



Lecture_1



Microprocessors Systems

CSE 238

Prof. M. E I-Banna



CSE 238 ILOs

ILO #	Description
1	Understand the general working of computers.
2	Comprehend the internal structure of computers, CPU, memory and buses.
3	Comprehend the concept of pipelining and pre-fetch queue.
4	Know the 8088/8086 internal four types of registers, general purpose, index and pointers, segment, as well as flag registers.
5	Differentiate between logical and physical memory in accordance to the memory structure of 8086/8088 Intel's processors.
6	Define different memory segments in Intel's 8086/8088 processors.
7	Know the different addressing modes of Intel's processors.



CSE 238 **ILOs**

ILO #	Description
9	Comprehend and use the instruction set of 8086/8088 processors.
10	Hardware specifications and pin configuration of Intel's 8086/8088 microprocessors
11	Understand how memory and I/O ports can be interfaced with microprocessors.
12	Use a microprocessor kit to exercise the above gained skills by writing assembly codes to drive and test different operations.



Microprocessors Systems CSE 238

Term: **Spring 2024**

Instructors: Dr. M. El-Banna, Room: 4th floor EE-Building
Dr. Hania Farag, Room: 3rd floor EE Building

Classes : G3 and G4 Dr. Banna MON 02:00 - 04:40 G4 K5
TUE 11:10 - 02:50 G3 K5

Office Hours: Monday 12:00 – 02:00
Tuesday 10:00 – 11:00
Thursday 10:00 – 12:00



Microprocessors Systems CSE 238

Lab TAs : Eng. A. Shasly and TBD

Topics to be covered:

Software

Chapter 1: General knowledge of computers

Chapter 2: Introduction to the Intel's 8086/8088 microprocessor

Chapter 3: Addressing Modes

Chapter 4: Data movement instructions

Chapter 5: Logical and arithmetic instructions.

Chapter 6: machine coding

Chapter 7: Program control instructions

Hardware

Chapter 1: Hardware specifications of 8086/8088

Chapter 2: Memory Interfacing



Microprocessors Systems CSE 238

Text Book: The INTEL Microprocessors, Eighth Edition
Barry B. Brey

Grading Policy:

Midterm	30%
LAB	20%
Tutorials attendance/Quizzes	10% (5% +5%)
Final	40%



Microprocessors Systems CSE 238

Useful Codes and links:

Course code: **9dlbhy3**

Website : http://eng.staff.alexu.edu.eg/staff/mbanna/public_html/

Youtube channel: "BannaElectronics"

https://www.youtube.com/channel/UCaKy_YOLJaPieD4ve59LZEQ/playlists



General Knowledge of Computers

What is a computer composed of ?

1. CPU
2. Memory
3. I/O devices



The CPU connects with

1. Memory (RAM, ROM, and HD)
2. I/O devices (KB, Printers, and Monitor)

Using

1. Address Bus
2. Data bus
3. Control Bus



What is The CPU composed of ?

1. Registers
2. Arithmetic Logic Unit
3. Program Counter
4. Instruction Decoder



What is a Microprocessor ?

All four CPU components integrated into a single chip called a microprocessor in 1971

CISC Vs RISC



CISC

1. Large number of instructions
2. Different lengths
3. Multiple cycles
4. Complex operations
5. Interfaced Memory

RISC

- Small number of instructions
- Fixed lengths
- Single cycle
- Simple operations
- Built-in Memory



Chapter 1: Introduction

- There are two big microprocessors families:
 - Intel (80xxx).
 - Motorola (68xxx).
- The Intel Microprocessors are used in the PCs.
- In this course, we study the Intel μ Ps.

1.1. Evolution of the Intel's μ Ps.

Product	8080	8085	8086	8088	80286	80386	80486
Year introduced	1974	1976	1978	1979	1982	1985	1989
Clock rate (MHz)	2-3	3-8	5-10	5-10	6-16	16-33	25-50
No. Transistors	4500	6500	29000	29000	130000	275000	1.2 million
Physical memory	64K	64K	1M	1M	16M	4G	4G
External data bus	8	8	16	8	16	32	32
Address bus	16	16	20	20	24	32	32
Data type (bits)	8	8	8,16	8,16	8,16	8,16,32	8,16,32

1.1. Evolution of the Intel's μ Ps.

Product	P pro	P II	P III	P 4
Year introduced	1995	1997	1978	2000
Clock rate (MHz)	150-166	266-333	1000, 1G	3.2G
No. Transistors	21M	?	?	?
Physical memory	4-64G	64G	64G	64G
External data bus	64	64	64	64
Address bus	32-36	36	36	36
Data type (bits)	8,16,32,64	8,16,32,64	8,16,32,64	8,16,32,64

1.1. Evolution of the Intel's μPs.

Intel® Core™ Processors HX-Series (14th gen): SKU Comparison

Processor Number		Intel® Core™ i9 Processor 14900HX	Intel® Core™ i7 Processor 14700HX	Intel® Core™ i7 Processor 14650HX	Intel® Core™ i5 Processor 14500HX	Intel® Core™ i5 Processor 14450HX
Processor Cores (P-cores + E-cores) ¹¹		24 (8+16)	20 (8+12)	16 (8+8)	14 (6+8)	10 (6+4)
Processor Threads		32	28	24	14	10
Intel® Smart Cache (LLC)		36 MB	33 MB	30 MB	24 MB	20 MB
Max Turbo Frequency ¹⁰	P-core	Up to 5.8 GHz	Up to 5.5 GHz	Up to 5.2 GHz	Up to 4.9 GHz	Up to 4.8 GHz
	E-core	Up to 4.1 GHz	Up to 3.9 GHz	Up to 3.7 GHz	Up to 3.5 GHz	Up to 3.5 GHz
Overclocking ³	CPU	✓	✓	✓	✓	✓
	GFX	✓	✓	✓	✓	✓
	Memory	✓	✓	✓	✓	✓
Graphics Max Frequency		Up to 1.65 GHz	Up to 1.6 GHz	Up to 1.6 GHz	Up to 1.55 GHz	Up to 1.5 GHz
Processor Graphics		Intel® UHD Graphics				
Total PCIe Lanes		1x15 Gen5 + 1x4 Gen4 (CPU)				
Max Memory Speed ⁹		DDR5-5600, DDR4-3200				

1.1. Evolution of the Intel's μ Ps.

Intel® Core Desktop Processors Comparison

	Intel Core i9 Processors	Intel Core i7 Processors	Intel Core i5 Processors	
Max Turbo Frequency [GHz]	Up to 6.0	Up to 5.6	Up to 5.3	Up to 4.7
Intel® Turbo Boost Max Technology 3.0 Frequency [GHz]	Up to 5.8	Up to 5.6	N/A	N/A
Performance-core Max Turbo Frequency [GHz]	Up to 5.6	Up to 5.5	Up to 5.1	Up to 4.7
Efficient-core Max Turbo Frequency [GHz]	Up to 4.4	Up to 4.3	Up to 3.9	N/A
Processor Cores (P-cores + E-cores)	24 (8P + 16E)	20 (8P + 12E)	14 (6P + 8E)	4 (4P + 0E)
Intel® Hyper-Threading Technology	Yes			
Total Processor Threads	32	28	20	8
Intel® Thread Director	Yes			No
Intel® Smart Cache (L3) Size [MB]	36 MB	33 MB	24 MB	12 MB
Total L2 Cache Size [MB]	32 MB	28 MB	20 MB	5 MB
Max Memory Speed [MT/s]	Up to DDR5-5600 DDR4-3200			DDR5 4800 DDR4 3200

1.1. Evolution of the Intel's μ Ps.

- The execution time are given as:

8080	2 μ s
8085	1.3 μ s
8086/8088	400 ns

- The advantages of the 8086/8088:
 - Hardware multiplication and division.
 - Larger addressable memory space.
 - Large number of internal registers which are accessible in 200 ns.



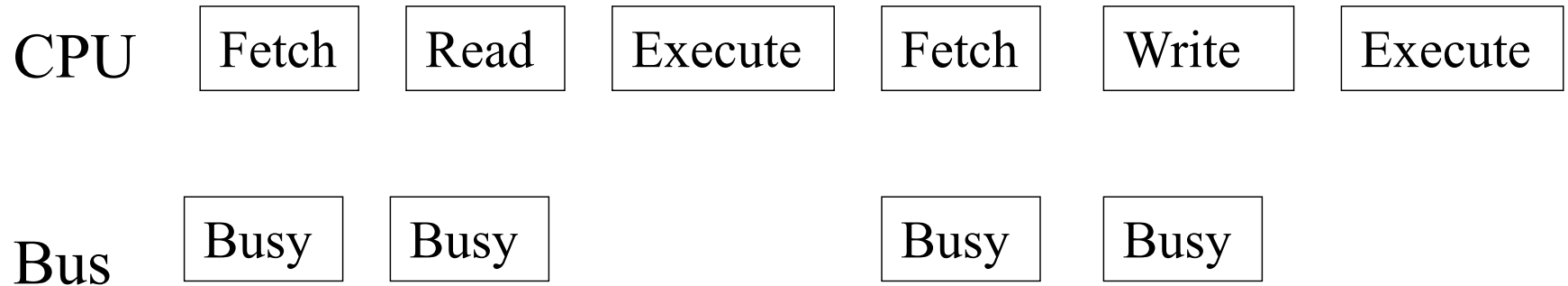
1.2. 8086/8088 Architecture

- The steps of fetching and executing the instruction:
 1. An instruction is fetched from memory, then it is decoded within the μP .
 2. Operands are read from/written to either the data memory segment or internal registers.
 3. The μP is now ready to execute the next instruction.



1.2. 8086/8088 Architecture

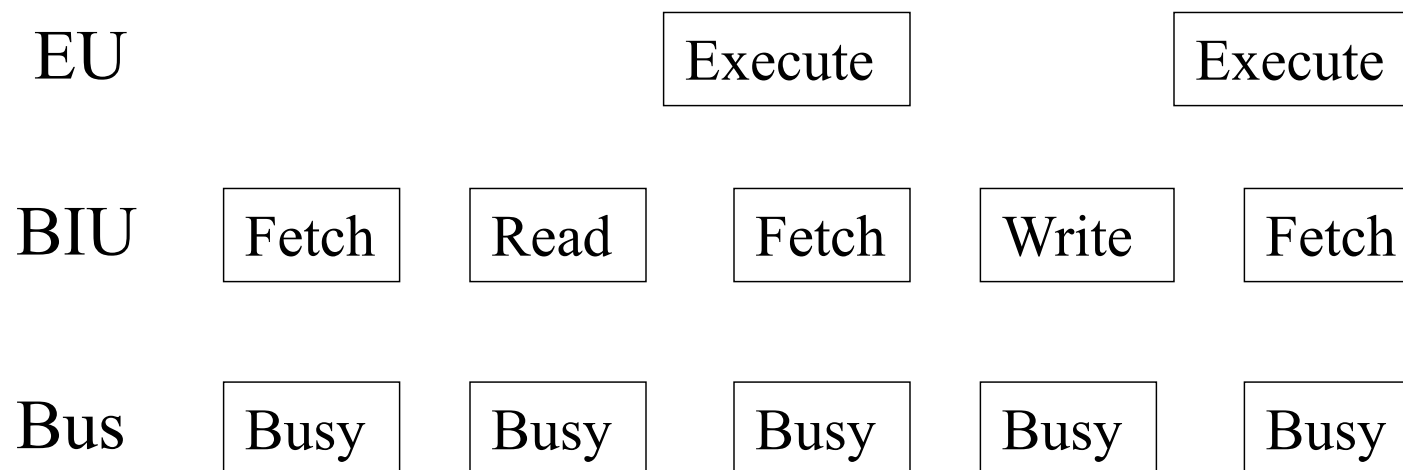
- The normal operation of an 8085 is



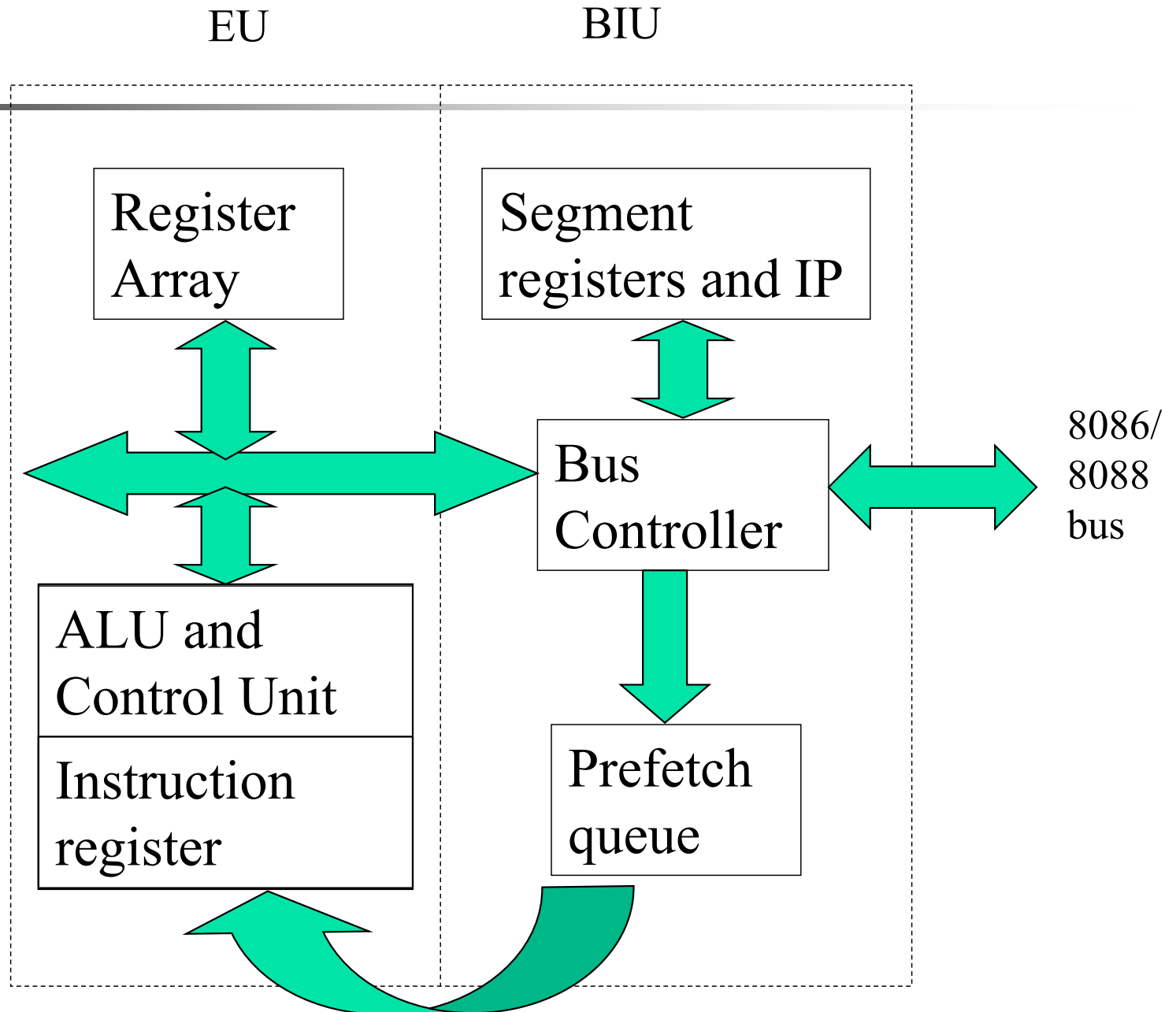


1.2. 8086/8088 Architecture

- The normal operation of an 8086/8088 depends on pipelining



2.1. Basic Internal Architecture



1.2.1.1. Bus Interface Unit (BIU)



- The main purposes of the BIU are:
 1. To keep the prefetch queue filled with instructions.
 2. To generate and accept the system control signals.
 3. To provide the system with the memory address or I/O port number.
 4. To act as window between the EU and memory for data.

1.2.1.1. Bus Interface Unit (BIU)



- The prefetch queue is FIFO memory.
- The 8086 queue is 2 byte-wide queue and 3 locations deep.
- The 8088 queue is a byte-wide queue and 4 bytes deep.

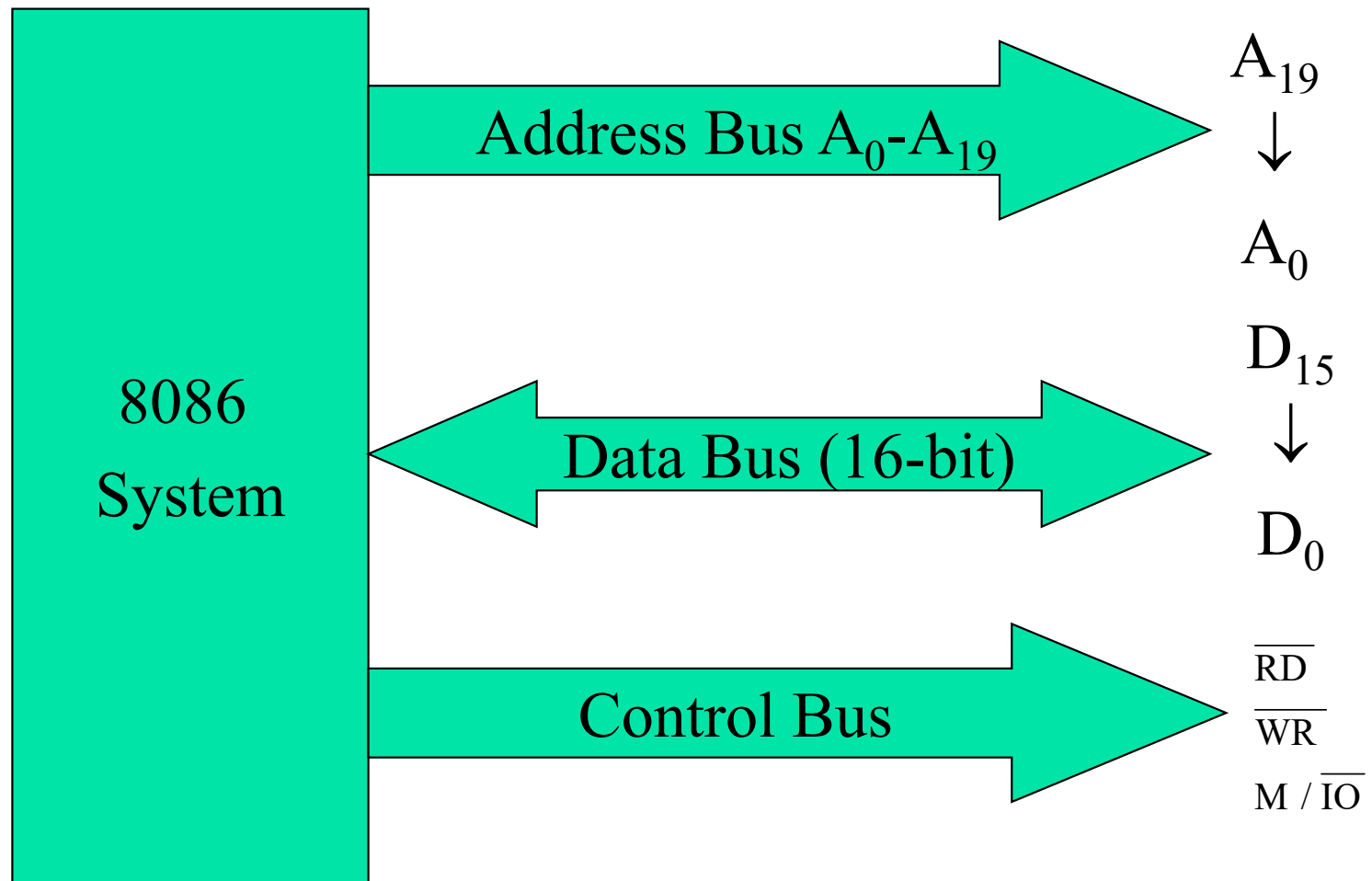


1.2.1.2. Execution Unit (EU)

- The EU carries out instructions that are fetched from the prefetch queue.
- The ALU performs arithmetic and logic operations on memory or register data.
- The register array holds information temporarily
- The instruction register
 1. Receives the instruction from the prefetch queue.
 2. Decodes the instruction to be executed.

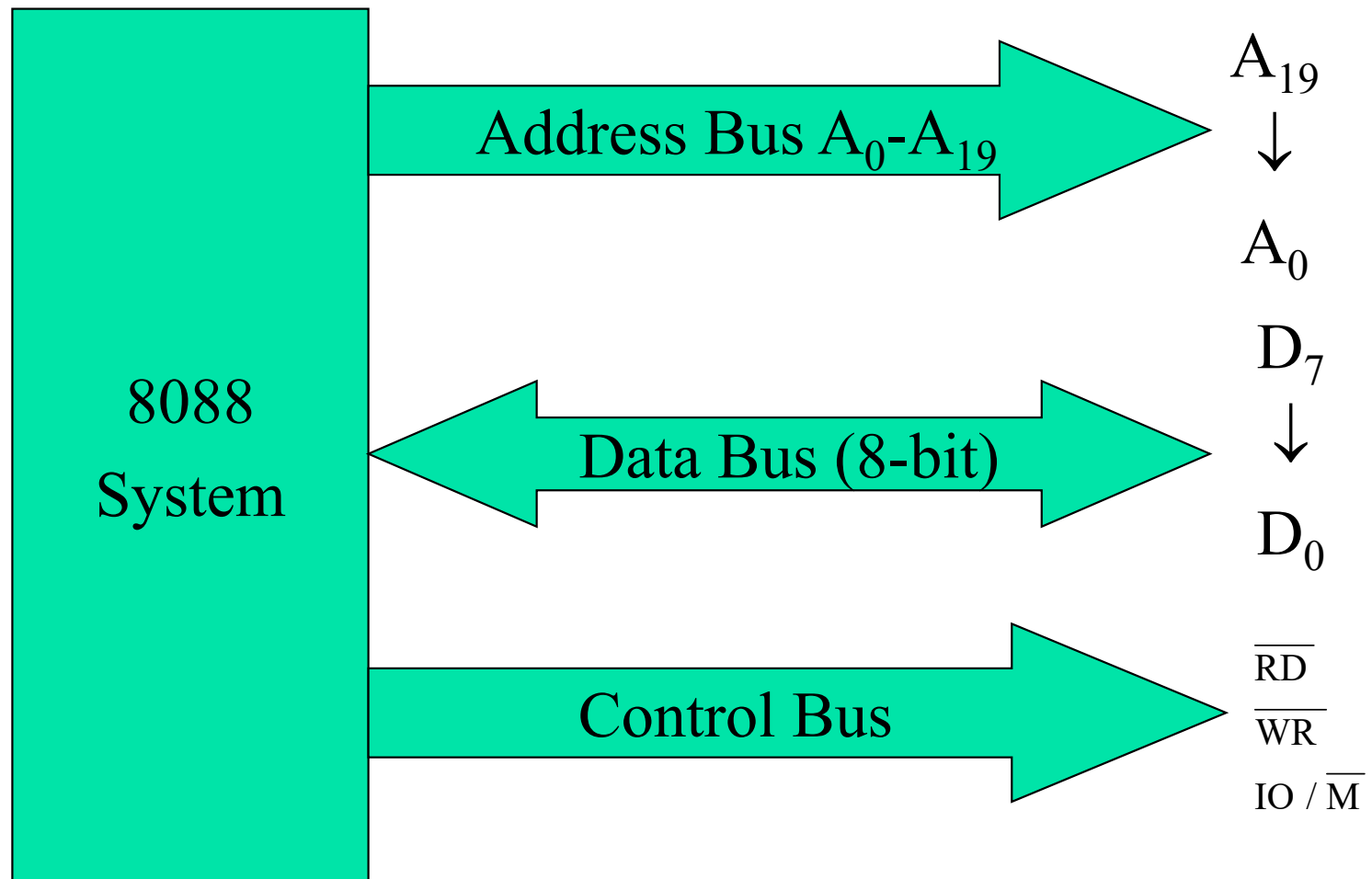
1.2.2. System Architecture

- The 8086 system



1.2.2. System Architecture

- The 8088 system





Lecture_2



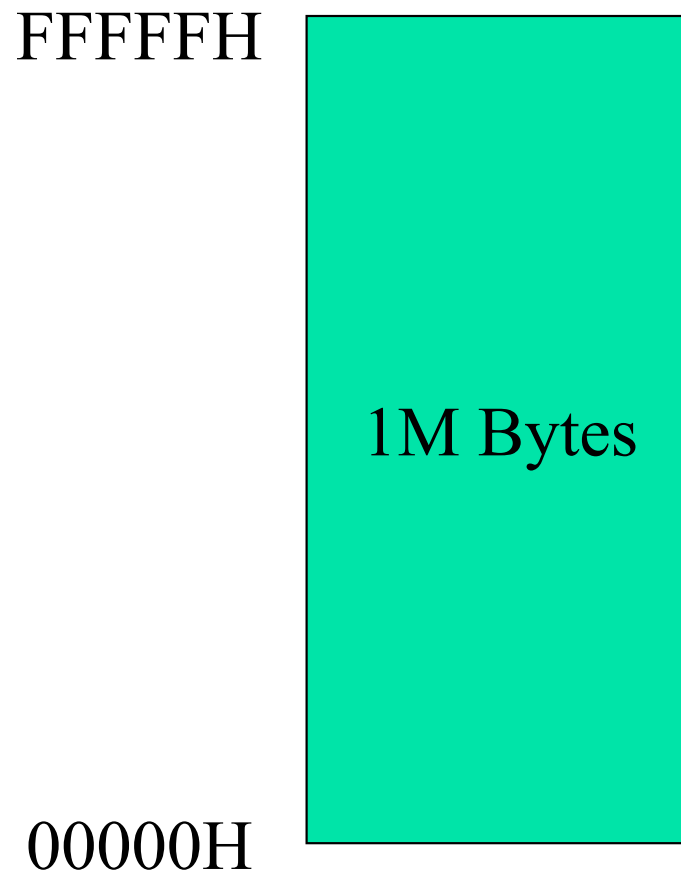
1.3. 8086/8088 Memory

- 8086/8088 has 1M (1,048,576) Byte.
- The memory can be studied from two points-of-view:
 - The programmer \Rightarrow Logical Memory.
 - The hardware designer \Rightarrow Physical Memory.



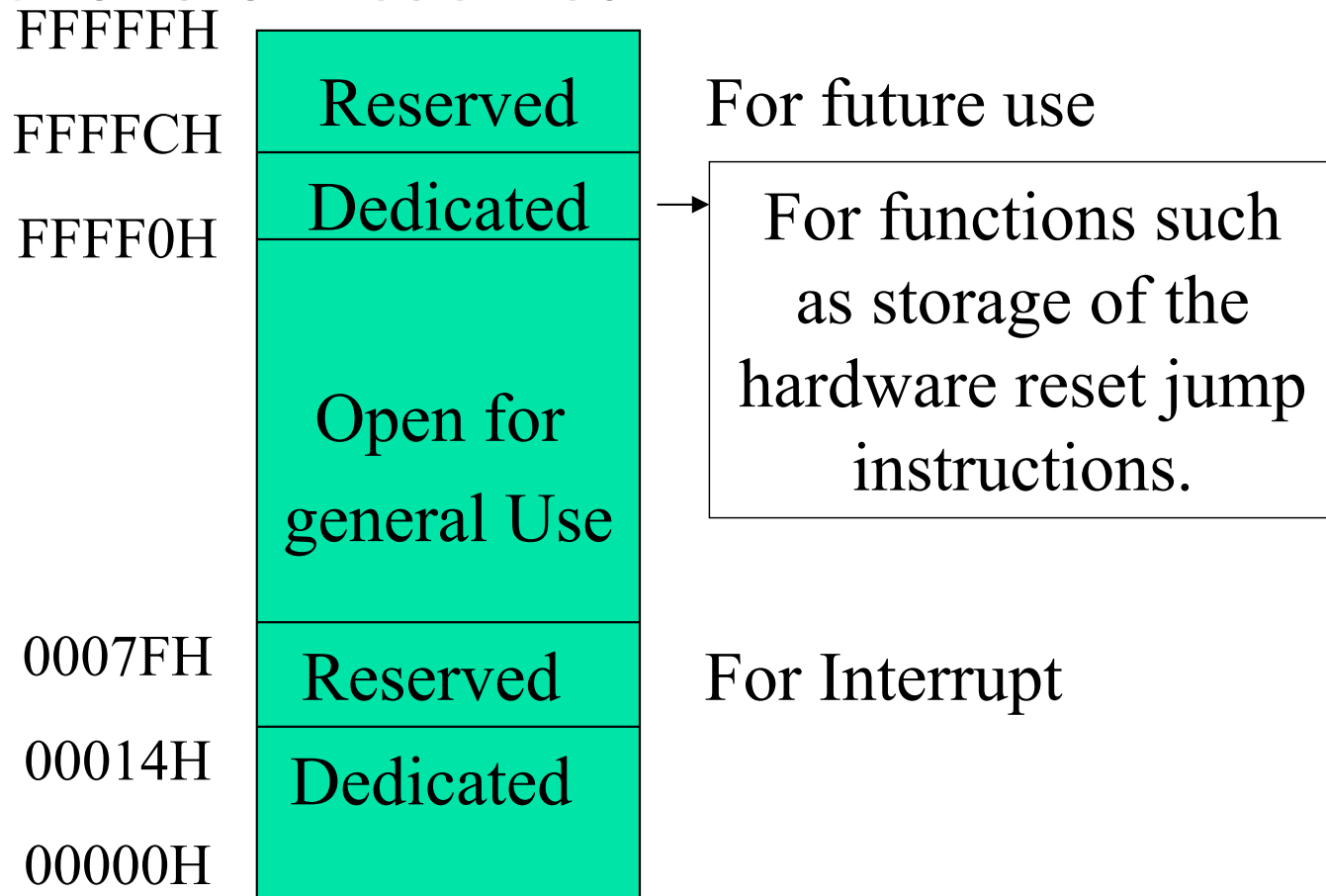
1.3.1. Logical Memory

- The logical memory is the same for both 8086 and 8088 μ Ps.



1.3.1. Logical Memory

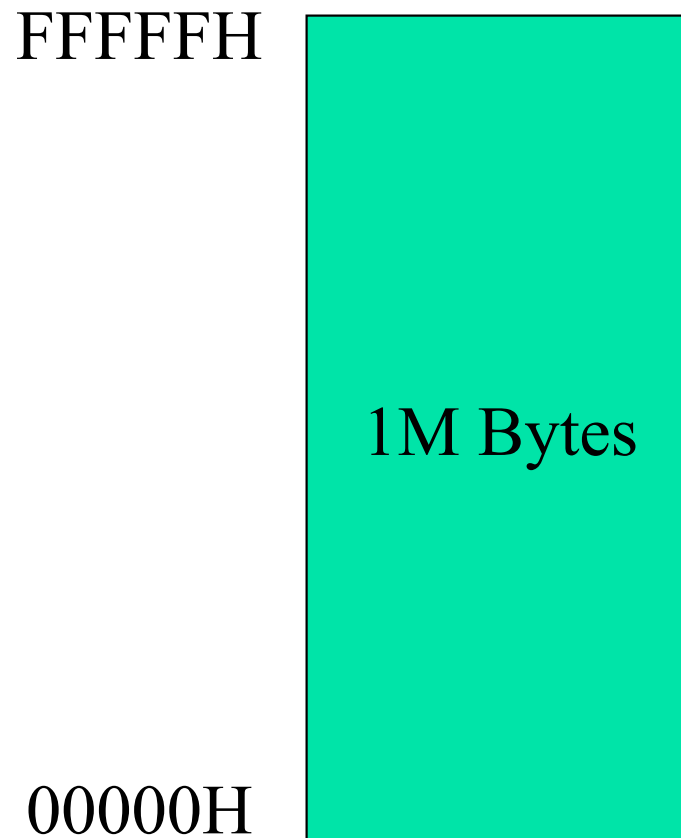
- Some address locations have dedicated functions or reserved.





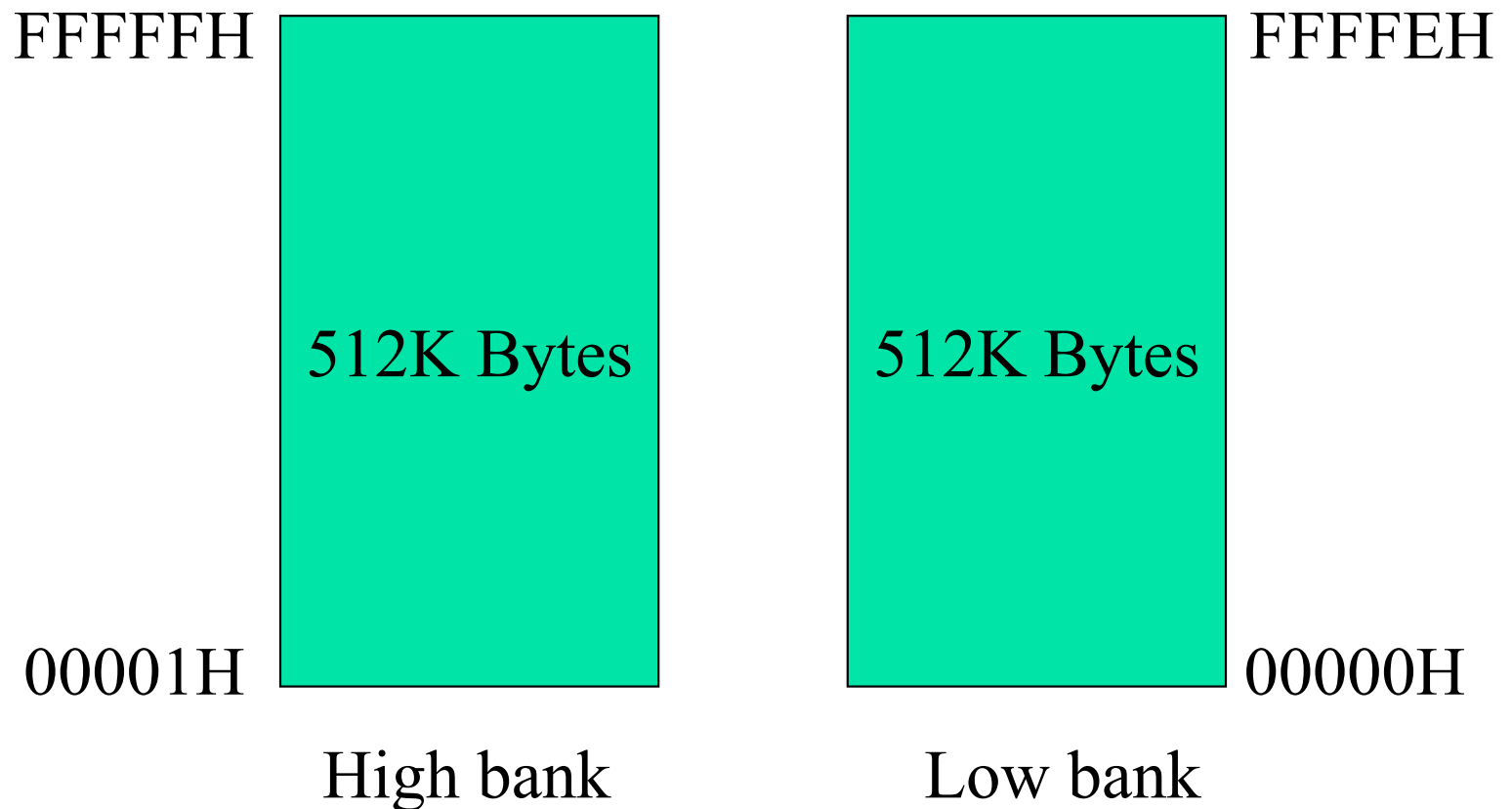
1.3.2. Physical Memory

- The physical memory of the 8088 μ P is identical to its logical memory.



1.3.2. Physical Memory

- The physical memory of the 8086 μ P





1.3.2. Physical Memory

- The advantages of this organization is:
 - The 8086 can address any byte or word of data.
- To transfer 16 bits from/to memory:
 - 8086 requires 1 operation.
 - 8088 requires 2 operations.
- The 8086 software executes more efficiently.



1.4. Registers

- The registers can be divided into:
 - Data or general purpose registers.
 - Pointer and index registers.
 - Segment registers.
 - Flag register.



1.4.1. General Purpose Registers

- There are 4 general-purpose data registers.
- Used for temporary storage of frequently used intermediate results.
- The advantage of storing the data in the internal registers instead of the memory is that they can be accessed much faster.

1.4.1. General Purpose Registers

- These registers can be used to store byte or word.

AX
BX
CX
DX

16 bits

AH	AL
BH	BL
CH	CL
DH	DL

8 bits

8 bits



1.4.1. General Purpose Registers

- The primary functions of these registers include:
 - AX (Accumulator)
 1. Used with the arithmetic and logic operation.
 2. Used with the I/O devices
 - BX (Base)
 1. Hold the base address of data located in the memory.
 2. Hold the base address of a table of data referenced by the translate instruction (XLAT).

1.4.1. General Purpose Registers

- The primary functions of these registers include:
 - CX (Count)
 1. Used as a counter for certain instructions such as shift rotate and loop.
 2. Used as a counter for the string operations.
 - DX (Data)
 1. Used with the arithmetic instruction such as 16-bit multiplication and division.
 2. Hold the I/O port number for a variable I/O instruction.

1.4.2. Pointers & Index Registers

- There are five pointer and index registers.

SP (Stack Pointer)

BP (Base Pointer)

IP (Instruction Pointer)

DI (Destination Index)

SI (Source Index)

1.4.2. Pointers & Index Registers



- The functions of these registers are:
 - SP (Stack pointer)
 1. Used to address data in a LIFO (last-in, first-out) stack memory.
 2. Used with the PUSH and POP instructions and the subroutines.
 - BP (Base pointer)
 1. Used to address an array of data in the stack memory.

1.4.2. Pointers & Index Registers



- The functions of these registers are:
 - SI (Source index)
 1. Used to address the data.
 2. Used with the string instructions.
 - DI (Destination index)
 1. Used to address the data.
 2. Used with the string instructions.

1.4.2. Pointers & Index Registers



- The functions of these registers are:
 - IP (Instruction Pointer)
 1. Used to address the next instruction executed.
 2. Every time an instruction is fetched from memory, the 8086/8088 updates the value in IP such that it points to the first byte of the next instructions.



1.4.3. Segment Registers

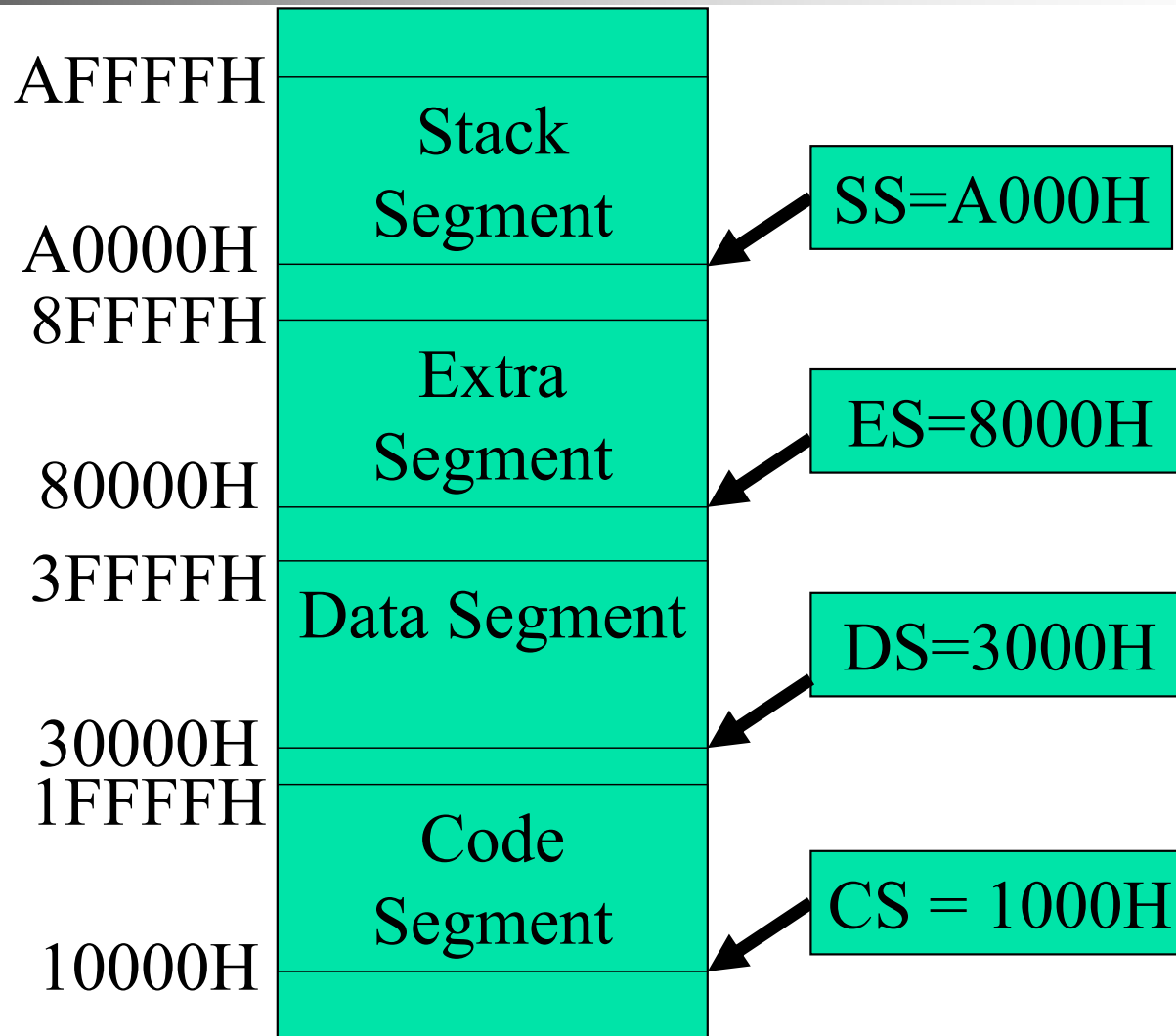
- The memory is 1M-byte.
- It can be partitioned into 64K (65536) byte segments.
- Not all this memory segments can be active at one time.
- Four segments only can be active at one time.
 1. Code segment \Rightarrow store the program or code.
 2. Data segment \Rightarrow store the data.
 3. Stack segment \Rightarrow store data as LIFO stack.
 4. Extra segment \Rightarrow used for string instructions.



1.4.3. Segment Registers

- Each segment is addressed independently by a special register called a segment register.
- There are 4 segments registers
 - CS
 - DS
 - SS
 - ES
- Each segment register identifies the segment's starting point or its lowest-addressed byte.
- Each register is 16-bit wide.

1.4.3. Segment Registers





1.4.3. Segment Registers

- Each segment should start at address ended by 0000B.
- The leftmost 16 bit are stored at the segment registers.
- Examples:

Segment Register	Memory Address Range
0100H	01000-10FFFH
1200H	12000H-21FFFH
2000H	20000H-2FFFFH



1.4.3. Segment Registers

- In the segment, data are addressed by the registers.
- CS \Rightarrow IP
- DS \Rightarrow BX, SI, DI.
- SS \Rightarrow BP, SP
- ES \Rightarrow DI



1.4.3. Segment Registers

- The address of any memory location is 20 bits.
- The size of the registers is 16 bits.
- Question:
How can the registers point to any memory location?



Lecture_3

1.4.3.1. Generating a memory address



- There are logical and physical address.
- A logical address is described by
 - The address stored at the segment register
 - An offset stored at the index registers, pointer registers, base registers or instruction pointer.
- Both the segment and the offset are 16 bits long.
- It can be written as: Segment:Offset
- Example: CS:0100H

1.4.3.1. Generating a memory address



- The physical address used to access memory should be 20 bits long.
- The physical address is computed as follows:
$$PA = \text{Segment} \times 10H + \text{Offset}.$$
- Example :
The segment register = 1000H and the Offset = 1234H
The physical address = $1000 \times 10 + 1234 = 11234H.$

1.4.3.1. Generating a memory address



- Example1:
CS = 1000H, DS = 3000H, SS = A000H, ES = 8000H
BX = 0200H.
- The physical address pointed to by the BX is
 $DS \times 10 + BX = 30200H$.



1.4.4. Flag Register

- The flag or status register is 16-bit register within the 8086/8088.
- The state of these flags indicates the conditions that are produced as the result of executing an arithmetic or logic instruction.
- There are nine flags divided into two groups:
 - Status flags: CF, PF, AF, ZF, SF and OF.
 - Control flags: TF, IF and DF.



1.4.4. Flag Register

- The flags are arranged as shown:

				O	D	I	T	S	Z		A		P		C
--	--	--	--	---	---	---	---	---	---	--	---	--	---	--	---

- The function of these flags are:

1. Carry flag (CF)

CF = 1 if there is a carry or borrow.

CF = 0 otherwise

2. Parity flag (PF)

PF = 1 if the result contains an even number of 1's.

PF = 0 if the result has an odd number of 1's.



1.4.4. Flag Register

- The function of these flags are:
- 3. Auxiliary carry flag (AF)
 - AF = 1 if there is a carry-out or a borrow-in between the low and high nibble.
 - AF = 0 if there is no carry-out or borrow-in.
- 4. Zero flag (ZF)
 - ZF = 1 if the result is zero.
 - ZF = 0 if the result is not zero.
- 5. Sign flag (SF)
 - SF = 1 if the result is negative number.
 - SF = 0 if the result is positive number.



1.4.4. Flag Register

- The function of these flags are:
 6. Trap flag (TF)
 - TF = 1 if the μ P works in the single-step mode.
 - TF = 0 if the 8086/8088 works in normal mode.
 7. Interrupt flag (IF)
 - IF = 1 if the μ P enables the maskable interrupt.
 - IF = 0 if the μ P disables the maskable interrupt.



1.4.4. Flag Register

- The function of these flags are:
 8. Direction flag (DF)

DF = 1 if the string instruction automatically decrements the address.
DF = 0 if the string instruction automatically increments the address.
 9. Overflow flag (OF)

OF = 1 if the signed result is out of range.
OF = 0 if the signed result is within the range.



1.5. Data Formats

- The data can be presented as:
 - ASCII
 - BCD
 - 8-bit signed and unsigned integers.
 - 16-bit signed and unsigned integers.
 - 32-bit signed and unsigned integers.
 - Short and long real numbers (floating-point numbers).



1.5.1. ASCII Data

- Used to represent alphanumeric characters.
- It is 7-bit code.
- In some systems, 8th bit holds the parity.
- In the printer system,
 - The 8th bit holds 0 for alphanumeric.
 - The 8th bit holds 1 for graphics characters.

1.5.1. ASCII Data

TABLE 2.6 ASCII and EBCDIC Codes in Hex.

Character	ASCII	EBCDIC	Character	ASCII	EBCDIC	Character	ASCII	EBCDIC	Character	ASCII	EBCDIC
@	40		a	60	81	blank	20	40	NUL	00	
A	41	C1	b	61	82	!	21	5A	SOH	01	
B	42	C2	c	62	83	"	22	7F	STX	02	
C	43	C3	d	63	84	#	23	7B	ETX	03	
D	44	C4	e	64	85	\$	24	5B	EOT	04	37
E	45	C5	f	65	86	%	25	6C	ENQ	05	
F	46	C6	g	66	87	&	26	50	ACK	06	
G	47	C7	h	67	88	'	27	7D	BEL	07	
H	48	C8	i	68	89	(28	4D	BS	08	16
I	49	C9	j	69	8A)	29	5D	HT	09	05
J	4A	D1	k	6A	91	*	2A	5C	LF	0A	25
K	4B	D2	l	6B	92	+	2B	4E	VT	0B	
L	4C	D3	m	6C	93	,	2C	6B	FF	0C	
M	4D	D4	n	6D	94	-	2D	60	CR	0D	15
N	4E	D5	o	6E	95	.	2E	4B	SO	0E	
O	4F	D6	p	6F	96	/	2F	61	SI	0F	
P	50	D7	q	70	97	0	30	F0	DLE	10	
Q	51	D8	r	71	98	1	31	F1	DC1	11	
R	52	D9	s	72	99	2	32	F2	DC2	12	
S	53	E2	t	73	A2	3	33	F3	DC3	13	
T	54	E3	u	74	A3	4	34	F4	DC4	14	
U	55	E4	v	75	A4	5	35	F5	NAK	15	
V	56	E5	w	76	A5	6	36	F6	SYN	16	
W	57	E6	x	77	A6	7	37	F7	ETB	17	
X	58	E7	y	78	A7	8	38	F8	CAN	18	
Y	59	E8	z	79	A8	9	39	F9	EM	19	
Z	5A	E9	{	7A	A9	:	3A		SUB	1A	
[5B			7B		;	3B	5E	ESC	1B	
\	5C		}	7C	4F	<	3C	4C	FS	1C	
]	5D		~	7D		=	3D	7E	GS	1D	
^	5E		DEL	7E	07	>	3E	6E	RS	1E	
_	5F	6D				?	3F	6F	US	1F	



1.5.2. BCD

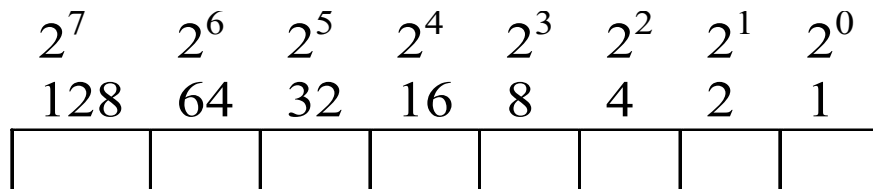
- Binary Coded Decimal (BCD) is 4-bit binary codes from 0000 (0) to 1001 (9).
- It can be stored as:
 - Packed BCD \Rightarrow 2 BCD digits / memory byte.
 - Unpacked BCD \Rightarrow 1 BCD digit / memory byte.



1.5.3. Byte

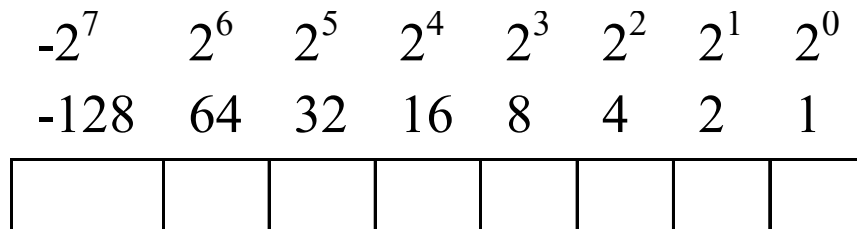
- Byte data are stored as:

1. Unsigned



0 → 255

2. Signed



-128 → 127



1.5.3. Byte

- The negative numbers are presented in its 2's complement.
- Example:
 - If the value is 81H
As unsigned byte, $81H = 129_{10}$.
As signed byte, $81H = (-128+1)_{10} = -127_{10}$.



1.5.4. Word

- To store the word in the memory,
 - It takes two bytes (two memory locations).
 - The least significant byte of the word is stored at the lower-addressed byte.
 - The most significant byte is stored at the higher-addressed byte.
- Example: Store 1234H at address 10000H

10001H	12
10000H	34



1.5.5. Double Word

- Double word is formed as 4 bytes.
- Example: Store 12345678H at address 10000H.

10003H	12
10002H	34
10001H	56
10000H	78



1.5.5. Double Word

- Example: Store DS:Offset = 1000H:2356H at the address 10000H

10003H	10
10002H	00
10001H	23
10000H	56



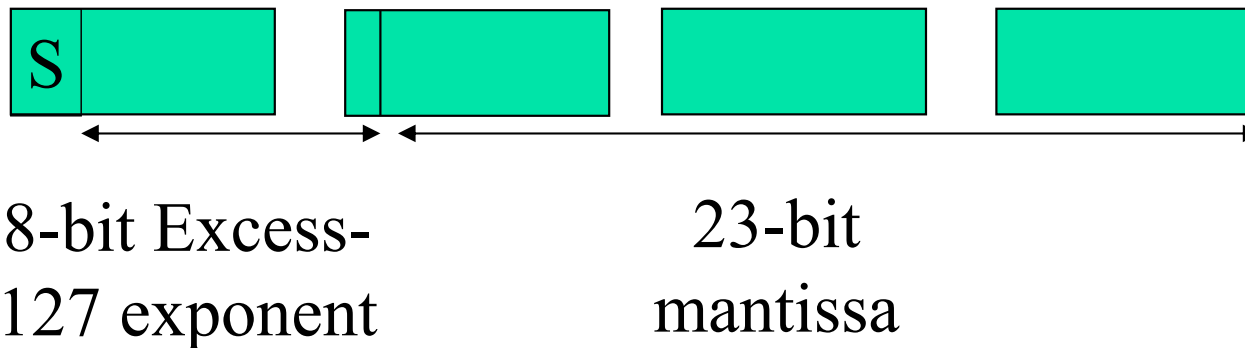
1.5.6. Real Numbers

- A real or floating-point number is composed of:
 - A mantissa
 - An exponent.
- Example: Decimal $12 = 1100$
It can be written as 1.1×2^3
The mantissa is 1 and the exponent is 3.

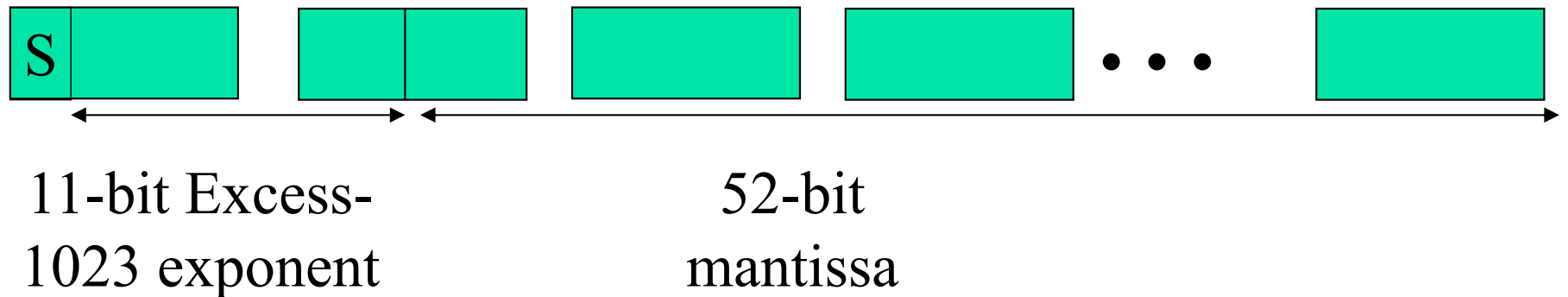
1.5.6. Real Numbers

- It can be stored as:

- 4-byte short form.



- 8-byte long form.





1.5.6. Real Numbers

- Examples:

Decimal	Binary	Normalized	Sign	Biased Exponent	Mantissa
+12	1100	1.1×2^3	0	10000010	1000000 00000000 00000000
-12	1100	-1.1×2^3	1	10000010	1000000 00000000 00000000
+100	1100100	1.1001×2^6	0	10000101	1001000 00000000 00000000
-1.75	1.11	-1.11×2^0	1	01111111	1100000 00000000 00000000
0.25	.01	1.0×2^{-2}	0	01111101	0000000 00000000 00000000
0.0	0	0	0	00000000	0000000 00000000 00000000



Lecture_4



Chapter 2: Addressing Modes

- In this chapter, we will discuss:
 - The addressing modes.
 - The stack.
- There are two types of the addressing modes:
 - Data addressing modes.
 - Program memory addressing modes.



2.1. Data Addressing Modes

- MOV instruction is one of the simplest and most flexible instruction.
- It is written as:
 MOV Destination, Source
- Move (copy) the data from the source to the destination.
- The addressing modes discuss how to describe the source and the destination.



1.2. Data Addressing Modes

- Data addressing modes are:
 - a) Register Addressing.
 - b) Immediate Addressing.
 - c) Direct Addressing.
 - d) Register Indirect Addressing.
 - e) Base-Plus-Index Addressing.
 - f) Register-Relative Addressing.
 - g) Base-Relative-Plus-Index Addressing.
 - i) Port Addressing.



2.3. Register Addressing

- It is used to transfer a byte or word from the source register to the destination register.
- The 8-bit registers are AH,AL,BH,BL,CH,CL,DH,DL.
- The 16-bit registers are AX,BX,CX,DX,SP,BP,SI,DI,CS,DS,SS,ES.
- The rules of using this addressing mode:
 - a) The source and destination registers have the same size.
 - b) It is not allowed to MOV from one segment to another.



2.3. Register Addressing

- Examples on the register addressing

Assembly Language	Operation
MOV AL,BL	BL → AL
MOV CH,CL	CL → CH
MOV AX,CX	CX → AX
MOV SP,BP	BP → SP
MOV DS,AX	AX → DS
MOV SI,DI	DI → SI
MOV DI,SI	SI → DI
MOV BX,ES	ES → BX
MOV CS,DS	Not allowed
MOV BL,BX	Not allowed
MOV AX,DL	Not allowed



2.3. Register Addressing

- Example on the register addressing

MOV BX,CX

Ex: Before execution

AX	
BX	76AF
CX	1234
DX	

After execution

AX	
BX	1234
CX	1234
DX	



2.4. Immediate Addressing

- It is used to transfer the immediate byte or word of data to the destination register.
- The data is stored immediately in the instruction.
- Examples

Assembly Language	Operation
MOV BL,44	2CH → BL
MOV AX,44H	0044H → AX
MOV SI,0	0000H → SI
MOV CH,100	64H → CH
MOV SP,3000H	3000H → SP



2.4. Immediate Addressing

- Example on the Immediate addressing:
`MOV AX, 3456H`

Before execution

AX	6291 H
BX	
CX	
DX	

After execution

AX	3456H
BX	
CX	
DX	



2.5. Direct Addressing

- It is used to transfer a byte or word between the memory and a register.
- The memory address of the data is stored by its effective memory address or the offset.
- The physical address of the data is computed as: $\text{Segment Register} \times 10\text{H} + \text{EA}$.
- The effective address can be written as:
 - Direct (Label)
 - Displacement



2.5.1. Direct Addressing

- The effective address is presented by the label.
- Example: MOV AX, BETA
- This stands for “move the contents of the memory location labeled as BETA into the register AX”. The physical address of the data is $DS \times 10H + BETA$ and $DS \times 10H + BETA + 1$.
- Examples:

Assembly Language	Operation
MOV AL,NUMBER	The contents of the memory location $DS \times 10H + NUMBER$ is copied, a byte, into AL.
MOV THERE,AX	The content of the AX is copied to the memory location whose address is $DS \times 10H + THERE$ and $DS \times 10H + THERE + 1$

2.5.1. Direct Addressing

- Example: MOV AL, NUMBER1
- Where NUMBER1 = 1234H and DS = 1000H
- The PA = DS × 10H + 1234 = 11234H
- Only with the AX, AL and AH registers

Ex: Before execution

AH	AL	XX	XX
----	----	----	----

Memory

Address	Content
11234	12
11235	

After execution

AH	AL	XX	12
----	----	----	----

2.5.2. Displacement Addressing

- The effective address is given in the instruction.
- Example: `MOV CX, [1234H]`
- This stands for “move the contents of the memory location whose effective address is 1234H into the register CX”. The physical address of the data is $DS \times 10H + 1234H$ and $DS \times 10H + 1234H + 1$.
- Examples:

Assembly Language	Operation
<code>MOV CL,[2000H]</code>	The contents of the memory location $DS \times 10H + 2000H$ is copied a byte into CL.
<code>MOV THERE,BX</code>	The content of the BX is copied to the memory location whose address is $DS \times 10H + THERE$ and $DS \times 10H + THERE + 1$

2.5.2. Displacement Addressing

- Example: MOV CX, [2000H]
where DS = 1000H
- Physical address = DS × 10H + 2000H = 12000H

Before execution

BH	BL	XX	XX
CH	CL	XX	XX

Memory

Address	Content
12000	ED
12001	BD
12002	
12003	

After execution

BH	BL	XX	XX
CH	CL	BD	ED



2.6. Register Indirect

- It is used to transfer a byte or word between a register and the memory location addressed by a register.
- The data is addressed at the memory location pointed to by any of the following registers: BX, BP, SI and DI.
- An Example of this mode is given as: MOV AX,[SI]
- This stands for “move the content of memory location whose effective address is stored in SI to the register AX”.
- The physical address is $DS \times 10H + SI$ and $DS \times 10H + SI + 1$.



2.6. Register Indirect

- The following table illustrates several MOV instructions using register indirect mode.

Assembly Language	Operation
MOV CX,[BX]	The contents of the memory location $DS \times 10H + BX$ and $DS \times 10H + BX + 1$ are copied as word into CX.
MOV [BP],DL	The content of the DL is copied to the memory location whose address is $SS \times 10H + BP$.
MOV [DI],BH	The content of the BH is copied to the memory location whose address is $ES \times 10H + DI$.

2.6. Register Indirect

■ Example: MOV AX, [SI]

Before execution

AX	XXXX
SI	1234

Memory: $DS \times 10 + SI = 11234$

Address	Content
11234	56
11235	87

After execution

AX	8756
SI	1234

■ Example: MOV CX, [BP]

Before execution

CX	XXXX
BP	2000

Memory: $SS \times 10 + BP = 22000$

Address	Content
22000	56
22001	87

After execution

CX	8756
BP	2000

2.7. Base-Plus-Index Addressing



- It is used to transfer a byte or word between a register and the memory location addressed by a base register plus an index register.
- The base register holds the beginning address of a memory array.
- The index register holds the relative position of the data in the array.

2.7. Base-Plus-Index Addressing

- Example: `MOV [BX + SI], AL`

“move the content of the register AL to the memory location whose effective address is the sum of the content of SI and the content of BX”.

- The physical address = $DS \times 10H + BX + SI$
- Example:

Assembly Language	Operation
<code>MOV CX,[BX+SI]</code>	The contents of the memory location $DS \times 10H + SI + BX$ and $DS \times 10H + SI + BX + 1$ are copied as word into CX.
<code>MOV [BP+DI],DL</code>	The content of the DL is copied to the memory location whose address is $SS \times 10H + BP + DI$.

2.8. Register Relative Addressing



- It is used to transfer a byte or word between a register and the memory location indicated by the sum of the content of a register and a direct or indirect displacement.
- The register can be BX, BP, SI or DI.
- Example: `MOV [BX +BETA], AL`

This stands for “move the content of the register AL to the memory location whose effective address is the sum of BETA and the content of BX”.

The physical address is $DS \times 10H + BX + BETA$.

2.8. Register Relative Addressing

- Examples:

Assembly Language	Operation
MOV CX,[BX]+BETA	The contents of the memory location $DS \times 10H + BX + BETA$ and $DS \times 10H + BX + BETA + 1$ are copied as word into CX.
MOV [BP+NEWS],DL	The content of the DL is copied to the memory location whose address is $SS \times 10H + BP + NEWS$.
MOV CX,ARRAY[SI]	The contents of the memory location $DS \times 10H + SI + ARRAY$ and $DS \times 10H + SI + ARRAY + 1$ are copied as word into CX.
MOV [DI]BETA,DL	The content of the DL is copied to the memory location whose address is $DS \times 10H + DI + BETA$.

2.8. Register Relative Addressing

- Examples: MOV [BX]+BETA, AL

Before execution

AX	BEED
BX	1000

Memory: $DS \times 10 + BX + BETA = 11234$

Address	Content
11234	ED
11235	87

After execution

AX	BEED
BX	1000

Memory After = Memory Before "87ED"

2.9. Base-Relative-Plus-Index Addressing

- It is used to transfer a byte or word between a register and the memory location addressed by a base register plus an index register plus a displacement.

- An Example: `MOV [BX][SI]BETA,AL`

This stands for “move the content of the register AL to the memory location whose effective address is the sum of BETA and the content of SI and the content of BX”.

The physical address = $DS \times 10H + BX + SI + BETA$.

2.9. Base-Relative-Plus-Index Addressing

■ Examples

Assembly Language	Operation
MOV CX,[BX+SI+BETA]	The contents of the memory location $DS \times 10H + SI + BX + BETA$ and $DS \times 10H + SI + BX + BETA + 1$ are copied as word into CX.
MOV BETA[BP+DI],DL	The content of the DL is copied to the memory location whose address is $SS \times 10H + BP + DI + BETA$.

2.9. Base-Relative-Plus-Index Addressing

- Example: MOV CX,[BP][DI]BETA

Before execution

DI	0200
CX	XXXX
BP	0400

Memory: $SS \times 10 + BP + DI + BETA = 20000 + 0400 + 0200 + 1234 = 21834$

Address	Content
21834	FC
21835	AB
21836	
21837	

After execution

DI	0200
CX	ABFC
BP	0400

Summary

Type	Instruction	Source	Address Generation	Destination
Register	MOV AX,BX	Register BX		Register AX
Immediate	MOV CH,3AH	Data 3AH		Register CH
Direct	MOV [1234H],AX	Register AX	$DS \times 10H + DISP$ 10000H + 1234H	Memory address 11234H
Register indirect	MOV [BX],CL	Register CL	$DS \times 10H + BX$ 10000H + 0300H	Memory address 10300H
Base-plus-index	MOV [BX+SI],BP	Register SP	$DS \times 10H + BX + SI$ 10000H + 0300H + 0200H	Memory address 10500H
Register relative	MOV CL,[BX+4]	Memory address 10304H	$DS \times 10H + BX + 4$ 10000H + 0300H + 4	Register CL
Base relative-plus-index	MOV ARRAY[BX+SI],DX	Register DX	$DS \times 10H + ARRAY + BX + SI$ 10000H + 1000H + 0300H + 0200H	Memory address 11500H
Scaled index	MOV [EBX+2×ESI],AX	Register AX	$DS \times 10H + EBX + 2 \times ESI$ 10000H + 00000300H + 00000400H	Memory address 10700H

Notes: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, and DS = 1000H



Lecture_5

Chapter 4.

Data Movement Instructions

- In this chapter, we will discuss the data movement instructions.
- These instructions are provided to move data either between its internal registers or between an internal register and a memory location.
- All these instructions do not affect the flags.



4.1. MOV

- MOV is used to transfer a byte or a word of data from a source operand to a destination operand.
- The following table shows this instruction:

Mnemonic	Meaning	Format	Operation	Flags affected
MOV	Move	MOV D,S	(S) → (D)	None



4.1. MOV

- These operands take different forms as shown in the following table:

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Segment Register	Register-16
Segment Register	Memory-16
Register-16	Segment Register
Memory	Segment Register



4.1. MOV

- The rules of the MOV instruction:
 - It cannot transfer data directly between two memory locations.
 - It cannot transfer data between two segment registers.
 - The source and the destination have the same size.



4.2. PUSH/POP

- ✦ Stack segment has 64K bytes.
- ✦ SS register is used to store the lowest address in the stack segment.
- ✦ SP is used to point to the top of the stack.
- ✦ There are four instructions to move the word data between the registers and the stack:
PUSH, POP, PUSHF and POPF.



4.2. PUSH/POP

- The following table shows these instructions

Mnemonic	Meaning	Format	Operation	Flags affected
PUSH	Push word onto stack	PUSH S	$((SP)) \leftarrow (S)$	None
POP	Pop word from stack	POP D	$(D) \leftarrow ((SP))$	None
PUSHF	Push flags onto stack	PUSHF	$((SP)) \leftarrow (\text{flags})$	None
POPF	Pop flags from stack	POPF	$(\text{flags}) \leftarrow ((SP))$	None

✦ The operand take the following forms:

- ◆ Register
- ◆ Memory
- ◆ Segment registers
- ◆ Flag Register



4.3. Load Effective-Address

- ✦ There are three load effective-address instructions used to load a register or a register and a segment register with an address.
- ✦ The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load effective address	LEA Reg16,EA	(EA)→(Reg16)	None
LDS	Load register and DS	LDS Reg16,Mem32	(Mem32)→(Reg16) (Mem32+2)→(DS)	None
LES	Load register and ES	LES Reg16,Mem32	(Mem32)→(Reg16) (Mem32+2)→(ES)	None



4.3.1. LEA

- It is used to load a register with the address of the data specified by the operand, not the data.
- By comparing an LEA with a MOV:
 - `LEA BX,[DI]` ; loads the address specified by `[DI]` into the BX register `BX=DI`.
 - `MOV BX,[DI]` ;load the data stored at the memory location addressed by DI into BX.
- `LEA BX,[DI]` is equivalent to `MOV BX,DI`.



4.3.1. LEA

- Example 1: LEA CX, [DI]

Before Execution: CX = 1000H, DI = 2000H

After Execution: CX = 2000H, DI = 2000H

- Example 2: LEA CX, [BX+DI]

- If BX = 1000H and DI = 2000H

After the execution CX = BX + DI = 3000H.

- If BX = 1000H and DI = FF00H

After the execution CX = BX + DI = 0F00H.

4.3.2. LDS and LES

Mnemonic	Meaning	Format	Operation	Flags affected
LDS	Load register and DS	LDS Reg16,Mem32	(Mem32)→(Reg16) (Mem32+2)→(DS)	None
LES	Load register and ES	LES Reg16,Mem32	(Mem32)→(Reg16) (Mem32+2)→(ES)	None

- They load a 16-bit register with an offset address and either the DS or ES segment register with a new segment address.
- They use any of the valid memory addressing modes.



4.3.2. LDS and LES

- Example 1:

What is the result of executing: LDS SI,[200]?

Before Execution: DS = 1200H, SI = 0001H

Address	Contents
12203	13
12202	00
12201	00
12200	20

$$PA = DS \times 10 + 200 = 12200H$$

After Execution: SI = 0020H, DS = 1300H



4.3.2. LDS and LES

- Example 2:

What is the result of executing: `LES BX,[DI]`?

Before Execution:

`BX = 0000H, DS = 1000H, DI = 1000H`

$PA = DS \times 10 + DI = 11000H$

Address	Contents
11003	89
11002	30
11001	6F
11000	2A

After Execution: `BX = 6F2AH, ES = 8930H`



4.4. String Data Transfers

- There are three string data transfer instructions: LODS, STOS and MOVS.
- Each instruction allows data to be transferred as a block or group or as a single byte or word.
- These instructions use:
 - SI to point to the source data.
 - DI to point to the destination data.
 - D flag to select the auto-increment (D=0) or auto-decrement (D = 1) mode of operation for SI and DI during the string operation.



4.4. String Data Transfers

- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
MOVS	Move String	MOVS	$((ES)0+(DI)) \leftarrow ((DS)0+(SI))$ $(SI) \leftarrow (SI) \pm 1 \text{ or } 2$ $(DI) \leftarrow (DI) \pm 1 \text{ or } 2$	None
MOVSB	Move String Byte	MOVSB	$((ES)0+(DI)) \leftarrow ((DS)0+(SI))$ $(SI) \leftarrow (SI) \pm 1$ $(DI) \leftarrow (DI) \pm 1$	None
MOVSW	Move String Word	MOVSW	$((ES)0+(DI)) \leftarrow ((DS)0+(SI))$ $((ES)0+(DI)+1) \leftarrow ((DS)0+(SI)+1)$ $(SI) \leftarrow (SI) \pm 2$ $(DI) \leftarrow (DI) \pm 2$	None
LODS	Load String	LODS	$(AL \text{ or } AX) \leftarrow ((DS)0+(SI))$ $(SI) \leftarrow (SI) \pm 1 \text{ or } 2$	None
STOS	Store String	STOS	$((ES)0+(DI)) \leftarrow (AL \text{ or } AX)$ $(DI) \leftarrow (DI) \pm 1 \text{ or } 2$	None



4.4. String Data Transfers

- Example: What is the result of executing: LODSW?
Before execution: DS = 1000H, SI = 1000H,
AX = XXXXH, DF = 0

Address	Contents
11002	FA
11001	A0
11000	32

After the execution: DS = 1000H, SI = 1002H,
AX = A032H, DF = 0



4.4. String Data Transfers

- In most applications, the string operations must be repeated in order to process arrays of data.
- This is done by inserting a repeat prefix before the instruction that is to be repeated.
- The repeat prefix is shown:

Prefix	Used With	Meaning
REP	MOVS, STOS	Repeat while not end of string. CX \neq 0.



4.4. String Data Transfers



Example 1: What is the function of this program?

LES DI,LIST1 ;Loads DI,ES by the first address
in the destination list (LIST1).

LDS SI,LIST2 ;Loads SI,DS by the first address
in the source list (LIST2).

CLD ;DF = 0 so select auto-increment

MOV CX,100 ;Loads the counter CX = 100

REP MOVSB ;Transfers 100 bytes from LIST2
to LIST1.

This program moves 100 bytes from LIST2 to LIST1



4.4. String Data Transfers

✦ Example2: What is the function of this program?

LES DI,BUFFER ;Loads DI,ES by the first address
in the destination list.

MOV CX,10 ;Loads the counter CX = 10

CLD ;DF = 0 so select auto-increment

MOV AL,0 ;Clears AL

REP STOSB ;Stores AL in the memory location
addressed by [DI]

This program clears 10 memory locations whose starting
address is BUFFER



4.4. String Data Transfers

✦ Example3: What is the function of this program?

LES DI,BUFFER ;Loads DI,ES by the first address
in the destination list.

MOV CX,5 ;Loads the counter CX = 5

CLD ;DF = 0 so select auto-increment

MOV AX,0 ;Clears AX

REP STOSW ;Stores AX in the memory location
addressed by [DI]

This program clears 10 memory locations whose starting
address is BUFFER

4.5. Miscellaneous Data Transfer Instructions

- These instructions are:
 - XCHG
 - LAHF and SAHF
 - IN and OUT
 - XLAT



4.5.1. XCHG

- It exchanges the contents of any register with the contents of any register or memory location.
- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(D)↔(S)	None

✦ The allowed operands for these instructions are shown in the following table:

Destination	Source
Accumulator	Register-16 bits
Memory	Register
Register	Register



4.5.1. XCHG

- Example: If $BX = 11AAH$, $DS = 1200H$ and the memory location addressed by SUM contains $1E87$ where $SUM = 1234H$.

What is the result of executing the following instruction: `XCHG SUM,BX`.

$$PA = DS \times 10H + SUM = 12000 + 1234 = 13234H$$

After the execution: $BX = 1E87H$ and the memory locations (34 & 35) = $11AAH$



4.5.2. LAHF and SAHF

- LAHF and SAHF transfer the least significant flag byte to and from the AH register.
- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
LAHF	Load AH from flags	LAHF	(AH) ← (Flags)	None
SAHF	Store AH into flags	SAHF	(Flags) ← (AH)	SF,ZF,AF,PF,CF



4.5.2. LAHF and SAHF



Example: Trace the following program.

MOV AL,0 ;Clears AL

LAHF ;Loads AH by the flags.

XCHG AH,AL ;Exchanges the data between AH
and AL.

SAHF ;Stores AH into the flags.

This program clears the SF,ZF,AF,PF,CF.



4.5.3. IN and OUT

- The IN and OUT instructions are used to transfer the data between the I/O device and the microprocessor.
- This transfer goes through the AL or AX
- They use the port addressing mode.



4.5.4. XLAT

- The XLAT (translate) instruction has been provided to simplify implementation of the lookup table operation.
- The following table shows this instruction:

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT Source-table	$((AL)+(BX)+(DS)0) \rightarrow (AL)$	None

✦ The procedure of this instruction:

- ◆ It adds the contents of AL to the contents of the BX register to form a memory address in the data segment.
- ◆ It loads the data stored at this address into the AL.



4.5.4. XLAT

Example: BCD-to-7-segment LED display

The 7-segment LED display lookup table is stored at location `TABLE = 1000H` in the data segment where `DS = 4000H`.

Write a program to convert the contents of the accumulator into 7-segment code. Assuming that the 7-segment code is already stored at `TABLE`.

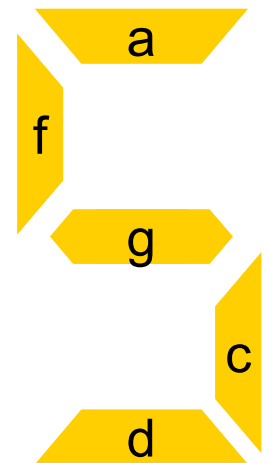
4.5.4. XLAT

 The program is:

```
MOV BX, TABLE    ;Initialize BX by the offset address
MOV AL,05H        ;Load AL by 5H (5 BCD number)
XLAT               ;AL will be loaded by the 7-
                  ;segment code equivalent to 5.

                  ;PA = DS × 10+ BX + AL.
```

PA contains 6D (01101101)
h g f e d c b a





Lecture_6

Chapter 5.

Arithmetic and Logic Instructions

- In this chapter, we will discuss the arithmetic and logic instructions.
- The execution of these instructions affects the status of the flags.
- The affected flags are carry flag (CF), sign flag (SF), zero flag (ZF), parity flag (PF) and overflow flag (OF).



5.1. Addition

- Addition instructions include: ADD, ADC, INC.
- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
ADD	Addition	ADD D,S	(S)+(D)→(D) carry → CF	OF,SF,ZF,AF,PF,CF
ADC	Add with carry	ADC D,S	(S)+(D)+(CF)→(D) carry → CF	OF,SF,ZF,AF,PF,CF
INC	Increment by 1	INC D	(D)+1→(D)	OF,SF,ZF,AF,PF,CF



5.1. Addition

- The allowed operands for ADD and ADC are:

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

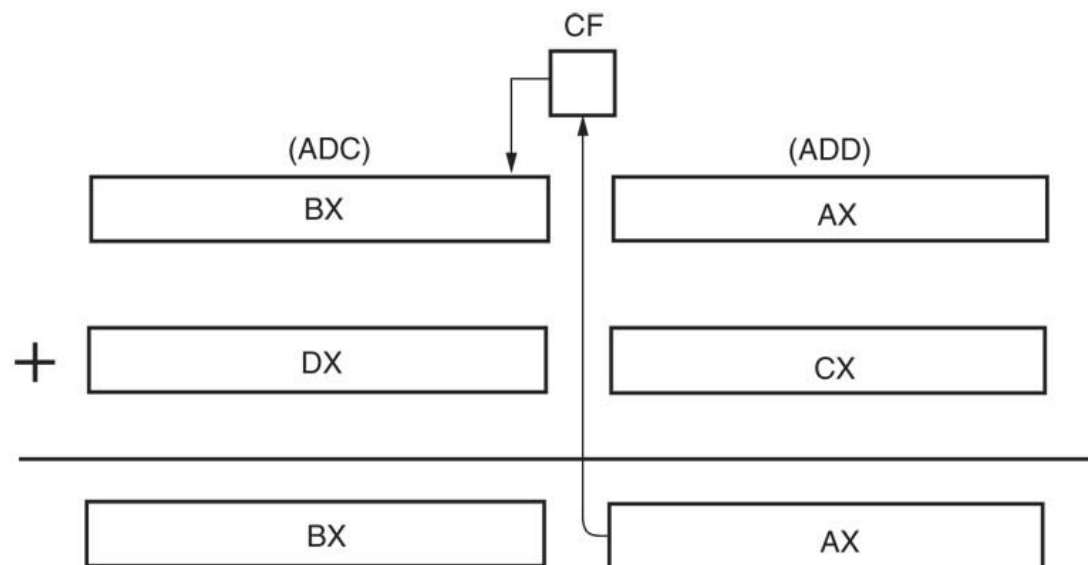
- The allowed operands for the INC

Destination
Register-16 bit
Register-8 bit
Memory

5.1. Addition

- Example: What are the result of executing the following program:

The program	Affected Register	Affected flags
MOV AL, EDH	AL = EDH	None
ADD AL, 21H	AL = ED+21 = 0EH	ZF=0, CF=1, AF=0, SF=0, PF=0, OF=0.
ADC AL,11H	AL =0E+11+1=20H	ZF=0, CF=0, AF=1, SF=0, PF=0, OF=0.
INC AL	AL = 20+1=21H	ZF=0, CF=0, AF=0, SF=0, PF=1, OF=0.





5.1.Subtraction

- Subtraction instructions include: SUB, SBB, DEC, NEG.
- The following table shows these instructions.

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D)-(S)→(D) Borrow → CF	OF,SF,ZF,AF, PF,CF
SBB	Subtract with borrow	SBB D,S	(D)-(S)-(CF)→(D) Borrow = CF	OF,SF,ZF,AF, PF,CF
DEC	Decrement by 1	DEC D	(D)-1→(D)	OF,SF,ZF,AF, PF,CF
NEG	Negate	NEG D	0-(D) →(D) 1 → SF	OF,SF,ZF,AF, PF,CF



5.2. Subtraction

- The allowed operands for SUB and SBB are:

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

- The allowed operands for DEC and NEG are

Destination
Register-16 bit
Register-8 bit
Memory



5.2. Subtraction

- Example: What are the result of executing the following program:

The program	Affected Register	Affected flags
MOV CX, 1527H	CX = 1527H	None
MOV BX, 1234H	BX = 1234H	None
SUB CX, 44H	CX = 1527 - 0044 = 14E3	ZF=0, CF=0, AF=0, SF=0, PF=0, OF=0.
SBB BX, CX	BX = 1234 - 14E3 - 0 = FD51	ZF=0, CF=1, AF=0, SF=1, PF=0, OF=0.
DEC BX	BX = FD50H	ZF=0, CF=0, AF=0, SF=1, PF=1, OF=0.
NEG BX	BX = 02B0H	ZF=0, CF=1, AF=0, SF=0, PF=1, OF=0.



5.3. Multiplication

- ✦ Both 8- and 16-bit multiplication on either signed or unsigned numbers can be performed.
- ✦ This instruction results in shorter programs and faster execution.
- ✦ Multiplication always results in a double-width product.
- ✦ For example, if two 8-bit numbers are multiplied, then the product is always 16 bits.
- ✦ Likewise, if two 16-bit numbers are multiplied, then the product is always 32 bits.



5.3. Multiplication

- In 8-bit multiplication,
 - The multiplicand is always in the AL register.
 - The programmer can choose the multiplier.
 - The 16-bit product is stored in AX.
- In 16-bit multiplication,
 - The multiplicand is always in the AX register.
 - The programmer can choose the multiplier.
 - The 32-bit product is stored in DX and AX. DX will contain the most significant 16 bits of the products and AX will contain the least significant 16 bits.
- In the signed multiplication, the product is in true form if positive and in two's complement form if negative.



5.3. Multiplication

✦ The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
MUL	Multiply (Unsigned)	MUL S	(AL)×(S8)→(AX) (AX)×(S16)→(DX),(AX)	OF,SF,ZF,AF,PF,CF
IMUL	Multiply (signed)	IMUL S	(AL)×(S8)→(AX) (AX)×(S16)→(DX),(AX)	OF,SF,ZF,AF,PF,CF

✦ The allowed operands are:

Source
Register-8 bit
Register-16 bit
Memory-8 bit
Memory-16 bit



5.3. Multiplication

■ Example 1: What are the results of this program:

MOV BL, 5 ;BL = 05H

MOV CL, 10 ;CL = 0AH

MOV AL, CL ;AL = 0A H

MUL BL ;AX = AL × BL = 0AH × 05H = 32H

■ Example 2: IF AL = -1 = FFH and CL = -2 = FEH, what is the result of executing:

a) MUL CL;

AX = AL × CL = 11111111 × 11111110 =
1111110100000010 = FD02H.

b) IMUL CL;

AX = -1 × -2 = 2 = 0002H.



5.4. Division

- Both 8- and 16-bit division on either signed (IDIV) or unsigned numbers (DIV) can be performed.
- Numbers are divided into its double-width dividend.
- For example, an 8-bit division always converts the 8-bit dividend into a 16-bit dividend.
- Likewise, in 16-bit division, the 16-bit dividend is always converted into 32-bit dividend.



5.4. Division

- In 8-bit division,
 - The dividend is located in AX register and the divisor is the operand selected for the instruction.
 - The results are two 8-bit numbers: the quotient (AL) and the remainder in (AH).
- In 16-bit division,
 - The dividend is located in DX and AX registers. DX will contain the most significant 16 bits of the dividend and AX will contain the least significant 16 bits.
 - The results are two 16-bit numbers: the quotient in (AX) and the remainder in (DX).
- The signs of the remainder and the quotient are the same.

5.4. Division

✦ The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
DIV	Division (Unsigned)	DIV S	1) $Q((AX)/(S8)) \rightarrow (AL)$ $R((AX)/(S8)) \rightarrow (AH)$ 2) $Q((DX,AX)/(S16)) \rightarrow (AX)$ $R((DX,AX)/(S16)) \rightarrow (DX)$	OF,SF,ZF,AF,PF,CF
IDIV	Division (signed)	IDIVS	1) $Q((AX)/(S8)) \rightarrow (AL)$ $R((AX)/(S8)) \rightarrow (AH)$ 2) $Q((DX,AX)/(S16)) \rightarrow (AX)$ $R((DX,AX)/(S16)) \rightarrow (DX)$	OF,SF,ZF,AF,PF,CF

✦ The allowed operands are:

Source
Register-8 bit
Register-16 bit
Memory-8 bit
Memory-16 bit



5.4. Division

- There are two instructions (CBW and CWD) used before the division instructions.
- In 8-bit division, CBW (convert byte to word) converts the signed number in AL to a 16-bit signed number in AX.
- In 16-bit division, CWD (convert word to double word) converts the signed 16-bit number in AX to a 32-bit signed number in both DX and AX.
- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
CBW	Convert B to W	CBW	(MSB of AL)→(All bits of AH)	None
CWD	Convert W to D	CWD	(MSB of AX)→(All bits of DX)	None



5.4. Division

- Example 1: What are the results of the following program:

```
MOV AL, A1H      ;AL = A1H.
```

```
CBW              ;AX = FFA1H
```

```
CWD              ; DX = FFFFH and AX = FFA1H.
```

- Example 2: IF $AX = 0012H$ and $CL = 03H$, what is the results of executing:

```
DIV CL
```

$0012H/03H = 06H$; The quotient is 6 and the remainder is 0

$AX = 0006H$



5.4. Division

- Example 3: What are the results of executing the following program:

MOV AX,-100 ;AX = -100 = -64H= FF9CH.

MOV CX,9 ;CX = 9 = 0009H.

CWD ; DX = FFFFH and AX = FF9CH

IDIV CX ; DX = -1 = FFFFH and

; AX = -11 = -000B H = FFF5H



5.5. BCD and ASCII Arithmetic

- The 8086/8088 allows arithmetic manipulation of both binary coded decimal (BCD) and American Standard Code for Information Interchange (ASCII).
- BCD operations are used in applications that require little arithmetic, such as point of sales terminal (POS).
- ASCII operations are used in systems that employ ASCII-coded data to store numbers.

5.5.1. BCD Arithmetic

- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
DAA	Decimal adjust for addition	DAA		OF,SF,ZF,AF,PF,CF
DAS	Decimal adjust for subtraction	DAS		OF,SF,ZF,AF,PF,CF
AAM	Adjust AL after multiplication	AAM	Q((AL)/10)→AH R((AL)/10)→AL	OF,SF,ZF,AF,PF,CF
AAD	Adjust AX before division	AAD	(AH)×10+AL→AL 00→AH	OF,SF,ZF,AF,PF,CF

- ✦ DAA, DAS and AAM are used after adding, subtracting or multiplying BCD numbers.
- ✦ The AAD instruction is used before a division to preadjust the numbers before using the DIV.



5.5.1. BCD Arithmetic

- Example: DAA (BX + DX) → CX

Trace the following program

```
MOV DX, 1234H           ;DX = 1234H
MOV BX, 3099H           ;BX = 3099.
MOV AL, BL
ADD AL, DL              ;AL=CDH
DAA                     ;AL = 33 and CF = 1
MOV CL, AL
MOV AL, BH
ADC AL, DH              ;AL = BH + DH + CF = 43H
DAA                     ; stays as is, it is a valid BCD
MOV CH, AL
```



5.5.1. BCD Arithmetic

- Example: DAS (BX - DX) → CX

Trace the following program

```
MOV DX, 1234H      ;DX = 1234H
```

```
MOV BX, 3099H     ;BX = 3099.
```

```
MOV AL, BL
```

```
SUB AL, DL
```

```
DAS
```

```
MOV CL, AL
```

```
MOV AL, BH
```

```
SBB AL, DH
```

```
DAS
```

```
MOV CH, AL
```

5.5.1. BCD Arithmetic

- Example: AAM (05 X 05)

Trace the following program

```
MOV AL, 5
```

```
;AL = 05H
```

```
MOV CL, 5
```

```
;CL = 05H
```

```
MUL CL
```

```
;AX now contains 0019H = 25
```

```
AAM
```

```
;AH = 02 and AL = 05
```

- Example : AAD (72/9)

```
MOV AX, 0702H
```

```
MOV BL, 9
```

```
AAD
```

```
;makes AX = 48H
```

```
DIV BL
```

```
;Q = 08H → AL R = 00H → AH
```



5.5.2. ASCII Arithmetic

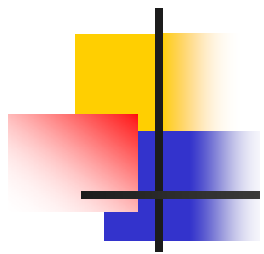
- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
AAA	ASCII adjust for addition	AAA		OF,SF,ZF,AF,PF,CF
AAS	ASCII adjust for subtraction	AAS		OF,SF,ZF,AF,PF,CF

- ✦ They are used with ASCII-coded numbers.
- ✦ These range from 30H through 39H and represent the number 0 through 9.
- ✦ They always use register AX as the source and destination.

TABLE 2.6 ASCII and EBCDIC Codes in Hex.

Character	ASCII	EBCDIC	Character	ASCII	EBCDIC	Character	ASCII	EBCDIC	Character	ASCII	EBCDIC
@	40		a	61	81	blank	20	40	NUL	00	
A	41	C1	b	62	82	!	21	5A	SOH	01	
B	42	C2	c	63	83	"	22	7F	STX	02	
C	43	C3	d	64	84	#	23	7B	ETX	03	
D	44	C4	e	65	85	\$	24	5B	EOT	04	37
E	45	C5	f	66	86	%	25	6C	ENQ	05	
F	46	C6	g	67	87	&	26	50	ACK	06	
G	47	C7	h	68	88	'	27	7D	BEL	07	
H	48	C8	i	69	89	(28	4D	BS	08	16
I	49	C9	j	6A	91)	29	5D	HT	09	05
J	4A	D1	k	6B	92	*	2A	5C	LF	0A	25
K	4B	D2	l	6C	93	+	2B	4E	VT	0B	
L	4C	D3	m	6D	94	,	2C	6B	FF	0C	
M	4D	D4	n	6E	95	-	2D	60	CR	0D	15
N	4E	D5	o	6F	96	.	2E	4B	SO	0E	
O	4F	D6	p	70	97	/	2F	61	SI	0F	
P	50	D7	q	71	98	0	30	F0	DLE	10	
Q	51	D8	r	72	99	1	31	F1	DC1	11	
R	52	D9	s	73	A2	2	32	F2	DC2	12	
S	53	E2	t	74	A3	3	33	F3	DC3	13	
T	54	E3	u	75	A4	4	34	F4	DC4	14	
U	55	E4	v	76	A5	5	35	F5	NAK	15	
V	56	E5	w	77	A6	6	36	F6	SYN	16	
W	57	E6	x	78	A7	7	37	F7	ETB	17	
X	58	E7	y	79	A8	8	38	F8	CAN	18	
Y	59	E8	z	7A	A9	9	39	F9	EM	19	
Z	5A	E9	{	7B		:	3A		SUB	1A	
[5B			7C	4F	;	3B	5E	ESC	1B	
\	5C		}	7D		<	3C	4C	FS	1C	
]	5D		~	7E		=	3D	7E	GS	1D	
^	5E		DEL	7F	07	>	3E	6E	RS	1E	
_	5F	6D				?	3F	6F	US	1F	





5.5.2. ASCII Arithmetic

- Example

MOV AX, 31H ; 31 is ASCII Code of 1

ADD AL, 39H ; 39 is ASCII code of 9

; results in 6AH

AAA ;clears AH if the sum is less than 10

; and adds 01H to AH if the sum is

; is greater than 10.

;AAA results in 01H in AH and 00H

; in AL...AX → 0100

ADD AX, 3030H ; results in 3130 which is ASCII 10



Lecture_7



5.6. Comparison

- The comparison instruction (CMP) is actually a subtraction that does not change anything except the flag bits.
- This instruction is almost always followed by a conditional jump instruction which tests the flag bits that are changed by CMP.



5.6. Comparison

✦ The following table shows this instruction

Mnemonic	Meaning	Format	Operation	Flags affected
CMP	Compare	CMP D,S	(D)-(S) is used to change the flag bits only.	OF,SF,ZF,AF,PF,CF

✦ The allowed operands are:

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate



5.6. Comparison



Example 1: What is the function of this program?

CMP AL,10H ;It performs $AL - 10H$ and the result
;is not stored but it affects the flag
;values (ZF, SF, etc).

JZ ONTEN ;If $ZF = 1$ then the program jumps to
;address ONTEN.
;If $ZF = 0$ then no jump occurs.



5.6. Comparison

- ✦ Example 2: Trace the following program and the values of the flag bits.

Instruction	Registers	ZF	SF	CF	AF	OF	PF
Initial states		0	0	0	0	0	0
MOV AX,1234H	AX = 1234H	0	0	0	0	0	0
MOV BX,ABCDH	BX = ABCDH	0	0	0	0	0	0
CMP AX,BX	AX-BX = 1234 - ABCD = 6667H	0	0	1	1	0	0



5.7. Basic Logic Operations

- ✦ The AND, OR, XOR and NOT are performed.
- ✦ They perform their respective logic operations bit-by-bit on the source and destination operands.
- ✦ The result is stored in the destination operand.
- ✦ The following table shows these instructions.

Mnemonic	Meaning	Format	Operation	Flags affected
AND	Logical AND	AND D,S	$(S) \bullet (D) \rightarrow (D)$	OF,SF,ZF,AF,PF,CF
OR	Logical OR	OR D,S	$(S) + (D) \rightarrow (D)$	OF,SF,ZF,AF,PF,CF
XOR	Logical XOR	XOR D,S	$(S) \oplus (D) \rightarrow (D)$	OF,SF,ZF,AF,PF,CF
NOT	Logical NOT	NOT D	$(\bar{D}) \rightarrow (D)$	OF,SF,ZF,AF,PF,CF



5.7. Basic Logic Operations

- The allowed operands for AND, OR and XOR are:

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

- The allowed operands for NOT are

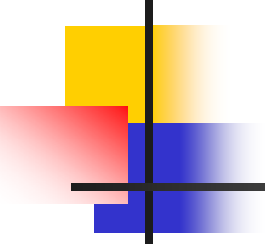
Destination
Register-16 bit
Register-8 bit
Memory



5.7.1. AND

- It is used to clear bits of a binary number selectively (often called masking).
- The main idea of the masking is based on:
If $X \text{ AND } 0 = 0$, this bit is cleared.
If $X \text{ AND } 1 = X$, this bit is passed without changes.
- Example 1: If $AX = 1234H$ and $BX = 000F$, the result of executing the instruction: `AND AX,BX`
 $AX = 1234 \text{ AND } 000F = 0004H$.
In this example, the leftmost 12 bits are cleared and the rightmost 4 bits are passed.

5.7.1. AND



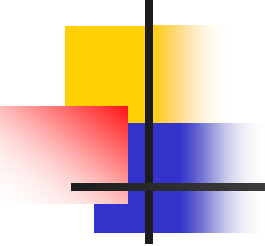
x x x x	x x x x	Unknown number
• 0 0 0 0	1 1 1 1	Mask
<hr/>		
0 0 0 0	x x x x	Result



5.7.2. OR

- It selectively sets bits of a binary number.
- The main idea of the OR operation is based on:
If $X \text{ OR } 1 = 1$, this bit is set.
If $X \text{ OR } 0 = X$, this bit is passed without any change.
- Example : If $AX = 1234H$ and $BX = FFF0$, the result of executing the instruction: `OR AX,BX`
 $AX = 1234 \text{ OR } FFF0 = FFF4H$.
In this example, the leftmost 123 bits are set and the rightmost 4 bits are passed.

5.7.2. OR



x x x x	x x x x	Unknown number
+ 0 0 0 0	1 1 1 1	Mask
<hr/>		
x x x x	1 1 1 1	Result



5.7.3. XOR

- It selectively inverts bits of a binary number.
- The main idea of the XOR operation is based on:
If $X \text{ XOR } 1 = \overline{X}$, this bit is inverted.
- If $X \text{ XOR } 0 = X$, this bit is passed without change.
- Example: If $AX = 1234H$ and $BX = 00FF$, the result of executing the instruction: `XOR AX,BX`
 $AX = 1234 \text{ XOR } 00FF = 12CBH$.

In this example, the leftmost 8 bits are passed and the rightmost 8 bits are inverted.



5.7.3. XOR

x x x x	x x x x	Unknown number
\oplus 0 0 0 0	1 1 1 1	Mask
<hr/>		
x x x x	$\bar{x} \bar{x} \bar{x} \bar{x}$	Result



5.7.4. NOT

- The NOT instruction or one's complement inverts each bit position of a number.
- Example: If $AX = 1234H$, the result of executing the instruction: NOT AX
 $AX = EDCBH$



5.7. Basic Logic Operations



Example: Trace the following program.

MOV AL, 55H	;AL = 55H.
AND AL, 1FH	;AL = 55 AND 1F = 15H
OR AL, C0H	;AL = 15 OR C0 = D5H
XOR AL, 0FH	;AL = D5 XOR 0F = DAH
NOT AL	;AL = NOT(DA) = 25H



5.7.5. TEST

- It performs the AND operation, but it affects only the flag register and not the operands of the instruction.
- TEST is used in the same manner as CMP, but to test a single bit rather than an entire number.
- Example : What is the result of executing:
`TEST AX,1.`

It tests the rightmost bit in AX for a 1 or 0.

If the ZF = 1, then the rightmost bit of AX is a 0.

If the ZF = 0, then the rightmost bit of AX is a 1.



5.8. Shift Instructions

The following table shows these instructions

Mnemonic	Meaning	Format	Operation	Flags affected
SHL	Shift logical left	SHL D,Count	Shift the D left by the number of bit positions equal to Count and fill the vacated bits positions on the right with zeros	OF, CF
SAL	Shift arithmetic left	SAL D,Count	Shift the D left by the number of bit positions equal to Count and fill the vacated bits positions on the right with zeros	OF, CF
SHR	Shift logical right	SHR D,Count	Shift the D right by the number of bit positions equal to Count and fill the vacated bits positions on the left with zeros	OF, CF
SAR	Shift arithmetic right	SAR D,Count	Shift the D right by the number of bit positions equal to Count and fill the vacated bits positions on the left with the original most significant bit	OF,SF,ZF, AF,PF,CF



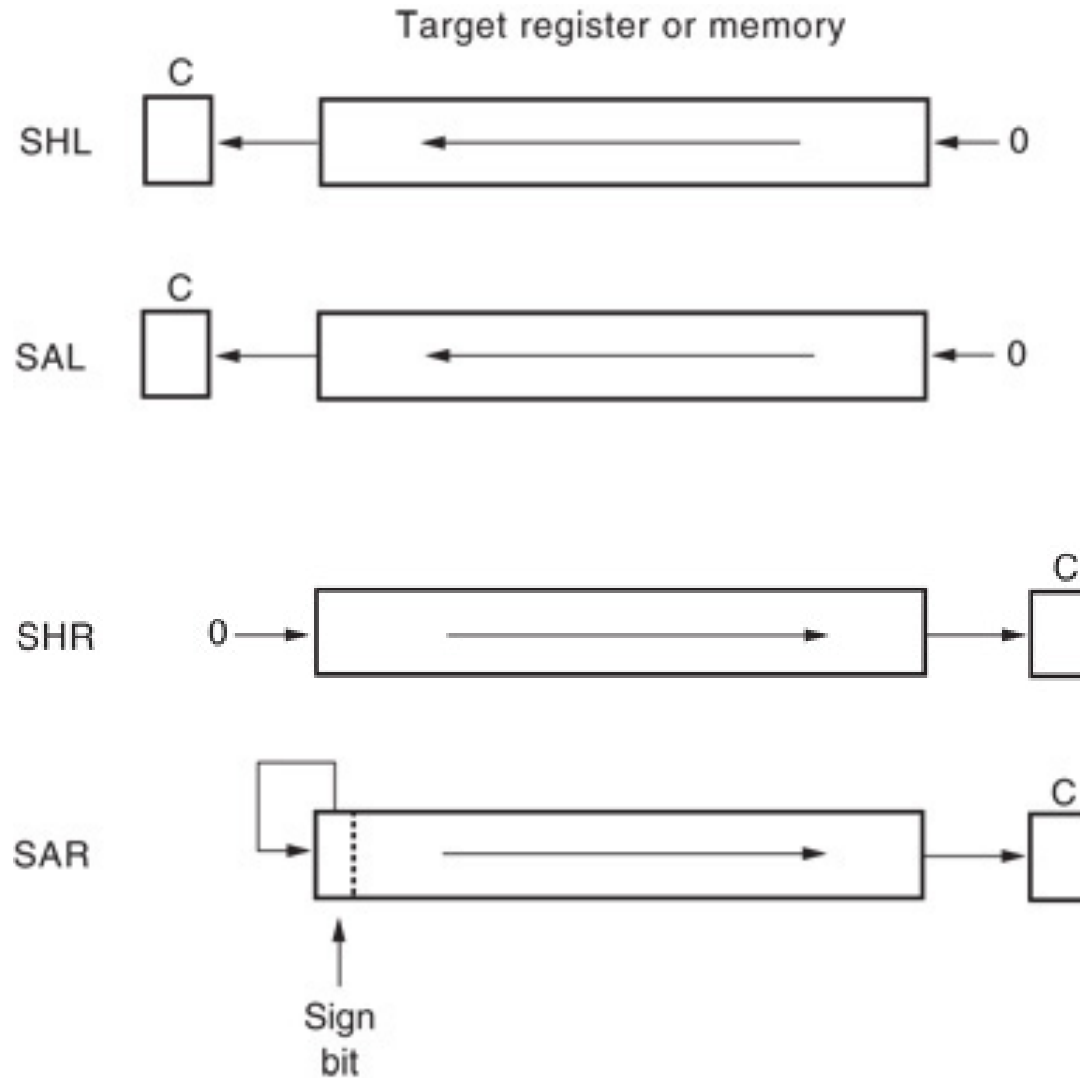
5.8. Shift Instructions

- The allowed operands for these instructions are shown in the following table:

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

- ✦ The arithmetic right shift will always divide a signed number by 2.
- ✦ A logical right shift will always divide an unsigned number by a 2.
- ✦ A left shift will always multiply a number by a 2.

5.8. Shift Instructions





5.8. Shift Instructions

- Example: IF $AX = 091AH = 0000\ 1001\ 0001\ 1010B$, What is the result of executing:

a) `SHL AX,1`

$AX = 0001\ 0010\ 0011\ 0100B = 1234H.$

b) `SAL AX,1`

$AX = 0001\ 0010\ 0011\ 0100B = 1234H.$



5.8. Shift Instructions

■ Example: IF $CL = 02H$ and $AX = 891AH = 1000\ 1001\ 0001\ 1010B$, What is the result of executing the following instructions:

a) SAR AX,CL

The first shift is $AX = 1100\ 0100\ 1000\ 1101B$, $CF = 0$

The second shift is $AX = 1110\ 0010\ 0100\ 0110B = E246H$, $CF = 1$.

b)SHR AX,CL

The first shift is $AX = 0100\ 0100\ 1000\ 1101B$, $CF = 0$

The second shift is $AX = 0010\ 0010\ 0100\ 0110B = 2246H$, $CF = 1$.



5.9. Rotate Instructions

The following table shows these instructions

Mnemonic	Meaning	Format	Operation	Flags affected
ROL	Rotate left	ROL D,Count	Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the leftmost bit goes back into the rightmost bit position.	OF, CF
ROR	Rotate right	ROR D,Count	Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes back into the leftmost bit position.	OF, CF
RCL	Rotate left through carry	RCL D,Count	Same as ROL except carry is attached to (D) for rotation.	OF, CF
RCR	Rotate right through carry	RCR D,Count	Same as ROR except carry is attached to (D) for rotation.	OF, CF



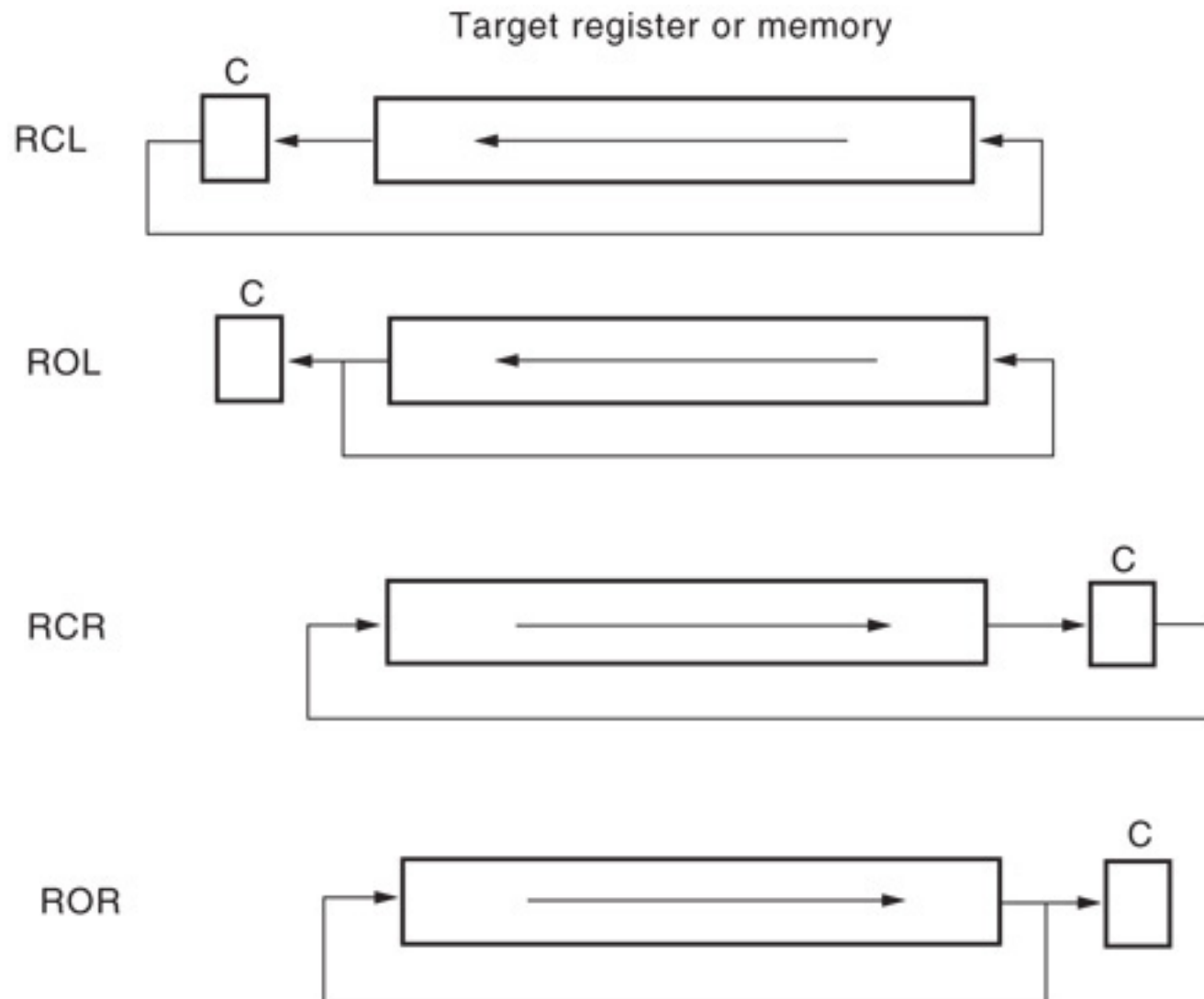
5.9. Rotate Instructions

- The allowed operands for these instructions are shown in the following table:

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

- ✦ The difference between the rotate and shift instructions is that bits moved out are not lost but they are reloaded at the other end.

5.9. Rotate Instructions





5.8. Rotate Instructions

Example: If CL=4H, AX=1234H =0001 0010 0011 0100,
What is the result of executing the following instructions:

- a) ROL AX,1 ;AX = 0010 0100 0110 1000B, CF = 0
- b) RCL AX,CL ;AX = 0100 1000 1101 0000, CF = 0
 ;AX = 1001 0001 1010 0000, CF = 0
 ;AX = 0010 0011 0100 0000, CF = 1
 ;AX = 0100 0110 1000 0001, CF = 0
- c) RCR AX,CL ;AX = 0010 0011 0100 0000, CF = 1
 ;AX = 1001 0001 1010 0000, CF = 0
 ;AX = 0100 1000 1101 0000, CF = 0
 ;AX = 0010 0100 0110 1000, CF = 0
- d) ROR AX,1 ;AX = 0001 0010 0011 0100, CF = 0



5.9. String Comparisons

- String comparison operations allow a section of memory to be compared for a particular value or two sections of memory to be compared for a match or no-match condition.
- The following table shows these instructions:

Mnemonic	Meaning	Format	Operation	Flags affected
SCASB	Scan	SCASB	(AL)-(DI) changes the flags.	OF,SF,ZF,AF,PF,CF
SCASW	Scan	SCASW	(AX)-(DI) changes the flags.	OF,SF,ZF,AF,PF,CF
CMPSB	Compare	CMPSB	(SI)-(DI) changes the flags.	OF,SF,ZF,AF,PF,CF
CMPSW	Compare	CMPSW	(SI)-(DI) changes the flags.	OF,SF,ZF,AF,PF,CF



5.9. String Comparisons

- In most applications, the string operations must be repeated in order to process arrays of data.
- This is done by inserting a repeat prefix before the instruction that is to be repeated.
- The repeat prefix is shown:

Prefix	Used With	Meaning
REPE/REPZ	CMPS, SCAS	Repeat while not end of string and strings are equal. CX \neq 0 or ZF = 1
REPNE/REPNZ	CMPS, SCAS	Repeat while not end of string and strings are not equal. CX \neq 0 or ZF = 0



5.9. String Comparisons



Example: Trace the following program.

MOV DI, BLOCK ;Load DI by the starting address in
the block.

CLD ;Select auto-increment by clear DF.

MOV CX, 100H ;Load CX by the block length 256.

MOV AL, 0 ;Load AL by 0.

REPNE SCASB ;Repeat the scan instruction until
finding 0 or $CX = 0$.



5.9. String Comparisons



Example: Trace the following program.

MOV SI, LINE	;Load SI by the starting address in the source block.
MOV DI, TABLE	;Load DI by the starting address in the destination block
MOV CX,10	;Load CX by the block length 10.
CLD	;Select auto-increment by clear DF.
REPE CMPSB	;Repeat the compare instruction if they are equal or until CX = 0.



Lecture_8

Chapter 3.

Machine Language Coding

- In this chapter, we will discuss the conversion from assembly language to machine Language.
- Each assembly instruction must be converted to its equivalent machine code instruction.



3.1. Introduction

- The machine code should specify:
 - What operation is to be performed.
 - Whether the operation is performed on byte or word data.
 - What operand or operands are to be used.
 - Whether the operands are located in registers or a register and memory location.
 - If the operand is stored in memory, how its address is to be generated.



3.1. Introduction

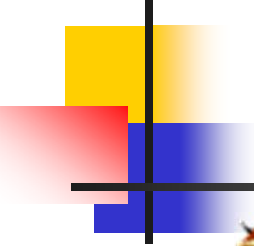
- The machine code instruction can be encoded in up to 6 bytes.
- Single-byte instructions generally specify a simpler operation with a register or a flag bit.
- Example on single-byte instruction, Complement Carry (CMC) is equivalent to F5H.
- Single Operand : INC DL

3.2. General Instruction Format

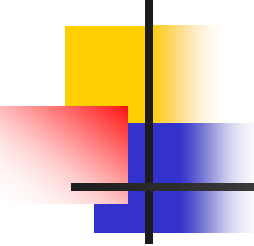
- Most multibyte instructions use the general instruction format as follows:

Byte 1		Byte 2				Byte 3	Byte 4	Byte 5	Byte 6
						Low	High	Low	High
Opcode	D	W	MOD	REG	R/M	Disp/Data	Disp/Data	Data	Data

3.2. General Instruction Format

- 
- ✦ Byte 1 contains three kinds of information:
 - ✦ Opcode field (6 bits) specifies the operation.
 - ✦ Register direction bit (D bit) specifies whether the register operand specified in byte 2 is the source or destination operand.
 - $D = 1$ if the register is a destination operand.
 - $D = 0$ if the register is a source operand.
 - Data size bit (W) specifies whether the operation will be performed on byte or word.
 - $W = 0$ for byte.
 - $W = 1$ for word.

3.2. General Instruction Format



Byte 1					Byte 2					Byte 3	Byte 4	Byte 5	Byte 6
										Low	High	Low	High
Opcode		D	W	MOD	REG	R/M		Disp/Data	Disp/Data	Data	Data		

- Byte 2 contains three fields.
 - The register is used for an operand
 - Where the other operand is stored. It can be in either a register or a memory location.



3.2. General Instruction Format

- The 3-bit register field (REG) is used to identify the register for an operand (first or second depending on D).
- The register is defined as source or destination by the D bit in byte 1.
- The following table shows the value for the register (REG) field:

REG	W=0 (Byte)	W=1 (Word)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

3.2. General Instruction Format

- The 2-bit mode field (MOD) indicates whether the other operand is in a register or memory.
- The following table shows the mode (MOD) field encoding:

Code	Explanation
00	Memory mode, no displacement
01	Memory mode, 8-bit displacement follows.
10	Memory mode, 16-bit displacement follows.
11	Register mode (no displacement).

3.2. General Instruction Format

- The 3-bit register/memory (R/M) field specifies the other operand as shown in the following table.

MOD = 11			Effective Address Calculation			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	CL	CX	001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	DL	DX	010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	BL	BX	011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	AH	SP	100	(SI)	(SI)+D8	(SI)+D16
101	CH	BP	101	(DI)	(DI)+D8	(DI)+D16
110	DH	SI	110	Direct Address	(BP)+D8	(BP)+D16
111	BH	DI	111	(BX)	(BX)+D8	(BX)+D16

3.2. General Instruction Format



- Byte 3 and byte 4 are used to store the displacement or the data.

The low byte is stored first then the high byte is stored.

- Byte 5 and byte 6 are used to store the data if byte 3 and 4 contain the displacement.

3.2. General Instruction Format

- Example 1:

Find the machine code of: MOV BL, AL.

If the MOV opcode is 100010.

Byte 1		Byte 2				Byte 3	Byte 4	Byte 5	Byte 6
						Low	High	Low	High
Opcode	D	W	MOD	REG	R/M	Disp/Data	Disp/Data	Data	Data

The first byte has opcode = 100010, D = 1, W = 0.

The second byte has: REG = 011, MOD = 11, R/M = 000

MOV BL, AL = 100010 1 0 11 011 000 = 8AD8H

Byte 1					Byte 2				
100010					1	0	11	011	000

3.2. General Instruction Format

- Example 2:
Find the machine code of: ADD AX, [SI].
If the ADD opcode is 000000.

Byte 1		Byte 2				Byte 3	Byte 4	Byte 5	Byte 6
						Low	High	Low	High
Opcode	D	W	MOD	REG	R/M	Disp/Data	Disp/Data	Data	Data

The first byte has opcode = 000000, D = 1, W = 1.

The second byte has: REG = 000, MOD = 00, R/M = 100

ADD AX, [SI] = 000000 1 1 00 000 100 = 03 04H

3.2. General Instruction Format

- Example 3:
Find the machine code of: XOR CL, [1234].
If the XOR opcode is 001100.

Byte 1		Byte 2				Byte 3	Byte 4	Byte 5	Byte 6
						Low	High	Low	High
Opcode	D	W	MOD	REG	R/M	Disp/Data	Disp/Data	Data	Data

The first byte has opcode = 001100, D = 1, W = 0.

The second byte has: REG = 001, MOD = 00, R/M = 110

XOR CL, [1234] = 00110010 00001110 00110100 00010010
= 32 0E 34 12 H

3.2. General Instruction Format

- Example 4:
Find the machine code of: ADD [BX][DI]+1234H, AX.
If the ADD opcode is 000000.

Byte 1		Byte 2				Byte 3	Byte 4	Byte 5	Byte 6
						Low	High	Low	High
Opcode	D	W	MOD	REG	R/M	Disp/Data	Disp/Data	Data	Data

The first byte has opcode = 000000, D = 0, W = 1.

The second byte has: REG = 000, MOD = 10, R/M = 001.

$$\begin{aligned}
 &00000001 \quad 10000001 \quad 00110100 \quad 00010010 \\
 &= 01 \ 81 \ 34 \ 12 \ H
 \end{aligned}$$

3.2. General Instruction Format

- The drawbacks of this format:
 1. The application of sign extension.
 2. Not used for the segment register.
- It cannot be used to encode all the instructions.
- Minor modifications must be made to encode few instructions.
- There is a table that can be used to encode all the instruction set.

3.3. Instruction Set Table

- It shows all the instructions.
- Example: MOV: Move

Register/memory to/from register	100010dw	Mod Reg R/M	Disp-lo	Disp-hi		
Immediate to register/memory	1100011w	Mod 000 R/M	Disp-lo	Disp-hi	Data	Data if w=1
Immediate to register	1011 w Reg	Data	Data if w = 1			
Memory to accumulator	1010000w	Addr-lo	Addr-hi			
Accumulator to memory	1010001w	Addr-lo	Addr-hi			
Register/memory to segment register	10001110	Mod 0 SR R/M	Disp-lo	Disp-hi		
Segment register to register/memory	10001100	Mod 0 SR R/M	Disp-lo	Disp-hi		



3.3. Instruction Set Table

- Example:

Find the machine code of:

MOV [BP][DI]+1234h,ABCDH

✦ From the table:

Immediate register/memory	to	1100011w	Mod 000 R/M	Disp-lo	Disp-hi	Data	Data if w=1
------------------------------	----	----------	-------------	---------	---------	------	----------------

$W = 1, MOD = 10, R/M = 011$

The machine code =

11000111 10000011 00110100 00010010 11001101 10101011
= C7 83 34 12 CD AB H.

3.3. Instruction Set Table

■ Examples:

Add=Addition

Register to either Reg/memory	000000 d w	Mod R/M	Reg	Disp-lo	Disp-hi		
Immediate to register/memory	100000 s w	Mod R/M	000	Disp-lo	Disp-hi	Data	Data if s,w=01
Immediate to accumulator	0000010 w	Data		Data if w = 1			

DEC: Decrement

Register/memory	1111111 w	Mod 001 R/M	Disp-lo	Disp-hi
Register	01001 Reg			

INC: Increment

Register/memory	1111111 w	Mod 000 R/M	Disp-lo	Disp-hi
Register	01000 Reg			



3.3. Instruction Set Table

- The following table shows 1-bit field and their functions

Field	Value	Function
S	0	No Sign Extension
S	1	Sign extend 8-bit immediate data to 16 bits if $W = 1$
V	0	Shift/Rotate count is one.
V	1	Shift/Rotate count is specified in CL register.
Z	0	Repeat/Loop while zero flag is clear.
Z	1	Repeat/Loop while zero flag is set.



3.3. Instruction Set Table

- The instructions that involve segment register need a 2-bit field to encode these registers.
- This field is called the SR field.
- They are defined as shown in the following table.

Register	SR
ES	00
CS	01
SS	10
DS	11



3.3. Instruction Set Table

- Example:

Find the machine code of:

MOV [BP][DI]+1234h,DS

✦ From the table:

Segment register to register/memory	10001100	Mod 0 SR R/M	Disp-lo	Disp-hi		
-------------------------------------	----------	--------------	---------	---------	--	--

Mod = 10, SR = 11, R/M = 011

The machine code = 10001100 10011011 00110100 00010010
= 8C 9B 34 12 H.



3.3. Instruction Set Table

- Example: Encode the following Program

```
MOV AX,1020H
```

```
MOV DS,AX
```

```
MOV SI,100H
```

```
MOV DI,120H
```

```
MOV CX,10H
```

```
AGAIN: MOV AH,[SI]
```

```
MOV [DI],AH
```

```
INC SI
```

```
INC DI
```

```
DEC CX
```

```
JNZ AGAIN
```



3.3. Instruction Set Table

Label	Memory location	Assembly Instruction	Machine Code
AGAIN	200,201,202	MOV AX,1020H	B82010
	203,204	MOV DS,AX	8ED8
	205,206,207	MOV SI,100H	BE0001
	208,209,20A	MOV DI,120H	BF2001
	20B,20C,20D	MOV CX,10H	B91000
	20E,20F	MOV AH,[SI]	8A24
	210,211	MOV [DI],AH	8825
	212	INC SI	46
	213	INC DI	47
	214	DEC CX	49
215,216	JNZ AGAIN	75F7	



Lecture_9



2.10. Port Addressing

- It is used with the IN and OUT instructions to access input and output ports.
- There are two types:
 - Direct addressing
 - Indirect addressing



2.10. Port Addressing

- Direct Addressing:
 - The port number is given directly.
 - Example: `IN AL , 15H`
 - This stands for “input the data from the byte-wide input port at address 15H of the I/O address space to register AL”



2.10. Port Addressing

- Indirect Addressing:
 - The port number is stored in register DX
 - Example: `IN AL , DX`
 - This stands for “input the data from byte-wide input port whose address is specified by the contents of register DX to register AL”.

2.11. Program Memory Addressing Modes



- These modes are used with the JMP and CALL instructions.
- They can be divided into:
 - Direct
 - Relative
 - Indirect



2.12. Stack Memory

- It is used to:
 - Hold data temporarily.
 - Stores return addresses from subroutines.
- It is a LIFO (last-in, first-out) memory.
- Data are stored using PUSH or CALL.
- Data are removed using POP or RET.
- Two important registers are used in the stack:
 - SS (stack segment) points to the beginning of the stack.
 - SP (Stack Pointer) points to the top of the stack.



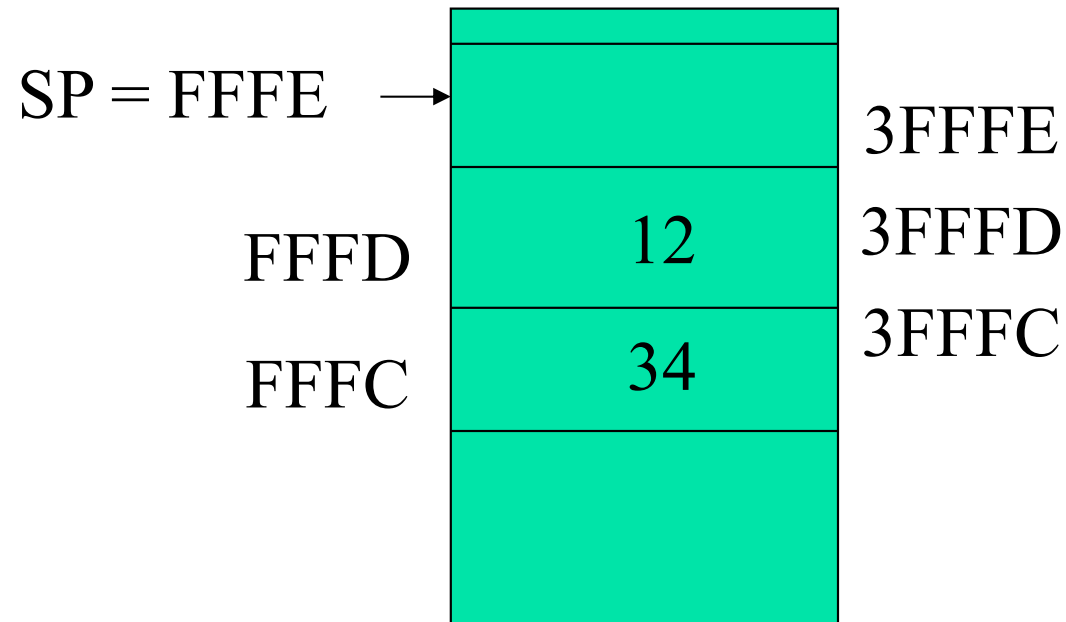
2.12. Stack Memory

- How to Store a word to the stack:
 - The high-order byte is placed in the location addressed by $SP - 1$.
 - The low-order byte is placed in the location addressed by $SP - 2$.
 - Decrement SP by 2.

2.12. Stack Memory

- Example: PUSH AX

where AX = 1234H, SS = 3000H, SP = FFFE_H



After Execution: SP = FFDC

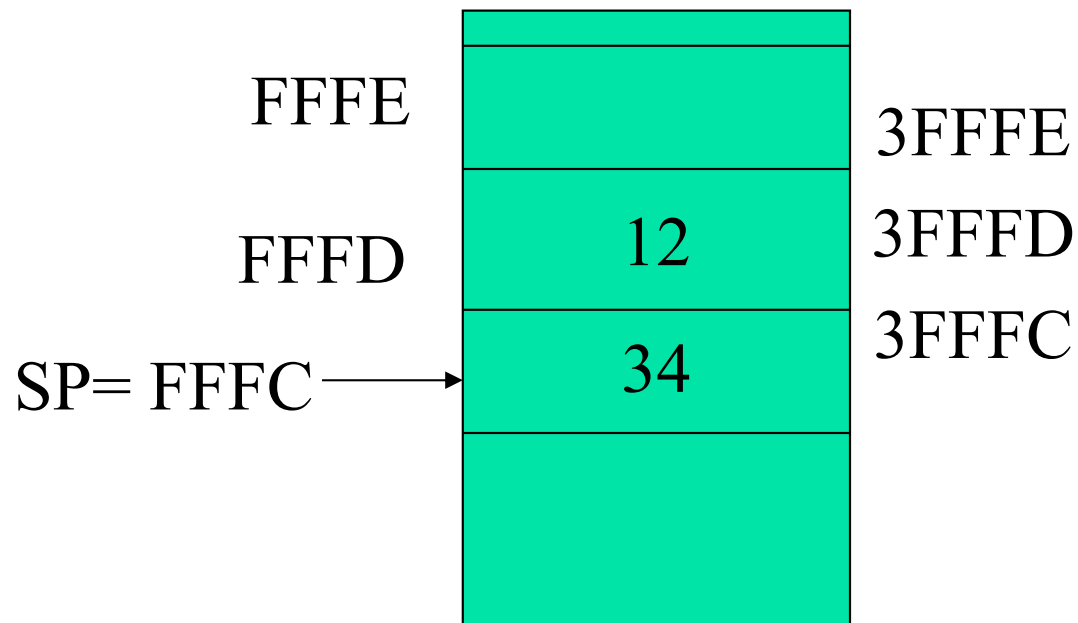


2.12. Stack Memory

- How to remove a word from the stack:
 - The low-order byte is removed from the location addressed by SP.
 - The high-order byte is removed from the location addressed by SP + 1.
 - Increment SP by 2.

2.12. Stack Memory

- Example: POP AX
where SS = 3000H, SP = FFFCH



After Execution: AX = 1234 and SP = FFFE

Chapter 6.

Program Control Instructions

- In this chapter, we will discuss the program control instructions.
- They include: JMP and subroutines.



6.1. Flag Control Instructions

- These instructions affect the state of the flags.
- The following table shows these instructions.

Mnemonic	Meaning	Format	Operation	Flags affected
CLC	Clear CF	CLC	$(CF) \leftarrow 0$	CF
STC	Set CF	STC	$(CF) \leftarrow 1$	CF
CMC	Complement CF	CMC	$(CF) \leftarrow \overline{(CF)}$	CF
CLI	Clear IF	CLI	$(IF) \leftarrow 0$	IF
STI	Set IF	STI	$(IF) \leftarrow 1$	IF
CLD	Clear DF	CLD	$(DF) \leftarrow 0$	DF
STD	Set DF	STD	$(DF) \leftarrow 1$	DF



6.2. Jump Instructions

- It is used to skip over sections of a program to any part of the program.
- The jump alters the execution path of instructions in the program.
- Program execution is not intended to return to the next sequential instruction after the jump instruction.



6.2. Jump Instructions

- CS and IP keep track of the next instruction to be executed.
- The jump instruction involves altering the contents of these registers (CS and IP).
- Thus, execution continues at an address other than that of the next sequential instruction.
- There are two different types of jump instructions:
 - Unconditional
 - Conditional



6.2. Jump Instructions

- In an unconditional jump,
 - No status requirements or conditions are imposed for the jump to occur.
 - As the instruction is executed, the jump takes place to change the execution sequence.
- In a conditional jump,
 - If the status conditions exist, the jump instruction is executed.
 - Otherwise, execution continues with the next sequential instruction of the program.
- The condition depends on the status flags such as CF, PF, OF, SF, ZF.



6.2.1. Unconditional Jump

- The unconditional jump instruction is given as:

Mnemonic	Meaning	Format	Operation	Flags affected
JMP	Unconditional jump	JMP Operand	Jump is initiated to the address specified by the operand	None

- The allowed operands are:

Operands
Short-Label
Near-Label
Far-Label
Memory Pointer 16
Register Pointer 16
Memory Pointer 32



Lecture_10



6.2.1. Unconditional Jump

- ✦ There are two basic kinds of unconditional jumps:
 - Intra-segment (Short and Near jump)
 - Inter-segment (Far jump)



6.2.1. Unconditional Jump

- ✦ Intra-segment jump is limited to addresses within the current code segment.
- ✦ It is achieved by modifying the value of IP only.
- ✦ It is divided into short and near.

- ✦ Inter-segment jump is used to addresses outside the current code segment.
- ✦ This type is achieved by modifying the value in CS and IP registers.



6.2.1. Unconditional Jump

- ✦ The machine codes of the unconditional jump:

Short JMP	OPCODE (EBH)	DISP			
Near JMP	OPCODE (E9H)	IP Low	IP High		
Intersegment JMP	OPCODE (EAH)	IP Low	IP High	CS Low	CS High

- ✦ Short jump allows jumps or branches to memory locations within +127 and -128 bytes from memory location following the jump.
- ✦ Near jump allows jumps or branches to memory locations within +32K and -32K bytes from memory location following the jump.



6.2.1. Unconditional Jump

✦ Example: What is the content of the IP register after the JMP instruction?

a) JMP 04

Before Execution: CS = 1000H, IP = 0002H

After the execution of JMP:

New IP = IP + 04H = 0006H

b) JMP 200DH

Before Execution: CS = 1000H, IP = 200BH

After the execution of JMP: New IP = 200DH



6.2.1. Unconditional Jump

✦ Example: What is the content of the IP register after the JMP instruction?

c) JMP A300:0127

Before Execution: CS = 1000H, IP = 0002H

After the execution of JMP:

New CS = A300H, New IP = 0127H



6.2.1. Unconditional Jump

✦ Example: What is the content of the IP register after the JMP instruction?

d) JMP [BX]

CS = 0CDEH, BX = 1000H, DS = 1000H

IP = 0102H

Address	Contents
11002	F2
11001	02
11000	00

After the execution of JMP:

New IP = 0200H

PA = CS×10+IP=0CDE0+0200=0CFE0H



6.2.1. Unconditional Jump

✦ Example: What is the content of the IP register after the JMP instruction?

d) `JMP DWORD PTR[BX]`

`CS = 0CDEH, BX = 1000H, DS = 1000H`

`IP = 0102H`

Address	Contents
11000	00
11001	02
11002	10
11003	0A

After the execution of JMP:

New IP = 0200H, New CS = 0A10H



6.2.2. Conditional Jump

- The conditional jump instruction is given as:

Mnemonic	Meaning	Format	Operation	Flags affected
Jcc	conditional jump	Jcc Operand	If the specified condition cc is true the jump to the address specified by the operand is initiated; Otherwise the next instruction is executed.	None

- ✦ Conditional jumps are all short jumps.
- ✦ The range of the jump is always within +127 to –128 bytes from the address of the next instruction.



6.2.2. Conditional Jump

✦ A list of conditional jump is given as:

Opcode	Condition Tested	Function
JA/JNBE	CF = 0 and ZF = 0	Jumps above/jumps not below or equal to
JAE/JNB	CF = 0	Jumps above or equal to/jumps not below
JB/JNAE	CF = 1	Jumps below/jumps not above or equal to
JBE/JNA	CF = 1 or ZF = 1	Jumps below or equal to/jumps not above
JC	CF = 1	Jumps carry set
JE/JZ	ZF = 1	Jumps equal/ jumps 0
JG/JNLE	OF = ZF and SF	Jumps greater/jumps not less than or equal to
JGE/JNL	SF = OF	Jumps greater than or equal to/jumps not less than
JL/JNGE	SF = OF	Jumps less than/jumps not greater than or equal to
JLE/JNG	Z = 1 or S = 0	Jumps less than or equal to/jumps not greater than
JNC	CF = 0	Jumps no carry
JNE/JNZ	ZF = 0	Jumps not equal to/jumps not 0
JNO	OF = 0	Jumps no overflow
JNP/JPO	SF = 0	Jumps no parity/jumps parity odd
JNS	PF = 0	Jumps no sign (positive)
JO	OF = 1	Jumps on overflow
JP/JPE	PF = 1	Jump parity/jumps parity even
JS	SF = 1	Jumps sign (negative)
JCXZ	CX = 0	Jumps if CX = 0



6.2.2. Conditional Jump

- Example 1: Trace the following program

```
CMP AX,BX      ;The result of AX-BX affects the flags
JB DIFF2       ;Jump if Below (CF = 1)
DIFF1 MOV DX,AX ;DX = AX
SUB DX,BX      ;DX = AX-BX affects the CF
JMP DONE
DIFF2 MOV DX,BX ;DX = BX
SUB DX,AX      ;DX = BX - AX affects the CF.
DONE NOP
```

This program tests whether $AX < BX$ or not.

If $AX > BX$, $DX = AX - BX$

If $AX < BX$, $DX = BX - AX$



6.2.2. Conditional Jump

- Example 1: Trace the following program

```
SCAN: MOV DI,OFFSET TABLE    ;Load DI by the address of
                                the first byte in array TABLE
                                MOV CX,100                ;Load counter CX = 100
                                MOV AL, 0AH              ;Load AL by 0AH
                                CLD                       ;Select auto increment.
                                REPNE SCASB              ;Search for the byte 0A
                                JCXZ NOT_FOUND
NOT_FOUND -----
```

This program scans a table of 100 bytes for a 0AH.



6.3. Loop Instructions

- There are two different types of loop instructions:
 - Unconditional
 - Conditional loop.
- The unconditional loop (LOOP) instruction is a combination of the conditional jump and the decrement of CX instructions.
 - It will decrement the contents of register CX.
 - If CX is not 0, jump to the label associated with loop.
 - If CX becomes a 0, the next sequential instruction in the program is executed.



6.3. Loop Instructions

- The conditional loop instructions include LOOPE/LOOPZ, LOOPNE/LOOPNZ.
- They work as LOOP instruction except that they check for two conditions: the contents of both CX and ZF flag.
 - If CX is not 0 and the ZF condition is satisfied, jump to the label associated with loop.
 - If CX is 0 or the ZF condition is not satisfied, the next sequential instruction in the program is executed.



6.3. Loop Instructions

- The following table shows the loop instructions

Mnemonic	Meaning	Format	Operation	Flags affected
LOOP	Loop	LOOP Short-label	$(CX) \leftarrow (CX) - 1$ Jump is initiated to location defined by short label if $(CX) \neq 0$; Otherwise, execute next sequential instruction	None
LOOPE/ LOOPZ	Loop while equal/Loop while zero	LOOPE/LOOPZ Short-label	$(CX) \leftarrow (CX) - 1$ Jump to location defined by short label if $(CX) \neq 0$ and $(ZF) = 1$; Otherwise, execute next sequential instruction	None
LOOPNE/ LOOPNZ	Loop while not equal/ Loop while not zero	LOOPNE/LOOPNZ Short-label	$(CX) \leftarrow (CX) - 1$ Jump to location defined by short label if $(CX) \neq 0$ and $(ZF) = 0$; Otherwise, execute next sequential instruction	None



6.3. Loop Instructions

- Example 1: Trace the following program:

```
MOV CX, 05    ;Load the counter CX = 5
MOV DX, 00    ;Load DX by zero
AGAIN:      NOP        ;No operation (Just Waiting)
            INC DX     ;Increment DX by 1
            LOOP AGAIN ;Go to AGAIN until CX = 0.
```

- ✦ This program increment the value of DX five times
DX = 5 and CX = 0.



6.3. Loop Instructions

- Example 2: Trace the following program:

```
MOV DL,05           ;Load DL by 05H
MOV AX,0A00H        ;Load AX by 0A00H
MOV DS,AX           ;Initialize the DS =AX
MOV SI,0200H        ;Initialize SI by 0200H
MOV CX, 0FH         ;Load CX = 15
AGAIN: INC SI        ;Increment SI by 1
      CMP [SI],DL    ;[SI]-DL affect the ZF
      LOOPNE AGAIN   ;Repeat this
```

comparison loop until CX=0 or ZF=1.



6.3. Loop Instructions

- It compares a block of data pointed to by SI to DL.
- The data stored at the memory location starting at address DS:0200 are: 4, 6, 3, 9, 5, 6, D, F, 9, BA, AB, 32, E9, 4C, F7, 80,..
- The length of the block is 15 numbers.
 - 1st loop: compare 4 and 5, ZF = 0, SI = 0200, CX = 0E
 - 2nd loop: compare 6 and 5, ZF = 0, SI = 0201, CX = 0D
 - 3rd loop: compare 3 and 5, ZF = 0, SI = 0202, CX = 0C
 - 4th loop: compare 9 and 5, ZF = 0, SI = 0203, CX = 0B
 - 5th loop: compare 5 and 5, ZF = 1, SI = 0204, CX = 0AThe loop is terminated.



6.4. Subroutines

- The subroutine is a very important part of any computer's software architecture.
- It is a group of instructions that performs a given task.
- It is used many times by the program but need to be stored once in the memory.
- This saves memory space and makes the task of programming much simpler because it takes less time to code a program.
- The disadvantage of a subroutine is that the computer takes a small additional time to link to the subroutine (CALL) and return from it (RET).



Lecture_11



6.4. Subroutines

- The subroutine is a very important part of any computer's software architecture.
- It is a group of instructions that performs a given task.
- It is used many times by the program but need to be stored once in the memory.
- This saves memory space and makes the task of programming much simpler because it takes less time to code a program.
- The disadvantage of a subroutine is that the computer takes a small additional time to link to the subroutine (CALL) and return from it (RET).



6.4. Subroutines

- The stack is used to store the return address so that the subroutine may return to the program at the point after the CALL instruction in the program.
- The subroutine is called a procedure as it would be in a higher-level language.
- There are two basic instructions: CALL and RET.



6.4. Subroutines

- These instructions are as shown:

Mnemonic	Meaning	Format	Operation	Flags affected
CALL	Call subroutine	CALL Operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS is saved on the stack.	None
RET	Return	RET or RET operand	Return to the main program by restoring IP (and CS for far-procedure). If operand is present, it is added to the content of SP.	None



6.4. Subroutines

- The operands of the CALL are as shown:

Operands
Near-procedure
Far-procedure
Memory pointer (16 bits)
Register pointer (16 bits)
Memory pointer (32 bits)

- The operands of the RET are as shown:

Operands
None
Displacement (16 bits)



6.4. Subroutines

- ✦ CALL instruction transfers the flow of a program to a procedure.
- ✦ CALLs differ from JUMPs because they save the contents of IP on the stack if the CALL is near or IP and CS on the stack if it is far.
- ✦ They are divided into:
 - ✦ Near CALL
 - ✦ Far CALL



6.4. Subroutines

- Near CALL instruction is 3 bytes long and its second and third bytes contain the offset location of the near procedure.
- It causes the jump to the subroutine.
- It also pushes the IP register onto the stack.
- Because the IP register contains the address of the next instruction to be executed.



6.4. Subroutines

- Far CALL instruction is 5 bytes long.
- Bytes 2 and 3 contain the IP of the subroutine and bytes 4 and 5 contain the new code segment (CS) value for the subroutine.
- It causes the jump to the subroutine.
- It also pushes both the IP and the CS registers onto the stack.
- Because the IP and CS registers contain the address of the next instruction to be executed, the return address is pushed onto the stack.



6.4. Subroutines

- RET instruction removes either a 16-bit number (near return) from the stack and places it in the IP or a 32-bit number (far return) and places it in IP and CS.
- When IP or IP and CS are changed, the location is changed to the address of the instruction that immediately follows the most recent CALL to a procedure.
- There is another form of the RET instruction. This form allows a number added to the contents of the stack pointer (SP) before the return.



6.4. Subroutines

- PROC is used to indicate the start of a procedure (subroutine), the name of the procedure and the type of CALL and RET instructions used by the assembler.
- The name of the subroutine can be any valid assembly language name.
- The type can be near (intrasegment) or far (intersegment) depending on whether the procedure is located within the code segment or some distance from it.
- ENDP is used to indicate the end of the procedure and the name of that procedure.



6.4. Subroutines

- Example 1: What is the result of executing:
CALL 1002H
- Before execution:
CS = 1000H, IP = 0003H,
SS = A000H, SP = FFFFH
- After the execution:
CS = 1000H, IP = 1002H,
SS = A000H, SP = FFFDH



6.4. Subroutines

- What is the result of executing the following

a) RET

Before execution:

CS=1000H, IP = 1006H, SS = A000H, SP = FFFDH

After the execution:

CS=1000H, IP = 0003H, SS = A000H, SP = FFFFH

b) RET 4

It adds a 4 to the SP before removing IP from the stack.



6.4. Subroutines

- Example 3: Trace the following program:

```
MOV SI, OFFSET COMPUTE ;Load SI by the compute
CALL [SI]                ;Call procedure addressed by SI
...
COMPUTE PROC NEAR        ;Procedure is of type near call
    PUSH DX              ;Push DX to the stack
    MOV DX,AX            ;Move AX to DX
    IN AX, DATA         ;Load AX by the input port DATA
    OUT PORT,AX          ;Output (AX) to the output port PORT
    MOV AX, DX           ;Restore the content of (AX)
    POP DX               ;Pop (DX) from the stack
    RET                  ;Return to the next instruction
COMPUTE ENDP            ;End the procedure
```



Lecture_12



6.5. Interrupts

- An interrupt can be:
 - Hardware-generated subroutine call(externally derived)
 - Software-generated subroutine call (internally derived).
- It interrupts the program currently executing by calling the interrupt service subroutine.
- Software interrupt are special types of call instructions in the 8086/8088 microprocessor.

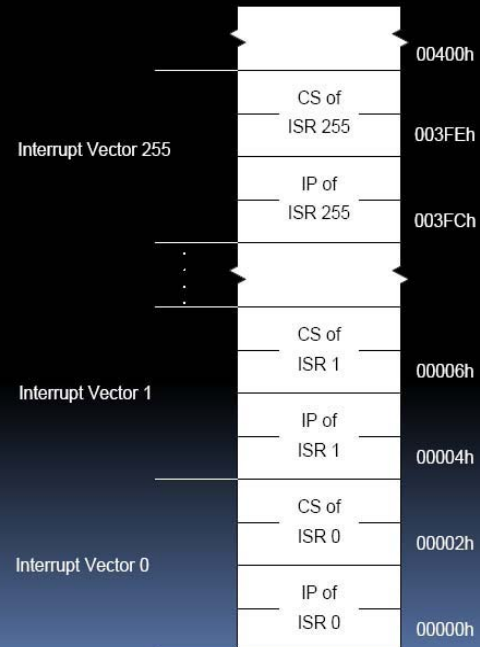


6.5.1. Interrupt Vectors

- An interrupt vector is a 4-byte number stored in the first 1024 bytes of the memory (00000H-003FFH).
- There are 256 interrupt vectors.
- They are used for either hardware or software.
- Each vector contains the address of the interrupt service subroutine, called interrupt.
- The first 2 bytes of the vector contain the number that is loaded into IP register and the next 2 bytes contain the number that is loaded into the CS register in response to an interrupt.

6.5.1. Interrupt Vectors

Interrupt vector table





6.5.1. Interrupt Vectors

- The interrupt vector map is as shown:

Number	Address	Function
0	0H-3H	Divide error
1	4H-7H	Single step
2	8H-BH	NMI (hardware interrupt)
3	CH-FH	Breakpoint
4	10H-13H	Interrupt on overflow
5-31	14H-7FH	Reserved for future use
32-255	80H-3FFH	User interrupts

6.5.1. Interrupt Vectors

<i>Number</i>	<i>Address</i>	<i>Microprocessor</i>	<i>Function</i>
0	0H–3H	All	Divide error
1	4H–7H	All	Angle-step
2	8H–BH	All	NMI pin
3	CH–FH	All	Breakpoint
4	10H–13H	All	Interrupt on overflow
5	14H–17H	80186–Pentium Pro	Bound instruction
6	18H–1BH	80186–Pentium Pro	Invalid opcode
7	1CH–1FH	80186–Pentium Pro	Coprocessor emulation
8	20H–23H	80386–Pentium Pro	Double fault
9	24H–27H	80386	Coprocessor segment overrun
A	28H–2BH	80386–Pentium Pro	Invalid task state segment
B	2CH–2FH	80386–Pentium Pro	Segment not present
C	30H–33H	80386–Pentium Pro	Stack fault
D	34H–37H	80386–Pentium Pro	General protection fault (GPF)
E	38H–3BH	80386–Pentium Pro	Page fault
F	3CH–3FH	—	Reserved
10	40H–43H	80286–Pentium Pro	Floating-point error
11	44H–47H	80486SX	Alignment check interrupt
12	48H–4FH	Pentium/Pentium Pro	Machine check exception
13–1F	50H–7FH	—	Reserved
20–FF	80H–3FFH	—	User interrupts



6.5.2. Interrupt Instructions

- The 8086/8088 has three different interrupt instructions available to the programmer:
INT, INTO and INT 3.
- There are 256 different software interrupt (INT) instructions available to the programmer.
- Each INT instruction has a numeric operand whose values ranges from 0 to 255.
- Each instruction is 2 bytes long, except INT 3 that is 1-byte software interrupt instruction.



6.5.2. Interrupt Instructions

- A software interrupt instruction (INT N) is executed as
 - 1) It pushes the flags onto the stack.
 - 2) It clears the I and T flags.
 - 3) It pushes CS onto the stack.
 - 4) It fetches the new CS location from the vector table.
 - 5) It pushes IP onto the stack.
 - 6) It fetches the new IP location from the vector table.
 - 7) It jumps to this new location.



6.5.2. Interrupt Instructions

- The INT 3 is a 1-byte long instruction.
- Vector number 3 is called breakpoint interrupt.
- It is easy for the programmer to insert a 1-byte instruction at any point in the software.
- The software at the INT 3 interrupt service subroutine displays all the registers and waits.
- It can be used to debug a faulty program.



6.5.2. Interrupt Instructions

- Interrupt on overflow (INTO) is a conditional interrupt instruction.
- If the overflow flag (OF) is set and the INTO instruction is encountered in a program, the subroutine whose address is stored at vector 4 will be called.
- IF OF is clear and INTO instruction is encountered, no interrupt will be called.
- The INTO instruction is most widely used following signed arithmetic to detect overflow error condition.
- It appears after every addition and subtraction.



6.5.2. Interrupt Instructions

- The interrupt return instruction (IRET) is used with the software or hardware interrupt.
- IRET is executed as follows:
 - 1) It pops stack data into IP.
 - 2) It pops stack data back into CS.
 - 3) It pops stack data back into the flags.
- Whenever the IRET pops the flag back into the flag register, the prior contents of IF and TF are restored.



6.5.2. Interrupt Instructions

- Example: Write an interrupt that add the contents of DI, SI, BP and BX and save the result in AX. Each time this function is required, an INT 50 instruction is used to call it.

```
ADDEM:    ADD AX,BX
          ADD AX,SI
          ADD AX,DI
          ADD AX,BP
          IRET
```

```
;Set up vector address ORG 000C8H
```

```
;Set the program at address 200=C8H
```



6.5.3. Interrupt Control

- There are two instructions used to control the hardware interrupt structure:
 - Set interrupt flag (STI)
 - Clear interrupt flag (CLI).
- When the $IF = 0$, the INTR pin is disabled.
- When the $IF = 1$, INTR is enabled.



6.6. Miscellaneous Instructions

- These instructions provide control of the carry bit, sample the $\overline{\text{TEST}}$ pin and perform various other functions.
- Most of these instructions are used in hardware control.
- These instructions include:
 - Controlling the Carry Flag Bit (CF)
 - WAIT
 - HLT
 - NOP
 - LOCK Prefix
 - ESC



6.6.1. WAIT

- WAIT instruction tests the hardware pin
- This pin is used to test a variety of external hardware events.
- If this pin is a logic 1, the 8086/8088 will become idle and wait for it to become a logic 0.



6.6.2. HLT

- HLT stops the execution of software.
- There are only two ways to execute a halt:
 - By interrupt
 - By a hardware system reset.



6.6.3. NOP

- When the microprocessor encounters a NOP, it takes 3 clocking periods to execute.
- NOP is used in time delay software.
- In machine language programs, it is advisable for the programmer to leave patch areas every 50 bytes in case the program needs modification in the future.
- These patch areas normally contain NOP instructions so that the program's operation will not be affected by these patch areas.



6.6.4.LOCK Prefix

- The LOCK prefix is a byte placed before any 8086/8088 instruction to inhibit external coprocessor in the system from gaining access to system buses.



6.6.5. ESC

- ESC is an opcode for an external coprocessor.
- ESC passes information to the 8087 arithmetic coprocessor.
- Whenever the escape instruction is executed, the 8086/8088 performs a NOP and the external coprocessor receives a 6-bit opcode encoded in the ESC instruction.
- The ESC instruction accesses a memory location so that the coprocessor can read or write data if necessary.



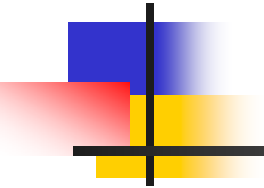
Sample Print-Program


- `six EQU 6`
- `MOV AH,six`
- `MOV DL,'M'`
- `INT 21H`
- `MOV DL,'0'`
- `INT 21H`
- `MOV DL,'h'`
- `INT 21H`
- `MOV DL,'a'`
- `INT 21H`
- `MOV DL,'m'`
- `INT 21H`
- `MOV DL,'e'`
- `INT 21H`
- `MOV DL,'d'`
- `INT 21H`
- `MOV DL,' '`
- `INT 21H`
- `MOV DL,'B'`
- `INT 21H`
- `MOV DL,'a'`
- `INT 21H`
- `MOV DL,'n'`
- `INT 21H`
- `MOV DL,'n'`
- `INT 21H`
- `MOV DL,'a'`
- `INT 21H`



Lecture_13

Part II: 8086/8088 Hardware

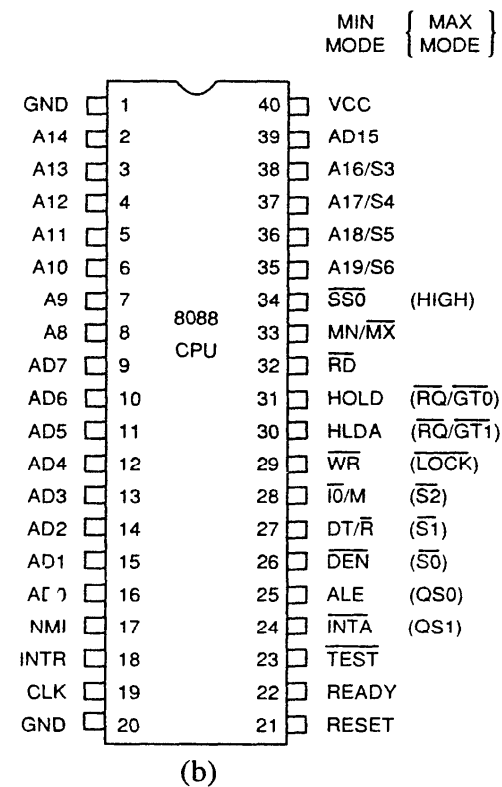
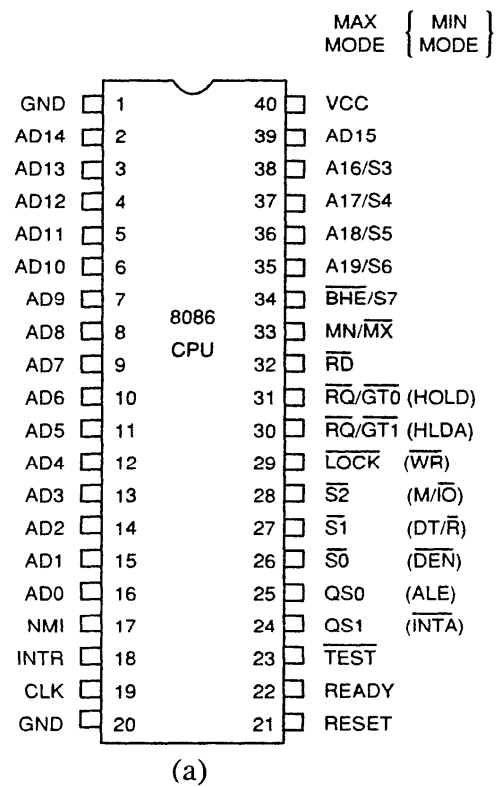




Chapter 1: 8086/8088 Hardware Specifications

- This chapter will discuss:
 - The pin functions of both 8086 and 8088 μ Ps.
 - Clock Generator.
 - Bus buffering
 - Bus latching
 - Timing
 - Wait states
 - Minimum and maximum mode operations.

1.2. Pinouts and the Pin Functions





1.2.1. The Pinout

- Both processors are packaged in 40-pin dual in-line packages (DIPs).
- The differences between the two microprocessors are:
 1. The 8086 has 16-bit data bus (AD_0 - AD_{15}), but the 8088 has 8-bit data bus (AD_0 - AD_7).
 2. In the 8088, pin 34 is an $\overline{SS0}$, while on the 8086, it is a $\overline{BHE/S7}$
 3. The 8086 has M/\overline{IO} , but the 8088 has IO/\overline{M}

•

1.2.2. Power Supply Requirements

- Both microprocessors require +5V with a supply voltage tolerance $\pm 10\%$.
- 8086 draws a maximum supply current of 360 mA, 8088 draws a maximum supply current of 340 mA.
- Both microprocessors operate in ambient temperature between 32°F and 180°F.
- Extended temperature-range versions are available.
- There is a CMOS version (80C86/80C88):
 - It requires a very low supply current (10mA).
 - It has an extended temperature range (-40°F to 225°F).



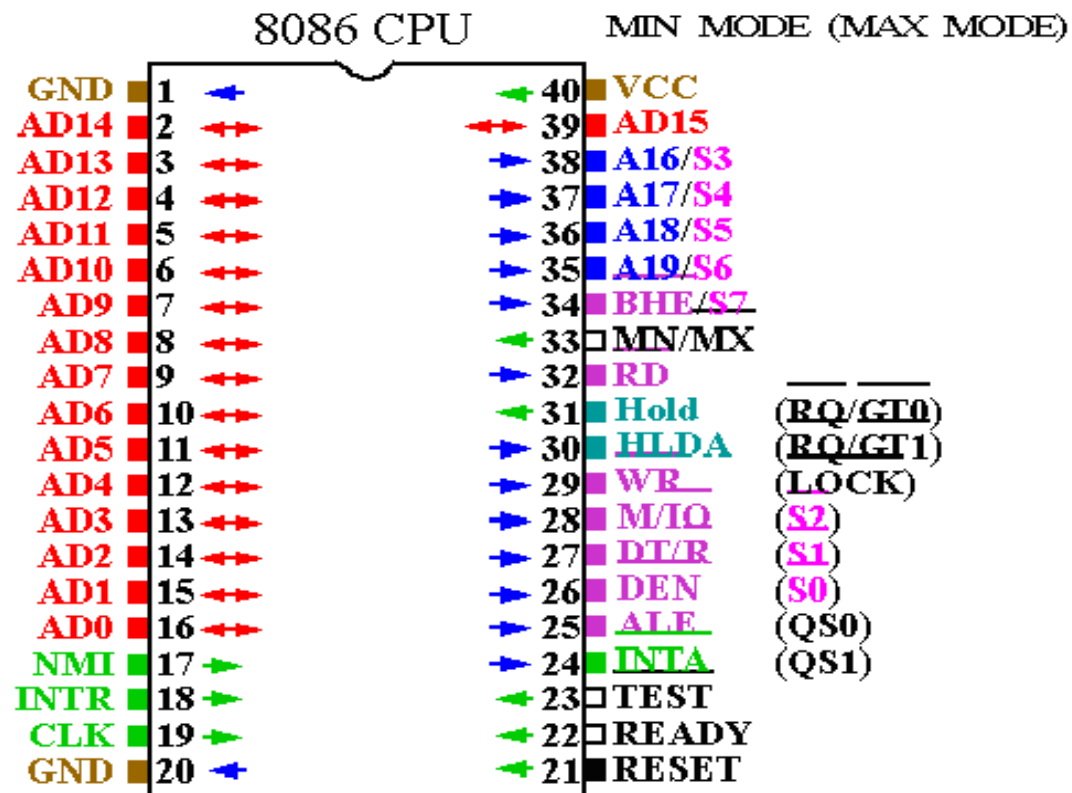
1.2.3. DC Characteristics

- To select the proper interface components, the following characteristics are very important:
 - Input characteristics.
 - Output characteristics.

Input Characteristics			Output Characteristics		
Logic Level	Voltage	Current	Logic Level	Voltage	Current
0	0.8V max	10 μ A max	0	0.45V max	2.0 mA max
1	2.0 V min	10 μ A max	1	2.4V min	-400 μ A max

- ✦ Standard logic gates have a maximum logic 0 output voltage of 0.4V, and the 8086/8088 has a maximum of 0.45V.
- ✦ That may result in more loads on the connection.

1.2.4. 8086 Pin Functions





1.2.4. 8086 Pin Functions

1. Vcc: The +5V, $\pm 10\%$ power supply pin.
2. GND: The ground connection, two pins.
3. CLK (clock): It provides the basic timing.
4. AD₀-AD₁₅: Multiplexed address (ALE=1)/data bus (ALE=0).
5. A16/S3-A19/S6 (multiplexed Address/Status): They carry the address during ALE and the status for the remainder cycle.

1. S6 is always 0.
2. S5 indicates the condition of the IF.
3. S4 and S3 refer to the used segment.

S4	S3	Function
0	0	ES
0	1	SS
1	0	CS or No segment
1	1	DS



1.2.4. 8086 Pin Functions

6. \overline{RD} (Read): It becomes logic 0, when the data bus receives (reads) data from the memory or I/O port.
7. \overline{WR} Write: It becomes logic 0, when the data bus drives (writes) data to the memory or I/O port.
8. ALE (Address Latch Enable): When it is logic 1, the address/data bus contains a memory or I/O address.
9. DT/\overline{R} (Data Transmit/Receive): It is used to control the direction of the data bus transmitting/receiving data.
10. \overline{DEN} (Data bus Enable): When it is logic 0, the address/data bus contains a memory or I/O data



1.2.4. 8086 Pin Functions

11. \overline{TEST} (test): If it is logic 0, the program will be executed. If it becomes a logic 1, WAIT will wait for it to become 0.
12. READY: It is logic 1 so the instructions are executed without wait states. If it is logic 0, the wait state will be inserted.
13. NMI (Nonmaskable Interrupt): When it becomes logic 1, INT 2 will be called at the end of the current instruction.
14. INTR (Interrupt Request): If INTR is held high during IF = 1, the 8086/8088 enters into an interrupt acknowledge cycle after the current instruction is completed.
15. \overline{INTA} (Interrupt Acknowledge): It is generated by the microprocessor in response to INTR. It causes the interrupt vector to be put onto the data bus.



1.2.4. 8086 Pin Functions

16. MN/\overline{MX} (Minimum/Maximum Mode): It is used to select minimum or maximum mode operation.
17. M/\overline{IO} (Memory or I/O): It indicates if the address bus contains memory or I/O address.
18. RESET (Reset): if it is held high for 4 clock cycles, the 8086 will be reset and restart at FFFF0H.
19. HOLD (Hold): It requests a direct memory access (DMA). When 1, microprocessor stops and places address, data and control bus in high-impedance state.
20. HLDA (Hold Acknowledge): It indicates that the microprocessor has entered the hold state.



1.2.4. 8086 Pin Functions

21. $\overline{RQ}/\overline{GT}_0$ and $\overline{RQ}/\overline{GT}_1$ (Request/grant) pins direct memory accesses (DMA) during maximum mode operation.
22. \overline{LOCK} (lock): An output is used to lock peripherals off the system. Activated by using the LOCK: prefix on any instruction.
23. QS_1 and QS_0 (Queue Status): These bits show status of internal instruction queue.

QS_1	QS_0	Function
0	0	Indicates no operation
0	1	Indicates first byte of opcode from queue.
1	0	Empties the queue.
1	1	Indicates subsequent byte from queue.



1.2.4. 8086 Pin Functions

24. $\overline{S_2}, \overline{S_1}, \overline{S_0}$ (Status): They Indicate function of current bus cycle (decoded by 8288).

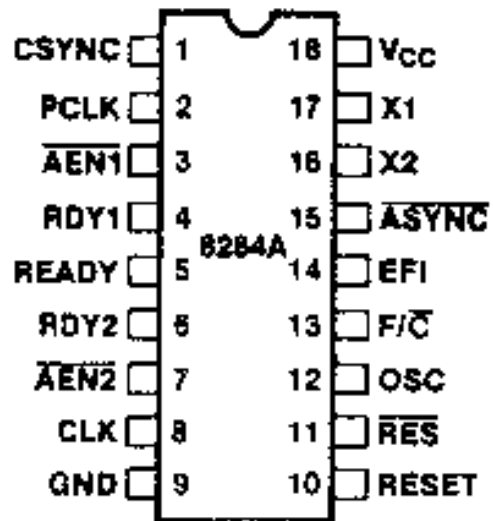
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Function
0	0	0	Indicates an interrupt acknowledge
0	0	1	Indicates an I/O read
0	1	0	Indicates an I/O write
0	1	1	Indicates a halt
1	0	0	Indicates a code access
1	0	1	Indicates a memory read
1	1	0	Indicates a memory write
1	1	1	Remains passive



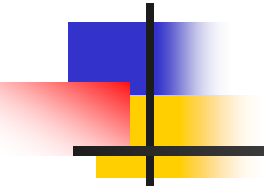
1.3. Clock Generator (8284A)

- The 8284A is an ancillary component to the 8086/8088.
- It provides the following functions:
 - Clock generation
 - RESET synchronization
 - READY synchronization.
 - A TTL level peripheral clock signal.

1.3.1. Pin Functions



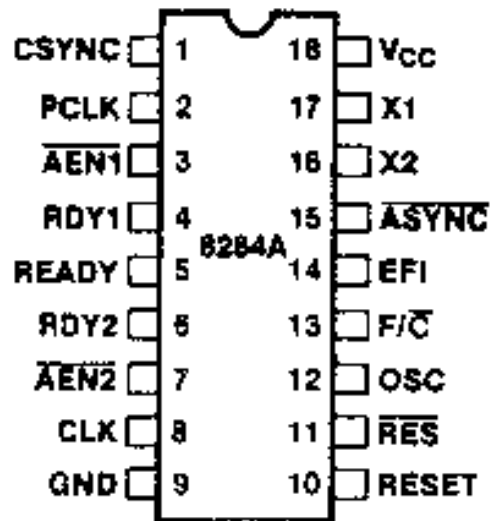
Part II: 8086/8088 Hardware





Lecture_14

1.3.1. Pin Functions





1.3.1. Pin Functions

1. Vcc: The +5V, $\pm 10\%$ power supply pin.
2. GND: The ground connection
3. F/\bar{C} (Frequency/Crystal): it is used to select the clocking source for the 8284A.
4. X₁ and X₂: (Crystal inputs): They are connected to an external crystal.
5. EFI (External Frequency Input): An input used to supply the timing.
6. CLK (clock): It provides the CLK input signal to the 8086/8088 and other devices.



1.3.1. Pin Functions

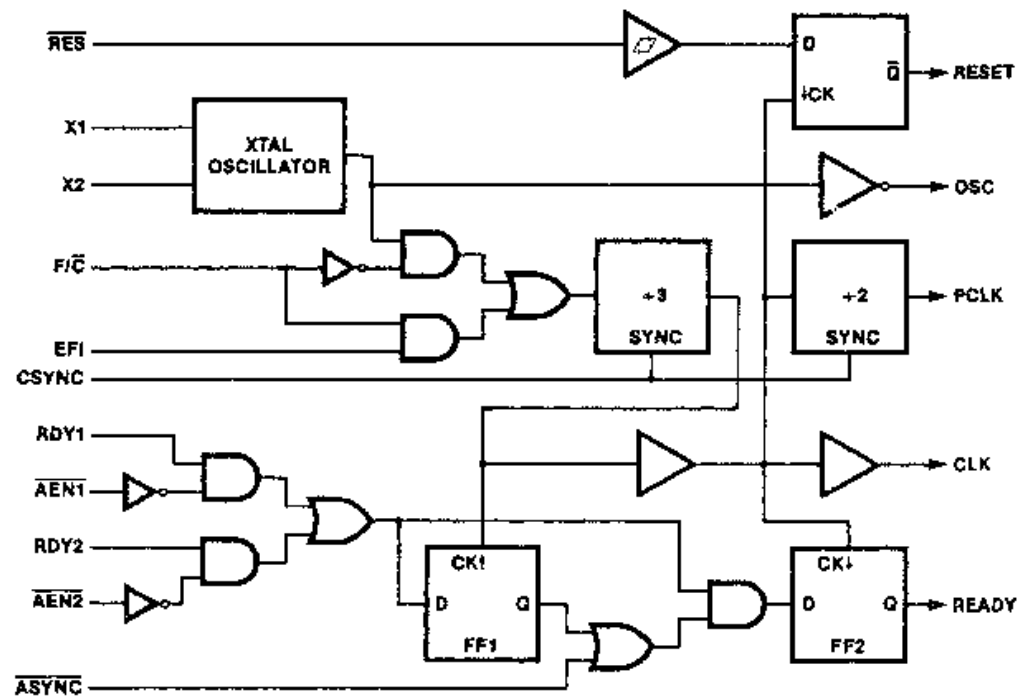
7. PCLK (Peripheral Clock): It provides a clock signal to the peripherals.
8. OSC (Oscillator Output): It provides an EFI to other 8284A clock generators.
9. CSYNC (Clock Synchronization): It is used whenever EFI input provides synchronization in systems with multiple processors.
10. \overline{RES} (Reset Input): It is often connected to an RC network that provides power-on resetting.
11. RESET (Reset Output): It is connected to the 8086/8088 RESET input pin.



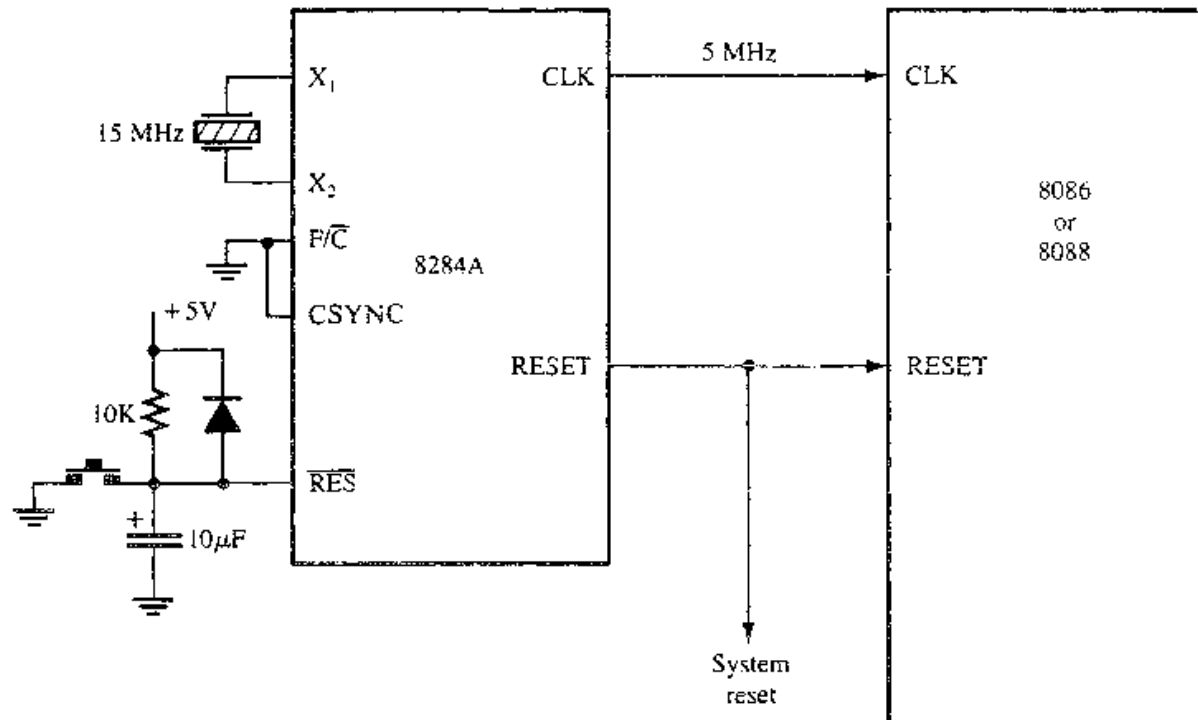
1.3.1. Pin Functions

12. $\overline{AEN}_1, \overline{AEN}_2$ (Address Enable): They are provided to qualify the bus ready signals RDY_1 and RDY_2 .
13. RDY_1 and RDY_2 (Bus Ready): They cause wait states in an 8086/8088-based system.
14. \overline{ASYNC} (Ready Synchronization Select): It is used to select either one or two stages of synchronization for the RDY_1 and RDY_2 .
15. $READY$ (Ready): An output pin connects to the 8086/8088 $READY$ input. It is synchronized with the RDY_1 and RDY_2 inputs.

1.3.2. Operation of the Clock



1.3.3. Operation of the RESET

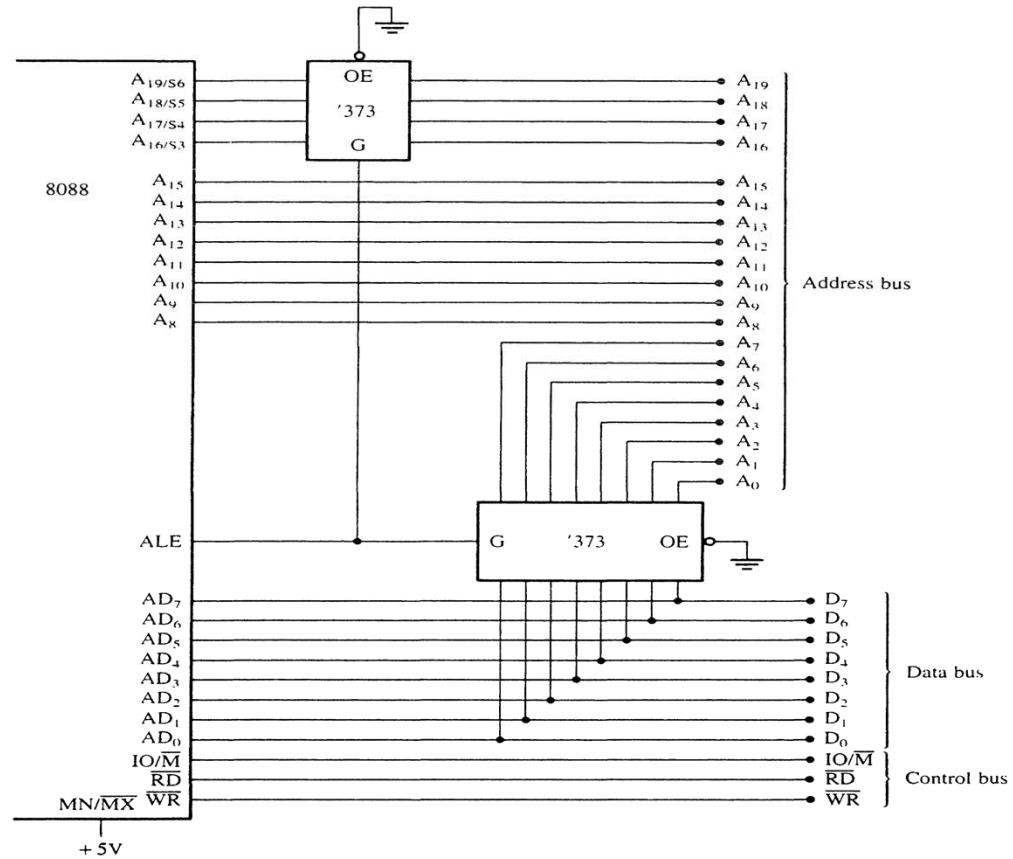




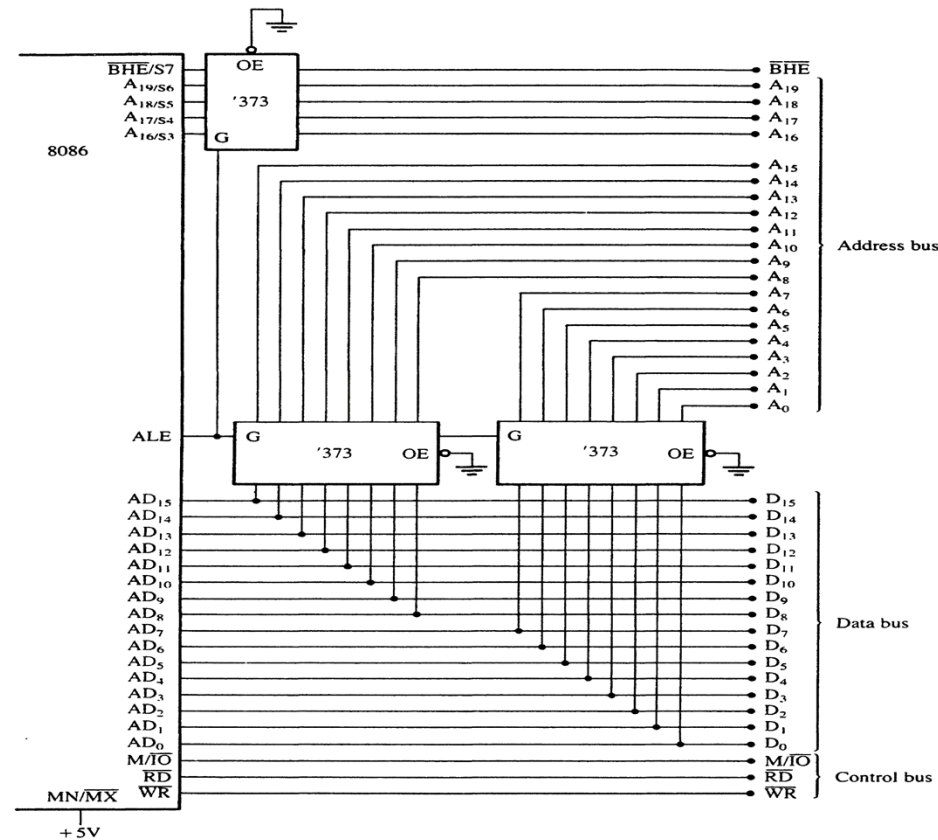
1.4. Bus Buffering and Latching

- For very large systems, the buses are buffered. WHY?
- The address/data bus are multiplexed to save the number of pins required for the 8086/8088 IC.
- They must be demultiplexed. WHY?
- All computer systems have three buses:
 - Address bus
 - Data bus
 - Control bus
- These buses must be present to interface memory and I/O.

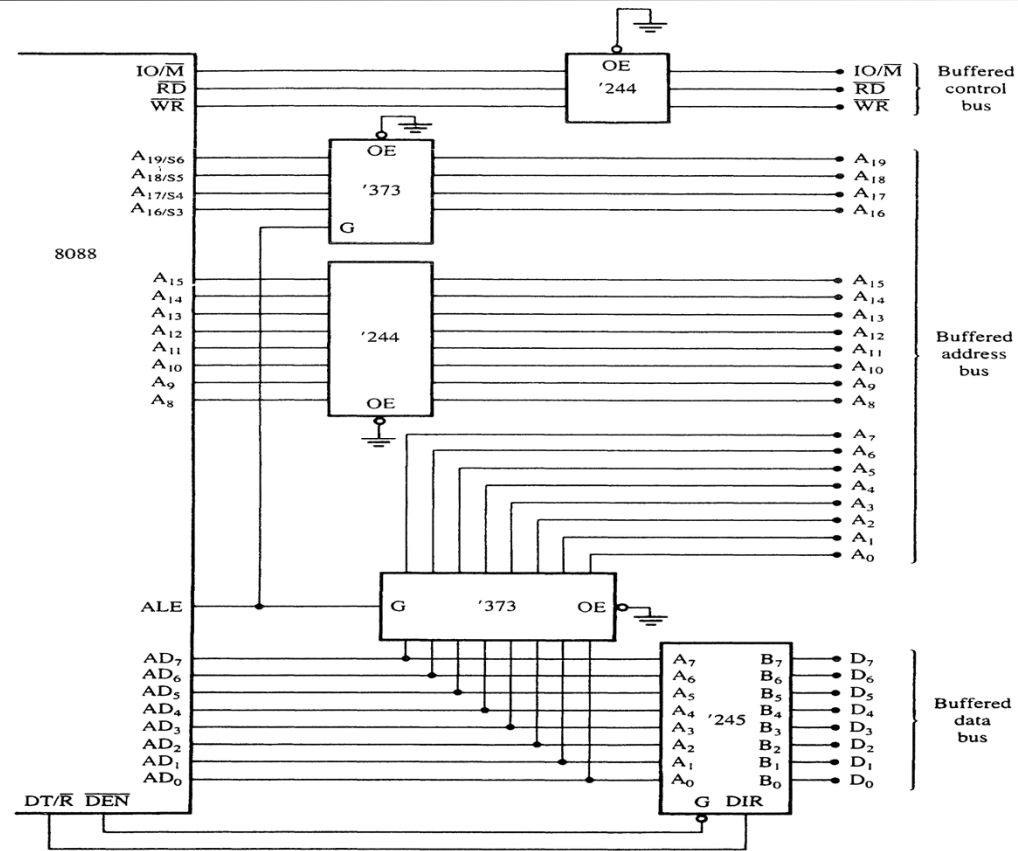
1.4.1 Demultiplexing the 8088



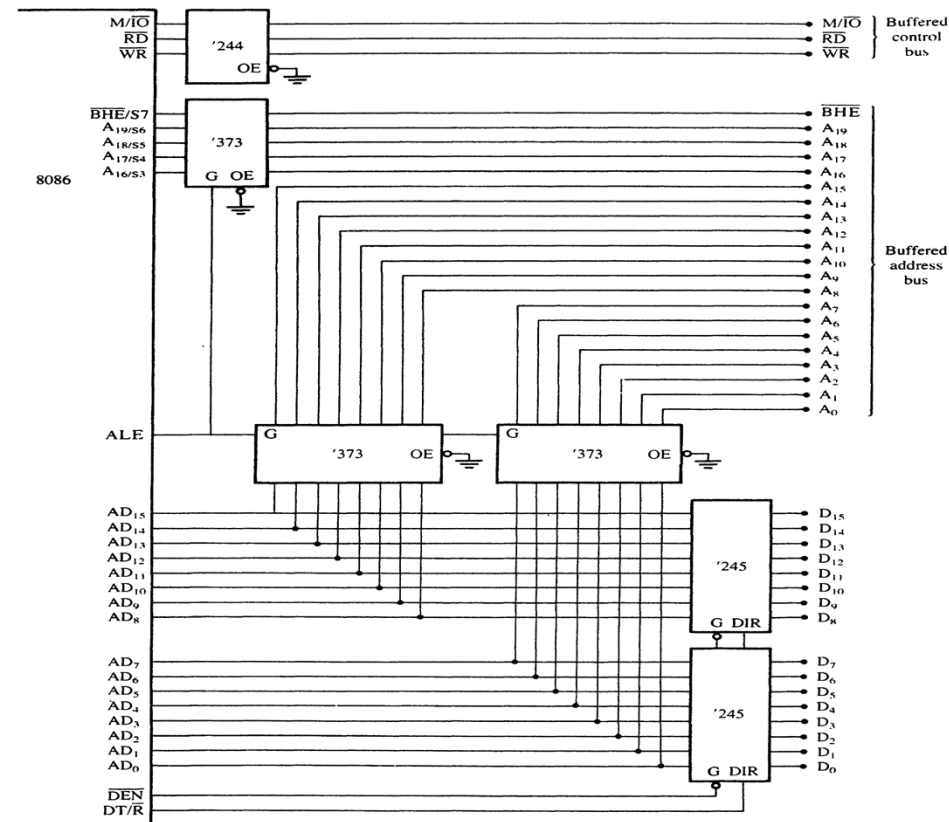
1.4.2. Demultiplexing the 8086



1.4.3. The Fully Buffered 8088



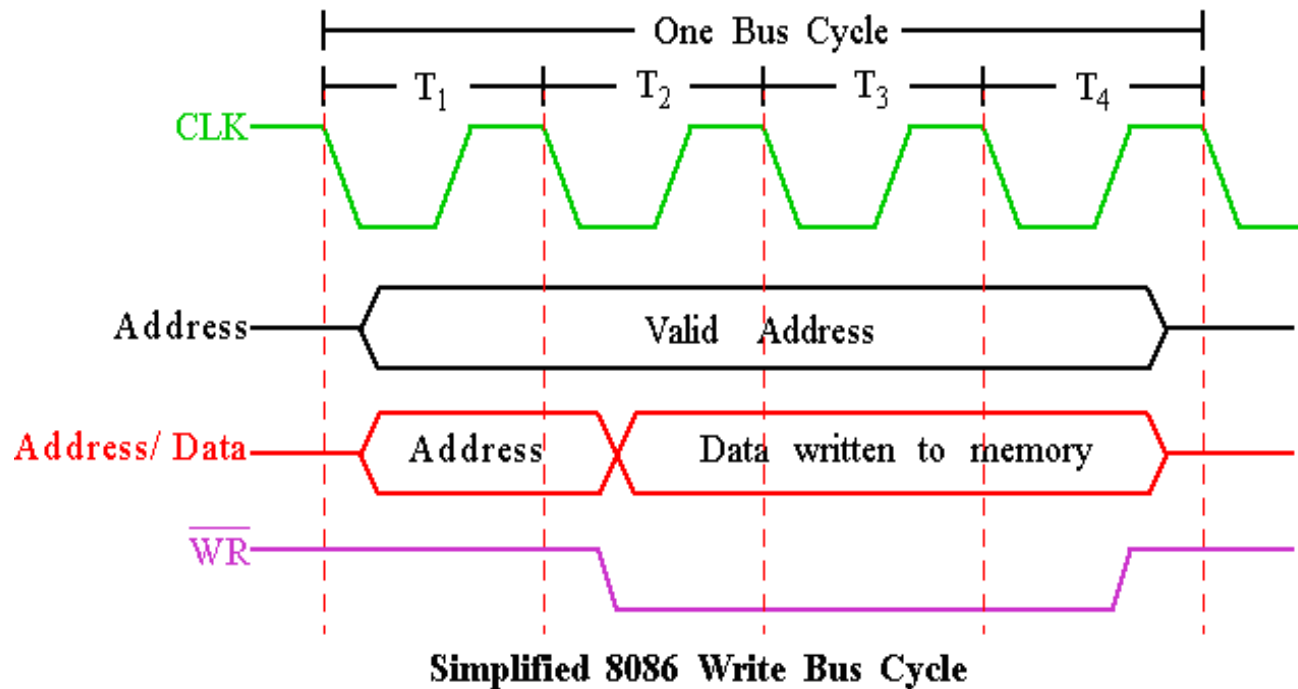
1.4.4. The Fully Buffered 8086



1.5. Bus Timing

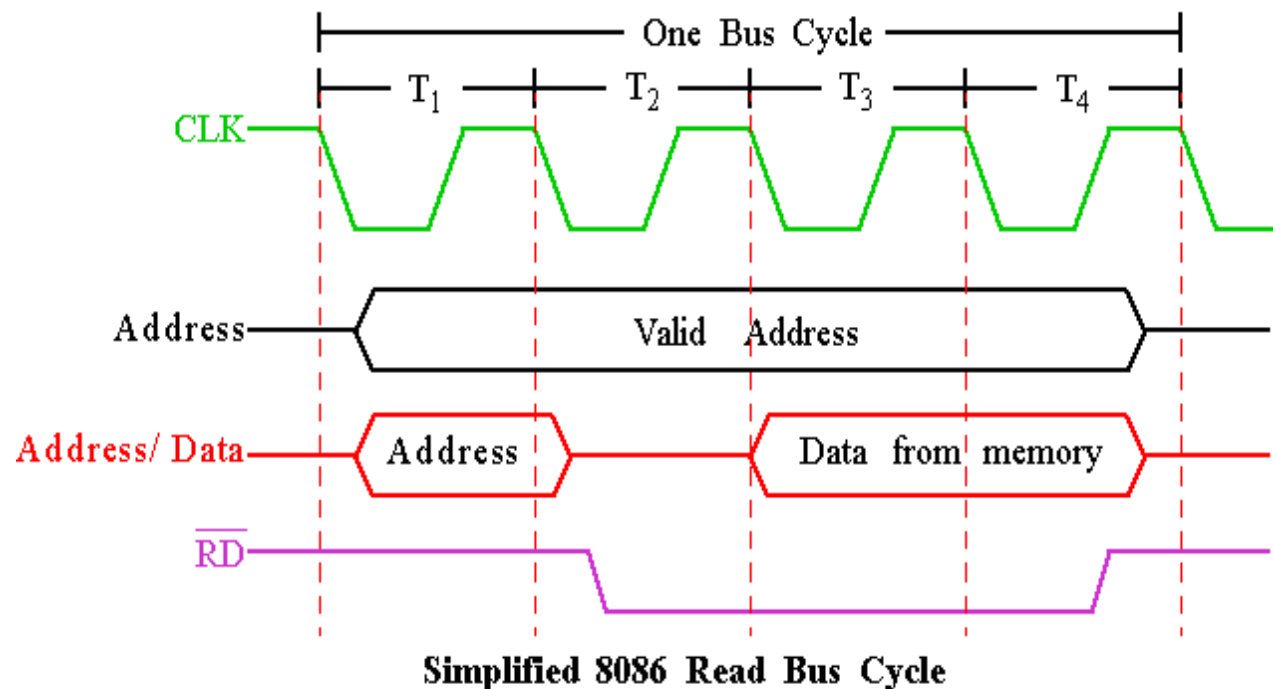
1.5.1. Basic Bus Operation

- Simplified 8086/8088 write bus cycle:



1.5.1. Basic Bus Operation

- Simplified 8086/8088 read bus cycle:





1.5.2. Timing In General

- The 8086/8088 uses the memory and I/O in periods of time called bus cycle.
- Each bus cycle equal to 4 system-clocking periods (T states).
- If the clock is operated at 5 MHz, one bus cycle is completed in 800ns.
- The 8086/8088 reads or writes data at the rate of 1.25 million times a second.



1.5.2. Timing In General

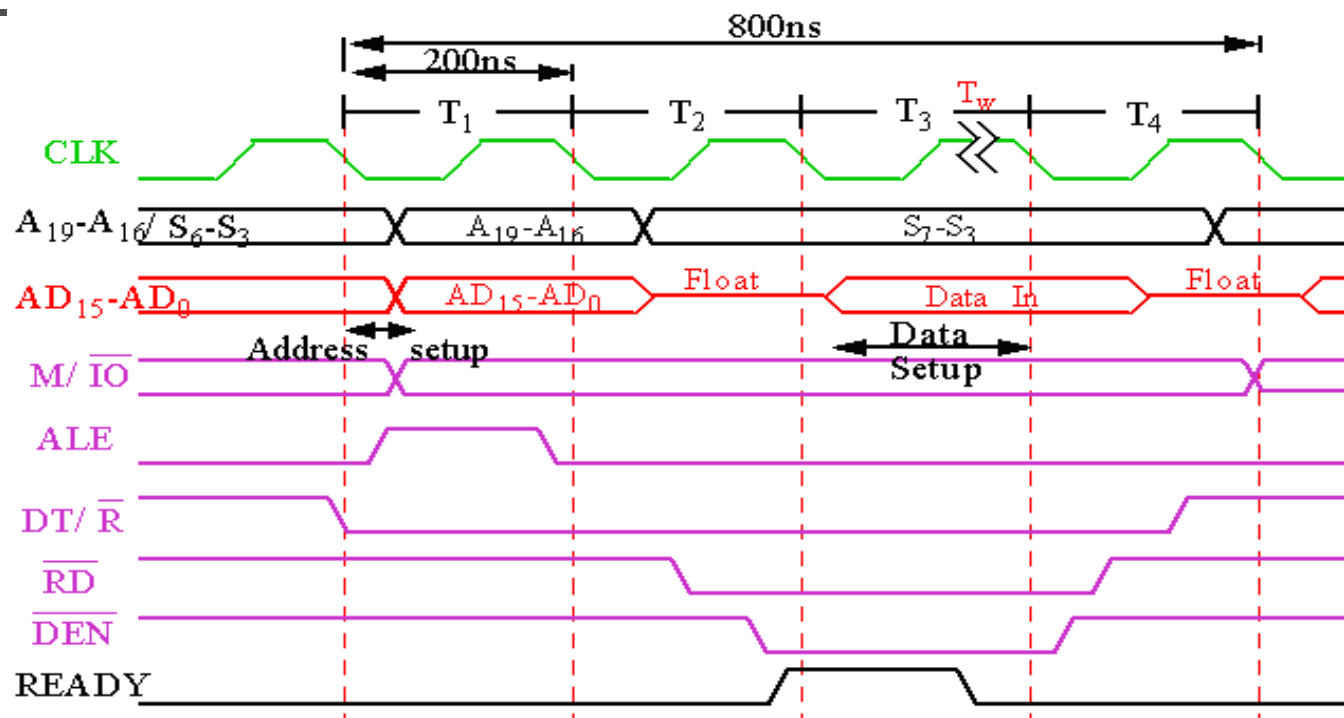
- During the first clocking period (T1):
 - The address is placed on the Address/Data bus.
 - Control signals (M/ IO, ALE and DT/ R) specify memory or I/O, latch the address onto the address bus and set the direction of data transfer on data bus.
- During the second clocking period (T2):
 - The 8086/8088 issues the RD or WR signal for read or write the data.
 - The 8086/8088 issues DEN which enables the memory or I/O device to receive the data for writes and the 8086/8088 to receive the data for reads.



1.5.2. Timing In General

- During the third clocking period (T3):
 - This cycle is provided to allow memory to access data.
 - READY is sampled at the end of T2 .
 - If low, T3 becomes a wait state.
 - Otherwise, the data bus is sampled at the end of T 3 .
- During the fourth clocking period (T4):
 - All bus signals are deactivated, in preparation for next bus cycle.
 - Data is sampled for reading.
 - Data writes occur for writing.

1.5.3. Read Timing



Bus Timing for a Read Operation



1.5.3. Read Timing

- Memory specs (memory access time) must match constraints of system timing.
- Access time is the amount of time that the microprocessor allows the memory to access the data for the read operation.
- For example, bus timing for a read operation shows almost 600ns are needed to read data.
- However, memory must access faster due to setup times, e.g. address setup and data setup.
 - This subtracts off about 110ns .
 - Therefore, memory must access in at least 460ns minus another 30ns guard band for buffers and decoders.

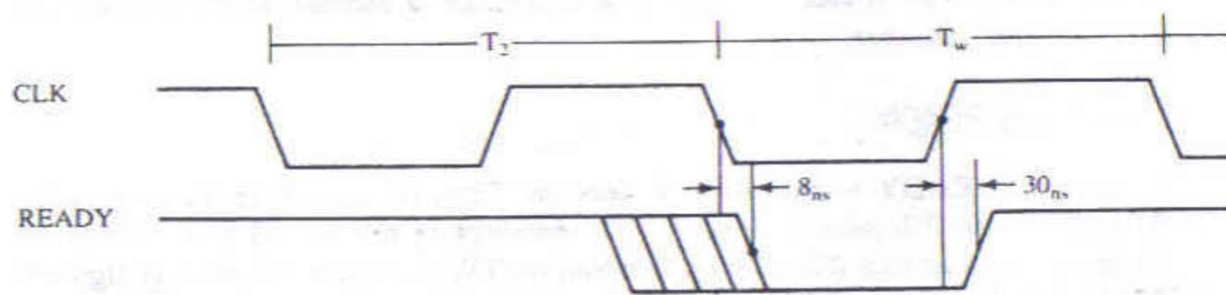


1.6. READY and the Wait State

- READY input causes wait states for slower memory and I/O components.
- A wait state (T_w) is an extra clocking period to stretch the bus cycle.
- If one wait state is inserted, the memory access time will stretch to $(460 + 200) 660$ ns.

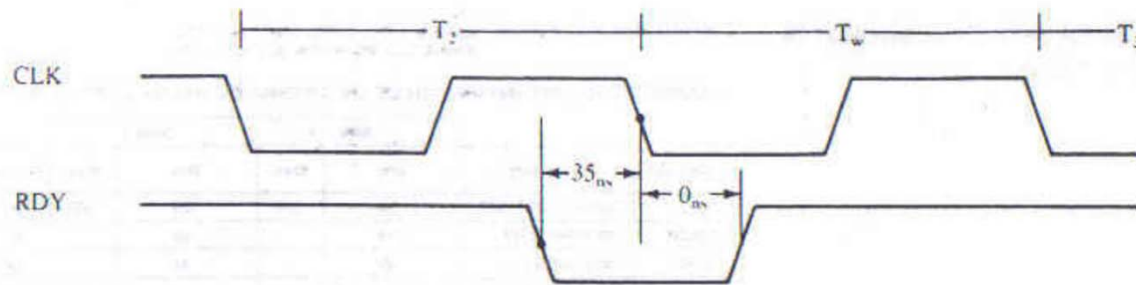
1.6.1. The READY Input

- The READY input is sampled at the end of T_2 and again, if applicable, at the middle of T_w .
- If READY is logic 0 at the end of T_2 , T_w will be inserted between T_2 and T_3 .
- READY is sampled again at the middle of T_w to determine the next state will be T_w or T_3 .

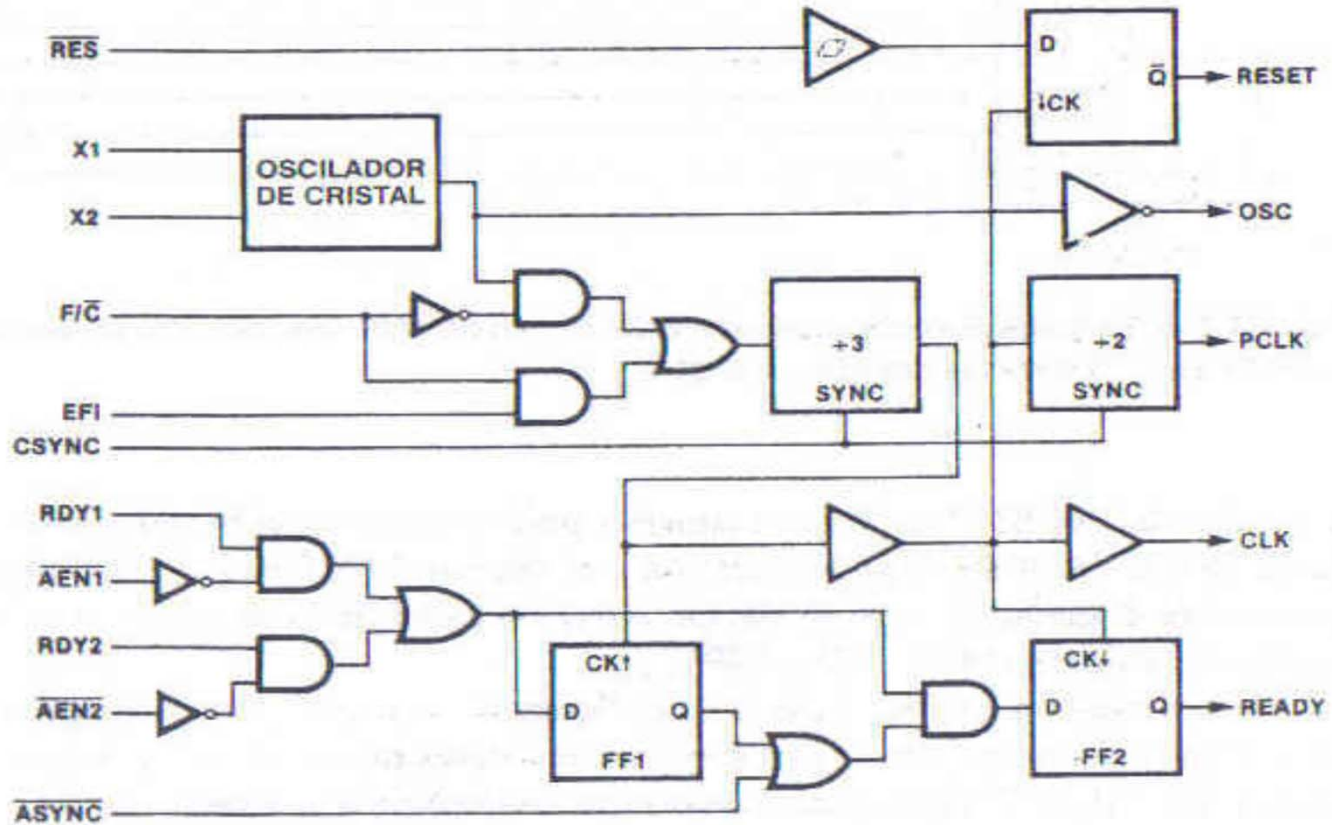


1.6.2. RDY and the 8284A

- RDY is the synchronized ready input to the 8284A.
- The timing diagram for this input is as shown:

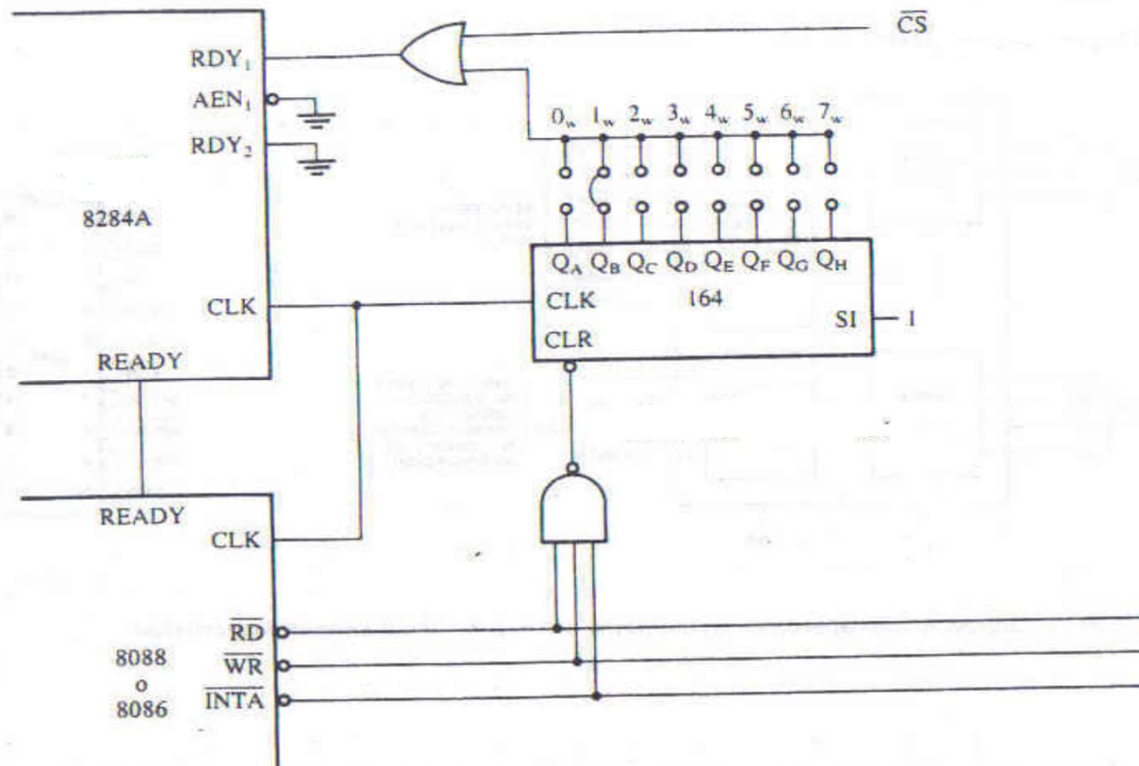


1.6.2. RDY and the 8284A

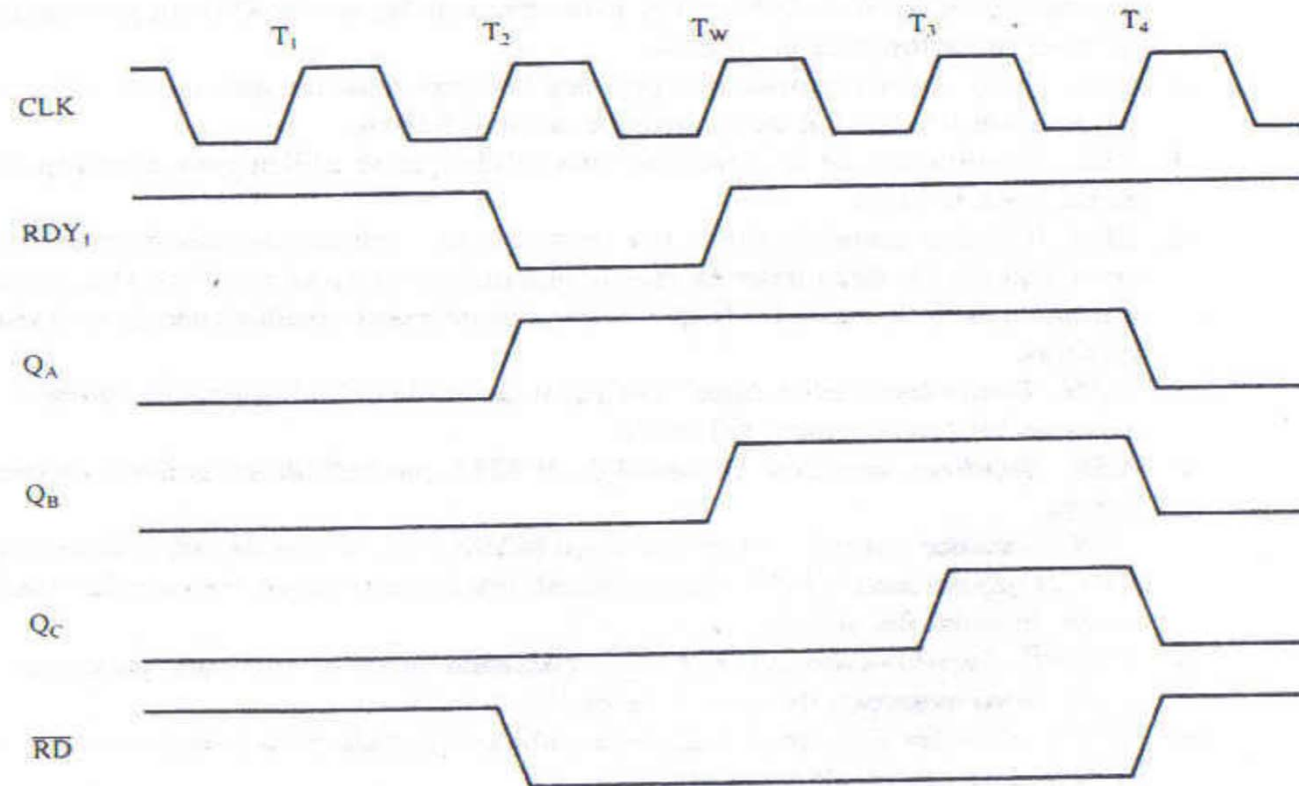


1.6.2. RDY and the 8284A

- A circuit that will cause between 0 and 7 wait states.



1.6.2. RDY and the 8284A





1.7. Minimum Mode versus Maximum Mode

- Minimum mode:
 - It is the least expensive way to operate the 8086/8088.
 - It costs less because all the control signals are generated inside the microprocessor.
 - It allows the 8085A peripherals to be used.
- Maximum mode:
 - It is dropped from the Intel family beginning from 80286.
 - All the control signals must be externally generated.
 - An external bus controller is used.
 - It is used only when the system contains external coprocessor.



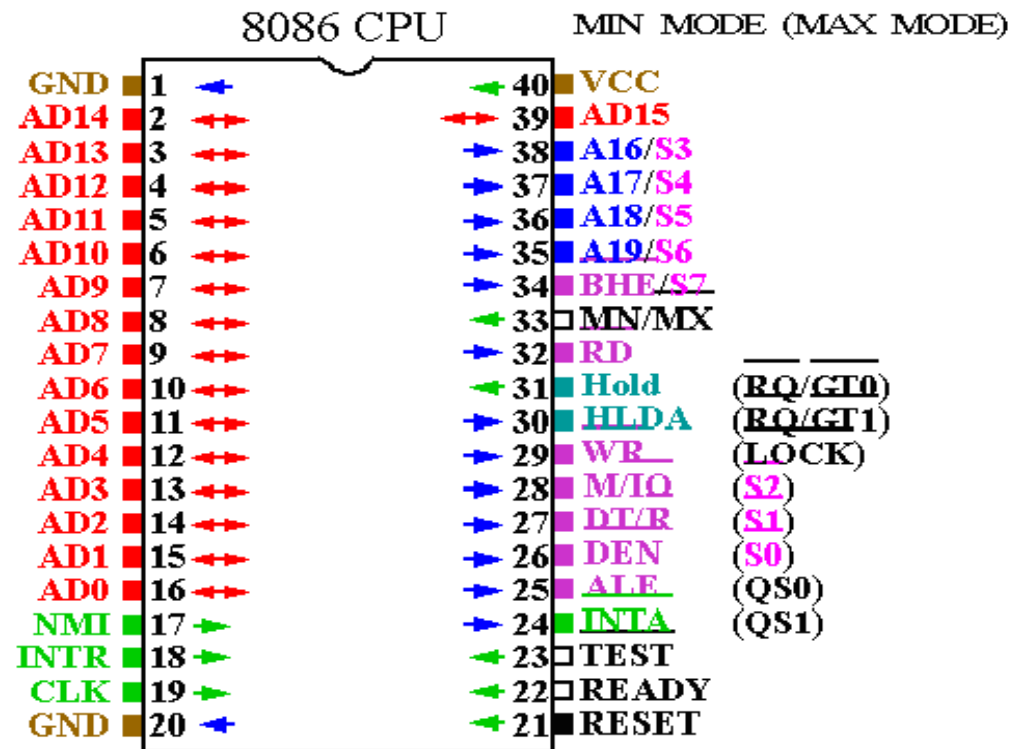
Lecture_15



Chapter 2: Memory Interface

- This chapter will discuss:
 - Memory Devices:
ROM, EEPROM, SRAM, DRAM.
 - Addressing Decoding.
 - 8088 Memory Interface.
 - 8086 Memory Interface.

2.0 8086 Pin Functions

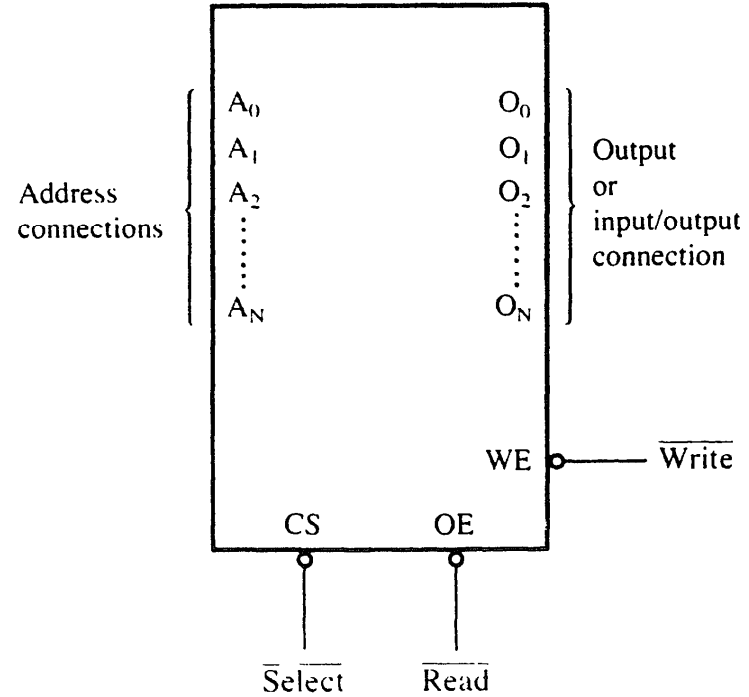




2.1. Introduction

- Every microprocessor-based system has memory system.
- All systems contain two types of memory:
 - Read-only memory (ROM): It stores system software and permanent system data.
 - Random access memory (RAM): It stores temporary data and application software.

2.2. Memory Devices



2.2.1. Memory Pin Connections

a) Address Connections

- All memory devices have address inputs.
- They select a memory location within the memory device.
- Address inputs are labeled from A_0 to A_N .
- N is the total number of address pins minus 1.
- Example: The 2K memory:
 - It has 11 address lines.
 - The labels are (A_0-A_{10}) .
 - If the start address is 10000H so the end address is:
 $10000H + ((2*1024)_{10} = 800H) - 1 = 107FFH$

2.2.1. Memory Pin Connections

b) Data Connections

- All memory devices have a set of data outputs or input/outputs.
- They are used to enter the data for storage or extract the data for reading.
- Data pins are labeled from D_0 through D_7 for a byte-wide memory.
- That means the memory stores 8 bits of data in each memory location.
- Memory devices are defined by memory locations times bit per location.
- Examples: $1K \times 8$, $16K \times 1$, $64K \times 4$. Memory Organization
- Capacity : 8Kbits, 16Kbits, 256Kbits



2.2.1. Memory Pin Connections

c) Selection Connections

- Each memory device has one or more inputs that selects or enables the memory device.
- They can be ChipSelect(\overline{CS}), ChipEnable(\overline{CE}) or Select(\overline{S})
 - ✦ If they are active (logic 0), the memory device performs a read or write operation.
 - ✦ If they are inactive (logic 1), the memory is disabled and do not do any operation.
 - ✦ If more than one selection connection is present. All must be activated to read or write.



2.2.1. Memory Pin Connections

d) Control Connections

- All memory devices have some control input or inputs
- Most ROM has output Enable (\overline{OE}) or (\overline{G})
- If both (\overline{OE}) and (\overline{CE}) are active (logic 0), the output is enabled
- If (\overline{OE}) is inactive, the output is disabled
- A RAM device has either one or two control inputs
- If one control input, it is called R/\overline{W}
- If two control inputs, they are labeled \overline{WE} (or \overline{W}), and \overline{OE} (or \overline{G})



2.2.2. ROM Memory

- ROM permanently stores system software.
- It does not change when the power is disconnected.
- It is called nonvolatile memory.
- It is available in different forms:
 - PROM (Programmable Read-Only Memory).
 - EPROM (Erasable Programmable Read-Only Memory).
 - EEPROM
 - Flash Memory, faster but erasable in large pieces.



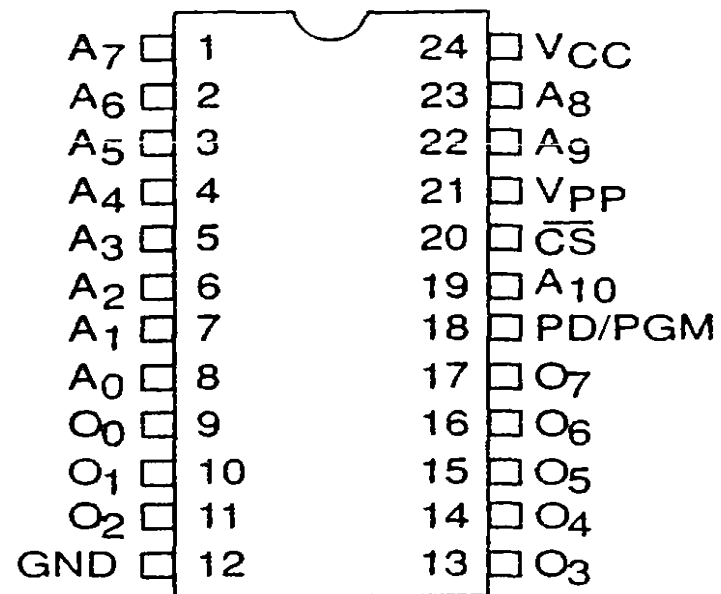
2.2.2. ROM Memory

- The 27XXX series of the EPROM includes:
 - 2704 (512 × 8).
 - 2716 (2K × 8).
 - 2764 (8K × 8).
 - 27256 (32K × 8).
 - 271024 (128K × 8).
 - 2708 (1K × 8).
 - 2732 (4K × 8).
 - 27128 (16K × 8).
 - 27512 (64K × 8).
- Each EPROM has:
 - Address pins
 - 8 data connections
 - One or more selection inputs and one output enable pin. Chip Enable (\overline{CE}), Output Enable (\overline{OE})

2.2.2. ROM Memory

Example: 2716 (2K × 8) EPROM

PIN CONFIGURATION



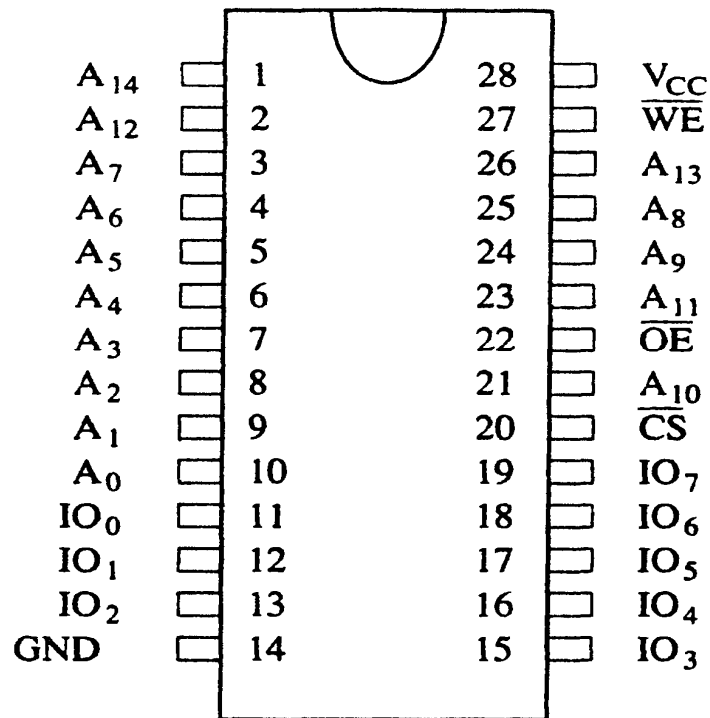


2.2.3. Static RAM

- Static RAM memory devices retain data as long as the DC power is applied.
- They are called volatile memory. Because they will not retain data without power.

2.2.3. Static RAM

- Example: 52256 (32K × 8)



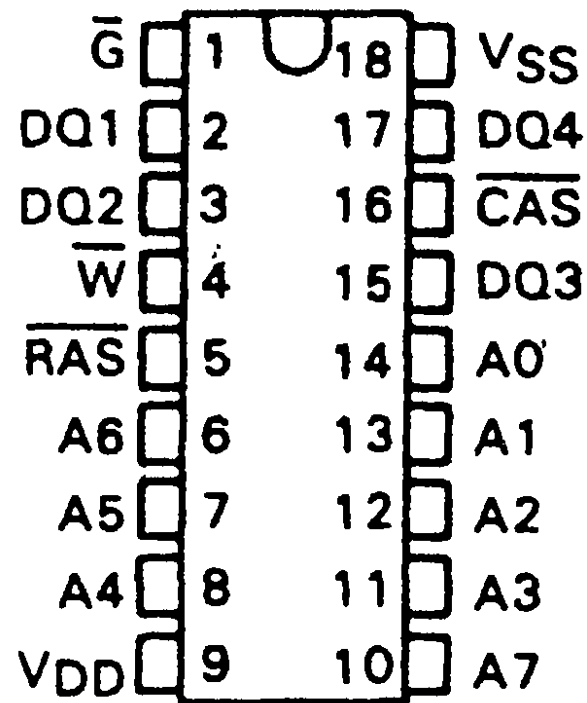


2.2.4. Dynamic RAM

- DRAM retains the data for only 2 or 4 ms on an integrated capacitor.
- In the DRAM, the entire contents of the memory are refreshed with 256 reads in 2 or 4ms interval.
- Refreshing also occurs during a write, a read or during a special refresh cycle.

2.2.4. Dynamic RAM

- Example: TMS4464 (64K × 4)





2.2.4. Dynamic RAM

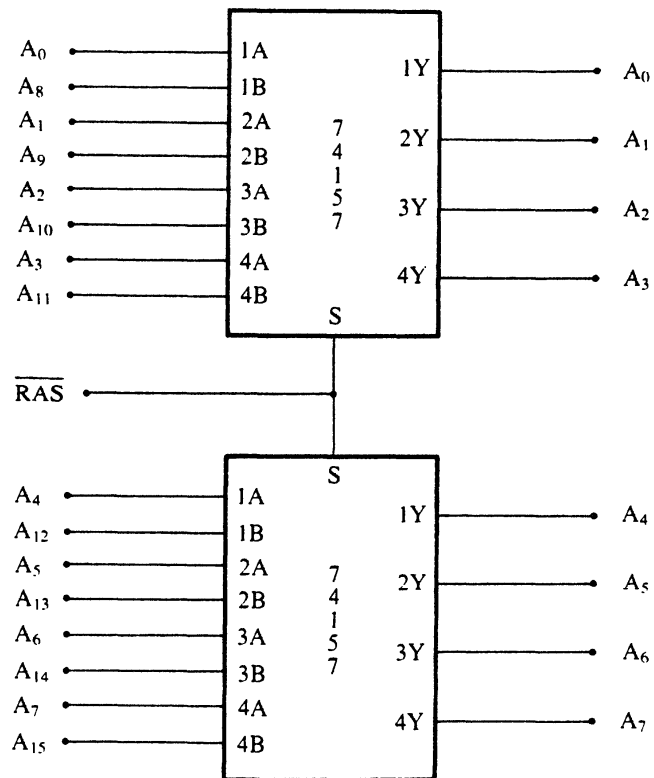
- When RAS is active (0), the first eight bits are placed on the pins and strobed into an internal row latch.
- When CAS is active (0), the next eight bits are placed on the pins and strobed into an internal column latch.
- The 16-bit address is held in the internal latches.
- The CAS performs the function of the chip select.

2.2.4. Dynamic RAM

Address Multiplexer

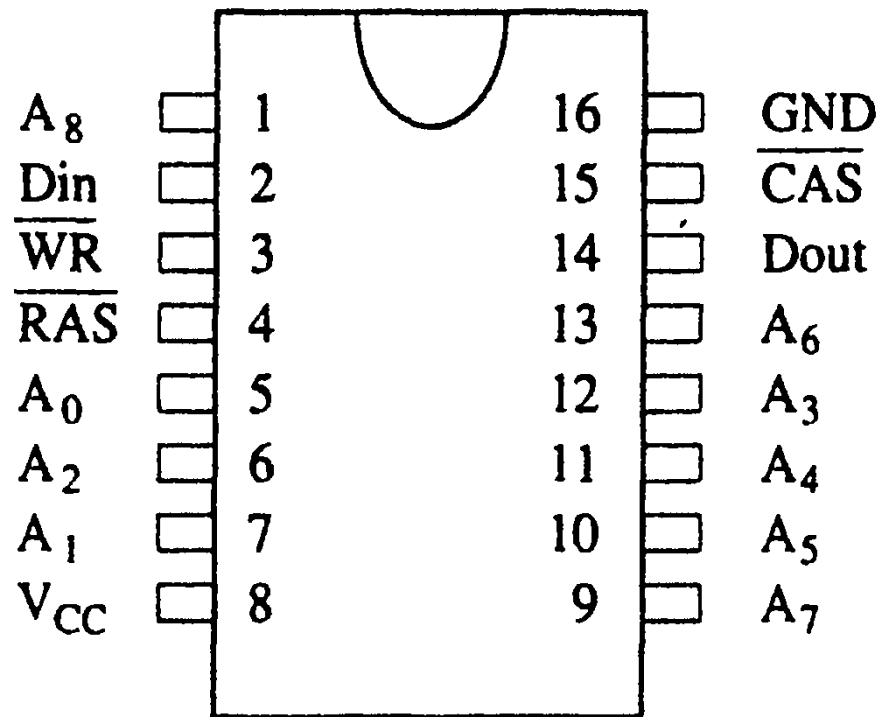
$\overline{RAS} = 0$ As go to Ys

$\overline{RAS} = 1$ Bs go to Ys

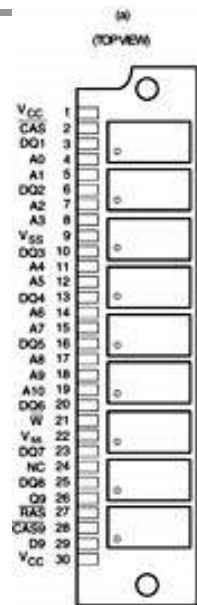


2.2.4. Dynamic RAM

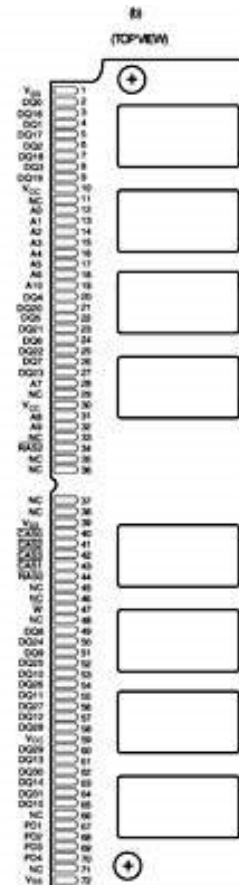
- Example: 41256 (256K × 1)



30- and 72-pin SIMM DRAM

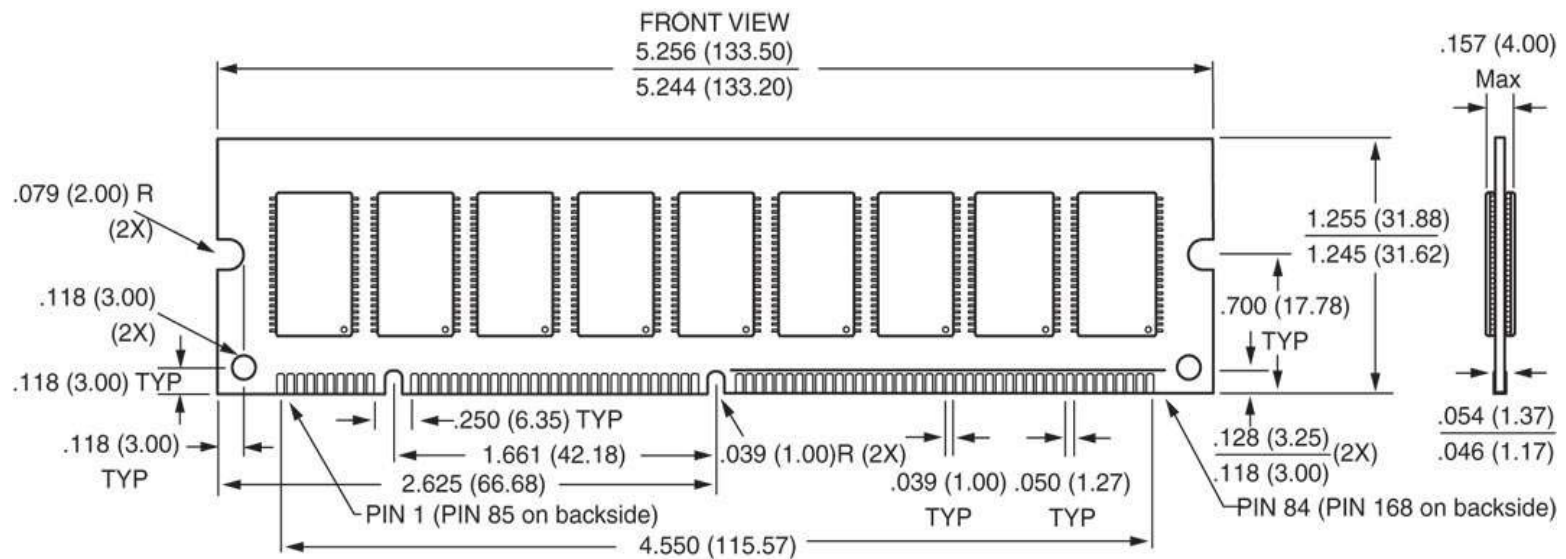


4M x 9



4M x 36

168-pin DIMM DRAM XM x 64

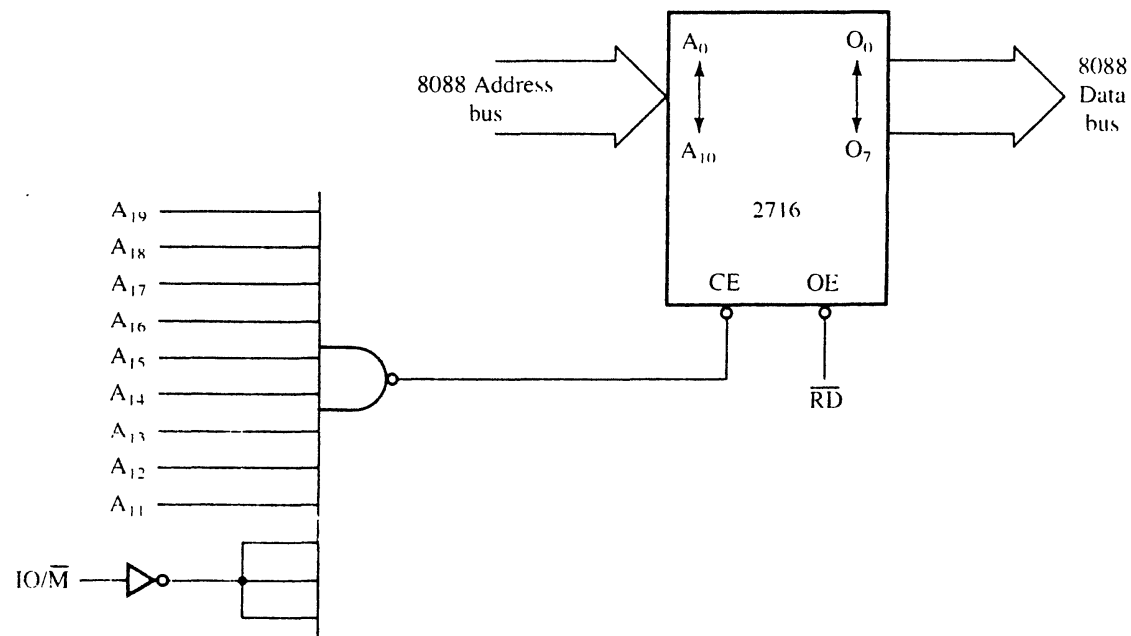




2.3. Address Decoding

- The microprocessor has 20 bits address.
- The memory device has less than 20 bits address.
- The solution is to use decoder.
- For a 2KB chip only 11 bits are connected to the memory and the other 9 bits are decoded.
- A simple NAND Gate decoder is shown

2.3. Address Decoding



- The address range is: from 1111 1111 1000 0000 0000
to 1111 1111 1111 1111 1111

2.3.2. Sample Decoder Circuit

- The 74LS138 can be used to connect 8X8KB EPROMs.

