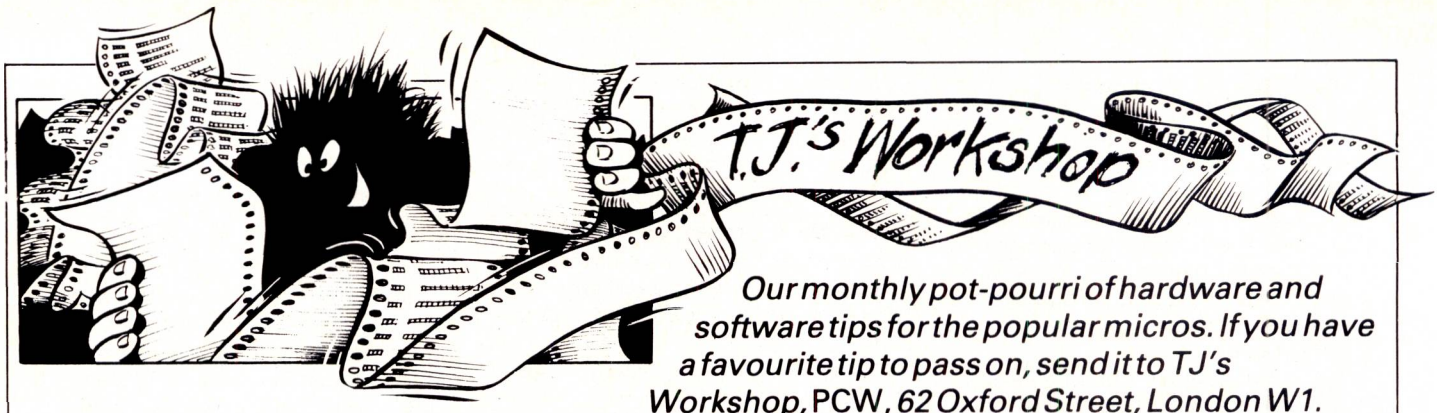

Our monthly pot-pourri of hardware and software tips for the popular micros. If you have a favourite tip to pass on, send it to TJ's Workshop, PCW, 62 Oxford Street, London W1. Please keep your contributions concise. We will pay £5-£30 for any tips we publish. PCW can accept no responsibility for damage 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 INKEY$ (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 whenever 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