code of character	STR(exp)		TRACE OFF	TRACE ON	USR(address (, integer-exp))	PLAY 0, int (1(int(255; 1- 1/8 second)		ZONE (integer-exp) 1 < integer-exp < 16
ummy	RUN [lineno]	CSAVE "filename"	SGN(exp)	STRINGS (length,string)	STRS(exp)		TROFF	TRON	USR(parameter)			WIDTH(exp)
	RUN [lineno]	CSAVE ["filename"]	SNG (exp)	STRINGS (length,string)	STRS(exp)	CALL 0 — similar effect						
	RUN [lineno]	CSAVE "filename"	SGN(exp)	STRINGS(length, string)	STRS(string)		TROFF	TRON	USR(parameter)			
0 exp]	RUN [lineno]	SAVE "filename" [lineno [to lineno]] . . .	SGN(exp)	FILLS(string, length)	Note: conversion automatic on assignment	Note: disk handling done through Basic			See CALL (or use EXEC)	PAUSE [delay]	REPEAT name IF cond EXIT name END REPEAT name	WIDTH [#chanel,] exp
	RUN [lineno]	CSAVE "filename" [cass] or SAVE "filename" [disk/ floppy tape]	SGN(exp)	STRINGS(length, string)	STRS(exp)		TROFF	TRON	USR(parameter)			
B: —	RUN [lineno]	CSAVE ["filename"]	SGN(exp)		STRS(exp)				USR(parameter)			
	Run [lineno/ var/exp]	SAVE "filename" [CODE start, length] [SCREENS]	SGN(exp)		STRS(exp)				USR addr	PAUSE no. of frames (50/second)		

VZ200

VZ-200 CASSETTE INLAYS

This program is for all you VZ-200/300 users who have piles of cassette tapes and want to index their contents so it's easy to find what you want This program uses the PP-40, a printer/plotter distributed by Dick Smith, and makes extensive use of the graphics command supported by this printer The program contains comments for those users unfamiliar with the required commands, and for those who are thinking of converting the program

Ian Dutfield,
Cromer, NSW

```
5 GOSUB 1000 'TITLE
10 'CASSETTE TAPE INSERTS
20 'BY IAN DUTFIELD
25 'FOR THE VZ-200
30 '   16/3/85
40.'FOR USE WITH PP40
50 'PRINTER
60 'USE IN 40 COLUMN MODE
70 'SET PRINTER TO TEXT MODE
75 ' CAN BE CONVERTED TO OTHER
   PRINTERS.
80 LPRINT CHR$(12)
90 'CR AND LINEFEED
100 LPRINT CHR$(13)
110 LPRINT CHR$(10)
120 'SET COLOUR TO BLACK
130 'FIRST GO INTO GRAPHIC MODE
140 LPRINT CHR$(18)
150 LPRINT "C0"
160 'RETURN TO TEXT
170 LPRINT CHR$(12)
```

YC Mar 86 p 103
1 of 3

VZ200

```
180 LPRINT " *** CASSETTE INLAYS ***"
190 LPRINT ""
200 ' INTO GRAPHIC MODE TO
210 ' PRINT NUMBERS AND LINES
220 LPRINT CHR$(18)
230 LPRINT "S1"
240 ' SET SIZE
245 ' PRINT NUMBERS
255 LPRINT "P1."
260 'DRAW LINE
270 LPRINT "J446,0"
280 ' GO BACK TO PRINT NUMBER
290 LPRINT "R-200,0"
300 ' PRINT OTHER NUMBER
310 LPRINT "P2."
315 LPRINT"R-292,-30"
320 LPRINT"P3."
321 LPRINT"J446,0"
322 LPRINT"R-200,0"
323 LPRINT"P4."
324 LPRINT"R-292,-30"
325 LPRINT"P5."
326 LPRINT"J446,0"
327 LPRINT"R-200,0"
328 LPRINT"P6."
329 LPRINT"R-292,-30"
330 LPRINT"P7."
340 LPRINT"J446,0"
350 LPRINT"R-200,0"
360 LPRINT"P8."
370 LPRINT"R-292,-30"
380 LPRINT"P9."
390 LPRINT"J446,0"
400 LPRINT" R-200,0"
410 LPRINT"P10."
420 LPRINT"R-315,-30"
430 LPRINT"P11."
440 LPRINT"J446,0"
450 LPRINT"R-200,0"
460 LPRINT"P12."
470 LPRINT"R-315,-30"
```

YC mar 86 p 104

2 of 3.

```
480 LPRINT"P13."
490 LPRINT"J446,0"
500 LPRINT"R-200,0"
510 LPRINT"P14."
520 LPRINT"R-315,-30"
920 SOUND 31,1
930 PRINT"(INVERSE) FINISHED":FOR T=1 TO
1500:NEXT:RUN
1000 'TITLE PAGE
1010 CLS
1030 COLOR 8,0
1035 POKE 30744,1
1040 PRINT@0,"CTRL+Q,CTRL+T*30,CTRL+W";
1045 PRINT@448,"CTRL+E,CTRL+Y*30,CTRL+R"
;
1060 FOR Y=32 TO 416 STEP 32
1070 PRINT@Y,"CTRL+U"
1080 NEXT Y
1090 FOR Y=63 TO 447 STEP 32
2000 PRINT@Y,"CTRL+I"
2010 NEXT Y
2040 PRINT@109,"VZ-200"
2050 PRINT@195,"*** CASSETTE - INLAYS **
*"
2060 PRINT@298,"BY IAN DUTFIELD"
2070 PRINT@388,"PRESS ANY KEY TO CONTINU
E"
2080 IF INKEY$="" THEN GOTO 3000
2090 IF INKEY$="" THEN GOTO 3000
2095 SOUND 31,1:GOTO 4000
3000 SOUND 28,1
3010 PRINT@388,"(INVERSE)PRESS ANY KEY T
O CONTINUE"
3020 SOUND 10,1
3030 GOTO 2070
4000 CLS
4005 POKE 30744,0
4010 INPUT"(INVERSE)SET UP PRINTER AND P
RESS <RET>";P$
4020 PRINT:PRINT:PRINT"PRINTING"
4030 RETURN
```

YC Mar 86 p 105
3 of 3.

Consider the *BASICs*

Tear yourself away from the darkroom and plug-in to Kim Kohen's use of home computers with photography. This combination is only as limited as your imagination.

It seems just about everything we do these days is somehow influenced by a computer. Evidence of this comes in the fact that most of the cameras and lenses we see on sale now, have either been designed by or have as an integral part, something resembling a microprocessor. This has enabled designers to create far more accurate and 'foolproof' cameras.

My involvement with computers is not so complex. I had tinkered with home computers for around 18 months before I started realising their potential for the photographer. I decided that because a great deal of photography is taken up with time in the darkroom, then this was the first area that I should explore. It occurred to me that most photo timers these days are electronic rather than mechanical, so I figured that this would be the first task I would make my computer perform.

I am not a computer expert and I do not have mega-buck super powerful computers. I use probably the cheapest computer on the market, a Dick Smith VZ 300, which at the time of writing was retailing for $99.00. When you consider the cost of the Seiko watch you're probably using as a timer now, the computer would have to be considered great value.

Most home computers use the computer language called BASIC. To get the computer to do exactly what you want, it is necessary to have a program written in this language. There are numerous books available on BASIC and with a little patience it is a fairly straightforward language to understand.

Computer Timing

OK, back to the timers. For quite a while I had been processing films at home using my digital wristwatch as the only form of timer. This is OK in black and white where there are only a couple of steps to time. The problem was that an ever increasing amount of my work was being done on colour transparencies. With the number of steps and the precision required for E6 films, processing them can be quite a handful. This is where the computer comes in.

The thing that computers do best is count. This meant that it was just a matter of getting the computer to time the necessary processing steps for me by making it count. If this sounds difficult, just have a look at a BASIC manual to see how easy it really is. The technique needed is called a 'nested loop'. In a nested loop, the computer is told to count to a certain number, but also to wait a certain time before going to the next number. Confused? Don't worry. Have a look at Table 1 and you should get a better idea of how it works.

Now for my E6 program I had a few definite requirements. I wanted an audible warning as I was approaching a chemistry change, and I wanted a 15 sec. allowance in which to change chemistry. As well as that I wanted a time display so that at any stage during processing I could see at a glance how much time was remaining. It took quite a bit of time but I finally worked out the right program to perform all of these functions.

Table 1. This is the first developer timing step in my E6 program. Lines 176 and 180 display on the screen that the timing has started. Line 182 tells the computer to count from 318 to 0 in steps of −1. This is the development time in seconds. Line 188 is just a display of the time. Lines 190 and 200 are telling the computer to count to 381 BEFORE it counts the the next number from line 182. You see it takes the computer approx 1 second to count to 381. So this means the computer will count down from 318 and take 1 second between counts. Lines 205 to 220 tell the computer to make a sound every second for the last 10 seconds of

It would take too much space to reprint the entire program here. Although it is fairly simple, it does take up quite a bit of room. In the six months I have been using the program, I have processed over 100 rolls of film with a 100% success rate. (That's better than most labs).

Of course the timer principle has many applications. I have just finished a program that times Cibachrome processing and automatically adjusts it's timing according to what temperature the user inputs.

Outside the Darkroom

There are obviously many other applications for home computers in photography. They don't all have to be in the darkroom

This computer plugs into most television sets. It is amazing just how valuable it can be to the photographer, from timing film processing to designing filing sytems.

This is a typical plug-in type memory expansion unit. It gives the user an extra 16K of Random Access Memory. Most of the author's photography programs require 3K of RAM to run.

Australian Photography May 86. 54-55 1 of 2.

the timing step. This is to warn about a chemistry change approaching. Simple isn't it!!

```
176 PRINT @ 74, "FIRST DEV"
180 PRINT @ 135, "TIMING COMMENCED"
182 FOR S=318 TO 0 STEP -1
188 PRINT @ 265, "SECONDS :"S
190 FOR X=1 TO 381
200 NEXT X
205 IF S= 10 THEN 207
206 NEXT S
207 FOR T=30 TO 10 STEP -2
210 SOUND T,6
215 NEXT T
220 CLS
```

Table 2. This program will calculate the hyperfocal distance of your lenses at a certain aperture. It will run on a Dick Smith VZ 200/300 and probably most other home computers.

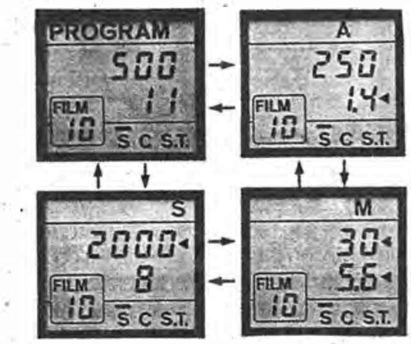
```
 10 REM "HYPERFOCAL DISTANCE"
 20 CLS
 30 PRINT @ 101, "TYPE IN FOCAL LENGTH"
 40 PRINT @ 169, "OF THE LENS."
 50 INPUT L
 60 CLS
 70 PRINT @ 102, "TYPE IN THE MAXIMUM"
 80 PRINT @ 172, "APERTURE."
 90 INPUT F
100 H = 1* (L/F)
110 CLS
120 PRINT @ 100, "HYPERFOCAL DISTANCE IS:"
130 PRINT @ 203, H "METRES"
140 FOR X=1 TO 2300: NEXT X
150 CLS
160 PRINT @ 64, "DO YOU WISH TO CONTINUE OR
    STOP?"
170 PRINT @ 195, "PRESS 'RETURN' TO
    CONTINUE"
180 PRINT @ 231, "PRESS 'Q' TO QUIT"
190 INPUT C$
200 IF C$ = "Q" THEN GO TO 220
210 GOTO 10
220 CLS
230 END
```

either. One really simple program I have written works out the correct aperture to use when using extension tubes for close-ups. Another one lets you work out the hyperfocal length of your various lenses. (Table 2). This in itself is no big deal, but once you know the hyperfocal length of your lens, you can then calculate accurate depth of field tables, and even optimum focusing distances for greatest depth of field. Naturally, you would work this out on your computer too.

So now I've conviced you that without a computer your life's ambition of great photography will not be achieved. Before you rush out and spend a small fortune on the latest whiz-bang computer, there are a few things you should know. The most important thing to do before you buy, is to decide exactly what you want the computer to do. This will allow you to determine the type of computer, and the amount of memory you are likely to need. Most of the photography programs I use require only about 3K of RAM to run. RAM or 'random access memory', is the memory used to store the users programs. The type of programs you run will depend on the amount of RAM you have available. The more complex the program, the more RAM it requires. ROM or 'read only memory, is the computers inbuilt memory. The ROM cannot be programmed by the user. The BASIC language is part of the ROM.

If you are only going to use the computer for simple timing tasks then a computer with 16K of RAM will be quite adequate. However, if you want to run business-type programs like word processors or spreadsheets, then a machine with a larger memory will be necessary. Something to remember here is that many computers RAM can be doubled by the fitting of plug-in memory expansion packs or boards. Go to a recognised computer shop and ask about any particular computer and its functions.

This article is, of course, only scratching the surface. Programs can be written for storing details of where photos were taken, at what aperture, shutter speed, film types etc. Computer filing systems can be designed for instant information on the location of your precious slides or negatives. How about a program for keeping track of how much money you spend on photography each year? You could take it one step further and work out your tax return on the computer. Who knows, the computer may even be a legitimate tax deduction.●



The liquid crystal 'computer' display of the Minolta 7000. These displays will become even more popular in the future.

If anyone is interested in the programs mentioned in this article, or if you have written any programs in BASIC relating to photography write to Kim Kohen, 47 Allingham St, Bankstown 2200. NSW.

2 of 2

ADDENDUM

45 CLEAR 50 : Reset stack ptrs.

VZ Pause

VZ Pause is a short routine for the VZ-200 which enables the computer to be 'paused' at any time. A pause can be initiated by pressing Shift-X. A short beeb will be produced to confirm that a pause has begun and pause can be terminated by pressing Shift-C, and again a short beeb will confirm this. The routine uses interrupts, and so will work with any software that does not disturb these interrupts. To use, type in the routine, and then CSAVE it immediately, as the program self-destructs when run. When the program is run, the pause facility becomes operational.

The program works in the following fashion:

- Lines 10-20 lower the RAMTOP to create space for a short machine language program
- Lines 30-40 set the address for the interrupt exit
- Lines 50-80 POKE the machine language program into the memory
- Line 90 makes the interrupt operational
- Line 100 clears the Basic routine from memory. This is necessary to prevent the system crashing should the routine be RUN twice.

```
10 TM=PEEK(30898)*256+PEEK(30897)-35
20 POKE30897,TM-INT(TM/256)*256:POKE30898,INT(TM/256)
30 TM=TM+1                              :'NEXT ADDR.
40 POKE30846,TM-INT(TM/256)*256:POKE30847,INT(TM/256)
50 TM=TM-65536                          :'CONVERT TO  SIGNED DEC.
60 FORA=ØTO31
70 READB:POKETM+A,B
80 NEXT
90 POKE30845,205                        :'CALL for INTERUPT EXIT.
100 NEW
110 DATA33,150,Ø,1,70,Ø,58,251,104,254,121,192,205,92,52,58,251
120 DATA104,254,115,32,249,33,200,Ø,1,60,Ø,205,92,52,201
```

Australian Personal Computer Page 209

Jun 86 7(6)

ACTION: When a key is depressed, the keyboard scanning routine sets an INTERRUPT. The interrupt routine is vectored out of ROM to a interrupt exit.

787D/E/F Hex. (30845/6/7 Dec.) is the 3 byte interupt exit set by this routine. It is called by the interupt.

Lines 40 and. 90 set this to CALL (Tom+1). [CD LSB MSB]

78B1/2 Hex (30897/8 Dec.) contain the LSB + MSB of top of memory pointer.

68FB Hex. is the row address of the keyboard matrix. where.-

bit	5	4	3	2	1	Ø
corresponds to.	V	Z	C	shift	X	B

and the column lines go low when a key is depressed.

```
21 96 ØØ       LD HL,ØØ96H   ; load HL with 150D  - pitch for sound.
Ø1 46 ØØ       LD BC,ØØ46H   ; load BC with 70D   - duration for sound.
3A FB 68       LD A,(68FBH)  ; check row address of keyboard matrix
FE 79          CP 79H        ; compare with <shift X> or 121D. - set flags.
CØ             RET NZ        ; return if not <shift X> depressed. → exit NOT<shiftX>
CD 53 34       CALL 345CH    ; call sound routine.
3A FB 68  LOOP LD A,(68FBH)  ; check row address of keyboard matrix.    ←┐
FE 73          CP 73H        ; compare with <shift C> or 115D - set flags │ pause
20 F9          JR NZ,LOOP    ; if not <shift C> then jump back 7 bytes to loop. ┘ loop.
21 C8 ØØ       LD HL,ØØC8H.  ; load HL with 200D - higher pitch for exit sound.
Ø1 3C ØØ       LD BC,ØØ3CH   ; load BC with 60D - shorter duration for exit sound
CD 5C 34       CALL 345CH    ; call sound routine.
C9.            RET           ; return to interupt exit and ROM routines.
```

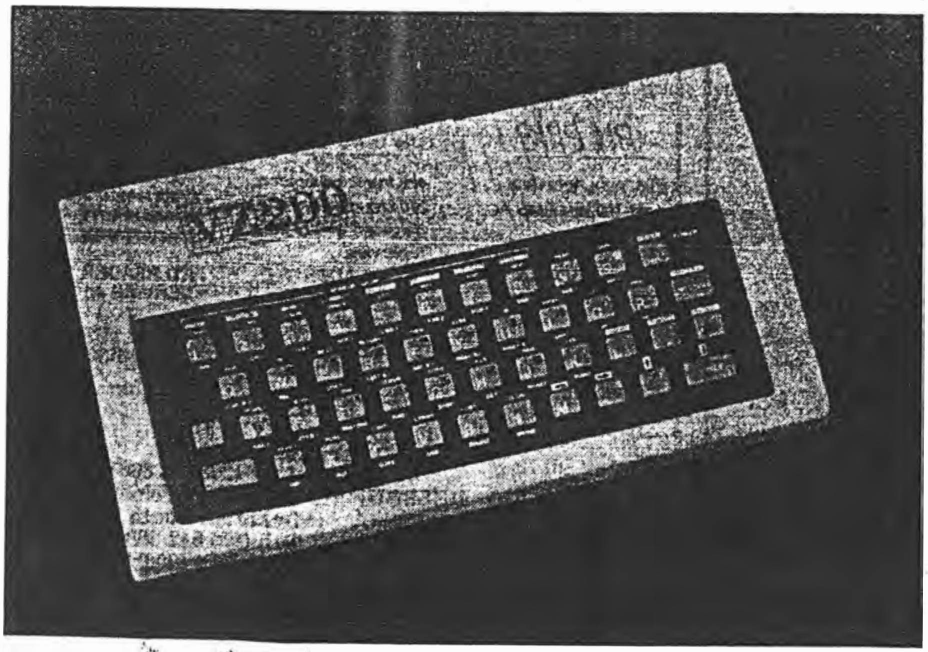
VZ SOFTWARE MODIFICATIONS

Fast Graphics on a VZ200/300? It can be done! Here is the good oil!

Chris Griffin

I BOUGHT A VZ200 soon after they were released as an 'upgrade' from my old 6800-based CHIP-8 machine. But it soon became obvious something was missing. It seemed I could get speed or high resolution, but not both. I wanted something that was fast *and* took full advantage of the 128 x 64 dot colour graphics; so, 'VZChip-8' was born.

VZChip-8 is a 'low memory' interpreter (about 1.5K all up), designed for VZ200s/300s with only 8K of memory. Figure 1 shows a memory map of a typical VZ computer running my Chip-8 'system'. Notice the presence of an editor. This is used to write your Chip-8 program and can also be used to write machine code programs. It is a separate program in its own right — a stand-alone component in the CHIP-8 system, so I have decided to discuss it first.



The Chip-8/machine code editor

This program is about 1K long and allows you to work entirely independently of BASIC. In fact, it allows you to talk directly to the central processor. Programs are written in hexadecimal — or base 16, and consist of a string of op-codes and arguments. If you don't understand you should get hold of a book on machine code programming for the Z80.

The basic requirements of an editor are that it be able to write, run and modify programs, print listings and save to tape or disk. I have included a few extras because I find them helpful, but otherwise, the editor consists only of these things.

Editor commands consist of a single letter. Its features revolve around the memory pointer. This is just like an arrow, pointing to a particular place in the VZ's memory. The editor uses the arrow to indicate where it is to store or retrieve the information it needs. For example, if you want to list a program beginning at memory location 8260, you first set the memory pointer to 8260, then instruct the editor to list. How do you do all of these things? Easy; using the following commands:

A prints out the ASCII value of the next character typed.

B returns to BASIC; this is used for saving to disk and loading from tape or disk.

D converts a hexadecimal number to its decimal equivalent.

G is used to run machine code program.

H help, prints out a message to remind you of something.

L lists memory to the screen, beginning at the memory pointer.

M sets the memory pointer to a particular place.

O outputs (saves) a program to tape; produces B programs which run automatically when you CLOAD them.

P puts data to memory, beginning at the memory pointer position. This command is used for writing and modifying programs.

S searches for a particular byte (or two), and points the memory pointer to the place where a match occurs.

T type; the same as list, except to the printer.

V vector; places the pointer at the memory location which is stored at the present pointer position.

X eXtension; allows for user defined commands, and others; an extension is used to activate Chip-8 programs.

Command extensions: X

Commands beginning with X are two characters long: the second character is a

1 of 3.

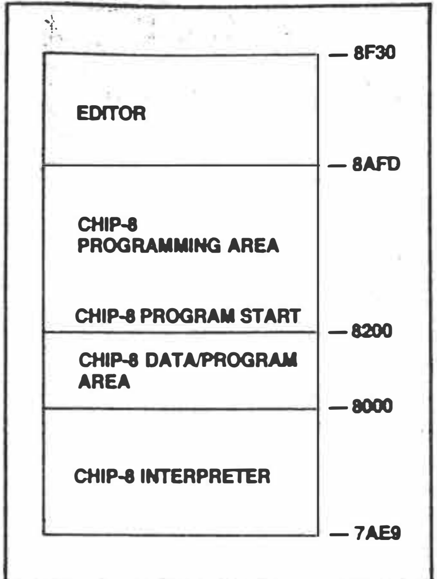
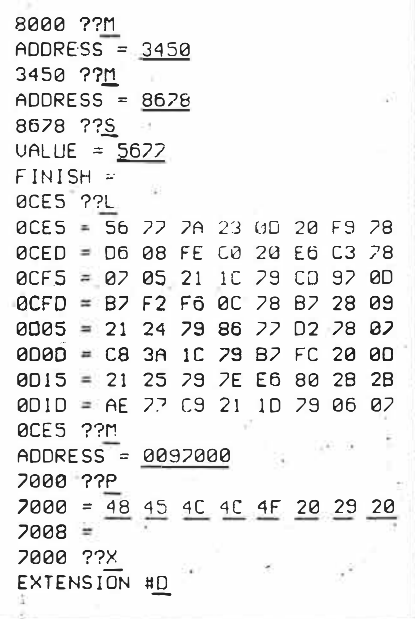


Figure 1. Memory map of an operational VZChip-8 programming environment.



```
8000 ??M
ADDRESS = 3450
3450 ??M
ADDRESS = 8678
8678 ??S
VALUE = 5677
FINISH =
0CE5 ??L
0CE5 = 56 77 7A 23 0D 20 F9 78
0CED = D6 08 FE C0 20 E6 C3 78
0CF5 = 07 05 21 1C 79 CD 97 0D
0CFD = B7 F2 F6 0C 78 B7 28 09
0D05 = 21 24 79 86 77 D2 78 07
0D0D = C8 3A 1C 79 B7 FC 20 0D
0D15 = 21 25 79 7E E6 80 28 2B
0D1D = AE 77 C9 21 1D 79 06 07
0CE5 ??M
ADDRESS = 0097000
7000 ??P
7000 = 48 45 4C 4C 4F 20 29 20
7008 =
7000 ??X
EXTENSION #D
```

Figure 2. Some of the editor commands in operation.

number (between 0 and F). Some X commands are already defined:

XO prints out a message beginning at the memory pointer position; (all messages use the byte 00 to signify the end).

XD directs all output to the video screen.

XE directs all output to the printer; for instance, Figure 2 was generated in this fashion.

XC We shall use the XC command to activate the Chip-8 interpreter but since it hasn't yet been installed XO just clears the screen. The process of adding your own X commands will become obvious when we discuss connection of the Chip-8 interpreter.

LISTING 1. USING THE EDITOR.

```
0 ' CHIP-8 INTERPRETOR PART I
1 ' EDITOR PROGRAM
2 ' DON'T BREAK THIS PROGRAM ONCEIT
3 ' BEGINS RUNNING...
4 '
5 CLS:PRINT@200,"PLEASE WAIT!!"
10 GOSUB50:IFA$="XX"THENGOSUB50:D=X:GOSU
B50:D=D*256+X:GOTO10
15 IFA$="ZZ"THENPOKE30863,112:POKE30862,
0:GOTO70
20 POKED,X:T=T+X:D=D+1:GOTO10
50 READA$:IFA$="XX"ORA$="ZZ"THENRETURN
51 X=ASC(LEFT$(A$,1))-48:B=ASC(RIGHT$(A$
,1))-48
60 X=(X+(X>9)*7)*16+(B+(B>9)*7)
65 RETURN
70 IFT=118309,PRINTUSR(1)
75 CLS:PRINT"AN ERROR HAS BEEN MADE, CHE
CK "
80 PRINT"THE LISTING CAREFULLY"
99 'MAIN PROGRAM LISTING
100 DATAXX,70,00,01,30,04,21,00,72,11,FD
,8A,ED,B0,C3,FD,8A
110 DATAXX,72,00,C3,E5,8B,7C,CD,05,8B,7D
,F5,1F,1F,1F,1F,CD,0E,8B
120 DATAF1,E6,0F,C6,30,FE,3A,38,02,C6,07
,18,18,E5,C5,CD
130 DATAF4,2E,B7,20,FA,CD,F4,2E,B7,28,FA
,0E,30,10,FE,0D
140 DATA20,FB,C1,E1,C9,E5,C5,CD,E4,8E,36
,20,CD,2A,03,2A
150 DATA20,78,36,AF,C1,E1,C9,E5,C5,F5,CD
,50,34,F1,18,E7
160 DATAE5,C5,CD,1A,8B,47,FE,0D,28,0B,FE
,30,38,F4,FE,3A
170 DATA30,10,E6,0F,21,3E,80,F5,78,CD,44
,88,F1,FE,80,C1
180 CATAE1,C9,FE,41,38,DC,FE,47,30,D8,D6
,07,18,E4,1A,B7
190 DATAC8,CD,32,8B,13,18,F7,CD,00,8B,11
,A0,8B,CD,7B,8B
200 DATA06,08,3E,20,CD,32,8B,7E,23,CD,05
,8B,10,F4,3E,0D
210 DATAC3,32,8B,20,3D,00,CD,7B,88,3E,20
,CD,32,8B,21,00
220 DATA00,06,00,CD,4D,8B,C8,29,29,29,29
,85,6F,04,18,F3
230 DATA21,E9,7A,22,F9,78,21,07,8F,22,8E
,78,2D,CD,F6,8E
240 DATAAF,32,9C,78,3E,11,32,3B,78,32,00
,68,3E,03,32,39
250 DATA78,21,00,80,22,10,78,C9,F3,31,FF
,BF,CD,BD,8B,11
260 DATADD,8D,CD,7B,8B,2A,10,78,CD,00,8B
,11,2D,8C,CD,7B
270 DATA8B,CD,1A,8B,FE,41,38,F9,FE,5B,30
,F5,47,CD,44,8B
280 DATA3E,0D,CD,32,8B,21,31,8C,7E,FE,FF
,28,D8,23,B8,28
290 DATA04,23,23,18,F3,5E,23,56,D5,E1,CD
,2C,8C,18,C6,E9
300 DATA20,3F,3F,00,4C,59,8C,4D,65,8C,47
,6F,8C,53,79,8C
310 DATA50,8D,8C,56,27,8D,41,32,8D,44,53
,8D,4F,64,8D,48
320 DATA46,8E,42,4C,8E,54,57,8E,58,98,8E
,FF,2A,10,78,0E
330 DATA08,CD,84,8B,0D,20,FA,C9,11,04,8E
,CD,A3,8B,22,10
340 DATA78,C9,11,0E,8E,CD,A3,8B,78,B7,C8
,E9,11,16,8E,CD
350 DATAA3,8B,78,87,C8,FE,03,F5,30,01,65
,E5,11,1E,8E,CD
360 DATAA3,8B,ED,5B,10,78,13,78,B7,20,03
,2A,10,78,C1,1A
370 DATA13,B8,20,0F,F1,38,06,F5,1A,B9,20
,07,F1,1B,ED,53
380 DATA10,78,C9,DF,20,E9,11,27,8E,C3,7B
,8B,00,00,00,00
390 DATA2A,10,78,06,00,CD,00,8B,11,1D,8D
,CD,7B,8B,3E,08
400 DATAF5,3E,20,CD,32,8B,CB,78,20,20,CD
,1A,8B,FE,22,28
410 DATA1D,00,00,CD,18,8D,28,14,87,87,87
,87,F5,CD,4D,8B
420 DATAD1,28,09,82,77,23,F1,30,20,D6,18
,27,F1,C9,CB,F8
430 DATA3E,41,CD,FF,8E,18,CF,CD,1A,8B,FE
,22,20,06,CB,B8
440 DATA3E,AF,18,EE,F5,CD,44,8B,F1,18,D9
,E5,C5,C3,52,8B
450 DATA20,3D,00,3E,0D,CD,32,8B,18,98,2A
,10,78,7E,23,66
460 DATA6F,22,10,78,C9,11,32,8E,CD,7B,8B
,CD,1A,8B,F5,CD
470 DATA44,8B,3E,0D,CD,32,88,11,16,8E,CD
,7B,8B,F1,CD,05
480 DATA8B,3E,0D,C3,32,88,11,40,8E,CD,A3
,8B,11,16,8E,CD
490 DATA7B,8B,CD,AF,0F,18,EA,11,39,8E,CD
,7B,8B,21,9D,7A
500 DATA06,10,CD,1A,88,F5,CD,44,8B,F1,FE
,01,C8,FE,0D,28
510 DATA04,77,23,10,ED,36,00,3E,11,90,32
,D6,7A,11,0E,8E
520 DATACD,A3,8B,E5,11,1E,8E,CD,A3,8B,F3
,0E,F1,CD,5B,35
530 DATAD1,CD,A3,8D,F3,C9,D8,01,9A,01,0B
,79,B0,20,FB,DD
540 DATA21,23,78,7B,CD,11,35,DD,77,00,AF
,DD,77,01,7A,CD
550 DATAD7,8D,7D,CD,D7,8D,7C,CD,D7,8D,CD
,E8,3A,D8,1A,13
560 DATACD,D7,8D,DF,20,F4,E5,C3,FA,34,CD
,11,35,C3,8E,38
570 DATA1F,56,5A,2D,32,30,30,20,48,45,58
,20,45,44,49,54
580 DATA4F,52,0D,56,45,52,20,32,2E,31,0D
,28,43,29,20,43
590 DATA47,27,38,35,0D,0D,00,41,44,44,52
,45,53,53,20,3D
600 DATA00,53,54,41,52,54,20,3D,00,56,41
,4C,55,45,20,3D
610 DATA00,46,49,4E,49,53,48,20,3D,00,4E
,4F,54,20,46,4F
620 DATA55,4E,44,0D,00,43,48,41,52,20,3D
,00,4E,41,4D,45
630 DATA20,3D,00,48,45,58,20,3D,00,11,64
,8E,C3,7B,8B,FB
640 DATACD,7A,1E,ED,7B,E8,78,C3,19,1A,21
,9C,78,36,01,E5
650 DATACD,59,8C,E1,36,00,C9,43,4F,4D,4D
,41,4E,44,53,20
660 DATA41,52,45,0D,41,2C,42,2C,44,2C,47
,2C,48,2C,4C,2C
680 DATA4D,2C,4F,2C,50,2C,53,2C,54,2C,56
,2C,58,0D,00,45
690 DATA58,54,45,4E,53,49,4F,4E,20,23,00
,11,8C,8E,CD,7B
700 DATA8B,CD,4D,8B,C8,87,C6,AF,6F,26,8E
,F1,CD,4E,8D,C3
710 DATA22,8C,DA,8E,E4,8B,E4,8B,E4,8B,E4
,8B,E4,8B,E4,8B
720 DATAE4,8B,E4,8B,E4,8B,E4,8B,E4,8B,C9
,01,D5,8E,CF,8E
730 DATAE4,8B,3E,01,32,9C,78,C9,AF,32,9C
,78,C9,ED,5B,10
740 DATA78,CD,7B,8B,C3,4E,8D,2A,20,78,47
,3A,9C,78,B7,78
750 DATAC8,FE,80,D8,C6,20,E6,7F,C9,21,FC
,8A,22,B1,78,2D
760 DATA18,12,32,40,8B,3E,01,C3,44,8B,F3
,31,FF,8F,CD,CD
```

Errata to Listing

NOTE: We have had complaints from readers who could not get the editor listed last month running. Printed below are corrections to lines 70 and 380, and two new lines 770, 780 to be added. As well as this, we understand that in some issues of the magazine, the figure 32 between 90 and D6 in line 510 was printed so indistinctly as to look like 37. So if you have any problems after amending the listing, check line 510.

CORRECTIONS TO THE 'EDITOR' LISTING.
THE FOLLOWING ARE THE CORRECTED LINES.

```
70 IFT = 118550, PRINTUSR (1)

380 DATA10,78,C9,DF,20,E9,F1,11,
27,8E,C3,7B,8B,00,00,00

770 DATA8B,C3,Ec,8B,2D,22,A0,78,
C9,00,00,00
780DATAXX
```

NB. THE LAST TWO LINES NEED TO BE ADDED TO THE PROGRAM. ●

ETI Aug 86
p 89.
3 of 3

Using the editor

Key in the listing given (Listing 1), save a copy of it, then run the program. You will have to wait a while, until everything is set up. If an error results, check the listing carefully. An introductory message will be printed when the editor is installed. Save a copy in this form to tape or disk. To do this tape users should type: OVZEDITOR (cr) 8AFD (cr) 8F30 (cr), where (cr) means the RETURN key. The last (cr) is not typed until the tape recorder is on and in record mode.

Alternatively, type BBSAVE "VZEDITOR",8AFD,8F30('cr'). Both Bs are essential. The first is needed to exit the editor. This step eliminates the delay from occurring every time the editor program is run. It saves the machine code part, produced by Listing 1, to the relevant medium.

Commands

Now, try out some commands: particularly M,L,H and T (if you have a printer). It is a good idea not to use the G or K commands just yet.

You will find that many commands prompt for ADDRESSes, START locations, STOP locations, etc. The answer accepted by the computer consists of the *last* four digits of whatever is typed in. If you meant to type 88D8, and instead, entered 8BE, just type in the right response and the problem is fixed, so that 8BE88D8 is interpreted as 88D8. This is important because the editor is *not* equipped with a backspace facility.

The P command, as I said before, allows you to put data in memory. To test it out, set the memory pointer to 7080 (use M7080 (cr)) and type P. Now, type in the following data: 48454C4C4F (cr). Notice that the word HELLO appears on the screen as you type. You have stored the ASCII values for HELLO at location 7080-7084, which is in screen memory.

How did I know to use 4845 . . .? I looked it up; but that's a laborious task if you want to enter lots of words into memory. Instead, you can use an easier form: type M70C0 (cr) P", the " (shift 2) allows for character data entry — the computer does all of the conversions for you! (Notice that while in this mode, the normally blue cursor turns into an 'A'.) After typing in the required word, pressing another " returns the cursor to blue again, so you can enter hexadecimal data as usual.

S is used to search for one or two bytes, depending on what you type in, from the memory pointer to the end position (which you also type in). If a two-byte search is required, make sure the search string is more than two digits long. For example, to search for 6A00 in the region of memory 8200 to 8500, type M8200 (cr) S6A00 (cr) 8500 (cr). The message NOT FOUND means that 6A00 could not be found anywhere between locations 8200 and 8500. ●

IMPORTANT EDITOR MEMORY LOCATIONS

The editor has a small collection of useful subroutines. These can be used when prototyping a Chip-8 program or when writing machine code programs. Care should be taken to ensure that calls to these subroutines are not present in the final program, unless the editor is to be included in the final program.

Location	Description	Registers altered
8AFD	Jump location, COLD START.	HL,BC,DE,AF
8B00	Show HL register pair as a hexadecimal value.	AF
8B05	Show A register as a hexadecimal value.	AF
8B1A	Wait for a key press, A contains the ASCII value of the key that was pressed.	AF
8B32	Show the character stored in A.	none
8B44	Show character in A, and beep.	none
8B4D	Get a hexadecimal key (0-F, or (cr)) and put the value in A, A equals 80 if (cr) is pressed.	AF
8B7B	Show a string using DE as the pointer, up to the character stored as 00.	DE,AF
8BA3	Shows a message off DE, and gets a two-byte number from the keyboard; the number is stored in HL, while B contains the number of keys pressed.	HL,B,DE,AF

The following locations contain prompt messages used by the editor. Each message consists of a string of ASCII characters ending with the byte 00. These messages can be changed to suit your own personal requirements.

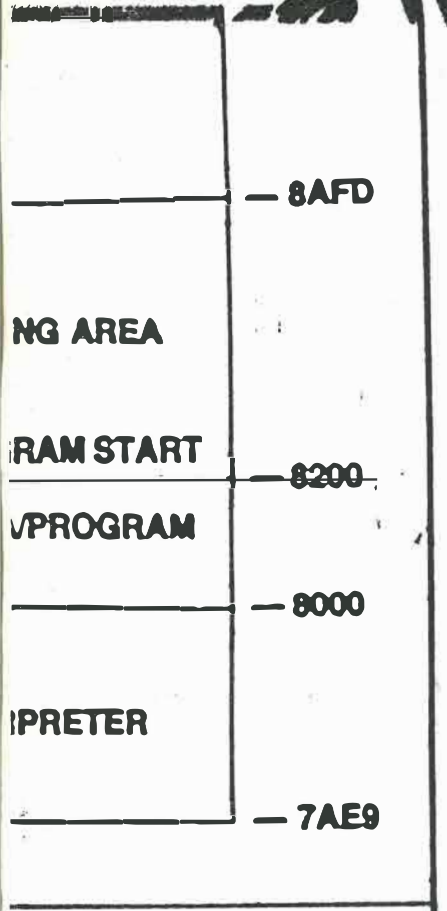
Location	Length	Description
8DDD	38	Introductory message; this is the heading displayed when the editor first begins.
8E64	39	Help message; the 39 characters here are reserved for a simple memo which is called up by pressing H.
8C2D	3	Prompt string, normally consists of a space and two question marks.

Example: to change the help message, type:

M8E64 (cr) P"this is the new message (cr) "00 (cr)

Make sure that whatever you type as the message is less than the maximum size of 39 characters.

Next month: the CHIP-8 interpreter.



ap of an operational
ing environment.

```
0
8
7A 23 0D 20 F9 78
FE C0 20 E6 C3 78
21 1C 79 CD 97 0D
F6 0C 78 B7 28 09
79 86 77 D2 78 07
1C 79 B7 FC 20 0D
79 7E E6 80 2B 2B
C9 21 1D 79 06 07
17000
4C 4C 4F 20 29 20
```

he editor commands in

0 and F). Some X com- defined:
essage beginning at the r position; (all messages to signify the end).
ut to the video screen.
ut to the printer; for in- 2 was generated in this

he XC command to acti- interpreter but since it installed XO, just clears process of adding your ds will become obvious connection of the Chip-

LISTING 1, USING THE EDITOR.

```
0 ' CHIP-8 INTERPRETOR PART I
1 ' EDITOR PROGRAM
2 ' DON'T BREAK THIS PROGRAM ONCEIT
3 ' BEGINS RUNNING...
4 '
5 CLS:PRINT@200,"PLEASE WAIT!!"
10 GOSUB50:IFA$="XX"THENGOSUB50:D=X:GOSU
B50:D=D*256+X:GOTO10
15 IFA$="ZZ"THENPOKE30863,112:POKE30862,
0:GOTO70
20 POKED,X:T=T+X:D=D+1:GOTO10
50 READA$:IFA$="XX"ORA$="ZZ"THENRETURN
51 X=ASC(LEFT$(A$,1))-48:B=ASC(RIGHT$(A$
,1))-48
60 X=(X+(X>9)*7)*16+(B+(B>9)*7)
65 RETURN
70 IFT=118~~389~~550,PRINTUSR(1)
75 CLS:PRINT"AN ERROR HAS BEEN MADE, CHE
CK "
80 PRINT"THE LISTING CAREFULLY"
99 'MAIN PROGRAM LISTING
100 DATAXX,70,00,01,30,04,21,00,72,11,FD
,8A,ED,B0,C3,FD,8A
110 DATAXX,72,00,C3,E5,8B,7C,CD,05,8B,7D
,F5,1F,1F,1F,1F,CD,0E,8B
120 DATAF1,E6,0F,C6,30,FE,3A,38,02,C6,07
,18,18,E5,C5,CD
130 DATAF4,2E,B7,20,FA,CD,F4,2E,B7,28,FA
,0E,30,10,FE,0D
140 DATA20,FB,C1,E1,C9,E5,C5,CD,E4,8E,36
,20,CD,2A,03,2A
150 DATA20,78,36,AF,C1,E1,C9,E5,C5,F5,CD
,50,34,F1,18,E7
160 DATAE5,C5,CD,1A,8B,47,FE,0D,28,0B,FE
,30,38,F4,FE,3A
170 DATA30,10,E6,0F,21,3E,80,F5,78,CD,44
,8B,F1,FE,80,C1
180 DATAE1,C9,FE,41,38,DC,FE,47,30,D8,D6
,07,18,E4,1A,B7
190 DATAC8,CD,32,8B,13,18,F7,CD,00,8B,11
,A0,8B,CD,7B,8B
200 DATA06,08,3E,20,CD,32,8B,7E,23,CD,05
,8B,10,F4,3E,0D
210 DATAC3,32,8B,20,3D,00,CD,7B,8B,3E,20
,CD,32,8B,21,00
220 DATA00,06,00,CD,4D,8B,C8,29,29,29,29
,85,6F,04,18,F3
230 DATA21,E9,7A,22,F9,78,21,07,8F,22,8E
,78,2D,CD,F6,8E
240 DATAAF,32,9C,78,3E,11,32,3B,78,32,00
,68,3E,03,32,39
250 DATA78,21,00,80,22,10,78,C9,F3,31,FF
,8F,CD,BD,8B,11
260 DATADD,8D,CD,7B,8B,2A,10,78,CD,00,8B
,11,2D,8C,CD,7B
270 DATA8B,CD,1A,8B,FE,41,38,F9,FE,58,30
,F5,47,CD,44,8B
280 DATA3E,0D,CD,32,8B,21,31,8C,7E,FE,FF
,28,D8,23,B8,28
290 DATA04,23,23,18,F3,5E,23,56,D5,E1,CD
,2C,8C,18,C6,E9
300 DATA20,3F,3F,00,4C,59,8C,4D,65,8C,47
,8F,8C,53,79,8C
310 DATA50,BD,8C,56,27,8D,41,32,8D,44,53
,8D,4F,64,8D,48
320 DATA46,8E,42,4C,8E,54,57,8E,58,98,8E
,FF,2A,10,78,0E
330 DATA08,CD,84,8B,0D,20,FA,C9,11,04,8E
,CD,A3,8B,22,10
340 DATA78,C9,11,0E,8E,CD,A3,8B,78,B7,C8
,E9,11,16,8E,CD
350 DATAA3,8B,78,B7,C8,FE,03,F5,30,01,65
,E5,11,1E,8E,CD
360 DATAA3,8B,ED,5B,10,78,13,78,B7,20,03
,2A,10,78,C1,1A
370 DATA13,B8,20,0F,F1,38,06,F5,1A,B9,20
,07,F1,1B,ED,53
                                       F1
380 DATA10,78,C9,DF,20,E9,11,27,8E,C3,7B
                           ^
,8E,00,00,00
390 DATA2A,10,78,06,00,CD,00,8B,11,1D,8D
,CD,7B,8B,3E,08
400 DATAF5,3E,20,CD,32,8B,CB,78,20,2D,CD
,1A,8B,FE,22,28
410 DATA1D,00,00,CD,18,8D,28,14,87,87,87
,87,F5,CD,4D,8B
420 DATAD1,28,09,82,77,23,F1,3D,20,D6,18
,27,F1,C9,CB,F8
430 DATA3E,41,CD,FF,8E,18,CF,CD,1A,8B,FE
,22,20,06,CB,B8
440 DATA3E,AF,18,EE,F5,CD,44,8B,F1,18,D9
,E5,C5,C3,52,8B
450 DATA20,3D,00,3E,0D,CD,32,8B,18,9B,2A
,10,78,7E,23,66
460 DATA6F,22,10,78,C9,11,32,8E,CD,7B,8B
,CD,1A,8B,F5,CD
470 DATA44,8B,3E,0D,CD,32,8B,11,16,8E,CD
,7B,8B,F1,CD,05
480 DATA8B,3E,0D,C3,32,8B,11,40,8E,CD,A3
,8B,11,16,8E,CD
490 DATA7B,8B,CD,AF,0F,18,EA,11,39,8E,CD
,7B,8B,21,9D,7A
500 DATA06,10,CD,1A,8B,F5,CD,44,8B,F1,FE
,01,C8,FE,0D,28
510 DATA04,77,23,10,ED,36,00,3E,11,90,32
,D6,7A,11,0E,8E
520 DATACD,A3,8B,E5,11,1E,8E,CD,A3,8B,F3
,0E,F1,CD,5B,35
530 DATAD1,CD,A3,8D,F3,C9,D8,01,9A,01,0B
,79,B0,20,FB,DD
                                        AF
540 DATA21,23,78,7B,CD,11,35,DD,77,00,AF
,DD,77,01,7A,CD
550 DATAD7,8D,7D,CD,D7,8D,7C,CD,D7,8D,CD
,E8,3A,D8,1A,13
560 DATACD,D7,8D,DF,20,F4,E5,C3,FA,34,CD
,11,35,C3,8E,38      end of prog.
570 DATA1F,56,5A,2D,32,30,30,20,4B,45,58
,20,45,44,49,54
580 DATA4F,52,0D,56,45,52,20,32,2E,31,0D
,28,43,29,20,43
590 DATA47,27,38,35,0D,0D,00,41,44,44,52
,45,53,53,20,3D
600 DATA00,53,54,41,52,54,20,3D,00,56,41
,4C,55,45,20,3D
610 DATA00,46,49,4E,49,53,48,20,3D,00,4E
,4F,54,20,46,4F
620 DATA55,4E,44,0D,00,43,48,41,52,20,3D
,00,4E,41,4D,45
630 DATA20,3D,00,48,45,58,20,3D,00,11,64
,8E,C3,7B,8B,FB
640 DATACD,7A,1E,ED,7B,E8,78,C3,19;1A,21
,9C,78,36,01,E5
650 DATACD,59,8C,E1,36,00,C9,43,4F,4D,4D      8EC
,41,4E,44,53,20
660 DATA41,52,45,0D,41,2C,42,2C,44,2C,47
,2C,48,2C,4C,2C                                No
680 DATA4D,2C,4F,2C,50,2C,53,2C,54,2C,58       8E6
,2C,58,0D,00,45
690 DATA58,54,45,4E,53,49,4F,4E,20,23,00      8C9
,11,8C,8E,CD,7B
700 DATA8B,CD,4D,8B,C8,87,C6,AF,6F,26,8E
,F1,CD,4E,8D,C3
710 DATA22,8C,DA,8E,E4,8B,E4,8B,E4,8B,E4
,8B,E4,8B,E4,8B
720 DATAE4,8B,E4,8B,E4,8B,E4,8B,E4,8B,C9
,01,D5,8E,CF,8E
730 DATAE4,8B,3E,01,32,9C,78,C9,AF,32,9C
,78,C9,ED,5B,10
740 DATA78,CD,7B,8B,C3,4E,8D,2A,20,7B,47
,3A,9C,78,B7,78
750 DATAC8,FE,80,D8,C6,20,E6,7F,C9,21,FC
,8A,22,B1,78,2D
760 DATA18,12,32,40,8B,3E,01,C3,44,8B,F3
,31,FF,8F,CD,CD.
770 DATA 8B,C3,EC,8B,2D,22,A0,78,
         C9,00,00,00
780 DATA ZZ  ETI Aug 86 p 87
```

see Errata ETI Oct 86 p 33

A CHIP-8 INTERPRETER — for VZ200/300

Chris Griffin

How's it going? Did you get the editor from the last article in August '86, typed in, up, and running? If you had any trouble refer to the note at the end of the article. In this article I use the editor to set up the Chip-8 interpreter, to write and run Chip-8 programs. I will also mention details of this particular dialect and show a few simple programs to get you started.

THE CHIP-8 interpreter (Listing 1) is a machine language program which executes instructions beginning at location 8200 (this is in hex — remember!). The interpreter has an 'address space of 4K, meaning that it can only access 4096 bytes of memory. Therefore only three hex digits are required to specify an address. 8200 is referred to as 200 by the Chip-8 interpreter, 54A refers to 854A, etc. So, if from time to time, I drop the leading 8, don't be too bothered about it!

Each Chip-8 instruction consists of two bytes of hexadecimal data — a total of four digits. Between 200 and AFC, the locations in which a program may be stored, there is thus room for about 1150 instructions. You can also use locations (8)000 to (8)1FF to store parts of the program, but never forget that execution is from location 200, so you'll have to use this section of memory for subroutines or shape data.

Chip-8 is a 'what you write is what you get' sort of language in that there is no way to break out of a program that is running, unless you have allowed for this possibility. This is one aspect that could take a little getting used to, but don't worry, you will! The Chip-8 interpreter has in this regard a trade off. A little speed is gained in the sacrifice; and for me, the speed is worth it!

The language of Chip-8 supports only 16 variables, an index register, and a stack pointer (which is rarely used in programs — it is more useful to the interpreter itself!).

The variables, labelled by a 'V', followed by a number (0,1,2...D,E or F), are each one byte long. They can only be used to store numbers in the range 0 to 255, so all operations involving variables are limited in this way. If any extra space is required to store the answer to a calculation, VF is used for the extra piece. (It is called the carry, and is only relevant to a few arithmetic commands. Larger number manipulation is available to a limited degree, using the index register called 'I'. This is a 12-bit number (3 hex digits) and is used to point to memory locations in much the same way that the editor program has a memory pointer. When you store 6B0 in the index register, it points to location 86B0, as might be expected! The index register is an important part of the system as it is used extensively in graphics manipulation; it also allows more than 16 variables to be used by a single program, if desired.

OK, now let's get things up and running!

Getting started

Load your copy of the editor program (ETI August 86 issue), and run it. Then, type in Listing 1 beginning at location 7AE9 (type M7AE9 (cr) P then the data shown in the listing). Check the things typed, to make sure they are correct and type in the following:

(i) M9BDF (cr) P0082 (cr)

This sets the memory pointer to 8200 whenever the editor is run.

(ii) M8EC7 (cr) PE97A (cr)

This connects the Chip-8 interpreter to the editor, allowing it to be activated by pressing XC. 8EC7 is the location which contains the start address for the routine which we want activated by XC — and we store 7AE9, the interpreter start address, here. By the way, locations 8EBF to

1 of 5.

8ECD contain the start addresses for all of the X commands (XC through XF), so it's easy to add your own!

(iii) M8200 (cr) PF000 (cr)
A very short Chip-8 program, just to test things out.

Now, save everything. Use OVZCHIP8 (cr) 7AE9 (cr) 8F30 (cr) if you have a tape system, or use BBSAVE "VZCHIPS", 7AE9, 8F30 (cr) if disks are your forte (after saving to disk, you can restart the editor with ?USR(O)).

Let's run the Chip-8 program entered in (iii) above, by pressing XC. The screen should have flashed, and the editor restarted. If it has, so far so good. If not, check that the interpreter you typed in is the same as mine! Tape users will probably have to start all over again!! (This is because B: programs run automatically from tape, but not from disk.) When everything works thus far, read on...

Chip-8 graphics

Graphics takes place on the VZ's mode 1 screen. The individual points are labelled with two coordinates in exactly the same manner as BASIC (except, everything is in hex). Chip-8 allows you to display points (like BASIC), entire shapes (of up to 8 x 16 dots) and line drawings in 256 sizes (although there are some restrictions!) in any combination of colours you care to imagine. (Of course, only four colours can be used at once in this mode — there is little that can be done about this.) An object can be positioned anywhere on the screen, even overlapping another object. Overlapping objects are stored on the screen in exclusive-or form. Table 1 shows the consequences of this in colour mode 0 (COLOR, 0), which is read as: 'if a red object is placed on a blue area of the screen, the overlap is displayed in yellow' etc. Funny idea? Not really! These conditions allow you to remove objects by simply re-displaying them. If we number the colours 0 for green, 1 for yellow, 2 for blue, 3 for red, and change to COLOR, 1 mode the same sort of ideas apply to buff, cyan, magenta and orange.

A collision occurs if the following pairs of colours overlap: 1&1, 2&2, 3&3, 3&1, 3&2. Collisions are registered through an object called 'HIT'. HIT equals 1 means that there has been a collision, HIT equals 0, otherwise. After a graphics command has been executed, HIT is stored in VF (variable F), to allow you to check for collision with Chip-8 instructions.

Shape drawing

A 'SHAPE' is eight dots wide, and between 1 and 16 dots long, and is considered as residing in a grid (see Figure 1 for an example 8 x 9 shape in its grid). Each row of the shape is represented by two bytes of data, that is, four dots to each byte. The colour of each dot can be independently defined using the *number* of the colour that is required.

For the first row of the shape down, we have two green dots (which are in essence *invisible*) five blue dots, and one green dot. The colour codes are 0,0,2,2,2,2,2,0. Group this information into clusters of two digits: 00 22 22 20, then for each cluster, multiply the first digit by 4 and add the second to it, giving 0 A A 8 in our example. The two bytes used to describe this row are thus 0A and A8. Every other row is complete in exactly the same manner and the data stored in a segment of memory.

TABLE 1. COLOUR OVERLAP

Overlapping colours	Green	Yellow	Blue	Red
Green	Green	Yellow	Blue	Red
Yellow	Yellow	Green	Red	Blue
Blue	Blue	Red	Green	Yellow
Red	Red	Blue	Yellow	Green

		B	B	B	B	B	
		B	Y	B	Y	B	
		B	B	B	B	B	
				R			
			R	R	R		
	R	R		R		R	R
				R			
			R		R		
		R				R	

Figure 1. Example of a nine row shape (a robot figure). Each square is filled with the colour that is desired. Those with no colour are green by default, as this behaves invisibly.
Y — yellow colour value is 1
B — blue colour value is 2
R — red colour value is 3
The last row, for example, is 00300030, which is 0C0C in hex.

To put this shape up onto the screen, we set the index register I to point to the first byte of the shape data, and use a SHOW command. From the table of Chip-8 commands (Table 2), it is obvious that the SHOW command is Dxyn, but what does that mean? An example should make this clearer: D456 will show a shape, six rows long, with the top left hand corner at (V4,V5). If we want to display the example shape at (V3,V4), then use the command D349 — the 9 means that our shape is nine rows long.

Let's write up a real Chip-8 program now.

Writing Chip-8 programs

To write a Chip-8 program, simply put the instructions, one after another, in memory from location 200 onwards. Consider the short program that we typed in earlier; pressing XC did nothing much, so what was the Chip-8 program? Well, it consisted of the single instruction F000, which from Table 2, 'jumps back to the editor, or restarts the program if the editor is not found' — in other words: END! So, that's why nothing much happened! For a real program, see Listing 2a. Type this one in (from 8200), and run it XC. You should get the picture we designed earlier in the top left hand corner of the screen. Press a key, and the program ends. Do you understand what went on? The comments given may be of some help! Notice that we didn't need to switch on mode 1 graphics — it's automatic! (Chip-8 operates entirely in this mode.) For more examples, we need more concepts so read on.

Colour registers

The colour register is another VZ/Chip-8 object — like HIT. This, however, is used to store colour data for some commands (Fx29, 8xyD and 8xyE). The register takes on the following values for colours: 00 — invisible or colour 0, 55 — colour 1, AA — colour 2, FF — colour 3. All other values give combinations of these, and are best experimented with! To load the colour register with 55, we could use the following sequence of code. 6F55 FFCC, which says, load VF with 55, then load the colour register with VF. Once the colour is set, we can use 8xyD to plot a point, or Fx29 to draw a number, in the colour that we have defined. Type in and run Listing 2b for an idea of colour register graphics operation.

Joysticks and keyboard

The command ExB4, reads both joysticks at once, and assigns Vx to one of the following values, depending on the joystick position: 00 — nothing, 2E — up, 20 — down, 4D — left, 2C — right, 0D — fire. These codes were chosen as they correspond to the cursor control keys on the VZ keyboard. Using ExB3 instead of ExB4 reads the keyboard and allows the result of this command to be treated in an identical manner to the ExB4 command it replaces. The break key returns a value of 01 if it is pressed, so it too can be easily tested for. ▶

2 of 5.

Printing out numbers

See Listing 2c for an example of number printing. The Chip-8 interpreter has shape data for the numbers 0,1,2,3...D,E,F automatically built in. All that is required is to retrieve them. The statement Fx29 does just that: retrieves the shape data for the last digit of Vx. If V8 is 7A, F829 retrieves data for the number A, and sets the index register to point to the place where the retrieve data is stored, so that the next display command will show the correct thing. (The data is stored in system memory and will never get in the way of one of your Chip-8 programs.) That's OK for single digit numbers. But what about bigger ones, like 8A, EB etc, or even decimal numbers (for game scores, for instance)?

The process of printing decimal numbers is easy, but fairly long, if you write in Chip-8. See Listing 2d, which repeatedly counts from 0 to 99, for an example. Some important commands are the following.

(i) Fx33, converts Vx to a three digit decimal number, and stores each digit in a different memory location, pointed to by the index register. The hundreds get stored at I, tens at I plus 1, and units at I plus 2, so that if we could load these values into variables, each digit could be displayed in the usual way.

(ii) F265 loads the memory from I, into variables V0, V1 and V2. V0 contains the hundreds, V1 the tens, V2 the units. We can now easily display each digit.

Notice also that the printing process is put in a subroutine at location 228, this saves me repeating the whole process in order to remove the numbers. (Recall: to remove things in Chip-8, simply re-display them.)

TABLE 2 — VZ/CHIP-8 COMMAND SUMMARY

0000 No operation. Does nothing.
00A0 Store I on the subroutine stack.
00A8 Take I off the subroutine stack.
00AE Load I with the subroutine stack pointer.
00C0 Set colour to set 0 (green background).
00C1 Set colour to set 1 (buff background).
00E0 Clear the screen.
00EE Return from a subroutine.
0nnn For nnn larger than 0FF, calls a machine code routine at location 8nnn. Allows user machine code subroutines.
1nnn Go to 8nnn.
2nnn Go sub 8nnn.
3xyy Skip the next instruction if Vx equals yy.
4xyy Skip the next instruction if Vx does not equal yy.
5xy0 Skip the next instruction if Vx equals Vy.
6xyy Load Vx with yy.
7xyy Add yy to Vx.
8xy0 Load Vx with Vy.
8xy1 Load Vx with Vx OR Vy.
8xy2 Load Vx with Vx AND Vy.
8xy3 Load Vx with Vx XOR Vy (exclusive or).
8xy4 Load Vx with Vx plus Vy (the carry is stored in VF).
8xy5 Load Vx with Vx minus Vy (the carry is stored in VF).
8xy6 Load Vx with Vx multiplied by Vy (carry is in VF).
8xyD Plot a point at coordinates (Vx,Vy) with colour as in the colour register.
8xyE Draw a shape with data pointed to by I, of size VF, beginning at the point (Vx,Vy).
9xy0 Skip next instruction if Vx does not equal Vy.
AnnnLoad I with 8nnn.
BnnnGo to 8nnn plus V0.
Cxyy Load Vx with a random number ANDed with yy.
Dxyn Show a pattern with data pointed to by I, consisting of n rows with the top left hand corner at (Vx,Vy).
Ex9E Skip the next instruction if Vx equals the key that is down.
ExA1 Skip the next instruction if Vx does not equal the key that is down.
ExB3 Load Vx with the key that is currently down.
Ex84 Load Vx with the present joystick position.
F000 Jump back to the editor or restart the program if no editor is present.
Fx02 Set the sound pitch to Vx.
Px0A Wait for a key to be pressed and load Vx with that key.
Fx18 Beep for Vx cycles.
Fx19 Produce white noise (hiss) for Vx cycles.
Fx1E Add Vx to I.
Fx29 Produce a digit pattern for the last digit of Vx and point I at this pattern (colour is given by colour register).
Fx33 Convert Vx to a decimal number and store each digit in a different byte (100s, 10s, 1s in 3 bytes from 1).
Fx55 Store V0 through Vx to memory pointed to by I (on completion, I is I plus x plus 1).
Fx65 Load V0 through Vx from memory pointed to by I (on completion, I is I plus x plus 1). Opposite of Fx55.
FxCC Load the colour register with Vx.
Any other commands should be avoided — their functions are not defined, but in general, they do *not* represent no operation.

TABLE 3. PITCH/DURATION VALUES FOR SOUND COMMANDS

Pitch	Duration 2	Duration 1	Duration ½	Duration ¼
C 79	79	3C	1E	0F
Db 72	80	40	20	10
D 6C	88	44	22	11
Eb 66	90	48	24	12
E 60	98	4C	26	13
F 5B	A0	50	28	14
Gb 55	AB	55	2B	15
G 50	B5	5B	2D	17
Ab 4C	C0	60	30	18
A 48	CB	66	33	19
Bb 44	D7	6C	36	1B
B 40	E4	72	39	1C
C 3B	F2	79	3B	1E

(Other octaves can be approximated by halving and doubling the pitch and duration values.)

PLOT	LEFT	RIGHT	DOWN	UP	FOUR	TWO	ONE

Figure 2. 8xyE allocation of bits. A '1' in the bit position activates the associated words, eg, PLOT UP and LEFT 5 is 11001101.

How to draw large shapes

8xyE is a command designed to draw large shapes on the graphics screen. Often, the object to be drawn is simple in structure, yet too big for a single 8 x 16 dot shape so under these circumstances, this command is used. 8xyE uses data pointed to by the index register, and also a 'SIZE' value stored in VF, to draw the shape from the point (Vx, Vy). VF equals 1 allows the shape to be drawn exactly as defined. VF equals 2 draws the shape twice the size in both x and y directions, etc. Shape data is given by a series of bytes, from two to as many as required. (Shape data for this command has no maximum length.) The last byte is always 00, required to tell the interpreter when the end has been reached! Each byte, which is made up of eight bits, contains eight pieces of information; Figure 2 gives the key to this. The process of drawing a shape involves directing an invisible cursor about the screen (in eight possible directions), leaving trails as we go if required! A typical instruction to the cursor might be: PLOT UP 2 DOTS, which is coded as 1 0 0 0 1 0 1 0 using 1s and 0s. To get this in hexadecimal form, group data into groups of four : 1000 1010. For each group, convert the binary number into hexadecimal, in this example: 8A.

Example: A square. To draw a square, imagine the following cursor instructions: ▶

3 of 5.

LISTING 1.

```
7AE9 = F3 31 FF 8F 3E 09 32 3B    7C71 = 7F C9 21 20 7F 34 6E 26    7DF9 = 1A 2A 10 7F 06 64 CD 09
7AF1 = 78 CD 9C 7B 00 00 00 21    7C79 = 24 3A 21 7F 86 2B AE 32    7E01 = 7E 06 0A CD 09 7E 77 C9
7AF9 = FF 7F 22 1C 7F 21 00 82    7C81 = 21 7F A1 12 C9 79 E6 0F    7E09 = 0E 00 18 02 0C 90 B8 30
7B01 = 22 1E 7F 2A 1E 7F 46 23    7C89 = B7 20 02 3E 10 D9 47 D9    7E11 = FB 71 23 C9 1A 32 E5 7D
7B09 = 4E 23 22 1E 7F 78 E6 0F    7C91 = CD E5 7B 7E 26 00 87 87    7E19 = 32 5F 7E C9 1C 4B 06 00
7B11 = 5F 16 7F C6 80 08 78 1F    7C99 = 6F 29 29 29 44 4D 1A E6    7E21 = 58 2A 10 7F C3 F5 7E 1C
7B19 = 1F 1F E6 1E C6 2E 6F 26    7CA1 = 03 D9 CD 70 7E 5F 08 D9    7E29 = 4B 06 00 58 2A 10 7F EB
7B21 = 7B 08 47 7E 23 6E 67 CD    7CA9 = AF 32 0F 7F 2A 10 7F 56    7E31 = C3 28 7F 1A 4F 46 AF 32
7B29 = 7D 7E 18 D7 E9 7B 4E 7B    7CB1 = 23 5E 23 E5 2E 00 79 87    7E39 = 0F 7F CB 79 C0 78 FE 40
7B31 = 61 7D C4 7B D4 7B E0 7B    7CB9 = 28 09 CB 3A CB 1B CB 1D    7E41 = D0 87 87 6F 26 00 29 29
7B39 = F0 7B FC 7B FF 7C 03 7B    7CC1 = 3D 20 F7 7A CD E4 7C 7B    7E49 = 29 79 51 00 1F 1F E6 1F
7B41 = F6 7C 63 7C 68 7C 73 7C    7CC9 = CD E4 7C 7D CD E4 7C D9    7E51 = 4F 06 70 09 7A E6 03 C6
7B49 = 86 7D 00 7D 3D 7B B7 20    7CD1 = 79 C6 20 4F 78 CE 00 E6    7E59 = 6C 5F 16 7E 1A E6 FF 57
7B51 = 70 79 FE EE 20 0F 2A 1C    7CD9 = 07 47 08 5F 08 D9 E1 10    7E61 = AE 77 A2 BA C8 3E 01 32
7B59 = 7F 23 46 23 4E 22 1C 7F    7CE1 = CE D9 C9 D9 B7 28 11 60    7E69 = 0F 7F C9 C0 30 0C 03 4F
7B61 = ED 43 1E 7F C9 FE AE 38    7CE9 = 69 16 70 19 57 AE 77 A2    7E71 = D9 1A 1F 1F E6 1F C9 1A
7B69 = 09 20 2C 2A 1C 7F 22 10    7CF1 = BA 28 05 3E 01 32 0F 7F    7E79 = 4F 46 3A 0F 7F 5F DD 2A
7B71 = 7F C9 FE A8 20 0F 2A 1C    7CF9 = 7B 3C E6 1F 5F D9 C9 79    7E81 = 10 7F AF 32 0F 7F 18 11
7B79 = 7F 23 46 23 4E ED 43 10    7D01 = FE B3 28 19 30 1E D9 CD    7E89 = D5 7B 08 78 84 47 79 85
7B81 = 7F 22 1C 7F C9 FE A0 C0    7D09 = F4 2E D9 47 1A B8 79 28    7E91 = 4F 08 3D 20 F5 15 20 F1
7B89 = 2A 1C 7F ED 5B 10 7F 73    7D11 = 06 FE A1 C0 C3 D7 7B FE    7E99 = D1 CD CB 7E C8 CB 7F 28
7B91 = 2B 72 2B 22 1C 7F C9 FE    7D19 = 9E C0 C3 D7 7B D9 CD F4    7EA1 = E7 D5 7B 08 C5 D9 C1 CD
7B99 = E0 20 13 21 00 70 11 01    7D21 = 2E D9 12 C9 DB 20 06 05    7EA9 = 3B 7E D9 78 84 47 79 85
7BA1 = 70 75 01 FF 07 ED B0 3A    7D29 = 1F 30 02 10 FB 3E 37 80    7EB1 = 4F 08 3D 20 EE 15 20 EA
7BA9 = 3B 78 32 00 68 C9 E6 F0    7D31 = 6F 26 7D 7E 12 C9 00 0D    7EB9 = D1 CD CB 7E C8 CB 7F 20
7BB1 = FE C0 C0 79 17 17 17 17    7D39 = 2C 4D 20 2E 79 FE 29 28    7EC1 = E0 C5 D9 C1 CD 3B 7E D9
7BB9 = E6 10 C6 09 32 3B 78 18    7D41 = 48 30 44 FE 18 28 44 30    7EC9 = 18 BE 21 00 00 DD 7E 00
7BC1 = E6 C5 C9 2A 1C 7F ED 5B    7D49 = 51 FE 02 20 09 1A 6F 26    7ED1 = B7 C8 E6 07 20 02 3E 08
7BC9 = 1E 7F 73 2B 72 2B 22 1C    7D51 = 00 23 22 96 7D C9 FE 0A    7ED9 = 57 DD 7E 00 DD 23 CB 27
7BD1 = 7F 18 8D 1A B9 C0 2A 1E    7D59 = 20 1B D9 CD F4 2E B7 20    7EE1 = 28 01 2C CB 6F 28 01 2C
7BD9 = 7F 23 23 22 1E 7F C9 1A    7D61 = FA CD F4 2E B7 28 FA CD    7EE9 = CB 67 28 01 24 C5 5F 28
7BE1 = B9 C8 18 F2 79 1F 1F 1F    7D69 = F4 2E B7 28 F4 08 CD 50    7EF1 = 01 25 67 C9 ED B0 22 10
7BE9 = 1F E6 0F 6F 26 7F C9 CD    7D71 = 34 08 D9 12 C9 21 FE 8A    7EF9 = 7F C9 00 00 00 00 00 11
7BF1 = E5 7B 4E 18 DE CD E5 7B    7D79 = 7E FE E5 20 04 23 7E FE    7F01 = 11 11 11 11 11 11 11 11
7BF9 = 4E 18 E4 79 12 C9 1A 81    7D81 = 8B C2 E9 7A C3 FD 8A 18    7F09 = 11 11 11 11 11 11 11 11
7C01 = 12 C9 CD E5 7B 79 E6 0F    7D89 = 65 18 42 1A 6F 26 00 29    7F11 = 11 11 11 11 11 11 11 11
7C09 = FE 06 28 2F 30 47 FE 03    7D91 = 29 23 4D 44 21 2D 00 C3    7F19 = 11 11 11 11 11 11 11 11
7C11 = 28 13 30 15 B7 20 03 7E    7D99 = 5C 34 FE 1E 20 0C 2A 10    7F21 = 11 00 00 00 00 00 00 E0
7C19 = 12 C9 3D 20 04 1A B6 12    7DA1 = 7F 1A 4F 06 00 09 22 10    7F29 = 80 EB 22 10 7F C9 00 FC
7C21 = C9 1A A6 12 C9 1A AE 12    7DA9 = 7F C9 1A 6F D9 16 21 5A    7F31 = CC CC CC FC 30 30 30 30
7C29 = C9 FE 04 20 0A 1A 86 12    7DB1 = 4A D9 3A 3B 78 57 0E 10    7F39 = 30 FC 0C FC C0 FC FC 0C
7C31 = 3E 00 8F 32 0F 7F C9 1A    7DB9 = D9 CD 73 7C D9 AA 57 32    7F41 = FC 0C FC C0 C0 CC FC 0C
7C39 = 96 18 F4 D5 4E 1A 5F 06    7DC1 = 00 68 06 70 10 FE 0D 20    7F49 = FC C0 FC 0C FC FC C0 FC
7C41 = 08 16 00 62 6A 29 CB 11    7DC9 = EF 2D 20 EA C9 1A E6 0F    7F51 = CC FC FC 0C 0C 0C 0C FC
7C49 = 30 01 19 10 F8 D1 7D 12    7DD1 = 47 87 87 80 C6 30 5F 16    7F59 = CC FC CC FC FC CC FC 0C
7C51 = 7C 32 0F 7F C9 FE 0D CA    7DD9 = 7F 0E 05 41 21 12 7F 22    7F61 = FC FC CC FC CC CC F0 CC
7C59 = 34 7E FE 0E CA 78 7E C9    7DE1 = 10 7F 1A E6 FF 77 23 13    7F69 = FC CC F0 FC C0 C0 C0 FC
7C61 = 00 00 ED 43 10 7F C9 3A    7DE9 = 36 00 23 10 F5 C9 FE 65    7F71 = F0 CC CC CC F0 FC C0 FC
7C69 = 00 7F 6F 26 00 09 22 1E    7DF1 = 28 2A 30 20 FE 33 20 2F    7F79 = C0 FC FC C0 F0 C0 C0 00
                                                                      7F81 = 00 0
```

PLOT RIGHT 1 DOT, PLOT DOWN 1 DOT, PLOT LEFT 1 DOT, PLOT UP 1 DOT, END. From Figure 2, the codes are: 1 0 1 0 0 0 0 1, 1 0 0 1 0 0 0 1, 1 1 0 0 0 0 0 1, 1 0 0 0 1 0 0 1, '00'. That is: A1 91 C1 89 00 in hex. The program shown in Listing 2e uses this data to draw squares of random sizes all over the screen — try it!

Using sound commands

Table 3 shows pitch and duration values used in VZ/Chip-8 sound commands. The values given here are not tuned to a standard pitch, but are chosen so that the scale sounds reasonably tuneful when played.

To play a note, of duration V1, at pitch V2, use a segment of code like: F292 F118. Be sure to use the correct duration for the pitch under consideration, otherwise your tunes will sound uneven! You

4 of 5

LISTING 2a.

```
8200 — 6A 00 — put '00' to VA
       A2 0A — point I at 820A, the start of the shape data
       DA A9 — show a nine row shape at (VA,VA) ie (0,0)
       FB 0A — wait for a key to be pressed, store its value in VB
       F0 00 — end
820A — 0A A8 )
       09 98 |
       0A A8 |
       00 C0 |
       03 FO >          data for the shape in Figure 1.
       3C CF |
       00 C0 |
       03 30 |
       0C 0C )
```

LISTING 2b. RANDOM DOTS

```
8200 — CA 7F — put a random number (less than 7F) to VA
       CB 3F — put a random number (less than 3F) to VB
       CC FF — put a random number in VC
       FC CC — load the colour register with VC (ie: random colours)
       8A BD — plot a point at (VA,VB), a random screen position
       EF B3 — scan the keyboard and load the key pressed into VF
       3F 01 — if that key is '01' (the BREAK key), skip the next instruction
       12 00 — otherwise, go back to the start (plot another point)
       F0 00 — end; If BREAK key is down, the program will end
```

LISTING 2c. SCREEN FULL O' NUMBERS

```
8200 — 6F AA
       FF CC — load colour register with blue
       6A 00 — '00' to VA
       6B 00 — '00' to VB
8208 — 6C 00 — '00' to VC
820A — FC 29 — prepare to show VC as a number
       DA B5 — show the number at (VA,VB)
       7A 08 — increase VA by '08', the next number will be beside the one just shown
       7C 01 — increase VC by '01', the next number to display is one more than the last
       3C 10 — if the whole row has been shown, skip next instruction
       12 0A — otherwise, go back to 820A and show another number
       7B 08 — prepare to show on next row; increase VB by '08'
       3B 40 — if we have finished the last row, skip next instruction
       12 08 — otherwise, go back to 8208, begin a new row
       FF 0A — full screen; wait for a key to be pressed
       F0 00 — end
```

LISTING 2d. COUNTING

```
8200 — 6F FF FF CC 6A 00 22 28 6B 00 6C 00 7C 01 3C 00
8210 — 12 0C 7B 01 3B 06 12 0A 22 28 7A 01 4A 64 6A 00
8220 — EF B3 3F 01 12 06 F0 00 A2 40 FA 33 A2 40 F2 65
8230 — 6B 00 6C 00 F1 29 DB C5 7B 04 F2 29 DB C5 00 EE
8240 — 00 00 00 00
```

LISTING 2e. LOTS OF SQUARES

```
8200 — 65 FF F5 CC 6A 00 C6 7F C7 3F C5 1F 86 55 87 55
8210 — 85 54 75 01 8F 50 A2 24 86 7E 7A 01 3A 20 12 06
8220 — FF 0A F0 00 A1 91 C1 89 00 00
```

LISTING 2f. CHIRP

```
8200 — CE 07 7E 02 CA 0F FA 02 FE 18 7A 01 3A 18 12 06
8210 — EF B3 3F 01 12 00 F0 00
```

don't have to stick to the pitch and duration values shown in Table 3, so other effects, such as sirens, can be created. A sample sound program is shown in Listing 2f.

Saving completed programs

When you have written a program, and are satisfied that it does what you want, save it. There are two options here:

(i) Save the program *with* the editor. This is for programs which still have not been fully finished. Save all memory from 7AE9 to 8F30.

(ii) Save the program *without* the editor. This is for complete programs, only save memory from 7AE9 to the end of your Chip-8 program.

In either of the above cases, tape users will have to put up with the program running whenever it is loaded, so if the program is incomplete, make sure it ends otherwise you will never be able to edit it!

NOTE: We have had complaints from readers who could not get the editor listed last month running. Printed below are corrections to lines 70 and 380, and two new lines 770, 780 to be added. As well as this, we understand that in some issues of the magazine, the figure 32 between 90 and D6 in line 510 was printed so indistinctly as to look like 37. So if you have any problems after amending the listing, check line 510.

CORRECTIONS TO THE 'EDITOR' LISTING.
THE FOLLOWING ARE THE CORRECTED LINES.

```
70 IFT = 118550, PRINTUSR (1)

380 DATA10,78,C9,DF,20,E9,F1,11,
27,8E,C3,7B,8B,00,00,00

770 DATA8B,C3,Ec,8B,2D,22,A0,78,
C9,00,00,00
780DATA[illegible]
```

NB. THE LAST TWO LINES NEED TO BE ADDED TO THE PROGRAM. ●

Those who couldn't be bothered typing in Listing 1 can get a copy (tape only) by writing to 'Chris Griffin, PO Box 233, Diamond Creek, Victoria 3089' and including $5 with the letter (for postage, packing, tape, and my time!).

5 of 5.

LISTING 1.

```
7AE9 = F3 31 FF 8F 3E 09 32 3B
7AF1 = 78 CD 9C 7B 00 00 00 21
7AF9 = FF 7F 22 1C 7F 21 00 82
7B01 = 22 1E 7F 2A 1E 7F 46 23
7B09 = 4E 23 22 1E 7F 78 E6 0F
7B11 = 5F 16 7F C6 80 08 78 1F
7B19 = 1F 1F E6 1E C6 2E 6F 26
7B21 = 7B 08 47 7E 23 6E 67 CD
7B29 = 2D 7B 18 D7 E9 7B 4E 7B
7B31 = 61 7B C4 7B D4 7B E0 7B
7B39 = F0 7B FC 7B FF 7C 03 7B
7B41 = F6 7C 63 7C 68 7C 73 7C
7B49 = 86 7D 00 7D 3D 7B B7 20
7B51 = 70 79 FE EE 20 0F 2A 1C
7B59 = 7F 23 46 23 4E 22 1C 7F
7B61 = ED 43 1E 7F C9 FE AE 38
7B69 = 09 20 2C 2A 1C 7F 22 10
7B71 = 7F C9 FE A8 20 0F 2A 1C
7B79 = 7F 23 46 23 4E ED 43 10
7B81 = 7F 22 1C 7F C9 FE A0 C0
7B89 = 2A 1C 7F ED 5B 10 7F 73
7B91 = 2B 72 2B 22 1C 7F C9 FE
7B99 = E0 20 13 21 00 70 11 01
7BA1 = 70 75 01 FF 07 ED B0 3A
7BA9 = 3B 78 32 00 68 C9 E6 F0
7BB1 = FE C0 C0 79 17 17 17 17
7BB9 = E6 10 C6 09 32 3B 78 18
7BC1 = E6 C5 C9 2A 1C 7F ED 5B
7BC9 = 1E 7F 73 2B 72 2B 22 1C
7BD1 = 7F 18 8D 1A B9 C0 2A 1E
7BD9 = 7F 23 23 22 1E 7F C9 1A
7BE1 = B9 C8 18 F2 79 1F 1F 1F
7BE9 = 1F E6 0F 6F 26 7F C9 CD
7BF1 = E5 7B 4E 18 DE CD E5 7B
7BF9 = 4E 18 E4 79 12 C9 1A 81
7C01 = 12 C9 CD E5 7B 79 E6 0F
7C09 = FE 06 28 2F 30 47 FE 03
7C11 = 28 13 30 15 B7 20 03 7E
7C19 = 12 C9 3D 20 04 1A B6 12
7C21 = C9 1A A6 12 C9 1A AE 12
7C29 = C9 FE 04 20 0A 1A 86 12
7C31 = 3E 00 8F 32 0F 7F C9 1A
7C39 = 96 18 F4 D5 4E 1A 5F 06
7C41 = 08 16 00 62 6A 29 CB 11
7C49 = 30 01 19 10 F8 D1 7D 12
7C51 = 7C 32 0F 7F C9 FE 0D CA
7C59 = 34 7E FE 0E CA 78 7E C9
7C61 = 00 00 ED 43 10 7F C9 3A
7C69 = 00 7F 6F 26 00 09 22 1E
```

```
7C71 = 7F C9 21 20 7F 34 6E 26
7C79 = 24 3A 21 7F 86 2B AE 32
7C81 = 21 7F A1 12 C9 79 E6 0F
7C89 = B7 20 02 3E 10 D9 47 D9
7C91 = CD E5 7B 7E 26 00 87 87
7C39 = 6F 29 29 29 44 4D 1A E6
7CA1 = 03 D9 CD 70 7E 5F 08 D9
7CA9 = AF 32 0F 7F 2A 10 7F 56
7CB1 = 23 5E 23 E5 2E 00 79 87
7CB9 = 28 09 CB 3A CB 1B CB 1D
7CC1 = 3D 20 F7 7A CD E4 7C 7B
7CC9 = CD E4 7C 7D CD E4 7C D9
7CD1 = 79 C6 20 4F 78 CE 00 E6
7CD9 = 07 47 08 5F 08 D9 E1 10
7CE1 = CE D9 C9 D9 B7 28 11 60
7CE9 = 69 16 70 19 57 AE 77 A2
7CF1 = BA 28 05 3E 01 32 0F 7F
7CF9 = 7B 3C E6 1F 5F D9 C9 79
7D01 = FE B3 28 19 30 1E D9 CD
7D09 = F4 2E D9 47 1A B8 79 28
7D11 = 06 FE A1 C0 C3 D7 7B FE
7D19 = 9E C0 C3 D7 7B D9 CD F4
7D21 = 2E D9 12 C9 DB 20 06 05
7D29 = 1F 30 02 10 FB 3E 37 80
7D31 = 6F 26 7D 7E 12 C9 00 0D
7D39 = 2C 4D 20 2E 79 FE 29 28
7D41 = 48 30 44 FE 18 28 44 30
7D49 = 51 FE 02 20 09 1A 6F 26
7D51 = 00 23 22 96 7D C9 FE 0A
7D59 = 20 1B D9 CD F4 2E B7 20
7D61 = FA CD F4 2E B7 28 FA CD
7D69 = F4 2E B7 28 F4 08 CD 50
7D71 = 34 08 D9 12 C9 21 FE 8A
7D79 = 7E FE E5 20 04 23 7E FE
7D81 = 8B C2 E9 7A C3 FD 8A 18
7D89 = 65 18 42 1A 6F 26 00 29
7D91 = 29 23 4D 44 21 2D 00 C3
7D99 = 5C 34 FE 1E 20 0C 2A 10
7DA1 = 7F 1A 4F 06 00 09 22 10
7DA9 = 7F C9 1A 6F D9 16 21 5A
7DB1 = 4A D9 3A 3B 78 57 0E 10
7DB9 = D9 CD 73 7C D9 AA 57 32
7DC1 = 00 68 06 70 10 FE 0D 20
7DC9 = EF 2D 20 EA C9 1A E6 0F
7DD1 = 47 87 87 80 C6 30 5F 16
7DD9 = 7F 0E 05 41 21 12 7F 22
7DE1 = 10 7F 1A E6 FF 77 23 13
7DE9 = 36 00 23 10 F5 C9 FE 65
7DF1 = 28 2A 30 20 FE 33 20 2F
```

```
7DF9 = 1A 2A 10 7F 06 64 CD 09
7E01 = 7E 06 0A CD 09 7E 77 C9
7E09 = 0E 00 18 02 0C 90 B8 30
7E11 = FB 71 23 C9 1A 32 E5 7D
7E19 = 32 5F 7E C9 1C 4B 06 00
7E21 = 58 2A 10 7F C3 F5 7E 1C
7E29 = 4B 06 00 58 2A 10 7F EB
7E31 = C3 28 7F 1A 4F 46 AF 32
7E39 = 0F 7F CB 79 C0 78 FE 40
7E41 = D0 87 87 6F 26 00 29 29
7E49 = 29 79 51 00 1F 1F E6 1F
7E51 = 4F 06 70 09 7A E6 03 C6
7E59 = 6C 5F 16 7E 1A E6 FF 57
7E61 = AE 77 A2 BA C8 3E 01 32
7E69 = 0F 7F C9 C0 30 0C 03 4F
7E71 = D9 1A 1F 1F E6 1F C9 1A
7E79 = 4F 46 3A 0F 7F 5F DD 2A
7E81 = 10 7F AF 32 0F 7F 18 11
7E89 = D5 7B 08 78 84 47 79 85
7E91 = 4F 08 3D 20 F5 15 20 F1
7E99 = D1 CD CB 7E C8 CB 7F 28
7EA1 = E7 D5 7B 08 C5 D9 C1 CD
7EA9 = 3B 7E D9 78 84 47 79 85
7EB1 = 4F 08 3D 20 EE 15 20 EA
7EB9 = D1 CD CB 7E C8 CB 7F 20
7EC1 = E0 C5 D9 C1 CD 3B 7E D9
7EC9 = 18 BE 21 00 00 DD 7E C8
7ED1 = B7 C8 E6 07 20 02 3E 08
7ED9 = 57 DD 7E 00 DD 23 CB 77
7EE1 = 28 01 2D CB 6F 28 01 2C
7EE9 = CB 67 28 01 24 CB 5F 28
7EF1 = 01 25 B7 C9 ED B0 22 10
7EF9 = 7F C9 00 00 00 00 00 11
7F01 = 11 11 11 11 11 11 11 11
7F09 = 11 11 11 11 11 11 11 11
7F11 = 11 11 11 11 11 11 11 11
7F19 = 11 11 11 11 11 11 11 11
7F21 = 11 00 00 00 00 00 00 ED
7F29 = B0 EB 22 10 7F C9 00 FC
7F31 = CC CC CC FC 30 30 30 30
7F39 = 30 FC 0C FC C0 FC FC 0C
7F41 = FC 0C FC C0 C0 CC FC 0C
7F49 = FC C0 FC 0C FC FC C0 FC
7F51 = CC FC FC 0C 0C 0C 0C FC
7F59 = CC FC CC FC FC CC FC 0C
7F61 = FC FC CC FC CC CC F0 CC
7F69 = FC CC F0 FC C0 C0 C0 FC
7F71 = F0 CC CC CC F0 FC C0 FC
7F79 = C0 FC FC C0 F0 C0 C0 00
7F81 = 00 0
```

PLOT RIGHT 1 DOT, PLOT DOWN 1 DOT, PLOT LEFT 1 DOT, PLOT UP 1 DOT, END. From Figure 2, the codes are: 10100001, 10010001, 11000001, 10001001, '00'. That is: A1 91 C1 89 00 in hex. The program shown in Listing 2e uses this data to draw squares of random sizes all over the screen — try it!

Using sound commands

Table 3 shows pitch and duration values used in VZ/Chip-8 sound commands. The values given here are not tuned to a standard pitch, but are chosen so that the sc sounds reasonably tuneful when played.

To play a note, of duration V1, at pi V2, use a segment of code like: F. F118. Be sure to use the correct durat for the pitch under consideration, otl wise your tunes will sound uneven! Y

Hardware and software aspects of screen handling on the VZ-200/300 Part 1

Bob Kitch

This article describes the hardware aspects of the Motorola MC6847 Video Display Generator chip which is used in a number of microcomputers. Although this is an older device and lacks some of the features of newer chips, it is nevertheless a well-used device and is quite easy to interface and comprehend. To illustrate the MC6847, its use in the VZ-200 and VZ-300 computers is detailed. Additionally, some software implementations are explained and some simple hardware modifications to the VZ are given to improve screen resolution and display appearance.

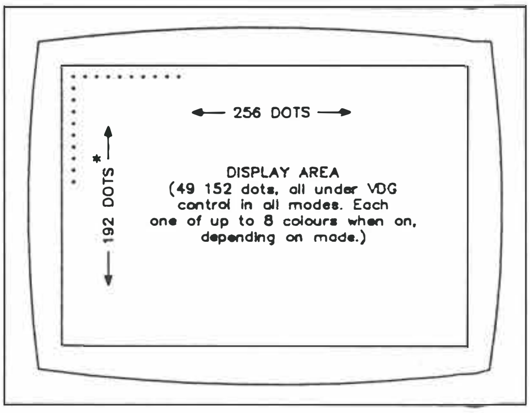
THE MOTOROLA MC6847 Video Display Generator (VDG) chip (sometimes referred to as a Cathode Ray Tube Controller — CRTC) is used to interface data read from the video RAM section of memory and to produce a modulated RF video signal or monitor output. The MC6847 is capable of operating in 14 different display modes. However, only a few of these are usually implemented in a particular installation. The MC6847 was conceived as one of the family of devices to interface with the Motorola M6800 and M68000 microprocessor families, but it can easily be adapted to other microprocessors. The VDG can be found in video games, home computers, process control displays, communications and graphics applications.

The VDG has the complex task of converting data from the screen memory into the form necessary for the raster scan display used in television and monitors. On these devices, the image is 'drawn' on the screen one horizontal scan line at a time. The 'spot' moves across the screen from right to left and its brightness or colour (chroma) is varied to produce the required display. In practice, the whole screen is built up in two passes, the first on even-numbered lines and the second on odd-numbered lines, by a process called 'interlacing' which helps to avoid flicker. The process occurs every 20 ms, or 50 half-frames are drawn every second.

Two types of VDG chip are produced by Motorola — the MC6847 for non-interlaced displays and the MC6847Y which interlaces the video display. the suffix 'P' after the device number identifies a plastic package. An enhanced version — the MC6847T1 — is also available but it is not strictly compatible with the MC6847 as it requires less external circuitry and has some additional features.

A timing or clock pulse is required to tie the scan rate and memory access cycles of the VDG in with that of the microprocessor (MPU) — otherwise chaos would reign on the bus systems! An external (to the VDG) clock is used to synchronise both the VDG and the MPU. A clock frequency of 3.58 MHz is usually selected to give the correct scan rates. If a common clock is used then often the speed of the MPU is restricted by the video display.

The format of the display area under the control of the VDG is actually 256 'dots' across by 192 'dots' down giving a total of 49 152 fundamental picture elements (pixels) under the 'control' of the VDG. Each pixel may be one of up to eight colours, depending upon the mode selected (see Figure 1.).


256 DOTS
192 DOTS *
DISPLAY AREA
(49 152 dots, all under VDG control in all modes. Each one of up to 8 colours when on, depending on mode.)

* One on each non–interlaced line. For interlace, the lines of the odd field are copied into the even field thus doubling the number of displayed dots.

Figure 1. Typical Format of the Monitor Screen. The border is black in Alphanumeric and Semigraphic modes and green or buff in Graphic modes.

As you will have observed, the MC6847 does not utilise the entire video screen. The standard video screen consists of 262 scan lines extending across the screen, but the usable display window is offset from the top by 25 lines and extends 192 lines down the screen with a further 25 lines at the bottom being offset. Across the screen, the timing pulses are blanked-off to reduce the useable horizontal width. The linearity of images is better in the central portion of a screen and this is used by the VDG.

The screen is 'memory mapped' with each pixel on the screen being represented by a byte (or a number of bits thereof) in the video RAM. There is a one-to-one correspondence between the X-Y location of the pixel on the screen and the address of its control information in memory. The sequence of memory addresses, which are accessed to extract data to be converted to a video signal, is controlled by the VDG. The VDG also keeps track of the position of the moving spot and produces the necessary timing signals to synchronise the display to the computer. It produces, for instance, the horizon- ▶

1 of 4

tal sync pulse to indicate when the end of the video line has been reached so that the spot can 'flyback' to the beginning of the next scan line. This pulse also permits the MPU to access video memory during the blanking period, thereby avoiding flicker.

The decoding of the data input to the VDG is usually done by a character generator. This may be a pre-programmed, on-chip ROM in the MC6847 or an external, perhaps programmable, character generator.

The display modes that the MC6847 may operate in are set out in Table 1. this tabulation summarises much of the information about the VDG chip. The way in which these features are selected is in-line with most digital devices. The pin assignment diagram for the MC6847 is shown on Figure 2. The chip is an N-channel, silicon gate device with most signals being TTL compatible. The device is housed in a 40-pin DIL package. The amount of memory required by the various display modes is a trade-off against element size or resolution of the display in pixels. This feature will become more apparent later.

The lines into, or out of, the VDG can be grouped into six classes but classes i) to iv) are the most important to this discussion.

i) **Address Lines.** (DAO — DA12) These permit up to 8K of video memory to be directly addressed, although only 6K is ever required. The absolute location of the video memory in the computer system will depend upon the address decoding used. The starting address is located at the upper left-hand corner of the screen. The activity of the address lines is regulated by the *MS pin and the display mode selected.

ii) **Date Lines.** (DDO — DD7) These are used to input values in RAM memory to be mapped onto the screen. The values are decoded within the chip with repsect to shape, luminance and chroma (see later).

iii) **Mode control Lines.** There are eight important lines into the VDG which control the 14 display modes. These are detailed in Table 1. Three major types of display may be selected: **(a)** Alphanumerics, **(b)** Semigraphics and, **(c)** Graphics.

The implementation of these displays within the VDG is quite different in each case.

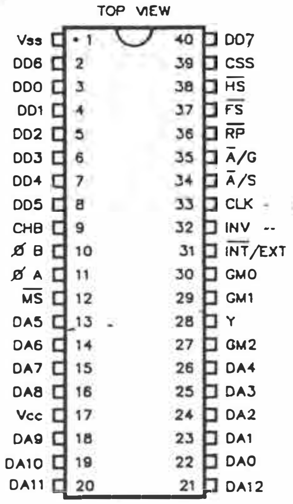


Figure 2. Pin-out for Motorola MC6847 Video Display Generator chip as used in the VZ computers.

Switching the screen to Alphanumerics or Graphics mode is determined by the (*A/G) line.

Switching the screen between Alphanumerics or Semigraphics mode is set by the (*A/S) line.

Selection of the internal (on-chip) or external character sets held in ROM is set by the (*INT/EXT) line. In Semigraphics mode this line determines whether SG4 or SG6 mode is selected.

Normal or inverse Alphanumeric displays are set by the (INV) line. Three lines (GMO, GM1, GM2) are used to select one-of-eight Graphics modes to be used.

An eighth control line (CSS) selects the colour set to be used in the particular mode selected. Most modes have two colour sets available.

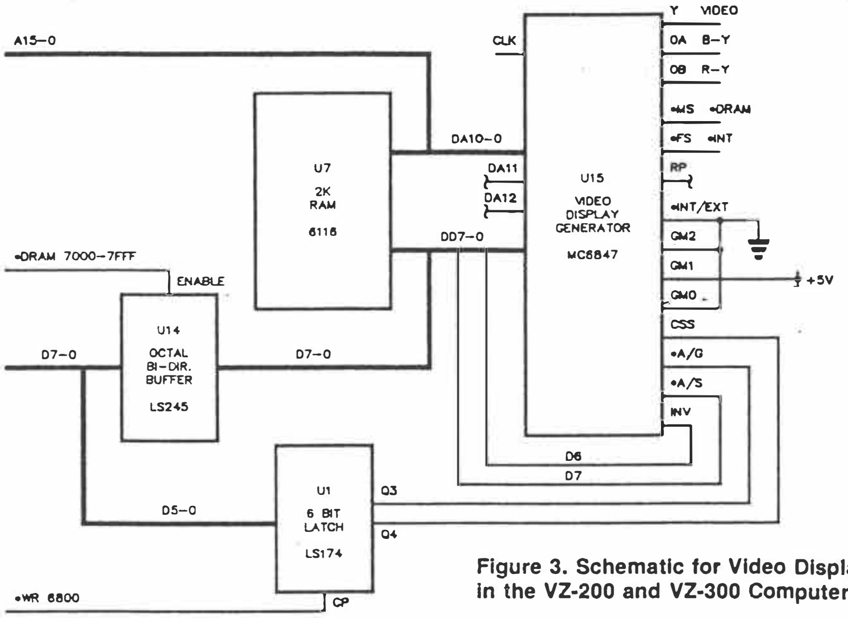
In Alphanumeric and Semigraphics 4 modes, one-of-two background colours is selected and in Semigraphics 6 and Full Graphics modes one-of-two colour sets is selected.

The operating mode of *A/S, *INT/EXT, CSS and INV may be changed on a character by character basis in Alphanumerics and Semigraphics mode.

TABLE 1:
SUMMARY OF DISPLAY MODES FOR MC6847 VDG

	colours	bytes	memory	element	Control	Lines					
	available	video RAM	mapping	size	*A/G	*A/S	*INT/EXT	INV	GMO	GM1	GM2
Four ALPHANUMERIC Display Modes											
i) Internal ROM Alphanumerics	2	512	byte	BX12	0	0	0	0	x	x	x
ii) Internal ROM Alphanumerics — Inverted	2	512	byte	BX12	0	0	0	1	x	x	x
iii) External ROM Alphanumerics	2	512	byte	BX12	0	0	1	0	x	x	x
iv) External ROM Alphanumerics — Inverted	2	512	byte	BX12	0	0	1	1	x	x	x
Two SEMI-GRAPHIC Display Modes											
v) 32 by 16 Semigraphics 4 (SG4)	8	512	byte	BX12	0	1	0	x	x	x	x
vi) 32 By 16 Semigraphics 6 (SG6)	4	512	byte	BX12	0	1	1	x	x	x	x
Eight GRAPHIC Display Modes											
vii) 64 by 64 Colour Graphics One (CG1)	4	1024	2 bit	3x4	1	x	x	x	0	0	0
iix) 128 by 64 Resolution Graphics One (RG1)	2	1024	1 bit	2x3	1	x	x	x	0	0	1
ix) 128 by 64 Colour Graphics Two (CG2)	4	2048	2 bit	2x3	1	x	x	x	0	1	0
x) 128 by 96 Resolution Graphics Two (RG2)	2	1536	1 bit	2x2	1	x	x	x	0	1	1
xi) 128 by 96 Colour Graphics Three (CG3)	4	3072	2 bit	2x2	1	x	x	x	1	0	0
xii) 128 by 192 Resolution Graphics Three (RG3)	2	3072	1 bit	sx1	1	x	x	x	1	0	1
xiii) 128 by 192 Colour Graphics Six (CG6)	4	6144	2 bit	2x1	1	x	x	x	1	1	0
xiv) 256 by 192 Resolution Graphics Six (RG6)	2	6144	1 bit	1x1	1	x	x	x	1	1	1

The IEEE standard for electrical state relationships uses the suffix '' instead of the overbar '‾' to designate when an electrical signal is active low.*



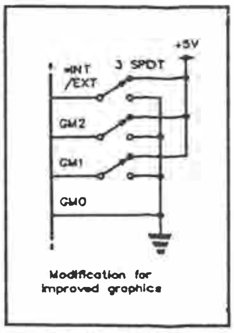


Figure 3. Schematic for Video Display in the VZ-200 and VZ-300 Computers.

iv) **Power Supply.**
Vss: 0 V supply — normally ground.
Vcc: +5 supply.

v) **Video Lines.**
These are four analogue signals:

OA B-Y chroma — a three-level signal used in combination with OB and Y to specify one-of-eight colours.

OB R-Y chroma — a four-level signal; the fourth is used as colour burst timing reference.

Y luminance — a six-level signal containing composite sync, blanking and four levels of luminance.

CHB chroma bias or a test point — not used in applications.

vi) **Device Synchronising Controls.**

*MS memory select, three-state control to allow the MPU to address the video RAM.

CLK 3.579 MHz clock.

*FS field sync to indicate the end of the active display area during which time the MPU may have access to the video RAM without causing undesirable flicker on the screen.

*HS horizontal sync to the TV receiver.

*RP row preset — important when an external character generator ROM is used.

From this brief description, a grasp of how the VDG operates may be gleaned. We will now examine how this particular VDG chip is used in a home computer application — the VZ computer.

The MC6847 in the VZ-200/300 computer

In the VZ computer a number of display modes using the MC6847 are available. Specifically, modes (i), (ii), (v) and (ix) on Table 1 are implemented as standard on the VZ. These modes are 'soft switched' or software selectable from the ROM-resident BASIC and will be described in detail later in this article.

The video display system in the VZ consists of a number of components or 'blocks' — but the heart of the display system is the VDG just described. This device interfaces with 2K of dynamic video RAM which occupies 7000H to 77FFH of the memory map for the Z8OA MPU used in the VZs. Additionally, a hex write-only latch mapped at 6800H (but extending to 6FFFH due to simplified address decoding) controls, via software, the display modes implemented on the VZ.

The analogue outputs from the BDG are processed by further video circuitry which need not concern us here. All of these blocks are synchronised by a 3.58 MHz clock. This is an instance where the full speed of the Z80A (4 MHz) is not realised due to impositions by the video display.

More significantly however, the architecture of the VZ has only allowed 2K of RAM for the video display. This effectively prohibits the implementation of some of the hi-res graphics modes. [Specifically, modes (xi) to (xiv) in Table 1]. The VZ does not contain an external character generator ROM and relies entirely upon the VDG on-chip character ROM. Clearly, the VZ is manufactured to a price (and a very attractive one at that!) and was designed to interface with Microsoft's BASIC Level II ROM routines. Despite these comments, there are opportunities to make a few slight and simple changes to the hardware around the VDG to implement additional display modes with improved resolution. It is also possible to add an external character generator — but more of these later.

Figure 3 is a diagrammatic representation of the way in which the MC6847 VDG is interconnected in the VZ computers. The address lines DAO-DA10 (11 lines) are connected to U7 — a 6116 2K RAM chip — which is mapped as the video RAM section of memory. Lines DA11 and DA12 are not connected, thereby limiting the addressable video memory to 2K. Data lines DDO-DD7 (eight lines) are connected into the data bus from the MPU of which the 2K video RAM memory of course forms a part. The way in which the eight control lines are connected is of interest as these determine the type of displays available on the VZ.

Reference to Table 1 will indicate how the control lines are configured. The Graphics display group consist of GM0, GM1 and GM2. As can be seen from Figure 3, both GM0 and GM2 are tied low (to ground) whilst GM1 is tied high, to the +5 V Supply. Similarly, *INT/EXT is permanently tied low, thereby enabling the on-chip character generator ROM. The configuration of GM0-GM2 to 010B means that only Colour Graphics Two (CG2) is implemented when Graphics mode is selected. ▶

3 of 4.

The remaining four control lines are interesting as they are not 'hard-wired' but are set up to be 'soft switched' — although two quite different techniques are used.

The INV line is connected to bit 6, or DD6, of the data bus. Thus, whilst in Alphanumeric mode, the second most significant bit of a byte contained in video RAM controls whether a normal or inverse character is displayed. The line that selects between Alphanumeric and Semigraphic modes — *A/S — is similarly connected to the most significant bit or DD7. thus this bit determines whether the VDG should interpret a particular byte as an ASCII character or a graphics shape.

The remaining two lines are connected into the Output Latch mapped into 6800H. As mentioned before, this is a 6-bit write-only latch. It permits certain software commands to set or reset a particular bit of the latch and hence switch or control specific hardware interfaces. Figure 4 is a schematic of the portions of the latch which is of interest to us here. A copy of the latch is held in RAM at location 783BH. The *A/G line, which selects between hi- or lo-res screens, is connected to bit 3 of the Output Latch. If this bit is low or 0, then the screen is in lo-res mode which corresponds to Alphanumeric and Semigraphic modes. If the bit is high or 1, then hi-res or Graphics (CG2) mode is selected. It is quite simple to see that the MODE (X) command in BASIC directly sets this bit of the latch — where X maybe 1 or 0. Note that bit 3 of the latch corresponds to a value of 0BH on the latch.

The Colour Select line (CSS) is connected to bit 4 on the latch which maps as a value of 0FH. The effect of this line differs according to the mode selected. The CSS pin selects the background colour of the display and in so doing determines the colour set which may be displayed. When CSS is low or 0 the background colour is green, but if set high or 1, then in lo-res the background colour is orange, but if in hi-res then the background is buff. Sounds a little confusing — but actually it isn't, given a little thought and reflection on Table 1 and Figure 1. Furthermore, in hi-res mode this pin selects which of the two colour sets (each containing four colours) will be selected. Colour set 0 consists of green, yellow, blue and red, whilst colour set 1 consists of buff, cyan, magenta and orange. Clearly, this pin is set by the COLOR F, B command where B determines the background colour and F determines foreground colour.

An understanding of the operation of the mode control lines gives a good insight into how the BASIC interpreter interfaces with the hardware and the real world via the screen display.

For the hardware enthusiasts, and others closely following this article, the penny should have dropped as to how other screen modes can be made selectable on the VZ by some simple hardware alterations.

Improved graphics on the VZ computer

One of the disappointing features of the graphics capability of the VZ is that the Semigraphics (SG4) and Graphics (CG2) modes have rectangular characters and elements which considerably detract from the appearance of the displays. This feature can be remedied.

The following simple hardware modifications are outlined for those who feel they are competent tackle it. They involve the installation of three switches on the VZ. Figure 3 provides an indication of what is required.

If *INT/EXT can be switched high, then Semigraphic mode SG6 becomes available on the computer. This has the advantage of giving higher screen resolution and, although the characters are still rectangular, their elements are square rather than rectangular as in the standard implementation of SG4 mode.

In Graphics mode, only CG2 is available in the VZ. By switching GM1 and GM2 it is possible with the 2K of video memory to implement a further three modes (CG1, RG1 and RG2). There is little point in switching GM0 as there is insufficient memory to cover modes (xi) to (xiv). The element size in SG6 and CG1 is the same (3x4 pixels) and so there is little to choose between them — although their usage of memory is different and the characters in SG6 mode can be 'specified' through the keyboard as is done in SG4 mode on the VZ.

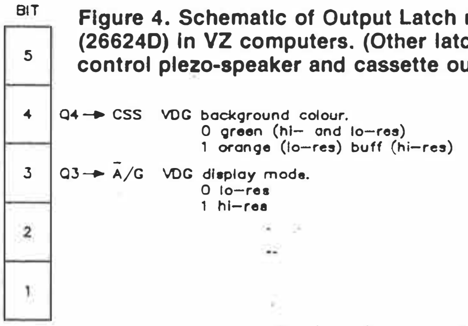

BIT
5
4
3
2
1
Q4 → CSS VDG background colour.
0 green (hi- and lo-res)
1 orange (lo-res) buff (hi-res)
Q3 → $\bar{A}$/G VDG display mode.
0 lo-res
1 hi-res

Figure 4. Schematic of Output Latch mapped at 6800H (26624D) in VZ computers. (Other latches are used to control piezo-speaker and cassette output)

RG1 has the same resolution as the standard MODE (1) display but is only two-colour and consequently uses only half the memory space. the real benefit of adding the switches is in obtaining RG2 mode on the VZ. Although this only two colour, the element size is 2x2 pixels and is square. This is a great mode for plotting graphs for instance, where the screen resolution is 128 elements across by 96 elements down the screen.

To achieve this modification, use three SPDT toggle switches. Wire one side of each switch to +5 V, or pin 17 on the VDG, and wire the other side of each switch to ground or pin 1 of the chip. Cut the tracks leading from pins 27, 29 and 31 (GM2, GM1 and *INT/EXT) and wire the chip side to the centre terminal of a switch. This enables the three control lines to be switched high or low. (See inset on Figure 3.)

There you have it! It remains now to develop suitable software to drive these additional modes. The possibilities opened by the 'square' modes of SG6 and RG2 are exciting. (Who is going to submit some drivers for this conversion?)

As an afterthought, whilst you have got the VZ on the bench, why not add a RESET switch? A normally closed push-button switch inserted into the 'reset on power-up' line overcomes the annoying business of powering-down the VZ for resetting.

— continued next month.

4 of 4.

Hardware and software aspects of screen handling on the VZ-200/300

Concluding with coverage of the software interface in the VZ and the MC6847 VDG, looking at the standard screen modes.

Part 2

Bob Kitch

IT IS NOW OPPORTUNE to briefly discuss the software interface in the VZ and the VDG. I will only discuss the standard screen modes used on the VZ — not the additional modes mentioned in Part 1.

Lo-res/Text/Mode (O).

In the lo-res mode the screen is formatted into 16 lines down the usable window with each line containing 32 characters. Thereby providing 512 addressable characters on the screen. A quick calculation (or look at Table 1) will show that each character is composed of 8 by 12 pixels (or dots). Furthermore, each character is 'described' in a single byte in the video RAM section of memory. The upper left-hand character on the screen is memory mapped onto address 7000H (28672D), and the lower right-hand character is mapped onto 7000H + 1FFH (29183D). A memory map for the lo-res screen is given in Figure 5.

A formula is often used to calculate the address of a particular character on the screen. Let AA be the position of the character ACROSS the line (which ranges from 0 to 31) and let AD be the line number DOWN the screen (ranges from 0 to 15). i.e: working in the SE quadrant of an X-Y axis system. The relationship between (AA,AD) and the address in RAM is —

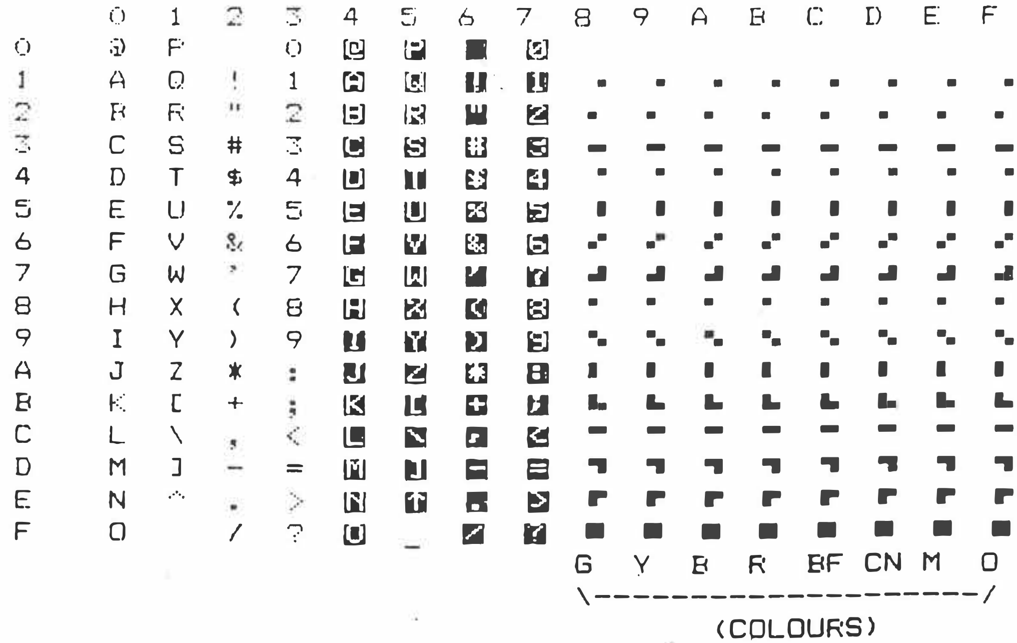
MAPPED ADDRESS = START ADDRESS + (32 • AD + AA) or
Addr = 28672 + (32 • AD + AA)

This calculation is often used in games to POKE values into selected memory locations or when screen formatting via the use of the PRINT@ statement where it is performed 'transparently'.

When the VZ is 'soft switched' to MODE (0) three of the modes in the VDG become available. There are internal ROM Alphanumerics (Normal and Inverse) and Semigraphics 4. There is no user-definable external character generator available in a standard VZ and also the Semigraphic 6 mode is not implemented due to hardware limitations. (Although I understand that the LASER 200 had SG6 rather than SG4 implemented as standard — but see previous section).

Let's digress for a while to describe how the on-chip customised character generator located in ROM on the VDG actually formats the 8 by 12 pixels to form each character. Firstly, in text mode. Table 2 shows the actual character set with corresponding codes resident in the VDG ROM. Figure 6 shows a typical character in Alphanumeric Mode (Internal). The spacing between characters across the line and between lines is set by the format held in the character generator. A Non-ASCII type character code is used on the VZ such that lower case (and control) ASCII characters are not represented. The 'lower case' ASCII values are used to signal 'inverse' characters by setting bit 6 high.

An Alphanumeric character in 'normal' mode is colour selectable as either green or orange with a black background. In 'inverse' mode, the character is black with the background



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	P		0	@	P	█	0								
1	A	Q	!	1	A	Q	!	1	▗	▗	▗	▗	▗	▗	▗	▗
2	B	R	"	2	B	R	"	2	▖	▖	▖	▖	▖	▖	▖	▖
3	C	S	#	3	C	S	#	3	▄	▄	▄	▄	▄	▄	▄	▄
4	D	T	$	4	D	T	$	4	▝	▝	▝	▝	▝	▝	▝	▝
5	E	U	%	5	E	U	%	5	▐	▐	▐	▐	▐	▐	▐	▐
6	F	V	&	6	F	V	&	6	▞	▞	▞	▞	▞	▞	▞	▞
7	G	W	'	7	G	W	'	7	▟	▟	▟	▟	▟	▟	▟	▟
8	H	X	(	8	H	X	(	8	▘	▘	▘	▘	▘	▘	▘	▘
9	I	Y	)	9	I	Y	)	9	▚	▚	▚	▚	▚	▚	▚	▚
A	J	Z	*	:	J	Z	*	:	▌	▌	▌	▌	▌	▌	▌	▌
B	K	[	+	;	K	[	+	;	▙	▙	▙	▙	▙	▙	▙	▙
C	L	\	,	<	L	\	,	<	▀	▀	▀	▀	▀	▀	▀	▀
D	M	]	–	=	M	]	–	=	▜	▜	▜	▜	▜	▜	▜	▜
E	N	^	.	>	N	↑	.	>	▛	▛	▛	▛	▛	▛	▛	▛
F	O		/	?	O	_	/	?	█	█	█	█	█	█	█	█
									G	Y	B	R	BF	CN	M	O

\--------------------/

(COLOURS)

TABLE 2.
Alphanumeric and Semigraphic 4 character set for the VZ-200 and VZ-300 held in MC6847 on-chip ROM.
(Users — note errors in shape table held in VZ ROM for inverse J, X, 3 and 5).

1 of 4.

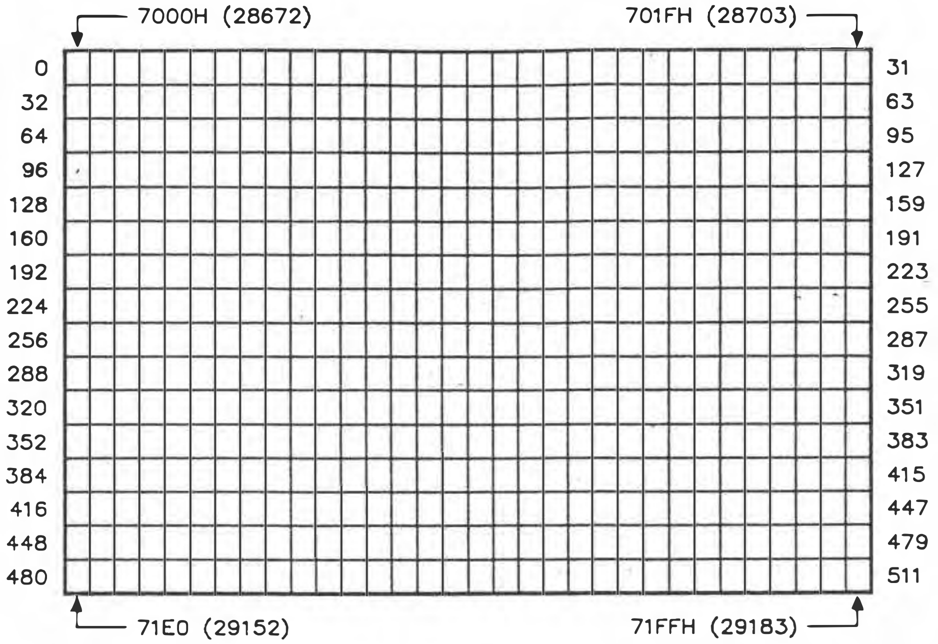


Figure 5. Screen addressing for MODE (0) or lo-res displays on VZ computers. This mode corresponds to Alphanumeric and Semigraphic 4 on VDG and is 32 by 16 characters in size. Each character is byte-mapped as indicated.

being selectable from green or orange. Remember that the Inverse mode of the MC6847 is set by bit 6 of the data value contained in video RAM. (see also Figures 1 and 6).

An understanding of this involves looking at individual bits within the bytes and also looking at how these bits can control and reset certain control lines on the VDG (as outlined in Part 1).

In text mode there are 64 characters in each of the Normal (0-63) and Inverse (64-127) sets. This implies that a 6-bit code is used to encode the character shape and that bit 6 determines whether Normal or Inverse.

For example:—

b7	b6	b5	b4	b3	b2	b1	b0	
0	0	1	0	0	1	0	1	Binary = 37D or '%' normal.
0	1	1	0	0	1	0	1	Binary = 101D or '%' inverse.

Note the way that bit 6 determines normal/inverse. Also note that bit 7 does not change. The most significant bit (MSB) is used to indicate text character to the on-chip ROM.

In summary, for the character source, a 6-bit ASCII code is used to call the elemnent from the on-chip ROM, the seventh bit indicates normal or inverse illumination, and the eighth bit is held low to indicate Alphanumeric mode.

b7	b6	b5	b4	b3	b2	b1	b0
alpa *A/S	inv INV	6-bit		ASCII			

In graphics mode the Semigraphics 4 mode of the VDG is used. The 8 by 12 pixel character is divided into four 'rectangular' quadrants of size 4 by 6. The quadrants are 'psuedo-addressable' by selecting the correct area as shown on Figure 7.

In Semigraphics mode, a more comprehensive form of encoding is used. The character codes extend from 128 to 255, implying that the MSB (or bit 7) is set to 1 (or high) to indicate that a graphics character is encoded in the byte. The graphic block character contains 16 discrete patterns involving 'switching' on or off the four quadrants. The four low-order bits handle a quadrant a piece (refer Figure 7). Additionally one-of-eight illumination colours is encoded in the next three bits (bits 6 to 4).

For example:—

b7	b6	b5	b4	b3	b2	b1	b0	
0	0	1	0	0	1	0	1	Binary = 217D or ▙ cyan
0	1	1	0	0	1	0	1	Binary = 145D or ▙ yellow

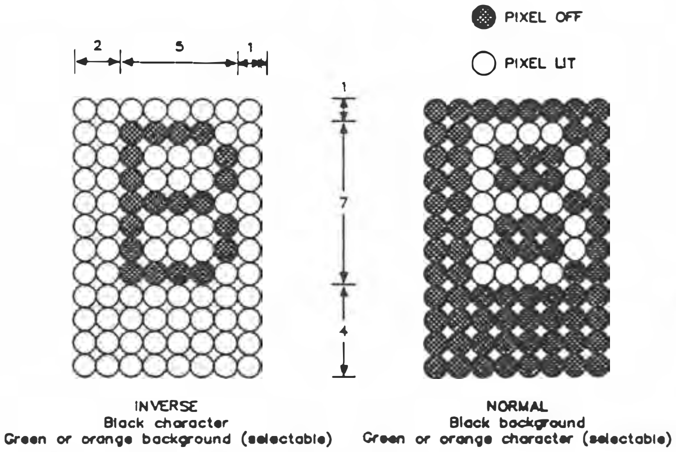
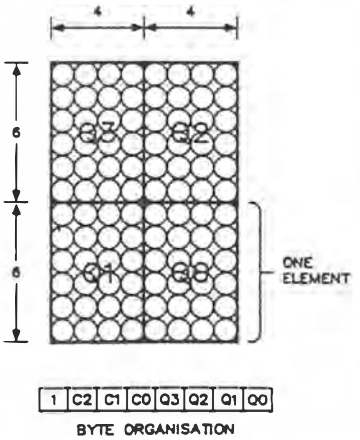

PIXEL OFF
PIXEL LIT
2
5
1
1
7
4
INVERSE
Black character
Green or orange background (selectable)
NORMAL
Black background
Green or orange character (selectable)

Figure 6. Format of Alphanumeric Mode — Internal on MC6847. Each character is 12 by 8 pixels and each screen is 32 by 16 characters. A 6-bit ASCII code specifies the character from an on-chip ROM.

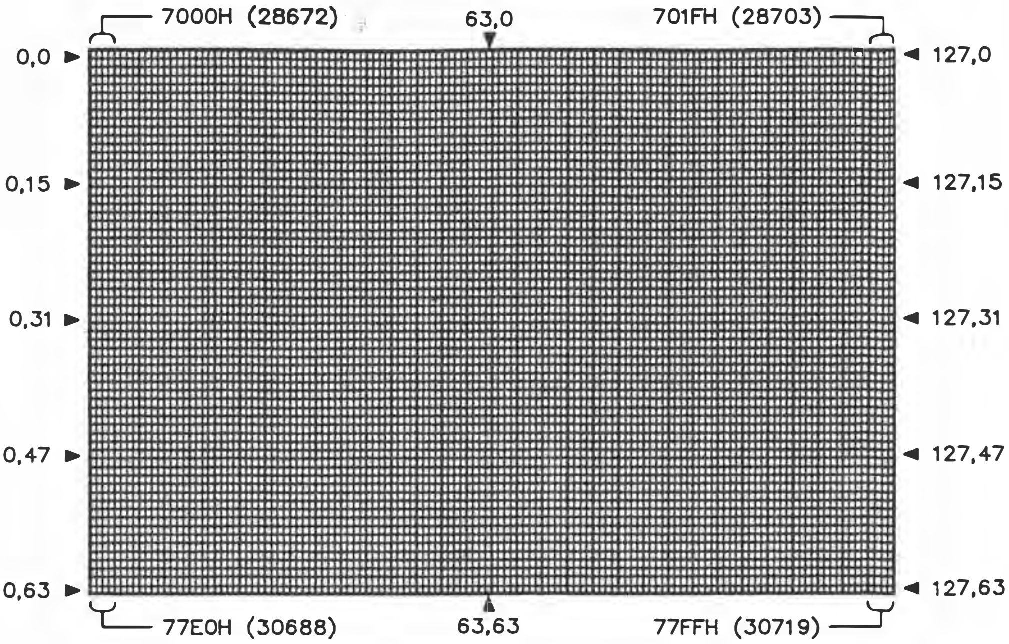


Qx	C2	C1	C0	COLOUR
0	X	X	X	BLACK
1	0	0	0	GREEN
1	0	0	1	YELLOW
1	0	1	0	BLUE
1	0	1	1	RED
1	1	0	0	BUFF
1	1	0	1	CYAN
1	1	1	0	MAGENTA
1	1	1	1	ORANGE

Figure 7. Format of Semigraphic 4 Mode on MC6847. Each character is 12 by 8 pixels but elements or quadrants can be individually illuminated giving a screen resolution of 64 by 32 elements in up to eight colours.

2 of 4.

Figure 8. Screen Addressing for MODE (1) or hi-res displays on the VZ computers. This mode corresponds to Colour Graphics 2 on the VDG and is 128 by 64 elements in size. Each element is mapped with two bits.



In summary, for Semigraphics mode it can be seen that each of the four least significant bits controls one of the quadrants, whilst the next three bits determine the colour of the illumination. The most significant bit is set high to indicate a graphics block is encoded.

b7	b6	b5	b4	b3	b2	b1	b0
graphic *A/S	colour			G3	G2	G1	G0

In this mode, although the screen is formatted into 32 by 16 graphics blocks, in fact the quadrant resolution is actually 64 by 32 and with all of the eight colours available. This may be thought of as an intermediate resolution display mode.

Thus it can be seen that Alphanumerics in either Normal or Inverse style and Semigraphics blocks of up to eight colours can be individually set on the lo-res screen by byte mapping. Different forms of encoding the necessary information are used in each case. These features combine to make MODE(0) quite a powerful display despite its lack of resolution.

Hi-res/Graphics/Mode(1)

In hi-res or MODE(1), the screen has 128 by 64 elements individually addressable. This corresponds to 8192 elements and with only 2K of video RAM available, then some sort of trade-off in features over lo-res must ensue. In hi-res, each element is 2 by 3 pixels in size and is (noticeably) rectangular in shape. Video RAM addressing extends from 7000H (28672D) to 71FFH (30719D) — 2048 bytes as shown in Figure 8.

This mode corresponds to Colour Graphics Two (CG2) on the VDG chip. Each byte addresses four consecutive elements across the screen. Each element may be one-of-four colours (selected from either of the two colour sets). Note the trade-off in colours and the different way in which elements are addressed on the screen — such that MODE(0) and MODE(1) screens cannot be mixed.

There are a couple of ways in which each element may be illuminated.

The simplest (and slowest) way is by using the BASIC commands of SET and RESET. These commands alter two bits of the appropriate byte in the video RAM area. The processing is very slow because of this limitation and the fact that it is done through the BASIC interpreter. Listing 1 provides a simple illustration of this method. The program fills the entire screen with hi-res elements according to the COLOR command. The use of integer index variables speeds up the program a little.

LISTING 1

```
10 '***SNAIL GRAPHICS DEMO***
20 '***       HI-RES       ***
30 '***    VERSION 1.2     ***
40 '***  R.B.K.  22/5/86   ***
50 '*** EXECUTION TIME 43.7 SECS.
100 'SET TO HI-RES
120 MODE(1)
130 COLOR 3,0
140 SOUND 10,1
200 FOR V%=0 TO 63
210 FOR H%=0 TO 127
220 SET(H%,V%)
230 NEXT H%
240 NEXT V%
250 SOUND 10,1
260 STOP
270 END
```

LISTING 2

```
10 '***SNAIL GRAPHICS DEMO***
20 '***       HI-RES       ***
30 '***    VERSION 2.3     ***
40 '***  R.B.K.  22/5/86   ***
50 '*** EXECUTION TIME 8.3 SECS.
100 'SET TO HI-RES
120 MODE(1)
130 COLOR ,0
140 V%=170:SOUND 10,1
200 FOR I%=28672 TO 30719
210 POKE I%,V%
220 NEXT I%
250 SOUND 10,1
260 STOP
270 END
```

LISTING 3

```
10 '***NEAR-LIGHT-SPEED GRAPHICS DEMO***
20 '***            HI-RES             ***
30 '***          VERSION 1.2          ***
40 '***         R.B.K.  22/5/86       ***
50 '*** EXECUTION TIME 0.5 SECS.
100 '***LOAD BLOCK MOVE MACHINE CODE.***
110 FOR I%=-28687 TO -28674
120 READ A%: POKE I%,A%
130 NEXT
140 DATA 33,0,112,17,1,112,1,255,7,54,170,237,176,201
200 '***INITIALIZE USR() TO ADDRESS 8FF1H OR -28687D.***
210 POKE 30862,241: POKE 30863,143
300 '***SET TO HI-RES.***
310 MODE(1)
320 COLOR ,0
330 SOUND 10,1
340 X=USR(0)
350 SOUND 10,1: SOUND 0,9
360 COLOR ,1
370 SOUND 10,1: SOUND 0,9
380 STOP
390 END
```

3 of 4

LISTING 4

```
10 '***************************
20 '***  2000 VZ SCREENS  ***
30 '***    VERSION 1.2    ***
40 '***   R.B.K. 18/5/86  ***
50 '***************************
60 '
100 '***FIND TOP OF MEMORY.
110 M1=PEEK(30898):L1=PEEK(30897):'***PRESERVE TOM POINTERS.
120 TM=M1*256+L1-20                 :'***RESERVE TOP 20 BYTES.
130 MS=INT(TM/256):LS=TM-MS*256
140 POKE 30898,MS:POKE 30897,LS
150 '
200 '***SET UP LOADING OF USR() ROUTINE.
210 TM=TM+1                         :'***NEXT ADDR IN RESERVED MEM.
220 MS=INT(TM/256):LS=TM-MS*256
230 POKE 30863,MS:POKE 30862,LS
240 AD=TM+10                        :'***ADDR. FOR CHARACTER BYTE.
250 IF TM>32767 THEN TM=TM-65536 :'***CONVERT TO SIGNED INTEGER.
260 IF AD>32767 THEN AD=AD-65536
270 '
300 '***LOAD MACHINE CODE.
310 FOR ID=TM TO TM+13
320   READ VL:POKE ID,VL
330 NEXT
340 '
400 '***Z-80 BLOCK MOVE SUBROUTINE.
410 DATA 33,0,112       :'LD HL,7000H  (#28672D START VIDEO RAM)
420 DATA 17,1,112       :'LD DE,7001H  (#28673D NEXT OR DEST.)
430 DATA 1,255,7        :'LD BC,07FFH  (#2047D SIZE OF VIDEO RAM)
440 DATA 54,85          :'LD (HL),55H  (#85D YELLOW OR CHAR."U")
450 DATA 237,176        :'LDIR         (BLOCK MOVE INSTRUCTION)
460 DATA 201            :'RET
470 '
500 '***INITIALIZE DELAYS - CONTROL SPEED OF EXECUTION BY D.
510 T=0        :'***TONE  0 IS REST.       RANGE IS 0 TO 31
520 D=4        :'***DURATION  9 IS LONG.   RANGE IS 1 TO 9
530 P=30744    :'***ADDR. FOR INVERSE CONTROL.
540 POKE P,0   :'***SET UP SCREEN.
550 '
600 '***SET UP DEMO LOOP.
610 FOR ID=0 TO 255
620   POKE AD,ID                    :'***SET CHARACTER BYTE.
630   '***SCREEN MESSAGE.
640   MODE(0)                       :'***SET *A/G LO.
650   POKE P,0                      :'***SET INV LO.
660   PRINT@234," CHAR = ";ID:SOUND T,D
670   '***LO-RES SCREENS.
680   '***LO-RES GREEN CHARACTER ON BLACK BACKGROUND.
690   X=USR(0):COLOR,0:SOUND T,D:'***SET CSS LO.
700   '***LO-RES ORANGE CHARACTER ON BLACK BACKGROUND.
710   COLOR,1:SOUND T,D             :'***SET CSS HI.
720   POKE P,1                      :'***SET INV HI.
730   '***LO-RES BLACK CHARACTER ON GREEN BACKGROUND.
740   X=USR(0):COLOR,0:SOUND T,D:'***SET CSS LO.
750   '***LO-RES BLACK CHARACTER ON ORANGE BACKGROUND.
760   COLOR,1:SOUND T,D             :'***SET CSS HI.
770   '***HI-RES SCREENS.
780   MODE(1)                       :'***SET *A/G HI.
790   POKE P,0                      :'***SET INV LO.
800   '***HI-RES COLOR SET 0 - GREEN SURROUND.
810   X=USR(0):COLOR,0:SOUND T,D:'***SET CSS LO.
820   '***HI-RES COLOR SET 1 - BUFF SURROUND.
830   COLOR,1:SOUND T,D             :'***SET CSS HI.
840   POKE P,1                      :'***SET INV HI.
850   '***HI-RES COLOR SET 0.
860   X=USR(0):COLOR,0:SOUND T,D:'***SET CSS LO.
870   '***HI-RES COLOR SET 1.
880   COLOR,1:SOUND T,D             :'***SET CSS HI.
890   '***RESET CONTROLS.
900   POKE P,0:COLOR,0:CLS
910 NEXT
920 '
930 '***RESET TOM POINTERS.
940 POKE 30898,M1:POKE 30897,L1
950 STOP:END
```

A quicker way is to POKE values into each byte, thereby setting four elements at a time. Listing 2 demonstrates this technique. This program also fills the entire hi-res screen with elements whose colours are determined by the variables V%.

The quickest way is to use a machine language program to load appropriate values into the video RAM. This technique is a very rapid way to fill the screen. Listing 3 is an example of this method. This program POKEs machine code into hi-memory. The subroutine uses the very efficient Z80 Block Move command to fill the screen according to the value stored at address -28677D. It is fast!

Both of the last two methods require that an understanding of the value to enter into RAM is known. This requires a knowledge of how each byte is organised in CG2 mode.

As mentioned previously, each byte controls four elements which can be selected from four colours. Bits are treated in pairs (dibits!) with each pair corresponding to an element. Each dibit can have a value of 00B to 11B to indicate colour. This is set out on Table 3.

Four example, suppose we want an entirely BLUE screen. Then POKE (128 + 32 + 8 + 2) or 170D into the appropriate area of the screen. If, however, a striped screen consisting of RED-GREEN-BLUE-YELLOW vertical bands is required, then POKE (192 + 0 + 8 + 1) or 201D.

TABLE 3:
CONFIGURATION OF BYTES IN MODE (1).

3	2	1	0	Element #

0 0
0
	0 0
	0
		0 0
		0
			0 0
			0
				Bin. = four GREEN/BUFF elements.
				Dec. = 0D

0 1
64
	0 1
	16
		0 1
		4
			0 1
			1
				Bin. = four YELLOW/CYAN elements.
				Dec. = 85D

1 0
128
	1 0
	32
		1 0
		8
			1 0
			2
				Bin. = four BLUE/MAGENTA elements.
				Dec. = 170D

1 1
192
	1 1
	48
		1 1
		12
			1 1
			3
				Bin. = four RED/ORANGE elements.
				Dec. = 255D

The decimal numbers corresponding to each element position AND colour provide the value that needs to be POKE'd or loaded.

Although only four colours are available, there are two colour sets available. These are called by the COLOR command.

COLOR, 0 sets the background colour to green and the 'strong' colours of yellow, blue and red are available.

COLOR, 1 sets the background to buff and the 'pastelle' colours of cyan, magenta and orange are available.

To think back to the RESET command mentioned before, it should be apparent that this command simply resets each dibit or element back to 00B, or the background colour.

Finale

Well there we have it! For those who have perservered thus far I have included Listing 4 which is entitled '2000 VZ Screens'. It is about as exciting as watching a Late Night Movie — and takes about as long to run! Actually it illustrates all of the features discussed in this article. For those who wish to sit-it-out — watch those control lines operate!

AEM Oct. 86 p. 121.

4 of 4.

Addendum.

Add the following lines to listing 4.

145 CLEAR 50

945 CLEAR 50

REFERENCE LISTING OF VZ-200/300 MAGAZINE ARTICLES

Since its introduction in early 1983, over one hundred articles on the VZ-200 and 300 have appeared in magazines. some articles review the hardware and others describe peripherals, some excellent games have been published and a very useful set of utility routines has emerged.

This bibliography for the VZ computer is a must for the serious VZ User.

UTILITIES

Oct.	83	APC	52, 4	BASIC program conversion. (Surya)	(2)
Nov.	83	APC	57, 9	Program conversion Pt. 2 (Surya)	(2)
Nov.	83	APC	89-95	BASIC converter chart. (Surya)	(7)
Feb.	84	APC	140-1	Program conversion Pt. 2 (Surya)	(2)
Mar.	84	APC	42-3	Program conversion — Apple II (Surya)	(2)
Apr.	84	APC	71-2	Program conversion — TRS 80/System 80 (Surya)	(1)
May	84	APC	75-6	Program conversion — Atari (Surya)	(2)
Jun.	84	APC	67	Program conversion — Sinclair (Surya)	(1)
Jul.	84	APC	129-30	Program conversion — BBC (Surya)	(2)
Mar.	84	ETI	63	More functions for the VZ-200. (Olney)	(1)
Apr.	85	ETI	117	Notes and errata for Olney.	(–)
Jul.	84	M80	3-4	VZED — three new functions.	(1)
Aug.	84	M80	2	VZ-200 output latch.	(1)
Aug.	84	M80	9, 15, 16	Memory peek VZED. (Carson)	(1)
Aug.	84	M80	3-4	Microsoft ROM BASIC Level I bug.	(1)
Apr.	85	APC	97	VZ-200 bug. (Tritscher)	(–)
Aug.	85	APC	31	VZ bug. (Tritscher)	(–)
Aug.	84	APC	94	VZ-200 moving message and trace. (Batterson)	(1)
Nov.	84	APC	125	Trace function. (Breffit)	(–)
Nov.	84	APC	125	VZ-200 correction. (Kelly)	(–)
Oct.	84	ETI	135-7	Extending VZ-200 BASIC. (Olney)	(3)
Nov.	84	APC	125-6	TRON/TROFF function for VZ-200. (Thompson)	(1)
Nov.	84	APC	208-12	MON-200 machine code monitor. (Stamboulidas)	(5)
Nov.	84	PCG	55-56	Lprinter. (Quinn)	(2)

[Cont. on 2 of 2.]

— from page 114

AEM	Australian Electronics Monthly
APC	Australian Personal Computer
BYC	Bumper Book of Programs by YC
CC	Creative Computing (US)
CFG	Computer Fun and Games
CT	Computing Today (UK)
CHC	Choice
EA	Electronics Australia
ETI	Electronics Today International
M80	Micro-80
MC	Micro Choice (UK)
PCG	Personal Computer Games
PCN	Personal Computer News (UK)
PE	Practical Electronics (UK)
WM	Which Micro (UK)
YC	Your Computer

The numbers in brackets are the number of sheets in each article. A dash (–) indicates that the article is on the same sheet as the item above.

If Users wish to obtain copies of the articles referred to in this bibliography they may —

i) contact me for copies . . . or . . .
ii) buy back copies of the magazine from the distributor . . . or . . .
iii) borrow from your local library.

Compiled by —
Bob Kitch, 7 Eurella St., KENMORE, QLD 4069. Ph. (07) 378 3745

PLEASE ADVISE OF ANY ADDITIONAL ARTICLES . . . or . . . CHANGES, ALTERATIONS OR BUGS IN LISTINGS to assist other users.

Nov.	84	PCG	suppl.	VZ-200 reverse video.	(1)
Feb.	85	APC	171	BASIC understanding (Hobson)	(1)
Feb.	85	APC	20	VZ-200 into puberty — Olney's extended BASIC.	(1)
Apr.	85	PCG	62-64	Find. (Stamboulidas)	(3)
Apr.	85	APC	19	Use of RND in dice and card games. (Holland)	(1)
Apr.	85	APC	103	VZ variable definition. (Stamboulidas)	(1)
Apr.	85	APC	95	Variable GO TO on VZ. (Olsen)	(1)
Jul.	85	APC	176	Correction to VZ variable GO TO	(-)
May	85	APC	52-3	Lusco support for VZ-200. (Young)	(1)
May	85	ETI	99-101	VZ-200 hardware interrupt. (Olney)	(3)
May	85	APC	110	Background VZ. (Williams)	(1)
Aug.	85	APC	130	VZ-200 instant colour. (Willows)	(-)
Aug.	85	APC	130-3	Reversed REM. (Quinn)	(1)
Sep.	85	APC	145	Real-time clock. (Griffin)	(1)
Oct.	85	APC	218	APC benchmark BASIC programs.	(1)
Oct.	85	APC	147	VZ deletions. (Quinn)	(1)
Nov.	85	APC	189	VZ EDITOR/ASSEMBLER tips. (Lam)	(1)
Nov.	85	ETI	94-5	Olney's Level II BASIC for VZ-300/300. (Rowe)	(2)
Jan.	86	APC	83, 5	VZ user graphics.	(1)
Feb.	86	APC	127	Machine language calls.	(1)
Mar.	86	APC	chart	APC BASIC converter chart 1986.	(8)
Mar.	86	YC	103-5	VZ-200 cassette inlays. (Dutfield)	(3)
Jun.	86	APC	209	VZ pause.	(1)

GAMES

Dec.	83	APC	161-3	Missile Command. (Whitwell)	(2)
Jan.	84	YC	65	Graphic Sine Waves for VZ-200. (Nickasen)	(1)
Apr.	84	APC	178-80	Moon Lander. (Alley)	(2)
Jul.	84	APC	174-8	Blockout. (Pritchard)	(3)
Jul.	84	M80	7, 22	Battleships. (Carson)	(1)
Jul.	84	M80	7, 20, 21	Junior Maths. (Carson)	(2)
Aug.	84	M80	9, 16	Contest Log VZED. (Carson)	(1)
Aug.	84	M80	9, 16, 17	Dog Race VZED. (Carson)	(1)
Oct.	84	PCG	55-7	High Resolution Graphics Plotting. (Thomson)	(3)
Nov.	84	PCG	82	Tips for 'Ladder Challenge', 'Panik' and 'Asteroids'.	(1)
Jan.	85	PCG	54	POKEs to 'Ghost Hunter'.	(-)
—	85	BYC	146-7	Gold Simulation. (McCleary)	(2)
Mar.	86	CFG	4-5	Gold Simulation. (McCleary)	(-)
—	85	BYC	147	Knight's Cross. (Lucas)	(1)
Jan.	85	APC	129-31	Sketcher. (Leon)	(3)
Jan.	85	YC	88-89	Punch. (Rowe)	(2)
Jan.	85	PCG	44-48	Space Station Defender. (Shultz)	(5)
Mar.	85	YC	105-9	Decoy. (Rowe)	(2)
Apr.	85	YC	160	Painter. (Daniel)	(1)
Apr.	85	PCG	65-7	Roadrace. (Thompson)	(3)
May	85	YC	106	Number Sequence. (Thompson)	(1)
May/Jun	85	PCG	63-7	Sketchpad. (Thompson)	(5)
Jun.	85	YC	70	Morse Tutor program. (Heath)	(1)
Jan.	86	YC	150-1	Morse Tutor — again. (Heath)	(2)
Jul.	85	YC	81	Electric Tunnel. (Daniel)	(1)
Aug.	85	YC	114	Number Slide. (Daniel)	(1)
Oct.	85	PCG	47-52	Cube. (McMullan)	(6)
Oct.	85	YC	105-7	Yahtzee. (Thompson)	(3)
Mar.	86	APC	208-9	VZ Frog. (Alley)	(1)
May	86	ETI	93	Balloon Safari, The Drop and Flatten. (Sheppard)	(1)

BUSINESS

Aug.	84	APC	172-7	Database VZ-200. (Barker)	(6)
Oct.	84	APC	214	WP for VZ-200. (McQuillan)	(-)
Oct.	85	APC	82-3	Comment on Barker's and Quinn's DB. (Lukes)	(-)
Oct.	84	APC	126-30	Minicalc Spreadsheet. (Stamboulidas)	(5)
Dec.	84	APC	214	Correction to Minicalc.	(1)
May	85	APC	162-3	Micro Type (WP). (Browell)	(2)
Jul.	85	APC	164-6	Database. (Quinn)	(2)

PERIPHERALS

Feb.	84	EA	131-2	Real-world interface.	(1)
Aug.	84	EA	65	Improved graphics on VZ-200. (Dimond)	(1)
Aug.	84	PCG	83	I/O card for VZ-200. (ad)	(1)
Oct.	84	APC	214	Serial help request. (Pope)	(1)
Dec.	84	APC	36	Add-ons for VZ-200. (Bleckendorf)	(-)
Oct.	85	YC	140	VZ-200/300 Modem. (ad)	(-)
Nov.	84	ETI	106-12	A 'Glass-Teletype' using the VZ-200 Pt I	(7)
Dec.	84	ETI	93-7	A 'Glass-Teletype' using the VZ-200 Pt II	(5)
Aug.	85	ETI	72-8	VZ-200 terminal.	(7)
Jun.	86	EA	106	VZ serial terminal. (ad DSE kit K6317)	(-)

Feb.	86	ETI	72-4	Modifying VZ-200 16K memory expansion. (Olney)	(3)
Mar.	86	ETI	48	Talking VZ-200. (Bennets)	(1)
Sep.	86	AEM	89-92	VZ-200/300 Screen-handling. (Kitch)	(4)

COMMERCIAL SOFTWARE REVIEWS

Mar.	84	APC	190-1	Review of DSE 'Matchbox', 'Biorythms', 'Circuit', and 'Poker'. (Davies)	(2)
Aug.	84	pcg	46-47	Review of DSE 'Panik' and 'Ladder Challenge'.	(1)
Oct.	84	PCG	90-91	Review of DSE 'knights and Dragons', 'Ghost Hunter', 'Othello', and 'Invaders'.	(2)
Nov.	84	PCG	90-96	Review of LYSCO 'Cub Scout' and DSE 'Dracula's Castle'.	(1)
Jan.	85	PCG	65	Review of DSE 'Air Traffic Controller' and 'Tennis'.	(1)
Feb.	85	PCG	76	Review of DSE 'Defence Penetrator' and 'Star Blaster'.	(1)
Mar.	85	PCG	76-77	Review of DSE 'Planet Patrol' and 'Learjet'.	(1)
Apr.	85	PCG	94-99	Review of DSE 'Asteroids', 'Super Snake' and 'Lunar Lander'.	(1)
Apr.	85	ETI	103	Logbook and Morse on VZ-200.	(1)
Oct.	85	PCG	68-9	Review of DSE 'Duel'.	(1)
Nov.	85	PCG	70-1	Review of DSE 'Attack of the Killer Tomatoes'.	(1)

HARDWARE REVIEWS

Apr.	83	APC	58-66	VZ-200. (Hartnell)	(5)
Apr.	83	CC	38-43	Review of VZ-200	(3)
May	83	CC	26-30	Video Technology VZ-200 PC. (Ah1)	(3)
Jun.	83	EA	137	New low-cost computer — VZ-200.	(1)
Jun.	83	ETI	30	Dick Smith colour computer.	(1)
Jun.	83	YC	6	DSE VZ-200.	(-)
Aug.	84	PCG	12	VZ-200.	(-)
Jul.	83	ETI	32-7	DSE's personal colour computer. (Harrison)	(3)
Jul.	83	EA	130-3	The VZ-200: colour, graphics and sound. (Vernon)	(4)
Jul.	83	PCN	16	Timing the Laser's phazer. (Stokes)	(1)
Sep.	83	WM	40	Laser.	(-)
Aug.	83	YC	20-33	Cash and Carry Computers. (Bell)	(9)
Sep.	83	CC	202-4	Review of VZ-200 and PP40	(1)
Oct.	83	APC	77-8	VZ-200.	(1)
Oct.	83	WM	135	Texet TX8000.	(1)
Oct.	83	CT	12	The Laser 200.	(-)
Dec.	83	CT	11	Laser 200.	(-)
Nov.	83	CT	37-40	A look at the Laser. (Green)	(4)
Nov.	83	WM	42-108	The Laser — a shot in the dark.	(3)
Feb.	84	CC	218-21	Laser PP40 Printer/Plotter.	(2)
Spring	84	MC	52-4	Laser 200. (Green)	(3)
Jun.	84	EA	12-9	Buying your first computer. (Vernon)	(6)
Aug.	84	EA	30-3	An important role for small computers. (Williams)	(4)
Oct.	84	PCG	82-87	Home micro supertest. Pt. 3 (Bollington)	(5)
Nov.	84	PCG	14-19	Home micro supertest. Pt. 4 (Bollington)	(4)
Nov.	84	EA	78-80	VZ-200 as a WP (DSE E&F tape WP). (Williams)	(2)
Dec.	84	CHC	28-31	Review of video games consoles.	(4)
Jul.	85	ETI	102-6	Dick Smith's new VZ-300. (Rowe)	(5)
Aug.	85	EA	22-7	WP on the new VZ-300. (Williams)	(5)
Dec/Jan	86	PCG	11-15	How to buy a micro — VZ-300 compared.	(4)

GENERAL PROGRAMMING

Jan.	83	PE	3/1-3/5	PE Micro-file #3 — Z80. (Coles	(5)
Mar.	84	APC	73-85	Teach yourself assembler Pt. 1 (Overaa)	(6)
Apr.	84	APC	57-64	(8080, Z80, 6502) Pt. 2 (Overaa)	(5)
May	84	APC	89-98	(8080, Z80, 6502) Pt. 3 (Overaa)	(5)
Jun.	84	APC	53-60	(8080, Z80, 6502) Pt. 4 (Overaa)	(5)
Jul.	84	APC	61-64	(8080, Z80, 6502) Pt. 5 (Overaa)	(3)
Aug.	84	APC	110-116	(8080, Z80, 6502) Pt. 6 (Overaa)	(5)
Sep.	84	APC	145-151	(8080, Z80, 6502) Pt. 7 (Overaa)	(4)
Jan.	85	APC	122-124	Sort at input. (Ithell)	(1)
Feb.	85	APC	103-109	The basic art — algorithms, structures (Liardet)	(4)
Mar.	85	APC	98-109	Pick a number — arithmetic. (Liardet)	(5)
Apr.	85	APC	79-87	It takes all sorts — sorting. (Liardet)	(5)
Oct.	85	APC	82	The Art of Programming — Progress. (Hjaltson)	(-)
Jun.	85	APC	170-171	Comment on binary search. (Lamich)	(1)
Jun.	85	APC	171-173	Comment on disbribution sort. (Riordon)	(1)
Oct.	85	YC	107-8	Sorting out the sorts. (Jankowski)	(1)

— to page 121

2 of 2.

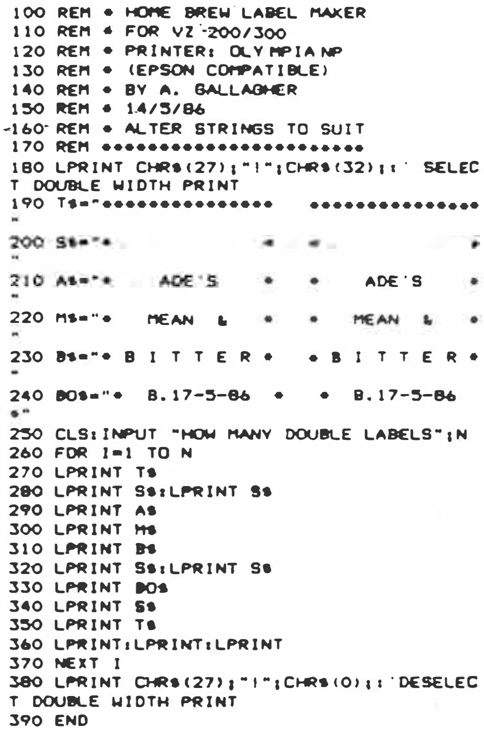
[Cont. on 1 of 2]

Home brew label maker

A program for programmers who like beer. By altering the strings in lines 190-240, the program can be customised for any user (and, indeed, for other labels besides home brew). Once you have set up your label, you need to remember to change the string BO$ in line 240 to correspond with your date of bottling. Make sure that all of the strings have the same length to ensure a neat label.

The program should be easy to translate for other computers and printers. Line 180 activates double width print on my Olympia printer; line 380 deactivates it.

Adrian Gallagher
Bendigo, Vic



```
100 REM • HOME BREW LABEL MAKER
110 REM • FOR VZ-200/300
120 REM • PRINTER: OLYMPIA NP
130 REM • (EPSON COMPATIBLE)
140 REM • BY A. GALLAGHER
150 REM • 14/5/86
160 REM • ALTER STRINGS TO SUIT
170 REM ••••••••••••••••••••••••
180 LPRINT CHR$(27);"!";CHR$(32);: ' SELEC
T DOUBLE WIDTH PRINT
190 T$="••••••••••••••••   ••••••••••••••••
"
200 S$="•              •  •              •
"
210 A$="•    ADE'S     •  •    ADE'S     •
"
220 M$="•    MEAN &    •  •    MEAN &    •
"
230 B$="• B I T T E R •   • B I T T E R •
"
240 BO$="•  B.17-5-86  •  •  B.17-5-86
•"
250 CLS:INPUT "HOW MANY DOUBLE LABELS";N
260 FOR I=1 TO N
270 LPRINT T$
280 LPRINT S$:LPRINT S$
290 LPRINT A$
300 LPRINT M$
310 LPRINT B$
320 LPRINT S$:LPRINT S$
330 LPRINT BO$
340 LPRINT S$
350 LPRINT T$
360 LPRINT:LPRINT:LPRINT
370 NEXT I
380 LPRINT CHR$(27);"!";CHR$(0);: 'DESELEC
T DOUBLE WIDTH PRINT
390 END
```

A COMPUTER LOGGER FOR THE VZ-200/300

By Alex Johnson Jr.

Many hams have purchased the VZ-200 computer marketed by Dick Smith Electronics and the more recent VZ-300 model. Some have also taken advantage of the various projects and kits that allow the computer to be utilised for RTTY and CW.

Being the son of an amateur, I couldn't help but wonder, "why leave it there?". I also couldn't help but notice the time and trouble involved in keeping a log. Every time a contact is made and the callsign rang a bell, valuable time was lost flipping through log pages to track down who, where and when.

Problems were also observed during contests when, with each contact you make, you have to either mentally or physically flip through the log to see if the station has been worked before and, in the case of some contests, to see if the required time between duplicate contacts has elapsed.

So, if you have a computer handy in the shack, why not also use it to relieve the everyday drudgery of log keeping?

The program listed here is short (as log programs go) and written in BASIC so even the most cautious user can type it in without much trouble. It requires 24K of memory, an 80-column printer and a cassette recorder. The program is written specifically for the Dick Smith GP-100 dot matrix printer, but should work with most printers without any worries.

The program includes many "secrets, tricks and short-cuts" that I have discovered after working with the VZ for some time. These are used throughout the program to save memory, so please type the program in exactly as indicated in the listing (although you can leave out the spaces outside PRINT/INPUT statements) as the memory is balanced and juggled between string and memory needs.

To save you some counting, long stretches of spaces inside PRINT statements have been printed as (x spaces). When you encounter this, just type the number of spaces indicated by 'x'. When you encounter the term (rev), this indicates reversed text as used in program listing on the VZ-200/300. The printer used to list the program does not reproduce reversed text very well.

I have personally checked the final printout and provided the editor with corrections and modifications needed to ensure that the program works effectively (and your editor has taken great care to correct the detected errors for a bug-free printout — ED).

TRICKS & SECRETS

Some of the memory-saving tricks used in the program include:

1. Beep each time a key is pressed . . .

```
POKE 30862,90: POKE 30863,52
X=USR(0)
```

2. Small quick beep that can't be switched off (see 'beep off') . . .

```
POKE 28761,1: POKE 28671,32
```

The VZ technical manual discusses the Peizo on page 7. Any address from 26624 to 28671 decimal (6800 to 6FFF hex) will result in a beep when POKEd with these values.

```
Beep on: POKE 30779,0
Beep off: POKE 30779,1
```

3. Memory left . . .

```
POKE 30862,212: POKE 30863,39
```

String memory . .

```
PRINT USR(X$)
```

RAM memory . . .

```
PRINT USR(X)
```

4. Sound abbreviations: the use of a semi-colon between notes and durations . . .

```
SOUND 16,2;21,7;15,3;22,8
```

1 of 5.

7 Oct. 1986. p38-42.

5. THEN, GOSUB and GOTO on IF . . THEN . . ELSE statements: THEN can be replaced by a comma; GOTO can be left out; just the line number needs to be typed in after the comma. If a GOSUB is needed, then the comma can be left off . . .

```
IF A = B,100; IF A = B, PRINT "HI"
IF A = B GOSUB 100
```

6. REM can be replaced with an inverted comma and NEXT may be used with no variable . . .

```
100 ' THIS IS FUN
200 FOR A = 1 to X: NEXT
```

There are many more "tricks" that I did not use in the program. If you have others or want to know the rest, write to me at 19 Banksia St, O'Connor, ACT, 2601.

PROGRAM DESCRIPTION

The program is broken into subroutines and components that can easily be used to help iron out any bugs. The following is a brief summary of the subroutines and components.

Lines 10-90: In Line 10 the screen background color is set and addresses 30862 and 30863 decimal (788E and 788F hex) are POKEd with the addresses of the beep routine in ROM, 13392 decimal (3450 hex). Starting entry number is entered and variables are DIMensioned and set. T% in Line 40 controls how many entries can be kept in RAM at the one time. If you have more than 28K of RAM you can increase this. See Lines 1020 to 1080.

Lines 90-176: The screen is set up. Note the number of periods or dots in these lines, as they are crucial in the log PRINTing process.

Line 180: Commands are entered. This is what is referred to as the command line or command entry point.

Lines 190-310: The input at the command line is checked for a valid input. If the input is valid, the program goes to the appropriate subroutine; otherwise it returns to Line 170.

Lines 320-340: The cursor is moved to the correct position to fill in the entries. Before the command line, a click is produced by toggling the Peizo high then low. This is done by POKEing decimal 28671 (6FFF hex), the byte before the start of screen RAM. See previous text.

Lines 350-360: The entries are checked for length and cut down to the correct size. This is necessary as the PRINTing is dictated by entry length, as can be seen from Lines 430 and 860-870. This method is used to save memory.

Lines 368-420: Previous entries are scanned through from the latest to the first — i.e. backwards. If a previous contact has been made, the most recent contact is displayed.

Lines 430-435: If the continuous PRINTing mode is on, this subroutine is used to make the hard copy. The importance of length of entries can be seen here as entries are simply PRINTed one after the other without TABs.

Lines 440-500: This is the 'FIND' subroutine where callsigns are compared with the one specified in Line 140. If a match is found, you can continue the log or execute a further search.

Lines 510-540: This is the 'DELETE' subroutine where callsigns are compared with the one specified in Line 140. If a match is found, you can continue the log or execute a further search.

Lines 550-630: The 'SORT' routine allows the user to sort in two fields, by entry or callsign. If callsign is selected, all callsigns in string RAM are compared and sorted into alphabetical order in Lines 570-630. If entry number is selected, the data in string RAM is sorted into numerical order of entry number. Both formats are completed on three common nested loops. The decision as to which field is to be sorted is made in Lines 590 and 600.

Line 640: 'FILES FULL' subroutine. This is used when the maximum allowable string RAM is reached. It is marked by an indicator in the top left corner of the screen and a series of beeps. The beeps in Line 640 are produced from the one SOUND statement with semi-colons separating each note-/duration pair. The files are considered full when the entry number is greater than T% — see Line 160. It is then necessary to SAVE the entries.

Lines 650-660: Continuous PRINTing mode is toggled on and off. When Z% = 1, it is on. When Z% = 0, it's off. See Lines 175 and 420-435.

Lines 670-720: Entries are displayed on the screen. While the entries are being displayed, pressing 'P' or the space bar will 'PAUSE' the screen, while 'S' will stop the list and return you to the main entry page.

Lines 730-870: Hard copy of the entries is made in this subroutine. The entries are PRINTed one after the other in Lines 860 and 870 and are not TABbed, as in Lines 430-435. Unlike continuous print, the pages are numbered and headed in columns. Entries are printed in the current (sorted) order — i.e. if a SORT by callsign has been carried out, the log will be printed in callsign order; if not, it will be printed in entry order. Stop and pause are similar to Lines 700-710.

Lines 880-920: Entries are SAVEd to tape in this subroutine in the current (sorted) order.

Lines 930-980: Previously sAVEd entries are LOADed from tape in this subroutine.

Lines 990-1000: The key beep is toggled on and off in this subroutine by PEEKing decimal 30779 (7836 hex), checking its value and adjusting Y%. When 30779 has a value of 0 and Y% = 1, the beep is off. If 30779 has a value of 1 and Y% = 0, the beep is on. If you are wondering why I used Y% to switch the indicator in Line 136 and didn't simply PEEK 30779, it isn't because I didn't think of it but rather because, for some reason or other, the value in 30779 is intermittently misread and therefore unreliable alone.

Lines 1020-1080: Memory left subroutine. This is useful if you have a computer with more than the basic 24K. By adjusting T% in Line 40, and keeping an eye on this subroutine, you can have more entries in string RAM at the one time. The amount of memory is calculated by calling a routine at decimal 10196 (27D4 hex). When USR(X) is used, the amount of free RAM in bytes is derived. USR(X$) derives the amount of string RAM in bytes. Line 170 is used to reset decimal 30862 and 30863 to the beep routine. See Line 10.

USING THE PROGRAM

Once the program is typed in and appears to be error free, SAVE it before you attempt to RUN it as it may contain an error that causes the program to crash and be lost. Once it is SAVEd, it is then safe to start as the SAVEd copy can be loaded and edited if the program crashes.

2 of 5.

Once you type RUN and press RETURN, the first screen will ask you to enter the starting entry number. If this is the first time you are using the program, it will be '1'. If you are going to LOAD previously saved entries, just press RETURN with no number.

There will be a slight pause as the computer works itself out, then the main entry screen will appear with a beep. The cursor will be on the bottom of the screen — commands are entered from this point and nowhere else. If you wish to complete an entry, press RETURN. The cursor will then move up to the date — type it in the DDMMYY format as indicated (1st October, 1986 would be '011086'). Remember to put a zero in front of the number if it is a single digit (i.e. 4th is 04, March is 03).

An important point to note is not to go beyond the dotted markers for each parameter. If you do, the computer will automatically remove the extra characters but your screen will end up in rather a mess. If the screen does happen to get into a mess, just use the CLEAN command to reprint it.

When you have the date typed in, press RETURN and the cursor will move down to callsign. Type this in, not forgetting to press RETURN once you have finished. The cursor will then move down to the time. As with date, time should be entered in the HHMM format — i.e. 7 PM EST would be entered as 1900 (or 1100 UTC).

Be careful not to use semi-colons or commas in any of the entries, especially in 'remarks', as this will cause an error in the computer — 'extra ignored' or 'redo' — making life just a bit confused. If you do encounter such problems, forget the entry you were typing in and press RETURN until you hear the command line click. You can now see the reason for this click that cannot be turned off. Type 'CLEAN' as before and restart the entry.

If the screen is complete and correct, type 'ZZ' and press RETURN. This fills in the entry.

SPECIAL COMMANDS

Once you have a few entries in the log, you can have some real fun. Remember **commands can only be entered on the bottom line of the screen — the command line,** and it is only necessary to enter the first two letters of each command.

FIND is used to look through the log entries for a specified callsign. When the callsign is located, it will be displayed by filling in the details on the entry screen. You can then continue the search for further contacts by pressing 'F', or resume log entries by pressing 'C'.

DELETE removes an entry, placing a void on the callsign and removing all other information. You are asked for the entry number, so if you're unsure of the number, use DISPLAY or FIND to locate it.

SORT rearranges the entries in alphabetical order (press 'C' for callsign) or in entry order (press 'E' for entry). This is a BASIC program, so sorting does take a long time. Make yourself a coffee and have a break while it sorts.

RESTART simply starts the program over again. All entries are removed from the memories and all variables are reset, so SAVE your log before using RESTART.

DISPLAY prints all entries onto the screen. if you wish to pause while the entries are listing, press 'P' or the space bar. To stop the list completely, press 'S'. Once printing is complete, press 'C' to continue log entries.

CP or Continuous Print is used to keep a running hard copy of all entries as they are made. CP pages are not headed or numbered. When CP is activated, it is indicated on the command line. CP is deactivated by retyping CP on the command line.

PRINT makes a hard copy of the entire log on the printer. You must first enter the page length and the inter-page length. The page length must be more than six lines. Each page is numbered and headed and also has the date displayed on it. Stop and Pause commands are the same as for DISPLAY.

SAVE is used to send log details from RAM to tape. The SAVEd entries all have the same file number, so take note of the counter number on the datasette each time you SAVE.

LOAD is used to retrieve previously SAVEd files from tape.

MEMORY displays the amount of RAM, string memory and file space remaining.

BEEP turns the key beep on and off. When BEEP is activated it is indicated on the command line. To deactivate it, type BE again.

CLEAN is used to clear and reprint the screen if it becomes messy through an error.

ZZ enters the on-screen details in the log file.

I have two versions of the program: the one listed here (tape version) and another for VZs with a disk drive. The disk version is, I must admit, far superior in speed, capacity and commands. If difficulties are met typing this program in, and you would like a tape or disk copy of the working program, please drop me a line at the address mentioned earlier, with a stamped self-addressed envelope, and I will offer some suggestions.

PROGRAM LISTING

```
0010 POKE 30744,1: POKE 30862,80: POKE 30863,52: CLEAR
     10000:CLS
0015 PRINT "SERIAL: TPE2.310586": PRINT
0020 PRINT "COMPUTER LOG BOOK": PRINT "BY ALEX JOHNSON"
0030 PRINT "(C) COPYRIGHT 1986": PRINT@386,"STARTING ENTRY
     [space]NUMBER";
0040 T%=99: INPUT S%: IF S%)9999 OR S%(0, 10 ELSE CLS
0050 DIM I$(10,T%), J$(10), K$(10): D$="000000": N%=-1
0060 Z$="[32 spaces]"
0065 Y$="......................"
0070 IF C%=1, C%=0: CLS: GOTO 80 ELSE N%=N%+1
0080 N$=MID$(STR$(N%+S%),2,LEN(STR$(N%+S%)))
0090 IF LEN(N$)(4, N$="0"+N$: GOTO 90
0100 X=USR(0): PRINT@0,Z$;Z$;Z$;
0110 PRINT@96,"ENTRY   ? ";N$: PRINT "DDMMYY  ? ";D$
0120 PRINT "CALLSIGN? VK....": PRINT "HHMM    ? ...."
0130 PRINT "RECD R/S ..": PRINT "SENT R/S .."
0140 PRINT "FREQ MHZ? .......": PRINT "MODE    ? ..."
0150 PRINT "QTH     ? ..........": PRINT NAME    ? .......
     ..."
0160 PRINT "REMARKS ? ................": IF N%)T%,GOSUB 640
0170 PRINT@448, "[8 spaces]?[22 spaces]";
```

3 of 5.

```
0175 IF Z%=1, PRINT@448, "[rev]CP"
0176 IF Y%=1, PRINT@451, "[rev]BE"
0180 PRINT@456, "";: INPUT A$: IF A$="", 320 ELSE A$=LEFT$
     (A$,2)
0190 IF A$="FI", 440
0200 IF A$="DE", 510
0210 IF A$="SO", 550
0220 IF A$="RE", RUN
0230 IF A$="DI", 670
0240 IF A$="PR", 730
0250 IF A$="CP", 650
0260 IF A$="SA", 880
0270 IF A$="LO", 930
0280 IF A$="ME", 1020
0290 IF A$="BE", 990
0300 IF A$="ZZ", 350
0305 IF A$="CL", C%=1: GOTO 70
0310 GOTO 170
0320 FOR A%=1 TO 10: PRINT@(A%*32)+104, "";: IF A%=1, INPUT
     D$: GOTO 340
0330 INPUT  I1$(A%,N%)
0340 NEXT: POKE 28671,1: POKE 28671,32: GOTO 170
0350 PRINT@0,"[rev]CHECKING": I1$(0,N%)=N$: I1$(1,N%)=D$
0351 I1$(2,N%)=I1$(2,N%)+Y$: I1$(3,N%)=I1$(3,N%)+Y$
0352 I1$(4,N%)=I1$(4,N%)+Y$: I1$(5,N%)=I1$(5,N%)+Y$
0353 I1$(6,N%)=I1$(6,N%)+Y$: I1$(7,N%)=I1$(7,N%)+Y$
0354 I1$(8,N%)=I1$(8,N%)+Y$: I1$(9,N%)=I1$(9,N%)+Y$
0355 I1$(10,N%)=I1$(10,N%)+Y$
0356 I1$(2,N%)=LEFT$(I1$(2,N%),6): I1$(3,N%)=LEFT$
     (I1$(3,N%),4)
0358 I1$(4,N%)=LEFT$(I1$(4,N%),2): I1$(5,N%)=LEFT$
     (I1$(5,N%),2)
0360 I1$(6,N%)=LEFT$(I1$(6,N%),7): I1$(7,N%)=LEFT$
     (I1$(7,N%),3)
0362 I1$(8,N%)=LEFT$(I1$(8,N%),10): I1$(9,N%)=LEFT$
     (I1$(9,N%),10)
0364 I1$(10,N%)=LEFT$(I1$(10,N%),16)
0366 IF N%=0, 420
0368 FOR A%=N%-1 TO 0 STEP -1: IF I1$(2,N%)=I1$(2,A%), 380
     ELSE NEXT
0370 GOTO 420
0380 PRINT@0,I1$(0,A%);"   [rev]FOUND": PRINT I1$(2,A%);
     "[space]"; I1$(1,A%);
0390 PRINT "[space]"; I1$(3,A%); "[space]"; I1$(9,A%)
0400 PRINT@13, "[rev]P)RINT D)ELETE"
0410 A$=INKEY$: IF A$="P", 420 ELSE IF A$="D", 100 ELSE 410
0420 IF Z%=1, 430 ELSE 70
0430 FOR A%=0 TO 10: LPRINT I1$(A%,N%);: IF A%<10, LPRINT
     "[space]";
0435 NEXT: LPRINT: GOTO 70
0440 PRINT@0, "CALLSIGN TO FIND ? VK....": PRINT@17,"";:
     INPUT B$
```

```
0450 PRINT@0, "[rev]SEARCHING"; Z$: FOR A%=0 TO N%-1
0460 IF I1$(2,A%)=B$, 470 ELSE NEXT: GOTO 100
0470 PRINT@0, "[rev]FOUND";Z$: FOR B%=0 TO 10: PRINT@(B%
     *32)+106, I1$(B%,A%)
0480 NEXT: SOUND 0,2:  PRINT@6, "[rev]F)URTHER C)ONTINUE"
0490 C$=INKEY$: A$=INKEY$: IF A$="F", 500 ELSE IF A$="C",
     100 ELSE 490

0500 X=USR(0): PRINT@0, "[rev]SEARCHING";Z$: NEXT: GOTO 100
0510 PRINT@0, "";: INPUT "ENTRY TO DELETE ";A%: A%=A%-S%
0520 IF A%<0 OR A%>N%-1, 100
0530 PRINT@0, "[rev]DELETING";Z$: FOR B%=1 TO 10:
     I1$(B%,A%)="[space]": NEXT
0540 I1$(2,A%)="VOID": SOUND 0,3: GOTO 100
0550 PRINT@0, "SORT BY [rev]E)NTRY C)ALLSIGN"
0560 A$=INKEY$: IF A$="E" OR A$="C", X=USR(0): GOTO 570
     ELSE 550
0570 PRINT@0, "[rev]SORTING";Z$
0580 FOR A%=0 TO N%-1: FOR D%=0 TO 10: J$(D%)=I1$(D%,A%):
     NEXT D%
0590 FOR B%=A% TO N%-1

0600 IF A$="C", IF J$(2)<=I1$(2,B%), 630
0605 IF A$="E", IF J$(0)<=I1$(0,B%), 630
0610 FOR D%=0 TO 10: K$(D%)=I1$(D%,B%): I1$(D%,B%)=J$(D%)
0620 J$(D%)=K$(D%): NEXT D%
0630 NEXT B%: FOR D%=0 TO 10: I1$(D%,A%)=J$(D%): NEXT D%,A%
     : GOTO 100
0640 PRINT@0, "[rev]FILES FULL": SOUND 31,1; 31,1; 31,1;
     31,1; 0,5: RETURN
0650 IF Z%=0, Z%=1 ELSE Z%=0
0660 GOTO 100
0670 CLS: PRINT "[rev]P)AUSE S)TOP": PRINT: SOUND 0,1: FOR
     A%=0 TO N%-1
0680 FOR B%=0 TO 10: IF B%=6 OR B%=9, PRINT "[7 spaces]";:
     IF B%=9, PRINT "  ";
0690 PRINT I1$(B%,A%);" ": NEXT: PRINT: PRINT

0700 A$=INKEY$: IF A$="[space]" OR A$="P", X=USR(0): SOUND
     0,5: GOTO 700
0710 IF A$="S", 715 ELSE NEXT
0715 PRINT "[rev]C)ONTINUE"
0720 A$=INKEY$: IF A$="C", CLS: GOTO 100 ELSE 720
0730 PRINT@0, "";: INPUT "PAGE LENGTH"; L%: IF L%<7, 730
0732 PRINT@0, Z$: PRINT@0, "";: INPUT "INTER-PAGE LENGTH";
     I%
0734 PRINT@0, Z$: PRINT@0, "";: INPUT "PAGE NUMBER";P%:
     P%=P%-1
0736 PRINT@0, "SET UP PRINTER [rev]S)TART WHEN READY"
0740 A$=INKEY$: IF A$="S", X=USR(0): GOTO 750 ELSE 740
0750 PRINT@0, "[rev]PRINTING[rev off] [rev]P)AUSE S)TOP";
     Z$: GOSUB 760: GOTO 840
0760 P%=P%+1: LPRINT CHR$(14); "COMPUTER LOG BOOK";
     CHR$(15);
0770 LPRINT TAB(45); "PAGE "; P%
```

```
0780 IF P%>1, D%=4: GOTO 815
0790 D%=6: LPFINT " BY ALEX JOHNSON"; TAB(60);
     MID$(D$,1,2); "/";
0800 LPRINT MID$(D$,3,2); "/"; MID$(D$,5,2)
0810 LPRINT " (C) COPYRIGHT 1986)"
0815 LPRINT: LPRINT
0820 LPRINT "ENRY DATE   CALSGN TIME R  S  FREQ MD QTH
     [8 spaces]";
0830 LPRINT "NAME[7 spaces]REMARKS": LPRINT: RETURN
0840 FOR A%=0 TO N%-1: D%=D%+1
0843 B$=INKEY$: A$=INKEY$
0845 IF A$="[space]" OR A$="P", X=USR(0): SOUND 0,5: GOTO
     843
0848 IF A$="S", 100
0850 IF D%>L%, FOR D%=1 TO I%: LPRINT: NEXT: GOSUB 760
0860 FOR B%=0 TO 10: LPRINT I1$(B%,A%);: IF B%<10, LPRINT
     "[space]";
0870 NEXT: LPRINT: NEXT: GOTO 100
0880 CLS: PRINT "[rev]SAVE[rev off] FILENAME ";: INPUT C$"
0885 PRINT@5, "[rev]S)TART WHEN READY[rev off]";Z$
0890 A$=INKEY$: IF A$="S", X=USR(0): PRINT@4, "[rev]ING";Z$
     :GOTO 900 ELSE 890
0900 PRINT# "LOG.DATA.START",C$,S%,N%
0905 FOR A%=0 TO N%-1: A$="": FOR B%=0 TO 10: A$=A$+I1$
     (B%,A%): NEXT
0910 PRINT# "LOG",A$
0920 PRINT@98, A% C$: NEXT: GOTO 980
0930 CLS: PRINT "[rev]LOAD[rev off] FILENAME ";: INPUT C$"
0935 PRINT@5, "[rev]S)TART WHEN READY";Z$
0940 A$=INKEY$: IF A$="S", X=USR(0): PRINT@4, "[rev]ING";
     Z$: GOTO 950 ELSE 940
0950 INPUT# "LOG.DATA.START", A$ S%, N%: IF A$=C$, 960 ELSE
     950
0960 FOR A%=0 TO N%-1: INPUT# "LOG",A$
0961 I1$(0,A%)=MID$(A$,1,4): I1$(1,A%)=MID$(A$,5,10)
0962 I1$(2,A%)=MID$(A$,11,16): I1$(3,A%)=MID$(A$,17,20)
0963 I1$(4,A%)=MID$(A$,21,22): I1$(5,A%)=MID$(A$,23,24)
0964 I1$(6,A%)=MID$(A$,25,31): I1$(7,A%)=MID$(A$,32,34)
0965 I1$(8,A%)=MID$(A$,35,46): I1$(9,A%)=MID$(A$,47,56)
0966 I1$(10,A%)=MID$(A$,57,72)
0970 PRINT A% C$: NEXT
0980 SOUND 20,9: C%=1: GOTO 70
0990 IF PEEK (30779)=0, POKE 30779,1: Y%=0: GOTO 100
1000 POKE 30779,0: Y%=1: GOTO 100
1010 SOUND 0,5: PRINT@0, Z$: GOTO 100
1020 CLS: PRINT "[rev]MEMORY LEFT": POKE 30862,212: POKE
     30863,39
1030 PRINT: PRINT: PRINT T%-N%+1; "FILES LEFT"
1040 PRINT USR(X); "BYTES OF RAM FREE"
1050 PRINT USR(X$); "BYTES OF STRING RAM FREE": PRINT
1060 PRINT: PRINT: PRINT "[rev]C)ONTINUE"
1070 POKE 30862,80: POKE 30863,52
1080 A$=INKEY$: IF A$="C", C%=1: GOTO 70 ELSE 1080
```

5 of 5.

Basic program for vented box enclosures

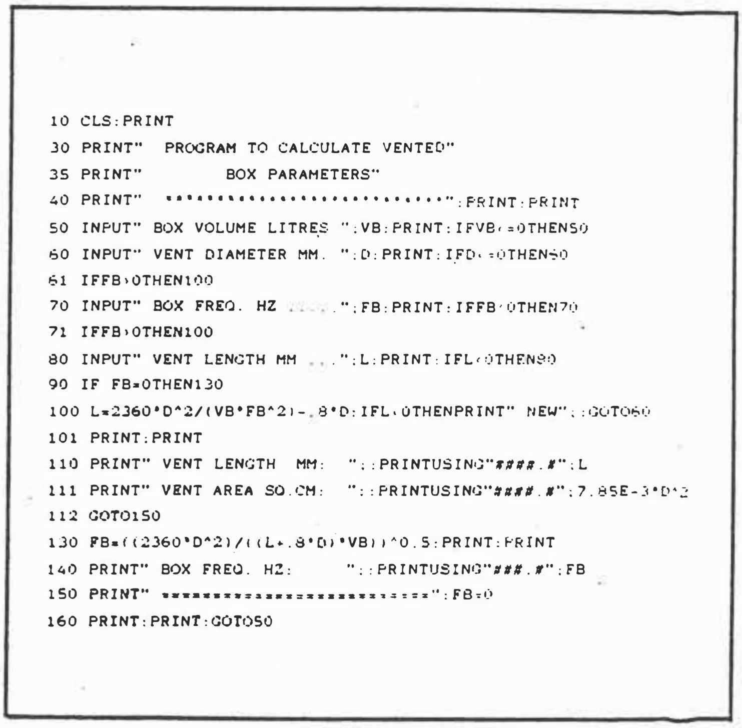
This short BASIC program for VZ computers will design the size of the vent needed in a bass reflex enclosure to tune it to a given frequency. It calculates the length of the vent from the given diameter, box volume and box frequency. Also the tuned frequency of an existing enclosure can be found from the cabinet volume and vent dimensions.

Surprising though it may be, the woofer size or type does not affect the tuned frequency; this means that you won't need any speaker data.

If the program gives a vent length of about 20mm then just a hole in the baffle is needed. Remember, however, that any vent should have a diameter not less than one quarter of the woofer diameter to prevent excessive air velocity.

For checking an existing design press RETURN when "BOX FREQ. HZ..." appears. This frequency is then calculated using the other data. If "NEW VENT DIAMETER MM. " appears, enter a new larger diameter and try again since the desired frequency cannot be achieved with the previous value.

Phil Allison,
Summer Hill, NSW.



```
10 CLS:PRINT
30 PRINT"  PROGRAM TO CALCULATE VENTED"
35 PRINT"         BOX PARAMETERS"
40 PRINT"  ****************************":PRINT:PRINT
50 INPUT" BOX VOLUME LITRES ":VB:PRINT:IFVB<=0THEN50
60 INPUT" VENT DIAMETER MM. ":D:PRINT:IFD<=0THEN60
61 IFFB>0THEN100
70 INPUT" BOX FREQ. HZ ....":FB:PRINT:IFFB<0THEN70
71 IFFB>0THEN100
80 INPUT" VENT LENGTH MM ...":L:PRINT:IFL<0THEN80
90 IF FB=0THEN130
100 L=2360*D^2/(VB*FB^2)-.8*D:IFL<0THENPRINT" NEW";:GOTO60
101 PRINT:PRINT
110 PRINT" VENT LENGTH  MM:  ";:PRINTUSING"####.#":L
111 PRINT" VENT AREA SQ.CM:  ";:PRINTUSING"####.#":7.85E-3*D^2
112 GOTO150
130 FB=((2360*D^2)/((L+.8*D)*VB))^0.5:PRINT:PRINT
140 PRINT" BOX FREQ. HZ:      ";:PRINTUSING"###.#":FB
150 PRINT" ****************************":FB=0
160 PRINT:PRINT:GOTO50
```

Memory mapping and computer number systems — using the VZ200/300

Bob Kitch

This contribution will hopefully stimulate users of the VZ200/300 (or perhaps other small micros) to think about *what* actually lies behind the keyboard or monitor. Therein resides, not simply a collection of electronic components, but a truly creative, near-art form; only restricted by the users' ingenuity. I also hope to provide a firm foundation for users to understand how they should visualise or conceive the internals of their computer. This will lead to more imaginative and rewarding use of their somewhat meagre hardware resources.

THE COMPUTER can be conceptualised (thought of) on two distinct planes: **(i)** the tangible, mechanical or physical level; and **(ii)** the intractable, esoteric or conceptual level. These two "states" are often synonymously associated with the hardware and software aspects of computing but they are not quite analogous as a brief consideration should reveal.

The realisation that the computer can in reality adopt any position between these two end-states sheds some insight into how useful a computer can be as a problem solving tool or as a creative device.

The computer is a *virtual* machine. It is incapable of doing mechanical work such as that done by an internal combustion engine. Furthermore, a computer can be *configured* via suitable programming to carry out any function that we may envisage for it. Again the analogy with a tool, for instance a spanner, is instructive. A shifting spanner has only one use — it is dedicated to that job (although I have seen some tradesmen use it as a hammer!). The important notion in computing is that our imagination is the limiting factor in determining the usefulness of the computer. We may wish to use it to monitor the security of our home or to create fantasies of our mind in intellectual and role-playing games, to carry out tedious and repetitive number crunching, or to correct text for us — etc. The spectrum of jobs is vast, and increasing almost daily.

Transformation

Somewhere between the conception of an idea and the translation of this into a computer-based chore, lies the fundamental task of the programmer. The use of the operation called "transformation" is vital to the succes of this translation. The transformation procedure takes a particular notion in our minds (the "object") and produces a "model" of this in the computer. The model may be termed the "image". A good computer image is a skilful combination of the joint hardware and software aspects of the particular computing configuration.

Often a number of step-wise transformations are required to reach the desired goal or end-point. The distribution of tasks proportioned between hardware and software depends upon

i) the resources available, and
ii) the particular talents of the person undertaking the implementation.

Electrical engineers tend to solve problems with hardware intensive solutions, whilst programmers often develop elaborate algorithmic software solutions.

Not surprisingly, *transformation* has a well developed and rigorous expression in mathematics where the somewhat allied ideas of *correspondence* (between similar objects) and *function* (connecting objects) have relevance. The box entitled "Transformation Concepts" accompanying this article futher elaborates upon some of the powerful transformation concepts — in layman's language.

The way in which "correspondence" occurs in computer science and with which perhaps most programmers are familiar, lies in the various types of codes and coding principles which are employed to connect the diversity of ideas under software control. Note that in transformations from object to image the direction of the conceptual movement may be in either direction or sense.

Thus *encoding* represents transforming the object into the image and *decoding* represents returning the object from the image. Also, multiple levels of coding are often used, depending upon where we are positioned in the hardware-software spectrum.

Codes

Consider the following code types:

i) Codes used by electronic circuits to perform digital operations e.g: binary codes.

ii) Codes used to convert decimal numbers into binary form e.g: binary coded decimal (BCD) and gray scale.

iii) Codes used to convert decimal numbers and alphabetic symbols into digital form e.g: ASCII, EBCDIC and Baudot code.

iv) Codes used by computers to perform a prescribed series of operations e.g: Z-80 instruction code and PDP8/E. ▶

1 of 6.

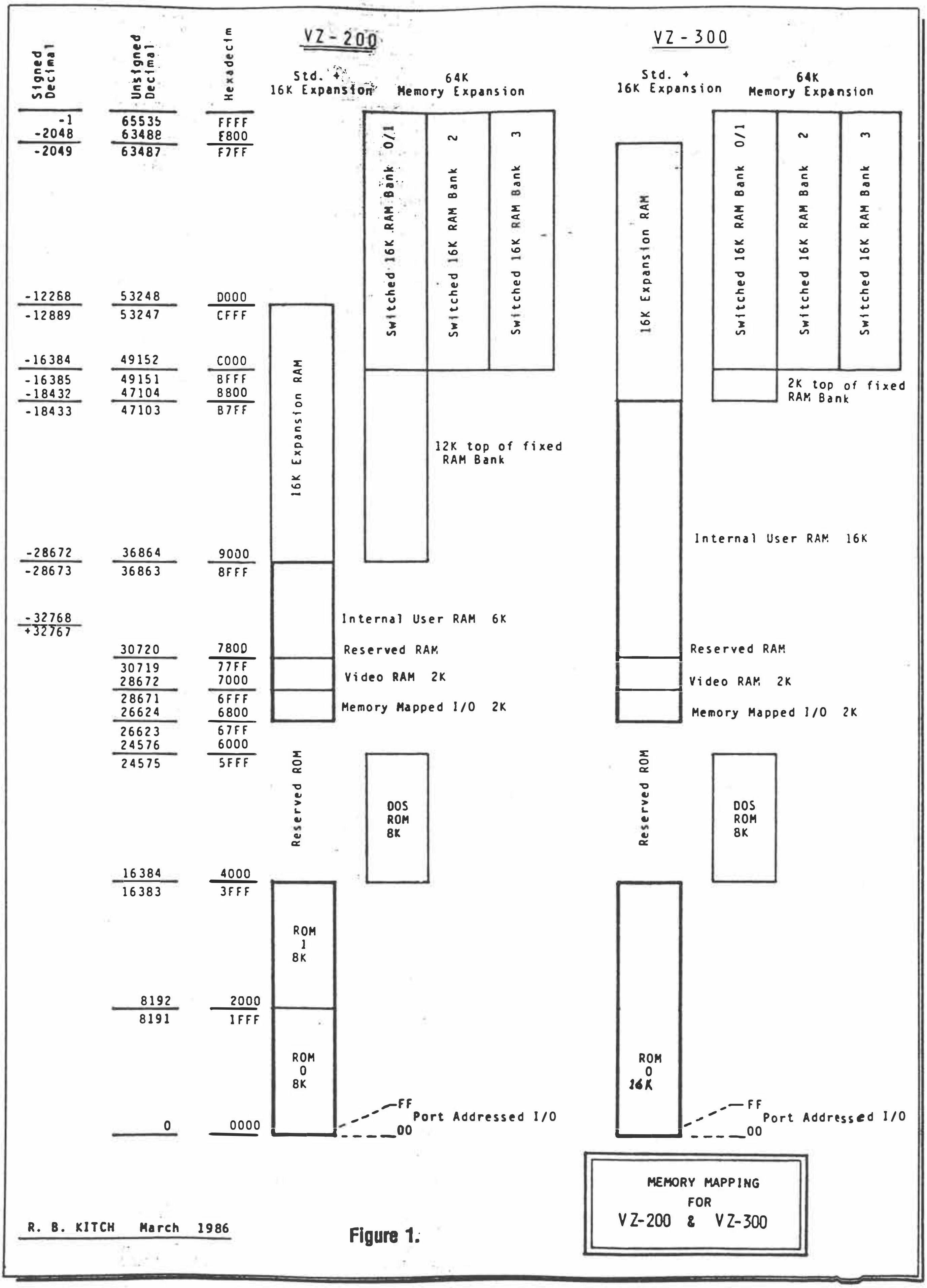


Figure 1.

2 of 6.

NUMBER BASE CONVERSION & MEMORY MAPPING

In the accompanying article the need to be able to change number representations, according to differing bases, becomes apparent.

Three bases are usually cited and often freely interchanged. These are:

base 10 — decimal	(dec./D)	uses symbols 0-9
base 16 — hexadecimal	(hex./H)	uses symbols 0-9, A-F
base 2 — binary	(bin./B)	uses symbols 0 and 1

The first system is the most familiar to us. The last is the number system of digital computers. The hex system is a convenient intermediate form between decimal and binary systems. (A fourth system to base 8, or octal — using symbols 0-7 — is sometimes employed and is also a convenient intermediate form — see later).

The accompanying table is an indispensable reference for converting base numbers. I always have this chart alongside me when programming — although some people may be fortunate enough to have an electronic calculator with base conversion functions.

Because there are three base numbers, it follows that there are six possible types of conversion. At the conclusion of this box you should be familiar with each conversion and be able to manipulate the resulting numbers.

DESCRIPTION OF TABLE

Table 1 is composed of six columns.

Column 1 (left-hand most) represents single hex digit ranging from OH to FH.

Columns 2 to 5 are labelled Most Significant 3-0 for decimal numbers.

```
MSO corresponds with 16**0*N (1*N)
MS1     "    "    "  16**1*N (16*N)
MS2     "    "    "  16**2*N (256*N)
MS3     "    "    "  16**3*N (4096*N)
```

Column 6 is the four-bit binary number corresponding to the hex digit in column 1.

One hex digit can represent half-a-byte (one-*nibble*) of binary information. Hence the close relationship between hex and binary representations. A 16-bit (two-byte) binary number maps onto four hex digits. A single byte maps onto two hex digits. (Octal or base-8 numbers map onto three bits of binary hence an eight-bit binary number can be represented by three octal digits.)

CONVERSION PROCEDURE

A. We will start converting a hex address value into its corresponding decimal and binary values.

1. Converting hex to dec. We will do this using an example. For instance, what is the decimal mapping of address 345CH? Note that the Most Significant Byte (MSB) is 34H and the Least Significant Byte (LSB) is 5CH.

The corresponding decimal for 3H (actually 3000H) appears in column MS3 and maps as 12288D. Similarly, the 4H (400H) in position MS2 maps as 1024D; 5H or 50H maps as 80D in MS1 and finally, CH corresponds to 12D from MSO.

Thus,

```
 3000H ——► 12288D
  400H ——►  1024D
   50H ——►    80D
 + CH  ——►  + 12D
 ______     ______
 345CH ——► 13404D
```

So 345CH maps as 13404D. A little involved, but easy with the table.

2. Converting hex to bin. Remember I said that hex and binary systems are closely related. Again, what is the binary mapping of address 345CH?

```
  3    4    5    C   H — from column 1
  |    |    |    |
0011 0100 0101 1100 B — from column 6
```

So the binary address for 345Ch would be —

MSB 00110100B LSB 01011100B

It could hardly be simpler!

See how difficult it would be to remember binary, but hex is much more concise and memorable?

B. Let us now take a decimal number and convert it into hex and then binary.

3. Converting dec to hex. What is the hex mapping of 22010D? This involves a little scanning of MS3-MSO of the table.

First scan down MS3 for a decimal number which is equal to, or just less than, 22010D. This is seen to be 20480D which maps as 5000H. Subtract this value from 22010D and look for the number just lower than this is MS2. For example 22010D — 20480D = 1530D. The number just lower than this in MS2 is 1280D which maps as 500H. The remainder from this operation is 250D which corresponds to 240D or FOH in MS1. The final remainder is 10D which maps as AH in MSO.

Thus:

```
 22010D
-20480D ——► 5000H
- 1280D ——►  500H
-  240D ——►   FOH
-   10D ——► +  AH
 ______     ______
     0D ——► 55FAH
```

It should be easy to convert this hex number into binary equivalent.

55FAH maps as 01010101 11111010 B

C. Let's now start with a binary number and convert it to hex and then to decimal (as previously done).

4. Converting bin to hex. By now you should be getting the idea. Simple isn't it? For example, convert the two-byte address 10011111 11010011B (looks horrible doesn't it?) into its hex value and then decimal value.

```
1001 1111 1101 0011 B — from column 6
  9    F    D    3  H — from column 1
```

Furthermore,

```
 9000H ——► 36864D
  F00H ——►  3840D
   D0H ——►   208D
 +  3H ——►  +  3D
 ______     ______
 9FD3H ——► 40915D
```

For those that have been following closely, 40915D is an unsigned decimal and mapped as a signed decimal it is

40915 — 65536 = -24621D

(see later in main article if unsure)

So in summary, we now have four ways of mapping the same address:

hex	9FD3H
unsigned decimal	40915D
signed decimal	-24621D
binary	MSB 10011111B LSB 11010011B

As a final comment and for completeness, it should be said that all the examples given herein are for unsigned decimal numbers in the range of 0 to 65535D. These map onto two-byte numbers ranging from 0000H to FFFFH in hex and 00000000 00000000 to 11111111 11111111 in binary.

The same principles apply for single-byte numbers except that the range of unsigned decimals is reduced to 0 to 255D and 00H to FFH in hex. Only MS1 and MS0 need be used in converting single-byte numbers.

Given this background then, it should be easy to calculate the appropriate values to POKE into addresses 30862D (788EH) and 30863D (788FH) to initialise the USR() command on the VZ. But more of that next time.

If you want some practice in number base conversion and require some additional confidence in following the procedures set out herein then take some addresses from the memory map and practise converting them. (I hope I get them right!)

v) Codes used by programmers to describe a problem to the computer e.g: BASIC, FORTRAN, and SAS.

vi) Codes used by the populace to have work done by a computer which is often transparent to the user. Everyday-type language is often used to communicate to the computer. (i.e: no special skills are required) e.g: POS ('Point-of-sale') terminals or pushbutton data entry panels on microwave ovens etc.

All of these forms of transformation (or coding) describe a relation or function between any object (the notion) and its correst,onding image (the programme). Flowcharting is often an intermediate coding step in the transformation process.

The memory image

Towards the hardware end of the spectrum previously alluded to lies the memory or storage system of the computer. Both the programme (or driver) and data are stored in memory which is sequentially addressed in the present generation of Von Neumann machines. Often a successful programmer "needs to get close" to this physical device — particularly in a small microcomputer environment where the memory resource is usually limited. 4K of memory usually requires some smart coding to get a worthwhile programme running — and often in machine code. Larger machines sometimes use a virtual or paged memory system so that the programmer does not need to get close to the hardware limitations. Such things as programme and storage overlaying can be done to make the memory system appear larger than it actually is. The new generation of 16- and 32-bit microprocessors include on-chip memory management functions (e.g: the 80286) to handle memory paging.

The usual way of describing the memory system of a particular computer is via the "Memory Map". This is a transformation of the actual (object) memory chips contained in the computer. This conceptual diagram (image) is an aid for the programmer. It is not a map in the same sense as a geographic (or road) map, but rather it has a one-to-one correspondence with the actual memory system. It does not actually point up any directions in the memory, in the way that a road map does. The memory map is simply a useful programmers' image of the storage which can be accessed by the CPU and the way it is organised.

VZ memory maps

(You thought I was never going to get to it!) Figure 1 is a *Universal Memory Map)* for all the VZ-200 and VZ-300 compluters. These are expandable machines in that additional memory modules, disc systems and various other peripherals can be added onto the standard system. Eight distinct types of machine are detailed:

a) standard "8K" VZ-200 and
b) standard "18K" VZ-300 (both shown in the dark outline)

In the standard machine an area of 10K is reserved for plug-in ROM cartridges. To each of the types can be added:

i) a 16K memory expansion module or
ii) a 64K memory expansion module, and additionally
iii) a disc system containing an 8K DOS can be added which utilises portion of the reserved ROM area.

Thereby eight types of VZ configuration are possible and shown in Figure 1.

A study of the range of memory expansion modules added to the VZ-200 or VZ-300 indicates that they occupy different areas of memory. This clearly shows why expansion modules are not interchangeable between models. Fortunately all of the "system areas" are compatible across models — otherwise software would not be transportable. All memory addresses below the reserved RAM (communications area) are the same on either system. This includes video RAM, memory mapped I/O, port addressed I/O and DOS ROM. As most of the peripherals are mapped into the I/O areas, these devices are also compatible between models.

Numbering systems for memory mapping

The three columns extending down the left-hand side of the map are the memory address ranges in the computer that are handled by the Z-80 microprocessor. Again the concept of "mapping" is worth noting — because the CPU uses none of the techniques shown in the columns to actually address memory! The actual (object) addressing method is a 16-bit wide binary sytem which, with suitable decoding, can resolve all the addressing functions necessary. A binary view of the addressing is unnecessarily complicated to obtaining a clear image of the VZ's address space.

An explanation of the three numbering systems used on the memory map follows.

Two forms of decimal (base 10) notation and one of hexadecimal (base 16) are shown. These are image numbering systems of the actual (object) 16-bit binary (base 2) method used by the Z-80 (Port addressed I/O uses only eight-bits of the Least Significant Byte of the address, to uniquely identify the 256 I/O ports).

If you are not particulary familiar with converting or dealing with numbers derived from differing bases, then read the boxes called "Number Base Conversion" accompanying this article.

Unsigned decimal addressing

This number system is shown in the central column of the memory map. It is perhaps the easiest to understand and explain. With a 16-bit binary number as used on the address bus, it is possible to uniquely map 2**16 or 65536 memory locations. These addresses may furthermore be mapped into a one-dimensional vector with memory location OD (2**1-1) mapped at the bottom and memory location 65535D (2**16-1) mapped at the top. This convention of "top" and "bottom" may be inverted — but top of memory is conventionally referred to as the bigger decimal number — so it makes little logical sense to have "top" at the bottom! (Note that some memory maps are drawn in this inverted sense).

Another sense of mapping is apparent and worth mentioning here. This type of map is a byte-mapped transformation as each address is actually eight-bits wide. Most data processing programming deals with bytes as the fundamental units of information. However, the Z-80 can be addressed down to bit level and hence another bit-mapped image containing 524288 (65536*8) bits could be conceived. Some controller applications make use of bit mapping because often the available RAM for programme use is rather restricted and usually the definition or resolution of the process is two-state and can be aptly modelled by a single-bit.

In the unsigned decimal mapping methods, magnitude or size of the address number uniquely defines the location of the address in memory. Relational operators such as "greater than" and "less than" work correctly. This image of addressing is most easily visualised but it bears a difficult relationship to the 16-bit object addressing.

Hexadecimal addressing

This system is shown in the third column and has a stronger relationship to the two-byte wide addressing used by the CPU ▶

4 of 6.

TRANSFORMATION CONCEPTS

In a *transformation*, the point being transformed is called the *object*. A transformation *maps* an object onto its' image according to some relation.

An *image* is the result when an object is transformed. e.g:

```
   X ——► X+2
   3 ——►  5
   0 ——►  2
   6 ——►  8
  -7 ——► -5
object   image
```

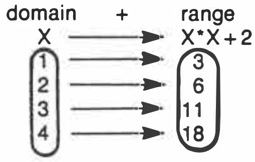
"the image of 3 is 5"

Relations are a way of connecting sets of numbers — a mapping is a special relation.

In a *mapping*, any number in the set being mapped is an object, but the entire set being mapped is usually called the *domain*.

The *domain* of a *function* is a set of numbers mapped by the function.
The domain is the object set.

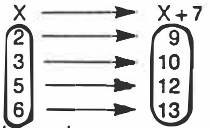
e.g: domain + range



"the set (1, 2, 3, 4) is the domain"

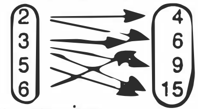
A mapping is a *relation* in which, for every object mapped, there is one, and only one, image.

e.g:



is a *valid* mapping.

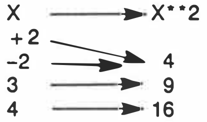
But X is a factor of



is NOT a mapping.

Functions are special relations in which each object is uniquely mapped onto one image.

e.g:



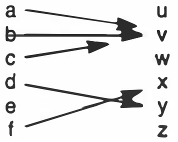
is a *valid* function.

```
But   X ——► X**0.5 (square root of X)
      1 ——► +1 or -1
      4 ——► +2 or -2
      9 ——► +3 or -3
```

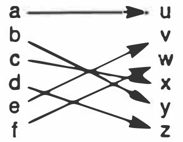
is NOT a valid function.

Correspondence has four types:
Mappings are:

Many to one correspondence

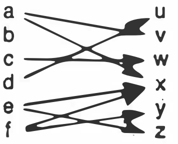


One to one correspondence

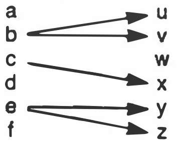


NOT mappings are:

Many to many correspondence



One to many correspondence



bus system. Each nibble (half-a-byte or four-bits) of the address is mapped onto one hexadecimal digit.

Whilst this system may appear a little unfamiliar, it has magnitude and sense — the same as the unsigned decimal notation. Therefore, similar connotations apply to the hexadecimal system as to the unsigned decimal system.

The correspondence between "top of memory" in an unexpanded VZ-200 as being 36863D or 8FFFH should be obvious from the memory map. It is simply a different way (by virtue of the number base difference) of image-mapping the same object.

In certain applications it is more convenient to use decimal notation — and in others it is clearer to use hexadecimal. If it is necessary to get close to the hardware, such as when designing the address decoding for a peripheral expansion, then hexadecimal, with its closer relationship to bus addressing, is better. Alternatively, when a programmer is wanting to locate a routine in memory, there is less need to get close to the machine, (e.g: when PEEKing or POKEing), and the more familiar decimal system is easier. In reality, experienced programmers or engineers readily flip from one to the other — particularly if they have a "smart" electronic calculator with base conversion functions.

Up to this point, all should appear to be logical, orderly and comprehensible. Unfortunately, the people who wrote the Microsoft version of the BASIC interpreter resident in the VZ (and previously used in the TRS-80 Level II, System-80 and PET) must have thought that unsigned decimal and hexadecimal were too logical and easily understood! If you try to PEEK into an address higher than 32767D or 7FFFH you will obtain an "OVERFLOW ERROR" message during run time. A look at the Reference Manual informs you that the valid address range is from -32768D to +32767D. Fair enough, but can one now assme that "top of memory" is +32767D and "bottom of memory" is -32768D. A reasonable deduction, but unfortunately, entirely incorrect! Is our faith in mathematics and logic (relational operators) misplaced?

Signed decimal addressing

The culprit is the signed decimal numbering system shown in the left hand column of the memory map. This number system is closely derived from the 16-bit binary system. The signed decimal numbering is developed from the two's complement binary system which is a method that facilitates the ▶

TABLE 1.

CONVERSION DECIMAL — HEXADECIMAL — BINARY

	Dec.				
	MSB		LSB		
	4096	256	16	1	
Hex.	MS3	MS2	MS1	MS0	Bin.
0	0	0	0	0	0000
1	4096	256	16	1	0001
2	8192	512	32	2	0010
3	12288	768	48	3	0011
4	16384	1024	64	4	0100
5	20480	1280	80	5	0101
6	24576	1536	96	6	0110
7	28672	1792	112	7	0111
8	32768	2048	128	8	1000
9	36864	2304	144	9	1001
A	40960	2560	160	10	1010
B	45056	2816	176	11	1011
C	49152	3072	192	12	1100
D	53348	3328	208	13	1101
E	57344	3584	224	14	1110
F	61440	3840	240	15	1111

5 of 6.

manipulation of negative numbers. *Do not be overwhelmed if the terms are unfamiliar as it is not essential to understand their derivation.* There exists a simple relationship between the familiar unsigned decimal and the signed decimal systems.

The simplest way of expressing the relationship is that if the unsigned decimal address is greater than 32767D then subtract 65536D from the unsigned decimal value — thereby obtaining a (negative) signed decimal. If the unsigned decimal is less than or equal to 32767D then the signed decimal value maps directly. Expressing this in BASIC is as follows:

UD = unsigned decimal value
SD = signed decimal value

To convert UD to SD:

```
15 IF UD > 32767 THEN SD = UD — 65536
                 ELSE  SD = UD
```

To convert SD to UD

```
25 IF SD < 0 THEN UD = SD + 65536
             ELSE  UD = SD
```

Refer to the mapping in the extreme left hand column of the memory map where the signed decimal system is detailed. Bottom of memory is still OD but top of memory is -1D. A very important discontinuity occurs in the numbering system at mid-memory, where adjacent bytes are numbered 32767D and -32768D. Relational operators do not work in this mapping system.

Suppose one wanted to PEEK into each consecutive memory address over the entire range of memory from OD to -1D (note!). As remarked previously, it is necessary to use signed decimals when PEEKing.

The loop written in BASIC —

```
10 FOR SD = -32768 TO +32767
20 V = PEEK (SD)
30 PRINT SD, V
40 NEXT SD
```

will not provide a consecutive listing of memory. It will commence at the base of the upper half of memory (SD = 32768D) and proceed to the top of memory (SD = -1). It will then leap to the bottom of memory (SD = OD) and proceed to the mid memory (SD = +32767D) position. Not quite what was intended!

To achieve the desired result, the following loop could be written:

```
10 FOR UD = 0 TO 65535
20 SD = UD: IF UD > 32767 THEN SD = UD — 65536
30 V = PEEK (SD)
40 PRINT SD, UD, V
50 NEXT UD
```

This will correctly step-up through memory consecutively from bottom to top (but slowly!)

Uses of the memory map

Having worked thus far through this exposition, what are some of the uses to which the memory map can be put? The first use is when it provides the programmer with a clear image (that word again) of how the addressable memory of the computer is *organised.* A number of advanced programming techniques for the BASIC interpreter also become available. For example, the *utilisation* of the memory by a BASIC programme can be determined. Overlaying of the Programme Statement Table by another routine but with retention of the Variable List Table, becomes possible. Also Assembly Language routines can be loaded into Free Space and called by the USR statement. Overwriting and corruption of programmes (images) can be avoided by reference to the map during loading. If, however, this does inadvertently occur, then the memory map becomes an important load map for debugging purposes.

A more detailed description of the I/O area (including the video RAM) mapping for the peripheral devices, and the communications area would provide more information for advanced programming techniques. Perhaps, with the Editor's indulgence, we may be able to explore these interesting areas at a future date? Meanwhile, get to *understand* your VZ'd, practise number base conversions and let your imagination run with applications for the VZ.

6 of 6.

Feedline Data Calculations for the VZ200/300

Rick Buhre VK4AIM
41 Mogford Street, Mackay, Qld. 4740

This program came about when the price of the VZ200 dropped dramatically.

The story of how this program came about is simple, but I believe it could be of interest. It all began when the price dropped on the VZ200 and Wal VK4AIV, bought one.

After learning the basics of its operation, he began to search for useful programs involving amateur radio, finding them few and far between.

Much later, I purchased a VZ300 at the same price as Wal's VZ200 and naturally asked Wal what programs he had.

Upon discovering the scarcity, I sat down and wrote a series of short programs to ease the problems of endless work with calculator, pen and paper, for amateur radio work.

Copies of these programs were given to Wal, who tidied them up and tied them together. This listing is part of the result.

The program is to enable those interested to quickly calculate parameters for the construction of coaxial cable or open wire feeder sections for matching antennas to feedlines.

The calculations are derived from standard amateur radio books and simply are converted into Basic statements.

They are as follows:

COAXIAL CABLE DATA

1 Impedance of a cable of a given size.
2 Inside diameter of outer conductor for a given impedance and inner conductor size.
3 Outside diameter of inner conductor for a given impedance and outer conductor size.
4 Cut off frequency for a cable of given size and impedance.

OPEN WIRE FEEDER DATA

1 Impedance of feeders of known wire size and spacing.
2 Spacing required for a given wire size and impedance.

There is space in the program for future additions to be inserted. I hope many amateurs will find it of use.

```
10 CLS:GOSUB3000
20 PRINT@99,"1- COAXIAL CABLE DATA"
30 PRINT@195,"2- OPEN WIRE FEEDER DATA "
40 PRINT@291,"3- "
50 PRINT@387, "4- "
60 PRINT@448,"CHOOSE OPTION":INPUTN
70 IFN=1THEN100
80 IFN=2THEN2000
85 REM************************
90 REM************************
100 GOSUB3000
110 PRINT@99,"1-IMPEDANCE OF COAXIAL"
120 PRINT@131,"            CABLE"
130 PRINT@195,"2-INSIDE DIA.OF OUTER"
140 PRINT@227,"          CONDUCTOR"
150 PRINT@291,"3-OUTSIDE DIA.OF INNER"
160 PRINT@387,"4-CUT OFF FREQUENCY"
170 PRINT@448,"CHOOSE OPTION":INPUTN
180 IFN=1THEN500
190 IFN=2THEN1000
200 IFN=3THEN1200
210 IFN=4THEN1400
220 IFN<1THEN1010
230 IFN>4THEN1010
235 REM************************
240 REM************************
500 GOSUB2500
510 INPUT"ENTER INSIDE DIAMETER OF OUTER CONDUCTOR";D1
520 INPUT"ENTER OUTSIDE DIAMETER OF INNER CONDUCTOR";D0
530 X=SQR(K)
540 Y=D1/D0
550 Z=LOG(Y)/2.30259
560 W=138*Z/X
570 PRINTW;"OHMS IMPEDENCE"
```

1 of 3 p. 10-12.

```
580 PRINT"ANOTHER TRY?Y,N"
590 INPUTA$
600 IFA$=CHR$(89)THEN500
610 IFA$=CHR$(78)THEN10
620 REM**********************
630 REM**********************
1000 GOSUB2500
1010 INPUT"ENTER IMPEDANCE";Z
1020 INPUT"ENTER OUTSIDE DIAMETEROF INNER CONDUCTOR";D
1030 X=SQR(K):Y=Z*X/138
1040 W=(10^Y)*D
1050 PRINT"INSIDE DIAMETER OF OUTER CONDUCTOR=";W
1060 PRINT"ANOTHER TRY?Y,N"
1070 INPUTA$
1080 IFA$=CHR$(89)THEN1000
1090 IFA$=CHR$(78)THEN10
1091 REM**********************
1092 REM**********************
1200 GOSUB2500
1210 INPUT"ENTER INPEDANCE";Z
1220 INPUT"ENTER INSIDE DIAMETER OF OUTER CONDUCTOR";D
1230 T=SQR(K)
1240 U=Z*T/138
1250 V=10^U
1260 W=1/V
1280 X=W*D
1290 PRINT"OUTSIDE DIAMETER OF INNER CONDUCTOR=";X
1300 PRINT"ANOTHER TRY?Y,N"
1310 INPUTA$
1320 IFA$=CHR$(89)THEN1200
1330 IFA$=CHR$(78)THEN10
1390 REM**********************
1391 REM**********************
1400 GOSUB2500
1410 INPUT"ENTER INSIDE DIA.OUTER CONDUCTOR";D1
1420 INPUT"ENTER OUTSIDE DIA.INNER CONDUCTOR";D0
1430 Z=SQR(K)
1440 X=7520/(D1+D0)*Z
1450 PRINT"CUT OFF FREQUENCY=";X;"MHZ"
1460 PRINT"ANOTHER TRY?Y,N"
1470 INPUTA$
1480 IFA$=CHR$(89)THEN1400
1490 IFA$=CHR$(78)THEN10
1491 REM**********************
1492 REM**********************
1616 REM**********************
2000 GOSUB3000
2010 PRINT@99,"1-IMPEDANCE OF OPEN"
2020 PRINT@131,"      WIRE FEEDER"
2030 PRINT@195,"2-SPACING OF OPEN"
2040 PRINT@227,"       WIRE FEEDER"
2050 PRINT@291,"3- "
2060 PRINT@387,"4-"
2070 PRINT@448,"CHOOSE OPTION":INPUTN
2090 IFN=2THEN2400
2100 IFN=1THEN2200
2110 A$=INKEY$:IFA$<>CHR$(45)THEN2110
2120 IFA$=CHR$(45)THEN10
```

2 of 3.

```
2191 REM*********************
2200 CLS:PRINT"OPEN WIRE IMPEDANCE"
2210 INPUT"SPACING";D1
2220 INPUT"DIA OF WIRE";D2
2230 X=D1/D2
2240 W=X+SQR((X*X)-1)
2250 Y=LOG(W)/2.30259
2260 Z=Y*276
2270 PRINTZ;"OHMS IMPEDANCE"
2280 PRINT"ANOTHER TRY?Y,N"
2290 INPUTA$
2300 IFA$=CHR$(89)THEN2200
2310 IFA$=CHR$(78)THEN10
2400 CLS:PRINT"TO FIND SPACING OPEN WIRE"
2410 INPUT"ENTER IMP";Z0
2420 INPUT"WIRE DIA";D
2430 X=Z0/276:Y=10^X:A=D*(Y*Y-1):S=A/(2*Y):PRINT"SPACING=";S
2440 PRINT"ANOTHER TRY?Y,N"
2450 INPUTA$
2460 IFA$=CHR$(89)THEN2400
2470 IFA$=CHR$(78)THEN10
2500 CLS:PRINT:PRINT"DIELECTRIC CONSTANTS:":PRINT"AIR=1"
2510 PRINT"POLYTHENE=2.26":PRINT"FOAM POLYTHENE=1.2"
2520 PRINT"TEFLON=2.1"
2560 INPUT"ENTER DIELECTRIC CONSTANT";K
2570 RETURN
2571 REM*********************
3000 CLS:PRINT@0," ******************************"
3010 PRINT@32," *     +++++MENU+++++       *"
3020 PRINT@64," *                          *"
3030 PRINT@96," *                          *"
3040 PRINT@128," *                          *"
3050 PRINT@160," *                          *"
3060 PRINT@192, " *                          *"
3070 PRINT@224," *                          *"
3080 PRINT@256," *                          *"
3090 PRINT@288," *                          *"
3100 PRINT@320," *                          *"
3110 PRINT@352," *                          *"
3120 PRINT@384," *                          *"
3130  PRINT@416," ******************************"
3150 RETURN
```

3 of 3.

Estimating noise in op amp stages

Estimating the noise performance of an op amp stage is easy with a little circuit analysis and a short BASIC program to take care of the maths. The program requires only two resistance values and a figure for bandwidth to compute the noise levels for six popular op amps.

by PHIL ALLISON

There are several sources of noise in an op amp stage which together account for the total background hiss level. These are the op amp itself (particularly the active devices employed in the input stage), the resistors used for gain setting, and the noise generated by the resistance of the signal source.

It must be appreciated that any resistor has a self noise level caused by thermal agitation of its free electrons. This noise, commonly known as white noise, is random and spreads across the whole frequency spectrum. Its magnitude is given by a simple formula:

where

En = RMS noise voltage

L = Boltzmann's constant 1.38×10^{-23}

T = temperature in degrees K (degrees C + 273)

B = bandwidth of measurement

R = resistor value in ohms

For example: a 10kΩ resistor at room temperature and measured with a 20kHz bandwidth will generate a noise voltage of 1.8μV. (Try some other values on your calculator to get a feel for the quantities involved).

The program presented here can be used to select the best op amp for a given application or to examine the effect on noise performance of design changes to a circuit.

Before the program can be used, two resistance values must be derived from the circuit of the op amp stage in question. These I have called *source resistance* and *input resistance*. The first is just the value in ohms of the internal resistance of the device generating the input signal.

For example, for a 200-ohm microphone use a value of 200 for the source resistance, and for a high impedance microphone (internal step-up transformer type) use a value of 50,000. If noise testing is to be done with the input shorted then use a value of 1 (one ohm) as the program will not accept a value of 0.

Input resistance

The input resistance has to be determined from the circuit of the gain stage in question and here a little analysis is needed. Note that the input resistance is not the same as the input impedance for the circuits of Fig.1 and Fig.2.

There are two common types of op amp gain stages: (1) the inverting stage as shown in Fig.1; and (2) the non-inverting stage as shown in Fig.2. The input impedance of the inverting type is equal to R1, while the input impedance of the non-inverting type is equal to Rin and may be almost any value. The signal gains of these two stages are given by the formulas beneath each diagram.

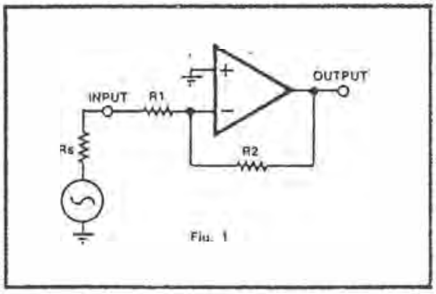


Fig.1: inverting op amp stage.
Gain = R2/R1.

Don't worry if your circuit has capacitors in series with the input or feedback ground (Fig.2) as normally these can be neglected.

In Fig.1, the input resistance is equal to R1 in parallel with R2. If R2 is more than ten times R1, then just use the value of R1.

For Fig.2, the input resistance is the same as for Fig.1 (ie, R1 in parallel R2), but if Rin is less than ten times R1 then calculate Rin in parallel with R1 and R2 as well. If there is a resistor in series with the input, add this to the input resistance.

The figure for bandwidth can be any value up to the circuit bandwidth. For audio purposes, a figure of about 16kHz is commonly adopted for specifications.

The program will, in a couple of seconds, compute the *equivalent input noise* (EIN) and noise figure for six op amps. Other op amps can easily be added to the list.

The EIN is a standard way of specifying input stage noise as it is independant of the overall gain. If you multiply the EIN figure by the gain of the stage, then you will have the noise voltage expected at the output.

The noise figure is also calculated so that the standard of performance of a circuit can be seen at a glance. It compares the stage in question with an imaginary noiseless stage and quotes the difference in decibels. A figure of 1dB would be very good and hardly worth trying to improve upon. This figure is

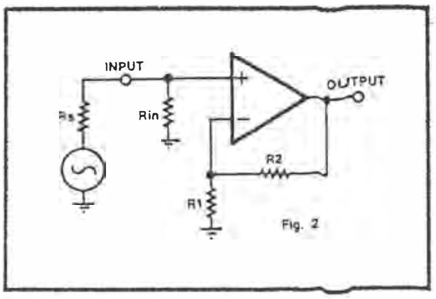
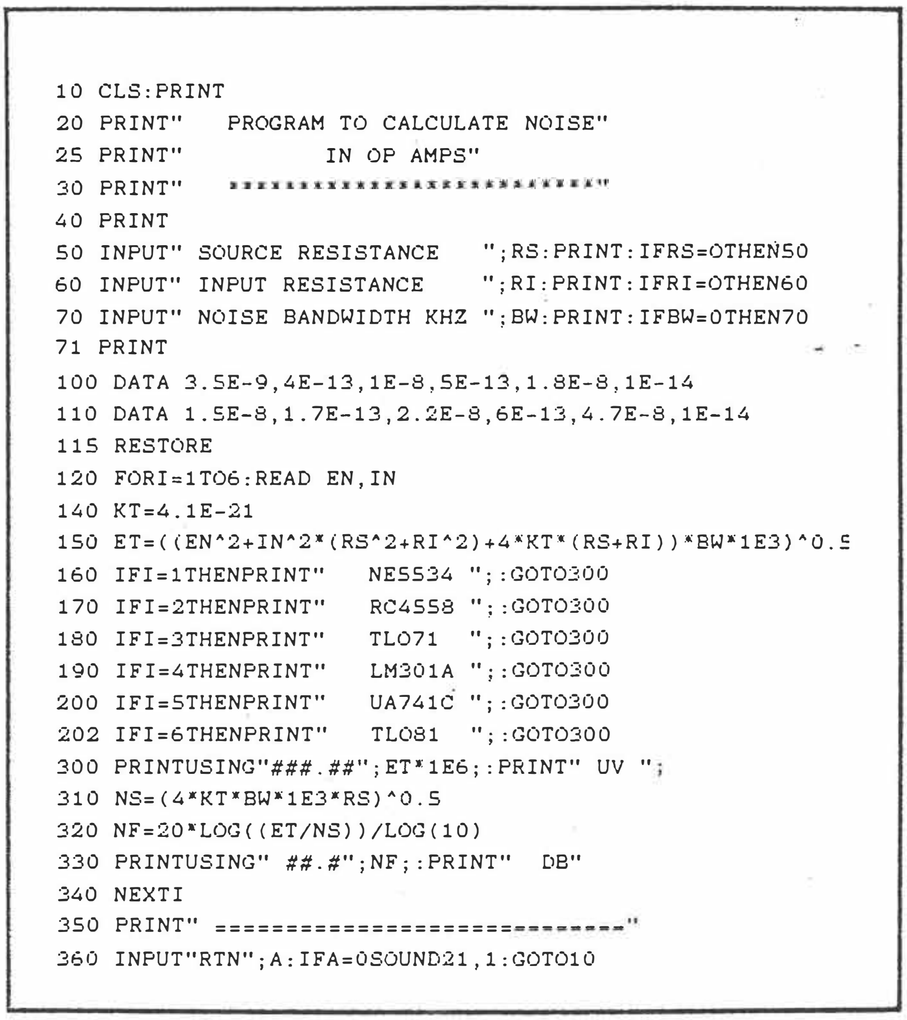


Fig.2: non-inverting op amp stage.
Gain = (R1 + R2)/R1.

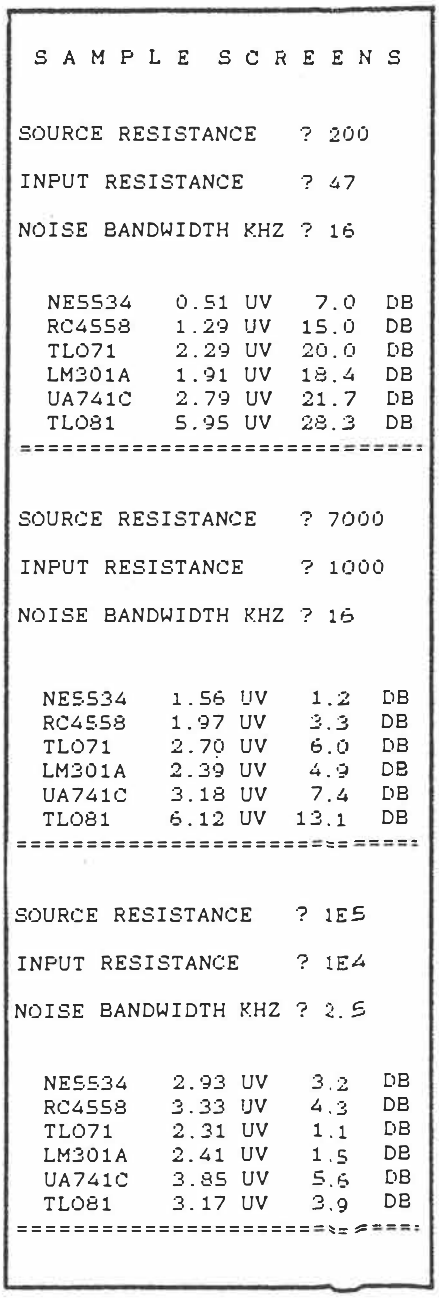
1 of 2.



```
10 CLS:PRINT
20 PRINT"   PROGRAM TO CALCULATE NOISE"
25 PRINT"           IN OP AMPS"
30 PRINT"   ***************************"
40 PRINT
50 INPUT" SOURCE RESISTANCE   ";RS:PRINT:IFRS=0THEN50
60 INPUT" INPUT RESISTANCE    ";RI:PRINT:IFRI=0THEN60
70 INPUT" NOISE BANDWIDTH KHZ ";BW:PRINT:IFBW=0THEN70
71 PRINT
100 DATA 3.5E-9,4E-13,1E-8,5E-13,1.8E-8,1E-14
110 DATA 1.5E-8,1.7E-13,2.2E-8,6E-13,4.7E-8,1E-14
115 RESTORE
120 FORI=1TO6:READ EN,IN
140 KT=4.1E-21
150 ET=((EN^2+IN^2*(RS^2+RI^2)+4*KT*(RS+RI))*BW*1E3)^0.5
160 IFI=1THENPRINT"   NE5534 ";:GOTO300
170 IFI=2THENPRINT"   RC4558 ";:GOTO300
180 IFI=3THENPRINT"   TL071  ";:GOTO300
190 IFI=4THENPRINT"   LM301A ";:GOTO300
200 IFI=5THENPRINT"   UA741C ";:GOTO300
202 IFI=6THENPRINT"   TL081  ";:GOTO300
300 PRINTUSING"###.##";ET*1E6;:PRINT" UV ";
310 NS=(4*KT*BW*1E3*RS)^0.5
320 NF=20*LOG((ET/NS))/LOG(10)
330 PRINTUSING" ##.#";NF;:PRINT"  DB"
340 NEXTI
350 PRINT" ============================="
360 INPUT"RTN";A:IFA=0SOUND21,1:GOTO10
```

Left: this program was written for the VZ300 computer but should work with little alteration on almost any computer running BASIC. The program runs each time return is pressed, so that you can enter new values.

Below: these sample screen printouts show the results for six common op amps for various circuit conditions. The program calculates both the equivalent input noise (in microvolts) and the noise performance (in dB).



```
S A M P L E   S C R E E N S

SOURCE RESISTANCE   ? 200

INPUT RESISTANCE    ? 47

NOISE BANDWIDTH KHZ ? 16


   NE5534    0.51 UV    7.0  DB
   RC4558    1.29 UV   15.0  DB
   TL071     2.29 UV   20.0  DB
   LM301A    1.91 UV   18.4  DB
   UA741C    2.79 UV   21.7  DB
   TL081     5.95 UV   28.3  DB
=============================


SOURCE RESISTANCE   ? 7000

INPUT RESISTANCE    ? 1000

NOISE BANDWIDTH KHZ ? 16


   NE5534    1.56 UV    1.2  DB
   RC4558    1.97 UV    3.3  DB
   TL071     2.70 UV    6.0  DB
   LM301A    2.39 UV    4.9  DB
   UA741C    3.18 UV    7.4  DB
   TL081     6.12 UV   13.1  DB
=============================


SOURCE RESISTANCE   ? 1E5

INPUT RESISTANCE    ? 1E4

NOISE BANDWIDTH KHZ ? 2.5


   NE5534    2.93 UV    3.2  DB
   RC4558    3.33 UV    4.3  DB
   TL071     2.31 UV    1.1  DB
   LM301A    2.41 UV    1.5  DB
   UA741C    3.85 UV    5.6  DB
   TL081     3.17 UV    3.9  DB
=============================
```

independent of gain, bandwidth and signal level.

Low noise tips

To optimise a design, the value of input resistance must be kept as low as possible. For an inverting stage, this is limited by the minimum acceptable input impedance. There is no such problem with the non-inverting stage, making it the preferred type for low noise stages. Most op amps will drive loads down to 1000 ohms or so, hence R1 plus R2 can equal this. The NE5534 can drive loads down to 600 ohms.

Don't worry about using expensive "low noise" resistors as these make no difference in an op amp stage where there is little or no DC across the resistors. Noise caused by a large voltage across a resistor is called excess noise and varies widely with resistor type.

Using the program

The formula for noise in the program appears in line 150. This sums all the noise sources involved using the published data for each op amp in turn and the result is quoted in microvolts. This data appears in lines 100 and 110 as EIN voltage and EIN current figures in volts and amps per Hz respectively. Line 320 computes the noise figure by dividing the result of line 150 by the noise of the source resistance and converting this to decibels.

When return is pressed the program runs again so that you can enter new values.

Due to device variations and the use of averaged values in the EIN data, the computed figures are not precise but are close enough to measured results to allow valid comparisons between circuits and op amps.

The program was written for a VZ300 computer but should work with little alteration on almost any computer running BASIC.

References

R.A. Fairs, Resistor Survey. *Wireless World*, October 1975.

Walter G. Jung, *IC Op Amp Cookbook*.

2 of 2.

BEAM HEADINGS AND QTH LOCATORS ON YOUR MICRO

By Greg Baker

The LOCATOR program is a dual purpose program combining a QTH Locator program and a Great Circle program. The program demands as input either (a) the QTH Locator, or (b) the latitude and longitude of the target station.

If the QTH Locator is provided as an input, the program calculates latitude and longitude of the centre of the locator square then the great circle bearing and path distances. If the latitude and longitude of the target station are input, the program calculates the QTH Locator square then the great circle bearings and path distances.

The program has been written for and tested on an unexpanded Dick Smith VZ-200 computer. The entire program is written in BASIC and should be adaptable to most BASIC versions.

QTH Locators

QTH Locators are an alternative to the use of latitude and longitude for specifying the location of amateur radio stations around the world. For this purpose, the earth's surface is first divided into 18 x 18 = 324 fields, each 20 degrees wide in longitude and 10 degrees wide in latitude.

Each of these fields is then divided into 10 x 10 = 100 squares, each 2 degrees wide in longitude and 1 degree wide in latitude. These squares are further sub-divided into 24 x 24 = 576 sub-squares of 5 minutes longitude by 2.5 minutes latitude. Figure 1 shows how these fields, squares and sub-squares are labeled.

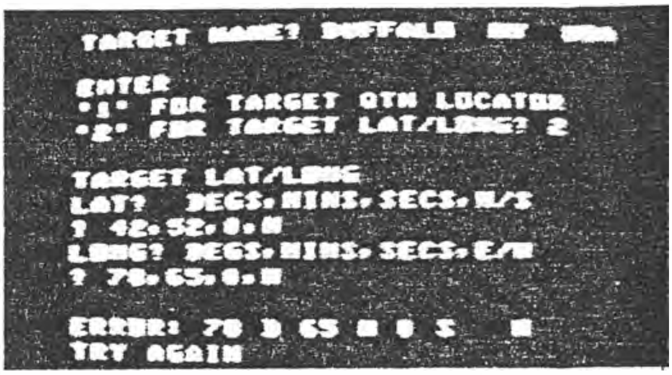
From these labels, a six-character QTH Locator is formed. Note that the two character field, square and sub-square labels are longitude first, latitude second, and are labeled consecutively from west to east for longitude and south to north for latitude.

The full six character locator has the form f1f2d1d2s1s2 where f1f2 is the alpha field locator, d1d2 is the numeric square locator, and s1s2 is the alpha sub-square locator. For example, the author's QTH is at 35°24.4' South latitude by 149°57.3' East longitude, which corresponds to a QTH Locator of QF44XO.

It is not necessary to always use the six character QTH Locator. If a coarser grid with less accuracy is satisfactory, the first four character's can be used. For less accuracy again, use just the first two characters. Further details of the QTH Locator system can be found in Tony Gilbert's 'Traffic' column, ARA Vol 7, No 9, Page 5.

Great Circle Bearings And Distances

Great Circle bearings are the true bearings for beam aiming. Due to the curvature of the earth, bearings obtained from standard (mercator projection) maps are not accuracte over more than a few degrees. Two bearings 180° apart are usually given — the short path bearing and the long path bearing. Similarly, there are two Great Circle distances — that for the short path and that for the long path.



```
TARGET NAME? BUFFALO NY USA

ENTER
'1' FOR TARGET QTH LOCATOR
'2' FOR TARGET LAT/LONG? 2

TARGET LAT/LONG
LAT?  DEGS,MINS,SECS,N/S
? 42,52,0,N
LONG? DEGS,MINS,SECS,E/W
? 78,65,0,W

ERROR: 78 D 65 M 0 S   W
TRY AGAIN
```

For more details on Great Circle bearings, see articles in ARA Vol 6, No 9, and ARA Vol 7, No 2, both available from ARA Reprints (Back Issues Department).

Flowchart and Algorithms

Unlike some other locator programs, the main calculations used here are neat and compact. The program incorporates extensive error checking, which is good for the VZ-200 but may not work on other systems.

Because the calculations are complex, great care should be taken to type them is correctly. Statements to be particularly careful with are those in lines 390, 400, 510 and 520.

The program flowchart is shown in Figure 2.

Originating Station

The program as it is written incorporates the latitude and longitude of Mount Ainslie, Canberra, as the location of the station from which the bearings are calculated. To function correctly from any other location, latitude and longitude for that QTH need to be inserted at lines 100 and 110 respectively.

Minutes of arc should be divided by 60 and added to the degrees. Seconds of arc should be divided by 3600 and added to the degree to give a decimalised latitude and longitude. Then the latitude and longitude should be give a sign — positive for north latitudes and east longitude; negative for south latitudes and west longitudes.

1 of 5.   7 Apr 87.

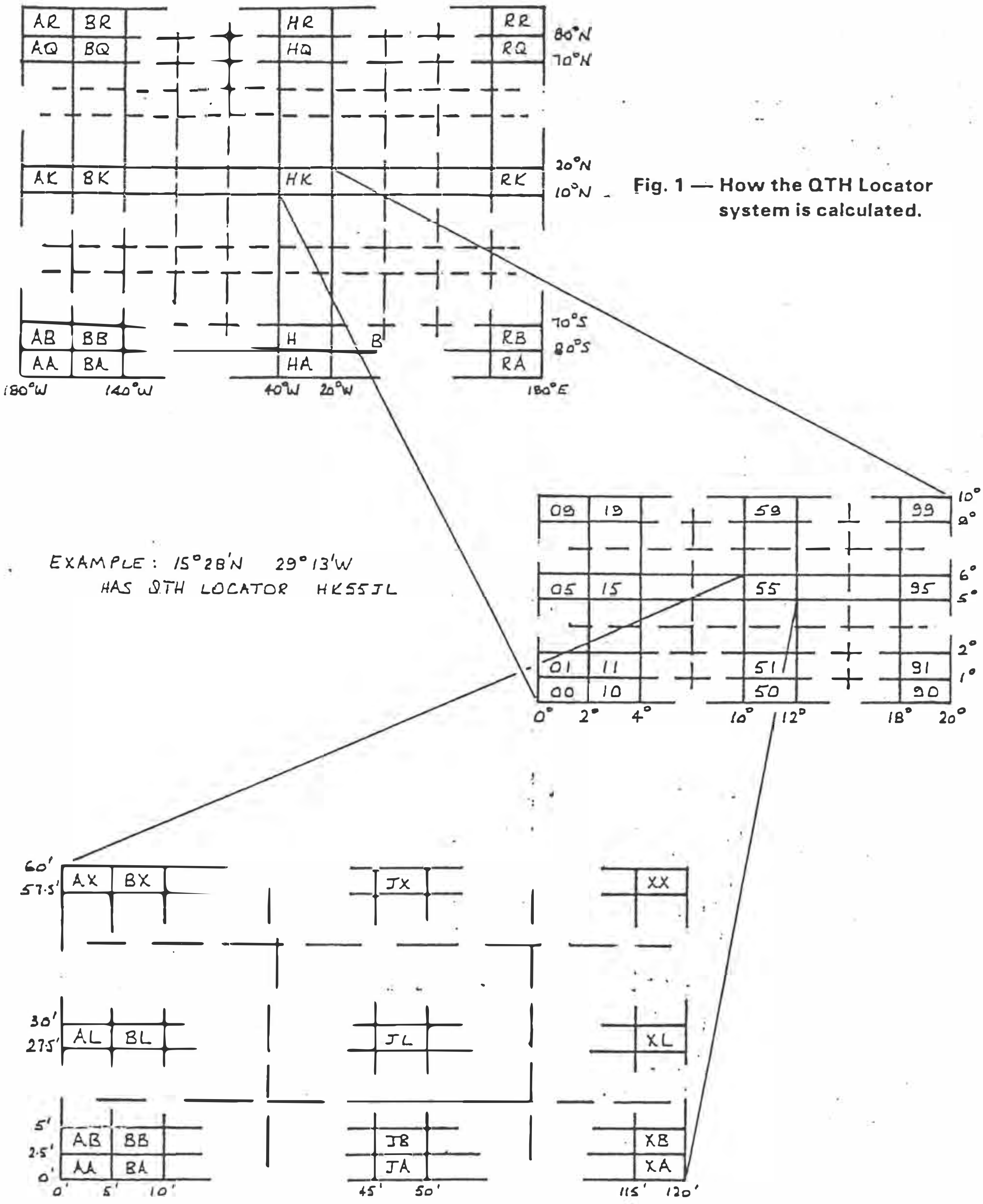


Fig. 1 — How the QTH Locator system is calculated.

For example, a station at 33°55' South by 151°10' East has a decimalised latitude of -33 + 55/60 = -33.91667 and longitude of +151 + 10/60 = +151.16667.

Alternatively, because the program allows the origin to be changed while it is running — for use away from the normal QTH for example — the user could type in their own QTH every time the program is run, although it would be easier to make the change permanent. Final output prints a new origin reminder message if this option has been exercised.

Using The Program

On running the program, the user is asked whether s/he wants to alter the latitude and longitude of their station. Enter 'Y' to choose this option or any other character to bypass it. If 'Y' is selected, you will be asked to enter the new decimalised latitude and longitude of origin.

If a valid latitude and longitude is entered, the program proceeds. Otherwise an error message is displayed for a short period and the user is requested to re-enter the origin coordinates.

Next, the program requests the target QTH name, followed by the option to enter the QTH Locator or the latitude and longitude of the location. The target name is truncated to 22 characters after entry and further truncated to nine characters if the new origin option is chosen to allow room on the printout for the new origin reminder message.

If the user chooses to enter a QTH Locator, a valid two, four or six character locator must be entered before the program will proceed to the Great Circle calculations which will use the latitude and longitude of the locator field, square or sub-square centre as the target location.

Similarly latitude and longitude, if entered, must be valid before the program will proceed.

Once great circle bearings and distances are calculated, the program prints results and asks the user to enter another target.

A few typical outputs are shown in photographs accompanying this article.

Warnings

The great circle section of the program produces errors if the target is within 50 kilometres of the origin station (when it wouldn't be usual to use a great circle program anyway), or if the target is close to either the north or south pole (although, again, it wouldn't be usual to use a great circle program to point your beam due north or south anyway).

Note that ARA Vol 9, No 4, has an article on short range, beam headings for VHF and UHF enthusiasts.

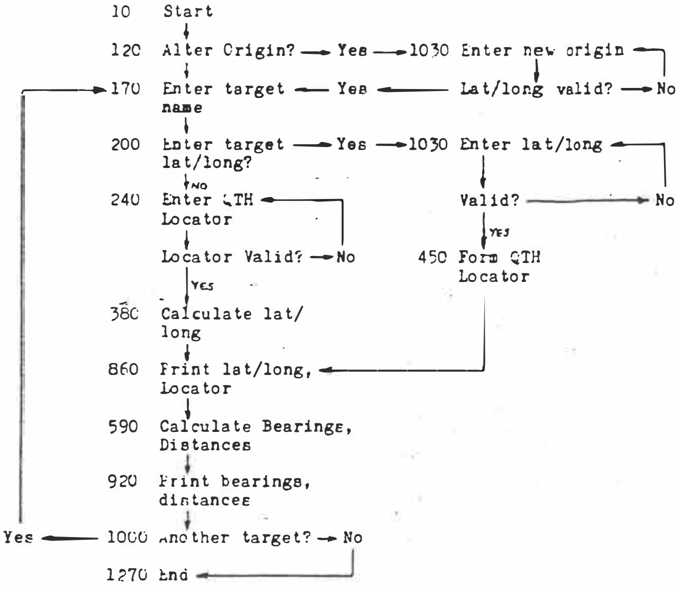
Test Data

Table 1 shows program output data for the origin station located at 35°16' South, 149° East as incorporated in program statements at lines 100 and 110. This test data shold be used to check the program before the data in lines 100 and 110 is changed for your QTH.

Copies of the program for VZ-200 can be obtained on cassette from the author for $7:00 post-paid. Write to Greg Baker, PO Box 93, Braidwood, NSW 2622. Comments and suggestions (with an SASE for reply) can be sent to the same address.

Debugged disk copies of the program modified for **Commodore VIC-20, C-64 or C-128** can be obtained by sending $10 or a blank formatted disk and $5 (includes postage) to High-Tech Media, 4 Renshaw St, Doncaster East 3109.

Diagram 2. Flcwchart

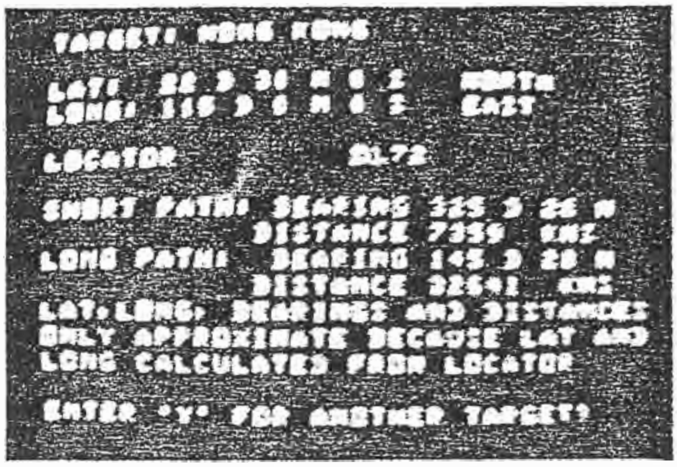

10 Start
120 Alter Origin? → Yes → 1030 Enter new origin
Lat/long valid? → No
170 Enter target name ← Yes
200 Enter target lat/long? → Yes → 1030 Enter lat/long
NO
240 Enter QTH Locator
Valid? → No
YES
Locator Valid? → No
450 Form QTH Locator
YES
380 Calculate lat/long
860 Print lat/long, Locator
590 Calculate Bearings, Distances
920 Print bearings, distances
Yes ← 1000 Another target? → No
1270 End

	Target: Name Latitude	**QTH Locator:** Longitude	**Short Path:** True Bearing	**Short Path:** Distance
1.	Buffalo 42°52'N	FN02MU 78°55'W	63°26'	15826 km
2.	Hong Kong 22°15'N	OL72CF 114°15'E	324°40'	7374
3.	Falklands 51°30'S	GD08FL 59°30'W	162°37'	9963
4.	Auckland 36°55'S	RF73JB 174°47'S	102°07'	2301

PROGRAM LISTING FOR VZ-200

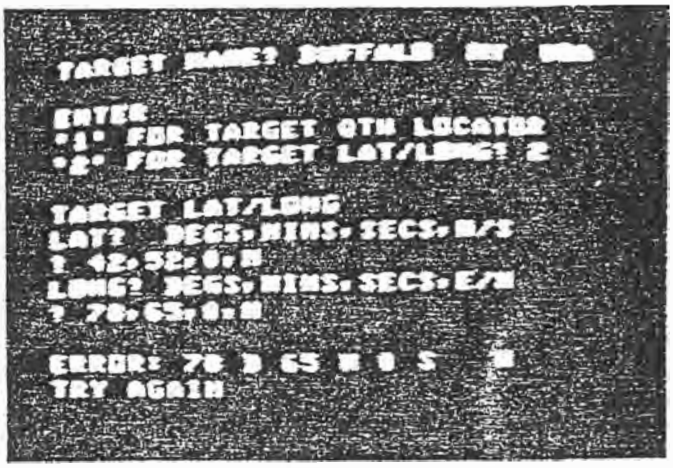
```
0010 REM PROGRAM "LOCATOR"
0020 REM GREG BAKER, BRAIDWOOD, 2622
0030 DIM C(6),CB(3),CM(3),CT(3),L(2,3),M(6),N$(2),S(2),
T(2,2),TG(2)
0035 DIM F$(2),G$(2),H%(2)
0040 DATA 65,82,10,48,57,1,65,88,0.041667
0045 DATA "NORTH","EAST","SOUTH","WEST"
0047 W$=" "
0050 FOR I=1 TO 3
0060 READ CB(I),CT(I),CM(I)
0070 NEXT I
0072 FOR I=1 TO 2
0074 READ F$(I),G$(I)
0076 NEXT I
0080 REM ORIGIN STATION LAT/LONG
0090 REM INSERT YOUR OWN QTH HERE
0100 T(1,1)=-35.2667
0110 T(2,1)=149.1667
0120 CLS
0122 PRINT"ENTER 'Y' TO ALTER ORIGIN";:INPUT Y$
0130 IF Y$<>"Y" THEN 170
0140 PRINT@192,"NEW ORIGIN LAT/LONG"
0145 C$=" ★NEW ORIGIN★"
0150 K=1
```

3 of 5.



```
TARGET: HONG KONG

LAT:  22 D 30 M 0 S   NORTH
LONG: 115 D 0 M 0 S   EAST

LOCATOR          OL72

SHORT PATH: BEARING 325 D 26 M
            DISTANCE 7359  KMS
LONG PATH:  BEARING 145 D 26 M
            DISTANCE 32641  KMS
LAT,LONG, BEARINGS AND DISTANCES
ONLY APPROXIMATE BECAUSE LAT AND
LONG CALCULATED FROM LOCATOR

ENTER 'Y' FOR ANOTHER TARGET?
```

```
0160 GOSUB 1030
0170 CLS
0175 PRINT''TARGET NAME'';:INPUT T$
0180 T$=LEFT$(T$,22)
0190 FL=0
0200 PRINT@64,''ENTER: '1' FOR TARGET QTH
LOCATOR''
0210 INPUT'' '2' FOR TARGET LAT/LONG'';Y
0220 IF Y=2 THEN 420
0230 IF Y=1 THEN 240
0235 PRINT@152,'' '':GOTO 200
0240 PRINT@192,''LOCATOR'';:INPUT Q$
0250 FL=1
0260 X=LEN(Q$)
0270 IF X=2 OR X=4 OR X=6 THEN 290
0280 PRINT@201,'' '':GOTO 240
0290 FOR I=1 TO 6
0300 C(I)=0: NEXT I
0310 FOR J=1 TO X
0320 C(J)=ASC(MID$(Q$,J,1))
0330 JJ=INT((J+1)/2)
0340 REM TEST VALIDITY OF LOCATOR
0350 IF C(J)<CB(JJ) OR C(J)>CT(JJ) THEN 280
0360 C(J)=C(J)-CB(JJ)
0370 NEXT J
0380 REM CALCULATE LATITUDE/LONGITUDE
0390 T(1,2)=-
90+C(2)★10+C(4)+C(6)/24+CM(X/2)/2
0400 T(2,2)=-
180+C(1)★20+C(3)★2+C(5)/12+CM(X/2)
0401 FOR I=1 TO 2
0402 IF T(I,2)<0 THEN H%(I)=2 ELSE H%(I)=1
0403 H%(I)=1
0404 T=ABS(T(I,2))
0405 L(I,1)=INT(T)
0406 L(I,3)=(T-L(I,1))★60
0407 L(I,2)=INT(L(I,3))
0408 L(I,3)=INT((L(I,3)-L(I,2))★60+0.5)
0409 NEXT I
0410 GOTO 585
0420 PRINT@192,''TARGET LAT/LONG''
0430 K=2 0440 GOSUB 1030
0450 REM FORM TARGET LOCATOR
```



```
TARGET NAME? BUFFALO NY USA

ENTER
'1' FOR TARGET QTH LOCATOR
'2' FOR TARGET LAT/LONG? 2

TARGET LAT/LONG
LAT?  DEGS,MINS,SECS,N/S
? 42,52,0,N
LONG? DEGS,MINS,SECS,E/W
? 78,65,0,W

ERROR: 78 D 65 M 0 S   W
TRY AGAIN
```

```
0460 FOR J=1 TO 2
0470 TG(J)=T(J,2)+90★J
0490 IF TG(J)=180★J THEN TG(J)=TG(J)-0.0001
0500 FOR K=3 TO 7 STEP 2
0510 M(K-J)=INT(TG(J)/(J★CM((K-1)/2)))
0520 TG(J)=TG(J)-M(K-J)★J★CM((K-1)/2)
0530 NEXT K
0540 NEXT J
0550 Q$=''''
0560 FOR I=1 TO 6
0570 Q$=Q$+CHR$(M(I)+CB(INT((I+1)/2)))
0580 NEXT I
0585 GOSUB 860
0590 REM CALCULATE BEARING AND DISTANCE
0600 P=T(2,1)-T(2,2)
0610 PS=1
0620 IF P<0 THEN PS=0
0630 P=ABS(P)
0640 PM=0
0650 IF P>180 THEN PM=1
0660 E=57.29578
0670 PI=3.141592654
0680 P=P/E
0690 PA=(90-T(1,1))/E
0700 PB=(90-T(1,2))/E
0710 ZZ=COS(P)★SIN(PA)★SIN(PB)+COS(PA)★COS(PB)
0720 GOSUB 1250
0730 AB=AC
0740 SK=INT(6366.707★AB+0.5)
0750 LK=40000-SK
0760 ZZ=(COS(PB)-
COS(PA)★COS(AB))/(SIN(PA)★SIN(AB))
0770 GOSUB 1250
0780 A=AC★E
0790 A=ABS(360★(PS-PM)^2-A)
0800 A1=INT(A)
0810 A2=INT((A-A1)★60+0.5)
0820 B=180+A
0830 IF B>=360 THEN B=B-360
0840 B1=INT(B)
0850 B2=INT((B-B1)★60+0.5)
0855 GOTO 920
0860 REM PRINT RESULTS
```

4 of 5.

```
0870 CLS
0880 PRINT"TARGET: ";T$
0885 IF LEN(C$)>0 THEN PRINT@17,C$
0890 PRINT@64,"LAT:
";L(1,1);"D";L(1,2);"M";L(1,3);"S ";
0895 PRINT@86,F$(H%(1))
0900 PRINT
@96,"LONG:";L(2,1);"D";L(2,2);"M";L(2,3);"S ";
0905 PRINT@118,G$(H%(2))
0910 PRINT"LOCATOR ",Q$
0915 RETURN
0920 PRINT@224,"SHORT PATH:
BEARING";A1;"D";A2;"M"
0930 PRINT" DISTANCE";SK;" KMS"
0940 PRINT"LONG PATH: BEARING";B1;"D";B2;"M"
0950 PRINT" DISTANCE";LK;" KMS"
0960 IF FL=0 THEN 1000
0970 PRINT"LAT,LONG, BEARINGS AND DISTANCES
ONLY"
0980 PRINT"APPROXIMATE BECAUSE LAT AND
LONG"
0990 PRINT"CALCULATED FROM LOCATOR"
1000 PRINT@480,"ENTER 'Y' FOR ANOTHER
TARGET";:INPUT Y$
1010 IF Y$="Y" THEN 170
1020 GOTO 1270
1030 REM INPUT LATITUDE/LONGITUDE
1035 S(1)=0: S(2)=0
1040 PRINT@224,"LATITUDE? DEGS";:INPUT L(1,1)
1041 INPUT" MINS";L(1,2)
1042 INPUT" SECS";L(1,3)
1043 INPUT" N/S ";N$(1)
1050 IF N$(1)<>"N" THEN 1070
1060 S(1)=1: GOTO 1080
```

```
1070 IF N$(1)="S" THEN S(1)=-1
1080 INPUT"LONGITUDE? DEGS";L(2,1)
1081 INPUT" MINS";L(2,2)
1082 INPUT" SECS";L(2,3)
1083 INPUT" E/W ";N$(2)
1090 IF N$(2)<>"E" THEN 1110
1100 S(2)=1: GOTO 1120
1110 IF N$(2)="W" THEN S(2)=-1
1120 FOR I=1 TO 2
1130 IF S(I)=0 THEN 1160
1132 H%(I)=1
1134 IF S(I)<0 THEN H%(I)=2
1140 T=90+(I-1)★90
1150 IF L(I,1)>=0 AND L(I,1)<=T THEN 1180
1160  PRINT"ERROR:";L(I,1);"D";L(I,2);"M";L(I,3);"S
";N$(I)
1170 PRINT "TRY AGAIN"
1172 FOR V=1 TO 1500
1174 NEXT V
1175 PRINT@224,W$
1176 FOR V=1 TO 7
1177 PRINT W$
1178 NEXT V
1179 GOTO 1030
1180 FOR J=2 TO 3
1190 IF L(I,J)<0 OR L(I,J)>60 THEN 1160
1200 NEXT J
1210 T(I,K)=L(I,1)+L(I,2)/60+L(I,3)/3600
1220 T(I,K)=T(I,K)★S(I)
1230 NEXT I
1240 RETURN
1250 AC=-ATN(ZZ/SQR(1-ZZ★ZZ))+PI/2
1260 RETURN
1270 END
```

5 of 5.

Towards a VZ-Epson printer patch

Part 1

Larry Taylor

Fed up with your clackerty old printer and long for an upgrade to one of the popular Epson or Epson-type dot matrix printers? Compatibility with the VZ has always been a problem – until now.

FED UP with your clackerty GP-100, and its less than perfect print quality? Do you long to upgrade, but know that whatever you choose, it won't be totally friendly towards your VZ?

Are you the owner of an Epson-type printer, but suffer frustration, as I did, at its lack of compatability? If so, then take heart, there is hope. The answer is a *printer patch*, that is, a program specifically written to take the place of the existing ROM routines. In this case, the aim is to make the VZ fully compatible with Epson-type printers. Recently, after many hours spent reading and experimenting, I succeeded in producing just such a program.

Having first decided to take the plunge and purchase a VZ computer, I developed a very great need, some short time later, to be able to obtain a printout of my programming efforts. On close examination of available finances, I was left with a choice between the Seikosha GP-100, a slow, noisy machine featuring an unattractive print style, and the BMC BX-80, a noticeably quieter, faster printer, possessing several attractive fonts.

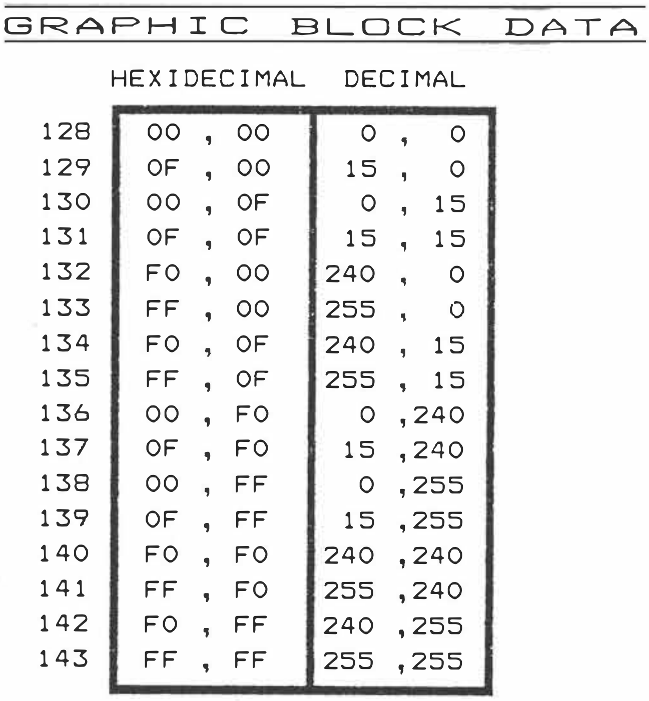
Although a seemingly easy decision, I was immediately faced with a dilemma. The former, whilst initially unattractive, especially so to anyone with sensitive hearing, had two very desirable features: namely, the ability to print the VZ's inverse and graphics characters, in addition to providing, via the COPY command, a dump of the HI-RES screen. These two factors very nearly persuaded me to choose the GP-100, but, after much deliberation, I opted for the superior print quality of the BX-80. In so doing, I resigned myself to having to go without the former's obvious advantages.

No one had at this stage even remotely hinted that I could have the best of both worlds by means of a software patch. Hindered by a lack of information and minimal understanding of computer and printer operations, I perservered with the rather primitive approach of removing all inverse and graphics characters from programs before doing a printout.

A start

Desperate to overcome this huge waste of time, I first began to deal with the problem of printing graphics characters. I realised that my printer was capable of dot graphics and that it should be able, whilst in this mode, to reproduce the shapes I desired. My early efforts, however, ended in frustration as the VZ steadfastly refused to interpret my data correctly. Only when I discovered that I could send the data directly out the ports, thus bypassing the VZ's printer driver routine, did I achieve any success.

Listing 1 gives an example of how this was accomplished. By referring to the table below, you may change the graphics block data in the listing to enable any of the other graphics charactrs to be printed. Later it will become clearer how the data to print each block was calculated.



GRAPHIC BLOCK DATA

	HEXIDECIMAL	DECIMAL
128	00 , 00	0 , 0
129	OF , 00	15 , 0
130	00 , OF	0 , 15
131	OF , OF	15 , 15
132	FO , 00	240 , 0
133	FF , 00	255 , 0
134	FO , OF	240 , 15
135	FF , OF	255 , 15
136	00 , FO	0 ,240
137	OF , FO	15 ,240
138	00 , FF	0 ,255
139	OF , FF	15 ,255
140	FO , FO	240 ,240
141	FF , FO	255 ,240
142	FO , FF	240 ,255
143	FF , FF	255 ,255

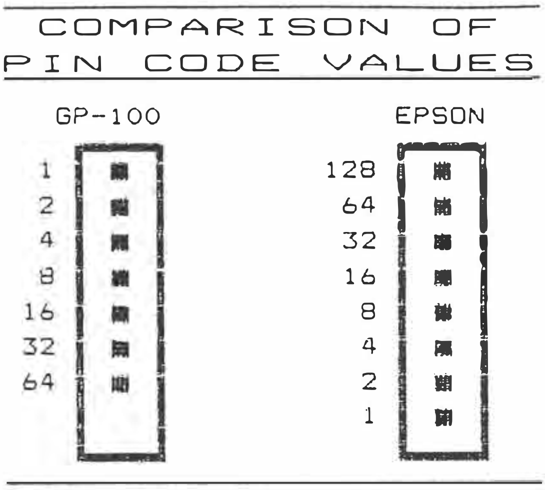
Being an avid user of Steve Olney's Extended Basic, I used my new-found knowledge to write an assembly routine, which linked into the listing routine of his program. It simply checked for graphics and inverse characters. Graphics characters were printed and inverse ones changed to non-inverse. Useful, but not totally satisfactory. On the way I had independently developed my own table of data (above), to print the graphics blocks, only to later discover that there exists in the VZ's ROM a set of data for graphics characters and another for inverse.

The graphics table occupies addresses from 02AFH to 02CEH, whilst the inverse data commences at 3B94H and ends at 3CD3H. The graphics shapes are stored in two-byte form and the inverse characters in five-byte blocks. Their existence makes it a simple enough matter to expand on the program in Listing 1 and print the graphics blocks using the ROM data instead of our own, as in Listing 2. The same may be done with the inverse characters and Listing 3 shows how this is accomplished. Unfortunately, you will notice that the resultant characters, when printed, are in fact upside down. To understand why this occurs, it is necessary to offer a brief explanation of the differences between the code values used to control firing of the pins in the printheads of Epson-type printers, and those of the GP-100 family.

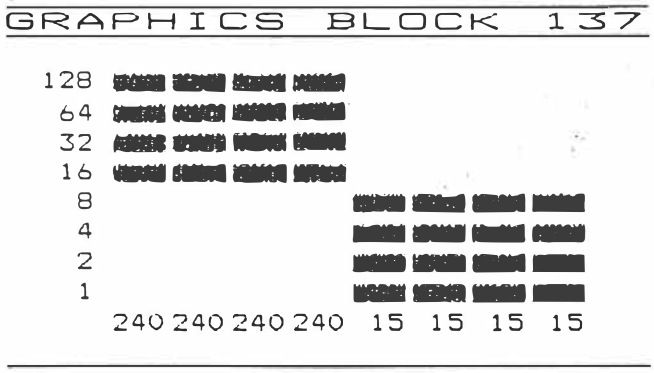
1 of 3.

The Epson-type printer

Printers of the Epson-type have eight addressable pins, while the GP-100 has the equivalent of seven pins only. In addition, the value 1, which fires the bottom pin on an Epson printer, actually triggers the top pin on the GP-100. The diagram below illustrates the differences.



To calculate the code which is required to produce a particular dot pattern we simply have to add up the values of the corresponding pins. The representation of the graphics block, CHR$(137), can be used to demonstrate how this is done. You may recall that the data values used in Listing 1 to reproduce this particular character were 240 and 15. Notice how these codes correspond to the totals at the base of each column in the diagram. If we examine the first column on the left, we can see that only the top four pins have been fired. By totalling vertically the values assigned to those pins, we arrive at the sum of 240. The same procedure is used to determine the Epson compatible code for each of the remaining columns.



It can be done

Nevertheless, data which has been prepared primarily for the GP-100, as is the case with the ROM tables, will produce inverted images if sent to an Epson printer. It is necessary, threfore, to convert the data before it can be used. Adding Listing 4 to Listing 3 will produce the desired result. I wouldn't however, advise any of you to hold your breath whilst waiting for the data to be printed. Hence, I have provided Listing 5, an assembler program, which effects the same result, only much more swiftly.

Having now managed to make the characters appear in their more conventional form, a closer examination of them will reveal numerous inaccuracies. Some, such as the 3 and 5, are more noticeable than others, but no less than a dozen of the characters are flawed. After progressing so far, this is a disappointing development but one which will prove, later, to be not insurmountable. In the interim, we need to explore further how we might utilise our somewhat imperfect data.

2 of 3.

Fortunately, the designers of the ROM foresaw the possibility that potential users may want to use a different printer. As a result, a vector has been used to point to the location of the printer driver. All output to the printer is directed via a driver routine, which, among other things, checks for control codes and keeps track of line feeds. In the VZ, a block of the communications area of RAM from 7825H to 782CH has been set aside for printer operations, allowing temporary storage of values such as the number of lines printed. Of greatest interest to us is the contents of 7826H-7827H. This is the start of the driver routine, and the cause of our problems, because it is geared to expect that owners of VZeds will be using GP-100 type printers. However, since the previous address lies in RAM, it is possible to insert a pointer to our own driver routine at this location. Once accomplished, all future LPRINT and LLIST commands will be directed, ultimately, to our own printer routine.

We have now proceeded part way to installing a valuable routine for owners of Epson-type printers, but we are still unable to make use of the COPY command. The primary advantage of which is that it allows a dump of the HI-RES screen to be made to the printer. Implementing this very desirable feature will prove to be somewhat more challenging.

LISTING 1 : PRINT A SINGLE GRAPHICS BLOCK

```
100 REM #####################################
101 REM # PUT PRINTER IN GRAPHICS MODE      #
102 REM #####################################
110 LPRINTCHR$(27);CHR$(75);
120 FOR T=1 TO 2
130   READ D:GOSUB 510
140 NEXT T
200 REM #####################################
205 REM # READ EACH DATA VALUE IN TURN      #
210 REM # AND THEN PRINT IT FOUR TIMES      #
215 REM #####################################
220 FOR N%=1 TO 2
230   READ D
240   GOSUB 510:GOSUB 510
250   GOSUB 510:GOSUB 510
400 NEXT N%
410 LPRINT:END
500 REM #####################################
501 REM # OUTPUT TO PRINTER VIA THE PORTS   #
502 REM #####################################
510 IF INP(0)<>254 THEN GOTO510
520 OUT 13,D:OUT 14,D
530 RETURN
540 REM #####################################
545 REM # NUMBER OF BYTES TO BE PRINTED     #
550 REM # IN LOW BYTE, HIGH BYTE FORM       #
555 REM #####################################
560 DATA 8,0
565 REM #####################################
570 REM # GRAPHIC BLOCK DATA                #
575 REM #####################################
580 DATA 240,15
```

LISTING 2 : PRINT THE ROM GRAPHICS BLOCKS

```
100 REM #####################################
101 REM # PUT PRINTER IN GRAPHICS MODE      #
102 REM #####################################
110 LPRINTCHR$(27);CHR$(75);
120 FOR T=1 TO 2
130   READ D:GOSUB 510
140 NEXT T
150 REM #####################################
151 REM # LOCATION GRAPHICS TABLE 02CEH     #
152 REM #####################################
```

▶

```
160 M=687
200 REM #####################################
205 REM # READ DATA FOR GRAPHICS BLOCKS     #
210 REM # AND PRINT EACH VALUE 4 TIMES      #
215 REM #####################################
220 FOR N%=1 TO 32
230   D=PEEK(M)-128 :M=M+1
240   GOSUB 510:GOSUB 510
250   GOSUB 510:GOSUB 510
260 REM #####################################
265 REM # THIS LINE SEPARATES CHARACTERS    #
270 REM # FROM EACH OTHER BY A DOT WIDTH    #
275 REM #####################################
280  IF N%/2 = INT(N%/2) THEN D=0 :GOSUB 510
400 NEXT N%
410 LPRINT:END
500 REM #####################################
501 REM # OUTPUT TO PRINTER VIA PORTS       #
502 REM #####################################
510 IF INP(0)<>254 THEN GOTO510
520 OUT 13,D:OUT 14,D
530 RETURN
540 REM #####################################
545 REM # NUMBER OF BYTES TO BE PRINTED     #
550 REM # IN LOW BYTE, HIGH BYTE FORM       #
555 REM #####################################
560 DATA 144,0
```

LISTING 3 : PRINT THE ROM INVERSE CHARACTERS

```
100 REM #####################################
101 REM # PUT PRINTER IN GRAPHICS MODE      #
102 REM #####################################
110 LPRINTCHR$(27);CHR$(75);
120 FOR T=1 TO 2
130   READ D:GOSUB 510
140 NEXT T
150 REM #####################################
151 REM # LOCATION OF INVERSE TABLE 3B94H   #
152 REM #####################################
160 M=15252
200 REM #####################################
201 REM # NUMBER OF INVERSE CHARACTERS      #
202 REM #####################################
210 FOR N%=1 TO 64
220       D=255:GOSUB 510
230 REM #####################################
231 REM # NUMBER OF BYTES PER CHARACTER     #
232 REM #####################################
240   FOR R%=1 TO 5
250       D=PEEK(M):M=M+1
339 REM #####################################
340 REM # PRINT ONE COLUMN                  #
341 REM #####################################
350       GOSUB 510
360   NEXT
370     D=255:GOSUB 510
400 NEXT N%
410 LPRINT:END
500 REM #####################################
501 REM # OUTPUT TO PRINTER VIA THE PORTS   #
502 REM #####################################
510 IF INP(0)<>254 THEN GOTO510
520 OUT 13,D:OUT 14,D
530 RETURN
535 REM #####################################
540 REM # NUMBER OF BYTES TO BE PRINTED     #
550 REM # IN LOW BYTE, HIGH BYTE FORM       #
555 REM #####################################
560 DATA 192,1
```

LISTING 4 : CONVERT THE DATA FOR THE EPSON PRINTER

```
260 REM #####################################
261 REM # CHANGE CODE FROM GP-100 TO EPSON #
262 REM #####################################
270       IF D=189 OR D=255 THEN 320
280       V=0:E=0
290      FOR F%=7 TO 0 STEP -1
300       P=2^F%:IF D<P THEN 320
310       E=E+2^V:D=D-P
320       V=V+1
330      NEXT:D=E
```

LISTING 5 : PRINT THE ROM INVERSE CHARACTERS

```
0001 ;####################
0002 ;#  PUT PRINTER IN  #
0003 ;#  GRAPHICS  MODE  #
0004 ;####################
0005      LD   A,27
0006      CALL 3ABAH
0007      LD   A,75
0008      CALL 3ABAH
0009      LD   A,192
0010      CALL 3ABAH
0011      LD   A,1
0012      CALL 3ABAH
0013 ;####################
0014 ;# LOCATION OF THE  #
0015 ;#  INVERSE TABLE   #
0016 ;####################
0017      LD   HL,3B94H
0018 ;####################
0019 ;# NUMBER OF INVERSE#
0020 ;#    CHARACTERS    #
0021 ;####################
0022      LD   B,64
0023 NEXT PUSH BC
0024      LD   A,255
0025      CALL 3ABAH
0026 ;####################
0027 ;# NUMBER OF BYTES  #
0028 ;#  PER CHARACTER   #
0029 ;####################
0030      LD   B,5
0031 PRNT LD   A,(HL)
0032      CALL CVRT
0033      CALL 3ABAH
0034      INC  HL
0035      DJNZ PRNT
0036      LD   A,255
0037      CALL 3ABAH
0038      POP  BC
0039      DJNZ NEXT
0040      RET
0041 ;####################
0042 ;# CHANGE CODE FROM #
0043 ;# GP-100 TO EPSON  #
0044 ;####################
0045 CVRT PUSH BC
0046      LD   B,8
0047 ROTA RR   A
0048      RL   C
0049      DJNZ ROTA
0050      LD   A,C
0051      POP  BC
0052      RET
```

— from page 30

chromium to resist corrosion) and a solid "beta alumina" electrolyte separates anode and cathode. The cell is sealed and filled with argon.

During discharge, sodium ions pass through the electrolyte from anode to cathode, forming sodium sulphide at the cathode, the reaction generating the current. Recharging is achieved as with other storage batteries, by passing a current through it in reverse. One problem, though. These cells will only deliver power when operated above 270 degrees Celsius. They have an operating temperature ceiling of 410 degrees C. They must be heated to 'start up' and to maintain them within the operating temperature range, they have to be fully charged and then at least 80% discharged each day. If unused for nine hours, temperature falls below the 270 degrees C.

Sodium-sulphur cells exhibit a terminal voltage of around 2 V and may last some five years or 6000 charge-discharge cycles, which betters the typical lead-acid battery life cycle. In addition, its terminal voltage remains constant until it reaches about 70% of its discharge capacity before tapering off.

Suggested application encompass commercial vehicles such as delivery vans and buses, and military submarines. Satellite applications are also suggested as sodium-sulphur cells are only 20% of the weight of equivalent NiCad batteries of the same Ah output.

3 of 3

A VZ-Epson printer patch — the search continues

Larry Taylor

Part 2

IN THE PREVIOUS instalment, printing of the VZ's inverse and graphics characters had been made possible. At this point, the ideal enhancement to our printer patch would be to enable the VZ's COPY command to function correctly when matched with an EPSON type printer. This should be possible, but we must first examine why the usual means for intercepting BASIC key words, during programme execution, won't work in the case of the COPY command.

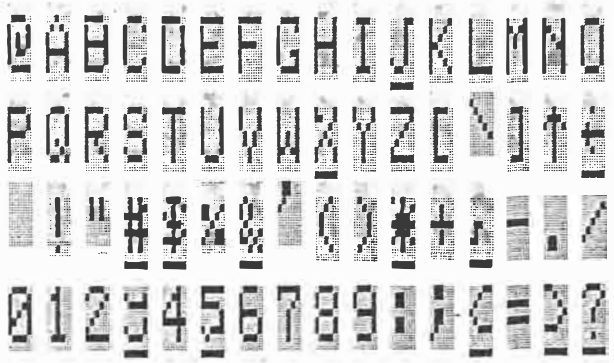
The VZ's ROM owes much to that used in the earlier TRS-80 computers. The COPY routine, however, is one of a number of additions which greatly enhance the VZ's capabilities. As such, it contains none of the DOS exits, which are to be found in the older sections of the ROM. These exits, or "vectors", are calls to an area in the communications area of RAM, and provide the means by which some BASIC commands may be altered or redirected. Since the VZ DOS makes no use of these vectors, none have been provided in the newer sections of the ROM. My initial hopes dashed, I began to investigate the method used to integrate the DOS into the VZ's operating system. In doing so, I uncovered an alternative vector, one which would make it possible for us to not only intercept the COPY command, but also open the door to further enhancements to the VZ's BASIC.

How so?

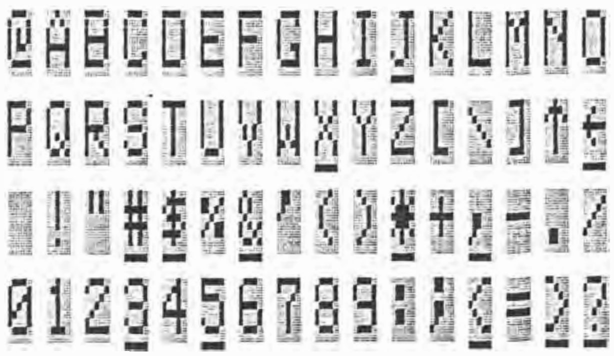
It is important to understand, initially, why this type of modification is possible. When we write a BASIC programme, we are creating what we hope will be a precise set of instructions. Unfortunately, before the computer can understand and respond to our commands, each instruction in turn has to be painstakingly translated or intrpreted. This is the reason for BASIC's slowness, and it can really only be effectively overcome by having the programme translated or compiled prior to execution. Yet, because a BASIC programme is interpreted as it runs, it is possible that additional commands may be added to the language, provided they are intercepted and executed prior to reaching the VZ's own interpreter. This is precisely what happens when a disk operating system is added. New commands enabling disk operations to be performed; supplementing the existing BASIC. In the case of the COPY command, we are seeking to redirect it to a routine compatible with EPSON type printers, and on completion, have it return as though all had proceeded normally.

As I undertook to produce this extension to the patch, I found myself venturing much further than I had originally intended. The project involved modifying the existing ROM routine, as well as enhancing the COPY command to provide for a second screen dump routine of my own design. Furthermore, I allowed for a copy of the LO-RES screen without the usual linefeeds. I also sought to eliminate those unfortunate flaws in the inverse character data. Listing 1, which was kindly supplied by Bob Kitch, enables a closer examination of the inverse characters held in ROM, by displaying them on the HIRES screen. By relocating the ROM table to RAM at the top of memory the necessary modifications to the data have been made possible.

VZ ROM, INVERSE CHARACTER SHAPE TABLE



VZ ROM, PRINTER PATCH MODIFIED TABLE



(note changes to underlined characters)

The accompanying illustration allows a comparison to be made between the ROM characters, at top, and those in the shape table addressed by the printer patch. Incidentally, should you decide that you still don't like the look of the amended characters, it is possible, using the same approach, to either further refine them, or even custom design a completely new set.

Inspired at having overcome this obstacle, and because I have written a number of programs using an Extended BASIC, I wanted the routine to be able to list those commands, which would not normally be recognised. The final aim was to deal with the printer's unimpressive performance, signalled by a dramatic decrease in speed, each time it had to print a graphics or inverse character. The solution I chose to minimise these delays was to feed the data into a section of RAM, which would act as a collection area or buffer, prior to printing. A discussion in detail of how each of these refinements was implemented would only serve to complicate what is otherwise a relatively straightforward procedure. I have elected, instead, to demonstrate how to intercept and enhance an existing keyword on a smaller scale by using another of the VZ's commands.

1 of 3.

Enhanced CLS

Tandy's Colour Computer has an enhanced CLS command which enables the user to clear the screen to any one of nine background colours. The syntax is CLSn, where n may be a number in the range 0-8. To illustrate how enhancements to the existing language can be accomplished, this command will be necessary to examine further how the VZ operates.

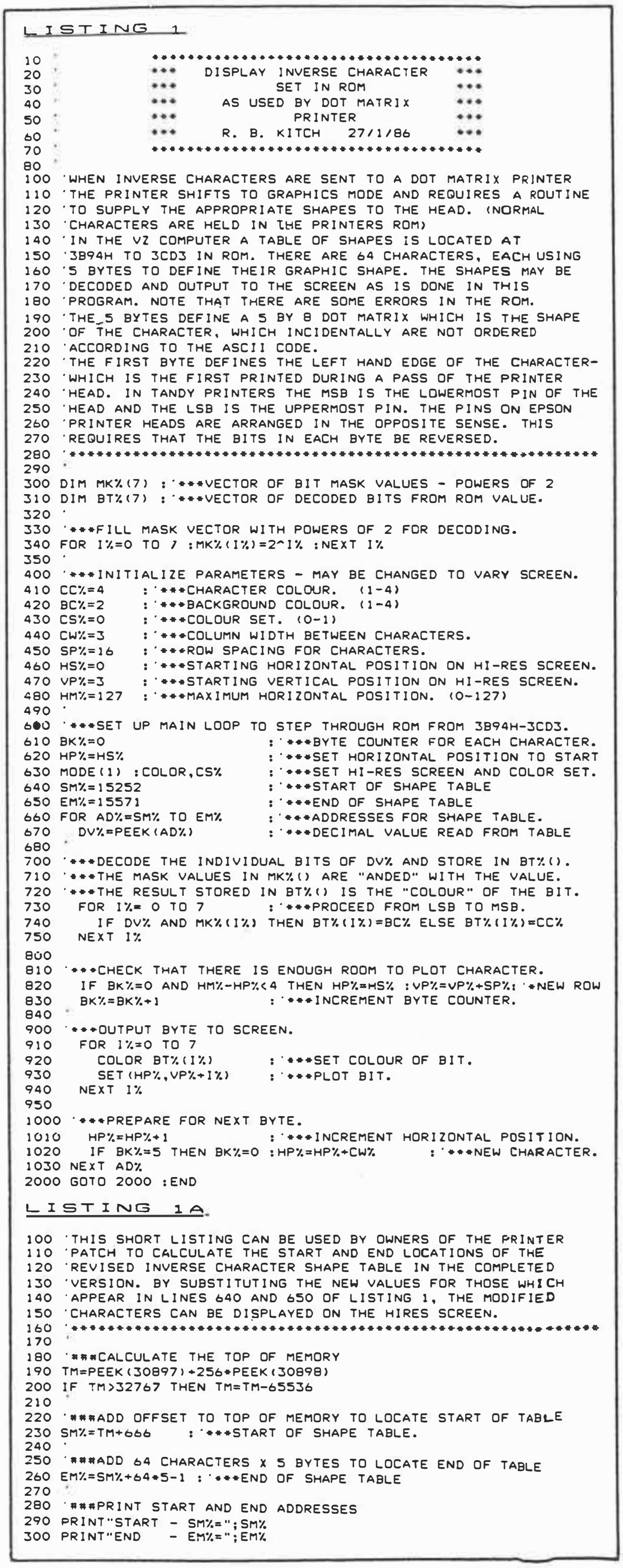
When a BASIC program is RUN, control passes to a machine language ROM routine, the Execution Driver at 1D5AH, which scans each line of the BASIC programme as it comes to it and begins to translate it. Part of the translation process involves looking for tokens. These are values in the range 128-250 (80H-FAH) that take the place of BASIC reserved words e.g: CLS = 132 (84H). Once the word has been identified and checked for correct syntax, control is passed to the corresponding ROM routine before returning to continue the translation.

On power-up, the address of the routine which examines each byte in a line of BASIC, is stored at 7804H. This is the vector hinted at earlier, and in a non-disk VZ it will normally contain a pointer to the RST 10H routine at 1D78H. Because this vector is in RAM it can be easily changed. This was done so that at a later stage the DOS could be included.

At least three different versions of the VZ DOS could be included that I am aware of, and two of these display the same version number on power up. Consequently, the only fixed location common to all three versions is a jump table commencing at 4005H. This makes it difficult to refer to an actual address within the DOS, where command processing is carried out. However, since all processing must be channelled via the above-mentioned vector, a peek at this address will uncover the whereabouts of the DOS interpreter. A close examination of this region of the DOS will reveal how the added disk commands are interpreted and implemented. This information will enable us to introduce into the system an enhanced command of our own choosing. The trick is to ensure that, as far a the VZ's interpreter is concerned, nothing unusual has happened.

The accompanying assembly language programme in Listing 2, with its associated comments, shows in greater detail how this is accomplished. If you do not have access to an Editor Assembler, Listing 3 is a BASIC version, which pokes the routine into memory. Having adjusted the top of memory pointer, the address at 7804H is stored and replaced by our own. The programme then locates the new routine at the top of the memory. Now each time a byte is to be examined during execution it must first pass through our checkpoint. Once the origin of the call is established, the routine looks for the CLS token, 132 (84H).

Only when it has been located does the routine proceed to examine the next byte. This is checked to see if it lies in the range 0-9. Once it has passed this test, the clear screen routine is implemented, after first calculating the appropriate value, with which to fill the screen. You will notice that not only is it necessary to check for the new command, but also to provide the routine which implements it. In this case a simple block load to the screen has been used. Control is then returned to the ROM processing routine, which prepares to examine the byte following ournew command. So, as far as the VZ knows, everything is continuing normally. Tricky isn't it?



LISTING 1

```
10 '               ***************************************
20 '               ***   DISPLAY INVERSE CHARACTER   ***
30 '               ***          SET IN ROM           ***
40 '               ***     AS USED BY DOT MATRIX     ***
50 '               ***           PRINTER             ***
60 '               ***     R. B. KITCH   27/1/86     ***
70 '               ***************************************
80 '
100 'WHEN INVERSE CHARACTERS ARE SENT TO A DOT MATRIX PRINTER
110 'THE PRINTER SHIFTS TO GRAPHICS MODE AND REQUIRES A ROUTINE
120 'TO SUPPLY THE APPROPRIATE SHAPES TO THE HEAD. (NORMAL
130 'CHARACTERS ARE HELD IN THE PRINTERS ROM)
140 'IN THE VZ COMPUTER A TABLE OF SHAPES IS LOCATED AT
150 '3B94H TO 3CD3 IN ROM. THERE ARE 64 CHARACTERS, EACH USING
160 '5 BYTES TO DEFINE THEIR GRAPHIC SHAPE. THE SHAPES MAY BE
170 'DECODED AND OUTPUT TO THE SCREEN AS IS DONE IN THIS
180 'PROGRAM. NOTE THAT THERE ARE SOME ERRORS IN THE ROM.
190 'THE 5 BYTES DEFINE A 5 BY 8 DOT MATRIX WHICH IS THE SHAPE
200 'OF THE CHARACTER, WHICH INCIDENTALLY ARE NOT ORDERED
210 'ACCORDING TO THE ASCII CODE.
220 'THE FIRST BYTE DEFINES THE LEFT HAND EDGE OF THE CHARACTER-
230 'WHICH IS THE FIRST PRINTED DURING A PASS OF THE PRINTER
240 'HEAD. IN TANDY PRINTERS THE MSB IS THE LOWERMOST PIN OF THE
250 'HEAD AND THE LSB IS THE UPPERMOST PIN. THE PINS ON EPSON
260 'PRINTER HEADS ARE ARRANGED IN THE OPPOSITE SENSE. THIS
270 'REQUIRES THAT THE BITS IN EACH BYTE BE REVERSED.
280 '*************************************************************
290 '
300 DIM MK%(7) :'***VECTOR OF BIT MASK VALUES - POWERS OF 2
310 DIM BT%(7) :'***VECTOR OF DECODED BITS FROM ROM VALUE.
320 '
330 '***FILL MASK VECTOR WITH POWERS OF 2 FOR DECODING.
340 FOR I%=0 TO 7 :MK%(I%)=2^I% :NEXT I%
350 '
400 '***INITIALIZE PARAMETERS - MAY BE CHANGED TO VARY SCREEN.
410 CC%=4     :'***CHARACTER COLOUR.  (1-4)
420 BC%=2     :'***BACKGROUND COLOUR. (1-4)
430 CS%=0     :'***COLOUR SET. (0-1)
440 CW%=3     :'***COLUMN WIDTH BETWEEN CHARACTERS.
450 SP%=16    :'***ROW SPACING FOR CHARACTERS.
460 HS%=0     :'***STARTING HORIZONTAL POSITION ON HI-RES SCREEN.
470 VP%=3     :'***STARTING VERTICAL POSITION ON HI-RES SCREEN.
480 HM%=127   :'***MAXIMUM HORIZONTAL POSITION. (0-127)
490 '
600 '***SET UP MAIN LOOP TO STEP THROUGH ROM FROM 3B94H-3CD3.
610 BK%=0                     :'***BYTE COUNTER FOR EACH CHARACTER.
620 HP%=HS%                   :'***SET HORIZONTAL POSITION TO START
630 MODE(1) :COLOR,CS%        :'***SET HI-RES SCREEN AND COLOR SET.
640 SM%=15252                 :'***START OF SHAPE TABLE
650 EM%=15571                 :'***END OF SHAPE TABLE
660 FOR AD%=SM% TO EM%        :'***ADDRESSES FOR SHAPE TABLE.
670   DV%=PEEK(AD%)           :'***DECIMAL VALUE READ FROM TABLE
680 '
700 '***DECODE THE INDIVIDUAL BITS OF DV% AND STORE IN BT%().
710 '***THE MASK VALUES IN MK%() ARE "ANDED" WITH THE VALUE.
720 '***THE RESULT STORED IN BT%() IS THE "COLOUR" OF THE BIT.
730   FOR I%= 0 TO 7          :'***PROCEED FROM LSB TO MSB.
740     IF DV% AND MK%(I%) THEN BT%(I%)=BC% ELSE BT%(I%)=CC%
750   NEXT I%
800
810 '***CHECK THAT THERE IS ENOUGH ROOM TO PLOT CHARACTER.
820   IF BK%=0 AND HM%-HP%<4 THEN HP%=HS% :VP%=VP%+SP%: '*NEW ROW
830   BK%=BK%+1               :'***INCREMENT BYTE COUNTER.
840 '
900 '***OUTPUT BYTE TO SCREEN.
910   FOR I%=0 TO 7
920     COLOR BT%(I%)         :'***SET COLOUR OF BIT.
930     SET(HP%,VP%+I%)       :'***PLOT BIT.
940   NEXT I%
950
1000 '***PREPARE FOR NEXT BYTE.
1010   HP%=HP%+1              :'***INCREMENT HORIZONTAL POSITION.
1020   IF BK%=5 THEN BK%=0 :HP%=HP%+CW%       :'***NEW CHARACTER.
1030 NEXT AD%
2000 GOTO 2000 :END
```

LISTING 1A

```
100 'THIS SHORT LISTING CAN BE USED BY OWNERS OF THE PRINTER
110 'PATCH TO CALCULATE THE START AND END LOCATIONS OF THE
120 'REVISED INVERSE CHARACTER SHAPE TABLE IN THE COMPLETED
130 'VERSION. BY SUBSTITUTING THE NEW VALUES FOR THOSE WHICH
140 'APPEAR IN LINES 640 AND 650 OF LISTING 1, THE MODIFIED
150 'CHARACTERS CAN BE DISPLAYED ON THE HIRES SCREEN.
160 '**************************************************************
170 '
180 '***CALCULATE THE TOP OF MEMORY
190 TM=PEEK(30897)+256*PEEK(30898)
200 IF TM>32767 THEN TM=TM-65536
210 '
220 '***ADD OFFSET TO TOP OF MEMORY TO LOCATE START OF TABLE
230 SM%=TM+666     :'***START OF SHAPE TABLE.
240 '
250 '***ADD 64 CHARACTERS X 5 BYTES TO LOCATE END OF TABLE
260 EM%=SM%+64*5-1 :'***END OF SHAPE TABLE
270 '
280 '***PRINT START AND END ADDRESSES
290 PRINT"START  - SM%=";SM%
300 PRINT"END    - EM%=";EM%
```

The VZ will now respond to the CLSn command, when entered, either directly from the keyboard, or from within a program, with one exception. For some unexplained reason, during IF-THEN-ELSE processing the ROM accesses the byte examine routine at 1D78H directly, instead of via a RST 10H call. This means there is no efficient method for our programme to intercept the new command, when it is used in an IF-THEN-ELSE statement. The problem can best be overcome, by means of a minor change in syntax, when entering the programme line. Using the line,

100 IF X = 4 THEN CLS4

should clear the screen to red, when X = 4.

What actually happens is that the screen clears normally, followed by a SYNTAX ERROR message, indicating the routine at 1D78H has not recognised our enhanced command. ▶

2 of 3.

LISTING 2

```
;************************
;* ENHANCED CLS COMMAND *
;* BY LARRY TAYLOR 1986 *
;************************
;
;THIS SECTION RELOCATES
;THE PROGRAM TO THE TOP
;OF AVAILABLE MEMORY.
;
VCTR EQU  7A28H          ;SET VCTR AS 7A28H
     LD   SP,7700H       ;LOAD STACK POINTER
     LD   HL,(7881H)     ;GET THE TOP OF MEMORY
     LD   BC,ENDP-NVCT   ;GET LENGTH OF PROGRAM
     PUSH BC             ;SAVE PROGRAM LENGTH
     XOR  A              ;RESET ALL FLAGS
     SBC  HL,BC          ;TAKE LENGTH FROM TOP OF MEMORY
     LD   (7881H),HL     ;LOAD NEW TOP OF MEMORY
     PUSH HL             ;SAVE NEW TOP OF MEMORY
     XOR  A              ;RESET ALL FLAGS
     LD   BC,33H         ;RESERVE 50 BYTES STRING SPACE
     SBC  HL,BC          ;TAKE SPACE FROM TOP OF MEMORY
     LD   (78A0H),HL     ;LOAD START OF STRING SPACE
     POP  DE             ;RETRIEVE TOP OF MEMORY
     INC  DE             ;INCREASE BY ONE
     LD   HL,(7804H)     ;GET CURRENT RST10H VECTOR
     LD   (VCTR),HL      ;STORE IT IN 7A28H
     LD   (7804H),DE     ;LOAD NEW VECTOR
     LD   HL,NVCT        ;GET START OF PROGRAM TO MOVE
     POP  BC             ;RETRIEVE PROGRAM LENGTH
     LDIR                ;MOVE TO NEW LOCATION
     CALL 1B4DH          ;DO A NEW
     JP   1A19H          ;JUMP TO READY MESSAGE

;
;START OF THE PROCESSING
;ROUTINE FOR NEW COMMAND.
;
NVCT EXX                 ;SAVE ALL REGISTERS
     LD   HL,1D5BH       ;CHECK TO
     POP  DE             ;SEE IF THE
     OR   A              ;RETURN
     SBC  HL,DE          ;ADDRESS
     PUSH DE             ;IS 1D5BH
     EXX                 ;RESTORE ALL REGISTERS
     JP   NZ,1D78H       ;IF NOT GO TO NORMAL PROCESSING
     PUSH HL             ;SAVE STRING ADDRESS
     CALL 1D78H          ;GET NEXT VALUE FROM STRING
     JR   NZ,CONT        ;IF NOT ZERO THEN CONTINUE
POP  POP  HL             ;ELSE RESTORE STRING ADDRESS
     LD   DE,(VCTR)      ;RETRIEVE ORIGINAL VECTOR
     PUSH DE             ;AND JUMP
     RET                 ;TO IT
CONT CP   84H            ;CHECK FOR CLS TOKEN
     JR   NZ,POP         ;IF NOT FOUND RETURN TO CALLER
     INC  HL             ;MOVE TO NEXT VALUE IN STRING
     LD   A,(HL)         ;GET NEXT VALUE AFTER CLS TOKEN
     SUB  30H            ;REDUCE IT TO RANGE 0-8
     JR   Z,EXEC         ;IF ZERO THEN EXECUTE COMMAND
     LD   B,8            ;LOAD B REG WITH UPPER LIMIT
CMPR CP   B              ;CHECK IF A=B
     JR   Z,EXEC         ;IF YES THEN EXECUTE COMMAND
     DJNZ CMPR           ;REDUCE B AND CONTINUE CHECK
     JR   POP            ;NO MATCH SO RETURN TO CALLER
EXEC POP  DE             ;RETRIEVE OLD STRING ADDRESS
     POP  DE             ;RETRIEVE OLD RETURN ADDRESS
     LD   DE,1D1EH       ;LOAD NEW RETURN ADDRESS
     PUSH DE             ;SAVE NEW RETURN ADDRESS
     INC  HL             ;MOVE TO NEXT VALUE IN STRING
     PUSH HL             ;SAVE CURRENT STRING ADDRESS
     ADD  A,A            ;MULTIPLY CLS
     ADD  A,A            ;VALUE BY 16 TO
     ADD  A,A            ;CALCULATE THE
     ADD  A,A            ;COLOUR OFFSET
     JR   NZ,SKIP        ;IF RESULT NOT ZERO THEN SKIP
     INC  A              ;IF ZERO INCREASE TO ONE
SKIP ADD  A,7FH          ;ADD 127 TO GET GRAPHICS BLOCK
;
;CLEAR SCREEN ROUTINE
;
     LD   HL,7000H       ;LOAD START OF SCREEN ADDRESS
     LD   (7820H),HL     ;SET CURSOR POSITION
     LD   DE,7001H       ;LOAD START OF SCREEN PLUS ONE
     LD   BC,01FFH       ;NUMBER OF BYTES TO MOVE
     LD   (HL),A         ;LOAD GRAPHICS BLOCK INTO HL
     LDIR                ;DO A BLOCK FILL OF THE SCREEN
     POP  HL             ;RETRIEVE STRING ADDRESS
     RET                 ;RETURN TO 1D1EH TO CONTINUE
ENDP DEFB 0              ;END OF PROGRAM MARKER
```

To have the command function properly, insert a colon between the THEN and the new command as below,

100 IF X = 4 THEN:CLS4

Now, when X = 4 the THEN part of the statement will be executed, including, as is usual, any additional commands in the remainder of the line. However, once the colon is reached, the BASIC ROM returns to its usual processing, via the RST 10H routine, and the CLS4 command is then interpreted on its own and not as part of the IF-THEN statement. This is the same solution suggested in the VZ-DOS manual, when using disk commands, which are affected in exactly the same way.

This is essentially the approach I have used to produce a VZ-EPSON Printer Patch, which enables all the normal printer functions for Epson or Epson-compatible printers. As well as providing the ability to LLIST and LPRINT all inverse and graphics characters, the COPY command is intercepted by the patch. As a result, its function has been enhanced to allow a proper dump of both the LO-RES and HI-RES screens. Corrections have been made to the flawed inverse character data, and when listing, the routine is capable of recognising all the hidden commands, which may have been entered using an Extended BASIC. The patch relocates to the top of available RAM and can be used with Steve Olney's EXTENDED BASIC, already resident in memory, enabling ready access to the functions of both. I hope that the techniques used here to produce what I have found to be an extremely useful utility will encourage others to attempt further such developments.

LISTING 3

```
100 REM ************************************************************
110 REM * ENHANCED CLS COMMAND BY LARRY TAYLOR 1986                *
120 REM ************************************************************
130 REM * CALCULATE THE NEW TOP OF MEMORY POINTER                  *
140 REM ************************************************************
150 NB=79:TM=(PEEK(30897)+PEEK(30898)*256)-NB
160 HB=INT(TM/256):LB=TM-HB*256
170 POKE30897,LB:POKE30898,HB
180 REM ************************************************************
190 REM * RESET THE BASIC STACK POINTER                            *
200 REM ************************************************************
210 CLEAR50
220 REM ************************************************************
230 REM * LOCATION OF SET UP PROGRAM                               *
240 REM ************************************************************
250 EB=31274
260 EH=INT((EB+1)/256):EL=EB+1-EH*256
270 REM ************************************************************
280 REM * LOAD USER EXECUTION PROGRAM POINTER                      *
290 REM ************************************************************
300 POKE30862,EL:POKE30863,EH
310 REM ************************************************************
320 REM * LOAD 23 BYTE SET UP PROGRAM                              *
330 REM ************************************************************
340 FOR T=1TO23
350  READD
360  POKEEB+T,D
370  CS=CS+D
380 NEXT
390 REM ************************************************************
400 REM * GET NEW TOP OF MEMORY AND MOVE TO NEXT LOCATION          *
410 REM ************************************************************
420 TM=PEEK(30897)+PEEK(30898)*256
430 IFTM>32767THENTM=TM-65536
440 REM ************************************************************
450 REM * LOAD 79 BYTE ENHANCED CLS ROUTINE                        *
460 REM ************************************************************
470 FOR T=1TO79
480  READD
490  POKETM+T,D
500  CS=CS+D
510 NEXT
520 REM ************************************************************
530 REM * IF DATA CHECKSUM VERIFIES EXECUTE SET UP PROGRAM         *
540 REM ************************************************************
550 IFCS<>10968THENPRINT"- ERROR IN DATA ENTRY -":END
560 X=USR(0)
570 REM ************************************************************
580 REM * SET UP EXECUTION ROUTINE DATA IN DECIMAL FORM            *
590 REM ************************************************************
600 DATA 243,49,0,119,42,4,120,34,40,122,42,177,120,35,34,4,120
610 DATA 205,77,27,195,25,26
620 REM ************************************************************
630 REM * ENHANCED CLS ROUTINE DATA IN DECIMAL FORM                *
640 REM ************************************************************
650 DATA 217,33,91,29,209,183,237,82,213,217,194,120,29,229,205
660 DATA 120,29,32,7,225,237,91,40,122,213,201,254,132,32,245
670 DATA 35,126,214,48,40,9,6,8,184,40,4,16,251,24,230,209,209
680 DATA 17,30,29,213,35,229,135,135,135,135.32,1,60,198,127,33
690 DATA 0,112,34,32,120,17,1,112,1,255,1,119,237,176,225,201
```

Perhaps additional enhancements to the VZ's BASIC could be explored. The Commodore 64 is served by a number of enhanced BASICs, why not the VZ? Programs which make use of such BASICs require that the language be loaded before they will function properly. However, this is little different to programs using disk commands needing the DOS to be interpreted correctly. Certainly, the opportunity exists to endow the humble VZ with a brand new bag of tricks.

For anyone interested, copies of the completed VZ-Epson Printer Patch, may be obtained on tape for $15, from:

J.C.E. D'Alton
VSOFTWAREZ
39 Agnes St
Toowong
Qld 4066

3 of 3

New life for an old VZ

Graeme Meager

Since the introduction of the VZ200 computer in early 1983 many users have been mystified by the fact that the computer did not support full level II BASIC. This article describes a method of gaining 24 extra level II BASIC commands for the VZ 200 or 300 without sacrificing any memory or software compatibility.

RECENTLY a team of enthusiasts released a revamped 16K ROM (read only memory) for the VZ with the convenience of LEVEL II BASIC on power-up and with some technical knowledge, every user can smarten up their computer.

As many users may remember, the existing ROMs were a major cause of breakdowns and possibly there are still many old VZs put away in cupboards which can be brought back to life with these new ROMs. This particular occurence prompted one user to investigate the viability of producing an EPROM to replace the original BASIC ROM. When it was discovered an EPROM was available that was pin compatible with the old 16K ROM, the task for VZ300 owners was made very simple. VZ200 owners should not dispair, with the addition of just two diodes and one resistor both 8K ROMs can be replaced by this single 16K chip.

Before entering into details of the hardware modifications, I will briefly describe the extra facilities the new ROM will provide and how they have been implemented.

THE ADDITIONAL BASIC COMMANDS:

TRON	TROFF	DELETE	AUTO
FIX	CINT	ERROR X	ERR
POS	ON	DEFINT	DEFSNG
RANDOM	MEM	ON ERROR	VARPTR
DEFDBL	RESUME	FRE	CDBL
ERL	STRINGS	DEFSTR	ON (GOTO)

Inverse characters

Owners of GP 100 and compatible printers will be familiar with the badly represented inverse character set; these errors have been corrected in the new ROM. For the owners of EPSON and compatible printers, a version of the EPROM with the modified control codes and inverse character tables is currently being compiled.

The above BASIC commands have been integrated with the original command set, which as a major consideration, enables all existing software to run unimpeded in the new system. The new ROM provides all commands without those messy loader routines, machine code calls and it is DOS (disk operating system) compatible.

The software

Statement and command execution in the VZ is by interpretation. This means that a routine dedicated to the statement type or command is called to interpret each line and perform the necessary operations. This is a common method of system command execution and is used by many other BASIC systems. Within the BASIC ROM there is a table known as the RESERVE WORD LIST. This table contains all of the words reserved for use by the BASIC interpreter.

When a line is read by the interpreter it scans this list and if the word (command) is present it will allocate a TOKEN value in the range 80 (HEX) to FB (HEX). This token will be written into memory as the BASIC command. From here on the interpreter will act on these tokens and not the original word. Each of the new commands have their own token with the allocated range and will be acted on in the same way the existing commands are. At this stage it should be noted that the original LEVEL II BASIC did not support routines for commands such as COPY, COLOR, MODE, SOUND, CRUN, CLOAD and VERIFY. These commands have used tokens originally set for other LEVEL II reserved words. The new VZ ROM actually supports more BASIC commands than the original LEVEL II ROM in the TRS-80 and SYSTEM 80 (for non-disk systems).

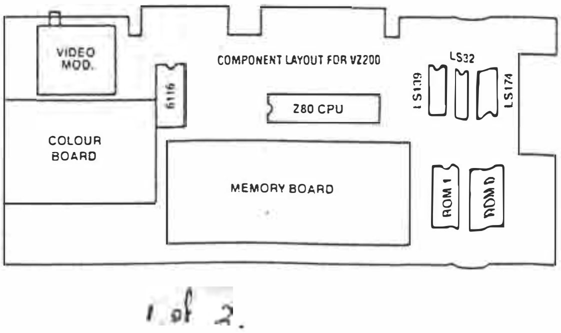
Once a value has been allocated, execution is passed to the VERB ADDRESS TABLES. Here the table is used to direct the interpreter to the routines specified by each TOKEN. There are two VERB ADDRESS TABLES: the first is used for statements that begin with a — VERB — for example END, RANDOM or PRINT. If the statement does not begin with a token, control goes to the assignment statement processing. The second table contains the addresses of verb routines which only occur on the right side of an equals sign or compliment the first verb — for example PEEK, FRE, SGN.

The new commands have been implimented by writing new values into the above tables, so the interpreter can be driected to the relevent processing routines.

As mentioned earlier, a number of areas in the ROM had to be re-organised. For example, the token 9E in the VZ ROM is allocated to the word SOUND and not the word ERROR, as originally written. Routines within the ROM had to be corrected so that when the interpreter was confronted with a format such as "ON ERROR GOSUB . . . " it would recognise the line as correct syntax.

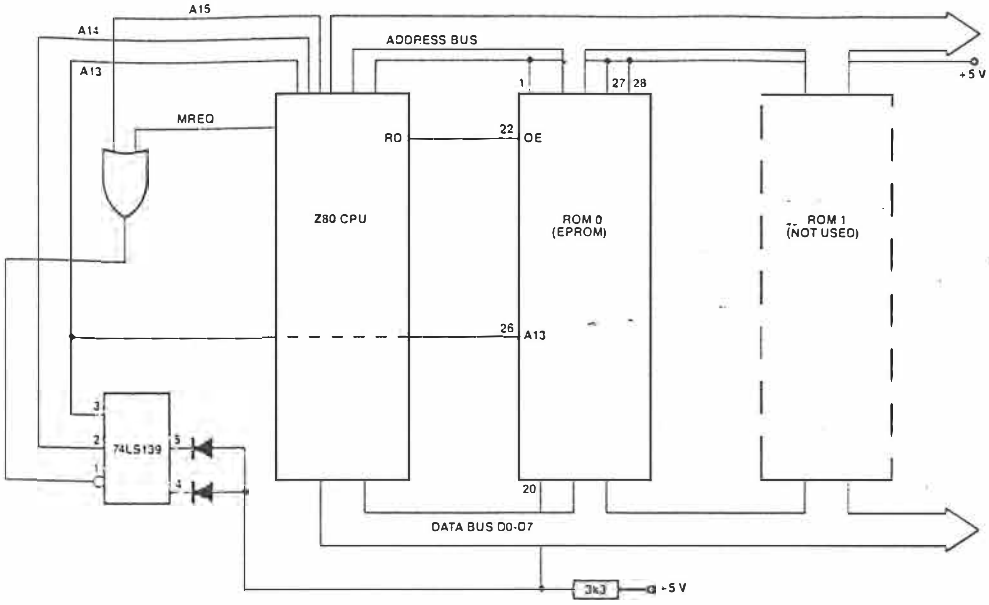
Other commands and routines are under investigation, and as they are proven compatible I understand they will be released as an update to enhance the new ROM on a change-over basis at a minimal price to purchasers. Each of the EPROMS released carry a programmed serial number to identify their generation and is apparent in the start-up header which reads as follows:

```
LASERLINK BASIC
VER. 2 #2130
READY
```



1 of 2.

VZ200 MODIFIED CIRCUIT



The hardware

Firstly, readers should be aware of the following points:

(a) any hardware modifications will void any warranty if current,
(b) this project should only be attempted by someone with reasonable soldering and desolder skills,
(c) to date, the modification has been carried out on VZ200s, both early and recent VZ300s (brown keyboard) and the LASER 200/310.

A check of compatibility with the following details should be made before commencement.

The case of the computer can be separated by removing the six screws from the bottom half. Care should be taken not to snap any of the keyboard cables. The main circuit board must then be separated by removing the screws holding it to the base. The wires to the piezo transducer will not have to be disconnected if they are long enough to rotate the board to gain access to the solder side.

The next step is to remove the RF shield by desoldering the lugs and braids attaching it to the board. For the VZ300, the diagram here should help locate the 28-pin ROM. The old ROM should be carefully desoldered and removed to be replaced by a DIL socket that is provide with the new EPROM. The unit can then be assembled and tested.

For the VZ200, two 8K ROMs can be replaced with a single 16K ROM by adding the necessary addressing circuitry and one extra memory address line. From the extract of the VZ200 circuit shown here, the 74LS139 decoder allows addressing of 000-1FFF(HEX), the first 8K ROM and 2000-3FFF(HEX) for the second 8K. These outputs need to be combined by diodes to access the full 16K. A resistor is needed to pull the chip select pin (active low) high during non-access periods. To read the full 16K, address line 13 is needed. The second diagram will help locate the two 24-pin ROMs which can be removed in the same manner. As it will be noticed, the board caters for a 28-pin socket so no extra holes are needed.

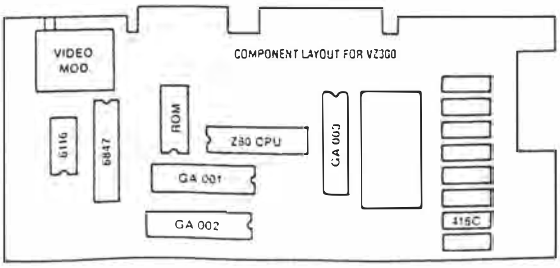
The 28-pin socket should be inserted in the position nearest the regulator heatsink. Pin 26 of the socket should be disconnected from the +5 V common with a sharp knife to cut the printed circuit track. Pin 27 should then be connected to pin 28 (+5 V). A piece of hookup wire will be needed to connect pin 26 (A13) to pin 3 of the Z80 CPU. As shown in the diagram the two diodes and the 3k3 pullup resistor can be soldered on the bottom of the board using spaghetti to insulate them from other components. The diodes are connected between pins 4 and 5 of the 74LS139 and pin 20 of the EPROM, which is in turn tied high by the 3k3 resistor.

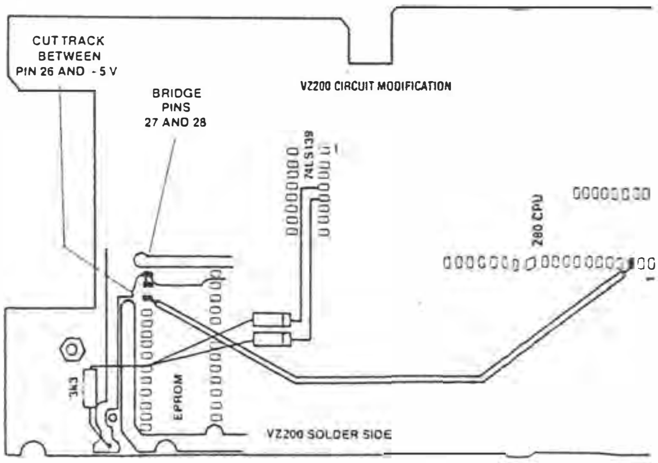
Check carefully for any solder bridges on both sides of the board, and when you are certain everything is correct, you can re-assemble and test.

At $35 (postage paid) the new EPROM is available from

LASERLINK
20 Brunker Rd
Broadmeadow 2292 NSW
(049) 62 1678

The EPROM comes complete with socket and full documentation which includes demonstration listings for each of the 24 new commands. A list of state agents can be obtained from the above address. All in all, you'll find it a worthwhile enhancement.





2 of 2.

RESTORE FILE

This is probably the most useful utility program ever made for the VZ200/300. After running out this program and typing in new, start typing in a program. Now type in new to erase the memory; type in PRINT USR(0) and *hey presto* your program is back in memory. This program is excellent if you're the type of person who gets angry with their programs.

R. Banks & M. Saunders
Mackay Qld

```
1  I=31058
10 DATA21,E9,7A,36,01,E5,CD,F8,1A,E1,7E,FE,00,28,0A,23,3E,FF,BC
20 DATA20,F5,BD,C8,20,F1,23,7E,FE,00,20,EB,23,7E,FE,00,20,E5,23
30 DATA22,F9,78,3E,00,FE,00,CD,7A,1E,C3,66,00,END
40 READA$:IFA$="END"THENPOKE30862,82:POKE30863,121:END
50 A=ASC(A$)-48:IFA>9THENA=A-7
60 B=ASC(RIGHT$(A$,1))-48:IFB>9THENB=B-7
70 POKEI,A*16+B:I=I+1:GOTO40
```

YCBB 1988 p 88.

51 bytes long program loaded into 7952 H – 7984 H (31058 – 31088)
End of BASIC statement, marked by .null. byte. End of BASIC program 2 x .null. bytes.
HL-reg used a-a PST ptr. until 3 x .null. detected. (EOS + EOB).

```
7952   21 E9 7A         LD HL,7AE9H       Set ptr. to SOB.
  55   36 01            LD (HL),1         Put non-.null. (dummy) into SOB.
  57   E5               PUSH HL           Save SOB on stack.        78A4/5
  58   CD F8 1A         CALL 1AF8 H       Line pointers routine. (set SOB ptr.)
  5B   E1               POP HL            Restore ptr.
  5C   7E          LP1  LD A,(HL)         Put byte of PST into A-reg.
  5D   FE 00            CP .null.         Is it an EOS .null. ?
  5F   28 0A            JR Z, LP2         Yes, go check on EOB. .nulls.
7961   23               INC HL            Bump ptr. to next in PST.
  62   3E FF            LD A, FFH         Check for Tom.
  64   BC               CP H              Hi byte of ptr.
  65   20 F5            JR NZ, LP1        Go back to test PST byte.
  67   BD               CP L              Lo byte of ptr.
  68   C8               RET Z             Tom (FFFFH) reached. - exit/error.?
  69   20 F1            JR NZ, LP1        Go back to test PST byte.
  6B   23          LP2  INC HL            Bump ptr. to next in PST.
  6C   7E               LD A,(HL)         Put byte of PST into A-reg.
  6D   FE 00            CP .null.         Is it first EOB .null.?
  6F   20 EB            JR NZ, LP1        Go back to test PST byte.
7971   23               INC HL            Bump ptr. to next in PST.
  72   7E               LD A,(HL)         Put byte of PST into A-reg.
  73   FE 00            CP .null.         Is it second EOB. null. ?
  75   20 E5            JR NZ, LP1        Go back to test PST byte.
  77   23               INC HL            Bump ptr. to next above PST.
  78   22 F9 78         LD (78F9 H),HL    Set EOB ptr. to HL addr.
  7B   3E 00            LD A, 00          Zero A-reg.
  7D   FE 00            CP 00             Reset Z-Flag.
  7F   CD 7A 1E         CALL 1E7A H       CLEAR routine.
7982   C3 66 00         JP 0066 H.        NMI routine, reset, IPL entry to BASIC.
```

B FILE COPIER by Rick Buhre.
(Via Dave Boyce 30\1\87)

```
10 DATA229,33,57,120,203,182,203,158,225,243,205,140,53
20 DATA229,205,177,53,33,66,56,205,244,55,205,231,53
30 DATA62,240,50,210,122,195,115,54,201
40 FORI=31067TO31101:READA:POKEI,A:NEXT
```

B FILE COPIER INSTRUCTIONS
CRUN B FILE COPIER THEN POKE31217,176:.(RETURN)
THE [redacted] PROMPT WILL APPEAR,LOAD MACHINE LANGUAGE PR
-OGRAM TO BE COPIED
WHEN READY PROMPT APPEARS POKE31067,243:POKE31068,14
(RETURN) POKE31069,241:POKE31070,195:POKE31071,172
(RETURN) POKE31072,52:POKE30884,PEEK(30750) (RETURN)
POKE30885,PEEK(30751):POKE31217,176:.'FILE NAME'(RETURN)

35 byte program to load B-file to tape.
Program loaded into RAM used for DOS vectors.

```
795B   E5         PUSH HL        Save HL reg.
  5C   21 39 78   LD HL, 7839H   Point to FLAG2
  5F   CB B6      RES 6,(HL)     Reset bit 6 to zero (CRUN flag)
  61   CB 9E      RES 3,(HL)     Reset bit 3 to zero (VERIFY flag)
  63   E1         POP HL         Restore HL reg.
  64   F3         DI             Disable interupts.
  65   CD 8C 35   CALL 358C      Pick up name.
  68   E5         PUSH HL        Save HL reg.
  69   CD B1 35   CALL 35B1
  6C   21 42 38   LD HL, 3842H   Point to 'WAITING' text
  6F   CD F4 37   CALL 37F4
  72   CD E7 35   CALL 35E7      Tape saving routine CLOAD
  75   3E F0      LD A, F0H      Auto-execute flag.
  77   32 D2 7A   LD (7AD2H), A  Buffer for cassette I/O.
  7A   C3 73 36   JP 3673        Put up 'LOADING' message.
797D.  C9         RET.           ??
```

then. reset first six bytes of program.

```
795B   F3         DI
  5C   0E F1      LD C, F1H
795E   C3 B3 34   JP 34B3        Part of CSAVE.
```

30750\1	781E/F	Part of DCB for cassette CLOAD (programme address)
30884\5	78A4/5	Start of BASIC ptr.
31217	79F1	Set to B0 in I/O buffer.

String file name

Recently I required a program to save data to a disk file on VZ300. Unfortunately, I discovered you cannot use a string as a file name and so I developed this little program. It searches through RAM to find where the program begins and then locates the disk file handling lines and stores their RAM location in an array. When a file is to be accessed it pokes the file-name into these locations. When the program begins, nothing will happen for a few seconds while the program searches for the required lines.

T. Hand,
Bentleigh, Vic

```
10 GOTO 1000
20 REM LOAD FROM FILE F$
30 GOSUB 10000:REM CHANGE FILENAME
40 REM **
50 OPEN"      ",0
60 REM **
70 IN#"      ",A,B
80 REM **
90 CLOSE"      "
100 RETURN
110 :
120 REM SAVE TO FILE F$
130 GOSUB 10000:REM CHANGE FILENAME
160 REM **
170 OPEN"      ",1
180 REM **
190 PR#"      ",A,B
200 REM **
210 CLOSE"      "
220 RETURN
230 :
240 REM ERASE FILE F$
250 GOSUB 10000:REM CHANGE FILENAME
260 REM **
270 ERA"      "
280 RETURN
290 REM ^^
300 :
310 :
320 :
330 :IT IS VERY IMPORTANT TO ENTER
340 :THE LINES WITH     REM **
350 :AS THESE ARE USED TO LOCATE THE
360 :PLACE TO CHANGE THE FILE NAME.
370 :
380 :THESE THREE ROUTINES ALSO SHOULD
390 :BE AT THE TOP OF THE PROGRAM
400 :TO SAVE TIME WHILE SEARCHING
410 :FOR THEIR LOCATION IN MEMORY.
420 :
430 :WHEN SAVING OR LOADING DATA,
440 :THE LINES WITH IN# AND PR#
450 :CAN BE CHANGED TO STORE YOUR
460 :OWN DATA
470 :
480 :
490 :
500 REM MAIN PROGRAM
1000 GOSUB 20000:REM INITIALIZE
1010 CLS
1020 PRINT "DO YOU WANT TO "
1030 PRINT "SAVE, RE-SAVE OR LOAD"
1040 A$=INKEY$:IF LEN(A$)=0 GOTO 1040
1050 IF A$="R" THEN GOSUB 2000
1060 IF A$="S" THEN GOSUB 3000
1070 IF A$="L" THEN GOSUB 4000
1080 GOTO 1010
1980 :
1990 REM RE-SAVE A FILE
2000 ER=-1
2010 GOSUB 3000:REM ENTER DATA
2020 ER=0
2030 RETURN
2900 ********************************
2910 :THIS ROUTINE CAN BE CHANGED
2920 :TO ALLOW ENTRY OF YOUR OWN
2930 :DATA.  THE ABOVE IS JUST AN
2940 :EXAMPLE OF SAVING DATA TO A
2950 :DISK FILE.
2960 ********************************
2980 :
2990 REM SAVE TO A FILE
3000 CLS
3010 INPUT"PLEASE ENTER THE FIRST VALUE";A
3020 INPUT"PLEASE ENTER THE SECOND VALUE";B
3030 GOSUB 5000
3040 IF ER THEN GOSUB 250
3050 GOSUB 130
3060 RETURN
3980 :
3990 REM LOAD FROM A FILE
4000 CLS
4010 GOSUB 5000
4020 GOSUB 30
4030 CLS
4040 PRINT "FIRST VALUE ENTERED WAS - "
4050 PRINT A
4060 PRINT "SECOND VALUE ENTERED WAS -"
4070 PRINT B
4080 A$=INKEY$:IF LEN(A$)<>0 GOTO 4080:REM CLEAR BUFFER
4090 PRINT:PRINT
4100 PRINT "PRESS SPACE BAR TO CONTINUE"
4110 A$=INKEY$:IF A$<>" " GOTO 4110:REM WAIT FOR SPACE
4120 RETURN
4980 :
4990 REM ASK FOR FILENAME
5000 CLS
5010 INPUT "PLEASE ENTER THE FILENAME";F$
5020 F1$=MID$(F$+"      ",1,6)
5030 RETURN
9980 :
9990 REM CHANGE FILE NAMES TO F$
10000 FOR I=1 TO 7
10010 IF F(I)=0 GOTO 10080
10020 C=0
10030 FOR J=1 TO LEN(F1$)
10040 POKE F(I)+C,ASC(MID$(F1$,J,1))
10050 C=C+1
10060 NEXT J
10070 NEXT I
10080 RETURN
19980 :
19990 REM INITIALIZE ROUTINE
20000 DIM F(7)
20010 C=1
20020 FOR I=31500 TO 33000
20030 IF NOT(PEEK(I)=42 AND PEEK(I+1)=42) GOTO 20080
20040 FOR J=I TO I+20
20050 IF PEEK(J)=34 THEN F(C)=J+1:C=C+1:GOTO 20080
20060 NEXT J
20070 PRINT "ERROR FINDING FILE NAMES":END
20080 IF PEEK(I)=94 AND PEEK(I+1)=94 GOTO 20100
20090 NEXT I
20100 RETURN
```

```
10 REM DISK DIRECTORY DUMPER
20 REM "BY G.TUNNY (C)OPYRIGHT 1988"
30 REM*************************
40 LPRINTCHR$(27);CHR$(21);:REM SET SINGLE LINE FEED
50 CLS:PRINT"       DISK DUMPER          ":REM INVERSE
60 INPUT"HEADING FOR DISK";H$
70 INPUT"INSERT DISK AND HIT RETURN";XZ$
80 LPRINT"----";H$;"----
85 LPRINT
90 POKE30876,1
100 STATUS
105 LPRINT
107 POKE30876,1
110 DIR
120 FORI=1TOLEN(H$)+7
130 LPRINT"-";:NEXTI
135 LPRINT"-"
140 INPUT"ANOTHER COPY";Y$
150 IFY$="YES"ORY$="Y"THENRUN
```

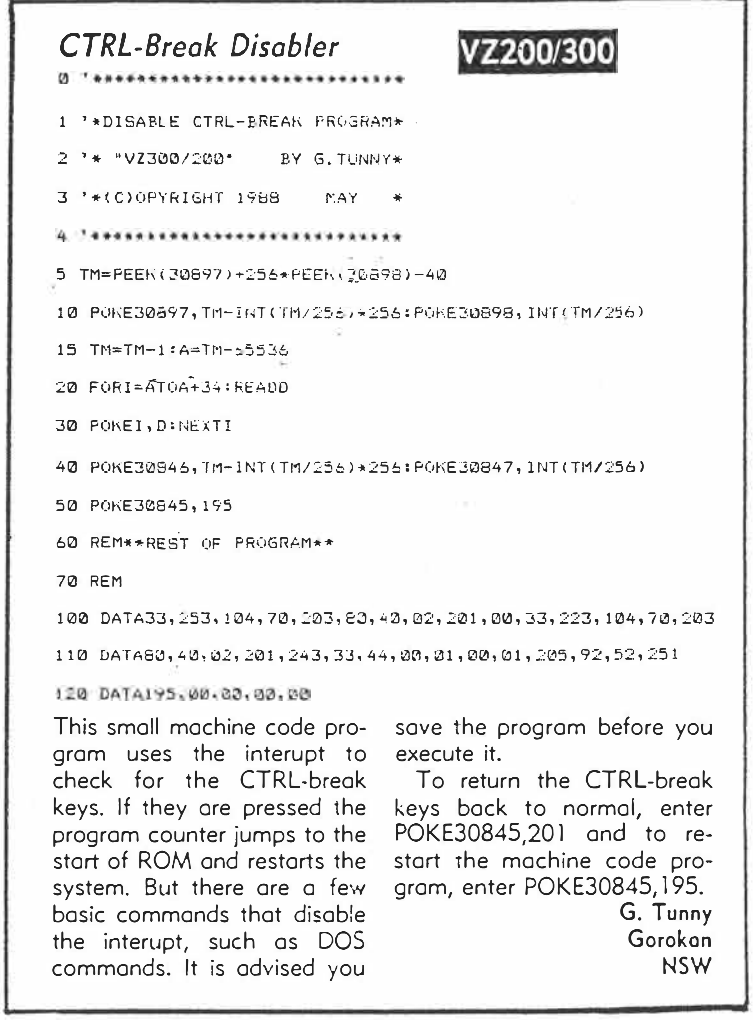
; OUTPUT DEVICE CODE
1 = PRINTER
0 = VIDEO
-1 = CASSETTE

* ESC 21.
may not work
on Epsons.

Disk Directory Dumper

This handy little program dumps the disk directory and the disc status directly on to the printer.

G. Tunny
Gorokan
NSW



CTRL-Break Disabler

VZ200/300

```
0 '*****************************
1 '*DISABLE CTRL-BREAK PROGRAM*
2 '* "VZ300/200"    BY G.TUNNY*
3 '*(C)OPYRIGHT 1988    MAY    *
4 '*****************************
5 TM=PEEK(30897)+256*PEEK(30898)-40
10 POKE30897,TM-INT(TM/256)*256:POKE30898,INT(TM/256)
15 TM=TM-1:A=TM-65536
20 FORI=ATOA+34:READD
30 POKEI,D:NEXTI
40 POKE30846,TM-INT(TM/256)*256:POKE30847,INT(TM/256)
50 POKE30845,195
60 REM**REST OF PROGRAM**
70 REM
100 DATA33,253,104,70,203,80,40,02,201,00,33,223,104,70,203
110 DATA80,40,02,201,243,33,44,00,01,00,01,205,92,52,251
120 DATA195,00,00,00,00
```

This small machine code program uses the interupt to check for the CTRL-break keys. If they are pressed the program counter jumps to the start of ROM and restarts the system. But there are a few basic commands that disable the interupt, such as DOS commands. It is advised you save the program before you execute it.

To return the CTRL-break keys back to normal, enter POKE30845,201 and to restart the machine code program, enter POKE30845,195.

G. Tunny
Gorokan
NSW

ETI OCTOBER '88
124

Set interupt exit, initiated by keyboard scanning routine 787D/E/F to JP start of program. (30845/6/7)

78B1/2. TOM ptr. 788E/F USR ptr.
30897/8. 30862/3.

```
21 FD 68    LD HL, 68FD     Row addr. k'bd. B/X/SHFT/C/Z/V
46          LD B,(HL)       Load matrix into B reg.
CB 50       BIT 2, B.       Bit 2 is SHFT key.
28 02       JR Z, 02        If zero then SHFT key depressed.
C9          RET                ... else return.
00          NOP
21 DF 68    LD HL, 68DF     Row addr. k'bd. G/S/CTRL/D/A/F
46          LD B, (HL)      Load matrix into B reg.
CB 50       BIT 2, B        Bit 2 is CTRL key.
28 02       JR Z, 02        If zero then CTRL key depressed.
C9          RET                .... else return.
F3          DI              Disable interupts.
21 2C 00    LD HL, 002C     Set HL (Freq.) to 44D
01 00 01    LD BC, 0001     Set BC (duration) to 1D.
CD 5C 34    CALL 345C       Sound routine.
FB          EI              Enable interupts.
C3 00 00    JP 0000.        Cold start computer.
00 00       NOP's.
```

(35 bytes)

VZBUG – A useful program for memory related work on the VZ200 or VZ300

Have you ever wanted to look inside a VZ memory chip? There are two ways to do this. The first is to get a hacksaw and cut the chip in half. The second method is to use VZBUG. We think you'll find VZBUG much more informative than the hacksaw.

ONE OF THE DISADVANTAGES of the modern home computer is that the user never really gets the opportunity to get into the guts of the machine. Most of the time the small home micro is in BASIC mode, and the user doesn't have any idea why the computer does what it does. VZBUG remedies this by letting you get into the "nitty-gritty" of your VZ's insides.

VZBUG is ideal for fixing jammed programs, or for other memory related work. In addition, you can use VZBUG for loading and saving data onto cassettes, clearing the screen, typing text into memory and printing it – a mini word processor!

Once you have VZBUG installed you will wonder how you ever got on without it.

Functions

There are seven main functions in VZBUG. All numbers are entered from the keyboard in hexadecimal. The functions are called after the program is loaded with the following commands:

C – Clear screen
G – Goto memory location and execute program
I – Insert ASCII into memory
L – Load from cassette
D – Display memory location
O – Output memory
S – Save to cassette

To terminate the program and return to BASIC, simply enter **G1A19**, which translates to "goto HEX 1A19 and execute". 1A19 is the return-to-BASIC address contained in the VZ ROMs.

Clear screen – just type "C" and the screen clears, returning the prompt character to the top left hand corner of the screen.

Goto – type "G" and the computer will ask you for a memory location. Enter the location in HEX and the computer will jump to that location and execute what i there. If there is not a valid program at that address the computer might lock up, so be careful.

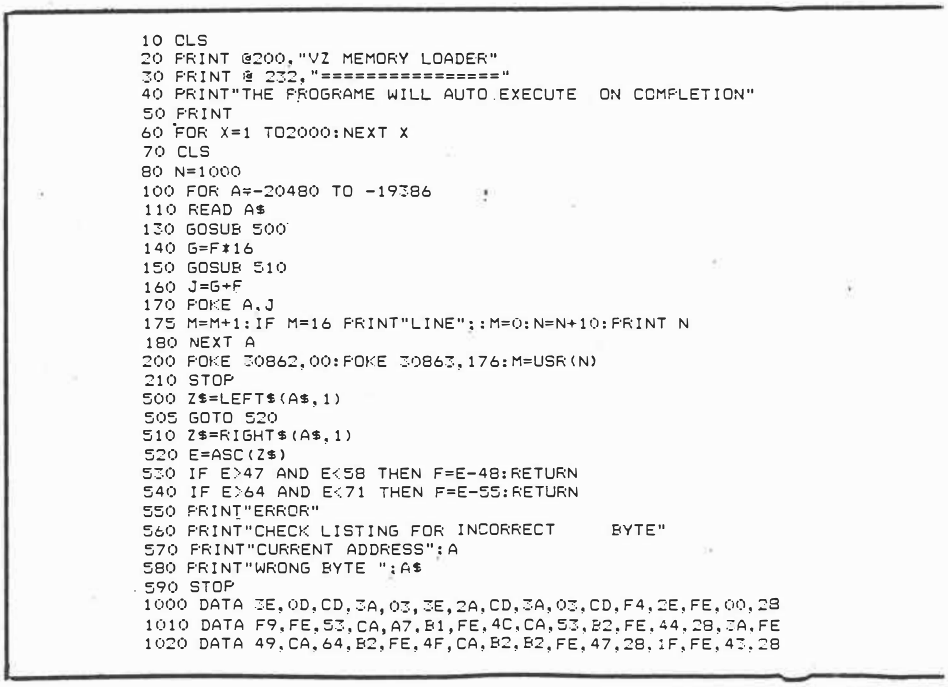
Insert ASCII into memory – type "I" and an asterisk will appear on the screen. Enter the start address (again in HEX), and start typing. This is in effect a mini word processor. To exit the command and return to the VZBUG command loop, simply type CTRL "E".

Load cassette – typing "L" will result in the word "WAITING" will appear on the screen. Press PLAY on the cassette player and the next program on the tape will be loaded, in the same manner as the BASIC CRUN command. CTRL BREAK will terminate the load and return you to BASIC.

Display and alter memory – this command allows you to display and alter any memory address in the VZ RAM area. Type "D" followed by the address you wish to access, e.g. **DCF00** will display the contents of memory location CF00. If you wish to change the contents, simply type in the new data, in HEX of course. If the data typed is O.K., press RETURN to proceed to the next memory byte. To return to the VZBUG command loop, simply type "N".

Output memory – there are four different ways of accessing the VZ's memory with this command. They are:

"Output to printer in ASCII" - This prints out the contents of the locations selections on your printer in ASCII format. This is used to print out text created with the "I" command. The output is terminated by the HEX byte "00", which is the terminating character of the "I" command.



```
10 CLS
20 PRINT @200,"VZ MEMORY LOADER"
30 PRINT @ 232,"================"
40 PRINT"THE PROGRAME WILL AUTO.EXECUTE  ON COMPLETION"
50 PRINT
60 FOR X=1 TO2000:NEXT X
70 CLS
80 N=1000
100 FOR A=-20480 TO -19386
110 READ A$
130 GOSUB 500
140 G=F*16
150 GOSUB 510
160 J=G+F
170 POKE A,J
175 M=M+1:IF M=16 PRINT"LINE";:M=0:N=N+10:PRINT N
180 NEXT A
200 POKE 30862,00:POKE 30863,176:M=USR(N)
210 STOP
500 Z$=LEFT$(A$,1)
505 GOTO 520
510 Z$=RIGHT$(A$,1)
520 E=ASC(Z$)
530 IF E>47 AND E<58 THEN F=E-48:RETURN
540 IF E>64 AND E<71 THEN F=E-55:RETURN
550 PRINT"ERROR"
560 PRINT"CHECK LISTING FOR INCORRECT     BYTE"
570 PRINT"CURRENT ADDRESS":A
580 PRINT"WRONG BYTE ";A$
590 STOP
1000 DATA 3E,0D,CD,3A,03,3E,2A,CD,3A,03,CD,F4,2E,FE,00,28
1010 DATA F9,FE,53,CA,A7,B1,FE,4C,CA,53,B2,FE,44,28,3A,FE
1020 DATA 49,CA,64,B2,FE,4F,CA,B2,B2,FE,47,28,1F,FE,43,28
```

1 of 2.

"Output to printer in HEX" - This prints out the contents of selected locations on your printer in HEX code. Only 256 bytes are printed and then the program stops, displaying a "?" prompt on the screen. Press RETURN to print out the next 256 bytes or "E" to return to the VZBUG loop.

"Output to screen in ASCII" – Same as the first option, but the output is directed to the screen, not the printer.

"Output to screen in HEX" – Same as the second option, but output is directed to the screen and blocks of 16 bytes are displayed at a time. To return to command loop, press "N".

These options are selected with the following command line parameters:
Select O for output, then:

S/P	to select Screen or Printer output, enter start address in HEX,
H/A	to select HEX or ASCII format.

e.g. to display address B000 on the screen in HEX,
type O,S,B000,H

Save on cassette – this command allows you to save a block of memory to cassette. Type "S" followed by the name you wish to allocate to the block (14 characters maximum). CTRL "E" finishes the entry of the file name. You must also enter the start and end addresses of the block and then select either "B" or "A", depending on whether you want the block saved as a load-only or auto-execute routine. The "B" parameter saves the program as load-only, whereas using the "A" parameter will create an auto-executing file. If you use the "A" parameter, be certain that the start address is a valid execute address, or the computer may lock up.

Getting VZBUG going

VZBUG is loaded as a BASIC program shown in the accompanying listing. I would strongly suggest that you enter the program in a number of stages, saving your work progressively. Take your time – maybe you should consider entering the data in two or three sittings, rather than a single eye-blurring, mind-boggling session.

Before you run the program initially, SAVE IT to cassette. As is always the case with machine-language-loading BASIC programs, a single error in entering the DATA statements can result in a computer lock-up, and the loss of all data in memory.

When the program is loaded it pokes into memory all the HEX code contained in the DATA statements at the end of the listing. It also checks to see if you have accidentally entered a non-HEX byte, and if so displays the address and contents of the incorrect byte. You can use this to locate and correct the error, by comparing the listings.

If you enter an incorrect but nevertheless valid HEX byte, the program will not trap it, and it may cause lock-up, so proceed slowly and carefully.

The program occupies addresses B000 to B447. It cannot be moved as it contains absolute addresses. I am prepared to supply reassembled programs at a different address, if you drop me a line at *my* address (see end of article), including a blank cassette and cheque/money order for $10.

Useful subroutines

Here are some additional useful subroutines I have implemented in VZBUG for users.

Executing hexadecimal address B151 instructs the computer to accept either two or four bytes from the keyboard, convert them to HEX and store them at HEX CFFA/B. The size of the input, two or four bytes, is determined by the check byte located at CFFF. If the check byte is HEX AB, then two characters will be accepted. Any other data will allow four bytes to be accepted.

Calling address B19F converts HEX to ASCII, and is used to display HEX data on the screen. The value to be converted is the one resident in the accumulator, after conversion is completed, the converted value is held in the accumulator.

Location B42F contains a routine to convert ASCII input from the keyboard into HEX. As with address B19F, the accumulator is used for both the original and converted values. The D and E registers are also used for this.

Besides these useful subroutines, there are many more contained in the VZ ROMs. Included with the assembler tape from Dick Smith Electronics is a full listing of the useful VZ subroutines.

Ready set go!

Now is the time to roll up your sleeves, polish your glasses, take the phone off the hook, and enter in the VZBUG listing. REMEMBER – take it easy, be careful, double and triple check, and **save before you run.** *HAPPY COMPUTING!*

Reg Batger
13 Hillview Rd,
Kellyville 2153 NSW

```
1130 DATA CB,3F,CB,3F,CB,3F,CD,2F,B4,CD,3A,03,4F,3A,F2,CF
1140 DATA E6,0F,CD,2F,B4,CD,3A,03,C9,3E,20,CD,3A,03,3E,00
1150 DATA 32,FF,CF,CD,F4,2E,FE,00,28,F9,FE,0D,28,29,FE,4E
1160 DATA CA,00,B0,3E,AB,32,FF,CF,CD,51,B1,3A,F6,CF,CD,9F
1170 DATA B1,CB,27,CB,27,CB,27,CB,27,47,3A,F7,CF,CD,9F,B1
1180 DATA 80,ED,5B,F0,CF,12,2A,F0,CF,23,22,F0,CF,CD,38,B1
1190 DATA 3E,0D,CD,3A,03,C3,B2,B0,CD,50,34,CD,50,34,CD,50
1200 DATA 34,CD,50,34,CD,50,34,CD,50,34,CD,50,34,CD,50,34
1210 DATA C9,DD,21,F4,CF,DD,22,F4,CF,CD,F4,2E,FE,00,CA,59
1220 DATA B1,11,36,B4,47,1A,B8,CA,72,B1,FE,FF,CA,59,B1,13
1230 DATA 18,F3,CD,3A,03,DD,2A,F4,CF,DD,77,02,DD,23,DD,22
1240 DATA F4,CF,CD,38,B1,3A,FF,CF,FE,AB,CA,96,B1,3A,F4,CF
1250 DATA FE,F8,C8,C3,59,B1,3A,F4,CF,FE,F6,C8,C3,59,B1,DE
1260 DATA 30,FE,0A,F8,DE,07,C9,CD,38,B1,3E,0D,CD,3A,03,3E
1270 DATA 4E,CD,3A,03,3E,41,CD,3A,03,3E,4D,CD,3A,03,3E,45
1280 DATA CD,3A,03,3E,2D,CD,3A,03,DD,21,D0,CF,3E,22,DD,77
1290 DATA 00,DD,23,DD,22,E0,CF,CD,87,B2,3E,22,DD,77,00,3E
1300 DATA 0D,CD,3A,03,CD,ED,B1,CD,FC,B1,C3,0B,B2,3E,53,CD
1310 DATA 3A,03,CD,6B,B0,2A,FA,CF,22,A4,78,C9,3E,45,CD,3A
1320 DATA 03,CD,6B,B0,2A,FA,CF,22,F9,78,C9,3E,42,CD,3A,03
1330 DATA 3E,20,CD,3A,03,3E,41,CD,3A,03,3E,20,CD,3A,03,3E
1340 DATA 3F,CD,3A,03,CD,F4,2E,FE,00,28,F9,FE,42,CA,45,B2
1350 DATA FE,41,20,F0,3E,0D,CD,3A,03,21,D0,CF,0E,F1,F3,CD
1030 DATA 16,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1040 DATA 00,00,00,00,00,18,C3,CD,C9,01,18,B4,CD,3A,03,CD
1050 DATA 38,B1,CD,6B,B0,2A,FA,CF,E9,CD,3A,03,CD,38,B1,CD
1060 DATA 6B,B0,2A,FA,CF,22,F0,CF,C3,B2,B0,CD,51,B1,3E,0D
1070 DATA CD,3A,03,3A,F8,CF,CD,9F,B1,CB,27,CB,27,CB,27,CB
1080 DATA 27,32,FA,CF,3A,F9,CF,CD,9F,B1,47,3A,FA,CF,80,32
1090 DATA FA,CF,3A,F6,CF,CD,9F,B1,CB,27,CB,27,CB,27,CB,27
1100 DATA 32,FD,CF,3A,F7,CF,CD,9F,B1,47,3A,FD,CF,80,32,FB
1110 DATA CF,C9,2A,F0,CF,7C,CD,C9,B0,7D,CD,C9,B0,3E,20,CD
1120 DATA 3A,03,7E,CD,C9,B0,C3,E9,B0,32,F2,CF,E6,F0,CB,3F
```

```
1360 DATA AC,34,C3,00,B0,3E,0D,CD,3A,03,21,D0,CF,CD,A9,34
1370 DATA C3,00,B0,3E,22,32,FA,CF,32,FB,CF,21,FA,CF,CD,5F
1380 DATA 36,C3,00,B0,3E,0D,CD,3A,03,CD,38,B1,CD,ED,B1,DD
1390 DATA 2A,A4,78,DD,22,E0,CF,CD,87,B2,3E,00,DD,77,00,3E
1400 DATA 0D,DD,77,01,C3,00,B0,3E,08,CD,3A,03,CD,F4,2E,FE
1410 DATA 00,CA,8C,B2,FE,87,C8,CD,3A,03,DD,2A,E0,CF,DD,77
1420 DATA 00,DD,23,DD,22,E0,CF,CD,38,B1,3E,A0,CD,3A,03,C3
1430 DATA 87,B2,CD,3A,03,3E,00,32,E2,CF,CD,38,B1,3E,0D,CD
1440 DATA 3A,03,3E,53,CD,3A,03,3E,2F,CD,3A,03,3E,50,CD,3A
1450 DATA 03,CD,F4,2E,FE,00,28,F9,FE,50,28,06,FE,53,28,05
1460 DATA 18,EF,32,E2,CF,3E,0D,CD,3A,03,CD,ED,B1,3E,48,CD
1470 DATA 3A,03,3E,2F,CD,3A,03,3E,41,CD,3A,03,CD,F4,2E,FE
1480 DATA 00,28,F9,FE,48,CA,25,B3,FE,41,CA,0F,B3,18,ED,3E
1490 DATA 0D,CD,3A,03,2A,A4,78,3A,E2,CF,FE,50,CA,0F,B4,CD
1500 DATA 75,2B,C3,00,B0,3E,0D,CD,3A,03,3A,E2,CF,FE,50,28
1510 DATA 51,3A,A5,78,CD,63,B3,3A,A4,78,CD,63,B3,3E,0D,CD
1520 DATA 3A,03,2A,A4,78,06,10,7E,CD,63,B3,23,10,F9,22,A4
1530 DATA 78,CD,F4,2E,FE,00,28,F9,FE,0D,28,D5,FE,4E,CA,00
1540 DATA B0,18,EE,32,F0,CF,E6,F0,CB,3F,CB,3F,CB,3F,CB,3F
1550 DATA CD,2F,B4,CD,3A,03,3A,F0,CF,E6,0F,CD,2F,B4,CD,3A
1560 DATA 03,C9,3E,10,32,E4,CF,3A,A5,78,4F,CD,C4,B3,3A,A4
1570 DATA 78,4F,CD,C4,B3,0E,20,CD,8D,05,3E,10,32,E9,CF,3A
1580 DATA E8,CF,FE,00,28,56,3D,32,E8,CF,2A,A4,78,7E,23,22
1590 DATA A4,78,CD,C4,B3,0E,20,CD,8D,05,18,E3,0E,0A,CD,8D
1600 DATA 05,C3,00,B0,32,F0,CF,E6,F0,CB,3F,CB,3F,CB,3F,CB
1610 DATA 3F,CD,2F,B4,4F,CD,8D,05,3A,F0,CF,E6,0F,CD,2F,B4
1620 DATA 4F,CD,8D,05,C9,3E,3F,CD,3A,03,CD,F4,2E,FE,00,28
1630 DATA F9,FE,0D,28,8D,FE,45,CA,00,B0,18,EE,0E,0A,CD,8D
1640 DATA 05,3A,E4,CF,3D,FE,00,28,DC,32,E4,CF,C3,87,B3,22
1650 DATA E6,CF,7E,FE,00,CA,00,B0,FE,0D,CA,27,B4,4F,CD,8D
1660 DATA 05,2A,E6,CF,23,18,E8,3E,0A,4F,CD,8D,05,18,F2,11
1670 DATA 36,B4,83,5F,1A,C9,30,31,32,33,34,35,36,37,38,39
1680 DATA 41,42,43,44,45,46,FF
```

2 of 2.

Clock

This is another of my interrupt controlled programs for all you VZ owners out there. This machine code program could be put into games as an accurate timer. Because this program does not depend on basic, it will not lose track of time when you break out of the program. There are only a few commands that will make it lose a second or two, such as DOS or sound commands.

The storage locations used for the seconds, minutes and hours are written in the Basic program and can be poked to change them. It is advisable to save this program before you run it because machine code has a nasty habit of crashing.

G. Tunny
Gorokan
NSW

```
1 '****************
2 '*   CLOCK      *
3 '* BY G.TUNNY   *
4 '* (C)OPYRIGHT *
5 '*  JULY 1988   *
6 '****************
10 DATA33,192,121,53,192,54,60,58,197,121,60
20 DATA254,60,40,4,50,197,121,201,33,197
30 DATA121,54,0,58,194,121,60,254,60,40
40 DATA4,50,194,121,201,33,194,121,54,0
50 DATA1,0,1,33,42,0,205,92,52,58
60 DATA195,121,60,254,13,40,4,50,195,121
70 DATA201,33,195,121,54,1,201,0,0          70 bytes.
100 TM=PEEK(30897)+256*PEEK(30898)-~~20~~ 70
110 POKE30897,TM-INT(TM/256)*256:POKE30898,INT(TM/256)
120 TM=TM-1:A=TM-65536
130 FORI=0TO68
140 READD:POKEI+A,D
150 NEXTI
160 POKE30846,TM-INT(TM/256)*256:POKE30847,INT(TM/256)
170 POKE30845,195
200 CLS
210 S=31173:' STORAGE LOCATION FOR SECONDS
220 M=31170:' STORAGE LOCATION FOR MINUTES
230 H=31171:' STORAGE LOCATION FOR HOURS
240 PRINT"**ENTER CURRENT TIME**"
250 PRINT:INPUT"MINUTES";A:POKEM,A
260 INPUT"HOURS";A:POKEH,A
270 CLS
280 PRINT@20,"SECONDS",PEEK(S);" "
290 PRINT@0,PEEK(H);":";PEEK(M);"  "
300 GOTO 280
```

ETI NOVEMBER '88
120

1 of 2.

A very interesting application of interrupt use.

The interrupt vector 30845/6/7 or 787D/E/F H is "stolen" and used to enter the machine language routine detailed on next sheet. 787DH is set to RETurn during initialization at 3E37H. It is CALLed by the Interrupt Service Routine at 2EBCH. every 20 m.sec.

The interrupt is called 50 times per second. A critical value for timing, may need altering to maintain correct time.

Note that on the hour, a beep is made.

Four temporary registers are used in the Comms. Area. These may cause problems. in some applications. They are

31170	79C2 H	–	MIN.
31171	79C3 H	–	HOUR.
31168	79C0H	–	KOUNT.
31173	79C5H	–	SEC.

CLOCK routine   ETI Nov 88. p 120.

```
21 C0 79    LD HL, KOUNT     Point HL at interrupt counter             Count down secs.
35          DEC (HL)         Decrement counter
C0          RET NZ           Return to mainline if not zero, else continue to set time.
36 3C       LD (HL), 60      Reset counter. (critical value)           Sec. routine
3A C5 79    LD A, (SEC)      Put SEC into A
3C          INC A            Increment A.
FE 3C       CP 60            Compare with 60.
28 04       JR Z, 4          If zero go to min. routine, else continue.
32 C5 79    LD (SEC), A      Reset SEC
C9          RET              Return to mainline.
21 C5 79    LD HL, SEC ←     Point HL at SEC                           Min. routine.
36 00       LD (HL), 0       Set SEC to zero
3A C2 79    LD A, (MIN)      Put MIN into A
3C          INC A            Increment A
FE 3C       CP 60.           Compare with 60.
28 04       JR Z, 4          If zero go to hour routine, else continue.
32 C2 79    LD (MIN), A      Reset MIN
C9          RET              Return to mainline.
21 C2 79    LD HL, MIN ←     Point HL at MIN                           Hour routine.
36 00       LD (HL), 0       Set MIN to zero.
01 00 01    LD BC, 256       Set duration to 256.
21 2A 00    LD HL, 42        Set tone to 42
CD 5C 34    CALL 345CH       Sound beep every hour.
3A C3 79    LD A, (HOUR)     Put HOUR into A
3C          INC A            Increment A
FE 0D       CP 13            Compare with 13.
28 04       JR NZ, 4         If zero go to hour reset routine, else continue.
32 C3 79    LD (HOUR), A     Reset HOUR.
C9          RET              Return to mainline.
21 C3 79    LD HL, HOUR ←    Point HL at HOUR                          Hour reset routine.
36 01       LD (HL), 1       Set HOUR to one.
C9          RET              Return to mainline.                       RBM. 11/88
00 00       NOP's.
```

2 of 2.

Hello program

This hello program loads the directory onto the screen and conveniently allows the user to load, run or erase programs without typing lengthy filenames.

If there are any filenames that you don't want to come up in the hello program, rename the filenames to have an asterisk at the front.
e.g. a file – 'picture' becomes (*Picture).

G Tunny
Gorokan
NSW

ETI FEBRUARY '89
118-9

```
1 GOTO1800
3 REM!!
4 LOAD"          "
5 REM!!
6 BLOAD"          "
10 REM!!
20 RUN"          "
30 REM!!
40 BRUN"          "
50 REM!!
60 ERA"          ":RUN
90 REM
91 '************************
92 '* DOS "HELLO" PROGRAM *
93 '*      VZ300/200       *
94 '*  WRITTEN BY G.TUNNY *
95 '*   (C)OPYRIGHT 1988   *
96 '*       NOVEMBER       *
97 '************************
100 T=PEEK(30884)+256*PEEK(30885)
120 REM
130 T=T+1
135 IFT>32767THENT=T-65536
140 P=PEEK(T)
150 IFP=33ANDPEEK(T+1)=33THEN200
160 IFP=94ANDPEEK(T+1)=94THENRETURN
170 GOTO130
200 T=T+1
210 IFPEEK(T)=34THEN230
220 GOTO200
230 T=T+1
240 FORI=1TOLEN(F$)
250 C$=MID$(F$,I,1)
260 C=ASC(C$)
270 POKET,C
280 T=T+1
290 NEXTI
300 IFL=7THEN160
310 FORI=LTOL
320 POKET,32:T=T+1
330 NEXTI
340 GOTO160
1000 T=28672:C=65:A=1
1005 P=PEEK(T):F(A)=P
1010 POKET,C:POKET+1,93
1020 T=T+32:C=C+1:A=A+1
1030 P=PEEK(T):IFP=96THEN1050
1040 GOTO1005
1050 DIMF$(20):DIMNF$(20)
1060 T=28673:C=1:F=1
1065 F$(C)=""
1070 FORJ=1TO8:IFPEEK(T+J)=96THENNEXTJ:GOTO1100
1075 P=PEEK(T+J):IFP>95THENP=P-64
1080 F$(C)=F$(C)+CHR$(P)
1090 NEXTJ
1100 C=C+1:T=T+32:FN=FN+1
1110 IFPEEK(T)=96THEN1150
1120 GOTO1065
1150 GOSUB2100
1200 FORI=1TOLEN(B$):A$=INKEY$:A$=INKEY$
1210 PRINT@448,MID$(B$,I,30)
1220 X=USR(X)
1230 IFA$=""THENNEXTI:GOTO1200
1235 IFA$="1"THENLD=1:GOSUB1900
1237 IFA$="2"THENLD=2:GOSUB2000
1240 A=ASC(A$):IFA<65ORA>90THEN1200
1250 A=A-64
1255 IFA>FNTHENSOUND31,1:GOTO1200
1260 F$=F$(A)
1270 IFF(A)=68,CLS:PRINTTAB(6);"CANNOT LOAD DATA FILE!":GOTO1500
1280 GOSUB100
1283 IFLD=1ANDF(A)=84THEN3
1285 IFLD=1ANDF(A)=66THEN5
1287 IFLD=2THEN60
1290 IFF(A)=84THEN10
1300 IFF(A)=66THEN30
1310 CLS:PRINT"     IDENTIFICATION ERROR!!"
1320 STOP
1500 FORI=1TO3:SOUND31,1;24,2;17,4:NEXTI
1510 RUN
1800 CLEAR1500:F$="FILENAME":POKE30862,80:POKE30863,52
1810 CLS:DIR:DIMF(20)
1820 B$="..............................TYPE LETTER TO RUN....1-"
1830 B$=B$+"LOAD....2-ERASE....WRITTEN BY G.TUNNY (C)OPYRIGHT "
1835 B$=B$+"1988........"
1840 GOTO1000
1900 IFINKEY$<>""THEN1900
1910 PRINT@480,"  TYPE LETTER TO LOAD    ";
1920 IFINKEY$=""THEN1920
1930 RETURN
2000 IFINKEY$<>""THEN2000
2010 PRINT@480,"  TYPE LETTER TO ERASE   ";
2020 IFINKEY$=""THEN2020
2030 RETURN
2100 K=FN:FORI=1TOFN
2110 L$=LEFT$(F$(I),1)
2120 L=ASC(L$)
2130 IFL=42THENNEXTI:GOTO2200
2140 Z=Z+1:NF$(Z)=F$(I):F(Z)=F(I)
2150 X=USR(X)
2160 NEXTI:GOTO2200
2200 SC=2:Y=1:FN=Z
2210 PRINT@SC,NF$(Y);"
2220 IFY=FNTHEN2250
2230 Y=Y+1:SC=SC+32
2240 GOTO2210
2250 FORI=1TOFN
2260 F$(I)=NF$(I)
2270 NEXTI
2280 FORI=FNTO14
2290 PRINT@I*32,"
2300 NEXTI:RETURN
```

VZ 200/300

Visisort

This program implements eight sort techniques at selectable speeds of which O is the fastest. Sort data can be either letters or numbers which can be chosen by the computer or the user. The program is approximately 5.6 k long and runs on the VZ 200/300, but not on the unexpanded VZ 200. Instructions to use it are contained in the program.

PJ Sheppard
Christchurch
New Zealand

ETI FEBRUARY '89
119 - 120

pg 1 of 2.

```
40 REM         VISISORT
50 REM       FOR VZ200/300
60 REM     BY P.J.SHEPPARD
80 CLS
100 PRINT@200,"--< VISISORT >--"
105 ' INITIALISE VARIABLES, ETC
110 CLEAR500:DIMS9(20,2),A$(13),S$(13):X1=400:X2=25:Q$=CHR$(34
120 PP=36:N1=154:N2=250:N3=186:N6=411:A1=30:A2=36:A3=36
130 SOUND0,9
140 CLS
150 PRINT:PRINT"       * V I S I S O R T *"
160 PRINT"    -------------------":PRINT
170 PRINT" (1)..STANDARD BUBBLE SORT
180 PRINT" (2)..BUBBLE SORT WITH SINKER
190 PRINT" (3)..SUPER BUBBLE SORT
200 PRINT" (4)..EXCHANGE SORT
210 PRINT" (5)..DELAYED REPLACEMENT SORT
220 PRINT" (6)..SHELL SORT
230 PRINT" (7)..SHELL - METZNER SORT
240 PRINT" (8)..QUICK - SORT
250 PRINT" (9)..EXIT PROGRAM
260 PRINT@451,"SELECT SORT ROUTINE ...";
265 D$=INKEY$
270 FORJ=0TO0:CH$=INKEY$:J=((CH$<"1")OR(CH$>"9")):NEXT
280 CH=VAL(CH$):IFCH=9THEN2320ELSEPRINTCH:SOUND30,7;0,2
290 CLS
300 GOSUB430
310 FORK=0TONS:A$(K)=S$(K):PRINT@32*K+32,A$(K);:NEXT
320 NE=0:NC=0:NO=0:NI=186
330 IFNS>9THENNI=184
335 ' SORT STATUS
340 PRINT@76,"NO OF ITEMS .. "USING"##";NS+1
350 PRINT@108,"------------------";
360 PRINT@140,"COMPARISONS .   0";
370 PRINT@172,"EXCHANGES ...   0";
380 PRINT@204,"------------------";
390 PRINT@236,"TOTAL ACTIONS   0";
400 PRINT@268,"------------------";
405 ' SORT ROUTINES
410 IFCH=1 GOSUB1210ELSEIFCH=2 GOSUB1480ELSEIFCH=3 GOSUB1330
414 IFCH=4 GOSUB1590ELSEIFCH=5 GOSUB1690ELSEIFCH=6 GOSUB1840
418 IFCH=7 GOSUB1970ELSEIFCH=8 GOSUB2120
420 GOSUB860
430 PRINT@300,"******************";
440 PRINT@332,"* SORT COMPLETE *";
450 PRINT@364,"******************";
460 PRINT@423,"RESORT ORIGINAL";
470 PRINT@461,"LIST......(Y/N)";
475 D$=INKEY$
480 FORJ=0TO0:RF$=INKEY$:J=((RF$<>"Y")AND(RF$<>"N")):NEXT:GOTO140
485 SOUND30,1
490 IFRF$="Y"THEN750
495 ' SORT DATA
500 FF=0:NS$=""
510 PRINT"HOW MANY ITEMS TO SORT"
520 PRINT"SELECT BETWEEN 4 AND 14 -- ";
525 D$=INKEY$::SS$=""
530 SS$=INKEY$:SOUND0,1
535 IFSS$=""THEN525ELSESOUND30,1
540 IFSS$="1"ANDNS$<"1"THENNS$=NS$+SS$:FF=1:PRINTSS$;:GOTO530
550 IFFF=1AND(SS$<"0"ORSS$>"4")ORFF=0AND(SS$<"4"ORSS$>"9"),530
560 PRINTSS$:NS$=NS$+SS$:NS=VAL(NS$)-1
570 PRINT:PRINT"SUPPLIED BY COMPUTER OR USER ?"
580 PRINT"PRESS "Q$"C"Q$" OR "Q$"U"Q$"...";
585 D$=INKEY$:RI$=INKEY$
590 IFRI$="U"THEN680ELSEIFRI$="C"THENPRINT"COMPUTER"ELSE585
595 ' COMPUTER DATA
600 PRINT:PRINT"NUMBERS OR LETTERS?"
610 PRINT"PRESS "Q$"N"Q$" OR "Q$"L"Q$"...";
615 D$=INKEY$
620 FORJ=0TO0:R$=INKEY$:J=((R$<>"N")AND(R$<>"L")):NEXT
630 IFR$="N"THENPRINT"NUMBERS"ELSEIFR$="L"THENPRINT"LETTERS"
635 SOUND0,1
640 FORK=0TONS
650 IFR$="L"THENS$(K)=CHR$(RND(26)+64)ELSES$(K)=STR$(RND(89)+10)
660 NEXTK
670 GOTO750
675 ' USERS DATA
680 CLS
690 PRINT"ONE CHARACTER PER LINE MAXIMUM"
700 PRINT"(LETTERS OR NUMBERS ONLY)"
710 FORK=0TONS
720 PRINT"ITEM #"K"= ";
725 D$=INKEY$:FORJ=0TO0:S$(K)=INKEY$
730 J=(S$(K)<"0")OR("Z"<S$(K))OR(("9"<S$(K))AND(S$(K)<"A"))
735 NEXT
740 PRINTS$(K):SOUND28,1:NEXTK
750 RF$="N"
760 CLS
770 RETURN
780 PRINT@336,"PRESS <RETURN> TO";
790 PRINT@428,"START THE SORT...";
795 D$=INKEY$
800 KI$=INKEY$:IFKI$<>CHR$(13)THEN800
810 GOSUB860
815 ' SORT SPEED
820 PRINT@396,"SPEED SET AT :-"SQR(X1/25);
```

```
930 PRINT@428,"0 - 9 ALTERS SPEED";
840 PRINT@460,C$"."C$" STOPS PROGRAM";
850 RETURN
860 FORSC=1TO7:PRINT@268+SC*32,"                ";:NEXT
865 RETURN
870 '     POINTER ROUTINE
880 NC=NC+1:PRINT@N1,USING"###";NC
890 ND=ND+NE:PRINT@N2,USING"###";ND
900 PRINT@32*X+PP,"<--";
910 PRINT@32*Y+PP,"<--";
920 FORTI=1TOX1:NEXT
930 PRINT@32*X+PP,"   ";
940 PRINT@32*Y+PP,"   ";
950 D$=INKEY$:W$=INKEY$
960 IFW$<>""THENX1=VAL(W$)^2*25:X2=SQR(X1):
    PRINT@N6,VAL(W$);
970 IFW$="."THENGOSUB860:PRINT@333,"SORT  -
    TERMINATED";:GOTO460
980 RETURN
990 '  "SWITCHEM" ROUTINE
1000 FORK=0TO4
1010 PRINT@32*I+K+32,"  "A$(I);
1020 PRINT@32*J+K+A1,"  "A$(J);
1030 NEXTK
1040 PRINT@32*I+32,"  "A$(I);
1050 DF=J-I
1060 FORK=1TODF
1070 PRINT@32*(I+K-1)+A2,"        ";
1080 PRINT@32*(I+K)+A2,A$(I);
1090 FORTI=1TOX2:NEXT
1100 PRINT@32*(J-K+1)+A3,"      ";
1110 PRINT@32*(J-K)+A3,A$(J);
1120 NEXTK
1130 FORK=4TO0STEP-1
1140 PRINT@32*J+K+32,A$(I)"    ";
1150 PRINT@32*I+K+32,A$(J)"    ";
1160 NEXTK
1170 NE=NE+1
1180 PRINT@N3,USING"###";NE;
1190 TE$=A$(I):A$(I)=A$(J):A$(J)=TE$
1200 RETURN
1210 :PRINT@2,"--------- BUBBLE SORT -------
1220 GOSUB780
1230 FL=0
1240 FORI=0TONS-1
1250 X=I:J=I+1:Y=J:GOSUB880
1260 IFA$(I)<=A$(J)THEN1290
1270 FL=1
1280 GOSUB1000
1290 NEXTI
1300 IFFL=1THEN1230
1310 GOSUB420
1320 RETURN
1330 :PRINT@2,"------ SUPER BUBBLESORT -----
1340 GOSUB780
1350 FORI=0TONS-1
1360 J=I+1:X=I:Y=J:GOSUB880
1370 IFA$(I)<=A$(J)THEN1460
1380 GOSUB1000
1390 I1=I
1400 IFI=0THEN1450
1410 J=I:I=I-1:X=I:Y=J:GOSUB880
1420 IFA$(I)<=A$(J)THEN1450
1430 GOSUB1000
1440 GOTO1400
1450 I=I1
1460 NEXTI
1470 RETURN
1480 :PRINT@2,"-- BUBBLE SORT  WITH [illegible] -
1490 SE=NS:GOSUB780
1500 FL=0:SE=SE-1
1510 FORI=0TOSE
1520 X=I:J=I+1:Y=J:GOSUB880
1530 IFA$(I)<=A$(J)THEN1550
1540 FL=1:GOSUB1000
1550 NEXTI
1560 IFFL=1THEN1500
1570 GOSUB420
1580 RETURN
1590 :PRINT@2,"------- EXCHANGE  SORT -------",
1600 GOSUB780
1610 FORI=0TONS-1
1620 FORJ=I+1TONS
1630 X=I:Y=J:GOSUB880
1640 IFA$(I)<=A$(J)THEN1660
1650 GOSUB1000
1660 NEXTJ,I
1670 GOSUB420
1680 RETURN
1690 :PRINT@2," -- DELAYED REPLACEMENT SORT --
1700 GOSUB780
1710 J=0:R=0:I=-1
1720 I=I+1
1730 IFI=NSTHEN1820
1740 J=I:R=J+1
1750 X=J:Y=R:GOSUB880
1760 IFA$(R)>=A$(J)THEN1780
1770 J=R
1780 R=R+1:IFR<=NSTHEN1750
1790 IFI=JTHEN1720
1800 GOSUB1000
1810 GOTO1720
1820 GOSUB420
1830 RETURN
1840 :PRINT@2,"--------- SHELL SORT ---------
1850 GOSUB780
1860 M=NS
1870 M=INT(M/2):IFM=0THEN1950
1880 H=0
1890 I=H:FL=0:J=I+M:X=I:Y=J:GOSUB880
1900 IFA$(I)<=A$(J)THEN1930
1910 GOSUB1000
1920 FL=1
1930 H=H+1:IFJ<NSTHEN1890
1940 IFFL=1ANDM=1THEN1880ELSE1870
1950 GOSUB420
1960 RETURN
1970 :PRINT@2,"--- SHELL - METZNER SORT ---
1980 GOSUB780
1990 M=NS
2000 M=INT(M/2):IFM=0THEN2100
2010 P=NS-M
2020 H=0
2030 I=H
2040 J=I+M:X=I:Y=J:GOSUB880
2050 IFA$(I)<=A$(J)THEN2080
2060 GOSUB1000
2070 I=I-M:IFI>=0THEN2040
2080 H=H+1
2090 IFH>PTHEN2000ELSE2030
2100 GOSUB420
2110 RETURN
2120 :PRINT@2,"--------- QUICK SORT ---------
2130 GOSUB780
2140 I1=0:J1=NS:P=0
2150 PRINT@330,"STACK COUNTER.."USING"###";P
2160 I=I1
2170 PRINT@322,"SUPER-RECORD..."A$(I)
2180 J=J1:S=-1
2190 X=I:Y=J:GOSUB880
2200 IFA$(X)<=A$(Y)THEN2220
2210 GOSUB1000
2220 S=-S
2230 IFS=1THENI=I+1ELSEJ=J-1
2240 IFI<JTHEN2190
2250 IFI+2>J1THEN2270
2260 P=P+1:SS(P,1)=I+1:SS(P,2)=J1
2270 J1=I-1
2280 IFI1<J1THEN2150
2290 IFP=0THEN420
2300 I1=SS(P,1):J1=SS(P,2):P=P-1
2310 GOTO2150
2320 '                         EXIT
2325 CLEAR50:CLS:END
```

Pg 2 of 2

```
*** RESTORE FOR VZ-2ØØ/3ØØ ***
1 DATA 237,91,33,121,2Ø5,44,27,21Ø,217,3
Ø,11,237,67,255,12Ø,2Ø1
2 FORQ=31389TO314Ø4:READA:POKEQ,A:NEXT
3 POKE3Ø862,157:POKE3Ø863,122
```

Hint for VZ-200/300

EVER wanted to restore to a particular line number? This short routine will let you do it. In VZ basic, RESTORE simply sets the DATA LINE POINTER to the byte before the first program line. This routine makes the line number in the statement X=USR (line number) and calls a ROM routine to find the line in memory. It then moves back one byte, and stores this as the new DATA LINE POINTER.

An undefined statement error is given if no such line exists. This routine is stored in the cassette name buffer, but can be stored anywhere in memory.

Shane Rowe,
Spring Hill, Qld.

ETI Nov 89. p 73.

```
7A9D   ED 5B 21 79   LD DE,(7921H)   ; put no. passed by USR() into DE.
7AA1   CD 2C 1B      CALL 1B2CH      ; search for line no in DE
7AA4   D2 D9 1E      JP NC,1ED94     ; jump to UL error handling if
                                     ; line no. doesn't exist.
7AA7   0B            DEC BC          ; point to previous byte.
7AA8   ED 43 FF 78.  LD (78FFH),BC   ; put byte into DATA LINE PTR.
7AAC   C9            RET             ; return to BASIC code with DATA
                                     ; statements. RESTORED
```

NB. USR ptrs. set to 7A9D.

7A9D–7AAC is cassette buffer in Coms. area.

Hex/dec and dec/hex conversion

THIS short VZ listing does exactly what the name suggests and it can easily be adopted to other machines.

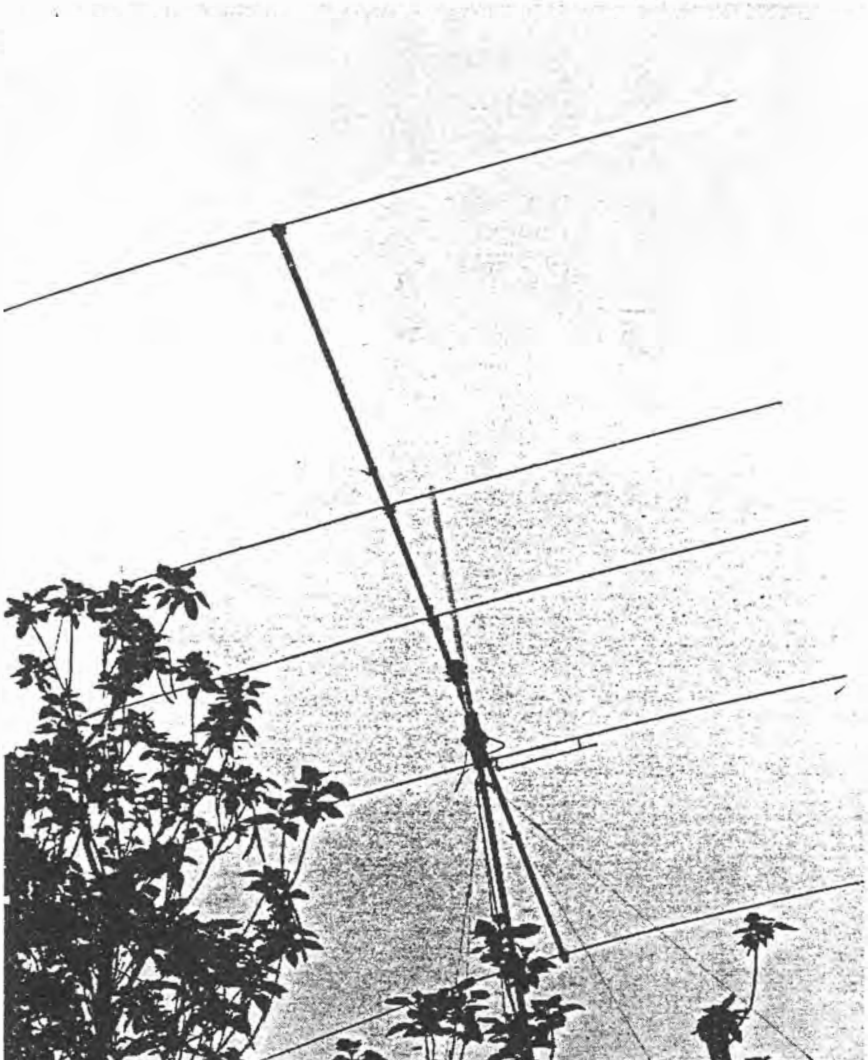
David Maunder,
Quirindi, NSW.

```
0 REM HEX/DEC TO DEC/HEX CONVERSION  WRITTEN BY DAVID MAUNDER.
1 REM  [C]OPYRIGHT 01/04/89 .THIS PROGRAM JUST CONVERTS
2 REM  HEXADECIMAL TO DECIMAL AND VICE VERSA .IT IS WRITTEN
3 REM  FOR THE VZ-200 AND VZ-300 BUT CAN BE VERY EASILY ADAPTED
4 REM  TO OTHER MACHINES
5 REM--------------------------------
6 H$="0123456789ABCDEF":EF%=0:ER$="?ERROR":FF=65536
7 CLS:PRINT"HEX/DEC TO DEC/HEX CONVERSION"
8 PRINT"WRITTEN BY D.MAUNDER"
9 PRINT
10 PRINT"WHICH:"
11 PRINT"        1 HEXADECIMAL TO DECIMAL"
12 PRINT"        2 DECIMAL TO HEXADECIMAL"
13 PRINT"        3 QUIT"
14 INPUT"?";A
15 IFA=1THEN19
16 IFA=2THEN28
17 IFA=3THENPOKE30845,199
18 GOTO7
19 CLS:
20 PRINT"          HEXADECIMAL TO DECIMAL"
21 PRINT"  RETN = CONTINUE        - = ABORT"
22 PRINT:INPUT"HEX#";H$:IFH$="-"THEN7
23 GOSUB40:IFEF%THENPRINTER$:GOTO22ELSEPRINT"DEC#=";D
24 Q$=INKEY$:Q$=INKEY$:IFQ$=""THEN24
25 IFQ$="-"THEN7
26 IFQ$=CHR$(13)THEN22
27 GOTO24
28 CLS
29 PRINT"          DECIMAL TO HEXADECIMAL"
30 PRINT"  RETN = CONTINUE        - = ABORT"
31 PRINT:INPUT"DEC#";N
32 IFN<0ORN>65535THENPRINTER$:GOTO31
33 FORI=1TO4:GOSUB46:NEXT
34 PRINT"HEX#=";N$
35 N$=""
36 Q$=INKEY$:Q$=INKEY$:IFQ$=""THEN36
37 IFQ$="-"THEN7
38 IFQ$=CHR$(13)THEN31
39 GOTO36
40 EF%=0:D=0:LN%=LEN(H$):IFLN%>4THEN45
41 FORI%=1TOLN%:B$=MID$(H$,I%,1)
42 IF(B$=>"0"ANDB$=<"9")OR(B$=>"A"ANDB$=<"F")THEN43ELSE45
43 J%=ASC(B$)-48:IFJ%>9THENJ%=J%-7
44 D=D*16+J%:NEXT:RETURN
45 EF%=1:RETURN
46 EF%=0
47 A=INT(N/16):Z=N-16*A:N$=MID$(H$,(Z+1),1)+N$:N=A:RETURN
```

ETI Nov 89. p 73.

DO YOUR OWN BEAM HEADINGS

Greg Baker explains how



Directional beam antennas are useful for CBers and SWLs alike. Greg Baker describes a simple home computer program to tell you where to point your beam.

With the freeing of the rules on CB antennas, the use of directional beams has become an option to push a bigger signal just where you want it. The problem, of course, is to know exactly where to point the beam when you have it set up and connected to your transceiver.

Browsing through a road map or atlas will give you some idea of where to point, but, if your beam is efficient (which means that it will have a narrow directional lobe) or you want the biggest possible DX signal, a great deal more precision is needed.

That precision can come from messing around with a calculator and set of formulas, but, it can come far easier come from a home computer...and now that we all have computers or access to them, there's no excuse not to be spot on with your beam headings.

So, push the kids off the computer for a couple of hours and tell them to kick a football around the yard instead of playing computer games. Then type in the program listed over the page and run it on your favourite DX targets.

It you and the machine can't get on, call the kids back. They'll love to lord it over you and tell you what to do. Let them enjoy it, it's good for them. They can have the ego trip....all you want are the headings.

PROGRAMMED IN BASIC

The program is written in BASIC, the standard home computer language. There are no fancy features used, so it should run without modification on most machines. It has been developed and tested on the Dick Smith VZ200.

As it is listed, the program assumes you are in Sydney. If your base station, or mobile for that matter, is elsewhere, replace line 40 with your latitude in L(1,1) and longitude in L(2,1).

Remember the sign on the latitude.

Remember also that latitudes and longitudes are the usual way to precisely locate a place on the face of the earth. Latitude is the number of degrees north or south of the equator run-

1. of 3.

DO YOUR OWN BEAM HEADINGS

ning from zero at the equator to 90 degrees at the poles. Longitude is the number of degrees west or east of the north south line running through Greenwich in England.

PROGRAM:

```
10 DIM L(2,2),D$(2)
20 D$(1) = 'ORIGIN'
30 D$(2) = 'TARGET'
40 L(1,1) = -33.9 : L(2,1) = 151.2
50 E = 57.29578 : PI = 3.14159
60 PRINT 'NEW ORIGIN? Y OR N'
70 INPUT Y$
80 IF Y$ <> 'Y' THEN 110
90 K = 1
100 GOSUB 350
110 K = 2
120 GOSUB 350
130 P = L(2,1) - L(2,2)
140 PS = 1
150 IF P < 0 THEN PS=0
160 P = ABS(P)
170 PM = 0
180 IF P > 180 THEN PM = 1
190 P = P/E
200 PA = (90 - L(1,1))/E
210 PB = (90 - L(I,2))/E
220 Z = COS(P)*SIN(PA)*SIN(PB)
+COS(PA)*COS(PB)
230 GOSUB 460
240 KM% = 6366.7 * M
250 Z = (COS(PB)-COS(PA)*COS
(M))/(SIN(PA)*SIN(M))
260 GOSUB 460
270 A = M * E
280 A% = ABS(360 * (PS-PM) *
(PS-PM) - A)
290 PRINT 'BEARING IS: '; A% ;'
DEGREES'
300 PRINT 'DISTANCE IS: '; KM% ;
'KILOMETRES'
310 PRINT 'CONTINUE? Y OR N'
320 INPUT Y$
330 IF Y$ <> 'Y' THEN END
340 GOTO 60
350 PRINT D$(K);' LATITUDE?'
360 INPUT L(1,K)
370 PRINT D$(K); ' LONGITUDE?'
380 INPUT L(2,K)
390 FOR I = 1 TO 2
400 T = 90 + (I-1) * 90
410 IF ABS(L(I,K)) <= T THEN 440
420 PRINT 'ERROR: TRY AGAIN'
430 GOTO 350
440 NEXT I
450 RETURN
460 M = - ATN(Z/SQR(1- Z * Z)) +
PI/2
470 RETURN
```

USING THE PROGRAM

When you **RUN** the program, it will ask you whether you want to change the origin latitude or longitude. If you are mobile or a friend wants to use the program from his location, you can temporarily change the origin here by typing Y and INPUTing the new latitude and longitude. Otherwise type N to continue.

The program then asks for latitude and longitude of the target. Type in the target latitude and longitude using the list below. **Remember to type in the minus sign for latitudes from the list.**

If the target you want is not in the list, turn to an atlas or gazetteer (list of place names) and look up the latitude and longitude of the target you require.

The program will function to and from places other than in Australia, so if you want to listen in to what is happening elsewhere, use the latitude and longitude of the place you are interested in.

DON'T FORGET THE MINUS MARK

Make sure all places south of the equator, ie southern latitudes, are input with a minus sign in front of them. Northern latitudes are positive and thus require no sign.

Similarly, places up to 180 degrees west of Greenwich in England should have a negative sign. All DX targets in our region are east of Greenwich and hence are all positive numbers and need no sign.

You will need to convert the latitudes and longitudes you have found in your atlas or gazetteer to values which this program can use. Notice that the latitudes and longitudes from an atlas or gazetteer are written in the form of degrees and minutes. Convert these by dividing the minutes by 60 with a calculator and adding to the degrees. Thus Grenfell in N.S.W is 33 degrees 54 minutes South, 148 degrees 11 minutes East. The latitude t o use in the program is 33 plus (54 divided by 60) which equals 33 + 0.9 or 33.9 and this becomes - 33.9 when you add the negative sign for the south latitude. Similarly, the longitude is 148 plus (11v60) = 148 + 0.1833 = 148.2 when you round it for ease of INPUTing.

WARNING: The program may produce errors if your chosen target is within about fifty kilometres of the origin or you want to see if there is anyone at the poles calling CQ DX.

Still, in either case you wouldn't need this program anyway. Up to fifty kilometres you don't need the precision of this program, and for that lone CBer at the pole, just point your beam due north or south. And even then DoTaC rules mean you won't be allowed to reply to that plaintive call for a ragchew from the wilderness.

TEST DATA

When you have typed the program into the computer and double-checked that you have typed it properly, you should test it on the following DX paths. Note that for each of these you will need to change the origin latitude and longitude where the program requests it. You will also need to re-RUN the program for each new origin. This involves, at the end of each test path, typing N when asked if you want to continue. Then start again with another RUN.

HOW TO USE THE BEARINGS

The program will output the true bearing of the target from the origin and the distance in kilometres.

ORIGIN	TARGET	BEARING (Degrees)	DISTANCE (Kms)
Sydney -33.9,151.2	Lismore -28.8,153.3	19	600
Whyalla -33.0,137.6	Adelaide -34.9,138.6	156	230
Geraldton -28.8,114.6	Brisbane -27.5,153.0	97	3748
end of chart			

If you didn't get these results, you will find a typing error in your program.

2. of 3

The distance is useful in finding out whether the target is within the coverage of the ground wave, in the blank area within the skip zone but outside the ground wave coverage area or in useful DX range beyond the skip distance.

The true bearing differs from a magnetic bearing given by an ordinary compass and it differs by different amounts depending on where you are. The difference is called the local magnetic variation though sometimes it is called declination.

To find the magnetic (compass) bearing from the true bearing output by the program, subtract the magnetic variation at the origin station from the computer calculated true bearing. Approximate magnetic variations are given in the table of latitudes and longitudes below.

Notice that when magnetic north is east of true north the variation is easterly and given a positive sign. When magnetic north is west of true north (as it is in some parts of Western Australia) the variation is westerly and given a negative sign.

Regardless of the sign though of the variation, you must add it to the true bearing to get magnetic bearing. To find the variation at origin s other than on the list you will need to use the nearest from the list or check out a good army survey map at your local library.

Align the beam with this magnetic bearing, remembering to keep your compass away from such large amounts of steel as your car. Once you have found the bearings of your most usual DX targets, mark them near the beam so that you can easily align the antenna next time.

PLACES AND THEIR LATITUDE/LONGITUDE

Place	Latitude	Longitude	Magnetic Variation
A.C.T.			
Canberra	-35.3	149.1	12
NEW SOUTH WALES			
Albury	-36.1	146.9	12
Armidale	-30.5	151.7	12
Bathurst	-33.5	149.6	13
Broken Hill	-32.0	141.5	9
Dubbo	-32.3	148.7	12
Goulburn	-34.8	149.7	14
Grafton	-29.7	152.9	12
Lismore	-28.8	153.3	11
Lithgow	-33.5	150.2	15
Newcastle	-32.9	151.8	15
Orange	-33.3	149.2	13
Sydney	-33.9	151.2	15
Tamworth	-31.1	151.0	11
Taree	-31.9	152.4	12
Wagga	-35.1	147.4	14
Wollongong	-34.4	150.9	14
VICTORIA			
Ballarat	-37.6	144.0	11
Bendigo	-36.8	144.4	11
Geelong	-38.2	144.4	12
Hamilton	-37.8	142.1	11
Horsham	-36.8	142.3	11
Melbourne	-37.8	145.0	13
Mildura	-34.2	142.2	11
Morwell	-38.2	146.4	12
Shepparton	-36.4	145.4	12
Wangaratta	-36.4	146.3	12
Warrnambool	-38.4	142.5	11
QUEENSLAND			
Brisbane	-27.5	153.0	11
Bundaberg	-24.8	152.4	10
Cairns	-16.9	145.7	7
Gladstone	-23.9	151.3	10
Gympie	-26.2	152.6	11
Mackay	-21.2	149.2	9
Maryborough	-25.5	152.6	11
Mount Isa	-20.8	139.5	7
Rockhampton	-23.4	150.5	10
Townsville	-19.2	146.8	8
Warwick	-28.2	152.0	11
SOUTH AUSTRALIA			
Adelaide	-34.9	138.6	9
Mount Gambier	-37.9	140.8	10
Port Augusta	-32.5	137.8	8
Port Lincoln	-34.7	135.8	7
Whyalla	-33.0	137.6	8
WESTERN AUSTRALIA			
Albany	-35.0	117.9	-4
Bunbury	-33.3	115.6	-3
Geraldton	-28.8	114.6	-2
Kalgoorlie	-30.8	121.5	1
Perth	-32.0	115.8	-3
TASMANIA			
Burnie	-41.1	145.9	15
Devonport	-41.2	146.3	15
Hobart	-42.9	147.3	16
Launceston	-41.4	147.1	15
NORTHERN TERRITORY			
Alice Springs	-23.7	133.9	5
Darwin	-12.4	130.9	4

PREFIX/ COUNTRY	CENTRED ON CITY	SHORT PATH	LONG PATH	KILOMETRES (SHORT PATH)
MINNESOTA	ST PAUL	57	237	15017
MISSISSIPPI	JACKSON	75	255	14982
MISSOURI	JEFFERSON CITY	66	246	14984
MONTANA	HELENA	52	232	13622
NEBRASKA	LINCOLN	62	242	14659
NEVADA	CARSON CITY	58	238	12740
NEW HAMPSHIRE	CONCORD	60	240	16738
NEW JERSEY	TRENTON	66	246	16493
NEW MEXICO	SANTA FE	66	246	13723
NEW YORK	ALBANY	61	241	16561
NORTH CAROLINA	RALEIGH	74	254	15981
NOSTH DAKOTA	BISMARCK	54	234	14460
OHIO	COLUMBUS	66	246	15784
OKLAHOMA	OKLAHOMA CITY	68	248	14262
OREGON	SALEM	51	231	12760
PENNSYLVANIA	HARRISBURG	66	246	16223
RHODE ISLAND	PROVIDENCE	63	243	16764
SOUTH CAROLINA	COLUMBIA	78	258	15918
SOUTH DAKOTA	PIERRE	57	237	14447
TENNESSEE	NASHVILLE	71	251	15389
TEXAS	AUSTIN	75	255	14226
UTAH	SALT LAKE CITY	58	238	13426
VERMONT	MONTPELIER	58	238	16644
VIRGINIA	RICIMOND	71	251	16677
WASHINGTON	OLYMPIA	49	229	12860
WEST VIRGINIA	CHARLESTON	69	249	15882
WISCONSIN	MADDISON	60	240	15296
WYOMING	CHEYENNE	60	240	14005
KC6 EASTERN CAROLINE IS. (O-27)	-	0	180	4529
KG4 GUATANAMO BAY (NA-8)	-	100	280	15834
KG6R/S/T MARIANA IS. (O-27)	TINIAN	2	182	5421
KH1/KB6 AMERICAN PHOENIX IS. (O-31)	-	55	235	5554
KH2/KG6 GUAM (O-27)	APIA HARBOUR	0	180	5244
KH3/KJ6 JOHNSTON IS. (O-31)	-	48	228	7435
KH4/KM6 MIDWAY IS. (O-31)	-	35	215	7974
KH5KP6 PALMYRA/JARVIS IS. (O-31)	-	62	242	7138
KH6 HAWAII IS. (O-31)	HONOLULU	53	233	8648
KH7/KH6 KURE IS. (O-31)	-	34	214	7969
KH8/KH6 AMERICAN SAMOA	FAGATOGO	73	253	5200
KH9/KW6 WAKE IS. (O-31)	-	25	205	6347
KP1/KC4 NAVASSA IS.(NA-8)	-	103	283	15757
KP2/KV4 AMERICAN VIRGIN IS. (NA-8)	-	112	292	16639
KP3/KS4/HK0 SERRANA BANK	-	105	285	15042
KP4 PUERTO RICO (NA-8)	SAN JUAN	111	291	16543
KP4 DESECHO IS. (NA-8)	-	110	290	16416
KX6 MARSHALL IS. (O-31)	KWAJALEIN	31	211	5364
KZ PANAMA CANAL ZONE (NA-7)	-	111	291	14754

Bint Services produce a computer based 'beam heading list' which has both short and long path bearings to all amateur callsign areas — cost for the listing (which is based on the lat/long of your QTH is $15.

3 of 3

GAMES

Nov/Dec	83	SYN	22-24	Projectile Plotting (Grosjean)	(2)
Dec.	83	APC	161-3	Missile Command. (Whitwell)	(2)
Feb.	84	BB	50-51	Caddy and Reaction Test. (Hartnell)	(2)
Jan.	84	YC	65	Graphic Sine Waves for VZ-200. (Nickasen)	(1)
Apr.	84	APC	178-80	Moon Lander. (Alley)	(2)
Jul.	84	APC	174-8	Blockout. (Pritchard)	(3)
Jul.	84	M80	7,22	Battleships. (Carson)	(1)
Jul.	84	M80	7,20,21	Junior Maths. (Carson)	(2)
Aug.	84	M80	9,16	Contest Log VZED. (Carson)	(1)
Aug.	84	M80	9,16,17	Dog Race VZED. (Carson)	(1)
Oct.	84	PCG	55-7	High Resolution Graphics Plotting. (Thompson)	(3)
Nov.	84	PCG	82	Tips for 'Ladder Challenge', 'Panik' and 'Asteroids'.	(1)
Jan.	85	PCG	54	POKE's to 'Ghost Hunter'.	(-)
-	85	BYC	146-7	Golf Simulation. (McCleary)	(2)
Mar.	86	CFG	4-5	Golf Simulation. (McCleary)	(-)
-	85	BYC	147	Knight's Cross. (Lucas)	(1)
Jan.	85	APC	129-31	Sketcher. (Leon)	(3)
Jan.	85	YC	88-89	Punch. (Rowe)	(2)
Jan.	85	PCG	44-48	Space Station Defender. (Shultz)	(5)
Feb.	85	CI	27-28	Lost. (Potter)	(2)
Mar.	85	YC	105-9	Decoy. (Rowe)	(2)
Mar.	85	CI	-	Mouse Maze. (Crandall)	(1)
Apr.	85	YC	160	Painter. (Daniel)	(1)
Apr.	85	PCG	65-7	Roadrace. (Thompson)	(3)
May	85	YC	106	Number Sequence. (Thompson)	(1)
May/Jun	85	PCG	63-7	Sketchpad. (Thompson)	(5)
Jun	85	YC	70	Morse Tutor program. (Heath)	(1)
Jan.	86	YC	150-1	Morse Tutor - again. (Heath)	(2)
Jul.	85	YC	81	Electric Tunnel. (Daniel)	(1)
Aug.	85	YC	114	Number Slide. (Daniel)	(1)
Oct.	85	PCG	47-52	Cube. (McMullan)	(6)
Oct.	85	YC	105-7	Yahtzee. (Thompson)	(3)
Mar.	86	APC	208-9	VZ Frog. (Alley)	(1)
May	86	ETI	93	Balloon Safari, The Drop and Flatten. (Sheppard)	(1)
Jul.	86	YC	75	Simon. (Proctor)	(1)
-	88	BYC	76	Drawing Program. (Winter)	(1)
-	88	BYC	77	Tea-pot Song. (Winter)	(1)
-	88	BYC	78	Ping Tennis. (Duncan)	(1)
-	88	BYC	79-82	Concentration. (Vella)	(4)
-	88	BYC	83	Super Snake Trapper. (Duncan)	(1)
-	88	BYC	84	Worm. (Thompson)	(1)
-	88	BYC	85	Dogfight. (Thompson)	(1)
-	88	BYC	86-87	Bezerk. (Banks & Saunders)	(2)
-	88	BYC	87	Arggggh! (Banks & Saunders)	(1)
-	88	BYC	87	Encode/Decode. (Banks & Saunders)	(1)
-	88	BYC	88	Catch. (Banks & Saunders)	(1)
Apr.	88	ETI	65	U-foe. (Alderton)	(1)
Jul.	88	ETI	73	Disintegrator. (Stibbard)	(1)
Aug.	88	ETI	65	Star Fighter. (Roberts)	(1)
Nov.	88	ETI	121	Drawing Board. (Maunder)	(1)
May	89	ETI	87-88	Camel (Maunder)	(2)

Plotting a Projectile

David Grosjean

In this issue we will compare programming the VZ200, the color and sound computer by Video Techonology, and the TS1000. The project we will undertake is the plotting of a projectile.

Starting with a Clear Screen

Let's start with a simple clear screen and plot statement.

TS1000:

```
 10 CLS
200 PLOT X,Y
```

VZ200:

```
5 CLS
40 MODE(1):COLOR 4
200 SET(X,Y)
```

If you look at the VZ200 program, you will notice that the computer has to be put into a special graphics mode with line 40. This means that you cannot have the medium resolution graphics and text on the screen at the same time. This will become a problem when we try to turn this into a game.

The Projectile Equations

The equations for the horizontal and vertical position of a projectile are:
X=V*COS(A)*T
Y=V*SIN(A)*T-1/2*G*(T*T)

V is the velocity; T is the time; G is the effect of gravity. These equations can be worked into the program like this:

TS1000:

```
 20 LET V=1000
 30 LET D=57.3
 40 LET A=45
 50 LET C=V*SIN (A/D)
 60 LET C1=V*COS (A/D)
 80 FOR T=0 TO 44 STEP .5
 90 LET X=C1*T
100 LET Y=C*T-16*T*T
180 LET X=X/500
190 LET Y=Y/500
220 NEXT T
```

VZ200:

```
10 A=45
20 V=1000:G=32
30 D=57.3
50 C=V*SIN(A/D)
60 C1=Y*COS(A/D)
80 FOR T=0 TO 45 STEP .5
90 X=C1*T
100 Y=C*T-16*T*T
180 X=X/250
190 Y=Y/250
220 NEXT T
```

As you will notice, the range on the VZ200 increased due to the higher resolution of the graphics, but we did not change the velocity of the projectile. Instead, we changed the number which we divide X and Y by to fit the projectile on the different screen size.

In these programs, D is a factor that converts degrees to radians which are what the computer wants. C and C1 are constants for each firing angle. When you RUN this program on the VZ200, you will notice that the plot is upside down. This is because the vertical distances are measured from top to bottom instead of bottom to top as on the TS1000. Change line 190 in the VZ200 program to

```
190 Y=61-Y/250
```

Setting the Gun Angle

Now we can modify the programs to accept a gun angle from 1 to 90 degrees.

TS1000:

```
 40 PRINT "ANGLE OF GUN?"
 45 INPUT A
 70 LET T1=2*C/32
 80 FOR T=0 TO T1 STEP .5
230 GOTO 50
```

VZ200:

```
10 INPUT "ANGLE OF GUN";A
70 T1=2*C/G
80 FOR T=0 TO T1 STEP .5
230 GOTO 50
```

Making a Game

Now that we have a working, however simple, projectile program, let's try to make a game out of it. The following games are our projectile programs tightened up a bit and with the provisions for a target.

Setting up the Target

On the VZ200 the range is 127,000 yards, and on the TS1000 32,000 yards (1000 yards for every horizontal position on the screen). This will throw the equation off a little since the gun cannot shoot the

p 22.-24.

1 of 2.

projectile 127,000 yards. (If this bothers you, think of the yards on the VZ200 as 11-inch feet.)

Although there are 64 pixel positions on the TS1000, the target is a T which takes up two pixels. You can hit the left or the right of the T so the number of effective horizontal positions is reduced to half. Notice that, since the VZ200 cannot have text and graphics on the screen at once, line 100 forms a special target, while on the TS1000, a simple PRINT AT command in line 60 does the same thing.

TS1000:

```
 20 LET V=1000
 40 LET K=INT (20000*RND)+12000
 50 CLS
 52 PRINT "RANGE = 32000 YDS"
 60 PRINT AT 21,INT (K/1000);"T
"
 70 PRINT AT 1,0;"ANGLE OF GUN?
"
 80 INPUT A
 90 IF A<1 OR A>90 THEN GOTO 90
120 LET C=V*SIN (A/57.3)
130 LET C1=V*COS (A/57.3) .,
140 LET T1=2*C/32
150 FOR T=0 TO T1 STEP .5
160 LET X=C1*T/500
170 LET Y=T*(C-16*T)/500
180 PLOT X,Y
190 NEXT T
```

VZ200:

```
20 V=1000
40 K=INT(97000*RND(●))+30000
50 PRINT "RANGE = 127000 YDS"
60 PRINT "TARGET AT";K;"YDS"
70 INPUT "ANGLE OF GUN";A
80 IF A<1 OR A>89 THEN 70
90 MODE(1):COLOR4
100 FORL=1 TO 4:FORL1=1 TO 4:SET
(INT(K/1000-4)+L1,59+L):NEXT:NEXT
130 C=V*SIN(A/57.3)
140 C1=V*COS(A/57.3)
150 T1=2*C/32
160 FOR T=0 TO T1 STEP .5
170 X=C1+T/250
180 Y=61-(T*(C-16*T)/250)
190 SET(X,Y)
210 GOTO 210
```

Detecting a Hit

We now have a target, but it is of no use unless the computer can detect its destruction. The following lines detect a hit. Notice how the techniques of detecting a hit target differ. The VZ200 must compare each position of the target, which is four positions wide, with the last position of the projectile; the TS1000 does the same thing but uses the PRINT AT position used by the target to compare to the last position of the projectile. This is, of course, simpler. Line 300 in the VZ200 version is a special "explosion" accompanied with some sounds. You can experiment at this point to find a better explosion.

TS1000:

```
 200 IF INT (X/2)=INT (K/1000) T
HEN GOTO 300
 250 GOTO 50
 300 PRINT AT 21,INT (K/1000)-2;
"▛▜▛"
 310 PAUSE 250
 340 GOTO 30
```

VZ200:

```
220 FOR L=1 TO 4:IF INT(K/1000)-
L=INT(X) THEN 300
225 NEXT L
250 GOTO 50
300 FORL=1 TO 30:SET(40+87*RND(0
),40+22*RND(0)):SOUND31,1:NEXT L
310 PRINT "HIT! HIT! HIT!"
340 GOTO 30
```

Making the Next Shot

Now we can add the response the computer will make to a missed target. The following lines tell how far away your shot was from the target and lets you try again. Line 210 in the VZ200 version is a delay loop so you have time to see the last position of the projectile.

TS1000:

```
 210 LET E=INT (K-(32000*SIN (.0
35*A)))
 220 IF E<100 THEN PRINT AT 0,0;
"OVER BY ";ABS E;" YDS"
 230 IF E>100 THEN PRINT AT 0,0;
"UNDER BY ";ABS E;" YDS"
 240 PAUSE 250
```

VZ200:

```
210 FOR L=1 TO 3000:NEXT L
230 IF INT(K/1000)>X THEN PRINT
"UNDER BY";K-X*1000;"YDS"
240 IF INT(K/1000)<X THEN PRINT
"OVER BY";X*1000-K;"YDS"
```

Providing Your Shots

The computer can now detect hits and misses. This is where the game part comes in. The following lines provide you with 5 individual targets with a maximum of 5 attempts to hit each target. If you fail to hit a target in 5 shots, you lose. S is the number of shots you have taken per target; S1 is your total number of shots; and Z is the total number of targets.

TS1000:

```
   5 LET Z=0
  10 LET S1=0
  30 LET S=0
  55 IF S=5 THEN GOTO 260
 100 LET S1=S1+1
 110 LET S=S+1
 260 PRINT AT 0,0;"ENEMY GOT YOU
FIRST"
 270 GOTO 370
 320 LET Z=Z+1
 330 IF Z=5 THEN GOTO 350
```

VZ200:

```
10 S1=0:Z=0
30 S=0
55 IF S=5 THEN 260
110 S=S+1
120 S1=S1+1
260 PRINT "THE ENEMY GOT YOU FIR
ST!"
270 GOTO 370
320 Z=Z+1
330 IF Z=5 THEN 350
```

Evaluation and Restart

Finally, we need an evaluation and a mechanism to restart the game. The following lines do this.

TS1000:

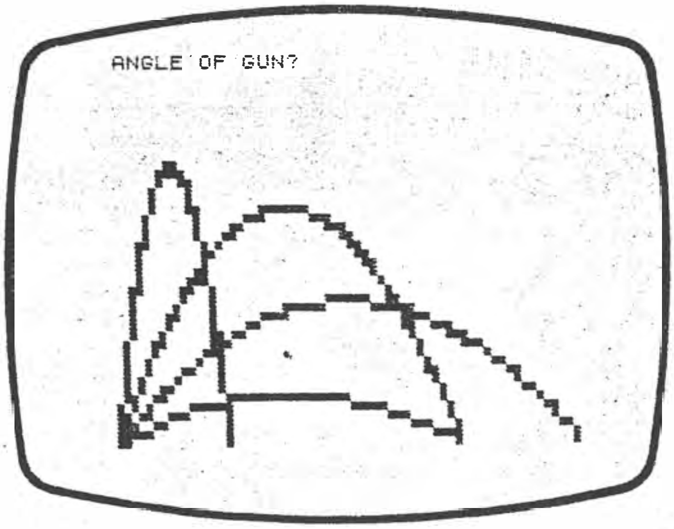
```
 350 PRINT AT 0,0;S1;" ROUNDS US
ED
 355 IF S1<10 THEN PRINT "GREAT
JOB"
 360 IF S1>15 THEN PRINT "YOU CA
N DO BETTER"
 370 PRINT "PLAY AGAIN?  "
 380 INPUT Z$
 390 IF Z$="Y" THEN RUN
```

VZ200:

```
350 PRINT S1;"ROUNDS USED"
355 IF S1<10 THEN PRINT "GREAT J
OB!"
360 IF S1>15 THEN PRINT "YOU COU
LD HAVE DONE BETTER"
370 INPUT "PLAY AGAIN";Z$
380 IF Z$="Y" THEN RUN
```

Improving on the Game

Of course, these artillery-type games are very simple. They provide a basic game which you can elaborate on or experiment with to develop different possibilities. You might want to improve on the graphics or sound on the VZ200 or perhaps make a really BIG explosion. Although the TS1000 has no color or sound, the program can still be greatly improved. You could add hi-res graphics through either a hardware add-on or a software program. You might want to add a sound unit which will give the sound effects or add a routine to provide some sound (e.g., AUDISY). ▣



ANGLE OF GUN?

2 of 2.

Missile Command

by Keith Whitwell

This is the first program we've received for the VZ-200 and it's from someone who's only in grade 8. It is a Basic version of the famous arcade game of the same name and uses the following keys for control of the "cross-hairs":

Y: UP
B: DOWN
G: LEFT
H: RIGHT

F: FIRE
U: ACCELERATE
N: STOP

Other instructions are included in the listing.

```
0 REM MISSILE COMMAND, KEITH WHITWELL, 8F, SPLC, 31/10/83
1 CLS:PRINT"          MISSILE COMMAND":PRINT
2 PRINT "   BY KEITH WHITWELL, 8F, SPLC.":PRINTTAB(13)"1983"
3 PRINT:PRINT
4 PRINT:INPUT"INSTRUCTIONS";A$:IFLEFT$(A$,1)="Y"
  THEN GOSUB 3000
5 INPUT"LEVEL OF SKILL(1 (V.HARD)-4)";LS:IF LS>4 OR LS<1 THEN 5
6 FOR I=1 TO 4:C(I)=1:NEXT
8 GOSUB 2000
9 A=63:K=32:S=0
10 MODE(1):COLOR 4,1
11 MI=0
15 Y=50
16 Q=0
17 SC=SC+S:S=0
18 CN=0
20 FOR X=1TO127:SET(X,INT(Y)):Y=Y+.03:IF INT(Y)>Q
   THEN GOSUB1000
21 Q=INT(Y):NEXT
30 SN=RND(5)+5
31 COLOR 6:SET(SN,62):COLOR 7:SET(SC,62)
90 X=A:Y=K
100 A$=INKEY$
101 IF A$="Y" THEN N=-1:M=0
102 IF A$="B" THEN N=1:M=0
103 IF A$="H" THEN M=1:N=0
104 IF A$="G" THEN M=-1:N=0
105 IF A$="F" THEN COLOR 3:GOSUB 1100
106 IF A$="U" THEN GOSUB 1300
107 IF A$="N" THEN N=0:M=0
110 IF X+M>120OR Y+N>48 OR Y+N<5 OR X+M<5 THENN=0:M=0
120 COLOR 1:GOSUB 1050:X=X+M:Y=Y+N:COLOR 3:GOSUB 1050
130 COLOR 4:FOR I=1 TO 4:Y(I)=Y(I)+1:J=RND(2)-2:
    :IF J=0 THEN J=1
131 IF X(I)+J<5 OR X(I)+J>120 THEN J=-J
132 X(I)=X(I)+J
140 P=POINT (X(I),Y(I)):IF P=4 THEN COLOR 4:GOSUB 1200
141 IF P=2 THEN COLOR 4:GOSUB 1500
150 COLOR 3:SET(X(I),Y(I)):COLOR 4
160 NEXT
300 GOTO 100
```

A.P.C. Dec. 83. 4(12) 161-163
1. of 2.

```
1000 DATA1,18,1,1,13,2,15,17,2,1,13,3,16,16,3,1,7,4,9,
     12,4,4,7,5
1001 DATA 9,12,5,4,7,6,9,12,6,5,6,7,10,11,7,10,11,8
1010 CN=CN+1:IF C(CN)=0 THEN RETURN ELSE SC=SC+10
1011 COLOR 6:FOR I=1 TO 14:READ R,T,V
1020 FOR H=R TO T:SET(X+H,Y-V):NEXT:NEXT:RESTORE:COLOR 4:RETURN
1050 FOR H=-1 TO 1:SET(X+H,Y+1):SET(X+H,Y-1):NEXT
1051 SET(X,Y-1):SET(X,Y+1)
1052 RETURN
1100 REM
1101 FOR I=1 TO 4:IF X(I)<X+LS AND X(I)>X-LS THEN 1103
1102 NEXT
1103 IF Y(I)<Y+LS AND Y(I)>Y-LS THEN S=S+1:SET(S,62):GOTO 1110
1104 RETURN
1110 IF S=SN THEN A=X:K=Y:GOTO 2200
1200 FOR E=1 TO 5:SET (X(I)-E,Y(I)):SET(X(I)+E,Y(I))
1201 SET(X(I),Y(I)-E):SET(X(I),Y(I)+E):NEXT
1210 X(I)=RND(110)+6:Y(I)=6
1300 IF M=-1 THEN M=-4
1320 IF M=1 THEN M=4
1350 IF N=-1 THEN N=-4
1360 IF N=1 THEN N=4
1390 RETURN
1500 IF X(I)<20 THEN C(1)=0
1501 IF X(I)>33 AND X(I)<53 THEN C(2)=0
1502 IF X(I)>66 AND X(I)<86 THEN C(3)=0
1503 IF X(I)>100 THEN C(4)=0
1510 FOR C=1 TO 4:IF C(C)=0 THENF=F+1
1511 NEXT
1520 IF F=4 THEN GOSUB 1200:GOTO 2100
1530 F=0:GOTO 1200
2000 FOR I=1 TO 4:X(I)=RND(110)+5:Y(I)=RND(5)+6:NEXT
2001 RETURN
2100 FOR F=1TO30:X(I)=RND(100)+10:Y(I)=RND(40)+9:GOSUB
     1200:NEXT
2110 SC=SC+S
2111 FOR I=1 TO 1000:NEXT:CLS:PRINT@64,"SCORE="SC
2112 IF SC>HS THEN PRINT"YOU HAVE THE HIGH SCORE!!!":HS=S
2113 PRINT:PRINT"HIT 'S' TO START A NEW GAME"
2120 IF INKEY$="S" THEN 1
2121 GOTO 2120
2200 FOR Q=1 TO SL-1
2210 FOR I=1 TO 4:IF C(I)=0 THEN C(I)=1:I=9
2220 NEXT:GOTO 10
3000 CLS:PRINT"          MISSILE COMMAND"
3010 PRINT:PRINT"YOU COMMAND OUR ANTI-MISSILE
     MISSILES.YOUR";
3015 PRINT" JOB IS TO KEEP THEMISSILES FROM LANDING ON THE "
3020 PRINT"CITIES BY MOVING YOUR SIGHTS    ONTO THE HEAD OF ";
3025 PRINT"ONE OF THE FOURMISSILES COMING DOWN AND PRESS  ";
3030 PRINT "'F', WHICH FIRES YOUR MISSILE.
     IF IT HITS, THEN ";
3035 PRINT"THE MISSILE    WILL EXPLODE."
3040 PRINT:INPUT"HIT RETURN";A$
3045 CLS:PRINT"          MISSILE COMMAND":PRINT
3050 PRINT"    Y  UP"
3051 PRINT"    B  DOWN"
3052 PRINT"    G  LEFT"
3053 PRINT"    H  RIGHT"
3054 PRINT:PRINT"    F  FIRE"
3055 PRINT"    U  ACCELERATE"
3056 PRINT"    N  STOP"
3060 PRINT:INPUT"HIT RETURN";A$
3065 CLS:PRINT"          MISSILE COMMAND":PRINT
3070 PRINT"SCORING IS:"
3071 PRINT"          1 POINT PER MISSILE DISTROYED"
3072 PRINT"          10 POINTS PER CITY  NOT BLOWN UP"
3075 PRINT:PRINT"YOU WILL GET EXTRA CITIES EVERY"
3076 PRINT"SCREEN, THE NUMBER DEPENDS ON   THE LEVEL OF SKILL."
3080 PRINT:PRINT"THE GAME ENDS WHEN ALL OF YOUR  CITIES ARE";
3081 PRINT" DISTROYED"
3090 PRINT:INPUT"HIT RETURN";A$
3091 RETURN
```

2 of 2.

Two games to key in

The following programs are reprinted with the permission of Dick Smith Electronics from *Getting Started* (on the VZ200), by Tim Hartnell and Neville Preteborn.

Getting Started and another four books written especially for the VZ200 are now available in New Zealand from Dick Smith Electronics and its dealers.

Out on the Fairway

A golf game called Caddy. You have nine holes to negotiate, as you'll see when you play the game, the computer obligingly keeps the score card for you. After each hole, it will tell you how you are doing to date, and will work out your average score per hole. All you have to do is hit the ball! If you overshoot, the computer will automatically make sure the next shot is back towards the hole. You'll find it pretty tricky going, especially on holes with a high difficulty factor.

Here's the listing, golf pro:

```
10 REM CADDY
20 DIM X(9):CO=0:H$=CHR$(216)
30 U=224:L$="
40 FOR Z=1 TO 9
50 SC=0
60 J=RND(12)
70 Q=RND(3)+2
80 IF Q=5 THEN Q$="FIVE"
90 IF Q=4 THEN Q$="FOUR"
100 IF Q=3 THEN Q$="THREE"
110 CLS:PRINT:PRINT
120 IF Z=2 THEN PRINT "SCORE UP TO THIS
                      HOLE IS"X(1)
130 IF Z>2 THEN PRINT "SCORE UP TO THIS
                      HOLE IS"K
140 PRINT "<<< HOLE NUMBER"Z">>>"
150 PRINT:PRINT "DIFFICULTY FACTOR IS "Q$
160 GOSUB 430
170 PRINT:INPUT "ENTER STROKE STRENGTH"
                    ;A:SOUND 31,2
180 PRINT@U,L$:IF J>24 THEN A=-A
190 J=J+INT(A/RND(Q))
200 IF J=24 THEN GOSUB 490
205 IF J>30 THEN J=30:GOTO 205
207 IF J<1 THEN J=1
210 IF J<>24 THEN PRINT@U+J-1,H$
215 IF J<>24 THEN PRINT@352,L$:PRINT L$
220 SC=SC+1
230 PRINT@448,"AFTER THAT STROKE YOUR
                     SCORE IS"SC
240 FOR P=1 TO 2500:NEXT P
250 IF J<>24 THEN 110
260 C=C+SC
270 X(Z)=SC
280 IF Z=1 THEN 390
290 K=0
300 PRINT "THE GAME SO FAR:"
310 FOR J=1 TO Z
320 K=K+X(J)
330 PRINT "HOLE"J"TOOK JUST"X(J)"STROKES"
340 FOR M=1 TO 300:NEXT M
350 NEXT J
360 IF Z<9 THEN PRINT:PRINT "THE AVERAGE SO
              FAR IS"INT((K+.5)/Z)
370 FOR P=1 TO 1000:NEXT P
380 IF Z>1 THEN PRINT:PRINT "THE SCORE FOR"
              Z"HOLES IS"C
390 IF Z=1 THEN PRINT:PRINT "THE SCORE FOR
              THE FIRST HOLE IS"C
400 FOR M=1 TO 2500:NEXT M
410 NEXT Z
420 GOTO 560
```

1 of 2.

```
430 IF J>30 THEN J=30
435 PRINT@196,""
440 PRINT TAB(J-1);H$
450 PRINT "######################\ /######"
460 PRINT "####################### #######"
470 PRINT "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"
480 RETURN
490 PRINT@416,"YOU DID IT!!"
500 PRINT@311,H$
510 FOR P=1 TO 300:NEXT P
520 SOUND 21,4:SOUND 16,2:SOUND 16,1:
          SOUND 18,4:SOUND 16,4
530 SOUND 0,1:SOUND 20,4:S@UND 21,4
540 FOR P=1 TO 2000:NEXT P
550 RETURN
560 PRINT:PRINT "END OF THAT ROUND, GOLFER!"
570 PRINT:PRINT "YOU SCORED"C
580 PRINT "AND YOUR AVERAGE WAS"INT((C+.5)/9)
590 PRINT:PRINT
600 PRINT "ENTER 'Y' FOR ANOTHER ROUND, OR
                'N' TO QUIT"
610 A$=INKEY$
620 IF A$<> "Y" AND A$<> "N" THEN 610
630 IF A$="Y" THEN RUN
640 PRINT:PRINT "OK, THANKS FOR
                PLAYING, CHAMP"
```

Testing your Speed

Reaction Test, is great fun to play. You enter the program, type in RUN, and the message STAND BY appears. After an agonising wait, STAND BY will vanish, to be replaced with the words, "OKAY, HIT THE 'Z' KEY!". As fast as you can, you leap for the Z key and press it, knowing that the computer is counting all the time.

The computer tells you how quickly you reacted, and compares this with your previous best time. "THE BEST SO FAR IS . . ." appears on the screen, and the computer than waits for you to take your hands off the keyboard before the whole thing begins again.

The game continues until you manage to get your reaction time to below 10, which is not an easy task.

Line 20 sets the variable HS to 1000. The variable C is set to zero in Line 50 and incremented by one every time this line is revisited, which occurs when you have not managed to get to the 'Z' key. Lines 55 and 60 check to see if you have touched the Z key, and if not, send the program back to 50 where C is incremented.

Once you've managed to get to Z, the program 'falls through' to line 65 where you are told your score. This is compared with the best score (variable name HS) in the following line, and HS is adjusted to C if C is the lower of the two.

The next line (80) puts in a short pause, and then checks to make sure you have taken your hands off the keyboard. It stays cycling through 80 and 85 until you take your hands off the keys. The NEXT W then sends the program back to the line after the FOR (line 15) and the next round of the game begins.

The FOR/NEXT continues only so long as HS stays greater than 10 (as you can see in line 15). Once you get a high score below 11, the program continues through the NEXT to line 15 where the words "YOU'RE THE CHAMP!" appear on the screen, and SOUND 31, 1 is activated.

```
5 REM - REACTION TEST -
7 CLS
10 LET HS=1000
15 FOR W=0 TO 999: IF HS<10 THEN 90
20 PRINT@236,"STAND BY"
25 GOSUB 105
30 GOSUB 100
35 IF A$<>"" THEN 25
40 LET C=0
45 PRINT@134,"OKAY - HIT THE 'Z' KEY!"
50 LET C=C+1
55 GOSUB 100: IF C>=200 THEN GOTO 90
60 IF A$<>"Z" THEN 50
65 PRINT: PRINT "YOUR SCORE IS";C
70 IF C<HS THEN LET HS=C: SOUND 30,2
75 PRINT: PRINT "THE BEST SO FAR IS";HS
80 GOSUB 105: GOSUB 100
85 IF A$<>"" THEN 80
90 NEXT W
95 PRINT: PRINT "YOU'RE THE CHAMP!":
          SOUND 31,5: END
100 LET A$=INKEY$: RETURN
105 FOR P=0 TO 499+RND(999): NEXT P:
          CLS: RETURN
```

2 of 2.

VZ200

Graphic Sine Waves for VZ200.

By Dean Nickasen, Murrumbeena, VIC

This program will draw sinewaves in the graphic symbols of the VZ200. Lines 10 to 90 input the values for the sinewave. Lines 100 to 200 plot the graph. The purpose of line 210 is to keep the computer in the graphics mode.

To modify the program for other computers, lines 100 to 200 will have to be changed. Instead of setting points, a PRINT TAB(Z) statement will work. The program will also work on the VZ200 in this manner.

```
IO REM GRAPHIC SINE WAVES
20 REM BY DEAN NICLASEN
30 REM SEPTEMBER 1983
40 CLS
50 PRINT" ENTER THE LOWEST LIMIT FOR X";:INPUT A
60 PRINT
70 PRINT" ENTER THE UPPER LIMIT FOR X";:INPUT B
80 PRINT
90 PRINT ENTER EXPANSION AND SHIFT";:INPUT E,S
IOO MODE(1)
IIO FOR X=A TO B STEP(B-A)/80
I20 Y=2*COS(4*X-.349)+3*SIN(3*X+1.309)
I30 Z*E*Y+S
I40 IF Z>127 OR G>63 THEN 2IO
I50 COLOR 3,0
I60 SET(S+40,G)
I70 COLOR 2,0
I80 SET(Z+40,G)
I90 G=G+1
200 NEXT X
2IO GOTO 210
```

YC Jan. 84 p 65.

Moon Lander

by A Alley

This program is an arcade-type game for the VZ-200, and is fashioned after the video game of the same name. The aim is to land as many times as possible on the red landing pads provided without running out of fuel or crashing into the rocky landscape. The keys Y, G, and H are used to control the various motions of the ship.

The main outline of the program is as follows:

Line numbers 90 to 140 clear the preceding screen.

Line numbers 220 to 445 draw the landscape and landing pad.

Line numbers 500 to 620 handle actual game play.

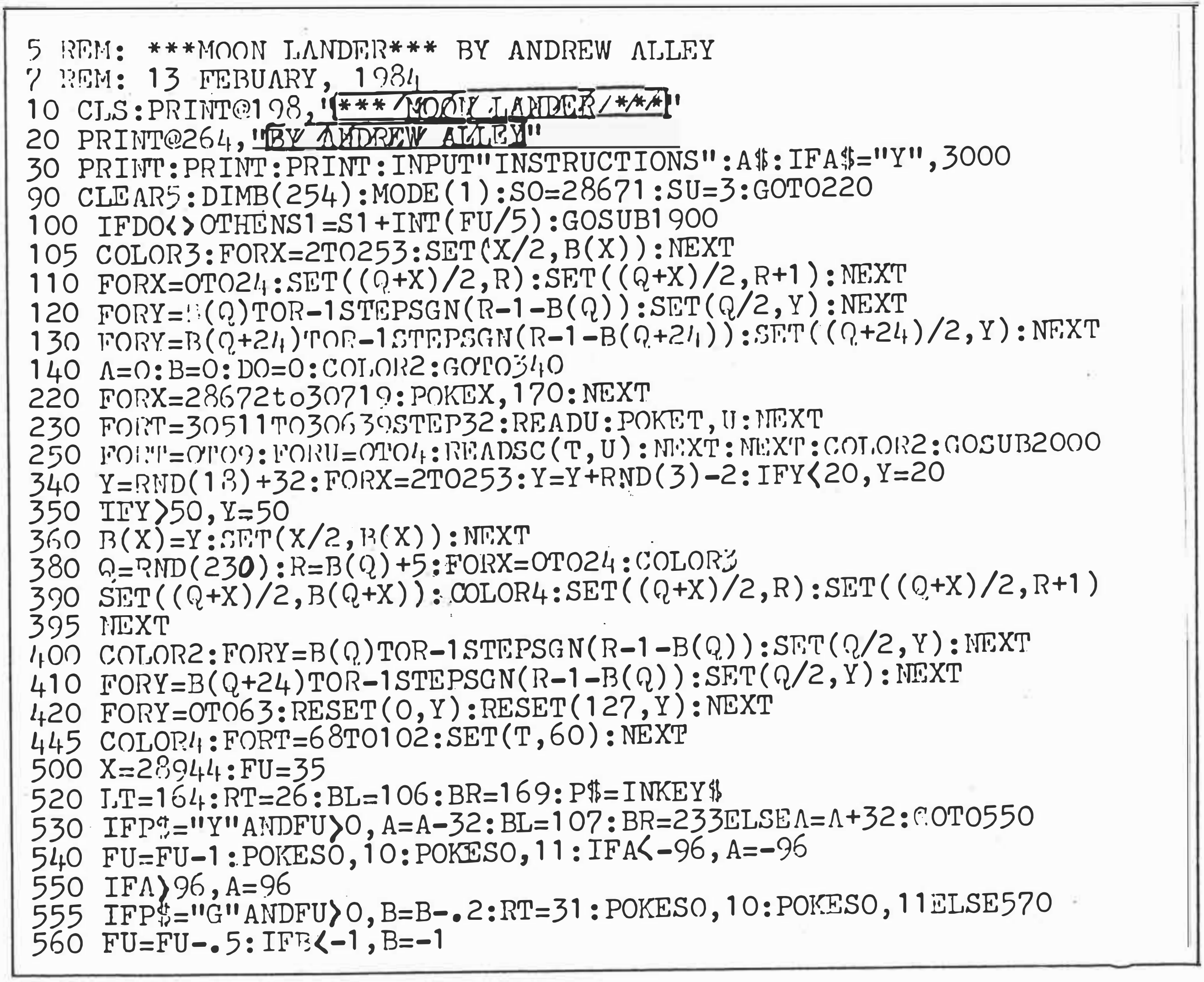
Line numbers 1000 to 1260 detect landings and crashes and take the appropriate course.

Line numbers 1400 to 1420 draw the ship.

Line numbers 1900 to 2020 are the subroutine to display the score, number of ships remaining and so on.

Line numbers 3000 to 3190 are instructions.

Care should be taken when piloting the space ship as it will drift after being moved in any direction. At the end of each successful mission, bonus points will be added to the score. It should be kept in mind that a player need not land on each landscape; he may simply thrust upwards to the top of the screen and another landscape will be drawn up. An extra ship will be awarded at each 100 points.



```
5 REM: ***MOON LANDER*** BY ANDREW ALLEY
7 REM: 13 FEBUARY, 1984
10 CLS:PRINT@198,"***MOON LANDER***"
20 PRINT@264,"BY ANDREW ALLEY"
30 PRINT:PRINT:PRINT:INPUT"INSTRUCTIONS":A$:IFA$="Y",3000
90 CLEAR5:DIMB(254):MODE(1):SO=28671:SU=3:GOTO220
100 IFDO<>OTHENS1=S1+INT(FU/5):GOSUB1900
105 COLOR3:FORX=2TO253:SET(X/2,B(X)):NEXT
110 FORX=OTO24:SET((Q+X)/2,R):SET((Q+X)/2,R+1):NEXT
120 FORY=B(Q)TOR-1STEPSGN(R-1-B(Q)):SET(Q/2,Y):NEXT
130 FORY=B(Q+24)TOR-1STEPSGN(R-1-B(Q+24)):SET((Q+24)/2,Y):NEXT
140 A=0:B=0:DO=0:COLOR2:GOTO340
220 FORX=28672to30719:POKEX,170:NEXT
230 FORT=30511TO30639STEP32:READU:POKET,U:NEXT
250 FORT=OTO9:FORU=OTO4:READSC(T,U):NEXT:NEXT:COLOR2:GOSUB2000
340 Y=RND(18)+32:FORX=2TO253:Y=Y+RND(3)-2:IFY<20,Y=20
350 IFY>50,Y=50
360 B(X)=Y:SET(X/2,B(X)):NEXT
380 Q=RND(230):R=B(Q)+5:FORX=OTO24:COLOR3
390 SET((Q+X)/2,B(Q+X)):COLOR4:SET((Q+X)/2,R):SET((Q+X)/2,R+1)
395 NEXT
400 COLOR2:FORY=B(Q)TOR-1STEPSGN(R-1-B(Q)):SET(Q/2,Y):NEXT
410 FORY=B(Q+24)TOR-1STEPSGN(R-1-B(Q)):SET(Q/2,Y):NEXT
420 FORY=OTO63:RESET(0,Y):RESET(127,Y):NEXT
445 COLOR4:FORT=68TO102:SET(T,60):NEXT
500 X=28944:FU=35
520 LT=164:RT=26:BL=106:BR=169:P$=INKEY$
530 IFP$="Y"ANDFU>0,A=A-32:BL=107:BR=233ELSEA=A+32:GOTO550
540 FU=FU-1:POKESO,10:POKESO,11:IFA<-96,A=-96
550 IFA>96,A=96
555 IFP$="G"ANDFU>0,B=B-.2:RT=31:POKESO,10:POKESO,11ELSE570
560 FU=FU-.5:IFB<-1,B=-1
```

APC Apr 84 5(4) p.178–180 1 of 2.

```
570 IFP$="H"ANDFU>0,B=B+.2:LT=244:POKESO,10:POKESO,11ELSE585
580 FU=FU-.5:IFB>1,B=1
585 FORX1=XTOX+128STEP32:POKEX1,170:POKEX1+1,170:NEXT
590 X=X+A+B:FOR X1=XTOX+128STEP32
600 IFPEEK(X1)<>170ORPEEK(X1+1)<>170,1000
610 NEXT:IFX<28800,100
620 GOSUB1400:RESET(FU+68,60):GOTO520
1000 IFPEEK(X1)<>255ORPEEK(X1+1)<>255,1050
1005 IFA>64,1050
1010 DO=DO+1:IFDO<>1,1040
1012 IFPEEK(X+128)<>170ORPEEK(X+129)<>170,X=X-32:GOTO1012
1020 GOSUB1400:POKEX+1,74:POKEX-31,130:POKEX-63,138
1025 SOUND4,5:SOUND11,5:SOUND16,5:SOUND20,3:SOUND19,6
1027 POKEX-31,170:POKEX-63,170
1030 S1=S1+5:FORX1=XTOX+128STEP32:POKEX1,170:POKEX1+1,170:NEXT
1040 GOSUB1900:IFFU<=0,1100
1045 X=X-32:A=0:B=0:GOTO590
1050 IFPEEK(X1)=42ORPEEK(X1+1)=168,X=X-B:B=0:GOTO590
1100 GOSUB1400:FORT=1TO8:E(T)=X+32+INT(T/4):F(T)=RND(2)*32
1105 G(T)=(T-4)*.1:NEXT
1110 FORT1=1TO12:FORT=1TO8:POKEE(T),170:E(T)=E(T)-F(T)+G(T)
1120 POKEE(T),190:POKESO,10:POKESO,11:NEXT:NEXT
1125 FORT=1TO8:POKESO,10:POKESO,11:FORT1=1TO15:NEXT:NEXT
1220 FORX1=XTOX+128STEP32:POKEX1,170:POKEX1+1,170:NEXT
1230 FORT=1TO8:POKEE(T),170:NEXT:SU=SU-1
1250 IFSU =0,GOSUB1900:FORT=1TO2000:NEXTELSE1260
1252 PRINT@236,"GAME OVER":PRINT:PRINT"SCORE";S1+S2*10+S3*100
1255 SOUND12,8:PRINT:INPUT"ANOTHER GAME"; A$:IFA$="Y",RUNELSEEND
1260 GOSUB1900:GOTO100
1400 POKEX,165:POKEX+1,90:POKEX+32,144:POKEX+33,6:POKEX+64,LT
1410 POKEX+65,RT:POKEX+96,152:POKEX+97,38:POKEX+128,BL
1420 POKEX+129,BR:RETURN
1900 IFS1>9,S1=S1-10:S2=S2+1:GOTO1900
1910 IFS2>9,S2=S2-10:S3=S3+1:SU=SU+1:GOTO1910
1920 IFS3>9,S3=S3-10:GOTO1920
2000 I=-1:FORH=30500TO30628STEP32:I=I+1:POKEH.SC(S3,I)
2010 POKEH+1,SC(S2,I):POKEH+2,SC(S1,I)
2020 POKEH+5,SC(SU,I):NEXT:RETURN
2495 DATA42,42,10,42,42,86,102,102,102,86,154,90,154
2500 DATA154,86,86,166,86,106,86,86,166,150,166,86,102,102,86
2505 DATA166,166,86,106,86,166,86,86,106,86,102,86,86,102,166
2510 DATA166,166,86,102,86,102,86,86,102,86,166,166
3000 CLS:PRINTTAB(9)"**MOON LANDER**"
3020 PRINT:PRINT" PILOT A SPACE MODULE ONTO THE"
3030 PRINT"      SURFACE OF THE MOON.":PRINT
3040 PRINT"YOU MUST LAND ON THE LANDING PAD"
3050 PRINT" THE CRAFT WILL DRIFT WHEN YOU"
3070 PRINT"     THRUST IN ANY DIRECTION.":PRINT
3080 PRINT"YOU ARE AWARDED AN EXTRA MODULE"
3085 PRINT" UPON REACHING EACH 100 POINTS":PRINT
3090 PRINT"        WATCH YOUR FUEL!":PRINT
3095 PRINTTAB(8)"PRESS ANY KEY";
3100 FORT=1TO10:A$=INKEY$:NEXT
3110 IFINKEY$="",3110
3115 CLS:PRINTTAB(9)"**MOON LANDER**"
3120 PRINT:PRINT"CONTROLS:":PRINT
3130 PRINT"    *W* ..........MAIN THRUSTER":PRINT
3140 PRINT"     *Z* ....LEFT AUX. THRUSTER":PRINT
3150 PRINT"      *M* ...RIGHT AUX. THRUSTER":PRINT
3155 PRINT"  BONUS POINTS AWARDED FOR FUEL  REMAINING.":PRINT
3160 PRINTTAB(11)"GOOD LUCK!"
3170 PRINT:PRINTTAB(9)"PRESS ANY KEY";
3180 FORT=1TO10:A$=INKEY$:NEXT
3190 IFINKEY$="",3180ELSE90
```

APC 5(4) Apr 84 p 178-180

2 of 2.

Disassembly of lines 10-30 for USR.

30862, 241 ≡ F1 H

30863, 143 ≡ 8F H. Start add. 8FF1 H

-28687 ≡ 36849 ≡ 8FF1 } 14 bytes

-28674 ≡ 36862 ≡ 8FFE

```
                                          #28672D
FF1   21  ØØ  7Ø    LD HL,7ØØØH         ; start video RAM
                                          #28673D
F4    11  Ø1  7Ø    LD DE,7ØØ1H         ; next
                                          #2047D
F7    Ø1  FF  Ø7    LD BC,Ø7FFH         ; size of video RAM.
                                          #85D
FA    36  55        LD (HL),55H         ; color byte.
FC    ED  BØ        LDIR                ; Block move.
                                        ; Repeat LDI until
                                          BC=Ø.
FE    C9            RET
```

NB: LDI — i, assign (HL) to (DE)
ii, inc HL
iii, inc DE
iv, dec BC

LDIR — repeat LDI until BC=Ø.

Blockout

by B Pritchard

Blockout is a game for the unexpanded VZ 200 which will work with joysticks or from the keyboard. The object of the game is to trap your opponent by boxing him/her/it, in with the lethal trail that you (and your opponent) leave as you move around the screen.

The main points of the program are:

- Lines 10 to 30 are a short machine language which will set the whole screen white when called.
- Lines 185 to 190 initialise the variables.
- Line 195 sets up the screen.
- Line 200 checks to see if the computer has to move (otherwise it gets the players move from the keyboard or the left joystick).
- Lines 205 to 240 process the left player's movements.
- Line 245 collects the right player's move from the keyboard or the right joystick.
- Lines 250 to 285 process the right player's move.
- Lines 300 to 325 check if either player has hit a line or run off the edge of the screen.
- Lines 400 to 440 calculate and display each player's score.
- Lines 500 to 595 control the computer's movements.
- Lines 1000 onwards are the instructions and keyboard controls.

```
 0 REM ******************
 1 REM **    BLOCKOUT    **
 2 REM **      BY        **
 3 REM ** B.PRITCHARD   **
 4 REM **    29/4/84    **
 5 REM ******************
 10 FORI=-28687TO-28674
 15 READA:POKEI,A
 20 NEXT
 25 DATA33,0,112,17,1,112,1,255,7,54,85,237,176
   ,201
 30 POKE30862,241:POKE30863,143
 35 CLS:PRINTTAB(7)"*** BLOCKOUT ***":PRINT
 40 INPUT"INSTRUCTIONS";A$
 45 IFLEFT$(A$,1)="Y"THEN1000
 50 INPUT"ONE OR TWO PLAYERS";PL
 55 IFPL<>1ANDPL<>2THEN5Ø
 60 IFPL=2THEN75
 65 RI$="YOU":LE$="I"
 70 GOTO185
 75 INPUT"LEFT PLAYERS NAME";LE$
```

APC 5(7) Jul 84 p 174, 175 & 178.
1 of 3.

```
80 INPUT"RIGHT PLAYERS NAME";RI$
185 X1=1:Y1=Ø:X2=-1:Y2=Ø
190 AX=0:AY=32:BX=127:BY=32
195 MODE(1):X=USR(Ø)
200 IFPL=1THEN500ELSEA=PEEK(27ØØØ):AA=(INP(43)
    AND31)
205 IFA=239THENX1=-1:Y1=Ø
210 IFA=253THENX1=1:Y1=Ø
215 IFA=247THENX1=Ø:Y1=-1
220 IFA=223THENX1=Ø:Y1=1
225 IFAA=27THENX1=-1:Y1=Ø
230 IFAA=23THENX1=1:Y1=Ø
235 IFAA=3ØTHENX1=Ø:Y1=-1
240 IFAA=29THENX1=Ø:Y1=1
245 B=PEEK(267ØØ):BB=(INP(46)AND31)
250 IFB=223THENX1=-1:Y1=Ø
255 IFB=247THENX1=1:Y1=Ø
260 IFB=253THENX1=Ø:Y1=-1
265 IFB=239THENX1=Ø:Y1=1
270 IFBB=27THENX1=-1:Y1=Ø
275 IFBB=23THENX1=1:Y1=0
280 IFBB=3ØTHENX1=Ø:Y1=-1
285 IFBB=29THENX1=Ø:Y1=1
300 AX=AX+X1:AY=AY+Y1
305 IFAX<ØORAX>127ORAY<ØORAY>63THEN4ØØ
310 IFPOINT(AX,AY)<>2THEN4ØØ
315 BX=BX+X2:BY=BY+Y2
320 IFBX<ØORBX>127ORBY<ØORBY>63THEN405
325 IFPOINT(BX,BY)<>2THEN405
330 COLOR3:SET(AX,AY)
335 COLOR4:SET(BX,BY)
340 GOTO200
400 BS=BS+1:W$=RI$:GOTO41Ø
405 AS=AS+1:W$=LE$
410 CLS:PRINTWS;" WON":PRINT
415 PRINT"LEFT SCORE","RIGHT SCORE"
420 PRINT:PRINTTAB(3)AS,TAB(3)BS
425 PRINT@451,"PRESS ANY KEY TO CONTINUE"
426 PRINTTAB(1Ø)"(N=NEW GAME)"
430 A$-INKEY$:IFA$=""THEN430
435 IFINKEY$=A$ORINKEY$=""THEN435
440 IFINKEY$="N"THENRUNELSEPOKE27ØØØ,0:POKE267ØØ,Ø
    :GOTO185
500 IFRND(4Ø)<>1THEN51Ø
505 IFRND(2)=1THENX1=RND(3)-2:Y1=ØELSEX1=Ø
506 IFX1=ØTHENY1=RND(3)-2
510 IFAX+X1<ØORAX+X1>127ORAY+Y1<ØORAY+Y1>63THEN
    525
515 IFPOINT(AX+X1,AY+Y1)=2THEN245
525 IFAX-1<ØORAY<ØORAY>63THENA1=1ELSEIFPOINT
    (AX-1,AY)<>2,A1=1
530 IFAX+1>127ORAY<ØORAY>63THENA2=1ELSEIFPOINT
    (AX+1,AY)<>2,A2=1
535 IFAY-1<ØORAX<ØORAX>127THENA3=1ELSEIFPOINT
```

APC 5(7) Jul 84.
2 of 3.

```
    (AX,AY-1)<>2,A3=1
540 IFAY+1>63ORAX<ØORAX>127THENA4=1ELSEIFPOINT
    (AX,AY+1)<>2,A4=1
545 IFA1=1ANDA2=1ANDA3=1ANDA4=1THENA1=Ø:A2=Ø
    A3=Ø:A4=Ø:GOTO245
546 A1=Ø:A2=Ø:A3=Ø:A4=Ø
55Ø R=RND(4)
555 IFR=1ANDAX-1<-1ANDAY<-1ANDAY>64THENIFPOINT
    (AX-1,AY)=2,58Ø
56Ø IFR=2ANDAX+1 >128ANDAY<-1ANDAY>64THENIFPOINT
    (AX+1,AY)=2,585
565 IFR=3ANDAY-1<-1ANDAX<-1ANDAX>128THENIFPOINT
    (AX,AY-1)=2,59Ø
57Ø IFR=4ANDAY+1>64ANDAX<-1ANDAX>128THENIFPOINT
    (AX,AY+1)=2,595

575 GOTO55Ø
58Ø X1=-1:Y1=Ø:GOTO245
585 X1=1:Y1=Ø:GOTO245
59Ø X1=Ø:Y1=-1:GOTO245
595 X1=Ø:Y1=1:GOTO245
1ØØØ PRINT@64," AS YOU MOVE AROUND THE SCREEN"

1ØØ5 PRINT"YOU WILL LEAVE A TRAIL."
1Ø1Ø PRINT" YOU CANNOT RUN INTO YOUR TRAIL"
1Ø15 PRINT",OR YOUR OPPONENTS TRAIL,OR RUN"


1Ø2Ø PRINT"OFF THE EDGE OF THE SCREEN."
1Ø25 PRINT"(DOUBLING BACK INTO YOURSELF IS"
1Ø3Ø PRINT"THE SAME AS RUNNING INTO YOUR","TRAIL"
1Ø35 PRINT" WHEN PLAYING ONE PLAYER ONLY"
1Ø4Ø PRINT"(AGAINST THE COMPUTER),USE THE"
1Ø45 PRINT"RIGHT SET OF CONTROLS"
1Ø5Ø PRINT@48Ø,"PRESS ANY KEY TO CONTINUE";
1Ø55 A$=INKEY$:IFA$=""THEN1Ø55
1Ø6Ø IFINKEY$=A$ORINKEY$=""THEN1Ø6Ø
1Ø65 CLS:PRINTTAB(6)"KEYBOARD CONTROLS"
1Ø7Ø PRINT:PRINT"RIGHT PLAYER:"


1Ø75 PRINTTAB(14)"(M)=LEFT"
1Ø8Ø PRINTTAB(14)"(,)=RIGHT"
1Ø85 PRINTTAB(14)"(.)=UP"
1Ø9Ø PRINTTAB(14)"(SPACE)=DOWN"
1Ø95 PRINT:PRINT"LEFT PLAYER:"
11ØØ PRINTTAB(14)"(Z)=LEFT"
11Ø5 PRINTTAB(14)"(X)=RIGHT"
111Ø PRINTTAB(14)"(C)=UP"

1115 PRINTTAB(14)"(V)=DOWN"
112Ø PRINT@ 48Ø,"PRESS ANY KEY TO CONTINUE";
1125 A$=INKEY$:IFA$=""THEN1125
113Ø IFINKEY$=A$ORINKEY$=""THEN113Ø
1135 GOTO35
```

APC 5(7) Jul 84
3 of 3.

Micro-80 Jul 84 4(7)
P 7 and 22.

BATTLESHIPS (VZED 8K)

This is the old board game of Battleships and cruisers. The screen is divided into a 9 x 9 grid. The computer 'hides' a total of 10 ships at random around this grid. There are four types of ships — 1 Battleship which occupies four adjacent squares, two Cruisers which occupy three adjacent squares each, three destroyers which occupy two adjacent squares each and four submarines occupying yes, you've got it, one square each.

You must enter the coordinates of a square in the grid, at which time the computer prints either a letter in that square, denoting the type of vessel hit, or will print an asterisk if the square is empty. The object of the game is to sink all the vessels with the least possible number of shots. Good hunting!

BATTLESHIPS VZ 200

```
3 CLS:COLOR,1:PRINT@170,"FOR VZ-200"
4 PRINT@201,"BY R. CARSON":PRINT@235,"ADELAIDE"
5 PRINT@33,"***THE GAME OF BATTLESHIPS***":REM COPYRIGHT
6 PRINT@425,"INSTRUCTIONS?":PRINT@456,">>Y=YES  N=NO<<"
7 K$=INKEY$
8 I$=INKEY$:IF I$=""THEN8
9 IF I$="Y"THEN 12
10 IF I$="N"THEN90
11 IF I$<>"Y"THEN7:IF I$<>"N"THEN7
12 CLS:PRINT"THE PLAYING AREA REPRESENTS AN  ";
13 PRINT"AREA OF SEA. THE COMPUTER IS    ";
14 PRINT"CONTROLLING TEN SHIPS, A BATTLE-";
15 PRINT"SHIP, 2 CRUISERS, 3 DESTROYERS  ";
16 PRINT"AND 4 SUBMARINES. OF COURSE, I  ";
17 PRINT"CAN'T TELL YOU WHERE THEY ARE,  ";
18 PRINT"ONLY THE COMPUTER KNOWS, UNTIL  ";
19 PRINT"YOU HIT THEM. THE SHIPS ARE     ";
20 PRINT"DIFFERENT SIZES, AND ARE IDENTI-";
21 PRINT"FIED BY THE INITIAL LETTER. THE ";
22 PRINT"BATTLESHIP OCCUPIES FOUR SQUARES";
23 PRINT"LIKE THIS: BBBB, ACROSS OR DOWN."
24 PRINT:PRINT"   PRESS <SPACE> TO CONTINUE"
25 K$=INKEY$
26 I$=INKEY$:IF I$<>" "THEN 26
32 CLS:PRINT"THE CRUISERS THREE SQUARES, THE ";
33 PRINT"DESTROYERS TWO SQUARES, AND THE ";
34 PRINT"SUBMARINES ONE SQUARE, ALWAYS IN";
35 PRINT"A STRAIGHT LINE. SHIPS MAY TOUCH";
36 PRINT"OR LAY ALONGSIDE EACH OTHER.YOU ";
37 PRINT"FIRE A SHOT BY GIVING TWO       ";
38 PRINT"NUMBERS. THE FIRST ON THE LEFT, ";
39 PRINT"THE SECOND AT THE TOP. IF YOU   ";
40 PRINT"HIT ANYTHING, A LETTER WILL BE  ";
41 PRINT"PRINTED TO TELL YOU WHICH TYPE  ";
42 PRINT"OF SHIP YOU HIT. TO SINK IT, YOU";
43 PRINT"MUST HIT ALL THE SQUARES OF     THAT ";
44 PRINT"PARTICULAR SHIP."
45 PRINT:PRINT"   PRESS <SPACE> TO CONTINUE"
46 K$=INKEY$
47 I$=INKEY$:IF I$<>" "THEN47
52 CLS:PRINT"IF YOU MISS, THEN * IS PRINTED  ";
53 PRINT"TO REMIND YOU THAT YOU HAVE SHOT";
54 PRINT"INTO THAT SQUARE BEFORE."
55 PRINT:PRINT"YOUR NUMBER OF SHOTS IS SHOWN AT";
56 PRINT"THE BOTTOM OF THE SCREEN AND THE";
57 PRINT"BEST SCORE YOU ACHIEVED DURING A";
58 PRINT"SERIES OF GAMES. THE GAME ENDS  ";
59 PRINT"WHEN ALL SHIPS HAVE BEEN SUNK."
60 PRINT:PRINT:PRINT"           HAPPY HUNTING"
61 PRINT:PRINT:PRINT"      PRESS <SPACE> TO START"
62 K$=INKEY$
63 I$=INKEY$:IF I$<>" "THEN 63
90 CLS
95 X=0
100 A=100
110 DIM G(100)
120 D=0
125 CLS:PRINT@196,"WAIT---ARRANGING FLEET"
130 C=0
140 FOR B=1 TO 100
150 G(B)=0
160 NEXT B
170 E=4
180 F=1
190 H=INT(RND(X)*2)
195 W=0
200 IF H=0 THEN J=RND(9)
202 IF H=1 THEN J=RND(4)
205 IF H=1 THEN K=RND(9)
212 IF H=0 THEN K=RND(4)
220 L=0
230 P=10*J+K
250 FOR M=0 TO (E-1)
255 IF H=0 THEN R=P+M
260 IF H=1 THEN R=P+10*M
280 IF L=0 AND G(R)<>0 THEN W=W+1
290 IF L=1 THEN G(R)=E
300 NEXT M
305 IF W>0 AND W<10 THEN 190
306 IF W=10 THEN 140
310 IF L=1 THEN 400
320 L=1
330 GOTO 250
400 F=F+1
410 IF F<4 THEN E=3
420 IF F>3 AND F<7 THEN E=2
430 IF F>6 THEN E=1
440 IF F=11 THEN 700
445 GOTO 190
450 PRINT@435,"      ";:INPUTS
458 IF S<11 THEN 450
460 IF S>99 THEN 450
465 T=INT((S)/10)
470 U=S-T*10
472 IF G(S)=5 THEN 450
475 IF G(S)=4 THEN S1$="B"
480 IF G(S)=3 THEN S1$="C"
485 IF G(S)=2 THEN S1$="D"
490 IF G(S)=1 THEN S1$="S"
495 IF G(S)=0 THEN S1$="*"
500 V=U*2+T*32+101
510 PRINT@V,S1$
520 C=C+1
530 IF G(S)>0 THEN D=D+1
535 G(S)=5
540 PRINT@418,"SHOTS:";C;
550 IF D<20 THEN 450
560 IF A<82 THEN PRINT@457,"BEST SCORE:";A
570 IF C<A THEN A=C
580 PRINT@482,"ANOTHER GAME?>>Y=YES  N=NO<<";
585 K$=INKEY$
590 I$=INKEY$:IF I$=""THEN590
595 IF I$="Y"THEN120
597 IF I$="N"THENCLS:END
600 IF I$<>"Y"THEN 585
610 IF I$=<>"N"THEN 585
700 CLS:PRINT@39,"***BATTLESHIPS***"
720 PRINT:PRINTTAB(7)"1 2 3 4 5 6 7 8 9"
730 FOR N=1 TO 9
740 PRINTTAB(4);N;". . . . . . . . ."
750 NEXT N
760 GOTO 450
```

JUNIOR MATHS (VZED 8K)

This program tests the four basic mathematical functions: Addition, Division, Subtraction and Multiplication. Whilst not an educational program in the strictest sense, it does serve to reinforce lessons already learnt. You are first asked to choose the type of problem after which a graphics screen is presented with an area for the questions and answers and a representation of a persons head with a non-committal expression and some ominous blue water at the bottom. 10 questions are presented one at a time. A correct answer is rewarded by a smile and some uplifting music whilst an incorrect answer causes a frown and depressing music. In this event, the correct answer is also displayed. When the ten questions have been presented, your score and percentage correct are shown.

Now comes the odd bit which may cause our mailbags to bulge with irate letters from outraged child psychologists. In the original version, the author "punished" an imperfect score by raising the water level until it covered the head. He soon found that children using it would deliberately enter incorrect answers just to see this happen. So he reversed the procedure. Now to submerge the hapless head, one must get a perfect score! By the way, the level of difficulty is appropriate to children aged from 9-11.

JUNIOR MATHS VZ 200

```
6 CLEAR1000:CLS:COLOR.1:REM COPYRIGHT - R. CARSON - 1983.
7 FORPO=0TO223 STEP1:PRINT@PO.CHR$(160);:NEXT
10 COLOR 6
20 PRINT@67," █  █   ████ ████ █████ "
30 PRINT@99," █  █▄▄   ▄█ █▄▄  █   █ "



40 PRINT@131." ██ ▀▀ █▀▀ █▀▀ █ █"
41 PRINT@163."  █   ████ ████ ████ ":SOUND30.2
42 PRINT@256,"                                      "
45 FORL=1TO800:NEXT
46 A$="       WRITTEN BY R. CARSON       "
47 FORL=1TOLEN(A$)
48 PRINT@256,RIGHT$(A$,L);:NEXT
49 FORT=1TO2500:NEXT
50 T$="       WRITTEN BY R. CARSON       "
51 FORP=LEN(T$)TO1STEP-1:PRINT@256.RIGHT$(T$.P):NEXT
57 B$="    ENJOY THIS EDUCATIONAL GAME  "
58 FORL=1TOLEN(B$)
59 PRINT@256,RIGHT$(B$,L);:NEXT
60 FORJ=1TO2500:NEXT
61 T$="    ENJOY THIS EDUCATIONAL GAME  "
62 FORL=LEN(T$)TO1STEP-1:PRINT@256.RIGHT$(T$.L):NEXT
63 FORI=1TO800:NEXTI
70 SOUND20.3:PRINT"    YOUR CHOICE OF PROBLEMS"
71 PRINT:PRINT"          A = ADDITION"
72 PRINT"          D = DIVISION"
73 PRINT"          S = SUBTRACTION"
74 PRINT"          M = MULTIPLICATION"
79 K$=INKEY$
80 A$=INKEY$:IFA$=""THEN80
81 IFA$="M"GOTO60662
82 IFA$="D"GOTO60665
83 IFA$="A"GOTO60669
84 IFA$="S"GOTO60672
85 IFA$<>"M"ANDA$<>"D"ANDA$<>"A"ANDA$<>"S"THEN76
89 REM
100 C=0:G=0:P=0
101 CLS:COLOR,0
110 COLOR7:PRINT@32,"      ██████   ███████████████"
120 COLOR7:PRINT@64," █▄▄███████▄▄█ ███████████████"
125 COLOR 2
130 PRINT@97,"   ▄█████▄   ":COLOR7:PRINT@110,"████████████████"
132 COLOR 2
135 PRINT@129," ▄██ ███ ██▄ ":COLOR7:PRINT@142."████████████████"
140 COLOR2
145 PRINT@161,"███████████":COLOR7:PRINT@174."████████████████"
150 PRINT@193,"▀▀▀▀▀ ▀████":COLOR7:PRINT@206."████████████████"
155 COLOR 2
160 PRINT@225,"███████████":COLOR7:PRINT@238,"████████████████"
165 COLOR 2
170 PRINT@257,"  █▀▀▀▀▀▀█  ":COLOR7:PRINT@270,"████████████████"
175 COLOR2
180 PRINT@289."   ▀███▀   ":COLOR7:PRINT@302,"████████████████"
185 COLOR2
190 PRINT@321."    ███    ":COLOR7:PRINT@334."████████████████"
195 COLOR 4
200 PRINT@353."███████████":COLOR7:PRINT@366,"████████████████"
203 COLOR7:PRINT@398,"████████████████"
205 COLOR 3
207 PRINT@385,"███████████"
210 PRINT@417,"███████████":COLOR7:PRINT@430,"████████████████"
215 COLOR 3
220 PRINT@449."███████████":COLOR7:PRINT@462,"████████████████"
223 SOUND30,5
225 COLOR,0
```

Micro-80 Jul 84 4(7): p 7, 20-21.
1 of 2.

```
228 IFA$="D"TNENPRINT@83,"DIVISION":SOUND30,2
229 IFA$="A"THENPRINT@83,"ADDITION":SOUND30,2
230 IFA$="S"THENPRINT@81,"SUBTRACTION":SOUND30,2
231 IFA$="M"THENPRINT@79,"MULTIPLICATION":SOUND30,2
250 PRINT@208,"I WILL ASK YOU":SOUND30,3
252 COLOR2:PRINT@129," ██ ███ █ "
255 PRINT@240,"SOME PROBLEMS," SOUND30,3
256 COLOR2:PRINT@129," ██ ███ █ "
257 PRINT@272,"IF YOU GET";QW ;SOUND30,3
258 COLOR2:PRINT@129," ██ ███ █ "
260 PRINT@304."CORRECT, THE  ":SOUND30,3
262 COLOR2:PRINT@129," ██ ███ █ "
265 PRINT@336,"WATER WILL GET":SOUND30,3
266 COLOR2:PRINT@129," ██ ███ █ "
267 PRINT@368,"DEEPER.        ":SOUND30,3
268 COLOR2:PRINT@129," ██ ███ █ "
270 FORI=1TO5000:NEXTI
273 COLOR7
274 PRINT @79,"██████████████"
275 PRINT@176,"████████████"
276 PRINT@208,"████████████"
277 PRINT@240,"████████████"
278 PRINT@272,"████████████"
279 PRINT@304,"████████████"
280 PRINT@336,"████████████"
290 PRINT@368,"████████████"
60020 COLOR,0
60022 IFA$="M"GOSUB60700
60030 IFA$="D"GOSUB60710
60035 IFA$="A"GOSUB60720
60040 IFA$="S"GOSUB60730
60045 COLOR 2:PRINT@257," ██▀▀▀▀██ "
60050 IFA$="M"THENA=Y*Z:PRINT@176,Y"X"Z"="
60055 IFA$="D"THENB=Y*Z:A=Y:PRINT@176,B"-:-"Z"="
60060 IFA$="A"THENA=V+W  :PRINT@176,V"+W"="
60065 IFA$="S"THENJ=V+W:A=V:PRINT@176,J"-"W"="
60110 PRINT@240,"ANSWER";:INPUTD$
60115 FORZ=1TOLEN(D$)
60119 NEXTZ
60120 X=VAL(D$)
60125 IFX=ATHENC=C+1:SOUND25,2:PRINT@82,C;"CORRECT"
60130 COLOR 2:PRINT@257," ██▀▀▀▀██ "
60136 IFC=QW THEN60261
60140 IFX<>ATHEN60175
60150 FORT=1TO1000
60155 NEXTT
60160 COLOR7:PRINT@175,"████████████"
60165 COLOR7:PRINT@304,"█████████":PRINT@239,"████████████████"
60170 COLOR7:PRINT@240,"█████████":GOTO60020
60175 COLOR2:PRINT@257," ██▀▀▀▀██ "
60180 SOUND 16,3
60190 SOUND 11,2
60200 SOUND 11,1
60210 SOUND 13,3
60220 SOUND 11,3
60230 SOUND 0,2
60240 SOUND 15,4
60250 SOUND 16,4
60251 PRINT@368,"ANSWER IS"A:G=G+1:FORV=1TO2500:NEXT
```

```
60253 COLOR7:PRINT@175,"████████████"
60255 COLOR7:PRINT@335,"██████"
60256 COLOR7:PRINT@367,"████████████████"
60257 COLOR7:PRINT@240,"████████████████"
60260 GOTO60020
60261 FORI=1TO1500:NEXTI:COLOR7:SOUND20,3
60262 PRINT @82,"████████████"
60263 PRINT@176,"████████████"
60264 PRINT@208,"██████I█████"
60265 PRINT@240,"████████████"
60266 PRINT@272,"█████DID█████"
60267 PRINT@304,"████████████"
60268 PRINT@336,"███PROMISE███"
60269 PRINT@368,"████████████":FORI=1TO1500:NEXTI
60270 SOUND20,2:COLOR3:PRINT@358,"██":FORI=1TO200:NEXTI
60271 COLOR3:PRINT@357,"████":FORI=1TO200:NEXTI
60272 COLOR3:PRINT@356,"█████":FORI=1TO200:NEXTI
60273 COLOR3:PRINT@355,"██████":FORI=1TO200:NEXTI
60274 COLOR3:PRINT@354,"████████":FORI=1TO200:NEXTI
60275 COLOR2:PRINT@257," ██▀▀▀▀██ "
60276 COLOR3:PRINT@353,"████████████":FORI=1TO200:NEXTI
60278 COLOR3:PRINT@321,"████████████":FORI=1TO200:NEXTI
60280 COLOR3:PRINT@289,"████████████":FORI=1TO200:NEXTI
60290 COLOR3:PRINT@257,"████████████":FORI=1TO200:NEXTI
60300 COLOR3:PRINT@225,"████████████":FORI=1TO200:NEXTI
60305 COLOR2:PRINT@129," ██▀▀▀▀██ "
60310 COLOR3:PRINT@193,"████████████":FORI=1TO200:NEXTI
60320 COLOR3:PRINT@161,"████████████":FORI=1TO200:NEXTI
60330 COLOR3:PRINT@129,"████████████":FORI=1TO200:NEXTI
60340 COLOR3:PRINT@97, "████████████"
60350 FORI=1TO1500:NEXTI
60351 QW  =0
60352 SOUND20,3:PRINT@210,C;"CORRECT"
60353 PRINT@274,G;"WRONG"
60354 PRINT@338,INT(C*100/(C+G));"PERCENT"
60355 FORI=1TO1500:NEXTI
60356 COLOR7:PRINT@337,"████████████"
60359 COLOR 7
60360 SOUND30,2:PRINT@208,"███ ENTER  Y ███"
60370 PRINT@271,"█ TO PLAY AGAIN"
60380 PRINT@334,"█ N TO FINISH ";:INPUTC$
60390 IFC$="Y"THEN 60750
60400 CLS:PRINT:PRINT:PRINT:PRINT"                    BYE":END
60662 Y=RND(12):Z=RND(12)
60663 QW=RND(50):IFQW<10THENQW=10
60664 GOTO100
60665 Y=RND(12):Z=RND(12)
60666 QW=RND(50):IFQW<10THENQW   =10
60668 GOTO100
60669 V=RND(100):W=RND(100)
60670 QW=RND(50):IFQW<10THENQW   =10
60671 GOTO100
60672 V=RND(100):W=RND(100)
60673 QW=RND(50):IFQW<10THENQW=10
60675 GOTO100
60700 Y=RND(12):Z=RND(12):RETURN
60710 Y=RND(12):Z=RND(12):RETURN
60720 V=RND(100):W =RND100):RETURN
60730 V=RND(100):W=RND(100):RETURN
60750 CLS:PRINT:PRINT:PRINT:GOTO70
```

Micro-80 Jul 84 4(7): p 7, 20-21.
2 of 2

CONTEST LOG (VZED)

```
170 CLEAR 2000
180 DIM C1$(2000)
190 CLS
200 REM
210 CLS:PRINT:PRINT" NEXT CALL SIGN, SEE BELOW":PRINT
211 PRINT:PRINTTAB(3);" LIST  :- LIST WITHOUT SORT"
212 PRINT:PRINTTAB(3);" SORT  :- SORT CALL SIGNS"
213 PRINT:PRINTTAB(3);" PRINT:- LIST ON PRINTER"
214 PRINT:PRINTTAB(3);" END   :- END PROG."
215 PRINT:PRINTTAB(3);"       :- ENTER CALLSIGN"
216 PRINT:PRINT" ENTER :- ";:INPUT A1$
220 IF A1$="SORT" THEN 500
230 IF A1$="LIST" THEN 700
235 IF A1$="END" THEN CLS:END
236 IF A1$="PRINT" THEN 950
240 FOR I=1TO LEN(A1$)
245 NEXT I
260 CLS
270 REM
280 FOR I=1 TO N
290 IF A1$=C1$(I) THEN 400
300 NEXT I
310 REM
320 N=N+1
325 C1$(N)=A1$
330 PRINT:PRINT:PRINT TAB(3)" ";A1$;" IS NEW CALL SIGN"
340 PRINT:PRINT:PRINT TAB(5)" ";N;" CALLS LOGGED "
345 FOR X = 1 TO 1000
350 NEXT X
360 GOTO 200
400 REM
410 PRINT:PRINT:PRINT TAB(4)" ";A1$;" ALREADY LOGGED"
420 GOTO 340
500 REM
510 CLS:PRINT:PRINT TAB(12);"SORTING":PRINT
520 FOR I=1 TO N
530 A1$=C1$(I)
540 PRINT "*";
550 FOR J=I TO N

560 IF A1$<=C1$(J) THEN 580
570 B1$=C1$(J)
572 C1$(J)=A1$
574 A1$=B1$
580 NEXT J
590 C1$(I)=A1$
600 NEXT I
610 PRINT:PRINT:PRINTTAB(9);"SORT COMPLETE"
620 PRINT:PRINT:PRINT:PRINT" DO YOU WANT A PRINTOUT?":PRINT
621 PRINT"  [PRINT] = PRINTOUT TO PRINTER"
622 PRINT"  [YES]   = PRINTOUT TO VDU"
623 PRINT"  [NO]    = RETURN TO MENU"
625 K$=INKEY$
626 I$=INKEY$:IFI$=""THEN625
627 IFI$<>"Y"ANDI$<>"P"ANDI$<>"N"THEN625
630 IF I$="N" THEN 190
635 IF I$="P" THEN 950
640 IF I$="Y" THEN 700
700 PRINT
702 CLS:PRINT:PRINT TAB(7);"CALL SIGNS LOGGED":PRINT
710 FOR I=1 TO N
750 PRINT C1$(I),
760 NEXT I
764 PRINT:PRINT"  PRESS >>SPACE<< TO CONTINUE"
765 K$=INKEY$
770 I$=INKEY$:IFI$<>" "THEN765
780 GOTO 900
900 REM
910 CLS:PRINT:PRINT:PRINT:PRINT"   DO YOU WANT TO STOP NOW?"
912 PRINT:PRINT"          Y=YES      N=NO "
913 PRINT:PRINT:PRINT:PRINT:PRINT TAB(5)" ";N;" CALLS LOGGED "
915 K$=INKEY$
917 X$=INKEY$:IF X$=""THEN 917
920 IF X$="Y" THEN CLS:END
925 IF X$="N" THEN 200
927 IFX$<>"Y"ANDX$<>"N"THEN917
950 LPRINT TAB(15);"CALL SIGNS LOGGED"
955 LPRINT
960 FOR I=1 TO N+1
970 LPRINT C1$(I),
980 NEXT I
990 GOTO 900
```

CONTEST LOG VZED
by Ron Carson

This program should be of advantage to any radio amateur or shortwave listener who owns a VZ200.

As the title suggests the program is ideal for RD contests or any other type of log from which you wish to get a hard copy of call signs worked. To operate, it requires a printer to be connected to the computer.

The menu gives you 5 options:
LIST—List of all entries
SORT—Sort into alphanumeric order
PRINT—Printout to printer
END—End Program
—Enter Callsign

If you go into the sort mode all entries are placed in alphanumeric order, then you will be asked if you require a printout to printer
Printout to VDU
return to menu (cont)

After each entry you will be told if the last callsign entered is a new one or entered before. If already entered it will not be retained in data.

Do not enter END until you have your hard copy, as END or break will destroy all of your entries.

Micro-80
4(8) 1984
P 9 + 16.

```
100 REM  ***** DOG RACE *****
101 REM  AS PRINTED IN MICRO-80          FOR TRS80 - SYS80
102 REM  MODIFIED BY R. CARSON           FOR VZ 200
103 REM
104 REM
105 REM
109 CLS:PRINT:PRINT
110 PRINT"       **** DOG RACE ****"
115 PRINT:PRINT:PRINT:PRINT"   PRESS ANY KEY TO CONTINUE"
117 PRINT:PRINT:PRINT:PRINT"  PRESS <SPACE> TO START RACE"
120 I$=INKEY$

125 A3$=INKEY$:IFA3$=""THEN120
130 CLS:MODE(1)
131 COLOR4:FORX=0TO127:SET(X,0):NEXT:FORX=0TO127:SET(X,1):NEXT
134 FORX=0TO127:SET(X,2):NEXT
135 FORX=0TO127:SET(X,42):NEXT:FORX=0TO127:SET(X,43):NEXT
136 FORX=0TO127:SET(X,44):NEXT:COLOR3
137 FORX=0TO123:SET(X,12):NEXT
138 FORX=0TO123:SET(X,22):NEXT
139 FORX=0TO123:SET(X,32):NEXT
140 A=22:B=5:C=22:D=15:G=22:H=25:I=22:J=35
145 COLOR2
150 REM DRAW STAT DOG
160 X=A:Y=B:GOSUB370
170 X=C:Y=D:GOSUB370
180 X=G:Y=H:GOSUB370
190 X=I:Y=J:GOSUB370
210 COLOR2:FORY=4TO40STEP5:SET(124,Y):NEXTY
220 I$=INKEY$
225 K$=INKEY$:IFK$<>" "THEN225
230 Z=RND(4)
235 P=RND(5)
240 IFZ=1THENX=A:Y=B:GOSUB410:A=X:GOTO280
250 IFZ=2THENX=C:Y=D:GOSUB410:C=X:GOTO280
260 IFZ=3THENX=G:Y=H:GOSUB410:G=X:GOTO280
270 IFZ=4THENX=I:Y=J:GOSUB410:I=X:GOTO280
280 IFX<130THENGOTO230
285 FORW=1TO1000:NEXTW
290 IFA>=130THENPRINT"NO. 1 IS THE WINNER PAY";O$;P*15;"CENTS"
300 IFC>=130THENPRINT"NO. 2 IS THE WINNER PAY";O$;P*15;"CENTS"
310 IFG>=130THENPRINT"NO. 3 IS THE WINNER PAY";O$;P*15;"CENTS"
320 IFI>=130THENPRINT"NO. 4 IS THE WINNER PAY";O$;P*15;"CENTS"
330 FORF=1TO1000:NEXTF
340 INPUT"WOULD YOU LIKE ANOTHER RACE (Y/N)";A2$
350 IFA2$="Y"THEN100
360 IFA2$="N"THENCLS:END
370 SET(X-9,Y):SET(X-20,Y):SET(X-6,Y+1):SET(X-7,Y+1)
380 SET(X-8,Y+1):SET(X-19,Y+1):SET(X-10,Y+4):SET(X-17,Y+4)
390 SET(X-11,Y+5):SET(X-16,Y+5)
400 FORU=9TO18:FORV=2TO3:SET(X-U,Y+V):NEXTV:NEXTU:RETURN
410 RESET(X-20,Y):RESET(X-19,Y+1):SET(X-17,Y+1):SET(X-16,Y)
420 SET(X-5,Y+1):SET(X-4,Y+1):RESET(X-9,Y):SET(X-6,Y)
430 RESET(X-18,Y+2):RESET(X-17,Y+2):SET(X-8,Y+2):SET(X-7,Y+2)
440 RESET(X-8,Y+1):RESET(X-7,Y+1):RESET(X-11,Y+5):RESET(X-10,Y+4)
450 SET(X-8,Y+4):SET(X-7,Y+5):RESET(X-18,Y+3):RESET(X-17,Y+3)
460 SET(X-8,Y+3):SET(X-7,Y+3):RESET(X-17,Y+4):SET(X-15,Y+4)
470 RESET(X-17,Y+1):SET(X-15,Y+1):RESET(X-16,Y+2):RESET(X-16,Y+3)
480 RESET(X-15,Y+2):RESET(X-15,Y+3):RESET(X-16,Y+5)
490 RESET(X-15,Y+5):RESET(X-15,Y+4)
500 SET(X-13,Y+4):SET(X-12,Y+5):RESET(X-8,Y+4):SET(X-6,Y+4)
510 SET(X-6,Y+2):SET(X-5,Y+2):SET(X-6,Y+3):SET(X-5,Y+3)
520 SET(X-3,Y+1):SET(X-2,Y+1):RESET(X-6,Y):SET(X-5,Y)
530 RESET(X-6,Y+1):RESET(X-5,Y+1):X=X+4:RETURN
```

DOG RACE VZED

by Ron Carson

This program was published in Micro-80 some time ago for the TRS-80 and System-80. Now it has been modified to run in your VZ200.

I have only written the bare bones program. Although it runs well and is useable as is, it gives you the chance to expand the program to suit your needs.

After loading the program you are asked to do two things:

1. Press any key to continue.

2. Press SPACE TO START RACE

After the race is over the winning dog is printed in the text mode, and you are asked if you want to race again or end.

You will see there are plenty of options for you to look into to make this a really great game and a lot of fun.

Micro-80
4(8) 1984
P 9, 16 & 17.

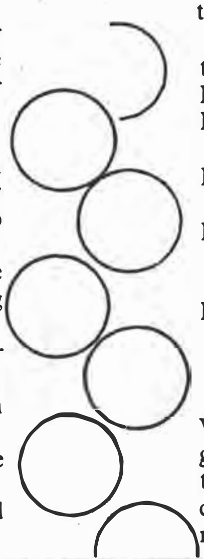
High Resolution Graphics Plotting

The following two programs demonstrate the high resolution graphics capabilities of the VZ-200. Both programs will run on the unexpanded (8k) computer.

Circle Plotting

Here is a quick but fairly accurate program to get your VZ-200 to draw circles. The following notes explain the program and will help in conversion to other machines.

Line 10 Sets high resolution graphics mode.
Line 20 These variables set the circles' centre to a position in the centre of the screen. By altering these variables it is possible to place the circles anywhere on the screen.
Line 30 These variables determine the shape of the circles. Eclipses can be formed by altering the values of these variables.
Line 40 R is the radius of the circle, N is the number of points to be plotted in the circle.
Line 2000 A is set at 2 x Pi which is a circle in radians.
Line 2030/2040 Contain the formulae which determine the value of the X and Y co-ordinate.
Line 2050 SET (X,Y) is the equivalent to HPLOT and PLOT X,Y in other versions of Basic.



Three Dimensional Plotting

This is a simple program for evolving three dimensional representations of trigonometrical functions on the VZ-200.

The following notes explain the main points in the program.

Line 100 Sets the high resolution graphics mode.
Line 110/115 V and H set the vertical and horizontal screen dimensions of the plot.
Line 170 Assumes that the point with co-ordinates 0,0 is at the top left hand corner of the screen.
Line 175 Sets the points on the screen. SET is equivalent to PLOT and HPLOT on other systems.
Line 155 Is the nucleus of the plot; this trigonometrical formula is the function to be plotted.

Variations are found in lines 255, 355, 455, etc., the program plotting a series of seven designs, pausing between plots. Pressing any key at the end of each plot clears the screen and then commences drawing the next design.

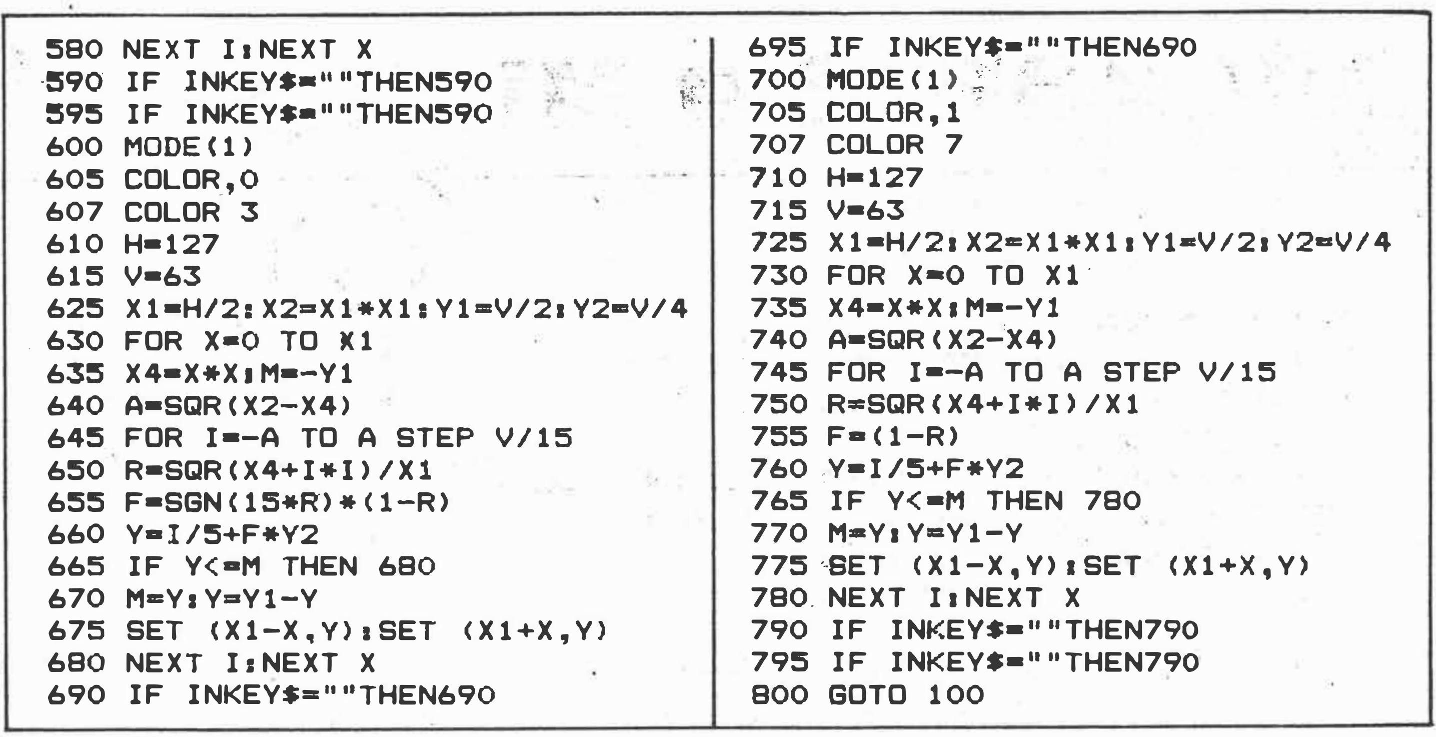
```
 1 CLS:'************************
 2 '** VZ-200 CIRCLE PLOTTER **
 3 '*        IAN A.THOMPSON     *
 4 '***************************
 6 PRINT@71," CIRCLE PLOTTER "
 7 PRINT@257,"IAN THOMPSON,
  COLLOROY PLATEAU"
8 IF INKEY$=""THEN8
9 IF INKEY$=""THEN8
10 MODE(1):COLOR,0:COLOR 3
15 REM****R=30
20 CX=60:CY=30
30 OX=1.5:OY=1
40 R=30:N=150
50 GOSUB 2000
100 REM****R=25
110 COLOR 2
120 CX=60:CY=30
130 OX=1.5:OY=1
140 R=25:N=130
150 GOSUB 2000
200 REM****R=20
210 COLOR 4
220 CX=60:CY=30
230 OX=1.5:OY=1
240 R=20:N=110
250 GOSUB 2000
300 REM****R=15
310 COLOR,1:COLOR 7
320 CX=60:CY=30
330 OX=1.5:OY=1
340 R=15:N=90
350 GOSUB 2000
400 REM****R=10
410 COLOR,1:COLOR 6
420 CX=60:CY=30
430 OX=1.5:OY=1
440 R=10:N=70
450 GOSUB 2000
500 REM****R=5
510 COLOR,1:COLOR 8
520 CX=60:CY=30
530 OX=1.5:OY=1
540 R=5:N=50
550 GOSUB 2000
1000 FOR A=1 TO 800:NEXT A
1010 COLOR,0
1020 FOR A=1 TO 800:NEXT A
1030 COLOR,1
1040 GOTO 1000
2000 A=2*(22/7)
2010 C=A/N
2020 FOR I=0 TO A STEP C
2030 X=R*SIN(I):X=INT(X*OX+CX+0.499)
2040 Y=R*COS(I):Y=INT(Y*OY+CY+0.499)
2050 SET(X,Y)
2060 NEXT I
2070 RETURN
```

P.C. Games Oct. 84
p55-57.
1 of 3.

```
5 CLS:SOUND25,6
10 PRINT@41," 3-DIMENSION "
   :'FOR THE UNEXPANDED VZ-200
20 PRINT@102,"[BY IAN  THOMPSON]"
30 PRINT:PRINT@162,"THIS IS
   A SIMPLE PROGRAM FOR"
40 PRINT@194,"EVOLVING
   THREE-DIMENSIONAL"
50 PRINT@226,"REPRESENTATIONS
   OF  TRIG-"
60 PRINT@258,"ONOMETRICAL
   FUNCTIONS."
70 PRINT@448,"PRESS ANY KEY TO
   START PLOTTING"
90 IF INKEY$="" THEN 90
91 IF INKEY$="" THEN 90
100 MODE(1)
105 COLOR,0
107 COLOR 2
110 H=117
115 V=63
125 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
130 FOR X=0 TO X1
135 X4=X*X:M=-Y1
140 A=SQR(X2-X4)
145 FOR I=-A TO A STEP V/10
150 R=SQR(X4+I*I)/X1
155 F=(R-1)*SIN(R*12)
160 Y=I/5+F*Y2
165 IF Y<=M THEN 180
170 M=Y:Y=Y1-Y
175 SET (X1-X,Y):SET (X1+X,Y)
180 NEXT I:NEXT X
190 IF INKEY$=""THEN190
195 IF INKEY$=""THEN190
200 MODE(1)
205 COLOR,0
207 COLOR 3
210 H=117
215 V=63
225 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
230 FOR X=0 TO X1
235 X4=X*X:M=-Y1
240 A=SQR(X2-X4)
245 FOR I=-A TO A STEP V/10
250 R=SQR(X4+I*I)/X1
255 F=COS(9*R)*(1-R)*2
260 Y=I/5+F*Y2
265 IF Y<=M THEN 280
270 M=Y:Y=Y1-Y
275 SET (X1-X,Y):SET (X1+X,Y)
280 NEXT I:NEXT X
290 IF INKEY$=""THEN290
295 IF INKEY$=""THEN290
```

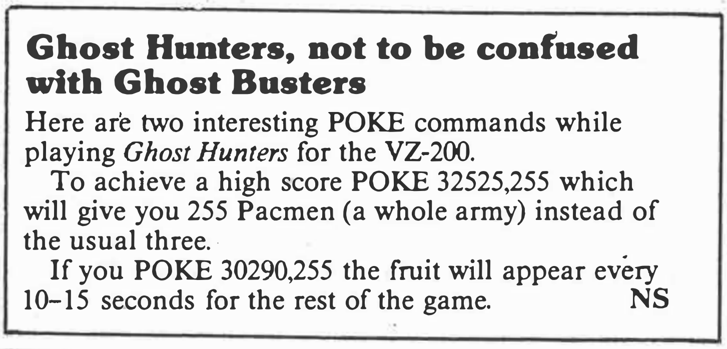
```
300 MODE(1)
305 COLOR,1
307 COLOR 7
310 H=127
315 V=63
325 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
330 FOR X=0 TO X1
335 X4=X*X:M=-Y1
340 A=SQR(X2-X4)
345 FOR I=-A TO A STEP V/20
350 R=SQR(X4+I*I)/X1
355 F=COS(20*R)*(1-R)
360 Y=I/5+F*Y2
365 IF Y<=M THEN 380
370 M=Y:Y=Y1-Y
375 SET (X1-X,Y):SET (X1+X,Y)
380 NEXT I:NEXT X
390 IF INKEY$=""THEN390
395 IF INKEY$=""THEN390
400 MODE(1)
405 COLOR,0
407 COLOR 4
410 H=127
415 V=63
425 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
430 FOR X=0 TO X1
435 X4=X*X:M=-Y1
440 A=SQR(X2-X4)
445 FOR I=-A TO A STEP V/20
450 R=SQR(X4+I*I)/X1
455 F=ATN(20*R)*(1-R)
460 Y=I/5+F*Y2
465 IF Y<=M THEN 480
470 M=Y:Y=Y1-Y
475 SET (X1-X,Y):SET (X1+X,Y)
480 NEXT I:NEXT X
490 IF INKEY$=""THEN490
495 IF INKEY$=""THEN490
500 MODE(1)
505 COLOR,1
507 COLOR 8
510 H=127
515 V=63
525 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
530 FOR X=0 TO X1
535 X4=X*X:M=-Y1
540 A=SQR(X2-X4)
545 FOR I=-A TO A STEP V/20
550 R=SQR(X4+I*I)/X1
555 F=LOG(25*R)*(1-R)
560 Y=I/5+F*Y2
565 IF Y<=M THEN 580
570 M=Y:Y=Y1-Y
575 SET (X1-X,Y):SET (X1+X,Y)
```

P.C. Games Oct 84
p 55-57
2 of 3.



```
580 NEXT I:NEXT X
590 IF INKEY$=""THEN590
595 IF INKEY$=""THEN590
600 MODE(1)
605 COLOR,0
607 COLOR 3
610 H=127
615 V=63
625 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
630 FOR X=0 TO X1
635 X4=X*X:M=-Y1
640 A=SQR(X2-X4)
645 FOR I=-A TO A STEP V/15
650 R=SQR(X4+I*I)/X1
655 F=SGN(15*R)*(1-R)
660 Y=I/5+F*Y2
665 IF Y<=M THEN 680
670 M=Y:Y=Y1-Y
675 SET (X1-X,Y):SET (X1+X,Y)
680 NEXT I:NEXT X
690 IF INKEY$=""THEN690
695 IF INKEY$=""THEN690
700 MODE(1)
705 COLOR,1
707 COLOR 7
710 H=127
715 V=63
725 X1=H/2:X2=X1*X1:Y1=V/2:Y2=V/4
730 FOR X=0 TO X1
735 X4=X*X:M=-Y1
740 A=SQR(X2-X4)
745 FOR I=-A TO A STEP V/15
750 R=SQR(X4+I*I)/X1
755 F=(1-R)
760 Y=I/5+F*Y2
765 IF Y<=M THEN 780
770 M=Y:Y=Y1-Y
775 SET (X1-X,Y):SET (X1+X,Y)
780 NEXT I:NEXT X
790 IF INKEY$=""THEN790
795 IF INKEY$=""THEN790
800 GOTO 100
```

P.C. Games Oct 84
p 55-57
3 of 3.



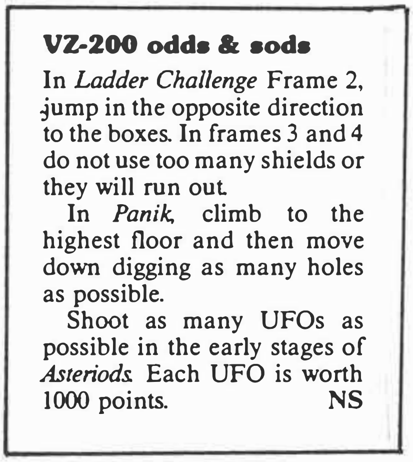
Ghost Hunters, not to be confused with Ghost Busters

Here are two interesting POKE commands while playing *Ghost Hunters* for the VZ-200.

To achieve a high score POKE 32525,255 which will give you 255 Pacmen (a whole army) instead of the usual three.

If you POKE 30290,255 the fruit will appear every 10–15 seconds for the rest of the game. **NS**

PCG Jan 85 2(1) p. 54.



VZ-200 odds & sods

In *Ladder Challenge* Frame 2, jump in the opposite direction to the boxes. In frames 3 and 4 do not use too many shields or they will run out.

In *Panik*, climb to the highest floor and then move down digging as many holes as possible.

Shoot as many UFOs as possible in the early stages of *Asteriods*. Each UFO is worth 1000 points. NS

PCG Nov 84 1(4) p 82.

VZ 200

GOLF SIMULATION

This draws a golf course in graphics mode with endless variations on bunkers, water hazards and roughs, and allows the player to actually 'play' the shots giving a choice of club, hitting strength and direction.

Gary McCleary
Emu Plains NSW

```
40 REM GOLF SIMULATION
50 REM BY GARY J MCCLEARY
51 REM DEC. 1983
100 CLS
110 PRINT@33,"WELCOME TO GLENLAY GOLF CO
URSE"
111 PRINT
112 PRINT"IN GOLF THE OBJECT OF THE GAME
"
113 PRINT"IS TO HIT THE BALL FROM THE"
114 PRINT"TEE(T) TO THE HOLE IN THE"
115 PRINT"FEWEST NUMBER OF SHOTS."
120 PRINT
125 PRINT"WILL THERE BE 1 OR 2 PLAYERS?"
130 K$=INKEY$
133 I$=INKEY$:WW=RND(DD):DD=DD+1:IFDD>10
0THENDD=1:IFI$=""THEN133
135 IFI$="1"THENPL=1:LP=0:GOTO145
137 IFI$="2"THENPL=2:LP=0:GOTO145
140 GOTO130
145 CLS
155 PRINT"YOUR GOLF BAG CONTAINS A:"
158 PRINT
160 PRINT"1 WOOD MAX.RANGE 251 METRES"
165 PRINT"2 IRON MAX.RANGE 221 METRES"
170 PRINT"5 IRON MAX.RANGE 164 METRES"
175 PRINT"7 IRON MAX.RANGE 127 METRES"
180 PRINT"9 WEDGE MAX RANGE 87 METRES"
185 PRINT"PUTTER MAX.RANGE 41 METRES"
190 PRINT"AND IS ONLY USED ON THE GREEN"
194 PRINT
195 PRINT"TO ACHIEVE GREATER HEIGHT"
200 PRINT"USE A HIGH NUMBERED IRON"
205 PRINT
210 PRINT"SPACE CONTINUES THE GAME"
250 GOSUB20980
300 HO=1:TT=0:T1=0:T2=0:GF=0
350 PA=RND(3)+2
351 PZ=RND(2)
352 IFPA=3THENP=3:SX=63:GOTO400
354 IFPA=4THENP=4.8
366 IFPA=5THENP=6.5
368 IFPZ=1THENSX=0
370 IFPZ=2THENSX=119
400 REM
420 ZB=RND(3):ZW=RND(3):ZJ=RND(3)
430 J3=RND(9)+2
450 A=RND(107)+7:BB=RND(7)+10
453 G=RND(5)+2:B=RND(9)+2:W=RND(10)+3
455 IFZJ=1THENJ3=0
456 IFZB=1THENB=0
457 IFZW=1THENW=0
458 C=RND(103)+9:D=13+RND(6)
459 MD=INT(SQR((A-SX)^2+(BB-63)^2)*P)
460 HB=SQR((A-C)^2+(BB-D)^2)
465 IFHB<=G+B+3THEN458
466 E=13+RND(100):F=14+RND(35)
468 BW=SQR((C-E)^2+(D-F)^2)
470 WH=SQR((A-E)^2+(BB-F)^2)
472 IFBW<=B+W+3THEN466
474 IFWH<=W+G+3THEN466
480 J1=RND(103)+9:J2=RND(6)+13
485 HJ=SQR((A-J1)^2+(BB-J2)^2)
490 IFHJ<=G+J3+3THEN458
492 JW=SQR((J1-E)^2+(J2-F)^2)
494 IFJW<=J3+W+3THEN466
500 CLS
506 X=SX:Y=63:R1=0:B1=0:W1=0
507 SC=0
509 CLS
510 PRINT"THIS IS HOLE NUMBER" HO
511 PRINT
512 PRINT"PLAYER" LP+1
513 PRINT
514 PRINT"PAR"PA;  MD "METRES"
515 SC=0:X=SX:Y=63:R1=0:B1=0:W1=0
517 GOSUB20980
522 GOSUB20000
523 GOSUB20980
524 CLS
525 PRINT"WHICH CLUB DO YOU WISH TO USE"
527 INPUTCL
530 IFCL=1THENAV=29+RND(11):GOTO600
540 IFCL=2THENAV=19+RND(11):GOTO600
550 IFCL=5THENAV=69+RND(6):GOTO600
560 IFCL=7THENAV=74+RND(6):GOTO600
570 IFCL=9THENAV=79+RND(6):GOTO600
580 CLS:PRINT"YOU DO NOT HAVE ONE OF THO
SE":GOTO525
600 CLS
602 PRINT"IN WHICH DIRECTION DO YOU WISH
"
610 PRINT"TO HIT? (0TO360 DEGREES)"
620 PRINT"MEASURED ANTICLOCKWISE FROM"
630 PRINT"THE RIGHT"
635 GOSUB60300
640 INPUTAZ
645 CLS
650 PRINT"HOW HARD DO YOU WISH TO HIT"
660 INPUT"0TO50";V
665 CLS
668 PS=3.141592654/180
670 IFV<0THENV=0
675 IFV>50THENV=50
677 SC=SC+1
680 RA=V*V*SIN(2*AV*PS)/9.81
682 RS=RA/P
685 HT=((SIN(AV*PS)*V)^2)/(19.62)
686 IFR1=1THEN12000
687 IFB1=1THEN13000
690 X=X+RS*COS(AZ*PS)
700 Y=Y-RS*SIN(AZ*PS)
710 H=INT(X):K=INT(Y)
715 H1=0
720 IFH<0THENH=0:H1=1
725 IFH>=127THENH=126:H1=1
730 IFK<0THENK=0:H1=1
735 IFK>63THENK=63:H1=0
736 X=H:Y=K
740 IFH1=1THEN9000
742 OI=SQR((A-H)^2+(BB-K)^2)
745 REM
746 IFOI<=GANDGF=1THEN790
747 GOSUB20000
754 COLOR2
755 K$=INKEY$
760 I$=INKEY$
765 SET(H,K):SET(H+1,K)
770 RESET(H,K):RESET(H+1,K)
775 IFI$=""THEN760
780 IFI$<>" "THEN760
790 DI=SQR((A-H)^2+(88-K)^2)
792 OB=SQR((C-H)^2+(D-K)^2)
794 OW=SQR((E-H)^2+(F-K)^2)
796 OJ=SQR((J1-H)^2+(J2-K)^2)
800 DM=OI*P
810 IFOI<=GTHENGF=1:GOTO8000
812 IFOB<=BANDB<>0THEN7000
813 IFOJ<=J3ANDJ3<>0THEN7000
814 IFOW<=WANDW<>0THEN10000
816 CLS
817 PRINT"THAT SHOT WENT "INT(RA)"METRES
"
819 PRINT
820 PRINT"DISTANCE FROM THE HOLE"
822 PRINTINT(DM)"METRES"
825 PRINT"NUMBER OF STROKES="SC
827 IFPA=4ORPA=5THEN1000
830 IFH<40ANDK>31THEN11000
835 IFH>86ANDK>31THEN11000
840 IFK<=8THEN11000
845 GOTO2000
1000 IFPZ=2THEN1500
1100 IFH>16ANDK>31THEN11000
1110 IFK<=8THEN11000
1120 GOTO2000
1500 IFH<111ANDK>31THEN11000
1510 IFK<=8THEN11000
1520 GOTO2000
2000 GOTO525
7000 B1=1
7005 BH=124.5
7010 PRINT"YOU ARE IN THE BUNKER"
7020 PRINT"YOU ARE ADVISED TO USE THE WE
DGE"
7030 GOTO525
8000 GF=1:GOTO60060
8004 CLS
8008 PRINT"YOU ARE ON THE GREEN AND WILL
"
8010 PRINT"BE USING THE PUTTER"
8020 PRINT"WHICH DIRECTION (0TO360)"
8025 GOSUB60300
8030 INPUTAZ
8035 CLS
```

from "Bumper Book of Programs" by Y.C. 1985 1 of 2.

also in "Computer Fun and Games" #2 Mar 86.

```
8040 PRINT"HOW HARD DO YOU WANT TO HIT"
8050 INPUT"(0TO25)";V
8060 IFV<0THENV=0
8065 IFV>25THENV=25
8070 AV=70
8075 CLS
8200 GOTO677
9000 SOUND4,2:SC=SC+1:GOTO745
10000 W1=0
10005 SC=SC+1
10010 H=H+2*W:K=K+2*W
10020 GOTO60000
11000 R1=1
11005 RH=111+RND(15)
11010 PRINT
11011 PRINT"YOU ARE IN THE ROUGH"
11012 IFRH>123THENB$="TALL TREES":GOTO11
018
11014 IFRH>118THENB$="MEDIUM TREES":GOTO
11018
11016 IFRH>=112THENB$="LOW SCRUB":GOTO11
018
11018 PRINT"YOUR NEXT SHOT MUST CLEAR SO
ME"
11019 PRINTB$
11020 PRINT
11030 GOTO525
12000 IFHT<RHTHENRA=RND(6):GOTO12100
12010 RA=RA/2
12100 R1=0:GOTO682
13000 IFHT<BHTHENRA=0:GOTO13100
13010 RA=RA/2
13100 B1=0:GOTO682
15000 SOUND20,1:SOUND15,1
15002 IFLP=0THENT1=T1+SC:TT=T1:P1=P1+SC-
PA:Q=P1
15003 IFLP=1THENT2=T2+SC:TT=T2:P2=P2+SC-
PA:Q=P2
15005 A$=" FOR THIS HOLE"
15008 CLS
15010 PRINT@39,"CONGRATULATIONS"
15015 PRINT@73,"PLAYER"LP+1
15020 PRINT
15030 PRINT"YOU ARE IN THE HOLE"
15040 PRINT"FOR "SC" SHOTS"
15060 IFSC=PA-2THENPRINT"EAGLE";A$
15062 IFSC=PA-1THENPRINT"BIRDIE";A$
15064 IFSC=PATHENPRINT"PAR";A$
15066 IFSC=PA+1THENPRINT"BOGEY";A$
15068 IFSC=PA+2THENPRINT"DOUBLE BOGEY";A
$
15069 IFSC=1THENPRINT"HOLE IN ONE!!!":GO
TO15072
15070 PRINT
15072 PRINT"YOUR TOTAL SO FAR IS"TT
15074 IFQ=0THENPRINT"YOU ARE ON PAR FOR
THE COURSE"
15076 IFQ>0THENPRINT"YOU ARE "Q" OVER PA
R FOR THE     COURSE
15078 IFQ<0THENQ=ABS(Q):PRINT"YOUR TOTAL
 IS"Q"UNDER PAR"
15080 PRINT:PRINT
16008 PRINT"     PRESS THE SPACE"
16010 K$=INKEY$
16020 I$=INKEY$:KD=RND(DD)
16030 DD=DD+1:IFDD>100THENDD=1
16040 IFI$=""THEN16020
16050 IFI$<>" "THEN16020
16060 CLS
16100 IFPL=1THENHO=HO+1:GOTO350
16200 IFPL=2ANDLP=1THENLP=0:HO=HO+1:GOTO
350
16210 IFPL=2ANDLP=0THEN:LP=1:GOTO510
20000 COLOR4
20001 MODE(1):GF=0
20002 IFPA=4ORPA=5THEN20112
20005 FORI=0TO127STEP2
20010 SET(I,8):SET(RND(126),RND(7))
20020 NEXT
20030 FORI=0TO40STEP2
20040 SET(I,31):SET(RND(40),31+RND(31))
20050 NEXT
```

```
20060 FORI=86TO127STEP2
20070 SET(I,31):SET(RND(40)+86,31+RND(31
))
20080 NEXT
20090 FORI=31TO63STEP2
20100 SET(40,I):SET(86,I)
20110 NEXT
20111 GOTO20200
20112 IFPZ=2THEN20140
20115 FORI=0TO127STEP2
20119 SET(I,B):SET(RND(126),RND(7))
20120 NEXT
20122 FORI=16TO127STEP2
20124 SET(I,31):SET(RND(110)+16,31+RND(3
1))
20126 NEXT
20128 FORI=31TO63STEP2
20130 SET(16,I)
20132 NEXT
20134 GOTO20200
20140 FORI=0TO127STEP2
20142 SET(I,8):SET(RND(126),RND(7))
20144 NEXT
20150 FORI=0TO111STEP2
20152 SET(I,31):SET(RND(110),RND(31)+31)
20154 NEXT
20156 FORI=31TO63STEP2
20158 SET(111,I)
20160 NEXT
20162 GOTO20200
20200 FORI=A-GTOA+G
20210 FORJ=BB-GTOBB+G
20220 SET(I,J)
20225 NEXT:NEXT
20226 COLOR2
20228 FORI=BB-11TOBB:RESET(A,I):NEXT
20232 FORI=BB-11TOBB:SET(A,I):NEXT
20233 FORJ=BB-11TOBB-8
20234 FORI=ATOA+4
```

```
20235 SET(I,J):NEXT:NEXT
20236 IFZB=1THEN20265
20238 COLOR2
20240 FORI=C-BTOC+BSTEP2
20250 FORJ=D-BTOD+BSTEP2
20260 SET(I,J)
20264 NEXT:NEXT
20265 IFZJ=1THEN20273
20266 COLOR2
20267 FORI=J1-J3TOJ1+J3STEP2
20268 FORJ=J2-J3TOJ2+J3STEP2
20269 SET(I,J)
20270 NEXT:NEXT
20273 IFZW=1THEN20349
20275 COLOR3
20280 FORI=E-WTOE+WSTEP2
20290 FORJ=F-WTOF+WSTEP2
20300 SET(I,J)
20310 NEXT:NEXT
20349 COLOR4
20350 FORI=SX-2TOSX+2
20360 SET(I,60)
20365 NEXT
20370 FORI=60TO63
20380 SET(SX,I)
20385 NEXT
20970 RETURN
20980 K$=INKEY$
20982 I$=INKEY$:IFI$=""THEN20982
20984 IFI$<>" "THEN20982
20990 RETURN
60000 CLS
60010 PRINT"YOU WERE IN THE WATER AND HA
VE"
60020 PRINT"BEEN REPOSITIONED FURTHER BA
CK"
60030 PRINT"WITH A PENALTY OF 1"
60040 FORI=1TO3000:NEXT
60050 GOTO715
60060 MODE(1)
```

```
60070 GS=INT(47/(2*G))
60080 HH=2*(H-A)*GS+63
60090 KK=(K-BB)*GS+31
60093 COLOR4
60095 FORI=12TO106STEP2
60100 SET(I,8):SET(I,55)
60110 NEXT
60120 FORI=8TO55STEP2
60130 SET(12,I):SET(106,I)
60140 NEXT
60145 COLOR2
60150 FORI=12TO31
60160 SET(63,I)
60165 NEXT
60170 FORI=63TO75
60180 FORJ=12TO18
60190 SET(I,J)
60200 NEXT:NEXT
60210 FORI=63-GSTO63+GS
60220 FORJ=31-GS/2TO31+GS/2
60230 SET(I,J)
60240 NEXT:NEXT
60243 COLOR4
60245 K$=INKEY$
60246 I$=INKEY$
60250 SET(HH,KK):SET(HH+1,KK)
60270 IFI$=""THEN60246
60280 IFI$<>" "THEN60246
60285 IFDI<=.5THEN15000
60290 GOTO8004
60300 PRINT@176,"90"
60310 PRINT@208,"."
60312 PRINT@240,"."
60314 PRINT@272,"."
60320 PRINT@297,"180...BALL...0"
60330 PRINT@336,"."
60332 PRINT@368,"."
60334 PRINT@400,"."
60340 PRINT@432,"270"
60360 RETURN
```

Golf Simulation

from "Bumper Book of Programs."

by Y.C. 1985 2 of 2.

```
100 REM "KNIGHT'S CROSS"
110 REM "LUKE LUCAS. PAPUA NEW GUINEA"
120 REM "OCTOBER 83"
130 REM "RECODED BY R.B.K. 10-DEC-85"
140 CLS
150 MODE(1)
160 FOR R% = 1 TO 24
170     C% = RND(3) + 1                : COLOR C%
180     FOR A% = -R% TO R%
190         ZY = SQR(R%*R% + A%*A%) : Y% = INT(ZY - 0.5)
200         SET( 60 + A%, 30 + Y%)
210         SET( 60 - A%, 34 - Y%)
220         SET( 65 + Y%, 32 - A%)
230         SET( 55 - Y%, 32 + A%)
240     NEXT A%
250 NEXT R%
260 FOR R% = 1 TO 12
270     FOR A% =  -R% TO R%
280         ZY = SQR(R%*R% - A%*A%) : Y% = INT(ZY - 0.5)
290         C% = RND(3) + 1            : COLOR C%
300         SET( 60 + A%, 30 + Y%)
310         SET( 60 + A%, 30 - Y%)
320         SET( 12 + A%, 30 + Y%)
330         SET( 12 + A%, 30 - Y%)
340         SET(114 + A%, 30 + Y%)
350         SET(114 + A%, 30 - Y%)
360         SET( 60 + A%, 13 - Y%)
370         SET( 60 + A%, 50 + Y%)
380     NEXT A%
390 NEXT R%
400 C% = RND(3) + 1     : COLOR C%
410 FOR X% =  0 TO 127
420     SET( X%,  0)
430     SET( X%,  1)
440     SET( X%, 62)
450     SET( X%, 63)
460 NEXT X%
470 FOR Y% =  0 TO 63
480     SET(  0, Y%)
490     SET(  1, Y%)
500     SET(126, Y%)
510     SET(127, Y%)
520 NEXT Y%
530 FOR T = 1 TO 2000   : NEXT T
540 GO TO 150
```

VZ200

```
1 REM"KNIGHT'S CROSS"
2 REM"LUKE LUCAS,PAPUA NEW GUINEA"
3 REM"OCTOBER,1983"
10 CLS
14 MODE(1)
15 FORR=1TO24
16 C=RND(3)+1:COLORC
20 FORA=-RTOR
30 Y=SQR(A*A+R*R):Y= INT(Y-.5)
40 SET(A+60,30+Y)
41 SET(60-A,34-Y)
42 SET(Y+65,32-A)
43 SET(55-Y,32+A)
60 NEXT:NEXT
70 FORR=1TO12
80 FORA=-RTOR
90 Y=SQR(R*R-A*A):Y=INT(Y-.5)
95 C=RND(3)+1:COLORC
96 SET(A+60,30+Y):SET(A+60,30-Y)
100 SET(A+12,30+Y)
110 SET(A+12,30-Y)
120 SET(A+114,30+Y)
130 SET(A+114,30-Y)
140 SET(A+60,13-Y)
170 SET(A+60,50+Y)
200 NEXT:NEXT
300 C=RND(3)+1:COLORC
345 FORX=0TO127:FORY=0TO1:
SET(X,Y):NEXT:NEXT
350 FORX=0TO127:FORY=62TO63:
SET(X,Y):NEXT:NEXT
360 FORX=0TO1:FORY=0TO63:
SET(X,Y):NEXT:NEXT
370 FORX=126TO127:FORY=0TO63:
SET(X,Y):NEXT:NEXT
400 FORT=1TO2000:NEXT
410 COPY
1000 GOTO14
```

KNIGHTS CROSS

The program is purely graphics and works as follows:

(170) Line 16 sets random colour.

(180-240) Lines 30-60 creates what I call an inverted German Cross in multi colours.

(270-380) Lines 90-200 draw a circle in the cross.

(410-520) Lines 345-370 draw a square.

(530) Line 370 pauses to display the image.

The end result looks like the 'Knights Cross with oak leaves' just like the Germans issued their war heroes.

It shows how we can use the capabilities of the VZ200 todraw very intricate designs by allowing the composition and placement of the A Z Y in the lines (200-230 and 300-370) 40-43 and 100-170, i.e. A+60 change to A-60 or A+60, 30+Y change to 30+Y,A-60 all sorts of wonderful patterns can be created.

G. Lucas
Boroko PNG

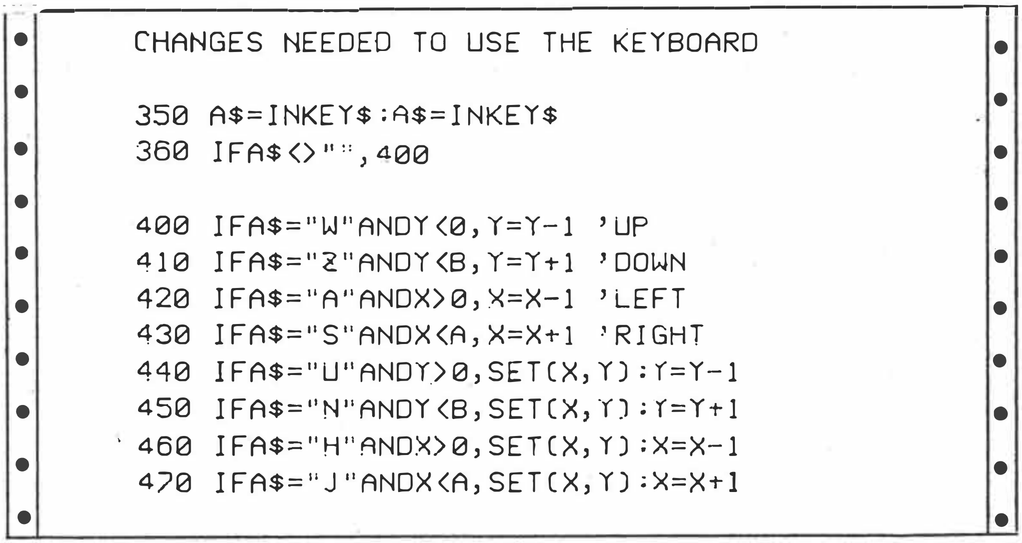
from "Bumper Book of Programs"
by Y.C. 1985.

Sketcher

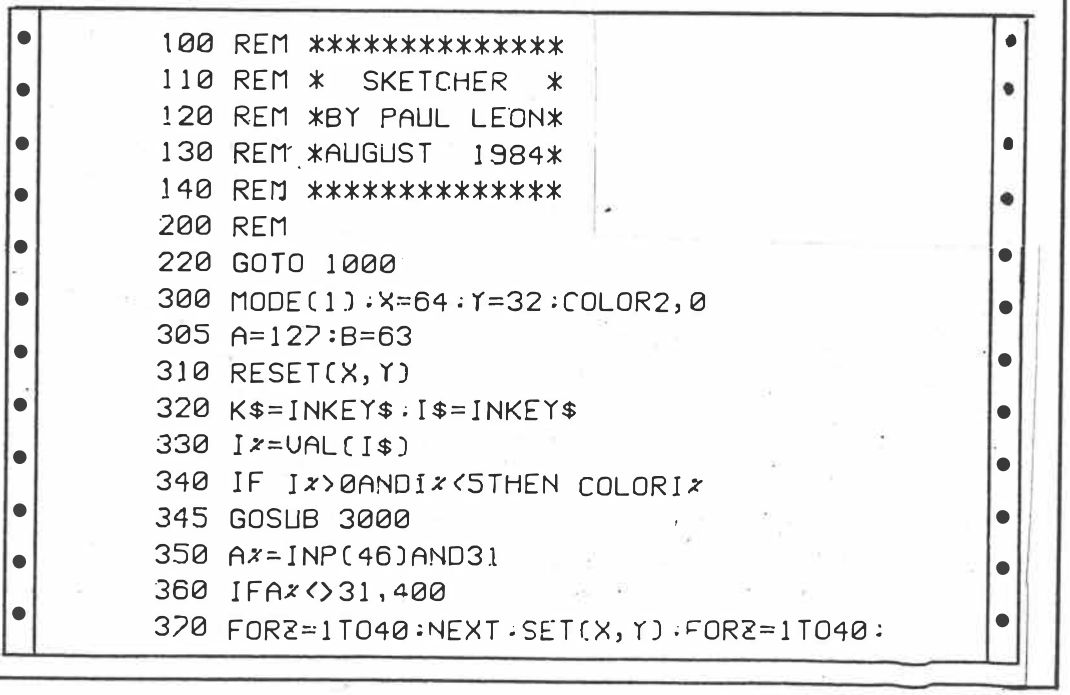
by P Leon

"Sketcher" was written for the unexpanded VZ-200. It allows you to draw in 4 colours, rubout, clear the screen, and get a hard copy of your artwork (if you have a suitable printer attached). There are instructions in the program. The program was written to use joysticks but if you do not have any or would like to use the keyboard, the changes you will need are at the end of the program listing. These are the keys you would use if you use the keyboard.

```
RUBOUT      DRAW

  W           U

A   S       H   J

  Z           N
```

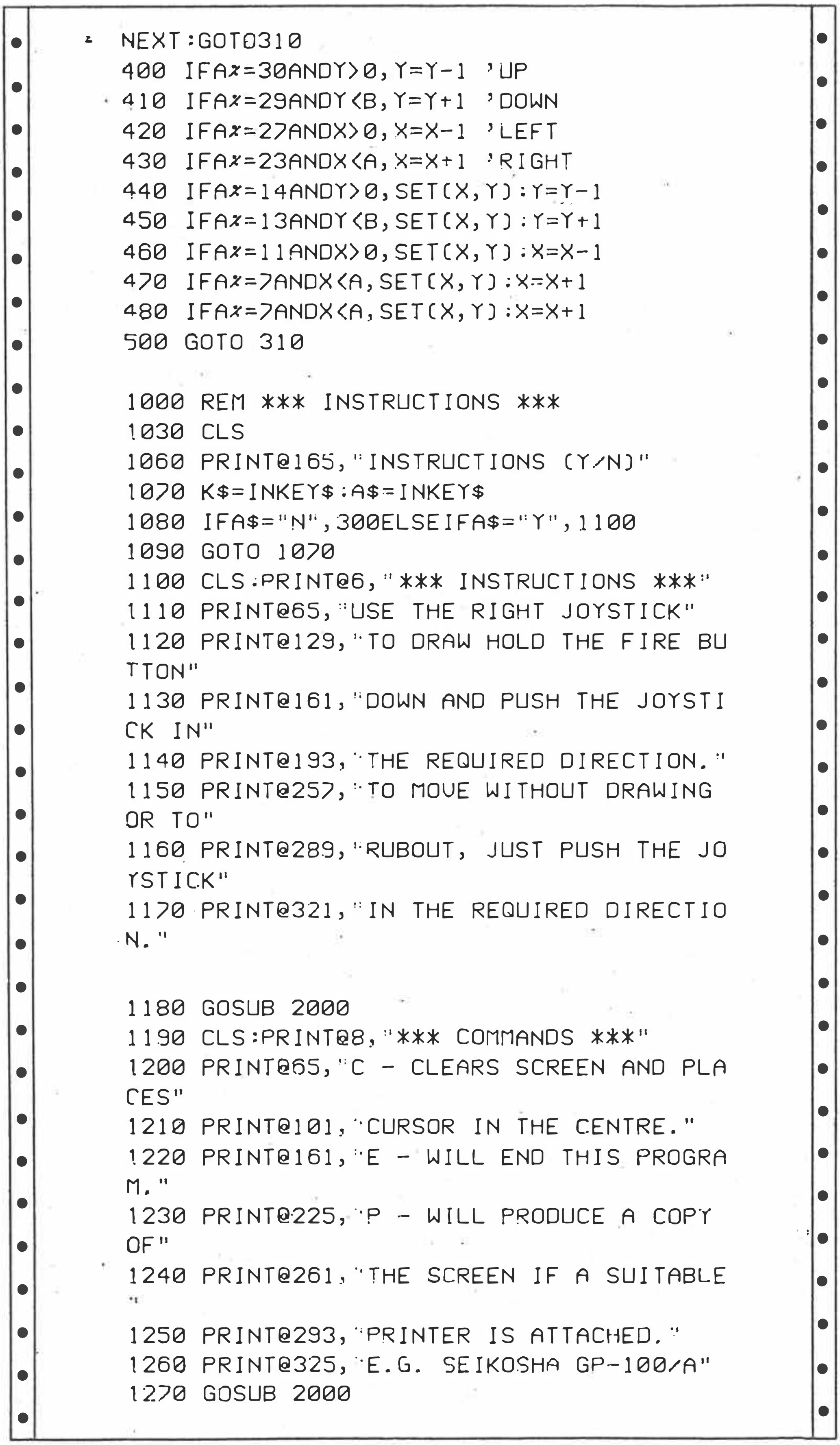


```
CHANGES NEEDED TO USE THE KEYBOARD

350 A$=INKEY$:A$=INKEY$
360 IFA$<>"",400

400 IFA$="W"ANDY<0,Y=Y-1 'UP
410 IFA$="Z"ANDY<B,Y=Y+1 'DOWN
420 IFA$="A"ANDX>0,X=X-1 'LEFT
430 IFA$="S"ANDX<A,X=X+1 'RIGHT
440 IFA$="U"ANDY>0,SET(X,Y):Y=Y-1
450 IFA$="N"ANDY<B,SET(X,Y):Y=Y+1
460 IFA$="H"ANDX>0,SET(X,Y):X=X-1
470 IFA$="J"ANDX<A,SET(X,Y):X=X+1
```



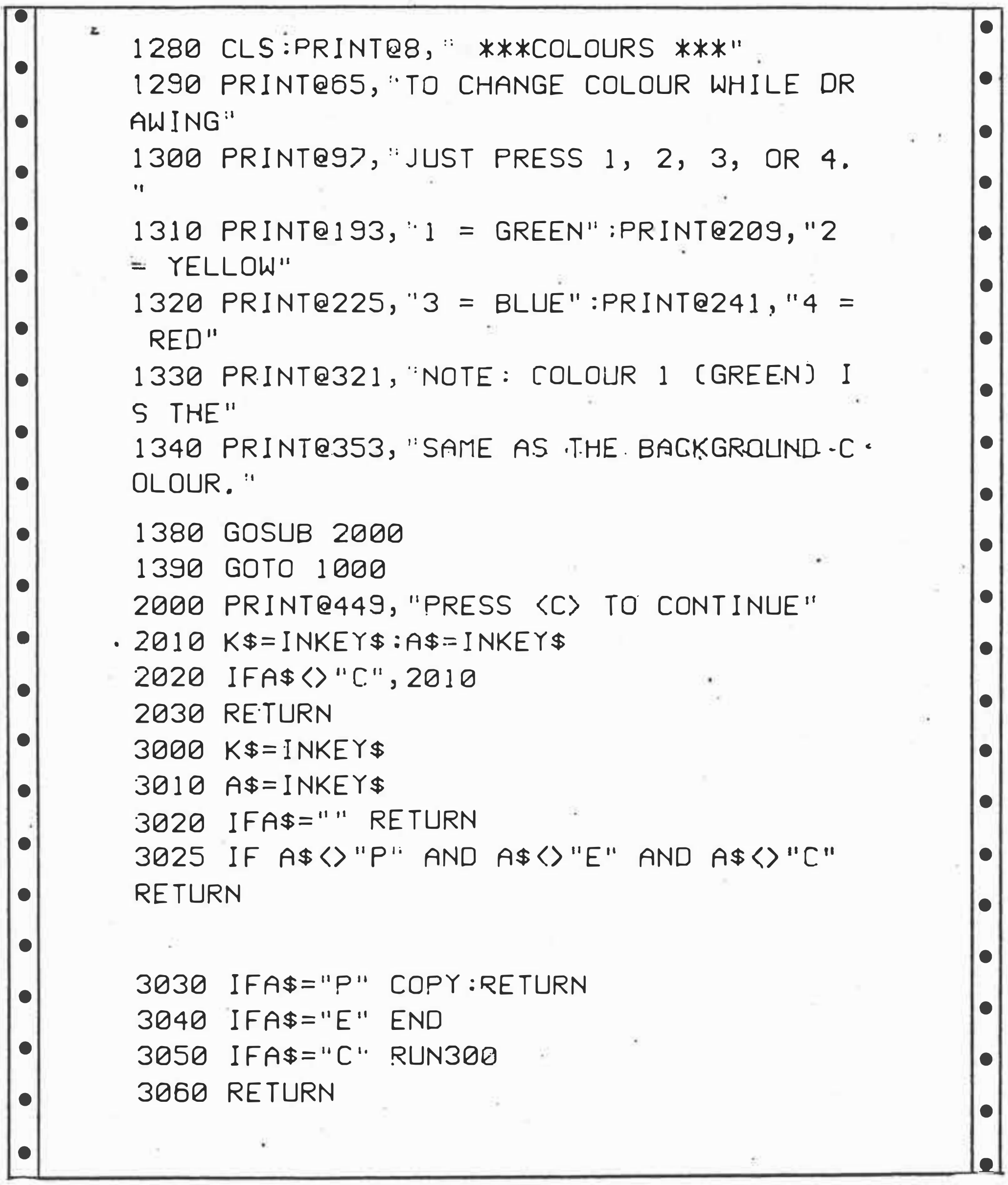
```
100 REM **************
110 REM *  SKETCHER  *
120 REM *BY PAUL LEON*
130 REM *AUGUST  1984*
140 REM **************
200 REM
220 GOTO 1000
300 MODE(1):X=64:Y=32:COLOR2,0
305 A=127:B=63
310 RESET(X,Y)
320 K$=INKEY$:I$=INKEY$
330 I%=VAL(I$)
340 IF I%>0ANDI%<5THEN COLORI%
345 GOSUB 3000
350 A%=INP(46)AND31
360 IFA%<>31,400
370 FORZ=1TO40:NEXT:SET(X,Y):FORZ=1TO40:
```

A.P.C. Jan 85. V6(1)
p 129 – 131
1 of 3.



```
NEXT:GOTO310
400 IFA%=30ANDY>0,Y=Y-1 'UP
410 IFA%=29ANDY<B,Y=Y+1 'DOWN
420 IFA%=27ANDX>0,X=X-1 'LEFT
430 IFA%=23ANDX<A,X=X+1 'RIGHT
440 IFA%=14ANDY>0,SET(X,Y):Y=Y-1
450 IFA%=13ANDY<B,SET(X,Y):Y=Y+1
460 IFA%=11ANDX>0,SET(X,Y):X=X-1
470 IFA%=7ANDX<A,SET(X,Y):X=X+1
480 IFA%=7ANDX<A,SET(X,Y):X=X+1
500 GOTO 310

1000 REM *** INSTRUCTIONS ***
1030 CLS
1060 PRINT@165,"INSTRUCTIONS (Y/N)"
1070 K$=INKEY$:A$=INKEY$
1080 IFA$="N",300ELSEIFA$="Y",1100
1090 GOTO 1070
1100 CLS:PRINT@6,"*** INSTRUCTIONS ***"
1110 PRINT@65,"USE THE RIGHT JOYSTICK"
1120 PRINT@129,"TO DRAW HOLD THE FIRE BU
TTON"
1130 PRINT@161,"DOWN AND PUSH THE JOYSTI
CK IN"
1140 PRINT@193,"THE REQUIRED DIRECTION."
1150 PRINT@257,"TO MOVE WITHOUT DRAWING
OR TO"
1160 PRINT@289,"RUBOUT, JUST PUSH THE JO
YSTICK"
1170 PRINT@321,"IN THE REQUIRED DIRECTIO
N."


1180 GOSUB 2000
1190 CLS:PRINT@8,"*** COMMANDS ***"
1200 PRINT@65,"C - CLEARS SCREEN AND PLA
CES"
1210 PRINT@101,"CURSOR IN THE CENTRE."
1220 PRINT@161,"E - WILL END THIS PROGRA
M."
1230 PRINT@225,"P - WILL PRODUCE A COPY
OF"
1240 PRINT@261,"THE SCREEN IF A SUITABLE
"

1250 PRINT@293,"PRINTER IS ATTACHED."
1260 PRINT@325,"E.G. SEIKOSHA GP-100/A"
1270 GOSUB 2000
```

A.P.C. Jan 85 V6(1)
p 129-131
2 of 3.



```
1280 CLS:PRINT@8," ***COLOURS ***"
1290 PRINT@65,"TO CHANGE COLOUR WHILE DR
AWING"
1300 PRINT@97,"JUST PRESS 1, 2, 3, OR 4.
"
1310 PRINT@193,"1 = GREEN":PRINT@209,"2
= YELLOW"
1320 PRINT@225,"3 = BLUE":PRINT@241,"4 =
 RED"
1330 PRINT@321,"NOTE: COLOUR 1 (GREEN) I
S THE"
1340 PRINT@353,"SAME AS THE BACKGROUND C
OLOUR."
1380 GOSUB 2000
1390 GOTO 1000
2000 PRINT@449,"PRESS <C> TO CONTINUE"
2010 K$=INKEY$:A$=INKEY$
2020 IFA$<>"C",2010
2030 RETURN
3000 K$=INKEY$
3010 A$=INKEY$
3020 IFA$="" RETURN
3025 IF A$<>"P" AND A$<>"E" AND A$<>"C"
RETURN


3030 IFA$="P" COPY:RETURN
3040 IFA$="E" END
3050 IFA$="C" RUN300
3060 RETURN
```

A.P.C. Jan 85. V 6(1).
p 129-131
3 of 3.

Punch

Sock! Biff! Erk! Punch is a game for two players that simulates a boxing match. Full instructions are in the program.

Grant Rowe
Arncliffe NSW

□□□□

```
1 REM PUNCH - BOXING GAME
2 REM ONLY JOYSTICKS NEEDED,
3 REM WITH BASIC VZ-200 CONSOLE.
10 CLS:COLOR8,0:PRINT:PRINT
11 PRINT"  █████ █   █ █   █ ████ █   █"
12 PRINT"  █   █ █   █ ██  █ █    █   █"
13 PRINT"  █   █ █   █ █ █ █ █    █   █"
14 PRINT"  █████ █   █ █ █ █ █    █████"
15 PRINT"  █     █   █ █  ██ █    █   █"
16 PRINT"  █     █   █ █   █ █    █   █"
17 PRINT"  █     █████ █   █ ████ █   █"
18 SOUND28,1:SOUND29,8
19 PRINT:PRINT"   LADIES & GENTLEMEN,"
20 PRINT"THIS IS THE MAIN EVENT!!"
21 PRINT"LEFT PLAYER(7-LETTERS)";:INPUTL
$:IFLEN(L$)>7THEN21
22 PRINT"RIGHT PLAYER(7-LETTERS)";:INPUT
R$:IFLEN(R$)>7THEN22
23 FORI=1TO16:PRINT:NEXTI
24 PRINT"PUNCH IS A BOXING TYPE GAME,"
25 PRINT"FOR TWO PLAYERS.EACH PLAYER"
26 PRINT"USES HIS JOYSTICK TO MOVE AND"
27 PRINT"PUNCH HIS BOXERS ARMS.TO PUNCH,
"
29 PRINT"A PLAYER PUSHES HIS JOYSTICK,"
30 PRINT"IN EITHER THE LEFT OR RIGHT"
31 PRINT"POSITION.TO MOVE HIS ARMS,YOU"
32 PRINT"PUSH THE STICK EITHER UP OR "
33 PRINT"DOWN!"
34 PRINT" PRESS S TO START."
35 IFINKEY$="S"THEN100ELSEGOTO35
100 CLS:PRINT@35,L$;:PRINT@55,R$;
101 PRINT@388,"BLOWS";:PRINT@404,"BLOWS"
;
102 COLOR5:FORI=355TO380:PRINT@I,"█";:NE
XTI
103 PRINT@483,"1'''5'''10      1'''5'''1
0";
104 X=419:Y=435:V=0
105 COLOR2:PRINT@100,"████
  ████";
106         PRINT@132,"████
  ████";
107         PRINT@164,"████
  ████";
108 Q=0:FORI=1TO5:Q=Q+32:PRINT@164+Q,"HH
HH                ████";
109 NEXTI
120 F=200:F2=264:F3=215:F4=279:M2=0:N=0:
N2=0:M=0:M3=0:M4=0:N3=0
121 N4=0
130 A=(INP(43)AND31):B=(INP(46)AND31):IF
A=31ANDB=31THEN141
131 W=0:IFA=30ORA=29THENPRINT@F,"    ";:P
RINT@F2,"    ";
132 IFB=30ORB=29THENPRINT@F3-2,"    ";:PR
INT@F4-2,"    ";
133 IFA=30ANDN=0ANDN2=0THENF=F-32:IFF<20
0THENF=200:W=3
134 IFA=30ANDW=3THENF2=F2-32:IFF2<232THE
NF2=232
135 IFA=29ANDN=0ANDN2=0THENF2=F2+32:IFF2
>352THENF2=F2-32:W=4
136 IFA=29ANDW=4THENF=F+32:IFF>320THENF=
F-32
137 IFB=30ANDN3=0ANDN4=0THENF3=F3-32:IFF
3<215THENF3=215:W=1
138 IFB=30ANDW=1THENF4=F4-32:IFF4<247THE
NF4=247
139 IFB=29ANDN3=0ANDN4=0THENF4=F4+32:IFF
4>352THENF4=F4-32:W=2
140 IFB=29ANDW=2THENF3=F3+32:IFF3>320THE
NF3=F3-32
141 IFA=27THENF=F+1:M=M+1:N=1:GOSUB270
142 IFA=23THENF2=F2+1:M2=M2+1:N2=1:GOSUB
280
143 IFA<>23ANDN2=1THEN145ELSEIFA<>27ANDN
=1THEN144ELSE146
144 PRINT@F,"    ";:F=F-1:M=M-1:IFM=0THEN
N=0:GOTO146ELSEGOTO146
145 PRINT@F2,"    ";:F2=F2-1:M2=M2-1:IFM2
=0THENN2=0
146 IFB=27THENF3=F3-1:M3=M3-1:N3=1:GOSUB
250
147 IFB=23THENF4=F4-1:M4=M4-1:N4=1:GOSUB
260
148 IFB<>23ANDN4=1THEN210ELSEIFB<>27ANDN
3=1THEN200
150 GOSUB500:GOTO130
200 PRINT@F3-2,"    ";:F3=F3+1:M3=M3+1:IF
M3=0THENN3=0
202 GOTO150
210 PRINT@F4-2,"    ";:F4=F4+1:M4=M4+1:IF
M4=0THENN4=0
215 GOTO150
250 IFPEEK(28669+F3)=159THENGOSUB500:SOU
ND1,4:GOSUB300ELSEGOTO252
251 PRINT@F3-2,"          ";:F3=F3+8:M3=M3
+8:B=31:RETURN
252 IFPEEK(28669+F3)=175ANDM3<-3THENSOUN
D25,1:GOTO253ELSERETURN
253 SOUND27,1:PRINT@F3-2,"      ";:F3=F3+
3:M3=M3+3:RETURN
260 IFPEEK(28669+F4)=159THENGOSUB500:SOU ▶
```

YC Jan 85 p88-89 1 of 2.

```
ND1,4:GOSUB300ELSEGOTO262
261 PRINT@F4-2,"           ";:F4=F4+8:M4=M4
+8:RETURN
262 IFPEEK(28669+F4)=175ANDM4<-3THENSOUN
D25,1:GOTO263ELSERETURN
263 SOUND27,1:PRINT@F4-2,"      ";:F4=F4+
3:M4=M4+3:RETURN
270 IFPEEK(28674+F)=159THENSOUND1,4:GOSU
B400ELSEGOTO272
271 PRINT@F-8,"            ";:F=F-8:M=M-8:
RETURN
272 IFPEEK(28674+F)=175ANDM>3THENSOUND25
,1:GOTO273ELSERETURN
273 SOUND27,1:PRINT@F-3,"      ";:F=F-3:M
=M-3:RETURN
280 IFPEEK(28674+F2)=159THENSOUND1,4:GOS
UB400ELSEGOTO282
281 PRINT@F2-8,"            ";:M2=M2-8:F2=
F2-8:RETURN
282 IFPEEK(28674+F2)=175ANDM2>3THENSOUND
25,1:GOTO283ELSERETURN
283 SOUND27,1:PRINT@F2-3,"      ";:F2=F2-
3:M2=M2-3:RETURN
300 COLOR4:PRINT@X,"■";:PRINT@X+32,"■";:
X=X+1:IFX=429THEN302
301 RETURN
302 GOSUB700
303 PRINT@55,"WINNER";:FORI=1TO100:SOUND
RND(30),1:NEXTI:GOTO10
400 COLOR4:PRINT@Y,"■";:PRINT@Y+32,"■";:
Y=Y+1:IFY=445THEN402
401 RETURN
402 GOSUB700
403 PRINT@35,"WINNER";:FORI=1TO100:SOUND
RND(30),1:NEXTI:GOTO10
500 COLOR3:PRINT@F," ■■";:PRINT@F2," ■■"
;:PRINT@F3-2,"■■ ";
501 PRINT@F4-2,"■■ ";:COLOR2:PRINT@F,"■"
;:PRINT@F2,"■";
502 PRINT@F3,"■";:PRINT@F4,"■";
503 RETURN
700 IFY=445THENR=17ELSER=0
710 FORI=100TO324STEP32:PRINT@I+R,"
   ";:NEXTI
720 RETURN
```

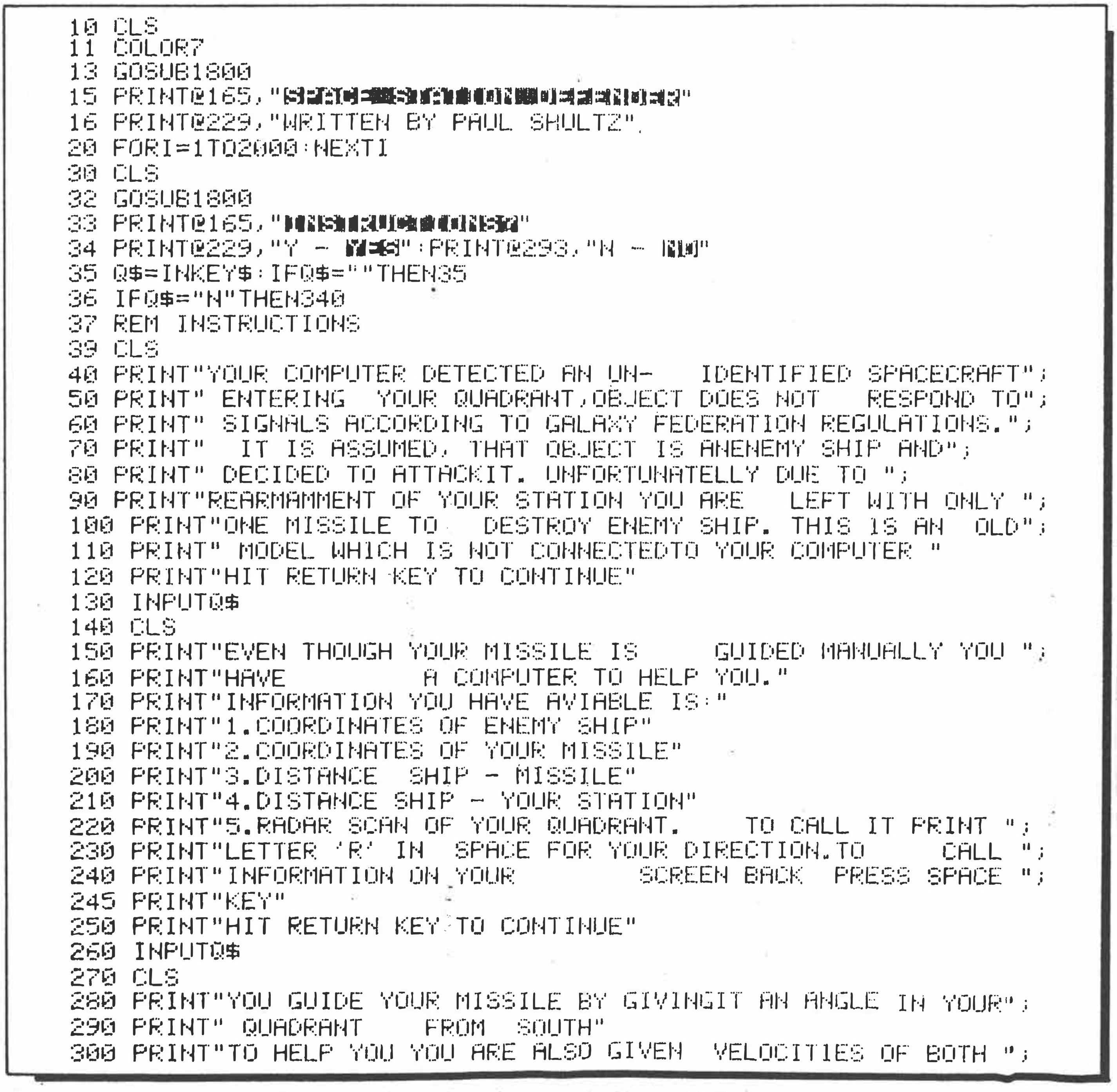
□□□□ ■

YC Jan 85 p 88-89 2 of 2.

SPACE STATION DEFENDER

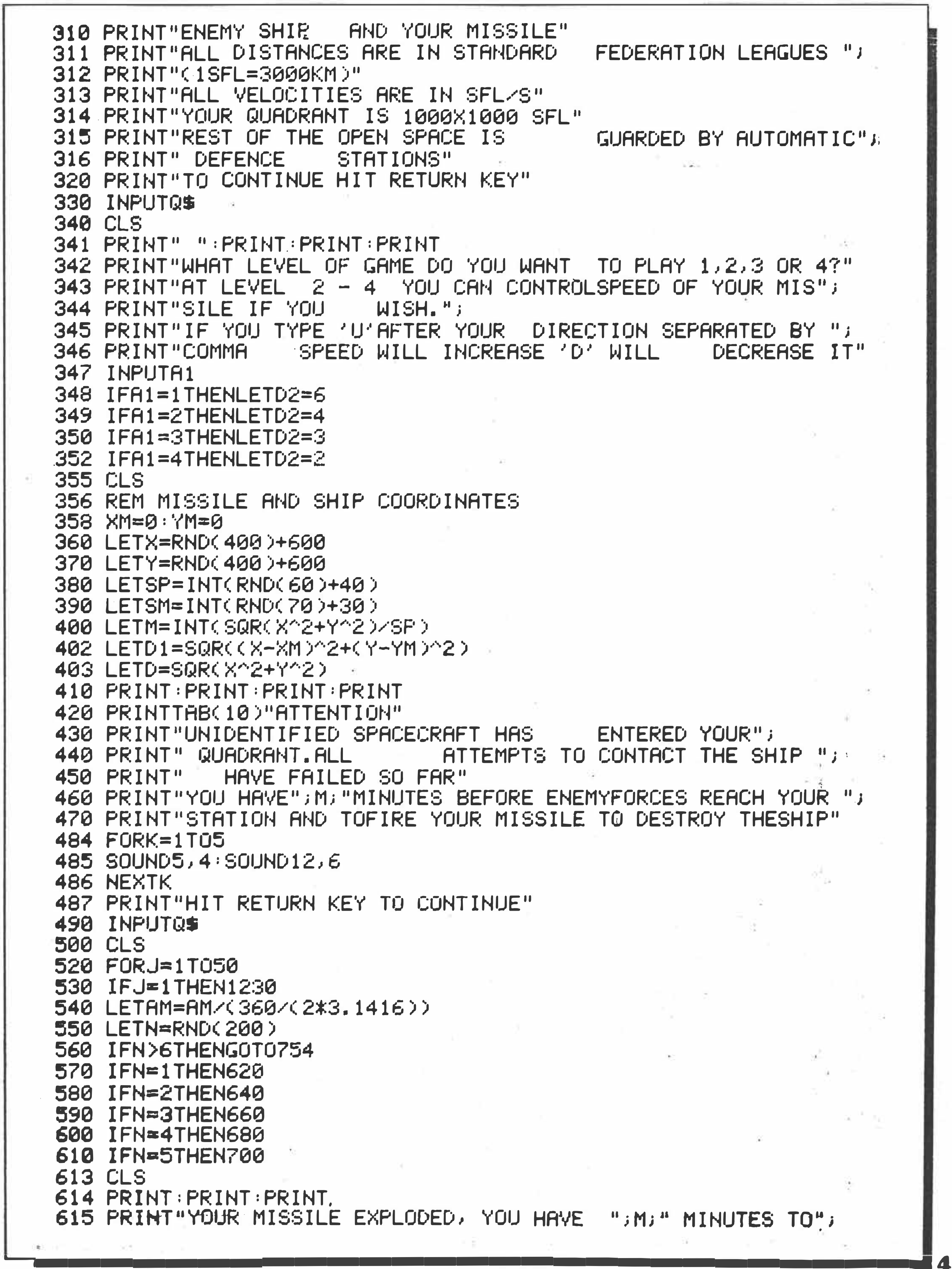
By Paul Shultz

In *Space Station Defender* you are requested (strangely enough) to defend a space station. You only have one missile left and your automatic missile control has broken down. You operate in the fourth quadrant and guide the missile by giving it an angle from 180 degrees. This is because of the radar scan display. Instructions are detailed in the listing and the only point to remember is that even in levels 2 to 5 velocity does not change. A comma must also be typed behind the angle figure.



```
10 CLS
11 COLOR7
13 GOSUB1800
15 PRINT@165,"SPACE STATION DEFENDER"
16 PRINT@229,"WRITTEN BY PAUL SHULTZ".
20 FORI=1TO2000:NEXTI
30 CLS
32 GOSUB1800
33 PRINT@165,"INSTRUCTIONS?"
34 PRINT@229,"Y - YES":PRINT@293,"N - NO"
35 Q$=INKEY$:IFQ$=""THEN35
36 IFQ$="N"THEN340
37 REM INSTRUCTIONS
39 CLS
40 PRINT"YOUR COMPUTER DETECTED AN UN-   IDENTIFIED SPACECRAFT";
50 PRINT" ENTERING  YOUR QUADRANT,OBJECT DOES NOT   RESPOND TO";
60 PRINT" SIGNALS ACCORDING TO GALAXY FEDERATION REGULATIONS.";
70 PRINT"  IT IS ASSUMED, THAT OBJECT IS ANENEMY SHIP AND";
80 PRINT" DECIDED TO ATTACKIT. UNFORTUNATELLY DUE TO ";
90 PRINT"REARMAMMENT OF YOUR STATION YOU ARE   LEFT WITH ONLY ";
100 PRINT"ONE MISSILE TO  DESTROY ENEMY SHIP. THIS IS AN  OLD";
110 PRINT" MODEL WHICH IS NOT CONNECTEDTO YOUR COMPUTER "
120 PRINT"HIT RETURN KEY TO CONTINUE"
130 INPUTQ$
140 CLS
150 PRINT"EVEN THOUGH YOUR MISSILE IS     GUIDED MANUALLY YOU ";
160 PRINT"HAVE           A COMPUTER TO HELP YOU."
170 PRINT"INFORMATION YOU HAVE AVIABLE IS:"
180 PRINT"1.COORDINATES OF ENEMY SHIP"
190 PRINT"2.COORDINATES OF YOUR MISSILE"
200 PRINT"3.DISTANCE  SHIP - MISSILE"
210 PRINT"4.DISTANCE SHIP - YOUR STATION"
220 PRINT"5.RADAR SCAN OF YOUR QUADRANT.    TO CALL IT PRINT ";
230 PRINT"LETTER 'R' IN  SPACE FOR YOUR DIRECTION.TO      CALL ";
240 PRINT"INFORMATION ON YOUR          SCREEN BACK  PRESS SPACE ";
245 PRINT"KEY"
250 PRINT"HIT RETURN KEY TO CONTINUE"
260 INPUTQ$
270 CLS
280 PRINT"YOU GUIDE YOUR MISSILE BY GIVINGIT AN ANGLE IN YOUR";
290 PRINT" QUADRANT    FROM  SOUTH"
300 PRINT"TO HELP YOU YOU ARE ALSO GIVEN  VELOCITIES OF BOTH ";
```

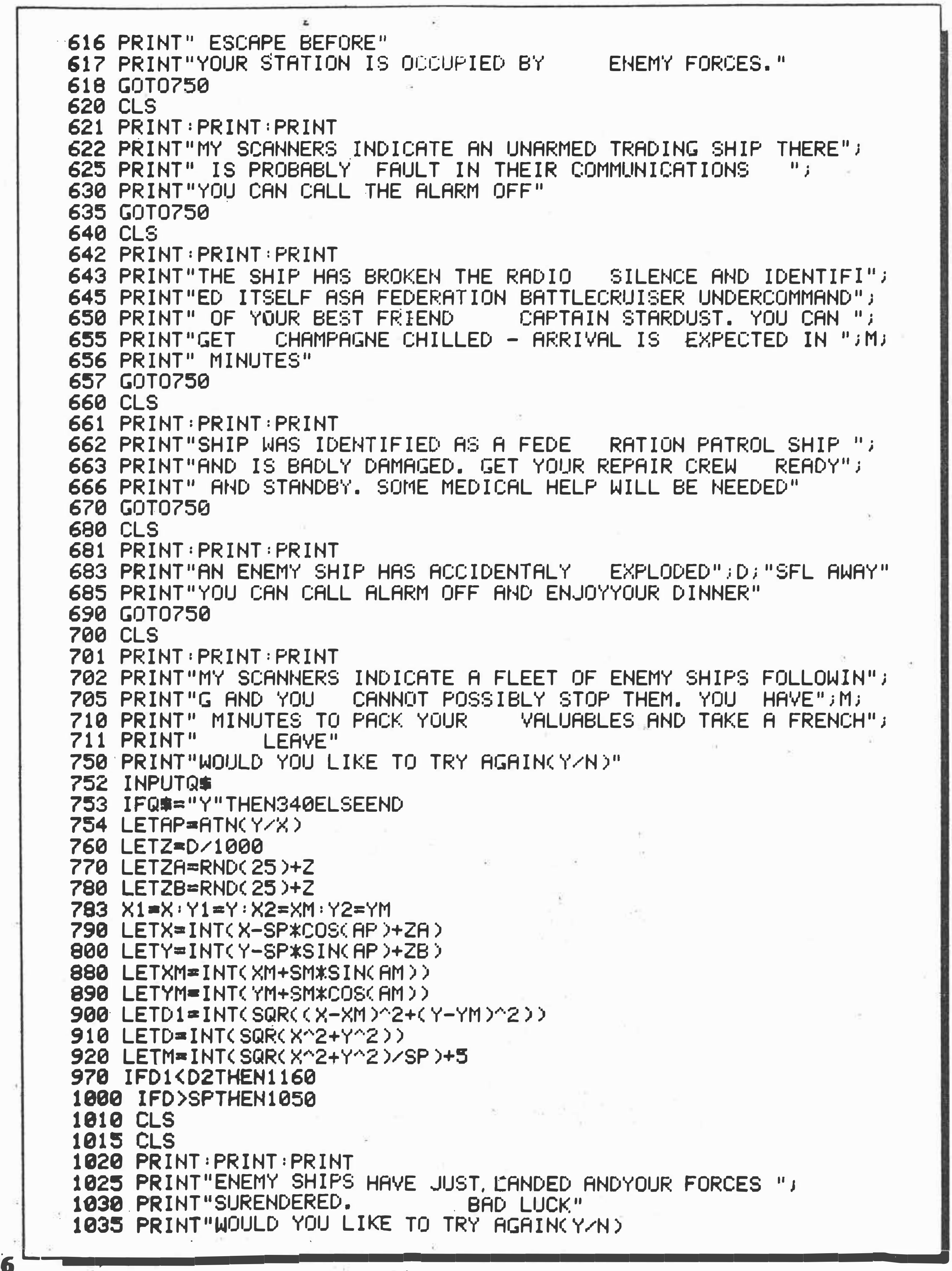
PCG. Jan 85   1 of 5



```
310 PRINT"ENEMY SHIP   AND YOUR MISSILE"
311 PRINT"ALL DISTANCES ARE IN STANDARD   FEDERATION LEAGUES ";
312 PRINT"(1SFL=3000KM)"
313 PRINT"ALL VELOCITIES ARE IN SFL/S"
314 PRINT"YOUR QUADRANT IS 1000X1000 SFL"
315 PRINT"REST OF THE OPEN SPACE IS        GUARDED BY AUTOMATIC";
316 PRINT" DEFENCE    STATIONS"
320 PRINT"TO CONTINUE HIT RETURN KEY"
330 INPUTQ$
340 CLS
341 PRINT"  ":PRINT:PRINT:PRINT
342 PRINT"WHAT LEVEL OF GAME DO YOU WANT  TO PLAY 1,2,3 OR 4?"
343 PRINT"AT LEVEL  2 - 4  YOU CAN CONTROLSPEED OF YOUR MIS";
344 PRINT"SILE IF YOU    WISH.";
345 PRINT"IF YOU TYPE 'U'AFTER YOUR  DIRECTION SEPARATED BY ";
346 PRINT"COMMA    SPEED WILL INCREASE 'D' WILL    DECREASE IT"
347 INPUTA1
348 IFA1=1THENLETD2=6
349 IFA1=2THENLETD2=4
350 IFA1=3THENLETD2=3
352 IFA1=4THENLETD2=2
355 CLS
356 REM MISSILE AND SHIP COORDINATES
358 XM=0:YM=0
360 LETX=RND(400)+600
370 LETY=RND(400)+600
380 LETSP=INT(RND(60)+40)
390 LETSM=INT(RND(70)+30)
400 LETM=INT(SQR(X^2+Y^2)/SP)
402 LETD1=SQR((X-XM)^2+(Y-YM)^2)
403 LETD=SQR(X^2+Y^2)
410 PRINT:PRINT:PRINT:PRINT
420 PRINTTAB(10)"ATTENTION"
430 PRINT"UNIDENTIFIED SPACECRAFT HAS     ENTERED YOUR";
440 PRINT" QUADRANT.ALL        ATTEMPTS TO CONTACT THE SHIP ";
450 PRINT"    HAVE FAILED SO FAR"
460 PRINT"YOU HAVE";M;"MINUTES BEFORE ENEMYFORCES REACH YOUR ";
470 PRINT"STATION AND TOFIRE YOUR MISSILE TO DESTROY THESHIP"
484 FORK=1TO5
485 SOUND5,4:SOUND12,6
486 NEXTK
487 PRINT"HIT RETURN KEY TO CONTINUE"
490 INPUTQ$
500 CLS
520 FORJ=1TO50
530 IFJ=1THEN1230
540 LETAM=AM/(360/(2*3.1416))
550 LETN=RND(200)
560 IFN>6THENGOTO754
570 IFN=1THEN620
580 IFN=2THEN640
590 IFN=3THEN660
600 IFN=4THEN680
610 IFN=5THEN700
613 CLS
614 PRINT:PRINT:PRINT.
615 PRINT"YOUR MISSILE EXPLODED, YOU HAVE  ";M;" MINUTES TO";
```

Jan 85

2 of 5



```
616 PRINT" ESCAPE BEFORE"
617 PRINT"YOUR STATION IS OCCUPIED BY     ENEMY FORCES."
618 GOTO750
620 CLS
621 PRINT:PRINT:PRINT
622 PRINT"MY SCANNERS INDICATE AN UNARMED TRADING SHIP THERE";
625 PRINT" IS PROBABLY  FAULT IN THEIR COMMUNICATIONS   ";
630 PRINT"YOU CAN CALL THE ALARM OFF"
635 GOTO750
640 CLS
642 PRINT:PRINT:PRINT
643 PRINT"THE SHIP HAS BROKEN THE RADIO   SILENCE AND IDENTIFI";
645 PRINT"ED ITSELF ASA FEDERATION BATTLECRUISER UNDERCOMMAND";
650 PRINT" OF YOUR BEST FRIEND     CAPTAIN STARDUST. YOU CAN ";
655 PRINT"GET   CHAMPAGNE CHILLED - ARRIVAL IS  EXPECTED IN ";M;
656 PRINT" MINUTES"
657 GOTO750
660 CLS
661 PRINT:PRINT:PRINT
662 PRINT"SHIP WAS IDENTIFIED AS A FEDE   RATION PATROL SHIP ";
663 PRINT"AND IS BADLY DAMAGED. GET YOUR REPAIR CREW   READY";
666 PRINT" AND STANDBY. SOME MEDICAL HELP WILL BE NEEDED"
670 GOTO750
680 CLS
681 PRINT:PRINT:PRINT
683 PRINT"AN ENEMY SHIP HAS ACCIDENTALY   EXPLODED";D;"SFL AWAY"
685 PRINT"YOU CAN CALL ALARM OFF AND ENJOYYOUR DINNER"
690 GOTO750
700 CLS
701 PRINT:PRINT:PRINT
702 PRINT"MY SCANNERS INDICATE A FLEET OF ENEMY SHIPS FOLLOWIN";
705 PRINT"G AND YOU   CANNOT POSSIBLY STOP THEM. YOU  HAVE";M;
710 PRINT" MINUTES TO PACK YOUR     VALUABLES AND TAKE A FRENCH";
711 PRINT"     LEAVE"
750 PRINT"WOULD YOU LIKE TO TRY AGAIN(Y/N)"
752 INPUTQ$
753 IFQ$="Y"THEN340ELSEEND
754 LETAP=ATN(Y/X)
760 LETZ=D/1000
770 LETZA=RND(25)+Z
780 LETZB=RND(25)+Z
783 X1=X:Y1=Y:X2=XM:Y2=YM
790 LETX=INT(X-SP*COS(AP)+ZA)
800 LETY=INT(Y-SP*SIN(AP)+ZB)
880 LETXM=INT(XM+SM*SIN(AM))
890 LETYM=INT(YM+SM*COS(AM))
900 LETD1=INT(SQR((X-XM)^2+(Y-YM)^2))
910 LETD=INT(SQR(X^2+Y^2))
920 LETM=INT(SQR(X^2+Y^2)/SP)+5
970 IFD1<D2THEN1160
1000 IFD>SPTHEN1050
1010 CLS
1015 CLS
1020 PRINT:PRINT:PRINT
1025 PRINT"ENEMY SHIPS HAVE JUST LANDED ANDYOUR FORCES ";
1030 PRINT"SURENDERED.          BAD LUCK"
1035 PRINT"WOULD YOU LIKE TO TRY AGAIN(Y/N)
```

Jan 85   3 of 5.

```
1040 INPUTQ$
1045 IFQ$="Y"THEN340ELSEEND
1050 IFD1<5THEN1160
1055 IFD1>SM+SPTHEN1230
1060 X3=(X-X1)/10:Y3=(Y-Y1)/10
1070 X4=(XM-X2)/10:Y4=(YM-Y2)/10
1080 FORL1=1TO10
1090 X1=X1+X3:Y1=Y1+Y3
1100 X2=X2+X4:Y2=Y2+Y4
1110 D1=INT(SQR((X1-X2)^2+(Y1-Y2)^2))
1120 IFD1<D2THEN1160
1130 NEXTL1
1150 GOTO1230
1155 CLS
1160 CLS
1161 PRINT:PRINT:PRINT:PRINT
1170 PRINT"CONGRATULATIONS YOUR MISSILE HASJUST DESTROYED THE "
1180 PRINT"ENEMY SHIP."
1190 PRINT"NOW YOU HAVE TIME TO RELAX AND  CELEBRATE."
1200 PRINT"DO YOU WANT TO PLAY AGAIN(Y/N)
1210 INPUTQ$
1220 IFQ$="Y"THEN340ELSEEND
1230 CLS
1231 REM COORDINATE PRINTOUT
1233 SOUND9,6
1235 PRINT"ENEMY SHIP IS";D;"SFL"
1243 PRINT"FROM YOU"
1245 PRINT
1250 PRINT"DISTANCE IN SFL"
1260 PRINT"ENEMY SHIP   MISSILE"
1270 PRINT"SOUTH:";Y;"  ";YM
1280 PRINT"EAST :";X;"  ";XM
1285 PRINT
1290 PRINT"ENEMY SHIP IS";D1;
1300 PRINT"SFL"
1305 PRINT"FROM YOUR MISSILE"
1306 PRINT
1310 PRINT"VELOCITY IN SFL/S"
1320 PRINT"ENEMY SHIP   MISSILE"
1330 PRINTUSING"####.#";SP;
1331 PRINT"     ";
1335 PRINTUSING"####.#";SM
1336 PRINT"YOUR DIRECTION IN DEGREES?"
1337 IFA1=1THEN1340
1338 INPUTQ$,A$
1339 GOTO1350
1340 INPUTQ$
1350 IFQ$="R"THEN1400
1360 LETAM=VAL(Q$)
1362 IFA1=1THEN1730
1363 REM VELOCITY UPDATE
1364 IFA$="U"ANDSM<97THENLETSM=SM+3
1365 IFA$="D"ANDSM>3THENLETSM=SM-3
1370 GOTO1730
1390 REM RADAR SCAN - GRAPHICS
1400 LETE=X/1000*126:LETF=Y/1000*60
1403 LETE1=XM/1000*126:LETF1=YM/1000*60
```

4 of 5.

Jan 85

```
1406 MODE(1)
1407 C$=INKEY$
1408 IFC$=" "THENGOTO1230
1410 COLOR2
1420 SET(1,0):SET(2,0):SET(0,1)
1430 SET(1,1):SET(2,1):SET(3,1)
1440 SET(0,2):SET(1,2):SET(2,2)
1450 SET(3,2):SET(1,3):SET(2,3)
1460 COLOR4
1470 FORI=1TO3
1480 SET(E,F+I)

1500 NEXT
1505 C$=INKEY$
1506 IFC$=" "THEN1230
1510 SET(E+1,F):SET(E+1,F+1)
1520 SET(E+2,F+1):SET(E,F+3)
1530 SET(E+1,F+2):SET(E+1,F+2)
1540 SET(E+2,F+2):SET(E+2,F+3)
1542 C$=INKEY$
1543 IFC$=" "THEN1230
1545 IFE1<0ORF1<0THEN1590
1547 IFE1>120ORF1>60THEN1590
1550 COLOR3
1555 FORI=1TO3
1556 SET(E1+1,F1+I)
1558 NEXT
1560 SET(E1+1,F1):SET(E1,F1+1)
1570 SET(E1+2,F1+1):SET(E1,F1+2).
1580 SET(E1+2,F1+2):SET(E1+2,F1+2)
1590 COLOR2

1600 FORI=1TO120
1610 SET(I,I/2):SET(I,60)
1620 NEXT
1625 C$=INKEY$
1626 IFC$=" "THEN1230
1630 FORI=3TO60
1640 SET(0,I)
1650 NEXT
1660 FORI=1TO2
1670 SET(24,61+I):SET(48,61+I)
1680 SET(72,61+I):SET(96,61+I)
1685 C$=INKEY$
1686 IFC$=" "THEN1230
1690 SET(96,61+I):SET(120,61+I)

1700 SET(I,12):SET(I,24):SET(I,36)
1710 SET(I,48):SET(I,60)
1720 GOTO1407
1730 NEXTJ
1800 PRINT"████████████████████████████████"
1810 FORI=1TO13:PRINT"█                                                  █";:NEXT
1820 PRINT"████████████████████████████████"
1830 RETURN
```

Jan 85   5 of 5.

VZ200

M. POTTER

Page 27   Computer Input Feb 85

LOST FORREST

This program will fit into an unexpanded VZ-200 but only just and spaces should be missed out where they are not necessary.

The Player can move from one location to another by using the commands:

N - NORTH
S - SOUTH
W - WEST
E - EAST

The command 'HELP' will display a list of verbs (or words) that the computer understands.

These words will aid the player in his search for the missing 'Green God' and several other treasures.

When the player thinks he has all of the missing treasures he should return to the location of the 'large trees' and type the command 'SCORE'. If all the treasures have been found then a winning message will be displayed.

```
6Ø CLEAR1ØØ
7Ø V=17:W=18:G=14
8Ø GOSUB13ØØ
9Ø CLS:PRINT"LOST FOREST"
1ØØ PRINT"-----------"
1Ø5 PRINT"MOVE NO.";M:IFM>5ØTHEN4ØØØ
11Ø PRINT"YOUR LOCATION"
12Ø PRINTD$(RM)
13Ø PRINT"EXITS:  ";
14Ø FORI=1TOLEN(R$(RM)):PRINTMID$(R$(RM),I,1);",";:NEXTI
17Ø PRINT:FORI=1TOG
19Ø IFL(I)=RMANDF(I)=ØTHENPRINT"YOU CAN SEE ";O$(I);" HERE"
2ØØ NEXTI
21Ø PRINT"===========================":PRINTM$:M$="WHAT"
23Ø INPUT"WHAT WILL YOU DO NOW";Q$
24Ø V$="":W$="":VB=Ø:OB=Ø
25Ø FORI=1TOLEN(Q$)
26Ø IFMID$(Q$,I,1)=" "ANDV$=""THENV$=LEFT$(Q$,I-1)
27Ø IFMID$(Q$,I+1,1)<>" "ANDV$<>""THENW$=MID$(Q$,I+1,LEN(Q$)-1)
275 IFMID$(Q$,I+1,1)<>" "ANDV$<>""THENI=LEN(Q$)
28Ø NEXTI
29Ø IFW$=""THENV$=Q$
3ØØ FORI=1TOV:IFV$=V$(I)THENVB=I
31Ø NEXTI
33Ø FORI=1TOW:IFW$=O$(I)THENOB=I
34Ø NEXTI
36Ø IFW$>""ANDOB=ØTHENM$="THAT'S SILLY"
37Ø IFVB=ØTHENVB=V+1
38Ø IFW$=""THENM$="I NEED TWO WORDS"
39Ø IFVB>VANDOB>ØTHENM$="YOU CAN'T '"+Q$+"'"
4ØØ IFVB>VANDOB=ØTHENM$="YOU DON'T MAKE SENSE"
410 IFVB<VANDOB>ØANDC(OB)=ØTHENM$="YOU DON'T HAVE '"+W$+"'"
46Ø IFVB=1THENGOSUB5ØØELSEIFVB=0THENGOSUB57Ø
462 IFVB=3ORVB=4ORVB=5ORVB=6ORVB=7THENGOSUB64Ø
466 IFVB=8ORVB=9THENGOSUB98Ø
47Ø IFVB=1ØTHENGOSUB1Ø3ØELSEIFVB=11THENGOSUB1Ø6Ø
472 IFVB=12THENGOSUB11ØØELSEIFVB=13THENGOSUB112Ø
474 IFVB=14THENGOSUB115ØELSEIFVB=15THENGOSUB117Ø
476 IFVB=16THENGOSUB121ØELSEIFVB=17THENGOSUB123Ø
478 IFVB=18THENGOSUB129Ø
49Ø GOTO9Ø
5ØØ PRINT"WORDS I KNOW: "
51Ø FORI=1TOV:PRINTV$(I);",";:NEXTI
54Ø M$="":PRINT:GOSUB128Ø
56Ø RETURN
57Ø PRINT"YOU ARE CARRYING: "
58Ø FORI=1TOG:IFC(I)=1THENPRINTO$(I);",";
59Ø NEXTI
61Ø M$="":PRINT:GOSUB128Ø
63Ø RETURN
64Ø D=Ø:IFOB=ØTHEND=VB-3
78Ø IFRM=6ANDF(8)<>2THENPRINT"YOU FALL ":GOTO35ØØ
785 IFRM=6ANDF(8)=2ANDD=4THENF(8)=Ø
79Ø IFF(12)=ØANDRM=8THENPRINT"THE EAGLE ATTACKS":GOTO3ØØØ
8ØØ IFRM=1ANDF(11)=ØTHENPRINT"HAWK ATTACKS SAVAGELY":GOTO3ØØØ
```

COMPUTER INPUT
Feb 85 p 27.

1 of 2.

```
850 RL=LEN(R$(RM)):F(17)=Ø
860 FORI=1TORL:U$=MID$(R$(RM),I,1)
880 IF(U$="N"ANDD=1ANDF(17)=Ø)THENRM=RM-3:M=M+1:F(17)=1
890 IF(U$="S"ANDD=2ANDF(17)=Ø)THENRM=RM+3:M=M+1:F(17)=1
900 IF(U$="W"ANDD=3ANDF(17)=Ø)THENRM=RM-1:M=M+1:F(17)=1
910 IF(U$="E"ANDD=4ANDF(17)=Ø)THENRM=RM+1:M=M+1:F(17)=1
920 NEXTI
930 M$="OK"
935 IFF(17)=ØTHENM$="YOU CAN'T GO THAT WAY"
940 IFD<1THENM$="GO WHERE"
970 RETURN
980 IFOB>9THENM$="I CAN'T GET "+W$:RETURN
982 IF(OB=1OROB=2)ANDC(6)=ØTHENM$="YOU NEED A CLOTH BAG":RETURN
985 IFL(OB)<>RMTHENM$="IT'S NOT HERE"
990 IFF(OB)<>ØTHENM$="WHAT "+W$+"?"
1000 IFC(OB)=1THENM$="YOU ALREADY HAVE IT"
1010 IFOB>ØANDL(OB)=RMANDF(OB)=ØTHENC(OB)=1:M$="YOU HAVE "+W$
1011 IFOB>ØANDL(OB)=RMANDF(OB)=ØTHENF(OB)=1
1020 RETURN
1030 IFRM=8ANDOB=5ANDC(5)=1ANDF(12)=ØTHENM$="IT FLYS OFF!!"
1031 IFRM=8ANDOB=5ANDC(5)=1ANDF(12)=ØTHENF(12)=1:C(5)=Ø
1040 IFRM=1ANDOB=5ANDC(5)=1ANDF(11)=ØTHENM$="MISSED!!!":C(5)=Ø
1046 IFOB=7ANDC(7)=1THENM$="IT BREAKS, OBJECT FALLS OUT"
1047 IFOB=7ANDC(7)=1THENF(1)=Ø:C(7)=Ø:L(1)=RM
1048 IFC(OB)=1THENM$="WHAT A TEMPER!!":C(OB)=Ø
1050 RETURN
1060 IFRM=1ANDOB=11ANDC(4)=1THENM$="IT FLYS OFF WITH BERRIES"
1070 IFRM=1ANDOB=11ANDC(4)=1THENF(11)=1:C(4)=Ø:RETURN
1090 RETURN
1100 IFRM=3ANDOB=10ANDF(10)=ØTHENM$="YOU MAY NEED THIS"
1101 IFRM=3ANDOB=10ANDF(10)=ØTHENF(6)=Ø:F(10)=1
1110 RETURN
1120 IFOB=13ANDRM=9THENM$="OK":RM=6:M=M+1
1140 RETURN
1150 IFRM=6ANDOB=8ANDC(8)=1THENM$="TRY THIS EXIT":F(8)=2:C(8)=Ø
1160 RETURN
1170 IFRM=17ANDOB=14ANDC(9)=1ANDF(14)=ØTHENM$="YOU DIG A HOLE"
1175 IFRM=17ANDOB=14ANDC(9)=1ANDF(14)=ØTHENF(14)=1:F(3)=Ø
1180 RETURN
1210 IFC(OB)=1THENC(OB)=Ø:L(OB)=RM:F(OB)=Ø:M$="DONE"
1220 RETURN
1230 S=Ø:FORI=1TOG:IFC(I)=1THENS=S+10
1240 NEXTI
1271 IFS=>5ØANDC(1)*C(2)*C(3)*C(6)*C(9)<>ØTHENPRINT"WELL DONE ";
1272 IFS=>5ØANDC(1)*C(2)*C(3)*C(6)*C(9)<>ØANDRM=16THEN1274
1273 PRINT"SCORE SO FAR=";S:GOTO128Ø
1274 SC=(5Ø-M)+S+1:PRINT"YOUR SCORE=";SC
1275 PRINT"YOU'VE WON!!":GOTO3Ø2Ø
1280 INPUT"PRESS RETURN TO CONTINUE";CT$
1290 RETURN
1300 DIMR$(17),D$(17),O$(W),V$(V),C(W),L(G),F(W)
1320 DATA13,Ø,17,15,11,3,13,6,6,3,1,8,9,17
1330 FORI=1TOG:READL(I):NEXTI
1360 DATAHELP,INV,GØ,N,S,W,E,GET,TAKE,THROW,FEED,ROLL,CLIMB
1365 DATADROP,DIG,LEAVE,SCORE
1380 FORI=1TOV:READV$(I):NEXTI
1410 DATAE,WE,SW,E,SWE,NSW,E,NE,NSW,SE,SWE,NW,NE,NSWE,SW,E
1420 DATANW,N
1430 FORI=ØTO17:READR$(I):NEXTI
1520 DATATREES,MORE TREES,EVEN MORE TREES,ANOTHER TREE,TREES
1530 DATAA TREE,UP TREE,TREE,TALL TREES,GREEN TREE,MORE TREES
1540 DATAOAK TREE,BIG TREE,OVERGROWN TREE,SHORT TREE
1550 DATATREES,LARGE TREES,MANY TREES
1580 FORI=ØTO17:READD$(I):NEXTI
1610 DATAGREEN GOD,LAURELS,GOLD LEAF,BERRIES,ACORN,SACK
1620 DATABEEHIVE,ROPE,TROWEL,LOG,HAWK,EAGLE,GREEN TREE
1630 DATATREE,NORTH,SOUTH,EAST,WEST
1640 FORI=1TOW:READO$(I):NEXTI
1670 F(1)=1:F(3)=1:F(6)=1:RM=RND(18)-1
1675 IFRM<9THEN167Ø
1680 M=1:RETURN
3000 FORI=1TO2ØØØ:NEXTI:CLS:PRINT"YOU ARE NOW OFFICIALLY DEAD"
3010 PRINT"YOU HAVE NOT MANAGED TO COMPLETEYOUR FOREST WALK"
3020 PRINT:INPUT"DO YOU WANT TO PLAY AGAIN";PA$
3030 IFLEFT$(PA$,1)="Y"THEN6Ø
3040 CLS:END
3500 FORI=1TO2ØØØ:NEXTI:CLS
3510 PRINT"YOU'VE LANDED IN A WOLF TRAP"
3520 PRINT:GOTO3Ø1Ø
4000 CLS:PRINT"YOUR TIME HAS RUN OUT AND'YOU   ARE TOO WEAK TO";
4010 PRINT"CONTINUE":GOTO3Ø1Ø
```

COMPUTER INPUT p 28. Feb. 85 2 of 2.

```
1 REM DECOY GAME FOR VZ-200
2 REM WRITTEN BY GRANT ROWE
4 POKE30862,80:POKE30863,52
5 COLOR8,0
10 CLS
15 PRINT
20 PRINT"        ▛▖ ▛▘ ▛▘ ▛▘ ▌▌"
30 PRINT"        ▌▌ ▛▘ ▌  ▌▌  ▌"
40 PRINT"        ▀  ▀▀ ▀▀ ▀▀  ▀"
50 PRINT
55 PRINT"LEFT JOYSTICK TO MOVE SHUTTLE,"
56 PRINT"EITHER BUTTON TO FIRE."
57 PRINT"YOU ARE TO HOVER OVER A PART"
58 PRINT"OF THE PLANET,ZELTA.WHILE OUR"
59 PRINT"FIGHTERS ARE TO ATTACK ON"
60 PRINT"THE OTHER SIDE OF THE PLANET,"
61 PRINT"YOU ARE THE DECOY FOR ZELTA"
62 PRINT"SHIPS AND GROUND FIRE..."
63 PRINT"WARNING:DON'T LEAVE ATMOSPHERE.
"
64 PRINT"GOOD LUCK..PRESS S TO START."
70 L$=INKEY$:IFL$="S"THENGOTO100ELSEGOTO
70
100 S=0:M=3:H=20
120 MODE(1):COLOR,0
130 FORI=127TO0STEP-1:COLOR3:SET(I,62):N
EXTI
135 P=20:P2=31
140 FORI=127TO0STEP-1:X=RND(4)
150 IF X=2ORX=3THENJ=3
160 IFX=1ORX=4THENJ=2
161 COLORJ:IFX=3THENSET(I,60)
163 SET(I,61):NEXTI
170 N=0
180 K=0:Z=0:KY=0:GZ=0:KZ=0:JR=15
190 BN=0
194 COLOR4:XN=0
195 GOSUB500:FORI=1TO100:UX=USR(UX):COLO
R,XN:XN=XN+1
196 IFXN>1THENXN=0
197 NEXTI:COLOR,0
200 A=(INP(43)AND31)
201 CR=RND(10):IFH<14THENCR=RND(20)
202 IFH<6THENCR=RND(28)
203 COLOR4:R=RND(126):SET(R,CR)
204 IFCR>JRTHENJR=CR
205 IF A=31THENGOTO300
210 IF A=26THENGOSUB550:GOSUB600:GOSUB61
0:GOTO300
220 IF A=25THENGOSUB550:GOSUB600:GOSUB63
0:GOTO300
230 IF A=22THENGOSUB550:GOSUB610:GOSUB62
0:GOTO300
240 IF A=21THENGOSUB550:GOSUB620:GOSUB63
0:GOTO300
250 IF A=30THENGOSUB550:GOSUB610:GOTO300
260 IF A=29THENGOSUB550:GOSUB630:GOTO300
270 IF A=27THENGOSUB550:GOSUB600:GOTO300
280 IF A=23THENGOSUB550:GOSUB620
300 GOSUB500
305 AZ=(INP(39)AND31)
310 IF A=15ORAZ=15THENGOSUB900
315 IFN=1THENGOSUB990:GOTO330
320 N=RND(H):IFN=1THENC=P-3:C2=60:IFS>25
00THENKY=1
330 IFK=1THENGOSUB750:GOTO340
335 K=RND(H):IFK=1THENE=30+RND(75):EN=58
340 IFZ=1THENGOSUB800:GOTO346
345 IFK=1THENZ=RND(H):IF Z=1THENL=E+2:L2
=EN+3
346 XG=RND(2):IFGZ=1THENGOSUB400:GOTO370
347 IFK=1ANDEN>P2-5ANDEN<P2+2ANDXG=1THEN
VZ=E-3:VY=EN:GZ=1
370 IFKZ=1THENGOTO380
371 IFK=1THENKZ=RND(H):IF KZ=1THENYZ=E+3
:YY=EN-3
372 GOTO200
380 RESET(YZ,YY):RESET(YZ+1,YY):YY=YY-2:
IFYY<12THENKZ=0:GOTO200
381 IFYZ>P-8ANDYZ<P+1ANDYY>P2-2ANDYY<P2+
2THEN1000
382 COLOR4:SET(YZ,YY):SET(YZ+1,YY):GOTO2
00
400 RESET(VZ,VY):RESET(VZ+1,VY):RESET(VZ
+2,VY)
410 VZ=VZ-3:IFVZ<1THENGZ=0:RETURN
420 COLOR4:SET(VZ,VY):SET(VZ+1,VY):SET(V
Z+2,VY)
430 IF VY>P2-3ANDVY<P2+2ANDVZ>P-8ANDVZ<P
+1THEN1000
440 RETURN
500 COLOR8:SET(P,P2):SET(P-1,P2):SET(P-2
,P2):SET(P-3,P2)
505 SET(P-4,P2):SET(P-5,P2):SET(P-6,P2):
SET(P-3,P2+1)
510 SET(P-4,P2+1):SET(P-5,P2+1):SET(P-4,
P2-1):SET(P-5,P2-1)
515 SET(P-5,P2-2):COLOR6:SET(P-3,P2-1):R
ETURN
550 RESET(P,P2):RESET(P-1,P2):RESET(P-2,
P2):RESET(P-3,P2)
555 RESET(P-4,P2):RESET(P-5,P2):RESET(P-
6,P2):RESET(P-3,P2+1)
560 RESET(P-4,P2+1):RESET(P-5,P2+1):RESE
T(P-4,P2-1)
565 RESET(P-5,P2-1):RESET(P-5,P2-2):RESE
T(P-3,P2-1):RETURN
600 P=P-5:IFP<10THENP=106
605 RETURN
610 P2=P2-4:IFP2<JRTHEN2000
615 RETURN
620 P=P+5:IFP>106THENP=10
625 RETURN
630 P2=P2+4:IFP2>55THENP2=55
```

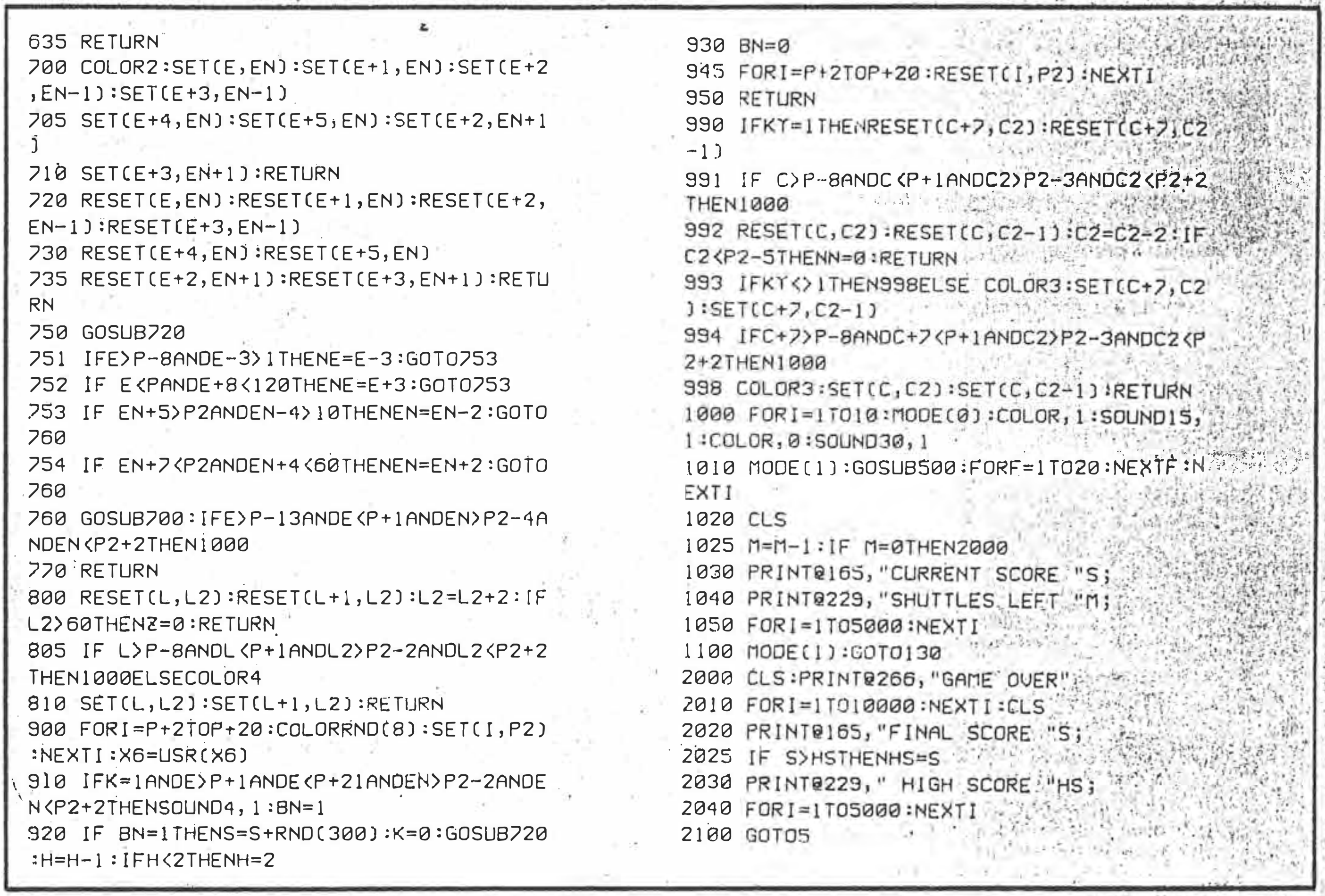
Decoy

Decoy can be played on the VZ-200, and requires joysticks. High resolution is used and instructions are given in the program. As the game progresses, so does the degree of difficulty.

Grant Rowe
Arncliffe, NSW

►

YC Mar 85 p 105 + 109.
1 of 2.



```
635 RETURN
700 COLOR2:SET(E,EN):SET(E+1,EN):SET(E+2
,EN-1):SET(E+3,EN-1)
705 SET(E+4,EN):SET(E+5,EN):SET(E+2,EN+1
)
710 SET(E+3,EN+1):RETURN
720 RESET(E,EN):RESET(E+1,EN):RESET(E+2,
EN-1):RESET(E+3,EN-1)
730 RESET(E+4,EN):RESET(E+5,EN)
735 RESET(E+2,EN+1):RESET(E+3,EN+1):RETU
RN
750 GOSUB720
751 IFE>P-8ANDE-3>1THENE=E-3:GOTO753
752 IF E<PANDE+8<120THENE=E+3:GOTO753
753 IF EN+5>P2ANDEN-4>10THENEN=EN-2:GOTO
760
754 IF EN+7<P2ANDEN+4<60THENEN=EN+2:GOTO
760
760 GOSUB700:IFE>P-13ANDE<P+1ANDEN>P2-4A
NDEN<P2+2THEN1000
770 RETURN
800 RESET(L,L2):RESET(L+1,L2):L2=L2+2:IF
L2>60THENZ=0:RETURN
805 IF L>P-8ANDL<P+1ANDL2>P2-2ANDL2<P2+2
THEN1000ELSECOLOR4
810 SET(L,L2):SET(L+1,L2):RETURN
900 FORI=P+2TOP+20:COLORRND(8):SET(I,P2)
:NEXTI:X6=USR(X6)
910 IFK=1ANDE>P+1ANDE<P+21ANDEN>P2-2ANDE
N<P2+2THENSOUND4,1:BN=1
920 IF BN=1THENS=S+RND(300):K=0:GOSUB720
:H=H-1:IFH<2THENH=2
930 BN=0
945 FORI=P+2TOP+20:RESET(I,P2):NEXTI
950 RETURN
990 IFKY=1THENRESET(C+7,C2):RESET(C+7,C2
-1)
991 IF C>P-8ANDC<P+1ANDC2>P2-3ANDC2<P2+2
THEN1000
992 RESET(C,C2):RESET(C,C2-1):C2=C2-2:IF
C2<P2-5THENN=0:RETURN
993 IFKY<>1THEN998ELSE COLOR3:SET(C+7,C2
):SET(C+7,C2-1)
994 IFC+7>P-8ANDC+7<P+1ANDC2>P2-3ANDC2<P
2+2THEN1000
998 COLOR3:SET(C,C2):SET(C,C2-1):RETURN
1000 FORI=1TO10:MODE(0):COLOR,1:SOUND15,
1:COLOR,0:SOUND30,1
1010 MODE(1):GOSUB500:FORF=1TO20:NEXTF:N
EXTI
1020 CLS
1025 M=M-1:IF M=0THEN2000
1030 PRINT@165,"CURRENT SCORE "S;
1040 PRINT@229,"SHUTTLES LEFT "M;
1050 FORI=1TO5000:NEXTI
1100 MODE(1):GOTO130
2000 CLS:PRINT@266,"GAME OVER"
2010 FORI=1TO10000:NEXTI:CLS
2020 PRINT@165,"FINAL SCORE "S;
2025 IF S>HSTHENHS=S
2030 PRINT@229," HIGH SCORE "HS;
2040 FORI=1TO5000:NEXTI
2100 GOTO5
```

"Decoy"

YC Mar 85. p 105 + 109
2 of 2.

MOUSE MAZE
D.CRANDALL
From COMPUTER INPUT MARCH 1985

Help "mouse" by moving him around the maze so that he gets the cheese,seeing how quickly he can do it.All other instructions are in the game.

```
1 CLS
2 PRINT@43,"MOUSE MAZE"
3 PRINT@96,"MOVE THE MOUSE (*) ARO
UND THE  "
4 PRINT@128,"MAZE HOLDING DOWN <<C
TRL>> AND  "
5 PRINT@160,"USING THE CURSOR KEYS
 SO THAT"
6 PRINT@192,"HE CAN GET THE CHEESE
 (#)."
7 REM'CHROMATIC SCALE'
8 REM'FROM A TO D#'
10 FORS=1TO31
15 SOUNDS,1
20 NEXT
21 PRINT@257,"HIT 'I' FOR THE INVI
SIBLE MAZE"
22 PRINT@322,"HIT 'V' FOR THE VISI
BLE MAZE"
23 IFINKEY$="I"THENPOKE30744,1
24 IFINKEY$="V"THENPOKE30744,0
25 IFINKEY$=""THENGOTO23
27 PRINT@450,"HIT 'S'TO START"
30 IFINKEY$<>"S"THEN30
40 SOUND28,1
85 Q=28671
86 QQ=28736
87 QR=28863
88 P=28736
89 ST=0
90 C=128
95 T=28700+RND(430)
100 CLS
105 FORX=1TO32:POKE28671+X,C:POKE2
9151+X,C:NEXT
110 FORY=1TO32:POKEQQ+32*Y,C:NEXT
115 FORY=1TO9:POKEQR+32*Y,C:NEXT
117 TM=0
120 REM READ POKES
130 READA
135 IFA=-99THEN490
140 POKEQ+A,C
150 GOTO130
490 POKEP,42
495 POKET,35
500 IFP=TTHEN800
502 Z$=INKEY$:Z$=INKEY$:ST=ST+1
503 IFZ$=""THEN502
505 Z=ASC(Z$)
510 IFZ=9THENN=P+1:GOTO550
515 IFZ=8THENN=P-1:GOTO550
520 IFZ=27THENN=P-32:GOTO550
525 IFZ=10THENN=P+32:GOTO550
530 TM=TM+1
540 GOTO502
550 X=PEEK(N)
555 IFX=CTHEN502
560 POKEP,96:POKEN,106
565 P=N
570 GOTO500
800 REM
801 FORT=1TO31
802 SOUNDT,1
804 NEXT
805 CLS
810 PRINT@224,"YOU GOT THE CHEESE
IN A TIME OF";ST;"!!"
820 PRINT:PRINT"    WANT TO TRY AG
AIN? (Y/N)";
830 Z$=INKEY$:IFZ$=""THEN830
840 IFZ$="Y"THENRUN
850 IFZ$="N"THENEND
860 GOTO830

1000 DATA33,37,41,48,54,67,71,73,75,76,77,78,80,82,83,84,86,88
1005 DATA84,96,128,160
1010 DATA90,91,92,93,94,99,100,101,102,103,105,108,112,116,122
1020 DATA131,135,137,138,140,142,144,146,148,149,150,152,153,154
1030 DATA156,157,158,159,165,167,172,174,176,182,186
1040 DATA195,196,197,199,200,201,202,203,204,206,208,209,210
1050 DATA211,212,214,216,218,219,221,222,223
1060 DATA227,229,235,238,246,248,250,259,260,261,262,263,264,265
1070 DATA267,269,270,272,273,274,275,277,280,284,285,286,287
1080 DATA291,301,307,309,311,312,313,314,315,316,323,325,326,327
1090 DATA328,329,330,331,332,333,335,337,341,346,350
1100 DATA355,357,361,367,371,373,374,376,380,382,391
1110 DATA395,396,397,398,399,400,401,402,408,409,410,411,312
1120 DATA419,420,421,422,423,424,425,426,427,431,436,437,438
1130 DATA439,440,444,445,446,461,465,474
1140 DATA-99
```

COMPUTER INPUT MAR. 85.

VZ200

PAINTER

Painter is a challenging game where scoring is difficult. The program uses joysticks but can easily be modified to use the keyboard instead.

The aim of the game is to paint as much of the screen as possible before you run out of space. You must avoid crossing your tracks, the border around the screen and the randomly placed red landmines.

Bruce Daniel
Mudgee, NSW

```
10 ' PAINTER - BY BRUCE DANIEL
20 HS=0
30 CLS
40 FORI=28704 TO 29119 : POKE I,128 : NEXTI
50 FORI=1 TO 30:POKE 28672+I,179 :POKE 29120+I,188 :NEXTI
60 FORI=28704 TO 29088 STEP 32:POKE I,181:POKE I+31,186 :NEXTI
70 POKE 28672,177:POKE 28703,178:POKE 29120,180:POKE 29151,184
80 FORI=1T04+RND(4):POKE 28672+RND(12)*32+RND(28)+34,191:NEXTI
90 SC=0:MV=1:CP=28704:COLOR2
100 PRINT@495,"HIGH SCORE:";:HS$=STR$(HS)
110 HS$=RIGHT$(HS$,LEN(HS$)-1)
120 IFLEN(HS$)<3THENHS$="0"+HS$:GOTO120ELSEPRINTHS$;:SOUND23,3
130 PRINT@481,"SCORE :";:SC$=STR$(SC):SC$=RIGHT$(SC$,LEN(SC$)-1)
140 IFLEN(SC$)<3THENSC$="0"+SC$:GOTO140ELSEPRINTSC$;
150 JK=INP(43)ANDINP(46)AND31
160 IFJK=30THENMV=-32ELSEIFJK=29THENMV=32
170 IFJK=27THENMV=-1ELSEIFJK=23THENMV=1
180 CP=CP+MV
190 IFPEEK(CP)<>128THEN220
200 POKECP,159:SC=SC+1:GOTO130
210 '
220 PRINT@267,"GAME OVER";' INVERSE
230 SOUND16,1
240 IFSC>HSTHENHS=SC
250 PRINT@417,"     PRESS <FIRE> TO PLAY     ";'INVERSE
260 JK=INP(43)ANDINP(46)AND31
270 IFJK<>15THEN260
280 GOTO 30
```

Your Computer. Apr. 85.
p 160.

Joystick Movement.

JK = INP (43) AND INP (46) AND 31

IF JK = 30 THEN MV = -32 ELSE IF JK = 29, MV = 32

IF JK = 27 THEN MV = -1 ELSE IF JK = 23, MV = 1

CP = CP + MV

CP is position of moving object.

30 = ↑ 29 = ↓ 27 = ← 23 = →

15 = FIRE.

ROADRACE

By Ian Thompson

Imagine yourself at the wheel of a high speed racing car winding along a treacherous course. To stay on course, you must steer accurately or risk a collision with the side fences. By adjusting the road width and visibility conditions, *Roadrace* can be made as easy or as challenging as you wish.

The road width can be set between 4 and 15 characters, the degree of difficulty changing with different widths. Visibility can be set to any of four settings. When visibility is good, the car appears high on the screen. This allows a good view of the twisting road ahead. When visibility is poor, the car appears low on the screen allowing only a brief look at the coming road.

After a five step starting light count down the race begins, the twisting and turning road moving continuously on the screen.

The car is steered by the use of the left and right cursor control keys.

The race proceeds until the car crashes off the road. Each collision is considered to terminate one day of the race. After each day, you are shown the distance achieved that day along with the cumulative distance achieved for consecutive days of the race.

Main routines

140- 250	Variable initialisation and graphics display.
300- 420	Accepts road conditions from user.
500- 540	Initialises the road.
600- 650	Determines the next road condition.
700- 750	Updates the car position, determines if crash has occurred.
800-1050	Processes end of race.
1400-1600	Draws next road segment.
2000-2200	Initialises string variables.
3000-3640	Initial graphics display.
4000-4090	Graphics to start race.

Main variables

W	Road width.
V	Visibility.
M	Distance driven on current day.
N	Number of days of the race.
T	Total distance driven for whole race.
H	Elapsed time during race.
L$,R$	String characters to move car left, right.
L	Position of left side of road.
LC,RC	Random value to move road left, right.
EL,ER	Leftmost, rightmost allowable road position.
Q$	User replies.
Z	Screen location of car.
RS$,RL$	Strings to display road segments.
G	First address of screen memory.
C$	Character string for car.

The program occupies 2.8k of memory.

Apr. 1985 p 65-67

1 of 3.

Modifications for TRS-80

The following line modifications will allow the program to run on the TRS-80 Color Computer.

```
 160 CR=3: CC=3
 210 G=1024
 730 IF PEEK (Z+G) < >144 THEN 800
 740 IF PEEK (Z+G+1) < > 144 THEN 800
 910 PRINT@480,CHR$ (143)
4000 Q=175:K=179
```

The SOUND and COLOR statements must also be changed as appropriate for the TRS-80.

```
0 '***************************
1 '*     R O A D R A C E     *
2 '***************************
3 '*    V Z - 2 0 0 (8K)     *
4 '* IAN THOMPSON -COLLAROY *
5 '***************************
100 SOUND 28,6
140 CLEAR 200
150 LC=0.45
160 CR=3:CC=3
170 L$=CHR$(77):R$=CHR$(44)
200 RC=1-LC
210 G=28672
250 GOSUB 3000
300 GOSUB 3600
310 T=0:N=0
315 CLS:PRINT
320 INPUT"ENTER ROAD WIDTH (4-15)";W
330 W=INT(W):PRINT
340 IF W<4 OR W>15 THEN 310
350 PRINT"VISIBILITY CONDITIONS"
360 PRINT"  1 - TERRIBLE"
370 PRINT"  2 - BAD"
380 PRINT"  3 - FAIR"
390 PRINT"  4 - GOOD"
395 PRINT@280,""
400 INPUT"ENTER VISIBILITY (1-4)";V
410 V=INT(V):GOSUB 2000
420 IF V<1 OR V>4 THEN 395
500 N=N+1:EL=449:ER=478-W:H=0
510 Z=527-64*V:L=463-INT(W/2)
520 FOR J=1 TO 16:PRINT@480,B$;
530 GOSUB 1400:Q$=INKEY$:NEXT
540 PRINT@Z,C$;:GOSUB 4000
600 H=H+1:Q=RND(0):PRINT@480,B$;
610 IF Q>RC AND L<ER THEN 640
620 IF Q<LC AND L>EL THEN 650
630 GOSUB 1400:GOTO 700
640 GOSUB 1600:GOTO 700
650 GOSUB 1500
700 Q$=INKEY$
710 IF Q$=L$ THEN Z=Z-1
720 IF Q$=R$ THEN Z=Z+1
730 IF PEEK(Z+G)<>144 THEN 800
```

Apr 85.

p 65-67

2 of 3.

```
740 IF PEEK(Z+G+1)<>144 THEN 800
750 PRINT@Z,C$;:GOTO 600
800 FOR J=1 TO 6:Q$=INKEY$
810 PRINT@Z,D$;:SOUND 31,2
820 FOR K=1 TO 10:NEXT
830 PRINT@Z,C$;
840 FOR K=1 TO 10:NEXT:NEXT
900 M=H/50:T=T+M
910 PRINT@480,CHR$(127)
920 PRINT"YOU WENT";M;"KILOMETERS"
925 PRINT"FOR A TOTAL OF";T;"KILOMETERS"
930 PRINT"IN";N;"DAY(S)":PRINT
940 PRINT"HIT <C> - CONTINUE RACE"
950 PRINT"    <R> - RESTART RACE"
960 PRINT"    <Q> - QUIT"
970 Q$=INKEY$
980 IF Q$="C" THEN 500
990 IF Q$="R" THEN 1010
1000 IF Q$<>"Q" THEN 970
1010 PRINT
1020 PRINT"AVERAGE KILOMETERS PER DAY "
1030 PRINT"WAS";T/N;"KM."       ← FOR J=0 TO 500: NEXT (delay)
1040 IF Q$="R" THEN 310
1050 END
1400 COLOR2:PRINT@L,RS$;:COLOR3:RETURN
1500 COLOR2:L=L-1:PRINT@L,RL$;:COLOR3:RETURN
1600 COLOR2:PRINT@L,RR$;:L=L+1:COLOR3:RETURN
2000 Q=121+CC*16:K=118+CC*16
2010 C$=CHR$(Q)+CHR$(K)
2020 Q=127+CR*16:RS$=CHR$(Q)
2030 FOR J=1 TO W
2040 RS$=RS$+CHR$(128):NEXT
2050 RS$=RS$+CHR$(Q)
2060 Q=119+CR*16:K=120+CR*16
2070 RL$=CHR$(Q)+CHR$(K)
2080 FOR J=1 TO (W-1)
2090 RL$=RL$+CHR$(128):NEXT
2100 RL$=RL$+CHR$(Q)+CHR$(K)
2110 Q=116+CR*16:K=123+CR*16
2120 RR$=CHR$(Q)+CHR$(K)
2130 FOR J=1 TO (W-1)
2140 RR$=RR$+CHR$(128):NEXT
2150 RR$=RR$+CHR$(Q)+CHR$(K)
2160 B$="":FOR J=1 TO 32
2170 B$=B$+CHR$(128):NEXT
2180 D$=CHR$(128)+CHR$(128)
2200 RETURN
3000 W=7:GOSUB 2000:CLS
3010 FOR J=1 TO 15:READ Q
3015 COLOR3
3020 PRINT@Q,RS$;:NEXT
3030 FOR J=1 TO 600:NEXT
3035 COLOR2
3040 RESTORE:FOR J=1 TO 6
3050 READ Q:PRINT@Q+3,C$;
3060 FOR K=1 TO 100:NEXT:NEXT
3070 T$="ROADRACE":FOR J=1 TO 8
3080 READ Q:PRINT@Q+3,C$;
3090 Q$=CHR$(128)+MID$(T$,J,1)
3100 FOR K=1 TO 100:NEXT
3110 PRINT@Q+3,Q$;:NEXT
3120 READ Q:PRINT@Q+3,C$;
3130 SOUND 26,4
3140 FOR J=1 TO 500:NEXT
3160 RETURN
3200 DATA 12,44,77,110,141,172
3210 DATA 205,238,271,304,337
3220 DATA 370,403,436,469
3600 A$=INKEY$:IF INKEY$<>"" THEN 3600
3610 PRINT@448,"HIT ANY KEY TO BEGIN"
3620 Q=RND(0):Q$=INKEY$
3630 IF Q$="" THEN 3620
3640 RETURN
4000 Q=175:K=179
4010 N$=CHR$(Q)+CHR$(Q)+CHR$(Q)
4020 M$=CHR$(Q)+CHR$(K)+CHR$(Q)
4030 Q=Z-INT(W/2)-5:K=Q-128
4040 FOR J=K TO Q STEP 32
4045 COLOR4
4050 PRINT@J,N$;:NEXT
4060 FOR J=K TO Q STEP 32
4070 FOR R=1 TO 300:NEXT
4080 PRINT@J,M$;:SOUND 28,4
4090 NEXT:COLOR2
4100 RETURN
```

Apr 85. p. 65-67

3 of 3.

NUMBER SEQUENCE

This program prints various sequences of numbers, each ending with a blank. You must enter the next number in the sequence — the computer indicates if your entry was correct.

A series of ten questions is asked, then your score is given.

Because the program is written in standard Microsoft BASIC, it should be easily transported to other computers. The random number statements in lines 120-140 may need modification, according to your particular version of BASIC.

Ian Thompson
Collaroy Plateau NSW

```
1  '******************************
2  '*      NUMBER SEQUENCE       *
3  '* FOR THE UNEXPANDED VZ-200 *
4  '* IAN THOMPSON - COLLAROY    *
5  '******************************
10 CLS:PRINT@104,"NUMBER SEQUENCE"
12 PRINT@325,"IAN THOMPSON,COLLAROY"
15 PRINT@485,"PRESS ANY KEY TO START"
20 IF INKEY$="" THEN 20
21 IF INKEY$="" THEN 20
25 CLS:PRINT"         NUMBER SEQUENCE":PRINT
30 PRINT"THIS PROGRAM WILL PRINT VARIOUS"
35 PRINT"SEQUENCES OF NUMBERS, EACH       "
40 PRINT"ENDING WITH A BLANK (----)."
45 PRINT"WHEN YOU SEE A '?', TYPE IN THE"
50 PRINT"NUMBER THAT YOU THINK THE "
55 PRINT"COMPUTER MIGHT HAVE PRINTED IN "
60 PRINT"PLACE OF THE BLANK."
70 PRINT
75 PRINT"********************************"
80 LET R=0
90 LET W=0
100 FOR I=1 TO 10
110 PRINT"PROBLEM";I
120 LET A=INT(10*RND(0)+1)
130 LET B=INT(10*RND(0)+1)
140 LET G=RND(3)
150 IF A>B THEN 285
160 IF G=1 THEN 170
162 IF G=2 THEN 210
164 IF G=3 THEN 250
170 LET X=2*A+3*B
180 PRINT A;",";B;",";A+B;",";A+2*B;", ----";
190 INPUT Y
200 GOTO 410
210 LET X=A*A*B*B*B
220 PRINT A;",";B;",";A*B;",";B*A*B;", ----";
230 INPUT Y
240 GOTO 410
250 LET X=-B
260 PRINT A;",";B;",";B-A;",";-A;", ----";
270 INPUT Y
280 GOTO 410
285 IF G=1 THEN 300
290 IF G=2 THEN 340
300 LET X=A*5
310 PRINT A;",";2*A;",";3*A;",";4*A;", ----";
320 INPUT Y
330 GOTO 410
340 LET X=16*A
350 PRINT A;",";2*A;",";4*A;",";8*A;", ----";
360 INPUT Y
410 IF X=Y THEN 450
420 PRINT"NO; THE COMPUTER'S SEQUENCE HAS ";X;"."
430 LET W=W+1
440 GOTO 470
450 PRINT"THAT'S RIGHT!"
460 LET R=R+1
470 PRINT
480 NEXT I
485 SOUND 15,5
490 PRINT"================================"
500 PRINT"SCORE: ";R;" RIGHT,";W;" WRONG
505 PRINT:PRINT"================================"
510 PRINT" PRESS <SPACE> FOR ANOTHER SET   OF";
520 PRINT" QUESTIONS."
530 A$=INKEY$:IF A$ <> " "THEN 530
535 RUN
```

YC May 85 p. 106.

SKETCHPAD
By Ian Thompson

This program allows you to use the computer as a sketchpad. Two versions of the sketchpad are available, the first being low resolution graphics using the characters above the T, I, D and J keys. The second version makes use of high resolution graphics to allow drawings of much finer detail.

In both programs you control the crea tion of the picture using the arrow keys.

Low resolution graphics

During the running of the program, use is made of the eight colour keys along the top of the keyboard to change colour during drawing. As well as the colour keys 1-8, the following keys are also available for use while drawing.
upper J graphics
upper D graphics
upper T graphics
upper I graphics
G — light green background
O — orange background
G and B — dark green background
O and B — red background
Z — rubout background
C — clear screen
R — re-run the program
P — copy to printer [GP-100]
H — move to high resolution

High resolution graphics

In this mode you have a choice of two background colours, green and buff.

These colours, and the foreground colours for drawing are selected from the eight colour keys along the top of the keyboard.

The following summarises the colours available.

GREEN BACKGROUND
1 — rubout
2 — yellow
3 — blue
4 — red
BUFF BACKGROUND
5 — rubout
6 — cyan
7 — magenta
8 — orange

The following keys are also used to control the program.
C — clear the screen
R — re-run the program
P — copy to printer [GP-100]
L — move to low resolution graphics

Due to limitations of the printer, the Print statements in lines 815, 900, 1000, 1085 and 1115 should be entered in inverse text.

The program occupies 6.2k of memory.

```
2   '***************************
5   '*   VZ-200 SKETCHPAD       *
10  '*   16K EXPANSION REQUIRED *
15  '***************************
20  '*   IAN A.THOMPSON         *
25  '*   COLLAROY PLATEAU -- NSW *
30  '***************************
32  '
35 SOUND 25,6
95 GOTO 800'TITLE GRAPHICS & INSTRUCTIONS
100 GOSUB 2000'INITIALISES CURSOR CONTROL (ARROW) KEYS
130 PRINT@(32*Y+X)," ";
135 PRINT@(32*Y+X),CHR$(143);'UPPER CASE J
```

May/Jun 85
63 - 67
1 of 5.

```
155 IFC$="R"THEN RUN
175 IFC$="D"THEN 400
176 IFC$="T"THEN 500
178 IFC$="I"THEN 200
180 IFC$="Z"THEN 300
181 IFC$="1"THEN COLOR1:GOTO100
182 IFC$="2"THEN COLOR2:GOTO100
183 IFC$="3"THEN COLOR3:GOTO100
184 IFC$="4"THEN COLOR4:GOTO100
185 IFC$="5"THEN COLOR5:GOTO100
186 IFC$="6"THEN COLOR6:GOTO100
187 IFC$="7"THEN COLOR7:GOTO100
188 IFC$="8"THEN COLOR8:GOTO100
189 IFC$="G"THEN POKE30744,0:COLOR,0:GOTO100
190 IFC$="O"THEN POKE30744,0:COLOR,1:GOTO100
191 IFC$="B"THEN POKE30744,1:GOTO100
192 IFC$="H"THEN 1000
193 IFC$="C"THEN 960
194 IFC$="P"THEN COPY:GOTO100
195 GOTO 100
200 GOSUB 2000
230 PRINT@(32*Y+X)," ";
235 PRINT@(32*Y+X),CHR$(133);'UPPER CASE I
255 IFC$="R"THEN RUN
275 IFC$="D"THEN 400
276 IFC$="T"THEN 500
278 IFC$="J"THEN 100
280 IFC$="Z"THEN 300
281 IFC$="1"THEN COLOR1:GOTO200
282 IFC$="2"THEN COLOR2:GOTO200
283 IFC$="3"THEN COLOR3:GOTO200
284 IFC$="4"THEN COLOR4:GOTO200
285 IFC$="5"THEN COLOR5:GOTO200
286 IFC$="6"THEN COLOR6:GOTO200
287 IFC$="7"THEN COLOR7:GOTO200
288 IFC$="8"THEN COLOR8:GOTO200
289 IFC$="G"THEN POKE30744,0:COLOR,0:GOTO200
290 IFC$="O"THEN POKE30744,0:COLOR,1:GOTO200
291 IFC$="B"THEN POKE30744,1:GOTO200
292 IFC$="H"THEN 1000
293 IFC$="C"THEN 960
294 IFC$="P"THEN COPY:GOTO200
295 GOTO 200
300 GOSUB 2000
330 PRINT@(32*Y+X)," ";
335 PRINT@(32*Y+X),CHR$(128);'UPPER CASE Z
355 IFC$="R"THEN RUN
360 IFC$="C"THEN 960
370 IFC$="9"THEN 600
375 IFC$="J"THEN 100
380 IFC$="D"THEN 400
385 IFC$="T"THEN 500
387 IFC$="I"THEN 200
389 IFC$="G"THEN POKE30744,0:COLOR,0:GOTO300
390 IFC$="O"THEN POKE30744,0:COLOR,1:GOTO300
391 IFC$="B"THEN POKE30744,1:GOTO300
394 IFC$="P"THEN COPY:GOTO300
395 GOTO 300
400 GOSUB 2000
```

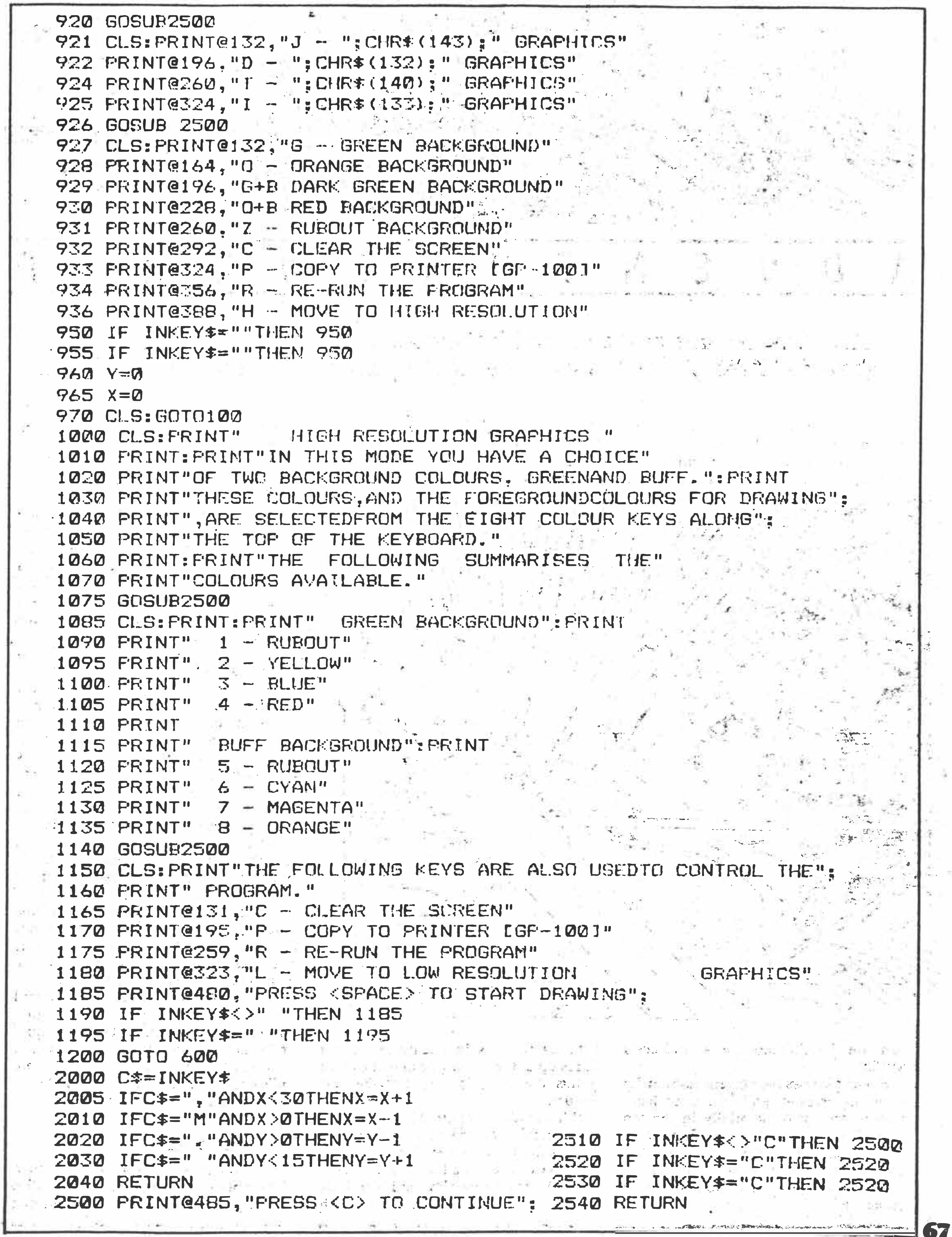
May/Jun 85
63-67
2 of 5.

```
430 PRINT@(32*Y+X)," ";
435 PRINT@(32*Y+X),CHR$(132);'UPPER CASE D
455 IFC$="R"THEN RUN
475 IFC$="I"THEN 200
476 IFC$="T"THEN 500
478 IFC$="J"THEN 100
480 IFC$="Z"THEN 300
481 IFC$="1"THEN COLOR1:GOTO400
482 IFC$="2"THEN COLOR2:GOTO400
483 IFC$="3"THEN COLOR3:GOTO400
484 IFC$="4"THEN COLOR4:GOTO400
485 IFC$="5"THEN COLOR5:GOTO400
486 IFC$="6"THEN COLOR6:GOTO400
487 IFC$="7"THEN COLOR7:GOTO400
488 IFC$="8"THEN COLOR8:GOTO400
489 IFC$="G"THEN POKE30744,0:COLOR,0:GOTO400
490 IFC$="O"THEN POKE30744,0:COLOR,1:GOTO400
491 IFC$="B"THEN POKE30744,1:GOTO400
492 IFC$="H"THEN 1000
493 IFC$="C"THEN 960
494 IFC$="P"THEN COPY:GOTO400
495 GOTO 400
500 GOSUB 2000
530 PRINT@(32*Y+X)," ";
535 PRINT@(32*Y+X),CHR$(140);'UPPER CASE T
555 IFC$="R"THEN RUN
575 IFC$="I"THEN 200
576 IFC$="D"THEN 400
578 IFC$="J"THEN 100
580 IFC$="Z"THEN 300
581 IFC$="1"THEN COLOR1:GOTO500
582 IFC$="2"THEN COLOR2:GOTO500
583 IFC$="3"THEN COLOR3:GOTO500
584 IFC$="4"THEN COLOR4:GOTO500
585 IFC$="5"THEN COLOR5:GOTO500
586 IFC$="6"THEN COLOR6:GOTO500
587 IFC$="7"THEN COLOR7:GOTO500
588 IFC$="8"THEN COLOR8:GOTO500
589 IFC$="G"THEN POKE30744,0:COLOR,0:GOTO500
590 IFC$="O"THEN POKE30744,0:COLOR,1:GOTO500
591 IFC$="B"THEN POKE30744,1:GOTO500
592 IFC$="H"THEN 1000
593 IFC$="C"THEN 960
594 IFC$="P"THEN COPY:GOTO500
595 GOTO 500
600 REM****MODE 1 SKETCHER
605 CLS
610 MODE(1)
630 X=0
640 Y=0
650 C$=INKEY$
675 IFC$=","ANDX<127THENX=X+1
680 IFC$="M"ANDX>0THENX=X-1
685 IFC$="."ANDY>0THENY=Y-1
689 IFC$=" "ANDY<63THENY=Y+1
690 SET(X,Y)
691 IFC$="1"THEN COLOR1,0:GOTO650
692 IFC$="2"THEN COLOR2:GOTO650
693 IFC$="3"THEN COLOR3:GOTO650
```

may/Jun 85
63- 67
3 of 5.

```
694 IFC$="4"THEN COLOR4:GOTO650
695 IFC$="5"THEN COLOR5,1:GOTO650
696 IFC$="6"THEN COLOR6:GOTO650
697 IFC$="7"THEN COLOR7:GOTO650
698 IFC$="8"THEN COLOR8:GOTO650
700 IFC$="L"THEN GOTO900
701 IFC$="R"THEN RUN
703 IFC$="C"THEN 600
704 IFC$="P"THEN COPY:GOTO650
705 GOTO 650
800 CLS:POKE30744,1:COLOR3,0
810 PRINT:PRINT
815 PRINT"      S K E T C H  P A D "
825 FORN=1TO1000
826 NEXTN
830 A$="IAN THOMPSON, COLLAROY PLATEAU"
835 FORN=1TOLEN(A$)
840 PRINT@289,RIGHT$(A$,N);
845 NEXT
847 PRINT@450,"COPYRIGHT <C> FEBRUARY 1985"
849 FORN=1TO1500
850 NEXTN
855 CLS:PRINT"THIS PROGRAM ALLOWS YOU TO USE "
856 PRINT"THE COMPUTER AS A SKETCHPAD."
857 PRINT
858 PRINT"TWO VERSIONS OF THE SKETCHPAD  "
860 PRINT"ARE AVAILABLE, THE FIRST BEING "
862 PRINT"LOW RESOLUTION GRAPHICS USING  "
864 PRINT"THE CHARACTERS ABOVE THE T,I,D  AND J KEYS."
865 PRINT
866 PRINT"THE SECOND VERSION MAKES USE OF"
868 PRINT"HIGH  RESOLUTION  GRAPHICS TO "
870 PRINT"ALLOW  DRAWINGS  OF MUCH FINER  DETAIL TO BE MADE.":PRINT
871 GOSUB2500
874 CLS:PRINT"IN BOTH PROGRAMS, YOU CONTROL  "
875 PRINT"THE  CREATION  OF THE  PICTURE "
876 PRINT"USING THE  ARROW  KEYS IN THE"
878 PRINT"LOWER RIGHT HAND CORNER OF THE "
880 PRINT"KEYBOARD."
885 PRINT@227,"INPUT CHOICE"
886 PRINT@291,"A - LOW RESOLUTION"
888 PRINT@355,"B - HIGH RESOLUTION"
889 PRINT@241,"";
890 PRINT@241,"";:INPUTA$
892 IFA$="A"THEN GOTO 900
894 IFA$="B"THEN GOTO 1000
896 GOTO890
900 CLS:PRINT"   LOW RESOLUTION GRAPHICS "
902 PRINT:PRINT"DURING  THE  RUNNING  OF  THE  "
904 PRINT"PROGRAM USE IS MADE OF THE EIGHTCOLOUR KEYS ALONG";
906 PRINT" THE  TOP OF   THE  KEYBOARD  TO";
908 PRINT" CHANGE COLOURSDURING DRAWING."
910 PRINT:PRINT"AS WELL AS THE COLOUR KEYS 1-8."
912 PRINT"THE  FOLLOWING  KEYS  ARE  ALSO"
914 PRINT"AVAILABLE  FOR USE  DURING THE  DRAWING."
916 GOSUB2500
917 CLS:PRINT:PRINT"IT IS SUGGESTED THAT YOU MAKE"
918 PRINT"A NOTE ON A PIECE OF PAPER OF"
919 PRINT"THE FOLLOWING KEYS TO BE USED   DURING DRAWING."
```

May/Jun 85
63-67.
4 of 5.



```
920 GOSUB2500
921 CLS:PRINT@132,"J - ";CHR$(143);" GRAPHICS"
922 PRINT@196,"D - ";CHR$(132);" GRAPHICS"
924 PRINT@260,"T - ";CHR$(140);" GRAPHICS"
925 PRINT@324,"I - ";CHR$(133);" GRAPHICS"
926 GOSUB 2500
927 CLS:PRINT@132,"G - GREEN BACKGROUND"
928 PRINT@164,"O - ORANGE BACKGROUND"
929 PRINT@196,"G+B DARK GREEN BACKGROUND"
930 PRINT@228,"O+B RED BACKGROUND"
931 PRINT@260,"Z - RUBOUT BACKGROUND"
932 PRINT@292,"C - CLEAR THE SCREEN"
933 PRINT@324,"P - COPY TO PRINTER [GP-100]"
934 PRINT@356,"R - RE-RUN THE PROGRAM"
936 PRINT@388,"H - MOVE TO HIGH RESOLUTION"
950 IF INKEY$=""THEN 950
955 IF INKEY$=""THEN 950
960 Y=0
965 X=0
970 CLS:GOTO100
1000 CLS:PRINT"    HIGH RESOLUTION GRAPHICS "
1010 PRINT:PRINT"IN THIS MODE YOU HAVE A CHOICE"
1020 PRINT"OF TWO BACKGROUND COLOURS, GREENAND BUFF.":PRINT
1030 PRINT"THESE COLOURS,AND THE FOREGROUNDCOLOURS FOR DRAWING";
1040 PRINT",ARE SELECTEDFROM THE EIGHT COLOUR KEYS ALONG";
1050 PRINT"THE TOP OF THE KEYBOARD."
1060 PRINT:PRINT"THE  FOLLOWING  SUMMARISES  THE"
1070 PRINT"COLOURS AVAILABLE."
1075 GOSUB2500
1085 CLS:PRINT:PRINT"  GREEN BACKGROUND":PRINT
1090 PRINT"  1 - RUBOUT"
1095 PRINT"  2 - YELLOW"
1100 PRINT"  3 - BLUE"
1105 PRINT"  4 - RED"
1110 PRINT
1115 PRINT"  BUFF BACKGROUND":PRINT
1120 PRINT"  5 - RUBOUT"
1125 PRINT"  6 - CYAN"
1130 PRINT"  7 - MAGENTA"
1135 PRINT"  8 - ORANGE"
1140 GOSUB2500
1150 CLS:PRINT"THE FOLLOWING KEYS ARE ALSO USEDTO CONTROL THE";
1160 PRINT" PROGRAM."
1165 PRINT@131,"C - CLEAR THE SCREEN"
1170 PRINT@195,"P - COPY TO PRINTER [GP-100]"
1175 PRINT@259,"R - RE-RUN THE PROGRAM"
1180 PRINT@323,"L - MOVE TO LOW RESOLUTION          GRAPHICS"
1185 PRINT@480,"PRESS <SPACE> TO START DRAWING";
1190 IF INKEY$<>" "THEN 1185
1195 IF INKEY$=" "THEN 1195
1200 GOTO 600
2000 C$=INKEY$
2005 IFC$=","ANDX<30THENX=X+1
2010 IFC$="M"ANDX>0THENX=X-1
2020 IFC$="."ANDY>0THENY=Y-1
2030 IFC$=" "ANDY<15THENY=Y+1
2040 RETURN
2500 PRINT@485,"PRESS <C> TO CONTINUE";
2510 IF INKEY$<>"C"THEN 2500
2520 IF INKEY$="C"THEN 2520
2530 IF INKEY$="C"THEN 2520
2540 RETURN
```

May/Jun 85
63-67
5 of 5.

MORSE TUTOR PROGRAM

This program runs on the standard TRS80 MC10 with 4 Kbytes of memory, and should also be suitable for the TRS80 CoCo. It runs random Morse in groups of five characters. You may select the number of characters to be reproduced (up to 200), the speed (up to 15 words per minute) and to have letters, numbers or both. A delay between letters and words may also be selected.

The program starts by sounding the preset characters, and on completion they are printed on the screen. There is provision to re-run without resetting the variables, and an auto-run facility that prints the checklist on-screen, pauses, then re-runs.

When you call for 200 characters the computer is using very close to 4 Kbytes. For this reason, line numbers were kept low to take up less memory and no 'anti-crash' programming has been done. If you make an incorrect entry during the menu setup, the program may indicate an error, in which case you will have to re-run the program.

If you're using a CoCo use the word 'pause' instead of 'delay' in lines 12, 29 and 80; the CoCo doesn't seem to like the word 'delay'.

Basil Heath,
Hamilton, Qld

```
1 CLS
2 PRINT"AUTO-RUN":PRINT"YES(1)":
PRINT"NO-(2)"
4 CLEAR 500
5 DATA 63,62,60,56,48,32,33,35,3
9,47,6,17,21,9,2,20,11,16,4,30,1
3,18,7,5,15,22,27,10,8,3,12,24,1
4,25,29,19
6 INPUT R
7 DIM B$(36)
8 FOR I=1 TO 36:READ J:LET B$(I)
=CHR$(J)
9 NEXT I:CLS
10 INPUT"SPEED(WPM)(MAX 15)?";SP
EED
11 LET SPEED=7.5/SPEED
12 INPUT"DELAY(0-15)?";DELAY:DEL
AY=DELAY*50
13 INPUT"NO:-CHARACTERS(MAX 200)
";N
14 INPUT"LETTERS(1)NUMBERS(2)OR
BOTH(3)?";L
15 DIM T$(N)
16 CLS:PRINT TAB(5)"MORSE TUTOR
PROGRAM":FOR I=1 TO N
17 LET T$(I)=CHR$(RND(10*-1*(L<>
1)+26*-1*(L<>2))+10*-1*(L=1)):NE
XT I
18 FOR I=1 TO N
19 LET X=ASC(B$(ASC(T$(I))))
20 GOSUB 65
23 IF I=INT(I/5)*5 THEN 29
25 IF I=N THEN 32
27 NEXT I
29 FOR Z=1 TO INT(200*SPEED+(DEL
AY*5)):NEXT Z
30 GOTO 25
32 FOR I=1 TO N
35 IF ASC(T$(I))>10 THEN 39
37 PRINT CHR$(ASC(T$(I))+47);
38 GOTO 40
39 PRINT CHR$(ASC(T$(I))+54);
40 IF I=INT(I/25)*25 THEN 46
41 IF I=INT(I/5)*5 THEN 44
42 IF I=N THEN 49
43 NEXT I
44 PRINT" ";
45 GOTO 42
46 PRINT
47 GOTO 42
49 IF R<>2 THEN 90
50 PRINT:PRINT:PRINT
51 PRINT"PRESS KEY(1)(ENTER)TO R
E-TRY":PRINT "PRESS KEY(2)(ENTER
) TO EXIT"
52 INPUT P:IF P<>2 THEN 16
53 DATA 80,82,79,71,82,65,77,32,
66,89,58,45,32,66,46,72,69,65,84
,72,32,86,75,52,65,66,72
54 CLS:PRINT:PRINT
57 FOR I=1 TO 27
59 READ A
61 PRINT CHR$(A);
63 NEXT I:END
65 LET Y=X/2:LET X=INT(Y)
67 Q=(2*SPEED*(1+(Y-X)*4))
70 SOUND 200,Q
75 IF X=1 THEN 80
77 FOR Z=1 TO INT(40*SPEED):NEXT
Z
78 GOTO 65
80 FOR Z=1 TO INT(120*SPEED+(DEL
AY*3)):NEXT Z
85 RETURN
90 PRINT:PRINT:PRINT"PRESS-'BREA
K'-TO EXIT"
95 FOR I=1 TO 10000:NEXT I:GOTO
16
```

YC Jun. 85 p. 70

(Program for MC10 - but see VZ version in YC Jan 86 p 150.)

VZ200

MORSE TUTOR (again)

In the June '85 issue of *Your Computer* we published a Morse Tutor program written by Basil Heath for the TRS-80 MC10. It was wrongly listed as being intended for the VZ200. As a result, Basil received several letters and phone calls from VZ200 users who pointed out first that the print had been reduced so much it was difficult to read, and second that the program didn't work (for obvious reasons).

Basil has very kindly collaborated with a friend who owns a VZ200 in rewriting his program for that machine. Here we've listed both versions (with the right machine headings, this time). Our apologies to the many people we mislead by this mistake.

```
1 REM MORSE TUTOR PROGRAM:CLS
2 PRINT"AUTO-RUN":PRINT"YES(1)":PRINT"NO(2)"
4 CLEAR 500
5 DATA 63,62,60,56,48,32,33,35,39,47,6,17,21,9,2,20,11,16,4,30
6 DATA 13,18,7,5,15,22,27,10,8,3,12,24,14,25,29,19
7 INPUTR:DIMB$(36)
8 FORI=1TO36:READJ:LETB$(I)=CHR$(J)
9 NEXTI:CLS
10 INPUT"SPEED(WPM)(MAX 10)?";SPEED
11 LET SPEED=5.0/SPEED
12 INPUT"DELAY(0-15)?";DELAY:DELAY=DELAY*50
13 INPUT"NO:-CHARACTERS(MAX 200)";N
14 INPUT"LETTERS(1)NUMBERS(2)OR BOTH(3)?";L
15 DIM T$(N)
16 CLS:PRINTTAB(5)"MORSE TUTOR PROGRAM":FORI=1TON
17 LETT$(I)=CHR$(RND(10*-1*(L<>1)+26*-1*(L<>2))+10*-1*(L=1))
18 NEXTI:FORI=1TON
19 LETX=ASC(B$(ASC(T$(I))))
20 GOSUB65
23 IF I=INT(I/5)*5THEN29
25 IFI=NTHEN32
27 NEXTI
29 FOR Z=1 TO INT(200*SPEED+(DELAY*5)):NEXTZ
30 GOTO25
32 FORI=1TON
34 IF ASC(T$(I))>10 THEN39
37 PRINTCHR$(ASC(T$(I))+47);
38 GOTO40
39 PRINTCHR$(ASC(T$(I))+54);
40 IF I=INT(I/25)*25THEN46
41 IF I=INT(I/5)*5THEN44
42 IF I=NTHEN49
43 NEXTI
44 PRINT" ";
45 GOTO42
46 PRINT
47 GOTO42
49 IF R<>2THEN90:PRINT:PRINT:PRINT
50 PRINT"PRESS KEY(1)(ENTER) TO RE-TRY"
51 PRINT"PRESS KEY(2)(ENTER) TO EXIT"
52 INPUTP:IFP<2THEN16
53 DATA 80,82,79,71,82,65,77,32,66,89,58,45,32,66,46,72,69,65
54 DATA 84,72,32,86,75,52,65,66,72
55 CLS:PRINT:PRINT
57 FORI=1TO27
59 READA
61 PRINT CHR$(A);
63 NEXTI:END
65 LETY=X/2:LETX=INT(Y)
67 Q=(2*SPEED*(1+(Y-X)*4))
70 SOUND25,Q
75 IFX=1THEN80
77 FORZ=1TOINT(40*SPEED):NEXTZ
78 GOTO65
80 FORZ=1TOINT(120*SPEED+(DELAY*3)):NEXTZ
85 RETURN
90 PRINT:PRINT:PRINT"PRESS-'BREAK'-TO EXIT"
95 FORI=1TO10000:NEXTI:GOTO16
```

YC Jan 86 p150-151 1 of 2.

TRS-80 MC10

```
        1 CLS
        2 PRINT"AUTO-RUN":PRINT"YES(1)":PRINT"NO-(2)"
        4 CLEAR 500
        5 DATA 63,62,60,56,48,32,33,35,39,47,6,17,21,9,2,20,11,16,4,30,13,18,7,5,15,22,2
        7,10,8,3,12,24,14,25,29,19
        6 INPUT R
        7 DIM B$(36)
8 FOR I=1 TO 36:READ J:LET B$(I)=CHR$(J)
9 NEXT I:CLS
10 INPUT"SPEED(WPM)(MAX 15)?";SPEED
11 LET SPEED=7.5/SPEED
12 INPUT"DELAY(0 15)?";DELAY:DELAY=DELAY*50
13 INPUT"NO:-CHARACTERS(MAX 200)";N
14 INPUT"LETTERS(1)NUMBERS(2)OR BOTH(3)?";L
15 DIM T$(N)
16 CLS:PRINT TAB(5)"MORSE TUTOR PROGRAM":FOR I=1 TO N
17 LET T$(I)=CHR$(RND(10*-1*(L<>1)+26*-1*(L<>2))+10*-1*(L=1)):NEXT I
18 FOR I=1 TO N
19 LET X=ASC(B$(ASC(T$(I))))
20 GOSUB 65
23 IF I=INT(I/5)*5 THEN 29
25 IF I=N THEN 32
27 NEXT I
29 FOR Z=1 TO INT(200*SPEED+(DELAY*5)):NEXT Z
30 GOTO 25
32 FOR I=1 TO N
34 IF ASC(T$(I))>10 THEN 39
37 PRINT CHR$(ASC(T$(I))+47);
38 GOTO 40
39 PRINT CHR$(ASC(T$(I))+54);
40 IF I=INT(I/25)*25 THEN 46
41 IF I=INT(I/5)*5 THEN 44
42 IF I=N THEN 49
43 NEXT I
44 PRINT" ";
45 GOTO 42
46 PRINT
47 GOTO 42
49 IF R<>2 THEN 90
50 PRINT:PRINT:PRINT
51 PRINT"PRESS KEY(1)(ENTER)TO RE-TRY":PRINT "PRESS KEY(2)(ENTER)TO EXIT"
52 INPUT P:IF P<>2 THEN 16
53 DATA 80,82,79,71,82,65,77,32,66,89,58,45,32,66,46,72,69,65,84,72,32,86,75,52,
65,66,72
54 CLS:PRINT:PRINT
57 FOR I=1 TO 27
59 READ A
61 PRINT CHR$(A);
63 NEXT I:END
65 LET Y=X/2:LET X=INT(Y)
67 Q=(2*SPEED*(1+(Y-X)*4))
70 SOUND 200,Q
75 IF X=1 THEN 80
77 FOR Z=1 TO INT(40*SPEED):NEXT Z
78 GOTO 65
80 FOR Z=1 TO INT(120*SPEED+(DELAY*3)):NEXT Z
85 RETURN
90 PRINT:PRINT:PRINT"PRESS-'BREAK'-TO EXIT"
95 FOR I=1 TO 10000:NEXT I:GOTO 16
```

YC Jan 86 p. 150-152 2 of 2.

ELECTRIC TUNNEL

The object of the game is to travel along the tunnel, avoiding the electrically charged walls.

The program uses joysticks for control, but by modifying lines 170 and 180 the program could use the keyboard:

170 KY$=INKEY$

180 IF KY$="M" THEN Z=Z-1 ELSE IF KY$="," THEN=Z Z+1

The PEEK in line 190 checks to see if the position in front of you is clear. Scoring is based on the distance you travel along the tunnel.

Bruce Daniel,
Mudgee, NSW

```
0 '  ELECTRIC TUNNEL
1 ' WRITTEN BY    BRUCE DANIEL
2 '
10 CLS : COLOR 2,0.
20 P$ = CHR$(143)
30 FOR I=1 TO 10 : P$=P$+CHR$(176)
40 NEXT I:P$=P$+CHR$(143)
50 IF INKEY$<>"" THEN X=RND(0) :GOTO 50
100 PP=16-INT(LEN(P$)/2)
110 Z=16
130 PRINT TAB(PP);P$ :POKE 28672+Z,99
140 IF RND(2)=1 THEN PP=PP+RND(3)-2
150 IF PP<3 THENPP=3ELSE IFPP>(32-LEN(P$)-3)THENPP=32-LEN(P$)-3
160 IF CN<16 THEN 290
170 JK= INP(43) AND INP(46) AND 31
180 IF JK=27 THEN Z=Z-1ELSE IF JK=23 THEN Z=Z+1
190 L=PEEK(28704+Z):IF L<>144 AND L<>176 AND L<>128 THEN 400
290 CN=CN+1:IF CN/30<>INT(CN/30) THEN 130
300 Q=LEN(P$)
310 IF Q<=5 THEN 130
320 P$=LEFT$(P$,1)+MID$(P$,2,Q-3)+RIGHT$(P$,1)
330 GOTO 130
400 PRINT:POKE 28672+Z,45
410 COLOR,1:SOUND31,1:SOUND31,1:SOUND23,1:SOUND23,1
420 SOUND13,1:SOUND13,1:SOUND4,5
425 '
440 SOUND 0,2
450 COLOR,0


460 FORI=1TO5
470 FORTD=1TO25:NEXTTD
480 PRINT@0,"-+* CRASH CRASH CRASH CRASH *+- ";
490 FORTD=1TO25:NEXTTD
500 PRINT@0,"                                 ";
510 FORTD=1TO25:NEXTTD,I
520 PRINT@128,"SCORE:";'INVERSE 'SCORE'
530 SC=INT(CN*1.2-DN):PRINTSC;
540 PRINT@480,"  PRESS <RETURN> TO TRY AGAIN";
550 IF INKEY$<>CHR$(13) THEN 550
560 RUN
```

Your Computer Jul 85
p 81.

VZ 200

NUMBER SLIDE

Number slide is a computer version of the puzzles that used to be given away with breakfast cereal. This version has been adapted from a ZX81 program printed in this magazine a few years ago.

The idea is to rearrange the numbers in correct order after the computer has mixed them up. The program should work on other computers without much modification.

Bruce Daniel
Mudgee NSW

```
10 DIMA(9):BS$=CHR$(8)+CHR$(8)+CHR$(127)+CHR$(127)
12 Z$="
15 CLS
20 FORX=1TO9
30 LETA(X)=0
40 NEXTX
50 LETA(5)=-32
60 FORX=1TO9
70 IFX=5THENGOTO130
80 LETP=RND(8)
90 FORY=1TO9
100 IFA(Y)=PTHENGOTO80
110 NEXTY
120 LETA(X)=P
130 NEXTX
140 PRINT@224,Z$;Z$;:PRINT@0,;
200 FORX=1TO3
210 FORY=1TO3
220 PRINTCHR$(A(Y+(X-1)*3)+64);" ";
240 NEXTY
250 IFX=1THENPRINT"   123"
260 IFX=2THENPRINT"   456"
270 IFX=3THENPRINT"   789"
280 PRINT
290 NEXTX
300 PRINT
310 PRINT@256,"MOVE FROM:   ";BS$;
320 INPUTF
340 IFF>9THENGOTO310
350 PRINT
360 PRINT@320,"MOVE TO:   ";BS$;
370 INPUTT
380 PRINT:IF(F=3ANDT=4)OR(F=4ANDT=3)ORT=0THEN310
390 IFT>9OR(F=6ANDT=7)OR(F=7ANDT=6)THENGOTO360
400 IFNOTA(T)=-32THENGOTO360
410 IFABS(F-T)=1ORABS(F-T)=3THENGOTO430
420 GOTO310  `
430 LETA(T)=A(F)
440 LETA(F)=-32
450 CLS
470 FORI7=1TO7
480 IFA(I7)>A(I7+1)THENGOTO200
490 NEXTI7
500 PRINT"CONGRATULATIONS, "
510 PRINTTAB(5)"YOU HAVE SOLVED THE PUZZLE."
520 PRINT
530 INPUT"TRY AGAIN (Y/N) ";X$
540 IFX$<>"N"THENRUN
550 CLS:END
```

YC Aug. 85 p. 114.

CUBE

By Maurice McMullan

This program was written for the VZ-200 computer and requires a 16k expansion module. The program is a variation of one written by J. Schultz which was published in *Australian Personal Computer* 1982.

It allows the player to manipulate the Rubik's Cube by using various commands. The commands consist of a series of instructions which rotate the sides of the cube in a clockwise direction through a number of right angles.

Special instructions permit:

1. Set up a random cube (to test the player's ability to solve the cube).
2. Store a cube on cassette.
3. Restore a cube from cassette.
4. To go back to the previous cube if current instructions do not produce the desired effect.
5. If all else fails the program will arrive at a "solved" cube (by cheating of course).

A simple error detection routine determines if a side designator is incorrect and if so the command containing it is ignored.

```
4 CLS: PRINT@233,"***CUBE***";

6 PRINT@291,"WRITTEN BY M.MC.MULLAN":FORA=1TO3000:NEXT

7 CLS:PRINT@229,"INSTRUCTIONS?(YORN)";

8 F$=INKEY$:K$=INKEY$:IFK$=""THEN8

9 IFK$="Y"THENGOTO2000

10 CLS:COLOR,1:CLEAR 420

20 C$(1)="B":C$(2)="F":C$(3)="R":C$(4)="L":C$(5)="D":C$(6)="U"

30 CD(1)=2:CD(2)=6:CD(3)=18:CD(4)=12:CD(5)=4:CD(6)=21
```

V 2(7): Oct. 85
p 47-52.
1 of 6.

```
40 DIMI(9,6)

50 REM SET UP FOR PERFECT CUBE

55 FORA=1TO9:FORB=1TO6:I(A,B)=B:NEXT:NEXT

60 GOSUB400:GOTO720

100 REM SUBROUTINE TO TURN FACE

110 ITEMP=I(8,N):JTEMP=I(7,N)

120 FORINC=6TO1STEP-1:I(INC+2,N)=I(INC,N):NEXT

130 I(2,N)=ITEMP:I(1,N)=JTEMP:RETURN

200 REM SUBROUTINE TO CHANGE AN EDGE

210 FORREP=1TOGO:RESTORE

230 IFCOM>1THENFORDUM=1TO(COM-1)*24:READSKIP:NEXT DUM

240 FORI2NC=1TO3

250 READPO,FA:ITEMP=I(PO,FA)

260 FORINC=1TO3

270 READP2,F2:I(PO,FA)=I(P2,F2)

280 PO=P2:FA=F2:NEXT INC:I(P2,F2)=ITEMP:NEXT I2NC

320 N=COM:GOSUB100:NEXT REP:RETURN

400 REM SUBROUTINE FOR PRINTING CUBE

410 RESTORE

420 RESTORE:FORDUM=1TO144:READSKIP:NEXT
430 FORY=0TO64STEP32:FORX=29098TO29100:READPO,FA

470 POKEX+Y,CD(I(PO,FA)):NEXT:NEXT

485 FORA=0TO256STEP128:FORY=0TO64STEP32

495 FORX=29034TO29036:READPO,FA

505 POKEX-Y-A,CD(I(PO,FA)):NEXT:NEXT:NEXT

515 FORA=0TO8STEP8:FORY=0TO64STEP32:FORX=28966TO28968

530 READPO,FA:POKEX+A+Y,CD(I(PO,FA)):NEXT:NEXT:NEXT:RETURN

600 REMSET UP INSTRUCTION TO GO BACK TO PREVIOUS SET UP

605 X$="/":U=0

610 U=U+2:A$=MID$(Y$,U-1,2)
```

V 2(7) : Oct 85 p 47-52   2 of 6.

```
630 E$=LEFT$(A$,1):D$=MID$(A$,2,1)

650 J=4-(ASC(D$)-48)

660 G$=CHR$(J+48)

662 H$=E$+G$

664 X$=H$+X$

670 IFU+1=LEN(Y$)THENRETURNELSE610

720 PRINT@0,"                                    ";

725 X$="":PRINT@0,"ENTER COMMANDS ";:INPUTX$

727 LL=0

728 LL=LL+1:IFMID$(X$,LL,1)<>"/"THEN728

729 LL=LL-1

730 AA=0

740 AA=AA+2:Z$=MID$(X$,AA-1,2)

745 REM Q = END GAME

750 IFLEFT$(Z$,1)="Q"THENCLS:COLOR,0:END

755 REM Y = PERFECT CUBE

760 IFLEFT$(Z$,1)="Y"THEN50

765 REM I = RETURN TO LAST ATTEMPT

770 IFLEFT$(Z$,1)="I"THENGOSUB600:GOTO727
777 REM X= RANDOM CUBE

778 IFLEFT$(Z$,1)="X"THEN784

779 REM T = STORE CURRENT CUBE ON TAPE

780 IFLEFT$(Z$,1)="T"THEN960

781 REM P = RESTORE CUBE FROM TAPE

782 IFLEFT$(Z$,1)="P"THEN990

783 Y$=X$:GOTO820:REM SAVE CURRRENT CUBE

784 REM SET UP RANDOM CUBE

785 X$="F2":FORJ=1TO9

786 F=RND(5)+1:X$=X$+C$(INT(F))+CHR$(INT(RND(2)+49)):NEXT

787 X$=X$+"/":GOTO727
```

V 2(7): Oct. 85. p 47-52   3 of 6.

```
820 REM DETERMINE WHICH-SIDE AND HOW FAR TO ROTATE
825 G=0:A=0
830 A=A+1:IFMID$(Z$,1,1)=C$(A)THENG=1
840 IFG=0ANDA<6THEN830
850 IFG=1THENCOM=A:GOTO870
860 JP=0:PRINT@0,"ERROR      IN   ";:GOSUB2500:IFJP=1THEN720
865 GOTO727
870 Z$=MID$(Z$,2,1)
890 IFASC(Z$)>=49ANDASC(Z$)<=51THENGO=ASC(Z$)-48:GOTO910
895 IFASC(Z$)=48THEN930ELSEZ$=CHR$(ASC(Z$)-4):GOTO890
910 GOSUB200
920 GOSUB400
930 IFAA<LLTHEN740
940 GOTO720
960 REM STORE CUBE ON TAPE:
961 CLS:PRINT@166,"START TAPE TO RECORD";
962 PRINT@200,"PRESENT SOLUTION";
963 PRINT@259,"PRESS ANY KEY TO CONTINUE"
965 F$=INKEY$:D$=INKEY$:IFD$=""THEN965
970 A$="":FORA=1TO9:FORB=1TO6:A$=A$+CHR$(I(A,B)+48):NEXT:NEXT
975 PRINT#"TEMSOL",A$
980 Z$="Q":GOTO750
990 REM RESTORE CUBE FROM TAPE
991 CLS:PRINT@166,"START TAPE TO INPUT";
992 PRINT@200,"STORED SOLUTION";
993 PRINT@259,"PRESS ANY KEY TO CONTINUE"
994 F$=INKEY$:D$=INKEY$:IFD$=""THEN994
995 INPUT#"TEMSOL",A$
996 L=1:FORA=1TO9:FORB=1TO6:I(A,B)=ASC(MID$(A$,L,1))-48:L=L+1
997 NEXT:NEXT
```

V 2(7): Oct 85 , 47-52   4 of 6.

```
998 CLS:GOSUB400:GOTO720
1000 REMDATA FOR MOVES
1010 DATA3,4,5,6,7,3,5,5,4,4,6,6,8,3,6,5,5,4,7,6,1,3,7,5
1020 DATA7,4,1,5,3,3,1,6,8,4,2,5,4,3,2,6,1,4,3,5,5,3,3,6
1030 DATA7,2,3,5,3,1,7,6,8,2,4,5,4,1,8,6,1,2,5,5,5,1,1,6
1040 DATA7,1,7,5,3,2,3,6,8,1,8,5,4,2,4,6,1,1,1,5,5,2,5,6
1050 DATA1,3,1,2,1,4,1,1,2,3,2,2,2,4,2,1,3,3,3,2,3,4,3,1
1060 DATA5,3,5,1,5,4,5,2,6,3,6,1,6,4,6,2,7,3,7,1,7,4,7,2
1100 REM DATA FOR PRINTING
1110 DATA1,5,2,5,3,5,8,5,9,5,4,5,7,5,6,5,5,5
1120 DATA3,2,2,2,1,2,4,2,9,2,8,2,5,2,6,2,7,2
1130 DATA3,6,2,6,1,6,4,6,9,6,8,6,5,6,6,6,7,6
1140 DATA7,1,6,1,5,1,8,1,9,1,4,1,1,1,2,1,3,1
1150 DATA5,4,6,4,7,4,4,4,9,4,8,4,3,4,2,4,1,4
1160 DATA5,3,6,3,7,3,4,3,9,3,8,3,3,3,2,3,1,3
2000 REM INSTRUCTIONS
2005 CLS:PRINT"******** INSTRUCTIONS *********":PRINT
2010 PRINT"THIS PROGRAM ALLOWS ONE TO PLAY"
2020 PRINT"WITH THE RUBIC CUBE"
2030 PRINT"SIDES ARE LETTERED:-"
2040 PRINT"        B      BACK"
2050 PRINT"        F      FRONT"
2060 PRINT"        U      UPPER"
2070 PRINT"        L      LEFT"
2080 PRINT"        R      RIGHT"
2090 PRINT"        D      DOWN"
2100 PRINT:PRINT"***PRESS <C> TO CONTINUE ***":PRINT
2110 F$=INKEY$:D$=INKEY$:IFD$<>"C"THEN2110
2120 CLS:PRINT"INSTRUCTIONS ARE ENTERED AS :-"
2130 PRINT" 1.ROTATION OF SIDES."
```

V 2(7) : Oct 85.   p 47-52   5 of 6.

```
2140 PRINT"SIDES ARE ROTATED IN A CLOCKWISE"

2150 PRINT"DIRECTION THROUGH A NUMBER OF RIGHT"

2160 PRINT"ANGLES.THE SENSE OF THE ROTATION"

2170 PRINT"OF A FACE IS TAKEN WHEN ONE "

2180 PRINT"LOOKS DIRECTLY AT THAT FACE"

2184 PRINT:PRINT"**** PRESS <C> TO CONTINUE ***":PRINT

2185 F$=INKEY$:D$=INKEY$:IFD$<>"C"THEN2185

2190 CLS:PRINT" AN EXAMPLE OF AN INSTRUCTION IS"

2200 PRINT"         R2L3U1B3/":PRINT

2205 PRINT"  MUST END COMMANDS WITH A /":PRINT

2210 PRINT"THIS MEANS ROTATE:-"

2215 PRINT" RIGHT FACE THROUGH 180DEG"

2220 PRINT" LEFT FACE THROUGH 270DEG"

2225 PRINT" UPPER FACE THROUGH 90DEG"

2230 PRINT" BACK FACE THROUGH 270DEG"

2250 PRINT:PRINT"**** PRESS <C> TO CONTINUE ****":PRINT
2260 F$=INKEY$:D$=INKEY$:IFD$<>"C"THEN2260
2270 CLS:PRINT" 2. SPECIAL INSTRUCTIONS"
2280 PRINT"   Q      QUIT GAME"
2290 PRINT"   Y      SET UP PERFECT CUBE"
2300 PRINT"   I      RETURN TO LAST ATTEMPT"
2310 PRINT"   X      SET UP RANDOM CUBE"
2320 PRINT"   T      STORE CUBE ON TAPE"
2330 PRINT"   P      RESTORE CUBE FROM TAPE"
2333 PRINT:PRINT"****PRESS <C> TO CONTINUE****":PRINT
2334 F$=INKEY$:D$=INKEY$:IFD$<>"C"THEN2334

2340 GOTO10

2500 REM ERROR ROUTINE

2510 FORJ=1TOLLSTEP2

2520 IFMID$(X$,J,2)=Z$THEN2540
2530 NEXTJ:PRINT@0,"NO ERROR FOUND?":JP=1:RETURN

2540 IFJ=1THENJP=1:RETURN

2545 Y$=MID$(X$,1,J-1)+"/"

2550 GOSUB600:RETURN
```

V. 2(7): 47-52 Oct 85   6 of 6.

YAHTZEE

This is a VZ200 version ~~for~~ the dice game Yahtzee, designed for an unlimited number of players.

Each player throws his or her dice up to three times each turn. After the first and second throws you can hold any dice you wish to keep, re-throwing the balance. After the third throw you must enter your score in the table provided.

Once a score has been recorded for a particular category, that category can't be used again. The game ends after 13 rounds.

Because of the limitations of the printer used to produce the listing it's wise to include the graphics |shift Js| in lines 2020 and 2050. The sections underlined should be inserted in inverse text.

The program occupies 5.9 Kbytes of memory.

Ian Thompson,
Collaroy Plateau, NSW

Main Variable Used

N$( )	Player's name
S1$( )	Titles ot category
S2$( )	Description of category
SC$( )	Update score for each category
DF( )	Spots on dice
SC( )	Score
ND( )	Random number for dice
NP	Number of players
IP	User update of scoresheet
TURN	Turn number

Main Routines

0- 46	Title graphics
100	Initialises screen background Clears memory for variables
130	Input players' names
140	Initialises variables
160-170	Input players names
165	Limits player's name to 11 characters
180-190	Set number of player turns per number of players
210	Random number generator for dice throw
290-320	Print score table
330-420	User update of score
470-570	Updates total score
600-650	Displays final score and placings
1000-1120	Data statements for score table
1130-1160	Data statements for spots on dice
2020-2050	Displays dice
9000-20040	User update of score subroutine
21000-22140	Instructions

□

Your Computer Oct. 85
p 105-107
1 of 3.

```
0 '*****************************
1 '*  VZ-200  Y A H T Z E E   *
2 '* IAN THOMPSON - COLLAROY  *
3 '*****************************
4 CLS:SOUND 25,6:COLOR,0
5 FOR X=1 TO 32:POKE 28671+X,204:POKE 29151+
X,195
6 POKE 28672,174:POKE 28703,173:POKE 29152,1
71:POKE 29183,167
7 NEXT X
8 FOR N=28704 TO 29120 STEP 32
9 POKE N,202
10 NEXT N
11 FOR O=28735 TO 29151 STEP 32
12 POKE O,197
13 NEXT O
22 PRINT@106," YAHTZEE "
24 A$=" IAN A.THOMPSON "
26 B$="COLLAROY PLATEAU"
28 FOR N=1 TO LEN(A$)
30 PRINT@231,RIGHT$(A$,N);
32 PRINT@263,RIGHT$(B$,N);
34 NEXT
35 FOR I=1 TO 500:NEXTI
36 PRINT@454,"INSTRUCTIONS (Y/N)?"
38 IF INKEY$<>"" THEN 38
40 A$=INKEY$
42 IF A$="N" THEN SOUND30,1:GOTO100
44 IF A$<>"Y" THEN 40
45 SOUND30,1
46 GOSUB 21000:'INSTRUCTIONS
100 POKE 30744,0:COLOR 5,0:CLEAR 1000
120 R=RND(0)
130 CLS:PRINT@128,"NO. OF PLAYERS";:INPUT NP
135 SOUND 31,1
140 DIM SC$(13,NP),S1$(13),S2$(13),N$(NP),DF
(6,6),SC(NP),YF(NP)
145 GOSUB 1000
150 FOR I=1 TO NP
160 CLS:PRINT@128,"PLAYER #";I;:INPUT"'S NAM
E";N$(I)
162 SOUND 31,1
165 IF LEN(N$(I))>11 THEN SOUND 20,1;10,1:GO
TO 160
170 NEXT
180 FOR TURN = 1 TO 13
190 FOR PL=1 TO NP
210 FOR R=1 TO 5:ND(R)=RND(6):NEXT
220 GOSUB 2000
230 GOSUB 3000
240 PRINT@416,"REMEMBER THESE,THEN"
250 PRINT"HIT ANY KEY TO CONTINUE":A$=INKEY$
260 A$=INKEY$:IF A$="" THEN 260
270 FOR I=1 TO 6:N(I)=0:NEXT
280 FOR I=1 TO 5:N(ND(I))=N(ND(I))+1:NEXT
290 CLS:PRINT"CHOOSE A CATEGORY,";N$(PL)
300 FOR I=1 TO 13:PRINTUSING"##)  ";I;
310 PRINTS1$(I);S2$(I);SC$(I,PL)
320 NEXT
330 PRINT:INPUT"WHICH [1-13]";IP
340 IF IP<1 OR IP>13 THEN 290
345 IF IP=12 THEN 12000
350 IF SC$(IP,PL)<>"" THEN 15000
360 IF IP<7 THEN SC$(IP,PL)=STR$(IP*N(IP))
370 IF IP=7 OR IP=8 THEN 7000
380 IF IP=9 THEN 9000
390 IF IP=10 THEN 10000
400 IF IP=11 THEN 11000
420 IF IP=13 THEN 13000
430 NEXT PL:NEXT TURN
440 FOR PL=1 TO NP:FOR I=1 TO 13
460 NEXT
470 FOR I=1 TO 6
480 SC(PL)=SC(PL)+VAL(SC$(I,PL)):NEXT
490 IF SC(PL)>62 THEN SC(PL)=SC(PL)+35
500 FOR I=7 TO 13
510 SC(PL)=SC(PL)+VAL(SC$(I,PL))
520 NEXT:NEXT
530 FOR I=1 TO NP-1
540 HI=0:FOR J=1 TO NP
550 IF SC(J)>HI THEN HI=SC(J):P=J
560 NEXT
570 D=SC(I):SC(I)=SC(P):SC(P)=D
580 D$=N$(I):N$(I)=N$(P):N$(P)=D$
585 NEXT
590 SOUND 20.1:SOUND 10.1:SOUND 20.1
600 CLS:PRINT"AND THE PLACINGS ARE";
620 PRINT:PRINT:FOR I=1 TO NP
630 PRINTUSING"##";I;
640 PRINT"] ";N$(I);TAB(25);SC(I)
650 NEXT
660 PRINT@480,"ANOTHER GAME (Y/N)?";
670 GOSUB 2000●
680 IF YN$="Y" THEN RUN
690 CLS:PRINT@162,"THANKS FOR THE GAME   BYE
":END
1000 'DATA STATEMENTS
1040 FOR I=1 TO 13
1050 READ S1$(I),S2$(I):NEXT
1060 DATA"ACES",".  [SUM OF 1'S] -","TWOS","
.  [SUM OF 2'S] -"
1070 DATA"THREES",".[SUM OF 3'S] -","FOURS".
". [SUM OF 4'S] -"
1080 DATA"FIVES",". [SUM OF 5'S] -","SIXES",
". [SUM OF 6'S] -"
1090 DATA"3 OF A KIND",".  [SUM] -","4 OF A
KIND",".  [SUM] -"
1100 DATA"FULL HOUSE",".     [25] -","SM. STR
AIGHT",".  [30] -"
1110 DATA"LGE. STRAIGHT",". [40] -","YAHTZEE
",".         [50] -"
1120 DATA"CHANCE",".        [SUM] -"
1130 FOR I=1 TO 6:FOR J=1 TO I
1140 READ DF(I,J):NEXT:NEXT
1150 DATA 66,33,99.1,66,131,33.35,97,99
1160 DATA 1,3,66,129,131,1,3,65,67,129,131
1190 RETURN
2000 CLS:PRINT N$(PL);"'S ROLL"
2010 FOR R=96 TO 224 STEP 32:FOR S=2 TO 26 S
TEP 6
2015 COLOR 5
2020 PRINT@R+S,"   ";'THREE SHIFT J'S
2030 NEXT:NEXT
2040 FOR D=1 TO 5:FOR N=1 TO ND(D)
2045 COLOR 3
2050 PRINT@91+D*6+DF(ND(D).N)," ";'ONE SHIFT
 J
2060 NEXT:NEXT
2070 RETURN
3000 FOR K=1 TO 2:F=1
3010 FOR J=1 TO 5
3020 P=252+J*6:RR(J)=0
3030 PRINT@P,"^^^"
3035 PRINT:PRINT
3040 PRINT"REROLL THIS ONE [Y/N]?"
3050 PRINT"[S FOR SCOREBOARD]":PRINT"[M FOR
MISTAKE]":YN$=INKEY$
3060 YN$=INKEY$
3070 IF YN$<>"Y" AND YN$<>"N" AND YN$<>"S" A
```

Your Computer Oct 85
p 105-107
2 of 3.

```
ND YN$<>"M" THEN3060
3072 IF YN$="Y" THEN SOUND 20,1
3074 IF YN$="N" THEN SOUND 10,1
3076 IF YN$="S" THEN SOUND 15,1
3078 IF YN$="M" THEN SOUND 20,2;10,1
3080 IF YN$<>"S" THEN 3130
3090 CLS:PRINT TAB(5);N$(PL);"'S SCORES"
3100 FORI=1TO13:PRINTUSING"##] ";I;
3105 PRINT S1$(I);S2$(I);SC$(I,PL)
3110 NEXT:PRINT"HIT ANY KEY TO RETURN:":A$=I
NKEY$
3120 A$=INKEY$:IF A$="" THEN 3120 ELSE GOSUB
 2000:GOTO 3020
3130 PRINT@P,"    "
3140 IF YN$="M" THEN 3010
3150 IF YN$="Y" THEN RR(J)=1
3160 NEXT
3170 FOR I=1 TO 5:IF RR(I)=1 THEN ND(I)=RND(
6):F=0
3180 NEXT:IF F THEN K=2
3190 GOSUB 2000
3200 NEXT
3210 RETURN
7000 FOR I=1 TO 6:IFN(I)>IP-5 THEN 7030
7010 NEXT
7020 GOTO 16000
7030 SC=0
7040 FOR I=1 TO 5:SC=SC+ND(I):NEXT
7050 SC$(IP,PL)=STR$(SC)
7060 GOTO 430
9000 FOR I=1 TO 6
9010 IF N(I)>2 THEN N(I)=N(I)-3:GOTO 9040
9020 NEXT
9030 GOTO 16000
9040 FOR I=1 TO 6
9050 IF N(I)>1 THEN 9080
9060 NEXT
9070 GOTO 16000
9080 SC$(9,PL)=" 25"
9090 GOTO 430
9090 GOTO 430
10000 FOR I=1 TO 3:F=1:FOR J=I TO I+3
10010 IF N(J)=0 THEN F=0
10020 NEXT
10030 IF F THEN 10060
10040 NEXT
10050 GOTO 16000
10060 SC$(10,PL)=" 30"
10070 GOTO 430
11000 FOR I=1 TO 2:F=1:FOR J=I TO I+4
11010 IF N(J)=0 THEN F=0
11020 NEXT
11030 IF F THEN 11060
11040 NEXT
11050 GOTO 16000
11060 SC$(11,PL)=" 40"
11070 GOTO 430
12000 FOR I=1 TO 6
12010 IF N(I)=5 THEN 12040
12020 NEXT
12030 GOTO 16000
12040 SC$(12,PL)=" 50"
12050 IF YF(PL) THEN SC$(12,PL)=STR$(VAL(SC$
(12,PL))+100)
12060 YF(PL)=1
12070 GOTO 430
13000 SC=0:FOR I=1 TO 5
13010 SC=SC+ND(I):NEXT
13020 SC$(13,PL)=STR$(SC)
13030 GOTO 430
```

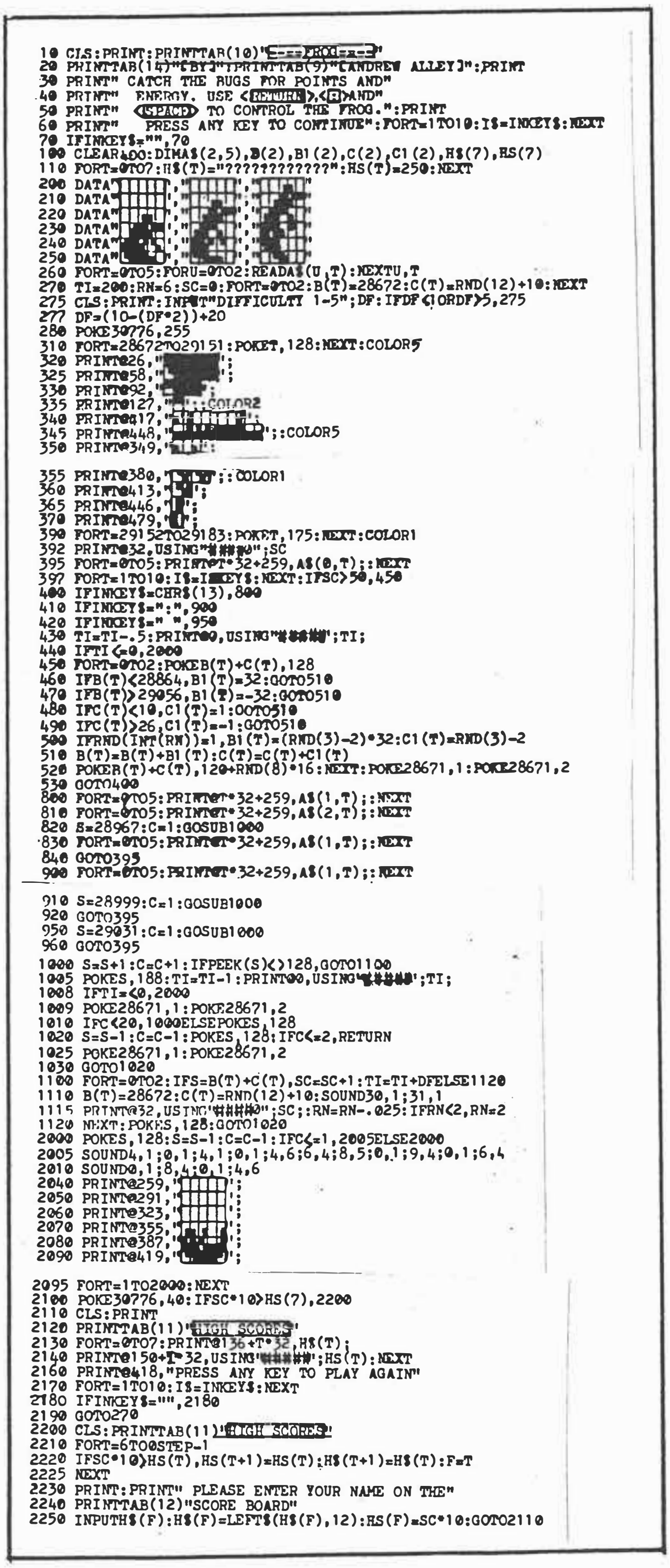
```
15000 SOUND 15,1:CLS
15010 PRINT@128,"YOU'VE ALREADY DONE"
15020 PRINT"THE ";S1$(IP);" ";N$(PL)
15030 FOR I=1 TO 2000:NEXT
15040 GOTO 290
16000 SOUND 15,1:CLS
16010 PRINT@128,"YOU'RE NOT ELIGIBLE FOR"
16020 PRINT"A ";S1$(IP);" ";N$(PL)
16025 IF IP=12 AND YF(PL) THEN SOUND 0,8:GOT
O 290
16030 PRINT:PRINT:PRINT"DO YOU WANT IT ANYWA
Y [Y/N]?";
16040 GOSUB 20000
16050 IF YN$="N" THEN 290
16060 SC$(IP,PL)=" 0"
16070 IF IP=12 THEN YF(PL)=1
16080 GOTO 430
20000 YN$=INKEY$
20010 YN$=INKEY$:IF YN$="" THEN  20010
20020 IF YN$<>"Y" AND YN$<>"N" THEN 20000
20030 IF YN$="Y" THEN SOUND 20,1 ELSE SOUND
10,1
20040 RETURN
21000 CLS:PRINT"INSTRUCTIONS"
21010 PRINT:PRINT"IN THIS DICE GAME EACH PLA
YER"
21020 PRINT"CAN THRUW UP TO THREE TIMES EACH
";
21030 PRINT"TURN. AFTER THE FIRST THROW, HE"
21040 PRINT"CAN SET ASIDE ANY DICE HE WISHES
";
21050 PRINT"TO KEEP,AND RETHROW THE BALANCE.
";
21060 PRINT"HE CAN DO THE SAME AFTER THE"
21070 PRINT"SECOND AND THIRD THROWS. HE CAN,
";
21080 PRINT"OF COURSE, STOP BEFORE THE THIRD
";
21090 PRINT"THROW IF HE WISHES."
21100 PRINT"ONCE THE PLAYER HAS DECIDED TO"
21110 PRINT"STOP, HE MUST DECIDE INTO WHICH"
21120 PRINT"CATEGORY TO ENTER HIS SCORE."
21130 GOSUB 22100
21140 DIM S1$(13),S2$(13)
21150 FOR I=1 TO 13
21152 READ S1$(I),S2$(I):NEXT
21154 CLS
21156 PRINT
21160 FOR I=1 TO 13
21170 PRINT S1$(I):S2$(I):NEXT
21172 PRINT@300,"[SUM OF HOUSE] -"
21174 PRINT@334,"[1,2,3,4,5,] -"
21176 PRINT@366,"[2,3,4,5,6,] -"
21178 PRINT@394,"[FIVE OF A KIND] -"
21180 PRINT@427,"[ANY FIVE DICE] -"
21200 GOSUB 22100
21210 CLS:PRINT"THE GAME ENDS AFTER 12 ROUND
S."
21220 PRINT"ONCE A SCORE HAS PEEN RECORDED"
21230 PRINT"FOR A PARTICULAR CATEGORY, THAT"
21240 PRINT"CATEGORY CAN'T BE USED AGAIN."
21250 GOSUB 22100
21260 RETURN
22100 PRINT@485,"PRESS <C> TO CONTINUE";
22110 IF INKEY$<>"C" THEN 22100
22120 IF INKEY$="" THEN 22100
22130 IF INKEY$="C" THEN 22130
22140 SOUND 30,1:RETURN
```

Your Computer Oct 85
p 105-107
3 of 3.

VZ Frog
by A Alley

Frog begins with a brief instruction screen and asks for the difficulty level (1 to 5). The program then draws a scene of the swamp with the full moon, several water plants and a large frog. Unfortunately, this frog is suffering from a permanent energy crisis. You, as the player, must try to keep him alive by making him eat as many of the insects flying around as possible. This requires a good deal of energy, and so too many misses will result in the frog's untimely demise. The insects get smarter as the game proceeds, and tend to duck out of the way just before the frog eats them.

APC Mar 86 7(3)
p 208-209.



```
10 CLS:PRINT:PRINTTAB(10)"[---FROG---]"
20 PRINTTAB(14)"[BY]":PRINTTAB(9)"[ANDREW ALLEY]":PRINT
30 PRINT" CATCH THE BUGS FOR POINTS AND"
40 PRINT"  ENERGY. USE <RETURN>,<:>AND"
50 PRINT"  <SPACE> TO CONTROL THE FROG.":PRINT
60 PRINT"   PRESS ANY KEY TO CONTINUE":FORT=1TO10:I$=INKEY$:NEXT
70 IFINKEY$="",70
100 CLEAR400:DIMA$(2,5),B(2),B1(2),C(2),C1(2),H$(7),HS(7)
110 FORT=0TO7:H$(T)="????????????":HS(T)=250:NEXT
200 DATA"","",""
210 DATA"","",""
220 DATA"","",""
230 DATA"","",""
240 DATA"","",""
250 DATA"","",""
260 FORT=0TO5:FORU=0TO2:READA$(U,T):NEXTU,T
270 TI=200:RN=6:SC=0:FORT=0TO2:B(T)=28672:C(T)=RND(12)+10:NEXT
275 CLS:PRINT:INPUT"DIFFICULTY 1-5";DF:IFDF<1ORDF>5,275
277 DF=(10-(DF*2))+20
280 POKE30776,255
310 FORT=28672TO29151:POKET,128:NEXT:COLOR5
320 PRINT@26,"";
325 PRINT@58,"";
330 PRINT@92,"";
335 PRINT@127,"";:COLOR2
340 PRINT@417,"";
345 PRINT@448,"";:COLOR5
350 PRINT@349,"";
355 PRINT@380,"";:COLOR1
360 PRINT@413,"";
365 PRINT@446,"";
370 PRINT@479,"";
390 FORT=29152TO29183:POKET,175:NEXT:COLOR1
392 PRINT@32,USING"####0";SC
395 FORT=0TO5:PRINT@T*32+259,A$(0,T);:NEXT
397 FORT=1TO10:I$=INKEY$:NEXT:IFSC>50,450
400 IFINKEY$=CHR$(13),800
410 IFINKEY$=":",900
420 IFINKEY$=" ",950
430 TI=TI-.5:PRINT@0,USING"#####";TI;
440 IFTI<=0,2000
450 FORT=0TO2:POKEB(T)+C(T),128
460 IFB(T)<28864,B1(T)=32:GOTO510
470 IFB(T)>29056,B1(T)=-32:GOTO510
480 IFC(T)<10,C1(T)=1:GOTO510
490 IFC(T)>26,C1(T)=-1:GOTO510
500 IFRND(INT(RN))=1,B1(T)=(RND(3)-2)*32:C1(T)=RND(3)-2
510 B(T)=B(T)+B1(T):C(T)=C(T)+C1(T)
520 POKEB(T)+C(T),120+RND(8)*16:NEXT:POKE28671,1:POKE28671,2
530 GOTO400
800 FORT=0TO5:PRINT@T*32+259,A$(1,T);:NEXT
810 FORT=0TO5:PRINT@T*32+259,A$(2,T);:NEXT
820 S=28967:C=1:GOSUB1000
830 FORT=0TO5:PRINT@T*32+259,A$(1,T);:NEXT
840 GOTO395
900 FORT=0TO5:PRINT@T*32+259,A$(1,T);:NEXT
910 S=28999:C=1:GOSUB1000
920 GOTO395
950 S=29031:C=1:GOSUB1000
960 GOTO395
1000 S=S+1:C=C+1:IFPEEK(S)<>128,GOTO1100
1005 POKES,188:TI=TI-1:PRINT@0,USING"#####";TI;
1008 IFTI=<0,2000
1009 POKE28671,1:POKE28671,2
1010 IFC<20,1000ELSEPOKES,128
1020 S=S-1:C=C-1:POKES,128:IFC<=2,RETURN
1025 POKE28671,1:POKE28671,2
1030 GOTO1020
1100 FORT=0TO2:IFS=B(T)+C(T),SC=SC+1:TI=TI+DFELSE1120
1110 B(T)=28672:C(T)=RND(12)+10:SOUND30,1;31,1
1115 PRINT@32,USING"####0";SC;:RN=RN-.025:IFRN<2,RN=2
1120 NEXT:POKES,128:GOTO1020
2000 POKES,128:S=S-1:C=C-1:IFC<=1,2005ELSE2000
2005 SOUND4,1;0,1;4,1;0,1;4,6;6,4;8,5;0,1;9,4;0,1;6,4
2010 SOUND0,1;8,4;0,1;4,6
2040 PRINT@259,"";
2050 PRINT@291,"";
2060 PRINT@323,"";
2070 PRINT@355,"";
2080 PRINT@387,"";
2090 PRINT@419,"";
2095 FORT=1TO2000:NEXT
2100 POKE30776,40:IFSC*10>HS(7),2200
2110 CLS:PRINT
2120 PRINTTAB(11)"HIGH SCORES"
2130 FORT=0TO7:PRINT@136+T*32,H$(T);
2140 PRINT@150+T*32,USING"#####";HS(T):NEXT
2160 PRINT@418,"PRESS ANY KEY TO PLAY AGAIN"
2170 FORT=1TO10:I$=INKEY$:NEXT
2180 IFINKEY$="",2180
2190 GOTO270
2200 CLS:PRINTTAB(11)"HIGH SCORES"
2210 FORT=6TO0STEP-1
2220 IFSC*10>HS(T),HS(T+1)=HS(T):H$(T+1)=H$(T):F=T
2225 NEXT
2230 PRINT:PRINT" PLEASE ENTER YOUR NAME ON THE"
2240 PRINTTAB(12)"SCORE BOARD"
2250 INPUTH$(F):H$(F)=LEFT$(H$(F),12):HS(F)=SC*10:GOTO2110
```

Balloon Safari, The Drop & Flatten

**Paul Sheppard,
Christchurch, NZ**

These three programs have been written for the standard VZ200/300 computers, in BASIC. They each have instructions within them. In the third program, SHFT appears in some program lines. This means one should type those letters in quotes in conjunction with the SHIFT key.

```
5 CLS:GOSUB7000:CLS
9  ' / "BALLOON SAFARI" \
10 ' \ BY PAUL SHEPPARD /
11 '   \  VZ-KIWISOFT.  /
13 CLK=26624:BO$=CHR$(95)
14 Z$="TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT"
'31 SHFT-"T"
15 A$="  YJJES " 'SHFT
16 B$=" IJJJJJJ " 'SHFT
17 C$="  WJJJJF " 'SHFT
18 D$="   GYYF "  'SHFT
19 E$="   1YR  "  'SHFT
20 BP=44
22 COLOR6:PRINT@480,Z$;
25 GOSUB5000
40 R=RND(3)
50 IFR=1THENBP=BP-1:IFBP<32THENBP=BP+2
70 IFR=3THENBP=BP+1:IFBP>52THENBP=BP-2
80 GOSUB1000
85 I$=INKEY$:I$=INKEY$
90 IFI$=" "ANDKE=0THENSOUND31,1:KE=1:OE=
BP+130:AM=AM+1:GOSUB5000
100 IFKE=1THENGOSUB2000:IFAM=20ANDKE=0TH
EN10000
110 IFSH=0THENGOSUB3000
120 GOSUB4000
130 GOTO40
1000 COLOR7:PRINT@BP,A$:PRINT@BP+32,B$:P
RINT@BP+64,C$
1010 PRINT@BP+96,D$:PRINT@BP+128,E$:RETU
RN
2000 PRINT@OE," ";:OE=OE+32
2005 IFOE=>510THENOE=0:KE=0:SOUND1,3:PRI
NT@480,Z$;:RETURN
2010 PRINT@OE,BO$;
2020 IFNOT(OE>=BLANDOE<=BL+4)THENRETURN
2030 EX$="TEERYT " 'SHFT
2040 PRINT@OE," ";:COLOR4:PRINT@BL,EX$;:
EX$=""
2050 FORT1=1TO5:FORT=1TO20:POKECLK,1:POK
ECLK,0:NEXTT,T1
2060 OE=0:KE=0:KI=KI+1:GOSUB5000:RETURN
3000 BL=474:SH=1:SH$="YQQWTY " 'SHFT
3010 RETURN
4000 BL=BL-1:COLOR1:PRINT@BL,SH$;'FORP=0
TO5:NEXT
4010 IFBL<449,PRINT@BL,"        ";:SH=0
4020 RETURN
5000 PRINT@0,"GATORS ARROWED"KI" ARROWS
LEFT"20-AM
5010 IFAM>=21THENENDELSERETURN
7000 PRINTTAB(9)"BALLOON SAFARI"
7010 PRINTTAB(9)"--------------"
7020 PRINT
7030 PRINT"  THIS IS A SINGLE PLAYER GAM
E"
7040 PRINT"WHERE YOU, ALONG WITH DR BELL
AMY";
7050 PRINT"TRAVEL ACROSS THE AFRICAN PLA
INS";
7060 PRINT"IN YOUR BALLOON HIGH IN THE S
KY.";
7070 PRINT" IN SEARCH OF KILLER ALLIGATO
RS,";
7080 PRINT"(CALLED 'GATORS' BY THE LOCAL
S)"
7090 PRINT"WITH ONLY 20 ARROWS, YOU ARE
TO"
7100 PRINT"RID THE SWAMPS OF THESE FENES
!!":PRINT
7105 PRINT"TO DROP YOUR ARROW ON THE GAT
OR PRESS THE SPACE KEY.
7110 PRINT@482,"   PRESS ANY KEY TO BEGI
N";
7120 I$=INKEY$:IFINKEY$=""GOTO7120
7130 CLS:PRINT@160," BY THE WAY, THE WIN
DS CAN MAKE"
7140 PRINT"YOUR BALLOON GO IN ANY DIRECT
ION"
7150 PRINTTAB(7)"SO AIM CAREFULLY !!"
7160 PRINT@482,"   PRESS ANY KEY TO BEGI
N";
7170 I$=INKEY$:IFINKEY$=""GOTO7170
7180 RETURN
10000 FORT=1TO500:NEXT:CLS:PRINT"YOUR BO
MB SUPPLY IS EXHAUSTED.
10010 PRINT"YOUR SCORE IS"KI"HITTING"KI/
20*100"%."
10040 PRINT@484,;:INPUT"PRESS RETURN TO
START";Q$
10050 RUN
```

```
10 ' /   "THE  DROP"    \
20 ' \ BY PAUL SHEPPARD /
30 '   \  VZ-KIWISOFT   /
40 TS=28672:BS=29183
50 SC=0:BL=20:BA=15:EB=0:ET=0
60 GOSUB800
70 GOSUB500
100 'MAIN LOOP
110 GOSUB200
120 IFFL=1THENFL=0:GOTO900
130 GOTO100
200 'MOVE BALL
205 IFEB>=20,BL=BL+1:ET=ET+1:EB=SC-20*ET
207 PRINT@45,BL;
210 FORX=2TO30
220 POKETS+X-1+32*3,32
230 POKETS+X+32*3,BA
240 A$=INKEY$:A$=INKEY$
250 IFA$=" "GOTO300
260 NEXTX
270 POKETS+X-1+32*3,32
280 GOTO200
300 'DROP BALL
310 FORY=TS+X+32*4TOTS+X+32*14STEP32
320 DP=PEEK(Y)
330 IFDP>48ANDDP<58GOTO400
340 POKEY-32,32
350 POKEY,BA
360 NEXTY
370 POKEY-32,32:SOUNDRND(5),2
380 BL=BL-1:IFBL<1THENFL=1:RETURN
390 RETURN
400 'CHECK NUMBER
405 POKEY-32,32
410 N=DP-48:SC=SC+N:EB=EB+N:FORT=1TO20
420 POKEY,DP
425 FORF=1TO21-T:NEXTF
430 POKEY,32:POKE26624,1:POKE26624,0
435 FORF=1TO21-T:NEXTF:NEXTT
440 GOSUB700
450 PRINT@58,SC;
460 RETURN
500 'SET UP SCREEN
510 CLS
520 FORI=1TO30
530 PRINT@0+I,CHR$(143);
535 PRINT@64+I,CHR$(143);
540 PRINT@479+I,CHR$(143);
550 NEXTI
555 PRINT@510,CHR$(143);
560 FORI=0TO478STEP32
570 PRINT@I,CHR$(143);
580 PRINT@I+31,CHR$(143);
590 NEXTI
595 POKE29183,239
600 PRINT@33,"BALLS LEFT =";
610 PRINT@51,"SCORE =";
620 FORN=1TO9
630 GOSUB700
640 NEXTN
650 RETURN
700 'PLACE NUMBER
710 X=RND(30)+1:Y=RND(5)+10:PO=TS+X+(Y*3
2)
720 IFPEEK(PO)<>32GOTO710
730 POKEPO,N+48
740 RETURN
800 'INSTRUCTIONS
810 CLS:PRINT:POKE30886,31
820 PRINT"             THE DROP
830 PRINT:PRINT:PRINT" DROP THE BALL ON
THE NUMBERS
835 PRINT:PRINT"EXTRA BALL FOR EVERY 20
POINTS
840 PRINT:PRINT" PRESS SPACE TO DROP THE
 BALL
850 PRINT:PRINT:PRINT"    PRESS ANYKEY T
O START
860 A$=INKEY$:A$=INKEY$:IFA$=""THEN860
870 RETURN
900 'END
905 IFSC>HS,HS=SC
910 CLS:POKE30886,31
920 PRINT@139,"GAME OVER
930 PRINT@195,"SCORE ="SC"  HIGH SCORE =
"HS
940 PRINT:PRINT"    DO YOU WANT ANOTHER
GO ?"
950 A$=INKEY$:A$=INKEY$:IFA$=""THEN950
960 IFA$="Y"THEN10ELSEEND
```

```
10 REM /   "FLATTEN"   \
20 REM \ BY P.SHEPPARD /
30 REM  \ VZ-KIWISOFT /
40 TS=28672:BS=29183:PS=TS+32+1:VR=30912
+ASC("I")
50 SS=191:GL=29151:L=32
60 GOSUB800
100 IFPEEK(30744)THENSP=32:BM=31ELSESP=9
6:BM=95
110 CLS:PRINT"DIFFICULTY LEVEL (1-6)":SO
UND31,1:POKE30873,1
120 DI=PEEK(30873):IFDI=1THEN120ELSED=DI
-48:IFD<1ORD>6GOTO110
130 IFD<>6GOTO150
140 PRINT"ARE YOU THAT GOOD (Y/N)";:SOUN
D31,1:POKE30873,1
145 A=PEEK(30873):IFA=1GOTO145ELSEIFA=89
POKEVR,2ELSEPOKEVR,4
150 CLS
160 GOSUB500
200 REM MAIN LOOP
210 GOSUB300
220 GOTO200
300 REM MOVE PLANE
305 POKE26624,1:POKE26624,0
310 PS=PS+1:IFPS>BSGOTO600
320 P=PEEK(PS)
330 IFP=SSGOTO600
340 A$=INKEY$:A$=INKEY$
350 IFA$=" "ANDFL=0THENFL=1:BO=PS:TU=0
360 IFFL=1GOSUB400
370 POKEPS-2,SP
380 POKEPS-1,155:POKEPS,159
385 IFPS+1>GLGOTO700
390 RETURN
400 REM DROP BOMB
405 SB=PEEK(BO+32)
410 POKEBO,SP:REM PATH
430 BO=BO+32:IFBO>GL+32THENFL=0:RETURN
440 POKEBO,BM:REM BOMB
450 IFSB=70,POKEPS-1,SP:POKEPS-2,SP:PS=P
S-(INT((PS-TS)/L)*L)+L
460 RETURN
500 REM SET UP CITY
505 PRINT@7,"DIFFICULTY LEVEL"D;:IFD=6AN
DA=89THENPRINT"*"
510 FORI=1TO30
520 FORY=0TORND(2+D)*32STEP32
530 POKEGL+I-Y,SS
550 NEXTY:I=I+RND((7-D)/2)
560 NEXTI
570 FORI=1TO25
580 POKEGL+I,70
590 I=I+RND(6)
595 NEXTI:RETURN
600 REM GAME END
610 FORI=PS-1TOGLSTEP32
620 POKEI-1,180:POKEI,176
625 FORT=1TO100:NEXTT
630 POKEI-1,SP:POKEI,SP
640 NEXTI
650 FORT=1TO500:NEXT
660 CLS:PRINT@202,"YOU CRASHED
665 SOUND1,6;2,1;3,3;1,4;6,6
670 GOTO730
700 REM LANDED SAFELY
710 CLS
720 PRINT@192,"YOU LANDED SAFE, CONGRATU
LATIONS
730 PRINT@266,"ANOTHER GO?
735 SOUND31,1
740 A$=INKEY$:A$=INKEY$:IFA$=""GOTO740
750 IFA$="Y"RUN
760 SOUND1,5:END
800 REM INSTRUCTIONS
810 CLS:PRINT:PRINTTAB(12)"FLATTEN"
820 PRINT:PRINT:PRINT"YOU MUST LAND YOUR
 PLANE SAFELY
830 PRINT:PRINT" BY DESTROYING THE CITY
BELOW.
840 PRINT:PRINT"  DROP BOMBS BY PRESSING
 SPACE
850 PRINT:PRINT"GAIN EXTRA HEIGHT BY DES
TROYING
860 PRINTTAB(9)"FUEL DUMPS ("CHR$(198)")
870 PRINT:PRINTTAB(6)"PRESS ANYKEY TO ST
ART
875 SOUND31,1
880 A$=INKEY$:A$=INKEY$:IFA$=""GOTO880
890 RETURN
```

VZ-200

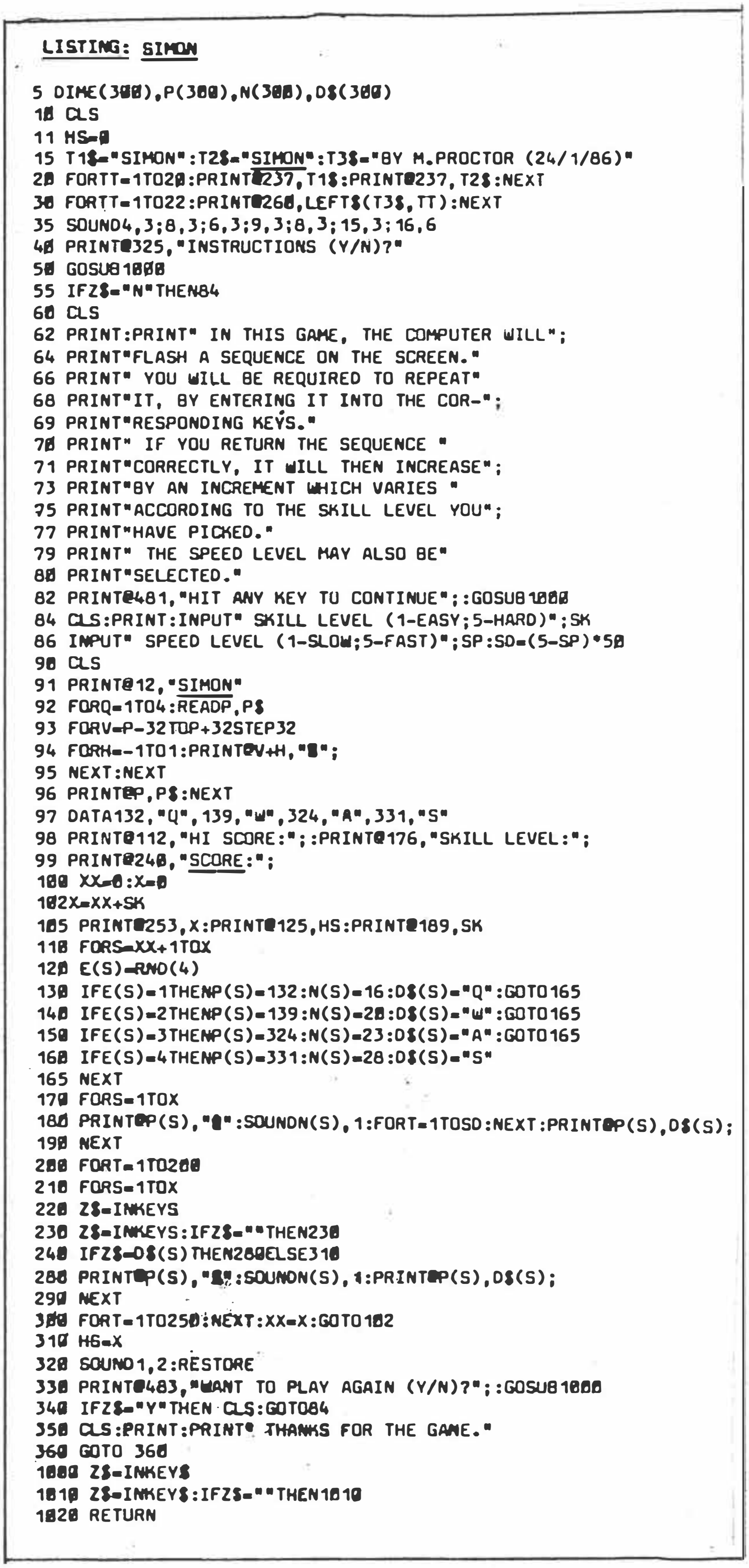
SIMON

This program was inspired by the commercial toy of the same name, and involves repeating a sequence of ever-increasing difficulty. Full operating instructions are presented in the program.

Although written on and for a VZ-200, the BASIC is simple and fairly universal, so conversion to other machines will present no difficulty. The program's simplicity also makes it highly flexible, providing room for improvement and experimentation, which is encouraged.

Michael Proctor,
Killara, NSW.

YC Jul. 86.
p 75.



LISTING: SIMON

```
5 DIME(300),P(300),N(300),D$(300)
10 CLS
11 HS=0
15 T1$="SIMON":T2$="SIMON":T3$="BY M.PROCTOR (24/1/86)"
20 FORTT=1TO20:PRINT@237,T1$:PRINT@237,T2$:NEXT
30 FORTT=1TO22:PRINT@260,LEFT$(T3$,TT):NEXT
35 SOUND4,3;8,3;6,3;9,3;8,3;15,3;16,6
40 PRINT@325,"INSTRUCTIONS (Y/N)?"
50 GOSUB1000
55 IFZ$="N"THEN84
60 CLS
62 PRINT:PRINT" IN THIS GAME, THE COMPUTER WILL";
64 PRINT"FLASH A SEQUENCE ON THE SCREEN."
66 PRINT" YOU WILL BE REQUIRED TO REPEAT"
68 PRINT"IT, BY ENTERING IT INTO THE COR-";
69 PRINT"RESPONDING KEYS."
70 PRINT" IF YOU RETURN THE SEQUENCE "
71 PRINT"CORRECTLY, IT WILL THEN INCREASE";
73 PRINT"BY AN INCREMENT WHICH VARIES "
75 PRINT"ACCORDING TO THE SKILL LEVEL YOU";
77 PRINT"HAVE PICKED."
79 PRINT" THE SPEED LEVEL MAY ALSO BE"
80 PRINT"SELECTED."
82 PRINT@481,"HIT ANY KEY TU CONTINUE";:GOSUB1000
84 CLS:PRINT:INPUT" SKILL LEVEL (1-EASY;5-HARD)";SK
86 INPUT" SPEED LEVEL (1-SLOW;5-FAST)";SP:SD=(5-SP)*50
90 CLS
91 PRINT@12,"SIMON"
92 FORQ=1TO4:READP,P$
93 FORV=P-32TOP+32STEP32
94 FORH=-1TO1:PRINT@V+H,"■";
95 NEXT:NEXT
96 PRINT@P,P$:NEXT
97 DATA132,"Q",139,"W",324,"A",331,"S"
98 PRINT@112,"HI SCORE:";:PRINT@176,"SKILL LEVEL:";
99 PRINT@240,"SCORE:";
100 XX=0:X=0
102X=XX+SK
105 PRINT@253,X:PRINT@125,HS:PRINT@189,SK
110 FORS=XX+1TOX
120 E(S)=RND(4)
130 IFE(S)=1THENP(S)=132:N(S)=16:D$(S)="Q":GOTO165
140 IFE(S)=2THENP(S)=139:N(S)=20:D$(S)="W":GOTO165
150 IFE(S)=3THENP(S)=324:N(S)=23:D$(S)="A":GOTO165
160 IFE(S)=4THENP(S)=331:N(S)=28:D$(S)="S"
165 NEXT
170 FORS=1TOX
180 PRINT@P(S),"■":SOUNDN(S),1:FORT=1TOSD:NEXT:PRINT@P(S),D$(S);
190 NEXT
200 FORT=1TO200
210 FORS=1TOX
220 Z$=INKEYS
230 Z$=INKEYS:IFZ$=""THEN230
240 IFZ$=D$(S)THEN280ELSE310
280 PRINT@P(S),"■":SOUNDN(S),1:PRINT@P(S),D$(S);
290 NEXT
300 FORT=1TO250:NEXT:XX=X:GOTO102
310 HS=X
320 SOUND1,2:RESTORE
330 PRINT@483,"WANT TO PLAY AGAIN (Y/N)?";:GOSUB1000
340 IFZ$="Y"THEN CLS:GOTO84
350 CLS:PRINT:PRINT" THANKS FOR THE GAME."
360 GOTO 360
1000 Z$=INKEY$
1010 Z$=INKEY$:IFZ$=""THEN1010
1020 RETURN
```

DRAWING PROGRAM

This is my version of a hi-res drawing program with a joystick option and printout capability for the VZ200/300.

R. Winter
Morphett Vale SA

```
0 CLS:PRINT:PRINT" PRESS 'K' FOR KEYBOARD CONTROL"
1 PRINT:PRINT"     OR 'J' FOR JOYSTICK "
2 PRINT" **NO COPY AVIALABLE WITH 'J'**"
3 C$=INKEY$:C$=INKEY$
4 IFC$="K"GOTO7
5 IFC$="J"GOTO200
6 IFC$=""GOTO3
7  CLS:PRINT:PRINT"    KEYBOARD INSTRUCTIONS "
8 PRINT:PRINT" USE ARROW KEYS FOR L/R/U/D"
9  PRINT:PRINT" USE 'A' FOR U/L - 'S' FOR U/R"
10       PRINT" USE 'Z' FOR D/L - 'X' FOR D/R"
11 PRINT:PRINT"     'R' TO RUB OUT"
12 PRINT"     'P' FOR PRINTED COPY"
13 PRINT"     'U' TO CLEAR SCREEN"
14 PRINT@453,"- PRESS 'G' TO GO -"
15 S$=INKEY$:S$=INKEY$
16 IF S$<>"G"GOTO15
17 MODE(1)
18 X=35:Y=35
19 SET(X,Y)
20 K$=INKEY$:FORT=1TO80:NEXT:K$=INKEY$
50 IFK$="M"X=X-1:GOSUB182
55 IFK$="A"X=X-1:Y=Y-1:GOSUB182
60 IFK$=","X=X+1:GOSUB182
65 IFK$="S"X=X+1:Y=Y-1:GOSUB182
70 IFK$="."Y=Y-1:GOSUB182
75 IFK$="Z"Y=Y+1:X=X-1:GOSUB182
80 IFK$=" "Y=Y+1:GOSUB182
85 IFK$="X"Y=Y+1:X=X+1:GOSUB182
100 IFK$="R"RESET(X,Y)
110 IFK$="P"COPY
115 IFK$="U"CLS:FORH=1TO200:NEXT:GOTO17
120 GOTO20
182 IFX<0X=0
184 IFY<0Y=0
186 IFX>127X=127
188 IFY>63Y=63
190 SET(X,Y)
195 RETURN
200 CLS:PRINT:PRINT" USE LH STICK FOR 8 DIRECTIONS"
202 PRINT:PRINT"   PRESS 'C' TO CLEAR SCREEN"
203 PRINT@422,"- PRESS 'G' TO GO -"
204 S$=INKEY$:S$=INKEY$
205 IF S$<>"G"GOTO204
206 MODE(1)
208 X=35:Y=35
210 SET(X,Y)
220 A=(INP(43)AND31)
222 L$=INKEY$:L$=INKEY$
250 IFA=27X=X-1:GOSUB182
255 IFA=26X=X-1:Y=Y-1:GOSUB182
260 IFA=23X=X+1:GOSUB182
265 IFA=22X=X+1:Y=Y-1:GOSUB182
270 IFA=30Y=Y-1:GOSUB182
275 IFA=25Y=Y+1:X=X-1:GOSUB182
280 IFA=29Y=Y+1:GOSUB182
285 IFA=21Y=Y+1:X=X+1:GOSUB182
300 IFA=15RESET(X,Y)
310 IFL$="C"CLS:FORJ=1TO200:NEXT:GOTO206
320 GOTO220
```

YCDB 1988.

TEA-POT SONG

This is a computer variation of my four year old daughter's favourite song

R. Winter
Morphett Vale SA

```
0 CLS:PRINT@136,"-██████████████-":FORT=1TO400:NEXTT
5 GOSUB100:GOSUB108:GOSUB115
6 FORT=1TO500:NEXTT
10 PRINT@43,"I'M A LITTLE TEA-POT"
11 SOUND16,2;18,2;20,2;21,2;23,4;28,4
15 PRINT@35,"                "
16 PRINT@67,"                "
17 PRINT@99,"                "
18 GOSUB102:GOSUB108
19 PRINT@76,"SHORT...  "
21 SOUND25,4
25 PRINT@288,"   ███▄▀▄███ "
26 PRINT@320,"   ███▀▄▀███ "
27 PRINT@352," ██▀▀▀▀▀▀▀▀██"
29 PRINT@85,"AND STOUT."
30 SOUND28,4;23,6
35 PRINT@256," ████████████ "
36 PRINT@288," █ ███▄▀▄███ "
37 PRINT@320," █ ███▀▄▀███ "
39 PRINT@141,"HERE IS MY HANDLE,"
40 SOUND21,4;21,2;21,2;20,4;20,4
41 PRINT@256," █████████████"
42 PRINT@288," █ ██▄▀▄██    "
43 PRINT@320," █ ██▀▄▀██    "
44 PRINT@352," ██▀▀▀▀▀▀▀    "
45 PRINT@224,"      ██     █ "
46 PRINT@192,"     █▄▄▄█   ██"
50 PRINT@174,"HERE IS MY SPOUT."
54 SOUND18,4;18,2;18,2;16,6
55 PRINT@43,"    WHEN I GET ALL    "
56 PRINT@76,"    STEAMED UP       "
58 PRINT@141,"                     "
59 PRINT@172,"                        "
60 SOUND16,2;18,2;20,2;21,2;23,4;28,4
61 PRINT@96,"     \!!/   "
62 FORT=1TO20:NEXTT
63 PRINT@64,"    0 00 0  "
64 FORT=1TO20:NEXTT
65 PRINT@32,"   \ \!!/ / "
66 FORT=1TO20:NEXTT
67 PRINT@ 0," 0 0 00 0 0"
69 PRINT@142,"HEAR ME SHOUT...  "
70 SOUND25,4;28,4;23,6
71 PRINT@0,"                          "
72 PRINT@32,"                                "
75 PRINT@64,"  TIP ME OVER....               "
76 PRINT@96,"                    "
78 PRINT@142,"                       "
79 SOUND28,6;25,2;23,4;21,4
80 PRINT@192,"      █▄▄▄█          "
81 PRINT@224,"       ██        ██ "
82 PRINT@256," ████████████ █   █"
83 PRINT@288," █ ██▄▀▄██ ██      "
86 PRINT@384,"      █████ "
87 PRINT@416,"    ██   █ "
88 PRINT@448,"         ██"
89 PRINT@81,"POUR ME OUT!":SOUND20,4;18,4;16,6
90 PRINT@432," ██▀█":PRINT@464," ██ "
91 FORT=1TO200:NEXTT
92 PRINT@304,"V":SOUND16,1:PRINT@336,"V":SOUND15,1:PRINT@369,"V"
93 SOUND14,1:PRINT@401,"V":SOUND13,1:FORT=1TO20:NEXTT:SOUND12,8
94 FORT=1TO1500:NEXTT
95 PRINT@480," ":PRINT"      HIT 'Y' TO RUN AGAIN";
96 A$=INKEY$:A$=INKEY$:IFA$="Y"GOTO5
97 IFA$=""GOTO96
99 IFA$<>"Y"GOTO96
100 CLS
101 PRINT
102 PRINT"      █▀▀▀▀█      "
103 PRINT"     ██▀▄▄▀██     "
104 PRINT"      █▄▄▄▄█      "
105 PRINT"        ██        "
106 PRINT"    ██████████    "
107 RETURN
108 PRINT"   █  ██▄▀▄██ █   "
109 PRINT"   █  ██▀▄▀██ █   "
110 PRINT" ██   ▀▀▀▀▀▀  ██  "
112 PRINT"      ██████ "
113 PRINT"      █    █ "
114 RETURN
115 PRINT"      █    █ "
116 PRINT"      █    █ "
117 PRINT"      █    █ "
118 PRINT"    ██    ██ "
120 RETURN
```

YCBB 1988.

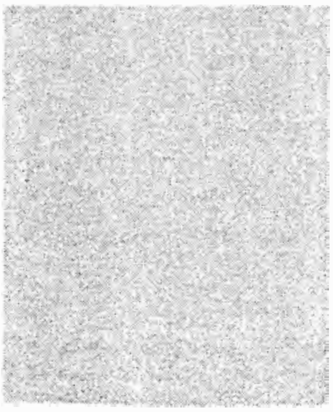
PING TENNIS

A two player game of tennis with no net! You can move as close to your opposition as you like. You can also hit the ball into the walls on the sides. The first person to three sets wins. The first person to 21 points wins a set. Like tennis, you have to win the set by two or more points or the set continues. This game requires joysticks.

Because of my printers' limitations, I could not include graphics symbols in the program print-out so here is a list of them:

line 115
"Shift J(x16)"
line 120
"Shift J(x16)"

R. Duncan
Crafers SA



```
1 POKE30744,1:CLS
99 XX=1:X=1
100 CLS:B=7:C=1:D=7:E=30
105 FORZ=28672TO29152STEP32:POKEZ,175:NEXT
110 FORZ=28703TO29183STEP32:POKEZ,191:NEXT
115 COLOR3:PRINT@0,"                ":PRINT@0,BC:PRINT@6,CC
120 COLOR4:PRINT@16,"                ":PRINT@28,DC:PRINT@22,EC
210 POKE28672+(32*YX+XY),32
215 POKE28672+(32*YY+XX),15
220 YX=YY:XY=XX
225 IFYY>14THENY=RND(2)-2
226 IFYY<2THENYY=1:Y=RND(2)-1
230 IFXX>30THEN500
235 IFXX<1THEN400
240 XX=XX+X:YY=YY+Y
245 A=(INP(43)AND31)
250 IFA=30THENPOKE28704+(32*B+C),32:B=B-1
255 IFA=29THENPOKE28704+(32*B+C),32:B=B+1
256 IFA=27THENPOKE28704+(32*B+C),32:C=C-1
257 IFA=23THENPOKE28704+(32*B+C),32:C=C+1
260 IFB>14THENB=14
262 IFC<1THENC=1
265 IFB<0THENB=0
267 IFC>29THENC=29
270 POKE28704+(32*B+C),175
275 IFABS(B+1-YY)<2ANDC=XXTHENX=+1:Y=RND(3)-2
280 F=(INP(46)AND31)
285 IFF=30THENPOKE28704+(32*D+E),32:D=D-1
290 IFF=29THENPOKE28704+(32*D+E),32:D=D+1
295 IFF=27THENPOKE28704+(32*D+E),32:E=E-1
300 IFF=23THENPOKE28704+(32*D+E),32:E=E+1
305 IFD>14THEND=14
307 IFE<2THENE=2
310 IFD<0THEND=0
312 IFE>30THENE=30
315 POKE28704+(32*D+E),191
320 IFABS(D+1-YY)<2ANDE=XXTHENX=-1:Y=RND(3)-2
371 GOTO210
400 DC=DC+1:IFDC>20ANDDC-BC>1THENDC=0:BC=0:EC=EC+1
405 IFEC>2THENEND
410 XX=1:X=1:GOTO100
500 BC=BC+1:IFBC>20ANDBC-DC>1THENDC=0:BC=0:CC=CC+1
505 IFCC>2THENEND
510 XX=30:X=-1:GOTO100
```

YCBB 1988

CONCENTRATION

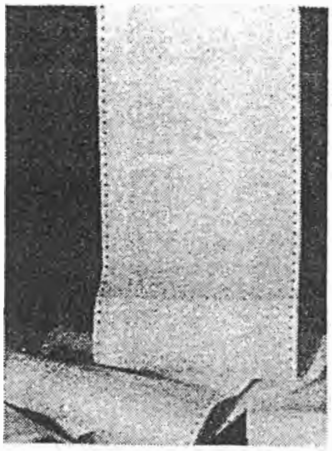
The program called Concentration and is based on the age old game of the same name. Between ten and fifty cards can be selected to appear on the screen. Behind these cards are randomly hidden pairs of cards. The game is finished when all the pairs of cards have been uncovered.

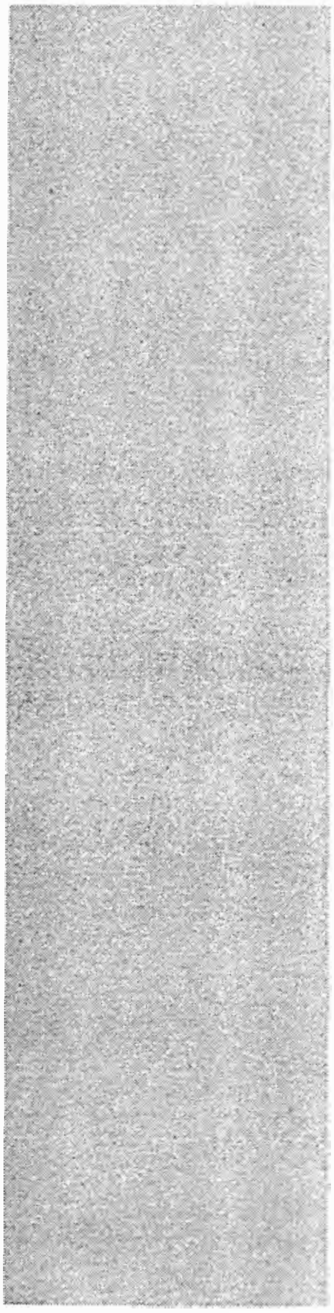
This program is a real test of your concentration.

L. Vella
Leopold Vic.

```
 12 REM  A GAME OF CONCENTRATION
 13 REM  BY L.J. YELLA
 14 REM  1986
 15 POKE30744,1
 16 REM 17-24 PREPARE FOR INTRODUCTION
 17 DATA 28672,28688,28928,28944,175,198,227,245
 18 DIM J(4):DIM H(4)
 19 FOR B=1 TO 4
 20 READ J(B)
 21 NEXT
 22 FOR B=1 TO 4
 23 READ H(B)
 24 NEXT
 25 IF PEEK(28672)=159 GOTO 47
 26 CLS
 27 REM 28-39 DRAWS INTRODUCTION
 28 GOTO 34
 29 POKE AF,AG
 30 AH=AH+1:IF AH=16 THEN AH=0:AF=AF+16
 31 AF=AF+1
 32 AI=AI+1:IFAI=128 GOTO 34
 33 GOTO 29
 34 SOUND 31,1
 35 AJ=AJ+1
 36 IF AJ=5 THEN GOTO 46
 37 AG=H(AJ):AF=J(AJ)
 38 AH=0:AI=0
 39 GOTO 29
 46 GOSUB 10180
 47 CLS:PRINT@100,"DO YOU WANT INSTRUCTIONS"
 48 PRINT@172,"(Y OR N)":FOR B=1 TO 300:NEXT
 49 F$=INKEY$
 50 H$=INKEY$:IF H$="" GOTO 49
 60 IF H$="Y"OR H$="N" GOTO 70
 62 CLS:PRINT@299,"YOU DID NOT PRESS (Y OR N)"
 63 GOSUB 12300
 64 GOTO 47
 70 IF H$="Y"GOSUB 10620
 71 CLS
 72 POKE30744,1
 73 GOSUB 10480
 74 GOSUB 10240
 75 DIM A(99):REM 75-180 NUMBER POSITION
 76 DATA 28706,28709,28712,28715,28718,28721,28724,28727,28730
 77 DATA 28733,28770,28773,28776,28779,28782,28785,28788,28791
 79 DATA 28794,28797,28834,28837,28840,28843,28846,28849,28852
 80 DATA 28855,28858,28861,28898,28901,28904,28907,28910,28913
 100 DATA 28916,28919,28922,28925,28962,28965,28968,28971,28974
 120 DATA 28977,28980,28983,28986,28989
 140 FOR B=1 TO 50
 160 READ A(B)
 180 NEXT B
 190 REM 200-290 GRAPHIC CHARACTERS
 200 DIM C(25)
 220 DATA 208,239,175,191,255,223,143,227,217,188,165,133,172
 240 DATA 243,140,231,179,163,252,131,185,250,136,169,184
 260 FOR B=1 TO 25
 280 READ C(B)
 290 NEXT
 295 REM 300-390 SHUFFLES CARDS
 300 DIM D(AK)
 310 FOR B=1 TO AK
 320 E=RND(AL)
 330 IF C(E)=0 GOTO 320
 340 IF C(E) >300 THEN C(E)=C(E)-200 :I=1
 350 D(B)=C(E)
 360 IF I=1 THEN C(E)=0
 370 IF I=0 THEN C(E)=C(E)+200
 380 I=0
 390 NEXT
 700 DIM H$(4)
 705 REM 710-735 SCOREBOARD IMFORMATION
 710 B=RND(2):IF B=2 THEN X=1
 720 PRINT@353,"                                        "
```

YC88 1988 1 of 4





```
724 IF W=1 THEN X=0
725 PRINT@449,B$;" ";Y
726 IFW<>1 PRINT@464,A$;" ";Z
727 PRINT@425,"< SCOREBOARD >"
728 IF Z+Y=AL GOTO 12100
729 IF X=1 PRINT@353,A$
730 IF X=0 PRINT@353,B$
735 IF W=1 PRINT@464,"ATTEMPTS";   AC
738 REM 739-834 NUMBER SELECTION
739 C=1
740 IFR=4ANDA=1ORR=4ANDAA=1ORR=4ANDG=1ORR=4ANDGG=1:C=3
741 A=0:AA=0:K=0:G=0:GG=0:NP=0
742 IFC=1:H$(1)="":H$(2)="":H$(3)="":H$(4)=""
743 IFC=3:H$(3)="":H$(4)=""
744 IFC=1 PRINT@418,"       "
746 IFC=3 PRINT@422,"  "
747 IFC=1THEN U=418
748 IFC=3 U=422
750 FOR R=C TO 4
755 IF R<=2 PRINT@385,"SELECT YOUR FIRST NUMBER       "
760 IF R>=3 PRINT@385,"SELECT YOUR SECOND NUMBER      "
765 F$=INKEY$
770 G$=INKEY$:IF G$="" THEN 765
775 IF G$="0"ORG$="1"ORG$="2"ORG$="3"ORG$="4"ORG$="5" THEN K=1
780 IFG$="6"ORG$="7"ORG$="8"ORG$="9" THEN K=1
785 IF K=1 THEN 790 ELSE 765
790 PRINT@U,G$
795 SOUND 31,1
800 U=U+1
810 IF U=420 LET U=422
815 H$(R)=G$
820 C$=H$(1)+H$(2)+H$(3)+H$(4)
825 D$=LEFT$(C$,2)
830 E$=RIGHT$(C$,2)
834 S=VAL(D$)
835 REM 836-970 NUMBER PARAMETERS CHECK
836 IF R=2 GOSUB 856:IF AA=1 OR G=1 OR GG=1 OR A=1 GOTO 739
837 IF R=2 GOTO 1020
840 T=VAL(E$)
841 IF R=4 GOSUB 856:IF AA=1 OR G=1 OR GG=1 OR A=1 GOTO 740
842 K=0
850 NEXT
852 PRINT@385,"                                 "
855 GOTO 1020
856 L=A(S):M=A(T)
857 IF R=2 N=PEEK(L):O=PEEK(L-1)
858 P=PEEK(M):Q=PEEK(M-1)
859 IF R=2:IF N<48 OR N>57 THEN LET A=1
860 IF R=4:IF P<48 OR P>57 THEN LET A=1
861 IF S>AK OR T>AK THEN G=1
862 IF R=4 AND S=T THEN AA=1
863 IF S=0 OR R=4 AND T=0 THEN GG=1
866 IF GG=1PRINT@385,"YOUR NUMBER WAS (00)          ":GOSUB12300
870 IFG=1PRINT@385,"THIS NUMBER IS LARGER THAN";AK:GOSUB12300
872 IF AA=1PRINT@385,"THE TWO NUMBERS ARE THE SAME":GOSUB12300
950 IF A=1ANDG=0ANDAA=0ANDGG=0 THEN LETNP=1
960 IFNP=1PRINT@385,"THIS SQUARE HAS BEEN PICKED":GOSUB12300
970 RETURN
1010 REM 1020-1120 DISPLAY SQUARE
1020 IF R=2 POKE A(S),D(S):POKE A(S)-1,D(S):SOUND1,1:GOTO 842
1040 POKE A(T),D(T):POKE A(T)-1,D(T)
1050 IF W=1 THEN AC=AC+1
1060 IF PEEK(L) = PEEK(M) GOSUB 3000 ELSE 1100
1080 GOTO 720
1100 SOUND 10,1
1105 PRINT@385,"THESE TWO ARE NOT MATCHED"
1110 FOR B=1 TO V:NEXT
1115 PRINT@385,"                          "
1120 POKE L,N:POKE L-1,O:POKE M,P:POKE M-1,Q
1130 REM 1140-1160 WHO GOES NEXT
1140 IF X=0 THEN X=1:GOTO 720
1160 IF X=1 THEN X=0:GOTO 720
2999 REM 3000-3160 SCOREBOARD
3000 PRINT@418,"       "
3007 IF X=0 PRINT@352," ";B$;" YOU MATCHED A PAIR"
```

YCBB 1988   2 of 4

```
3008 IF X=1 PRINT@352," ";A$;" YOU MATCHED A PAIR"
3009 SOUND 16,2;18,2;20,2;21,2;23,2;25,2;27,2;28,2
3010 IF W=1 GOTO 3140
3020 IF X=0 THEN Y=Y+1
3040 IF X=1 THEN Z=Z+1
3060 IF Z+Y=AL RETURN
3080 IFX=0 PRINT@385,"YOU CAN HAVE ANOTHER GO          "
3100 IFX=1 PRINT@385,"YOU CAN HAVE ANOTHER GO          "
3110 SOUND 10,2;20,2
3115 FOR B=1 TO V:NEXT
3120 RETURN
3140 Y=Y+1
3160 RETURN
10180 COLOR 8,1
10190 PRINT@99,"A GAME OF"
10200 PRINT@193,"CONCENTRATION"
10220 PRINT@433,"BY  L.J  VELLA"
10225 FOR B=1 TO 2000:NEXT
10230 CLS
10235 RETURN
10240 COLOR 2,0
10260 PRINT"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10280 PRINT@32,"■ 1■ 2■ 3■ 4■ 5■ 6■ 7■ 8■ 9■10■■"
10300 PRINT@64,"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10310 IF AK>=20 PRINT@96,"■11■12■13■14■15■16■17■18■19■20■■"
10320 IF AK>=20 PRINT@128,"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10340 IF AK>=30 PRINT@160,"■21■22■23■24■25■26■27■28■29■30■■"
10360 IF AK>=30 PRINT@192,"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10380 IF AK>=40 PRINT@224,"■31■32■33■34■35■36■37■38■39■40■■"
10400 IF AK>=40 PRINT@256,"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10420 IF AK=50  PRINT@288,"■41■42■43■44■45■46■47■48■49■50■■"
10440 IF AK=50  PRINT@320,"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■"
10445 PRINT@353,"   PLEASE WAIT WHILE I AM       "
10450 PRINT@388," SHUFFLING THE CARDS "
10460 RETURN
10480 PRINT" SELECT THE NUMBER OF PLAYERS                  (1 OR 2)"
10481 FOR B=1 TO 200:NEXT
10482 F$=INKEY$
10483 H$=INKEY$:IF H$="" GOTO 10482
10484 W=VAL(H$)
10485 IF W=1 OR W=2 GOTO 10520
10490 CLS:PRINT@226,"YOU DID NOT PICK A (1) OR (2)"
10495 GOSUB 12300:CLS:GOTO 10480
10520 CLS:FOR B=1 TO 300:NEXT
10521 CLS:PRINT"  ENTER THE NAME OF THE FIRST"
10522 INPUT"  PLAYER AND PRESS RETURN        ";B$
10523 IFLEN(B$)=0GOTO10521
10525 IF LEN(B$)<11 GOTO 10540
10530 CLS:PRINT@100,"YOUR NAME HAD MORE THAN"
10531 PRINT@138,"TEN LETTERS"
10535 GOSUB 12300:CLS:GOTO 10520
10540 IF W=1 THEN X=0:GOTO10580
10550 CLS:PRINT"  ENTER THE NAME OF SECOND"
10555 INPUT"  PLAYER AND PRESS RETURN        ";A$
10560 IFLEN(A$)=0GOTO10550
10562 IF LEN(A$)<11 GOTO 10580
10564 CLS:PRINT@100,"YOUR NAME HAD MORE THAN"
10565 PRINT@138,"TEN LETTERS"
10566 GOSUB 12300:CLS:GOTO10550
10580 CLS:PRINT@99,"WOULD YOU PLEASE SELECT THE"
10582 PRINT@131,"AMMOUNT OF TIME THE SYMBOLS
10584 PRINT@163,"STAY DISPLAYED ON THE SCREEN"
10586 PRINT@233,"1 = 1 SECONDS"
10588 PRINT@265,"2 = 2 SECONDS"
10590 PRINT@297,"3 = 3 SECONDS"
10591 PRINT@329,"4 = 4 SECONDS"
10592 PRINT@361,"5 = 5 SECONDS"
10594 F$=INKEY$
10595 H$=INKEY$:IF H$="" GOTO 10592
10597 V=VAL(H$)/2*1000
10598 IF V>=500 AND V<=2500 THEN GOTO 10599 ELSE 10594
10599 CLS
10600 CLS:PRINT@99,"WOULD YOU PLEASE SELECT HOW"
10601 PRINT@131,"MANY SQUARES YOU WOULD LIKE"
10602 PRINT@163,"TO PLAY AND THEN PRESS":PRINT@195,"RETURN"
```

YCBB 1988 3 of 4.

```
10603 PRINT@227,"NOTE YOU CAN ONLY SELECT"
10604 PRINT@259,"10,20,30,40,OR 50"
10605 PRINT@292," "
10606 INPUT"     SELECT HOW MANY SQUARES";AK
10607 IF AK=10 OR AK=20ORAK=30 OR AK=40 OR AK=50 THEN GOTO 10612
10609 CLS:PRINT@100,"YOU DID NOT SELECT"
10610 PRINT@132,"10,20,30,40,OR 50":GOSUB 12300
10611 GOTO 10600
10612 AL=AK/2
10619 CLS:RETURN
10620 CLS:AB=0
10630 COLOR,0
10635 PRINT"          INSTRUCTIONS"
10640 PRINT"                      "
10660 PRINT"   YOUR GAME OF CONCENTRATION IS"
10680 PRINT@96,"VERY EASY TO PLAY.SIMPLY ATTEMPT"
10700 PRINT@128,"TO MATCH THE PAIRS OF SYMBOLS"
10720 PRINT@160,"HIDDEN BEHIND THE CARDS.IT CAN"
10740 PRINT@192,"BE PLAYED BY ONE OR TWO PERSONS."
10760 PRINT@224,"TWO PERSONS"
10780 PRINT@256,"   INITIALLY THE COMPUTER THROWS"
10800 PRINT@288,"A DICE TO SELECT WHO GOES FIRST."
10810 PRINT@320,"NEXT THE COMPUTER WILL ASK IN"
10820 PRINT@352,"TURN,EACH PLAYER TO PICK THEIR"
10830 PRINT@384,"FIRST AND SECOND NUMBER,WHICH"
10840 GOSUB 12000
10850 PRINT"REPRESENTS TWO CARDS.IF THESE"
10860 PRINT@32,"TWO CARDS HAVE IDENTICAL SYMBOLS"
10870 PRINT@64,"THE CARDS WILL STAY DISPLAYED"
10880 PRINT@96,"AND THE COMPUTER WILL ALLOCATE"
10890 PRINT@128,"A POINT TO THE PERSON WHO SELEC-"
10900 PRINT@160,"-TED THEM,AS WELL AS GIVING THAT"
10910 PRINT@192,"PERSON ANOTHER TURN.IF THE TWO"
10920 PRINT@224,"ARE NOT THE SAME THE CARDS WILL"
10940 PRINT@256,"TURN OVER TO THEIR ORIGINAL NUM-"
10950 PRINT@288,"BER.THE IDEA OF THE GAME IS TO"
10960 PRINT@320,"REMEMBER WHAT SYMBOLS ARE UNDER"
10970 PRINT@352,"EACH CARD, SO AS TO ASSIST IN"
10980 PRINT@384,"SELECTING A MATCHED PAIR OF"
10990 GOSUB12000
11000 PRINT"CARDS LATER ON.THE PLAYER WITH"
11010 PRINT@32,"THE GREATEST NUMBER OF POINTS AT"
11020 PRINT@64,"THE END OF THE GAME WINS."
11030 PRINT@96,"ONE PLAYER"
11040 PRINT@128,"   IF ONLY ONE PLAYER PLAYS,THE"
11050 PRINT@160,"COMPUTER SHOWS HOW MANY ATTEMPTS"
11060 PRINT@192,"WERE MADE TO DISPLAY ALL THE PA-"
11070 PRINT@224,"IRS OF SYMBOLS.NOTE WHEN A NUMB-"
11080 PRINT@256,"ER BETWEEN 1 & 9 IS REQUIRED,"
11090 PRINT@288,"SELECT A ZERO FIRST.FOR EXAMPLE 01,05,09.
11420 PRINT@352,"PRESS (I) TO REPEAT INSTRUCTIONS"
11440 AB=1
12000 PRINT@416,"    PRESS SPACE TO CONTINUE"
12020 F$=INKEY$
12040 H$=INKEY$:IF H$="" GOTO 12020
12060 IF H$=" " GOTO 12080
12070 IF AB=1 AND H$="I" GOTO 10620
12075 GOTO 12020
12080 CLS:RETURN
12100 IF W=2 AND Z>Y PRINT@352,"  ";A$;"  YOU WON       "
12120 IF W=2 AND Y>Z PRINT@352,"  ";B$;"  YOU WON       "
12125 IF W=2 AND Y=Z PRINT@352,"    YOU BOTH WIN IT'S A DRAW   "
12130 IF W=2 PRINT@418,"        "
12140 IF W=1 PRINT@352,"                              "
12155 IF W=1PRINT@418,B$;"                 "
12160 IF W=1PRINT@448,"  YOU FINISHED IN";AC;"ATTEMPTS"
12220 PRINT@384," PRESS SPACE FOR A NEW GAME   "
12230 SOUND 20,1;10,1;20,1;10,1;20,1;10,1
12240 F$=INKEY$
12260 H$=INKEY$:IF H$=""GOTO 12240
12280 IF H$=" "THEN RUN ELSE 12240
12300 SOUND 31,1;29,1;27,1;25,1;23,1;21,1;19,1;17,1;15,1;13,1
12340 SOUND 11,1;9,1;7,1;5,1;3,1;1,1
12360 RETURN
```

YC BB 1988   4 of 4

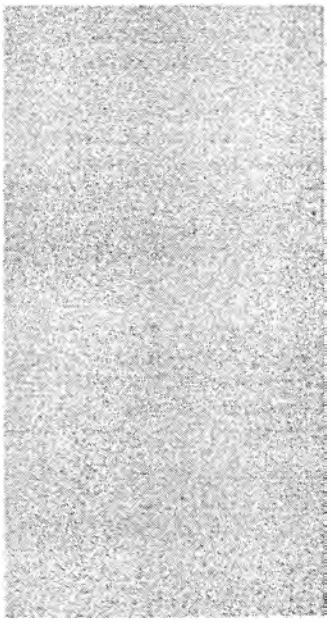
SUPER SNAKE TRAPPER

Super Snake Trapper is a two-player game of skill. You have to move your snake around the screen without hitting the walls, the other snake or yourself. If you do hit something your score goes down. If it reaches zero you lose. If you are about to crash you can press the fire button and you will be put somewhere randomly on the screen, but the computer might land you on something and you will lose points. Joysticks are required to play this game.

Because of my printer's limitations I could not include graphics symbols in the program printout so here is a list of them:

line 15
"Shift A,Shift Y (x21),Shift S"
line 20
"Shift I,Ctrl:,SUPER SNAKE TRAPPER,Ctrl:,Shift"
line 25
"Shift D,Shift T(x21),Shift F"
line 30
"Shift J,BY ROBERT DUNCAN, Shift J"
line 55
"Shift J"
line 60
"Shift J"
line 700
"Shift J(x2)""Shift J(x2)"
line 710
"Shift J(x2)""Shift J(x2)"

R. Duncan
Crafers SA



```
5 POKE 30744,1:CLS
10 PRINT@292,"HIT ANY KEY TO CONTINUE"
15 COLOR RND(6)+2:PRINT@36,"
20 PRINT@68," SUPER SNAKE TRAPPER "
25 PRINT@100,"        2 spaces            "
30 PRINT@167," BY ROBERT DUNCAN "
35 PRINT@203,"(19/12/84)"
40 RRINT@292,"                          ":SOUND 15,1
45 K$=INKEY$:IF INKEY$=""THEN 10
50 PRINT@356,"TYPE IN PLAYER 1'S NAME"
55 COLOR 3:PRINT@423,;:INPUT S$:PRINT@423," "
60 PRINT@371,"2":COLOR 4:PRINT@455,;:INPUT T$:PRINT@455," "
70 PRINT@294,"PRESS <<S>> TO START"
75 SOUND 15,1:SOUND 16,1
80 K$=INKEY$:IF INKEY$<>"S"THEN 75
100 US=37500:VS=37500
105 CLS:MODE(1):COLOR 2:FOR A=0 TO 127:SET(A,0):SET(A,63):NEXT
106 AZ=3750:FOR A=1 TO 62:SET(0,A):SET(127,A):NEXT
110 FOR A=1 TO 62:SET(0,A):SET(127,A):NEXT
115 W=17:X=17:Y=110:Z=17:W1=1:X1=0:Y1=-1:Z1=0
120 AZ=AZ-1:COLOR 3:U=(INP(43)AND 31)
124 IFU=15 THEN X=RND(62):W=RND(126)
125 IF U=30 THEN W1=0:X1=-1
130 IF U=29 THEN W1=0:X1=1
135 IF U=27 THEN W1=-1:X1=0
140 IF U=23 THEN W1=1:X1=0
141 IF U=26 THEN W1=-1:X1=-1
142 IF U=25 THEN W1=-1:X1=1
143 IF U=22 THEN W1=1:X1=-1
144 IF U=21 THEN W1=1:X1=1
145 W=W+W1:X=X+X1
150 IF W=Y AND X=Z THEN 300
155 IF POINT(W,X)=2:US=US-AZ:N$=S$:GOTO 400 ELSE160
160 IF POINT(W,X)=3:US=US-AZ:N$=S$:GOTO 500 ELSE 165
165 IF POINT(W,X)=4:US=US-AZ:N$=S$:GOTO 600 ELSE 170
170 SET(W,X)
200 COLOR 4:V=(INP(46)AND 31)
204 IF V=15 THEN Y=RND(126):Z=RND(62)
205 IF V=30 THEN Y1=0:Z1=-1
210 IF V=29 THEN Y1=0:Z1=1
215 IF V=27 THEN Y1=-1:Z1=0
220 IF V=23 THEN Y1=1:Z1=0
221 IF V=26 THEN Y1=-1:Z1=-1
222 IF V=25 THEN Y1=-1:Z1=1
223 IF V=22 THEN Y1=1:Z1=-1
224 IF V=21 THEN Y1=1:Z1=1
225 Y=Y+Y1:Z=Z+Z1
230 IF W=Y AND X=Z THEN 300
235 IF POINT(Y,Z)=2:VS=VS-AZ:N$=T$:GOTO 400 ELSE 240
240 IF POINT(Y,Z)=4:VS=VS-AZ:N$=T$:GOTO 500 ELSE 245
245 IF POINT(Y,Z)=3:VS=VS-AZ:N$=T$:GOTO 600 ELSE 250
250 SET(Y,Z)
255 GOTO 120
300 MODE(0):CLS:VS=VS-AZ:US=US-AZ
325 PRINT@38,"YOU HAD A COLLISSION"
350 GOTO 700
400 MODE(0):CLS:PRINT@32,N$;",YOU HIT THE WALL":GOTO 700
500 MODE(0):CLS:PRINT@32,N$;",YOU HIT YOUR OWN TAIL":GOTO 700
600 MODE(0):CLS:PRINT@32,N$;",YOU HIT THE OTHER SNAKE":GOTO 700
700 COLOR 3:PRINT@203,"  ";:PRINTUSING"#####,";US;:PRINT"  "
710 COLOR 4:PRINT@267,"  ";:PRINTUSING"#####~~,";VS;:P~~RINT"  "
720 IF VS<1 AND US<1 THEN PRINT@362,"IT IS A DRAW":GOTO 760
730 IF US<1 THEN PRINT@362,;T$;" WON":GOTO 760
740 IF VS<1 THEN PRINT@362,;S$;" WON":GOTO 760
750 FOR A=0 TO 3000:NEXT:GOTO 105
760 PRINT@455,"ANOTHER GAME (Y/N)?"
770 K$=INKEY$:I$=INKEY$:IF I$="Y"THEN RUN ELSE IF I$<>"N"THEN770
```

YCBB 1988.

WORM

The idea of this game is to move from one side of the screen to the other without hitting the dots, the walls, or your own tail. If you do manage to reach the other side, bonus points are awarded before proceeding to a new frame.

I. Thompson
Collaroy Plateau NSW

```
0 '************************
1 '*       W O R M        *
2 '*   FOR UNEXPANDED     *
3 '*     VZ-200/300       *
4 '*  BY IAN THOMPSON     *
5 '************************
8 '
9 GOSUB 1000
10 CLS:POKE 30744,0
30 U$="Q":O$="A":L$="M":R$=","
35 Z=1
40 SC=0
45 X=10:Y=5:Y1=1:X1=0
50 MODE(1):COLOR 3,1:REM - CHANGE TO COLOR 3,0 FOR B&W MONITOR
60 FOR A=1 TO 127:SET(A,0):SET(A,63):NEXT
70 FOR A=0 TO 63:SET(0,A):SET(127,A):NEXT
80 FOR A=111  TO 127:COLOR 4:SET(A,63):NEXT
85 COLOR 3
90 FOR A=1 TO 45:SET(63,A):SET(95,A):SET(79,A+18)
100 SET(47,A+18):SET(24,A):SET(111,A+18):NEXT A
110 A$=INKEY$:IF A$=R$ THEN X1=1:Y1=0
120 IF A$=O$ THEN X1=0:Y1=1
130 IF A$=L$ THEN X1=-1:Y1=0
140 IF A$=U$ THEN X1=0:Y1=-1
170 X=X+X1:Y=Y+Y1:IF POINT(X,Y)=2 THEN 300
180 IF POINT(X,Y)=4 THEN 380
190 IF POINT(X,Y)=3 THEN 350
200 SET(X,Y)
210 SC=SC+1
220 COLOR 2:SET(RND(127),RND(63))
290 GOTO 110
300 SOUND 25,6
302 CLS:PRINT@96,"FRAME NO.";Z:PRINT
303 PRINT"YOU HAVE BEEN DESTROYED"
305 PRINT
310 PRINT"SCORE:";SC
312 PRINT:IF SC>=HSC THEN HSC=SC
315 PRINT"HIGHEST SCORE:";HSC
320 PRINT:INPUT"ANOTHER FRAME (Y/N)";A$:IFA$="Y"THENZ=Z+1:GOTO40
330 IF A$="N" THEN CLS:PRINT @ 237,"BYE!":END
340 GOTO 300
350 SOUND 25,6:CLS:PRINT@32,FRAME NO."Z;
352 PRINT:PRINT
355 PRINT"YOU HIT YOUR OWN TAIL  "
360 PRINT"YOU HAVE BEEN DESTROYED":PRINT:GOTO310
380 CLS:SOUND30,7:PRINT:PRINT"500 POINTS BONUS":SC=SC+500:FORA=1TO 500
390 NEXT A:GOTO 45
1000 CLS:PRINT@75," W O R M "
1010 PRINT@225,"IAN THOMPSON,COLLAROY PLATEAU"
1020 FOR I=1 TO 1000:NEXT I
1030 CLS:PRINT"THE IDEA OF THIS GAME IS TO MOVE";
1035 CLS:PRINT"FROM ONE SIDE OF THE SCREEN TO"
1040 PRINT"THE OTHER,AND INTO THE RED LINE."
1050 PRINT"ONCE YOU HIT THE RED LINE YOU"
1060 PRINT"GET 500 POINTS BONUS AND YOU"
1070 PRINT"START A SECOND FRAME."
1080 PRINT:PRINT"THE DANGERS ARE THE WALLS, THE"
1090 PRINT"DOTS, AND YOUR OWN TAIL."
1100 PRINT@482,"PRESS <RETURN> TO CONTINUE";
1105 INPUT A$
1110 CLS:PRINT@72,"DIRECTION KEYS"
1120 PRINT@139,"Q = UP"
1130 PRINT@203,"A = DOWN"
1140 PRINT@267,"M = LEFT"
1150 PRINT@331,", = RIGHT"
1160 PRINT@482,"PRESS <RETURN> TO START";
1170 INPUT A$
1180 RETURN
```

YC88 1988.

DOGFIGHT

You are in a plane and must endeavour to shoot down another plane. Using the arrow keys, you position the target plane in the dead center of the sights. You shoot with the Z key.

I. Thompson
Collaroy Plateau NSW

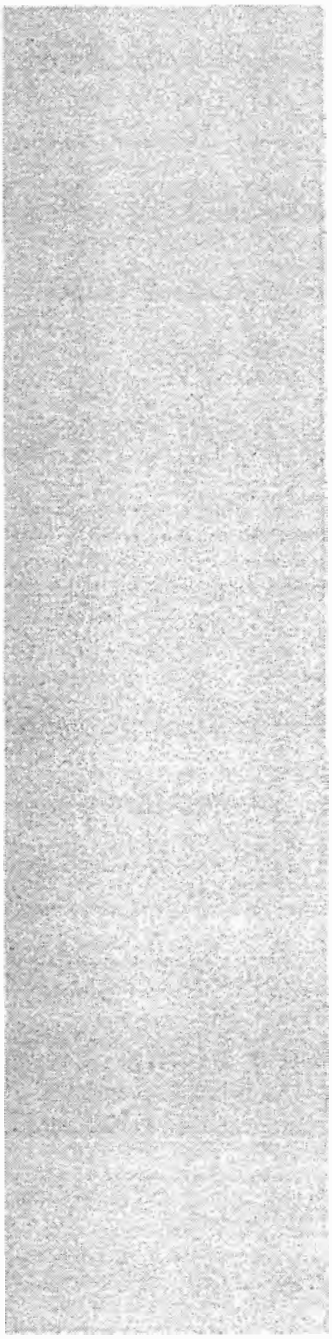
```
0  ***************************
1 '*    D O G   F I G H T     *
2 '*      FOR 8K VZ-200       *
3 '*    BY IAN A.THOMPSON     *
4 '***************************
5 CLS:SOUND 25,6:PRINT@134," D O G   F I G H T "
6 PRINT@225,"IAN THOMFSON,COLLAROY PLATEAU"
7 PRINT@449,"INSTRUCTIONS (Y/N)";
8 INPUT AN$
9 IF LEFT$(AN$,1)="Y" THEN 1500 ELSE 10
10 SC=0:ML=0
20 MODE(1):COLOR 3
30 FOR K=0 TO 127
40 IF K>63 THEN 60
50 IF K>24 AND K<40 THEN SET(50,K):SET(78,K) ELSE SET(64,K)
60 IF K>49 AND K<79 THEN SET(K,25):SET(K,39) ELSE SET(K,32)
70 NEXT:X=RND(123)+1:Y=63:COLOR 4:GOTO 140
110 RESET(A,4):RESET(A+1,4):RESET(A-1,4)
120 RESET(A+2,4):RESET(A-2,4):RESET(A,Y-1)
130 Y=Y-1:IF Y=1 THEN 200
140 SET(X,Y):SET(X+1,Y):SET(X-1,Y)
150 SET(X+2,Y):SET(X-2,Y):SET(X,Y-1)
152 RESET(X,Y):RESET(X+1,Y):RESET(X-1,Y)
154 RESET(X+2,Y):RESET(X-2,Y):RESET(X,Y-1)
160 GOSUB 1000
170 GOTO 110
200 ML=ML+1:IF ML=10 THEN 210 ELSE 20
210 CLS
220 PRINT:PRINT"YOU SHOT DOWN";SC,"OUT OF 10 PLANES"
230 PRINT:PRINT:PRINT"ANOTHER GO";
240 INFUT AN$
250 IF LEFT$(AN$,1)="Y" THEN RUN ELSE 260
260 CLS:PRINT"THANKS FOR THE GAME, BYE!":END
1000 A=X
1010 IF INKEY$="," THEN 1050
1020 IF INKEY$="M" THEN 1070
1030 IF INKEY$="Z" THEN 1100
1040 RETURN
1050 X=X+2:IF X>125 THEN X=125
1060 RETURN
1070 X=X-2:IF X<2 THEN X=2
1080 RETURN
1100 SOUND 15,2
1110 FOR K=34 TO 64 STEP 2
1120 SET(K,96-K):SET(127-K,96-K)
1130 NEXT
1140 FOR K=34 TO 64 STEP 2
1150 RESET(K,96-K):RESET(127-K,96-K)
1160 NEXT
1170 IF X<67 AND X>61 AND Y<34 AND Y>30 THEN 1200
1180 SOUND 1,2
1190 RETURN
1200 SOUND 29,2;31,2
1205 FOR T=1 TO 3
1210 FOR K=1 TO 20
1220 SET(RND(10)+59,RND(10)+27)
1230 NEXT
1240 FOR K=1 TO 30
1250 SET(RND(30)+49,RND(30)+17)
1260 NEXT
1265 NEXT
1270 SC=SC+1:ML=ML+1:IF ML=10 THEN 210
1280 GOTO 20
1500 CLS:PRINT"THE GAME IS CALLED DOG-FIGHT, "
1510 PRINT"AND AS THE NAME SUGGESTS,YOU AREIN A PLANE AND";
1520 FRINT" MUST ENDEAVOUR TOSHOOT DOWN ANOTHER PLANE."
1530 FRINT"SIGHTS APPEAR ON THE SCREEN, ANDYOU MUST MOVE YOUR";
1540 PRINT" PLANE (USING THE LEFT AND RIGHT ARROW KEYS)  TO ";
1550 FRINT"GET THE TARGET PLANE DEAD IN THE CENTRE OF THE ";
1560 FRINT" SIGHTS."
1570 FRINT"YOU SHOOT WITH THE 'Z'KEY."
1580 FRINT"YOU'LL BE GIVEN 10 PLANES TO "
1590 FRINT"SHOOT DOWN, AND AT THE END TOLD"
1600 FRINT"HOW MANY YOU MANAGED TO GET. YOU";
1610 PRINT"WILL THEN BE OFFERED A NEW GAME.";
1620 FRINT" FRESS <S> TO START THE GAME."
1630 IF INKEY$<>"S" THEN 1630
1640 IF INKEY$="S" THEN 10
```

YCBB 1988

BEZERK

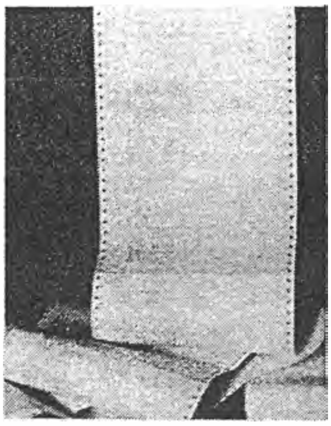
Bezerk is a games program written for the VZ200/300. The idea is that when in playing mode you move a dot around the screen running through the red dots. If you do not reach the red dots in time and you touch them they will turn yellow and you *die!* Do not touch the walls or anything yellow. At the end of the game you will be given a bonus point for every red dot you ran over.

R. Banks & M. Saunders
Mackay Qld



```
0 DATA243,1,100,0,32,20,0,205,92,52,201:CLEAR200
1 FORI=31059TO31068:READA:POKEI,A:NEXT:POKE30862,82:POKE30863,121
10 DATAQ,A,K,L:FORX=1TO4:READA$(X):NEXT
11 DATA13,4,14,4,13,2,11,2,13,2,14,3,11,7,8,2,13,2,11,2,8,2
12 DATA11,1,11,1,13,2,8,2,13,2,11,2,8,2,11,1,11,1,13,2
14 DATA13,2,18,2,16,2,13,2,16,1,16,1,18,2,13,2,18,2,16,2,13,2
15 DATA16,1,16,1,18,2,15,2,20,2,18,2,15,2,18,1,18,2,20,2
17 DATA13,2,18,2,16,2,13,2,16,1,16,1,18,2
18 DATA8,2,13,2,11,2,8,2,11,1,11,1,13,2
50 '
52 DIMSD(56),SF(56):DIMH(100):FORI=1TO56:READSF(I),SD(I):NEXT
60 FORI=1TO10:FORY=1TO5:READA(I,Y):NEXT:NEXT
69 CLS:PRINTTAB(13)"BEZERK":HP=1
70 PRINT@32,"[C]HANGE KEYS OR [S]TART GAME":SP=8:GOTO5000
71 GOSUB2000:IFA$="S"THEN200ELSEIFA$="I"THEN7000
72 IFA$<>"C"THEN71ELSE1000
73 PRINT@96,"UP - "A$(1):PRINT"DOWN - "A$(2):PRINT"LEFT - "A$(3)
74 PRINT"RIGHT - "A$(4):GOTO71
200 X=USR(0):MODE(1):FORX=29TO96:SET(X,5):SET(X,42):NEXT
210 FORY=5TO42:SET(29,Y):SET(28,Y):SET(96,Y):SET(97,Y):NEXT:WL=0
211 GOSUB3900
310 TD=0:X=62:Y=22:IY=0:IX=1:P(0,0)=0:P(0,1)=22:PT=0:T=-1:PH=0
311 DC=0:TN=RND(40):GOTO510
410 XR=RND(16)+7:YR=RND(37)+5:XY=32*YR+XR+28672
420 IFPEEK(XY)>0ORPEEK(XY+1)>0THEN410
430 V=RND(9):T=INT(400/V):TC=0:TN=-1:POKEXY,255
510 A$=INKEY$:IFA$=""THEN520ELSEIFA$=A$(1)THENIY=-1:IX=0:GOTO520
511 IFA$=A$(2)THENIY=1:IX=0:GOTO520
512 IFA$=A$(3)THENIY=0:IX=-1:GOTO520
513 IFA$=A$(4)THENIY=0:IX=1
520 X=X+IX:Y=Y+IY:IFPOINT(X,Y)<>1THEN570
521 POKE31060,30:POKE31063,1:S=USR(0)
530 RESET(PT,PH):SET(X,Y):PT=X:PH=Y
550 TC=TC+1:IFTC=TTHEN960
560 TD=TD+1:IFTD=TNTHEN410ELSE510
570 IFPOINT(X,Y)=4THEN910ELSEG=G+1:WT=WT+WL
580 FORX=1TOHP:WL=WL+1:GOSUB3900:POKEH(X),0:S=USR(0):NEXT
590 FORI=1TO7:SOUNDSF(I),SD(I):NEXT
610 IFX<30ORX>95ORY<6ORY>41THENM$="HIT THE WALL":GOTO620
611 M$="HIT A BLOCK"
620 CLS:PRINT"YOU HAVE "M$:PRINT"* GAME OVER *"
700 PRINT"THIS WAS GAME NUMBER"G
710 PRINT"YOUR SCORE WAS-"WL:PRINT"THE AVERAGE SO FAR-"INT(WT/G)
720 PRINT"THE PREVIOUS BEST WAS"W1:IFWL>W1THENW1=WL
736 POKE30777,25:INPUT"ENTER YOUR NAME";SC$:POKE30744,RND(2)-1
737 SC$=LEFT$(SC$,13):SC$=SC$+
740 CLS:POKE30777,25:GOTO69
910 POKEXY,85:DC=V:TN=RND(40)+V:TD=0:T=-1:XR=1:WL=WL+V
911 POKE31060,40:FORI=1TO2:FORU=1TO20STEP3:POKE31063,U:S=USR(0)
912 NEXT:NEXT:GOSUB3900:H(HP)=XY:HP=HP+1:GOTO0530
960 POKEXY,85:XR=1:TN=RND(40):TD=0:T=-1:GOTO510
1000 PRINT@96,"UP":GOSUB2000:A$(1)=A$:PRINT@96,"UP - "A$(1)
1010 PRINT@128,"DOWN":GOSUB2000:A$(2)=A$:PRINT@128,"DOWN - "A$(2)
1020 PRINT@160,"LEFT":GOSUB2000:A$(3)=A$:PRINT@160,"LEFT - "A$(3)
1030 PRINT@192,"RIGHT":GOSUB2000:A$(4)=A$:PRINT@192,"RIGHT - "A$(4)
1040 GOTO71
2000 SOUNDSF(SP),SD(SP):A$=INKEY$
2010 SP=SP+1:IFSP>56THENSP=9
2011 IFA$=""THENB$="":GOTO2000
2012 IFA$=B$THEN2000ELSEB$=A$:RETURN
2020 GOSUB2000:IFINKEY$="Y"ORINKEY$="N"THEN74ELSENEXT:GOTO74
3900 SD$=STR$(WL):SD$=RIGHT$(SD$,LEN(SD$)-1):B=28688
3901 FORI=LEN(SD$)TO1STEP-1
3902 IFMID$(SD$,I,1)<>MID$(SE$,I,1)THEN3912
3903 B=B-1:NEXT:SE$=SD$:RETURN
3912 C=VAL(MID$(SD$,I,1))+1
3915 FORU=0TO4:POKEB+32*U,A(C,U+1):NEXT:GOTO3903
4000 DATA252,204,204,204,252
4010 DATA48,240,48,48,252
4020 DATA252,12,252,192,252
4030 DATA252,12,60,12,252
4040 DATA192,192,204,252,12
4050 DATA252,192,252,12,252
4060 DATA252,192,252,204,252
4070 DATA252,12,12,12,12
4080 DATA252,204,252,204,252
4090 DATA252,204,252,12,252
5000 PRINT"OR [I] FOR INSTRUCTIONS"
5001 PRINTTAB(10)"────────────────────":FORI=1TO10
```

YC63 1988 1 of 2.



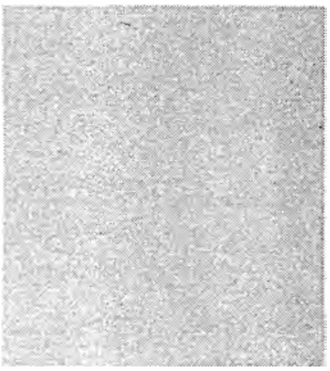
```
5002 PRINTTAB(10)"█                              █":NEXT
5003 PRINTTAB(10)"▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀";
5010 FORI=0TO9:IFSC$(I+1)=""THENI=12:GOTO5030
5020 PRINT@139+32*I,SC$(I+1)SC(I+1)
5030 NEXT:GOTO73
6000 FORI=1TO10:IFWL=>SC(I)THEN6010ELSENEXT:GOTO740
6010 FORU=10TOISTEP-1:SC$(U)=SC$(U-1):SC(U)=SC(U-1):NEXT
6020 SC$(I)=SC$:SC(I)=WL:GOTO740
7000 CLS:PRINT@7,"[illegible]"
7010 PRINT"THE OBJECT OF THIS GAME IS TO  -RUN OVER THE RED";
7020 PRINT" RECTANGLES.    WHEN YOU RUN OVER ONE, IT WILL"
7030 PRINT"TURN YELLOW. THE DOTS WILL ALSO TURN YELLOW IF YOU";
7040 PRINT" DON'T RUN    OVER THEM IN TIME."
7050 PRINT"IF YOU TOUCH ANYTHING YELLOW,   YOU WILL DIE!"
7060 PRINT"EACH RED DOT WILL GIVE YOU      BETWEEN 1 AND 9";
7070 PRINT" POINTS. WHEN YOUHAVE DIED YOU WILL BE GIVEN A
7080 PRINT"BONUS FOR EACH DOT YOU RAN OVER."
7090 PRINT"PRESS [illegible]"
7100 GOSUB2000:IFA$=CHR$(13)THEN69ELSE7100
```

1988.   YCBB p 87.   2 of 2.

ARGGGGH!

This exciting prog:am written for the VZ200/300 re uires a good deal of skill. Weave yourself in and out of the yellow dots, avoiding them and the walls, until a hole appears in the top middle of the screen. You are only allowed to go back on yourself a few times, so beware.

R. Banks & M. Saunders
Mackay QLD



```
1 COLOR2
2 POKE30862,82:POKE30863,121:POKE31059,243:POKE31059,201
3 PD%=50
4 PH%=990
5 PX%=63:PY%=34
6 EM%=5:BT%=50-PD%
10 MODE(1)
20 FORA%=0TO127:SET(A%,5):SET(A%,63):NEXT
30 FORA%=5TO63:SET(0,A%):SET(127,A%):NEXT
40 COLOR3:A$=INKEY$:SC=SC+6-EM%:IFA$=""THEN50
45 PL%=0:PU%=0
50 IFA$="W"THENPU%=-1
60 IFA$="S"THENPU%=1
70 IFA$="K"THENPL%=-1
80 IFA$="L"THENPL%=1
90 PX%=PX%+PL%:PY%=PY%+PU%
95 IFEM%<0THENEM%=0
100 IFPOINT(PX%,PY%)=3THENGOTO1100
105 IFPOINT(PX%,PY%)<>1THENPRINT"YOUR SCORE IS"SC:END
110 SET(PX%,PY%):COLOR2:EX%=RND(126):EY%=RND(57)+5
115 IFPY%<5THENPRINT"YOUR SCORE IS"SC"SO FAR...":PD=PD-5:GOTO4
120 EC%=EC%+1:IFEC%<EM%THEN40
125 EC%=0
126 IFRND(1000)>PH%THENEM%=EM%-1:PH%=PH%-PD%
127 IFEM%=0THENRESET(62,5):RESET(63,5):RESET(64,5)
130 IFPOINT(EX%,EY%)<>1THENRESET(EX%,EY%)ELSESET(EX%,EY%)
140 GOTO40
1000 FORI=1TO1000:NEXT:GOTO4
1100 BT%=BT%+1:IFBT%>50THEN105
1105 SC=SC-BT%
1106 IFSC<0THENSC=0
1110 GOTO110
```

YCBB 1988 p. 87.

ENCODE/ DECODE

Encode/Decode is an encoding and decoding program written for the VZ200/300. When run it will ask you to input a word or secret message. After typing in your secret message, on the line below will appear the message in code form. It will then ask you to input a secret message in jumbled form which it will then decode.

R. Banks & M. Saunders
Mackay Qld

```
10 INPUT"ENTER WORD";A$:PRINTLEFT$(A$,1);:A=ASC(A$)
20 FORI=2TOLEN(A$):B=ASC(MID$(A$,I,1))+(A-64):IFB>90THENB=B-26
60 PRINTCHR$(B);:A=B:NEXT:PRINT:GOTO100
100 INPUT"ENTER WORD";A$:PRINTLEFT$(A$,1);:A=ASC(A$)
110 FORI=2TOLEN(A$):B=ASC(MID$(A$,I,1))-(A-64):IFB<65THENB=B+26
150 PRINTCHR$(B);:A=ASC(MID$(A$,I,1)):NEXT:PRINT:GOTO10
```

YC88 1988.

CATCH

A lot of skill and patience is required to use this program. You must trap the other moving dot into one spot on the screen. To do so use the following commands –

W — up
S — down
K — left
L — right
E, R — to turn yourself visible and invisible.

Please note that the designers of this program takes no responsibility if you hit your computer through frustration and anger!

R. Banks & M. Saunders
Mackay Qld

```
1 X%=USR(0)
2 MODE(1)
3 COLOR3:FORA%=0TO127:SET(A%,0):SET(A%,63):NEXT:FORA%=0TO63
4 SET(0,A%):SET(127,A%):NEXT
5 B%=3:X1%=RND(126):Y1%=RND(62)
1000 X%=RND(126):Y%=RND(62)
1001 A%=RND(4):FORGG%=1TORND(20)
1002 GOTO4000
1005 G%=X%:H%=Y%
1010
1020 X%=X%+1:GOTO1100
1030 X%=X%-1:GOTO1100
1040 Y%=Y%+1:GOTO1100
1050 Y%=Y%-1
1100 IFPOINT(X%,Y%)=3Y%=H%:X%=G%:GOTO1001
1110 COLOR3:SET(X%,Y%):COLOR2:SET(X%,Y%):NEXT:GOTO1001
4000 XS%=X1%:YS%=Y1%:IF(PEEK(26624)OR192)=255THEN4070
4010 K%=PEEK(26751)OR192:IFK%=255THEN4040
4020 IF(K%AND8)=0X1%=X1%-1:GOTO4060
4030 IF(K%AND2)=0X1%=X1%+1:GOTO4060
4040 IF(PEEK(26873)OR192)=255THEN4050ELSEK%=PEEK(26873)OR192
4041 IF(K%AND2)=0Y1%=Y1%-1:GOTO4060
4042 IF(K%AND32)=0B%=1:GOTO4060
4043 IF(K%AND1)=0B%=3:GOTO4060
4050 IF(PEEK(26877)AND2)=0Y1%=Y1%+1
4060 IFX1%<1ORX1%>126ORY1%<1ORY1%>62X1%=XS%:Y1%=YS%
4070 IFB%=1THEN4090
4080 IFPOINT(X1%,Y1%)=2X1%=XS%:Y1%=YS%
4090 COLOR2:SET(X1%,Y1%):COLORB%:SET(X1%,Y1%):GOTO1005
5000 DATA33,0,0,17,0,112,1,0,1,237,176,201
6000 FORA%=0TO16383:IFPEEK(A%)=108THEN6020
6010 NEXT:END
6020 IFPEEK(A%+1)=55THENPRINTA%
6030 GOTO6010
```

YC 88 1988.

```
2 CLS:COLOR2,1
3PRINTTAB(13)"U-FOE"
5INPUT"INSTRUCTIONS (Y/N)":A$:IFA$="Y"THENGOSUB2000
7P=68:H=63:Y=1:L=20:X=63:M=3
10MODE(1):U=0
15GOSUB4000
20FORA=1TO30:SET(RND(127).RND(4)):NEXT
25CLOR3
27FORA=1TO5:SET(H+A.L):NEXT
28COLOR2
30 FORA=1TO10:SET(X+A.Y):NEXT
35COLOR3:Y=Y+1
40 FORA=1TO14:SET(X-2+A.Y):NEXT
45COLOR4:Y=Y+1
50 FORA=1TO24:SEET(X+A-7.Y):NEXT
60 Y=Y+1
70FORA=1TO14:SET(X-2+A.Y):NEXT
75P=68:U=U+1:IFU>4THEN7ELSEIFM=2THENM=3ELSEIFM=3THENM=4ELSEM=2
76COL ORM
80FORA=5TO9:IFA>20THENL=32
85IFA>32THENL=44
90I=RND(20):D=RND(2)
100FORJ=1TOI
110IFA>44ANDPOINT(P.A)THENGOTO10
120IFD=1THENP=P-1ELSEP=P+1
122IFP<1THENP=1:D=2:A=A+1:GOTO120
124IFP>126THENP=126:D=D+1:A=A+1:GOTO120
127IFA=LANDP=>HANDP<=H+6THENSOUND25.5:P=68:L=20:H=63:GOTO75
130SET(P.A)
135GOSUB1000:COLORM
140NEXT
145NEXT
150SOUND3.4:3.4;3,4:ST=ST+1:IFST=3THEN5000ELSEL=20:H=63:GOTO75
1000A$=INKEY$:IFA$=""THENRETURN
1005COLORRND(3)+1
1006IFH>121ANDA$=","THEEN1040
1008IFH<1ANDA$="M"THEN1040
1010IFA$="M"THENSET(H-1.L):RESET(H+5.L):H=H-1
1020IFA$=","THENSET(H+6.L):RESET(H.L):H=H+1
1025COLOR4
1040RETURN
2000CLS:PRINTTAB(13)"U-FOE"
2010PRINT"YOU MUST DEFEND THE CITY AT THE BOTTOM OF THE ";
2020PRINT"SCREEN FROM ALIEN MISSILES WITH YOUR THREE ":
2030PRINT"SUPERSAUCERS(TM). THE CITY CAN WITHSTAND THREE ";
2040PRINT"DIRECT HITS BEFORE IT IS DESTROYED."
2050PRINT"CONTROLS ARE:"
2060PRINT"M-LEFT        .-RIGHT"
2070PRINT"IF YOU MISS THE MISSILE WITH YOUR FIRST SAUCER ";
2080PRINT"ANOTHER WILL APPEAR DIRECTLY UNDER IT AS SOON ":
2090PRINT"AS ONE OF CONTROLS IS PRESSED. THE SAME ";
2100PRINT"WILL HAPPEN WITH THE SECOND BUT NOT THE THIRD."
2110PRINT"PRESS ANY KEY TO PLAY."
2120IFINKEY$=""THEN2120
2130RETURN
4000FORA=40TO80
4010I=RND(ABS((A-39)-20))+44
4020FORJ=63TOISTEP-1
4030SET(A.J)
4040NEXT:NEXT
4045CLOR1
4050FORA=1TO30:SET(RND(18)+48.RND(13)+49):NEXT
4060COLOR2
4070RETURN
5000SOUND5.5;4.5;1.5
5010CLS:PRINTTAB(12)"SCHMUCK!"
5020PRINT"YOU LET THE CITY BE DESTROYED!"
```

U-FOE

Full instructions are included in the text, but the idea is to defend a city at the bottom of the screen with three flying saucers.

L. Alderton
Dunnedoo
NSW

ETI Apr. 88 p 65.

```
10 REM#DISINTEGRATOR#
20 MODE(0):CLS:COLOR,0
30 POKE30862,80:POKE30863,52
40 GOSUB900
50 GOSUB800
100 'START FRAME
110 CLS:POKE30744,1:D=16:N=3:U=0
120 FORL=29152TO29183
130 POKEL,255:NEXT
140 M=D+16:P=0
150 PRINT@0,"     CHAMP:"T$:PRINT@17,"PLAYER: "U$
160 PRINT@32,"HI SCORE: "T:PRINT@49,"  SCORE: "U
170 PRINT@64,"     BOMBS: "M:PRINT@81,"  CRAFT: "N
180 FORL=29121TO29151STEP2
190 K=(RND(5)-1)*16+172
200 H=RND(7)*32:P=P+H
210 FORX=HTO0STEP-32
220 POKEL-X,K
230 NEXT:NEXT
240 PRINT@139,"<S>=START"
250 A$=INKEY$:A$=INKEY$:IFA$<>"S"THEN250
260 PRINT@139,"                 ":SOUND31,1
270 L=28767:C=253.5:Z=.5:B=2
300 'MOVE CRAFT
310 A$=INKEY$
320 POKEL,32:L=L+1:POKEL,C+Z
330 IFRND(10)>5THENX=USR(X)
340 Z=-Z:FORI=0TOD*2:NEXT
350 IFB<2THEN370
360 IFA$=" "ANDM>0THEN400
370 B=B+1
390 IFPEEK(L+1)<>32THEN600
395 GOTO310
400 'DROP BOMB
410 SOUND20,1:M=M-1:F=L+32:B=0
420 PRINT@73,M
430 IFPEEK(F+32)=255THENSOUND10,1:POKEF,32:GOTO460
440 POKEF,32:F=F+32:POKEF,243
450 X=USR(X):GOTO430
460 FORY=29089TO29119STEP2
470 IFPEEK(Y)<>32THEN310
480 NEXT
500 'COLLECT POINTS
510 SOUND31,1;31,1
520 IFD>0THEND=D-2
530 M=M*(2000-(D*100))
540 IFD=0THENM=5000
550 U=U+M+P:POKEL,32:GOTO140
600 'WIPEOUT
610 SOUND15,1
620 COLOR,1:POKE30744,0
630 FORI=1TO50:NEXT:SOUND5,1
640 POKE30744,1:COLOR,0
650 N=N-1:M=(32-M)*10:U=U+M
660 FORL=28864TO29151
670 POKEL,32:NEXT
680 IFN=0THEN700
690 GOTO140
700 'END ROUND
710 IFU$=T$THEN730
720 IFU=TTHENT$="A DRAW"
730 IFU>TTHENT=U:T$=U$
740 PRINT@9,"          "
745 PRINT@9,T$:PRINT@88,N
750 PRINT@41,T:PRINT@56,U
760 PRINT@96,"<T>TRY AGAIN <N>NEW GAME <E>END"
765 PRINT@170,"#GAME OVER#"
770 A$=INKEY$
775 A$=INKEY$:IFA$=""THEN770
780 IFA$="T"THEN100
785 IFA$="N"THEN50
790 IFA$="E"THENCLS:END
795 GOTO770
800 'INITIAL
810 CLS:POKE30744,0
830 PRINT"PLAYER,PLEASE INPUT YOUR NAME"
840 PRINT"  NO MORE THAN SEVEN LETTERS"
850 PRINT:INPUTU$
860 S=LEN(U$)
870 IFS<1ORS>7THEN840
880 RETURN
900 'INSTRUCT
905 CLS:PRINTTAB(8);"#DISINTEGRATOR#"
910 PRINTTAB(7);"(BY ALAN STIBBARD)"
915 PRINT"YOU ARE IN A CRAFT WHICH HOVERS"
920 PRINT"OVER TALL STRUCTURES.YOUR TASK"
925 PRINT"IS TO DESTROY THESE BY DROPPING"
930 PRINT"BOMBS DOWN ON TO THEM BEFORE"
935 PRINT"YOUR ALTITUDE GETS TOO LOW AND"
940 PRINT"YOU CRASH INTO ONE OF THEM."
945 PRINT"THE GAME BECOME'S MORE DIFFICULT";
950 PRINT"AS YOU SUCCEED EACH FRAME.THE"
955 PRINT"NUMBER OF BOMBS WILL DECREASE;"
960 PRINT"AND THE SPEED OF THE CRAFT WILL"
965 PRINT"INCREASE.HIGHEST SCORER WINS!!.
970 PRINT"*BOMBS NOT DROPPED ARE A BONUS."
975 PRINT"*THE <SPACE> KEY DROPS THE BOMBS";
980 PRINT"  HIT RETURN KEY TO CONTINUE";:INPUTS$
985 RETURN
```

Disintegrator

This game is run on the VZ-200 or 300. All the instructions and comments are explained in the program.

A. Stibbard
Stanmore NSW

Star fighter

A game where you hit a UFO 10 times under a time and ammunition limit . . . <− (M) (,) −> (Z) FIRE (Q) END (R) RERUN. GOOD LUCK!

Murray Roberts
Eltham
Vic.

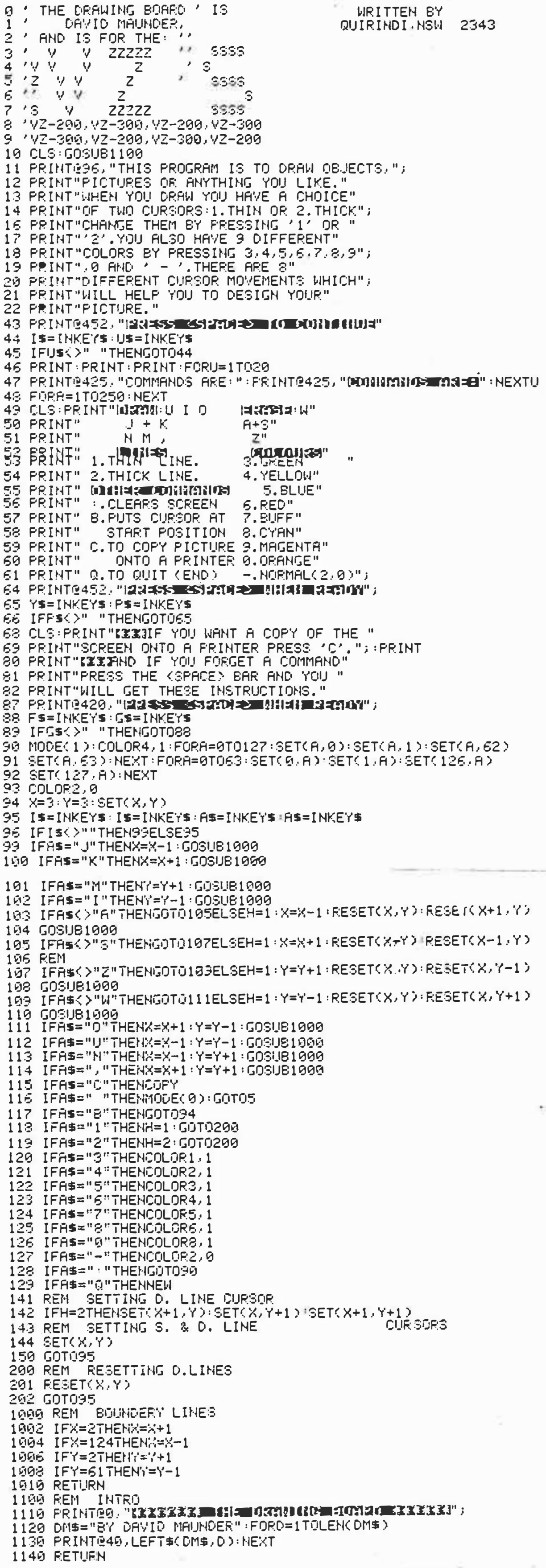
VZ 200/300

```
100 REMARKABLE ELECTRONICS TODAY    PROGRAM BY M.ROBERTS 1988...
110 '       (03) 439 2106 ......
120 ' V Z 3 0 0  OR  V Z 2 0 0
130 CLS:PRINT" PROGRAM FROM FEED FORWARD FROM ";
140 PRINT"E L E C T R O N I C S  T O D A Y";
150 PRINT@224,"NAME PLEASE";:INPUTNAME$
160 GOSUB610
170 TIME=1:M=0:H=1:BS=40:CLS
180 CLS
190 E=0:D=29168:E=INT(RND(29)):E=E+28832
200 POKED,30:POKEE-1,35:POKEE,35:POKEE+1,35
210 P=INT(RND(30))
220 P1=INT(RND(30))
230 P2=INT(RND(30))
240 P3=INT(RND(30))
250 IFP1=P2ORP1=PORP1=P3ORP2=PORP3=PORP3=P2,210
260 P=P+29088:P1=P1+29088:P2=P2+29088:P3=P3+29088
270 POKEP1,125:POKEP2,125:POKEP,125:POKEP3,125
280 A$=INKEY$:B$=INKEY$
290 PRINT@10,"BULLETS ";BS
300 IFB$=","ANDD<29183,GOTO400
310 IFB$="M"ANDD>29152,GOTO430
320 IFB$="Z"ANDPEEK(D-64)<>125,SOUND10,1:GOTO460
330 TIME=TIME+1
340 IFTIME>100,910
350 PRINT@22,"TIME"TIME
360 IFB$="Q",END
370 IFB$="R",GOTO170
380 GOSUB560
390 GOTO280
400 REM <-
410 TIME=TIME+1:POKED,96:GOSUB560:D=D+1:POKED,30:GOTO280
420 GOTO280
430 REM ->
440 TIME=TIME+1:POKED,96:GOSUB560:D=D-1:POKED,30:GOTO280
450 GOTO280
460 TIME=TIME+1:R=D:IFBS<1,920
470 BS=BS-1:FORA=1TO9
480 R=R-32:POKER,33:GOSUB560:M=0:POKER,96:NEXT
490 IFPEEK(R-32)=35,H1=H1+1:PRINT@0,H1;" HIT":GOTO510
500 GOTO280
510 SOUND31,1;20,1;10,1
520 IFH1>=10,820
530 REM GOT BULLETS
540 REM
550 GOTO280
560 POKEE-1,96:POKEE+1,96:POKEE,96:IFH=1ANDE<28863,E=E+1
570 IFH=2ANDE>28832,E=E-1
580 IFE=28862,H=2
590 IFE=28833,H=1
600 POKEE+1,35:POKEE,35:RETURN
610 FORA=1TO40:PRINT" STAR FIGHTER ";:NEXT
620 SOUND31,9
630 CLS:FORA=1TO400:NEXT
640 PRINT"  GREETINGS STAR FIGHTER":FORA=1TO800:NEXT
650 PRINT"  I AM YOUR ENEMY ";:FORA=1TO600:NEXT:PRINT"ZOD"
660 FORA=1TO800:NEXT
670 PRINT" TRY TO DESTROY ME, IF YOU CAN";:FORA=1TO1000:NEXT
680 PRINT"    PRESS":FORA=1TO300:NEXT
690 PRINT"    M FOR <-":FORA=1TO1000:NEXT
700 PRINT"    , FOR ->":FORA=1TO1000:NEXT
710 PRINT"    SPACE FOR FIRE":FORA=1TO1000:NEXT
720 PRINT"    Q TO QUIT":FORA=1TO1000:NEXT
730 PRINT"    R TO RESTART":FORA=1TO1000:NEXT
740 C$=" ANY  KEY"
750 PRINT@448,LEFT$(C$,31);:PRINT@458,LEFT$(C$,31);
760 PRINT@468,LEFT$(C$,31);:PRINTCHR$(28);:PRINT@478,"   ";
770 FORA=1TO30:A$=INKEY$:IFA$<>" ",NEXTELSERETURN
780 C$=MID$(C$,2)+LEFT$(C$,1)
790 GOTO750
800 RETURN
810 A$=INKEY$:A$=INKEY$:IFA$="",810ELSERETURN
820 CLS
830 PRINT:PRINT:PRINT:PRINT
840 PRINT"W   E   L   L       D   O   N   E";
850 PRINT:PRINT:PRINT
860 PRINT"S  T  A  R - F  I  G  H  T  E  R";
870 PRINT:PRINT
880 PRINT"            ";NAME$
890 PRINT:PRINT"    ANY KEY FOR ANOTHE GAME"
900 A$=INKEY$:A$=INKEY$:IFA$="",900
910 RUN
920 CLS
930 PRINT:PRINT:PRINT:PRINT
940 PRINT"B   A   D           L   U   C   K";
950 PRINT:PRINT:PRINT
960 PRINT"S  T  A  R - F  I  G  H  T  E  R";
970 PRINT:PRINT
980 PRINT"            ";NAME$
990 PRINT:PRINT"    ANY KEY FOR ANOTHER GAME"
1000 A$=INKEY$:A$=INKEY$:IFA$="",900
1010 RUN
```

Drawing board

This program allows the user to create pictures in Mode <1> with 8 different cursor movements. All the instructions are contained in the program.

D. Maunder
Quirindi
NSW



```
0 ' THE DRAWING BOARD ' IS                 WRITTEN BY
1 '     DAVID MAUNDER,                   QUIRINDI.NSW  2343
2 ' AND IS FOR THE: ''
3 '  V   V  ZZZZZ   ''  SSSS
4 'V V   V     Z     ' S
5 'Z  V V     Z     '   SSSS
6 ''  V V    Z            S
7 'S   V    ZZZZZ        SSSS
8 'VZ-200,VZ-300,VZ-200,VZ-300
9 'VZ-300,VZ-200,VZ-300,VZ-200
10 CLS:GOSUB1100
11 PRINT@96,"THIS PROGRAM IS TO DRAW OBJECTS,";
12 PRINT"PICTURES OR ANYTHING YOU LIKE."
13 PRINT"WHEN YOU DRAW YOU HAVE A CHOICE"
14 PRINT"OF TWO CURSORS:1.THIN OR 2.THICK";
16 PRINT"CHANGE THEM BY PRESSING '1' OR "
17 PRINT"'2'.YOU ALSO HAVE 9 DIFFERENT"
18 PRINT"COLORS BY PRESSING 3,4,5,6,7,8,9";
19 PRINT",0 AND ' - '.THERE ARE 8"
20 PRINT"DIFFERENT CURSOR MOVEMENTS WHICH";
21 PRINT"WILL HELP YOU TO DESIGN YOUR"
22 PRINT"PICTURE."
43 PRINT@452,"PRESS <SPACE> TO CONTINUE"
44 I$=INKEY$:U$=INKEY$
45 IFU$<>" "THENGOTO44
46 PRINT:PRINT:PRINT:FORU=1TO20
47 PRINT@425,"COMMANDS ARE:":PRINT@425,"COMMANDS ARE:":NEXTU
48 FORA=1TO250:NEXT
49 CLS:PRINT"DRAW:U I O    ERASE:W"
50 PRINT"      J + K         A+S"
51 PRINT"      N M ,          Z"
52 PRINT"     LINES          COLOURS"
53 PRINT" 1.THIN LINE.     3.GREEN      "
54 PRINT" 2.THICK LINE.     4.YELLOW"
55 PRINT" OTHER COMMANDS     5.BLUE"
56 PRINT" :.CLEARS SCREEN   6.RED"
57 PRINT" B.PUTS CURSOR AT  7.BUFF"
58 PRINT"   START POSITION  8.CYAN"
59 PRINT" C.TO COPY PICTURE 9.MAGENTA"
60 PRINT"    ONTO A PRINTER 0.ORANGE"
61 PRINT" Q.TO QUIT (END)   -.NORMAL(2,0)";
64 PRINT@452,"PRESS <SPACE> WHEN READY";
65 Y$=INKEY$:P$=INKEY$
66 IFP$<>" "THENGOTO65
68 CLS:PRINT"***IF YOU WANT A COPY OF THE "
69 PRINT"SCREEN ONTO A PRINTER PRESS 'C'.";:PRINT
80 PRINT"***AND IF YOU FORGET A COMMAND"
81 PRINT"PRESS THE <SPACE> BAR AND YOU "
82 PRINT"WILL GET THESE INSTRUCTIONS."
87 PRINT@420,"PRESS <SPACE> WHEN READY";
88 F$=INKEY$:G$=INKEY$
89 IFG$<>" "THENGOTO88
90 MODE(1):COLOR4,1:FORA=0TO127:SET(A,0):SET(A,1):SET(A,62)
91 SET(A,63):NEXT:FORA=0TO63:SET(0,A):SET(1,A):SET(126,A)
92 SET(127,A):NEXT
93 COLOR2,0
94 X=3:Y=3:SET(X,Y)
95 I$=INKEY$:I$=INKEY$:A$=INKEY$:A$=INKEY$
96 IFI$<>""THEN99ELSE95
99 IFA$="J"THENX=X-1:GOSUB1000
100 IFA$="K"THENX=X+1:GOSUB1000
101 IFA$="M"THENY=Y+1:GOSUB1000
102 IFA$="I"THENY=Y-1:GOSUB1000
103 IFA$<>"A"THENGOTO105ELSEH=1:X=X-1:RESET(X,Y):RESET(X+1,Y)
104 GOSUB1000
105 IFA$<>"S"THENGOTO107ELSEH=1:X=X+1:RESET(X,Y):RESET(X-1,Y)
106 REM
107 IFA$<>"Z"THENGOTO109ELSEH=1:Y=Y+1:RESET(X,Y):RESET(X,Y-1)
108 GOSUB1000
109 IFA$<>"W"THENGOTO111ELSEH=1:Y=Y-1:RESET(X,Y):RESET(X,Y+1)
110 GOSUB1000
111 IFA$="O"THENX=X+1:Y=Y-1:GOSUB1000
112 IFA$="U"THENX=X-1:Y=Y-1:GOSUB1000
113 IFA$="N"THENX=X-1:Y=Y+1:GOSUB1000
114 IFA$=","THENX=X+1:Y=Y+1:GOSUB1000
115 IFA$="C"THENCOPY
116 IFA$=" "THENMODE(0):GOTO5
117 IFA$="B"THENGOTO94
118 IFA$="1"THENH=1:GOTO200
119 IFA$="2"THENH=2:GOTO200
120 IFA$="3"THENCOLOR1,1
121 IFA$="4"THENCOLOR2,1
122 IFA$="5"THENCOLOR3,1
123 IFA$="6"THENCOLOR4,1
124 IFA$="7"THENCOLOR5,1
125 IFA$="8"THENCOLOR6,1
126 IFA$="0"THENCOLOR8,1
127 IFA$="-"THENCOLOR2,0
128 IFA$=":"THENGOTO90
129 IFA$="Q"THENNEW
141 REM  SETTING D. LINE CURSOR
142 IFH=2THENSET(X+1,Y):SET(X,Y+1):SET(X+1,Y+1)
143 REM  SETTING S. & D. LINE                    CURSORS
144 SET(X,Y)
150 GOTO95
200 REM  RESETTING D.LINES
201 RESET(X,Y)
202 GOTO95
1000 REM  BOUNDERY LINES
1002 IFX=2THENX=X+1
1004 IFX=124THENX=X-1
1006 IFY=2THENY=Y+1
1008 IFY=61THENY=Y-1
1010 RETURN
1100 REM  INTRO
1110 PRINT@0,"******* THE DRAWING BOARD ******";
1120 DM$="BY DAVID MAUNDER":FORD=1TOLEN(DM$)
1130 PRINT@40,LEFT$(DM$,D):NEXT
1140 RETURN
```

Camel

Camel is a popular game that is played by many people. There are many versions of the game. This is one. Instructions to play are included in the program.

D. Maunder,
Quirindi,
NSW.

ETI May 89.
p 87-88.
1 of 2.

```
 0 REM  THERE ARE MANY DIFFERENT VERSIONS OF CAMEL FOR DIFFERENT
 1 REM  COMPUTERS.I HAVE CONVERTED CAMEL FOR THE VZ-200 & VZ-300
 2 REM  ORIGINAL IDEA FROM HEATH USERS GROUP.
 3 REM ************************
 4 REM CAMEL OCCUPIES ABOUT 6K
 5 REM 'CAMEL' CONVERTED AND REWRITTEN BY D.MAUNDER  [C]OPYRIGHT
 6 REM      06/02/89  FOR THE
7 REM V V V     V   ZZZZZZ V V
8 REM Z Z V     V       Z  Z Z
9 REM ! !  V   V       Z   ! !
10 REM 2 3  V   V      Z    2 3
11 REM 0 0   V V      Z     0 0
12 REM 0 0    V      ZZZZZZ 0 0
13 REM  VZ200/VZ300 COMPUTERS
14 REM ************************
15 GOSUB214:PRINT"▮      WELCOME TO CAMEL":REM INVERSE
16 PRINT
17 PRINT
18 PRINT"▮ WOULD YOU LIKE INSTRUCTIONS?"
19 PRINT"▮                 Y/N"
20 T$=INKEY$
21 IFT$="N"THENGOTO36
22 IFT$="Y"THENGOTO25
23 IFT$<>"N"ORT$<>"Y"THENGOTO20
24 GOTO20
25 CLS:PRINT"           C  A  M  E  L":REM INVERSE"
26 PRINT
27 PRINT"THE OBJECT OF THE GAME IS TO"
28 PRINT"TRAVEL 200 KILOMETRES ACROSS THE";
29 PRINT"GREAT GOBI DESERT."
30 PRINT"A TRIBE OF KNOCK KNEED DESERT   PYGMIES WILL BE ";
31 PRINT"CHASING YOU."
32 PRINT"YOU HAVE TWO LITRES OF WATER    WHICH WILL LAST YOU ";
33 PRINT"SIX DRINKS"
34 PRINT:PRINT
35 INPUT"   PRESS <RETURN> TO CONTINUE";VZ$:REM INVERSE
36 POKE30777,35:FORB=1TO30
37 PRINT@384,"**GOOD LUCK AND GOOD CAMELING**"
38 PRINT@384,;
39 PRINT"**GOOD LUCK AND GOOD CAMELING**"
40 NEXTB
41 POKE30777,67:GOSUB213:CLS
42 GOSUB200:PRINT
43 PRINT"YOU ARE IN THE MIDDLE OF THE    DESERT AT AN OASIS"
44 GOSUB213
45 IFC>60THEN141
46 Z=Z-1
47 IFZ=1THENPRINT"-----WARNING-----  GET A DRINK"
48 IFZ=0THEN194
49 P=P+1
50 J=RND(10)+2.5
51 IFQ>0THEN111
52 IFP<4THEN60
53 D=D+J
54 IFD<CTHEN59
55 PRINT"THE PYGMIES HAVE CAPTURED YOU.."
56 PRINT"CAMEL & PEOPLE SOUP IS THEIR    FAVOURITE DISH !!!"
57 GOSUB210
58 GOTO188
59 PRINT"THE PIGMIES ARE"C-D"KILOMETRES  BEHIND YOU.."
60 PRINT"YOU HAVE TRAVELLED"C"
61 PRINT"KILOMETRES SO FAR..."
62 PRINT:INPUT"WHAT IS YOUR COMMAND?";A$
63 POKE30777,67
64 IFA$="D"THENPRINT"DRINK FROM CANTEEN.":GOTO100
65 IFA$="M"THENPRINT"MODERATE SPEED AHEAD.":GOTO79
66 IFA$="F"THENPRINT"FULL SPEED AHEAD.":GOTO86
67 IFA$="N"THENPRINT"NIGHT STOP.":GOTO94
68 IFA$="S"THENPRINT"STATUS CHECK.":GOTO97
69 IFA$="H"THENPRINT"HOPE FOR HELP":GOTO73
70 IFA$="*"THENPRINT"COMMANDS":GOSUB200:GOTO62
71 PRINT"INVALID COMMAND."
72 GOSUB200:GOTO62
73 T=RND(10)+1
74 IFT<2THEN140
75 PRINT"HELP HAS FOUND YOU IN A STATE   OF UNCONCIOUSNESS"
76 S=3
77 Z=4
78 GOTO45
79 F=F+1
80 IFF=8THEN138
81 GOSUB105
82 I=RND(10)+1
83 C=C+I
84 PRINT"YOUR CAMEL LIKES THIS PACE."
85 GOTO45
86 F=F+3
87 IFF>7THEN138
88 GOSUB105
89 I=2*RND(10)+1
90 C=C+I
91 PRINT"YOUR CAMEL IS BURNING ACROSS THEDESERT SANDS.."
92 PRINT
93 GOTO45
94 PRINT"YOUR CAMEL THANKS YOU!!!"
95 F=0
96 GOTO46
97 PRINT"YOUR CAMEL HAS"7-F"GOOD DAYS LEFT FOR TRAVELLING."
98 PRINT"YOU HAVE"S"DRINKS LEFT IN YOUR  CANTEEN"
99 PRINT"YOU CAN GO"Z"COMMANDS WITHOUT   DRINKING.."
100 S=S-1
101 IFS<0THEN140
102 PRINT"YOU HAD BETTER WATCH OUT FOR AN OASIS."
103 Z=4
104 GOTO62
105 A=RND(20)
106 IFA>3THEN131
107 PRINT"WILD BERBERS ARE HIDDEN IN THE  SAND HAVE
    CAPTURED YOU"
```

```
108 PRINT"LUCKILY THE LOCAL SHEIK HAS      AGREED TO THEIR "
109 PRINT"DEMANDS ....BUT.....WATCH OUT FOR THE PYGMIES!!!..."
110 PRINT"YOU HAVE A NEW CHOICE OF SUB-    COMMANDS:"
111 PRINT"KEY          DESCRIPTION"
112 PRINT" E      :       ATTEMPT AN ESCAPE."
~~113 PRINT~~" W      :       WAIT FOR PAYMENT."
114 INPUT"YOUR SUB-COMMAND ??";B$
115 IFB$="E"THENPRINT"ESCAPE.":GOTO118
116 IFB$="W"THENPRINT"WAIT FOR PAYMENT.":GOTO126
117 GOTO114
118 I=RND(10)+1
119 IFI<5THEN123
120 PRINT"CONGRATULATIONS, YOU SUCCESS-    FULLY ESCAPED!!"
121 Q=0
122 GOTO45
123 PRINT"YOU WERE MORTALLY WOUNDED BY A  PIG STABBER WHILE"
124 PRINT"ESCAPING.."
125 GOTO166
126 I=RND(10)
127 IFI>5THEN148
128 PRINT"YOUR RANSOM HAS BEEN PAID AND    YOU ARE FREE TO GO.."
129 PRINT"THE LOCAL SULTAN IS COLLECTING..JUST WAIT...."
130 GOTO45
131 A=RND(10)+1
132 IFA>4THEN148
133 PRINT"YOU HAVE ARRIVED AT AN OASIS....YOUR CAMEL IS ";
134 PRINT"FILLING YOUR       CANTEEN AND EATING FIGS"
135 Z=4
136 S=6
137 RETURN
138 PRINT"YOU DIRTY RAPSCALLION!!YOU RAN  YOUR POOR CAMEL TO";
139 PRINT" DEATH!!"
140 GOTO166
141 PRINT"********************************";
142 PRINT"*YOU WIN,A PARTY IS BEING GIVEN*";
143 PRINT"*IN YOUR HONOR.....THE PYGMIES *";
144 PRINT"*ARE PLANNING TO ATTEND......   *";
145 PRINT"********************************";
146 FORT=1TO6:SOUND0,2:SOUND23,2:SOUND0,2:NEXTT
147 GOTO188
148 I=RND(10)
149 IFI>5THEN160
150 PRINT"YOU HAVE BEEN CAUGHT IN A SAND- STORM....GOOD LUCK!!"
151 M=RND(10)+1
152 N=RND(10)+1
153 IFN<5THEN156
154 C=C+M
155 GOTO157
156 C=C-M
157 PRINT"YOUR NEW POSITION IS"C"KILOMETRES SO FAR!!"
158 PRINT"THE PYGMIES ARE"C-D"KILOMETRES BEHIND YOU"
159 RETURN
160 I=RND(10)
161 IFI>5THENRETURN
162 D=D+1
163 PRINT"YOUR CAMEL HURT HIS HUMP."
164 PRINT"LUCKILY THE PYGMIES WERE FOOT-  WEARY!!!"
165 RETURN
166 U=RND(5)
167 PRINT"********************************";
168 PRINT"***  YOU DIED IN THE DESERT   ***";
169 PRINT"********************************";
170 GOSUB210
171 IFU=1THENGOTO176
172 IFU=2THENGOTO179
172 IFU=2THENGOTO179
173 IFU=3THENGOTO181
174 IFU=4THENGOTO184
175 IFU=5THENGOTO187
176 PRINT"THE NATIONAL CAMELS UNION ISN'T"
177 PRINT"COMING TO YOUR FUNERAL"
178 GOTO188
179 PRINT"YOUR BODY WAS EATEN BY VULTURES & IMPORTED CANNIBALS"
180 GOTO188
181 PRINT"THE LOCAL SHEIK NOW USES YOUR    SKULL AS A CHANGE"
182 PRINT"PURSE!!!!!!!!"
183 GOTO188
184 PRINT"PEOPLE WITH LITTLE INTELLIGENCE SHOULD STAY OUT OF"
185 PRINT"THE DESERT."
186 GOTO188
187 PRINT"TURKEYS SHOULD FLY,NOT RIDE      CAMELS!!!"
188 PRINT:PRINT
189 PRINT"WANT A NEW CAMEL AND A NEW GAME":A$=INKEY$
190 A$=INKEY$
191 IFA$="N"THENGOTO196
192 IFA$="Y"THENGOTO36
193 GOTO190
194 PRINT"YOU RAN OUT OF WATER....SORRY                   CHUM!!"
195 GOTO166
196 PRINT"********************************"
197 PRINT"*****        CHICKEN        *****"
198 PRINT"********************************"
199 FORA=1TO2000:NEXT:POKE30845,199
200 PRINT"                COMMANDS":REM INVERSE
201 PRINT"KEY    DESCRIPTIONS"
202 PRINT" D  :  DRINK FROM YOUR CANTEEN"
```

```
203 PRINT" F   :  AHEAD FULL SPEED"
204 PRINT" M   :  AHEAD MODERATE SPEED"
205 PRINT" N   :  STOP FOR NIGHT"
206 PRINT" S.  :  STATUS CHECK"
207 PRINT" H   :  HOPE FOR HELP"
208 PRINT" *   :  LIST OF COMMANDS"
209 RETURN
210 FORT=1TO3:SOUND1,2:SOUND0,2:SOUND0,2:NEXTT
211 SOUND 1,6
212 RETURN
213 Z=4:S=6:C=0:D=0:F=0:P=0:Q=0:RETURN
214 POKE30777,67:CLS:COLOR8
215 PRINT@0," ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ ";
216 PRINT@480," ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━";
217 FORB=32TO448STEP32
218 PRINT@B," ▮";
219 PRINT@B+31,"▮";
220 NEXTB
221 POKE29183,184
222 PRINT@32,;
223 RETURN
```

2 of 2

BUSINESS

Aug.	84	APC	172-7	Database VZ-200. (Barker)	(6)
Oct.	84	APC	214	WP for VZ-200. (McQuillan)	(-)
Oct.	85	APC	82-3	Comment on Barker's and Quinn's DB. (Lukes)	(-)
Oct.	84	APC	126-30	Minicalc Spreadsheet. (Stamboulidas)	(5)
Dec.	84	APC	214	Correction to Minicalc.	(1)
May	85	APC	162-3	Micro Type(WP). (Browell)	(2)
Jul.	85	APC	164-6	Database. (Quinn)	(2)
Feb.	88	ETI	72	VZ Wordprocessor. (Tunny)	(1)

Database VZ-200

by Ted Barker

This is an information storage and retrieval program for the VZ-200 with 16k expansion used in conjunction with a suitable cassette recorder and 80 column printer. The program has been adapted from one written for the Commodore VIC 20/64 by John Stilwell of Madison, WI, USA which was published in the February, 1984 issue of the magazine *RUN*.

When you run the program you will be asked to enter a file name, [RETURN] Without entering a file name will result in a default to the file title 'NO NAME'. Some three seconds later a list of one-letter commands will be displayed. [M] will display a full menu, detailing the meanings of the one-letter commands. (Menu may be called at any time without affecting file entries).

Information is entered into pages, [P], each of which holds 10 line numbers. [E]. The total amount of information which may be filed is determined by the value of 'N' (number of lines) in Line 140. In the listing 'N' has a value of 400 which should allow up to 32 characters of entry per line.

Commands

'C' (Catalogue) will display the file name together with any lines you have designated as catalogue entries. (See Using The Catalogue). 'P' (Page), will ask you to enter a page number, (1 to 40 in the program listed). Enter a page number and press [RETURN] and the page, together with 10 lines will be displayed, ready for reading or making an entry. 'E' (Entry) asks for a line number; enter the required line number and press [RETURN]; enter the information you wish to file and press [RETURN] once more; your entry will then be confirmed on screen. 'I' (Insert) follows the same procedure as Enter. When you enter the desired information, it will be entered at the designated line number. All lines with a higher number will be incremented by one and no information will be lost. 'N' (New File Name) will ask 'Are you sure?'. Entering a new file name will result in the loss of any information stored in the current file. 'S' (Save to Tape) will ask 'Are you sure?'. If your answer is 'Y' just follow the screen prompts. As each entry is saved onto tape, its line number will be displayed at lower left screen. This serves as a check that the saving process is working OK. The word 'COMPLETE' will appear when all of your file is saved to tape.

'L' (Load From Tape) will again ask for confirmation. Load is similar in operation to Save except that you will be asked to enter a file name. During loading, the word 'WAITING' will appear as usual. This will be followed by the word 'FOUND'. The word 'LOADING' does *not* appear. As each item is loaded its line number will be displayed at lower left screen as a check that the loading procedure is going according to plan.

'H' (Hard Copy) will allow you to produce a print-out of your file. The file name will be enhanced, followed by the remainder of your file in unenhanced type. You will have the option of printing the entire file or of specifying a starting and ending line number.

'D' (Delete) follows the same procedure as Enter and Insert. A line number entered after the 'D' command will result in that line being deleted from the file. Higher line numbers will be decreased by one, leaving no gaps in your file. No information will be lost, except for the line you deleted. 'A' (Alphabetize) allows entries to be placed in order after they have been entered. You may place the entire file in alphabetical order, or specify starting and ending line numbers. (Note remarks in Using The Catalogue.) 'M' (Menu) displays menu on screen in case you forget what the single letter commands mean.

Using the catalogue

If you wish to split your files into separate categories you may display category headings in the file catalogue. To do this, the entry is made in the usual way but with an inverse 'C' preceding the entry. Line 170 in the program reads this character, ASC(195), and places those entries in the catalogue, together with the page number on which they appear.

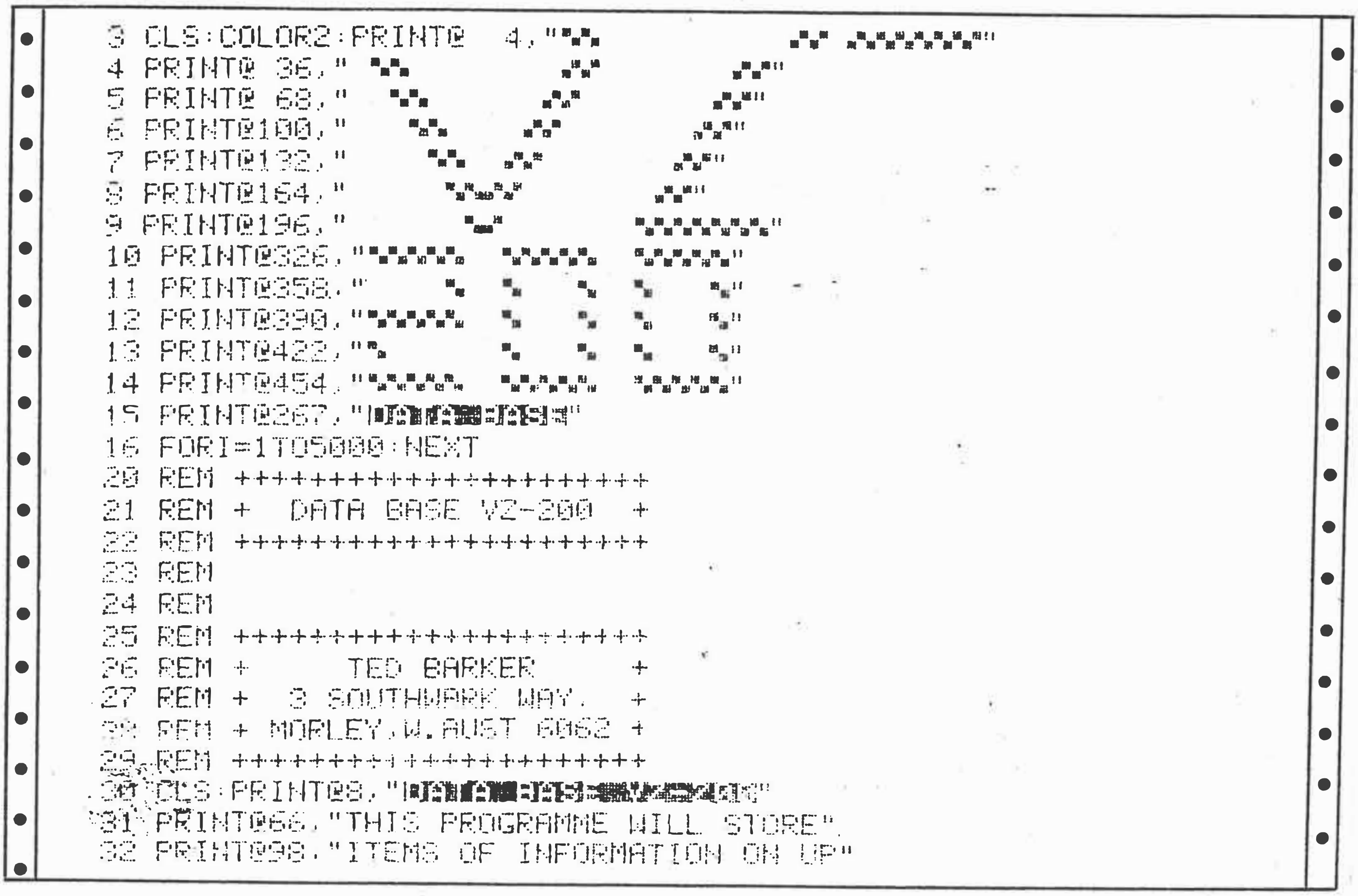
As these entries still appear in their correct position in the body of the file, it may be an advantage to enter the whole of that entry in inverse print, thus making the category headings stand out when going through the file. When using the Alphabetize routine on a file containing inverse 'C' entries, it is essential that the line number after the inverse 'C' entry is used as a starting line and that the ending number should not be greater than the entry containing the next inverse 'C'. If this rule is not observed the inverse 'C' entries will be alphabetized with the rest of the file, thus destroying its usefulness as a category heading.

If you enter a command letter by mistake, just enter another command instead of a line number. Omit all line numbers below 100 as this will allow a little more memory available for your database.

When you have completed all your entries, make one more entry on the next line. This entry should be 'END'. If you do this the alphabetize and catalogue routines will run much more quickly.

A.P.C. Aug 84. V5(8).
p 172-177
1 of 6.

(see updated version in A.P.C. Jul 85)



```
3 CLS:COLOR2:PRINT@  4,"▚▚                    ▚▚ ▚▚▚▚▚▚▚"
4 PRINT@ 36," ▚▚            ▚▚          ▚▚"
5 PRINT@ 68,"  ▚▚         ▚▚          ▚▚"
6 PRINT@100,"   ▚▚      ▚▚          ▚▚"
7 PRINT@132,"    ▚▚   ▚▚          ▚▚"
8 PRINT@164,"     ▚▚▚▚▚          ▚▚"
9 PRINT@196,"       ▚▚          ▚▚▚▚▚▚▚▚"
10 PRINT@326,"▚▚▚▚▚  ▚▚▚▚▚  ▚▚▚▚▚"
11 PRINT@358,"    ▚  ▚   ▚  ▚   ▚"
12 PRINT@390,"▚▚▚▚▚  ▚   ▚  ▚   ▚"
13 PRINT@422,"▚      ▚   ▚  ▚   ▚"
14 PRINT@454,"▚▚▚▚▚  ▚▚▚▚▚  ▚▚▚▚▚"
15 PRINT@267,"[illegible]"
16 FORI=1TO5000:NEXT
20 REM ++++++++++++++++++++++
21 REM +  DATA BASE VZ-200  +
22 REM ++++++++++++++++++++++
23 REM
24 REM
25 REM ++++++++++++++++++++++
26 REM +      TED BARKER    +
27 REM +  3 SOUTHWARK WAY.  +
28 REM + MORLEY,W.AUST 6062 +
29 REM ++++++++++++++++++++++
30 CLS:PRINT@8,"[illegible]"
31 PRINT@66,"THIS PROGRAMME WILL STORE"
32 PRINT@98,"ITEMS OF INFORMATION ON UP"
```

A.P.C. Aug 84 V5(8).
p 172-177
2 of 6.

WP for VZ-200

Many thanks to you and to Ted Barker with his VZ-200 Database. It's nice to know somebody remembers the little people. As Dick Smith continually sprooks, there have been 'over 25,000 sold', and that's 25,000 people out there with no usable software to speak of.

Does anybody know of a suitable word-processor type program for the VZ-200? I can't find one!

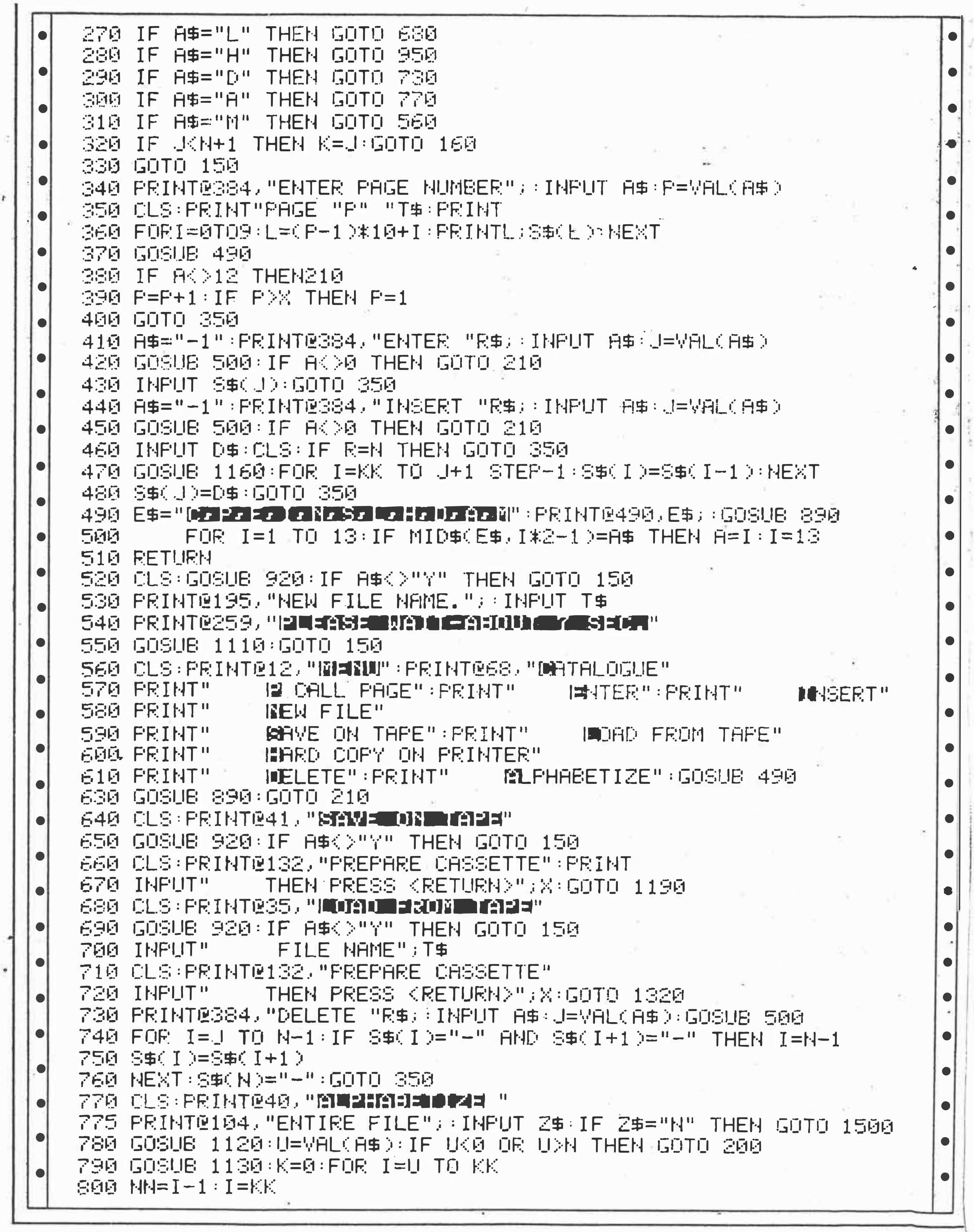
Again, many thanks, and keep up the good work.

Ben McQuillan

APC 5(10) Oct. 84 p 214.

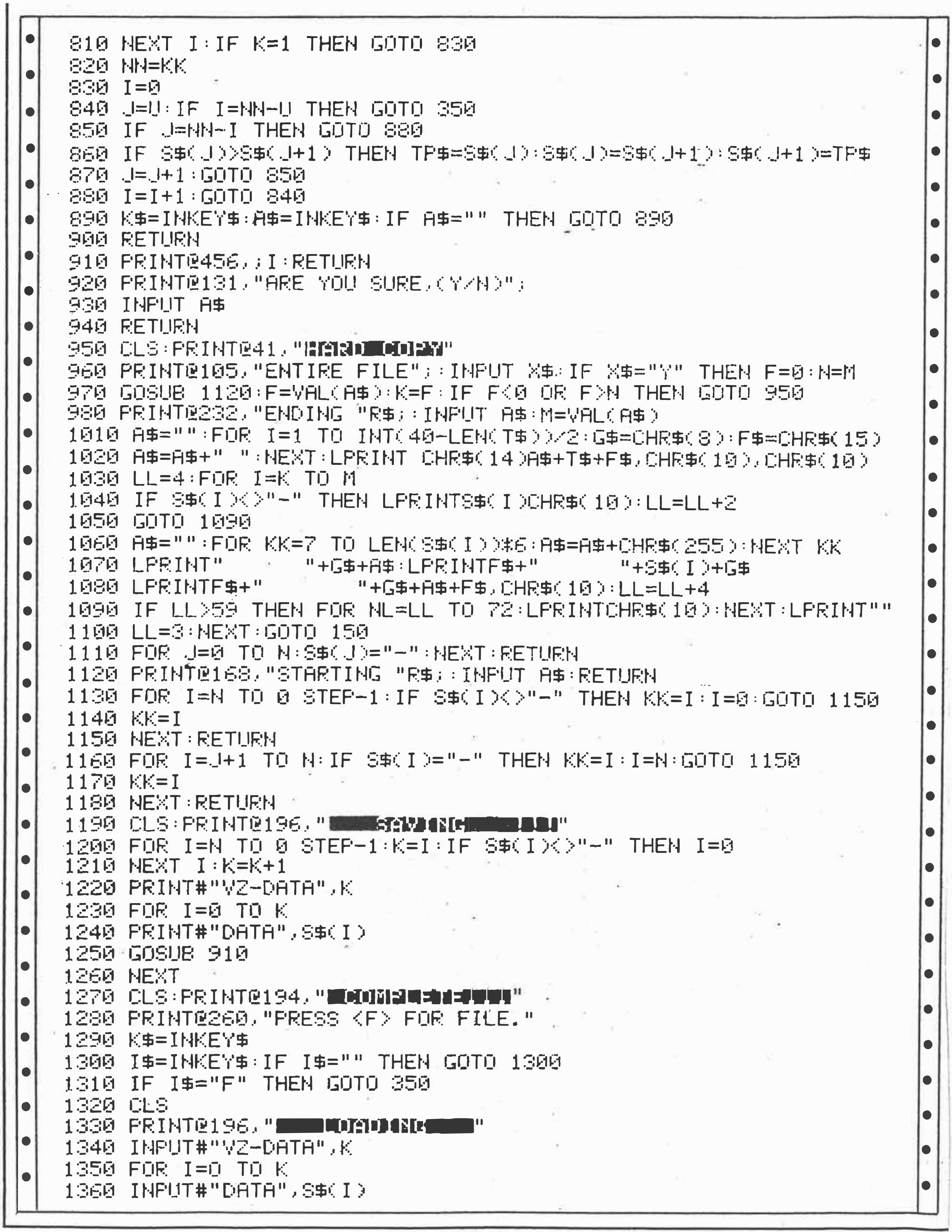
```
33 PRINT@130,"TO 40 PAGES, EACH PAGE"
34 PRINT@162,"CONTAINING 10 LINES."
35 PRINT@226,"YOU MAY ENTER,INSERT,DELETE"
36 PRINT@258,"OR OVERWRITE INFORMATION-"
37 PRINT@290,"ALPHABETIZE OR PRINT ALL"
38 PRINT@322,"OR PART OF YOUR FILES-"
39 PRINT@354,"WHICH CAN THEN BE SAVED"
40 PRINT@386,"AND/OR RETRIEVED FROM TAPE."
41 PRINT@450,"PRESS <SPACE> TO CONTINUE";
42 K$=INKEY$:I$=INKEY$:IF I$<>" "THEN 42
43 CLS:PRINT@8,"EQUIPMENT"
44 PRINT@34,"YOU WILL REQUIRE THE 16K"
45 PRINT@66,"EXPANSI ON WITH YOUR VZ-200,"
46 PRINT@98,"A CASSETTE RECORDER AND"
47 PRINT@130,"SUITABLE PRINTER."
48 PRINT@194,"WHEN RETRIEVING A FILE"
49 PRINT@226,"FROM TAPE, THE WORD"
50 PRINT@258,"LOADING WILL NOT APPEAR."
51 PRINT@290,"YOU WILL SEE LOADING.
52 PRINT@322,"FOLLOWED BY FOUND.
53 PRINT@354,"THE NUMBER OF EACH FILE"
54 PRINT@386,"ENTRY WILL BE DISPLAYED"
55 PRINT@418,"AS EACH ENTRY IS LOADED."
58 PRINT@451,"PRESS <SPACE> TO CONTINUE"
59 K$=INKEY$:I$=INKEY$:IF I$<>" " THEN 59
60 PRINT@482,"PRESS <SPACE> TO CONTINUE";
61 K$=INKEY$:I$=INKEY$:IF I$<>" " THEN 51
62 CLS:PRINT@66,"WHEN ENTERING THIS"
63 PRINT@98,"PROGRAMME, YOU WILL HAVE MORE"
64 PRINT@130,"MEMORY FOR YOUR FILES IF"
65 PRINT@162,"YOU OMIT LINES BELOW 100."
66 PRINT@298,"GOOD LUCK!"
68 PRINT@451,"PRESS <SPACE> TO CONTINUE"
69 K$=INKEY$:I$=INKEY$:IF I$<>" " THEN 69
100 CLS:PRINT@200,PEEK(30897)+256*PEEK(30898):CLEAR 12000
110 CLS:PRINT"FILE NAME.";:INPUT T$:IF T$="" THEN T$="NO NAME
120 PRINT@134,"PLEASE WAIT ABOUT 45 SECS"
130 T$=LEFT$(T$,14)
140 N=400  :P=1:X=(N+1)/10:DIMS$(N):R$="LINE NUMBER"
145 GOSUB 1110:GOTO 200
150 K=0
160 CLS:PRINT@7,"CATALOGUE";T$:PRINT:PRINT"PAGE:"
165 FOR J=K TO N
170 IFASC(S$(J))=195,PRINTINT(J/10+1);RIGHT$(S$(J),LEN(S$(J))-1)
180 IFS$(J)="END" THEN 200
190 NEXTJ
200 GOSUB 490:IF A$="A"THEN GOTO 200
210 IF A$="C" THEN GOTO 150
220 IF A$="P" THEN GOTO 340
230 IF A$="E" THEN GOTO 410
240 IF A$="I" THEN GOTO 440
250 IF A$="N" THEN GOTO 520
260 IF A$="S" THEN GOTO 640
```

Aug 84. 3 of 6. V5(8)



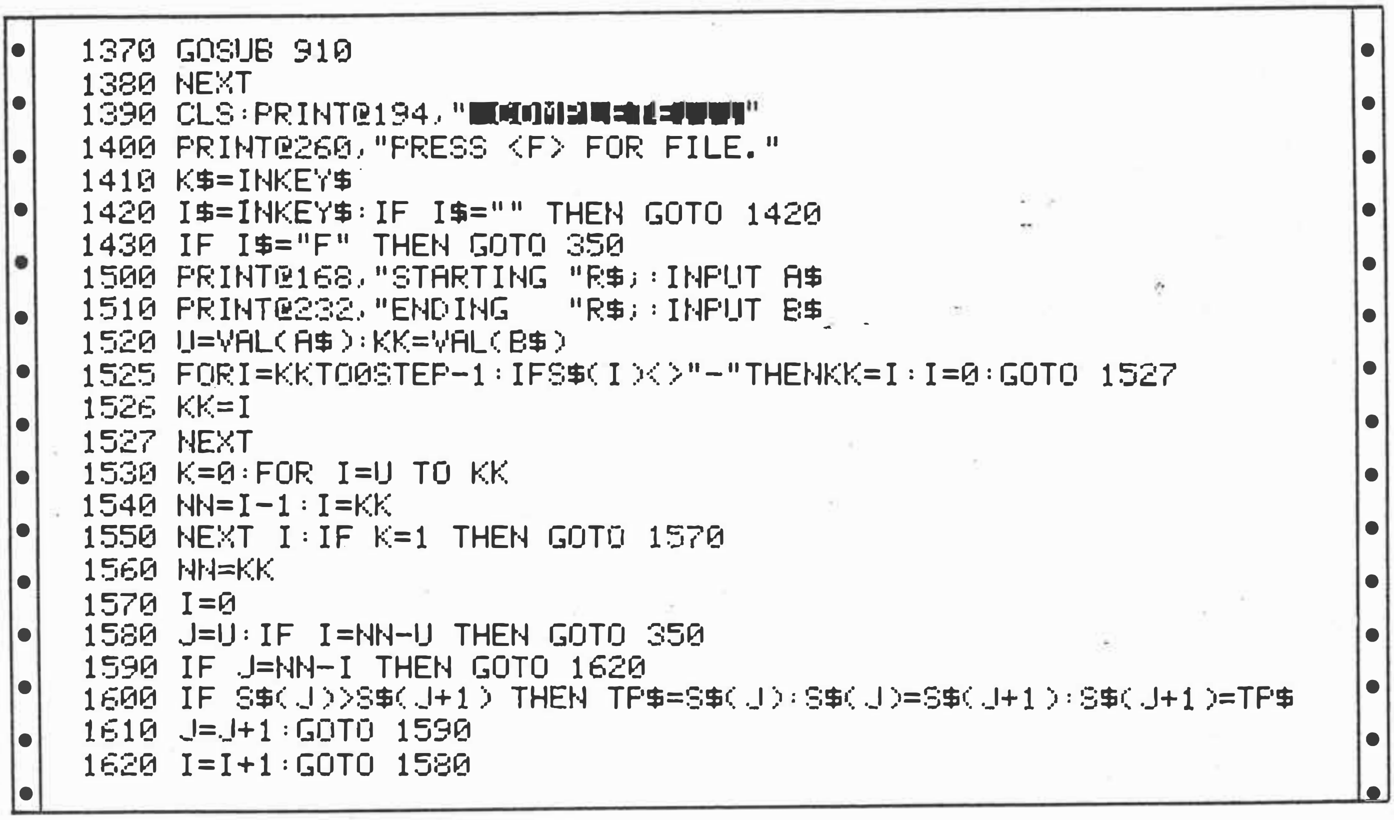
```
270 IF A$="L" THEN GOTO 680
280 IF A$="H" THEN GOTO 950
290 IF A$="D" THEN GOTO 730
300 IF A$="A" THEN GOTO 770
310 IF A$="M" THEN GOTO 560
320 IF J<N+1 THEN K=J:GOTO 160
330 GOTO 150
340 PRINT@384,"ENTER PAGE NUMBER";:INPUT A$:P=VAL(A$)
350 CLS:PRINT"PAGE "P" "T$:PRINT
360 FORI=0TO9:L=(P-1)*10+I:PRINTL;S$(L):NEXT
370 GOSUB 490
380 IF A<>12 THEN210
390 P=P+1:IF P>X THEN P=1
400 GOTO 350
410 A$="-1":PRINT@384,"ENTER "R$;:INPUT A$:J=VAL(A$)
420 GOSUB 500:IF A<>0 THEN GOTO 210
430 INPUT S$(J):GOTO 350
440 A$="-1":PRINT@384,"INSERT "R$;:INPUT A$:J=VAL(A$)
450 GOSUB 500:IF A<>0 THEN GOTO 210
460 INPUT D$:CLS:IF R=N THEN GOTO 350
470 GOSUB 1160:FOR I=KK TO J+1 STEP-1:S$(I)=S$(I-1):NEXT
480 S$(J)=D$:GOTO 350
490 E$="[illegible]":PRINT@490,E$;:GOSUB 890
500      FOR I=1 TO 13:IF MID$(E$,I*2-1)=A$ THEN A=I:I=13
510 RETURN
520 CLS:GOSUB 920:IF A$<>"Y" THEN GOTO 150
530 PRINT@195,"NEW FILE NAME.";:INPUT T$
540 PRINT@259,"PLEASE WAIT-ABOUT [illegible] SEC."
550 GOSUB 1110:GOTO 150
560 CLS:PRINT@12,"MENU":PRINT@68,"CATALOGUE"
570 PRINT"     C CALL PAGE":PRINT"     ENTER":PRINT"     INSERT"
580 PRINT"     NEW FILE"
590 PRINT"     SAVE ON TAPE":PRINT"     LOAD FROM TAPE"
600 PRINT"     HARD COPY ON PRINTER"
610 PRINT"     DELETE":PRINT"     ALPHABETIZE":GOSUB 490
630 GOSUB 890:GOTO 210
640 CLS:PRINT@41,"SAVE ON TAPE"
650 GOSUB 920:IF A$<>"Y" THEN GOTO 150
660 CLS:PRINT@132,"PREPARE CASSETTE":PRINT
670 INPUT"     THEN PRESS <RETURN>";X:GOTO 1190
680 CLS:PRINT@35,"LOAD FROM TAPE"
690 GOSUB 920:IF A$<>"Y" THEN GOTO 150
700 INPUT"      FILE NAME";T$
710 CLS:PRINT@132,"PREPARE CASSETTE"
720 INPUT"     THEN PRESS <RETURN>";X:GOTO 1320
730 PRINT@384,"DELETE "R$;:INPUT A$:J=VAL(A$):GOSUB 500
740 FOR I=J TO N-1:IF S$(I)="-" AND S$(I+1)="-" THEN I=N-1
750 S$(I)=S$(I+1)
760 NEXT:S$(N)="-":GOTO 350
770 CLS:PRINT@40,"ALPHABETIZE "
775 PRINT@104,"ENTIRE FILE";:INPUT Z$:IF Z$="N" THEN GOTO 1500
780 GOSUB 1120:U=VAL(A$):IF U<0 OR U>N THEN GOTO 200
790 GOSUB 1130:K=0:FOR I=U TO KK
800 NN=I-1:I=KK
```

4 of 6.   Aug 84. V5(8).



```
810 NEXT I:IF K=1 THEN GOTO 830
820 NN=KK
830 I=0
840 J=U:IF I=NN-U THEN GOTO 350
850 IF J=NN-I THEN GOTO 880
860 IF S$(J)>S$(J+1) THEN TP$=S$(J):S$(J)=S$(J+1):S$(J+1)=TP$
870 J=J+1:GOTO 850
880 I=I+1:GOTO 840
890 K$=INKEY$:A$=INKEY$:IF A$="" THEN GOTO 890
900 RETURN
910 PRINT@456,;I:RETURN
920 PRINT@131,"ARE YOU SURE,(Y/N)";
930 INPUT A$
940 RETURN
950 CLS:PRINT@41,"HARD COPY"
960 PRINT@105,"ENTIRE FILE";:INPUT X$:IF X$="Y" THEN F=0:N=M
970 GOSUB 1120:F=VAL(A$):K=F:IF F<0 OR F>N THEN GOTO 950
980 PRINT@232,"ENDING "R$;:INPUT A$:M=VAL(A$)
1010 A$="":FOR I=1 TO INT(40-LEN(T$))/2:G$=CHR$(8):F$=CHR$(15)
1020 A$=A$+" ":NEXT:LPRINT CHR$(14)A$+T$+F$,CHR$(10),CHR$(10)
1030 LL=4:FOR I=K TO M
1040 IF S$(I)<>"-" THEN LPRINTS$(I)CHR$(10):LL=LL+2
1050 GOTO 1090
1060 A$="":FOR KK=7 TO LEN(S$(I))*6:A$=A$+CHR$(255):NEXT KK
1070 LPRINT"        "+G$+A$:LPRINTF$+"          "+S$(I)+G$
1080 LPRINTF$+"          "+G$+A$+F$,CHR$(10):LL=LL+4
1090 IF LL>59 THEN FOR NL=LL TO 72:LPRINTCHR$(10):NEXT:LPRINT""
1100 LL=3:NEXT:GOTO 150
1110 FOR J=0 TO N:S$(J)="-":NEXT:RETURN
1120 PRINT@168,"STARTING "R$;:INPUT A$:RETURN
1130 FOR I=N TO 0 STEP-1:IF S$(I)<>"-" THEN KK=I:I=0:GOTO 1150
1140 KK=I
1150 NEXT:RETURN
1160 FOR I=J+1 TO N:IF S$(I)="-" THEN KK=I:I=N:GOTO 1150
1170 KK=I
1180 NEXT:RETURN
1190 CLS:PRINT@196,"   SAVING     "
1200 FOR I=N TO 0 STEP-1:K=I:IF S$(I)<>"-" THEN I=0
1210 NEXT I:K=K+1
1220 PRINT#"VZ-DATA",K
1230 FOR I=0 TO K
1240 PRINT#"DATA",S$(I)
1250 GOSUB 910
1260 NEXT
1270 CLS:PRINT@194," COMPLETE    "
1280 PRINT@260,"PRESS <F> FOR FILE."
1290 K$=INKEY$
1300 I$=INKEY$:IF I$="" THEN GOTO 1300
1310 IF I$="F" THEN GOTO 350
1320 CLS
1330 PRINT@196,"    LOADING    "
1340 INPUT#"VZ-DATA",K
1350 FOR I=0 TO K
1360 INPUT#"DATA",S$(I)
```

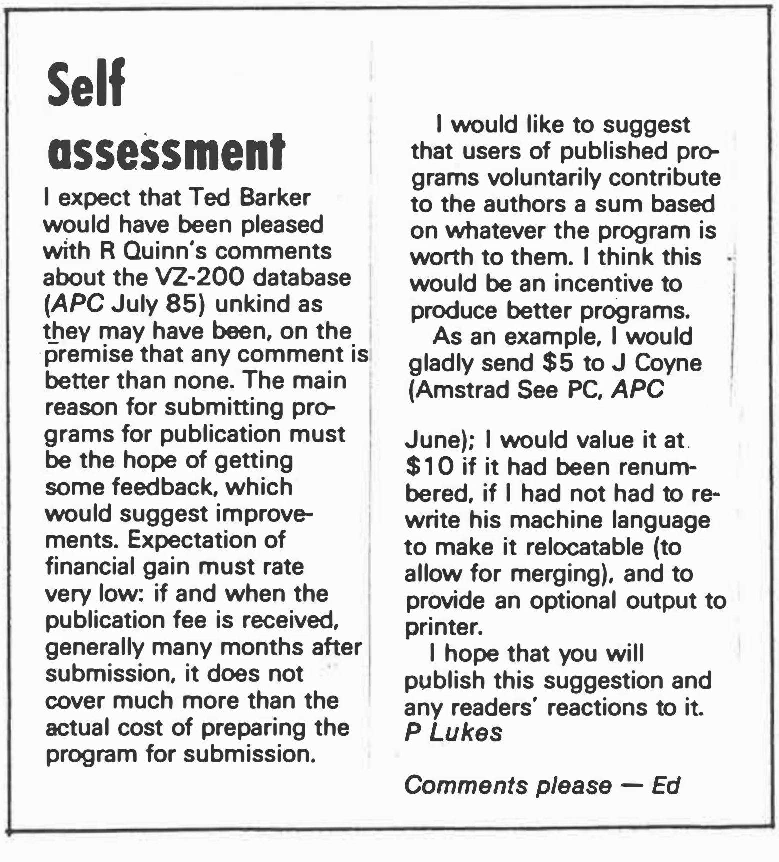
V5(8): Aug 84   5 of 6.



```
1370 GOSUB 910
1380 NEXT
1390 CLS:PRINT@194,"COMPLETE"
1400 PRINT@260,"PRESS <F> FOR FILE."
1410 K$=INKEY$
1420 I$=INKEY$:IF I$="" THEN GOTO 1420
1430 IF I$="F" THEN GOTO 350
1500 PRINT@168,"STARTING "R$;:INPUT A$
1510 PRINT@232,"ENDING   "R$;:INPUT B$
1520 U=VAL(A$):KK=VAL(B$)
1525 FORI=KKTO0STEP-1:IFS$(I)<>"-"THENKK=I:I=0:GOTO 1527
1526 KK=I
1527 NEXT
1530 K=0:FOR I=U TO KK
1540 NN=I-1:I=KK
1550 NEXT I:IF K=1 THEN GOTO 1570
1560 NN=KK
1570 I=0
1580 J=U:IF I=NN-U THEN GOTO 350
1590 IF J=NN-I THEN GOTO 1620
1600 IF S$(J)>S$(J+1) THEN TP$=S$(J):S$(J)=S$(J+1):S$(J+1)=TP$
1610 J=J+1:GOTO 1590
1620 I=I+1:GOTO 1580
```

A.P.C. Aug 84. V5(8).
172-177
6 of 6.

APC
Oct 85 6(10)
p 82-83.



Self assessment

I expect that Ted Barker would have been pleased with R Quinn's comments about the VZ-200 database (*APC* July 85) unkind as they may have been, on the premise that any comment is better than none. The main reason for submitting programs for publication must be the hope of getting some feedback, which would suggest improvements. Expectation of financial gain must rate very low: if and when the publication fee is received, generally many months after submission, it does not cover much more than the actual cost of preparing the program for submission.

I would like to suggest that users of published programs voluntarily contribute to the authors a sum based on whatever the program is worth to them. I think this would be an incentive to produce better programs.

As an example, I would gladly send $5 to J Coyne (Amstrad See PC, *APC* June); I would value it at $10 if it had been renumbered, if I had not had to rewrite his machine language to make it relocatable (to allow for merging), and to provide an optional output to printer.

I hope that you will publish this suggestion and any readers' reactions to it.

P Lukes

Comments please — Ed

Minicalc Spreadsheet

by Chris Stamboulidis

Minicalc is a spreadsheet program requiring a 24k VZ-200 system and an optional 80-column printer. It is based on a program written by Barry Spencer in the April 1984 issue of *Rainbow* magazine.

It features the following facilities:

- 9 x 43 cells on the spreadsheet
- tape storage and retrieval of data and functions
- dump to a printer
- column and row addition functions as well as +, –, *, /, ↑, absolute and integer functions
- non-destructive function view command to display formulae assigned to any cell

After you RUN the program, you will be greeted by the title screen and asked whether you require instructions. Hitting the 'Y' key will display the commands available and the syntax required to implement them. Note that when entering formulae for the cell functions, it is often necessary to use commas (such as when specifying cells). Unfortunately, the INPUT statement in Basic will not accept characters entered after a comma unless the entire input is enclosed in quotation marks. The result otherwise is an '? extra ignored' error message.

Hitting 'Y' will enter the spreadsheet proper and the upper left section will be displayed (there are 16 overlapping sections in all). The '>' prompt means that you may now enter a command.

To enter data, simply type Gx,y and hit (RETURN); x and y specify a cell x positions across and y positions down; this can be thought of as a GOTO command. When the 'G' cursor appears in the specified cell, you may enter numbers of strings up to 8 characters in length. From here, you may use the cursor control keys to move around the displayed section of the spreadsheet, entering data as you go. To get back to the command mode, simply hit (RETURN).

To enter formulae, use Fx,y where x and y specify the cell in which the result will be displayed. An 'F' cursor will appear in the cell specified and you will be prompted to type in the function into the two upper-most screen lines. Remember to use quote marks here, and hit (RETURN) when finished.

The four pre-set functions are:

- Ca,b gives the sum of the values appearing in the column from row a to row b
- Ra,b does the same in a row from column a to column b
- 'A' at the beginning or end of a formula takes the absolute value of the result
- 'I' at the beginning or end takes the integer value of the result. When specifying cells, use square brackets eg, [3,13].

To view a function in a particular cell, use Vx,y, hitting (RETURN) to get back to the command mode.

Movement from section to section within the spreadsheet is via the MU, MD, ML, MR commands (move up, down, left & right). MH returns you to the upper leftmost section.

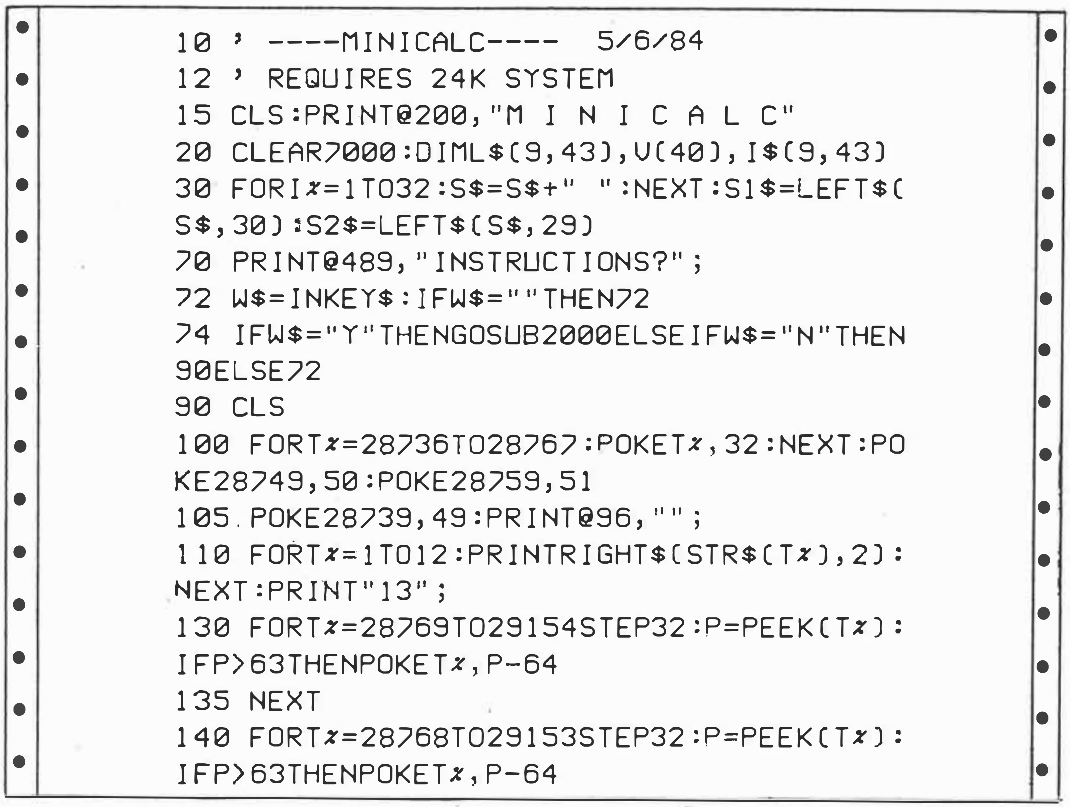
S and L are used for saving and loading from tape, and P enters the print mode.

U will update the entire spreadsheet, ie, all formulae will be calculated and the results displayed. Note that calculations occur from top to bottom, so that if a formula refers to a cell below it, you must update twice.

Finally, when typing in the program, the following characters should be entered in inverse text:

line 180	; "?"	CHR$(255)
line 450	; "G"	CHR$(199)
line 510	; "F"	CHR$(198)

See also APC 5(12): 214.
CORRECTION.

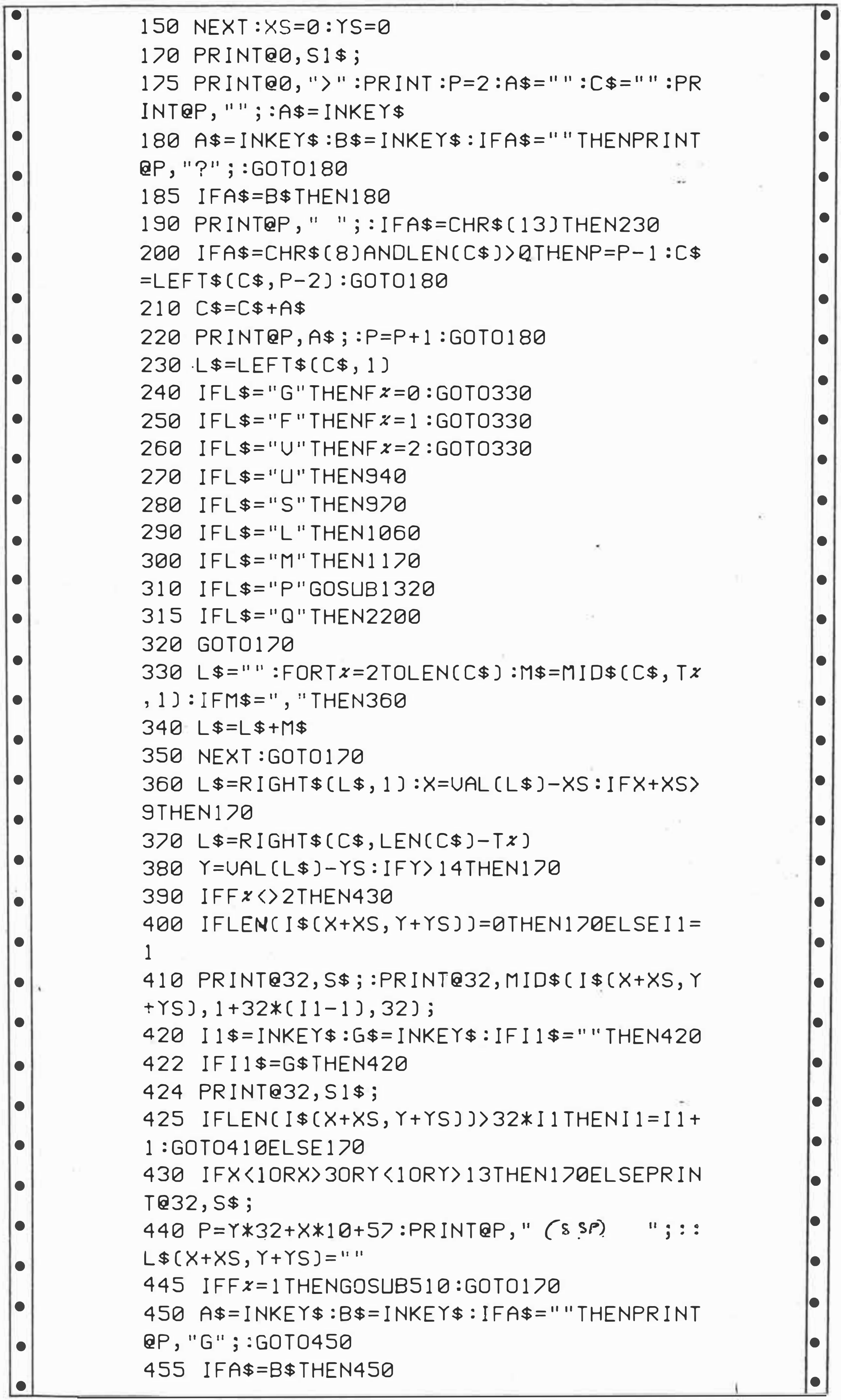


```
10 ' ----MINICALC----  5/6/84
12 ' REQUIRES 24K SYSTEM
15 CLS:PRINT@200,"M I N I C A L C"
20 CLEAR7000:DIML$(9,43),V(40),I$(9,43)
30 FORI%=1TO32:S$=S$+" ":NEXT:S1$=LEFT$(
S$,30):S2$=LEFT$(S$,29)
70 PRINT@489,"INSTRUCTIONS?";
72 W$=INKEY$:IFW$=""THEN72
74 IFW$="Y"THENGOSUB2000ELSEIFW$="N"THEN
90ELSE72
90 CLS
100 FORT%=28736TO28767:POKET%,32:NEXT:PO
KE28749,50:POKE28759,51
105 POKE28739,49:PRINT@96,"";
110 FORT%=1TO12:PRINTRIGHT$(STR$(T%),2):
NEXT:PRINT"13";
130 FORT%=28769TO29154STEP32:P=PEEK(T%):
IFP>63THENPOKET%,P-64
135 NEXT
140 FORT%=28768TO29153STEP32:P=PEEK(T%):
IFP>63THENPOKET%,P-64
```

Oct. 1984 V.5 (10)

p 126-130

1 of 5.

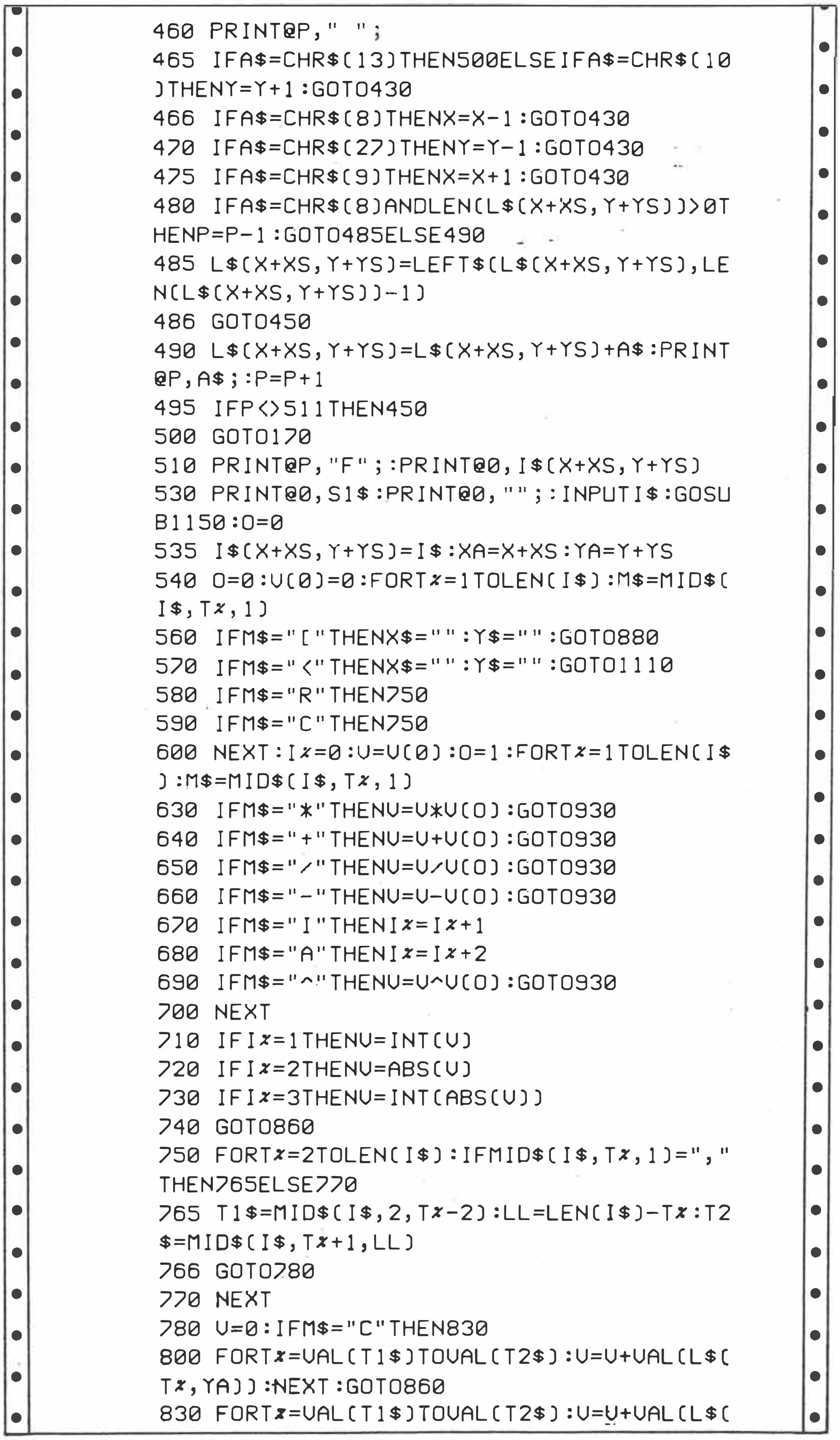


```
150 NEXT:XS=0:YS=0
170 PRINT@0,S1$;
175 PRINT@0,">":PRINT:P=2:A$="":C$="":PR
INT@P,"";:A$=INKEY$
180 A$=INKEY$:B$=INKEY$:IFA$=""THENPRINT
@P,"?";:GOTO180
185 IFA$=B$THEN180
190 PRINT@P," ";:IFA$=CHR$(13)THEN230
200 IFA$=CHR$(8)ANDLEN(C$)>0THENP=P-1:C$
=LEFT$(C$,P-2):GOTO180
210 C$=C$+A$
220 PRINT@P,A$;:P=P+1:GOTO180
230 L$=LEFT$(C$,1)
240 IFL$="G"THENF%=0:GOTO330
250 IFL$="F"THENF%=1:GOTO330
260 IFL$="V"THENF%=2:GOTO330
270 IFL$="U"THEN940
280 IFL$="S"THEN970
290 IFL$="L"THEN1060
300 IFL$="M"THEN1170
310 IFL$="P"GOSUB1320
315 IFL$="Q"THEN2200
320 GOTO170
330 L$="":FORT%=2TOLEN(C$):M$=MID$(C$,T%
,1):IFM$=","THEN360
340 L$=L$+M$
350 NEXT:GOTO170
360 L$=RIGHT$(L$,1):X=VAL(L$)-XS:IFX+XS>
9THEN170
370 L$=RIGHT$(C$,LEN(C$)-T%)
380 Y=VAL(L$)-YS:IFY>14THEN170
390 IFF%<>2THEN430
400 IFLEN(I$(X+XS,Y+YS))=0THEN170ELSEI1=
1
410 PRINT@32,S$;:PRINT@32,MID$(I$(X+XS,Y
+YS),1+32*(I1-1),32);
420 I1$=INKEY$:G$=INKEY$:IFI1$=""THEN420
422 IFI1$=G$THEN420
424 PRINT@32,S1$;
425 IFLEN(I$(X+XS,Y+YS))>32*I1THENI1=I1+
1:GOTO410ELSE170
430 IFX<1ORX>3ORY<1ORY>13THEN170ELSEPRIN
T@32,S$;
440 P=Y*32+X*10+57:PRINT@P," (S SP)   ";::
L$(X+XS,Y+YS)=""
445 IFF%=1THENGOSUB510:GOTO170
450 A$=INKEY$:B$=INKEY$:IFA$=""THENPRINT
@P,"G";:GOTO450
455 IFA$=B$THEN450
```

Oct 84   V 5(10)

126-130

2 of 5.

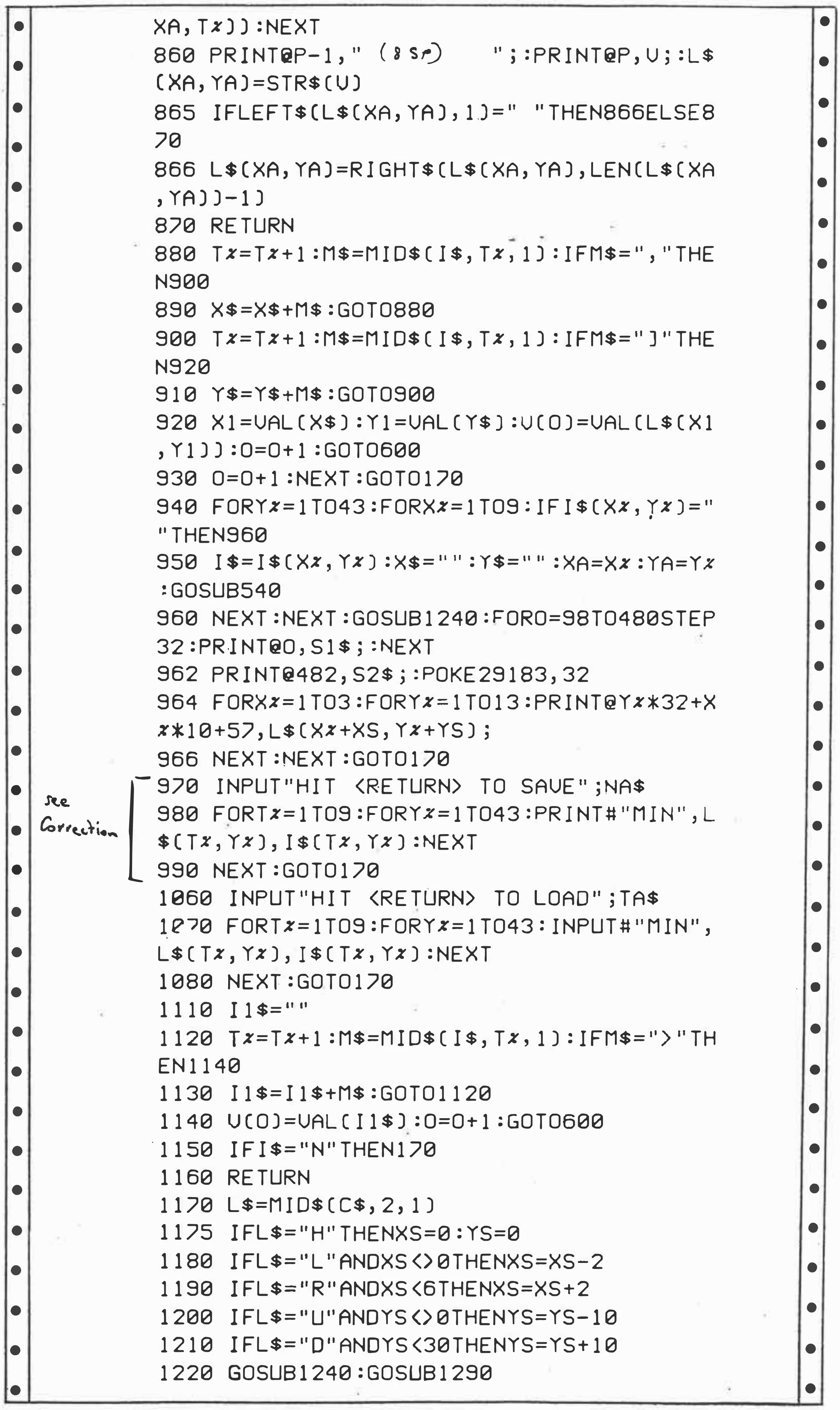


```
460 PRINT@P," ";
465 IFA$=CHR$(13)THEN500ELSEIFA$=CHR$(10
)THENY=Y+1:GOTO430
466 IFA$=CHR$(8)THENX=X-1:GOTO430
470 IFA$=CHR$(27)THENY=Y-1:GOTO430
475 IFA$=CHR$(9)THENX=X+1:GOTO430
480 IFA$=CHR$(8)ANDLEN(L$(X+XS,Y+YS))>0T
HENP=P-1:GOTO485ELSE490
485 L$(X+XS,Y+YS)=LEFT$(L$(X+XS,Y+YS),LE
N(L$(X+XS,Y+YS))-1)
486 GOTO450
490 L$(X+XS,Y+YS)=L$(X+XS,Y+YS)+A$:PRINT
@P,A$;:P=P+1
495 IFP<>511THEN450
500 GOTO170
510 PRINT@P,"F";:PRINT@0,I$(X+XS,Y+YS)
530 PRINT@0,S1$:PRINT@0,"";:INPUTI$:GOSU
B1150:O=0
535 I$(X+XS,Y+YS)=I$:XA=X+XS:YA=Y+YS
540 O=0:V(0)=0:FORT%=1TOLEN(I$):M$=MID$(
I$,T%,1)
560 IFM$="["THENX$="":Y$="":GOTO880
570 IFM$="<"THENX$="":Y$="":GOTO1110
580 IFM$="R"THEN750
590 IFM$="C"THEN750
600 NEXT:I%=0:V=V(0):O=1:FORT%=1TOLEN(I$
):M$=MID$(I$,T%,1)
630 IFM$="*"THENV=V*V(O):GOTO930
640 IFM$="+"THENV=V+V(O):GOTO930
650 IFM$="/"THENV=V/V(O):GOTO930
660 IFM$="-"THENV=V-V(O):GOTO930
670 IFM$="I"THENI%=I%+1
680 IFM$="A"THENI%=I%+2
690 IFM$="^"THENV=V^V(O):GOTO930
700 NEXT
710 IFI%=1THENV=INT(V)
720 IFI%=2THENV=ABS(V)
730 IFI%=3THENV=INT(ABS(V))
740 GOTO860
750 FORT%=2TOLEN(I$):IFMID$(I$,T%,1)=","
THEN765ELSE770
765 T1$=MID$(I$,2,T%-2):LL=LEN(I$)-T%:T2
$=MID$(I$,T%+1,LL)
766 GOTO780
770 NEXT
780 V=0:IFM$="C"THEN830
800 FORT%=VAL(T1$)TOVAL(T2$):V=V+VAL(L$(
T%,YA)):NEXT:GOTO860
830 FORT%=VAL(T1$)TOVAL(T2$):V=V+VAL(L$(
```

Oct. 84 V5(10).

126-130

3 of 5.



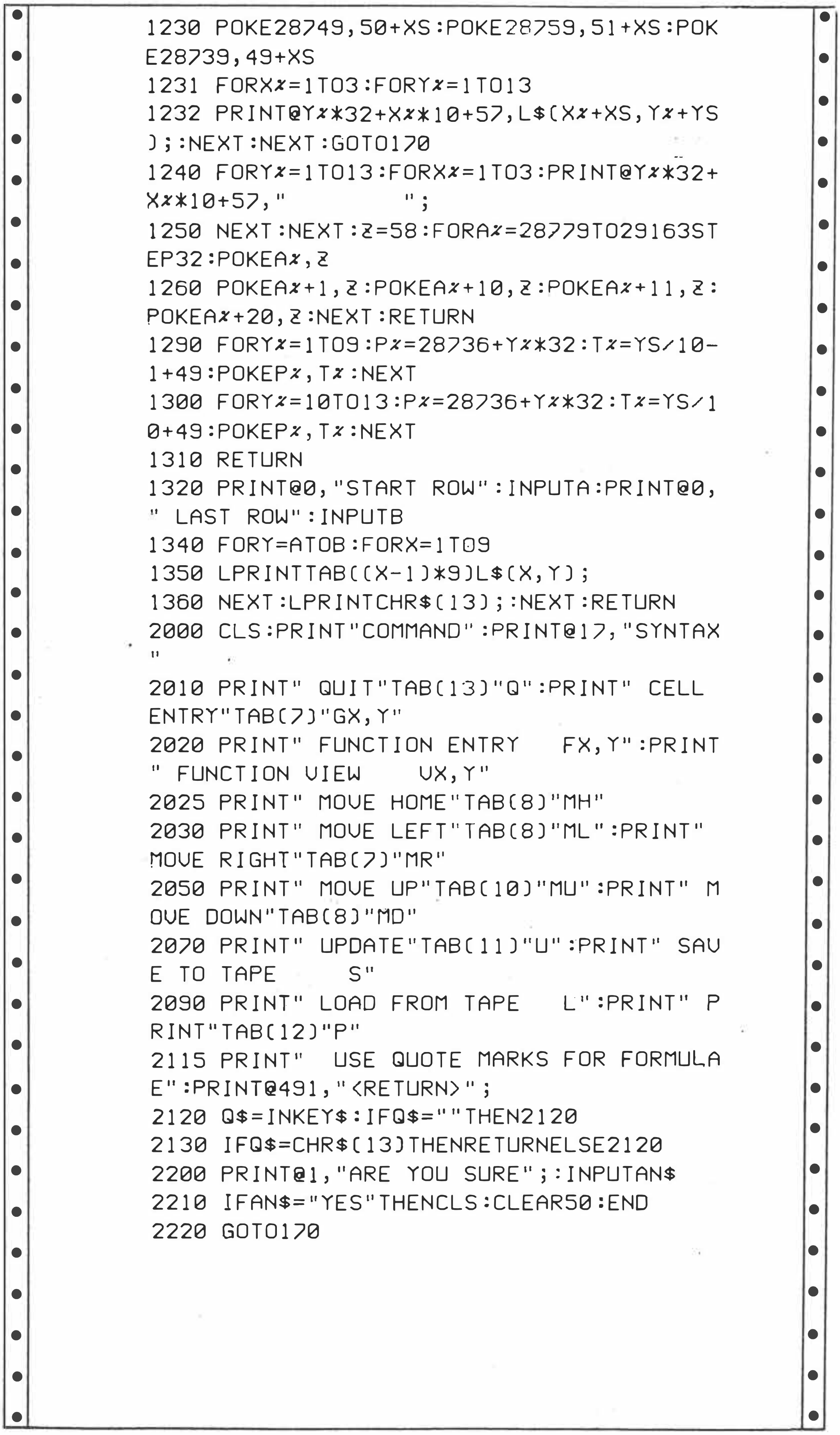
```
XA,T%)):NEXT
860 PRINT@P-1," (8 sp)    ";:PRINT@P,V;:L$
(XA,YA)=STR$(V)
865 IFLEFT$(L$(XA,YA),1)=" "THEN866ELSE8
70
866 L$(XA,YA)=RIGHT$(L$(XA,YA),LEN(L$(XA
,YA))-1)
870 RETURN
880 T%=T%+1:M$=MID$(I$,T%,1):IFM$=","THE
N900
890 X$=X$+M$:GOTO880
900 T%=T%+1:M$=MID$(I$,T%,1):IFM$=")"THE
N920
910 Y$=Y$+M$:GOTO900
920 X1=VAL(X$):Y1=VAL(Y$):V(O)=VAL(L$(X1
,Y1)):O=O+1:GOTO600
930 O=O+1:NEXT:GOTO170
940 FORY%=1TO43:FORX%=1TO9:IFI$(X%,Y%)="
"THEN960
950 I$=I$(X%,Y%):X$="":Y$="":XA=X%:YA=Y%
:GOSUB540
960 NEXT:NEXT:GOSUB1240:FORO=98TO480STEP
32:PRINT@O,S1$;:NEXT
962 PRINT@482,S2$;:POKE29183,32
964 FORX%=1TO3:FORY%=1TO13:PRINT@Y%*32+X
%*10+57,L$(X%+XS,Y%+YS);
966 NEXT:NEXT:GOTO170
970 INPUT"HIT <RETURN> TO SAVE";NA$
980 FORT%=1TO9:FORY%=1TO43:PRINT#"MIN",L
$(T%,Y%),I$(T%,Y%):NEXT
990 NEXT:GOTO170
1060 INPUT"HIT <RETURN> TO LOAD";TA$
1070 FORT%=1TO9:FORY%=1TO43:INPUT#"MIN",
L$(T%,Y%),I$(T%,Y%):NEXT
1080 NEXT:GOTO170
1110 I1$=""
1120 T%=T%+1:M$=MID$(I$,T%,1):IFM$=">"TH
EN1140
1130 I1$=I1$+M$:GOTO1120
1140 V(O)=VAL(I1$):O=O+1:GOTO600
1150 IFI$="N"THEN170
1160 RETURN
1170 L$=MID$(C$,2,1)
1175 IFL$="H"THENXS=0:YS=0
1180 IFL$="L"ANDXS<>0THENXS=XS-2
1190 IFL$="R"ANDXS<6THENXS=XS+2
1200 IFL$="U"ANDYS<>0THENYS=YS-10
1210 IFL$="D"ANDYS<30THENYS=YS+10
1220 GOSUB1240:GOSUB1290
```

see Correction

Oct 84 VS(10)

126-130

4 of 5.

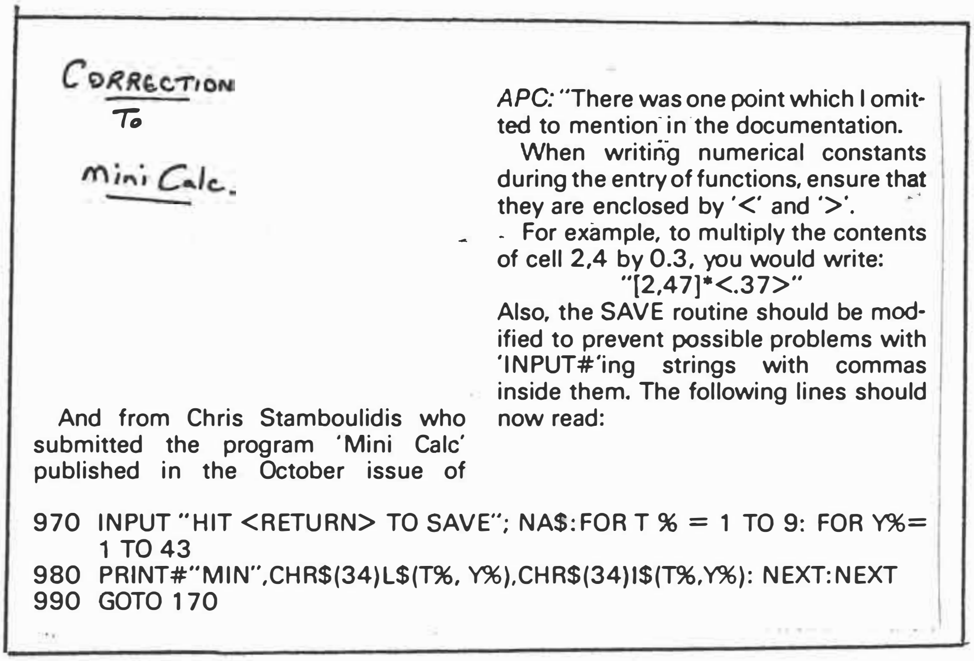


```
1230 POKE28749,50+XS:POKE28759,51+XS:POK
E28739,49+XS
1231 FORX%=1TO3:FORY%=1TO13
1232 PRINT@Y%*32+X%*10+57,L$(X%+XS,Y%+YS
);:NEXT:NEXT:GOTO170
1240 FORY%=1TO13:FORX%=1TO3:PRINT@Y%*32+
X%*10+57,"          ";
1250 NEXT:NEXT:Z=58:FORA%=28779TO29163ST
EP32:POKEA%,Z
1260 POKEA%+1,Z:POKEA%+10,Z:POKEA%+11,Z:
POKEA%+20,Z:NEXT:RETURN
1290 FORY%=1TO9:P%=28736+Y%*32:T%=YS/10-
1+49:POKEP%,T%:NEXT
1300 FORY%=10TO13:P%=28736+Y%*32:T%=YS/1
0+49:POKEP%,T%:NEXT
1310 RETURN
1320 PRINT@0,"START ROW":INPUTA:PRINT@0,
" LAST ROW":INPUTB
1340 FORY=ATOB:FORX=1TO9
1350 LPRINTTAB((X-1)*9)L$(X,Y);
1360 NEXT:LPRINTCHR$(13);:NEXT:RETURN
2000 CLS:PRINT"COMMAND":PRINT@17,"SYNTAX
"
2010 PRINT" QUIT"TAB(13)"Q":PRINT" CELL
ENTRY"TAB(7)"GX,Y"
2020 PRINT" FUNCTION ENTRY   FX,Y":PRINT
" FUNCTION VIEW    VX,Y"
2025 PRINT" MOVE HOME"TAB(8)"MH"
2030 PRINT" MOVE LEFT"TAB(8)"ML":PRINT"
MOVE RIGHT"TAB(7)"MR"
2050 PRINT" MOVE UP"TAB(10)"MU":PRINT" M
OVE DOWN"TAB(8)"MD"
2070 PRINT" UPDATE"TAB(11)"U":PRINT" SAV
E TO TAPE      S"
2090 PRINT" LOAD FROM TAPE   L":PRINT" P
RINT"TAB(12)"P"
2115 PRINT"  USE QUOTE MARKS FOR FORMULA
E":PRINT@491,"<RETURN>";
2120 Q$=INKEY$:IFQ$=""THEN2120
2130 IFQ$=CHR$(13)THENRETURNELSE2120
2200 PRINT@1,"ARE YOU SURE";:INPUTAN$
2210 IFAN$="YES"THENCLS:CLEAR50:END
2220 GOTO170
```

Oct. 84 V5(10).

126-130

5 of 5.



Correction To Mini Calc.

And from Chris Stamboulidis who submitted the program 'Mini Calc' published in the October issue of *APC:* "There was one point which I omitted to mention in the documentation.

When writing numerical constants during the entry of functions, ensure that they are enclosed by '<' and '>'.

For example, to multiply the contents of cell 2,4 by 0.3, you would write:

"[2,47]*<.37>"

Also, the SAVE routine should be modified to prevent possible problems with 'INPUT#'ing strings with commas inside them. The following lines should now read:

```
970  INPUT "HIT <RETURN> TO SAVE"; NA$:FOR T % = 1 TO 9: FOR Y%=
     1 TO 43
980  PRINT#"MIN",CHR$(34)L$(T%, Y%),CHR$(34)I$(T%,Y%): NEXT:NEXT
990  GOTO 170
```

APC 5(12) Dec. 84 p 214.

Micro Type

by G Browell

As a VZ-200 addict, and in response to the plaintive plea of S Hobson (Basic Understanding *APC* February 1985), the following program is sumbitted to make him happy and for all VeeZedders seeking a Basic word processor . . .

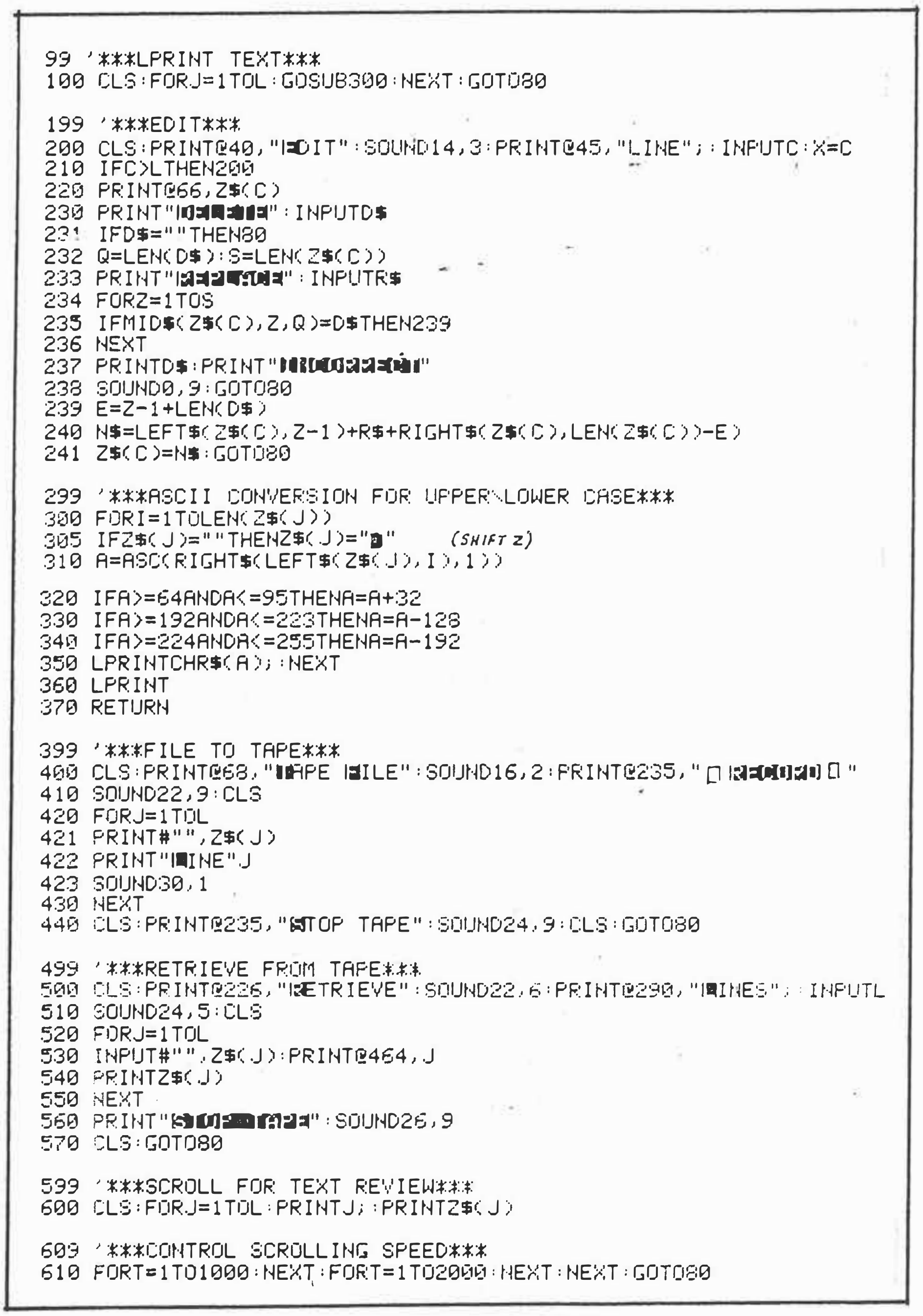
PS If S Hobson's idea is taken to extreme one could write a REM for just about every line — Hobson's Choice?!

NOTE: Use inverse for commas, colons and semi-colons. Use inverse for upper case. Use a blank graphic (Shift Z) to indent the first line of a paragraph. Scroll before Editing and before committing to printer. Should the program 'Break' just type GOTO80 <RETURN>.

```
0 '****************************
1 '* AD LIB VEEZED MICRO CLUB *
2 '*   <MICRO TYPE> PROGRAM   *
3 '*   FOR VZ-200 AND GP-100  *
4 '*   GORDON BROWELL  MAY 84 *
5 '****************************

10 CLS:CLEAR5000:PRINT@235,"[illegible]":SOUND16,6

11 '***ESTIMATE NUMBER OF LINES***
12 PRINT@235,"I■INES      ";:L=0:X=0:INPUTL:DIMZ$(L)
15 PRINT@299,"1. [illegible]":PRINT@331,"2. [illegible]"
16 PRINT@400,"";:INPUTR:IFR=2THEN510
30 FORJ=1TOL:CLS:PRINT@3,"I■INE"J;"OF";L

39 '***TYPING GUIDE***
40 PRINT@34,"[illegible]"
50 PRINT@128,"[illegible]":SOUND22,1

59 '***PRINT PREVIOUS LINE***
60 PRINT@162,"I■INE"J-1:PRINT@226,Z$(J-1)

69 '***ENTER TEXT***
70 PRINT@32,"■■":INPUTZ$(J):NEXT:CLS

79 '***MENU***
80 CLS:A$=INKEY$:A$=""
81 PRINT@40,"[illegible]";J-1:PRINT@104,"■CROLL"
82 PRINT@116,"I■DIT":PRINT@168,"I■RINT"
83 PRINT@232,"I■ILE":PRINT@296,"I■ETRIEVE":PRINT@360,"E■IT"
85 PRINT@421,"I[illegible]";X
90 A$=INKEY$:IFA$=""THEN90
91 A$=INKEY$:IFA$=""THEN90
92 IFA$="S"THEN600
93 IFA$="E"THEN200
94 IFA$="P"THEN100
95 IFA$="F"THEN400
96 IFA$="R"THEN500
97 IFA$="X"THENRUN
```

APC May 85 6(5) p 162-163

1 of 2.



```
99 '***LPRINT TEXT***
100 CLS:FORJ=1TOL:GOSUB300:NEXT:GOTO80

199 '***EDIT***
200 CLS:PRINT@40,"EDIT":SOUND14,3:PRINT@45,"LINE";:INPUTC:X=C
210 IFC>LTHEN200
220 PRINT@66,Z$(C)
230 PRINT"DELETE":INPUTD$
231 IFD$=""THEN80
232 Q=LEN(D$):S=LEN(Z$(C))
233 PRINT"REPLACE":INPUTR$
234 FORZ=1TOS
235 IFMID$(Z$(C),Z,Q)=D$THEN239
236 NEXT
237 PRINTD$:PRINT"INCORRECT"
238 SOUND0,9:GOTO80
239 E=Z-1+LEN(D$)
240 N$=LEFT$(Z$(C),Z-1)+R$+RIGHT$(Z$(C),LEN(Z$(C))-E)
241 Z$(C)=N$:GOTO80

299 '***ASCII CONVERSION FOR UPPER\LOWER CASE***
300 FORI=1TOLEN(Z$(J))
305 IFZ$(J)=""THENZ$(J)="■"      (SHIFT Z)
310 A=ASC(RIGHT$(LEFT$(Z$(J),I),1))

320 IFA>=64ANDA<=95THENA=A+32
330 IFA>=192ANDA<=223THENA=A-128
340 IFA>=224ANDA<=255THENA=A-192
350 LPRINTCHR$(A);:NEXT
360 LPRINT
370 RETURN

399 '***FILE TO TAPE***
400 CLS:PRINT@68,"TAPE FILE":SOUND16,2:PRINT@235,"□ RECORD □"
410 SOUND22,9:CLS
420 FORJ=1TOL
421 PRINT#"",Z$(J)
422 PRINT"LINE"J
423 SOUND30,1
430 NEXT
440 CLS:PRINT@235,"STOP TAPE":SOUND24,9:CLS:GOTO80

499 '***RETRIEVE FROM TAPE***
500 CLS:PRINT@226,"RETRIEVE":SOUND22,6:PRINT@290,"LINES";:INPUTL
510 SOUND24,5:CLS
520 FORJ=1TOL
530 INPUT#"",Z$(J):PRINT@464,J
540 PRINTZ$(J)
550 NEXT
560 PRINT"STOP TAPE":SOUND26,9
570 CLS:GOTO80

599 '***SCROLL FOR TEXT REVIEW***
600 CLS:FORJ=1TOL:PRINTJ;:PRINTZ$(J)

609 '***CONTROL SCROLLING SPEED***
610 FORT=1TO1000:NEXT:FORT=1TO2000:NEXT:NEXT:GOTO80
```

APC May 85 6(5) p162-163

2. of 2.

Database

by Robert Quinn

In the August 1984 issue of *APC* a database program for the VZ-200 was published, submitted by Ted Barker.

We have since received a letter from Robert Quinn of Wagga Wagga who insists that the program was a "sloppy, incompetent piece of coding, filled with errors, omissions and redundancies". Tough stuff, eh!

But Quinn has put his money where his mouth is and supplied his own version of the program for VZ owners. Over to Robert . . .

"When ENTERing or INSERTing data. If RETURN key is pressed without any data being placed in the line, Database will enter/insert a row of nine asterisks. This can be used as a handy divider, clearly separating one block of lines from another. The asterisks can be replaced with any characters you wish, to serve any purpose you can think of (the Basic lines in the listing are 430 and 460).

AUTO repeat of INPUT in same place on screen, with warning buzz, if you try to enter a line number longer than the maximum permitted, or if an END line number is less than a START line number. Fractional/negative entries for line numbers are rectified.

Where START and END line numbers are requested (HARD COPY, ALPHABETIZE). If you want the entire file operated on, simply press RETURN key twice. To START at line 0 and END at a certain line number, press RETURN, enter END line number and press RETURN. To START at a certain line number and work to end of file, enter START line number and press RETURN twice.

The ALPHABETIZE routine will display its progress on the screen: each successive line number will appear in bottom left of screen as the new start and the numbers of all the lines processed each time are rapidly displayed to the right.

PAGE CALL (press P). If RETURN is pressed without entering page number, Database takes this as page one. Then each successive press of P will display the next page; to restart PAGE CALL, simply press RETURN and P keys.

CATALOGUE has HALT and QUIT options. If there are too many catalogue lines to fit on one screen, CATALOGUE will display the excess lines in scroll fashion with pause between each line. Press any character key other than M to HALT CATALOGUE; again press any character other than M to CONTINUE CATALOGUE. Or press M key to QUIT CATALOGUE before CATALOGUE finishes its listing.

By altering/adding six lines of coding an INCREMENT LINE NUMBER is implemented. This allows you to enter successive lines of data without the irritation of having to press the E key again, then enter line number, then press RETURN. Simply press CTRL key and the ENTER LINE NUMBER will be incremented, allowing you to enter the data for the new line straight away. If you want to change the last line you entered/inserted/deleted, press X key and enter data.

Cassette SAVEing and LOADing is far too slow and unreliable for me to ever make use of the SAVE/LOAD routines of Database (imagine trying to SAVE/LOAD 399 lines of data, each line a separate data file on tape). Besides I have a disk drive and so will be designing DISK SAVE/LOAD routines. However, the cassette SAVE/LOAD routines could be sped up. One way would be to fit several Database lines in each data file to tape. I leave that to someone else to develop who has the interest/need.

Another option is to have the SAVE routine only record those Database lines which have data in them, skipping the empty lines until the next data-containing line is reached. Of course it would only speed up CSAVEing and CLOADing if there are lots of empty lines between lines of data. I have designed the modifications that should do the trick and appended the code to the end of the listing. If you wish to try it then enter those modified/additional lines to the program.

In the listing INVERSE characters are underlined. K=K+1: of line 1270 can be deleted (a harmless redundancy of BARKER that slipped through)."

A.P.C. Jul 85 V6(7).
p164-166.
1 of 2.

ROBERT QUINN
9 MARCONI STREET,
KOORINGAL: WAGGA WAGGA,
N.S.W. 2650

DATABASE: R.Q. VERSION

```
100 COLOR,1:CLEAR12000:F$="        ":E$="C,
P,E,I,N,S,L,H,D,A,M"
110 CLS:PRINT"FILE NAME.";:INPUTT$:IFT$=
""THENT$="NO NAME"
130 T$=LEFT$(T$,14):Y$="ARE YOU SURE (Y/
N)"
140 N=400:P=1:X=(N+1)/10:DIMS$(N):R$="LI
NE NUMBER"
145 FORJ=0TON:S$(J)="-":NEXT:N=N-1:CLS:G
OTO200

150 K=0:CLS:PRINT@0,"CATALOGUE  ";T$:PRI
NT:PRINT"PAGE"
160 FORJ=0TON:AK=ASC(S$(J)):IFAK=195ANDK
>13THENSOUND0,2
170 IFAK=195,PRINTINT(J/10+1);RIGHT$(S$(
J),LEN(S$(J))-1):K=K+1
175 A$=INKEY$:A$=INKEY$:IFA$<>""THENZ=NO
TZ:SOUND30,2
180 IFS$(J)="END"ORA$="M"THENJ=N:GOTO190
185 IFZ<0THEN175
190 NEXT:Z=0

195 REM CONTROL KEYS
200 GOSUB490
210 IFA$="C"THEN150
220 IFA$="P"THEN340ELSEIFA$="E"THEN410EL
SEIFA$="I"THEN440
230 IFA$="N"THEN520ELSEIFA$="S"THEN1210E
LSEIFA$="L"THEN1320
240 IFA$="H"THEN950ELSEIFA$="D"THEN730EL
SEIFA$="A"THEN1500
250 IFA$="M"THEN560
260 GOTO200


290 REM GENERAL LINE INPUT
300 A=0:PRINT@384,C$;R$;F$;:PRINT@402,""
;:INPUTA$
310 J=ABS(INT(VAL(A$))):IFA$=""THENA$="Z
"ELSEGOSUB500
320 IFJ>NTHENSOUND30,2:GOTO300ELSERETURN



330 REM PAGE CALL
340 PRINT@384,"ENTER PAGE NUMBER";:INPUT
A$:P=ABS(INT(VAL(A$)))
345 IFP=0THENP=1ELSEIFP>XTHENSOUND30,2:G
OTO340
350 CLS:PRINT"PAGE "P" "T$:PRINT
360 FORI=0TO9:L=(P-1)*10+I:PRINTL;S$(L):
NEXT
370 GOSUB490
380 IFA<>2THEN210
390 P=P+1:IFP>XTHENP=1
400 GOTO350
405 REM ENTER
410 C$=" ENTER ":GOSUB300
420 IFA>0THEN210
430 INPUTS$(J):IFS$(J)=""THENS$(J)="****
*****"
435 P=INT(J/10+1):GOTO350

438 REM INSERT
440 C$="INSERT ":GOSUB300:IFA>0THEN210
450 INPUTD$:CLS
460 IFD$=""THEND$="*********"
470 FORI=J+1TON:KK=I:IFS$(I)="-"THENI=N:
NEXTELSENEXT
480 FORI=KKTOJ+1STEP-1:S$(I)=S$(I-1):NEX
T
485 S$(J)=D$:P=INT(J/10+1):GOTO350
488 REM CHANGED YOUR MIND?
490 PRINT@490,E$;:GOSUB890
500 A=0:FORI=1TO11:IFASC(MID$(E$,I*2-1))
-128=ASC(A$)THENA=I:I=12
510 NEXT:RETURN
515 REM RERUN
520 CLS:PRINT@131,Y$;:INPUTA$:IFA$="Y"TH
ENRUNELSECLS:GOTO200

550 REM MENU
560 CLS:PRINT@6," MENU ":PRINT@68,"CATAL
OGUE"
570 PRINT"    PAGE CALL":PRINT"    ENTER
":PRINT"    INSERT"
580 PRINT"    NEW FILE"
590 PRINT"    SAVE ON TAPE":PRINT"    LO
AD FROM TAPE"
600 PRINT"    HARD COPY ON PRINTER"
610 PRINT"    DELETE":PRINT"    ALPHABET
IZE"
620 GOSUB490:GOTO210
```

```
1405 REM END OF SAVE/LOAD
1410 CLS:PRINT@448," COMPLETE ":SOUND30,
2;20,2;10,2;0,5:GOTO350



1490 REM ALPHABETIZE
1500 CLS:PRINT@40,"ALPHABETIZE":C$=" STA
RT ":GOSUB300
1505 IFA>0THEN1650ELSEK=J:PRINT@448,K
1510 C$=" END  ":GOSUB300:M=J:IFA>0THEN
1650ELSEIFA$="Z"THENM=N
1515 IFM<=KTHENSOUND30,2:GOTO1510
1520 FORI=MTOKSTEP-1:M=I:IFS$(I)<>"-"THE
NI=0:GOTO1540
1540 NEXT:I=0
1580 J=K:PRINT@448,I+K;"  ";:IFI=M-KTHEN
350
1590 IFJ=M-ITHEN1620
1600 IFS$(J)>S$(J+1)THENTP$=S$(J):S$(J)=
S$(J+1):S$(J+1)=TP$ .
1610 J=J+1:PRINT@455,J;"  ";:GOTO1590
1620 I=I+1:GOTO1580
1650 CLS:PRINT@490,E$:GOTO210
```

FOR INCREMENTING ENTER LINE NUMBER

```
145 FORJ=0TON:S$(J)="-":NEXT:N=N-1:J=0:C
LS:GOTO200

192 NEXT:Z=0:J=0

255 IFA$="X"THENSOUND10,1:PRINT@384,R$;J
:GOTO430

890 A$=INKEY$:A$=INKEY$:IFA$=CHR$(13)THE
NA$="Z"
895 IFPEEK(26877)=25.ANDJ<NTHENJ=J+1:CLS
:A$="X"
900 IFA$=""THEN890ELSERETURN
```

NEW SAVE/LOAD ROUTINES

```
1240 FORI=NTO0STEP-1:K=I:IFS$(I)<>"-"THE
NI=0:NEXTELSENEXT
1250 FORI=0TOK:IFS$(I)<>"-"THENF=F+1:NEX
TELSENEXT
1260 INPUT"    THEN PRESS <RETURN>";C
1270 CLS:PRINT@448," SAVING "
1275 PRINT#"VZ-DATA",F:SOUND0,3
1280 FORI=0TOK:IFS$(I)="-"THEN1300
1290 PRINT#"DATA",I,S$(I).PRINT@456," LI
NE ";I;:SOUND0,3
1300 NEXT:F=0:GOTO1410


1400 INPUT#"DATA",F,S$(F):PRINT@416,"LIN
E";F;:NEXT
```

```
720 REM DELETE
730 C$="DELETE ":GOSUB300
735 IFA>0THEN210
740 FORI=JTON:IFS$(I)="-"ANDS$(I+1)="-"T
HENI=N
750 S$(I)=S$(I+1)
760 NEXT:S$(N+1)="-":P=INT(J/10+1):GOTO3
50



880 REM WAITING
890 A$=INKEY$:A$=INKEY$:IFA$=""THEN890EL
SEIFA$=CHR$(13)THENA$="Z"
900 RETURN



940 REM HARD COPY
950 CLS:PRINT@41,"HARD COPY":C$=" START
":GOSUB300:IFA>0THEN210
960 K=J
970 C$=" END  ":GOSUB300:M=J:IFA>0THEN2
10ELSEIFA$="Z"THENM=N
980 IFM<KTHENSOUND30,2:GOTO970
990 LPRINT"        **** "T$:LPRINT
1030 FORI=KTOM:AK=ASC(LEFT$(S$(I),1))
1040 IFAK=195THENGOSUB1070:GOTO1060
1050 IFS$(I)<>"-"THENLPRINTS$(I)
1060 NEXT:K=0:GOTO200
1070 FORR=1TOLEN(S$(I)):AK=ASC(MID$(S$(I
),R,1))
1080 IFAK>223THENAK=AK-192ELSEIFAK>191TH
ENAK=AK-128
1090 LPRINTCHR$(AK);:NEXT:LPRINT:RETURN



1200 REM SAVE ON TAPE
1210 CLS:PRINT@41,"SAVE ON TAPE"
1220 PRINT@131,Y$;:INPUTA$:CLS:IFA$<>"Y"
THEN200
1230 PRINT@132,"PREPARE CASSETTE":PRINT
1240 INPUT"    THEN PRESS <RETURN>";C
1250 CLS:PRINT@448," SAVING "
1260 FORI=NTO0STEP-1:K=I:IFS$(I)<>"-"THE
NI=0
1270 NEXT:K=K+1:PRINT#"VZ-DATA",K
1280 FORI=0TOK
1290 PRINT#"DATA",S$(I):PRINT@456," LINE
 ";I
1300 NEXT:GOTO1410



1310 REM LOAD FROM TAPE
1320 CLS:PRINT@35,"LOAD FROM TAPE"
1330 PRINT@131,Y$;:INPUTA$:IFA$<>"Y"THEN
CLS:GOTO200
1340 INPUT"     FILE NAME";T$
1350 CLS:PRINT@132,"PREPARE CASSETTE"
1360 INPUT"    THEN PRESS <RETURN>";C
1380 CLS:INPUT#"VZ-DATA",K
1390 FORI=0TOK
1400 INPUT#"DATA",S$(I):PRINT@458,"LINE"
;I;:NEXT
```

"Database"

A.P.C. Jul 85. V6(7)

P 164-166

2 of 2.

VZ Wordprocessor VZ200/300

This word processor has two different modes, normal and repeat. On the repeat mode the computer will allow the user to repeat a letter on the keyboard by holding the key down.

The control keys are:
CTRL-E = Select Mode
CTRL-P = Print
CTRL-O = Clear Screen

G. Tunny
Gorokan, NSW

```
1 '******************
2 '*VZ-WORDPROCESSOR*
3 '*                *
4 '* BY  GLEN TUNNY *
5 '*(C)OPYRIGHT 1987*
6 '******************
8 GOSUB 1000
10 CLS
20 B=96:C=32:MD$="NORMAL"
25 PRINT@0,"MODE:"
30 A$=INKEY$:A$=INKEY$
40 POKE28672+C,B
45 PRINT@6,MD$
46 IF A$=CHR$(135) AND MD$="NORMAL"THENMD$="REPEAT":GOTO30
47 IF A$=CHR$(135) AND MD$="REPEAT"THENMD$="NORMAL":GOTO30
50 IF A$=CHR$(8)ANDC>32 THEN C=C-1:GOTO  150
60 IF A$=CHR$(9)ANDC<446 THEN C=C+1:GOTO  150
70 IF A$=CHR$(27)ANDC>63 THEN C=C-32:GOTO 150
80 IF A$=CHR$(10)ANDC<416 THEN C=C+32:GOTO 150
90 IF A$=CHR$(13) THEN 500
100 IF A$=CHR$(178) THEN 600
110 IF A$=CHR$(140) THEN 10
120 IF A$="" THEN B$="":GOTO 150
130 PRINT@C,A$:IF MD$="NORMAL" AND A$=B$ THEN 150
140 B$=A$:C=C+1:SOUND10,1
150 B=PEEK(28672+C):POKE28672+C,32:IFINKEY$="",FORI=1TO45:NEXTI
180 GOTO 30
500 W=INT(C/32):W=W+1:W=W*32:C=W
505 IF C>448 THENC=C-32
510 GOTO 150
600 PRINT@0,"                    ":A=INP(12)
610 IF A=13 THEN 630
620 GOTO 700
630 PRINT@8,"<PRINTER ERROR>":REM [INVERSE]
640 SOUND23,1
650 PRINT@8,"
660 SOUND 27,1
670 A=INP(12)
680 IF A=13 THEN 630
690 GOTO 600
700 COPY
710 GOTO 30
720 NEXT
800 FORI=1TO15
810 K$=INKEY$:K$=INKEY$
820 IF K$="" THEN NEXTI
830 RETURN
1000 CLS
1010 REM
1020 A$="VZ-WORDPROCESSOR"
1030 B$=" BY GLEN TUNNY "
1040 C$="(C)OPYRIGHT 1987"
1045 D$="<<HIT ANY KEY>>"
1050 FORI=1TOLEN(A$)
1060 PRINT@40,RIGHT$(A$,I)
1070 POKE26624,1:POKE26624,0
1080 NEXTI
1090 FORI=1TOLEN(B$)
1100 PRINT@72,RIGHT$(B$,I)
1110 POKE 26624,1:POKE26624,0
1111 NEXTI
1120 FORI=1TOLEN(C$)
1130 PRINT@104,RIGHT$(C$,I)
1140 POKE 26624,1:POKE26624,0
1150 NEXTI
1160 FORI=1TOLEN(D$)
1170 PRINT@136,RIGHT$(D$,I)
1180 POKE26624,1:POKE26624,0
1190 NEXTI
1200 FORI=1TO500:NEXTI
1210 A$=INKEY$:A$=INKEY$
1220 IF A$="" THEN 1210
1230 RETURN
```

That's All
Folks!