SHARP MZ-700 SERIES

SOLO SOFTWARE

BEGINNING MACHINE CODE on the SHARP MZ-700, MZ-80A and MZ-80K

Solo Software Ltd.
95B Blackpole Trading Estate
Worcester WR3 8TJ
Tel: 0905 58351

MACHINE CODE OR ASSEMBLER?

Assembler is a simple way of entering a machine code program to avoid the necessity of learning around 500 numeric codes. In assembler, the codes are represented by simple words or initial letters that indicate the function that they perform.

Although not the same as BASIC commands, they will soon become familiar.

DECIMAL OR HEX?

Decimal is the normal method of counting as used with our UK currency. It has a base of ten which means that every time you count above nine the number in the next column on the left is increased by one and you start the first column from zero again.

In hex (hexadecimal) the base is sixteen and the numbers from 10 to 15 are represented by the letters A to F.

Therefore to count in hex the numbers read

0 1 2 3 4 5 6 7 8 9 A B C D E F

Once you reach F the next number is 10 but this is not TEN but SIXTEEN. To allow you to tell when hex is being used, the $ symbol goes before the hex number. The hex number $25 is equivalent to the decimal number 37. That is the first digit multiplied by 16 (32) plus the second digit (5). The largest hex figure you are likely to see is $FFFF which stands for the decimal 65535. This is how it works out.

Remember that $F=15

Position	Calculation	Total
F———	15 x 16 x 16 x 16	= 61440
—F——	15 x 16 x 16	= 3840
——F—	15 x 16	= 240
———F	15 x 1	= 15
TOTAL		= 65535

Try to work out what these hex numbers are. The answers are printed on the last page of this booklet.

(1) $2A (2) $C4 (3) $AF

(4) $F1 (5) $AC2 (6) $624

(7) $1200 (8) $1FFF

In S-BASIC, if you type PRINT $FF the computer will print the decimal equivalent (in this case 255). The SOLO BASIC or S-BASIC will also convert from decimal to hex. PRINT HEX$(255) . will give e result FF on the screen.

FROM BASIC INTO MACHINE CODE

We must assume that you have a reasonable working knowledge of BASIC and understand the POKE command which you probably use mainly to place characters onto the screen. A POKE places a numeric value into a specific memory location. This is the simplest form of

machine code programming. A PEEK tells you what value occupies a memory location. USR (followed by a number in brackets) calls a machine code subroutine. You will probably have used USR(62) to make a BEEP sound on the computer in BASIC. After the subroutine is completed, the sequence RETURNS to the section of the program following the command (just like GOSUB in BASIC).

LIMIT (followed by a number in brackets) stops a BASIC program encroaching on the area where machine code subroutines are stored.

Here is a simple example of a machine code program called from BASIC which will give a double BEEP whenever USR(49000) is called.

```
10 DATA 205,62,0,205,62,0,201
20 LIMIT (48000)
30 FOR X=49000 TO 49006
40 READ A
50 POKE X,A
60 NEXT
70 USR (49000)
```

In assembler this routine reads
CALL NN 62 CALL NN 62 RET

Note that the final command is RET which returns you to BASIC.

Assembler programs can be written in this way or entirely in machine code.

THE TUTORIAL CASSETTE

On side A you will find a copy of our own SOLO BASIC followed by the MACHINE CODE TUTORIAL written in SOLO BASIC. On side B of the cassette is the same TUTORIAL written in standard Sharp S-BASIC.

We recommend the SOLO BASIC version as it works on the MZ-80A and 80K as well as on the MZ-700. Also, the banking system in S-BASIC makes screen routines more awkward. It is usually simpler to recover from a 'crash' in SOLO BASIC. You will also find that SOLO BASIC is far more economical in its use of memory, leaving you more space to play with.

Having loaded the BASIC and then the Tutorial, type RUN and wait while the necessary data is loaded into memory. If the program is ever stopped with the BREAK key you will find that RUN will no longer work (to avoid re-loading data). Type GOTO3000 instead which takes you to the menu.

Item (1) on the menu gives each machine code number and the appropriate command in assembler which may be easier to remember.

Item (2) is a hard-copy print-out of the above codes.

Item (3) gives each assembler code on request (one at a time).

Item (4) allows you to read a machine code routine in assembler which will let you know (eventually) what is happening where. The main use, at first, is to allow you to read the monitor and BASIC program resident in your computer. You won't understand much at this stage but it will come to mean more in the future.

The code in the monitor and BASIC is mainly sequences of instructions but does include messages, address tables, flags, etc. The assembler prints the CHR$ of each location to help you to recognise when this occurs.

Item (5) is the dissembler that converts your assembler into true machine code for you. You must choose a safe area to place your program and we recommend 40000 to 52000 in SOLO BASIC. In S-BASIC only 50000-53000 is available. All BASIC languages have safe areas inside them and you should find these first using the assembler.

Remember to enter carefully using the correct spacing and the number must be entered as N, NN or E. Then you are asked for the number in decimal.

Here are some aids that the computer will accept.

RESTART	prompts 'Restart where?'
ERROR	cancels the last entry.
MC	allows entry in hex.
AS	returns you to assembler.
SAVE	saves your routine from the starting point to current location.
CHECK	reads back your code from the start address entered.
END	will return you to the menu. Now test it using USR.

Item (6) will find a number stored in two locations. For example, you can find all the addresses where the BASIC start is stored, and subsequently raise them by poking to give extra space.

Item (7) reads an address table. In SOLO BASIC, 6687 will give you the addresses where the routines for each reserved word starts. In S-BASIC the number is 12615.

Item (8) gives the list of reserved words. The relevant starts are

SP-5025 BASIC	5336
SOLO BASIC	16826
S-BASIC	10997

Item (9) is only available in SOLO BASIC. It deletes the data section of the program to allow the use of 35000 to 40000 for your code.

REGISTERS

Registers are similar to memory locations. They contain 8 'bits' of information which are either on or off. This is a binary system of switching where the value of 0 means OFF and the value 1 means ON. It follows that, as you have eight 'switches' in each location, some may be ON while others can be OFF giving a variety of information. The numbers generated by a set of eight binary switches vary from 0 to 255.

Therefore 01010101 = 85

We do not wish or need to go into great detail of how the binary figures are worked out. But if you think of the right-hand digit having a value of 1 if it is ON and 0 if it is OFF, the next digit is 2 if ON and 0 if OFF and so on. Each succeeding digit has double the value of the previous one to the right so the left-hand digit in a series of eight (as above) is 128 if ON and 0 if OFF.

In this manual we will only deal with eight registers. These are called the A B C D E H and L registers.

A is the ACCUMULATOR. Used for simple addition and subtraction, comparison as well as general purpose storage.

F is the FLAG register which we will discuss later.

The other six registers are either used singly or in pairs as below

BC DE HL

When used in pairs, the total number stored is between 0 and 65535.

BC, DE, HL are general purpose registers but Coding is often simpler if HL is used to keep track of a program, DE to hold the start address of message and BC to count for a loop.

There are other registers but we will not concern ourselves with them as this is a tutorial for beginners.

THE FLAGS

The F register contains the eight binary bits of information that will signal the current state of affairs.

To keep things simple we will only deal with two flags at this stage. The ZERO FLAG is set if the result of a check is 0. The CARRY FLAG is set if the operation performed takes the relevant register higher than 255 or lower than 0.

Example: If A holds 240 and 20 is added to it, A will now hold 4 and carry flag will be set.

NOTE the flag is set to 1 if the answer is YES and reset to 0 for NO.

INC, DEC, NOP

INC D increases the value of D by 1.

DEC D decreases the value of D by 1.

If a value is decreased lower than 0 the flag will

be set and the value will, like a mileometer, go back to 255. A register pair can be increased or decreased, but the flags will not be set. An increase above 255 sets the flag and goes back to 0.

NOP means 'do nothing' and simply creates space for amendments later.

LD

The LD instruction performs a variety of similar but slightly different functions. The instruction consists of three parts, each separated by a space in this assembler.

The first part of the instruction is LD which you can think of as load, or store in.

The second part of the instruction is the destination. This can be a register or a register pair which is named in capital letters without brackets. It can also be a memory location. If this is the case the second part of the instruction must be in brackets. If you want to name a specific location start the instruction LD (NN) . . . and the program will then ask for NN as a number between 0 and 65535. However the required location may already be stored in one of the register pairs, e.g. HL, and in this case you would write HL in BRACKETS.

After this second part comes a space. Think of this space as the word "with" or "from". For example LD A B becomes "load A from B".

The third part of the instruction may consist of (1) the contents of a register, (2) the contents of the address contained in a register, (3) the contents of a named memory location or (4) simply a number.

Examples of the four possibilities are given by (1) LD A B, (2) LD A (HL), (3) LD A (NN), (4) LD A N.

Sometimes the destination register is a pair. Note that LD HL (NN) will load L from the contents of location NN and H from location NN+1. Also LD (NN) HL will load L into NN and H into NN+1. Near equivalent BASIC commands are:

LD A B	LET A = B
LD (NN) A	POKE NN, A
LD A (NN)	A = PEEK (NN)

PUSH, POP, EX

The vital rule is 'When in doubt, PUSH and POP'.

PUSH takes the contents of a register pair and puts them onto a stack.

POP takes the top contents of a stack and puts them into a register.

When writing a routine, usually start with PUSH and end with POP. All registers used in the

routine should be included and we advise that you think of them as 'nested' like loops in BASIC so first in is last out. Try to PUSH in alphabetical order and POP in reverse order.

PUSHing and POPping can also be used to load or exchange contents of registers but a more common command is EX (for example EX DE HL).

JP

This is the JUMP command rather like a GOTO in BASIC except that the destination is a memory location rather than a line number. You might use the JP to goto the SYNTAX ERROR routine which is 5006. The command would be JP NN 5006. This is stored as 195 142 17. 195 is the JP instruction and 5006 is converted to (17 x 256) + 142. In memory the low byte comes first, followed by the high byte. But in a register pair, e.g. BC, B is high and C is low.

CP, OR, XOR, SUB, ADD

CP means COMPARE and is used for a sort of IF...THEN statement. To check whether or not certain conditions apply, (say does A contain 63), the command is CP N 63. If the instruction is to jump if this condition is met, the previous command will be JP Z NN which means only jump if the Zero flag is set.

These are the conditions we shall use:

Z = Zero flag set	NZ = Zero flag reset
C = Carry flag set	NC = Carry flag reset

The CP command checks the value by deducting (in this case) 63 from the accumulator and if the result is zero the flag is set and the jump takes place. The accumulator is not altered by the CP command.

XOR A sets the accumulator register to 0 and sets the Zero flag.

LD A H followed by OR L sets the accumulator to 0 and sets the Zero flag if, and only if, the register pair HL is 0.

SUB and ADD subtract or add the contents of a named register or a number to the accumulator and change the flags accordingly.

CALL, RET

This is similar to a GOSUB in BASIC going to a subroutine and returniing to the point from where it left.

Both CALL and RET can be conditional using Z, NZ, C or NC as with the JP command discussed previously. It is useful to remember that the monitor contains many useful routines that you can CALL.

IMPORTANT — ALWAYS ISSUE A RET AFTER A CALL TO AVOID DISASTERS.

JR E

This more complex JUMP can be omitted if you are not ready yet.

The JR is a jump of relative distance in bytes rather than to a specific location. The jump can be forwards (up to 127 bytes) or backward (up to 128 bytes) and is measured from the location that follows JRE. The difficulty in backward jumps is that the number of bytes is calculated in reverse starting at 256 so to jump 10 bytes backward, the command would be JR E 246. A forward jump of 10 is JP E 10.

JR can also be conditional by inserting Z, NZ, C or NC between JR and E.

If in doubt about JR, we suggest you stick to JP for the moment.

MZ-700 BANKING

S-BASIC uses a banking system which disables a section of memory when BASIC is loaded. To avoid damaging BASIC, there is an area of memory from 11433 to 12456 that can be used. This does not require the use of LIMIT.

The other method is to CALL 242 and CALL 234. These routines use the HL and A registers only and not any other registers.

SOLO BASIC does not use the banking system so do not use the above CALLs.

DJNZ, LDIR, LDDR, CPIR, CPDR

DJNZ E is comparable to the FOR...NEXT loop in BASIC. When it is used, the B register is decremented and if it is not zero a relative jump is made (normally backwards). The B register is therefore loaded with the of time the loop is to be executed. As an example, the following program will colour the first five lines of the screen in red on white (in SOLO BASIC). In S-BASIC, LD (HL) A should be replaced by CALL NN 242 and an adjustment should be made to the jump.

```
PUSH BC
PUSH HL
LD A N 39
LD HL NN 55296
LD B N 200
LD (HL) A
INC HL
DJNZ E 252
POP HL
POP BC
RET
```

By POKEing the location of the 39 with different numbers and recalling the routine, a flashing effect will take place.

LDIR transfers a block of memory elsewhere in a single instruction. HL is loaded with the bottom block's present location, DE is loaded with its destination, BC is loaded with the number of

bytes to be moved and LDIR does the rest. Sometimes it is essential to use LDDR instead, which decreases HL into DE.

CPIR searches memory for a particular value. HL is loaded with the start point of the search and A is loaded with the value to be searched for. BC is the number of bytes from HL upwards that the search should take before the search is abandoned. Since BC is decremented, (and HL is increased) on every operation of CPIR, the next instruction will only take place when A and (HL) are identical or BC is zero. CPDR is the same except that HL is decreased.

LDI, LDD, CPI, CPD are simpler versions of the above commands that only transfer or compare a single byte.

CRASHES

Crashes are an inevitable fact of life when you start machine code programming.

The main causes of crashes are going into infinite loops, over-running an area of memory unintentionally or disabling an essential function. Fortunately, no damage can befall the computer by crashing but it is possible to lose several hours of programming unless you can recover.

In SOLO BASIC, pressing the reset button and typing J1260 can save the entire memory content. Then try to RUN and if this does not work, try to LIST. You may well find that all is well.

On the MZ-80K which has no reset button, a crash will often take you directly to monitor. Type GOTO$1260 to recover.

In S-BASIC on the MZ-700, pressing reset followed by [CTRL] and reset together normally clears the crash but reverts you to the position where the crash occurred which is not too helpful. The only way is to reset to monitor and use the M option to locate the problem and rectify. However, to do this, you will need to use hex.

WRITING IN MACHINE CODE

There is a library of useful sub-routines built into the monitor of the MZ-700 and a list of these can be found in the manual (pages 151 to 153).

The calls from 0-71 are common to all Sharp BASICs.

As a guide to writing your own programs, in SOLO BASIC you will find two areas that might help or interest you.

The first is the routine at 21500 which is a time-saver to avoid having to continually write a prompt at the end of each page to 'Press any key to continue'. The main areas of work are done by monitor sub-routines. CALL 21 prints the prompt and CALL 27 waits for a key to be pressed.

The second is the colour routines. 20560 to 20575 hold the registers and 20600 to 20999 consist of the various sub-routines employed. The colour command became COL followed by a number from 1 to 9. The next routine at 21000 sends the interpreter to the various locations required for the different colour commands.

Bear in mind that this manual and cassette is only the introduction to machine code and there is much more to learn. However, the information contained here should help you start writing your own assembler programs.

For dedicated programmers, we heartily recommend the PEEKING and POKEING book available for the MZ-700 and this can be obtained from Solo Software. It contains many pieces of information about memory locations and the monitor which is invaluable to both BASIC and machine code programmers.

Answers to Hex questions:

(1) 42	(2) 196	(3) 175
(4) 241	(5) 2754	(6) 1572
(7) 4608	(8) 8191	