Machine Language Programming Guide

For 380Z and 480Z

RESEARCH
MACHINES

Guide to Machine Language Programming for the 380Z and 480Z

Release 1, November 1981

Additional copies of this publication may be ordered from Research Machines Limited at the address above. Please ask for 'Guide to Machine Language Programming for the 380Z and 480Z'.

SECOND EDITION

November, 1981

This document is a revision of the 380Z Assembler's Guide written by Colin Opie of the Advisory Unit for Computer Based Education, Hatfield, Hertfordshire, in August 1980.

Research Machines Limited would like to extend their thanks to AUCBE for permission to reprint this manual.

FIRST EDITION	**1st Printing**	**250**	**August 1980**
	2nd Printing	**250**	**October 1980**
	3rd Printing	**250**	**February 1981**
SECOND EDITION	Reprinted for Research Machines Limited		November 1981

PREFACE

This is one of a series of texts for users of the Research Machines 380Z microcomputer system. At least three categories of user exist:

(a) Users who wish to use the microcomputer as a computing aid and who are mainly interested in high level manipulation

(b) Those who will turn also to low level assembly programming

(c) Users who would take an interest in the hardware aspects of a microcomputer system

In this text the emphasis lies for those interested in assembly programming.

The Z80 microprocessor is proving itself to many users as a powerful device. It is true that the Z80, like any other microprocessor, will be succeeded at some time, though at present it does represent a "state-of-the-art" in microcomputer technology.

Research Machines Computer Systems 380Z microcomputer uses the power and versatility of the Z80 extremely well. Many people, including industrial and educational establishments are using this particular system. Owing to the number of different configurations available in this system a very general, though comprehensive, approach has been taken for this text. In this way it should make no difference whether the reader is using a 'single Cassette System' or a 'Dual Floppy Disc with Twin-controlled Cassette' system.

A very general introduction to the machine side of the 380Z is first given. This is intended to provide a ground knowledge for the subsequent sections.

The 'Front Panel' of the 380Z is one of its more useful attributes and the first section deals with this. Prior to any machine programming with the 380Z a good working knowledge of this 'panel' is more than useful. It is true that 'Assemblers' take a lot of hard work out of machine-level programming, but this does not mean that assemblers should be a good starting point. The 'front panel' when used properly is an extremely useful tool in the instruction of machine-level programming.

Section 2 deals with the enormous instruction set of the Z80. Similar types of instructions are grouped together to aid explanations. The mechanism of the instruction is given, the effect on flags shown, and so on. In this way it is intended that the text can be a little more than just an 'instruction reference'.

Two appendices exist which include instruction tables, and program examples.

C.N. OPIE
1980

Acknowledgements:

Much time and effort has been given by various people, notably Peter Andrews, Ken Maynard and Dennis Pitchforth. Unfortunately this edition does not appreciably show their support. Many thanks go to these people and the secretarial staff for all their hard work.

CONTENTS

INTRODUCTION

SECTION 1 - FRONT PANEL

SECTION 2 - Z80 INSTRUCTIONS

APPENDICES

INTRODUCTION

To many people low-level programming has the air of confusion and mystery. Bits, bytes, hexadecimal and masking etc. are just terms which increase the complexity. The situation need not be so desperate!

INTRODUCTION

It is assumed, or at least hoped, that the reader has some knowledge of a high level programming language such as BASIC, FORTRAN, etc. Writing programs at a high level consists of defining a problem and then breaking it down into simple enough steps to be able to program the steps using the computer language. For example to add two numbers and print the result in a high level language we could supply a computer with the following program:

```
LET FIRST_NUMBER := 23
LET SECOND_NUMBER := 47
LET RESULT := FIRST_NUMBER+SECOND_NUMBER
PRINT RESULT
STOP
```

Programming at a low-level is similar but the steps involved need to be much smaller. Instead of working with general names such as 'RESULT' it is necessary to work with 'registers', 'accumulators', and 'memory locations'. It is necessary therefore to know about the architecture of the 'Central Processing Unit' (CPU) or 'Microprocessor Unit' (MPU) so that e.g. register names and properties are known.

In this particular text the emphasis is on how to program the 380Z microcomputer which uses the 'Z80' microprocessor unit. Although the manual aims to enable a user to do this it is advisable to read relevant literature on computer architecture and machine code programming. The main problem with general texts is that it is impossible for them to instruct somebody on how e.g. to put a character onto a screen or out to a printer or in from a keyboard. Normally such routines as character input and output are available through the machine operating system and it is only necessary to know how to set up values for the routine and then how to access the routine. In the case of the 380Z this is done through the 'EMT' instructions and full details of these can be found in the Firmware Reference Manual.

To ease your way in to low-level programming it is suggested that you use the 'front panel' facility of the 380Z. Short programs, or even single instructions can be entered into memory and then executed. In this way you should be able to see literally what the instruction does.

Before a note is made of how instructions look at machine level it would be good to take a brief look at the 'architecture' of the Z80* microprocessor and a few terms which will appear.

*Zilog trademark

THE Z80 CPU

The 380Z uses a member of the 8080 family, the Z80, as its central processor unit (CPU). The Z80 has six general purpose registers in the main register set designated as B, C, D, E, H, and L. There is one accumulator - A, and one flag register - F. This microprocessor unit (MPU) is an 8-bit device and therefore each of the registers occupy an 8-bit byte. Very often the six general purpose registers are used in pairs, i.e. BC, DE and HL, for 16-bit operations. The B register has a special property in as much as it can be used for loop counting. In addition to the above there is an alternate register set designated A', B', C', D', E', F', H', and L' which can be manipulated in exactly the same way as the main register set. There are a set of instruction codes which enable the choice of either the A, F registers or the A', F'

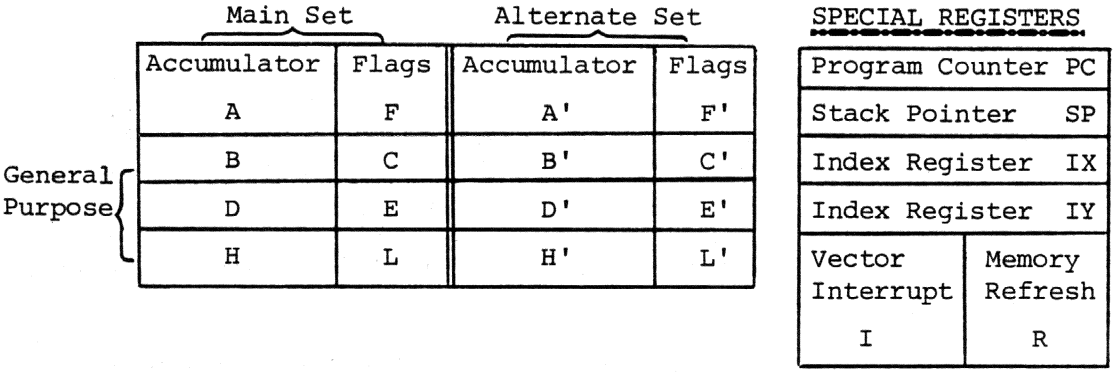


FIG. A) Z80-CPU Register Set

registers, and a choice of either the B, C, D, E, H, L registers or the B', C', D', E', H', L' registers. This gives a combination of four options on the registers to be used.

Also there are two 8-bit special purpose registers and four 16-bit special purpose registers. In practice the two 8-bit special purpose registers, I and R, are arranged as 8 bits and 7 bits respectively. The I or 'interrupt vector' register is used, as the name implies, when using interrupt levels and is not likely to be used by the beginner. The R or 'refresh' register is not normally used by the programmer but is used by the CPU to refresh 'dynamic' RAM (random-access memory).

(i) Program Counter PC

As soon as a byte is fetched from memory the program counter is incremented and so made to point to the next location in store. For a four-byte instruction the program counter will be incremented four times and will therefore automatically point to the next instruction as the current instruction is being executed. Thus the PC register is used as a pointer to the next byte of information to be fetched from memory into the CPU.

(ii) Stack Pointer SP

The stack is used to store or exchange data and works on the 'last-in-first-out' (LIFO) principal. Data is PUSHED onto the stack and PULLED off the stack under program control. As data is put onto the stack the SP register is decremented and as data is retrieved from the stack the pointer is incremented.

(iii) Index Registers IX, IY

These are extremely useful special purpose registers which allow the use of 'indexed' addressing within a program. The register is set to a particular location in memory and by the use of particular instructions can act as a pointer to memory within a range of -128_{10} to $+127_{10}$ bytes from the reference.

BITS AND BYTES

As mentioned earlier the Z80-CPU is an 8-bit device. This means that the CPU will work with 8-bit patterns. The contents of memory locations for example are stored as a pattern of 8 bits, each 'bit' being either a logical 1 or a logical Ø. When 8 bits are taken as a unit like this they are commonly called a 'byte'. Another term commonly found in use is the 'word' length. The word length of a CPU is the length in bits of the 'patterns' it usually deals with. So the Z80-CPU has a word length of 8 bits, which is 1 byte.

To add to the confusion, 1Ø24 bytes of information give 1Kbytes, (note: not 1ØØØ!). The length of a memory is usually measured in kilobytes.

HEXADECIMAL

As the name suggests, this is base sixteen arithmetic. The numbers from Ø to fifteen are represented by:

Ø 1 2 3 4 5 6 7 8 9 A B C D E F

In binary the number of bits needed to specify 16 variations is 4, in other words a pattern of four bits can be represented by the appropriate HEX value. It follows then that two HEX digits are required to specify an 8-bit pattern, four HEX digits a 16-bit pattern and so on. Where doubt may arise when reading and writing HEX numbers, they should always start with a digit and end with H, e.g.

5H 5CH ØAFH

MEMORY CONTENTS AND ADDRESSES

It will be noticed that any of the special purpose registers which store pointers to memory (i.e. addresses), e.g. PC register, are 16-bit registers. Suppose that the addresses were only 8 bits long like the memory contents. This means that the highest memory address would be ØFFH = 2^8 - 1 = 255 (decimal); not very high. To enable the MPU to address an adequate amount of memory, 16-bit addresses are used. This

means that the highest address (without paging techniques) is:

$$\text{ØFFFFH} = 2^{16} - 1 = 65\ 535 \text{ (decimal)}.$$

This provides then the equivalent of 64Kbytes of memory. Much better.

Digressing a little, if the contents of one memory location was 3EH (i.e. 3E in hex.), it may not be clear whether this is an instruction code or data, and if data what kind of data. For simplicity suppose that if it were data then this would be the ASCII code in hex for the appropriate character. Above it is seen that with 8 bits, 256_{10} patterns are possible. The ASCII set only requires 128_{10} possibilities, thus this can be accommodated easily. With 8 bits it is also possible to provide 256_{10} different instruction codes straight off for the CPU. If two-byte instruction codes are implemented the choice is increased by a factor of 256_{10}. If now two address bytes are used on top of the selection code the number of possibilities available far exceeds the range required. For these reasons alone a data word size of 1 byte (8 bits) is perfectly adequate. In practice the instructions occupy from one to four bytes. Owing to the large choice available many of the instructions are in the 'implied' mode. This means that the source and destination of the data is implied by the instruction code, e.g.

CODE (HEX)	MNEMONIC	DESCRIPTION
A2	AND D	Take the 8-bit pattern in the A register and do a bit by bit logical AND with the 8-bit pattern in the D register and put the result into the A register.

INSTRUCTIONS

The Z80 microprocessor has 158 different instructions in its repertoire, and an additional two pseudo-instructions are implemented via the 380Z monitor EMTs. Both 8-bit and 16-bit arithmetic is possible. With 8-bit arithmetic, one operand and the result is always in the accumulator. The arithmetic operations available with 8-bits in this category are addition and subtraction only, multiplication and division must be done by software routines. There are, however, two special arithmetic operations which can be used on any register, these being 'increment' and 'decrement'. Boolean operations are also available in this category, these being, AND, OR, EXCLUSIVE-OR, NEGATE, and COMPLEMENT. Note that 'negate' provides the two's complement and 'complement' provides the one's complement - both operations being performed on the accumulator A.

Only limited arithmetic operations are permitted with 16-bits and no boolean operations are available. Unlike adding and subtracting with or without the carry in 8-bit arithmetic, 16-bit arithmetic does not give the option of subtraction without the carry, i.e.

SIZE	MNEMONICS
8-bit	ADD, ADC, SUB, SBC
16-bit	ADD, ADC, SBC

Note that 'increment' and 'decrement' are permitted with 16-bit registers.

Other instructions are concerned with 8-bit loads, 16-bit loads, Jumps -Calls/Returns, and Rotates/Shifts, and other more specialised operations.

FLAG REGISTER

As mentioned previously, the Z80-CPU used in the 380Z system, contains an 8-bit flag register. All the flags available in the CPU can be used when programming or debugging at assembly level with the 380Z. These flags are set in the MPU when most arithmetic and logical operations are performed. They are not usually affected by load instructions except in the more rare instructions such as LD A,R and LDIR. Note that any instruction which does not affect the flags may be interspersed freely between the setting of a flag and inspecting it.

Six of the eight bits in the flag register (F) are used. The 'N' and 'H' flags are used in Binary-Coded-Decimal (BCD) arithmetic and are not likely to be used by the beginner.

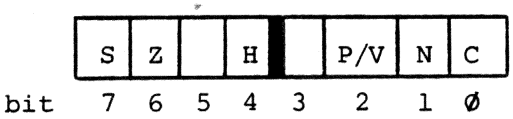


FIG.B) FLAG REGISTER BIT REPRESENTATION

The remaining flags are:

Sign - set if bit 7 of the result is set (i.e. the number is negative)
Zero - set if the result is zero
P/V - set if the parity of a logical operation is even, or
set if the result of arithmetic causes overflow
Carry - set if the result of arithmetic or logic causes a bit
to flow out of the accumulator

There are only two instructions specifically provided to manipulate the flags:

SCF - set the carry flag
CCF - complement the carry flag

In addition, the following instructions are useful:

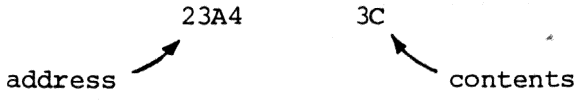
AND A - clears the carry flag but does not affect 'A'
XOR A - sets 'A' to zero and clears all but the P/V flag
and the Z flag.

Section 2 of this text covers the instruction set of the Z80 in detail and indicates how, and by what instructions, the flags are affected. The table below gives the 'flag notation' which will be used:

FLAG NOTATION	MEANING
•	Flag not affected
0	Flag reset
1	Flag set
X	Flag unknown
↕	Flag is affected according to the result of the operation.
IFF	Content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

PROGRAMS

Computers work with patterns of 0s and 1s The Z80 MPU is an eight bit device and therefore works essentially with patterns 8 bits wide. The program counter collects the contents of memory locations and the MPU operates accordingly on the pattern collected. With this in mind suppose we have the following contents in memory:



The pattern '3C' could be data or it could be an instruction. It is important to realise this when programming at machine level. If the program counter is set in the middle of data (e.g. a series of characters representing a message) and then the Z80 allowed to execute the code then the characters will be interpreted as instructions and anything could happen.

So far the patterns of 0s and 1s have been represented by hexadecimal digits, mainly to improve the readability (and save on space when writing the code on paper!). Every instruction to the Z80 MPU has a particular pattern and this could be up to 4 bytes long. This means that the length of a pattern is variable (i.e. 1 to 4 bytes). In the above example the pattern is only one byte long and if it is an instruction then it corresponds to 'take the pattern in the accumulator, treat it as a binary number , and increment that number by 1'. Obviously that last description is a bit long and therefore 'mnemonics' are used to represent the action. In this case it is 'INC A' (INCrement Accumulator). When programming using the 'front panel' it is necessary to enter the hexadecimal representations. When an 'Assembler' is used it is possible to write the program using the mnemonics, which are even easier to read and follow, and the Assembler will work out the necessary machine code.

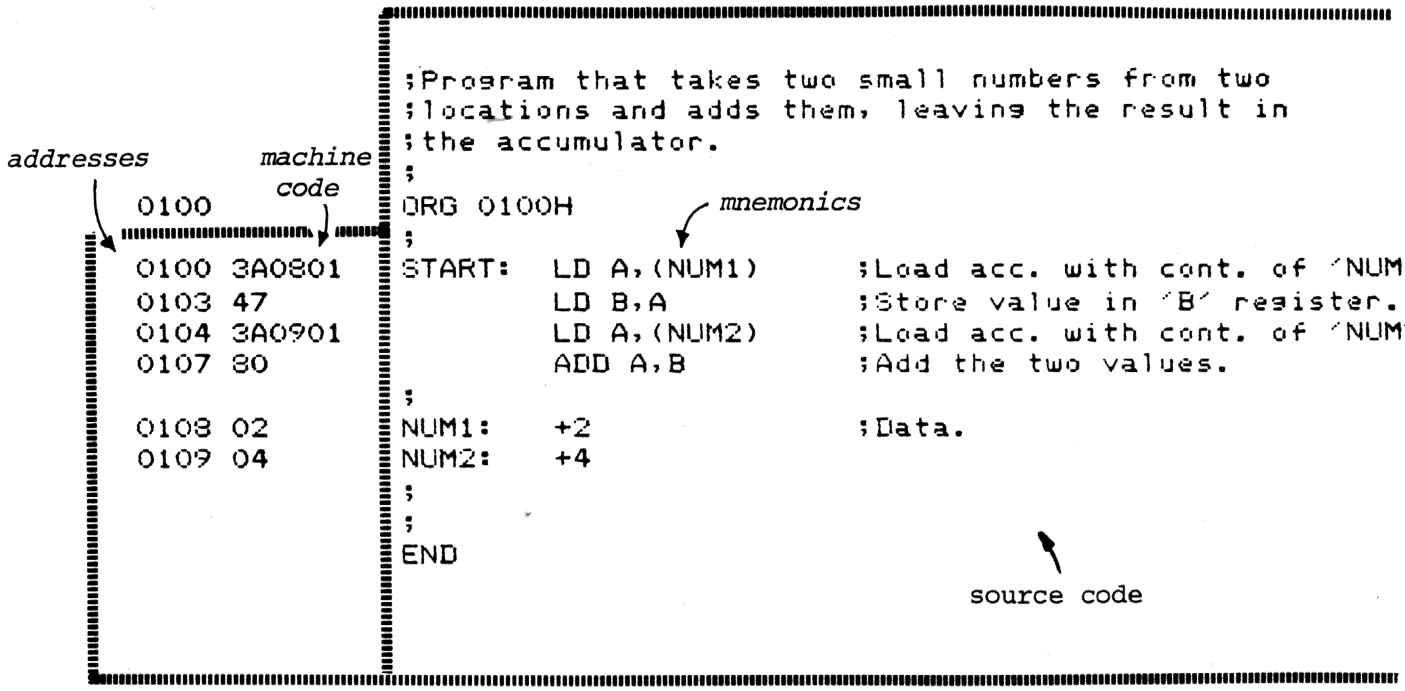
Just remember that there are two ways of getting the Program into the 380Z:

i) machine code - hexadecimal representation,

ii) assembler code - mnemonic representation.

Much more will be said about the Assembler and assembly code later on but it may be advantageous to have a look at a very short program to see how things are related.

Take a look at the following code:




```
                    ;Program that takes two small numbers from two
                    ;locations and adds them, leaving the result in
                    ;the accumulator.
                    ;
                    ORG 0100H
                    ;
0100 3A0801         START:  LD A,(NUM1)     ;Load acc. with cont. of 'NUM1'.
0103 47                     LD B,A          ;Store value in 'B' register.
0104 3A0901                 LD A,(NUM2)     ;Load acc. with cont. of 'NUM2'.
0107 80                     ADD A,B         ;Add the two values.
                    ;
0108 02             NUM1:   +2              ;Data.
0109 04             NUM2:   +4
                    ;
                    ;
                    END
```

NOTES:

i) 'ORG' is merely an instruction to the assembler to say 'start assembling the code at this memory address'. Note that the first address is therefore at Ø1ØØH.

ii) The 380Z is an eight bit computer and therefore each memory location can only store an eight bit code. The first instruction takes three bytes and therefore uses three locations. Owing to this the second instruction starts in the fourth location from the start. This means that if the program was to be entered using the 'Front Panel' the following would be stored:

Address	Contents
Ø1ØØ	3A
Ø1Ø1	Ø8
Ø1Ø2	Ø1
Ø1Ø3	47
Ø1Ø4	3A
Ø1Ø5	Ø9
Ø1Ø6	Ø1
Ø1Ø7	8Ø
Ø1Ø8	Ø2
Ø1Ø9	Ø4

iii) A peculiarity will be seen about the way addresses are stored for instructions. Take 'LD A,(NUM1)' as an example. The instruction has three bytes, one for the instruction itself (3A) and two for the address (NUM1). Now 'NUM1'=location(Ø1Ø8H) but in the instruction the 'high' byte and the 'low' byte are swapped over. This is always the case with Z80 code and must be carefully adhered to when using the 'Front Panel' to enter programs. When the assembler works out the machine code from the mnemonics the swop is automatically carried out and so mistakes cannot be so easily made. Note also that this means the following is also true:

LD A,(12A8H) ⇒ 3A A8 12

iv) There is no way of stopping this program if it were allowed to run freely. The program could only be run by using the 'single step' facility (see later) of the 'Front Panel' and watching the effects of the instructions.

v) The 'END' instruction is nothing to do with the program itself but is another directive to the assembler to say that 'this is the end of the code'.

vi) 'Reading' from a memory location or an MPU register does not destroy the contents of that bit of storage. For example the value of +2 will not disappear from location Ø1Ø8H by putting it into the accumulator. 'Reading' from one location to another is really 'copying'.

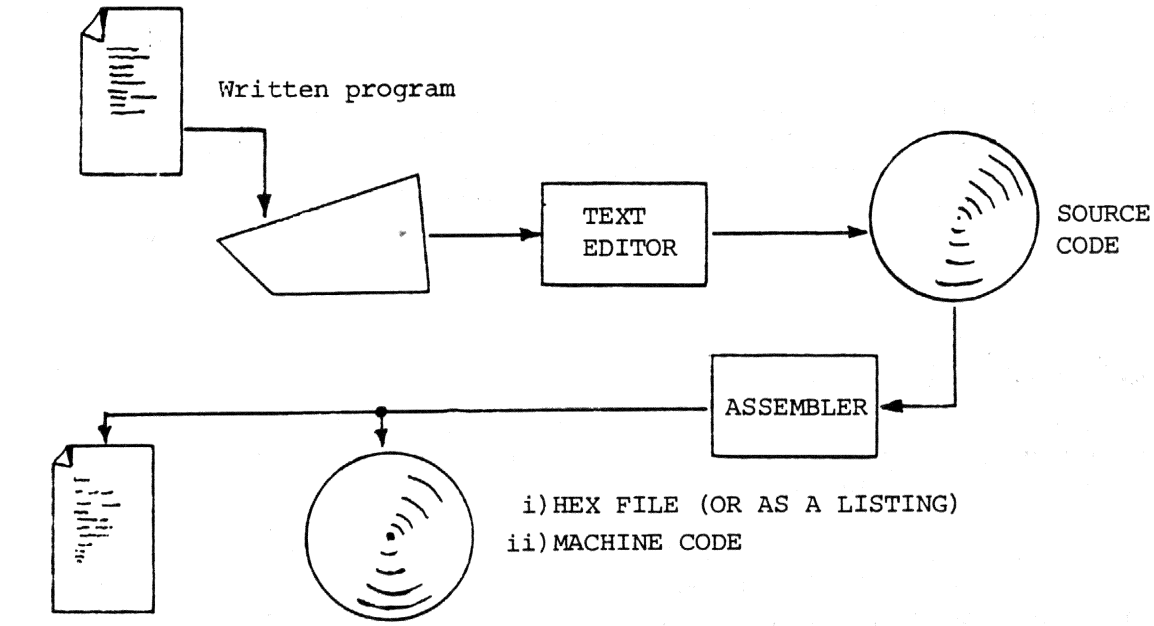
PROGRAM LAYOUT

During the early stages of machine level programming the importance of layout will not appear so relevant because the programs will not be very long. Once the essentials are grasped however the Assembler will start to be used and the programs will become more detailed. When this happens layout is of the utmost importance. There as no hard and fast rules about what should go where but one thing is for sure - absolutely lace the code with comments. An instruction may be placed in the code for one of a number of reasons. For example, 'XOR A' (exclusively-OR accumulator with itself) will unset the carry flag and set the contents of the accumulator to zero, amongst other things. It is important that a comment exists to say what effect is primarily desired from the instruction.

Present documentation by RML on TXED,ZAS/M should be used in conjunction with this manual. It will be helpful to take a closer look at the overall operation of assembler work, and to this end consider again the figure on p.9.

The small block on the left is the machine code and is the form in which the program would be entered using say the front panel. This machine code is produced by the 'Assembler' which uses the 'source code' as its input. The source code is all that which is inside the large block on the right. Once the program or program segment has been written it can be entered into the computer using the RML Text Editor (TXED), and this can then be saved on tape or disc as a source code file. Again this is just the information in the block on the right. The RML Assembler (ZAS/ZASM) can then use this file to produce two new files, a 'Listing file' and a 'Hex file'. A listing file, if printed, would look like the above two blocks joined together. A 'Hex file' is basically just a coded form of the machine code, and can be printed.

So far then we have the following situation:



Obviously the files can be produced on a cassette tape or a disc depending on what kind of system is being used. Note especially that a distinction has been made between the hex file and the machine code. First take a look at the cassette assembler. In this assembler a hex file can be stored or listed and is just a character representation of the machine code. This form is only really useful for producing machine code listings or for sending a copy of the machine code between computers over transmission lines. It is possible as will be seen to get the assembler to produce a machine code file. This is not 'listable', at least not in the sense that the listing would be intelligible. It is the machine runnable form and can be loaded and run in the same way as BASIC or TXED. The

difference is important to realise. The disc version of the assembler (ZASM) is slightly different. In this case it is not possible to get the assembler to produce the machine code file, only the hex file. A utility program called 'LOAD' is then used which takes the hex file as its input and produces a machine code file. Note that with all disc machine-runnable files the 'extension' of this file will be '.COM'.

HOW TO PROGRESS - A SUGGESTION

Having got this far it should be possible to start looking at the instruction set of the Z80 MPU, keeping in mind the standard Zilog mnemonics for the codes. Chapter 2 deals with the standard Z80 instructions. In order to write useful programs at machine level it will also be necessary to understand the special 380Z monitor routine instructions (e.g. for input/output). Details of these can be found in the Firmware Reference Manual. Before much more is done however it may be wise to first come to grips with using the 380Z 'Front Panel' facility. This is an essential and powerful tool when programming the 380Z at machine level. Learning even a subset of the features will be a major advantage.

Once familiarity is achieved with the front panel a close look can be made at the programming side. Appendix 2 contains documented examples and these can be used to gradually get to know the instruction set. Not every Z80 instruction is covered but hopefully enough is included to enable the reader to explore the other instructions as and when it becomes necessary. The chapters on the instruction set etc. can be consulted if a more detailed look at the particular instruction types is desired.

As familiarity increases the Assembler will (or should!) start to be used, probably (and advisably) using the Text Editor to produce the source program.

It is hoped that the Appendices will provide quick reference guides to help when writing machine level programs of your own.

SECTION 1

FRONT PANEL

Other than working with the 380Z at a high level, e.g. using BASIC, there are two other modes possible. One is the 'monitor' mode and the other, the 'front panel' mode. When the 380Z is first switched on it is automatically in monitor mode, where the command prompt is a right-arrow (→). This indicates that the user is communicating with the cassette operating system. With discs the bootstrap (B) monitor command can be executed which, provided the 'Operating System' disc is in unit 'A', will load the disc operating system. This will output a short message followed by a new prompt: A>. The 'A' before the arrow indicates the disc unit which will be accessed by default. In either of the above states the front panel mode will be entered on receipt of a 'CTRL-F', and will appear similar to:

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7BFA 32 6F C0 FF 01 0E 0F 80
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 AC E6
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  06FF SZ H VNC     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                       ↑
E6A0 C9    E6A8 C9    E6B0 0C    E6B8 0C
E6A1 F7    E6A9 CD    E6B1 FF    E6B9 FF
E6A2 21    E6AA AD    E6B2 B7    E6BA F1
E6A3 FE    E6AB E0    E6B3 28    E6BB FE
E6A4 06   >E6AC 18<   E6B4 05    E6BC 01
E6A5 28    E6AD F3    E6B5 3E    E6BD C0
E6A6 02    E6AE F5    E6B6 17    E6BE 3A
E6A7 B7    E6AF 3A    E6B7 32    E6BF 0C
```

FIG 1.1 380Z Front Panel

This 'front panel' is software written and controlled, providing an excellent mechanism for the input and debugging of machine level programs. Commands and delimiters are combined to form capabilities for data input, single stepping, block memory shifts, flag checking, register updating and so on. Useful utilities also exist for pattern searching and hexadecimal calculations.

Note that while in this mode the scroller only operates on the bottom four lines, and that the command prompt is now an exclamation mark (!).

SUMMARY OF FRONT PANEL COMMANDS

COMMAND	ACTION
(Memory Pointer)	
M	Set memory pointer to given address
I	Set memory pointer from memory contents (absolute)
R	Set memory pointer from memory contents (relative)
U	Set memory pointer from Program Counter (PC)
(Memory Pointer Delimiters)	
SPACE	Remain at present location
CARRIAGE RETURN	Increment memory pointer
-	Decrement memory pointer
/	Move back memory pointer by eight
LINE FEED	Advance memory pointer by eight
(Register Pointer)	
.	Increment Register Pointer
<	Increment I/O Pointer
>	Decrement I/O Pointer
,	Set I/O port to given value
(Utilities)	
X	Switch Register Display to Alternate set
P	Fill and Test Memory between limits
S	Shift Memory Content
G	Get first occurrence of specified pattern of bytes
N	Find next occurrence of pattern
H	Hexadecimal Calculator
Z	Execute single instruction; 'single step'.
K	Continue program execution
J	Set Program Counter to specified address and continue execution.
(Exit)	
CTRL-B	Exit to COS monitor
CTRL-C	Exit to CP/M

DISPLAY DESCRIPTION

There are three main areas for the panel, as shown below. At the top is a block of memory relating to the register. In the middle is a line display for the I/O ports, and at the bottom is a block of 32 locations plus associated contents of actual memory.

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7BFA 32 6F C0 FF 01 0E 0F 80
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 AC E6
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  06FF SZ H VNC     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                       ↑
E6A0 C9    E6A8 C9    E6B0 0C    E6B8 0C
E6A1 F7    E6A9 CD    E6B1 FF    E6B9 FF
E6A2 21    E6AA AD    E6B2 B7    E6BA F1
E6A3 FE    E6AB E0    E6B3 28    E6BB FE
E6A4 06   >E6AC 18<   E6B4 05    E6BC 01
E6A5 28    E6AD F3    E6B5 3E    E6BD C0
E6A6 02    E6AE F5    E6B6 17    E6BE 3A
E6A7 B7    E6AF 3A    E6B7 32    E6BF 0C
```

FIG 1.2 - 380Z Front Panel

Consider first the top region, the register display area. The registers are denoted by their standard Zilog mnemonics: PC, SP, IY, IX, HL, DE, BC and AF; referring to the Program Counter, Stack Pointer, two Index Registers, three 16-bit registers, Accumulator and Flag Register. Immediately to the right of these mnemonics are four-digit hexadecimal numbers. These numbers display the contents of the registers. To the right again, of all but the A and F registers, is a row of eight two-digit hexadecimal numbers. These numbers represent the memory contents as addressed by the contents of the registers. The register pointer, at the foot of the fourth column from the left, points to the actual memory contents as addressed by the register. To the left of this byte are four bytes of memory directly below the register content, and to the right are three bytes of memory directly above the register content.

In the case of the 'AF' row this would not be very meaningful and so the content of the flag register is displayed using the letters: S, Z, H, V, N, and C to indicate the presence of any of these flags. For example, the presence of the C means that the Carry flag has been set. Note that these letters are the standard symbols used by Zilog, with the letter V acting for the Zilog P/V symbol.

Consider now the middle region, the I/O port display area. This line display is associated with the upper register region in as much as the register pointer to the left of the mnemonics, initially pointing

to the Program Counter, can be made to point to this row.

The lower part of the display, the memory location display, is completely separate from the above two regions. It consists of a block of 32 memory location addresses together with their associated contents. Approximately half way down the second block of eight are the memory pointers. When modifying or inserting memory content it is at the location pointed to by these pointers that the memory content can be changed.

MEMORY POINTER COMMANDS (M, I, R, U)

These four commands provide a mechanism for aligning the memory pointer on the front panel to specific locations.

1.3.1 M

Format: e.g. !M > 21C4 ↵

Typing M as a command to the front panel prompt (!) will result in the command being echoed plus a prompt (>) for a hexadecimal value. This value should be a four-digit address. If less than four digits are entered then leading zeroes will be assumed:

e.g. 1AF = Ø1AF

If more than four digits are entered then the digits displayed will be shifted to the left displaying the last four digits entered, this being the address used for the move. (On some machines you may have to DELETE).

On entering the address and pressing the RETURN key the memory location display will adjust itself so that the memory pointer is aligned with the location specified.

```
>PC  0203 00 C3 09 02 CD E9 03 C3
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 A2 13
 BC  0000 03 02 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑
1396 F5    139E 3E    13A6 C1    13AE FA
1397 1E    139F 2C    13A7 08    13AF C9
1398 00    13A0 CD    13A8 4F    13B0 EB
1399 CD    13A1 2D    13A9 ED    13B1 3E
139A 0F   >13A2 05<   13AA 78    13B2 29
139B 09    13A3 CD    13AB AB    13B3 CD
139C 28    13A4 C9    13AC A0    13B4 2D
139D 08    13A5 13    13AD 28    13B5 05
```

```
>PC  0203 00 C3 09 02 CD E9 03 C3
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 AC E6
 BC  0000 03 02 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑
E6A0 C9    E6A8 C9    E6B0 0C    E6B8 0C
E6A1 F7    E6A9 CD    E6B1 FF    E6B9 FF
E6A2 21    E6AA AD    E6B2 B7    E6BA F1
E6A3 FE    E6AB E0    E6B3 28    E6BB FE
E6A4 06   >E6AC 18<   E6B4 05    E6BC 01
E6A5 28    E6AD F3    E6B5 3E    E6BD C0
E6A6 02    E6AE F5    E6B6 17    E6BE 3A
E6A7 B7    E6AF 3A    E6B7 32    E6BF 0C
```

FIG. 1.3 - EFFECT of 'M' COMMAND

The example above (Fig. 1.3) shows the effect of the command: M > E6AC, on the original display.

1.3.2 I

Format: ! I

This command sets the memory pointer to the address as specified by the memory contents presently pointed to. Note that it is only necessary to enter the command in response to the prompt (!), NO RETURN is required.

Suppose the memory display around the pointer is as follows:

```
   EØ4A  F5
   EØ4B  CD
→  EØ4C  56 ←
   EØ4D  EØ
   EØ4E  D5
```

On the 'I' command the byte pointed to by the memory pointer and the following byte (56,EØ) will be taken as the absolute address to move to, in standard machine format which is LSB first followed by MSB; which in this case would give an address of EØ56. The memory display around the pointer would then become:

```
   EØ54  E3
   EØ55  C9
→  EØ56  D5 ←
   EØ57  21
   EØ58  Ø8
```

1.3.3 R

Format: ! R

This command is similar in operation to the 'I' command and sets the memory pointer to the relative address as specified by the memory contents presently pointed to. The result of the relative move will be exactly the same as that under normal program execution. In view of this the 'R' command is an extremely useful tool for testing relative calls and jumps in user programs.

Suppose the memory contents around the pointer are as follows:

```
   E69A  28
→  E69B  FC ←
   E69C  EF
```

On the 'R' command the byte pointed to by the memory pointer will be taken as the relative position of the location to move to. The code FC is interpreted as -4 (decimal) and therefore the memory contents around the pointer will now be:

```
   E697  C9
→  E698  F7 ←
   E699  1D
   E69A  28
```

Note that the pointer has apparently only skipped backwards three locations and not the four which were specified. This is because under normal program execution when an instruction is fetched, the program counter (PC) is automatically advanced to the first byte of the next instruction. On executing a relative jump this has to be taken into account.

1.3.4 U

Format: !U

This is another command (like I and R) which sets the memory pointer by reference to an already existing location, this time the Program Counter. The figure below (Fig.1.4) shows the effect of the 'U' command. In the initial state the memory pointer is aligned to some particular location in memory, and the program counter (PC) is pointing to location E6AC. On execution of the 'U' command the memory display is updated so that the memory pointer now aligns itself with the address given by the program counter.

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 35 20
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                        ↑
2029 45    2031 4E    2039 47    2041 55
202A D0    2032 C4    203A CE    2042 53
202B AB    2033 4F    203B 49    2043 D2
202C AD    2034 D2    203C 4E    2044 46
202D AA   >2035 BE<   203D D4    2045 52
202E AF    2036 BD    203E 41    2046 C5
202F DE    2037 BC    203F 42    2047 49
2030 41    2038 53    2040 D3    2048 4E
```

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 AC E6
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                        ↑
E6A0 C9    E6A8 C9    E6B0 0C    E6B8 0C
E6A1 F7    E6A9 CD    E6B1 FF    E6B9 FF
E6A2 21    E6AA AD    E6B2 B7    E6BA F1
E6A3 FE    E6AB E0    E6B3 28    E6BB FE
E6A4 06   >E6AC 18<   E6B4 05    E6BC 01
E6A5 28    E6AD F3    E6B5 3E    E6BD C0
E6A6 02    E6AE F5    E6B6 17    E6BE 3A
E6A7 B7    E6AF 3A    E6B7 32    E6BF 0C
```

FIG. 1.4 - EFFECT OF 'U' COMMAND

When 'single stepping' through machine level programs this command is extremely useful in that it updates the memory display quickly, to correspond with the program counter movements.

1.4 Reading and Modifying Memory

The memory pointer delimiters, as outlined in the summary are given by:

SPACE
CARRIAGE RETURN
- (minus)
/ (backslash)
LINE FEED

and used to read and modify local blocks of memory. To read memory, i.e. look at the contents given by the display, the delimiters are used on their own. To modify memory locations the delimiters are used in conjunction with a two-digit hexadecimal value, which is to be the new contents of the memory location pointed to by the memory pointer.

1.4.1 Reading

When purely examining the contents of local memory the delimiters are used on their own in response to the prompt (!). They are not echoed back in any way although the four-line scroller will scroll up at the bottom of the display. Used in this mode they have the following effects:

SPACE	=	Remain at present location. Do nothing.
CARRIAGE RETURN	=	Increment the memory pointer. If initially at address ED73; will now point to ED74.
-	=	Decrement the memory pointer. If initially at address EC5A; will now point to EC59.
/	=	Move the memory pointer backwards eight locations. If initially at address ED1Ø; will now point to EDØ8.
LINE FEED	=	Advance the memory pointer by eight locations. If initially at address EC2Ø; will now point to EC28.

1.4.2 Modifying

When modifying memory locations the above delimiters are used together with a two-digit hexadecimal value. If only one digit is entered as the value then a leading zero is assumed:

e.g. 3 < delimiter> = Ø3 < delimiter>

If more than two digits are entered then the last two digits entered will be taken as the value.

Format: e.g. ! 3B < delimiter >

To modify the memory content of the location pointed to by the memory point the hexadecimal value must be entered followed by any one of the delimiters, in response to the prompt (!). The value specified will be entered into the location originally pointed to and then the memory pointer will be adjusted as defined by the delimiter.

For example, suppose that the command: 4F < carriage return> had been input with the initial memory display around the memory pointer being:

```
  21A3   CD
  21A4   34
→ 21A5   ØØ ←
  21A6   ØØ
```

The initial location pointed to (21A5) would have a new content (4F) entered and then the memory pointer would be modified accordingly; in this case incremented by one location. The memory display around the pointer would therefore become:

```
  21A4   34
  21A5   4F
→ 21A6   ØØ ←
  21A7   ØØ
```

Note that if an attempt is made to modify the contents of ROM (Read-Only-Memory) then although the command will be accepted and the effect of the delimiter exercised, the memory modification will be nullified.

1.5 Modifying Registers

First of all, another look at the front panel. In the upper region of the display, the Register display, to the left of the register mnemonic for the Program Counter (PC) there is a pointer. By using the register pointer delimiter '.' (full stop), this pointer can be moved down to point to any of the register mnemonics (including the IO display).

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 AC E6
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                     ↑
E6A0 C9    E6A8 C9    E6B0 0C    E6B8 0C
E6A1 F7    E6A9 CD    E6B1 FF    E6B9 FF
E6A2 21    E6AA AD    E6B2 B7    E6BA F1
E6A3 FE    E6AB E0    E6B3 28    E6BB FE
E6A4 06   >E6AC 18<   E6B4 05    E6BC 01
E6A5 28    E6AD F3    E6B5 3E    E6BD C0
E6A6 02    E6AE F5    E6B6 17    E6BE 3A
E6A7 B7    E6AF 3A    E6B7 32    E6BF 0C
```

(a) 380Z Front Panel

→PC	PC	PC
SP	→SP	SP
IY	IY	→IY
IX	IX	IX
HL	HL	HL
DE	DE	DE
BC	BC	BC
AF	AF	AF
IO	IO	IO

On each input of '.' the register pointer moves down one row.

(b) Successive effects of '.' command

FIG 1.5 FRONT PANEL AND EFFECT OF '.' COMMAND

On reaching the IO display, another '.' input will set the register pointer back to the program counter (PC).

The purpose in being able to move the register pointer around in this way is that it is the register pointed to which can be modified. To modify the register pointed to a four-digit hexadecimal value is expected plus the '.' delimiter.

Format: e.g. !Ø2Ø3 <.>

Normal hexadecimal input rules apply here, meaning that leading zeroes are assumed if necessary and the last four digits will be accepted if more than four are entered.

A point to watch out for here is that if the wrong delimiter is used, e.g. < carriage return >, then it will be a memory location which will be altered - not the register. It will be noticed that the use of the register pointer delimiter in this mode does not result in the register pointer being moved. Hence the delimiter has two roles.

(a) When used alone it specifies a shift of the register pointer;
(b) When used together with a hexadecimal value it specifies a modification to the appropriate register.

As soon as the delimiter is entered the hexadecimal value given will be entered into the register pointed to. Note that the contents of the eight-byte row displayed to the right of the register content display also changes to coincide with the memory locations denoted by the new content of the register. In the case of the AF register note that e.g. ØØFF will place 'FF' into the flag register and hence display all the flags available.

1.6 Modifying IO Ports

The Input/Output ports are addressed using the delimiters given in the summary at the beginning of this chapter. The 'contents' display of the ports will not alter however unless particular ports are explicitly addressed. It is unusual for the beginner to use these ports and they are normally only used when dealing with control applications.

1.7 Utilities (X, P, S, G, N, H, Z, K, J)

Nine routines exist within the 'front panel' of the 380Z and are designed specifically to ease the formation and debugging of machine level programs.

1.7.1 X

Format: ! X

This command will enable the alternate set of registers, available within the Z80 microprocessor, to be displayed in place of the original set.

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL' 2A30 00 00 20 85 00 58 00 00
 DE' 0000 AC E6 00 00 C3 2F E4 00
 BC' 8518 FF FF FF FF FF FF FF FF
 AF' 0C95 S  H V C    ↑

 IO 0000 FF FF FF FF FF FF FF FF
                     ↑
1497 D6    149F 21    14A7 FE    14AF CD
1498 03    14A0 00    14A8 1A    14B0 A3
1499 F1    14A1 01    14A9 37    14B1 14
149A C3    14A2 C9    14AA C8    14B2 D0
149B 69   >14A3 CD<   14AB B7    14B3 FE
149C 05    14A4 E5    14AC 28    14B4 0A
149D 36    14A5 26    14AD F5    14B5 28
149E 00    14A6 D8    14AE C9    14B6 04
```

FIG. 1.6 - ALTERNATE REGISTER SET

Each successive 'X' command input to the prompt (!) will exchange the set of registers displayed. Note that when the alternate set is being displayed each of them (HL', DE', BC', AF') is flagged with an apostrophe. The set of registers displayed by the front panel is the set which will be used in the running of a program, so it is important before running programs from the front panel that the correct set is available.

1.7.2 P

Format: e.g.

```
!P
FIRST > 0200
LAST > 0209
WITH > AF
```

Two functions are carried out when the command 'P' is entered in response to the command prompt (!). A section of memory is 'filled' and then 'tested'. On entry of the command the routine prompts for the first and last addresses to be filled. It then prompts for the two-digit hexadecimal value which is to be entered into the locations. An ?ERR? message will be output and a return made to the Cassette Operating System (prompt →), if the final address is less than the first address.

On entering the 'fill with' byte, all locations between the specified limits, inclusive, will be filled with this byte. This command has no effect on the memory pointer whatsoever.

As the memory is being filled, it is also read back to check that the specified change has taken place. If an error is detected then the ?ERR? message is output and the front panel re-entered with the memory pointer aligned to the location causing the error. To continue the fill and test routine at the next byte the 'K' command can be entered, or an exit to either of the operating systems made (CTRL-B,CTRL-C). This facility does provide a very convenient mechanism for testing memory integrity, memory type (RAM/ROM), or end of user memory. Note that non-existent memory reads as FF(hex.).

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  2A30 00 00 20 85 00 58 00 00
 DE  0000 AC E6 00 00 C3 2F E4 00
 BC  8518 FF FF FF FF FF FF FF FF
 AF  0C95 S  H V C    ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑
01F4 00    01FC 00    0204 E9    020C C3
01F5 00    01FD 00    0205 03    020D 00
01F6 00    01FE 00    0206 C3    020E 04
01F7 00    01FF 00    0207 03    020F F7
01F8 00   >0200 C3<   0208 04    0210 22
01F9 00    0201 09    0209 CD    0211 FE
01FA 00    0202 02    020A E9    0212 03
01FB 00    0203 CD    020B 03    0213 28
```

a) Initial state

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 00 02
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑
01F4 00    01FC 00    0204 AF    020C C3
01F5 00    01FD 00    0205 AF    020D 00
01F6 00    01FE 00    0206 AF    020E 04
01F7 00    01FF 00    0207 AF    020F F7
01F8 00   >0200 AF<   0208 AF    0210 22
01F9 00    0201 AF    0209 AF    0211 FE
01FA 00    0202 AF    020A E9    0212 03
01FB 00    0203 AF    020B 03    0213 28
```

(b) After format example (above) entered

FIG 1.7 - EFFECT OF 'P' COMMAND (e.g.)

1.7.3 S

Format: e.g.
```
! S
FIRST> 0200
LAST> 0203
TO> 0208
```

A block of memory, as defined by the user in response to the FIRST and LAST prompts, is shifted to another area of memory store as defined by the user in response to the TO prompt.

The shift routine is entered by typing 'S' in response to the command prompt(!), and three four-digit hexadecimal addresses are required for operation. All contents of the locations specified by the user, inclusive, are moved sequentially to another area of memory starting at the address specified. The memory pointer is not affected by this command. It is possible to shift a block up or down the memory and the source area and the destination area are allowed to overlap. If however the final address is less than the first address specified, an ?ERR? message will be output and a return made to the Cassette Operating System(prompt→).

Note that the memory contents are not tested, as in the 'P' command, and therefore any attempt to overwrite ROM (Read Only Memory), or access non-existant memory, will be nullified- though not pointed out to the user.

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 00 02
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑
01F4 00    01FC 00    0204 E9    020C C3
01F5 00    01FD 00    0205 03    020D 00
01F6 00    01FE 00    0206 C3    020E 04
01F7 00    01FF 00    0207 03    020F F7
01F8 00   >0200 C3<   0208 04    0210 22
01F9 00    0201 09    0209 CD    0211 FE
01FA 00    0202 02    020A E9    0212 03
01FB 00    0203 CD    020B 03    0213 28
```

(a) Initial state

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 00 02
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑
01F4 00    01FC 00    0204 E9    020C C3
01F5 00    01FD 00    0205 03    020D 00
01F6 00    01FE 00    0206 C3    020E 04
01F7 00    01FF 00    0207 03    020F F7
01F8 00   >0200 C3<   0208 C3    0210 22
01F9 00    0201 09    0209 09    0211 FE
01FA 00    0202 02    020A 02    0212 03
01FB 00    0203 CD    020B CD    0213 28
```

(b) After format example (above) entered.

FIG 1.8 - EFFECT OF 'S' COMMAND (e.g.)

An extra point here is that the front panel 'S' command can also be executed directly from the Cassette Operating System (prompt→), but not the Basic Disc Operating System (normal prompt A>).

1.7.4 G, N

These two commands are taken together as they both provide for a 'memory search'.

Format: e.g.

```
! G
> AF
> E5
> E1
>
```

```
! N
```

The 'G' command enables a particular byte pattern to be searched for within memory. On entering this command the routine will prompt (>) for the byte values in the pattern. Prompts will continue until a RETURN is pressed with no associated byte.

On the final RETURN the routine will search for the specified pattern within memory. If the pattern is found then the memory pointer is modified to point to the location containing the first byte of the first occurrence of the pattern.

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  01D5 80 C3 0F 04 00 FF EE 00
 IX  7B00 7F C8 F7 01 C9 CD C8 7B
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 00 02
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑

01F4 00    01FC 00    0204 E9    020C C3
01F5 00    01FD 00    0205 03    020D 00
01F6 00    01FE 00    0206 C3    020E 04
01F7 00    01FF 00    0207 03    020F F7
01F8 00   >0200 C3<   0208 04    0210 AF
01F9 00    0201 09    0209 CD    0211 E5
01FA 00    0202 02    020A E9    0212 E1
01FB 00    0203 CD    020B 03    0213 28
```

(a) Initial state

```
>PC  E6AC C9 CD AD E0 18 F3 F5 3A
 SP  FFBE 9E E6 AC E6 3F E4 FF 00
 IY  E1AE 20 56 4E 43 CD E2 E6 2A
 IX  FFEC 3C E9 00 00 7F E1 42 06
 HL  EED0 18 00 F0 BA C0 F5 00 01
 DE  FF04 04 00 F5 BE C1 F5 10 02
 BC  0000 AC E6 00 00 C3 2F E4 00
 AF  0642  Z    N     ↑

 IO 0000 FF FF FF FF FF FF FF FF
                      ↑

0204 E9    020C C3    0214 0C    021C 32
0205 03    020D 00    0215 FE    021D 3C
0206 C3    020E 04    0216 05    021E 02
0207 03    020F F7    0217 C0    021F AF
0208 04   >0210 AF<   0218 3A    0220 C9
0209 CD    0211 E5    0219 3C    0221 CD
020A E9    0212 E1    021A 02    0222 1D
020B 03    0213 28    021B 2F    0223 26
```

(b) After format example (above) entered.

FIG 1.9 - EFFECT OF 'G' COMMAND (e.g.)

Once a pattern has been entered using the 'G' command, the 'N' command will get the next occurrence of the same pattern. On successive entries of 'N' the memory pointer may be seen to point to location FF4E, this is the store used by the routines for the pattern. Note therefore that if the memory pointer never moves from FF4E, this is an indication that the pattern does not exist in usable memory.

1.7.5 H

Format: e.g. ! H
> 342A
> 342A

Typing 'H' provides the user with a hexadecimal calculator. This is therefore an extremely useful tool for working out relative addresses, amongst other things.

The routine prompts (>) for two hexadecimal values. After the second value has been entered the routine will print out the sum and the DIFFERENCE of the specified values. For example, using the format figures (above) the scrolling area at the bottom of the screen will read as follows:

```
! H
> 342A
> 342A   6854   0000
          Sum   Difference
```

Be careful when determining relative addresses. The 'R' command should always be used to check these.

1.7.6 Z,K

Format: ! Z ! K

In front panel mode it is possible to step through machine level programs an instruction at a time, by using the 'Z' command. The next instruction to be obeyed will be given by the program counter (PC) - not the memory pointer. After any instruction has been executed the contents of all affected registers will be modified appropriately, with the program counter (PC) containing the address of the next instruction to be obeyed.

Complex instructions such as LDIR will require a multiple number of 'single steps' for completion, owing to the way in which the Z80 executes such instructions.

One particular difficulty with single stepping is the use of routines which are time dependent, e.g. the keyboard input routines. A convenient way round this is to use the BREAK code (RST 38H, FF hex). This

break code is inserted into the first byte of the instruction following the time dependent routine. The 'K' command can now be given which will cause the program to be executed at normal speed, using the values presently in the registers. When the time dependent routine has been fully executed (for example with a keyboard input routine this could be after a key has been pressed), the break point will be encountered, causing a fresh display of the front panel and a return to front panel mode. At this point the program counter will contain the address of this 'break' instruction and the memory pointer will be pointing to the same. The original content of this location can now be replaced and the single stepping continued as normal.

Single stepping is achieved in the 380Z by enabling a counter which causes a non-maskable interrupt (NMI) after the execution of a single instruction. Prior to the execution of the instruction the contents of locations ØØ66 to ØØ68 hex. are saved (NMI transfer vector) and stored immediately afterwards. For this reason it is not possible to single step correctly through these locations. This is not a serious handicap as in nearly all cases single stepping is carried out within user programs and these should never access this area anyway, for program instructions.

1.7.7 J

Format: e.g. ! J > 243A

The 'J' (JUMP) command is used to start the normal execution of a machine level program at the address specified by the user. On issuing the 'J' command a prompt (>) is given for an address. When the address is entered the stack pointer is reset, the specified address is placed into the program counter (PC) and normal execution initiated.

Note that the 'J' command is also available in the Cassette Operating System monitor mode, (prompt →) - but not in the Basic Disk Operating System, (normal prompt A >).

1.8 Exits (CTRL-B, CTRL-C)

An exit can be made from front panel mode at any time, and in two ways.

1.8.1 Control - B

Typing control-B while in front panel mode will force an exit to the Cassette Operating System monitor (prompt →). The stack pointer is reset and the full screen scroller returned.

1.8.2 Control - C

Typing control-C while in front panel mode (or monitor mode) will force an exit to the CP/M Disc Operating System. This is only the case if the 'systems' disc is in unit A. If the disc or disc unit is not available for some reason then the error message ?BOOT? will appear and a return to the Cassette Operating System made.

1.9 Calling the Front Panel

Finally, the Front Panel may be entered via a defined address which can be 'called'. As long as the PC is not altered in any way, the 'K' command will return to the instruction following the CALL, and normal execution will be resumed.

SECTION 2

2. Z80 INSTRUCTION SET

The Z80 microprocessor unit is the third member of the family, the first two being the 8008 (48 instructions) and the 8080 (78 instructions). A set of 158 instructions comprises the repertoire of the Z80. This implies that although the Z80 is a powerful MPU(microprocessor unit) it is not an easy device to master well. As time and practice progress however the full qualities of this excellent MPU will begin to evolve.

In many cases it is possible to group a number of instructions and consider them all in one go. By describing, in as much detail as necessary, the operation of one of the instructions in any group, the operation of the remaining instructions will also be known. The source and destination of the data and affected flags are likely to be the only differences.

It is on this principle that the instruction set of the Z80 has been organised here. Wherever possible, instructions have been grouped together. For instance, all eight-bit loads are considered together. If it is possible to sub-divide these groups so that each sub-division contains an operationally similar set of instructions, this has also been done.

Considerable effort has been put in to arrange the layout of all groups and sub-divisions of groups of instructions to allow for easy reference. It should not, and is not intended to be, necessary to know all the instructions in any one section before progressing on to the next. Reasonably complex, though not necessarily efficient, machine level programs can be written by referencing the appropriate beginnings of sections. As expertise increases, each section can be read into a little more fully.

Two non-standard (ZILOG) mnemonics have been introduced by Research Machines Ltd. effectively increasing the instruction range of the Z80. This capability has come about by virtue of pseudo-instructions provided by the Cassette Operating System monitor of the 380Z. One of these, the Call-Relative (CALR) instruction, will be dealt with in the section on 'Calls and Returns'. The second is the 'trap' instruction (EMT) and details of this can be found in the COS Reference Manual.

At present this section has been included mainly for 'completeness' of the text as a whole. It is not intended to be a teaching text on machine level programming, the bibliography will refer to many good books on this.

2.1 8-BIT LOADS

In all but two of these instructions, both of which are rarely used, the flag register remains unaffected by the operation. In all cases the source content remains unchanged after the transfer. Both these points are important as it is extremely useful to carry out a number of load instructions and not have to be concerned about the flags, and to make a data transfer without corrupting the source.

2.1.1 'REGISTER-REGISTER' AND 'IMMEDIATE VALUE - REGISTER' LOADS

REPERTOIRE:

	Source							
	A	B	C	D	E	H	L	n
Destination								
A	7F	78	79	7A	7B	7C	7D	3E
B	47	4Ø	41	42	43	44	45	Ø6
C	4F	48	49	4A	4B	4C	4D	ØE
D	57	5Ø	51	52	53	54	55	16
E	5F	58	59	5A	5B	5C	5D	1E
H	67	6Ø	61	62	63	64	65	26
L	6F	68	69	6A	6B	6C	6D	2E

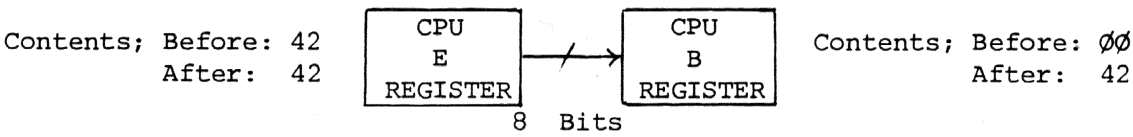
NB The above table ignores the four more rarely used instructions two of which do affect the flag register.

e.g.	MNEMONIC	HEX CODE	ACTION
a)	LD C,A	4F	Load contents of A register into the C register.
b)	LD D,32H	16 32	Load 32 hex into the D register.

All 'register-register' loads for this group are single byte instructions and the contents of any register can be copied into any other register, with the exception of the flag register. All 'immediate value-register' loads are two-byte instructions. The first byte defines the destination register and the second byte is the hex value for the load.

OPERATION:

e.g. LD B,E



Flags affected: none

2.1.2 'MEMORY-REGISTER' LOADS

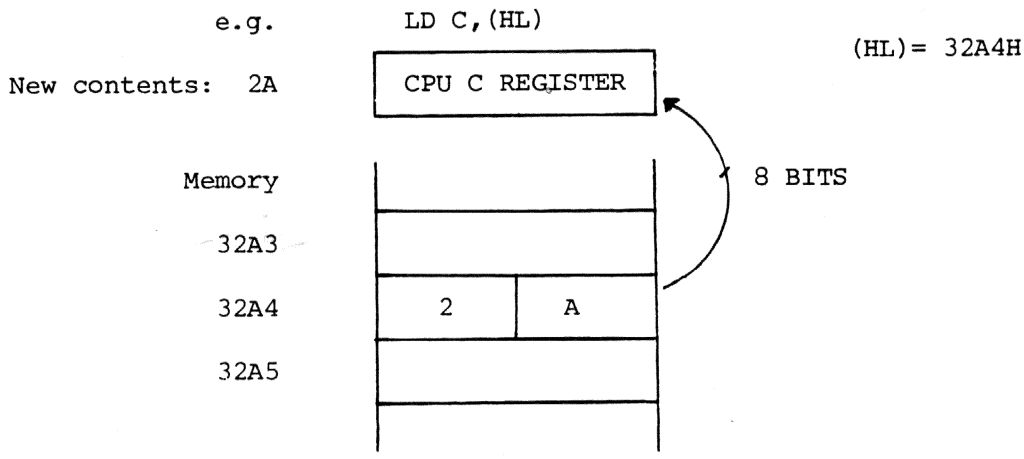
REPERTOIRE:

	Source					
	(HL)	(BC)	(DE)	(IX+d)	(Iy+d)	(nn)
Destination						
A	7E	ØA	1A	DD7E	FD7E	3A
B	46			DD46	FD46	
C	4E			DD4E	FD4E	
D	56			DD56	FD56	
E	5E			DD5E	FD5E	
H	66			DD66	FD66	
L	6E			DD6E	FD6E	

e.g.	MNEMONIC	HEX CODE	ACTION
a)	LD B,(HL)	46	Load into the B register the contents of the memory location pointed to by the HL register pair.
b)	LD D,(IX+2BH)	DD 56 2B	Load into the D register the contents of the memory location pointed to by the index register (IX) plus the offset of 2B hex.
c)	LD A,(34B2H)	3A B2 34	Load into the accumulator the contents of memory location 34B2 hex.

These are either single-byte or three-byte instructions. Note that when using the contents of BC or DE, or using an absolute address 'nn' as the source, the A register is the only possible destination.

OPERATION:



Note that exactly the same effect would have been achieved if the instruction; LD C, (IY+Ø4H) where (IY)=32AØH, was executed.

Flags affected: none.

2.1.3 'REGISTER-MEMORY' AND 'IMMEDIATE VALUE-MEMORY' LOADS

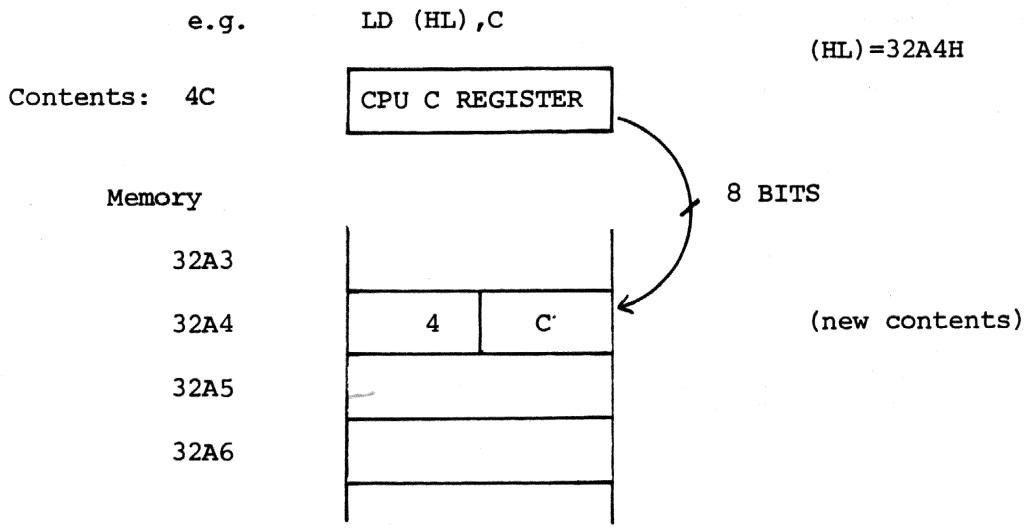
REPERTOIRE:

	Source							
	A	B	C	D	E	H	L	n
Destination								
(HL)	77	7O	71	72	73	74	75	36
(BC)	Ø2							
(DE)	12							
(IX+d)	DD77	DD7Ø	DD71	DD72	DD73	DD74	DD75	DD36
(IY+d)	FD77	FD7Ø	FD71	FD72	FD73	FD74	FD75	FD36
(nn)	32							

e.g.	MNEMONIC	HEX CODE	ACTION
a)	LD(HL),C	71	Load into the memory location given by the HL register pair the contents of the C register.
b)	LD(HL),4AH	36 4A	Load into the memory location given by the HL register pair the hex value of 4A.
c)	LD(IY+Ø4H),E	FD 73 Ø4	Load into the memory location given by the IY index register plus the offset of Ø4H, the contents of the E register.
d)	LD(2A47H),A	32 47 2A	Load into the memory location 2A47H, the contents of the accumulator.
e)	LD(IX+2CH),6FH	DD 36 2C 6F	Load into the memory location given by the contents of the IX index register plus the offset of 2C hex, the hex value of 6F.

This range of 8-bit loads can be one-byte to four-bytes long for the instruction. Note especially that the only register which can be used to enter a value directly to a named location, is the A register; and that if a given hex value is to be loaded into a location, the location must be specified in either the HL register pair or one of the index registers.

OPERATION:



Note that exactly the same effect would have been acheived if the instruction; LD(IX+Ø4H),C where (IX)=32AØH , was executed.

Flags affected: none.

2.1.4 'ACCUMULATOR ↔ MEMORY REFRESH/INTERRUPT VECTOR REGISTERS' LOADS

REPERTOIRE:

LD A,I)
LD A,R) affect the flag register.

I=Interrupt Vector Register
R=Memory Refresh Register

LD I,A
LD R,A

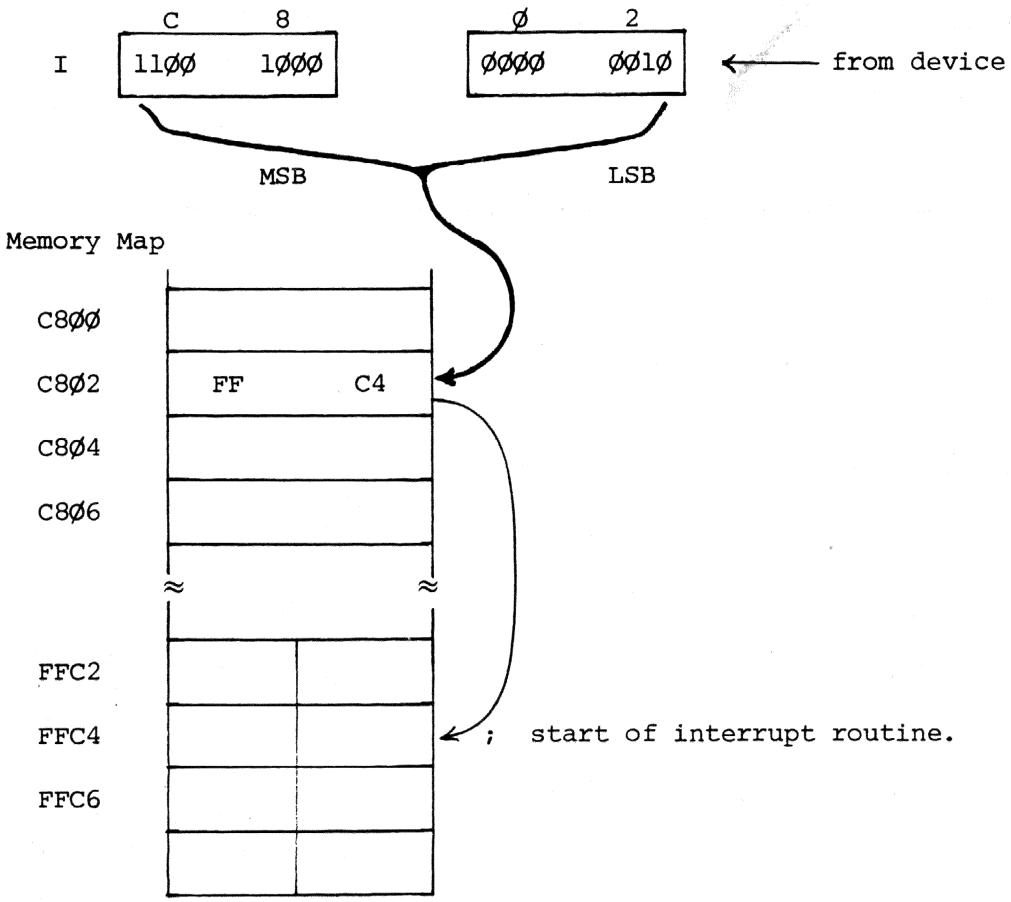
Refresh Register

The 'R' register is used by the CPU to enable automatic refreshing of external 'dynamic' RAM (Random Access Memory). Most systems will have dynamic RAM in them and for this reason it is not advisable to tamper with the 'R' register. Certainly it is not normally used by the programmer.

Interrupt Vector Register

The 'I' register is, as the name suggests, used to set up an interrupt vector table which will allow up to 128 (decimal) interrupt routines to be accessed, using a particular 'mode' of interrupt handling.

Register 'I' is loaded with the most significant eight bits of the address specifying the position of the interrupt vector table. When an interrupt occurs the interrupting device is signalled to hand over the least significant eight bits of the address, and these are combined with the contents of the 'I' register. The address now obtained in turn contains the address of the interrupt routine.



The 'I' register is in fact used in one of three possible interrupt modes which can be called upon under program control.

Flags affected: Under LD A,I and LD A,R operations,

C	Z	P/V	S	N	H
•	↕	IFF	↕	0	0

2.2 16-BIT LOADS

In all cases of the 16-bit load the flag register remains unaffected by the operation. As with the 8-bit loads the contents of the source in all transfers remains unaffected.

The stack operations (i.e. POP, PUSH) are also 16-bit loads but these are covered separately, purely because the mnemonics allow for a convenient group. Most texts on the subject will include all the 16-bit transfers in one section.

2.2.1 'REGISTER-MEMORY' AND 'REGISTER-REGISTER' LOADS

REPERTOIRE:

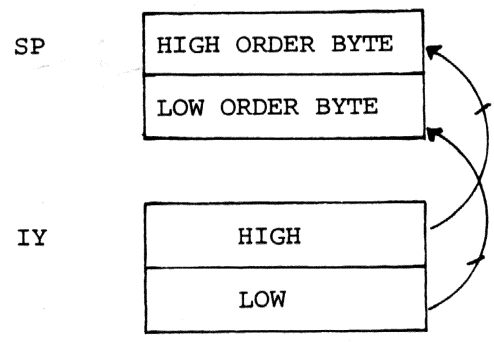
	Source					
	BC	DE	HL	IX	IY	SP
Destination						
(nn)	ED43	ED53	22	DD22	FD22	ED73
SP			F9	DDF9	FDF9	

e.g.	MNEMONIC	HEX CODE	ACTION
a)	LD SP,IX	DD F9	Load the contents of the IX index register into the SP (stack pointer) register.
b)	LD(3A42H),DE	ED 53 42 3A	Load the LSB (E) of DE into memory location 3A42 hex, and the MSB(D) into 3A4B hex.

Register to memory transfers are either three-byte or four-byte instructions; register to register transfers being either one-byte or two-byte instructions. Note that in register-register transfers the only possible destination is the Stack Pointer.

OPERATION (REGISTER-REGISTER):

e.g. LD SP,IY

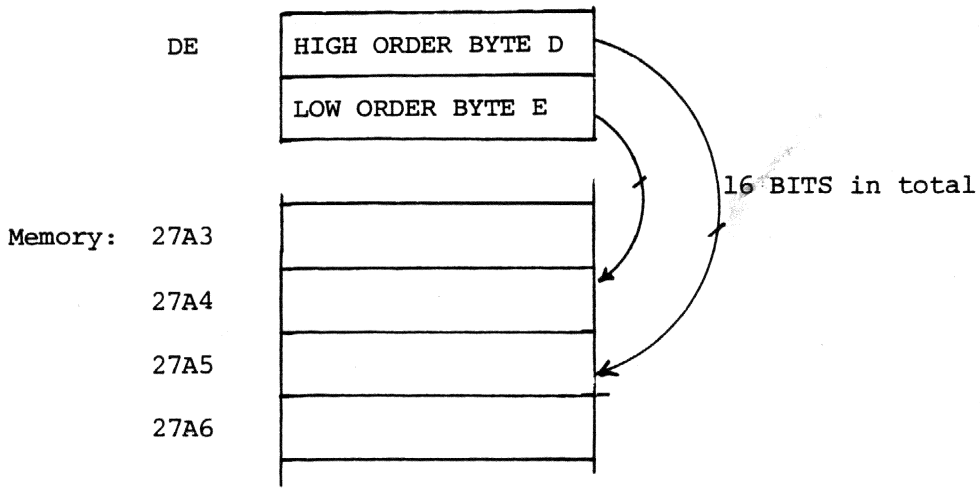


16-BITS in total.

Flags affected: none.

OPERATION (REGISTER-MEMORY)

e.g. LD (27A4H),DE



Note that the register-memory transfer places the LSB chronologically before the MSB within the memory. This is standard practice for all 16-bit storage in memory.

Flags affected: none.

2.2.2 'MEMORY-REGISTER' AND 'IMMEDIATE VALUE-REGISTER' LOADS

REPERTOIRE:

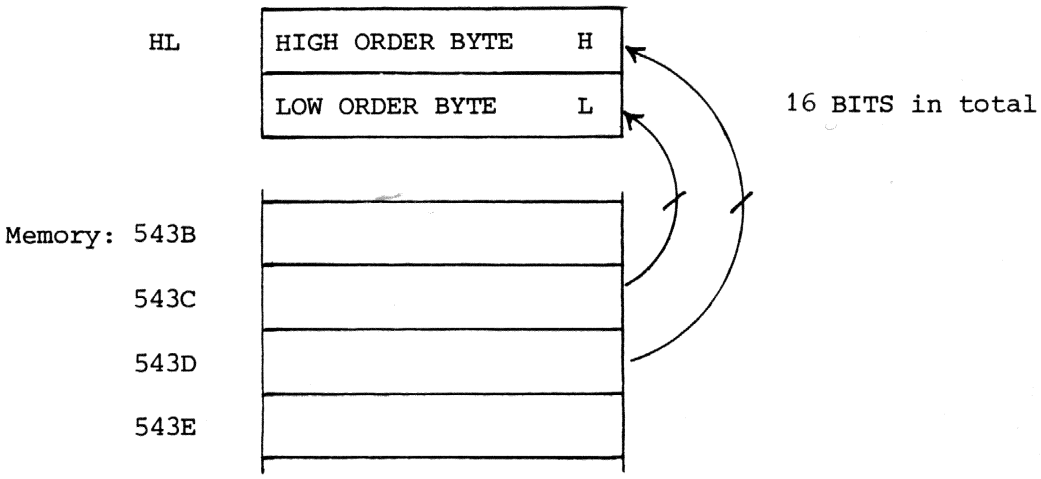
	Source	
	(nn)	nn
Destination		
BC	ED4B	Ø1
DE	ED5B	11
HL	2A	21
IX	DD2A	DD21
IY	FD2A	FD21
SP	ED7B	31

e.g.	MNEMONIC	HEX CODE	ACTION
a)	LD DE,47A9H	11 A9 47	Load the next value 47A9H into the DE register pair.
b)	LD BC,(3C42H)	ED 4B 42 3C	Load the contents of memory location 3C42 hex into the low order byte (C) of the BC register pair, and the contents of memory location 3C43 hex into the high order byte (B).

These instructions are either three or four bytes long. The instructions concerning the Stack Pointer register provide an easy mechanism for the setting up of local 'stack areas'

OPERATION:

e.g. LD HL,(543CH)



Flags affected: none.

2.3 PUSHES AND POPS

The PUSH and POP instructions are really 16-bit load instructions of the form LD(SP), R and LD R,(SP) respectively, where R is a register pair. From this it is seen that the PUSH and POP instructions enable input and output from the stack area.

REPERTOIRE:

PUSH)	F5	C5	D5	E5	DDE5	FDE5
)	AF	BC	DE	HL	IX	IY
POP)	F1	C1	D1	E1	DDE1	FDE1

e.g.	MNEMONIC	HEX CODE	ACTION
a)	PUSH BC	C5	Store the contents of the BC register pair into the top of the stack region.
b)	POP IY	FD E1	Transfer the top two bytes of the stack region into the IY register pair.

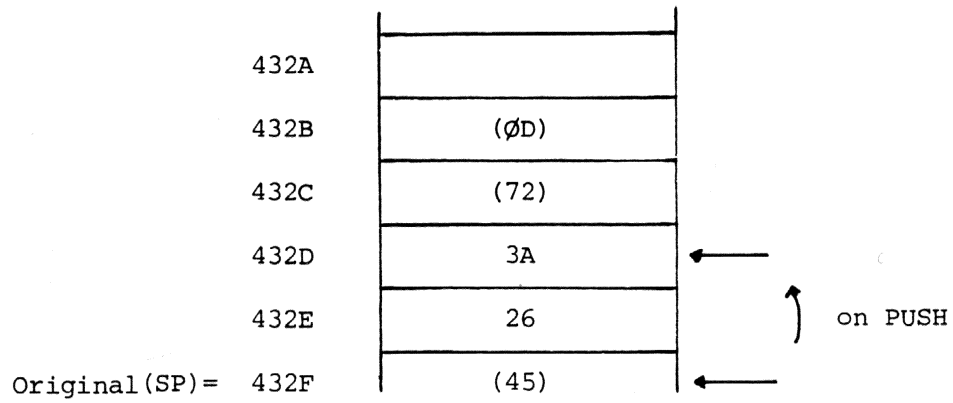
OPERATION:

The stack, which is just a dedicated patch for storing data in a controlled manner, works on a Last-in-First-out (LIFO) principle. This means that the last bit of data stored away will be the first to be retrieved.

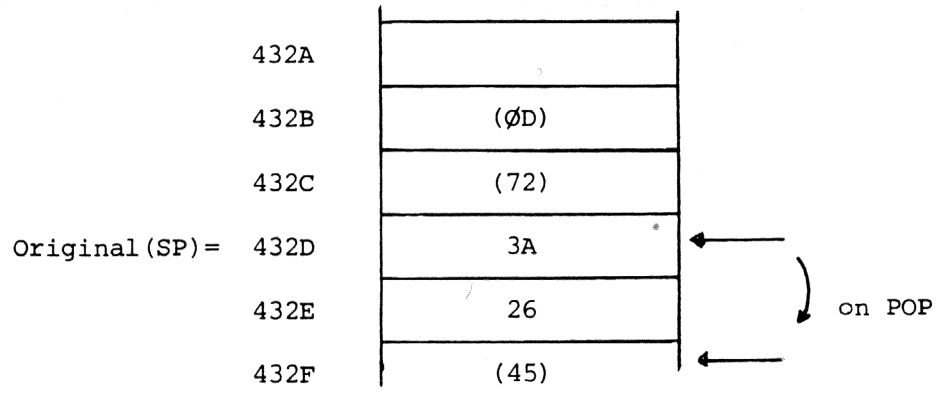
On a PUSH the stack pointer is decremented and the MSB of the register stored; the stack pointer is again decremented and the LSB of the register stored.

On a POP the contents of the memory location, pointed to by the stack pointer, is transferred into the LSB of the destination register. The stack pointer is incremented and the new memory content transferred to the MSB of the destination register. Finally the stack pointer is again incremented.
Consider:

a) PUSH AF
(AF)=263AH



b) POP BC
(BC)←263AH



STACK OPERATION The two operations above combine to form the pseudo-instruction: LD BC,AF

Flags affected: none.

2.4 8-BIT ARITHMETIC

There are six basic 8-bit arithmetic operations available; ADD (Add), ADC (Add with Carry), SUB (Subtract), SBC (Subtract with Carry), INC (Increment) and DEC (Decrement). No instructions exist for any kind of multiply or divide and hence these can only be implemented by software routines or dedicated hardware. All 8-bit arithmetic operations use the accumulator (A) for one of the operands and the result, the other operand being the contents of an 8-bit register, the contents of a memory location, or an absolute 8-bit value.

2.4.1 'ACCUMULATOR AND REGISTER' ARITHMETIC

REPERTOIRE:

	Second Operand						
(MNEMONIC) Instruction	A	B	C	D	E	H	L
ADD	87	8Ø	81	82	83	84	85
ADC	8F	88	89	8A	8B	8C	8D
SUB	97	9Ø	91	92	93	94	95
SBC	9F	98	99	9A	9B	9C	9D
INC	3C	Ø4	ØC	14	1C	24	2C
DEC	3D	Ø5	ØD	15	1D	25	2D

NB The first operand is always the accumulator and the result is always in the accumulator. Naturally this does not apply to 'INC' and 'DEC'.

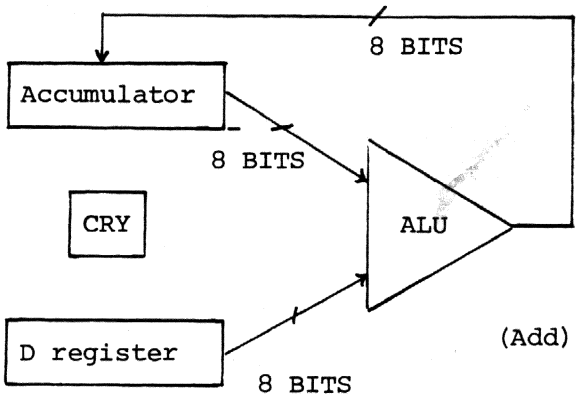
e.g.	MNEMONIC	HEX CODE	ACTION
a)	ADD A,A	87	Double the value in the accumulator.
b)	INC L	2C	Add one to the value in the L register.

These are all single-byte instructions and have varying effects on the flag register. Notice that it is illegal to use the flag register as one of the operands.

OPERATION:

a) Without carry:

Before	After
47	BA
1	Ø
73	73



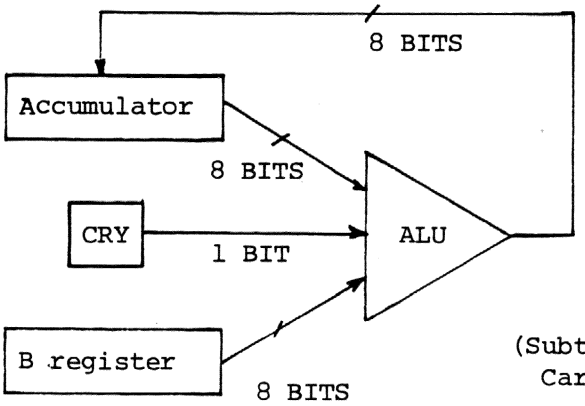
e.g. ADD A,D

Note that contents of carry flag are ignored on input to the arithmetic/logic unit.

(Add)

b) With carry:

Before	After
C4	41
1	Ø
82	82

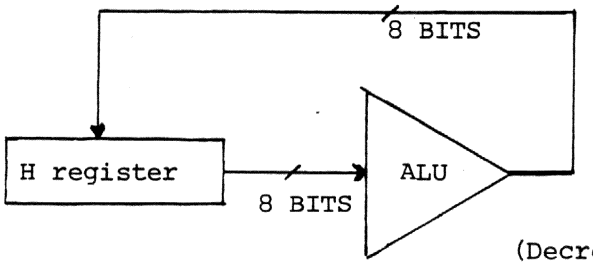


e.g. SBC A,B

(Subtract with Carry)

c) Increment/decrement:

Before	After
47	46



e.g. DEC H

(Decrement)

Flags affected:

ADD; ADC

C	Z	P/V	S	N	H
↕	↕	V	↕	ø	↕

SUB; SBC

C	Z	P/V	S	N	H
↕	↕	V	↕	1	↕

INC

C	Z	P/V	S	N	H
•	↕	V	↕	ø	↕

DEC

C	Z	P/V	S	N	H
•	↕	V	↕	1	↕

2.4.2 'ACCUMULATOR AND MEMORY CONTENTS' ARITHMETIC

REPERTOIRE:

	Second Operand (Memory)		
(MNEMONIC) Instruction	(HL)	(IX+d)	(IY+d)
ADD	86	DD86	FD86
ADC	8E	DD8E	FD8E
SUB	96	DD96	FD96
SBC	9E	DD9E	FD9E
INC	34	DD34	FD34
DEC	35	DD35	FD35

NB These are either single or triple byte instructions. Apart from the INC and DEC instructions the accumulator is always the first operand and the destination for the result. Note therefore that with the Z80 it is not possible to directly increment (e.g.) the contents of a location, this must be performed through a register.

e.g.	MNEMONIC	HEX CODE	ACTION
a)	SUB A,(HL)	96	Subtract from the accumulator the value given by the contents of the location in HL.
b)	DEC(IY+Ø3)	FD35 Ø3	If e.g. IY=3624H then decrement the contents of location '3627H'.

OPERATION

The operation of these instructions and the flags affected by them are similar to those shown in 2.4.1. The only difference being that contents of memory locations are used in the operations instead of 8-bit general purpose registers.

2.4.3 'ACCUMULATOR AND ABSOLUTE VALUE' ARITHMETIC

REPERTOIRE:

Four instructions exist in this category:

i)	ADD A, n	C6	n = any 8-bit value.
ii)	ADC A, n	CE	
iii)	SUB A,n	D6	
iv)	SBC A,n	DE	

e.g.	MENMONIC	HEX CODE	ACTION
	ADD A,+3	C6 Ø3	Take the value in the accumulator, add +3 to it and put the result in the accumulator.

Note that all these instructions are two-byte instructions and that INC and DEC do not exist as they are superfluous.

OPERATION

Again the operation and effects on flags are as given in 2.4.1 the only difference being that absolute values are used instead of 8-bit general purpose registers.

2.4.4 GENERAL PURPOSE ARITHMETIC

In addition to the above 8-bit arithmetic instructions there are three other instructions vital to number crunching. These are:

i) CPL - 'Complement' the accumulator
ii) NEG - 'Negate' the accumulator
iii) DAA - 'Denary' adjust the accumulator

The first two are the instructions for obtaining '1's-complement' and '2's-complement' respectively of the contents of the accumulator. A number of the books given in the bibliography describe the use of the 'complement' representation of numbers very well.

In addition to these two there is the 'DAA' instruction and this is used to obtain a denary number in the accumulator equivalent to the hexadecimal number originally in it. Hence the instruction is used when programming with binary-coded-decimal (BCD) numbers. As an example of its use consider the following:

e.g. Suppose the two BCD numbers 16 and 48 were in the accumulator and 'C' register respectively. In BCD what would be required is:

```
  16     (BCD)
+ 48     (BCD)
  64     (BCD)
```

However the instruction 'ADD A,C' would obviously work in hex, and therefore the following results:

```
  16     (BCD)
+ 48     (BCD)
  5E     (HEX)
```

The 'DAA' instruction would therefore be used to turn the contents of the accumulator from (SE) to (64) and hence obtain the correct BCD representation.

MNEMONIC	HEX CODE
CPL	2F
NEG	ED44
DAA	27

Flags affected:

CPL

C	Z	P/V	S	N	H
.	.	.	.	1	1

NEG

C	Z	P/V	S	N	H
↕	↕	V	↕	1	↕

DAA

C	Z	P/V	S	N	H
↕	↕	P	↕	.	↕

2.5 16-BIT ARITHMETIC

Unlike the eight bit arithmetic set the sixteen bit category is quite limited. Only five operations exist because there is no 'SUB' instruction and all operations take place between CPU registers.

REPERTOIRE:

a)

(MNEMONIC) Instruction	Second Operand BC	DE	HL	SP	IX	IY
ADD	Ø9	19	29	39		
ADC	ED4A	ED5A	ED6A	ED7A		
SBC	ED42	ED52	ED62	ED72		
INC	Ø3	13	23	33	DD23	FD23
DEC	ØB	1B	2B	3B	DD2B	FD2B

NB All of this group have 'HL' as the first operand and the destination for the result, except for INC, DEC.

b)

(MNEMONIC) Instruction	Second Operand			
ADD	DDØ9	DD19	DD29	DD39

NB All of this group have 'IX' as the first operand and the destination for the result.

c)

(MNEMONIC) Instruction	Second Operand			
ADD	FDØ9	FD19	FD29	FD39

NB All of this group have 'IY' as the first operand and the destination for the result.

OPERATION

A pictorial representation of the operation of these instructions can be obtained from 2.4.1, remembering that 16-bit values are being used and that the operands are limited to 16-bit CPU registers.

Examples:

a) Without carry:

Before 'ADD HL,DE'		After	
HL	6768	HL	676C
DE	ØØØ4	DE	ØØØ4

b) With carry:

Before 'SBC HL,BC'		After	
HL	5454	HL	4332
BC	1121	BC	1121
CY	1	CY	Ø

c) Increment/Decrement:

Before 'INC IX'		After	
IX	F4BC	IX	F4BD

Flags affected:

	C	Z	P/V	S	N	H
ADD	↕	•	•	•	Ø	X
ADC	↕	↕	V	↕	Ø	X
SBC	↕	↕	V	↕	1	X

Note that 16-bit INC,DEC do not affect the flags in any way.

2.6 JUMPS

At strategic points within a machine level program it is necessary to jump unconditionally or conditionally, to another part of the program. For instance it may be necessary to by-pass a small local store of data, or branch off to a choice of program segments depending upon given conditions.

All the jump commands in the Z80 instruction set require an absolute address to branch to, and certain registers may be used to supply this address. Note that this applies only to the JUMP instructions, there are further instructions (called JUMP-RELATIVE instructions) which operate in a different way. The flags are used but not altered by JUMP instructions.

REPERTOIRE:

	Condition								
(MNEMONIC) Instruction	none	C	NC	Z	NZ	PE	PO	M	P
JP nn	C3	DA	D2	CA	C2	EA	E2	FA	F2
JP (HL)	E9								
JP (IX)	DDE9								
JP (IY)	FDE9								

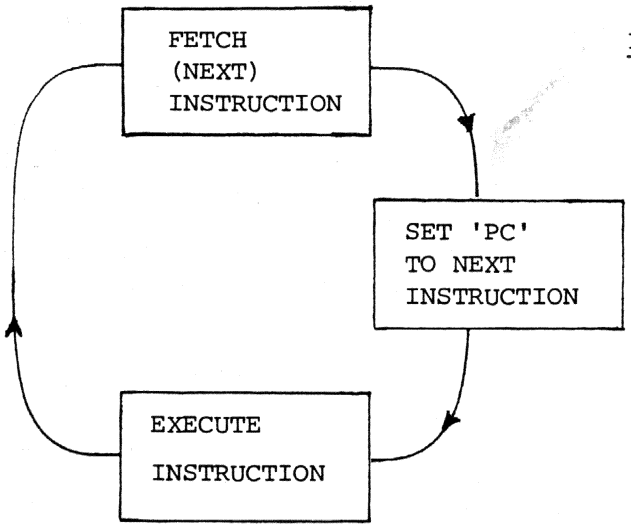
Note that the address to jump to cannot be obtained from the registers if a condition accompanies the instruction.

CODE	CONDITION	FLAG STATE	
C	carry	C=1	)
NC	noncarry	C=Ø	)
Z	zero	Z=1	)
NZ	nonzero	Z=Ø	) As on 380Z
PE	parity even	V=1	) front panel.
PO	parity odd	V=Ø	)
M	sign negative	S=1	)
P	sign positive	S=Ø	)

e.g.	MNEMONIC	HEX CODE	ACTION
a)	JP 43A2H	C3 A2 43	Jump unconditionally to memory location 43A2 hex.
b)	JP NZ,1AC3 H	C2 C3 1A	Jump to memory location 1AC3 hex on condition that the zero flag is not set.

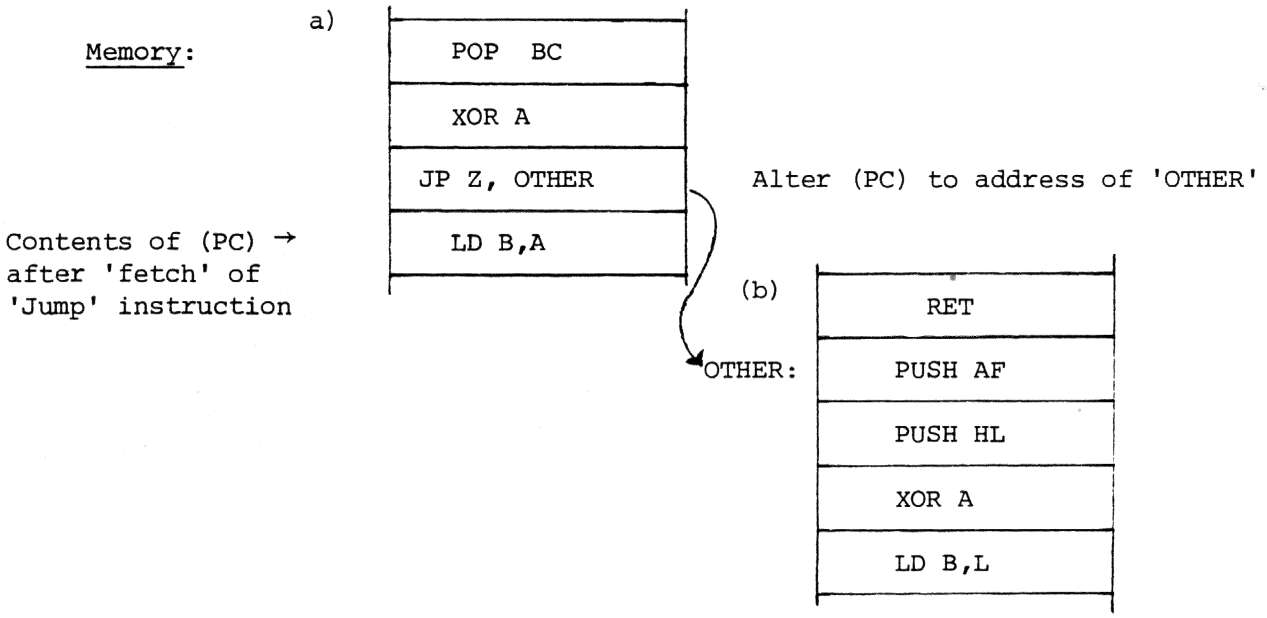
OPERATION:

A very simple schematic of the fetch-execute cycle for instructions in a program might be as follows:



FETCH-EXECUTE CYCLE

If the instruction executed is a JUMP then, provided any conditions are met, the Program Counter (PC) will be loaded with the address given by the jump instruction. In this way, execution of the program instructions has been transferred to a given point in the program.



In the above figure the Program Counter would be pointing to the (LD B,A) instruction as the previous JUMP instruction was being executed. The successful JUMP instruction would alter the contents of the program counter to point to the instruction at location 'OTHER'. Program execution then continues from this point.

Flags affected: none.

2.7 RELATIVE JUMPS

These instructions perform exactly the same function as the absolute jump instructions in that they transfer control from one part of a program to another. The difference exists in the way in which the destination to jump to is calculated. In the absolute jump instructions it is a simple use of collecting the two-byte address given or the register containing the address, and loading the address into the Program Counter, hence transferring control. With relative jumps an 'offset' is specified which basically says go forward so many places or back so many places. This kind of instruction for transferring control is necessary for Position Independent Code (PIC). PIC is code which can be loaded anywhere in the memory of the microcomputer and still run. Obviously if absolute addresses were specified then this would not be possible as control would be passed to the wrong location.

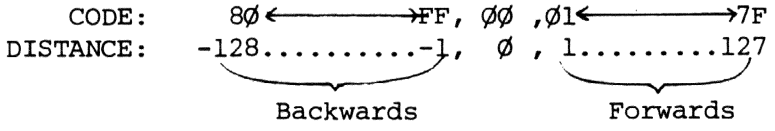
The repertoire for relative jumps is not so grand in the Z80 as the absolute codes. An unconditional relative jump is permitted together with conditional relative jumps using the carry flag and the zero flag only. There is also a special relative jump instruction which uses the zero flag and the 'B' register for setting up control loops.

REPERTOIRE:

	Condition				
(MNEMONIC)	none	C	NC	Z	NZ
Instruction					
JR dis	18	38	3Ø	28	2Ø
Special					
DJNZ dis					1Ø

In no circumstances can a register be used to specify the relative position to jump to. The distance (dis) to move forward or backwards is specified by one and only one byte after the instruction code, in a notation known as "Two's Complement". In the context of the relative jump this means that if the most significant bit of the offset byte is a zero (codes ØØ-$7F_{16}$), then the jump is positive i.e. forwards. If the most significant list is a 'one' (codes 8Ø-FF_{16}) then the jump is negative i.e. backwards. This means that it is possible to jump forward up to 127_{10} places and backward up to 128_{10} places.

RELATIVE JUMP VALUES



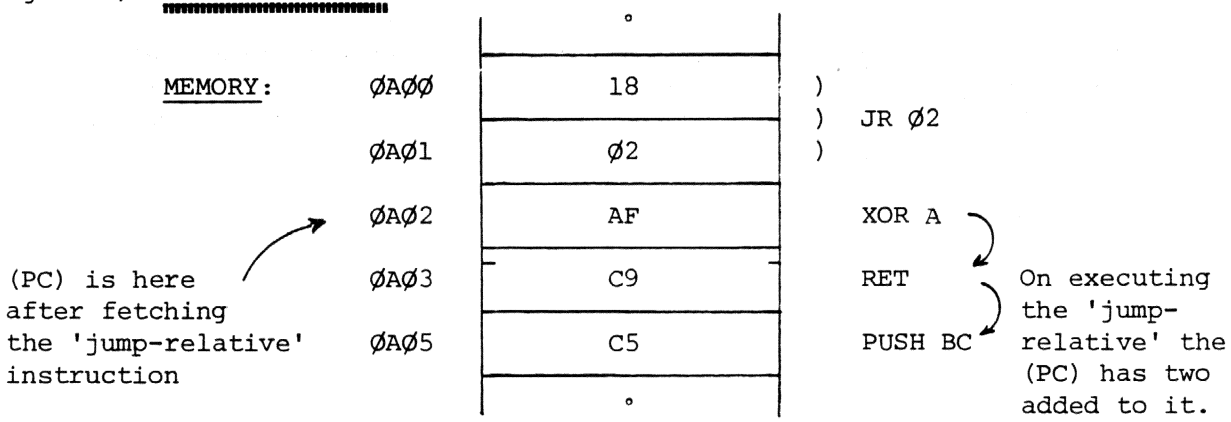
e.g.	MNEMONIC	HEX CODE	ACTION
a)	JR 2AH	18 2A	Jump relative unconditionally forward forty two locations.
b)	JR ØFCH	18 FC	Jump relative unconditionally backward four locations.
(special) c)	DJNZ ØFCH	1Ø FC	Decrement the value of the 'B' register and jump backward four locations if the new value of 'B' is not zero.

(Decrement and Jump if Not Zero)

BE CAREFUL!

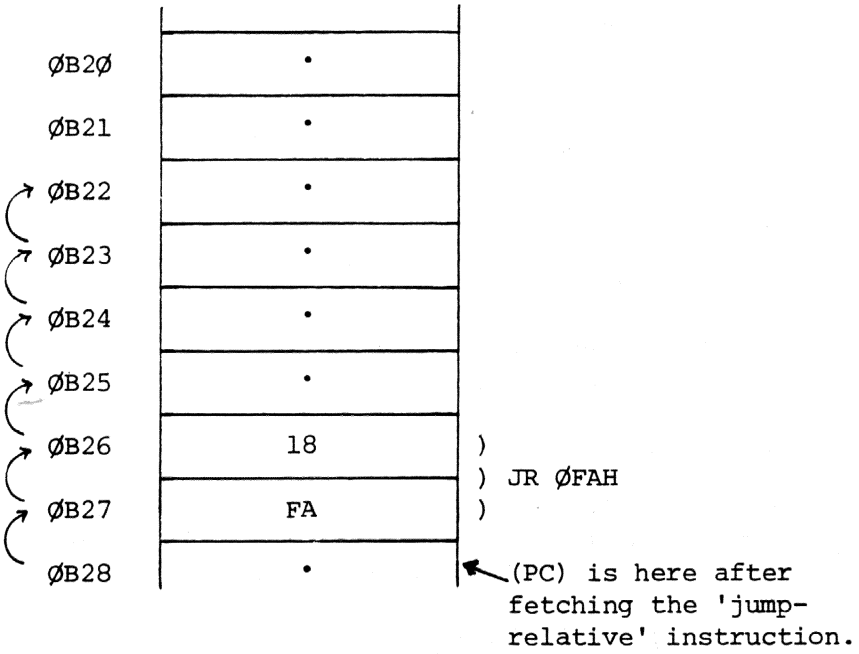
When using these instructions it is vitally important to remember that once an instruction has been fetched from memory the Program Counter is automatically incremented to the start of the next instruction.

e.g. a) GOING FORWARD:



In the above example program execution would normally have continued at location 'ØAØ2'. On execution of the jump relative instruction however the program counter has two added to it, hence program execution is transferred to location 'ØAØ4'.

b) GOING BACKWARD:



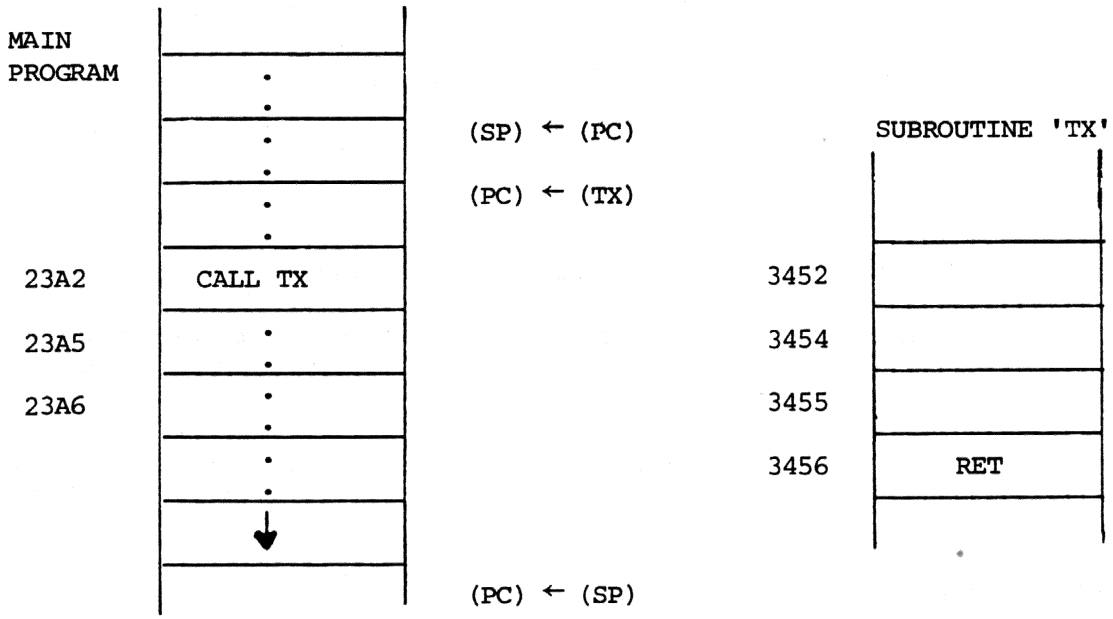
In this case program execution would normally have continued at location 'ØB28'. On executing the jump-relative instruction the (PC) is to have six deducted from it (i.e. FA_{16}=-6). Program execution is therefore transferred to location 'ØB22'.

Flags affected: none.

2.8 SUBROUTINE INSTRUCTIONS

From the previous two sections it is seen that a 'JUMP' can alter the sequence in which the instructions are executed. They are therefore a form of 'Control' instruction. Another control mechanism which is required is one to provide access to and returns from a 'subroutine'. One use of a subroutine is to minimise code, e.g. if an output routine is used often it is wasteful to have to code it every time it is needed. It is obviously better to 'CALL' the routine and then 'RETURN' from it on each occasion. The overall principles at machine level are no different from say subroutines in BASIC or procedures in PASCAL although the intricacies are different.

The Z80 uses the 'CALL' instruction to access a subroutine and a "RET' instruction to return execution to the calling program.



OPERATION OF CALL/RET

Remember that the Z80 is a 'stack' orientated device and subroutine access is achieved using the stack. When a 'CALL' is executed the contents of the program counter are stored on the stack (this being the address of the next instruction) and execution is transferred to the address given by the 'CALL'. At the end of the subroutine a 'RET' instruction places the contents of the stack into the program counter and therefore execution will continue back at the main program. Note very carefully that this means there must not be an uneven number of 'PUSHES' and 'POPS' in the subroutine, or anything could happen!

Apart from the straight 'CALL' and 'RET' it is possible to call and return on the condition of some flag, as given below.

REPERTOIRE:

(MNEMONIC) Instruction	Flag Condition								
	-	C	M	NC	NZ	P	PE	PO	Z
CALL	CD	DC	FC	D4	C4	F4	EC	E4	CC
RET	C9	D8	F8	DØ	CØ	FØ	E8	EØ	C8

NB All the 'CALL's are three-byte instructions and all the 'RET's are single byte instructions.

Flag Mnemonics

-	(unconditional)
C	if carry flag set
M	if sign negative
NC	if carry flag unset
NZ	if non-zero
P	if sign positive
PE	if parity even
PO	if parity odd
Z	if zero

e.g.	MNEMONIC	HEX CODE	ACTION
a)	CALL Z,42A3H	CC A3 42	If zero flag is set then call the subroutine at 42A3H
b)	RET PE	E8	Return to calling program if parity flag shows 'even'.

Flags affected: none.

2.9 RELATIVE CALLS

When writing position-independent code (PIC) it was shown in the section on 'jumps that a 'jump-relative' would be used. Similarly for subroutines a 'call-relative' is needed. Despite the enormous instruction set of the Z80 such an instruction does not exist. Research Machines, in their wisdom, have however implemented such an instruction using one of the 'page-zero' restart calls, (cf. section 2.10). For the Assembler a new mnemonic has been devised to cater for the new instruction. Only the one instruction exists and this is for an unconditional call-relative, i.e. no conditional call-relatives are implemented.

MNEMONIC	HEX CODE	ACTION
CALR 23H	EF 23	(Operation is similar to 'jump-relative' except that the return address is placed on the stack).

2.10 RESTART CALLS

Within the Z80 there are eight instructions which operate like a 'CALL' but the addresses are implied, i.e. they are predefined. Often this is referred to as'page-zero' addressing because the most significant byte of the implied addresses is zero. In the Z80 they are known as 'restart' instructions.

REPERTIORE:

(MNEMONIC) Instruction	HEX CODE	ADDRESS (CALLED)
RST Ø	C7	Ø
RST 8	CF	8
RST 1ØH	D7	1ØH
RST 18H	DF	18H
RST 2ØH	E7	2ØH
RST 28H	EF	28H
RST 3ØH	F7	3ØH
RST 38H	FF	38H

NB Naturally these can be used by the programmer but care should be exercised over their use. A number of them are used by RML to implement certain functions (e.g. EMT,CALR) and it would be disastrous to corrupt the operation of these.

2.11 LOGICAL OPERATIONS

The formal subject of Boolean Algebra has a number of applications and certainly it would be very odd to think of a computer processor without some form(s) of logical instructions. Three major operators are available, 'AND, OR, XOR'. (Note that a 'NOT' operator is given by the arithmetic 'CPL' instruction). Only 8-bit logical operations are permitted and the result and one of the operands is always the accumulator.

REPERTOIRE:

	Second Operand										
(MNEMONIC) Instruction	(HL)	(IX+d)	(IY+d)	A	B	C	D	E	H	L	n
AND	A6	DDA6	FDA6	A7	AØ	A1	A2	A3	A4	A5	E6
OR	B6	DDB6	FDB6	B7	BØ	B1	B2	B3	B4	B5	F6
XOR	AE	DDAE	FDAE	AF	A8	A9	AA	AB	AC	AD	EE

	MNEMONIC	HEX CODE	ACTION
a)	AND A,B	AØ	Logically AND the accumulator and the 'B' register.
b)	OR A,7FH	F6 7F	Logically OR the accumulator with the value '7FH'.
c)	AND A,(IX+5)	DD A6 Ø5	Logically AND the accumulator with the value given in location (IX)+5.

OPERATION:

The effects correspond to normal Boolean rules:

AND A	B	S	OR A	B	S	XOR A	B	S
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

where S = result and A,B = operands.

Flags affected:

AND

C	Z	$P/_V$	S	N	H
Ø	↕	P	↕	Ø	1

OR, XOR

C	Z	$P/_V$	S	N	H
Ø	↕	P	↕	Ø	Ø

2.12 BIT OPERATIONS

These instructions are confined to operating on bits inside eight-bit values, and enable particular specified bits to be SET, RESET (logical Ø), or TESTED. There are a lot of instructions catering for this and therefore their mnemonics etc. are not shown here. The appendices can be referred to in order to find the codes.

Basically the range is split up into two categories, one very large and one very small. The small group consists of just two instructions which enable the 'carry' flag to be complemented (CCF) or set (SCF). The large group enables any bit (no. Ø-7) in the operands:

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

to be:

i)	SET	e.g.	SET 5,(HL)
ii)	RESET	e.g.	RES 3,E
iii)	TESTED	e.g.	BIT Ø,(IX+3)

When the 'BIT' instructions are used the zero flag will contain the complement of the 'bit' specified, i.e. if the bit is a '1' then the zero flag will be unset; if the bit is a 'Ø' then the zero flag will be set.

Flags affected:

SET, RES

BIT

C	Z	$P/_V$	S	N	H
.	↕	X	X	Ø	1

2.13 COMPARISONS

It is often necessary to compare bytes of information and then to act on the comparison accordingly. On some computers this comparison process destroys the value being checked and therefore it is often necessary to store the value away and regain it every time. The Z80 allows the byte in the accumulator to be compared with a byte from somewhere else (e.g. in a lookup table) with neither operand being destroyed. The flag register is affected so that a decision can be made on what to do following the comparison.

REPERTOIRE:

(Mnemonic) Instruction	(HL)	(IX+d)	(IY+d)	A	B	C	D	E	H	L	n
CP	BE	DDBE	FDBE	BF	B8	B9	BA	BB	BC	BD	FE

N.B. The result and the first operand are always in the accumulator. The operation is such that the flag register is altered according to the operation:

A - (value)

So the comparison is performed internally as a subtraction although the accumulator is not affected.

Flags affected:

C	Z	$P/_V$	S	N	H
↕	↕	V	↕	1	↕

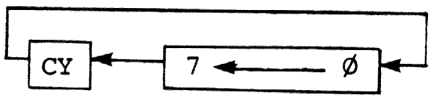
2.14 ROTATES AND SHIFTS

A number of instructions exist to enable an 8-bit byte to have its contents shifted or rotated a 'bit' at a time. Such operations are useful e.g. in arithmetic routines or parallel serial transformation routines etc. The actual codes for the instructions and their mnemonics can be obtained from the appendices. At this point the main concern is to what form these operations take.

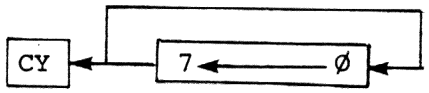
Four main 'ROTATE' sets exist and three 'SHIFT' sets of instructions. In each of the seven cases the eight-bit value can come from:

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

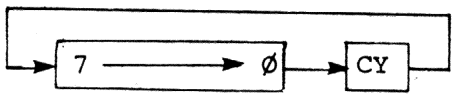
ROTATES i) RL (rotate left)



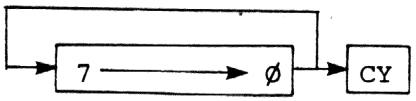
ii) RLC (rotate left and carry)



iii) RR (rotate right)

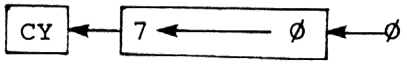


iv) RRC (rotate right and carry)

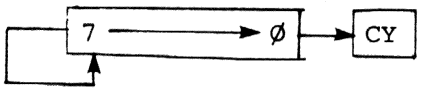


SHIFTS

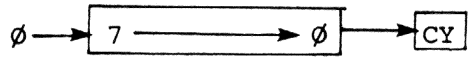
i) SLA (shift left arithmetic)



ii) SRA (shift right arithmetic)



iii) SRL (shift right logical)

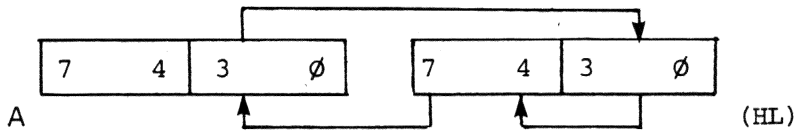


When the accumulator is being used as the source for the 8-bits in a 'rotate' instruction then another set of instructions exist which perform exactly the same operations as above but are twice as fast, and only take up one byte instead of two for the instruction. The main reason for having them is to maintain compatability with the 8080 device.

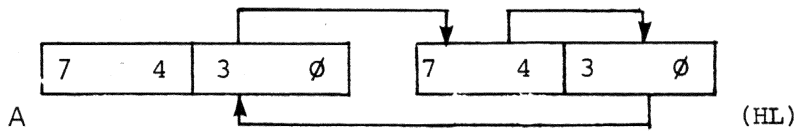
i)	RLA	(17)	equivalent to	RL A (CB 17)
ii)	RLCA	(Ø7)	"	RLC A (CB Ø7)
iii)	RRA	(1F)	"	RR A (CH 1F)
iv)	RRCA	(ØF)	"	RRC A (CB ØF)

In addition to the above, two nibble-orientated 'rotates' are also available, and these work between the accumulator and (HL):

i) RLD (Rotate digits left)



ii) RRD (Rotate digits right)



Owing to the fact that they are 'nibble-orientated' these instructions prove useful in BCD operations, BCD ASCII conversion, etc.

Flags affected:

RLA, RLCA
RRA, RRCA

C	Z	$P/_V$	S	N	H
↕	.	.	.	Ø	Ø

RLD, RRD

C	Z	$P/_V$	S	N	H
.	↕	P	↕	Ø	Ø

(All others)

C	Z	$P/_V$	S	N	H
↕	↕	P	↕	Ø	Ø

2.15 GENERAL INSTRUCTIONS

A small number of general instructions exist which can be split into three sections, two of which only contain one instruction!

a) No-operation instruction:

MNEMONIC	HEX CODE	ACTION
NOP	ØØ	Do nothing

This instruction has a number of uses which will become apparent as programming expertise increases. For example it is useful for providing 'expansion' areas in small blocks of code. As NOPs the Z80 will just plough through them - doing nothing. Flags are unaffected.

b) Halt instruction:

MNEMONIC	HEX CODE	ACTION
HALT	76	Halt the processor

As far as the 380Z is concerned it is never an advantage to literally 'HALT' the processor chip. In any event the only method of recovering from a 'HALT' is to provide an 'interrupt' signal or (on the 380Z) press the reset button. In general - don't use it!

c) Exchange instructions:

There are six instructions within this category, four of which deal with the normal CPU and two others which are concerned with the alternate register set.

i) CPU exchanges

MNEMONIC	HEX CODE	ACTION
EX(SP),HL	E3	Exchange contents of SP with HL
EX(SP),IX	DDE3	Exchange contents of SP with IX
EX(SP),IY	FDE3	Exchange contents of SP with IY
EX DE,HL	EB	Exchange DE with HL.

Operation
The operation of all exchanges is similar throughout the set. As an example suppose DE = 24A3 and HL=65A5, on execution of 'EX DE,HL' the register pair DE=65A5 and HL=24A3.

ii) Alternate register set

MNEMONIC	HEX CODE	ACTION
EX AF,AF'	Ø8	Exchange AF register pair with the alternate pair.
EXX	D9	Exchange register pairs BC,DE,HL with their alternate pairs.

NB: These are the only instructions relating to the alternate set of registers and therefore the alternate set can only be used as a form of temporary storage area.

Flags affected: none.

2.16 INPUT/OUTPUT

Before much is said about the input and output instructions for the Z80 it is worth saying the following. As far as the 'User' is concerned these instructions are rarely used, except perhaps in control applications or communications software. These instructions are not used by the 'user' for (e.g. input from the keyboard or output to the screen. The method of achieving such transfers is through the RML 'EMT' pseudo-instructions (see the Firmware Reference Manual).

Z80 INPUT/OUTPUT

The Z80 has, owing to its 16-bit address bus, the capability of addressing up to 64K of memory. In addition to this and quite independent from it, the Z80 can address 256 peripheral (input/output) addresses, in the range Ø-FF (hex). Various bits of hardware can be attached to the Z80 CPU by the data paths of the system and be addressed quite independently from memory contents. For example, to obtain the contents of memory at location 23 (hex) the instruction:

```
LD A,(23H)
```

could be used. If a peripheral device was addressable as an input device at the same address then:

```
IN A,(23H)
```

would obtain the value from the peripheral.

i) INPUT

It is possible to input from a peripheral device to any of the CPU 8-bit general purpose registers (except the flag register) when the address of the device is held in the 'C' register. Alternatively the device address can be an absolute value whereupon the destination must be the accumulator.

i.e.

MNEMONIC	HEX CODE
IN A,(n)	BD n
IN A,(C)	ED 78
IN B,(C)	ED 4Ø
IN C,(C)	ED 48
IN D,(C)	ED 5Ø
IN E,(C)	ED 58
IN H,(C)	ED 6Ø
IN L,(C)	ED 68

ii) OUTPUT

Output to peripheral devices is limited to the same constraints as those for input:

i.e.	MNEMONIC	HEX CODE
	OUT(n),A	D3 n
	OUT(C),A	ED 79
	OUT(C),B	ED 41
	OUT(C),C	ED 49
	OUT(C),D	ED 51
	OUT(C),E	ED 59
	OUT(C),H	ED 61
	OUT(C),L	ED 69

Flags affected:

In all but one case the flags are unaffected. The one which affects the flags is 'IN A,(C)' and it does so as shown below:

C	Z	$P/_V$	S	N	H
.	↕	P	↕	ø	↕

2.17 'BLOCK' INSTRUCTIONS

Four of the categories described in this chapter, (8-BIT LOADS, COMPARES, INPUTs and OUTPUTs), have a set of related instructions which are designed for use in managing blocks of data. There are four block instructions for each of the four categories and provide similar facilities. The mnemonics and codes are as shown below, the operation being described later:

LOAD		COMPARE		INPUT		OUTPUT	
MNEMONIC	CODE	MNEMONIC	CODE	MNEMONIC	CODE	MNEMONIC	CODE
LDI	EDAØ	CPI	EDA1	INI	EDA2	OUTI	EDA3
LDIR	EDBØ	CPIR	EDB1	INIR	EDB2	OTIR	EDB3
LDD	EDA8	CPD	EDA9	IND	EDAA	OUTD	EDAB
LDDR	EDB8	CPDR	EDB9	INDR	EDBA	OTDR	EDBB

Mnemonic endings mean the following:

I - increment

IR - increment and repeat

D - decrement

DR - decrement and repeat

OPERATION

i) Loads: (e.g.) LDI : (DE)← (HL)
DE ← DE+1
HL ← HL+1
BC ← BC-1

NB LDIR is the same as LDI except that the operation is repeated until BC = Ø. The two decrement instructions are as their 'increment' counterparts except that DE,HL are decremented instead of incremented. Note that these instructions are very useful e.g. in shifting blocks of data.

ii) Compares: (e.g.) CPI : Compare A : (HL)
HL ← HL+1
BC ← BC-1

NB CPIR is the same as CPI except that the operation is repeated until either A=(HL) or BC=Ø. The two decrement instructions are as their 'increment' counterparts except that DE,HL are decremented instead of incremented. Note that these instructions are very useful e.g. in string processing applications.

iii) Inputs: (e.g) INI : (HL)← (C) Peripheral device address
HL ← HL+1
B ← B+1

NB INIR is the same as INI except that the operation is repeated until B=Ø. The two decrement instructions are as their 'increment' counterparts except that HL is decremented instead of incremented. These instructions are useful e.g. in fast data-logging environments.

iv) Outputs: (e.g) OUTI : (C) Peripheral device address← (HL)
HL← HL+1
B ←B-1

NB OTIR is the same as OUTI except that the operation is repeated until B=Ø. The two decrement instructions are as their 'increment' counterparts except that HL is decremented instead of incremented.

Flags affected: (Refer to Zilog manual or Appendix 1).

2.18 INTERRUPTS

Not a great deal is going to be said here about the various interrupt modes and operations available with the Z80 microprocessor. A number of the books in the bibliography deal generously with interrupt programming and it is therefore wasteful to merely reproduce that information here.

Seven instructions exist for interrupt programming on top of the 'LD I,A' instruction discussed in section 2.1.4. Two of these deal with enabling interrupts, two with returning from interrupt routines, and three with setting the 'mode' or type of maskable interrupt response.

i) Enabling interrupts

MNEMONIC	HEX CODE	ACTION
DI	F3	Disable interrupts. This stops the CPU from recognising any maskable interrupts.
EI	FB	Enable interrupts. Enables the CPU to recognise the occurrence of maskable interrupts.

ii) Returns

There are two kinds of interrupt, maskable and non-maskable, and both of their service operations dictate that a different 'return from subroutine' be used instead of the normal 'RET' instruction.

MNEMONIC	HEX CODE	ACTION
RETI	ED4D	Return from a maskable interrupt service routine.
RETN	ED45	Return from a non-maskable interrupt service routine.

iii) Interrupt modes

MNEMONIC	HEX CODF	ACTION
IM0	ED46	Set interrupt mode Ø
IM1	ED56	Set interrupt mode 1
IM2	ED5E	Set interrupt mode 2.

THIS PAGE INTENTIONALLY LEFT BLANK.

APPENDIX 1

Z80 INSTRUCTION TABLES

Z80 INSTRUCTION TABLES

The complete Z80 Microprocessor instruction set is given in the following pages. Three sets of tables are present and these consist of:

a) Z80 instructions sorted by mnemonic,

b) Z80 instructions sorted by op-code,

c) details of operation, timing and effects on flags.

All these tables are reprinted from Zilog data sheets and books and due acknowledgement to Zilog UK is hereby made for this.

A.1.1 Z80 INSTRUCTIONS SORTED BY MNEMONIC

OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT
8E	ADC A, (HL)	FD09	ADD IY, BC	CB4D	BIT 1, L
DD8E05	ADC A, (IX + d)	FD19	ADD IY, DE	CB56	BIT 2, (HL)
FD8E05	ADC A, (IY + d)	FD29	ADD IY, IY	DDCB0556	BIT 2, (IX + d)
8F	ADC A, A	FD39	ADD IY, SP	FDCB0556	BIT 2, (IY + d)
88	ADC A, B	A6	AND (HL)	CB57	BIT 2, A
89	ADC A, C	DDA605	AND (IX + d)	CB50	BIT 2, B
8A	ADC A, D	FDA605	AND (IY + d)	CB51	BIT 2, C
8B	ADC A, E	A7	AND A	CB52	BIT 2, D
8C	ADC A, H	A0	AND B	CB53	BIT 2, E
8D	ADC A, L	A1	AND C	CB54	BIT 2, H
CE20	ADC A, N	A2	AND D	CB55	BIT 2, L
ED4A	ADC HL, BC	A3	AND E	CB5E	BIT 3, (HL)
ED5A	ADC HL, DE	A4	AND H	DDCB055E	BIT 3, (IX + d)
ED6A	ADC HL, HL	A5	AND L	FDCB055E	BIT 3, (IY + d)
ED7A	ADC HL, SP	E620	AND N	CB5F	BIT 3, A
86	ADD A, (HL)	CB46	BIT 0, (HL)	CB58	BIT 3, B
DD8605	ADD A, (IX + d)	DDCB0546	BIT 0, (IX + d)	CB59	BIT 3, C
FD8605	ADD A, (IY + d)	FDCB0546	BIT 0, (IY + d)	CB5A	BIT 3, D
87	ADD A, A	CB47	BIT 0, A	CB5B	BIT 3, E
80	ADD A, B	CB40	BIT 0, B	CB5C	BIT 3, H
81	ADD A, C	CB41	BIT 0, C	CB5D	BIT 3, L
82	ADD A, D	CB42	BIT 0, D	CB66	BIT 4, (HL)
83	ADD A, E	CB43	BIT 0, E	DDCB0566	BIT 4, (IX + d)
84	ADD A, H	CB44	BIT 0, H	FDCB0566	BIT 4, (IY + d)
85	ADD A, L	CB45	BIT 0, L	CB67	BIT 4, A
C620	ADD A, N	CB4E	BIT 1, (HL)	CB60	BIT 4, B
09	ADD HL, BC	DDCB054E	BIT 1, (IX + d)	CB61	BIT 4, C
19	ADD HL, DE	FDCB054E	BIT 1, (IY + d)	CB62	BIT 4, D
29	ADD HL, HL	CB4F	BIT 1, A	CB63	BIT 4, E
39	ADD HL, SP	BC48	BIT 1, B	CB64	BIT 4, H
DD09	ADD IX, BC	CB49	BIT 1, C	CB65	BIT 4, L
DD19	ADD IX, DE	CB4A	BIT 1, D	CB6E	BIT 5, (HL)
DD29	ADD IX, IX	CB4B	BIT 1, E	DDCB056E	BIT 5, (IX + d)
DD39	ADD IX, SP	CB4C	BIT 1, H	FDCB056E	BIT 5, (IY + d)

Reprinted courtesy of Zilog

OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT
CB6F	BIT 5, A	DD2B	DEC IX	71	LD (HL), C
CB68	BIT 5, B	FD2B	DEC IY	72	LD (HL), D
CB69	BIT 5, C	2D	DEC L	73	LD (HL), E
CB6A	BIT 5, D	3B	DEC SP	74	LD (HL), H
CB6B	BIT 5, E	F3	DI	75	LD (HL), L
CB6C	BIT 5, H	102E	DJNZ DIS	3620	LD (HL), N
CB6D	BIT 5, L	FB	EI	DD7705	LD (IX + d), A
CB76	BIT 6, (HL)	E3	EX (SP), HL	DD7005	LD (IX + d), B
DDCB0576	BIT 6, (IX + d)	DDE3	EX (SP), IX	DD7105	LD (IX + d), C
FDCB0576	BIT 6, (IY + d)	FDE3	EX (SP), IY	DD7205	LD (IX + d), D
CB77	BIT 6, A	08	EX AF, AF'	DD7305	LD (IX + d), E
CB70	BIT 6, B	EB	EX DE, HL	DD7405	LD (IX + d), H
CB71	BIT 6, C	D9	EXX	DD7505	LD (IX + d), L
CB72	BIT 6, D	76	HALT	DD360520	LD (IX + d), N
CB73	BIT 6, E	ED46	IM 0	FD7705	LD (IY + d), A
CB74	BIT 6, H	ED56	IM 1	FD7005	LD (IY + d), B
CB75	BIT 6, L	ED5E	IM 2	FD7105	LD (IY + d), C
CB7E	BIT 7, (HL)	ED78	IN A, (C)	FD7205	LD (IY + d), D
DDCB057E	BIT 7, (IX + d)	DB20	IN A, (N)	FD7305	LD (IY + d), E
FDCB057E	BIT 7, (IY + d)	ED40	IN B, (C)	FD7405	LD (IY + d), H
CB7F	BIT 7, A	ED48	IN C, (C)	FD7505	LD (IY + d), L
CB78	BIT 7, B	ED50	IN D, (C)	FD360520	LD (IY + d), N
CB79	BIT 7, C	ED58	IN E, (C)	328405	LD (NN), A
CB7A	BIT 7, D	ED60	IN H, (C)	ED438405	LD (NN), BC
CB7B	BIT 7, E	ED68	IN L, (C)	ED538405	LD (NN), DE
CB7C	BIT 7, H	34	INC (HL)	228405	LD (NN), HL
CB7D	BIT 7, L	DD3405	INC (IX + d)	DD228405	LD (NN), IX
DC8405	CALL C, NN	FD3405	INC (IY + d)	FD228405	LD (NN), IY
FC8405	CALL M, NN	3C	INC A	ED738405	LD (NN), SP
D48405	CALL NC, NN	04	INC B	0A	LD A, (BC)
CD8405	CALL NN	03	INC BC	1A	LD A, (DE)
C48405	CALL NZ, NN	0C	INC C	7E	LD A, (HL)
F48405	CALL P, NN	14	INC D	DD7E05	LD A, (IX + d)
EC8405	CALL PE, NN	13	INC DE	FD7E05	LD A, (IY + d)
E48405	CALL PO, NN	1C	INC E	3A8405	LD A, (NN)
CC8405	CALL Z, NN	24	INC H	7F	LD A, A
3F	CCF	23	INC HL	78	LD A, B
BE	CP (HL)	DD23	INC IX	79	LD A, C
DDBE05	CP (IX + d)	FD23	INC IY	7A	LD A, D
FDBE05	CP (IY + d)	2C	INC L	7B	LD A, E
BF	CP A	33	INC SP	7C	LD A, H
B8	CP B	EDAA	IND	ED57	LD A, I
B9	CP C	EDBA	INDR	7D	LD A, L
BA	CP D	EDA2	INI	3E20	LD A, N
BB	CP E	EDB2	INIR	46	LD B, (HL)
BC	CP H	E9	JP (HL)	DD4605	LD B, (IX + d)
BD	CP L	DDE9	JP (IX)	FD4605	LD B, (IY + d)
FE20	CP N	FDE9	JP (IY)	47	LD B, A
EDA9	CPD	DA8405	JP C, NN	40	LD B, B
EDB9	CPDR	FA8405	JP M, NN	41	LD B, C
EDA1	CPI	D28405	JP NC, NN	42	LD B, D
EDB1	CPIR	C38405	JP NN	43	LD B, E
2F	CPL	C28405	JP NZ, NN	44	LD B, H, NN
27	DAA	F28405	JP P, NN	45	LD B, L
35	DEC (HL)	EA8405	JP PE, NN	0620	LD B, N
DD3505	DEC (IX + d)	E28405	JP PO, NN	ED488405	LD BC, (NN)
FD3505	DEC (IY + d)	CA8405	JP Z, NN	018405	LD BC, NN
3D	DEC A	382E	JR C, DIS	4E	LD C, (HL)
05	DEC B	182E	JR DIS	DD4E05	LD C, (IX + d)
0B	DEC BC	302E	JR NC, DIS	FD4E05	LD C, (IY + d)
0D	DEC C	202E	JR NZ, DIS	4F	LD C, A
15	DEC D	282E	JR Z, DIS	48	LD C, B
1B	DEC DE	02	LD (BC), A	49	LD C, C
1D	DEC E	12	LD (DE), A	4A	LD C, D
25	DEC H	77	LD (HL), A	4B	LD C, E
2B	DEC HL	70	LD (HL), B	4C	LD C, H

Reprinted courtesy of Zilog

OBJ CODE	SOURCE STATEMENT
4D	LD C, L
0E20	LD C, N
56	LD D, (HL)
DD5605	LD D, (IX + d)
FD5605	LD D, (IY + d)
57	LD D, A
50	LD D, B
51	LD D, C
52	LD D, D
53	LD D, E
54	LD D, H
55	LD D, L
1620	LD D, N
ED5B8405	LD DE, (NN)
118405	LD DE, NN
5E	LD E, (HL)
DD5E05	LD E, (IX + d)
FD5E05	LD E, (IY + d)
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
1E20	LD E, N
66	LD H, (HL)
DD6605	LD H, (IX + d)
FD6606	LD H, (IY + d)
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
2620	LD H, N
2A8405	LD HL, (NN)
218405	LD HL, NN
ED47	LD I, A
DD2A8405	LD IX, (NN)
DD218405	LD IX, NN
FD2A8405	LD IY, (NN)
FD218405	LD IY, NN
6E	LD L, (HL)
DD6E05	LD L, (IX + d)
FD6E05	LD L, (IY + d)
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
2E20	LD L, N
ED7B8405	LD SP, (NN)
F9	LD SP, HL
DDF9	LD SP, IX
FDF9	LD SP, IY
318405	LD SP, NN
EDA8	LDD
EDB8	LDDR
EDA0	LDI
EDB0	LDIR
ED44	NEG
00	NOP
B6	OR (HL)
DDB605	OR (IX + d)
FDB605	OR (IY + d)
B7	OR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
F620	OR N
EDB8	OTDR
EDB3	OTIR
ED79	OUT (C), A
ED41	OUT (C), B
ED49	OUT (C), C
ED51	OUT (C), D
ED59	OUT (C), E
ED61	OUT (C), H
ED69	OUT (C), L
D320	OUT (N), A
EDAB	OUTD
EDA3	OUTI
F1	POP AF
C1	POP BC
D1	POP DE
E1	POP HL
DDE1	POP IX
FDE1	POP IY
F5	PUSH AF
C5	PUSH BC
D5	PUSH DE
E5	PUSH HL
DDE5	PUSH IX
FDE5	PUSH IY
CB86	RES 0, (HL)
DDCB0586	RES 0, (IX + d)
FDCB0586	RES 0, (IY + d)
CB87	RES 0, A
CB80	RES 0, B
CB81	RES 0, C
CB82	RES 0, D
CB83	RES 0, E
CB84	RES 0, H
CB85	RES 0, L
CB8E	RES 1, (HL)
DDCB058E	RES 1, (IX + d)
FDCB058E	RES 1, (IY + d)
CB8F	RES 1, A
CB88	RES 1, B
CB89	RES 1, C
CB8A	RES 1, D
CB8B	RES 1, E
CB8C	RES 1, H
CB8D	RES 1, L
CB96	RES 2, (HL)
DDCB0596	RES 2, (IX + d)
FDCB0596	RES 2, (IY + d)
CB97	RES 2, A
CB90	RES 2, B
CB91	RES 2, C
CB92	RES 2, D
CB93	RES 2, E
CB94	RES 2, H
CB95	RES 2, L
CB9E	RES 3, (HL)
DDCB059E	RES 3, (IX + d)
FDCB059E	RES 3, (IY + d)
CB9F	RES 3, A
CB98	RES 3, B
CB99	RES 3, C
CB9A	RES 3, D
CB9B	RES 3, E
CB9C	RES 3, H
CB9D	RES 3, L
CBA6	RES 4, (HL)
DDCB05A6	RES 4, (IX + d)
FDCB05A6	RES 4, (IY + d)
CBA7	RES 4, A
CBA0	RES 4, B
CBA1	RES 4, C
CBA2	RES 4, D
CBA3	RES 4, E
CBA4	RES 4, H
CBA5	RES 4, L
CBAE	RES 5, (HL)
DDCB05AE	RES 5, (IX + d)
FDCB05AE	RES 5, (IY + d)
CBAF	RES 5, A
CBA8	RES 5, B
CBA9	RES 5, C
CBAA	RES 5, D
CBAB	RES 5, E
CBAC	RES 5, H
CBAD	RES 5, L
CBB6	RES 6, (HL)
DDCB05B6	RES 6, (IX + d)
FDCB05B6	RES 6, (IY + d)
CBB7	RES 6, A
CBB0	RES 6, B
CBB1	RES 6, C
CBB2	RES 6, D
CBB3	RES 6, E
CBB4	RES 6, H
CBB5	RES 6, L
CBBE	RES 7, (HL)
DDCB05BE	RES 7, (IX + d)
FDCB05BE	RES 7, (IY + d)
CBBF	RES 7, A
CBB8	RES 7, B
CBB9	RES 7, C
CBBA	RES 7, D
CBBB	RES 7, E
CBBC	RES 7, H
CBBD	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
ED4D	RETI
ED45	RETN
CB16	RL (HL)
DDCB0516	RL (IX + d)
FDCB0516	RL (IY + d)
CB17	RL A
CB10	RL B
CB11	RL C
CB12	RL D
CB13	RL E

Reprinted courtesy of Zilog

OBJ CODE	SOURCE STATEMENT
CB14	RL H
CB15	RL L
17	RLA
CB06	RLC (HL)
DDCB0506	RLC (IX + d)
FDCB0506	RLC (IY + d)
CB07	RLC A
CB00	RLC B
CB01	RLC C
CB02	RLC D
CB03	RLC E
CB04	RLC H
CB05	RLC L
07	RLCA
ED6F	RLD
CB1E	RR (HL)
DDCB051E	RR (IX + d)
FDCB051E	RR (IY + d)
CB1F	RR A
CB18	RR B
CB19	RR C
CB1A	RR D
CB1B	RR E
CB1C	RR H
CB1D	RR L
1F	RRA
CB0E	RRC (HL)
DDCB050E	RRC (IX + d)
FDCB050E	RRC (IY + d)
CB0F	RRC A
CB08	RRC B
CB09	RRC C
CB0A	RRC D
CB0B	RRC E
CB0C	RRC H
CB0D	RRC L
0F	RRCA
ED67	RRD
C7	RST 0
D7	RST 10H
DF	RST 18H
E7	RST 20H
EF	RST 28H
F7	RST 30H
FF	RST 38H
CF	RST 8
9E	SBC A, (HL)
DD9E05	SBC A, (IX + d)
FD9E05	SBC A, (IY + d)
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
DE20	SBC A, N
ED42	SBC HL, BC
ED52	SBC HL, DE
ED62	SBC HL, HL
ED72	SBC HL, SP
37	SCF
CBC6	SET 0, (HL)
DDCB05C6	SET 0, (IX + d)
FDCB05C6	SET 0, (IY + d)
CBC7	SET 0, A
CBC0	SET 0, B
CBC1	SET 0, C
CBC2	SET 0, D
CBC3	SET 0, E
CBC4	SET 0, H
CBC5	SET 0, L
CBCE	SET 1, (HL)
DDCB05CE	SET 1, (IX + d)
FDCB05CE	SET 1, (IY + d)
CBCF	SET 1, A
CBC8	SET 1, B
CBC9	SET 1, C
CBCA	SET 1, D
CBCB	SET 1, E
CBCC	SET 1, H
CBCD	SET 1, L
CBD6	SET 2, (HL)
DDCB05D6	SET 2, (IX + d)
FDCB05D6	SET 2, (IY + d)
CBD7	SET 2, A
CBD0	SET 2, B
CBD1	SET 2, C
CBD2	SET 2, D
CBD3	SET 2, E
CBD4	SET 2, H
CBD5	SET 2, L
CBD8	SET 3, B
CBDE	SET 3, (HL)
DDCB05DE	SET 3, (IX + d)
FDCB05DE	SET 3, (IY + d)
CBDF	SET 3, A
CBD9	SET 3, C
CBDA	SET 3, D
CBDB	SET 3, E
CBDC	SET 3, H
CBDD	SET 3, L
CBE6	SET 4, (HL)
DDCB05E6	SET 4, (IX + d)
FDCB05E6	SET 4, (IY + d)
CBE7	SET 4, A
CBE0	SET 4, B
CBE1	SET 4, C
CBE2	SET 4, D
CBE3	SET 4, E
CBE4	SET 4, H
CBE5	SET 4, L
CBEE	SET 5, (HL)
DDCB05EE	SET 5, (IX + d)
FDCB05EE	SET 5, (IY + d)
CBEF	SET 5, A
CBE8	SET 5, B
CBE9	SET 5, C
CBEA	SET 5, D
CBEB	SET 5, E
CBEC	SET 5, H
CBED	SET 5, L
CBF6	SET 6, (HL)
DDCB05F6	SET 6, (IX + d)
FDCB05F6	SET 6, (IY + d)
CBF7	SET 6, A
CBF0	SET 6, B
CBF1	SET 6, C
CBF2	SET 6, D
CBF3	SET 6, E
CBF4	SET 6, H
CBF5	SET 6, L
CBFE	SET 7, (HL)
DDCB05FE	SET 7, (IX + d)
FDCB05FE	SET 7, (IY + d)
CBFF	SET 7, A
CBF8	SET 7, B
CBF9	SET 7, C
CBFA	SET 7, D
CBFB	SET 7, E
CBFC	SET 7, H
CBFD	SET 7, L
CB26	SLA (HL)
DDCB0526	SLA (IX + d)
FDCB0526	SLA (IY + d)
CB27	SLA A
CB20	SLA B
CB21	SLA C
CB22	SLA D
CB23	SLA E
CB24	SLA H
CB25	SLA L
CB2E	SRA (HL)
DDCB052E	SRA (IX + d)
FDCB052E	SRA (IY + d)
CB2F	SRA A
CB28	SRA B
CB29	SRA C
CB2A	SRA D
CB2B	SRA E
CB2C	SRA H
CB2D	SRA L
CB3E	SRL (HL)
DDCB053E	SRL (IX + d)
FDCB053E	SRL (IY + d)
CB3F	SRL A
CB38	SRL B
CB39	SRL C
CB3A	SRL D
CB3B	SRL E
CB3C	SRL H
CB3D	SRL L
96	SUB (HL)
DD9605	SUB (IX + d)
FD9605	SUB (IY + d)
97	SUB A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
D620	SUB N
AE	XOR (HL)
DDAE05	XOR (IX + d)
FDAE05	XOR (IY + d)
AF	XOR A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
EE20	XOR N

Reprinted courtesy of Zilog

A.1.2 Z80 INSTRUCTIONS SORTED BY OP-CODE

OBJ CODE	SOURCE STATEMENT
00	NOP
018405	LD BC, NN
02	LD (BC), A
03	INC BC
04	INC B
05	DEC B
0620	LD B, N
07	RLCA
08	EX AF, AF'
09	ADD HL, BC
0A	LD A, (BC)
0B	DEC BC
0C	INC C
0D	DEC C
0E20	LD C, N
0F	RRCA
102E	DJNZ DIS
118405	LD DE, NN
12	LD (DE), A
13	INC DE
14	INC D
15	DEC D
1620	LD D, N
17	RLA
182E	JR DIS
19	ADD HL, DE
1A	LD A, (DE)
1B	DEC DE
1C	INC E
1D	DEC E
1E20	LD E, N
1F	RRA
202E	JR NZ, DIS
218405	LD HL, NN
228405	LD (NN), HL
23	INC HL
24	INC H
25	DEC H
2620	LD H, N
27	DAA
282E	JR Z, DIS
29	ADD HL, HL
2A8405	LD (HL), (NN)
2B	DEC HL
2C	INC L
2D	DEC L
2E20	LD L, N
2F	CPL
302E	JR NC, DIS
318405	LD SP, NN
328405	LD (NN), A
33	INC SP
34	INC (HL)
35	DEC (HL)
3620	LD (HL), N
37	SCF
382E	JR C, DIS
39	ADD HL, SP
3A8405	LD A, (NN)
3B	DEC SP
3C	INC A
3D	DEC A
3E20	LD A, N
3F	CCF
40	LD B, B
41	LD B, C
42	LD B, D
43	LD B, E
44	LD B, H, NN
45	LD B, L
46	LD B, (HL)
47	LD B, 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
4E	LD C, (HL)
4F	LD C, A
50	LD D, B
51	LD D, C
52	LD D, D
53	LD D, E
54	LD D, H
55	LD D, L
56	LD D, (HL)
57	LD D, 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
5E	LD E, (HL)
5F	LD E, 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
66	LD H, (HL)
67	LD H, 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
6E	LD L, (HL)
6F	LD L, 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
76	HALT
77	LD (HL), A
78	LD A, B
79	LD A, C
7A	LD A, D
7B	LD A, E
7C	LD A, H
7D	LD A, L
7E	LD A, (HL)
7F	LD A, A
80	ADD A, B
81	ADD A, C
82	ADD A, D
83	ADD A, E
84	ADD A, H
85	ADD A, L
86	ADD A, (HL)
87	ADD A, A
88	ADC A, B
89	ADC A, C
8A	ADC A, D
8B	ADC A, E
8C	ADC A, H
8D	ADC A, L
8E	ADC A, (HL)
8F	ADC A, A
90	SUB B
91	SUB C
92	SUB D
93	SUB E
94	SUB H
95	SUB L
96	SUB (HL)
97	SUB 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
9E	SBC A, (HL)
9F	SBC A, A
A0	AND B
A1	AND C
A2	AND D
A3	AND E
A4	AND H
A5	AND L
A6	AND (HL)
A7	AND A
A8	XOR B
A9	XOR C
AA	XOR D
AB	XOR E
AC	XOR H
AD	XOR L
AE	XOR (HL)
AF	XOR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
B6	OR (HL)
B7	OR A
B8	CP B
B9	CP C
BA	CP D
BB	CP E
BC	CP H
BD	CP L
BE	CP (HL)
BF	CP A
C0	RET NZ
C1	POP BC
C28405	JP NZ, NN
C38405	JP NN
C48405	CALL NZ, NN
C5	PUSH BC
C620	ADD A, N
C7	RST 0
C8	RET Z
C9	RET
CA8405	JP Z, NN
CC8405	CALL Z, NN
CD8405	CALL NN
CE20	ADC A, N
CF	RST 8
D0	RET NC
D1	POP DE
D28405	JP NC, NN
D320	OUT (N), A
D48405	CALL NC, NN
D5	PUSH DE
D620	SUB N
D7	RST 10H
D8	RET C
D9	EXX
DA8405	JP C, NN
DB20	IN A, (N)
DC8405	CALL C, N
DE20	SBC A, N
DF	RST 18H
E0	RET PO
E1	POP HL
E28405	JP PO, NN
E3	EX (SP), HL
E48405	CALL PO, NN
E5	PUSH HL
E620	AND N
E7	RST 20 H
E8	RET PE
E9	JP (HL)
EA8405	JE PE NN
EB	EX DE, HL
EC8405	CALL PE, NN
EE20	XOR N
EF	RST 28H
F0	RET P
F1	POP AF
F28405	JP P, NN
F3	DI
F48405	CALL P, NN
F5	PUSH AF
F620	OR N
F7	RST 30H
F8	RET M
F9	LD SP, HL
FA8405	JP M, NN
FB	EI
FC8405	CALL M, NN
FE20	CP N
FF	RST 38H
CB00	RLC B
CB01	RLC C
CB02	RLC D
CB03	RLC E
CB04	RLC H
CB05	RLC L
CB06	RLC (HL)
CB07	RLC A
CB08	RRC B
CB09	RRC C
CB0A	RRC D
CB0B	RRC E
CB0C	RRC H
CB0D	RRC L
CB0E	RRC (HL)
CB0F	RRC A
CB10	RL B
CB11	RL C
CB12	RL D
CB13	RL E
CB14	RL H
CB15	RL L
CB16	RL (HL)
CB17	RL A
CB18	RR B
CB19	RR C
CB1A	RR D
CB1B	RR E
CB1C	RR H
CB1D	RR L
CB1E	RR (HL)
CB1F	RR A
CB20	SLA B
CB21	SLA C
CB22	SLA D
CB23	SLA E
CB24	SLA H
CB25	SLA L
CB26	SLA (HL)
CB27	SLA A
CB28	SRA B
CB29	SRA C
CB2A	SRA D
CB2B	SRA E
CB2C	SRA H

Reprinted courtesy of Zilog

OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT
CB2D	SRA L	CB77	BIT 6, A	CB89	RES 7, C
CB2E	SRA (HL)	CB78	BIT 7, B	CB8A	RES 7, D
CB2F	SRA A	CB79	BIT 7, C	CB8B	RES 7, E
CB38	SRL B	CB7A	BIT 7, D	CB8C	RES 7, H
CB39	SRL C	CB7B	BIT 7, E	CB8D	RES 7, L
CB3A	SRL D	CB7C	BIT 7, H	CB8E	RES 7, (HL)
CB3B	SRL E	CB7D	BIT 7, L	CB8F	RES 7, A
CB3C	SRL H	CB7E	BIT 7, (HL)	CBC0	SET 0, B
CB3D	SRL L	CB7F	BIT 7, A	CBC1	SET 0, C
CB3E	SRL (HL)	CB80	RES 0, B	CBC2	SET 0, D
CB3F	SRL A	CB81	RES 0, C	CBC3	SET 0, E
CB40	BIT 0, B	CB82	RES 0, D	CBC4	SET 0, H
CB41	BIT 0, C	CB83	RES 0, E	CBC5	SET 0, L
CB42	BIT 0, D	CB84	RES 0, H	CBC6	SET 0, (HL)
CB43	BIT 0, E	CB85	RES 0, L	CBC7	SET 0, A
CB44	BIT 0, H	CB86	RES 0, (HL)	CBC8	SET 1, B
CB45	BIT 0, L	CB87	RES 0, A	CBC9	SET 1, C
CB46	BIT 0, (HL)	CB88	RES 1, B	CBCA	SET 1, D
CB47	BIT 0, A	CB89	RES 1, C	CBCB	SET 1, E
CB48	BIT 1, B	CB8A	RES 1, D	CBCC	SET 1, H
CB49	BIT 1, C	CB8B	RES 1, E	CBCD	SET 1, L
CB4A	BIT 1, D	CB8C	RES 1, H	CBCE	SET 1, (HL)
CB4B	BIT 1, E	CB8D	RES 1, L	CBCF	SET 1, A
CB4C	BIT 1, H	CB8E	RES 1, (HL)	CBD0	SET 2, B
CB4D	BIT 1, L	CB8F	RES 1, A	CBD1	SET 2, C
CB4E	BIT 1, (HL)	CB90	RES 2, B	CBD2	SET 2, D
CB4F	BIT 1, A	CB91	RES 2, C	CBD3	SET 2, E
CB50	BIT 2, B	CB92	RES 2, D	CBD4	SET 2, H
CB51	BIT 2, C	CB93	RES 2, E	CBD5	SET 2, L
CB52	BIT 2, D	CB94	RES 2, H	CBD6	SET 2, (HL)
CB53	BIT 2, E	CB95	RES 2, L	CBD7	SET 2, A
CB54	BIT 2, H	CB96	RES 2, (HL)	CBD8	SET 3, B
CB55	BIT 2, L	CB97	RES 2, A	CBD9	SET 3, C
CB56	BIT 2, (HL)	CB98	RES 3, B	CBDA	SET 3, D
CB57	BIT 2, A	CB99	RES 3, C	CBDB	SET 3, E
CB58	BIT 3, B	CB9A	RES 3, D	CBDC	SET 3, H
CB59	BIT 3, C	CB9B	RES 3, E	CBDD	SET 3, L
CB5A	BIT 3, D	CB9C	RES 3, H	CBDE	SET 3, (HL)
CB5B	BIT 3, E	CB9D	RES 3, L	CBDF	SET 3, A
CB5C	BIT 3, H	CB9E	RES 3, (HL)	CBE0	SET 4, B
CB5D	BIT 3, L	CB9F	RES 3, A	CBE1	SET 4, C
CB5E	BIT 3, (HL)	CBA0	RES 4, B	CBE2	SET 4, D
CB5F	BIT 3, A	CBA1	RES 4, C	CBE3	SET 4, E
CB60	BIT 4, B	CBA2	RES 4, D	CBE4	SET 4, H
CB61	BIT 4, C	CBA3	RES 4, E	CBE5	SET 4, L
CB62	BIT 4, D	CBA4	RES 4, H	CBE6	SET 4, (HL)
CB63	BIT 4, E	CBA5	RES 4, L	CBE7	SET 4, A
CB64	BIT 4, H	CBA6	RES 4, (HL)	CBE8	SET 5, B
CB65	BIT 4, L	CBA7	RES 4, A	CBE9	SET 5, C
CB66	BIT 4, (HL)	CBA8	RES 5, B	CBEA	SET 5, D
CB67	BIT 4, A	CBA9	RES 5, C	CBEB	SET 5, E
CB68	BIT 5, B	CBAA	RES 5, D	CBEC	SET 5, H
CB69	BIT 5, C	CBAB	RES 5, E	CBED	SET 5, L
CB6A	BIT 5, D	CBAC	RES 5, H	CBEE	SET 5, (HL)
CB6B	BIT 5, E	CBAD	RES 5, L	CBEF	SET 5, A
CB6C	BIT 5, H	CBAE	RES 5, (HL)	CBF0	SET 6, B
CB6D	BIT 5, L	CBAF	RES 5, A	CBF1	SET 6, C
CB6E	BIT 5, (HL)	CBB0	RES 6, B	CBF2	SET 6, D
CB6F	BIT 5, A	CBB1	RES 6, C	CBF3	SET 6, E
CB70	BIT 6, B	CBB2	RES 6, D	CBF4	SET 6, H
CB71	BIT 6, C	CBB3	RES 6, E	CBF5	SET 6, L
CB72	BIT 6, D	CBB4	RES 6, H	CBF6	SET 6, (HL)
CB73	BIT 6, E	CBB5	RES 6, L	CBF7	SET 6, A
CB74	BIT 6, H	CBB6	RES 6, (HL)	CBF8	SET 7, B
CB75	BIT 6, L	CBB7	RES 6, A	CBF9	SET 7, C
CB76	BIT 6, (HL)	CBB8	RES 7, B	CBFA	SET 7, D

Reprinted courtesy of Zilog

OBJ CODE	SOURCE STATEMENT
CBF8	SET 7, E
CBFC	SET 7, H
CBFD	SET 7, L
CBFE	SET 7, (HL)
CBFF	SET 7, A
DD09	ADD IX, BC
DD19	ADD IX, DE
DD218405	LD IX, NN
DD228405	LD (NN), IX
DD23	INC IX
DD29	ADD IX, IX
DD2A8405	LD IX, (NN)
DD2B	DEC IX
DD3405	INC (IX + d)
DD3505	DEC (IX + d)
DD360520	LD (IX + d), N
DD39	ADD IX, SP
DD4605	LD B, (IX + d)
DD4E05	LD C, (IX + d)
DD5605	LD D, (IX + d)
DD5E05	LD E, (IX + d)
DD6605	LD H, (IX + d)
DD6E05	LD L, (IX + d)
DD7005	LD (IX + d), B
DD7105	LD (IX + d), C
DD7205	LD (IX + d), D
DD7305	LD (IX + d), E
DD7405	LD (IX + d), H
DD7505	LD (IX + d), L
DD7705	LD (IX + d), A
DD7E05	LD A, (IX + d)
DD8605	ADD A, (IX + d)
DD8E05	ADC A, (IX + d)
DD9605	SUB (IX + d)
DD9E05	SBC A, (IX + d)
DDA605	AND (IX + d)
DDAE05	XOR (IX + d)
DDB605	OR (IX + d)
DDBE05	CP (IX + d)
DDE1	POP IX
DDE3	EX (SP), IX
DDE5	PUSH IX
DDE9	JP (IX)
DDF9	LD SP, IX
DDCB0506	RLC (IX + d)
DDCB050E	RRC (IX + d)
DDCB0516	RL (IX + d)
DDCB051E	RR (IX + d)
DDCB0526	SLA (IX + d)
DDCB052E	SRA (IX + d)
DDCB053E	SRL (IX + d)
DDCB0546	BIT 0, (IX + d)
DDCB054E	BIT 1, (IX + d)
DDCB0556	BIT 2, (IX + d)
DDCB055E	BIT 3, (IX + d)
DDCB0566	BIT 4, (IX + d)
DDCB056E	BIT 5, (IX + d)
DDCB0576	BIT 6, (IX + d)
DDCB057E	BIT 7, (IX + d)
DDCB0586	RES 0, (IX + d)
DDCB058E	RES 1, (IX + d)
DDCB0596	RES 2, (IX + d)
DDCB059E	RES 3, (IX + d)
DDCB05A6	RES 4, (IX + d)
DDCB05AE	RES 5, (IX + d)
DDCB0586	RES 6, (IX + d)
DDCB058E	RES 7, (IX + d)
DDCB05C6	SET 0, (IX + d)
DDCB05CE	SET 1, (IX + d)
DDCB05D6	SET 2, (IX + d)
DDCB05DE	SET 3, (IX + d)
DDCB05E6	SET 4, (IX + d)
DDCB05EE	SET 5, (IX + d)
DDCB05F6	SET 6, (IX + d)
DDCB05FE	SET 7, (IX + d)
ED40	IN B, (C)
ED41	OUT (C), B
ED42	SBC HL, BC
ED438405	LD (NN), BC
ED44	NEG
ED45	RETN
ED46	IM 0
ED47	LD I, A
ED48	IN C, (C)
ED49	OUT (C), C
ED4A	ADC HL, BC
ED4B8405	LD BC, (NN)
ED4D	RETI
ED50	IN D, (C)
ED51	OUT (C), D
ED52	SBC HL, DE
ED538405	LD (NN), DE
ED56	IM 1
ED57	LD A, I
ED58	IN E, (C)
ED59	OUT (C), E
ED5A	ADC HL, DE
ED5B8405	LD DE, (NN)
ED5E	IM 2
ED60	IN H, (C)
ED61	OUT (C), H
ED62	SBC HL, HL
ED67	RRD
ED68	IN L, (C)
ED69	OUT (C), L
ED6A	ADC HL, HL
ED6F	RLD
ED72	SBC HL, SP
ED738405	LD (NN), SP
ED78	IN A, (C)
ED79	OUT (C), A
ED7A	ADC HL, SP
ED7B8405	LD SP, (NN)
EDA0	LDI
EDA1	CPI
EDA2	INI
EDA3	OUTI
EDA8	LDD
EDA9	CPD
EDAA	IND
EDAB	OUTD
EDB0	LDIR
EDB1	CPIR
EDB2	INIR
EDB3	OTIR
EDB8	LDDR
EDB9	CPDR
EDBA	INDR
EDBB	OTDR
FD09	ADD IY, BC
FD19	ADD IY, DE
FD218405	LD IY, NN
FD228405	LD (NN), IY
FD23	INC IY
FD29	ADD IY, IY
FD2A8405	LD IY, (NN)
FD2B	DEC IY
FD3405	INC (IY + d)
FD3505	DEC (IY + d)
FD360520	LD (IY + d), N
FD39	ADD IY, SP
FD4605	LD B, (IY + d)
FD4E05	LD C, (IY + d)
FD5605	LD D, (IY + d)
FD5E05	LD E, (IY + d)
FD6605	LD H, (IY + d)
FD6E05	LD L, (IY + d)
FD7005	LD (IY + d), B
FD7105	LD (IY + d), C
FD7205	LD (IY + d), D
FD7305	LD (IY + d), E
FD7405	LD (IY + d), H
FD7505	LD (IY + d), L
FD7705	LD (IY + d), A
FD7E05	LD A, (IY + d)
FD8605	ADD A, (IY + d)
FD8E05	ADC A, (IY + d)
FD9605	SUB (IY + d)
FD9E05	SBC A, (IY + d)
FDA605	AND (IY + d)
FDAE05	XOR (IY + d)
FDB605	OR (IY + d)
FDBE05	CP (IY + d)
FDE1	POP IY
FDE3	EX (SP), IY
FDE5	PUSH IY
FDE9	JP (IY)
FDF9	LD SP, IY
FDCB0506	RLC (IY + d)
FDCB050E	RRC (IY + d),
FDCB0516	RL (IY + d)
FDCB051E	RR (IY + d)
FDCB0526	SLA (IY + d)
FDCB052E	SRA (IY + d)
FDCB053E	SRL (IY + d)
FDCB0546	BIT 0, (IY + d)
FDCB054E	BIT 1, (IY + d)
FDCB0556	BIT 2, (IY + d)
FDCB055E	BIT 3, (IY + d)
FDCB0566	BIT 4, (IY + d)
FDCB056E	BIT 5, (IY + d)
FDCB0576	BIT 6, (IY + d)
FDCB057E	BIT 7, (IY + d)
FDCB0586	RES 0, (IY + d)
FDCB058E	RES 1, (IY + d)
FDCB0596	RES 2, (IY + d)
FDCB059E	RES 3, (IY + d)
FDCB05A6	RES 4, (IY + d)
FDCB05AE	RES 5, (IY + d)
FDCB0586	RES 6, (IY + d)
FDCB05C6	SET 0, (IY + d)
FDCB05CE	SET 1, (IY + d)
FDCB05D6	SET 2, (IY + d)
FDCB05DE	SET 3, (IY + d)
FDCB05E6	SET 4, (IY + d)
FDCB05EE	SET 5, (IY + d)
FDCB05F6	SET 6, (IY + d)
FDCB05FE	SET 7, (IY + d)

Reprinted courtesy of Zilog

8-BIT LOAD GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments r, s	Reg.
LD r, s	r ← s	•	•	X	•	X	•	•	•	01 r s		1	1	4	r, s	Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000	B
										← n →					001	C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010	D
LD r, (IX+d)	r ← (IX+d)	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	011	E
										01 r 110					100	H
										← d →					101	L
LD r, (IY+d)	r ← (IY+d)	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	111	A
										01 r 110						
										← d →						
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7		
LD (IX+d), r	(IX+d) ← r	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19		
										01 110 r						
										← d →						
LD (IY+d), r	(IY+d) ← r	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19		
										01 110 r						
										← d →						
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110	36	2	3	10		
										← n →						
LD (IX+d), n	(IX+d) ← n	•	•	X	•	X	•	•	•	11 011 101	DD	4	5	19		
										00 110 110	36					
										← d →						
										← n →						
LD (IY+d), n	(IY+d) ← n	•	•	X	•	X	•	•	•	11 111 101	FD	4	5	19		
										00 110 110	36					
										← d →						
										← n →						
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7		
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7		
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010	3A	3	4	13		
										← n →						
										← n →						
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7		
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7		
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010	32	3	4	13		
										← n →						
										← n →						
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9		
										01 010 111	57					
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9		
										01 011 111	5F					
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9		
										01 000 111	47					
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9		
										01 001 111	4F					

Notes: r, s means any of the registers A, B, C, D, E, H, L
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

16-BIT LOAD GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001 ← n → ← n →		3	3	10	dd Pair 00 BC 01 DE
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	10 HL 11 SP
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	X	•	X	•	•	•	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	X	•	X	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	ED	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	X	•	X	•	•	•	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	X	•	X	•	•	•	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	X	•	X	•	•	•	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	X	•	X	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	ED	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	X	•	X	•	•	•	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	X	•	X	•	•	•	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101 11 111 001	FD F9	2	2	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	X	•	X	•	•	•	11 qq0 101		1	3	11	qq Pair 00 BC 01 DE
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	X	•	X	•	•	•	11 011 101 11 100 101	DD E5	2	4	15	10 HL 11 AF
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	X	•	X	•	•	•	11 111 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	X	•	X	•	•	•	11 qq0 001		1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 001	FD E1	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
e.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ flag is affected according to the result of the operation.

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE↔HL	•	•	X	•	X	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF↔AF'	•	•	X	•	X	•	•	•	00 001 000	08	1	1	4	
EXX	(BC↔BC' DE↔DE' HL↔HL')	•	•	X	•	X	•	•	•	11 011 001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔(SP+1) L ↔(SP)	•	•	X	•	X	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H ↔(SP+1) IX_L ↔(SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H ↔(SP+1) IY_L ↔(SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 100 000	ED A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A – (HL) HL ← HL+1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A – (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A – (HL) HL ← HL-1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A – (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc, content into packed BCD following add or subtract with packed BCD operands	↕	↕	X	↕	X	P	•	↕	00 100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	X	1	X	•	1	•	00 101 111	2F	1	1	4	Complement accumulator (One's complement)
NEG	$A \leftarrow \overline{A}+1$	↕	↕	X	↕	X	V	1	↕	11 101 101 01 000 100	ED 44	2	2	8	Negate acc, (two's complement)
CCF	$CY \leftarrow \overline{CY}$	•	•	X	X	X	•	0	↕	00 111 111	3F	1	1	4	Complement carry flag
SCF	CY ← 1	•	•	X	0	X	•	0	1	00 110 111	37	1	1	4	Set carry flag
NOP	No operation	•	•	X	•	X	•	•	•	00 000 000	00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110	76	1	1	4	
DI*	IFF ← 0	•	•	X	•	X	•	•	•	11 110 011	F3	1	1	4	
EI*	IFF ← 1	•	•	X	•	X	•	•	•	11 111 011	FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 01 000 110	ED 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 01 010 110	ED 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 01 011 110	ED 5E	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

*Interrupts are not sampled at the end of EI or DI

8-BIT ARITHMETIC AND LOGICAL GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	X	↕	X	V	0	↕	10 [000] r		1	1	4	r Reg.
ADD A, n	A ← A + n	↕	↕	X	↕	X	V	0	↕	11 [000] 110		2	2	7	000 B
										← n →					001 C
															010 D
ADD A, (HL)	A ← A+(HL)	↕	↕	X	↕	X	V	0	↕	10 [000] 110		1	2	7	011 E
ADD A, (IX+d)	A ← A+(IX+d)	↕	↕	X	↕	X	V	0	↕	11 011 101	DD	3	5	19	100 H
										10 [000] 110					101 L
										← d →					111 A
ADD A, (IY+d)	A ← A+(IY+d)	↕	↕	X	↕	X	V	0	↕	11 111 101	FD	3	5	19	
										10 [000] 110					
										← d →					
ADC A, s	A ← A+s+CY	↕	↕	X	↕	X	V	0	↕	[001]					s is any of r, n,
SUB s	A ← A - s	↕	↕	X	↕	X	V	1	↕	[010]					(HL), (IX+d),
SBC A, s	A ← A - s - CY	↕	↕	X	↕	X	V	1	↕	[011]					(IY+d) as shown for
AND s	A ← A ∧ s	↕	↕	X	1	X	P	0	0	[100]					ADD instruction.
OR s	A ← A ∨ s	↕	↕	X	0	X	P	0	0	[110]					The indicated bits
XOR s	A ← A ⊕ s	↕	↕	X	0	X	P	0	0	[101]					replace the [000] in
CP s	A - s	↕	↕	X	↕	X	V	1	↕	[111]					the ADD set above.
INC r	r ← r + 1	↕	↕	X	↕	X	V	0	•	00 r [100]		1	1	4	
INC (HL)	(HL) ← (HL)+1	↕	↕	X	↕	X	V	0	•	00 110 [100]		1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	↕	↕	X	↕	X	V	0	•	11 011 101	DD	3	6	23	
										00 110 [100]					
										← d →					
INC (IY+d)	(IY+d) ← (IY+d)+1	↕	↕	X	↕	X	V	0	•	11 111 101	FD	3	6	23	
										00 110 [100]					
										← d →					
DEC s	s ← s - 1	↕	↕	X	↕	X	V	1	•	[101]					s is any of r, (HL),
															(IX+d), (IY+d) as
															shown for INC.
															DEC same format
															and states as INC.
															Replace [100] with
															[101] in OP Code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow, P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

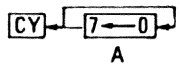
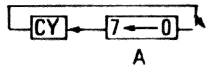
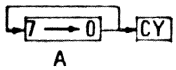
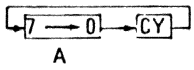
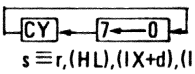
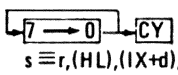
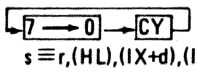
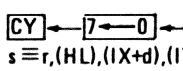
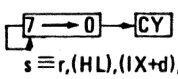
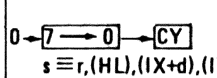
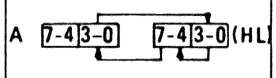
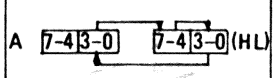
16-BIT ARITHMETIC GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	↕	00 ss1 001		1	3	11	ss Reg. 00 BC
ADC HL, ss	HL ← HL+ss+CY	↕	↕	X	X	X	V	0	↕	11 101 101 01 ss1 010	ED	2	4	15	01 DE 10 HL 11 SP
SBC HL, ss	HL ← HL-ss-CY	↕	↕	X	X	X	V	1	↕	11 101 101 01 ss0 010	ED	2	4	15	
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101 00 pp1 001	DD	2	4	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101 00 rr1 001	FD	2	4	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101 00 100 011	DD 23	2	2	10	
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101 00 100 011	FD 23	2	2	10	
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6	
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101 00 101 011	DD 2B	2	2	10	
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101 00 101 011	FD 2B	2	2	10	

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

ROTATE AND SHIFT GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		•	•	X	0	X	•	0	‡	00 000 111	07	1	1	4	Rotate left circular accumulator
RLA		•	•	X	0	X	•	0	‡	00 010 111	17	1	1	4	Rotate left accumulator
RRCA		•	•	X	0	X	•	0	‡	00 001 111	0F	1	1	4	Rotate right circular accumulator
RRA		•	•	X	0	X	•	0	‡	00 011 111	1F	1	1	4	Rotate right accumulator
RLC r		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 r	CB	2	2	8	Rotate left circular register r
RLC (HL)		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 110	CB	2	4	15	r Reg. 000 B 001 C
RLC (IX+d)	r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	11 011 101 11 001 011 ← d → 00 000 110	DD CB	4	6	23	010 D 011 E 100 H 101 L 111 A
RLC (IY+d)		‡	‡	X	0	X	P	0	‡	11 111 101 11 001 011 ← d → 00 000 110	FD CB	4	6	23	
RL s	 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	010					Instruction format and states are as shown for RLC's. To form new Op-Code replace 000 of RLC's with shown code
RRC s	 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	001					
RR s	 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	011					
SLA s	 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	100					
SRA s	 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	101					
SRL s	 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	111					
RLD		‡	‡	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	Rotate digit left and right between the accumulator and location (HL).
RRD		‡	‡	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	The content of the upper half of the accumulator is unaffected

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

BIT SET, RESET AND TEST GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	X	‡	X	1	X	X	0	•	11 001 011 01 b r	CB	2	2	8	
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	X	‡	X	1	X	X	0	•	11 001 011 01 b 110	CB	2	3	12	
BIT b, $(IX+d)_b$	$Z \leftarrow \overline{(IX+d)_b}$	X	‡	X	1	X	X	0	•	11 011 101 11 001 011 ← d → 01 b 110	DD CB	4	5	20	
BIT b, $(IY+d)_b$	$Z \leftarrow \overline{(IY+d)_b}$	X	‡	X	1	X	X	0	•	11 111 101 11 001 011 ← d → 01 b 110	FD CB	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 [11] b r	CB	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 [11] b 110	CB	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 011 101 11 001 011 ← d → [11] b 110	DD CB	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 101 11 001 011 ← d → [11] b 110	FD CB	4	6	23	
RES b, s	$s_b \leftarrow 0$ $s \equiv r, (HL), (IX+d), (IY+d)$	•	•	X	•	X	•	•	•	[10]					To form new O_p-Code replace [11] of SET b, s with [10] Flags and time states for SET instruction

r	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	Bit Tested
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notes: The notation s_b indicates bit b (0 to 7) of location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 ← n → ← n →		3	3	10	cc | Condition 000 | NZ non zero 001 | Z zero 010 | NC non carry 011 | C carry 100 | PO parity odd 101 | PE parity even 110 | P sign positive 111 | M sign negative
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	00 011 000 ← e-2 →	18	2	3	12	
JR C, e	If C = 0, continue	•	•	X	•	X	•	•	•	00 111 000 ← e-2 →	38	2	2	7	If condition not met
	If C = 1, PC ← PC+e											2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	X	•	X	•	•	•	00 110 000 ← e-2 →	30	2	2	7	If condition not met
	If C = 0, PC ← PC+e											2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	X	•	X	•	•	•	00 101 000 ← e-2 →	28	2	2	7	If condition not met
	If Z = 1, PC ← PC+e											2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	X	•	X	•	•	•	00 100 000 ← e-2 →	20	2	2	7	If condition not met
	If Z = 0, PC ← PC+e											2	3	12	If condition is met
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101 11 101 001	FD E9	2	2	8	
DJNZ, e	B ← B-1 If B = 0, continue	•	•	X	•	X	•	•	•	00 010 000 ← e-2 →	10	2	2	8	If B = 0
	If B ≠ 0, PC ← PC+e											2	3	13	If B ≠ 0

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <126, 129>

e-2 in the op-code provides an effective address of pc+e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

CALL AND RETURN GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC \leftarrow nn$	•	•	X	•	X	•	•	•	11 001 101 ← n ← ← n ←	CD	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	X	•	X	•	•	•	11 cc 100 ← n ← ← n ←		3 3	3 5	10 17	If cc is false If cc is true
RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$	•	•	X	•	X	•	•	•	11 001 001	C9	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	X	•	X	•	•	•	11 cc 000		1 1	1 3	5 11	If cc is false If cc is true
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 101 101 01 001 101	ED 4D	2	4	14	
RETN[1]	Return from non maskable interrupt	•	•	X	•	X	•	•	•	11 101 101 01 000 101	ED 45	2	4	14	
RST p	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC_H \leftarrow 0$ $PC_L \leftarrow p$	•	•	X	•	X	•	•	•	11 t 111		1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

[1] RETN loads $IFF_2 \rightarrow IFF_1$

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

INPUT AND OUTPUT GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 ← n ←	DB	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 01 r 000	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 100 010	ED A2	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1 ①	X	X	X	X	1	•	11 101 101 10 110 010	ED B2	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 101 010	ED AA	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1 ①	X	X	X	X	1	•	11 101 101 10 111 010	ED BA	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11 010 011	D3	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 101 101 01 r 001	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 100 011	ED A3	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1 ①	X	X	X	X	1	•	11 101 101 10 110 011	ED B3	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	↕ ①	X	X	X	X	1	•	11 101 101 10 101 011	ED AB	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1 ①	X	X	X	X	1	•	11 101 101 10 111 011	ED BB	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX 2

SAMPLE PROGRAMS

SAMPLE PROGRAMS

In this appendix a number of example programs are given which, it is hoped, will help the novice understand and write assembly-level programs.

The appendix is split into two parts:

a) Programs which can be viewed using the 'Front Panel' and

b) Programs which can be entered, run, and then viewed/used via the screen and keyboard.

Note that only a few examples are given, and that they attempt to be self-documenting. It should always be the case that programs are as self-documenting as possible, and assembly level programs are far from an exception!!

A.2.1 'FRONT PANEL' PROGRAMS

Five very short programs are given below, more or less just to give a flavour of the size of the programming 'steps' that can be taken at machine level. Also they show a little about program layout. All of the programs can be entered using the 'front panel' of the 380Z and then 'single-stepped' through to see things happening.

```
                ;PROG1
                ;=====
                ;Program that takes two small numbers from two
                ;locations and adds them, leaving the result in
                ;the accumulator.
                ;
0100            ORG 0100H
                ;
0100 3A0801     START:    LD A, (NUM1)   ;Load acc. with cont. of 'NUM1'.
0103 47                   LD B,A         ;Store value in 'B' register.
0104 3A0901               LD A, (NUM2)   ;Load acc. with cont. of 'NUM2'.
0107 80                   ADD A,B        ;Add the two values.
                ;
0108 02         NUM1:     +2             ;Data.
0109 04         NUM2:     +4
                ;
                ;
0000            END


0108  NUM1      0109 NUM2     0100 START

No errors
```

```
                ;PROG2
                ;=====
                ;This program sets all the general purpose
                ;registers to zero by using a series of
                ;'stack' instructions.
                ;
0100            ORG 0100H
                ;
                ;Values:
                ;=======
0000 =                  ZERO EQU 00
                ;
0100 210000     START:  LD HL,ZERO      ;Set HL=0000.
0103 E5                 PUSH HL         ;Place value on stack..
0104 E5                 PUSH HL
0105 E5                 PUSH HL
0106 F1                 POP  AF         ;Take values off
0107 C1                 POP  BC         ;stack, setting
0108 D1                 POP  DE         ;registers=0000.
                ;
                ;
0000            END

0100 START  0000 ZERO

No errors
```

```
                ;PROG3:
                ;=====
                ;Program to successively enter a value
                ;into consecutive locations.
                ;
0100            ORG 0100H
                ;
                ;Values:
                ;======
0110 =                  START EQU 0110H
0006 =                  NUM   EQU 6
00F8 =                  VALUE EQU 0F8H
                ;
0100 3EF8       GO:     LD A,VALUE      ;Value to enter.
0102 211001             LD HL, START    ;Where to put first.
0105 0606               LD B,NUM        ;How many..
0107 77         RP1:    LD (HL),A       ;Store value &
0108 23                 INC HL          ;repeat until
0109 10FC               DJNZ RP1        ;finished.
                ;
                ;Note that 'INC HL' is needed to increment
                :the store address for the value.
                ;
                ;
0000            END

0100 GO     0006 NUM     0107 RP1     0110 START     00F8 VALUE

No errors
```

```
                ;PROG4:
                ;=====
                ;To show a little bit about 'jumps' this
                ;program takes a value in the accumulator
                ;and decrements it, compares the new value
                ;to zero and repeats the operation if no
                ;match occurs.  When the contents of the
                ;accumulator are zero the program moves
                ;on, replacing the original value in
                ;the accumulator.
                ;
0100            ORG 0100H
                ;
                ;Values:
                ;======
0000 =                  ZERO  EQU 0
0005 =                  INIT EQU 5
                ;
                ;
0100 3E05       START:  LD A, INIT      ;Get value.
0102 3D         ST1:    DEC A           ;Decrement and
0103 FE00               CP ZERO         ;compare..
0105 20FB               JR NZ,ST1       ;Repeat or
0107 3E05               LD A, INIT      ;replace & cont..
                ;
                ;
0000            END


0005 INIT    0102 ST1    0100 START    0000 ZERO

No errors
```

```
                ;PROG5:
                ;=====
                ;To illustrate the CALL/RET operation
                ;the following program takes an initial
                ;value into the accumulator and an
                ;'increase' value into the B register.
                ;The accumulator will then be increased
                ;by the value in 'B' via a subroutine
                ;which increments the accumulator 'B'
                ;times.
                ;
0100            ORG 0100H
                ;
                ;Values:
                ;======
0004 =                  INIT  EQU 4
0005 =                  INCR  EQU 5
                ;
0100 3E04       START:  LD A, INIT      ;Get initial value.
0102 0605               LD B, INCR      ;Set up loop..
0104 CD0901     RP1     CALL UPIT       ;Call increment routine.
0107 10FB               DJNZ RP1        ;Repeat..
                                        ;END OF PROG5.
                                        ;============
                ;
                ;UPIT-Subroutine to increment accumulator.
                ;
0109 3C         UPIT:   INC A
010A C9                 RET             ;Return.
                ;
                ;
                ;
                ;
                ;Note that at the end of the program
                ;the contents of the accumulator will
                ;be INIT+INCR=9.
                ;
0000            END

0005  INCR      0004  INIT      0104  RP1       0100  START     0109  UPIT


No errors
```

A.2.2 PROGRAMS VIEWED FROM THE SCREEN

Three programs are given here which start to demonstrate the use of the RML 'EMT' instructions for input and output, e.g. of ASCII characters.

```
                ;PROG6:
                ;=====
                ;This is a very short program which
                ;when run will echo everything typed
                ;on the keyboard onto the screen.
                ;
0100            ORG 0100H
                ;
                ;EMT values
                ;==========
                ; (RML input/output routines)
                ;
0022 =                  KBDWF   EQU 22H
0001 =                  OUTC    EQU 01H
                ;
                ;
0100 F722       START:  EMT KBDWF       ;Get character from
                                        ;keyboard..
0102 F701               EMT OUTC        ;Echo it..&..
0104 C30001             JP START        ;repeat.
                ;
                ;
                ;Note that this program will continue
                ;running until e.g. the RESET button
                ;is pressed.
                ;
                ;
0000            END

0022 KBDWF    0100  START    0001  OUTC

No errors
```

```
                ;PROG7:
                ;=====
                ;This program is similar to PROG6 except
                ;that a subroutine is called which converts
                ;the character entered into lower case,
                ;before it is echoed back.
0100            ORG  0100H
                ;EMT values
                ;==========
0022 =                  KBDWF   EQU  22H
0001 =                  OUTC    EQU  01H
                ;
                ;Values
                ;======
0020 =                CNVRT   EQU 20H
                ;
                ;
0000 F722       START:    EMT  KBDWF       ;Get character.
0002 CD0900               CALL CON1        ;Convert it.
0005 F701                 EMT  OUTC        ;Echo it..&..
0007 18F7                 JR START         ;repeat.
                ;
                ;CON1-Subroutine to convert upper case
                ;to lower case by adding 20H.
                ;
0009 C620       CON1:     ADD A, CNVRT   ;Add on 20H.
000B C9                   RET            ;Return.
                ;
                ;
0000            END


0020  CNVRT    0009 CON1    0022 KBDWF    0000 START    0001 OUTC

No errors
```

```
                ;PROG8:
                ;=====
                ;This is just a simple program which will
                ;continually output the ASCII character
                ;set from <space> to <z>.
0100            ORG  0100H
                ;EMT values
                ;==========
0001 =                  OUTC EQU 01H
                ;
                ;Values:
                ;======
0020 =                  DATUM EQU ' '
007A =                  LIMIT EQU 'Z'
000D =                  CRET  EQU ODH ;<CR/LF>
                ;
                ;
0000 3E20       START:  LD A,DATUM    ;Start value.
0002 F701       ST1:    EMT OUTC      ;Output it.
0004 FE7A               CP LIMIT      ;Finished?
0006 CAOCOO             JP Z,ST2      ;Yes->
0009 3C                 INC A         ;No-increment &
000A 18F6               JR ST1        ;repeat.

000C 3EOD       ST2:    LD A,CRET     ;Output a newline.
000E F701               EMT OUTC
0010 C30000             JP START      ;Start again!!
                ;
                ;
0000            END

000D CRET       0020 DATUM      007A LIMIT      0002 ST1      000C ST2
0000 START      0001 OUTC

No errors
```

BIBLIOGRAPHY

BIBLIOGRAPHY

1. MOSTEK, *Z80 Programming Manual*, Mostek Corporation, 1977.

2. C.D. Kraft and W.N. Toy, *Mini/Microcomputer Hardware Design*, (Chapter 6, App C), Prentice Hall, 1979.

3. D. Johnson, J. Hilburn, and P.Julich, *Digital Circuits and Microcomputers*, Prentice Hall, 1979.

4. L. Nashelsky, *Intro.to Digital Computer Technology*, Wiley, 1977.

5. R. Zaks, *Microprocessors - from chips to systems*, Sybex, 1977

6. A. Osborne, *An Intro. to Microcomputers - Vol.1 Basic Concepts*, Sybex, 1977.

7. E. Nichols, J. Nichols and P. Rony, *Z-80 Microprocessor - Programming and Interfacing Vol. 1,2*, Sams, 1979.

8. W. Barden Jr., *The Z80 microcomputer handbook*, Sams, 1978.

9. A. Lippiatt, *The Architecture of Small Computer Systems*, Prentice Hall, 1979.