

 OSBORNE/McGraw-Hill

Z80

ASSEMBLY LANGUAGE PROGRAMMING



Lance A. Leventhal

Scanned and converted to PDF by HansO, 2001

Chapter 3 page 1-42

Contents (Continued)

Chapter		Page
3	The Z80 Assembly Language Instruction Set	3-1
	CPU Registers and Status Flags	3-2
	Z80 Memory Addressing Modes	3-4
	Implied	3-5
	Implied Block Transfer with Auto-Increment/Decrement	3-7
	Implied Stack	3-8
	Indexed	3-10
	Direct	3-11
	Program Relative	3-12
	Base Page	3-13
	Register Direct	3-14
	Immediate	3-15
	Abbreviations	3-18
	Instruction Mnemonics	3-21
	Instruction Object Codes	3-21
	Instruction Execution Times	3-21
	Status	3-21
	Instruction Descriptions	3-43
	8080A/Z80 Compatibility	3-164
	Zilog Z80 Assembler Conventions	3-170
	Assembler Field Structure	3-170
	Labels	3-170
	Reserved Names	3-170
	Pseudo-operations	3-170
	Examples	3-171
	Labels with Pseudo-operations	3-172
	Addresses	3-172
	Conditional Assembly	3-174
	Macros	3-174

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.**

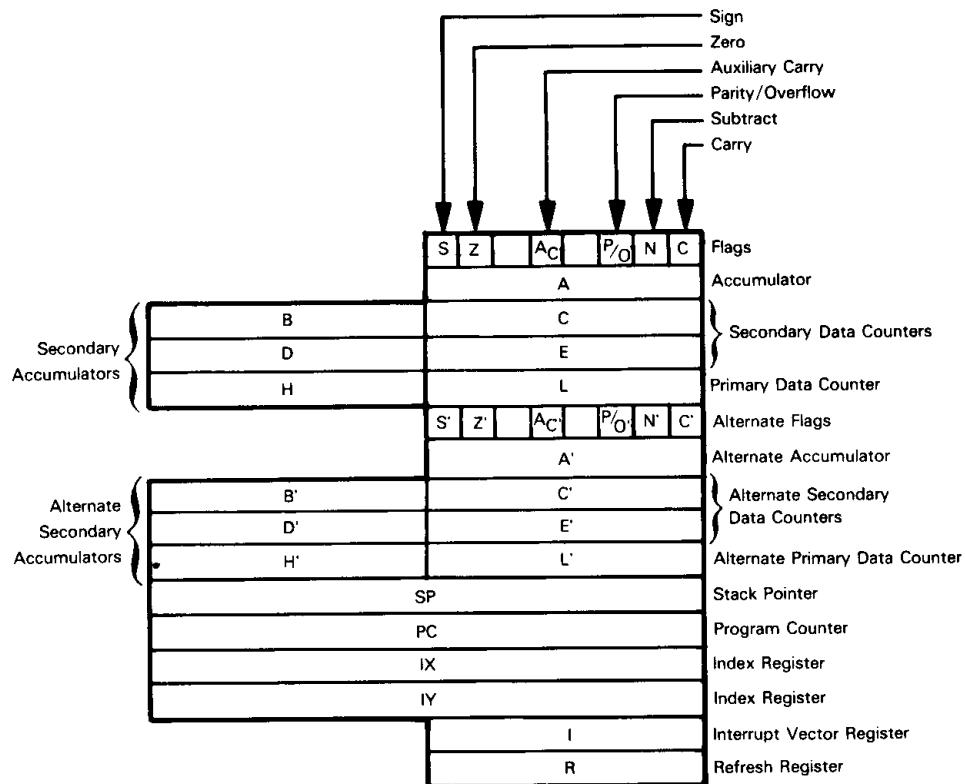
Interrupts, direct memory access, and the Stack architecture for the Z80 will be described in later chapters of this book, in conjunction with assembly language programming discussions of the same subjects.

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                    Low-order  
byte                            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 non-zero 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:**

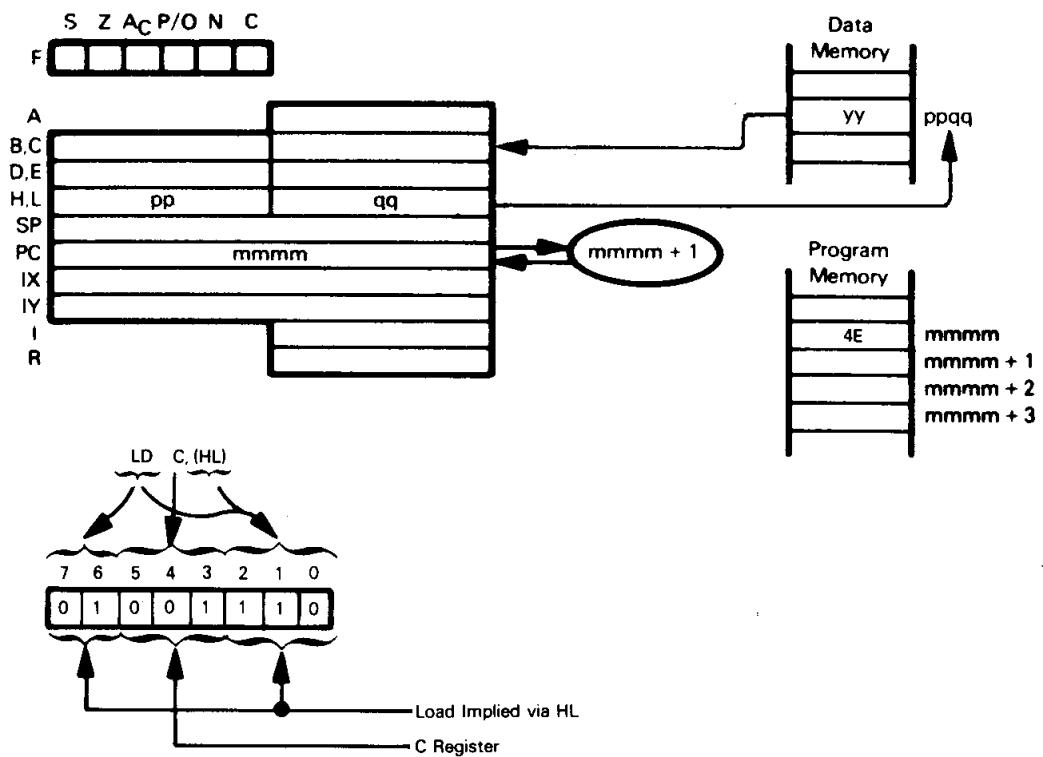
- **Implied**
- **Implied Block Transfer with Auto-Increment/Decrement**
- **Implied Stack**
- **Indexed**
- **Direct**
- **Program Relative**
- **Base Page**
- **Register Indirect**
- **Immediate**

## 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)

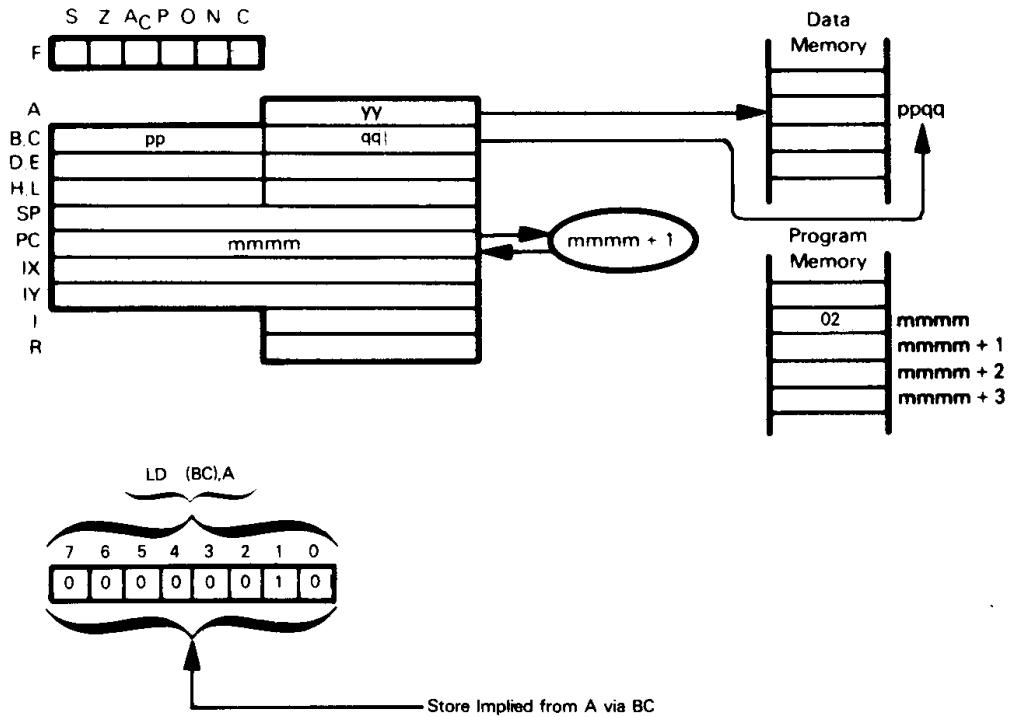
loads the C register with the contents of the memory location currently pointed to by HL. This is illustrated as follows:



**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:



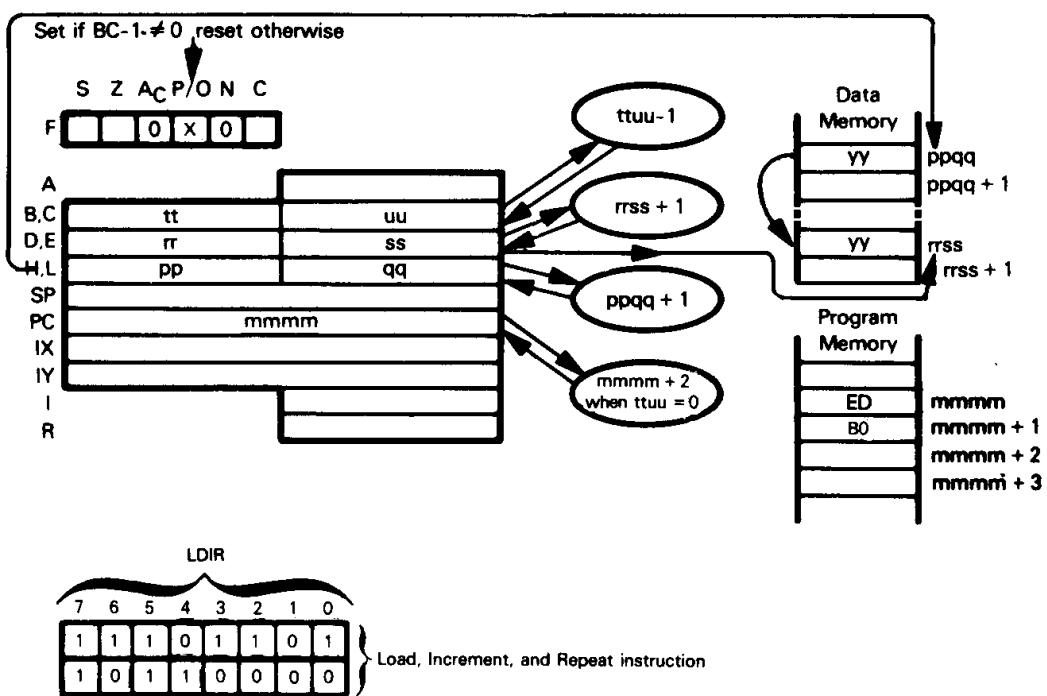
## Implied Block Transfer With Auto-Increment/Decrement

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:



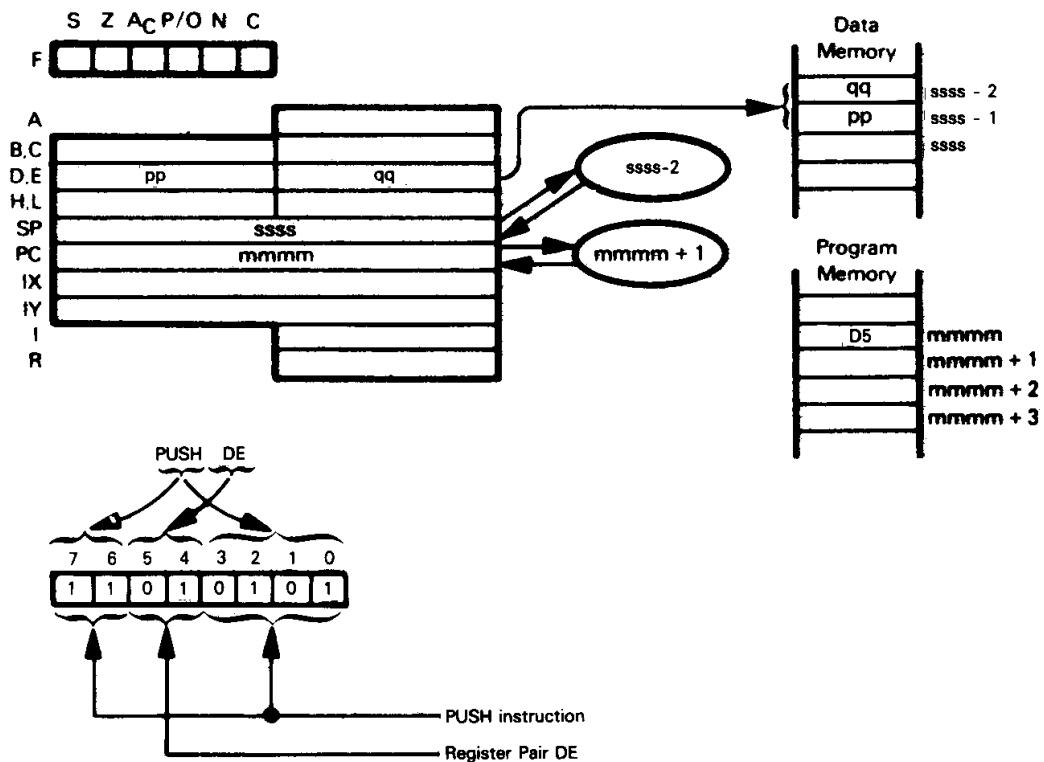
A similar group of Input/Output instructions is provided, allowing a block of data to be input or output between memory and an I/O device. The I/O port number is taken as the contents of the C register, with the single B register used as the byte counter. Memory is addressed by HL.

## 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

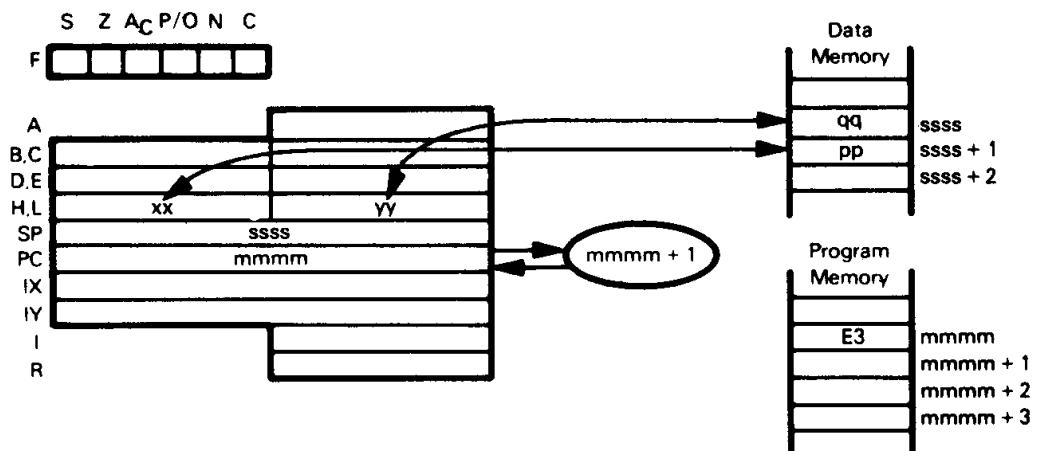
is illustrated as follows:



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:

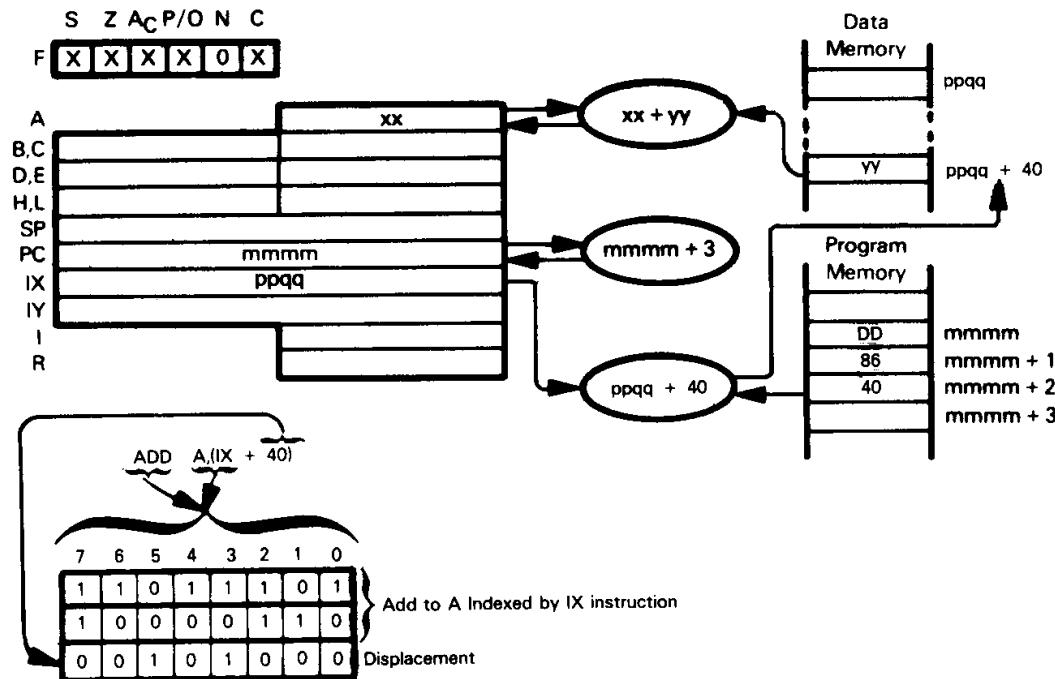


## Indexed

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).

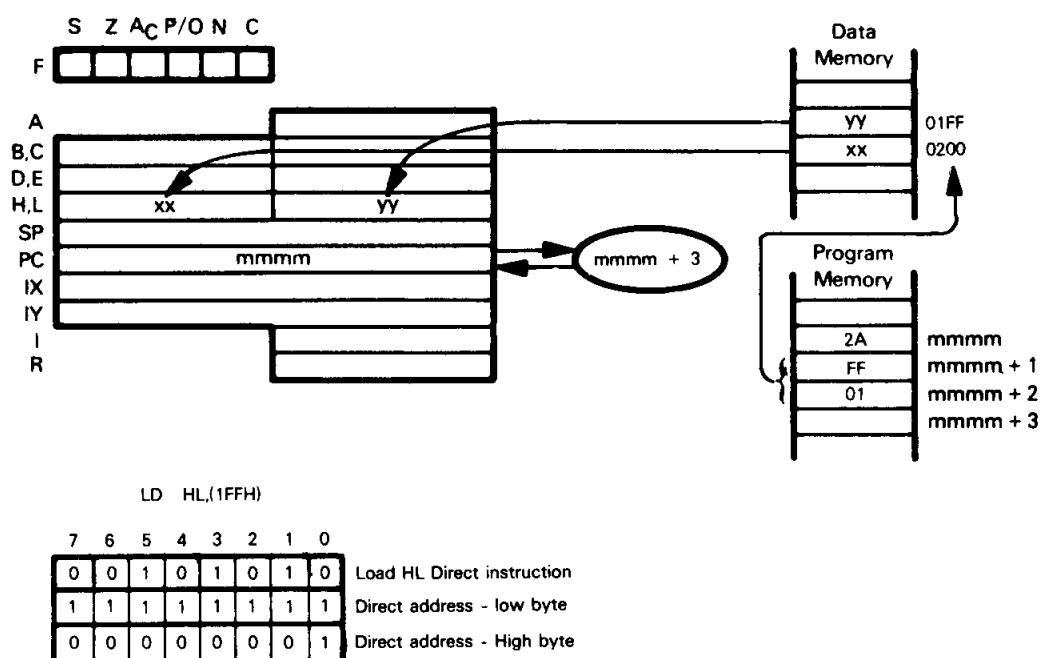
The instruction

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:



**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.

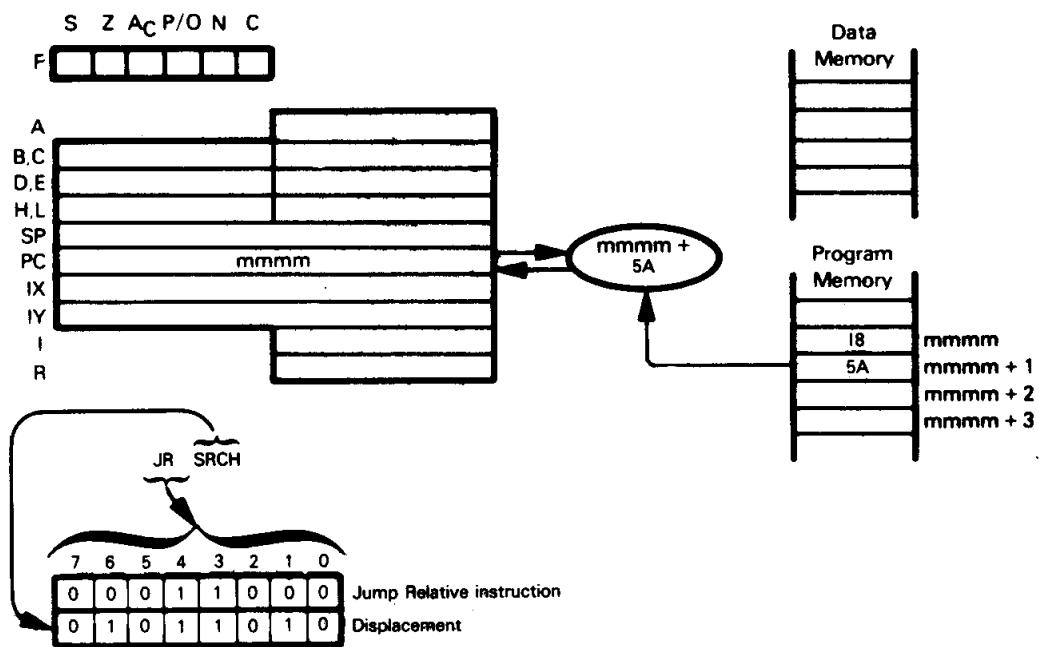
## Program Relative

**Jump Relative instructions provide program relative addressing in the range -128, +128 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

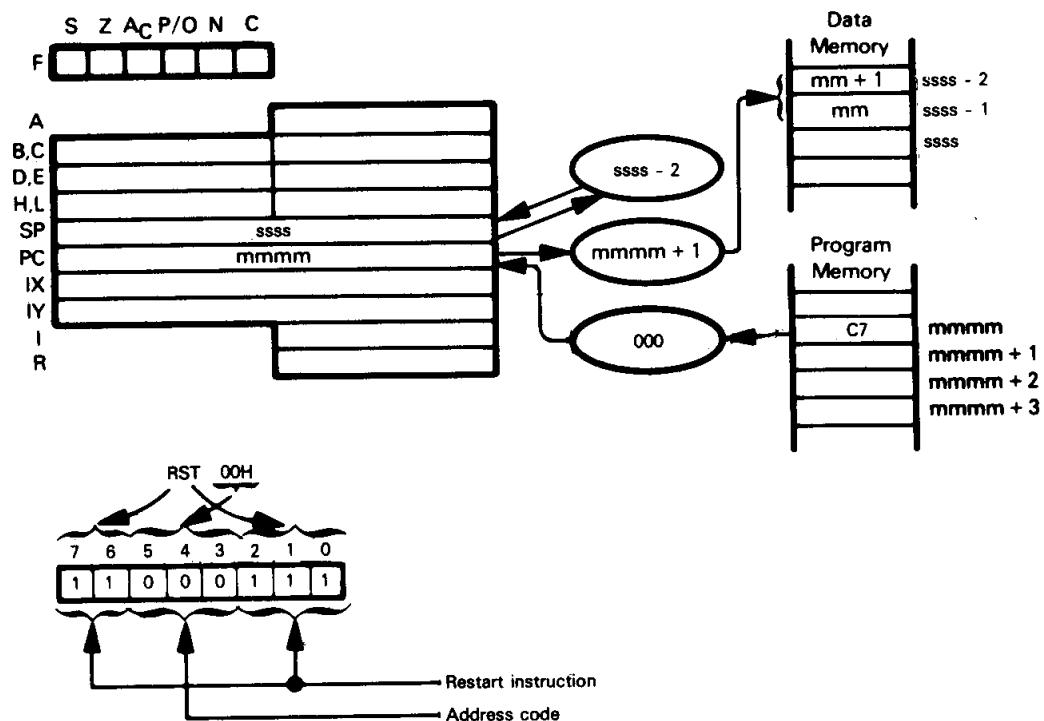
The Z80 has a **modified base page addressing** mode for the Restart instruction. This is a special Call instruction that **allows a single-byte instruction to jump to one of eight subroutines located at specific points in lower core**. The effective address is calculated from a 3-bit code stored in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



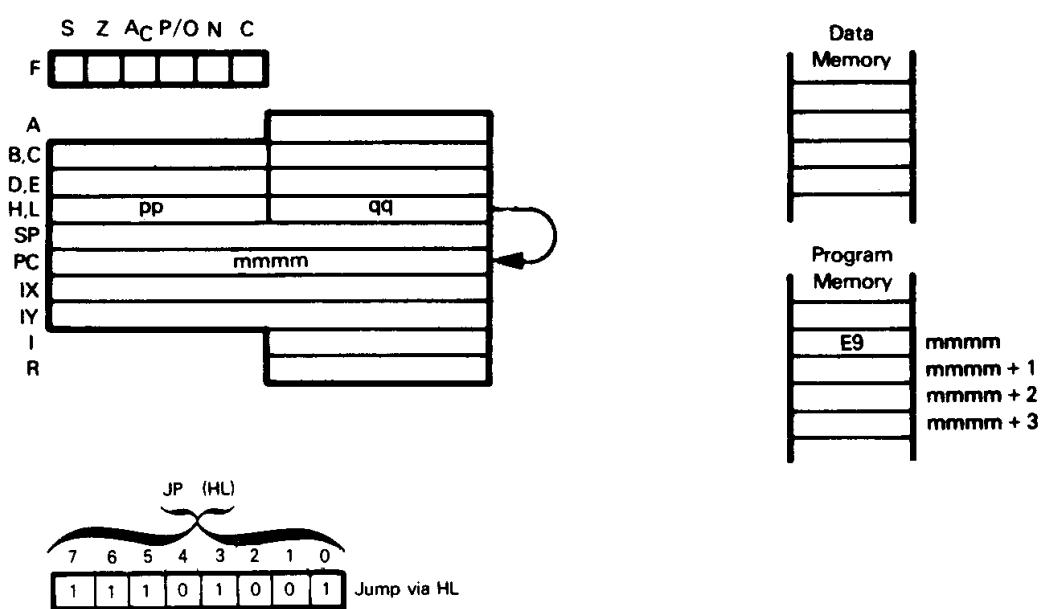
## Register Indirect

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<sub>16</sub> into Register Pair BC. This may be illustrated as follows:

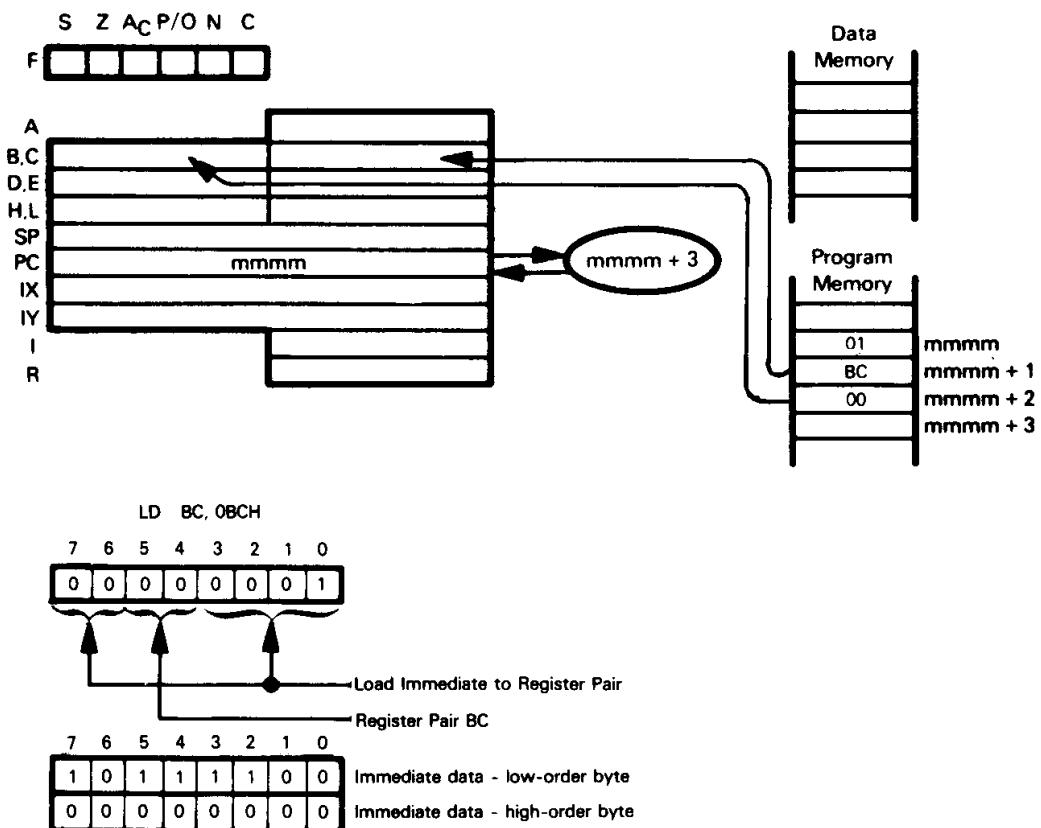


Table 3-1. Frequently Used Instructions of the Z80

| <b>Instruction Code</b> | <b>Meaning</b>                         |
|-------------------------|----------------------------------------|
| 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:<br>NZ - Non-Zero (Z = 0)<br>Z - Zero (Z = 1)<br>NC - Non-carry (C = 0)<br>C - Carry (C = 1)<br>PO - Parity Odd (P = 0)<br>PE - Parity Even (P = 1)<br>P - Positive Sign (S = 0)<br>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:<br>BC<br>DE<br>HL<br>AF                                                                                                                                                        |                                                                           |
| R           | The Refresh register (8 bits)                                                                                                                                                                                       |                                                                           |
| reg         | Any of the following registers:<br>A<br>B<br>C<br>D<br>E<br>H<br>L                                                                                                                                                  |                                                                           |
| rp          | Any of the following register pairs:<br>BC<br>DE<br>HL<br>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)<br>ccc Condition code    000 = non-zero<br>001 = zero<br>010 = no carry<br>011 = carry<br>100 = parity odd<br>101 = parity even<br>110 = positive sign<br>111 = negative sign |                                                                           |
| ddd         | Destination register — same coding as rrr                                                                                                                                                                           |                                                                           |
| ppqq        | A 16-bit memory address                                                                                                                                                                                             |                                                                           |
| rrr         | Register                                                                                                                                                                                                            | 111 = A<br>000 = B<br>001 = C<br>010 = D<br>011 = E<br>100 = H<br>101 = L |
| sss         | Source register — same coding as rrr                                                                                                                                                                                |                                                                           |
| x           | Index register                                                                                                                                                                                                      | 0 = IX<br>1 = IY                                                          |
| xx          | Register pair                                                                                                                                                                                                       | 00 = BC<br>01 = DE<br>10 = HL<br>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                | <p>The Z80 has the following status flags:</p> <table> <tbody> <tr><td>C</td><td>- Carry status</td></tr> <tr><td>Z</td><td>- Zero status</td></tr> <tr><td>S</td><td>- Sign status</td></tr> <tr><td>P/O</td><td>- Parity/Overflow status</td></tr> <tr><td>AC</td><td>- Auxiliary Carry status</td></tr> <tr><td>N</td><td>- Subtract status</td></tr> </tbody> </table> <p>The following symbols are used in the status columns:</p> <table> <tbody> <tr><td>X</td><td>- flag is affected by operation</td></tr> <tr><td>(blank)</td><td>- flag is not affected by operation</td></tr> <tr><td>1</td><td>- flag is set by operation</td></tr> <tr><td>0</td><td>- flag is reset by operation</td></tr> <tr><td>U</td><td>- flag is unknown after operation</td></tr> <tr><td>P</td><td>- flag shows parity status</td></tr> <tr><td>O</td><td>- flag shows overflow status</td></tr> <tr><td>I</td><td>- flag shows interrupt enabled/disabled status</td></tr> </tbody> </table> | C | - Carry status | Z | - Zero status | S | - Sign status | P/O | - Parity/Overflow status | AC | - Auxiliary Carry status | N | - Subtract status | 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 |
| C                       | - Carry status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| Z                       | - Zero status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| S                       | - Sign status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| P/O                     | - Parity/Overflow status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| AC                      | - Auxiliary Carry status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| N                       | - Subtract status                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| 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                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| $\vee$                  | Logical OR                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |   |                |   |               |   |               |     |                          |    |                          |   |                   |   |                                 |         |                                     |   |                            |   |                              |   |                                   |   |                            |   |                              |   |                                                |
| $\vee\!\vee$            | 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., 01ssss001).

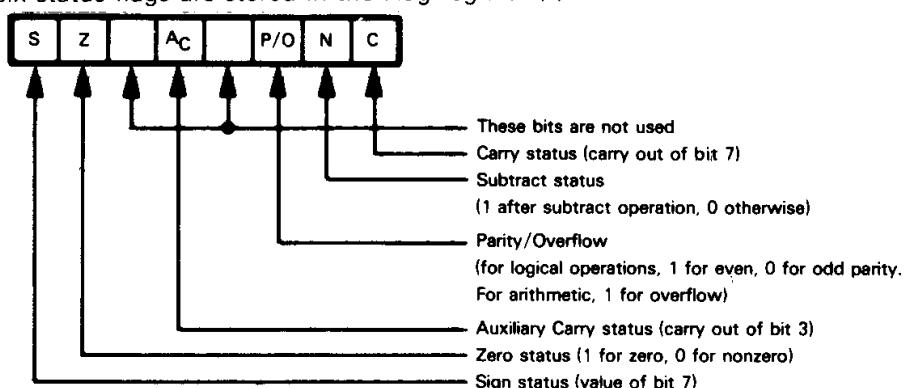
## INSTRUCTION EXECUTION TIMES

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.

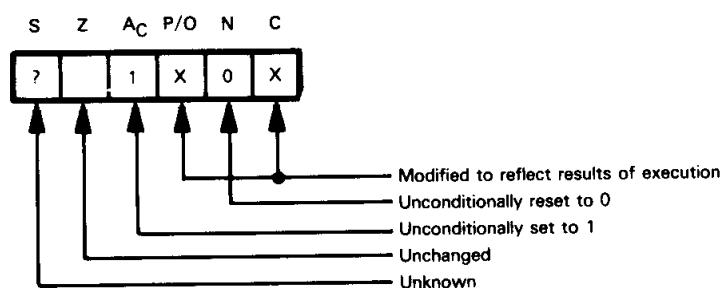
When two possible execution times are shown (i.e., 5/11), it indicates that the number of clock periods depends on condition flags. The first time is for "condition not met," whereas the second is for "condition met."

## STATUS

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



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



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.

**STATUS  
CHANGES  
WITH  
INSTRUCTION  
EXECUTION**

\*\* 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 | AC                                        | N                                                                                                                                                                                                                          |
|      | IN       | A, (port) | DB yy       | 2     | 10           |        |   |   |     | [ A ] ← [ port ]                          | Input to Accumulator from directly addressed I/O port.                                                                                                                                                                     |
|      |          | reg, (C)  | ED 01ddd000 | 2     | 11           | X      | X | P | X   | Address Bus: A0-A7: port<br>A8-A15: [ A ] |                                                                                                                                                                                                                            |
|      | INR      |           | ED B2       | 2     | 20/15**      | 1      | ? | ? | ?   | [ reg ] ← [ [ C ] ]                       | Input to register from I/O port addressed by the contents of C.**                                                                                                                                                          |
|      |          |           | ED BA       | 2     | 20/15**      | 1      | ? | ? | ?   | [ [ HL ] ] ← 0;                           | If second byte is 70 <sub>16</sub> , only the flags will be affected.                                                                                                                                                      |
|      | INDR     |           |             |       |              |        |   |   |     | [ [ HL ] ] ← [ [ C ] ]                    | Repeat until [ B ] = 0:                                                                                                                                                                                                    |
|      |          |           |             |       |              |        |   |   |     | [ 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.** |
|      | INI      |           |             |       |              |        |   |   |     | Repeat until [ B ] = 0:                   |                                                                                                                                                                                                                            |
|      |          |           |             |       |              |        |   |   |     | [ [ 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.**                                                 |
|      |          |           |             |       |              |        |   |   |     | [ B ] ← [ B ] - 1                         |                                                                                                                                                                                                                            |
|      |          |           |             |       |              |        |   |   |     | [ [ HL ] ] ← [ [ HL ] ] + 1               |                                                                                                                                                                                                                            |

\*\*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 | AC | N                                                                                                                                                                                                                        |  |
|      | IND      |          | ED AA       | 2     | 15           | X      | ? | ? | ?   | 1  | [[HL]] $\leftarrow$ [[C]]<br>[B] $\leftarrow$ [B] - 1<br>[HL] $\leftarrow$ [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] $\leftarrow$ [A]                                                                                                                                                                                                  |  |
|      |          |          |             |       |              |        |   |   |     |    | Output from Accumulator to directly addressed I/O port.                                                                                                                                                                  |  |
|      | OUT      | (C),reg  | ED 01sss001 | 2     | 12           |        |   |   |     |    | Address Bus: A0-A7: port<br>A8-A15: [A]                                                                                                                                                                                  |  |
|      | OTIR     |          | ED B3       | 2     | 20/15**      | 1      | ? | ? | ?   | 1  | [[C]] $\leftarrow$ [reg]                                                                                                                                                                                                 |  |
|      |          |          |             |       |              |        |   |   |     |    | Output from register to I/O port addressed by the contents of C..**                                                                                                                                                      |  |
|      |          |          |             |       |              |        |   |   |     |    | Repeat until [B] = 0:                                                                                                                                                                                                    |  |
|      |          |          |             |       |              |        |   |   |     |    | [[C]] $\leftarrow$ [[HL]]<br>[B] $\leftarrow$ [B] - 1<br>[HL] $\leftarrow$ [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:<br>[[C]] $\leftarrow$ [[HL]]<br>[B] $\leftarrow$ [B] - 1<br>[HL] $\leftarrow$ [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..** |  |
|      |          |          |             |       |              |        |   |   |     |    | I/O (Continued)                                                                                                                                                                                                          |  |

\*\*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 | C | Z | S | P/O | A <sub>C</sub> | N | Operation Performed                                                                                                                                                    |
|--------------------------|----------|-----------|------------------|-------|--------------|---|---|---|-----|----------------|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                          | OUTI     |           | ED A3            | 2     | 15           | X | ? | ? | ?   | 1              | 1 | $[(C)] \leftarrow [(HL)]$<br>$[B] \leftarrow [B] - 1$<br>$[HL] \leftarrow [HL] + 1$                                                                                    |
|                          | OUTD     |           | ED AB            | 2     | 15           | X | ? | ? | ?   | 1              | 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. ** |
|                          |          |           |                  |       |              |   |   |   |     |                |   | $[(C)] \leftarrow [(HL)]$<br>$[B] \leftarrow [B] - 1$<br>$[HL] \leftarrow [HL] - 1$                                                                                    |
|                          |          |           |                  |       |              |   |   |   |     |                |   | 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. **      |
|                          | LD       | A,(addr)  | 3A ppqq          | 3     | 13           |   |   |   |     |                |   | $[A] \leftarrow [addr]$                                                                                                                                                |
|                          | LD       | HL,(addr) | 2A ppqq          | 3     | 16           |   |   |   |     |                |   | Load Accumulator from directly addressed memory location.                                                                                                              |
|                          | LD       | rp,(addr) | ED 01xx0111 ppqq | 4     | 20           |   |   |   |     |                |   | Load HL from directly addressed memory.                                                                                                                                |
|                          | LD       | xy,(addr) | 11x11101 2A ppqq | 4     | 20           |   |   |   |     |                |   | $[rp(H)] \leftarrow [addr + 1]$ , $[rp(L)] \leftarrow [addr]$ or<br>$[xy(H)] \leftarrow [addr + 1]$ , $[xy(L)] \leftarrow [addr]$                                      |
|                          | LD       | (addr),A  | 32 ppqq          | 3     | 13           |   |   |   |     |                |   | Load register pair or Index register from directly addressed memory.                                                                                                   |
|                          | LD       | (addr),HL | 22 ppqq          | 3     | 16           |   |   |   |     |                |   | $[addr] \leftarrow [A]$                                                                                                                                                |
|                          | LD       | (addr),rp | ED 01xx0011 ppqq | 4     | 20           |   |   |   |     |                |   | Store Accumulator contents in directly addressed memory location.                                                                                                      |
|                          | LD       | (addr),xy | 11x11101 22 ppqq | 4     | 20           |   |   |   |     |                |   | $[addr + 1] \leftarrow [H]$ , $[addr] \leftarrow [L]$                                                                                                                  |
|                          | LD       | A,(BC)    | 0A               | 1     | 7            |   |   |   |     |                |   | Store contents of HL to directly addressed memory location.                                                                                                            |
|                          |          |           | 1A               | 1     | 7            |   |   |   |     |                |   | $[addr + 1] \leftarrow [rp(H)]$ , $[addr] \leftarrow [rp(L)]$ or<br>$[addr + 1] \leftarrow [xy(H)]$ , $[addr] \leftarrow [xy(L)]$                                      |
|                          |          |           |                  |       |              |   |   |   |     |                |   | Store contents of register pair or Index register to directly addressed memory.                                                                                        |
|                          |          |           |                  |       |              |   |   |   |     |                |   | $[A] \leftarrow [(BC)]$ or $[A] \leftarrow [(DE)]$                                                                                                                     |
|                          |          |           |                  |       |              |   |   |   |     |                |   | Load Accumulator from memory location addressed by the contents of the specified register pair.                                                                        |
| Primary Memory Reference |          |           |                  |       |              |   |   |   |     |                |   |                                                                                                                                                                        |

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/I | A <sub>C</sub>      | N                                                                                                                                                                                                                                       |
|      | LD       | reg,(HL)         | 01ddd110                    | 1      | 7            |        |   |   |     |                     | [reg] $\leftarrow$ [HL]                                                                                                                                                                                                                 |
|      | LD       | (BC),A<br>(DE),A | 02<br>12                    | 1<br>1 | 7<br>7       |        |   |   |     |                     | [BC] $\leftarrow$ [A] or [DE] $\leftarrow$ [A]                                                                                                                                                                                          |
|      | LD       | (HL),reg         | 01110ssss                   | 1      | 7            |        |   |   |     |                     | Store Accumulator to memory location addressed by the contents of the specified register pair.                                                                                                                                          |
|      | LD       | reg,(xy+disp)    | 111x111011 01ddd110<br>disp | 3      | 19           |        |   |   |     |                     | [HL] $\leftarrow$ [reg]                                                                                                                                                                                                                 |
|      | LD       | (xy+disp),reg    | 111x111011 01110sss<br>disp | 3      | 19           |        |   |   |     |                     | Store register contents to memory location addressed by the contents of HL.                                                                                                                                                             |
|      | LD       |                  |                             |        |              |        |   |   |     |                     | [reg] $\leftarrow$ [xy + disp]                                                                                                                                                                                                          |
|      | LD       |                  |                             |        |              |        |   |   |     |                     | Load register from memory location using base relative addressing.                                                                                                                                                                      |
|      | LD       |                  |                             |        |              |        |   |   |     |                     | [xy] $\leftarrow$ [reg]                                                                                                                                                                                                                 |
|      | LDIR     |                  | ED B0                       | 2      | 20/16**      |        |   |   |     |                     | Store register to memory location addressed relative to contents of Index register.                                                                                                                                                     |
|      | LDDR     |                  |                             |        |              |        |   |   |     |                     | Repeat until [BC] = 0:<br>[DE] $\leftarrow$ [HL]<br>[DE] $\leftarrow$ [DE] + 1<br>[HL] $\leftarrow$ [HL] + 1<br>[BC] $\leftarrow$ [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. |
|      |          |                  |                             |        |              |        |   |   |     |                     | Repeat until [BC] = 0:<br>[DE] $\leftarrow$ [HL]<br>[DE] $\leftarrow$ [DE] - 1<br>[HL] $\leftarrow$ [HL] - 1<br>[BC] $\leftarrow$ [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. |

Primary Memory Reference  
(Continued)

Block Transfer and Search

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

| Type                                  | Mnemonic | Operand | Object Code | Bytes | Clock Cycles | C | Z | S | P/O | AC | N | Operation Performed                                                                                                                                                                                             |  |
|---------------------------------------|----------|---------|-------------|-------|--------------|---|---|---|-----|----|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
|                                       |          |         |             |       |              |   |   |   |     |    |   | Status                                                                                                                                                                                                          |  |
|                                       | LDI      |         | ED A0       | 2     | 16           |   |   | x | 0   | 0  | 0 | $[(DE)] \leftarrow [(HL)]$                                                                                                                                                                                      |  |
|                                       | LDD      |         |             |       |              |   |   |   |     |    |   | $[DE] \leftarrow [DE] + 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[HL] \leftarrow [HL] + 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[BC] \leftarrow [BC] - 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | 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. |  |
|                                       | ED A8    |         |             | 2     | 16           |   | x | 0 | 0   | 0  | 0 | $[(DE)] \leftarrow [(HL)]$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[DE] \leftarrow [DE] - 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[HL] \leftarrow [HL] - 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[BC] \leftarrow [BC] - 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | 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.           |  |
|                                       | ED B1    |         |             | 2     | 20/16**      | x | x | x | x   | 1  | 1 | Repeat until $[A] = [HL]$ or $[BC] = 0$ :                                                                                                                                                                       |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[A] - [HL]$ (only flags are affected)                                                                                                                                                                          |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[HL] \leftarrow [HL] + 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[BC] \leftarrow [BC] - 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | 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.                       |  |
|                                       | ED B9    |         |             | 2     | 20/16**      | x | x | x | x   | 1  | 1 | Repeat until $[A] = [HL]$ or $[BC] = 0$ :                                                                                                                                                                       |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[A] - [HL]$ (only flags are affected)                                                                                                                                                                          |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[HL] \leftarrow [HL] - 1$                                                                                                                                                                                      |  |
|                                       |          |         |             |       |              |   |   |   |     |    |   | $[BC] \leftarrow [BC] - 1$                                                                                                                                                                                      |  |
|                                       | CPDR     |         |             |       |              |   |   |   |     |    |   | 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.                       |  |
| Block Transfer and Search (Continued) |          |         |             |       |              |   |   |   |     |    |   |                                                                                                                                                                                                                 |  |

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 <sub>C</sub> | N                   |                                                                                                                                        |  |
|      | CPI      |                       | ED A1                 | 2      | 16           | x      | x | x | x   | x              | x                   | [A] - [HL] (only flags are affected)<br>[HL] - [HL] + 1<br>[BC] - [BC] - 1                                                             |  |
|      | CPD      |                       | ED A9                 | 2      | 16           | x      | x | x | x   | x              | x                   | Compare contents of Accumulator with those of memory location addressed by contents of HL. Increment address and decrement byte count. |  |
|      | ADD      | A,(HL)<br>A,(xy+disp) | 86<br>11x1101 86 disp | 1<br>3 | 7<br>19      | x      | x | x | x   | x              | x                   | [A] - [A] + [HL] or [A] - [A] + [xy] + disp<br>Add to Accumulator using implied addressing or base relative addressing.                |  |
|      | ADC      | A,(HL)<br>A,(xy+disp) | 8E<br>11x1101 8E disp | 1<br>3 | 7<br>19      | x      | x | x | x   | x              | x                   | [A] - [A] + [HL] + C or [A] - [A] - [xy] + disp] + C<br>Add with Carry using implied addressing or base relative addressing.           |  |
|      | SUB      | (HL)<br>(xy+disp)     | 96<br>11x1101 96 disp | 1<br>3 | 7<br>19      | x      | x | x | x   | x              | x                   | [A] - [A] - [HL] or [A] - [A] - [xy] + disp<br>Subtract from Accumulator using implied addressing or base relative addressing.         |  |
|      | SBC      | A,(HL)<br>A,(xy+disp) | 9E<br>11x1101 9E disp | 1<br>3 | 7<br>19      | x      | x | x | x   | x              | x                   | [A] - [A] - [HL] - C or [A] - [A] - [xy] + disp] - C<br>Subtract with Carry using implied addressing or base relative addressing.      |  |
|      | AND      | (HL)<br>(xy+disp)     | A6<br>11x1101 A6 disp | 1<br>3 | 7<br>19      | 0      | x | x | P   | 1              | 0                   | [A] - [A] A [HL] or [A] - [A] A [xy] + disp<br>AND with Accumulator using implied addressing or base relative addressing.              |  |
|      | OR       | (HL)<br>(xy+disp)     | B6<br>11x1101 B6 disp | 1<br>3 | 7<br>19      | 0      | x | x | x   | 1              | 0                   | [A] - [A] V [HL] or [A] - [A] V [xy] + disp<br>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 <sub>C</sub> | N                   | [A] ← [A] $\vee$ [[HL]] or [A] ← [A] $\vee$ [[xy] + disp]                      | Exclusive-OR with Accumulator using implied addressing or base relative addressing.                         |  |
|                         | XOR      | (HL)<br>(xy + disp) | AE<br>11x11101 AE disp          | 1<br>3 | 7<br>19      | 0<br>* | x<br>x | p<br>0 | 1<br>x | 0<br>x         | 0<br>x              | [A] - [[HL]] or [A] - [[xy] + disp]                                            | Compare with Accumulator using implied addressing or base relative addressing. Only the flags are affected. |  |
|                         | CP       | (HL)<br>(xy + disp) | BE<br>11x11101 BE disp          | 1<br>3 | 7<br>19      | x<br>x | x<br>x | 0<br>x | 0<br>x | 1<br>x         | 1<br>x              | [A] - [[HL]] or [A] - [[xy] + disp]                                            | Increment using implied addressing or base relative addressing.                                             |  |
|                         | INC      | (HL)<br>(xy + disp) | 34<br>11x11101 34 disp          | 1<br>3 | 11<br>23     | x<br>x | x<br>x | 0<br>0 | x<br>x | 0<br>x         | 0<br>x              | [[HL]] $\leftarrow$ [[HL]] + 1 or [[xy] + disp] $\leftarrow$ [[xy] + disp] + 1 | Decrement using implied addressing or base relative addressing.                                             |  |
|                         | DEC      | (HL)<br>(xy + disp) | 35<br>11x11101 35 disp          | 1<br>3 | 11<br>23     | x<br>x | x<br>x | 0<br>0 | x<br>x | 1<br>x         | 1<br>x              | [[HL]] $\leftarrow$ [[HL]] - 1 or [[xy] + disp] $\leftarrow$ [[xy] + disp] - 1 | Decrement using implied addressing or base relative addressing.                                             |  |
| Memory Shift and Rotate |          |                     |                                 |        |              |        |        |        |        |                |                     |                                                                                |                                                                                                             |  |
|                         | RLC      | (HL)<br>(xy + disp) | CB 06<br>11x11101 CB disp<br>06 | 2<br>4 | 15<br>23     | x<br>x | x<br>x | p<br>0 | 0<br>0 | 0<br>0         | 0<br>0              | [[HL]] or [[xy] + disp]                                                        | Rotate contents of memory location (implied or base relative addressing) left with branch Carry.            |  |
|                         | RL       | (HL)<br>(xy + disp) | CB 16<br>11x11101 CB disp<br>16 | 2<br>4 | 15<br>23     | x<br>x | x<br>x | p<br>0 | 0<br>0 | 0<br>0         | 0<br>0              | [[HL]] or [[xy] + disp]                                                        | Rotate contents of memory location left through Carry.                                                      |  |
|                         | RRC      | (HL)<br>(xy + disp) | CB 0E<br>11x11101 CB disp<br>0E | 2<br>4 | 15<br>23     | x<br>x | x<br>x | p<br>0 | 0<br>0 | 0<br>0         | 0<br>0              | [[HL]] or [[xy] + disp]                                                        | Rotate contents of memory location 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 | AC | N                   |                                                                                                           |
|      | RR       | (HL)<br>(xy + disp) | CB 1E<br>11x11101 CB disp<br>1E | 2<br>4 | 15<br>23     | x      | x | x | P   | 0  | 0                   | [[ HL]] or [[ xy] + disp]<br>Rotate contents of memory location right through Carry                       |
|      | SLA      | (HL)<br>(xy + disp) | CB 26<br>11x11101 CB disp<br>26 | 2<br>4 | 15<br>23     | x      | x | x | P   | 0  | 0                   | [[ HL]] or [[ xy] + disp]<br>Shift contents of memory location left and clear LSB (Arithmetic Shift).     |
|      | SRA      | (HL)<br>(xy + disp) | CB 2E<br>11x11101 CB disp<br>2E | 2<br>4 | 15<br>23     | x      | x | x | P   | 0  | 0                   | [[ HL]] or [[ xy] + disp]<br>Shift contents of memory location right and preserve MSB (Arithmetic Shift). |
|      | SRL      | (HL)<br>(xy + disp) | CB 3E<br>11x11101 CB disp<br>3E | 2<br>4 | 15<br>23     | x      | x | x | P   | 0  | 0                   | [[ HL]] or [[ xy] + disp]<br>Shift contents of memory location right and clear MSB (Logical Shift)        |

Memory Shift and Rotate (Continued)

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 | AC                                     | N                                                                                  |
|                            | LD       | reg,data       | 00ddd110 yy        | 2     | 7            |        |   |   |     | [reg] ← data                           | Load immediate into register.                                                      |
|                            | LD       | rp,data16      | 00xx0001 yyyy      | 3     | 10           |        |   |   |     | [rp] ← data[16 or [xy] ← data16        | Load 16 bits of immediate data into register pair or Index register.               |
| Immediate                  | LD       | xy,data16      | 11x1110121 yyyy    | 4     | 14           |        |   |   |     | [ [HL] ] ← data or [xy] + disp] ← data | Load immediate into memory location using implied or base relative addressing.     |
|                            | LD       | (HL),data      | 36 yy              | 2     | 10           |        |   |   |     |                                        |                                                                                    |
|                            | LD       | (xy+disp),data | 11x1110136 disp yy | 4     | 19           |        |   |   |     |                                        |                                                                                    |
|                            | JP       | label          | C3 ppqq            | 3     | 10           |        |   |   |     | [PC] ← label                           |                                                                                    |
|                            | JR       | disp           | 18 (disp-2)        | 2     | 12           |        |   |   |     | [PC] ← [PC] + 2 + (disp-2)             | Jump to instruction at address represented by label.                               |
|                            | JP       | (HL)           | E9                 | 1     | 4            |        |   |   |     | [PC] ← [HL] or [PC] ← [xy]             | Jump relative to present contents of Program Counter.                              |
|                            | JP       | (xy)           | 11x11101 E9        | 2     | 8            |        |   |   |     |                                        | Jump to address contained in HL or Index register.                                 |
|                            | CALL     | label          | CD ppqq            | 3     | 17           |        |   |   |     |                                        |                                                                                    |
|                            | CALL     | cond,label     | 11ccc100 ppqq      | 3     | 10/17        |        |   |   |     |                                        |                                                                                    |
|                            | RET      |                | C9                 | 1     | 10           |        |   |   |     | [SP] - 1 ← [PC(HI)]                    |                                                                                    |
|                            | RET      | cond           | 11cc0000           | 1     | 5/11         |        |   |   |     | [SP] - 2 ← [PC(LO)]                    |                                                                                    |
|                            |          |                |                    |       |              |        |   |   |     | [SP] ← [SP] - 2                        |                                                                                    |
|                            |          |                |                    |       |              |        |   |   |     | [PC] ← label                           | Jump to subroutine starting at address represented by label.                       |
|                            |          |                |                    |       |              |        |   |   |     |                                        | Jump to subroutine if condition is satisfied; otherwise, continue in sequence.     |
|                            |          |                |                    |       |              |        |   |   |     | [PC(LO)] ← [[SP]]                      |                                                                                    |
|                            |          |                |                    |       |              |        |   |   |     | [PC(HI)] ← [[SP] + 1]                  |                                                                                    |
|                            |          |                |                    |       |              |        |   |   |     | [SP] ← [SP] + 2                        |                                                                                    |
|                            |          |                |                    |       |              |        |   |   |     |                                        | Return from subroutine.                                                            |
|                            |          |                |                    |       |              |        |   |   |     |                                        | Return from subroutine if condition is satisfied; otherwise, continue in sequence. |
| Subroutine Call and Return |          |                |                    |       |              |        |   |   |     |                                        |                                                                                    |

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 | AC | N                   |                                                                                              |
| Immediate Operate |          |            |                |       |              |        |   |   |     |    |                     |                                                                                              |
|                   | ADD      | A,data     | C6 yy          | 2     | 7            | X      | X | X | 0   | X  | 0                   | [A] $\leftarrow$ [A] + data                                                                  |
|                   | ADC      | A,data     | CE yy          | 2     | 7            | X      | X | X | 0   | X  | 0                   | Add immediate to Accumulator.                                                                |
|                   | SUB      | data       | D6 yy          | 2     | 7            | X      | X | X | 0   | X  | 1                   | [A] $\leftarrow$ [A] + data + C                                                              |
|                   | SBC      | A,data     | DE yy          | 2     | 7            | X      | X | X | 0   | X  | 1                   | Add immediate with Carry.                                                                    |
|                   | AND      | data       | E6 yy          | 2     | 7            | 0      | X | X | P   | 1  | 0                   | [A] $\leftarrow$ [A] - data                                                                  |
|                   | OR       | data       | F6 yy          | 2     | 7            | 0      | X | X | P   | 1  | 0                   | [A] $\leftarrow$ [A] V data                                                                  |
|                   | XOR      | data       | EE yy          | 2     | 7            | 0      | X | X | P   | 1  | 0                   | [A] $\leftarrow$ [A] OR immediate with Accumulator.                                          |
|                   | CP       | data       | FE yy          | 2     | 7            | X      | X | X | 0   | X  | 1                   | [A] $\leftarrow$ [A] $\Delta$ data                                                           |
|                   |          |            |                |       |              |        |   |   |     |    |                     | Exclusive-OR immediate with Accumulator.                                                     |
|                   |          |            |                |       |              |        |   |   |     |    |                     | Compare immediate data with Accumulator contents; only the flags are affected.               |
| Jump on Condition |          |            |                |       |              |        |   |   |     |    |                     |                                                                                              |
|                   | JP       | cond,label | 11cccc010 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)                                       |
|                   | JR       | NC,disp    | 30 (disp-2)    | 2     | 7/12         |        |   |   |     |    |                     | Jump relative to contents of Program Counter if Carry flag is set.                           |
|                   | JR       | Z,disp     | 28 (disp-2)    | 2     | 7/12         |        |   |   |     |    |                     | If C = 0, then [PC] $\leftarrow$ [PC] + 2 + (disp - 2)                                       |
|                   | JR       | NZ,disp    | 20 (disp-2)    | 2     | 7/12         |        |   |   |     |    |                     | Jump relative to contents of Program Counter if Carry flag is reset.                         |
|                   | DJNZ     | disp       | 10 (disp-2)    | 2     | 8/13         |        |   |   |     |    |                     | If Z = 1, then [PC] $\leftarrow$ [PC] + 2 + (disp - 2)                                       |
|                   |          |            |                |       |              |        |   |   |     |    |                     | Jump relative to contents of Program Counter if Zero flag is set.                            |
|                   |          |            |                |       |              |        |   |   |     |    |                     | If Z = 0, then [PC] $\leftarrow$ [PC] + 2 + (disp - 2)                                       |
|                   |          |            |                |       |              |        |   |   |     |    |                     | Jump relative to contents of Program Counter if Zero flag is reset.                          |
|                   |          |            |                |       |              |        |   |   |     |    |                     | [B] $\leftarrow$ [B] - 1                                                                     |
|                   |          |            |                |       |              |        |   |   |     |    |                     | If [B] $\neq$ 0, then [PC] $\leftarrow$ 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 <sub>C</sub> | N                   |                                                                                           |                                                                                                                                |
| LD                     | LD       | dst,src | 01ddssss    | 1     | 4            |        |   |   |     |                |                     | [dst] $\leftarrow$ [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,l     | ED 57       | 2     | 9            | x      | x | - | 0   | 0              | 0                   | [A] $\leftarrow$ [l]                                                                      | Move contents of interrupt Vector register to Accumulator.                                                                     |
| LD                     |          | A,R     | ED 5F       | 2     | 9            | x      | x | - | 0   | 0              | 0                   | [A] $\leftarrow$ [R]                                                                      | Move contents of Refresh register to Accumulator.                                                                              |
| LD                     |          | I,A     | ED 47       | 2     | 9            |        |   |   |     |                |                     | [I] $\leftarrow$ [A]                                                                      | Load Interrupt Vector register from Accumulator.                                                                               |
| LD                     |          | R,A     | ED 4F       | 2     | 9            |        |   |   |     |                |                     | [R] $\leftarrow$ [A]                                                                      | Load Refresh register from Accumulator.                                                                                        |
| LD                     |          | SP,HL   | F9          | 1     | 6            |        |   |   |     |                |                     | [SP] $\leftarrow$ [HL]                                                                    | Move contents of HL to Stack Pointer.                                                                                          |
| LD                     |          | SP,xy   | 11x11101 F9 | 2     | 10           |        |   |   |     |                |                     | [SP] $\leftarrow$ [xy]                                                                    | Move contents of Index register to Stack Pointer.                                                                              |
| LD                     |          | DE,HL   | EB          | 1     | 4            |        |   |   |     |                |                     | [DE] $\leftarrow$ [HL]                                                                    | Exchange contents of DE and HL.                                                                                                |
| EX                     |          | AF,AF'  | 08          | 1     | 4            |        |   |   |     |                |                     | [AF] $\leftarrow$ [AF']                                                                   | Exchange program status and alternate program status.                                                                          |
| EX                     |          |         | D9          | 1     | 4            |        |   |   |     |                |                     | $(BC)$ $\leftarrow$ $(BC')$<br>$(DE)$ $\leftarrow$ $(DE')$<br>$(HL)$ $\leftarrow$ $(HL')$ | Exchange register pairs and alternate register pairs.                                                                          |
| Register-Register Move |          |         |             |       |              |        |   |   |     |                |                     |                                                                                           |                                                                                                                                |

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                 | AC | N |                                                                                              |
|      | ADD      | A,reg   | 10000rrrr   | 1     | 4            | x      | x | 0 | x                   | 0  | 0 | [A] - [A] + [reg]                                                                            |
|      | ADC      | A,reg   | 10001rrrr   | 1     | 4            | x      | x | 0 | x                   | 0  | 0 | Add contents of register to Accumulator.<br>[A] - [A] + [reg] + C                            |
|      | SUB      | reg     | 10010rrrr   | 1     | 4            | x      | x | 0 | x                   | 1  | 0 | Add contents of register and Carry to Accumulator.<br>[A] - [A] - [reg]                      |
|      | SBC      | A,reg   | 10011rrrr   | 1     | 4            | x      | x | 0 | x                   | 1  | 1 | Subtract contents of register from Accumulator.<br>[A] - [A] - [reg] - C                     |
|      | AND      | reg     | 10000rrrr   | 1     | 4            | 0      | x | x | p                   | 1  | 0 | Subtract contents of register and Carry from Accumulator.<br>[A] - [A] ^ [reg]               |
|      | OR       | reg     | 10110rrrr   | 1     | 4            | 0      | x | x | p                   | 1  | 0 | AND contents of register with contents of Accumulator.<br>[A] - [A] V [reg]                  |
|      | XOR      | reg     | 10101rrrr   | 1     | 4            | 0      | x | x | p                   | 1  | 0 | OR contents of register with contents of Accumulator.<br>[A] - [A] $\Delta$ [reg]            |
|      | CP       | reg     | 10111rrrr   | 1     | 4            | x      | x | 0 | x                   | 1  | 1 | Exclusive-OR contents of register with contents of Accumulator.<br>[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]                                                                           |
|      | ADC      | HL,RP   | ED 01xx1010 | 2     | 15           | x      | x | 0 | x                   | ?  | 0 | 16-bit add register pair contents to contents of HL.                                         |
|      | SBC      | HL,RP   | ED 01xx0010 | 2     | 15           | x      | x | x | 0                   | ?  | 1 | 16-bit add with Carry register pair contents to contents of HL.                              |
|      | ADD      | IX,PP   | DD 00xx1001 | 2     | 15           | x      |   |   |                     | ?  | 0 | [HL] - [HL] - [rp] - C                                                                       |
|      | ADD      | IY,RR   | FD 00xx1001 | 2     | 15           | x      |   |   |                     | ?  | 0 | 16-bit subtract with Carry register pair contents from contents of HL.<br>[IX] - [IX] + [pp] |
|      |          |         |             |       |              |        |   |   |                     |    |   | 16-bit add register pair contents to contents of Index register IX<br>(pp = BC, DE, IX, SP). |
|      |          |         |             |       |              |        |   |   |                     |    |   | [IY] - [IY] + [rr]                                                                           |
|      |          |         |             |       |              |        |   |   |                     |    |   | 16-bit add register pair contents to contents of Index register IY<br>(rr = BC, DE, IY, SP). |

Register-Register Operate

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

| Type                      | Mnemonic | Operand | Object Code | Bytes | Clock Cycles | C | Z | S | P/O | AC | N | Status                                                                                                    | Operation Performed |
|---------------------------|----------|---------|-------------|-------|--------------|---|---|---|-----|----|---|-----------------------------------------------------------------------------------------------------------|---------------------|
| Register Shift and Rotate |          |         |             |       |              |   |   |   |     |    |   |                                                                                                           |                     |
|                           | 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            | x | x | x | p   | x  | 1 | $[A] \leftarrow [\bar{A}]$                                                                                |                     |
|                           | NEG      |         | ED 44       | 2     | 8            | x | x | x | 0   | x  | 1 | Complement Accumulator (ones complement).                                                                 |                     |
|                           | INC      | reg     | 00rrr100    | 1     | 4            | x | x | x | 0   | x  | 0 | Negate Accumulator (twos complement).                                                                     |                     |
|                           | INC      | rp      | 00xx0011    | 1     | 6            |   |   |   |     |    |   | $[reg] \leftarrow [reg] + 1$                                                                              |                     |
|                           | INC      | xy      | 11x11101 23 | 2     | 10           |   |   |   |     |    |   | Increment register contents.                                                                              |                     |
|                           | DEC      | reg     | 00rrr101    | 1     | 4            | x | x | x | 0   | x  | 1 | $[rp] \leftarrow [rp] + 1$ or $[xy] \leftarrow [xy] + 1$                                                  |                     |
|                           | DEC      | rp      | 00xx1011    | 1     | 6            |   |   |   |     |    |   | Decrement contents of register or Index register.                                                         |                     |
|                           | DEC      | xy      | 11x11101 2B | 2     | 10           |   |   |   |     |    |   | $[reg] \leftarrow [reg] - 1$                                                                              |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | Decrement contents of register pair or Index register.                                                    |                     |
|                           | RLCA     |         | 07          | 1     | 4            | x |   |   |     |    |   |                                                                                                           |                     |
|                           | RLA      |         | 17          | 1     | 4            | x |   |   |     |    |   |                                                                                                           |                     |
|                           | RRCA     |         | 0F          | 1     | 4            | x |   |   |     |    |   |                                                                                                           |                     |
| Rotate                    |          |         |             |       |              |   |   |   |     |    |   |                                                                                                           |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | Rotate Accumulator left with branch Carry.                                                                |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | $[A]$                                                                                                     |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | Rotate Accumulator left through Carry.                                                                    |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | $[A]$                                                                                                     |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | Rotate Accumulator right with branch Carry.                                                               |                     |
|                           |          |         |             |       |              |   |   |   |     |    |   | $[A]$                                                                                                     |                     |

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 AC N |                                                                            |
|      | RRA      |         | 1F          | 1     | 4            | X              |                                                                            |
|      | RLC      | reg     | CB 000000rr | 2     | 8            | X X X P O O    | Rotate Accumulator right through Carry.<br>[A]                             |
|      | RL       | reg     | CB 00010rrr | 2     | 8            | X X X P O O    | Rotate contents of register left with branch Carry.<br>[reg]               |
|      | RRC      | reg     | CB 00001rrr | 2     | 8            | X X X P O O    | Rotate contents of register left through Carry.<br>[reg]                   |
|      | RR       | reg     | CB 00011rrr | 2     | 8            | X X X P O O    | Rotate contents of register right with branch Carry.<br>[reg]              |
|      | SLA      | reg     | CB 00100rrr | 2     | 8            | X X X P O O    | Rotate contents of register right through Carry.<br>[reg]                  |
|      |          |         |             |       |              |                | Shift contents of register left and clear LSB (Arithmetic Shift).<br>[reg] |

Register Shift and Rotate (Continued)

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

| Type | Mnemonic | Operand | Object Code | Bytes | Clock Cycles | Status    | Operation Performed                                                                                                                                          |
|------|----------|---------|-------------|-------|--------------|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      |          |         |             |       |              |           |                                                                                                                                                              |
|      | SRA      | reg     | CB 00101rrr | 2     | 8            | x x x p o | Shift contents of register right and preserve MSB (Arithmetic Shift).                                                                                        |
|      | SRL      | reg     | CB 00111rrr | 2     | 8            | x x x p o | Shift contents of register right and clear MSB (Logical Shift).                                                                                              |
|      | RLD      |         | ED 6F       | 2     | 18           | x x p o   | 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 o   | Rotate one BCD digit right between the Accumulator and memory location (implied addressing). Contents of the upper half of the Accumulator are not affected. |

Register Shift and Rotate (Continued)

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 <sub>C</sub> | N                                                                                                                      |                     |
| Bit Manipulation |               |         |                              |       |              |        |   |   |     |                |                                                                                                                        |                     |
| BIT              | b,reg         |         | CB 01bbbrrr                  | 2     | 8            | x      | ? | ? | 1   | 0              | z ← $\bar{reg}[b]$                                                                                                     |                     |
| BIT              | b,(HL)        |         | CB 01bbb110                  | 2     | 12           | x      | ? | ? | 1   | 0              | Zero flag contains complement of the selected register bit.                                                            |                     |
|                  | b,(xy + disp) |         | 11x11101 CB disp<br>01bbb110 | 4     | 20           |        |   |   |     |                | $z \leftarrow [HL][b]$ or $z \leftarrow [xy] + disp[b]$                                                                |                     |
| SET              | b,reg         |         | CB 11bbbrrr                  | 2     | 8            |        |   |   |     |                | Zero flag contains complement of selected bit of the memory location (implied addressing or base relative addressing). |                     |
| SET              | b,(HL)        |         | CB 11bbb110                  | 2     | 15           |        |   |   |     |                | reg[b] ← 1                                                                                                             |                     |
|                  | b,(xy + disp) |         | 11x11101 CB disp<br>11bbb110 | 4     | 23           |        |   |   |     |                | Set indicated register bit.                                                                                            |                     |
| RES              | b,reg         |         | CB 10bbbrrr                  | 2     | 8            |        |   |   |     |                | [HL][b] ← 1 or [xy] + disp[b] ← 1                                                                                      |                     |
| RES              | b,(HL)        |         | CB 10bbb110                  | 2     | 15           |        |   |   |     |                | Set indicated bit of memory location (implied addressing or base relative addressing).                                 |                     |
|                  | b,(xy + disp) |         | 11x11101 CB disp<br>10bbb110 | 4     | 23           |        |   |   |     |                | reg[b] ← 0                                                                                                             |                     |
|                  |               |         |                              |       |              |        |   |   |     |                | Reset indicated register bit.                                                                                          |                     |
|                  |               |         |                              |       |              |        |   |   |     |                | $[HL][b] \leftarrow 0$ or $[xy] + disp[b] \leftarrow 0$                                                                |                     |
|                  |               |         |                              |       |              |        |   |   |     |                | Reset indicated bit in memory location (implied addressing or base relative addressing).                               |                     |
| Stack            |               |         |                              |       |              |        |   |   |     |                |                                                                                                                        |                     |
| PUSH             | pr            |         | 11xx0101                     | 1     | 11           |        |   |   |     |                | [SP]-1) ← [pr(H)]                                                                                                      |                     |
|                  | xy            |         | 11x11101 E5                  | 2     | 15           |        |   |   |     |                | [SP]-2) ← [pr(LO)]                                                                                                     |                     |
|                  |               |         |                              |       |              |        |   |   |     |                | [SP] ← [SP]-2                                                                                                          |                     |
| POP              | pr            |         | 11xx0001                     | 1     | 10           |        |   |   |     |                | Put contents of register pair or Index register on top of Stack and decrement Stack Pointer.                           |                     |
|                  | xy            |         | 11x11101 E1                  | 2     | 14           |        |   |   |     |                | [pr(LO)] ← [SP]]                                                                                                       |                     |
|                  |               |         |                              |       |              |        |   |   |     |                | [pr(H)] ← [SP] + 1                                                                                                     |                     |
| EX               | (SP),HL       |         | E3                           | 1     | 19           |        |   |   |     |                | [SP] ← [SP] + 2                                                                                                        |                     |
|                  | (SP),xy       |         | 11x11101 E3                  | 2     | 23           |        |   |   |     |                | Put contents of top of Stack in register pair or Index register and increment Stack Pointer.                           |                     |
|                  |               |         |                              |       |              |        |   |   |     |                | [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 <sub>C</sub>      | N |                                                                                                                                                                           |  |
|           | DI<br>EI<br>RST    | n           | F3<br>FB<br>11xx111                       | 1<br>1                | 4<br>4<br>11            |        |   |   |     |                     |   | Disable interrupts.<br>Enable interrupts.<br>[[ SP]-1] $\leftarrow$ [PC(H)]<br>[[ SP]-2] $\leftarrow$ [PC(L)]<br>[ SP] $\leftarrow$ [ SP]-2<br>[ PC] $\leftarrow$ (8•n)16 |  |
| Interrupt | RETI<br>RETN<br>IM | 0<br>1<br>2 |                                           |                       |                         |        |   |   |     |                     |   | Restart at designated location.<br>Return from interrupt.<br>Return from nonmaskable interrupt.<br>Set interrupt mode 0, 1, or 2.                                         |  |
|           | SCF<br>CCF         |             | ED 4D<br>ED 45<br>ED 46<br>ED 56<br>ED 5E | 2<br>2<br>2<br>2<br>2 | 14<br>14<br>8<br>8<br>8 |        |   |   |     |                     |   |                                                                                                                                                                           |  |
|           |                    |             | 37                                        | 1                     | 4                       | 1      |   |   |     | 0                   | 0 | C $\leftarrow$ 1<br>Set Carry flag.<br>C $\leftarrow$ $\bar{C}$<br>Complement Carry flag.                                                                                 |  |
| States    | NOP<br>HALT        |             | 00<br>76                                  | 1<br>1                | 4<br>4                  |        |   |   |     | ?                   | 0 | No operation — volatile memories are refreshed.<br>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  | OBJECT CODE | INSTRUCTION  |
|-------------|--------------|-------------|--------------|
| 00          | NOP          | 39          | ADD HL,SP    |
| 01 yyyy     | LD BC,data16 | 3A ppqq     | LD A,(addr)  |
| 02          | LD (BC),A    | 3B          | DEC SP       |
| 03          | INC BC       | 3C          | INC A        |
| 04          | INC B        | 3D          | DEC A        |
| 05          | DEC B        | 3E YY       | LD A,data    |
| 06 yy       | LD B,data    | 3F          | CCF          |
| 07          | RLCA         | 4 0sss      | LD B,reg     |
| 08          | EX AF,AF'    | 46          | LD B,(HL)    |
| 09          | ADD HL,BC    | 4 1sss      | LD C,reg     |
| 0A          | LD A,(BC)    | 4E          | LD C,(HL)    |
| 0B          | DEC BC       | 5 0sss      | LD D,reg     |
| 0C          | INC C        | 56          | LD D,(HL)    |
| 0D          | DEC C        | 5 1sss      | LD E,reg     |
| 0E yy       | LD C,data    | 5E          | LD E,(HL)    |
| 0F          | RRCA         | 6 0sss      | LD H,reg     |
| 10 disp-2   | DJNZ disp    | 66          | LD H,(HL)    |
| 11 yyyy     | LD DE,data16 | 6 1sss      | LD L,reg     |
| 12          | LD (DE),A    | 6E          | LD L,(HL)    |
| 13          | INC DE       | 7 0sss      | LD (HL),reg  |
| 14          | INC D        | 76          | HALT         |
| 15          | DEC D        | 7 1sss      | LD A,reg     |
| 16 yy       | LD D,data    | 7E          | LD A,(HL)    |
| 17          | RLA          | 8 0rrr      | ADD A,reg    |
| 18 disp-2   | JR disp      | 86          | ADD A,(HL)   |
| 19          | ADD HL,DE    | 8 1rrr      | ADC A,reg    |
| 1A          | LD A,(DE)    | 8E          | ADC A,(HL)   |
| 1B          | DEC DE       | 9 0rrr      | SUB reg      |
| 1C          | INC E        | 96          | SUB (HL)     |
| 1D          | DEC E        | 9 1rrr      | SBC A,reg    |
| 1E yy       | LD E,data    | 9E          | SBC A,(HL)   |
| 1F          | RRA          | A 0rrr      | AND reg      |
| 20 disp-2   | JR NZ,disp   | A6          | AND (HL)     |
| 21 yyyy     | LD HL,data16 | A 1rrr      | XOR reg      |
| 22 ppqq     | LD (addr),HL | AE          | XOR (HL)     |
| 23          | INC HL       | B 0rrr      | OR reg       |
| 24          | INC H        | B6          | OR (HL)      |
| 25          | DEC H        | B 1rrr      | CP reg       |
| 26 yy       | LD H,data    | BE          | CP (HL)      |
| 27          | DAA          | C0          | RET NZ       |
| 28 disp-2   | JR Z,disp    | C1          | POP BC       |
| 29          | ADD HL,HL    | C2 ppqq     | JP NZ,addr   |
| 2A ppqq     | LD HL,(addr) | C3 ppqq     | JP addr      |
| 2B          | DEC HL       | C4 ppqq     | CALL NZ,addr |
| 2C          | INC L        | C5          | PUSH BC      |
| 2D          | DEC L        | C6 YY       | ADD A,data   |
| 2E          | LD L,data    | C7          | RST 00H      |
| 2F          | CPL          | C8          | RET Z        |
| 30 disp-2   | JR NC,disp   | C9          | RET          |
| 31 yyyy     | LD SP,data16 | CA ppqq     | JP Z,addr    |
| 32 ppqq     | LD (addr),A  | CB 0 0rrr   | RLC reg      |
| 33          | INC SP       | CB 06       | RLC (HL)     |
| 34          | INC (HL)     | CB 0 1rrr   | RRRC reg     |
| 35          | DEC (HL)     | CB 0E       | RRRC (HL)    |
| 36 yy       | LD (HL),data | CB 1 0rrr   | RL reg       |
| 37          | SCF          | CB 16       | RL (HL)      |
| 38          | JR C,disp    | CB 1 1rrr   | RR reg       |

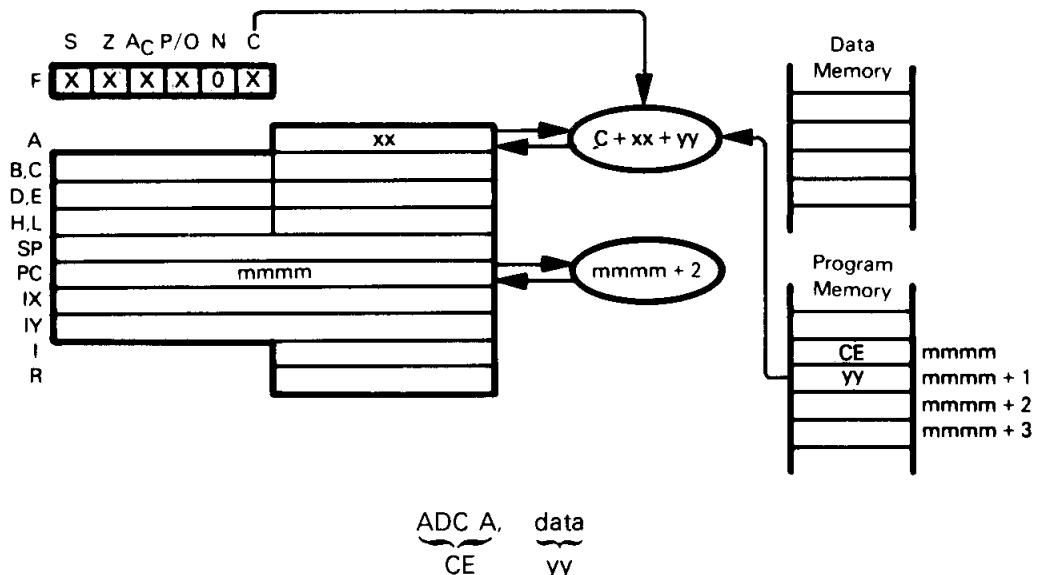
Table 3-5. Instruction Object Codes in Numerical Order (Continued)

| OBJECT CODE         | INSTRUCTION         | OBJECT CODE         | INSTRUCTION       |
|---------------------|---------------------|---------------------|-------------------|
| CB 1E               | RR (HL)             | DD CB disp 10bbb110 | RES b,(IX + disp) |
| CB 2 0rr            | SLA reg             | DD CB disp 11bbb110 | SET b,(IX + disp) |
| CB 26               | SLA (HL)            | DD E1               | POP IX            |
| CB 2 1rr            | SRA reg             | DD E3               | EX (SP),IX        |
| CB 2E               | SRA (HL)            | DD E5               | PUSH IX           |
| CB 3 1rr            | SRL reg             | DD E9               | JP (IX)           |
| CB 3E               | SRL (HL)            | DD F9               | LD SP,IX          |
| CB 01bbbrrr         | BIT b,reg           | DE yy               | SBC A,data        |
| CB 01bbb110         | BIT b,(HL)          | DF                  | RST 18H           |
| CB 10bbbrrr         | RES b,reg           | E0                  | RET PO            |
| CB 10bbb110         | RES b,(HL)          | E1                  | POP HL            |
| CB 11bbbrrr         | SET b,reg           | E2 ppqq             | JP PO,addr        |
| CB 11bbb110         | SET b,(HL)          | E3                  | EX (SP),HL        |
| CC ppqq             | CALL Z,addr         | E4 ppqq             | CALL PO,addr      |
| CD ppqq             | CALL addr           | E5                  | PUSH HL           |
| CE yy               | ADC A,data          | E6 yy               | AND data          |
| CF                  | RST 08H             | E7                  | RST 20H           |
| D0                  | RET NC              | E8                  | RET PE            |
| D1                  | POP DE              | E9                  | JP (HL)           |
| D2 ppqq             | JP NC,addr          | EA ppqq             | JP PE,addr        |
| D3 yy               | OUT (port),A        | EB                  | EX DE,HL          |
| D4 ppqq             | CALL NC,addr        | EC ppqq             | CALL PE,addr      |
| D5                  | PUSH DE             | ED 01ddd000         | IN reg,(C)        |
| D6 yy               | SUB data            | ED 01sss001         | OUT (C),reg       |
| D7                  | RST 10H             | ED 01xx 2           | SBC HL,rp         |
| D8                  | RET C               | ED 01xx 3 ppqq      | LD (addr),rp      |
| D9                  | EXX                 | ED 44               | NEG               |
| DA ppqq             | JP C,addr           | ED 45               | RETN              |
| DB yy               | IN A,(port)         | ED 010nn110         | IM m              |
| DC ppqq             | CALL C,addr         | ED 47               | LD I,A            |
| DD 00xx 9           | ADD IX,pp           | ED 01xx A           | ADC HL,rp         |
| DD 21 yyyy          | LD IX,data16        | ED 01xx B ppqq      | LD rp,(addr)      |
| DD 22 ppqq          | LD (addr),IX        | ED 4D               | RETI              |
| DD 23               | INC IX              | ED 4F               | LD R,A            |
| DD 2A ppqq          | LD IX,(addr)        | ED 57               | LD A,I            |
| DD 2B               | DEC IX              | ED 5F               | LD A,R            |
| DD 34 disp          | INC (IX + disp)     | ED 87               | RRD               |
| DD 35 disp          | DEC (IX + disp)     | ED 6F               | RLD               |
| DD 36 disp yy       | LD (IX + disp),data | ED A0               | LDI               |
| DD 01ddd110 disp    | LD reg,(IX + disp)  | ED A1               | CPI               |
| DD / 0sss disp      | LD (IX + disp),reg  | ED A2               | INI               |
| DD 86 disp          | ADD A,(IX + disp)   | ED A3               | OUTI              |
| DD 8E disp          | ADC A,(IX + disp)   | ED A8               | LDD               |
| DD 96 disp          | SUB (IX + disp)     | ED A9               | CPD               |
| DD 9E disp          | SBC A,(IX + disp)   | ED AA               | IND               |
| DD A8 disp          | AND (IX + disp)     | ED AB               | OUTD              |
| DD AE disp          | XOR (IX + disp)     | ED B0               | LDI               |
| DD B8 disp          | OR (IX + disp)      | ED B1               | CPIR              |
| DD BE disp          | CP (IX + disp)      | ED B2               | INIR              |
| DD CB disp 06       | RLC (IX + disp)     | ED B3               | OTIR              |
| DD CB disp 0E       | RRC (IX + disp)     | ED B8               | LDDR              |
| DD CB disp 16       | RL (IX + disp)      | ED B9               | CPDR              |
| DD CB disp 1E       | RR (IX + disp)      | ED BA               | INDR              |
| DD CB disp 26       | SLA (IX + disp)     | ED BB               | OTDR              |
| DD CB disp 2E       | SRA (IX + disp)     | EE yy               | XOR data          |
| DD CB disp 3E       | SRL (IX + disp)     | EF                  | RST 28H           |
| DD CB disp 01bbb110 | BIT b,(IX + disp)   |                     |                   |

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

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

## ADC A,data — ADD IMMEDIATE WITH CARRY TO ACCUMULATOR



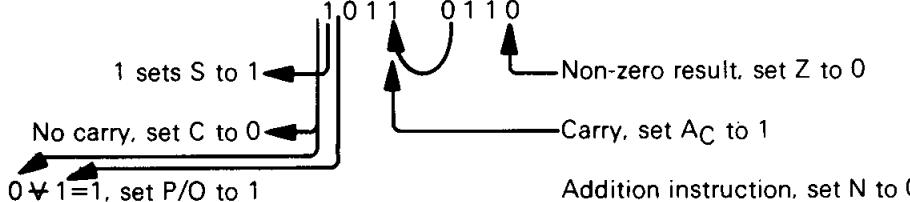
Add the contents of the next program memory byte and the Carry status to the Accumulator.

Suppose  $xx=3A_{16}$ ,  $yy=7C_{16}$ , and  $\text{Carry}=0$ . After the instruction

ADC A,7CH

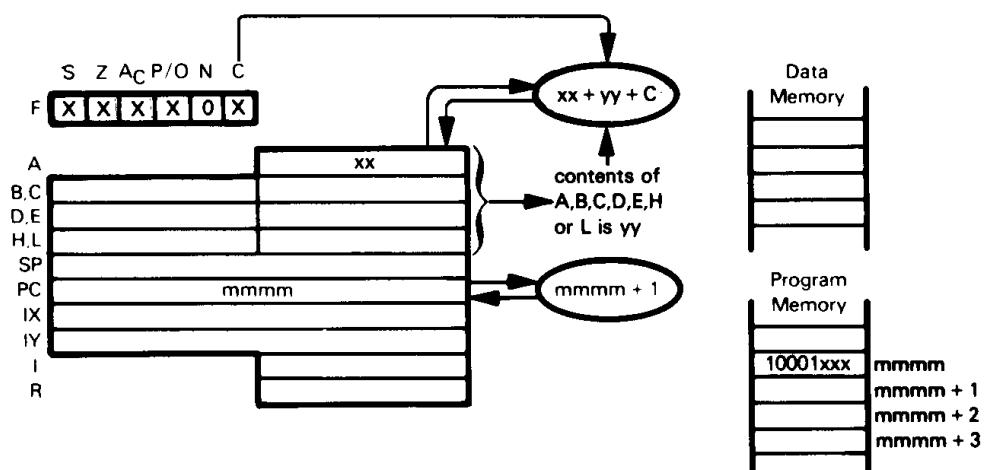
has executed, the Accumulator will contain  $B6_{16}$ :

$$\begin{array}{r}
 3A = 0011\ 1010 \\
 7C = 0111\ 1100 \\
 \text{Carry} = 0
 \end{array}
 \quad \underline{1011\ 0110}$$



The ADC instruction is frequently used in multibyte addition for the second and subsequent bytes.

## ADC A,reg — ADD REGISTER WITH CARRY TO ACCUMULATOR



ADC A,  
 $\underbrace{10001}_{\text{10001}}$   $\underbrace{\text{xxx}}_{\text{reg}}$   
 000 for reg=B  
 001 for reg=C  
 010 for reg=D  
 011 for reg=E  
 100 for reg=H  
 101 for reg=L  
 111 for reg=A

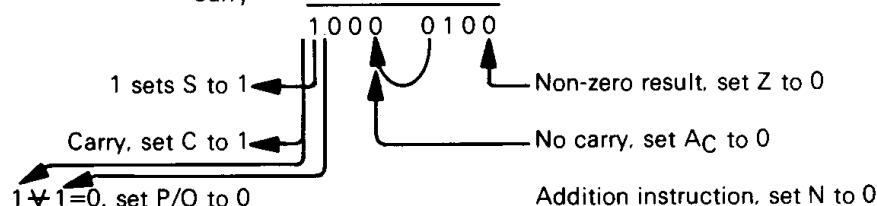
Add the contents of Register A, B, C, D, E, H or L and the Carry status to the Accumulator.

Suppose xx=E316, Register E contains A016, and Carry=1. After the instruction

ADC A,E

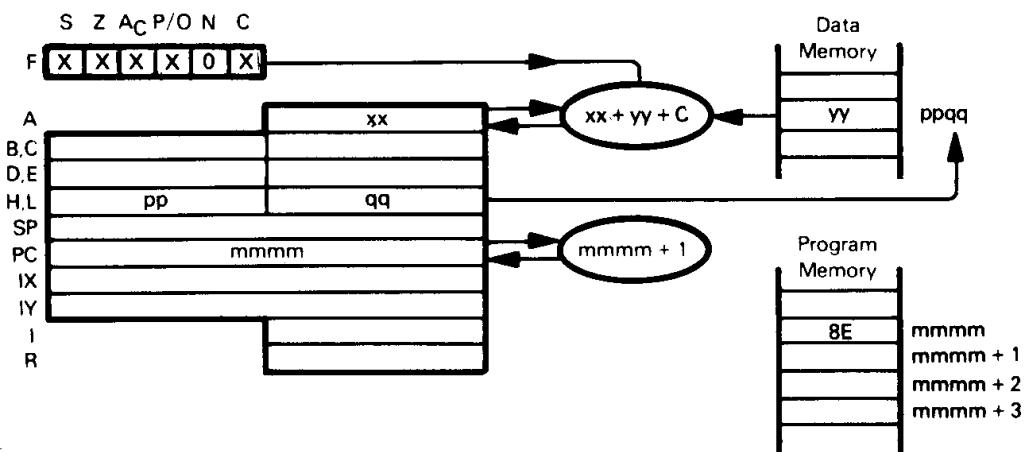
has executed, the Accumulator will contain 8416:

$$\begin{array}{r}
 E3 = 1110 0011 \\
 A0 = 1010 0000 \\
 \text{Carry} = \underline{1} \\
 \hline
 \end{array}$$



The ADC instruction is most frequently used in multibyte addition for the second and subsequent bytes.

**ADC A,(HL) — ADD MEMORY AND CARRY TO  
ADC A,(IX+disp) ACCUMULATOR  
ADC A,(IY+disp)**



The illustration shows execution of ADC A,(HL):

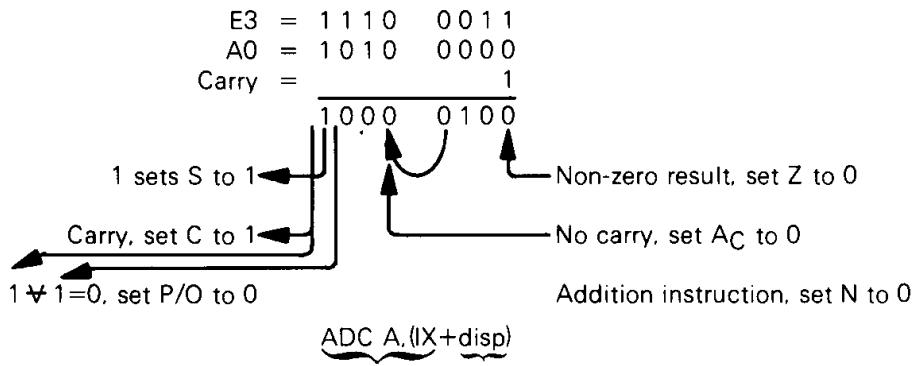
ADC A,(HL)  
8E

Add the contents of memory location (specified by the contents of the HL register pair) and the Carry status to the Accumulator.

Suppose xx=E316, yy=A016, and Carry=1. After the instruction

ADC A,(HL)

has executed, the Accumulator will contain 8416:



Add the contents of memory location (specified by the sum of the contents of the IX register and the displacement digit d) and the Carry to the Accumulator.

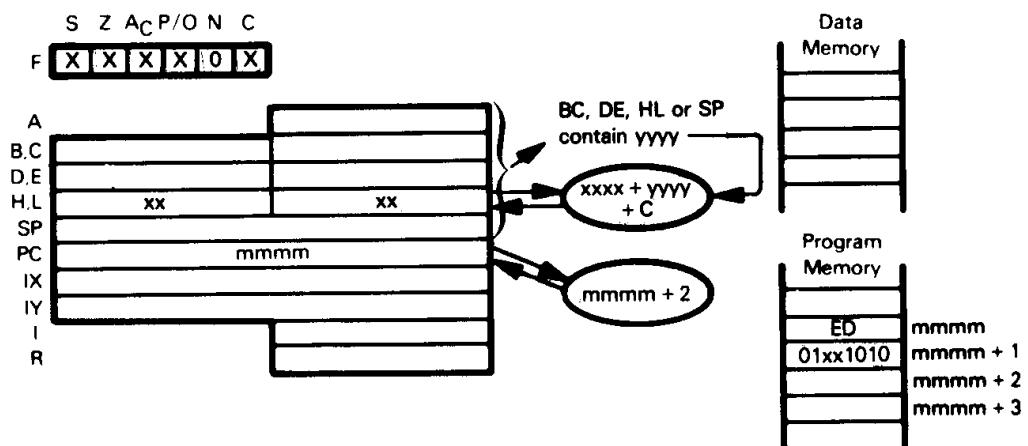
ADC A,(IY+disp)

FD 8E d

This instruction is identical to ADC A,(IX+disp), except that it uses the IY register instead of the IX register.

The ADC instruction is most frequently used in multibyte addition for the second and subsequent bytes.

## ADC HL,rp — ADD REGISTER PAIR WITH CARRY TO H AND L

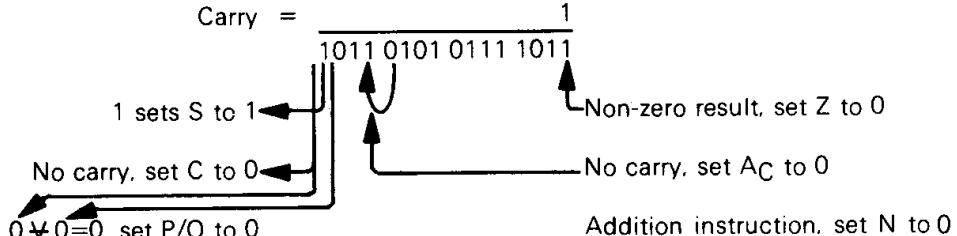


Add the 16-bit value from either the BC, DE, HL register pair or the Stack Pointer, and the Carry status, to the HL register pair.

Suppose HL contains A536<sub>16</sub>, BC contains 1044<sub>16</sub>, and Carry=1. After execution of  
ADC    HL,BC

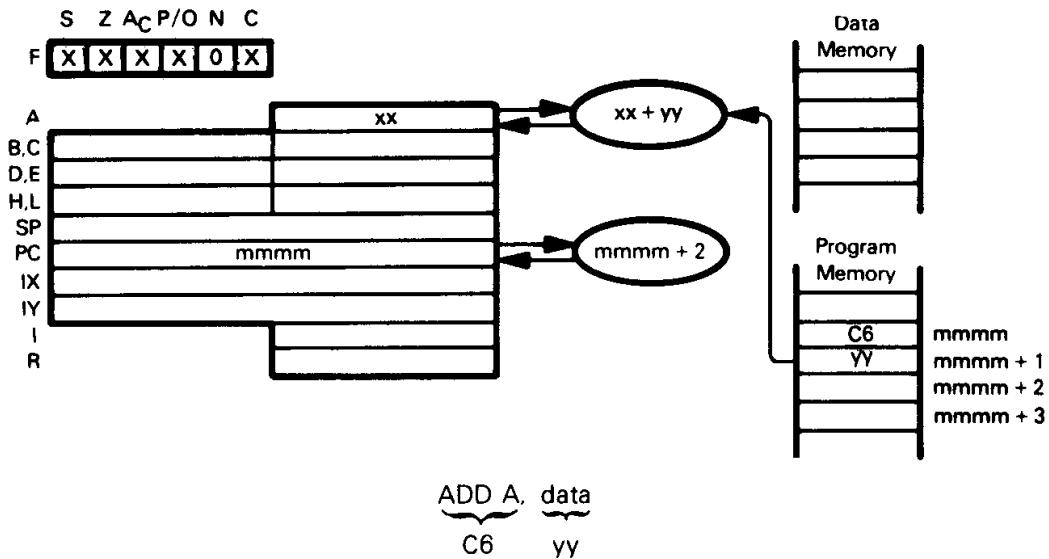
the HL register pair will contain:

$$\begin{array}{rcl} A536 & = & 1010\ 0101\ 0011\ 0110 \\ 1044 & = & 0001\ 0000\ 0100\ 0100 \\ \text{Carry} & = & 1 \end{array}$$



The ADC instruction is most frequently used in multibyte addition for the second and subsequent bytes.

## ADD A,data — ADD IMMEDIATE TO ACCUMULATOR

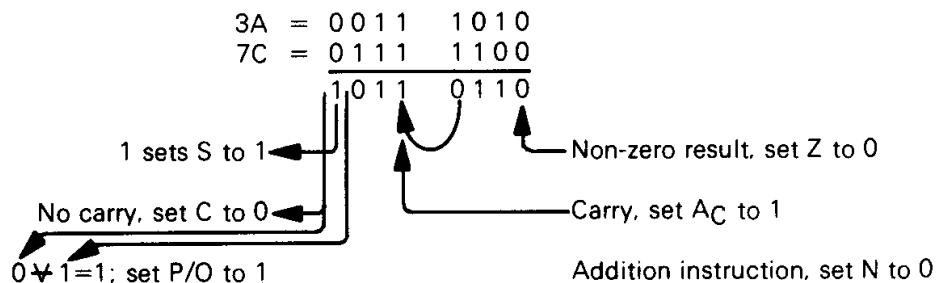


Add the contents of the next program memory byte to the Accumulator.

Suppose  $xx=3A16$ ,  $yy=7C16$ , and Carry=0. After the instruction

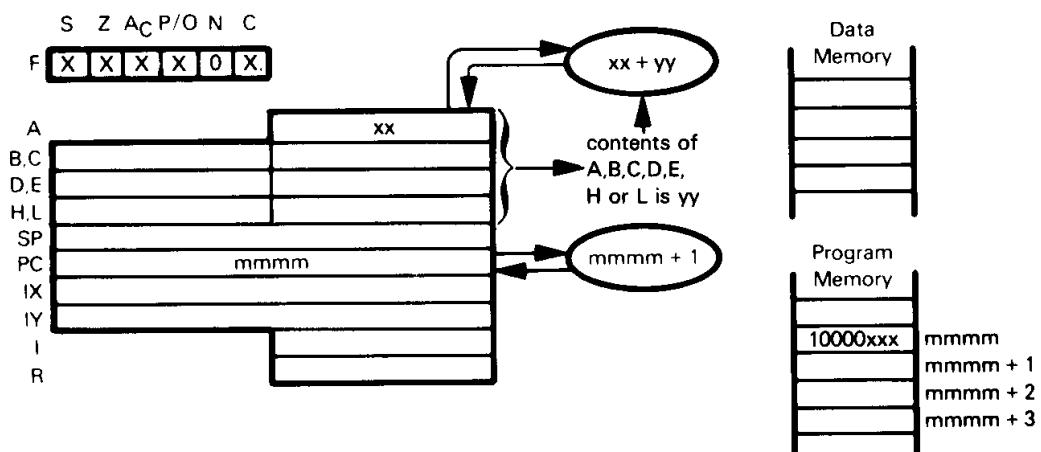
ADD A,7CH

has executed, the Accumulator will contain B616:



This is a routine data manipulation instruction.

## ADD A,reg — ADD CONTENTS OF REGISTER TO ACCUMULATOR



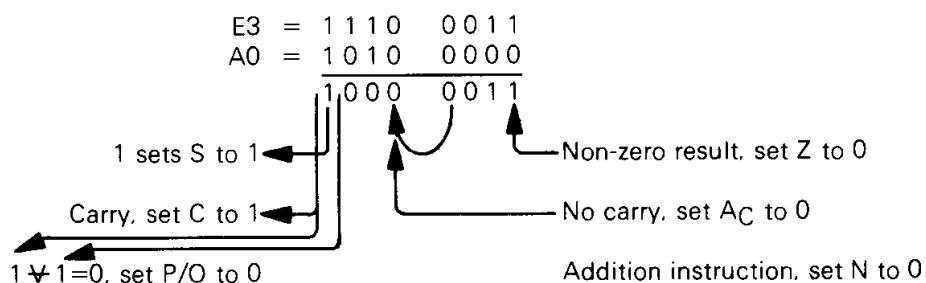
$\begin{array}{r} \text{ADD} \quad \text{reg} \\ \hline 10000 \quad \text{xxx} \\ \quad \quad \quad \text{000} \quad \text{for reg=B} \\ \quad \quad \quad 001 \quad \text{for reg=C} \\ \quad \quad \quad 010 \quad \text{for reg=D} \\ \quad \quad \quad 011 \quad \text{for reg=E} \\ \quad \quad \quad 100 \quad \text{for reg=H} \\ \quad \quad \quad 101 \quad \text{for reg=L} \\ \quad \quad \quad 111 \quad \text{for reg=A} \end{array}$

Add the contents of Register A, B, C, D, E, H or L to the Accumulator.

Suppose xx=E316. Register E contains A016. After execution of

ADD A,E

the Accumulator will contain 8316:

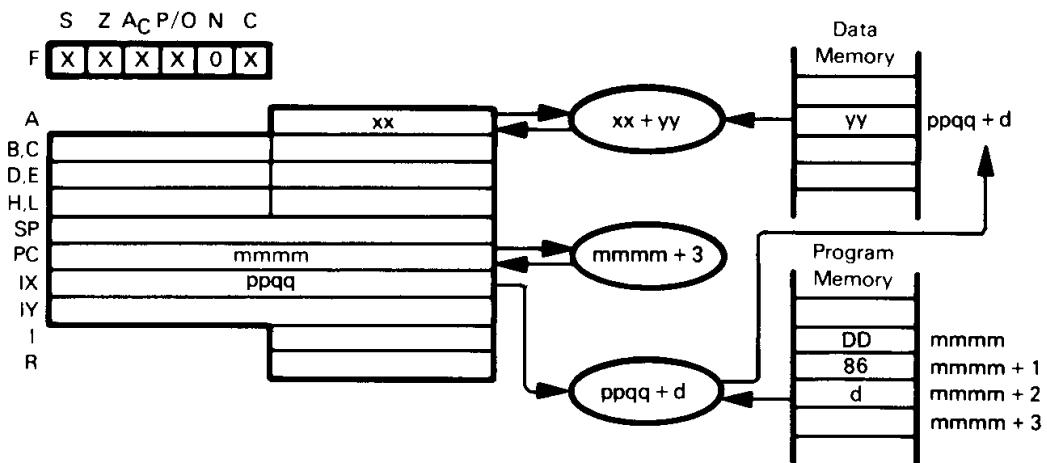


This is a routine data manipulation instruction

## ADD A,(HL) — ADD MEMORY TO ACCUMULATOR

**ADD A,(IX+disp)**

**ADD A,(IY+disp)**



The illustration shows execution of ADD A,(IX+disp).

ADD A,(IX+disp)

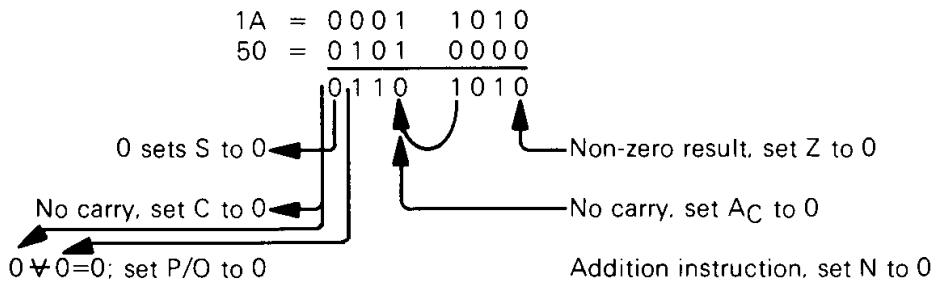
DD 86 d

Add the contents of memory location (specified by the sum of the contents of the IX register and the displacement digit d) to the contents of the Accumulator.

Suppose  $ppqq=4000_{16}$ ,  $xx=1A_{16}$ , and memory location  $400F_{16}$  contains  $50_{16}$ . After the instruction

ADD A,(IX+0FH)

has executed, the Accumulator will contain  $6A_{16}$ .



ADD A,(IY+disp)

FD 86 d

This instruction is identical to ADD A,(IX+disp), except that it uses the IY register instead of the IX register.

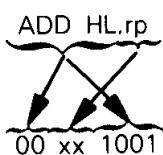
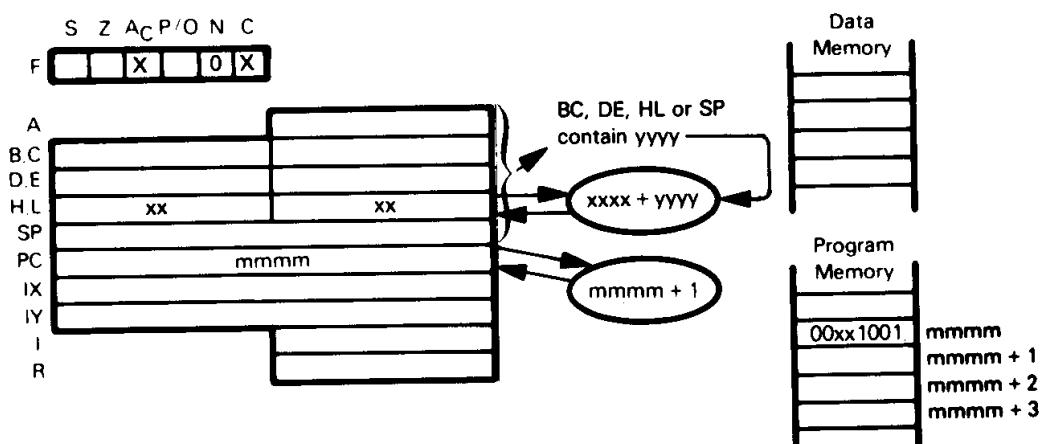
ADD A,(HL)

86

This version of the instruction adds the contents of memory location, specified by the contents of the HL register pair, to the Accumulator.

The ADD instruction is a routine data manipulation instruction.

## ADD HL,rp — ADD REGISTER PAIR TO H AND L



00 for rp is register pair BC  
 01 for rp is register pair DE  
 10 for rp is register pair HL  
 11 for rp is Stack Pointer

Add the 16-bit value from either the BC, DE, HL register pair or the Stack Pointer to the HL register pair.

Suppose HL contains 034A<sub>16</sub> and BC contains 214C<sub>16</sub>. After the instruction

ADD HL,BC

has executed, the HL register pair will contain 249616.

$$\begin{array}{r}
 034A = 0000\ 0011\ 0100\ 1010 \\
 214C = 0010\ 0001\ 0100\ 1100 \\
 \hline
 & 0010\ 0100\ 1001\ 0110
 \end{array}$$

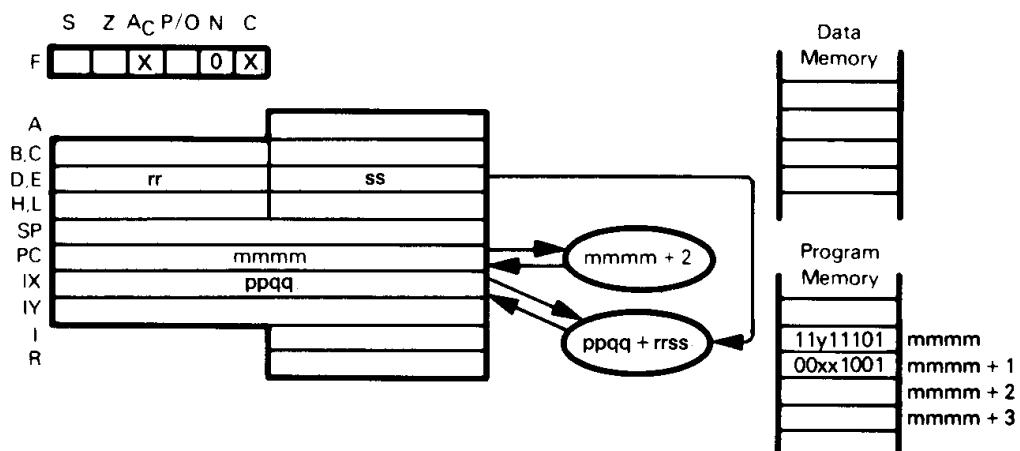
No carry, set C to 0

No carry, set AC to 0

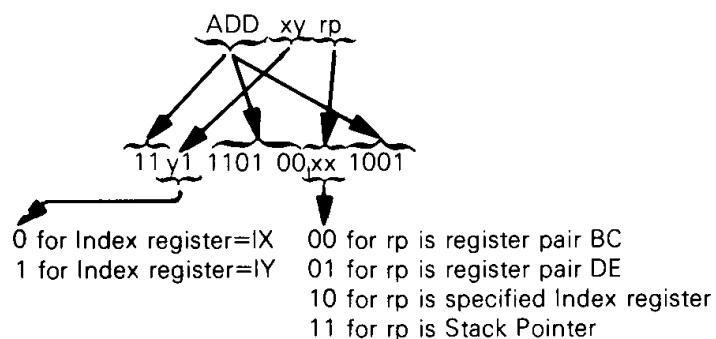
Addition instruction, set N to 0

The ADD HL,HL instruction is equivalent to a 16-bit left shift.

## ADD xy, rp — ADD REGISTER PAIR TO INDEX REGISTER



The illustration shows execution of ADD IX,DE.



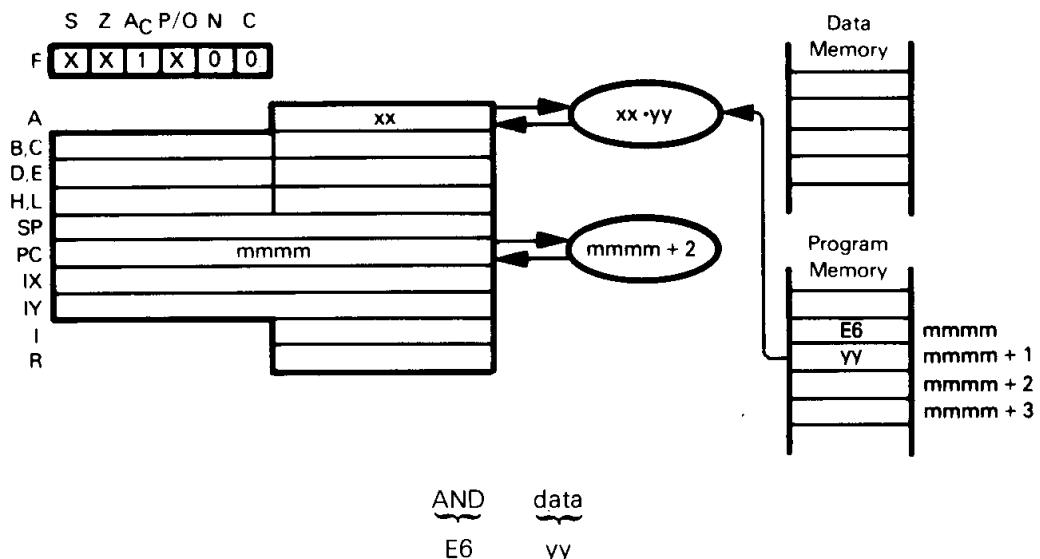
Add the contents of the specified register pair to the contents of the specified Index register.

Suppose IY contains 4FF0<sub>16</sub> and BC contains 000F<sub>16</sub>. After the instruction

ADD IY,BC

has executed, Index Register IY will contain 4FFF<sub>16</sub>.

## AND data — AND IMMEDIATE WITH ACCUMULATOR



AND the contents of the next program memory byte to the Accumulator.

Suppose  $xx=3A_{16}$ . After the instruction

AND 7CH

has executed, the Accumulator will contain 3816.

$$\begin{array}{r}
 3A = 0011\ 1010 \\
 7C = 0111\ 1100 \\
 \hline
 0011\ 1000
 \end{array}$$

0 sets S to 0

Three 1 bits, set P/O to 0

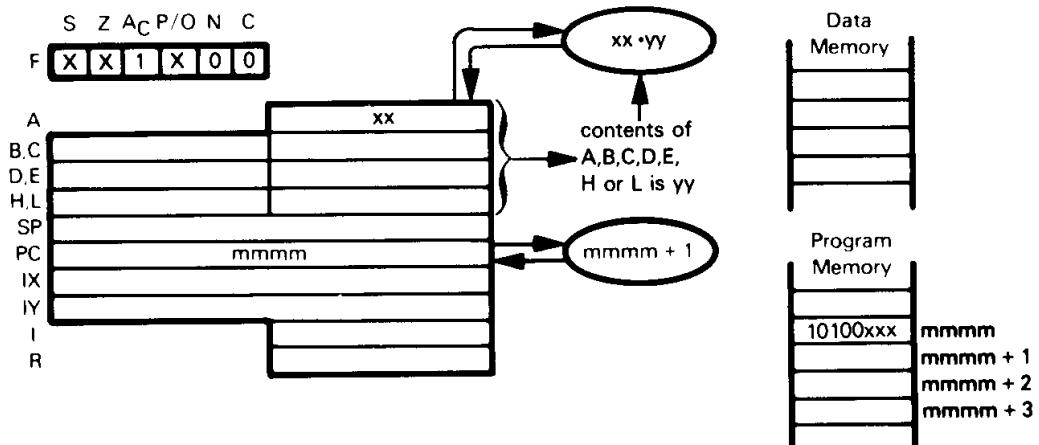
Non-zero result, set Z to 0

This is a routine logical instruction; it is often used to turn bits "off". For example, the instruction

AND 7FH

will unconditionally set the high order Accumulator bit to 0.

## AND reg — AND REGISTER WITH ACCUMULATOR



| <u>AND</u> | <u>reg</u>    |
|------------|---------------|
| 10100      | xxx           |
|            | 000 for reg=B |
|            | 001 for reg=C |
|            | 010 for reg=D |
|            | 011 for reg=E |
|            | 100 for reg=H |
|            | 101 for reg=L |
|            | 111 for reg=A |

AND the Accumulator with the contents of Register A, B, C, D, E, H or L. Save the result in the Accumulator.

Suppose xx=E316, and Register E contains A016. After the instruction

AND E

has executed, the Accumulator will contain A016.

$$\begin{array}{r}
 E3 = 1110 \quad 0011 \\
 A0 = 1010 \quad 0000 \\
 \hline
 1010 \quad 0000
 \end{array}$$

1 sets S to 1

Two 1 bits, set P/O to 1

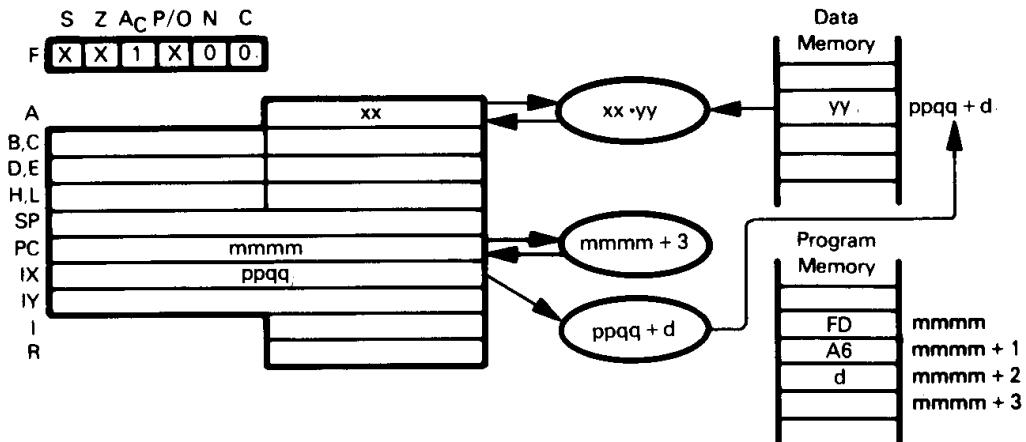
Non-zero result, set Z to 0

AND is a frequently used logical instruction.

## AND (HL) — AND MEMORY WITH ACCUMULATOR

**AND (IX+disp)**

**AND (IY+disp)**



The illustration shows execution of AND (IY+disp).

$\underbrace{\text{AND (IY+disp)}}$   
 FD A6 d

AND the contents of memory location (specified by the sum of the contents of the IY register and the displacement digit d) with the Accumulator.

Suppose xx=E316, ppqq=400016, and memory location 400F16 contains A016. After the instruction

AND (IY+0FH)

has executed, the Accumulator will contain A016.

$$\begin{array}{r}
 E3 = 1110 \quad 0111 \\
 A0 = 1010 \quad 0000 \\
 \hline
 1010 \quad 0000
 \end{array}$$

1 sets S to 1

Two 1 bits, set P/O to 1

Non-zero result, set Z to 0

$\underbrace{\text{AND (IX+disp)}}$   
 DD A6 d

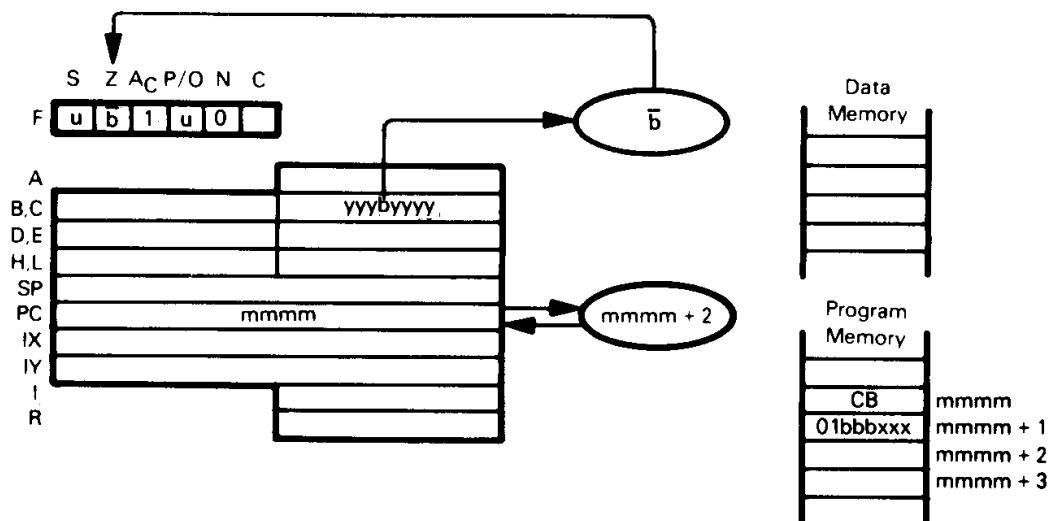
This instruction is identical to AND (IY+disp), except that it uses the IX register instead of the IY register.

$\underbrace{\text{AND (HL)}}$   
 A6

AND the contents of the memory location (specified by the contents of the HL register pair) with the Accumulator.

AND is a frequently used logical instruction.

## BIT b,reg — TEST BIT b IN REGISTER reg



| <u>BIT</u> | <u>b,</u> | <u>reg</u> |
|------------|-----------|------------|
| CB 01      | bbb       | xxx        |
| Bit Tested |           | Register   |
| 0          | 000       | B          |
| 1          | 001       | C          |
| 2          | 010       | D          |
| 3          | 011       | E          |
| 4          | 100       | H          |
| 5          | 101       | L          |
| 6          | 110       | A          |
| 7          | 111       |            |

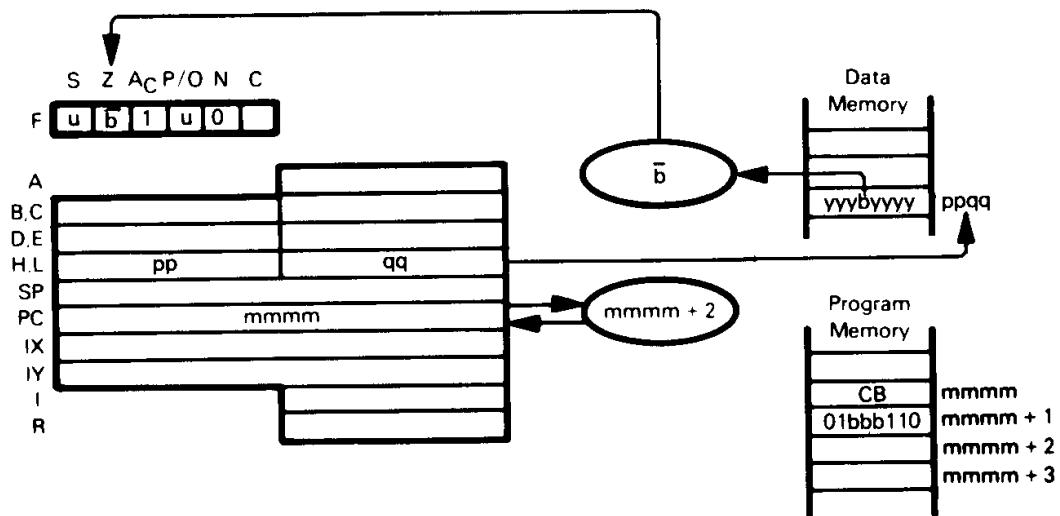
Place complement of indicated register's specified bit in Z flag of F register.

Suppose Register C contains 1110 1111. The instruction BIT 4,C will then set the Z flag to 1, while bit 4 in Register C remains 0. Bit 0 is the least significant bit.

## **BIT b,(HL) — TEST BIT b OF INDICATED MEMORY POSITION**

**BIT b,(IX+disp)**

**BIT b,(IY+disp)**



The illustration shows execution of BIT 4,(HL). Bit 0 is the least significant bit.

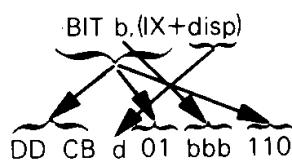
| <u>BIT</u> | <u>b,</u>  | <u>(HL)</u> |
|------------|------------|-------------|
| CB 01      | <u>bbb</u> | 110         |
| Bit Tested | <u>bbb</u> |             |
| 0          | 000        |             |
| 1          | 001        |             |
| 2          | 010        |             |
| 3          | 011        |             |
| 4          | 100        |             |
| 5          | 101        |             |
| 6          | 110        |             |
| 7          | 111        |             |

Test indicated bit within memory position specified by the contents of Register HL, and place bit's complement in Z flag of the F register.

Suppose HL contains 4000H and bit 3 in memory location 4000H contains 1. The instruction

BIT 3,(HL)

will then set the Z flag to 0, while bit 3 in memory location 4000H remains 1.



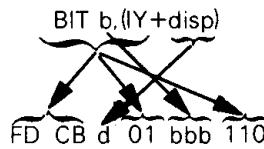
bbb is the same as in BIT b,(HL)

Examine specified bit within memory location indicated by the sum of Index Register IX and disp. Place the complement in the Z flag of the F register.

Suppose Index Register IX contains 4000H and bit 4 of memory location 4004H is 0. The instruction

BIT 4, (IX+4H)

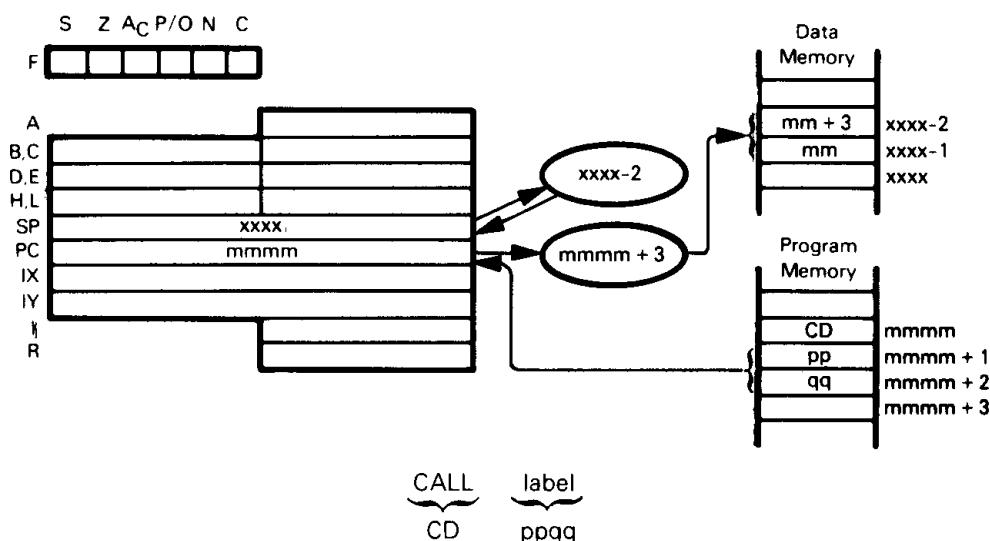
will then set the Z flag to 1, while bit 4 of memory location 4004H remains 0.



bbb is the same as in BIT b, (HL)

This instruction is identical to BIT b, (IX+disp), except that it uses the IY register instead of the IX register.

## **CALL label — CALL THE SUBROUTINE IDENTIFIED IN THE OPERAND**



Store the address of the instruction following the CALL on the top of the stack: the top of the stack is a data memory byte addressed by the Stack Pointer. Then subtract 2 from the Stack Pointer in order to address the new top of stack. Move the 16-bit address contained in the second and third CALL instruction object program bytes to the Program Counter. The second byte of the CALL instruction is the low-order half of the address, and the third byte is the high-order byte.

Consider the instruction sequence:

CALL      SUBR  
AND      7CH

-

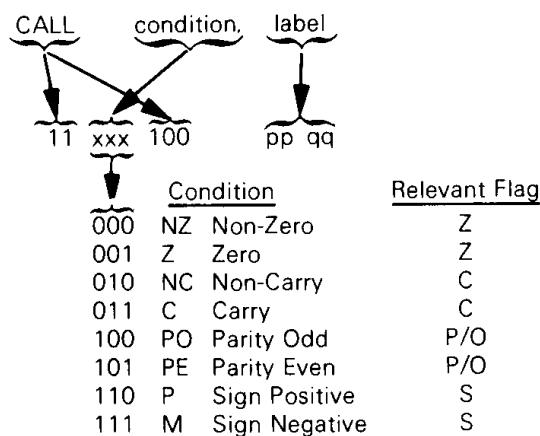
-

-

SUBR

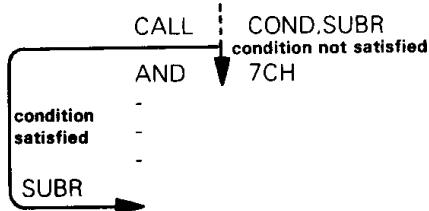
After the instruction has executed, the address of the AND instruction is saved at the top of the stack. The Stack Pointer is decremented by 2. The instruction labeled SUBR will be executed next.

**CALL condition,label — CALL THE SUBROUTINE IDENTIFIED IN THE OPERAND IF CONDITION IS SATISFIED**



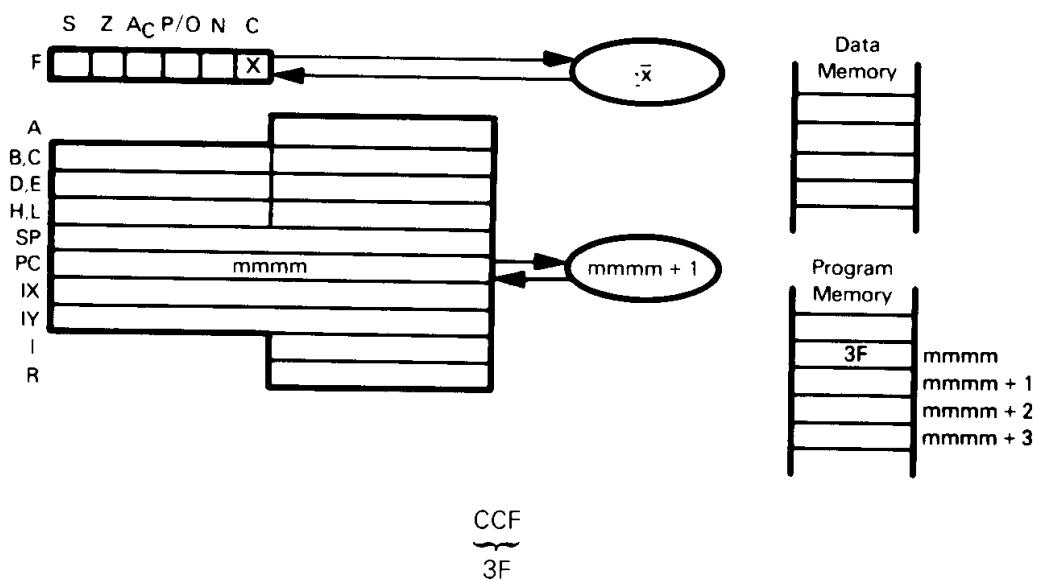
This instruction is identical to the CALL instruction, except that the identified subroutine will be called only if the condition is satisfied; otherwise, the instruction sequentially following the CALL condition instruction will be executed.

Consider the instruction sequence:



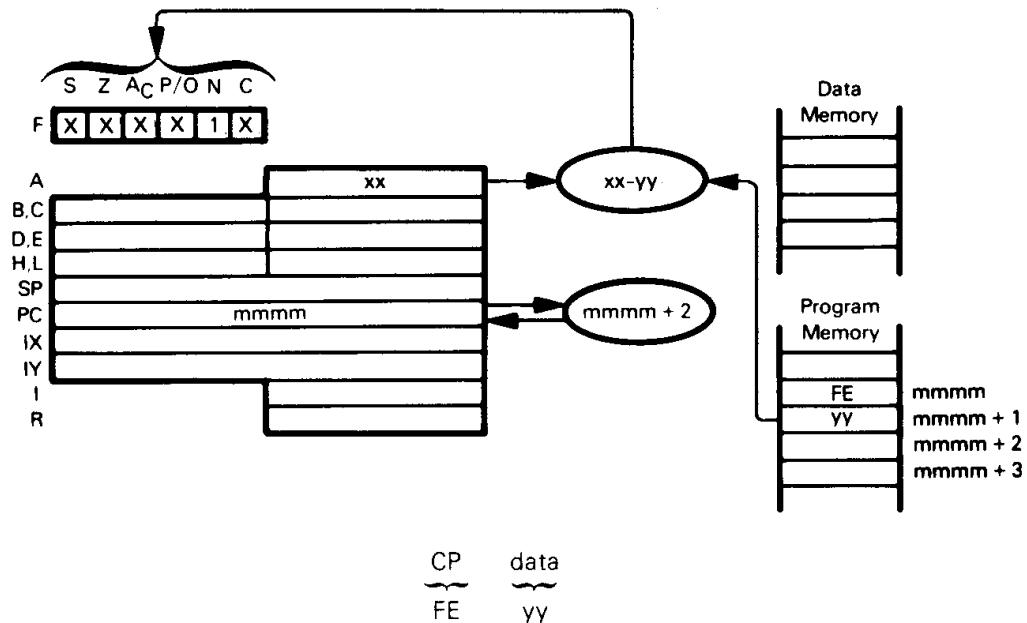
If the condition is not satisfied, the AND instruction will be executed after the CALL COND.SUBR instruction has executed. If the condition is satisfied, the address of the AND instruction is saved at the top of the stack, and the Stack Pointer is decremented by 2. The instruction labeled SUBR will be executed next.

## CCF — COMPLEMENT CARRY FLAG



Complement the Carry flag. No other status or register contents are affected.

## CP data — COMPARE IMMEDIATE DATA WITH ACCUMULATOR



Subtract the contents of the second object code byte from the contents of the Accumulator, treating both numbers as simple binary data. Discard the result; i.e., leave the Accumulator alone, but modify the status flags to reflect the result of the subtraction.

Suppose  $xx = E3_{16}$  and the second byte of the CP instruction object code contains  $A0_{16}$ . After the instruction

CP 0A0H

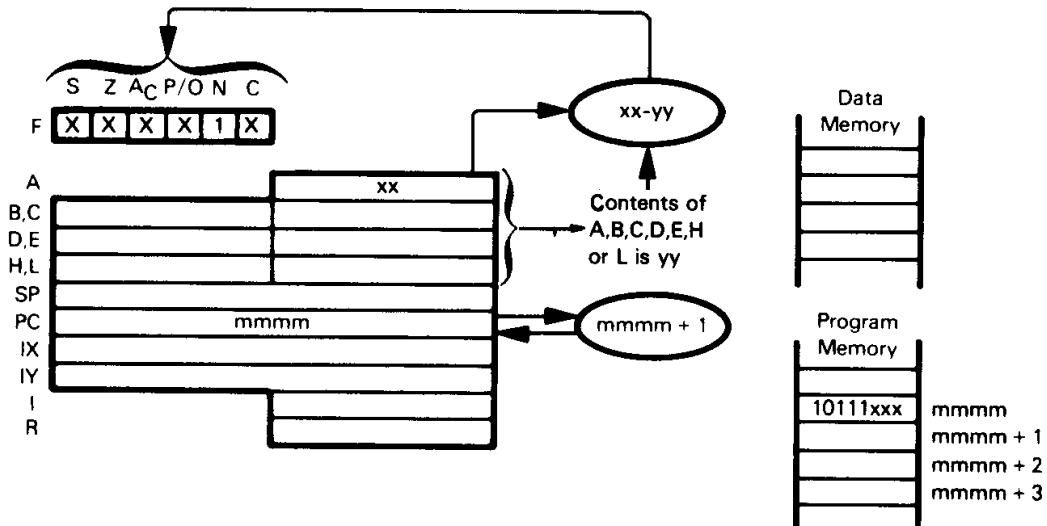
has executed, the Accumulator will still contain  $E3_{16}$ , but statuses will be modified as follows:

$$\begin{array}{r}
 E3 = 1110\ 0011 \\
 A0 = 1010\ 0000 \\
 \hline
 0100\ 0011
 \end{array}$$

Non-zero result, set Z to 0  
 No borrow, set AC to 0  
 1 ≠ 1 = 0, set P/O to 0  
 Subtract instruction, set N to 1

Notice that the resulting carry is complemented.

## CP reg — COMPARE REGISTER WITH ACCUMULATOR



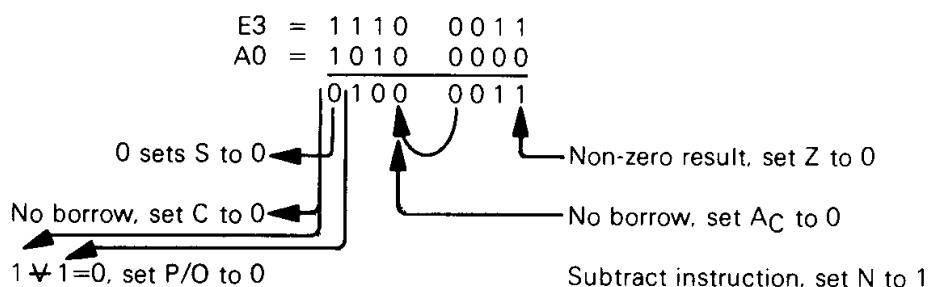
| CP    | reg       |
|-------|-----------|
| 10111 | xxx       |
| 000   | for reg=B |
| 001   | for reg=C |
| 010   | for reg=D |
| 011   | for reg=E |
| 100   | for reg=H |
| 101   | for reg=L |
| 111   | for reg=A |

Subtract the contents of Register A, B, C, D, E, H or L from the contents of the Accumulator, treating both numbers as simple binary data. Discard the result; i.e., leave the Accumulator alone, but modify status flags to reflect the result of the subtraction.

Suppose xx=E3<sub>16</sub> and Register B contains A0<sub>16</sub>. After the instruction

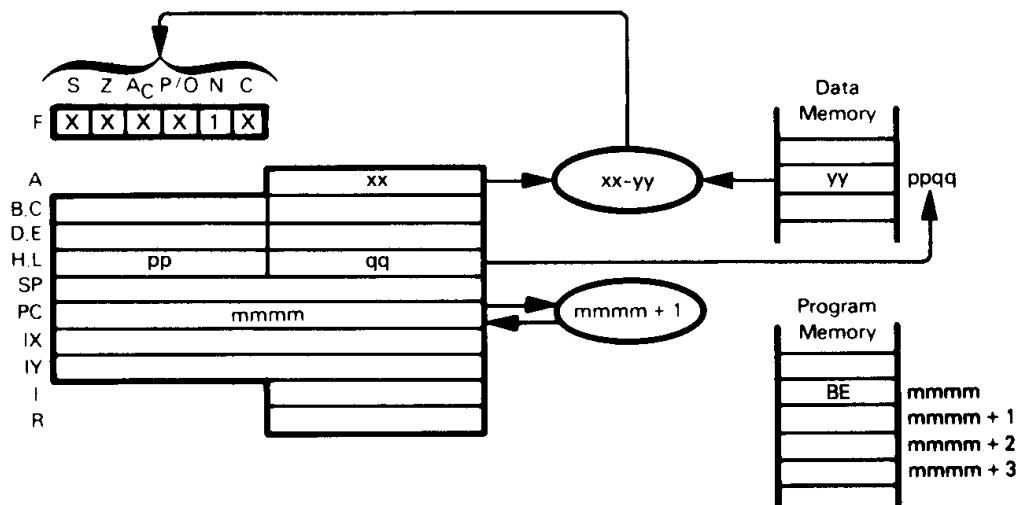
CP B

has executed, the Accumulator will still contain E3<sub>16</sub>, but statuses will be modified as follows:



Notice that the resulting carry is complemented.

**CP (HL) — COMPARE MEMORY WITH ACCUMULATOR**  
**CP (IX+disp)**  
**CP (IY+disp)**



The illustration shows execution of CP (HL):

CP (HL)  
BE

Subtract the contents of memory location (specified by the contents of the HL register pair) from the contents of the Accumulator, treating both numbers as simple binary data. Discard the result; i.e., leave the Accumulator alone, but modify status flags to reflect the result of the subtraction.

Suppose xx=E3<sub>16</sub> and yy=A0<sub>16</sub>. After execution of

CP (HL)

the Accumulator will still contain E3<sub>16</sub>, but statuses will be modified as follows:

|                       |       |                                    |
|-----------------------|-------|------------------------------------|
| E3 =                  | 1110  | 0011                               |
| A0 =                  | 0110  | 0000                               |
|                       | <hr/> |                                    |
|                       | 0100  | 0011                               |
| 0 sets S to 0         |       | Non-zero result, set Z to 0        |
| No borrow, set C to 0 |       | No borrow, set A <sub>C</sub> to 0 |
| 1 ≠ 1=0, set P/O to 0 |       | Subtract instruction, set N to 1   |

Notice that the resulting carry is complemented.

CP (IX+disp)

DD BE d

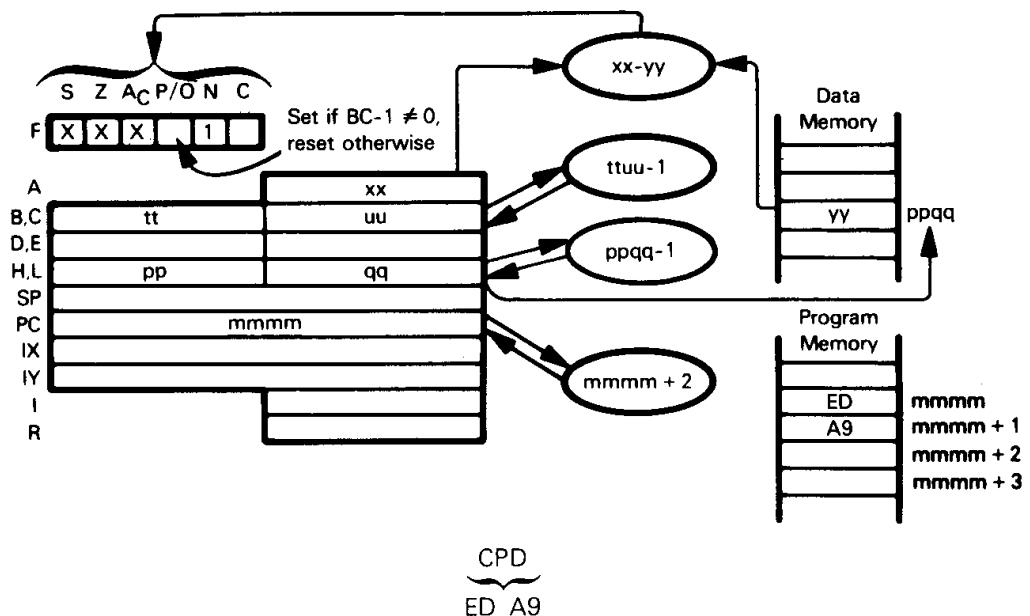
Subtract the contents of memory location (specified by the sum of the contents of the IX register and the displacement value d) from the contents of the Accumulator, treating both numbers as simple binary data. Discard the result; i.e., leave the Accumulator alone, but modify status flags to reflect the result of the subtraction.

CP (IY+disp)

FD BE d

This instruction is identical to CP (IX+disp), except that it uses the IY register instead of the IX register.

### CPD — COMPARE ACCUMULATOR WITH MEMORY. DECREMENT ADDRESS AND BYTE COUNTER

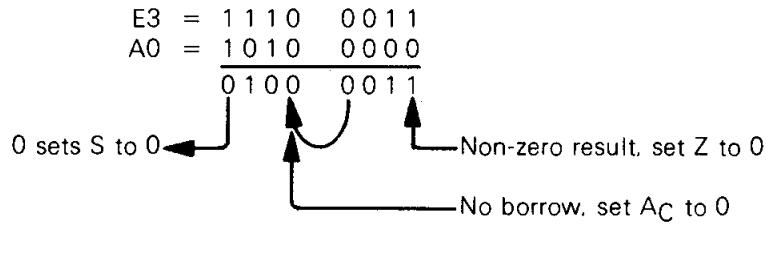


Compare the contents of the Accumulator with the contents of memory location (specified by the HL register pair). If A is equal to memory, set Z flag. Decrement the HL and BC register pairs. (BC is used as the Byte Counter.)

Suppose  $xx=E3_{16}$ ,  $ppqq=4000_{16}$ , BC contains  $0001_{16}$ , and  $yy=A0_{16}$ . After the instruction

CPD

has executed, the Accumulator will still contain  $E3_{16}$ , but statuses will be modified as follows:



The P/O flag will be reset because  $BC-1=0$

Subtract instruction involved, set N to 1

Carry not affected.

The HL register pair will contain  $3FFF_{16}$ , and BC=0.

**CPDR — COMPARE ACCUMULATOR WITH MEMORY.  
DECREMENT ADDRESS AND BYTE COUNTER.  
CONTINUE UNTIL MATCH IS FOUND OR BYTE  
COUNTER IS ZERO**

CPDR  
ED B9

This instruction is identical to CPD, except that it is repeated until a match is found or the byte counter is zero. After each data transfer, interrupts will be recognized and two refresh cycles will be executed.

Suppose the HL register pair contains  $5000_{16}$ , the BC register pair contains  $00FF_{16}$ , the Accumulator contains  $F9_{16}$ , and memory has contents as follows:

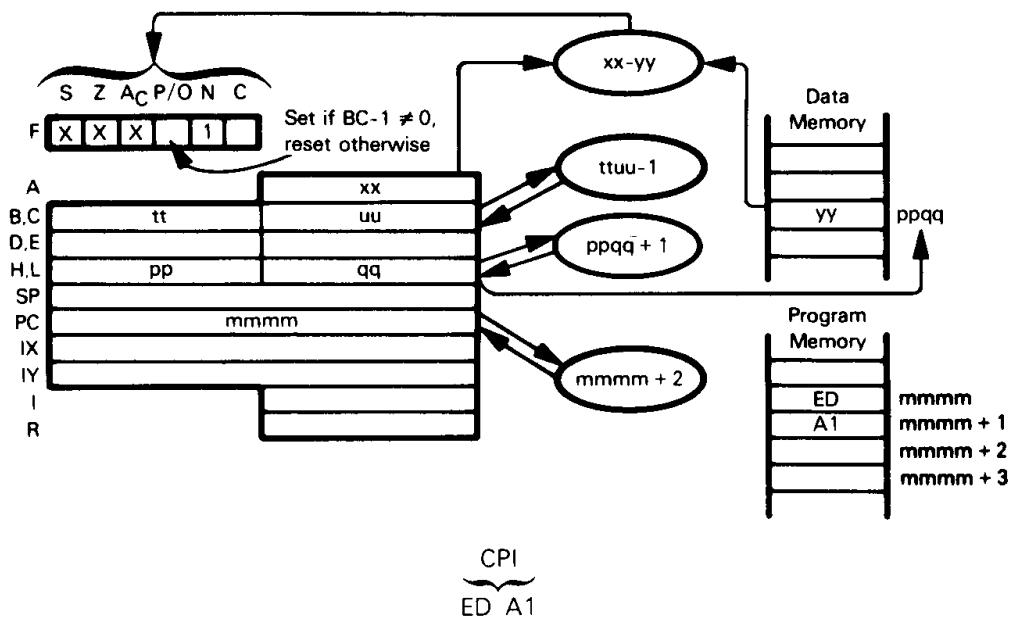
| Location    | Contents  |
|-------------|-----------|
| $5000_{16}$ | $AA_{16}$ |
| $4FFF_{16}$ | $BC_{16}$ |
| $4FFE_{16}$ | $19_{16}$ |
| $4FFD_{16}$ | $7A_{16}$ |
| $4FFC_{16}$ | $F9_{16}$ |
| $4FFB_{16}$ | $DD_{16}$ |

After execution of

CPDR

the P/O flag will be 1, the Z flag will be 1, the HL register pair will contain  $4FFB_{16}$ , and the BC register pair will contain  $00FA_{16}$ .

**CPI — COMPARE ACCUMULATOR WITH MEMORY.  
DECREMENT BYTE COUNTER.  
INCREMENT ADDRESS**



Compare the contents of the Accumulator with the contents of memory location (specified by the HL register pair). If A is equal to memory, set the Z flag. Increment the HL register pair and decrement the BC register pair (BC is used as Byte Counter).

Suppose  $xx=E3_{16}$ ,  $ppqq=4000_{16}$ , BC contains  $0032_{16}$ , and  $yy=E3_{16}$ . After the instruction

CPI

has executed, the Accumulator will still contain  $E3_{16}$ , but statuses will be modified as follows:

$$\begin{array}{r}
 E3 = 1111 \quad 0011 \\
 -E3 = 0000 \quad 1101 \\
 \hline
 0000 \quad 0000
 \end{array}$$

Result is 0, set Z to 1

No borrow, set AC to 0

The P/O flag will be set because  $BC-1 \neq 0$ .

Subtract instruction involved, set N to 1.

Carry not affected.

The HL register pair will contain  $4001_{16}$ , and BC will contain  $0031_{16}$ .

**CPIR — COMPARE ACCUMULATOR WITH MEMORY.  
DECREMENT BYTE COUNTER.  
INCREMENT ADDRESS.  
CONTINUE UNTIL MATCH IS FOUND  
OR BYTE COUNTER IS ZERO**

CPIR  
ED B1

This instruction is identical to CPI, except that it is repeated until a match is found or the byte counter is zero. After each data transfer interrupts will be recognized and two refresh cycles will be executed.

Suppose the HL register pair contains 4500<sub>16</sub>, the BC register pair contains 00FF<sub>16</sub>, the Accumulator contains F9<sub>16</sub>, and memory has contents as follows:

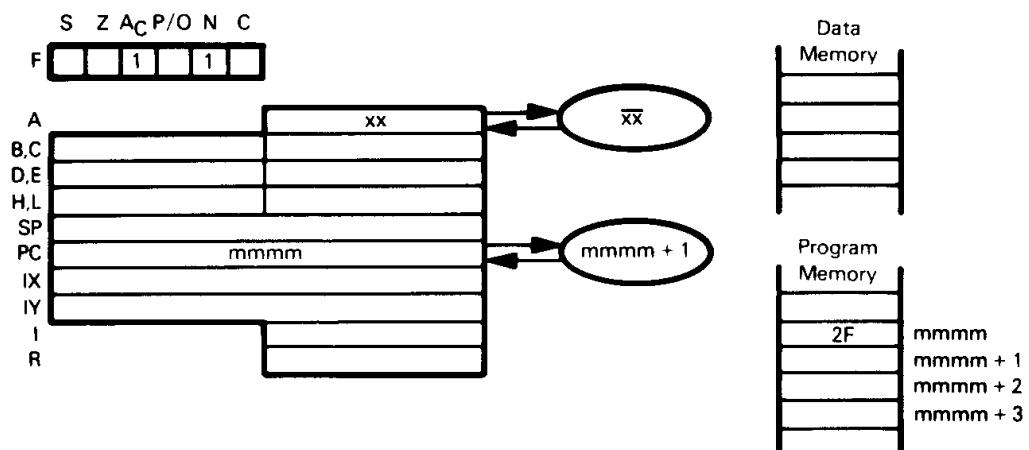
| Location           | Contents         |
|--------------------|------------------|
| 4500 <sub>16</sub> | AA <sub>16</sub> |
| 4501 <sub>16</sub> | 15 <sub>16</sub> |
| 4502 <sub>16</sub> | F9 <sub>16</sub> |

After execution of

CPIR

the P/O flag will be 1, and the Z flag will be 1. The HL register pair will contain 4503<sub>16</sub>, and the BC register pair will contain 00FC<sub>16</sub>.

## CPL — COMPLEMENT THE ACCUMULATOR



CPL  
2F

Complement the contents of the Accumulator. No other register's contents are affected.

Suppose the Accumulator contains 3A<sub>16</sub>. After the instruction

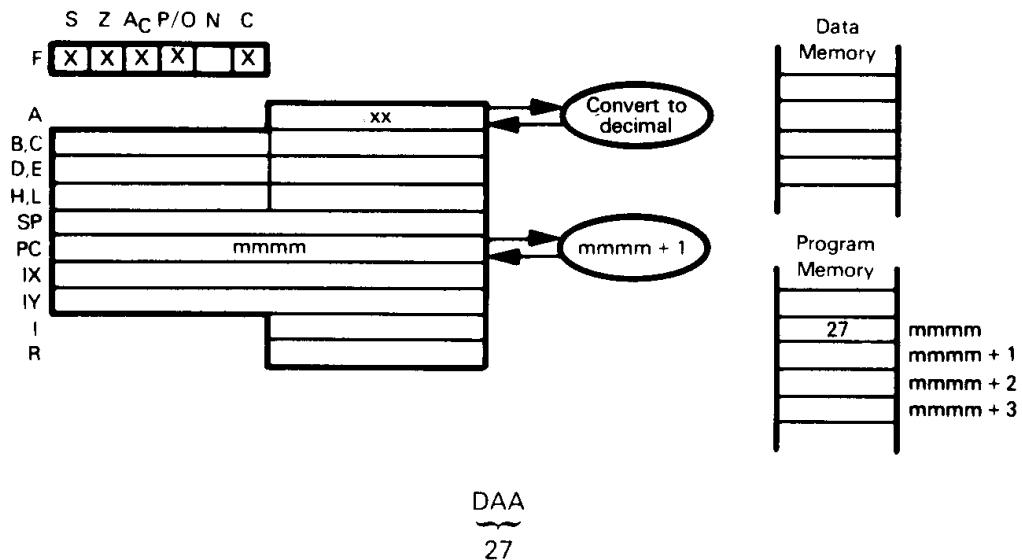
CPL

has executed, the Accumulator will contain C5<sub>16</sub>.

$$\begin{array}{ll} 3A = & 0011 \quad 1010 \\ \text{Complement} = & 1100 \quad 0101 \end{array}$$

This is a routine logical instruction. You need not use it for binary subtraction; there are special subtract instructions (SUB, SBC).

## DAA — DECIMAL ADJUST ACCUMULATOR



Convert the contents of the Accumulator to binary-coded decimal form. This instruction should only be used after adding or subtracting two BCD numbers; i.e., look upon ADD DAA or ADC DAA or INC DAA or SUB DAA or SBC DAA or DEC DAA or NEG DAA as compound, decimal arithmetic instructions which operate on BCD sources to generate BCD answers.

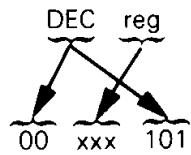
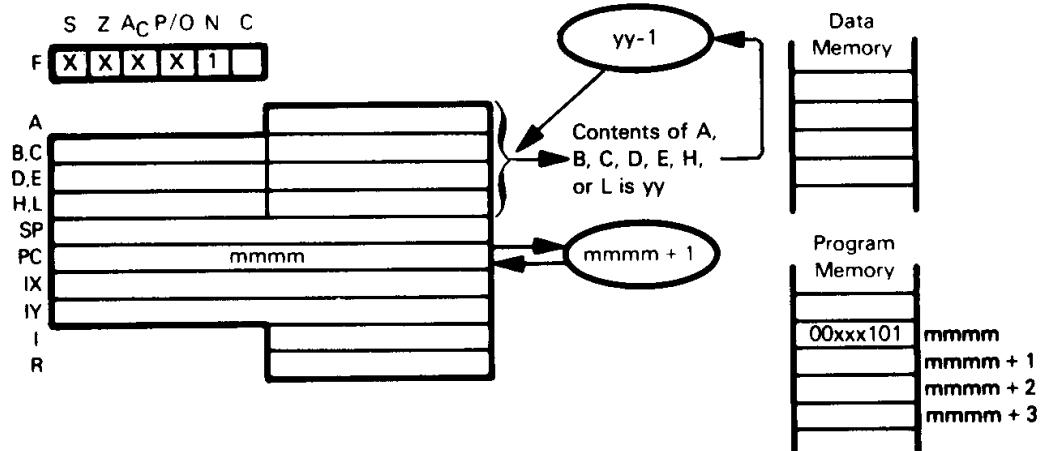
Suppose the Accumulator contains 39<sub>16</sub> and the B register contains 47<sub>16</sub>. After the instructions

```
ADD B
DAA
```

have executed, the Accumulator will contain 86<sub>16</sub>, not 80<sub>16</sub>.

Z80 CPU logic uses the values in the Carry and Auxiliary Carry, as well as the Accumulator contents, in the Decimal Adjust operation.

## DEC reg — DECREMENT REGISTER CONTENTS



|     |           |
|-----|-----------|
| 000 | for reg=B |
| 001 | for reg=C |
| 010 | for reg=D |
| 011 | for reg=E |
| 100 | for reg=H |
| 101 | for reg=L |
| 111 | for reg=A |

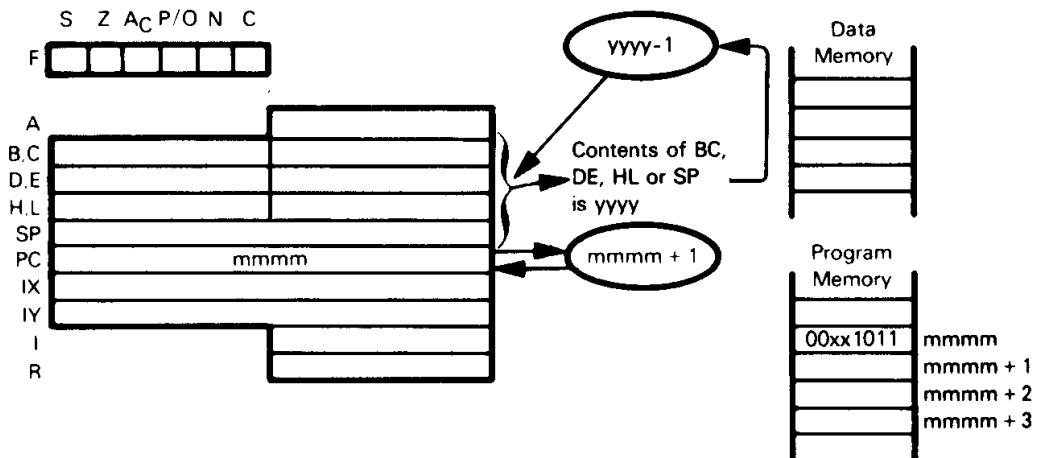
Subtract 1 from the contents of the specified register.

Suppose Register A contains 5016. After execution of

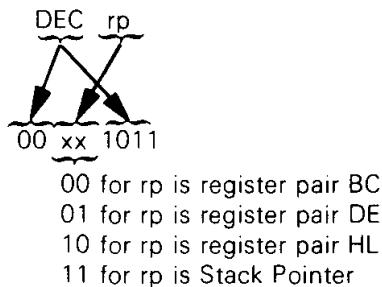
DEC A

Register A will contain 4F16.

**DEC rp — DECREMENT CONTENTS OF SPECIFIED REGISTER**  
**DEC IX PAIR**  
**DEC IY**



The illustration shows execution of DEC rp:



Subtract 1 from the 16-bit value contained in the specified register pair. No status flags are affected.

Suppose the H and L registers contain 2F00<sub>16</sub>. After the instruction

DEC HL

has executed, the H and L registers will contain 2EFF<sub>16</sub>.

DEC IX  
 DD 2B

Subtract 1 from the 16-bit value contained in the IX register.

DEC IY  
 FD 2B

Subtract 1 from the 16-bit value contained in the IY register.

Neither DEC rp, DEC IX nor DEC IY affects any of the status flags. This is a defect in the Z80 instruction set, inherited from the 8080. Whereas the DEC reg instruction is used in iterative instruction loops that use a counter with a value of 256 or less, the DEC rp (DEC IX or DEC IY) instruction must be used if the counter value is more than 256. Since the DEC rp instruction sets no status flags, other instructions must be added to simply

test for a zero result. This is a typical loop form:

```

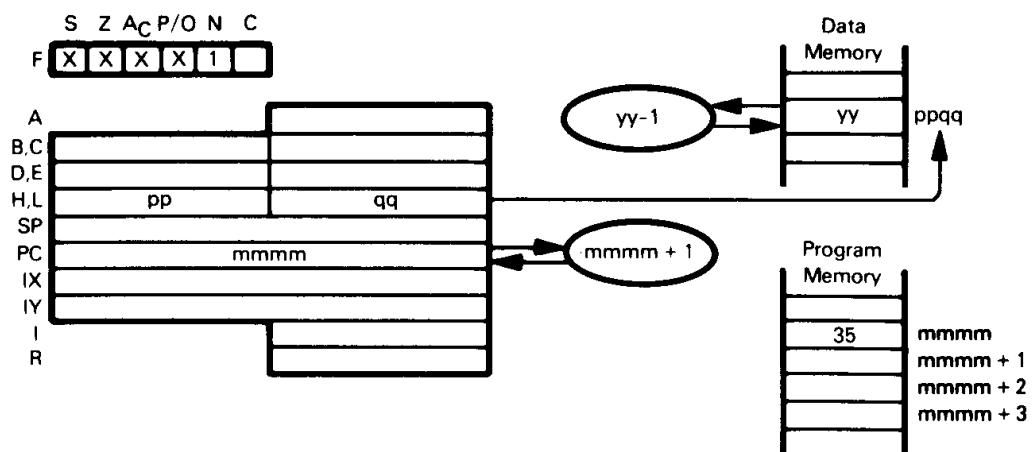
LD      DE,DATA ;LOAD INITIAL 16-BIT COUNTER VALUE
LOOP   -          ;FIRST INSTRUCTION OF LOOP
-
DEC    DE      ;DECREMENT COUNTER
LD     A,D    ;TO TEST FOR ZERO, MOVE D TO A
OR     E       ;THEN OR A WITH E
JP     NZ,LOOP ;RETURN IF NOT ZERO

```

### **DEC (HL) — DECREMENT MEMORY CONTENTS**

**DEC (IX+disp)**

**DEC (IY+disp)**



The illustration shows execution of DEC (HL):

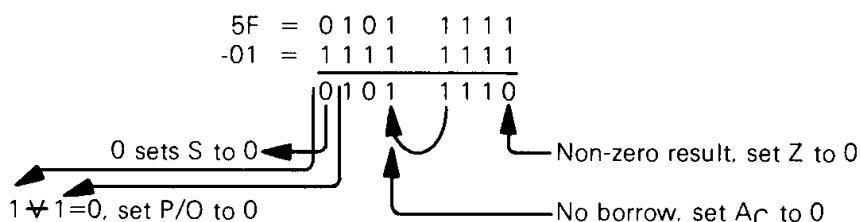
$\overbrace{\text{DEC (HL)}}$   
**35**

Subtract 1 from the contents of memory location (specified by the contents of the HL register pair).

Suppose  $ppqq=4500_{16}$ ,  $yy=5F_{16}$ . After execution of

**DEC (HL)**

memory location  $4500_{16}$  will contain  $5E_{16}$ .



Subtract instruction, set N to 1

DEC (IX+disp)

DD 35 d

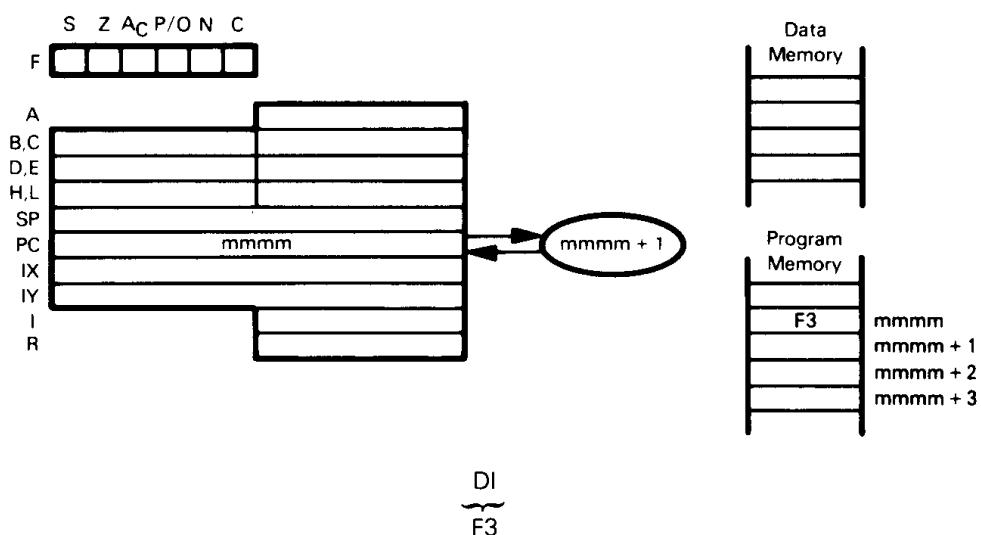
Subtract 1 from the contents of memory location (specified by the sum of the contents of the IX register and the displacement value d).

DEC (IY+disp)

FD 35 d

This instruction is identical to DEC (IX+disp), except that it uses the IY register instead of the IX register.

## DI — DISABLE INTERRUPTS

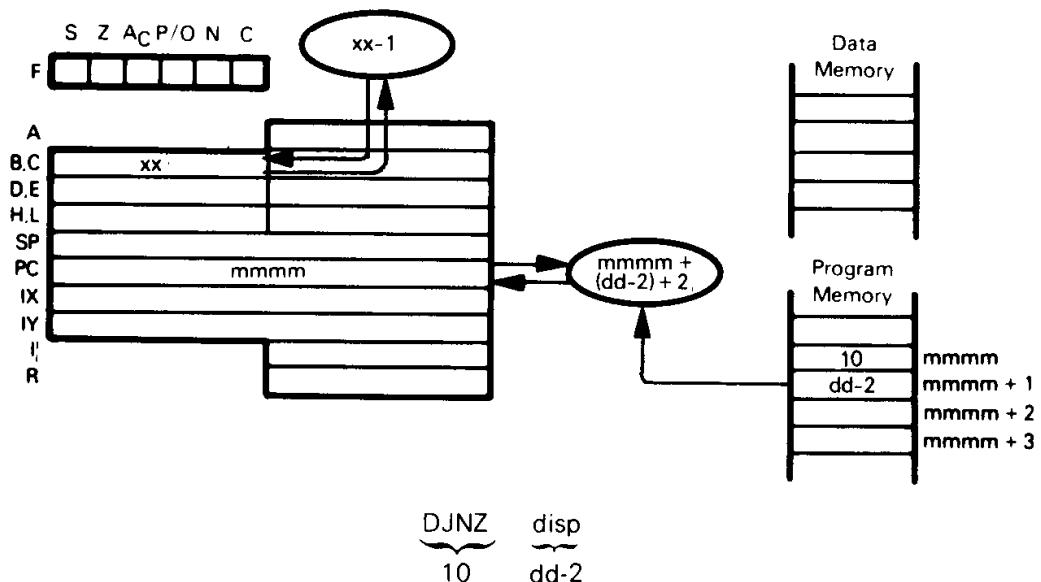


When this instruction is executed, the maskable interrupt request is disabled and the INT input to the CPU will be ignored. Remember that when an interrupt is acknowledged, the maskable interrupt is automatically disabled.

The maskable interrupt request remains disabled until it is subsequently enabled by an EI instruction.

No registers or flags are affected by this instruction.

**DJNZ disp — JUMP RELATIVE TO PRESENT  
CONTENTS OF PROGRAM COUNTER IF  
REG B IS NOT ZERO**

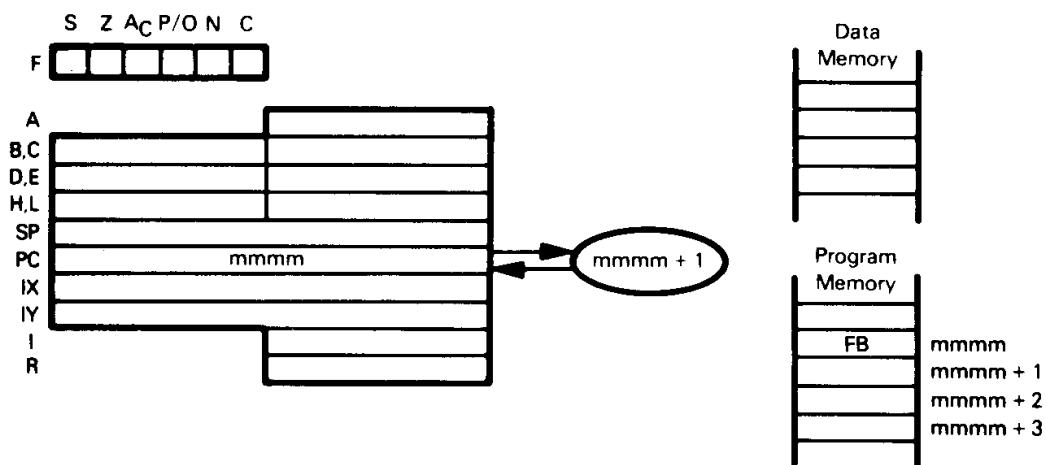


Decrement Register B. If remaining contents are not zero, add the contents of the DJNZ instruction object code second byte and 2 to the Program Counter. The jump is measured from the address of the instruction operation code, and has a range of -126 to +129 bytes. The Assembler automatically adjusts for the twice-incremented PC.

If the contents of B are zero after decrementing, the next sequential instruction is executed.

The DJNZ instruction is extremely useful for any program loop operation, since the one instruction replaces the typical "decrement-then-branch on condition" instruction sequence.

**EI — ENABLE INTERRUPTS**



EI  
FB

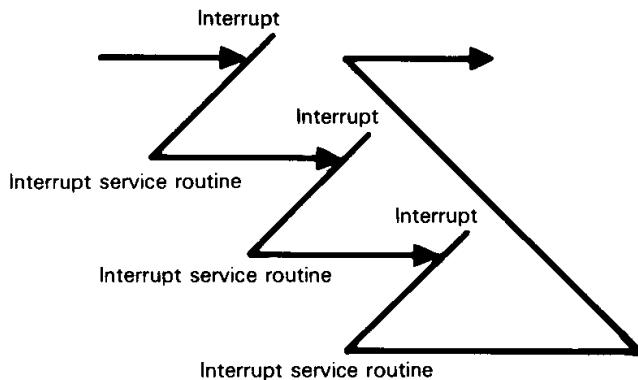
Execution of this instruction causes interrupts to be enabled, but not until one more instruction executes.

Most interrupt service routines end with the two instructions:

EI ;ENABLE INTERRUPTS  
RET ;RETURN TO INTERRUPTED PROGRAM

If interrupts are processed serially, then for the entire duration of the interrupt service routine all maskable interrupts are disabled — which means that in a multi-interrupt application there is a significant possibility for one or more interrupts to be pending when any interrupt service routine completes execution.

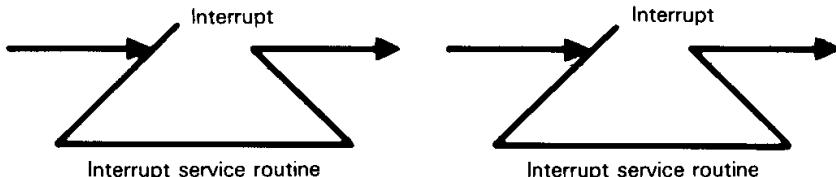
If interrupts were acknowledged as soon as the EI instructions had executed, then the Return instruction would not be executed. Under these circumstances, returns would stack up one on top of the other — and unnecessarily consume stack memory space. This may be illustrated as follows:



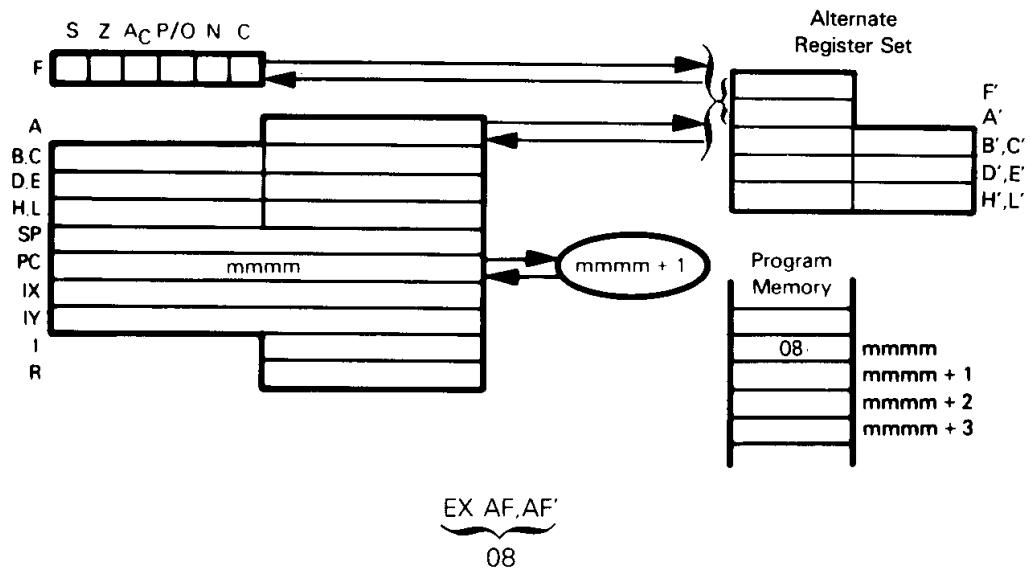
By inhibiting interrupts for one more instruction following execution of EI, the Z80 CPU ensures that the RET instruction gets executed in the sequence:

-  
-  
-  
EI ;ENABLE INTERRUPTS  
RET ;RETURN FROM INTERRUPT

It is not uncommon for interrupts to be kept disabled while an interrupt service routine is executing. Interrupts are processed serially:



## EX AF,AF' — EXCHANGE PROGRAM STATUS AND ALTERNATE PROGRAM STATUS



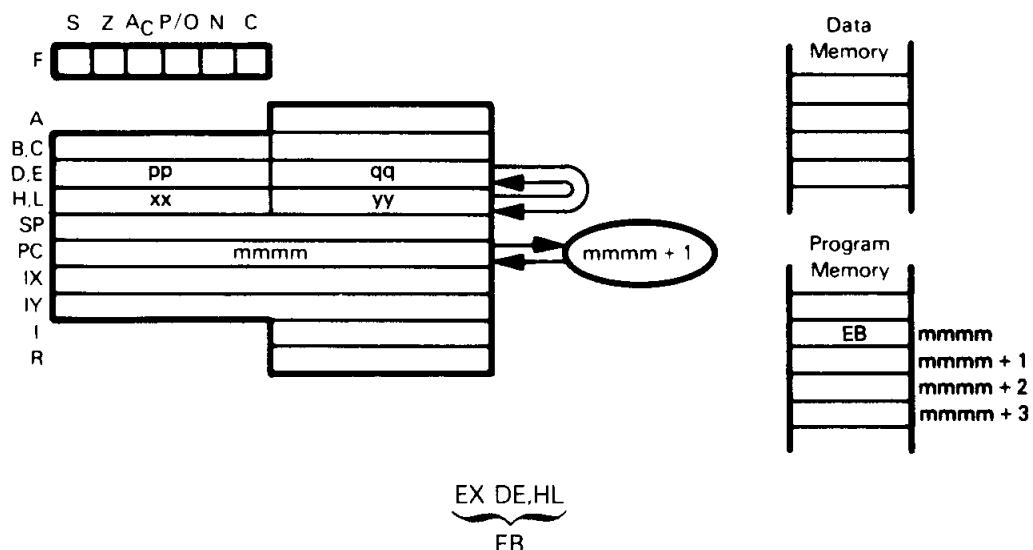
The two-byte contents of register pairs AF and A'F' are exchanged.

Suppose AF contains 4F9916 and A'F' contains 10AA16. After execution of

EX AF,AF'

AF will contain 10AA16 and AF' will contain 4F9916.

## EX DE,HL — EXCHANGE DE AND HL CONTENTS



The D and E registers' contents are swapped with the H and L registers' contents.

Suppose pp=0316, qq=2A16, xx=4116 and yy=FC16. After the instruction

**EX DE,HL**

has executed, H will contain 0316, L will contain 2A16, D will contain 4116 and E will contain FC16.

The two instructions:

EX DE,HL  
LD A,(HL)

are equivalent to:

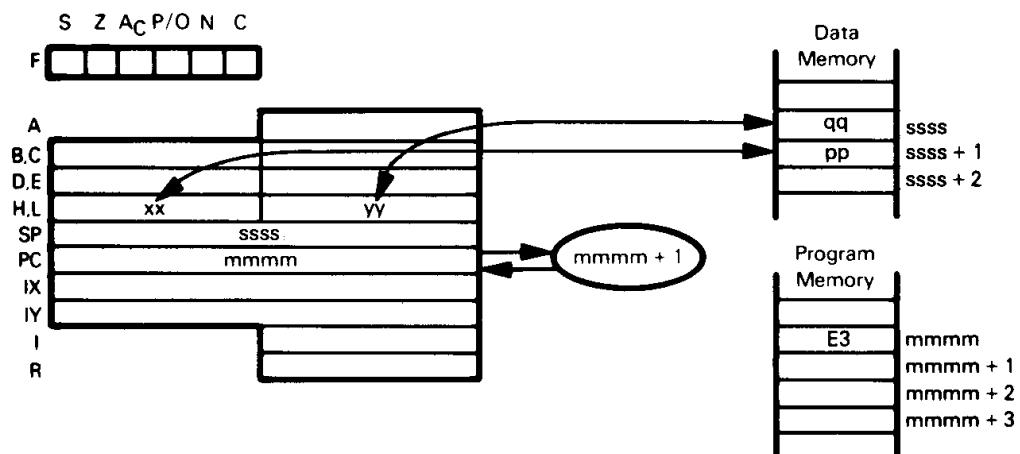
LD A,(DE)

but if you want to load data addressed by the D and E register into the B register,

EX DE,HL  
LD B,(HL)

has no single instruction equivalent.

**EX (SP),HL — EXCHANGE CONTENTS OF REGISTER AND  
EX (SP),IX      TOP OF STACK  
EX (SP),IY**



The illustration shows execution of EX (SP),HL.

$\underbrace{\text{EX (SP),HL}}_{\text{E3}}$

Exchange the contents of the L register with the top stack byte. Exchange the contents of the H register with the byte below the stack top.

Suppose  $xx=2116$ ,  $yy=FA16$ ,  $pp=3A16$ ,  $qq=E216$ . After the instruction

$\text{EX (SP),HL}$

has executed, H will contain 3A16, L will contain E216 and the two top stack bytes will contain FA16 and 2116 respectively.

The EX (SP),HL instruction is used to access and manipulate data at the top of the stack.

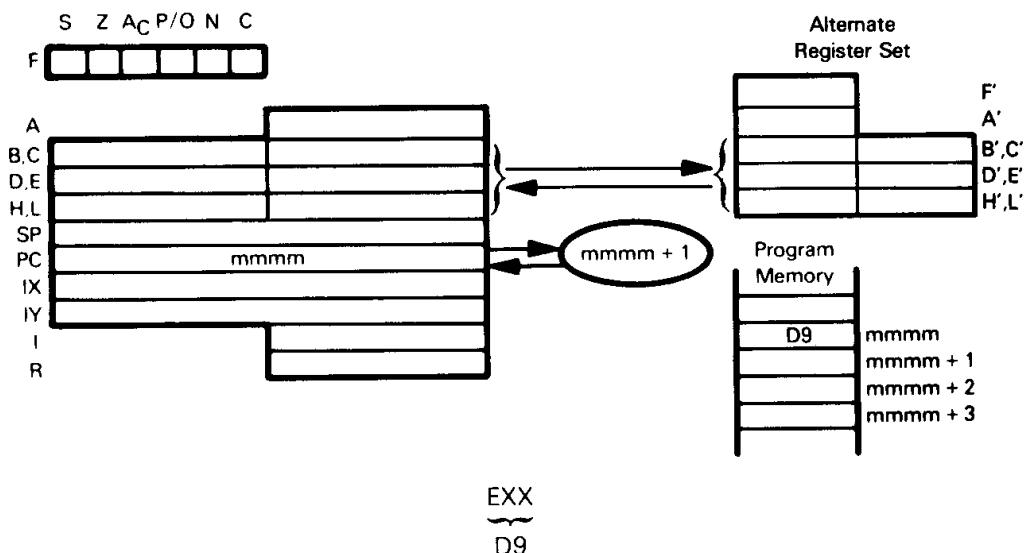
$\underbrace{\text{EX (SP),IX}}_{\text{DD E3}}$

Exchange the contents of the IX register's low-order byte with the top stack byte. Exchange the IX register's high-order byte with the byte below the stack top.

$\underbrace{\text{EX (SP),IY}}_{\text{FD E3}}$

This instruction is identical to EX (SP),IX, but uses the IY register instead of the IX register.

## EXX — EXCHANGE REGISTER PAIRS AND ALTERNATE REGISTER PAIRS



The contents of register pairs BC, DE and HL are swapped with the contents of register pairs B'C', D'E', and H'L'.

Suppose register pairs BC, DE and HL contain 490116, 5F0016 and 725116 respectively, and register pairs B'C', D'E', H'L' contain 000016, 10FF16 and 333316 respectively. After the execution of

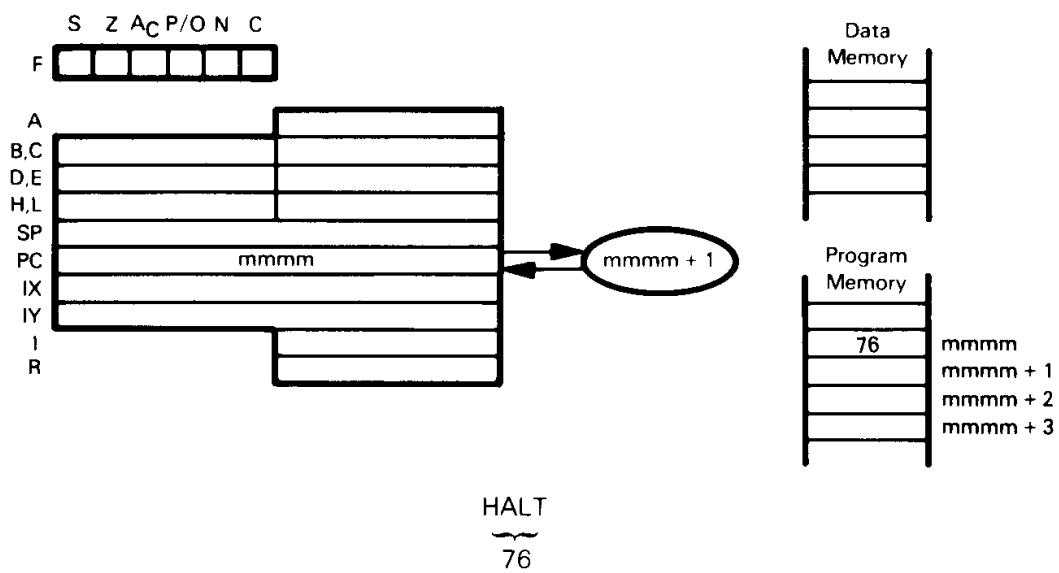
EXX

the registers will have the following contents:

BC: 0000<sub>16</sub>; DE: 10FF<sub>16</sub>; HL: 3333<sub>16</sub>  
B'C': 490116; D'E': 5F0016; H'L': 725116

This instruction can be used to exchange register banks to provide very fast interrupt response times.

## HALT

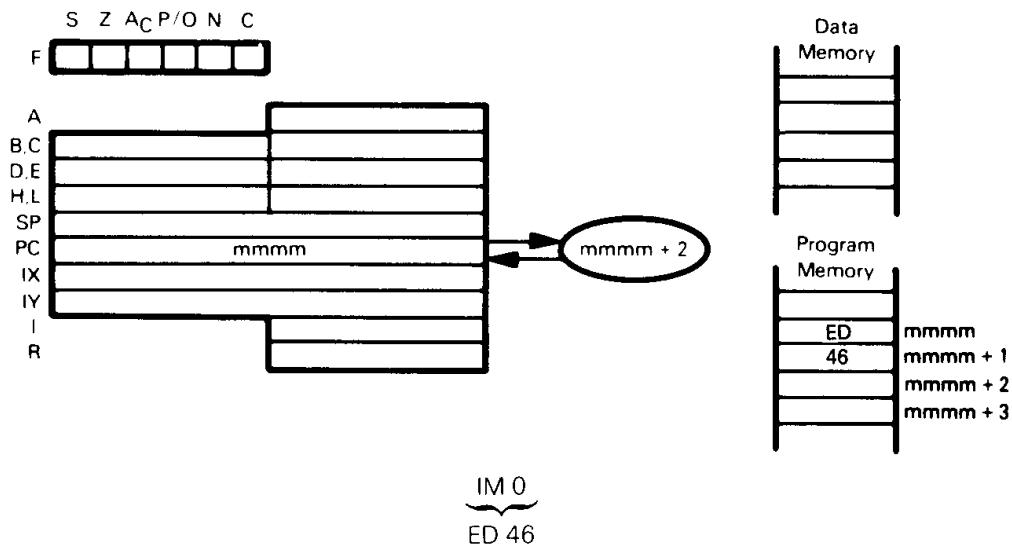


HALT

76

When the HALT instruction is executed, program execution ceases. The CPU requires an interrupt or a reset to restart execution. No registers or statuses are affected; however, memory refresh logic continues to operate.

## IM 0 — INTERRUPT MODE 0



This instruction places the CPU in interrupt mode 0. In this mode, the interrupting device will place an instruction on the Data Bus and the CPU will then execute that instruction. No registers or statuses are affected.

## IM 1 — INTERRUPT MODE 1

IM 1  
ED 56

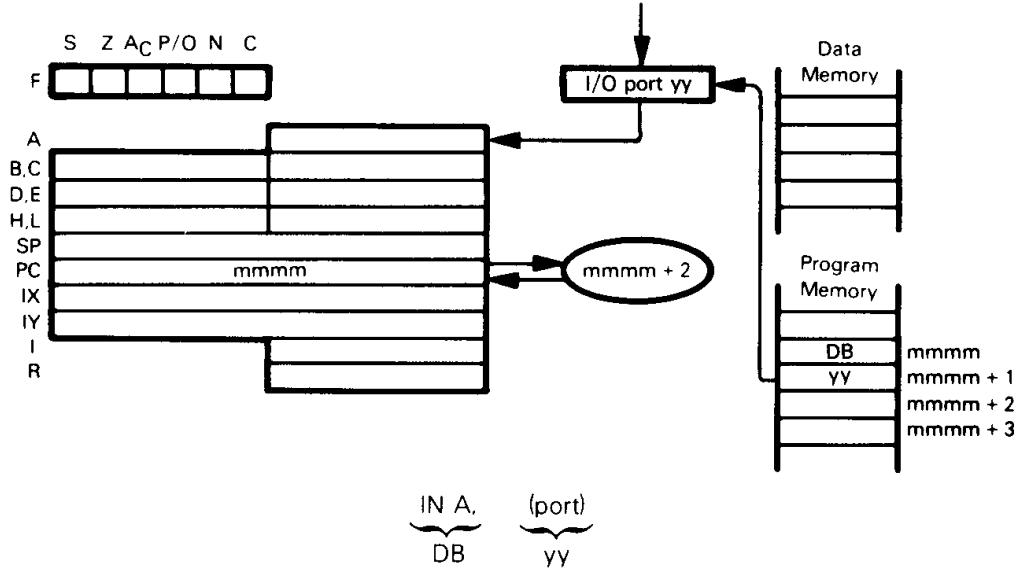
This instruction places the CPU in interrupt mode 1. In this mode, the CPU responds to an interrupt by executing a restart (RST) to location 003816.

## IM 2 — INTERRUPT MODE 2

IM 2  
ED 5E

This instruction places the CPU in interrupt mode 2. In this mode, the CPU performs an indirect call to any specified location in memory. A 16-bit address is formed using the contents of the Interrupt Vector (I) register for the upper eight bits, while the lower eight bits are supplied by the interrupting device. Refer to Chapter 12 for a full description of interrupt modes. No registers or statuses are affected by this instruction.

## IN A,(port) — INPUT TO ACCUMULATOR



Load a byte of data into the Accumulator from the I/O port (identified by the second IN instruction object code byte).

Suppose 3616 is held in the buffer of I/O port 1A16. After the instruction

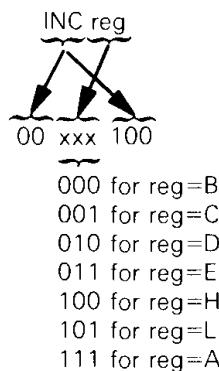
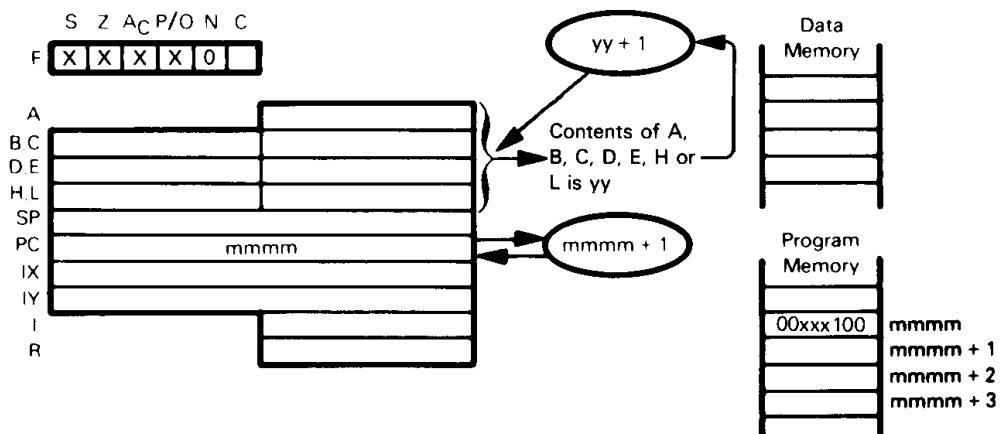
IN A,(1AH)

has executed, the Accumulator will contain 3616.

The IN instruction does not affect any statuses.

Use of the IN instruction is very hardware dependent. Valid I/O port addresses are determined by the way in which I/O logic has been implemented. It is also possible to design a microcomputer system that accesses external logic using memory reference instructions with specific memory addresses.

## INC reg — INCREMENT REGISTER CONTENTS



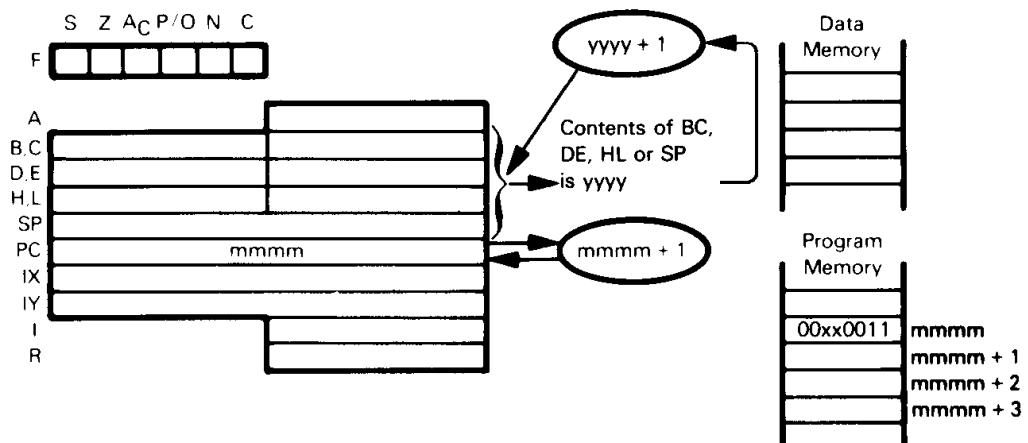
Add 1 to the contents of the specified register.

Suppose Register E contains A816. After execution of

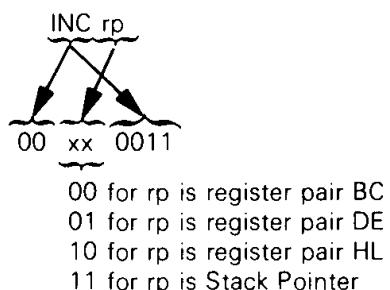
INC E

Register E will contain A916.

**INC rp — INCREMENT CONTENTS OF SPECIFIED REGISTER PAIR**  
**INC IX**  
**INC IY**



The illustration shows execution of INC rp:



Add 1 to the 16-bit value contained in the specified register pair. No status flags are affected.

Suppose the D and E registers contain 2F7A<sub>16</sub>. After the instruction

INC DE

has executed, the D and E registers will contain 2F7B<sub>16</sub>.

INC IX  
 DD 23

Add 1 to the 16-bit value contained in the IX register.

INC IY  
 FD 23

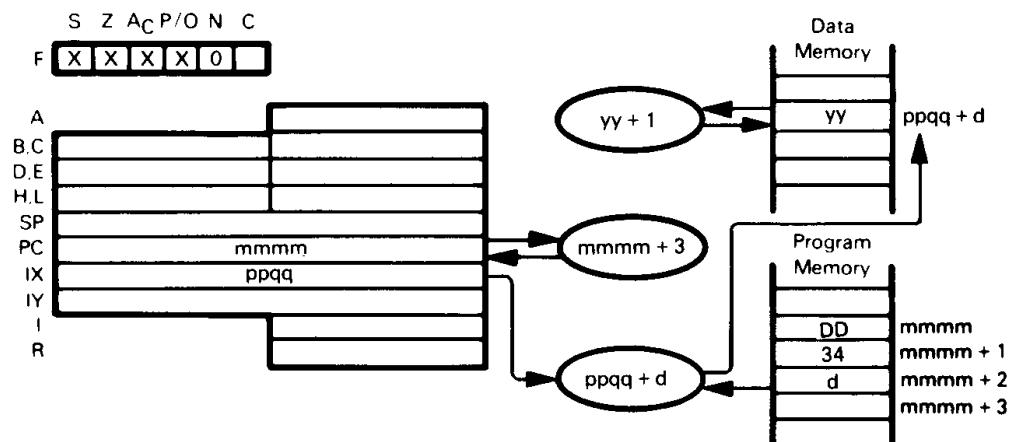
Add 1 to the 16-bit value contained in the IY register.

Just like the DEC rp, DEC IX and DEC IY, neither INC rp, INC IX nor INC IY affects any status flags. This is a defect in the Z80 instruction set inherited from the 8080.

## INC (HL) — INCREMENT MEMORY CONTENTS

INC (IX+disp)

INC (IY+disp)



The illustration shows execution of INC (IX+disp):

INC (IX+disp)

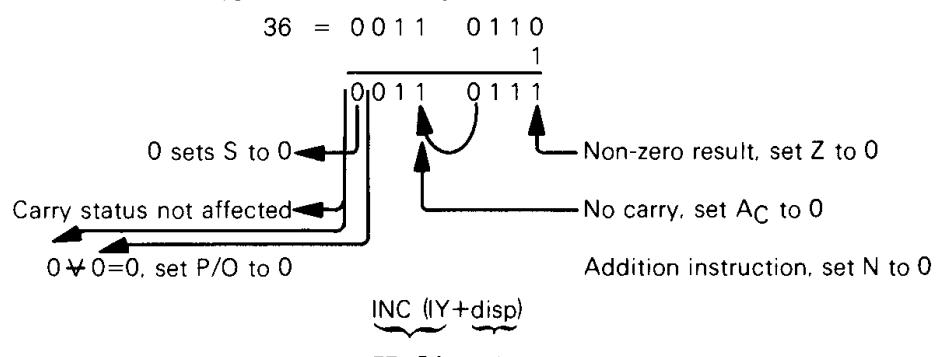
DD 34 d

Add 1 to the contents of memory location (specified by the sum of the contents of Register IX and the displacement value d).

Suppose ppqq=4000<sub>16</sub> and memory location 400F<sub>16</sub> contains 36<sub>16</sub>. After execution of the instruction

INC (IX+0FH)

memory location 400F<sub>16</sub> will contain 37<sub>16</sub>.

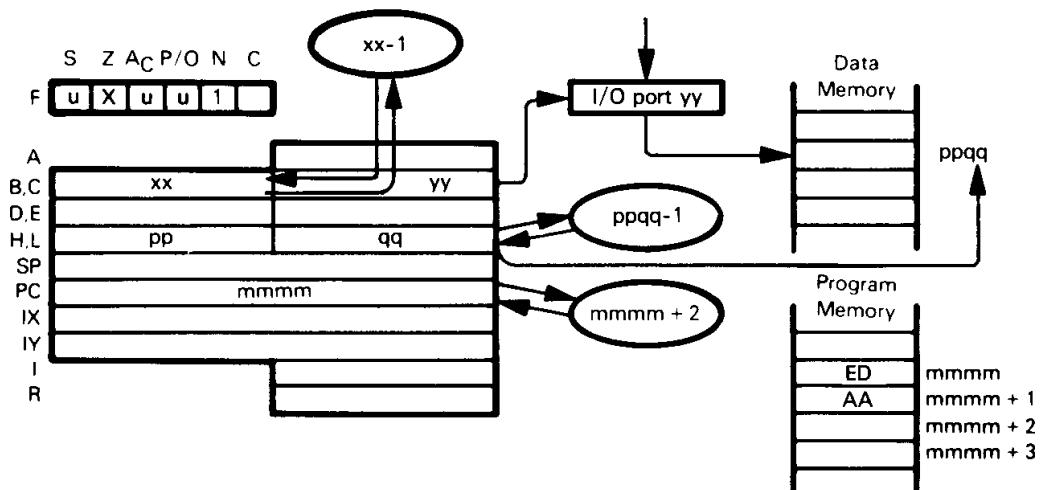


This instruction is identical to INC (IX+disp), except that it uses the IY register instead of the IX register.

INC (HL)  
34

Add 1 to the contents of memory location (specified by the contents of the HL register pair).

## IND — INPUT TO MEMORY AND DECREMENT POINTER



IND  
ED AA

Input from I/O port (addressed by Register C) to memory location (specified by HL). Decrement Registers B and HL.

Suppose  $xx=0516$ ,  $yy=1516$ ,  $ppqq=240016$ , and  $1916$  is held in the buffer of I/O port  $1516$ . After the instruction

IND

has executed, memory location  $240016$  will contain  $1916$ . The B register will contain  $0416$  and the HL register pair  $23FF16$ .

## INDR — INPUT TO MEMORY AND DECREMENT POINTER UNTIL BYTE COUNTER IS ZERO

INDR  
ED BA

INDR is identical to IND, but is repeated until Register B=0.

Suppose Register B contains  $0316$ , Register C contains  $1516$ , and HL contains  $240016$ . The following sequence of bytes is available at I/O port  $1516$ :

1716, 5916 and AE16

After the execution of

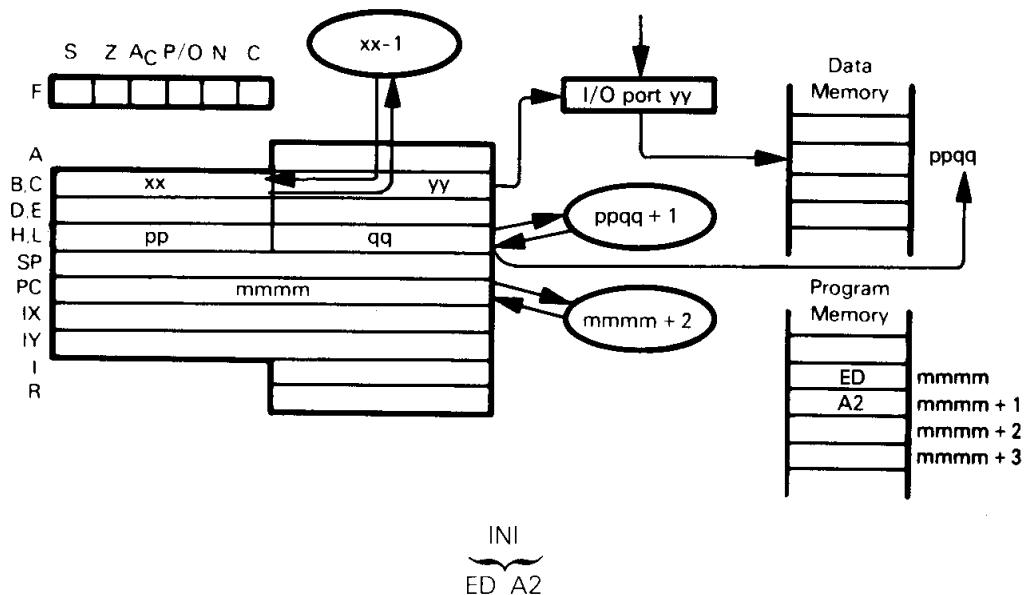
INDR

the HL register pair will contain  $23FD16$  and Register B will contain zero, and memory locations will have contents as follows:

| Location | Contents |
|----------|----------|
| 2400     | 1716     |
| 23FF     | 5916     |
| 23FE     | AE16     |

This instruction is extremely useful for loading blocks of data from an input device into memory.

## INI — INPUT TO MEMORY AND INCREMENT POINTER



Input from I/O port (addressed by Register C) to memory location (specified by HL). Decrement Register B; increment register pair HL.

Suppose  $xx=0516$ ,  $yy=1516$ ,  $ppqq=240016$ , and  $1916$  is held in the buffer of I/O port  $1516$ .

After the instruction

INI

has executed, memory location  $240016$  will contain  $1916$ . The B register will contain  $0416$  and the HL register pair  $240116$ .

## INIR — INPUT TO MEMORY AND INCREMENT POINTER UNTIL BYTE COUNTER IS ZERO

INIR  
ED B2

INIR is identical to INI, but is repeated until Register B=0.

Suppose Register B contains  $0316$ , Register C contains  $1516$ , and HL contains  $240016$ . The following sequence of bytes is available at I/O port  $1516$ :

1716, 5916 and AE16

After the execution of

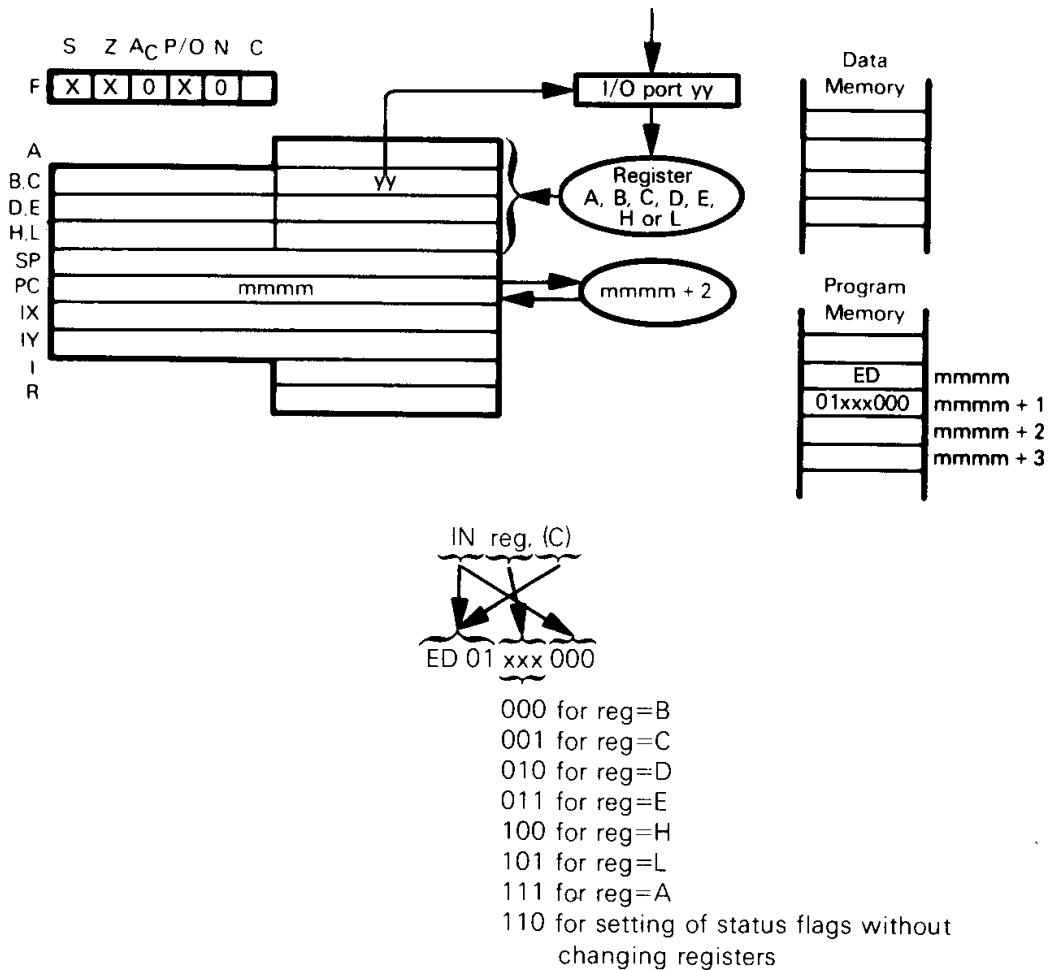
INIR

the HL register pair will contain  $240316$  and Register B will contain zero, and memory locations will have contents as follows:

| Location | Contents |
|----------|----------|
| 2400     | 1716     |
| 2401     | 5916     |
| 2402     | AE16     |

This instruction is extremely useful for loading blocks of data from a device into memory.

## IN reg,(C) — INPUT TO REGISTER



Load a byte of data into the specified register (reg) from the I/O port (identified by the contents of the C register).

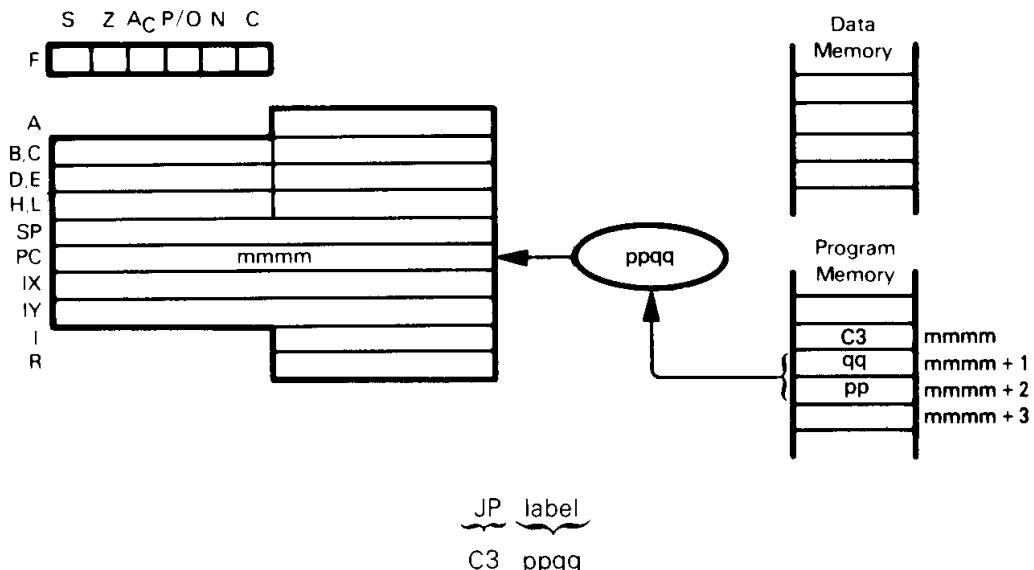
Suppose 42<sub>16</sub> is held in the buffer of I/O port 36<sub>16</sub>, and Register C contains 36<sub>16</sub>. After the instruction

IN D,(C)

has executed, the D register will contain 42<sub>16</sub>.

During the execution of the instruction, the contents of Register B are placed on the top half of the Address Bus, making it possible to extend the number of addressable I/O ports.

## JP label — JUMP TO THE INSTRUCTION IDENTIFIED IN THE OPERAND



Load the contents of the Jump instruction object code second and third bytes into the Program Counter; this becomes the memory address for the next instruction to be executed. The previous Program Counter contents are lost.

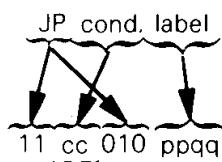
In the following sequence:

JP           NEXT  
AND        7FH

NEXT       CPL

The CPL instruction will be executed after the JP instruction. The AND instruction will never be executed, unless a Jump instruction somewhere else in the instruction sequence jumps to this instruction.

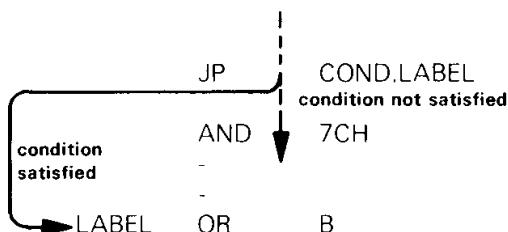
**JP condition,label — JUMP TO ADDRESS IDENTIFIED IN THE OPERAND IF CONDITION IS SATISIFIED**



|     | Condition       | Relevant Flag |
|-----|-----------------|---------------|
| 000 | NZ Non-Zero     | Z             |
| 001 | Z Zero          | Z             |
| 010 | NC No Carry     | C             |
| 011 | C Carry         | C             |
| 100 | PO Parity Odd   | P/O           |
| 101 | PE Parity Even  | P/O           |
| 110 | P Sign Positive | S             |
| 111 | M Sign Negative | S             |

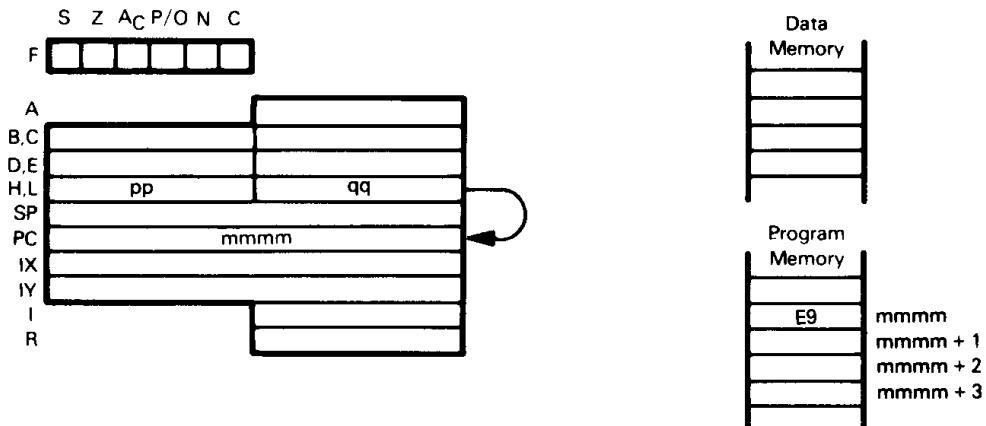
This instruction is identical to the JP instruction, except that the jump will be performed only if the condition is satisfied; otherwise, the instruction sequentially following the JP condition instruction will be executed.

Consider the instruction sequence



After the JP cond,label instruction has executed, if the condition is satisfied then the OR instruction will be executed. If the condition is not satisfied, the AND instruction, being the next sequential instruction, is executed.

**JP (HL) — JUMP TO ADDRESS SPECIFIED BY CONTENTS  
OF 16-BIT REGISTER**  
**JP (IX)**



The illustration shows execution of JP (HL):

JP (HL)  
E9

The contents of the HL register pair are moved to the Program Counter; therefore, an implied addressing jump is performed.

The instruction sequence

|    |        |
|----|--------|
| LD | H,ADDR |
| JP | (HL)   |

has exactly the same net effect as the single instruction

|    |      |
|----|------|
| JP | ADDR |
|----|------|

Both specify that the instruction with label ADDR is to be executed next.

The JP (HL) instruction is useful when you want to increment a return address for a subroutine that has multiple returns.

Consider the following call to subroutine SUB:

|      |     |                  |
|------|-----|------------------|
| CALL | SUB | ;CALL SUBROUTINE |
| JP   | ERR | ;ERROR RETURN    |
|      |     | :GOOD RETURN     |

Using RET to return from SUB would return execution of JP ERR; therefore, if SUB executes without detecting error conditions, return as follows:

|     |      |                           |
|-----|------|---------------------------|
| POP | HL   | ;POP RETURN ADDRESS TO HL |
| INC | HL   | ;ADD 3 TO RETURN ADDRESS  |
| INC | HL   |                           |
| INC | HL   |                           |
| JP  | (HL) | ;RETURN                   |

JP (IX)  
DD E9

This instruction is identical to the JP (HL) instruction, except that it uses the IX register

instead of the HL register pair.

JP (IY)  
FD E9

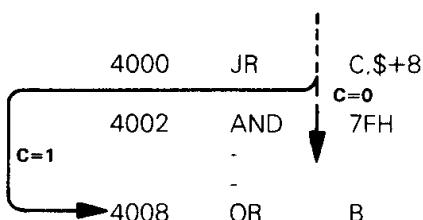
This instruction is identical to the JP (HL) instruction, except that it uses the IY register instead of the HL register pair.

### **JR C,disp — JUMP RELATIVE TO CONTENTS OF PROGRAM COUNTER IF CARRY IS SET**

JR C, disp  
38 dd-2

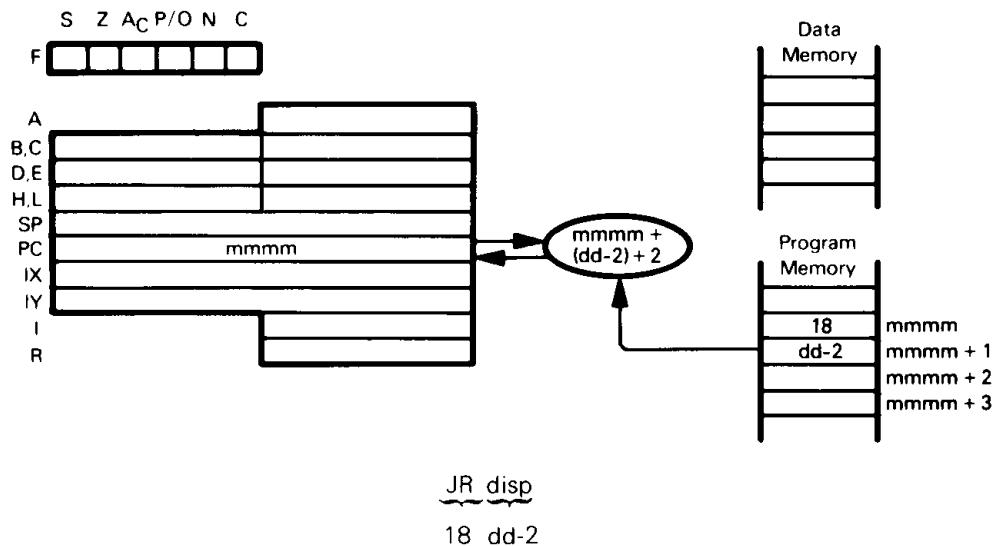
This instruction is identical to the JR disp instruction, except that the jump is only executed if the Carry status equals 1; otherwise, the next instruction is executed.

In the following instruction sequence:



After the JR C,\$+8 instruction, the OR instruction is executed if the Carry status equals 1. The AND instruction is executed if the Carry status equals 0.

## JR disp — JUMP RELATIVE TO PRESENT CONTENTS OF PROGRAM COUNTER



JR disp

18 dd-2

Add the contents of the JR instruction object code second byte, the contents of the Program Counter, and 2. Load the sum into the Program Counter. The jump is measured from the address of the instruction operation code, and has a range of -126 to +129 bytes. The Assembler automatically adjusts for the twice-incremented PC.

The following assembly language statement is used to jump four steps forward from address 4000<sub>16</sub>.

JR \$+4

Result of this instruction is shown below:

| Location | Instruction |
|----------|-------------|
| 4000     | 18          |
| 4001     | 02          |
| 4002     | -           |
| 4003     | -           |
| 4004     | -           |

← new PC value

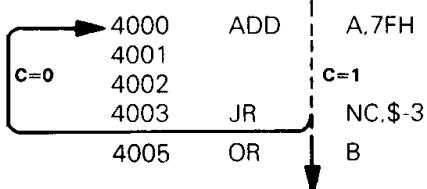
## **JR NC,disp — JUMP RELATIVE TO CONTENTS OF PROGRAM COUNTER IF CARRY FLAG IS RESET**

JR NC,disp

30 dd-2

This instruction is identical to the JR disp instruction, except that the jump is only executed if the Carry status equals 0; otherwise, the next instruction is executed.

In the following instruction sequence:



After the JR NC,\$-3 instruction, the OR instruction is executed if the Carry status equals 1. The ADD instruction is executed if the Carry status equals 0.

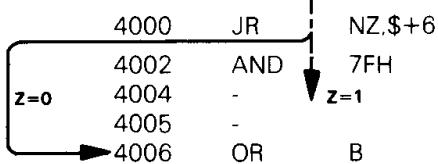
## **JR NZ,disp — JUMP RELATIVE TO CONTENTS OF PROGRAM COUNTER IF ZERO FLAG IS RESET**

JR NZ,disp

20 dd-2

This instruction is identical to the JR disp instruction, except that the jump is only executed if the Zero status equals 0; otherwise, the next instruction is executed.

In the following instruction sequence:



After the JR NZ,\$+6 instruction, the OR instruction is executed if the Zero status equals 0. The AND instruction is executed if the Zero status equals 1.

## **JR Z,disp — JUMP RELATIVE TO CONTENTS OF PROGRAM COUNTER IF ZERO FLAG IS SET**

JR Z,disp  
28 dd-2

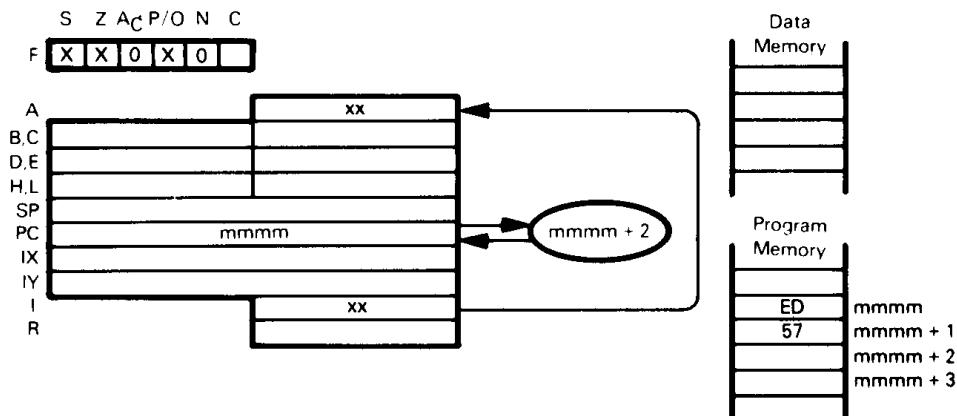
This instruction is identical to the JR disp instruction, except that the jump is only executed if the Zero status equals 1; otherwise, the next instruction is executed.

In the following instruction sequence:

|            |     |            |
|------------|-----|------------|
| 4000       | JR  | Z,\$+6     |
| 4002       | AND | 7FH        |
| <b>z=1</b> | -   | <b>z=0</b> |
| 4005       | -   |            |
| 4006       | OR  | B          |

After the JR Z,\$+6 instruction, the OR instruction is executed if the Zero status equals 1. The AND instruction is executed if the Zero status equals 0.

## **LD A,I — MOVE CONTENTS OF INTERRUPT VECTOR OR LD A,R — REFRESH REGISTER TO ACCUMULATOR**



The illustration shows execution of LD A,I:

LD A,I  
ED 57

Move the contents of the Interrupt Vector register to the Accumulator, and reflect interrupt enable status in Parity/Overflow flag.

Suppose the Interrupt Vector register contains 7F<sub>16</sub>, and interrupts are disabled. After execution of

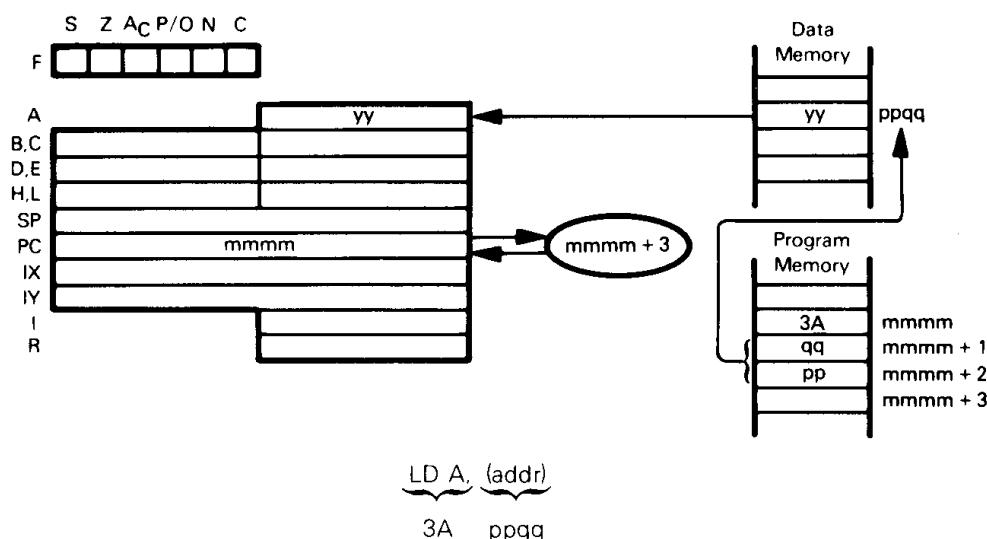
LD A,I

Register A will contain 7F<sub>16</sub>, and P/O will be 0.

LD A,R  
ED 5F

Move the contents of the Refresh register to the Accumulator. The value of the interrupt flip-flop will appear in the Parity/Overflow flag.

## LD A,(addr) — LOAD ACCUMULATOR FROM MEMORY USING DIRECT ADDRESSING



Load the contents of the memory byte (addressed directly by the second and third bytes of the LD A,(addr) instruction object code) into the Accumulator. Suppose memory byte 084A<sub>16</sub> contains 2016. After the instruction

```
label EQU 084AH
-
-
-
LD A.(label)
```

has executed, the Accumulator will contain 2016.

Remember that EQU is an assembler directive rather than an instruction; it tells the Assembler to use the 16-bit value 084A<sub>16</sub> wherever the label appears.

The instruction

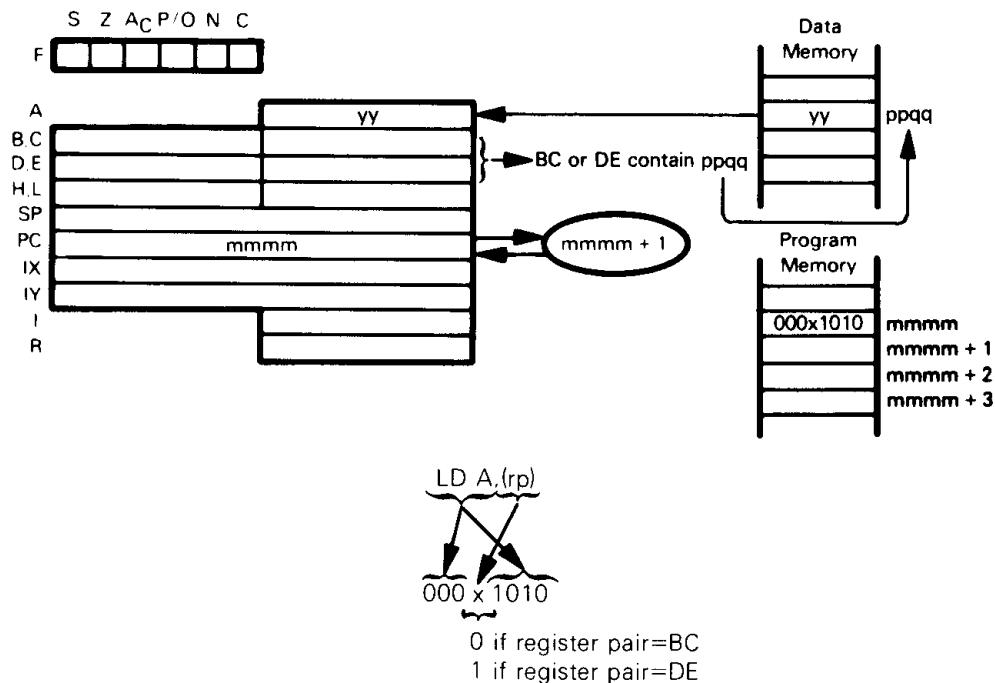
```
LD A.(label)
```

is equivalent to the two instructions

```
LD HL,label
LD A.(HL)
```

When you are loading a single value from memory, the LD A,(label) instruction is preferred; it uses one instruction and three object program bytes to do what the LD HL,label, LD A,(HL) combination does in two instructions and four object program bytes. Also, the LD HL,label, LD A,(HL) combination uses the H and L registers, which LD A,(label) does not.

## LD A,(rp) — LOAD ACCUMULATOR FROM MEMORY LOCATION ADDRESSED BY REGISTER PAIR



Load the contents of the memory byte (addressed by the BC or DE register pair) into the Accumulator.

Suppose the B register contains 0816, the C register contains 4A16, and memory byte 084A16 contains 3A16. After the instruction

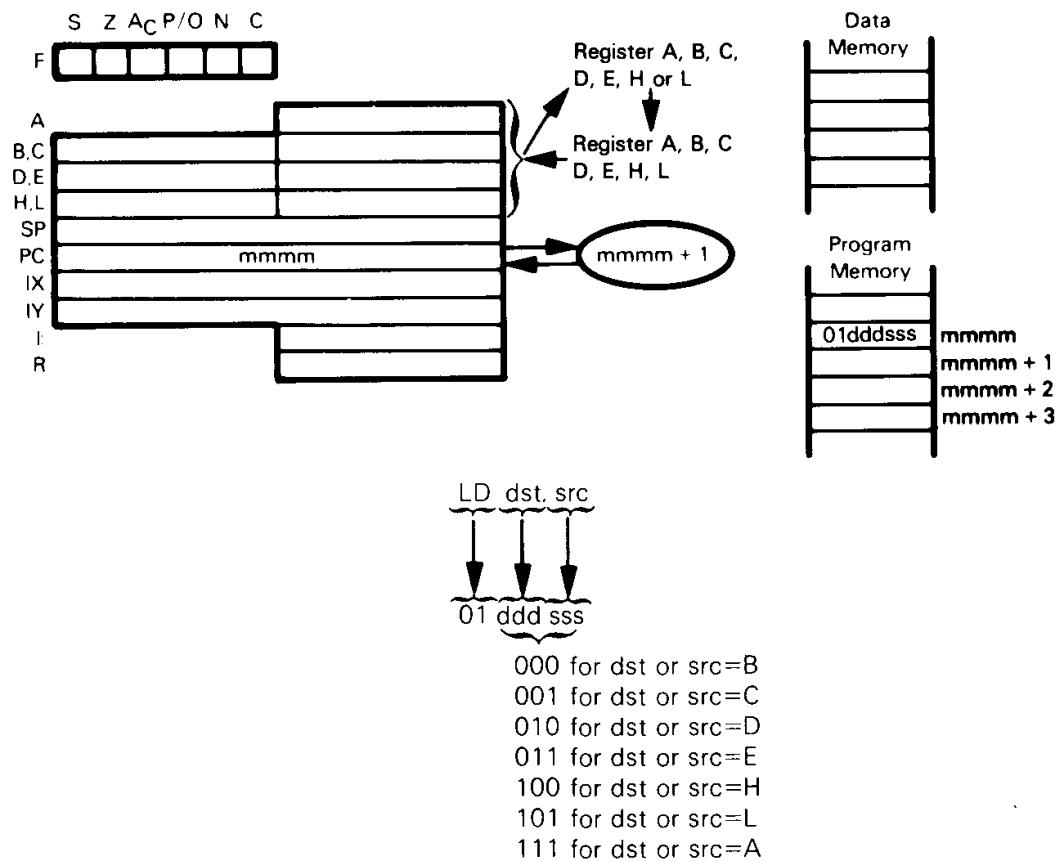
LD A,(BC)

has executed, the Accumulator will contain 3A16.

Normally, the LD A,(rp) and LD rp,data will be used together, since the LD rp,data instruction loads a 16-bit address into the BC or DE registers as follows:

|    |          |
|----|----------|
| LD | BC,084AH |
| LD | A,(BC)   |

## LD dst,src — MOVE CONTENTS OF SOURCE REGISTER TO DESTINATION REGISTER



The contents of any designated register are loaded into any other register.

For example:

LD A,B

loads the contents of Register B into Register A.

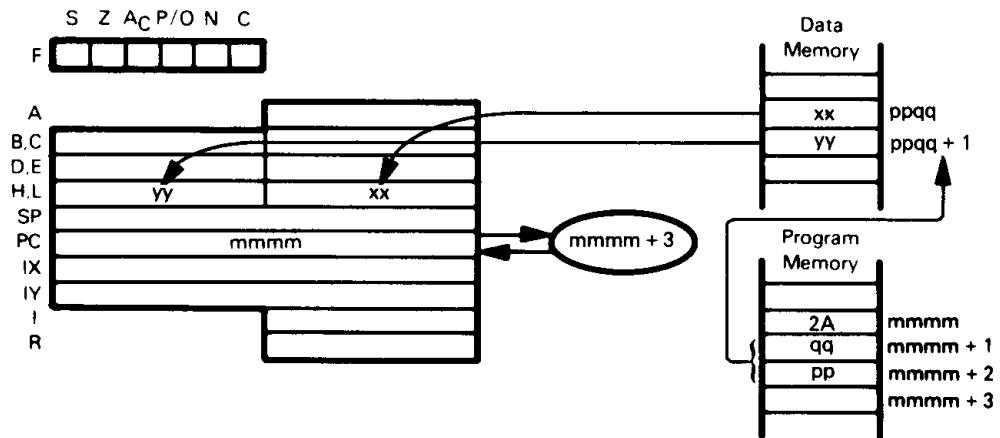
LD L,D

loads the contents of Register D into Register L.

LD C,C

does nothing, since the C register has been specified as both the source and the destination.

**LD HL,(addr) — LOAD REGISTER PAIR OR INDEX REGISTER  
FROM MEMORY USING DIRECT ADDRESSING**  
**LD rp,(addr)**  
**LD IX,(addr)**  
**LD IY,(addr)**



The illustration shows execution of LD HL(ppqq):

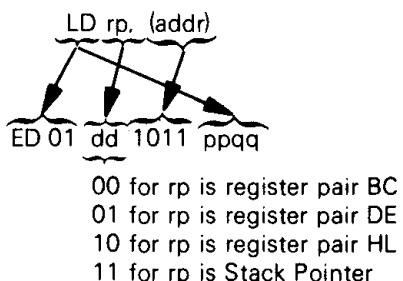
LD HL,addr  
2A ppqq

Load the HL register pair from directly addressed memory location.

Suppose memory location 400416 contains AD16 and memory location 400516 contains 1216. After the instruction

LD HL,(4004H)

has executed, the HL register pair will contain 12AD16.



Load register pair from directly addressed memory.

Suppose memory location 49FF16 contains BE16 and memory location 4A0016 contains 3316. After the instruction

LD DE,(49FFH)

has executed, the DE register pair will contain 33BE16.

LD IX,addr  
DD 2A ppqq

Load IX register from directly addressed memory.

Suppose memory location D11116 contains FF16 and memory location D11216 contains 5616. After the instruction

$\underbrace{\text{LD IX},(\text{D}111\text{H})}_{\text{FD } 2\text{A ppqq}}$

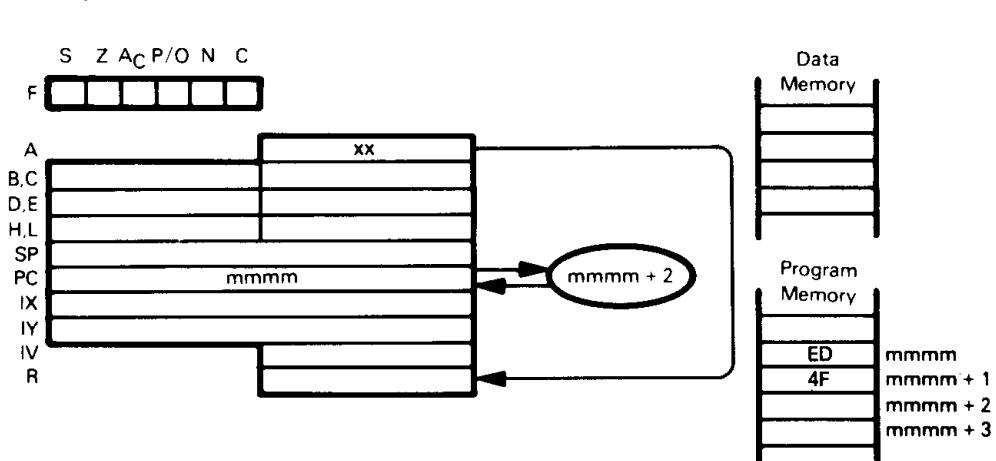
has executed, the IX register will contain 56FF16.

$\underbrace{\text{LD IY},(\text{addr})}_{\text{FD } 2\text{A ppqq}}$

Load IY register from directly addressed memory.

Affects IY register instead of IX. Otherwise identical to LD IX(addr).

### **LD I,A — LOAD INTERRUPT VECTOR OR REFRESH LD R,A REGISTER FROM ACCUMULATOR**



The illustration shows execution of LD R,A:

$\underbrace{\text{LD R,A}}_{\text{ED } 4\text{F}}$

Load Refresh register from Accumulator.

Suppose the Accumulator contains 7F16. After the instruction

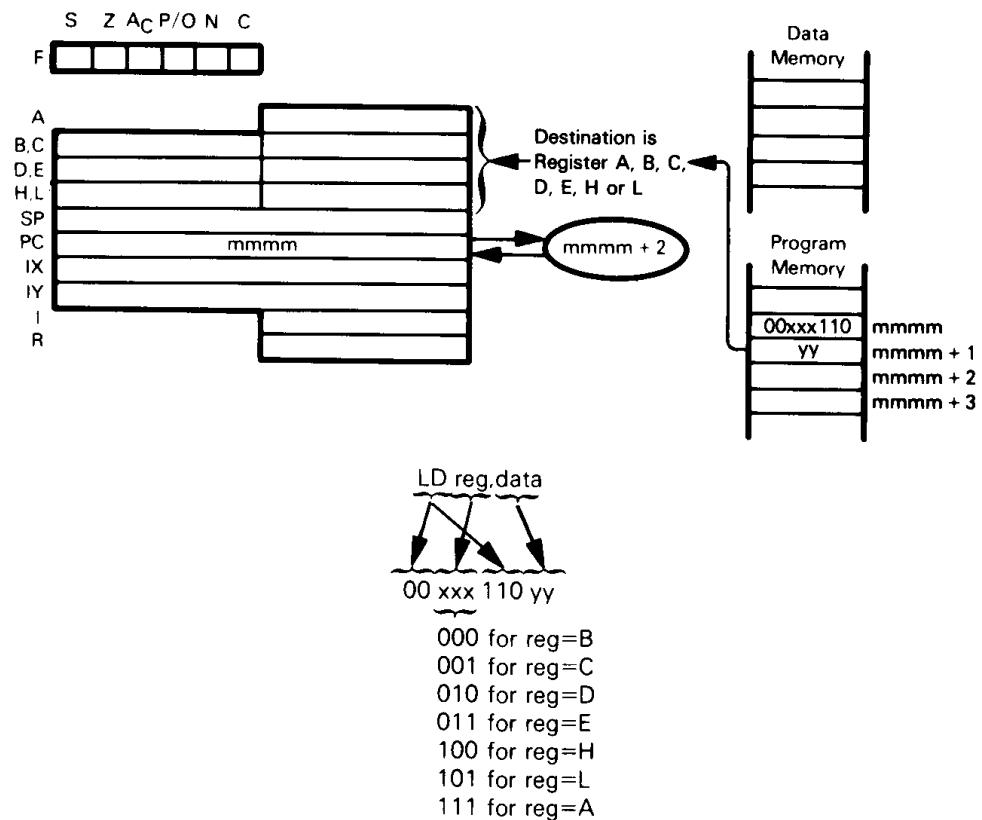
$\underbrace{\text{LD R,A}}_{\text{ED } 4\text{F}}$

has executed, the Refresh register will contain 7F16.

$\underbrace{\text{LD I,A}}_{\text{ED } 47}$

Load Interrupt Vector register from Accumulator.

## LD reg,data — LOAD IMMEDIATE INTO REGISTER



Load the contents of the second object code byte into one of the registers.

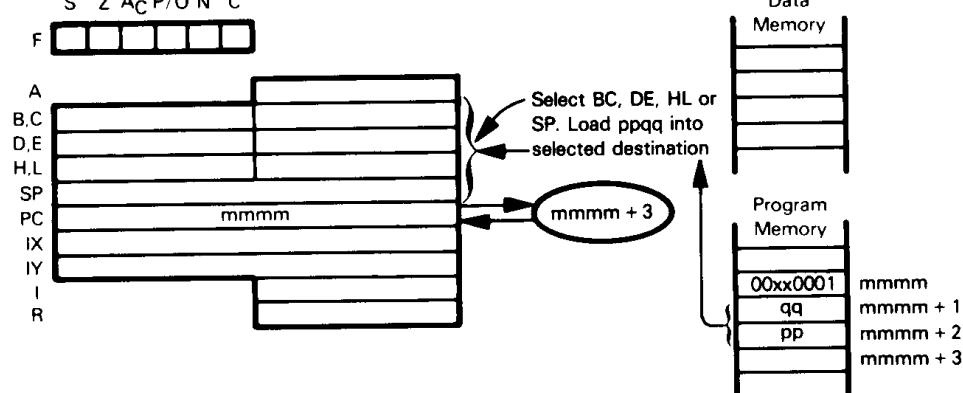
When the instruction

LD A,2AH

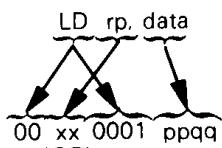
has executed, 2A16 is loaded into the Accumulator.

**LD rp,data — LOAD 16 BITS OF DATA IMMEDIATE INTO  
REGISTER**

**LD IX,data**



The illustration shows execution of LD rp,data:



- 00 for rp is register pair BC
- 01 for rp is register pair DE
- 10 for rp is register pair HL
- 11 for rp is Stack Pointer

Load the contents of the second and third object code bytes into the selected register pair. After the instruction

LD SP,217AH

has executed, the Stack Pointer will contain 217A16.

LD IX, data  
DD 21 ppqq

Load the contents of the second and third object code bytes into the Index register IX.

LD IY, data  
FD 21 ppqq

Load the contents of the second and third object code bytes into the Index Register IY.

Notice that the LD rp,data instruction is equivalent to two LD reg,data instructions.

For example:

LD        HL,032AH

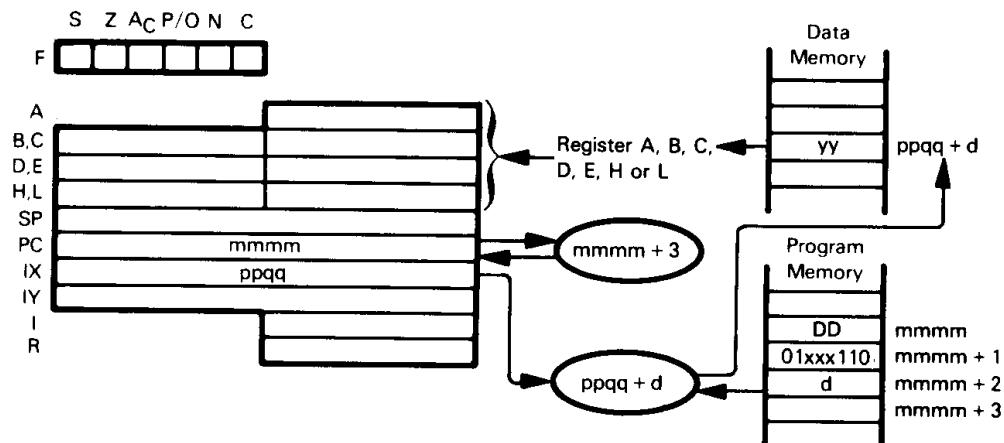
is equivalent to

LD        H,03H  
LD        L,2AH

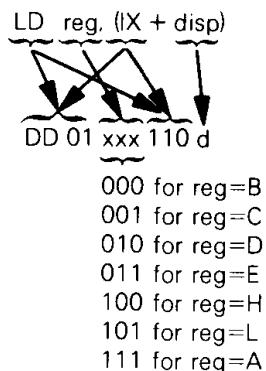
## LD reg,(HL) — LOAD REGISTER FROM MEMORY

LD reg,(IX+disp)

LD reg,(IY+disp)



The illustration shows execution of LD reg,(IX+disp):

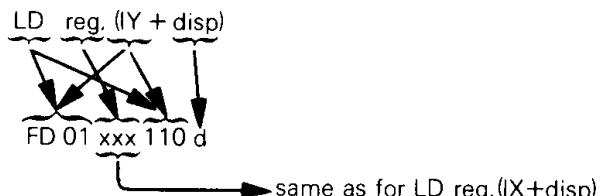


Load specified register from memory location (specified by the sum of the contents of the IX register and the displacement digit d).

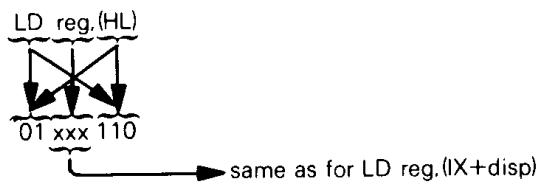
Suppose ppqq=400416 and memory location 401016 contains FF16. After the instruction

LD B(IX+0CH)

has executed, Register B will contain FF16.

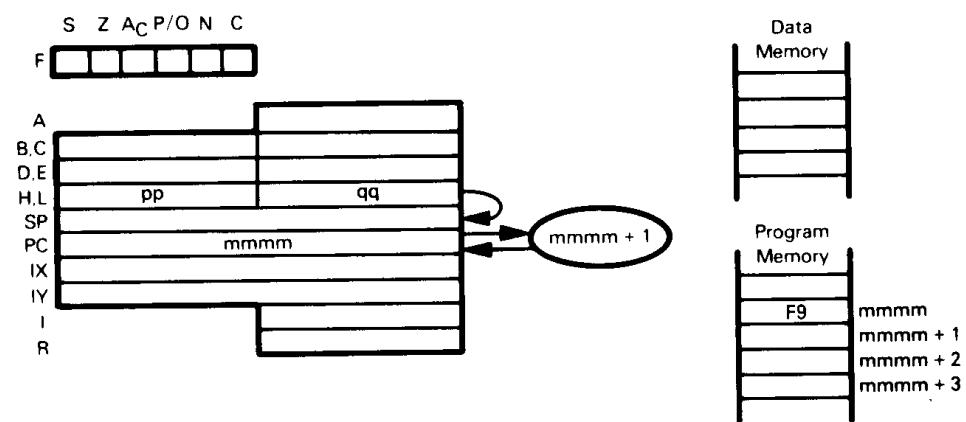


This instruction is identical to LD reg,(IX+disp), except that it uses the IY register instead of the IX register.



Load specified register from memory location (specified by the contents of the HL register pair).

**LD SP,HL — MOVE CONTENTS OF HL OR INDEX REGISTER TO STACK POINTER**  
**LD SP,IX**  
**LD SP,IY**



The illustration shows execution of LD SP,HL:

**LD SP,HL**  
F9

Load contents of HL into Stack Pointer.

Suppose pp=0816 and qq=3F16. After the instruction

LD SP,HL

has executed, the Stack Pointer will contain 083F16.

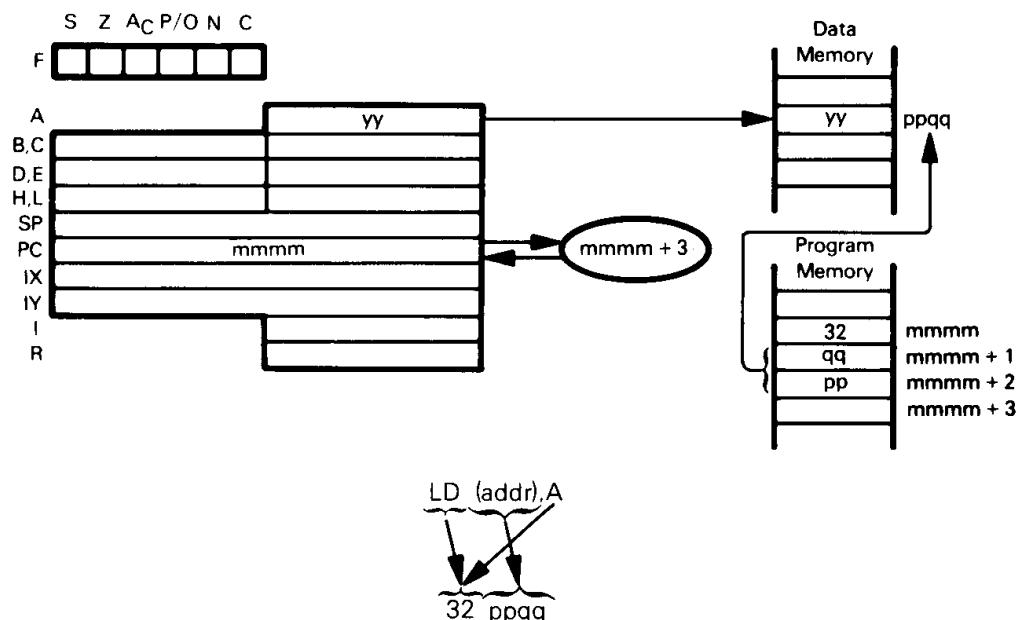
**LD SP,IX**  
DD F9

Load contents of Index Register IX into Stack Pointer.

**LD SP,IY**  
FD F9

Load contents of Index Register IY into Stack Pointer.

## LD (addr),A — STORE ACCUMULATOR IN MEMORY USING DIRECT ADDRESSING



Store the Accumulator contents in the memory byte addressed directly by the second and third bytes of the LD (addr),A instruction object code.

Suppose the Accumulator contains 3A16. After the instruction

```
label    EQU    084AH
-
-
LD      (label).A
```

has executed, memory byte 084A16 will contain 3A16.

Remember that EQU is an assembler directive rather than an instruction; it tells the Assembler to use the 16-bit value 084AH whenever the word "label" appears.

The instruction

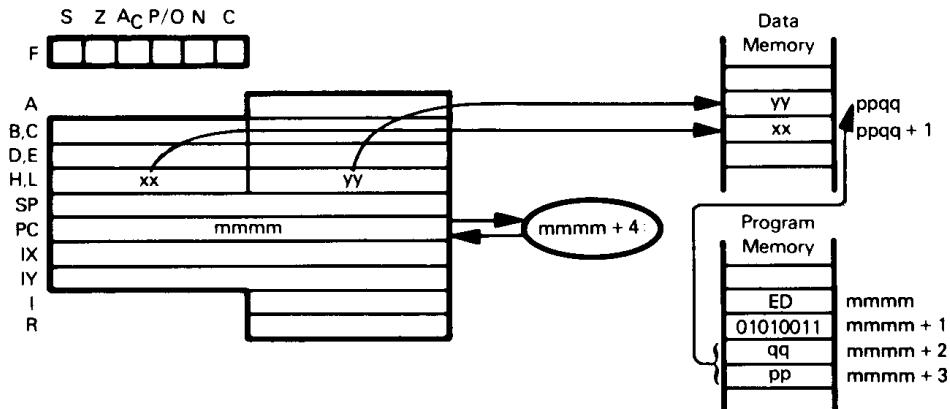
LD (addr),A

is equivalent to the two instructions

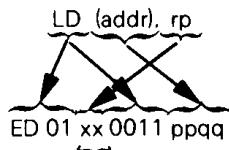
```
LD H,label
LD (HL),A
```

When you are storing a single data value in memory, the LD (label),A instruction is preferred because it uses one instruction and three object program bytes to do what the LD H(label), LD (HL),A combination does in two instructions and four object program bytes. Also, the LD H(label), LD (HL),A combination uses the H and L registers, while the LD (label),A instruction does not.

**LD (addr),HL — STORE REGISTER PAIR OR INDEX  
LD (addr),rp    REGISTER IN MEMORY USING DIRECT  
LD (addr),xy    ADDRESSING**



The illustration shows execution of LD (ppqq),DE:



00 for rp is register pair BC  
01 for rp is register pair DE  
10 for rp is register pair HL  
11 for rp is Stack Pointer

Store the contents of the specified register pair in memory. The third and fourth object code bytes give the address of the memory location where the low-order byte is to be written. The high-order byte is written into the next sequential memory location.

Suppose the BC register pair contains 3C2A16. After the instruction

label EQU 084AH

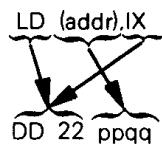
LD (label),BC

has executed, memory byte 084A16 will contain 2A16. Memory byte 084B16 will contain 3C16.

Remember that EQU is an assembler directive rather than an instruction; it tells the Assembler to use the 16-bit value 084A16 whenever the word "label" appears.



This is a three-byte version of LD (addr),rp which directly specifies HL as the source register pair.



Store the contents of Index register IX in memory. The third and fourth object code bytes give the address of the memory location where the low-order byte is to be written. The high-order byte is written into the next sequential memory location.

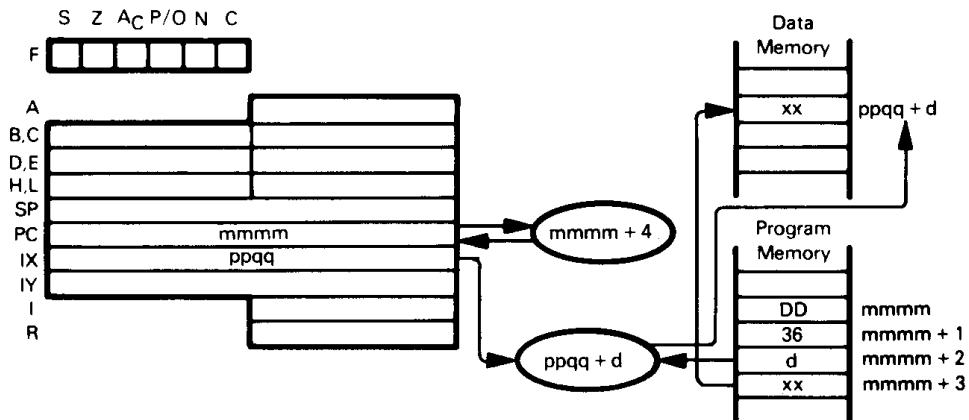


This instruction is identical to the LD (addr),IX instruction, except that it uses the IY register instead of the IX register.

## **LD (HL),data — LOAD IMMEDIATE INTO MEMORY**

**LD (IX+disp),data**

**LD (IY+disp),data**



The illustration shows execution of LD (IX+d),xx:

LD (IX+disp),data

DD 36 d xx

Load Immediate into the Memory location designated by base relative addressing.

Suppose ppqq=5400<sub>16</sub>. After the instruction

LD (IX+9),FAH

has executed, memory location 5409<sub>16</sub> will contain FA<sub>16</sub>.

LD (IY+disp),data

FD 36 d xx

This instruction is identical to LD (IX+disp),data, but uses the IY register instead of the IX register.

LD (HL),data

36 xx

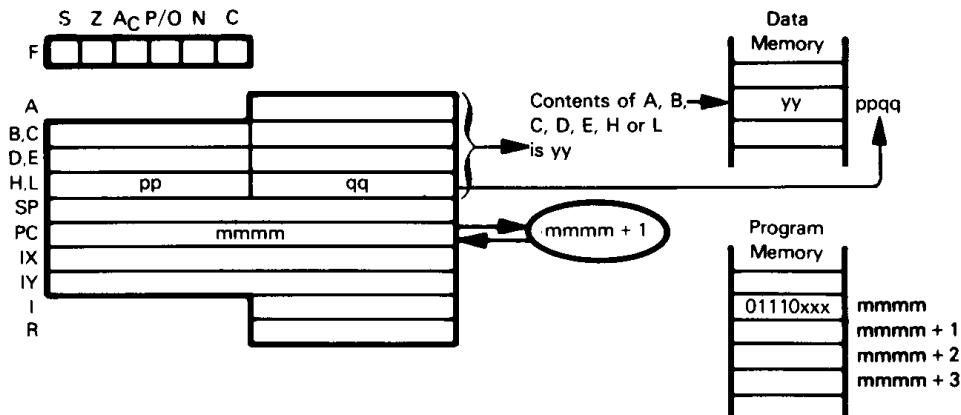
Load Immediate into the Memory location (specified by the contents of the HL register pair).

The Load Immediate into Memory instructions are used much less than the Load Immediate into Register instructions.

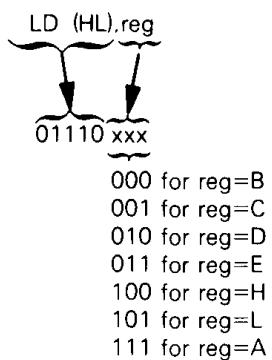
## **LD (HL),reg — LOAD MEMORY FROM REGISTER**

**LD (IX+disp),reg**

**LD (IY+disp),reg**



The illustration shows execution of LD (HL),reg:

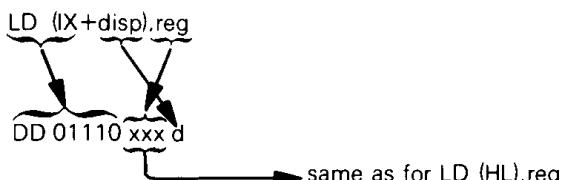


Load memory location (specified by the contents of the HL register pair) from specified register.

Suppose ppqq=450016 and Register C contains F916. After the instruction

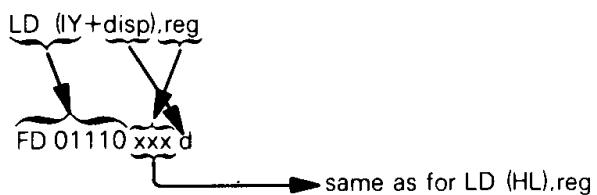
LD (HL),C

has executed, memory location 450016 will contain F916.



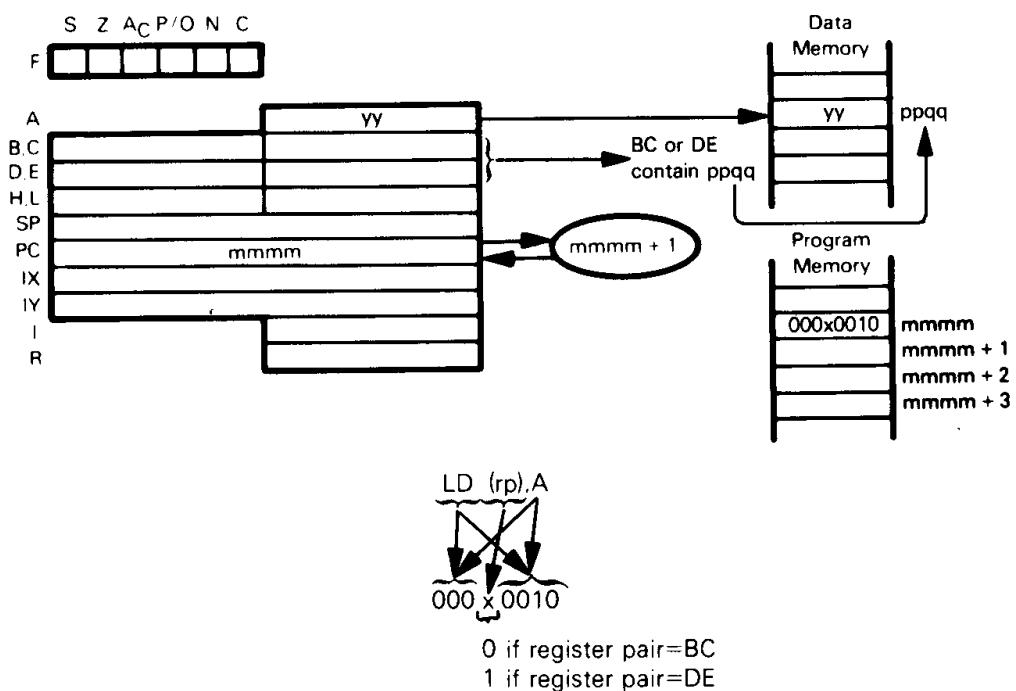
Load memory location (specified by the sum of the contents of the IX register and the

displacement value d) from specified register.



This instruction is identical to LD (IX+disp),reg, except that it uses the IY register instead of the IX register.

### LD (rp),A — LOAD ACCUMULATOR INTO THE MEMORY LOCATION ADDRESSED BY REGISTER PAIR



Store the Accumulator in the memory byte addressed by the BC or DE register pair.

Suppose the BC register pair contains 084A<sub>16</sub> and the Accumulator contains 3A<sub>16</sub>. After the instruction

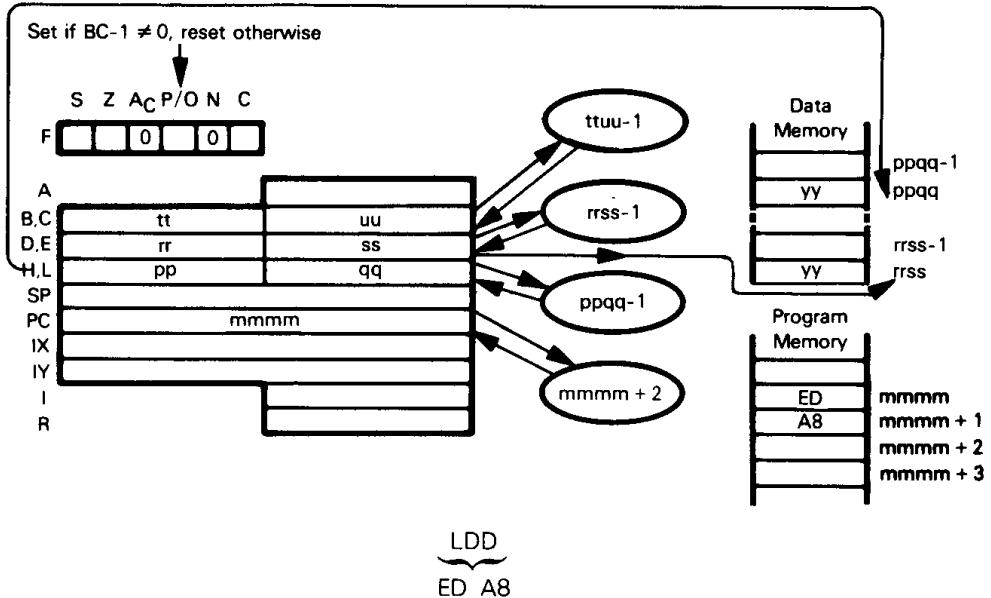
LD (BC),A

has executed, memory byte 084A<sub>16</sub> will contain 3A<sub>16</sub>.

The LD (rp),A and LD rp,data will normally be used together, since the LD rp,data instruction loads a 16-bit address into the BC or DE registers as follows:

LD BC,084AH  
LD (BC),A

## LDD — TRANSFER DATA BETWEEN MEMORY LOCATIONS, DECREMENT DESTINATION AND SOURCE ADDRESSES



Transfer a byte of data from memory location addressed by the HL register pair to memory location addressed by the DE register pair. Decrement contents of register pairs BC, DE, and HL.

Suppose register pair BC contains  $004F_{16}$ , DE contains  $4545_{16}$ , HL contains  $2012_{16}$ , and memory location  $2012_{16}$  contains  $18_{16}$ . After the instruction

LDD

has executed, memory location  $4545_{16}$  will contain  $18_{16}$ , register pair BC will contain  $004E_{16}$ , DE will contain  $4544_{16}$ , and HL will contain  $2011_{16}$ .

**LDDR — TRANSFER DATA BETWEEN MEMORY LOCATIONS UNTIL BYTE COUNTER IS ZERO. DECREMENT DESTINATION AND SOURCE ADDRESSES**

LDDR  
ED B8

This instruction is identical to LDD, except that it is repeated until the BC register pair contains zero. After each data transfer, interrupts will be recognized and two refresh cycles will be executed.

Suppose we have the following contents in memory and register pairs:

| <u>Register/Contents</u> | <u>Location/Contents</u> |
|--------------------------|--------------------------|
| HL 201216                | 201216 1816              |
| DE 454516                | 201116 AA16              |
| BC 000316                | 201016 2516              |

After execution of

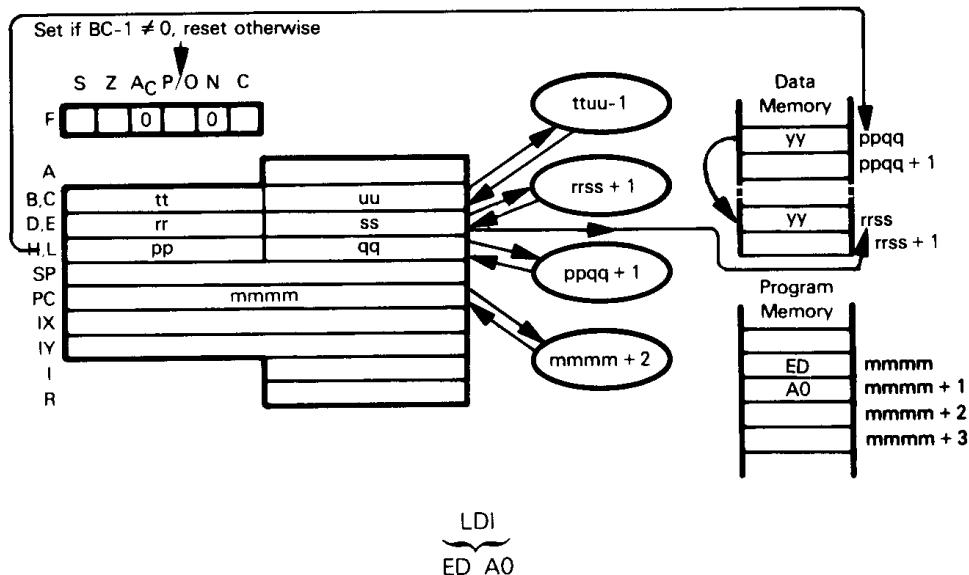
LDDR

register pairs and memory locations will have the following contents:

| <u>Register/Contents</u> | <u>Location/Contents</u> | <u>Location/Contents</u> |
|--------------------------|--------------------------|--------------------------|
| HL 200916                | 201216 1816              | 454516 1816              |
| DE 454216                | 201116 AA16              | 454416 AA16              |
| BC 000016                | 201016 2516              | 454316 2516              |

This instruction is extremely useful for transferring blocks of data from one area of memory to another.

## LDI — TRANSFER DATA BETWEEN MEMORY LOCATIONS, INCREMENT DESTINATION AND SOURCE ADDRESSES



Transfer a byte of data from memory location addressed by the HL register pair to memory location addressed by the DE register pair. Increment contents of register pairs HL and DE. Decrement contents of the BC register pair.

Suppose register pair BC contains 004F<sub>16</sub>, DE contains 4545<sub>16</sub>, HL contains 2012<sub>16</sub>, and memory location 2012<sub>16</sub> contains 18<sub>16</sub>. After the instruction

LDI

has executed, memory location 4545<sub>16</sub> will contain 18<sub>16</sub>, register pair BC will contain 004E<sub>16</sub>, DE will contain 4546<sub>16</sub>, and HL will contain 2013<sub>16</sub>.

## **LDIR — TRANSFER DATA BETWEEN MEMORY LOCATIONS UNTIL BYTE COUNTER IS ZERO. INCREMENT DESTINATION AND SOURCE ADDRESSES**

LDIR  
ED 80

This instruction is identical to LDI, except that it is repeated until the BC register pair contains zero. After each data transfer, interrupts will be recognized and two refresh cycles will be executed.

Suppose we have the following contents in memory and register pairs:

| Register/Contents | Location/Contents |
|-------------------|-------------------|
| HL 201216         | 201216 1816       |
| DE 454516         | 201316 CD16       |
| BC 000316         | 201416 F016       |

After execution of

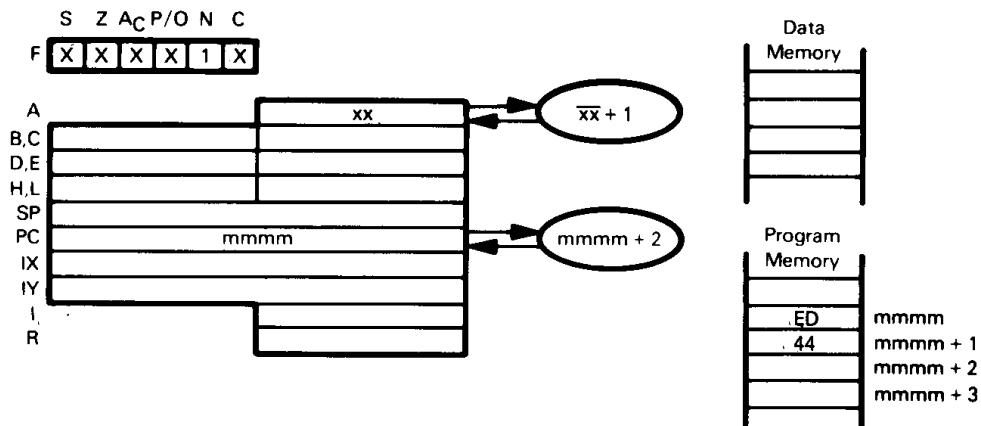
LDIR

register pairs and memory will have the following contents:

| Register/Contents | Location/Contents | Location/Contents |
|-------------------|-------------------|-------------------|
| HL 201516         | 201216 1816       | 454516 1816       |
| DE 454816         | 201316 CD16       | 454616 CD16       |
| BC 000016         | 201416 F016       | 454716 F016       |

This instruction is extremely useful for transferring blocks of data from one area of memory to another.

## **NEG — NEGATE CONTENTS OF ACCUMULATOR**



Negate contents of Accumulator. This is the same as subtracting contents of the Accumulator from zero. The result is the two's complement. 80H will be left unchanged.

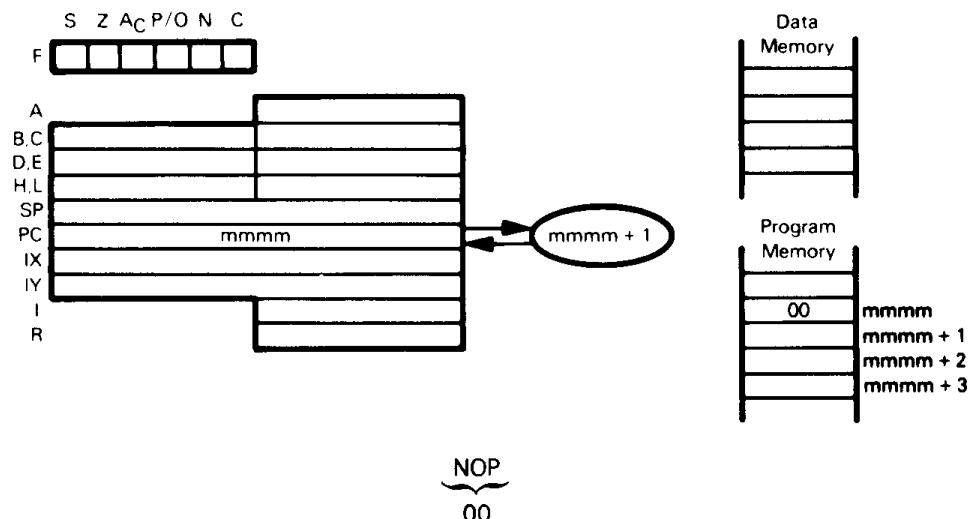
Suppose xx=5A16. After the instruction

NEG

has executed, the Accumulator will contain A616.

$$\begin{array}{ll} 5A & = 0101 \quad 1010 \\ \text{Two's complement} & = 1010 \quad 0110 \end{array}$$

## NOP — NO OPERATION

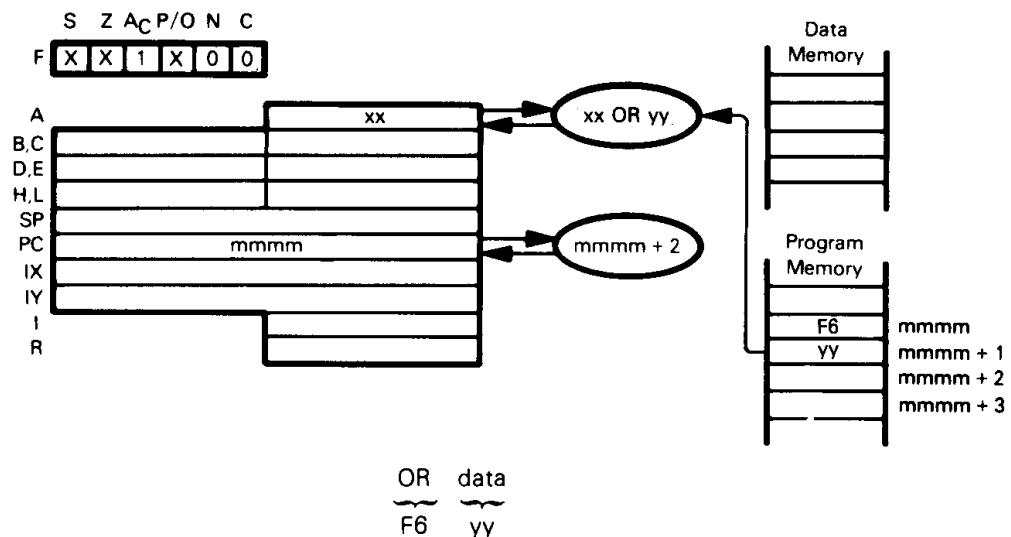


This is a one-byte instruction which performs no operation, except that the Program Counter is incremented and memory refresh continues. This instruction is present for several reasons:

- 1) A program error that fetches an object code from non-existent memory will fetch 00. It is a good idea to ensure that the most common program error will do nothing.
- 2) The NOP instruction allows you to give a label to an object program byte:  
HERE NOP
- 3) To fine-tune delay times. Each NOP instruction adds four clock cycles to a delay.

NOP is not a very useful or frequently used instruction.

## OR data — OR IMMEDIATE WITH ACCUMULATOR



OR the Accumulator with the contents of the second instruction object code byte.

Suppose  $xx=3A_{16}$ . After the instruction

OR 7CH

has executed, the Accumulator will contain  $7E_{16}$ .

$$\begin{array}{r}
 3A = 0011 \quad 1010 \\
 7C = 0111 \quad 1100 \\
 \hline
 0111 \quad 1110
 \end{array}$$

Six 1 bits, set P/O to 1

Non-zero result, set Z to 0

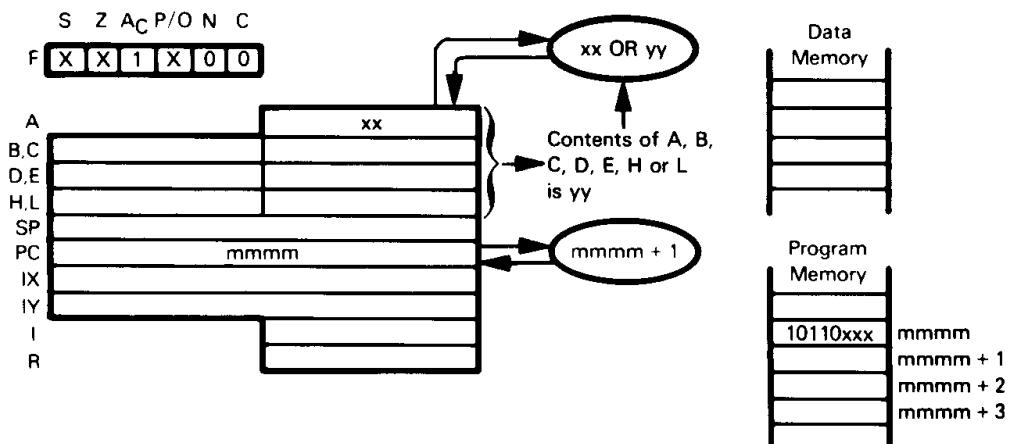
0 sets S to 0

This is a routine logical instruction; it is often used to turn bits "on". For example, the instruction

OR 80H

will unconditionally set the high-order Accumulator bit to 1.

## OR reg — OR REGISTER WITH ACCUMULATOR



$\underbrace{\text{OR}}_{10110}$      $\underbrace{\text{reg}}_{\text{xxx}}$   
 000 for reg=B  
 001 for reg=C  
 010 for reg=D  
 011 for reg=E  
 100 for reg=H  
 101 for reg=L  
 111 for reg=A

Logically OR the contents of the Accumulator with the contents of Register A, B, C, D, E, H or L. Store the result in the Accumulator.

Suppose xx=E3<sub>16</sub> and Register E contains A8<sub>16</sub>. After the instruction

OR E

has executed, the Accumulator will contain EB<sub>16</sub>.

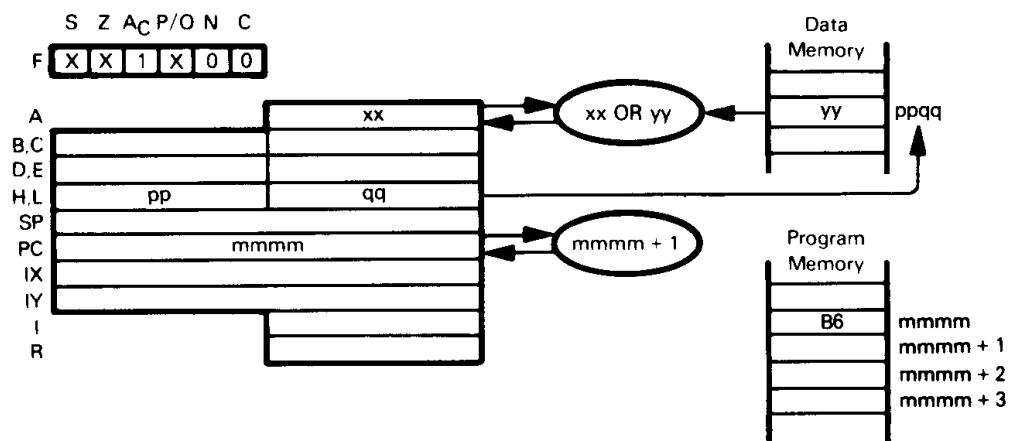
$$\begin{array}{r}
 \text{E3} = 1110 \quad 0011 \\
 \text{A8} = 1010 \quad 1000 \\
 \hline
 1110 \quad 1011
 \end{array}$$

1 sets S to 1      Six 1 bits, set P/O to 1  
 Non-zero result, set Z to 0

## OR (HL) — OR MEMORY WITH ACCUMULATOR

OR (IX+disp)

OR (IY+disp)



The illustration shows execution of OR (HL):

$\overbrace{\text{OR (HL)}}$   
 B6

OR contents of memory location (specified by the contents of the HL register pair) with the Accumulator.

Suppose xx=E3<sub>16</sub>, ppqq=4000<sub>16</sub>, and memory location 4000<sub>16</sub> contains A8<sub>16</sub>. After the instruction

OR (HL)

has executed, the Accumulator will contain EB<sub>16</sub>.

$$\begin{array}{r}
 \text{E3} = 1110 \quad 0011 \\
 \text{A8} = 1010 \quad 1000 \\
 \hline
 1110 \quad 1011
 \end{array}$$

1 sets S to 1

Six 1 bits, set P/O to 1

Non-zero result, set Z to 0

$\overbrace{\text{OR (IX+disp)}}$

DD B6 d

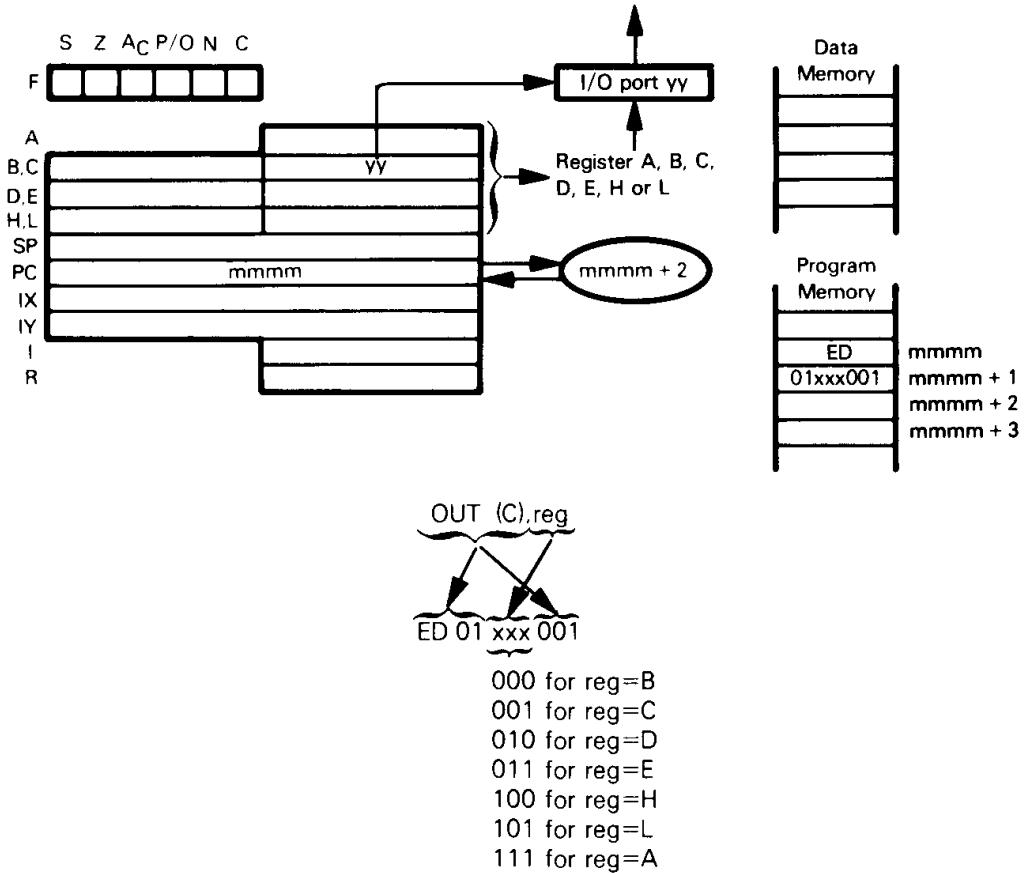
OR contents of memory location (specified by the sum of the contents of the IX register and the displacement value d) with the Accumulator.

$\overbrace{\text{OR (IY+disp)}}$

FD B6 d

This instruction is identical to OR (IX+disp), except that it uses the IY register instead of the IX register.

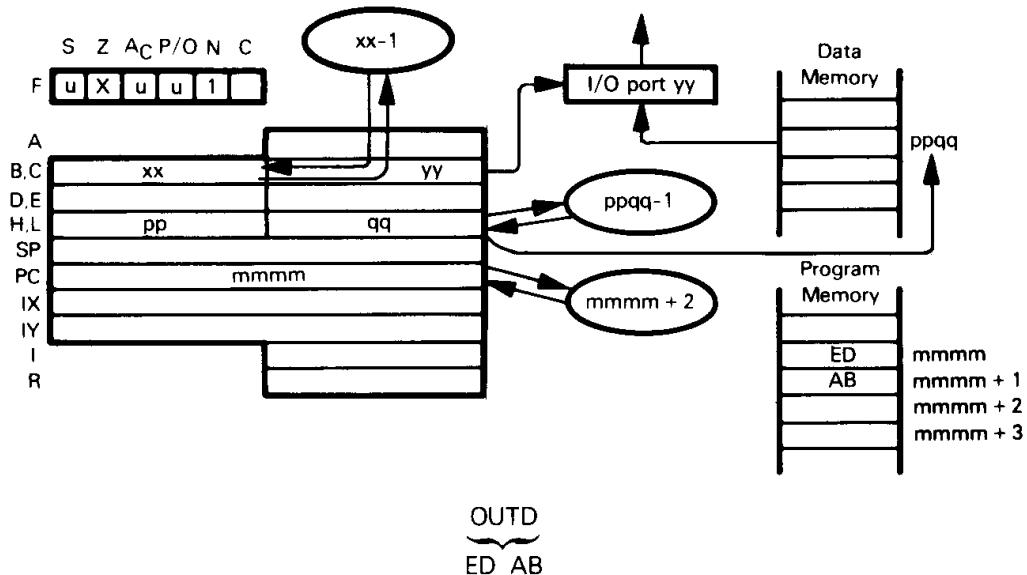
## OUT (C),reg — OUTPUT FROM REGISTER



Suppose yy=1F<sub>16</sub> and the contents of H are AA<sub>16</sub>. After the execution of  
OUT (C),H

AA<sub>16</sub> will be in the buffer of I/O port 1F<sub>16</sub>.

## OUTD — OUTPUT FROM MEMORY. DECREMENT ADDRESS



Output from memory location specified by HL to I/O port addressed by Register C. Registers B and HL are decremented.

Suppose  $xx=0A16$ ,  $yy=FF16$ ,  $ppqq=500016$ , and memory location  $500016$  contains  $7716$ . After the instruction

OUTD

has executed,  $7716$  will be held in the buffer of I/O port  $FF16$ . The B register will contain  $0916$ , and the HL register pair  $4FFF16$ .

## OTDR — OUTPUT FROM MEMORY. DECREMENT ADDRESS, CONTINUE UNTIL REGISTER B=0

OTDR  
 $\underbrace{\quad}_{ED}$  BB

OTDR is identical to OUTD, but is repeated until Register B contains 0.

Suppose Register B contains  $0316$ , Register C contains  $FF16$ , and HL contains  $500016$ . Memory locations  $4FFE16$  through  $500016$  contain:

| Location/Contents |        |
|-------------------|--------|
| $4FFE16$          | $CA16$ |
| $4FFF16$          | $1B16$ |
| $500016$          | $F116$ |

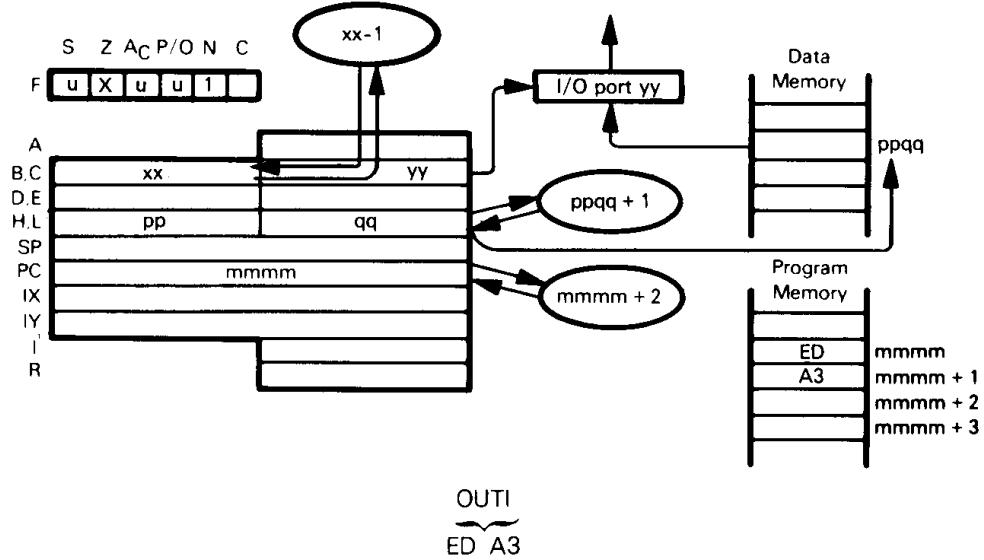
After execution of

OTDR

register pair HL will contain  $4FFD16$ , Register B will contain zero, and the sequence  $F116$ ,  $1B16$ ,  $CA16$  will have been written to I/O port  $FF16$ .

This instruction is very useful for transferring blocks of data from memory to output devices.

## OUTI — OUTPUT FROM MEMORY. INCREMENT ADDRESS



Output from memory location specified by HL to I/O port addressed by Register C. Register B is decremented and the HL register pair is incremented.

Suppose  $xx=0A16$ ,  $yy=FF16$ ,  $ppqq=500016$ , and memory location  $500016$  contains  $7716$ . After the instruction

OUTI

has executed,  $7716$  will be held in the buffer of I/O port  $FF16$ . The B register will contain  $0916$  and the HL register pair will contain  $500116$ .

## OTIR — OUTPUT FROM MEMORY. INCREMENT ADDRESS, CONTINUE UNTIL REGISTER B=0

OTIR  
ED B3

OTIR is identical to OUTI, except that it is repeated until Register B contains 0.

Suppose Register B contains  $0416$ , Register C contains  $FF16$ , and HL contains  $500016$ . Memory locations  $500016$  through  $500316$  contain:

| Location/Contents |      |
|-------------------|------|
| 500016            | CA16 |
| 500116            | 1B16 |
| 500216            | B116 |
| 500316            | AD16 |

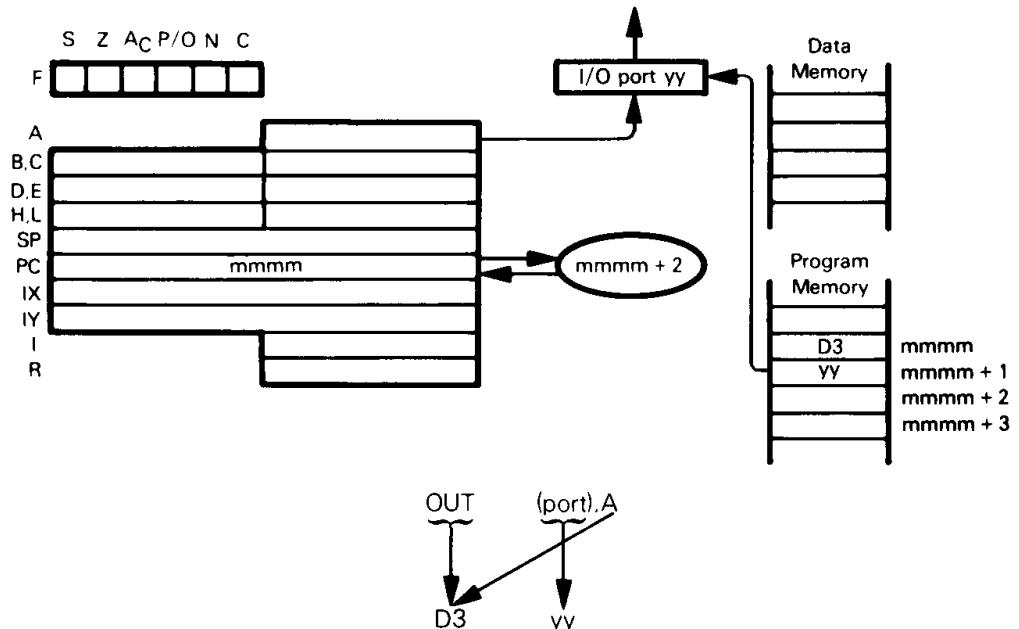
After execution of

OTIR

register pair HL will contain  $500416$ . Register B will contain zero and the sequence CA16, 1B16, B116 and AD16 will have been written to I/O port  $FF16$ .

This instruction is very useful for transferring blocks of data from memory to an output device.

## OUT (port),A — OUTPUT FROM ACCUMULATOR



Output the contents of the Accumulator to the I/O port identified by the second OUT instruction object code byte.

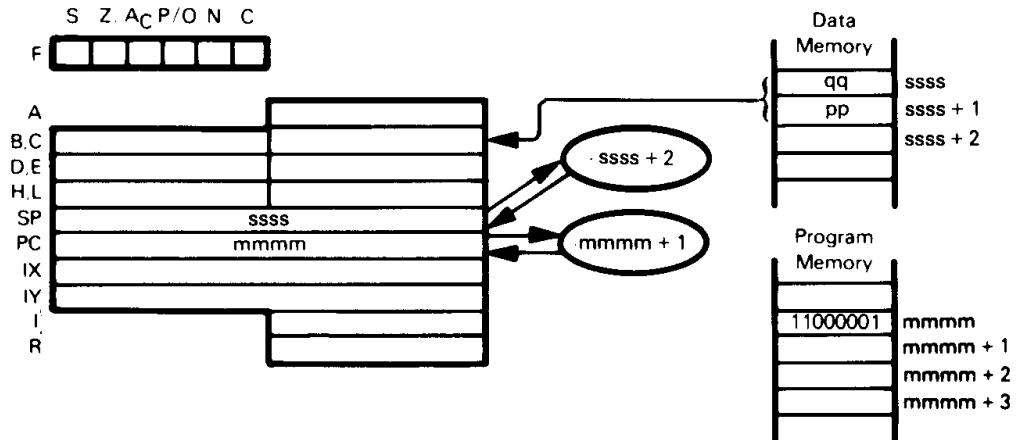
Suppose 36<sub>16</sub> is held in the Accumulator. After the instruction

OUT (1AH),A

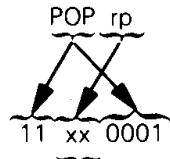
has executed, 36<sub>16</sub> will be in the buffer of I/O port 1A<sub>16</sub>.

The OUT instruction does not affect any statuses. Use of the OUT instruction is very hardware-dependent. Valid I/O port addresses are determined by the way in which I/O logic has been implemented. It is also possible to design a microcomputer system that accesses external logic using memory reference instructions with specific memory addresses. OUT instructions are frequently used in special ways to control microcomputer logic external to the CPU.

**POP rp — READ FROM THE TOP OF THE STACK**  
**POP IX**  
**POP IY**



The illustration shows execution of POP BC:



00 for rp is register pair BC  
 01 for rp is register pair DE  
 10 for rp is register pair HL  
 11 for rp is register pair A and F

POP the two top stack bytes into the designated register pair.

Suppose qq=0116 and pp=2A16. Execution of

POP HL

loads 0116 into the L register and 2A16 into the H register. Execution of the instruction

POP AF

loads 01 into the status flags and 2A16 into the Accumulator. Thus, the Carry status will be set to 1 and other statuses will be cleared.

POP IX  
 DD E1

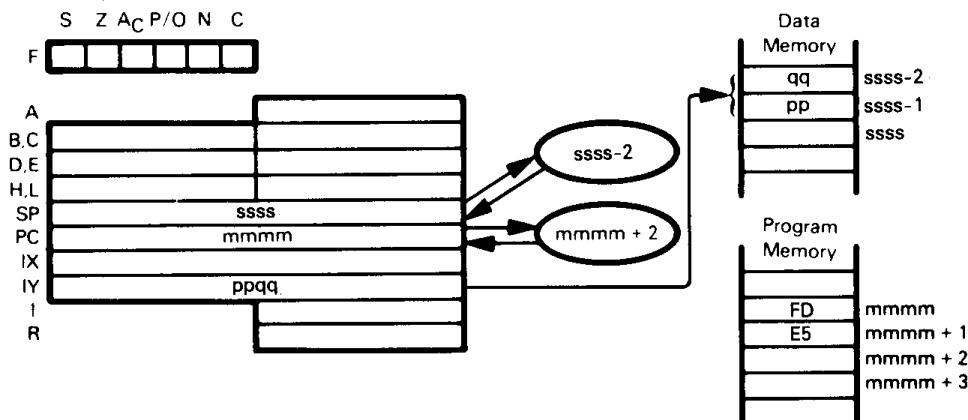
POP the two top stack bytes into the IX register.

POP IY  
 FD E1

POP the two top stack bytes into the IY register.

The POP instruction is most frequently used to restore register and status contents which have been saved on the stack; for example, while servicing an interrupt.

**PUSH rp — WRITE TO THE TOP OF THE STACK**  
**PUSH IX**  
**PUSH IY**



The illustration shows execution of PUSH IY:

**PUSH IY**  
FD E5

PUSH the contents of the IY register onto the top of the stack.

Suppose the IY register contains 45FF<sub>16</sub>. Execution of the instruction

**PUSH IY**

loads 45<sub>16</sub>, then FF<sub>16</sub> onto the top of the stack.

**PUSH IX**  
DD E5

PUSH the contents of the IX register onto the top of the stack.



00 for rp is register pair BC  
 01 for rp is register pair DE  
 10 for rp is register pair HL  
 11 for rp is register pair A and F

PUSH contents of designated register pair onto the top of the stack.

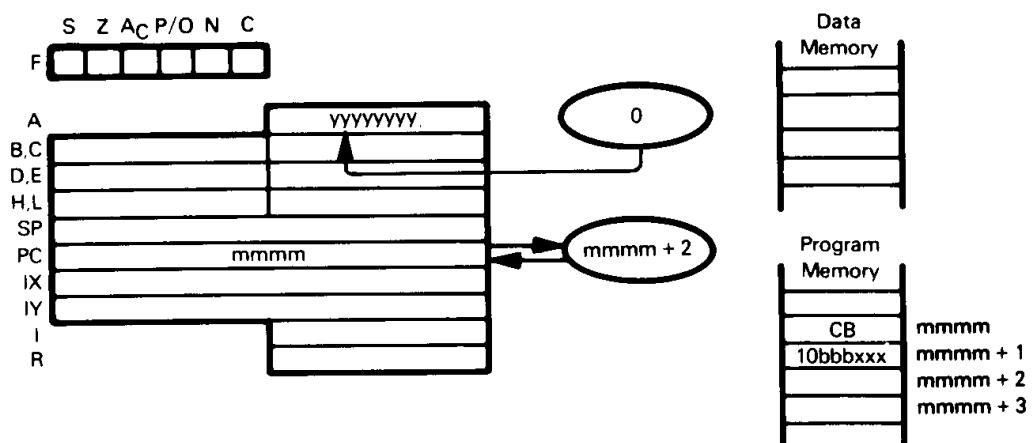
Execution of the instruction

**PUSH AF**

loads the Accumulator and then the status flags onto the top of the stack.

The PUSH instruction is most frequently used to save register and status contents; for example, before servicing an interrupt.

## RES b,reg — RESET INDICATED REGISTER BIT



| <u>RES</u> | <u>b,reg</u> |            |                 |
|------------|--------------|------------|-----------------|
| <u>CB</u>  | <u>10</u>    |            |                 |
| <u>Bit</u> | <u>bbb</u>   | <u>xxx</u> | <u>Register</u> |
| 0          | 000          | 000        | B               |
| 1          | 001          | 001        | C               |
| 2          | 010          | 010        | D               |
| 3          | 011          | 011        | E               |
| 4          | 100          | 100        | H               |
| 5          | 101          | 101        | L               |
| 6          | 110          | 111        | A               |
| 7          | 111          |            |                 |

Reset indicated bit within specified register.

After the instruction

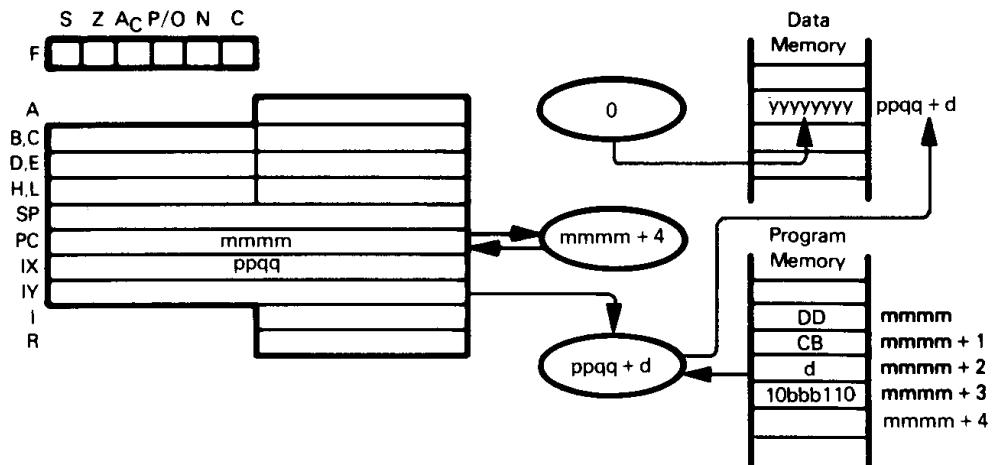
RES 6,H

has executed, bit 6 in Register H will be reset. (Bit 0 is the least significant bit.)

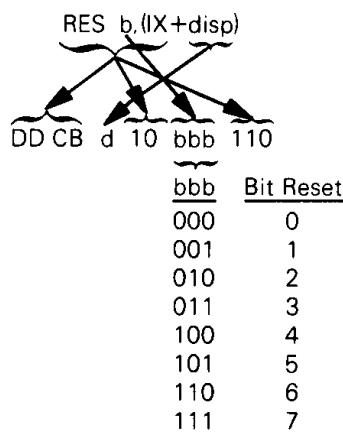
**RES b,(HL) — RESET BIT b OF INDICATED MEMORY POSITION**

**RES b,(IX+disp)**

**RES b,(IY+disp)**



The illustration shows execution of SET b,(IX+disp). Bit 0 is execution of SET b,(IX+disp). Bit 0 is the least significant bit.

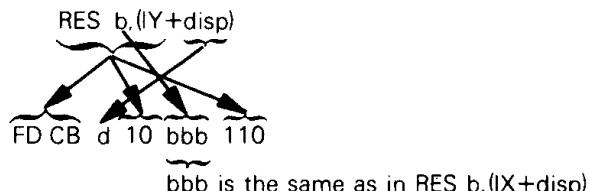


Reset indicated bit within memory location indicated by the sum of Index Register IX and d.

Suppose IX contains 411016. After the instruction

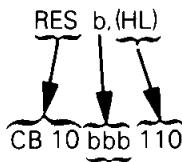
RES 0,(IX+7)

has executed, bit 0 in memory location 411716 will be 0.



This instruction is identical to RES b,(IX+disp), except that it uses the IY register instead.

of the IX register.



bbb is the same as in RES b,(IX+disp)

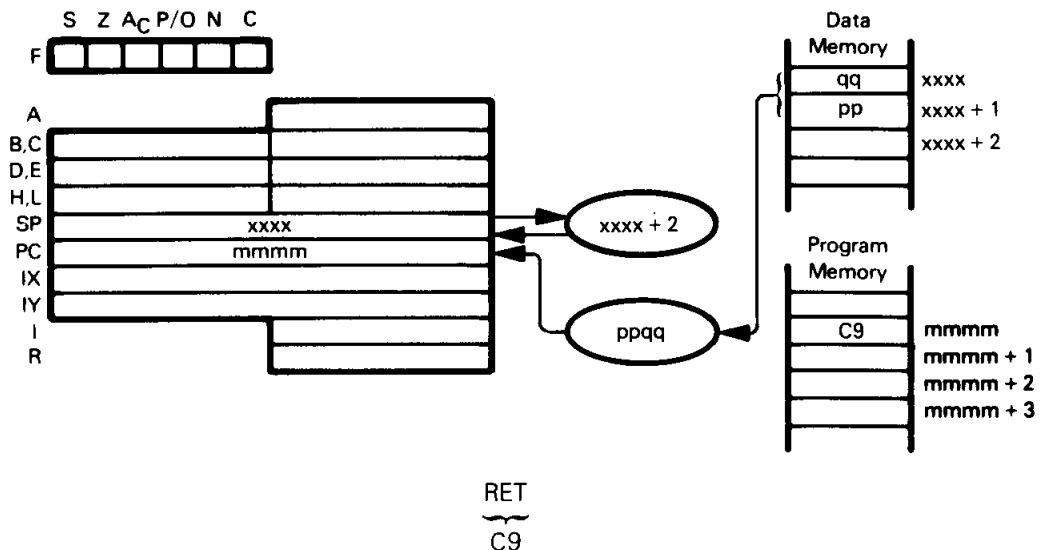
Reset indicated bit within memory location indicated by HL.

Suppose HL contains 4444<sub>16</sub>. After execution of

RES 7,(HL)

bit 7 in memory location 4444<sub>16</sub> will be 0.

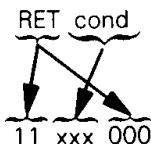
## RET — RETURN FROM SUBROUTINE



Move the contents of the top two stack bytes to the Program Counter; these two bytes provide the address of the next instruction to be executed. Previous Program Counter contents are lost. Increment the Stack Pointer by 2, to address the new top of stack.

Every subroutine must contain at least one Return (or conditional Return) instruction; this is the last instruction executed within the subroutine, and causes execution to return to the calling program.

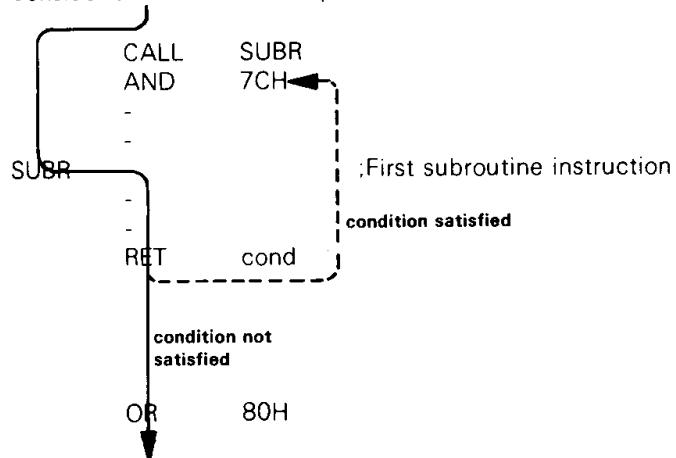
## RET cond — RETURN FROM SUBROUTINE IF CONDITION IS SATISFIED



|     |    | Condition     | Relevant Flag |
|-----|----|---------------|---------------|
| 000 | NZ | Non-Zero      | Z             |
| 001 | Z  | Zero          | Z             |
| 010 | NC | Non-Carry     | C             |
| 011 | C  | Carry         | C             |
| 100 | PO | Parity Odd    | P/O           |
| 101 | PE | Parity Even   | P/O           |
| 110 | P  | Sign Positive | S             |
| 111 | M  | Sign Negative | S             |

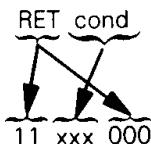
This instruction is identical to the RET instruction, except that the return is not executed unless the condition is satisfied; otherwise, the instruction sequentially following the RET cond instruction will be executed.

Consider the instruction sequence:



After the RET cond is executed, if the condition is satisfied then execution returns to the AND instruction which follows the CALL. If the condition is not satisfied, the OR instruction, being the next sequential instruction, is executed.

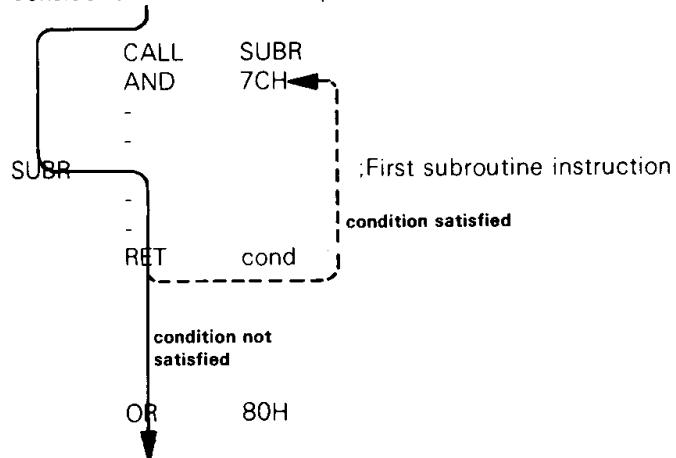
## RET cond — RETURN FROM SUBROUTINE IF CONDITION IS SATISFIED



|     |    | Condition     | Relevant Flag |
|-----|----|---------------|---------------|
| 000 | NZ | Non-Zero      | Z             |
| 001 | Z  | Zero          | Z             |
| 010 | NC | Non-Carry     | C             |
| 011 | C  | Carry         | C             |
| 100 | PO | Parity Odd    | P/O           |
| 101 | PE | Parity Even   | P/O           |
| 110 | P  | Sign Positive | S             |
| 111 | M  | Sign Negative | S             |

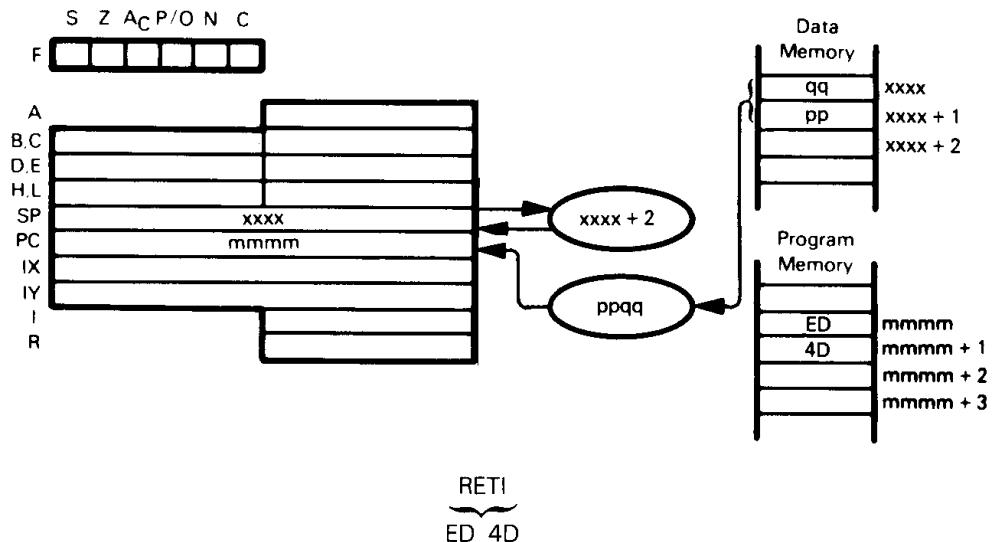
This instruction is identical to the RET instruction, except that the return is not executed unless the condition is satisfied; otherwise, the instruction sequentially following the RET cond instruction will be executed.

Consider the instruction sequence:



After the RET cond is executed, if the condition is satisfied then execution returns to the AND instruction which follows the CALL. If the condition is not satisfied, the OR instruction, being the next sequential instruction, is executed.

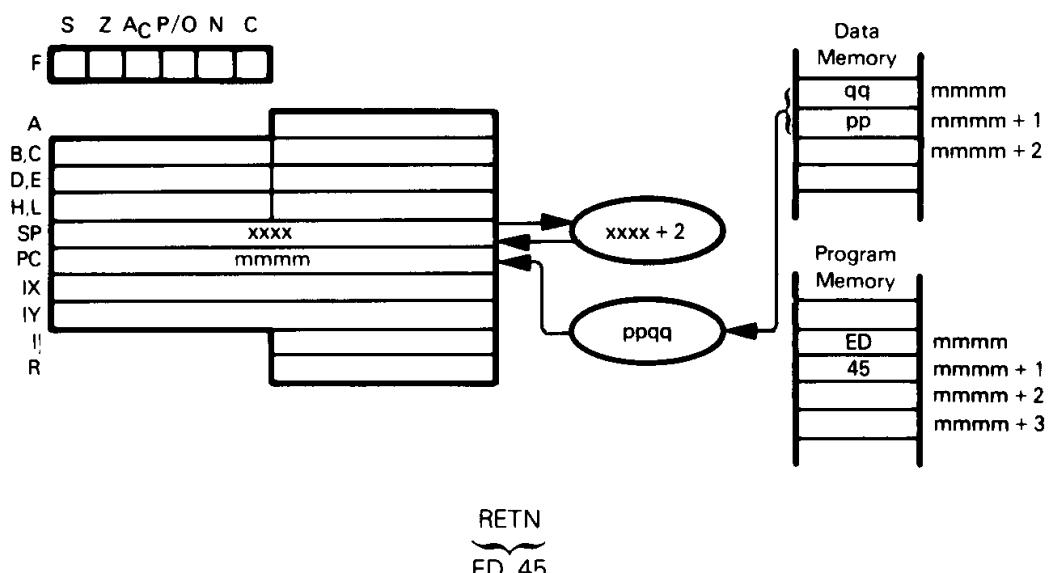
## RETI — RETURN FROM INTERRUPT



Move the contents of the top two stack bytes to the Program Counter; these two bytes provide the address of the next instruction to be executed. Previous Program Counter contents are lost. Increment the Stack Pointer by 2, and address the new top of stack.

This instruction is used at the end of an interrupt service routine, and, in addition to returning control to the interrupted program, it is used to signal an I/O device that the interrupt routine has been completed. The I/O device must provide the logic necessary to sense the instruction operation code: refer to An Introduction to Microcomputers: Volume 2 for a description of how the RETI instruction operates with the Z80 family of devices.

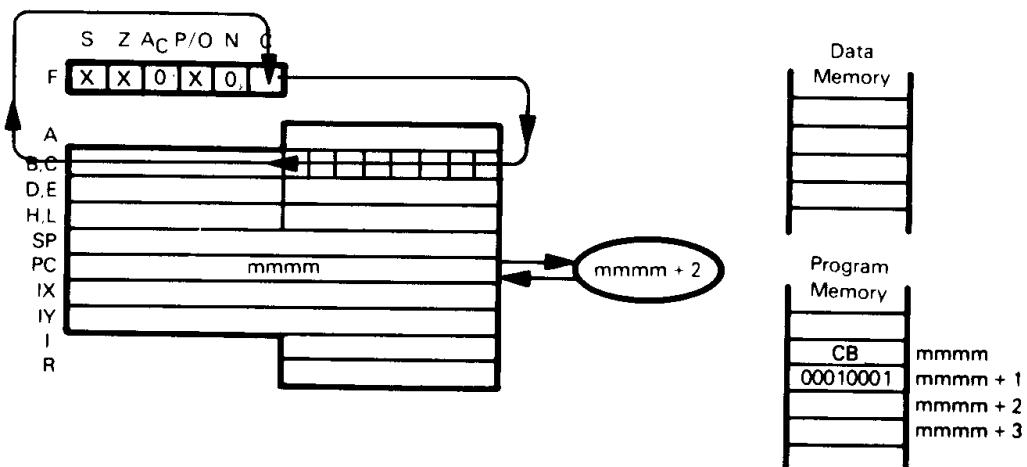
## RETN — RETURN FROM NON-MASKABLE INTERRUPT



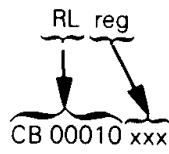
Move the contents of the top two stack bytes to the Program Counter; these two bytes provide the address of the next instruction to be executed. Previous Program Counter contents are lost. Increment the Stack Pointer by 2 to address the new top of stack. Restore the interrupt enable logic to the state it had prior to the occurrence of the non-maskable interrupt.

This instruction is used at the end of a service routine for a non-maskable interrupt, and causes execution to return to the program that was interrupted.

## RL reg — ROTATE CONTENTS OF REGISTER LEFT THROUGH CARRY



The illustration shows execution of RL C:



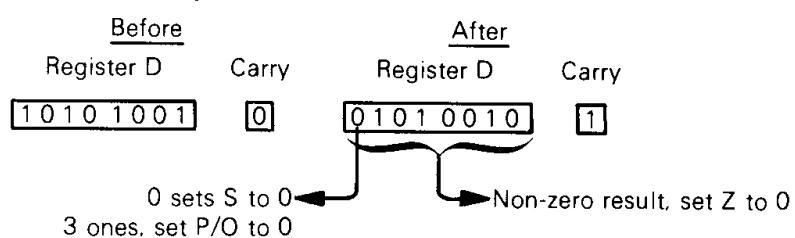
- 000 for reg=B
- 001 for reg=C
- 010 for reg=D
- 011 for reg=E
- 100 for reg=H
- 101 for reg=L
- 111 for reg=A

Rotate contents of specified register left one bit through Carry.

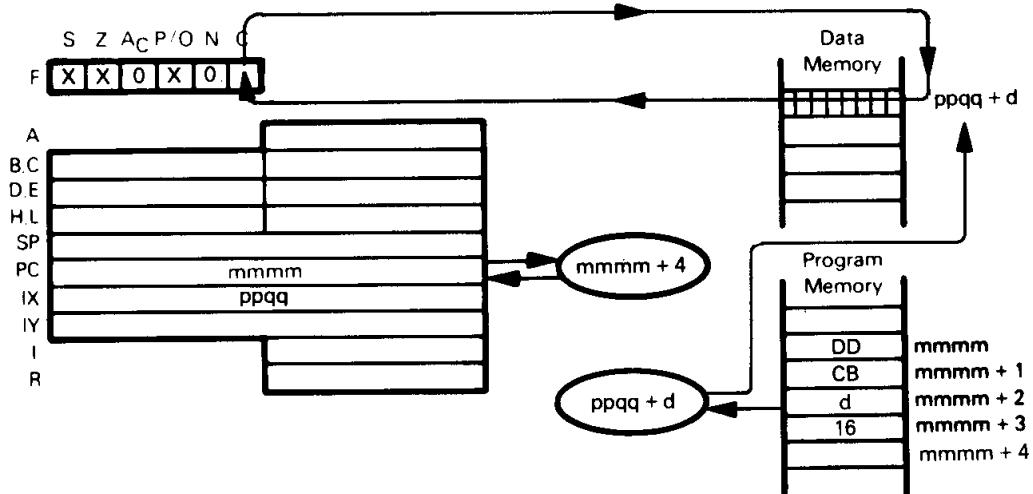
Suppose D contains A9<sub>16</sub> and Carry=0. After the instruction

RL D

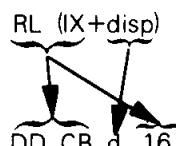
has executed, D will contain 52<sub>16</sub> and Carry will be 1:



**RL (HL) — ROTATE CONTENTS OF MEMORY LOCATION**  
**RL (IX+disp) LEFT THROUGH CARRY**  
**RL (IY+disp)**



The illustration shows execution of RL (IX+disp):

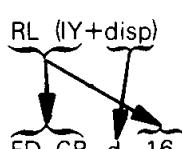
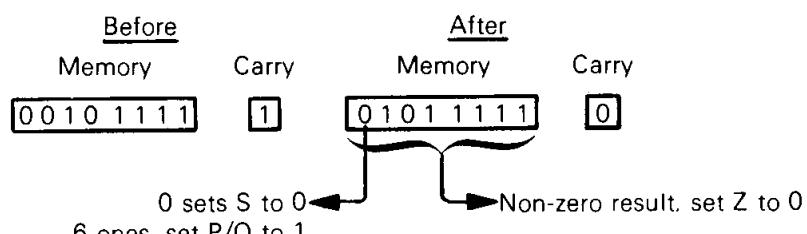


Rotate contents of memory location (specified by the sum of the contents of Index Register IX and displacement integer d) left one bit through Carry.

Suppose the IX register contains 4000<sub>16</sub>, memory location 4007<sub>16</sub> contains 2F<sub>16</sub>, and Carry is set to 1. After execution of the instruction

RL (IX+7)

memory location 4007<sub>16</sub> will contain 5F<sub>16</sub>, and Carry is 0:

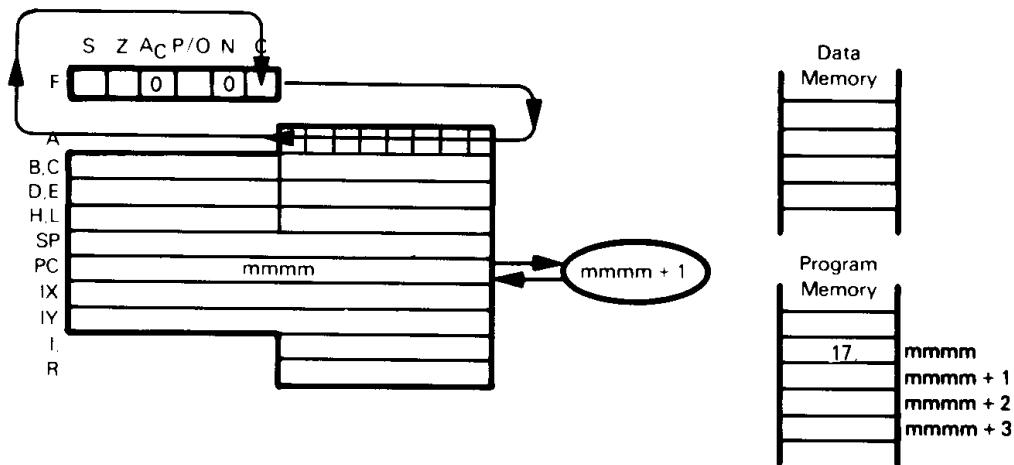


This instruction is identical to RL (IX+disp), but uses the IY register instead of the IX register.

$\overbrace{\text{RL (HL)}}$   
 CB 16

Rotate contents of memory location (specified by the contents of the HL register pair) left one bit through Carry.

### RLA — ROTATE ACCUMULATOR LEFT THROUGH CARRY



$\overbrace{\text{RLA}}$   
 17

Rotate Accumulator contents left one bit through Carry status.

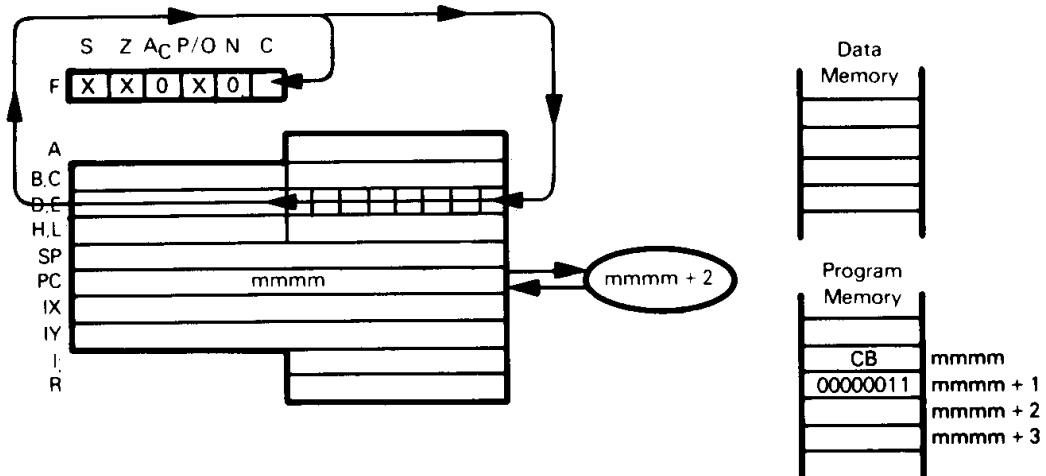
Suppose the Accumulator contains 2A<sub>16</sub> and the Carry status is set to 1. After the instruction

RLA

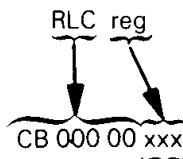
has executed, the Accumulator will contain F5<sub>16</sub> and the Carry status will be reset to 0:

| <u>Before</u>   |       | <u>After</u>    |       |
|-----------------|-------|-----------------|-------|
| Accumulator     | Carry | Accumulator     | Carry |
| 0 1 1 1 1 0 1 0 | 1     | 1 1 1 1 0 1 0 1 | 0     |

## RLC reg — ROTATE CONTENTS OF REGISTER LEFT CIRCULAR



The illustration shows execution of RLC E:



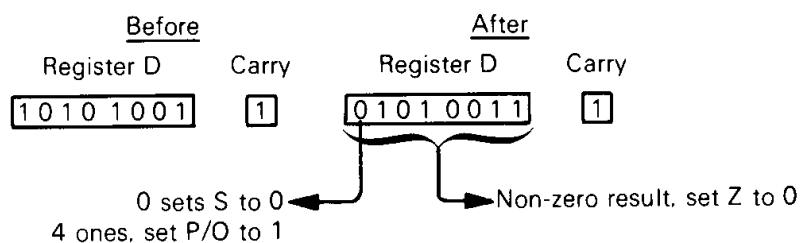
- 000 for reg=B
- 001 for reg=C
- 010 for reg=D
- 011 for reg=E
- 100 for reg=H
- 101 for reg=L
- 111 for reg=A

Rotate contents of specified register left one bit, copying bit 7 into Carry.

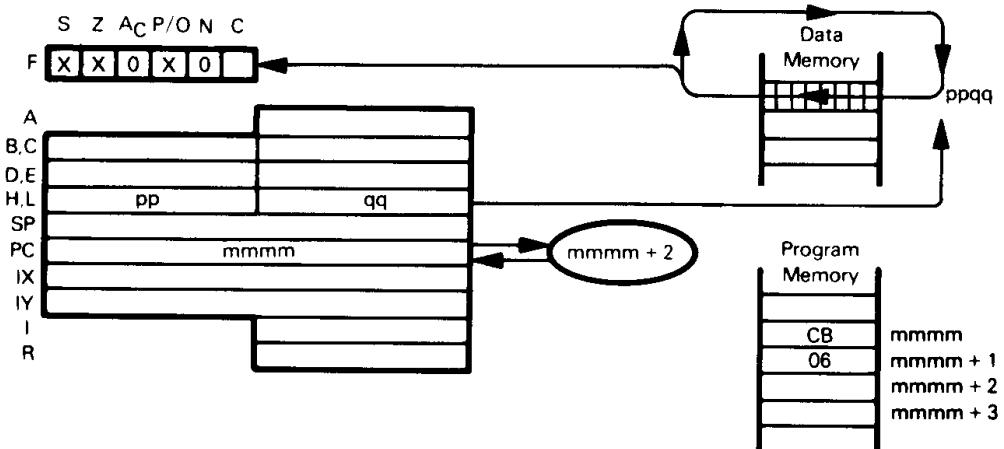
Suppose Register D contains A9<sub>16</sub> and Carry is 1. After execution of

RLC D

Register D will contain 53<sub>16</sub> and Carry will be 1:



**RLC (HL) — ROTATE CONTENTS OF MEMORY LOCATION**  
**RLC (IX+disp) LEFT CIRCULAR**  
**RLC (IY+disp)**



The illustration shows execution of RLC (HL):

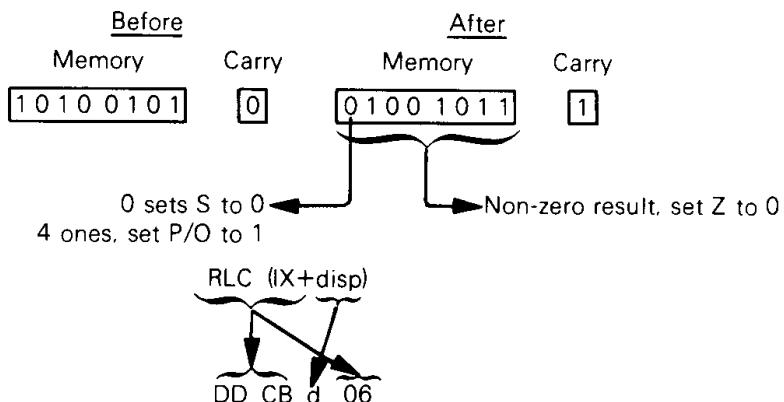
RLC (HL)  
CB 06

Rotate contents of memory location (specified by the contents of the HL register pair) left one bit, copying bit 7 into Carry.

Suppose register pair HL contains 54FF<sub>16</sub>. Memory location 54FF<sub>16</sub> contains A5<sub>16</sub>, and Carry is 0. After execution of

RLC (HL)

memory location 54FF<sub>16</sub> will contain 4B<sub>16</sub>, and Carry will be 1:

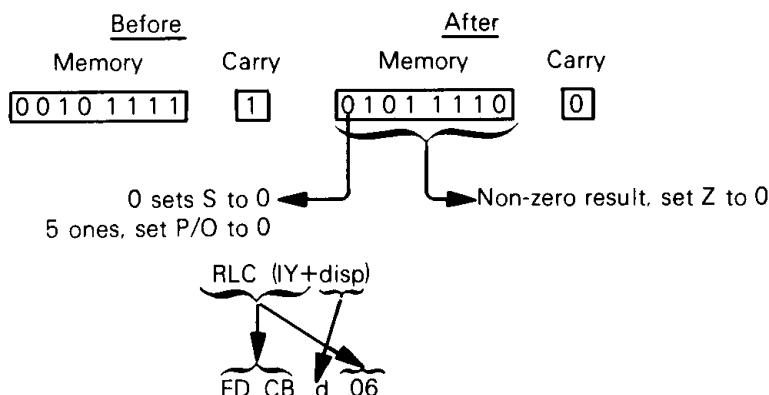


Rotate memory location (specified by the sum of the contents of Index register IX and displacement integer d) left one bit, copying bit 7 into Carry.

Suppose the IX register contains 4000<sub>16</sub>, Carry is 1, and memory location 4007<sub>16</sub> contains 2F<sub>16</sub>. After the instruction

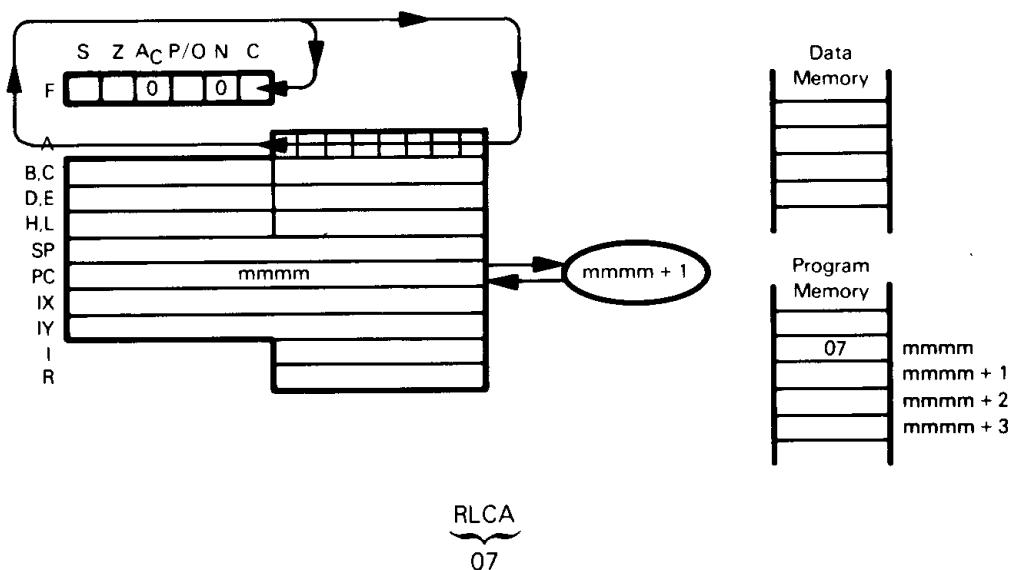
RLC (IX+7)

has executed, memory location 400716 will contain 5E16, and Carry will be 0:



This instruction is identical to RLC (IX+disp), but uses the IY register instead of the IX register.

### RLCA — ROTATE ACCUMULATOR LEFT CIRCULAR



Rotate Accumulator contents left one bit, copying bit 7 into Carry.

Suppose the Accumulator contains 7A16 and the Carry status is set to 1. After the instruction

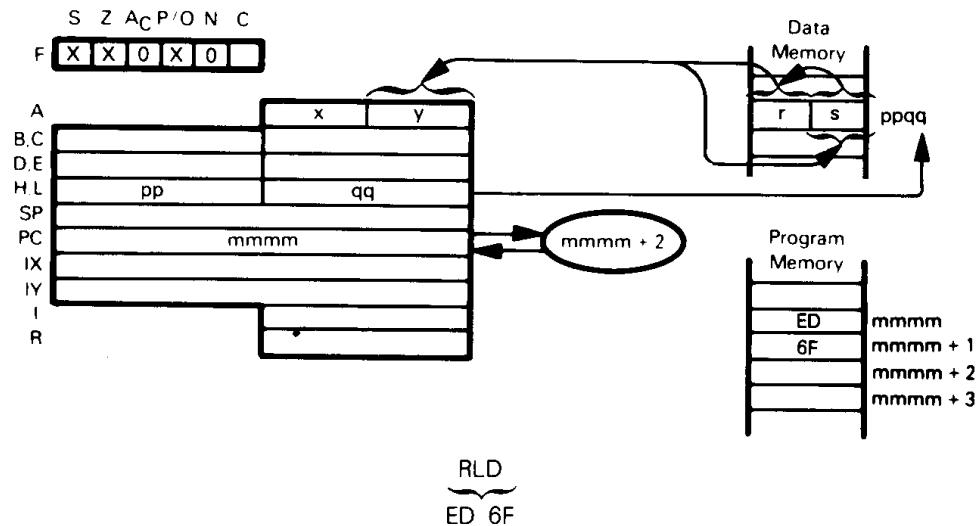
RLCA

has executed, the Accumulator will contain F416 and the Carry status will be reset to 0:

| <u>Before</u>                                                                                                                                                                                                                | <u>After</u>                                                                                                                                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Accumulator                                                                                                                                                                                                                  | Accumulator                                                                                                                                                                                                                  |
| <div style="display: flex; align-items: center; justify-content: space-between;"> <span style="border: 1px solid black; padding: 2px;">0111 1010</span> <span style="border: 1px solid black; padding: 2px;">1</span> </div> | <div style="display: flex; align-items: center; justify-content: space-between;"> <span style="border: 1px solid black; padding: 2px;">1111 0100</span> <span style="border: 1px solid black; padding: 2px;">0</span> </div> |

RLCA should be used as a logical instruction.

## RLD — ROTATE ONE BCD DIGIT LEFT BETWEEN THE ACCUMULATOR AND MEMORY LOCATION

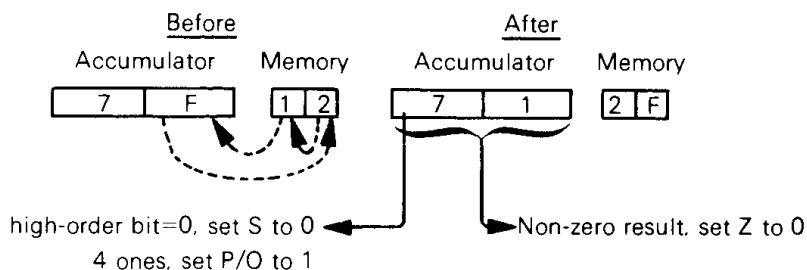


The four low-order bits of a memory location (specified by the contents of register pair HL) are copied into the four high-order bits of the same memory location. The previous contents of the four high-order bits of that memory location are copied into the four low-order bits of the Accumulator. The previous four low-order bits of the Accumulator are copied into the four low-order bits of the specified memory location.

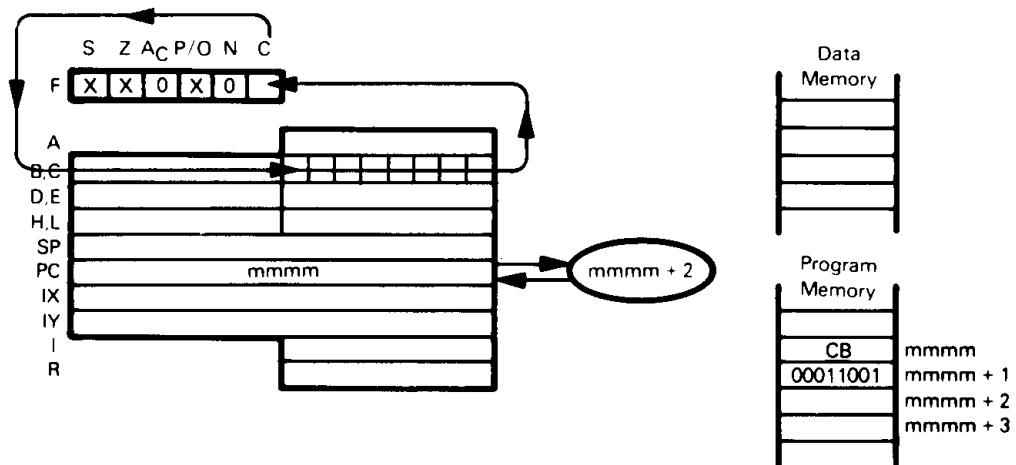
Suppose the Accumulator contains  $7F_{16}$ , HL register pair contains  $4000_{16}$ , and memory location  $4000_{16}$  contains  $12_{16}$ . After execution of the instruction

RLD

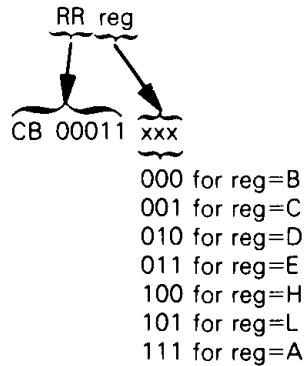
the Accumulator will contain  $71_{16}$  and memory location  $4000_{16}$  will contain  $2F_{16}$ :



## RR reg — ROTATE CONTENTS OF REGISTER RIGHT THROUGH CARRY



The illustration shows execution of RR C:

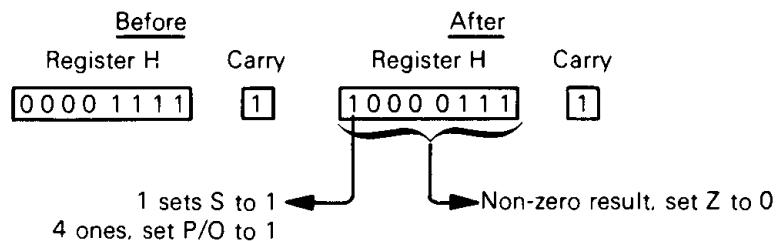


Rotate contents of specified register right one bit through Carry.

Suppose Register H contains 0F16 and Carry is set to 1. After the instruction

RR H

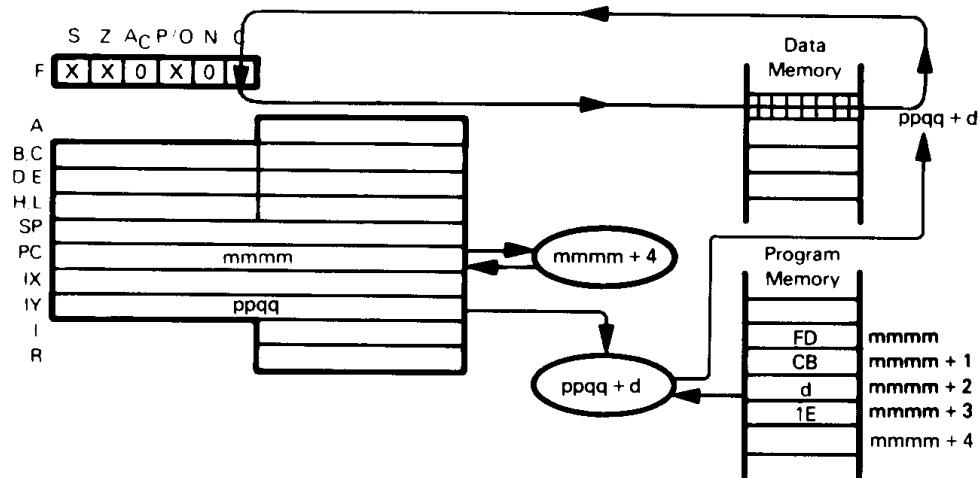
has executed, Register H will contain 87<sub>16</sub>, and Carry will be 1:



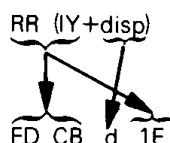
**RR (HL) — ROTATE CONTENTS OF MEMORY LOCATION  
RIGHT THROUGH CARRY**

**RR (IX+disp)**

**RR (IY+disp)**



The illustration shows execution of RR (IY+disp):

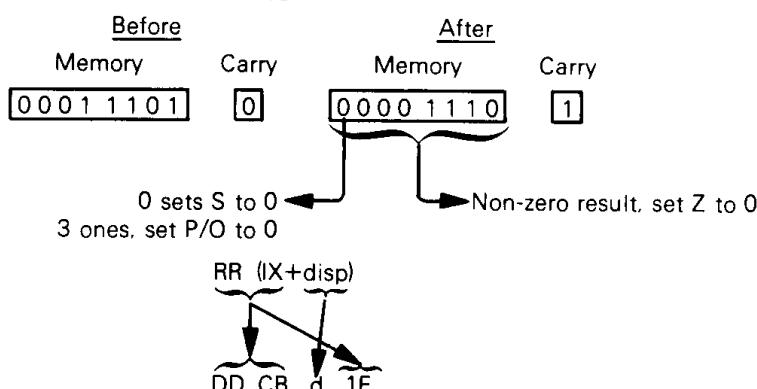


Rotate contents of memory location (specified by the sum of the contents of the IY register and the displacement value d) right one bit through Carry.

Suppose the IY register contains 4500<sub>16</sub>, memory location 450F<sub>16</sub> contains 1D<sub>16</sub>, and Carry is set to 0. After execution of the instruction

RR (IY+0FH)

memory location 450F<sub>16</sub> will contain 0E<sub>16</sub>, and Carry will be 1:

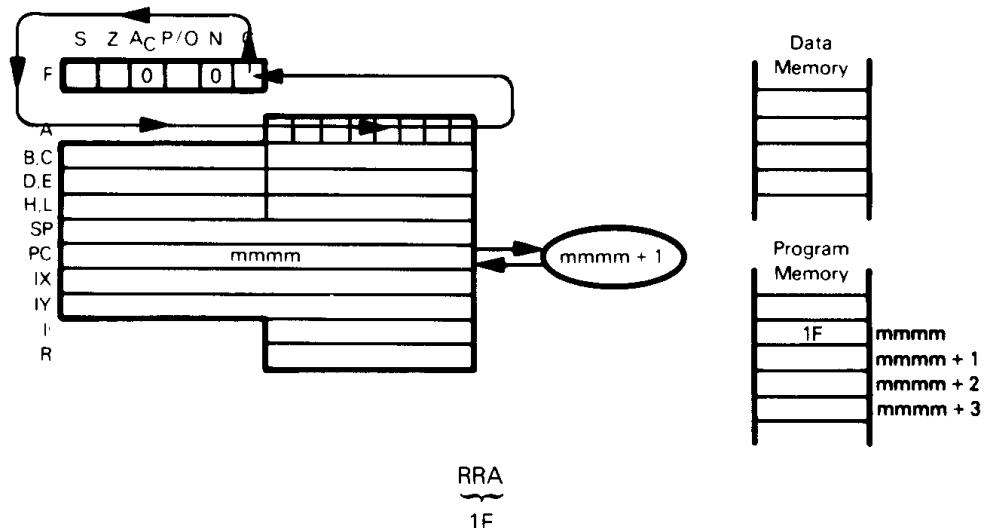


This instruction is identical to RR (IY+disp), but uses the IX register instead of the IY register.

RR (HL)  
CB 1E

Rotate contents of memory location (specified by the contents of the HL register pair) right one bit through Carry.

### RRA — ROTATE ACCUMULATOR RIGHT THROUGH CARRY



Rotate Accumulator contents right one bit through Carry status.

Suppose the Accumulator contains 7A<sub>16</sub> and the Carry status is set to 1. After the instruction

RRA

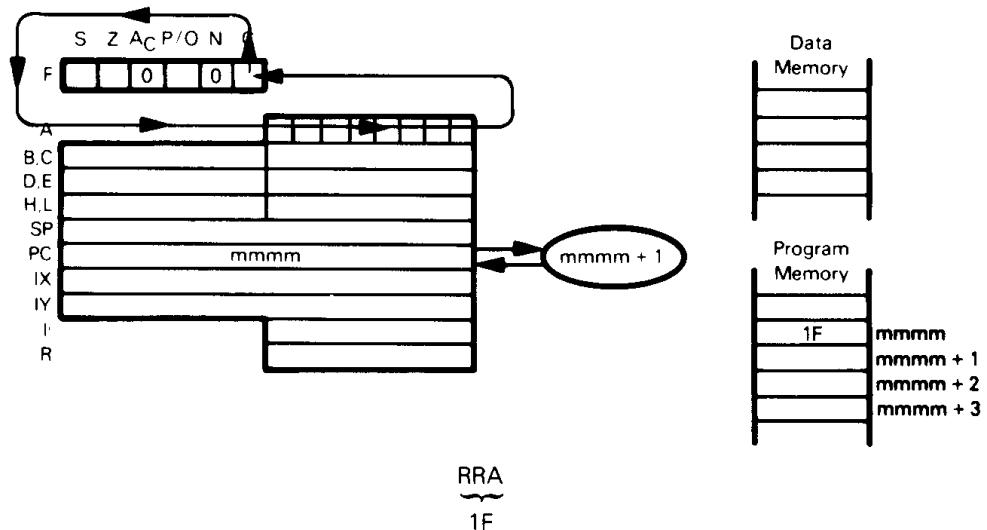
has executed, the Accumulator will contain BD<sub>16</sub> and the Carry status will be reset to 0:

| Before          |       | After           |       |
|-----------------|-------|-----------------|-------|
| Accumulator     | Carry | Accumulator     | Carry |
| 0 1 1 1 1 0 1 0 | 1     | 1 0 1 1 1 1 0 1 | 0     |

RR (HL)  
CB 1E

Rotate contents of memory location (specified by the contents of the HL register pair) right one bit through Carry.

### RRA — ROTATE ACCUMULATOR RIGHT THROUGH CARRY



Rotate Accumulator contents right one bit through Carry status.

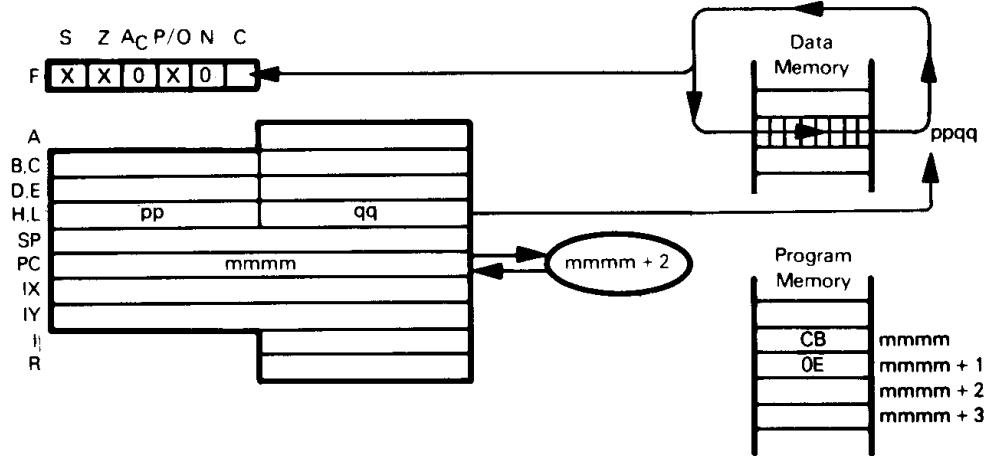
Suppose the Accumulator contains 7A<sub>16</sub> and the Carry status is set to 1. After the instruction

RRA

has executed, the Accumulator will contain BD<sub>16</sub> and the Carry status will be reset to 0:

| Before          |       | After           |       |
|-----------------|-------|-----------------|-------|
| Accumulator     | Carry | Accumulator     | Carry |
| 0 1 1 1 1 0 1 0 | 1     | 1 0 1 1 1 1 0 1 | 0     |

**RRC (HL) — ROTATE CONTENTS OF MEMORY LOCATION  
 RRC (IX+disp) RIGHT CIRCULAR  
 RRC (IY+disp)**



The illustration shows execution of RRC (HL):

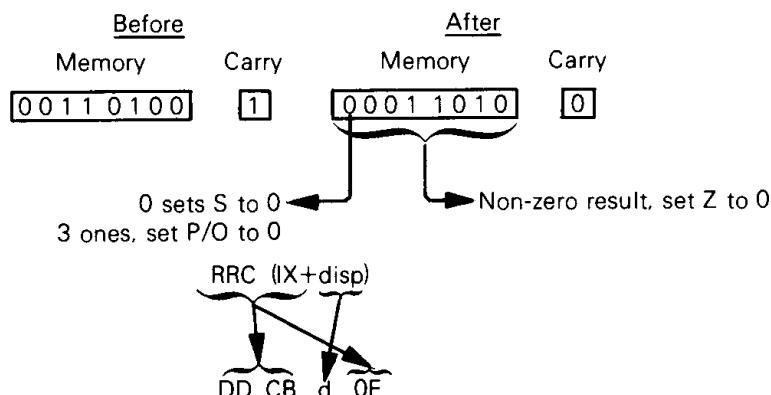
RRC (HL)  
CB 0E

Rotate contents of memory location (specified by the contents of the HL register pair) right one bit circularly, copying bit 0 into the Carry status.

Suppose the HL register pair contains 4500<sub>16</sub>, memory location 4500<sub>16</sub> contains 34<sub>16</sub>, and Carry is set to 1. After execution of

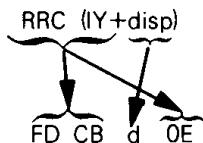
RRC (HL)

memory location 4500<sub>16</sub> will contain 1A<sub>16</sub>, and Carry will be 0:



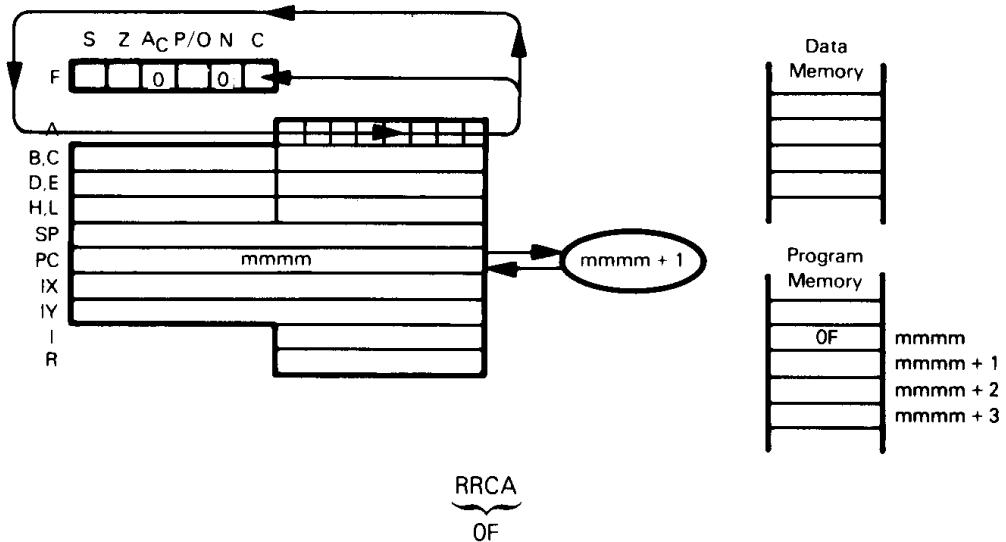
Rotate contents of memory location (specified by the sum of the contents of the IX

register and the displacement value d) right one bit circularly, copying bit 0 into the Carry status.



This instruction is identical to the RRC (IX+disp) instruction, but uses the IY register instead of the IX register.

## RRCA — ROTATE ACCUMULATOR RIGHT CIRCULAR



Rotate Accumulator contents right one bit circularly, copying bit 0 into the Carry status.

Suppose the Accumulator contains 7A<sub>16</sub> and the Carry status is set to 1. After the instruction

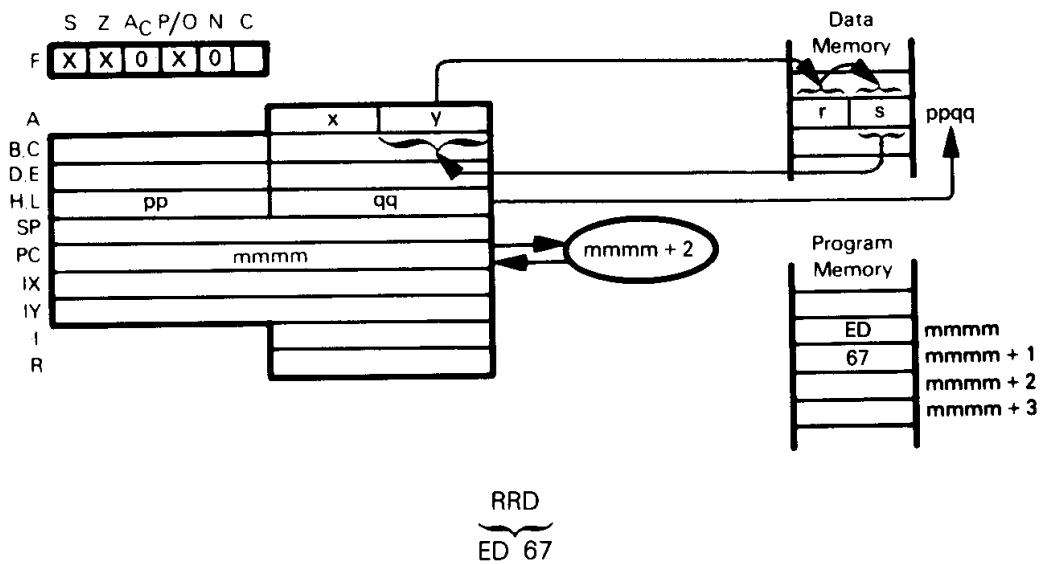
RRCA

has executed, the Accumulator will contain 3D<sub>16</sub> and the Carry status will be reset to 0:

| Before      |       | After       |       |
|-------------|-------|-------------|-------|
| Accumulator | Carry | Accumulator | Carry |
| 01111010    | 1     | 00111101    | 0     |

RRCA should be used as a logical instruction.

## RRD — ROTATE ONE BCD DIGIT RIGHT BETWEEN THE ACCUMULATOR AND MEMORY LOCATION



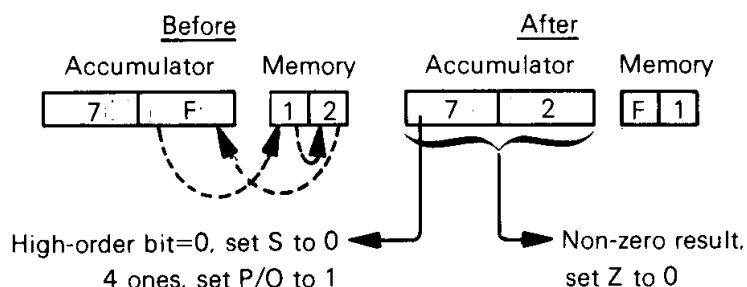
RRD  
ED 67

The four high-order bits of a memory location (specified by the contents of register pair HL) are copied into the four low-order bits of the same memory location. The previous contents of the four low-order bits are copied into the four low-order bits of the Accumulator. The previous four low-order bits of the Accumulator are copied into the four high-order bits of the specified memory location.

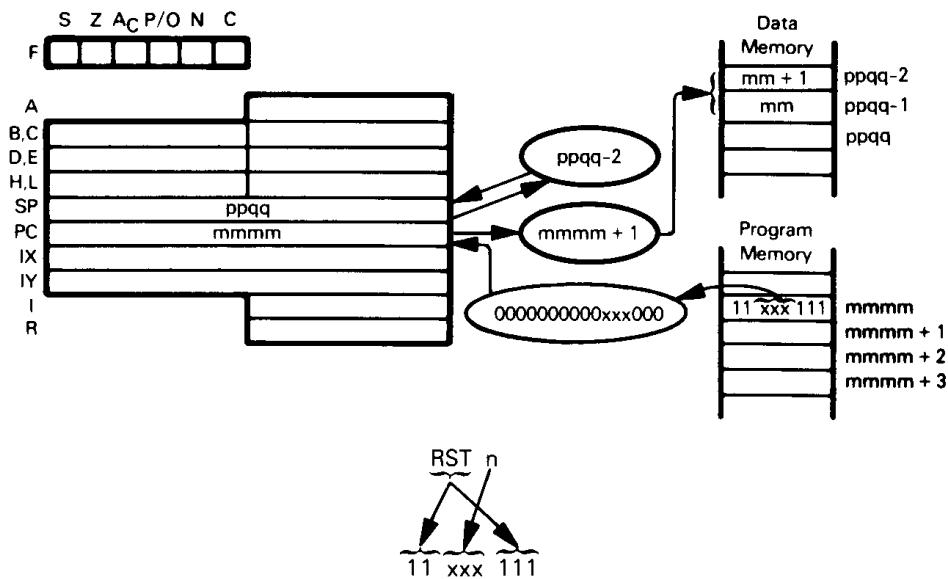
Suppose the Accumulator contains 7F<sub>16</sub>, HL register pair contains 4000<sub>16</sub>, and memory location 4000<sub>16</sub> contains 12<sub>16</sub>. After execution of the instruction

RRD

the Accumulator will contain 72<sub>16</sub> and memory location 4000<sub>16</sub> will contain F1<sub>16</sub>:



## RST n — RESTART



Call the subroutine originated at the low memory address specified by n.

When the instruction

RST 18H

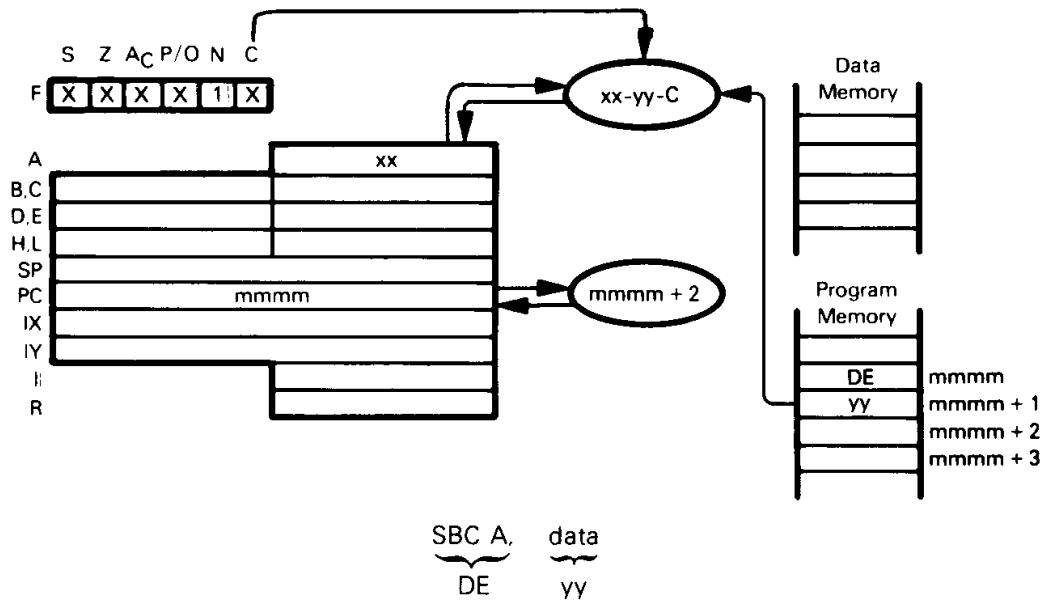
has executed, the subroutine originated at memory location 001816 is called. The previous Program Counter contents are pushed to the top of the stack.

Usually, the RST instruction is used in conjunction with interrupt processing, as described in Chapter 12.

If your application does not use all RST instruction codes to service interrupts, do not overlook the possibility of calling subroutines using RST instructions. Origin frequently used subroutines at appropriate RST addresses, and these subroutines can be called with a single-byte RST instruction instead of a three-byte CALL instruction.

**SUBROUTINE  
CALL USING  
RST**

## SBC A,data — SUBTRACT IMMEDIATE DATA FROM ACCUMULATOR WITH BORROW

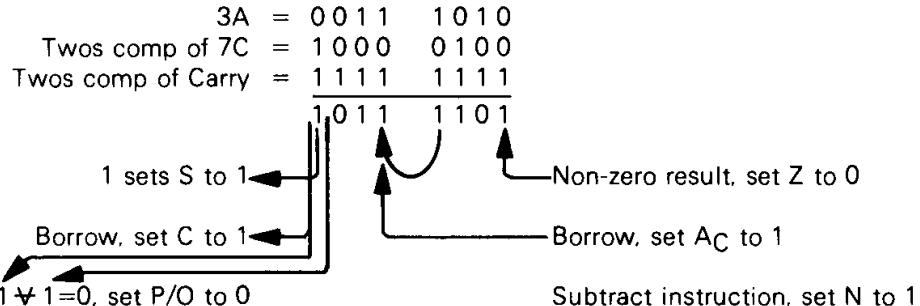


Subtract the contents of the second object code byte and the Carry status from the Accumulator.

Suppose  $xx=3A_{16}$  and  $\text{Carry}=1$ . After the instruction

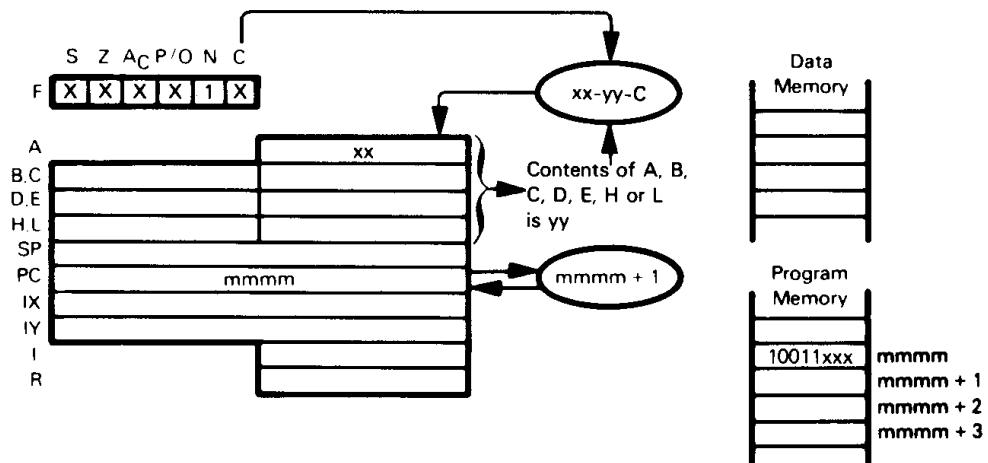
SBC A,  
DE      data  
      yy

has executed, the Accumulator will contain  $BD_{16}$ .



The Carry flag is set to 1 for a borrow and reset to 0 if there is no borrow.

## SBC A,reg — SUBTRACT REGISTER WITH BORROW FROM ACCUMULATOR



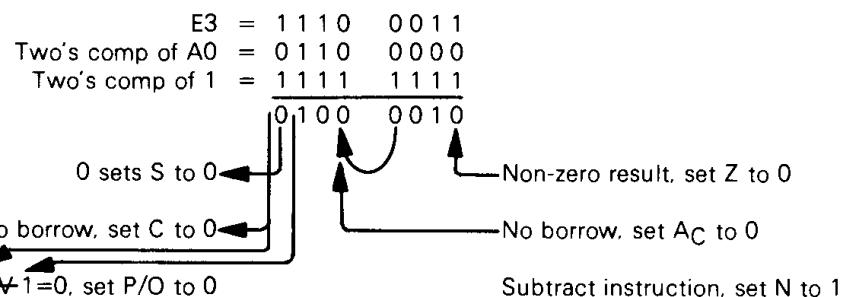
**SBC A, reg**  
10011    xxx  
 000 for reg=B  
 001 for reg=C  
 010 for reg=D  
 011 for reg=E  
 100 for reg=H  
 101 for reg=L  
 111 for reg=A

Subtract the contents of the specified register and the Carry status from the Accumulator.

Suppose  $xx = E3_{16}$ . Register E contains  $A0_{16}$ , and Carry=1. After the instruction

SBC A,E

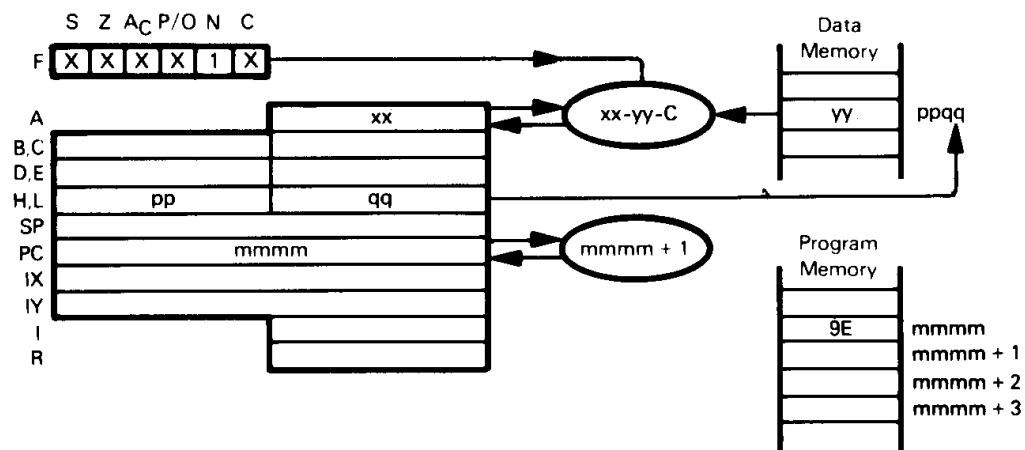
has executed, the Accumulator will contain  $42_{16}$ .



The Carry flag is set to 1 for a borrow and reset to 0 if there is no borrow.

**SBC A,(HL) —**  
**SBC A,(IX+disp)**  
**SBC A,(IY+disp)**

# **SUBTRACT MEMORY AND CARRY FROM ACCUMULATOR**



The illustration shows execution of SBC A,(HL):

SBC A,(HL)  
9E

Subtract the contents of memory location (specified by the contents of the HL register pair) and the Carry from the Accumulator.

Suppose Carry=0,  $ppqq=4000_{16}$ ,  $xx=3A_{16}$ , and memory location  $4000_{16}$  contains  $7C_{16}$ . After execution of the instruction

SBC A.(HL)

the Accumulator will contain BE16.

The diagram illustrates the subtraction process for the instruction  $3A - 7C$ . The inputs are  $3A = 0011\ 1010$  and  $7C = 1000\ 0100$ . The first stage (LSB) produces a sum of  $1011$  and a borrow-out of  $1$ . The second stage (MSB) receives this borrow and the inputs  $1110$  and  $0$ , producing a sum of  $1110$  and a borrow-out of  $1$ .

Annotations explain the logic:

- "1 sets S to 1" points to the first stage's sum output.
- "Non-zero result, set Z to 0" points to the second stage's sum output.
- "Borrow, set C to 1" points to the first stage's borrow-out.
- "Borrow, set  $A_C$  to 1" points to the second stage's borrow-out.
- " $0 \neq 0=0$ , set P/O to 0" is at the bottom left.
- "Subtract instruction, set N to 1" is at the bottom right.

The Carry flag is set to 1 for a borrow and reset to 0 if there is no borrow.

SBC A,(IX+disp)

Subtract the contents of memory location (specified by the sum of the contents of the `bx` and `cx` registers) from the contents of memory location specified by the value in `ax`.

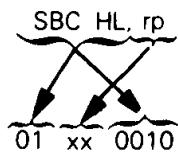
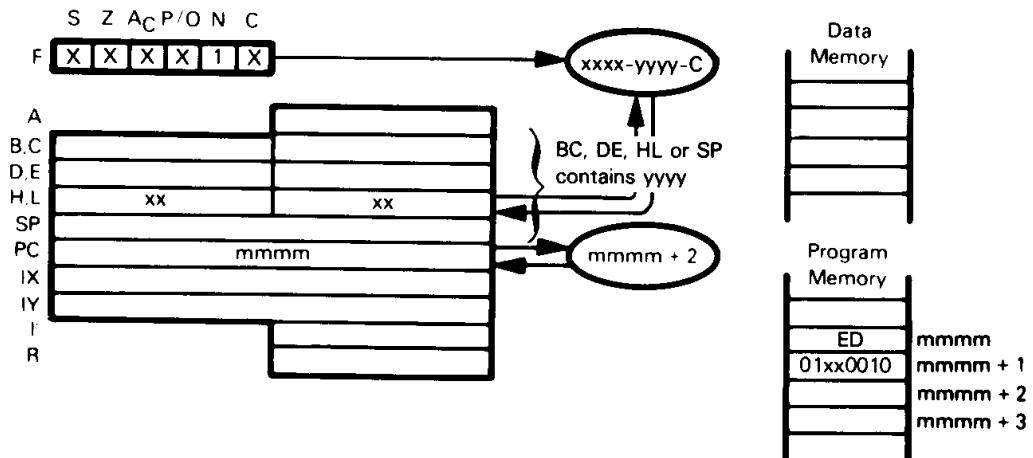
CRS-1 (V1.1)

—  
—

This instruction is identical to the SBC A,(IX+disp) instruction, except that it uses the IY register instead of the IX register.

3-147

## SBC HL, rp — SUBTRACT REGISTER PAIR WITH CARRY FROM H AND L



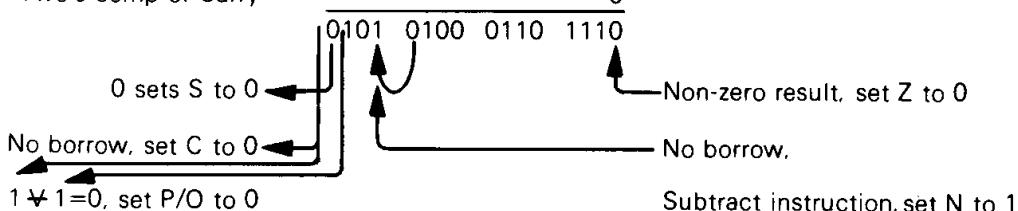
00 for rp is register pair BC  
 01 for rp is register pair DE  
 10 for rp is register pair HL  
 11 for rp is Stack Pointer

Subtract the contents of the designated register pair and the Carry status from the HL register pair.

Suppose HL contains F4A2<sub>16</sub>, BC contains A034<sub>16</sub>, and Carry=0. After the instruction  
**SBC HL,BC**

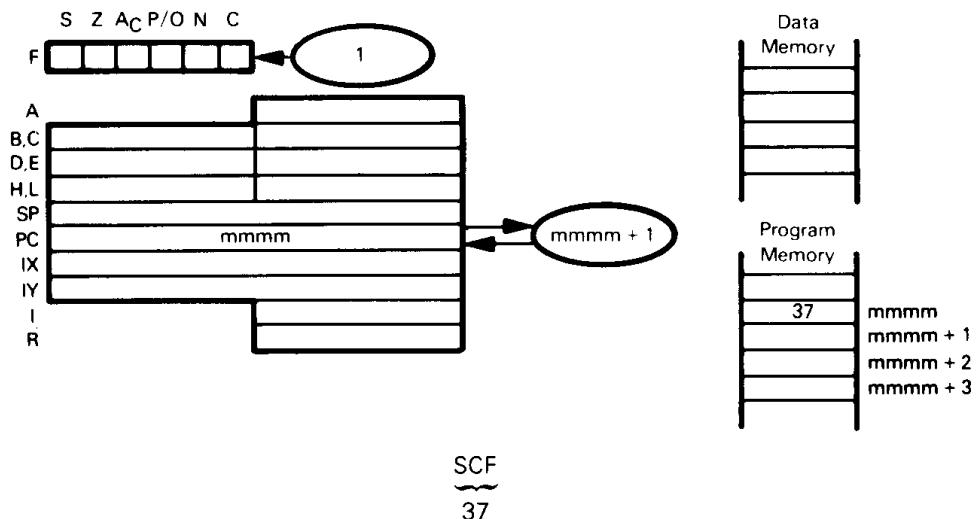
has executed, the HL register pair will contain 546E<sub>16</sub>:

Two's comp of F4A2 = 1111 0100 1010 0010  
 Two's comp of A034 = 0101 1111 1100 1100  
 Two's comp of Carry = 0



The Carry flag is set to 1 for a borrow and reset to 0 if there is no borrow.

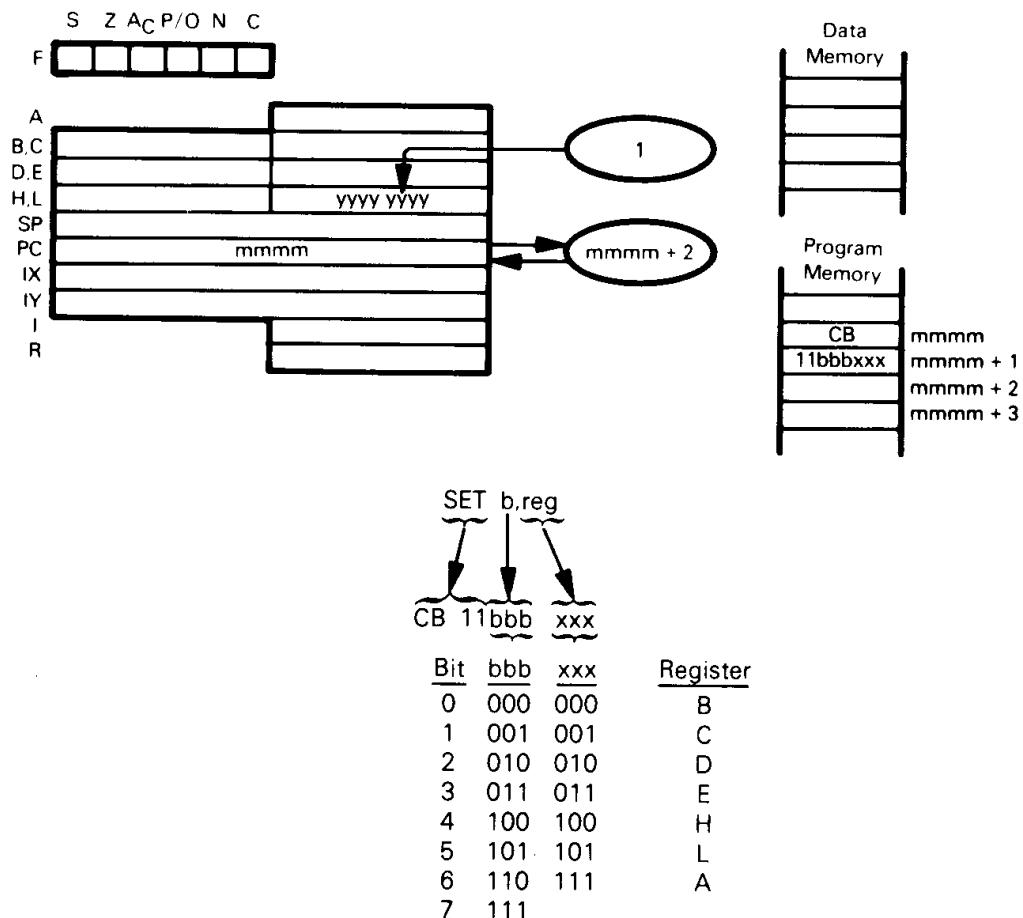
## SCF — SET CARRY FLAG



SCF  
37

When the SCF instruction is executed, the Carry status is set to 1 regardless of its previous value. No other statuses or register contents are affected.

## SET b,reg — SET INDICATED REGISTER BIT

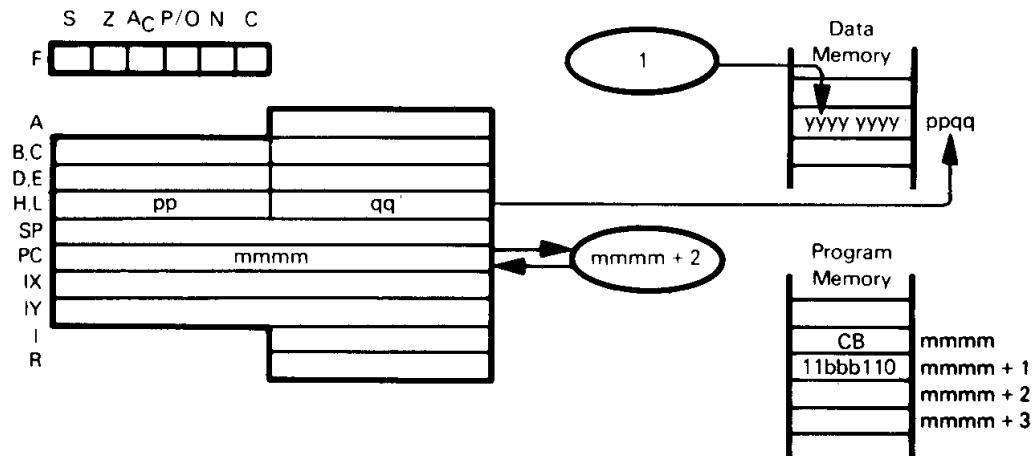


SET indicated bit within specified register. After the instruction

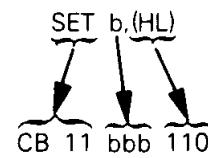
SET 2,L

has executed, bit 2 in Register L will be set. (Bit 0 is the least significant bit.)

**SET b,(HL) — SET BIT b OF INDICATED MEMORY POSITION**  
**SET b,(IX+disp)**  
**SET b,(IY+disp)**



The illustration shows execution of SET b,(HL). Bit 0 is the least significant bit.



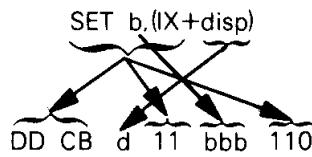
| Bit Set | <u>bbb</u> |
|---------|------------|
| 0       | 000        |
| 1       | 001        |
| 2       | 010        |
| 3       | 011        |
| 4       | 100        |
| 5       | 101        |
| 6       | 110        |
| 7       | 111        |

Set indicated bit within memory location indicated by HL.

Suppose HL contains  $400016$ . After the instruction

$\text{SET } 5,(\text{HL})$

has executed, bit 5 in memory position  $400016$  will be 1.

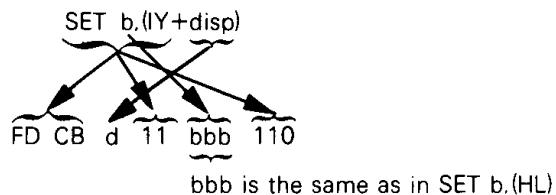


bbb is the same as in SET b,(HL)

Set indicated bit within memory location indicated by the sum of Index Register IX and displacement.

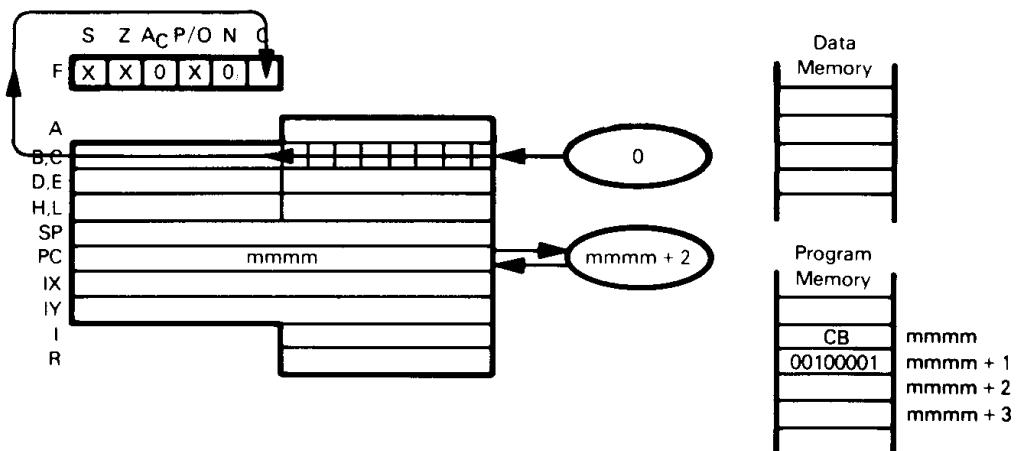
Suppose Index Register IX contains 4000<sub>16</sub>. After execution of  
SET 6,(IX+5H)

bit 6 in memory location 4005<sub>16</sub> will be 1.

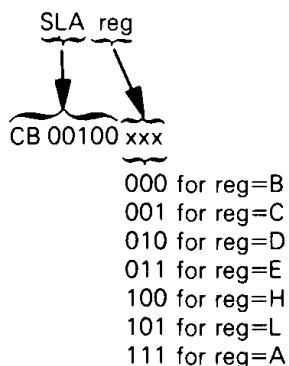


This instruction is identical to SET b,(IX+disp), except that it uses the IY register instead of the IX register.

### SLA reg — SHIFT CONTENTS OF REGISTER LEFT ARITHMETIC



The illustration shows execution of SLA C:

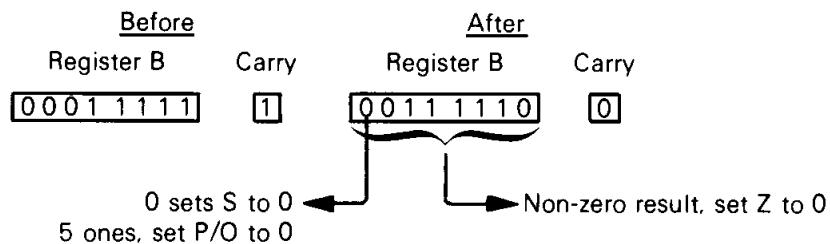


Shift contents of specified register left one bit, resetting the least significant bit to 0.

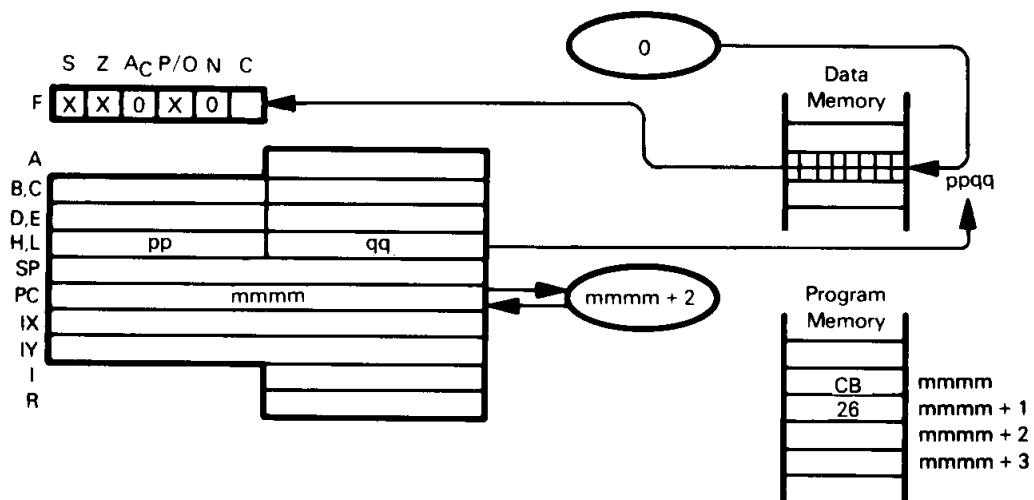
Suppose Register B contains 1F<sub>16</sub>, and Carry=1. After execution of

SLA B

Register B will contain 3E<sub>16</sub> and Carry will be zero.



**SLA (HL) — SHIFT CONTENTS OF MEMORY LOCATION**  
**SLA (IX+disp) LEFT ARITHMETIC**  
**SLA (IY+disp)**



The illustration shows execution of SLA (HL):

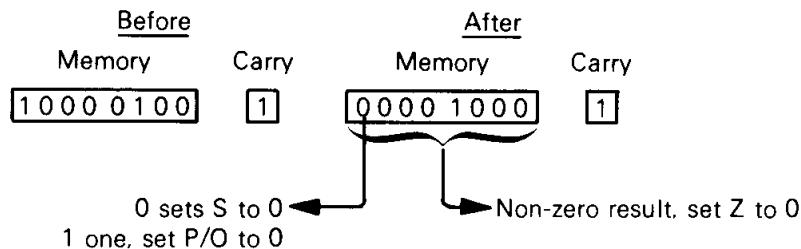
SLA (HL)  
CB 26

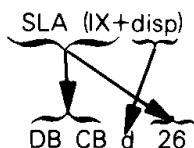
Shift contents of memory location (specified by the contents of the HL register pair) left one bit, resetting the least significant bit to 0.

Suppose the HL register pair contains `4500`<sub>16</sub>, memory location `4500`<sub>16</sub> contains `84`<sub>16</sub>, and Carry=0. After execution of

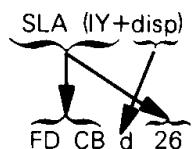
SLA (HL)

memory location `4500`<sub>16</sub> will contain `08`<sub>16</sub>, and Carry will be 1.



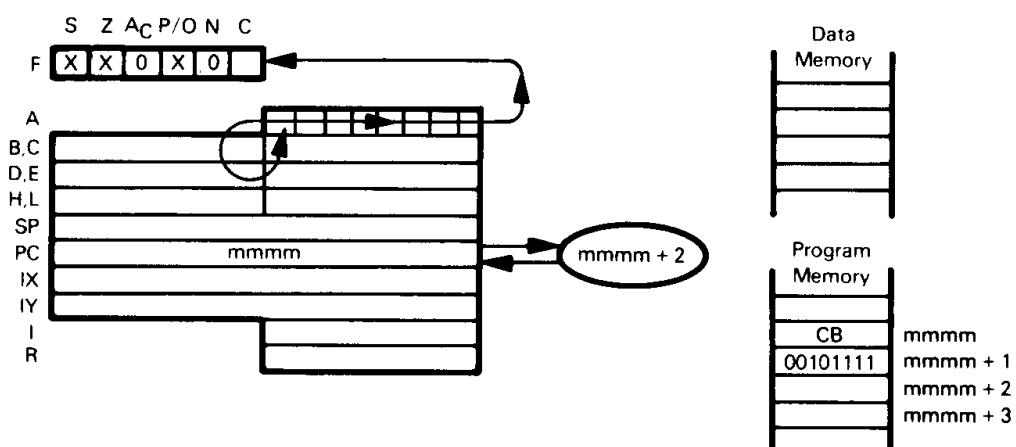


Shift contents of memory location (specified by the sum of the contents of the IX register and the displacement value d) left one bit arithmetically, resetting least significant bit to 0.

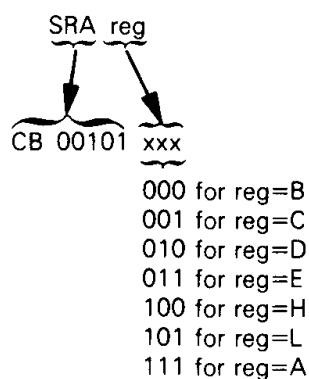


This instruction is identical to SLA (IX+disp), but uses the IY register instead of the IX register.

### SRA reg — ARITHMETIC SHIFT RIGHT CONTENTS OF REGISTER



The illustration shows execution of SRA A:

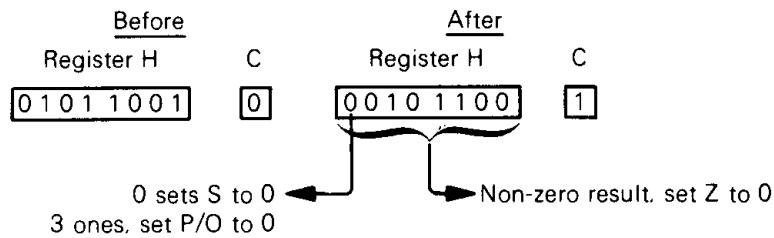


Shift specified register right one bit. Most significant bit is unchanged.

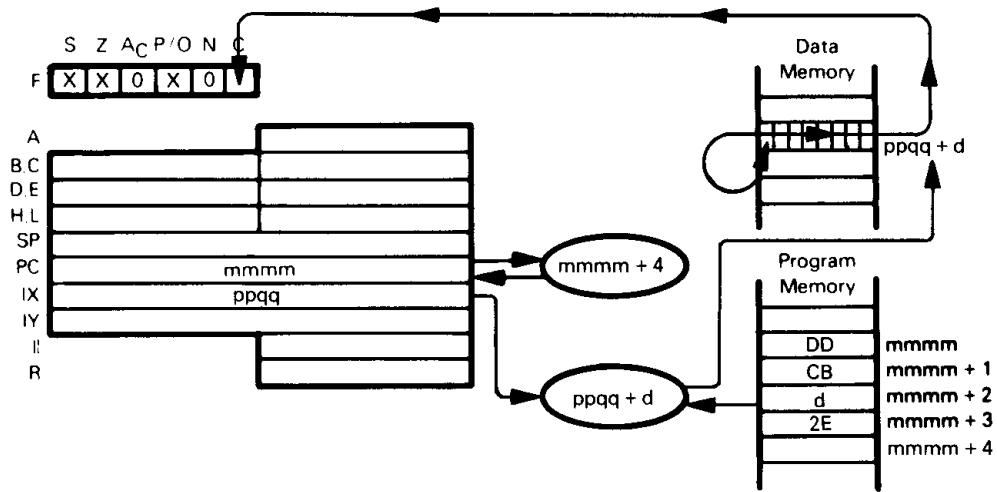
Suppose Register H contains 59<sub>16</sub>, and Carry=0. After the instruction

SRA H

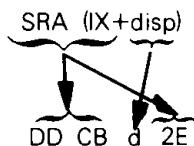
has executed, Register H will contain 2C<sub>16</sub> and Carry will be 1.



### SRA (HL) — ARITHMETIC SHIFT RIGHT CONTENTS OF MEMORY POSITION



The illustration shows execution of SRA (IX+disp):

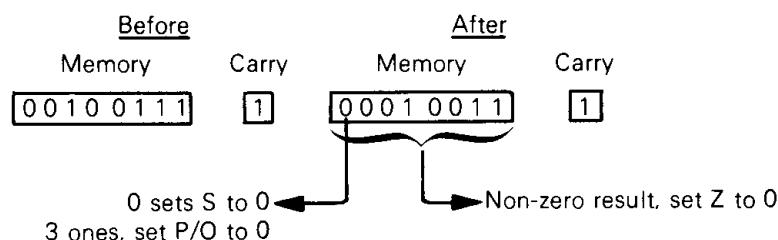


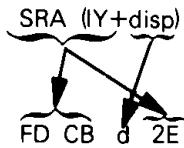
Shift contents of memory location (specified by the sum of the contents of Register IX and the displacement value d) right. Most significant bit is unchanged.

Suppose Register IX contains 340016, memory location 34AA16 contains 2716, and Carry=1. After execution of

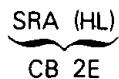
SRA (IX+0AAH)

memory location 34AA16 will contain 1316, and Carry will be 1.



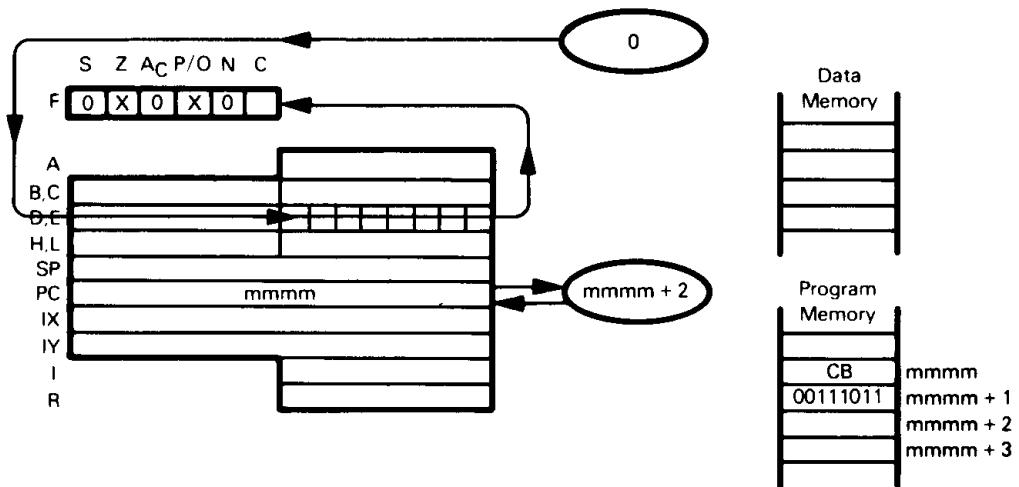


This instruction is identical to SRA (IX+disp), but uses the IY register instead of the IX register.

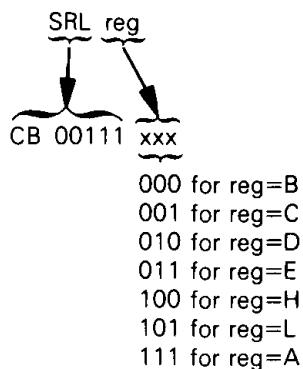


Shift contents of memory location (specified by the contents of the HL register pair) right one bit. Most significant bit is unchanged.

### SRL reg — SHIFT CONTENTS OF REGISTER RIGHT LOGICAL



The illustration shows execution of SRL E:



Shift contents of specified register right one bit. Most significant bit is reset to 0.

Suppose Register D contains  $1F_{16}$ , and Carry=0. After execution of

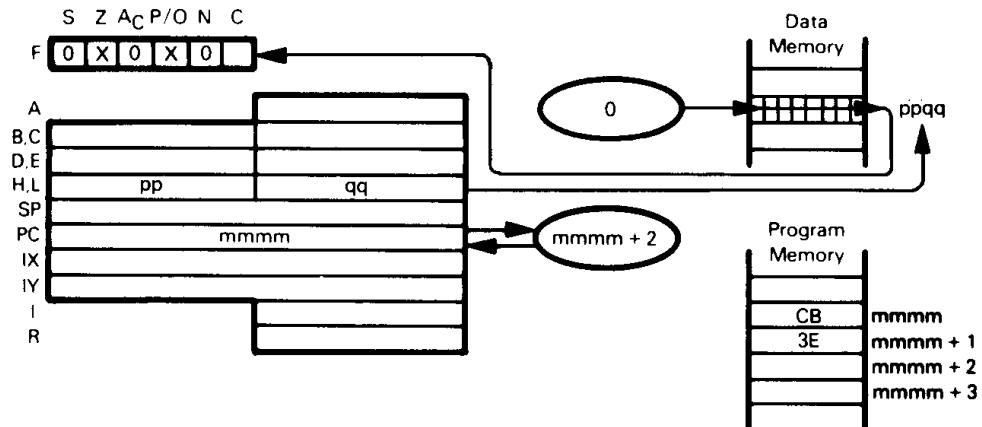
SRL D

Register D will contain  $0F_{16}$ , and Carry will be 1.

| Before     |       | After      |       |
|------------|-------|------------|-------|
| Register D | Carry | Register D | Carry |
| 00011111   | 0     | 00001111   | 1     |

4 ones, set P/O to 1      Non-zero result, set Z to 0

### SRL (HL) — SHIFT CONTENTS OF MEMORY LOCATION RIGHT LOGICAL



The illustration shows execution of SRL (HL):

SRL (HL)  
CB 3E

Shift contents of memory location (specified by the contents of the HL register pair) right one bit. Most significant bit is reset to 0.

Suppose the HL register pair contains 2000<sub>16</sub>, memory location 2000<sub>16</sub> contains 8F<sub>16</sub>, and Carry=0. After execution of

SRL (HL)

memory location 2000<sub>16</sub> will contain 47<sub>16</sub>, and Carry will be 1.

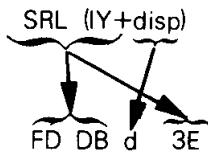
| Before   |       | After    |       |
|----------|-------|----------|-------|
| Memory   | Carry | Memory   | Carry |
| 10001111 | 0     | 01000111 | 1     |

4 ones, set P/O to 1      Non-zero result, set Z to 0

SRL (IX+disp)

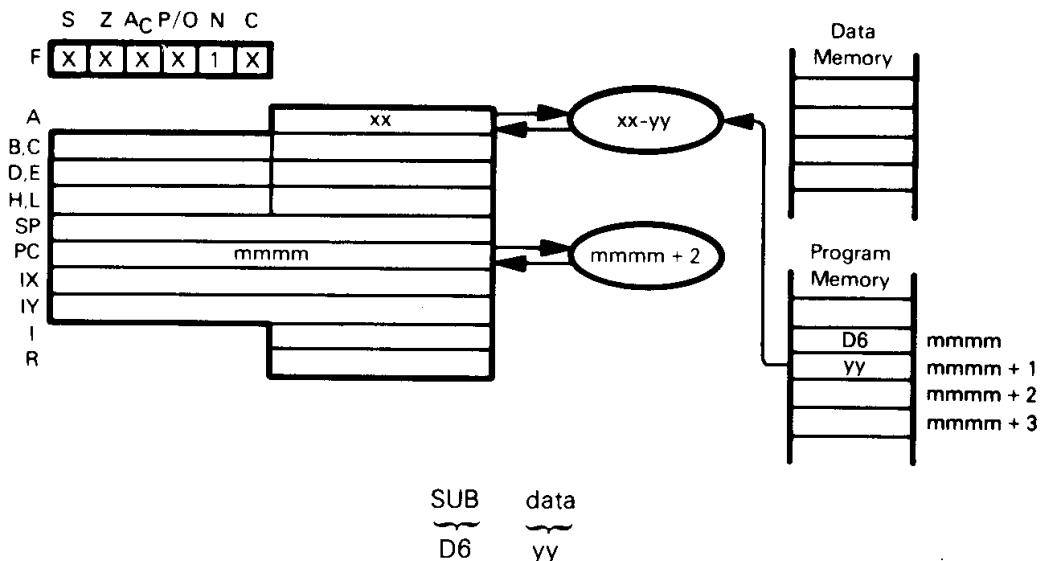
DD CB d 3E

Shift contents of memory location (specified by the sum of the contents of the IX register and the displacement value d) right one bit. Most significant bit is reset to 0.



This instruction is identical to SRL (IX+disp), but uses the IY register instead of the IX register.

## SUB data — SUBTRACT IMMEDIATE FROM ACCUMULATOR

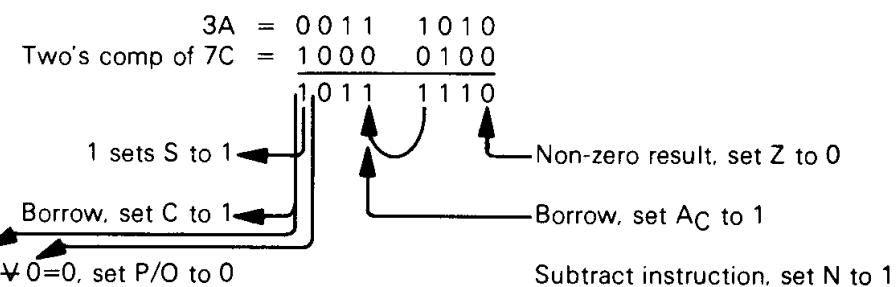


Subtract the contents of the second object code byte from the Accumulator.

Suppose  $xx=3A_{16}$ . After the instruction

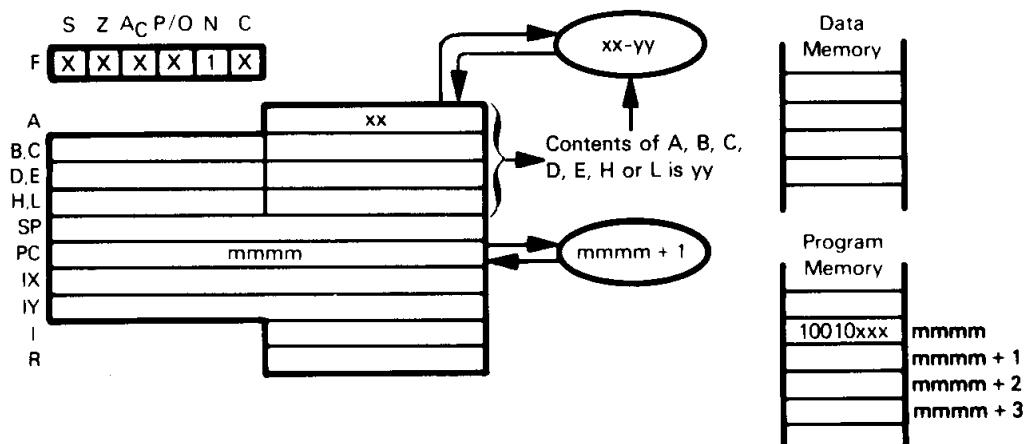
SUB 7CH

has executed, the Accumulator will contain  $BE_{16}$ .



Notice that the resulting carry is complemented.

## SUB reg — SUBTRACT REGISTER FROM ACCUMULATOR



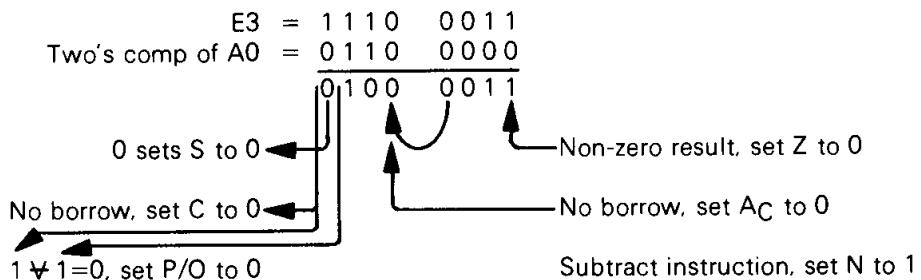
| <u>SUB</u> | <u>reg</u>    |
|------------|---------------|
| 10010      | xxx           |
|            | 000 for reg=B |
|            | 001 for reg=C |
|            | 010 for reg=D |
|            | 011 for reg=E |
|            | 100 for reg=H |
|            | 101 for reg=L |
|            | 111 for reg=A |

Subtract the contents of the specified register from the Accumulator.

Suppose xx=E3 and Register H contains A016. After execution of

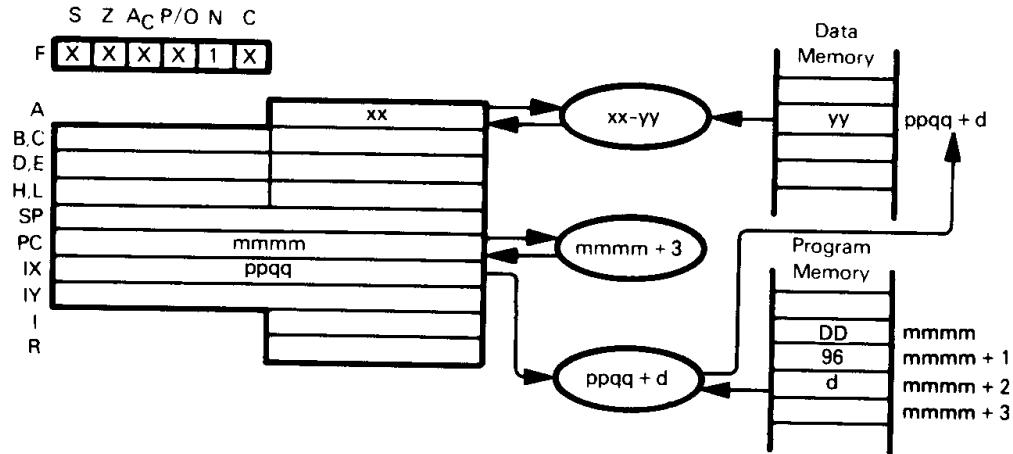
SUB H

the Accumulator will contain 4316.



Notice that the resulting carry is complemented.

**SUB (HL) — SUBTRACT MEMORY FROM ACCUMULATOR**  
**SUB (IX+disp)**  
**SUB (IY+disp)**



The illustration shows execution of SUB (IX+disp):

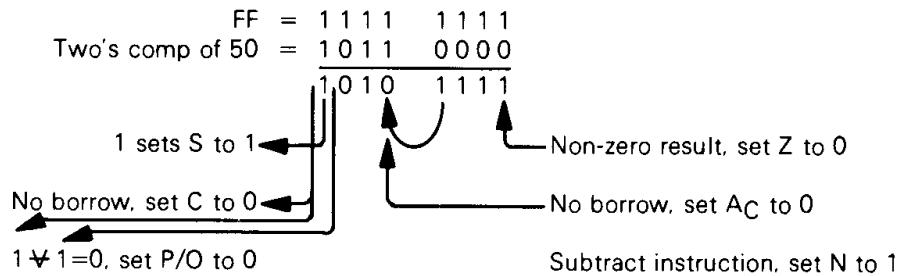
SUB (IX+disp)  
DD 96 d

Subtract contents of memory location (specified by the sum of the contents of the IX register and the displacement value d) from the Accumulator.

Suppose ppqq=4000<sub>16</sub>, xx=FF16, and memory location 40FF16 contains 50<sub>16</sub>. After execution of

SUB (IX+OFFH)

the Accumulator will contain AF16.



Notice that the resulting carry is complemented.

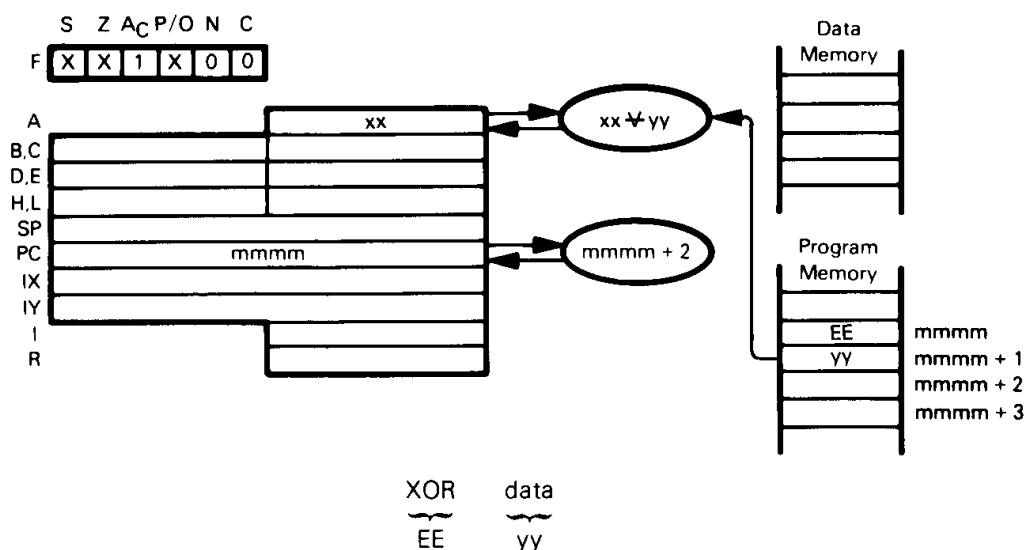
SUB (IY+disp)  
FD 96 d

This instruction is identical to SUB (IX+disp), except that it uses the IY register instead of the IX register.

SUB (HL)  
96

Subtract contents of memory location (specified by the contents of the HL register pair) from the Accumulator.

## XOR data — EXCLUSIVE-OR IMMEDIATE WITH ACCUMULATOR



Exclusive-OR the contents of the second object code byte with the Accumulator.

Suppose xx=3A16. After the instruction

XOR 7CH

has executed, the Accumulator will contain 4616.

$$\begin{array}{r}
 3A = 0011\ 1010 \\
 7C = 0111\ 1100 \\
 \hline
 0100\ 0110
 \end{array}$$

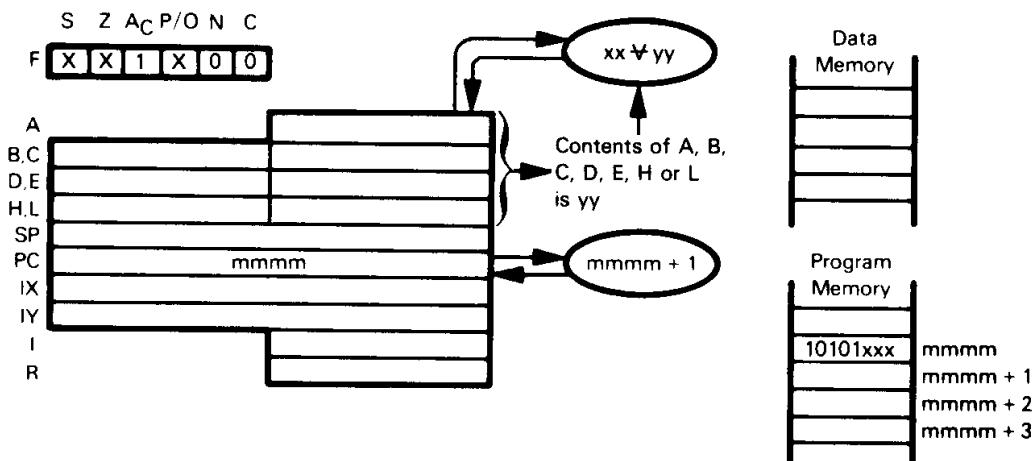
Non-zero result, set Z to 0

Three 1 bits, set P/O to 0

0 sets S to 0

The Exclusive-OR instruction is used to test for changes in bit status.

## XOR reg — EXCLUSIVE-OR REGISTER WITH ACCUMULATOR



|                                         |                                                                                                                                                              |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\overbrace{\text{XOR}}^{\text{10101}}$ | $\overbrace{\text{reg}}^{\text{xxx}}$<br>000 for reg=B<br>001 for reg=C<br>010 for reg=D<br>011 for reg=E<br>100 for reg=H<br>101 for reg=L<br>111 for reg=A |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|

Exclusive-OR the contents of the specified register with the Accumulator.

Suppose xx=E316 and Register E contains A016. After the instruction

XOR E

has executed, the Accumulator will contain 4316.

$$\begin{array}{r}
 \text{E3} = 1110 \quad 0011 \\
 \text{A0} = 1010 \quad 0000 \\
 \hline
 0100 \quad 0011
 \end{array}$$

0 sets S to 0

Non-zero result, set Z to 0

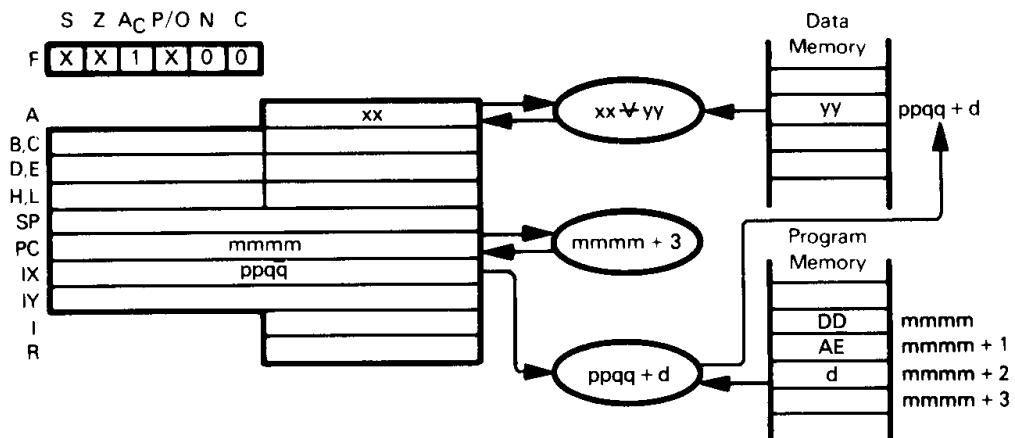
Three 1 bits, set P/O to 0

The Exclusive-OR instruction is used to test for changes in bit status.

## XOR (HL) — EXCLUSIVE-OR MEMORY WITH ACCUMULATOR

**XOR (IX+disp)**

**XOR (IY+disp)**



The illustration shows execution of XOR (IX+disp):

XOR (IX+disp)

DD AE d

Exclusive-OR contents of memory location (specified by the sum of the contents of the IX register and the displacement value d) with the Accumulator.

Suppose xx=E316, ppqq=450016, and memory location 45FF16 contains A016. After the instruction

XOR (IX+0FFH)

has executed, the Accumulator will contain 4316.

$$\begin{array}{r}
 E3 = 1110\ 0011 \\
 A0 = 1010\ 0000 \\
 \hline
 0100\ 0011
 \end{array}$$

0 sets S to 0

Non-zero result, set Z to 0

Three 1 bits, set P/O to 0

XOR (IY+disp)

FD AE d

This instruction is identical to XOR (IX+disp), except that it uses the IY register instead of the IX register.

XOR (HL)  
AE

Exclusive-OR contents of memory location (specified by the contents of the HL register pair) with the Accumulator.

## **8080A/Z80 COMPATIBILITY**

**Although the Z80 microprocessor can certainly be used on its own merits, one of its important characteristics is its compatibility with the 8080A microprocessor.** This compatibility has the following features:

- 1) All 8080A machine language instructions are also Z80 machine language instructions.
- 2) All 8080A registers are also Z80 registers (see Table 3-6).
- 3) Almost all 8080A programs will run on a Z80, with some minor differences to be noted later.
- 4) The Z80 has instructions, registers, and other features not present on the 8080A, so Z80 programs will not generally run on 8080A processors.

**Note that this compatibility does not extend to assembly language source statements** since Z80 assemblers and 8080A assemblers use different operation code mnemonics. **Table 3-7 contains a list of the 8080A mnemonic codes and the corresponding Z80 codes, while Table 3-8 is the same list organized by Z80 codes.**

Readers should note the binary coding limitations that this compatibility places on the extra features of the Z80 microprocessor. The 8080A has some unused operation codes (see Table 3-9) that are used for some of the Z80's extra instructions. But there are simply not enough such codes to cover the large number of features in a simple form.

Thus, many of the added Z80 instructions require a 2-byte operation code. The first byte is CB, DD, ED, or FD. Note the following meanings of these codes from Table 3-9:

- CB — a register or bit operation  
DD — an operation involving register IX  
ED — a miscellaneous non-8080A instruction not covered elsewhere  
FD — an operation involving register IY

The second byte of the operation code describes the actual operation to be performed.

The end result is that these multi-byte instructions execute rather slowly (and use more memory) because an additional memory access is required. The reader should be aware of this variation in execution times and try to use faster executing instructions when possible. This warning particularly applies to the extra shift instructions (RLC, RRC, RL, RR, SRA, SRL) and to instructions involving the index registers IX and IY.

**There are a few minor incompatibilities between the 8080A and the Z80.** These are:

- 1) The Z80 uses the P (or P/O) flag to indicate two's complement overflow after arithmetic operations. The 8080A always uses this flag for parity.
- 2) The Z80 and 8080A execute the DAA instruction differently. On the Z80, this instruction will correct decimal subtraction as well as decimal addition. On the 8080A, it will correct only decimal addition.
- 3) The Z80 rotate instructions clear the AC flag. The 8080A rotate instructions do not affect the AC flag.

**8080A/Z80 COMPATIBILITY FEATURES**

**8080A/Z80 ASSEMBLY LEVEL CONVERSION**

**8080A UNUSED OPERATION CODES**

**2-BYTE OPERATION CODES**

**FASTER AND SLOWER EXECUTING INSTRUCTIONS**

**8080A/Z80 INCOMPATIBILITIES**

Table 3-6. Register and Flag Correspondence between Z80 and 8080A

| Z80 Register          | 8080A Register                |
|-----------------------|-------------------------------|
| A                     | A                             |
| A'                    | None                          |
| B                     | B                             |
| B'                    | None                          |
| C                     | C                             |
| C'                    | None                          |
| D                     | D                             |
| D'                    | None                          |
| E                     | E                             |
| E'                    | None                          |
| F                     | Least Significant Half of PSW |
| F'                    | None                          |
| H                     | H                             |
| H'                    | None                          |
| I                     | None                          |
| IX                    | None                          |
| IY                    | None                          |
| L                     | L                             |
| L'                    | None                          |
| R                     | None                          |
| PC                    | PC                            |
| SP                    | SP                            |
| Z80 Register Pairs    | 8080A Register Pairs          |
| BC                    | B                             |
| DE                    | D                             |
| HL                    | H                             |
| AF                    | PSW                           |
| Z80 Flags             | 8080A Flags                   |
| C (Carry)             | C (Carry)                     |
| H (Half-Carry)        | AC (Auxiliary Carry)          |
| N (Subtract)          | None                          |
| P/O (Parity/Overflow) | P (Parity)                    |
| S (Sign)              | S (Sign)                      |
| Z (Zero)              | Z (Zero)                      |

The Z80 is not compatible with the extra features of the 8085 microprocessor. The codes used for RIM and SIM on the 8085 are used for relative jumps (NZ and NC) on the Z80.

Instruction timings on the 8080A, 8085, and Z80 all differ. Programs that depend on precise instruction timings will therefore execute properly only on the processor for which they were written.

The N flag on the Z80 occupies bit 2 of the F register; the corresponding bit in the Processor Status Word of the 8080A is always a logic '1'.

#### 8085/Z80 INCOMPATIBILITIES

#### TIMING INCOMPATIBILITIES

Table 3-7 Correspondence between 8080A and Z80 Mnemonics

| 8080A Mnemonic | Z80 Mnemonic      | 8080A Mnemonic    | Z80 Mnemonic        |
|----------------|-------------------|-------------------|---------------------|
| ACI data       | ADC A,data        | LHLD addr         | LD HL,(addr)        |
| ADC reg or M   | ADC A,reg or (HL) | LXI rp,data16     | LD rp,data16        |
| ADD reg or M   | ADD A,reg or (HL) | MOV reg,reg or M  | LD reg,reg or (HL)  |
| ADI data       | ADD A,data        | MOV reg or M,reg  | LD reg or (HL),reg  |
| ANA reg or M   | AND reg or (HL)   | MVI reg or M,data | LD reg or (HL),data |
| ANI data       | AND data          | NOP               | NOP                 |
| CALL addr      | CALL addr         | ORA reg or M      | OR reg or (HL)      |
| CC addr        | CALL C,addr       | ORI data          | OR data             |
| CM addr        | CALL M,addr       | OUT port          | OUT (port),A        |
| CMA            | CPL               | PCHL              | JP (HL)             |
| CMC            | CCF               | POP pr            | POP pr              |
| CMP reg or M   | CP reg or (HL)    | PUSH pr           | PUSH pr             |
| CNC addr       | CALL NC,addr      | RAL               | RLA                 |
| CNZ addr       | CALL NZ,addr      | RAR               | RRA                 |
| CP addr        | CALL P,addr       | RC                | RET C               |
| CPE addr       | CALL PE,addr      | RET               | RET                 |
| CPI data       | CP data           | RLC               | RLCA                |
| CPO addr       | CALL PO,addr      | RM                | RET M               |
| CZ addr        | CALL Z,addr       | RNC               | RET NC              |
| DAA            | DAA               | RNZ               | RET NZ              |
| \$ DAD rp      | ADD HL, rp        | RP                | RET P               |
| DCR reg or M   | DEC reg or (HL)   | RPE               | RET PE              |
| DCX rp         | DEC rp            | RPO               | RET PO              |
| DI             | DI                | RRCA              |                     |
| EI             | EI                | RST n             | RST n               |
| HLT            | HALT              | RZ                | RET Z               |
| IN port        | IN A,(port)       | SBB reg or M      | SBC A,reg or (HL)   |
| INR reg or M   | INC reg or (HL)   | SBI data          | SBC A,data          |
| INX rp         | INC rp            | SHLD addr         | LD (addr),HL        |
| JC addr        | JP C,addr         | SPHL              | LD SP,HL            |
| JM addr        | JP M,addr         | STA addr          | LD (addr),A         |
| JMP addr       | JP addr           | STAX B or D       | LD (BC) or (DE),A   |
| JNC addr       | JP NC,addr        | STC               | SCF                 |
| JP addr        | JP P,addr         | SUB reg or M      | SUB reg or (HL)     |
| JNZ addr       | JP NZ,addr        | SUI data          | SUB data            |
| JPE addr       | JP PE,addr        | XCHG              | EX DE,HL            |
| JPO addr       | JP PO,addr        | XRA reg or M      | XOR reg or (HL)     |
| JZ addr        | JP Z,addr         | XRI data          | XOR data            |
| LDA addr       | LD A,(addr)       | XTHL              | EX (SP),HL          |
| LDAX B or D    | LD A,(BC) or (DE) |                   |                     |

Table 3-8. Correspondence between Z80 and 8080A Mnemonics

| Z80 Mnemonic      | 8080A Mnemonic | Z80 Mnemonic        | 8080A Mnemonic |
|-------------------|----------------|---------------------|----------------|
| ADC A,data        | ACI data       | INC rp              | INX rp         |
| ADC A,(HL)        | ADC M          | INC xy              | —              |
| ADC A,reg         | ADC reg        | INC (xy + disp)     | —              |
| ADC A,(xy + disp) | —              | IND                 | —              |
| ADC HL,rp         | —              | INDR                | —              |
| ADD A,data        | ADI data       | INI                 | —              |
| ADD A,(HL)        | ADD M          | INIR                | —              |
| ADD A,reg         | ADD reg        | JP addr             | JMP addr       |
| ADD A,(xy + disp) | —              | JP C,addr           | JC addr        |
| ADD HL,rp         | DAD rp         | JP (HL)             | PCHL           |
| ADD IX,pp         | —              | JP M,addr           | JM addr        |
| ADD IY,rr         | —              | JP NC,addr          | JNC addr       |
| AND data          | ANI data       | JP NZ,addr          | JNZ addr       |
| AND (HL)          | ANA M          | JP P,addr           | JP addr        |
| AND reg           | ANA reg        | JP PE,addr          | JPE addr       |
| AND (xy + disp)   | —              | JP PO,addr          | JPO addr       |
| BIT b,(HL)        | —              | JP Z,addr           | JZ addr        |
| BIT b,reg         | —              | JP xy               | —              |
| BIT b,(xy + disp) | —              | JR C,disp           | —              |
| CALL addr         | CALL addr      | JR disp             | —              |
| CALL C,addr       | CC addr        | JR NC,disp          | —              |
| CALL M,addr       | CM addr        | JR NZ,disp          | —              |
| CALL NC,addr      | CNC addr       | JR Z,disp           | —              |
| CALL NZ,addr      | CNZ addr       | LD A,(addr)         | LDA addr       |
| CALL P,addr       | CP addr        | LD A,(BC) or (DE)   | LDAX B or D    |
| CALL PE,addr      | CPE addr       | LD A,I              | —              |
| CALL PO,addr      | CPO addr       | LD A,R              | —              |
| CALL Z,addr       | CZ addr        | LD (addr),A         | STA addr       |
| CCF               | CMC            | LD (addr),BC or DE  | —              |
| CP data           | CPI data       | LD (addr),HL        | SHLD addr      |
| CP (HL)           | CMP M          | LD (addr),SP        | —              |
| CP reg            | CMP reg        | LD (addr),xy        | —              |
| CP (xy + disp)    | —              | LD (BC) or (DE),A   | STAX B or D    |
| CPD               | —              | LD BC or DE,(addr)  | —              |
| CPDR              | —              | LD HL,(addr)        | LHLD addr      |
| CPI               | —              | LD (HL),data        | MVI M,data     |
| CPIR              | —              | LD (HL),reg         | MOV M,reg      |
| CPL               | CMA            | LD I,A              | —              |
| DAA               | DAA            | LD R,A              | —              |
| DEC (HL)          | DCR M          | LD reg,data         | MVI reg,data   |
| DEC reg           | DCR reg        | LD reg,(HL)         | MOV reg,M      |
| DEC rp            | DCX rp         | LD reg,reg          | MOV reg,reg    |
| DEC xy            | —              | LD reg,(xy + disp)  | —              |
| DEC (xy + disp)   | —              | LD rp,data16        | LXI rp,data16  |
| DI                | DI             | LD SP,(addr)        | —              |
| DJNZ disp         | —              | LD SP,HL            | SPHL           |
| EI                | EI             | LD SP,xy            | —              |
| EX AF,AF'         | —              | LD xy,data16        | —              |
| EX DE,HL          | XCHG           | LD xy,(addr)        | —              |
| EX (SP),HL        | XTHL           | LD (xy + disp),data | —              |
| EX (SP),xy        | —              | LD (xy + disp),reg  | —              |
| EXX               | —              | LDDD                | —              |
| HALT              | HLT            | LDI                 | —              |
| IM m              | —              | LDDIR               | —              |
| IN A,(port)       | IN port        | NEG                 | —              |
| IN reg,(C)        | —              | NOP                 | —              |
| INC (HL)          | INR M          | OR data             | ORI data       |
| INC reg           | INR reg        |                     |                |

— indicates that there is no corresponding instruction.

Table 3-8. Correspondence between Z80 and 8080A Mnemonics (Continued)

| Z80 Mnemonic      | 8080A Mnemonic | Z80 Mnemonic      | 8080A Mnemonic |
|-------------------|----------------|-------------------|----------------|
| OR (HL)           | ORA M          | RR (HL)           | —              |
| OR reg            | ORA reg        | RR reg            | —              |
| OR (xy + disp)    | —              | RR (xy + disp)    | —              |
| OTDR              | —              | RRA               | RAR            |
| OTIR              | —              | RRC (HL)          | —              |
| OUT (C),reg       | —              | RRC reg           | —              |
| OUT (port),A      | OUT port       | RRC (xy + disp)   | —              |
| OUTD              | —              | RRCA              | RRC            |
| OUTI              | —              | RRD               | —              |
| POP pr            | POP pr         | RST n             | RST n          |
| POP xy            | —              | SBC A,data        | SBI data       |
| PUSH pr           | PUSH pr        | SBC A,(HL)        | SBB M          |
| PUSH xy           | —              | SBC A,reg         | SBB reg        |
| RES b,(HL)        | —              | SBC A,(xy + disp) | —              |
| RES b,reg         | —              | SBC HL,pr         | —              |
| RES b,(xy + disp) | —              | SCF               | STC            |
| RET               | RET            | SET b,(HL)        | —              |
| RET C             | RC             | SET b,reg         | —              |
| RET M             | RM             | SET b,(xy + disp) | —              |
| RET NC            | RNC            | SLA (HL)          | —              |
| RET NZ            | RNZ            | SLA reg           | —              |
| RET P             | RP             | SLA (xy + disp)   | —              |
| RET PE            | RPE            | SRA (HL)          | —              |
| RET PO            | RPO            | SRA reg           | —              |
| RET Z             | RZ             | SRA (xy + disp)   | —              |
| RETI              | —              | SRL (HL)          | —              |
| RETN              | —              | SRL reg           | —              |
| RL (HL)           | —              | SRL (xy + disp)   | —              |
| RL reg            | —              | SUB data          | SUI data       |
| RL (xy + disp)    | —              | SUB (HL)          | SUB M          |
| RLA               | RAL            | SUB reg           | SUB reg        |
| RLC (HL)          | —              | SUB (xy + disp)   | —              |
| RLC reg           | —              | XOR data          | XRI data       |
| RLC (xy + disp)   | —              | XOR (HL)          | XRA M          |
| RLCA              | RLC            | XOR reg           | XRA reg        |
| RLD               | —              | XOR (xy + disp)   | —              |

— indicates that there is no corresponding instruction

Table 3-9. Unused 8080A Operation Codes and Their Z80 Meanings

| <b>8080A Operation Code</b> | <b>Z80 Use</b>                                                                                                                                                                                                                      |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 08                          | EX AF,AF'                                                                                                                                                                                                                           |
| 10                          | DJN7 disp                                                                                                                                                                                                                           |
| 18                          | JR disp                                                                                                                                                                                                                             |
| 20 (RIM on 8085)            | JR NZ,disp                                                                                                                                                                                                                          |
| 28                          | JR Z,disp                                                                                                                                                                                                                           |
| 30 (SIM on 8085)            | JR NC,disp                                                                                                                                                                                                                          |
| 38                          | JR C,disp                                                                                                                                                                                                                           |
| CB                          | BIT, RES, RL, RLC, RR, RRC, SET, SLA, SRA, SRL                                                                                                                                                                                      |
| D9                          | EXX                                                                                                                                                                                                                                 |
| DD                          | All instructions involving Register IX.                                                                                                                                                                                             |
| ED                          | ADC HL,rp LD A,I NEG<br>CPD LD A,R OTDR<br>CPDR LD (addr),rp OTIR<br>CPI LD I,A OUT (C),reg<br>CPIR LD R,A OUTD<br>IM m LD rp,(addr) OUTI<br>IN reg,(C) LDD RETI<br>IND LDDR RETN<br>INDR LDI RLD<br>INI LDIR RRD<br>INIR SBC HL,rp |
| FD                          | All instructions involving Register IY.                                                                                                                                                                                             |