

The Microprocessor and its Architecture

Chapter

Chapt. 2

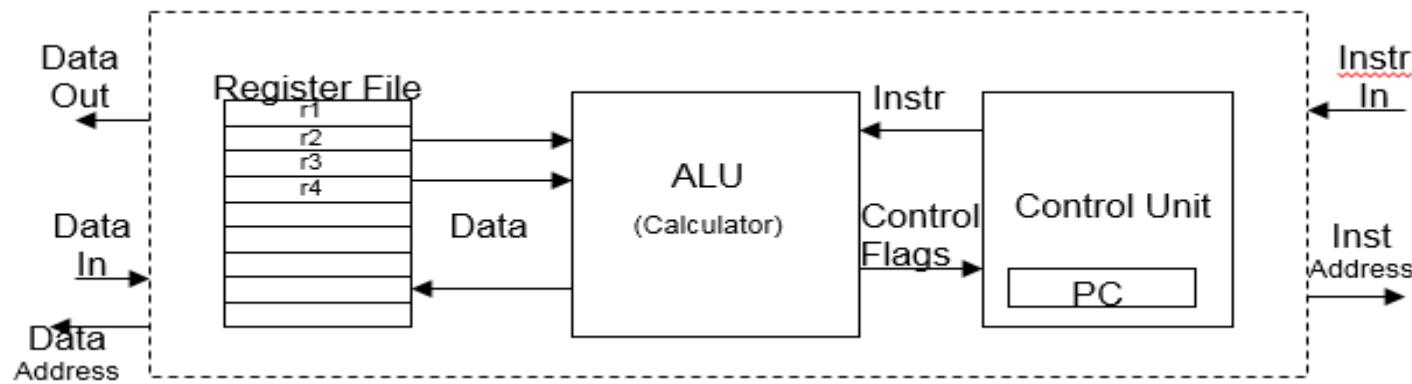
Chapter 2

Topics

- 2.1 Internal microprocessor architecture – micro-architecture
 - ✓ The programmer's model, i.e. the registers model(GP ,Segment Register and Flag)
 - ✓ The processor (organization) model(EU & BIU)
- 2.2 Von Neumann architecture. We will Discuss in chap 6
- 2.3 Real mode memory addressing
- 2.4 Introduction to protected mode memory addressing
- 2.5 Execution and bus interface units

Intro

- Control Unit
 - Fetches instructions from memory, Interprets them, Controls ALU
- ALU
 - Does all computations
- Register File
 - Stores variables



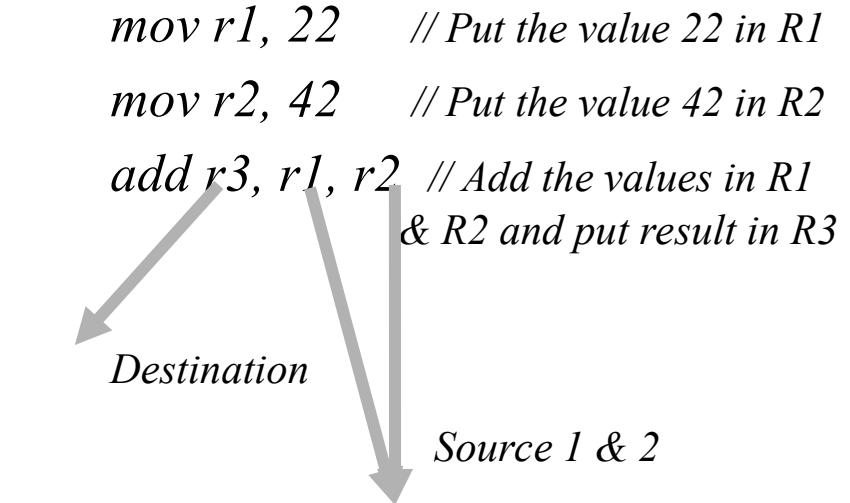
- The first step in any processor design would be to decide on an ISA
- ISA is the interface provided by the architect to the external world
 - The instructions supported with their opcode (The binary representation of instruction mnemonics)
 - The width (number of bits) of data, instruction, data address, instruction address
 - Other information necessary to the compiler like number of registers in the register file etc.

Assembly Code

- High Level Language
(Like C, C++, Java)
- Assembly language

```
void main ()  
{  
    int a = 22;  
    int b = 42;  
    int c = a + b;  
}
```

*This conversion is done by **compiler***

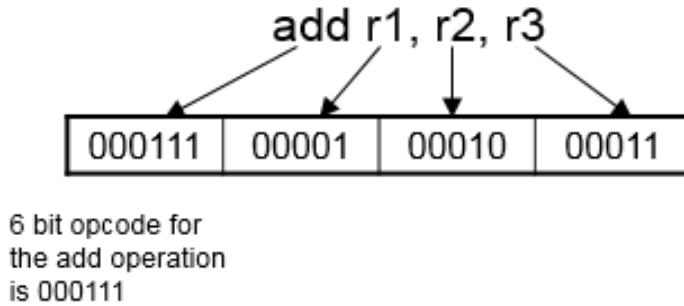


Types of Instructions

- ALU
 - add, sub, mult, or, and, xor
 - Operands may be Register-register, Register-memory, memory-memory
 - Immediate operands (will be discussed later)
- MEM
 - load, store
 - Direct addressed: load r1, 1234H
 - Register Addressed: load r1, (r2)
- Control
 - jmp, branch
 - Change value of PC to required location

Converting Instructions to binary codes

- Each instruction is encoded into a binary format and stored in the instruction memory.
- The control unit decodes it and gives appropriate signals to ALU



Assuming that the register file has 32 registers, each register has a 5 bit code, from r1 to r31,

$$r1 = 00001, r31 = 11111$$

Thus total length of instruction = $6 + 5*3 = 21$ bits

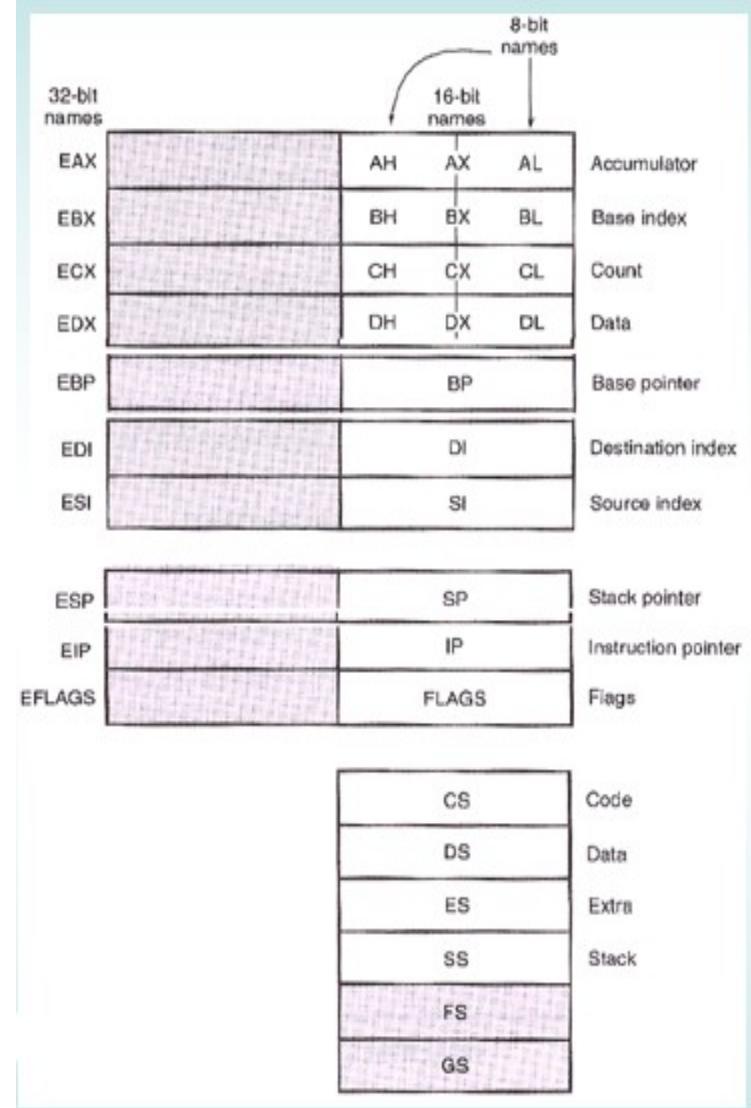
This is an example of fixed length encoding scheme.

The programming model

- The **programming model** of the 8086 through the Pentium II is considered to be **program visible**
- because its registers are used during **application programming** and are specified by the instructions.
- Other registers, detailed later in this chapter, are considered to be **program invisible** because they are not addressable directly during applications programming,
- but may be used **indirectly** during system programming. Only the 80286 and above contain the program-invisible registers used to control and operate the protected memory system

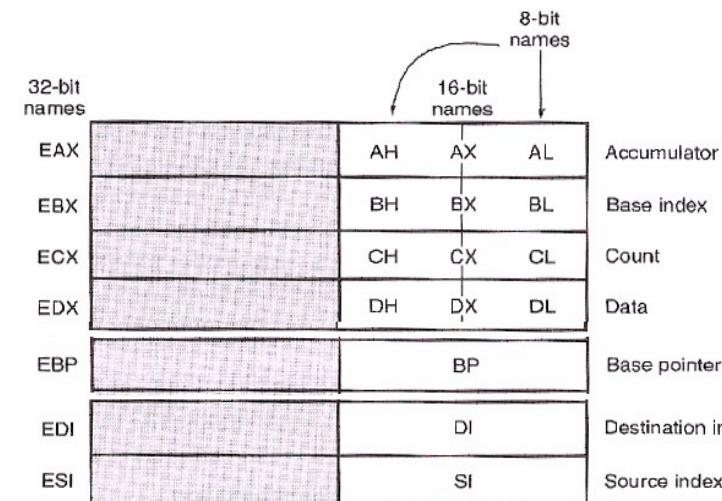
The programming model

- The shaded areas registers exist only on the **80386 through the Pentium II**.
- Figure 2-1 illustrates the **programming model** of the **8086** through the **Pentium II** microprocessor.
- The earlier **8086**, **8088**, and **80286** contain 16-bit(2 byte) internal architectures,
- Shown in Figure 2-1. The **80386**, **80486**, **Pentium**, **Pentium Pro**, and **Pentium II** microprocessors contain **full 32-bit internal architectures**.



General Purpose Register

- The **programming model** contains **8, 16, and 32 bit registers**.
- The **8-bit registers** are **AH, AL, BH, BL, CH, CL, DH, and DL** and are referred to when an instruction is formed using these two-letter designations.
- For example, an **ADD AL,AH** instruction adds the **8-bit contents of AH to AL**. (Only AL changes due to this instruction.)
- The **16-bit registers** are **AX, BX, CX, DX, SP,BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS, and GS**. These registers are also referenced with the **two-letter designations**.
For example, an **ADD DX,CX** instruction adds the 16-bit contents of CX to DX.



General Purpose Register

- The **extended 32-bit registers** are EAX, **EBX**, **ECX**, **EDX**, **ESP**, **EBP**, **EDI**, **ESI**, **EIP**, and **EFLAGS**.
- These registers are referenced as general purpose for the two new 16-bit registers, and by a three-letter designation for the 32-bit registers. For example, an ADD ECX,EBX instruction adds the 32-bit contents of EBX to ECX. (Only ECX changes due to this instruction.)

32-bit names	16-bit names			8-bit names
EAX	AH	AX	AL	Accumulator
EBX	BH	BX	BL	Base index
ECX	CH	CX	CL	Count
EDX	DH	DX	DL	Data
EBP			BP	Base pointer
EDI			DI	Destination
ESI			SI	Source index

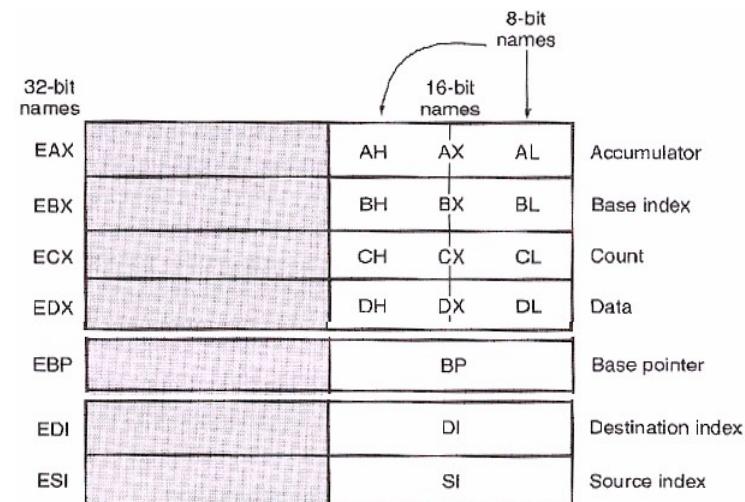
General Purpose Register

- Some registers are **general-purpose** or **multipurpose registers**, while some have **special purposes**.
- The **multipurpose registers**(General purpose) include **EAX**, **EBX**, **ECX**, **EDX**, **EBP**, **EDI**, and **ESI**.
- These **registers hold various data sizes** (bytes, words, or double words) and are **used** for almost **any purpose**, as dictated by a program.

32-bit names	16-bit names			
EAX	AH	AX	AL	Accumulator
EBX	BH	BX	BL	Base index
ECX	CH	CX	CL	Count
EDX	DH	DX	DL	Data
EBP			BP	Base pointer
EDI			DI	Destination
ESI			SI	Source index

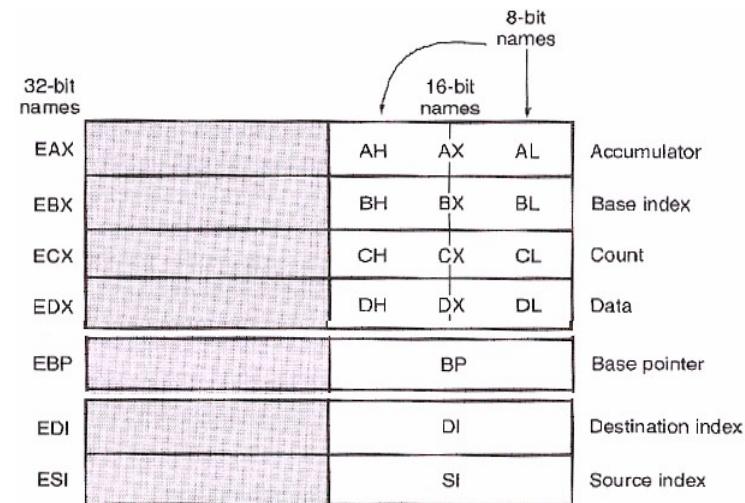
General Purpose Register

- **EAX(Accmulator)** is referenced as a **32-bit register (EAX)**, as a 16-bit register (AX), or as either of two 8-bit registers (AH and AL). Note that if an 8- or 16-bit register is addressed, only that portion of the 32-bit register changes without affecting the remaining bits
- The accumulator is used for instructions such as multiplication, division, and some of the adjustment instructions. For these instructions, the accumulator has a special purpose,



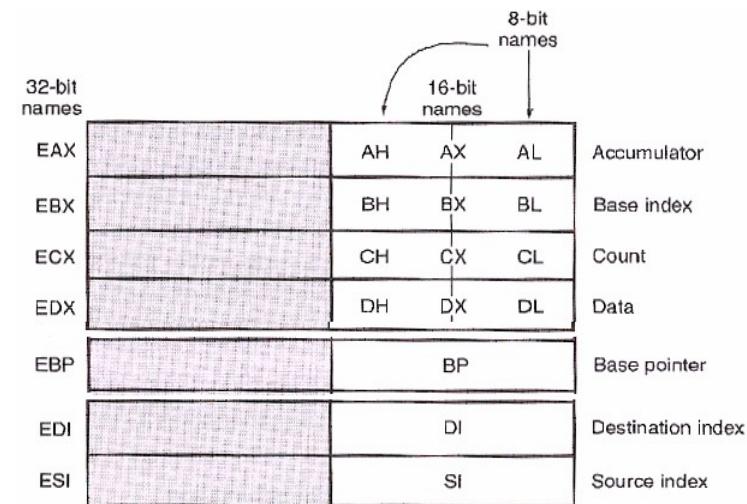
General Purpose Register

- It is generally considered to be a multipurpose register.
- In the 80386 and above, the EAX register may also hold the **offset address** of a location in the memory system.



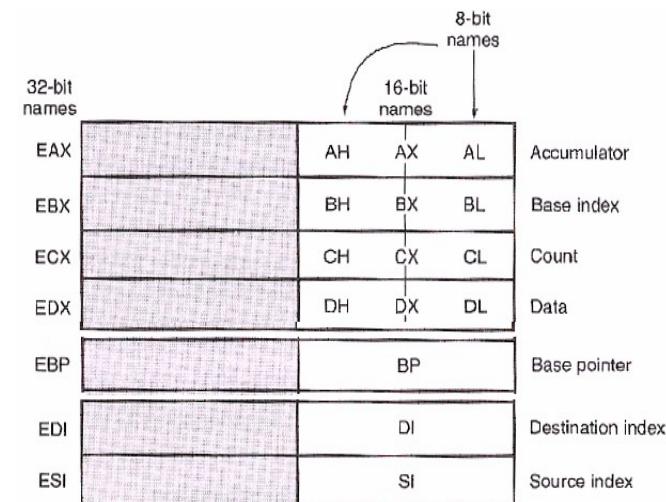
General Purpose Register

- but is generally considered to be a **multipurpose register**.
- In the 80386 and above, the EAX register may also hold the **offset address** of a location in the memory system.



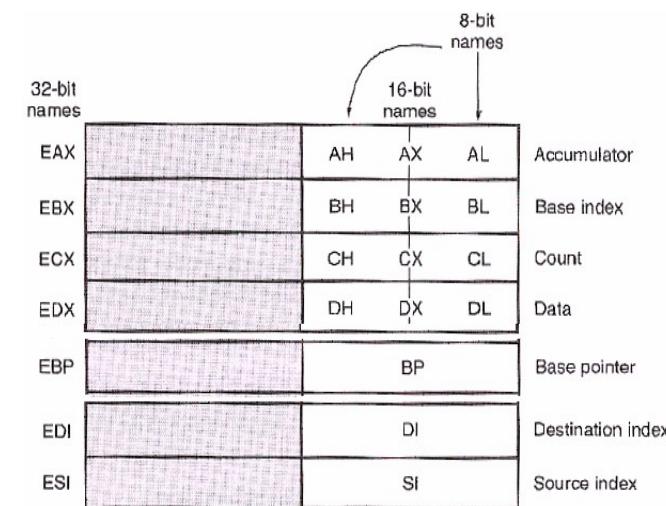
General Purpose Register

- **EBX(Base Index)** is addressable as **EBX**, **BX**, **BH**, or **BL**.
- The **BX register sometimes holds the offset address** of a location in the memory system in all versions of the microprocessor.
- In the 80386 and above, EBX also can address memory data.
- **ECX(Count)** is a general-purpose register that also holds the **count** for **various instructions**.
- In the 80386 and above, the ECX register also can hold the offset address of memory data.



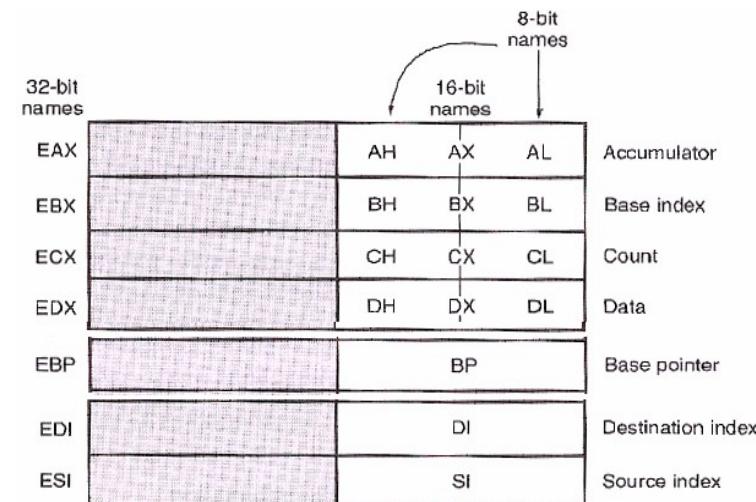
General Purpose Register

- Instructions that **use a count** are the **repeated string instructions** (REP/REPE/REPNE); and shift, rotate, and LOOP/LOOPD instructions.
- The **shift** and **rotate instructions** use **CL** as the **count**, the repeated string instructions use **CX**, and the LOOP/LOOPD instructions use either **CX** or **ECX**.
- EDX (Data)** is a general-purpose register that **holds** a part of the **result** from a **multiplication** or part of the dividend before a division.
- In the 80386 and above, this register can also address memory data



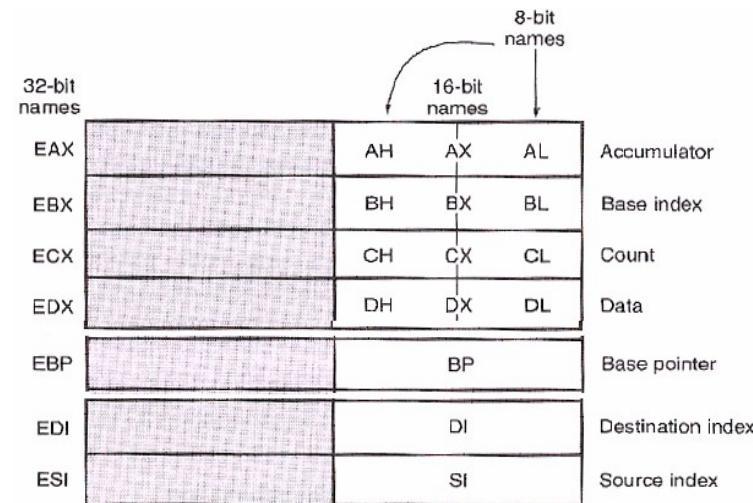
General Purpose Register

- **EBP(Base pointer)** points to a **memory location** in all versions of the **microprocessor** for **memory data transfers**.
- This register is addressed as either BP or EBP.
- **EDI(Destination Index)** often **addresses string destination** data for the **string instructions**.
- It also functions as either a 32-bit (EDI) or 16-bit (DI) general-purpose register.



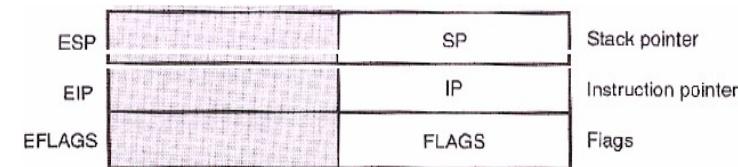
General Purpose Register

- **ESI (source index)** is used as either ESI or SI.
- The source index register often **addresses source string data** for the string instructions.



Special-purpose Registers

- special-purpose registers include EIP, ESP,EFLAGS and the segment registers CS(Code Segment), DS, ES, SS, FS & GS
- **EIP(Instruction Pointer)** addresses the **next instruction** in a section of memory defined as a **code segment**.
- The **instruction pointer**, which points to the **next instruction** in a program, is used by the microprocessor to find the **next sequential instruction** in a program located within the code segment.
- The instruction pointer can be modified with a jump or a call instruction.



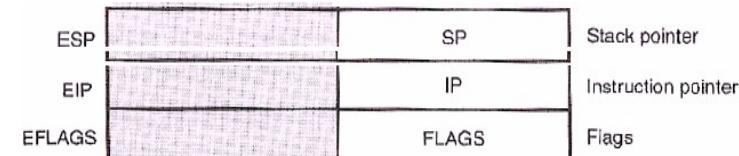
Special-purpose Registers

- **ESP (stack pointer)** addresses an area of memory called the stack.
- The **stack memory stores data** through this pointer and is explained later in the text with instructions that **address stack data**. This register is referred to as SP if used as a 16-bit register and ESP if referred to as a 32-bit register.

ESP	SP	Stack pointer
EIP	IP	Instruction pointer
EFLAGS	FLAGS	Flags

Special-purpose Registers

- **EFLAGS** indicate the **condition** of the microprocessor and **control** its operation.
 - Figure 2-2 shows the flag registers of all versions of the microprocessor.
 - Note that the flags are upward-compatible from the 8086/8088 to the Pentium II microprocessor.
 - The 8086-80286 contain a FLAG register (16 bits) and the 80386 and above contain an EFLAG register (32-bit extended flag register)



– Figure 2.2

Type of Flags

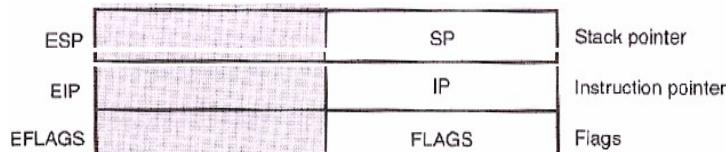
R	R	R	R	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
---	---	---	---	----	----	----	----	----	----	---	----	---	----	---	----

R= reserved
U= undefined
OF= overflow flag
DF= direction flag
IF= interrupt flag
TF= trap flag

SF= sign flag
ZF= zero flag
AF= auxiliary carry flag
PF= parity flag
CF= carry flag

Special-purpose Registers

- The flags never change for any **data transfer** or **program control operation**.
 - It **Changed only** by **arithmetic** and **logic operation**



Special-purpose Registers

- A(**auxiliary carry**) holds the **carry** (half-carry) after **addition** or **the borrow** after subtraction between bits positions 3 and 4 of the result.
- If there is a **carry from d3 to d4** of an **operation** this bit is **set to 1**, otherwise cleared (set to 0).



X = RESERVED

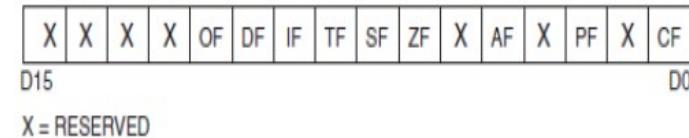
Special-purpose Registers

- **C(Carry)** holds the **carry after addition** or the **borrow after subtraction**.
- This flag is set whenever there is a carry out, **either from d7 after an 8-bit operation**, or **from d15 after a 16-bit data operation**.
- **P(Parity)** is a logic **0** for **odd parity** and a logic **1** for **even parity**.
- **Parity** is a **count of ones** in a **number** expressed as **even** or **odd**.
- After certain operations, the parity of the result's low-order byte is checked. If the **byte** has an **even number of 1s**, the **parity flag is set to 1**; otherwise, it is cleared(set zero)



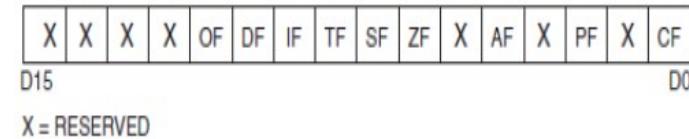
Special-purpose Registers

- **Z (zero)** The **zero flag** shows that the **result** of an **arithmetic** or **logic operation** is **zero**. If **Z = 1**, the **result** is **zero**; if **Z = 0**, the **result** is **not zero**.
- **S (sign)** The **sign flag holds the arithmetic sign** of the **result** after an **arithmetic** or **logic instruction executes**.
- **MSB** is used as the **sign bit of the binary representation** of the **signed numbers**.
- If **S = 1**, the sign bit (leftmost bit of a number) is set or negative; if **S = 0**, the sign bit is cleared or positive.



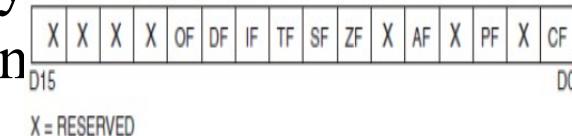
Special-purpose Registers

- **T (trap)** The trap flag enables trapping through an on-chip debugging feature. (A program is debugged to find an error or bug.)
 - If the T flag is **enabled (1)**, the **microprocessor interrupts** the **flow** of the program on conditions as indicated by the debug registers and control registers.
 - If the **T flag** is a **logic 0**, the **trapping (debugging) feature** is **disabled**. The CodeView program can use the trap feature and debug registers to debug faulty
 - When this flag is set it allows the program to single step, meaning to execute one instruction at a time. Used for debugging purposes.



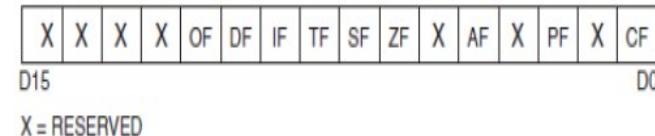
Special-purpose Registers

- **I (interrupt)** The **interrupt flag controls** the **operation** of the INTR (interrupt request) input pin.
- This bit is set or cleared to enable or disable only the external interrupt requests
- **If I = 1**, the INTR pin is enabled; if I = 0, the INTR pin is disabled. The state of the I flag bit is controlled by the STI (set I flag) and CLI (clear I flag) instruction
- **D (direction)** The **direction flag** selects either the **increment** or **decrement mode** for the DI and/or SI registers during string instructions.
- **If D = 1**, the registers are automatically **decremented**; **if D = 0**, the registers are automatically **incremented**.



Special-purpose Registers

- **O (overflow)** Overflows occurs when **signed numbers** are **added** or **subtracted**.
- This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
- An **overflow indicates** that the **result** has **exceeded** the capacity of the machine.
- For example, if a **7FH (+127)** is added—using an 8-bit addition—to a **01H (+1)**, the result is **80H (-128)**. This result represents an overflow condition indicated by the overflow flag for **signed addition**. For unsigned operations, the overflow flag is ignored.
- **Sign Bit has Changed set 1 otherwise set 0**



Example

- Example, if the Register AL=7FH and the Instruction ADD AL,1 execute. What will be the Value if the flag Following is happen

- The Following will be happen in the Flag

$AL = 80H$

$CF = 0$; there is **no carry**

$PF = 0$; $80H$ has an **odd number of ones**

$AF = 1$; there is a carry out of bit 3 into bit 4

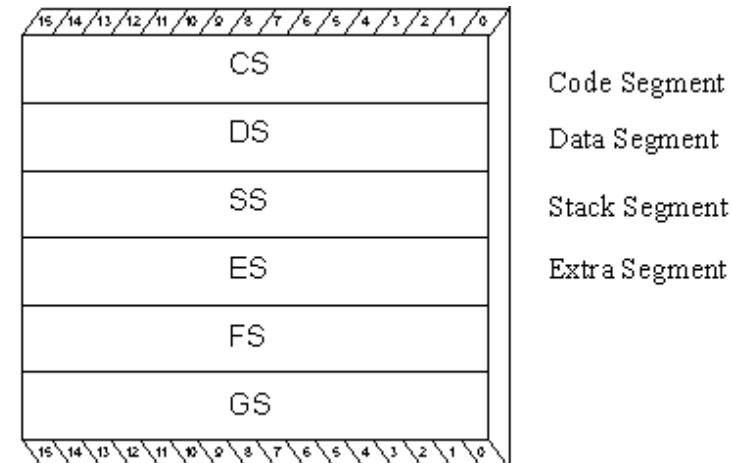
$ZF = 0$; the result is **not zero**

$SF = 1$; bit **seven is one**

$OF = 1$;The **Sign Bit has Changed**

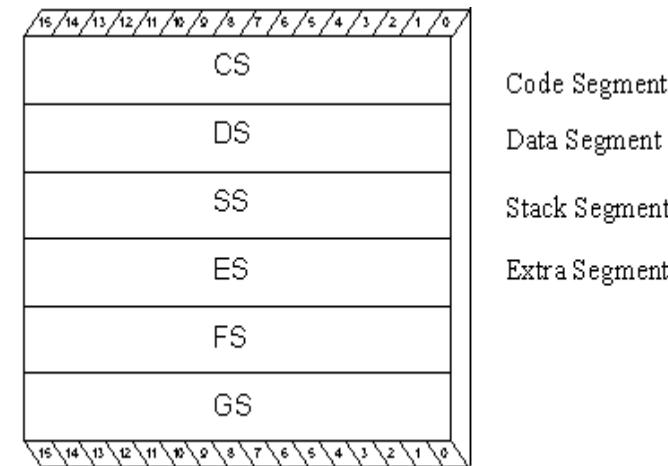
Segment Register

- *Segment Registers.* Additional registers, called segment registers, **generate memory addresses** when combined with other **registers** in the microprocessor.
- There are either four or six segment registers in various versions of the microprocessor.
- A segment register functions differently in the real mode when compared to the protected mode operation of the microprocessor.



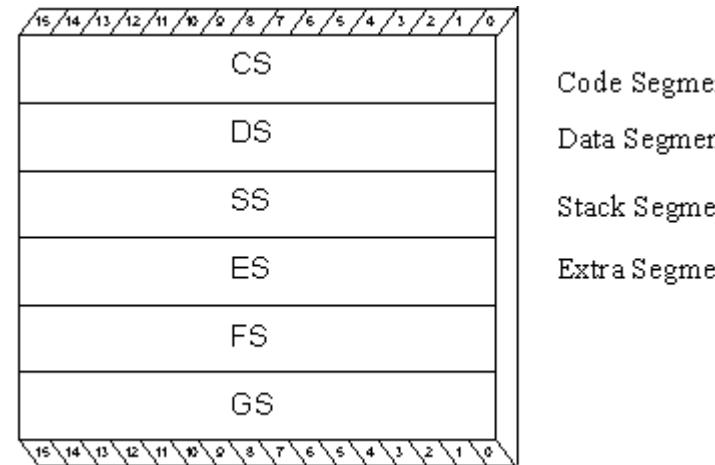
Segment Register

- The **code segment(CS)** is a section of **memory** that holds the **code (programs and procedures)** used by the microprocessor.
- The **code segment register** defines the **starting address** of the section of memory holding code.
- The **Data Segment(DS)** is a section of memory that **contains most data used by a program**.
- Data are accessed in the **data segment** by an **offset address** or the contents of other registers that hold the **offset address**. As with the code segment and other segments.



Segment Register

- The **extra segment (ES)** is an additional data segment that is used by some of the string instructions to **hold destination data**.
- The **stack segment (SS)** defines the area of **memory used for the stack**.
- The stack entry point is determined by the **stack segment and stack pointer registers**. The BP register also addresses data within the stack segment.



Real mode memory Addressing

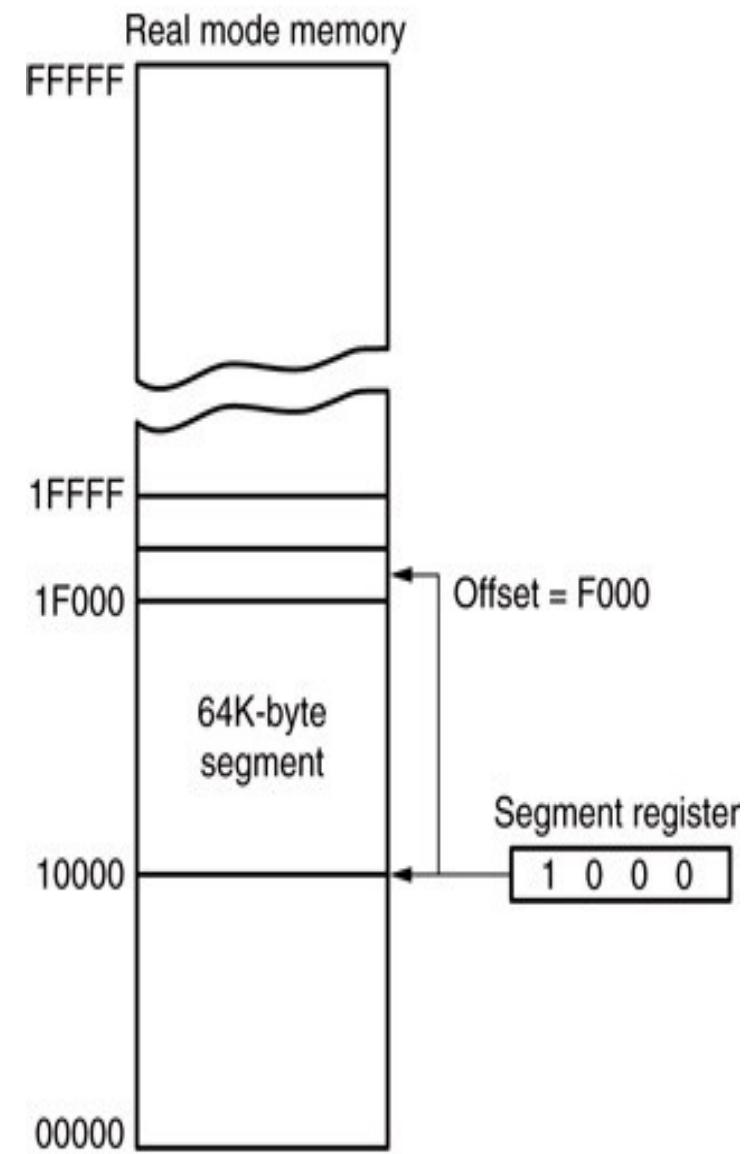
- The **80286** and **above operate** in either the **real** or **protected mode**.
- Only the **8086** and **8088** **operate exclusively** in the **real mode**.
- **Real mode operation** allows the microprocessor to **address** only the **first 1M byte** of **memory space** even if it is the Pentium II microprocessor.
- The first 1M byte of memory is called either the **real memory**, **conventional memory** or **Dos Memory System**.
- The **DOS operating system** requires the microprocessor to operate in the **Real mode**.

Segments And Offsets

- A combination of a **segment address** and an **offset address** access a **memory location** in the **real mode**.
- **All real mode memory addresses** must **consist** of a **segment address**(CS,DS,ES), plus (IS) an offset address.
- The **segment address**, located within **one** of the **segment registers**, **defines the beginning address** of any 64K-byte(each Segment Address has 16 bit $2^{16} = 2^6 * 2^{10} = 64K$ byte) memory segment.
- The **offset address** selects any **location** within the 64K byte memory segment.
- **Segments** in the **real mode** always have a **length** of **64K bytes**.

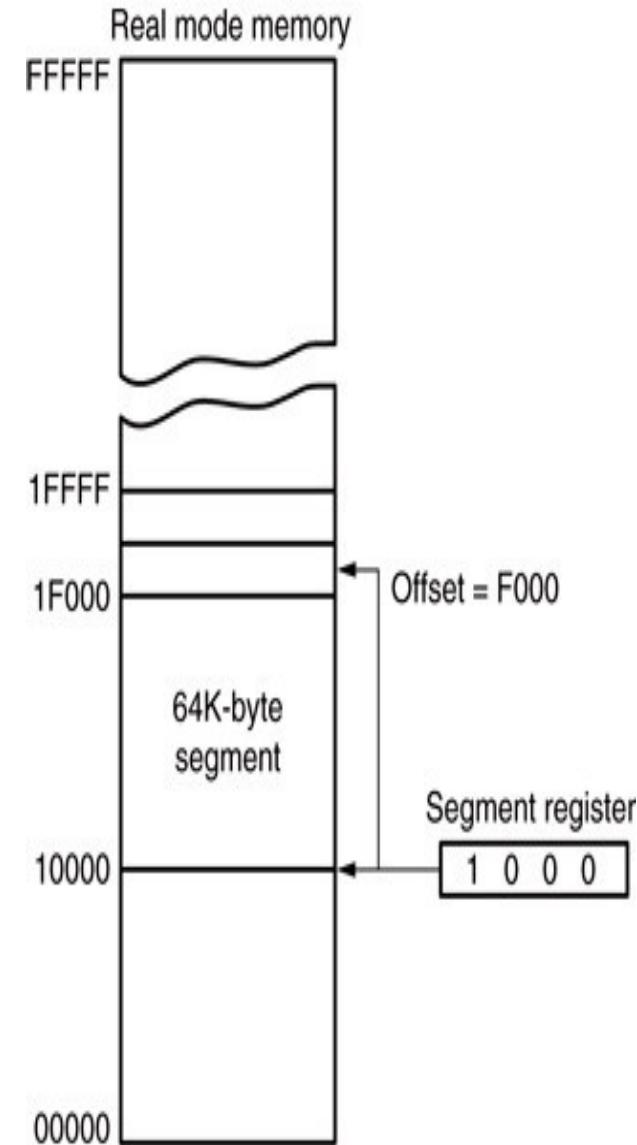
Segment and Offset

- Figure 2-3 shows how the segment plus offset addressing scheme selects a memory location.
- This illustration shows a **memory segment** that begins at **location 10000H** and ends at **location 1FFFFH**— 64K bytes in length.
- It also shows how an **offset address**, sometimes called a **displacement**, of FOOOH selects location 1FOOOH in the memory system.
- Note that the offset or displacement is the distance above the start of the segment, as shown in Figure 2-3.



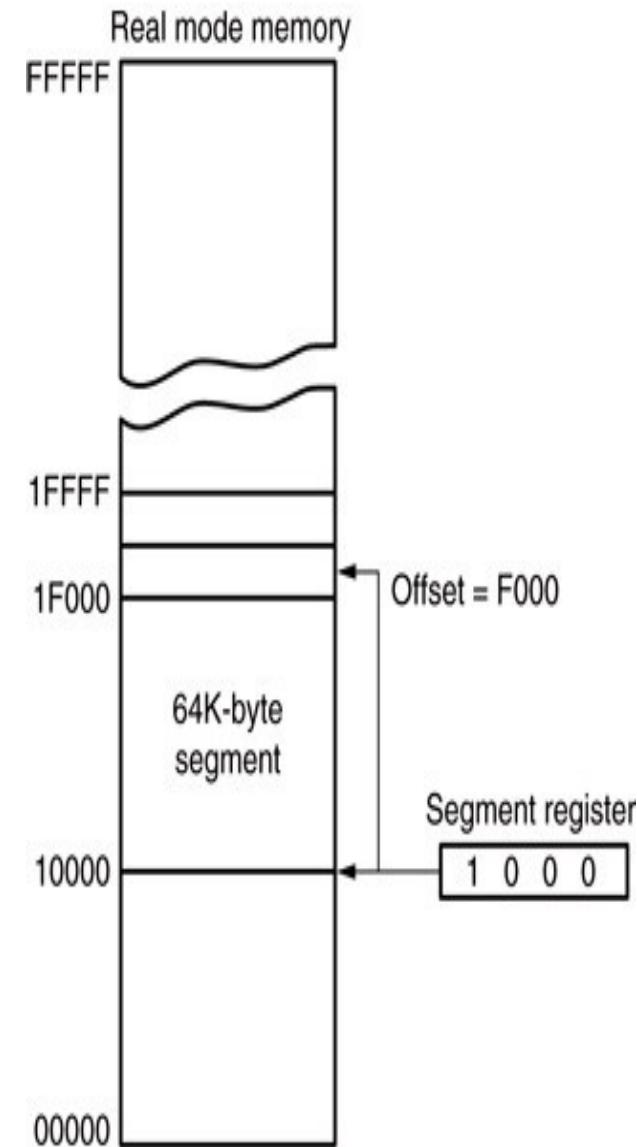
Segment and Offset

- The segment register in Figure 2-3 contains a 1000H, yet it addresses a starting segment at location 10000H.
- In the **real mode**, each segment register is internally appended with a **OH** on its rightmost end (Why ?). This forms a 20-bit memory address, allowing it to access the **start of a segment**.



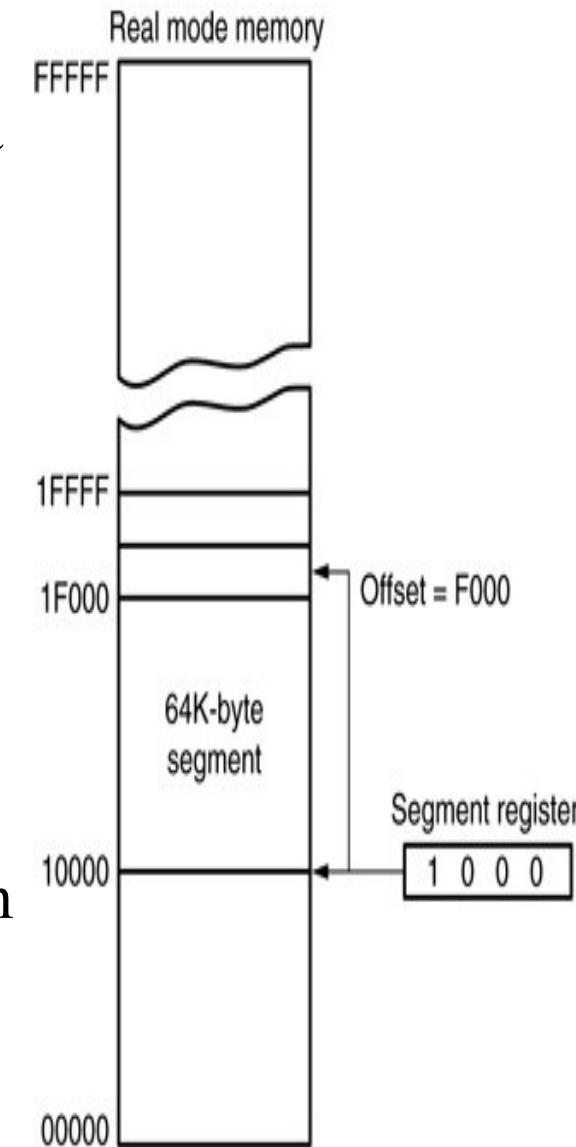
Segment and Offset

- The **microprocessor must generate a 20-bit memory address** to access a **location** within the **first 1M of memory**.
- Simply $EA = \text{segment register (SR)} \times 10H + \text{offset}$
- $EA = \text{Effective Address or Physical address}$
- $EA = \text{segment register (SR)} \times 10H + \text{offset}$



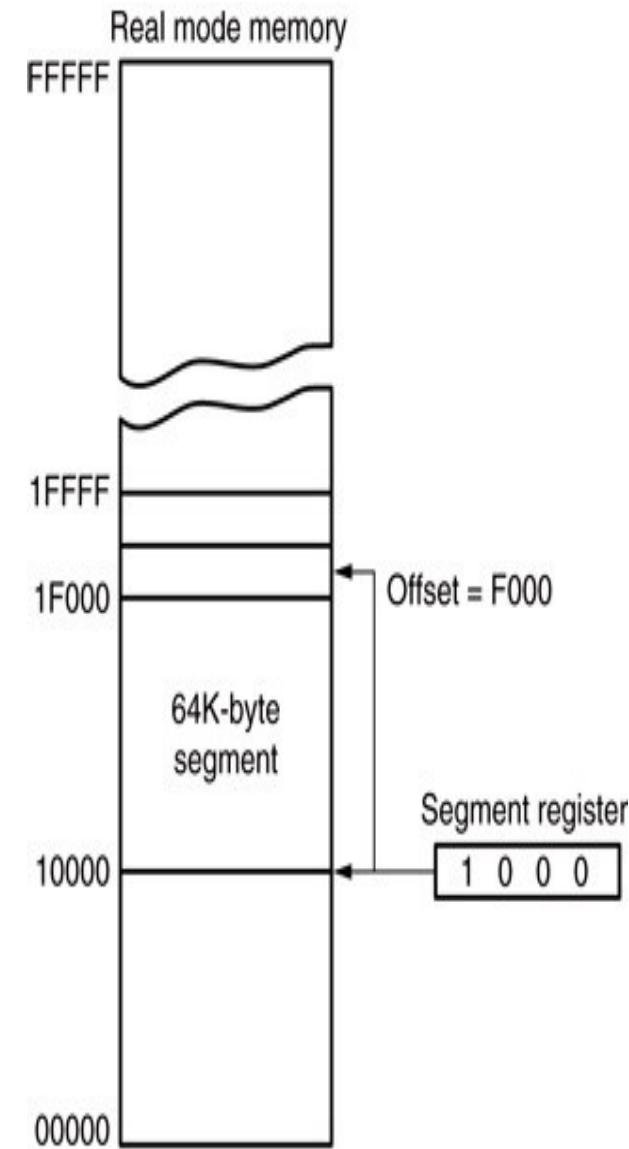
Segment and Offset

- For example, when a **segment register contains a **1200H****, it addresses a 64K-byte memory segment beginning at location **12000H**. Likewise, if a segment register contains a **1201 H**, it addresses a memory segment beginning at location **1201 OH**. Because of the internally appended OH(*10H),
- **real mode segments** can begin only at a **16-byte** boundary in the memory system.
- Once the beginning address is known, the **ending address** is found by adding FFFFH. Why
- because a **real mode** segment of memory is **64K** in length



Segment and Offset

- For example, if a **segment register** contains **3000H**, the **first address** of the segment is **30000H(3000H*10H)**, and the **last address** is **30000H + FFFFH** or **3FFFFH**.
- Segment and offset address is sometimes written as 1000:2000.
- a segment address of 1000H; an offset of 2000H



Example

- Find the starting And ending segment Address of the following Segment Register

2000H,2001H,2100H,ABOOH & 1234H

Segment and Offset

Segment Address	Starting Address (*10H)	Ending Address(Add FFFF)
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH
ABOOH	AB000H	BAFFFH
1234H	123450H	2233FH

Example

- If the segment register CS,DS and SS have values 1000H, 2000H and 3000H respectively . What will be the 20bit starting and ending address of the code, data and stack segment ?

Example

Code Segment:	20-bit start address	= CS x10h +0000H
	20-bit end address	= 10000h
		= CS x10H +FFFFH
		= 1FFFFH
Data Segment :	20-bit start address	= DS x10h +0000H
	20-bit end address	= 20000H
		= DS x10H +FFFFH
		= 2FFFFH
Stack Segment :	20-bit start address	= SS x10h +0000H
	20-bit end address	= 30000H
		= SS x10H +FFFFH
		= 3FFFFH

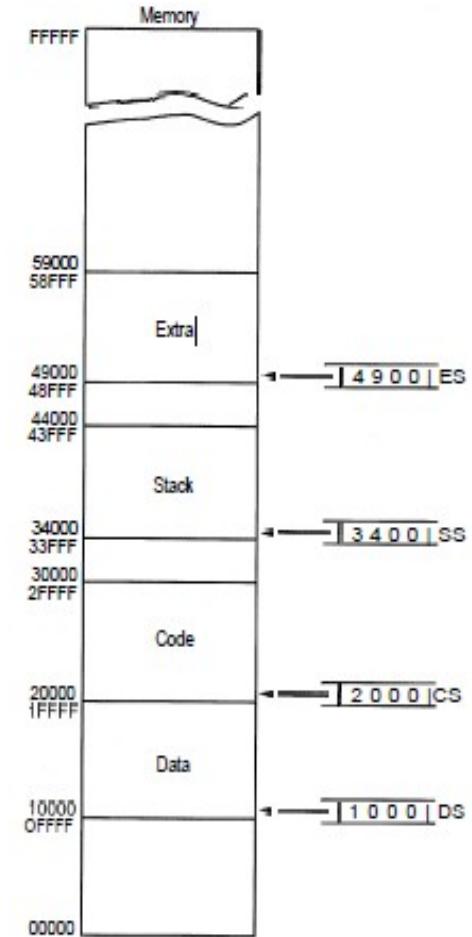
Default Segment and Offset Registers

- The microprocessor has a set of rules that apply to segments whenever memory is addressed.
- These rules, which apply in the real and protected mode, define the segment register and offset register combination.
- For example, the code segment register is always used with the instruction pointer to address the next instruction in a program.
- This combination is **CS:IP** or **CS:EIP**, depending upon the microprocessor's mode of operation.
- The **code segment** register defines the start of the code segment and the **instruction pointer** locates the next instruction within the code segment

Default Segment and Offset Registers

- For example, if CS = 1400H and IP/EIP = 1200H, the microprocessor fetches its next instruction from memory location 14000H + 1200H or 15200H
- Another of the default combinations is the **stack**. Stack data are referenced through the stack segment at the memory location addressed by either the stack pointer (SP/ESP) or the base pointer (BP/EBP).
- These combinations are referred to as SS:SP (SS:ESP) or SS:BP (SS:EBP).
- For example, if SS = 2000H and BP = 3000H, the microprocessor addresses memory location 23000H for the stack segment memory location..

- The 8086-80286 microprocessors allow four memory segments and the 80386 and above allow six memory segments.
- Figure 2-4 shows a system that contains four memory segments.
- Note that a memory segment can touch or even overlap if 64K bytes of memory are not required for a segment.
- Think of segments as windows that can be moved over any area of memory to access data or code. Also note that a program can have more than four or six segments, but can only access four or six segments at a time.



Protected mode of memory Accessing

- Allows access to data and programs located within & **above** the first 1M byte of memory.
- **Protected mode** is where Windows operates.
- In place of a segment address, the segment register contains a **selector** that selects a **descriptor** from a descriptor table.
- The **descriptor** describes the memory **segment's location, length,** and **access rights**

Selectors and Descriptors

- The **descriptor** is located in the segment register & describes the location, length, and access rights of the segment of memory.
 - it selects one of **8192** descriptors from one of two tables of descriptors
- In protected mode, this segment number can address any memory location in the system for the code segment.
- Indirectly, the register still selects a memory segment, but not directly as in real mode.

- **Global descriptors** contain segment definitions that apply to all programs.
- **Local descriptors** are usually unique to an application.
 - a global descriptor might be called a **system descriptor**, and local descriptor an **application descriptor**

The Figure 2–6 shows the format of a descriptor for the 80286 through the Core2.

- each descriptor is **8 bytes** in length
- global and local descriptor **tables** are a maximum of **64K bytes** in length

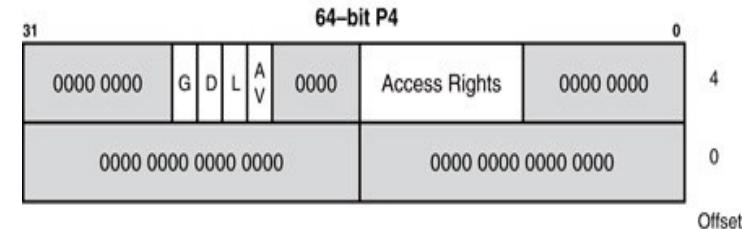
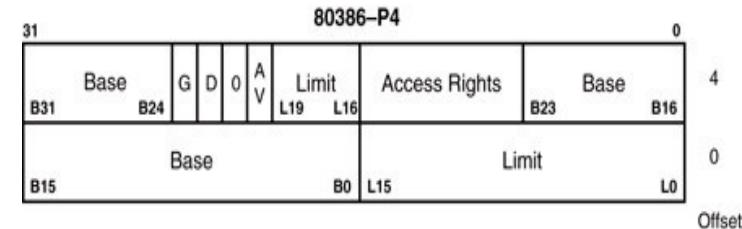
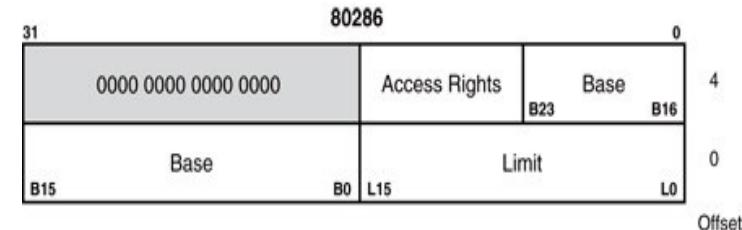
The **base address** of the descriptor indicates the starting location of the memory segment.

- the **paragraph boundary** limitation is removed in protected mode
- segments may begin at **any address**

The G, or **granularity bit** allows a segment length of 4K to 4G bytes in steps of 4K bytes.

- 32-bit offset address** allows segment lengths of **4G bytes**

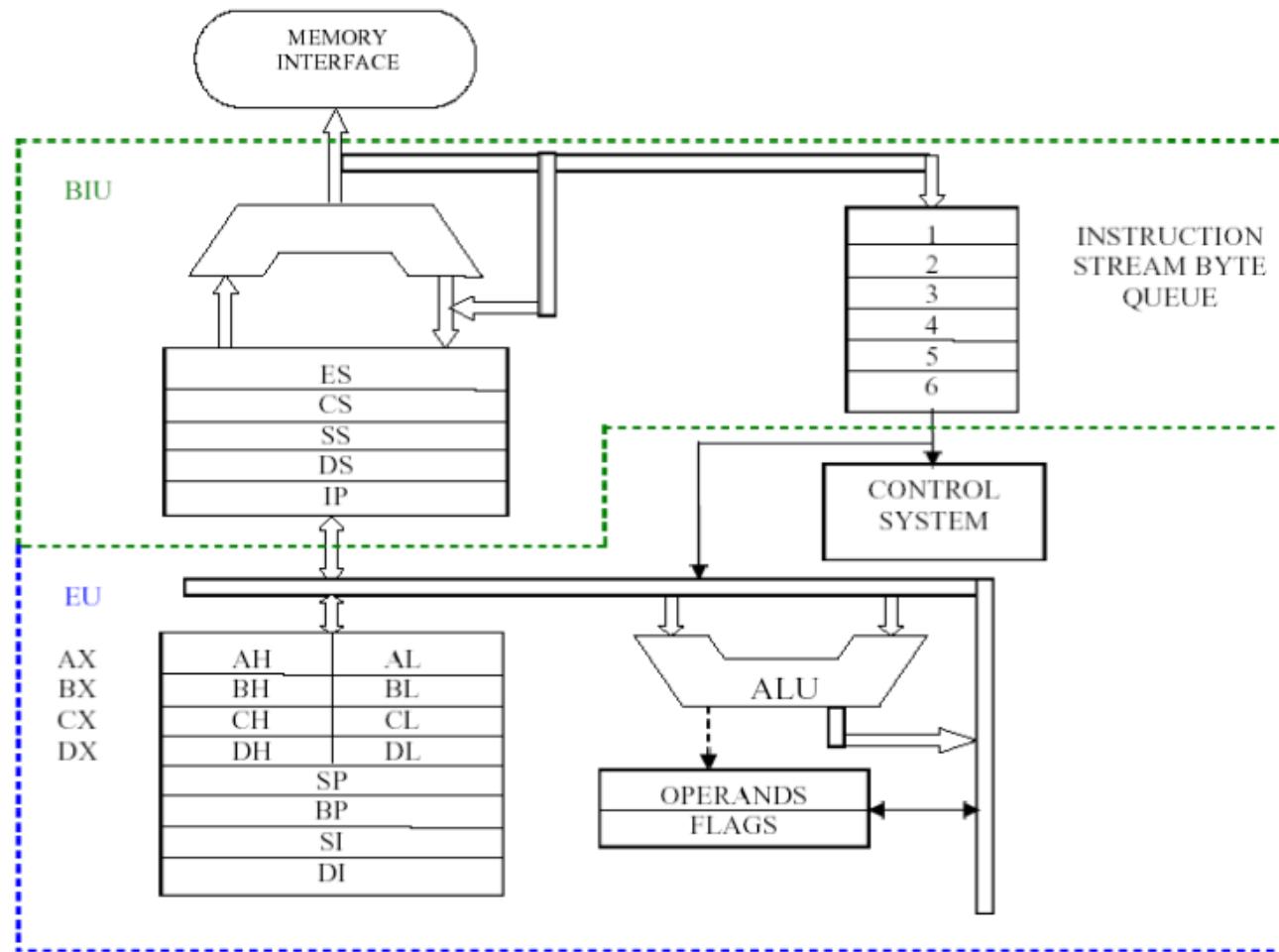
- 16-bit offset address** allows segment lengths of **64K bytes**



- Descriptors are chosen from the **descriptor table** by the segment register.
 - register contains a 13-bit selector field, a table selector bit, and requested privilege level field
- The **TI bit** selects either the global or the local descriptor table.
- **Requested Privilege Level** (RPL) requests the access privilege level of a memory segment.
 - If privilege levels are violated, system normally indicates an application or **privilege level violation**

Execution And Bus Interface

Block Diagram of 8086 Microprocessor



EU and BIU

- 8086 microprocessor the first 16-bit microprocessor what it means ?
- It means it can be transfer 16 bit data at a time(per cycle)
- Intel 8086 contains two independent functional units:
 - ✓ Bus Interface Unit (BIU)
 - ✓ Execution Unit (EU)

EU and BIU

- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands.
- The instruction bytes are transferred to the instruction queue.
- EU executes instructions from the instruction system byte queue.

EU and BIU

- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as **Pipelining**.
- **Pipeline** means Fetching the next instruction while the current instruction executes.
- BIU contains Instruction queue, Segment registers , Instruction pointer, Address adder.
- EU contains Control circuit , Instruction decoder, ALU, Pointer and Index register, Flag register

EU and BIU

- Bus Interface Unit:
- It provides a full 16 bit bidirectional data bus and 20 bit address bus.
- The bus interface unit is responsible for performing all external bus operations.
- This queue permits pre fetch of up to six bytes of instruction code. When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.

EU and BIU

- These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
- After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.
- The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
- These intervals of no bus activity, which may occur between bus cycles are known as ***Idle state***.

EU and BIU

- If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
- The BIU also contains a dedicated adder which is used to generate the 20 bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.

EU and BIU

- For example, the physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

EU and BIU

- **EXECUTION UNIT :** The Execution unit is responsible for decoding and executing all instructions.
- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.

EU and BIU

- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

Summary

- The programming model contains 8, 16, and 32 bit registers.
- The 8-bit registers are AH, AL, BH, BL, CH, CL, DH, and DL
- The 16-bit registers are AX, BX, CX, DX, SP, BP, DI, SI, IP, FLAGS, CS, DS, ES, SS, FS, and GS.
- The extended 32-bit registers are EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP, and EFLAGS.
- AX/EAX register is also known as Accumulator register that stores operands for arithmetic operation like divided, rotate.
- **BX/EBX (Base index).** This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.
- CX/ECX It is defined as a counter. It is primarily used in loop instruction to store loop counter.
- **DX Register:** DX register is used to contain I/O port address for I/O instruction.

Summary

- DX/EDX – Data register used to contain I/O port address for I/O instruction.
- BP/EBP – registers sometimes hold the offset address of a location in the memory system
- DI/EDI – Destination index, used by string instructions.
- SI/ESI – Source index used by string instructions
- IP/EIP – Instruction pointer which holds the address of the next instruction to be executed.
- SP/ESP – Stack pointer addresses the stack.
- EFLAGS controls certain operations and indicates the status of the 80836 (carry, sign, etc).

Summary

- **Carry flag**: This flag indicates an **overflow condition** for arithmetic operations
- **Auxiliary flag**: When an operation is performed at ALU, it results in a **carry/barrow**
- **Parity flag**: This flag is used to indicate the **parity of the result**, i.e. when the lower order 8-bits of the result contains **even number of 1's**, then the Parity **Flag is set**.
For **odd number of 1's**, the **Parity Flag is reset**.
- **Zero flag**: This flag is **set to 1** when the result of **ALU Operation** is **zero** else it is **set to 0**.
- **Sign flag**: This flag holds the sign of the result, i.e. when the **result of the operation is negative**, then the sign flag **is set to 1** else **set to 0**.
- **Overflow flag**: This flag represents the result when the system capacity is exceeded.
- **Control flags** controls the operations of the **execution unit**
- **Trap flag**: It is used for **single step control** and **allows** the user to **execute one instruction at a time for debugging**.
- Interrupt Flag :-
- Direction Flag :-

CF, the Carry Flag: This flag is set whenever there is a carry out, either from d7 after an 8-bit operation, or from d15 after a 16-bit data operation.

PF, the Parity Flag: After certain operations, the parity of the result's low-order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise, it is cleared.

AF, the Auxiliary Carry Flag: If there is a carry from d3 to d4 of an operation this bit is set to 1, otherwise cleared (set to 0).

ZF, the Zero Flag: The ZF is set to 1 if the result of the arithmetic or logical operation is zero, otherwise, it is cleared (set to 0).

SF, the Sign Flag: MSB is used as the sign bit of the binary representation of the signed numbers. After arithmetic or logical operations the MSB is copied into SF to indicate the sign of the result.

TF, the Trap Flag: When this flag is set it allows the program to single step, meaning to execute one instruction at a time. Used for debugging purposes.

IF, Interrupt Enable Flag: This bit is set or cleared to enable or disable only the external interrupt requests.

DF, the Direction Flag: This bit is used to control the direction of the string operations.

OF, the Overflow Flag: This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.

Summary

- **Real mode operation** allows the microprocessor to address only the first 1M byte of memory space—even if it is the Pentium II microprocessor.
- The **segment address**, located within one of the segment registers, defines the beginning address of any 64K-byte(each Segment Address has 16 bit $2^{16} = 2^6 * 2^{10} = 64K$ byte) memory segment.
- The **offset address** selects any location within the 64K byte memory segment.
- Allows access to data and programs located within & **above** the first 1M byte of memory.
- **Protected mode** is where Windows operates
- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands.
- The instruction bytes are transferred to the instruction queue.
- EU executes instructions
-

»End



ADMAS UNIVERSITY

Department Computer Science

Computer ARCTUCTURE AND Organization

Assignment 1 submission date May 18 prep by abdu.j

1. Briefly explain microprocessor Age
2. Compare and Contrast 8085 and 8086 Microprocessor with Diagram
3. Explain four control bus connections with diagram
4. Explain the following three main parts of main memory
 - a. ***Transient Program Area (TPA)*** – 640 Kbytes.
 - b. ***System Area*** – 384 Kbytes.
 - c. ***Extended Memory system (XMS)***
5. Compare and contrast general purpose register with special purpose register
6. If the segment register CS, DS and SS have values 1000H, 2000H and 3000H respectively. What will be the 20bit starting and ending address of the code, data and stack segment?
7. If the Register AL=7FH and the Instruction ADD AL, 1 execute. What will be the Value
CF, PF, AF, ZF, SF, OF

Computer Organization and Architecture

Chapter

1

Chapter 1: Introduction to the Microprocessors and Computers

1

Key word

- μ P= Microprocessor
- Clock rate:- the speed which μ P executes Instruction
- Clock Speeds are expressed in Megahertz (MHZ) or Gigahertz(GHZ)
- RISC (reduced instruction set computer)
- CISC(Complex Instruction set computer)
- Instruction Set: It is the set of instructions that the microprocessor can understand.
- Bandwidth: It is the number of bits processed in a single instruction

Basic terminologies

- **Instruction Set:** It is the set of instructions that the microprocessor can understand.
- **Bandwidth:** It is the number of bits processed in a single instruction.
- **Clock Speed:** It determines the number of operations per second the processor can perform. It is expressed in megahertz (MHz) or gigahertz (GHz).It is also known as Clock Rate.
- **Word Length:** It depends upon the width of internal data bus, registers, ALU, etc. An 8-bit microprocessor can process 8-bit data at a time. The word length ranges from 4 bits to 64 bits depending upon the type of the microcomputer.
- **Data Types:** The microprocessor has multiple data type formats like binary, BCD, ASCII, signed and unsigned numbers.

Content on this personation

1. A historical background
2. The microprocessor – based personal computer
3. The Intel and Motorola microprocessors
4. Review of the numbering system
5. Review of computer data representation

Content on this personation

- Historically we divide the computer in four Ages
 - The Mechanical Age
 - The Electrical Age
 - The Microprocessor Age
 - The Intel Age – The Modern Microprocessor(1970,1980,1990,2000)

The Mechanical Age

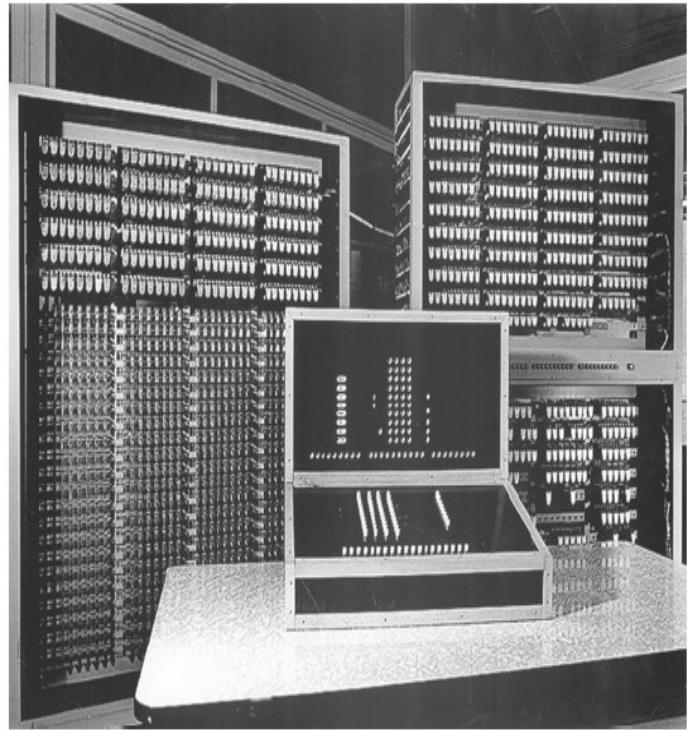
- In these Age the Dream was calculating with a Machine
- Idea of computing system not new.
- Babylonians / Chinese invented the **abacus**.
 - The first Mechanical calculator
 - strings of beads perform calculations
- In 1642 mathematician Blaise Pascal invented a calculator constructed of gears and wheels.
- Analytical Engine – Charles Babbage, 1823

The Electrical Age

- 1800s saw start of the electric motor.
- Also a mass of electrically motor-driven adding machines based on the Pascal mechanical calculator.
- Monroe also pioneer of electronic calculators, making desktop models.
- In 1889, **Herman Hollerith** developed the **punched card** for storing data.
- In 1896 Hollerith formed Tabulating Machine Company.
- After a number of mergers, Tabulating Machine Co. was formed into International Business Machines Corporation(IBM)
- Mechanical-electric machines dominated information processing world until 1941.

Z3 Electromechanical

- ✓ German inventor **Konrad Zuse** (1941) , invented the first **modern electromechanical computer (Z3)** .
- ✓ used in aircraft and missile design during World War II
- ✓ Zuse been given adequate funding, likely would have developed a much more powerful computer system.
- ✓ Zuse today receiving belated honors for pioneering work in the area



- **British Inventor Alan** (1943) Develop **First electronic computer** placed in operation to break secret German military codes (Colossus).
 - ✓ Used Vacuum tube
 - ✓ not-programmable (fixed-program)special-purpose computer
- ENIAC(Electronic Numerical Integrator and Calculator)
Pennsylvania, 1946
 - ✓ **first modern (general-purpose, programmable) electronic computer**
 - ✓ performing about 100,000 operations per second
 - ✓ using over 17,000 vacuum tubes and over 500 miles of wires
 - ✓ weighed over 30 tons
- programmed by rewiring its circuits
 - ✓ process took many workers several days
 - ✓ workers changed electrical connections on plug boards like
 - ✓ early telephone switchboards
- Required frequent maintenance
 - ✓ vacuum tube service life a problem

The Microprocessor Age

- December 23, 1947, **John Bardeen, William Shockley, and Walter Brattain** develop the **Transistor** at Bell Labs.
- Followed by 1958 invention of the **integrated circuit (IC)** by Jack Kilby of Texas Instruments.
- **IC** led to development of **digital integrated circuits** in the 1960s.
- **First microprocessor developed** at Intel (short for Integrated Electronics) Corporation in 1971.
- Intel engineers **Federico Faggin, Ted Hoff, and Stan Mazor** developed the **4004 microprocessor**.

What is Microprocessors

- What is Microprocessors ??
- The **brain** or **engine** of the PC is the **processor** (sometimes called **microprocessor**), or central processing unit (**CPU**).
- **Integrated circuit** that contains the entire central processing unit of a computer on a single chip.
- A microprocessor (uP) is a computer that is fabricated on an integrated circuit (IC).
- Computers had been around for 20 years before the first microprocessor was developed at **Intel** in 1971.
- The micro in the name microprocessor refers to the physical size.
- **Intel didn't invent** the **electronic computer**, but they were the first to succeed in packing an entire computer on a single **chip** (IC)
- What the Cpu Perform ??
- The CPU performs the system's **calculating** and **processing**.

Microprocessors

- Microprocessors can be characterized based on what ??

–the word size(data handling capacity)

- ✓ 8 bit, 16 bit, 32 bit, etc. processors

–Instruction set structure

- ✓ RISC (Reduced Instruction Set Computer), CISC (Complex Instruction Set Computer)

–Functions

- ✓ General purpose, special purpose such image processing, floating point calculations

–memory organization

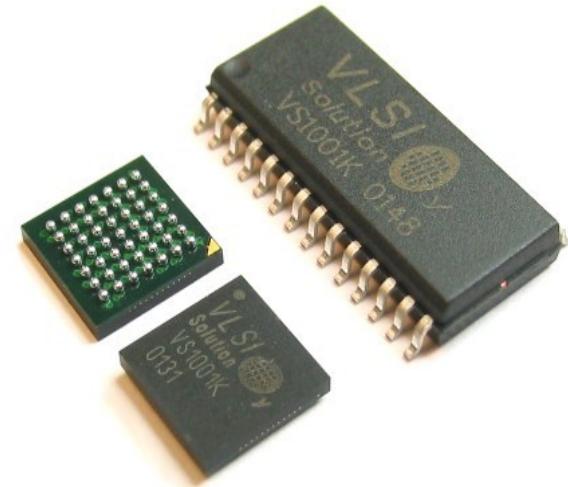
- ✓ Von-Neumann architecture
- ✓ Harvard architecture

Features of a Microprocessor

- Here is a list of some of the **most prominent features** of any microprocessor:
- **Cost-effective:** The microprocessor chips are available at **low prices** and results its **low cost**.
- **Size:** The microprocessor is of **small size chip**, hence is portable.
- **Low Power Consumption:** Microprocessors are manufactured by using **metal-oxide semiconductor technology**, which has low power consumption.
- **Versatility:** The microprocessors are versatile as we can **use the same chip in a number of applications** by configuring the software program.
- **Reliability:** The **failure rate** of an IC in microprocessors is **very low**, hence it is reliable.

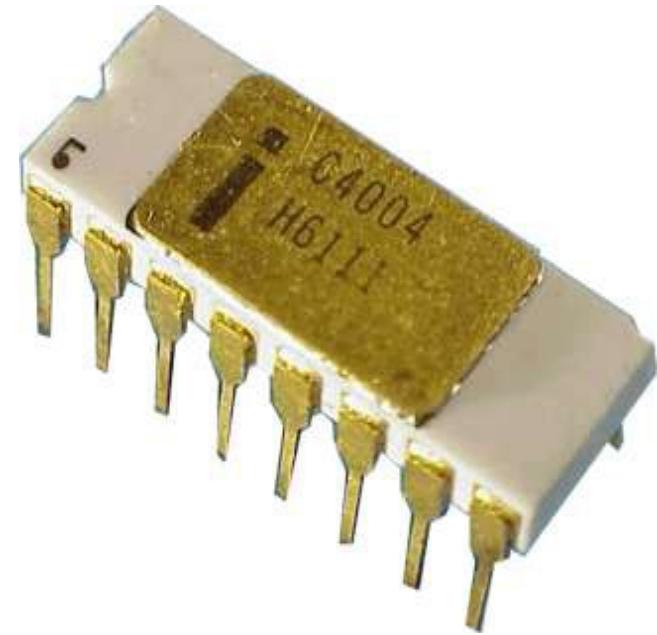
The Intel Age

- Intel didn't invent the electronic computer, but they were the first to succeed in **packing an entire computer** on a single *chip* (IC)
- Intel is generally credited with **creating the first microprocessor** in 1971 with the introduction of a chip called the 4004
- The first microprocessor was developed by what was then a small company **called Intel (short for Integrated Electronics)** in the early 1970s.



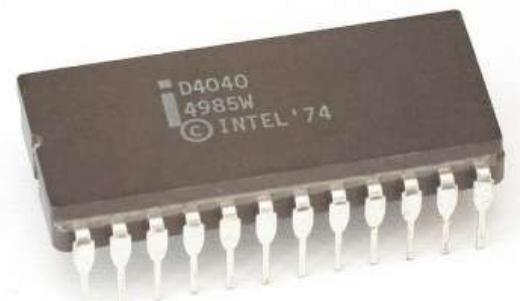
Intel 4004

- World's first microprocessor
- A **4-bit** microprocessor-programmable controller on a chip.
- ✓ A **bit** is a binary digit with a value of one or zero
- ✓ 4-bit-wide memory location often called a **nibble**
- The instruction set contained 45 instructions.
- 4096 location addressable
- Executed instructions at 50 KIPs (kilo instructions per second).
- ✓ slow compared to 100,000 instructions per second by 30-ton ENIAC computer in 1946
- Main problems with early microprocessor were **speed, word width, and memory size**.



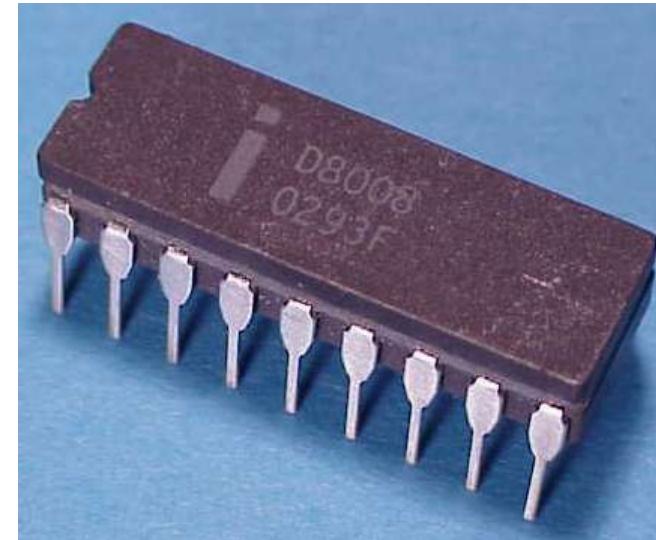
Intel 4040

- Evolution of 4-bit microprocessor ended when
- Intel released the 4040, an updated 4004.
- operated at a **higher speed**; lacked **improvements** in **word width** and **memory size**



Intel 8008

- Intel released 8008 in 1971.
- It was First **8** bit μP
- Extended 8-bit version of 4004 microprocessor
- Addressed expanded memory of 16K bytes.
- A **byte** is generally an 8-bit-wide binary number and a **K** is 1024.
- memory size often specified in K bytes
 - ✓ Contained additional instructions, 48 total.
 - ✓ Provided opportunity for application in more advanced systems.
- engineers developed demanding uses for 8008
- small memory size, slow speed, and instruction set limited 8008 usefulness.



Intel 8008

- Intel introduced 8080 microprocessor in 1971.
- **first of the modern 8-bit microprocessors**
- Motorola Corporation introduced MC6800 microprocessor about six months later.
- other companies soon introduced their own versions of the 8-bit microprocessor

<i>Manufacturer</i>	<i>Part Number</i>
Fairchild	F-8
Intel	8080
MOS Technology	6502
Motorola	MC6800
National Semiconductor	IMP-8
Rockwell International	PPS-8
Zilog	Z-8

cont.

- Only **Intel** and **Motorola** continue to create new, improved microprocessors.
- IBM also produces Motorola-style microprocessors
 - ✓ **Motorola sold its microprocessor division.**
 - ✓ **now called Freescale Semiconductors, Inc.**
- **Zilog** still manufactures microprocessors.
 - ✓ microcontrollers and embedded controllers instead of general-purpose microprocessors

8080

- 8080 addressed **four times** more memory.
- ✓ 64K bytes vs 16K bytes for 8008
- Executed additional instructions; 10x faster.
- ✓ addition taking 20 μ s on an 8008-based system
- required only 2.0 μ s on an 8080-based system
- ✓ TTL (transistor-transistor logic) compatible.
- the 8008 was not directly compatible
- Interfacing made easier and less expensive.



Intel 8085

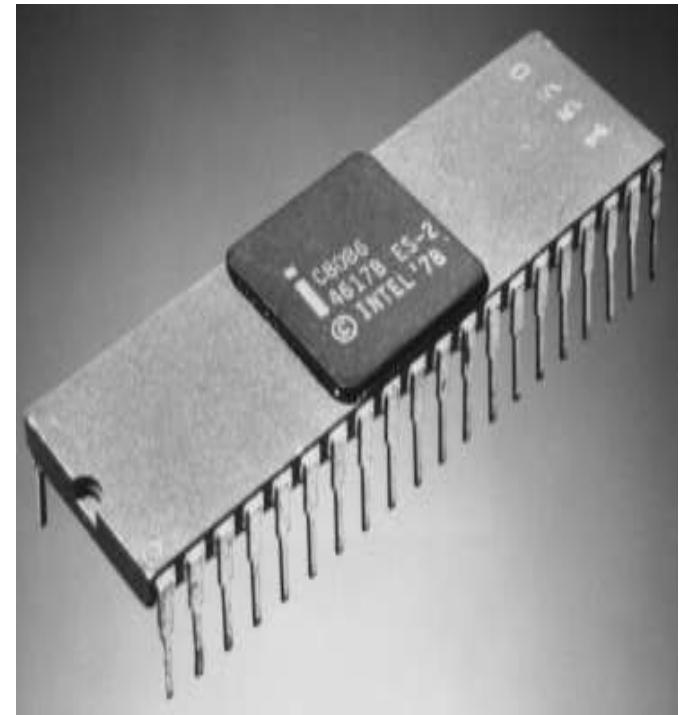
- In 1977 Intel Corporation introduced an **updated version** of the **8080**—the **8085**.
- **Last 8-bit, general-purpose microprocessor** developed by Intel.
- Slightly more advanced than 8080; executed software at an even higher speed.
- ✓ 769,230 instructions per second vs 500,000 per second on the 8080).
- Main advantages of 8085 were its internal clock generator and system controller, and higher clock frequency.



- higher level of **component integration** reduced the **8085's cost** and **increased** its **usefulness**
- Intel has sold over 100 million of the 8085.
 - ✓ its **most successful 8-bit, general-purpos microprocessor.**
 - ✓ also manufactured by many other companies, meaning over 200 million in existence
- Applications that contain the 8085 will likely continue to be popular.
- Zilog Corporation sold 500 million of their 8-bit Z80microprocessors.
- The Z-80 is machine language—compatible with the 8085.
- Over 700 million microprocessors execute 8085/Z-80 compatible code.

INTEL 8086

- Introduced in 1978.
- **It was first 16-bit µP.**
- Its clock speed is 4.77 MHz, 8 MHz and 10 MHz, depending on the version.
- Its **data bus is 16-bit and address bus is 20-bit.**
- It had 29,000 transistors.
- Could execute 2.5 million instructions per second.
- It could **access 1 MB of memory.**
- It had 22,000 instructions.
- It had ***Multiply*** and ***Divide*** instructions.



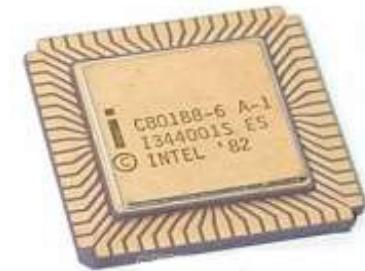
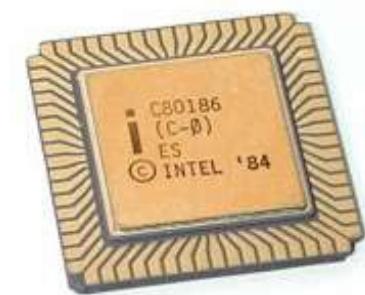
INTEL 8088

- Introduced in 1979.
- It was also 16-bit μP.
- It was created as a **cheaper version of Intel's 8086**.
- It was a **16-bit processor** with an **8-bit external bus**.
- Could execute 2.5 million instructions per second.
- This **chip** became the **most popular** in the computer industry when IBM used it for its first PC.



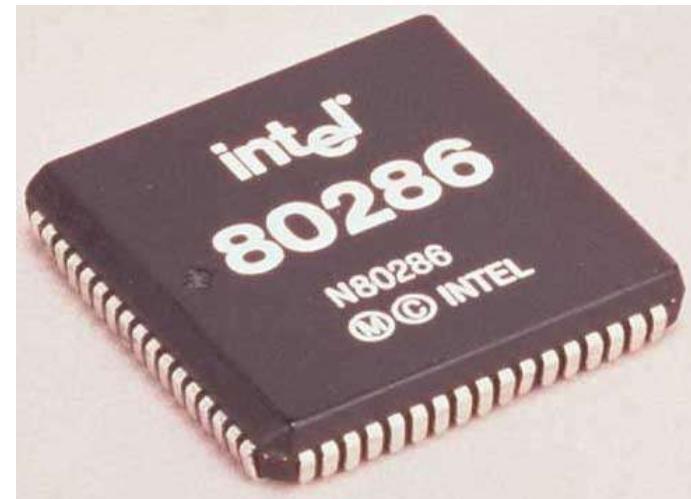
INTEL 80186 & 80188

- Introduced in 1982.
- They were **16-bit µPs**.
- Clock speed was 6 MHz.
- 80188 was a cheaper version of 80186 with an 8-bit external data bus.
- They had additional components like:
 - ✓ Interrupt Controller
 - ✓ Clock Generator
 - ✓ Local Bus Controller
 - ✓ Counters



INTEL 80286

- Even the 1M-byte memory system proved limiting for databases and other applications.
- Intel introduced the 80286 in 1983 . Some books say 82 and update version of 8086
- Its data bus is **16-bit** and **address bus is 24-bit**.
- It could address **16 MB of memory**.
- It had 1,34,000 transistors.
- It could execute 4 million instructions per second.



INTEL 80386

- Applications demanded **faster μPs speeds, more memory, and wider data paths.**
- Led to the 80386 in 1986 by Intel and
- It was **first 32-bit μP**.
- Intel's first practical μPs to contain a 32-bit data bus and 32-bit memory address
- Intel produced an earlier, **unsuccessful 32-bit μPs called iapx-432**
- Through 32-bit buses, 80386 addressed up to **4G bytes of memory**.
- Intel 80386 became the best selling microprocessor in history.
- 80386 DX , 80386 SX , 80386 SL Different version of 80386



INTEL 80486

- Introduced in 1989 ,It was also **32-bit μP.**
- It had 1.2 million transistors.
- Its clock speed varied from 16 MHz to 100 MHz depending upon the various versions.
- ≡ It had five different versions:
 - ✓ 80486 DX
 - ✓ 80486 SX
 - ✓ 80486 DX2
 - ✓ 80486 SL
 - ✓ 80486 DX4
- 8 KB of cache memory was introduced.



INTEL PENTIUM

- Introduced 1993, **Pentium** was similar to **80386** and **80486** microprocessors.
- Originally labeled the P5 or 80586.
- ✓ Intel decided not to use a number because it appeared to be **impossible to copyright a number**
- operated with a clocking frequency of 60 MHz & 66 MHz.
- Cache size was increased to 16K bytes from.
 - ✓ 8K for instruction
 - ✓ 8KB for data
- Memory system up to **4G bytes**.
- Its data bus is 32-bit and address bus is 32-bit.
 - ✓ Wider Data bus virtual reality software and video to operate at more realistic rates
 - ✓ Allow full-frame video displays
- Could execute 110 MIPS(million instructions per second)



- Intel may allow Pentium to replace some RISC (**reduced instruction set computer**) machines.
- Some newer RISC processors execute more than one instruction per clock.
- Motorola, Apple, and IBM produce PowerPC, a RISC with two integer units and a floating point unit.
- boosts Macintosh performance, but slow to efficiently emulate Intel microprocessors
- PowerPC could not keep pace with the Pentium line from Intel

INTEL PENTIUM PRO

- Introduced in **1995**,
- **It was also 32-bit μP.**
- It had 21 million transistors.
- Internal 16K level-one (L1) cache.
 - ✓ 8K data, 8K for instructions
 - ✓ contains 256K level-two (L2) cache
- It was primarily used in server systems.



INTEL PENTIUM II

- Released 1997, represents new direction for Intel.
- Intel has placed Pentium II on a **small circuit board**, instead of being an integrated circuit.
- It was also 32-bit μP.
- Its clock speed was 233 MHz to 500 MHz.
- Could execute 333 million instructions per second.
- **MMX (multimedia extensions) technology was supported.**
- L2 cache & processor were on one circuit.



INTEL PENTIUM II XEON

- Intel announced Xeon in mid-1998,
It was also **32-bit μP**
 - ✓ specifically designed for **high-end workstation** and **server applications**
- Xeon available with 32K L1 cache and L2 cache size of 512K, 1M, or 2M bytes.
- Newer product represents strategy change.
 - ✓ Intel produces a professional and home/business version of the Pentium II



INTEL PENTIUM III

- Introduced in 1999, It was also 32-bit μP.
- Its clock speed varied from 500 MHz to 1.4GHz.
- It had 9.5 million transistors.
- Faster core than Pentium II; still a P6 or Pentium Pro processor.



INTEL PENTIUM IV

- Introduced in **2000**,
- **It was also 32-bit μP.**
- Its clock speed was from 1.3 GHz to 3.8 GHz.
- L1 cache was of 32 KB & L2 cache of 256 KB.
- It had 42 million transistors.
- **All internal connections** were made from **aluminum to copper**.
- Copper is a better conductor.
 - ✓ should allow increased clock frequencies
 - ✓ especially true now that a method for using copper has surfaced at IBM

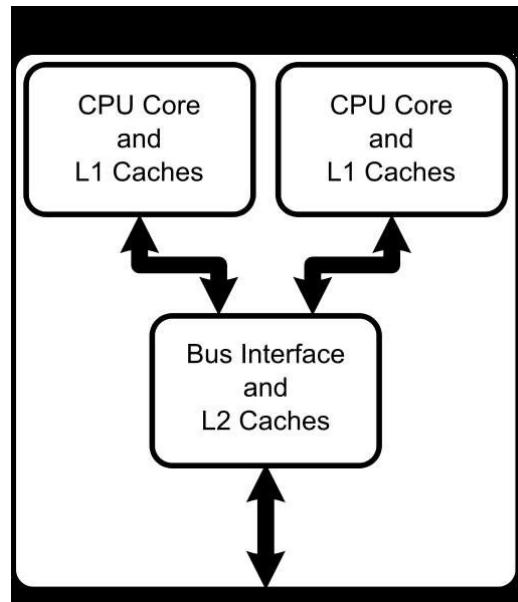


INTEL DUAL CORE

- Introduced in 2006,
- It is 32-bit or 64-bit μP.
- **It has two cores.**
- Both the cores have **their own internal bus** and **L1 cache**, but **share the external bus** and L2 cache (*Next Slide*).
- It supported SMT technology.
- **SMT: Simultaneously Multi-Threading**
- E.g.: Adobe Photoshop supported SMT.



Dual CPU core chip



INTEL CORE 2

- Introduced in 2006,
- **It is a 64-bit μP.**
- Its clock speed is from 1.2 GHz to 3 GHz.
- It has 291 million transistors.
- It has 64 KB of L1 cache per core and 4 MB of L2 cache.
- It is launched in three different versions:
 - ✓ Intel Core 2 Duo
 - ✓ Intel Core 2 Quad
 - ✓ Intel Core 2 Extreme



INTEL CORE I7

- Introduced in 2008,
- **It is a 64-bit μP.**
- It has **4 physical cores**.
- Its clock speed is from 2.66 GHz to 3.33 GHz.
- It has 781 million transistors.
- It has 64 KB of L1 cache per core, 256 KB of L2 cache and 8 MB of L3 cache.



INTEL CORE I5

- Introduced in 2009, It is a 64-bit μ P.
- It has 4 physical cores.
- Its clock speed is from 2.40 GHz to 3.60 GHz.
- It has 781 million transistors.
- It has 64 KB of L1 cache per core, 256 KB of L2 cache and 8 MB of L3 cache.



INTEL CORE I3

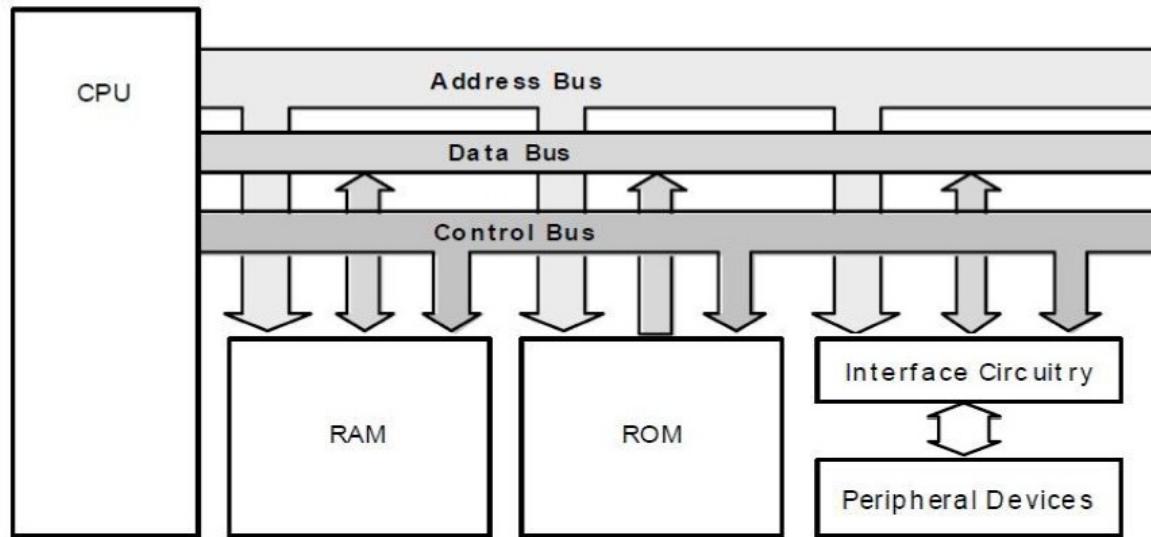
- Introduced in 2010, It is a 64-bit μ P.
- It has 2 physical cores.
- Its clock speed is from 2.93 GHz to 3.33 GHz.
- It has 781 million transistors.
- It has 64 KB of L1 cache per core, 512 KB of L2 cache and 4 MB of L3 cache.



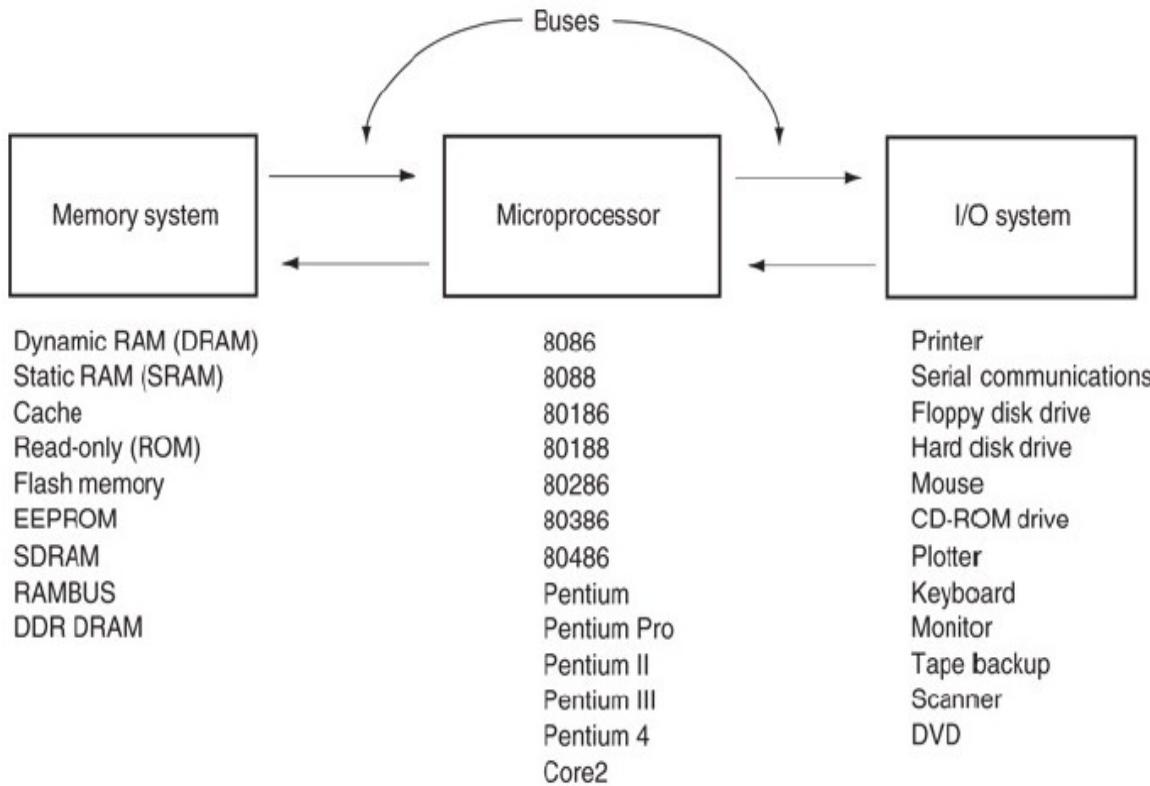
THE MICROPROCESSOR BASED PERSONAL COMPUTER SYSTEM

- Computers have undergone many changes recently.
- Machines that once **filled large areas** **reduced** to **small desktop computer systems** because of the **microprocessor**.
- although compact, they possess computing power only dreamed of a few years ago
- The next slide shows block diagram of the personal computer.
- Applies to any computer system, from early mainframe computers to the latest systems.
- Diagram composed of three blocks (Microprocessor , Memory and I/O System and memory system) interconnected by buses.
- a **bus** is the set of common connections that carry the same type of information

The block diagram of a microprocessor-based computer system (the CPU is the



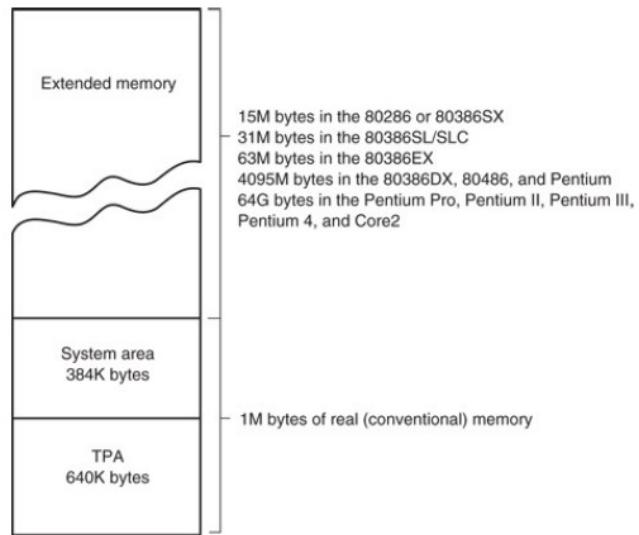
The block diagram of a μ P based computer system



The Memory and I/O System

- The memory structure remains same for all the Intel 80x86 through Pentium IV personal computer systems
- the **memory map** of a personal computer system divided into three main parts (next slide show the diagram of memory)
- The memory system(Main memory) is divided into three main parts:
 - ✓ *Transient Program Area (TPA)* – 640 Kbytes.
 - ✓ *System Area* – 384 Kbytes.
 - ✓ *Extended Memory system (XMS)* – amount of memory depends on the microprocessor used in the personal computer system.

The memory map of a personal computer.



- It noted that the **Extended memory system** is not available in those computers based on **8086** or **8088**.
- In these old computers the **TPA** and **System area** exists but not the Extended memory system.
- Type of microprocessor present determines whether an extended memory system exists.
- The **TPA** is of **size 640 Kb** and **System area** is of **size 384Kb**.
- The **TPA** and **System area** together forms the **real** or **conventional memory** which is of size **1024Kb** or **1 Mb**.

Why real

✓ because each Intel microprocessor is designed to function in this area using its real mode of operation.

- 80286 through the Core2 contain the TPA (640K bytes) and system area (384K bytes) and extended memory

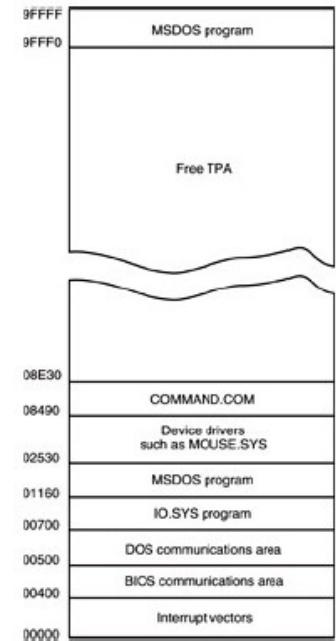
- **Extended memory up** to **15M bytes** in the 80286 and 80386SX;
- **4095M bytes** in 80486 ,80386DX, Pentium microprocessors, excluding the 1Mb real or conventional memory.
- The Pentium Pro through Core2 computer systems have up to 1M less than 4G or 1 M less than 64G of extended memory.

The TPA

- The transient program area (TPA) holds the DOS (**disk operating system**) operating system; other programs that control the computer system.
 - ✓ The **TPA** is a **DOS concept** and not really applicable in Windows
 - ✓ also stores any **currently active** or inactive DOS application programs
 - ✓ length of the TPA is 640K bytes

memory map

- **DOS memory map** shows how areas of **TPA** are used for **system programs**, **data** and **drivers**.
- also shows a large area of memory available for **application programs**
- hexadecimal number to left of each area represents the memory addresses that begin and end each data area
- a hexadecimal number ends with an H to indicate it is a hexadecimal value.
 - ✓ 1234H is 1234 hexadecimal
 - ✓ also represent hexadecimal data as 0x1234 for a 1234 hexadecimal
- The memory map of the TPA



Interrupt Vectors

- occupy the area between 00000 and 00400
- It responsible for **accessing various** features of the **DOS**, **BIOS** and **other application programs.**
- Areas contain transient data to access I/O devices and internal features of the system.

BIOS and DOS communication area

- BIOS is nothing but Basic Input/output System.
- BIOS is a collection of programs that is stored in the **ROM** or **flash memory** that is **used** to control the **Input / Output devices** that is connected to the computer system.
- The BIOS and DOS communication areas have transient data that can be **used** by programs to **access** the **I/O devices** or other parts of the computer system.

MSDOS

- MSDOS occupies two parts of the TPA.
 - ✓ One is at the top of TPA which is considerably small and 16 bytes in length.
 - ✓ The other is at the bottom and it is **larger**.
- The memory size occupied by the DOS depends on the version of the DOS installed

IO.SYS

- The IO.SYS loads into the TPA from the disk whenever an computer system is started.
- IO.SYS contains programs that allow DOS to use keyboard, video display, printer, and other I/O devices often found in computers.
- The IO.SYS program links DOS to the programs stored on the system BIOS ROM.

Device Drivers

- Drivers are programs that **control installable I/O devices**.
- ✓ mouse, disk cache, hand scanner, CD-ROM memory (**Compact Disk Read-Only Memory**), DVD (**Digital Versatile Disk**), or **installable devices**, as well as programs
- Installable drivers control or drive devices or programs added to the computer system.
- DOS drivers normally have an extension of .SYS; MOUSE.SYS.
- Windows uses a file called SYSTEM.INI to load drivers used by Windows.

COMMAND.COM

- COMMAND.COM (**command processor**) **controls operation** of the computer from the **keyboard** when operated in the **DOS mode**.
- COMMAND.COM processes DOS commands as they are typed from the keyboard.
- If COMMAND.COM is erased, the computer cannot be used from the keyboard in DOS mode.
 - ✓ never erase COMMAND.COM, IO.SYS, or MSDOS.SYS to make room for other software
 - ✓ your computer will not function

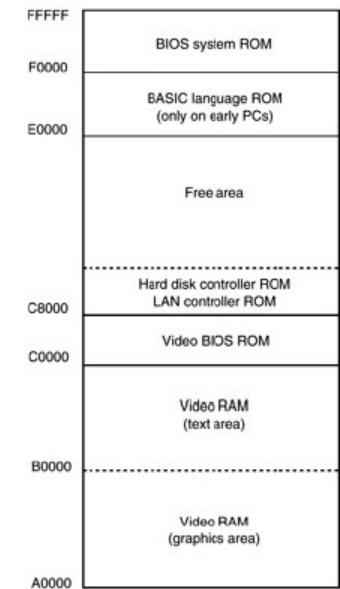
Free TPA

- The **free TPA** holds the **active DOS application programs**.
- These DOS application programs can be exemplified as the word processor , spreadsheet and CAD programs.
- In addition to these, **free TPA** also holds the TSR (Terminate and Stay Resident) programs.
- These remain in the free TPA in an inactive state until initiated by a hot-key or an interrupt. An example of TSR is the calculator program that is activated upon the ALT+C hotkey

- The System area which is smaller than the TPA is considerably important.
- The **system area** contains programs on read only (ROM) or flash memory, and areas of read/write (RAM) memory for data storage.

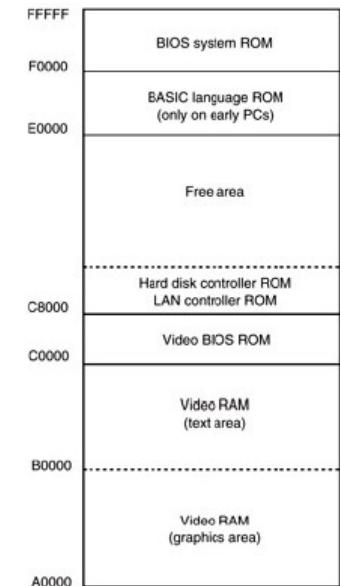
The System Area

- The System area which is smaller than the TPA is considerably important.
 - The **system area** contains programs on **read only (ROM)** or **flash memory**, and areas of **read/write (RAM) memory for data storage.**
 - First area of system space contains **video display RAM** and video control programs on ROM or flash memory..
 - The Video display RAM is stored in two parts
 - ✓ First from A0000H to A7FFFH and is for the graphical data
 - ✓ second from B0000H to B7FFFH for stores the text data.
 - **video BIOS** contains programs that **control** the **video display** of the computer and is located on ROM or falsh memory
- System Area of personal computer



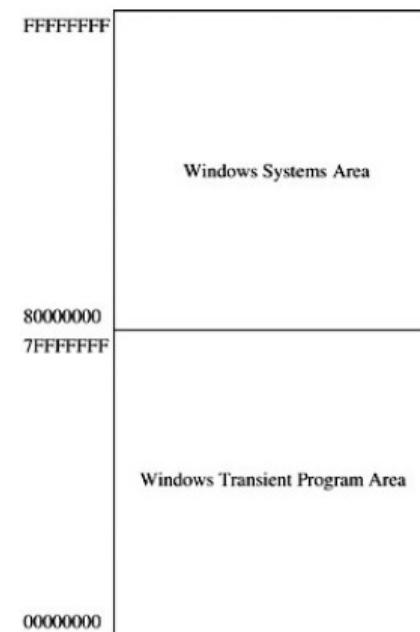
The System Area

- free system area and is called the open system area.
- It is mostly used as the extended memory system
- Expanded memory system allows a 64K-byte page frame of memory for use by applications.
- BASIC language on ROM found in the older IBM based systems.
- In almost all the newer systems this particular area is kept open or free and is also used as RAM to aid the faster operation of DOS application programs.
- System BIOS ROM
 - ✓ controls operation of basic I/O devices connected to the computer system
 - ✓ does not control operation of video
- System Area of personal computer



Windows Systems

- Modern computers use a different memory map with Windows than DOS memory maps.
- The Windows memory map has two main areas; a TPA and system area.
- The difference between it and the DOS memory map are sizes and locations of these areas.
- The memory map used by Windows XP



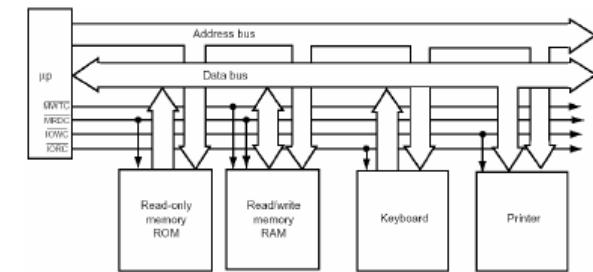
I/O Space and Microprocessor

- I/O devices allow the microprocessor to communicate with the outside world.
- I/O (input/output) space in a computer system extends from I/O port 0000H to port FFFFH.
- μP Controls memory and I/O through connections called buses.
 - ✓ buses select an I/O or memory device, transfer data between I/O devices or memory and the microprocessor, control I/O and memory systems
- Memory and I/O controlled via instructions stored in memory, executed by the μP .

- A common group of wires that interconnect components in a computer system.
- Transfer address, data, & control information between microprocessor, memory and I/O.
- Three buses exist for this transfer of information: address, data, and control.
- The address bus requests a memory location from the memory or an I/O location from the I/O devices.
- The data bus transfers information between the microprocessor and its memory and I/O address space.
- Control bus lines select and cause memory or I/O to perform a read or write operation.
- Advantage of a wider data bus is speed in applications using wide data.

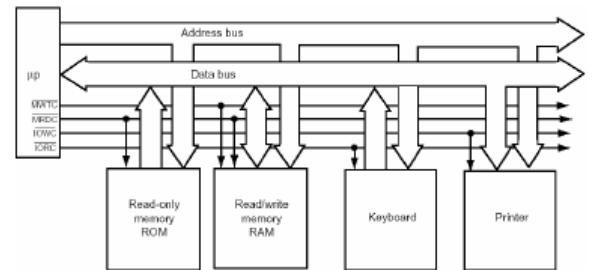
Buses

- The memory map used by Windows XP



- In most computer systems, there are four control bus connections

- ✓ **MRDC (memory read control)**
- ✓ **MWTC (memory write control)**
- ✓ **IORC (I/O read control)**
- ✓ **IOWC (I/O write control).**



Motorola

- At the same time Motorola was developing its own family of microprocessors, the 68000 series
- These were developed as 32-bit processors from start
- As a result, Apple was able to develop its Macintosh computers with true graphical OS from the start

Motorola 68000 (1979)

- Same time as Intel 8086
- 8MHz clock speed
- 32-bit architecture
- 16-bit data bus, 24-bit address bus
- 16 32-bit registers (8 data, 8 address)
- No segment registers required as direct addressing used
- Used pre-fetching to speed up execution

Motorola 68020 (1984)

- 32-bit data and address buses
- Pipeline had 3 stages
- 256 cache added

Motorola 68040 (1991)

- 32-bit data and address buses
- Pipeline had 6 stages
- Floating point unit added
- 4Kbyte caches for data and programs added

Motorola 68060 (1994)

- Superscalar – 3 execution units, 2 integer and 1 FP
- 10 stage pipelines
- 8Kbyte caches for data and programs

Motorola series

- Used in Sun workstations, Apple Macintosh computers, and later Atari computers
- No longer in use in main computer market
- Still used in embedded systems
- Motorola and IBM designed the first PowerPC chip to

Main Characteristics of Motorola series

Processor	External Bus Speed	Data bus width	Address Bus Width	Address Space	Cache
68000	8-20 MHz	32 bit	32 bit	16 Mbytes	none
68020	12-33 MHz	32 bit	32 bit	4 Gbytes	256 bytes
68040	25-40 MHz	32 bit	32 bit	4 Gbytes	4 Kbytes
68060	50-75 MHz	32 bit	32 bit	4 Gbytes	2x4 Kbytes

- In the final years of the 68000 processors, Apple, Motorola and IBM defined a specification for open system software and hardware, and Motorola and IBM designed the first PowerPC chip to meet this specification.

Review of numbering system

- Use of a microprocessor requires working knowledge of numbering systems.
 - binary, decimal, and hexadecimal
- Before converting numbers between bases, digits of a number system must be understood.
- First digit in any numbering system is always zero.
- A decimal (base 10) number is constructed with 10 digits: 0 through 9.
- A base 8 (**octal**) number; 8 digits: 0 through 7.
- A base 4(**Nibble**) number; 4 digits: 0 through 3.
- A base 2 (**binary**) number; 2 digits: 0 and 1.

Computer Organization and Architecture

Chapter

3

Chapter 3:Input output organization

1

Content on this personation

- Peripheral Device
- Input output interface
- Asynchronous data transfer
- Priority interrupt
- Direct memory access(DMA)
- Input output processor (IOP)
- Serial communication

I|O

- The **input-output** subsystem of a computer, referred to as **I|O**.
- **I|O** provides an efficient **mode of communication** between the **central system** and the **outside environment**.
- **Programs** and **data** must be entered into computer memory **for processing** and results obtained from computations must be **recorded** or **displayed** for the user.
- A computer serves **no useful purpose without** the ability to **receive information** from an **outside source** and **to transmit results in a meaningful form**.
- **Input / output** devices attached to the computer are also called **peripherals** .

Peripheral Device

- **Input Devices**
 - Keyboard
 - Optical input devices
 - ✓ Card Reader
 - ✓ Paper Tape Reader
 - ✓ Bar code reader
 - ✓ Digitizer
 - ✓ Optical Mark Reader
 - Magnetic Input Devices
 - Magnetic Stripe Reader
 - Screen Input Devices
 - ✓ Touch Screen
 - ✓ Light Pen
 - ✓ Mouse
 - Analog Input Devices
- **Output Device**
 - Card Puncher, Paper Tape Puncher
 - CRT
 - Printer (Impact, Ink Jet, Laser, Dot Matrix)
 - Plotter
 - Analog
 - Voice

I/O Interface

- **Input-output interface** provides a method for transferring information between **internal storage** and external **I/O devices**.
- **Peripherals connected** to a **computer** need special **communication links** for interfacing them with the central processing unit. Why ?
- The purpose of the communication link is to **resolve the differences** that exist between the **central computer(Cpu)** and each peripheral(**I/O Device**).
- The Major Difference are
 - ✓ **Data Transfer Rate**
 - ✓ **Unit of Information**
 - ✓ **Operating Modes**
 - ✓ **Nature(Made up of)**

The major differences

- I. **Peripherals are electromechanical** and **electromagnetic devices** and their manner of operation is different from the operation of the CPU and memory, which are **electronic devices**. Therefore, a conversion of signal values may be required.
- II. The **data transfer rate** of peripherals is **usually slower** than the transfer rate of the **CPU**, and consequently, a synchronization mechanism may be needed.
- III. **Data codes and formats** in **peripherals(byte)** differ from the word format in the **CPU** and **memory(Word)**.
- IV. The **operating modes of peripherals**(**Asynchronous**) are different from each other and each must be controlled so as not to disturb the operation of other **peripherals connected to the CPU(synchronous)** .

Interface

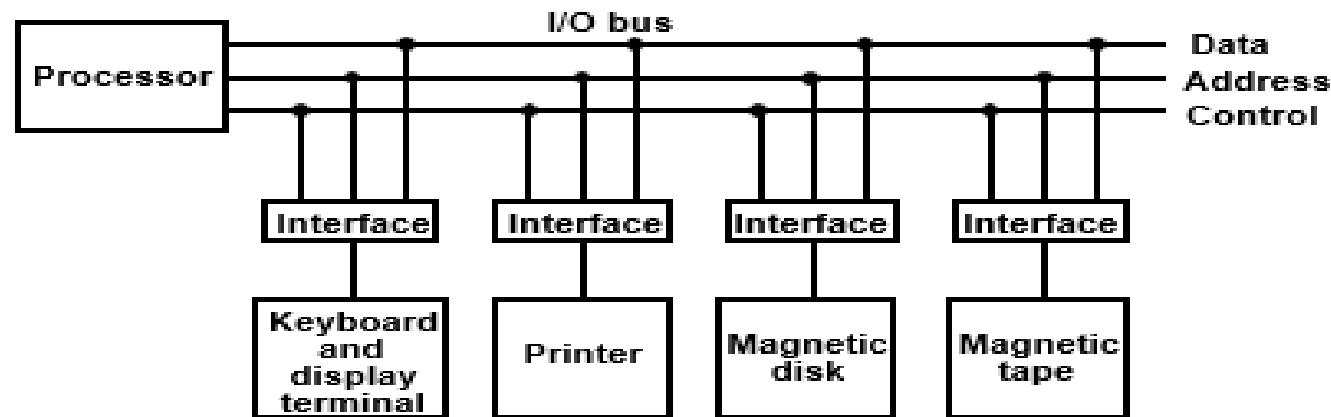
- To **Resolve these differences**, computer systems include **special hardware** components between the **CPU** and **peripherals** to **supervise** and **synchronize** all input and output transfers.
- These components are called **Interface units**.
- Simply the **Interface Unit** Define as
 - **Special hardware components between the CPU and Peripherals**
 - **Supervise and Synchronize** all input and output transfers

Interface

- The **I/O bus** consists of **Data lines**, **Address lines**, and **Control lines**.
- **Each peripheral** device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus.
- **Interface**
 - ✓ Decodes the device address (device code)
 - ✓ Decodes the commands (operation)
 - ✓ Provides signals for the peripheral controller
 - ✓ Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory
- Eg . The **printer** controller controls the **paper motion**, the **print timing**, and the **selection of printing characters**.
- A controller may be housed separately or may be physically integrated with the peripheral

Interface

- A typical **communication link** between the **processor and several peripherals** is shown in Figure below
- The **I/O bus** from the **processor** is **attached** to all peripheral interfaces.
- To communicate with a particular device, the processor places a device address on the address lines.
- When the **interface detects** its own address, it **activates the path** between the **bus lines** and the **device that it controls**.
- All **peripherals** whose address **does not correspond** to the address in the bus are **disabled** by their interface



Inteface

- There are four types of commands that an interface may receive.
- They are classified as **control, status, data output, and data input**
 - A **control command** is issued to **activate the peripheral** and **to inform** it **what to do**.
 - A **status command** is used to **test various status conditions** in the interface and the peripheral
 - A **data output command** causes the interface to **respond by transferring data from the bus into one of its registers**.
 - The **data input command** is the **opposite** of the **data output**. In this case the **interface** receives an item of data from the **peripheral** and places it in its buffer register.

Interface

- **Example1** , a **magnetic tape unit** may be instructed to **backspace** the **tape** by **one record**, to rewind the tape,
- or to start the tape moving in the **forward direction**. The particular **control command** issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operation.
- **Example 2** the computer may wish to **check** the status of the peripheral before a transfer is initiated. During the transfer, one or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals(**Status Command**)

I/O versus Memory Bus

- In addition to communicating with I/O, the processor must communicate with the **memory unit**. Like the **I \ O bus**,
- the memory bus contains **data**, **address**, and **read/write control** lines.
- There are three ways that computer buses can be used to communicate with memory and I/O:
 1. Use **two separate buses**, one for memory and the other for I/O.
 2. Use **one common bus** for both memory and VO but have **separate control lines** for each.
 3. Use **one common bus** for memory and VO with **common control lines**.

Read more about

Asynchronous Data Transfer

- The **internal operations** in a digital system are **synchronized** by means of **clock pulses** supplied by a common pulse generator.
- **Clock pulses** are applied to all registers within a **unit** and all data transfers among internal registers occur **simultaneously** during the occurrence of a clock pulse.
- Two units, such as a CPU and an I\O interface, are designed independently of each other.
- If the registers in the interface **share a common clock** with the CPU registers, the transfer between the two units is said to be **Synchronous**.
- In most cases, the **internal timing** in each **unit is independent** from the other
- in Such Case each uses **its own private clock** for internal registers. In that case, the two units are said to be **Asynchronous** to each other. This approach is widely used in most computer systems.

Asynchronous and Synchronous Data Transfer

■ Synchronous Data Transfer

- ✓ All **Data transfers occur simultaneously** during the occurrence of a clock pulse
- ✓ Registers in the **interface share a common** clock with **CPU** registers
- ✓ The Data transfer Rate almost the same

■ Asynchronous Data Transfer

- ✓ Internal timing in each unit (***CPU and Interface***) is **independent**
- ✓ Each unit **uses its own private** clock for internal registers
- ✓ Data Transfer Rate is Different

Asynchronous Data

- **Two type of Asynchronous Data Transfer Methods**
- **Strobe pulse**
 - ✓ A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur
- **Handshaking**
 - ✓ A control signal is accompanied with each data being transmitted to indicate the presence of data
 - ✓ The receiving unit responds with another control signal to acknowledge receipt of the data

Strobe Control

- The **strobe control** method of asynchronous data transfer employs a single control line to time each transfer.
- The **strobe** may be activated by either the **source** or the **destination unit**
- The Figure shows a source-initiated transfer.
- The **data bus** carries the binary information from **source unit** to the **destination unit**.
- Typically, **the bus** has **multiple lines** to transfer an entire **byte** or **word**.
- The **strobe** is a single line that informs the destination unit when a **valid data word** is available in the bus.

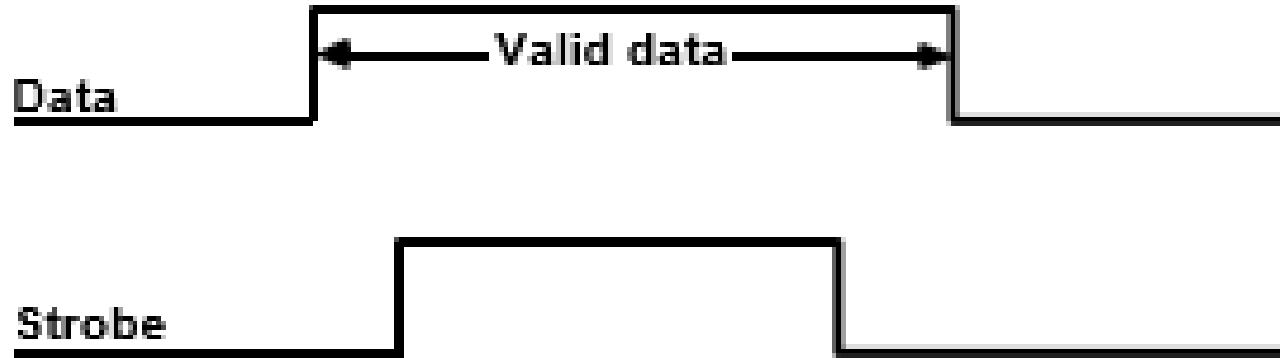
Strobe Control

- the source unit first places the data on the data bus.
- After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse
- The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- The source removes the data from the bus a brief period after it disables its **strobe pulse**.
- the source does not have to change the information in the data bus.
- the strobe signal is disabled indicates that the data bus does not contain valid data. New valid data will be available only after the strobe is enabled again.

Strobe Control



Timing Diagram



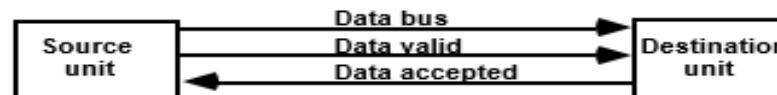
Handshaking

- The disadvantage of the strobe method is that the source unit that initiates the transfer has **no way of knowing** whether the **destination unit** has actually **received the data** item that was placed in the bus.
- Similarly, a **destination unit** that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.
- The **handshake** method solves this problem by introducing a **second control signal** that provides a **reply** to the unit that **initiates the transfer**.

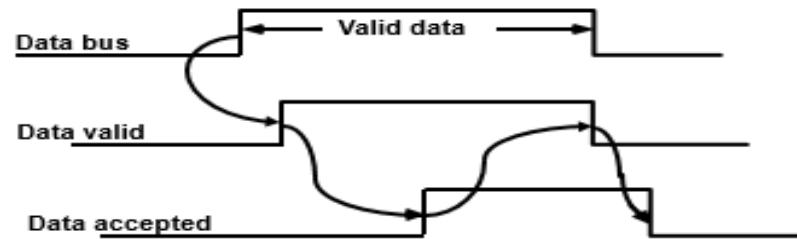
Handshaking

- **One control line** is in the same direction as the data flow in the bus from the source to the destination.
- It is used by the **source unit** to inform the destination unit whether there are valid data in the bus.
- The other **control line** is in the other direction from the **destination to the source**. It is used by the destination unit to inform the source whether it can accept data.
- The sequence of control during the transfer depends on the unit that initiates the transfer.

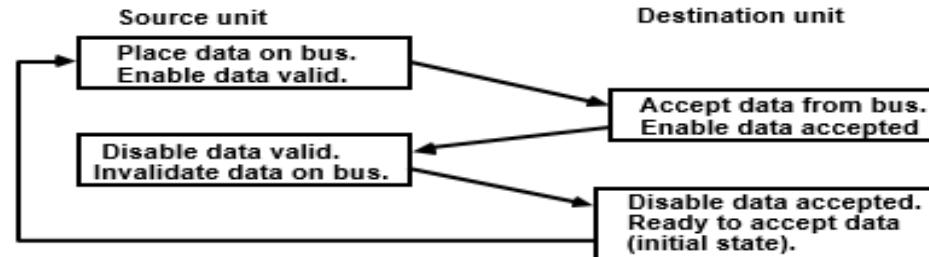
Block Diagram



Timing Diagram



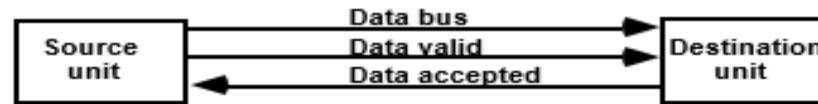
Sequence of Events



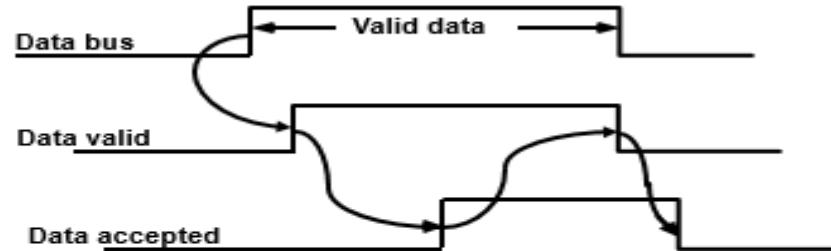
Handshaking

- The Figure shows the data transfer procedure when **initiated by the source**.
- The **two handshaking lines** are **data valid**, which is generated by the source unit, and **data accepted**, generated by the **destination unit**.
- The timing diagram shows the exchange of signals between the two units.

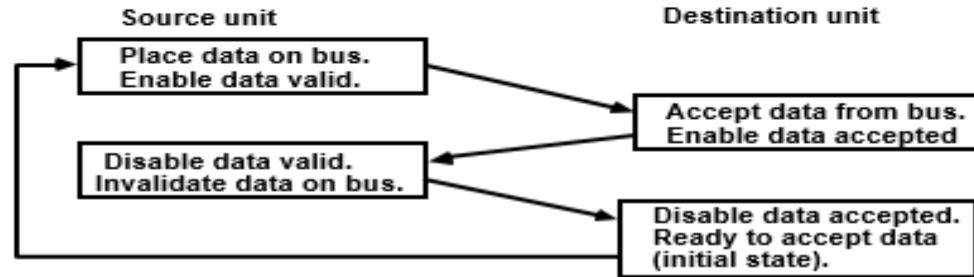
Block Diagram



Timing Diagram



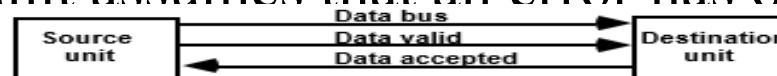
Sequence of Events



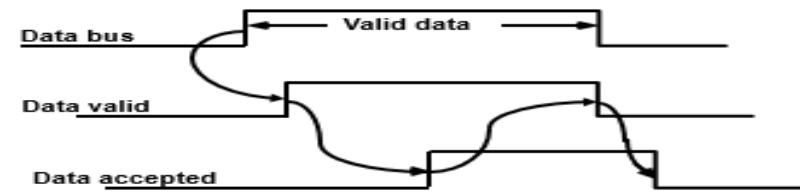
Handshaking

- The sequence of events listed shows the four possible states that the system can be at any given time
- The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal.
- The data accepted signal is activated by the destination unit after it accepts the data from the bus.
- **Timeout** : If the return **handshake signal does** not respond within a given time period, the unit assumes that an error has occurred

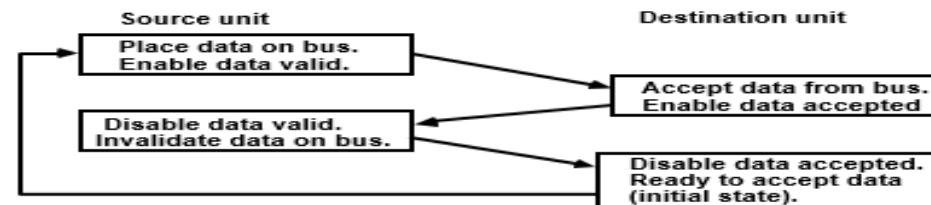
Block Diagram



Timing Diagram

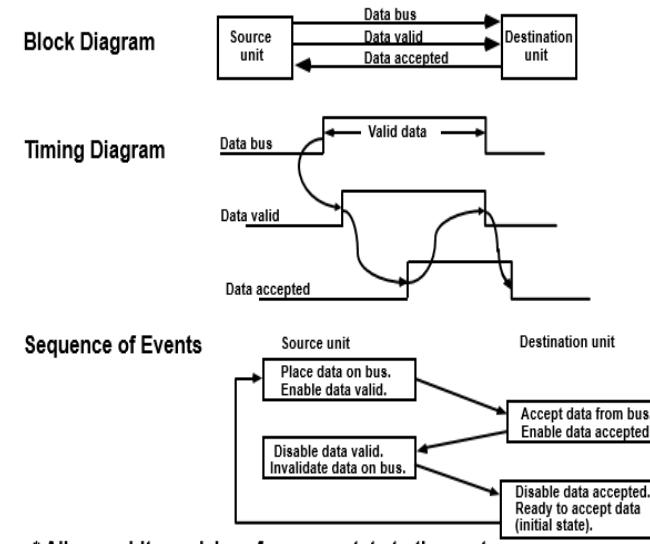


Sequence of Events



Handshaking

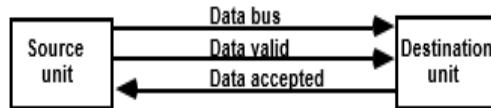
- The **source unit** initiates the transfer by **placing the data** on the bus and enabling its data valid signal.
- The **data accepted signal** is activated by the **destination unit** after it accepts the data from the bus.
- The **source unit** then **disables its data valid signal**, which invalidates the data on the bus.
- The **destination unit** then **disables its data accepted signal** and the system goes into its initial state.
- The **source** does not send the next data item until after the **destination unit shows its readiness** to accept new data by disabling its data accepted signal.
- The same as for destination initiated



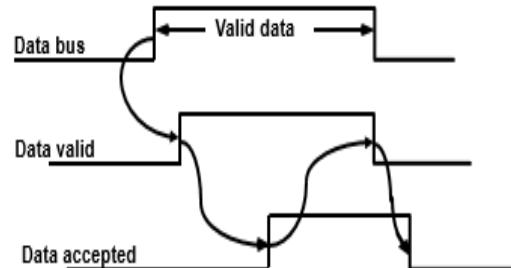
Source and destination initiated handshaking

SOURCE-INITIATED TRANSFER USING HANDSHAKE

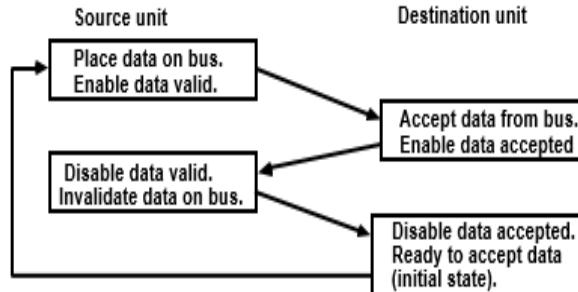
Block Diagram



Timing Diagram



Sequence of Events



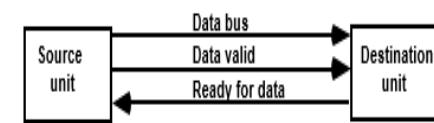
* Allows arbitrary delays from one state to the next

* Permits each unit to respond at its own data transfer rate

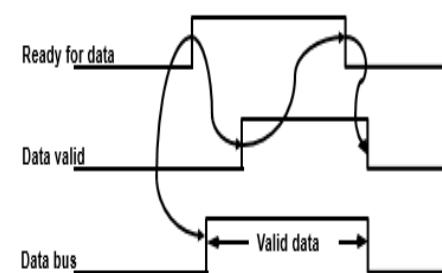
* The rate of transfer is determined by the slower unit

DESTINATION-INITIATED TRANSFER USING HANDSHAKE

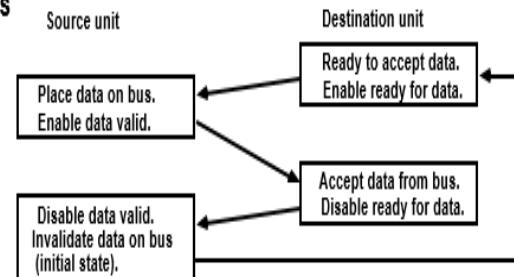
Block Diagram



Timing Diagram



Sequence of Events



* Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units

* If one unit is faulty, data transfer will not be completed

-> Can be detected by means of a *timeout* mechanism

Asynchronous Serial Transfer

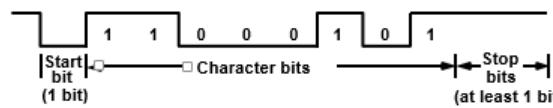
- The **transfer of data** between two units may be done in **parallel** or **serial**.
- In **parallel data transmission**, each **bit** of the message has **its own path** and the total **message is transmitted at the same time**. This means that an n-bit message must be **transmitted through n separate conductor paths**.
- In **serial data transmission**, each bit in the message is **sent in sequence one at a time**. This method requires the use of **one pair of conductors** or one conductor and a common ground.
- **Parallel transmission** is faster but **requires many wires**. It is used for **short distances** and where **speed is important**.

Asynchronous Serial Transfer

- **Serial transmission** is **slower but is less expensive** since it requires **only one** pair of conductors.
- Serial transmission can be synchronous or asynchronous.
- **In synchronous transmission**, the two units **share** a **common clock** frequency and bits are transmitted continuously at the rate dictated by the clock pulses.
- Serial transmission can be synchronous or asynchronous. In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses.
- Synchronization signals are transmitted periodically between the two units to keep their clocks in step with each other.

Asynchronous Serial Transfer

- In **Asynchronous transmission**, binary information is **sent only** when it is available and the line **remains idle** when there is no information to be transmitted.
- This is in contrast to **Synchronous transmission**, where bits must be **transmitted continuously** to keep the **clock frequency** in **both units synchronized** with each other.
- A serial **Asynchronous data** transmission technique used in many interactive terminals employs special bits that are inserted at both ends of the character code.
- With this technique, each character consists of three parts: a **start bit**, the **character bits**, and **stop bits**.



- the 1-state when no characters are transmitted. The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character. The last bit called the stop bit is always a 1.

Individual Assignment -2

- **About mode of Transfer Modes of Transfer**
- Binary information received from an external device is usually **stored in memory** for later processing.
- Information transferred from the central computer into an external device originates in the memory unit.
- The CPU merely executes the **i/o instructions** and **may accept the data temporarily**, but the ultimate source or destination is the memory unit.
- Data transfer between the central computer and i/o devices may be handled in a variety of modes.
- Some modes use the **CPU as an intermediate path**; others transfer the data **directly to and from the memory unit**.
- Data transfer to and from peripherals may be handled in one of three possible modes:
 1. Programmed i/o
 2. Interrupt-initiated i/o
 3. Direct memory access (DMA)

Modes of Transfer

- Binary information received from an external device is usually **stored in memory** for later processing.
- Information transferred from the central computer into an external device originates in the memory unit.
- The CPU merely executes the **i/o instructions** and **may accept the data temporarily**, but the ultimate **source** or **destination** is the **memory unit**.
- Data transfer between the central computer and i/o devices may be handled in a variety of modes.

Modes of Transfer

- Some modes use the **CPU as an intermediate path**; others transfer the data **directly to** and **from the memory unit**.
- Data transfer to and from peripherals may be handled in one of three possible modes:
 1. Programmed i/o
 2. Interrupt-initiated i/o
 3. Direct memory access (DMA)

Programmed i/o

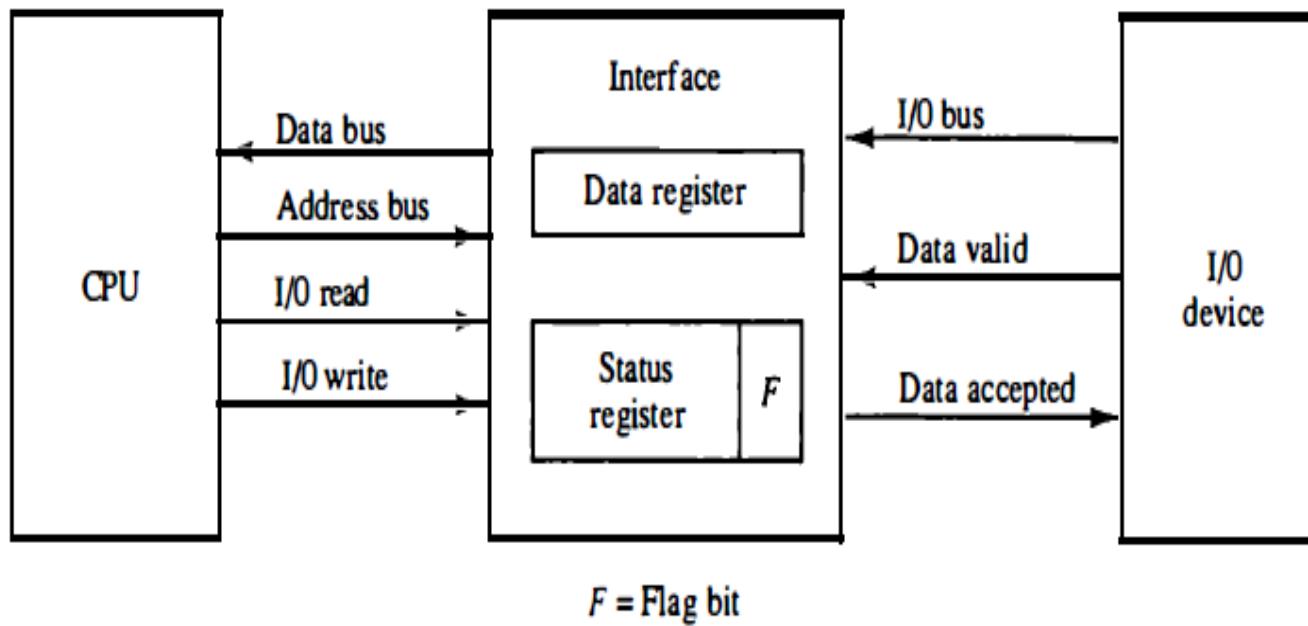
- Programmed I/O operations are the result of **I/o instructions written** in the **computer program**.
- Each **data item transfer** is initiated by an **instruction** in the **program**.
- Usually, the transfer is **to** and from a **CPU register** and **peripheral**.
- Other instructions are needed to transfer the data **to** and **from** CPU and memory.
- Transferring data under **program control** requires **constant monitoring** of the **peripheral** by the **CPU**.
- Once a data transfer is initiated, the **CPU** is required to **monitor** the interface to see when a transfer can again be made.
- It is up to the **programmed instructions executed** in the **CPU** to keep close tabs on everything that is taking place in the interface unit and the i/o device.

Example Programmed i|o

- In the programmed $I|O$ method, the **I|O device** does **not** have **direct access to memory**.
- A transfer from an **I/O device** to **memory** requires the execution of several instructions by the **CPU**, including an **input instruction** to **transfer** the data from the **device** to the **CPU** and a **store instruction** to transfer the data from the CPU to memory.

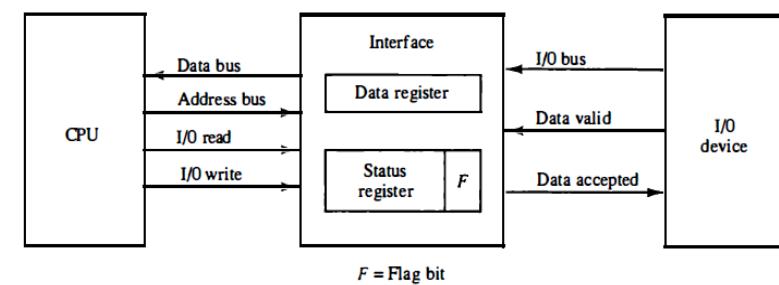
Example Programmed i/o

- Other instructions may be needed to **verify** that the **data** are available from the device and to **count** the numbers of words **transferred**.
- An example of data transfer from an VO device through an interface into the CPU is shown in the below diagram



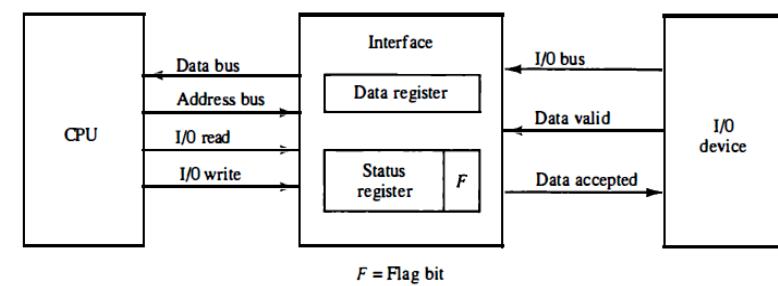
Example of Programmed I|O

- The device transfers **bytes of data** one at a **time** as they are available.
- When a **byte** of data is available, the device places it in the **I|O bus** and enables its **data valid line**.
- The **interface accepts** the byte into its **data register** and **enables** the **data accepted line**.
- The interface sets a bit in the status register that we will refer to as an F or "**flag**" bit.
- The device can now **disable the data valid line**, but it will not transfer another byte until the **data accepted** line is **disabled** by the interface.
- A program is written for the computer to **check** the **flag** in the **status register** to determine if a byte has been placed in the data register by the **I|O device**.



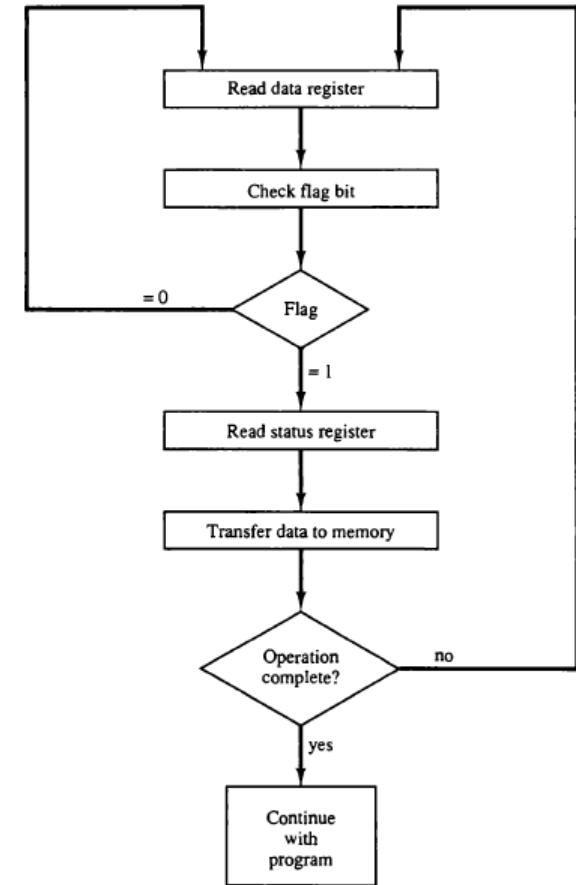
Example of Programmed I/O

- This is done by reading the **status register** into a **CPU register** and checking the value of the flag bit.
- If the **flag** is equal to **1**, the **CPU reads** the **data** from the data register.
- The **flag bit** is then cleared to **0** by either the **CPU** or the **interface**, depending on how the interface circuits are designed.
- Once the **flag is cleared**, the interface **disables** the **data accepted line** and the **device** can then transfer the **next data byte**.



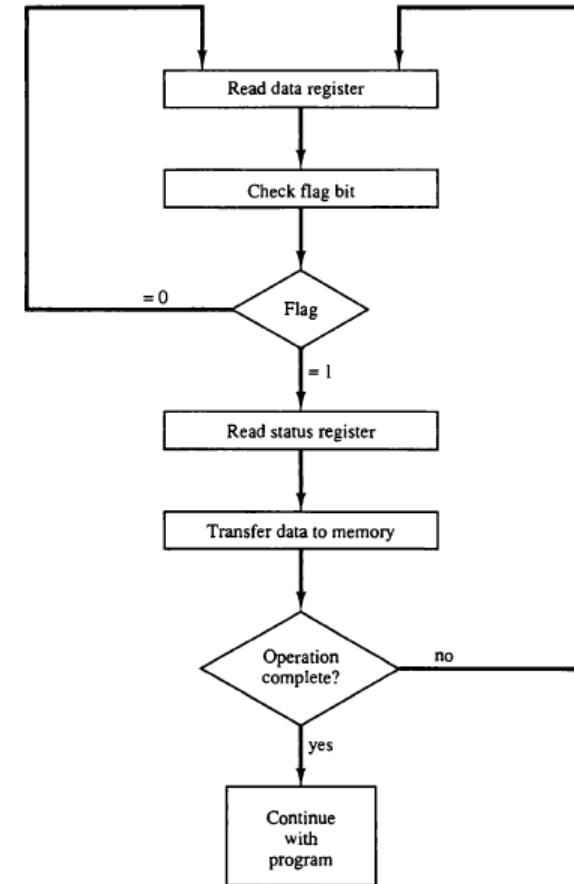
Example of Programmed I/O

- A **flowchart** of the program that must be written for the CPU is shown in the left Side.
- It is assumed that the device is sending a sequence of bytes that must be **stored in memory**.
- The transfer of each byte requires three instructions:
 - ✓ **Read the status register.**
 - ✓ **Check the status** of the **flag** bit and branch to step 1 if not set or to step 3 if set.
 - ✓ **Read the data register.**



Example of Programmed I/O

- The programmed I/O method is particularly useful in **small low-speed computers** or in **systems that are dedicated to monitor** a device continuously.
- The difference in information **transfer rate** between the CPU and the I/O device makes this type of transfer **inefficient**.
- The **Disadvantage** of the programmed I/O method is CPU **wasting time** while **checking** the **flag instead** of doing some other useful processing task.



Interrupt-initiated i/o

- In the **programmed I /o method**, the **CPU** stays in a **program loop** until the i/o unit indicates that it is ready for data transfer
- This is a **time-consuming process** since it keeps the **processor busy needlessly**. It can be avoided by using an **interrupt facility** and **special commands** to inform the interface to issue an **interrupt request** signal when the data are available from the device
- An alternative to the CPU constantly monitoring the **flag** is to let the **interface** inform the computer when it is **ready** to **transfer data**. This mode of transfer uses the **interrupt facility**.
- While the CPU is **running a program**, it **does not check the flag**. However, when the **flag is set**, the computer is momentarily(quickly) **interrupted** from proceeding with the **current program** and is informed of the fact that the **flag** has been set.
- The CPU deviates(Turns) from what it is doing to take care of the **input or output transfer**. After the transfer is completed, the computer returns to the **previous program** to continue what it was doing **before the interrupt**.

Priority Interrupt

- Data transfer between the CPU and an i/o device is initiated by the CPU.
- However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU.
- The readiness of the device can be determined from an interrupt signal.
- The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer.
- In a typical application a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request.

Priority Interrupt

- The **first task** of the interrupt system is to identify the **source** of the **interrupt**.
- There is also the possibility that **several sources** will **request service simultaneously**. In this case the system must also decide which device to service first.
- A **priority interrupt** is a **system** that **establishes a priority over the various sources** to determine which **condition** is to be **serviced first** when **two or more requests** arrive **simultaneously**.
- The system may also determine which conditions are permitted to **interrupt** the computer while another **interrupt** is being serviced.

- Devices with **high speed transfers** such as **magnetic disks** are given **high priority**, and **slow devices** such as **keyboards** receive **low priority**.
- When two devices interrupt the computer at the same time, the computer services the device, with the **higher priority first**.
- Establishing the priority of simultaneous interrupts can be done by software or hardware.
 - 1) Software : Polling
 - 2) Hardware : Daisy chain, Parallel priority

DMA

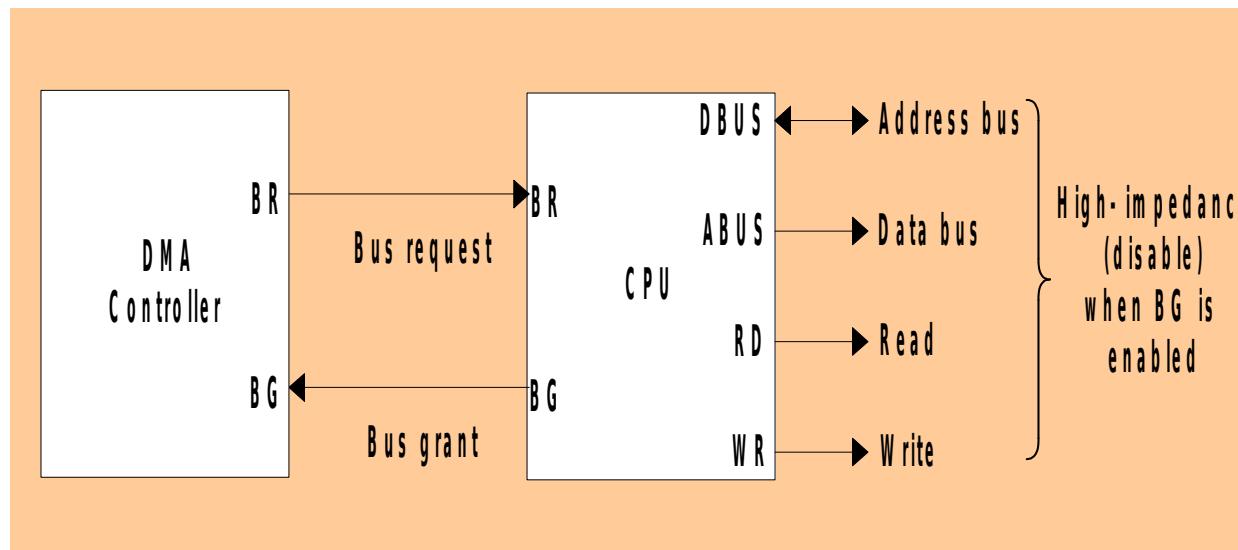
- The **transfer** of data between a **fast storage** device such as **magnetic disk** and **memory** is often **limited** by the **speed** of the **CPU**.
- Removing the **CPU** from the **path** and **letting** the **peripheral device manage** the **memory buses directly** would improve the **speed** of **transfer**.
- This Transfer Technique is called **Direct memory Access (DMA)**
- During the transfer , The Cpu is **idle** and has **no control** of the **memory buses**
- The **DMA Controller** Takes over the **buses** to **manage** the **transfer directly** between the **I|o device** and **memory**

DMA

- The **DMA Controller** needs the usual circuits of an interface to communicate with the cpu and I|O deice
- In Addition, it needs an **address register**, a word **counter register** and a set of **Address line**
- The **Address registers** and **address line** are used for **direct communication** with **memory**
- The **Word Count register** Specifies the **number of Words** that Must be **transferred**.
-

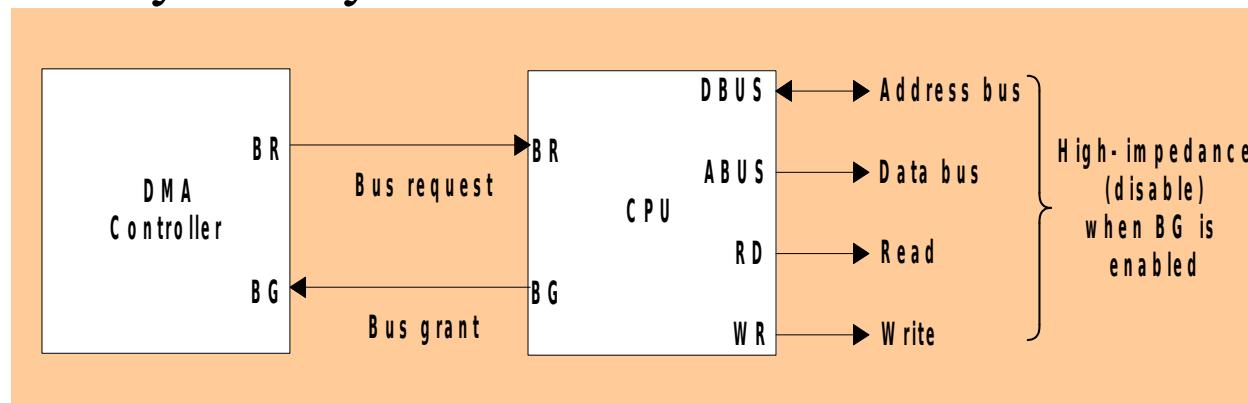
DMA

- The **Two Control Signal** in the Cpu control that Facilitate the **DMA Transfer (Bus Requested & bus Grant BR, BR)**
- The **BR signal** used by the **DMAC** to **Request cpu** to relinquish (**turn down**) control of the **buses**.
- When **BR input active**, The **Cpu terminate** the **execution** of the **current Instruction** and place the **address bus**, the **data bus** **read-write line** in high- Impedance state(Disable)



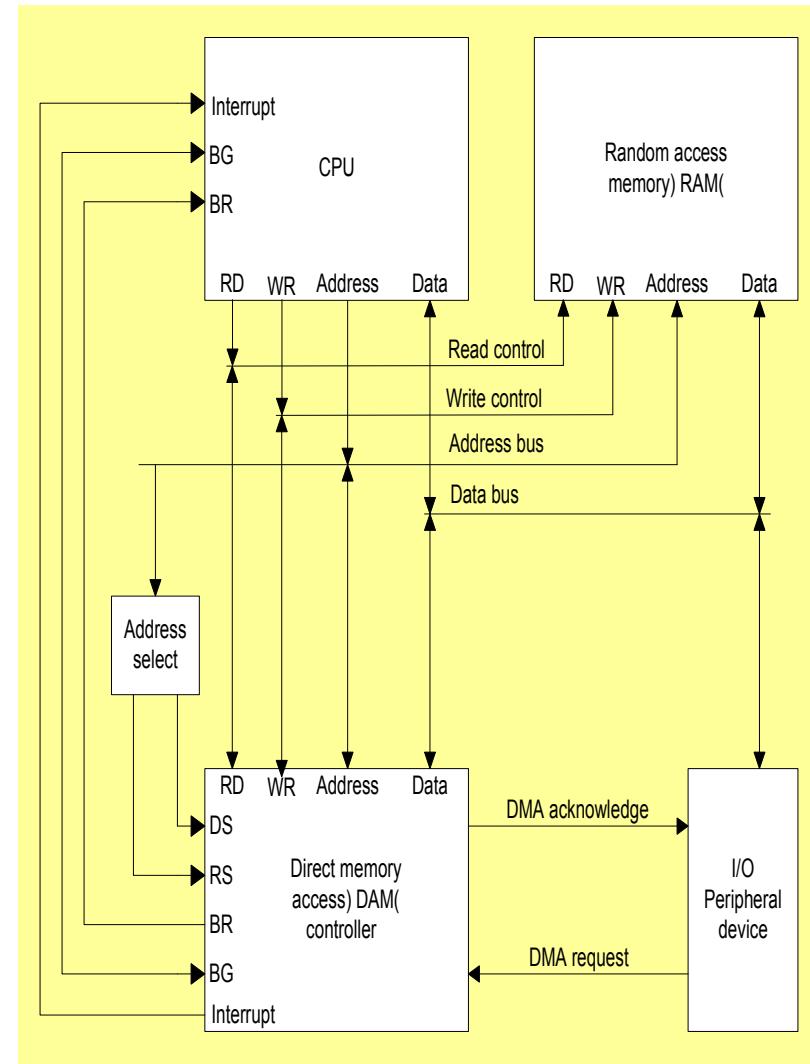
DMA

- The **Cpu active BG** output to inform **The DMA** to inform External **DMA** that the bus are in High –Impedance state
- Then The DMA that Originate the **bus request** can **control the bus** to **transfer Form I|O** to memory with out the intervention of **of CPU**
- When the **DMA terminate** the transfer it **disable** the **bus request line** , the **Cpu disable** the **bus grant** take control of the buses, and return to normal operation
- When the **DMA control** the bus, it **communicate Directly** with memory. That is way we say **DMA**



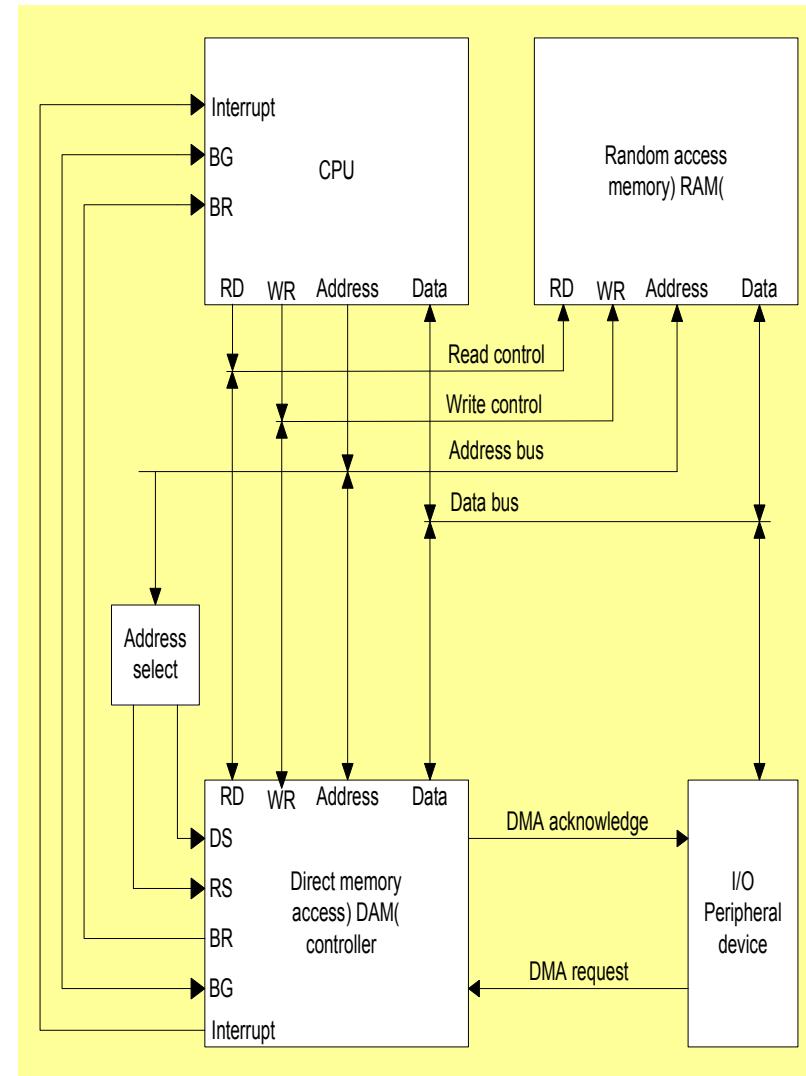
DMA Transfer (I/O to Memory)

- The Cpu Communicate with the DMA through the Address and data buses.
 - The DMA has its own address, which activates the DS and RS line.
 - The CPU initializes the DMA through the data bus
 - Once the DMA receives the start control command , it can the start transfer between the I|O device and memory
 - When the peripheral device send DMA request the DMAC active BR line. Informing the Cpu to give up the bus line
 - The CPU respond with BG line
 - Then DMA puts the current value of its address registers into address bus , initiates the RD and WR signals
 - The DMA sending ACK Signal to I|O device



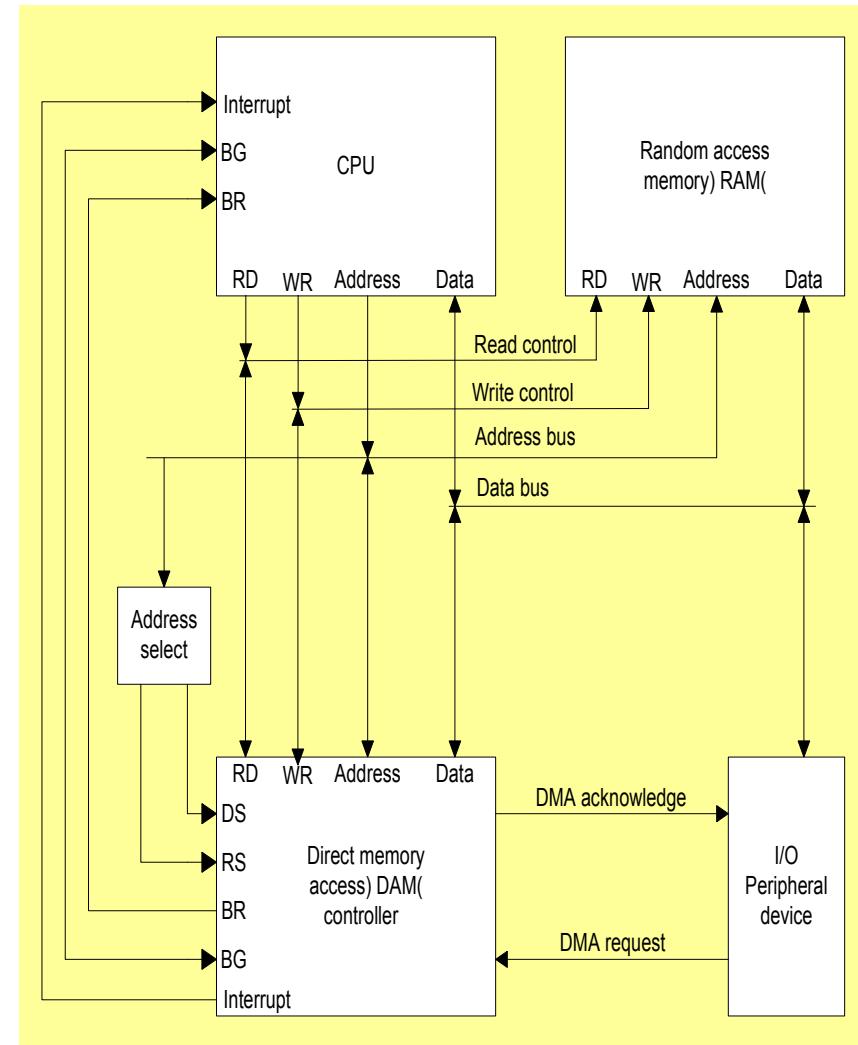
DMA Transfer (I/O to Memory)

- When **Peripheral device** receive **DMA ACK**, it puts a word in the **data bus(for write)** or **receives** a word from the **data bus(for read)**.
- For Each **word transfer**, the **DMA Increment** the **address registers** and **decrements** its word **count register**
- If the **word count** does not reach **zero**, the **DMA checks** the request line coming from the peripheral device.
- For a high **speed device**, the line will be **active** as soon as the **previous transfer** is **complete**
- If the **peripheral speed** is **slower**, the **DMA disable** the **bus request** line so the **Cpu** can **continuous** to **executes** its program



DMA Transfer (I/O to Memory)

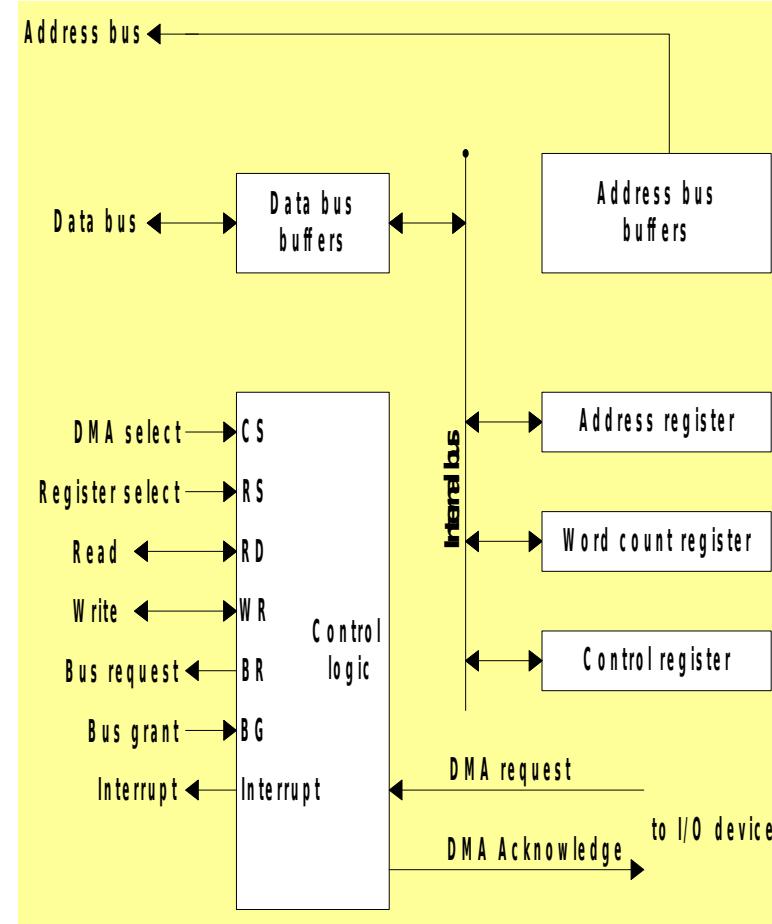
- 1) I/O Device sends a DMA request
- 2) DMAC activates the **BR** line
- 3) CPU responds with **BG** line
- 4) DMAC sends a DMA acknowledge to the I/O device
- 5) I/O device puts a word in the data bus (*for memory write*)
- 6) DMAC write a data to the address specified by **Address register**
- 7) Decrement **Word count register**
- 8) if **Word count register = 0**
EOT interrupt
- 9) if **Word count register $\neq 0$** DMAC checks the DMA request from I/O device(another request)



DMA Controller (Intel 8237 DMAC)

- The **DMA Controller** needs the an **address register**, a **word counter registers**, and a **set of Address lines**
- The **Address register** and **address lines** used for **Direct communication with Memory**
- The **word Count register** specify the number of word that must be **transferred**.
- The **data transfer** may be done directly between the **device(I|O)** and memory under control of **DMA**
- The **Register** in the **DMA** are selected by the **CPU** through the **address bus** by enabling the **DS (DMA selector through CS)** and **RS selector Inputs**
- **The RD(Read) and WR (Write) inputs** are Bidirectional
- When the **BG(Bus grant)** inputs is 0, the Cpu can communicate with the **DAM Register** through the data Bus to **read** from or writ to the **DMA registrars**.

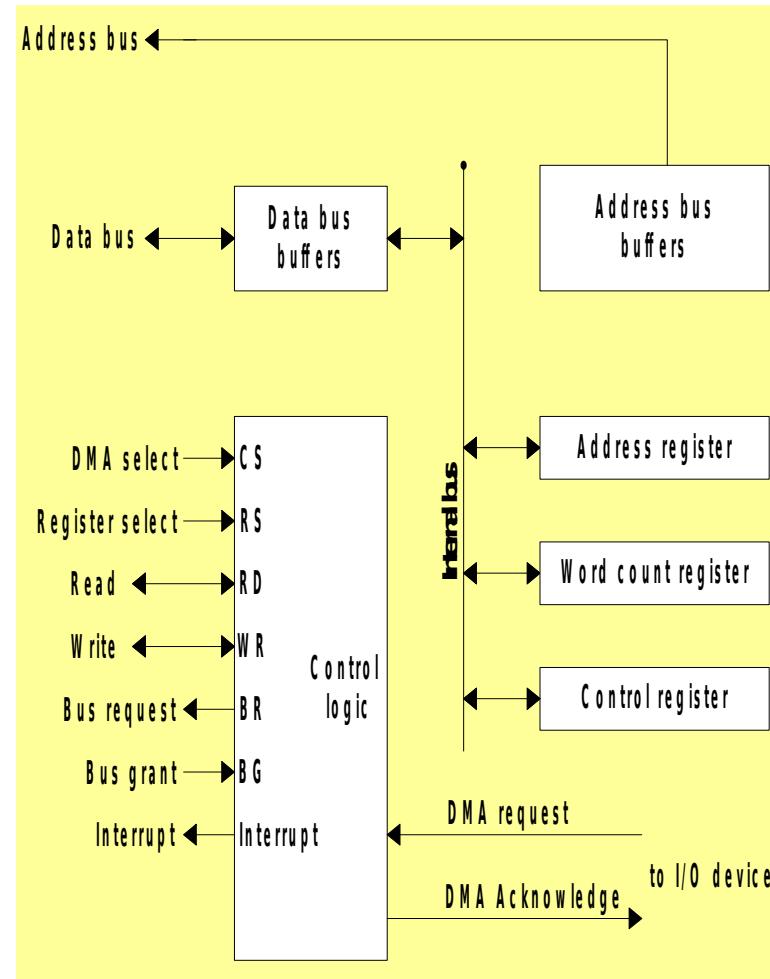
Block Diagram DMA



DMA Controller (Intel 8237 DMAC)

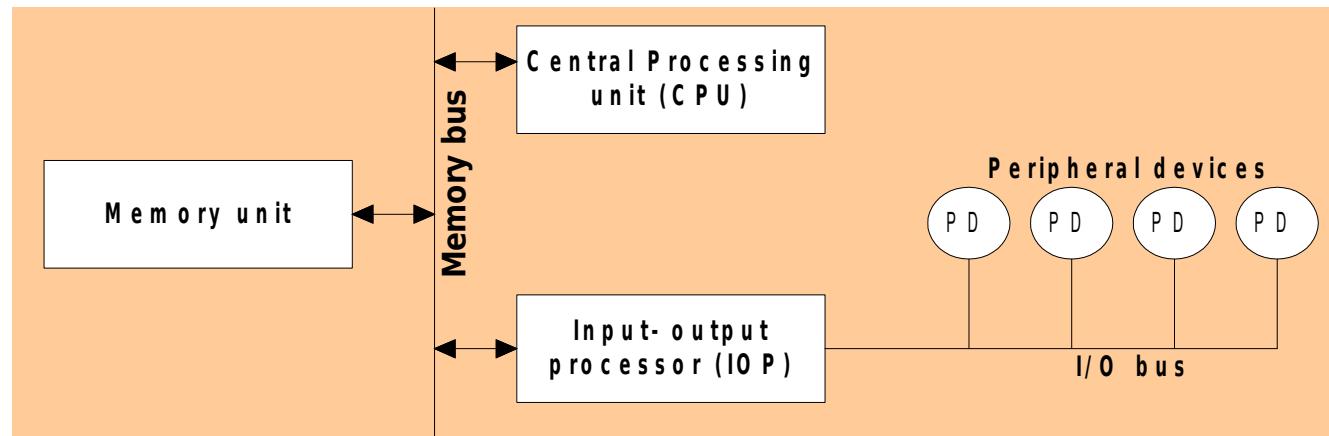
- When **BG = 1**, then Cpu has relinquished (give up) the **buses** and the **DMA** can **communicate Directly** with the **memory** by Specifying an address in the **address bus** and activating the **RD or WR control**.
- the **DMA controller** has **three registers**: an **address registers**, a **word count register**, and a **control register**.
- The **Address Registers** Contain an address to specify the **desire location in memory**.
- The Address register is **incremented** after **each words** that is **transferred to memory**.
- The **Word Count register** holds the number of **word** to be **transferred**.

Block Diagram DMA

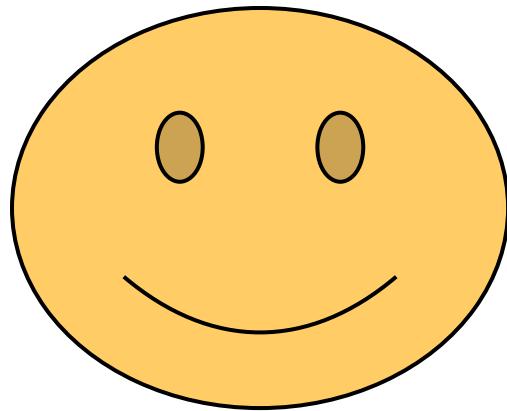


Input –output Processor(IOP)

- The **IOP** is similar to a **CPU except** that is **design** to handle the detail of I/O processing
- **IOP** Communicate Directly with all **I/O Device**
- The **IOP** can **fetch** and **executes** its own instruction under supervision of CPU
- **IOP** instructions are specifically **design** to **facilitate** the **I/O transfer**
- The **IOP** provide a **path** for transfer of **data** between various **peripheral device** and the **memory unit**
- In most **computer systems**, the **CPU** is the **master** while the **IOP** is a slave **processor**.



Thank You!!!





ADMAS UNIVERSITY

Department Computer Science
Computer Organization and Architecture
Individual Assignment 2

Instructor: - Ebdu. J
Submission date June 5, 2020

Instruction

1. Don't forget to write your **Name, ID and Department**
2. There are only **four questions** for this assignment.
3. **Do independently** without copying from each other's. Copying from other invalid your work.
4. There is only **one page** included for this assignment.
5. The answer must **PDF or Document** formats. **Image or Picture** formats are **not acceptable**
6. Submit your Assignment to the following email address
“abdybunafc@gmaile.com”
7. For more information visit the telegram channel “ **<https://t.me/COA20>** ”

Questions

1. Why we need Special hardware Called Interface unit components between the CPU and Peripherals
2. Explain Three Types of mapping in Cache Organization
3. Explain type of Asynchronous Data Transfer Methods
4. Discuss the three different data transfer modes between input output devices and CPU?

MEMORY ORGANIZATION

Chapter

4

MEMORY ORGANIZATION

- Memory Hierarchy
- Main Memory
- Auxiliary Memory
- Associative Memory
- Cache Memory
- Virtual Memory

Memory Hierarchy

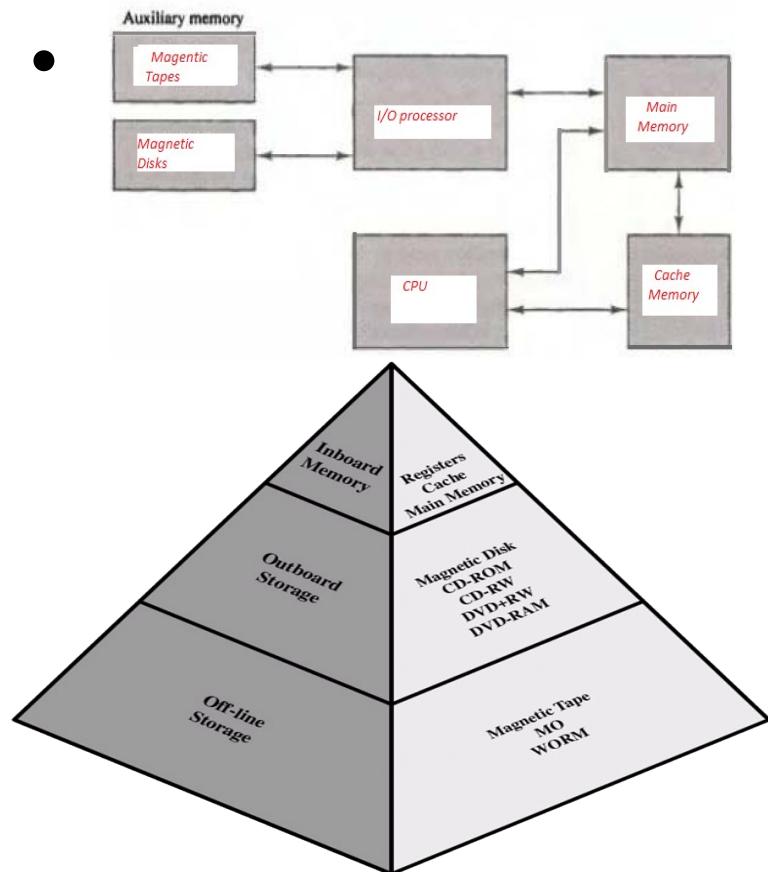
- The **memory unit** Is an **Essential Component** in any **Digital Computer** since It Is needed for **Storing Programs** and **Data**.
- There is just not **enough space** in one memory unit to accommodate all the programs used in a typical computer
- **Most computer users** gather and continue to gather large amounts of **data-processing software**.
- Not all gathered information is needed by the **processor** at the **same time**.
- it is more economical to use **low-cost storage devices** to serve as a **backup** for storing the information that is **not currently used** by the **CPU**.

Memory Hierarchy

- The **memory unit** that communicates **directly** with the **CPU** is called the **Main Memory**.
- Devices that **provide backup** storage are called **Auxiliary Memory**.
- The most common auxiliary memory devices used in computer systems are **magnetic disks** and **tapes**.
- They are used for storing **system programs**, **large data files**, and other **backup information**.
- **Only programs** and **data** currently needed by the **processor** reside in **main memory**.
- All other information Is stored in **auxiliary memory** and **transferred** to **main memory** when needed.

Memory Hierarchy

- At the **bottom** of the **hierarchy** are the **relatively slow magnetic tapes** used to store **removable files**.
- Next are the **magnetic disks** used as **backup storage**.
- The **main memory** occupies a **central position** by being able to **communicate directly** with the **CPU** and with **auxiliary** memory devices through an I/O processor.
- When **programs not exist** in **main memory** are needed by the **CPU**, they are brought in from **auxiliary memory**
- **Programs not currently** needed in **main memory** are transferred into **auxiliary memory** to provide space for **currently used programs** and **data**



Cache

- A special **very-high speed memory**
- used to **increase** the **speed** of **processing**. How ???
- by making **current programs** and **data available** to the **CPU** at a **rapid rate**.
- The **cache memory** is employed in computer systems to **compensate** for the speed differential between **main memory access time** and **processor logic(CPU)**.
- **CPU logic** is usually **faster** than **main memory access time**, with the result that processing speed is **limited primarily** by the speed of **main memory**.
- A technique used to compensate for the mismatch in **operating speeds** is to employ an **extremely fast, small cache** between the **CPU** and **main memory** whose access time Is close to processor logic clock cycle time.

Cache

- The **cache** is used for **storing segments** of programs **currently** being **executed** in the **CPU** and **temporary data** frequently needed in the **present calculations**.
- The **reason** for having **two or three levels** of **memory** hierarchy is **economics**.
- As the **storage capacity** of the **memory increases**, the **cost** per bit for storing binary information **decreases** and the **access time** of the **memory** becomes **longer**.
- The auxiliary memory has a **large storage capacity**, is **relatively inexpensive**, but has **low access speed** compared to **main memory**.
- The **cache** memory is **very small**, **relatively expensive**, and has **very high access speed**.
- The **overall goal** of using a **memory hierarchy** is to obtain the **highest-possible average access speed** while **minimizing** the total cost of the **entire memory system**.

Cache

- Auxiliary and cache memories are used for different purposes.
- The cache holds those parts of the program and data that are most heavily used,
- while the auxiliary memory holds those parts that are not presently used by the CPU.
- CPU has direct access to both cache and main memory but not to auxiliary memory.
- Many operating systems are designed to enable the CPU to process a number of independent programs concurrently. This concept, called multiprogramming
- suppose that a program is being executed in the CPU and an I/O transfer is required.
- The CPU initiates the I/O processor to start executing the transfer.

Cache

- For example, suppose that a program is being **executed** in the **CPU** and an **I/O transfer** is **required**.
- The **CPU** initiates the **I/O processor** to start executing the transfer.
- This **leaves** the **CPU free** to execute another program.
- In a **multiprogramming system**, when one program is waiting for **input** or **output transfer**, there is another program ready to utilize the CPU.

Main Memory

- It is a **relatively large** and **fast memory** used to **store programs** and **data** during the **computer operation**.
- The principal technology used for the main memory is based on **semiconductor integrated circuits**.
- Integrated circuit **RAM chips** are available in **two possible** operating modes, **Static** and **Dynamic**
- The **Static RAM** consists essentially of **internal flip-flops** that store the binary information. The stored information **remains valid** as long as **power** is **applied** to the unit.
- The **Dynamic RAM stores** the binary information in the form of **electric charges** that are applied to **capacitors**.
- The **stored charge** on the capacitors tend to discharge with time and the capacitors must be periodically recharged by **refreshing** the **dynamic memory**.

Main Memory

- Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge.
- The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip.
- The static RAM is easier to use and has shorter read and write cycles.
- ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.
- the ROM portion of main memory is needed for storing an initial program called a bootstrap loader.
- The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.

Main Memory

- **RAM is volatile**, its contents are destroyed when **power is turned off**.
- The contents of **ROM** remain **unchanged** after power is turned off and on again.
- Review type of ROM
 - ✓ **Masked ROM(MROM)**
 - ✓ **Programmable ROM(PROM)**,
 - ✓ **Erasable and programmable ROM(EPROM)** and
 - ✓ **Electrically Erasable and programmable ROM(EEPROM)**

RAM and ROM Chips

- A **RAM chip** is better suited for **communication** with the **CPU** if it has one or more control inputs that select the chip only when needed.
- Another common feature is a **bidirectional data bus** that allows the **transfer** of **data** either **from memory** to **CPU** during a **read operation**, or **from CPU** to **memory** during a **write operation**.
- A bidirectional bus can be constructed with three-state buffers.
- a signal equivalent to **logic 1**, a signal equivalent to **logic 0**, or a high impedance state. The logic 1 and 0 are normal digital signals. The high impedance state behaves like an **open circuit**, which means that the output does **not carry a signal** and has no logic significance

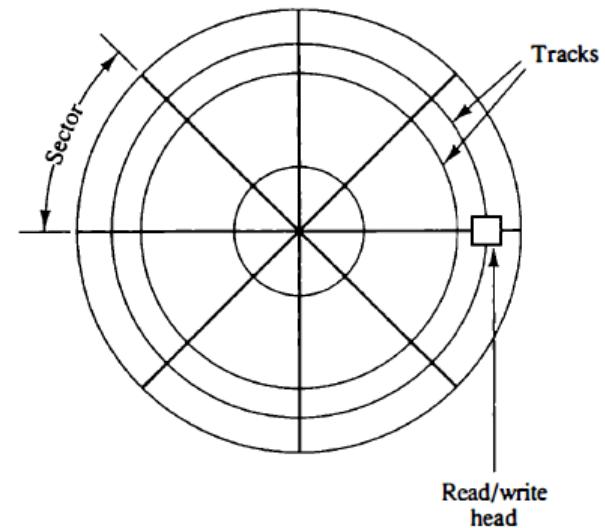
Auxiliary Memory

- The **most common auxiliary** memory devices used in computer systems are **magnetic disks** and **tapes**.
- The **important characteristics** of any device are its **access mode, access time, transfer rate, capacity, and cost**.
- The **average time** required to reach a **storage location** in memory and **obtain** its contents is called the **access time**.
- **auxiliary storage** is organized in records or blocks.
- A record is a specified number of characters or words. Reading or writing is always done on entire records.

Magnetic Disc

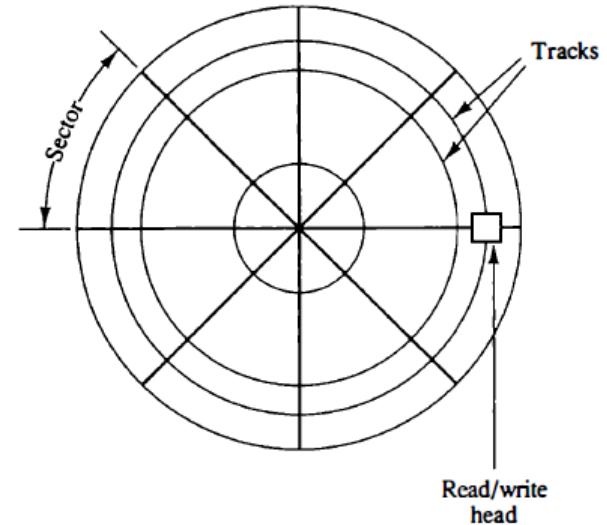
- The subdivision of one disk surface into tracks and sectors is shown in Fig.
- A **magnetic disk** is a circular plate constructed of metal or plastic coated with magnetized material.
- Bits are stored in the magnetized surface in spots along concentric circles called **tracks**.
- The **tracks** are commonly divided into sections called **sectors**.
- In most systems, the minimum quantity of information which can be transferred is a sector.
- Some units use a single read/write head for each disk surface

- **Magnetic Disc**



Magnetic Disc

- In other **Disk systems**, separate read/write heads are provided for each track in each surface.
- The address bits can then select a particular track electronically through a decoder circuit.
- This type of unit is **more expensive** and is found only in very large computer systems.
- Permanent timing tracks are used in disks to synchronize the bits and recognize the sectors.
- **Magnetic Disc**



Magnetic Tape

- Magnetic tape units can be stopped, started to move forward or in reverse, or can be rewound.
- However, they cannot be started or stopped fast enough between individual characters.
- For this reason, information is recorded in blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped. The tape starts moving while in a gap and attains its constant

Associative Memory

- Many **data-processing** applications require the **search** of items in a table stored in memory.
- An **assembler program** searches the symbol address table in order to extract the symbol's binary equivalent.
- An **account number** may be searched in a file to determine the **holder's name** and **account status**.
- The established way to search a table is to store all items where they can be addressed in sequence.
- The search procedure is a strategy for choosing a sequence of addresses, reading the content of memory at each address, and comparing the information read with the item being searched until a match occurs.
- Many search algorithms have been developed to minimize the number of accesses while searching for an item in a **random** or **sequential access memory**.

Associative Memory

- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- A memory unit accessed by **content** is called an **associative memory** or **content addressable memory (CAM)**.
- This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- When a word is written in an associative memory, no address is given. The memory is capable of finding an empty unused location to store the word.
- When a word is to be read from an associative memory, the content of the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading.

Associative Memory

- An **associative memory** is **more expensive** than a **random access memory** because each cell must have storage capability as well as logic circuits for **matching its content** with an external argument. For this reason, associative memories are used in applications where the search time is **very critical** and must be **very short**.

Cache Memory

- Analysis of a large number of typical programs has shown that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference locality of reference .
- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory.
- It is placed between the CPU and main memory.
- The cache memory access time is less than the access time of main memory by a factor of 5 to 10.
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

Cache Memory

- The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will approach the access time of the cache.
- Although the cache is only a small fraction of the size of main memory, a large fraction of memory requests will be found in the fast cache memory because of the locality of reference property of programs.

The basic operation of the cache

- When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory.
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- A block of words containing the one just accessed is then transferred from main memory to cache memory.
- The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed.
- In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory.
- The performance of cache memory is frequently measured in terms of a quantity called **hit ratio** .

The basic operation of the cache

- The performance of cache memory is frequently measured in terms of a quantity called hit ratio .
- When the CPU refers to memory and finds the word in cache, it is said to produce a hit .
- If the word is not found in cache, it is in main memory and it counts as a miss .
- The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.
- The hit ratio is best measured experimentally by running representative programs in the computer and measuring the number of hits and misses during a given interval of time.

The basic operation of the cache

- For example, a computer with cache access time of **100 ns**, a main memory access time of **1000 ns**, and a hit **ratio of 0.9** produces an average access time of 200 ns.
- This is a considerable improvement over a similar computer without a cache memory, whose access time is 1000 ns.
- The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is referred to as a mapping process

Three types of mapping

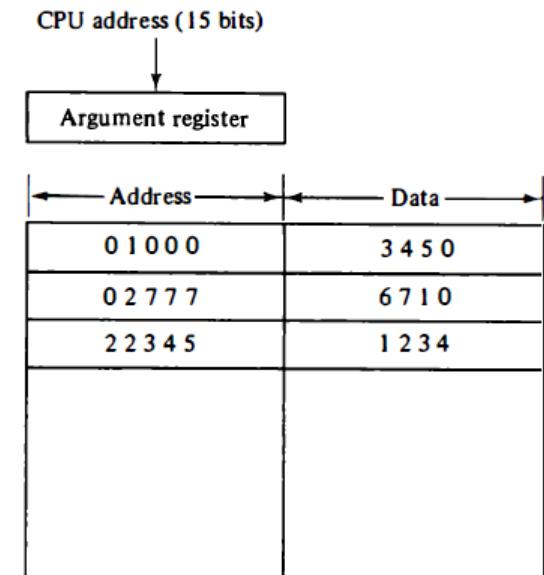
- Assignment for the next class
- Three Types of mapping in Cache Organization
 1. Associative mapping
 2. Direct mapping
 3. Set-associative mapping and

Associative Mapping

- The **basic characteristic** of **cache memory** is its **fast access time**.
- **very little** or no time must be wasted when **searching** for words in the cache.
- The **transformation** of data from **main memory** to cache memory is referred to as a **mapping process**.
- The CPU communicates with both memories(Main and Coach).
- It **first sends** a address to **cache**. If there is a **hit**, the CPU accepts the data from cache.
- If there is a **miss**, the CPU reads the word from **main memory** and the word is then transferred to **cache**.

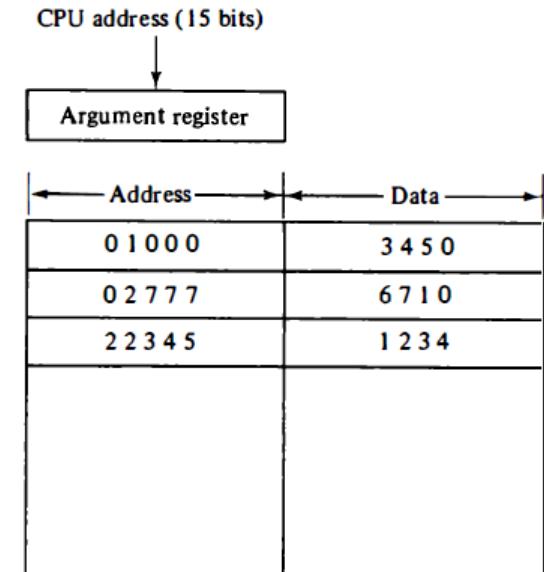
Associative Mapping

- The **fastest** and **most flexible** cache organization uses an **associative memory**.
 - The associative memory stores both the **address** and **content (data)** of the memory word.
 - This permits **any location** in cache to **store** any **word** from **main memory**.
 - The **diagram shows three words** presently stored in the **cache**.
 - The **address value** of 15 bits is shown as a **five-digit octal number** and its corresponding 12-bit word is shown as a four-digit octal number.
 - A **CPU address** of 15 bits is placed in the argument register and the **associative memory** is searched for a matching address.
 - If the address is found, the corresponding **12-bit data** is read and sent to the CPU.
- **Associative mapping cache**



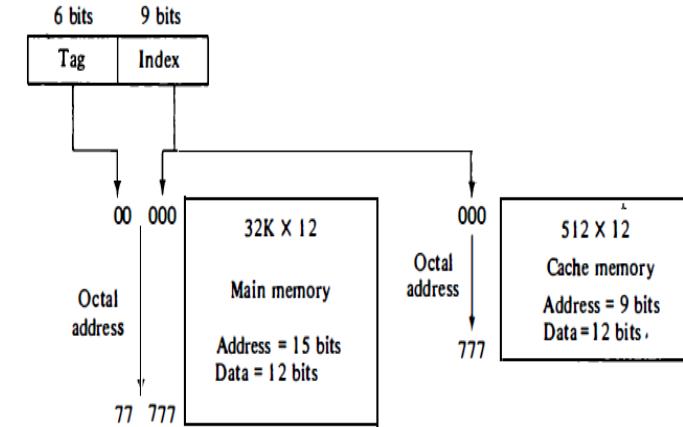
Associative Mapping

- If no match occurs, the main memory is accessed for the word.
- The address--data pair is then transferred to the associative cache memory.
- If the cache is full, an address--data pair must be displaced to make room for a pair that is needed and not presently in the cache.
- The decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache.
- A simple procedure is to replace cells of the cache in round-robin order whenever a new word is requested from main memory.
- This constitutes a first-in first-out (FIFO) replacement policy.
- Associative mapping cache (all numbers in octal).



Direct Mapping

- **Associative memories** are **expensive compared** to random-access memories because of the **added logic associated** with each cell
- The **Main memory** address of **15 bits** is divided into two fields.
- The **nine least significant** bits constitute the **index field** and the remaining **six bits** form the **tag field**.
- The figure shows that **main memory** needs an address that includes both the **tag** and the **index** bits.
- The **number** of bits in the **index field** is equal to the number of **address** bits required to **access** the **cache memory**.
- Addressing relationships b/n main and cache memories.



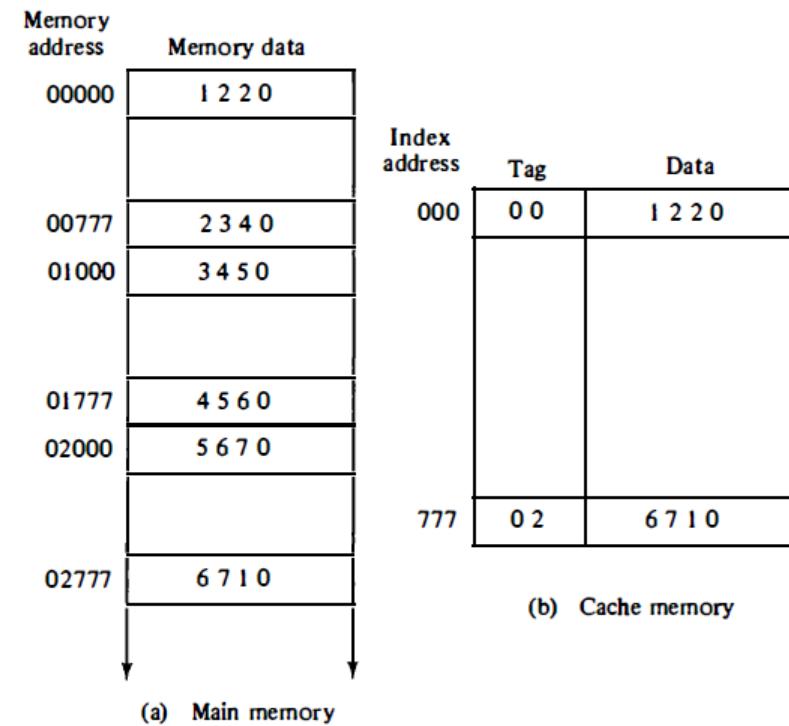
Direct Mapping

- The **Direct mapping** cache organization uses the **Tag bit (6 bit)** address to access the **main memory** and **index(9 bit) to access the cache**.
- The **internal organization** of the words in the **cache memory** is as shown in the diagram .
- **Each word** in cache consists of the **data word** and its **associated tag**.
- When a new word is first brought into the cache, the **tag bits** are stored alongside the **data bits**.
- **When the CPU generates a memory request**, the **index field** is used for the address to access the **cache**.
- the **tag field** of the **CPU address** is **compared** with the **tag** in the **word read** from the **cache**. If the two **tags match**, there is a hit and the desired data word is in cache.
- If there is **no match**, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value.
- The disadvantage of direct mapping is that the hit ratio can drop considerably if **two or more** words whose **addresses** have **the same index** but **different tags** are accessed repeatedly.
- Coach memory Organization .

Index address	Tag	Data
000	0 0	1 2 2 0
777	0 2	6 7 1 0

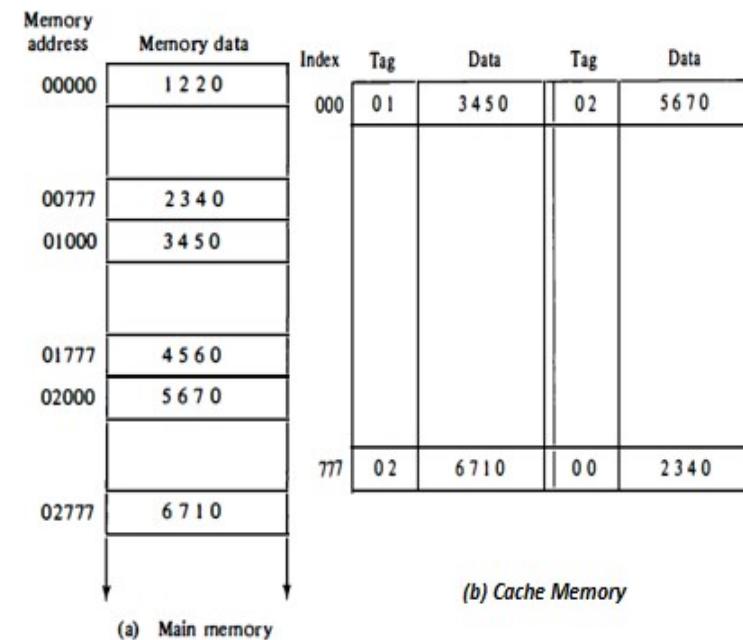
Direct Mapping

- To see how the **direct-mapping organization** operates, consider the numerical example shown in the diagram.
 - The **word** at **address zero** is presently stored in the cache (**index = 000**, **tag = 00**, **data = 1220**).
 - Suppose that the **CPU** now wants to access the word at address **02000**. The index address is **000**,
 - So it is used to **access** the **cache**. The two tags are then compared. The cache tag is **00** but the address tag is **02**, which does not produce a match.
 - Therefore, the main **memory** is accessed and the data word **5670** is transferred to the CPU.
 - The cache word at index address **000** is then replaced with a tag of **02** and data of **5670**.
- Direct mapping cache organization

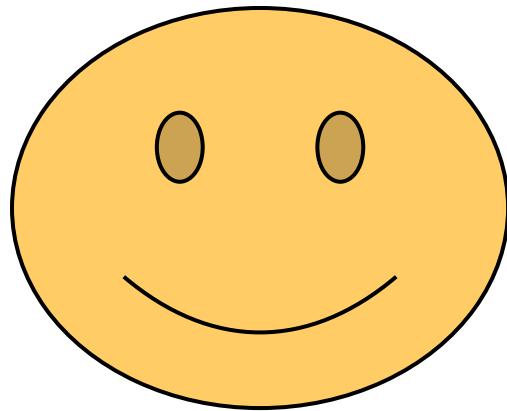


Set-Associative Mapping

- A third type of cache organization, called set-associative mapping, is **an improvement** over the **direct mapping organization** in that each **word** of cache can store **two or more words** of **memory** under the **same index address**.
- Each **data word** is **stored together** with its **tag** and the number of **tag-data** items in one word of cache(index) is said to form a **set**
- Each index address refers in to two data words and their associated tags.
- The words stored at addresses **01000** and **02000** of **main memory** are stored in cache memory at **index address 000**.
- Similarly, the words at addresses **02777** and **00777** are stored in cache at **index address 777**.
- set - associative mapping cache with main memory



Thank You!!!



Digital Logic Circuits

Chapter

Chapt. 5

Table of Content

- 5.1 Review of logic gates and Boolean algebra
- 5.2 Combinational circuits (half adder, full adder)
- 5.3 Flip flops
- 5.4 Sequential circuits, Integrated circuits
- 5.5 Decoders
- 5.6 Multiplexers
- 5.7 Registers, Shift registers
- 5.8 Binary counters

Review of logic gates and Boolean algebra

- In the hierarchical structuring of a computer, the **logic gate level** is above the **device level**.
- **logical elements** are primary components of the **computer hardware**.
- A **gate** is an **electronic circuit** that produces an **output signal** that is a simple **Boolean operation** on its input signals.
- The Basic gates used in digital logic are **AND**, **OR**, **NOT**, **NAND**, and **NOR**. We have also **XOR** gate for defining Boolean algebra with other gates.

Logic gates (elements)

- **logical elements** are primary components of the **computer hardware**.
- A **gate** is an **electronic circuit** that produces an **output signal** that is a simple **Boolean operation** on its input signals.
- The Basic gates used in digital logic are **AND**, **OR**, **NOT**, **NAND**, and **NOR**. We have also **XOR** gate for defining Boolean algebra with other gates.

Logic gates (elements)

- Each gate is defined in three ways: **graphical symbols**, **algebraic notation**, and **truth table**.
- A **truth table** is a table that shows the output of a logical function for all possible combinations of input values.
- Each gate has **one** or **two** or **more inputs** and only **one-output signals**.
- The signals are either **0** or **1**. When the values at the input are **changed**, the correct output signal appear almost instantaneously.

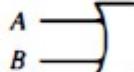
And Gate

- An **AND gate** is an electronic circuit that generates an **output signal of 1** only if **all input** signals are also **1**.
- Operation is termed as **logical multiplication** and is represented by the symbol “.”

Name	Graphic symbol	Algebraic function	Truth table															
AND		$x = A \cdot B$ <p>or</p> $x = AB$	<table border="1"><thead><tr><th>A</th><th>B</th><th>x</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></tbody></table>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	0																
0	1	0																
1	0	0																
1	1	1																

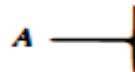
OR Gate

- An **OR gate** is an electronic circuit that generates an **output signal of 1** if **any** of the **input signals** is **1**.
- Operation is termed as **logical Addition** and is represented using the **symbol +**
- **A+B** is read as **A OR B**.

Name	Graphic symbol	Algebraic function	Truth table															
OR		$x = A + B$	<table border="1"><thead><tr><th>A</th><th>B</th><th>x</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></tbody></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

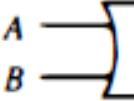
NOT Gate

- NOT gate is an electronic circuit that generates an output signal which is the reverse of the input signal.
- A NOT gate is also known as an inverter or Complement because it inverts the input.
- A NOT gate always has a single input.
- Symbols used are $-$, \sim , $'$.

Name	Graphic symbol	Algebraic function	Truth table
Inverter	 A ————— x	$x = A'$	$\begin{array}{c cc} A & x \\ \hline 0 & 1 \\ 1 & 0 \end{array}$

NOR GATE (Combines OR and NOT gates)

- A NOR gate is an electronic circuit that generates an output signal of 1 when all input signals are 0 and it will be a 0 if any input represents a 1.
- NOR gate is a complemented OR gate.

Name	Graphic symbol	Algebraic function	Truth table															
NOR	 $x = (A + B)'$	$x = (A + B)'$	<table border="1"><thead><tr><th>A</th><th>B</th><th>x</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	0
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

NAND GATE (Combines AND & NOT)

- A **NAND gate** is an electronic circuit that generates an **output signal** of **1** if any one of the input is a **0** and will be a **0** when all input signals are **1**.
- NAND gate is a complemented AND gate.

Name	Graphic symbol	Algebraic function	Truth table															
NAND	 $x = (AB)'$	$x = (AB)'$	<table border="1"><thead><tr><th>A</th><th>B</th><th>x</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	x	0	0	1	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	1																
0	1	1																
1	0	1																
1	1	0																

Exclusive-OR gate(XOR Gate)

- **XOR gate** is an electronic circuit that generates an **output signal** of **0** if **both** the **inputs** are **same**.
- The **symbol** \oplus is used to represent **XOR** operation in Boolean expression.

Name	Graphic symbol	Algebraic function	Truth table															
Exclusive-OR (XOR)		$x = A \oplus B$ <p>or</p> $x = A'B + AB'$	<table border="1"><thead><tr><th>A</th><th>B</th><th>x</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></tbody></table>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	0
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

Buffer

- A **buffer** does not produce any **particular logic** function since the binary value of the **output** is the same as the binary value of the **input**.
- This circuit is used merely for **power amplification**.
- For example, a buffer that uses 3 volts for binary 1 will produce an output of 3 volts when its input is 3 volts

Name	Graphic symbol	Algebraic function	Truth table
Buffer	 A ————— x x = A		A x --- --- 0 0 1 1

exclusive-NOR gates

- Exclusive -NOR gates a complement of Expulsive -OR (XOR)
- **Exclusive -NOR gate** is an electronic circuit that generates an **output signal of 1 if both the inputs are same**

Name	Graphic symbol	Algebraic function	Truth table															
Exclusive-NOR or equivalence		$x = (A \oplus B)'$ or $x = A'B' + AB$	<table border="1"><thead><tr><th>A</th><th>B</th><th>x</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></tbody></table>	A	B	x	0	0	1	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Operator Precedence

- The expression is read from **left** to **right**.
- Expressions in **brackets** are **evaluated first**.
- All **NOT** operations are **performed next**.
- All **AND** operations are performed after that.
- All **OR** operations are performed at the end.

- Boolean Algebra

Basic Identities of Boolean Algebra

$$(1) \quad x + 0 = x$$

$$(2) \quad x \cdot 0 = 0$$

$$(3) \quad x + 1 = 1$$

$$(4) \quad x \cdot 1 = x$$

$$(5) \quad x + x = x$$

$$(6) \quad x \cdot x = x$$

$$(7) \quad x + x' = 1$$

$$(8) \quad x \cdot x' = 0$$

$$(9) \quad x + y = y + x$$

$$(10) \quad xy = yx$$

$$(11) \quad x + (y + z) = (x + y) + z$$

$$(12) \quad x(yz) = (xy)z$$

$$(13) \quad x(y + z) = xy + xz$$

$$(14) \quad x + yx = (x + y)(x + z)$$

$$(15) \quad (x + y)' = x'y'$$

$$(16) \quad (xy)' = x' + y'$$

$$(17) \quad (x')' = x$$

- Identities 15 and 16 are called DeMorgan's theorems

Combinational and Sequential circuits

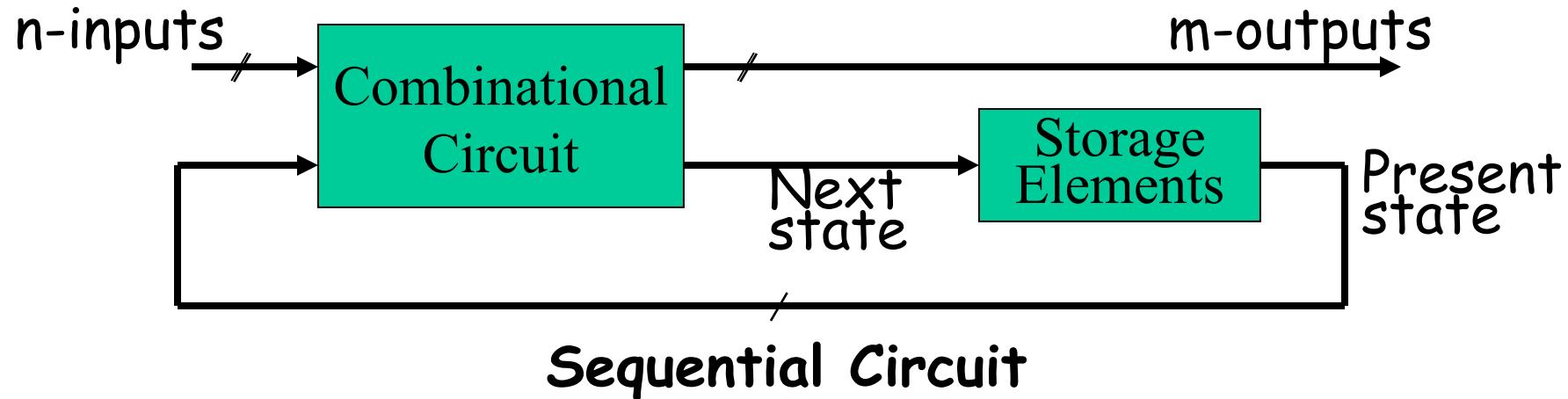
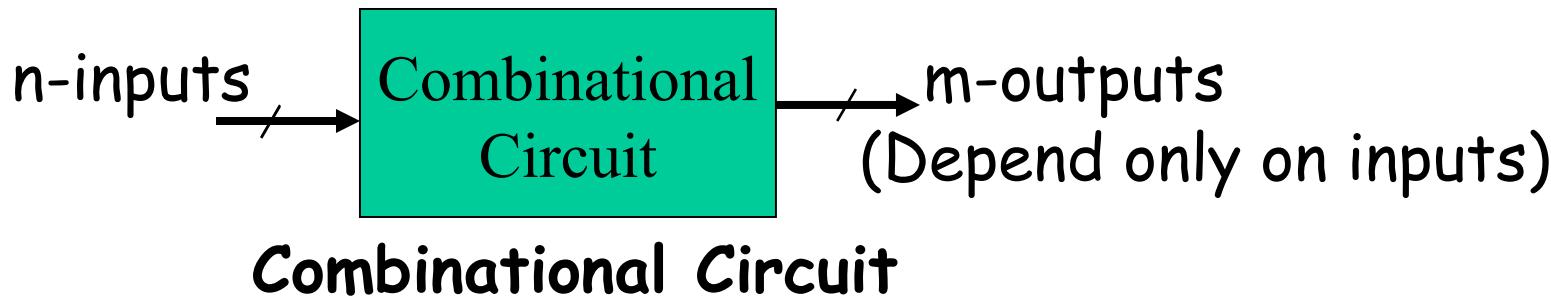
Combinational Logic

- **Logic circuits** for digital systems may be **combinational** or **sequential**.
- Digital systems may be divided into two broad categories:
 - **combinational logic**
 - where the outputs are determined only by the current states of the inputs
 - Has no memory
 - Output depends only on current input
 - Example Full Adder
 - **sequential logic**
 - where the outputs are determined not only by the current inputs but also by the sequence of inputs that led to the current state
 - Output depends not only on current input but also on past input values, e.g., design a counter , Flip Flop
 - Need some type of memory to remember the past input values

Combinational vs. Sequential Circuits

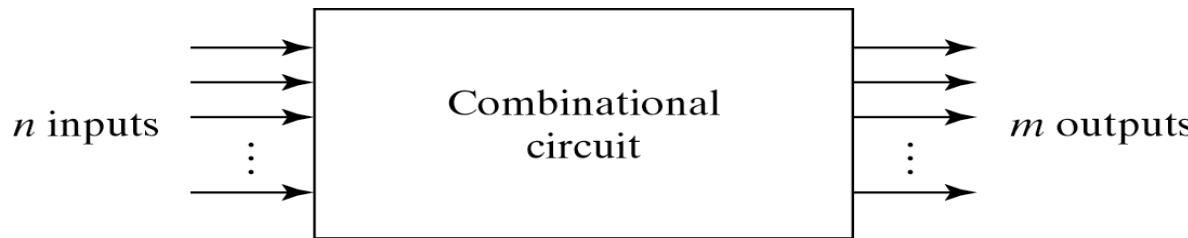
- Combinational circuits are **memory-less**. Thus, the output value depends ONLY on the current input values.
- Sequential circuits consist of combinational logic as well as **memory elements** (used to store certain circuit states). Outputs depend on BOTH current input values and previous input values (kept in the storage elements).

Combinational vs. Sequential Circuits



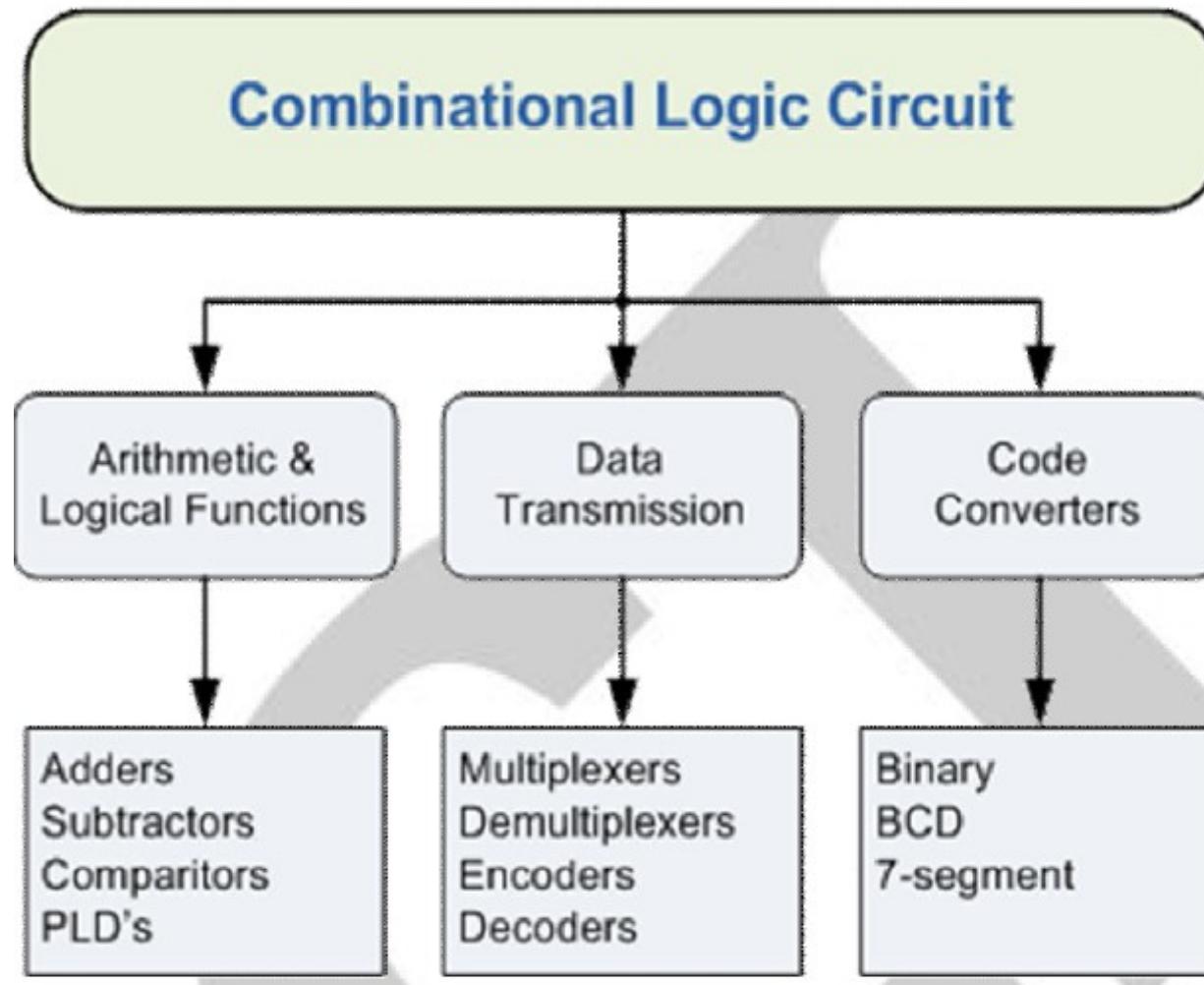
Combinational Circuits

- A combinational circuit is a connected arrangement of logic gates with a set of **inputs** and **outputs**.
- A combinational circuit consists of input variables, logic gates, and output variables.
- Hence, a combinational circuit can be described by:
 1. A truth table that lists the output values for each combination of the input variables, or
 2. m Boolean functions, one for each output variable.



Block Diagram of Combinational Circuit

Classification of Combinational Logic



To Design Circuits

- The design of combinational circuits starts from the **verbal outline** of the **problem** and ends in a **logic circuit diagram**.
- The procedure involves the following **five steps**
 1. The **problem is stated**.
 2. The **input** and **output variables** are assigned letter symbols(**Block Diagram**).
 3. The **truth table** that defines the **relationship** between **inputs** and **outputs** is derived.
 4. The **simplified Boolean functions** for each output are obtained.
 5. The **logic diagram is drawn**.

Half-Adder and Full-Adder

Half-Adder

- A combinational circuit that performs the **arithmetic addition** of **two bits** is called a **half-adder**.
- The most basic digital arithmetic circuit is the addition of two binary digits.
- The half adder circuit is designed to add two single bit binary number A and B.
- It is the basic building block for addition of two single bit numbers.
- This circuit has two outputs carry and sum.



Block diagram

Design Half adder

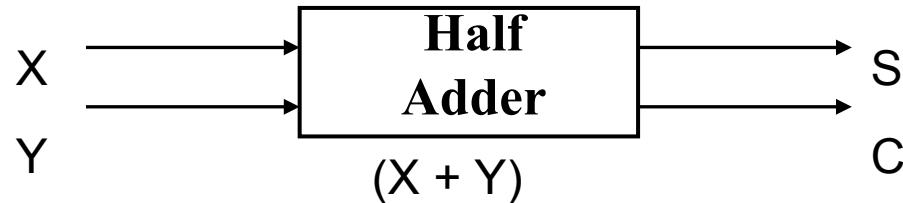
Design procedure

Step 1:- State Problem

Build a Half Adder to add two bits

Step 2 :-Determine and label the inputs & outputs of circuit(Block Diagram).

Two inputs and **two outputs labeled**, as follows:



We assign symbols x and y to the two input variables, and S (for sum) and C(carry)

Step 3 Draw truth table

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Design Half Adder

Step 4 Obtain simplified Boolean function.

The C output is 0 unless both inputs are 1 (AND gate.). The S output 0 if both x and y have the same value (exclusive-OR gate). The Boolean functions for the two outputs can be obtained directly from the truth table:

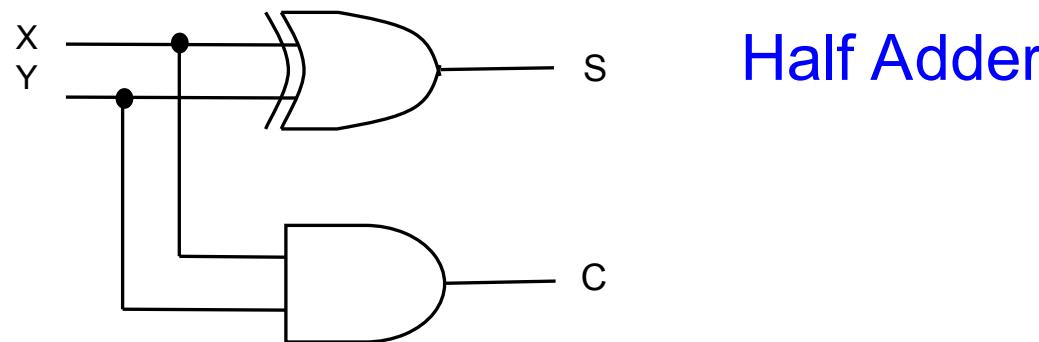
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = X \cdot Y$$

$$S = X' \cdot Y + X \cdot Y' = X \oplus Y$$

Step 5 Draw logic diagram.

- The logic diagram is shown in Fig. below. It consists of an exclusive-OR gate and an AND gate.



Design full Adder

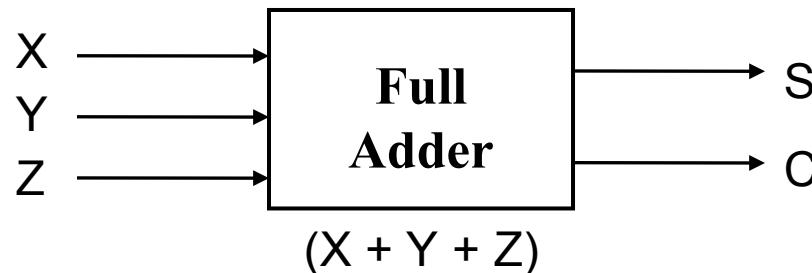
- Full adder is developed to overcome the **drawback of Half Adder circuit.**
- A **full-adder** is a combinational circuit that forms the arithmetic **sum of three input bits**.

Step 1:- State Problem

Build a full adder to add three bits

Step 2 :-Determine and label the inputs & outputs of circuit(Block Diagram).

Three inputs and two outputs labelled, as follows



- We assign symbols **x**, **y** and **z** to the **three input** variables, and **S** (for sum) and **C**(carry) for two output

Full Adder

- Step3:- Draw truth table

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step4 : Simplification(Using k-map)

From K-map simplification for **S(Sum)** there is no group

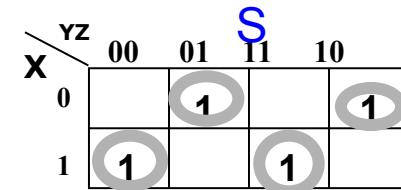
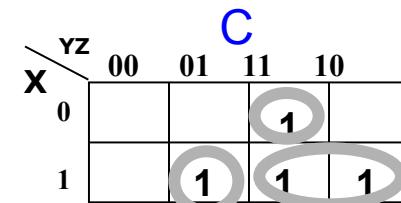
So we take each individual as group so

$$S = X'Y'Z + X'YZ' + XY'Z' + XYZ$$

$$S = X'(Y'Z + YZ') + X(Y'Z' + XY)$$

$$S = X'.(Y \oplus Z) + X.(Y \oplus Z)'$$

$$S = X \oplus (Y \oplus Z) \text{ or } (X \oplus Y) \oplus Z$$



Full Adder

- Step3:- Draw truth table

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

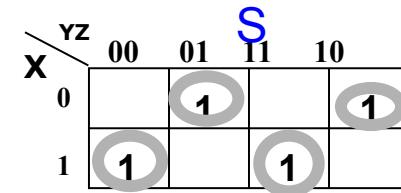
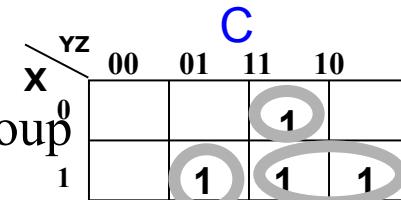
Step4 : Simplification(Using k-map)

From K-map simplification for **c(carry)** there is three group

$$C = X'YZ + XY'Z + XY$$

$$C = Z(X'Y + XY') + XY$$

$$C = Z(X \oplus Y) + XY$$



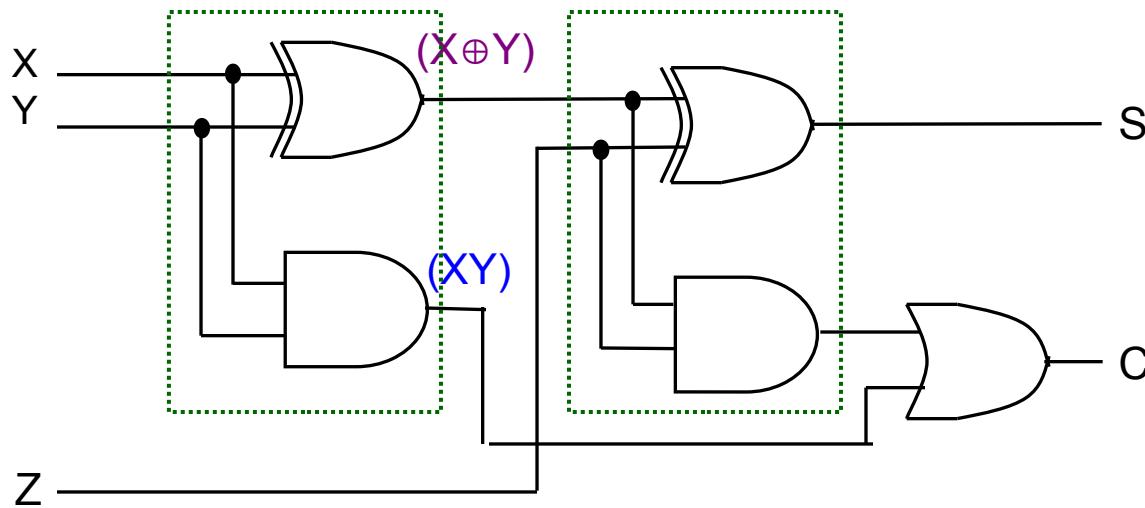
Full Adder

Step 5 Draw logic diagram

- The **logic diagram** of full adder as we discuss in the previous two slide. It consists of two exclusive-OR gate , two AND gate and one ORgat

$$C = X \cdot Y + (X \oplus Y) \cdot Z$$

$$S = (X \oplus Y) \oplus Z$$



Full Adder made from two Half-Adders (+ OR gate).

- **Half Subtractor:** The **half-subtractor** is a combinational circuit which is used to **perform subtraction** of **two bits**.
- **Full subtractor:** It subtracts **one bit** from the other by **taking previous** borrow into account and generates difference and borrow

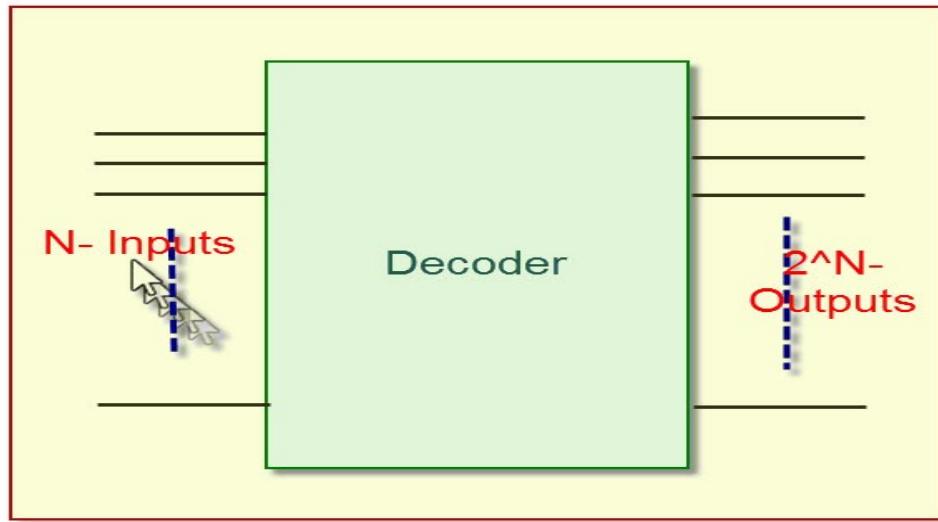
Decoders and Encoder

Decoder

- Discrete quantities of information are **represented** in **digital computers** with **binary codes**.
- A **binary code** of **n bits** is capable of representing up to **2^n distinct elements** of the coded information.
- A **decoder** is a combinational circuit that converts **binary information** from the **n coded inputs** to a maximum of **2^n unique outputs**.
- If the n-bit coded information has unused bit combinations, the decoder may have less than **2^n outputs**.
- E.g. if the number of input is **$n=3$** the number of output lines can be **$m = 2^3 = 8$** . It is also known as 1 of 8 because one output line is selected out of **8 available lines** (**3×8 Decoder**)
- E.g. **2×4** line Decoder (it is also called one four line decoder)

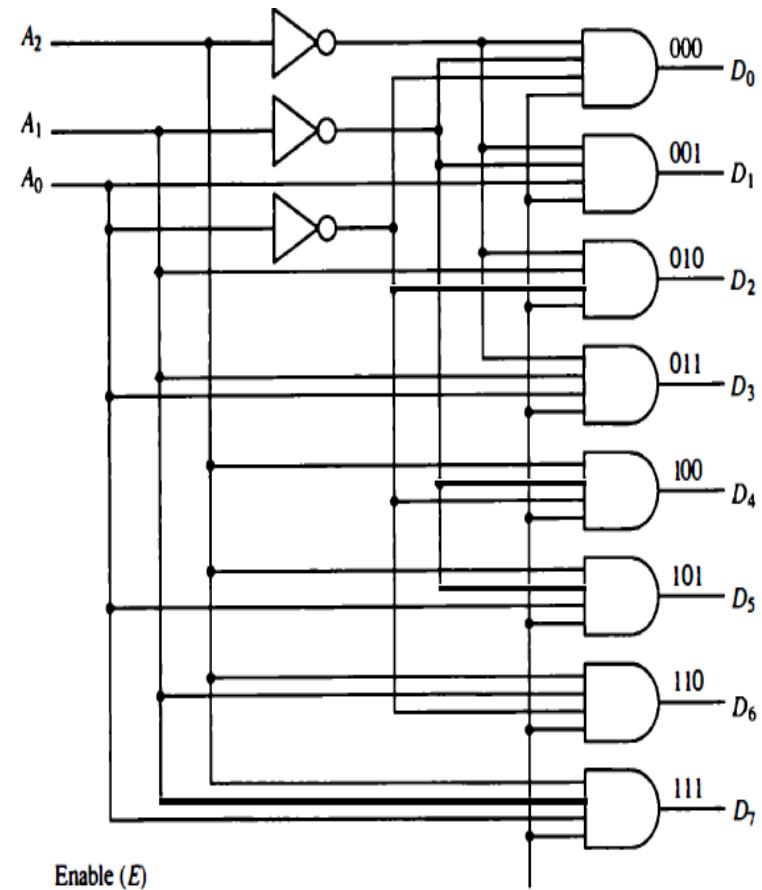
Decoder

- Binary decoders
 - Converts an **n-bit code** to a single **active output**
 - **Only one output** is active any given input
 - Can be developed using AND/OR gates
 - Can be used to implement logic circuits



3-to-8-line decoder(binary-to-octal conversion)

- The **logic diagram** of a 3-to-8-line decoder is shown in Fig.
- The three data inputs, **A₀**, **A₁** and **A₂** are **decoded** into **eight outputs**,
- **each output representing** one of the combinations of the **three binary input variables**.
- The **three inverters** provide the **complement of the inputs**, and each of the **eight AND gates** generates one of the binary combination.
- The **input variables represent** a **binary number** and the **outputs represent** the **eight digits of the octal number system**.
- a **3-to-8-line decoder** can be used for **decoding** any **3-bit code** to provide eight outputs, one for each combination of the binary code.



3-to-8-line decoder(binary-to-octal conversion)

- The **decoder** of Fig. has **one enable input, E**.

The decoder is **enabled** when E is **equal to 1** and **disabled** when E is **equal to 0**.

- The operation of the **decoder** can be clarified using the **truth table**. When the **enable input E is equal to 0**, all the outputs are equal to 0 regardless of the values of the other three data inputs.

- The **three x**'s in the table designate don't-care conditions

- When the **enable input** is **equal to 1**, the decoder operates in a **normal fashion**.

- For each possible input combination**, there are **seven outputs** that are **equal to 0** and only one that is **equal to 1**.

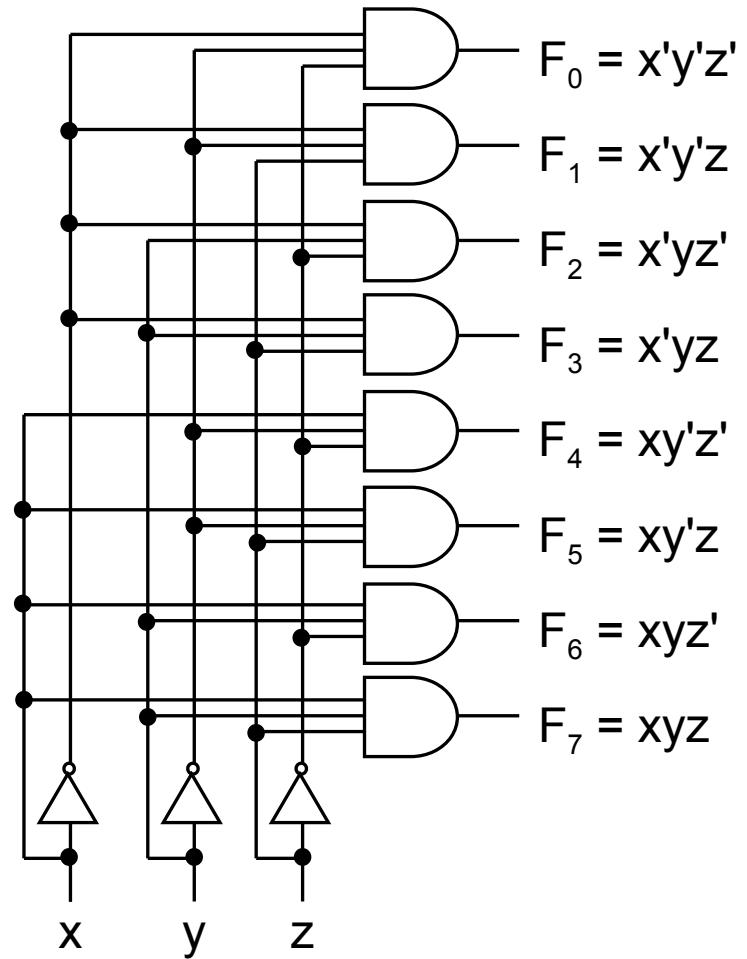
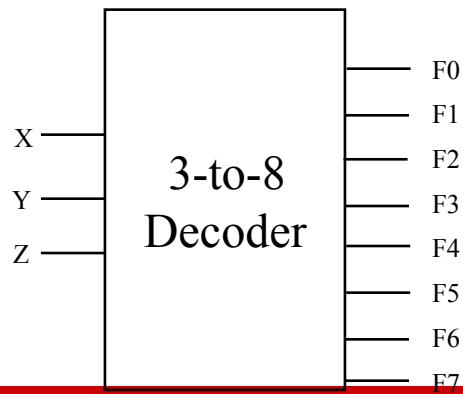
- The **output variable whose value** is equal to 1 represents the **octal number equivalent** of the **binary number** that is available in the **input data lines**.

Enable	Inputs			Outputs								
	E	A ₂	A ₁	A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	x	x	x	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0

3-to-8 Line Binary Decoder

Truth Table:

x	y	z	F ₀	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	F ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Application of decoder

- Decoders are widely used in the **memory system** of computer, where they respond to the address code input from the CPU to activate the memory storage location specified by the address code.
- Decoders are also used to **convert binary data** to a form suitable for displaying on **decimal read outs**.
- Decoders can be used to **implement combinational circuits**, **Boolean functions** etc.

Example

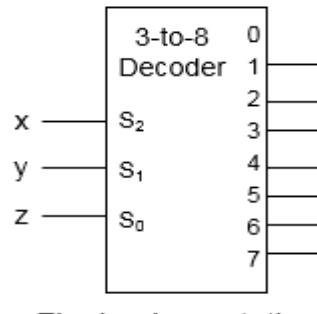
- Implement full adder with a decoder.

State1:- Problem

Build a full Adder using Decoder

Step 2 :-Determine and label the inputs & outputs of circuit(Block Diagram).

three inputs and **eight outputs** labelled, as follows:(**3 to 8 decoder**)



Step3:- Draw truth table

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Example

State4:- Simplification(Using k-map) at these case we just take the **SOP** form of **C** and **S** from truth table

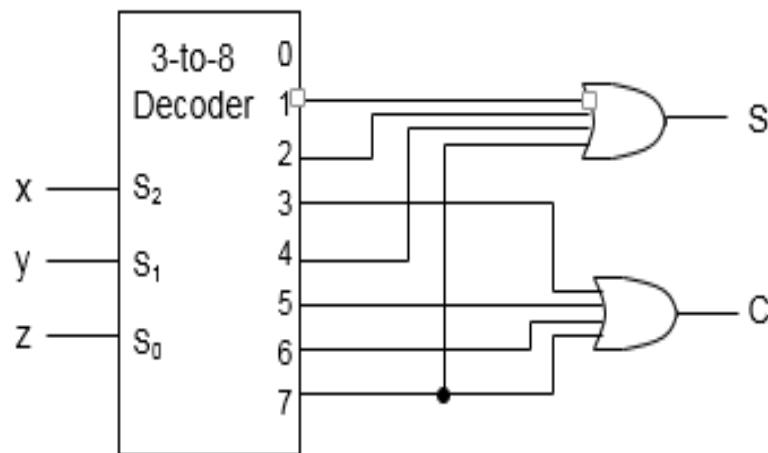
$$S(x, y, z) = \Sigma (1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma (3, 5, 6, 7)$$

Step5:-Draw logic diagram(Decoder)

- The logic diagram of **full adder** using **decoder** are explain as follow the m₃, m₅, m₆, m₇ (Minterms) used for construct carry and m₁, m₂, m₄, m₇ used for construct sum and connect by AND gate because we use Sop form

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0

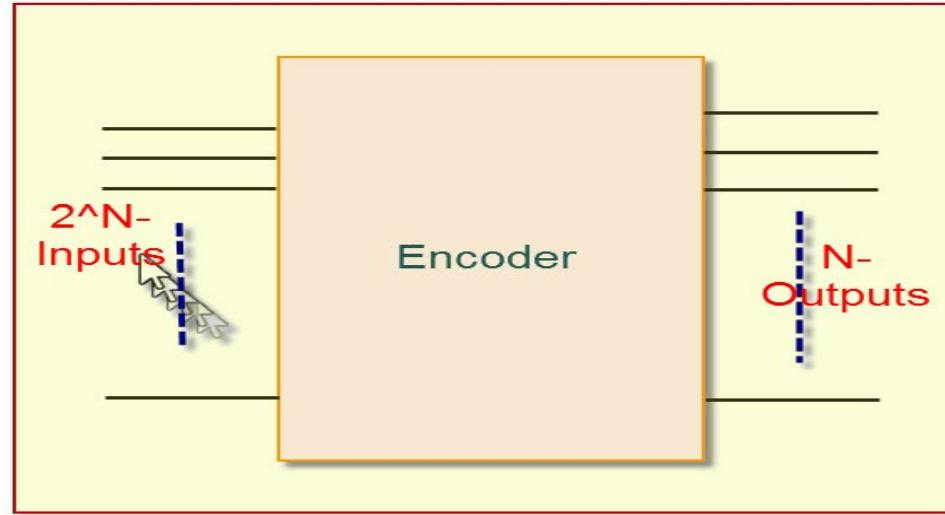


•Implementing F₁

Encoder

Encoder

- An **encoder** is a digital circuit that performs the **inverse operation of a decoder**.
- An **encoder** has **2^n (or less) input lines** and **n output lines**.
- The output lines generate the binary code corresponding to the input value.



8-to-3 Binary Encoder

- An example of an encoder is the **octal-to-binary encoder** whose truth table is given in Table
- It has **eight inputs**, one for each of the **octal digits**, and **three outputs** that generate the **corresponding binary number**.
- It is assumed that **only one input** has a value of **1** at **any given time**; otherwise the circuit has no meaning.

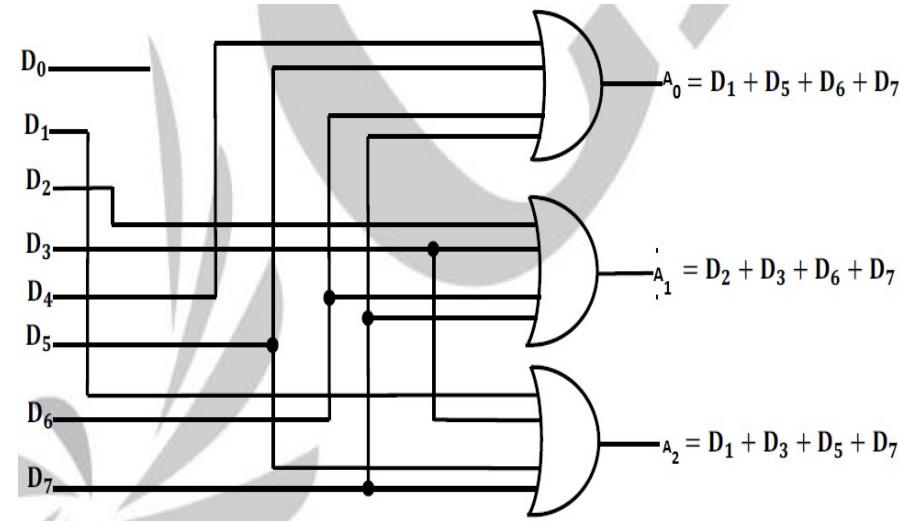
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Inputs			Outputs		
								A_2	A_1	A_0			
0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	1	0	1
0	1	0	0	0	0	0	0	0	1	1	1	0	0
1	0	0	0	0	0	0	0	1	1	1	1	1	1

Encoder

- The encoder can be implemented with **OR** gates whose inputs are determined directly from the truth table.
- Output $A_0 = 1$ if the input octal digit is 1 or 3 or 5 or 7.
- Similar conditions apply for the other two outputs.
- These conditions can be expressed by the following Boolean functions:

 - $A_0 = D_1 + D_3 + D_5 + D_7$
 - $A_1 = D_2 + D_3 + D_6 + D_7$
 - $A_2 = D_4 + D_5 + D_6 + D_7$

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1



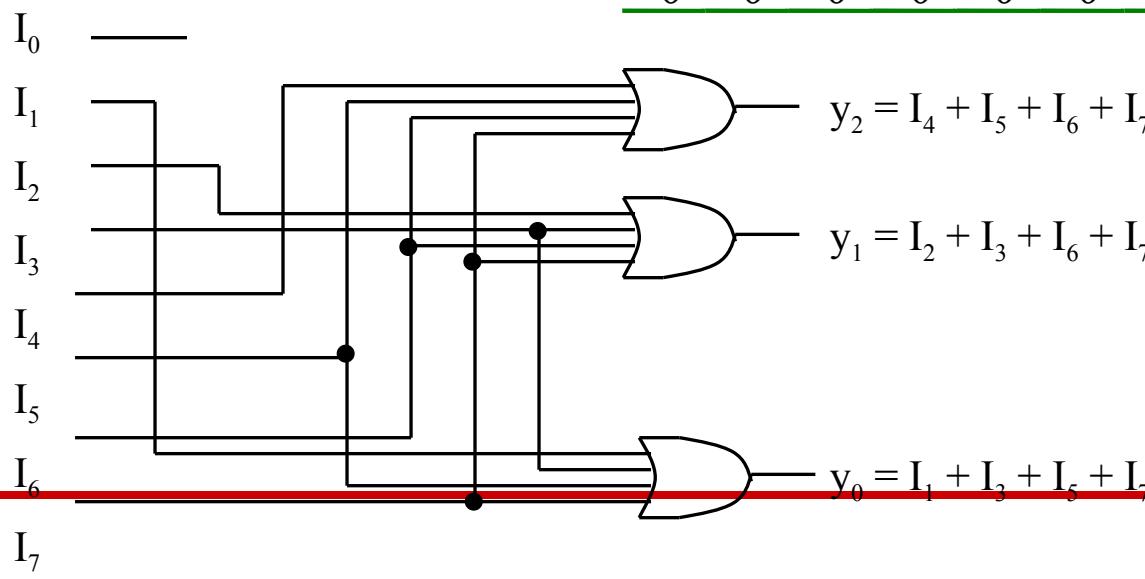
8-to-3 Binary Encoder

At any one time, only
one input line has a value of 1.

Decoders are widely used in storage
devices (e.g. memories)

Encoders all for data compression

Inputs								Outputs		
I ₀	I ₁	I ₂	I ₃	I ₄	I ₅	I ₆	I ₇	y ₂	y ₁	y ₀
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	1	0	1	0	1
0	0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	0	1	1	1	1

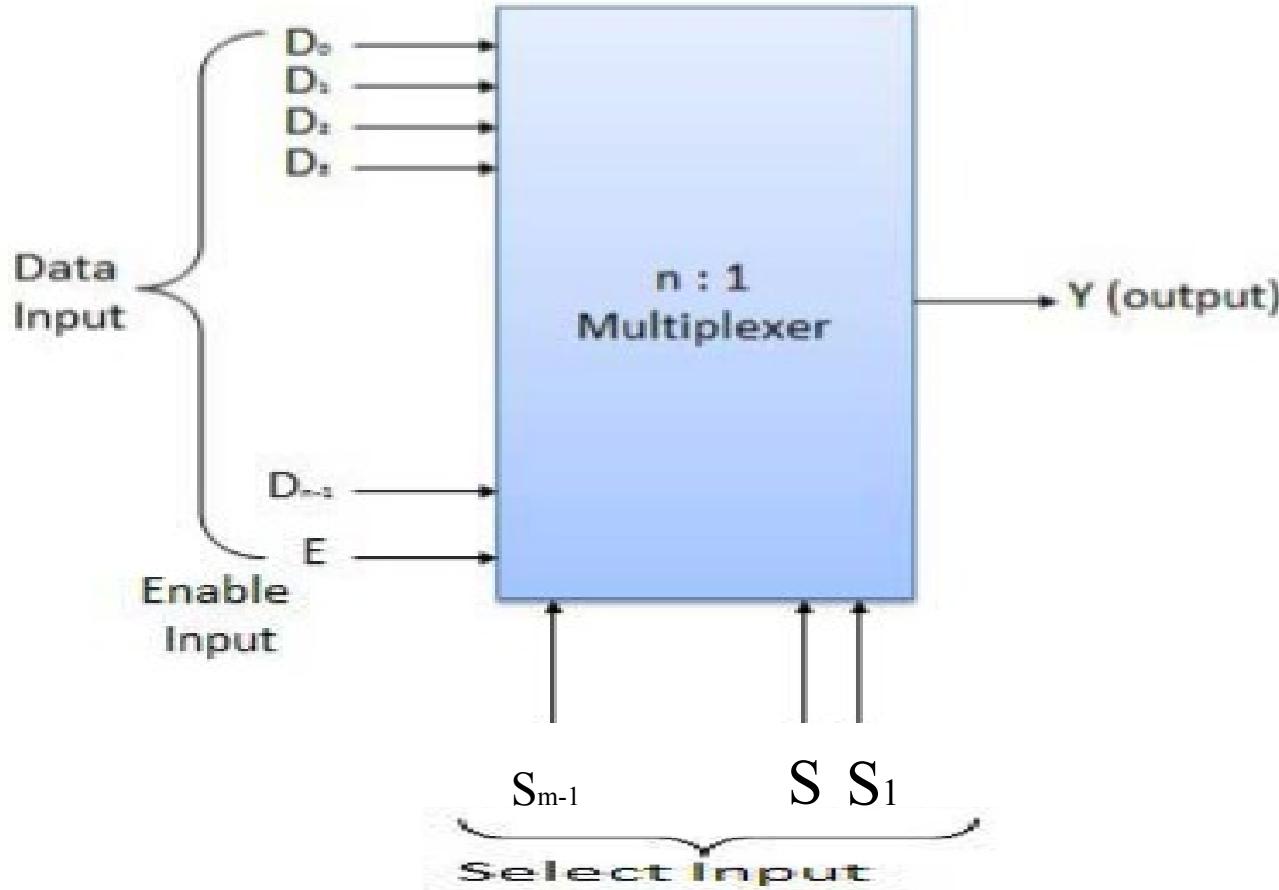


Multiplexers and Demultiplexer

multiplexer

- A **multiplexer** is a **combinational circuit** that receives binary information from one of **2^n input data lines** and directs it to a **single output line**.
- The selection of a **particular input data line** for the output is determined by a set of **selection inputs**.
- A **2^n -to-1** multiplexer has **2^n input data lines** and **n input selection lines** whose bit combinations determine which input data are selected for the output.

Block diagram

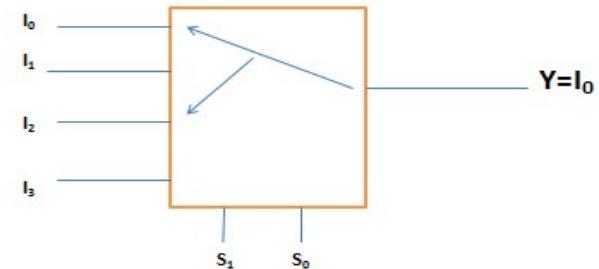


Relation b/n data input, select line

- MUX
 - It is **combinational circuit** that selects **binary information** from **many input lines** and **directs it to output line**.
 - It is simply a **DATA SELECTOR**
 - How to move a select line/selector variable inside MUX circuit ?

example

S1 S0	input selected for output	
If 0 0	I0	y=I0
If 0 1	I1	Y=I1
If 1 0	I2	Y=I2
If 1 1	I3	Y=I3



$$n=2^m \Rightarrow m = \log_2 n$$

let $n=4$

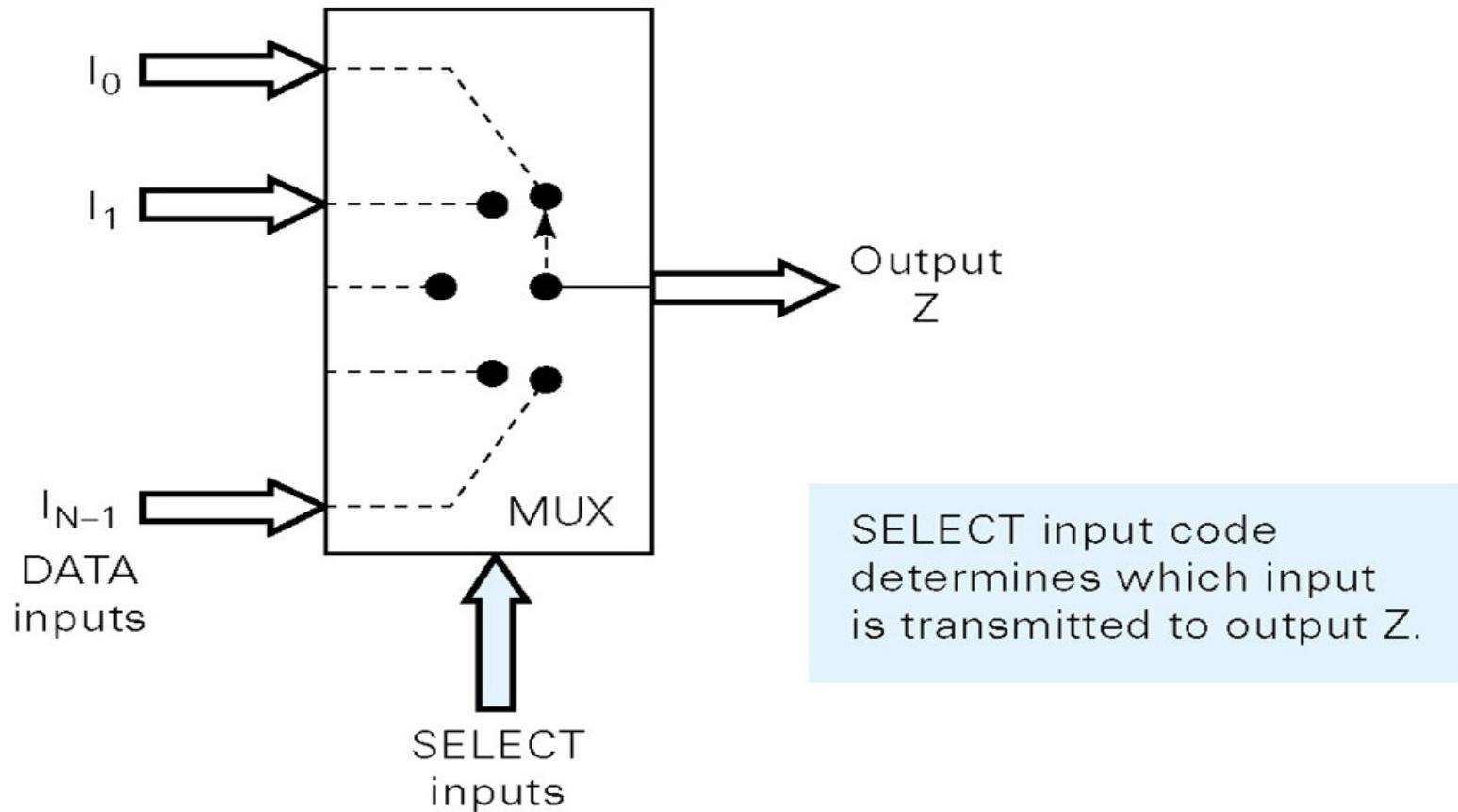
$$m = \log_2 4 \Rightarrow 2 \log_2 2 \Rightarrow 2.1 = 2$$

or

$$n=2m$$

1

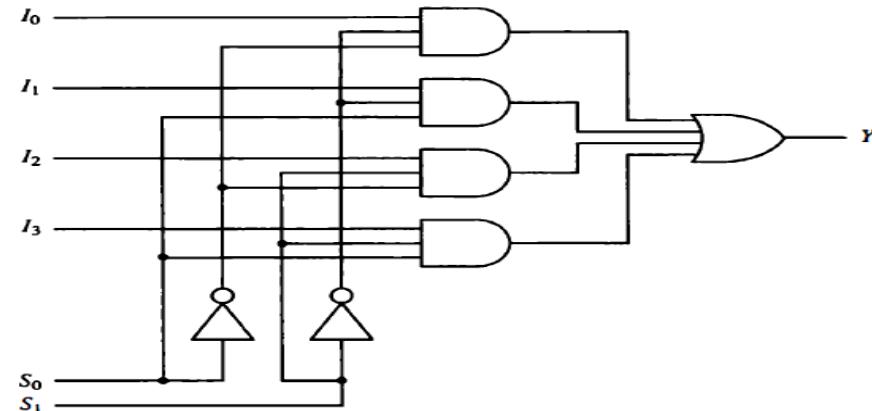
Functional diagram of MUX



4 to- 1 -line multiplexer

- The **4-to-1 line multiplexer** of Fig. has **six inputs** and **one output**.
- The function table for the multiplexer is shown in **truth table**
- The **table demonstrates** the relationship between the **four data inputs** and the **single output** as a function of the **selection** inputs S_1 and S_0
- When the **selection inputs** are **equal** to **00**, output **Y** is equal to input I_0
- When the **selection inputs** are equal to **01**, **input I_1** has a path to output **Y**, and similarly for the other two combinations.
- The multiplexer is also called **a data selector**, since it selects one of many data inputs and steers the binary information to the output.

- 4 to- 1 -line multiplexer.



- Functional

Table for 4-to1 Mux

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Type of multiplexers

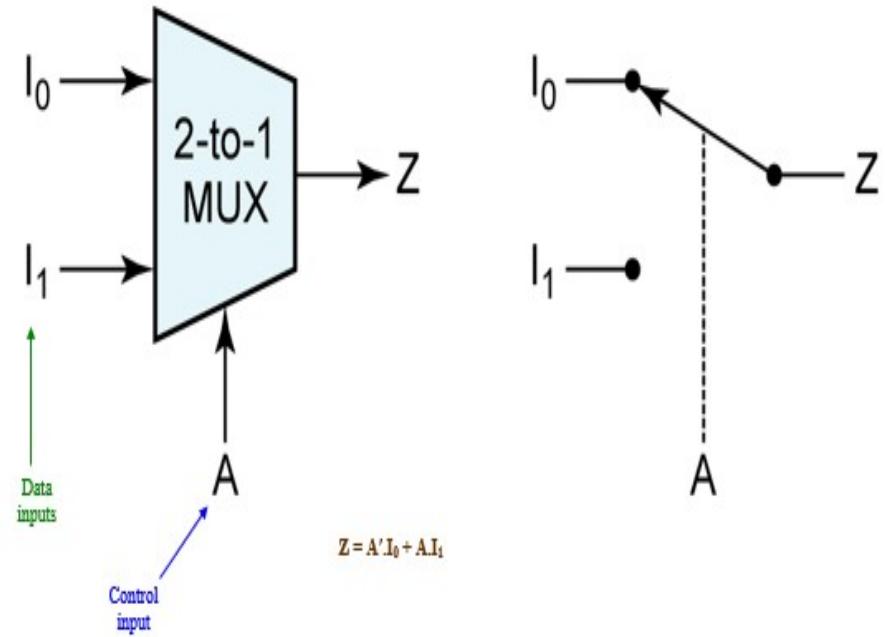
- Multiplexers come in multiple variations
 - 2 : 1 multiplexer
 - 4 : 1 multiplexer
 - 16 : 1 multiplexer
 - 32 : 1 multiplexer

Type of multiplexers

- Multiplexers come in multiple variations
 - 2 : 1 multiplexer
 - 4 : 1 multiplexer
 - 16 : 1 multiplexer
 - 32 : 1 multiplexer

2:1 Multiplexer

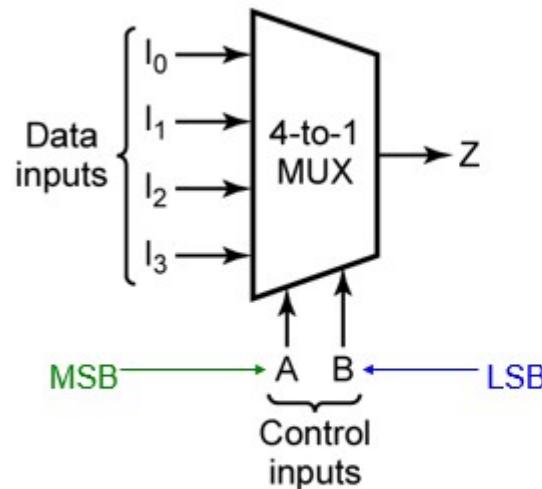
- 2:1
- Step1: Construct 2:1 Mux



4:1 Multiplexer

- Step :1 Construct 4:1 mux

- 4:1

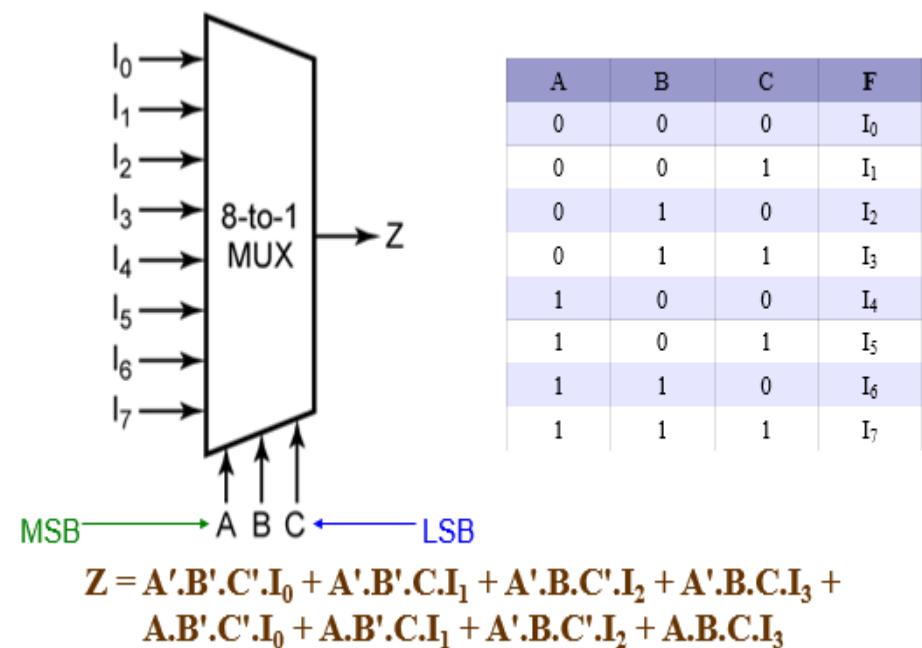


A	B	F
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

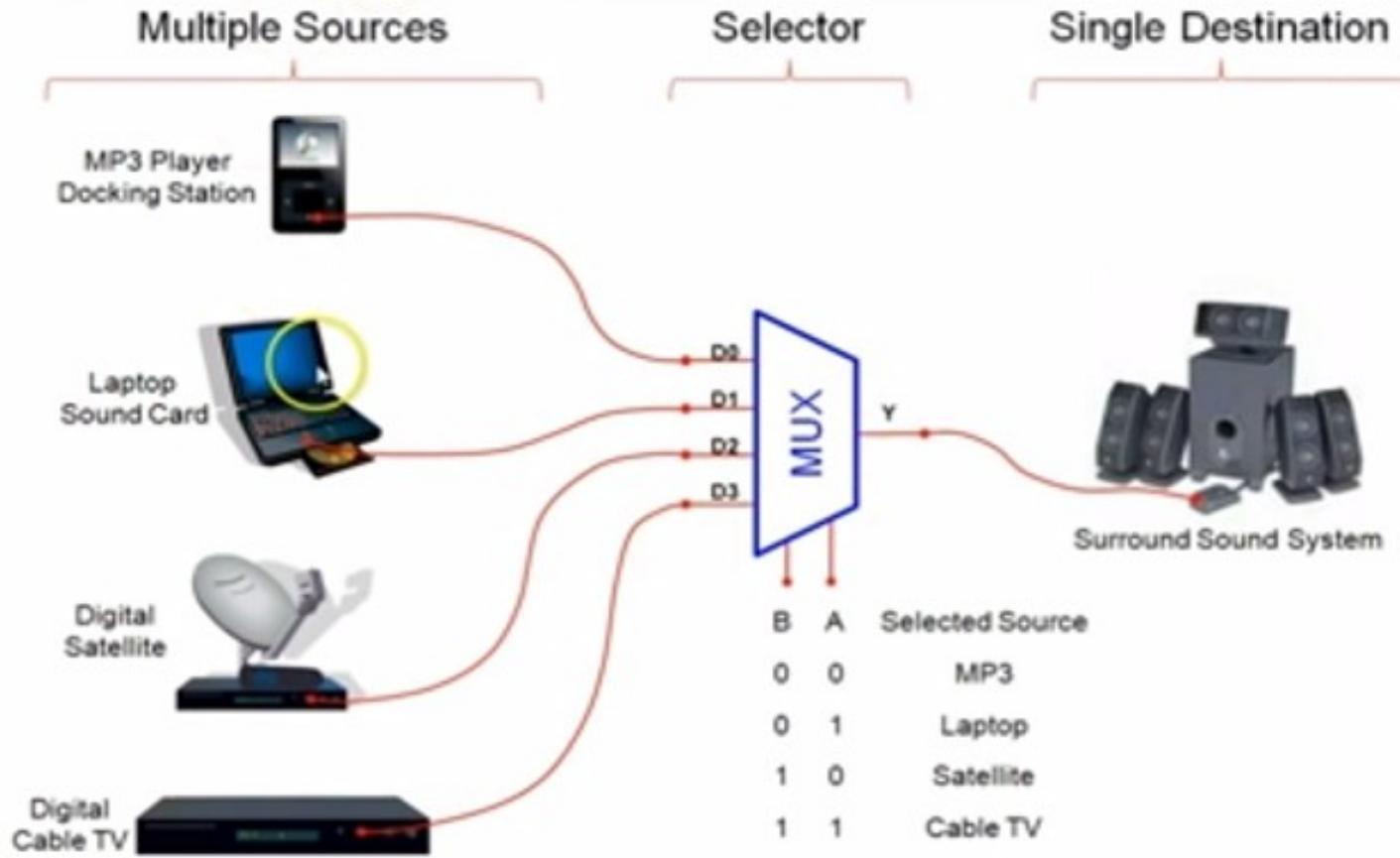
$$Z = A'.B'.I_0 + A'.B.I_1 + A.B'.I_2 + A.B.I_3$$

8:1 Multiplexer

- Step :1 Construct 8:1 mux



Typical Application of a MUX



Applications of Multiplexer

- **Multiplexer** are used in various fields where **multiple data** need to be **transmitted using a single line**. Following are some of the applications of multiplexers
- **Communication system** – Communication system is a set of system that **enable communication** like transmission system, relay and branch station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. **Multiplexer** allow the process of **transmitting** different type of data such as **audio**, **video** at the same time using a **single transmission line**.
- **Telephone network** – In **telephone network**, **multiple audio signals** are **integrated on a single line** for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.
- **Computer memory** – **Multiplexers** are used to **implement huge amount** of **memory** into the computer, at the same time **reduces** the **number of copper lines required** to connect the memory to other parts of the computer circuit.
- **Transmission from the computer system of a satellite** – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites

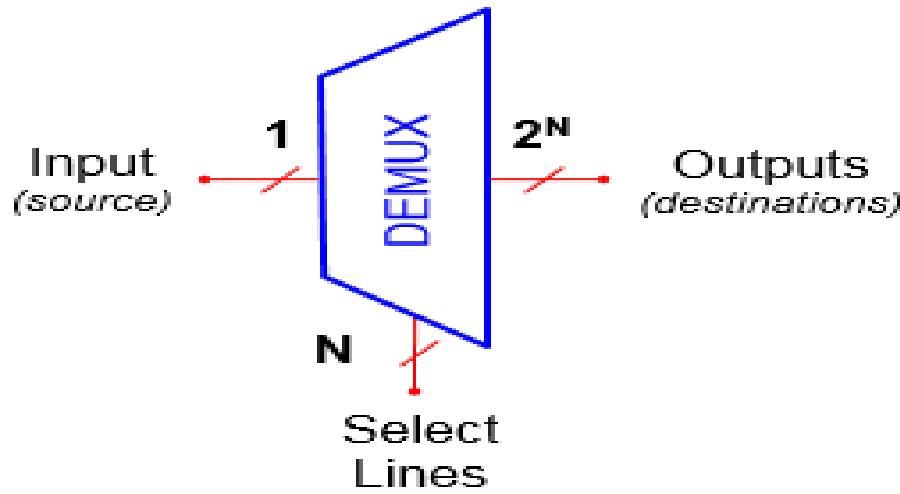
Demultiplexer

Demultiplexer

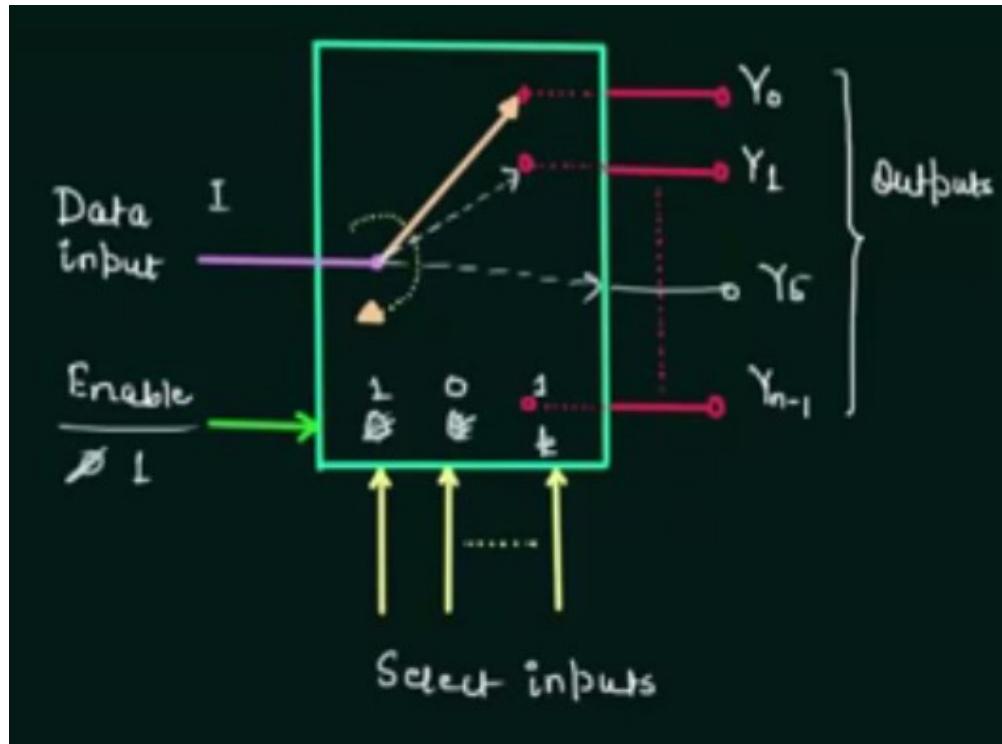
- "Demultiplexer", is a logical circuit that takes a **single input source** and sends it to one of **several 2^n possible output lines**.
- The selection of specific output line is controlled by the bit values of **'n' selection** lines.
- A **demultiplexer performs** the **opposite function** of a **multiplexer**.
- At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.
- DEMUX Types
 - 1-to-2 (1 select line)
 - 1-to-4 (2 select lines)
 - 1-to-8 (3 select lines)
 - 1-to-16 (4 select lines)

Demultiplexer

- Demultiplex Block Diagram



Relation of data input with select lines



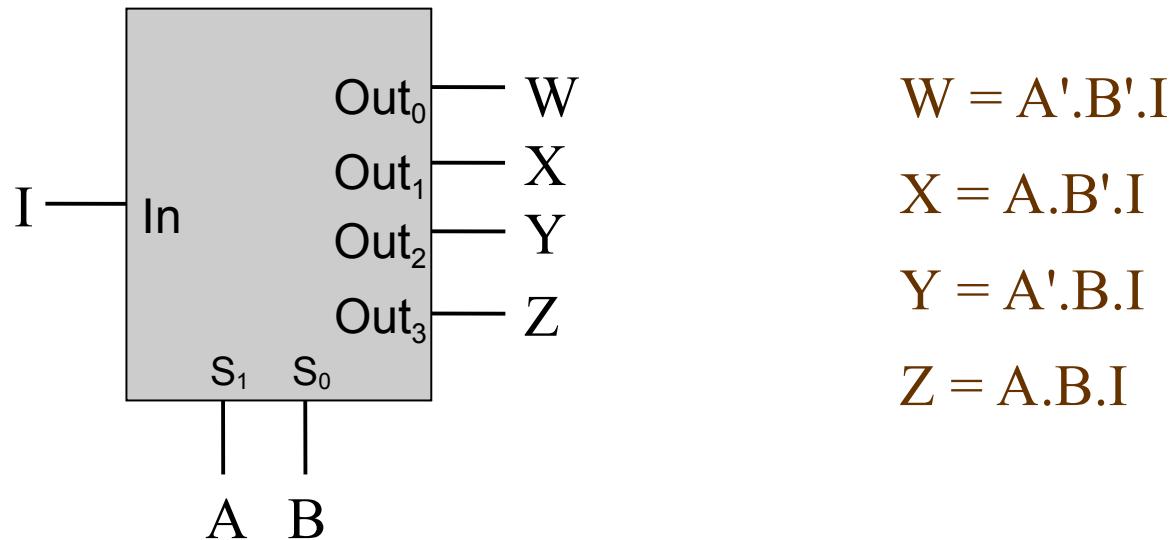
$$2^M = n$$

or

$$n = 2^M \Rightarrow m = \log_2 n$$

where n = outputs

Demultiplexers

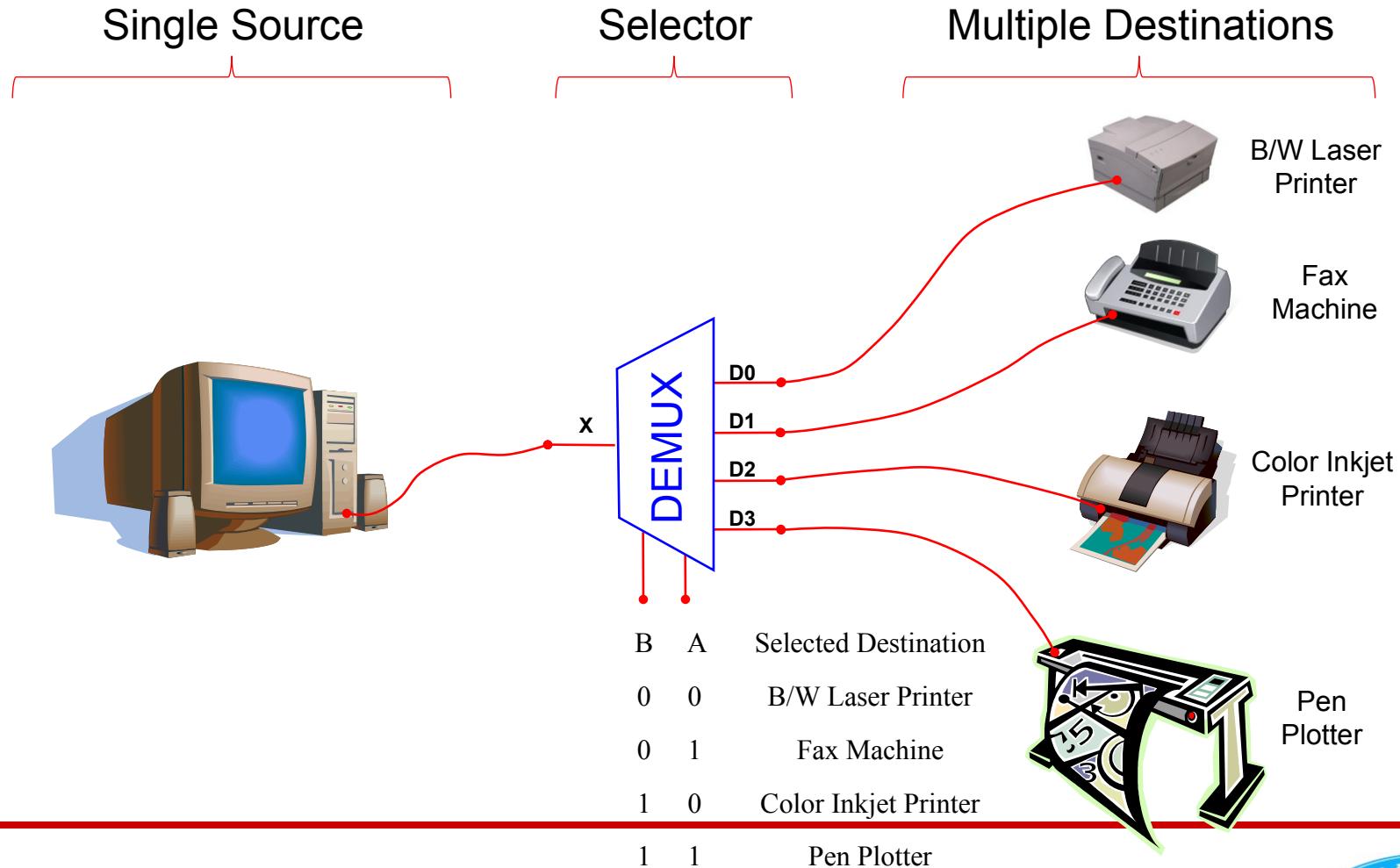


A	B	W	X	Y	Z
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

Applications of Demultiplexer

- Demultiplexer is used to connect a **single source** to **multiple destinations**. The **main application area** of **demultiplexer** is **communication system** where multiplexer are used.
- Most of the **communication system** are **bidirectional** i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync.
- Demultiplexer are also used for **reconstruction** of parallel data and ALU circuits.
- **Communication System** – Communication system use multiplexer to **carry multiple data** like **audio**, **video** and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The **multiplexer** and **demultiplexer work together** to carry out the **process of transmission** and **reception of data** in **communication system**.
- **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of **ALU** can be **stored** in **multiple registers** or **storage units** with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
- **Serial to parallel converter** – A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

Typical Application of a DEMUX

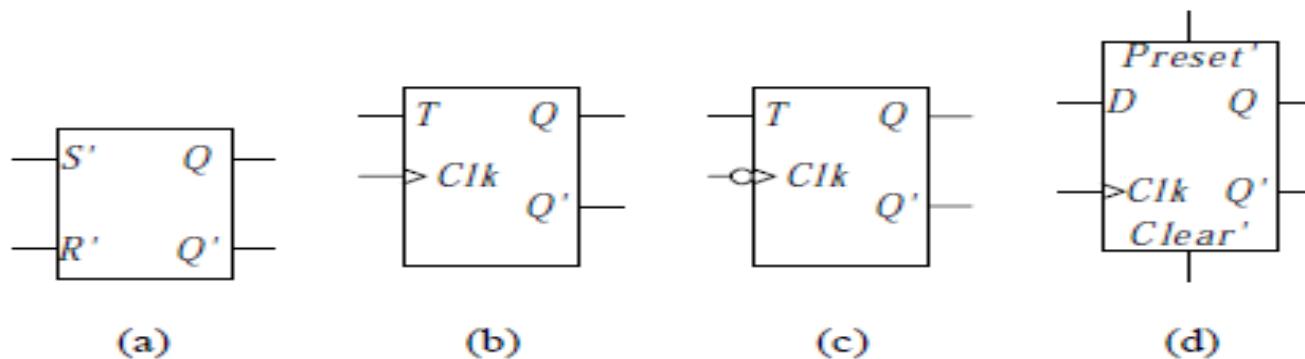


End

Flip flops

Logic Symbol

- The **logic** or **graphical symbol** describes the flip-flop's inputs and outputs, the names given to these signals, and whether they are active high or low.
- All the flip-flops have Q and Q' as their outputs. All of them also have a CLK input.
- The **small triangle** at the **clock input indicates** that the circuit is a flip-flop and so it is triggered by the edge of the **clock signal**; if there is a **circle** in front, then it is the **falling edge**, otherwise, it is the rising edge of the clock signal.
- **Without the small triangle**, the circuit is a **latch**. In addition, the **flip-flops** have one or two more inputs that characterize the flip-flop and give it its name.

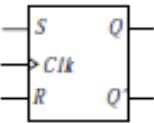
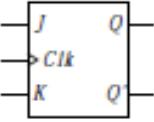
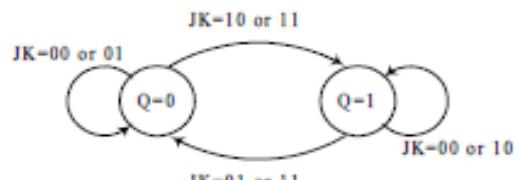
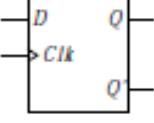
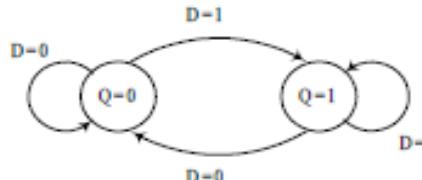
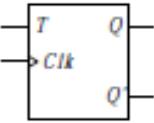
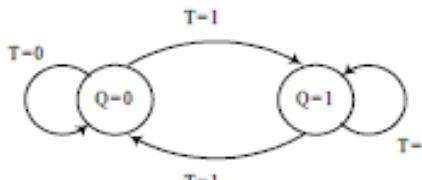


1

Various logic symbols: (a) Active low SR latch; (b) positive-edge-triggered active high T flip-flop; (c) negative-edge-triggered T flip-flop; (d) positive-edge-triggered D flip-flop with asynchronous active low preset and clear.

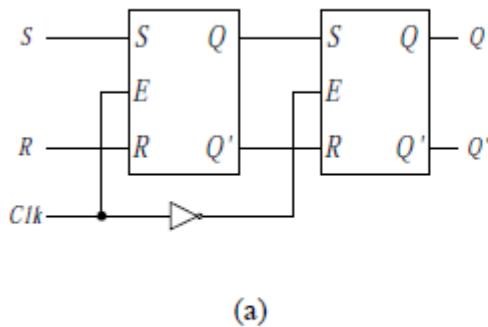
Type of flip Flop

- There are basically **four main types** of flip-flops: **SR**, **D**, **JK**, and **T**.
- However, **J-K FFs** are a lot **more complex** to build than D-types and so have fallen out of favor in modern designs, e.g., for field programmable gate arrays (FPGAs) and VLSI chips
- The **major differences** in these **flip-flop types** are in the number of inputs they have and how they **change state**.
- Each type can have different variations such as **active high or low inputs**, whether they **change state** at the **rising or falling edge** of the clock signal, and whether they have asynchronous inputs or not.
- The flip-flops can be described fully and uniquely by its logic symbol(Block Diagram), characteristic table, characteristic equation, state diagram, or excitation table,

Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table																																																								
SR 	<table border="1"> <thead> <tr> <th>S</th><th>R</th><th>Q</th><th>Q_{next}</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>x</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>x</td></tr> </tbody> </table>	S	R	Q	Q_{next}	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0	1	0	0	1	1	0	1	1	1	1	0	x	1	1	1	x	 $Q_{next} = S + R'Q$ $SR = 0$	<table border="1"> <thead> <tr> <th>O</th><th>Q_{next}</th><th>S</th><th>R</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>x</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>x</td><td>0</td></tr> </tbody> </table>	O	Q_{next}	S	R	0	0	0	x	0	1	1	0	1	0	0	1	1	1	x	0
S	R	Q	Q_{next}																																																								
0	0	0	0																																																								
0	0	1	1																																																								
0	1	0	0																																																								
0	1	1	0																																																								
1	0	0	1																																																								
1	0	1	1																																																								
1	1	0	x																																																								
1	1	1	x																																																								
O	Q_{next}	S	R																																																								
0	0	0	x																																																								
0	1	1	0																																																								
1	0	0	1																																																								
1	1	x	0																																																								
JK 	<table border="1"> <thead> <tr> <th>J</th><th>K</th><th>O</th><th>Q_{next}</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	J	K	O	Q_{next}	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0	1	0	0	1	1	0	1	1	1	1	0	1	1	1	1	0	 $Q_{next} = J'K'Q + JK' + JKQ'$ $= JK'Q + JK'Q + JK'Q' + JKQ'$ $= K'Q(J'+J) + JQ'(K'+K)$ $= K'Q + JQ'$	<table border="1"> <thead> <tr> <th>O</th><th>Q_{next}</th><th>J</th><th>K</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td><td>x</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>x</td></tr> <tr><td>1</td><td>0</td><td>x</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>x</td><td>0</td></tr> </tbody> </table>	O	Q_{next}	J	K	0	0	0	x	0	1	1	x	1	0	x	1	1	1	x	0
J	K	O	Q_{next}																																																								
0	0	0	0																																																								
0	0	1	1																																																								
0	1	0	0																																																								
0	1	1	0																																																								
1	0	0	1																																																								
1	0	1	1																																																								
1	1	0	1																																																								
1	1	1	0																																																								
O	Q_{next}	J	K																																																								
0	0	0	x																																																								
0	1	1	x																																																								
1	0	x	1																																																								
1	1	x	0																																																								
D 	<table border="1"> <thead> <tr> <th>D</th><th>O</th><th>Q_{next}</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>x</td><td>1</td></tr> </tbody> </table>	D	O	Q_{next}	0	0	0	1	x	1	 $Q_{next} = D$	<table border="1"> <thead> <tr> <th>O</th><th>Q_{next}</th><th>D</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	O	Q_{next}	D	0	0	0	0	1	1	1	0	0	1	1	1																																
D	O	Q_{next}																																																									
0	0	0																																																									
1	x	1																																																									
O	Q_{next}	D																																																									
0	0	0																																																									
0	1	1																																																									
1	0	0																																																									
1	1	1																																																									
T 	<table border="1"> <thead> <tr> <th>T</th><th>O</th><th>Q_{next}</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	T	O	Q_{next}	0	0	0	0	1	1	1	0	1	1	1	0	 $Q_{next} = TQ' + T'Q = T \oplus Q$	<table border="1"> <thead> <tr> <th>O</th><th>Q_{next}</th><th>T</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	O	Q_{next}	T	0	0	0	0	1	1	1	0	1	1	1	0																										
T	O	Q_{next}																																																									
0	0	0																																																									
0	1	1																																																									
1	0	1																																																									
1	1	0																																																									
O	Q_{next}	T																																																									
0	0	0																																																									
0	1	1																																																									
1	0	1																																																									
1	1	0																																																									

SR Flip-Flop

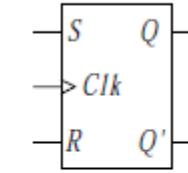
- Like SR latches, SR flip-flops are useful in **control applications** where we want to be able to **set** or **reset** the data bit.
- However, unlike SR latches, SR flip-flops change their content only at the active edge of the clock signal. Similar to SR latches, SR flip-flops can enter an undefined state when both inputs are asserted simultaneously.



(a)

S	R	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	x	x
1	1	1	x	x

(b)



(c)

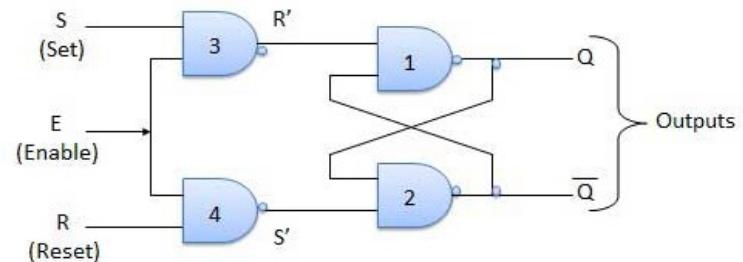
SR flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

S-R Flip Flop

- Operation

S.N.	Condition	Operation
1	$S = R = 0$: No change	If $S = R = 0$ then output of NAND gates 3 and 4 are forced to become 1. Hence R' and S' both will be equal to 1. Since S' and R' are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.
2	$S = 0, R = 1, E = 1$	Since $S = 0$, output of NAND-3 i.e. $R' = 1$ and $E = 1$ the output of NAND-4 i.e. $S' = 0$. Hence $Q_{n+1} = 0$ and Q_{n+1} bar = 1. This is reset condition.
3	$S = 1, R = 0, E = 1$	Output of NAND-3 i.e. $R' = 0$ and output of NAND-4 i.e. $S' = 1$. Hence output of S-R NAND latch is $Q_{n+1} = 1$ and Q_{n+1} bar = 0. This is the reset condition.
4	$S = 1, R = 1, E = 1$	As $S = 1, R = 1$ and $E = 1$, the output of NAND gates 3 and 4 both are 0 i.e. $S' = R' = 0$. Hence the Race condition will occur in the basic NAND latch.

Circuit Diagram



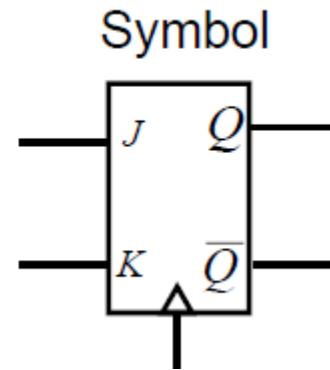
Truth Table

Inputs			Outputs		Comments
E	S	R	Q_{n+1}	\bar{Q}_{n+1}	
1	0	0	Q_n	\bar{Q}_n	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	x	x	Indeterminate

J-K Flip-Flop

- The J-K FF is similar in function to a clocked RS FF, but with the illegal state replaced with a new ‘toggle’ state

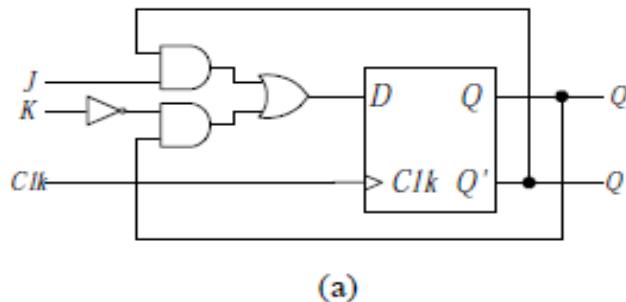
$J\ K$	Q'	\bar{Q}'	comment
0 0	Q	\bar{Q}	hold
0 1	0	1	reset
1 0	1	0	set
1 1	\bar{Q}	Q	toggle



- Where Q' is the next state and Q is the current state

JK Flip-Flop

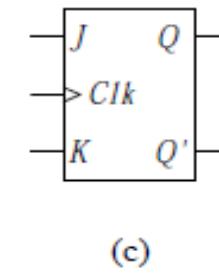
- JK flip-flops are very similar to SR flip-flops. The J input is just like the S input in that when asserted, it sets the flip-flop. Similarly, the K input is like the R input where it clears the flip-flop when asserted. The only difference is
- when both inputs are asserted. For the SR flip-flop, the next state is undefined, whereas, for the JK flip-flop, the next state is the inverse of the current state. In other words, the JK flip-flop toggles its state when both inputs are asserted. The circuit, truth table and the logic symbol for the JK flip-flop is shown in Figure 17.



(a)

J	K	Q	Q_{next}	Q_{next}'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

(b)

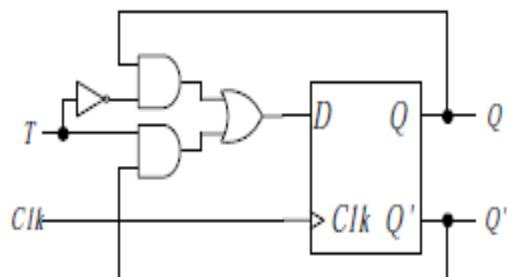


(c)

JK flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

T Flip Flop

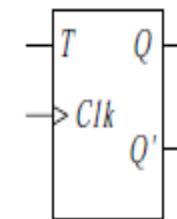
- The T flip-flop has one input in addition to the clock.
- T stands for toggle for the obvious reason. When T is asserted ($T = 1$), the flip-flop state toggles back and forth, and when T is de-asserted, the flip-flop keeps its current state.
- The T flip-flop can be constructed using a D flip-flop with the two outputs Q and Q' feedback to the D input through a multiplexer that is controlled by the T input as shown in Figure 18.



(a)

T	Q	Q_{next}	Q_{next}'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

(b)



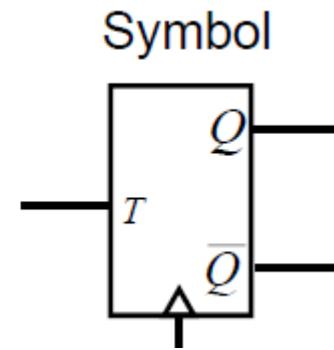
(c)

T flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

T Flip-Flop

- This is essentially a J-K FF with its J and K inputs connected together and renamed as the T input

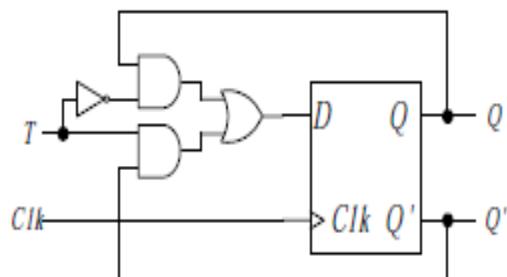
T	Q'	\bar{Q}'	comment
0	Q	\bar{Q}	hold
1	\bar{Q}	Q	toggle



Where Q' is the next state and Q is the current state

T Flip Flop

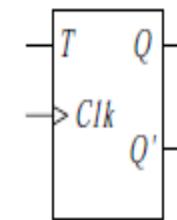
- The T flip-flop has one input in addition to the clock.
- **T** stands for **toggle** for the obvious reason. When T is asserted ($T = 1$), the flip-flop state toggles back and forth, and when T is de-asserted, the flip-flop keeps its current state.
- The T flip-flop can be constructed using a D flip-flop with the two outputs Q and Q' feedback to the D input through a multiplexer that is controlled by the T input as shown in Figure 18.



(a)

T	Q	Q_{next}	Q_{next}'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

(b)

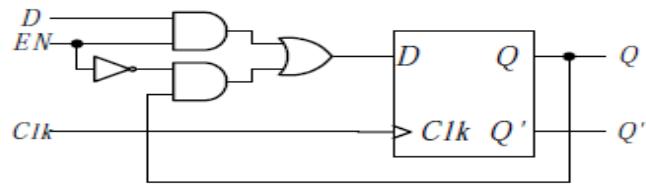


(c)

T flip-flop: (a) circuit; (b) truth table; (c) logic symbol.

D Flip-Flop

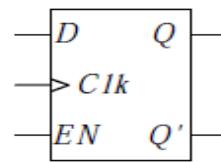
- A commonly desired function in D flip-flops is the ability to hold the last value stored rather than load in a new value at the clock edge. This is accomplished by adding an enable input called EN or CE (clock enable) through a multiplexer as shown in Figure a.
- When $EN = 1$, the primary D signal will pass to the D input of the flip-flop, thus updating the content of the flip-flop. When $EN = 0$, the bottom AND gate is enabled and so the current content of the flip-flop, Q , is passed back to the input, thus, keeping its current value. Notice that changes to the flip-flop value occur only at the rising edge of the clock. The truth table and the logic symbol for the D flip-flop with enabled is shown in (b) and (c) respectively.



(a)

C _{lk}	EN	D	Q	Q _{next}	Q _{next'}
0	×	×	0	0	1
0	×	×	1	1	0
1	×	×	0	0	1
1	×	×	1	1	0
↑	0	×	0	0	1
↑	0	×	1	1	0
↑	1	0	×	0	1
↑	1	1	×	1	0

(b)



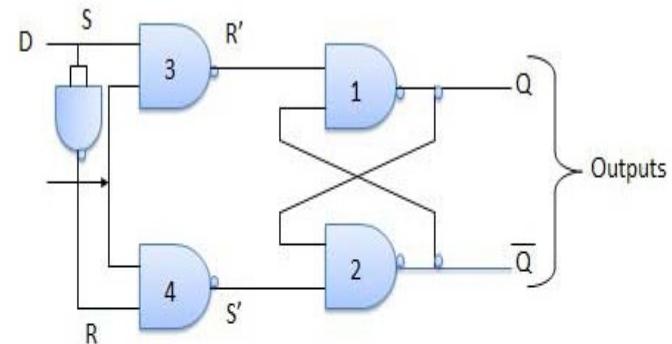
(c)

D Flip Flop

• Operation

S.N.	Condition	Operation
1	$E = 0$	Latch is disabled. Hence no change in output.
2	$E = 1$ and $D = 0$	If $E = 1$ and $D = 0$ then $S = 0$ and $R = 1$. Hence irrespective of the present state, the next state is $Q_{n+1} = 0$ and $Q_{n+1} \text{ bar} = 1$. This is the reset condition.
3	$E = 1$ and $D = 1$	If $E = 1$ and $D = 1$, then $S = 1$ and $R = 0$. This will set the latch and $Q_{n+1} = 1$ and $Q_{n+1} \text{ bar} = 0$ irrespective of the present state.

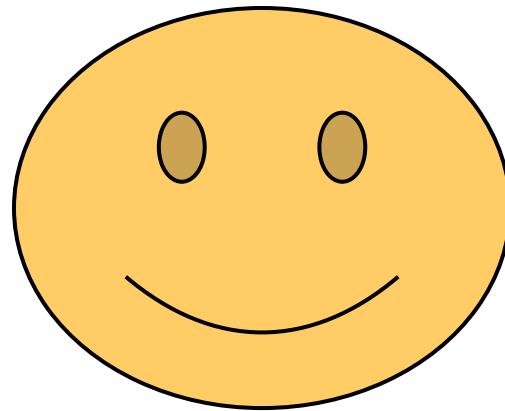
Circuit Diagram



Truth Table

Inputs	Outputs	Comments		
E	D	Q_{n+1}	\bar{Q}_{n+1}	
1	0	0	1	Rset
1	1	1	0	Set

Thank You!!!



Advanced computer Architecture concept

Chapter

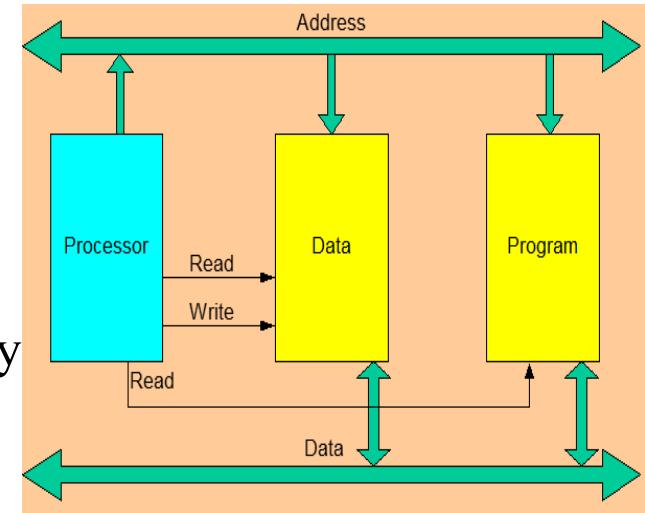
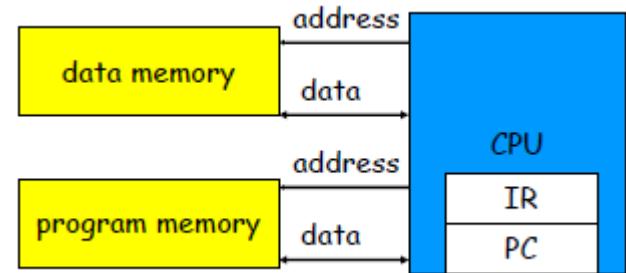
Content

- **Von Neumann Architecture**
 - ✓ Von Neumann Architecture vs. Harvard architecture
- **CISC and RISC computer**
- **Parallel Processing**
- **Symmetric Multiprocessing**
- **Clustering**

- We can classified processors (CPU) based on
 - **The word size(data handling capacity)**
 - 8 bit, 16 bit, 32 bit, etc. processors
 - **Instruction set structure**
 - RISC (Reduced Instruction Set Computer), CISC (Complex Instruction Set Computer)
 - **Functions**
 - General purpose, special purpose such image processing, floating point calculations
 - **Memory organization**
 - Von-Neumann architecture
 - Harvard architecture

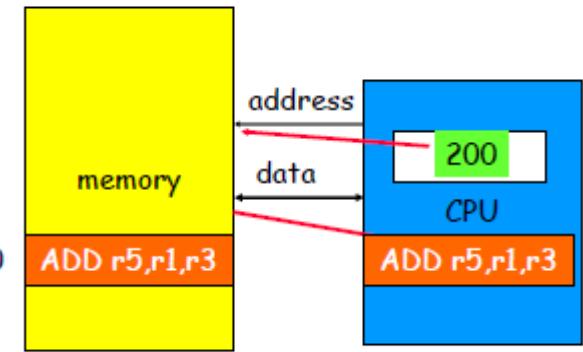
Harvard Architecture(HA)

- The name **Harvard Architecture** comes from the **Harvard Mark I** relay-based computer
- It is **developed** by **Harvard university's staff.**
- The most obvious **characteristic** of the **Harvard Architecture** is that it has physically **separate signals** and **Storage** for **code** and **Data memory**. what means this ??
- **Separate memories for Data and Instructions.**
- **Two sets of Address/Data buses between CPU and memory**
- It is possible to access **program memory** and **data memory** simultaneously.
- it is **impossible** for program contents to be **modified** by the **program itself**. Why ??
- code (or program) memory is read-only and data memory is read-write.



Von-Neumann architecture(VNA)

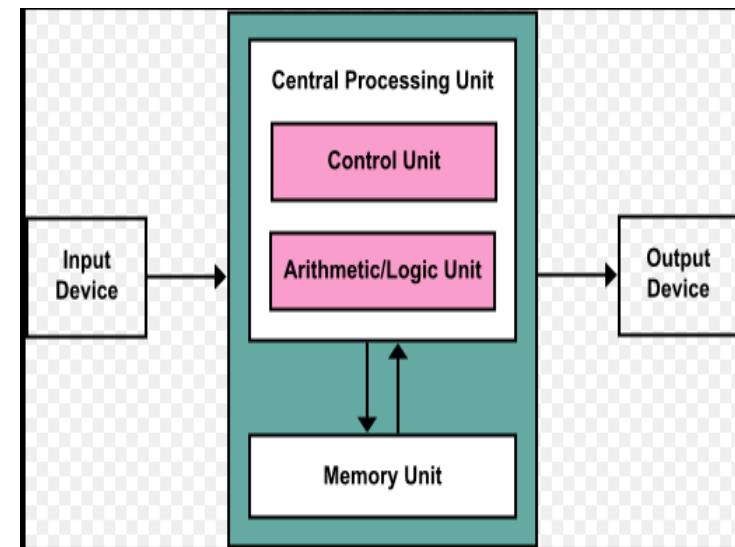
- **John von Neumann** was a Hungarian-American mathematician, physicist, inventor, computer scientist, and polymath.
 - He made major **contributions** to a number of fields, including mathematics, physics, economics, computing, and statistics(Wikipedia)
 - The **von Neumann Architecture** is **named** after the mathematician and early computer scientist **John von Neumann**.
- VNA



Von-Neumann architecture(VNA)

- Von Neumann machines have **shared signals** and **memory** for **code(Instruction)** and **data**. Thus,
- The program can be **easily modified** by itself since it is **stored** in **read-write memory**
 - **Instructions** and **Data** are stored in the same memory.
 - **Instructions** and **Data** share one memory system.
- VNA has only **one bus** which is used for both **data transfers** and **instruction fetches**
- Therefore **data transfers** and **instruction fetches** must be **scheduled** – they can not be **performed** at the **same time**.

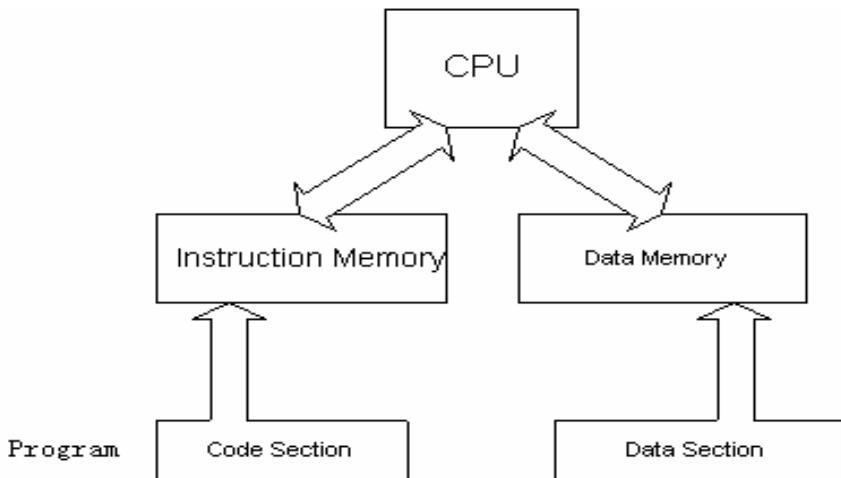
- VNA



HA VS VNA

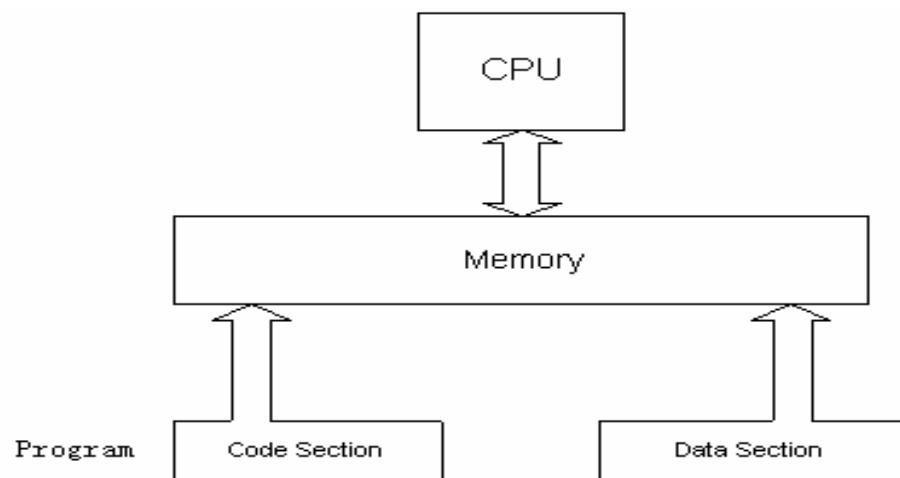
• Harvard Architecture

- ✓ has separate **data** and **instruction**
- ✓ Has Separate Bus for **Instruction** ad **Data**
- ✓ **Perform one step command** for instruction and data at the same time (shows that 2 buses 2 work synchronically).
- ✓ it is **impossible** for program contents to be modified by the **program itself**



• Von Neumann Architecture

- ✓ Shared **Data** and **Instruction**
- ✓ has **one bus** for both **data transfers** and **instruction**
- ✓ **perform two command** (first read an instruction, and then read the data that the instruction requires.)
- ✓ program can be **easily modified**



CISC vs RISC

Instruction Set

- An **agreement** between **hardware** and **human** for making interaction.
- The **instruction set** chosen for a **particular computer** determines the way that **machine language programs** are constructed.
- As **digital hardware** became **cheaper** with the start of **integrated circuits**, computer instructions tended to increase both in number and complexity.
- A computer with a **fewer number** of **instructions with complex construct** classified as a **complex instruction set computer abbreviated CISC**.
- computers use **large number** of **instructions** with **simple constructs** so they can be **executed** much faster within the **CPU without** having to use memory as often. This type of computer is classified as a **reduced instruction set computer or RISC**.

CISC

- Pronounced **sisk**, and stands for **Complex Instruction Set Computer**
- **Most PC's** use **CPU** based on this architecture.
- The philosophy behind it is that **hardware** is always faster than software,
- therefore one should make a **powerful instruction set**, which provides programmers with assembly instructions to do a lot with **short programs**
- **On single Instruction** Capable to perform multiple operation (**MULT A,B**)

Note MULT refer as complex Instruction

- It is **Designed** to minimize the number of **instructions per program(Short program)** , ignoring the number of cycles per instruction(long time to execute b/c it is complex instruction).
- CPU designed to carry out **many operations** in a **single instruction**. These can be loading from and to memory
- The **Emphasis** is on **building complex instructions directly** into the **hardware**.
- because of the length of the code is **relatively short**, so very little RAM is required to store the instructions.

RISC

- Pronounced *risk*, and stands for **Reduced Instruction Set Computer**.
- Apple for instance uses RISC chips
- The philosophy behind it is that give emphasis for **software** than **hardware** base on **simpler** and **faster** instructions would be better, than the large, complex instruction

```
Load R1,A  
Load R1,B  
PROD A,B  
STORE R3,A
```

Note * The MULT command in CISC divided into three separate Command Load , Prod and Store

- The **Instruction** set processor is **simplified** to **Reduce execution time**
- **Each instruction** is meant to achieve **very small tasks**.
- **More RAM** is needed to store the **assembly level instructions**.
- The **compiler** must also perform more work to convert a high-level language statement into code of this form
- In RISC, **Pipelining** is easy as the execution of all instructions will be done in a uniform interval of time i.e. one click.

RISC VS CISC

Characteristics of RISC

- It consists of simple instructions.
- It large(Multiple) number of Instruction
- Emphasis on software
- More Space need (RAM)
- Pipelining Support
- Decoding Instruction is simple
- The performance is depend on the programmer
- Execution time Is very less
- Support High level Language

Characteristics of CISC

- It Consists Complex Instruction
- Small number of instructions.
- Emphasis on software
- Less Space need (RAM)
- Pipeline doesn't support
- Decoding Instruction is Complex
- Not Depend on the programmer
- Execution time is very high
- Support Assembly language

The simplest way to understand CISC and RISC Multiplying Two Numbers in Memory

CISC Approach

Let's say we want to find the product of two numbers - one stored in location 2:3 and 5:2

- The primary goal of CISC architecture is to complete a task in

as few lines of assembly as possible

▪ This is achieved by building processor hardware that is capable of understanding and executing a series of operations.

▪ For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT").

▪ When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

MULT 2:3, 5:2

▪ MULT is what is known as a "**complex instruction**." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions. It closely resembles a command in a higher level language. For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a * b."

▪ One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly. Because the length of the code is relatively short, very little RAM is required to store instructions. The emphasis is put on building complex instructions directly into the hardware.

▪ The **CISC approach** attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program

RISC Approach

- RISC processors only use simple instructions that can be executed within one clock cycle.
- In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly

LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A

- the "**MULT**" command described on the left could be divided into **three separate commands(STORE, LOAD and PROD)**
- At first, this may seem like a much **less efficient** way of completing the operation. Because there **are more lines of code, more RAM is needed to store the assembly level instructions**.
- The compiler must also perform more work to convert a high-level language statement into code of this form.
- The RISC strategy also brings some very important advantages. Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MULT" command. These RISC "reduced instructions" require less transistors of hardware space than the complex instructions, leaving more room for general purpose registers. Because all of the instructions execute in a uniform amount of time (i.e. one clock), **pipelining is possible**.
- ."MULT" command is executed, the processor automatically erases the registers. If one of the operands needs to be used for another computation, the processor must re-load the data from the memory bank into a register. In RISC, the operand will remain in the register until another value is loaded in its place.

Parallel Processing

CS221

Computer Organization and
Architecture



What is Parallel Processing?

- Before discuss about parallel processing Answer the few question
- How to make machines **solve problems better** and **faster**?
- Physical barriers limit the extent to which single processor performance can be improved—Clock Speed & Heat Dissipation
- What is the solution ??
- **Distribute** the computing load among **several processors**
- Using multiple processing elements **simultaneously** to **solve** a problem.
- accomplished by **breaking** the problem into **independent parts** so that each processing element can **execute** its part of the algorithm simultaneously with the others.
- The processing elements can include **resources** such as a single computer with **multiple processors**, **several networked computers**, **specialized hardware**, or any combination of the above

What is Parallel Processing?

- **Parallel processing** is a term used to indicate a **large class** of techniques that are used to provide **simultaneous data-processing tasks** for the **purpose of increasing the computational speed** of a computer system. Instead of **processing** each **instruction sequentially** as in a conventional computer,
- a **parallel processing system** is able to perform **concurrent data processing** to achieve **faster execution time**.

Parallel processing

- The **simultaneous** use of more than **one Cpu** to execute a program
- Ideally, **parallel processing** makes a **program run faster** because there are **more engines** (CPUs) running it.
- In practice, it is often difficult to divide a program in such a way that **separate CPUs** can **execute different portions** without **interfering** with each other.
- **Parallel processing** is a method of **simultaneously breaking up** and **running program tasks** on **multiple microprocessors**, thereby **reducing processing time**.
- **Parallel processing** may be accomplished via a computer with **two or more processors** or via a **computer network**. Parallel processing is also called **parallel computing**

Multitasking

- Multitasking allow to **execute** more than one tasks at the **same time**
- In multitasking **only one CPU** is involved but it can be **switched** from **one program** to **another program** so **quickly**. That is why it gives the appearances of exciting all of the programs at the same times
- Multitasking allow processor(programs) to run concurrently on the program
- Example running the spreadsheet program and you can working with word processor also.
- Multitasking is running heavyweight processor by a single OS
-

Parallel processing

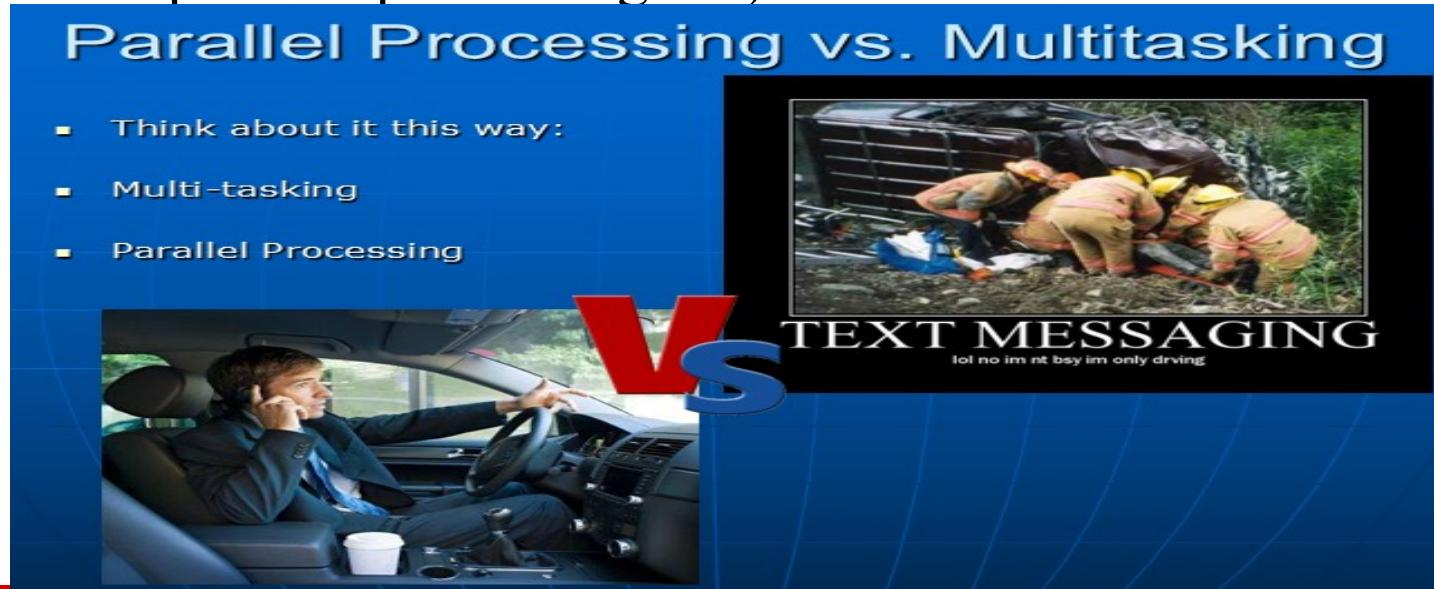
What do you
Think about
The Picture ?



(Human parallel processing ☺)

Parallel Processing vs. Multitasking

- Think about it this way:
- Multi-tasking
- Parallel Processing



VS

TEXT MESSAGING

lol no im nt bsy im only driving

Types of Multiprocessor Systems

Asymmetric Multiprocessing

- **master processor** schedules and allocates work to **slave processors**

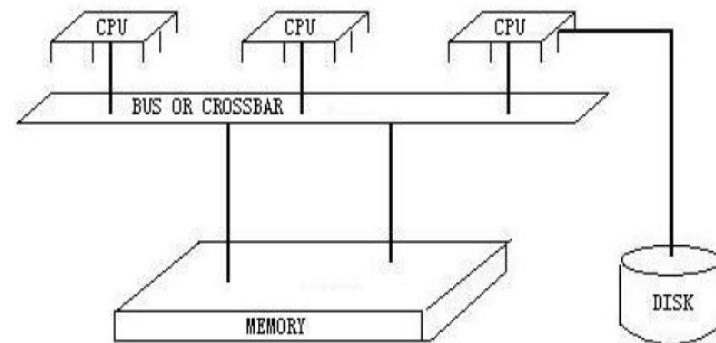
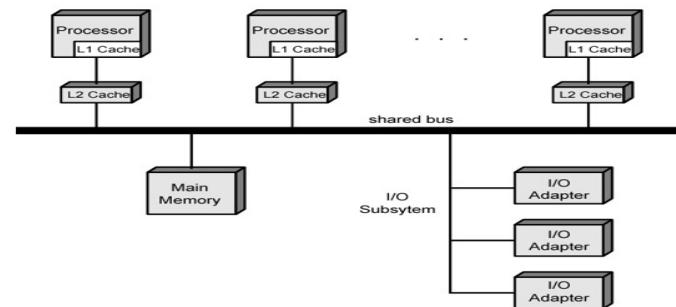
Symmetric Multiprocessing

- **Each processor** runs an **identical copy** of the operating system
- Typically each processor does self scheduling from the pool of available process

Symmetric Multiprocessors

- Each processor can perform the same functions and share same main memory and I/O facilities
- Processors are connected by a bus or other internal connection
- Memory access time is approximately the same for each processor
- It is a **multiprocessor** computer architecture where **two or more** identical processors are connected to a single **shared main memory**
- In the fig Show Three processors are connected to the same memory module through a bus

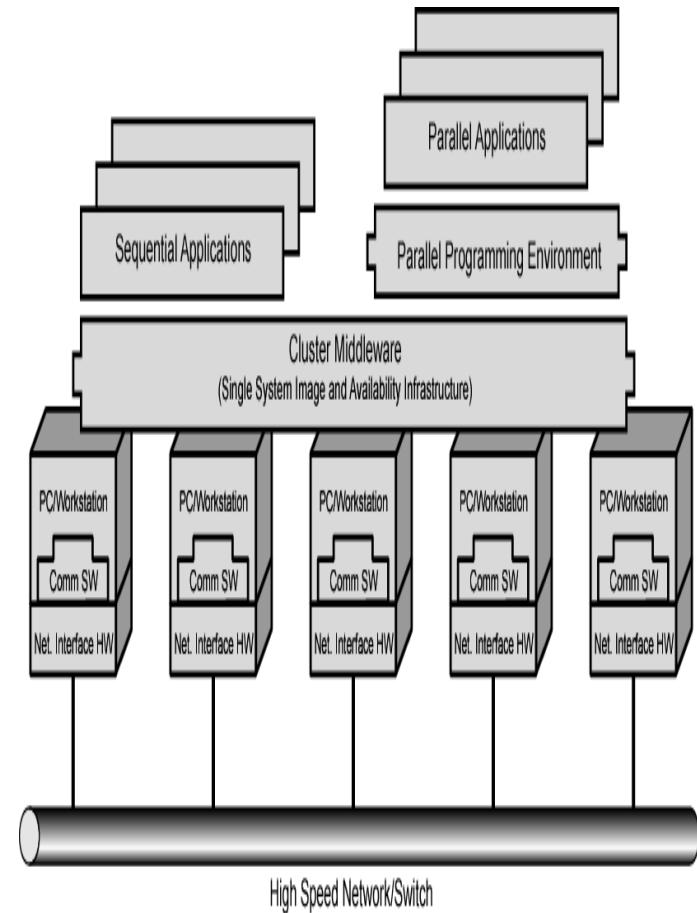
Symmetric Multiprocessor



Clustering

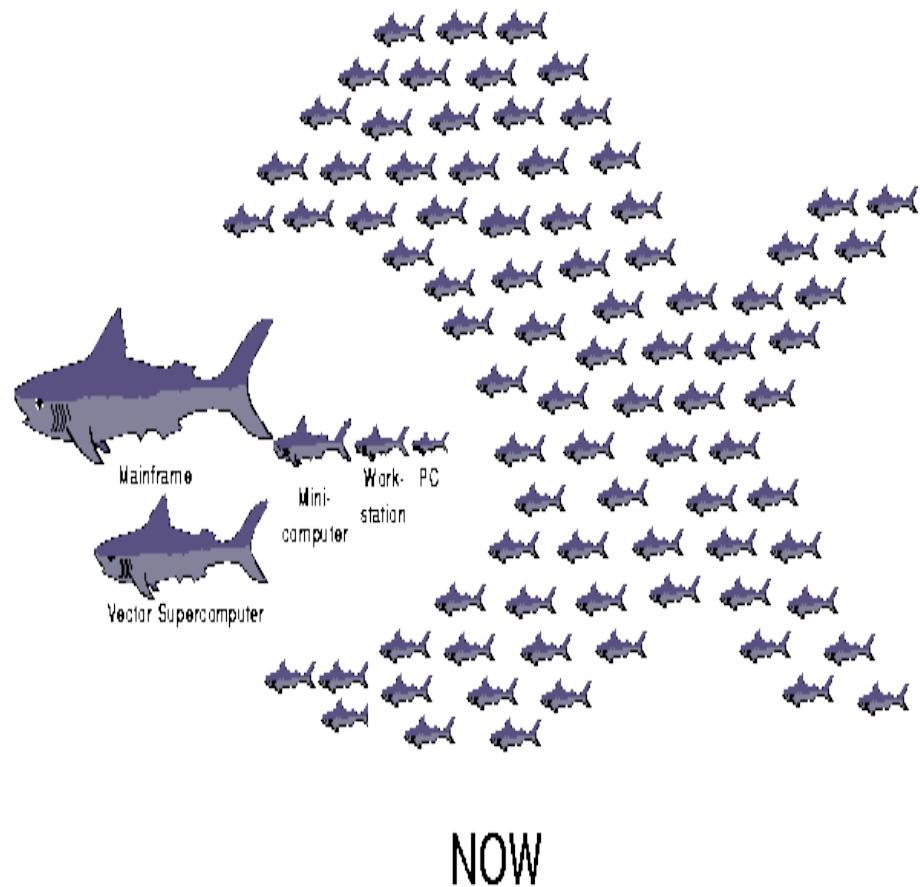
- A cluster computer is a type of **parallel or distributed processing system**,
 - which consists of a collection of interconnected stand-alone computers **working together** as a **single integrated** resource.
- The typical architecture of a cluster is shown in

- Clustering

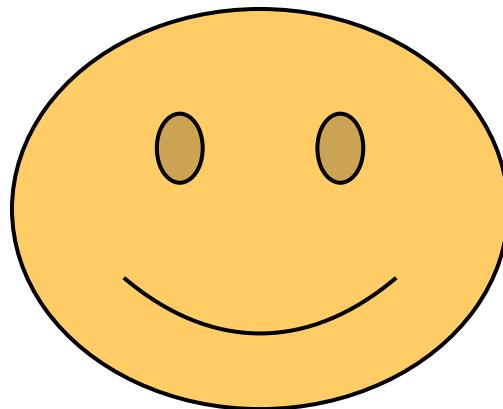


prominent components of cluster computers

- ✓ Multiple high performance computers (PCs, Workstations, SMP servers).
- ✓ Operating systems (Layered or Micro-Kernel based).
- ✓ High Performance Networks (Switched network fabrics, Gigabit Ethernet).
- ✓ Cluster Middleware (Single System Image, and System Availability Infrastructure).
- ✓ Parallel Programming Environments.
- ✓ Applications.



Thank You!!!





ADMAS UNIVERSITY
Department Computer Science
Computer ARCTUCTURE AND Organization
Assignment 3 and 4
Prepared by Ebdu. j

Individual Assignment 3 and 4

Submission Date: June 17

Instruction

- i. Don't forget to write your **Name, Section and ID**.
- ii. This Assignment contains **8 Questions**
- iii. Write your answer **Clearly and neatly**
- iv. Do independently without copying from each other's.
Copying from others Invalidate your work.
- v. Submit your assignment to the following email address:
"ebduj20@gmail.com" and the subject of your email
should be "**COA Assignment 3&4**"
- vi. Late submission will not accepted.
- vii.** You can submit the assignment in **Microsoft Word** or
PDF formant. **Image format not acceptable**

Questions

1. Explain briefly Memory Hierarchy
2. Explain Three Types of mapping in Cache Organization
3. Differentiate sequential circuit and combinational circuit using examples
4. Differentiae Decoder and encoder
5. Explain S-r and J_K Flip flop
6. Compare and contrast Von-Neumann architecture and Harvard architecture
7. Why we need multitasking

8. Explain the purpose of cluster computer