

Microprocessors

**Presented By
Pramod Ghimire
Room No: 205 (Block B)
E-mail: pramodg@nec.edu.np
<http://www.nec.edu.np/faculty/pramodg/>**



Course Objectives:

The objective of this course is to provide the knowledge of

- 1) The architecture and organization of a Microprocessor (8085/8086)
- 2) The basic operations, programming and applications of microprocessor
- 3) The interfacing I/O devices with the microprocessors
- 4) The foundation for the microprocessor based system design



Text Books

- 1) Microprocessor Architecture, Programming and Applications with 8085, **Latest edition** by Ramesh Goankar
- 2) Microprocessors and Interfacing, **3rd edition** by Douglas V Hall, and SSSP Rao
- 3) The Intel Microprocessors, Architecture, Programming and Interfacing, **8th edition** by Barry B Brey



You Need to Know

No of Units : 5

No of Books: 3

Goankar Covers: 4 Units (60 marks)

Barry/Hall Covers: 1 Unit (40 marks)

University Question Format: 70 % theory + 30% programs (2 programs from 8085, 2 programs from 8086)



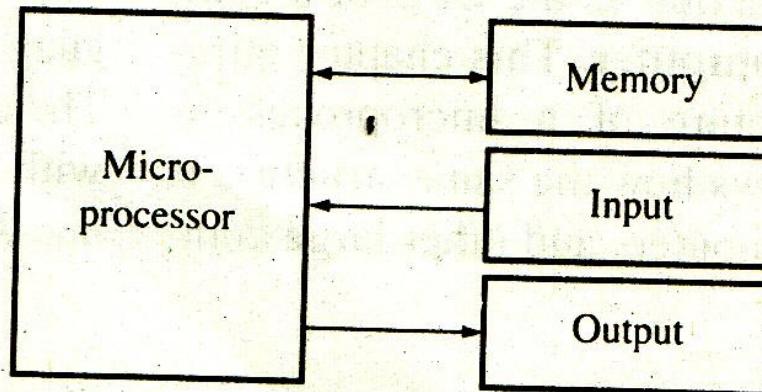
Unit 1: Introduction to Microprocessors



What is a Microprocessor ?

A microprocessor is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions, and provides result as output.

FIGURE 1.1
A Programmable Machine



Microprocessor Vs Microcontroller?

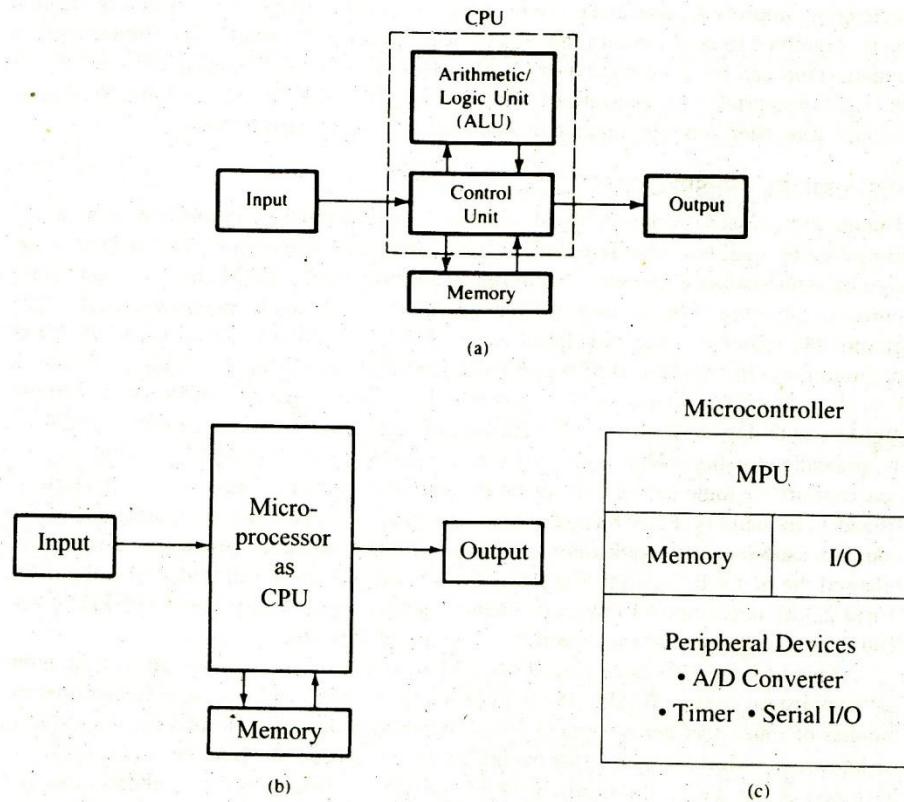


FIGURE 1.2

(a) Traditional Block Diagram of a Computer. (b) Block Diagram of a Computer with the Microprocessor as CPU; and (c) Block Diagram of a Microcontroller

Differentiate between a microprocessor and a microcontroller. (2014 Fall , 1(b) 5 marks)

Microprocessor

1. Does not have RAM, ROM, I/O pins
2. General purpose
3. For example 8085,8086

Microcontroller

1. All in one Microprocessor with RAM, ROM, I/O (System on a single Chip)
2. Specific purpose
3. For example 8031,8051



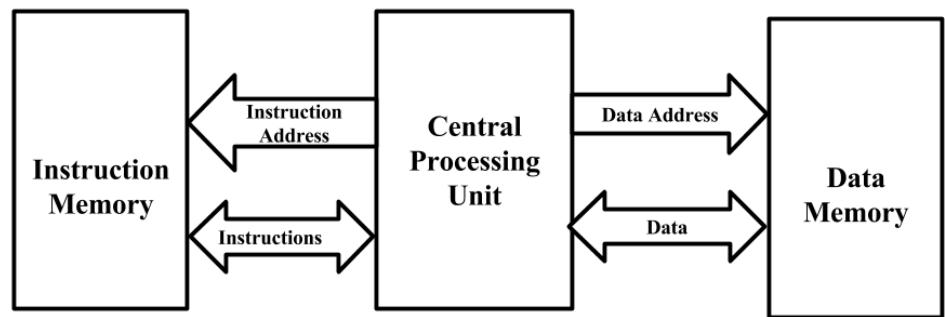
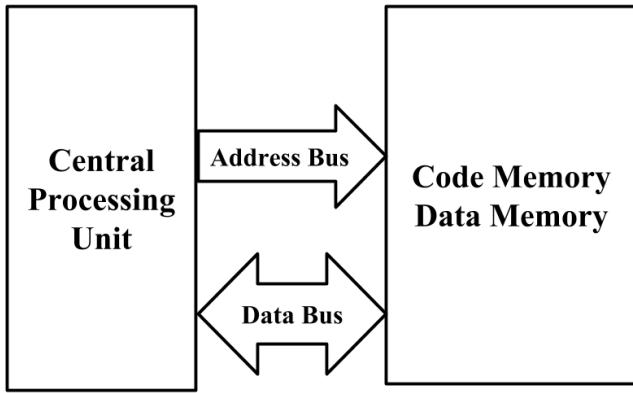
Types of Computer Architectures

There are two types of computer architectures:

- 1) Von Neumann architecture
- 2) Harvard architecture



Von Neumann Vs Harvard Architecture



1. Simplest architecture
2. It has one memory for code and data
3. One address bus for both the code and data memory
4. One data bus for the single memory
5. Slower w.r.t Harvard Architecture

1. Advanced architecture
2. Separate memory for code and data
3. Separate address bus for code and data
4. Separate data bus for code and data
5. Faster w.r.t Von Neumann Architecture

Evolution of Microprocessor

TABLE 1.1

Intel Microprocessors: Historical Perspective

Processor	Year of Introduction	Number of Transistors	Initial Clock Speed	Address Bus	Data Bus	Addressable Memory
4004	1971	2,300	108 kHz	10-bit	4-bit	640 bytes
8008	1972	3,500	200 kHz	14-bit	8-bit	16 K
8080	1974	6,000	2 MHz	16-bit	8-bit	64 K
8085	1976	6,500	5 MHz	16-bit	8-bit	64 K
8086	1978	29,000	5 MHz	20-bit	16-bit	1 M
8088	1979	29,000	5 MHz	20-bit	8-bit*	1 M
80286	1982	134,000	8 MHz	24-bit	16-bit	16 M
80386	1985	275,000	16 MHz	32-bit	32-bit	4 G
80486	1989	1.2 M	25 MHz	32-bit	32-bit	4 G
Pentium	1993	3.1 M	60 MHz	32-bit	32/64-bit	4 G
Pentium Pro	1995	5.5 M	150 MHz	36-bit	32/64-bit	64 G
Pentium II	1997	8.8 M	233 MHz	36-bit	64-bit	64 G
Pentium III	1999	9.5 M	650 MHz	36-bit	64-bit	64 G
Pentium 4	2000	42 M	1.4 GHz	36-bit	64-bit	64 G

*External 8-bit and internal 16-bit data bus

Moore's Law

In 1965, Intel cofounder Gordon Moore predicted that the numbers of transistors on a chip would double about every two years. This is known as “Moore’s Law”



Sample Questions

1. Describe the Von Neumann's architecture of a computer system. (5 marks) **Fall 2014**
2. What are the essential differences between: (8 marks) **Spring 2014**
 - i. Von Neumann and Harvard Architecture
 - ii. Microprocessor and Microcontroller
3. Enlist the greatest breakthrough in microprocessor so that modern processors are available for personal computer. (5 marks) **Spring 2015**
4. With reference of bus advancement differentiate between Harvard Architecture and Von Neumann architecture (5 marks) **Spring 2015**
5. Define Microprocessor. Compare Intel 8085, 8086, and 80386 Microprocessors on the basis of their features and internal architectures.(7 marks)**Fall 2016**



Microcomputer System

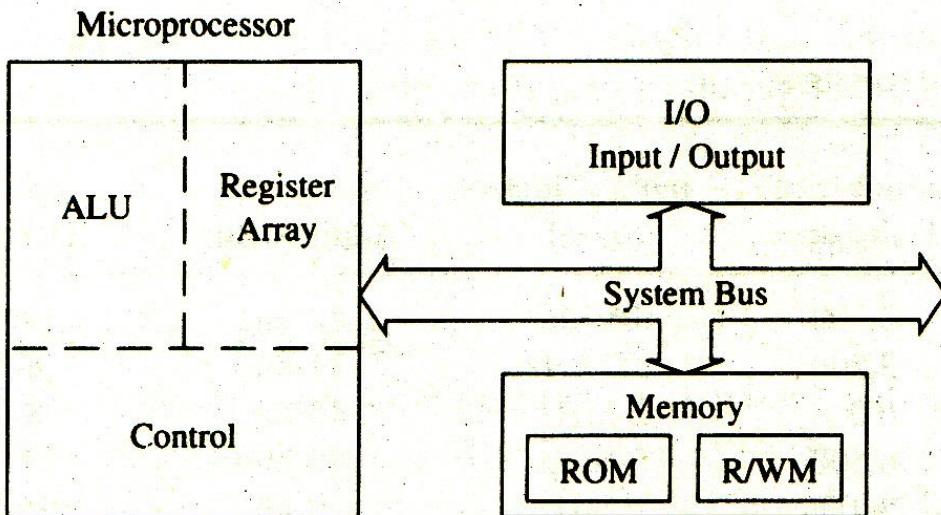


FIGURE 1.3
Microprocessor-Based System with Bus Architecture

A microcomputer consists of
Three components:
1) Microprocessor
2) I/O
3) Memory

These components are organized
around a common communication
path called a **bus**

Bus is a group of wires
to carry bits

Three Bus Architecture

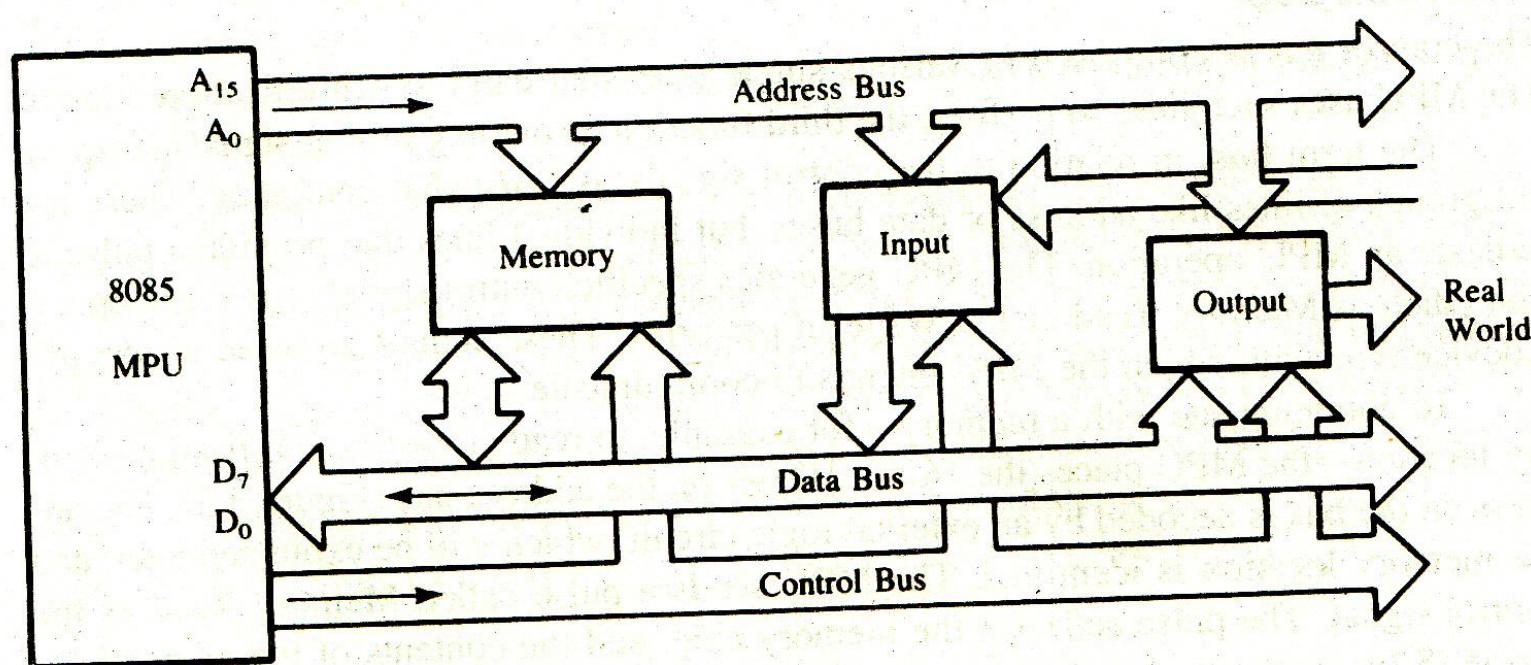


FIGURE 3.1
The 8085 Bus Structure

Different Type of System Buses

Address Bus:

The address bus is unidirectional: bits flow in one direction-from MPU to peripheral devices. The MPU uses the address bus to identify a peripheral device or a memory location. Each peripheral or memory location is identified by a binary number, called an address, and address bus is used to carry this address. Address bus is 16-bit long in 8085 microprocessor.

Data Bus:

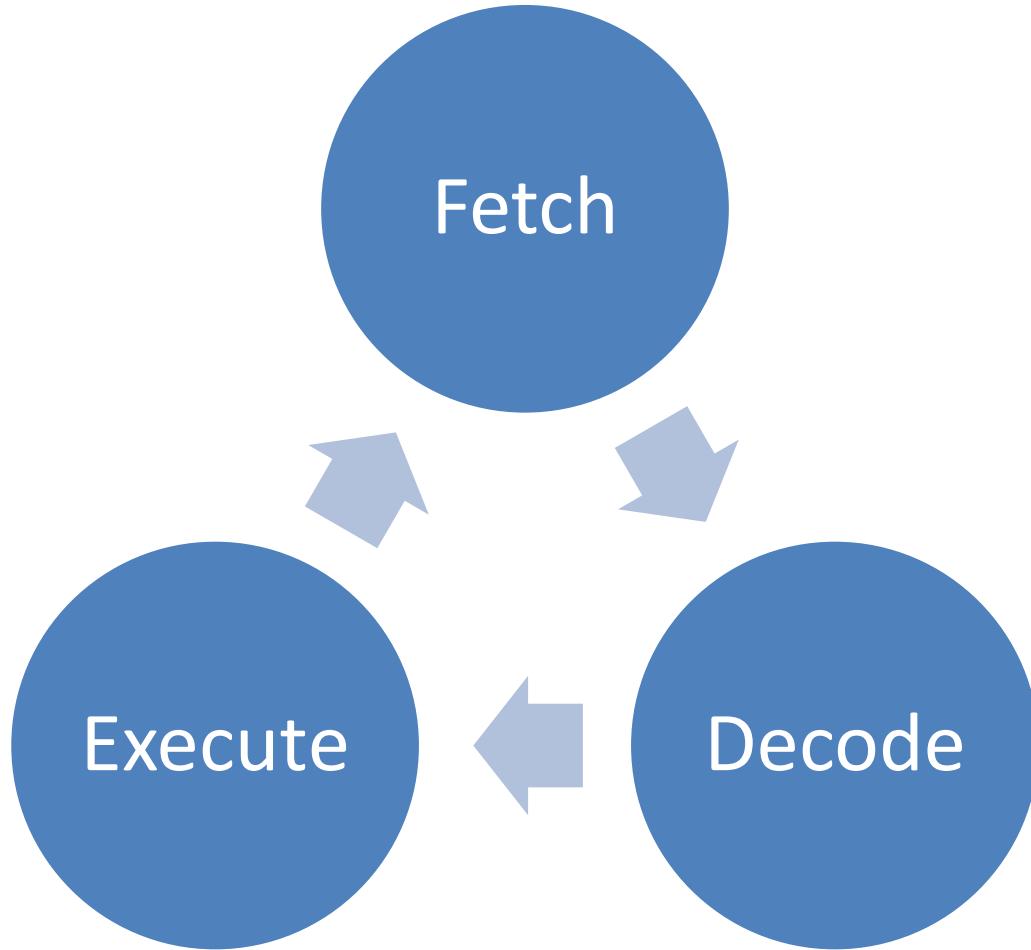
The data bus lines are bidirectional-data flow from in both directions between memory and peripheral devices. The MPU uses data bus to transfer binary information. In 8085 processor, data bus is 8-bit long.

Control Bus:

The control bus is comprised of various single lines that carry synchronization signals. The MPU uses such lines to provide timing signals



How Does Microprocessor Work ?



Explanation

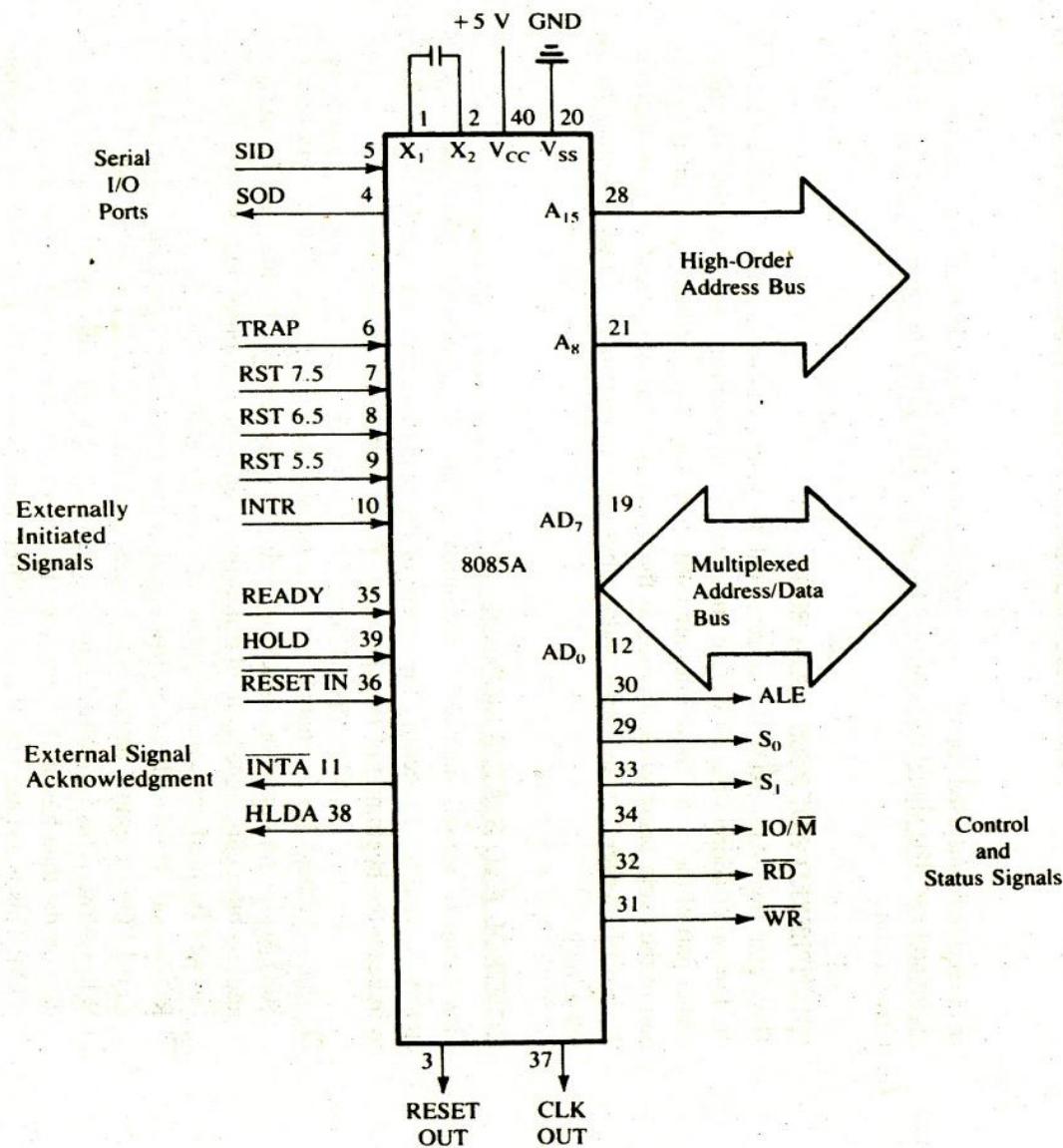
The user enters instructions in binary format into the memory. To execute a program, microprocessor **reads** each instruction from memory, **interprets** it, then **execute** it. That is the microprocessor **fetches** each instruction, **decodes** it, and **executes** it. This sequence is continued until all instructions are performed. This cycle is called Fetch, Decode, and Execute cycle.



Pin Diagram of 8085

X ₁	1	V _{CC}
X ₂	2	HOLD
RESET OUT	3	HLDA
SOD	4	CLK (OUT)
SID	5	RESETIN
TRAP	6	READY
RST 7.5	7	IO/M
RST 6.5	8	S ₁
RST 5.5	9	RD
INTR	10	WR
INTA	11	ALE
AD ₀	12	S ₀
AD ₁	13	A ₁₅
AD ₂	14	A ₁₄
AD ₃	15	A ₁₃
AD ₄	16	A ₁₂
AD ₅	17	A ₁₁
AD ₆	18	A ₁₀
AD ₇	19	A ₉
V _{SS}	20	A ₈
	21	
	22	
	23	
	24	
	25	
	26	
	27	
	28	
	29	
	30	
	31	
	32	
	33	
	34	
	35	
	36	
	37	
	38	
	39	
	40	

8085 Pinout



Programming Model of 8085

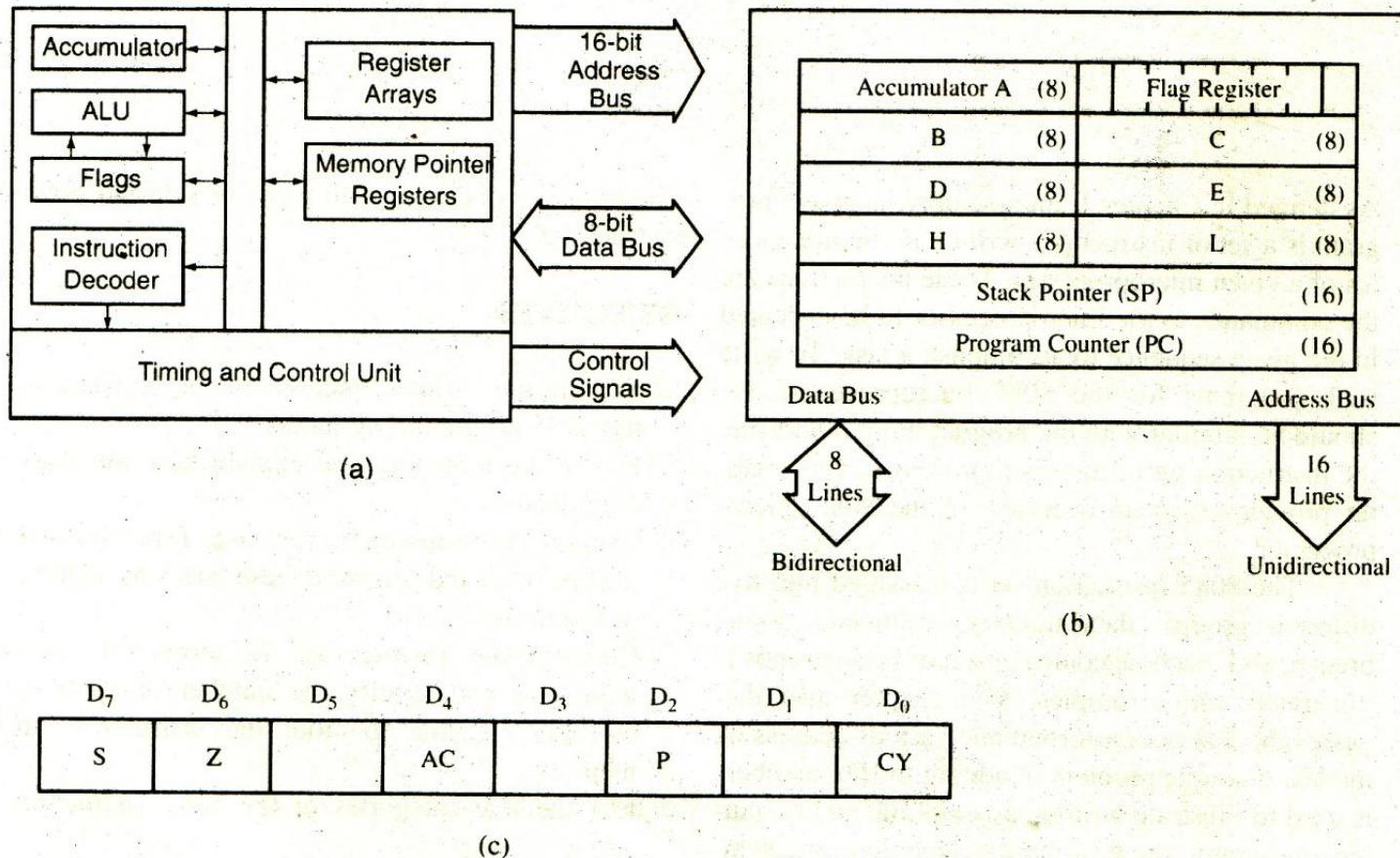


FIGURE 2.1

8085 Hardware Model (a), Programming Model (b), and Flag Register (c)

Internal Registers of 8085

- Six General Purpose Register (8-bit): B,C,D,E,H,L**
- Two 16-bit registers: SP, and PC**
- 8-bit flag register**



Flags of 8085

The five flags present in the 8085 functional block diagram indicate the arithmetic or logical result conditions in the accumulator or the other register or memory location.

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

Carry Flag (CY):

It is set to 1 if there is a carry during the arithmetic operation or if there is a borrow during arithmetic subtraction

Parity Flag (P):

Parity flag is set to 1 if the number of 1's in the arithmetic or logical result is even

Auxiliary Carry (AC):

If there is a carry or borrow from bit D3 to D4 during an arithmetic operation, AC flag is set. AC flag is used during BCD addition.

Zero Flag (Z):

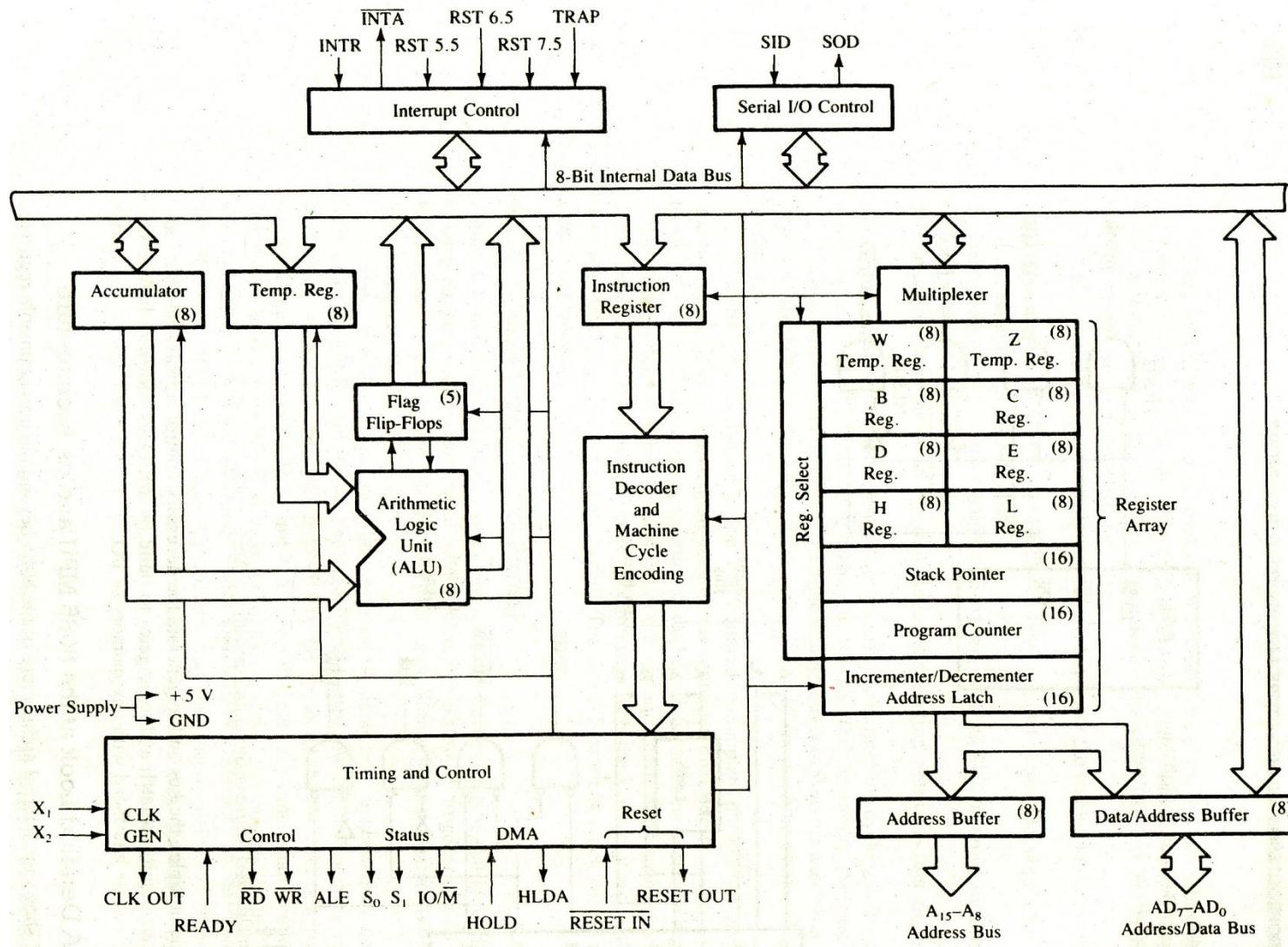
If the arithmetic or logical result is zero, Z flag is set.

Sign Flag (S):

If the MSB of the arithmetic or logical operation results in 1, S flag is set to 1. It is set if a number is a negative number



Functional Block Diagram of 8085



Description

Figure 4-7 (in text book) shows the internal architecture of the 8085 microprocessor beyond the programmable registers. It includes:

- 1) ALU
- 2) Timing and Control Unit
- 3) Instruction Register and Decoder
- 4) Register Array
- 5) Interrupt Control and
- 6) Serial I/O Control



ALU:

It includes accumulator (A-reg), temporary register, ALU, and five flags

Timing and Control Unit:

Synchronizes all the processor operations with the clock and generates the control signal necessary for communication between the microprocessor and peripherals

Instruction Register and Decoder:

When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to follow

Register Array:

Two additional register, called temporary register W and Z, are included in the register array. These registers are used to hold 8-bit data during the execution of some instructions. However, because they are used internally they are not available to the programmer.

Pin Configuration of 8085

8085 has forty pins, requires a +5V. It can operate with a 3-MHz single-phase clock.

A15-A8(Address Bus):

A15-A8 are used for the most significant bits, called the high-order address of a 16-bit address.

AD7-AD0 (Multiplexed address/data bus):

AD7-AD0 are bidirectional bus. They are used as the low-order address bus as well as the data bus. The low-order address bus can be separated from these signals by using a latch.

ALE(Address Latch Enable):

It is used to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines, A7-A0. ALE high indicates that the bits on AD7-AD0 are address bus

IO/ \overline{M} :

It is used to differentiate between I/O and memory operation. If it is at logic 1, it is an I/O operation else it is a memory operation

\overline{RD} and \overline{WR} :

Read and write signals



S1 and S2 (Status Signals):

They are two status signals

READY :

Peripheral ready if READY = 1 else microprocessor waits for an integral number of clock cycles until it goes high.

HOLD:

If HOLD = 1, it indicates that a peripheral such as a DMA controller is requesting the use of the address and data bus.

HLDA:

Hold Acknowledgement

INTR: Interrupt Request

RST 7.5, RST 6.5, and RST 5.5:

Restart Interrupts

TRAP: This is a non maskable interrupt



RESET IN:

When it goes low, the program counter is set to zero, the buses are tri-stated, and the microprocessor is reset.

RESET OUT:

This signal indicates that the microprocessor is being reset. The signal can be used to reset other devices.

SID and SOD: Serial Input Data and Serial Output Data

VCC and VSS: Power Supply and Ground

X1 and X2: A crystal is connected at these two pins. The frequency is internally divided by two, therefore, to operate a system at 3 MHz, the crystal should have a frequency of 6 MHz.

CLK (OUT): Clock Output. This signal can be used as the system clock for other devices.

Demultiplexing the AD7-AD0

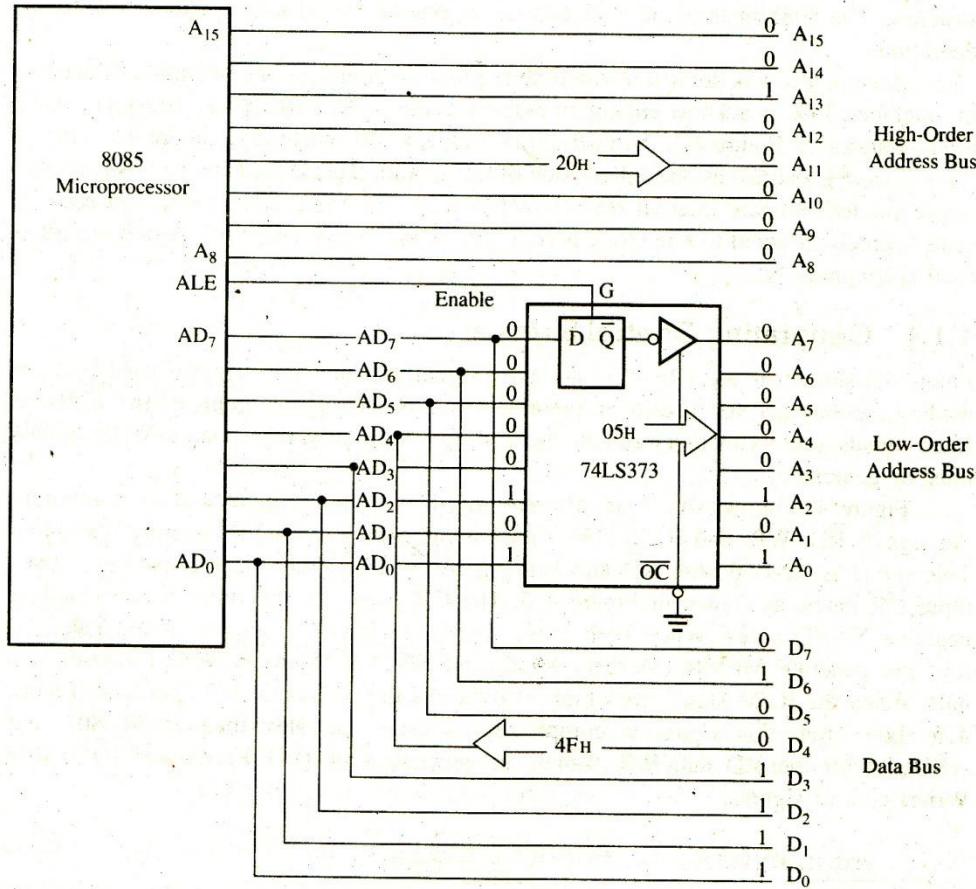


FIGURE 4.4

Schematic of Latching Low-Order Address Bus

Generation of Control Signals

FIGURE 4.5

Schematic to Generate Read/Write Control Signals for Memory and I/O

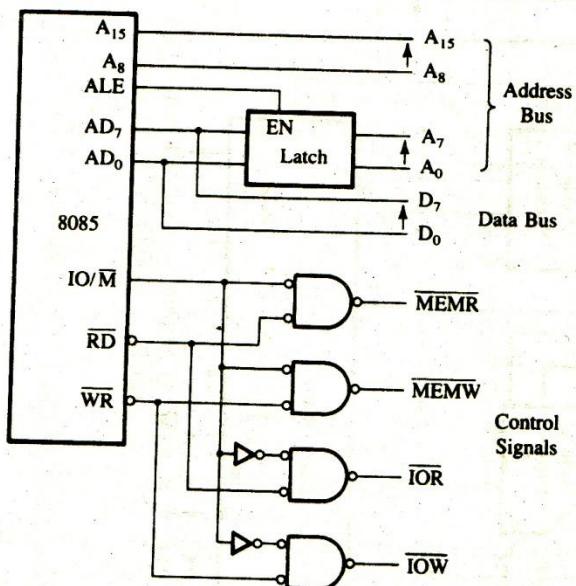
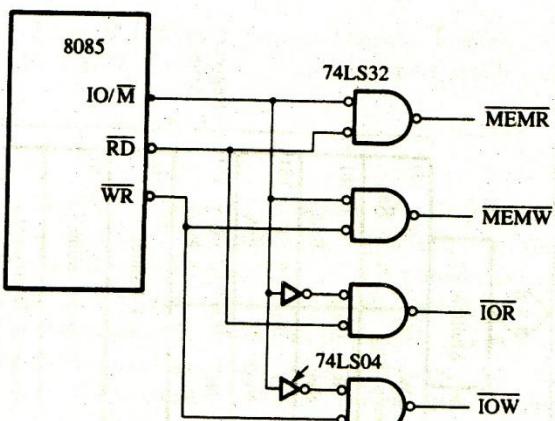


FIGURE 4.6

8085 Demultiplexed Address and Data Bus with Control Signals

The three control signals can be used to generate four different memory or I/O control signals as shown in the diagram

Overview of the 8085 Instruction Set

The 8085 microprocessor instruction set has **74 operation codes** that result in **246 instructions**

R = 8085 eight-bit register (A,B,C,D,E,H,L)

M = Memory register

Rs = Register source

Rd = Register destination (A,B,C,D,E,H,L)

Rp = Register pair (BC,DE,HL,SP)

() = Content of



Instruction and Classification

An instruction is a command to the microprocessor to perform a given task on specified data. Each instruction has two parts: one is task to be performed, called the operation code (op-code), and second is the data to be operated on, called the operand. The operand (data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or an 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

The 8085 instructions can be classified into the following five functional categories:

- 1) Data Transfer (Copy) Instruction
- 2) Arithmetic Instruction
- 3) Logical Instruction
- 4) Branching Instruction
- 5) Machine-Control Instruction



study these instructions. You are strongly advised not to attempt to read all these instructions at one time. However, you should be able to grasp an overview of the set by examining the frequently used instructions listed below.*

The following notations are used in the description of the instructions.

R = 8085 8-bit register	(A, B, C, D, E, H, L)
M = Memory register (location)	
Rs = Register source	
Rd = Register destination	(A, B, C, D, E, H, L)
Rp = Register pair	(BC, DE, HL, SP)
() = Contents of	

1. Data Transfer (Copy) Instructions. These instructions perform the following six operations.

- Load an 8-bit number in a register
- Copy from register to register
- Copy between I/O and accumulator
- Load 16-bit number in a register pair
- Copy between register and memory
- Copy between registers and stack memory

Mnemonics	Examples	Operation
1.1 MVI R,** 8-bit	MVI B, 4FH	Load 8-bit data (byte) in a register
1.2 MOV Rd, Rs**	MOV B, A MOV C, B	Copy data from source register Rs into destination register Rd
1.3 LXI Rp,** 16-bit	LXI B, 2050H	Load 16-bit number in a register pair
1.4 OUT 8-bit (port address)	OUT 01H	Send (write) data byte from the accumulator to an output device
1.5 IN 8-bit (port address)	IN 07H	Accept (read) data byte from an input device and place it in the accumulator
1.6 LDA 16-bit	LDA 2050H	Copy the data byte into A from the memory specified by 16-bit address
1.7 STA 16-bit	STA 2070H	Copy the data byte from A into the memory specified by 16-bit address
1.8 LDAX Rp	LDAX B	Copy the data byte into A from the memory specified by the address in the register pair
1.9 STAX Rp	STAX D	Copy the data byte from A into the memory specified by the address in the register pair



1.10	MOV R, M	MOV B, M	Copy the data byte into register from the memory specified by the address in HL register
1.11	MOV M, R	MOV M, C	Copy the data byte from the register into memory specified by the address in HL register

2. Arithmetic Instructions. The frequently used arithmetic operations are:

- Add
- Subtract
- Increment (Add 1)
- Decrement (Subtract 1)

Mnemonics	Examples	Operation
2.1 ADD R	ADD B	Add the contents of a register to the register to the contents of A
2.2 ADI 8-bit	ADI 37H	Add 8-bit data to the contents of A
2.3 ADD M	ADD M	Add the contents of memory to A; the address of memory is in HL register
2.4 SUB R	SUB C	Subtract the contents of a register from the contents of A
2.5 SUI 8-bit	SUI 7FH	Subtract 8-bit data from the contents of A
2.6 SUB M	SUB M	Subtract the contents of memory from A; the address of memory is in HL register
2.7 INR R	INR D	Increment the contents of a register
2.8 INR M	INR M	Increment the contents of memory, the address of which is in HL
2.9 DCR R	DCR E	Decrement the contents of a register
2.10 DCR M	DCR M	Decrement the contents of a memory, the address of which is in HL
2.11 INX Rp	INX H	Increment the contents of a register pair
2.12 DCX Rp	DCX B	Decrement the contents of a register pair

3. Logic and Bit Manipulation Instructions. These instructions include the following operations:

- AND
- OR
- X-OR (Exclusive OR)
- Compare
- Rotate Bits

Mnemonics	Examples	Operation
3.1 ANA R	ANA B	Logically AND the contents of a register with the contents of A



3.2	ANI 8-bit	ANI 2FH	Logically AND 8-bit data with the contents of A
3.3	ANA M	ANA M	Logically AND the contents of memory with the contents of A; the address of memory is in HL register
3.4	ORA R	ORA E	Logically OR the contents of a register with the contents of A
3.5	ORI 8-bit	ORI 3FH	Logically OR 8-bit data with the contents of A
3.6	ORA M	ORA M	Logically OR the contents of memory with the contents of A; the address of memory is in HL register
3.7	XRA R	XRA B	Exclusive-OR the contents of a register with the contents of A
3.8	XRI 8-bit	XRI 6AH	Exclusive-OR 8-bit data with the contents of A
3.9	XRA M	XRA M	Exclusive-OR the contents of memory with the contents of A; the address of memory is in HL register
3.10	CMP R	CMP B	Compare the contents of register with the contents of A for less than, equal to, or greater than
3.11	CPI 8-bit	CPI 4FH	Compare 8-bit data with the contents of A for less than, equal to, or greater than

4. Branch Instructions. The following instructions change the program sequence.

4.1	JMP 16-bit address	JMP 2050H	Change the program sequence to the specified 16-bit address
4.2	JZ 16-bit address	JZ 2080H	Change the program sequence to the specified 16-bit address if the Zero flag is set
4.3	JNZ 16-bit address	JNZ 2070H	Change the program sequence to the specified 16-bit address if the Zero flag is reset
4.4	JC 16-bit address	JC 2025H	Change the program sequence to the specified 16-bit address if the Carry flag is set
4.5	JNC 16-bit address	JNC 2030H	Change the program sequence to the specified 16-bit address if the Carry flag is reset
4.6	CALL 16-bit address	CALL 2075H	Change the program sequence to the location of a subroutine
4.7	RET	RET	Return to the calling program after completing the subroutine sequence

5. Machine Control Instructions. These instructions affect the operation of the processor.

5.1	HLT	HLT	Stop processing and wait
5.2	NOP	NOP	Do not perform any operation

This set of instructions is a representative sample; it does not include various instructions related to 16-bit data operations, additional jump instructions, and conditional Call and Return instructions.

2.6

WRITING AND HAND ASSEMBLING A PROGRAM

In previous sections, we discussed the 8085 instructions, recognized the number of bytes per instruction, looked at the relationship between the number of bytes of an instruction and memory registers needed for storage, and examined the processor's computing capability in the overview of the instruction set. Now let us pull together all these concepts in a simple illustrative program.

2.6.1 Illustrative Program: Subtracting Two Hexadecimal Numbers and Storing the Result in Memory

PROBLEM STATEMENT

Write instructions to subtract two bytes already stored in memory registers (also referred to as memory locations or memory addresses) 2051H and 2052H. Location 2051H holds the byte 49H and location 2051H holds the byte 9FH. Subtract the first byte, 49H, from the second byte, 9FH, and store the answer in memory location 2053H. Write instructions beginning at memory location 2030H.

PROBLEM ANALYSIS

This is a problem similar to the problem in Section 2.4. However, we need to note some specific points in this problem.

1. The data bytes to be subtracted are already stored in memory registers 2051H and 2052H. We do not need to write instructions to store these bytes. You should store these bytes by using a keyboard of your trainer, or if you are using a simulator, you should observe the numbers in those memory locations when you store them.
2. The program should be written starting at memory location 2030H. This memory location is selected arbitrarily to emphasize that you can write a program beginning at any available memory location.



Instruction Word Size

- 1-byte instruction
- 2-byte instruction
- 3-byte instruction

For Example:

MOV C,A

MVI A,32H

LDA 2050H



Addressing modes of Intel 8085

Addressing modes refers to the different ways by which the operands can be specified in an instruction. The five-addressing modes of 8085 are

- i) **Register Addressing Mode:** Operands are one of the internal registers

MOV A,C

- ii) **Direct Addressing Mode:** Operands are the direct 16-bit memory address

STA 8000H

- iii) **Register Indirect Addressing Mode :** Operand is indirectly specified

MOV A, M

- iv) **Immediate Addressing Mode:** Operand is the immediate data

MVI A,32H

- v) **Implicit Addressing Mode:** Operands are implicit in this addressing mode

CMA, RLC,XCHG etc



Assembly Language Instruction

Typical Format:

[Label:]	Mnemonics	[Operands]	[;Comments]
	HLT		
	MVI	A,20H	
	MOV	M,A	; copy A-reg content to ;address specified by HL-register pair
LOAD:	LDA	2050H	; load A-reg with contents of memory ; location 2050H
READ:	IN	07H	; Read data from Input port with ; address 07H

Mnemonics:

The notations and conventions used in specifying the operand is called as Mnemonics.



Next Week

Unit 2: Assembly Language Programming



Thank You

29 October 2017

Pramod Ghimire

Slide 41 of 601



Chapter 6



Data Transfer(Copy) Instructions

**Presented By
Pramod Ghimire**



Properties

The data transfer instructions copy data from a source into a destination without modifying the contents of the source. The previous contents of the destination are replaced by the contents of the source.

Important: In the 8085 processor, data transfer instructions do not affect the flags.



Several Instructions

MOV : Move

MVI: Move Immediate

OUT: Output to port

IN: Input to port

Examples:

MOV A,B (1-byte)

MVI A,32H (2-byte)

OUT 21H (2-byte)

IN 60H (2-byte)



**Example
6.1**

Load the accumulator A with the data byte 82H (the letter H indicates hexadecimal number), and save the data in register B.

Instructions MVI A, 82H,
 MOV B,A

The first instruction is a 2-byte instruction that loads the accumulator with the data byte 82H, and the second instruction MOV B,A copies the contents of the accumulator in register B without changing the contents of the accumulator.



Arithmetic Operations

ADD: Add

ADI: Add Immediate

SUB: Subtract

SUI: Subtract Immediate

INR: Increment

DCR: Decrement

For Example

ADD C

ADI 100D

SUB B

SUI 10D

INR B

DCR C



Properties

These arithmetic instructions (except INR and DCR)

- 1) Assume implicitly that the A-reg is one of the operand
- 2) Modify all flags according to the data conditions of the result
- 3) Place the result in the A-reg

The instructions INR and DCR

- 1) Affect the contents of the specified register
- 2) Affect all flags except the **CY flag**



**Example
6.3**

The contents of the accumulator are 93H and the contents of register C are B7H. Add both contents.

Instruction ADD C

	CY	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
(A) :	93H =	1	0	0	1	0	0	1	1
	+								
(C) :	B7H =	1	0	1	1	0	1	1	1

1 1 1 1 1 1 1 1 1 Carry

SUM (A) : **[1]** 4AH = **[2]** 0 1 0 0 1 0 1 0 CY

Flag Status:[†] S = 0, Z = 0, CY = 1

When the 8085 adds 93H and B7H, the sum is 14AH; it is larger than eight bits. Therefore, the accumulator will have 4AH in binary, and the CY flag will be set. The result in the accumulator (4AH) is not 0, and bit D₇ is not 1; therefore, the Zero and the Sign flags will be reset.

*R represents any of registers A, B, C, D, E, H, and L.

[†]The P and AC flags are not shown here. In this chapter, the focus will be on the Sign, Zero, and Carry flags.

ADI 01H Vs INR A

Assume the accumulator holds the data byte FFH. Illustrate the differences in the flags set by adding 01H and by incrementing the accumulator contents.

Example
6.5

Instruction ADI 01H

$$\begin{array}{r} \text{CY} \\ (\text{A}) : \text{FFH} = 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ + \\ (\text{Data}) : \text{01H} = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\ \hline & 1\ 1\ 1\ 1\ 1\ 1\ 1\ \text{Carry} \end{array}$$

(A) : $\boxed{1}\ 00H = \boxed{1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0$
CY

Flag Status: S = 0, Z = 1, CY = 1

After adding 01H to FFH, the sum in the accumulator is 0 with a carry. Therefore, the CY and Z flags are set. The Sign flag is reset because D₇ is 0.

Instruction INR A

The accumulator contents will be 00H, the same as before. However, the instruction INR will not affect the Carry flag; it will remain in its previous status.

Flag Status: S = 0, Z = 1, CY = NA

Subtraction

The 8085 performs the following steps internally to execute the instruction SUB (or SUI)

- Step 1: Convert subtrahend (the number to be subtracted) into it's 1's complement
- Step 2: Adds 1 to 1's complement to obtain 2's complement of the subtrahend
- Step 3: Add 2's complement to the minuend (the contents of the accumulator)
- Step 4: Complement the Carry flag



Register B has 65H and the accumulator has 97H. Subtract the contents of register B from the contents of the accumulator.

Example
6.6

Instruction SUB B

Subtrahend (B): 65H = 0 1 1 0 0 1 0 1

Step 1: 1's complement of 65H = 1 0 0 1 1 0 1 0
(Substitute 0 for 1 and 1 for 0)

+

Step 2: Add 01 to obtain 0 0 0 0 0 0 0 1
2's complement of 65H = 1 0 0 1 1 0 1 1
+

To subtract: 97H – 65H,
Add 97H to 2's complement of 65H = 1 0 0 1 0 1 1 1

1 1 1 1 1 Carry

Step 3:

CY **1** 0 0 1 1 0 0 1 0
0 0 1 1 0 0 1 0

Step 4: Complement Carry

Result (A): 32H

Flag Status: S = 0, Z = 0, CY = 0

If the answer is negative, it will be shown in the 2's complement of the actual magnitude.
For example, if the above subtraction is performed as 65H – 97H, the answer will be the 2's complement of 32H with the Carry (Borrow) flag set.

Logic Operations

ANA: AND

ANI: AND Immediate

ORA: OR

ORI: OR Immediate

XRA: X-OR

XRI: X-OR Immediate

CMA: Complement A

For Example:

ANA B

ANI 0FH

ORA C

ORI 03H

XRA B

XRI 46H

CMA



Properties

The logic instructions

- 1) Implicitly assume that the A-reg is one of the operands
- 2) Reset (clear) the CY flag. The instruction CMA is an exception; it does not affect any flags.
- 3) Modify the Z,P, and S flags according to the data conditions of the result
- 4) Place the result in the A-reg
- 5) Do not affect the contents of the operand register

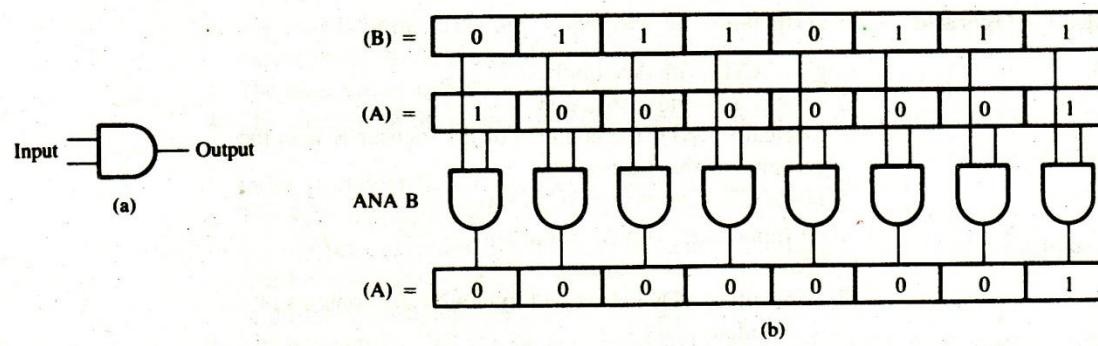


FIGURE 6.7
AND Gate (a) and a Simulated ANA Instruction (b)

OR,AND,NOT

Assume register B holds 93H and the accumulator holds 15H. Illustrate the results of the instructions ORA B, XRA B, and CMA.

**Example
6.7**

1. The instruction ORA B will perform the following operation:

$$\begin{array}{r} \text{OR} \\ \text{(B)} = 1 0 0 1 0 0 1 1 \quad (93H) \\ \text{(A)} = 0 0 0 1 0 1 0 1 \quad (15H) \\ \hline \text{(A)} = 1 0 0 1 0 1 1 1 \quad (97H) \end{array}$$

Flag Status: S = 1, Z = 0, CY = 0

The result 97H will be placed in the accumulator, the CY flag will be reset, and the other flags will be modified to reflect the data conditions in the accumulator.

2. The instruction XRA B will perform the following operation.

$$\begin{array}{r} \text{X-OR} \\ \text{(B)} = 1 0 0 1 0 0 1 1 \quad (93H) \\ \text{(A)} = 0 0 0 1 0 1 0 1 \quad (15H) \\ \hline \text{(A)} = 1 0 0 0 0 1 1 0 \quad (86H) \end{array}$$

Flag Status: S = 1, Z = 0, CY = 0

The result 86H will be placed in the accumulator, and the flags will be modified as shown.

3. The instruction CMA will result in

$$\begin{array}{r} \text{CMA} \\ \text{(A)} = 0 0 0 1 0 1 0 1 \quad (15H) \\ \text{(A)} = 1 1 1 0 1 0 1 0 \quad (EAH) \end{array}$$

The result EAH will be placed in the accumulator and no flags will be modified.



Setting and Resetting Specific Bits

**Example
6.8**

In Figure 6.8, keep the radio on (D_4) continuously without affecting the functions of other appliances, even if someone turns off the switch S_4 .

Solution

To keep the radio on without affecting the other appliances, the bit D_4 should be set by ORing the reading of the input port with the data byte 10H as follows:

$$\begin{array}{r} \text{IN } 00\text{H: } (A) = D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \\ \text{ORI } 10\text{H: } \quad = 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \\ \hline (A) = D_7 \ D_6 \ D_5 \ 1 \ D_3 \ D_2 \ D_1 \ D_0 \end{array}$$

Flag Status: CY = 0; others will depend on data.

The instruction IN reads the switch positions shown as D_7 - D_0 and the instruction ORI sets the bit D_4 without affecting any other bits.

In Figure 6.8, assume it is winter, and turn off the air conditioner without affecting the other appliances.

**Example
6.9**

To turn off the air conditioner, reset bit D_7 by ANDing the reading of the input port with the data byte 7FH as follows:

$$\begin{array}{r} \text{IN } 00\text{H: } (A) = D_7 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \\ \text{ANI } 7\text{FH: } \quad = 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ D_6 \ D_5 \ D_4 \ D_3 \ D_2 \ D_1 \ D_0 \end{array}$$

Flag Status: CY = 0; others will depend on the data bits.

The ANI instruction resets bit D_7 without affecting the other bits.

Solution

Branch Operations

The branch instructions are classified in three categories

- 1) Jump Instructions
- 2) Call and Return Instruction
- 3) Restart Instruction

For Example:

JMP START

JMP 5000H

JC

JNC

JZ

JNZ

JP

JM

JPE

JPO



Problems in Chapter 6

Question No 1: Specify the contents of the registers and the flag status as the following instructions are executed

Instruction	A	B	C	D	S	Z	CY
MVI A,00H	00	X	X	X	NA	NA	NA
MVI B,F8H	00	F8	X	X	NA	NA	NA
MOV C,A	00	F8	00	X	NA	NA	NA
MOV D,B	00	F8	00	F8	NA	NA	NA
HLT							



Question No 3: Write instructions to load the hexadecimal number 65H in register C, and 92H in the accumulator A. Display the number 65H at PORT 0 and 92H at PORT 1

MVI C,65H

MVI A,92H

OUT PORT 1 ; DISPLAY 92 AT PORT 1

MOV A,C

OUT PORT 0 ; DISPLAY 65 AT PORT 0

HLT



Question No 4: Write instructions to read the data at input PORT 07H and at PORT 08H. Display the input data from PORT 07H at PORT 00H, and store the input data from PORT 08H in register B.

IN 07H

OUT 00H

;Display data from input port 07h

IN 08H

MOV B,A

;Store data from port 08h in reg-B

HLT



Question No 5: Specify the output at PORT1 if the following program is executed.

MVI B,82H	; MOVE 82H TO B-REG
MOV A,B	; B-REG TO A-REG (A = 82H)
MOV C,A	; A-REG TO C-REG (C = 82H)
MVI D,37H	; D-REG = 37H
OUT PORT 1	; PORT 1 WILL HAVE A-REG CONTENT NOW
HLT	

Thus the output at PORT 1 = 82H



Question No 8: Specify the register contents and the flag status as the following instructions are executed. Specify also the output at PORT 0.

Inst	A	B	S	Z	CY
(Initial Values)	00	FF	0	0	0
MVI A,F2H	F2	FF	NA	NA	NA
MVI B,7AH	F2	7A	NA	NA	NA
ADD B	6C	7A	0	0	1
OUT PORT 0	6C	7A	NA	NA	
HLT					

Calculations:

$$F2H = \begin{array}{cccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}$$

$$+7AH = \begin{array}{cccccccc} 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \end{array}$$

$$16CH \quad \begin{array}{cccccccccc} 1 & \textcolor{red}{0} & 1 & \textcolor{red}{1} & 0 & & 1 & 1 & 0 & 0 \end{array}$$

CY S **6** C

Question No 9: What operation can be performed by using the instruction ADD A?

Solution:

The instruction ADD A will add the content of the accumulator to itself. This is eqvt to multiplying by 2.

Question No 10: What operation can be performed by using the instruction SUB A ? Specify the status of Z and CY.

Solution:

The instruction SUB A will clear the accumulator. The flag status will be

$$\text{CY} = 0$$

$$Z = 1$$



Question No 11: Specify the register contents and the flag status as the following instructions are executed.

	A	C	S	Z	CY
Initial Contents	XX	XX	0	0	0
MVI A,5EH	5E	XX	NA	NA	NA
ADI A2H	00	XX	0	1	1
MOV C,A	00	00	NA	NA	NA
HLT					
<u>Calculations:</u>					
	5EH =	0101	1110		
	+A2H =	1010	0010		
	<hr/>				
	100H	10000	0000		
		0	0	=> C = 1; S = 0, Z = 1	

Question No 13: Write a program using the ADI instruction to add the two hexadecimal numbers 3AH and 48H and to display the answer at an output port.

MVI A,3AH

ADI 48H

OUT PORT #

; # IS EQUIVALENT TO PORT NUMBER

HLT



Question No 14: Write instructions to

- a) Load 00H in the accumulator
- b) decrement the accumulator
- c) display the answer

MVI A,00H

DCR A

OUT PORT #

HLT

Calculations:

$$\begin{array}{r} (A) = \quad 0000 \quad 0000 \\ \quad -0000 \quad 0001 \\ \hline \end{array}$$

  1 1111 1111 => FFH
CY flag is not affected

The instruction DCR does not set the CY flag

Question No 15: The following instructions subtract two unsigned numbers. Specify the contents of register A and the status of S and CY. Explain the significance of the sign flag if it is set.

MVI A,F8H
SUI 69H

Calculations:

(A) = - 69H

$$\begin{array}{r} 1111 \\ -0110 \\ \hline 1000 \quad 1111 \end{array}$$

AC = 1

8 F

Solution:

A-reg = 8FH
S = 1
CY = 0

The sign flag has no meaning when subtracting unsigned numbers

Question No 18: Write a program to

- a) clear the accumulator
- b) add 47H (use ADI instruction)
- c) subtract 92H
- d) add 64H

Specify the answers you would expect at the output ports

SUB A	;clear A-reg, also can use XRA A
ADI 47H	
SUI 92H	
OUT PORT0	; display result 47H-92H
ADI 64H	
OUT PORT1	; display result after adding 64H



Calculations:

$$47H = 0100 \quad 0111$$

$$92H = 1001 \quad 0010$$

2's complement of 92H = 0110 1110

Now;

$$47H = 0100 \quad 0111$$

$$+92H = 0110 \quad 1110$$

no carry => **CY** 1011 0101 => B5H

Thus;

$$\begin{aligned} \text{Ans} &= -(\text{2's complement of } 1011 \quad 0101) \\ &= - (0100 \quad 1011) \\ &= -4BH \end{aligned}$$

To indicate (-), CY flag is set. But A contains **B5H**
(Please note that CY flag is also called as borrow flag.)

After Adding 64H:

$$\begin{array}{r} \text{B5H} = 1011 & 0101 \\ \text{64H} = 0110 & 0100 \\ \hline \text{carry} \Rightarrow & \boxed{1} & 0001 & 1001 & \Rightarrow 19\text{H} \end{array}$$

Thus:

(PORT0) = B5H

And (PORT1) = 19H



Question No 19: Specify the reason for clearing the accumulator before adding the number 47H directly to the accumulator in assignment 18.

SUB A ;clear A-reg, also can use XRA A

ADI 47H

SUI 92H

OUT PORT0 ; display result 47H-92H

ADI 64H

OUT PORT1 ; display result after adding 64H

Solution:

If a number is added before clearing the accumulator, the result will include the residual contents of the accumulator



Question No 20: What operations can be performed by using the instruction XRA A (Exclusive OR the contents of the accumulator with itself)? Specify the status of Z and CY.

Solution:

The instruction XRA clears the accumulator, , and the flag status is CY = 0, and Z = 1 .



Question No 21: Specify the register contents and the flag status (S,Z,CY) after the instruction ORA A is executed.

	A	B	S	Z	CY
MVI A,A9H	A9H	---	NA	NA	NA
MVI B,57H	A9H	57H	NA	NA	NA
ADD B	00H	57H	0	1	1
ORA A	00H	57H	0	1	0



Calculations:

CY = 1

AC = 1

$$(A) = A9H$$

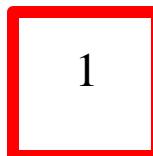
$$(B) = 57H$$



D7	D6	D5	D4	D3	D2	D1	D0
----	----	----	----	----	----	----	----

	1	0	1	0	1	0	0	1
--	---	---	---	---	---	---	---	---

	+	0	1	0	1	0	1	1
--	---	---	---	---	---	---	---	---



	0	0	0	0	0	0	0	0
--	---	---	---	---	---	---	---	---

$$(A) = 00H$$

$$(A) = 00H$$

	D7	D6	D5	D4	D3	D2	D1	D0
--	----	----	----	----	----	----	----	----

	0	0	0	0	0	0	0	0
--	---	---	---	---	---	---	---	---

	OR	0	0	0	0	0	0	0
--	----	---	---	---	---	---	---	---

	0	0	0	0	0	0	0	0
--	---	---	---	---	---	---	---	---

resets CY flag !

Question No 23: When the microprocessor reads an input port, the instruction IN does not set any flag. If the input reading is zero, what logic instruction can be used to set the zero flag without affecting the contents of accumulator?

Solution:

The instruction ORA A can set the zero flag without affecting the content of the accumulator.



Question No 24: Specify the register contents and the flag status as the following instructions are executed.

	A	B	S	Z	CY
	XX	XX	X	X	X
XRA A	00	---	0	1	0
MVI B,4AH	00	4A	NA	NA	NA
SUI 4FH	B1	4A	1	0	1
ANA B	00	4A	0	1	0
HLT					



Calculations:

A = 00H 0 0 0 0 0 0 0

$$4\text{FH} \quad = \quad 4\text{FH} \quad 0 \ 1 \ 0 \ 0 \quad 1 \ 1 \ 1 \ 1$$

$$\begin{array}{r} \text{2's complement of } 4\text{FH} \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$

And;

$$(00H) = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$$

$$(4FH) = \begin{matrix} & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{matrix}$$

No	<input type="text"/>	1	0	1	1	0	0	0	1
carry									
									
				B				1	

Thus Answer = **1B1H**



After ANA B instruction:

$$\begin{array}{rcl} \text{B1H} & = & 1\ 0\ 1\ 1 \\ 4\text{AH} & = & 0\ 1\ 0\ 0 \\ \hline & & 0\ 0\ 0\ 0 \end{array}$$

0 0 0 0 0 0 0 0

↓ ↓

0 0 => 00H

CY flag is reset by AND instruction



Question No 26: Load the data byte A8H in register C. Mask the high-order bits (D7-D4), and display the low-order bits (D3-D0) at an output port.

Solution:

MVI C,A8H	; load A8H in reg-C
MOV A,C	; to use AND operation, bring data to A-reg
ANI 0FH	; masking byte to mask D7-D0
OUT PORT0	; display the o/p
HLT	



Question No 27: Load the data byte 8EH in register D and F7H in register E. Mask the high-order bits (D7-D4) from both the data bytes. Exclusive-OR the low-order bits (D3-D0), and display the answer.

MVI D,8EH

MOV A,D

ANI 0FH

;mask D7-D4

MOV D,A

;save again in D-reg

MVI E,F7H

MOV A,E

ANI 0FH

;mask D7-D4

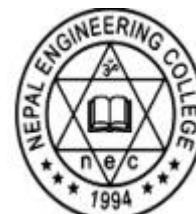
XRA D

;XOR A-reg content with D-reg content

OUT PORT0

; display the answer

HLT



Question No 28: Load the bit pattern 91H in register B and 87H in register C. Mask all the bits except D0 from registers B and C. If D0 is at logic 1 in both registers, turn on the light connected to the D0 position of output port 01H; otherwise, turn off the light.

D7	D6	D5	D4		D3	D2	D1	D0
0	0	0	0		0	0	0	1
		↓		0		↓		

MVI B,91H

MVI C,87H

MOV A,B

ANI 01H

; mask all bits of 91H except D0

MOV B,A

; save D0 from first byte

MOV A,C

ANI 01H

; mask all bits of 87H except D0

ANA B

; AND bits D0 of 91H and 87H

OUT PORT1

; Turn on/off light connected to D0

HLT



Question No 30: What is the output at PORT1 when the following instructions are executed.

MVI A,8FH

ADI 72H

JC DISPLAY

OUT PORT1

HLT

DISPLAY: XRA A

OUT PORT 1

HLT

Calculations:

$$A = 8FH$$

$$+ 72H$$

$$\begin{array}{r} 1 \\ 0 \\ 0 \\ 0 \end{array}$$

$$\begin{array}{r} 0 \\ 1 \\ 1 \\ 1 \end{array}$$

$$\begin{array}{r} 1 \\ 1 \\ 1 \\ 1 \end{array}$$

$$\begin{array}{r} 0 \\ 0 \\ 1 \\ 0 \end{array}$$

$$\begin{array}{r} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}$$

CY

0

$$\begin{array}{r} 0 \\ 0 \\ 0 \\ 1 \end{array}$$

1

Thus output = 01H

This program displays the sum at output if there is no carry.

If there is a carry, it displays 00H at the output port.

Question No 31: In Question 30, replace the instruction ADI 72H by the instruction SUI 67H, and specify the output.

$$\begin{array}{rcl} A & = & 8\text{FH} \\ & & \begin{array}{cccccc} 1 & 0 & 0 & 0 & & \\ & & & & & \end{array} \\ - & 67\text{H} & - \quad \begin{array}{cccccc} 0 & 1 & 1 & 0 & & \\ & & & & & \end{array} \\ \hline & & \begin{array}{cccccc} 1 & 1 & 1 & 1 & & \\ & & & & & \end{array} \\ & & \begin{array}{cccccc} 0 & 1 & 1 & 1 & & \\ & & & & & \end{array} \end{array}$$

$$\begin{array}{ccccccccc} & 0 & 0 & 1 & 0 & & 1 & 0 & 0 \\ & & \downarrow & & & & & \downarrow & \\ & 2 & & & & & 8 & & \end{array}$$

Output = 28H (because there is no carry in this case)

Question No 32: In the following program, explain the range of the bytes that will be displayed at PORT 2.

MVI A,BYTE1
MOV B,A
SUI 50H
JC DELETE ;if BYTE1 is less than 50H, there will be
; carry, if so jump to DELETE
MOV A,B
SUI 80H
JC DISPLAY ; if BYTE1 is less than 80H, there will be carry
; if so jump to DISPLAY
DELETE: XRA A
OUT PORT1
HLT
DISPLAY:MOVA,B
OUT PORT2 ; displayed output will be more than 49H but less than 80H
; (i.e 50H to 7FH)
HLT



Question No 33: Specify the address of the output port, and explain the type of numbers that can be displayed at the output port.

MVI A,BYTE ;get a data byte

ORA A ;set flags

JP OUTPRT ;jump if the byte is positive

XRA A

OUTPRT: OUT F2H

HLT

Solution:

The address of the outport port = F2H

All positive signed numbers and zero will be displayed at port F2H.



Question No 34: In Question 33, if BYTE1 = 92H, what is the output at port F2H ?

92H = 1 0 0 1 0 0 1 0

OR

92H = 1 0 0 1 0 0 1 0

1 0 0 1 0 0 1 0

S = 1

=> Number is negative, so o/p displayed will be 00H.



Question No 35: Explain the function of the following program. (2014 university exam)

```
MVI A,BYTE1      ; get a data byte  
ORA A            ; set flags (S,Z,P will change  
                  ; but CY and AC will be reset)  
JM OUTPRT       ; jump on minus to  
OUT 01H  
HLT  
OUTPRT: CMA  
ADI 01H  
OUT 01H  
HLT
```

Solution:

This routine displays the absolute value (magnitude) of BYTE1



Question No 36: In Question 35, if BYTE1 = A7H, what will be displayed at port 01H.

Solution:

	D7	D6	D5	D4		D3	D2	D1	D0
A7 =	1	0	1	0		0	1	1	1
(S = 1)	=> number is negative								

It's 2's complement =>

0	1	0	1	1	1	0	0	0	0
					+ 1				

0	1	0	1	1	1	0	0	1
				↓				
				5				
				↓				
				9				

=> 59H will be displayed.



Question No 38: Write instructions to clear the CY flag, to load number FFH in register B, and increment (B). If the CY flag is set, display 01 at the output port; otherwise, display the contents of register B. Explain your result.

XRAA ; clear CY flag (also can use ANA A,
SUB A, ORA A instead of XRA A)
MVI B,FFH ; LOAD FFH INTO B
INR B ; increment B (**S,Z,P,AC** are modified but CY is not ;modified)
MOV A,B
JNC DISPLAY
MVI A,01H
DISPLAY: OUT PORT0 ;output = 00H because INR does not set CY flag
HLT



Question No 39: Write instructions to clear the CY flag, to load number FFH in register C, and to add 01 to (C). If the CY flag is set, display 01 at an output port; otherwise, display the contents of register C. Explain your results. Are they the same as in Question 38 ?

ANAA ; clear the CY flag
MVI C,FFH ; LOAD FFH INTO C
MOV A,C ; to add 01H to C-reg, bring it to A-reg to add 01H
ADI 01H
JNC DISPLAY
MVI A,01H

DISPLAY: OUT PORT0 ;output = 01H because ADI sets CY flag
HLT

Calculations:

FFH	=	1	1	1	1	1	1	1	1
01H	=	0	0	0	0	0	0	0	1
<hr/>									
		1	0	0	0	0	0	0	0
		CY							



Question No 40: Write instructions to load two unsigned numbers in register B and register C, respectively. Subtract (C) from (B). If the result is in 2's complement, convert the result in absolute magnitude and display it at PORT1; otherwise, display the positive result.

MVI B,BYTE1

MVI C,BYTE2

MOV A,B

SUB C ; performs now B-C

JNC DISPLAY ; jump if result is positive

CMA ; take 1's complement

ADI 01H ; find 2's complement

DISPLAY: OUT PORT1

HLT



Chapter 7



Additional Instructions

- Looping
- Counting
- Indexing
- LXI,LDAX,STAX,INX,DCX,
- RLC,RAL,RRC,RAR,
- CMP,CPI



LXI : Load Immediate 16-bit data in register pair

- i. LXI B, 2050H
 - ii. LXI D,2070H
 - iii. LXI H,2090H
 - iv. LXI SP, 0FFFFH
- The LXI instruction perform functions similar to those of the MVI instructions, except that the LXI instructions load 16-bit data in register pairs and the stack pointer register (SP). These instructions do not affect the flags
 - This is a 3-byte instruction
 - In example (i), 50H is loaded in register C and 20H in register B



Example 7.2

Write instructions to load the 16-bit number 2050H in the register pair HL using LXI and MVI opcodes and explain the difference between the two instructions.

Method 1:

LXI H,2050H

Method 2:

MVI H,20H

MVI L,50H

Difference:

The LXI is functionally similar to two MVI instructions. One LXI is a 3-byte instruction whereas two MVI instructions are eqvt to 4-byte instruction.

MOV C,M

This is a 1-byte instruction that copies data from the memory location specified by the contents of HL register to C register

STAX B/D : Store Accumulator Indirect

This is a 1-byte instruction that copies data from the accumulator into the memory location specified by the contents of either BC or DE registers

eg: STAX B

STAX D

STA 2050H

This is a 3-byte instruction. **Stores Accumulator Direct** to the memory location specified by the 16-bit operand



INX Rp: Increment Register Pair

- ✓ This is a 1-byte instruction
- ✓ It treats the contents of two registers as one 16-bit number and increases the contents by 1
- ✓ The instruction set includes four instruction.

e.g: INX B

INX D

INX H

INX SP

(These instructions do not affect flags)



DCX Rp: Decrement Register Pair

- ✓ This is a 1-byte instruction
- ✓ It decreases the 16-bit contents of a register pair by 1
- ✓ The instruction set includes four instructions

e.g: DCX B

DCX D

DCX H

DCX SP

(These instructions do not affect flags)



Example 7.2.6: Sixteen bytes of data are stored in memory locations 2050H to 205FH. Transfer the entire block of data to new memory locations starting at 2070H

LXI H,2050H	; set HL as a pointer for the source memory
LXI D,2070H	; set DE as a pointer for the destination memory
MVI B,10H	; set up as byte counter
NEXT: MOV A,M	; get data byte from the source memory
STAX D	; store the data byte in destination memory
INX H	
INX D	; get ready to transfer next byte
DCR B	
JNZ NEXT	; go back to get next byte if byte counter is not zero
HLT	

Memory Location	Contents
HL=>2050H	10H
2051H	20H
2052H	30H
DE=>2070H	
2071H	



Arithmetic Operations Related to Memory

ADD M

Add Memory. This is a 1-byte instruction. It adds (M) to (A) and stores the result in A. The memory location is specified by the contents of HL

SUB M

Subtract Memory. This is a 1-byte instruction. It subtracts (M) from (A) and stores the result in A. The memory location is specified by (HL)

INR M

This is a 1-byte instruction. It increments the contents of a memory location by 1, not the memory address. The memory location is specified by (HL). All flags except the CY flag are affected.

DCR M

This is a 1-byte instruction. It decrements (M) by 1. The memory location is specified by (HL). All flags except the CY flag are affected.



Example 7.3.2: Six bytes of data are stored in memory locations starting at 2050H. 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.

```

XRA A
MOV B,A
MVI C,06H
LXI H,2050H
NXTBYT: ADD M
    JNC NXTMEM
    INR B
NXTMEM: INX H
    DCR C
    JNZ NXTBYT
    OUT PORT1
    MOV A,B
    OUT PORT2
    HLT
  
```

Memory Location	Contents
HL => 2050H	A2H
2051H	FAH
2052H	DFH
2053H	E5H
2054H	98H
2055H	8BH

LOGIC OPERATIONS:ROTATE



- **RLC : Rotate Accumulator Left**
- **RAL: Rotate Accumulator Left Through Carry**
- **RRC : Rotate Accumulator Right**
- **RAR: Rotate Accumulator Right Through Carry**

RLC

Each binary bit of the accumulator (A-reg) is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the CY flag. CY is modified according to bit D7. S,Z,P,AC are not affected.

Accumulator contents
before instruction

CY	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	1	0	1	0	0	1	1	1

Accumulator contents
after instruction

CY	0	1	0	0	1	1	1	1
1	0	1	0	0	1	1	1	1

RAL:

Each binary bit of the accumulator (A-reg) is rotated left by one position through the carry flag. Bit D7 is placed in the CY flag and the bit in the CY flag is placed in D0. CY is modified according to bit D7. S,Z,P,AC are not affected.

Accumulator content
before instruction

CY	0						
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	1	0	0	1	1	1

Accumulator contents
after instruction

CY	1						
0	1	0	0	1	1	1	0
0	1	0	0	1	1	1	0

Each binary bit of the accumulator (A-reg) is rotated left by one position. Bit D0 is placed in the position of D7 as well as in the CY flag. CY is modified according to bit D0. S,Z,P,AC are not affected.

Accumulator contents
before instruction

CY							
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	1	0	1	0	0	1	1

Accumulator contents
after instruction

CY							
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	0	1	0	0	1	1

RAR:

Each binary bit of the accumulator (A-reg) is rotated left by one position through the carry flag. Bit D0 is placed in the CY flag and the bit in the CY flag is placed in D7. CY is modified according to bit D7. S,Z,P,AC are not affected.

Accumulator contents
before instruction

CY	0						
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	1	0	0	1	1	1
1	0	1	0	0	1	1	1

Accumulator contents
after instruction

CY	1						
0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	1

Applications of Rotate Instructions

- ✓ Multiplication
- ✓ Division
- ✓ Serial Data Transfer

For example, if (A) is

0000 1000 = **08H**

By rotating 08H right: (A)

0000 0100 = **04H**

This is equivalent to dividing by 2

By rotating 08H left: (A)

0001 0000 = **10H**

This is equivalent to multiplying by 2 (10H = 16D)



Example 7.4.2: A set of ten current readings is stored in memory location starting at 2060H. The readings are expected to be positive (<127D). Write a program to

1. check each reading to determine whether it is positive or negative
2. reject all negative readings
3. add all positive readings.
4. output FFH to PORT1 at any time when the sum exceeds eight bits to indicate the overload; otherwise, display the sum.

Solution:

Memory Location	Contents
HL=>2060H	28H
2061H	D8H
2062H	C2H
2063H	21H
2064H	24H
2065H	30H
2066H	2FH
2067H	19H
2068H	F2H
2069H	9FH



MVI B,00H
MVI C,0AH
LXI H,2060H
NEXT: **MOV A,M**
RAL
JC REJECT
RAR
ADD B
JC OVERLOAD
MOV B,A
REJECT: **INX H**
DCR C
JNZ NEXT
MOV A,B
OUT PORT1
HLT
OVERLOAD: **MVI A,FFH**
OUT PORT1
HLT

Memory Location	Contents
HL=>2060H	28H
2061H	D8H
2062H	C2H
2063H	21H
2064H	24H
2065H	30H
2066H	2FH
2067H	19H
2068H	F2H
2069H	9FH



LOGIC OPERATIONS: COMPARE



LOGIC OPERATIONS: COMPARE

The 8085 instruction set has two types of Compare operations:
CMP and **CPI**

CMP: Compare with Accumulator

CPI: Compare Immediate (with Accumulator)

8085 compares a data byte with the contents of the accumulator by subtracting the data byte from (A), and indicates whether the data byte is $\geq \leq$ (A) by modifying the flags. However, the contents are not modified.



Instructions

CMP B : Compare register-B with Accumulator

This is a 1-byte instruction. It compares the data byte in register with the contents of the accumulator.

- i. If $(A) < (B)$, the **CY** flag is set and **Z** flag is reset
- ii. If $(A) = (B)$, the **Z** flag is set and **CY** flag is reset
- iii. If $(A) > (B)$, the **CY** and **Z** flags are reset

No contents are modified; however, all remaining flags (S,P,AC) are affected according to the result of the instruction.

CMP M : Compare Memory with Accumulator (1-byte long)

CPI 64H : Compare Immediate data with Accumulator (2-byte long)



Example 7.5.2: A set of current readings is stored in memory locations starting at 2050H. The end of the data string is indicated by the data byte **00H**. Add the set of readings. The answer may be larger than FFH. Display the entire sum at PORT1 and PORT2.

Solution:

Memory Location	Contents
HL=>2050H	32H
2051H	52H
2052H	F2H
2053H	A5H
2054H	24H
2055H	30H
2056H	2FH
2057H	19H
2058H	F2H
2059H	00H



	LXI H,2050H
NXTBYT:	MVI C,00H
	MOV B,C
	MOV A,M
	CPI 00H
	JZ DSPLAY
	ADD C
	JNC SAVE
	INR B
SAVE:	MOV C,A
	INX H
	JMP NXTBYT
DSPLAY:	MOV A,C
	OUT PORT1
	MOV A,B
	OUT PORT2
	HLT

Memory Location	Contents
HL=>2050H	32H
2051H	52H
2052H	F2H
2053H	A5H
2054H	24H
2055H	30H
2056H	2FH
2057H	19H
2058H	F2H
2059H	00H

University Question 1: Write a program for 8085 to arrange 10 bytes of data in ascending order. The data is stored in memory as an array starting from C100H.

	MVI B,0AH	
	DCR B	;COUNTER 1
START:	MVI C,9H	;COUNTER 2
	LXI H,0C100H	;INITIALIZE THE POINTER
BACK:	MOV A,M	
	INX H	
	CMP M	;COMPARE TWO NUMBER
	JC SORTED	
	MOV D,M	;EXCHANGE THE TWO NUMBER
	MOV M,A	
	DCX H	
	MOV M,D	
	INX H	
SORTED:	DCR C	
	JNZ BACK	
	DCR B	
	JNZ START	;START WITH THE ELEMENT NUMBER 1
	HLT	

(Also see page 259 in the text book,5e)



University Question 2: A sequence of ten unsigned numbers is stored at memory location C0C0H. Develop an ALP to find out the greatest number with comment.

MVI C,9H	;COUNTER
LXI H,0C0C0H	;INITIALIZE
MOV A,M	
NEXT: INX H	;POINT TO NEXT ELEMENT
CMP M	;AND THEN COMPARE WITH THE CONTENT IN A
JNC SKIP	
MOV A,M	;IF CARRY,NUMBER IN “M” IS GREATER,MAKE ;THAT NUMBER A REFERENCE
SKIP: DCR C	
JNC NEXT	
HLT	

**(NOTE THAT THE ANSWER IS IN A-REG,OUT 01H CAN BE USED TO
DISPLAY IT JUST BEFORE THE HLT)**



Example 7.5.3: A set of three readings is stored in memory starting at 2050H. Sort the readings in ascending order.

Solution:

Memory Location	Contents
2050H	87H
2051H	56H
2052H	42H

Bubble sort:

The technique involved is comparing two bytes at a time and placing them in proper sequence. For example, we compare the first two bytes (87H and 56H), and if the first byte is larger than the second byte, we will exchange their memory locations to arrange them in ascending order; otherwise, we will keep them in the same locations. We will follow the same procedure for the second and third bytes. The number of comparisons necessary is always $N-1$ where N is the number of bytes. Therefore, we need a counter of 2 for one complete comparison.

START:	LXI H,2050H	; set up HL as a memory pointer for bytes
	MVI D,00H	; clear register D to set up a flag
	MVI C,02H	; set register C for comparison count
CHECK:	MOV A,M	; get data byte
	INX H	; point to next byte
	CMP M	; compare bytes
	JC NXTBYT	; if (A) < second byte, do not exchange
	MOV B,M	; get second byte for exchange
	MOV M,A	; store first in second location
	DCX H	; point to first location
	MOV M,B	; store second byte in first location
	INX H	; get ready for next comparison
	MVI D,01H	; load 1 in D as a reminder for exchange
NXTBYT:DCR C		; decrement comparison count
	JNZ CHECK	; if comparison is not zero, go back
	MOV A,D	; get flag bit in A
	RRC	; place flag bit D0 in CY
	JC START	; if CY is 1, exchange occurred, go for next pass
	HLT	; end of sorting



Execution of Sort Program

Memory Locations	Initial Bytes	First Pass	Second Pass	Third Pass
2050H	87	56	42	42
2051H	56	42	56	56
2052H	42	87	87	87
Reg - D	0	1	1	0



Problems in Chapter 7



Question No 5: Specify the memory location and its contents after the following instructions are executed.

MVI B,F7H

MOV A,B

STA XX75H

HLT

Solution:

Assume that XX = 20

Thus XX75H = 2075H

Answer:

Location 2075H contains F7H.



Question No 6: Show the register contents as each of the following instructions is being executed.

	A	B	C	D	E	H	L
MVI C,FFH	-	-	FF	-	-	-	-
LXI H,2070H	-	-	FF	-	-	20	70
LXI D,2070H	-	-	FF	20	70	20	70
MOV M,C	-	-	FF	20	70	20	70
LDAX D	FF	-	FF	20	70	20	70
HLT	FF	-	FF	20	70	20	70

Question No 7: In Question 6, specify the contents of accumulator and the memory location 2070H after the execution of the instruction LDAX D.

A = FFH

(2070H) = FFH

Thus the content of 2070H is FFH.



Question No 8: Identify the memory locations that are cleared by the following instructions.

MVI B,00H

LXI H,2075H

MOV M,B

INX H

MOV M,B

HLT

MVI B,00H loads B-reg with 00H.

LXI H,2075H loads HL with 2075H

MOV M,B => places 00H in 2075H

INX H increases HL by 1, HL points to 2076H

MOV M,B => places 00H in 2076H

Answer: 2075H and 2076H



Question No 9: Specify the contents of registers A,D, and HL after execution of the following instructions.

LXI H,2090H	; set up register HL as a memory pointer
SUB A	; clear accumulator
MVI D,0FH	; set up register D as counter
LOOP: MOV M,A	; clear memory
INX H	; point to the next memory location
DCR D	; update counter
JNZ LOOP	; repeat until (D) = 0
HLT	

Solution:

A = 00H

D = 00H

$$\begin{aligned} \text{HL} &= 2090\text{H}+0\text{FH} \\ &= 209\text{FH} \end{aligned}$$

The loop runs $0F = 15$ times, and hence HL will be updated to 209FH. Also all the locations will contain 00H. (i.e clears locations from 2090H to 209FH)



Question No 11: Rewrite the instructions in Question 9 using the register BC as a memory pointer.

LXI B,2090H
SUB A
MVI D,0FH
LOOP: STAX B
INX B
DCR D
JNZ LOOP
HLT

Memory Location	Contents
	2093H
	2092H
	2091H
BC=>	2090H



Question No 12: Explain how many times the following loop will be executed.

LXI B,0007H

LOOP: DCX B

JNZ LOOP

Solution:

Infinite Loop!! DCX instruction does not affect Z flag. Infact it does not affect any flags at all !!!



Question No 13: Explain how many times the following loop will be executed.

LXI B,0007H ; BC = 00 07H
LOOP: DCX B ; BC = BC -1
MOV A,B ; A contains 00H
ORA C ; C decreases each time; after 7 times, C = 00H and ORing with
; A sets zero flag
JNZ LOOP

Solution:

Loop runs 7 times.

Calculations:

(A) = 0000 0000

OR

(C) = 0000 0000

0000 0000 => 00H

AC = 0, Z = 1, S = 0, P = 1, CY = 0



Question No 14: The following instructions are intended to clear ten memory locations starting from the memory address 0009H. Explain why a large memory block will be erased or cleared and the program will stay in an infinite loop.

LXI H,0009H

LOOP: MVI M,00H

DCX H

JNZ LOOP

HLT

Answer:

DCX instruction does not affect Z flag.



Question No 15: The following block of data is stored in the memory locations from 2055H to 205AH. Transfer the data to the locations 2080H to 2085H in the reverse order. (e.g the data byte 22H should be stored at 2085H and 37H at 2080H)

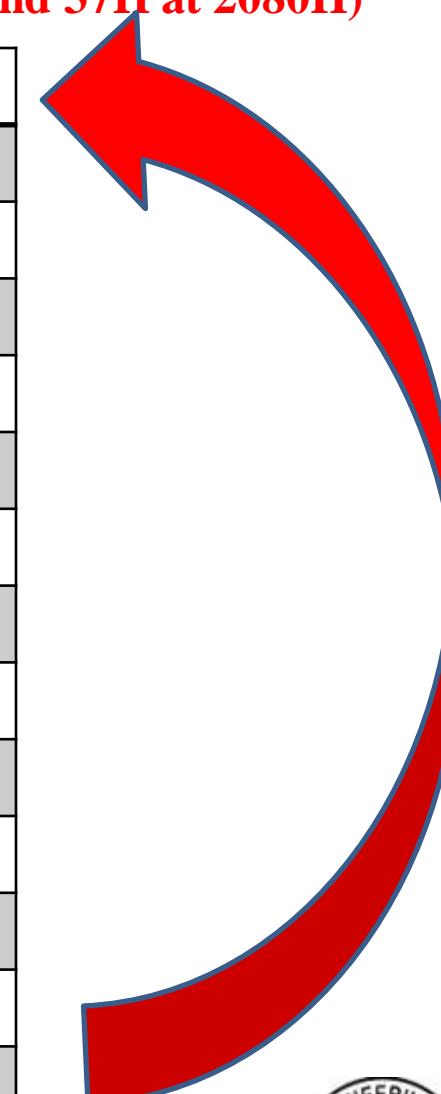
Data (H): 22,A5,B2,99,7F,37

DE =>

START:	LXI H,2055H	;source
	LXI D,2085H	;destination
	MVI B,06H	;counter
NEXT:	MOV A,M	;get data byte
	STAX D	;store data byte
	INX H	
	DCX D	
	DCR B	;next count
	JNZ NEXT	;repeat
	HLT	

HL =>

2085H	
2084H	
2083H	
2082H	
2081H	
2080H	
205AH	37
2059H	7F
2058H	99
2057H	B2
2056H	A5
2055H	22



Question No 18: A string of six data bytes is stored starting from memory location 2050H. The string includes some blanks(bytes with zero value). Write a program to eliminate the blanks from the string(Hint: To check a blank, set the zero flag by using the ORA. Use two memory pointers: one to get a byte and other to store the byte.

Data (H): F2,00,00,4A,98,00

START: MVI B,6	;counter
LXI H,2050H	;source
LXI D,2050H	;destination
LOOP: MOV A,M	;get byte
ORAA	;test if zero flag is set
JZ SKIP	
STAX D	;if not zero , store it and increment
INX D	; the index for next save
SKIP:INX H	; go to check next byte
DCR B	
JNZ LOOP	
HLT	

HL ,DE=>

Memory Locations	Contents
2055H	00
2054H	98
2053H	4A
2052H	00
2051H	00
2050H	F2



Question No 19: Write a program to add the following five data bytes stored in memory locations starting from 2060H, and display the sum. (The sum does not generate a carry. Use register pair DE as a memory pointer to transfer a byte from memory into a register).

Data (H): 1A,32,4F,12,27

START: LXI D,2060H	;index for data
MVI C,05H	; counter for data
MVI B,00H	; clear B
NEXT: LDAX D	; load data
ADD B	; add data byte
MOV B,A	; save partial sum
INX D	
DCR C	; next count
JNZ NEXT	
OUT PORT1	; display sum
HLT	

DE=>

Memory Locations	Contents
2065H	
2064H	27
2063H	12
2062H	4F
2061H	32
2060H	1A



Question No 20: Write a program to add the following data bytes stored in memory locations starting at 2060H and display the sum at the output port if the sum does not generate a carry . If a result generates a carry, stop the addition, and display 01H at the output port.

Data (H): First set: 37,A2,14,78,97

START: LXI H,2060H

;index for data

MVI C,05H

;counter

MVI B,00H

;clear B to store sum

NEXT: MOV A,M

ADD B

JC CARRY

MOV B,A

DCR C

JNZ NEXT

OUT PORT1

HLT

CARRY: MVI A,01H

OUT PORT1

HLT

DE=>

Memory Locations	Contents
2065H	
2064H	97
2063H	78
2062H	14
2061H	A2
2060H	37



Question No 22: Specify the contents of memory locations 2070H to 2074H after execution of the following instructions.

LXI H,2070H
MVI B,05H
MVI A,01H
STORE: MOV M,A
INR A
INX H
DCR B
JNZ STORE
HLT

Memory Locations	Contents
2075H	
2074H	05
2073H	04
2072H	03
2071H	02
2070H	01

Solution:

Locations 2070H to 2072H will contain 01H,02H,03H,04H,05H

HL =>



Question No 23: Specify the register contents of the registers, the memory location 2055H, and the flags as the following instructions are executed.

	A	H	L	S	Z	CY	M (2055H)
LXI H,2055H	--	20	55	NA	NA	NA	NA
MVI M,8AH	--	20	55	NA	NA	NA	8A
MVI A,76H	76	20	55	NA	NA	NA	8A
ADD M	00	20	55	0	1	1	8A
STA 2055H	00	20	55	NA	NA	NA	00
HLT							

Calculations:



AC = 1

$(M) = 8AH$ 1 0 0 0 1 0 1 0

$(A) = 76H$ 0 1 1 1 0 1 1 0

CY = 1

0 0 0 0 0 0 0 0

Question No 25: Identify the contents of the memory locations 2065H and the status of the flags S, Z, and CY when the instruction INR M is executed .

LXI H,2065H ; HL = 2065H

MVI M,FFH ; (2065H) contains FFH

INR M ; (2065H) contains 00H , all flags except CY affected

HLT

Calculations:



(M) = FFH

1 1 1 1

(A) = 01H

0 0 0 0



AC = 1

1 1 1 1

0 0 0 1

Thus :

S = 0

Z= 0

CY = NA

by INR

CY = 1

0 0 0 0



Chapter 8



Counters and Time Delays

- A counter is designed by loading an appropriate count in a register. A loop is set up to decrement the count for a down-counter or to increment the count for an up-counter.
- A timing delay is designed by loading a register with a delay count and setting up a loop to decrement the count until zero. The delay is determined by the clock period of the system and the time required to execute the instructions in the loop

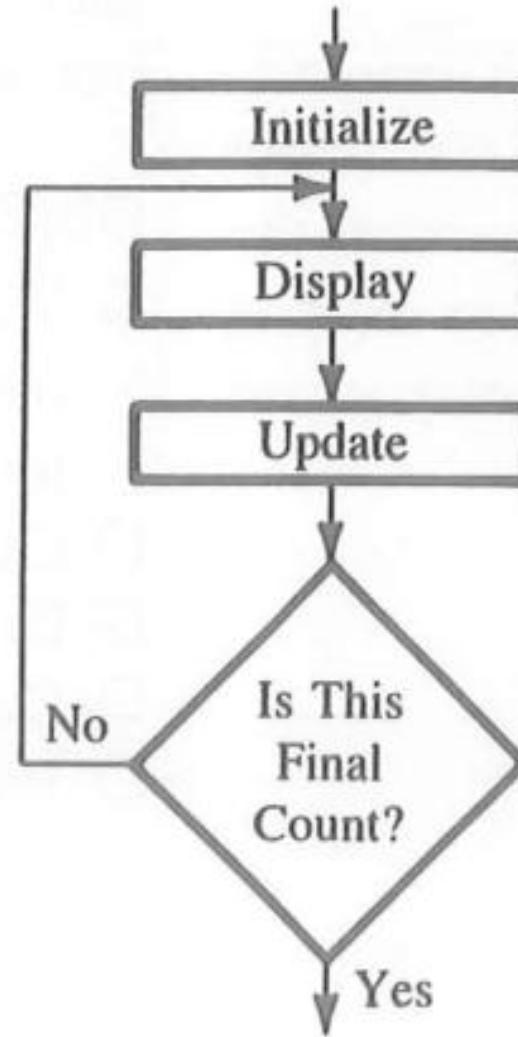
Objectives:

1. Time delays using **one register**
2. Using a **register pair**
3. Using a **loop-within-a-loop**



FIGURE 8.1

Flowchart of a Counter



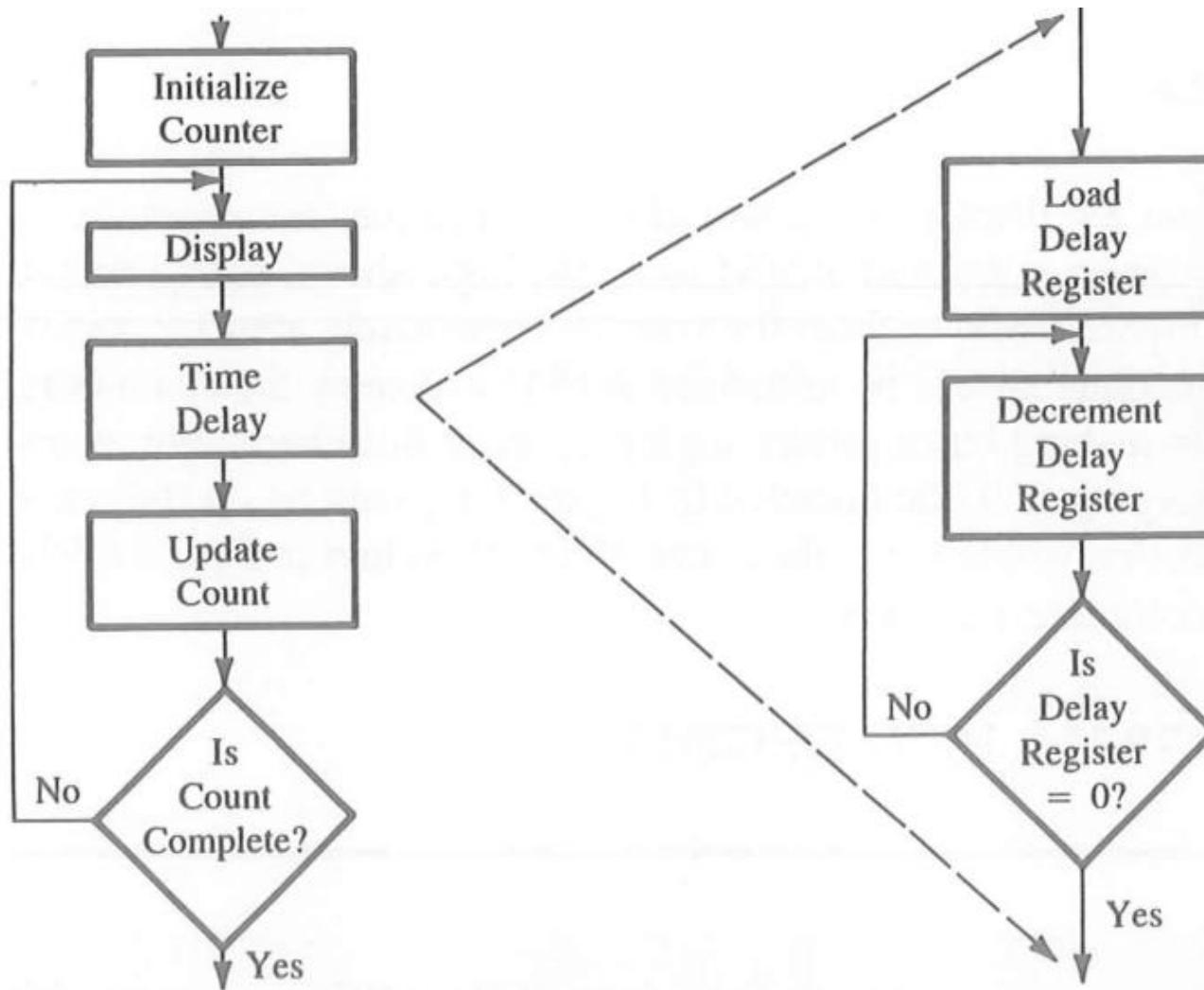


FIGURE 8.4
Flowchart of a Counter with a Time Delay

COUNTERS AND TIME DELAYS

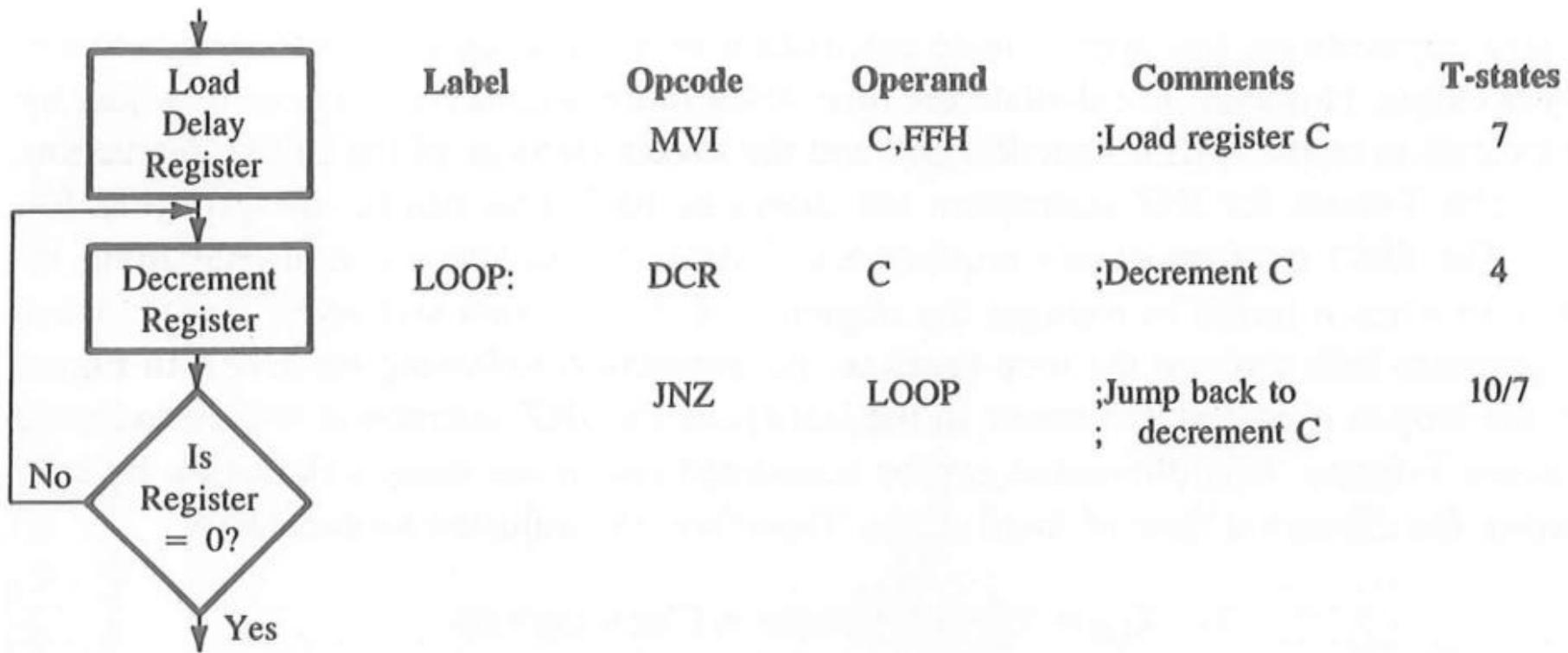


FIGURE 8.2
Time Delay Loop: Flowchart and Instructions

The time delay in the loop T_L with 2MHz clock frequency is calculated as

$$T_L = (T \times \text{Loop T-states} \times N10)$$

where

T_L = Time delay in the loop

T = System clock period

$N10$ = eqvt decimal number of the hexadecimal count loaded in the delay register

Time Delay Calculations:

$$T_L = (0.5 \times 10^{-6} \times 14 \times 255)$$

$$= 1785 \mu\text{s}$$

$$= 1.8 \text{ ms}$$

Adjusted Time Delay Calculations:

$$T_{LA} = T_L - (3 \text{ T-states} \times \text{Clock period})$$

$$= 1785 \mu\text{s} - 1.5 \mu\text{s}$$

$$= 1783.5 \mu\text{s}$$

Total Delay:

T_D = Time to execute instructions outside loop + Time to execute loop instructions

$$= T_O + T_{LA}$$

$$= (7 \times 0.5 \mu\text{s}) + 1783.5 \mu\text{s} = 1787 \mu\text{s}$$

$$= 1.8 \text{ ms}$$

Important:

- ✓ The difference between the loop delay T_L and these calculations is only 2 μs and can be ignored in most instances.
- ✓ The time delay can be varied by changing the count FFH; however , to increase the time delay beyond 1.8 ms in a 2 MHz microcomputer system, a register pair or loop within a loop technique should be used.



Time Delay Using a Register Pair

Label	Opcode	Operand	Comments	T-states
LOOP:	LXI	B,2384H	;Load BC with 16-bit count	10
	DCX	B	;Decrement (BC) by one	6
	MOV	A,C	;Place contents of C in A	4
	ORA	B	;OR (B) with (C) to set Zero flag	4
	JNZ	LOOP	;If result ≠ 0, jump back to LOOP	10/7

Note:

The time delay can be considerably increased by setting a loop and using a register pair with a 16-bit number (maximum FFFFH). The 16-bit number is decremented by using the instruction DCX. However, the instruction DCX does not set the Zero flag and, without the test flags, Jump instructions cannot check desired data conditions. Additional techniques, therefore, must be used to set the Zero flag



Delay Calculations

$$\begin{aligned}2384H &= 2 \times (16)^3 + 3 \times (16)^2 + 8 \times (16)^1 + 4(16^0) \\&= 9092_{10}\end{aligned}$$

If the clock period of the system = 0.5 μ s, the delay in the loop T_L is

$$\begin{aligned}T_L &= (0.5 \times 24 \times 9092_{10}) \\&\approx 109 \text{ ms (without adjusting for the last cycle)}$$

$$\begin{aligned}\text{Total Delay } T_D &= 109 \text{ ms} + T_O \\&\approx 109 \text{ ms (The instruction LXI adds only 5 } \mu\text{s.)}\end{aligned}$$

Loop Within a Loop

8.1.3 Time Delay Using a Loop within a Loop Technique

A time delay similar to that of a register pair can also be achieved by using two loops; one loop inside the other loop, as shown in Figure 8.3(a). For example, register C is used in the inner loop (LOOP1) and register B is used for the outer loop (LOOP2). The following instructions can be used to implement the flowchart shown in Figure 8.3(a).

MVI B,38H	7T
LOOP2: MVI C,FFH	7T
LOOP1: DCR C	4T
JNZ LOOP1	10/7T
DCR B	4T
JNZ LOOP2	10/7T

DELAY CALCULATIONS

The delay in LOOP1 is $T_{L1} = 1783.5 \mu\text{s}$. These calculations are shown in Section 8.1.1. We can replace LOOP1 by T_{L1} , as shown in Figure 8.3(b). Now we can calculate the delay in LOOP2 as if it is one loop; this loop is executed 56 times because of the count (38H) in register B:

$$\begin{aligned} T_{L2} &= 56(T_{L1} + 21 \text{ T-states} \times 0.5 \mu\text{s}) \\ &= 56(1783.5 \mu\text{s} + 10.5 \mu\text{s}) \\ &= 100.46 \text{ ms} \end{aligned}$$

Generating Pulse Waveform

Example 8.4:

Write a program to generate a continuous square wave with the period of $500 \mu\text{s}$. Assume the system clock period is 325 ns, and use bit D0 to output the square wave.

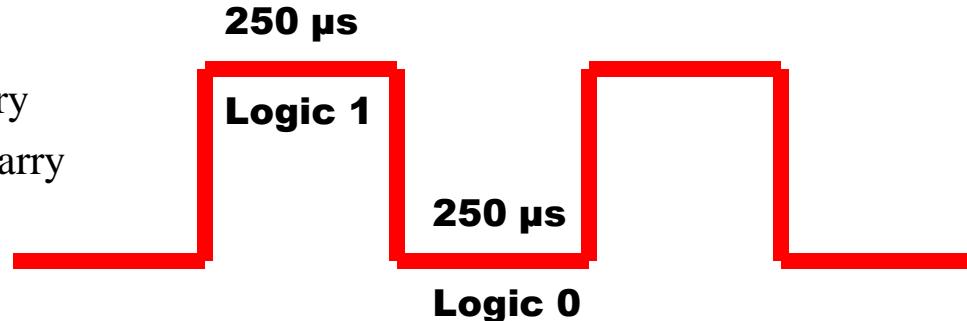
Note 1:

RAL: Rotate Accumulator Left Through Carry

RAR : Rotate Accumulator Right Through Carry

RLC : Rotate Accumulator Left

RRC: Rotate Accumulator Right



Note 2:

The period of the square wave is $500 \mu\text{s}$; therefore, the pulse should be on (logic 1) for $250 \mu\text{s}$ and off (logic 0) for the remaining $250 \mu\text{s}$. The alternate pattern of the 0/1 bits can be provided by loading the accumulator with the number AAH (**1010 1010**) and rotating the pattern once through each delay loop.

Label	Mnemonics	T-states	Comments
	MVI D,AAH	7T	; Load bit pattern AAH
ROTATE:	MOV A,D	4T	;Load bit pattern in A
	RLC	4T	; Change data from AAH to 55H and vice versa
	MOV D,A	4T	;Save (A)
	ANI 01H	7T	;Mask bits D7-D1
	OUT PORT1	10T	; Turn on or off the lights
	MVI B,COUNT	7T	; Load delay count for 250 µs
DELAY:	DCR B	4T	; Next count
	JNZ DELAY	10/7T	; Repeat until (B) = 0
	JMP ROTATE	10T	; Go back to change logic level

Delay Calculations In this problem, the pulse width is relatively small (250 μ s); therefore, to obtain a reasonably accurate output pulse width, we should account for all the T-states. The total delay should include the delay in the loop and the execution time of the instructions outside the loop.

1. The number of instructions outside the loop is seven; it includes six instructions before the loop beginning at the symbol ROTATE and the last instruction JMP.

$$\text{Delay outside the Loop: } T_O = 46 \text{ T-states} \times 325 \text{ ns} = 14.95 \mu\text{s}$$

2. The delay loop includes two instructions (DCR and JNZ) with 14 T-states except for the last cycle, which has 11 T-states.

$$\begin{aligned}\text{Loop Delay: } T_L &= 14 \text{ T-states} \times 325 \text{ ns} \times (\text{Count} - 1) + 11 \text{ T-states} \times 325 \text{ ns} \\ &= 4.5 \mu\text{s} (\text{Count} - 1) + 3.575 \mu\text{s}\end{aligned}$$

3. The total delay required is 250 μ s. Therefore, the count can be calculated as follows:

$$T_D = T_O + T_L$$

$$250 \mu\text{s} = 14.95 \mu\text{s} + 4.5 \mu\text{s} (\text{Count} - 1) + 3.575 \mu\text{s}$$

$$\text{Count} = 52.4_{10} = 34H$$



Problem No 16: Write an ALP using 8085 instruction, to generate a continuous square wave with the period of 250 μ s. Assume the clock period is 0.33 μ s and use bit D4 to output the square wave. Show the delay calculation also

Note1:

**Similar problem was done in class. See page 289 in text book,6e.
Use $T_D = 125 \mu\text{s}$ instead of $250 \mu\text{s}$ given in page 290 during your calculations**

Note2:

Also make sure to use bit D4 instead of bit D0 in your code. To do this, use the following code in instruction number 5 given in page 289

ANI 10H ; Mask all bits except D4

Note3:

Thus instead of solving this problem, we will see how to solve question number 17 given in page 294 of text book.



Rectangular Wave

**On-period
200 μ s**

**off-period
400 μ s**



Problem No 17: Write a program to generate a rectangular wave with a 200 μ s on-period and a 400 μ s off-period

TURNON: MVI A,01H (7T)

OUT PORT1 (10T)

MVI B, COUNT1 (7T)

;BIT PATTERN TO TURN ON D0

;ON-PERIOD BEGINS

;COUNT FOR 200 μ s DELAY

LOOP1: DCR B (4T)

JNZ LOOP1 (10T/7T)

MVI A,00H (7T)

OUT PORT1 (10T)

MVI B,COUNT2 (7T)

;BIT PATTERN TO TURN OFF D0

;OFF-PERIOD BEGINS

;COUNT FOR 400 μ s DELAY

LOOP2: DCR B (4T)

JNZ LOOP2 (10T/7T)

JMP TURNON (10T)

**;REPEAT BY DOING THE SAME THING
;AGAIN**

Delay Calculations

Assume clock speed = 325.5 ns, discard 7T from 10T/7T for simplicity

On-period delay:

$$T = T_0 + T_L$$

$$\begin{aligned}200 \mu s &= (7T+10T+7T) \times 325.5 \text{ ns} + (4T+10T) \times 325.5 \text{ ns} \times \text{COUNT1} \\&= 24 \times 325.5 \text{ ns} + 14 \times 325.5 \text{ ns} \times \text{COUNT1} \\&= 7.812 \mu s + 4.557 \mu s \times \text{COUNT1}\end{aligned}$$

$$\Rightarrow \text{COUNT1} = 42.17$$

$$\approx 42_{10}$$

Off-period delay:

$$T = T_0 + T_L$$

$$\begin{aligned}400 \mu s &= (10T+7T+10T+7T) \times 325.5 \text{ ns} + (4T+10T) \times 325.5 \text{ ns} \times \text{COUNT2} \\&= 34 \times 325.5 \text{ ns} + 14 \times 325.5 \text{ ns} \times \text{COUNT2} \\&= 11.067 \mu s + 4.5 \mu s \times \text{COUNT2}\end{aligned}$$

$$\Rightarrow \text{COUNT2} = 86.429$$

$$\approx 85_{10}$$

Problem No 18: A railway crossing signal has two flashing lights run by 8085 microprocessor. One light is connected to data bit D7 and the second light is connected to D6. Stating necessary assumptions you make, write an assembly language program to turn each signal light alternately on and off at an interval of 1 second

Note1:

$100 \times 10 \text{ msec} = 1 \text{ sec}$, so load 100 in B-Reg, find COUNT value to be stored in the D-Register. Since the delay is greater than 0.5 sec, we need to use loop under loop or need to call delay more than once. In this case two loops are used. The inner loop creates the delay for 10 msec. So approximately the 1 sec delay is created. Note that the adjustments are neglected in the delay calculations. Also assume clock speed is 325ns

Delay Calculations(should be shown after the program is written)

$$T_{\text{INNER}} = (6+4+4+10)\text{T-states} \times 325\text{ns} \times (\text{COUNT}-1) + (6+4+4+7)\text{T-States} \times 325\text{ns}$$

$$10 \text{ msec} = 24\text{T-States} \times 325\text{ns} \times (\text{COUNT}-1) + 21\text{T-States} \times 325\text{ns}$$

$$\text{COUNT} = 1282.17$$

$$= 1283_{10}$$

$$= 503_{16}$$



MVI L, AAH	;7T load the bit patter
ROTATE: MOV A,L	; 4T load bit pattern in A
RLC	; 4T, change bit pattern from AA to 55 ;&vice-versa
MOV L,A	; 4T, save A
ANI C0H	; 7T Mask bits D5-D0
OUT PORT1	; 10T turn on or off D7 and D6
MVI B,100	; 4T delay 1 is equal to 100
OUTER: LXI D,COUNT	;10T
INNER: DCX D	;6T
MOV A,E	; 4T
ORA D	; 4T should create 10 msec delay
JNZ INNER	;(10/7T)
DCR B	; 4T
JNZ OUTER	;(10/7T)
JMP ROTATE	;10T

(refer page 289 square wave example, question no 18 at page 294, and its solution given at page 793 in the text book,5e)

Chapter 9

Stack and Subroutines



Stack

- ✓ The stack is a group of memory locations in the R/W memory that is used for temporary storage of binary information during the execution of a program.
- ✓ The starting memory location of the stack is defined in the main program, and space is reserved, usually at the high end of the memory map



Subroutine

- ✓ A subroutine is a group of instructions that performs a subtask (e.g , time delay or arithmetic operation) of repeated occurrence.
- ✓ The subroutine is written as a separate unit, apart from the main program, and the microprocessor transfers the program execution from the main program to the subroutine whenever it is called to perform the task
- ✓ After the completion of the subroutine task, the microprocessor returns to the main memory.
- ✓ Before implementing the subroutine technique, the stack must be defined; the stack is used to store the memory address of the instruction in the main program that follows the subroutine call.



Defining Stack

- ✓ The beginning of the stack is defined in the program by using the instruction **LXI**.
- LXI SP , 16-bit
- ✓ It loads a 16-bit memory address in the stack pointer register of the microprocessor
- ✓ Once the stack location is defined, storing of data bytes begins at the memory address that is one less than the address in the stack pointer register
- ✓ For example, if the stack pointer register is loaded with the memory address 2099H (**LXI SP,2099H**), the storing of data bytes begin at **2098H** and continues in reverse numerical order (decreasing memory address such as 2098H, 2097H, etc)...
- ✓ Therefore, as a general practice, the stack is initialized at the highest available memory location to prevent the program from being destroyed by the stack information.



LXI SP, 2099H

SP =>

Memory Location	Contents
2093H	XX
2094H	XX
2095H	XX
2096H	XX
2097H	XX
2098H	XX
2099H	XX
209AH	XX
209BH	XX



Instructions

The programmer can store and retrieve the contents of a register pair by using PUSH and POP instructions. The instructions necessary for using the stack are:

LXI SP, 2999H ; load stack pointer

PUSH Rp ; Store Register Pair on Stack
; This is a 1-byte instruction
; There are four instructions

For example:

PUSH B ; copies the contents BC
; the SP is decremented
;contents of B-reg are copied in the location pointed by SP - 1
; SP is decremented
; contents of C-reg are copied in the location pointed by SP-2

PUSH D

PUSH H

PUSH PSW

(PSW = A+FLAGS)



POP Rp

- ; Retrieve Register Pair from Stack
- ; This is a 1-byte instruction
- ; There are four instructions

For example:

POP B

- ; It copies the contents of top two memory locations of the stack into the specified register pair
- ; The contents of the memory location pointed by SP are copied to C
- ; The SP is incremented by 1
- ; The contents of the memory location pointed by SP+1 are copied to B
- ; The SP is again incremented by 1

POP D

POP H

POP PSW

(PSW = A+FLAGS)

All three of these instructions belong to the data transfer (copy) group; thus, the contents of the source are not modified, and no flags are affected.



Points to Remember

- The contents of the stack pointer can be interpreted as the address of the memory location that is already used for storage. The retrieval of bytes begins at the address in the stack pointer; however, the storage begins at the next memory location (in the decreasing order).
- The storage and retrieval of data bytes on the stacks should follow the LIFO (Last-in-First-Out) sequence. There are four instructions



Subroutine

- ❑ A subroutine is a group of instructions written separately from the main program to perform a function that occurs repeatedly in the main program.

The 8085 has two instructions to implement subroutines:

- i. **CALL** (call a subroutine)
- ii. **RET** (return to main program from a subroutine)

CALL

- ✓ 3-byte instruction
- ✓ Transfers the program sequence to a subroutine address
- ✓ Saves the contents of the program counter (the address of the next instruction) on the stack
- ✓ Decrements the stack pointer register by 2
- ✓ Jumps unconditionally to the memory location specified by the 2nd and 3rd byte
- ✓ This instruction is accompanied by a return instruction in the subroutine

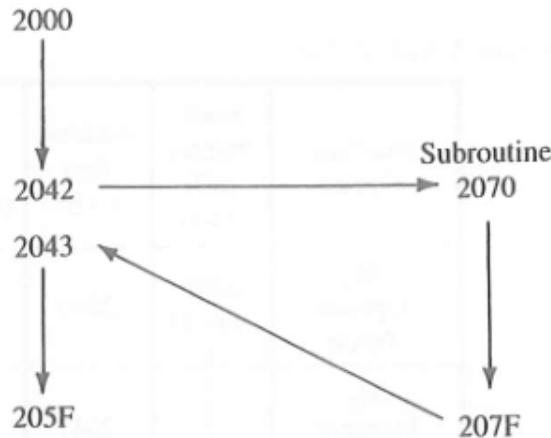
RET

- ✓ 1-byte instruction
- ✓ Inserts the two bytes from the top of the stack into the program counter and increments the stack pointer register by 2
- ✓ Unconditionally returns from a subroutine



FIGURE 9.9

Subroutine Call and Program Transfer



The program execution begins at 2000_{H} , continues until the end of $\text{CALL } 2042_{\text{H}}$, and transfers to the subroutine at 2070_{H} . At the end of the subroutine, after executing the RET instruction, it comes back to the main program at 2043_{H} and continues.

CALL EXECUTION

Memory Address	Machine Code	Mnemonics	Comments
2040	CD	CALL 2070H	;Call subroutine located at the memory
2041	70		; ; location 2070H
2042	20		
2043	NEXT	INSTRUCTION	

Restart, Conditional Call, and Return Instructions

Conditional Call

CC => call subroutine if carry flag is set

CNC => call subroutine if carry flag is reset

CZ

CNZ

CM

CP

CPE

CPO => call subroutine if parity flag is reset (P=0, Odd Parity)

Conditional Return

RC => return if carry flag is set

RNC

RZ

RNZ

RM

RP

RPE

RPO => return if parity flag is reset (P = 0, Odd Parity)



Restart Instruction (RST)

The 8085 instruction also has 8 Restart instructions.

- ✓ RST instructions are 1-byte call instructions that transfer the program execution to a specific location
- ✓ They are executed the same way as call instructions
- ✓ These instructions are generally used in conjunction with the interrupt process

RST 0 => Call 0000H

RST 1 => Call 0008H

RST 2 => Call 0010H

RST 3 => Call 0018H

RST 4 => Call 0020H

RST 5 => Call 0028H

RST 6 => Call 0030H

RST 7 => Call 0038H



Advanced Instructions

XTHL

- ✓ Exchange top of the stack with H and L
- ✓ 1-byte
- ✓ The contents of L are exchanged with the contents of the memory location shown by the stack pointer
- ✓ And the contents of H are exchanged with the contents of memory location of the SP+1

SPHL

- ✓ Copy H and L registers into the SP register
- ✓ 1-byte
- ✓ The contents of H specify the higher-order byte and the contents of L specify the low-order byte
- ✓ The contents of HL registers are not affected

$SP \leftarrow [HL]$

PCHL

- ✓ Copy H and L registers into the PC
- ✓ 1-byte instruction
- ✓ The contents of H specify the high-order byte and the contents of L specify the low-order byte

$PC \leftarrow [HL]$



Advanced Instructions

ADC R

ADC M

ACI 8-bit

These instructions add the contents of the operand, the carry, and the accumulator; All flags are affected

SBB R

SBB M

SBI 8-bit

These instructions subtract the contents of the operand and the borrow from the contents of the accumulator

DAD B

DAD D

DAD H

DAD SP

- ✓ Add register pair to register HL
- ✓ This is a 1-byte instruction
- ✓ Adds the contents of the operand (register pair or stack pointer) to the contents of HL registers
- ✓ The result is placed in HL registers
- ✓ The Carry flag is altered to reflect the result of the addition. No other flags are affected
- ✓ The instruction set includes four instructions.



Advanced Instructions

LHLD

- ✓ Load HL registers direct
- ✓ 3-byte instruction
- ✓ No flags are affected

Eg LHLD 2050H

- ✓ Content of 2050H => L
- ✓ Content of 2051H => H

SHLD

- ✓ Store HL registers direct
- ✓ 3-byte instruction
- ✓ No flags are affected

Eg SHLD 2050H

- ✓ Content of L => 2050H
- ✓ Content of H => 2051H

XCHG

- ✓ Exchange the contents of HL and DE
- ✓ 1-byte instruction
- ✓ No flags are affected
- ✓ Content of H ⇔ Content of D
- ✓ Content of L ⇔ Content of E



**Example
10.4**

Registers BC contain 2793H, and registers DE contain 3182H. Write instructions to add these two 16-bit numbers, and place the sum in memory locations 2050H and 2051H.

Before instructions:

B	27	93	C
D	31	82	E

Instructions

MOV A,C
 ADD E
 MOV L,A
 MOV A,B
 ADC D
 MOV H,A
 SHLD 2050H

A	93	F	93H
A	15	CY = 1	+ 82H
H		15	1/15H
			27H
			+ 31H
H	59	15	59H

Write instructions to display the contents of the stack pointer register at output ports.

Example
10.6

Instructions

LXI H,0000H	;Clear HL
DAD SP	;Place the stack pointer contents in HL
MOV A,H	;Place high-order address of the stack pointer in the accumulator
OUT PORT1	
MOV A,L	;Place low-order address of the stack pointer in the accumulator
OUT PORT2	

The instruction DAD SP adds the contents of the stack pointer register to the HL register pair, which is already cleared. This is the only instruction in the 8085 that enables the programmer to examine the contents of the stack pointer register.

Examples

Example 1: Write a program to clear a flag register and an accumulator

```
LXI SP, 4200H      ; Initialize stack  
LXI B,0000H      ; Clear BC register pair  
PUSH B           ; push the content of BC pair to stack  
POP PSW          ; Pop the top of stack to A-reg and flag register (PSW)  
HLT
```

Example 2: Write a program to exchange the contents of BC register and DE pair.

```
LXI SP, 4200H      ; initialize the stack  
PUSH B           ; store the contents of BC pair in stack  
PUSH D           ; store the contents of DE pair in stack  
POP B            ; move the contents of DE pair stored in stack to BC pair  
POP D            ; move the contents of BC pair stored in stack to DE pair  
HLT
```



Modify flag contents using PUSH/POP

Example 3: Write a program to reset the zero flag.

```
LXI SP,FFFFH      ; initialize the stack
PUSH PSW          ; first A is pushed, then flags are pushed
POP H              ; flags will be popped at L, A will be popped at H
MOV A,L
ANI BFH            ; Masking bit = 1011 1111 => BFH
MOV L,A
PUSH H              ; now H is pushed, L is pushed
POP PSW            ; at the end, flags will contain L's content (modified one), then H
                   ; content will be in A
HLT
```



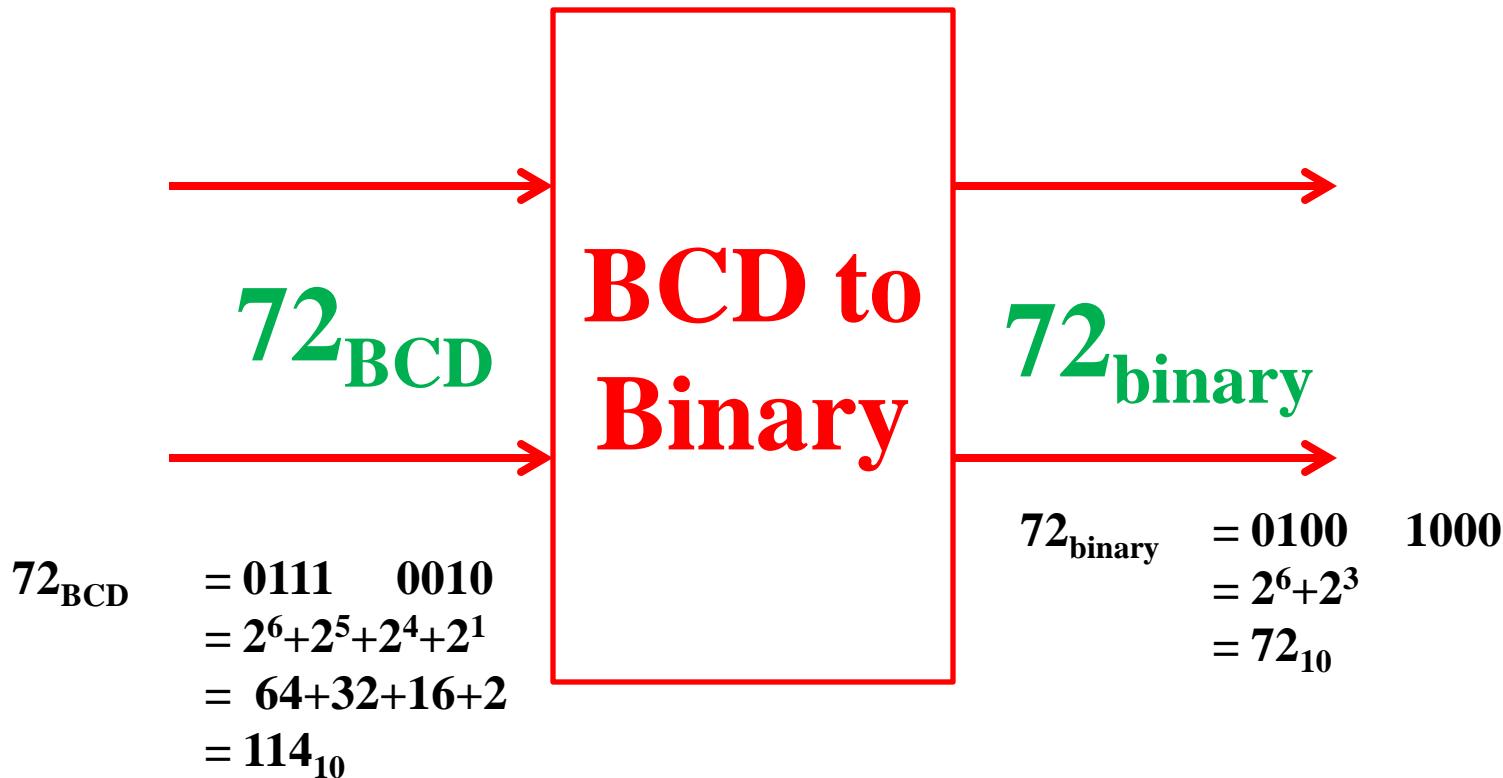
Chapter 10



Code Conversion, BCD Arithmetic, and 16-Bit Data Operations



BCD to 8-bit Binary Conversion



Program No 1: Write an assembly language program for 8085 to convert the BCD number at memory location 1200H into binary and store the result at memory location 1201H

$$72_{10} = 7 \times 10 + 2$$

$$\text{BCD}_2 = 7$$

$$\text{BCD}_1 = 2$$

Thus Binary = $10 \times \text{BCD}_2 + \text{BCD}_1$

Hence Load the number first in A, and extract BCD1 and BCD2 in C and D register first. Then add BCD2 ten times and add with BCD1 to get the binary value.

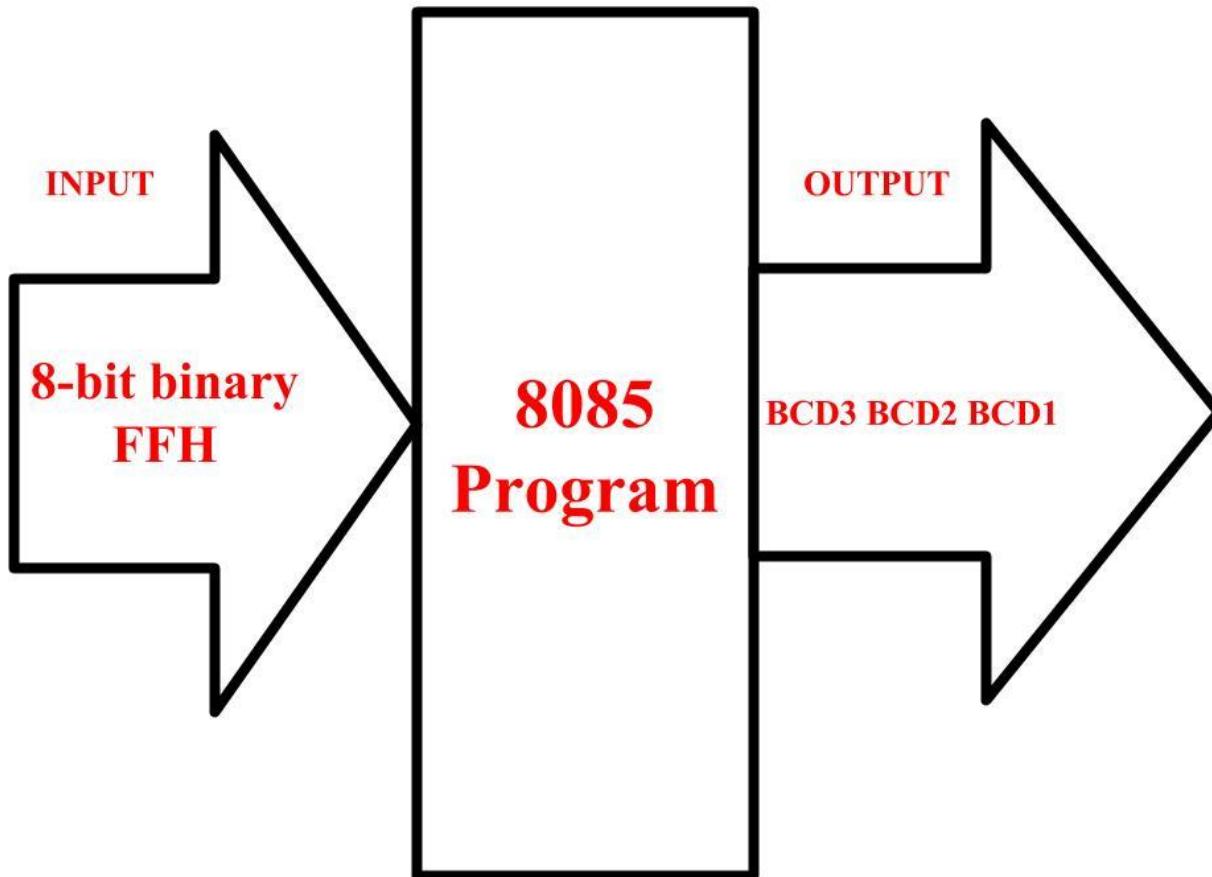


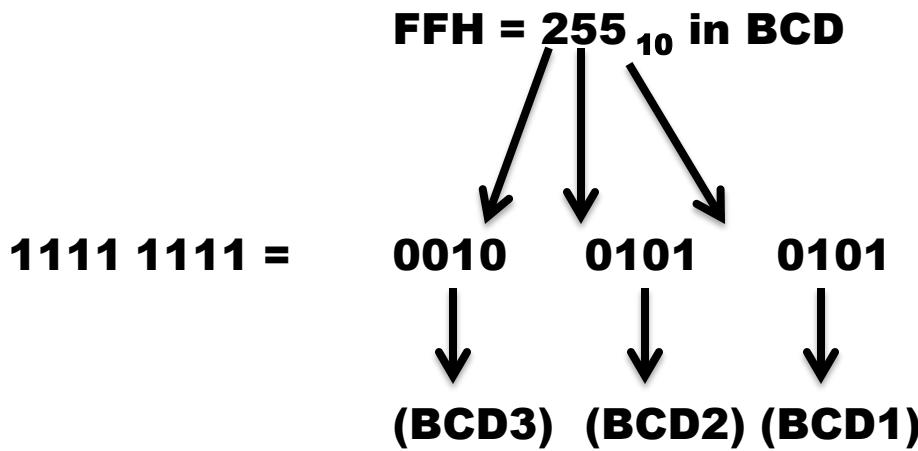
LDA 1200H	;load the number
MOV B,A	;save the number in B
ANI F0H	; mask lower nibble
RRC	
RRC	
RRC	
RRC	
MOV D,A	;save BCD2 in D-Reg
MOV A,B	;get the number again
ANI 0FH	;mask the upper nibble
MOV C,A	;save BCD1 in C-Reg
XRA A	;clear accumulator
MVI E,0AH	;set E as multiplier of 10
MULTIPLY: ADD E	;add 10 until (D)=0
DCR D	
JNZ MULTIPLY	;is multiplication complete?
ADD C	;If not go back and add again
STA 1201H	
HLT	

(Please see page 324 for reference in text book)



8-bit Binary to BCD conversion





Note1:

So we need 12 bits to convert an 8-bit binary number into a BCD equivalent. BCD2 and BCD1 can be stored in a single 8-bit memory location. BCD3 has to be stored in the next memory location. So assume that BCD2 and BCD1 will be stored at 2600H. Store the remaining BCD3 at location 2601H.

Note2:

If you look at page number 327 in the text book, you have a similar solution. Goankar saves all three digits BCD3, BCD2 , and BCD1 in different memory locations. Please use whichever method you find it easy to perform



Step1: Divide by 100

100) 255(2 \longleftarrow BCD3 (Quotient)

-100

55

255 Quotient

-100= 155 1

-100=55 1

1
1

} use E
as a counter

2 (BCD3)

Step2: Divide by 10

10) 55 (5 \longleftarrow BCD2 (Quotient))

-50

5 \longleftarrow BCD1 is the remainder

55 Quotient

-10=>45 1

-10=>35 1

-10=>25 1

-10=>15 1

-10=>05 1

} use D
as a counter

5(BCD2)

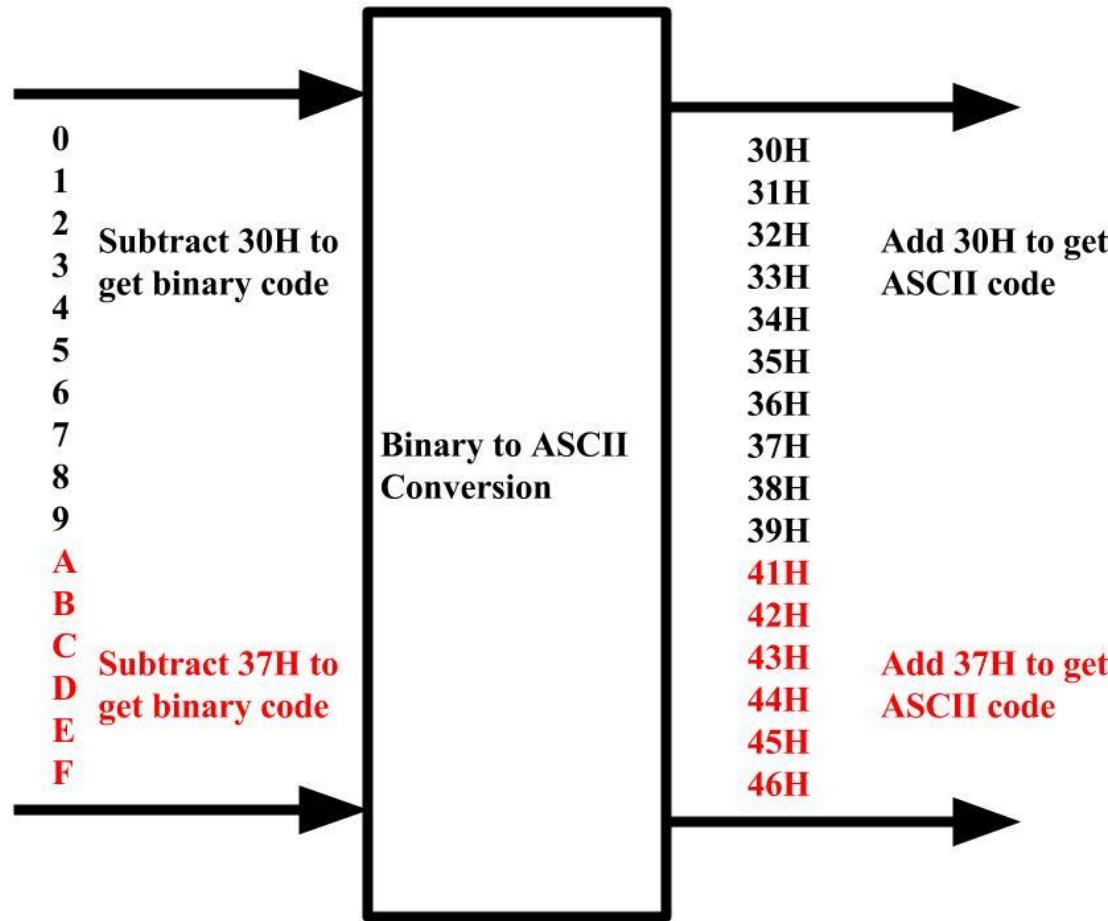
Step3: Remainder is the BCD1
i.e BCD1 = 05 in this case

Program No 2: Write an 8085 assembly language program to convert an 8-bit binary number into its equivalent BCD form

	MVI E,00H	;CLEAR E-REG FOR HUNDRED
	MOV D,E	;CLEAR D-REG ALSO
	LDA 2500H	;GET THE BINARY DATA IN A-REG
HUND:	CPI 100	;IF DATA IS LESS THAN 100,JUMP TO TEN
	JC TEN	
	SUI 100	;SUBTRACT 100 FROM DATA
	INR E	; FOR EACH SUBTRACTION, INCREMENT E
	JMP HUND	
TEN:	CPI 10	;COMPARE WITH 10
	JC UNIT	
	SUI 10	
	INR D	
	JMP TEN	
UNIT:	MOV C,A	;SAVE THE UNIT IN c-REG(BCD1)
	MOV A,D	;GET BCD2 IN A-REG
	RLC	;ROTATE BCD2 DIGIT TO UPPER NIBBLE, MAKE 05D TO 50D
	RLC	
	RLC	
	RLC	
	ADD C	;COMBINE BCD1 AND BCD2
	STA 2600H	;SAVE BCD2 BCD1 IN MEMORY
	MOV A,E	
	STA 2601H	;SAVE BCD3 IN MEMORY
	HLT	



Binary to ASCII Conversion



Program No 3: Write a program to convert ASCII into its equivalent Binary

Note1:

Assume that the number is stored at 2500H. Store the result at 2501H

LDA 2500H	;GET THE NUMBER
SUI 30H	;SUBTRACT 30H FROM THE READ DATA
CPI 10D	
JC LETTERS	;IF NUMBER IS ABOVE 9, NEED TO SUBTRACT ;37H,IE EXTRA 07H
SUI 07H	;SUBTRACT 07H
LETTERS: STA 2501H	;STORE ASCII AT MEMORY
HLT	



Program No 4: Write a program to convert binary into its equivalent ASCII

Note1:

Assume that the number is stored at 2000H. Store the result at 2001H

LDA 2000H	;GET THE NUMBER
CPI 0AH	;IS DIGIT LESS THAN 10
JC CODE	;IF IT IS , GO TO CODE AND ADD 30H ONLY
ADI 07H	;ADD 7H TO OBTAIN
CODE: ADI 30H	;CODE FOR DIGIT FROM A TO F
STA 2001H	;ADD BASE NUMBER 30H
HLT	;STORE THE ASCII CODE IN 2001H



DAA (Decimal Adjust Accumulator)

- **This is a 1-byte instruction**
- **It adjusts an 8-bit number in the accumulator to form two BCD numbers**
- **It uses the AC and the CY flags to perform the adjustment**
- **All flags are affected**

Note:

- **It adjusts a BCD sum**
- **Does not convert a binary number into BCD numbers**
- **Works only with addition when BCD numbers are used; does not work with subtraction**



Add two packed BCD numbers: 77 and 48.

Example 10.2

Addition:

$$\begin{array}{r}
 77 = 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1 \\
 + 48 = 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\
 \hline
 125 = 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1
 \end{array}$$

CY [1] $\frac{+ 0\ 1\ 1\ 0}{0\ 1\ 0\ 1}$

Solution

The value of the least significant four bits is larger than 9. Add 6.

The value of the most significant four bits is larger than 9. Add 6 and the carry from the previous adjustment.

$$\text{CY } \boxed{1} \quad \begin{array}{r} + 0 1 1 0 \\ \hline 0 0 1 0 \end{array} \quad 0 1 0 1$$

In this example, the carry is generated after the adjustment of the least significant four bits for the BCD digit and is again added to the adjustment of the most significant four bits.



Next Week

Unit 3: Bus structure, Memory and I/O Interfacing



Thank You

29 October 2017

Pramod Ghimire

Slide 191 of 601



Timing Diagram



Instruction Cycle

- ✓ The sequence of operations that a μ P has to carry out while executing the instruction is called instruction cycle
- ✓ Each instruction cycle of a μ P in turn consists of a number of machine cycle (MC)
- ✓ In 8085, instruction cycle may consists of 1 to 6 machine cycle

Machine Cycle

- ✓ The Machine Cycles (MC) are the basic operations performed by the μ P. To execute an instruction, the processor executes one or more machine cycles in a particular sequence
- ✓ The 8085 has seven basic machine cycle
 1. Opcode Fetch Cycle (4T or 6T)
 2. Memory Read Cycle (3T)
 3. Memory Write Cycle (3T)
 4. I/O Read Cycle (3T)
 5. I/O Write Cycle (3T)
 6. Interrupt Acknowledge Cycle (6T or 12T)
 7. Bus Idle Cycle (2T or 3T)



T-State

- ✓ The time required by the µP to execute a machine cycle is expresses in T-States. One T-State is equal to the time period of the internal clock signal of the µP. The T-State starts at the falling edge of a clock

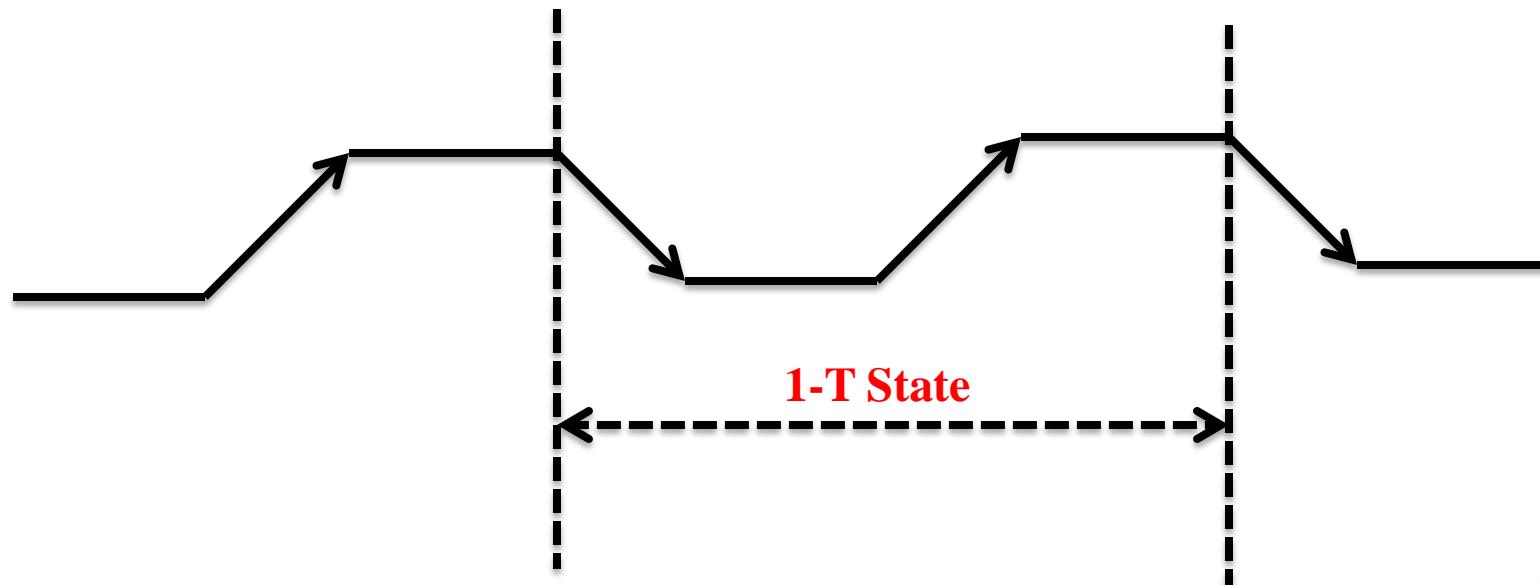


Figure 1: Clock Signal

Timing Diagram

- ✓ The timing diagram provides information about the various condition (high state, low state, high-Z state) of signals while a machine cycle is executed. The timing diagrams are supplied by the manufacturer of the µP.



Number of Bytes Vs Number of Machine Cycles

MOV A,C	=> 1 Byte instruction => 1 Machine Cycle => Opcode Fetch Only
MVI A,32H	=> 2 Byte instruction => 2 Machine Cycle => Opcode Fetch + Memory Read
STA 8000H	=> 3 Byte instruction => 4 Machine Cycle => Opcode Fetch + Memory Read + Memory Write
MOV A,M	=> 1 Byte instruction => 2 Machine Cycle => Opcode Fetch + Memory Read
IN 84H	=> 2 Byte instruction => 3 Machine Cycle => Opcode Fetch + Memory Read + I/O Read

Conclusion: There is no direct relationship between the number of bytes in an instruction and the number of machine cycles required to execute that instruction.



Opcode fetch (MOV C,A)

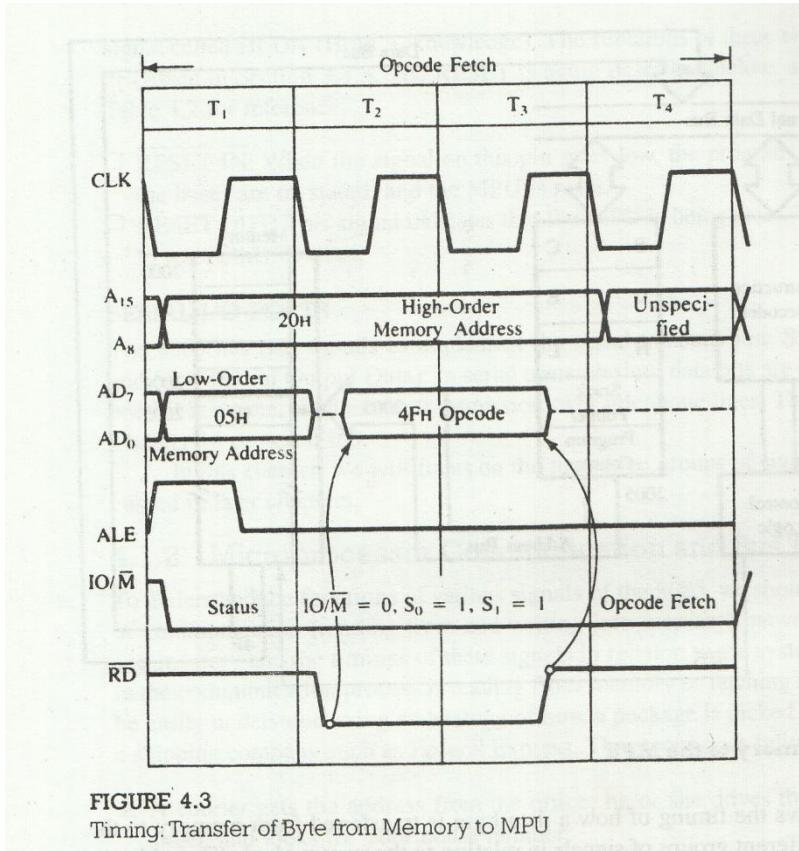


FIGURE 4.3

Timing: Transfer of Byte from Memory to MPU

- High-order memory address 20H placed on A₁₅-A₈
- Low-order address 05H placed on AD₇-AD₀
- ALE goes high
- IO/M bar goes low
- RD goes low between T₂ and T₃
- Opcode is placed on data bus
- 4FH is placed on instruction decoder. A-reg content is placed on C-reg content during T₄

TABLE 4.1

8085 Machine Cycle Status and Control Signals

Machine Cycle	Status			Control Signals
	IO/M	S₁	S₀	
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	INTA = 0
Halt	Z	0	0	$\overline{RD}, \overline{WR} = Z$ and $\overline{INTA} = 1$
Hold	Z	X	X	
Reset	Z	X	X	

NOTE: Z = Tri-state (high impedance)

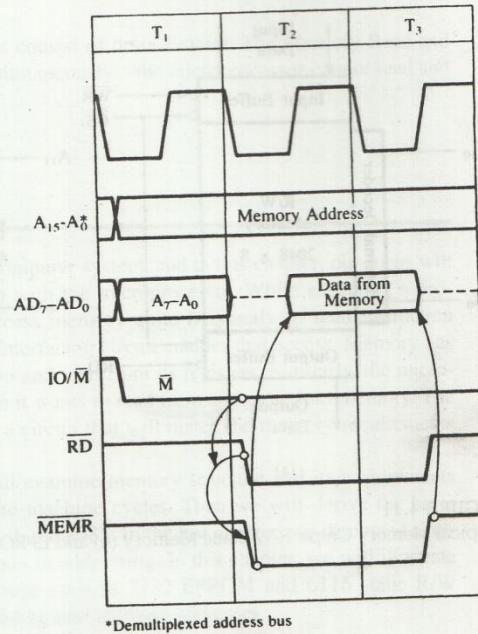
X = Unspecified

Memory Read

118

MICROPROCESSOR-BASED SYSTEMS: HARDWARE AND INTERFACING

FIGURE 4.12
Timing of the Memory Read Cycle

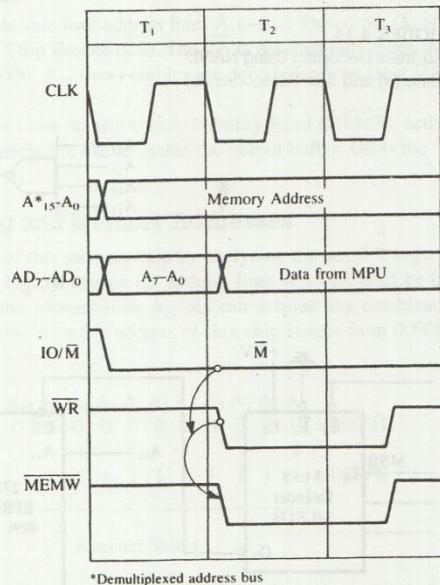


- ALE high during T₁ for a short time
- Data from memory placed while MEMR is held low

Memory Write

8085 MICROPROCESSOR ARCHITECTURE AND MEMORY INTERFACING

FIGURE 4.13
Timing of the Memory Write Cycle



119

- ALE high during T1 for a short time
- Data from MPU placed while MEMW is held low
- Data not valid when MEMW line goes high

Memory Write Correction

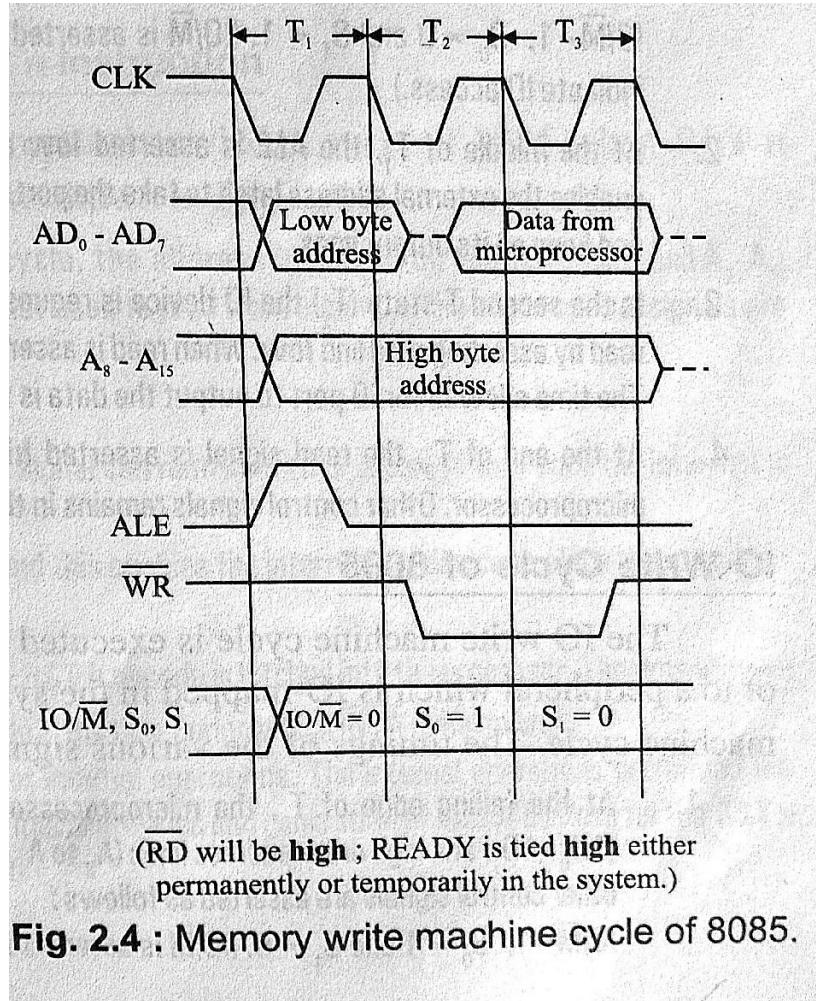


Fig. 2.4 : Memory write machine cycle of 8085.

Reference:
A Nagoor Kani,
“8085
Microprocessors & Its
Application”

Timing Diagram of STA 8000H

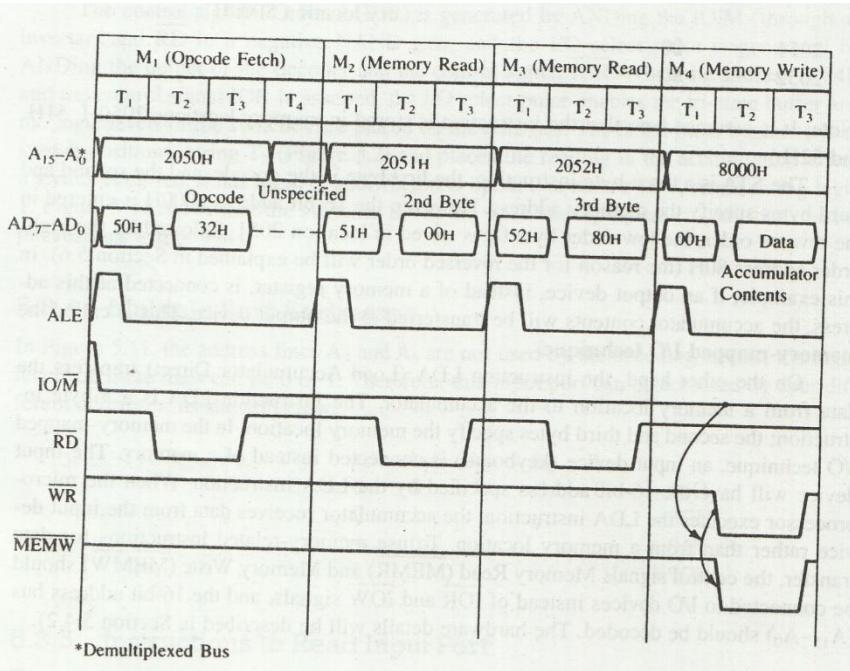


FIGURE 5.12

Timing for Execution of the Instruction: STA 8000H

- ALE high during T1 of three different Machine Cycles
- RD low during T2 and T3 of first three MC
- MEMW low during T2 and T3 of last MC

Look Into

- ❑ OUT 01H
- ❑ IN 84H
- ❑ MVI A,32H



Timing Diagram of OUT 01H

- ❑ In M1, higher and lower order address placed on data bus
- ❑ ALE goes high for 1 clock cycle
- ❑ Opcode placed in T2 to T3 when read line is low
- ❑ T4 of M1, opcode is decoded
- ❑ Thus in M2, next byte (i.e port address) is read from the memory
- ❑ In M3, data from MP is written to I/O device
- ❑ Port address is available from higher as well as lower order address of the address lines

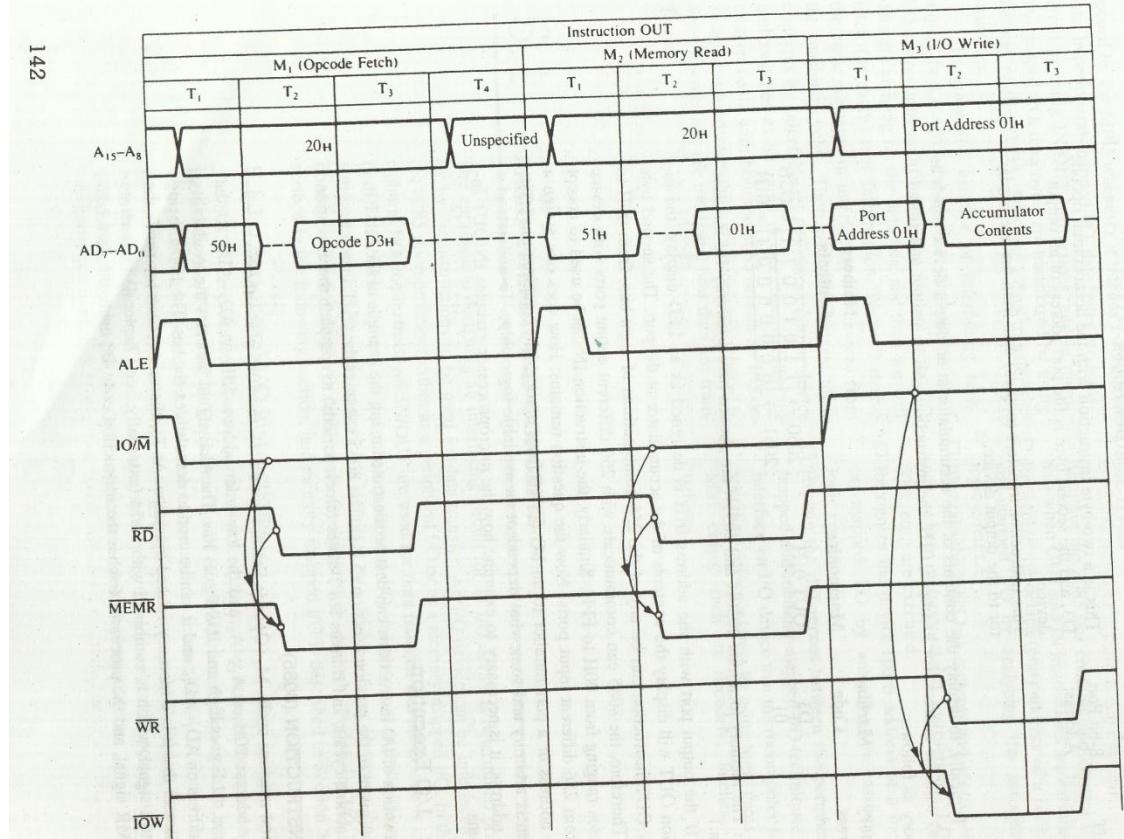


FIGURE 5.1
8085 Timing for Execution of OUT Instruction

Timing Diagram of IN 84H

- ❑ In M1, higher and lower order address placed on data bus
- ❑ ALE goes high for 1 clock cycle
- ❑ Opcode placed in T2 to T3 when read line is low
- ❑ T4 of M1, opcode is decoded
- ❑ Thus in M2, next byte (i.e port address) is read from the memory
- ❑ In M3, data from input port is read to MP
- ❑ Port address is available from higher as well as lower order address of the address lines

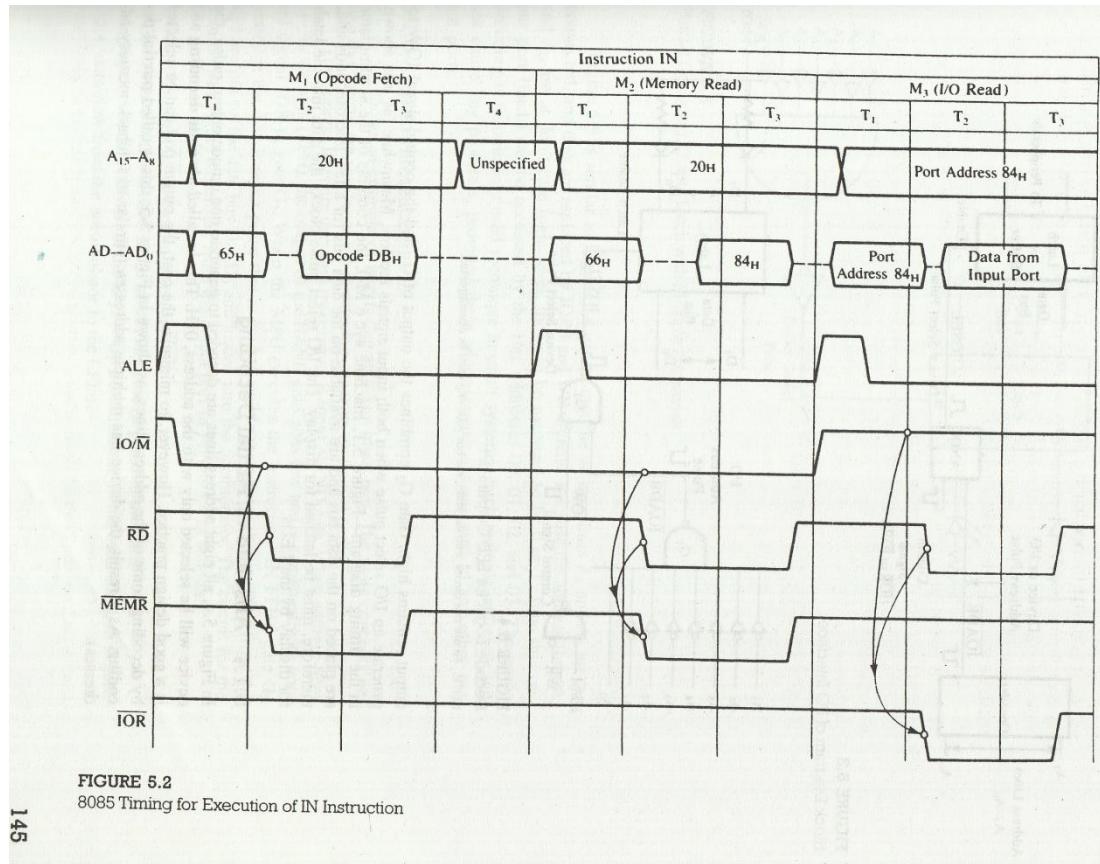


FIGURE 5.2
8085 Timing for Execution of IN Instruction

145

Timing Diagram of MVI A,32H

- ❑ In M1, higher and lower order address placed on data bus
- ❑ ALE goes high for 1 clock cycle
- ❑ Opcode placed in T2 to T3 when read line is low
- ❑ T4 of M1, Opcode is decoded
- ❑ Thus in M2, next byte (i.e port address) is read from the memory
- ❑ See Figure 4.10 in Text Book

8085 MICROPROCESSOR ARCHITECTURE AND MEMORY INTERFACING

113

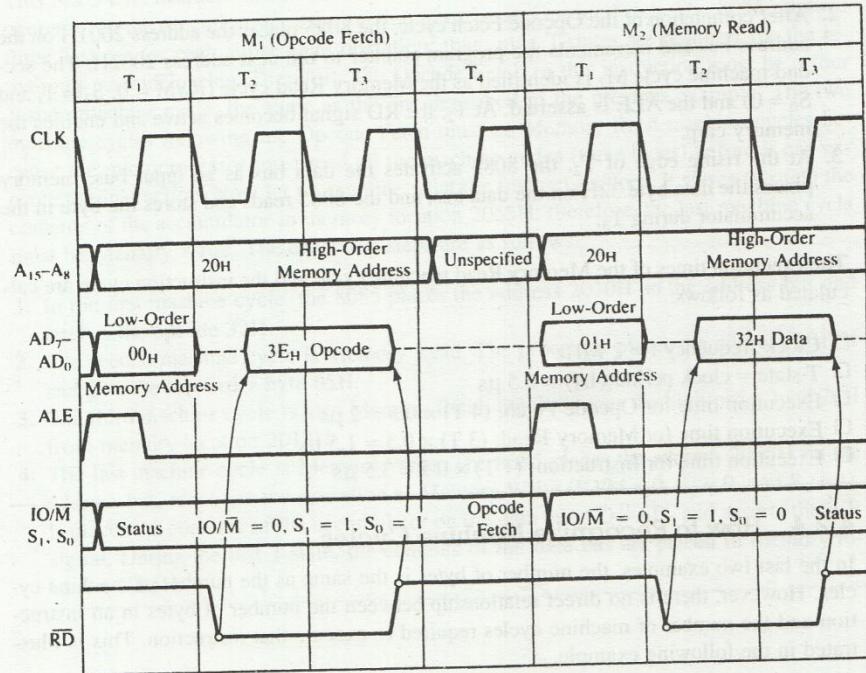


FIGURE 4.10
8085 Timing for Execution of the Instruction MVI A,32H

NEXT WEEK

MEMORY INTERFACING



THE END



Memory Interfacing



Memory Classification

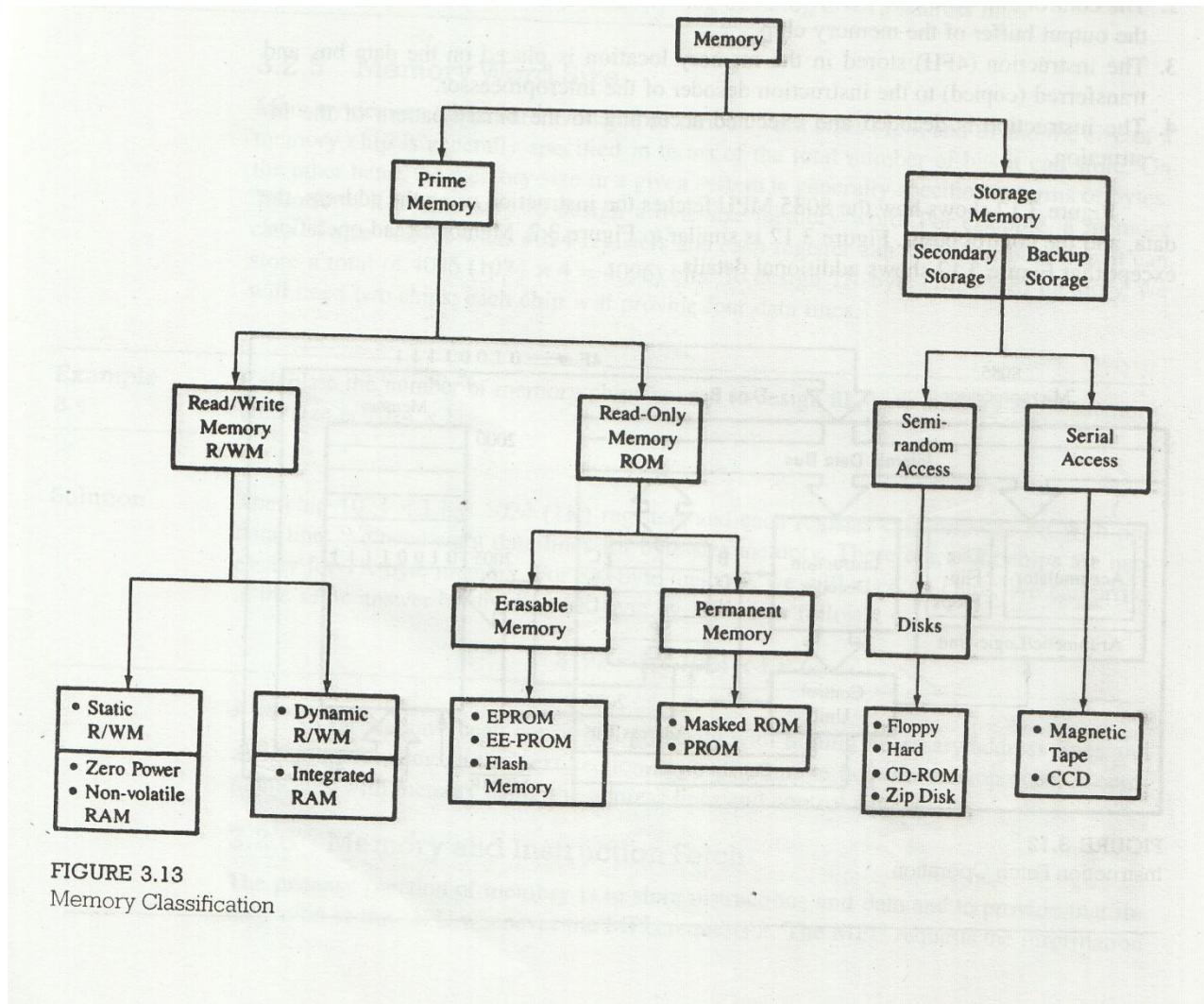


FIGURE 3.13
Memory Classification

SRAM Cell

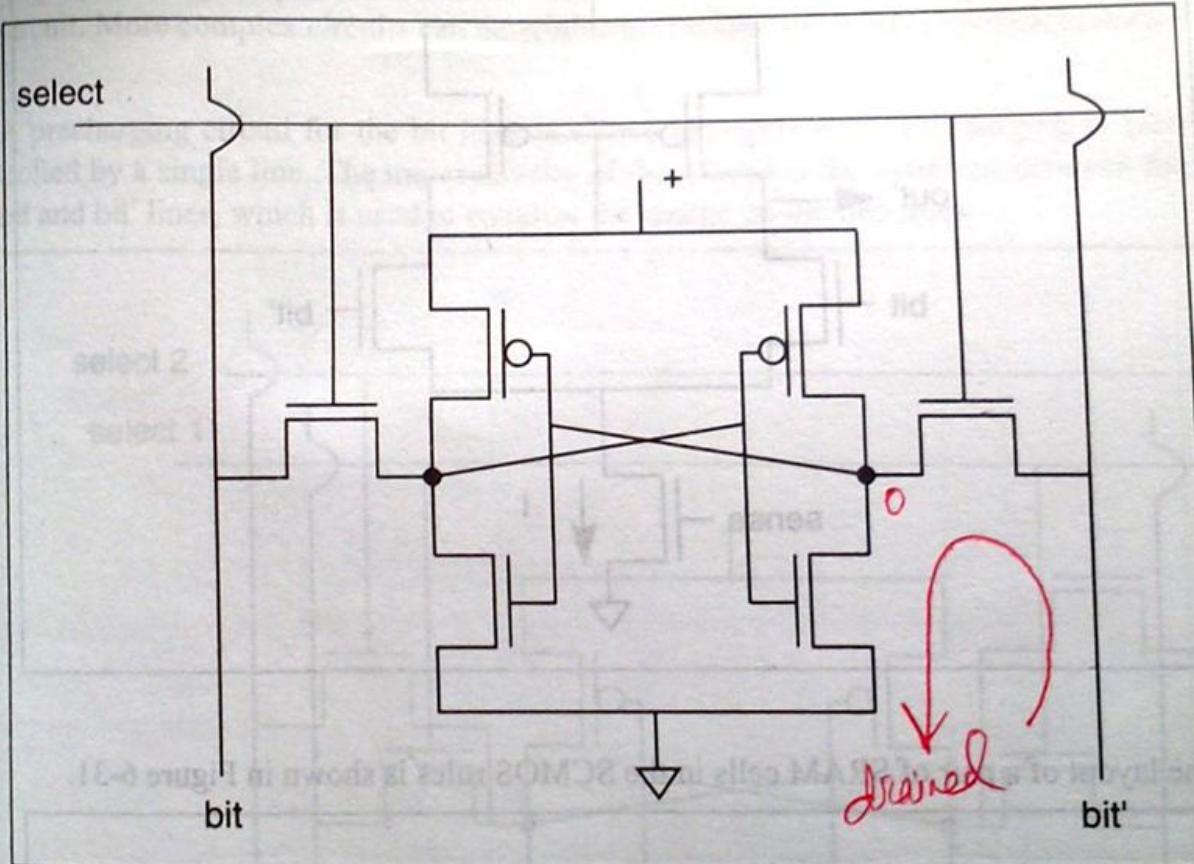


Figure 6-28:
Design of an
SRAM core cell.

Reference:
“ Modern
VLSI Design”,
Wayne Wolf
3e, page 243

Read Operation

- The value is stored in the middle four transistors, which form a pair of inverters connected in a loop.
- **To read**, bit and bit' are precharged to VDD before the select line is allowed to go high. One of the cell's inverters will have its output at 1, and the other at 0.
- If, for example, the right-hand inverter's output is 0, the bit' line will be drained to VSS through that inverter's pull down and the bit will remain high. If the opposite value is stored in the cell, the bit line will be pulled low while bit' remains high



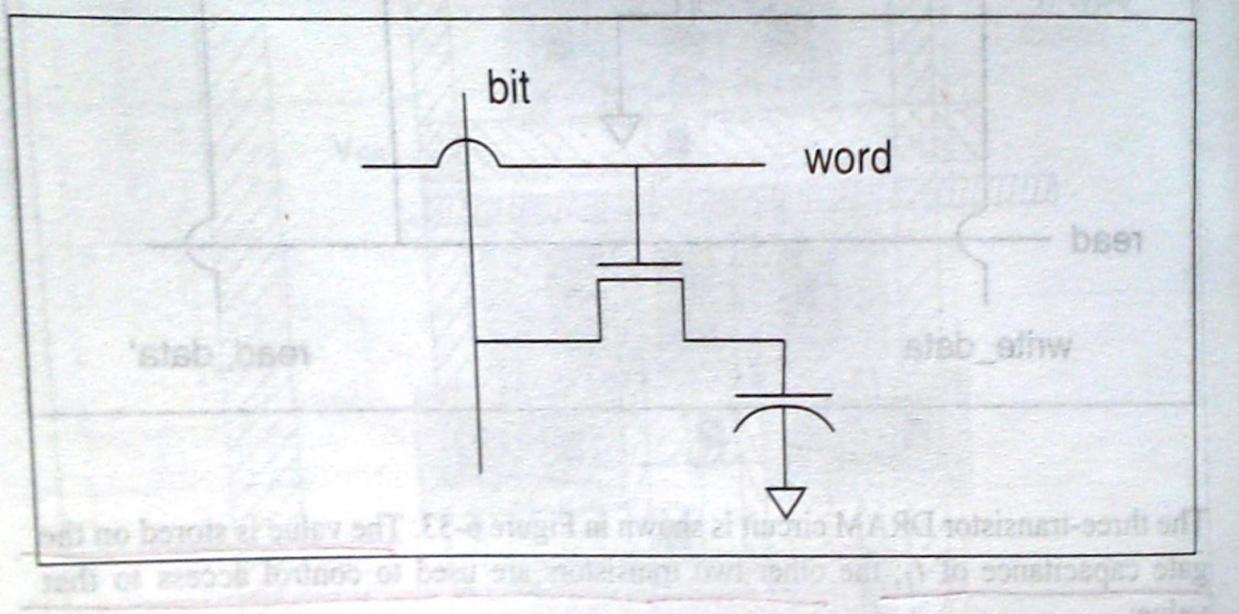
Write Operation

- **To write**, bit and bit' lines are set to the desired values, then select is set to 1. Charge sharing forces the inverters to switch values, if necessary, to store the desired value.
- The bit lines have much higher capacitance than the inverters, so the charge on the bit lines is enough to overwhelm the inverter pair and cause it to flip state.



DRAM Cell

Figure 6-34:
Circuit dia-
gram for a one-
transistor
DRAM core
cell.



Reference:
“ Modern
VLSI Design”,
Wayne Wolf
3e, page 248

Write Operation

- One-transistor/one-capacitor DRAM
- The cell has two external connections: a bit line and the word line
- The value is stored on a capacitor guarded by a single transistor
- Setting the word line high connects the capacitor to the bit line
- **To write** a new value, the bit line is set accordingly and the capacitor is forced to the proper value



Read Operation

- **When reading** the value, the bit line is first precharged before the word line is activated.
- If the storage capacitor is discharged, then charge will flow from the bit line to the capacitor, lowering the voltage on the bit line
- This read is destructive-the zero on the capacitor has been replaced by a one during reading
- As a result, additional circuitry must be placed on the bit lines to pull the bit line and storage capacitor to zero when a low voltage is detected on the bit line.



SRAM Vs DRAM

SRAM

- Uses 6-transistors
- Value stored as voltage
- High power consumption
- Expensive
- Faster
- Fabrication density low
- Additional refreshing circuitry not needed

DRAM

- DRAM uses only 1-transistor
- Value stored as charge
- Low power consumption
- Cheaper
- Slower
- Higher fabrication density
- Additional refreshing circuitry needed



R/W Memory Model

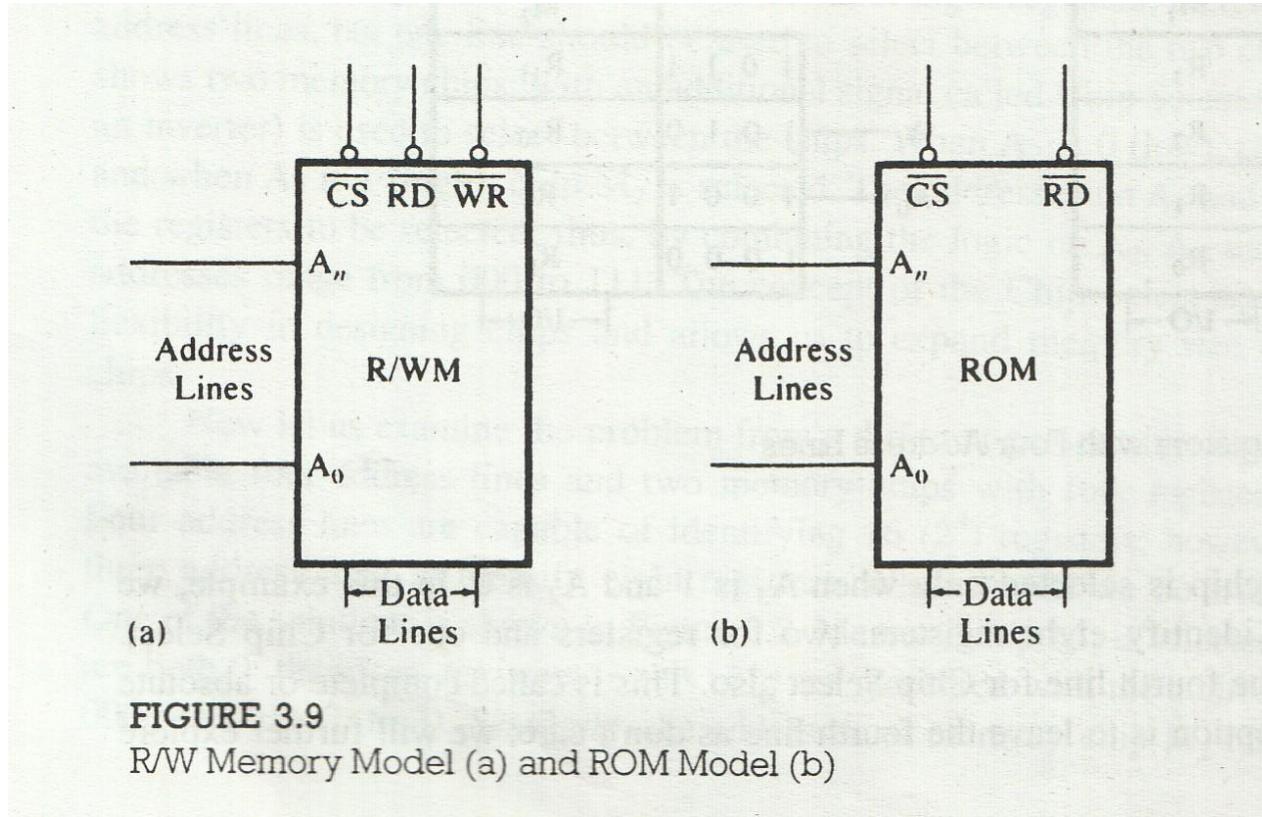


FIGURE 3.9

R/W Memory Model (a) and ROM Model (b)

Memory Map and Addresses

□ 8085=>

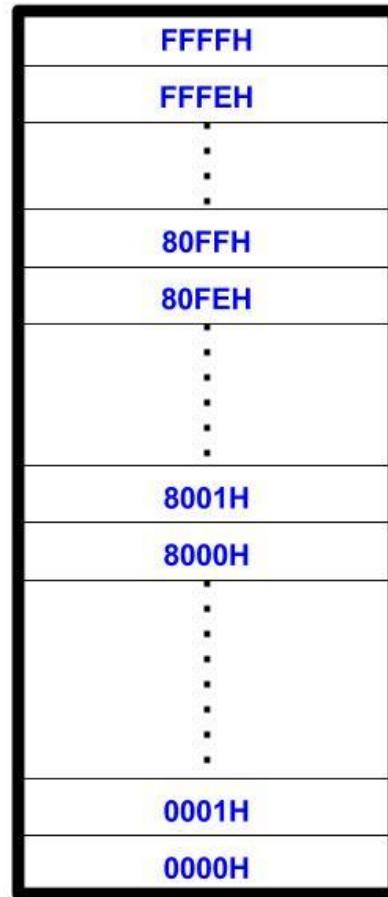
- ✓ 8-bit processor
- ✓ 16-bit address lines
- ✓ Can identify $2^{16} = 65,536$ memory registers each with 16-bit address
- ✓ The entire memory addresses can range from 0000H to FFFFH

□ Memory Map

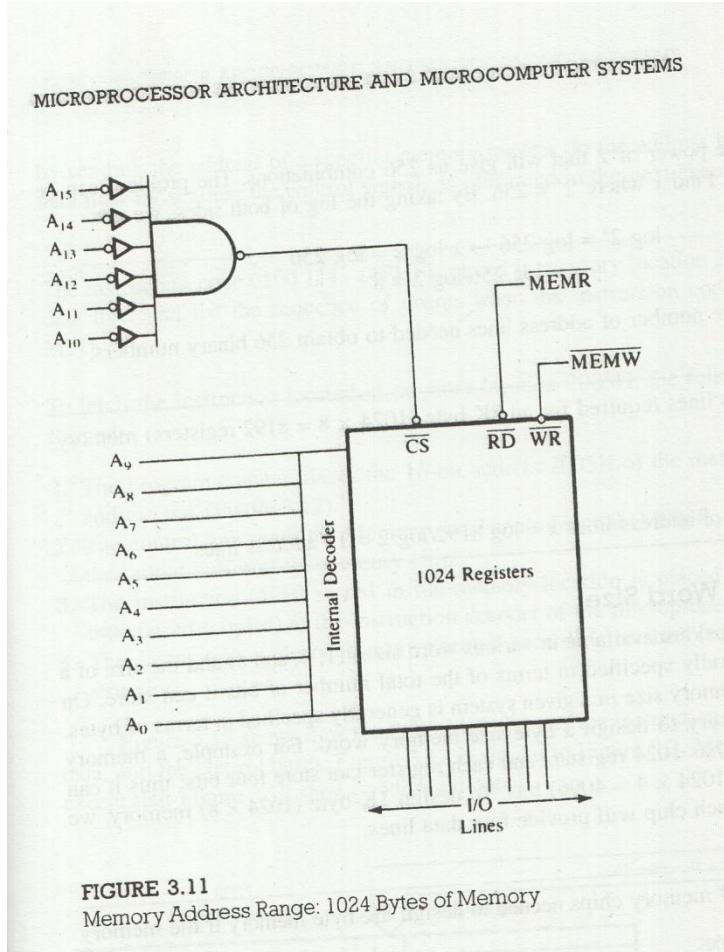
A memory map is a pictorial representation in which memory devices are located in the entire range of addresses.



Memory Map

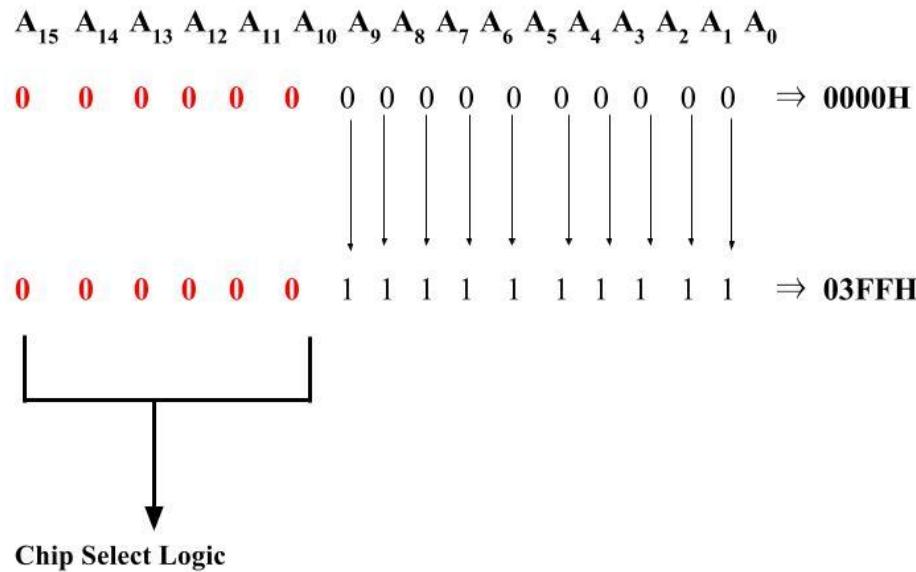


Memory Address Range of a 1K Memory Chip



- 1024 registers
- $A_9 - A_0$ required
- $A_{15} - A_{10}$ used for the chip select (\overline{CS}) signal
- $A_{15} - A_{10}$ should be at logic 0 to enable memory chip
- $A_9 - A_0$ can assume any values from all 0's to all 1's

Address Range Derivation



The memory addresses range from 0000H to 03FFH

The memory addresses can be changed to any other location by changing the hardware of chip select line. For example, if A_{15} is connected to the NAND gate without an inverter, the memory address will range from 8000H to 83FFH



Example 1

Design a circuit to interface an **8KB ROM**, a **4KB RAM**, and a **2 KB EPROM** with a microprocessor. [2010, 8marks]

Solution:

8KB ROM

$8K \times 8\text{bit}$

$\Rightarrow 2^{13} \times 8 \text{ bit}$

4KB RAM

$4K \times 8\text{bit}$

$\Rightarrow 2^{12} \times 8 \text{ bit}$

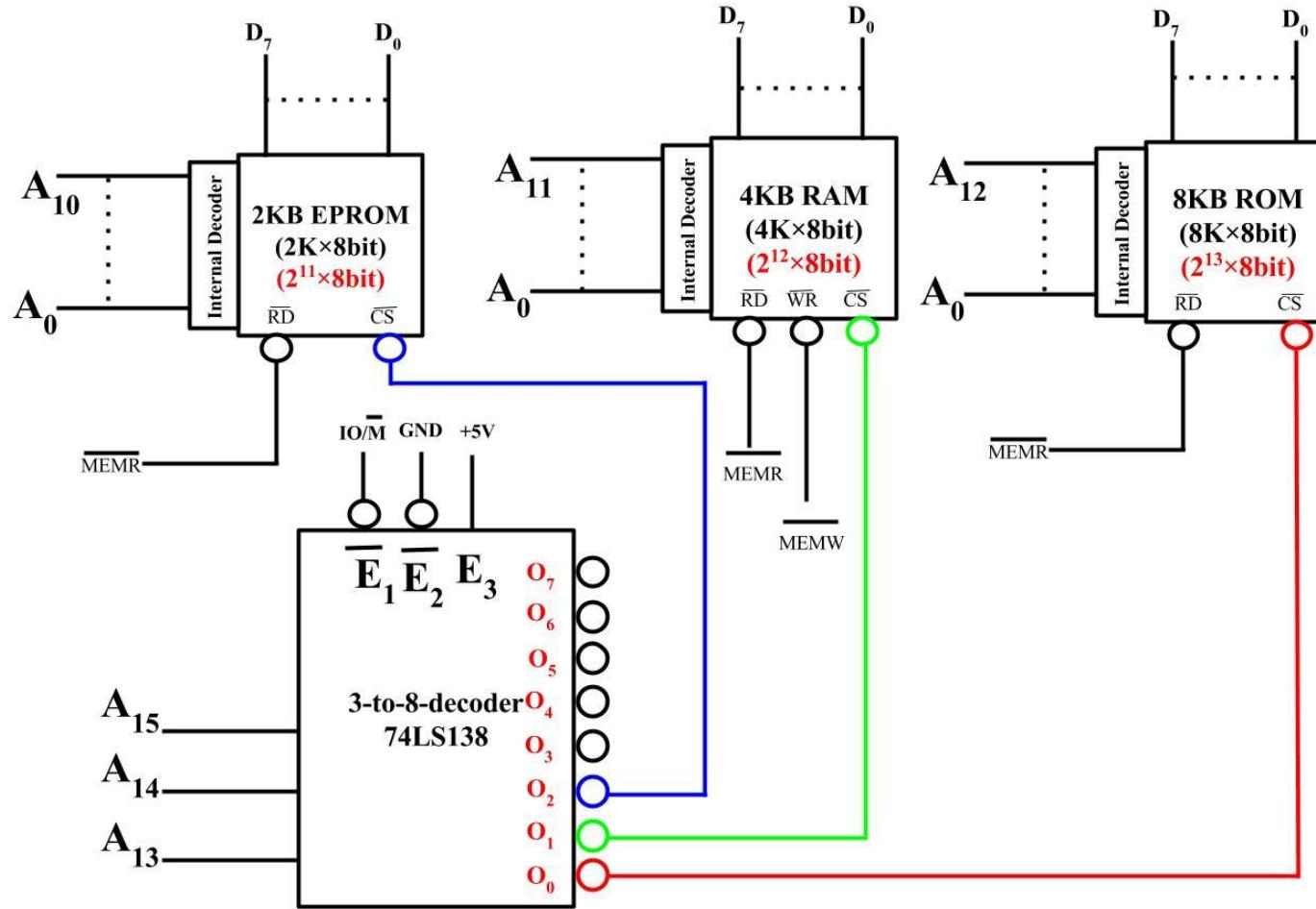
2KB EPROM

$2K \times 8\text{bit}$

$\Rightarrow 2^{11} \times 8 \text{ bit}$



Solution



Address Allocation for Memory

Memory Address Range of ROM

A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	⇒	0000H
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒ 0000H
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒ 1FFFH

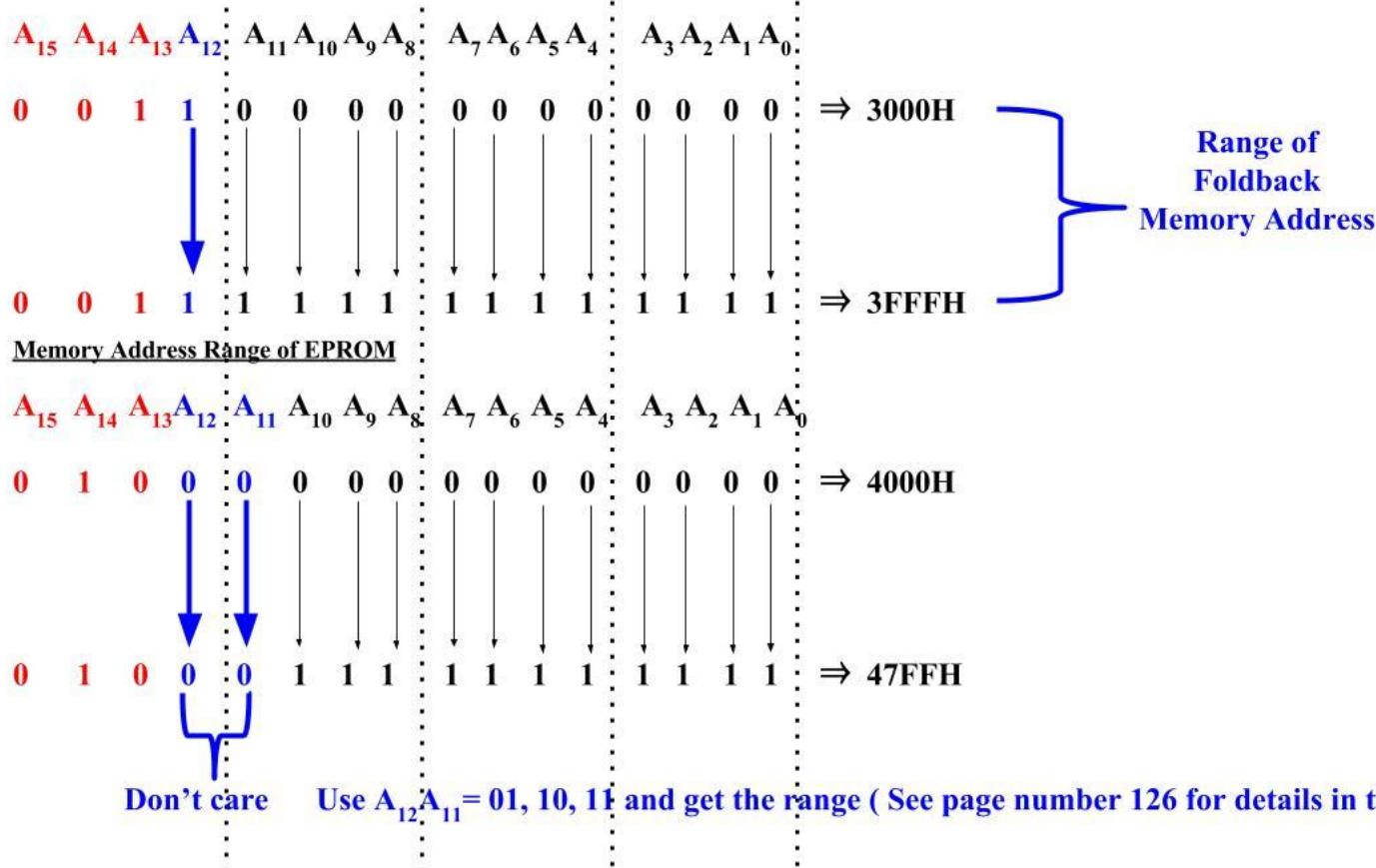
Memory Address Range of RAM

A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	⇒	2000H
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⇒ 2000H
0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	⇒ 2FFFH

Don't care

Foldback Memory or Mirror Memory

Foldback Memory or Mirror Memory



Input and Output (I/O) Devices

There are two different methods by which I/O devices can be interfaced; one uses an 8-bit address and the other uses a 16-bit address.

1) I/O with 8-bit addresses(Peripheral-Mapped I/O)

- ✓ Microprocessor uses eight address lines to identify an input or output devices
- ✓ It is also known as I/O-mapped I/O
- ✓ I/O space is separate from memory space
- ✓ The eight address lines can have 256 addresses, thus , the microprocessor can identify 256 input devices and 256 output devices with addresses ranging from 00H to FFH
- ✓ The input and output devices are differentiated by the control signal
- ✓ The entire range of I/O addresses from 00H to FFH is known as an I/O map
- ✓ And individual addresses are referred to as I/O device addresses, or I/O port numbers



2) I/O with 16-bit addresses(Memory-Mapped I/O)

- ✓ Microprocessor uses 16 address lines to identify an input or output devices
- ✓ I/O devices are connected as if it is a memory register
- ✓ It is also known as Memory-mapped I/O
- ✓ The microprocessor uses the same control signal (memory read or write) and instructions as those of memory
- ✓ The 16 address lines can have 65,536 addresses, thus , the microprocessor can identify 65,536 input devices and 65,536 output devices with addresses ranging from 0000H to FFFFH



Memory-Mapped I/O Vs I/O-Mapped I/O

Memory-Mapped I/O

- In this device address is 16-bit
- MEMR** and **MEMW** control signals are used to write and read I/O operations
- Instructions available are LDA, STA, MOV etc
- Data transfer is between any register and I/O device
- Maximum number of I/O devices are 65,536 theoretically

I/O-Mapped I/O

- In this device address is 8-bit
- IOR** and **IOW** control signals are used to control read and write I/O operations
- Instructions available are IN and OUT
- Data transfer is between accumulator and I/O device
- Maximum number of I/O devices are 256



NEXT WEEK

CHAPTER 12



THE END



Interrupts



Definition

Interrupts are events occurring asynchronously/synchronously sending signals to the CPU to inform their occurrence and for the CPU to take appropriate action.



Block Diagram

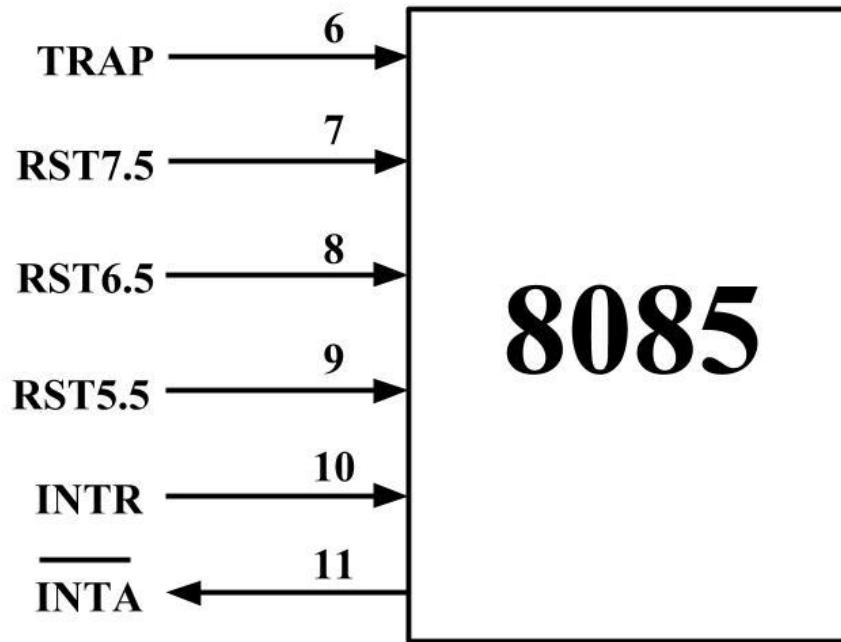


Figure 1: The 8085 µP signals

Interrupt Priority

- ❑ TRAP
- ❑ RST 7.5
- ❑ RST 6.5
- ❑ RST 5.5
- ❑ INTR

Decreasing Priority



Types of Interrupts

1) Non-Maskable

TRAP (cannot be masked, i.e delayed)

2) Maskable Interrupts (can be delayed or masked)

RST 7.5

RST 6.5

RST 5.5

INTR



8085 Vectored Interrupts

- The **RSTs** and **TRAP** are automatically vectored(transferred to specific locations without any external hardware)
- **INTR** is the only **non-vectored interrupt**, requires external hardware to supply a call location to restart the execution



Process of Interrupt Operation

Step 1: The interrupt process should be enabled by writing the instruction EI

Instruction 1: EI (Enable Interrupt)

- => 1-byte instruction
- => sets the interrupt enable flip-flop and enables the interrupt process
- => system reset or an interrupt disables the interrupt



Instruction 2: DI (Disable Interrupt)

- => 1-byte instruction
- => resets the interrupt enable flip-flop and disables the interrupt
- => It should be included in a program segment where an interrupt from an outside source cannot be tolerated



Step 2: When the μ P is executing a program, it checks the INTR line during the execution of each instruction

Step 3: If the line INTR is high and the interrupt is enabled, the μ P completes the current instruction, disables the interrupt Enable flip-flop and sends a signal called **INTA** - Interrupt Acknowledge (active low). The processor cannot accept any interrupt requests until the interrupt flip-flop is enabled again



Step 4: The signal **INTA** is used to insert a restart (RST) instruction through external h/w

Step 5: When the μ P receives an RST instruction, it saves the memory address of the next instruction on the stack

Step 6: Assuming that the task to be performed is written as a subroutine at the specified location, the processor performs the task. This subroutine is known as a service routine or Interrupt Service Routine (ISR)



- Step 7:** The ISR should include the instruction EI to enable the interrupt again
- Step 8:** At the end of the subroutine, the RET instruction retrieves the memory address where the program was interrupted and continues the execution



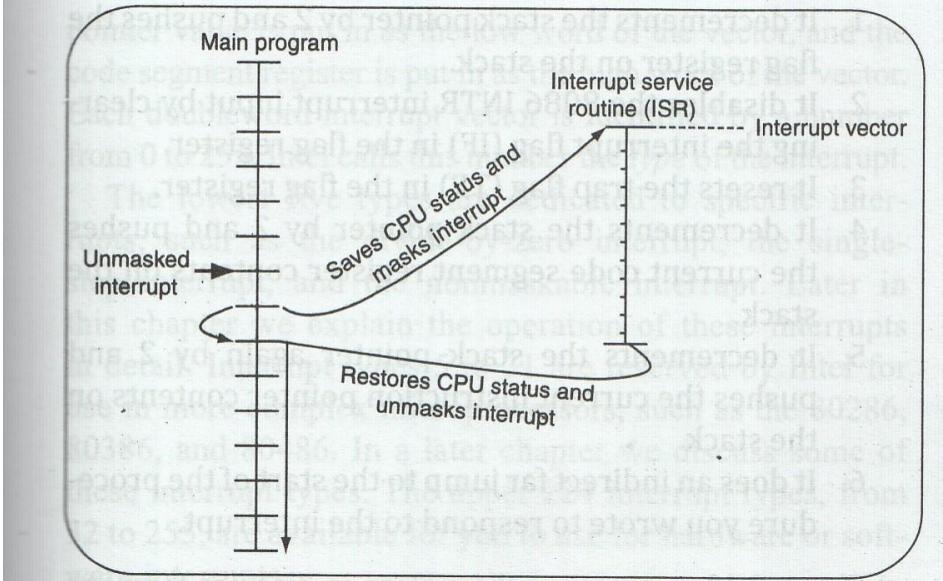


Fig. 9.9 Interrupt processing details.

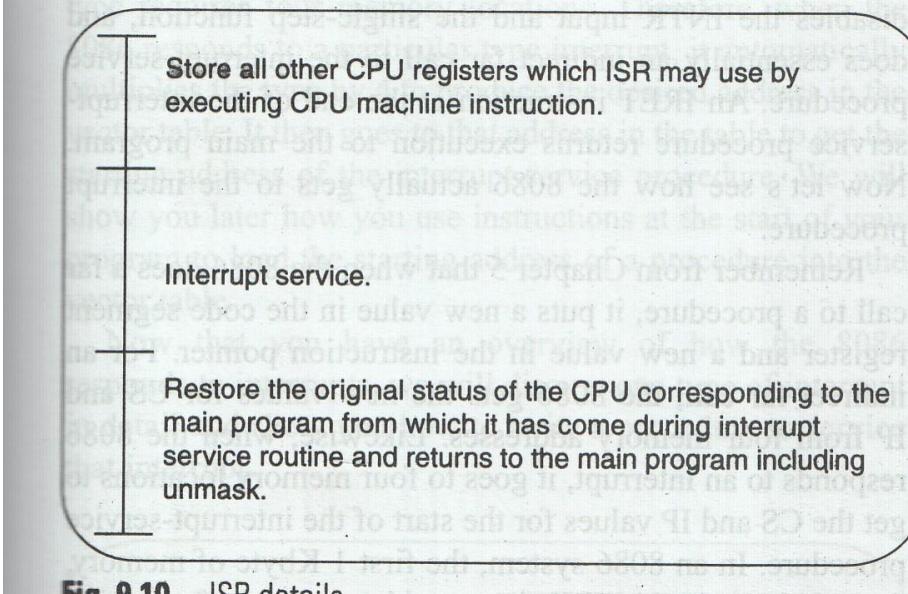


Fig. 9.10 ISR details

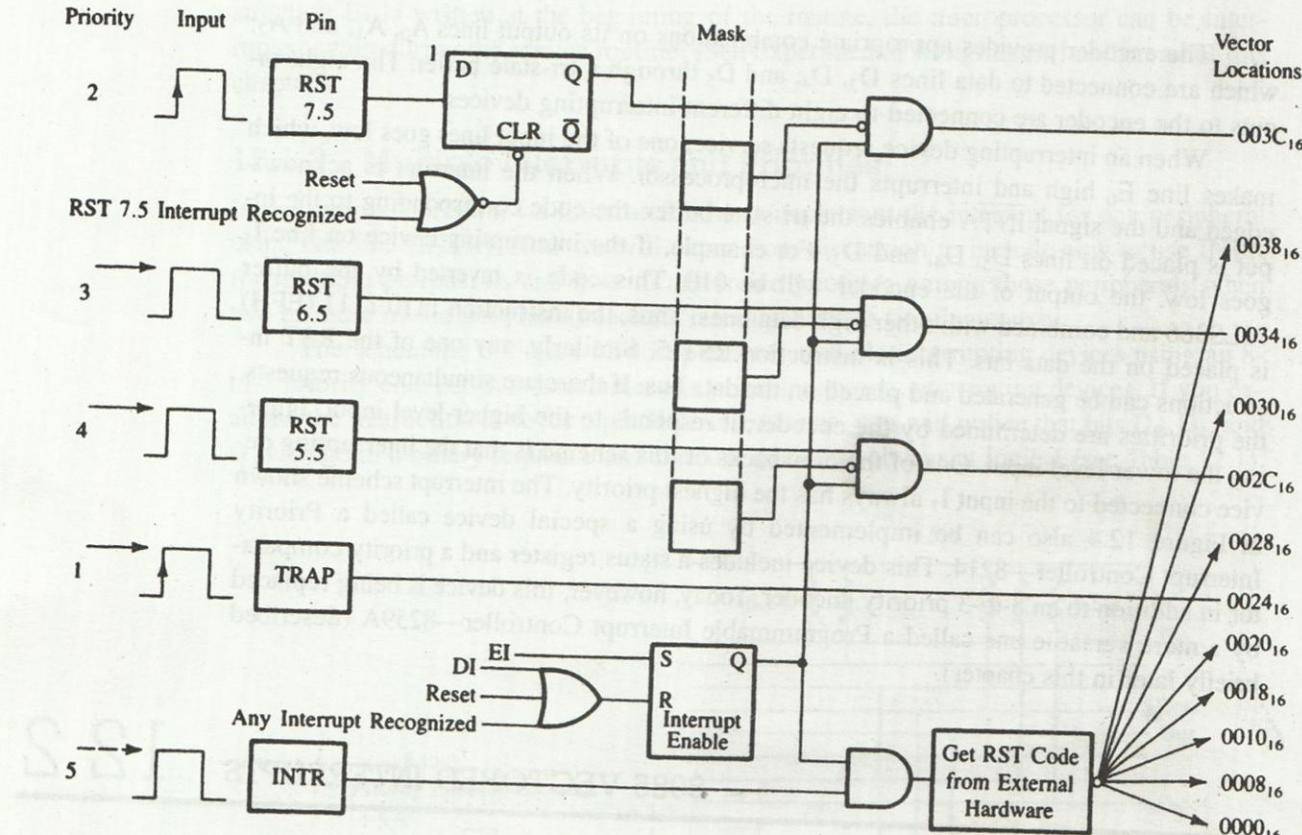


FIGURE 12.5-

The 8085 Interrupts and Vector Locations

SOURCE: Intel Corporation, *MCS 80/85 Student Study Guide* (Santa Clara, Calif.: Author, 1979).

TABLE 12.2
Summary of Interrupts in 8085

Interrupts	Type	Instructions	Hardware	Trigger	Vector
TRAP	Nonmaskable	<input type="checkbox"/> Independent of EI and DI	No external hardware	Level- and Edge-sensitive	0024H
RST 7.5	Maskable	<input type="checkbox"/> Controlled by EI and DI <input type="checkbox"/> Unmasked by SIM	No external hardware	Edge-sensitive	003CH
RST 6.5	Maskable	<input type="checkbox"/> Controlled by EI and DI <input type="checkbox"/> Unmasked by SIM	No external hardware	Level-sensitive	0034H
RST 5.5	Maskable	<input type="checkbox"/> Controlled by EI and DI <input type="checkbox"/> Unmasked by SIM	No external hardware	Level-sensitive	002CH
INTR 8085	Maskable	<input type="checkbox"/> Controlled by EI and DI	RST code from external hardware	Level-sensitive	038H ↑ 0000H



Interrupt Vector

- The starting address of an Interrupt-Service Procedure (ISP) is often called the Interrupt Vector or the Interrupt Pointer
- Interrupts are mapped onto a memory area called Interrupt Vector Table (IVT)
- The IVT is usually located in between 0000H to 00FFH
- IVT holds the vectors that redirect the µP to the correct location when an interrupt arrives



Interrupt	Hex-Code	Call Location	$\times 8$	Base-10	Base-16
RST 0	C7	0000H	0×8	0_{10}	0_{16}
RST 1	CF	0008H	1×8	8_{10}	08_{16}
RST 2	D7	0010H	2×8	16_{10}	10_{16}
RST 3	DF	0018H	3×8	24_{10}	18_{16}
RST 4	E7	0020H	4×8	32_{10}	20_{16}
RST 5	EF	0028H	5×8	40_{10}	28_{16}
RST 6	F7	0030H	6×8	48_{10}	30_{16}
RST 7	FF	0038H	7×8	56_{10}	38_{16}
<u>Also</u>					
RST 7.5		003CH	7.5×8	60_{10}	$003C_{16}$
RST 6.5		0034H	6.5×8	52_{10}	0034_{16}
RST 5.5		002CH	5.5×8	44_{10}	$002C_{16}$
TRAP		0024H	4.5×8	36_{10}	0024_{16}

Note:

TRAP \approx RST 4.5

TRAP

- TRAP, a nonmaskable interrupt known as NMI has the highest priority among the interrupt signals, it need not be enabled, and it cannot be disabled
- It is level-and edge-sensitive, meaning that the input should go high and stay high to be acknowledged



RST 7.5, RST 6.5, and RST 5.5

These are maskable interrupts. And are enabled under program control with two instructions: EI (Enable Interrupt), and SIM (Set Interrupt Mask)



SIM (SET Interrupt Mask)

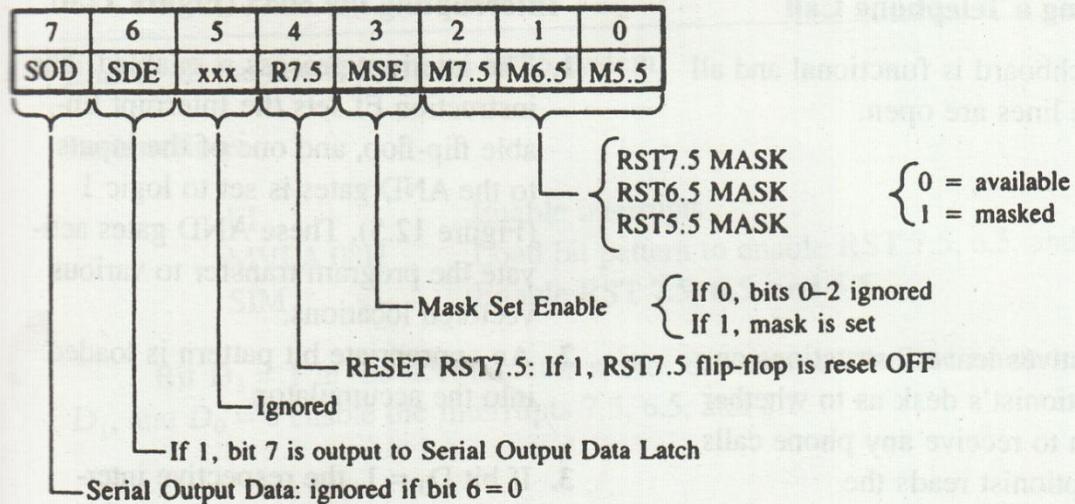


FIGURE 12.6
Interpretation of the Accumulator Bit Pattern for the SIM Instruction

SOURCE: Intel Corporation, *Assembly Language Programming Manual* (Santa Clara, Calif.: Author, 1979), pp. 3–59.



SIM (SET Interrupt Mask)

- =>This is a 1-byte instruction
- =>Can be used for three different functions
 - ✓ One function is to set mask for RST 7.5 RST 6.5, and RST 5.5
 - ✓ The second function is to reset RST 7.5 flip-flop
 - ✓ The third function is to implement serial I/O. Bit D7 and D6 of the accumulator are used for serial I/O and do not affect the interrupts



Example1:Enable all the interrupts in an 8085 system

EI ; enable interrupt
MVI A,08H ; load bit pattern to enable RST7.5, RST
6.5, and RST 5.5
SIM ; enable RST7.5, RST 6.5,
and RST 5.5
HLT



Example2: Reset RST 7.5 Interrupt

```
    EI          ; enable interrupt  
    MVI A,18H   ; load bit pattern to reset RST7.5, D4=1  
    SIM        ; reset RST 7.5 interrupt flip-flop  
    HLT
```



Pending Interrupt

Because there are several interrupt lines, when one interrupt request is being served, other interrupt requests may occur and remain pending. The 8085 has an additional instruction called RIM (Read Interrupt Mask) to sense these pending interrupts



RIM

The RIM instruction loads the accumulator with the following information:

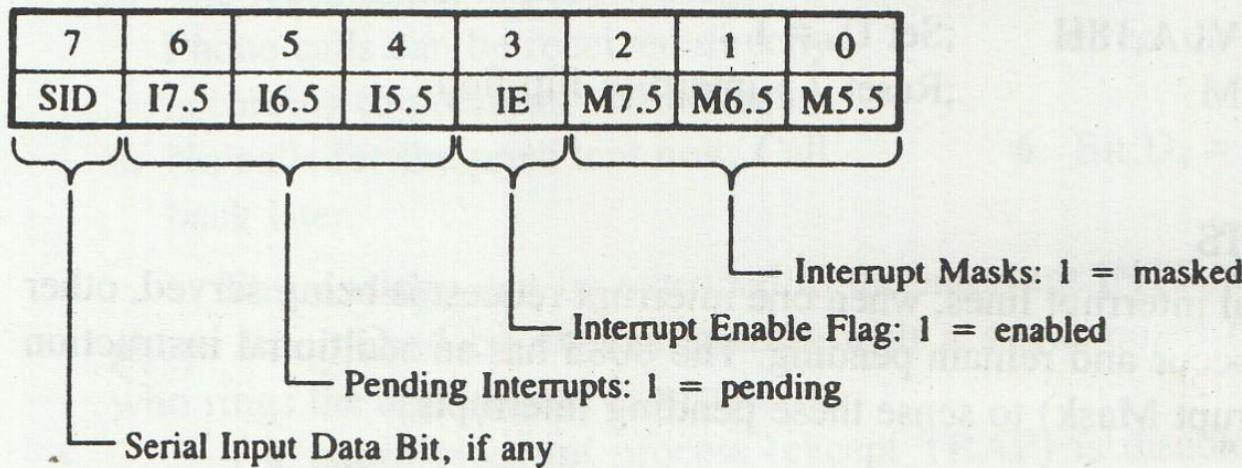


FIGURE 12.7

Interpretation of the Accumulator Bit Pattern for the RIM Instruction

SOURCE: Intel Corporation, *Assembly Language Programming Manual* (Santa Clara, Calif.: Author, 1979), pp. 3-49.



=> Read Interrupt Mask

=> 1 –byte instruction

=> can be used for the following functions

- ✓ To read the interrupt masks.
- ✓ To identify the pending interrupts
- ✓ To receive serial data



NEXT WEEK

8255, 8251,8259, 8237



THE END



8255-Programmable Peripheral Interface (8255-PPI)



Handshake Signal

The μ P can execute the instructions to transfer a byte in **μs**; on the other hand, the printer can take upto 10 to 25 **ms** to print a character. After transferring a character to the printer, the μ P should wait until the printer is ready for the next character; otherwise data will be lost.

To **prevent**, the loss of data or the μ P reading the same data more than once, signals are exchanged between the μ P and a peripheral prior to actual data transfer; these signals are called **handshake signals**.



Definition

The μ P and peripheral operate at different speeds, therefore, signals are exchanged prior to data transfer between the fast-responding μ P and slow-responding peripherals such as printers and data converters. These signals are called handshake signals. These signals are generally provided by programmable devices.



Interfacing Device

Should include:

- ✓ Input and Output registers (a group of latches to hold data)
- ✓ Tri-state buffers
- ✓ Capability for bidirectional data flow
- ✓ Handshake and interrupt signals
- ✓ Control logic
- ✓ Chip select logic
- ✓ Interrupt control logic

(Please see Figure 14.2 in page 428 to understand about control logic)



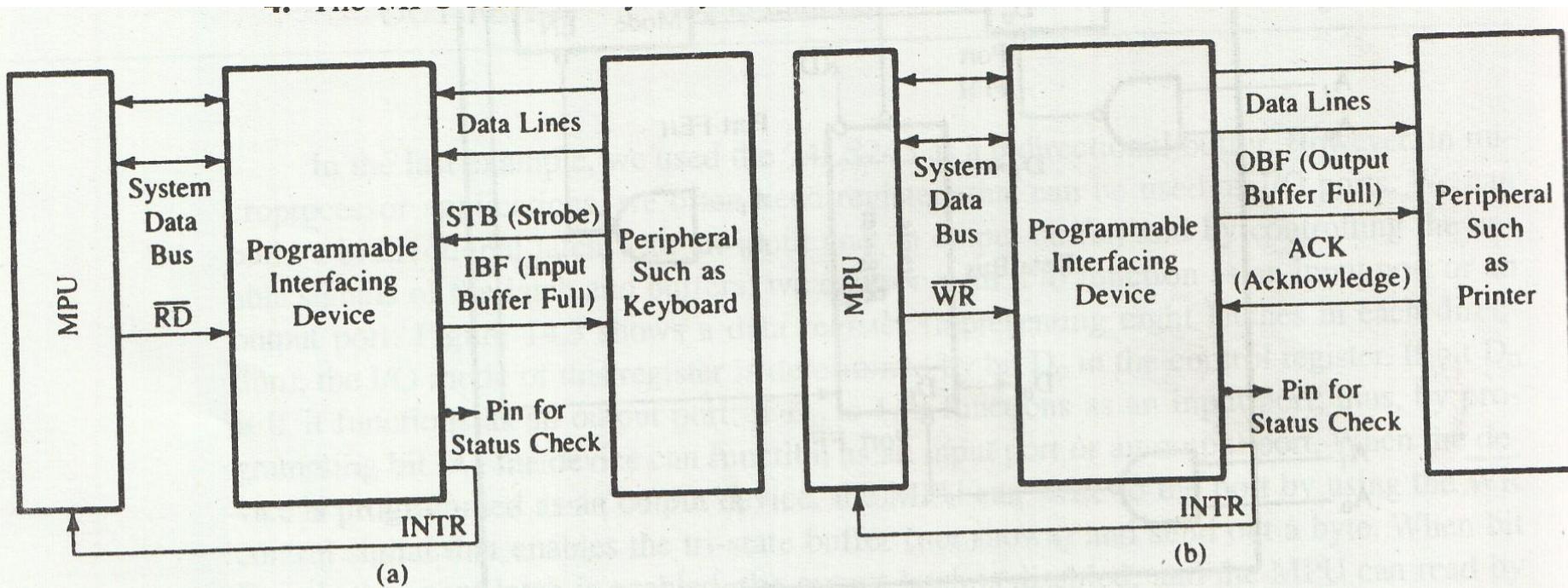


FIGURE 14.4
Interfacing Device with Handshake Signals for Data Input (a) and Data Output (b)

Programmable Devices With Handshake Signals

Data Input with Handshake:

- **Step 1:** A peripheral **strokes or places** a data byte in the input port and informs the interfacing device by sending handshake signal **STB** (strobe)
- **Step 2:** The device informs the peripheral that its input port is **full**-do not send the next byte until this one has been read. This message is conveyed to the peripheral by sending handshake signal **IBF** (Input Buffer Full)
- **Step 3:** The μP keeps checking the status until a byte is available. Or the interfacing device informs the μP, by sending an interrupt, that it has a byte to read.
- **Step 4:** The μP reads the byte by sending control signal **RD**



Data Output with Handshake:

- **Step 1:** The μP writes a byte into the I/O port of the programmable device by sending control signal **WR**
- **Step 2:** The device informs the peripheral, by sending handshake signal **OBF**(Output Buffer Full), that a byte is on the way
- **Step 3:** The peripheral acknowledges the byte by sending back the **ACK** (Acknowledge) signal to the device.
- **Step 4:** The device interrupts the μP to ask for the next byte or the μP finds out that the byte has been acknowledged through the status check.



- ✓ Handshake signals **ACK** and **STB** are input signals to the device and perform similar functions, although they are called by different names
- ✓ Handshake signals **OBF** and **IBF** are output signals from the device and perform similar functions
- ✓ The active levels and labels of these signals are arbitrary and vary considerably from device to device



Important Concepts

A programmable I/O device is likely to have the following elements:

- ✓ A control register in which the μ P can write an instruction
- ✓ A status register that can be read by the μ P
- ✓ I/O devices or registers
- ✓ Control logic
- ✓ Chip select logic
- ✓ Bidirectional data bus
- ✓ Handshake signals and Interrupt Logic

*A programmable I/O device is programmed by writing a specific word called the **control word**, according to the internal logic; its status can be verified by reading the **status register***



8255

- The 8255 is a widely used, programmable, parallel I/O device
- It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O
- The 8255 has 24 I/O pins that can be grouped primarily in two 8-bit ports **A** and **B**, with the remaining eight bits as port C
- The eight bits of port C can be used as individual bits or be grouped in two 4 –bit ports; **C_{UPPER}(C_U)** and **C_{LOWER} (C_L)**
- The functions of these ports are defined by writing a control word in the control register



8255 I/O Ports

GENERAL-PURPOSE PROGRAMMABLE PERIPHERAL DEVICES

461

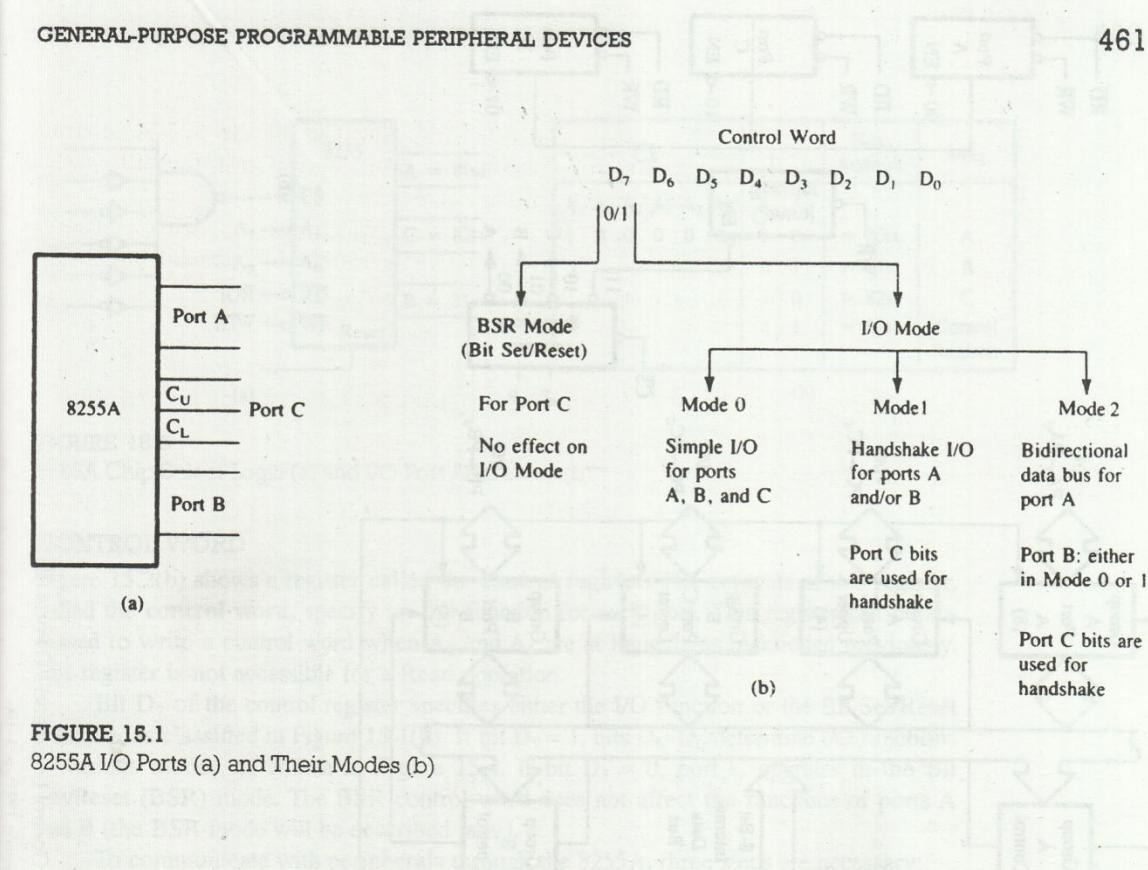
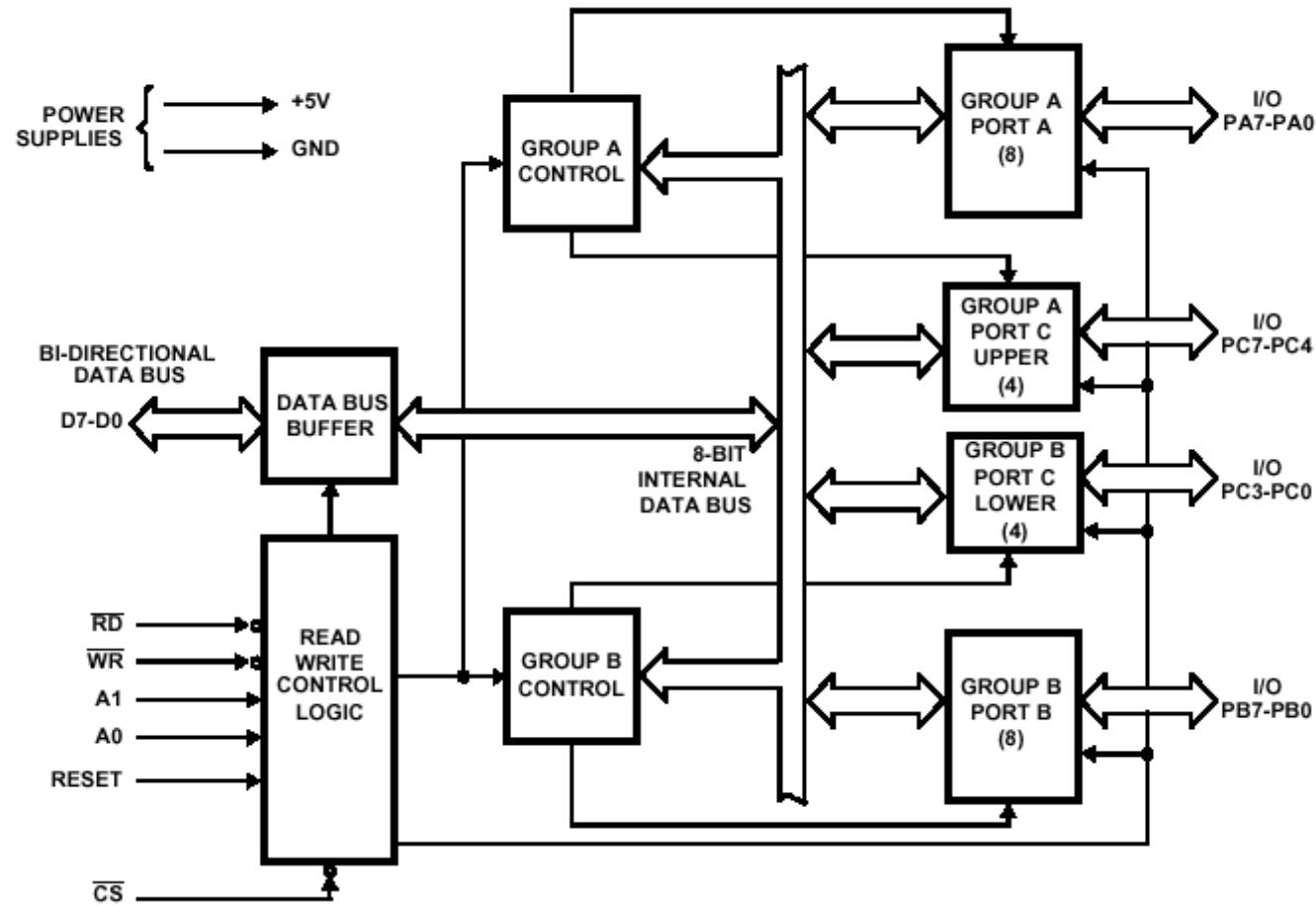


FIGURE 15.1
8255A I/O Ports (a) and Their Modes (b)

Block Diagram of 8255



Description

The block diagram shows two 8-bit ports (A and B), two 4-bit ports (C_U and C_L), **data bus buffer, and control logic**

Control Logic:

- ✓ **\overline{RD}** (Read): This control signal enables the read operation.
- ✓ **\overline{WR}** (Write): This control signal enables the write operation
- ✓ **RESET**: It clears the control register and sets all ports in the input mode
- ✓ **\overline{CS}** , A_0 & A_1 : These are device select signals. **\overline{CS}** Is connected to a decoded address, and A_0 & A_1 are generally connected to μP address lines A_0 & A_1 , respectively



Device Select Signals

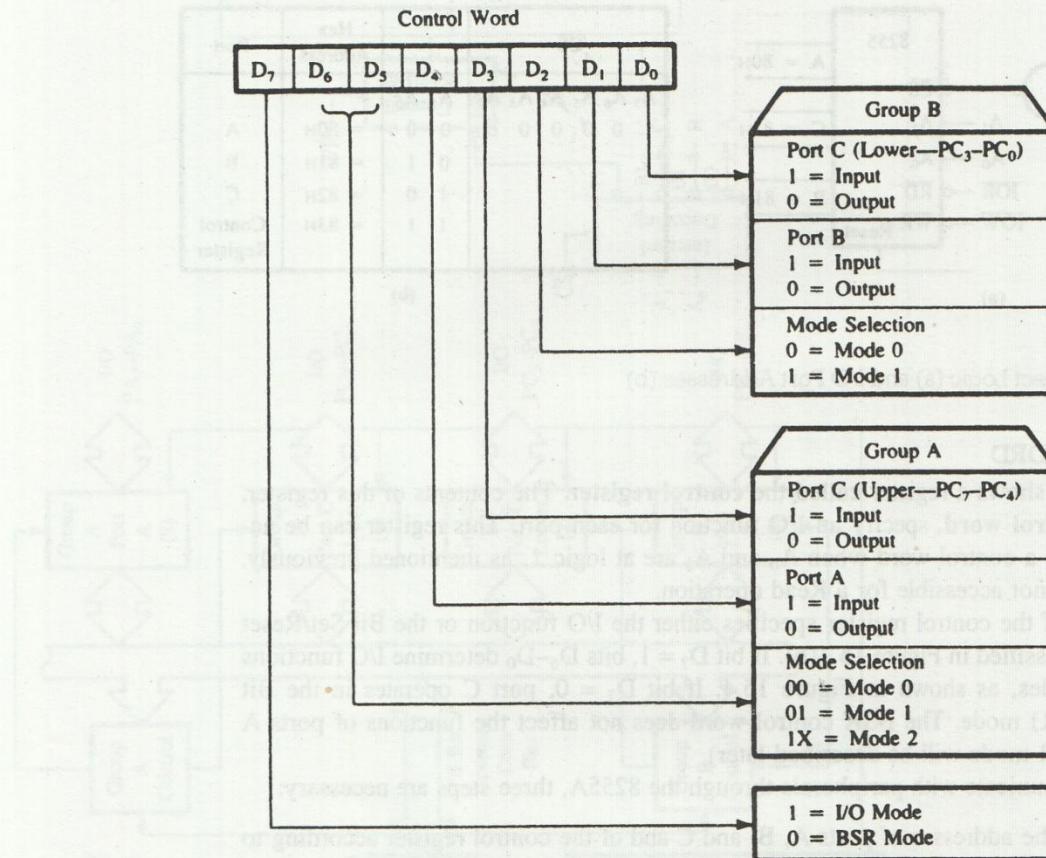
\overline{CS}	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255 is not selected



Control Word

- ✓ The contents of the control register is called the control word. It specifies an I/O function for each port. This register can be accessed to write a control word when A_0 & A_1 are at logic 1. The register is not accessible for a Read Operation.



**FIGURE 15.4**

8255A Control Word Format for I/O Mode

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-104.

Features

- ✓ Bit D₇ of the control register specifies either the I/O function or the Bit Set/Reset function
- ✓ If **D₇ = 1** , bits D₇ – D₆ determine **I/O** functions in various modes
- ✓ If **D₇ = 0** , port C operates in the **Bit Set/reset(BSR)** mode. The BSR control word does not affect the functions of ports A and B
- ✓ To communicate with the peripherals through the 8255, three steps are necessary
 - Determine the addresses of port A, port B, port C and control register according to the chip select logic and address lines A₀ , and A₁
 - Write a control word in the control register
 - Write I/O instructions to communicate with peripherals through ports A, B, and C



Mode 0 (Simple Input or Output)

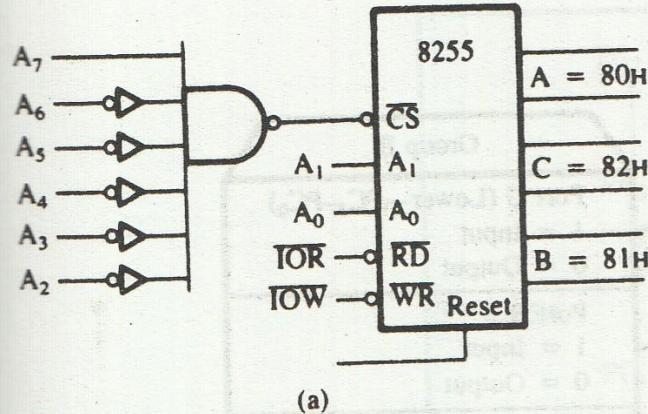
- ✓ In this mode, ports A and B are used as two simple 8-bit I/O ports and port C as two 4-bit ports. Each port (or half-port, in case of C) can be programmed to function as simply an input port or an output port
- ✓ The input/output features in Mode 0 are as follows
 - ❑ Output are latched
 - ❑ Inputs are not latched
 - ❑ Ports do not have handshake or interrupt capability



Example 1

463

GENERAL-PURPOSE PROGRAMMABLE PERIPHERAL DEVICES



(a)

CS		Hex Address	Port
A ₇ A ₆ A ₅ A ₄ A ₃ A ₂	A ₁ A ₀	= 80H	A
1 0 0 0 0 0	0 0	= 81H	B
	0 1	= 82H	C
	1 0		
	1 1	= 83H	Control Register

(b)

FIGURE 15.3
8255A Chip Select Logic (a) and I/O Port Addresses (b)

Example 2

Identify the mode 0 control word to configure port A and port C_U as output ports and port B and port C_L as input ports.

Solution:

D7	D6	D5	D4	D3	D2	D1	D0	Hex
1	0	0	0	0	0	1	1	83H

I/O function Port A in Mode 0 Port A = Outport Port C_U = output Port B in Mode 0 Port B = Input Port C_L = Input



BSR (Bit Set/Reset) Mode

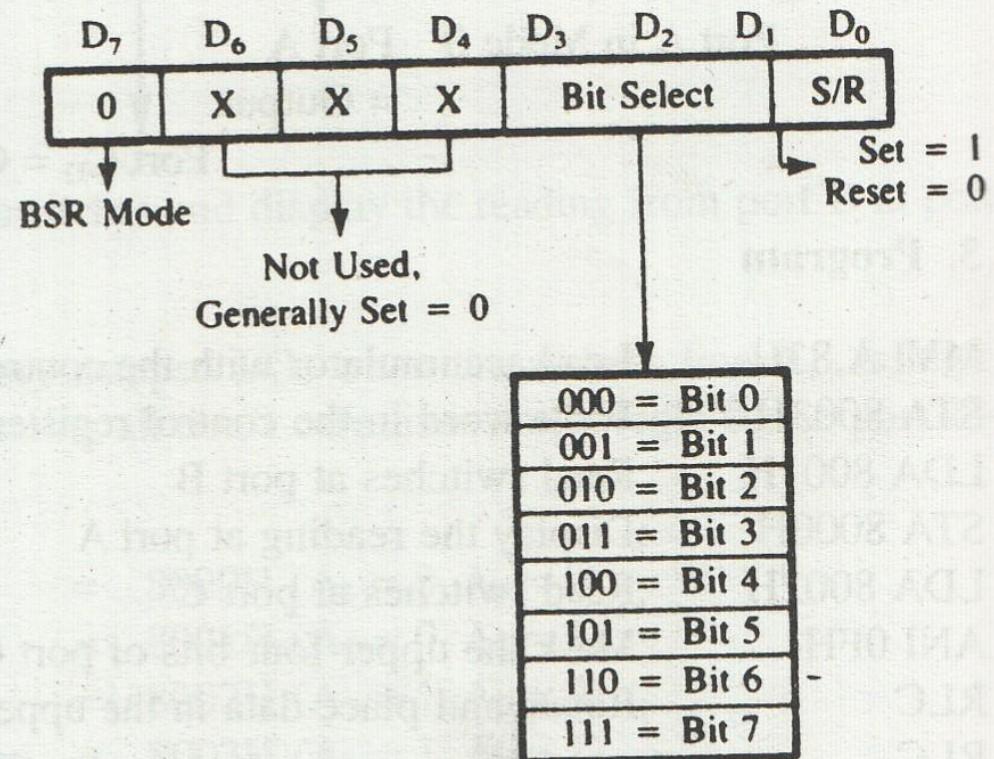
- ✓ The BSR mode is concerned only with the eight bits of port C, which can be set or reset by writing an appropriate control word in the control register
- ✓ A control with bit $D_7 = 0$ is recognized as a BSR control word, and it does not alter any previously transmitted control word with $D_7 = 1$; thus the I/O operations of ports A and B are not affected by a BSR control word
- ✓ In BSR mode, individual bits of port C can be used for applications such as an on/off switch



BSR Control Word

FIGURE 15.6

8255A Control Word Format in the BSR Mode



Example 3

Determine the BSR control word to set bits PC_7 and PC_3

D₇	D₆	D₅	D₄	D₃	D₂	D₁	D₀	Hex
0	0	0	0	1	1	1	1	=>0FH

Bit 7

D₇	D₆	D₅	D₄	D₃	D₂	D₁	D₀	Hex
0	0	0	0	0	1	1	1	=>07H

Bit 3

Mode 1: Input or Output with Handshake

- ✓ In Mode 1, handshake signals are exchanged between the µP and peripherals prior to data transfer. The features of this mode include the following:
- ✓ Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports
- ✓ Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions. Port A uses the upper three signals **PC₃**, **PC₄**, and **PC₅**. Port B uses the lower three signals: **PC₂**, **PC₁**, and **PC₀**
- ✓ Input and Output data are latched
- ✓ Interrupt logic is supported



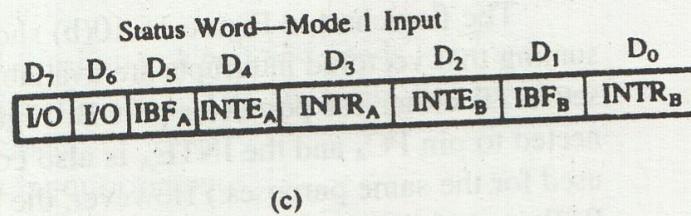
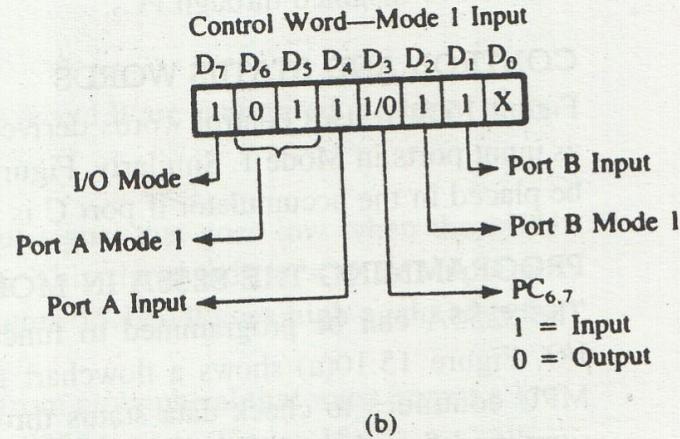
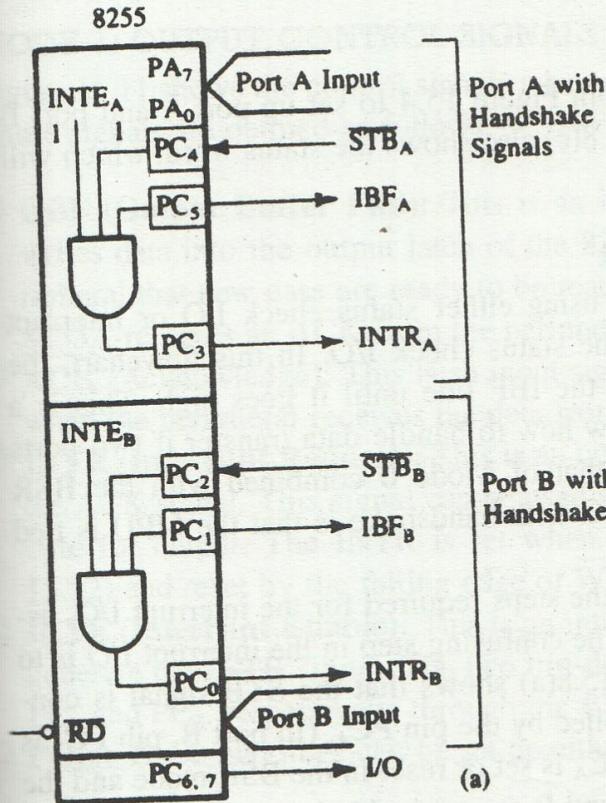


FIGURE 15.8

8255A Mode 1: Input Configuration

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-110.

Mode 2(Bidirectional Data transfer)

- ✓ In this Mode, port A can be configured as the bidirectional port and port B either in Mode 0 or Mode 1.
- ✓ Port A **uses five signals** from port C as handshake signals for data transfer
- ✓ The **remaining three signals** from port C can be used either as simple I/O or a handshake for port B



Example 4

The control word for 8255 is specified as follows 90H. Find out the operation mode of each port [2003, 8 marks]

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	Hex
1	0	0	1	0	0	0	0	=>90H
I/O function	Port A in Mode 0	Port A (Input Port)	Port C _U (Output Port)	Port B (Mode 0)	Port B (Output Port)	Port C _L (Output Port)		



Important

Example 5: Write different modes of 8255 PPI.[8 marks, 2009]

Example 6: Make the block diagram of Intel 8255 PPI, and explain the functions of the sub blocks [Fall 2014]



NEXT CLASS

8259



THANK YOU



8259-Programmable Interrupt Controller (8259-PIC)



Programmable Interface Device

A Programmable interface device is designed to perform various input/output functions. Such a device can be set up to perform specific functions by writing an instruction (or instructions) in its internal register, called the control word



Block Diagram

INTERRUPTS

397

Block Diagram

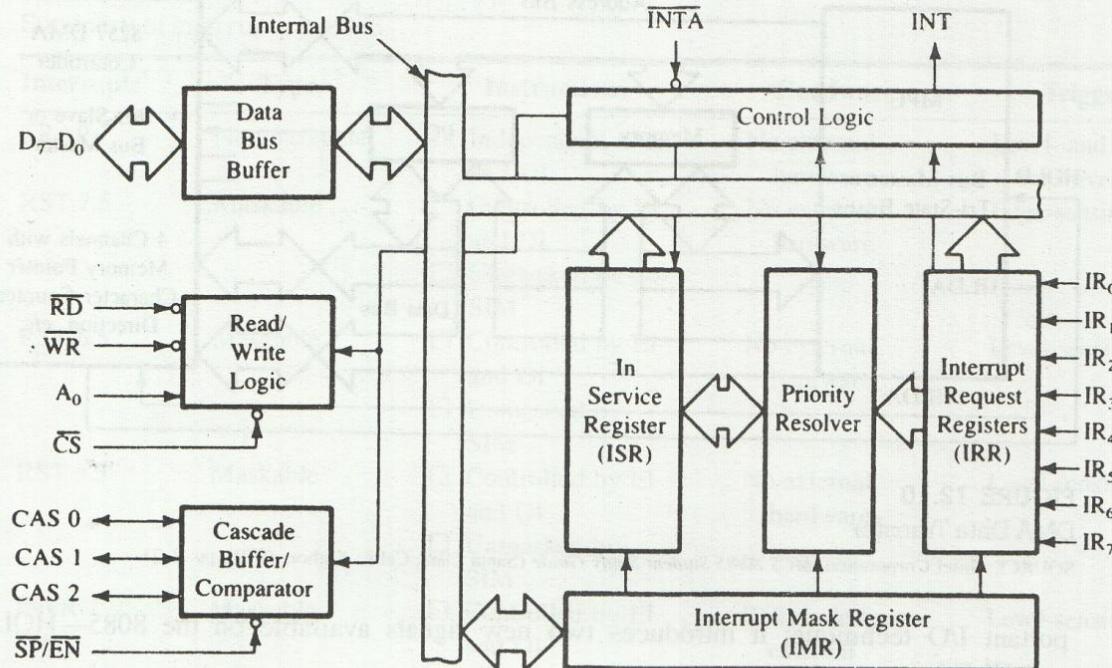


FIGURE 12.9

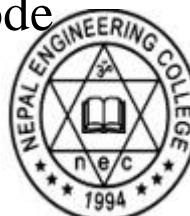
The 8259A Block Diagram

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), pp. 3-171.

Features

The 8259 interrupt controller can:

- ✓ Manage eight interrupts according to the instructions written into its control registers. This is eqvt to providing eight interrupt pins on the processor in place of one INTR (8085) pin
- ✓ Vector an interrupt request anywhere in the memory map
- ✓ Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode
- ✓ Mask each interrupt request individually
- ✓ Read the status of pending interrupts, in-service interrupts, and masked interrupts
- ✓ Be set up to accept either the level-triggered or the edge-triggered interrupt request
- ✓ Be expanded to 64 priority levels by cascading additional 8259s
- ✓ Be set up to work either with the 8085 μ P mode or the 8086/8088 μ P mode



Description

The internal block diagram of the 8259 includes eight block

- I. Control Logic
- II. Read/Write Logic
- III. Data Bus Buffer
- IV. Three Registers IRR, ISR, and IMR
- V. Priority Resolver and
- VI. Cascade Buffer



Read/Write Logic

When the address line A0 is at logic 0, the controller is selected to write a command or read a status.



Control Logic

- ✓ This block has two pins INT (interrupt) as an output, **INTA** (Interrupt Acknowledge) as an input
- ✓ The INT is connected to the interrupt pin of the μ P. Whenever a valid interrupt is asserted, this signal goes high. The **INTA** is the Interrupt Acknowledge signal from the μ P



Interrupt Registers and Priority Resolver

- ✓ The Interrupt Request Register (IRR) has eight input lines (**IR₀-IR₇**) for interrupts. When these lines go high, the requests are stored in the register
- ✓ The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stores the masking bits of the interrupt lines to be masked
- ✓ The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the μ P



Cascade Buffer/Comparator

- ✓ This block is used to expand the number of interrupt levels by cascading two or more 8259s



Interrupt Operation

To implement interrupts, the Interrupt Enable flip-flop in the µP should be enabled by writing the EI instruction, and the 8259 requires two types of control words; Initialization Command Words (ICWs) and Operational Command Words (OCWs)

ICWs:

ICWs are used to set up the proper conditions and specify RST vector addresses.

OCWs:

OCWs are used to perform functions such as masking interrupts, setting up status-read operations, etc.



After the initialization, the following sequence of events occurs when one or more interrupt request lines go high.

- I. The IRR stores the requests
- II. The priority resolver resolves the priority and sets the INT high when appropriate
- III. The processor acknowledges the interrupt by sending **INTA**
- IV. After the **INTA** is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being serviced, and the corresponding bit in the IRR is reset to indicate that the request is accepted. Then, the opcode for the CALL instruction is placed on the data bus
- V. When the processor decodes the CALL instruction, it places two more **INTA** signal on the data bus
- VI. When the 8259 receives the second **INTA**, it places the low order byte of the CALL address on the data bus. At the third **INTA**, it places the high-order byte on the data bus. CALL address is the vector memory location for the interrupt; this is placed in the control register during the initialization.
- VII. During the 3rd **INTA** pulse, the ISR bit is reset
- VIII. The program sequence is transferred to the memory location specified by the CALL



Programming the 8259

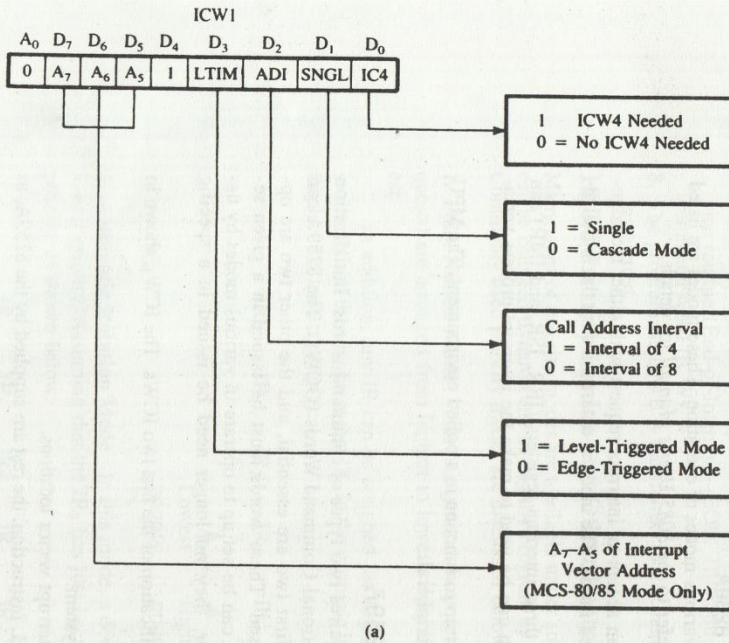
- ✓ The 8259 requires two types of command words, Initialization Command Words (ICWs) and Operational Command Words (OCWs)
- ✓ The 8259 can be initialized with four ICWs; the first two are essential, and the other two are optional based on the modes being used
- ✓ These words must be issued in a given sequence
- ✓ Once initialized, the 8259 can be set up to operate in various modes by using three different OCWs; however, they no longer need be issued in a specific sequence

(Please see page 510 in text book for ICW1 and ICW2)



- ✓ The ICW1 specifies; single or multiple 8259s in the system
- ✓ 4-or 8-bit interval between the interrupt vector location
- ✓ The address bits A_7-A_5 of the CALL instructions; the rest are supplied by the 8259





IR	Interval = 4							
	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
7	A ₇	A ₆	A ₅	I	I	I	0	0
6	A ₇	A ₆	A ₅	I	I	0	0	0
5	A ₇	A ₆	A ₅	I	0	1	0	0
4	A ₇	A ₆	A ₅	I	0	0	0	0
3	A ₇	A ₆	A ₅	0	1	1	0	0
2	A ₇	A ₆	A ₅	0	1	0	0	0
1	A ₇	A ₆	A ₅	0	0	1	0	0
0	A ₇	A ₆	A ₅	0	0	0	0	0

IR	Interval = 8							
	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
7	A ₇	A ₆	I	I	I	0	0	0
6	A ₇	A ₆	I	I	0	0	0	0
5	A ₇	A ₆	I	0	1	0	0	0
4	A ₇	A ₆	I	0	0	0	0	0
3	A ₇	A ₆	0	1	1	0	0	0
2	A ₇	A ₆	0	1	0	0	0	0
1	A ₇	A ₆	0	0	1	0	0	0
0	A ₇	A ₆	0	0	0	0	0	0

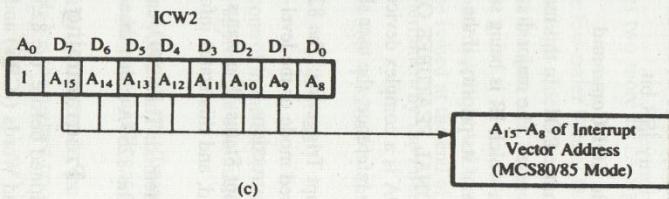


FIGURE 15.30
Initialization Command Words for the 8259A

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-181.

NEXT CLASS

8251



THANK YOU



8251-Programmable Communication Interface (8251-PCI)



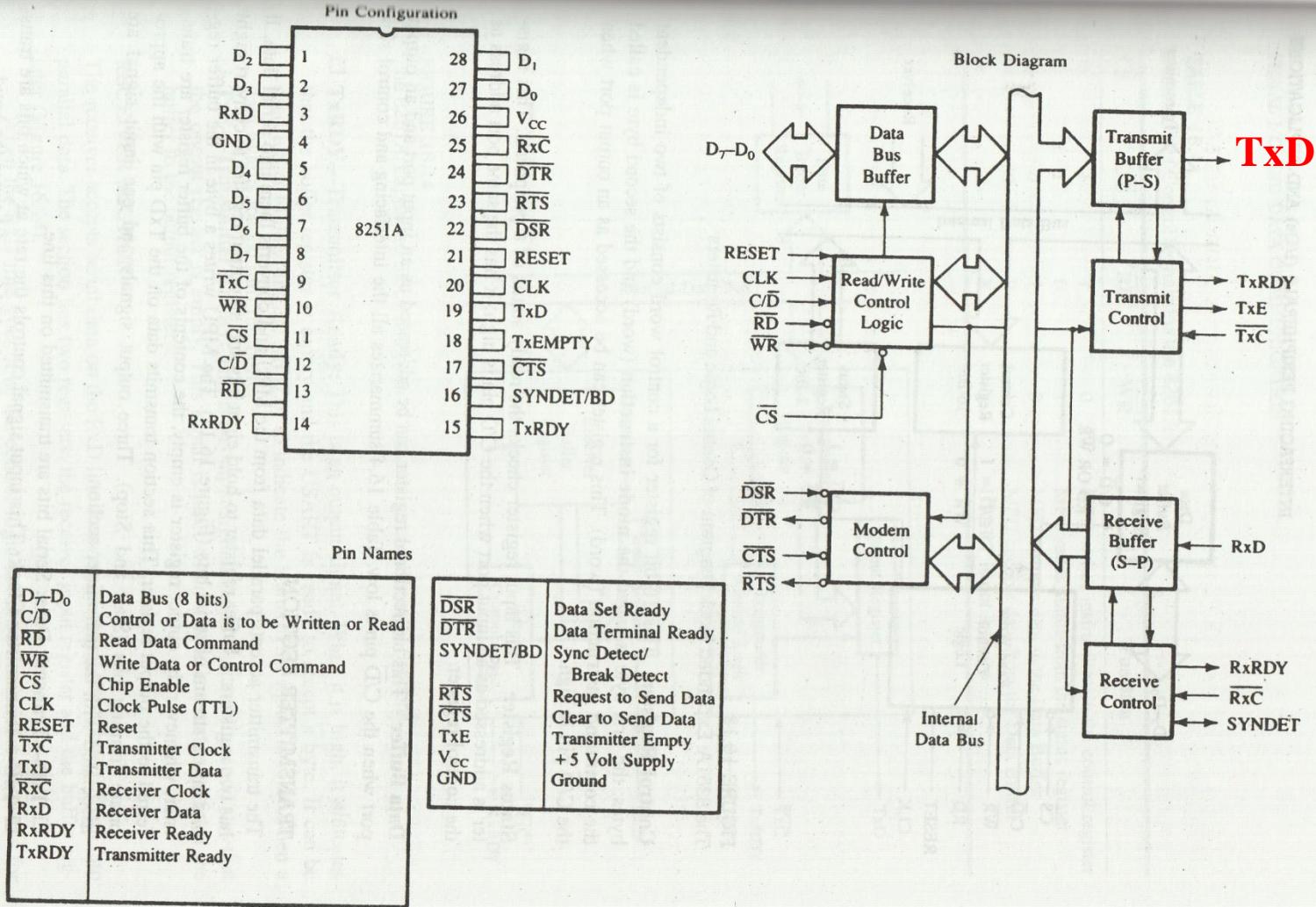


FIGURE 16.12

The 8251A: Block Diagram, Pin Configuration, and Description

SOURCE: Intel Corporation, *Connectivity* (Santa Clara, Calif.: Author, 1993), p. 2-2.

Features

- ✓ The 8251 is a programmable chip designed for **synchronous** and **asynchronous serial data communication**
- ✓ The INTEL 8251 is the industry standard Universal Synchronous/Asynchronous Receiver/Transmitter (**USART**) designed for data communications
- ✓ The 8251 is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique
- ✓ The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission
- ✓ Simultaneously, it can receive serial data streams and convert them into parallel data character for the CPU
- ✓ The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU



Description

The block diagram includes five section

- ✓ Read/write Control Logic
- ✓ Transmitter
- ✓ Receiver
- ✓ Data Bus Buffer and
- ✓ Modem Control



Read/Write Control Logic

The control logic interfaces the chip with the processor, determines the functions of the chip according to the control word in its register, and monitors the data flow



Transmitter

The transmitter section converts a parallel word received from the processor into serial bits and transmits them over the TxD line to a peripheral



Receiver

The transmitter section receives serial bits from a peripheral, converts them into a parallel word, and transfers the word to the μ P



Modem Control

The modem control is used to establish data communication through modems over telephone lines



Data Buffer

This bidirectional register can be addressed as an input and an output port when the **C/D** pin is low

Control/Data pin (C/D) :

When this signal is high, the control register or the status register is addressed; when it is low, the data buffer is addressed. The control register and the status register are differentiated by **RD** and **WR** signals respectively.



Control Register

This 16-bit register for a control word consists of two independent bytes; the first byte is called **Mode Instruction (Word)** and the second byte is called the **Command Instruction (Word)**. This register can be accessed as an output port when the **C/D** pin is high



Status Register

This input register checks the ready status of a peripheral. This register is addressed as an input port when the **C/D̄** pin is high. It has the same port address as the control register



Initialization of 8251

To implement serial communication, 8085 must inform 8251 of all the details, such as mode, baud, stop bits, parity etc. Therefore prior to data transfer, a set of control words must be loaded into 16-bit control register of the 8251. In addition, 8085 must check the readiness of a peripheral by reading the status register. The control words are divided into two formats: mode word and command word.

(Please see page number 547 to refer the format of Control Word and Status Word Format)



NEXT CLASS

8237



THANK YOU



8237-DMA Controller



Direct Memory Access

398

INTERFACING PERIPHERALS (I/Os) AND APPLICATIONS

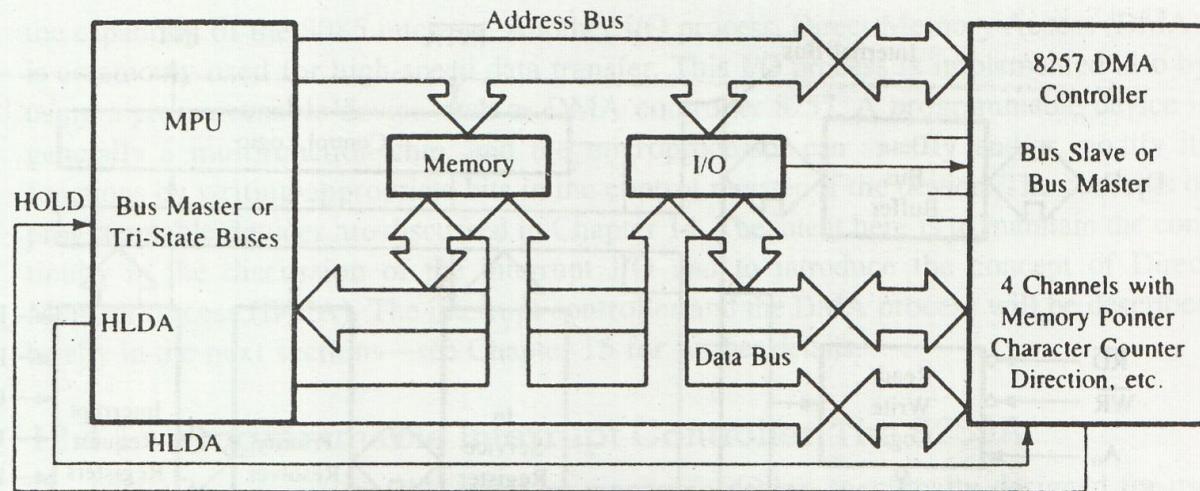


FIGURE 12.10
DMA Data Transfer

SOURCE: Intel Corporation, *MCS 80/85 Student Study Guide* (Santa Clara, Calif.: Author, 1979), pp. 2-21.

DMA Operation

- ✓ Direct Memory Access (DMA) is an I/O technique commonly used for high-speed data transfer; for example , data transfer between memory and a floppy disk.
- ✓ In DMA, μ P releases the control of the buses to a device called a DMA controller. The controller manages data transfer between memory and a peripheral under its control, thus bypassing the MPU
- ✓ It introduces two new signals **HOLD (pin 39)** and **HLDA (pin 38) (Hold acknowledge)**



HOLD

- ✓ This is an active high **input** signal
- ✓ Pin number 39
- ✓ The processor relinquishes (gives up) the buses in the following machine cycle once the MPU receives the HOLD request
- ✓ All buses are tri-stated and HLDA (Hold Acknowledge) signal is sent out
- ✓ MPU regains the control of the buses after HOLD goes low



HLDA

- ✓ This is an active high **output** signal
- ✓ Pin number 38
- ✓ It indicates that the MPU is giving up the control of the buses



DMA Controller Essentials

The DMA controller should have

- I. A data bus
- II. An address bus
- III. Read/Write control signals, and
- IV. Control signals to disable its role as a peripheral and to enable its role as a peripheral

(Note that DMA controller is a processor capable only of copying data at high speed from one location to another location)



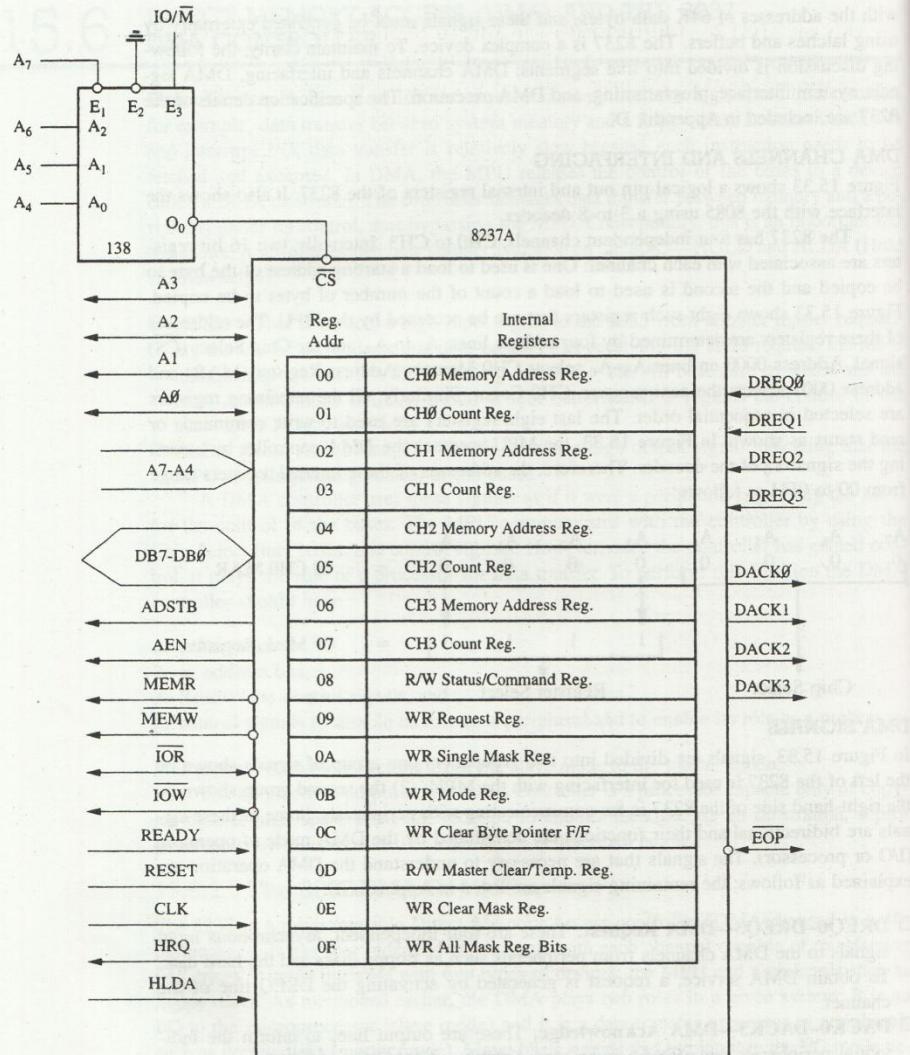


FIGURE 15.33
8237A—DMA Controller with Internal Registers

8237 DMA CONTROLLER Block Diagram

Features

- ✓ 8237 is a programmable Direct Memory Access controller (DMA) housed in a 40-pin package
- ✓ It has four independent channels with each channel capable of transferring 64K bytes
- ✓ It must interface with MPU and a peripheral device
- ✓ It is an I/O device to MPU
- ✓ It is a data transfer processor to peripheral device
- ✓ Many of its signals that are input in the I/O mode become outputs in the processor mode



Description

- ✓ The block diagram shows a logical pin out and internal registers of the 8237. It also shows the interface with the 8085 using a 3-to-8 decoder
- ✓ 8237 has four independent channels CH0-CH3. Two 16-bit registers are internally associated with each channel
- ✓ These registers are determined by A3-A0 and the chip select line (CS)
- ✓ **The 8237 signals are divided into two groups:**
 - 1) signals on **left** (used to communicate with MPU)
 - 2) signals on **right** (used to communicate with peripheral)
- ✓ Some of these signals are bidirectional and are determined by the DMA mode of operation (I/O or processor mode)



DMA Signals

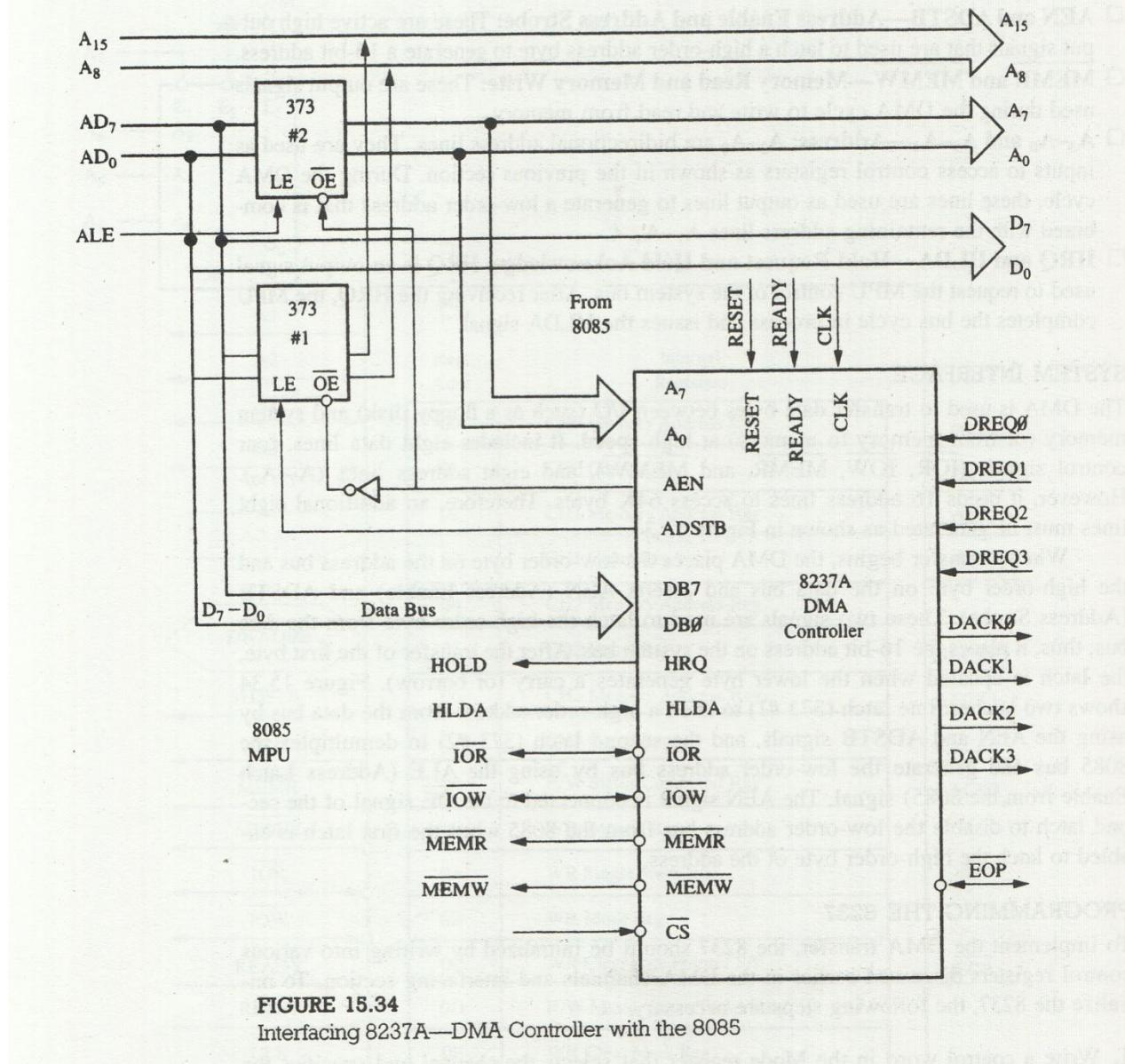
- ✓ To obtain DMA service , a request is generated by activating the **DREQ** line of the channel
- ✓ **DACK** are output lines to inform the individual peripherals that DMA is granted. DREQ and DACK are eqvt to handshake signals in I/O devices
- ✓ **AEN** and **ADSTB**-Address Enable and Address Strobe are used to latch a high-order byte to generate a 16-bit address
- ✓ After receiving the **HRQ** (Hold request), the MPU completes the bus cycle in process and issues the **HLDA** (Hold Acknowledgement) signal



System Interface

- ✓ When a transfer begins, the DMA places the low-order byte on the address bus and high-order byte on the data bus
- ✓ Then 8237 asserts AEN (Address Enable) and ADSTB (Address Strobe)
- ✓ These two signals are used to latch the high-order byte from the data bus and 8237 places the 16-bit address on the system bus





NEXT CLASS

Serial Communication (Chapter 16)



THANK YOU



Data Communication

(Chapter 16)

(Minimum 7 marks, Maximum 13 Marks)



Parallel Vs Serial Communication

Parallel

- i. Communication through AD0-AD7 pins in 8085
- ii. 8085 transfers eight bits of data simultaneously over eight data lines
- iii. Parallel communication over a very long distance can become very expensive

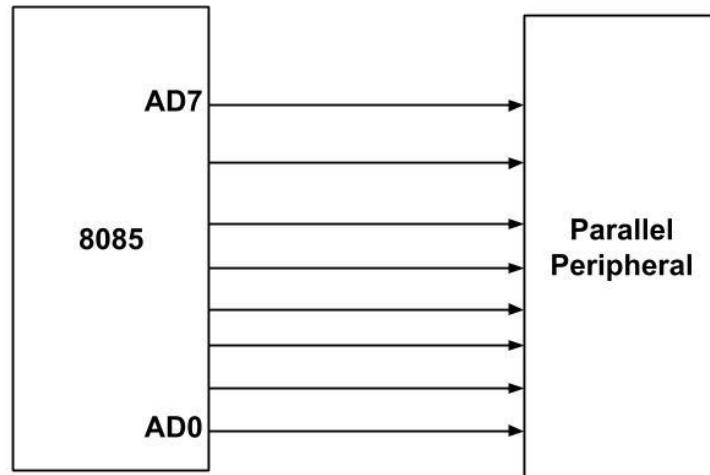
Serial

- i. Communication through SOD and SID pins in 8085
- ii. One bit at a time is transferred over a single line
- iii. Parallel-to-Serial and Serial-to-Parallel conversion needed at transmission and Reception side

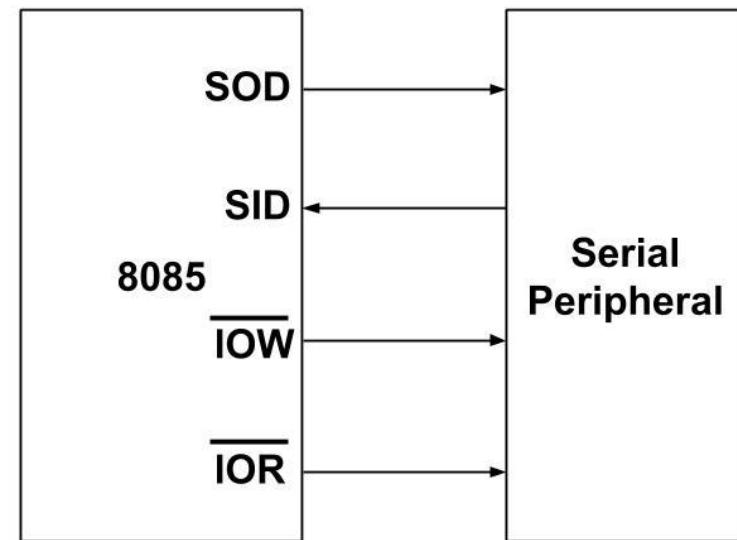


Block Diagram

Parallel



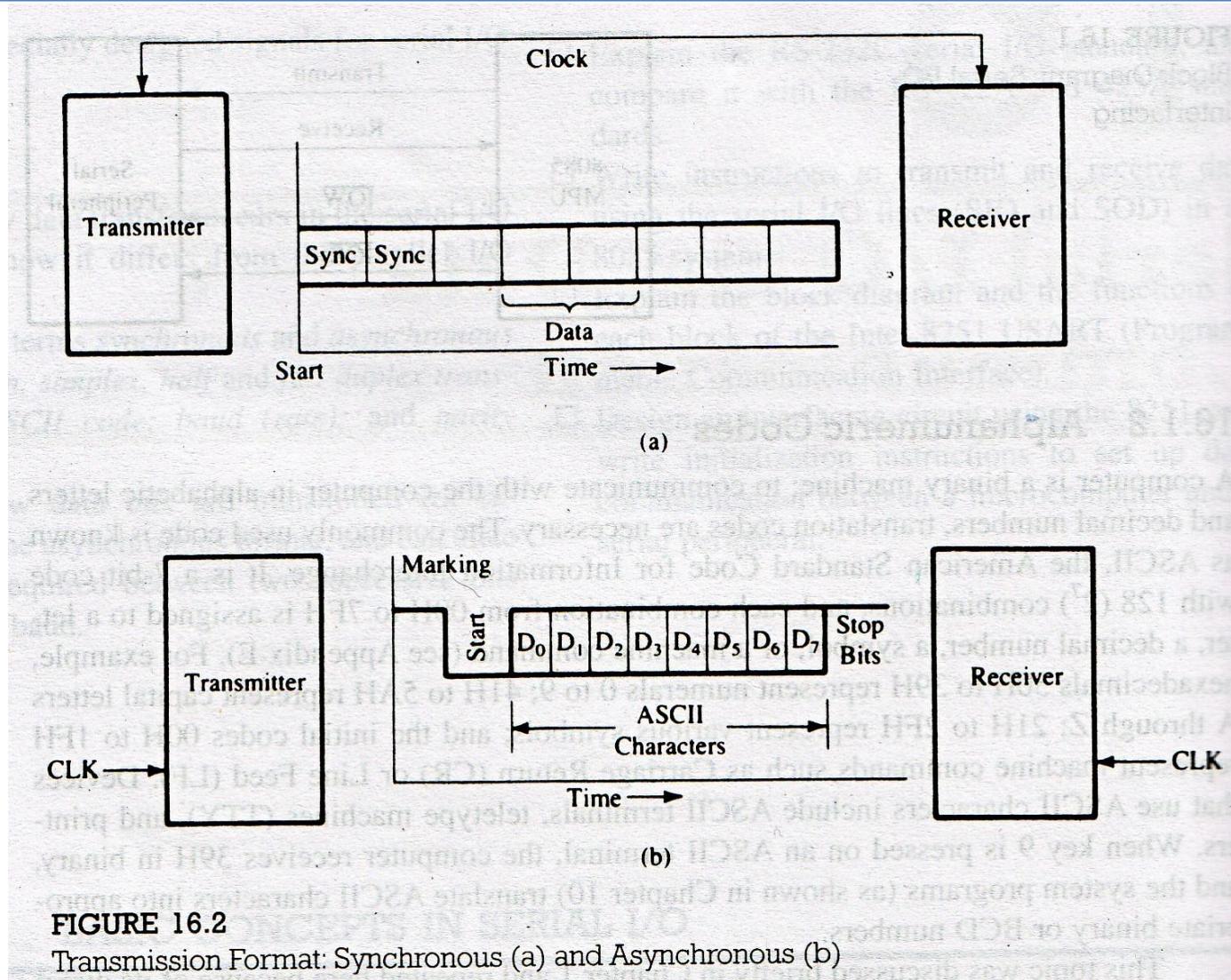
Serial



- ✓ The µP identifies the peripheral through a port address and enables it using Read or Write control signal
- ✓ The primary difference between Parallel I/O and Serial I/O is in the number of lines used for data transfer
- ✓ The Parallel I/O uses the entire data bus
- ✓ The Serial I/O uses one data line



Transmission Format



Synchronous Vs Asynchronous

Synchronous

- The transmitter is synchronized with the receiver on the same frequency
- A block of characters is transmitted along with the synchronization information
- Generally used for high-speed transmission (more than 20k bits/seconds)

Asynchronous

- The transmitter is not synchronized with the receiver by the same master clock.
- Asynchronous format is character-oriented. Each character carries the information of the start and the stop bit
- Generally used for low-speed transmission (less than 20k bits/seconds)



Standards in Serial I/O

- ✓ A standard include such items as assignment of pin positions for signals, voltage levels, speed of data transfer, length of cables, and mechanical specifications
- ✓ In Serial I/O, data can be transmitted as either current or voltage. When data are transmitted as voltage, the commonly used standard is known as RS-232C
- ✓ It is defined in reference to **Data Terminal Equipment (DTE)** and **Data Communication Equipment (DCE)**
- ✓ The voltage levels are **not compatible** with TTL Logic levels
- ✓ The rate of data transmission in RS-232C is restricted to a maximum of 20kbaud and a distance of 50ft
- ✓ For high-speed data transmission, two new standards **RS-422A** and **RS-423A** have been developed



RS-232 Standard

- 25 pin connectors
- Data signals, control signals, timing signals, and grounds (4 group)
- Logic level 0 => +3 V to +15 V
- Logic level 1 => -3 V to -15 V
- Normal level used is **± 12V**
- This is true negative level
- Other signals are compatible with TTL logic



- Voltage translators, called line drivers and line receivers are required to interface TTL Logic with the RS-232 signals because of incompatibility of voltage levels in these two standards
- The line driver, **MC1488** converts **logic 1** into approx **-9 V** and **logic 0** into approx **+9V**
- Line receiver **MC1489** perform the reverse operation done by **MC1488**
- Minimum interface between a computer and a peripheral requires three signals: pins 2, 3, and 7

Reasons for using higher voltage levels:

- i. RS-23S standard came into existence before TTL was defined.
- ii. This standard provides a higher level of noise margin-from -3 V to +3 V

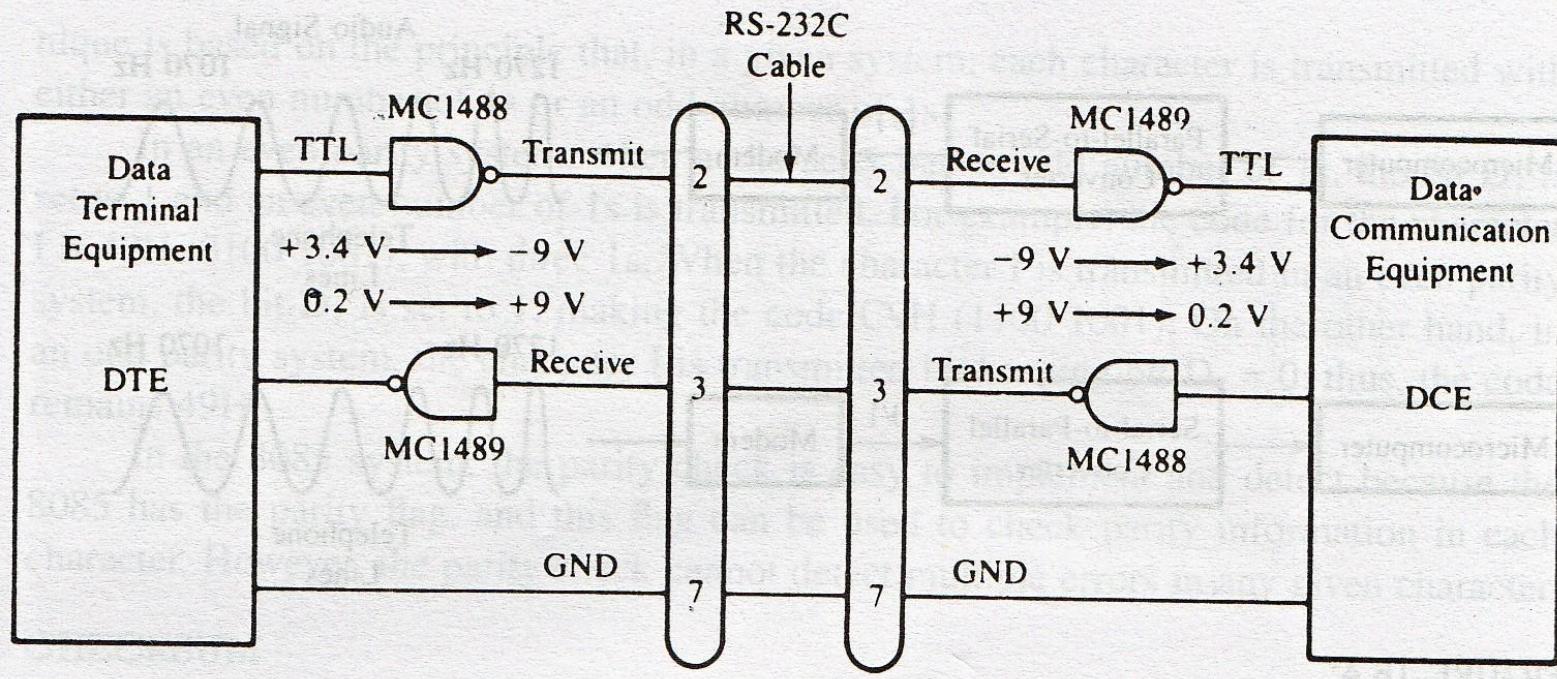


RS232 Signal Definitions and Pin Assignments

Signals	Pins	Signals
Secondary Transmitted Data	14	Protective Ground
Transmission Signal Element Timing (DCE Source)	15	Transmitted Data ($T \times D$) → DCE
Secondary Received Data	16	Received Data ($R \times D$) → DTE
Receiver Signal Element Timing (DCE Source)	17	Request to Send (RTS) → DCE
Unassigned	18	Clear to Send (CTS) → DTE
Secondary Request to Send	19	Data Set Ready (DSR) → DTE
DCE ← Data Terminal Ready (DTR)	20	Signal Ground
Signal Quality Detector	21	Received Line Signal Detector
Ring Indicator	22	(Reserved for Data Set Testing)
Data Signal Rate Selector (DTE/DCE Source)	23	(Reserved for Data Set Testing)
Transmit Signal Element Timing (DTE Source)	24	Unassigned
Unassigned	25	Sec. Rec'd. Line Sig. Detector
		Sec. Clear to Send

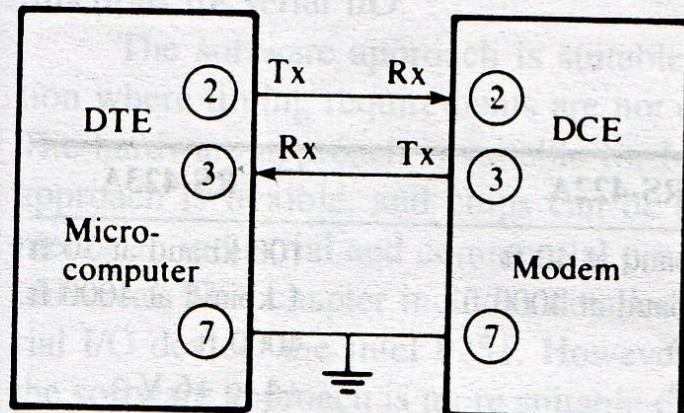


Minimum Configuration of RS232 Signals and Voltage Level

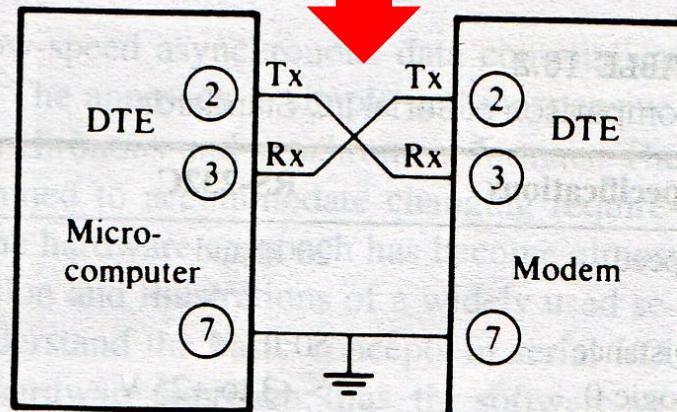


RS-232C Connections

Null-Modem Connection



(a)



(b)

FIGURE 16.6

RS-232C Connections: DTE to DCE (a) and DTE to DTE (b)

TABLE 16.1

RS-232C Signals Used with Handshake Data Communication

Pin No.	Signals ^a	Functions	
2	Transmitted Data	TxD	Output; transmits data from DTE to DCE
3	Received Data	RxD	Input; DTE receives data from DCE
4	Request to Send	RTS	General-purpose output from DTE
5	Clear to Send	CTS	General-purpose input to DTE; can be used as a handshake signal
6	Data Set Ready	DSR	General-purpose input to DTE; can be used to indicate that DCE is ready
7	Signal Ground	GND	Common reference between DTE and DCE
8	Data Carrier Detect	DCD	Generally used by DTE to disable data reception
20	Data Terminal Ready	DTR	Output; generally used to indicate that DTE is ready

^aSignals are referenced to DTE.**TABLE 16.2**

Comparison of Serial I/O Standards

Specifications	RS-232C	RS-422A	RS-423A
Speed	20 kbaud	10 Mbaud at 40 ft 100 kbaud at 4000 ft	100 kbaud at 30 ft 1 kbaud at 4000 ft
Distance	50 ft	4000 ft	4000 ft
Logic 0	> +3 to +25 V	B > A ^a	+4 to +6 V
Logic 1	< -3 to -25 V	B < A	-4 to -6 V
Receiver Input Voltage	± 15 V	± 7 V	± 12 V

^aB and A are differential input to the op amp.

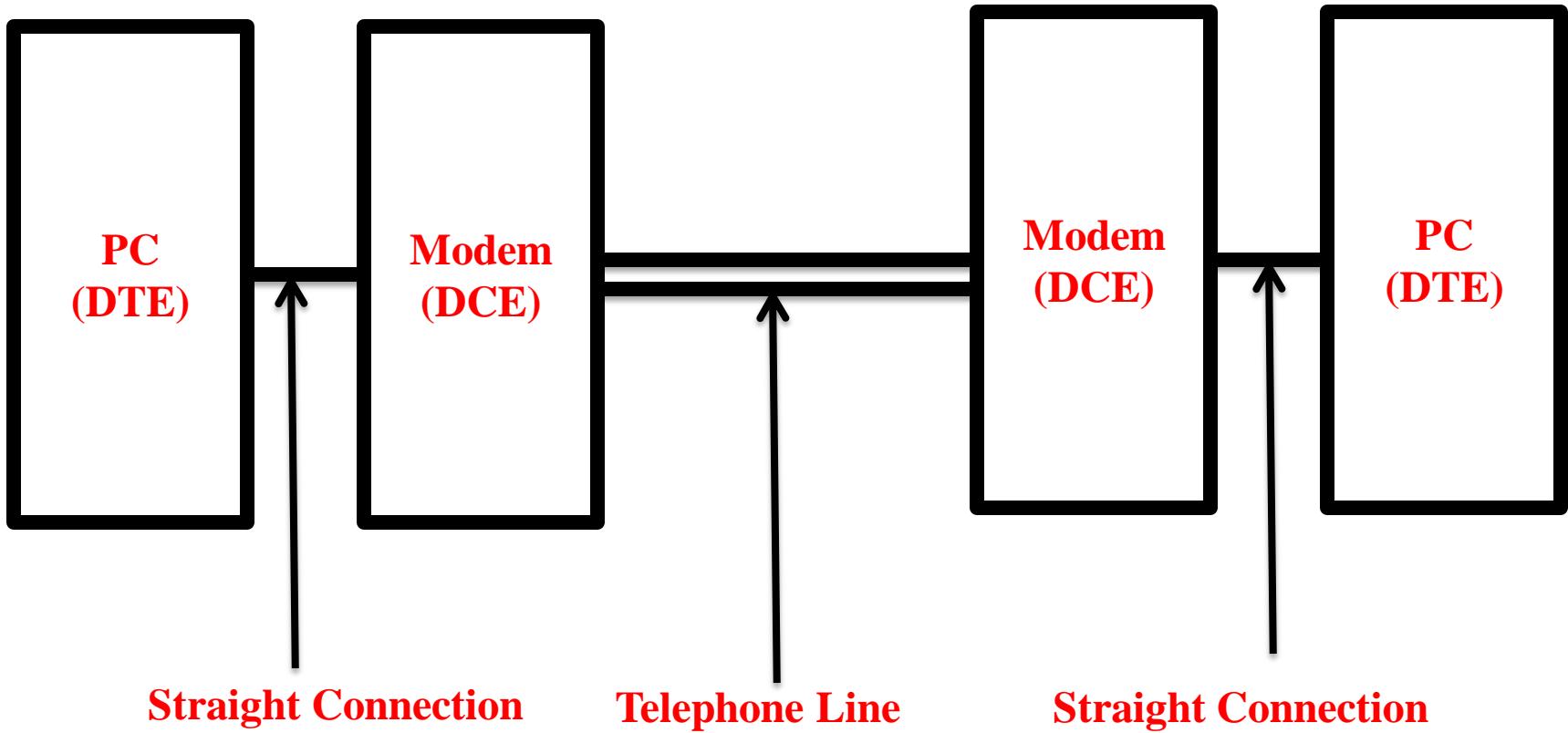
Typically, data transmission with a handshake requires eight lines

For High-Speed Communication , the standards RS-422A and RS-423 A are used



6 b) Two computers are to be connected with each other via modems using RS-232 standards, explain the connections? Also, show the connections for null modem.(Fall 2014, 8 marks)

Solution:



The micro computer or PC acts like a DTE (Data Terminal Equipment) and a modem acts like a DCE (Data Communication Equipment). The RS-232 standard is defined with respect to DTE and DCE devices. DTE transmits data through pin number 2 while DCE receives data via pin number 2 in a RS-232 standard. Hence the connection between a PC (DTE) and a Modem (DCE) is a straight through cable. While the direct connection between a PC and PC without modem is a crossover cable. The pin number 2 is connected to a pin number 3 because DTE sends data through pin 2 and it (DTE) receives data through pin number 3. This connection is known as **Null-Modem** connection. These two connection between DTE and DCE, DTE and DTE is shown in the Figure 2.



Null-Modem Connection

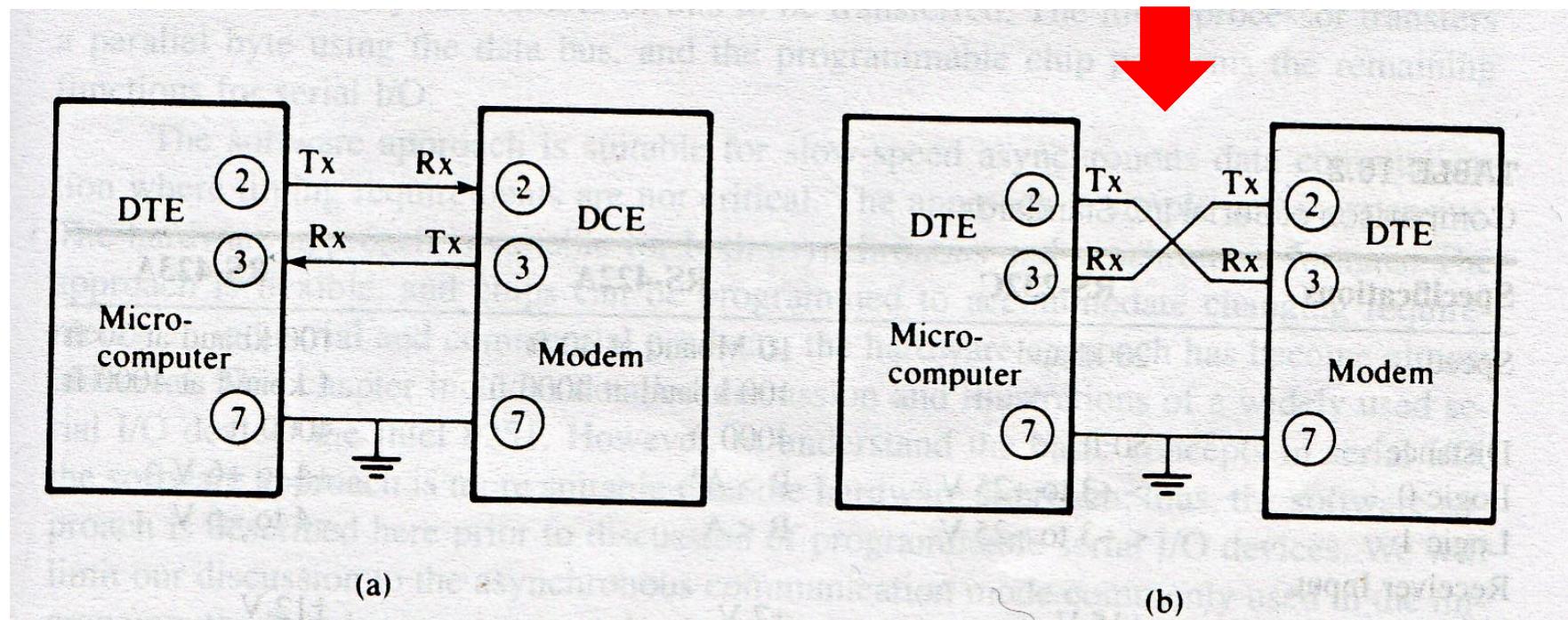


Figure 2: RS-232 Connections : DTE to DCE (a) and DTE to DTE (b)
In Text book, it is referenced at Figure 16.6

Serial I/O Lines: SOD and SID

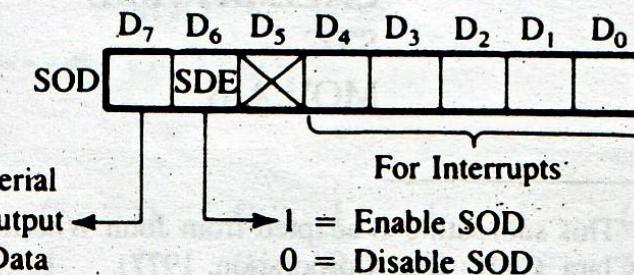
- ✓ The 8085 has two pins specifically designed for software-controlled Serial I/O. One is called **SOD** (Serial Output Data), and the other is called **SID** (Serial Input Data)
- ✓ Data transfer is controlled through two instructions: **SIM** and **RIM**. Instructions SIM and RIM are used for two different processes: interrupt and Serial I/O



SOD (Serial Output Data)

The instruction SIM is necessary to output data serially from the SOD line. It can be interpreted for serial output as shown in Figure 16.9

FIGURE 16.9.
Interpretation of Accumulator
Contents by the SIM Instruction



Example 1

MVI A,80H

; set D7 in the accumulator

RAR

; set D6=1 and bring carry bit to D7

SIM

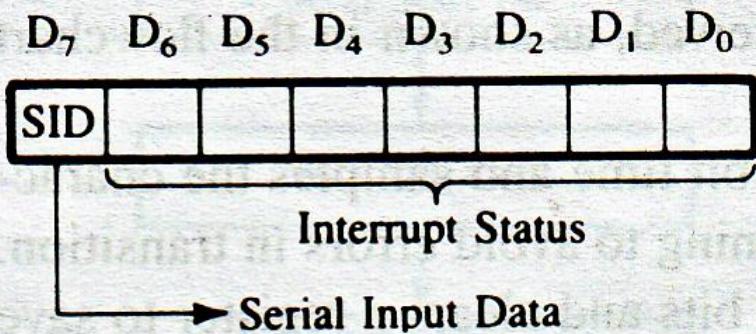
; output D7

HLT

In this set of instructions, the serial output lines is enabled by rotating 1 into bit position D6; the instruction SIM outputs the carry bit through bit position D7



SID(Serial Input Data)



:Read the SID line
and place the bit
in the accumulator
at D₇

FIGURE 16.10

Interpretation of Accumulator Contents After the RIM Instruction

Description

- ✓ Instruction RIM is used to input serial data through the SID line. Instruction RIM can be interpreted for serial I/O as in Figure 16.10 (Text Book, 6th edition)
- ✓ In the context of serial I/O, instruction RIM is similar to instruction IN, except RIM reads only one bit and places it in the accumulator at bit position D7
- ✓ Essentially, SID is a 1-bit input port and SOD is a 1-bit output port. Similarly, instruction RIM is equivalent to a 1-bit IN instruction and instruction SIM is equivalent to a conditional 1-bit OUT instruction



BISYNC

- ✓ It stands for Binary Synchronous Communication Protocol
- ✓ It is also referred as a byte-controlled protocol (BCP), because specified ASCII or EBCDIC character (bytes) are used to indicate the start of a message and to handshake between the transmitter and the receiver.
- ✓ The general message format for BISYNC is shown in the diagram (**Ref 14.33,D V Hall**)



NEXT CLASS

8086



THANK YOU



16-bit Microprocessor and Programming



INTEL 8086

- ✓ The Intel 8086/8088 is a 16-bit microprocessor housed in a 40 pin package and capable of addressing one megabyte of memory. Various versions of this chip can operate with clock frequencies from 5MHz to 10MHz
- ✓ The 8088 is functionally similar to the 8086, except that it has an 8-bit external data bus. Its internal architecture and instruction set are identical with those of the 8086
- ✓ The 8088 can be viewed as an 8-bit μP with the execution power of a 16-bit μP



Objectives

- I. Increase memory addressing capability
- II. Increase execution speed
- III. Facilitate programming in high-level languages
- IV. Provide a powerful instruction set
- V. Function in a multi-processor environment



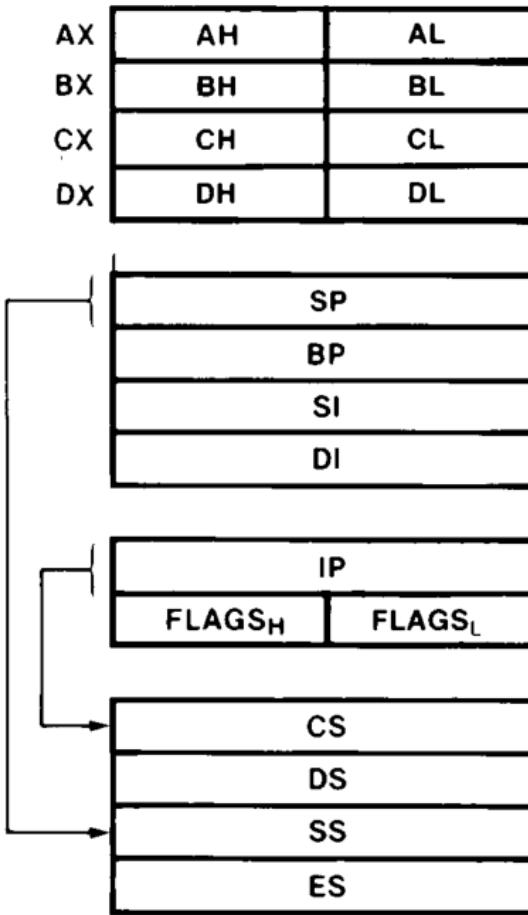
8087

- ✓ 16-bit data lines
- ✓ 20-bit address lines
- ✓ 7 register stack
- ✓ 40-pins
- ✓ Math Co-Processor

(When combined with the 8086/8088 µP, the 8087 dramatically increases the processing speed of computer applications which utilize mathematical operations such as CAM, numeric controllers, CAD or graphics)



8086 programming Model



ACCUMULATOR

BASE

COUNT

DATA

STACK POINTER

BASE POINTER

SOURCE INDEX

DESTINATION INDEX

INSTRUCTION POINTER

STATUS FLAGS

CODE SEGMENT

DATA SEGMENT

STACK SEGMENT

EXTRA SEGMENT

Reference: Intel Data Sheet
Page number: 12



Multipurpose Registers:

AX : Accumulator

BX : Base

CX : Count

DX : Data

BP : Base Pointer

SI : Source Index

DI : Destination Index

Special-Purpose Registers:

FLAGS

IP : Instruction Pointer

SP : Stack Pointer

Segment Registers:

CS : Code Segment

DS : Data Segment

ES : Extra Segment

SS : Stack Segment



Multipurpose Register

AX(Accumulator):

- ✓ AX is used as 16-bit accumulator, with lower 8-bits of AX designated as AL and higher 8-bits as AH. AL can be used as an 8-bit accumulator for 8-bit operation. The accumulator is used for instructions such as multiplication, division, and some of the adjustment instructions

Note:

- ✓ Usually the letters “L” and “H” specify the lower and higher bytes of a particular register
- ✓ Letter “X” is used to specify the complete 16-bit register



BX (Base Index Register):

- ✓ The BX register sometimes holds the offset address of a location in the memory system

CX (Count Register):

- ✓ It is a general-purpose register that holds the count for various instructions eg: LOOP instruction

DX(Data Register):

- ✓ It is a general-purpose register that holds a part of the result from a multiplication or part of the dividend before a division

BP(Base Pointer Register):

- ✓ It points to a memory location for memory data transfer

DI(Destination Index Register):

- ✓ Addresses string destination data for the string operation

SI(Source Index Register):

- ✓ Addresses source string data for the string instructions



Special-Purpose Registers

The special-purpose registers include IP, SP, and FLAGS

IP(Instruction Pointer):

- ✓ IP addresses the next instruction in a section of memory defined as a code segment. This register is 16-bit long
- ✓ It is used by 8086 to find the next sequential instruction in a program located within the code segment

SP (Stack Pointer Register):

- ✓ SP addresses an area of memory called the Stack

FLAGS:

- ✓ FLAGS indicate the condition of the 8086 and control its operation



Segment Registers

Segment registers generate memory addresses when combined with other registers in 8086

CS (Code Segment Register):

- ✓ The code segment is a section of memory that holds the code (programs and procedures) used by the 8086. The code segment register defines the starting address of the section of memory holding code. It defines the start of 64K byte section of memory

DS (Data Segment Register):

- ✓ The data segment is a section of memory that contains most data used by a program

ES (Extra Segment Register):

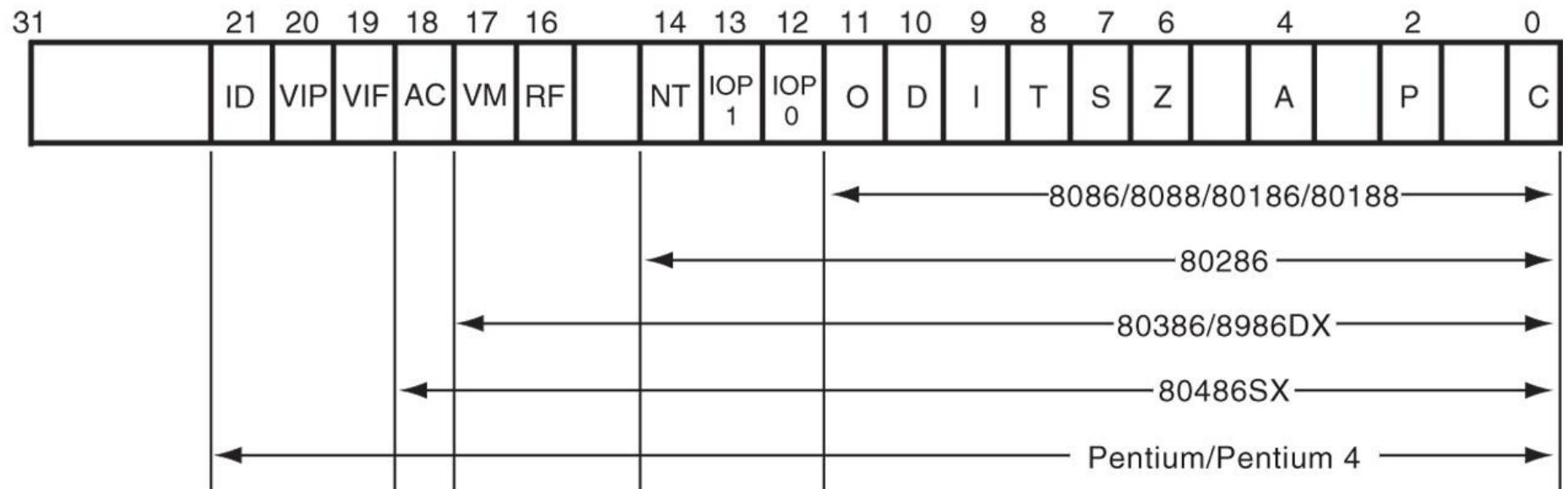
- ✓ The extra segment is an additional data segment that is used by some of the string instructions to hold destination data

SS (Stack Segment Register):

- ✓ The stack segment defines the area of the memory used for the stack. The stack entry point is determined by the stack segment and stack pointer register



FLAGS of 8086



The rightmost **five flags** and the **overflow flag** are changed by most arithmetic and logic operations

C(Carry Flag)

- ✓ Carry holds the carry after addition or the borrow after subtraction

P(Parity Flag)

- ✓ Parity is a logic 0 for odd parity and a logic 1 for even parity

A(Auxiliary Carry)

- ✓ The auxiliary carry holds the carry (half-carry) after addition or the borrow after the subtraction between bit position 3 and 4 of the result

Z(Zero Flag)

- ✓ If Z=1, the result is zero; if Z=0, the result is not zero

S(Sign Flag)

- ✓ If S=1, the sign bit is set and number is negative. If S=0, the sign bit is cleared or number is positive



T(Trap Flag)

- ✓ Is used for debugging feature

I(Interrupt Flag)

- ✓ It controls the operation of INTR (Interrupt Request) input pin.
- ✓ If I=1, the INTR pin is enabled
- ✓ If I=0, the INTR pin is disabled
- ✓ STI=> Set I flag
- ✓ CLI => Clear I flag

D(Direction Flag)

- ✓ The direction flag selects either the increment or decrement mode for the DI and / or SI register during string instructions
- ✓ If D=1, the registers are automatically decremented
- ✓ If D=0, the registers are automatically incremented
- ✓ STD=> Set D flag
- ✓ CLD => Clear D flag

O(Overflow Flag)

- ✓ Overflow occur when signed numbers are added or subtracted. An overflow indicates that the result has exceeded the capacity of the machine
- ✓ For unsigned operations, the overflow flag is ignored

For Example:

$$\begin{array}{r} 7FH \text{ (+127)} \\ +01H \text{ (+01)} \\ \hline 80H \text{ (-128)} \end{array}$$



Internal Block Diagram of 8086

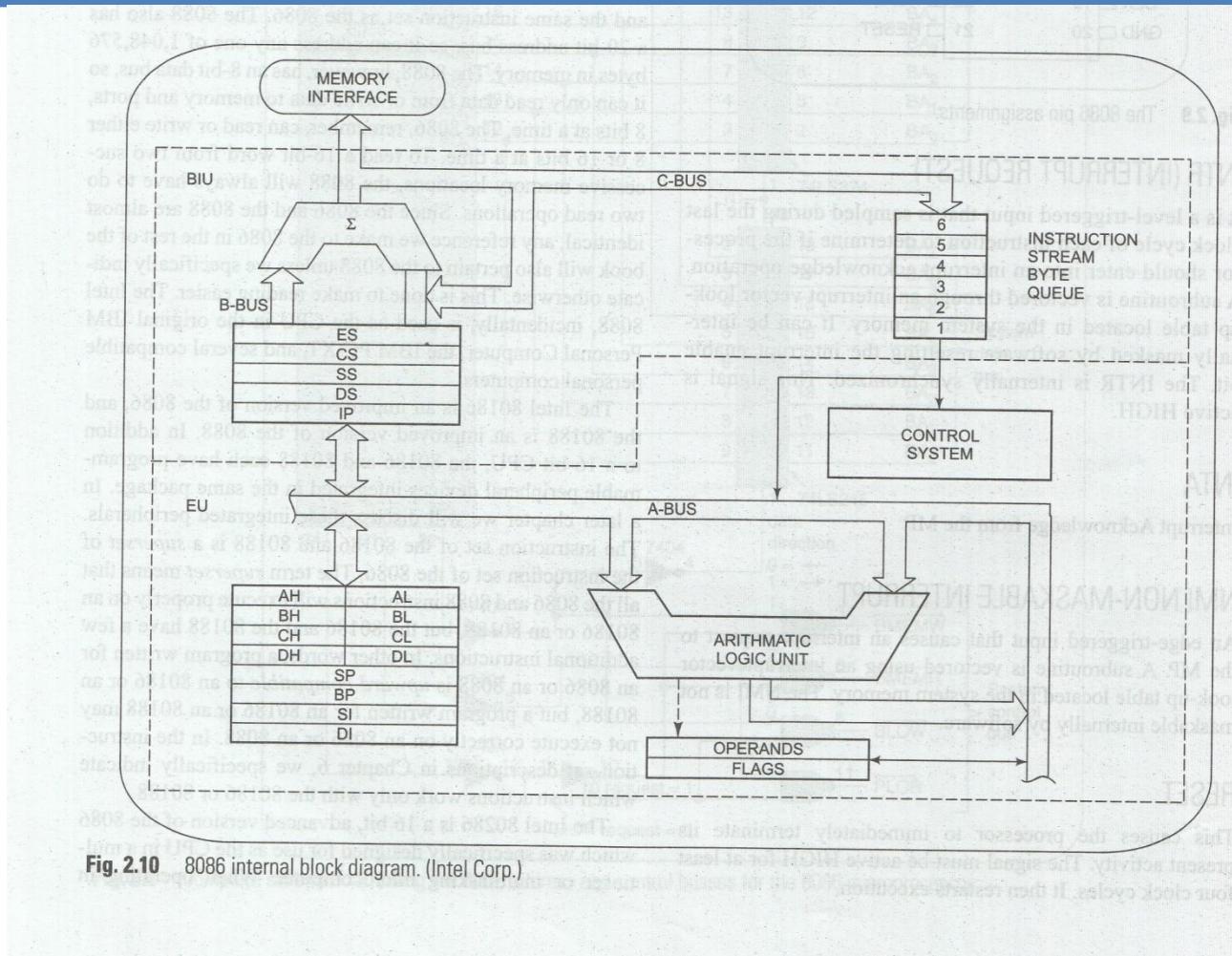


Fig. 2.10 8086 internal block diagram. (Intel Corp.)

BIU and EU

8086 CPU is divided into two independent functional parts, the bus interface unit or BIU , and the execution unit or EU. Dividing the work between these two units speeds up processing

BIU(Bus Interface Unit):

- ✓ The BIU sends out addresses, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory. In other words, the BIU handles all transfer of data and addresses on the buses for the execution

Execution Unit:

- ✓ The execution unit of the 8086 tells the BIU where to fetch instructions or data from, decodes instruction, and execute instructions
- ✓ The EU contains control circuitry which directs internal operations. A decoder in the EU translates instructions fetched from memory into a series of actions which the EU carries out. The EU has a 16-bit arithmetic logic unit which can add, subtract, AND, OR, XOR, increment, decrement, or shift binary number



The BIU Queue

- ✓ While the EU is decoding an instruction or executing an instruction which does not require the uses of the buses, the BIU fetches up to six instruction bytes for the following instructions. The BIU stores these pre-fetched bytes in a first-in-first-out register set called queue.
- ✓ When the EU is ready for its next instruction, it simply reads the instruction bytes for the instruction from the queue in the BIU. This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes
- ✓ Fetching the next instruction while the current instruction executes is called pipelining



Segment and Offsets

- ✓ The 8086 BIU sends out 20-bit addresses, so it can address any of 2^{20} bytes in memory. However, at any given time the 8086 works with only four (64-KB) segments within this 1 MB range
- ✓ Four segment register in the BIU are used to hold the upper 16 bits of the starting addresses of four memory segments that the 8086 is working with at a particular time. The four segment registers are the code segment (**CS**) register, the stack segment (**SS**) register, the extra segment (**ES**) register, and the data segment (**DS**) register

Note:

The first 1MB of memory is called real memory or DOS memory system



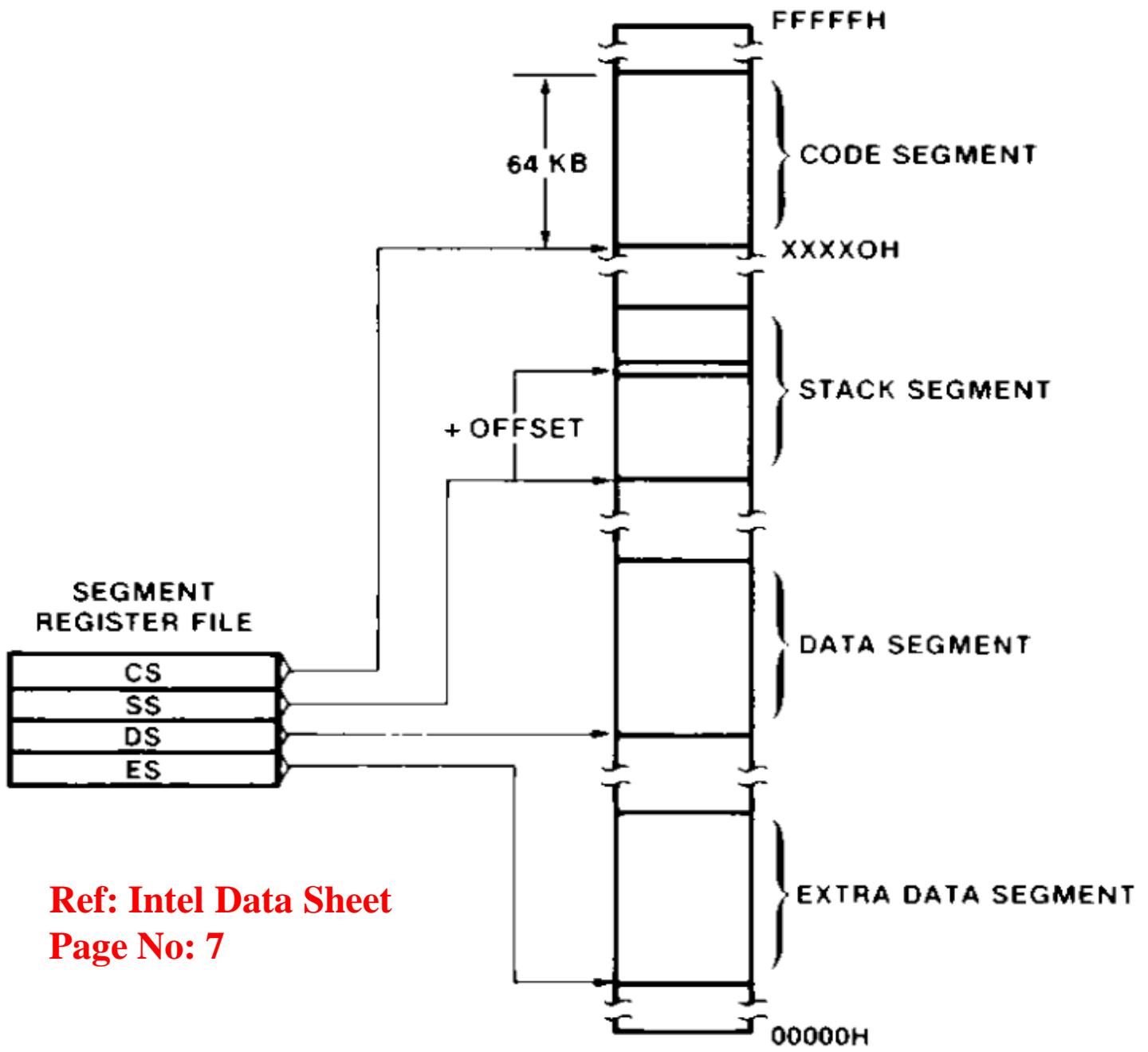
Segment Address:

- ✓ The segment address, located within one of the segment registers, defines the **beginning address** of any 64KB memory segment

Offset Address:

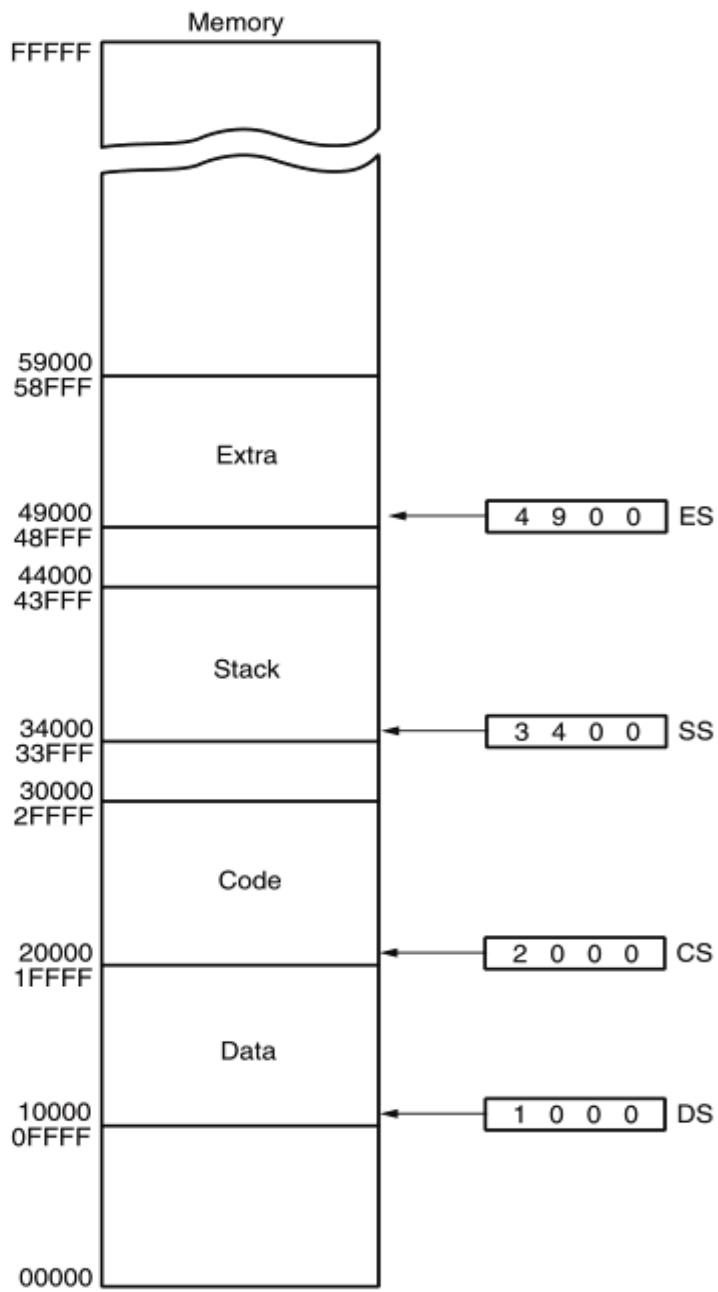
- ✓ The offset address selects any location within the 64KB memory segment. Offset address is sometimes called as a **displacement**. Note that the offset or displacement is the distance above the start of the segment.
- ✓ **Each segment register is internally appended with 0H on its rightmost end.** This forms a 20-bit memory address, allowing it to access the start of a segment. The 8086 must generate 20-bit memory address to access a location within the first 1MB of memory
- ✓ Because a read mode segment of memory is 64KB in length, once the beginning address is known, the ending address is found by adding FFFFH





Ref: Intel Data Sheet
Page No: 7

FIGURE 2–4 A memory system showing the placement of four memory segments.



Default Segment and Offset Registers

- ✓ The 8086 has a set of rules that apply to segments whenever memory is addressed
- ✓ For example, the code segment register is always used with the instruction pointer to address the next instruction in a program. This combination is **CS:IP**
- ✓ Another of the default combination is the stack. Stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (SP) or the base pointer (BP). These combinations are referenced to as **SS:SP** or **SS:BP**



Table: Default 16-bit segment and Offset Combination

Segment	Offset	Special Purpose
CS	IP	Instruction Address
SS	SP or BP	Stack Address
DS	BX,DI,SI,8 or 16-bit number	Data Address
ES	DI for string instructions	String Destination



Pin Description of Intel 8086



		MAX MODE	{ MIN MODE }
GND	1	40	V _{CC}
AD14	2	39	AD15
AD13	3	38	A16/S3
AD12	4	37	A17/S4
AD11	5	36	A18/S5
AD10	6	35	A19/S6
AD9	7	34	BHE/S7
AD8	8	33	MN/MX
AD7	9	32	RD
AD6	10	CPU	31 $\overline{RQ}/\overline{GTO}$ (HOLD)
AD5	11	30	$\overline{RQ}/\overline{GT1}$ (HLDA)
AD4	12	29	LOCK (WR)
AD3	13	28	S2 (M/IO)
AD2	14	27	S1 (DT/R)
AD1	15	26	S0 (DEN)
AD0	16	25	QS0 (ALE)
NMI	17	24	QS1 (INTA)
INTR	18	23	TEST
CLK	19	22	READY
GND	20	21	RESET

**Reference: Intel Data Sheet
or see page 2.9 in Hall, or
page 303 in Barry**



Address Data Bus

AD0-AD15:

- ✓ Pin Number: 2-13,39
- ✓ **AD0-AD7** carry low-order byte of data and **AD8-AD15** carry high-order byte of data. When **AD** lines carry address they are called **A** lines instead of **AD** lines. When **AD** lines carry data, they are called **D** lines. Whenever **ALE** is active (**logic 1**), these lines carry **A0-A15** address lines. When **ALE** is inactive(**logic 0**), these lines carry **D0-D15**



Address/Status

A19/S6-A16-S3:

- ✓ **Pin Numbers: 35-38**
- ✓ The address/Status bits are multiplexed to provide address signals A19-A16 and also status bits S3-S6
- ✓ S6=> is always a logic 0
- ✓ S5=> indicates the condition of the IF flag bit
- ✓ S4 and S3 => show which segment is accessed during the current bus cycle

S4	S3	Function
0	0	Extra Segment
0	1	Stack Segment
1	0	Code or no segment
1	1	Data Segment



Bus High Enable(BHE/S7)

- ✓ **Pin Number: 34**
- ✓ The Bus High Enable pin is used in the 8086 to enable the most-significant data bus bits (D15-D8) during a read or write operation. The state of S7 is always a logic 1

RD :

- ✓ **Pin Number: 32**
- ✓ Whenever the read signal is a logic 0, the data bus is receptive to data from the memory or I/O devices connected to the system

READY:

- ✓ **Pin Number: 22**
- ✓ If it is 0, 8086 enters into wait states and remain idle
- ✓ If it is 1, it has no effect on the operation of the 8086



RESET:

- ✓ **Pin Number: 21**
- ✓ Resets 8086, execution begins at **FFFF0H**

INTR:

- ✓ **Pin Number: 18**
- ✓ Interrupt Request

TEST:

- ✓ **Pin Number: 23**
- ✓ This pin is connected to the 8087 numeric coprocessor

MN/MX:

- ✓ **Pin Number: 33**
- ✓ Selects either the minimum or maximum mode operation for the 8086. If the minimum mode is selected, this pin is connected directly to +5 V



CLK:

- ✓ **Pin Number:19**
- ✓ Clock Input. The clock pulse must have 33% duty cycle

Vcc:

- ✓ **Pin Number:40**

GND:

- ✓ **Pin Number: 1,20**
- ✓ 8086 has two pins labeled GND. Both must be connected to ground for proper operation

NMI:

- ✓ **Pin Number 17**
- ✓ Non-Maskable Interrupt. Similar to INTR, NMI does not check to see whether the IF flag bit is a logic 1. If NMI is activated, this interrupt input uses interrupt vector 2



Minimum Mode

INTA:

- ✓ **Pin Number: 24**
- ✓ Interrupt Acknowledge
- ✓ This interrupt acknowledge signal is a response to the INTR input pin. This pin is normally used to gate the interrupt vector number onto the data bus in response to an interrupt request.

ALE:

- ✓ **Pin Number: 25**
- ✓ Address Latch Enable

DEN:

Pin Number: 26

- ✓ Data bus enable activates external data bus buffer



DT/R :

- ✓ **Pin Number: 27**
- ✓ Data Transmit/Receive
- ✓ 1=> 8086 is transmitting data via its data bus
- ✓ 0=> 8086 is receiving data

M/IO :

- ✓ **Pin Number: 28**
- ✓ 0=> 8086 data bus contains I/O address
- ✓ 1=> 8086 data bus contains memory address
- ✓ Note: In 8088, **IO/M** pin is used instead of **M/IO**

WR,HLD_A, and HOLD

- ✓ **Pin Number: 29,31, and 30**
- ✓ Their function similar to that in 8085



Maximum Mode

In order to achieve maximum mode for use with external coprocessor, **MN/MX** pin is connected to ground

QS1, QS0 (Queue Status)

- ✓ **Pin Number: 24,25**
- ✓ These pins provide status to allow external tracking of the internal 8086 instruction queue.
- ✓ They are used by numeric coprocessor 8087

QS1	QS2	Function
0	0	Queue is idle
0	1	First byte of Opcode
1	0	Queue is empty
1	1	Subsequent byte of Opcode



S₂,S₁,S₀ :

- ✓ **Pin Number: 26-28**
- ✓ Status bits
- ✓ The status bits indicate the function of the current cycle. These signals are normally decoded by the 8288 bus controller

S ₂	S ₁	S ₀	Function
0	0	0	Interrupt Ack
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

Please
see
Page 307
in Barry



LOCK:

- ✓ **Pin Number: 29**
- ✓ The LOCK output is used to lock peripherals off the system. In a multi-processor system, the other devices are informed through this signal that they should not issue **HOLD** request.

RQ/GT1 and RQ/GT0 :

- ✓ **Pin Number: 30,31**
- ✓ The request/grant pin request direct memory access(DMA) during maximum mode operation. These lines are bidirectional and are used to both request and grant a DMA operation



Minimum Vs Maximum Mode

Minimum Mode

1. MN/MX pin is connected to **5V**
2. The mode of operation is similar to that of **8085**
3. Minimum mode operation is the **least expensive way** to operate the 8086
4. All the control signals for memory and I/O are generated by 8086
5. The minimum mode allows the 8085 8-bit peripherals to be used with the 8086 without any special consideration

(Reference: Page 323, Barry) (Imp)

Maximum Mode

1. MN/MX pin is connected to **GND**
2. The maximum mode is unique and designed to be used whenever a **coprocessor** exists in a system
3. Maximum mode was **dropped** from INTEL family beginning with the **80286**
4. Some of the control signals must be **externally generated**
5. This requires the addition of an external bus controller-the 8288 bus controller



Addressing Modes of 8086

MOV AX, BX



Destination



Source



Data Movement is from right to left

Types

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Base-plus-index Addressing
6. Register Relative Addressing
7. Base Relative-plus-index addressing



Register Addressing Mode

- ✓ Register addressing transfers a copy of a byte or word from the source register or contents of a memory location to the destination register or memory location

For Example:

MOV AX,BX



Note:

- ✓ The memory-to-memory transfers are not allowed by any instruction except for the MOVS instruction



Immediate Addressing Mode

- ✓ Immediate addressing transfers the source, an immediate byte or word into the destination or memory location

For Example:

MOV CH,3AH

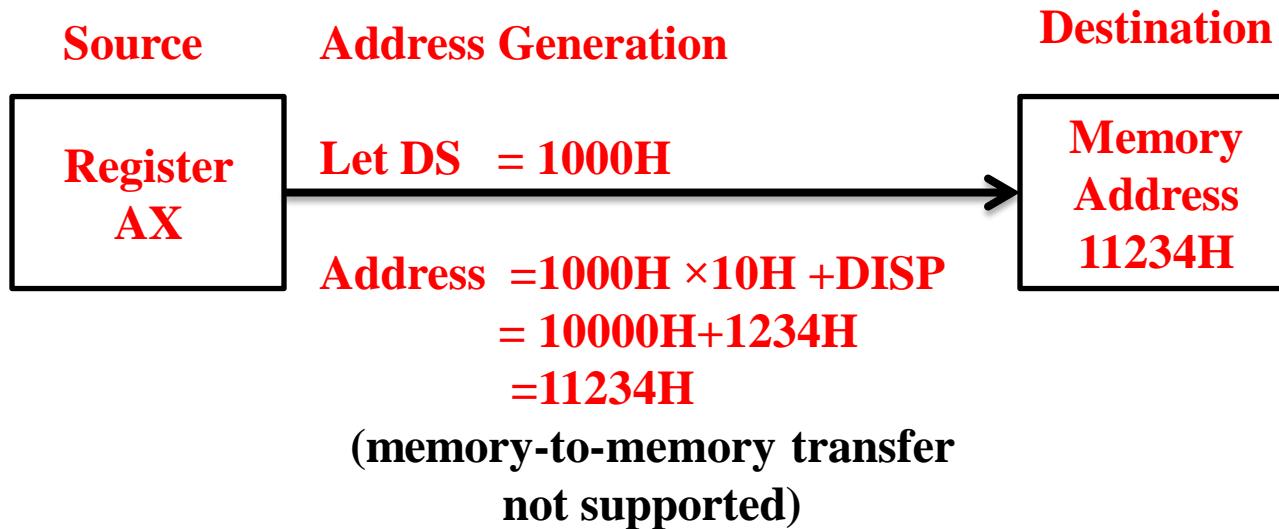


Direct Addressing Addressing Mode

- ✓ Direct addressing moves a byte or word between a memory location and a register

For Example:

MOV [1234H],AX

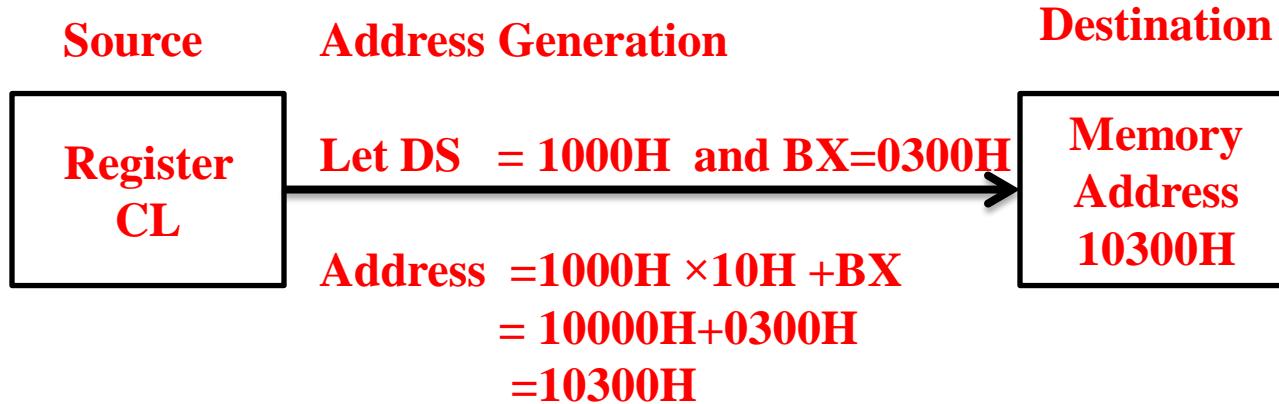


Register Indirect Addressing Mode

- ✓ Register Indirect addressing transfer a byte or word between a register and a memory location addressed by an index or base register. The index and base registers are BP, BX, DI, and SI

For Example:

MOV [BX],CL

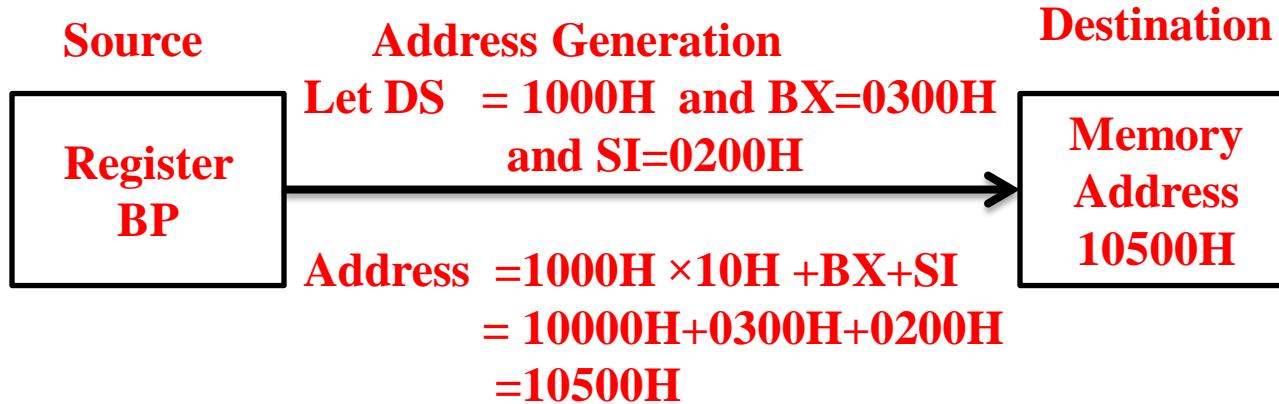


Base-plus-index Addressing Mode

- ✓ Base-plus-index addressing transfers a byte or word between a register and the memory location addressed by a base register (BP or BX) plus an index register (DI or SI)

For Example:

MOV [BX+SI],BP



- ✓ Whenever **BP** addresses memory data, **both the stack segment and BP generate the effective addresses**

MOV CH,[BP+SI]

; copies the byte contents of the **stack segment** memory location addressed by BP plus SI into CH

MOV [BX+SI], SP

; copies SP into **data segment** memory location addressed by BX plus SI

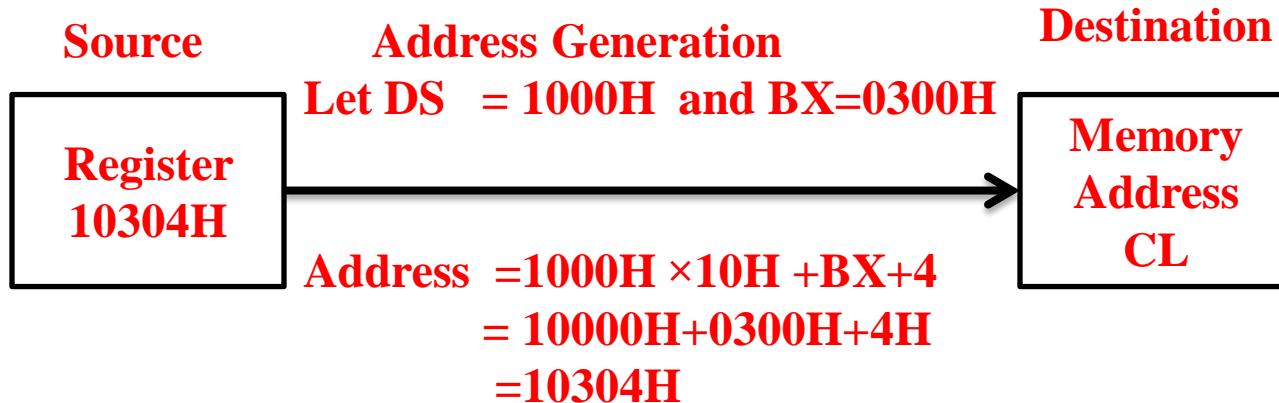


Register Relative Addressing Mode

- ✓ Register relative addressing moves a byte or word between a register and the memory location addressed by an index or base register plus a displacement

For Example:

MOV CL,[BX+4]

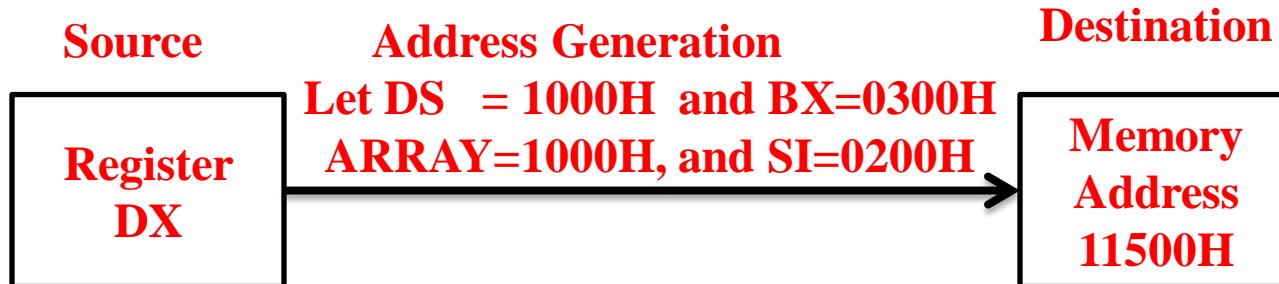


Base Relative-plus-index Addressing Mode

- ✓ Base relative-plus-index addressing transfer a byte or word between a register and the memory location addressed by a base and a index register plus a displacement

For Example:

MOV ARRAY[BX+SI], DX



$$\begin{aligned}\text{Address} &= 1000H \times 10H + \text{ARRAY} + \text{BX} + \text{SI} \\ &= 10000H + 1000H + 0300H + 0200H \\ &= 11500H\end{aligned}$$



Examples of Register Addressing Modes

- i. MOV AL, BL ; copies BL into AL
- ii. MOV CH, CL ; copies CL into CH
- iii. MOV AX, CX ; copies CX into AX
- iv. MOV SP, BP ; copies BP into SP
- v. MOV SI, DI ; copies DI into SI
- vi. MOV ES, DS ; not allowed (**segment-to-segment**)
- vii. MOV BL, DX ; not allowed (**mixed size**)
- viii. MOV **CS**,AX ; not allowed (the code segment register may not be the destination register)

Example 1: Copy the contents of CS to DS

Solution:

```
MOV AX,CS  
MOV DS,AX
```



Examples of Immediate Addressing Modes

- i. MOV BL,44 ; copies 44 decimal into BL
- ii. MOV AX,44H ; copies 0044H into AX
- iii. MOV CL, 0F2H ; “0” appended before F2
- iv. MOV BH,’A’ ; copies ASCII into BH
- v. MOV SI,0 ;copies 0000H into SI

Example 3-2:

```
.MODEL TINY ; choose single segment model
.CODE ; start of code segment
.STARTUP ; start of program
    MOV AX,0 ; place 0000H into AX
    MOV BX,0 ; place 0000H into BX
    MOV CX,0 ; place 0000H into CX
    MOV SI,AX ; place AX into SI
    MOV DI,AX ; place AX into DI
    MOV BP,AX ; place AX into BP
.EXIT ; exit to DOS
.END ; end of program
```



ASSEMBLER DIRECTIVE

- **MODEL TINY**

Statement directs the assembler to assemble the program into a single code segment

- **CODE**

Statement or directive indicates the start of the code segment

- **STARTUP**

Statement indicates the starting instruction in the program

- **EXIT**

Statement causes the program to exit to DOS

END

The END statement indicates the end of the program file



Example of Direct Data Addressing Mode

- i. MOV AL,NUMBER ; copies the byte contents of data segment memory location NUMBER into AL
- ii. MOV AX, COW ; copies the word contents of data segment memory location COW into AX
- iii. MOV ES:[2000H], AL ; copies AL into extra segment memory at offset address 2000H

Note:

Memory location NUMBER, COW are symbolic memory locations

Data addressing with a MOV instruction transfers data between a memory location, located within the data segment and the AL, AX



Examples of Direct Addressing Modes

Example 3-6:

.MODEL SMALL

; choose SMALL model

.DATA

DATA1	DB	10H	; place 10H into DATA 1
DATA2	DB	0	; place 0H into DATA 2
DATA3	DW	0	; place 0H into DATA 3
DATA4	DW	0AAAAAH	; place AAAAH into DATA 4

.CODE

; start of code segment

.STARTUP

MOV AL,DATA1	; copy DATA1 into AL
MOV AH,DATA2	; copy DATA2 into AH
MOV DATA3,AX	; copy AX into DATA3
MOV BX,DATA4	; copy DATA4 into BX

.EXIT

; exit to DOS

END

; end of program



- **DATA**

The data segment begins with a .DATA directive to inform the assembler where the data segment begins

- **MODEL SMALL**

The model size is adjusted from **TINY** to **SMALL** so that data segment can be included. The **SMALL** model allows one data and one code segment

DB and **DW** directives allocate memory locations in the data segment



Examples of Register Indirect Addressing Mode

- ✓ [] symbol denote indirect addressing in assembly language
 - ✓ The data segment is used by default with register addressing mode
 - ✓ If **BP** register addresses memory, the stack segment is used by default
- 1) **MOV CX, [BX]** ; copies word contents of the data segment memory location addressed by BX into CX
 - 2) **MOV [BP], DL** ; copies DL into the stack segment memory location addressed by BP



Example 3-7: Write a program to create a table of information that contains 50 samples taken from memory location 0000:046C

Note:

Location 0000:046C contains a counter in DOS that is maintained by the personal computer's real –time clock



```

.MODEL SMALL
.DATA
    DATAS DW 50 DUP (?)
.CODE
.STARTUP
    MOV AX,0
    MOV ES,AX
    MOV BX,OFFSET DATAS
    MOV CX,50
AGAIN: MOV AX, ES:[046CH]
    MOV [BX],AX
    INC BX
    INC BX
    LOOP AGAIN
.EXIT
END
;
```

; select SMALL model
;start data segment
; setup array of 50 words
;start of code segment
; start program
;copy 0000H to AX
; address segment 0000 with ES
;address DATAS with BX
; load counter with 50
; get clock value
; save clock values in DATAS
; increment BX to next element

; repeat 50 time
; exit to DOS
; end of program listing



OFFSET DIRECTIVE

1) MOV BX, DATAS

; copies the contents of memory location
DATAS into BX

2) MOV BX, **OFFSET** DATAS

; copies the offset address of
“DATAS” into BX

LOOP:

The LOOP instruction decrements the counter CX; if CX is not zero, LOOP causes a jump to memory location AGAIN. If CX becomes zero, no jump occurs and thus sequence of instructions ends.



Examples of Base-Plus-Index Addressing

- ✓ This type of addressing uses one base register (BP or BX) and one index register (DI or SI) to indirectly address memory
 - ✓ The base register often holds the beginning location of a memory array, whereas the index register holds the relative position of an element in the array
 - ✓ Whenever BP addresses memory data, both the Stack Segment register and BP register generate the effective address
- 1) MOV CX, [BX+DI]
; copies the word contents of the data segment memory location addressed by BX plus DI into CX
 - 2) MOV [BX+SI], SP
; copies SP into the data segment memory location addressed by BX plus SI
 - 3) MOV CH,[**BP+SI**]
;copies the contents of the **stack** segment memory location addressed by BP plus SI into CH



Example 3-8: Write a program which moves array element 10H into array element 20H

```
.MODEL SMALL ; select SMALL model
.DATA ; start data segment
    ARRAY DB 16 DUP (?) ;set up an array of 16 bytes
    DB 29H ;element 10H (i.e 17th element in ARRAY)
    DB 20 DUP (?) ; next 20 elements of ARRAY (i.e from element
                    ; number 167th onward upto 37th element)
.CODE ; start code segment
.STARTUP
    MOV BX,OFFSET ARRAY ; address ARRAY
    MOV DI, 10H ; 17th element addressed by DI
    MOV AL,[BX+DI] ; get the element in AL
    MOV DI,20H ; address the 33rd element
    MOV[BX+DI],AL ; save in element 20H
.EXIT ; exit to DOS
.END ; end program
```



Examples of Register Relative Addressing Mode

- i) MOV AX,[DI+100H]
; copies the word contents of the data segment memory location addressed by DI plus 100H into AX
- ii) MOV ARRAY[SI],BL
; copies BL into the data segment memory location addressed by ARRAY plus SI

Note

Example 3-9 and Example 3-8 are same



Examples of Base Relative-Plus Index Addressing Mode

- i) MOV DH,[BX+DI+20H]
; copies the byte contents of the data segment memory location addressed by BX plus DI plus 20H into DH
- ii) MOV AX,FILE[BX+DI]
; copies the word contents of the data segment memory location addressed by the sum of FILE, BX, and DI into AX

Note

This type of addressing mode often addresses a two-dimensional array of memory data
(Please see example 3-10) in page 97)



Stack Memory-Addressing Modes

- ✓ The stack plays an important role in all microprocessors. It holds data temporarily and stores the return addresses used by procedures. The stack memory is an LIFO(Last-In, First-Out) memory, which describes the way that data are stored and removed from the stack
- ✓ Data are placed onto the stack with PUSH instruction and removed with a POP instruction
- ✓ The stack memory is maintained by two registers: the stack pointer (Sp) and the stack segment register (SS)
 - i) **PUSHF** ; copies the flag register to the stack
 - ii) **POPF** ;removes a word from the stack and places it onto the flag register
 - iii) **PUSH AX** ; copies AX register to the stack
 - iv) **POP BX** ; removes a word from the stack and places it into the BX register
 - v) **PUSH DS** ; copies the DS register to the stack
 - vi) **POP CS** ; This instruction is illegal
- ✓ (The reason that data may not be popped from the stack into CS is that this only changes part of the address of the next instruction)
- ✓ (Please see example 3-15, model used is TINY in this program)



Initializing the Stack

In assembly language, a stack segment is set up as in Example 4-1. The first statement identifies the start of the stack segment and the last statement identifies the end of the stack segment. The assembler and linker program places the correct segment address in SS and the length of the segment (top of the stack) into Sp. There is no need to load these registers in your program unless you wish to change the initial values for some reason

Example 4-1:

STACK_SEG

STACK_SEG

SEGMENT STACK

DW 100H DUP (?)

ENDS



Alternative Way

An alternative method for defining the stack segment is used with one of the memory models for the MASM assembler only

Example 4-2:

```
.MODEL      SMALL  
.STACK      200H ; set stack size
```

If the “TINY” memory model is used, the stack size is automatically located at the very end of the segment, which allows for a larger stack area.



Load-Effective Address (LEA)

- ✓ The LEA instruction loads a 16-bit register with the offset address of the data specified by the operand

i) **LEA AX, NUMB ;** loads AX with the offset address of NUMB

The OFFSET directive performs the same function as an LEA instruction

For Example:

ii) **MOV BX, OFFSET LIST**

iii) **LEA BX, LIST**

These two instruction perform the same function



Example 4-3: A short program that loads SI with the address of DATA1 and DI with the address of DATA2. It then exchanges the contents of these memory location

```
.MODEL SMALL ;select small model
.DATA ;start data segment
    DATA1 DW 2000H ;define DATA1
    DATA2 DW 3000H ;define DATA2
.CODE ;start of code segment
.STARTUP ;start program
    LEA SI,DATA1 ; address DATA1 with SI
    MOV DI,OFFSET DATA2; address DATA2 with DI
    MOV BX,[SI]
    MOV CX,[DI]
    MOV [SI],CX
    MOV [DI],BX
.EXIT
END
```



LDS, LES, LSS

These instructions load any 16-bit register with an offset address, and the DS, ES, or SS Segment register

i) LDS BX, [DI]

;loads register BX with 1st two bytes in data segment offset by DI, next two bytes in DS segment register

ii) LES BX, CAT

;loads ES , and BX with 32-bit contents of data segment memory location CAT

iii) LSS SP, MEM => functions in 80386 and above



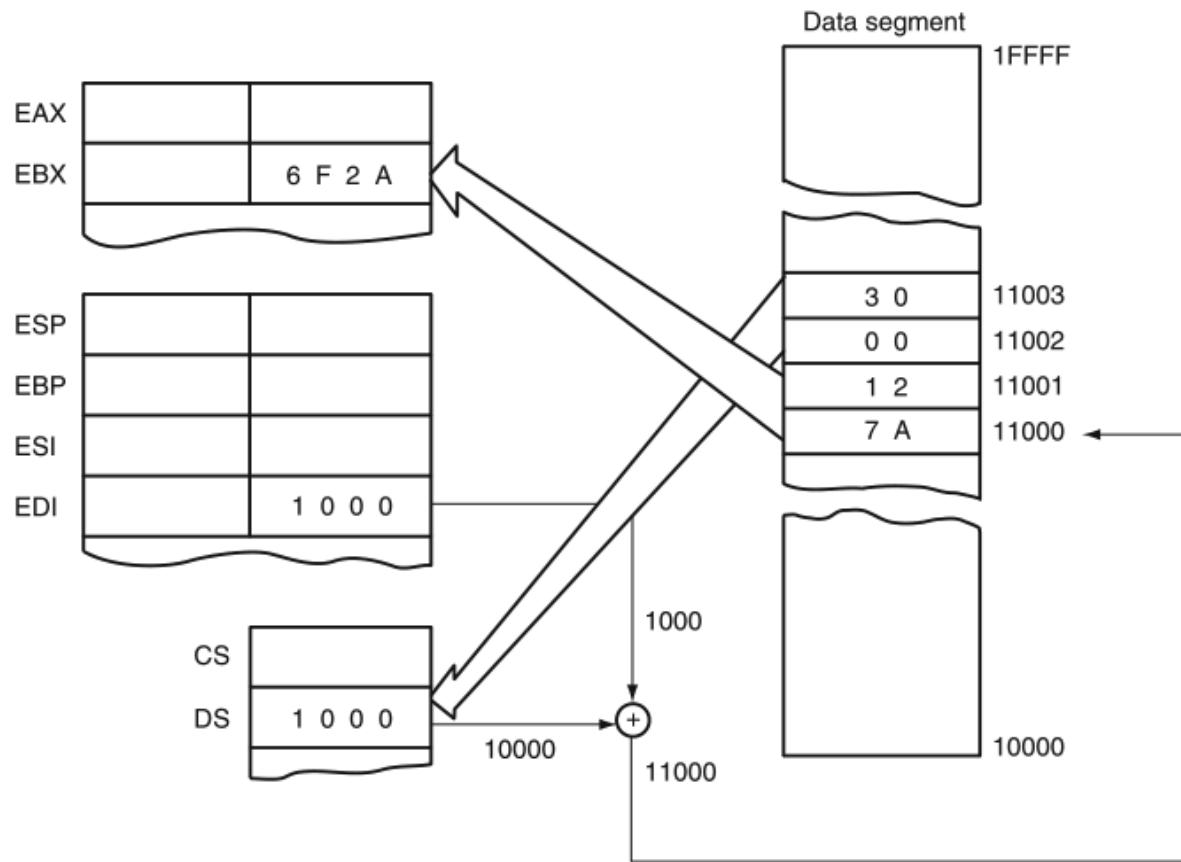


FIGURE 4–17 The LDS BX,[DI] instruction loads register BX from addresses 1100H and 11001H and register DS from locations 11002H and 11003H. This instruction is shown at the point just before DS changes to 3000H and BX changes to 127AH.

String Data Transfer

- i. LODS
- ii. STOS
- iii. MOVS

Before the string instructions are presented, the operation of the D flag-bit (direction), DI, and SI must be understood as they apply to the string instructions



The Direction Flag

- ✓ The direction flag (D, located in the flag register) selects the auto-increment (D=0) or the auto-decrement (D=1) operation for the DI and Si register during the string operation. The direction flag is used only used with the string operations
- ✓ CLD => clears the D flag (D=0)
- ✓ STD => sets the D flag (D=1)
- ✓ Whenever a string instruction transfers a byte, the contents of DI and / or SI are incremented by 2



DI and SI

- ✓ During the execution of a string instruction, memory accesses occur through either or both of DI and Si registers
- ✓ DI=> offset address accesses data in the **Extra Segment** for all string instructions that use it
- ✓ SI => offset address accesses data, by default, in the **Data Segment**



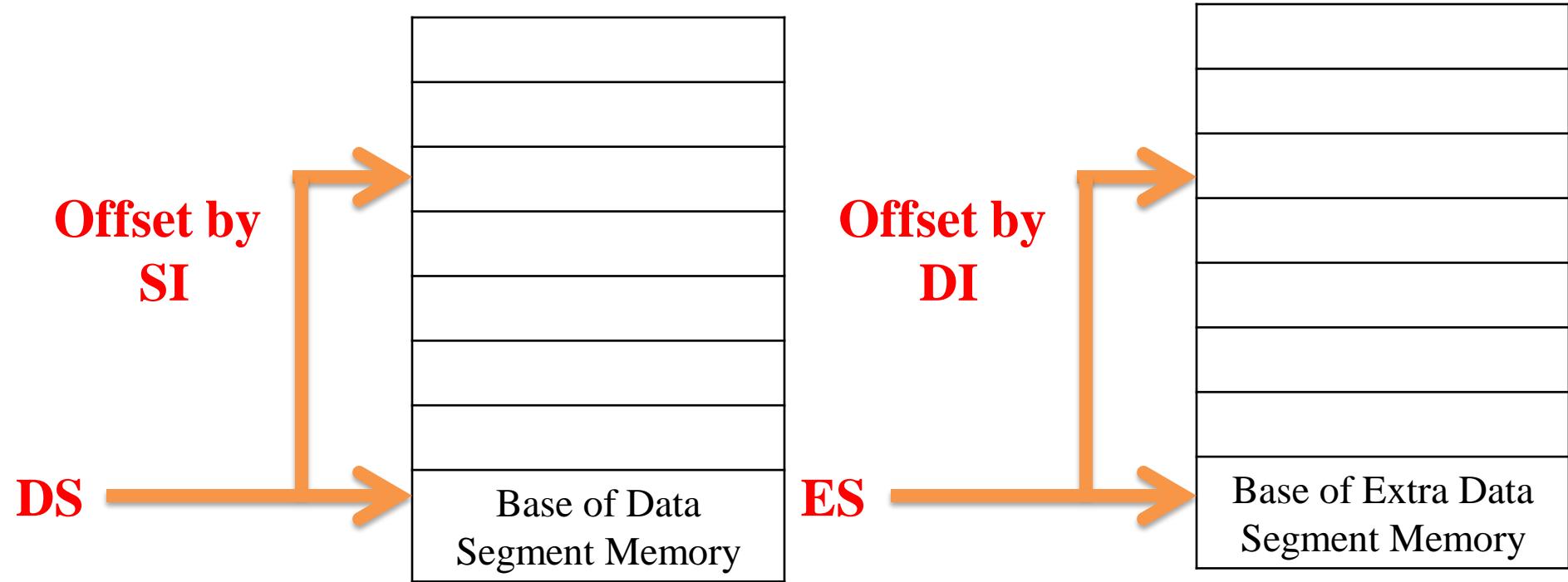
LODS and STOS

- ✓ The LODS instruction **loads** AL, AX with the data stored at the **Data Segment offset** address indexed by the **SI register**
 - i) LODSB AL=DS:[SI]; SI=SI±1
 - ii) LODSW AX=DS:[SI]; SI=SI±2
- ✓ The STOS instruction stores AL, AX at the **Extra Segment** memory location addressed by the **DI register**
- ✓ STOSB=> stores a byte in AL at the Extra Segment memory location addressed by DI
- ✓ STOSW=> stores a word in AX at the Extra Segment memory location addressed by DI

 - i) STOSB ES:[DI]=AL; DI=DI±1
 - ii) STOSW ES:[DI]=AX; DI=DI±2



LODSB VS STOSB



SI and DI automatically increases or decreases by 1 or 2 according to the value of D flag

STOS with REP

- ✓ The repeat prefix (REP) is added to any string data transfer instruction, except the LODS instruction
- ✓ The REP prefix causes CX to decrement by 1 each time the string instruction executes. After CX decrements, the string instruction repeats. If CX reaches a value of “0”, the instruction terminates and the program continues with the next sequential instruction.

For Example:

REP STOSB

If CX=100, then this instruction places the contents of AL in block of memory instead of a single byte of memory



MOVS

- ✓ The MOVS instruction transfers a byte or word from the Data Segment location addressed by SI to the Extra Segment addressed by DI
- ✓ The pointers are incremented or decremented , as indicated by the direction flag

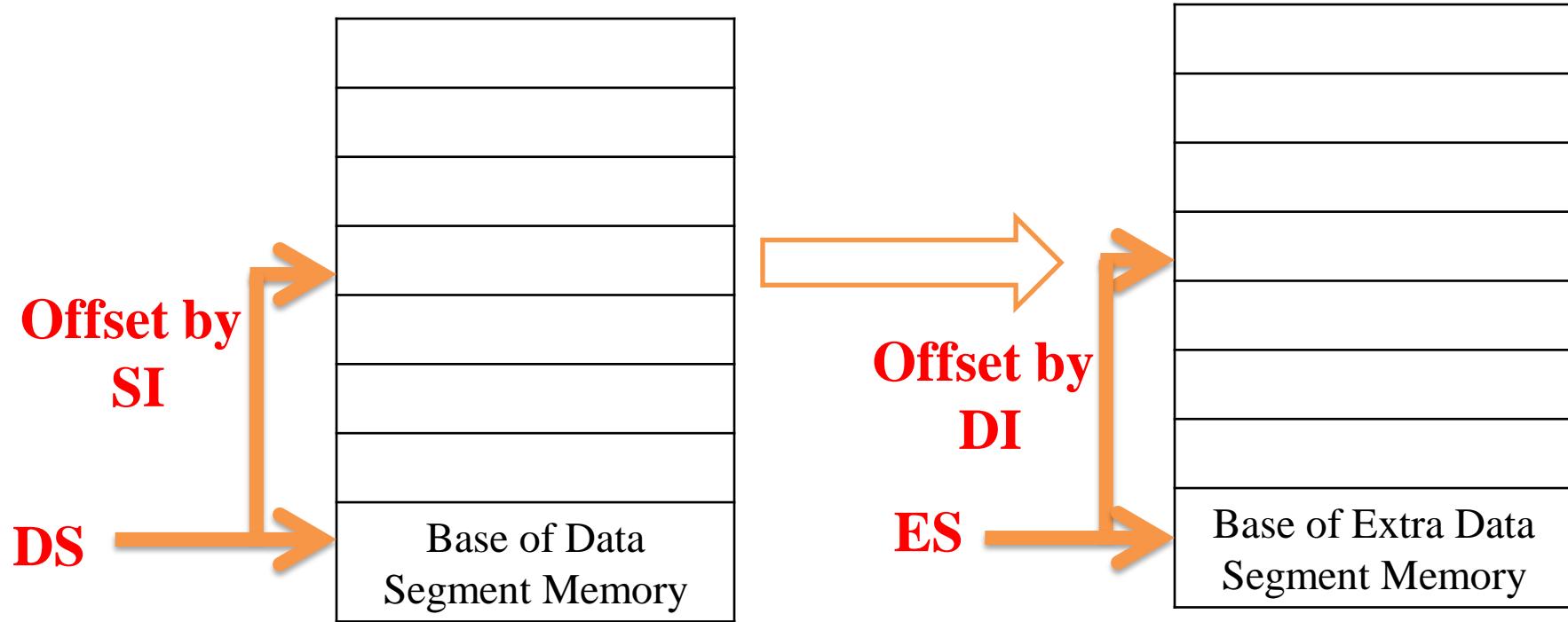
Table 4-14:

MOVSB ;ES:[DI]=DS:[SI] ; DI=DI±1 ; SI=SI±1

MOVSW ;ES:[DI]=DS:[SI] ; DI=DI±2 ; SI=SI±2



MOVS



SI and DI automatically increases or decreases by 1 or 2 according to the value of D flag

String Comparisons

SCAS and CMPS:

- ✓ The SCAS (scan string instruction) compares the AL register with a byte block memory, the AX register with a word block memory
- ✓ The SCAS instruction subtracts memory from AL or AX without affecting either the register or the memory location
- ✓ The Opcode used for byte comparison is SCASB
- ✓ The Opcode used for word comparison is SCASW
- ✓ In all cases, the contents of **the extra segment** memory location addressed by **DI** is compared with AL or AX
- ✓ Like the other string instructions, SCAS instructions use the destination flag (D) to select either auto-increment or auto-decrement operation for DI

Example 5-33

```
MOV DI,OFFSET BLOCK      ; address data
CLD                      ; auto-increment mode
MOV CX,100
XOR AL, AL                ; clear AL
REPNE SCASB              ; repeat while not equal to 00H or until CX=0000H
```



CMPS:

- ✓ The CMPS (Compare String Instruction) always compare two sections of memory data as bytes or words
 - a) CMPSB
 - b) CMPSW
- ✓ The contents of the **data segment** memory location addressed by **SI** are compared with the contents of the **extra segment** memory location addressed by **DI**
- ✓ The CMPS instruction increments or decrements both SI and DI
- ✓ The CMPS instruction is normally used with either the REPE or REPNE prefix

Example 5-35

```
MOV DI,OFFSET LINE    ; address LINE
MOV SI, OFFSET TABLE ;address TABLE
CLD                  ; auto-increment mode
MOV CX,10             ;load counter
REPE CMPSB           ; search
```

Explanation:

When **CX** register becomes “**0**” or an unequal condition exists, the CMPSB instruction stop execution. After the CMPSB instruction ends, the CX register is “0” or the flags indicate an equal condition when the two strings match. If CX is not “0” or the flags indicate a not-equal condition, the strings do not match



Miscellaneous Data Transfer Instructions

XCHG

- ✓ XCHG instruction exchanges the contents of a register with the contents of any other register or memory location

For Example:

XCHG AL,CL	; exchanges the contents of AL with CL
XCHG CX,BP	; exchanges the contents of CX with BP

XLAT

- ✓ The XLAT(translate) instruction converts the contents of the AL register into a number stored in a memory table

Example 4-11

TABLE	DB	3FH,66H,5BH,4FH	;look-up table
	DB	66H,6DH,7DH,27H	
	DB	7FH,6FH	
LOOK:	MOV AL,5	; load AL with 5 (test number)	
	MOV BX,OFFSET TABLE	; address lookup table	
	XLAT	; convert	

After Translation:

AL=6DH



IN and OUT

IN and OUT perform I/O operations. Note that the contents of AL, AX are transferred only between the I/O device and the µP

IN instruction transfers data from an external I/O device into AL or AX

OUT instruction transfers data from AL or AX to an external I/O device

For Example:

- i) IN AL,p8 ; 8-bits are input to AL from I/O port p8
- ii) IN AX,p8 ; 16-bits are input to AX from I/O port p8
- iii) OUT p8,AL ; 8-bits are output to I/O port p8 from AL
- iv) OUT p8,AX ; 16-bits are output to port p8 from AX
- v) OUT 61H, AL ; write to I/O port 61H
- vi) IN AL,0FCH ; read I/O from port FCH



NEXT CLASS

Full Segment Definition and Memory Model



THANK YOU



Memory Organization



- ✓ The assembler uses two basic formats for developing software. One method uses **memory models** and the other uses **full-segment definitions**. MASM uses memory models. The TASM also uses memory models, but they differ somewhat from the MASM models
- ✓ The full-segment definitions are common to most assemblers, including Intel assembler, and are often used for software development. The models are easier to use for simple task
- ✓ There are many models available to the MASM assembler from tiny to huge



Example 4-18: Write a program to copy the contents of a 100-byte block of memory (LISTA) into a second 100-byte block of memory(LISTB)

```
.MODEL SMALL ; Small model
.STACK 100H ; define stack
.DATA ; start data segment
    LISTA DB 100 DUP(?)
    LISTB DB 100 DUP(?)

.CODE ; start code segment
HERE: MOV AX,@DATA ; load ES and DS
      MOV ES,AX
      MOV DS,AX
      CLD ; move data
      MOV SI,OFFSET LISTA
      MOV DI,OFFSET LISTB
      MOV CX,100
      REP MOVSB

.EXIT 0 ; exit to DOS
END HERE
```



- ✓ The program shows how to define the stack, data, and code
- ✓ The .EXIT 0 directive returns to DOS with an error code of 0 (no error)
- ✓ @DATA is a special directive used to identify various segments
- ✓ If the .STARTUP directive is used (MASM version 6.X), the MOV AX,@DATA followed by MOV DS,AX segments can be eliminated
- ✓ .STARTUP directive also eliminates the need to store the starting address next to the END label



Full-Segment Definitions

```
STACK_SEG SEGMENT ' STACK'
DW 100H DUP(?)  

STACK_SEG ENDS  

DATA_SEG SEGMENT 'DATA'
LISTA DB 100 DUP(?)  

LISTB DB 100 DUP(?)  

DATA_SEG ENDS  

CODE_SEG SEGMENT 'CODE'  

ASSUME CS:CODE_SEG, DS:DATA_SEG  

ASSUME SS:STACK_SEG  

MAIN PROC FAR
    MOV AX,DATA_SEG
    MOV ES,AX
    MOV DS,AX
    CLD
    MOV SI,OFFSET LISTA
    MOV DI,OFFSET LISTB
    MOV CX,100
    REP MOVS
    MOV AH,4CH
    INT 21H
MAIN ENDP  

CODE_SEG ENDS
END MAIN
```



- ✓ Example 4-19 appears longer than the one pictured in Example 4-18
- ✓ Bit it is more structured
- ✓ The first segment defined is the STACK_SEG, which is clearly described with the SEGMENT and ENDS directive
- ✓ Within these directives, DW 100H DUP (?) sets aside 100H words for the stack segment. Because the word STACK appears next to SEGMENT, the assembler and linker automatically load stack register (SS) and stack pointer (SP)
- ✓ The data are defined in the DATA_SEG
- ✓ Here, two arrays of data appear as LISTA and LISTB
- ✓ Each array contains 100 bytes of space for the program
- ✓ The names of the segments in this program can be changed to any name
- ✓ CODE_SEG is organized as a far procedure
- ✓ Before the program begins, the code segment contains ASSUME statement
- ✓ ASSUME statement tells the assembler and linker that the names used for the code segment (CS) is CODE_SEG, it also tells the assembler and linker that the data segment is DATA_SEG and stack segment is STACK_SEG



Description

- ✓ After the program loads both the extra segment register and data segment register with the location of the data segment, it transfers 100 bytes from LISTA to LISTB
- ✓ Following this is a sequence of two instructions that return control back to DOS (Disk Operating system)
- ✓ Note that the program loader does not automatically initialize DS and ES. These registers must be loaded with the program
- ✓ The last statement in the program is END MAIN
- ✓ MAIN (i.e procedure starts label) is needed after END



A Sample Program

Example 4-20: Write a program using full-definitions, that reads a character from the keyboard and displays it on the CRT screen. Note that @ key ends the program

Note: This program illustrates the use of a few DOS function calls. The BIOS function calls allow the use of the keyboard, printer, disk drives, and everything else that is available in your computer system



```

CODE_SEG      SEGMENT 'CODE'
ASSUME       CS:CODE_SEG
MAIN         PROC FAR
MOV AH,06H      ;reads a key
MOV DL,0FFH
INT 21H
JE MAIN        ; if no key typed
CMP AL,'@'
JE MAIN1       ; if an @ key
MOV AH,06H      ; display key (echo)
MOV DL,AL
INT 21H
JMP MAIN        ; repeat

```

MAIN1:

```

MOV AH, 4CH      Is equivalent to .EXIT
INT 21H

```

```

MAIN ENDP
ENDS
END MAIN

```

CODE_SEG



Description

- ✓ This example program uses only a code segment because there is no data
- ✓ A stack segment should appear, but it has been left out because DOS automatically allocates a 128-byte stack for all programs
- ✓ The only time that the stack is used in this example is for the INT 21H instructions that call a procedure in DOS
- ✓ The stack is fewer than 128 bytes in this example
- ✓ The program uses a DOS functions 06H and 4CH. The function number is placed in AH before the INT 21H instruction executes. The 06H function reads the keyboard if DL = 0FFH, or displays the ASCII contents of DL if it is not 0FFH

The first section of the program moves 06H into AH and 0FFH into DL, so that a key is read from the keyboard. The INT 21H tests the keyboard. If no key is typed, it returns equal. The JE instruction tests equal condition and jumps to MAIN if no key is typed

- ✓ When a key is typed, the program continues to the next step, which compares the contents of AL with an @symbol
- ✓ Upon return from the INT 21H, the ASCII character of the typed key is found in AL. In this program, if an @ symbol is typed, the program ends
- ✓ If the @ symbol is not typed, the program continues by displaying the character typed on the keyboard with the next INT 21H instruction
- ✓ The second INT 21H moves the ASCII character into DL so it can be displayed on the CRT screen
- ✓ If the @ symbol is typed, the program continues at MAIN1, where it executes the DOS function code number 4CH. This causes the program to return to the DOS prompt so that the computer can be used for other tasks



Example 4-21: Same problem using memory model

.MODEL TINY

.CODE

.STARTUP

MAIN: MOV AH,6 ; read a key
 MOV DL,0FFH
 INT 21H
 JE MAIN ; if no key typed
 CMP AL, '@'
 JE MAIN1 ; if an @ key
 MOV AH, 06H ; display key (echo)
 MOV DL,AL
 INT 21H
 JMP MAIN ; repeat

MAIN1:

.EXIT ; exit to DOS

END



Structure

Structure of a main module using simplified directive

.MODEL SMALL

;This statement is required before you can use other simplified segment directives

.STACK

; use default 1-Kilobyte stack (128 bytes)

.DATA

; begin data segment

; place data declarations here

.CODE

.STARTUP

; begin code segment

; generate start-up code

; place instructions here

.EXIT

END

; generate exit code

Note:

If you do not use **.STARTUP**, you must give starting address as an argument to the **END** directive

For Example:

.CODE

START:

; place executable code here

END START



Segment Order Directives

You can control the order in which segments appear in the executable program with three directives

.SEQ

Arranges segments in the order in which you declare them

.ALPHA

Directive specifies alphabetical segment ordering within a module

.DOSSEG

Directive specifies the MS-DOS segment-ordering convention. It places segments in the standard order required by Microsoft Language

The **.DOSSEG** directive orders segments as follows

- i) Code Segment
- ii) Data Segment



Initializing DS and SS

- ✓ The DS register is automatically initialized to the correct value if you use .STARTUP. If you do not use .STARTUP with the MS-DOS, you must initialize DS using the following instructions

MOV AX,@DATA

MOV DS,AX

- ✓ The SS and SP registers are initialized automatically if you use the .STACK directive. If you want SS to be equal to DS, use .STARTUP



Levels of Programming

Machine Level Programming

The language in which the instructions are represented by binary codes is called machine language.

Features

- ✓ The machine language programs developed for one processor cannot be used for another processor.
- ✓ The machine level programs are machine dependent. It is highly tedious for a programmer to write programs in the machine language



Assembly Level Programming

In ALP, instructions are written using mnemonics. If the program is developed using mnemonics then it is called ALP

Features

- ✓ μP cannot execute the assembly language programs directly. The assembly language programs have to be converted to machine language for execution. This conversion is performed using a software called assembler.
- ✓ The mnemonics of one processor will not be same as that of another processor. The ALPs are machine dependent



High Level Programming

In high level programming the instructions will be in the form of statements written using symbols, English words and phrases

Features

- ✓ The programs written in high level languages are easy to understand and machine independent
- ✓ A high level language program has to be converted into machine language programs in order to be executed by the processor. The conversion is performed by a software called compiler



Types of Assembler

- ✓ One-pass assembler
- ✓ Two-pass assembler
- ✓ Macro assembler
- ✓ Resident assembler and
- ✓ Meta assembler

The assembler usually generates 2 output files called

1. Object file and (“.OBJ”)
2. List file (“.LIST”)

Any labels encountered are given an address and stored in a table by an assembler



One-pass assembler:

A one-pass assembler is an assembler in which the source codes are processed only once. A one-pass assembler is very fast and in one-pass assembler only backward reference may be used

Two-pass assembler:

The source codes are processed two times. In the first pass, the assembler assigns addresses to all the labels and attach values to all the variables used in the program. In the second pass, it converts the source code into machine code.



NEXT CLASS

DOS Function Call



THANK YOU



INT 21H (DOS FUNCTION CALL)



Selected DOS Function Call

- ❑ To use a DOS function call in a DOS program, place the function number in AH and other data that might be necessary in other registers.

Example-1

```
MOV AH,01H      ;load DOS function number  
INT 21H          ;access DOS  
;returns with AL=ASCII key code
```



Function Number 01H

Read the Keyboard

Entry

AH=01H

Exit

AL=ASCII character

This function call automatically echoes
whatever is typed to the video screen



Function Number 02H

Write to Standard Output Device

Entry

AH=02H

DL=ASCII character to be displayed

Exit

This function call normally displays data on the video display



Function Number 06H

Direct Console Read/Write

Entry

AH=06H

DL= 0FFH or DL= ASCII character to be displayed

Exit

AL=ASCII Character

If DL=0FFH on entry, then this function reads the console. If DL=ASCII Character, then this function displays the ASCII Character on the console (CON) video screen



Function Number 08H

Read standard input without echo

Entry

AH=08H

Exit

This function call reads the standard input device



Function Number 09H

Display a character string

Entry

AH=09H

DS:DX=address of the character string

Exit

The character string must end with an ASCII \$(24H). The character string can be of any length and may contain control characters such as carriage return (0DH) and line feed (0AH)



Function Number 0AH

Buffered Keyboard Input

Entry

AH=0AH

DS:DX=address of the keyboard input buffer

Exit

The first byte of the buffer contains the size of the buffer (up to 255). The second byte is filled with the number of the characters typed upon return. The third byte through the end of the buffer contains the character string typed, followed by a carriage return(0DH). This function continues to read the keyboard (displaying data as typed) until either the specified number of characters are typed or until the enter key is typed.



D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
Buffer Length	Actual Length								

Figure 1: Keyboard Buffer Structure

Input

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
08	XX								

Output

'H' 'E' 'L' 'L' 'O' 'enter' 'empty' 'empty'

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9
08	05	68	65	6C	6C	6F	0D	XX	XX



- ❑ DOS 0AH is invoked with DS:DX pointing to an input buffer
- ❑ Size of the buffer should be at least three bytes longer than the largest input string anticipated

Example-1 : To read a string of maximum length 5, the buffer can be defined as below

MAXLEN DB 8

ACTLEN DB ?

BUFFER DB 6 DUP(?)

To read a string into the buffer, invoke DOS function 0AH as

MOV AH , 0AH

MOV DX , OFFSET MAXLEN

INT 21H



Function Number 4CH

Terminate Program and return to DOS

MOV AH , 4CH

INT 21H



Example 4-21

An example DOS model program that reads a key and displays it. Note that an @ key ends the program

```
.MODEL TINY
.CODE
.STARTUP
MAIN:    MOV AH,6          ;read a key
          MOV DL,0FFH
          INT 21H
          JE MAIN        ;eqvt to JZ, if no key typed
          CMP AL,'@'
          JE MAIN1       ;if an @ key is typed,exit
          MOV AH,06H      ; display key (echo)
          MOV DL,AL
          INT 21H
          JMP MAIN        ;repeat
MAIN1:
.EXIT          ; exit to DOS
END
```

(see page number 151, also see example 4-20, programs can be written in this format as well)



NEXT CLASS

Arithmetic and Logic Instructions



THE END



Arithmetic and Logic Instructions



Addition(ADD and ADC)

- ✓ The bulk of the arithmetic instructions found in any microprocessor include addition, subtraction, and comparison
- ✓ More than 32,000 variations of the ADD instruction in the instruction set
- ✓ The only types of addition not allowed are memory-to-memory and segment registers
- ✓ The segment registers can only be moved, pushed, or popped



Examples:

- i) ADD AL,BL ; AL=AL+BL
- ii) ADD CX,DI ; CX=CX+DI
- iii) ADD CL,44H ; CL=CL+44H
- iv) ADD BX,245FH ; BX=BX+245FH
- v) ADD [BX],AL ; AL adds to the byte contents of the data segment memory location addressed by BX with the sum stored in the same memory location
- vi) ADD CL,[BP] ; The byte contents of the stack segment memory location addressed by BP add to CL with the sum stored in CL
- vii) ADD CL,TEMP ; The byte contents of data segment memory location TEMP add to CL with the sum stored in CL
- viii) ADC AL,AH ; AL=AL+AH+carry
- ix) ADC CX,BX ; CX=CX+BX+carry

[An addition-with-carry) instruction (ADC) adds the bit in the carry flag (C) to the operand data]



Increment Addition:

INC BL; BL=BL+1

INC SP ; SP=SP+1

(Increment addition adds 1 to a register or a memory location)

Subtraction:

- i) SUB CL,BL ; CL=CL-BL
- ii) SUB AX,SP ; AX=AX-SP
- iii) SUB DH,6FH ; DH=DH-6FH
- iv) SUB AH,TEMP ; subtracts the byte contents of memory location TEMP from AH and store the difference in AH

Decrement(DEC):

- i) DEC BH ; BH=BH-1
- ii) DEC CX ; CX=CX-1

Subtraction with Borrow:

- i) SBB AH,AL ; AH=AH-AL-Carry
- ii) SBB CL,2 ; CL=CL-2-Carry



Comparison

The comparison instruction (CMP) is a subtraction that changes only the flag bits; the destination operand never changes. A comparison is useful for checking the entire contents of a register or a memory location against another value. A CMP is normally followed by a conditional jump instruction, which test the condition of the flag bits

- i) CMP CL,BL ; CL-BL
- ii) CMP AX,SP ; AX-SP
- iii) CMP [DI],CH ; CH subtracts from the byte
 contents of the data segment
 memory location addressed by DI

Example 5-12:

CMP AL,10H ; compare AL against 10H
JAE SUBER ; if AL is 10H or above jump to location SUBER

Normally used:

JA(Jump Above)

JB(Jump Below)

JAE(Jump Above or Equal)

JBE(Jump Below or Equal)



Multiplication

Only modern microprocessor contain multiplication and division instruction. Earlier 8-bit microprocessor could not multiply or divide without the use of a program that multiplied or divided by using a series of shifts and addition or subtractions. Because microprocessor manufacturers were aware of this inadequacy, they incorporated multiplication and division instructions into the instruction sets of the newer microprocessors

Note:

- i) Multiplication is performed on bytes or words
- ii) The product after a multiplication is always a **double-width** product. If two 8-bit numbers are multiplied they generate a 16-bit product. If two 16-bit numbers are multiplied, they generate a 32-bit product
- iii) Some flag bits [**overflow (O)** and **carry (C)**] change when the multiply instruction executes and produce predictable outcomes



8-bit Multiplication

The multiplication instruction contains one operand because it always multiplies the operand times the contents of AL

- i) MUL CL ; AL is multiplied by CL and the unsigned product is in AX
- ii) IMUL DH ; AL is multiplied by DH and the signed product is in AX

Example 5-13

Write a short instruction to multiply the content of BL and CL.

Load BL

with 5, CL with 10. Store the result in DX register

```
MOV BL,5          ;load data  
MOV CL,10  
MOV AL,CL        ;position data  
MUL BL          ;multiply , result is in AX  
MOV DX,AX        ;hence move AX to DX
```

Use **IMUL** instead of **MUL** for signed 8-bit multiplication



16-bit Multiplication

- ✓ AX contains the multiplicand instead of AL
 - ✓ The 32-bit product appears in DX-AX instead of AX
 - ✓ DX register contains the Most Significant 16 bits of the product
 - ✓ AX contains the Least Significant 16 bits
-
- i) MUL CX ; AX is multiplied by CX and the unsigned product is in DX-AX
 - ii) IMUL DI ; AX is multiplied by DI and the signed product is in DX-AX

Note:

8086/8088 microprocessors can not perform immediate multiplication



Division

- ✓ Division occurs on 8-or 16-bit numbers
- ✓ The dividend is always a double-width dividend that is divided by the operand
- ✓ This means that an 8-bit division divides a 16-bit number by an 8-bit number; a 16-bit division divides a 32-bit number by a 16-bit number
- ✓ None of the flag bits change predictably for a division. A division can result in two different types of errors
 - 1) One is an attempt to divide by zero
 - 2) Other is a divide overflow

For Example:

If AX=3000 is divided by 2, answer = 1500, which does not fit into AL



8-bit Division

- ✓ An 8-bit division uses the AX register to store the dividend that is divided by the contents of any 8-bit register or memory location
 - ✓ The quotient moves into AL after division with AH containing a whole number remainder
 - ✓ Zero-extension is needed in 8-bit division
- i) DIV CL ; AX is divided by CL; the unsigned quotient is in AL and the remainder is in AH
 - ii) IDIV BL ; AX is divided by BL; the signed quotient is in AL and the signed remainder is in AH

Example 5-14

Divide number NUMB by number NUMB1. Store the quotient at ANSQ and remainder at ANSR

```
MOV AL,NUMB    ; get NUMB
MOV AH,0        ; zero-extend
DIV NUMB1      ; divide by NUMB1
MOV ANSQ,AL     ; save quotient
MOV ANSR,AH     ; save remainder
```



16-bit Division

DIV CX ; DX-AX is divided by CX and the unsigned quotient is in AX and unsigned remainder is in DX

Example 5-15:

-100 is in AX. Divide by +9 in CX register

Solution:

MOV AX, -100

; load -100

MOV CX, 9

; load +9

CWD

; CWD convert word to double word

(i.e -100 in AX is converted to -100 in DX-AX)

IDIV CX

;quotient -11 will be in AX and remainder -1 in DX



BCD Arithmetic

DAA: Decimal Adjust after Addition

DAS: Decimal Adjust after Subtraction

- ✓ DAA follows BCD addition i.e ADD or ADC
- ✓ DAS follows BCD subtraction
- ✓ The adjustment instructions function only with the AL register after BCD addition and subtraction
- ✓ DAS instruction functions as does the DAA instruction, except that it follows a subtraction instead of an addition



ASCII Arithmetic

The ASCII arithmetic function with ASCII-coded numbers.

There are four instructions used with ASCII arithmetic operations

- i) AAA (ASCII adjust after addition)
- ii) AAD(ASCII adjust before Division)
- iii) AAM(ASCII adjust after Multiplication)
- iv) AAS(ASCII adjust after Subtraction)

These instructions use register AX as the source and as the destination.



AAA

Example 5-10:ASCII addition in 8086

MOV AX,31H	;load ASCII '1'
ADD AX,39H	;load ASCII '9'
AAA	; AX will contain 10
ADD AX,3030H	; answer to ACII if we wish to display

Explanation:

- ✓ If 31H and 39H are added , result is 6AH. The ASCII addition (1+9) should produce a two-digit ASCII result equivalent to a decimal 10, which is a 31H and 30H in ASCII code
- ✓ If the **AAA** instruction is executed after the addition, the AX register will contain 0100H
- ✓ Although this is not ASCII code, it can be converted to ASCII code by adding **3030H** to AX which generates 3130H

AAD

Example 5-21: Divide 72 in unpacked BCD by 9

```
MOV BL,9          ;load divisor  
MOV AX,72H        ;load dividend  
AAD              ; adjust  
DIV BL           ; divide
```



AAM Instruction:

It is followed after Multiplication

Example 5-22:

```
MOV AL,5          ;load multiplicand  
MOV CL,3          ;load multiplier  
MUL CL           ;multiply  
AAM               ; adjust
```

AAS Instruction:

AAS instruction adjusts the AX register after ASCII Subtraction



Basic Logic Instructions

AND:

- i) AND AL,BL ; AL=AL and BL
- ii) AND CX,DX ; CX=CX and DX
- iii) AND CL,33H ; CL=CL and 33H

OR:

- i) OR AH,BL ; AH=AH or BL
- ii) OR SI,DX ; SI=SI or DX
- iii) OR SP,990DH ; SP=Sp or 990DH

XOR:

- i) XOR CH,DL ; CH=CH xor DL
- ii) XOR DI,BX ; DI=DI xor BX
- iii) XOR AH,0EEH ; AH=AH xor EEH

TEST:

The TEST instruction performs the AND operation. The difference is that the AND instruction changes the destination operand, whereas the TEST instruction does not.

- i) TEST DL,DH ; DL is ANDed with DH
- ii) TEST CX,BX ; CX is ANDed with BX

NOT and NEG:

- i) NOT CH ; CH is **one's** complemented
- ii) NEG CH ; CH is **two's** complemented
- iii) NEG AX ; AX is **two's** complemented



Shift and Rotate Instructions

Shift:

Shift instructions position or move numbers to the left or right within a register or memory location. The microprocessors instruction set contains four different shift operations. Two are logical shifts and two are arithmetic shifts

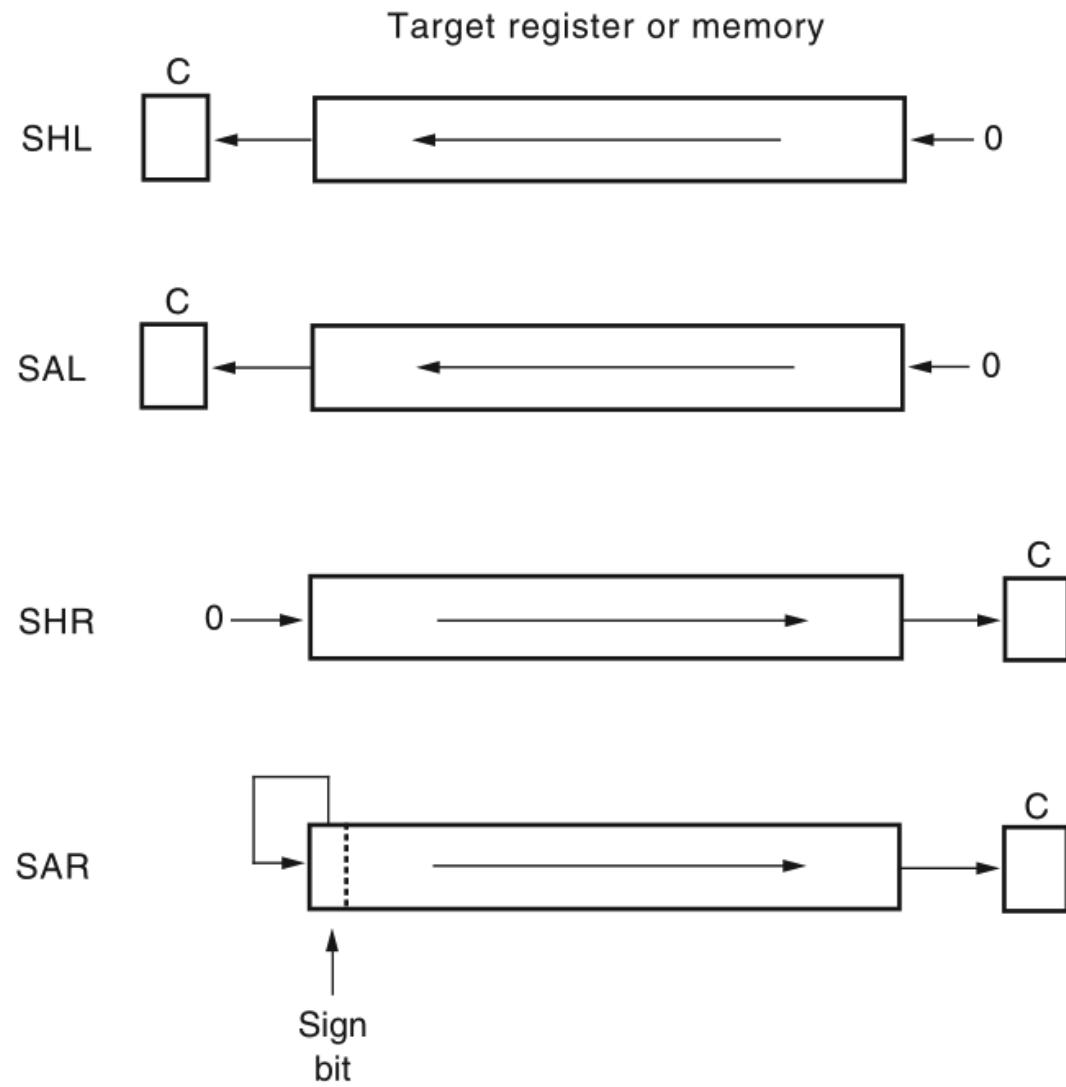
- i) SHL AX,1 ; AX is logically shifted left 1 place
- ii) SHR BX,12 ; BX is logically shifted right 12 places
- iii) SAR SI,2 ; SI is arithmetically shifted right 2 places
- iv) SAL DATA1,CL ; The contents of data segment memory
location DATA1 are arithmetically shifted
left the number of spaces specified by CL

Note:

- ✓ The arithmetic left shift and logical left shift are identical
- ✓ The **arithmetic right shift** and **logical right shift** are **different** because the arithmetic right shift copies the sign-bit through the number, whereas the logical right shift copies a 0 through the number
- ✓ Segment register cannot be shifted
(Please see **example 5-31** for an example to see how to multiply AX by 10, 18, or 5)



FIGURE 5–9 The shift instructions showing the operation and direction of the shift.



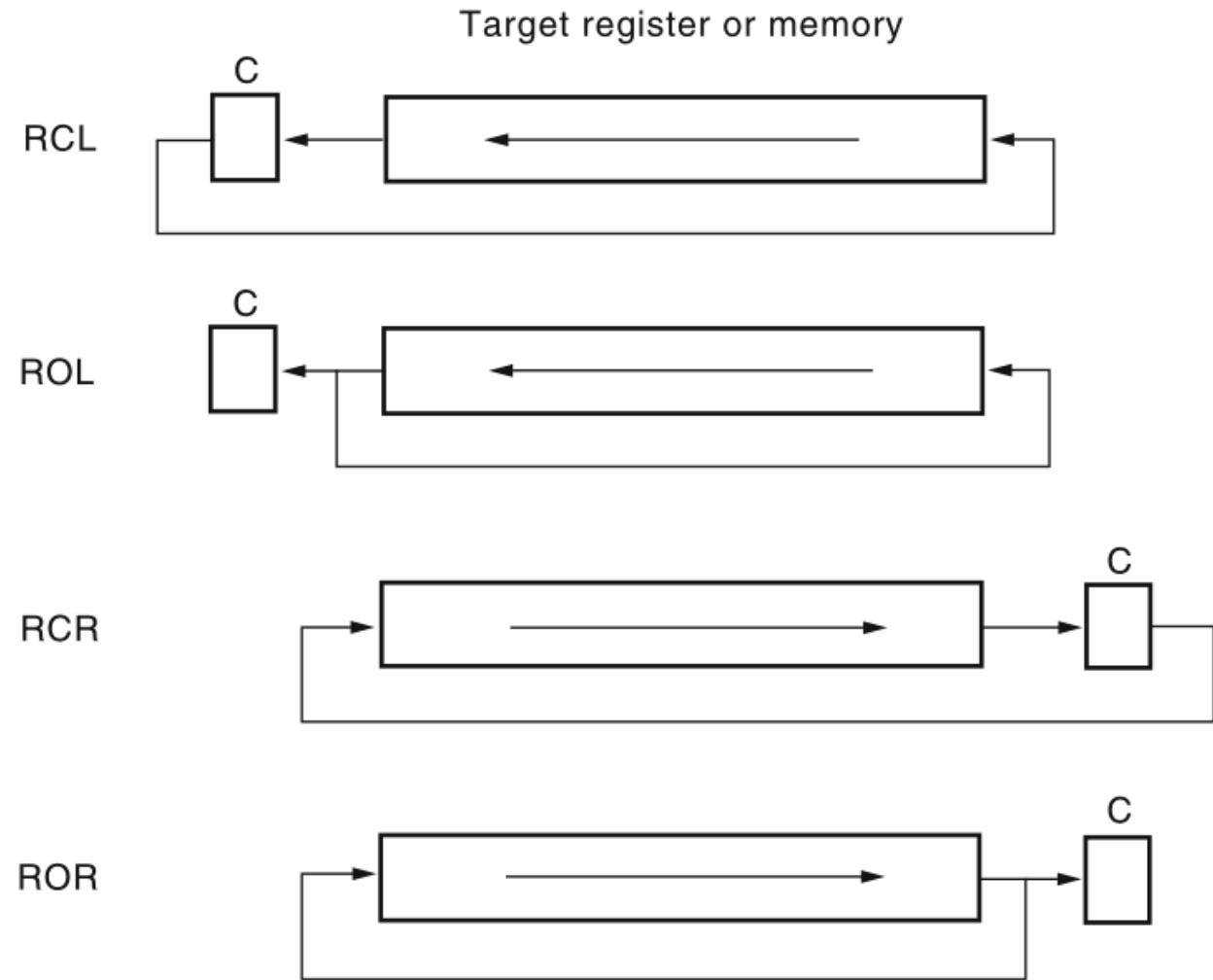
Rotate:

Rotate instructions position binary data by rotating the information in a register or memory location, either from one end to another or through the carry

- i) ROL SI,14 ; SI rotates left 14 places
- ii) RCR AH,CL ; AH rotates right through carry the number of places specified by CL



FIGURE 5–10 The rotate instructions showing the direction and operation of each rotate.



NEXT CLASS

Program Control Instructions



THANK YOU



Program Control Instruction



Program Control Instruction

The relational assembly language statements can be used

.IF
.ELSE
.ELSEIF
.ENDIF
.WHILE
.ENDW
.REPEAT
.UNTIL

Available in version **6.XX** of **MASM**



Conditional Jump and Conditional Sets

JA	Z=0 and C=0	Jump if above
JAE	C=0	Jump if above or equal
JB	C=1	Jump if below
JBE	Z=1 or C=1	Jump if below or equal
JC	C=1	Jump on carry
JE or JZ	Z=1	Jump if equal or jump if zero
JG		
JGE		
JL		
JLE		
JNC		
JNE/JNZ		
JNO		Jump on overflow

(Please see page 199 in text book,8e)



LOOP

- ❑ The LOOP instruction is a combination of a decrement CX and JNZ conditional jump.
- ❑ The LOOP decrements CX; if CX != 0, it jumps to the address indicated by the label
- ❑ If CX becomes 0, the next sequential instruction executes.



Example 6-7

Write a program that sums the contents of BLOCK1 and BLOCK2 and stores the result on top of data in BLOCK2

```
.MODEL SMALL
.DATA
BLOCK1      DW    100    DUP(?) ;100 words for BLOCK1
BLOCK2      DW    100    DUP(?) ;100 words for BLOCK2
.CODE
.STARTUP
    MOV AX,DS          ;overlap DS and ES
    MOV ES,AX
    CLD
    MOV CX,100         ;select auto-increment
    MOV SI,OFFSET BLOCK1 ;load counter
    MOV DI,OFFSET BLOCK2 ;address BLOCK1
    MOV DI,OFFSET BLOCK2 ;address BLOCK2
L1:LODSW
    ADD AX,ES:[DI]     ;add BLOCK2
    STOSW              ;save answer
    LOOP L1            ;repeat 100 times
.EXIT
END
```



Controlling the Flow of Program

Operator	Function
==	Equal or the same as
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
&	Bit test
!	Logical Inversion
&&	Logical AND
	Logical OR

(See page 203)



Example 6-9

Write a program that reads the keyboard and converts all lowercase data to uppercase before displaying it. The program should be terminated with a control-C (03H ASCII code)

```
.MODEL TINY
.CODE
.STARTUP
    MAIN1: MOV AH,06H ;read key without echo
        MOV DL,0FFH
        INT 21H
        JE MAIN1
        CMP AL,3           ;check for ctrl-C
        JE MAIN2
    .IF AL>= 'a' && AL<= 'z'
        SUB AL,20H         ; Please Note 'A' = 41H and 'a' =61H
    .ENDIF
        MOV DL,AL
        MOV AH,02H          ;echo character to display (missing in text book, add it)
        INT 21H
        JMP MAIN1           ;repeat
    MAIN2:
    .EXIT
END
```



Procedures

- ❑ A procedure is a group of instructions that usually performs one task. A procedure begins with the PROC directive and ends with the ENDP directive. Each directive appears with the name of the directive.

Example 6-14:

SUMS PROC NEAR

ADD AX,BX

ADD AX,CX

ADD AX,DX

RET

SUMS ENDP



Example 6-15

Write a DOS program that displays OK using a DISP procedure

.MODEL TINY

.CODE

.STARTUP

```
MOV BX,OFFSET DISP
MOV DL,'O'
CALL BX
MOV DL,'K'
CALL BX
```

.EXIT

DISP PROC NEAR

MOV AH,2

INT 21H

RET

DISP ENDP

END



CALL:

- ❑ The CALL instruction transfers the flow of the program to the procedure. The CALL instruction differs from the jump instruction because CALL saves a return address on the stack.

RET:

- ❑ The return instruction(RET) removes a 16-bit number(near return) from the stack and places it into IP, or a 32-bit number (far return) and places into IP and CS



Macros

- ❑ A macro is a group of instructions that perform one task, just as a procedure performs one task. The difference is that a procedure is accessed via a CALL instruction, whereas a macro, and all the instructions defined in the macro, is inserted in the program at the point of usage
- ❑ The MACRO and ENDM directives describe a macro sequence
- ❑ Macro sequences must always be defined before they are used in a program, so they are generally appear at the top of the code segment

Example 8-10:

MOVE MACRO A,B

PUSH AX

MOV AX,B

MOV A,AX

POP AX

ENDM

MOVE VAR1,VAR2

MOVE VAR3,VAR4

;MOVE VAR2 INTO VAR1

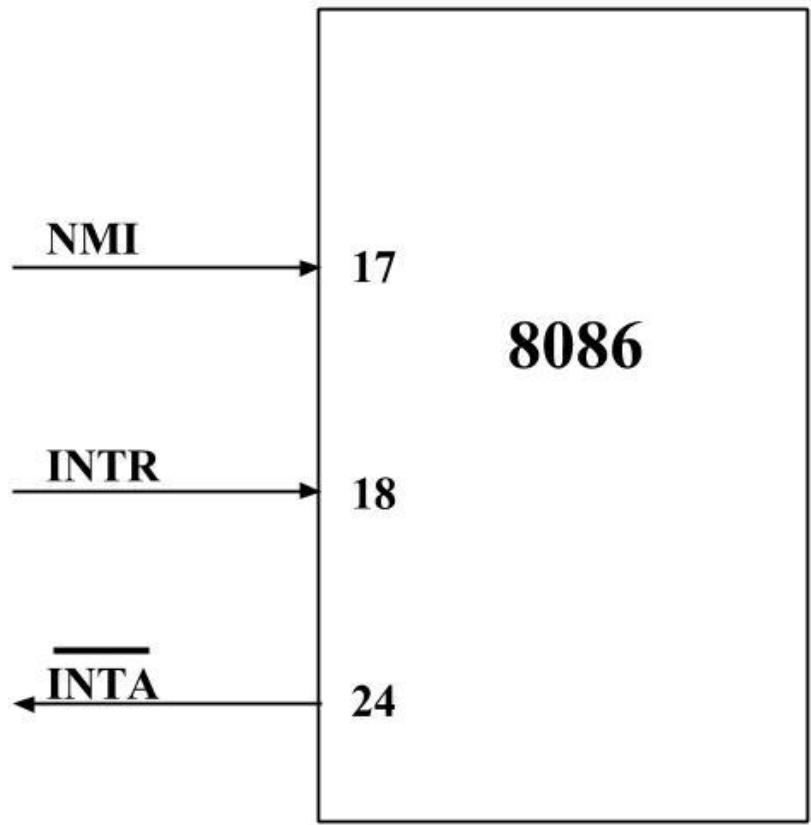
;MOVE VAR4 INTO VAR3



Interrupts in 8086

- ❑ An interrupt is either a **hardware-generated** CALL or a **software-generated** CALL.
- ❑ Software-generated CALL is usually called an **exception**
- ❑ Either type interrupts the program by calling an Interrupt Service Procedure(**ISP**)





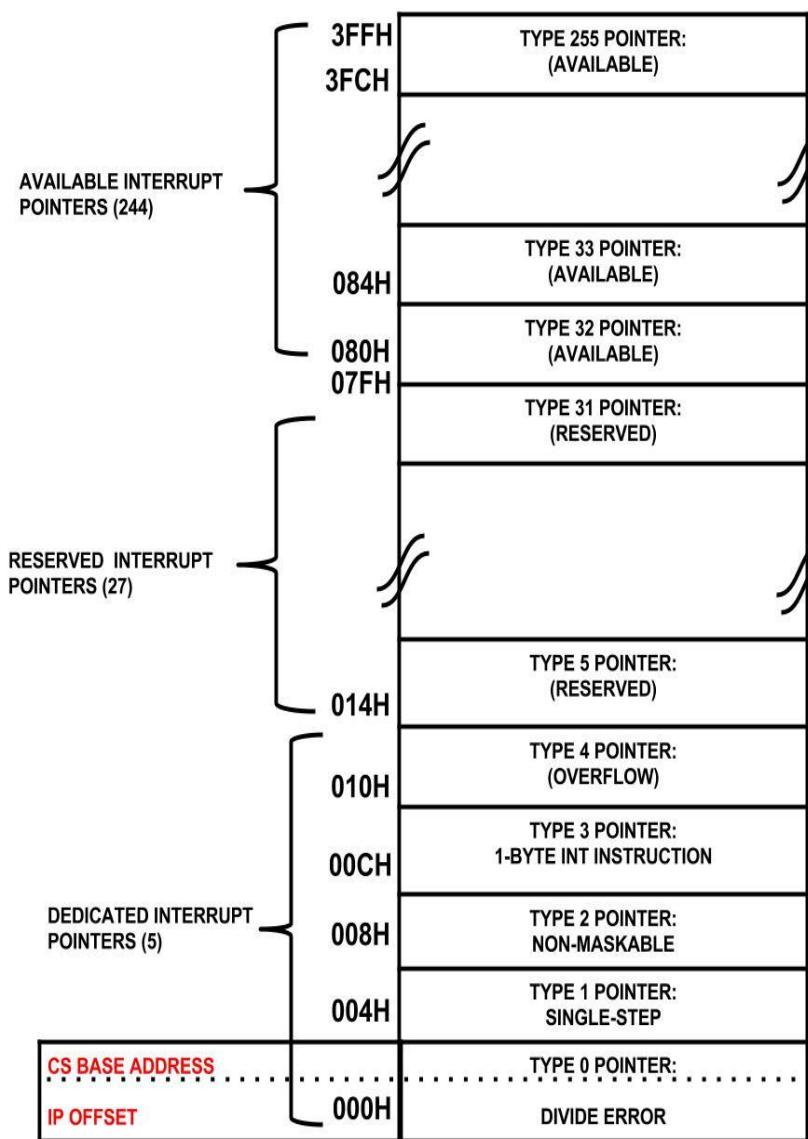
Interrupt Vectors

- An **interrupt vector**(also called **interrupt pointer**) is a **4-byte** number stored in the first **1024** bytes of the memory (00000H-003FFH)
- There are **256** different interrupts vectors, and each vector contains the address of an interrupt service procedure
- Each vector contains a value for IP and CS that forms the address of the ISP. The first 2 bytes contain the IP, and the last 2 bytes contain the CS
- INTEL reserves the first 32 interrupt vectors for the present and future microprocessor products. The remaining interrupt vectors (32-255) are available for the user
- The table is referred to as the interrupt-vector table(**IVT**)



Interrupt Vector Table

- The IVT shown depicts how 256 interrupt vectors are arranged in the table in memory.
- Each double word (32 bit) interrupt vector is identified by a number 0 to 255
- INTEL calls this number the **type** of the interrupt
- 0-4: Dedicated
- 5-31: Reserved
- 32-255: Available



Reference: Page 9.11, Hall

INTS

- There are 256 different software interrupt instructions (INTS) available to the programmer.
- Each INT instruction has a numeric operand whose range is 0-255 (00H-FFH)
- The address of the interrupt vector is determined by multiplying the interrupt type number by 4.
- For example, the INT 10H instruction calls the interrupt service procedure whose address is stored beginning at the memory location 40H($10H \times 4$) in the real mode
- Each INT instruction is 2 byte long. The first byte contains the opcode , and the second byte contains the vector type number



Whenever a software interrupt instruction executes:

- i. Pushes the flags onto the stack
- ii. Clears the T and I flag bits
- iii. Pushes CS onto the stack
- iv. Fetches the new value for CS from the interrupt vector
- v. Pushes IP on the stack
- vi. Fetches the new value for IP from the vector , and
- vii. Jumps to the new location addressed by CS and IP



IRET:

- The interrupt return instruction (IRET) is used only with software or hardware interrupt service procedure. The IRET instruction
 - i. Pop stack data back into the IP
 - ii. Pop stack data back into CS
 - iii. Pop stack data back into the flag registers

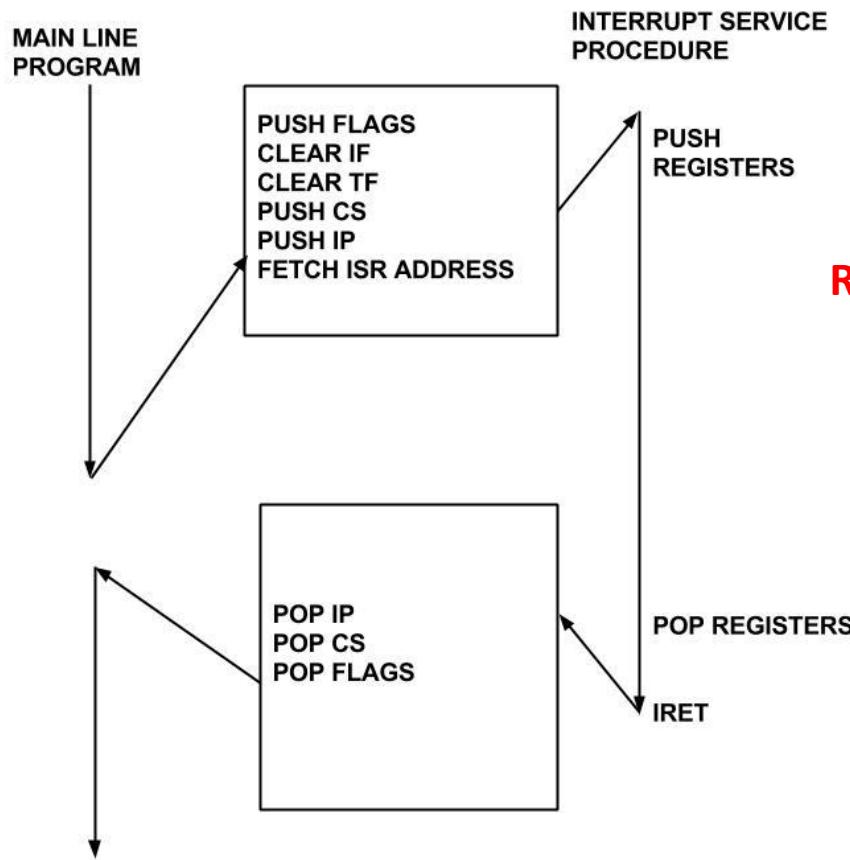
Interrupt Control:

STI: Set Interrupt Flag

CLI: Clear Interrupt Flag



8086 Interrupt Response



Reference: Page 9.10, Hall

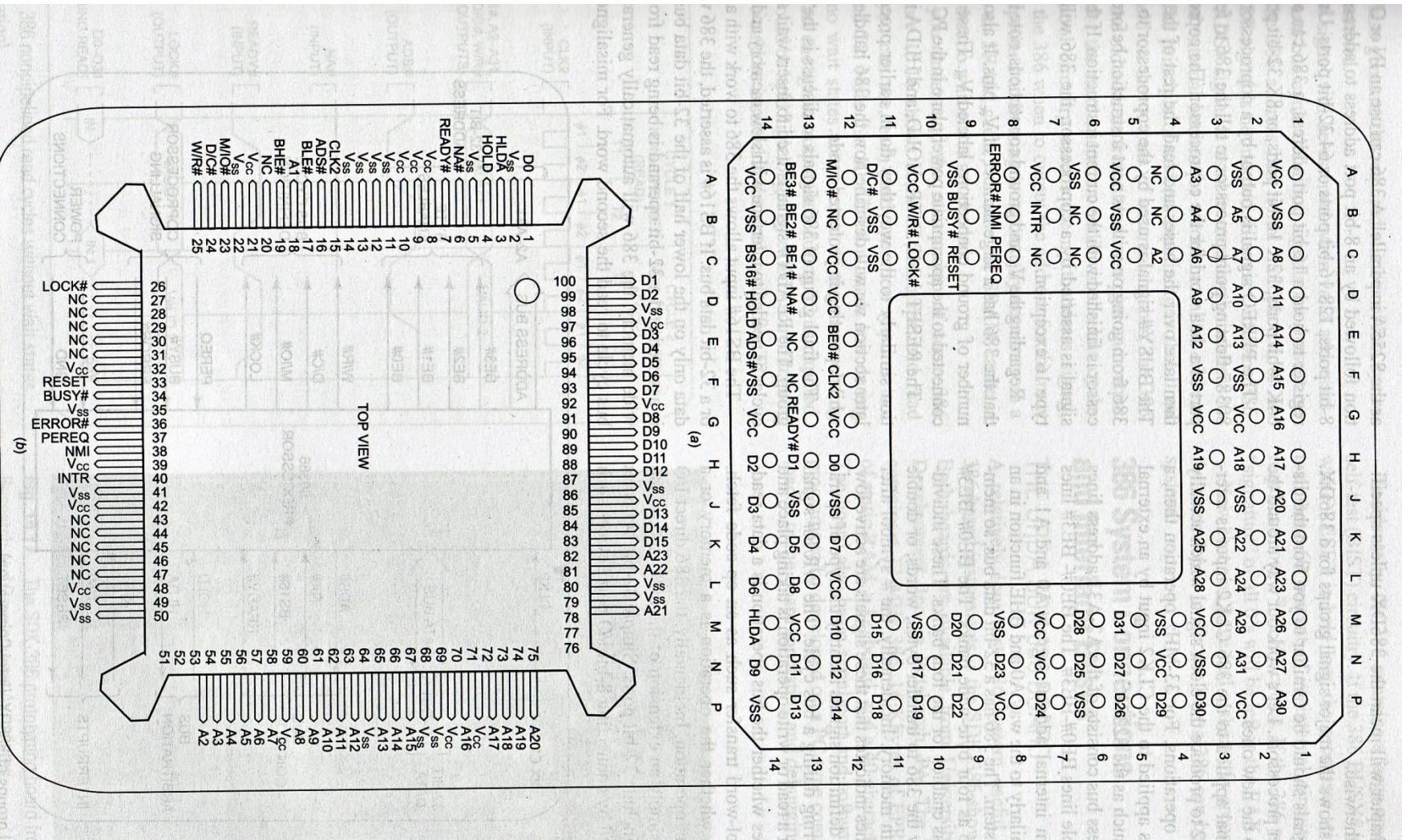
INTEL 80386 32-BIT Microprocessor



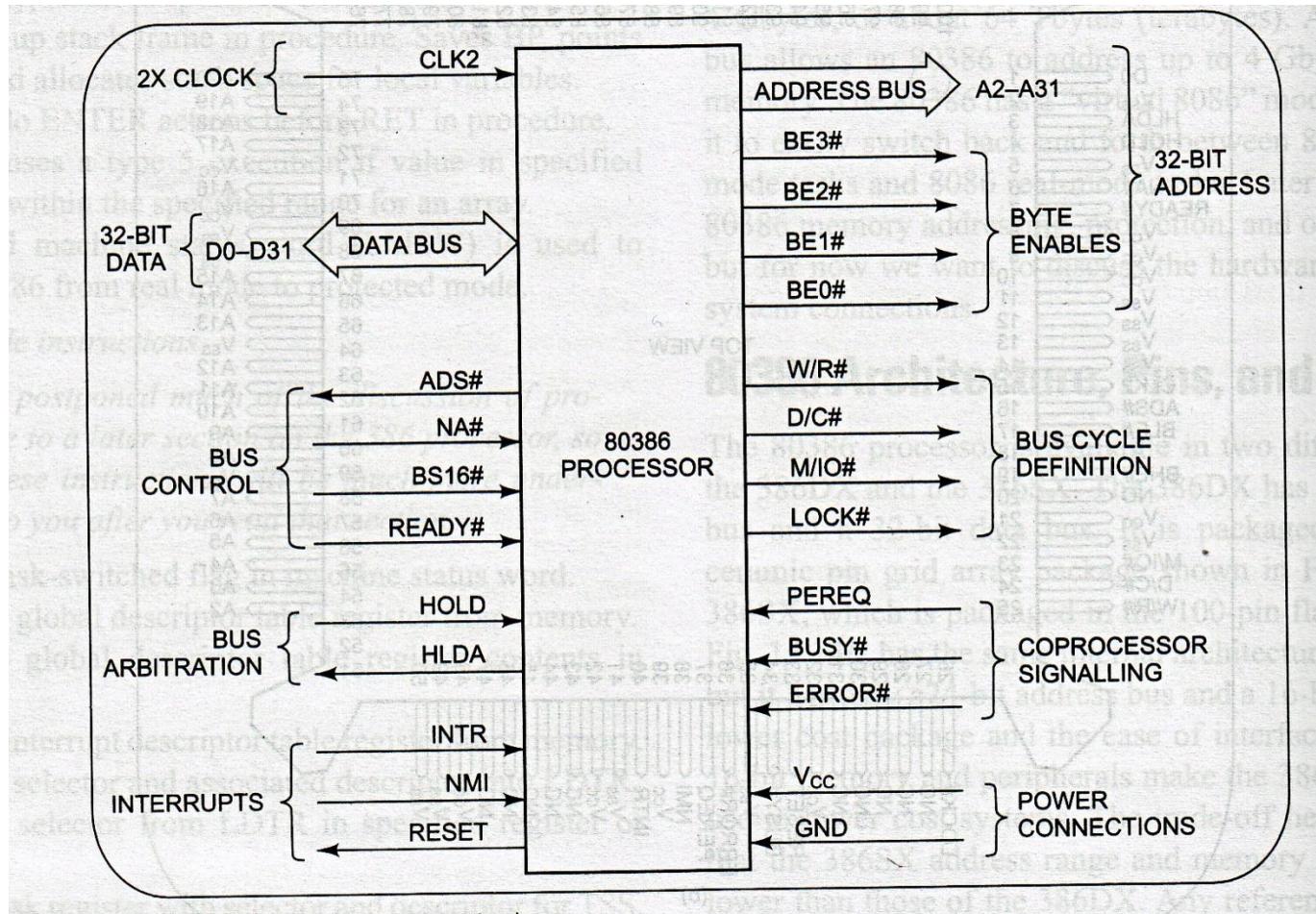
Features

- 32-bit microprocessor
- 80386 segments can be as large as 4GB
- Available in two different versions: 386DX, 386SX
- 386DX has 32-bit address bus and a 32-bit data bus. It is packaged in 132-pins
- 386SX has 100 pins. It has same architecture as the 386SX but it has a 24-bit address bus and a 16-bit data bus
- We consider 386DX as 80386 in general
- 80386 can run at 20,25 or 33 MHz clock speed





Signals groups of 80386



Refer page 15.16 in Hall

Signal groups

1. 32-bit address
2. Bus Cycle definition
3. Coprocessor Signalling
4. Power Connections
5. Interrupts
6. Bus Arbitration
7. Bus Control
8. 32-bit data
9. Clock



- The “#” symbol after BE signal names indicates that these signals are active low
- 80386 has a 32-bit data bus, so memory can be set up as four byte-wide banks.
- The **BEO#-BE3#** signals function as enables for the four banks
- Consists of A2-A31 address
- BEO#-BE3# lines are decoded from internal address signals A0, and A1



- The bus cycle definition signals identify the type of operation that is occurring during a bus cycle
- PREQ,BUSY#,ERROR# signals are used in relation to coprocessor such as **80387** floating point coprocessor
- 80386 has many Vcc and Vss pins
- RESET,NMI,INTR,HLDA, and HOLD functions a they do in 8086
- BS16# input allows 386 to work with a 16-bit and / or 32-bit data bus
- The ADS# signal will be asserted when valid addresses, BE signals, and bus cycle definitions signals are present on the buses
- The READY# signal is used to insert wait states in bus cycles as needed to interface with slow memory and I/O devices



THE END



SAMPLE 8086 PROGRAMS



Program No 1: Write an ALP that accepts a string from keyboard. If the string is “Nepal” display the message “Your Password is Correct” else display “Invalid Password”

.MODEL SMALL

.STACK 100H

.DATA

```
DATA1 DB 'Nepal'  
PROMPT1 DB 'ENTER PASSWORD : ','$'  
DISP1 DB 'YOUR PASSWORD IS CORRECT', '$'  
DISP2 DB 'INVALID PASSWORD','$'  
BUFFER1 DB 20 ;MAXIMUM 20 CHARACTERS  
          DB 0 ;TYPED NUMBER OF CHARACTER  
          DB 20 DUP (0) ;ACTUAL INPUT CHARACTER PLACED HERE
```

.CODE

.STARTUP

```
MOV AX,DS           ;POINT DS, AND ES TO DATA SEGMENT  
MOV ES,AX  
MOV DX, OFFSET PROMPT1 ;DISPLAY PROMPT1  
MOV AH,09H  
INT 21H
```



```
MOV AH,0AH
MOV DX,OFFSET BUFFER1
INT 21H
```

;GET THE STRING IN BUFFER1

```
MOV CX,5
CLD
MOV SI,OFFSET DATA1
MOV DI,OFFSET BUFFER1
INC DI

INC DI
REPE CMPSB
JZ CORRECT
```

;AUTO-INCREMENT

**;GET THE CHARACTER ,I.E POINT DI TO
THE THIRD CHARACTER IN BUFFER1**

;COMPARE TWO SEGMENT DATA

```
MOV DX,OFFSET DISP2
MOV AH,09H
INT 21H
JMP FINISH
```

CORRECT:
MOV DX,OFFSET DISP1
MOV AH,09H
INT 21H

FINISH:
.EXIT
END



Note: To get good UI use this

PROMPT1 DB 10,13,'ENTER 5 LETTER PASSWORD:','\$'

DISP1 DB 10,13,'YOUR PASSWORD IS CORRECT:','\$'

DISP2 DB 10,13,'INVALID PASSWORD:','\$'

13D => Carriage Return, ASCII 13D is the control character to bring the cursor to the start of a line

10D=> Line Feed, ASCII 10D is the control character that brings the cursor down to the next line on the screen

**CRLF in short used in some books(old version of Barry). Is equal to
\n in C programming**



Alternative Method

(Echoes every letter typed with an *)

```
.MODEL SMALL
.STACK 100H
.DATA
    DATA1 DB 'Nepal'
    PROMPT1 DB 10,13,'ENTER 5 LETTER PASSWORD:','$'
    DISP1     DB 10,13,'YOUR PASSWORD IS CORRECT', '$'
    DISP2     DB 10,13,'INVALID PASSWORD','$'
    CHAR DB 5 DUP(0)
.CODE
.STARTUP
```



MOV DX,OFFSET PROMPT1

MOV AH,09H

INT 21H

MOV BL,0

MOV CL,5

MOV SI,0

NEXT: MOV AH,08

INT 21H

MOV CHAR[SI],AL

CMP AL,DATA1[SI]

JZ MATCH_COUNT

BACK: MOV AH,02H

MOV DL,'*'

INT 21H

INC SI

DEC CL

JNZ NEXT

JMP VERIFY



MATCH_COUNT:INC BL

JMP BACK

VERIFY: **CMP BL,5**

JZ CORRECT

INVALID: **LEA DX,DISP2**

MOV AH,09H

INT 21H

JMP FINISH

CORRECT: **LEA DX,DISP1**

MOV AH,09H

INT 21H

FINISH:

.EXIT

END



Program 2: Write an ALP which reads a string from Keyboard. Check for palindrome. If it is a palindrome, display “Palindrome detected” else “No Palindrome detected” in the output screen

.MODEL SMALL

.STACK 100H

.DATA

STRING1 DB 10,13,’ENTER THE STRING: ’,’\$’

STRING2 DB 10,13,’PALINDROME DETECTED: ’,’\$’

STRING3 DB 10,13,’NO PALINDROME DETECTED: ’,’\$’

COUNT DB 0

BUFFER DB 30

DB 0

DB 255 DUP(0)

.CODE

.STARTUP

MOV BL,00H



MOV AH,09H

**MOV DX,OFFSET STRING1
INT 21H**

;PROMPT FOR STRING

MOV AH,0AH

**MOV DX,OFFSET BUFFER
INT 21H**

**;GET THE STRING AND PUT IN BUFFER
;SO POINT DX TO BUFFER TO SAVE**

MOV SI,0

MOV AL,BUFFER[SI+1]

**;THIS IS THE NUMBER OF THE
;CHARACTERS TYPED
;PUT THAT IN COUNT
;ALSO PUT IN CL**

MOV COUNT,AL

MOV CL,AL

;NUMBER DETECTED,NOW RUN THE CHECK

MOV CH,00H

**;CX HAS TO BE MOVED TO DI,SO PUT
;00H IN CH**

MOV SI,2

**;CHARACTERS IN THE BUFFER WILL
;BE STORED FROM THE 3RD BYTE
;ONWARDS, SO POINT SI TO THAT**

MOV DI,CX



ADD DI,SI

DEC DI

**;DECREMENT DI BY 1 TO POINT THE LAST
;CHARACTER**

BACK: MOV AL,BUFFER[SI]

CMP AL,BUFFER[DI]

JZ COUNT1

BACK1: DEC DI

INC SI

DEC CL

JNZ BACK

JMP FINISH

COUNT1: INC BL

JMP BACK1

FINISH: CMP COUNT,BL

JNZ NO_PALINDROME

MOV DX,OFFSET STRING2 ;DISPLAY PALINDROME OTHERWISE

MOV AH,09H

INT 21H

JMP FINISH1

29 October 2017

Pramod Ghimire

Slide 545 of 601



NO_PALINDROME: MOV DX,OFFSET STRING3 ;NO PALINDROME
MOV AH,09H
INT 21H

FINISH1:

.EXIT

END

For Example: asÅsa is palindrome



Program No 3: Write an ALP to check whether a given number is positive or negative. Also check if it is odd or even. Display the message in the output screen accordingly. For example if the number is 08H, the displayed message should be “Number is positive & it is an even number”. If the number is OFFH, the displayed message should be “Number is negative & it is an odd number”

.MODEL SMALL

.STACK 100H

.DATA

NUMBER DB 08H

STRING2 DB 10,13,’NUMBER IS POSITIVE ’,’\$’

STRING3 DB 10,13,’NUMBER IS NEGATIVE ’,’\$’

STRING4 DB ’& IT IS AN EVEN NUMBER ’,’\$’

STRING5 DB ’& IT IS AN ODD NUMBER ’,’\$’

.CODE

.STARTUP

MOV AL,NUMBER

;GET THE NUMBER FROM MEMORY TO AL

ROL AL,1

;ROTATE TO SEE IF THE CARRY IS 1 OR 0

JC NEGATIVE

;IF CY=1,NUMBER IS NEGATIVE,DISPLAY

;MESSAGE ACCORDINGLY



MOV DX,OFFSET STRING2
JMP FINISH_POSITIVE

;ELSE NUMBER IS POSITIVE

NEGATIVE: **MOV DX,OFFSET STRING3**

FINISH_POSITIVE: **MOV AH,09H**

INT 21H

MOV AL,NUMBER

ROR AL,1 ;**IF D0 IS 1, NUMBER IS ODD ELSE IT IS EVEN**

JC NUMBER_ODD

MOV DX,OFFSET STRING4

JMP FINISH_EVEN

NUMBER_ODD: **MOV DX,OFFSET STRING5**

FINISH_EVEN: **MOV AH,09H**

INT 21H

.EXIT

END



Program No 4: Write an 8086 ALP for MASM to display the text “POKHARA UNIVERSITY” in reverse order

.MODEL SMALL

.STACK 100H

.DATA

TEXT1 DB 'POKHARA UNIVARSITY'

COUNT EQU (\$-TEXT1) ;OR USE COUNT EQU 18 INSTEAD

TEXT2 DB COUNT DUP(?)

.CODE

.STARTUP

MOV CX,COUNT

MOV SI,0

MOV DI,0

ADD DI,COUNT

DEC DI

;POINT DI TO THE END OF TEXT2

REVERSE: MOV AL,TEXT1[SI]

;GET ‘P’

MOV TEXT2[DI],AL

;PLACE ‘P’ AT THE END OF STRING2



```
DEC DI  
INC SI  
LOOP REVERSE ;REPEAT UNTIL CX IS ZERO  
.EXIT  
END
```

REMEMBER:

TEXT1: POKHARA

TEXT2: ARAHKOP AFTER EXECUTION

YOU CAN MODIFY THE PROGRAM TO INPUT THE STRING FROM THE KEYBOARD AND DISPLAY THE REVERSE STRING MESSAGE. FOR THAT YOU WILL NEED TO USE FUNCTION NUMBER 0AH AGAIN



Program No 5: Write ALP to find the sum of two numbers which is input by the user through the keyboards and display the sum in the screen.

.MODEL SMALL

.STACK 100H

.DATA

MSG1 DB 10,13,'ENTER FIRST NUMBER: ','\$'

MSG2 DB 10,13,'ENTER SECOND NUMBER: ','\$'

MSG3 DB 10,13,'SUM IS: ','\$'

NUM1 DB ?

NUM2 DB ?

.CODE

.STARTUP

;PROMPT FOR ENTERING

MOV DX,OFFSET MSG1

MOV AH,09H

INT 21H



;GET THE NUMBER1 AND SAVE IN NUM1

MOV AH,01H

INT 21H

MOV NUM1,AL

;GET THE NUMBER2 AND SAVE IN NUM2

MOV DX,OFFSET MSG2

MOV AH,09H

INT 21H

MOV AH,01H

INT 21H

MOV NUM2,AL

;CLEAR AH, AND GET NUM1 IN AL. YOU WILL NEED AX TO USE AAA,

; SO AH HAS TO BE 00H

MOV AL,NUM1

MOV AH,00H

ADD AL,NUM2

;SUB AL,NUM2 ;USE THIS FOR SUBTRACTION

AAA

;AAS

ADD AX,3030H

;SEE PAGE 173 IN BARRY FOR DETAILS. NUMBERS ARE ADDED WITH ASCII

;ARITHMETIC. IT INVOLVES AX REGISTER. ANSWER IS IN BINARY. AND IT

;HAS TO BE CONVERTED TO ASCII IF YOU HAVE TO DISPLAY IT. THAT'S WHY

; 30H IS ADDED TO AH AND 30H IS ADDED TO AL



**;ANSWER IS IN AX, MSB ASCII IS IN AH, AND LSB ASCII IS IN AL.
;SAVE THOSE VALUES IN BH AND BL.BARRY USES PUSH
;INSTRUCTIONS TO SAVE AX. YOU CAN USE THAT WAY TOO**

MOV BL,AL

MOV BH,AH

MOV DX,OFFSET MSG3

MOV AH,09H

INT 21H

;DISPLAY HIGHER BYTE

MOV AH,02H

MOV DL,BH

INT 21H

;DISPLAY LOWER BYTE

MOV AH,02H

MOV DL,BL

INT 21H

.EXIT

END

(Please see Example 5-18 to Example 5-24 in the text book. Last example is important)



Program No 6: Write an ALP using 8086 instructions to find the factorial of a given number and store the result at effective address D000H

.MODEL SMALL

.STACK 100H

.DATA

NUM1 EQU 05H

;MAKE SURE ANS IS LESS THAN 255

.CODE

.STARTUP

MOV AL,1

MOV CL,0

LOOP1: INC CL

MUL CL

CMP CL,NUM1

JNZ LOOP1

MOV DS:[0D000H],AX

;PLEASE NOTE 0 BEFORE D IS NEEDED

.EXIT

END



Program No 7: Write an ALP using 8086 instructions to generate a square wave with period of 200 microseconds. Assume the address of the output device to be 55

```
.MODEL SMALL  
.STACK 100H  
.CODE  
.STARTUP
```

```
MOV DL,AA (4T)  
  
ROTATE: MOV AL,DL (2T)  
         ROR AL,1 (2T)  
         MOV DL,AL (2T)  
         AND AL,01H (4T)  
         OUT 55H,AL (10T)  
         MOV CX,COUNT (4T)
```

```
DELAY: DEC CX (3T)  
       JNZ DELAY (16T/4T)
```

```
JMP ROTATE (15T)
```

```
.EXIT
```

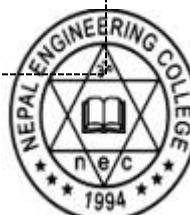
```
END (PLEASE SEE PAGE 794 ,APPENDIX B IN BARRY FOR DETAILS)
```

;LOAD BIT PATTERN
;LOAD BIT PATTERN IN A
;MOVE 0 TO 1 AND VICE-VERSA
;SAVE AL
;MASK ALL BIT EXCEPT D0
;TURN ON OR OFF THE WAVE
;LOAD COUNT

;DECREMENT THE COUNT
;REPEAT UNTIL CX=0

;GO BACK TO CHANGE LOGIC LEVEL

}
DELAY
LOOP



Delay calculations

$$\begin{aligned}\text{Delay Outside loop(T0)} &= (2T+2T+2T+4T+10T+4T+15T) \\ &= 39 \text{ T-states}\end{aligned}$$

$$\begin{aligned}\text{Assume clock speed of 8086} &= 10\text{MHz} \\ \Rightarrow T &= 1/f = 0.1 \mu\text{s}\end{aligned}$$

Thus,

$$T_0 = 39 * 0.1 \mu\text{s}$$

$$\begin{aligned}\text{Loop Delay(TL)} &= 19\text{T-states} * 0.1 \mu\text{s} * (\text{count}-1) + 7\text{T-States} * 0.1 \mu\text{s} \\ &= 1.9 * (\text{count}-1) \mu\text{s} + 0.7 \mu\text{s} \\ &= 1.9 \mu\text{s} * \text{count} - 1.2 \mu\text{s}\end{aligned}$$

$$\begin{aligned}\text{Total delay (TD)} &= T_0 + TL \\ 100 \mu\text{s} &= 3.9 \mu\text{s} + 1.9 \mu\text{s} * \text{count} - 1.2 \mu\text{s} \\ \Rightarrow \text{Count} &= 51.21D = 34H\end{aligned}$$

(Please note that you may ignore the last count T-states and simply take it as 19T-states inside the delay loop while calculating, Also you can use LOOP instruction in the program. In this case you should use (17T/5T) in your calculations)



Program No 8: Write an ALP to compare the two numbers located at memory address 2000H and 3000H. If the numbers are equal, load FFH at reg D else load 00H at same register.

.MODEL SMALL

.STACK 100H

.CODE

.STARTUP

MOV AL,DS:[2000H]

CMP AL,DS:[3000H]

JZ EQUAL

MOV DL,00H

;LOAD 00H IF NUMBERS ARE UNEQUAL

JMP FINISH

EQUAL: MOV DL,0FFH

FINISH:

.EXIT

END



Program No 9: Write an ALP for 8086 to add all the data of a table, which are between 50H and F0H. Display the result as the decimal value at output port 3000H. Assume table consists of array of 10 bytes of data.

.MODEL SMALL

.STACK 100H

.DATA

**TABLE1 DB 49H,49H,49H,49H,49H,51H
DB 0FFH,0FFH,0FFH,0FEH**

COUNT EQU 10

CARRY DB 00H

SUM DB 00H

.CODE

.STARTUP

MOV CL,COUNT

MOV BL,00H

MOV BH,00H

MOV SI,OFFSET TABLE1



NEXT: **MOV AL,DS:[SI]**
 CMP AL,50H
 JC SKIP
 CMP AL,0F0H
 JNC SKIP
 ADD AL,BL
 JNC SUM_ONLY
 INC BH ;COUNT CARRY IF ANY
SUM_ONLY: **MOV BL,AL** ;SAVE RESULT
SKIP: **INC SI**
 DEC CL
 JNZ NEXT
 MOV AL,BL
 MOV AH,BH
 MOV CARRY,AH ;NOT ASKED IN QUESTION.USE IT TO VERIFY
 MOV SUM,AL ; WHILE EXECUTING THE PROGRAM IN HOME
 ;IN THIS CASE; SUM IS 40H AND CARRY IS 01H
 OUT 3000H,AX
.EXIT
END



Program No 10: Write an ALP for 8086 to count the number of data bytes that are less than 20H from the block of data bytes of memory location 'ARRAY_DATA'. Store the result in memory location 'RESULT'

.MODEL SMALL

.STACK 100H

.DATA

**ARRAY_DATA DB 25H,0AH, 0AH, 0AH, 0AH, 0AH, 0AH, 0AH, 0AH, 0AH,25H
RESULT DB ?**

.CODE

.STARTUP

MOV BL,00H

MOV CX,0AH

MOV DI,0

**NEXT: MOV AL,ARRAY_DATA[DI]
CMP AL,20H
JNC SKIP
INC BL**



**SKIP: INC DI
LOOP NEXT
MOV RESULT,BL**

.EXIT

END

NOTE:

RESULT SHOULD BE 08 IN THIS CASE



Program No 11: Eight data bytes are stored from memory location D000H to D007H. Write an ALP to transfer the block of data to new location C000H to C007H

.MODEL SMALL

.STACK 100H

.DATA

.CODE

.STARTUP

MOV CX,8

MOV SI,0D000H ;POINT SI TO SOURCE

MOV DI,0C000H ;POINT DI TO DESTINATION

MOV AL,[SI] ;MOVE ELEMENT POINTED BY SI TO AL

MOV [DI],AL ;MOV AL TO MEMORY LOCATION POINTED BY DI

INC SI

INC DI

LOOP CX ;REPEAT UNTIL CX=0

.EXIT

END



Program No 12: Write an ALP for 8086 to find the number of positive and negative data in a series of ten bytes. Assume the series is stored at location DATA. Store the count at location RESULT1 and RESULT2

.MODEL SMALL

.STACK 100H

.DATA

DATA DB 8FH,8FH,9FH,9FH,9FH,9FH,93H,93H,69H,69H

POS1 DB 0

NEG1 DB 0

.CODE

.STARTUP

MOV CL,10

MOV SI,0

MOV DL,0

MOV DH,0

LOOP1: MOV AL, DATA[SI]

ROL AL,1

JNC POSITIVE

INC DH

JMP NEXT



POSITIVE:

INC DL

NEXT:

INC SI

DEC CL

JNZ LOOP1

MOV POS1,DL

MOV NEG1,DH

.EXIT

END

NOTE:

NEGATIVE=8 AND POSITIVE = 2 IN THIS CASE



Program No 13: Write an ALP for 8086 to convert ASCII value of a data to its hexadecimal equivalent and store the result at CONVERT

.MODEL SMALL

.STACK 100H

.DATA

DATA1 DB 39H

CONVERT DB ?

.CODE

.STARTUP

MOV AL,DATA1

SUB AL,30H

CMP AL,0AH

JC NUMBERS ;IF AL IS LESS DON'T SUBTRACT 07H

SUB AL,07H

NUMBERS: MOV CONVERT,AL

.EXIT

END



Program No 14: Write an ALP to swap two numbers in MASM

```
.MODEL SMALL
.STACK 100H
.DATA
    DATA1 DB 20H
    DATA2 DB 50H
.CODE
.STARTUP
    MOV AL,DATA1
    MOV BL,DATA2
    MOV DATA1,BL
    MOV DATA2,AL
.EXIT
END
```



Program No 15: Write an ALP to concatenate two different strings 'string1' and 'string2' and put the result in the 'string3'.

.MODEL SMALL

.STACK 100H

.DATA

STRING1 DB 10,13,'ENTER FIRST STRING(8 CHAR) LONG: ','\$'

STRING2 DB 10,13,'ENTERSECOND STRING(8 CHAR) LONG: ','\$'

STRING3 DB 10,13,'CONCATENATED STRING IS: ','\$'

BUF1 DB 10

DB 10 DUP(?),'\$'

BUF2 DB 10

DB 10 DUP(?),'\$'

BUF3 DB 16 DUP(?)

.CODE

.STARTUP



;PROMPT FOR FIRST STRING

MOV DX,OFFSET STRING1

MOV AH,09H

INT 21H

;SAVE STRING1 IN BUF1

MOV AH,0AH

MOV DX,OFFSET BUF1

INT 21H

;PROMPT FOR STRING2

MOV DX,OFFSET STRING2

MOV AH,09H

INT 21H

;SAVE STRING2 IN BUF2

MOV AH,0AH

MOV DX,OFFSET BUF2

INT 21H

;NOW START CONCATATION PROCESS

MOV CL,8

MOV SI,2

MOV DI,0



**;START PUTTING BUF1 VALUE TO BUF3
;BUT START READING FROM THE THIRD
;CHARACTER IN BUF1**

FIRST_HALF: MOV AL,BUF1[SI]

**MOV BUF3[DI],AL
INC SI
INC DI
DEC CL
JNZ FIRST_HALF**

**;START PUTTING BUF2 INTO BUF3. BUF2 SHOULD BE READ FROM
;THIRD CHARACTER. AND IN BUF3, THE CHARACTER SHOULD BE
;PUT AT THE EIGHTH PLACE**

**MOV CL,8
MOV SI,2
MOV DI,8**



```
SECOND_HALF: MOV AH, BUF2[SI]  
          MOV BUF3[DI], AL  
          INC SI  
          INC DI  
          DEC CL  
          JNZ SECOND_HALF
```

:DISPLAY CONCATENATED MESSAGE

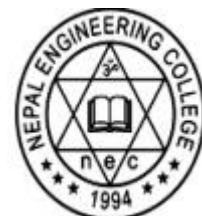
```
MOV DX,OFFSET STRING3  
MOV AH,09H  
INT 21H  
MOV DX, OFFSET BUF3  
MOV AH,09H  
INT 21H
```

```
.EXIT  
END
```

(THIS IS ONLY ONE WAY TO DO, YOU CAN FOLLOW BY DECLARING THE STRING INSIDE DATA SEGMENT AS WELL)



I KNOW YOU CAN PERFORM WELL IN YOUR UNIVERSITY EXAM



THE END



SAMPLE 8085 PROGRAMS



Program 1: Write a program for 8085 to arrange 10 bytes of data in ascending order. The data is stored in memory as an array starting from C100H

	MVI B,0AH	
	DCR B	;COUNTER 1
START:	MVI C,9H	;COUNTER 2
	LXI H,0C100H	;INITIALIZE THE POINTER
BACK:	MOV A,M	
	INX H	
	CMP M	;COMPARE TWO NUMBER
	JC SORTED	
	MOV D,M	;EXCHANGE THE TWO NUMBER
	MOV M,A	
	DCX H	
	MOV M,D	
	INX H	
SORTED:	DCR C	
	JNZ BACK	
	DCR B	
	JNZ START	;START WITH THE ELEMENT NUMBER 1
	HLT	

(Also see page 259 in the text book,5e)



Program No 2: A sequence of ten unsigned numbers is stored at memory location C0C0H. Develop an ALP to find out the greatest number with comment.

MVI C,9H	;COUNTER
LXI H,0C0C0H	;INITIALIZE
MOV A,M	
NEXT: INX H	;POINT TO NEXT ELEMENT
CMP M	;AND THEN COMPARE WITH THE CONTENT IN A
JNC SKIP	
MOV A,M	;IF CARRY,NUMBER IN “M” IS GREATER,MAKE ;THAT NUMBER A REFERENCE
SKIP: DCR C	
JNC NEXT	
HLT	

(NOTE THAT THE ANSWER IS IN A-REG,OUT 01H CAN BE USED TO DISPLAY IT JUST BEFORE THE HLT)



Program No 3: A railway crossing signal has two flashing lights run by 8085 microprocessor. One light is connected to data bit D7 and the second light is connected to D6. Stating necessary assumptions you make, write an assembly language program to turn each signal light alternately on and off at an interval of 1 second

Note1:

$100 \times 10 \text{ msec} = 1 \text{ sec}$, so load 100 in B-Reg, find COUNT value to be stored in the D-Register. Since the delay is greater than 0.5 sec, we need to use loop under loop or need to call delay more than once. In this case two loops are used. The inner loop creates the delay for 10 msec. So approximately the 1 sec delay is created. Note that the adjustments are neglected in the delay calculations. Also assume clock speed is 325ns

Delay Calculations(should be shown after the program is written)

$$T_{\text{INNER}} = (6+4+4+10)\text{T-states} \times 325\text{ns} \times (\text{COUNT}-1) + (6+4+4+7)\text{T-States} \times 325\text{ns}$$

$$10 \text{ msec} = 24\text{T-States} \times 325\text{ns} \times (\text{COUNT}-1) + 21\text{T-States} \times 325\text{ns}$$

$$\text{COUNT} = 1282.17$$

$$= 1283_{10}$$

$$= 503_{16}$$



MVI L, AAH	;7T load the bit patter
ROTATE: MOV A,L	; 4T load bit pattern in A
RLC	; 4T, change bit pattern from AA to 55 ;&vice-versa
MOV L,A	; 4T, save A
ANI C0H	; 7T Mask bits D5-D0
OUT PORT1	; 10T turn on or off D7 and D6
MVI B,100	; 4T delay 1 is equal to 100
OUTER: LXI D,COUNT	;10T
INNER: DCX D	;6T
MOV A,D	; 4T
ORA E	; 4T should create 10 msec delay
JNZ INNER	;(10/7T)
DCR B	; 4T
JNZ OUTER	;(10/7T)
JMP ROTATE	;10T

**(refer page 289 square wave example, question no 18 at page 294,
and its solution given at page 793 in the text book,5e)**

Program No 4: Write an 8085 assembly language program to add six bytes of data: 23H 41H, 56H, AFH, C5H and A7H and place the SUM and CARRY in memory location 2500H and 2501H respectively.

Note1:

Assume that the numbers are stored in the memory location starting from 2000H. Also assume that the sum does not exceed 16-bit

**LXI H,1FFFH
MVI B,06H
MVI C,00H
XRA A**

**;SET POINTER FOR DATA
;NUMBER OF DATA
;CLEAR C-REG TO ACCOUNT FOR CARRY
;INITIALIZE SUM = 0**

**REPEAT: INX H
ADD M
JNC AHEAD
INR C
AHEAD: DCR B
JNZ REPEAT
STA 2500H
MOV A,C
STA 2051H
HLT**

**;STORE SUM
;STORE CARRY**



Program No 5:Write a program to check the given number is odd or even

Note 1

Assume the number is at location 2500H. If odd, display 01 at port 01. If even display 0H at port 01. But this is not mentioned in the question

LDA 2500H,

RRC

JC ODD ;if there is carry, number is odd, so display 01H at port 01

MOV A,00H

JMP FINISH

ODD: MOV A,01H

FINISH: OUT 01H

HLT



Program No 6: Write an ALP to count number of 1's in a given byte.

Note 1:

Assume that the number is given at 2500H

LDA 2500H

MOV C,08H ; set C as a number of bits in a byte to count

MOV B,00H ; initialize the count to 0

REPEAT:RRC

JNC SKIP

INR B ;if there is carry, count it by incrementing B-reg

SKIP:DCR C

JNZ REPEAT

HLT



Program No 7: Write an assembly language program for 8085 to convert the BCD number at memory location 1200H into binary and store the result at memory location 1201H

$$72_{10} = 7 \times 10 + 2$$

$$\text{BCD}_1 = 7$$

$$\text{BCD}_2 = 2$$

$$\text{Thus Binary} = 10 \times \text{BCD}_1 + \text{BCD}_2$$

Hence Load the number first in A, and extract BCD1 and BCD2 in C and D register first. Then add BCD1 ten times and add with BCD2 to get the binary value.



LDA 1200H	;load the number
MOV B,A	;save the number in B
ANI F0H	; mask lower nibble
RRC	
RRC	
RRC	
RRC	
MOV D,A	;save BCD2 in D-Reg
MOV A,B	;get the number again
ANI 0FH	;mask the upper nibble
MOV C,A	;save BCD1 in C-Reg
XRA A	;clear accumulator
MVI E,0AH	;set E as multiplier of 10
MULTIPLY: ADD E	;add 10 until (D)=0
DCR D	
JNZ MULTIPLY	;is multiplication complete?
ADD C	;If not go back and add again
STA 1201H	
HLT	

(Please see page 324 for reference in text book)



Program No 8: Write a program for 8085, to examine the content of memory location 8050H to be even or odd. If the content is even, load FF H else load 00H in memory location 8060H

LDA 8050H	;LOAD THE DATA
RRC	;CHECK FOR 0 OR 1 IN THE BIT DO
JC ODD	
MVI A,00H	;NUMBER IS EVEN , SO LOAD 00H IN A
JMP SAVE	;SKIP NEXT INSTRUCTION
ODD: MVI A,0FFH	;NUMBER IS ODD, LOAD FFH IN A
SAVE: STA 8060H	
HLT	



Program No 9: Write a program to compare the two numbers located at memory address 2000h and 3000h. If the no equal load the FFH at reg. D else 00 at same register

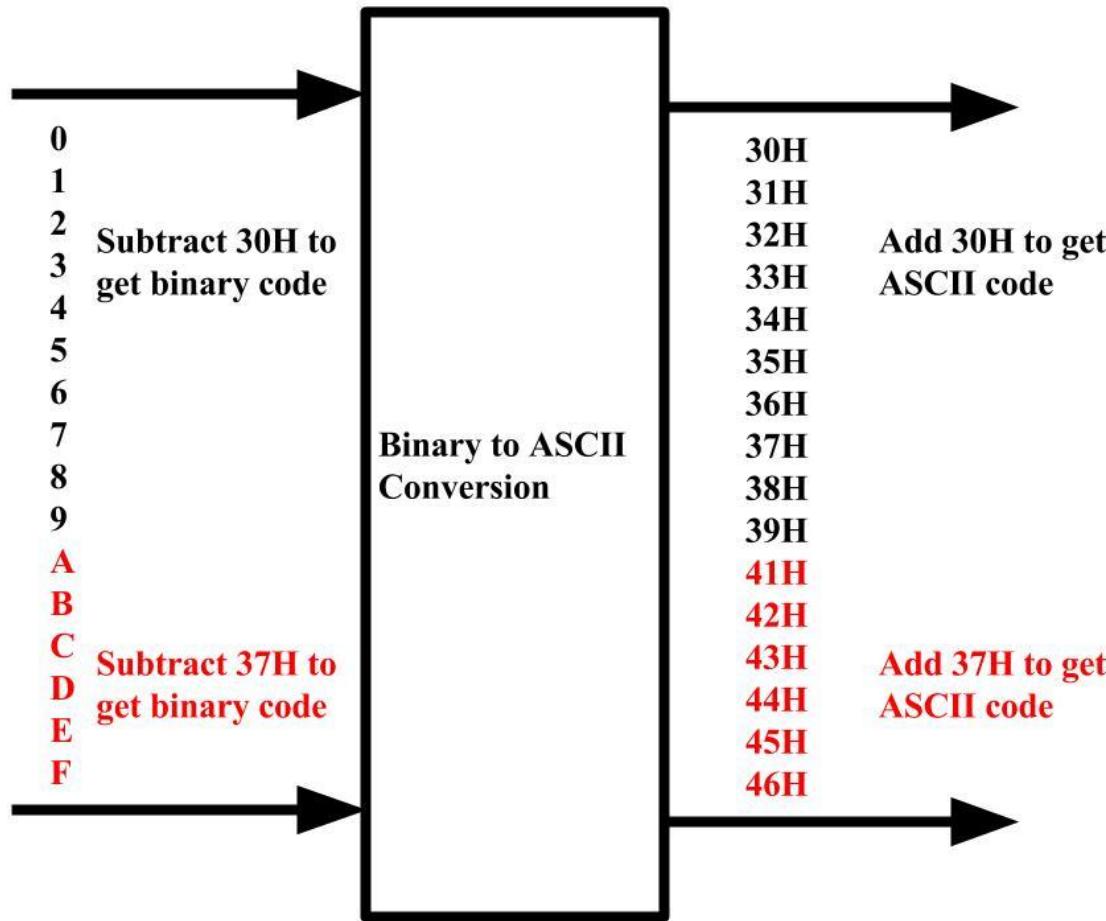
```
LXI H, 3000H
LDA 2000H
CMP M
JZ EQUAL

MVI A, 00H
JMP STORE
EQUAL: MVI A,FFH
STORE: MOV D,A
HLT
```

```
;POINT HL TO 3000H
;GET NUMBER FROM LOCATION 2000H TO A
;COMPARE TWO NUMBERS
;IF ZERO FLAG IS SET,TWO
;NUMBERS ARE EQUAL
;LOAD 00H IF NOT EQUAL
```



Binary to ASCII Conversion



Program No 10: Write a program to convert ASCII into its equivalent Binary

Note1:

Assume that the number is stored at 2500H. Store the result at 2501H

LDA 2500H	;GET THE NUMBER
SUI 30H	;SUBTRACT 30H FROM THE READ DATA
CPI 10D	
JC LETTERS	;IF NUMBER IS ABOVE 9, NEED TO SUBTRACT ;37H,IE EXTRA 07H
SUI 07H	;SUBTRACT 07H
LETTERS: STA 2501H	;STORE ASCII AT MEMORY
HLT	



Program No 11: Write a program to convert binary into its equivalent ASCII

Note1:

Assume that the number is stored at 2000H. Store the result at 2001H

LDA 2000H	;GET THE NUMBER
CPI 0AH	;IS DIGIT LESS THAN 10
JC CODE	;IF IT IS , GO TO CODE AND ADD 30H ONLY
ADI 07H	;ADD 7H TO OBTAIN
CODE: ADI 30H	;CODE FOR DIGIT FROM A TO F
STA 2001H	;ADD BASE NUMBER 30H
HLT	;STORE THE ASCII CODE IN 2001H

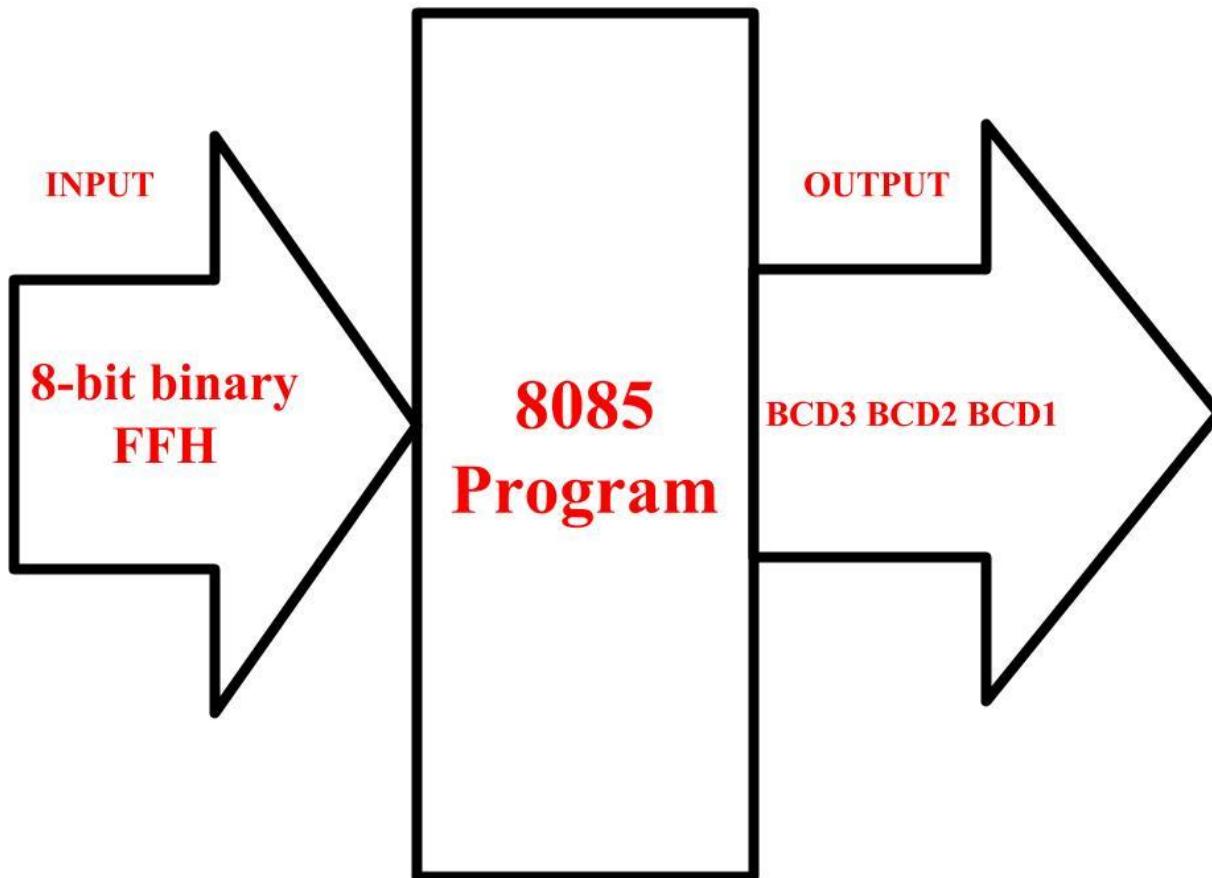


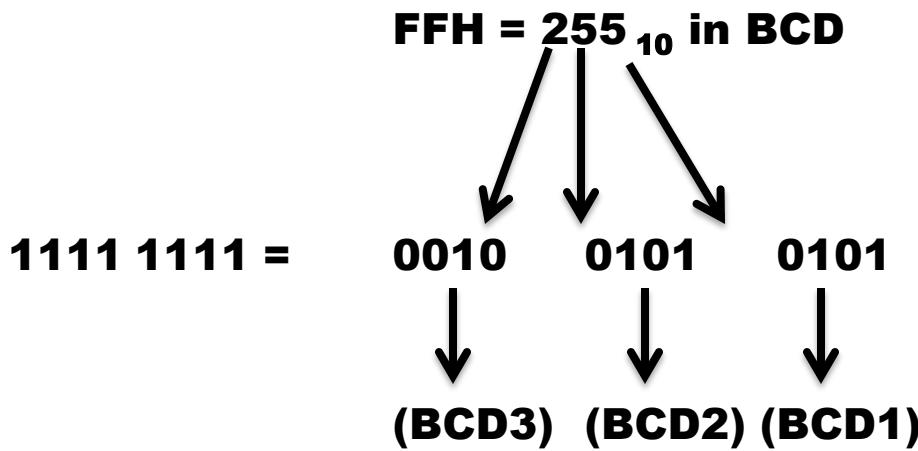
Program No 12: Write an 8085 assembly language program to check whether the content of memory location 2500H is even or odd number. If even display 01H at output port 60H else display 00H at the same port

LDA 2500H	;LOAD THE NUMBER
RRC	;BRING LSB TO CARRY POSITION TO CHECK
	;FURTHER FOR EVEN
JC ODD	
MOV A, 01H	;NUMBER IS EVEN,LOAD 01H
JMP DISPLAY	
ODD: MOV A, 00H	; NUMBER IS ODD,LOAD 00H
DISPLAY: OUT 60H	
HLT	



8-bit Binary to BCD conversion





Note1:

So we need 12 bits to convert an 8-bit binary number into a BCD equivalent. BCD2 and BCD1 can be stored in a single 8-bit memory location. BCD3 has to be stored in the next memory location. So assume that BCD2 and BCD1 will be stored at 2600H. Store the remaining BCD3 at location 2601H.

Note2:

If you look at page number 327 in the text book, you have a similar solution. Goankar saves all three digits BCD3, BCD2 , and BCD1 in different memory locations. Please use whichever method you find it easy to perform



Step1: Divide by 100

100) 255(2 \longleftarrow BCD3 (Quotient)

-100

55

255 Quotient

-100= 155 1

-100=55 1

} use E
as a counter

2 (BCD3)

Step2: Divide by 10

10) 55 (5 \longleftarrow BCD2 (Quotient)

-50

5 \longleftarrow BCD1 is the remainder

55 Quotient

-10=>45 1

-10=>35 1

-10=>25 1

-10=>15 1

-10=>05 1

} use D
as a counter

5(BCD2)

Step3: Remainder is the BCD1
i.e BCD1 = 05 in this case

Program No 13: Write an 8085 assembly language program to convert an 8-bit binary number into its equivalent BCD form

HUND: MVI E,00H MOV D,E LDA 2500H TEN: CPI 100 JC TEN SUI 100 INR E JMP HUND UNIT: CPI 10 JC UNIT SUI 10 INR D JMP TEN MOV C,A MOV A,D RLC RLC RLC RLC ADD C STA 2600H MOV A,E STA 2601H HLT	;CLEAR E-REG FOR HUNDRED ;CLEAR D-REG ALSO ;GET THE BINARY DATA IN A-REG ;IF DATA IS LESS THAN 100,JUMP TO TEN ;SUBTRACT 100 FROM DATA ; FOR EACH SUBTRACTION, INCREMENT E ;COMPARE WITH 10 ;SAVE THE UNIT IN c-REG(BCD1) ;GET BCD2 IN A-REG ;ROTATE BCD2 DIGIT TO UPPER NIBBLE, MAKE 05D TO 50D ;COMBINE BCD1 AND BCD2 ;SAVE BCD2 BCD1 IN MEMORY ;SAVE BCD3 IN MEMORY
---	---



Program No 14: Write an ALP using 8085 instruction, to generate a continuous square wave with the period of 250 μ s. Assume the clock period is 0.33 μ s and use bit D4 to output the square wave. Show the delay calculation also

Note1:

**Similar problem was done in class. See page 289 in text book,6e.
Use $T_D = 125 \mu\text{s}$ instead of $250 \mu\text{s}$ given in page 290 during your calculations**

Note2:

Also make sure to use bit D4 instead of bit D0 in your code. To do this, use the following code in instruction number 5 given in page 289

ANI 10H ; Mask all bits except D4

Note3:

Thus instead of solving this problem, we will see how to solve question number 17 given in page 294 of text book.



Rectangular Wave

**On-period
200 μ s**

**off-period
400 μ s**



Program No 15: Write a program to generate a rectangular wave with a 200 μ s on-period and a 400 μ s off-period

TURNON: MVI A,01H (7T)

OUT PORT1 (10T)

MVI B, COUNT1 (7T)

;BIT PATTERN TO TURN ON D0

;ON-PERIOD BEGINS

;COUNT FOR 200 μ s DELAY

LOOP1: DCR B (4T)

JNZ LOOP1 (10T/7T)

MVI A,00H (7T)

OUT PORT1 (10T)

MVI B,COUNT2 (7T)

;BIT PATTERN TO TURN OFF D0

;OFF-PERIOD BEGINS

;COUNT FOR 400 μ s DELAY

LOOP2: DCR B (4T)

JNZ LOOP2 (10T/7T)

JMP TURNON(10T)

;REPEAT BY DOING THE SAME THING

;AGAIN



Delay Calculations

Assume clock speed = 325.5 ns, discard 7T from 10T/7T for simplicity

On-period delay:

$$T = T_0 + T_L$$

$$\begin{aligned}200 \mu s &= (7T+10T+7T) \times 325.5 \text{ ns} + (4T+10T) \times 325.5 \text{ ns} \times \text{COUNT1} \\&= 24 \times 325.5 \text{ ns} + 14 \times 325.5 \text{ ns} \times \text{COUNT1} \\&= 7.812 \mu s + 4.557 \mu s \times \text{COUNT1}\end{aligned}$$

$$\Rightarrow \text{COUNT1} = 42.17$$

$$\approx 42_{10}$$

Off-period delay:

$$T = T_0 + T_L$$

$$\begin{aligned}400 \mu s &= (10T+7T+10T+7T) \times 325.5 \text{ ns} + (4T+10T) \times 325.5 \text{ ns} \times \text{COUNT2} \\&= 34 \times 325.5 \text{ ns} + 14 \times 325.5 \text{ ns} \times \text{COUNT2} \\&= 11.067 \mu s + 4.5 \mu s \times \text{COUNT2}\end{aligned}$$

$$\Rightarrow \text{COUNT2} = 86.429$$

$$\approx 85_{10}$$



Program No 16: Write an Assembly Language Program (ALP) using 8085 instructions to copy ten bytes of data from memory location COOOH – COO9H to new location D000H-D009H

MOV B,10D	;SET COUNTER
LXI H,0C000H	;POINT HL TO COOOH START ADDRESS
LXI D,0D00H	;POINT DE TO D000H NEW ADDRESS
NEXT: MOV A,M	
STAX D	
INX D	
INX H	
DCR B	
JNZ NEXT	;COPY UNTIL B-REG BECOMES ZERO
HLT	

(PLEASE SEE PAGE 239 FOR DETAILS)



CISC AND RISC COMPUTERS

There are two classes of computers called Complex Instructions Set Computer (CISC**) and Reduced Instruction Set Computer (**RISC**).**

Note1:

Most computers are designed with both RISC and CISC features. Hence they are neither RISC or CISC computers. Intel coined the word CRISC (Complex Reduced Instruction Set Computers) for such computers

Reference:

D V Hall, page number 1.9, third edition



CISC VS RISC

- | | |
|--|--|
| <ol style="list-style-type: none">1. Control unit is hardwired and microprogrammed2. Complex multiclock instructions3. Memory reference instructions4. Less registers in CPU5. Maximum memory addressing modes6. Small code sizes7. Variable instruction length8. INTEL processors (usually powers laptop and desktops) | <ol style="list-style-type: none">1. Control Unit is hardwired only2. Simple single clock instructions3. Register reference instructions except LOAD and STORE4. More registers in CPU5. Minimum memory addressing modes6. Large code sizes7. Fixed instruction Length8. ARM processors (usually powers smartphones and tablet PCs) |
|--|--|



8085 Vs 8086

- | | |
|---|---|
| <ol style="list-style-type: none">1. 8-bit processor2. 8-bit data bus3. 16 bit address bus4. Can address upto 64K memory location5. 8-bit flag6. No memory segmentation concept7. No concept of queue8. Basic clock speed is 3Mhz for 8085A, 5Mhz for 8085A-29. Less number of instruction set in comparison to 8086. For example no two 8-bit numbers can be multiplied or divided with 8085 instruction set directly10. Cannot work in multiprocessor environment11. 4 maskable interrupts and 1 non-maskable interrupt12. 1 Vcc and 1 GND pin13. Pin number 34 is IO/\overline{IO} | <ol style="list-style-type: none">1. 16-bit processor2. 16-bit data bus3. 20 bit address bus4. Can address 1M memory location5. 16-bit flag6. Memory segmentation concept introduced from 8086 onwards7. Internal architecture consists of 6 byte queue8. Frequency range of 8086 is 6-10MHz9. Large number of instruction set. For example multiplication and division of two 8-bit numbers can be done with a single instruction with 808610. Supports multiprocessor environment while 8086 works in maximum mode11. 1 maskable interrupt, and 1 non-maskable interrupt12. 1 Vcc but 2 GND pins13. Pin number 28 is M/ \overline{IO} |
|---|---|



Thank You

29 October 2017

Pramod Ghimire

Slide 601 of 601

