# Chapter 7
# Programming Techniques with Additional Instructions

# Register Pairs

- The 8085 is an 8-bit microprocessor. So, its general purpose registers are 8-bits wide.
  - But, it uses 16-bit addresses.
- To allow manipulation of the 16-bit addresses, the 8085 allows its registers to be paired.
  - In other words, it is possible to combine two 8-bit registers together into a 16-bit super register or a register pair.
- The 8085 recognizes only 3 register pairs: B and C, D and E and H and L.
  - It is **not** possible to pair B and E for example.

# Identifying Register Pairs

- A register pair is identified with the name of the first register in the pair.
- Therefore, the three register pairs are called:
  - B for the B,C pair
  - D for the D,E pair
  - H for the H,L pair

# Placing 16-bit Data into a Register Pair

- To place a 16-bit number into a register pair, we can place an 8-bit number in each of the two registers.

```
MVI    L 00H
MVI    H 32H
```

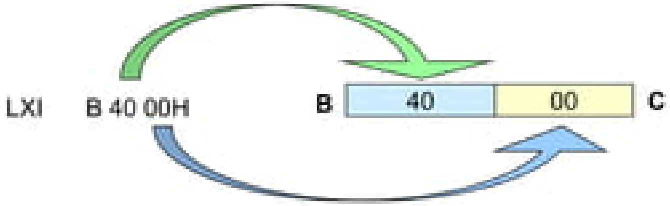
  - But, in what order?

```
MVI    L 00H      or      MVI    L 32H
MVI    H 32H              MVI    H 00H
```

# The LXI instruction

- The 8085 provides an instruction to place the 16-bit data into the register pair in one step.
  - **LXI Rp, <16-bit address>** (Load eXtended Immediate)

- The instruction **LXI B 4000H** will place the 16-bit number 4000 into the register pair B, C.
  - The upper two digits are placed in the 1st register of the pair and the lower two digits in the 2nd .

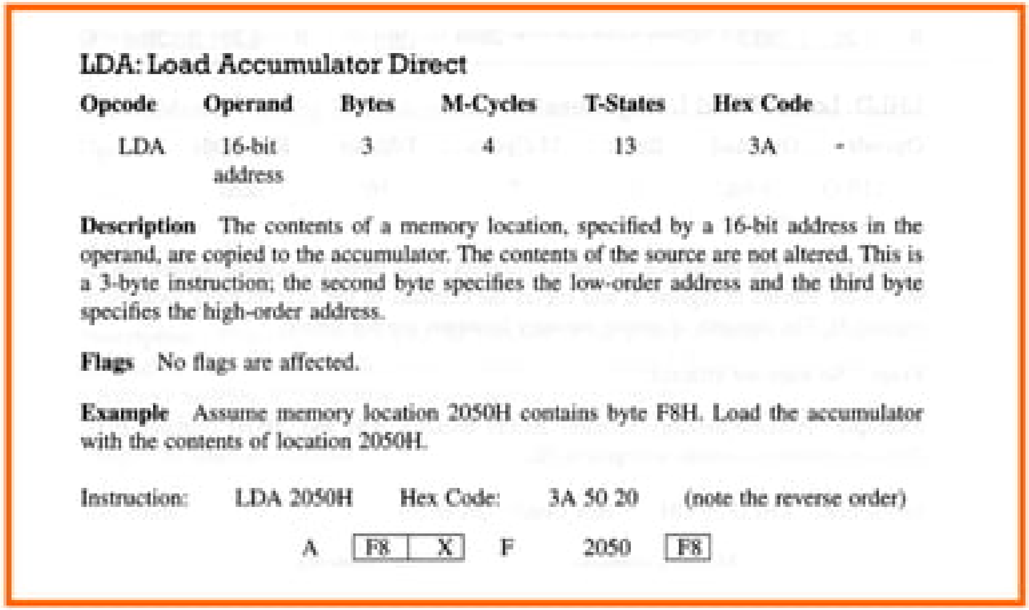


# The Memory "Register"

- Most of the instructions of the 8085 can use a memory location in place of a register.
  - The memory location will become the "memory" register M.
    - **MOV M, R**
    - **MOV M, B**
      - copy the data from register B into a memory location.
  - Which memory location?

- The memory location is identified by the contents of the HL register pair.
  - The 16-bit contents of the HL register pair are treated as a 16-bit address and used to identify the memory location.

# Direct Memory Access Operations



### LDA: Load Accumulator Direct

| Opcode | Operand | Bytes | M-Cycles | T-States | Hex Code |
|---|---|---|---|---|---|
| LDA | 16-bit address | 3 | 4 | 13 | 3A |

**Description** The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

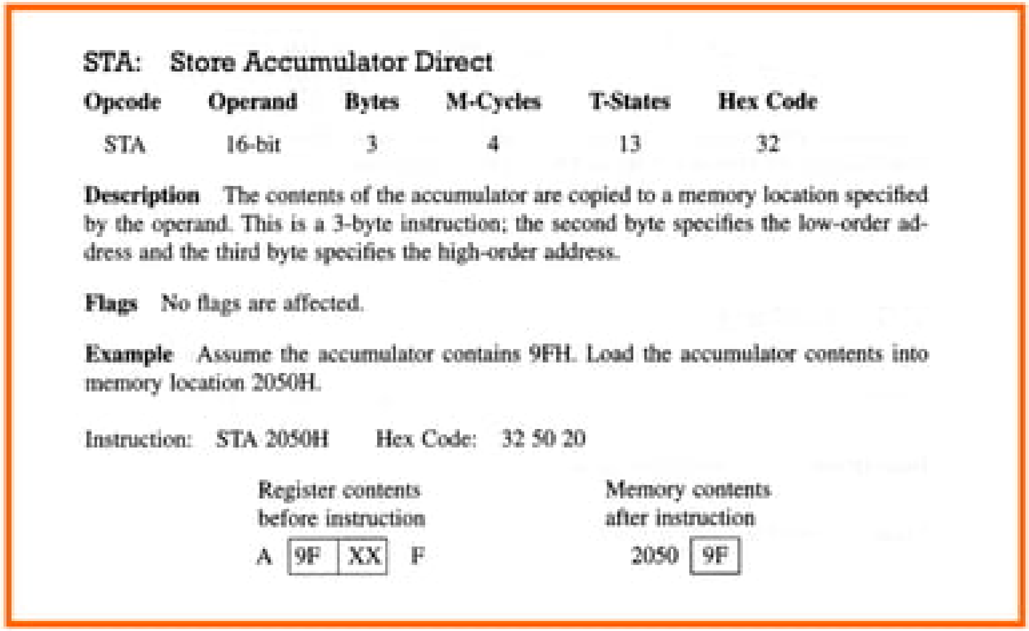
**Flags** No flags are affected.

**Example** Assume memory location 2050H contains byte F8H. Load the accumulator with the contents of location 2050H.

Instruction: LDA 2050H Hex Code: 3A 50 20 (note the reverse order)

A | F8 | X | F 2050 | F8 |

# Direct Memory Access Operations



**STA: Store Accumulator Direct**

| Opcode | Operand | Bytes | M-Cycles | T-States | Hex Code |
|---|---|---|---|---|---|
| STA | 16-bit | 3 | 4 | 13 | 32 |

**Description** The contents of the accumulator are copied to a memory location specified by the operand. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

**Flags** No flags are affected.

**Example** Assume the accumulator contains 9FH. Load the accumulator contents into memory location 2050H.

Instruction: STA 2050H Hex Code: 32 50 20

Register contents
before instruction

A | 9F | XX | F

Memory contents
after instruction

2050 | 9F |

# Indirect Addressing Mode

- Using data in memory directly (without loading first into a Microprocessor's register) is called Indirect Addressing.

- Indirect addressing uses the data in a register pair as a 16-bit address to identify the memory location being accessed.
  - The HL register pair is always used in conjunction with the memory register "M".
  - The BC and DE register pairs can be used to load data into the Accumultor using indirect addressing.

# Using the Other Register Pairs

- There is also an instruction for moving data from memory to the accumulator without disturbing the contents of the H and L register.

  - **LDAX B/D** (LoaD Accumulator eXtended)

    - Copy the 8-bit contents of the memory location identified by the Rp register pair into the Accumulator.
    - This instruction only uses the **BC** or **DE** pair.
    - It does not accept the **HL** pair.

# Indirect Memory Access Operations

- Use a register PAIR as an address pointer !

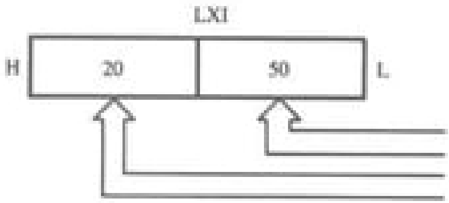
- We can define memory access operations using the memory location (16 bit address) stored in a register pair: BC, DE or HL.

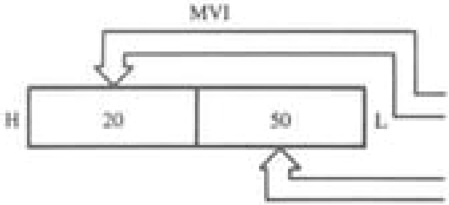
- First, we have be able to **load** the register pairs.

```
LXI B, (16-bit address)
LXI D, (16-bit address)
LXI H, (16-bit address)
```

- We can also increment / decrement register pairs.

# Loading Register Pairs





| Machine Code | Mnemonics | Comments |
|---|---|---|
| 21 | LXI H,2050H | ;Load HL registers |
| 50* | | ;50H in L register and |
| 20 | | ;20H in H register |
| 26 | MVI H,20H | ;Load 20H in register H |
| 20 | | |
| 2E | MVI L,50H | ;Load 50H in register L |
| 50 | | |

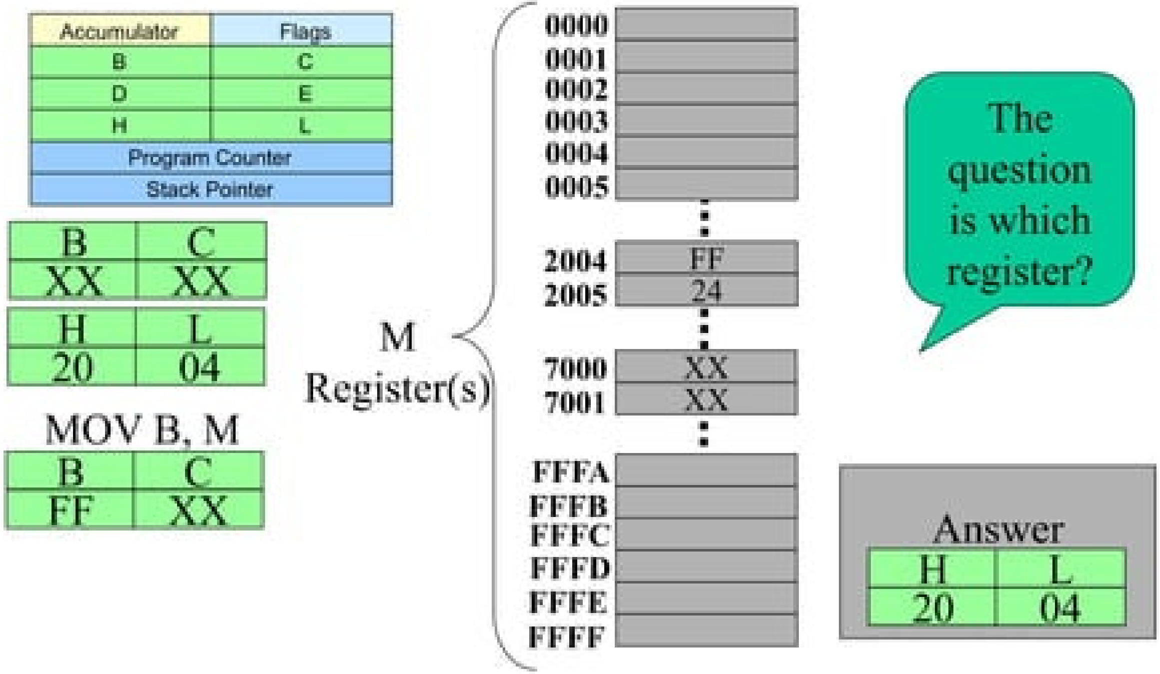
# Data Transfer from Memory to Processor

- Once the register pairs are loaded with the memory address, we can use them as **pointers**.

- MOV R,M
  Move the contents of the memory location stored in HL register pair into register (R).

- LDAX B / LDAX D
  Move the contents of the memory location stored in BC (or DE) register pair into the accumulator.

# The Memory "Register"



| Accumulator | Flags |
|---|---|
| B | C |
| D | E |
| H | L |
| Program Counter | |
| Stack Pointer | |

| B | C |
|---|---|
| XX | XX |

| H | L |
|---|---|
| 20 | 04 |

MOV B, M

| B | C |
|---|---|
| FF | XX |

M
Register(s)

| | |
|---|---|
| 0000 | |
| 0001 | |
| 0002 | |
| 0003 | |
| 0004 | |
| 0005 | |
| ⋮ | |
| 2004 | FF |
| 2005 | 24 |
| ⋮ | |
| 7000 | XX |
| 7001 | XX |
| ⋮ | |
| FFFA | |
| FFFB | |
| FFFC | |
| FFFD | |
| FFFE | |
| FFFF | |

The question is which register?

Answer

| H | L |
|---|---|
| 20 | 04 |

# The Memory “Register”

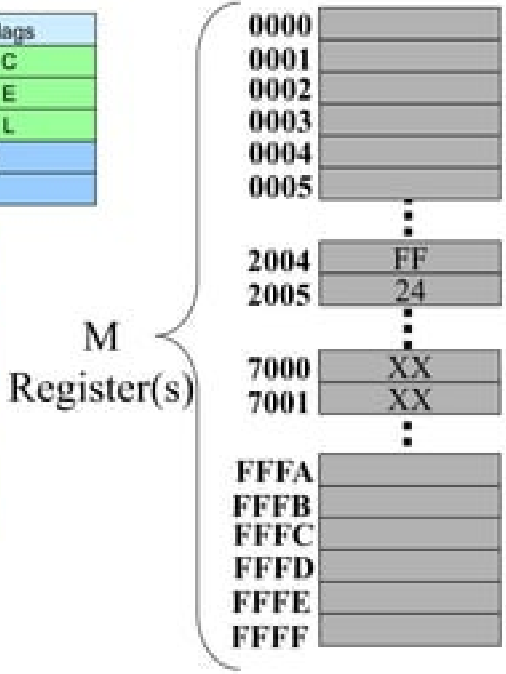
| Accumulator | Flags |
|---|---|
| B | C |
| D | E |
| H | L |
| Program Counter | |
| Stack Pointer | |

| B | C |
|---|---|
| FF | XX |

| H | L |
|---|---|
| 20 | 05 |

MOV C, M

| B | C |
|---|---|
| FF | 24 |



| | |
|---|---|
| 0000 | |
| 0001 | |
| 0002 | |
| 0003 | |
| 0004 | |
| 0005 | |
| 2004 | FF |
| 2005 | 24 |
| 7000 | XX |
| 7001 | XX |
| FFFA | |
| FFFB | |
| FFFC | |
| FFFD | |
| FFFE | |
| FFFF | |

The question is which register?

Answer

| H | L |
|---|---|
| 20 | 05 |

# The Memory “Register”

| Accumulator | Flags |
|---|---|
| B | C |
| D | E |
| H | L |
| Program Counter | |
| Stack Pointer | |

| A | F |
|---|---|
| XX | XX |

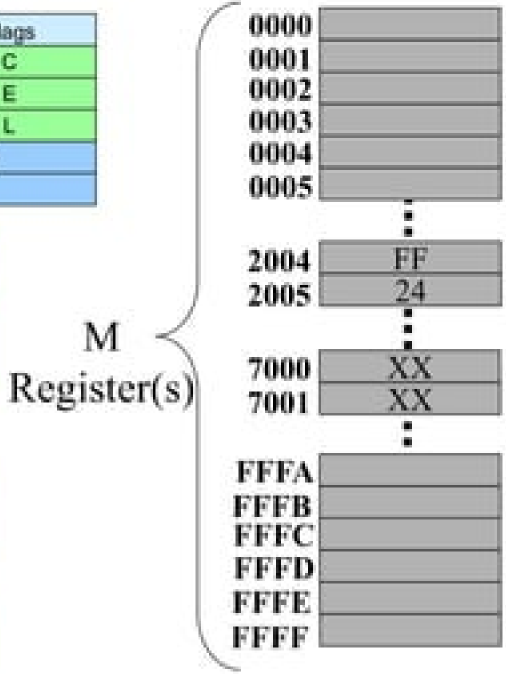
| B | C |
|---|---|
| XX | XX |

LXI B, 2005

| B | C |
|---|---|
| 20 | 05 |

LDAX B

| A | F |
|---|---|
| 24 | XX |



M
Register(s)

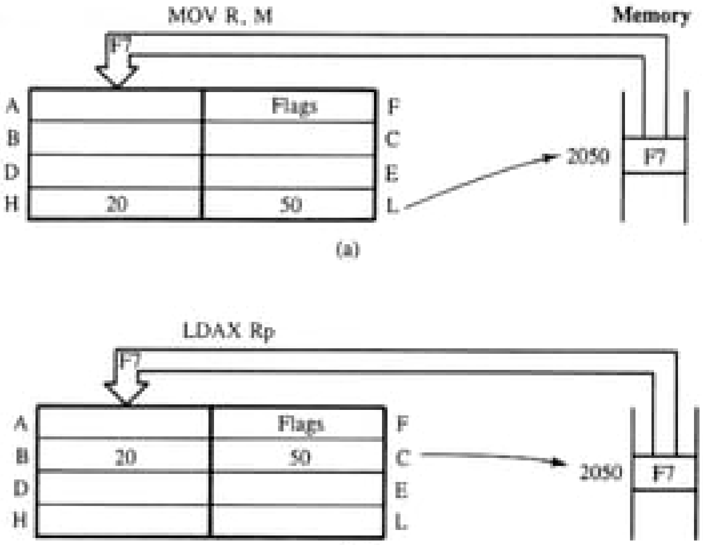
| | |
|---|---|
| 0000 | |
| 0001 | |
| 0002 | |
| 0003 | |
| 0004 | |
| 0005 | |
| ⋮ | |
| 2004 | FF |
| 2005 | 24 |
| ⋮ | |
| 7000 | XX |
| 7001 | XX |
| ⋮ | |
| FFFA | |
| FFFB | |
| FFFC | |
| FFFD | |
| FFFE | |
| FFFF | |

The question is which register?

Answer

| B | C |
|---|---|
| 20 | 05 |

# Data Transfer from Memory to Processor



(a)

(b)

| Machine Code | Mnemonics |
|---|---|
| 21 | LXI H,2050H |
| 50 | |
| 20 | |
| 7E | MOV A,M |

| | |
|---|---|
| 01 | LXI B,2050H |
| 50 | |
| 20 | |
| 0A | LDAX B |

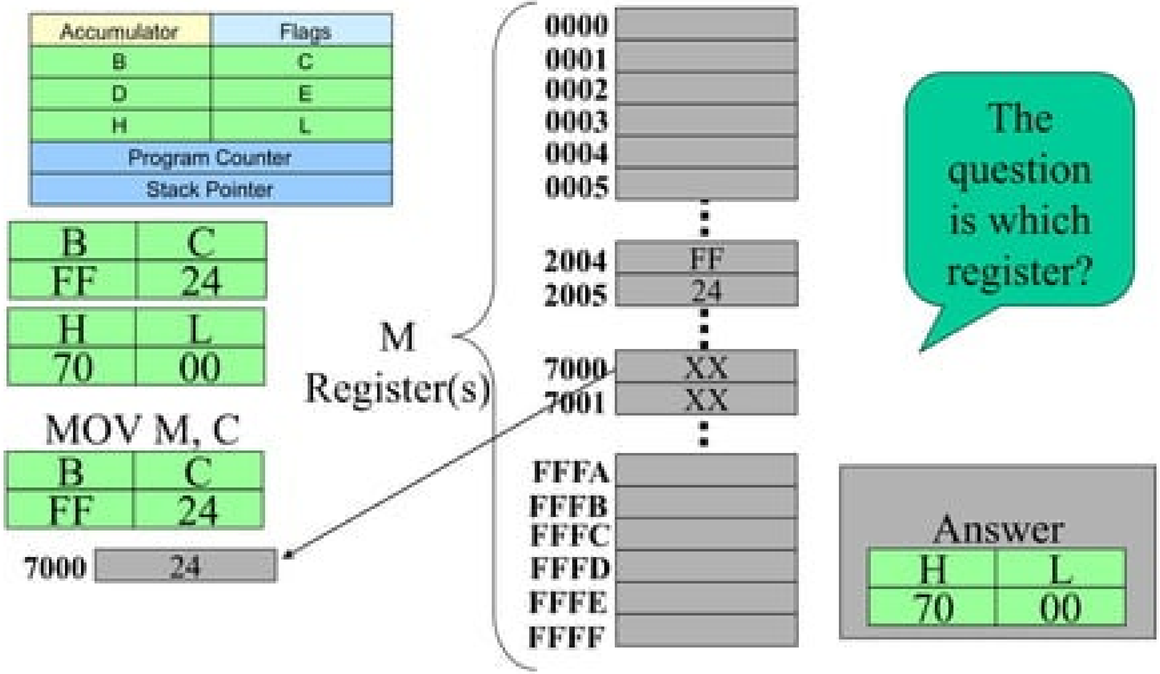
# Data Transfer from Processor to Memory

- Once the register pairs are loaded with the memory address, we can use them as **pointers**.

- MOV M,R
  Move the contents of the register (R) into memory location stored in HL register pair.

- STAX B / STAX D
  Move the contents of the accumulator into the memory location stored in BC (or DE) register pair.

# The Memory "Register"



| Accumulator | Flags |
|---|---|
| B | C |
| D | E |
| H | L |
| Program Counter | |
| Stack Pointer | |

| B | C |
|---|---|
| FF | 24 |

| H | L |
|---|---|
| 70 | 00 |

MOV M, C

| B | C |
|---|---|
| FF | 24 |

| | |
|---|---|
| 7000 | 24 |

M Register(s)

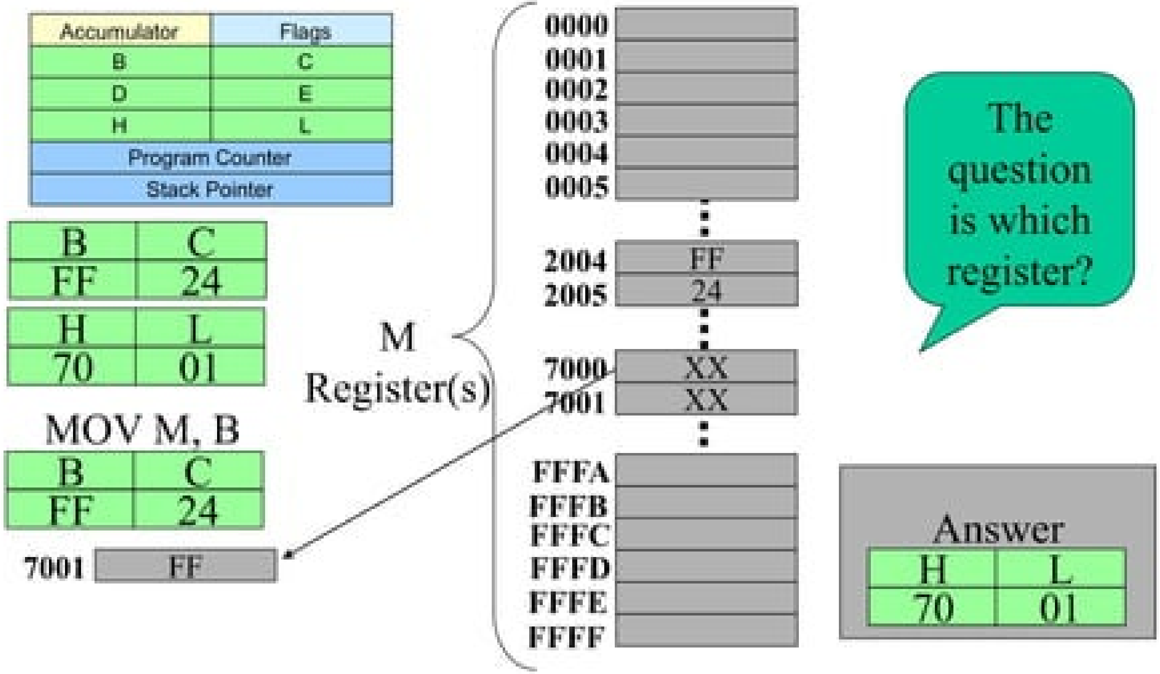
| | |
|---|---|
| 0000 | |
| 0001 | |
| 0002 | |
| 0003 | |
| 0004 | |
| 0005 | |
| 2004 | FF |
| 2005 | 24 |
| 7000 | XX |
| 7001 | XX |
| FFFA | |
| FFFB | |
| FFFC | |
| FFFD | |
| FFFE | |
| FFFF | |

The question is which register?

Answer

| H | L |
|---|---|
| 70 | 00 |

# The Memory "Register"



| Accumulator | Flags |
|---|---|
| B | C |
| D | E |
| H | L |
| Program Counter | |
| Stack Pointer | |

| B | C |
|---|---|
| FF | 24 |

| H | L |
|---|---|
| 70 | 01 |

MOV M, B

| B | C |
|---|---|
| FF | 24 |

| | |
|---|---|
| 7001 | FF |

M
Register(s)

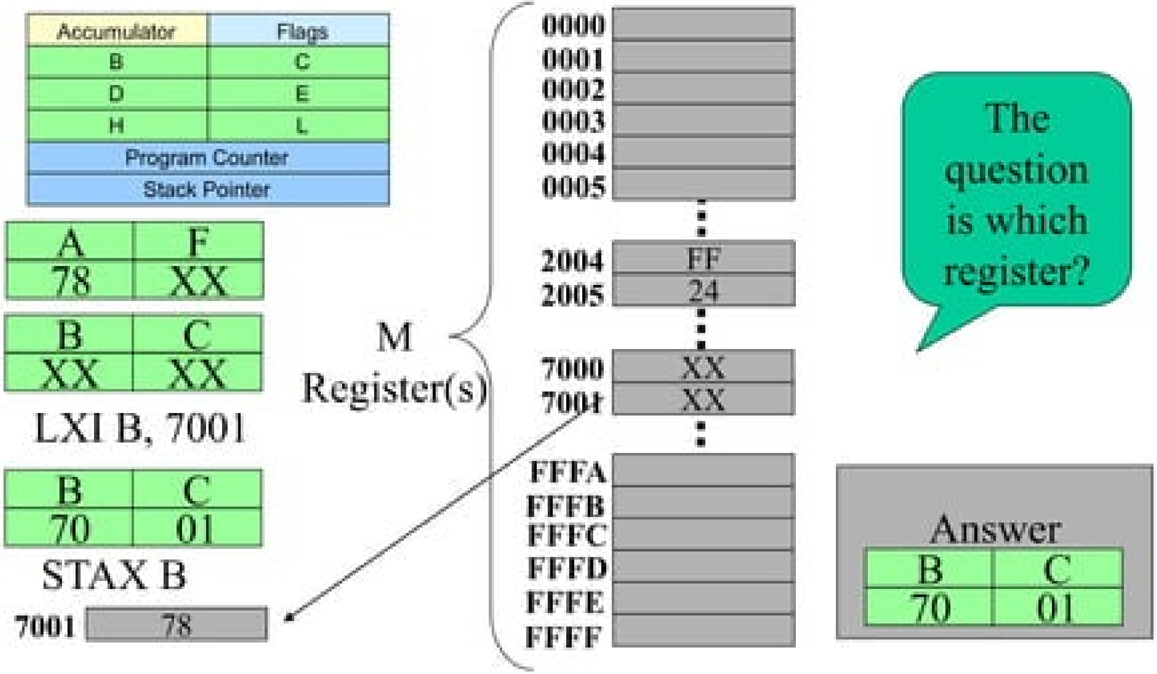
| | |
|---|---|
| 0000 | |
| 0001 | |
| 0002 | |
| 0003 | |
| 0004 | |
| 0005 | |
| 2004 | FF |
| 2005 | 24 |
| 7000 | XX |
| 7001 | XX |
| FFFA | |
| FFFB | |
| FFFC | |
| FFFD | |
| FFFE | |
| FFFF | |

The question is which register?

Answer

| H | L |
|---|---|
| 70 | 01 |

# The Memory "Register"



| Accumulator | Flags |
|---|---|
| B | C |
| D | E |
| H | L |
| Program Counter | |
| Stack Pointer | |

| A | F |
|---|---|
| 78 | XX |

| B | C |
|---|---|
| XX | XX |

LXI B, 7001

| B | C |
|---|---|
| 70 | 01 |

STAX B

| | |
|---|---|
| 7001 | 78 |

M
Register(s)

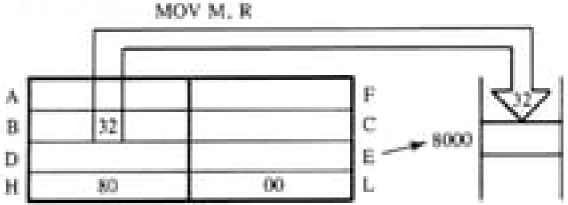
| | |
|---|---|
| 0000 | |
| 0001 | |
| 0002 | |
| 0003 | |
| 0004 | |
| 0005 | |
| 2004 | FF |
| 2005 | 24 |
| 7000 | XX |
| 7001 | XX |
| FFFA | |
| FFFB | |
| FFFC | |
| FFFD | |
| FFFE | |
| FFFF | |

The question is which register?

Answer

| B | C |
|---|---|
| 70 | 01 |

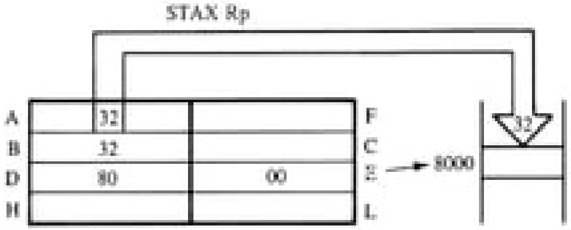
# Data Transfer from Processor to Memory



(a)

| Machine Code | Mnemonics |
|---|---|
| 21 | LXI H,8000H |
| 00 | |
| 80 | |
| 70 | MOV M,B |

This instruction copies the contents of the accumulator into memory. Therefore, it is necessary first to copy (B) into A.



(b)

| | |
|---|---|
| 11 | LXI D,8000H |
| 00 | |
| 80 | |
| 78 | MOV A,B |
| 12 | STAX D |

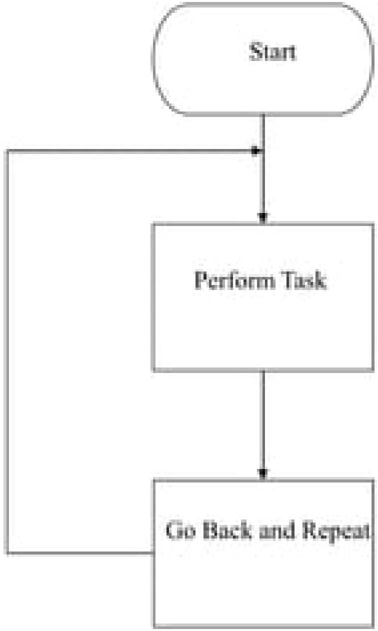
# Arithmetic Operations Related to 16 Bits or Register Pairs

- Now that we have a 16-bit address in a register pair, how do we manipulate it?
  - It is possible to manipulate a 16-bit address stored in a register pair as one entity using some special instructions.
    - **INX Rp** (Increment the 16-bit number in the register pair)
      - INX B
      - INX D
      - INX H
      - INX SP
    - **DCX Rp** (Decrement the 16-bit number in the register pair)
      - DCX B
      - DCX D
      - DCX H
      - DCX SP
  - The register pair is incremented or decremented as one entity. No need to worry about a carry from the lower 8-bits to the upper. It is taken care of automatically.

# Programing Tecniques: Looping, Counting, and Indexing

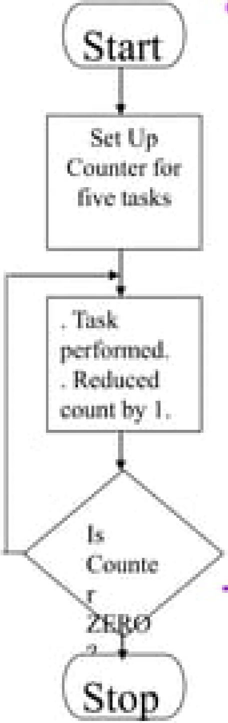
- The programming technique used to instruct the microprocessor to repeat tasks is called looping. A loop is set up by instructing the microprocessor to change the sequence of execution and perform the task again. This process is accomplished by using JUMP instructions. In addition, techniques such as counting and indexing are setting up a loop.
- Loops can be classified into two groups
  - Continuous loop: Repeats a task continuously
  - Conditional loop: Repeats a task until certain data conditions are met.

# Continuos Loop



- A program with a continuous loop does not stop repeating the task until the system is reset.

# Conditional Loop


Start
Set Up Counter for five tasks
. Task performed. . Reduced count by 1.
Is Counte r ZERO ?
Stop

- The microprocessor needs a counter to repeat the task five times, and when the counting is completed, it needs a flag.

1. Counter is set up by loading an appropriate count ain a register.
2. Counting is performed by either incrementing or decrementing the counter.
3. Loop is set up by a conditional JUMP instruction.
4. End of counting is indicating by a flag.

– It is easier to count down to zero than count up because the Z flag is set when the register becomes zero.

# Programing Tecniques: Looping, Counting, and Indexing

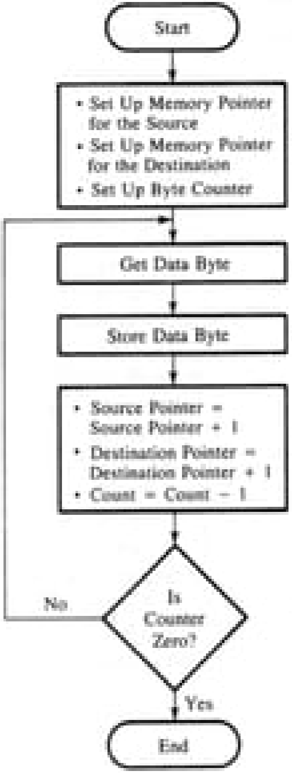
- Conditional loop, Counter, and Indexing Another type of loop includes indexing along with a counter. (Indexing means pointing or referencing objects with sequential numbers. In a library, books are arranged according to numbers, and they are referred to or stored by numbers. This is called indexing.) Similarly, data bytes are stored in memory locations, and those data bytes are referred to by their memory locations.

# Block Data Transfer Example

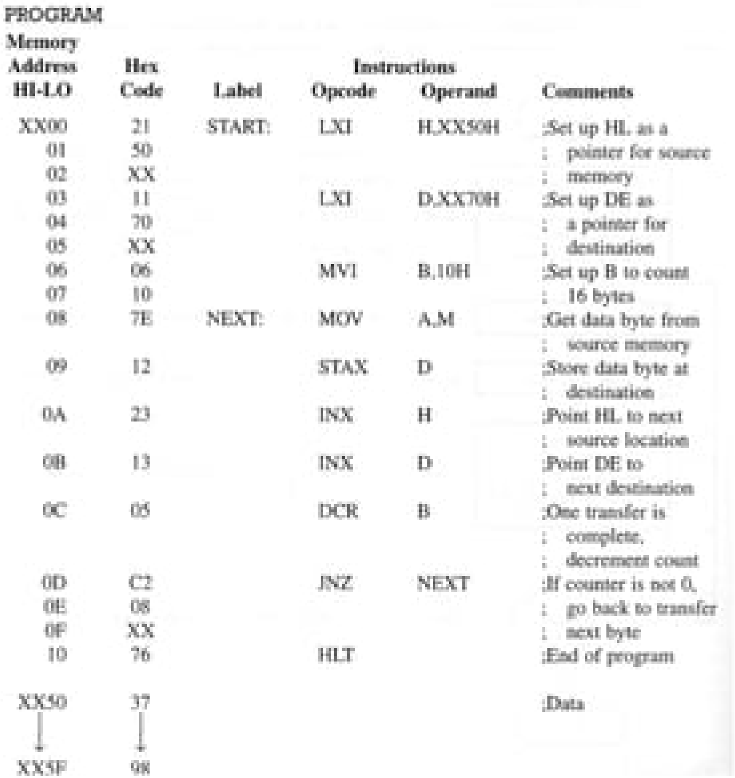
- 16 bytes of data are stored in memory locations XX50 to XX5F (16 consecutive memory addresses).

- Transfer (copy) the entire block of data to new memory locations starting at XX70.

- Make use of memory address pointers and indirect addressing !

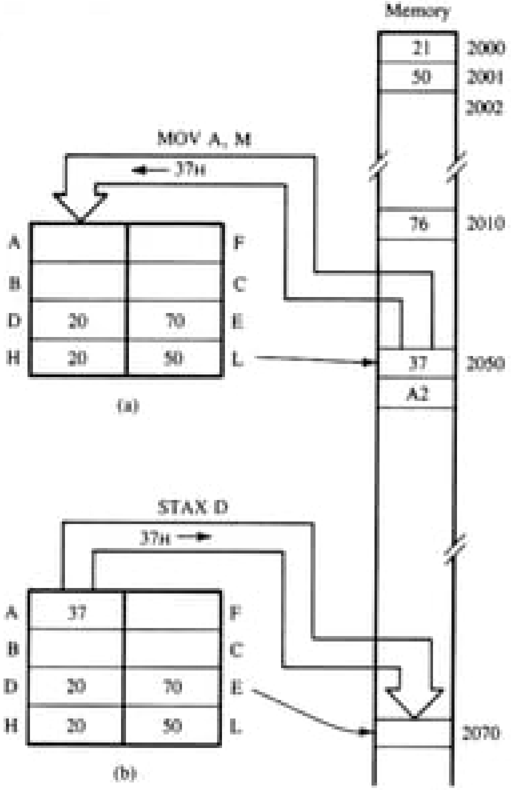

Start
• Set Up Memory Pointer for the Source
• Set Up Memory Pointer for the Destination
• Set Up Byte Counter
Get Data Byte
Store Data Byte
• Source Pointer = Source Pointer + 1
• Destination Pointer = Destination Pointer + 1
• Count = Count − 1
Is Counter Zero?
No
Yes
End

| | Mnemonics | Steps |
|---|---|---|
| Block 1 | LXI H,XX50H | 1. Set up HL as a pointer for the source memory |
| | LXI D,XX70H | Set up DE as a pointer for the destination memory |
| | MVI B,10H | Set up B as byte counter |
| Block 2 | NEXT: MOV A,M | 2. Get data byte from the source memory |
| Block 7 | STAX D | 3. Store the data byte in destination memory |
| Block 5 | INX H<br>INX D<br>DCR B | 4. Get ready to transfer next byte |
| Block 6 | JNZ NEXT | 5. Go back to get next byte if byte counter ≠0 |
| | HLT | |



PROGRAM

| Memory<br>Address<br>HI-LO | Hex<br>Code | Label | Instructions<br>Opcode | <br>Operand | Comments |
|---|---|---|---|---|---|
| XX00 | 21 | START: | LXI | H,XX50H | ;Set up HL as a |
| 01 | 50 | | | | ; pointer for source |
| 02 | XX | | | | ; memory |
| 03 | 11 | | LXI | D,XX70H | ;Set up DE as |
| 04 | 70 | | | | ; a pointer for |
| 05 | XX | | | | ; destination |
| 06 | 06 | | MVI | B,10H | ;Set up B to count |
| 07 | 10 | | | | ; 16 bytes |
| 08 | 7E | NEXT: | MOV | A,M | ;Get data byte from |
| | | | | | ; source memory |
| 09 | 12 | | STAX | D | ;Store data byte at |
| | | | | | ; destination |
| 0A | 23 | | INX | H | ;Point HL to next |
| | | | | | ; source location |
| 0B | 13 | | INX | D | ;Point DE to |
| | | | | | ; next destination |
| 0C | 05 | | DCR | B | ;One transfer is |
| | | | | | ; complete, |
| | | | | | ; decrement count |
| 0D | C2 | | JNZ | NEXT | ;If counter is not 0, |
| 0E | 08 | | | | ; go back to transfer |
| 0F | XX | | | | ; next byte |
| 10 | 76 | | HLT | | ;End of program |
| XX50 | 37 | | | | ;Data |
| ↓ | ↓ | | | | |
| XX5F | 98 | | | | |

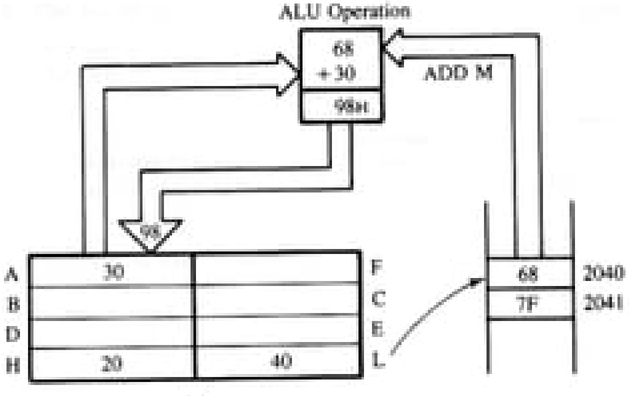
Memory
21
2000
50
2001
2002
MOV A, M
37H
76
2010
A
F
B
C
D
20
70
E
H
20
50
L
37
2050
A2
(a)
STAX D
37H
A
37
F
B
C
D
20
70
E
H
20
50
L
2070
(b)

# Arithmetic Operations Related to Memory

- These instructions perform an arithmetic operation using the contents of a memory location while they are still in memory.
  - ADD M
    - Add the contents of M to the Accumulator
  - SUB M
    - Sub the contents of M from the Accumulator
  - INR M / DCR M
    - Increment/decrement the contents of the memory location in place.

  - All of these use the contents of the HL register pair to identify the memory location being used.

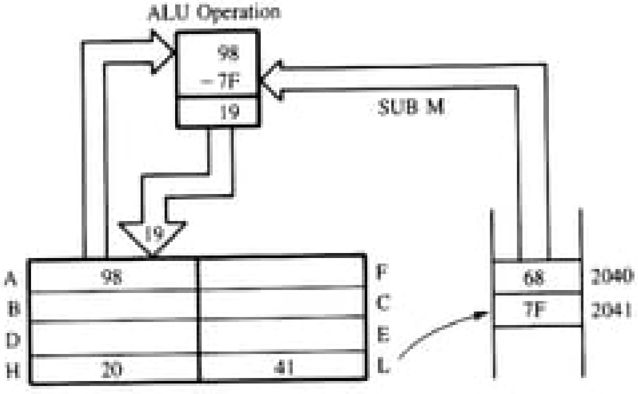
# Arithmetic Operations Using Memory



(a)

| Machine Code | Mnemonics |
|---|---|
| 21 | |
| 40 | |
| 20 | LXI H,2040H |
| 86 | ADD M |

# Arithmetic Operations Using Memory



(b)

23
96

INX H
SUB M

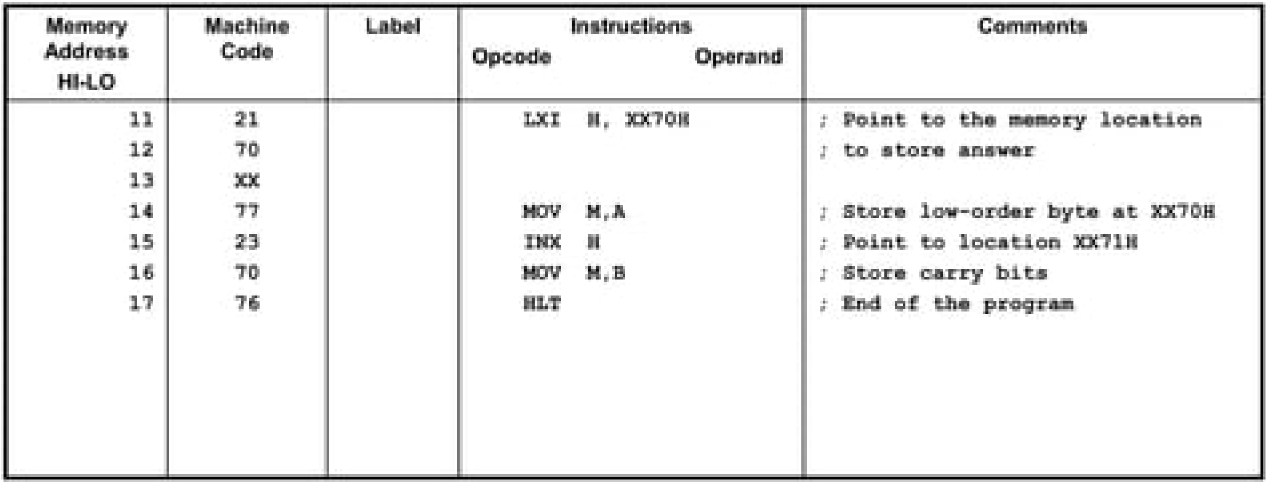
# Illustrative Program: Addition With Carry

- Six bytes of data are stored in memory locations starting at XX50H. Add all the data bytes. Use register B to save any carries generated, while adding the data bytes. Display the entire sum at two output ports, or store the sum at two consecutive memory locations XX70 H. And XX71 H.
  - Data(H): A2, FA, DF, E5, 98, 8B

| Memory Address HI-LO | Machine Code | Label | Instructions |  | Comments |
|---|---|---|---|---|---|
|  |  |  | Opcode | Operand |  |
| XX00 | AF |  | XRA | A | ; Clear (A) to save sum |
| 01 | 47 |  | MOV | B,A | ; Clear (B) to save carry |
| 02 | 0E |  | MVI | C,06H | ; Set up register C as counter |
| 03 | 06 |  |  |  |  |
| 04 | 21 |  | LXI | H, XX50H | ; Set up HL as memory pointer |
| 05 | 50 |  |  |  |  |
| 06 | XX |  |  |  |  |
| 07 | 86 | NXTBYTE: | ADD | M | ; Add byte from memory |
| 08 | D2 |  | JNC | NXTMEM | ; If no carry , do not increment |
| 09 | 0C |  |  |  | ; carry register |
| 0A | XX |  |  |  |  |
| 0B | 04 |  | INR | B | ; If carry, save carry bit |
| 0C | 23 | NXTMEM: | INX | H | ; Point to next memory location |
| 0D | 0D |  | DCR | C | ; One addition is completed |
|  |  |  |  |  | ; decrement counter |
| 0E | C2 |  | JNZ | NXTBYTE | ; If all bytes are not yet added, |
| 0F | 07 |  |  |  | ; go back to get next byte |
| 10 | XX |  |  |  |  |
| 11 | D3 |  | OUT | PORT1 | ; Display low-order byte of the |
| 12 | PORT1 |  |  |  | ; sum at PORT1 |
| 13 | 78 |  | MOV | A,B | ; Transfer carry to accumulator |
| 14 | D3 |  | OUT | PORT2 | ; Display Carry digits |
| 15 | PORT2 |  |  |  |  |
| 16 | 76 |  | HLT |  | ; End of program |

# Illustrative Program: Addition With Carry

## Storing in memory – Alternative to ouput display



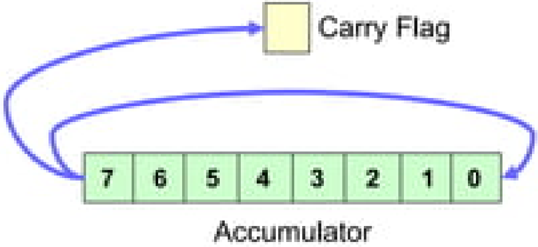
| Memory Address HI-LO | Machine Code | Label | Instructions | | Comments |
|---|---|---|---|---|---|
| | | | Opcode | Operand | |
| 11 | 21 | | LXI | H, XX70H | ; Point to the memory location |
| 12 | 70 | | | | ; to store answer |
| 13 | XX | | | | |
| 14 | 77 | | MOV | M,A | ; Store low-order byte at XX70H |
| 15 | 23 | | INX | H | ; Point to location XX71H |
| 16 | 70 | | MOV | M,B | ; Store carry bits |
| 17 | 76 | | HLT | | ; End of the program |

# Additional Logic Operations

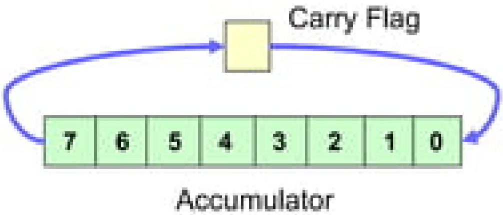
- Rotate
  - Rotate the contents of the accumulator one position to the left or right.
    - RLC Rotate the accumulator left.
      Bit 7 goes to bit 0 AND the Carry flag.
    - RAL Rotate the accumulator left through the carry.
      Bit 7 goes to the carry and carry goes to bit 0.
    - RRC Rotate the accumulator right.
      Bit 0 goes to bit 7 AND the Carry flag.
    - RAR Rotate the accumulator right through the carry.
      Bit 0 goes to the carry and carry goes to bit 7.

# RLC vs. RAL

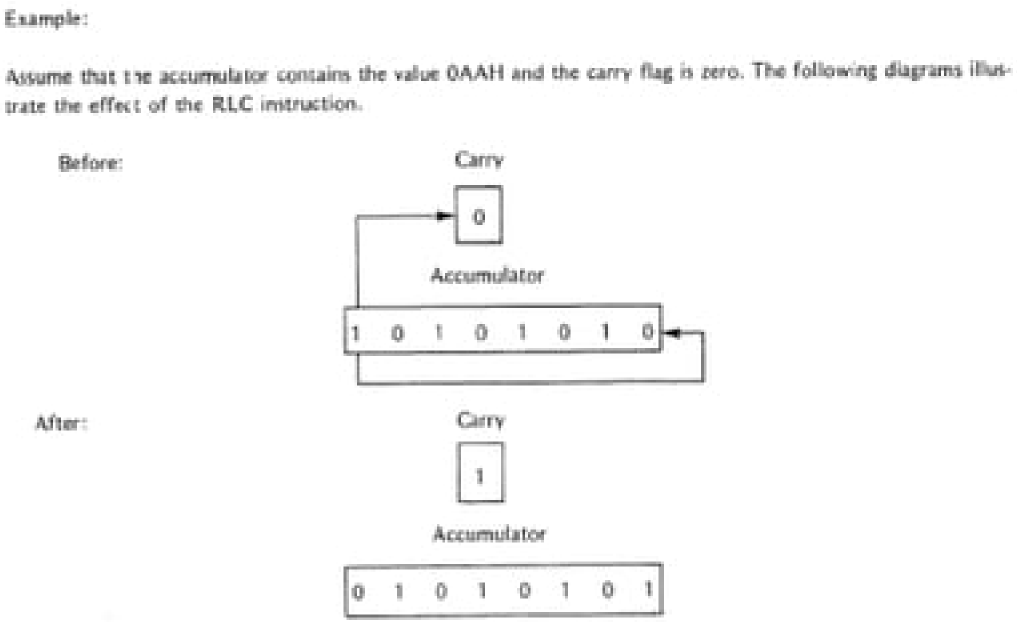
- RLC



- RAL



# RLC - Example



Example:

Assume that the accumulator contains the value 0AAH and the carry flag is zero. The following diagrams illustrate the effect of the RLC instruction.

Before:

Carry

0

Accumulator

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

After:

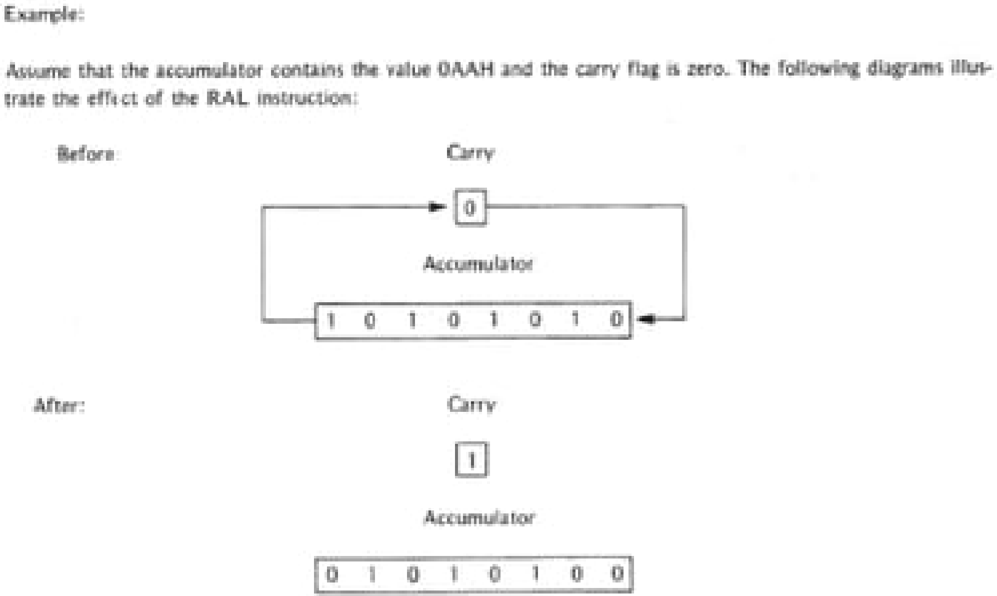
Carry

1

Accumulator

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

# RAL - Example



Example:

Assume that the accumulator contains the value 0AAH and the carry flag is zero. The following diagrams illustrate the effect of the RAL instruction:

Before:

Carry

0

Accumulator

1 0 1 0 1 0 1 0

After:

Carry

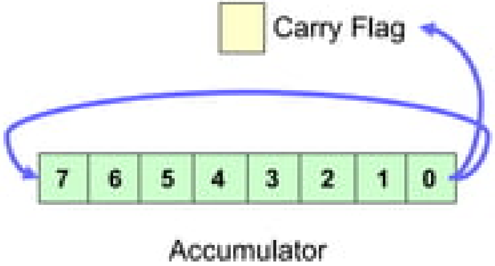
1

Accumulator

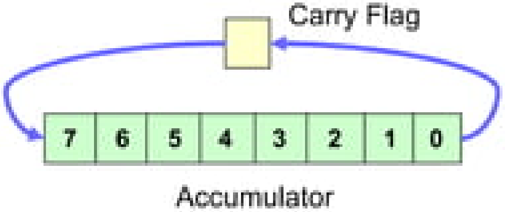
0 1 0 1 0 1 0 0

# RRC vs. RAR

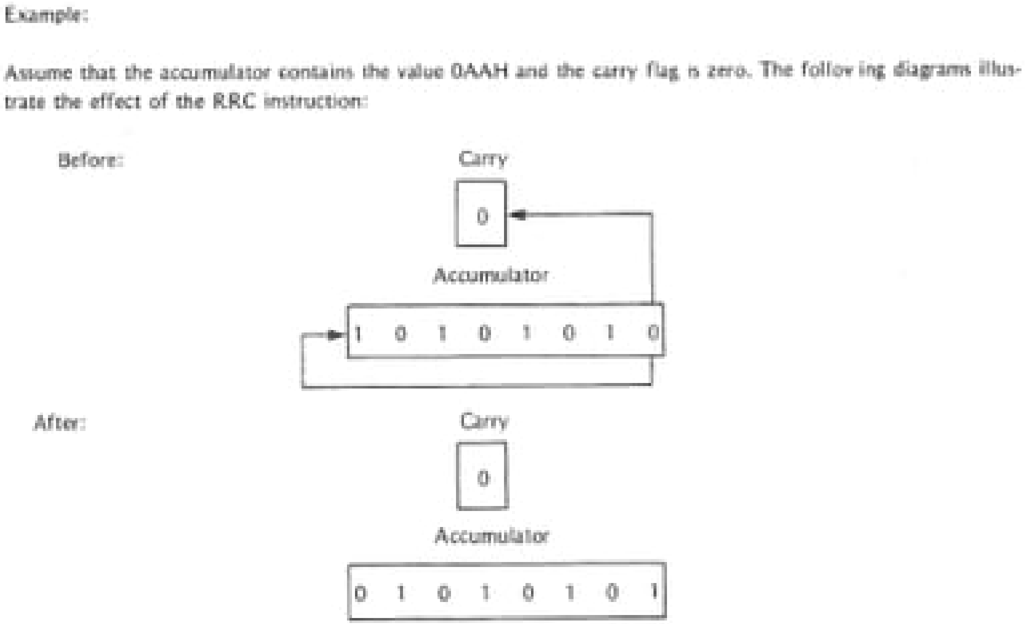
- RRC



- RAR



# RRC - Example



Example:

Assume that the accumulator contains the value 0AAH and the carry flag is zero. The follov ing diagrams illustrate the effect of the RRC instruction:

Before:

Carry

0

Accumulator

1 0 1 0 1 0 1 0

After:

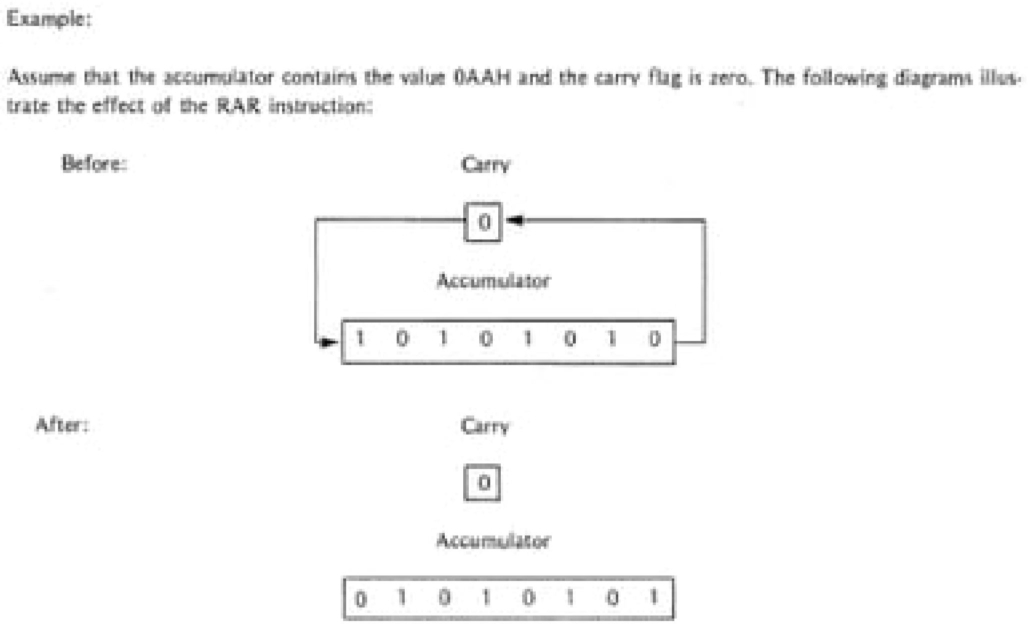
Carry

0

Accumulator

0 1 0 1 0 1 0 1

# RAR - Example



Example:

Assume that the accumulator contains the value 0AAH and the carry flag is zero. The following diagrams illustrate the effect of the RAR instruction:

Before:

Carry

0

Accumulator

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

After:

Carry

0

Accumulator

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

# Logical Operations

- Compare
  - Compare the contents of a register or memory location with the contents of the accumulator.

| | | |
|---|---|---|
| – CMP | R/M | Compare the contents of the register or memory location to the contents of the accumulator. |
| – CPI | # | Compare the 8-bit number to the contents of the accumulator. |

  - The compare instruction sets the flags (Z, Cy, and S).
  - The compare is done using an internal subtraction that does not change the contents of the accumulator.

    A – (R / M / #)

# Logical Operations - Compare

- CMP R/M: This is a 1-byte instruction
  - It compares the data byte in register or memory with the contents of the accumulator.
  - If (A)<(R/M), the CY flag is set and the Z flag is reset
  - If (A)=(R/M), the Z flag is set and the CY flag is reset
  - If (A)>(R/M), the CY and Z flags are reset
  - When memory is operand, its address is specified by (HL).
  - No contents are modified.
- CPI 8-bit: This is a 2 byte instruction
  - It compares the second byte with the contents of the accumulator.
  - If (A)<(8-bit), the CY flag is set and the Z flag is reset
  - If (A)=(8-bit), the Z flag is set and the CY flag is reset
  - If (A)>(8-bit), the CY and Z flags are reset
  - No contents are modified.