PROGRAM FILE

CP/M STAT Command Tip

by Thomas Hardy

The STAT command can be used to hide disk files from prying eyes. When you type in STAT *.* $SYS on CP/M systems, all disk files are completely hidden.

To see them again through the use of the DIR command, type STAT *.* $DIR.

BBC 3D Lettering

by Andrew Smith

This short program displays lettering in a 3D-perspective fashion. It displays the selected phrase in the bottom corner of the screen and scans over the actual characters. If the pixel being scanned is white, then a value of one will be registered by the POINT(x,y) line. From this, the character is built up after being slightly modified for perspective.

```
  10 REM Perspective type lettering
  20 REM    February 1986
  30 REM Andrew.F.G.Smith.
  40 :
  50 MODE1:SX=40:VDU 19,2,3;0;
  60 XMX=6:YMX=8:Col%=2
  70 REPEAT
  80 PRINT"Enter phrase:":INPUT K$
  90 CLS
 100 PROCTEXT(K$,100,600)
 110 PRINTTAB(0,30);"Save the screen?":INPUT Q$:IF Q$="Y" THEN OSCLI("SAVE pers
pec 3000 7800"):PRINT"OK."
 120 UNTIL 0
 130 :
 140 DEFPROCTEXT(S$,AX,BX)
 150 DX=0:EX=AX:CCX=LEN(S$)-1:GCOL 0,Col%
 160 PRINT TAB(0,30);S$;STRING$(38-LEN(S$)," ")
 170 FOR UX=1 TO LEN(S$)
 180 FOR YX=0 TO 32
 190 FOR IX=DX TO DX+32
 200 IF POINT(iX,YX+32)<>0 THEN PLOT 69,IX*XMX+AX,YX*YMX+BX
 210 NEXT IX
 220 AX=AX+CCX
 230 NEXT YX:EX=EX+SX
 240 AX=EX:DX=DX+32:CCX=CCX-2
 250 NEXT UX
 260 ENDPROC
```

Memotech MTX PANEL Fill Utility

by Terry Trotter

The Memotech computers have a 'front panel' display of registers for assembly language program debugging. There are several commands available on this front panel, but one that is missing is a memory FILL command. Memotech has provided a call in the ROM routines to a RAM location (FEXPAND) to allow expansion of the available commands. If a jump is placed at this location to the code to be executed, this will add a new command.

The FILL command checks for a valid command, in this case F, and if found, continues with the FILL command. Otherwise, it returns to the panel commands.

The program starts by transferring the rest of the program code to high memory, resetting the stack limit, setting up the jump, and returning to Basic. The utility is then safe in memory.

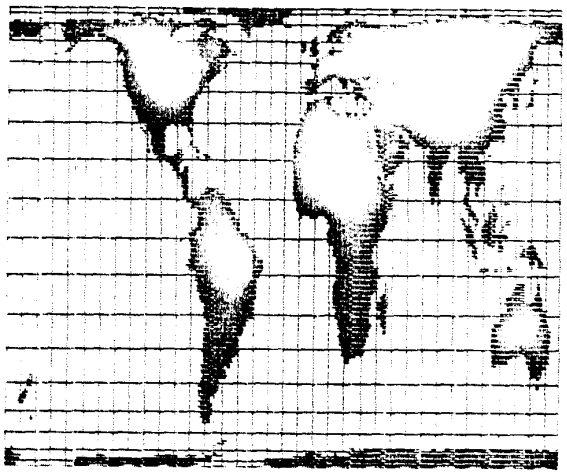
When the command has been activated by F, it will prompt for a start address, an end address, and a FILL value.

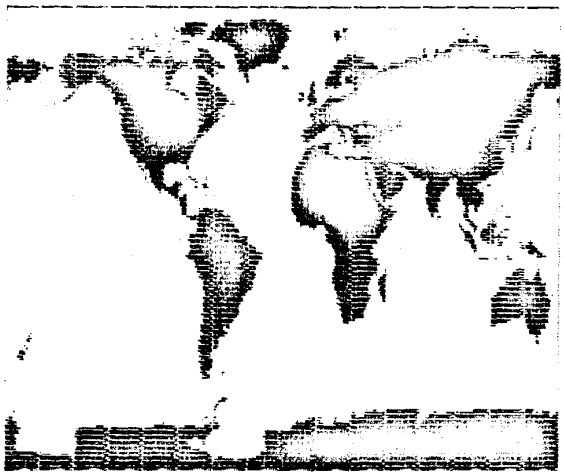
```
  10 CODE

4007 PANEXT: LD DE,£F6F0        ;point to where we will be
400A         LD (£FA92),DE      ;change stack limit
400E         LD (£FA9F),DE      ;change panel expand bytes
4012         LD HL,FILL ;point to the start
4015         LD BC,£27  ;how many bytes to move
4018         LDIR       ;move them !
401A         LD A,£C3   ;set up the jump to fill
401C         LD (£FA9E),A
```

PROGRAM FILE

```
401F            RET
4020 FILL:      LD A,(£FD7D)        ;was the last character a "F" ?
4023            CP "F"
4025            RET NZ      ;if not return
4026            RST 28      ;print Fill and get bytes
4027            DB £AB
4028            DB "Fil",£EC        ;last byte has bit 7 set to 1
402C            PUSH BC     ;save the start address on stack
402D            RST 28      ;print To and get bytes
402E            DB £AB
402F            DB "T",£EF ;last byte has bit 7 set to 1
4031            PUSH BC     ;save the end address on stack
4032            RST 28      ;print "With" and get byte
4033            DB £AB
4034            DB "Wit",£E8        ;last byte has bit 7 set to 1
4038            LD A,C      ;fetch the byte into A
4039            POP HL      ;get the end address
403A            POP DE      ;get the start address
403B            AND A       ;clear carry flag
403C            SBC HL,DE   ;calculate how many bytes
403E            LD B,H      ;set up length in BC
403F            LD C,L
4040            LD H,D      ;set up start address in HL
4041            LD L,E
4042            LD (DE),A   ;fill the first byte
4043            INC DE      ;point to the next with DE
4044            LDIR        ;fill the rest
4046            RET

Symbols:
FILL    4020    PANEXT  4007
```

Example output from Spectrum Globeplotter





Memotech MTX PANEL Fill Amendment

by Harrie Wijnands

This listing gives some changes and improvements to the utility published in 'Program File', *PCW* June. Please refer to that for instructions and use. The marked portions of the listing are the important changes.

```
10 CODE

4007 IMPFIL: LD DE,#F6F0        ;point to where we are
400A         LD (#FA92),DE      ;change stack limit
400E         LD (#FA9F),DE      ;change panel expand bytes
4012         LD HL,FILL ;point to the start
4015         LD BC,#29  ;how many bytes to move
4018         LDIR       ;move them !
401A         LD A,#C3   ;set up jump to fill
401C         LD (#FA9E),A
401F         RET
4020 FILL:   LD A,(#FD7D)       ;was the last character a "F" ?
4023         CP "F"
4025         RET NZ     ;if not return
4026         RST 28     ;print "Fill" and get bytes
4027         DB #AB
4028         DB "Fil",#EC       ;last byte has bit 7 set
402C         PUSH BC    ;save the start address on stack
402D         RST 28     ;print "To" and get bytes
402E         DB #AB
402F         DB "T",#EF ;last byte has bit 7 set
4031         PUSH BC    ;save the end address on stack
4032         RST 28     ;print "With" and get byte
4033         DB #AB
4034         DB "Wit",#E8       ;last byte has bit 7 set
4038         LD A,C     ;fetch the byte into A
4039         POP HL     ;get the end address
403A         POP DE     ;get the start address
403B         AND A      ;clear carry flag
403C         SBC HL,DE  ;calculate how many bytes
403E         RET C      ;don't fill if end address below start address
403F         LD (DE),A  ;fill the first byte
4040         RET Z      ;only one byte to fill if zero length
4041         LD B,H     ;set up length in BC
4042         LD C,L
4043         LD H,D     ;set up start address in HL
4044         LD L,E
4045         INC DE     ;point to the next byte with DE
4046         LDIR       ;fill the rest
4048         RET

Symbols:
FILL    4020    IMPFIL  4007
```

Commodore 64 OLD Routine

by Andy Lunness

This short program provides an OLD command to recover Basic programs once they have been accidentally NEWed. To use it, load it in and run it. If you accidentally delete a program, type SYS 49152 and it should reappear.

```
10 X=49152

20 READ A:IFA=-1 THEN 1000

30 POKE X,A:X=X+1:GOTO 20

40 DATA 165,43,164,44,133,34,132,35,160

50 DATA 3,200,177,34,208,251,200,152,24

60 DATA 101,34,160,0,145,43,165,35,105

70 DATA 0,200,145,43,136,162,3,230,34

80 DATA 208,2,230,35,177,34,208,244,201

90 DATA 208,243,165,34,105,02,133,45

100 DATA 165,35,105,0,133,46,96,-1

1000 PRINT"TYPE SYS 49152 TO RECOVER
     A BASIC PROGRAM."
```

Program works on EPSON, STAR & SHINWA. but with ...

T.J.'s Workshop

minimal changes to the codes sent in GRMODE and at the beginning to set the line feed spacing, it should also work with other dot matrix printers, such as the Seikosha range.

My own Star Gemini 10X printer takes 30 seconds to produce a screen dump.

The routine is fully relocatable. It can be loaded normally and then relocated to a suitable memory block by loading another program which will produce the image to be dumped to the printer.

There are two methods for relocating the routine: either the Move facility in PANEL, or a simple routine as shown below.

Symbols

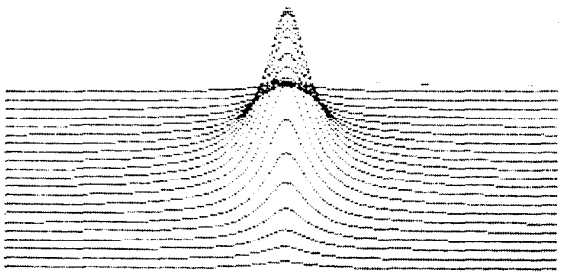
Relocation routine
4000 CODE

```
89E0    LD BC,£95      ; length of routine
89E3    LD DE,63900    ; start location of relocated routine
89E6    LD HL,£8007    ; start location of routine
89E9    LDIR
89EB    RET
```

The screen dump routine is then called by RAND USR (start address): for example, from above routine RAND USR (63900).

A Puttock

Sample screen dump



Listing of routine to produce above

```
1000 VS 4: CLS
1010 FOR X=0 TO 255
1020 FOR Y=40 TO 140 STEP 5
1030 LET Z=1/(((X-130)/125)^2+0.01)
1040 LET G=ABS(Y-40)*2.55
1050 LET R=EXP(((G-130)/125)^2+0.02)
1060 LET R=R^3
1070 LET Z=Z/R
1080 PLOT X,Y+Z
1090 NEXT : NEXT
```

Memotech MTX screen dump
by A Puttock

```
10 CODE

8007        LD A,0          ; Set up VDP to read the contents of VR 4
8009        OUT (2),A
800B        OUT (2),A
800D        DI              ; Stop any interrupts upsetting the screen dump
800E        LD B,27         ; Output ESC,A,8 to set the linefeed to 8/72 inches
8010        CALL £0CE3      ; Used to output data in B register to the printer
8013        LD B,65
8015        CALL £0CE3
8018        LD B,8
801A        CALL £0CE3
801D        LD HL,£0000     ; Zero memory locations used for character count
8020        LD (£FD9F),HL
8023        JR GRMODE       ; Set printer to column scan graphics mode
8025 LOOP:  JR GETCH        ; Read a character from the screen
8027 R1:    JR DECODE       ; Decode and print the character
8029 R2:    LD HL,(£FD9F)   ; Get character count
802C        LD A,L
802D        AND £1F         ; Is it the end of a line ?
802F        JR NZ,LOOP      ; No : Go and deal with the next character
8031        LD B,10         ; Yes : then output a line feed
8033        CALL £0CE3
8036        LD HL,(£FD9F)
8039        LD A,3
803B        CP H            ; Has the whole screen been dumped ?
803C        JR Z,END        ; Yes : then go and tidy up and return
803E GRMODE: LD B,27        ; No : put the printer back in graphics mode for
8040        CALL £0CE3      ; another line
8043        LD B,75         ; Graphics mode is obtained by sending the following
8045        CALL £0CE3      ; ESC,K,n1,n2,... giving a column scan of 256*n2+n1
8048        LD B,0          ; columns of one bit width
804A        CALL £0CE3
804D        LD B,1
804F        CALL £0CE3
8052        JR LOOP         ; Go back to draw the next line
8054 END:   LD B,27         ; Send ESC,@ to reinitialize the printer's conditions
8056        CALL £0CE3
8059        LD B,64
805B        CALL £0CE3
805E        EI              ; Let O.S. continue working
805F        RET             ; Return to BASIC
8060 GETCH: LD HL,(£FD97    ; Read the eight bytes representing a character into
8063        LD B,8          ; successive locations using INI, unfortunately INIR
8065        LD C,1          ; works too fast to produce reliable results
8067 INLP:  INI
8069        JR NZ,INLP
806B        JR R1
806D DECODE: LD C,8         ; Send a character to the printer column by column
806F D1:    LD B,0
8071        LD D,8
8073 D2:    OR 0            ; Locations used by PANEL for storing temporary
8075        DEC HL          ; register values are used to store the character
8076        RLC (HL)        ; currently being decoded and the character count
8078        JR NC,NBETB     ; this is done to allow full relocatability
807A        BET 7,B
807C NBETB: DR Q            ; The decoding is achieved by rotating each of the
807E        DEC D           ; bytes which make up a character and setting a bit
807F        JR Z,SEND       ; in the B register if the carry flag is set.
8081        RR B            ; The B register is also rotated and thus the column
8083        JR D2           ; scan of bits is put into the B register
8085 SEND:  CALL £0CE3      ; It is then sent to the printer
8088        LD HL,£FD9F     ; Get HL for the next column
808B        DEC C           ; Is the character finished
808C        JR NZ,D1        ; No : Go back and do next column of bits
808E        LD HL,(£FD9F)   ; Yes : increment the character count and return
8091        LD DE,£0001     ; using the JR R2 statement
8094        ADC HL,DE
8096        LD (£FD9F),HL
8099        JR R2
809B        RET

Symbols:
GRMODE 803E    GETCH   8060
LOOP   8025    DECODE  806D
R1     8027    R2      8029
END    8054    D1      806F
D2     8073    NBETB   807C
SEND   8085    INLP    8067
```

Acornsoft Forth

Missing from Acornsoft Forth for the BBC are two very useful words: CALL and USR.

Both of these prime the 6502 registers with values and execute a machine code JSR to a specified location. USR differs from CALL in that it returns a value computed from the A, X, Y and P registers when the routine ends.

Here is a screen to provide a USR function for Forth users.

Note that if you do not want any values returned on the stack from the machine code (that is, you want CALL instead of USR), omit lines 10 through 14 of the screen and insert the following at line 10: N JSR, XSAVE LDX, NEXT JMP.

Remember to load the assembler vocabulary before you compile this.

Richard Clarke

```
SCR  £60  3CH
 0   (A USR word for Acornsoft Forth)
 1   HEX CODE USR (a/x/y/addr . . . a/x/y)
 2   4C£ LDA, N STA,
 3   BOT LDA, N1+ STA,
 4   BOT 1+ LDA, N2+ STA,
 5   INX, INX, 0£ LDY,
 6   BEGIN, BOT LDA, N3+, Y STA,
 7   INY, INX, INX, 3£ CPY, 0= UNTIL,
 8   XSAVE STX, N3+ LDY,
 9   N4+ LDX, N5+ LDA,
10   NJSR, N1+ STA, N3+ STX, N5+ STY,
11   0£ LDA, N STA, N2+ STA, N4+ STA,
12   XSAVE LDX, 0£ LDY, BEGIN, DEX,
13   N, Y LDA, 00, X STA, INY,
14   6£ CPY, 0= UNTIL, NEXT JMP,
15   END-CODE DECIMAL
```

Dragon graphics page

A graphics page on the Dragon 32 may be saved quite simply using the CSAVEM command. G graphics, page in PM resides between 1 (&H0600) and 7 (&H1E00).

Thus to the tape the grapi

MEMOTECH MEMORY

David Miles takes an objective look at the MTX-500.

Table 1.	0	2000	4000	8000	C000	FFFF
Page 0	8K Monitor ROM on all pages	Front panel ROM (8K)	512 RAM only	500/512 RAM	500/512 RAM on all pages	
Page 1		Basic ROM (8K)		512 RAM only		

Table 2.			
06 00	0A 00	80	FF
Length of line	Line number	Token for Rem	End marker

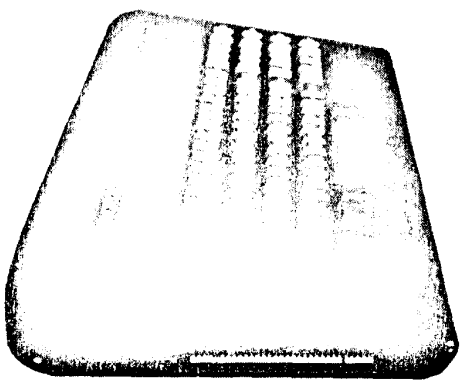
I HAVE HAD MY MTX-500 since late November and I have found it to be an exceptional machine, second only to the BBC.

The ROM is well-structured — unlike the messy layout of the Spectrum ROM — and this gives it the advantage of speed. However, there are bad features with every machine; the worst feature of the Memotech being the inadequate error messages. The cassette routines are also poor — in ROM between hex addresses 2AEE and 2B2F on page 1 — consisting of simple Load, Save and Verify routines with no provision made for Saving machine code, arrays, etc. The Circle command produces an ellipse, like the Oric-1.

This is not a programming error, but in the MTX-series the pixels are stretched laterally, so if a circle is plotted by Sin and Cos — which incidentally are faster than the BBC's mathematical functions — it will still end up as an oval.

The memory of a MTX-series machine is available on 16 64K pages, though only pages 0 and 1 are used in the unexpanded computers — see table 1.

The disadvantage of paged ROM is that it is difficult to keep jumping from one page to the next. For example, you cannot disassemble the Basic ROM on page 1 from the front panel because this is using page 0. The method to apply is to Poke the ROM routines into free RAM on page 0, so that it is possible to study that memory area from the front panel. The 8K monitor ROM contains the vital functions needed to set up the system ready for programming as well as all the graphics routines.



By disassembling the system-C 8K Basic ROM, I have discovered that the MTX-series use a token entry routine like the Sinclair and Commodore machines. This means that each Basic command, string or function has a number in a token table from 128 to 255. As the character codes of the function keys are between 128 and 144, they generate the first 16 tokens, eg., F1 is key 128 and Rem is token 128. By typing F1 and pressing Return, the word Rem is interpreted.

Print out Basic commands

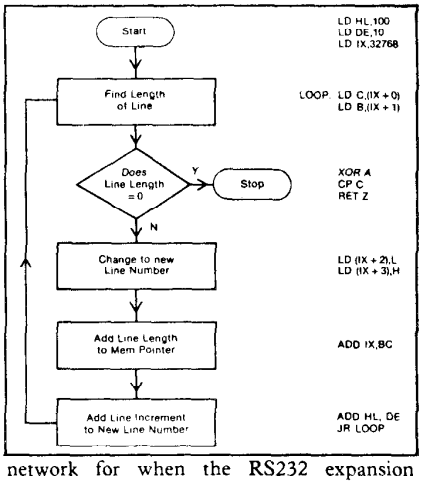
The program will print out the Basic commands, character codes and jump addresses by Peeking the token table:

```
 10 REM * TOKEN TABLE PEEKER *
 20 LET TJUMP = 9975:LET CHAR = 127
 30 FOR A = 9531 TO 9974
 40 LET Z$ = CHAR$(PEEK(A))
 50 IF ASC(Z$)>127 THEN LET
    Z$ = CHAR$(ASC(Z$) - 128) +
    CHR$(9)
 60 IF ASC(Z$)<32 THEN LET Z$ =
    CHR$(9)
 70 PRINT Z$;
 80 IF RIGHT$(Z$,1) = CHR$(9) THEN
    GOSUB 100
 90 NEXT A
100 LET M = PEEK(TJUMP):LET
    N = PEEK(TJUMP + 1)
110 LET ADDRESS = N*256 + M
120 LET CHAR = CHAR + 1:LET
    TJUMP = TJUMP + 2
130 IF CHAR,193 THEN LET
    ADDRESS = 0
140 PRINT CHAR, ADDRESS
150 RETURN
```

The first thing you will probably notice is that there are six commands in the token table that are not mentioned in the manual. They are:

USER, NODE, FK, OFF, INP, FRE

Inp is the equivalent of the Spectrum's In command. It reads a byte from a port, for instance Print Inp(6) reads the keyboard matrix. Print Fre(X) returns free memory in the RML research machines but in the MTX-series it outputs e to the power of x. e is a constant (2.7128) used as the base for natural logs and in calculus. User jumps to a user routine at address 64137 — named User in the system variables. Node is associated with the network for when the RS232 expansion becomes available.



The screen VRAM cannot be directly assessed. It is addressed by passing bytes through I/O ports 1 and 2. The necessary technical information is at the back of the manual, but try this routine:

```
10 PRINT " ";
20 FOR A = 1 TO 13
30 READ B
40 OUT (1),B
50 NEXT A
60 DATA 72,105,32,77,84,88,32,85,
   115,101,114,115,33
```

A program is stored in the MTX-500 from hex address 8000 (decimal 32768). If you type the line:

10 REM

and then study the area 8000 hex through the Front Panel, you will see the bytes in table 2.

As line numbers are held in memory as two bytes, lines may be numbered as anything between 0 and 65535. This memory layout makes program renumbering a simple affair.

The procedure for entering an assembly language program is given on page 129 of the manual.

The two Reset keys, when pressed together, are supposed to erase the memory contents so that the computer can accept another program. In fact, the program remains in memory and is just over-written — as an examination of the memory at address 8000 hex will reveal. This means that previously unstoppable programs such as Toado or Kilopede can now be broken into and listed. Load in the program, press the two Reset keys and Poke 64167,1. This Poke makes the computer "remember" that it has a program in memory. The program is now listable.

This final routine will print the amount of memory left in bytes. It should be assembled into the first program line and called by

PRINT USR(32768)

The registers used are BC — top of Basic — and HL — start of system variables.

```
LD BC, (64167)
LD HL, 64082
SBC HL, BC
PUSH HL
POP BC
RET
```

■

To round off Keith Hook's guide to Memotech programming, here are the diagrams we promised.

Memo completed

The two tables presented here are an extension to Keith Hook's introductory article on programming the Memotech, which we published last week.

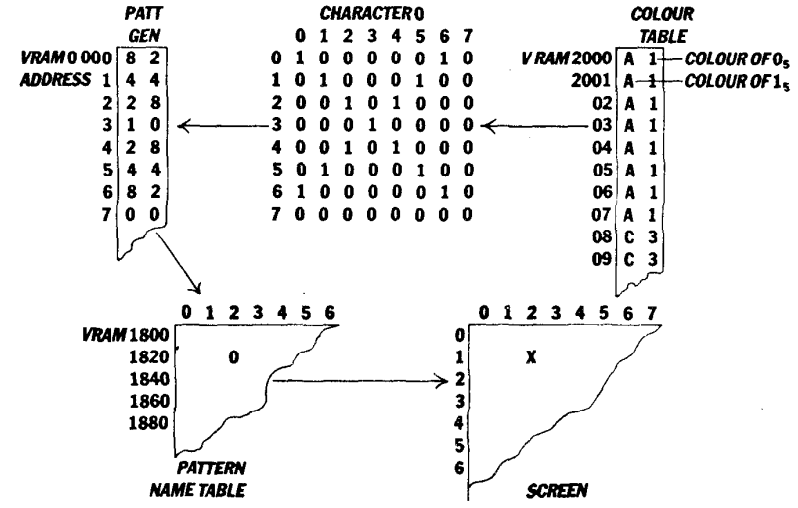
The Memotech offers three display modes which will handle text, sprites or more complex graphics screens, and therefore it's worthwhile spending a little time getting to know them.

Diagram 1 shows how mode 2 operates. As there are only 256 character codes available at any one time, the Memotech splits the pattern generator table into three sections of 2048 bytes, so the 256 character names are each used three times.

Table interaction is illustrated by diagram 2. Once you can put together the various effects possible with the Memotech's screen modes, you'll see how they can be used together in programs to obtain some very startling effects.

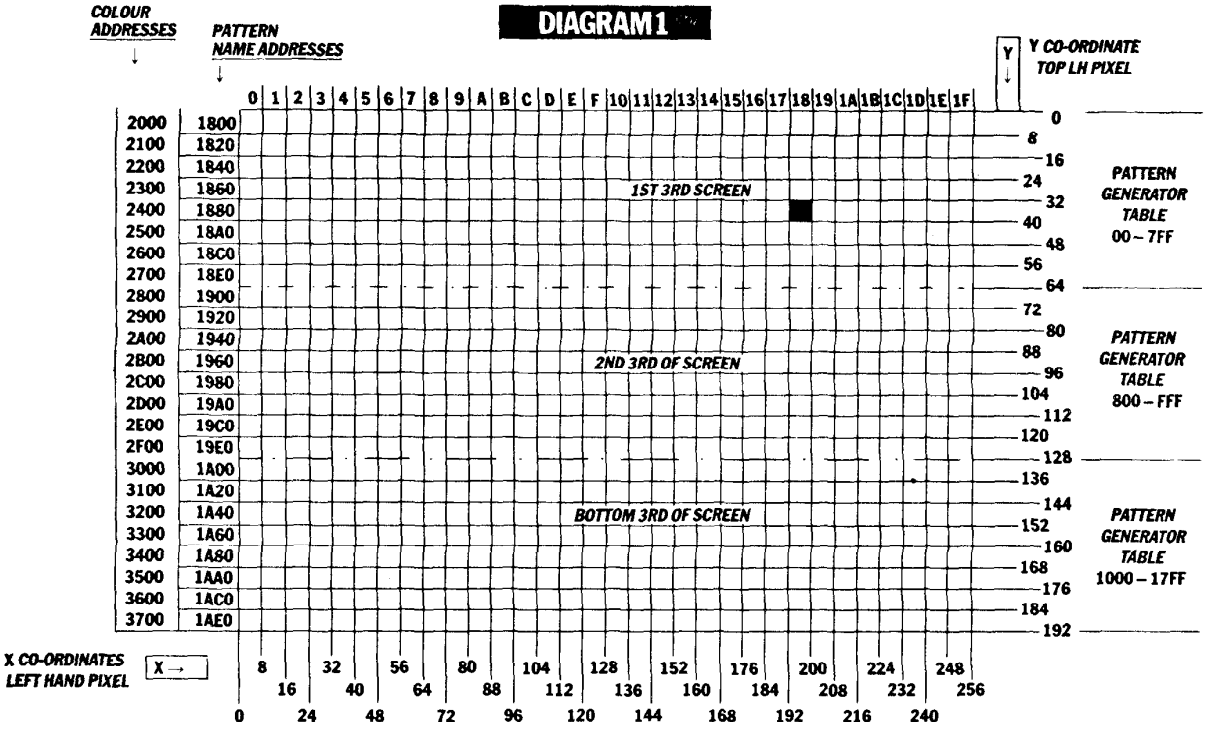
If you missed the first part of the article, back issues are available from the PCN Back Issues Service, 53-55 Frith Street, London W1A 2HG.

DIAGRAM 2 — **TABLE INTERACTION**



WILL CAUSE THE LETTER 'X' CHARACTER CODE 0 TO BE DISPLAYED AT LOCATION 35 ON THE SCREEN WITH BLACK BACKGROUND YELLOW PRINT.

DIAGRAM 1



SCREEN RELATIONSHIPS GRAPHIC MODE II

"LOCATION OF ■" BY PIXEL 192, 31 (SEE NOTES)
BY PATTERN NAME TABLE → 1898_{16}
BY COLOUR TABLE 2418_{16}

PROGRAM FILE

```
480 DATA 125,62,10,24,238,42,92,125,62,11,          791
490 DATA 24,231,0,0,0,0,0,0,0,0,                    255
500 REM
          ..........END OF DATA.............
600 REM ** Start of DEMO Adventure **
609 REM ** You MUST define GET varibles eg. V and N **
610 CLS:CLEAR 500:DEFINT V,N
619 REM ** Disable BREAK key **
620 POKE 16396,7
624 REM ** Verb table at 32384.Noun table follows **
625 VB=32384:NO=VB+25:E$=STRING$(3,255)
629 REM ** Make VERB table in VB$. (Max 240 chars) **
630 VB$="SAVEXATHR"+E$: REM SAVe,EXAmine,THRow
639 REM ** Do the same for NOUN table **
640 NO$="HILFIECOR"+E$: REM HILl,FIEld,CORn
650 A=VB:PO$=VB$:GOSUB660:A=NO:PO$=NO$:GOSUB660:GOTO680
660 FORI=1TOLEN(PO$):POKE A,ASC(MID$(PO$,I,1)):A=A+1
670 NEXT:RETURN
679 REM ** Tell A/D were the word tables are **
680 POKE 32120,VB AND255:POKE 32121,VB/256
690 POKE 32134,NO AND255:POKE 32135,NO/256

699 REM ** Get command from player **
700 PRINT"You are in a large field filled with corn.
Objects : A SCARE-CROW":PRINT"
-------> What now ? ";:GET V,N:PRINTSTRING$(63,45)
710 IF V=0 THENPRINT"
Sorry don't understand the verb":GOTO700
720 IF N=0 THENPRINT"
Sorry don't understand the noun":GOTO700
729 REM ** Jump to decoding routines **
730 ON V GOTO 800,850,900

800 REM ** Save Game **
810 '...CODE FOR SAVING GAME...
849 GOTO700

850 REM ** Examine **
860 '...CODE FOR EXAMINE...
895 PRINT"Nothing interesting"
899 GOTO700

900 REM ** THROW **
910 '...CODE FOR THROW...
949 GOTO700


and so on....
```

MTX OLD

by P Walker

I am assured by MTX500 and 512 programmers that one of the major dangers of programming an MTX is accidental erasure of a program. There are three software resets (PRINT USR (0),RST 0 and NEW), as well as hardware resets and misplaced POKEs all resulting in a lost program and re-initialised system variables. This utility provides a much needed OLD command to restore lost programs from one of the above-mentioned causes.

The program should be loaded and run before programming begins. The program places itself at the top of RAM, lowers RAMTOP and deletes its own listing, and is therefore transparent both to user and system. You can now type or load in a program. When you are ready to start debugging type (PRINT(USR(49010). If, for any reason, the program is cleared from memory, typing (PRINT(USR(49069)) while debugging will restore it.

To run on an MTX500 two minor changes are necessary:

(1) Change LD HL, 4069 to LD HL, 8069 in line 0.

(2) Change the two occurrences of LD IX, 4000 to LD IX, 8000 in line 2.

```
"OLD" FACILITY FOR THE MTX 512/500

0 CODE
      LD HL,#4069    ;START ADDRESS OF "OLD"
      LD DE,#BF72    ;ADDRESS "OLD" MOVED TO
      LD BC,#0076    ;LENGTH OF "OLD"
      LDIR           ;MOVE "OLD" TO #BF72
      LD A,#F6       ;OFF SET FOR (DJNZ) JUMPS USED IN "OLD"
      LD (#BFA8),A   ;POSITION OF (DJNZ)
      LD (#BFE4),A   ;POSITION OF 2nd (DJNZ) IN "OLD"
      RET
```

PROGRAM FILE

```
1 NEW
2 CODE
      LD IX,#4000   ;START ADDRESS OF PROGRAM TO BE RELOADED---1st SECTION OF "OLD"---
      LD HL,#BFE9   ;START ADDRESS FOR PROGRAM DATA TO BE SAVED
      LD B,#08      ;COUNTER FOR PROGRAM DATA
      CALL #BF9F    ;CALL SAVE ROUTINE
      LD IX,#FAA4   ;START ADDRESS OF VARIABLES TO BE SAVED
      LD B,#0A      ;COUNTER FOR VARIABLE DATA
      CALL #BF9F
      LD A,(#FACC)  ;TOP OF ARRAYS (LOW BYTE)
      CALL #BFAA    ;CALL SAVE ROUTINE
      LD A,(#FACD)  ;TOP OF ARRAYS (HIGH BYTE)
      CALL #BFAA
      LD A,(#FAD6)  ;TOP OF PAGE (LOW BYTE)
      CALL #BFAA
      LD A,(#FAD7)  ;TOP OF PAGE (HIGH BYTE)
      JP #BFAA      ;JUMP TO SAVE ROUTINE AND RETURN TO BASIC
      LD A,(IX+0)   ;SAVE ROUTINE AT #BF9F---LOAD A,DATA BYTE
      CALL #BFAA    ;CALL SAVE ROUTINE
      INC IX        ;MOVE POINTER TO NEXT DATA BYTE
      DJNZ #BF9F    ;CHECK DATA COUNTER/ IF NON ZERO REPEAT ROUTINE
      RET           ;RETURN FROM ROUTINE
      LD (HL),A     ;SAVE ROUTINE AT #BFAA---SAVE DATA BYTE TO HL LOCATION
      INC HL        ;SET POINTER TO NEXT SAVE ADDRESS
      RET           ;RETURN FROM ROUTINE
      LD IX,#4000   ;ADDRESS TO RELOAD PROGRAM---2nd SECTION OF "OLD"---
      LD HL,#BFE9   ;START ADDRESS OF PROGRAM DATA
      LD B,#08      ;COUNTER FOR PROGRAM DATA
      CALL #BFDB    ;CALL RELOAD ROUTINE
      LD IX,#FAA4   ;ADDRESS OF VARIABLES TO BE RELOADED
      LD B,#0A      ;COUNTER FOR VARIABLE DATA
      CALL #BFDB
      CALL #BFE6    ;CALL LOAD ROUTINE
      LD (#FACC),A  ;RELOAD TOP OF ARRAYS (LOW BYTE)
      CALL #BFE6
      LD (#FACD),A  ;TOP OF ARRAYS (HIGH BYTE)
      CALL #BFE6
      LD (#FAD6),A  ;TOP OF PAGE (LOW BYTE)
      CALL #BFE6
      LD (#FAD7),A  ;TOP OF PAGE (HIGH BYTE)
      RET           ;RETURN TO BASIC
      CALL #BFE6    ;RELOAD ROUTINE AT #BFDB
      LD (IX+0),A   ;RELOAD DATA BYTE INTO PROGRAM/VARIABLE SECTION OF MEMORY
      INC IX        ;MOVE POINTER TO NEXT MEMORY POSITION
      DJNZ #BFDB    ;CHECK DATA COUNTER/IF NON ZERO REPEAT ROUTINE
      RET           ;RETURN FROM ROUTINE
      LD A,(HL)     ;RELOAD ROUTINE AT #BFE6---RETRIEVE DATA BYTE
      INC HL        ;MOVE POINTER TO NEXT DATA LOCATION
      RET           ;RETURN FROM ROUTINE

  3 REM  C. P WALKER  1984
```

Commodore 64 Screendump

by Matthew Burt

Screendump allows the printing of high resolution or user-defined character screens on the Commodore MPS-801 printer. If a normal text screen is dumped, the standard '64 character set will be used instead of the MPS-801 set. There are two stages:

(1) Load and run to locate the routine in memory anywhere above B00 hex. There is a default option to put the routine above the Basic text and protect it.

(2) Call the specified address either from Basic or a machine code monitor.

The MPS-801 must be device four and contain paper that can accommodate 54 characters across.

Screendump leaves the printer in graphics mode, so ASCII character 15 must be sent before normal use.

Any screen that uses sprite graphics will be printed but the sprites will be invisible. The results with screens that use raster interrupts are, at best, highly unpredictable.

Lines 1000-1800 contain the data of the machine code routine in decimal. Lines 2040-2100 actually transfer the DATA into memory using a dummy OPEN in line 2060. Pay particular attention when typing lines 2000-2100. Any error here will not be detected in the same way as the DATA lines.

To prove the program works there are some example printouts of graphics dumps done with the utility. On running, the prompt

CODE ADDRESS (HEX) :

will be displayed. Type an address (such as C000 or C800 or 9000) or press RETURN for the top of memory option. If all is well, the message

CALL WITH SYS xxxx

will be displayed. Use this SYS to dump the screen to the MPS-801. Monitor users may wish to locate the code under the Basic ROM.

```
1000 DATA11,8,10,0,158,50,53,50,340
1010 DATA51,0,0,0,169,4,170,160,554
1020 DATA0,32,186,255,169,0,32,189,863
1030 DATA255,32,192,255,162,4,32,201,1133
```

```
840 RETURN
899 REM Dump to screen subroutine
900 AD=VAL("£"+A$):EA=VAL("£"+E$)
920 IF AD>EA THEN 1060
930 H$=HEX$(AD):L=LEN(H$)
940 PRINT "   ";RIGHT$("0000"+RIGHT$(H$,L-1),4);"
    ";
945 J=EA-AD+1:IF J>8 THEN J=8
950 FOR I=1 TO J
960 CN=PEEK(AD):H$=HEX$(CN):L=LEN(H$)
970 PRINT " ";RIGHT$("00"+RIGHT$(H$,L-1),2);
980 AD=AD+1
990 NEXT I
1000 PRINT
1010 K$=KEY$:IF K$=CHR$(24) THEN 1060
1030 IF K$=" " THEN GET K$:IF K$=CHR$(24) THEN 10
60
1050 GOTO 920
1060 RETURN
1099 REM Printer machine code listing routine
1100 AD=VAL("£"+A$):EA=VAL("£"+E$)
1120 IF AD>EA THEN 1260
1130 H$=HEX$(AD):L=LEN(H$)
1140 LPRINT RIGHT$("0000"+RIGHT$(H$,L-1),4);" ";
1145 J=EA-AD+1:IF J>8 THEN J=8
1150 FOR I=1 TO J
1160 CN=PEEK(AD):H$=HEX$(CN)
     L=LEN(H$)
1 0 LPRINT " ";RIGHT$("00"+RIGHT$(H$,L-1),2);
1180 AD=AD+1
1190 NEXT I
1200 LPRINT
1210 K$=KEY$:IF K$=CHR$(24) THEN 1260
1230 IF K$=" " THEN GET K$:IF K$=CHR$(24) THEN 12
60
1250 GOTO 1120
1260 RETURN
1500 END
```

Add Commands — Memotech

Most Memotech fans will be aware that the machine comes with a free text adventure euphemistically called 'The Operator's Manual'. For those that have ... to decipher the various ... of the manual, it should be apparent that whilst exceptional in certain areas (Assembler, Monitor, Logo graphics, Sprites and Noddy) the MTX series falls short in some others, after all 24K ROM can only hold so much.

Over the next few issues BYTE PACK will feature a set of short machine code utilities, from MTX expert Eric Roy of Kilmarnock, that will patch up certain discrepancies.

Before moving onto the utilities themselves, readers should note that as code cannot be assembled at a given address, at the end of each listing you will be required to use the PANEL's M(ove) command. This relocates your code in the last two pages of the BASIC program area (#BE00 to #CE00).

Listing 1 — 'OLD' enables the user to restore a BASIC program that has been NEW'd or lost after the two reset keys have been pressed. The program is quite simple containing two sets of routines. The first set (SAVEB, SAVESV) copies the first 8 bytes of a BASIC program along with the system variables connected with BASIC, Noddy, and arrays to the safe locations #BE88 to #BE9C. The second set (RESTB,RESTSV) restores these bytes to their original location. To test the program you should enter LET O=USR(48640) to call the first two routines and then reset or NEW. Finally, to restore the program call the second set by entering LET O=USR(48707).

Before testing, however, code should be save and then moved. Owners of 512 and 500 models should move their code from #4007 ending #408C and from #8007 ending #808C respectively to #BE00.

A final note for MTX500 users, change the LD HL,(#4000), LD HL,(#4002) etc in the SAVEB and RESTB parts of the listing to LD HL,(#8000), LD HL,(#8002) etc.

Listing 2 — 'MERGE' allows you to append a subroutine (SUN) onto the body of a program (MAIN). Whilst this method is somewhat circuitous, the process is invaluable for those who like to build up a library of generalised routines and so avoid duplicating their efforts on some later game or application.

Having typed in the program and with 'OLD' resident in memory, load the subroutine to be appended, enter CLEAR as a direct command and then enter LET O=USR(48800). The last instruction will call the routine SUB which uses 'OLD' to copy across the system variables and then moves a copy of your subroutine up in memory starting at #C400. This area is, in fact, reserved for the storage of variables, but as it is unlikely that any will be created whilst merging it can be safely overwritten. In any case, to be on the safe side #C400 instead of #C000 is used, which still allows programs of over 10K to be merged.

Now load the 'MAIN' program. Enter LET=USR(48820) to call the three routines MAIN,NONOD and MERGE. The first checks to see whether there are any Noddy pages to be merged and if so relocates them at the end of 'SUB'. The second moves both 'SUB' and any Noddy pages down to the end of 'MAIN'. The third adds the system values stored by OLD to those set when loading 'MAIN' to produce the new variables for the combined programs.

Again, before testing, ensure that the utility is saved and that it is moved from #4007,#4083 (MTX512) or #8007,#8083 (MTX500) to #BEA0.

Listing 3 — 'INTERUPTS' (his spelling, not ours) not only makes 'OLD' available at the touch of a key but also corrects the manual (which fails to point out that bit 7 along with either 4,5 or 6 in the INTFFF variable must also be set before interrupts are vectored through the USRINT location).

The routine causes the computer to interrupt what it's doing 64 times every second to check to see whether a key has been pressed. This is achieved by taking the value of the last-key-pressed system variable (#FD7C) and comparing it with the key code of the keys to be tested. In this case the computer checks to see whether the Space Bar has been pressed and if so performs SAVEB and SAVESV routines in 'OLD'. Next it checks F1 and restores any NEW'd program. Unfortunately, F1 will not work on a reset as this turns the interrupts off (default).

Key F2 will also switch the interrupts off and must be used prior to loading a program as the USRINT location is also saved and may not contain the address of the KEYS interrupt routine.

Keys F3 and F4 will be covered in the next issue which will include routines to renumber program lines and to save and load code directly as opposed to via a BASIC program. In the meantime #404B to #4051 can be omitted if desired.

One last point: before testing, the code should be moved from #4007 ending #4053 (#8007 ending #8053) to #BF60.

```
5 REM **** BYTE PACK ****
6 REM **** OCT 1984  ****
100 CODE
803B SAVEB:  LD HL,(£4000)      ; SAVEB saves the
first 8 bytes of basic.
803E         LD (£BE88),HL      ; Store for 8 byte
s £BE88 to £BE8F.
8041         LD HL,(£4002)
8044         LD (£BE8A),HL
8047         LD HL,(£4004)
804A         LD (£BE8C),HL
804D         LD HL,(£4006)
8050         LD (£BE8E),HL
8053 SAVESV: LD HL,(£FAA4)      ; Save Top of Nodd
y.
8056         LD (£BE90),HL      ; Store for system
 variables £BE90 to £BE9D.
8059         LD HL,(£FAA7)      ; Save Top of Curr
ent Basic Page.
805C         LD (£BE92),HL
805F         LD HL,(£FAAA)      ; Save Bottom of B
asic.
8062         LD (£BE94),HL
8065         LD HL,(£FAAC)      ; Save Top of Each
 Basic Page.
8068         LD (£BE96),HL
806B         LD HL,(£FACC)      ; Save Top of Arra
ys.
806E         LD (£BE98),HL
8071         LD HL,(£FACF)      ; Save Baselin.
8074         LD (£BE9A),HL
8077         LD HL,(£FAD6)      ; Save Patop.
807A         LD (£BE9C),HL      ; All system varia
```

```
ytes now saved.
807D          RET          ; End of save routine.
807E RESTB:   LD HL,(£BE88)      ; Restore 8 basic
bytes.
8081          LD (£4000),HL
8084          LD HL,(£BE8A)
8087          LD (£4002),HL
808A          LD HL,(£BE8C)
808D          LD (£4004),HL
8090          LD HL,(£BE8E)
8093          LD (£4006),HL
8096 RESTSV:  LD HL,(£BE90)      ; Restore system v
ariables saved.
8099          LD (£FAA4),HL
809C          LD HL,(£BE92)
809F          LD (£FAA7),HL
80A2          LD HL,(£BE94)
80A5          LD (£FAAA),HL
80A8          LD HL,(£BE96)
80AB          LD (£FAAC),HL
80AE          LD HL,(£BE98)
80B1          LD (£FACC),HL
80B4          LD HL,(£BE9A)
80B7          LD (£FACE),HL
80BA          LD HL,(£BE9C)
80BD          LD (£FAD6),HL      ; All system varia
bles restored.
80C0          RET

Symbols:
SAVEB   803B    SAVESV  8053
RESTB   807E    RESTSV  8096


5 REM **** BYTE PACK ****
6 REM **** OCT 1984  ****
100 CODE

803B INTON:   LD A,£C3    ;  Code for JP (Jump).
803D          LD (£FA98),A      ;  USERINT locatio
n.
8040          LD HL,£BF93       ;  Interupts vecto
red to this address = KEYS.
8043          LD (£FA99),HL     ;  USERINT +1,2 =
Interupt vector address.
8046          LD A,(£FD5E)      ;  INTFFF system v
ariable.
8049          OR £9F      ;  Set bits 4 & 7.
804B          LD (£FD5E),A      ;  INTERUPTS ON.
804E          RET
804F INTOFF:  LD A,(£FD5E)      ; INTFFF.
8052          AND £0F     ;  Bits 4 & 7 reset.
8054          LD (£FD5E),A      ;  INTERUPTS OFF.
8057          RET
8058 REN1:    LD HL,£64   ;  First line number=100.
805B          LD (£BF1E),HL     ;  Line number loc
ation.
805E          JR STEP
8060 REN9:    LD HL,£2328       ;  First line numb
er=9000.
8063          LD (£BF1E),HL
8066 STEP:    LD A,£0A    ;  Step between lines=10.
8068          LD (£BF1D),A      ;  Step location.
806B          JP £BF20    ;  Jump to renumber routin
e.
806E KEYS:    LD A,(£FD7C)      ;  Last key presse
d system variable.
8071          CP £49      ;  Is it SPACE BAR.
8073          JP Z,£BE00  ;  Yes save system variabl
es in OLD.
8076          CP £48      ;  Is it key 'F1'.
8078          JP Z,£BE43  ;  Yes restore NEW'ed prog
ram system variables.
807B          CP £46      ;  Is it key 'F2'.
807D          JR Z,INTOFF       ;  Yes switch inte
rupts off.
807F          CP £43      ;  Is it key 'F3'.
8081          JR Z,REN1   ;  Yes renumber program fr
om 100.
8083          CP £41      ;  Is it key 'F4'.
8085          JR Z,REN9   ;  Yes renumber program fr
om 9000.
8087          RET

Symbols:
INTON   803B    INTOFF  804F
REN1    8058    STEP    8066
REN9    8060    KEYS    806E

5 REM **** BYTE PACK ****
6 REM **** OCT 1984  ****
100 CODE

803B SUB:     CALL £BE00 ;  Store system variables
of 'SUB'.
803E          LD HL,(£FAAA)       ;  HL=Start addres
s of 'SUB'.
8041          LD DE,£C400         ;  DE=Destination
address of 'SUB'.
8044          LD BC,(£FACC)       ;  BC=Length of 'S
UB'.
8048          LDIR        ;  Move 'SUB' from HL to D
E.
804A          LD (£BE86),DE       ;  Store end addre
ss of 'SUB' after move.
804E          RET         ;  NOW LOAD 'MAIN' PROGRAM
.
804F MAIN:    LD HL,(£FAA4)
8052          LD DE,(£FAA7)
8056          AND A
8057          SBC HL,DE   ;  HL=Length of any Noddy
pages in 'MAIN'.
8059          JR Z,NONOD  ;  If no Noddy pages then
jump, else
805B          PUSH HL
805C          POP BC      ;  BC=Length of 'MAIN' Nod
dy.
805D          LD HL,(£FAA7)       ;  HL=Start addres
s of 'MAIN' Noddy.
8060          LD DE,(£BE86)       ;  DE=End of 'SUB
 program.
8064          LDIR        ;  Move 'MAIN' Noddy to en
d of 'SUB'.
8066          LD (£BE86),DE       ;  Store new end a
ddress of 'SUB'.
806A NONOD:   LD HL,(£BE86)       ;  HL=End address
of 'SUB'.
806D          LD BC,£C400         ;  BC=Start addres
s of 'SUB'.
8070          PUSH BC     ;  Save start address.
8071          AND A
8072          SBC HL,BC   ;  HL=Length of 'SUB'.
8074          PUSH HL
8075          POP BC      ;  BC=Length of 'SUB'.
8076          POP HL      ;  HL=Start address of 'SU
B'.
8077          LD DE,(£FAA7)       ;  DE=End of 'MAIN
' basic.
807B          LDIR        ;  Move 'SUB' to end of 'M
AIN'.
807D MERGE:   LD HL,(£BE90)       ;  Calculate 'NEW'
 system variables.
8080          LD DE,(£FACC)
8084          ADD HL,DE
8085          PUSH HL
8086          LD (£FAAC),HL       ;  'NEW' Top of ea
ch basic page.
8089          LD HL,(£FAA7)
808C          LD DE,(£FAAA)
8090          AND A
8091          SBC HL,DE
8093          PUSH HL
8094          LD BC,(£BE92)
8098          ADD HL,BC
8099          LD (£FAA7),HL       ;  'NEW' Top of cu
rrent basic page.
809C          POP HL
809D          LD BC,(£BE9C)
80A1          ADD HL,BC
80A2          LD (£FAD6),HL       ;  'NEW' PGTOP.
80A5          LD HL,(£FAA4)
80A8          AND A
80A9          SBC HL,DE
80AB          LD BC,(£BE98)
80AF          ADD HL,BC
80B0          LD (£FACC),HL       ;  'NEW' Top of ar
rays.
80B3          POP HL
80B4          LD (£FAA4),HL       ;  'NEW' Top of No
ddy.
80B7          RET
Symbols:
NONOD   806A    MERGE   807D
SUB     803B    MAIN    804F
```

EOT

```
570 IF A$="=" THEN MUSIC2,C,12,0:PLAY3,0,7,V
571 IF A$="G" THEN C=1
572 IF A$="H" THEN C=2
573 IF A$="J" THEN C=3
574 IF A$="K" THEN C=4
575 IF A$="L" THEN C=5
576 IF A$=";" THEN C=6
577 IF V<=10 THEN GOTO 582
578 IF A$="," THEN V=V-50
580 IF V>=17000 THEN GOTO 578
582 IF A$="." THEN V=V+50
584 IF A$="M" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
15
590 IF A$="Z" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
120
597 IF A$="S" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
306
598 IF A$="B" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
254
615 IF A$="C" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
650
620 GOTO 450
650 PLOT 15,20," ":PLOT15,20,"CHORD"
660 GET A$
670 IF A$="Q" THEN MUSIC2,C,1,0:PLAY3,0,1,V
680 IF A$="W" THEN MUSIC2,C,2,0:PLAY3,0,1,V
690 IF A$="E" THEN MUSIC2,C,3,0:PLAY3,0,1,V
700 IF A$="R" THEN MUSIC2,C,4,0:PLAY3,0,1,V
710 IF A$="T" THEN MUSIC2,C,5,0:PLAY3,0,1,V
720 IF A$="Y" THEN MUSIC2,C,6,0:PLAY3,0,1,V
730 IF A$="U" THEN MUSIC2,C,7,0:PLAY3,0,1,V
740 IF A$="I" THEN MUSIC2,C,8,0:PLAY3,0,1,V
750 IF A$="O" THEN MUSIC2,C,9,0:PLAY3,0,1,V
760 IF A$="P" THEN MUSIC2,C,10,0:PLAY3,0,1,V
770 IF A$="[" THEN MUSIC2,C,11,0:PLAY3,0,1,V
780 IF A$="]" THEN MUSIC2,C,12,0:PLAY3,0,1,V
781 IF A$="G" THEN C=1
782 IF A$="H" THEN C=2
783 IF A$="J" THEN C=3
784 IF A$="K" THEN C=4
785 IF A$="L" THEN C=5
786 IF V<= 10 THEN GOTO 789
787 IF A$="," THEN V=V-50
788 IF V>=17000 THEN GOTO 787
 89 IF A$="." THEN V=V+50
790 IF A$="Z" THEN GOTO 120
800 IF A$="B" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
254
805 IF A$=";" THEN C=6
810 IF A$="S" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
306
820 IF A$="D" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
445
825 IF A$="M" THEN MUSIC2,7,12,0:PLAY0,0,0,0:GOTO
15
830 GOTO 660
```

Next Month

Renumber — *Memotech*

Another practical machine code utility to enhance Memotech BASIC. Renumber is designed to work both on its own as well as slot into an interrupt-driven repertoire (see last issue).

Quite simply, the program enables you to renumber line numbers on a BASIC program. (It does not renumber line references on GOTOs, GOSUBs or RESTOREs.) This is extremely useful when it comes to merging one program or subroutine into the body of another (see last issue).

To test the program enter the following commands substituting step size for ST and the line number from which you wish to begin renumbering for FL:

POKE (48925), ST
POKE (48926), FL — INT (FL/256 ★256
POKE (48927), INT (FL/256)
LET R = USR (48928)

In the above, USR calls routine CHECK which examines the BASIC program area searching for a program. If such exists then ST and FL are loaded for use in the main loop.

The RENUMR routine finds the starting line in the program and then examines the first four bytes. Taken in pairs these bytes give the line length and the line's number. Having done this, control is passed to an assembler REPEAT...UNTIL loop. Current line length is found and the new line number poked into the succeeding two bytes. Step size is then added onto the last line number to give the next value to be poked in. Length is then added to the current line address and a check is made to see whether the end of the program has been reached. If it has not, then the cycle repeats.

```
10 REM **** BYTE PACK ****
20 REM **** NOV 1984  ****
100 CODE

803B CHECK:  LD HL,(#FACC)       ; Check for progra
m in memory.
803E         LD A,H
803F         OR L
8040         RET Z       ; Return if no program to
renumber.
8041         LD BC,(#BF1D)
8045         LD B,0      ;  BC=Step between lines 0
 to 255.
8047         LD HL,(#BF1E)       ; HL=First line nu
mber.
804A RENUMR: LD IX,(#FAAA)       ; IX=Start of basi
c.
804E REPEAT: LD E,(IX+0)
8051         LD D,(IX+1)         ; DE=Line length.
8054         LD (IX+2),L
8057         LD (IX+3),H         ; Poke new line nu
mber into place.
805A         ADD HL,BC   ; Add step size to HL.
805B         PUSH HL
805C         ADD IX,DE   ; IX=Address of next line.
805E         PUSH IX
8060         POP DE      ; DE=Address of next line.
8061         LD HL,(#FAAC)       ; HL=Top of basic.
8064         AND A
8065         SBC HL,DE   ; Subtract line address fr
om top of basic.
8067         JR C,END    ; END if address of line >
 top of basic.
8069         LD A,H
806A         OR L
806B         JR Z,END    ; END if address of line =
 top of basic.
806D         POP HL
806E         JR REPEAT   ; Not finished, renumber n
ext line.
8070 END:    POP HL
8071         RET

Symbols:
CHECK   803B     RENUMR  804A
REPEAT  804E     END     8070


110 REM ******************************
120 REM **** RENUMBER BASIC LINES ****
130 REM ****  MTX 500,512 MICROs  ****
140 REM ****  (c) E.Roy  June.84  ****
150 REM ******************************
```

```
30000 REM  PROCerror
30010 :
30020 REM   [PROCEDURES]P.Error
30030 :
30040 DEFPROCerror :REM error handling routine
30050
30060 REM Flush all buffers, Close Files
30070 REM and disable printer. Then print
30080 REM error line.
30090
30100 DIM char 10
30110 *FX4,0
30120 CLOSE#0 :REM close all files
30130 VDU3    :REM disable printer
30140 VDU7: PRINT "ERROR!  - Any key to continue" :A=GET
30150 VDU22,7
30160 REPORT: PRINT:CHR$131;"at line ";ERL
30170 IF ERR=17 THEN END  :REM allow escapes
30180 PRINT
30190 error$="L."+STR$(ERL)+CHR$10+CHR$10+CHR$13
30200 *FX15,1
30210 FOR I%=1 TO LENerror$
30220   A=ASC(MID$(error$,I%,1))
30230   $char="FX138,0,"+STR$(A)
30240   X%=char MOD 256
30250   Y%=char DIV 256
30260   CALL &FFF7
30270   NEXT
30280 END
30290 ENDPROC
```

Fig 2 P Error program

```
30000 REM  PROCload & PROCgetfile
30010 :
30020 REM   [PROCEDURES]P.Load
30030 :
30040 DEFPROCload
30050 REM Verify and *LOAD a file
30060 REM file is loaded at &3000 and reaches
30070 REM to "memtop"
30080 :
30090 DIM file 20
30100 PRINTTAB(0,20)SPC(30)
30110 REPEAT
30120   PROCgetfile :REM check filename
30130   file%=OPENIN(filename$)
30140   IF file%=0 THEN PRINTTAB(0,20);CHR$131;"File not feurd":VDU7:A=INKEY(200
)
30150   UNTIL file%<>0 :memtop=&3000+ EXT#(file%) :CLOSE#0
30160 REM now shove it to OSCLI
30170 $file="LOAD "+filename$+" 3000"+CHR$13
30180 X%=file MOD 256
30190 Y%=file DIV 256
30200 CALL &FFF7
30210 ENDPROC
30220 :
30230 DEFPROCgetfile :REM check filename
30240 REPEAT
30250   REPEAT
30260     PRINTTAB(0,20)SPC(30)
30270     VDU7:PRINTTAB(0,20);CHR$131;"Filename ";
30280     INPUT filename$
30290     UNTIL LEN filename$ <=9 :REM check length
30300   IF LEN filename$>7 AND MID$(filename$,2,1)<>"." THEN valid%=FALSE ELSE v
alid%=TRUE
30310   UNTIL valid%
30320 ENDPROC
```

Fig 3 P Load program

```
30000 REM  PROCkeyscan
30010 :
30020 REM   [PROCEDURES]P.Keyscan
30030 :
30040 DEFPROCkeyscan(format%,x%,y%,nchars%)
30050 REM scan keyboard for input line "com$"
30060 REM format% selects characters read -
30070 REM format%=0 gives "any key" read
30080 REM format%=1 gives "no lowercase" read
30090 REM format%=2 gives numerics only
30100 REM x% & y% are the input print positions
30110 REM nchars% maximum length of input string
30120 :
30130 REM initialise with "com$=STRING$(255," ") :wipe%=TRUE" before use
30140 :
30150 LOCAL A,I%
30160 return%=FALSE
30170 IF wipe% THEN PRINTTAB(x%,y%);SPC(LEN(com$)+3) :com$="" :wipe%=FALSE ELSE
PRINTTAB(x%,y%);com$;
30180 A=INKEY(0) :REM read any key
30190 IF A=-1 THEN ENDPROC
30200 IF format%=1 AND ( A>96 AND A<123 ) THEN A=A-32  :REM no lowercase
30210 IF format%=2 AND A<>13 AND A>127 AND ( A<48 OR A>57 ) THEN ENDPROC
30220 A$=CHR$(A)
30230 IF A$=CHR$127 AND LEN(com$)>0 THEN com$=LEFT$(com$,LEN(com$)-1)  :REM dele
te
30240 IF A$<>CHR$127 AND A$<>CHR$13 AND LEN(com$)<nchars% THEN SOUND1,-15,150,2
30250 IF A$<>CHR$127 AND A$<>CHR$13 AND LEN(com$)<nchars% THEN com$=com$+A$
30260 IF A$=CHR$13 THEN return%=TRUE :wipe%=TRUE
30270 PRINT;com$;" ";:VDU8
30280 ENDPROC
```

Fig 4 P Keyscan program

MEMOTECH PANEL EXPANSION UTILITY

Anyone who has spent time roaming around their memory via the PANEL command will find this assembler program of much use, as it allows the disassembled code to be printed out by using the instant hard copy.

For details of how to enter this program into your machine, see the assembler section of the Memotech manual.

Program notes

The program is label-driven and is totally independent of its position in memory, hence the lack of memory locations to the left of the assembler. Consequently, the program will run on any MTX regardless of its memory capacity.

This program was written by a member of the Memotech Owners' Club. Anyone wishing to enquire further about the activities of the club should send an SAE to: MOC, 23 Denmead Rd, Harefield, Southampton SO2 5GS. The annual subscription is £7. Any other Memotech submissions are always welcome.

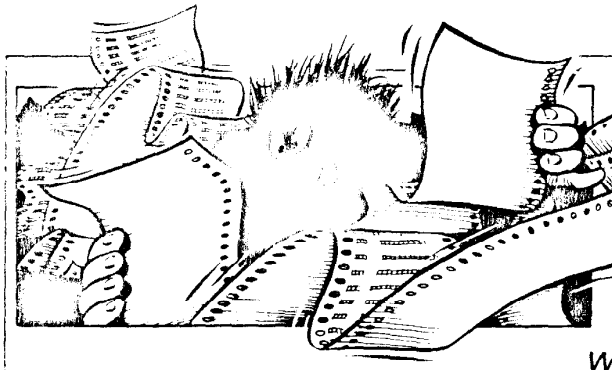
Label explanations

STRT-FINI : set system variable FEXPAND
PANEL : check for P key press
START : main program loop

Subroutines

LADDR : set VPD to VRAM read mode
DUMP : set screen width to 29 characters
LOOP2 : read screen and send to printer
BUFFER STATUS : output to printer
LPRINT : cause line feed and carriage return
WAIT:LOOP1 : idle loop

```
;
; PANEL LPRINT DUMP
;
;
STRT:    LD A,£C3        : jump code for panel
                           extension
         LD (£FA9E),A
         LH HL,PANEL     ; address for panel extension
FINI:    LD (£FA9F),HL   ; set fexpand
PANEL:   CP £50          ; is key 'P'
         RET NZ          ; if so then continue
         LD A,14         ; screen lines
         PUSH AF
         LD HL,£1C00     ; start of name table in VRAM
         LD DE,40        ; skip to next screen line
START:   CALL LADDR      ; set up address for VRAM
                           read
         CALL DUMP       ; print out one line
         POP AF
         DEC A           ; keep line countdown
         CP 0            ; has it finished
         RET Z           ; finish routine after 14 lines
         PUSH AF
         ADD HL,DE       ; address of next screen line
         JR START        ; do until finished
LADDR:   LD A,L          ; set up VRAM address
         OUT (2),A       ; LS byte first
         LD A,H          ; now set MS byte
         OUT (2),A
         CALL WAIT       ; timing pause for VDP
         RET
DUMP:    LD B,29         ; set screen width for dump
LOOP 2:  IN A,(1)        ; read screen character
         CALL WAIT       ; pause
         CALL BUFFER     ; load printer buffer
         DJNZ LOOP2      ; get next character
         CALL LPRINT     ; LPRINT one line of the
                           screen
         RET
BUFFER:  OUT (4),A       ; latch char into printer buffer
STATUS:  IN A,(4)        ; is PTR ready
         AND £1          ; ready bit check
         JRNZ,STATUS
         CALL WAIT       ; pause for printer
         IN A,(0)        ; strobe data into printer
                           buffer
         CALL WAIT       ; pause
         IN A, (4)       ; reset strobe signal
         RET
LPRINT   LD A,10         ; line feed
         CALL BUFFER     ; send to printer
         LD A,13         ;carriage return
         CALL BUFFER     ; send
         RET
WAIT:    PUSH BC         ; pause for a period
         LD B,50         ; of time to allow
LOOP 1:  DJNZ LOOP1      ; the printer and the
         POP BC          ; VRAM to respond
         RET
```



T.J.'s Workshop

Our monthly pot-pourri of hardware and software tips for the popular micros. If you have a favourite tip to pass on, send it to TJ's Workshop, PCW, 62 Oxford Street, London W1. Please keep your contributions concise. We will pay £5-£30 for any tips we publish. PCW can accept no responsibility for damage caused by using these tips, and readers should be advised that any hardware modifications may render the maker's guarantee invalid.

USE OF JOYSTICKS WITH MEMOTECH MTX

The manual for the Memotech MTX series micro does not make clear the method by which the joystick ports may be accessed within a user's program. Connecting joysticks to the Memotech quickly shows that the joysticks map into the keyboard as shown below.

This means that any game requiring joysticks can be played from the keyboard instead (albeit more clumsily). Also, it means that to use joysticks within your own programs, you need only read the keyboard (for example, with INKEY$ in Basic) to determine the joystick status.

The problem with using INKEYS (or the CHARGET routine in machine code) is that multiple key closures cannot be sensed in this way, so one is confined to the four primary directions plus fire. It is frequently desirable in a game to permit diagonal movement on the screen or to allow firing while moving, making it necessary to sense a number of key closures simultaneously (right and up, for example). To do this on the Memotech, one first needs to understand how the keyboard may be read directly.

The Memotech keyboard is arranged on two of the Z80's ports, 5 and 6. To sense the status of the keyboard, a byte has first to be output on port 5 to activate the appropriate sense lines of the keyboard. These lines are active low, so are activated by the presence of a zero in the appropriate bit of this 'sense byte'. The status of the keyboard read lines may then be determined by performing an input on port 5 (or 6) to yield a 'read byte'. Wherever a read line is active (because a key has been pressed), a zero will appear in the corresponding bit of the read byte. The problem is to determine the appropriate sense/read byte combinations for the keys of interest. (Normally, of course, this is all handled for us by the CHARGET routine in ROM).

The Basic routine in Fig 1 will cycle through the sense bytes to set each sense line in turn and display the resulting read byte. By running this routine while holding down keys, one can determine the combination needed to examine specific keys. The routine only inputs from port 5 as the majority of keys appear here (note that the space bar is one exception).

It's a simple matter to change the routine to investigate port 6, too. Be aware, however, that only the bottom two bits of the read byte from port 6 are keyboard read lines.

Once the sense/read byte combinations have been determined, they can be incorporated into a user-written keyboard read routine. Machine code is best for this as it's much quicker than Basic, and avoids the timing problems which close examination of the output from the Basic routine will reveal.

Two machine code routines for reading the joysticks (or equivalent keys of the keyboard) are given here: one to look at the right-hand joystick, the other the left. Each is used from Basic in exactly the same way; the differences between the two routines merely reflect the different sense/read byte combinations required. Ironically, the left-hand joystick is the more convenient to code for. Each routine will scan the appropriate joystick and set bits of an internal byte (called KEYS) to reflect the joystick status. These bits are set as follows:

KEYS:
- BIT 4 set if FIRE pressed;
- BIT 3 set if DOWN pressed;
- BIT 2 set if UP pressed;
- BIT 1 set if RIGHT pressed;
- (LSB) BIT 0 set if LEFT pressed.

The final value of this byte will, therefore, be determined by the combination of joystick controls active. The value may be retrieved in Basic using a PEEK instruction.

The complete program (Fig 2) shows the routines as they may be used from Basic (note that the variables KEYR and KEYL point to the KEYS bytes within the routines). The exact values of these variables will depend upon the memory size of your Memotech (adjust the variable MTX as indicated in the program) and also upon the degree of comment included in the machine code routines. Adjust the values to equal those indicated by the appropriate assembler symbol table (lines 20 and 30).

When the program is RUN, a balloon will appear which can be moved around the screen with either joystick (although the right-hand one has priority) and will change colour whenever the fire key is pressed. This program shows how easy (and convenient) it is to blend machine code and Basic on the Memotech to impressive effect.

Steve Benner

```
Right-hand joystick   : FIRE  — HOME key;
 LEFT, RIGHT, UP, DOWN  — corresponding cursor keys.
Left-hand joystick    : FIRE  — SPACE BAR;
                        LEFT  — Z key;
                       RIGHT  — C key;
                          UP  — B key;
                        DOWN  — M key.
```

```
290 REM ************
292 REM ** Routine to strobe keyboard
  ;
295 LET PORT=5
300 FOR S=0 to 7: LET SS=255-2^S: OUT (5),SS
305 LET R=INP(PORT): PRINT "Sense ";SS,"Read ";R
310 NEXT
315 PAUSE 1000: PRINT : PRINT : GOTO 300
```

Fig 1 Sense: read byte routine


```
1 GOTO 100
20 CODE
4010 GETRTJ:   XOR A      ;Clear A
4011           LD HL,KEYS
4014           LD (HL),A   ;Clear KEYS
4015 FIRE:     LD A,£DF   ;Strobe for HOME
4017           CALL STROBE
401A           JR. NZ,LEFT
401C           SET 4,(HL)
401E LEFT:     LD A,£F7   ;Strobe for left
4020           CALL STROBE
4023           JR NZ,RIGHT
4025           SET 0,(HL)
4027 RIGHT:    LD A,£EF   ;Strobe for right
4029           CALL STROBE
402C           JR NZ,UP
402E           SET 1,(HL)
4030 UP:       LD A,£FB   ;Strobe for up
4032           CALL STROBE
4035           JR NZ,DOWN
4037           SET 2,(HL)
4039 DOWN:     LD A,£BF   ;Strobe for down
403B           CALL STROBE
403E           JR NZ,DONE
4040           SET 3,(HL)
4042 DONE:     RET
4043 KEYS      DB 0
4044 STROBE:   OUT  (5),A  ;Do joystick strobe
4046           IN A,(5)
4048           CP 127
404A           RET
```

Symbols

GETRTJ	4010	KEYS	4043
STROBE	4044	LEFT	401E
RIGHT	4027	UP	4030
DOWN	4039	DONE	4042
FIRE	4015		

```
21 RETURN
30 CODE
41A6GETLTJ:    XOR A ;Clear A
41A7           LD HL,KEYS
41AA           LD (HL),A ;Clear KEYS
41AB FIRE:     LD A,127 ;Strobe SPACE-BAR
41AD           OUT  (5),A
41AF           IN A,(6)
41B1           BIT 0,A
41B3           JR NZ,STROBE
41B5           SET 4,(HL)
41B7 STROBE:   LD A,127 ;Strobe left joystick
41B9           OUT (5),A
41BB           IN A,(5)
41BD           LD D,A
41BE           AND £F0 ;Check bottom row keys
41C0           CP £F0
41C2           JR, NZ,DONE   ;Ignore if not
41C4           LD A,D ;Restore A
41C5           CPL ;Set all bits in one go!
41C6           ADD A,(HL) ;Add in FIRE bit
41C7           LD (HL),A
41C8 DONE:     RET
41C9 KEYS:     DB 0
41CA           RET
```

Symbols

FIRE	41AB	STROBE	41B7
DONE	41C8	GETLTJ	41A6
KEYS	41C9		

```
31 RETURN
97 REM ******************
98 REM **
99 REM **   MAIN CODE STARTS HERE — SET UP
   SCREEN FIRST
   ;
100 GENPAT 3,0,24,60,60,24,00,24,24,00
110 VS 4: CLS : COLOUR 0,1: COLOUR 4,1
120 CTLSPR 2,1: CTLSPR 6,1
125 LET X=10: LET Y=8: SPRITE 1,0,X,Y,0,0,10
126 REM
127 REM ********************
128 REM ** Set up SPEED; & PEEK locations (MTX=8
    for 500); See M.C for values
   ;
130 LET SPEED=4: LET MTX=4
150 LET KEYL=MTX*4096+256*1+12*16+09: LET
    KEYR=MTX*4096+4*16+3
190 REM
191 REM
192 REM ********************
193 REM **
194 REM ** Poll keyboard and recalculate coordinates
   ;
200 GOSUB 20:   LET JOYS=PEEK (KEYR): IF
  JOYS=0 THEN GOSUB 30: LET JOYS=PEEK (KEYL)
210 IF JOYS=0 THEN GOTO 200
215 IF JOYS>15 THEN LET JOYS=JOYS-16: ADJSPR
  1,1,RND*14+1
220 IF JOYS>7 THEN LET JOYS=JOYS-8: LET
  Y=Y+SPEED* (Y>10)
225 IF JOYS>3 THEN LET JOYS=JOYS-4: LET
  Y=Y-SPEED* (Y<180)
230 IF JOYS>1 THEN LET JOYS=JOYS-2: LET
  X=X-SPEED* (X<250)
235 IF JOYS>0 THEN LET JOYS=JOYS-1: LET
X=X+SPEED* (X>10)
240 ADJSPR 2,1,X: ADJSPR 3,1,Y: GOTO 200
250 REM
251 REM ************************************
    ****************************
```

Fig 2 Complete program

SORD TIPS

If you ever get fed up waiting *for long programs to load,* then perhaps you haven't found the secret of changing the rate at which programs are saved.

Type POKE &7019,&12 before you save a program, and the cassette baud rate will be almost doubled. (This works on BASIC-I and BASIC-G). If your cassette recorder cannot cope with the given value of &12, try others until you find the fastest you can safely use. The higher the value POKEd, the slower the baud rate.

Note: You do not need to change the POKEd value to load in files recorded at different speeds—the computer works out what speed it was saved at.

The manual for BASIC-G gives the impression that you *must save Basic programs by* using LIST "name". This isn't necessary—SAVE will do the job just as well, and much faster.

The advantages of using LIST, however, are that only certain lines need to be saved, if required and, more importantly, programs can be merged. For instance, you could save a frequently used subroutine with LIST, and then OLD it when ever you need it. The merged program lines will replace anything with the same number in memory, so it is best to have your subroutine renumbered to, say, 10000 onwards.

Another advantage of files saved with LIST is that they