STARTING Machine Code on the SHARP

MZ-80K

MZ-80A

MZ-700 SERIES

G. P. Ridley

Starting Machine Code on the SHARP

MZ-80K MZ-80A

MZ-700 SERIES

The Author wishes to express his thanks to Mike Shaw for his contributions to the text and program examples in the book.

First Published 1984

PUBLISHED BY:

D.C. BRENNAN Eng.
14 North Western Avenue
Watford, Herts. WD2 6AE

Contents

Introduction

This book has been written as an introduction to writing machine code programs and routines on the Sharp range of micro computers, MZ-80K, MZ-80A and the MZ-700 series.

Most newcomers to computing begin programming in Basic and for many machine code remains a grey area which most see in program listings as a series of numbers within DATA statements POKEd into high memory locations and then called by the USR command, and are left without a clue as to what is happening. One may have seen program listings in magazines which contain these routines for one model of the Sharp and often wished it could be converted to run on another model. In fact with many of these routines the conversion probabilities are high and there may be only one or two figures which need altering, but one must know what the routine is attempting to achieve in order to know which figures to alter.

Machine code, or assembly language, communicates directly with the Z80 and is far quicker than the Basic language which needs to be interpreted. This is one reason for a Basic program to contain a machine code routine to achieve greater speed, or it could be used to modify the Basic interpreter to do tasks it cannot normally do.

This book hopefully will make Assembly language clearer and more understandable to the average user, one will not need a degree to grasp what is going on and computer jargon, will be kept to minimum levels.

Good Luck.

1 Basic links

The Basic language is generally the simplest way of writing programs, it is easy to follow and debugging a faultv program is usually made quite easy with the editing facilities for altering lines in a program, so why use assembly language?

The main reason must be speed of execution, not purely based on games programs such as space invaders or the like which would not be worth playing if they were written in Basic, but more serious applications which will be shown in the book. In order to grasp some idea of the speed of a program written in assembler we will compare the execution time with a similar program written in Basic.
Whichever Sharp micro you have, K, A or 700, load Basic and enter this short program:-

```
10 PRINT "▣";
20 TI$="000000"
30 FORX = 0 TO 999
40 PRINT"B";
50 NEXT
60 PRINT TI$
```

Now enter 'RUN' followed by the Carriage Return key 'CR'

If one used the MZ-80A for that exercise the time taken to fill the screen should have been 4 seconds. The MZ-700 should have completed it in 2 seconds due to the faster operating speed of the Z80A chip which it uses. The MZ-80K would have required 16 seconds as the Basic Print command is far slower than it could be.

Let us alter the above program so that instead of printing one character after another using the Basic PRINT statement we shall print the characters directly to the screen area of memory using the POKE command.

Add line 15:-

```
15 Z=53248
```

And alter line 40 to:-

```
40 POKEZ+X,2
```

Enter 'RUN' and 'CR'

This time the MZ-80K, as it was not using the Print command, took almost 6 seconds along with the MZ-80A and the MZ-700 took 3 seconds.

Now if that program, although not the most interesting in the world but quite effective as an example, is written in machine code and called by the USR command from Basic the dramatic increase in speed will be instantly obvious.

Program 1 Direct Screen Addressing from Basic

Assembly language instructions used:-

```
LD DE,nnnn     LD HL,nnnn     LD A,nn     LD (DE),r
SBC HL,DE     INC DE     JR NZ,nn     RET
```

These instructions are detailed in chapter 2.

When entering the following program one should be using the standard Sharp Basic for your machine without modifications - i.e. without a toolkit or basic modifier - as one may have code written into the area of RAM we will use. The versions are:-

SP-5025	MZ-80K
SA-5510	MZ-80A
S-Basic	MZ-700

Enter 'NEW' and 'CR' and input these program lines, the left hand listing is for the MZ-80K and MZ-80A and the right hand for the MZ-700 only:-

MZ-80K & MZ-80A

```
10 LIMIT 49151
20 FOR X= 49152 TO 49166
30 READ A:POKEX,A:NEXT
40 DATA 62,2,17,0,208,33,232,211,
  18,19,237,82,32,247,201
```

MZ-700

```
10 LIMIT 49151
20 FOR X= 49152 TO 49172
30 READ A:POKEX,A:NEXT
40 DATA 243,211,227,62,2,17,0,
   208,33,232,211,18,19,237,82,
   32,247,211,225,251,201
```

Enter 'RUN' and 'CR'
The screen will instantly display the 'READY' message and one could be excused in thinking that not much has just happened. But happen it has in that now a Machine Code routine has been placed high in memory, starting at location 49152, which will print the entire screen with the letter 'B' in a fraction of the time taken previously using normal Basic PRINT or POKE statements.

Enter 'NEW' and 'CR' and this one line program:-

```
10 PRINT "▦":USR(49152)
```

Enter 'RUN' and 'CR'
Its speed is amazingly fast. Note that in order to achieve the similar result that the 700 routine took six additional bytes, this and other differences between the relative micros will be discussed later, although the main bulk of the numbers were the same proves that with small alterations most routines listed in programs for a particular Sharp can be converted to run on another Sharp.

As will be seen in the next chapter Assembly language is made up of several registers which we load with addresses and values, you can also check the codes in the Appendix. The previous listing for the K and A in disassembled form would look like this:-

```
1     C000    3E 02          LD A,02
2     C002    11 00 D0       LD DE,D000
3     C005    21 E8 D3       LD HL,D3E8
4     C008    12             LD (DE),A
5     C009    13             INC DE
6     C00A    ED 52          SBC HL,DE
7     C00C    20 F7          JR NZ,C005
8     C00E    C9             RET
```

In the K/A listing we POKEd the DATA into memory starting at address 49152 which if we convert to Hexadecimal gives us C000, use the conversion chart in the Appendix if you aren't sure. The first two items in the DATA line were 62,2 decimal. 62 converts to 3E hex which means we want to load the A register with the value of the next byte which in this case was 2 (line 1 above).

The next three bytes were 17,0,208 which convert to 11,00,D0 hex. 11 signifies Load the register pair DE with the following two bytes in reverse order, low address first, in this instance we want to load DE with the address of the top left corner of the screen (Video RAM) which is D000hex, therefore in reverse order the next two bytes will be 00 D0 (line 2).

These were followed by 33,232,211 which convert to 21,E8,D3. The 21 means Load register pair HL with the following two bytes in low byte first, just as we did with DE above. The address we want to load into HL is the bottom right hand location of the screen with 1 added to it which is D3E8hex, which in reverse order becomes E8 D3 (line 3). The screen has one thousand locations, 25 lines by 40 columns, therefore if the first screen position is known to be D000hex (53248 dec) the bottom right will be 999 positions higher, D3E7 (54247 dec). The reason for adding 1 to this number will be explained shortly.

The next number in the DATA line was 18 which converts to 12hex, this means Load whatever value is in the A register into the contents of the DE register. This line may sound complicated but it is similar to the way we POKE the screen in Basic. On the first time through this routine DE is set at D000, which is top left of the screen, so loading it with the A register which contains 2 is simply the same as POKE 53248,2 which displays the second character in the Display code into that screen location (line 4).

We now want to add 1 to DE so that the next time we loop round it will display the character in the A register in the next screen position, which will be D001 (53249 dec). This is achieved by 19 dec, 13hex which instruction is simply INC DE, increment DE by 1 (line 5).

The following two bytes were 237,82 which convert to ED,52 hex. This command is SBC HL,DE which means subtract DE from HL. The first time round the loop DE contains D000 and HL D3E8 so the result will be 03E8. The second time round DE will contain D001 and HL D3E8, and this time the result is 03E7, going down by 1 each time round and slowly getting nearer to zero (line 6).

The next two bytes 32,247 convert to 20, F7 hex. 20hex is in the relative jump family of instructions, which means the program will jump, not to a directly specified address, but a certain amount of memory locations in relation to its present address if a certain condition is true. 20hex means JUMP if not zero, remember in the previous line SBC HL,DE subtracted DE from HL, it also sets the zero bit of the F register if after the subtraction the result is zero, which is what will happen after the program has looped 1000 times. Therefore before we actually get to that stage when the whole screen will be full of 'B's it will not be zero so we want to loop back and do it all again, so the command is JR NZ, jump back if not zero. The jump is in relation to the program counter which will be at the next instruction location C00E, and we want it to jump back to C005 which is 9 bytes back. Start counting from 0 at C00E backwards, the first byte will be FF at C00D:-

```
1=FF C00D
2=FE C00C
3=FD C00B
```

```
        4=FC C00A
        5=FB C009
        6=FA C008
        7=F9 C007
        8=F8 C006
        9=F7 C005    so the line becomes 20 F7.
```

The reason we loaded the HL pair with D3E8, which is 1 after the final screen location is that we increment DE before testing whether to loop back or not, therefore after printing to the last square on the screen in line 5 register pair DE will contain D3E7, line 5 increments DE so now it will be at D3E8 and then it has HL subtracted from it and if the result is zero, which it will be after printing to each screen location, the program will not jump back but continue to line 8, the final line.

The final number in the DATA lines was 201 which converts to C9hex, this command is RET for return, just as one would use after a GOSUB routine in Basic. Remember that we went to this routine by the USR command which is a Call instruction just like the Basic GOSUB and when its task is completed we enter the RET command to return.

MZ-700 note

The 700 program, although similar, contained 6 additional bytes. The first three 243,211,227 convert to F3,D3,E3. When using S-Basic the Video Ram area is switched out to give the user more free RAM for programs therefore before attempting to access the V-RAM these three bytes must be entered to enable the V-RAM Area to be used. The opcodes are:-

```
        DI           ;F3
        OUT ($E3),A  ;D3E3
```

and after the routine to directly address the V-RAM the other three bytes are entered before the final one. 211,225,251 convert to D3,E1,FB and must always be entered after accessing the V-RAM to return to normal.

```
        OUT ($E1),A  ;D3E1
        EI           ;FB
```

Now that routine although it executed in a fraction of the time it took using Basic was not quick to program, and a lot of thought would go into producing a simple output such as that. It is also more complicated translating decimal values back to hex and then translating them into assembly language Mnemonics and operands. If one is to explore assembly language in more depth it would be a wise decision to invest in a Disassembler tape which will do most of the dirty work for you and produce a printout such as we have just seen, furthermore entering assembly language is made childs-play, well almost, if one purchases one of the assembler packages which allows one to enter opcodes and operands such as:- LD HL,D000 directly. After entering the listing one selects the assemble option and the assembler will then translate all the instructions into machine code automatically and output a version known as Object code. This small piece of jargon simply means assembled machine code ready to record on tape for future loading. It is virtually impossible to write machine code programs of any size without an assembler, it will pick up any false statements just like Basic does with the Syntax errors and it will allow one to run the programs and use breakpoints to stop the running at certain points so that one may check on the state of the registers etc. This is most important as the programs run so fast it would be difficult to make these checks without the facility. Furthermore one can add labels to any area in the program and simply enter a line JUMP to label, one would not have to know the exact address as the assembler would calculate its whereabouts and move to that location. The relative jump we made in the last program would also be calculated automatically, one could call it LOOP, and simply enter a line JR NZ,LOOP. And the most important asset is that lines can be added into the middle of a program, just like Basic, one would find that very difficult even in the previous program and that only contained 15 bytes.

In later examples of machine code we will use an Assembler, Editor and Debugger called 'ZEN'. There are several types of Assemblers one can use with the Sharps and most operate in a similar manner to 'ZEN' and if you have a another type you should find the examples easy to enter with an alternative Assembler.

Program 2 Block Move using Basic

New instructions used:-
LD BC,nnnn   LDIR   JR nn

There are four instructions which allow blocks of memory to be copied into other areas of memory we are going to use one of them here. At this stage one could be excused for not appreciating their usefulness fully, but when one gets into writing complete machine code programs ones attitude could alter. One example could be to copy the complete screen area to another part of memory, and when needed move it back to the display area immediately. One may have a program which is menu driven in which options the user can make are listed on screen. That complete screen display could be stored somewhere in RAM and when needed a USR call will immediately transpose that block of memory back to the screen in a flash. Once again there are small additions to the 700 listing which is on the right. Enter 'NEW' and 'CR'

MZ-80K & MZ-80A

```
10 LIMIT 45055
20 FOR X= 52992 TO 53011
30 READ A:POKEX,A:NEXT
40 DATA 33,0,208,17,0,176,24,6
50 DATA 33,0,176,17,0,208,1,232,
   3,237,176,201
```

MZ-700

```
10 LIMIT 45055
20 FOR X= 52992 TO 53017
30 READ A:POKEX,A:NEXT
40 DATA 33,0,208,17,0,176,24,6
50 DATA 33,0,176,17,0,208,1,
   232,3,243,211,227,237,176,
   211,225,251,201
```

Now enter 'RUN' and 'CR'
Once again the 'READY' message or 'Ready' if you have an A or 700 was displayed almost immediately, and we now have this block move routine in memory.

One does not need to write a separate program to demonstrate this program, providing there is a fair amount of text presently on the screen, if there isn't put something on the screen, anything.

Enter in direct mode (without a line number) USR(52992) and 'CR'. The 'Ready' will be displayed instantly and the total displayed area has been copied into memory locations B000 to B3E7 hex. It has not dissappeared off the screen it has been duplicated into the other area. If one was running a program the screen could now be cleared and the program continue until one needed to bring back the previous display.

Now clear the screen by entering 'SHIFT' and 'CLR' and to prove the point enter some characters onto the screen, it does not matter if one gets 'Syntax error' printed just get something on the screen.

Enter in direct mode USR(53000) and 'CR'
The screen will instantly change back to the previous display which was saved when we entered USR(52992)

One could save more than only one screen, providing they were moved to separate areas of memory, each screen contains 1000 bytes so one will need to adjust the program for different storage areas. Let us look at the assembled listing, remember it started at 52992 dec which is CF00 hex:-

```
MZ-80K & MZ-80A                          MZ-700
CF00  21 00 D0    LD HL,D000             CF00  21 00 D0    LD HL,D000
CF03  11 00 B0    LD DE,B000             CF03  11 00 B0    LD DE,B000
CF06  18 06       JR 06                  CF06  18 06       JR 06
CF08  21 00 B0    LD HL,B000             CF08  21 00 B0    LD HL,B000
CF0B  11 00 D0    LD DE,D000             CF0B  11 00 D0    LD DE,D000
CF0E  01 E8 03    LD BC,03E8             CF0E  01 E8 03    LD BC,03E8
CF11  ED B0       LDIR                   CF11  F3          DI
CF13  C9          RET                    CF12  D3 E3       OUT ($E3),A
                                         CF14  ED B0       LDIR
                                         CF16  D3 E1       OUT ($E1),A
                                         CF18  FB          EI
                                         CF19  C9          RET
```

MZ-700 note

Once again as we are directly addressing the Video Ram area the routine requires the memory configuration change as we did in program 1, this technique is lightly touched upon on page 127 of the 700 manual, but as one can see it is always the same three codes used, 243,211,227 dec to start with which convert to F3, D3, E3 hex, the operands can be seen listed, and 211,225,251 dec to end with before the RET instruction which convert to D3, E1, FB.

To copy the screen display to the storage area we first Load register pair HL with the first location of the screen area D000. Secondly we Load register DE with the first address in the area we are transferring the screen contents to, in this case B000. As we have two entry points, one for copying the screen into the storage area and the other for bringing it back to the screen the only differences are the addresses contained in HL and DE, the copying routine remains the same so we can use that part for transfers in both directions. The next instruction is a relative jump, JR, as we used in program 1 only this time there are no conditions to be met as before, the program simply jumps so many bytes in relation to the program counter. As before the program counter is at the next instruction (on the next line) so we count from there only this time it is forwards and not back. In the listing the PC is at CF08 and we want to skip to CF0E as the next two instructions simply load HL and DE with the addresses for copying back to the screen, so we want to miss them. Simply start counting from zero forwards:-

```
CF08=0
CF09=1
CF0A=2
CF0B=3
CF0C=4
CF0D=5
CF0E=6   so the line is JR 06.
```

Next BC gets Loaded with the amount of memory locations we wish to transfer, in this case 1000 which converts to 03E8. Now comes the clever bit, LDIR tranfers or copies the memory contents of whatever is stored at the address of HL down to the memory address of DE. Each time a transfer of one byte occurs so BC gets decremented and HL and DE get incremented. This transfer continues until BC equals zero.

The first location to get moved is the top left square of the screen, at this stage the DE and HL registers increase by 1 and BC decreases by 1, so now HL is at D001, DE is at B001, and BC is 03E7 and the routine continues so that the next screen location to be moved is the one to the right of the top left position, and gradually the routine works its way down through the whole screen area until HL=D3E7 (the bottom right position of the screen), DE=B3E7 and BC=0. Afterwhich a return is made back to Basic.

To recall the stored screen we enter the routine at a slightly higher address, at CF08 hex by USR(53000). This routine first loads HL with the start address of where the memory contents are to be copied from, B000. Then DE is Loaded with the start address of the destination area, which is the start of Video Ram, D000. BC once again is Loaded with the amount of memory locations to actually move 03E8. Then the move instruction is used again, if in a different routine one wanted to transfer only 10 bytes instead of 1000 one would load BC with 000A instead of 03E8.

Therefore at the start of this routine the contents of B000 are copied into address D000 and BC=03E8. Then the contents of B001 go into D001 (the second screen position from the top left) and BC decrements to 03E7. This loop again continues until BC is down to zero and HL=B3E7 and DE=D3E7 being the bottom right screen position, job done. Once again Return takes one back to where one came from, in the example on the previous page back to Ready, it could under program control have returned and carried on with the program currently running.

MZ-80K Note

When addressing the screen directly in this manner, or indeed when any POKEing to the screen takes place, one will notice a 'snow' effect over parts of the screen. One simple method of eliminating this is to temporarily switch off the screen and this is how it's done. To copy the screen contents one entered USR(52992). If this is altered, either in direct mode or within a program line to:-

```
POKE 59555,0:USR(52992):POKE 59555,1
```

then the screen will blank out, although it is hardly noticeable, and switch back on again after transferring so cutting out the snow. The actual copying takes place while the screen is disabled, but it is fast, try it.

MZ-80A Note

The screen map of the A is different so far as the top of the screen is not always address D000 (53248 dec). After a clear screen has been entered it is, but when the screen begins to scroll so the top left corner address begins to change. If it scrolled up 5 lines before one entered the copy screen routine the top left would be D0C8 (53248+5*40=53448 dec) and therefore the bottom left position would be D0C8+03E7 =D4AF (54447 dec). The address of the start of Video Ram is stored at 117D hex, and the more scrolling that takes place so the value held is altered. A solution could have been to alter the first loading of HL to LD HL,(117D) which would load the contents of 117D into HL, so pointing to the top left address, but if the screen was scrolled up to its highest this figure does not actually relate to the start of 1000 bytes of V-RAM, the screen area goes up to D7FF and then restarts at D000 which would not work correctly. The solution is when storing a screen of information make sure that screen was printed after a clear screen and that no scrolling has taken place, one has only printed to a maximum of 25 lines. Similarly when the screen is to brought back from storage ensure that a clear screen is entered before entering USR(53000).

Now we have the facility to store one screen in memory it is quite easy to modify the program to cater for four screens that can be brought back to the screen in a flash.

Simply LIST the program and using the cursor keys edit the following lines.

<u>MZ-80K and A</u>

```
    20 FOR X=53024 TO 53043
    Line 40 change 176 to 180
    Line 50 change the first 176 to 180 and RUN
```

<u>MZ-700</u>

```
    20 FOR X=53024 TO 53049
    Line 40 change 176 to 180
    Line 50 change the first 176 to 180 and RUN
```

We now have a second routine for saving another screen in storage, this is saved by USR(53024) and brought back to the display by USR(53032). The first program entered the routine at CF00 (52992 dec), this one is at CF20 (53024). In order to complete the four routines alter once again the same lines.

<u>MZ-80K and A</u>

```
    20 FOR X=53056 TO 53075
    Line 40 change 180 to 184
    Line 50 change the first 180 to 184 and RUN

    20 FOR X=53088 TO 53107
    Line 40 change 184 to 188
    Line 50 change the first 184 to 188 and RUN
```

<u>MZ-700</u>

```
    20 FOR X=53056 TO 53081
    Line 40 change 180 to 184
    Line 50 change the first 180 to 184 and RUN
```

```
   20 FOR X=53088 TO 53113
   Line 40 change 184 to 188
   Line 50 change the first 184 to 188 and RUN
```

Now the four different screens can be stored and restored by entering any of the following:-

```
USR(52992)  to bring back USR(53000)
USR(53024)    "     "     " USR(53032)
USR(53056)    "     "     " USR(53064)
USR(53088)    "     "     " USR(53096)
```

If one stores four screens in memory and perhaps enters a page number on each the following program will demonstrate how fast each one can be brought back to the display. Enter 'NEW' and 'CR'.

```
10 PRINT "ENTER PAGE No."
20 GETA$:IFA$=""THEN20
30 IF(VAL(A$)<1)+(VAL(A$)>4)THEN20
40 PN=VAL(A$)
50 PL=PN*32-32
60 PRINT"▣";
70 USR(53000+PL)
80 GOTO10
```

Notice that each routine is 32 bytes apart therefore after selecting the page number line 50 will calculate the amount of bytes to add to 53000. If page 1 was selected then multiplying 1 by 32 and then subtracting 32 would make the USR call 53000+0 making 53000, if it was page 2 then multiplying by 32 would give 64 and then subtracting 32 would make the USR 53032 and so on.

MZ-80K Note

```
   Line 70 could be altered to eliminate the snow to:-
   70 POKE59555,0:USR(53000+PL):POKE59555,1
```

2

The C.P.U.

In this Chapter, we're going to take a broad look at the way the Z80 chip interprets the machine code numbers, the Z80 Registers and the way they are generally used, and then at the different types of Assembler instruction. You'll find a complete list of these mnemonic instructions in the Appendices - listed alphabetically and numerically by the first byte of their instruction code. There are several books available which explain each Z80 instruction in greater depth, rather like an encyclopaedia and almost as large, but these are general references and do not show examples for specific micros like the Sharp. However if one requires more detailed information regarding the Z80 instruction set then the purchase should prove worthwhile.

BASIC has about 200 instructions - if you count all the subtle variations like 'IF-THEN GOSUB' and 'IF THEN PRINT'. Z80 machine code has nearly 700 - but don't panic, many of them are simply variations on a theme.

The difference, as you will have already appreciated, is that one BASIC instruction calls up a host of machine code instructions within the interpreter. When you write in machine code you have to generate those instructions yourself - although you can, of course, call up useful routines resident in the monitor (as indeed some of the demonstration programs in this book do).

It is possible to write programs without having a full knowledge of the entire instruction set - indeed many people do quite happily and successfully, adding to their knowledge as they gain experience. The same is true to some extent when programming in BASIC.

For example - how would you do a count of 1 to 1000 in BASIC? Probably:-

```
10 FOR I=1 TO 1000
20 NEXT
30 PRINT "ALL DONE"
```

Fine, but supposing you didn't know about FOR-NEXT loops? You'd probably tackle it this way:-

```
10 A=0
20 A=A+1
30 IF A<1000 THEN 20
40 PRINT "ALL DONE"
```

But supposing you didn't know about IF-THEN constructions either. You'd really have to put your thinking cap on:-

```
10 A=0
20 A=A+1
30 B=-1*(A<1000)-2*(A=1000)
40 ON B GOTO 20,50
50 PRINT "ALL DONE"
```

As you can see, the programs become longer - and take longer to run - when the most suitable commands are not used. Knowing all the commands at your disposal helps you to make your programs shorter and/or faster running...and your life easier. Usually machine code programs run fast enough even when written the 'long way round', but when a very large number of repetitive actions are involved, such as in a Chess Game program, even a few microseconds knocked out of a loop can result in a considerable time saving when the program is running.

Having said that, the programs in this book have been written to demonstrate principles, and are not necessarily the fastest or shortest way of achieving the desired result.

WHAT DO ALL THE NUMBERS MEAN?

Machine coding, as you know, is all about numbers. A number can mean one of two things to the Z80 central processing unit in your computer. It can mean an instruction or part of an instruction to do something. Or it can mean a piece of information to be worked on or used in some way. Fortunately, the Z80 knows exactly which of these the number represents (in a correctly written program), and acts accordingly.

Take an instruction to load Register A with the value '7' (we'll be discussing the Registers in more detail later). In Assembly language mnemonics this instruction is written LD A,7 . In machine code language, the instruction is represented by the two hex numbers '3E 07'. When the Z80 sees the first of these it says "3E means I must load the next number along into Register A". It takes up the 7, puts it into Register A, then looks to the number after the 7 for the next instruction. So it wouldn't be confused if it saw, for example, the two hex numbers '3E 3E' - this time it would load 3E hex (62 decimal) into its Register A, then look to the number after the second 3E for its next instruction.

Note that each single byte of information can have a value from 0 to FF hex (0 to 255 decimal). Let us take a look at that in more detail.

A byte consists of 8 bits, each bit being a binary 0 or 1. So the binary number 11001001 can be represented thus:-

```
          Bit No:  7 6 5 4 3 2 1 0
    Binary Value:  1 1 0 0 1 0 0 1
```

Wherever a '1' appears in the binary representation, raise 2 to the power of the corresponding Bit Number, add the results together, and you have the decimal value of the Binary number. Thus, using the above example:-

```
 2 to the power 7  = 128
 2 to the power 6  =  64
 2 to the power 3  =   8
 2 to the power 0  =   1 (any no. to the power 0 = 1)
                     ---
                     201
```

So the binary number 11001001 is 201 in decimal.

To convert a binary number to Hex, split the eight digits into two groups of four (called 'nibbles'). Thus:-

```
   Nibble 'bit' no.:  3 2 1 0   3 2 1 0
      Binary value:   1 1 0 0   1 0 0 1

      Left side:  2^3 = 8    Right side:  2^3 = 8
                  2^2 = 4                 2^0 = 1
                        --                      --
                        12                       9
```

Remembering that decimal 12 = C in hex, the hex value of binary 11001001 is C9.

HOW THE Z80 HANDLES 2-BYTE NUMBERS

Many instructions to the Z80 tell it to operate not on one byte - as in our 'LD A,7' - but on two bytes. For example, an Assembly instruction might be 'LD HL,49AFH' (the 'H' at the end tells the Assembler that 49AF is a hex number). Two-byte numbers increase the decimal values that can be represented from 0-255 to 0-65536 (0-FFFF hex) - which is absolutely vital for addressing or pointing to the memory locations in your computer.

In the instruction LD HL,49AFH, we want the High byte, 49 (hex) to go into the H Register, and the Low byte AF (hex) to go into the L Register. The machine code instruction for loading H and L Registers with 'direct' data is 21 hex. When the Z80 sees 21 hex as an instruction, it takes the NEXT number and loads it into the L Register. That's right - the L Register. Then it takes the following number and loads it into the H Register. So the machine code for LD HL,49AFH looks like this :-

```
          21 AF 49 (hex)
```

Note how, in actual machine code, the order of the two information bytes is reversed. Now you know why.

When using an Assembler, you don't have to worry about this point - the Assembler sorts it out for you. But if you are entering machine code by hand, as was shown in chapter 1, forget the order of the two information bytes at your peril.

Needless to say, when loading any Register pair with data (we'll discuss Register pairs later on), the Low byte always appears in the machine code listing before the High byte. In Assembly language remember, you write the number in the normal way, and let the Assembler put things in the correct order.

The elements that go to make up a Z80 chip include an Arithmetic-Logic-Unit, which performs all the (simple) arithmetical and logical functions, a 'control box' which makes sure data is passed in, decoded and acted on in the correct order, and a number of 8-bit (one byte) and 16 bit (two-byte) Registers. Just to confuse you, pairs of the one-byte Registers can also be used as two-byte Registers.

The Program Counter

Let us look first at the Program Counter (PC) two-byte Register. This holds the address of the NEXT instruction. It is automatically up-dated every time a new instruction is executed. However, the address it holds can be changed by, for example, a CALL instruction (like GOSUB in BASIC).

In this case, the address in the Program Counter is put aside - on the STACK - and the address CALLed is put in the Program Counter in its place. When the CALLed routine is done it meets a RET (RETURN) command, which takes the two-byte number ON THE TOP OF THE STACK and puts it back into the Program Counter. Execution then continues from that address. If you use the Stack (and you will use it), it is important to remember that the next instruction address after a RETurn is taken from the top of the STACK. Many a program has gone wild because a number has been unwittingly left on the stack: on the other hand, the fact that you know that the address of the next (apparent) instruction is on the Stack can be useful when, for example, transferring data to a subroutine.

A number of other instructions also affect the PC Register - jump instructions (JP or JR) for example. But for most instructions, the length of the instruction (including any information data elements) is added to the PC by the chip's control system, so that it knows where to look for the next instruction.

The Stack Pointer

Another two-byte Register, the Stack Pointer (SP), keeps track of the top of the Stack - since many instructions enable you, as well as the Z80, to use the Stack. The Stack area is within the RAM of your computer - and an address is set up by the ROM Monitor routines when you switch on. It is 10F0 hex on all three Sharp machines. (On the MZ700 with S-BASIC loaded, it's at FE00 hex).

You can if you wish set up your own address for the Stack but you must remember that the Stack runs BACKWARDS in memory, and it uses a last-in, first-out system. Think of it as a pile of plates, you can put plates on top or take them off the top, but you can't touch the plates anywhere else in the pile.

The other point about the Stack is that it ALWAYS accepts or delivers two-bytes of data. So, if we put 11A0H, 22B0H and 33C0H on the Stack in that order, it will look like this:-

Address	Contents
10EB	C0
10EC	33
10ED	B0
10EE	22
10EF	A0
10F0	11

The Stack Pointer in the Z80 will be pointing to the last (low) byte of the 33C0H data. If another piece of two-byte data - say 4567H - is put on the Stack, the Stack Pointer is DECREASED by one (decremented), the first (high) byte 45 hex is put into the address now pointed to by the Stack Pointer (10EA), the Stack Pointer address is DECREMENTED again, and then the low byte of the data, 67 hex, is put on the Stack (at 10E9).

When taking data off the Stack, the system works in reverse. In our example, first the Low order byte (67 hex) is removed, the Stack

Pointer is INCREMENTED, the high order byte (45 hex) is removed and the Stack Pointer INCREMENTED again. So now the Stack Pointer is once again pointing to the low order byte of the 33C0 hex data.

The 8-Bit Registers

There are two sets of 8-bit Registers:-

A, F, B, C, D, E, H, L

and A',F',B',C',D',E',H',L'

(Notice the F and F' Registers have been put next to the respective A Registers - that's because they are usually associated with the A Registers, and they have a function all of their own).

Only one set of these Registers can be used at a time. Why have two sets? So that you can 'stop' in the middle of one operation, switch to the alternate set, carry out an intermediate operation, then switch back and continue with the original operation. There are several ways of passing data between one set and the other.

Registers B and C, Registers D and E, and Registers H and L are also used as Register pairs to hold two-byte data. In a few commands, Registers A and F are also treated as a pair.

The A Register

The A Register is the Accumulator. It's where Almost All of the Action takes place. It is like Grand Central Station and in any program of consequence, it is kept extremely busy. Practically all comparisons, single-byte adding and subtracting instructions, and many special 'transfer' and 'load' instructions demand use of the A Register. God bless its cotton socks.

The B and C Registers

Several commands use the B Register or the B and C Registers together as a Byte Counter. (BC = Byte Counter - easy to remember).

Take for example the DJNZ Assembly command, which must always be followed by a Label. This instruction says 'Decrement whatever value is held in Register B by 1, and if it is NOT zero as a result, jump to the address denoted by the Label'. It's like a FOR-NEXT loop in BASIC, with the number of repetitions required being held in Register B. When B reaches zero, processing continues with the next instruction. (Note the mnemonic DJNZ = Decrement and Jump on Non Zero).

Similar commands (e.g. 'LDIR') use Registers B and C as a pair - permitting for example the transfer of large or small chunks of data from one area in the computer to another extremely quickly. The number of bytes to be transferred in this way is held in the Register pair BC.

Apart from these special uses, these two Registers can be used together or independently for your own requirements.

The D and E Registers

These too can be used independently, but are used together by some Z80 instructions to define a DEstination address. For example, the DEstination address of a block transfer of data (the 'LDIR' command again) is taken from Register pair DE: you have to put the address there, of course.

The H and L Registers

These Registers are used as a pair for quite a number of Z80 instructions. In the 'LDIR' command, for example, the start address of data to be transferred is taken from the contents of HL Registers - so don't forget to put it there. You'll find that there are quite a few commands which allow you to use the HL Registers to 'point' to data areas.

The F or 'Flag' Register

This is a very important Register indeed. Unlike the other 8-bit Registers, you cannot load data into it in the normal way. Its purpose is to hold Flag results of any logic and arithmetic operation undertaken, and for some other instructions, to 'flag' a status. The important point is that some of the Flags can be 'tested' to provide, for example, conditional jumps, calls or returns.

NOTE THAT WHILE MOST OF THE INSTRUCTIONS AFFECT SOME OR ALL OF THE FLAGS, FLAGS REMAIN IN A CURRENT STATE UNTIL AFFECTED BY A SUBSEQUENT INSTRUCTION. This means the state of a Flag can be tested several instructions after the instruction that affected it - but do be sure that the intermediate instructions do not affect the Flag in question. This feature can help to reduce the amount of coding needed. For example, all but two of the 'load' instructions do not affect the Flags at all. So if one of two subroutines are to be called, depending on the status of a particular Flag, and if both subroutines require the same 'load' at their start, then the 'load' can be done before the conditional test is made.

Certain bits of the Flag Register are allocated to specific functions, as follows:-

Bit Number:	7	6	5	4	3	2	1	0
Function:	S	Z	-	H	-	P/V	N	C
Testable:	*	*				*		*

The 'Testable' line indicates which of the Flags you can test in one way or another using the instructions available. Now we'll look at the functions of each one-bit flag.

The S or Sign Flag

This Flag 'repeats' the value of the most significant bit in the result of an arithmetic or logic operation, including 'shifts'. When a byte is transferred into the A Register, it 'repeats' the value of the most significant bit of that byte.

In many instances, bit 7 (the most significant) is used to indicate a particular condition. In 'two's complement' notation, for example (a brief discussion of which is given later in this chapter), bit 7 represents the SIGN of the number. This means the binary numbers are only 7 bits long, but represent from -128 to +128. In this instance, Bit 7 is 'SET' (equal to a '1') if the number is NEGATIVE and 'RESET' (equal to '0') if the number is POSITIVE. Bit 7 of a data byte can also play a role when a program is 'communicating' with input/output devices, such as a Printer. The S Flag enables Bit 7 of such a byte to be tested.

A number of Assembly commands allow the S Flag to be tested, by adding a 'P' (is it Positive?), or an 'M' (is it NEGATIVE?). The command JP (Jump), for example, can be turned into a CONDITIONAL jump by the addition of P - ' JP P,Label'. This tests the S Flag, and if it IS positive (i.e., equal to zero) as a result of some previous action, then the jump will occur. Otherwise processing continues with the next instruction.

The Z or Zero Flag

This Flag is used to indicate whether or not the result of an arithmetic operation is zero, or whether or not a 'comparison' test succeeds.

When a result is Zero or a comparison test succeeds, the Z Flag is set to a '1'. Otherwise, it is reset to a '0'.

The Z Flag can be tested by adding 'Z' (is it Zero?) or 'NZ' (is it Non-Zero?) to certain Assembly commands. For example, 'RET Z' (RETurn on Zero) provides a conditional return from a subroutine: if a previous operation has left the Z Flag set to '1', a RETurn will be made. Otherwise processing will continue with the next instruction. (As you can see, you don't have to worry too much about the actual value of the Z Flag bit - the Z80 looks at it and acts accordingly on your behalf).

The H or Half-Carry Flag

This Flag is used by the computer during Binary Coded Decimal arithmetic operations, to indicate whether or not there's been a carry from bit 3 to bit 4. It cannot be used in any conditional tests.

The P/V or Parity Overflow Flag

This Flag has three functions. Some instructions set or reset it according to whether the byte of a result has an even number of '1's (Parity Even = Flag set to "1"), or an odd number (Parity Odd = Flag reset to "0").

The second use of the P/V Flag is to indicate, during Binary Coded Decimal operations, whether or not Bit 7 (the 'Sign' Bit) has been affected by an overflow from Bit 6, thus accidentally changing the sign of the result.

Finally, during block transfer instructions, such as 'LDIR', this Flag is used to detect whether the counter has reached zero.

The Flag can be tested by adding 'PO' (is the Parity Odd?) or 'PE' (is the Parity Even?) to commands used to transfer program execution. For example, a CALL command can be turned into a conditional CALL if the Parity Flag is indicating 'odd', by writing 'CALL PO,Label' instead of the unconditional command 'CALL Label'.

The N or Subtract Flag

This Flag is used by the Z80 during its own Binary Coded Decimal calculations, and cannot be tested.

The C or Carry Flag

This Flag plays a dual role. First, it is used to indicate whether or not an addition or subtraction has resulted in a 'borrow'. If a borrow has occured, the Flag is set to "1". Otherwise it is reset to "0". Since comparison commands (e.g. CP B - which compares the contents of Register B with the contents of Register A) are achieved by subtracting the selected Register from Register A (and discarding the result), the Carry Flag can indicate whether the selected Register has a value greater than that in Register A (which produces a Carry), or has a value equal to or less than that in Register A (which produces a No Carry). Very useful.

The second use of the Carry Flag is in many of the rotate and shift instructions - which move data along the byte one way or the other in a particular manner. For these instructions, the Carry Flag is used as a 'ninth' Bit. For example, the RRA Assembly command (Rotate Right the Accumulator - Register A), moves Bit 0 of Register A into the Carry Flag, moves whatever was in the Carry Flag into Bit 7 of Register A, moves what was in Bit 7 to Bit 6 - and so on. Thus, this particular command effectively rotates the information held by the bits round one and includes the Carry Flag in the process.

With logical commands AND, OR, XOR, the Carry Flag is always set to '0' (No Carry). AND A and OR A will leave Register A intact, since the Register is being ANDed or ORed with itself, whilst XOR A not only clears the Carry Flag but also clears Register A, as there can be no 'exclusive' bits if it is being XORed with itself.

The Flag can be tested to produce conditional commands by the addition of 'C' (Carry) or 'NC' (No Carry) to the command. Thus a CALL command can be turned into a CALL if the Carry Flag is set, by writing 'CALL C,Label' instead of 'CALL Label.

How the Commands affect the Flags

The following Table shows how the Flags are affected by various types of Command. Commands not listed - e.g. 'PUSH' and most 'LD' commands - do not affect the Flags at all. Please note that, where unnecessary, the 'Register' element of the Command has not been included in the Table: thus the OR command could be OR A, OR B, OR C and so on - all having the same effect on the Flags. Only those Flags that can be tested have been included.

	FLAGS			
	C	Z	P/V	S
COMMAND				
ADD A,ADC,SUB,SBC, CP,NEG	?	?	?V	?
AND,OR,XOR	0	?	?P	?
INC,DEC	-	?	?V	?
ADD RR,CCF	?	-	-	-
RLA,RLCA,RRA,RRCA	?	-	-	-
RL,RLC,RR,RRC, SLA,SRA,SRL,DAA	?	?	?P	?
SCF	1	-	-	-
IN	-	?	?P	?
INI,IND,OUTI,OUTD	-	?		
INIR,INDR,OTIR,OTDR	-	1		
LDI,LDD	-		?	
LDIR,LDDR	-		0	
CPI,CPIR,CPD,CPDR	-	?	?	?
LD A,I; LD A,R;	-	?	IFF	?
BIT	-	?		

KEY:

?	= Depends on the result of the operation.
?P	= Depends on the Parity of result
?V	= Depends on overflow in result
0	= Flag reset to zero
1	= Flag set to 1
-	= Flag unaffected: previous state retained
IFF	= Contents of interrupt flip-flop

Where there are blanks, the Flags contain irrelevant information.

To summarise the conditional tests available for JumP, CALL,Jump Relative and RETurn commands:

Z	= If result is Zero, act.
NZ	= If the result is Not Zero, act.
C	= If there's a Carry, act.
NC	= If there's No Carry, act.
PO	= If Parity is Odd, act.
PE	= If Parity is even, act.
P	= If the Sign Flag is 'positive (S=0), act.
M	= If the Sign Flag shows a minus (S=1), act.

The Index Registers IX and IY

We now come to two very valuable 16-bit Registers in the Z80, the 'Index' Registers. Unlike Registers A to F, there is no 'second set' of Index Registers: their contents are accessible to both of the A to F Register sets.

The 'load' instruction commands related to these Registers can (indeed must, even if it's 0) include a displacement value. This enables, for example, data tables to be very easily set up, using the Register IX or IY to point to a 'base' address, and the displacement to point to the particular place required in the table.

An example will help to explain this. Supposing we decide to have a

Table of information that contains a number of names, addresses and telephone numbers. We allocate, say, 20 bytes to cover the name data, 60 bytes to cover the address data, 12 bytes to cover the telephone number data.

Our Table will then consist of a series of chunks, each 92 bytes long (20+60+12). We know that the telephone data for any name begins at the 80th byte from the start of the name. If we 'point' the IX Register to the start of the name in the Table, we know that the Telephone data will start at IX+80. This saves counting out the bytes to get to the correct address. A typical program might look like this:-

```
         LD B,11
         LD IX,NAME3
         LD DE,BUFFER
  GETTEL:LD A,(IX+80)
         LD (DE),A
         INC IX
         INC DE
         DJNZ GETTEL
         Next operation
```

The first instruction sets up Register B as a counter.

The second instruction loads up the IX Register with the 2-byte address we require - that for NAME3.

The next instruction loads up Registers DE to point to a BUFFER area, where we want to hold the Telephone number - possibly for printing out.

We then come to the start of a little loop which will collect the bytes of data from the Table. We collect one byte, then increment the value in the IX Registers, increment the value in the DE Registers (i.e move both to point to the next address along), then collect another byte and so on until our 'counter', B reaches zero.

Note that LD A,(IX+80) means load Register A with the data byte to be found at the address pointed to by IX+80. Similarly, LD (DE),A means load the data byte in A into the address held in the Register pair DE.

The IY Register can, of course, be used in a similar way. As well as 'loads', the Index Registers can be used for ADD, INC, RLC, BIT and SET commands - INC (IX+80), for example, means go to the address pointed to by IX+80. and whatever byte is stored there, add one to it.

How big can the displacement value be? Glad you asked - because the displacement value is treated as a signed number. That means it can be 7 bits long, with the Most Significant Bit representing the sign of the value. So, to answer your question, the displacement value can be anything from -128 to +127, '0' being treated as a positive value.

The I and R Registers

Two more 8-bit Registers exist in the Z80 which can be accessed by commands. These are 'I', which stands for the Interrupt-Page Register, and 'R', which is the Memory-Refresh Register.

The I Register is used in a special interrupt mode of operation to which the Z80 can be set (by command), and it stores the high-byte of an address that will be called in the event of an 'interrupt' process. The low-byte is generated by the device generating the 'interrupt'.

Let us very briefly examine the concept of an interrupt. When you write a program, providing all is well, it will run the way you want it to, branching to subroutines and returning to the main program as scheduled. However, some input/output devices demand attention even while your program is running quite happily. The 'internal clock' in your Sharp Computer is one of these 'devices'.

An interrupt signal is sent by the device to the Z80. It says 'Hang on, I need attention'. Your 'main' program stops while the interrupt request is attended to - it may be to update the am/pm detail - and then control is passed back to the main program, to continue where it left off.

The programmer can call on the interrupt process himself, and indeed, you'll find an interrupt 'Vector' or Jump Address is provided within the monitor routine of your Sharp.

There are three interrupt modes, called up by the commands IM 0, IM 1, and IM 2. In Interrupt Mode 0 - which is the mode your machine is in when you switch on - the external device must provide the instructions for what it wants the Z80 to do when it makes an interrupt request.

In Interrupt Mode 1 (which is the mode your monitor places the Z80 within microseconds of you switching on), when an interrupt request occurs an automatic jump is made by the Z80 to memory address 38 hex. The current location of any program running at the time is, of course, temporarily stored so that after the interrupt routine is complete, a return can be made to the original program. This interrupt mode always calls, to address 38 hex. In Sharp machines, this is in the ROM monitor - and it invokes a jump to address 1038 hex in the monitor's RAM work area. In Sharp machines, there is a jump from this address to a time-keeping routine (the actual location varies with each machine) which flips the stored a.m. information to p.m. This particular interrupt is invoked every 12 hours of running time (pretty obvious, that), which means if you use the jump at 1038 hex to go to a routine of your own, your routine will also be called when the clock demands attention.

The third mode operates in a similar manner, except that it starts by going to one of 128 addresses (instead of one), as supplied by the calling device in conjunction with the contents of the I Register. Note that bit 0 of the address byte from the calling device is always zero. The address pointed to, plus the next address, provide the 2-byte address of the interrupt handling routine, to which control is then passed.

In some programs it may be necessary to ensure that an interrupt does not occur during a specific process: a Dissable Interrupt command (DI) lets you do this - but for heaven's sake remember to Enable Interrupts (EI) again when that part of your program is complete. (MZ700 users may be familiar with this when calling the Video Ram from a machine code program whilst BASIC is loaded).

Finally, the Refresh 'R' Register: this is provided to refresh dynamic memories automatically. You can use this as a kind of 'software clock', but since its values run only from 0 to 255 decimal, it's not exactly the most useful Register available.

THE ASSEMBLY COMMANDS

There are a number of ways to classify the many Assembly commands you have at your disposal. We are going to use groupings which tally to some degree with the Z80 instruction set as given at the back of your Sharp Owner's Manual. These groupings can be further 'herded' together under five headings to cover instructions which:

1. Transfer data from one place to another
2. Manipulate and test the data in some way
3. Re-route program running sequence
4. Handle input/output devices
5. System controls

Before we go into the commands, it may be useful to spend a few brief moments looking at the way a command is carried out by the Z80.

Every instruction is executed in three phases. In Phase 1, the instruction is fetched from the correct place in the program. The Program Counter tells the Z80 where to look (we dealt with this earlier). The first - perhaps only - byte of the instruction is then placed in a Register the Z80 keeps all to itself (called, believe it or not, the Instruction Register). In Phase 2, the instruction is decoded by the Z80 - that is, it sets up the cycle of operations for the third phase, which is to actually execute the instruction.

Each phase operates within finite steps, called clock cycles or T-States. The cycles themselves operate in 'machine cycles' - called 'M Cycles'. The shortest machine cycle lasts three clock cycles. Now as each cycle means a discrete unit of time, the more cycles an instruction needs for its fetching, decoding and execution, the longer it takes to execute. Pretty obvious really.

The point of all this is, generally speaking the more bytes there are to an instruction, the longer it takes to execute. However, the 'complexity' of the instruction also plays a part, so some instructions take longer than others of the same byte length. For example, the one-byte instruction to Decrement Register pair BC - DEC BC - takes 1 machine cycle, 6 T-States, while DEC A, also a one byte instruction, takes 1 machine cycle, 4 T-States. DEC A is faster by 2 T-States - or one miserable microsecond if the clock is 'running' at 2 MHz. or even less at 3.5MHz on the 700.

For the newcomer to machine coding, this discussion on machine cycles and T-States should be quite enough to cope with: it is beyond the scope of this book to discuss the actual speed of every instruction, since that becomes important only when one has gained experience. As mentioned before, most machine code programs run quite fast enough without any fine pruning.

The 'Brackets' Convention

Before we finally get down to the commands, there is one 'convention' you must be perfectly clear about - and that is the use of 'brackets' within a command.

An address can be referred to in two ways. If we want the address itself, it is written in the normal way - 1234H, for example. If we wish to refer to the CONTENTS of the address, then the address is placed in brackets.

Thus the command 'LD HL,1234H' means 'load Registers HL with the address 1234 hex'. You will recall from an earlier discussion that the Low byte goes into Register L (34 hex), and the High byte goes into Register H (12 hex).

The command 'LD HL,(1234H)', on the other hand, means 'go to address 1234 hex, and whatever byte you find there, put it in Register L. Then go to the next address - 1235 hex - and put the byte you find there into Register H'. (Look back a few pages to refresh your memory on how the Z80 requires addresses to be stored). So if addresses 1234H and 1235H hold bytes 89 hex and 67 hex respectively, then HL will be left holding the value 6789 hex after this command.

Similarly, take the command 'LD A,(HL)'. This means 'go to the address pointed to by Registers HL, and put the byte you find there into Register A'. If the HL Registers had been 'set up' to hold 1234H, then whatever byte is at that address (in our example above, it was 89 hex) is loaded into Register A. If HL Registers had been 'set up' to hold 6789H, as in the second example above, then whatever byte is at the address 6789H gets loaded into Register A.

Note that the command 'LD A,HL' cannot exist, since you will be trying to load two bytes of data into a one-byte store. Even a Sharp computer can't do that.

1. DATA TRANSFER COMMANDS

In this section, we will be looking at all the different ways you can shift one or more bytes of data from one place in memory to another - and that includes shifting data around the Registers themselves. For convenience, it also includes the 'creation' of new data - that is, loading a Register with a specific value rather than a value to be found elsewhere in RAM. What we won't include in this section are the commands which read or write to input or output devices.

You may think this an obvious point to make, but we'll make it nonetheless: data remains in an address or Register until it is 'overwritten'. Thus, if we say 'Load Register A from Register B (LD A,B) then both Registers A and B will be holding the data that was in B, and the data that was in A will be lost.

The 8-Bit Load Group

All 8-bit transfers are achieved by a straightforward load instruction which takes the following format:-

```
LD destination,source
```

Thus a typical example might be LD B,D - which means load the contents of Register D into Register B.

The following table shows the 8-bit load commands available:-

	Source of the load													
	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
Load Dest.														
A	x	x	x	x	x	x	x	x	x	x	x	x	x	x
B	x	x	x	x	x	x	x	x			x	x		x
C	x	x	x	x	x	x	x	x			x	x		x
D	x	x	x	x	x	x	x	x			x	x		x
E	x	x	x	x	x	x	x	x			x	x		x
H	x	x	x	x	x	x	x	x			x	x		x
L	x	x	x	x	x	x	x	x			x	x		x
(HL)	x	x	x	x	x	x	x							x
(BC)	x													
(DE)	x													
(IX+d)	x	x	x	x	x	x	x							x
(IY+d)	x	x	x	x	x	x	x							x
(nn)	x													

The Registers down the left hand side represent the DESTINATIONS of a load, and the Registers across the top represent the SOURCE of a load, in the command format 'LD destination,source'. The x's denote where a command is available.

So reading across the top line, you can have as valid commands: LD A,A; LD A,B; LD A,C; and so on. Notice that no command is available to load Register D from the address pointed to by Register pair BC (i.e. there's no LD D,(BE) command). Sad - but no problem.

In the Table, 'nn' means a two-byte number, which could represent an address. You'll notice that only Register A can be loaded from the contents of a specific address (top line - LD A,(nn)). Also, at the end of the Table, you'll see only Register A can be loaded into a specified address. Let's discuss the ramifications of this.

If you want to load a specific address with a data byte, you can either do it by first placing the data byte in Register A (if it isn't already there), then do a 'LD (nn),A' command (nn being the required address). Or - take a look at the horizontal line for '(HL)'. If HL is loaded with the desired address - i.e. LD HL,nn (we'll come to that command later on), then data from any of the Registers A,B,C,D,E and yes, even H and L can be loaded into the desired address - using the LD (HL), 'register' command.

If you study the Table, you'll see that the same applies 'in reverse' - that is, you can load any of the Registers (including H and L) from the address pointed to by the HL Registers (vertical column (HL)). Thus, you can write LD C,(HL) - meaning load Register C with the contents of the address pointed to by HL. Easy isn't it, when you know how.

Now let's look at another aspect of this Table - that 'n' column on the right hand side. As you've probably already guessed, 'n' stands for a data byte - any value from 0 to FF hex or 255 decimal. Notice, now, how you can load a specific byte of data into the address pointed to by HL - the LD (HL),n command.

Therefore to display letter Z in the top left corner of the screen:-

```
    LD  HL,0D000H;the top left corner address
    LD (HL),26    ;'26' = letter Z display code
```

MZ-700 Note

When directly addressing the Video Ram from S-Basic 700 users will, of course, have to ensure that the Video Ram is enabled first as was shown in chapter 1, programs 1 and 2. If S-Basic is not resident i.e. one is running a machine code program which does not use Basic, then V-Ram will not require enabling and the above example will work without any extra instructions.

The observant will have noticed that the same thing can be done by using (IX+d) or (IY+d) as the destination - for example LD IX,0D000H; LD (IX+0),26. You could load a 'Z' into the second top left screen address by changing the second command to LD (IX+1),26. The 'IX+d' commands have more bytes of instruction code, and take longer to process: they're more often used for data tables. You cannot write LD (IX),26 by the way. Your Assembler won't like it - it always looks for the displacement value, even if it's zero.

Yet another way could be to load the top left corner of the screen with a 'Z' - LD A,26 (or it can be written LD A,1AH); LD (0D000H),A. You pay your money and you take your choice.

You may wonder, looking at the table, how you can load for example the contents of Register D into an address pointed to by Register pair BC - that is, how do you cope without a command LD (BC),D. Well, good Register management, in the first place. But that isn't always feasible. So you'll have to transfer the data in D to A (having first 'saved' A somewhere, if you want to keep it), using LD A, D; then simply use LD (BC),A.

Four commands missing from the Table which were discussed earlier but will not be required for a while are:-

```
LD A,I  (load A from the Interrupt Register)
LD A,R  (load A from the Refresh Register)
LD I,A  (Load Interrupt Register from A)
LD R,A  (load Refresh Register from A)
```

The 16-bit Load group

The basic format for 16-bit (two-byte) data loads is essentially the same as that for 8-bit loads, namely:-

```
LD destination, source
```

There are however some important exceptions, which we will come to in a moment. Since we are talking about two-byte loads, either the source or the destination must, of course, be a Register pair.

The following Table shows the commands available within the format 'LD destination,source':-

Load Dest.	BC	DE	HL	SP	IX	IY	nn	(nn)
BC							x	x
DE							x	x
HL							x	x
SP			x		x	x	x	x
IX							x	x
IY							x	x
(nn)	x	x	x	x	x	x		

Source of the load

Doesn't look a very busy Table, does it? It would appear that you can't - as an example - directly load Register pair BC from the contents of, say, Register DE. Appearances are correct: there is no LD BC,DE command. But as we shall see, this isn't really a problem.

In the Table, 'nn' of course represents two bytes of data - which could be an address, or simply a number for some arithmetical operation - while '(nn)' represents the CONTENTS of address 'nn'.

Probably the most important things to notice about this table are the absence of the A Register in a pairing, and the fact that the

Stack Pointer Register, SP, can be loaded from the contents of Register pair HL, or the two-byte Registers IX or IY, or with an immediate address -'nn', or from the contents of a specific address - '(nn)'. So there are several ways to set up the Stack Pointer - or even to change it during a program (as long as you know what you're doing).

The reverse isn't true, however: as far as load - LD - commands are concerned, the SP address can only be loaded into '(nn)' - to save its value.

Now, what about the other ways we have to transfer two bytes of data, and what about the poor old A Register? What the Table could have shown is an extra column and an extra row headed (SP) - that is, for example, a LD (SP),BC command, or a LD BC,(SP) command. These functions are possible - but they are not invoked by this type of command.

Let's see what LD (SP),BC means. '(SP)' means the contents of the address 'named' in the Stack Pointer Register. That's the top of the Stack. So 'LD (SP),BC' means - 'put the contents of Register pair BC onto the Stack'. Similarly, 'LD BC,(SP)' means - 'load Register pair BC from the contents at the top of the Stack'. In both instances, the address held in the Stack Pointer Register is 'updated' after the transfer of each byte (see the earlier discussion on the Stack Pointer).

There is a command all of its own to put the contents of a Register pair on the Stack, and another command to take two bytes off. The commands are PUSH and POP, respectively.

These are the Register pairs and two-byte Registers you can PUSH and POP:-

AF,BC,DE,HL,IX,IY

Thus, to store the contents of Register pair DE on the Stack, you can write PUSH DE. And to get the data at the top of the Stack into DE, you can write POP DE.

You noticed, didn't you - Register pair AF can be PUSHed and POPed to and from the Stack. That's so you can conveniently put aside what may be important data in both or either the A Register and the Flag Register.

Now, what about that poser we set earlier - loading BC from DE, for example. How do we do that? There are two ways. One, you can PUSH DE, then POP BC - that puts DE's data on the Stack, then reads it off into BC. Method two - use the two single-byte load commands, LD B,D; LD C,E. Both methods work, both methods are exactly two instruction bytes long, both methods are used quite extensively. But, the PUSH and POP method makes the Z80 look 'beyond' itself and into RAM area to execute the commands - whereas the LD Register,Register method doesn't. So the LD Register,Register method is faster (by 16 T-States, as it happens). If you want to put the two byte data that's in one of the Index Registers IX or IY into a Register pair, then you have no option but to go via the Stack. Notice, though, you do not specify the 'displacement' with the Registers: it's PUSH IX, not PUSH IX+d.

There are some more commands that enable you to shift two bytes of data from one place to another. They are called 'Exchanges'. Here they are:-

```
EX (SP),HL
EX (SP),IX
EX (SP),IY
EX DE,HL
EX AF,AF'
EXX
```

An Exchange is different from a load, in that the contents of both places designated are 'swapped'. Thus, the first three commands swap the contents at the top of the Stack with the respective

Register named - HL,IX or IY. This makes possible some nice progamming techniques.

For example, when a subroutine is called (through a CALL command) the address of the next instruction after the CALL is put on the Stack. That's the address that will be put back into the Program Counter when a RETurn is made from the subroutine. But supposing we choose to put after the CALL command not the next instruction, but an item or items of data that we wish to pass into the subroutine. In the subroutine, we do an EX (SP),HL command. So now what was in HL is on the top of the Stack, and what was on the top of the Stack - the address of where our data is - is in HL. We can pick up the data now by doing, for example, a 'LD A,(HL)' command. Now - and this is important - we increment HL so that it points (or 'bumps') over the information byte(s) to the address of the next instruction, and then do another EX (SP),HL. The correct address for the next instruction when we RETurn is now in the right place ready to be picked up by the Program Counter, and we've passed data into the subroutine for processing. That's by no means the only way to pass data into a subroutine, but it is a useful way.

The EX DE,HL command is invaluable when doing arithmetical operations, or when you want to exchange a DEstination address in DE and a source address in HL.

The EXX command exchanges the contents of the three Register pairs BC,DE and HL with their counterparts in the second Register set - BC', DE' and HL'. But not, you'll notice, the AF Registers - they have their own command EX AF,AF'. The information contained in the second Register set is not worked on, merely 'held in abeyance', so you have another way of temporarily holding onto data without setting up storage addresses or using the Stack. However, you'll find in some computers, the second set is used quite extensively to handle interrupt routines and so on, so if you unwittingly wipe out or leave 'strange' data in the second set, you could have some peculiar things happening.

The Block Transfer Group

We now come to the commands which enable any number of data bytes to be transferred from one place in RAM memory to another. These commands and their functions are:-

```
LDI  - Load (DE) from (HL)
       Increment DE and HL
       Decrement BC

LDIR - Load (DE) from (HL)
       Increment DE and HL
       Decrement BC
       Repeat until BC = 0

LDD  - Load (DE) from (HL)
       Decrement DE and HL
       Decrement BC

LDDR - Load (DE) from (HL)
       Decrement DE and HL
       Decrement BC
       Repeat until BC = 0
```

All of these commands transfer the data byte found at the address pointed to by the Register pair HL, to the address pointed to by the Register pair DE. After each data transfer the value held in Register pair BC is decremented. (Obviously, these three Register pairs must therefore be 'primed' before the block transfer command is invoked).

In the case of the LDI and LDIR commands, DE and HL are incremented after each transfer, while for the LDD and LDDR commands they are decremented after each transfer. Thus HL and DE are always left pointing to the correct addresses for the next data byte transfer.

With the LDIR and LDDR commands, the transfer of data continues until BC becomes zero, at which point processing continues with the next command.

With the LDI and LDD commands, processing continues with the next command after each transfer: this enables other actions to be taken before the next transfer of data - though you must remember not to 'upset' the values in the DE,HL, or BC Registers (unless that is all part of your cunning program). The LDI and LDD commands set the P/V Flag to zero if they decrement BC to zero. The following program will transfer only those data bytes that have their most significant bit (Bit 7) 'set' - that is, equal to '1': the program assumes that DE and HL have been set up with the Destination and Source 'start' addresses, and that BC is set to count the maximum number of bytes to be examined, and transferred if Bit 7 is equal to '1'.

```
NEXT:LD  A,(HL)    ;Get 'next' byte
     BIT 7,A       ;Test top bit
     JR NZ,MOVE    ;Byte wanted - shift it
     INC   HL      ;Byte unwanted - increment HL
     DEC   BC      ; and decrement the counter BC
TEST:LD  A,B       ;Check if BC is zero
     OR    C       ;by ORing B with C
     JR NZ,NEXT    ;Do it again if BC not zero
     JR    DONE    ;BC is zero - so finish
MOVE:LDI           ;Move the byte
     JP PE,NEXT    ;Do again if BC not zero
DONE:Your next command...
```

Instead of the 'JP PE,NEXT' command after the LDI, one could do a relative jump back to the 'TEST' point - JR TEST - which checks if BC has reached zero after being decremented. But we wanted to demonstrate the use of the JP PE command. Note, incidentally, one cannot do a Relative Jump (JR Label) when testing for parity. But more about this, and the other commands 'BIT 7,A',INC and DEC later.

You may ask why do we need both LDIR and LDDR commands. It is so that we never 'overwrite' data we want to shift.

Suppose for example we want to shift a data block of 1001H bytes from 8000H to 8500H. If we use the LDIR command with HL pointing to 8000H and DE pointing to 8500H, the first byte will be transferred from 8000 to 8500H - overwriting data within the block of 1001H bytes we're going to transfer.

In this instance, we would use the LDDR command - and set the HL Register to point to the END of the block we wish to shift (i.e. 9000H), and DE to the END of the destination area (i.e. 9500H). So now, by the time DE has been decremented to 9000H, we've already shifted the data from there, so it's o.k. to overwrite it.

2. DATA MANIPULATION & TEST COMMANDS

The 8-Bit Arithmetic and Logic Group

The simplest arithmetical operation that can be done on a single byte is to add one to it (INC) or deduct one from it (DEC). These operations can be performed on the following Registers and addresses pointed to by Registers:-

A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

The Z, P/V and S Flags are affected as a result of the operation.

The rest of the operations in this section ALL operate on Register A: the OTHER data byte source - even if that is Register A as well, must be specified. The following sources can be used for the 'other' data byte:-

A, B, C, D, H, L, (HL), (IX+d), (IY+d), n

The 'n' of course represents a specific value.

The commands available are:-

ADD A; ADC A; SUB; SBC; AND; OR; XOR; CP

We will examine each command:-

ADD A (examples - ADD A,B; ADD A,(HL); ADD A,2)
Note the A Register must be specified. This command simply adds the specified data byte to that in Register A. Thus ADD A,(HL) means add the contents of the address pointed to by HL to the contents of the A Register, leaving the result in the A Register. If the result exceeds FF hex (255 decimal), the Carry Flag is set, and A holds the result minus 256. Thus, with FF hex in Register A 'ADD A,2' would result in A holding '1', and the Carry Flag set to '1'.

The Z, P/V and S Flags are also affected according to the result of the ADD operation.

ADC A (examples - ADC A,B; ADC A,(HL); ADC A,2)
This is exactly the same as the ADD command, except that the contents of the Carry Register before the operation commences are also added to Register A. Thus if the Carry Flag is set and Register A holds 21 hex, 'ADC A,2' results in A holding 24 hex, and, because the operation did not require a 'carry', the Carry Flag would be reset to zero.

SUB (examples - SUB B; SUB,(HL); SUB 2)
Note that Register A is not specified (unless one wants to SUB A, i.e. subtract the contents of A from A). This command subtracts the specified data from Register A, and leaves the result in Register A. As with 'ADD', the Flags are affected according to the result.

SBC (examples - SBC B; SBC (HL); SBC 2)
Similar to the SUB command, except that the contents of the Carry Flag are also subtracted from Register A.

AND (examples - AND A; AND (HL); AND 0FH)
This performs a logic AND function between the A Register and the specified data byte, leaving the result in Register A.

'ANDing' means 'compare the two bytes, bit by bit. If both bits are a 1, then the corresponding bit of the result will be a '1'. Otherwise it's '0' '.

Thus, with 0A7H in Register A, 'AND 0FH' produces:-

```
                  10100111  (A7 hex, 167 decimal)
                  00001111  (0F hex, 15 decimal)
         Result = 00000111  (7)
```

This technique is often used to provide a 'mask' - that is, to eliminate parts of a byte that are not wanted. The 'masking' data - in the above example '0FH' - covers that part of the data byte we want to keep.

ANDing always resets the Carry Flag to zero. Thus AND A will reset the Carry Flag to zero, and leave Register A as it was before the operation: this command can therefore be used to clear the Carry Flag without upsetting Register A.

OR (examples - OR A, OR (HL), OR 80H)
This performs a logic OR function on the A Register, leaving the result in the A Register.

'ORing' means 'test the two data bytes, bit by bit. If either or both bits are a '1', then the corresponding bit in the result will be a '1'. Otherwise it's a '0' '

Thus with 1B hex in Register A, OR 80H produces:-

```
                 00011011  (1BH, 27 decimal)
                 10000000  (80H, 128 decimal)
        Result = 10011011  (9BH, 155 decimal)
```

This can be a useful way to add in bits to a byte: if A for example holds a value between 0 and 9, OR 20H will leave in A the Sharp display code for that number. OR 30H will leave in A the ASCII code for that number.

OR always clears the Carry Flag, and affects the other Flags according to the result. Thus, OR A leaves Register A unchanged, but clears the Carry Flag.

<u>XOR</u> (examples - XOR A, XOR (HL), XOR 0FH)
This performs a logic XOR function on the A Register, leaving the result in the A Register.

'XORing' means 'compare the two data bytes bit by bit. If one is a '1' and the other is a '0', then the corresponding bit of the result will be set to a '1'. Otherwise it will be '0' '. Thus if Register A holds 14H, then XOR 17H produces:-

```
                 00010100  (14H, 20 decimal)
                 00010111  (17H, 23 decimal)
        Result = 00000011  (3)
```

XOR always resets the Carry Flag, and affects the other Flags according to the result. XOR A must always result in Register A becoming zero - thus this is a useful command to clear Register A and the Carry Flag to zero: the Zero Flag will be set to '1' - meaning the value of Register A is zero.

CP (examples - CP B, CP (HL), CP 9)
This subtracts the specified data byte from the value held in Register A - AND DISCARDS THE RESULT: thus, only the Flags are affected by the command.

If the Test byte is greater than that in Register A, then the Carry Flag will be set.

If the test byte is the same as that in Register A, then the Zero Flag will be set.

If the test byte is equal to or less than that in Register A, then the Carry Flag is reset.

The Sign Flag and the P/V Flags will be set or reset according to the value in Register A.

The 16-Bit Arithmetic & Logic Group

As with the 8-bit Group, the simplest commands in this Group are INC and DEC. These commands can be used to increment or decrement Register pairs:-

    BC, DE, HL

and the 16-bit Registers:-

    SP, IX, IY

Note however that, unlike the 8-bit INC and DEC, for the 16-bit versions, the Flags are completely unaffected.

The following Table shows the ADD, ADC and SBC commands available (indicated by the x's):-

	This pair	BC	DE	HL	SP	IX	IY
ADD	HL	x	x	x	x		
ADD	IX	x	x		x	x	
ADD	IY	x	x		x		x
ADC	HL	x	x	x	x		
SBC	HL	x	x	x	x		

Note that the SUB command is not available - the Carry Flag is always involved on a subtract operation. If you don't want the Carry Flag involved - in case it may be set to '1', use an OR A command first to clear it.

The ADD, ADC and SBC functions are the same as those for the 8-bit commands except, of course, here they are operating on 16-bits.

The 8-Bit Shifts and Rotates

These commands operate on a specified byte of information, shifting or rotating its contents 'to the left' or 'to the right'.

The byte operated on can be in:-

A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

The commands available are as follows:-

RLC (Examples - RLC B; RLC (HL))
This moves the contents of bit 0 to bit 1, bit 1 to bit 2 and so on. Bit 7 is moved into the Carry Flag AND into bit 0. The data is thus ROTATED Left, with the Carry Flag reflecting Bit 7. Note, for Register A the command can be written RLC A or RLCA: RLCA is a different command, requiring one less instruction byte.

<u>RRC</u> (examples - RRC B; RRC (HL))
This moves the contents of bit 7 to bit 6, bit 6 to bit 5 and so on. The contents of bit 0 are moved into the Carry Flag AND bit 7. The data is thus ROTATED Right, with the Carry Flag reflecting bit 0. Note for Register A, the command can be written RRC A or RRCA: RRCA is the shorter, faster version of the two.

<u>RL</u> (examples - RL B; RL (HL))
This moves the contents of bit 0 to bit 1, bit 1 to bit 2 and so on. Bit 7 is moved into the Carry Flag, and the Carry Flag contents are moved into bit 0. Thus nine bits are involved in a ROTATE Left. Note that for the A Register this command can be written RLA instead of RL A, RLA being a shorter, faster command.

<u>RR</u> (examples RR B; RR (HL))
This moves the contents of the Carry Flag into bit 7, bit 7 into bit 6 and so on. Bit 0 is moved into the Carry Flag. Thus nine bits are involved in a ROTATE Right. For the A Register, the command can be written RRA instead of RR A, RRA being the shorter and faster of the two commands.

<u>SLA</u> (examples - SLA B; SLA (HL))
This moves bit 0 into bit 1, bit 1 into bit 2, and so on. Bit 7 is moved into the Carry Flag. A '0' is placed in bit 0. Thus the data is SHIFTED left.

<u>SRA</u> (examples - SRA B; SRA (HL))
This moves bit 7 into bit 6, bit 6 into bit 5 and so on. Bit 0 is moved into the Carry Flag. Bit 7 is 'refilled' with its original value (this is for 'signed' arithmetic' operations, to preserve the sign bit 7). Thus the data is SHIFTED right, arithmetically.

SRL (examples - SRL B; SRL (HL))
This moves bit 7 to bit 6, bit 6 to bit 5 and so on. Bit 0 is moved into the Carry Flag, and a '0' is placed in bit 7. Thus the data is SHIFTED right.

Decimal Arithmetic Rotates

We now come to two very special rotate functions, used when handling Binary Coded Decimal Arithmetic. Both commands operate between Register A, and the data byte in the address pointed to by the Register pair HL (i.e. '(HL)'). They are:-

RLD
This command puts the bottom nibble (lower four bytes) of the A Register into the bottom nibble of (HL), the bottom nibble of (HL) into the top nibble of (HL), and the top nibble of (HL) into the lower nibble of Register A. The nibbles are thus rotated. The top nibble of Register A is unaffected by the operation.

RRD
This does the same as RLD, but in the other direction. Thus, the bottom nibble of Register A is moved to the top nibble of (HL), the top nibble of (HL) is moved to the bottom nibble of (HL) and the bottom nibble of (HL) is moved to the bottom nibble of Register A. The top nibble of Register A is unaffected by the operation.

BIT MANIPULATION

Quite often, one wants to test a specific bit in a data byte, to see whether it's a '1' or a '0'. Equally it can be very useful to be able to set a specific bit to a '1', or reset it to '0'. The Z80 allows you to do this.

The three basic command words available are:-

```
BIT b,l: Test bit 'b' at location 'l'
SET b,l: Set bit 'b' at location 'l' to a '1'
RES b,l: Reset bit 'b' at location 'l' to a '0'
```

The bit 'b' can, of course, be any bit from 0 to 7. (Remember that bit 7 is the most significant, and bit 0 is the least significant).

The location 'l' can be any of the following:-

```
A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)
```

Thus there are three basic commands, each of which can operate on one of eight bits in ten different locations - a total of 240 commands in all. Typical examples of the three basic commands are now given.

BIT 3,B

This tests whether bit 3 of Register B is a '0' or a '1'. If it is a '0', the Zero Flag is set to a '1' so that a subsequent test for Zero would succeed. Thus, in this program segment:-

```
BIT 3,B
JP Z,WASZERO
```

a JumP will be made to the program segment labelled 'WASZERO' if BIT 3 of Register B is '0'. Otherwise, processing continues with the next command.

Note that whilst the Zero Flag is specifically set or reset by BIT commands, the Sign Flag 'S' and the Parity/Overflow Flag 'P/V' may or may not be affected - the information they contain is irrelevant and untestable. The Carry Flag is unaffected by the operation - it will contain a previously held value.

SET 7,(HL)

This command makes bit 7 of the data byte at the address pointed to by the HL Register pair equal to a '1'.

RES 5,(IX+3)

This command operates on the data byte at the address pointed to by the IX Register PLUS 3, resetting its bit 5 to a '0'. Thus if the IX Register holds '8000H', then the data byte at address '8003H' will have its bit 5 turned into a '0'.

These bit manipulation functions can prove invaluable in some types of program. To give just one broad example, in an Adventure game one data byte may be used to indicate the possible exits from a given location - a '0' meaning 'no exit', and a '1' meaning 'exit possible'. Bit 7 could represent North, bit 6 East and so on, with four bits 'left over' to represent say 'up', 'down' and two other possible ways out. Checking whether or not an exit is possible is then simply a matter of testing the appropriate bit: changing the status of an exit is simply a matter of 'SETting or RESetting it.

SPECIAL A and F REGISTER MANIPULATIONS

There are five instructions which operate specifically on Register A or on the Carry Flag in Register F. These are as follows:-

DAA

This is a very special command for use when performing Binary Coded Decimal arithmetic (BCD). In BCD, a four-bit nibble is used to store one decimal digit: thus one byte can store two decimal digits (this is referred to as 'packed BCD'). The values '11' to '15' decimal can all be represented within one nibble: however, for BCD we only want one decimal digit per nibble; and so the binary representations of '11' to '15' decimal are meaningless and not wanted.

Let us look at two examples. First, we will add '22' decimal to '43' decimal. The program to do this in Binary Coded decimal could be:-

```
LD  A,22H;22H = 0010 0010 binary,'22' in BCD
ADD A,43H;43H = 0100 0011 binary,'43' in BCD
```

As you can see, adding the binary values would yield 0110 0101 - which in BCD is '65'. Just what we wanted, so there's no problem. Now let us look at what happens if we add '26' decimal to '17' decimal. Using the program segment as before, the binary representation for this would be:-

```
0010 0110  (26H)
0001 0111  (17H)
```

and if we add these, we get

```
0011 1101  (3DH)
```

Here, the 'D' is meaningless as a decimal number. And that, patient reader, is where the DAA command comes in. Added after the 'ADD A' instruction in the program above, it Decimal Adjusts any result in the A Register. Thus, in the first example, the 'DAA' command would do nothing, for all is fine and dandy. But in the second example, it would see that things have gone wrong with the lower nibble, sort out exactly what had gone wrong (depending on whether we'd been adding or subtracting), and adjust the result accordingly. In the second example, it would leave Register A holding 0100 0011 - '43H' or 43 in BCD - which is correct. In this specific instance it achieves this result by adding a further 6 to the lower nibble, but don't worry about that. Sufficient to know that it makes the correct adjustment.

What you should know, however, is that to sort things out the DAA command makes use of the Flags - so after a DAA command, all the Flags are affected in some way.

<u>CPL</u>
This command 'complements' whatever value is held in the A Register: that is, every '0' becomes a '1', and every '1' becomes a '0'. Thus, if the A Register held the binary value '00101100', after a CPL command it would hold the binary value '11010011'.

This is called the 'one's complement' of the number, and is a way of representing positive and negative values. For example, a '5' in binary is represented by '00000101'. On the other hand '-5' can be represented by the 'one's complement', namely '11111010'. Notice that bit 7 is now '1' - representing a minus value. (See also the discussion on Flags).

The 'testable Flags are not affected.

<u>NEG</u>
In this command, the contents of Register A are subtracted from zero, and the resulting value is stored back in Register A. This is called the 'two's complement' of the number.

In two's complement representation, positive values are represented just as in 'one's complement' - i.e. in the usual signed binary way, with bit 7 showing the sign (0=positive,1=negative). Negative numbers however are represented as the 'one's complement' value PLUS one. Thus the two's complement of '-5' is '11111011'.

Why go to all this bother? Two's complement makes signed arithmetic easier for the computer to handle. Consider the sum '3 minus 5'.

```
        00000011 (+3)
        11111011 (-5)
```

Adding these (since we are representing the 'minus' as -5 in two's complement), we get:-

```
        11111110
```

Here, bit 7 tells us the answer is negative. Taking the two's complement of 1111110, therefore, we get 00000010 (two's complement, remember, is the one's complement of 1111110, which is 0000001, plus 1). Thus, the value is '2', and the Sign is negative. Answer, -2. Just what the doctor ordered.

The Z80 command NEG, then, obtains the two's complement of a value in Register A and leaves it in Register A, thus saving the bother of doing a one's complement (CPL) and adding 1 (ADD A,1). This is a very scant description of the principles behind one's and two's complement arithmetic, but it should be enough to give the newcomer to machine coding an idea of what it's all about.

Note that all the Flags may be affected by NEG command.

CCF

This command 'complements' the Carry Flag in the F Register. If the Carry Flag is '0', then CCF makes it a '1'. If the Carry Flag is '1', CCF makes it '0'.

SCF

This command makes the Carry Flag equal to a '1' (i.e. 'Set Carry Flag').

There isn't a command to 'reset' the Carry Flag - that is, to clear it. However, as mentioned before, AND A and OR A will do this, without affecting anything else. XOR A clears the Carry Flag as well, but also clears Register A - makes it '0' - and consequently also sets the Zero Flag and possibly affects the Sign Flag (which reflects bit 7, remember). Observant readers might see that an alternative way to clear the Carry Flag would be to set it first (SCF), then complement it (CCF) - but this takes two bytes of instruction code, whereas OR A takes one. So it's not much good as an alternative. But well spotted anyway.

BLOCK COMPARISONS

The last 'manipulation and test' commands to be examined are the 'block comparisons'. In many respects these are similar to the 'block transfer' commands discussed earlier. They enable a whole chunk of data to be 'searched' to find a byte that is the same as that in Register A. Like the block transfer commands, they need you to set up the Registers first: HL with the start address of the area to be searched, BC with the number of bytes to be searched, and A with the data byte we're looking for. The commands are:-

```
CPI   Increment HL
      Decrement BC

CPD   Decrement HL
      Decrement BC

CPIR  Increment HL
      Decrement BC
      Continue until BC=0 or A=(HL)

CPDR  Decrement HL
      Decrement BC
      Continue until BC=0 or A=(HL)
```

As with the block transfers, the CPI and CPD commands enable other operations to be undertaken within the 'search loop'. When a match is found, the Zero Flag is set. When BC reaches zero, the P/V Flag becomes 0 (Reset).

The CPIR and CPDR commands whiz through the block to be searched until BC reaches zero, or a match is found.

When a match is found, of course, Register pair HL will be pointing to the matching byte in the data block.

3. RE-ROUTING PROGRAM RUNNING SEQUENCE

We now come to the commands which let you change the 'batting order' of your program instructions - the commands which emulate the 'GOTO's' and 'GOSUB'S' in BASIC, and of course 'RETURN'. In machine coding, however, you have more scope.

Jumps and Relative Jumps

The BASIC 'GOTO' instruction can be emulated by a JumP (JP) or a Relative Jump (JR). A straight Jump is like a straight GOTO. The format is:-

```
          JP Label  or  JP address
```

'Label' of course representing the label you have given at a particular point in your Assembly Language program, or which has been defined by an EQUate.

Jumps can also be conditional - that is, any of the Flags can be tested, and the Jump made if the test succeeds. The format for this is:-

```
          JP cc,Label   or   JP cc,address
```

where cc represents any of the Flag conditions that can be tested (e.g. NZ,Z,NC,C,PO,PE,P,M - see the section on 'Flags'). Thus a typical instruction might be JP NZ,ENDGAME, which means 'if the Zero flag is not set (non zero condition) - as a result of a previous operation - then continue processing from the address labelled ENDGAME'.

Relative jumps need a little explaining. Their instruction codes are shorter than straight jumps. The address they provide a jump to is relative to the current address, and is given by a displacement value: consequently the actual address doesn't figure in the instruction code itself. If none of the addresses within the routine itself are 'mentioned' directly, the routine can be located anywhere in memory. It is thus called a 'rellocatable' routine. Many programmers write small subroutines (to do specific functions) in a rellocatable form, so that they can add the routines to any major program they are preparing. All they need then is the 'start' point of the routine - which is done by a label.

The format for a relative jump is:-

```
        JR Label   or  JR sc,Label
```

where 'sc' represents a conditional test. Unlike Jumps, which can test any of the Flags, only the Zero and Carry Flags can be tested for a conditional relative jump - i.e. Z, NZ, C or NC. So you cannot write, for example 'JR M,LABEL'.

The relative jump can be made forwards or backwards. The displacement value is in two's complement, and is added to the Program Counter plus 2. If you work it out, you'll find that relative jumps can be made to addresses within -126 and +129 bytes of the address of the first byte of the 'JR' instruction: fortunately, the Assembler calculates the displacement value for you when generating the machine code.

Special Jumps

There are four more kinds of jump you can do in machine coding. Three of these enable you to jump to an address specified in the Registers. They are:-

```
        JP (HL)
        JP (IX)
        JP (IY)
```

and they're extremely useful when using 'jump tables'. One could for example have a data table of items, each item being three bytes long. The first byte of each item would be the 'menu selector'. The next two bytes would be the address (in the order Low byte, High byte, remember) of the 'action' routine for that menu item. The 'menu selectors' through the table are searched (jumping over the next two bytes of the item where no match is found) until a match is found.

With HL pointing to the matching byte, it is then a simple matter to: INC HL (so it points to the Low Byte of the action address); LD E,(HL) - pick up the low byte in E; INC HL - point to the High byte of the action address); LD D,(HL) - pick it up; EX DE,HL - put the address into HL; JP (HL) - and go.

This procedure is just one of the many, many ways in which one can pick up the address of a required routine. It's also a fairly crude way, but it demonstrates a point.

The fourth kind of jump emulates to some extent the 'FOR-NEXT' loop in BASIC. It is a type of Relative Jump, and has the format:-

```
          DJNZ Label
```

For this instruction Register B is used as a counter, so you must set it up with a value equal to the number of times you want the operation done. At the beginning of the 'loop', you have a Label. When the DJNZ command is met, Register B is decremented and, if it is not zero as a result, a jump is made to the Label address. It is a Relative Jump, so the Label address must be within -126 and +129 bytes of the DJNZ instruction's address (the Assembler calculates the displacement for you).

You can jump out of the loop at any time - if a subsidiary test succeeds, perhaps. Register B will then be holding the number of operations left to do when the test succeeded - which may be useful information.

Calls

A 'CALL' command is just like 'GOSUB' in BASIC. Like the JP jump command, it can be unconditional:-

```
          CALL Label    or    CALL address
```

or conditional:-

```
          CALL cc,Label    or    CALL cc,address
```

the 'cc' representing one of the Flag tests, just as for the conditional Jump command.

When a CALL command is met, the Program Counter address for the next command is put on the Stack, ready for when a RETurn is made - we discussed this when reviewing the Registers of the Z80. You must therefore ensure that the Stack still has the RETurn address 'on top' when the RETurn is made (it's utter disaster if you don't).

Restore

There is another kind of special Call command, called RST - which stands for ReSTore. The format is:-

```
          RST a
```

where 'a' stands for one of the following:-

```
          00H, 08H, 10H, 18H, 20H, 28H, 30H or 38H.
```

When the RST command is encountered, the Program Counter address is put on the Stack (just as in a CALL command), and a jump is made to the specified address. The point about this instruction is that it is only one byte long, and provides an extremely fast jump.

You'll notice though that all the addresses concerned lie within the monitor (on Sharp machines). So, for example, RST 00H gives you a cold start - like pressing 'reset'. Only two of the other addresses are significant in Sharp monitors. One is 30H - which provides a jump to the Music playing routines (the string to be played must be in an area pointed to by Registers DE, and terminated by an 0DH byte). The other is 38H - the Interrupt routine vector. Any other 'RST' will throw your machine haywire - since you'll be calling the 'middle' of an instruction.

Returns

These RETurn control from a subroutine, just like 'RETURN' in BASIC. The format is:-

```
          RET     or     RET cc
```

where 'cc' is one of the Flag tests, as for the jump (JP) and CALL commands.

There are two special Return commands. The first is RETI (return from an interrupt), which must always be preceded by an EI (Enable Interrupt) command. The second is RETN, which provides a return from a non-maskable interrupt, and resets the Z80's interrupt Flag to the condition it held before the non-maskable interrupt was made.

4. INPUT/OUTPUT COMMANDS

There are a number of commands available for inputs from or outputs to peripheral devices. In many ways most of these are like the block transfer commands, in that they enable blocks of data to be transmitted either automatically or within a 'loop' performing other functions. These particular commands are:-

Input commands	Output commands
INI	OUTI
INIR	OTIR
IND	OUTD
INDR	OTDR

For the input commands, the peripheral device addressed by Register C is 'read', and the information is loaded into the address pointed to by Register pair HL. Then Register B is decremented, and Register pair HL incremented (INI, INIR) or decremented (IND, INDR).

For the Output commands, the procedure is reversed - that is, the contents of the address pointed to by HL is output to the peripheral device addressed by Register C, B being decremented and HL incremented or decremented after each transfer.

For the input or output commands ending with 'R', the procedure continues apace until B = 0.

Four other input and output commands are available. These are:-

Input commands	Output commands
IN A,(p)	OUT (p),A
IN r,(C)	OUT (C),r

IN A,(p) loads Register A with a byte of data read from the peripheral Port 'p'. Similarly, OUT (p),A outputs the data byte in A to the port 'p'.

IN r,(C) and OUT (C),r do the same kind of thing, except the port device is addressed by the C Register, and the specified Register 'r' can be any of:-

A, B, C, D, E, H, L

MZ700 users will be familiar with the OUT (p),A command - which is used for 'bank switching' the different RAM and ROM areas: OUT (0E0H),A for example, switches in a block of RAM in place of the Monitor ROM at addresses 0-0FFFH. In this instance the data in Register A is irrelevant.

5. SYSTEM CONTROLS

These commands are used for controlling the Z80 'system':-

NOP
This means, quite simply, No OPeration. That is, do nothing. Carry on with the next command you find. It's useful when writing programs in Assembly language, to provide a suitable spot for a 'Breakpoint'. Since it takes time to 'execute', it can also be used to provide a very short (a very, very short) delay - 2 usec on a 2MHz clock.

HALT
This shuts down the operation of the Z80 completely, until an interrupt is received, or a 'reset' performed.

DI,EI
These Disable or Enable the Interrupt procedures. Interrupts are discussed in the section on the Z80 Registers.

IM 0,1 or 2
The IM commands set the Z80 in a particular Interrupt Mode. See the discussion on Interrupts in the section on Z80 Registers.

NON Z80 COMMANDS (Pseudo Ops)

If using an Assembler, you'll find other commands are available which are essential for writing in Assembly Language. These are used by the Assembler to tell it what to do - reserve data space, assemble at a specific address and so on. They do not 'translate' into Z80 instruction codes, and will not normally appear in a dissassembled listing. Please refer to the manual for your Assembler for details of these commands.

3

Assembling

This chapter will deal with getting started on writing your own machine code programs using an Assembler/editor program such as ZEN which is widely available for the Sharp computers. Any differences on entering programs between ZEN and other assemblers should be minimal as the principles are the same. If you already know the methods of entering lines into an assembler then some of this chapter obviously could be skipped, as we will start from loading the assembler and describe some of the errors which can too easily be made by first time users. The first program we will enter simply prints the alphabet along one screen line, which is not very exciting, but it is nice and short and will demonstrate how lines are entered.

The Monitor section of memory (addresses between 0000 and 0FFF hex) within the Sharp contains several routines for what are simply termed as housekeeping jobs. These routines take care of tasks such as printing a character on screen, printing a new line, accessing the clock, reading a program from tape, verifying and saving of programs etc., and obviously they are made full use of when running any program, Basic or machine code, as it is far simpler to form a message to be printed from within your program and then simply call the monitor routine to get that message printed on the screen than writing a routine in your program to do the same job.

The Monitors of the MZ-80A and MZ-700 are listed in their respective manuals that are supplied with the machines, unfortunately the MZ-80K manual does not list the Monitor at all but that will not affect 'K' owners too much as each time we access a monitor routine it will be explained. A published Monitor listing of the MZ-80K should be available through Sharp dealers

ZEN loads directly from the Monitor, so as soon as the MZ has been switched on place the cassette in the computer and enter 'LOAD' (or simply 'L' on the MZ-80A and 700) followed by the 'CR' key and load the program normally. On completion the screen will display:-
ZEN >
Enter exactly, spaces included, all entries under the TO ENTER column (NOT THE DISPLAYED COLUMN) followed by a carriage return at the end of each line. There is an error in the program which has been entered deliberately and we will alter it later. Remember any calls or jumps to addresses between 0000H and 0FFFH are to routines within the monitor ROM section, and their functions will be shown.

```
                 TO
DISPLAYED        ENTER

ZEN >            E
   1             LOOP:EQU 1203H
   2             START:CALL 0006H
   3             LD A,"A"
   4             NEXT:CALL0012H
   5             INC A
   6             CP "Z"+1
   7             JR NZ,NEXT
   8             CALL 06H
   9             JP LOOP
  10             END
  11             .
ZEN >
```

At the end of a program one must enter 'END' on a separate line, and to cease entering and move back to command level a full stop must be entered on a separate line too.
Now we will analyse what we have entered.

Line 1 of the program was an equate line and this simply tells the assembler that the symbol 'LOOP' equates to 1203H which is the address we wish to jump to at the end of the program as one can see in line 9 we have entered JP LOOP, we don't need to specify an address to jump to as the assembler has noted which address LOOP equals. One reason for these equates is that if we wished to alter the address at some future stage we would not need to list the whole program and alter each line which contained this address, all that is required is to change the first line to the different address and the assembler will do the work for us. This address is the warm start entry point to the ZEN Assembler, when this short program finishes running we need to tell the computer where to jump to and the mainloop of ZEN seems to be as good a place at this stage, we don't want the program running off wildly into memory. A colon must be entered between the symbol and the letters EQU.

Line 2 contains a label 'START' this is where, when testing the program, we will execute from. Any line can have a label, for in this instance when testing we shall simply enter 'GSTART' which means goto the label start. This line calls a monitor routine at address 0006H which simply moves the cursor to the next line on screen and when that task is completed control returns to our program. This is similar to a GOSUB in basic but in this case the subroutine is already in ROM memory within the Monitor and all our program needs to do is call it.

Line 3 loads the A register with the value of the letter 'A'. ZEN is quite versatile in that it allows entries within quotes and it simply converts this to the Hex equivalent value of the letter, in fact this line would have the same meaning if we entered LD A,41H which is how it would be assembled and loaded into memory by ZEN anyway. 41Hex is the hexadecimal ASCII code for the letter 'A', or we could have entered LD A,65 which is the decimal ASCII value of the letter 'A' and so omitting the suffix H which signifies to ZEN that the value is decimal and ZEN must convert it to Hex.

Line 4 contains the label NEXT as we will jump back here to continue printing letters. It is followed after the colon by Call 0012H which once again is a subroutine in ROM Monitor which prints the ASCII value currently stored in register A, and returns to our program.

Line 5 increments register A so the first time round after printing A on the screen we want it to increase its value by 1, so it will increase from 41H to 42H, the letter 'B'.

Line 6 compares the value of register A to see if it has reached Z + 1, and if it hasn't line 7 tests and jumps back to NEXT to do it all again. Once again it is easier to enter line 6 as "Z"+1 but when it is assembled this will be automatically altered to the Hex value of Z plus 1 making 5Bhex.

Line 7 is the relative jump and here one can see the advantage of giving lines a label for one does not need to calculate the number of bytes to jump back, as we did in chapter 1, as the assembler does it for us. Furthermore one could add extra lines between 4 and 7 which will obviously alter the amount of bytes to jump back over without the need to adjust anything else as the assembler will adjust the relative jump automatically.

Line 8 again makes a call to 0006H to print a newline. Note that here we entered it as CALL 06H instead of 0006H, this was done to show that although it is sensible to enter the complete address if the address contains leading zeroes they do not have to be entered. Another rule when entering addresses is if it was CALL FF00H then ZEN could confuse the address for a label as it does not begin with a number, therefore if any address which commences with an alphabetical character needs to be entered it MUST be preceeded with a zero, in the hypothetical case of FF00H the correct entry should be CALL 0FF00H.

Line 9 puts us back under the control of ZEN when the program finishes.

The next task is to find out if we have entered the program correctly, some bright sparks may have noticed some errors already, as one will get errors when entering and it is better to discover some of the more common types of error messages at this early stage. Enter 'A' and 'CR', this tells ZEN we wish to assemble the program.

The screen will prompt for an 'OPTION' which will determine if we wish to assemble to a printer, by entering 'E' for external, or by entering 'V' for video to print on screen the assembled version, or if we enter the 'CR' key on its own it will be assembled internally only stopping at a line which contains any errors, which is the fastest option. So after the 'OPTION' prompt enter 'CR'.
The screen will display:-

```
ORG !
   2 START:CALL 0006H
ZEN >
```

which simply means we did not enter the origin of the program, which is where in memory we want it to reside. This is obviously a major omission as the assembler must know where to place the program.
Enter 'T' followed by 'CR' and the first line of the program will be displayed. 'T' is the target line you wish to be displayed, entering 'T4' would display line 4, whereas just entering 'T' on its own moves up to the first line.
Entering 'E', as we did to begin entering the program, will let us enter extra program lines from the current line, which after entering 'T' will be line 1, and as we enter these extra lines all the lines already in the program will simply shift up a line, the existing line 1 will remain intact but will now become line 2 etc.
We should also enter a line to determine where we wish the program to load into memory once it is assembled, this does not need to be the same address as the ORG address, but to keep this program as simple as we can we will load in the same place.

DISPLAYED	TO ENTER
ZEN >	E
1	ORG 8000H
2	LOAD 8000H
3	.
ZEN >	

Note the full stop to bring us back into command level.

Entering 'T' and 'CR' will display line 1:-

```
   1 ORG 8000H
ZEN >
```

Now entering 'P13' and 'CR' will list the program from line 1 through to the end of the program which is always displayed as 'EOF'. If one entered 'P8' only the first 8 lines would be listed, so if the whole program is to be listed ensure you enter 'P' followed by a value equal to, or larger than, the last line number. Notice that the existing lines in memory have been moved up 2 lines.

Once again enter 'A' and 'CR' followed by 'CR' in response to 'OPTION' prompt to see if our program is correct and will assemble. If one entered the program as shown it should stop and display:-

```
HUH?
   6 NEXT:CALL0012H
ZEN >
```

Faced with this error one must look at the line and discover the mistake because the prompt 'HUH?' does not tell us much, only this will happen many times when writing your own programs. The line looks O.K. but the fault lies in the basic fact that we did not enter a space between CALL and the address.
Enter 'N' and 'CR' and the line will be displayed with the cursor to the right of the line of characters:-

```
   6 NEXT:CALL0012H
```

Simply delete the characters from the right, DO NOT USE THE CURSOR KEYS, until the cursor is over the first zero after CALL and enter a space followed by 0012H and 'CR'.
The line should now look like this:-

```
   6 NEXT:CALL 0012H
```

Entering 'A' followed by 'CR' twice should result in no error message this time and the 'ZEN' prompt should be displayed almost immediately on the next line, which tells us that it assembled O.K. and is loaded into memory.
Enter 'GSTART' followed by 'CR' and the screen will display:-

```
BKPT >
```

this is asking us to enter a breakpoint in the program, for if one is testing certain parts of a lengthy program it can be halted at a

particular point, either an absolute address in memory or a label within the listing, and control will pass back to ZEN. This can be very useful as machine code programs run so quickly that it is very hard to keep track of them.
In this case we do not want to enter a breakpoint, so in response to the 'BKPT' prompt enter the 'CR' key.

The display should appear:-

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

ZEN >
```

Don't expect too much from your first machine code program, this was only to demonstrate the principles in entering code, but now we have lost all the bugs it seems a good time to assemble the program onto the screen to see what has happened. Enter 'A' and 'CR' and this time when prompted for 'OPTION' enter 'V' and 'CR' and the result should be this:-

```
PAGE   1

  1                       ORG  8000H
  2                       LOAD 8000H
  3               LOOP:   EQU  1203H
  4 8000 CD0600   START:  CALL 0006H
  5 8003 3E41             LD   A,"A"
  6 8005 CD1200   NEXT:   CALL 0012H
  7 8008 3C               INC  A
  8 8009 FE5B             CP   "Z"+1
  9 800B 20F8             JR   NZ,NEXT
 10 800D CD0600           CALL 06H
 11 8010 C30312           JP   LOOP
 12                       END
ZEN >
```

In the above program, due to its simplicity, we did not document the functions of any lines but in a longer program it will be essential to describe certain parts of the programs. Comments can be included in any line by simply adding a semi-colon followed by the comment. To add a comment to line 3 enter 'T3' and 'CR' followed by 'N' and 'CR' and line 3 should be displayed with the cursor to the right of the characters:-

```
   3 LOOP:EQU 1203H
```

add the following:-

```
                    ;JUMP ON COMPLETION  and'CR'
```

This line when listed will now show the comments after the semi-colon which will remind one at a future date what the line was achieving. Also a line may be entered with no operands just a semi-colon followed by the comments, these will be used on subsequent listings for clarity. If one has a printer the assembled listing to 'E' for external will show these comment fields formatted to the right of the paper, but they will not be displayed on screen when assembling to the 'V' for video option due to the limitations of the 40 column screen.

ALTERATIONS and ADDITIONS

If one followed and understood the instructions and how they worked try the following:-

Alter the program to print the alphabet from Z down to A.

Change line 5 to LD A,"Z"

line 7 to DEC A

line 8 to CP "A"-1

This will initially load register A with letter Z and instead of incrementing in line 7 it will decrement, so the first time round the value in register A will reduce to the letter Y and so on. Line 8 checks if has reached A-1 and if not loops back to print again.

SCREEN MESSAGES

One will eventually require messages and inputs to be printed on screen, and as this test program is short it is ideal for modifying quite simply. The first working line of the program after the equates, origin and load entries is line 4, so enter 'T4' and 'CR' and line 4 will get displayed:-

```
   4 START:CALL 0006H
ZEN >
```

Entering 'E' and 'CR' will now enable one to add lines to the program, and move the existing lines up in memory.

```
               TO
DISPLAYED     ENTER

   4          NEWSTART:LD DE,MESSAGE1
   5          CALL 0015H
   6          .
ZEN >
```

These new instructions are thus:-
LD DE,MESSAGE1 loads register pair DE with the address in memory of the start of a screen message which will have the label MESSAGE1 assigned to it. CALL 0015H is a monitor routine which prints, at the cursors current position on screen, the message which starts at the address stored in DE. Furthermore the message must end with the code for carriage return which is 0Dhex.
The next job is to enter MESSAGE1 into our program. List the program on screen to discover the last line number, as it is here we will place the string of characters in our message. END should appear as line 14, so enter 'T14' and 'CR' followed by 'E' and 'CR'

```
               TO
DISPLAYED     ENTER

   14         MESSAGE1:DB"▦▦▦▦TEST▦▦▦",0DH
   15         .
```

ZEN >

Notice that ZEN allows cursor control characters to be accepted into print strings also the string must be terminated, after the closing quotes, by a comma and '0DH' to signify the end of string. Unlike Basic ZEN only allows entries on one screen line, therefore if your message needed to be longer finish the first line of the message by adding the closing quotes and continue the message on the following line making sure it commences with 'DB"' and only enter ',0DH' at the end of message.
In order to run the program it must be assembled again, making sure no bugs have crept in. When assembling to the screen it will be seen that although long messages are not printed in full, the bytes representing that message are entered into memory.
Running the program can be entered as 'G8000H' or 'GNEWSTART'. It will be seen that the screen clears and 'TEST' gets printed on the third line, and the alphabet gets printed, in reverse order, 3 lines lower due to the cursor characters within the new print string.
Ensure your program lists as below, as we shall alter it further.

```
   1 ORG 8000H
   2 LOAD 8000H
   3 LOOP:EQU 1203H;JUMP ON COMPLETION
   4 NEWSTART:LD DE,MESSAGE1
   5 CALL 0015H
   6 START:CALL 0006H
   7 LD A,"Z"
   8 NEXT:CALL 0012H
   9 DEC A
  10 CP "A"-1
  11 JR NZ,NEXT
  12 CALL 06H
  13 JP LOOP
  14 MESSAGE1:DB"▒▒▒TEST▒▒▒",0DH
  15 END
EOF
```

USER INPUTS 1

We will assume that we wish the user to input a number from 1 to 9 in order for the alphabet to be printed several times. A routine exists within the monitor area that will stop the program and wait for a key to be pressed before continuing and can be utilised quite simply.

Alter line 14 by entering 'T14' and 'CR' followed by 'N' and 'CR' to alter MESSAGE1. With the cursor to the right of the line delete back to the Clear Screen symbol and alter the line to the following:-

```
14 MESSAGE1:DB"▦HOW MANY 1 to 9",0DH
```

Now the program will clear the screen and print the new message on the top line.

We also need to change the program to accept an input from the keyboard between 1 and 9. Enter 'T6' and 'CR' followed by 'E' and 'CR'.

```
              TO
DISPLAYED     ENTER

    6         TIMES:CALL 09B3H
    7         CALL 0BCEH
    8         CP 31H
    9         JR C,TIMES
   10         CP 3AH
   11         JR NC,TIMES
   12         SUB 30H
   13         LD B,A
   14         .
ZEN >
```

Line 6 (labelled TIMES) now calls a routine within the monitor (09B3H) which halts the program and waits for a key to be pressed.

The Display code of the key pressed is held in register A, but we require the ASCII equivalent of the key pressed, so line 7 calls another monitor routine at 0BCE hex which converts the contents of register A into ASCII code.

As we only require keys 1 to 9 to be accepted the contents of register A must be checked, and line 8 checks that the key pressed was equal to or greater than 31H, which is the ASCII code for the number 1 (check with the ASCII code table). It simply subtracts (temporarily) 31H from the A register and if it contained a lower ASCII code than 31H the carry flag will be set, hence line 9 is a relative jump back to line 6, for the processor to wait for another key to be pressed, if there was such a carry.

This then tests for a lower ASCII input and subsequently it must now check for a higher key than 9. Line 10 compares for 3AH, which in the ASCII table will be seen to equal the colon ':' which is one higher than 9. Line 11 is a relative jump back to line 6 if after subtracting 3AH from register A the carry flag is not set then the key pressed must have been equal to or higher than 3AH, which means the key was higher in the ASCII table than 9 and we must jump back and wait for another key. Assuming that a correct key was entered we now know register A contains a number between 31H and 39H and we must convert this to between 1 and 9, and line 12 does exactly that it subtracts 30H from register A leaving it with a value 1 to 9.

Line 13 loads register B with the contents of register A as B is to be the counter for the amount of times we will print the alphabet. One more line needs to be entered. Enter 'T21' and 'CR' then 'E' and 'CR'

```
              TO
DISPLAYED     ENTER

   21         DJNZ START
   22         .
ZEN >
```

This command was discussed in the 'Special jumps' section in chapter 2 and is a unique Z80 instruction for the B register which decrements B and executes a relative jump back to wherever you nominate, to carry out the instructions in the loop again until B decreases to zero, similar to a FOR..NEXT loop in Basic. In our case it jumps back to line 14 which is labelled START.

One will have to assemble the program before it is capable of being run. If errors occur during assembly refer back to the specified line and check it in this chapter. To run enter 'G8000H' or 'GNEWSTART' and 'CR' for BKPT.

The assembled listing:-

```
PAGE    1                                  TEST
   1                          ORG  8000H
   2                          LOAD 8000H
   3                LOOP:     EQU  1203H                ;JUMP ON COMPLETION
   4 8000 112C80    NEWSTART: LD   DE,MESSAGE1
   5 8003 CD1500              CALL 0015H
   6 8006 CDB309    TIMES:    CALL 09B3H
   7 8009 CDCE0B              CALL 0BCEH
   8 800C FE31                CP   31H
   9 800E 38F6                JR   C,TIMES
  10 8010 FE3A                CP   3AH
  11 8012 30F2                JR   NC,TIMES
  12 8014 D630                SUB  30H
  13 8016 47                  LD   B,A
  14 8017 CD0600    START:    CALL 0006H
  15 801A 3E5A                LD   A,"Z"
  16 801C CD1200    NEXT:     CALL 0012H
  17 801F 3D                  DEC  A
  18 8020 FE40                CP   "A"-1
  19 8022 20F8                JR   NZ,NEXT
  20 8024 CD0600              CALL 06H
  21 8027 10EE                DJNZ START
  22 8029 C30312              JP   LOOP
  23 802C 16484F57 MESSAGE1:  DB   "▥HOW MANY 1 to 9",0DH
  23 8030 204D414E
  23 8034 59203120
  23 8038 96B72039
  23 803C 0D
  24                          END
```

USER INPUTS 2

This section deals with user inputs of unspecified length, as against single key inputs as in the last section. We will dispense with the alphabet, I think we all agree it was becoming boring, and enter a string from the keyboard to be printed a number of times. Enter 'K' and 'CR' to kill the existing program followed by 'E' and 'CR'.

DISPLAYED	TO ENTER
1	ORG 8000H
2	LOAD 8000H
3	LOOP:EQU 1203H
4	PRTMES:EQU 0015H
5	BELL:EQU 003EH
6	NL:EQU 0006H
7	INPSTR:EQU 9000H
8	USER:EQU 0003H
9	WAITKY:EQU 09B3H
10	DACN:EQU 0BCEH
11	;
12	LD DE,MESS1
13	CALL BELL
14	CALL PRTMES
15	CALL NL
16	LD DE,INPSTR
17	CALL USER
18	CALL NL
19	LD DE,MESS2
20	CALL PRTMES
21	TIMES:CALL WAITKY
22	CALL DACN
23	CP 31H
24	JR C,TIMES
25	CP 3AH

```
  26          JR NC,TIMES
  27          SUB 30H
  28          LD B,A
  29          AGAIN:CALL NL
  30          LD DE,INPSTR
  31          CALL PRTMES
  32          CALL NL
  33          DJNZ AGAIN
  34          JP LOOP
  35          MESS1:DB"▣ENTER A STRING",0DH
  36          MESS2:DB"HOW MANY TIMES 1-9",0DH
  37          END
  38          .
ZEN>
```

In this example more addresses have been included in the EQU section, as on longer programs it will be far simpler to enter instructions i.e. CALL BELL than entering CALL 003EH within the program each time.

Line 7 denotes the area in which the input string will be stored when entered from the keyboard, 9000H.

Line 8 USER (0003H) is the monitor input routine which accepts input from the keyboard, the program continues after the string has been terminated by entering the 'CR' key.

Line 9 WAITKY is the routine used in the last program at 09B3H which waits for a single key input.

Line 10 is the label given to the routine which converts Display code to ASCII in register A.

The remainder of the program is similar to the previous one with the addition of line 30 which loads DE with our string which is stored at 9000H, afterwhich it is printed with CALL PRTMES (0015H)

Entering 'A' and 'CR' followed by 'V' and 'CR' should produce the assembled listing as below. To run the program enter G8000H and 'CR' for the 'BKPT' prompt, afterwhich the screen will clear and the 'ENTER A STRING' message will be printed. After one has entered a string of characters the 'HOW MANY TIMES 1-9' message will be shown and on entering a value between 1 and 9 the string will be printed with a clear line between each.

```
PAGE   1                    INPUT STRING

                             ORG  8000H
                             LOAD 8000H
                  LOOP:      EQU  1203H
                  PRTMES:    EQU  0015H
                  BELL:      EQU  003EH
                  NL:        EQU  0006H
                  INPSTR:    EQU  9000H
                  USER:      EQU  0003H
                  WAITKY:    EQU  09B3H
                  DACN:      EQU  0BCEH
                  ;
 8000 113D80                 LD   DE,MESS1
 8003 CD3E00                 CALL BELL
 8006 CD1500                 CALL PRTMES
 8009 CD0600                 CALL NL
 800C 110090                 LD   DE,INPSTR
 800F CD0300                 CALL USER
 8012 CD0600                 CALL NL
 8015 114D80                 LD   DE,MESS2
 8018 CD1500                 CALL PRTMES
 801B CDB309      TIMES:     CALL WAITKY
 801E CDCE0B                 CALL DACN
 8021 FE31                   CP   31H
 8023 38F6                   JR   C,TIMES
 8025 FE3A                   CP   3AH
 8027 30F2                   JR   NC,TIMES
 8029 D630                   SUB  30H
 802B 47                     LD   B,A
 802C CD0600      AGAIN:     CALL NL
 802F 110090                 LD   DE,INPSTR
 8032 CD1500                 CALL PRTMES
 8035 CD0600                 CALL NL
 8038 10F2                   DJNZ AGAIN
 803A C30312                 JP   LOOP
 803D 16454E54    MESS1:     DB   "■ENTER A STRING",0DH
 8041 45522041
 8045 20535452
 8049 494E470D
 804D 484F5720    MESS2:     DB   "HOW MANY TIMES 1-9",0DH
 8051 4D414E59
 8055 2054494D
 8059 45532031
 805D 2D390D
                             END
```

SAVING PROGRAMS

Although one probably won't need to save this program on tape it is a good idea to use this small program to practise getting it right, it is not so straightforward as saving a basic program, so making mistakes now will be less costly than when your own machine code masterpiece is at stake.

There are 2 methods of saving machine code programs. The first is to save the source file. Source files (or programs) are made up of the pure text which has been entered from the keyboard. One will require this option for saving unfinished programs, which obviously cannot be assembled in that state, for future loading using ZEN which would be achieved by entering 'R' and 'CR' after the ZEN prompt. Entering 'H' and 'CR' will now display the start and end of the source file and the top of memory. At this stage the last program should display:-

```
3000 31C9 CFFF
```

Although in some earlier versions of ZEN written for the MZ-80K this could be:-

```
2500 26C9 CFFF
```

as these took up less memory and the user file started lower in RAM at 2500H

If one enters 'Q3000H' (Q2500 on the early version) and 'CR' the text entered will be shown in memory byte by byte. To save a source file enter 'W' and 'CR' and one will be prompted for a file name, afterwhich it will be saved on tape as normal.

The second method is for saving the object file. Object files are the assembled program, and what gets saved is the pure machine code file, without comments, ready to run. In our program it could be saved and then run directly from the Monitor, without Basic or ZEN, by simply loading although one would need to alter the EQU LOOP from 1203H to the mainloop address of the ROM Monitor being used.

To test that one is conversant in saving an object file carry out the following:-

Alter what should be line 3 by entering 'T3' and 'CR' and it should get displayed. Now enter 'N' and 'CR' and alter the address following the EQU from 1203H to one of the following depending on your machine:-

MZ-80K alter 1203H to 0082H
MZ-80A alter 1203H to 0095H
MZ-700 alter 1203H to 00ADH

One will need to assemble these programs once again but if the above entry is correct that will take no time at all only this time assemble to the screen by entering 'V' and 'CR' as we MUST know the end address of the file. Line 36 shows the last few bytes in the program, and one can see this last line starts with address 805DH and contains 3 bytes making the last byte 805FH.

Place a fresh tape in the computer and enter 'WO' which stands for write object. One will be prompted for the START address so enter '8000H' and 'CR', it is important to enter the suffix 'H' otherwise ZEN will believe it is a decimal number which it is not.

Next prompt is for the STOP address so enter '805FH' and 'CR' which is the last byte of the program.

The next prompt is for the EXEC address which is where the program should run from. In this case we want to run from the same address as it loaded from so enter once again '8000H' and 'CR'. EXEC is added because a program does not always execute from its start address in memory. It may be that a program is written and then has some screen graphics titles added to the end of it but which one wants to run first, so the execution address could well be different to that of the loading one.

This is followed by the LOAD prompt for the address at which it should load into, and again enter '8000H'.

The final prompt is for a file name, we could simply call this 'TEST' and all that remains is to press the 'PLAY and RECORD' keys.

Once the file has been saved switch off the computer, wait a few seconds, (never switch off and on quickly) and turn it back on and load the test program which, after a few seconds, will automatically run, if you saved it correctly, and when finished will jump into the Monitor mainloop and display the '*' symbol.

Monitor Routines used in this chapter:-
0003H User input from keyboard
0006H Newline
0012H Print character stored in the A register
0015H Print message starting at address pointed to by DE
003EH Sound bell
0082H Mainloop MZ-80K
0095H Mainloop MZ-80A
00ADH Mainloop MZ-700
09B3H Wait for key input and store display code value in A reg
0BCEH Convert display code to ASCII in A reg

4

ROM Routines

This chapter demonstrates some of the routines which are provided in the Monitor section of memory.

TABLE CONSTRUCTION

The following program uses the keyboard input to produce notes within the range Low A to High D, which gives it some appeal, but its main purpose is to demonstrate one method of accessing tables.

The keys which will produce sounds are as follows:-

```
 2 3 4   6 7   9 0 -
Q W E R T Y U I O P
```

The Sharp requires the storage of 2 bytes in the ratio storage address at 11A1H and 11A2H to produce sound, and this 2 byte value is divided into 2Mhz to determine the frequency of the note to be played.

The generated note therefore:- freq.(hz) = 2 Mhz/ratio

I am not a musician so for any technical information on this subject may I suggest one obtains a manual on the 74LS221 chip. The dividing ratios and resulting frequencies are listed overleaf for the musically minded, but for this program all we are interested in is the dividing ratio column which must be entered in the table within the program to produce a recognisable note in relation to whatever key is pressed. We are only entering notes from Low A to High D, but with referring to the dividing ratio table one could modify the program to play other notes.

Scale	Frequency (Hz)	Dividing ratio
Low F	175	2CA4
F #	186	2A00
G	197	27A8
G #	208	2582
A	222	2331
A #	233	2187
B	245	1FE3
Middle C	261	1DEE
C #	277	1C34
D	294	1A92
D #	311	191E
E	329	17BF
F	350	1652
F #	373	14F1
G	394	13D4
G #	417	12BC
A	444	1198
A #	466	10C3
B	490	0FF1
High C	522	0EF7
C #	553	0E20
D	590	0D3D
D #	621	0C94
E	658	0BDF
F	699	0B2D
F #	745	0A7C
G	788	09EA

In the listing overleaf line 13 checks if the key entered is 'L' which will quit the program and return to ZEN. Line 18 checks for the end of table marker which is 0F0H which will signify that the key pressed was not in the table so no action should be taken.

New Monitor Routines:-

0044H Sounds note according to stored bytes at 11A1/11A2H

0047H Stops sound

```
PAGE    1

                      ORG  8000H
                      LOAD 8000H
              LOOP:   EQU  1203H
              GETKY:  EQU  001BH
              MSTA:   EQU  0044H             ;START MUSIC
              MSTP:   EQU  0047H             ;STOP MUSIC
              NOTE:   EQU  11A1H             ;NOTE STORAGE
              ;
8000 CD4700   START:  CALL MSTP
8003 CD1B00   GET:    CALL GETKY
8006 B7               OR   A
8007 28F7             JR   Z,START
8009 FE4C             CP   "L"               ;RETURN TO ZEN
800B CA0312           JP   Z,LOOP
800E 47               LD   B,A               ;KEY INTO REG B
800F 212B80           LD   HL,TABLE
8012 7E       COMPR:  LD   A,(HL)
8013 FEF0             CP   0F0H              ;END OF TABLE?
8015 28EC             JR   Z,GET             ;YES. INVALID KEY
8017 23               INC  HL
8018 B8               CP   B                 ;COMPARE KEY
8019 2804             JR   Z,FOUND
801B 23               INC  HL                ;NOT FOUND. JUMP
801C 23               INC  HL                ;TO NEXT IN TABLE
801D 18F3             JR   COMPR
801F 5E       FOUND:  LD   E,(HL)
8020 23               INC  HL
8021 56               LD   D,(HL)
8022 ED53A111         LD   (NOTE),DE
8026 CD4400           CALL MSTA
8029 18D8             JR   GET
              ;
              ;SCALE TABLE
802B 51       TABLE:  DB   "Q"
802C 3123             DW   2331H
802E 32               DB   "2"
```

```
8021 8721          DW    2187H
8031 57            DB    "W"
8032 E31F          DW    1FE3H
8034 33            DB    "3"
8035 EE1D          DW    1DEEH
8037 45            DB    "E"
8038 341C          DW    1C34H
803A 34            DB    "4"
803B 921A          DW    1A92H
803D 52            DB    "R"
803E 1E19          DW    191EH
8040 54            DB    "T"
8041 BF17          DW    17BFH
8043 36            DB    "6"
8044 5216          DW    1652H
8046 59            DB    "Y"
8047 F114          DW    14F1H
8049 37            DB    "7"
804A D413          DW    13D4H
804C 55            DB    "U"
804D BC12          DW    12BCH
804F 49            DB    "I"
8050 9811          DW    1198H
8052 39            DB    "9"
8053 C310          DW    10C3H
8055 4F            DB    "O"             ;ALPHA O KEY
8056 F10F          DW    0FF1H
8058 30            DB    "0"             ;NUMERIC 0 KEY
8059 F70E          DW    0EF7H
805B 50            DB    "P"
805C 200E          DW    0E20H
805E 2D            DB    "-"             ;MINUS KEY
805F 3D0D          DW    0D3DH
8061 F0            DB    0F0H            ;END OF TABLE MARK
                   END
```

TIME READ

This program displays a real-time digital clock in large characters. On running the program the cursor will flash while it waits for the hours and minutes to be entered in four digits i.e. 1254.

Descriptions of the program are being suppressed this time, although the listing contains some comments, in an effort to let the reader discover what is going on. If entered correctly one will find that altering the odd command here and there produces interesting results.

For instance line 59 can be altered from 10 to 16 to display the seconds in hex. Hours and minutes are made up of a series of blobs, the display code of which is in line 99, and can be changed. In fact several good machine code techniques can be gained from this program, but enter it correctly first and get it running before making alterations. To quit the program press SHIFT/BREAK which will return to ZEN mainloop at 1203H. This can be altered to the Monitor mainloop as was shown in chapter 3. One point which has not been covered is in lines 57/58 where the cursor, used for displaying the seconds, is positioned by entering the X/Y co-ordinates into HL and storing this at DSPXY (1171H) which in this case is 0513H - line 5, column 19 (13H) - as this is where the current cursor position is always stored.

New Monitor Routines used in this program:-

001EH	Shift/Break key check
0033H	Time set. Enter with DE=time in seconds as 4 digit hex number
003BH	Time read. Exit with DE " " " " " "
03C3H	Prints ASCII contents of reg A
03F9H	Converts ASCII contents of reg A to Hex
0DA6H	Checks vertical blanking on screen
0DDCH	Controls screen display depending on reg A (see respective manuals)

This source listing is derived from the object machine code program 'WEE BEN' and is included by kind permission of Knights T.V. and Computers of Aberdeen.

```
                        ORG   4600H
                        LOAD  4600H
              NL:       EQU   0006H
              USER:     EQU   0003H
              BUFF:     EQU   9000H
              TIMST:    EQU   0033H
              TIMRD:    EQU   003BH
              DSPXY:    EQU   1171H
              VBLNK:    EQU   0DA6H
              HEX:      EQU   03F9H
              PRTHX:    EQU   03C3H
              BREAK:    EQU   001EH
              LOOP:     EQU   1203H
              DPCT:     EQU   0DDCH
              ;
              ;
4600 CD0600   START:    CALL NL
4603 110090             LD   DE,BUFF           ;STORE TIME INPUT
4606 CD0300             CALL USER              ;INPUT TIME
4609 210000             LD   HL,0000H
460C 01A08C             LD   BC,8CA0H          ;36,000(10 HRS/SECS)
460F CDB346             CALL CONVHX            ;CONVERT TO HEX
4612 01100E             LD   BC,0E10H          ;3,600(1 HR/SECS)
4615 CDB346             CALL CONVHX
4618 015802             LD   BC,0258H          ;600(10 MINS/SECS)
461B CDB346             CALL CONVHX
461E 013C00             LD   BC,003CH          ;60(1 MIN/SECS)
4621 CDB346             CALL CONVHX
4624 EB                 EX   DE,HL             ;TIME IN SECONDS=DE
4625 CD3300             CALL TIMST             ;SET TIME
4628 3EC6               LD   A,0C6H            ;CLEAR SCREEN CHAR.
462A CDDC0D             CALL DPCT              ;DO IT
462D 3E4A               LD   A,4AH             ;CHAR TO SPLIT HR.MINS
462F 32F4D1             LD   (0D1F4H),A        ;DISPLAY 1st DOT
4632 326CD2             LD   (0D26CH),A        ;DISPLAY 2nd DOT
4635 CD1E00   TIME:     CALL BREAK             ;WANT TO STOP?
```

```
PAGE   2

4638 CA0312                JP   Z,LOOP         ;YES, BACK TO ZEN
463B CD3B00                CALL TIMRD          ;DE=TIME IN SECONDS
463E 210F0E                LD   HL,0E0FH       ;3,599(1 HOUR-1 SEC
4641 ED52                  SBC  HL,DE          ;ACTUAL SECS-59
4643 3805                  JR   C,PRTTIM       ;INCREASE MINUTES
4645 21C0A8                LD   HL,0A8C0H      ;43,200(12 HRS/SECS)
4648 19                    ADD  HL,DE
4649 EB                    EX   DE,HL
464A 01A08C      PRTTIM:   LD   BC,8CA0H
464D 2195D1                LD   HL,0D195H      ;SCRN POSN HRS(TENS)
4650 CD8446                CALL DISPLAY
4653 01100E                LD   BC,0E10H
4656 219DD1                LD   HL,0D19DH      ;SCRN POSN HRS(UNITS)
4659 CD8446                CALL DISPLAY
465C 015802                LD   BC,0258H
465F 21A7D1                LD   HL,0D1A7H      ;SCRN POSN MINS(TENS)
4662 CD8446                CALL DISPLAY
4665 013C00                LD   BC,003CH
4668 21AFD1                LD   HL,0D1AFH      ;SCRN POSN MINS(UNITS)
466B CD8446                CALL DISPLAY
466E 211305                LD   HL,0513H       ;X/Y POSN OF SECONDS
4671 227111                LD   (DSPXY),HL     ;POSITION CURSOR
4674 010A00                LD   BC,10
4677 CDC446                CALL COUNT
467A 07                    RLCA
467B 07                    RLCA
467C 07                    RLCA
467D 07                    RLCA
467E 83                    ADD  A,E
467F CDC303                CALL PRTHX          ;PRINT SECONDS
4682 18B1                  JR   TIME
4684 CDC446      DISPLAY:  CALL COUNT
4687 D5                    PUSH DE
4688 11D246                LD   DE,NUMBER
468B 47                    LD   B,A
468C 07                    RLCA
```

```
PAGE   3

468D 07                     RLCA
468E 80                     ADD  A,B
468F 83                     ADD  A,E
4690 5F                     LD   E,A
4691 0E05                   LD   C,05H        ;5 COLUMNS WIDE
4693 E5         COLPRT:     PUSH HL
4694 1A                     LD   A,(DE)
4695 D5                     PUSH DE
4696 112800                 LD   DE,40        ;LINE WIDTH
4699 0608                   LD   B,08H        ;LINES TO PRINT
469B CDA60D                 CALL VBLNK
469E 07         BLANK:      RLCA
469F 380E                   JR   C,BLOB       ;GET DISPLAY CHARACTER
46A1 3600                   LD   (HL),00H
46A3 19         NXTLN:      ADD  HL,DE        ;MOVE DOWN 1 LINE
46A4 10F8                   DJNZ BLANK
46A6 D1                     POP  DE
46A7 E1                     POP  HL
46A8 13                     INC  DE
46A9 23                     INC  HL
46AA 0D                     DEC  C            ;NEXT ROW
46AB 20E6                   JR   NZ,COLPRT
46AD D1                     POP  DE
46AE C9                     RET
46AF 3647       BLOB:       LD   (HL),47H     ;DISP CODE OF BLOB
46B1 18F0                   JR   NXTLN
46B3 1A         CONVHX:     LD   A,(DE)
46B4 13                     INC  DE
46B5 CDF903                 CALL HEX
46B8 FE00                   CP   00H
46BA C8                     RET  Z
46BB D5                     PUSH DE
46BC 50                     LD   D,B
46BD 59                     LD   E,C
46BE 47                     LD   B,A
46BF 19         CNHEX1:     ADD  HL,DE
```

```
PAGE   4

46C0 10FD                 DJNZ CNHEX1
46C2 D1                   POP  DE
46C3 C9                   RET
46C4 E5        COUNT:     PUSH HL
46C5 EB                   EX   DE,HL
46C6 AF                   XOR  A
46C7 ED42      COUNT1:    SBC  HL,BC
46C9 3803                 JR   C,CTEND
46CB 3C                   INC  A
46CC 18F9                 JR   COUNT1
46CE 09        CTEND:     ADD  HL,BC
46CF D1                   POP  DE
46D0 EB                   EX   DE,HL
46D1 C9                   RET
               ;
46D2 7E818181  NUMBER:    DB   7EH,81H,81H,81H,7EH  ;   0
46D6 7E
46D7 2141FF01             DB   21H,41H,0FFH,01H,01H ;   1
46DB 01
46DC 43858991             DB   43H,85H,89H,91H,61H  ;   2
46E0 61
46E1 46819191             DB   46H,81H,91H,91H,6EH  ;   3
46E5 6E
46E6 1C2444FF             DB   1CH,24H,44H,0FFH,04H ;   4
46EA 04
46EB F2919191             DB   0F2H,91H,91H,91H,8EH ;   5
46EF 8E
46F0 7E919191             DB   7EH,91H,91H,91H,4EH  ;   6
46F4 4E
46F5 C0808F90             DB   0C0H,80H,8FH,90H,0E0H;   7
46F9 E0
46FA 6E919191             DB   6EH,91H,91H,91H,6EH  ;   8
46FE 6E
46FF 72898989             DB   72H,89H,89H,89H,7EH  ;   9
4703 7E
                          END
```

LOADER PROGRAM

When loading a machine code program one may have seen a different screen message displayed than the usual one or, as is becoming more popular, the complete display could alter to a graphics title while the program appears to be still loading.

The answer lies in the fact that two programs have been loaded, the second automatically. The first short program contains the titles and a loading routine for the second larger program. When the first program has loaded it executes immediately so printing the titles on screen and enters a loading routine for the second. Execution is so fast that the tape stops for a minimal time and starts again almost without being noticed. Only one 'Loading program name' message appears on screen as the loading routine jumps into the middle of the Monitor loading routines so missing the loading message. These routines are different for the 3 Sharps and are mentioned in the listing.

The loader program begins with the screen title message, in the example it will display 'NOW LOADING MAIN PROGRAM', but this can be expanded upon as will be explained later. It continues by loading 9 bytes into a free area at the top of the Monitor area, at 11F5H, which actually control the loading of the second program, afterwhich it jumps to that routine to commence the loading.
Debugging and testing this loader program could cause problems if one allows it to execute line 23 (JP 11F5H), therefore a label (STP) has been added in order that when the BKPT option is displayed entering STP will stop the program after the titles and before it jumps to the loading routine.:-

```
ZEN>GSTART
BKPT>STP
```

This will enable one to thoroughly test out the titles and graphics before saving.

Although the ORG is set at 8000H, which is ideal for testing, before saving the object file it could be altered to 1200H and assembled again for loading at the lower address. This also means that the second program could be set to ORG 1200H before saving and the loader program will be overwritten and dissappear from memory as the main program loads in.
To test if the loader operates correctly it would be wise to enter and save the program as it is listed first. And without rewinding the tape enter a second program such as the short inputs program from the earlier chapter. Please alter the various EQU addresses for your Sharp model.

First save the source program for future alterations then save the object version by altering the ORG to 1200H and re-assemble. Do not alter the LOAD address as this would overwrite ZEN and crash out.
It must be remembered that although now it has been assembled from 1200H to 1243H at present it resides in memory from 8000H to 8043H, and it is this area of memory which must be saved on tape, with larger programs this can cause calculation problems. Now complete the enter the following where prompted:-

```
ZEN>WO
START>8000H
STOP>8043H
EXEC>1200H
LOAD>1200H
```

Once saved it should be verified by entering 'VO' and 'CR', and if it verifies OK then do not rewind the tape as it will be used from that position to save the object file of the main program. One should now enter 'K' to kill the file, the same as NEW in Basic, and enter or load in a small source program, previously recorded and fully tested, alter the ORG address to 1200H if one wishes, assemble and save the object file onto the same tape as the loader program has been saved.
One should now possess an object tape which will load directly from Monitor and after a few seconds should display the message 'NOW LOADING MAIN PROGRAM' on the fifth line of the screen whilst loading the main program.

```
                       ORG  8000H
                       LOAD 8000H
              NL:      EQU  0006H
              PRTMES:  EQU  0015H
              RDHDR:   EQU  04D8H
              ;MZ-80A ALTER ABOVE TO 04CFH
              ERROR:   EQU  0107H
              ;MZ-80A ALTER ERROR TO 00CFH
              ;MZ-80K ALTER ERROR TO 01A4H
              LOAD:    EQU  0121H
              ;MZ-80A ALTER LOAD TO 00E9H
              ;MZ-80K ALTER LOAD TO 00F4H
              ;
8000 112080   START:   LD   DE,MESS1
8003 CD1500            CALL PRTMES
8006 CD0600            CALL NL
              ;
8009 11F511   LDSHFT:  LD   DE,11F5H            ;Loader
800C 211780            LD   HL,LOADER           ;shifter
800F 010900            LD   BC,9                ;routine
8012 EDB0              LDIR
              ;
8014 C3F511   STP:     JP   11F5H
              ;
8017 CDD804   LOADER:  CALL RDHDR
801A DA0701            JP   C,ERROR
801D C32101            JP   LOAD
              ;
              ;INITIAL SCREEN DISPLAY
              ;WHILE MAIN PROG. LOADS
8020 16111111 MESS1:   DB   "[illegible]"
8024 11111313
8028 131313
802B 4E4F5720          DB   "NOW LOADING "
802F 4C4F4144
8033 494E4720
8037 4D41494E          DB   "MAIN "
803B 20
803C 50524F47          DB   "PROGRAM",0DH
8040 52414D0D
                       END
```

ADDING TITLES

If one required the title to cover the whole screen one solution would be to store the display code of each screen location in the loader program and use the LDIR instruction. This takes careful planing as each of the 1000 bytes which make up one complete screen would require entering individually although execution time is still remarkably fast. 700 owners could also specify individual colours for the whole screen area too.
To give an example we have only stored 3 screen lines of characters for displaying in the title and will use the previously saved (I hope) source loader program as a kernel from which to expand. Find the line with START as its label, it should have been 14, make it the current line and enter 'Z3' and 'CR' which will delete 3 lines. Now enter 'E' and 'CR' and begin entering:-

```
                TO
DISPLAYED     ENTER

   14         START:LD A,16H
   15         CALL 0012H
   16         LD DE,0D000H
   17         LD HL,MESS1
   18         LD BC,120
   19         LDIR
   20         .
ZEN >
```

Initially we load the ASCII for the Clear screen character (16H) into register A and CALL 0012H which will print the contents of A, and clear the screen. Although this is not strictly necessary on the MZ-80K and 700, as the top left position of screen is always the same location in RAM on these models, it is a must for the MZ-80A as the screen does move about after scrolling and this guarantees that the top left corner is D000H, also it will assist by clearing the screen when testing the additions under ZEN before saving the finished product. Next the top of screen (VRAM) is loaded into

register pair DE. HL points to the address where the characters to be displayed are stored at the end of our program, and BC is used as byte counter and contains the amount of characters we wish to transfer to the VRAM area, in this example 3 lines of 40 characters, 120 bytes. This has been entered as a decimal for clarity, as it is not suffixed by 'H', but could have been entered as LD BC,78H instead. The LDIR instruction simply transfers the contents of HL into the address of DE and increments both registers and decrements BC until BC equals zero.
We must now add the 120 bytes which make up the display.
After these additional lines MESS1 should have moved up to line 34 in the program, find it and make it the current line. Wipe out the remainder of the previous message by entering 'Z10' and 'CR' which should delete up to EOF and enter 'E' and 'CR' and enter these lines:-

```
                TO
DISPLAYED       ENTER

   34           DISPL:DB 4BH,78H,78H,78H,78H; L
   35           DB 78H,78H,78H,78H,78H,78H; I
   36           DB 78H,78H,78H,78H,78H,78H; N
   37           DB 78H,78H,78H,78H,78H,78H; E
   38           DB 78H,78H,78H,78H,78H,78H
   39           DB 78H,78H,78H,78H,78H,78H; NO.
   40           DB 78H,78H,78H,78H,4CH; 1
   41           DB 79H,0,0,0,0,0,0,0,0;  L
   42           DB 0,0,19H,0FH,07H,12H;  I
   43           DB 0,10H,12H,0FH,07H,12H;  N
   44           DB 01H,0DH,0,0EH,01H,0DH,05H;  E
   45           DB 0,0,0,0,0,0,0,0,0,0,0,79H;  2
   46           DB 6FH,78H,78H,78H,78H,78H;    L
   47           DB 78H,78H,78H,78H,78H,78H;    I
   48           DB 78H,78H,78H,78H,78H,78H;    N
   49           DB 78H,78H,78H,78H,78H,78H;    E
   50           DB 78H,78H,78H,78H,78H,78H
   51           DB 78H,78H,78H,78H,78H,78H;    NO.
```

```
  52        DB 78H,78H,78H,78H,4CH;    3
  53        END
  54        .
ZEN >
```

The label MESS1 has been altered to DISPL as it is no longer a message in the true sense of the word. We are not loading DE with a message and calling the print message monitor routine as we did before. All these bytes are in Display code, which must be used when direct screen addressing takes place, and they do not need to end with the carriage return code (0DH) as the end is pointed to by the size of register BC. If we had loaded BC with only 40 then it would have only transferred the first 40 bytes to the screen, although the complete block contained 120 bytes, one could have displayed only the final 40 bytes by loading HL with the starting address of the start of the third block of 40 bytes.
If one assembles this program and corrects any incorrect lines, it can be tested by entering 'GSTART' and for the BKPT prompt enter 'STP' as we did before.
'YOUR PROGRAM NAME' should have been displayed in the centre of line 2 with a surround. Obviously it can be altered at will, but after each item change remember to keep assembling and testing, as we have just done, and if it grows in size add the amount of bytes to BC in line 18, else they will not get displayed.
Altering the position of this display is straightforward as all that needs altering is the starting address which is held in register DE in line 16. At present it is loaded with the top left position (D000H) but can simply be moved down screen by adding 28H for each line. Suppose one wanted it displayed starting 8 lines down. 8x40=320, convert this to Hex, 140H, and add to D000H, which would become D140H. Alter line 16 by entering 'T16' and 'CR' followed by 'N' and 'CR' and using the delete key erase back and alter the address from 0D000H to 0D140H. Assemble again and test.

For displaying the same character in several successive locations, say complete rows of 40 characters, and to save time in entering each byte individually an alternative method can be employed.

At present the program begins the display 8 lines from the top, 9 including the top line, we could simply fill this area with the following additions. Make the target line 16 and enter 'E' and 'CR':-

```
               TO
DISPLAYED      ENTER

   16          LD D,44H
   17          LD HL,0D000H
   18          LD BC,0A0H
   19          CALL DISRTN
   20          LD D,46H
   21          LD BC,0A0H
   22          CALL DISRTN
   23          .
ZEN >
```

Line 16 loaded the display code for the diamond character into register D. Line 17 loaded HL with the top of VRAM (D000H) and line 18 loaded the byte counter (registers BC) with A0H which equals 160, 4 lines of 40 characters. Notice HL was only loaded with D000H once in line 17 as the subroutine (DISRTN) will increment HL and when it returns from DISRTN it will point to the start of line 5 of the screen (D0A0H) already. Line 20 loads the display code of the club character into D and line 21 loads up BC with the same amount of characters to display, 4 lines, and DISRTN is called yet again.
Now to add the subroutine DISRTN. If all is going well the last line of the program should be 60, make this the target and enter 'E' and 'CR'.

```
               TO
DISPLAYED      ENTER

   60          DISRTN:LD A,B
   61          OR C
   62          RET Z
```

```
    63          LD (HL),D
    64          INC HL
    65          DEC BC
    66          JR DISRTN
    67          .
ZEN >
```

First time round register B=0 and C=A0H. Testing for zero is carried out by loading B into A, and C is tested for being equal to zero by OR C which sets the zero flag if C=0. A return to the program is made only when C has reduced to zero. The contents of D, the character to display, is loaded into the address pointed to by HL and HL then gets incremented whilst BC is decremented and a relative jump takes us back to test again for C being equal to zero.
If one lists the program it will be seen that line 23 LD DE,0D140H is not necessary as at this point HL contains 0D140H and this line could be altered to EX DE,HL which exchanges the register contents, so this would effectively do the same job. Assemble the program as usual, making sure that after all these additions that the last line is still END, otherwise it will not assemble, and test as before entering 'GSTART' and 'STP' for the BKPT prompt. Once this is done one can experiment with the titles before actually recording the object file on tape.

ADDING COLOUR

This last subroutine could be used for block colouring of the screen on the MZ-700, but for individual bytes one would need to use the first method of entering byte by byte and use the LDIR instruction making sure HL contained the VRAM address plus 800H, which is where the colour codes for each byte of the screen area are stored, top left is D800H. Block colouring will also refer to this address.
After the clear screen routine in lines 14 and 15, labelled START, is where the block colours can be added. Firstly we colour the first 4 lines of the screen, which display the diamond symbol. We will set the foreground colour to red and background to white. Therefore the

value assigned to register D will be 27H, 2 is red and 7 white. Registers HL will point to the start of the colour RAM area (D800H), then BC will be loaded with A0H again, being the number of bytes to colour. This will be repeated with the next 4 lines being the club characters only the colour will be red on white (20H), and then the colour of the 3 lines of the original display will be altered to 62H, yellow on red, but this time BC should be equal to 3 lines, 40x3=120, 78H. Enter 'T16' and 'CR' followed by 'E' and 'CR'.

```
              TO
DISPLAYED    ENTER

   16        LD D,27H
   17        LD HL,0D800H
   18        LD BC,0A0H
   19        CALL DISRTN
   20        LD D,20H
   21        LD BC,0A0H
   22        CALL DISRTN
   23        LD D,62H
   24        LD BC,78H
   25        CALL DISRTN
   26        .
ZEN >
```

With careful planning and testing one should be capable of constructing a good title page for display while the main program is loading. Assemble the finished version to screen to obtain the address of the last byte in the program and save as we did in the last section, alter the ORG address if you wish but remember to save the file from where it is loaded (8000H) and add the size to the STOP address.
The assembled listing is printed overleaf.

MZ-80K & A note

The assembled listing includes the colour sections of this program, as these will not be required they have been suffixed with the letter X and may be omitted, as their inclusion will have no effect.

```
PAGE   1                        LOADER+TITLES

                                  ORG  8000H
                                  LOAD 8000H
                        NL:       EQU  0006H
                        PRTMES:   EQU  0015H
                        RDHDR:    EQU  04D8H
                        ;MZ-80A ALTER ABOVE TO 04CFH
                        ERROR:    EQU  0107H
                        ;MZ-80A ALTER ERROR TO 00CFH
                        ;MZ-80K ALTER ERROR TO 01A4H
                        LOAD:     EQU  0121H
                        ;MZ-80A ALTER LOAD TO 00E9H
                        ;MZ-80K ALTER LOAD TO 00F4H
                        ;
   8000 3E16            START:    LD   A,16H
   8002 CD1200                    CALL 0012H
   8005 1627                      LD   D,27H                  ;X
   8007 2100D8                    LD   HL,0D800H              ;X
   800A 01A000                    LD   BC,0A0H                ;X
   800D CDCB80                    CALL DISRTN                 ;X
   8010 1620                      LD   D,20H                  ;X
   8012 01A000                    LD   BC,0A0H                ;X
   8015 CDCB80                    CALL DISRTN                 ;X
   8018 1662                      LD   D,62H                  ;X
   801A 017800                    LD   BC,078H                ;X
   801D CDCB80                    CALL DISRTN                 ;X
   8020 1644                      LD   D,44H
   8022 2100D0                    LD   HL,0D000H
   8025 01A000                    LD   BC,0A0H
   8028 CDCB80                    CALL DISRTN
   802B 1646                      LD   D,46H
   802D 01A000                    LD   BC,0A0H
   8030 CDCB80                    CALL DISRTN
   8033 EB                        EX   DE,HL
   8034 215380                    LD   HL,DISPL
   8037 017800                    LD   BC,120
   803A EDB0                      LDIR
```

```
                   ;
8003C 11F511  LDSHFT:    LD    DE,11F5H                  ;Loader
```

```
                   ;
803C 11F511   LDSHFT:   LD    DE,11F5H                   ;Loader
803F 214A80             LD    HL,LOADER                  ;shifter
8042 010900             LD    BC,9                       ;routine
8045 EDB0               LDIR
                   ;
8047 C3F511   STP:      JP    11F5H
                   ;
804A CDD804   LOADER:   CALL  RDHDR
804D DA0701             JP    C,ERROR
8050 C32101             JP    LOAD
                   ;
                   ;INITIAL SCREEN DISPLAY
                   ;WHILE MAIN PROG. LOADS
8053 4B787878 DISPL:    DB    4BH,78H,78H,78H,78H        ; L
8057 78
8058 78787878           DB    78H,78H,78H,78H,78H,78H    ; I
805C 7878
805E 78787878           DB    78H,78H,78H,78H,78H,78H    ; N
8062 7878
8064 78787878           DB    78H,78H,78H,78H,78H,78H    ; E
8068 7878
806A 78787878           DB    78H,78H,78H,78H,78H,78H
806E 7878
8070 78787878           DB    78H,78H,78H,78H,78H,78H    ; NO.
8074 7878
8076 78787878           DB    78H,78H,78H,78H,4CH        ; 1
807A 4C
807B 79000000           DB    79H,0,0,0,0,0,0,0,0        ;  L
807F 00000000
8083 00
8084 0000190F           DB    0,0,19H,0FH,15H,12H        ;  I
8088 1512
808A 0010120F           DB    0,10H,12H,0FH,07H,12H      ;  N
808E 0712
8090 010D000E           DB    01H,0DH,0,0EH,01H,0DH,05H; E
```

```
8094 010D05
8097 00000000             DB    0,0,0,0,0,0,0,0,0,0,0,79H;  2
809B 00000000
809F 00000079
80A3 6F787878             DB    6FH,78H,78H,78H,78H,78H   ;   L
80A7 7878
80A9 78787878             DB    78H,78H,78H,78H,78H,78H   ;   I
80AD 7878
80AF 73787878             DB    78H,78H,78H,78H,78H,78H   ;   N
80B3 7878
80B5 78787878             DB    78H,78H,78H,78H,78H,78H   ;   E
80B9 7878
80BB 78787878             DB    78H,78H,78H,78H,78H,78H
80BF 7878
80C1 78787878             DB    78H,78H,78H,78H,78H,78H   ;   NO.
80C5 7878
80C7 7878786E             DB    78H,78H,78H,6EH           ;   3
80CB 78        DISRTN:    LD    A,B
80CC B1                   OR    C
80CD C8                   RET   Z
80CE 72                   LD    (HL),D
80CF 23                   INC   HL
80D0 0B                   DEC   BC
80D1 18F8                 JR    DISRTN
                          END
```

MEMORY DISPLAY

As was stated earlier a disassembler is essential for unravelling machine code, and there are some excellent commercial programs available. However if one doesn't yet possess such a utility the following program, although not as comprehensive, will assist in displaying memory contents. Space limits the extent of this memory dump program, but entered correctly it should set one well on the way to increased knowledge of what it is all about.

The program will display memory contents from a given start address to an end address displaying the code for each instruction on separate lines. Unlike commercial disassemblers it will not unfortunately list the operands too. For example if the first eight bytes of a given memory location 809C were:- 21 4D 81 18 21 04 13 1A the program would display as:-

```
809C 214D81
809F 1821
80A1 04
80A2 13
80A3 1A
```

which will ease disassembly.

The other feature is a memory dump display where rows of eight bytes have their contents displayed followed by the ASCII characters for each byte, which obviously simplifies finding screen messages embedded in the code. The program is not made much longer for this memory dump as most subroutines are common to both options, and it is realised that 700 owners have this facility already, but the mini-disassembler should still prove useful.

The display can be halted and restarted by pressing the Space bar and terminated whilst halted by pressing the 'CR' key. Note line 58 tests for code 66H, which is the value in A register for the 'CR' key after calling GETKY at 001BH and not 0DH as one might expect. Most of the equates have been used previously DSPXY (1171H) is used to store the cursor co-ordinates as in the previous program.

For testing under ZEN the MNLOOP equate should be altered to 1203H, and once tested to save the object file one should alter this back to the MNLOOP address for your respective machine as shoewn in chapter 3. The load address has been set to C000H which is safely high up in memory to allow most programs to load beneath it, but it obviously can be moved and saved elsewhere.

To save the object file in order to subsequently load it from Monitor, without ZEN being resident, carry out the following:-

```
WO
START 0C000H
STOP 0C248H
EXEC 0C000H
LOAD 0C000H
NAME MEMORY DUMP
```

New Monitor Routines used:-

096CH Prints display code of reg A
0BB9H Converts ASCII code of reg A to display code.

```
PAGE   1
                           ORG  0C000H
                           LOAD 0C000H
                 ADCN:     EQU  0BB9H
                 DACN:     EQU  0BCEH
                 GETKY:    EQU  001BH
                 WAITKY:   EQU  09B3H
                 PRINT:    EQU  0012H
                 PRTMES:   EQU  0015H
                 DSPXY:    EQU  1171H
                 PRNT3:    EQU  096CH
                 ;ON MZ-80K ALTER ABOVE TO 0970H
                 NL:       EQU  0006H
                 SPACE:    EQU  000CH
                 BELL:     EQU  003EH
                 MNLOOP:   EQU  00ADH
                 ;ON MZ-80K ALTER ABOVE TO 0082H
                 ;ON MZ-80A ALTER TO 0095H
                 ;
                 ;
C000 3E16        START:    LD   A,16H        ;CLEAR SCREEN
C002 CD1200                CALL PRINT
C005 CD0600      ENTKEY:   CALL NL
C008 1104C2                LD   DE,MESS1     ;INPUT MESSAGE
C00B CD1500                CALL PRTMES
C00E 3E3E                  LD   A,3EH        ;CHEVRON CHARACTER
C010 CD1200                CALL PRINT
C013 CDB309                CALL WAITKY       ;WAIT FOR INPUT
C016 CDCE0B                CALL DACN         ;CONVERT TO ASCII CODE
C019 FE21                  CP   21H          ;EXIT PROGRAM CHAR.
C01B CAAD00                JP   Z,MNLOOP
C01E FE44                  CP   44H          ;DISASSEMBLE CHAR.
C020 2809                  JR   Z,INPADD
C022 FE4D                  CP   4DH          ;MEMORY DUMP CHAR.
C024 2805                  JR   Z,INPADD     ;INPUT ADDRESSES
C026 CD3E00      WRONG:    CALL BELL         ;IF OTHER KEY
C029 18DA                  JR   ENTKEY       ;PRESSED, GO BACK
```

```
PAGE   2
C02B 32DFC0   INPADD:   LD   (FLAG),A    ;M OR D IN FLAG
C02E CD1200             CALL PRINT
C031 CD0600             CALL NL
C034 1137C2             LD   DE,MESS2    ;START ADDRESS MESSGE
C037 CD1500             CALL PRTMES
C03A CD81C1             CALL PSG4C       ;INPUT START ADD.
C03D 28E7               JR   Z,WRONG     ;IF CR KEY GO BACK
C03F EB                 EX   DE,HL       ;START ADD. IN DE
C040 D5                 PUSH DE          ;SAVE DE ON STACK
C041 1143C2             LD   DE,MESS3    ;END ADDRESS MESSAGE
C044 CD1500             CALL PRTMES
C047 D1                 POP  DE          ;BRING BACK START ADD.
C048 CD81C1             CALL PSG4C       ;INPUT END ADDRESS
C04B 28D9               JR   Z,WRONG     ;BAD INPUT, GO BACK
C04D CD1B00   SPCDWN:   CALL GETKY
C050 FE20               CP   20H         ;IS SPACE KEY DOWN
C052 2011               JR   NZ,DISASS   ;NO CARRY ON
C054 CD1B00   SPCDWN2:  CALL GETKY       ;SPACE DOWN,WAIT
C057 B7                 OR   A
C058 20FA               JR   NZ,SPCDWN2  ;NO KEY DOWN, GO BACK
C05A CD1B00   QUIT:     CALL GETKY
C05D FE66               CP   66H         ;IS IT CR
C05F 28C5               JR   Z,WRONG     ;YES GO BACK
C061 FE20               CP   20H
C063 20F5               JR   NZ,QUIT     ;WRONG KEY, CHECK AGAIN
C065 CD7BC1   DISASS:   CALL COMPR       ;COMP STRT & END
C068 38BC               JR   C,WRONG     ;END HIGHER THAN STRT
C06A CD0600             CALL NL
C06D 3ADFC0             LD   A,(FLAG)
C070 FE4D               CP   "M"         ;IS IT MEM DUMP
C072 286C               JR   Z,MEMDUMP   ;YES GOTO MEMDUMP
C074 ED5377C1           LD   (STADD),DE
C078 2279C1             LD   (ENDADD),HL
C07B EB                 EX   DE,HL       ;START IN HL
C07C CDDCC1             CALL HEX4
C07F 0601               LD   B,01H       ;SET BYTE COUNTER TO 1
```

```
PAGE   3
C081 1170C1            LD   DE,JRDJT   ;DE POINTS TO JR TABLE
C084 CD2AC1            CALL CHKJR      ;CHECK DE
C087 2841              JR   Z,LIST4    ;ITS IN JR TABLE
C089 79                LD   A,C        ;NOT FOUND CHECK IF IN
C08A EB                EX   DE,HL      ;THE DD,ED,FD GROUP
C08B FEDD              CP   0DDH
C08D 2829              JR   Z,LIST2
C08F FEED              CP   0EDH
C091 2809              JR   Z,LIST1
C093 FEFD              CP   0FDH
C095 2821              JR   Z,LIST2
C097 2146C1            LD   HL,TB8080  ;NOT FOUND, CHECK 8080
C09A 1821              JR   LIST3
C09C 04       LIST1:   INC  B
C09D 13                INC  DE
C09E 1A                LD   A,(DE)
C09F FE46              CP   46H
C0A1 2812              JR   Z,LIST1A
C0A3 FE56              CP   56H
C0A5 280E              JR   Z,LIST1A
C0A7 FE5E              CP   5EH
C0A9 280A              JR   Z,LIST1A
C0AB FE72              CP   72H
C0AD 2806              JR   Z,LIST1A
C0AF FE73              CP   73H
C0B1 2007              JR   NZ,LIST2A  ;IF INSTRUCTION STARTS
C0B3 0604              LD   B,4        ;WITH DD,ED,FD ADD 1
C0B5 B7       LIST1A:  OR   A          ;TO BYTE COUNTER AND
C0B6 1813              JR   LIST5      ;SET ADDR OF NEXT BYTE
C0B8 04       LIST2:   INC  B          ;POSITION WHICH IS
C0B9 13                INC  DE         ;SIGNIFICANT TO OP CODE
C0BA 215BC1   LIST2A:  LD   HL,Z80TB
C0BD CD36C1   LIST3:   CALL CHKZ80
C0C0 FEF0              CP   0F0H
C0C2 2807              JR   Z,LIST5    ;IF NOT FOUND DO NOT
C0C4 79                LD   A,C        ;ALTER BYTE COUNTER
```

```
PAGE   4
C0C5 FE05                CP   05H         ;IF IN FIRST HALF OF
C0C7 3801                JR   C,LIST4     ;TABLE B=B+1
C0C9 04                  INC  B           ;ELSE B=B+2
C0CA 04       LIST4:     INC  B
C0CB CD0C00   LIST5:     CALL SPACE       ;PRINT SPACE &
C0CE ED5B77C1            LD   DE,(STADD)  ;CODE OF INSTRUCTION
C0D2 1A       LIST6:     LD   A,(DE)
C0D3 CDE5C1              CALL HEX2
C0D6 13                  INC  DE
C0D7 10F9                DJNZ LIST6
C0D9 2A79C1              LD   HL,(ENDADD)
C0DC C34DC0              JP   SPCDWN
              ;
              ;
              FLAG:      DS   1           ;M 7 D
              ;
C0E0 EB       MEMDUMP:   EX   DE,HL       ;START ADD IN HL
C0E1 CD0600   MEM1:      CALL NL
C0E4 CDDCC1              CALL HEX4
C0E7 0608                LD   B,8         ;DISPLAY 8 BYTES
C0E9 0E17                LD   C,17H       ;ASCII START AT COL 23
C0EB 7E       MEM2:      LD   A,(HL)      ;PUT BYTE INTO A
C0EC F5                  PUSH AF          ;SAVE IT
C0ED CD21C1              CALL PTSP2H      ;PRINT SPACE & BYTE
C0F0 CD7BC1              CALL COMPR
C0F3 CA26C0              JP   Z,WRONG
C0F6 23                  INC  HL
C0F7 3A7111              LD   A,(DSPXY)   ;CURSOR POSITION
C0FA 81                  ADD  A,C         ;ADD 23 TO CURS. POSN.
C0FB 327111              LD   (DSPXY),A   ;NEW CURSOR POS.
C0FE F1                  POP  AF          ;BRING BACK BYTE
C0FF FE20                CP   20H         ;IF IT IS MORE THAN 19H
C101 3002                JR   NC,CONV     ;THEN CONVERT TO DISPL
C103 3E2E                LD   A,2EH       ;LESS THAN 20 ALTER TO .
C105 CDB90B   CONV:      CALL ADCN        ;CONV ASCII TO DISPL
C108 CD6C09              CALL PRNT3       ;PRINT DISPLAY CODE
```

```
PAGE   5
C10B 3A7111             LD   A,(DSPXY)
C10E 0C                 INC  C
C10F 91                 SUB  C
C110 327111             LD   (DSPXY),A
C113 0D                 DEC  C
C114 0D                 DEC  C
C115 0D                 DEC  C
C116 E5                 PUSH HL
C117 ED52               SBC  HL,DE           ;END YET?
C119 E1                 POP  HL
C11A CA26C0             JP   Z,WRONG         ;YES FINISH
C11D 10CC               DJNZ MEM2
C11F 18C0               JR   MEM1
              ;
              ;PRINT SPACE + 2HEX
              ;
C121 F5       PTSP2H:   PUSH AF
C122 CD0C00             CALL SPACE
C125 F1                 POP  AF
C126 CDE5C1             CALL HEX2
C129 C9                 RET
              ;
              ;
              ;JUMP RELATIVE CHECK
              ;
C12A 4E       CHKJR:    LD   C,(HL)
C12B 1B                 DEC  DE
C12C 13       CHKJR1:   INC  DE
C12D 1A                 LD   A,(DE)
C12E B9                 CP   C
C12F C8                 RET  Z
C130 FEF0               CP   0F0H            ;END OF TABLE?
C132 20F8               JR   NZ,CHKJR1       ;NO GO AGAIN
C134 B7                 OR   A
C135 C9                 RET
              ;
```

```
PAGE    6
                    ;
                    ;8080/Z80 CHECK
                    ;
C136 0E00           CHKZ80:    LD    C,00H
C138 2B                        DEC   HL
C139 23             CHKZ801:   INC   HL
C13A 0C                        INC   C
C13B 7E                        LD    A,(HL)
C13C FEF0                      CP    0F0H
C13E C8                        RET   Z
C13F 1A                        LD    A,(DE)
C140 AE                        XOR   (HL)
C141 23                        INC   HL
C142 A6                        AND   (HL)
C143 20F4                      JR    NZ,CHKZ801
C145 C9                        RET
                    ;
                    ;
                    ;8080 TABLE
                    ;
C146 06             TB8080:    DB    06H
C147 C7                        DB    0C7H
C148 C6                        DB    0C6H
C149 C7                        DB    0C7H
C14A DB                        DB    0DBH
C14B F7                        DB    0F7H
C14C CB                        DB    0CBH
C14D FF                        DB    0FFH
C14E 01                        DB    01H
C14F CF                        DB    0CFH
C150 22                        DB    22H
C151 E7                        DB    0E7H
C152 C2                        DB    0C2H
C153 C7                        DB    0C7H
C154 C4                        DB    0C4H
C155 C7                        DB    0C7H
```

```
PAGE    7
C156 C3                  DB    0C3H
C157 FF                  DB    0FFH
C158 CD                  DB    0CDH
C159 FF                  DB    0FFH
C15A F0                  DB    0F0H
               ;
               ;
               ; Z80 TABLE
               ;
C15B 46        Z80TB:    DB    46H
C15C C7                  DB    0C7H
C15D 70                  DB    70H
C15E F8                  DB    0F8H
C15F 86                  DB    86H
C160 C7                  DB    0C7H
C161 34                  DB    34H
C162 FE                  DB    0FEH
C163 36                  DB    36H
C164 FF                  DB    0FFH
C165 21                  DB    21H
C166 FF                  DB    0FFH
C167 2A                  DB    2AH
C168 FF                  DB    0FFH
C169 22                  DB    22H
C16A FF                  DB    0FFH
C16B CB                  DB    0CBH
C16C FF                  DB    0FFH
C16D 43                  DB    43H
C16E C7                  DB    0C7H
C16F F0                  DB    0F0H
               ;
               ;
               ;Z80 JR & DJNZ TABLE
               ;
C170 10        JRDJT:    DB    10H
C171 18                  DB    18H
```

```
PAGE   8
C172 20                    DB   20H
C173 28                    DB   28H
C174 30                    DB   30H
C175 38                    DB   38H
C176 F0                    DB   0F0H
                 ;
                 ;
                 STADD:    DS   2                 ;START ADD STORED
                 ENDADD:   DS   2                 ;END ADDR STORED
                 ;
                 ;
                 ;COMPARE DE,HL
                 ;
C17B 7C          COMPR:    LD   A,H
C17C 92                    SUB  D
C17D C0                    RET  NZ
C17E 7D                    LD   A,L
C17F 93                    SUB  E
C180 C9                    RET
                 ;
                 ;PRINT SPACE + GET 4 CHARS.
                 ;
C181 F5          PSG4C:    PUSH AF
C182 CD0C00                CALL SPACE
C185 F1                    POP  AF
C186 CD91C1      GET4C:    CALL GET2C
C189 C8                    RET  Z
C18A 67                    LD   H,A
C18B CD91C1                CALL GET2C
C18E C8                    RET  Z
C18F 6F                    LD   L,A
C190 C9                    RET
                 ;
                 ;GET 2 CHARACTERS
                 ;
C191 CDA4C1      GET2C:    CALL GET1C
```

```
PAGE   9
C194 C8                   RET  Z
C195 07                   RLCA
C196 07                   RLCA
C197 07                   RLCA
C198 07                   RLCA
C199 C5                   PUSH BC
C19A 47                   LD   B,A
C19B CDA4C1               CALL GET1C
C19E 2802                 JR   Z,GET2CA
C1A0 B0                   OR   B
C1A1 04                   INC  B
C1A2 C1         GET2CA:   POP  BC
C1A3 C9                   RET
                ;
                ;GET 1 CHARACTER
                ;
C1A4 CDD5C1     GET1C:    CALL GET
C1A7 FE0D                 CP   0DH
C1A9 C8                   RET  Z
C1AA FE66                 CP   66H
C1AC C8                   RET  Z
C1AD F5                   PUSH AF
C1AE FE30                 CP   30H
C1B0 381D                 JR   C,GT3
C1B2 FE3A                 CP   3AH
C1B4 3008                 JR   NC,GT1
C1B6 CD1200               CALL PRINT
C1B9 F1                   POP  AF
C1BA D630                 SUB  30H
C1BC 180E                 JR   GT2
C1BE FE41       GT1:      CP   41H
C1C0 380D                 JR   C,GT3
C1C2 FE47                 CP   47H
C1C4 3009                 JR   NC,GT3
C1C6 CD1200               CALL PRINT
C1C9 F1                   POP  AF
```

```
PAGE  10
C1CA D637                SUB  37H
C1CC FEF0      GT2:      CP   0F0H
C1CE C9                  RET
C1CF F1        GT3:      POP  AF
C1D0 CD3E00              CALL BELL
C1D3 18CF                JR   GET1C
               ;
               ;WAIT FOR KEY
               ;
C1D5 CDB309    GET:      CALL WAITKY
C1D8 CDCE0B              CALL DACN
C1DB C9                  RET
               ;
               ;
C1DC 7C        HEX4:     LD   A,H
C1DD CDE5C1              CALL HEX2
C1E0 7D                  LD   A,L
C1E1 CDE5C1              CALL HEX2
C1E4 C9                  RET
               ;
               ;
C1E5 F5        HEX2:     PUSH AF
C1E6 0F                  RRCA
C1E7 0F                  RRCA
C1E8 0F                  RRCA
C1E9 0F                  RRCA
C1EA E60F                AND  0FH
C1EC CDF6C1              CALL HEX1
C1EF F1                  POP  AF
C1F0 E60F                AND  0FH
C1F2 CDF6C1              CALL HEX1
C1F5 C9                  RET
               ;
               ;
C1F6 FE0A      HEX1:     CP   0AH
C1F8 3006                JR   NC,HXT1
```

```
PAGE  11
C1FA C630                ADD  A,30H
C1FC CD1200   HXT0:      CALL PRINT
C1FF C9                  RET
C200 C637     HXT1:      ADD  A,37H
C202 18F8                JR   HXT0
C204 45B09692 MESS1:     DB   "E0 "
C208 9D20
C20A 284429A6            DB   "(D)isassemble "
C20E A4A1A4A4
C212 92B39AB8
C216 9220
C218 B79D2028            DB   "or (M)emory dump"
C21C 4D2992B3
C220 B79DBD20
C224 9CA5B39E
C228 20202028            DB   "    (!) Monitor",0DH
C22C 2129204D
C230 B7B0A696
C234 B79D0D
C237 53544152 MESS2:     DB   "START ADDR=",0DH
C23B 54204144
C23F 44523D0D
C243 20454E44 MESS3:     DB   " END=",0DH
C247 3D0D
                         END
```

Appendix

HEX to OPCODE Conversion Table

This table is to assist when one knows the Hex value and wishes to know the opcode and the amount of bytes it should be followed by. When one attempts to convert decimal values in Basic DATA statements to Opcodes and Operands be sure to start with the first byte in the routine, else one could get false information.
Taking the second program in this book as an example the first byte has the decimal value of 33, convert this to hex and one will see it is 21hex. Now look in the table below to find what 21 signifies. It is LD HL with the next two bytes signified by aa bb, therefore in program two the following two bytes 0,208 will be the value, in reverse order, to load into HL. These convert to 00, D0 hex, so the first three bytes of program two convert to LD HL,D000. Now continue with the fourth value in the DATA line which is 17 which converts to 11hex. On checking below one will see it signifies LD DE,aabb and must have the next two bytes loaded into DE and so on. If one began converting at the wrong place, say at the third byte, and tried to convert 208 to hex (D0) and then looked in the table below it equals on its own RET NC which would be totally wrong, therefore it is essential to start at the beginning.
In the table nn equals a one byte value in the range 00h to FFh (0 to 255 dec) and bb aa two bytes in the same range.

00	NOP	0C	INC C
01 bb aa	LD BC,aabb	0D	DEC C
02	LD (BC),A	0E nn	LD C,nn
03	INC BC	0F	RRCA
04	INC B	10 nn	DJNZ nn
05	DEC B	11 bb aa	LD DE,aabb
06 nn	LD B,nn	12	LD (DE),A
07	RLCA	13	INC DE
08	EX AF,AF'	14	INC D
09	ADD HL,BC	15	DEC D
0A	LD A,(BC)	16 nn	LD D,nn
0B	DEC BC	17	RLA

18 nn	JR nn	3D	DEC A
19	ADD HL,DE	3E nn	LD A,nn
1A	LD A,(DE)	3F	CCF
1B	DEC DE	40	LD B,B
1C	INC E	41	LD B,C
1D	DEC E	42	LD B,D
1E nn	LD E,nn	43	LD B,E
1F	RRA	44	LD B,H
20 nn	JR NZ,nn	45	LD B,Ln
21 bb aa	LD HL,aabb	46	LD B,(HL)
22 bb aa	LD (aabb),HL	47	LD B,A
23	INC HL	48	LD C,B
24	INC H	49	LD C,C
25	DEC H	4A	LD C,D
26 nn	LD H,nn	4B	LD C,E
27	DAA	4C	LD C,H
28 nn	JR Z,nn	4D	LD C,L
29	ADD HL,HL	4E	LD C,(HL)
2A bb aa	LD HL,(nn)	4F	LD C,A
2B	DEC HL	50	LD D,B
2C	INC L	51	LD D,C
2D	DEC L	52	LD D,D
2E nn	LD L,nn	53	LD D,E
2F	CPL	54	LD D,H
30 nn	JR NC,nn	55	LD D,L
31 bb aa	LD SP,aabb	56	LD D,(HL)
32 bb aa	LD (aabb),A	57	LD D,A
33	INC SP	58	LD E,B
34	INC (HL)	59	LD E,C
35	DEC (HL)	5A	LD E,D
36 nn	LD (HL),nn	5B	LD E,E
37	SCF	5C	LD E,H
38 nn	JR C,nn	5D	LD E,L
39	ADD HL,SP	5E	LD E,(HL)
3A bb aa	LD A,(aabb)	5F	LD E,A
3B	DEC SP	60	LD H,B
3C	INC A	61	LD H,C

62	LD H,D	85	ADD A,L
63	LD H,E	86	ADD A,(HL)
64	LD H,H	87	ADD A,A
65	LD H,L	88	ADC A,B
66	LD H,(HL)	89	ADC A,C
67	LD H,A	8A	ADC A,D
68	LD L,B	8B	ADC A,E
69	LD L,C	8C	ADC A,H
6A	LD L,D	8D	ADC A,L
6B	LD L,E	8E	ADC A,(HL)
6C	LD L,H	8F	ADC A,A
6D	LD L,L	90	SUB B
6E	LD L,(HL)	91	SUB C
6F	LD L,A	92	SUB D
70	LD (HL),B	93	SUB E
71	LD (HL),C	94	SUB H
72	LD (HL),D	95	SUB L
73	LD (HL),E	96	SUB (HL)
74	LD (HL),H	97	SUB A
75	LD (HL),L	98	SBC A,B
76	HALT	99	SBC A,C
77	LD (HL),A	9A	SBC A,D
78	LD A,B	9B	SBC A,E
79	LD A,C	9C	SBC A,H
7A	LD A,D	9D	SBC A,L
7B	LD A,E	9E	SBC A,(HL)
7C	LD A,H	9F	SBC A,A
7D	LD A,L	A0	AND B
7E	LD A,(HL)	A1	AND C
7F	LD A,A	A2	AND D
80	ADD A,B	A3	AND E
81	ADD A,C	A4	AND H
82	ADD A,D	A5	AND L
83	ADD A,E	A6	AND (HL)
84	ADD A,H	A7	AND A

A8	XOR B	CB 00	RLC B
A9	XOR C	CB 01	RLC C
AA	XOR D	CB 02	RLC D
AB	XOR E	CB 03	RLC E
AC	XOR H	CB 04	RLC H
AD	XOR L	CB 05	RLC L
AE	XOR (HL)	CB 06	RLC (HL)
AF	XOR A	CB 07	RLC A
B0	OR B	CB 08	RRC B
B1	OR C	CB 09	RRC C
B2	OR D	CB 0A	RRC D
B3	OR E	CB 0B	RRC E
B4	OR H	CB 0C	RRC H
B5	OR L	CB 0D	RRC L
B6	OR (HL)	CB 0E	RRC (HL)
B7	OR A	CB 0F	RRC A
B8	CP B	CB 10	RL B
B9	CP C	CB 11	RL C
BA	CP D	CB 12	RL D
BB	CP E	CB 13	RL E
BC	CP H	CB 14	RL H
BD	CP L	CB 15	RL L
BE	CP (HL)	CB 16	RL (HL)
BF	CP A	CB 17	RL A
C0	RET NZ	CB 18	RR B
C1	POP BC	CB 19	RR C
C2 bb aa	JP NZ,aabb	CB 1A	RR D
C3 bb aa	JP aabb	CB 1B	RR E
C4 bb aa	CALL NZ,aabb	CB 1C	RR H
C5	PUSH BC	CB 1D	RR L
C6 nn	ADD A,nn	CB 1E	RR (HL)
C7	RST 00	CB 1F	RR A
C8	RET Z	CB 20	SLA B
C9	RET	CB 21	SLA C
CA bb aa	JP Z,aabb	CB 22	SLA D

CB 23	SLA E	CB 46	BIT 0,(HL)
CB 24	SLA H	CB 47	BIT 0,A
CB 25	SLA L	CB 48	BIT 1,B
CB 26	SLA (HL)	CB 49	BIT 1,C
CB 27	SLA A	CB 4A	BIT 1,D
CB 28	SRA B	CB 4B	BIT 1,E
CB 29	SRA C	CB 4C	BIT 1,H
CB 2A	SRA D	CB 4D	BIT 1,L
CB 2B	SRA E	CB 4E	BIT 1,(HL)
CB 2C	SRA H	CB 4F	BIT 1,A
CB 2D	SRA L	CB 50	BIT 2,B
CB 2E	SRA (HL)	CB 51	BIT 2,C
CB 2F	SRA A	CB 52	BIT 2,D
CB 30	SLI B	CB 53	BIT 2,E
CB 31	SLI C	CB 54	BIT 2,H
CB 32	SLI D	CB 55	BIT 2,L
CB 33	SLI E	CB 56	BIT 2,(HL)
CB 34	SLI H	CB 57	BIT 2,A
CB 35	SLI L	CB 58	BIT 3,B
CB 36	SLI (HL)	CB 59	BIT 3,C
CB 37	SLI A	CB 5A	BIT 3,D
CB 38	SRL B	CB 5B	BIT 3,E
CB 39	SRL C	CB 5C	BIT 3,H
CB 3A	SRL D	CB 5D	BIT 3,L
CB 3B	SRL E	CB 5E	BIT 3,(HL)
CB 3C	SRL H	CB 5F	BIT 3,A
CB 3D	SRL L	CB 60	BIT 4,B
CB 3E	SRL (HL)	CB 61	BIT 4,C
CB 3F	SRL A	CB 62	BIT 4,D
CB 40	BIT 0,B	CB 63	BIT 4,E
CB 41	BIT 0,C	CB 64	BIT 4,H
CB 42	BIT 0,D	CB 65	BIT 4,L
CB 43	BIT 0,E	CB 66	BIT 4,(HL)
CB 44	BIT 0,H	CB 67	BIT 4,A
CB 45	BIT 0,L	CB 68	BIT 5,B

CB 69	BIT 5,C	CB 8C	RES 1,H
CB 6A	BIT 5,D	CB 8D	RES 1,L
CB 6B	BIT 5,E	CB 8E	RES 1,(HL)
CB 6C	BIT 5,H	CB 8F	RES 1,A
CB 6D	BIT 5,L	CB 90	RES 2,B
CB 6E	BIT 5,(HL)	CB 91	RES 2,C
CB 6F	BIT 5,A	CB 92	RES 2,D
CB 70	BIT 6,B	CB 93	RES 2,E
CB 71	BIT 6,C	CB 94	RES 2,H
CB 72	BIT 6,D	CB 95	RES 2,L
CB 73	BIT 6,E	CB 96	RES 2,(HL)
CB 74	BIT 6,H	CB 97	RES 2,A
CB 75	BIT 6,L	CB 98	RES 3,B
CB 76	BIT 6,(HL)	CB 99	RES 3,C
CB 77	BIT 6,A	CB 9A	RES 3,D
CB 78	BIT 7,B	CB 9B	RES 3,E
CB 79	BIT 7,C	CB 9C	RES 3,H
CB 7A	BIT 7,D	CB 9D	RES 3,L
CB 7B	BIT 7,E	CB 9E	RES 3,(HL)
CB 7C	BIT 7,H	CB 9F	RES 3,A
CB 7D	BIT 7,L	CB A0	RES 4,B
CB 7E	BIT 7,(HL)	CB A1	RES 4,C
CB 7F	BIT 7,A	CB A2	RES 4,D
CB 80	RES 0,B	CB A3	RES 4,E
CB 81	RES 0,C	CB A4	RES 4,H
CB 82	RES 0,D	CB A5	RES 4,L
CB 83	RES 0,E	CB A6	RES 4,(HL)
CB 84	RES 0,H	CB A7	RES 4,A
CB 85	RES 0,L	CB A8	RES 5,B
CB 86	RES 0,(HL)	CB A9	RES 5,C
CB 87	RES 0,A	CB AA	RES 5,D
CB 88	RES 1,B	CB AB	RES 5,E
CB 89	RES 1,C	CB AC	RES 5,H
CB 8A	RES 1,D	CB AD	RES 5,L
CB 8B	RES 1,E	CB AE	RES 5,(HL)

CB AF	RES 5,A	CB D2	SET 2,D
CB B0	RES 6,B	CB D3	SET 2,E
CB B1	RES 6,C	CB D4	SET 2,H
CB B2	RES 6,D	CB D5	SET 2,L
CB B3	RES 6,E	CB D6	SET 2,(HL)
CB B4	RES 6,H	CB D7	SET 2,A
CB B5	RES 6,L	CB D8	SET 3,B
CB B6	RES 6,(HL)	CB D9	SET 3,C
CB B7	RES 6,A	CB DA	SET 3,D
CB B8	RES 7,B	CB DB	SET 3,E
CB B9	RES 7,C	CB DC	SET 3,H
CB BA	RES 7,D	CB DD	SET 3,L
CB BB	RES 7,E	CB DE	SET 3,(HL)
CB BC	RES 7,H	CB DF	SET 3,A
CB BD	RES 7,L	CB E0	SET 4,B
CB BE	RES 7,(HL)	CB E1	SET 4,C
CB BF	RES 7,A	CB E2	SET 4,D
CB C0	SET 0,B	CB E3	SET 4,E
CB C1	SET 0,C	CB E4	SET 4,H
CB C2	SET 0,D	CB E5	SET 4,L
CB C3	SET 0,E	CB E6	SET 4,(HL)
CB C4	SET 0,H	CB E7	SET 4,A
CB C5	SET 0,L	CB E8	SET 5,B
CB C6	SET 0,(HL)	CB E9	SET 5,C
CB C7	SET 0,A	CB EA	SET 5,D
CB C8	SET 1,B	CB EB	SET 5,E
CB C9	SET 1,C	CB EC	SET 5,H
CB CA	SET 1,D	CB ED	SET 5,L
CB CB	SET 1,E	CB EE	SET 5,(HL)
CB CC	SET 1,H	CB EF	SET 5,A
CB CD	SET 1,L	CB F0	SET 6,B
CB CE	SET 1,(HL)	CB F1	SET 6,C
CB CF	SET 1,A	CB F2	SET 6,D
CB D0	SET 2,B	CB F3	SET 6,E
CB D1	SET 2,C	CB F4	SET 6,H

CB F5	SET 6,L	DD 2B	DEC IX
CB F6	SET 6,(HL)	DD 34 nn	INC (IX+nn)
CB F7	SET 6,A	DD 35 nn	DEC (IX+nn)
CB F8	SET 7,B	DD 36 nn n1	LD (IX+nn),n1
CB F9	SET 7,C	DD 39	ADD IX,SP
CB FA	SET 7,D	DD 46 nn	LD B,(IX+nn)
CB FB	SET 7,E	DD 4E nn	LD C,(IX+nn)
CB FC	SET 7,H	DD 56 nn	LD D,(IX+nn)
CB FD	SET 7,L	DD 5E nn	LD E,(IX+nn)
CB FE	SET 7,(HL)	DD 66 nn	LD H,(IX+nn)
CB FF	SET 7,A	DD 6E nn	LD L,(IX+nn)
CC bb aa	CALL Z,aabb	DD 70 nn	LD (IX+nn),B
CD bb aa	CALL aabb	DD 71 nn	LD (IX+nn),C
CE nn	ADC A,nn	DD 72 nn	LD (IX+nn),D
CF	RST 08	DD 73 nn	LD (IX+nn),E
D0	RET NC	DD 74 nn	LD (IX+nn),H
D1	POP DE	DD 75 nn	LD (IX+nn),L
D2 bb aa	JP NC,aabb	DD 77 nn	LD (IX+nn),A
D3 nn	OUT (nn),A	DD 7E nn	LD A,(IX+nn)
D4 bb aa	CALL NC,aabb	DD 86 nn	ADD A,(IX+nn)
D5	PUSH DE	DD 8E nn	ADC A,(IX+nn)
D6 nn	SUB nn	DD 96 nn	SUB (IX+nn)
D7	RST 10	DD 9E nn	SBC A,(IX+nn)
D8	RET C	DD A6 nn	AND (IX+nn)
D9	EXX	DD AE nn	XOR (IX+nn)
DA bb aa	JP C,aabb	DD B6 nn	OR (IX+nn)
DB nn	IN A,(nn)	DD BE nn	CP (IX+nn)
DC bb aa	CALL C,nn	DD CB nn 06	RLC (IX+nn)
DD 09	ADD IX,BC	DD CB nn 0E	RRC (IX+nn)
DD 19	ADD IX,DE	DD CB nn 16	RL (IX+nn)
DD 21 bb aa	LD IX,aabb	DD CB nn 1E	RR (IX+nn)
DD 22 bb aa	LD (aabb),IX	DD CB nn 26	SLA (IX+nn)
DD 23	INC IX	DD CB nn 2E	SRA (IX+nn)
DD 29	ADD IX,IX	DD CB nn 36	SLI (IX+nn)
DD 2A bb aa	LD IX,(aabb)	DD CB nn 3E	SRL (IX+nn)

DD CB nn 46	BIT 0,(IX+nn)	E4 bb aa	CALL PO,aabb
DD CB nn 4E	BIT 1,(IX+nn)	E5	PUSH HL
DD CB nn 56	BIT 2,(IX+nn)	E6 nn	AND nn
DD CB nn 5E	BIT 3,(IX+nn)	E7	RST 20
DD CB nn 66	BIT 4,(IX+nn)	E8	RET PE
DD CB nn 6E	BIT 5,(IX+nn)	E9	JP (HL)
DD CB nn 76	BIT 6,(IX+nn)	EA bb aa	JP PE,aabb
DD CB nn 7E	BIT 7,(IX+nn)	EB	EX DE,HL
DD CB nn 86	RES 0,(IX+nn)	EC bb aa	CALL PE,aabb
DD CB nn 8E	RES 1,(IX+nn)	ED 40	IN B,(C)
DD CB nn 96	RES 2,(IX+nn)	ED 41	OUT (C),B
DD CB nn 9E	RES 3,(IX+nn)	ED 42	SBC HL,BC
DD CB nn A6	RES 4,(IX+nn)	ED 43 bb aa	LD (aabb),BC
DD CB nn AE	RES 5,(IX+nn)	ED 44	NEG
DD CB nn B6	RES 6,(IX+nn)	ED 45	RETN
DD CB nn BE	RES 7,(IX+nn)	ED 46	IM 0
DD CB nn C6	SET 0,(IX+nn)	ED 47	LD I,A
DD CB nn CE	SET 1,(IX+nn)	ED 48	IN C,(C)
DD CB nn D6	SET 2,(IX+nn)	ED 49	OUT (C),C
DD CB nn DE	SET 3,(IX+nn)	ED 4A	ADC HL,BC
DD CB nn E6	SET 4,(IX+nn)	ED 4B bb aa	LD BC,(aabb)
DD CB nn EE	SET 5,(IX+nn)	ED 4D	RETI
DD CB nn F6	SET 6,(IX+nn)	ED 4F	LD R,A
DD CB nn FE	SET 7,(IX+nn)	ED 50	IN D,(C)
DD E1	POP IX	ED 51	OUT (C),D
DD E3	EX (SP),IX	ED 53 bb aa	LD (aabb),DE
DD E5	PUSH IX	ED 56	IM 1
DD E9	JP (IX)	ED 57	LD A,I
DD F9	LD SP,IX	ED 58	IN E,(C)
DE nn	SBC A,nn	ED 59	OUT (C),E
DF	RST 18	ED 5A	ADC HL,DE
E0	RET PO	ED 5B bb aa	LD DE,(aabb)
E1	POP HL	ED 5E	IM 2
E2 bb aa	JP PO,aabb	ED 5F	LD A,R
E3	EX (SP),HL	ED 60	IN H,(C)

ED 61	OUT (C),H	F3	DI
ED 62	SBC HL,HL	F4 bb aa	CALL P,aabb
ED 67	RRD	F5	PUSH AF
ED 68	IN L,(C)	F6 nn	OR nn
ED 69	OUT (C),L	F7	RST 30
ED 6A	ADC HL,HL	F8	RET M
ED 6F	RLD	F9	LD SP,HL
ED 70	IN F,(C)	FA bb aa	JP M,aabb
ED 72	SBC HL,SP	FB	EI
ED 73 bb aa	LD (aabb),SP	FC bb aa	CALL M,aabb
ED 78	IN A,(C)	FD 09	ADD IY,BC
ED 79	OUT (C),A	FD 19	ADD IY,DE
ED 7A	ADC HL,SP	FD 21 bb aa	LD IY,aabb
ED 7B bb aa	LD SP,(aabb)	FD 22 bb aa	LD (aabb),IY
ED A0	LDI	FD 23	INC IY
ED A1	CPI	FD 29	ADD IY,IY
ED A2	INI	FD 2A bb aa	LD IY,(aabb)
ED A3	OUTI	FD 2B	DEC IY
ED A8	LDD	FD 34 nn	INC (IY+nn)
ED A9	CPD	FD 35 nn	DEC (IY+nn)
ED AA	IND	FD 36 nn n1	LD (IY+nn),n1
ED AB	OUTD	FD 39	ADD IY,SP
ED B0	LDIR	FD 46 nn	LD B,(IY+nn)
ED B1	CPIR	FD 4E nn	LD C,(IY+nn)
ED B2	INIR	FD 56 nn	LD D,(IY+nn)
ED B3	OTIR	FD 5E nn	LD E,(IY+nn)
ED B8	LDDR	FD 66 nn	LD H,(IY+nn)
ED B9	CPDR	FD 6E nn	LD L,(IY+nn)
ED BA	INDR	FD 70 nn	LD (IY+nn),B
ED BB	OTDR	FD 71 nn	LD (IY+nn),C
EE nn	XOR nn	FD 72 nn	LD (IY+nn),D
EF	RST 28	FD 73 nn	LD (IY+nn),E
F0	RET P	FD 74 nn	LD (IY+nn),H
F1	POP AF	FD 75 nn	LD (IY+nn),L
F2 bb aa	JP P,aabb	FD 77 nn	LD (IY+nn),A

FD 7E nn	LD A,(IY+nn)	FD CB nn D6	SET 2,(IY+nn)
FD 86 nn	ADD A,(IY+nn)	FD CB nn DE	SET 3,(IY+nn)
FD 8E nn	ADC A,(IY+nn)	FD CB nn E6	SET 4,(IY+nn)
FD 96 nn	SUB (IY+nn)	FD CB nn EE	SET 5,(IY+nn)
FD 9E nn	SBC A,(IY+nn)	FD CB nn F6	SET 6,(IY+nn)
FD A6 nn	AND (IY+nn)	FD CB nn FE	SET 7,(IY+nn)
FD AE nn	XOR (IY+nn)	FD E1	POP IY
FD B6 nn	OR (IY+nn)	FD E3	EX (SP),IY
FD BE nn	CP (IY+nn)	FD E5	PUSH IY
FD CB nn 06	RLC (IY+nn)	FD E9	JP (IY)
FD CB nn 0E	RRC (IY+nn)	FD F9	LD SP,IY
FD CB nn 16	RL (IY+nn)	FE nn	CP nn
FD CB nn 1E	RR (IY+nn)	FF	RST 38
FD CB nn 26	SLA (IY+nn)		
FD CB nn 2E	SRA (IY+nn)		
FD CB nn 36	SLI (IY+nn)		
FD CB nn 3E	SRL (IY+nn)		
FD CB nn 46	BIT 0,(IY+nn)		
FD CB nn 4E	BIT 1,(IY+nn)		
FD CB nn 56	BIT 2,(IY+nn)		
FD CB nn 5E	BIT 3,(IY+nn)		
FD CB nn 66	BIT 4,(IY+nn)		
FD CB nn 6E	BIT 5,(IY+nn)		
FD CB nn 76	BIT 6,(IY+nn)		
FD CB nn 7E	BIT 7,(IY+nn)		
FD CB nn 86	RES 0,(IY+nn)		
FD CB nn 8E	RES 1,(IY+nn)		
FD CB nn 96	RES 2,(IY+nn)		
FD CB nn 9E	RES 3,(IY+nn)		
FD CB nn A6	RES 4,(IY+nn)		
FD CB nn AE	RES 5,(IY+nn)		
FD CB nn B6	RES 6,(IY+nn)		
FD CB nn BE	RES 7,(IY+nn)		
FD CB nn C6	SET 0,(IY+nn)		
FD CB nn CE	SET 1,(IY+nn)		

Instruction set in Alphabetical order

8E	ADC A,(HL)	DD 39	ADD IX,SP
DD 8E nn	ADC A,(IX+nn)	FD 09	ADD IY,BC
FD 8E nn	ADC A,(IY+nn)	FD 19	ADD IY,DE
8F	ADC A,A	FD 29	ADD IY,IY
88	ADC A,B	FD 39	ADD IY,SP
89	ADC A,C		
8A	ADC A,D	A6	AND (HL)
8B	ADC A,E	DD A6 nn	AND (IX+nn)
8C	ADC A,H	FD A6 nn	AND (IY+nn)
8D	ADC A,L	A7	AND A
CE nn	ADC A,nn	A0	AND B
ED 4A	ADC HL,BC	A1	AND C
ED 5A	ADC HL,DE	A2	AND D
ED 6A	ADC HL,HL	A3	AND E
ED 7A	ADC HL,SP	A4	AND H
		A5	AND L
86	ADD A,(HL)	E6 nn	AND nn
DD 86 nn	ADD A,(IX+nn)		
FD 86 nn	ADD A,(IY+nn)	CB 46	BIT 0,(HL)
87	ADD A,A	DD CB nn 46	BIT 0,(IX+nn)
80	ADD A,B	FD CB nn 46	BIT 0,(IY+nn)
81	ADD A,C	CB 47	BIT 0,A
82	ADD A,D	CB 40	BIT 0,B
83	ADD A,E	CB 41	BIT 0,C
84	ADD A,H	CB 42	BIT 0,D
85	ADD A,L	CB 43	BIT 0,E
C6 nn	ADD A,nn	CB 44	BIT 0,H
09	ADD HL,BC	CB 45	BIT 0,L
19	ADD HL,DE		
29	ADD HL,HL	CB 4E	BIT 1,(HL)
39	ADD HL,SP	DD CB nn 4E	BIT 1,(IX+nn)
DD 09	ADD IX,BC	FD CB nn 4E	BIT 1,(IY+nn)
DD 19	ADD IX,DE	CB 4F	BIT 1,A
DD 29	ADD IX,IX	CB 48	BIT 1,B

CB 49	BIT 1,C
CB 4A	BIT 1,D
CB 4B	BIT 1,E
CB 4C	BIT 1,H
CB 4D	BIT 1,L
CB 56	BIT 2,(HL)
DD CB nn 56	BIT 2,(IX+nn)
FD CB nn 56	BIT 2,(IY+nn)
CB 57	BIT 2,A
CB 50	BIT 2,B
CB 51	BIT 2,C
CB 52	BIT 2,D
CB 53	BIT 2,E
CB 54	BIT 2,H
CB 55	BIT 2,L
CB 5E	BIT 3,(HL)
DD CB nn 5E	BIT 3,(IX+nn)
FD CB nn 5E	BIT 3,(IY+nn)
CB 5F	BIT 3,A
CB 58	BIT 3,B
CB 59	BIT 3,C
CB 5A	BIT 3,D
CB 5B	BIT 3,E
CB 5C	BIT 3,H
CB 5D	BIT 3,L
CB 66	BIT 4,(HL)
DD CB nn 66	BIT 4,(IX+nn)
FD CB nn 66	BIT 4,(IY+nn)
CB 67	BIT 4,A
CB 60	BIT 4,B
CB 61	BIT 4,C
CB 62	BIT 4,D
CB 63	BIT 4,E
CB 64	BIT 4,H
CB 65	BIT 4,L
CB 6E	BIT 5,(HL)
DD CB nn 6E	BIT 5,(IX+nn)
FD CB nn 6E	BIT 5,(IY+nn)
CB 6F	BIT 5,A
CB 68	BIT 5,B
CB 69	BIT 5,C
CB 6A	BIT 5,D
CB 6B	BIT 5,E
CB 6C	BIT 5,H
CB 6D	BIT 5,L
CB 76	BIT 6,(HL)
DD CB nn 76	BIT 6,(IX+nn)
FD CB nn 76	BIT 6,(IY+nn)
CB 77	BIT 6,A
CB 70	BIT 6,B
CB 71	BIT 6,C
CB 72	BIT 6,D
CB 73	BIT 6,E
CB 74	BIT 6,H
CB 75	BIT 6,L
CB 7E	BIT 7,(HL)
DD CB nn 7E	BIT 7,(IX+nn)
FD CB nn 7E	BIT 7,(IY+nn)
CB 7F	BIT 7,A
CB 78	BIT 7,B

CB 79	BIT 7,C	2F	CPL
CB 7A	BIT 7,D		
CB 7B	BIT 7,E	27	DAA
CB 7C	BIT 7,H		
CB 7D	BIT 7,L	35	DEC (HL)
		DD 35 nn	DEC (IX+nn)
DC bb aa	CALL C,aabb	FD 35 nn	DEC (IY+nn)
FC bb aa	CALL M,aabb	3D	DEC A
D4 bb aa	CALL NC,aabb	05	DEC B
CD bb aa	CALL aabb	0B	DEC BC
C4 bb aa	CALL NZ,aabb	0D	DEC C
F4 bb aa	CALL P,aabb	15	DEC D
EC bb aa	CALL PE,aabb	1B	DEC DE
E4 bb aa	CALL PO,aabb	1D	DEC E
CC bb aa	CALL Z,aabb	25	DEC H
		2B	DEC HL
3F	CCF	DD 2B	DEC IX
		FD 2B	DEC IY
BE	CP (HL)	2D	DEC L
DD BE nn	CP (IX+nn)	3B	DEC SP
FD BE nn	CP (IY+nn)		
BF	CP A	F3	DI
B8	CP B		
B9	CP C	10 nn	DJNZ nn
BA	CP D		
BB	CP E	FB	EI
BC	CP H		
BD	CP L	E3	EX (SP),HL
FE nn	CP nn	DD E3	EX (SP),IX
		FD E3	EX (SP),IY
ED A9	CPD	08	EX AF,AF'
ED B9	CPDR	EB	EX DE,HL
ED A1	CPI	D9	EXX
ED B1	CPIR		
		76	HALT

ED 46	IM 0
ED 56	IM 1
ED 5E	IM 2
ED 78	IN A,(C)
DB nn	IN A,(nn)
ED 40	IN B,(C)
ED 48	IN C,(C)
ED 50	IN D,(C)
ED 58	IN E,(C)
ED 70	IN F,(C)
ED 60	IN H,(C)
ED 68	IN L,(C)
34	INC (HL)
DD 34 nn	INC (IX+nn)
FD 34 nn	INC (IY+nn)
3C	INC A
04	INC B
03	INC BC
0C	INC C
14	INC D
13	INC DE
1C	INC E
24	INC H
23	INC HL
DD 23	INC IX
FD 23	INC IY
2C	INC L
33	INC SP
ED AA	IND
ED BA	INDR
ED A2	INI
ED B2	INIR

E9	JP (HL)
DD E9	JP (IX)
FD E9	JP (IY)
DA bb aa	JP C,aabb
FA bb aa	JP M,aabb
D2 bb aa	JP NC,aabb
C3 bb aa	JP aabb
C2 bb aa	JP NZ,aabb
F2 bb aa	JP P,aabb
EA bb aa	JP PE,aabb
E2 bb aa	JP PO,aabb
CA bb aa	JP Z,aabb
38 nn	JR C,nn
18 nn	JR nn
30 nn	JR NC,nn
20 nn	JR NZ,nn
28 nn	JR Z,nn
02	LD (BC),A
12	LD (DE),A
77	LD (HL),A
70	LD (HL),B
71	LD (HL),C
72	LD (HL),D
73	LD (HL),E
74	LD (HL),H
75	LD (HL),L
36 nn	LD (HL),nn
DD 77 nn	LD (IX+nn),A
DD 70 nn	LD (IX+nn),B
DD 71 nn	LD (IX+nn),C
DD 72 nn	LD (IX+nn),D
DD 73 nn	LD (IX+nn),E

DD 74 nn	LD (IX+nn),H
DD 75 nn	LD (IX+nn),L
DD 36 nn n1	LD (IX+nn),n1
FD 77 nn	LD (IY+nn),A
FD 70 nn	LD (IY+nn),B
FD 71 nn	LD (IY+nn),C
FD 72 nn	LD (IY+nn),D
FD 73 nn	LD (IY+nn),E
FD 74 nn	LD (IY+nn),H
FD 75 nn	LD (IY+nn),L
FD 36 nn n1	LD (IY+nn),n1
32 bb aa	LD (aabb),A
ED 43 bb aa	LD (aabb),BC
ED 53 bb aa	LD (aabb),DE
22 bb aa	LD (aabb),HL
DD 22 bb aa	LD (aabb),IX
FD 22 bb aa	LD (aabb),IY
ED 73 bb aa	LD (aabb),SP
0A	LD A,(BC)
1A	LD A,(DE)
7E	LD A,(HL)
DD 7E nn	LD A,(IX+nn)
FD 7E nn	LD A,(IY+nn)
3A bb aa	LD A,(aabb)
7F	LD A,A
78	LD A,B
79	LD A,C
7A	LD A,D
7B	LD A,E
7C	LD A,H
ED 57	LD A,I

7D	LD A,L
3E nn	LD A,nn
ED 5F	LD A,R
46	LD B,(HL)
DD 46 nn	LD B,(IX+nn)
FD 46 nn	LD B,(IY+nn)
47	LD B,A
40	LD B,B
41	LD B,C
42	LD B,D
43	LD B,E
44	LD B,H
45	LD B,L
06 nn	LD B,nn
ED 4B bb aa	LD BC,(aabb)
01 bb aa	LD BC,aabb
4E	LD C,(HL)
DD 4E nn	LD C,(IX+nn)
FD 4E nn	LD C,(IY+nn)
4F	LD C,A
48	LD C,B
49	LD C,C
4A	LD C,D
4B	LD C,E
4C	LD C,H
4D	LD C,L
0E nn	LD C,nn
56	LD D,(HL)
DD 56 nn	LD D,(IX+nn)
FD 56 nn	LD D,(IY+nn)

57	LD D,A
50	LD D,B
51	LD D,C
52	LD D,D
53	LD D,E
LD D,L	
16 nn	LD D,nn
ED 5B bb aa	LD DE,(aabb)
11 bb aa	LD DE,aabb
5E	LD E,(HL)
DD 5E nn	LD E,(IX+nn)
FD 5E nn	LD E,(IY+nn)
5F	LD E,A
58	LD E,B
59	LD E,C
5A	LD E,D
5B	LD E,E
5C	LD E,H
5D	LD E,L
1E nn	LD E,nn
66	LD H,(HL)
DD 66 nn	LD H,(IX+nn)
FD 66 nn	LD H,(IY+nn)
67	LD H,A
60	LD H,B
61	LD H,C
62	LD H,D
63	LD H,E
64	LD H,H
65	LD H,L
26 nn	LD H,nn

2A bb aa	LD HL,(aabb)
21 bb aa	LD HL,aabb
ED 47	LD I,A
DD 21 bb aa	LD IX,aabb
FD 2A bb aa	LD IY,(aabb)
FD 21 bb aa	LD IY,aabb
6E	LD L,(HL)
DD 6E nn	LD L,(IX+nn)
FD 6E nn	LD L,(IY+nn)
6F	LD L,A
68	LD L,B
69	LD L,C
6A	LD L,D
6B	LD L,E
6C	LD L,H
6D	LD L,L
2E nn	LD L,nn
ED 4F	LD R,A
ED 7B bb aa	LD SP,(aabb)
F9	LD SP,HL
DD F9	LD SP,IX
FD F9	LD SP,IY
31 bb aa	LD SP,aabb
ED A8	LDD
ED B8	LDDR
ED A0	LDI
ED B0	LDIR

ED 44	NEG
00	NOP
B6	OR (HL)
DD B6 nn	OR (IX+nn)
FD B6 nn	OR (IY+nn)
B7	OR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
F6 nn	OR nn
ED BB	OTDR
ED B3	OTIR
ED 79	OUT (C),A
ED 41	OUT (C),B
ED 49	OUT (C),C
ED 51	OUT (C),D
ED 59	OUT (C),E
ED 61	OUT (C),H
ED 69	OUT (C),L
D3 nn	OUT (nn),A
ED AB	OUTD
ED A3	OUTI
F1	POP AF
C1	POP BC
D1	POP DE
E1	POP HL

DD E1	POP IX
FD E1	POP IY
F5	PUSH AF
C5	PUSH BC
D5	PUSH DE
E5	PUSH HL
DD E5	PUSH IX
FD E5	PUSH IY
CB 86	RES 0,(HL)
DD CB nn 86	RES 0,(IX+nn)
FD CB nn 86	RES 0,(IX+nn)
CB 87	RES 0,A
CB 80	RES 0,B
CB 81	RES 0,C
CB 82	RES 0,D
CB 83	RES 0,E
CB 84	RES 0,H
CB 85	RES 0,L
CB 8E	RES 1,(HL)
DD CB nn 8E	RES 1,(IX+nn)
FD CB nn 8E	RES 1,(IY+nn)
CB 8F	RES 1,A
CB 88	RES 1,B
CB 89	RES 1,C
CB 8A	RES 1,D
CB 8B	RES 1,E
CB 8C	RES 1,H
CB 8D	RES 1,L
CB 96	RES 2,(HL)
DD CB nn 96	RES 2,(IX+nn)
FD CB nn 96	RES 2,(IY+nn)

CB 97	RES 2,A
CB 90	RES 2,B
CB 91	RES 2,C
CB 92	RES 2,D
CB 93	RES 2,E
CB 94	RES 2,H
CB 95	RES 2,L
CB 9E	RES 3,(HL)
DD CB nn 9E	RES 3,(IX+nn)
FD CB nn 9E	RES 3,(IY+nn)
CB 9F	RES 3,A
CB 98	RES 3,B
CB 99	RES 3,C
CB 9A	RES 3,D
CB 9B	RES 3,E
CB 9C	RES 3,H
CB 9D	RES 3,L
CB A6	RES 4,(HL)
DD CB nn A6	RES 4,(IX+nn)
FD CB nn A6	RES 4,(IY+nn)
CB A7	RES 4,A
CB A0	RES 4,B
CB A1	RES 4,C
CB A2	RES 4,D
CB A3	RES 4,E
CB A4	RES 4,H
CB A5	RES 4,L
CB AE	RES 5,(HL)
DD CB nn AE	RES 5,(IX+nn)
FD CB nn AE	RES 5,(IY+nn)
CB AF	RES 5,A
CB A8	RES 5,B
CB A9	RES 5,C
CB AA	RES 5,D
CB AB	RES 5,E
CB AC	RES 5,H
CB AD	RES 5,L
CB B6	RES 6,(HL)
DD CB nn B6	RES 6,(IX+nn)
FD CB nn B6	RES 6,(IY+nn)
CB B7	RES 6,A
CB B0	RES 6,B
CB B1	RES 6,C
CB B2	RES 6,D
CB B3	RES 6,E
CB B4	RES 6,H
CB B5	RES 6,L
CB BE	RES 7,(HL)
DD CB nn BE	RES 7,(IX+nn)
FD CB nn BE	RES 7,(IY+nn)
CB BF	RES 7,A
CB B8	RES 7,B
CB B9	RES 7,C
CB BA	RES 7,D
CB BB	RES 7,E
CB BC	RES 7,H
CB BD	RES 7,L
C9	RET
D8	RET C
F8	RET M
D0	RET NC
C0	RET NZ
F0	RET P
E8	RET PE

E0	RET PO
C8	RET Z
ED 4D	RETI
ED 45	RETN
CB 16	RL (HL)
DD CB nn 16	RL (IX+nn)
FD CB nn 16	RL (IY+nn)
CB 17	RL A
CB 10	RL B
CB 11	RL C
CB 12	RL D
CB 13	RL E
CB 14	RL H
CB 15	RL L
17	RLA
CB 06	RLC (HL)
DD CB nn 06	RLC (IX+nn)
FD CB nn 06	RLC (IY+nn)
CB 07	RLC A
CB 00	RLC B
CB 01	RLC C
CB 02	RLC D
CB 03	RLC E
CB 04	RLC H
CB 05	RLC L
07	RLCA
ED 6F	RLD
CB 1E	RR (HL)
DD CB nn 1E	RR (IX+nn)
FD CB nn 1E	RR (IY+nn)
CB 1F	RR A
CB 18	RR B
CB 19	RR C
CB 1A	RR D
CB 1B	RR E
CB 1C	RR H
CB 1D	RR L
1F	RRA
CB 0E	RRC (HL)
DD CB nn 0E	RRC (IX+nn)
FD CB nn 0E	RRC (IY+nn)
CB 0F	RRC A
CB 08	RRC B
CB 09	RRC C
CB 0A	RRC D
CB 0B	RRC E
CB 0C	RRC H
CB 0D	RRC L
0F	RRCA
ED 67	RRD
C7	RST 0
CF	RST 8h
D7	RST 10h
DF	RST 18h
E7	RST 20h
EF	RST 28h
F7	RST 30h
FF	RST 38h

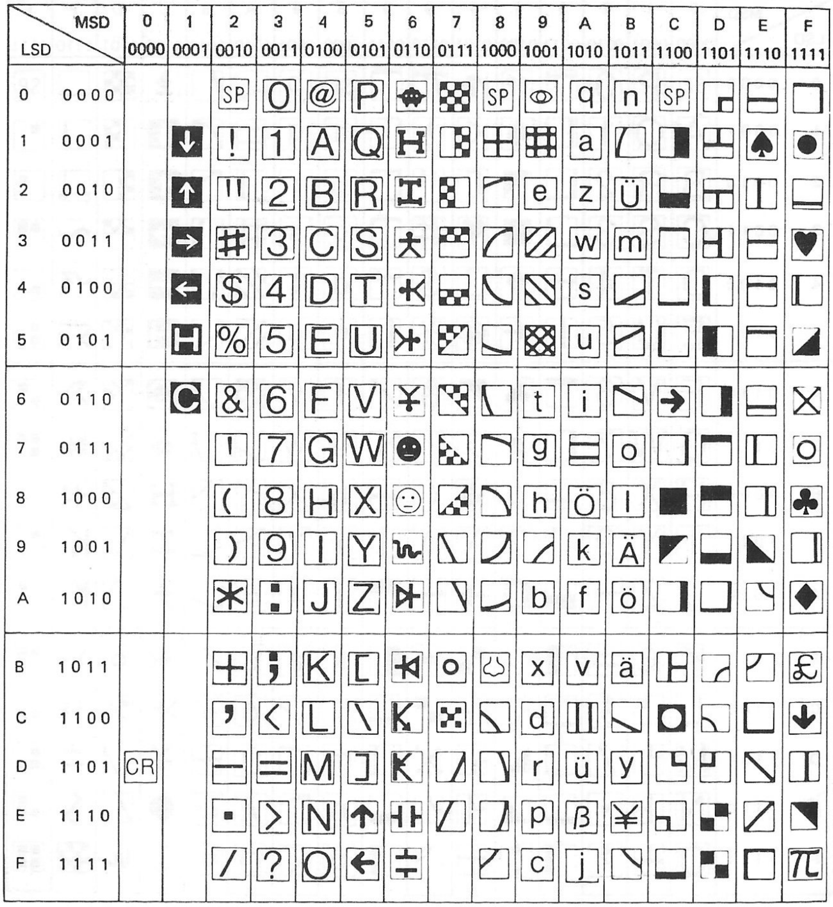
9E	SBC A,(HL)
DD 9E nn	SBC A,(IX+nn)
FD 9E nn	SBC A,(IY+nn)
9F	SBC A,A
98	SBC A,B
99	SBC A,C
9A	SBC A,D
9B	SBC A,E
9C	SBC A,H
9D	SBC A,L
DE nn	SBC A,nn
ED 42	SBC HL,BC
ED 52	SBC HL,DE
ED 62	SBC HL,HL
ED 72	SBC HL,SP
37	SCF
CB C6	SET 0,(HL)
DD CB nn C6	SET 0,(IX+nn)
FD CB nn C6	SET 0,(IY+nn)
CB C7	SET 0,A
CB C0	SET 0,B
CB C1	SET 0,C
CB C2	SET 0,D
CB C3	SET 0,E
CB C4	SET 0,H
CB C5	SET 0,L
CB CE	SET 1,(HL)
DD CB nn CE	SET 1,(IX+nn)
FD CB nn CE	SET 1,(IY+nn)
CB CF	SET 1,A
CB C8	SET 1,B
CB C9	SET 1,C
CB CA	SET 1,D
CB CB	SET 1,E
CB CC	SET 1,H
CB CD	SET 1,L
CB D6	SET 2,(HL)
DD CB nn D6	SET 2,(IX+nn)
FD CB nn D6	SET 2,(IY+nn)
CB D7	SET 2,A
CB D0	SET 2,B
CB D1	SET 2,C
CB D2	SET 2,D
CB D3	SET 2,E
CB D4	SET 2,H
CB D5	SET 2,L
CB DE	SET 3,(HL)
DD CB nn DE	SET 3,(IX+nn)
FD CB nn DE	SET 3,(IY+nn)
CB DF	SET 3,A
CB D8	SET 3,B
CB D9	SET 3,C
CB DA	SET 3,D
CB DB	SET 3,E
CB DC	SET 3,H
CB DD	SET 3,L
CB E6	SET 4,(HL)
DD CB nn E6	SET 4,(IX+nn)
FD CB nn E6	SET 4,(IY+nn)
CB E7	SET 4,A
CB E0	SET 4,B
CB E1	SET 4,C
CB E2	SET 4,D

CB E3	SET 4,E
CB E4	SET 4,H
CB E5	SET 4,L
CB EE	SET 5,(HL)
DD CB nn EE	SET 5,(IX+nn)
FD CB nn EE	SET 5,(IY+nn)
CB EF	SET 5,A
CB E8	SET 5,B
CB E9	SET 5,C
CB EA	SET 5,D
CB EB	SET 5,E
CB EC	SET 5,H
CB ED	SET 5,L
CB F6	SET 6,(HL)
DD CB nn F6	SET 6,(IX+nn)
FD CB nn F6	SET 6,(IY+nn)
CB F7	SET 6,A
CB F0	SET 6,B
CB F1	SET 6,C
CB F2	SET 6,D
CB F3	SET 6,E
CB F4	SET 6,H
CB F5	SET 6,L
CB FE	SET 7,(HL)
DD CB nn FE	SET 7,(IX+nn)
FD CB nn FE	SET 7,(IY+nn)
CB FF	SET 7,A
CB F8	SET 7,B
CB F9	SET 7,C
CB FA	SET 7,D
CB FB	SET 7,E
CB FC	SET 7,H
CB FD	SET 7,L

CB 26	SLA (HL)
DD CB nn 26	SLA (IX+nn)
FD CB nn 26	SLA (IY+nn)
CB 27	SLA A
CB 20	SLA B
CB 21	SLA C
CB 22	SLA D
CB 23	SLA E
CB 24	SLA H
CB 25	SLA L
CB 36	SLI (HL)
DD CB nn 36	SLI (IX+nn)
FD CB nn 36	SLI (IY+nn)
CB 37	SLI A
CB 30	SLI B
CB 31	SLI C
CB 32	SLI D
CB 33	SLI E
CB 34	SLI H
CB 35	SLI L
CB 2E	SRA (HL)
DD CB nn 2E	SRA (IX+nn)
FD CB nn 2E	SRA (IY+nn)
CB 2F	SRA A
CB 28	SRA B
CB 29	SRA C
CB 2A	SRA D
CB 2B	SRA E
CB 2C	SRA H
CB 2D	SRA L
CB 3E	SRL (HL)
DD CB nn 3E	SRL (IX+nn)
FD CB nn 3E	SRL (IY+nn)

CB 3F	SRL A
CB 38	SRL B
CB 39	SRL C
CB 3A	SRL D
CB 3B	SRL E
CB 3C	SRL H
CB 3D	SRL L
96	SUB (HL)
DD 96 nn	SUB (IX+nn)
FD 96 nn	SUB (IY+nn)
97	SUB A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
D6 nn	SUB nn
AE	XOR (HL)
DD AE nn	XOR (IX+nn)
FD AE nn	XOR (IY+nn)
AF	XOR A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
EE nn	XOR nn

ASC II



LSD \ MSD		0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0	0000			SP	0	@	P	✥	▩	SP	◎	q	n	SP	▗	☰	□
1	0001		↓	!	1	A	Q	H	▚	┼	╬	a	╱	▐	┴	♠	●
2	0010		↑	"	2	B	R	I	▞	╭	e	z	Ü	▄	┬	┃	▁
3	0011		→	#	3	C	S	大	▀	╭	▨	w	m	□	┤	═	♥
4	0100		←	$	4	D	T	✱	▄	╰	▧	s	◿	▁	▌	═	▏
5	0101		H	%	5	E	U	✈	▞	╰	⊠	u	◸	▕	▌	▔	◢
6	0110		C	&	6	F	V	¥	▚	╲	t	i	◹	→	▐	═	×
7	0111			'	7	G	W	●	▚	╮	g	≡	o	▕	▔	▏	o
8	1000			(	8	H	X	☺	▞	╲	h	Ö	l	▀	▀	▕	♣
9	1001			)	9	I	Y	∿	╲	╯	╱	k	Ä	◤	▄	◣	▕
A	1010			*	:	J	Z	⊳	╲	╰	b	f	ö	▐	▁	╮	◆
B	1011			+	;	K	[	⊲	o	◡	x	v	ä	┣	╭	╯	£
C	1100			,	<	L	\	K	⁙	╲	d	‖	╲	■	╲	□	↓
D	1101		CR	—	=	M	]	K	╱	╲	r	ü	y	┘	┘	╲	‖
E	1110			.	>	N	↑	╡╞	╱	╱	p	ß	¥	┌	▚	╱	◥
F	1111			/	?	O	←	÷		╱	c	j	╲	▁	▞	□	π

The above table is for the MZ-80K. Any differences are shown:-

	80	8B	90	93	94	BE	C0
MZ-80A/MZ-700	}	^	‾	`	~	{	|

DISPLAY

LSD \ MSD	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 0000	SP	P	0	▔	SP	↑	π	▯	SP	p	╲	╱	⤓	▦	⊥	SP
1 0001	A	Q	1	▏	♠	<	!	▕	a	q	☰	╭	⬇	▦	┤	▪
2 0010	B	R	2	▁	◥	[	"	□	b	r	‖	╮	⬆	▦	┘	▪
3 0011	C	S	3	▕	■	♥	#	▔	c	s	▦	╲	➡	▦	∼	▪▪
4 0100	D	T	4	⊟	◆	]	$	⊟	d	t	▨	╲	⬅	▦	⌐	▪
5 0101	E	U	5	◫	←	@	%	◫	e	u	▧	╰	H	▦	⧗	⁝
6 0110	F	V	6	▀	♣	◤	&	╱	f	v	▩	╯	C	▦	⅀	⁘
7 0111	G	W	7	▌	●	>	'	╲	g	w	◿	╱	▲	▦	⊳	▦
8 1000	H	X	8	▁	o	↓	(	⊟	h	x	╱	╲	H	▦	⊲	▪
9 1001	I	Y	9	▐	?	\	)	◫	i	y	╲	╰	I	\	K	⁚
A 1010	J	Z	−	▄	◘	→	+	▀	j	z	β	╯	大	\	K	⁝
B 1011	K	£	=	▐	╭	▚	*	▌	k	ä	ü	╱	⋉	o	⋋	⁙
C 1100	L	┗	;	▁	◟	▗	▚	▁	l	╱	ö	¥	⋊	⁙	╢	▪▪
D 1101	M	┛	/	▕	◣	┐	X	▐	m	╭	Ü	⊞	¥	╱	÷	▪
E 1110	N	⊢	▪	▁	◢	⊣	╯	▄	n	╮	Ä	♁	☻	╱	∑	▪
F 1111	O	┼	,	▕	⁚	┬	╮	▐	o	╲	Ö	◉	☺	∿	▦	⁘

The above table is for the MZ-80K. Any differences are shown:-

	40	80	A4	A5	BC	BE	BF	E5	F0
MZ-80A/MZ-700	}	|	`	~	{	^	‾	▧	▨

(MZ-80A)
(only)

HEX	DEC	DEC	H	D	D	H	D	D	H	D	D	H	D	D
	*256			*256			*256			*256			*256	
00	00000	0	34	13312	52	68	26624	104	9C	39936	156	D0	53248	208
01	00256	1	35	13568	53	69	26880	105	9D	40192	157	D1	53504	209
02	00512	2	36	13824	54	6A	27136	106	9E	40448	158	D2	53760	210
03	00768	3	37	14080	55	6B	27392	107	9F	40704	159	D3	54016	211
04	01024	4	38	14336	56	6C	27648	108	A0	40960	160	D4	54272	212
05	01280	5	39	14592	57	6D	27904	109	A1	41216	161	D5	54528	213
06	01536	6	3A	14848	58	6E	28160	110	A2	41472	162	D6	54784	214
07	01792	7	3B	15104	59	6F	28416	111	A3	41728	163	D7	55040	215
08	02048	8	3C	15360	60	70	28672	112	A4	41984	164	D8	55296	216
09	02304	9	3D	15616	61	71	28928	113	A5	42240	165	D9	55552	217
0A	02560	10	3E	15872	62	72	29184	114	A6	42496	166	DA	55808	218
0B	02816	11	3F	16128	63	73	29440	115	A7	42752	167	DB	56064	219
0C	03072	12	40	16384	64	74	29696	116	A8	43008	168	DC	56320	220
0D	03328	13	41	16640	65	75	29952	117	A9	43264	169	DD	56576	221
0E	03584	14	42	16896	66	76	30208	118	AA	43520	170	DE	56832	222
0F	03840	15	43	17152	67	77	30464	119	AB	43776	171	DF	57088	223
10	04096	16	44	17408	68	78	30720	120	AC	44032	172	E0	57344	224
11	04352	17	45	17664	69	79	30976	121	AD	44288	173	E1	57600	225
12	04608	18	46	17920	70	7A	31232	122	AE	44544	174	E2	57856	226
13	04864	19	47	18176	71	7B	31488	123	AF	44800	175	E3	58112	227
14	05120	20	48	18432	72	7C	31744	124	B0	45056	176	E4	58368	228
15	05376	21	49	18688	73	7D	32000	125	B1	45312	177	E5	58624	229
16	05632	22	4A	18944	74	7E	32256	126	B2	45568	178	E6	58880	230
17	05888	23	4B	19200	75	7F	32512	127	B3	45824	179	E7	59136	231
18	06144	24	4C	19456	76	80	32768	128	B4	46080	180	E8	59392	232
19	06400	25	4D	19712	77	81	33024	129	B5	46336	181	E9	59648	233
1A	06656	26	4E	19968	78	82	33280	130	B6	46592	182	EA	59904	234
1B	06912	27	4F	20224	79	83	33536	131	B7	46848	183	EB	60160	235
1C	07168	28	50	20480	80	84	33792	132	B8	47104	184	EC	60416	236
1D	07424	29	51	20736	81	85	34048	133	B9	47360	185	ED	60672	237
1E	07680	30	52	20992	82	86	34304	134	BA	47616	186	EE	60928	238
1F	07936	31	53	21248	83	87	34560	135	BB	47872	187	EF	61184	239
20	08192	32	54	21504	84	88	34816	136	BC	48128	188	F0	61440	240
21	08448	33	55	21760	85	89	35072	137	BD	48384	189	F1	61696	241
22	08704	34	56	22016	86	8A	35328	138	BE	48640	190	F2	61952	242
23	08960	35	57	22272	87	8B	35584	139	BF	48896	191	F3	62208	243
24	09216	36	58	22528	88	8C	35840	140	C0	49152	192	F4	62464	244
25	09472	37	59	22784	89	8D	36096	141	C1	49408	193	F5	62720	245
26	09728	38	5A	23040	90	8E	36352	142	C2	49664	194	F6	62976	246
27	09984	39	5B	23296	91	8F	36608	143	C3	49920	195	F7	63232	247
28	10240	40	5C	23552	92	90	36864	144	C4	50176	196	F8	63488	248
29	10496	41	5D	23808	93	91	37120	145	C5	50432	197	F9	63744	249
2A	10752	42	5E	24064	94	92	37376	146	C6	50688	198	FA	64000	250
2B	11008	43	5F	24320	95	93	37632	147	C7	50944	199	FB	64256	251
2C	11264	44	60	24576	96	94	37888	148	C8	51200	200	FC	64512	252
2D	11520	45	61	24832	97	95	38144	149	C9	51456	201	FD	64768	253
2E	11776	46	62	25088	98	96	38400	150	CA	51712	202	FE	65024	254
2F	12032	47	63	25344	99	97	38656	151	CB	51968	203	FF	65280	255
30	12288	48	64	25600	100	98	38912	152	CC	52224	204			
31	12544	49	65	25856	101	99	39168	153	CD	52480	205			
32	12800	50	66	26112	102	9A	39424	154	CE	52736	206			
33	13056	51	67	26368	103	9B	39680	155	CF	52992	207			

In each row the first column is the Hex code.
The second row is the Decimal equivalent multiplied by 256 for calculating the M.S.B.
The third row is for use with the L.S.B.

Index

000277