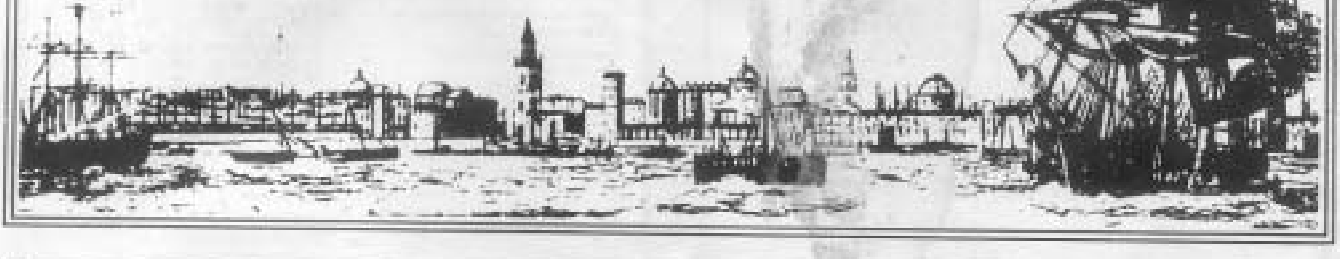
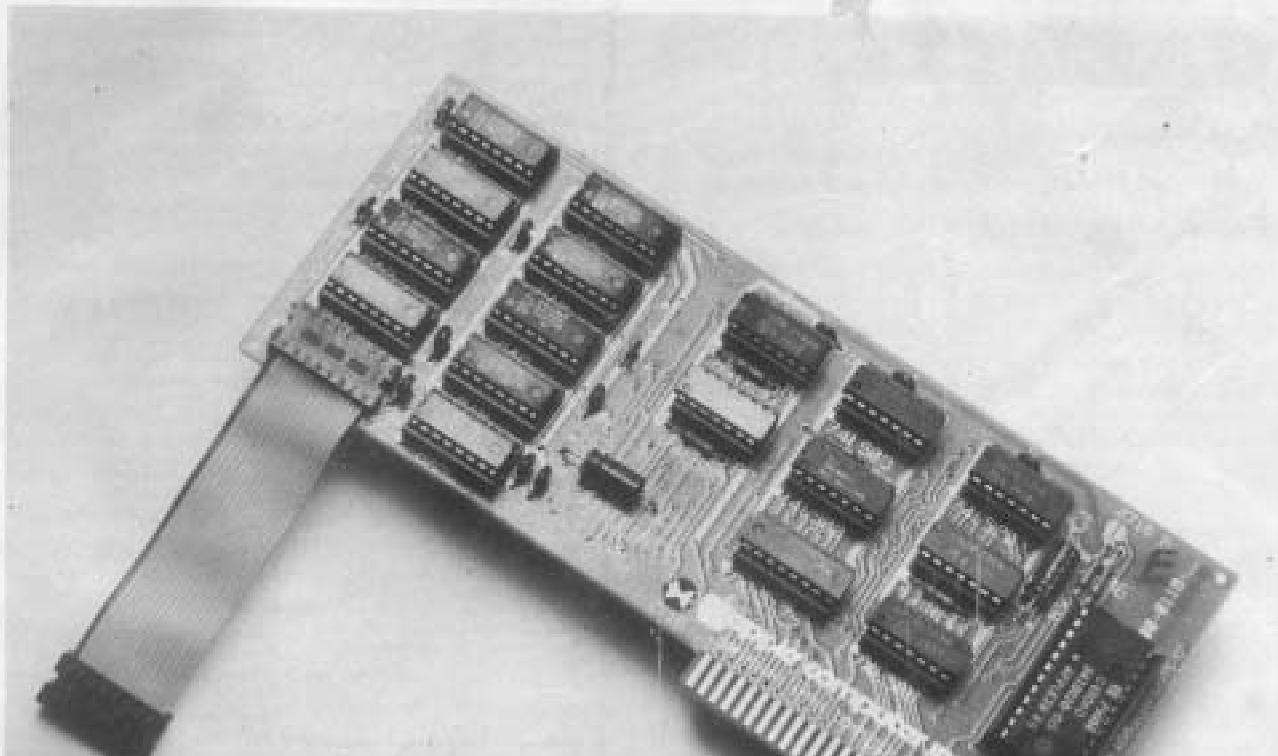
November 1979

First Edition 50p

LIVERPOOL SOFTWARE GAZETTE



APPLE PASCAL



First Impressions

LIVERPOOL SOFTWARE GAZETTE

Editor: Carl Phillips

Editorial assistant: Marie Beard

Contributing Editors; John Stout, Dr Martin Beer

Subscriptions: Christine Crofton

THE LIVERPOOL SOFTWARE GAZETTE is published bi-monthly by Microdigital, 25 Brunswick Street, Liverpool L2 0PJ. Tel: 051-227 2535.

SUBSCRIPTIONS: Within Great Britain, £6.00 per 12 issues. Individual copies, by post, 60p. Please address all subscription correspondence to Christine Crofton, Liverpool Software Gazette, 14 Castle Street, Liverpool L2 0TA.

ADVERTISING RATES: Full page: (17.5 cm x 24 cm) £180; Half page: (upright: 8.5 cm x 24 cm) £95, (landscape: (17.5 cm x 11.75 cm) £95; Quarter page: (8.5 cm x 11.75 cm) £52.

Editorial

WHAT?? Another microcomputer magazine!

This is the first edition of the 'Liverpool Software Gazette' Microdigital's contribution to the already frightening number of Microcomputer-related journals . . . But we like to think that we are different. Our aim is to try and provide as much information as possible for the Microcomputer user—in a presentable format for easy digestion. Something of a market gap exists in the need to furnish machine-specific information for users of personal systems. In our experience the average Microcomputer owner rapidly attains a standard of competance whereby the innumerable 'Beginning Basic', 'Hunt the Zombie, Snark' etc. articles of the monthly 'glossies' fail to interest or attract.

Since Microdigital staff are responsible for much of this magazine we make no particular claims of objectivity or independence. Nevertheless we will try and maintain a balanced viewpoint, with no particular emphasis on any machine.

Contributions and letters are particularly welcome—we look forward to hearing your comments, criticisms, suggestions, praise? etc.

May I take this opportunity to thank all those people who contributed articles and information for the first edition.

C. Phillips

DISCLAIMER

'All the information is the magazine has been thoroughly debugged and tested. However, no guarantees are made as to its truth or validity'.

Dear Reader,

WELCOME to our comic. For sometime now we have thought that a medium was needed for the interchange of knowledge between microcomputer users; this we hope is it. In our first issue we have attempted to set a high technical standard for content, this standard will be maintained in future issues.

These future issues will be initially bi-monthly, and we hope, monthly.

We welcome contributions, with correspondence and comment on all microcomputer Software related subjects; of course we will only know when we are going wrong when you tell us.

May I take this opportunity to thank all those whose labours have made this venture possible.

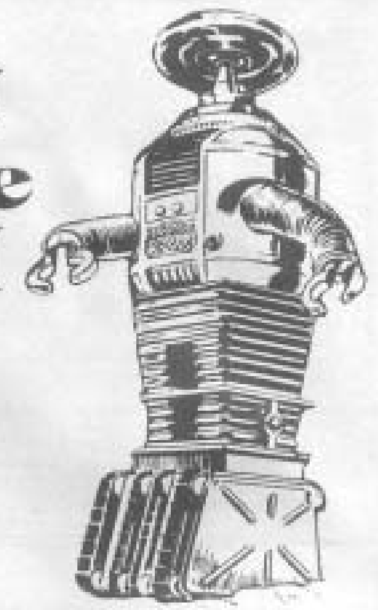
B. Everiss

BRUCE EVERISS

Contents

SARGON meets the NASCOM

J.Haigh



THE Sargon chess program, written by Dan and Kathe Spracklen, is published in Z80 assembly language by the Hayden Book Company. The assembled program can be run on a Nascom 1 with a single 8K RAM card, although the assembly language version, using the patches detailed below but with all remarks deleted, occupies 27K. Much of the program can be assembled as published, but all sections associated with input or output have to be adapted to the Nascom monitor routines.

The listing was produced on the TDL macro-assembler, which does not use the standard Z80 mnemonics and although a conversion table is provided at the back of the book it is very easy to make mistakes until you become familiar with the TDL codes. Several points are not covered in the table, for example the use of the full stop to denote the current address, and the assembler directives LOC, = , and .BYTE which replace ORG, EQU and DEFB. Thus if you want the program to run from £1000 to £3000 the beginning of the tables section translates to:

```
START   EQU £1ØØØ
        ORG START+£8Ø
TBASE   EQU START+£1ØØ
DIRECT  EQU Ø-TBASE
        DEFB 9, 11, -11, -9
```

The program can be assembled as published up to the end of subroutine BOOK; subroutines BITASN, ASNTBI, VALMOV, ROYALT, DIVIDE, MLTPLY and EXECMV are also unchanged. The graphics data base, the four subroutines which tabulate the moves (TBPLCL, TBCPCL, TBPLMV and TBCPMV), and subroutines PGIFND and MATED are omitted, which leaves fifteen sections of the program to be modified. The modifications include two patches to eliminate minor bugs from the original program. The first occurs if the computer is in stalemate; having scanned all its possible moves it selects the best one—and moves into check. This is cured by the addition of CALL INCHK after the machine has made its move on the internal board; if it finds that it has moved into check it displays the last legal position and prints 'Stalemate'.

The second bug appears when a board position has been set up for analysis. If the variable MOVENO is equal to one the computer will make its 'book' opening (P-K4 or P-Q4) without testing its legality. As the relevant square may be occupied by any piece, or may be empty, this can result in very strange moves. This idiosyncrasy is eliminated by initialising MOVENO to two in subroutine ANALYS.

A serious defect in the implementation of Sargon on a standard Nascom 1 is the lack of graphics. The best can be done to display the board is to use characters £ØØ and £7F for white and black squares, and to represent the pieces by letters, upper case for white and lower case for black. Bits and P.C.s of Wakefield sell a graphics kit which uses a 2708 EPROM to provide Nascom with 64 extra characters and their reverse-field equivalents. A set of chess pieces is one of the options available and it greatly improves the appearance of the display.

The most interesting stage begins when the program is assembled and running—there are over 800 unused bytes between the end of subroutine BOOK and the start of the standard messages and this space can be used for your own modifications. For example, you can store up to ten board positions here so that once a position is set up for analysis it can be recalled as required. An alternative driver routine can be added to enable two human players to play each other, or you can have the computer play itself at different levels of look-ahead. A useful addition is an internal store for moves with a simple routine to display the moves at a controllable rate, which gives you a system of the Tolinka type.

On a Nascom running at 2 Mhz typical response times at the six possible look-ahead levels are: 1-10 secs., 2-1 min., 3-10 mins., 4-1 hour, 5-6 hours, 6-24 hours; however, the times can vary quite widely and the figures given should only be taken as a rough guide.

Modifications to Sargon for Nascom 1

Graphics Data Base Omitted.

Standard Messages TITLE3 and BLANKR are omitted. The move list messages (MVENUM, MVEMSG, 0.0, 0.0.0, CKMSG, P.PEP), TITLE1, TITLE 2 and PCS are unchanged. SPACE is a string of five space characters (£20) and TITLE4 consists of thirteen space characters. The remaining messages should be rewritten as subroutines by inserting RST 40 (£EF) in front of the message and DEFB 0, £C9 at the end; INVAL1 and INVAL2 can be written as a single message. MTPL is a label within MTMSG which is used for the entry of the number of moves to checkmate; thus MTMSG is assembled as

```
MTMSG     RST 40
          DEFM /CHECKMATE IN /
MTPL      DEFB £32, £21, 0, £C9
```

Vairables This section is unchanged; INDXER is no longer needed for the graphics data base, but it is used for storing the current position of the move list.

Macro Definitions The macros are omitted and the space is used for the subroutine which erases the machine prompts and the subroutines which print the move list.

```
21 8A Ø8      CLRLIN   LD HL, £B8A       ;start of bottom line
22 18 ØC               LD (£C18), HL     ;reset the cursor
Ø6 3Ø                  LD B, 48          ;line length
36 2Ø                  LD (HL), 32       ;space character
2C                     INC L
1Ø FB                  DJNZ -3
C9                     RET

ED 5B 6Ø 19   PRTBLK   LD DE, (INDXER)   ;current list position
1C                     INC E             ;space at start
Ø1 Ø5 ØØ               LD BC, Ø5         ;message length
ED BØ                  LDIR              ;copy (HL) to (DE)
1C                     INC E             ;space at end
EB                     EX DE, HL         ;list position in HL
7D                     LD A, L
E6 3F                  AND £3F
FE 1B                  CP 1B             ;new line needed?
38 2D                  JR C, PR3-Ø       ;if not, jump
3A 64 19               LD A, (LINECT)    ;get line count
3C                     INC A             ;increment
FE ØE                  CP 14             ;line 14?
2Ø 1E                  JR NZ, PR1-Ø      ;if not, jump
3E ØD                  LD A, 13          ;reset line count
11 ØA Ø8               LD DE, £8ØA       ;top line
21 4A Ø8               LD HL, £84A       ;second line
Ø1 11 ØØ      PRØ1     LD BC, 17         ;line length
ED BØ                  LDIR              ;copy up one line
Ø1 2F ØØ               LD BC, £2F
Ø9                     ADD HL, BC        ;next lower line
EB                     EX DE, HL
Ø9                     ADD HL, DE        ;next upper line
EB                     EX DE, HL
```

```
3D                  DEC A               ;last line?
20 F1               JR NZ, PR01-0       ;if not, recycle
21 0A 0B            LD HL, £B0A
22 60 19            LD (INDXER), HL     ;reset list position
C9                  RET
11 2F 00     PR1    LD DE, £2F
19                  ADD HL, DE          ;next line
32 64 19     PR2    LD (LINECT), A      ;store line count
22 60 19     PRC    LD (INDXER), HL     ;store list position
C9                  RET

21 8A 28     PRTMVN LD HL, MVENUM
ED 5B 60 19         LD DE, (INDXER)     ;get list position
01 03 00            LD BC, 3
ED B0               LDIR                ;copy move number
ED 53 60 29         LD (INDXER), DE     ;store list position
C9                  RET
```

Main Program Driver The first five lines of this section are changed to;

```
31 FF 92     DRIVER LD SP, STACK        ;set stack pointer
EF 1E 00            DEFB £EF, £1E, 00   ;clear screen
CD 00 28            CALL GRTTNG         ;prompt
CD 9E 2C     DRIV01 CALL CHARTN         ;get answer
CD 65 29            CALL CLRLIN         ;erase line
```

After CALL INITBD the value of INDXER must be initialised by the insertion of;

```
21 0A 0B            LD HL, £B0A
22 60 29            LD (INDXER), HL
```

The twenty-one lines between CALL DSPBRD and DR0C are replaced by;

```
21 1E 29            LD HL, TITLE4       ;title address
11 CD 0B            LD DE, £BCD         ;title screen position
01 0D 00            LD BC, 13           ;title length
ED B0               LDIR                ;copy
CD B7 29     DRIV04 CALL PRTMVN         ;print move number
3A 20 10            LD A, (KOLOR)       ;computer's colour
A7                  AND A               ;is it white?
20 08               JR NZ, DR08-0       ;if not, jump
CD DC 2A            CALL CPTRMV         ;computer's move
CD C3 2B            CALL PLYRMV         ;player's move
18 06               JR, DR0C-0
CD C3 2B     DR08   CALL PLYRMV         ;player's move
CD DC 2A            CALL CPTRMV         ;computer's move
21 8C 28     DR0C   LD HL, MVENUM+2
```

The rest of this section is unchanged.

Interrogation for Fly and Colour

```
CD 65 29     INTERR CALL CLRLIN         ;clear bottom line
CD 4B 28            CALL CLRMSG         ;request colour choice
CD 9E 2C            CALL CHARTN         ;accept answer
FE 57               CP £57              ;is it W?
28 15               JR Z, IN04-0        ;if white, jump
97                  SUB A A             ;set computer's colour
32 20 10            LD (KOLOR), A       ;to white
21 79 28            LD HL, TITLE1       ;prepare titles
11 1E 29            LD DE, TITLE4
01 06 00            LD BC, 6
ED B0               LDIR
1C                  INC E               ;space between columns
21 7F 28            LD HL, TITLE2
18 14               JR, IN08-0
3E 80        IN04   LD A, £80           ;set computer's colour
32 20 10            LD (KOLOR), A       ;to black
21 7F 28            LD HL, TITLE2       ;prepare titles
11 1E 29            LD DE, TITLE4
01 06 00            LD BC, 6
ED B0               LDIR
```

```
1C                        INC E            ;space between columns
21 79 28                  LD HL, TITLE1
Ø1 Ø6 ØØ        INØ8      LD BC, 6
ED BØ                     LDIR
CD 65 29                  CALL CLRLIN      ;erase colour choice
CD Ø3 29                  CALL PLYDEP      ;request look-ahead
CD 9E 2C                  CALL CHARTR      ;accept answer
CD 65 29                  CALL CLRLIN      ;erase
21 27 1Ø                  LD HL, PLYMAX
```

The remaining eight lines are unaltered.

Computer Move Routine A patch is added to handle stalemate:

```
CD 9Ø 21        CPØC      CALL MOVE        ;make move
CD 97 1D                  CALL INCHK       ;computer in check?
A7                        AND A
28 17                     JR Z, CPS-Ø      ;if not, jump
CD FE 21                  CALL UNMOVE      ;delete illegal move
FF                        RST 4Ø           ;print string call
2Ø 53 54 41 4C 45 4D 41 54 45  DEFM / STALEMATE/
ØØ                        NOP              ;end of string
CD 97 2B                  CALL FMØ8        ;end of game
CD 98 2E        CPS       CALL EXECMV
```

The lines which tabulate the computer's moves, starting at PRTBLK MVEMSG, 5 and ending at CP1C, are changed to:

```
21 8D 28                  LD HL, MVEMSG    ;address of move
18 15                     JR, CP1C-Ø
CB 48           CP1Ø      BIT 1, B         ;King side castle?
28 Ø5                     JR Z, 7          ;if not, jump
21 92 28                  LD HL, O.O       ;address of O-O
18 ØC                     JR, CP1C-Ø
CB 5Ø                     BIT 2, B         ;Queen side castle?
28 Ø5                     JR Z, 7          ;if not, jump
21 97 28                  LD HL, O.O.O     ;address of O-O-O
18 Ø3                     JR, CP1C-Ø
21 3F 29                  LD HL, P.PEP     ;address of PxPep
CD 73 29        CP1C      CALL PRTBLK      ;note moved label
3A 21 1Ø                  LD A, (COLOR)
```

The next eight lines are unchanged; the macro CARRET is deleted and the program continues:

```
3A 54 11                  LD A, (SCORE+1)
FE FF                     CP £FF
21 9C 28                  LD HL, CKMSG     ;print CHECK
CD 73 29                  CALL PRTBLK
2A 6Ø 29                  LD HL, (INDEX)   ;get list position
2C                        INC L            ;increment over
2C                        INC L            ;move number
2C                        INC L            ;column
22 6Ø 29                  LD (INDEX), HL   ;store list position
21 85 28                  LD HL, SPACE     ;increment over
CD 73 29                  CALL PRTBLK      ;player's column
3A 54 11        CP24      LD A, (SCORE+1)  ;check for mate
FE FF                     CP £FF           ;player mated?
CØ                        RET NZ           ;if not, return
ØE ØØ                     LD C, Ø          ;player mate flag
CD 77 2B                  CALL FCDMAT      ;full checkmate?
C9                        RET              ;return
```

Forced Mate Handling At line nine CARRET is deleted and from this point the program is changed to:

```
21 9C 28                  LD HL, CKMSG     ;address of CHECK
CD 73 29                  CALL PRTBLK      ;print
CD 65 29                  CALL CLRLIN      ;clear bottom line
CD B9 28                  CALL UWIN        ;output YOU WIN
18 Ø3                     JR, FMØ8-Ø
CD C4 28        FMØ4      CALL IWIN        ;output I WIN
```

```
E1          FMØ8   POP HL              ;remove return
E1                 POP HL              ;addresses
CD 9E 2C           CALL CHARTR         ;any character restarts
EF 1E ØØ    FMØ9   DEFB £EF, £1E, ØØ   ;clear screen
CD D9 28           CALL AGAIN
C3 Ø9 2A           JP, DRIVØ1
CB 41       FMØC   BIT Ø, C            ;who has mate?
CØ                 RET NZ              ;return if player
C6 3Ø              ADD A, £3Ø          ;no. of moves to Ascii
32 AF 28           LD (MTPL), A        ;store in MTMSG
CD 65 29           CALL CLRLIN         ;clear bottom line
CD A1 28           CALL MTMSG          ;output CHECKMATE IN X
C9                 RET
```

<u>Tabulation routines</u> The four subroutines which tabulate the moves are omitted.

<u>Player's Move Analysis</u> Line 2 is changed to CP £2E (a full stop is entered to resign). After CALL EXECMV the program continues:

```
CD 65 29           CALL CLRLIN         ;erase move
3A 8D 28           LD A, (MVEMSG)      ;'from' position
4F                 LD C, A             ;save in C
3A 8E 28           LD A, (MVEMSG+1)    ;'to' position
57                 LD D, A             ;store in D
CD B4 2B           CALL BITASN         ;algebraic 'to' position
22 9Ø 28           LD (MVEMSG+3), HL   ;put in MVEMSG
51                 LD D, C             ;transfer 'from' to D
CD B4 2B           CALL BITASN         ;algebraic 'from'
22 8D 28           LD (MVEMSG), HL     ;put in MVEMSG
21 8D 28           LD HL, MVEMSG       ;address of message
CD 73 29           CALL PRTBLK         ;print
C9                 RET
CD 65 29    PLØ8   CALL CLRLIN         ;erase move
CD 44 29           CALL INVAL          ;output INVALID
EF 2Ø ØØ           DEFB £EF, £2Ø, ØØ   ;space after message
C3 C3 2B           JP, PLYRMV          ;return for next attempt
```

<u>Accept Input Character</u>

```
CD 3E ØØ    CHARTR CALL £3E            ;address of CHIN
FE 1F              CP £1F              ;newline
C8                 RET Z               ;if so, return
FE 1D              CP £1D              ;backspace?
C8                 RET Z               ;if so, return
CD 3B Ø1           CALL £13B           ;CRT routine
E6 7F              AND £7F
```

The remaining lines are unaltered.

Subroutines PCIFND and MATED are omitted.

<u>Set up Position for Analysis</u>

```
CD 65 29    ANALYS CALL CLRLIN         ;erase
CD 23 2B           CALL ANAMSG         ;prompt
CD 9E 2C           CALL CHARTR         ;accept answer
FE 4E              CP £4E              ;is it N?
CA ØØ ØØ           JP Z, Ø             ;return to monitor
21 ØA Ø8           LD HL, £Ø8ØA        ;top left of move list
22 6Ø 29           LD (INDXER), HL     ;initialise
3E Ø1              LD A, 1
32 64 29           LD (LINECT), A      ;initialise line count
3C                 INC A               ;set MOVENO to 2
32 26 1Ø           LD (MOVENO), A      ;to avoid book opening
CD 65 29    ANØ4   CALL CLRLIN         ;erase
CD CE 2D           CALL DSPBRD         ;display board
3E 15              LD A, 21            ;first board index
```

After CALL CHARTR (line 19) a full stop (£2E) is used to terminate the setting-up process, replacing £1B, and £1D and £1F are substituted for £Ø8 (backspace) and £ØD (carriage return). At AN19 CALL CLRLIN is inserted to erase the last input; the program is then unchanged up to AN18:

```
CD 65 29      AN1B     CALL CLRLIN      ;erase
CD F2 28               CALL CRTNES      ;prompt
CD 9E 2C               CALL CHARTE      ;accept answer
FE 4E                  CP £4E           ;is it N?
CA C3 2C               JP Z, AN04       ;if so, jump
CD 97 2D               CALL ROYALT      ;update K and Q positions
CD 65 29               CALL CLRLIN      ;erase
CD 86 2A               CALL INTERR      ;get colour and look-ahead
CD 65 29      AN1C     CALL CLRLIN      ;erase
CD CE 2D               CALL DSPBRD      ;display board
21 18 29               LD HL, TITLE4    ;print title
11 CD 0B               LD DE, £BCD
01 0D 00               LD BC, 13
ED B0                  LDIR
CD 2B 29               CALL WSMOVE      ;prompt
CD 9E 2C               CALL CHARTH      ;accept answer
FE 57                  CP £57           ;is it W?
CA 45 2A               JP Z, DRIV04     ;if so, jump
CD B7 29               CALL PRTMVN      ;print move number
21 85 28               LD HL, SPACE     ;space over
CD 73 29               CALL PRTBLK      ;white's column
3A 20 10               LD A, (KOLOR)    ;computer's colour
A7                     AND A            ;is it white?
20 06                  JR NZ, AN20-0    ;if not, jump
CD C3 2B               CALL PLYRMV      ;get player's move
C3 5C 2A               JP DR0C
CD DC 2A      AN20     CALL CPTRMV      ;get computer's move
C3 5C 2A               JP DR0C
```

Set up Empty Board

```
C5            DSPBRD   PUSH BC          ;save registers
D5                     PUSH DE
E5                     PUSH HL
F5                     PUSH AF
21 A2 0A               LD HL, £AA2      ;address of square a1
3E 7F                  LD A, £7F        ;black square
11 B0 FF               LD DE, -£50      ;line difference
0E 08                  LD C, 8          ;number of rows
06 08         DSP1     LD B, 8          ;number of columns
77            DSP2     LD (HL), A       ;print square
EE 7F                  XOR £7F          ;change colour
2C                     INC L            ;increment
2C                     INC L            ;twice
10 F9                  DJNZ DSP2-0      ;repeat eight times
19                     ADD HL, DE       ;go to next line
EE 7F                  XOR £7F          ;change colour
0D                     DEC C            ;;done?
20 F1                  JR NZ, DSP1-0    ;if not recycle
3E 15         BSETUP   LD A, 21         ;first board index
```

The remainder of the subroutine is unchanged.

Insert Piece Routine   The lines between JR Z, IP04 and IP2C are replaced by:

```
28 02                  JR Z, IP04-0
0E 20                  LD C, £20
E6 07         IP04     AND 7            ;delete flags
47                     LD B, A          ;save in B
11 B9 28               LD DE, PCS+6     ;list of pieces
7B                     LD A, E          ;get address of
90                     SUB B            ;piece letter
5F                     LD E, A          ;in DE
1A                     LD A, (DE)       ;letter in A
81                     ADD A, C         ;lower case if black
77                     LD (HL), A       ;put on board
F1            IP2C     POP AF
```

Board Index to Norm Address Subroutine After DEC D (line 9) the subroutine continues:

```
42                LD B, D          ;rank in B
21 E0 0A          LD HL, £AE0      ;zero point for norm
11 C0 FF          LD DE, -£40      ;row difference
19                ADD HL, DE
10 FD             DJNZ -1
47                LD B, A          ;file in B
2C                INC L            ;column difference
2C                INC L
10 FC             DJNZ -2
F1                POP AF
D1                POP DE
C1                POP BC
C9                RET
```

Square Blinker Between PUSH IX and POP IX the subroutine should be changed to:

```
48                LD C, B          ;save B
56                LD D, (HL)       ;save square contents
06 08     BLNK1   LD B, 8          ;blink rate
36 20             LD (HL), £20     ;space
FF                RST 56           ;delay
10 FD             DJNZ -1
72                LD (HL), D       ;replace square contents
06 08             LD B, 8          ;blink rate
FF                RST 56           ;delay
10 FD             DJNZ -1
0D                DEC C            ;count blinks
20 F0             JR NZ, BLNK1-0   ;recycle
```

Make Move Subroutine Lines 10 to 20 inclusive (MOV A, M to JRZ MM04) and line 25 (CALL INSPCE) are deleted. CALL DSPBRD is inserted between CALL BLNKER and POP HL.

TDL TO ZILOG Z80 INSTRUCTION SET CONVERSION TABLE

TDL		ZILOG	
ACI	x	ADC	A, x
ADC	x	ADC	A, x
ADD	u	ADD	A, u
ADI	x	ADD	A, x
ANy	u	AND	u
BIT	x, u	BIT	x, u
CALL	x	= CALL	x
CCD		CPD	
CCDR		CPDR	
CCI		CPI	
CCIR		CPRI	
CMA		CPL	
CMC		CCF	
CMP	u	CP	u
CPI	x	CP	x
Cu	x	CALL	u, x
DAA		= DAA	
DAD	rr	ADD	hl, rr
DADC	rr	ADC	hl, rr
DADX	x	ADD	Ix, x
DADY	x	ADD	Iy, x
DCR	u	DEC	u
DCX	rr	DEC	rr
DI		= DI	
DJNZ	n	DJNZ	e
DSBC	rr	SBC	hl, rr
EI		= EI	
EXAF		EX	AF, AF
EXX		= EXX	
HLT		HALT	
IN	x	IN	A, (x)
IND		INDR	
INI		= INI	
INIR		= INIR	
INP	x	IN	x, (c)
RARR		RR	
RET		= RET	
RETI		= RETI	
RETN		= RETN	
RLC		RLCA	
RLCR	u	RLC	u
RRCR		RRC	
RST	x	= RST	x
Ru		RET	u
SBy	u	SBC	A, u
SET	x, u	SET	x, u
SLAR		SLA	
SPrp		LD	sp, rp
SRAR		SRA	
STA	x	LD	(x), A
STAX	x	LD	(x, A)
STAx		LD	x, A
STC		SCF	
SUy	u	SUB	u
SrpD		LD	(y), rp
XCHG		EX	DE, HL
XRy	u	XOR	u
XTrp		EX	(sp), rp

TDL		ZILOG	
INR	u	INC	u
INX	rr	INC	rr
IMx		IM	x
JMP	x	JP	x
JMPR	x	JR	e
JRy	n	JR	y, e
Ju	x	JP	u, x
LDA	x	LD	A, (x)
LDAX	x	LD	A, (x)
LDAx		LD	A, x
LDD		= LDD	
LDDR		= LDDR	
LDI		= LDI	
LDIR		= LDIR	
LXI	rr, y	LD	rr, y
LrpD	y	LD	rp, (y)
MOV	u, v	LD	u, v
MVI	u, v	LD	u, v
NEG		= NEG	
NOP		= NOP	
ORy	u	OR	u
OUT	x	OUT	(x), A
OUTD		= OUTD	
OUTDR		OTDR	
OUTI		= OUTI	
OUTIR		OTIR	
OUTP	x	OUT	(c), x
PCrp		JP	(rp)
POP	rr	= POP	rr
PUSH	rr	= PUSH	rr
RAL		RLA	
RALR		RL	
RAR		RRA	

<u>Key</u>

e = n-pc
rp = 16 bit register
u) = as x and y, but may vary thus: M becomes (HL)
v)
dsp(x) becomes (IX + dsp)
dsp(y) becomes (IY + dsp)
P becomes PE
PO becomes PO
rr = register pair where:
B becomes BC
D becomes DE
PSW becomes AF
H becomes HL
X becomes IX
Y becomes IY
x) = same in TDL & Zilog
y)
= = identical

Pets Corner

J. Stout

The PET, according to COMPUTING (3 August 1979), is now the U.K.'s best selling microcomputer system, with over 10,000 installed. This section of 'The Liverpool Software Gazette' is devoted entirely to the PET, and I hope that everyone with access to a PET who reads this will try out the hints, routine or programs in it, correct any mistakes that may have crept in, make any suggestions and/or criticisms that they feel necessary, and most importantly of all contribute more hints, routines and programs. The section will not include details of hardware unless they are essential to the software, e.g. a music program using an amplifier circuit connected to the user port.

Listing Conventions

It would be nice for the section to contain only listings which have been produced by a PET, but with the present state of PET printers there are problems associated with this, since most programs will contain some graphic characters, if only the cursor control ones, so until proper listings can be generated the following convention is proposed:

(1) Cursor control characters are handled by enclosing a 2 or 3 character description of the effect they produce within brackets, e.g. (cls) for clear screen, (cd) for cursor down, (cu) for cursor up, (cl) for cursor left, (cr) for cursor right, (hme) for cursor home, (rvs) for reverse field on, (off) for reverse field off. This has the advantage that if a listing is not available a normal typewriter can produce a copy, and that it is easier to understand than possibly a true listing would be.

(2) Any other graphic character is dealt within a similar way, by enclosing the letter whose key is pressed together with shift to get the graphic within brackets. Thus (ASZX) represents the graphic character string consisting of the 4 playing card suits. Where confusion might arise, e.g. in things as 'Yes (Y) or No (N)' the characters could be replaced with square brackets. Normal lower case characters can simply be reproduced as lower case characters, taking care not to enclose them between brackets if possible.

Anyone with a better convention should get in touch with me and it can be presented for discussion in the section.

As examples of the convention here are a couple of useful routines which can help remove the problem of the PET breaking out of the program when a carriage return alone is entered as the response to an INPUT statement.

```
10 INPUT "Enter a number(cr)(cr).(cl)(cl)(cl)";A$
20 IF A$="." THEN PRINT "(cu)";:GOTO 10
30 A=VAL(A$):REM A NOW CONTAINS THE NUMBER ENTERED
```

Note that lower case characters not enclosed within brackets are simply treated as lower case characters. If a carriage return is entered as the only response to the question, then A$ has the value".", which is detected by line 20, and results in the question being asked again. Line 20 could be replaced by a line which accepted A$ "." as implying that a default value was to be assigned to A.

Another alternative to the simple INPUT statement, and one which is useful if the string to be input must contain commas, semi-colons etc, is to simulate the INPUT statement with a GET statement. For users of PETs with the old ROMs the following lines provide an INPUT-like statement which will not break out of the program when return alone is pressed.

```
10 POKE 525,0:GET A$:IF A$="" THEN 10
20 POKE 525,1:PRINT " (cl)";:REM CHARACTER TYPED IS NOW IN A$
```

(Note that the first character in the PRINT string in line 20 is a space). There is now a choice as to what to do with A$. A 'PRINT A$;:GOTO 10' will result in whatever is typed being printed on the screen (even the delete key will delete the last character printed), but the program is of course in an endless loop. The best thing is to decide on a terminator character, e.g. the return key, and test for it. The routine now becomes:

```
10 as above
20 as above
30 IF ASC(A$) = 13 THEN PRINT A$;:GOTO 10
40 PRINT:"INPUT TERMINATED AND PROGRAM CAN CONTINUE"
```

A better version of line 30 which removes the need for the second print is:

```
30 PRINT A$;:IF ASC(A$) = 13 THEN [illegible]
40 REM INPUT TERMINATED
```

This does still not get over the problem of remembering what has been typed in. To do this insert the following line:

```
5 L$="":REM SET L$ (FOR LINE STRING) TO NULL
```

and change the THEN 10 in line 30 toL$ = L$ + A$:GOTO 10. When the program exits to line 40 L$ will contain the characters which have been typed in. You can input up to 255 characters this way, the characters including commas, semi-colons, trailing spaces etc. One peculiarity of the routine as it stands is that while the delete key will result in the character on the screen being deleted the character in the string will not have been deleted, and more embarrassingly a delete character will have been **added** to L$. To get round this we need to detect the delete key (ASC("(del)") = 20), and chop off the last character in L$ using the LEFT$ function. Perhaps someone would like to take up the challenge of producing an uncrashable input routine using the ideas above, or any others in fact. The routine should return either 1, 2 or 3 in a variable TYPE, depending on whether the input was a number, a string or the default, i.e. simply return. An 'ON TYPE GOTO (or GOSUB)' could then be used to perform the appropriate action. The number (if it was one) should be returned in N, the string (if it was one) is S$, and N set to O, S$ set to "" if the default input was performed. It should take care of the delete key and ignore all other control characters, e.g. (cu), (cls) etc. It may be slow, but input will be slow anway, so it should not make too much difference.

The POKEs in the statements above are necessary to get the cursor to flash, without any lengthy timing loops. For the new ROMs the POKE address is 167, but apart from that everything else should be the same.

Interrupts

An interrupt is generated in the 6502 processor of the PET every sixtieth of a second, which (as long as the interrupts are enabled,) results in the 6502 (at the end of its current machine code instruction), saving the program counter (which will contain the information necessary for it to continue at the correct place when the interrupt is over) and the status of the processor (which contains the information necessary for it to continue doing the correct thing when the interrupt is over) on the stack. It then jumps to an interrupt routine whose address is at the top of the ROMs, $90, $91 (new ROMs). These addresses are in the third and first pages of RAM, and hence can be altered by the user, allowing a non-standard routine to gain control of the 6502 every 1/60 second.

Notes: All numbers preceeded by a dollar sign '$', are in hexadecimal, or base 16. An indirect JMP results in the processor JMPing to the address which is contained in the 2 bytes whose first address is contained in the rest of the JMP instruction. For example, the instruction JMP ($0219), (In machine cade 6C 19 02) would result in the processor taking its next instruction from (i.e. JMPing to) the address contained in locations $0219, $021A (low order byte of the address first). If $0219 contains $3A, and $021A $03, then a JMP ($0219) equivalent to a JMP $033A.

Given that a user routine can gain control after an interrupt what use is it? The main use is to implement a routine which you would like to be executed continuously, i.e. when a BASIC program is running, when the system is waiting for input and so on, and to be executed in this way **without** you having to call it explicitly. Examples might be a continuous memory tester, which cycles through all the memory again and again reporting any faults it detects, but being in effect transparent to the user until a fault is detected. A data gathering routine could be implemented in this way, constantly scanning the user port say, reading a value of some quantity from it, and storing it in some agreed location. A BASIC program could then access this information when it was ready, without having to explicitly trigger the reading routine. One might even implement a form of time sharing, where pages 0-3 of the memory would be swapped at regular intervals, the pointers in the swapped-in pages pointing to a different BASIC program from the pointers in the swapped-out pages. The users are varied, the PET itself using it to update the jiffy clock (which is where 1 jiffy = 1/60 second comes from) and to scan the keyboard for any keys being pressed.

The example shown here will enable you to alter the type of cursor display that you get from your PET. If you are tired of the same boring old cursor then read on. The key to the example is that location $0225 (old ROMs) or $A8 (new ROMs) contains a number which is decremented every time the interrupt routine is called (i.e. every sixtieth of a second). If decrementing this number results in it reaching zero then the current state of the character under the cursor (this state being either reverse field or normal) is flipped, and the contents of location $0225 ($A8) set to 20. Thus every 20 interrupts the character changes from reverse field to normal, or vice versa, and the timing for the cursor is 1/3 of a second between flips.

To produce a grey cursor we can gain control of the

6502 every interrupt, set the cursor timing control location to 1, and then continue with the interrupt as normal. Every time the interrupt is called results in the number being decremented to zero, hence the character under the cursor changes state, and we get the appearance of a grey cursor, actually one changing state every 1/60 second.

The alternative is to make a cursor that never changes state, which gives the appearance of being non-existent. This simply involves setting the contents of the cursor timing control location to 2 (or any number different from 1). The interrupt routine can never decrement 2 by 1 and get to zero, hence the state of the cursor character never changes.

To produce either of these effects we must first write a routine that changes the timing location to 1 (or 2) and then continues with the interrupt. To do this we must know the address that is in locations $0219, $021A ($90, $91). For the old ROMs this is $E685, i.e. $85 in location $0219 and $E6 in location $021A, for the new ROMs $E62E.

The second job is to write a routine that will change first the address in $0219, *$021A* ($90, $91) to that of the initial location of the routine. Finally we must have a routine which restores the original interrupt addresses otherwise tape input/output will not work properly (we will use a version of the second routine to do this).

Below is a BASIC program which should do the job properly, and underneath that is the assembly language program which has been POKEd into the second cassette buffer after the BASIC program has been run.

BASIC ROUTINE TO ALTER STATE OF CURSOR

```
10 POKE 59468,14:REM POKE TO LOWER CASE, NO REAL REASON
20 PRINT "(cls)Program to alter cursor timing.":PRINT
30 FOR I=826 TO 846
40 READ M:POKE I,M:REM POKE MACHINE CODE ROUTINE INTO SECOND
   CASSETTE BUFFER
50 NEXT I
60 PRINT "Machine code installed.":PRINT
70 INPUT "Grey cursor (G) or No cursor (N)(cr),(cl)(cl)(cl)";A$
80 IF A$="," THEN PRINT "(cu)";:GOTO 70
90 IF (A$<>"G")AND(A$<>"N") THEN PRINT "(cu)";:GOTO 70
100 IF A$="G" THEN POKE 840,1:GOTO 120
110 POKE 840,2
120 SYS(826)
130 END
140 DATA 120,169,71,141,25,2,169,3,141,26,3,88,96
150 DATA 169,1,141,37,2,76,133,230
```

To restore the original interrupt vector execute:

```
POKE 828,133:POKE 833,230:SYS(826)
```

All the above is for the old ROMs. To adapt this for the new ROMs make the following changes:

```
30 FOR I=826 TO 843
100 IF A$="G" THEN POKE 837,1:GOTO 120
110 POKE 837,2
140 DATA 120,169,69,133,144,169,3,133,145,88,96
150 DATA 169,1,133,168,76,46,230
```

and to restore the original interrupts addresses execute:

```
POKE 828,46:POKE 832,230:SYS(826)
```

The assembly language versions of the routines follows:

```
Address Op-Codes Assembler   Comments
033A    78       SEI         Disable interrupts (see below)
033B    A9 47    LDA#$47     Low byte of user routine's address
033D    8D 19 02 STA $0219   Low byte of interrupt routine's
                             address
0340    A9 03    LDA#$03     High byte of user routine's address
0342    8D 1A 02 STA $021A   High byte of interrupt routine's
                             address
0345    58       CLI         Enable interrupts
0346    60       RTS         Return from subroutine
```

If the interrupts were not disabled it would be possible, but unlikely, that the first byte of the interrupt routine address could have been altered, but not the second one, when an interrupt occurs, leading in all probability to a crash.

```
0347 A9 01    LDA#$01    1 in location $0348 means that cursor
                         will flip state every interrupt
0349 8D 25 02 STA $0225  Cursor timing constant location
034C 4C 85 E6 JMP $E685  Continue with interrupt
```

The routines for the new ROMs are slightly different, since the interrupt routine address is kept in page zero of the PET's RAM, together with the cursor timing constant, hence the instructions at locations $033D, $0342 and $0349 in the above version can be shortened by one byte each, using the page zero addressing mode of the 6502 processor.

Pascal and the PET

It is difficult to read any computer magazine or paper, whether professionally or personally orientated, without becoming aware of a computer programming language called Pascal. Developed in the late sixties and early seventies by Professor Niklaus Wirth, Pascal is a block structured language very much like ALGOL-60 or -68, with some features not found in either. It is especially suitable for structured programming, having all the control structures built into the language for the processes of SEQUENCE, SELECTION and ITERATION, the three basic building blocks for any structured program. Whereas in most other high-level languages one is restricted as to the type of data the language will handle, (e.g. BASIC with just real and integer types), Pascal allows the creation of new data types, which fit the problem to be solved, rather than fitting the problem to the language. For example, if a selection of programming was needed to sum the number of hours worked in a week, we might, in BASIC, allocate a code of the following form: 1 MONDAY, 2 TUESDAY, . . .,5 FRIDAY, and then perform the following loop

```
10 S=0
20 FOR I=1 TO 5
30 S=S+T(I)
40 NEXT I
```

Pascal allows the following types of construction:

```
type WEEKDAY= (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY);
var  DAY    : WEEKDAY;
     HOURS  : array [MONDAY..FRIDAY] of integer;
     TOTAL  : integer;

TOTAL:=0
for DAY:=MONDAY to FRIDAY do TOTAL:=TOTAL+HOURS [DAY];
```

Obviously you have to tell the computer more to start off with (since in Pascal all variables must be defined before they are used), but once that is done, (and it is a useful exercise even in languages which do not demand it) the program you write almost documents itself, especially as you can use long (at least 8 characters) variable names. This facility of being able to define the way the data for a program is represented is seen by Wirth to be as important as the choice of algorithm for the program (one of his books is titled 'Algorithms + Data Structures = Programs' Prentice-Hall 1976).

This article does not aim to teach Pascal, since there are enough books around which will do that easily, but rather to let PET users know how they can go about gaining some experience of Pascal. What follows applies in fact to almost any system with BASIC, although the particular implementation described is for a PET.

The September to November 1978 issues of BYTE contained a series on how to develop a 'Tiny' Pascal compiler, interpreter and translator (bearing a strong resemblence to a system described by Wirth in Algorithms + Data Structures = Programs, for a language called PL/O). The 'Tiny' Pascal referred to is a subset of Pascal, with for example only integer variables and constants, and only single dimension arrays, again of integers. However, it does support procedures and functions, (even recursive procedures), and provides an excellent way for someone to get acquainted with Pascal.

The compiler, which is written in BASIC, takes a program written in the subset of Pascal chosen and complies it into an intermediate form known as P-code (a form of machine code for a hypothetical processor). The interpreter can then interpret these P-codes in the same way as a BASIC interpreter interprets a BASIC program, providing single step, breakpoint and register examine facilities. When the program is working it can be translated from the P-code into the machine code of the processor it is to be run on—which will not only make it run faster but will probably result in its taking up less memory.

The original P-compiler (October 1978) was written in North Star BASIC, but is fairly easy to convert to PET BASIC (North Star BASIC makes the test in a FOR-NEXT **before** it performs the loop, hence FOR I = 1 TO 0:PRINT:NEXT I won't do a thing. One of the problems associated with the translation). The P-code interpreter (September 1978) was written in 'Tiny' Pascal, but is easy to translate into PET BASIC, and finally the P-code translator was written in BASIC for an 8080 microprocessor, hence will need completely rewriting, together with the run-time package which supports the translated P-code.

The compiler was designed as a bootstrap compiler by the authors (Kin Man Chung and Herbert Yuen) of the articles, so that when it was working a compiler for a more expanded subset of Pascal could be implemented using a 'Tiny' Pascal version of the bootstrap compiler. Even if this next step is not taken, the system remains an excellent way to get to know what a compiler does, and how it does it, and also an excellent way to get to know Pascal.

If sufficient interest is shown (please make your views felt, either to Microdigital or myself), and questions of copyright can be sorted out, it might be possible to publish the complete set of listings from the BYTE articles in this section. A version of the system is at present running on an 8K PET with 24K extra memory and a Compu/Think dual mini-floppy disk drive, although only using one of the drives. An editor is used to prepare the program in a file on the disk, the compiler reads the source text from the file, and the interpretor interprets the compiled P-code, **very** slowly (an interpreted program interpreting something is bound to be slow). The next stage is to rewrite the P-code interpreter in machine code for the PET, and possibly even develop the run-time package and translator for the 6502.

Stop Press

THE PET WAKES UP

A tip from Jim Butterfield for all Pet users and owners with *new* Roms:

If your machine crashes, either from BASIC or machine code the following hardware/software technique will reawaken it, with very little damage to memory, e.g. a Basic program should still be usable.

1. Ground the diagnostic sense pin on the user port (pin 5)
2. Ground the Reset Pin on the memory expansion bus (pin 22)
3. The Pet should awaken in the monitor, but the stack pointer value will be 01.
4. If you wish to re-enter Basic enter 'X (Return)', which should give 'READY'. Then enter 'CLR (Return)'. The Pet should now be usable.
5. If you wish to stay in the monitor, enter '; (Return)' which should give ?. Then cursor up and alter the SP value to FA and press (Return).

The new PETS mapped out—J. Butterfield

LOCATION HEX	DEC	PURPOSE
000-0002	0-2	USR Jump instruction
0003	3	Search character
0004	4	Scan-between-quotes flag
0005	5	Basic input buffer pointer: subscripts
0006	6	Default DIM flag
0007	7	Type: FF = string, 00 = floating point
0008	8	Type: 80 = integer, 00 = floating point
0009	9	DATA scan flag: LIST quote flag: memory flag
000A	10	Subscript flag: FNx flag
000B	11	0 = input; 64 = get; 152 = read
000C	12	ATN sign flag: comparison evaluation flag
000D	13	input flag: suppress output if negative
000E	14	current I/0 device for prompt-suppress
0011-0012	17-18	Basic integer address (for SYS, GOTO etc)
0013	19	Temporary string descriptor stack pointer
0014-0015	20-21	Last temporary string vector
0016-001E	22-30	Stack of descriptors for temporary strings
001F-0020	31-32	Pointer for number transfer
0021-0022	33-34	Misc. number pointer
0023-0027	35-39	Product staging area for multiplication
0028-0029	40-41	Pointer: Start-of-Basic memory
002A-002B	42-43	Pointer: End-of-Basic, Start-of-Variables
002C-002D	44-45	Pointer: End-of-Variables, Start-of-Arrays
002E-002F	46-47	Pointer: End-of-Arrays
0030-0031	48-49	Pointer: Bottom-of-Strings (moving down)
0032-0033	50-51	Utility string pointer
0034-0035	52-53	Pointer: Limit of Basic Memory
0036-0037	54-55	Current Basic line number
0038-0039	56-57	Previous Basic line number
003A-003B	58-59	Pointer to Basic statement (for CONT)
003C-003D	60-61	Line number, current DATA line
003E-003F	62-63	Pointer to current DATA item
0040-0041	64-65	Input vector
0042-0043	66-67	Current variable name
0044-0045	68-69	Current variable address
0046-0047	70-71	Variable pointer for FOR/NEXT
0048	72	Y save register; new-operator save
004A	74	Comparison symbol accumulator
004B-004C	75-76	Misc numeric work area
004D-0050	77-80	Work area; garbage yardstick
0051-0053	81-83	Jump vector for functions
0054-0058	84-88	Misc numeric storage area
0059-005D	89-93	Misc numeric storage area
005E-0063	94-99	Accumulator 1: E.M.M.M.M.S
0064	100	Series evaluation constant pointer
0065	101	Accumulator hi-order propogation word
0066-006B	102-107	Accumulator 2
006C	108	Sign comparison, primary vs. secondary
006D	109	low-order rounding byte for Acc 1
006E-006F	110-111	Cassette buffer length/Series pointer
0070-0087	112-135	Subrtn: Get Basic Char; 77,78 pointer
0088-008C	136-140	RND storage and work area
008D-008F	141-143	Jiffy clock for TI and TI$
0090-0091	144-145	Hardware interrupt vector
0092-0093	146-147	Break interrupt vector
0094-0095	148-149	NMI interrupt vector
0096	150	Status word ST
0097	151	Which key depressed: 255 = no key
0098	152	Shift key: 1 if depressed
0099-009A	153-154	Correction clock
009B	155	Keyswitch PIA: STOP and RVS flags
009C	156	Timing constant buffer
009D	157	Load = 0, Verify = 1
009E	158	characters in keyboard buffer
009F	159	Screen reverse flag
00A0	160	IEEE-488 mode
00A1	161	End-of-line-for-input pointer
00A3-00A4	163-164	Cursor log (row, column)
00A5	165	PBD image for tape I/0
00A6	166	Key image
00A7	167	0 flashing cursor, else no cursor
00A8	168	Countdown for cursor timing
00A9	169	Character under cursor
00AA	170	Cursor blink flag
00AB	171	EOT bit received
00AC	172	Input from screen/input from keyboard
00AD	173	X save flag
00AE	174	How many open files
00AF	175	Input device, normally 0
00B0	176	Output CMD device, normally 3
00B1	177	Tape character parity
00B2	178	Byte received flag
00B4	180	Tape buffer character
00B5	181	Pointer in filename transfer
00B7	183	Serial bit count
00B9	185	Cycle counter
00BA	186	Countdown for tape write
00BB	187	Tape buffer 1 count
00BC	188	Tape buffer 2 count
00BD	189	Write leader count; Read pass1/pass2
00BE	190	Write new byte; Read error flag
00BF	191	Write start bit; Read bit seq error
00C0	192	Pass 1 error log pointer
00C1	193	Pass 2 error correction pointer
00C2	194	0 = Scan; 1=15 Count, $40 = Load; $80 = End
00C3	195	Checksum
00C4-00C5	196-197	Pointer to screen line
00C6	198	Position of cursor on above line

E810	Diagnos Sense	IEEE EOI in	Cassette Sense 2	1	KEYBOARD ROW SELECT PA				59408
E811	Tape 1 Input Flag		Screen Blank Output (unused on 32K) CA2			DDRA Access	Cassette 1 Read Control CA1		59409
E812	KEYBOARD ROW INPUT								59410
E813	Retrace 1 Flag	. . .	Cassette 1 Motor Output CB2			DDRB Access	Retrace Interr Control CB1		59411
E820	IEEE.INPUT								59424
E821	ATN 1 Flag	. . .	IEEE NDAC out CA2			DDRA ACCESS	IEEE ATN in Control CA1		59425
E822	IEEE-OUTPUT								59426
E823	SRQ 1 Flag	. . .	IEEE DAV OUT CB2			DDRB ACCESS	IEEE SRQ in control CB1		59427
E840	DAV in	NRFD in	Retrace in	Cass 2 Motor	Cassette Output	ATN out	NFRD out	NDAC in PB	59456
E841									59457
E842	DIRECTION REGISTER B (FOR E840)								59458
E843	DIRECTION REGISTER A (FOR E84F) (R.U.P.)								59459
E844	TIMER 1							L	59460
E845	WRITE							H	59461
E846	TIMER 1							L	59462
E847	LATCH							H	59463
E848	TIMER 2							L	59464
E849								H	59465
E84A	SHIFT REGISTER								59466
E85B	TI Control PB7 out	One-Shot Free-Run	T2 Contr. PB6 Sense	Shift Rec. Control			PB, PA Latch Control		59467
E84C	CB2 (P.U.P. Control In/Out			CB1 in Cassette 2 Polarity	CA2 (Graphics, Lower Case) Control In/Out			CA1 in POLARITY	59468
E84D	IRQ Status	T1 INT	T2 INT	CB1 Cass 2 Int		SR INT	CA1 (P.U.D.B.) Int	CA2 Int	59469
E84E	Enable Clear/Set	T1 Int ENAB	T2 Int ENAB	CB1 Int Enab	CB2 Int Enab	SR Int ENAB	CA1 Int ENAB	CA2 Int ENAB	59470
E84F	PARALLEL USER PORT I/0 (PA)							PA	59471

Programming Practices and Technics

Dr. M.Beer

THIS is, I hope, the first of a regular series in which I shall look at various programming topics of interest to the micro-computer owner. The object is to cover many of the techniques used to ease the programming of a small computer by discussing both programming methods in general, and suitable software products as they appear on the British market. I do not intend to dwell too much on the topic of computer languages as, in general, it is possible to apply most modern programming techniques when writing in many computer languages. The choice of language should be determined by which provides the facilities required to solve the problem in hand, not my the methods used. It must be admitted, though, that by choosing the right programming language the application of systematic programming techniques is greatly simplified.

This first article will look at the use of one very common program, an assembler. Your microcomputer most likely came with facilities to run a high-level language, probably BASIC, and a simple monitor which allows you to load and execute programs written in machine code. These are fine to get you started. You can load an execute BASIC using the monitor (you do this on any machine, even if the monitor is hidden from view). Most programs you will write, or buy, will be written in BASIC, but on occasion you will find that BASIC does not give you the control over the microcomputer you require.

A typical case are subroutines to allow your microcomputer to communicate with other devices, such as printers, paper tape readers, or even other computers. If you are very lucky your microcomputer's monitor will allow you to list a section of memory in a pseudo-assembler format. This is normally called dis-assembly, and allows you to look at sections of program, already stored in the computer, in a more digestable form than the straight hexadecimal printout usually provided. It is possible that the monitor on your computer will even allow you to enter programs in the same form. The use of mnemonics, rather than the hexadecimal operation codes actually understood by the micro-computer, eases the programmer's lot considerably.

Mini-assemblers, such as these, are fine if you wish to write short subroutines to interface with BASIC programs. They are not very useful if you wish to write a reasonably long program which has to handle a number of different situations. The mini-assembler requires all data and addresses to be entered as hexadecimal numbers, so that, if, say, you wish to add an instruction you forgot, you have to rewrite a large section of the program. Deleting instructions is easier since they can be replaced by no-operations.

If the program is longer than a few dozen bytes, or rather complex, it is far easier to use a full symbolic assembler. The program is entered into the computer as a text file, using an editor, and can be stored either in the computer's main memory, or on floppy disc, or cassette. The editor is a program which allows the programmer to manipulate a file containing text by adding, deleting or changing its contents. Editors are very complex programs, which must be well written so that they protect the contents of valuable files from accidental corruption. I intend to discuss editors more fully in a later article, as they are an important software tool, and should be available on any suitable system.

The assembler normally does its work in two stages, called passes, the first creating a symbol table in which the values of all the symbols used are stored, and the second, where the code is actually generated. It is usual for a listing to be generated giving the code produced alongside the assembler statements originally entered. Since symbolic labels are used to refer to addresses adding, or deleting code is much simpler as the source file can be edited, and the assembler will recalculate them. By giving the various constants and data storage addresses used in the program meaningful names and by adding plenty of sensible comments the program text can be made quite readable. It should be obvious what the program segment in example 2 is attempting to achieve, whilst when the same program is presented in mini-assembler format (example 1) it is far from clear.

Although a symbolic assembler is required to do a lot more than a mini-assembler it is a great help when developing even moderate sized programs since it frees you from calculating addresses, which is always time consuming, and, particularly in the case of forward references, sometimes impossible.

These articles describe some of the work I have done in connection with a research project involving the study of programming methods for microcomputers. I would like to hear from anyone interested in this area, so that their views may be included in later articles. Programming techniques have, so far been neglected by microcomputer owners, who have either been too busy getting hardware to work, or have had an immediate problem to solve. Suppliers are naturally concerned to promote the advantages of the machines they provide, and have neglected the ready market for software. In the next few months I think this will change. Consideration should be given, when purchasing a microcomputer to the availability of software and other material, as these will extend the usefulness of the machine as time goes on.

Next month I shall look at compilers and interpreters and show why both are invaluable to the microcomputer user.

Example 1. A short subroutine entered using a mini-assembler.

```
300:    LDA $C001
        AND #$02
        BEQ $300
        LDA $C000
        ORA #$80
        RTS
```

Example 2. The same short subroutine entered using a full symbolic assembler.

```
; ROUTINE TO READ A CHARACTER AND LEAVE IT
; IN THE A REGISTER.
STATUS   EQU $C001
PORT     EQU $C000
MASK     EQU $02
PARITY   EQU $80
;
         ORG $300              ; START ADDRESS.
;
READCH   LDA STATUS            ; CHARACTER READY ?
         AND #MASK
         BEQ READCH
         LDA PORT              ; FETCH IT.
         ORA #PARITY           ; BIT 7 ALWAYS SET.
         RTS
```

5: A high level language in 3/4 K!
M5 SYSTEM—AN INTERPRETER FOR THE NASCOM ONE

Designed and Implemented by Raymond Anderson

0.0 The M5 Language

0.1 Nascom Implementation

The M5 interpreter was designed for implementation on small 8 bit microcomputers and the Nascom one standard system was an ideal choice because of its popularity and use of a fairly powerful processor (the Z80).

With only about 940 bytes available to the user, the language had to be compact enough to write decent programs in a small space, and also have a small interpreter to leave the maximum amount of spare memory. A simple editor was almost essential if programs of over about 50 bytes were to be written and debugged easily, and this required about 100 bytes.

The editor, interpreter and command mode are closely linked—for example, program variables are maintained over edits, and resets, and the editor will set up its cursor to inform the user where an error occured.

A compact M5 program can be difficult to follow initially, so error routines which give the exact location and type of a run-time error are included, despite the penalty in RAM usage. (Execution speed is unaffected by error checking).

M5 is a very fast interpreter, although loops are not as fast as in machine code because each loop involves a small search. A well written M5 program will carry out general calculations at about 1/3-1/5 of the speed of machine code. (M5 programs are usually much faster to write and debug of course!)

The user may write programs of about 230 bytes in length—quite large in M5. Overlarge programs may cause trouble when entered, but the most likely indication of an overflow is a lot of garbage appearing on the end of the program when it is listed.

0.2 Introduction

The M5 system is entered by typing EC60 when M5 has been entered into user RAM. The prompt 'M5:' should then appear at the bottom of the screen, indicating that the system is in the command mode. Commands which may be entered now are:

I	Input	a new program and destroy the previous one. System responds with a newline and waits for the user to enter a program. Input is terminated by a semi-colon, which returns the user to the command made.
L	List	the program currently in store and return to command mode.
R	Run	the current program starting at the first symbol, after printing a newline.
E	Edit	the current program, inserting the character pointer at the place the last instruction was executed—or where an error was found. (See section on editor.)
RS	RESET	the Nascom. This will cause a return to Nasbug. However, the current program and value of X will be maintained ready for typing EC60 to resume programming. RESET must also be used to star a looping program.

0.3 Initialisation

When entering M5 for the first time after loading it, it is best to initialise the user work area by entering and running a null program. This is done as follows: (Underlined characters are typed by the system.)

M5:Input

: (I.E. Terminate input after entering nothing!!!)

M5:R (Null program simply results in a carriage return.)

M5: (System is now initialised.)

0.4 Other commands

M5 will respond with a new prompt to any unknown command letter.

0.5 Errors on input

A backspace will delete the last character only when in input mode. It may seem misleading if used to backspace up a line. (Try it and see!)

Backspaces can be inserted into a string in the program by using the INSERT command in EDIT mode.
Semicolons are illegal characters inside an M5 program.
Shift-Backspace is a legal character in strings.

1.0 BASIC M5 LANGUAGE PRINCIPLES

1.1.0 M5 Arithmetic

The basic elements handled in standard M5 are 16 bit unsigned integers, which are adequate for most games and simple simulation or number manipulation. Numbers are in the range 0 - 65535 (decimal) and are modulo 65536 so 65536 seems the same as zero to the language.

Operators permitted in M5 are:

* (multiply) / (divide) + (add) – (subtract) # (-1) & (+1)

the last two are included for faster execution if required, and for compact programming of loop control. (See later).

1.1.1 The Stack

An important aspect of M5 which is quite powerful once it is understood, is its stack based (Reverse polish) expression analysis. This system requires no parentheses and it can be used to evaluate arbitrary expressions quickly. The M5 algebraic system is similar to that found on some calculators and the analogy with a calculator is used in these notes.

1.1.2 The Current Value

On a pocket calculator, the idea of a current value is easy to understand as it appears on the display and is often called "x". In M5 there is also a current value (called "X"), and it is altered only in the following circumstances:

1) If a number appears in the program (not in a string) x takes its value.
2) On encountering an identifier A-7 x takes the value stored there.
3) On encountering a ? (not after =) x takes its value from the keyboard.
4) After a diadic operator (/ - + *) x becomes the result.
5) If x is incremented or decremented (using & or #).

1.1.3 Variables

As in most other languages, M5 has variables A-7 and a special one @.
One of these variables becomes current by simply quoting it in the program.
(point 2 above).
X may be stored in asvariable by simply using =k where k is a variable name.
If = ? is used, the current value (x) is displayed as a decimal number on the screen. (This is how numbers are output in M5).

EXAMPLES (These are all legal M5 programs–Try if unsure!)

(i) A What is in location A is now also in x (the current value).
(ii) ABC x takes on the values in A then B then C and keeps the value C.

(iii) 23 x becomes 23.
(iv) 23A x becomes 23 , then x becomes A (i.e. the number in A).
(v) 23 456 x becomes 23 and then x becomes 456.
(vi) A=B x becomes A , then this value is stored in B.
(vii) A=B=C=D x becomes A , then this value is put into B , C and D.
(viii) A=? D=? x becomes A and this is displayed, then x becomes B and is displayed.
(ix) =?=A x what is in x (left from last program) is displayed and put in A).

N.B. If you want to check what is going on, put the characters: =? in your program and x at these points will be printed.
For neatness and readability use: =? " " which separates No's by a space.
E.G. 23=?" " 1 1 1 1 1 =?" will produce: 00023 11111 as output if run.

1.1.4 Calculating

When a comma is encountered in an M5 program, the value of x is put on the top of the stack–pushing down all other members.
We can represent the stack diagramatically to show what happens.

Imagine the M5 program A,33,,BA where initially A=1 and B=2
step: abcdefgh (could have run 1=A2=3 before)

and follow it step by step:

```
STEP SYMBOL MEANS    x    top-STACK-bottom->   y (top element of stack)
---- ------ -----    -    ------------------   -
 a     A    load A   1    -  -  -  -           unknown
 b     ,    push x   1    1  -  -  -             1
c-d    33   load 33  33   1  -  -  -             1
 e     ,    push x   33   33 1  -  -            33
 f     ,    push x   33   33 33 1  -            33
 g     B    load B   2    33 33 1  -            33
 h     A    load A   1    33 33 1  -            33
Note that the top member of the stack is called y .
```

So far, we have no means of removing items from the top of the stack. We do this by using operators such as + / * - .

The operators work on x and y and put the result in x, removing y from the stack.
Operators therefore do the following:

```
Operator      Function         Remarks

   #          x := x-1         This is the pound sign on the Nascom
   &          x := x+1         Much faster than ,1+ which is equivalent
   +          x := x+y         y is lost. Overflow not detected (M5 2.1)
   -          x := y-x         y is lost. Underflow not detected.
   *          x := x*y         y is lost. Overflowing bits put in B
   /          x := y/x         y is lost. Remainder is put in A
```

EXAMPLES

Program A,B+=? Initially A=1 B=2
step: abcdef

```
STEP SYMBOL MEANS    x    y  Rest of stack
---- ------ -----    -    ----------------

 a     A    load A   1    ?  -  -  -  -
 b     ,    push x   1    1  -  -  -  -
 c     B    load B   2    1  -  -  -  -
 d     +    x:=x+y   3    -  -  -  -  -
e-f    =?   display  3    -  -  -  -  -   ( 3 is displayed on screen  00003 ).
 The program displays the result of A+B
```

Program to evaluate (2*3) + (7-2) and display it.

Program 2,3*, 7,2- + = ? i.e. add result of 2,3* to 7,2- and display.
step: abcd e fghi j kl

```
STEP SYMBOL MEANS    x    y  Rest of stack
---- ------ -----    -    ----------------
 a     2    load 2   2    ?  -  -
 b     ,    push x   2    2  -  -
 c     3    load 3   3    2  -  -
 d     *    x:=x*y   6    -  -  -
 e     ,    push x   6    6  -  -
 f     7    load 7   7    6  -  -
 g     ,    push x   7    7  6  -
 h     2    load 2   2    7  6  -
 i     -    x:=y-x   5    6  -  -
 j     +    x:=x+y   11   -  -  -
 kl    =?   display  11   -  -  -          00011 appears on screen - the answer
```

NOTE The operators # and & only affect x and are equivalent to ,1- and ,1+ (although faster and shorter).

Imagine we want to store the result of multiplying N by M in A.

In Basic this is A=M*N
But in M5 this is M,N*=A

Here are some further examples of expressions:

```
    BASIC              M5
    =====              ==

Z=N*M*A            N,M*,A*=Z    OR   N,M,A,**=Z
Z=(N+M)*A          N,M+,A*=Z
Z=(N+M)*(A-M)      N,M+,A,M-*=Z
Z=N*N              N,*=Z
Z=N*N*N*N          N,*,*=Z   OR N,,,***=Z    (N,M, M5 ONLY NEEDS TO GET N ONCE )
```

1.2 Getting Data In

Data in M5 is Input from the keyboard. The program requests a number from the keyboard when it encounters a LOAD ? i.e. a ? in the program, not following =.

A number is terminated by any non numeric character. Usually the user will type a space after the number and the program will continue on the same line, otherwise he will use a newline after typing the number.

EXAMPLE ? , ? * = ? will prompt for a number, then another and print the product.

1.3 String print

Any string of characters surrounded by quotes '' " '' is printed to the display exactly as written– including newlines etc.

e.g. "Input the number"
or "NEW
LINE"

N.B. A jump will find labels in a string so beware of using (in a string.

A nicer version of the program above is:

"NUMBER" ?, "TIMES BY"?*" IS "=?

A newline is produced by a newline between quotes.

1.4 Loops and jumps

A way of repeating operations is almost essential in a programming language. In M5 this is done by using using jumps and labels.

A label is represented in M5 by in where n is any symbol which can be entered at the keyboard. Examples are: (A (! (1 (.

A jump is represented by lkn where n is a symbol which matches a label, and k is a condition code indicating what condition involving x or x and y must be true for the jump to occur.

Valid condition codes are as follows:

CONDITION CODE CHARACTERS:

Character	Jump occurs if:	Comments:
U	–unconditional–	U stands for unconditional
Z	value of x is 0	7 stands for zero
N	value of x is not 0	N stands for non zero
E	x=y (top 2 on stk)	E stands for equal
X	x=y	X looks like a notequal sign
L	x = y	L stands for less than or equal
G	x = y	G stands for greater than
M	–unconditional–	M is monitor . jump to editor.

EXAMPLES of valid jump symbols are:

)UA)NI (X$)G((Z. matching labels above.

when a jump symbol is reached, the condition indicated by K is tested and if it is found to be true, a jump is made to the first occurence of a label with matching identifier symbol.

EXAMPLES:

```
(i)     2000 (A  "HELLO" #  )NA        prints out "HELLO" 2000 times.
(ii)       0 (A =?  " "  &  )NA        prints out numbers from 0 to 65535
                                       separated by spaces. (Thinks 65536=0 ).
(iii)    (A  )UA                       loops until RESET is pressed.
(iv)    0=N (A N=? &=N , 5555 )GA      prints out numbers from 0 to 5555.
```

2.0 WRITING PROGRAMS

M5 is a powerful language when all its features are properly understood, but it can be a little confusing for the beginner. There is fortunately an easy way of generating programs which can be used until familiarity with M5 is achieved. The method is to write the program in a more standard language and then translate into M5. While this method does not exploit the valuable 'current variable' feature of M5, it will yield workable programs which are easier to follow in many ways. The program can then be optimised when it has started to work.

EXAMPLE: A Program to print a table of squares from 1 to 30.

BASIC	M5
10 PRINT "TABLE OF SQUARES"	"TABLE OF SQUARES "
20 N=0	0=N
30 N=N+1	(B N,1+ = N
40 PRINT N, N*N	N=? " " N,N*=? " "
50 IF N = 20 GOTO 30	N , 20)XB
60 END	)M

NOTE: Newlines in output must be included between quotes in M5 programs. The numbers in M5 are not spaced on output, hence the space in the line equivalent to line 40.

The M5 produced will be completely sound and will run at about the same speed as the tiny Basic program.

If the M5 is optimised, keeping N in "x" as much as possible and using the free layout and the & operator, the speed will be considerably faster, perhaps 4-5 times faster than a fast tiny basic.

Optimised:

```
"TABLE OF SQUARES
" 0=N (B N&=N=? " " ,*=? "
"N,20 )XB )M
```

3.0 THE EDITOR

3.0 Introduction

The M5 Editor is entered by typing E when in the command mode.

The edit prompt of E: will appear when the editor is ready to accept input.

The editor will show the point where the last instruction was executed when it is entered by positioning a cursor at this location. The cursor is a shaded in square which is denoted here by a — (underline).

The cursor indicates the current position of the character pointer, and the character pointed at by the cursor appears at the top right of the screen. All manipulation of text is done relative to this cursor because there are no line numbers in M5.

The character indicating end of file in M5 is a null character which appears as a box when it is pointed at.

A hazard in the M5 interpreter is that the pointer can be moved into the actual M5 Interpreter. A Rule must therefore be: DO NOT use any Delete or insert commands unless you can see where the pointer is positioned.

3.1 Commands

To manipulate the text of a program, the user must be able to position the cursor in the required area and then operate on the text. Commands to move the pointer are as follows:

>	Move cursor forward one place.
<	Move cursor backward one place.
R	Rewind–i.e. move cursor to the start of the file.
N	Move the cursor to the start of the next time (stop at end of prog.)

These commands may be repeated and if followed by a newline, will result in a printout of the text with the cursor in its new position.

EXAMPLE: You have typed in a program as follows:

```
(A "HELLO THERE " N=? " IS N
WHAT NUMBER DO YOU WANT" ; . . . . . . . . etc
```

And you want to move the cursor to the spelling error.

Use: RN i.e. move to start, move down a line, move in 5 characters.

Using a space instead of a newline will not print out the text but will carry out the actions and return the edit prompt.

Once we have moved the prompt to where we want to make adjustments we have commands to delete and insert characters.

D   Remove (delete) the character pointed at by the cursor.
         The cursor now points to the next character along.

Innnn;   Insert the string nnnn before the character pointer.
         The terminator is a ;* Cursor points to same character.

EXAMPLE: Edit ABCDERTYIJKLMNOP to replace RTY by FGH

ABCDEFRTYJKLMNOP

E:R   Move pointer to start the along 7 characters (to R)

ABCDEF–TYIJKLMNOP Character R appears at top R.H. side of screen.

E:D   Delete current character.

ABCDEF–YIJKLMNOP T appears at top right.

E:DD   Delete two more.

ABCDEF–JKLMNOP I appears at top right.

E:IGHI;   Insert correct characters.

ABCDEFGHI–KLMNOP string now correct– O still current character.

When editing is complete, the command W is used to return to command mode.

4.0 ERROR MESSAGES

When a large program is written concisely in M5, errors may be difficult to detect so good errr diagnostics at runtime were included.

If a syntax error occurs, one of the following messages will appear:

SYM	FRR	x	The symbol x is not allowed in M5 (except in a string).
10	ERR	x	The symbol x is not a valid identifier, and an attempt was made to copy a value into it. (e.g. =x occurred.)
JID	ERR	x	The label x was not found when a jump occurred to it.
JC	ERR	x	The symbol x occurred in a jump condition position and is not a valid code (one of U A N Z X G E M).
	ERR	x	The symbol x caused an error to occur. (Not one of above.)

In addition to giving the error type, the editing cursor is set up to point at the faulty symbol, so when the editor is entered from the monitor to correct the error, the cursor is in the correct position for amendments.

(N.B. in M6, JID errors are detected before the program starts to execute.)

5.0 SAMPLE PROGRAMS IN M5

```
Number summing program   [A"INPUT A NUMBER"?," THANKS
----------------------     NOW INPUT 2 MORE NUMBERS"?,"AND"?"GOOD!
                           THEIR SUM IS "++=? "
                           "INA "THEIR SUM WAS ZERO - TYPE 0 FOR MORE FUN OR
                           1 TO END "?]ZA "GOODBYE!" ]M
Factorial of a number!   1=N ? ]ZB [A  =N , N* =N N# ]NA [B N=?
----------------------
M5 24 hour clock!
-----------------           ]US [D  N#=N ]ND
[N.b. remove all            H=?" HRS "M=?" MINS "S=?" SECS
 spaces for good            " L=N S&=S , T ]XD
 timekeeping ]                0=S M&=M , T ]XD
                              0=M H&=H ,24 ]XD
                              0=H ]UD
[Start pt at end]           [S 1750=L  60=T
                            "SET HRS"?=H"SET MINS"?=M"SECS"?=S"
                            " ]UD
```

Note that the main timing loop is at the beginning for higher speed.
1750 is the timekeeping constant. make smaller to speed up clock.

```
Square root of a number!  256=N  ?=N  [I  N,M/ , M ]LS +,2/=M ]UI
------------------------
                             [S " "N=?" "
Method used is very fast but a little hard to follow.

Prime numbers:          1=T
--------------       [N T&&=T
                        1=G
                     [A G&&=G
                        T,G/,G ]GP
                        B ]NA ]UN
                     [P T=? "
                        " ]UN

This can be compacted to only one line of course, [ a bit baffling though ]:
1=T[NT&&=T1=G[AG&&=GT,G/,G]GPB]NA]UN[PT=?" "]UN
```

Hexadecimal object code listing 23 MAR 79 14.14

```
Addr          Bytes                              Bytes
0C50  D6 3F CD 01 0E 5E 23 56    18 3B E1 ED 52 EB 18 35
0C60  C3 3E 0E EF 3F 00 21 00    00 CD 25 0E CD 14 0E 38

0C70  F8 EB 18 21 62 6D FD 21    0A 0E AF FD 46 01 FD 4E
0C80  00 ED 42 38 03 3C 18 F9    09 C6 30 CD 3B 01 FD 23

0C90  FD 23 00 20 E5 DD 23 DD    7E 00 FE 20 28 F7 FE 1F
0CA0  28 F3 FE 3F 28 5D 30 A8    FE 2C 28 30 FE 3D 28 33

0CB0  FE 29 CA 74 0D FE 23 28    46 FE 26 28 3F FE 28 28
0CC0  36 FE 2D 28 95 FE 2A 28    39 FE 2F 28 56 FE 28 23

0CD0  0E FE 22 28 6C B7 CA 3E    0E C3 54 0D D5 18 B6 DD
0CE0  23 18 B2 DD 23 DD 7E 00    D6 3F 28 B9 DA C7 0D CD

0CF0  01 0E 73 23 72 18 9E E1    19 EB 18 99 13 18 96 18
0D00  13 93 C1 3E 10 21 00 00    CB 7A 28 04 09 30 01 13

0D10  3D 28 09 EB 29 EB 29 30    EF 13 18 EC EB 22 C0 08
0D20  C3 95 0C 42 4B 21 C0 00    D1 3E 10 29 EB 29 EB 30

0D30  02 23 B7 ED 42 13 F2 3C    0D 09 CB 83 3D 20 EC 18
0D40  DC DD 23 DD 7E 00 FE 22    CA 95 0C B7 CA 3E 0E CD

0D50  3B 01 18 ED D6 30 FE 0A    30 13 21 00 00 DD 7E 00
0D60  DD 23 CD 14 0E 38 F6 EB    DD 2B C3 97 0C EF 53 59

0D70  4D 00 18 57 DD 7E 01 FE    4E 28 31 FE 55 28 5B FE
0D80  5A 28 23 08 E1 E5 B7 ED    52 08 FE 45 28 24 FE 58

0D90  28 23 FE 4C 28 22 FE 47    28 23 FE 4D CA 3E 0E EF
0DA0  4A 00 DD 23 18 25 7A B3    28 30 18 14 7A B3 20 2A

0DB0  18 0E 03 18 F3 03 18 F6    08 30 1F 18 03 08 38 1A
0DC0  DD 23 DD 23 C3 95 0C EF    49 44 00 EF 20 45 52 52

0DD0  20 00 DD 7E 00 CD 3B 01    18 64 DD 4E 02 31 FA 0F
0DE0  21 FE 0E 06 28 7E 23 B8    28 0D B7 C2 E5 0D DD 23

0DF0  DD 23 EF 4A 00 18 D0 7E    B9 20 EA E5 DD E1 C3 95
0E00  0C 07 4F 06 00 21 BE 0A    09 C9 10 27 E8 03 64 00

0E10  0A 00 01 00 D6 30 FE 0A    D0 29 54 5D 29 29 19 5F
0E20  16 00 19 37 C9 CD 3E 00    C3 3B 01 EF 1F 00 21 FD

0E30  0E 23 7E B7 C8 CD 3B 01    18 F7 AF 77 23 77 EF 1F
0E40  4D 35 3A 00 CD 25 0E FE    4C CC 2B 0E FE 49 CA D3

0E50  0E FE 52 20 09 EF 1F 00    DD 21 FD 0E 18 A0 FE 45
0E60  20 DC DD E5 E1 4E 36 7F    E5 79 32 F6 0B CD 2B 0E

0E70  E1 71 EF 1F 45 3A 00 CD    25 0E FE 44 28 3A FE 1F
0E80  28 E3 FE 3E 20 01 23 FE    3C 20 01 2D FE 52 28 22

0E90  FE 4E 28 34 FE 57 28 A6    FE 49 20 DB CD 25 0E FE
0EA0  33 28 D4 E5 4E 77 23 79    B7 20 F9 77 23 77 E1 23


0EB0  18 EA 21 FF 0E 2B 18 BF    E5 DD E1 DD 7E 01 DD 77
0EC0  00 B7 28 B3 DD 23 18 F3    7E B7 28 AB 23 FE 1F 20

0ED0  F7 18 A4 EF 6E 70 75 74    1F 00 21 FD 0E 23 CD 25
0EE0  0E FE 3B CA 3A 0E 77 FE    1D 20 F2 2B 18 F0 D4
```

Execute from 0C60. Program starts at 0EFF.

I'm Pilot, fly me

D. Straker

HOW would you like to teach your wife/girlfriend '(substitute boss/teacher if applicable—ed)' etc., to write programs in half an hour? Impossible? Not if it's Pilot—and it's no idiot language either. It was started in 1971, as a language to be used for CAI (Computer Assisted Instruction) programming, and has, since then, grown both in the number of users—and the number of versions available. This account does not set out to set any standards or describe a complete language—it's intention is to whet the appetite of the programmer. If it looks o.k. to you, why not find out more, (or even add your own instructions), and write your own compiler/interpreter? It's been done in Basic and assembler before, and would make an excellent introduction to writing your own language!

Pilot is a text-oriented language, and hence the text gets a major share of the action. Instructions are one or two letters, and are separated from the text by a colon and a space. The text also does not need annoying quotes around them.
For example:

```
*LABELA
T:  Welcome to the Liverpool Software Gazette!
T:  What do you think of the show so far?
A:
M:  No!Terrible!Rubbish!
TY: I'm sorry, I didn't quite hear that,
TY: I'll ask the question again.
JY: LABELA
TN: It is rather splendid, isn't it!
J:  NEXTA
```

These few lines illustrate well the heart of the language, and once understood, they may be used to write a complete program. Let's look at them one by one:

(a) *LABELA—any line may be labelled by putting as asterisk in the first column (of course the label name must be unique within the program!) 6 letters is a common limit.

(b) T:—the most important instruction of all. It means type, or text, and can be used to display virtually anything.

(c) A:—Accept stops the program and waits for the user to input something.

(d) M:—Match provides Pilot with its unique ability to accept a large assortment of input data. This statement will allow: no, not, terrible, rubbish, (also nothing, knotted, etc.). The exclamation mark separates the options, and each option is looked for, in the reply to the last A: statement, not as a separate word, but as a character string. In effect, a 'window' is passed over the reply, looking for matches with the options given.

(e) TY:—This is not a new instruction, but the type of instruction with a conditioner in front of it. The text given is only displayed if the conditioner is true. The Y conditioner (yes) looks to see if the last M: statement did indeed find a match, and allows the statement to be obeyed only if a match was found. Hence, in this example, if the reply was no, nothing, terrible, rubbish, etc., then the program will type:

'I'm sorry, I didn't quite hear that,
I'll ask the question again.'

(f) JY:—Nothing to do with Jimmy Young, this is again an instruction with a conditioner. Jump is yes jumps to the label given if the last match was found, so this program jumps back to ask the initial question again, if an unfavourable reaction is given.

(g) TN:—Type is no is the opposite of TY:, hence in this example, if no match is found in the M: statement, the text is displayed:

'It is rather splendid isn't it!'

(h) J:—The unconditional jump cause a jump to the label specified, so this will jump to NEXTA.

And that is all there is to it!—You now can go and write your own Pilot programs using these few instructions.

More instructions may be added, and a few more will now be described:

Remarks may be added to aid clarity when reading the code. They are totally ignored when the program is running. The instruction is simply R:, followed by the remark.

Subroutines may be included, and start with a label, and end at the first return instruction, E: , that is met. A subroutine is called by U: , followed by the label name at the start of the routine. At the end of a subroutine, program control is returned to the instruction after the U: that called the routine.

Simple arithmetic may be done with the computer instruction, C:, where variables may be assigned values, so

```
C:  J = 2
sets J to 2, and

C:  K = K + 1
increments K by 1
```

These variables may be used in conditions, much as the Y or No shown earlier, so

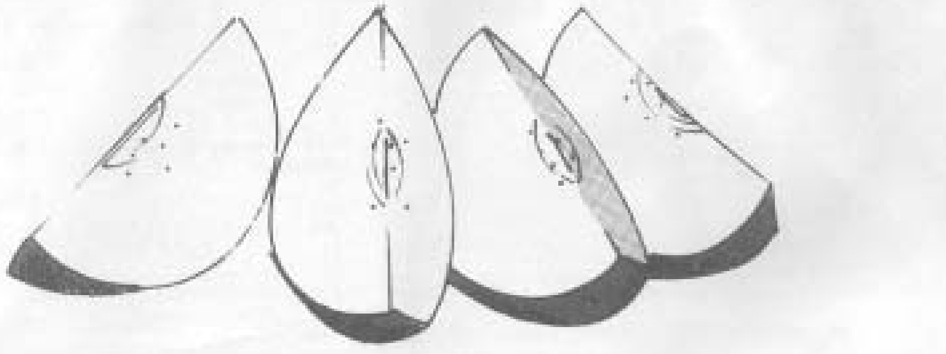
```
T (K>3):  Hello
will type 'Hello' only if K is greater than 3
```

These instructions allow freater flexibility, and this last example illustrates their use, along with the use of string variables. The full extent of Pilot has still not been explored, but if you have found the idea exciting, go out and find more on it, and when you have got an implementation working, why not write an article for this journal about it?

```
T:    Welcome to LSG Pilot
T:    Wasn't it easy to learn?
A:
M:    Yes!Definite!Very
TN:   Did you read it carefully enough?  Anyway,
T:    let's see what you can remember ...
T:    By the way, what is your name?
A:    $N
T:    Thanks, $N, now what was the compute instruction?
A:
M:    C:
TY:   Correct!
UN:   CONRAD
C:    B = 0
T:    How about a subroutine call?
*SUBROU
M:    U:
TY:   Good!
JY:   END
C:    B = B + 1
T:    Try again!
J (B  4):  SUBROU
T:    It's no good $N, the answer is U:
*END
T:    Thanks for playing, $N, 'bye for now!
J:  FINISH
*CONRAD
T:    I'll give you a clue - it rhymes with me - try again
A:
M:    D!D!E!G!P!T!V
TY:   Wrong one!
TY:   The answer's C:
M:    C:
TY:   That's better
E:
*FINISH
```

Apple Pips

C.Phillips



Apple Pips

A monthly selection of unclassifiable routines, hints, comments etc., for the Apple. Contributions are welcome!

Sound

PREAD (FBIE) is a subroutine in the Apple monitor which delays according to the value of the Apple's analogue and inputs.

Load x register with required input (Ø-3)

eg. the following routine will produce tones of varying pitch by altering PADDLE Ø.

```
$3ØØ    A2  ØØ        LDX   #$ØØ    ; PDL Ø
$3Ø2    2Ø  1E  FB    JSR   $FB1E   ; PREAD
$3Ø5    8D  3Ø  CØ    STA   $CØ3Ø   ; Toggle Speaker
$3Ø8    4C  ØØ  Ø3    JMP   $3ØØ    ; Start Over
```

Decimal to Hex Conversion (Requires Applesoft in ROM)

In Applesoft the & character causes an unconditional jump to $3F5. By vectoring to a suitable address and continuing we can extend the available repertoire of Applesoft functions indefinitely.

For example the following routine will evaluate any arbitary Basic expression and return the answer in hex.

```
To use type & <expression> <return> in immediate mode
     or   line no.   & <expression>   return in a program
eg.  & 10 <return> gives ØØØA.
     & 10 + 6 <return> gives ØØ1Ø.
     & 12/2 + 4 <return> gives ØØØA.
     & - 1 <return> gives FFFF.
     & ASC ("A") <return> gives ØØ41.
```

Should the expression give a range error the routine gives 'illegal quantity error'. If the expression is invalid 'Syntax error'.

```
3ØØ:  2Ø  67  DD  JSR  FRMNUM
3Ø3:  2Ø  52  E7  JSR  INTEGER
3Ø6:  A6  5Ø      LDX  $5Ø
3Ø8:  A5  51      LDA  $51
3ØA:  4C  41  F9  JMP  PRNTAX

3F5:  4C  ØØ  Ø3  JMP  $3ØØ
```

Once entered the routine resides happily with any Basic program and is not erased by New, Load, Save, delete etc. (Re-booting the DOS does clobber it).

To save on disk:

```
BSAVE DECHEX, A$ 3ØØ, L$F7   return
To use simply BLOAD DECHEX (do not BRUN)
```

Integer Basic to Applesoft Conversion

This short routine for Disk II users will convert on integer basic program text to Applesoft. Note that it does not correct for any syntax differences between the two languages. It is in Integer Basic.

```
1Ø D$=" ": REM CTRL D:DIM TITLE$ (3Ø)

2Ø INPUT "PROGRAM TITLE ",TITLE$

3Ø POKE 76, PEEK  2Ø2  : POKE 77

   , PEEK  2Ø3

4Ø PRINT D$ "LOAD ",TITLE$

5Ø PRINT D$, "OPEN ",TITLE$;" .TEX"

6Ø PRINT D$, "WRITE ",TITLE$;",TEX"

7Ø PRINT "FP"

8Ø LIST

9Ø PRINT D$, "CLOSE ",TITLE$;" . TEX"

1ØØ PRINT D$, "EXEC ",TITLE$;" . TEX"

11Ø END
```

APPLES' MINI-ASSEMBLER

TRYING to use the mini-assembler buried deep in Apples' firmware? Going crazy, typing every possible permutation of 'F666G'' and watching the machine crash? Cursing the retailer who has evidently sold you a defective ROM? Do you, by any chance, have an Applesoft Card plugged into Slot #Ø ? When ROM Applesoft is selected, it resides in memory from D000.F7FF—thereby replacing Integer Basic, the mini-assembler, floating point, and Sweet 16 firmware in the memory map.

So, to access these utilities use either:—

i) <reset> CØ8Ø <return> – Turns Applesoft Card off, under Software Control

F666G <return> – Enter mini-assembler

Or

ii) <Switch Applesoft Card off> – (Move switch down)
<reset>
F666G <return>

The assembler prompts with an "!". Since it is a one-pass tiny assembler symbolic addressing is not supported.—Syntax follows that of the Apple dis-assembler (MOS technology with minor differences); all numbers are assumed to be hex, therefore, use of the conventional dollar sign is unnecessary. Instructions that manipulate the accumulator have a blank in the operand field. Page zero references generate the correct two-byte instructions. When using relative branches, the destination address is entered and the two's complement value calculated and inserted by the Apple. To actually enter the source line type:—

<Start address:> <Source> <return>

<Start address:> is optional, if omitted type a space before entering the line. Assembly will continue at the current address.

The assembler echoes your source line with the relevant-hex bytes inserted. Should you make an error, the Apple refuses the instruction, sounds the bell, and prints an error pointing to the statement in question. Current address references are unchanged.
<$ Monitor Command> allows the execution of monitor commands with return to the mini-assembler—useful for disassembling to see where you are up to, or saving programs on tape.

The First National Meeting of the U.K. Apple User's Association.

Dr. Martin Beer.

The U.K. Apple Users' Association met for the first time, in London, on 25th September. This meeting was called to discuss the future organisation of the Association, to discuss and approve a proposed constitution and to elect officers for the forthcoming year. The Association has, so far, been sponsored by Dr. Tim Keen and Andy Witterick of Keen Computers Ltd. in Nottingham, whose not inconsiderable efforts have been rewarded with a founder membership of over eighty.

Dr. Tim Keen took the chair at the start of the meeting, which immediately discussed the problems of servicing its widely spread and diverse membership. The meeting felt that member's interests would be best served by the establishment of Local Area Groups in various parts of the country, and, if necessary, of Special Interest Groups to cover particular subjects. The need was expressed immediately for an ITT Special Interest Group, to provide help and information to owners and users of that machine. It was anticipated that most members would wish to belong to their local group, but that special arrangements should be made for those members who because of distance, or any other reason, do not wish to join one.

The new constitution was then proposed and accepted with various minor amendments. The Association now has the following aims and objectives:

a. to promote the exchange of ideas, personnel and management techniques, information and practical experiences between Apple and allied computer systems, and between Users and Apple Computer Inc. as manufacturer and their suppliers, in order to increase the effectiveness of Apple computer systems.

b. to enable Users to agree joint recommendations to Apple Computers Inc. for the development or improvement of Apple Computers Inc. products and services.

The Association is to be run by an Executive Committee of eight members which will meet regularily to organise the day-to-day running, and a Council, which will consist of the Executive Committee and representatives of the various groups, and meet at least twice a year to discuss policy issues. It is hoped, also to organise an annual Association meeting. Dr. Keen was elected the first Chairman, and Andy Witterick the first Secretary.

Merseyside Apple Group

We have already started an Apple Special Interest Group on Merseyside, as part of the Merseyside Microcomputer Group. We meet regularily at 7.00 p.m. on the third Thursday of every month at Riversdale College. The main purpose of the local groups is to meet other users and to discuss ideas, projects, problems etc. in a friendly and informal atmosphere. We normally have several Apples available for members to demonstrate their programs, and try out the latest products.

Whilst in London I was able to try the new PASCAL system very briefly. I was most impressed with the facilities provided. Not only is a full PASCAL compiler and operating system provided, but also a very useful relocatable macro-assembler. The operating system consits of a series of programs such as the compiler, the editor, the assembler and the file handler which are called in from disc when requested from the menu. This allows considerably more facilities to be provided than is possible with a fully resident system. A number of demonstration programs are included with the system on a separate disc which show the power and versitility of the system.

No doubt other programs will be written by users very soon. Since the turtle graphics works in the same way as an incremental plotter, by the programmer specifying the direction and length of the line, pattern and picture drawing are much easier. By booting the system with another disc the Apple reverts to running Integer and floating point BASIC and is fully compatible with your current system, so that all your programs can still be run without any hardware changes to the APPLE.

At first sight this is a very nicely organised and packaged system, which considerably increases the Apple's range and usefulness. I look forward to using the system seriously and to reviewing it in some detail at a later date.

The address of the Association is
The Secretary,
U.K. Apple Users Association,
5 The Poultry,
NOTTINGHAM.
My address is:
Dr. Martin Beer,
Computer Laboratory,
University of Liverpool.
Tel. 051-709-6022. Ext 2967.

ACORN MASTERMIND

Lawrence Hardwick

THIS programme plays the game of Bulls and Cows against the operator on an ACORN Microcomputer; although use is made of display and keyscan routines in the ACORN Monitor it is possible to adapt the programme for other 6502 based machines.

The programme maybe entered into the ACORN memory using the monitor in the normal way, to store it on tape locations 0200 to 03CC must be saved, the programme is executed from the label BEGIN at 02CC.

Subroutines

The main programme calls several subroutines given at the start of the programme listing;

MATCH — Calculates the number of Bulls and Cows that should be awarded for a comparison between two four-digit numbers. These numbers are stored in page zero at NUMA and NUMB, and the result is returned in the accumulator.

UNPACK — Takes the bottom twelve bits of the two bytes pointed to by register Y, and stores them three bits at a time in the location pointed to by X, i.e. at X, X 1, X 2 and X 3. (This is used to prepare numbers for the MATCH routine).

DISRAN — Displays the current contents of the display buffer using the Monitor scan routine in a single scan mode. Between each scan the routine cycles a pseudo-random sequence generator consisting of a fed-back shift register. This shift register stored at locations, RAN, RAN 1 and RAN 2 is twenty-three bits long with feedback from bits twenty-two and seventeen. The cycle of numbers generated will repeat every eight million shifts so the numbers generated in the bottom twelve bits of the register are fairly random.

MSSAGE — Puts the message in the message table at the end of the programme, pointed to by X, into the display buffer.

QOCTFE — Works much the same way as QDATFET in the ACORN Monitor, but fetches four octal numbers input from the keyboard and stores them in the packed form in the locations pointed to by the X register.

QOCTTD — Takes four octal digits in the packed form pointed to by X and puts their segment codes into the display buffer for the ACORN scan routine to display.

Main Programme

The method of the programme is described in the flow chart and by comments in the programme listing; the important part is NEWGU which tests to see if the programme's attempt at a guess is consistent with the information it has about its previous guesses. If the guess is consistent it is displayed, if not, a new attempt is made. Although this algorithm is not particularly efficient it is quick to notice if its opponent has cheated.

Playing Bulls and Cows

After the programme has been entered the display will show: rEAdY

—pressing any key will change the display to show four digits. The player now enters his first guess, the programme will only accept digits in the range 0 to 7 and subtracts eight from any other digits to bring it into this range. Any control key will terminate this entry which may be over-written until terminated.

In response to the control key the display may under very rare circumstances show:

YOU WIN

—otherwise two more digits will appear. The first digit indicates the number of Bulls (correct digit, correct position) and the second digit is the number of Cows (correct digit, incorrect position).

Pressing any control key will now cause the computer to display a four digit number and two dashes; the number is the computers guess at the players secret number and the dashes are a prompt for the player to provide the computer score which can now be entered as two digits, Bulls first again corrections may be overwritten until the entry is terminated by pressing any control key.

If four Bulls were scored the computer will respond rather obviously with the display:

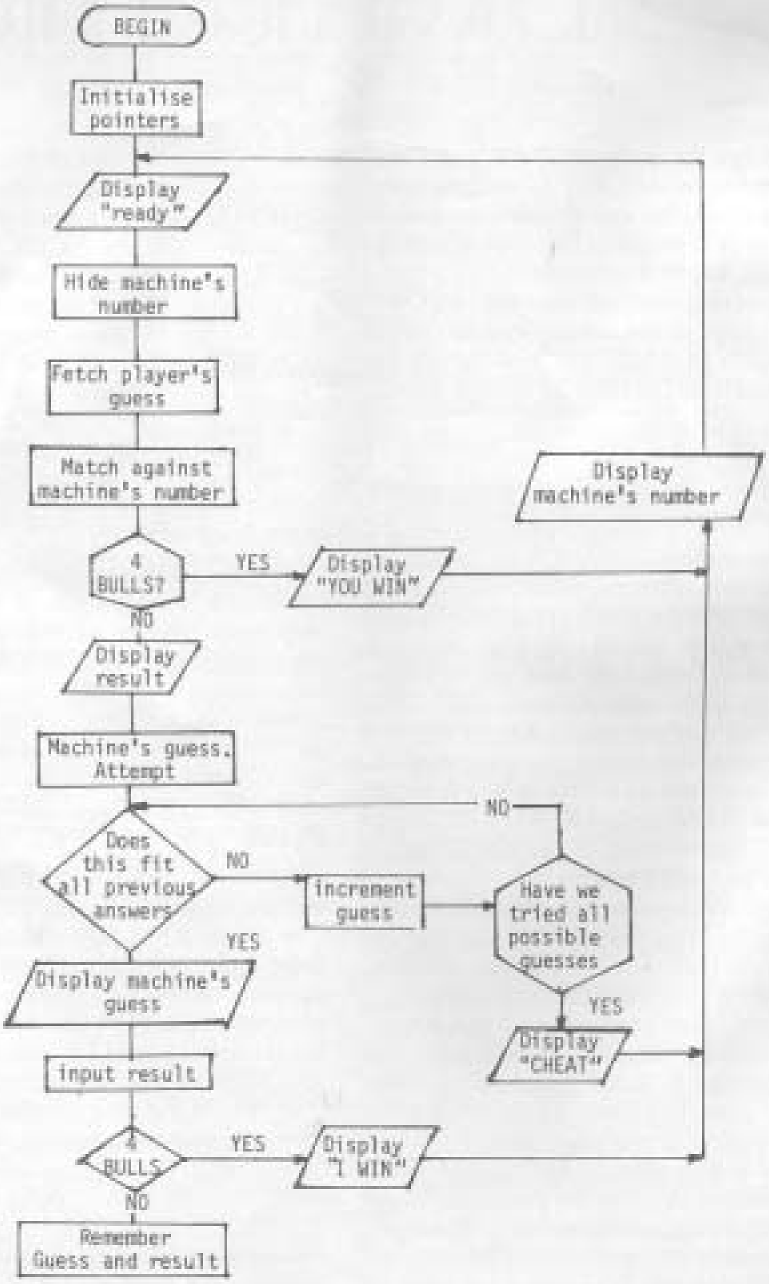
1 WIN

—otherwise the players previous guess will be displayed and his next attempt can be entered and terminated as before.

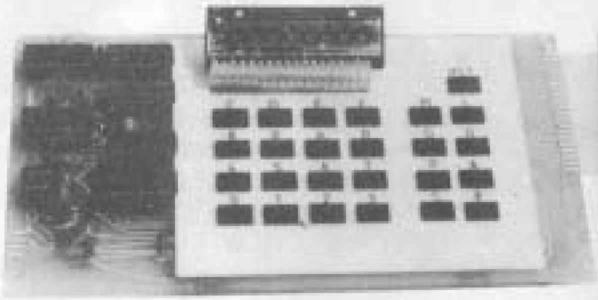
If the computer recognises that no number corresponds to the information that it has been given whether caused by an innocent oversight on the part of the player or by his hopeful dishonesty the computer will quite unequivocally display:

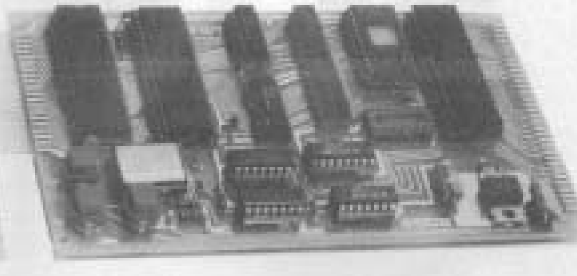
CHEAT

After any of these game-ending displays a further key depression will cause the computer to display its own secret number and one more key depression will cause READY to be displayed for the start of a new game.



Acorn at Microdigital





This compact stand-alone micro-computer is based on Eurocard modules and employs the highly popular 6502 MPU (as used in Apple Pet Kim etc) throughout the design philosophy has been provide full expandability, versatility and economy. Take a look at the full specifications and see how Acorn meets your requirements.

Acorn Technical specification

The Acorn consists of two single Eurocards:

1. MPU card 6502 microprocessor 512 x 8 ACORN monitor 1K x 8 RAM: 15-way I/O with 128 bytes of RAM: 1 MHz crystal: 5V regular sockets for 2K EPROM and second RAM I/O chip.
2. Keyboard card: 25 click-keys (16 hex, 9 control) 8 digit 7 segment display CUTS standard crystal controlled tape interface circuitry.

Compact, easy to use Acorn monitor includes the following features

- System program
- Set of sub-routines for use in programming
- Powerful de-bugging facility displays all internal registers
- Tape load and store routines

acorn Operating Manual

With Acorn, you'll receive an operating manual that covers computing in full, from first principles of binary arithmetic, to efficient hex programming with the 6502 instruction set. The manual also includes a listing of the monitor programs and the instruction set, and other useful tabulations plus sample programs.

	Nett	V.A.T.	Total
Kit	65.00	9.75	74.75
Ready Built	75.00	11.25	86.25

Acorn Memory

A high quality fibre glass through hole plated PCB with solder resist and component identification, this eurocard has provision for 8K of RAM (2114) and 8K of EPROM (2732).

The card is fully buffered for use with any system but has the advantage that the inputs of all cards except that being accessed are tri-stated and present no load to the bus, thus up to 4 cards may be directly connected to the bus before further buffering has to be added to the back plane. The memory card is the natural first step in expansion of the Acorn system and provides storage and working memory for the Acorn 4K fast basic.

	Nett	V.A.T.	Total
8K RAM (Kit)	95.00	14.25	109.25

Acorn V.D.U.

The Acorn V.D.U. Board connects to the Acorn Computer Bus and contains memory mapped character storage RAM which is transparently written to or read from by the C.P.U.

An MC 6845 programmable controller I.C. provides all the synchronisation signals to drive a 625 line 50 fields per second V.D.U. together with read addresses for the character R.A.M. Characters are then fed to an SAA5050 character generator IC which produces the necessary dot patterns to create the characters to refresh the V.D.U.

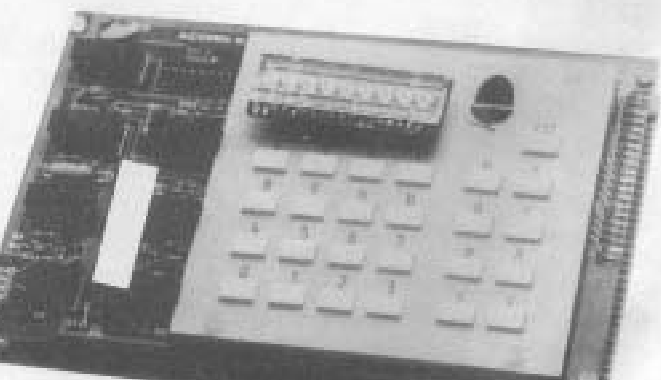
The SAA5050 produces Teletext standard characters and has Red, Green and Blue drive outputs giving coloured characters or graphics.

The R.G.B. and sync outputs may be used to drive a colour encoder and modulator for a U.H.F. Television, also provided is a 1 volt/75 ohm composite sync and video output which can directly drive a Monochrome Monitor on which the different colours will appear as different scales of grey.

The V.D.U. controller P.C.B. is supplied in kit form with a full set of I.C. sockets. The board operates from a single +5v supply from which it draws not more than 500 mA.

A new monitor R.O.M. will shortly be available for linking the V.D.U. and an ASCII keyboard to Acorns 4K Fast BASIC.

	Nett	V.A.T.	Total
V.D.U. Controller (Kit)	88.00	13.20	101.20



MICRODIGITAL

25 Brunswick Street, Liverpool L2 0PJ
Tel: 051-236 0707 (24 hour Mail Order)
051-227 3535 (All other Depts.)
Mail orders to: MICRODIGITAL LIMITED, FREEPOST (No Stamp Required) Liverpool L2 2AB

```
MASTER     ACORN 6502 Assembler   Page 02
0570: 0246 66 42         ROR    TEMPA  2 BYTE 3 BIT ROTATE
0580: 0248 6A            RORA
0590: 0249 66 42         ROR    TEMPA
0600: 024B 6A            RORA
0610: 024C 66 42         ROR    TEMPA
0620: 024E 6A            RORA
0630: 024F E8            INX           NEXT DIGIT
0640: 0250 88            DEY           Y IS A COUNTER
0650: 0251 D0 EB         BNE    UNLOOP ROUND AGAIN
0660: 0253 60            RTS           AND RETURN
0670: 0254 A9 1F  DISRAN LDAIM  $1F    SET SINGLE SCAN
0680: 0256 85 0E         STAZ   $0E
0690: 0258 20 0C FE DESCAN JSR  $FE0C  MONITOR SCAN CALL
0700: 025B 49 1F         EORIM  $1F     KEY?
0710: 025D D0 11         BNE    KEYFD       YES
0720: 025F A5 24         LDA    RAN    +02 GENERATE RANDOM
0730: 0261 29 42         ANDIM  $42    NUMBERS  NEXT BIT IN
0740: 0263 69 3E         ADCIM  $3E     BIT SIX OF ACC
0750: 0265 0A            ASLA          AND PUT IN CARRY
0760: 0266 0A            ASLA
0770: 0267 26 22         ROL    RAN     NOW ROTATE THE BITS
0780: 0269 26 23         ROL    RAN    +01 ROUND THE 3 BYTES
0790: 026B 26 24         ROL    RAN    +02
0800: 026D 4C 58 02      JMP    DESCAN AND ROUND AGAIN
0810: 0270 90 01  KEYFD  BCC    NORET  CONTROL KEY?
0820: 0272 60            RTS           YES SO RETURN
0830: 0273 A5 3F  NORET  LDA    ANSWER  DIGIT KEY SO
0840: 0275 0A            ASLA          ASSEMBLE NEW ANSWER
0850: 0276 0A            ASLA          LAST DIGIT UP 4 BITS
0860: 0277 0A            ASLA
0870: 0278 0A            ASLA
0880: 0279 05 0D         ORA    KEY     PUT IN NEW DIGIT
0890: 027B 85 3F         STA    ANSWER  STORE IN ANSWER
0900: 027D 20 60 FE      JSR    $FE60   ACCUMULATOR TO DISP
0910: 0280 4C 58 02      JMP    DESCAN  AND ROUND AGAIN
0920: 0283 A9 FF  MSSAGE LDAIM  $FF    MESSAGE TO DISP
0930: 0285 85 0E         STAZ   $0E    SET SCAN MODE FOR QOCTFE
0940: 0287 86 20         STX    MESSPO  SET UP POINTER
0950: 0289 A0 07         LDYIM  $07     8 DIGITS TO FETCH
0960: 028B B1 20  MLOOP  LDAIY  MESSPO POST INDEX FETCH
0970: 028D 99 10 00      STAAY  $0010   PUT IN DISPLAY BUFF
0980: 0290 88            DEY           NEXT DIGIT
0990: 0291 10 F8         BPL    MLOOP   ROUND AGAIN
1000: 0293 60     SUBRET RTS           OR RETURN
1010: 0294 20 AE 02 QOCTFE JSR  QOCTTD  DISPLAY OLD
1020: 0297 20 0C FE      JSR    $FE0C   MONITOR SCAN CALL
1030: 029A B0 F7         BCS    SUBRET  CONTROL KEY RETURN
1040: 029C A0 03         LDYIM  $03     3 BITS TO SHIFT
1050: 029E 29 07         ANDIM  $07     KEYS RANGE 0-7
1060: 02A0 16 01  SHIFT  ASLZX  $01     THIS IS THE 3
1070: 02A2 36 00         ROLZX  $00     BIT SHIFT
1080: 02A4 88            DEY
1090: 02A5 D0 F9         BNE    SHIFT
1100: 02A7 15 01         ORAZX  $01     PUT NEW KEY IN
1110: 02A9 95 01         STAZX  $01     STORE NEW ENTRY
1120: 02AB 4C 94 02      JMP    QOCTFE  AND ROUND AGAIN
```

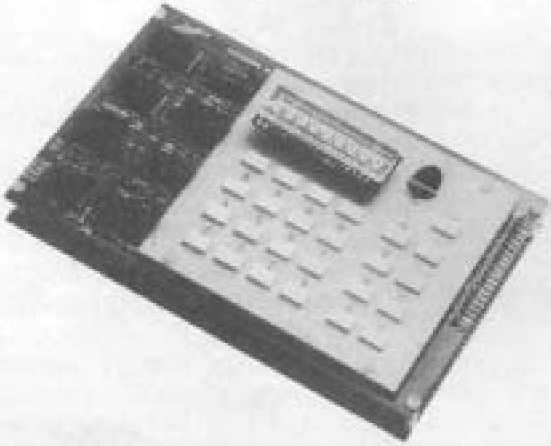
```
MASTER     ACORN 6502 Assembler    Page 03

1130: 02AE A0 04      QOCTTD LDYIM $04     4 OCTAL
1140: 02B0 B5 00             LDAZX $00     DIGITS TO DISPLAY
1150: 02B2 85 42             STA   TEMPA   USE TEMPA AND TEMPB
1160: 02B4 B5 01             LDAZX $01
1170: 02B6 85 43      DISLOP STA   TEMPB   SAVE LOWER BYTE
1180: 02B8 29 07             ANDIM $07     MASK DIGIT
1190: 02BA 20 7A FE          JSR   $FE7A   DIGIT TO DISPLAY BUFF
1200: 02BD A5 43             LDA   TEMPB   RELOAD LOWER BYTE
1210: 02BF 66 42             ROR   TEMPA   NOW 3 BIT 2 BYTE
1220: 02C1 6A                RORA         ROTATE
1230: 02C2 66 42             ROR   TEMPA
1240: 02C4 6A                RORA
1250: 02C5 66 42             ROR   TEMPA
1260: 02C7 6A                RORA
1270: 02C8 88                DEY          NEXT DIGIT
1280: 02C9 D0 EB             BNE   DISLOP  AND ROUND AGAIN
1290: 02CB 60                RTS          OR RETURN
1300: 02CC A9 FF      BEGIN  LDAIM $FF
1310: 02CE 85 22             STA   RAN
1320: 02D0 A9 44      START  LDAIM STACK   RESET STACK END
1330: 02D2 85 40             STA   GSEND
1340: 02D4 A9 03             LDAIM READY  /  SET MESS POINTER
1350: 02D6 85 21             STA   MESSPO +01
1360: 02D8 A2 A7             LDXIM READY   MESSAGE READY
1370: 02DA 20 83 02          JSR   MSSAGE
1380: 02DD 20 54 02          JSR   DISRAN  DISPLAY "READY"
1390: 02E0 A5 23             LDA   RAN    +01  PUT RANDOM NUMBER
1400: 02E2 85 26             STA   MYNO   +01  AS MY NUMBER
1410: 02E4 A5 22             LDA   RAN
1420: 02E6 29 0F             ANDIM $0F
1430: 02E8 85 25             STA   MYNO
1440: 02EA A2 C2      YOUGO  LDXIM BLANK   CLEAR DISPLAY
1450: 02EC 20 83 02          JSR   MSSAGE
1460: 02EF A9 FF             LDAIM $FF     SET SCAN MODE
1470: 02F1 85 0E             STAZ  $0E
1480: 02F3 A2 27             LDXIM YGU     FETCH YOUR GUESS
1490: 02F5 20 94 02          JSR   QOCTFE
1500: 02F8 A2 29             LDXIM NUMA    MY NUMBER TO NUMA
1510: 02FA A0 25             LDYIM MYNO
1520: 02FC 20 34 02          JSR   UNPACK
1530: 02FF A2 2D             LDXIM NUMB    YOUR NUMBER TO NUMB
1540: 0301 A0 27             LDYIM YGU
1550: 0303 20 34 02          JSR   UNPACK
1560: 0306 20 00 02          JSR   MATCH   AND COMPARE THEM
1570: 0309 C9 40             CMPIM $40     FOUR BULLS !!?
1580: 030B D0 18             BNE   NOWIN   PHEW !!
1590: 030D A2 B4             LDXIM YOUWIN DRAT YOU
1600: 030F 20 83 02   ENDOUT JSR   MSSAGE  END OF GAME
1610: 0312 20 54 02          JSR   DISRAN  DISPLAY, MESSAGE
1620: 0315 A2 C2             LDXIM BLANK   CLEAR DISPLAY
1630: 0317 20 83 02          JSR   MSSAGE
1640: 031A A2 25             LDXIM MYNO    DISPLAY MY NUMBER
1650: 031C 20 AE 02          JSR   QOCTTD
1660: 031F 20 54 02          JSR   DISRAN
1670: 0322 4C D0 02          JMP   START   READY TO PLAY AGAIN
1690: 0325 20 60 FE   NOWIN  JSR   $FE60  MONITOR ACC TO DISPLAY
```

```
MASTER     ACORN 6502 Assembler     Page 04
1700: 0328 20 54 02          JSR   DISRAN  DISPLAY BULLS/COWS
1710: 032B A5 22             LDA   RAN     RANDOM NUMBER IS MY GUESS
1720: 032D 29 0F             ANDIM $0F     AND REMEMBER WHERE WE
1730: 032F 85 3B             STA   MYGU      START
1740: 0331 85 3D             STA   STRT
1750: 0333 A5 23             LDA   RAN    +01
1760: 0335 85 3C             STA   MYGU   +01
1770: 0337 85 3E             STA   STRT   +01
1780: 0339 A0 3B     NEWGU   LDYIM MYGU    MY NUMBER
1790: 033B A2 2D             LDXIM NUMB    UNPACKED TO NUMB
1800: 033D 20 34 02          JSR   UNPACK
1810: 0340 A0 44             LDYIM STACK   RESET GUESS POINTER
1820: 0342 C4 40     NEWINF  CPY   GSEND   END OF STACK?
1830: 0344 84 41             STY   GUNO    STORE GUESS POINTER
1840: 0346 F0 30             BEQ   FOUND   YES STACK FINISHED
1850: 0348 A2 29             LDXIM NUMA    STACKED GUESS
1860: 034A 20 34 02          JSR   UNPACK  UNPACKED TO NUMA
1870: 034D 20 00 02          JSR   MATCH   COMPARE NEW ANSWER
1880: 0350 A4 41             LDY   GUNO    WITH OLD ANSWERS
1890: 0352 D9 02 00          CMPAY $0002
1900: 0355 D0 05             BNE   NOGOOD  DOES NOT FIT
1910: 0357 C8                INY          NEXT STACK ENTRY
1920: 0358 C8                INY
1930: 0359 C8                INY
1940: 035A D0 E6             BNE   NEWINF  TRY THIS ENTRY
1950: 035C E6 3C     NOGOOD  INC   MYGU   +01  INCREMENT
1960: 035E D0 08             BNE   NOTUP   MY GUESS AS THE LAST
1970: 0360 E6 3B             INC   MYGU    ONE WAS NO GOOD
1980: 0362 A5 3B             LDA   MYGU
1990: 0364 29 0F             ANDIM $0F
2000: 0366 85 3B             STA   MYGU
2010: 0368 A5 3C     NOTUP   LDA   MYGU   +01  IF WE COUNT
2020: 036A C5 3E             CMP   STRT   +01  ROUND TO THE START
2030: 036C D0 CB             BNE   NEWGU   THEN SOMEBODY IS
2040: 036E A5 3B             LDA   MYGU    CHEATING OTHERWISE
2050: 0370 C5 3D             CMP   STRT    TRY THIS NEW GUESS
2060: 0372 D0 C5             BNE   NEWGU
2070: 0374 A2 BC             LDXIM CHEAT   YOU ROTTER
2080: 0376 D0 97             BNE   ENDOUT  END OF GAME
2090: 0378 A5 3B     FOUND   LDA   MYGU    PUT THIS GOOD
2100: 037A 99 00 00          STAAY $0000   ON THE STACK
2110: 037D A5 3C             LDA   MYGU   +01
2120: 037F 99 01 00          STAAY $0001
2130: 0382 A2 C4             LDXIM PROMPT  "......__"TO DISP
2140: 0384 20 83 02          JSR   MSSAGE
2150: 0387 A2 3B             LDXIM MYGU    MY GUESS TO DISPLAY
2160: 0389 20 AE 02          JSR   BOCTTD
2170: 038C 20 54 02          JSR   DISRAN  USE DISRAN TO GET ANSWER
2180: 038F A5 3F             LDA   ANSWER
2190: 0391 C9 40             CMPIM $40     4 BULLS? I WIN
2200: 0393 D0 05             BNE   NOIWIN  NOT YET I DONT
2210: 0395 A2 AD             LDXIM IWIN    MESSAGE AND ENDGAME
2220: 0397 4C 0F 03          JMP   ENDOUT
2240: 039A A4 41     NOIWIN  LDY   GUNO   PUT ANSWER ON STACK
2250: 039C 99 02 00          STAAY $0002
2260: 039F C8                INY          UPDATE STACK END
```

```
MASTER     ACORN 6502 Assembler    Page 05

2270: 03A0 C8                  INY
2280: 03A1 C8                  INY
2290: 03A2 84 40               STY   GSEND
2300: 03A4 4C EA 02            JMP   YOUGO    AND ROUND AGAIN
2310: 03A7 00         READY    =     $00
2320: 03A8 50                  =     $50
2330: 03A9 79                  =     $79
2340: 03AA 77                  =     $77
2350: 03AB 5E                  =     $5E
2360: 03AC 6E                  =     $6E
2370: 03AD 00         IWIN     =     $00
2380: 03AE 00                  =     $00
2390: 03AF 06                  =     $06
2400: 03B0 00                  =     $00
2410: 03B1 1C                  =     $1C
2420: 03B2 04                  =     $04
2430: 03B3 54                  =     $54
2440: 03B4 00         YOUWIN   =     $00
2450: 03B5 6E                  =     $6E
2460: 03B6 3F                  =     $3F
2470: 03B7 3E                  =     $3E
2480: 03B8 00                  =     $00
2490: 03B9 1C                  =     $1C
2500: 03BA 04                  =     $04
2510: 03BB 54                  =     $54
2520: 03BC 00         CHEAT    =     $00
2530: 03BD 39                  =     $39
2540: 03BE 76                  =     $76
2550: 03BF 79                  =     $79
2560: 03C0 77                  =     $77
2570: 03C1 78                  =     $78
2580: 03C2 00         BLANK    =     $00
2590: 03C3 00                  =     $00
2600: 03C4 00         PROMPT   =     $00
2610: 03C5 00                  =     $00
2620: 03C6 00                  =     $00
2630: 03C7 00                  =     $00
2640: 03C8 00                  =     $00
2650: 03C9 00                  =     $00
2660: 03CA 08                  =     $08
2670: 03CB 08                  =     $08
```



Pascal bytes the Apple

C. Phillips

THE traditional bugbear of the microcomputer has been an almost complete lack of system software, with the only available programming language Basic unsuited to a wide variety of potential tasks. Basic is a superficially attractive way of programming a computer, its friendly, forgiving interactive nature plus its apparent simplicity mean simple programs are easily written and debugged. As a tool for more serious development work however Basic leaves a lot to be desired—much of computer science emphasises the need for a top down structured approach to problem solution, Basic on the other hand is unstructured and inconsistent (no real attempt is made at standardisation between implementations and the numerous 'Ad Hoc' extensions make life difficult for any programmer). The programming language Pascal has been hailed by many as much closer to that ideal 'The Programming Language'. Pascal is a modern, structured, heavily typed language that embodies many of the present ideas of computer science.

Until recently much of the discussion had been largely academic—the wide availability of Basic made it a De Facto standard whereas few Pascal implementations existed for small machines. The situation changed however with the announcement by the Department of Information Science at The University of California San Diego, that they had Pascal implementations up and running on a number of microprocessor based machines used for teaching purposes. This Pascal implementation is now available to the end user in a number of different guises for a number of different machines.

The Apple implementation is perhaps the most exciting development in that a complete Pascal system is available in a packaged, well documented form, at a relatively low cost.

The Pascal Language System consists of a fair amount of physical hardware viz:

1 x Apple Language Card
2 x Replacement Proms for Disk Controller Card
1 x I.C. Extractor (!)

5 x Systems Discs
- Apple O:
- Apple 1:
- Apple 2:
- Apple 3:
- Basics:

7 x System Manuals
- Applesoft Basic
- Applesoft Tutorial
- Integer Basic
- Pascal User Manual and Report
- Microcomputer Problem Solving Using Pascal
- Apple Pascal Reference Manual
- Apple Language System Installation and Operating Manual

Plus miscellaneous guarantees, errata sheets, bibliography, etc.

THE LANGUAGE CARD

The heart of the system is this plug-in card. On Board is an additional 16K of RAM, the 'Autostart' ROM and the usual chunk of TTL. Installation consists of plugging the card into slot £0, replacing a 4116 on the main Apple Board with a ribbon cable, and changing the two Proms on the Disc Controller Board.

USING THE SYSTEM WITH BASIC

The Language System works with any 48K Apple II, or Apple II Plus complete with one or more disc drives. The Basic and Pascal Systems are independent and incompatible with one another, existing files cannot be accessed by the Pascal system and it is necessary to re-boot the system when switching. Included with the 'Basic' portion of the system are the Apple Integer Basic and Applesoft Manuals, as well as a new 'volume' the Applesoft Tutorial. This is an excellent adaption of Jef Raskin's Integer Basic Manual.

To use either Basic the user inserts the 'Basics' Disc, switches on and when prompted inserts any existing 3.2. Disc. 'Autostart' entry into applications programs is no longer available using Basic—only Pascal. This apparent disadvantage is offset by a number of improvements in using Basic, firstly on switch on the system loaded the alternative Basic for your system (Apple II Owners get Applesoft, Apple II Plus Owners Integer Basic), into the RAM on the Language Card. Switching from Basic to Basic is accomplished instantaneously by typing "FP" or "INT" respectively and the appropriate RAM (write protected) or ROM is selected. Apple had the good sense to include the mini-assembler, sweet 16 and floating point routines along with the Integer Basic firmware loaded in from disc,for Apple II Plus users.

The existing F8ØØ ROM of Apple II users is replaced by the on-card 'Auto-Start' ROM in the Memory Map. This is a considerable improvement over its predecessor—it features dramatically improved On-Screen editing, and typing a (CTRL S) stops a listing or trace from flashing by (in fact the output routine simply halts on a (CR) and waits for a keystroke). The most debatable 'improvement' is 'Reset Key Protection'. On reset the Apple initialises and executes an indirect jump to location 03F2 in RAM.

Normally this is initialised as a warm start to Basic, so hitting reset is equivalent to < CTRL C > (however reset also clears variable values). In addition by changing the address to a suitable location it is possible for applications packages to retain control instead of landing the poor user in the middle of the system monitor (no more 'If you hit reset type 3DØ (Ø not 0) G return, then type 'Run' or GOTO 100 or whatever). The disadvantage comes if a rampant program should overwrite 03F2, it then becomes possible to crash the system so that you hvae to switch off and start over. Personally I feel the advantages outweigh the disadvantages but nevertheless it is uniquely irritating when it happens.

As a result of all this all existing Apple Software remains compatible (Apple II Plus owners can now run all that important Integer Basic Software like Startrek, Starwars without mods.). The only exception to this is if your program calls any part of the single-step simulator code or multiply/divide routines of the monitor which have been replaced by other subroutines in the F800 ROM (No software I know of does).

'AUTOSTART' CHANGES:

Deleted
Step=FA4Ø–FA85, FAA5–FAD6, FAD–FB18
Muplm, Divpm=FB6Ø–FBCØ

Moved:
IRQ/BREAK (FA86) is now at FA4Ø

Page 3 Vectors
Break Vector is at 3FØ. 3F1
Reset Vector is at 3F2. 3F3

USING THE SYSTEM WITH PASCAL

The Pascal System largely consists of the operating system, file handler, a 'window' text editor, the actual compiler, a linker and macro-assembler. A number of utilities and demonstration programs are included with the system.

Almost all of the system software assumes a screen width of 80 characters, Apples' 40 character screen therefore normally only shows the 'left page'. To see the other page the user switches with < CTRL A >.

While superficially unattractive I found the system worked well in practice; if required a full 80 x 24 upper and lower case terminal is supported via a communications card.

The operating system is largely menu driven with a prompt-line at the top of the screen indicating possible options. On booting the system a welcome message, the date the disk was last used, and this prompt line appears.
COMMAND:E(DIT),R(UN,F(ILE,C(OMP,L(INK,X(ECUTE,A(SSEM,D(EBUG,?

Typing the appropriate single letter will invoke the appropriate command. For example to use the editor the user types 'E'. To compile (if necessary) and execute a program 'R'. 'X' executes a codefile etc.

When a ? appears in a prompt line there are too many options to fit on the prompt line. Typing a '?' displays any remaining commands.

SYSTEM. WRK is a special default file used during program development or text editing. The workfile can be edited, compiled, saved, updated, or executed without the need to continually specify a filename. Most of the commands e.g. the editor automatically look for and load the workfile if it is present on the boot disk.

The operating system adds a suffix, depending on a files contents, of Text, Code or Data. For a program in the workfile there will usually be two files

SYSTEM WRK. TEXT Source Code
SYSTEM WRK. CODE Object Code

Filer

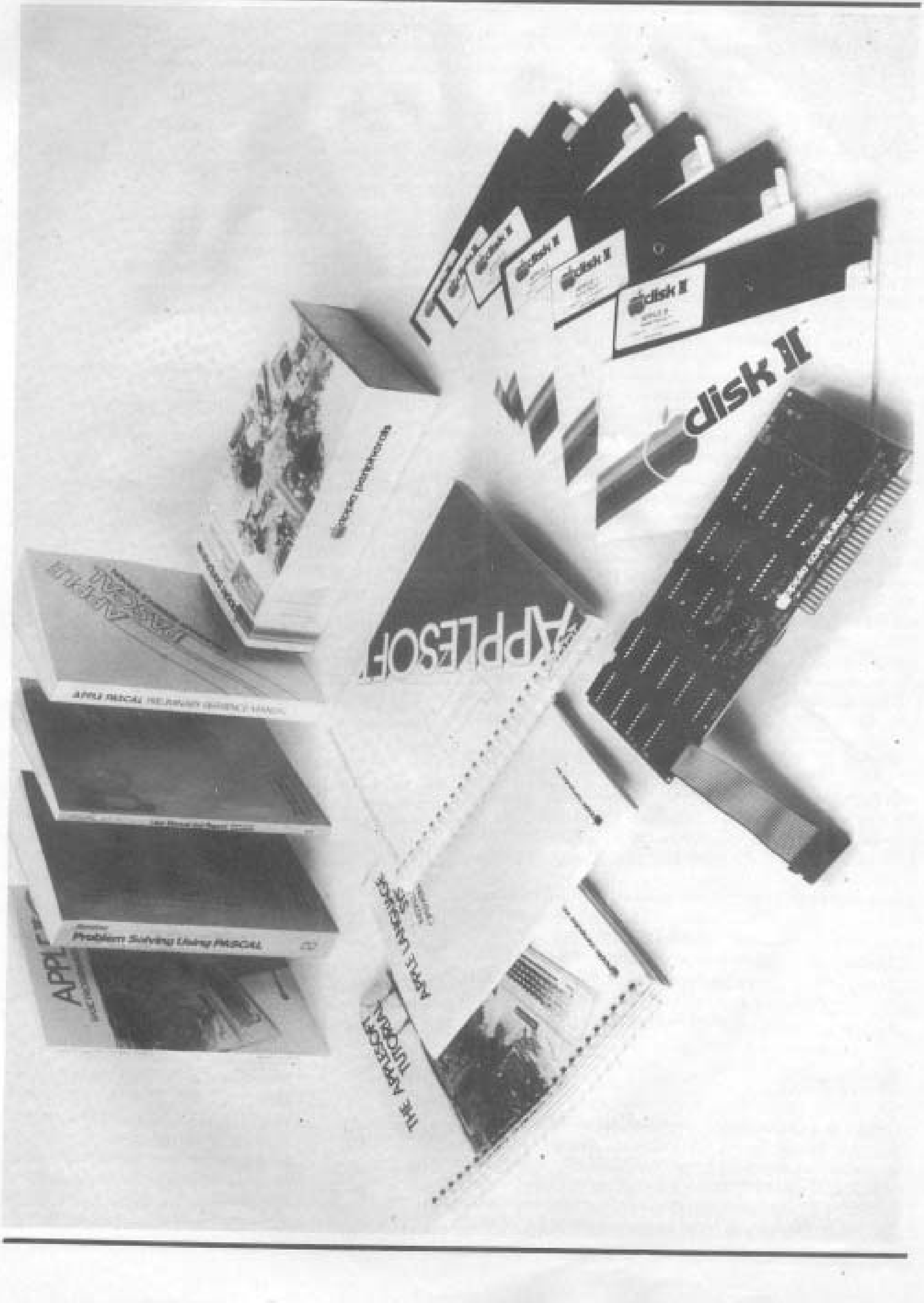
This is the general file handling utility of the system, specific peripherals for the system are addressed as 'volumes'; either volume name e.g. CONSOLE:, APPLE 0:, APPLE 1:, or volume number e.g. # 1 for CONSOLE:, # 4 for Disk (those correspond to the 'logical device numbers' of other operating systems).

In general, filenames can be referenced absolutely (i.e. the filename) or a set of files referenced by filenames with 'wildcard' characters. For example TOTAL =
TEXT will reference
TOTAL 1
TOTAL 2
TOTAL 3
TOTAL * Etc.

One particularly nice feature is the ? character. Operation is identical to the = character in specifying wildcards except that before the specified operation e.g. block deletion, the system requests verification, file by file, that the operation is to be carried out.

FILER COMMANDS

B(AD-BLOCKS:Tests all 280 sectors (each of 512 bytes for a total of 140K per drive) for


disk II
APPLESOFT
APPLE PASCAL PRELIMINARY REFERENCE MANUAL
Problem Solving Using PASCAL
THE APPLESOFT TUTORIAL
APPLE LANGUAGE

	damage. Reports those faulty.
C(HANGE	Renames a disk name or file name.
D(ATE	Sets current date. This is associated with any files saved during the current session and will be displayed on the directory listing.
E(XTENDED	DIRECTORY LIST: Displays disk name, contents of disk with file, name, size, date, starting block, datafile, for example:

APPLE Ø

SYSTEM. PASCAL	36	4–MAY–79	6	DATA
SYSTEM. MISCINFO	1	4–MAY–79	42	DATA
MICRODIGITAL TEXT	71	30–SEP–79	43	TEXT
< UNUSED >	172			

3/3 FILES, 172 UNUSED, 172 IN LARGEST

G(ET	loads specified file as system workfile. E(DIT,R(UN, or C(OMPILE will use this file.
K(RUNCH	Repacks disk so that most efficient use is made of space.
L(IST DIRECTORY	Displays simplified version of systems directory
M(AKE	Creates a disk file with specified size. Used to create a 'dummy' file on the diskette.
N(EW	Clear the workfile. Deletes SYSTEM. WRK from boot diskette.
P(REFIX	Changes default volume name to specified name.
Q(UIT	Quits filer, returns to outermost command level.
R(EMOVE	Remove specified file(s) from diskette directory—system asks for verification.
S(AVE	Saves workfile under specified name.
T(RANSFER	This is the PIP-like program (familiar to CP/M or DEC 10 uses) that is used to transfer files from disk to disk, disk to printer, etc.
V(OLUMES	Gives devices and diskettes currently on-line by volume and number
W(HAT	Name and state of workfile.
X(AMINE	Attempts to repair corrupts blocks on disk. Marks blocks that cannot be fixed.

TEXT EDITOR

This is a cursor-based window editor—similar to the Electric Pencil Tm of CP/M based systems. It makes program development or general word-processing very simple and effective with a very 'clean' and logical user interface (the requirement that a given command should behave as the 'typical user' expects is often overlooked by programmers. It is particularly important in highly used system programs—a text editor is often the users primary interface with a given computer system).

Essentially the editor commands are as follows: (the more complex each as F(IND, R(EPLACE or I(NSERT have further prompt lines as options).

On invoking the editor the current workfile is read in. If no workfile exists the system prompts for a filename or creates a new file.

COMMANDS — CURSOR MOVES *	
CTRL L	Cursor Up
CTRL 0	Cursor down
RIGHT ARROW KEY	Cursor right
LEFT ARROW KEY	Cursor left
SPACE BAR	More 1 space in set
tion	direction
CTRL I	Tab to next position
RETURN	Move to next line in set direction.
=	Move to start of latest text found, replaced, or inserted.

* These can all be prefixed by a 'repeat factor' which is an integer specifying how many times a particular operation is to be carried out e.g. 10 CTRL-L moves the cursor 10 lines down. If the repeat factor is '/' the move or command is repeated as many times as possible in the file.

DIRECTION SET

< +,	Set direction to backwards
> – ,	Set direction forwards

A(DJUST	Adjusts indentation of the line the cursor is on. Left or rigth arrow key moves the line left or right, a CTRL O or L will adjust the line above or below by the same amount
C(OPY	Copies a diskette file, or the copy buffer back into the file at the cursor position.
D(ELETE	Deletes all text moved over by the cursor. Backspacing 'undeletes'
F(IND	Operates in L(ITERAL or T(OKEN mode. Looks in the set direction for the repeat factor occurence of a specified string. Typing an S repeats the search from the new cursor position.
I(NSERT	Inserts text into file at cursor position
J(UMP	Jumps to the files B(EGINNING, E(ND or a M(ARKER (see set)
M(ARGIN	Starting at cursor position adjusts all text between two blank lines to the margins which have been S(ET. A command character (see S(ET) inhibits this.

P(AGE	Move up or down repeat factor pages.
Q(UIT	Leaves the editor. You may U(PDATE the workfile on disk W(RITE to a specified file. E(XIT without updating (text is lost) or R(ETURN to the editor.
R(EPLACE	Operation is similar to F(IND except the user specifies <target string> < replacement string> Replaces target with substitute string repeat factor times. V(ERIFY option asks for permission to replace on each occurence.
S(ET	allows the user to set parameters: M(ARKER assigns a string name to a specified cursor position. Sets options in the E(NVIRONMENT for A(UTO indent F(ILLING M(ARGINS T(OKEN C(OMAND characters
V(ERIFY	Redisplays screen with cursor: centred.
X(CHANGE	Replaces character under cursor with character typed Backspace deletes.
Z(AP	Deletes all text between the current cursor position and the start of the latest text found, replaced or inserted.

COMPILER

This is a one pass recursive descent design which compiles to an intermediate P-Code that is machine-independant and reasonably portable. The code is actually executed by a run-time interpreter which could be resident on a 6502, 8080, Z-80, 6800, LSI-11 etc.

To invoke the compiler the user types either R(UN or C(OMPILE at the outermost command level. R(UN will load the workfile and saves the updated file SYSTEM. WRK. CODE to Disk. If during compilation a syntax error is detected the system, by default, gives the user the option of continuing compilation by hitting the spacebar, exiting to the command level by pressing 'ESC' or entering the E(DITOR with the cursor pointing to the offending symbol.

When required e.g. in processing external declarations, or linkages to library routines, the linker is automatically invoked by the compiler.

Compiler time options follow the conventions of Wirth in 'Pascal User Manual and Report'.

(*$ option *). Multiple options may be specified by (*$ Option, $ Option *) etc.

COMPILER OPTIONS

C	Following characters are placed directly into codefile. Used for inserting copyright notices etc.
G +	Allows GOTO statements
G − (default)	Forbids the dreaded GOTO
I + (default)	Generates 1/0 Checking Code.
I −	No 1/0 Checking.
I filename	Includes normal sourcefile in compilation
L +	Sends compiler listing to SYSTEM.LST.TEXT
L − (default)	No compiled listing
L filename	Sends compiled listing to filename
P	Pages listing
Q +	Supress Screen messages
Q − (default)	Sends procedure names and line numbers during a compile to CONSOLE.
R + (default)	Generates range checking code for subscripts, veriables.
	No range for checking.
S +	Puts compiler in swapping mode (portions of compiler brought on and off disk) Allows more space for user symbol table compiles more slowly.
S ++	Extreme version of S
S − (default)	No swapping mode entire compiler in memory.
U + (default)	Compiles on user lex level
U −	Compiles on system lex level
U filename	Specifies name of file, if other then SYSTEM. LIBRARY, in finding external pre-defined routines—UNITS.

The linker is normally invoked automatically when R(UN is typed. It can also be invoked directly to link files other than the workfile or to procedures and Units defined externally that do not reside in the library file SYSTEM. LIBRARY.

ASSEMBLER

As a companion to the Pascal compiler there is also a 6502 macro-assembler, generating relocatable code that can be linked and executed with Pascal programs.

The Assembler is invoked by typing 'A' from the outermost command level. By default, the system aaumes that the current workfile is the source to be assembled.

The assembler is largely oriented to the needs of the Pascal system: directives are:

.PROC	< identifier >	[,expression	Proceedure]
.FUNC	< identifier >	[,expression	Function]
.END			

label definitions, space allocation directives.

label	.ASCII''	< character string >
label	.BYTE	< valuelist >
label	.BLOCK	< length ,value >
label	.WORD	< valuelist >
label	.EQV	< value >
	.ORG	
	.ABSOLUTE	
	.INTERP	

Macro directives:

.MACRO	identifier
.ENDM	

Conditional assembler directives

label	.IF	<expression>
	.ELSE	
	.ENDC	

Pascal communication directives

	.CONST	< idlist >
	.PUBLIC	< idlist >
	.PRIVATE	< identifier: integer>

list

External references

	.DEF	<identifier list >
	.REF	<identifier list >

Listing Control directive

.LIST, .NOLIST
.MACROLIST, .NOMACROLIST
.PATCHLIST, .NOPATCHLIST
.PAGE
.TITLE <title >

File directive

.INCLUDE file identifier .TEXT

Extensions

The Apple implementation includes a number of extensions to standard Pascal as defined in Pascal User Manual and Report. These include a predefined data type 'string' defined a packed array 1..80 of char. A large number of systems intrinsics dealing with strings and file handling, plus such facilities as SEGMENT which allow large programs to overlay from the disk. One of the nicest features of the system are the extensions made for the Apples' special features; the graphics, sound and analogue inputs (usually paddles or joysticks!). These are implemented as a set of predefined routines called (UNITS). To use within your program you simply declare:

USES < UNITNAMED > (UNITNAME) E.G.
PROGRAM DEMO;
USES TURTLEGRAPHICS, APPLESTUFF;
INITURTLE;

etc.

The graphics extensions are based on the 'turtle graphics' system developed by Semour Papyert at MIT. Commands follow those of a 'Turtle' dragging a pencil along the screen (similar in fact to X, Y plotter operation). Complete patterns and plots are produced with consumate ease.

The Apple screen resolution is 280 x 192 points and 12 colours are defined (although due to the vagarities of your average colour television set only about 4 or 5 will be discernible).

The 'turtle' starts off in the centre of the screen, facing right, it can turn or walk in the direction it is facing. As it walks it leaves a trail.

Procedures:

INITTURTLE; Sets graphic mode, clears screen. Turtle placed in centre of screen. Pencolour is set to none. Full screen used.

GRAFMODE; Sets graphics mode. Used to switch between text and graphics

TEXTMODE; Sets text screen

VIEWPORT (LEFT, RIGHT, TOP, BOTTOM) Use only defined position of screen for graphics.

PENCOLOUR (PENMODE); Sets colour of turtle drawings.

FILLSCREEN (PENMODE) Fills graphics screen with colour specified

MOVETO (X, Y) Draws a line with current colour from last point drawn to co-ordinates (X, Y)

TURN TO (ANGLE) Moves turtle from present angle to specified angle.

TURN (ANGLE) Turtle rotates from present angle through ANGLE in a counterclockwise direction.

MOVE (DISK) Moves turtle specified distance.

Functions:

TURTLEX: Value of current turtle X co-ordinates (Integer)

TURTLEANG: Value of current turtle angle (Integer)

SCREENBIT (X, Y): True if point X, X is not block (Boolean)

DRAWBLOCK: Allows you to put a specified array of dots in memory onto the screen to form a picture with a wide variety of options.
e.g. a sample declaration is

DRAWBLOCK (VAR SOURCE; ROWSIZE; XSKIP, YSKIP, WIDTH, HEIGHT, XSCREEN, YSCRENN, MODE: INTEGER)

Hi-Resolution Characters

One of the more inconvenient features of the Apple in its inability to mix text and graphics on the hi-resolution screen. A number of programs have been written to do this but almost all suffered from a poor user interface—disagreeing with the Disk Operating System over input, output etc. A number of 'Turtlegraphics' procedures are designed to allow the user to put character sets up on the graphics screen. The character set is stored in an array called SYSTEM. CHARSE T and may be user-defined. The present set, stored on APPLE 1: give Upper and Lower case, sigma, and a number of graphics symbols such as Chess pieces etc.

WCHAR (CH) puts character CH at current location of turtle

WSTRING (S) prints string S at current turtle location

CHARTYPE (MODE) defines mode for character write

Using Applestuff

This is a set of UNITS designed to interface with the Apple I/O and speaker.

RANDOM function returns a pseude random integer between 0 and 32767.

RANDOMIZE causes the RANDOM number generator to initialise at an upredictable point.

PADDLE (SELECT) Returns on integer in the range 0 to 255 which represents the position of the paddle. SELECT is an integer specifying which of 4 paddles (0-3) is read.

BUTTON (SELECT) Reads paddle switch (one of three). True if pressed. Will also sense cassette inputs.

TTLOUT (SELECT DATA) Set one of four TTL outputs.

NOTE (PITCH, DURATION) Self-explanatory!

In addition there are the transcellental functions:

ALL ANGLE and NUMBER arguments are real, ANGLE is in RADIANS

SIN (ANGLE)
COS (ANGLE)
EXP (ANGLE)
ATAN (NUMBER)
LN (NUMBER)
LOG (NUMBER)
SORT (NUMBER)

Pascal Slot Use

Slot	Device	Pascal Use
0	Language Card	P-Code Interpreter, I/0
1	Printer	PRINTER: or # 6
2	Modem	REMIN: REMOUT:, # 7 or # 8
3	External Console	CONSOLE:, # 1
4	Disk for example	
5	Disk for example	
6	First disks	DISK NAME : or # 4
7	PAL Card	N/A

Peripheral Cards

MOST non-Apple peripheral cards will work with the Pascal System, for example the Trendcom—100 printer and interface card works with no modifications or ill effects. In the case of peripherals such as Mountain Hardwares Apple Clock, the Speechlab Voice recognition card or any 'homebrewn' peripherals the easiest method would appear to be to write short assembly language routines which can then be addressed as UNITS. With the appropriate routines installed in SYSTEM-LIBRARY the user then simply has to say (for example):

```
PROGRAM CLOCKANDVOICE;
        USES CLOCK, VOICE;
              rest of program
```

No doubt these drivers will be available from the appropriate manufacturers before too long.

One drive systems.

Although the Pascal system will work with only one disk drive, a fair amount of copying and transfering of programs from disk to disk is necessary. For example:

The demonstration programs supplied with the Pascal systems on APPLE 3: require a fair amount of work before they will actually compile and run (this does **not** apply to multi-drive systems). I found that the easiest method was as follows:

Initialise a disk with the FORMAT program of APPLE3—call it DEMO1: or something appropriate. Transfer on to this disk.

From APPLE 0:
SYSTEM . PASCAL
SYSTEM . MISCINFO
SYSTEM . COMPILER
SYSTEM . FILER
SYSTEM . LIBARY

From APPLE1:
SYSTEM . CHARSET

From APPLE2:
SYSTEM . LINKER

From APPLE3:

SPIRODEMO . TEXT
HILBERT . TEXT
GRAFDEMO . TEXT
GRAFCHARS . TEXT etc.

You should (hopefully) now have a 'demonstration disk' which will compile and execute these programs. (When booting use APPLE3:, then insert DEMO1: in drive and press 'reset'). By loading the appropriate program using G(ET and then quitting the filer and executing R(UN, the program should should correctly compile, with the library routines automatically inserted. A codefiler (SYSTEM.WRK.CODE) is written to disk and then executed.

Overall the system appears to be very powerful and flexible. The Pascal implementation is a complete implementation, as per Wirth's original specification, with a significant number of extensions that make life easier for the personal user. The actual implementation is imbedded within a powerful operating system environment that is similar to that of much larger, and more expensive hardware.

Accompanying documentation is very much of a 'preliminary' nature (although it is far, far better than much of present microcomputer documentation). The reference manual is just that—no attempt is made at a tutorial and while 'Microcomputer problem solving using Pascal' is excellent I suspect the beginner is going to be left with a lot of questions unanswered.

Together with such products as the Winchester floppy disks now available for Apple, the Pascal system expands the number of potential applications for the machine.

N.B.

This review is based on 48 hours sleepless use of the system. It was written, typed, proofread and printed within the space of three days. Please forgive any errors of fact, or grammar that may have crept in.

ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC
ETC **ETCETERA** ETC
ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC
ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC ETC

RANDOM RUMOURS:

6809 PASCAL

Motorola's Austin home-of-the-6809 plant is reportedly nearing completion of a 6809 compiler for Pascal. When questioned late August they gave the standard 'It'll be ready in ten days or so' (read we've gotta get the bugs out yet). It will be interesting to see how it compares with the ubiquitous UCSD Pascal.

APPLE III

According to a pseudo-reliable source it sill be a bit-slice machine with plug in microporgrammed instruction sets on ROM, designed specifically for high level languages-Pascal, FORTRAN and APL. A probable introductory date is late 1980.

6809 BASIC

Following hot on the heels of their 6800 Basic, Technical Systems Consultants have developed a superb (by all accounts) 6809 version. Extremely fast, occupies about 9.5K of memory with all the facilities, plus more, of a MICROSOFT Basic.

NEW BOOK ON PASCAL

Ken Bowles, the man behind UCSD Pascal is writing a new book, to be published by BYTE, specifically for the hobbyist using Pascal on his personal computer system.

ACORN

A 4K BASIC on EPROM (ROM in 2 months) should be available by the time you read this. Very fast, with 32 bit integer arithmetic, clever design means it works with the disk and a forthcoming floating point package.

In prototype ACORN have a bus compatible 6809 board with 2K monitor, 1K RAM. Monitor supports VDU and ASC11 keyboard, file handling on tape and includes disk bootstrap.

TWO NEW TEXAS MACHINES

Following the launch of the T.I. 99/4 this summer Texas have two new machines waiting in the wings. One is the T.I. 99/3 yes-you-guessed-it a stripped down T.I. 99/4 with 8K RAM, the other the T.I. 99/7 aimed specifically at the small business market with common applications software in ROM (would you believe 500K!)

HARDWIRED LISP MACHINE

The Pascal microengine, announced last summer, which executes Pascal P-Code in hardware (the instruction set is microcoded) looks like having some competition. An as yet unknown company is to introduce a personal computer that executes LISP at the machine level, i.e. quickly. It presumably developed from work done at the A.I. labs of MIT who have had a baby LISP speaking computer for some time now—which incidentally crunches numbers as quickly as the equivalent FORTRAN systems on the big dinosaurs.

HIGH RESOLUTION GRAPHICS FOR THE SORCERER

Exidy inc., have developed a high resolution colour graphics board for the Sorcerer to plug into the S-100 bus. Apparently it is due for production 'anytime now'. I wonder if it will work with PAL as well as NTSC?

NEWBEAR 77/68

Newbear have a 6809 CPU board and companion disk system up and running on their 77/78 uvs structure. The PCB'S are 'at the manufacturer'. CPU board has 6809, 1K monitor, 1K RAM, RS-232 and cassettee interface, I/O protocols are as SWTPC.

The disk controller is a stand-alone system based on the 6800 MPU, handing two 8" hard sectored, single density drives. It should be interfaceable to a wide variety of systems.

IBM

Once again that well known manufacturer of typewriters and large computers is rumoured to be introducing a personal computer system. Amongst other 'features' it is said to have a 'three year technology lead' and 'will decimate the marketplace'.

68000

Motorola have samples of their wonderful (and complex) 68000 16 bit micro working at their Austin, Texas plant—thereby confounding the critics who said it would never work ... mind you, it remains to be seen if they can actually produce the beast in large quantities at an economic price.

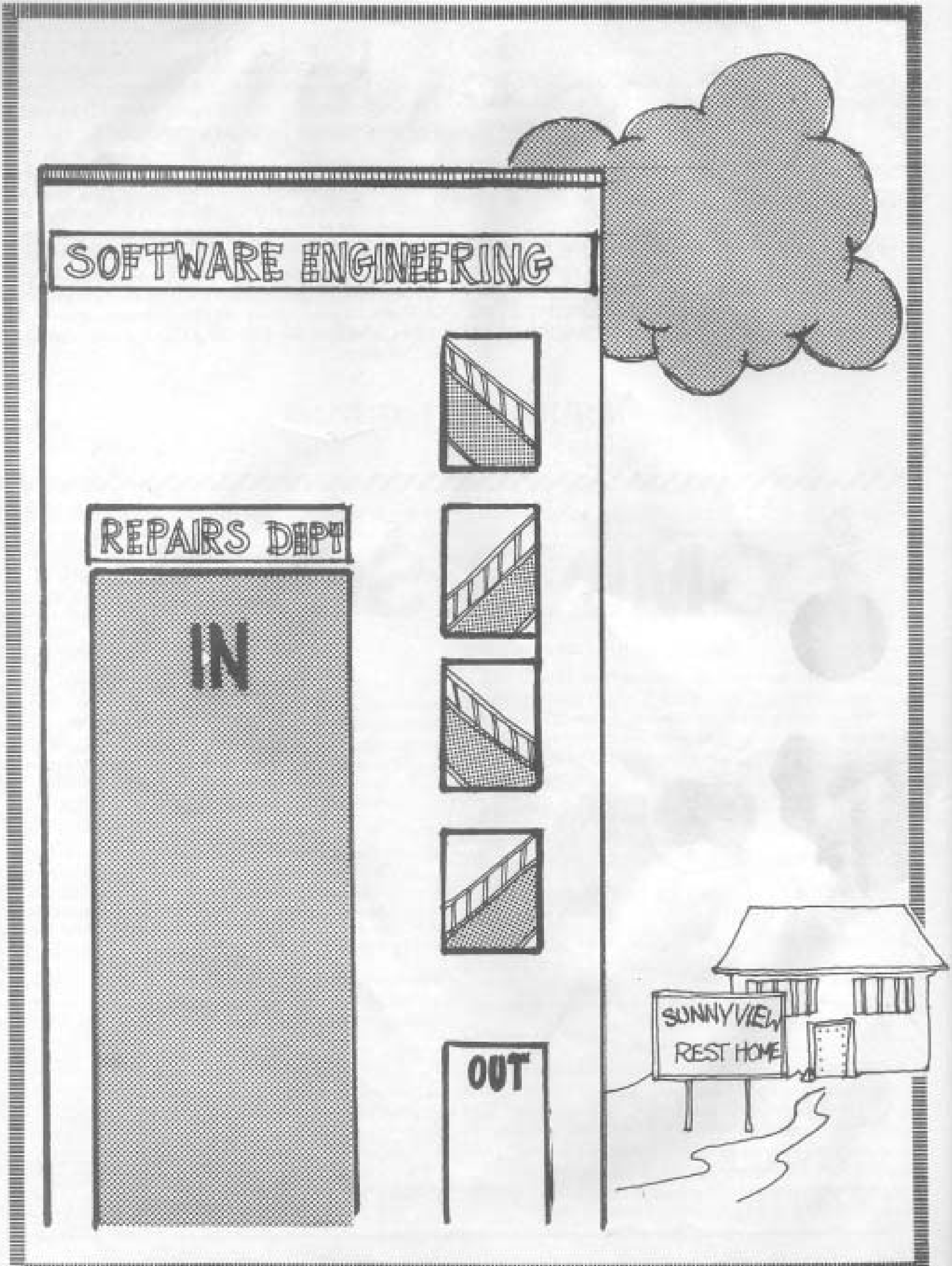
NEWBURY LABS.

Newbury laboratories, one of the few successful British V.D.U. manufacturers, are developing an 'upmarket' small computer system based on the Z-80 with an in-built printer.

NASCOM.

Nascom Microcomputers are working on a new 'packaged' computer system ... (actually a Nascom-2 with colour board in a case). Due to be released early next year they are planning to hold a competition for its name (how about one of the mythical Greek Gods).

For the Nascom-1 they have a 'Tiny Pascal' running in 4k Bytes of memory, a labelling disassembler whose output is compatible with ZEAP and a text editor—all 'in the works'.


SOFTWARE ENGINEERING
REPAIRS DEPT
IN
OUT
SUNNYVIEW
REST HOME

Two Apples Newton would have been proud of

The Pascal System
A complete system for the development and use of applications programs in Pascal, Basic or Assembly language.

48K APPLE II PLUS
Apple II Plus, with extended(Applesoft) Basic in ROM. 48K of RAM. High-resolution Black and White graphics on a matrix of 280 x 192 individually addressable points. Autostart ROM with on-screen editing, power-on books to application programs and reset key protection. 2K system monitor, fast 1500 baud cassette interface, hand controllers.

Disc System
This consists of an intelligent interface card, a powerful D.O.S. and one mini-floppy drive.

Features
* Storage capacity of 116K Bytes/Diskette (140K with language card installed)
* Powered directly from the Apple
* Fast access time — 600 m.sec (max) across 35 tracks.
* Random or sequential file access.

Pascal Language System
Includes

The Language Card — 16K Bytes of RAM memory which replaces Apples ROM firmware in the memory map.

Auto-start ROM.

5 Discs containing the Pascal compiler editor, macro assembler, linker, filer and runtime utilities, Applesoft and Integer Basic interpreters.

The language system provides the most powerful set of software development tools available to the microcomputer programmer.

Apple II Plus 48K	£988.00
Disc System with Controller	£398.00
Pascal Language System	£296.00
	£1662.00
Plus 15% V.A.T.	£249.30
Total	£1911.30

The Graphics System
A complete hi-resolution colour graphics system using the ITT 2020.

ITT 2020
48K RAM. PALSOFT Basic on ROM high resolution graphics on a matrix of 360 x 192 points. Low resolution graphics in 15 colours on a matrix of 40 x 48 points. Fast 1500 baud cassette interface to normal domestic cassette recorder.

ITT 2020 16K Colour Board	£822.00
32K RAM	£128.00
	£955.00
Plus 15% V.A.T.	£143.25
Total	£1098.25

Peripherals

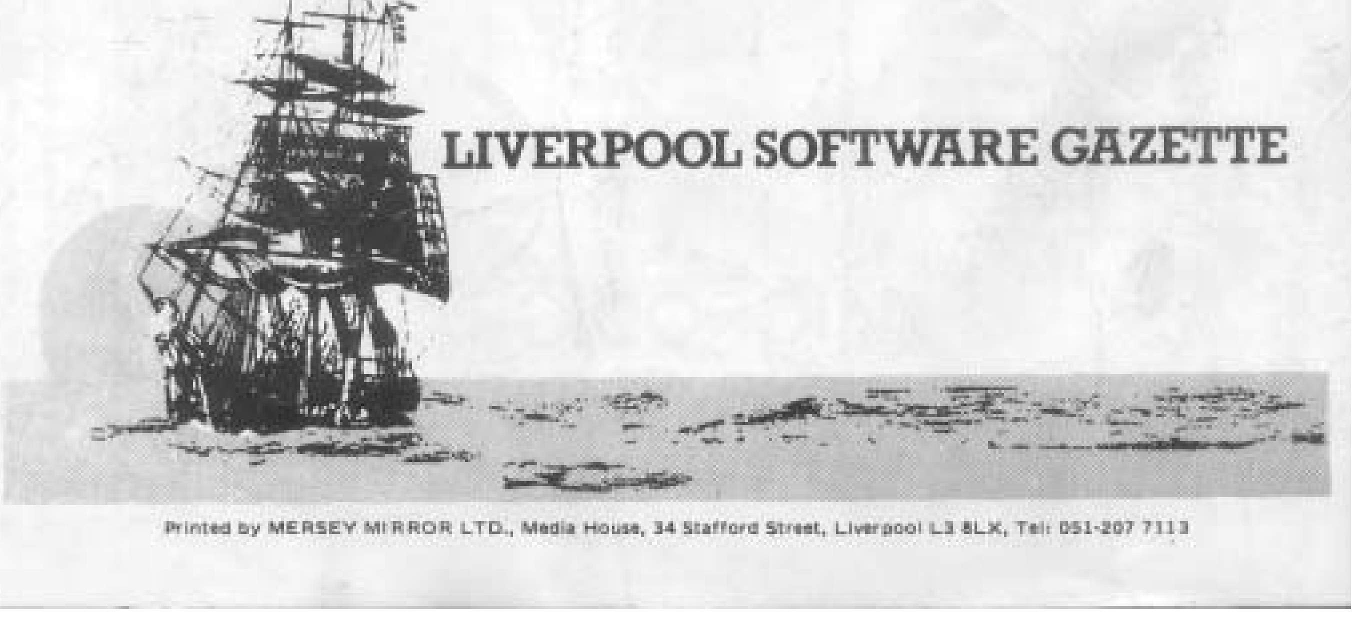
	Nett	V.A.T.	TOTAL
Parallel Printer Card	110.00	16.50	126.50
Applesoft Card	110.00	16.50	126.50
Integer Basic Card	110.00	16.50	126.50
Communications Card	132.00	19.80	151.80
Clock Card	140.00	21.00	161.00
Light Pen	165.00	24.75	189.75
Voice Recognition Card	127.00	[illegible]	[illegible]
Eurocolour Card	69.00	10.35	79.35
Carrying Case	25.00	3.75	28.75
Supertalker	190.00	28.50	218.50
Lower Case Adapter	40.00	6.00	46.00

MICRODIGITAL

25 Brunswick Street, Liverpool L2 0PJ
Tel: 051-236 0707 (24 hour Mail Order)
051-227 2535 (All other Depts.)

Mail orders to: MICRODIGITAL LIMITED,
FREEPOST (No Stamp Required)
Liverpool L2 2AB.

apple computer
Sales and Service



LIVERPOOL SOFTWARE GAZETTE

Printed by MERSEY MIRROR LTD., Media House, 34 Stafford Street, Liverpool L3 8LX, Tel: 051-207 7113