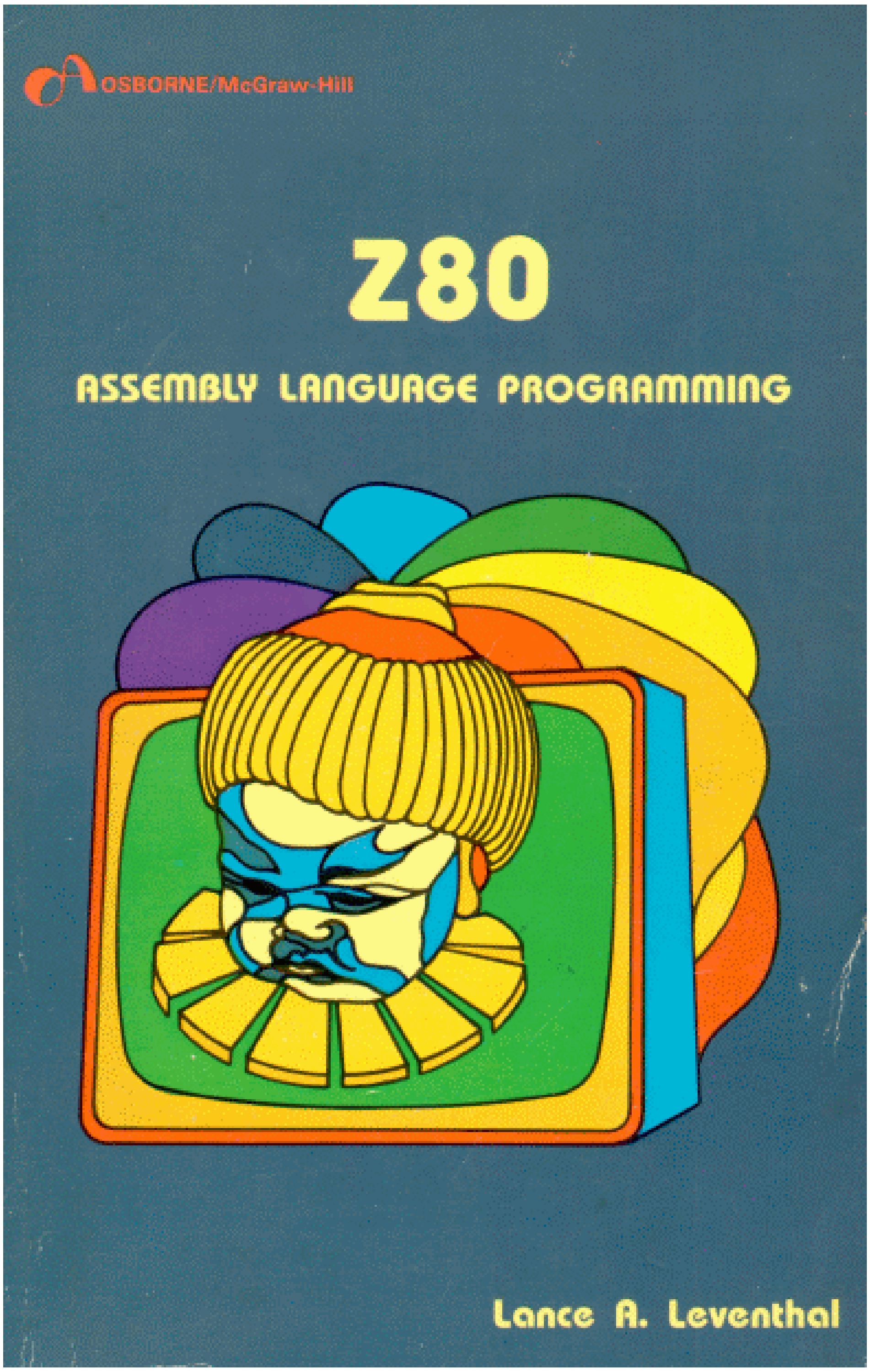


OSBORNE/McGraw-Hill

Z80

ASSEMBLY LANGUAGE PROGRAMMING

Lance A. Leventhal

Chapter 3 page 1-42

Contents (Continued)

In this PDF: Chapter 3 page 1-42

Chapter 3
THE Z80 ASSEMBLY LANGUAGE INSTRUCTION SET

We are now ready to start writing assembly language programs. We begin in this chapter by defining the individual instructions of the Z80 assembly language instruction set, plus the syntax rules of the Zilog assembler.

We do not discuss any aspects of microcomputer hardware, signals, interfaces, or CPU architecture in this book. This information is described in detail in An Introduction to Microcomputers: Volume 2 — Some Real Microprocessors and Volume 3 — Some Real Support Devices, while Z80 Programming for Logic Design discusses assembly language as an extension of digital logic. In this book, **we look at programming techniques from the assembly language programmer's viewpoint, where pins and signals are irrelevant and there are no important differences between a minicomputer and a microcomputer.**

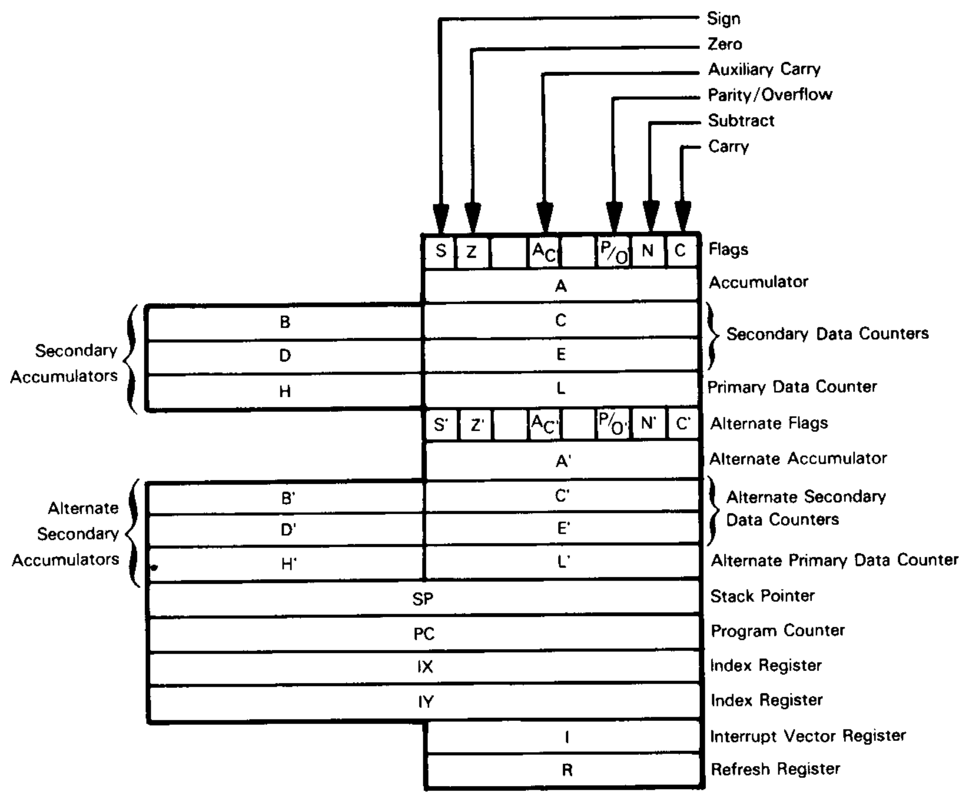
Interrupts, direct memory access, and the Stack architecture for the Z80 will be described in later chapters of this book, in conjunction with assembly language programming discussions of the same subjects.

This chapter contains a detailed definition of each assembly language instruction. These definitions are identical to those found in Chapter 6 of Z80 Programming for Logic Design.

The detailed description of individual instructions is preceded by a general discussion of the Z80 instruction set that divides instructions into those which are commonly used, infrequently used, and rarely used. If you are an experienced assembly language programmer, this categorization is not particularly important — and, depending on your own programming prejudices, it may not even be accurate. If you are a novice assembly language programmer, we recommend that you begin by writing programs using only instructions in the "commonly used" category. Once you have mastered the concepts of assembly language programming, you may examine other instructions and use them where appropriate.

CPU REGISTERS AND STATUS FLAGS

The CPU registers and status flags for the Z80 may be illustrated as follows:



The Accumulator is the primary source and destination for one-operand and two-operand instructions. For example, the shortest and fastest data transfers between the CPU and I/O devices are performed through the Accumulator. In addition, more Memory Reference instructions move data between the Accumulator and memory than between any other register and memory. All 8-bit arithmetic and Boolean instructions take one of the operands from the Accumulator and return the result to the Accumulator. An instruction must therefore **load the Accumulator before the Z80 can perform any 8-bit arithmetic or Boolean operations.**

The B, C, D, E, H, and L registers are all secondary registers. Data stored in any of these six registers may be accessed with equal ease; such data can be moved to any other register or can be used as the second operand in two-operand instructions.

There are, however, some important differences in the functions of Registers B, C, D, E, H, and L.

Registers H and L are the primary Data Pointer for the Z80. That is to say, you will normally use these two registers to hold the 16-bit memory address of data being accessed. Data may be transferred between any registers and the memory location addressed by H and L. Since HL is the primary Data Pointer, it often takes fewer bytes of object code and less instruction cycles to perform operations with it. The Z80 programmer should try to address data memory via Registers H and L whenever possible.

Within your program logic, always reserve Registers H and L to hold a data memory address.

Registers B, C, D, and E provide secondary data storage; frequently, the second operand for two-operand instructions is stored in one of these four registers. (The first operand is stored in the Accumulator, which is also the destination for the result.)

There are **a limited number of instructions** that **treat Registers B and C, or D and E, as 16-bit Data Pointers.** But these instructions move data between memory and the Accumulator only.

In your program logic you should normally use Registers B, C, D, and E as temporary storage for data or addresses.

Registers IX and IY are index registers. They provide a limited indexing capability of the type described in An Introduction to Microcomputers: Volume 1 for short instructions.

The alternate registers F', A', B', C', D', E', H', and L' provide a duplicate set of general purpose registers. Just two single-byte Exchange instructions select and deselect all alternate registers; one instruction exchanges AF and the alternate AF' as a register pair, and one instruction exchanges BC, DE, and HL with the alternate BC', DE', and HL'. Once selected, all subsequent register operations are performed on the active set until the next exchange selects the inactive set. **The alternate registers can be reserved for use when a fast interrupt response is required.** Or, they may be used in any desired way by the programmer.

There are a number of instructions that handle 16 bits of data at a time. These instructions refer to pairs of CPU registers as follows:

F	and	A
B	and	C
D	and	E
H	and	L
F'	and	A'
B'	and	C'
D'	and	E'
H'	and	L'
High-order byte		Low-order byte

The combination of the Accumulator and flags, treated as a 16-bit unit, is used only for Stack operations and alternate register switches. Arithmetic operations access B and C, D and E, or H and L as 16-bit data units.

The Carry status flag holds carries out of the most significant bit in any arithmetic operation. The Carry flag is also included in Shift instructions; it is reset by Boolean instructions.

The Subtract flag is designed for internal use during decimal adjust operations. This flag is set to 1 for all Subtract instructions and reset to 0 for all Add instructions.

The Parity/Overflow flag is a multiple use flag, depending on the operation being performed. For arithmetic operations, it is an overflow flag. For input, rotate, and Boolean operations, it is a parity flag, with 1 = even parity and 0 = odd parity. During block transfer and search operations, it remains set until the byte counter decrements to zero; then it is reset to zero. It is also set to the current state of the interrupt enable flip-flop (IFF2) when a LD A,I or LD A,R instruction is executed.

The Zero flag is set to 1 when any arithmetic or Boolean operation generates a zero result. The Zero status is set to 0 when such an operation generates a nonzero result.

The Sign status flag acquires the value of the most significant bit of the result following the execution of any arithmetic or Boolean instruction.

The Auxiliary Carry status flag holds any carry from bit 3 to 4 resulting from the execution of an arithmetic instruction. The purpose of this status flag is to simplify Binary-Coded-Decimal (BCD) operations; this is the standard use of an Auxiliary Carry status flag as described in An Introduction to Microcomputers: Volume 1, Chapter 3.

All of the above status flags keep their current value until an instruction that modifies them is executed. Merely changing the value of the Accumulator will not necessarily change the value of the status flags. For example, if the Zero flag is set, and a load immediate to the Accumulator is executed, that causes the Accumulator to acquire a non-zero value; the value of the Zero flag remains unchanged.

The 16-bit Stack Pointer allows you to implement a Stack anywhere in addressable memory. The size of the Stack is limited only by the amount of addressable memory present. In reality you will rarely use more than 256 bytes of memory for your Stack. You should use the Stack for accessing subroutines and processing interrupts. Do not use the Stack to pass parameters to subroutines. This is not very efficient within the limitations of the Z80 instruction set. The Z80 Stack is started at its highest address. A Push decrements the Stack Pointer contents; a Pop increments the Stack Pointer contents.

The Interrupt Vector register and the Refresh register are special-purpose registers not normally used by the programmer.

The Interrupt Vector register is used to store the page address of an interrupt response routine; the location on the page is provided by the interrupting device. This scheme allows the address of the interrupt response routine to be changed while still providing a very fast response time for the interrupting device.

The Refresh register contains a memory refresh counter in the low-order seven bits. This counter is incremented automatically after each instruction fetch and provides the next refresh address for dynamic memories. The high-order bit of the Refresh register will remain set or reset, depending on how it was loaded at the last LD R,A instruction.

Z80 MEMORY ADDRESSING MODES

The Z80 provides extensive addressing modes. These include:

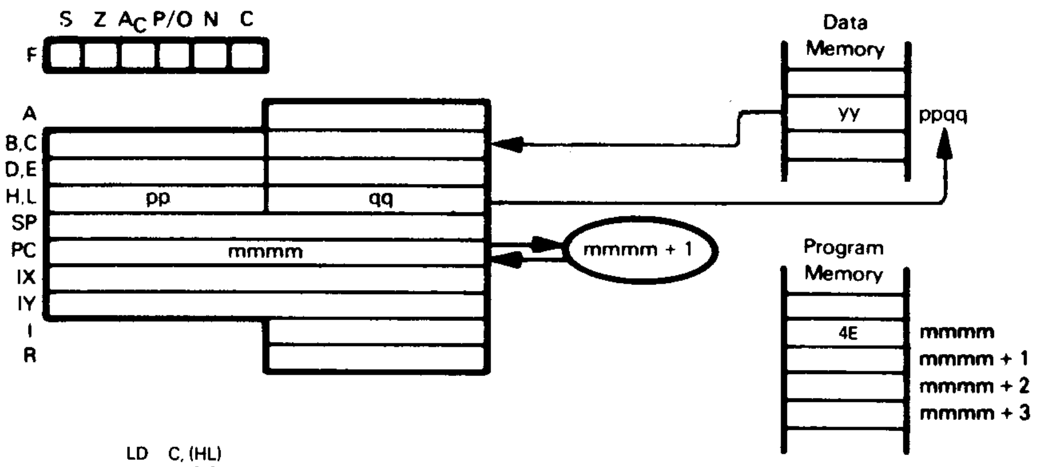
- **Implied**
- **Implied Block Transfer with Auto-Increment/Decrement**
- **Implied Stack**
- **Indexed**
- **Direct**
- **Program Relative**
- **Base Page**
- **Register Indirect**
- **Immediate**

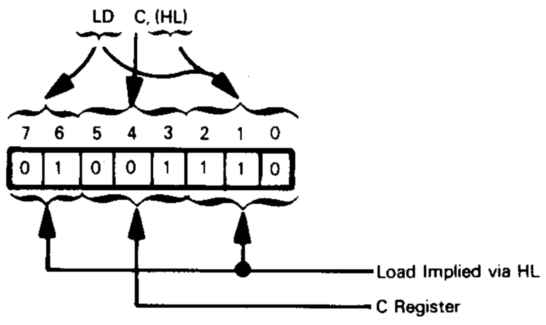
Implied

In implied memory addressing, the H and L registers hold the address of the memory location being accessed. Data may be moved between the identified memory location and any one of the seven CPU registers A, B, C, D, E, H, or L. For example, the instruction

LD C,(HL)

loads the C register with the contents of the memory location currently pointed to by HL. This is illustrated as follows:

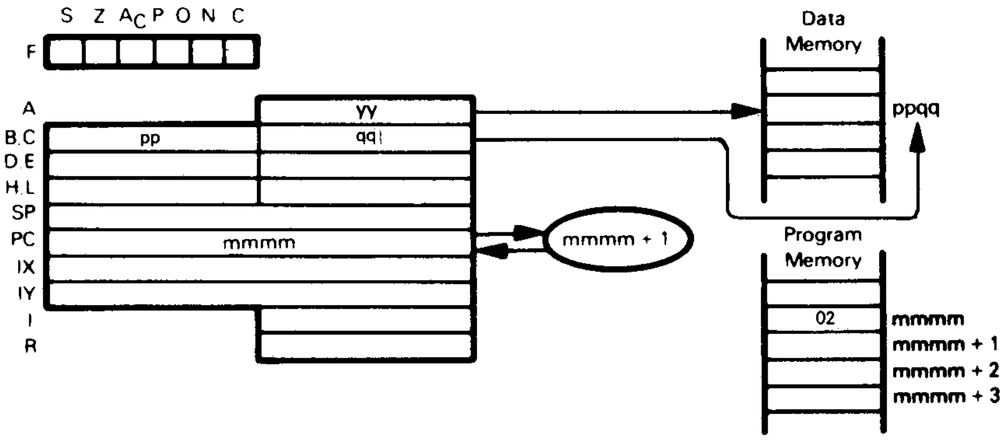


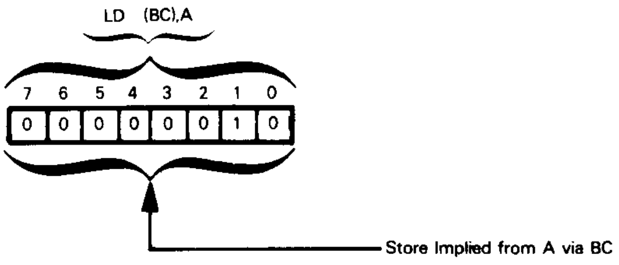


A limited number of instructions use Registers B and C or D and E as the Data Pointer. These instructions move data between the Accumulator and the memory location addressed by Registers B and C or Registers D and E. The instruction

LD (BC),A

stores the contents of A into the memory location currently addressed by Register Pair BC. This is illustrated as follows:





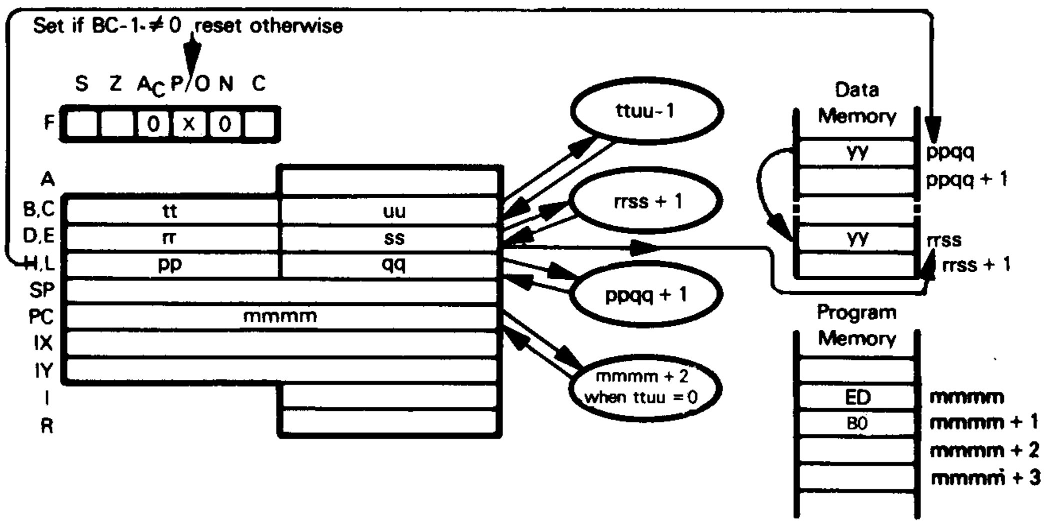
Implied Block Transfer With Auto-Increment/Decrement

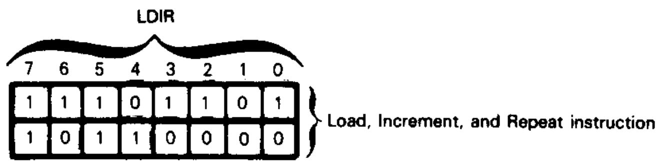
Block Transfer and Search instructions operate on a block of data whose size is set by the programmer as the contents of the BC register pair. In this form of addressing, a byte of data is moved from the memory location addressed by HL to the memory location addressed by DE; then HL and DE are incremented and BC is decremented. Data transfer continues until BC reaches zero, at which point the instruction is terminated. Variations include allowing other instructions to follow each data transfer, with the programmer supplying the loopback; auto-decrementing HL and DE instead of auto-incrementing; and a complementary set of Block Search instructions that compare the memory byte addressed by HL with the contents of the A register, setting a flag if a match is found.

The Load, Increment, and Repeat instruction

LDIR

is illustrated as follows:





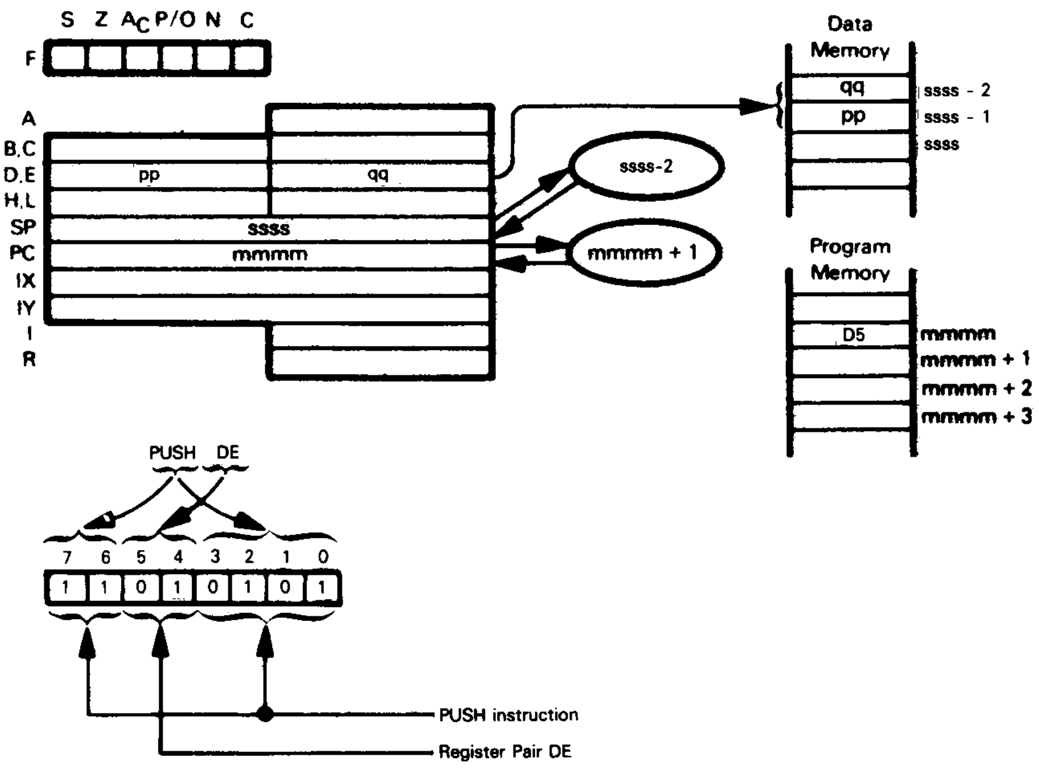
A similar group of Input/Output instructions is provided, allowing a block of data to be input or output between memory and an I/O device. The I/O port number is taken as the contents of the C register, with the single B register used as the byte counter. Memory is addressed by HL.

Implied Stack

Since the Stack is part of Read/Write memory, we must consider Stack instructions as Memory Reference instructions. **Push and Pop instructions move two bytes of data between a register pair and the addressed Stack Pointer location,** i.e., current top-of-stack. The Z80 Stack address is decremented with each Push and incremented with each Pop. The instruction

PUSH DE

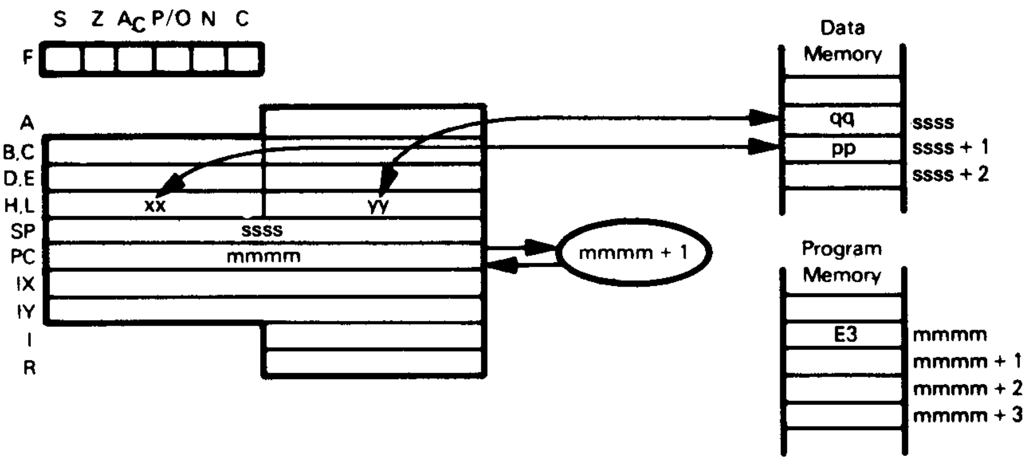
is illustrated as follows:



The Z80 also has instructions that exchange the two top-of-stack bytes with a 16-bit register — HL or one of the two index registers. The instruction

EX (SP),HL

is illustrated as follows:

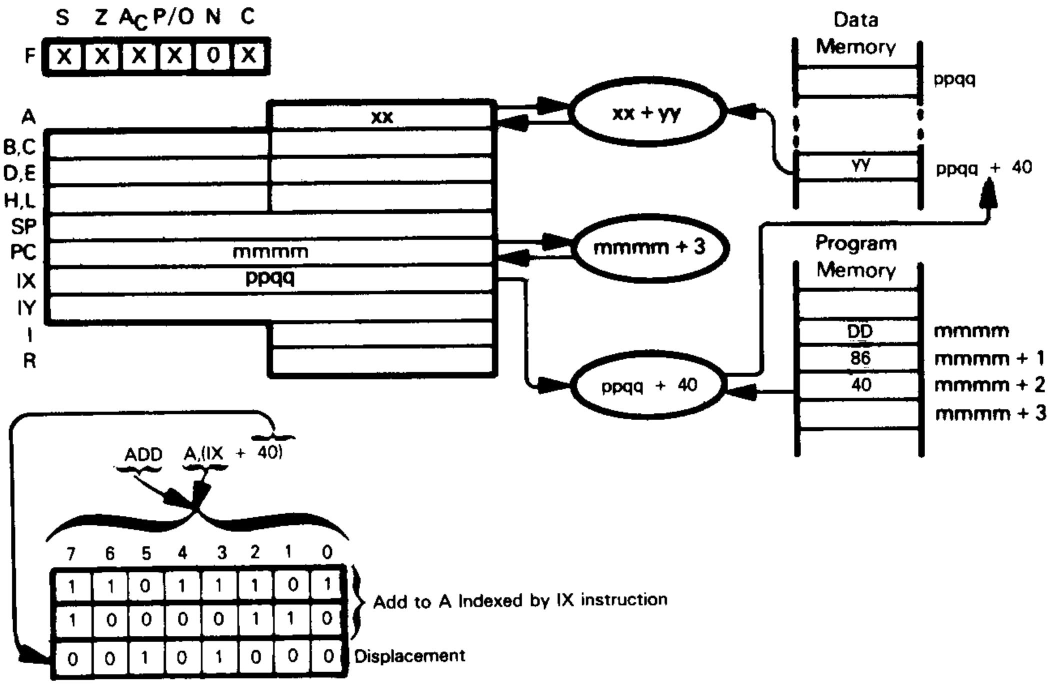


Indexed

The Z80 has two 16-bit index registers, called IX and IY. They may be used interchangeably. All memory reference operations for which (HL) can be specified can alternatively be specified as an indexed operation. The difference between implied addressing using HL and indexed addressing using IX and IY is that **the index operand includes a displacement value that is added to the index address.** In the instruction

ADD A,(IX+40H)

the memory address is the sum of the contents of the IX register and 40_{16}. This may be illustrated as follows:



Direct

Direct addressing can be used to load the Accumulator with any 8-bit value from memory, load BC, DE, HL, SP, IX, or IY with any 16-bit memory value, and jump or call subroutines direct at any memory location. The 16-bit direct address is stored in the last two bytes of the instruction, in low-byte high-byte order (this is the reverse of the standard high-low scheme).

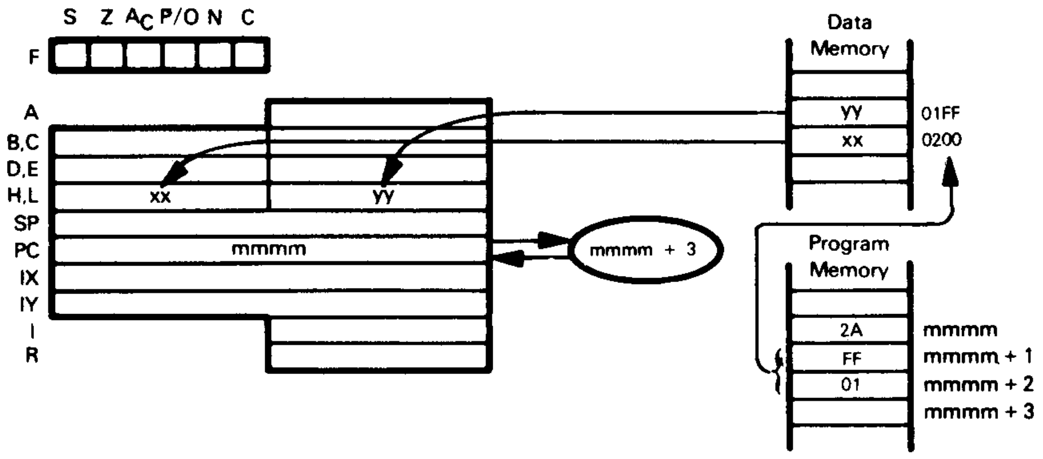
The instruction

LD A,(NETX)

loads the A register with the contents of the memory location addressed by the label NETX. The instruction

LD HL,(1FFH)

loads the L register with the contents of memory location $01FF_{16}$ and the H register with the contents of memory location 0200_{16}. This may be illustrated as follows:



LD HL,(1FFH)

7	6	5	4	3	2	1	0	
0	0	1	0	1	0	1	0	Load HL Direct instruction
1	1	1	1	1	1	1	1	Direct address - low byte
0	0	0	0	0	0	0	1	Direct address - High byte

The direct Jump instructions provide jumps and jumps-to-subroutines, both unconditional and conditional. These are all 3-byte instructions, with the direct address stored in the second and third bytes of the instruction, as shown above for Load Direct.

There are three additional addressing modes used by Z80 Branch instructions: program relative, base page, and register indirect. In general, they are shorter and/or faster than direct jumps but may have more limited addressing capabilities.

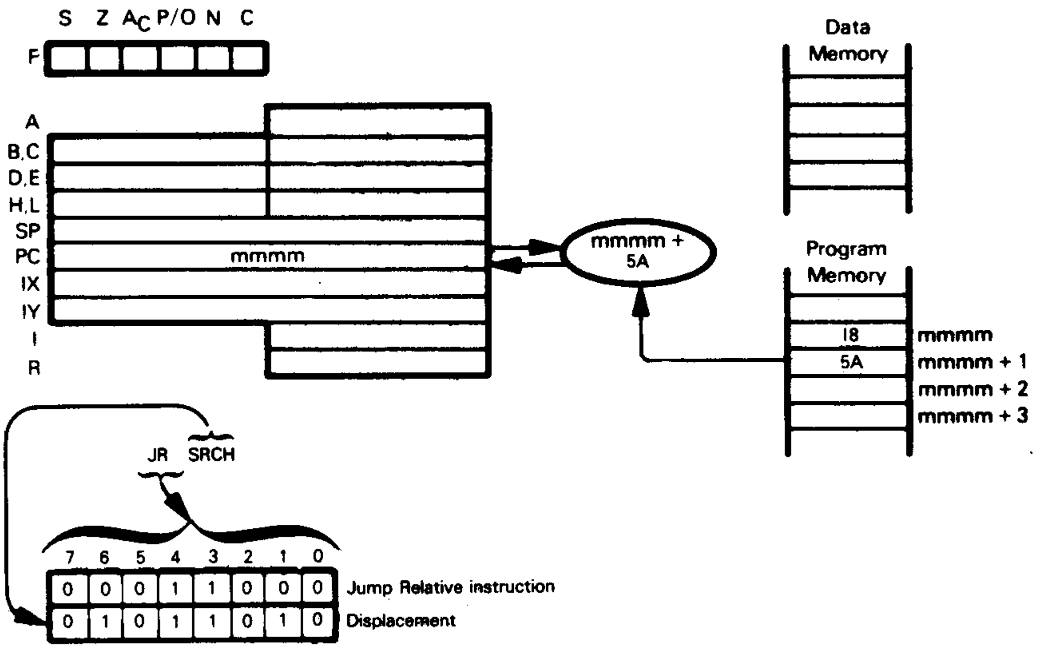
Program Relative

Jump Relative instructions provide program relative addressing in the range -126, +129 bytes from the first byte of the Program Relative instruction. These instructions are all 2-byte instructions, with the signed displacement value stored in the second byte of the instruction. **There are unconditional and conditional relative jumps, as well as a Decrement and Jump If Not Zero instruction (DJNZ) that facilitates loop control.**

Given the instruction

JR SRCH

assume that SRCH is a label addressing a location $5A_{16}$ bytes up in memory from the JR op-code byte. The operation may be illustrated as follows:



Base Page

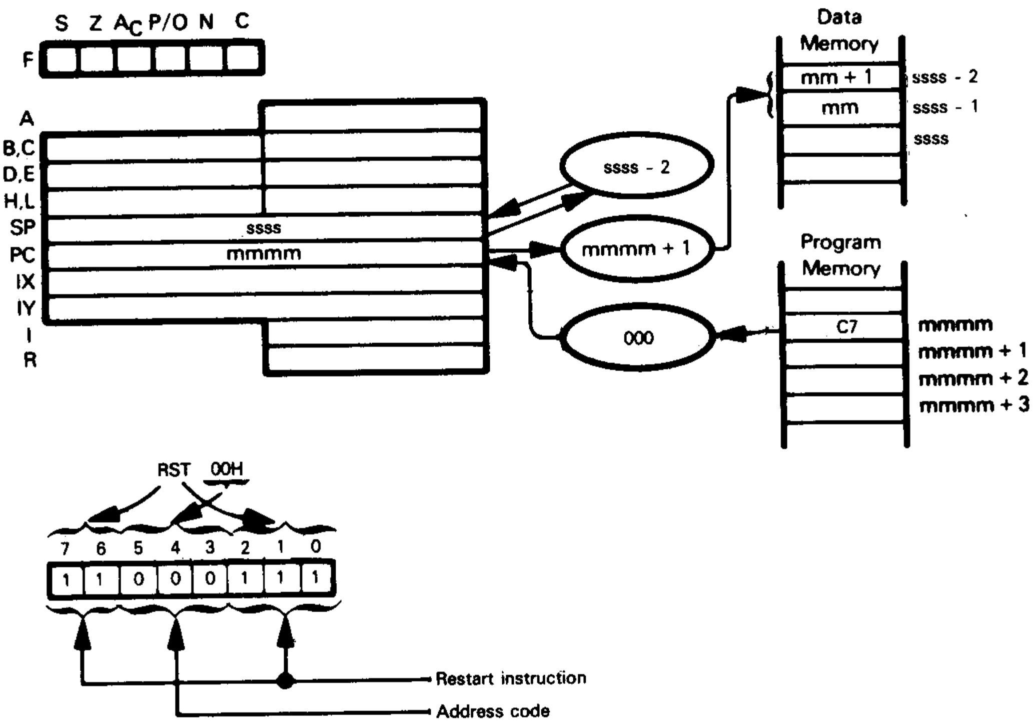
The Z80 has a **modified base page addressing** mode for the Restart instruction. This is a special Call instruction that **allows a single-byte instruction to jump to one of eight subroutines located at specific points in lower core.** The effective address is calculated from a 3-bit code stored in the instruction, as follows:

Lower Core Address	3-Bit Code
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

The decoded address value is loaded into the low-order byte of the Program Counter; the high-order byte of the Program Counter is set to zero. For example, the instruction

RST 00H

is illustrated as follows:



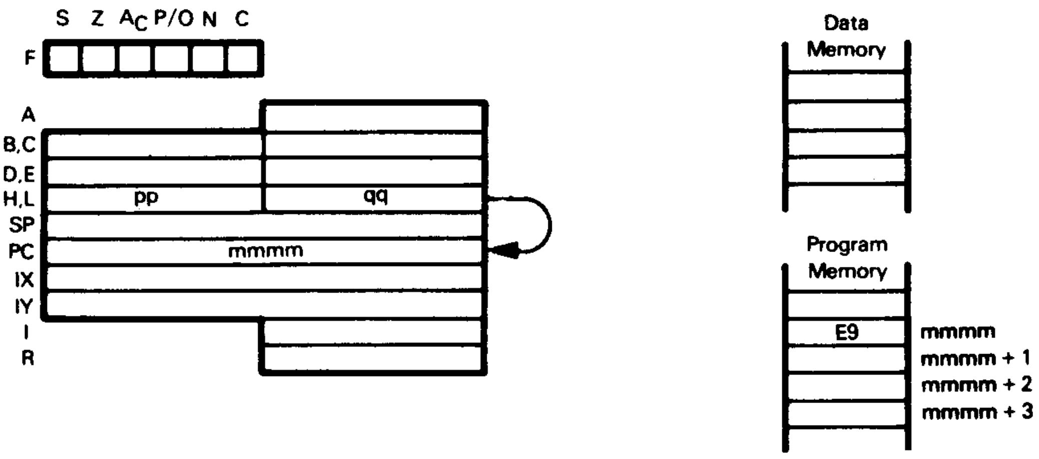
Register Indirect

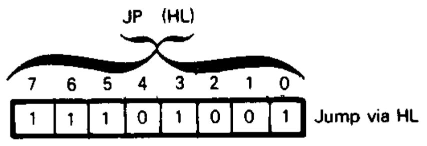
In standard indirect addressing, a memory location contains the effective address, and the instruction specifies the address of the memory location containing the effective address. In register indirect addressing, a register contains the effective address, and the instruction specifies which of the registers contains the effective address. Note that for a Load, for instance, this is just another way of describing implied addressing. However, **the Z80 has Jump instructions that allow a jump to the memory location whose address is contained in the specified register.** This is a form of indirect addressing, and is described separately because, while most microcomputers have implied addressing, very few have register indirect jumps.

The instruction

```
JP    (HL)
```

directs that a jump is to be taken to the memory location whose address is contained in HL. This may be illustrated as follows:





Immediate

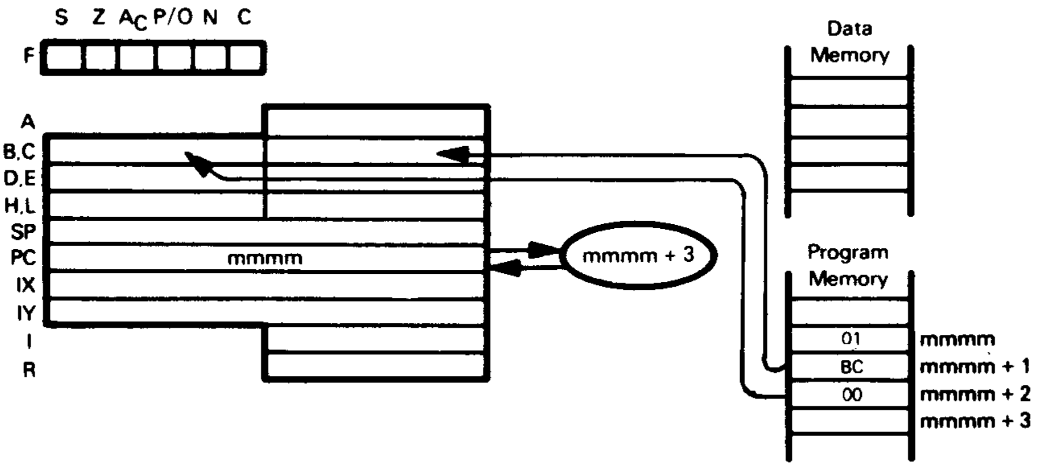
Some texts identify Immediate instructions as Memory Reference instructions. An Immediate instruction is a 2-, 3-, or 4-byte instruction in which the last one or two bytes hold fixed data that is loaded into a register or memory location. **The Z80 provides Immediate instructions to:**

- **load 8-bit data into any of the 8-bit registers,**
- **load 16-bit data into any of the register pairs or 16-bit registers,**
- **store 8-bit data into any memory location using implied or indexed addressing,**
- **perform arithmetic and logical operations using the Accumulator and 8-bit immediate data.**

The instruction

LD BC,0BCH

loads the immediate data value BC_{16} into Register Pair BC. This may be illustrated as follows:



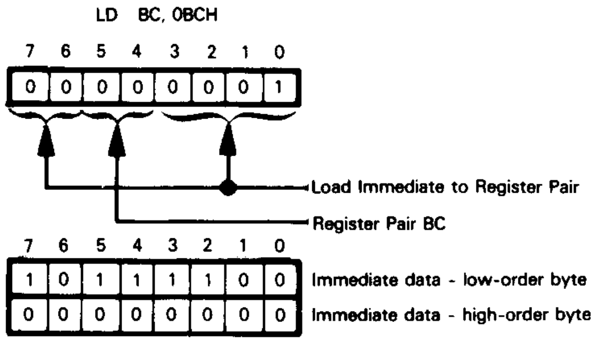


Table 3-1. Frequently Used Instructions of the Z80

Instruction Code	Meaning
ADC A	Add with Carry to Accumulator
ADD	Add
AND	Logical AND
CALL addr	Call Subroutine
CALL cond.addr	Call Conditional
CP	Compare
DEC	Decrement
DJNZ	Decrement and Jump If Not Zero
IN	Input
INC	Increment
JR	Jump Relative
JR cond.addr	Jump Relative Conditional
LD reg.(HL)	Load Register
LD A.(addr)	Load Accumulator Direct
LD data	Load Immediate
LD (HL).reg	Store Register
LD (addr).A	Store Accumulator Direct
LD dst.src	Move Register-to-Register
OUT	Output
POP	Pop from Stack
PUSH	Push to Stack
RET	Return from Subroutine
RET cond	Return Conditional
RLA	Rotate Accumulator Left Through Carry
RRA	Rotate Accumulator Right Through Carry
SLA	Shift Left Arithmetic
SRL	Shift Right Logical
SUB	Subtract

Table 3-2. Occasionally Used Instructions of the Z80

Instruction Code		Meaning
BIT		Test Bit
CPD, CPDR		Compare, Decrement, (Repeat)
CPI, CPIR		Compare, Increment, (Repeat)
CPL		Complement Accumulator
DAA		Decimal Adjust Accumulator
DI		Disable Interrupts
EI		Enable Interrupts
EX		Exchange
HALT		Halt
IND, INDR		Input, Decrement, (Repeat)
INI, INIR		Input, Increment, (Repeat)
JP	addr	Jump
JP	cond,addr	Jump Conditional
LD	A,(BC) or (DE)	Load Accumulator Secondary
LD	HL,(addr)	Load HL Direct
LD	reg,(xy+disp)	Load Register Indexed
LD	rp,(addr)	Load Register Pair Direct
LD	xy,(addr)	Load Index Register Direct
LD	(BC) or (DE),A	Store Accumulator Secondary
LD	(addr),HL	Store HL Direct
LD	(xy+disp),reg	Store Register Indexed
LD	(addr),rp	Store Register Pair Direct
LD	(addr),xy	Store Index Register Direct
LD	(HL),data	Store Immediate to Memory
LD	(xy+disp),data	Store Immediate to Memory Indexed
LDD, LDDR		Load, Decrement, (Repeat)
LDI, LDIR		Load, Increment, (Repeat)
NEG		Negate (Twos Complement) Accumulator
NOP		No Operation
OR		Logical OR
OUTD, OTDR		Output, Decrement, (Repeat)
OUTI, OTIR		Output, Increment, (Repeat)
RES		Reset Bit
RETI		Return from Interrupt
RL		Rotate Left Through Carry
RLC		Rotate Left Circular
RLCA		Rotate Accumulator Left Circular
RR		Rotate Right Through Carry
RRC		Rotate Right Circular
RRCA		Rotate Accumulator Right Circular
SET		Set Bit
SRA		Shift Right Arithmetic
XOR		Logical Exclusive OR

Table 3-3. Seldom Used Instructions of the Z80

Instruction Code	Meaning
ADC HL,rp	Add Register Pair with Carry to HL
CCF	Complement Carry Flag
EXX	Exchange Register Pairs and Alternatives
IM n	Set Interrupt Mode
RETN	Return from Non-Maskable Interrupt
RLD	Rotate Accumulator and Memory Left Decimal
RRD	Rotate Accumulator and Memory Right Decimal
RST	Restart
SBC	Subtract with Carry (Borrow)
SCF	Set Carry Flag
LD A,I	Load Accumulator from Interrupt Vector Register
LD A,R	Load Accumulator from Refresh Register
LD I,A	Store Accumulator to Interrupt Vector Register
LD R,A	Store Accumulator to Refresh Register
LD SP,HL	Move HL to Stack Pointer
LD SP,xy	Move Index Register to Stack Pointer

ABBREVIATIONS

These are the abbreviations used in this chapter:

A,F,B,C,D,E,H,L	The 8-bit registers. A is the Accumulator and F is the Flag Word.
AF',BC',DE',HL'	The alternate register pairs
addr	A 16-bit memory address
x(b)	Bit b of 8-bit register or memory location x
cond	Condition for program branching. Conditions are: NZ - Non-Zero (Z = 0) Z - Zero (Z = 1) NC - Non-carry (C = 0) C - Carry (C = 1) PO - Parity Odd (P = 0) PE - Parity Even (P = 1) P - Positive Sign (S = 0) M - Negative Sign (S = 1)
data	An 8-bit binary data unit
data16	A 16-bit binary data unit
disp	An 8-bit signed binary address displacement
xx(HI)	The high-order 8 bits of a 16-bit quantity xx
I	Interrupt Vector register (8 bits)
IX IY	The Index registers (16 bits each)
label	A 16-bit instruction memory address
xx(LO)	The low-order 8 bits of a 16-bit quantity xx
LSB	Least Significant Bit (Bit 0)
MSB	Most Significant Bit (Bit 7)
PC	Program Counter
port	An 8-bit I/O port address

pr	Any of the following register pairs: BC DE HL AF
R	The Refresh register (8 bits)
reg	Any of the following registers: A B C D E H L
rp	Any of the following register pairs: BC DE HL SP
SP	Stack Pointer (16 bits)
xy	Either one of the Index registers (IX or IY)

Object Code	bbb	Bit number 000 (LSB) to 111 (MSB)	
	ccc	Condition code	000 = non-zero 001 = zero 010 = no carry 011 = carry 100 = parity odd 101 = parity even 110 = positive sign 111 = negative sign
	ddd	Destination register — same coding as rrr	
	ppqq	A 16-bit memory address	
	rrr	Register	111 = A 000 = B 001 = C 010 = D 011 = E 100 = H 101 = L
	sss	Source register — same coding as rrr	
	x	Index register	0 = IX 1 = IY
	xx	Register pair	00 = BC 01 = DE 10 = HL 11 = SP (rp) or AF (pr)
	xxx	Restart code (000 to 111)	
	yy	An 8-bit binary data unit	
	yyyy	A 16-bit binary data unit	

Statuses	The Z80 has the following status flags: C - Carry status Z - Zero status S - Sign status P/O - Parity/Overflow status A_C - Auxiliary Carry status N - Subtract status The following symbols are used in the status columns: X - flag is affected by operation (blank) - flag is not affected by operation 1 - flag is set by operation 0 - flag is reset by operation U - flag is unknown after operation P - flag shows parity status O - flag shows overflow status I - flag shows interrupt enabled/disabled status
[[ ]]	Memory addressing: 1) the contents of the memory location whose address is contained in the designated register, 2) an I/O port whose address is contained in the designated register.
[ ]	The contents of a register or memory location. For example: $[[HL]] \leftarrow [[HL]] + 1$ indicates that the contents of the memory location addressed by the contents of HL are incremented, whereas: $[HL] \leftarrow [HL] + 1$ indicates that the contents of the HL register itself are incremented.
$\wedge$	Logical AND
V	Logical OR
$\veebar$	Logical Exclusive-OR
$\leftarrow$	Data is transferred in the direction of the arrow
$\leftarrow \rightarrow$	Data is exchanged between the two locations designated on either side of the arrows.

INSTRUCTION MNEMONICS

Table 3-4 summarizes the Z80 instruction set. The MNEMONIC column shows the instruction mnemonic (IN, OUT, LD). The OPERAND column shows the operands, if any, used with the instruction mnemonic.

The fixed part of an assembly language instruction is shown in UPPER CASE. The variable part (immediate data, I/O device number, register name, label or address) is shown in lower case.

For closely related operands, each type is listed separately without repeating the mnemonic. For instance, examples of the format entry

```
      LD  rp,(addr)
          xy,(addr)
are:  LD  BC,(DAT2)
      LD  IX,(MEM)
```

INSTRUCTION OBJECT CODES

The object code and instruction length in bytes are shown in Table 3-4 for each instruction variation. Table 3-5 lists the object codes in numerical order.

For instruction bytes without variations, object codes are represented as two hexadecimal digits (e.g., 3F).

For instruction bytes with variations in one of the two digits, the object code is shown as one 4-bit binary digit and one hexadecimal digit (e.g., 11 x 1 D) in Table 3-5. For other instruction bytes with variations, the object code is shown as eight binary digits (e.g., 01sss001).

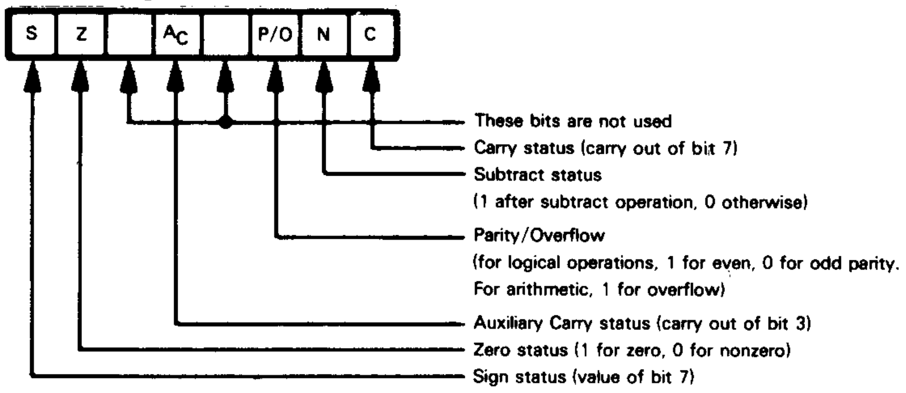
INSTRUCTION EXECUTION TIMES

Table 3-4 lists the instruction execution times in clock periods. Real time can be obtained by dividing the given number of clock periods by the clock frequency. For example, for an instruction that requires 7 clock periods, a 4 MHz clock will result in a 1.75 microsecond execution time.

When two possible execution times are shown (i.e., 5/11), it indicates that the number of clock periods depends on condition flags. The first time is for "condition not met," whereas the second is for "condition met."

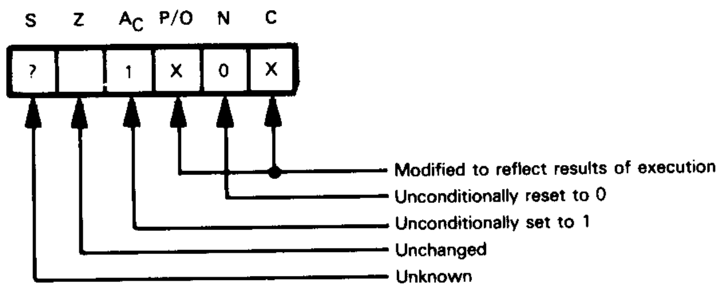
STATUS

The six status flags are stored in the Flag register (F) as follows:



S
Z
A_C
P/O
N
C
These bits are not used
Carry status (carry out of bit 7)
Subtract status
(1 after subtract operation, 0 otherwise)
Parity/Overflow
(for logical operations, 1 for even, 0 for odd parity.
For arithmetic, 1 for overflow)
Auxiliary Carry status (carry out of bit 3)
Zero status (1 for zero, 0 for nonzero)
Sign status (value of bit 7)

In the individual instruction descriptions, the effect of instruction execution on status is illustrated as follows:


S
Z
A_C
P/O
N
C
?
1
X
0
X
Modified to reflect results of execution
Unconditionally reset to 0
Unconditionally set to 1
Unchanged
Unknown

STATUS CHANGES WITH INSTRUCTION EXECUTION

An X identifies a status that is set or reset. A 0 identifies a status that is always cleared. A 1 identifies a status that is always set. A blank means the status does not change. A question mark (?) means the status is not known.

** Address Bus: A0-A7: [C]
A8-A15: [B]

Table 3-4. A Summary of the Z80 Instruction Set

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed:
						C	Z	S	P/O	A_C	N	
I/O	IN	A,(port)	DB yy	2	10							[A] ← [port] Input to Accumulator from directly addressed I/O port. Address Bus: A0-A7: port A8-A15: [A]
	IN	reg,(C)	ED 01ddd000	2	11		X	X	P	X	0	[reg] ← [[C]] Input to register from I/O port addressed by the contents of C.** If second byte is 70_{16} only the flags will be affected.
	INIR		ED B2	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from low addresses to high. Contents of B serve as a count of bytes remaining to be transferred.**
	INDR		ED BA	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a block of data from I/O port addressed by contents of C to memory location addressed by contents of HL, going from high addresses to low. Contents of B serve as a count of bytes remaining to be transferred.**
	INI		ED A2	2	15		X	?	?	?	1	[[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement byte count and increment destination address.**

**Address Bus: A0-A7: [C]
A8-A15: [B]

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
I/O (Continued)	IND		ED AA	2	15		X	?	?	?	1	[[HL]] ← [[C]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a byte of data from I/O port addressed by contents of C to memory location addressed by contents of HL. Decrement both byte count and destination address.**
	OUT	(port),A	D3 yy	2	11							[port] ← [A] Output from Accumulator to directly addressed I/O port. Address Bus: A0-A7: port A8-A15: [A]
	OUT	(C),reg	ED 01sss001	2	12							[[C]] ← [reg] Output from register to I/O port addressed by the contents of C.**
	OTIR		ED B3	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from low memory to high. Contents of B serve as a count of bytes remaining to be transferred.**
	OTDR		ED BB	2	20/15**		1	?	?	?	1	Repeat until [B] = 0: [[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a block of data from memory location addressed by contents of HL to I/O port addressed by contents of C, going from high memory to low. Contents of B serve as a count of bytes remaining to be transferred.**

**Address Bus: A0-A7: [C]
A8-A15: [B]

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
I/O (Continued)	OUTI		ED A3	2	15		X	?	?	?	1	[[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] + 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement byte count and increment source address.**
	OUTD		ED AB	2	15		X	?	?	?	1	[[C]] ← [[HL]] [B] ← [B] - 1 [HL] ← [HL] - 1 Transfer a byte of data from memory location addressed by contents of HL to I/O port addressed by contents of C. Decrement both byte count and source address.**
Primary Memory Reference	LD	A,(addr)	3A ppqq	3	13							[A] ← [addr] Load Accumulator from directly addressed memory location.
	LD	HL,(addr)	2A ppqq	3	16							[H] ← [addr + 1], [L] ← [addr] Load HL from directly addressed memory.
	LD	rp,(addr) xy,(addr)	ED 01xx1011 ppqq 11x11101 2A ppqq	4 4	20 20							[rp(HI)] ← [addr + 1], [rp(LO)] ← [addr] or [xy(HI)] ← [addr + 1], [xy(LO)] ← [addr] Load register pair or Index register from directly addressed memory.
	LD	(addr),A	32 ppqq	3	13							[addr] ← [A] Store Accumulator contents in directly addressed memory location.
	LD	(addr),HL	22 ppqq	3	16							[addr + 1] ← [H], [addr] ← [L] Store contents of HL to directly addressed memory location.
	LD	(addr),rp (addr),xy	ED 01xx0011 ppqq 11x11101 22 ppqq	4 4	20 20							[addr + 1] ← [rp(HI)], [addr] ← [rp(LO)] or [addr + 1] ← [xy(HI)], [addr] ← [xy(LO)] Store contents of register pair or Index register to directly addressed memory.
	LD	A,(BC) A,(DE)	0A 1A	1 1	7 7							[A] ← [[BC]] or [A] ← [[DE]] Load Accumulator from memory location addressed by the contents of the specified register pair.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Primary Memory Reference (Continued)	LD	reg,(HL)	01ddd110	1	7							[reg] ← [[HL]] Load register from memory location addressed by contents of HL.
	LD	(BC),A (DE),A	02 12	1 1	7 7							[[BC]] ← [A] or [[DE]] ← [A] Store Accumulator to memory location addressed by the contents of the specified register pair.
	LD	(HL),reg	01110sss	1	7							[[HL]] ← [reg] Store register contents to memory location addressed by the contents of HL.
	LD	reg,(xy+disp)	11x11101 01ddd110 disp	3	19							[reg] ← [[xy] + disp] Load register from memory location using base relative addressing.
	LD	(xy+disp),reg	11x11101 01110sss disp	3	19							[[xy] + disp] ← [reg] Store register to memory location addressed relative to contents of Index register.
Block Transfer and Search	LDIR		ED B0	2	20/16**				0	0	0	Repeat until [BC] = 0: [[DE]] ← [[HL]] [DE] ← [DE] + 1 [HL] ← [HL] + 1 [BC] ← [BC] - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from low addresses to high. Contents of BC serve as a count of bytes to be transferred.
	LDDR		ED B8	2	20/16**				0	0	0	Repeat until [BC] = 0: [[DE]] ← [[HL]] [DE] ← [DE] - 1 [HL] ← [HL] - 1 [BC] ← [BC] - 1 Transfer a block of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE, going from high addresses to low. Contents of BC serve as a count of bytes to be transferred.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Block Transfer and Search (Continued)	LDI		ED A0	2	16				X	0	0	[[DE]] ← [[HL]] [DE] ← [DE] + 1 [HL] ← [HL] + 1 [BC] ← [BC] - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Increment source and destination addresses and decrement byte count.
	LDD		ED A8	2	16				X	0	0	[[DE]] ← [[HL]] [DE] ← [DE] - 1 [HL] ← [HL] - 1 [BC] ← [BC] - 1 Transfer one byte of data from the memory location addressed by the contents of HL to the memory location addressed by the contents of DE. Decrement source and destination addresses and byte count.
	CPIR		ED B1	2	20/16**		X	X	X	X	1	Repeat until [A] = [[HL]] or [BC] = 0: [A] - [[HL]] (only flags are affected) [HL] ← [HL] + 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from low addresses to high. Stop when a match is found or when the byte count becomes zero.
	CPDR		ED B9	2	20/16**		X	X	X	X	1	Repeat until [A] = [[HL]] or [BC] = 0: [A] - [[HL]] (only flags are affected) [HL] ← [HL] - 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory block addressed by contents of HL, going from high addresses to low. Stop when a match is found or when the byte count becomes zero.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Block Transfer and Search (Continued)	CPI		ED A1	2	16		X	X	X	X	1	[A] - [[HL]] (only flags are affected) [HL] ← [HL] + 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count.
	CPD		ED A9	2	16		X	X	X	X	1	[A] - [[HL]] (only flags are affected) [HL] ← [HL] - 1 [BC] ← [BC] - 1 Compare contents of Accumulator with those of memory location addressed by contents of HL. Decrement address and byte count.
Secondary Memory Reference	ADD	A,(HL) A,(xy +disp)	86 11x11101 86 disp	1 3	7 19	X	X	X	O	X	0	[A] ← [A] + [[HL]] or [A] ← [A] + [[xy] + disp] Add to Accumulator using implied addressing or base relative addressing.
	ADC	A,(HL) A,(xy +disp)	8E 11x11101 8E disp	1 3	7 19	X	X	X	O	X	0	[A] ← [A] + [[HL]] + C or [A] ← [A] + [[xy] + disp] + C Add with Carry using implied addressing or base relative addressing.
	SUB	(HL) (xy + disp)	96 11x11101 96 disp	1 3	7 19	X	X	X	O	X	1	[A] ← [A] - [[HL]] or [A] ← [A] - [[xy] + disp] Subtract from Accumulator using implied addressing or base relative addressing.
	SBC	A,(HL) A,(xy+disp)	9E 11x11101 9E disp	1 3	7 19	X	X	X	O	X	1	[A] ← [A] - [[HL]] - C or [A] ← [A] - [[xy] + disp] - C Subtract with Carry using implied addressing or base relative addressing.
	AND	(HL) (xy + disp)	A6 11x11101 A6 disp	1 3	7 19	O	X	X	P	1	0	[A] ← [A] Λ [[HL]] or [A] ← [A] Λ [[xy] + disp] AND with Accumulator using implied addressing or base relative addressing.
	OR	(HL) (xy + disp)	B6 11x11101 B6 disp	1 3	7 19	O	X	X	P	1	0	[A] ← [A] V [[HL]] or [A] ← [A] V [[xy] + disp] OR with Accumulator using implied addressing or base relative addressing.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

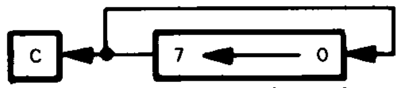
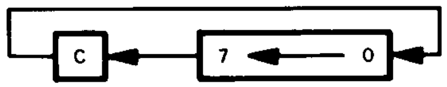
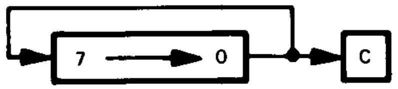
Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Secondary Memory Reference (Continued)	XOR	(HL)	AE	1	7	0	X	X	P	1	0	[A] ← [A] $\veebar$ [[HL]] or [A] ← [A] $\veebar$ [[xy] + disp] Exclusive-OR with Accumulator using implied addressing or base relative addressing.
		(xy + disp)	11x11101 AE disp	3	19							
	CP	(HL)	BE	1	7	X	X	X	O	X	1	[A] - [[HL]] or [A] - [[xy] + disp] Compare with Accumulator using implied addressing or base relative addressing. Only the flags are affected.
		(xy + disp)	11x11101 BE disp	3	19							
	INC	(HL)	34	1	11		X	X	O	X	0	[[HL]] ← [[HL]] + 1 or [[xy] + disp] ← [[xy] + disp] + 1 Increment using implied addressing or base relative addressing.
		(xy + disp)	11x11101 34 disp	3	23							
	DEC	(HL)	35	1	11		X	X	O	X	1	[[HL]] ← [[HL]] - 1 or [[xy] + disp] ← [[xy] + disp] - 1 Decrement using implied addressing or base relative addressing.
		(xy + disp)	11x11101 35 disp	3	23							
Memory Shift and Rotate	RLC	(HL)	CB 06	2	15	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Rotate contents of memory location (implied or base relative addressing) left with branch Carry.
		(xy + disp)	11x11101 CB disp 06	4	23							
	RL	(HL)	CB 16	2	15	X	X	X	P	0	0	 [[HL]] or [[xv] + disp] Rotate contents of memory location left through Carry.
		(xy + disp)	11x11101 CB disp 16	4	23							
	RRC	(HL)	CB 0E	2	15	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Rotate contents of memory location right with branch Carry.
		(xy + disp)	11x11101 CB disp 0E	4	23							

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

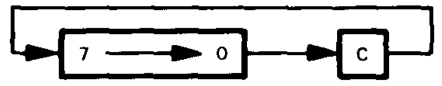
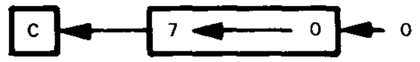
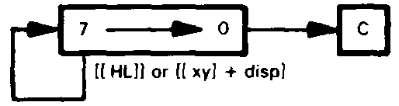
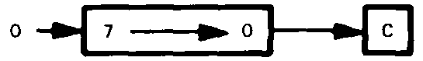
Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Memory Shift and Rotate (Continued)	RR	(HL) (xy + disp)	CB 1E 11x11101 CB disp 1E	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Rotate contents of memory location right through Carry
	SLA	(HL) (xy + disp)	CB 26 11x11101 CB disp 26	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Shift contents of memory location left and clear LSB (Arithmetic Shift).
	SRA	(HL) (xy + disp)	CB 2E 11x11101 CB disp 2E	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Shift contents of memory location right and preserve MSB (Arithmetic Shift).
	SRL	(HL) (xy + disp)	CB 3E 11x11101 CB disp 3E	2 4	15 23	X	X	X	P	0	0	 [[HL]] or [[xy] + disp] Shift contents of memory location right and clear MSB (Logical Shift)

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Immediate	LD	reg,data	00ddd110 yy	2	7							[reg] ← data Load immediate into register.
	LD	rp,data16 xy,data16	00xx0001 yyyy 11x11101 21 yyyy	3 4	10 14							[rp] ← data16 or [xy] ← data16 Load 16 bits of immediate data into register pair or Index register.
	LD	(HL),data (xy + disp), data	36 yy 11x11101 36 disp yy	2 4	10 19							[[HL]] ← data or [[xy] + disp] ← data Load immediate into memory location using implied or base relative addressing.
Jump	JP	label	C3 ppqq	3	10							[PC] ← label Jump to instruction at address represented by label.
	JR	disp	18 (disp-2)	2	12							[PC] ← [PC] + 2 + (disp-2) Jump relative to present contents of Program Counter.
	JP	(HL) (xy)	E9 11x11101 E9	1 2	4 8							[PC] ← [HL] or [PC] ← [xy] Jump to address contained in HL or Index register.
Subroutine Call and Return	CALL	label	CD ppqq	3	17							[[SP] - 1] ← [PC(HI)] [[SP] - 2] ← [PC(LO)] [SP] ← [SP] - 2 [PC] ← label Jump to subroutine starting at address represented by label.
	CALL	cond,label	11ccc100 ppqq	3	10/17							Jump to subroutine if condition is satisfied; otherwise, continue in sequence.
	RET		C9	1	10							[PC(LO)] ← [[SP]] [PC(HI)] ← [[SP] + 1] [SP] ← [SP] + 2 Return from subroutine.
	RET	cond	11ccc000	1	5/11							Return from subroutine if condition is satisfied; otherwise, continue in sequence.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Immediate Operate	ADD	A,data	C6 yy	2	7	X	X	X	O	X	0	[A] $\leftarrow$ [A] + data Add immediate to Accumulator.
	ADC	A,data	CE yy	2	7	X	X	X	O	X	0	[A] $\leftarrow$ [A] + data + C Add immediate with Carry.
	SUB	data	D6 yy	2	7	X	X	X	O	X	1	[A] $\leftarrow$ [A] - data Subtract immediate from Accumulator.
	SBC	A,data	DE yy	2	7	X	X	X	O	X	1	[A] $\leftarrow$ [A] - data - C Subtract immediate with Carry.
	AND	data	E6 yy	2	7	0	X	X	P	1	0	[A] $\leftarrow$ [A] $\wedge$ data AND immediate with Accumulator.
	OR	data	F6 yy	2	7	0	X	X	P	1	0	[A] $\leftarrow$ [A] $\vee$ data OR immediate with Accumulator.
	XOR	data	EE yy	2	7	0	X	X	P	1	0	[A] $\leftarrow$ [A] $\veebar$ data Exclusive-OR immediate with Accumulator.
	CP	data	FE yy	2	7	X	X	X	O	X	1	[A] - data Compare immediate data with Accumulator contents; only the flags are affected.
Jump on Condition	JP	cond,label	11ccc010 ppqq	3	10							If cond, then [PC] $\leftarrow$ label Jump to instruction at address represented by label if the condition is true.
	JR	C,disp	38 (disp-2)	2	7/12							If C = 1, then [PC] $\leftarrow$ [PC] + 2 + (disp − 2) Jump relative to contents of Program Counter if Carry flag is set.
	JR	NC,disp	30 (disp-2)	2	7/12							If C = 0, then [PC] $\leftarrow$ [PC] + 2 + (disp −2) Jump relative to contents of Program Counter if Carry flag is reset.
	JR	Z,disp	28 (disp-2)	2	7/12							If Z = 1, then [PC] $\leftarrow$ [PC] + 2 + (disp −2) Jump relative to contents of Program Counter if Zero flag is set.
	JR	NZ,disp	20 (disp-2)	2	7/12							If Z = 0, then [PC] $\leftarrow$ [PC] + 2 + (disp −2) Jump relative to contents of Program Counter if Zero flag is reset.
	DJNZ	disp	10 (disp-2)	2	8/13							[B] $\leftarrow$ [B] - 1 If [B] $\neq$ 0, then [PC] + 2 + (disp −2) Decrement contents of B and Jump relative to contents of Program Counter if result is not 0.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Register-Register Move	LD	dst,src	01dddsss	1	4							[dst] ← [src] Move contents of source register to destination register. Register designations src and dst may each be A, B, C, D, E, H or L.
	LD	A,I	ED 57	2	9		X	X	I	0	0	[A] ← [I] Move contents of Interrupt Vector register to Accumulator.
	LD	A,R	ED 5F	2	9		X	X	I	0	0	[A] ← [R] Move contents of Refresh register to Accumulator.
	LD	I,A	ED 47	2	9							[I] ← [A] Load Interrupt Vector register from Accumulator.
	LD	R,A	ED 4F	2	9							[R] ← [A] Load Refresh register from Accumulator.
	LD	SP,HL	F9	1	6							[SP] ← [HL] Move contents of HL to Stack Pointer.
	LD	SP,xy	11x11101 F9	2	10							[SP] ← [xy] Move contents of Index register to Stack Pointer.
	EX	DE,HL	EB	1	4							[DE] ←→ [HL] Exchange contents of DE and HL.
	EX	AF,AF'	08	1	4							[AF] ←→ [AF'] Exchange program status and alternate program status.
	EXX		D9	1	4							$\begin{pmatrix}[BC]\\ [DE]\\ [HL]\end{pmatrix} \longleftrightarrow \begin{pmatrix}[BC']\\ [DE']\\ [HL']\end{pmatrix}$ Exchange register pairs and alternate register pairs.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Register-Register Operate	ADD	A,reg	10000rrr	1	4	X	X	X	O	X	0	[A] ← [A] + [reg] Add contents of register to Accumulator.
	ADC	A,reg	10001rrr	1	4	X	X	X	O	X	0	[A] ← [A] + [reg] + C Add contents of register and Carry to Accumulator.
	SUB	reg	10010rrr	1	4	X	X	X	O	X	1	[A] ← [A] - [reg] Subtract contents of register from Accumulator.
	SBC	A,reg	10011rrr	1	4	X	X	X	O	X	1	[A] ← [A] - [reg] - C Subtract contents of register and Carry from Accumulator.
	AND	reg	10000rrr	1	4	0	X	X	P	1	0	[A] ← [A] Λ [reg] AND contents of register with contents of Accumulator.
	OR	reg	10110rrr	1	4	0	X	X	P	1	0	[A] ← [A] V [reg] OR contents of register with contents of Accumulator.
	XOR	reg	10101rrr	1	4	0	X	X	P	1	0	[A] ← [A] ⊻ [reg] Exclusive-OR contents of register with contents of Accumulator.
	CP	reg	10111rrr	1	4	X	X	X	O	X	1	[A] - [reg] Compare contents of register with contents of Accumulator. Only the flags are affected.
	ADD	HL,rp	00xx1001	1	11	X				?	0	[HL] ← [HL] + [rp] 16-bit add register pair contents to contents of HL.
	ADC	HL,rp	ED 01xx1010	2	15	X	X	X	O	?	0	[HL] ← [HL] + [rp] + C 16-bit add with Carry register pair contents to contents of HL.
	SBC	HL,rp	ED 01xx0010	2	15	X	X	X	O	?	1	[HL] ← [HL] - [rp] - C 16-bit subtract with Carry register pair contents from contents of HL.
	ADD	IX,pp	DD 00xx1001	2	15	X				?	0	[IX] ← [IX] + [pp] 16-bit add register pair contents to contents of Index register IX (pp = BC, DE, IX, SP).
	ADD	IY,rr	FD 00xx1001	2	15	X				?	0	[IY] ← [IY] + [rr] 16-bit add register pair contents to contents of Index register IY (rr = BC, DE, IY, SP).

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

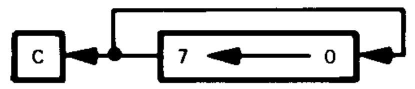
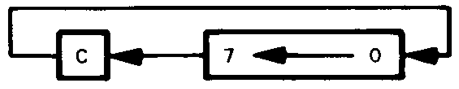
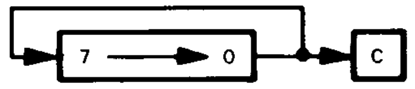
Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Register Operate	DAA		27	1	4	X	X	X	P	X		Decimal adjust Accumulator, assuming that Accumulator contents are the sum or difference of BCD operands.
	CPL		2F	1	4					1	1	$[A] \leftarrow [\overline{A}]$ Complement Accumulator (ones complement).
	NEG		ED 44	2	8	X	X	X	O	X	1	$[A] \leftarrow [\overline{A}] + 1$ Negate Accumulator (twos complement).
	INC	reg	00rrr100	1	4		X	X	O	X	0	[reg] ← [reg] + 1 Increment register contents.
	INC	rp xy	00xx0011 11x11101 23	1 2	6 10							[rp] ← [rp] + 1 or [xy] ← [xy] + 1 Increment contents of register or Index register.
	DEC	reg	0Urrr101	1	4		X	X	O	X	1	[reg] ← [reg] - 1 Decrement register contents.
	DEC	rp xy	00xx1011 11x11101 2B	1 2	6 10							[rp] ← [rp] - 1 or [xy] ← [xy] - 1 Decrement contents of register pair or Index register.
Register Shift and Rotate	RLCA		07	1	4	X				0	0	 [A] Rotate Accumulator left with branch Carry.
	RLA		17	1	4	X				0	0	 [A] Rotate Accumulator left through Carry.
	RRCA		0F	1	4	X				0	0	 [A] Rotate Accumulator right with branch Carry.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

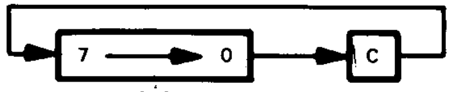
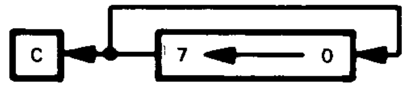
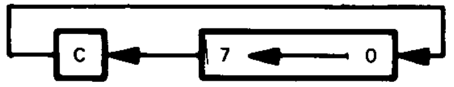
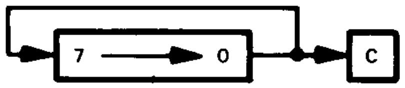
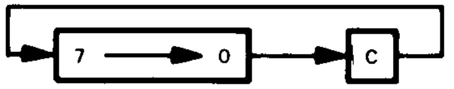
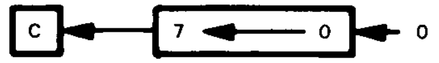
Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Register Shift and Rotate (Continued)	RRA		1F	1	4	X				0	0	 [A] Rotate Accumulator right through Carry.
	RLC	reg	CB 00000rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register left with branch Carry.
	RL	reg	CB 00010rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register left through Carry.
	RRC	reg	CB 00001rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register right with branch Carry.
	RR	reg	CB 00011rrr	2	8	X	X	X	P	0	0	 [reg] Rotate contents of register right through Carry.
	SLA	reg	CB 00100rrr	2	8	X	X	X	P	0	0	 [reg] Shift contents of register left and clear LSB (Arithmetic Shift).

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

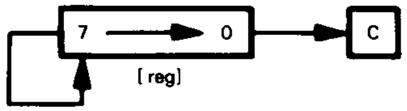
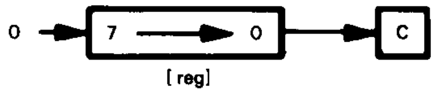
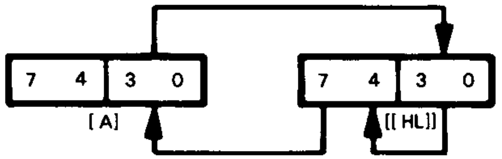
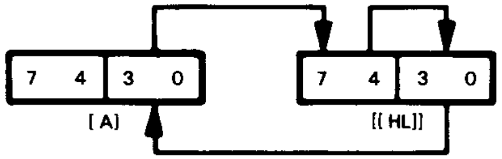
Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Register Shift and Rotate (Continued)	SRA	reg	CB 00101rrr	2	8	X	X	X	P	0	0	 Shift contents of register right and preserve MSB (Arithmetic Shift).
	SRL	reg	CB 00111rrr	2	8	X	X	X	P	0	0	 Shift contents of register right and clear MSB (Logical Shift).
	RLD		ED 6F	2	18		X	X	P	0	0	 Rotate one BCD digit left between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.
	RRD		ED 67	2	18		X	X	P	0	0	 Rotate one BCD digit right between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Bit Manipulation	BIT	b,reg	CB 01bbbrrr	2	8		X	?	?	1	0	Z ← $\overline{\text{reg(b)}}$ Zero flag contains complement of the selected register bit.
	BIT	b,(HL) b,(xy + disp)	CB 01bbb110 11x11101 CB disp 01bbb110	2 4	12 20		X	?	?	1	0	Z ← $\overline{\text{[[HL]](b)}}$ or Z ← $\overline{\text{[[xy] + disp](b)}}$ Zero flag contains complement of selected bit of the memory location (implied addressing or base relative addressing).
	SET	b,reg	CB 11bbbrrr	2	8							reg(b) ← 1 Set indicated register bit.
	SET	b,(HL) b,(xy+disp)	CB 11bbb110 11x11101 CB disp 11bbb110	2 4	15 23							[[HL]](b) ← 1 or [[xy] + disp](b) ← 1 Set indicated bit of memory location (implied addressing or base relative addressing).
	RES	b,reg	CB 10bbbrrr	2	8							reg(b) ← 0 Reset indicated register bit.
	RES	b,(HL) b,(xy + disp)	CB 10bbb110 11x11101 CB disp 10bbb110	2 4	15 23							[[HL]](b) ← 0 or [[xy] + disp](b) ← 0 Reset indicated bit in memory location (implied addressing or base relative addressing).
Stack	PUSH	pr xy	11xx0101 11x11101 E5	1 2	11 15							[[SP]-1] ← [pr(HI)] [[SP]-2] ← [pr(LO)] [SP] ← [SP]-2 Put contents of register pair or Index register on top of Stack and decrement Stack Pointer.
	POP	pr xy	11xx0001 11x11101 E1	1 2	10 14							[pr(LO)] ← [[SP]] [pr(HI)] ← [[SP] + 1] [SP] ← [SP] + 2 Put contents of top of Stack in register pair or Index register and increment Stack Pointer.
	EX	(SP),HL (SP),xy	E3 11x11101 E3	1 2	19 23							[H] ←→ [[SP] + 1] [L] ←→ [[SP]] Exchange contents of HL or Index register and top of Stack.

Table 3-4. A Summary of the Z80 Instruction Set (Continued)

Type	Mnemonic	Operand	Object Code	Bytes	Clock Cycles	Status						Operation Performed
						C	Z	S	P/O	A_C	N	
Interrupt	DI		F3	1	4							Disable interrupts.
	EI		FB	1	4							Enable interrupts.
	RST	n	11xxx111	1	11							[[SP]-1] ← [PC(HI)] [[SP]-2] ← [PC(LO)] [SP] ← [SP]-2 [PC] ← $(8 \cdot n)_{16}$ Restart at designated location.
	RETI		ED 4D	2	14							Return from interrupt.
	RETN		ED 45	2	14							Return from nonmaskable interrupt.
	IM	0	ED 46	2	8							Set interrupt mode 0, 1, or 2.
		1	ED 56	2	8							
		2	ED 5E	2	8							
Status	SCF		37	1	4	1				0	0	C ← 1 Set Carry flag.
	CCF		3F	1	4	X				?	0	C ← $\overline{C}$ Complement Carry flag.
	NOP		00	1	4							No operation — volatile memories are refreshed.
	HALT		76	1	4							CPU halts, executes NOPs to refresh volatile memories.

**Execution time shown is for one iteration.

Table 3-5. Instruction Object Codes in Numerical Order

OBJECT CODE	INSTRUCTION	
00	NOP	
01 yyyy	LD	BC,data16
02	LD	(BC),A
03	INC	BC
04	INC	B
05	DEC	B
06 yy	LD	B,data
07	RLCA	
08	EX	AF,AF'
09	ADD	HL,BC
0A	LD	A,(BC)
0B	DEC	BC
0C	INC	C
0D	DEC	C
0E yy	LD	C,data
0F	RRCA	
10 disp-2	DJNZ	disp
11 yyyy	LD	DE,data16
12	LD	(DE),A
13	INC	DE
14	INC	D
15	DEC	D
16 yy	LD	D,data
17	RLA	
18 disp-2	JR	disp
19	ADD	HL,DE
1A	LD	A,(DE)
1B	DEC	DE
1C	INC	E
1D	DEC	E
1E yy	LD	E,data
1F	RRA	
20 disp-2	JR	NZ,disp
21 yyyy	LD	HL,data16
22 ppqq	LD	(addr),HL
23	INC	HL
24	INC	H
25	DEC	H
26 yy	LD	H,data
27	DAA	
28 disp-2	JR	Z,disp
29	ADD	HL,HL
2A ppqq	LD	HL,(addr)
2B	DEC	HL
2C	INC	L
2D	DEC	L
2E	LD	L,data
2F	CPL	
30 disp-2	JR	NC,disp
31 yyyy	LD	SP,data16
32 ppqq	LD	(addr),A
33	INC	SP
34	INC	(HL)
35	DEC	(HL)
36 yy	LD	(HL),data
37	SCF	
38	JR	C,disp
39	ADD	HL,SP
3A ppqq	LD	A,(addr)
3B	DEC	SP
3C	INC	A
3D	DEC	A
3E yy	LD	A,data
3F	CCF	
4 0sss	LD	B,reg
46	LD	B,(HL)
4 1sss	LD	C,reg
4E	LD	C,(HL)
5 0sss	LD	D,reg
56	LD	D,(HL)
5 1sss	LD	E,reg
5E	LD	E,(HL)
6 0sss	LD	H,reg
66	LD	H,(HL)
6 1sss	LD	L,reg
6E	LD	L,(HL)
7 0sss	LD	(HL),reg
76	HALT	
7 1sss	LD	A,reg
7E	LD	A,(HL)
8 0rrr	ADD	A,reg
86	ADD	A,(HL)
8 1rrr	ADC	A,reg
8E	ADC	A,(HL)
9 0rrr	SUB	reg
96	SUB	(HL)
9 1rrr	SBC	A,reg
9E	SBC	A,(HL)
A 0rrr	AND	reg
A6	AND	(HL)
A 1rrr	XOR	reg
AE	XOR	(HL)
B 0rrr	OR	reg
B6	OR	(HL)
B 1rrr	CP	reg
BE	CP	(HL)
C0	RET	NZ
C1	POP	BC
C2 ppqq	JP	NZ,addr
C3 ppqq	JP	addr
C4 ppqq	CALL	NZ,addr
C5	PUSH	BC
C6 yy	ADD	A,data
C7	RST	00H
C8	RET	Z
C9	RET	
CA ppqq	JP	Z,addr
CB 0 0rrr	RLC	reg
CB 06	RLC	(HL)
CB 0 1rrr	RRC	reg
CB 0E	RRC	(HL)
CB 1 0rrr	RL	reg
CB 16	RL	(HL)
CB 1 1rrr	RR	reg

Table 3-5. Instruction Object Codes in Numerical Order (Continued)

OBJECT CODE	INSTRUCTION	
CB 1E	RR	(HL)
CB 2 0rrr	SLA	reg
CB 26	SLA	(HL)
CB 2 1rrr	SRA	reg
CB 2E	SRA	(HL)
CB 3 1rrr	SRL	reg
CB 3E	SRL	(HL)
CB 01bbbrrr	BIT	b,reg
CB 01bbb110	BIT	b,(HL)
CB 10bbbrrr	RES	b,reg
CB 10bbb110	RES	b,(HL)
CB 11bbbrrr	SET	b,reg
CB 11bbb110	SET	b,(HL)
CC ppqq	CALL	Z,addr
CD ppqq	CALL	addr
CE yy	ADC	A,data
CF	RST	08H
D0	RET	NC
D1	POP	DE
D2 ppqq	JP	NC,addr
D3 yy	OUT	(port),A
D4 ppqq	CALL	NC,addr
D5	PUSH	DE
D6 yy	SUB	data
D7	RST	10H
D8	RET	C
D9	EXX	
DA ppqq	JP	C,addr
DB yy	IN	A,(port)
DC ppqq	CALL	C,addr
DD 00xx 9	ADD	IX,pp
DD 21 yyyy	LD	IX,data16
DD 22 ppqq	LD	(addr),IX
DD 23	INC	IX
DD 2A ppqq	LD	IX,(addr)
DD 2B	DEC	IX
DD 34 disp	INC	(IX + disp)
DD 35 disp	DEC	(IX + disp)
DD 36 disp yy	LD	(IX + disp),data
DD 01ddd110 disp	LD	reg,(IX + disp)
DD 7 0sss disp	LD	(IX + disp),reg
DD 86 disp	ADD	A,(IX + disp)
DD 8E disp	ADC	A,(IX + disp)
DD 96 disp	SUB	(IX + disp)
DD 9E disp	SBC	A,(IX + disp)
DD A6 disp	AND	(IX + disp)
DD AE disp	XOR	(IX + disp)
DD B6 disp	OR	(IX + disp)
DD BE disp	CP	(IX + disp)
DD CB disp 06	RLC	(IX + disp)
DD CB disp 0E	RRC	(IX + disp)
DD CB disp 16	RL	(IX + disp)
DD CB disp 1E	RR	(IX + disp)
DD CB disp 26	SLA	(IX + disp)
DD CB disp 2E	SRA	(IX + disp)
DD CB disp 3E	SRL	(IX + disp)
DD CB disp 01bbb110	BIT	b,(IX + disp)
DD CB disp 10bbb110	RES	b,(IX + disp)
DD CB disp 11bbb110	SET	b,(IX + disp)
DD E1	POP	IX
DD E3	EX	(SP),IX
DD E5	PUSH	IX
DD E9	JP	(IX)
DD F9	LD	SP,IX
DE yy	SBC	A,data
DF	RST	18H
E0	RET	PO
E1	POP	HL
E2 ppqq	JP	PO,addr
E3	EX	(SP),HL
E4 ppqq	CALL	PO,addr
E5	PUSH	HL
E6 yy	AND	data
E7	RST	20H
E8	RET	PE
E9	JP	(HL)
EA ppqq	JP	PE,addr
EB	EX	DE,HL
EC ppqq	CALL	PE,addr
ED 01ddd000	IN	reg,(C)
ED 01sss001	OUT	(C),reg
ED 01xx 2	SBC	HL,rp
ED 01xx 3 ppqq	LD	(addr),rp
ED 44	NEG	
ED 45	RETN	
ED 010nn110	IM	m
ED 47	LD	I,A
ED 01xx A	ADC	HL,rp
ED 01xx B ppqq	LD	rp,(addr)
ED 4D	RETI	
ED 4F	LD	R,A
ED 57	LD	A,I
ED 5F	LD	A,R
ED 67	RRD	
ED 6F	RLD	
ED A0	LDI	
ED A1	CPI	
ED A2	INI	
ED A3	OUTI	
ED A8	LDD	
ED A9	CPD	
ED AA	IND	
ED AB	OUTD	
ED B0	LDIR	
ED B1	CPIR	
ED B2	INIR	
ED B3	OTIR	
ED B8	LDDR	
ED B9	CPDR	
ED BA	INDR	
ED BB	OTDR	
EE yy	XOR	data
EF	RST	28H

Table 3-5. Instruction Object Codes in Numerical Order (Continued)

OBJECT CODE	INSTRUCTION	
F0	RET	P
F1	POP	AF
F2 ppqq	JP	P,addr
F3	DI	
F4 ppqq	CALL	P,addr
F5	PUSH	AF
F6 yy	OR	data
F7	RST	30H
F8	RET	M
F9	LD	SP,HL
FA ppqq	JP	M,addr
FB	EI	
FC ppqq	CALL	M,addr
FD 00xx 9	ADD	IY,rr
FD 21 yyyy	LD	IY,data16
FD 22 ppqq	LD	(addr),IY
FD 23	INC	IY
FD 2A ppqq	LD	IY,(addr)
FD 2B	DEC	IY
FD 34 disp	INC	(IY + disp)
FD 35 disp	DEC	(IY + disp)
FD 36 disp yy	LD	(IY + disp),data
FD 01ddd110 disp	LD	reg,(IY + disp)
FD 7 0sss disp	LD	(IY + disp),reg
FD 86 disp	ADD	A,(IY + disp)
FD 8E disp	ADC	A,(IY + disp)
FD 96 disp	SUB	(IY + disp)
FD 9E disp	SBC	A,(IY + disp)
FD A6 disp	AND	(IY + disp)
FD AE disp	XOR	(IY + disp)
FD B6 disp	OR	(IY + disp)
FD BE disp	CP	(IY + disp)
FD CB disp 06	RLC	(IY + disp)
FD CB disp 0E	RRC	(IY + disp)
FD CB disp 16	RL	(IY + disp)
FD CB disp 1E	RR	(IY + disp)
FD CB disp 26	SLA	(IY + disp)
FD CB disp 2E	SRA	(IY + disp)
FD CB disp 3E	SRL	(IY + disp)
FD CB disp 01bbb110	BIT	b,(IY + disp)
FD CB disp 10bbb110	RES	b,(IY + disp)
FD CB disp 11bbb110	SET	b,(IY + disp)
FD E1	POP	IY
FD E3	EX	(SP),IY
FD E5	PUSH	IY
FD E9	JP	(IY)
FD F9	LD	SP,IY
FE yy	CP	data
FF	RST	38H