



AL-AMEEN ENGINEERING COLLEGE

(AUTONOMOUS)

Accredited by NAAC with “A” Grade:: An ISO Certified Institution

(Affiliated to Anna University, Chennai & Approved by AICTE, New Delhi)

Karundevanpalayam, NanjaiUthukkuli Post, Erode – 638 104, Tamilnadu, INDIA.

DEPARTMENT OF EEE

LECTURE NOTES

COURSE TITLE: 20CSO02–COMPUTER ARCHITECTURE

III YEAR / V SEMESTER

REGULATION 2020

PREPARED BY

Mr.S.N.SYED JAMESHA, ME., AP / ECE

Functional Units - Basic Operational Concepts - Bus Structures - Performance - Memory Locations and Addresses - Memory Operations - Instruction and Instruction Sequencing – Addressing Modes - Basic I/O Operations.

Introduction:

Computer types

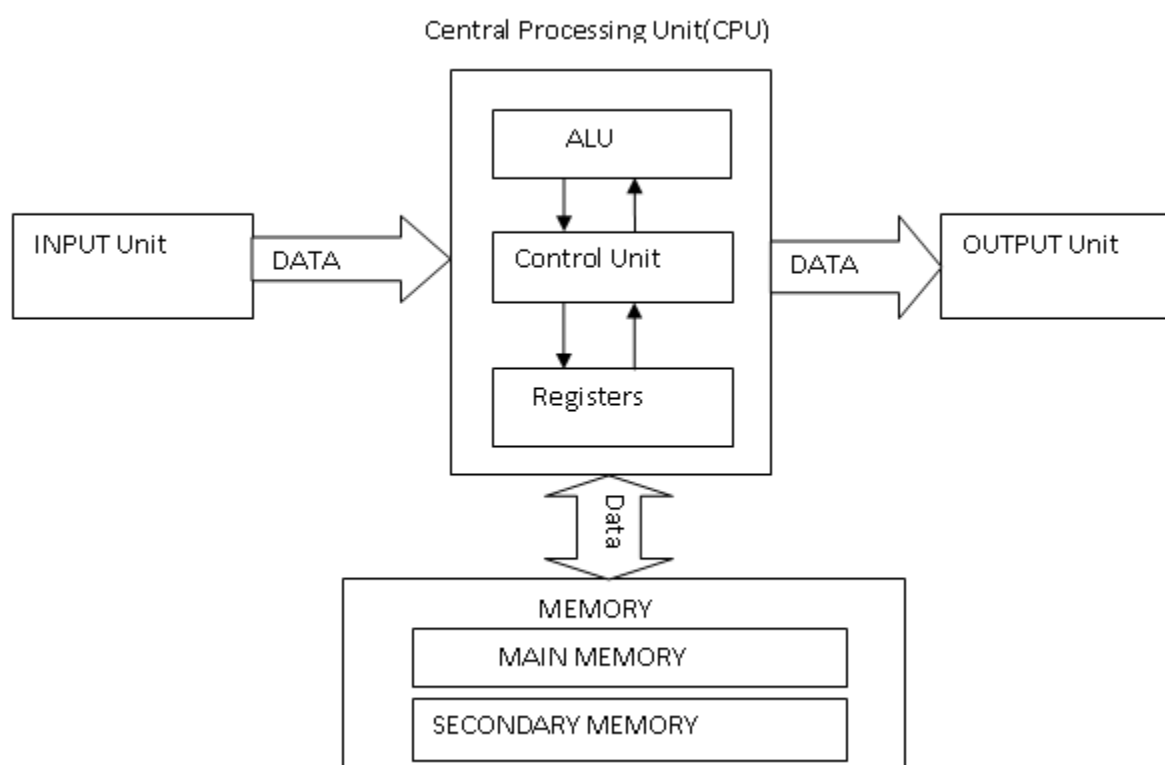
A computer can be defined as a fast electronic calculating machine that accepts the (data) digitized input information process it as per the list of internally stored instructions and produces the resulting information.

List of instructions are called programs & internal storage is called computer memory. The different types of computers are

1. **Personal computers:** - This is the most common type found in homes, schools, Business offices etc., It is the most common type of desk top computers with processing and storage units along with various input and output devices.
2. **Note book computers:** - These are compact and portable versions of PC
3. **Work stations:** - These have high resolution input/output (I/O) graphics capability, but with same dimensions as that of desktop computer. These are used in engineering applications of interactive design work.
4. **Enterprise systems:** - These are used for business data processing in medium to large corporations that require much more computing power and storage capacity than work stations. Internet associated with servers have become a dominant worldwide source of all types of information.
5. **Super computers:** - These are used for large scale numerical calculations required in the applications like weather forecasting etc.,

Functional Units:

A computer consists of five functionally independent main parts input, memory, arithmetic logic unit (ALU), output and control unit.



Input Unit:-

Computers take coded information via input unit. The most famous input device is keyboard. Whenever we press any key it is automatically being translated to corresponding binary code & transmitted over a cable to memory or processor.

Memory Unit:-

It stores programs as well as data and there are two types- Primary and Secondary Memory. Primary Memory is quite fast which works at electronic speed. Programs should be stored in memory before getting executed. Random Access Memory are those memory in which location can be accessed in a shorter period of time after specifying the address. Primary memory is essential but expensive so we went for secondary memory which is quite cheaper. It is used when large amount of data & programs are needed to store, particularly the information that we don't access very frequently. Ex- Magnetic Disks, Tapes

Arithmetic & Logic Unit:-

All the arithmetic & Logical operations are performed by ALU and this operation are initiated once the operands are brought into the processor.

Output Unit: – It displays the processed result to outside world.

Basic Operational Concepts

- Instructions take a vital role for the proper working of the computer.

- An appropriate program consisting of a list of instructions is stored in the memory so that the tasks can be started.

- The memory brings the Individual instructions into the processor, which executes the specified operations.

- Data which is to be used as operands are moreover also stored in the memory.

Example:

Add LOCA, R0

- This instruction adds the operand at memory location LOCA to the operand which will be present in the Register R0.

- The above mentioned example can be written as follows:

Load LOCA, R1

Add R1, R0

- First instruction sends the contents of the memory location LOCA into processor Register R0, and meanwhile the second instruction adds the contents of Register R1 and R0 and places the output in the Register R1.

The memory and the processor are swapped and are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals.

- The data is then transferred to or from the memory.

Analysing how processor and memory are connected:–

- Processors have various registers to perform various functions :-

- Program Counter: - It contains the memory address of next instruction to be fetched.

- Instruction Register:- It holds the instruction which is currently being executed.

MDR: - It facilitates communication with memory. It contains the data to be written into or read out of the addressed location.

MAR :- It holds the address of the location that is to be accessed

There are n general purpose registers that is R0 to Rn-1

Performance:-

Performance means how quickly a program can be executed.

In order to get the best performance it is required to design the compiler, machine instruction set & hardware in a coordinated manner.

Connection B / W Processor & Memory

Connection B/W Processor & Memory

The above mentioned block diagram consists of the following components

- 1) Memory
- 2) MAR
- 3) MDR
- 4) PC
- 5) IR
- 6) General Purpose Registers
- 7) Control Unit
- 8) ALU

The instruction that is currently being executed is held by the Instruction Register.

IR output is available to the control circuits, which generates the timing signal that control the various processing elements involved in executing the instruction.

The Memory address of the next instruction to be fetched and executed is contained by the Program Counter.

It is a specialized register.

It keeps the record of the programs that are executed.

Role of these registers is to handle the data available in the instructions. They store the data temporarily.

Two registers facilitate the communication with memory.

These registers are:

- 1) MAR (Memory Address Register)
- 2) MDR (Memory Data Register)

Memory Address Register:

The address of the location to be accessed is held by MAR.

Memory Data Register:

It contains the data to be written into or to be read out of the addressed location.

Working Explanation

A **PC** is set to point to the first instruction of the program. The contents of the **PC** are transferred to the **MAR** and a Read control signal is sent to the memory. The addressed word is fetched from the location which is mentioned in the **MAR** and loaded into **MDR**. This post thus contains all the important basic operational concepts.

Bus structures:

BUS: A group of lines(wires) that serves as a connecting path for several devices of a computer is called abus.

The following are different types of busses:

AddressBus 2. DataBus 3. Control Bus

The Data busCarries(transfer) data from one component (source) to other component (destination) connected to it. The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bi-directional. This means that CPU can read data on these lines from memory or from a port, as well as send data out on these lines to a memorylocation.

The Address busis the set of lines that carry(transfer) address information about where in memory the data is to be transferred to or from. It is an unidirectional bus. The address bus consists of 16, 20, 24 or more parallel signal lines. On these lines CPU sends out the address of the memory location.

The Control Buscarries the Control and timing information. Including these three the following are various types of busses. They are

System Bus: A System Bus is usually a combination of address bus, data bus, and control bus respectively.

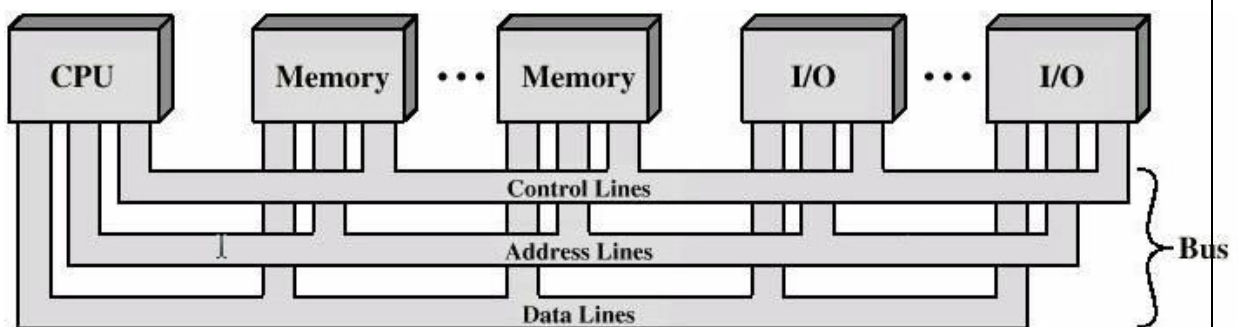
Internal Bus: The bus that operates only with the internal circuitary of the CPU.

External Bus: Buses which connects computer to external devices is nothing but external bus.

Back Plane: A Back Plane bus includes a row of connectors into which system modules can be plugged in.

I/O Bus: The bus used by I/O devices to communicate with the CPU is usually referred as I/O bus. Synchronous Bus: While using Synchronous bus, data transmission between source and destination units takes place in a given timeslot which is already known to these units.

Asynchronous Bus: In this case the data transmission is governed by a special concept. That is handshaking control signals.



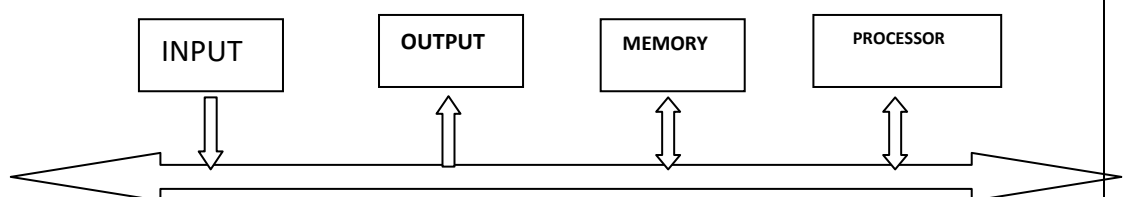
The Bus interconnection Scheme:-

Single bus structure :-

A group of lines(wires) that serves as a connecting path for several devices of a computer is called a bus.

In addition to the lines that carry the data, the bus must have lines for address and control purposes.

The simplest way to interconnect functional units is to use a single bus, as shown below.



All units are connected to this bus. Because the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time.

Bus control lines are used to arbitrate multiple requests for use of the bus.

ADVANTAGE:

Its is low cost and its flexibility for attaching peripheral devices

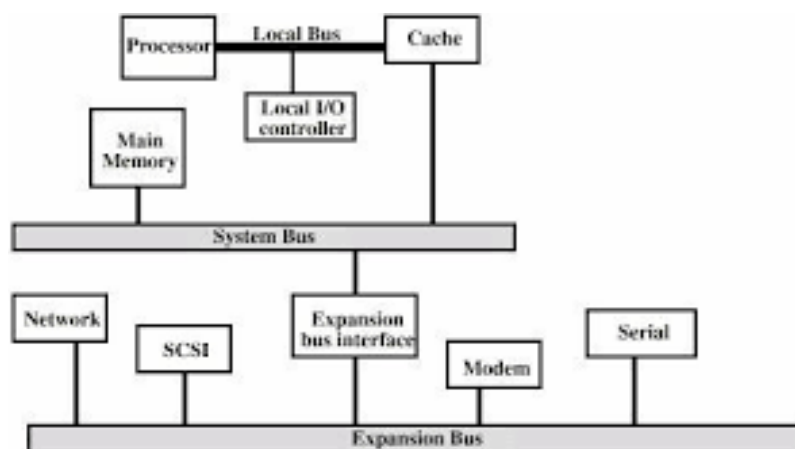
DISADVANTAGE:

low performance because at time only one transfer

Traditional / Multiple bus Structure: There is a local bus that connects the processor to cache memory and that may support one or more local devices. There is also a cache memory controller that connects this cache not only to this local bus but also to the system bus.

On the system, the bus is attached to the main memory modules. In this way, I/O transfers to and from the main memory across the system bus do not interfere with the processor's activity. An expansion bus interface buffers data transfers between the system bus and the I/O controllers on the expansion bus.

Some typical I/O devices that might be attached to the expansion bus include: Network cards (LAN), SCSI (Small Computer System Interface), Modem, Serial Com etc..



Performance

Performance: - The most important measure of the performance of a computer is how quickly it can compute programs. The speed with which a computer executes programs is affected by the design of its hardware and its machine language instructions. To represent the performance of a processor, we should consider only the periods during which the processor is active.

At the start of execution, all program instructions and the required data are stored in the memory as shown below. As execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache. When the execution of instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache. Later, if the same instruction or data item is needed a second time, it is read directly from the cache.

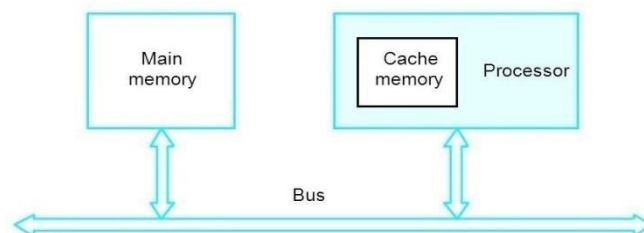


Figure 1.5. The processor cache.

Computer performance is often described in terms of clock speed (usually in MHz or GHz). This refers to the cycles per second of the main clock of the CPU. Performance of a computer depends on the following factors.

Processor clock:-

Processor circuits are controlled by a timing signal called a clock. A clock is a microchip that regulates speed and timing of all computer functions.

Clock Cycle is the speed of a computer processor, or CPU, which is the amount of time between two pulses of an oscillator. Generally speaking, the higher number of pulses per second, the faster the computer processor will be able to process information

CPU clock speed, or clock rate, is measured in Hertz — generally in gigahertz, or GHz. A CPU's clock speed rate is a measure of how many clock cycles a CPU can perform per second

To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.

The length P of one clock cycle is an important parameter that affects processor performance.

Its inverse is the clock rate, $R = 1/P$, which is measured in cycles per second.

If the clock rate is 500(MHz) million cycles per second, then the corresponding clock period is 2 nanoseconds.

Basic performance equation:-The Performance Equation is a term used in computer science. It refers to the calculation of the performance or speed of a central processing unit(CPU).

Basically the Basic Performance Equation [BPE] is an equation with 3 parameters which are required for the calculation of "Basic Performance" of a given system. It is given by

$$T = (N*S)/R$$

Where 'T' is the processor time [Program Execution Time] required to execute a given program written in some high level language .The compiler generates a machine language object program corresponding to the source program.

'N' is the total number of steps required to complete program execution. 'N' is the actual number of instruction executions, not necessarily equal to the total number of machine language instructions in the object program. Some instructions are executed more than others (loops) and some are not executed at all (conditions).

'S' is the average number of basic steps each instruction execution requires, where each basic step is completed in one clock cycle. We say average as each instruction contains a variable number of steps depending on the instruction.

'R' is the clock rate [In cycles per second]

Pipelining and Super scalar operation:-

A substantial improvement in performance can be achieved by overlapping the execution of successive instructions, using a technique called pipelining.

Consider the instruction

Add R1, R2, R3

Which adds the contents of registers R1 and R2, and places the sum into R3

The contents of R1 and R2 are first transferred to the inputs of the ALU.

After the add operation is performed, the sum is transferred to R3.

Processor can read the next instruction from the memory while the addition operation is being performed.

Then, if that instruction also uses the ALU, its operands can be transferred to the ALU inputs at the same time that the result of add instruction is being transferred to R3.

Thus, pipelining increases the rate of executing instructions significantly.

Super scalar operation:-

A higher degree of concurrency can be achieved if multiple instruction pipelines are implemented in the processor.

This means that multiple function units are used, creating parallel paths through which different instructions can be executed in parallel.

With such an arrangement, it becomes possible to start the execution of several instructions in every clock cycle.

This mode of execution is called super scalar operation.

Clock rate:-

There are two possibilities for increasing the clock rate, R .

First, improving the Integrated Circuit technology makes logic circuit faster, which reduces the time needed to complete a basic step. This allows the clock period, P , to be reduced and the clock rate, R , to be increased.

Second, reducing the amount of processing done in one basic step also makes it possible to reduce the clock period, P .

Instruction set: CISC and RISC:-

The terms CISC and RISC refer to design principles and techniques.

RISC: Reduced instruction set computers.

Simple instructions require a small number of basic steps to execute.

For a processor that has only simple instructions, a large number of instructions may be needed to perform a given programming task. This could lead to a large value of N and a small value for S .

It is much easier to implement efficient pipelining in processors with simple instruction sets.

CISC: Complex instruction set computers.

Complex instructions involve a large number of steps.

If individual instructions perform more complex operations, fewer instructions will be needed, leading to a lower value of N and a larger value of S .

Complex instructions combined with pipelining would achieve good performance.

Optimizing Compiler:-

A compiler translates a high-level language program into a sequence of machine instructions.

To reduce N, we need to have a suitable machine instruction set and a compiler that makes good use of it.

An optimizing compiler takes advantage of various features of the target processor to reduce the product N * S.

The compiler may rearrange program instructions to achieve better performance.

Performance measurement:-

SPEC rating.

A nonprofit organization called "System Performance Evaluation Corporation" (SPEC) selects and publishes representative application programs for different application domains.

The SPEC rating is computed as follows.

SPEC rating = Running time on the reference computer

Running time on the computer under test.

Thus SPEC rating of 50 means that the computer under test is 50 times faster than the reference computer for these particular benchmarks.

The test is repeated for all the programs in the SPEC suite, and the geometric mean of the results is computed.

Let SPEC_i be the rating for program 'i' in the suite. The overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \sqrt[n]{\prod_{i=1}^n (\text{SPEC}_i)}$$

Where n is the number of programs in the suite.

Memory Locations:

Memory locations and addresses are fundamental concepts in computer architecture and programming. They refer to the physical or logical locations where data and instructions are stored in a computer's memory.

Memory Location: A memory location is a specific location in the computer's memory where data can be stored or retrieved. Each memory location has a unique identifier known as an address, which helps the computer's processor access and manipulate the data stored in that location.

Memory Address: A memory address is a numerical value that represents the location of data in memory. It is used by the processor to read or write data to/from a specific memory location. Memory addresses are typically expressed in hexadecimal notation, and each address corresponds to a byte (or a group of bytes) in memory.

For example, let's say a computer has 8 bits of memory (very simplified scenario for illustration purposes). Each bit can store either 0 or 1, giving us 256 possible combinations (2^8). The memory locations are numbered from 0 to 255, and each location contains 1 byte (8 bits) of data.

In this case, the memory addresses would range from 0x00 to 0xFF (in hexadecimal) or from 0 to 255 (in decimal). To access the data stored at a specific memory location, the processor sends the memory address along with a read or write signal to the memory unit.

When a program is executed, its instructions and data are loaded into specific memory locations. The program counter, a special register, keeps track of the memory address of the instruction being executed, allowing the processor to fetch the next instruction in sequence.

The concept of memory locations and addresses is crucial for understanding how computers store and retrieve data during program execution, making it an essential aspect of computer science and programming.

Memory Operations:

Memory operations refer to the actions or processes involved in reading from or writing to computer memory. In the context of computer systems, memory refers to the storage space where data and instructions are temporarily stored during the execution of programs. Memory operations play a crucial role in the functioning of a computer system, and they are essential for the execution of any program.

There are several common memory operations:

Reading: During the execution of a program, the CPU (Central Processing Unit) needs to fetch data or instructions from memory to process them further. This process involves reading the content of a specific memory location and transferring it to the CPU for processing.

Writing: When data needs to be stored or updated in memory, the CPU performs a write operation. The CPU sends the data to the specific memory location, and it gets stored there, replacing the previous content if any.

Load: A load operation is used to fetch data from memory and transfer it into a CPU register, where it can be manipulated or used by the CPU. Loading data from memory into registers is faster than accessing data directly from memory.

Store: A store operation is used to take data from a CPU register and write it to a specific memory location, effectively storing the data in memory.

Copy: Copying data from one memory location to another involves reading the data from the source location and writing it to the destination location. This is a common operation used in various algorithms and data manipulation tasks.

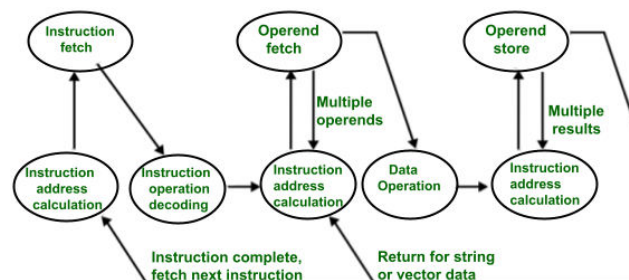
Move: Similar to copying, moving data also involves transferring data from one memory location to another. However, in this case, the data is removed from the source location after being transferred to the destination.

Allocate and Deallocate: Memory operations include allocating memory to a program or data structure when it is needed and deallocating it when it is no longer required. Memory allocation and deallocation are crucial for managing memory efficiently and avoiding memory leaks.

Initialize: Initializing memory means setting its content to a predefined value, typically 0 or null. This is often done to ensure that the memory is in a known state before storing data in it.

Memory operations are fundamental to the proper functioning of computer programs and the overall performance of a computer system.

Instruction and instruction sequencing:



Instruction and instruction sequencing are fundamental concepts in the field of computer science and computer architecture. They play a crucial role in how computers execute tasks and process data.

Instruction: In the context of computing, an instruction is a basic operation or command that a computer processor can understand and execute. Instructions are the building blocks of programs and represent specific tasks that the processor can perform, such as arithmetic operations (addition, subtraction, etc.), logical operations (AND, OR, NOT), data movement (load and store), and control flow (branching and jumping).

Instructions are encoded in binary form and are represented as sequences of 0s and 1s, which the processor can interpret and execute. Each type of processor or CPU (Central Processing Unit) has its own instruction set, which is a specific collection of instructions it can understand and execute.

Instruction Sequencing: Instruction sequencing refers to the order in which instructions are executed by the computer processor. In a typical computer program, a series of instructions are arranged in a specific order to achieve the desired task or computation. The processor fetches, decodes, and executes instructions one by one in the specified sequence.

The instruction sequencing is controlled by the program counter (PC), a special register in the CPU that holds the memory address of the next instruction to be executed. During the

execution of a program, the PC is incremented, and the processor fetches the instruction from the memory location pointed by the PC, decodes it to understand the operation, and then executes it. This process continues until the program is completed.

The ability to sequence instructions correctly is crucial to ensure that a program executes the desired tasks accurately and efficiently. Incorrect sequencing or flow control can lead to program errors and unintended behaviors.

Instruction sequencing is also affected by control flow instructions like branches and jumps, which allow the program to change its flow and execute instructions conditionally or non-sequentially based on certain conditions.

In summary, instructions and instruction sequencing are fundamental concepts in computer architecture that enable computers to execute tasks and process data in a structured and organized manner. They are essential to the functioning of a computer and the execution of computer programs.

Addressing modes:

In computer architecture, an addressing mode is a technique used by a processor to specify the location of operands in memory during instruction execution. The choice of addressing mode can have a significant impact on the efficiency and flexibility of a computer's instruction set architecture. Different addressing modes are designed to handle various memory access requirements and optimize the execution of specific instructions. Here are some common addressing modes:

Addressing Modes:



- ☞ Immediate
- ☞ Direct
- ☞ Indirect
- ☞ Register
- ☞ Register Indirect
- ☞ Displacement
- ☞ Stack

Immediate Addressing: The operand value is directly specified within the instruction itself. For example, "MOV R1, #10" means move the value 10 into register R1.

Register Addressing: The operand is stored in a register. For example, "ADD R2, R3" means add the value in register R3 to the value in register R2.

Direct Addressing: The memory address of the operand is explicitly given in the instruction. For example, "LOAD R1, 0x1000" means load the value from memory address 0x1000 into register R1.

Indirect Addressing: The memory address of the operand is stored in a register. The instruction accesses the value in the memory address pointed to by the register. For example, "LOAD R1, (R2)" means load the value from the memory address stored in register R2 into register R1.

Register indirect addressing mode is a concept used in computer architecture and assembly language programming. It is one of the addressing modes that processors use to access data stored in memory.

Displacement addressing mode is a type of addressing mode used in computer architectures and assembly languages. In this addressing mode, the effective address of an operand is calculated by adding a constant displacement value to the base address or index register value.

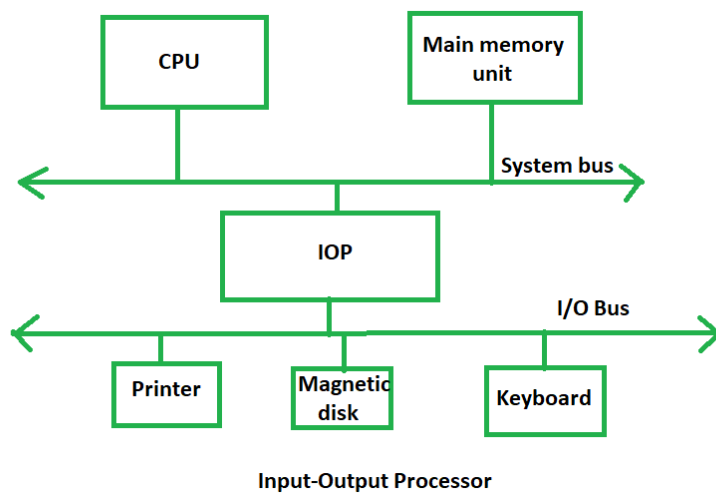
Effective Address = Base Address (or Index Register Value) + Displacement

Stack Addressing: The operand is implicitly at the top of the stack. This mode is often used for subroutine calls and managing function parameters and local variables.

Basic Input and output operations:

Basic Input/output (I/O) operations in computer architecture refer to the processes of transferring data between the central processing unit (CPU) and external devices, such as keyboards, monitors, hard drives, printers, and network interfaces. These operations allow the computer to interact with its surroundings and enable users to input data and receive output from the system.

There are two main types of I/O operations:



Input Operations: Input operations involve transferring data from external devices to the computer's memory or registers for processing. Common input devices include keyboards, mice, scanners, and sensors. When you press a key on the keyboard or move the mouse, the corresponding data is sent to the CPU for further processing. The CPU then responds based on the input data received.

Output Operations: Output operations involve transferring data from the computer's memory or registers to external devices for display or storage. Common output devices include monitors, printers, speakers, and storage devices (e.g., hard drives, USB drives). The CPU sends data to these devices to display information on the screen or print documents, among other tasks.

The process of I/O operations typically involves the following steps:

Initiation: The computer's operating system or software initiates the I/O operation by sending a request to read or write data to/from an external device.

Data Transfer: The data is transferred between the CPU and the external device through specialized I/O channels or buses.

Control: The CPU communicates with the device controller to coordinate the data transfer and manage the I/O process.

Interrupts: In many cases, I/O operations are slower than CPU operations. To avoid wasting CPU cycles while waiting for I/O to complete, the CPU uses interrupts. An interrupt is a signal that interrupts the normal flow of the CPU, allowing it to handle the I/O request and then return to its previous task.

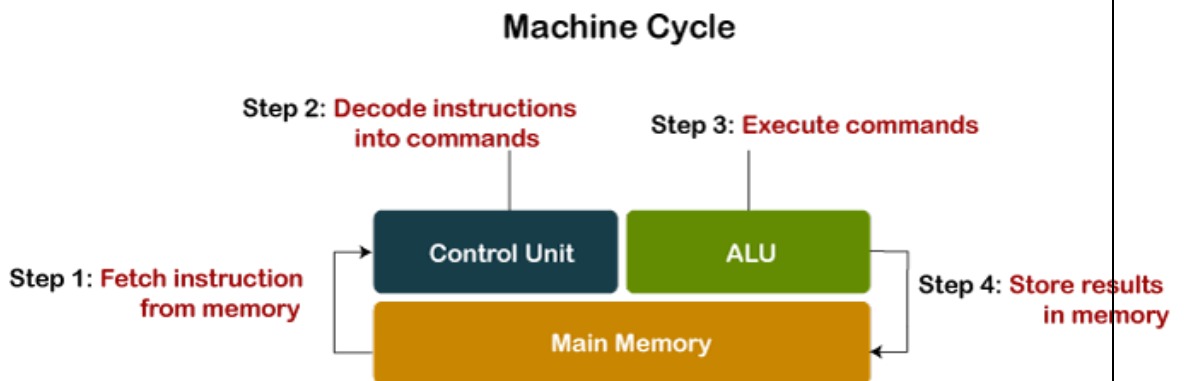
Modern computers use various techniques to optimize I/O operations, such as buffering, direct memory access (DMA), and I/O scheduling algorithms. These techniques aim to reduce the overall time spent on I/O operations and improve system performance.

Unit 2

ARITHMETIC FOR COMPUTER

INTRODUCTION:

An arithmetic unit, or ALU, enables computers to perform mathematical operations on binary numbers. They can be found at the heart of every digital computer and are one of the most important parts of a CPU (Central Processing Unit). This note explores their basic function, anatomy and history.



The operations performed by ALU are:

Logical Operations: The logical operations consist of NOR, NOT, AND, NAND, OR, XOR, and more.

Bit-Shifting Operations: It is responsible for displacement in the locations of the bits to the by right or left by a certain number of places that are known as a multiplication operation.

Arithmetic Operations: Although it performs multiplication and division, this refers to bit addition and subtraction. But multiplication and division operations are more costly to make. In the place of multiplication, addition can be used as a substitute and subtraction for division.

Addition and Subtraction of Signed Numbers:

A signed-magnitude method is used by computers to implement floating-point operations. Signed-2's complement method is used by most computers for arithmetic operations executed on integers. In this approach, the leftmost bit in the number is used for signifying the sign; 0 indicates a positive integer, and 1 indicates a negative integer. The remaining bits in the number supported the magnitude of the number.

Example: -2410 is defined as –

10011000

In this example, the leftmost bit 1 defines negative, and the magnitude is 24.

The magnitude for both positive and negative values is the same, but they change only with their signs.


The range of values for the sign and magnitude representation is from **-127** to **127**.

There are eight conditions to consider while adding or subtracting signed numbers

Addition and Subtraction of Signed Magnitude Numbers :

	Addition of Magnitudes	Subtraction of Magnitudes		
		P>Q	P<Q	P=Q
(+P) + (+Q)	+(P+Q)			
(+P) + (-Q)		+(P-Q)	-(Q-P)	+(P-Q)
(-P) + (+Q)		-(P-Q)	+(Q-P)	+(P-Q)
(-P) + (-Q)	-(P+Q)			
(+P) - (+Q)		+(P-Q)	-(Q-P)	+(P-Q)
(+P) - (-Q)	+(P+Q)			

	Addition of Magnitudes	Subtraction of Magnitudes		
$(-P) - (+Q)$	$-(P+Q)$			
$(-P) - (-Q)$		$-(P-Q)$	$+(Q-P)$	$+(P-Q)$



Examples

Example of adding two magnitudes when the result is the sign of both operands:

$$\begin{array}{r}
 +3 \ 0 \ 011 \\
 + +2 \ 0 \ 010 \\
 \hline
 +5 \ 0 \ 101
 \end{array}$$

Example of adding two magnitudes when the result is the sign of the larger magnitude:

$$\begin{array}{r}
 -3 \ 1 \ 011 \\
 + +2 \ 0 \ 010 \\
 \hline
 = (-+3 \ 011) \\
 = (+2) \ 010 \\
 \hline
 -1 \ 1 \ 001
 \end{array}$$

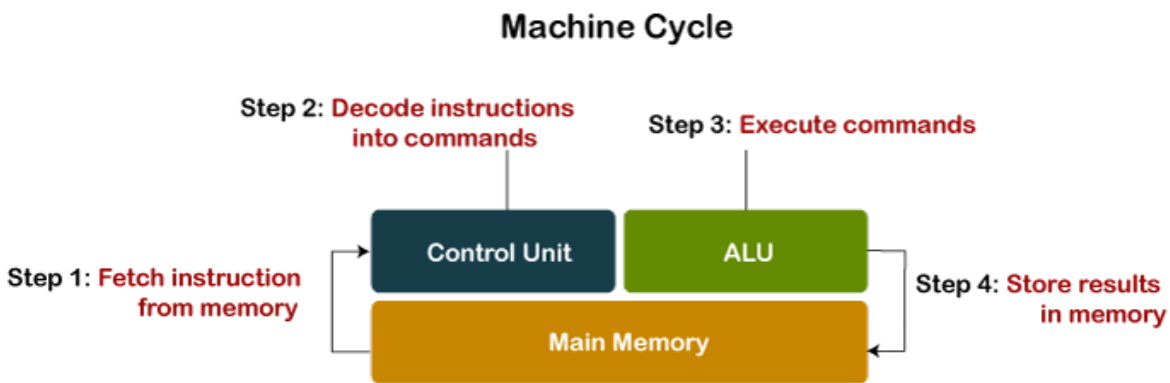
UNIT-II

ARITHMETIC UNIT

Addition and Subtraction of Signed Numbers - Design of Fast Adders - Multiplication of Positive Numbers - Signed Operand Multiplication - Fast Multiplication - Integer Division - Floating Point Numbers and Operations. Positive Numbers and Operations.

INTRODUCTION:

An arithmetic unit, or ALU, enables computers to perform mathematical operations on binary numbers. They can be found at the heart of every digital computer and are one of the most important parts of a CPU (Central Processing Unit). This note explores their basic function, anatomy and history.



The operations performed by ALU are:

- **Logical Operations:** The logical operations consist of NOR, NOT, AND, NAND, OR, XOR, and more.
- **Bit-Shifting Operations:** It is responsible for displacement in the locations of the bits to the by right or left by a certain number of places that are known as a multiplication operation.
- **Arithmetic Operations:** Although it performs multiplication and division, this refers to bit addition and subtraction. But multiplication and division operations are more costly to make. In the place of multiplication, addition can be used as a substitute and subtraction for division.

Addition and Subtraction of Signed Numbers:

A signed-magnitude method is used by computers to implement floating-point operations. Signed-2's complement method is used by most computers for arithmetic operations executed on integers. In this approach, the leftmost bit in the number is used for signifying the sign; 0 indicates a positive integer, and 1 indicates a negative integer. The remaining bits in the number supported the magnitude of the number.

Example: -2410 is defined as –

10011000

In this example, the leftmost bit 1 defines negative, and the magnitude is 24.

The magnitude for both positive and negative values is the same, but they change only with their signs.

The range of values for the sign and magnitude representation is from **-127** to **127**.

There are eight conditions to consider while adding or subtracting signed numbers

Addition and Subtraction of Signed Magnitude Numbers :

	Addition of Magnitudes	Subtraction of Magnitudes		
		P>Q	P<Q	P=Q
(+P) + (+Q)	+(P+Q)			
(+P) + (-Q)		+(P-Q)	-(Q-P)	+(P-Q)
(-P) + (+Q)		-(P-Q)	+(Q-P)	+(P-Q)
(-P) + (-Q)	-(P+Q)			
(+P) - (+Q)		+(P-Q)	-(Q-P)	+(P-Q)
(+P) - (-Q)	+(P+Q)			
(-P) - (+Q)	-(P+Q)			
(-P) - (-Q)		-(P-Q)	+(Q-P)	+(P-Q)

Examples

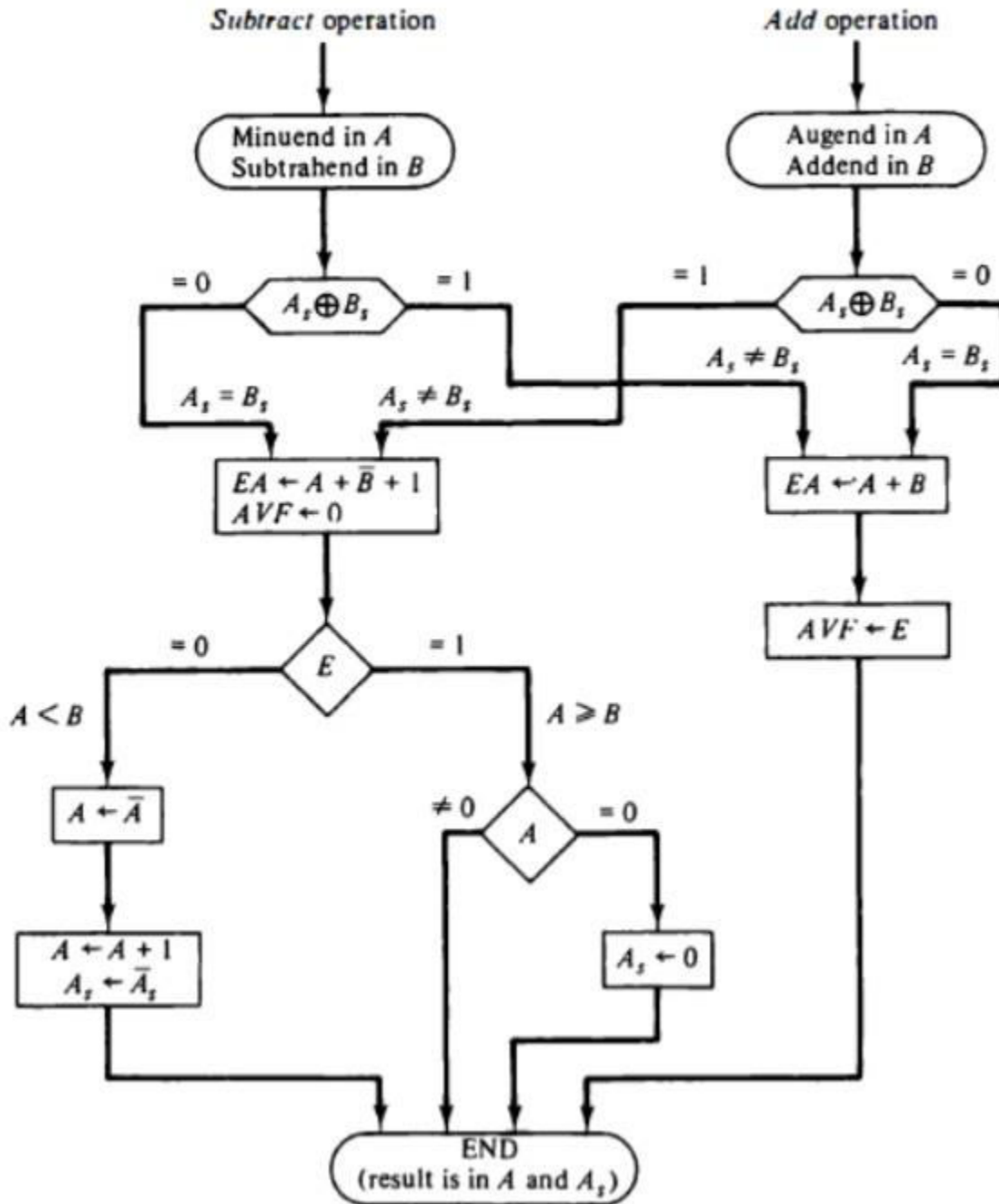
Example of adding two magnitudes when the result is the sign of both operands:

$$\begin{array}{r}
 +3 \ 0 \ 011 \\
 + \ +2 \ 0 \ 010 \\
 \hline
 +5 \ 0 \ 101
 \end{array}$$

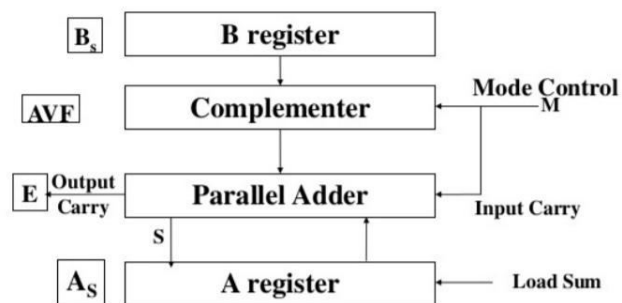
Example of adding two magnitudes when the result is the sign of the larger magnitude:

$$\begin{array}{r}
 -3 \ 1 \ 011 \\
 + \ +2 \ 0 \ 010 \\
 \hline
 = (-) \ +3 \ 011 \\
 = \ +2 \ 010 \\
 \hline
 -1 \ 1 \ 001
 \end{array}$$

Flowchart of Addition and Subtraction with signed Magnitude Data:

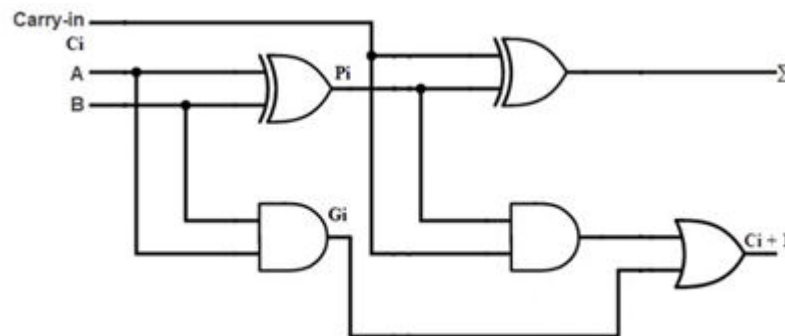


Hardware for signed magnitude addition and subtraction:



Design of Fast Adders :

- A carry-look ahead adder (CLA) or fast adder is a type of adder used in digital logic.
- A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits.
- It can be contrasted with the simpler, but usually slower, ripple-carry adder (RCA), for which the carry bit is calculated alongside the sum bit, and each stage must wait until the previous carry bit has been calculated to begin calculating its own sum bit and carry bit.
- The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger-value bits of the adder
- A carry-Lookahead adder is a fast parallel adder as it reduces the propagation delay by more complex hardware, hence it is costlier.
- This method makes use of logic gates so as to look at the lower order bits of the augend and addend to see whether a higher order carry is to be generated or not.

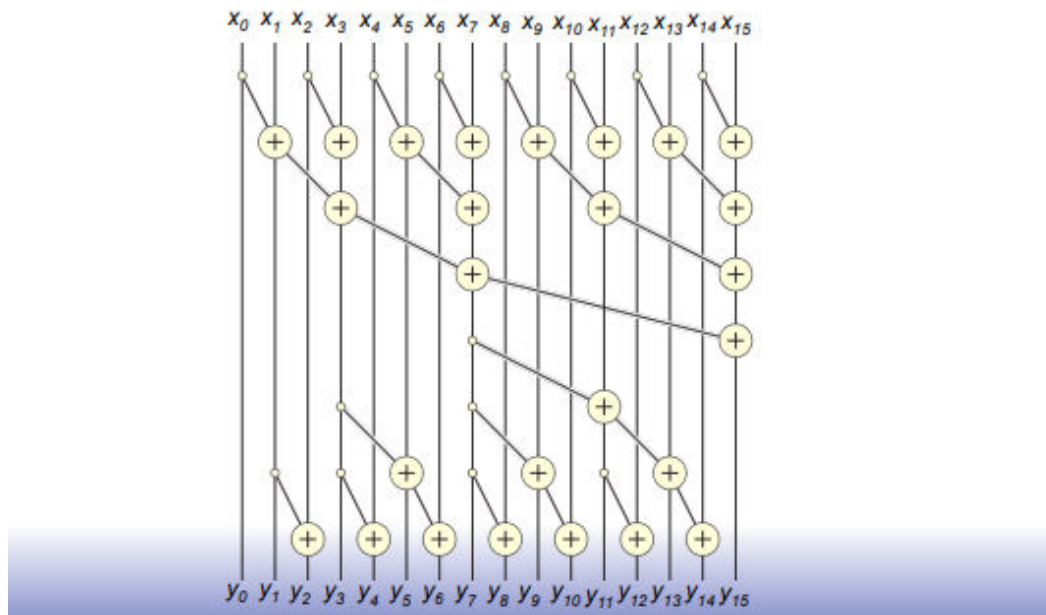


Contents for fast addition are,

1. Carry Look Ahead Adder
2. Parallel Prefix Adder trees
3. Carry Skip Adder
4. Carry Increment Adder
5. Carry Select Adders
6. Carry Save Addition
7. BCD Addition

Parallel Prefix, or "Scan"

- If "+" is an associative operator, and x_0, \dots, x_{p-1} are input data then parallel prefix operation computes: $y_j = x_0 + x_1 + \dots + x_j$ for $j=0,1,\dots,p-1$



16-bit adder Figure is the parallel prefix graph of a HanCarlson adder.

This adder has a hybrid design combining stages from the Brent-Kung and KoggeStone adder. The Han-Carlson adder is efficient and suitable for VLSI implementation.

Carry skip adders:

A carry-skip adder consists of a simple ripple carry-adder with a special speed up carry chain called a skip chain. This chain defines the distribution of ripple carry blocks, which compose the skip adder. The addition of two binary digits at stage i , where i is not equal to 0, of the ripple carry adder depends on the carry in, C_i , which in reality is the carry out, C_{i-1} , of the previous stage. Therefore, in order to calculate the sum and the carry out, C_{i+1} , of stage i , it is imperative that the carry in, C_i , be known in advance.

It is interesting to note that in some cases C_{i+1} can be calculated without knowledge of C_i .

Boolean Equations of a Full Adder:

$P_i = A_i \oplus B_i$ Equ. 1 --carry propagate of i th stage

$S_i = P_i \oplus C_i$ Equ. 2 --sum of i th stage

$C_{i+1} = A_i B_i + P_i C_i$ Equ. 3 --carry out of i th stage

Supposing that $A_i = B_i$, then P_i in equation 1 would become zero (equation 4). This would make C_{i+1} to depend only on the inputs A_i and B_i , without needing to know the value of C_i .

$A_i = B_i ; P_i = 0$ Equ. 4 --from #Equation 1

If $A_i = B_i = 0 ; C_{i+1} = A_i B_i = 0$ --from equation 3

If $A_i = B_i = 1 ; C_{i+1} = A_i B_i = 1$ --from equation 3

Therefore, if Equation 4 is true then the carry out, C_{i+1} , will be one if $A_i = B_i = 1$ or zero if $A_i = B_i = 0$. Hence the output can be computed with the carry out at any stage of the addition provided equation 4 holds. These would enable to build an adder whose average time of computation would be proportional to the longest chains of zeros and of different digits of A and B.

Carry Increment Adder:

B. Carry Increment Adder (CIA)

The design of Carry Increment Adder (CIA) consists of RCA's and incremental circuitry. The incremental circuit can be designed using HA's in ripple carry chain with a sequential order. The addition operation is done by dividing total number of bits in to group of 4bits and addition operation is done using several 4bit RCA's. The architecture of CIA is shown in fig. 6 .

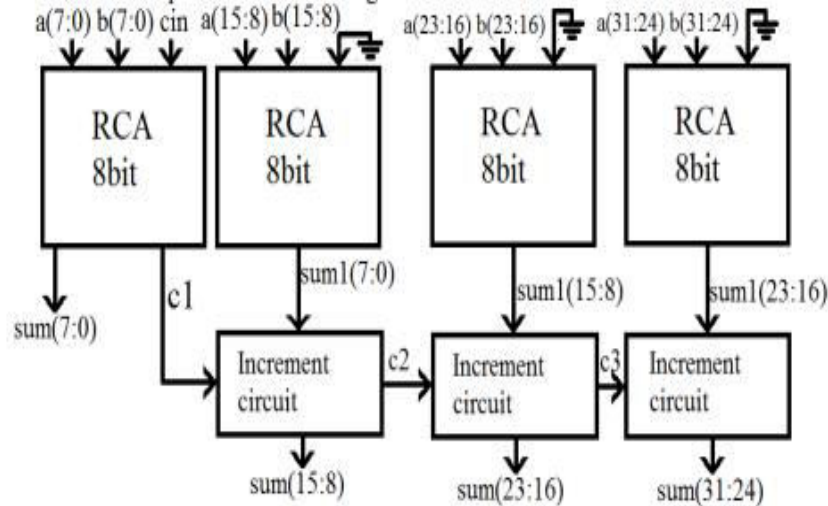
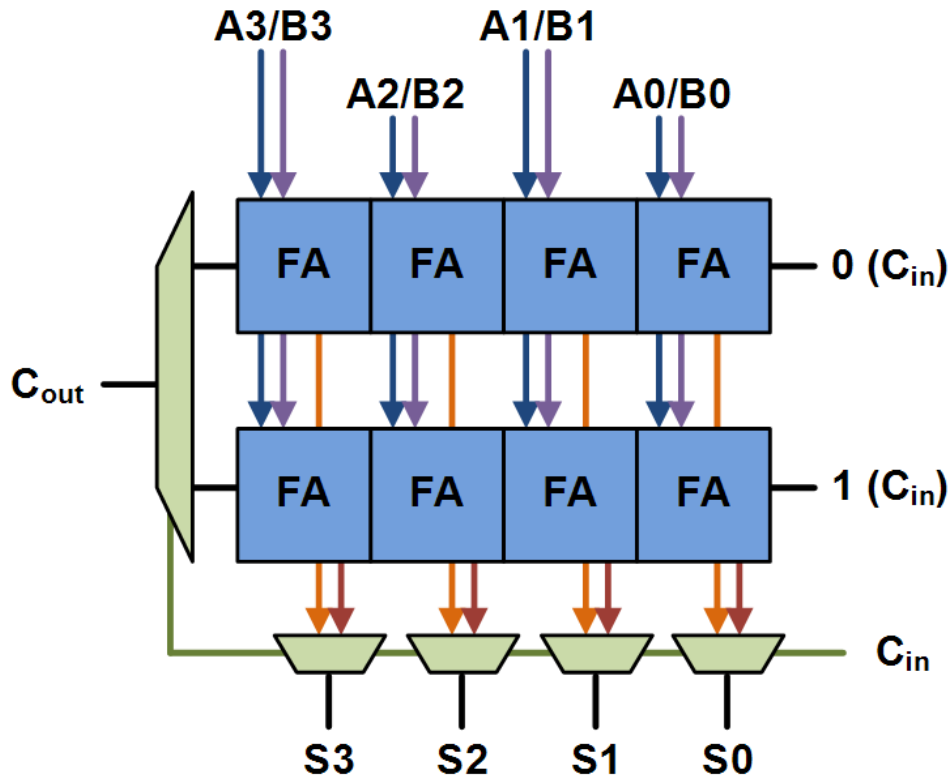


Figure 6. 32-Bit Carry Increment Adder (CIA)

In carry increment adder, various RCA blocks are used to compute the results. The first RCA block is given carry-in as input along with addends to get carry (c_1) and sum. For the rest RCA blocks, the carry-in is given as logic '0' to get temporary sum (sum_1) and temporary carry which are given to increment circuit. Increment circuit consists of half adders which add temporary sum and carries to get the actual sum (sum) and carry (c_y). The carry-out of increment circuit is obtained by performing OR operation between carry (c_y) and carry-out of the previous stage. As the carry-in for the RCA stages is logic '0', and hence the carry propagation delay decreases.

Carry-select adder:



Above is the basic building block of a carry-select adder, where the block size is 4. Two 4-bit ripple-carry adders are multiplexed together, where the resulting carry and sum bits are selected by the carry-in. Since one ripple-carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result.

Carry-save adder:

CSA is a type of digital adder, used to efficiently compute the sum of three or more binary numbers. It differs from other digital adders in that it outputs two (or more) numbers, and the answer of the original summation can be achieved by adding these outputs together. A carry save adder is typically used in a binary multiplier, since a binary multiplier involves addition of more than two binary numbers after multiplication.

A big adder implemented using this technique will usually be much faster than conventional addition of those numbers.

BCD Addition:

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD or **Binary Coded Decimal** is that number system or code which has the binary numbers or digits to represent a decimal number.

A decimal number contains 10 digits (0-9). Now the equivalent binary numbers can be found out of these 10 decimal numbers. In case of **BCD** the binary number formed by four binary digits, will be the equivalent code for the given decimal digits. In **BCD** we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively.

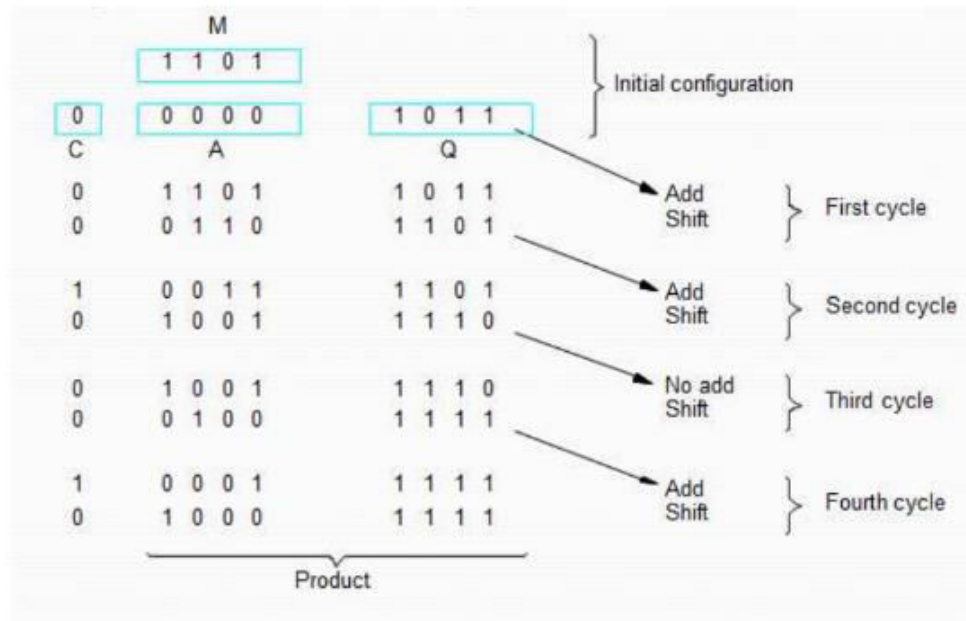
Multiplication of Positive Numbers:

MULTIPLICATION OF POSITIVE NUMBERS

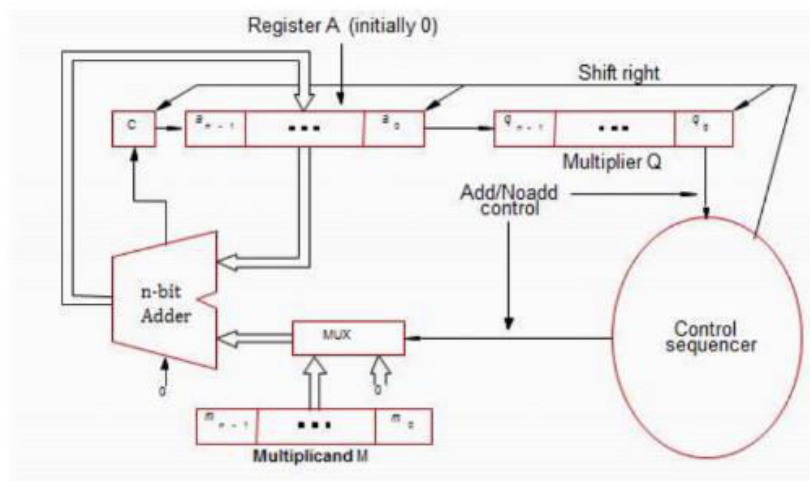
The usual algorithm for multiplying integers by hand is illustrated in Figure: 1 for the binary system.

1 1 0 1	(13) Multiplicand M
1 0 1 1	(11) Multiplier Q
<hr/>	
1 1 0 1	
1 1 0 1	
0 0 0 0	
1 1 0 1	
<hr/>	
1 0 0 0 1 1 1 1	(143) Product P

Figure: 1 Manual multiplication algorithm



The simplest way to perform multiplication is to use the adder circuitry in the ALU for a number of sequential steps. The block diagram in Figure: 3 shows the hardware arrangement for sequential multiplication.



Signed Operand Multiplication:

Multiplication of two fixed point binary number in *signed magnitude representation* is done with process of *successive shift and add operation*.

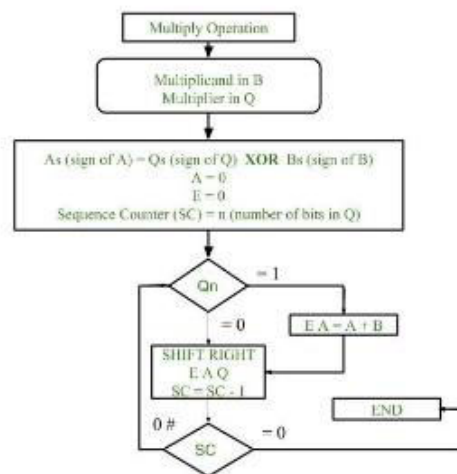
$$\begin{array}{r}
 10111 \text{ (Multiplicand)} \\
 \times 10011 \text{ (Multiplier)} \\
 \hline
 10111 \\
 10111 \\
 00000 \\
 00000 \\
 10111 \\
 \hline
 011011010 \text{ (Product)}
 \end{array}$$

In the multiplication process we are considering successive bits of the multiplier, least significant bit first.

If the multiplier bit is 1, the multiplicand is copied down else 0's are copied down.

The numbers copied down in successive lines are shifted one position to the left from the previous number.

Flowchart of Multiplication:



1. Initially multiplicand is stored in B register and multiplier is stored in Q register.
2. Sign of registers B (Bs) and Q (Qs) are compared using **XOR** functionality (i.e., if both the signs are alike, output of XOR operation is 0 unless 1) and output stored in As (sign of A register).

Note: Initially 0 is assigned to register A and E flip flop. Sequence counter is initialized with value n, n is the number of bits in the Multiplier.

3. Now least significant bit of multiplier is checked. If it is 1 add the content of register A with Multiplicand (register B) and result is assigned in A register with carry bit in flip flop E.

position, i.e., content of E is shifted to most significant bit (MSB) of A and least significant bit of A is shifted to most significant bit of Q.

4. If $Q_n = 0$, only shift right operation on content of E A Q is performed in a similar fashion.
5. Content of Sequence counter is decremented by 1.
6. Check the content of Sequence counter (SC), if it is 0, end the process and the final product is present in register A and Q, else repeat the process.

Example:

Multiplicand = 10111
Multiplier = 10011

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q Qn = 1; add B	0	0000 1011	10011	101
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Qn = 1; add B	0	10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Qn = 0; shift right EAQ	0	01000	10110	010
Qn = 0; shift right EAQ	0	00100	01011	001
Qn = 1; add B	0	10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000

Final product in AQ
0110110101

Fast Multiplication:

There are two techniques discussed for speeding the Multiplication Operation

- i. **Reducing Maximum number of Summands:** - Bit Pair Recoding of Multipliers reduces the maximum number of summands (versions of multiplicand) that must be added to $n/2$ for n bit operands[1].
- ii. **Faster Summands addition:** - Summand addition uses
 - a) Carry save – In this carry generated by Full Adders in i^{th} row propagated to $(i+1)^{th}$ row Full Adders instead of rippling through adder in same row and
 - b) Summands Reduction Technique - Techniques like 3 to 2, 4 to 2 reduction of summands [1].

i. Reducing Maximum number of Summands using Bit Pair Recoding of Multipliers

Bit-pair recoding of the multiplier – It is a modified Booth Algorithm, In this it uses one summand for each pair of booth recoded bits of the multiplier.

Step 1: Convert the given Multiplier into a Booth Recode the Multiplier.

Activate Windows

Step 2: Group the recoded Multiplier bits in pairs and observe the following

Go to Settings to activate

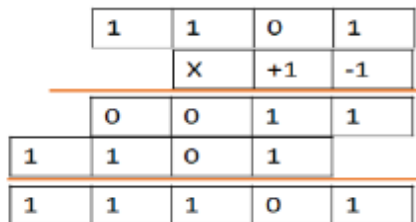
For example - If the pair is (+1 -1) - It is equivalent to the pair (0 +1).

Reason: A pair (+1 -1) means adding (-1) time the Multiplicand M at shifted position i with (+1) times Multiplicand at shifted position (i+1) is equivalent to a pair (0 +1) which adds +1 times Multiplicand at position i.

$$\text{A pair (+1 -1)} = (2^1 \times M - 2^0 \times M) = (2M - 1M) = +1M$$

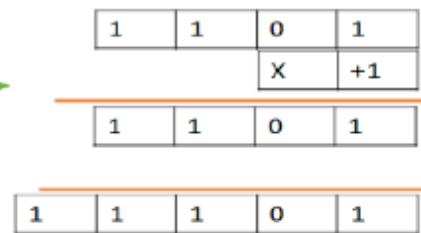
Let us say Multiplicand = (1 1 0 1) and Bit pair (+1 -1)

a) By Booth Recoding



By Bit Pairing →

b) By Bit Pairing

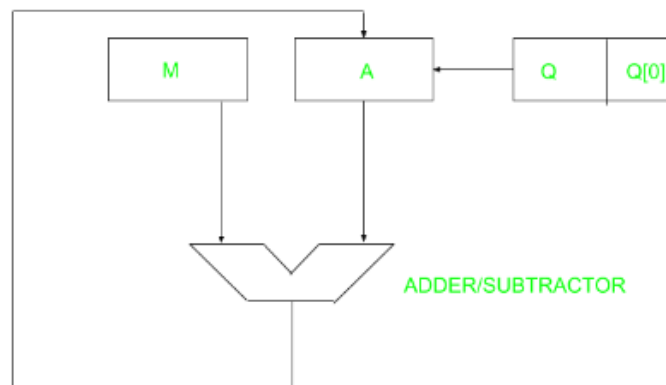


By Sign Extension ->

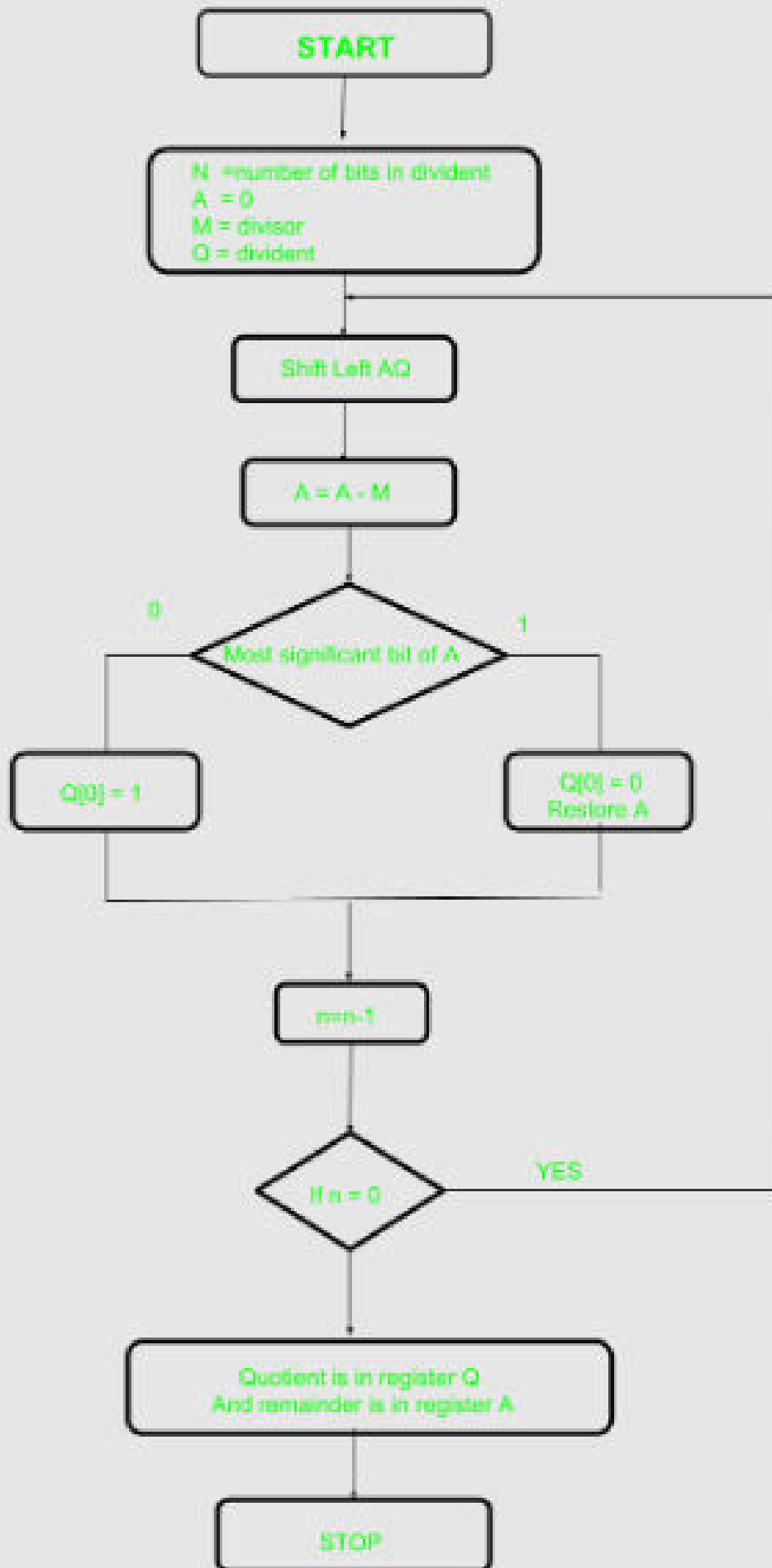
Integer Division:

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**. Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

In this article, will be performing restoring algorithm for unsigned integer. Restoring term is due to fact that value of register A is restored after each iteration.



Here, register Q contain quotient and register A contain remainder. Here, n-bit



Let's pick the step involved:

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Then the content of register A and Q is shifted left as if they are a single unit
- **Step-3:** Then content of register M is subtracted from A and result is stored in A
- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
- **Step-5:** The value of counter n is decremented
- **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder

Floating Point Numbers and Operations:

The floating-point representation can implement operations for high range values. The numerical evaluations are carried out using floating-point values. It can create calculations easy, scientific numbers are described as follows –

The number 5,600,000 can be described as $0.56 * 10^7$.

Therefore, 0.56 is the mantissa and 7 is the value of the exponent.

Binary numbers can also be described in exponential form. The description of binary numbers in the exponential form is called floating-point representation. The floating-point representation breaks the number into two parts, the left-hand side is a signed, fixed-point number known as a mantissa and the right-hand side of the number is known as the exponent. The floating-point values are also authorized with a sign; 0 denoting the positive value and 1 denoting the negative value.

The general structure of floating-point representation of a binary number –

$$x = (x_0 * 2^0 + x_1 * 2^1 + x_2 * 2^2 \pm \dots \mp b_{-(n-1)} * 2^{-(n-1)})$$

mantissa * 2^{Exponent}

In the following syntax, the decimal point is transferred left for negative exponents of two and right for positive exponents of two. Both the mantissa and the exponent are signed values enabling negative numbers and negative exponents commonly.

Example – Convert 111101.1000110 into floating-point value.

$111101.1000110 = 1.111011000110 * 2^5$ Converted to floating-point value

→ Denotes negative sign value

In this example, the integer value is converted to a floating-point value by changing the radix point next to the signed integer and scaling up the number to the exponential form by multiplying the value with the base 2. The value remains unaltered and this phase is known as the normalized method.

*_*_*_*_*_*_*_*_*_*

Unit-3

Basic Processing Unit

Fundamental Concepts - Execution of a Complete Instruction - Multiple Bus Organization - Hardwired Control
– Micro programmed Control – Microinstructions- Microprogram Sequencing- Wide Branch Addressing

Introduction:

UNIT-IV: THE MEMORY SYSTEM

Basic Concepts, Semiconductor RAM, Types of Read-only Memory (ROM), Cache Memory, Performance Considerations, Virtual Memory, Secondary Storage.

4.1 Basic Concepts:

The maximum size of the memory that can be used in any computer is determined by the addressing scheme.

If MAR is k bits long and MDR is n bits long, then the memory may contain upto 2^k addressable locations and the n -bits of data are transferred between the memory and processor.

This transfer takes place over the processor bus.

The processor bus has,

- Address Line
- Data Line
- Control Line (R/W, MFC – Memory Function Completed)

The control line is used for co-ordinating data transfer.

The processor reads the data from the memory by loading the address of the required memory location into MAR and setting the R/W line to 1.

The memory responds by placing the data from the addressed location onto the data lines and confirms this action by asserting MFC signal.

Upon receipt of MFC signal, the processor loads the data onto the data lines into MDR register.

The processor writes the data into the memory location by loading the address of this location into MAR and loading the data into MDR sets the R/W line to 0.

- Measures for the speed of a memory:
 - memory access time.
 - It is the time that elapses between the initiation of an Operation and the completion of that operation.
 - memory cycle time.
 - It is the minimum time delay that required between the initiation of the two successive memory operations.

RAM (Random Access Memory):

In RAM, if any location that can be accessed for a Read/Write operation in fixed amount of time, it is independent of the location's address.

Cache Memory:

It is a small, fast memory that is inserted between the larger slower main memory and the processor.

It holds the currently active segments of a program and their data.

Virtual memory:

The address generated by the processor does not directly specify the physical locations in the memory.

The address generated by the processor is referred to as a virtual / logical address. The virtual address space is mapped onto the physical memory where data are actually stored.

The mapping function is implemented by a special memory control circuit is often called the memory management unit.

Only the active portion of the address space is mapped into locations in the physical memory.

The remaining virtual addresses are mapped onto the bulk storage devices used, which are usually magnetic disk.

As the active portion of the virtual address space changes during program execution, the memory management unit changes the mapping function and transfers the data between disk and memory.

Thus, during every memory cycle, an address processing mechanism determines whether the addressed in function is in the physical memory unit.

If it is, then the proper word is accessed and execution proceeds. If it is not, a page of words containing the desired word is transferred from disk to memory.

This page displaces some page in the memory that is currently inactive.

Semiconductor RAM

Semi-Conductor memories are available is a wide range of speeds. Their cycle time ranges from 100ns to 10ns.

INTERNAL ORGANIZATION OF MEMORY CHIPS:

Memory cells are usually organized in the form of array, in which each cell is capable of storing one bit of information.

Each row of cells constitutes a memory word and all cells of a row are connected to a common line called as word line.

The cells in each column are connected to Sense / Write circuit by two bit lines.

The Sense / Write circuits are connected to data input or output lines of the chip. During a write operation, the sense / write circuit receive input information and store it in the cells of the selected word.

The data input and data output of each senses / write ckt are connected to a single bidirectional data line that can be connected to a data bus of the cptr.

R / W · Specifies the required operation.

CS · Chip Select input selects a given chip in the multi-chip memory system

Static Memories:

Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memory.

- Two inverters are cross connected to form a latch.
- The latch is connected to two bit lines by transistors T1 and T2.
- These transistors act as switches that can be opened / closed under the control of the word line.
- When the word line is at ground level, the transistors are turned off and the latch retain its state.

Read Operation:

- In order to read the state of the SRAM cell, the word line is activated to close switches T1 and T2.
- If the cell is in state 1, the signal on bit line b is high and the signal on the bit line b' is low. Thus b and b' are complements of each other.
- Sense / write circuit at the end of the bit line monitors the state of b and b' and set the output accordingly.

Write Operation:

- The state of the cell is set by placing the appropriate value on bit line b and its complement on b' and then activating the word line. This forces the cell into the corresponding state.
- The required signal on the bit lines are generated by Sense / Write circuit.
- Transistor pairs (T3, T5) and (T4, T6) form the inverters in the latch.
- In state 1, the voltage at point X is high by having T5, T6 on and T4, T5 are OFF.
- Thus T1 and T2 returned ON (Closed), bit line b and b' will have high and low signals respectively.
- The CMOS requires 5V (in older version) or 3.3V (in new version) of power supply voltage.
- The continuous power is needed for the cell to retain its state

Merit :

- It has low power consumption because the current flows in the cell only when the cell is being activated accessed.
- Static RAM's can be accessed quickly. Its access time is few Nano seconds.

Demerit:

- SRAM's are said to be volatile memories because their contents are lost when the power is interrupted.

Fast Page Mode:

Transferring the bytes in sequential order is achieved by applying the consecutive sequence of column address under the control of successive CAS signals.

This scheme allows transferring a block of data at a faster rate. The block of transfer capability is called as Fast Page Mode.

Synchronous DRAM:

- Here the operations are directly synchronized with clock signal.
- The address and data connections are buffered by means of registers.
- The output of each sense amplifier is connected to a latch.
- A Read operation causes the contents of all cells in the selected row to be loaded in these latches.
- Data held in the latches that correspond to the selected columns are transferred into the data output register, thus becoming available on the data output pins.

Latency:

- It refers to the amount of time it takes to transfer a word of data to or from the memory.
- For a transfer of single word, the latency provides the complete indication of memory performance.
- For a block transfer, the latency denotes the time it takes to transfer the first word of data.

Bandwidth:

- It is defined as the number of bits or bytes that can be transferred in one second.
- Bandwidth mainly depends upon the speed of access to the stored data & on the number of bits that can be accessed in parallel.

Double Data Rate SDRAM (DDR-SDRAM):

- The standard SDRAM performs all actions on the rising edge of the clock signal.
- The double data rate SDRAM transfer data on both the edges (leading edge, trailing edge).
- The Bandwidth of DDR-SDRAM is doubled for long burst transfer.
- To make it possible to access the data at high rate, the cell array is organized into two banks.
- Each bank can be accessed separately.
- Consecutive words of a given block are stored in different banks.
- Such interleaving of words allows simultaneous access to two words that are transferred on successive edge of the clock.

Larger Memories:

Dynamic Memory System:

- The physical implementation is done in the form of Memory Modules.
- If a large memory is built by placing DRAM chips directly on the main system printed circuit board that contains the processor, often referred to as Motherboard; it will occupy large amount of space on the board.
- These packaging considerations have led to the development of larger memory units known as SIMMs & DIMMs.
 - SIMM-Single Inline memory Module
 - DIMM-Dual Inline memory Module
- SIMM & DIMM consists of several memory chips on a separate small board that plugs vertically into single socket on the motherboard.

4.3 Cache Memory

Ideally, computer memory should be fast, large and inexpensive. Unfortunately, it is impossible to meet all the three requirements simultaneously. Increased speed and size are achieved at increased cost. Very fast memory systems can be achieved if SRAM chips are used. These chips are expensive and for the cost reason it is impracticable to build a large main memory using SRAM chips. The alternative used to use DRAM chips for large main memories. The processor fetches the code and data from the main memory to execute the program. The DRAMs which form the main memory are slower devices. So it is necessary to insert wait states in memory read/write cycles. This reduces the speed of execution. The solution for this problem is in the memory system small section of SRAM is added along with the main memory, referred to as cache memory. The program which is to be executed is loaded in the main memory, but the part of the program and data accessed from the cache memory. The cache controller looks after this swapping between main memory and cache memory with the help of DMA controller, Such cache memory is called **secondary cache**. Recent processors have the built in cache memory called **primary cache**.

The size of the memory is still small compared to the demands of the large programs with the voluminous

data. A solution is provided by using secondary storage, mainly magnetic disks and magnetic tapes to implement large memory spaces, which is available at reasonable prices. To make efficient computer system it is not possible to rely on a single memory component, but to employ a memory hierarchy which uses all different types of memory units that gives efficient computer system. A typical memory hierarchy is illustrated below in the figure:

- Fastest access is to the data held in processor registers. Registers are at the top of the memory hierarchy.
- Relatively small amount of memory that can be implemented on the processor chip. This is processor cache.
- Two levels of cache. Level 1 (L1) cache is on the processor chip. Level 2 (L2) cache is in between main memory and processor.
- Next level is main memory, implemented as SIMMs. Much larger, but much slower than cache memory.
- Next level is magnetic disks. Huge amount of inexpensive storage.
- Speed of memory access is critical, the idea is to bring instructions and data that will be used in the near future as close to the processor as possible.

The effectiveness of cache mechanism is based on the property of "Locality of reference".

Locality of Reference:

Many instructions in the localized areas of the program are executed repeatedly during some time period and remainder of the program is accessed relatively infrequently.

It manifests itself in 2 ways. They are,

- **Temporal**(The recently executed instruction are likely to be executed again very soon.)
- **Spatial**(The instructions in close proximity to recently executed instruction are also likely to be executed soon.) If the active segment of the program is placed in cache memory, then the total execution time can be reduced significantly.

The term Block refers to the set of contiguous address locations of some size.

The cache line is used to refer to the cache block.

- The Cache memory stores a reasonable number of blocks at a given time but this number is small compared to the total number of blocks available in Main Memory.
- The correspondence between main memory block and the block in cache memory is specified by a mapping function.
- The Cache control hardware decide that which block should be removed to create space for the new block that contains the referenced word.
- The collection of rule for making this decision is called the replacement algorithm.
- The cache control circuit determines whether the requested word currently exists in the cache.
- If it exists, then Read/Write operation will take place on appropriate cache location. In this case Read/Write hit will occur.
- In a Read operation, the memory will not involve.
- The write operation is proceeding in 2 ways. They are,
 - Write-through protocol
 - Write-back protocol

Write-through protocol:

Here the cache location and the main memory locations are updated simultaneously.

Write-back protocol:

- This technique is to update only the cache location and to mark it as with associated flag bit called dirty/modified bit.

- The word in the main memory will be updated later, when the block containing this marked word is to be removed from the cache to make room for a new block.
- If the requested word currently not exists in the cache during read operation, then read miss will occur.
- To overcome the read miss Load – through / Early restart protocol is used.

Read Miss:

The block of words that contains the requested word is copied from the main memory into cache.

Load – through:

- After the entire block is loaded into cache, the particular word requested is forwarded to the processor.
- If the requested word not exists in the cache during write operation, then Write Miss will occur.
- If Write through protocol is used, the information is written directly into main memory.
- If Write back protocol is used then block containing the addressed word is first brought into the cache and then the desired word in the cache is over-written with the new information.

Cache Memories – Mapping Functions

First generation processors, those designed with vacuum tubes in 1950 or those designed with integrated circuits in 1965 or those designed as microprocessors in 1980

were generally about the same speed as main memory. On such processors, this naive

model was perfectly reasonable. By 1970, however, transistorized supercomputers were being built where the central processor was significantly faster than the main memory, and

by 1980, the difference had increased, although it took several decades for the performance difference to reach today's extreme.

Solution to this problem is to use what is called a **cache memory** between the

central processor and the main memory. Cache memory takes advantage of the fact that, with any of the memory technologies available for the past half century, we have had a choice between building large but slow memories or small but fast memories. This was known as far back as 1946, when Berks, Goldstone and Von Neumann proposed the use of a memory hierarchy, with a few fast registers in the central processor at the top of the hierarchy, a large main memory in the middle, and a library of archival data, stored off-line, at the very bottom.

A cache memory sits between the central processor and the main memory. During any particular memory cycle, the cache checks the memory address being issued by the processor. If this address matches the address of one of the few memory locations held in the cache, the cache handles the memory cycle very quickly; this is called a **cache hit**. If the address does not, then the memory cycle must be satisfied far more slowly by the main memory; this is called a **cache miss**.

The correspondence between the main memory and cache is specified by a Mapping function. When the cache is full and a memory word that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that constitutes the Replacement algorithm.

Mapping Functions

There are three main mapping techniques which decides the cache organization:

1. Direct-mapping technique
2. Associative mapping Technique
3. Set associative mapping technique

To discuss possible methods for specifying where memory blocks are placed in the cache, we use a specific small example, a cache consisting of 128 blocks of 16 word each, for a total of 2048(2k) word, and assuming that the main memory is addressable by a 16-bit address. The main memory has 64k word, which will be viewed as 4K blocks of 16 word each, the consecutive addresses refer to consecutive word.

Direct Mapping Technique

The cache systems are divided into three categories, to implement cache system. As shown in figure, the lower order 4-bits from 16 words in a block constitute a **word field**. The second field is known as **block field** used to distinguish a block from other blocks. Its length is 7-bits, when a new block enters the cache; the 7-bit cache block field determines the cache position in which this block must be stored. The third field is a **Tag field**, used to store higher order 5-bits of the memory address of the block, and to identify which of the 32blocks are mapped into the cache.

It is the simplest mapping technique, in which each block from the main memory has only one possible location in the cache organization. For example, the block I of the main memory maps on to block $i \text{ module } 128$ of the cache. Therefore, whenever one of the main memory blocks 0, 128, 256, Is loaded in the cache, it is stored in the block 0. Block 1, 129, 257,..... are stored in block 1 of the cache and so on.

Associative Mapping Technique

The figure shows the associative mapping, where in which main memory block can be placed into any cache block position, in this case, 12 tag bits are required to identify a memory block when it is resident in the cache. The tag bits of an address received from the processor are compared to the tag bits of each block of the cache, to see if the desired

block is present. This is called associative-mapping technique. It gives the complete freedom in choosing the cache location in which to place the memory block.

Set-Associative Mapping

It is a combination of the direct and associative-mapping techniques can be used. Blocks of the cache are grouped into sets and the mapping allows a block of main memory to reside in any block of the specific set. In this case memory blocks 0, 64,128.....4032 mapped into cache set 0, and they can occupy either of the two block positions within this set. The cache might contain the desired block. The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present this two associative search is simple to implement.

Replacement Algorithms

In a direct-mapped cache, the position of each block is fixed, hence no replacement strategy exists. In associative and set-associative caches, when a new block is to be brought into the cache and all the positions that it may occupy are full, the cache controller must decide which of the old blocks to overwrite. This is an important issue because the decision can be a factor in system performance.

The objective is to keep blocks in the cache that are likely to be referenced in the near future. It's not easy to determine which blocks are about to be referenced. The property of locality of reference gives a clue to a reasonable strategy. When a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. This block is called the least recently used (LRU) block, and the technique is called the **LRU Replacement algorithm**.

The LRU algorithm has been used extensively for many access patterns, but it can lead to poor performance in some cases. For example, it produces disappointing results when accesses are made to sequential elements of an array that is slightly too large to fit into the cache. Performance of the LRU algorithm can be improved by introducing a small amount of randomness in deciding which block to replace.

Example of mapping techniques

Direct mapped Cache:

Associate mapped cache:

Set Associative mapped cache:

4.5 Performance Considerations:

- Two Key factors in the commercial success are the performance & cost ie the best possible performance at low cost.
- A common measure of success is called the Price/ Performance ratio. Performance depends on how fast the machine instruction are brought to the processor and how fast they are executed.
- To achieve parallelism(ie. Both the slow and fast units are accessed in the same manner),interleaving is used.

Interleaving:

- If the main memory system is divided into a number of memory modules. Each module has its own address buffer register (ABR) and data buffer register (DBR).
- Memory access operations may proceed in more than one module at the same time. Thus the aggregate rate of transmission of words to and from the main memory system can be increased.
- Two methods of address layout are indicated they are
 - Consecutive words in a module
 - Consecutive words in a consecutive module
- **Consecutive words in a module**

- Consecutive words are placed in a module.
- High-order k bits of a memory address determine the module.
- Low-order m bits of a memory address determine the word within a module.
- When a block of words is transferred from main memory to cache, only one module is busy at a time.

- **Consecutive words in a consecutive module**

•

- Consecutive words are located in consecutive modules.
- Consecutive addresses can be located in consecutive modules.
- While transferring a block of data, several memory modules can be kept busy at the same time.
- This is called **interleaving**
- When requests for memory access involve consecutive addresses, the access will be to different modules.
- Since parallel access to these modules is possible, the average rate of fetching words from the Main Memory can be increased.

Example:

block from the interleaved memory is

$$1 + 8 + 4 + 4 = 17 \text{ cycles}$$

Thus, interleaving reduces the block transfer time by more than a factor of 2.

Hit Rate and Miss Penalty

An excellent indicator of the effectiveness of a particular implementation of the memory hierarchy is the success rate in accessing information at various level of the hierarchy. A successful access to data in a cache is called a hit.

The number of hits stated as fraction of all attempted access is called the hit rate, and the miss rate is the number of misses stated as a fraction of attempted accesses.

- Hit rate can be improved by increasing block size, while keeping cache size constant.
- Block sizes that are neither very small nor very large give best results.
- Miss penalty can be reduced if load-through approach is used when loading new blocks into cache.

Example:

$$\frac{\text{Time without cache}}{\text{Time with cache}} = \frac{130 \times 10}{100(0.95 \times 1 + 0.05 \times 17) + 30(0.9 \times 1 + 0.1 \times 17)} = 5.04$$

This result suggests that the computer with the cache performs five times better.

It is also interesting to consider how effective this cache is compared to an ideal cache that has a hit rate of 100 percent (in which case, all memory references take one cycle). Our rough estimate of relative performance for these caches is

$$\frac{100(0.95 \times 1 + 0.05 \times 17) + 30(0.9 \times 1 + 0.1 \times 17)}{130} = 1.98$$

Example 2:

Repeating the calculation in Example 5.2 gives:

$$\frac{\text{Time without cache}}{\text{Time with cache}} = \frac{130 \times 36}{100(0.95 \times 1 + 0.05 \times 60) + 30(0.9 \times 1 + 0.1 \times 60)} = 7.77$$

Thus, accounting for the differences between processor and system bus clock speeds shows that the cache has an even greater positive effect on the performance.

Caches on processor chip:

In high-performance processors two levels of caches are normally used. The L1 cache(s) is on the processor chip. The L2 cache, which is much larger, may be implemented externally using SRAM chips. But, a somewhat smaller L2 cache may also be implemented on the processor chip,

If both L1 and L2 caches are used, the L1 cache should be designed to allow very fast access by the processor because its access time will have a large effect on the clock rate of the processor. A cache cannot be accessed at the same speed as a register file because the cache is much bigger and, hence, more complex. A practical way to speed up access to the cache is to access more than one word simultaneously and then let the processor use them one at a time. This technique is used in many commercial processors.

The L2 cache can be slower, but it should be much larger to ensure a high hit rate. Its speed is less critical because it only affects the miss penalty of the L1 cache. A workstation computer may include an L1 cache with the capacity of tens of kilobytes and an L2 cache of several megabytes.

Including an L2 cache further reduces the impact of the main memory speed on the performance of a computer. The average access time experienced by the processor in a system with two levels of caches is

$$t_{ave} = h_1 C_1 + (1 - h_1)h_2 C_2 + (1 - h_1)(1 - h_2)M$$

where

h_1 is the hit rate in the L1 cache.

h_2 is the hit rate in the L2 cache.

C_1 is the time to access information in the L1 cache.

C_2 is the time to access information in the L2 cache.

M is the time to access information in the main memory.

The number of misses in the L2 cache, given by the term $(1 - h_1)(1 - h_2)$, should be low. If both h_1 and h_2 are in the 90 percent range, then the number of misses will be less than 1 percent of the processor's memory accesses. Thus, the miss penalty M will be less critical from a performance point of view.

Other enhancements:

Write buffer

- Write-through:
 - Each write operation involves writing to the main memory.
 - If the processor has to wait for the write operation to be complete, it slows down the processor.
 - Processor does not depend on the results of the write operation.
 - Write buffer can be included for temporary storage of write requests.
 - Processor places each write request into the buffer and continues execution.
 - If a subsequent Read request references data which is still in the write buffer, then this data is referenced in the write buffer.
- Write-back:
 - Block is written back to the main memory when it is replaced.
 - If the processor waits for this write to complete, before reading the new block, it is slowed down.
 - Fast write buffer can hold the block to be written, and the new block can be read first.

Prefetching

- New data are brought into the processor when they are first needed.
- Processor has to wait before the data transfer is complete.
- Prefetch the data into the cache before they are actually needed, or a before a Read miss occurs.
- Prefetching can be accomplished through software by including a special instruction in the machine language of the processor.
 - Inclusion of prefetch instructions increases the length of the programs.
- Prefetching can also be accomplished using hardware:
 - Circuitry that attempts to discover patterns in memory references and then prefetches according to this pattern.

Lockup-Free Cache

- Prefetching scheme does not work if it stops other accesses to the cache until the prefetch is completed.
- A cache of this type is said to be "locked" while it services a miss.
- Cache structure which supports multiple outstanding misses is called a lockup free cache.
- Since only one miss can be serviced at a time, a lockup free cache must include circuits that keep track of all the outstanding misses.
- Special registers may hold the necessary information about these misses.

4.6 VIRTUAL MEMORY:

- Techniques that automatically move program and data blocks into the physical main memory when they are required for execution is called the **Virtual Memory**.
- The binary address that the processor issues either for instruction or data are called the virtual / Logical address.
- The virtual address is translated into physical address by a combination of hardware and software components. This kind of address translation is done by MMU (Memory Management Unit).

- When the desired data are in the main memory, these data are fetched /accessed immediately.
- If the data are not in the main memory, the MMU causes the Operating system to bring the data into memory from the disk.
- Transfer of data between disk and main memory is performed using DMA scheme.

Fig: Virtual Memory Organization

- Memory management unit (MMU) translates virtual addresses into physical addresses.
- If the desired data or instructions are in the main memory they are fetched as described previously.
- If the desired data or instructions are not in the main memory, they must be transferred from secondary storage to the main memory.
- MMU causes the operating system to bring the data from the secondary storage into the main memory.

Address Translation:

In address translation, all programs and data are composed of fixed length units called

Pages.

The Page consists of a block of words that occupy contiguous locations in the main memory.

The pages are commonly range from 2K to 16K bytes in length. The cache bridge speed up the gap between main memory and secondary storage and it is implemented in software techniques.

Each virtual address generated by the processor contains virtual Page number (Low order bit) and offset(High order bit) Virtual Page number+ Offset. Specifies the location of a particular byte (or word) within a page.

Page Table:

It contains the information about the main memory address where the page is stored & the current status of the page.

Page Frame:

An area in the main memory that holds one page is called the page frame.

Page Table Base Register:

- It contains the starting address of the page table.
- Virtual Page Number+Page Table Base register. Gives the address of the corresponding entry in the page table.ie)it gives the starting address of the page if that page currently resides in memory.

Control Bits in Page Table:

- The Control bit specifies the status of the page while it is in main memory. Function:
- The control bit indicates the validity of the page ie) it checks whether the page is actually loaded in the main memory.
- It also indicates that whether the page has been modified during its residency in the memory; this information is needed to determine whether the page should be written back to the disk before it is removed from the main memory to make room for another page.

Fig: Virtual Memory Address Translation

- The Page table information is used by MMU for every read & write access.
- The Page table is placed in the main memory but a copy of the small portion of the page table is located within MMU.
- This small portion or small cache is called Translation Look Aside Buffer (TLB).
- This portion consists of the page table entries that corresponds to the most recently accessed pages and also contains the virtual address of the entry.

- When the operating system changes the contents of page table , the control bit in TLB will invalidate the corresponding entry in the TLB. Given a virtual address, the MMU looks in TLB for the referenced page.
- If the page table entry for this page is found in TLB, the physical address is obtained immediately. If there is a miss in TLB, then the required entry is obtained from the page table in the main memory & TLB is updated.
- When a program generates an access request to a page that is not in the main memory, then Page Fault will occur.

- The whole page must be brought from disk into memory before an access can proceed. When it detects a page fault, the MMU asks the operating system to generate an interrupt.
- The operating System suspend the execution of the task that caused the page fault and begin execution of another task whose pages are in main memory because the long delay occurs while page transfer takes place.
- When the task resumes, either the interrupted instruction must continue from the point of interruption or the instruction must be restarted.
- If a new page is brought from the disk when the main memory is full, it must replace one of the resident pages. In that case, it uses LRU algorithm which removes the least referenced Page.
- A modified page has to be written back to the disk before it is removed from the main memory. In that case, write – through protocol is used.

MEMORY MANAGEMENT REQUIREMENTS:

Management routines are part of the Operating system. Assembling the OS routine into virtual address space is called "System Space". The virtual space in which the user application programs reside is called the "User Space". Each user space has a separate page table. The MMU uses the page table to determine the address of the table to be used in the translation process. Hence by changing the contents of this register, the OS can switch from one space to another. The process has two stages. They are,

- User State
- Supervisor state.

User State: In this state, the processor executes the user program.

Supervisor State: When the processor executes the operating system routines, the processor will be in supervisor state. Privileged Instruction:

In user state, the machine instructions cannot be executed. Hence a user program is prevented from accessing the page table of other user spaces or system spaces.

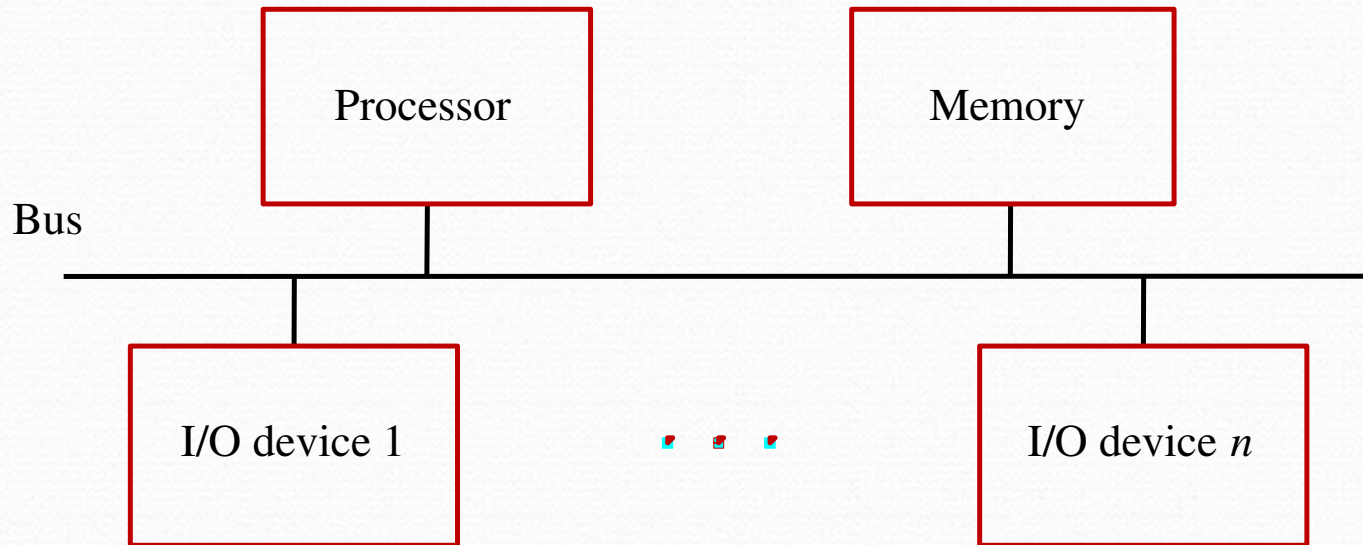
The control bits in each entry can be set to control the access privileges granted to each program. ie) One program may be allowed to read/write a given page, while the other programs may be given only read access.

INPUT/OUTPUT ORGANIZATION

UNIT 5

Accessing I/O Devices

Accessing I/O devices

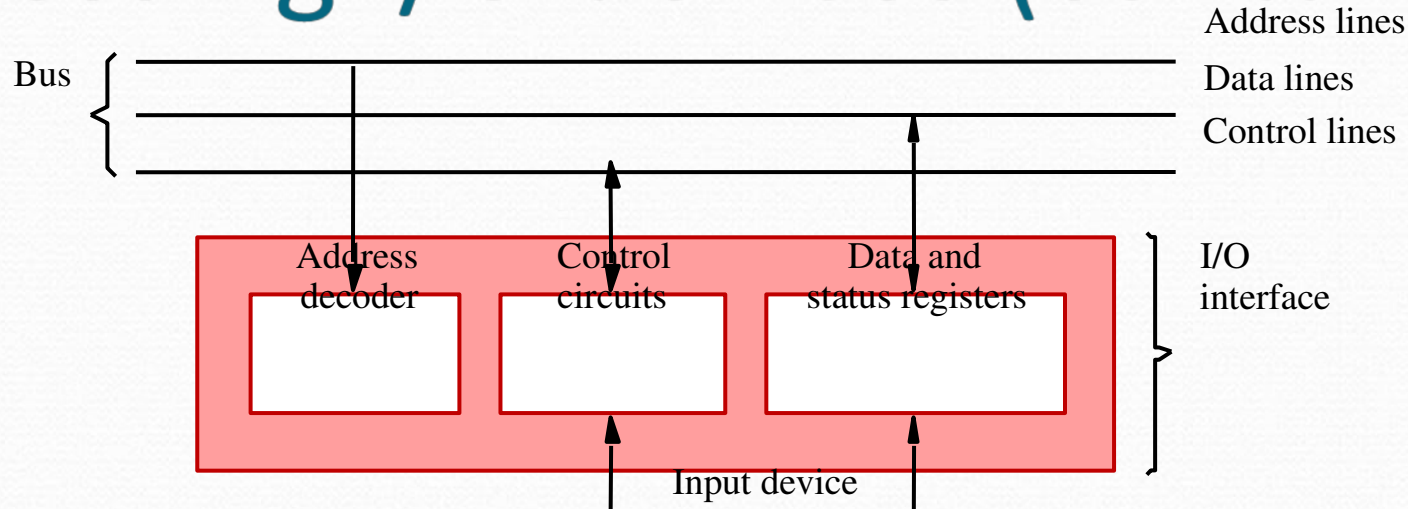


- *Multiple I/O devices may be connected to the processor and the memory via a bus.*
- *Bus consists of three sets of lines to carry address, data and control signals.*
- *Each I/O device is assigned an unique address.*
- *To access an I/O device, the processor places the address on the address lines.*
- *The device recognizes the address, and responds to the control signals.*

Accessing I/O devices (contd..)

- I/O devices and the memory may share the same address space:
 - Memory-mapped I/O.
 - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
 - Simpler software.
- I/O devices and the memory may have different address spaces:
 - Special instructions to transfer data to and from I/O devices.
 - I/O devices may have to deal with fewer address lines.
 - I/O address lines need not be physically separate from memory address lines.
 - In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

Accessing I/O devices (contd..)



- *I/O device is connected to the bus using an I/O interface circuit which has:
 - Address decoder, control circuit, and data and status registers.*
- *Address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.*
- *Data register holds the data being transferred to or from the processor.*
- *Status register holds information necessary for the operation of the I/O device.*
- *Data and status registers are connected to the data lines, and have unique addresses.*
- *I/O interface circuit coordinates I/O transfers.*

Accessing I/O devices (contd..)

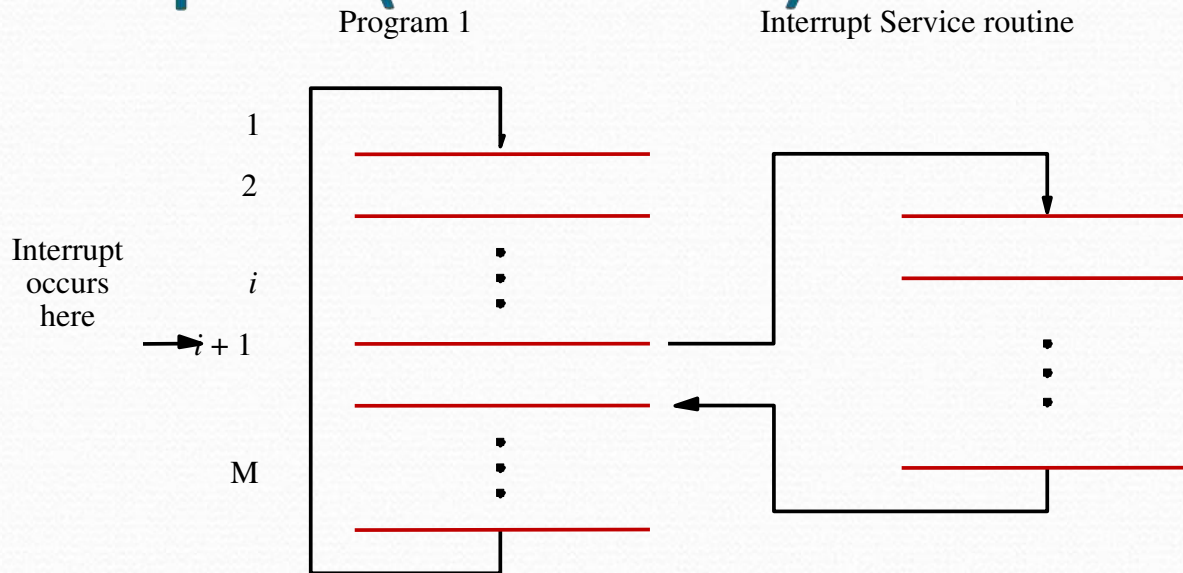
- Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.
- **Program-controlled I/O:**
 - Processor repeatedly monitors a status flag to achieve the necessary synchronization.
 - Processor polls the I/O device.
- **Two other mechanisms used for synchronizing data transfers between the processor and memory:**
 - **Interrupts.**
 - **Direct Memory Access.**

Interrupts

Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
 - Do so by sending a hardware signal called an interrupt to the processor.
 - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.
- Processor can perform other useful tasks while it is waiting for the device to be ready.

Interrupts (contd..)



- Processor is executing the instruction located at address i when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine.
- When an interrupt occurs, control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC ($i+1$), must be saved in a known location.
- This will enable the return-from-interrupt instruction to resume execution at $i+1$.
- Return address, or the contents of the PC are usually stored on the processor stack.

Interrupts (contd..)

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
 - A subroutine performs a task that is required by the calling program.
 - Interrupt-service routine may not have anything in common with the program it interrupts.
 - Interrupt-service routine and the program that it interrupts may belong to different users.
 - As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
 - This will enable the interrupted program to resume execution upon return from interrupt service routine.

Interrupts (contd..)

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
 - Increases the total execution time.
 - Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called interrupt latency.
- In order to reduce the interrupt latency, **most processors save only the minimal amount of information:**
 - This minimal amount of information includes Program Counter and processor status registers.

- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

Interrupts (contd..)

- When a processor receives an interrupt-request, it must branch to the interrupt service routine.
- It must also inform the device that it has recognized the interrupt request.
- This can be accomplished in two ways:
 - Some processors have an explicit interrupt-acknowledge control signal for this purpose.
 - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
 - Sometimes such alterations may be undesirable, and must not be allowed.
 - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
 - First instruction of an interrupt service routine can be *Interrupt-disable*.
 - Last instruction of an interrupt service routine can be *Interrupt-enable*.

Interrupts (contd..)

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
 - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

Interrupts (contd..)

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, **how does it know which device is requesting an interrupt?**
- **This information is available in the status register of the device requesting an interrupt:**
 - The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.
- **Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.**

- Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

Interrupts (contd..)

- The device requesting an interrupt may identify itself directly to the processor.
 - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
 - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
 - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine.

Interrupts (contd..)

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
 - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

Interrupts (contd..)

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, **how does it know which device is requesting an interrupt?**
- This information is available in the status register of the device requesting an interrupt:
 - The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.
- **Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.**

- Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

Interrupts (contd..)

- The device requesting an interrupt may identify itself directly to the processor.
 - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
 - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
 - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine.

Interrupts (contd..)

- Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.
- In general, same arrangement is used when multiple devices can send interrupt requests to the processor.
 - During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
 - Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.
- However, for certain devices this delay may not be acceptable.
 - Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

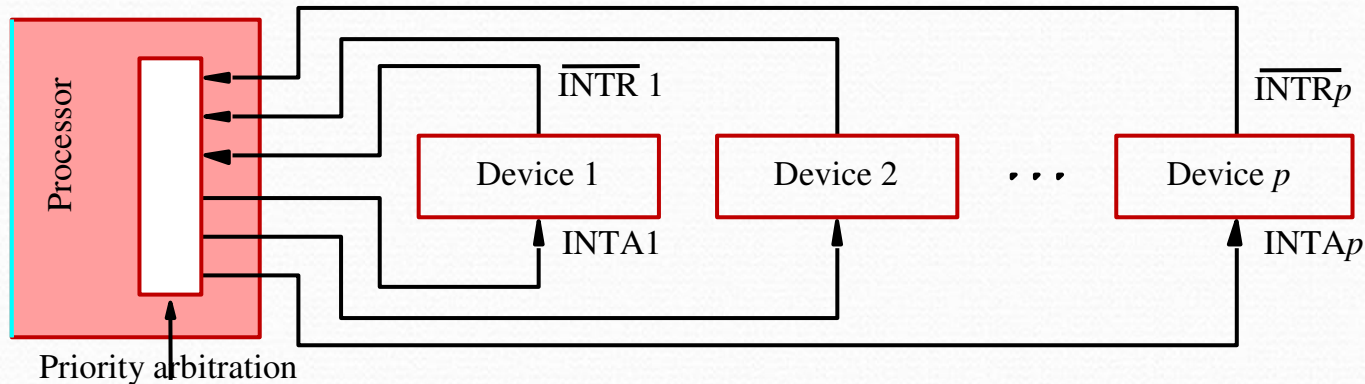
Interrupts (contd..)

- I/O devices are organized in a priority structure:
 - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.
- A priority level is assigned to a processor that can be changed under program control.
 - Priority level of a processor is the priority of the program that is currently being executed.
 - When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.
 - If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request.

Interrupts (contd..)

- Processor's priority is encoded in a few bits of the processor status register.
 - Priority can be changed by instructions that write into the processor status register.
 - Usually, these are privileged instructions, or instructions that can be executed only in the supervisor mode.
 - Privileged instructions cannot be executed in the user mode.
 - Prevents a user program from accidentally or intentionally changing the priority of the processor.
- If there is an attempt to execute a privileged instruction in the user mode, it causes a special type of interrupt called as privilege exception.

Interrupts (contd..)



- Each device has a separate interrupt-request and interrupt-acknowledge line.
- Each interrupt-request line is assigned a different priority level.
- Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
- If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.

Interrupts (contd..)

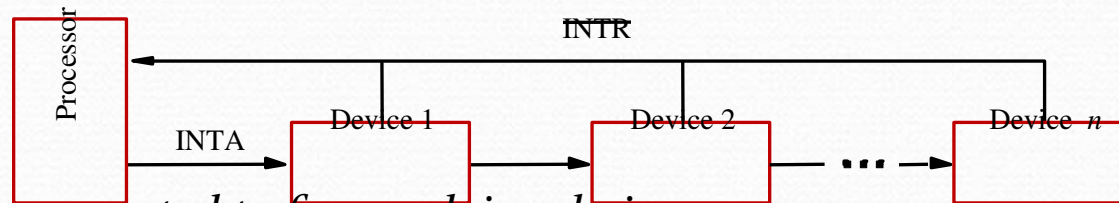
- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
 - Each device has its own interrupt request and interrupt acknowledge line.
 - A different priority level is assigned to the interrupt request line of each device.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

Interrupts (contd..)

Polling scheme:

- If the processor uses a polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt.
- In this case the priority is determined by the order in which the devices are polled.
- The first device with status bit set to 1 is the device whose interrupt request is accepted.

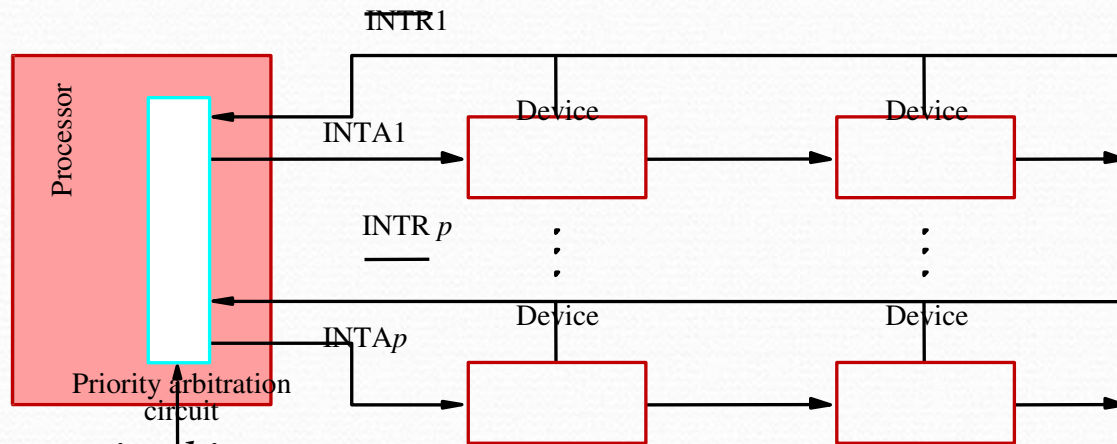
Daisy chain scheme:



- Devices are connected to form a daisy chain.
- Devices share the interrupt-request line, and interrupt-acknowledge line is connected to form a daisy chain.
- When devices raise an interrupt request, the interrupt-request line is activated.
- The processor in response activates interrupt-acknowledge.
- Received by device 1, if device 1 does not need service, it passes the signal to device 2.
- Device that is electrically closest to the processor has the highest priority.

Interrupts (contd..)

- When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt-acknowledge line.
- When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.
- A combination of priority structure and daisy chain scheme can also be used.



- Devices are organized into groups.
- Each group is assigned a different priority level.
- All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain.

Interrupts (contd..)

- Only those devices that are being used in a program should be allowed to generate interrupt requests.
- To control which devices are allowed to generate interrupt requests, the interface circuit of each I/O device has an interrupt-enable bit.
 - If the interrupt-enable bit in the device interface is set to 1, then the device is allowed to generate an interrupt-request.
- Interrupt-enable bit in the device's interface circuit determines whether the device is allowed to generate an interrupt request.
- Interrupt-enable bit in the processor status register or the priority structure of the interrupts determines whether a given interrupt will be accepted.

Exceptions

- Interrupts caused by interrupt-requests sent by I/O devices.
- Interrupts could be used in many other situations where the execution of one program needs to be suspended and execution of another program needs to be started.
- In general, **the term exception is used to refer to any event that causes an interruption.**
 - Interrupt-requests from I/O devices is one type of an exception.
- Other types of exceptions are:
 - Recovery from errors
 - Debugging
 - Privilege exception

Exceptions (contd..)

- Many sources of errors in a processor.
For example:
 - Error in the data stored.
 - Error during the execution of an instruction.
- When such errors are detected, exception processing is initiated.
 - Processor takes the same steps as in the case of I/O interrupt-request.
 - It suspends the execution of the current program, and starts executing an exception-service routine.
- Difference between handling I/O interrupt-request and handling exceptions due to errors:
 - In case of I/O interrupt-request, the processor usually completes the execution of an instruction in progress before branching to the interrupt-service routine.

- In case of exception processing however, the execution of an instruction in progress usually cannot be completed.

Exceptions (contd..)

- Debugger uses exceptions to provide important features:
 - Trace,
 - Breakpoints.
- Trace mode:
 - Exception occurs after the execution of every instruction.
 - Debugging program is used as the exception-service routine.
- Breakpoints:
 - Exception occurs only at specific points selected by the user.
 - Debugging program is used as the exception-service routine.

Exceptions (contd..)

- Certain instructions can be executed only when the processor is in the supervisor mode. These are called privileged instructions.
- If an attempt is made to execute a privileged instruction in the user mode, a privilege exception occurs.
- Privilege exception causes:
 - Processor to switch to the supervisor mode,
 - Execution of an appropriate exception-servicing routine.

Direct Memory Access

Direct Memory Access (contd..)

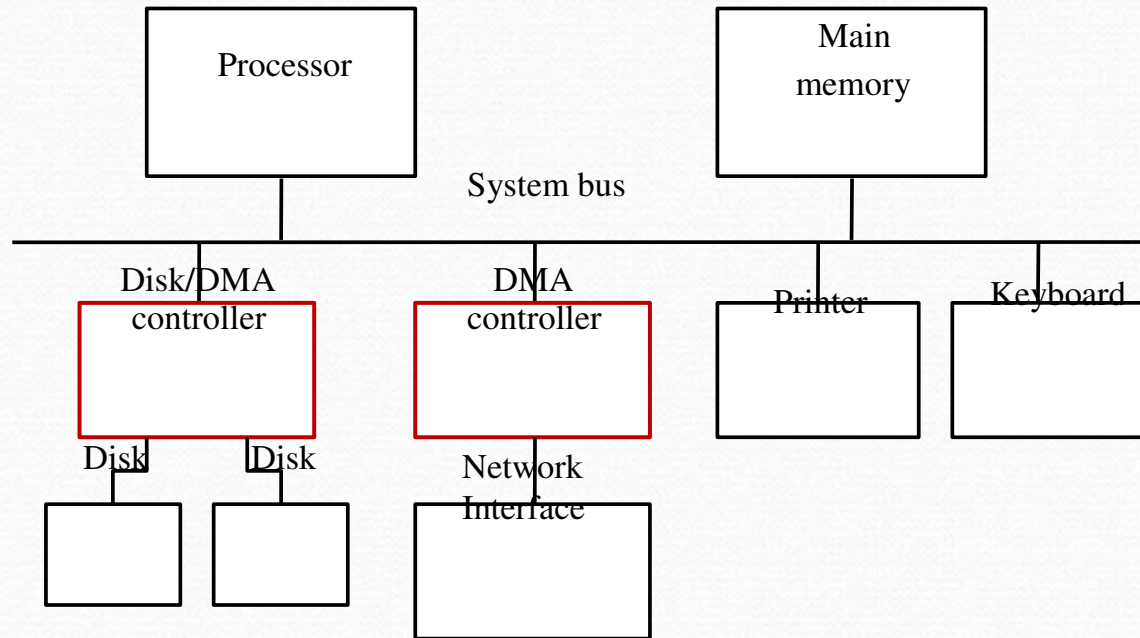
- **Direct Memory Access (DMA):**
 - A special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor.
- Control unit which performs these transfers is a part of the I/O device's interface circuit. This control unit is called as a DMA controller.
- DMA controller performs functions that would be normally carried out by the processor:
 - For each word, it provides the memory address and all the control signals.
 - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

Direct Memory Access (contd..)

- DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.
 - However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.
- To initiate the DMA transfer, the processor informs the
DMA controller of:
 - Starting address,
 - Number of words in the block.
 - Direction of transfer (I/O device to the memory, or memory to the I/O device).

- Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

Direct Memory Access



- *DMA controller connects a high-speed network to the computer bus.*
- *Disk controller, which controls two disks also has DMA capability. It provides two DMA channels.*
- *It can perform two independent DMA operations, as if each disk has its own DMA controller. The registers to store the memory address, word count and status and control information are duplicated.*

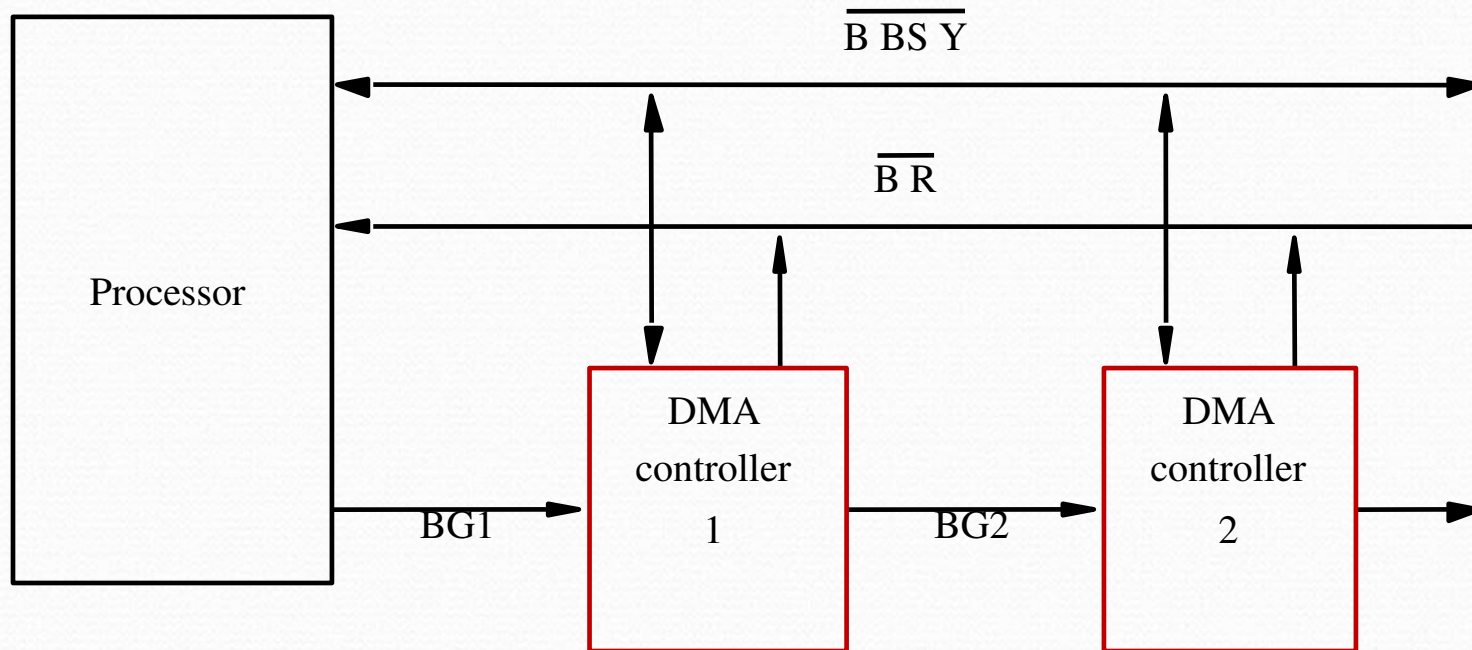
Direct Memory Access (contd..)

- Processor and DMA controllers have to use the bus in an interwoven fashion to access the memory.
 - DMA devices are given higher priority than the processor to access the bus.
 - Among different DMA devices, high priority is given to high-speed peripherals such as a disk or a graphics display device.
- Processor originates most memory access cycles on the bus.
 - DMA controller can be said to “steal” memory access cycles from the bus. This interweaving technique is called as “cycle stealing”.
- An alternate approach is to provide a DMA controller an exclusive capability to initiate transfers on the bus, and hence exclusive access to the main memory. This is known as the block or burst mode.

Bus arbitration

- Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.
- The device that is allowed to initiate transfers on the bus at any given time is called the bus master.
- When the current bus master relinquishes its status as the bus master, another device can acquire this status.
 - The process by which the next device to become the bus master is selected and bus mastership is transferred to it is called bus arbitration.
- **Centralized arbitration:**
 - A single bus arbiter performs the arbitration.
- **Distributed arbitration:**
 - All devices participate in the selection of the next bus master.

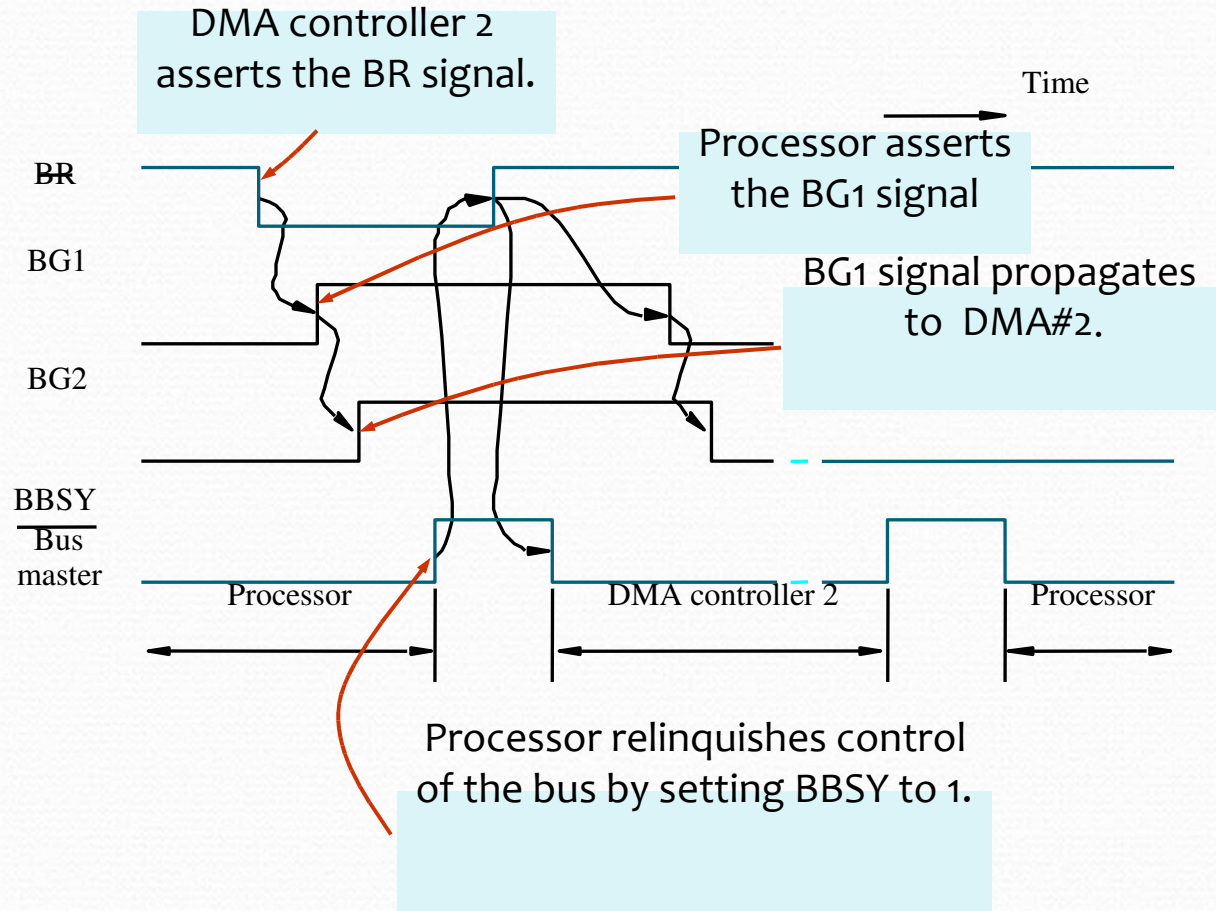
Centralized Bus Arbitration



Centralized Bus Arbitration(cont.,)

- *Bus arbiter may be the processor or a separate unit connected to the bus.*
- *Normally, the processor is the bus master, unless it grants bus membership to one of the DMA controllers.*
- *DMA controller requests the control of the bus by asserting the Bus Request (BR) line.*
- *In response, the processor activates the Bus-Grant₁ (BG₁) line, indicating that the controller may use the bus when it is free.*
- *BG₁ signal is connected to all DMA controllers in a daisy chain fashion.*
- *BBSY signal is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted BR can acquire control of the bus.*

Centralized arbitration (contd..)

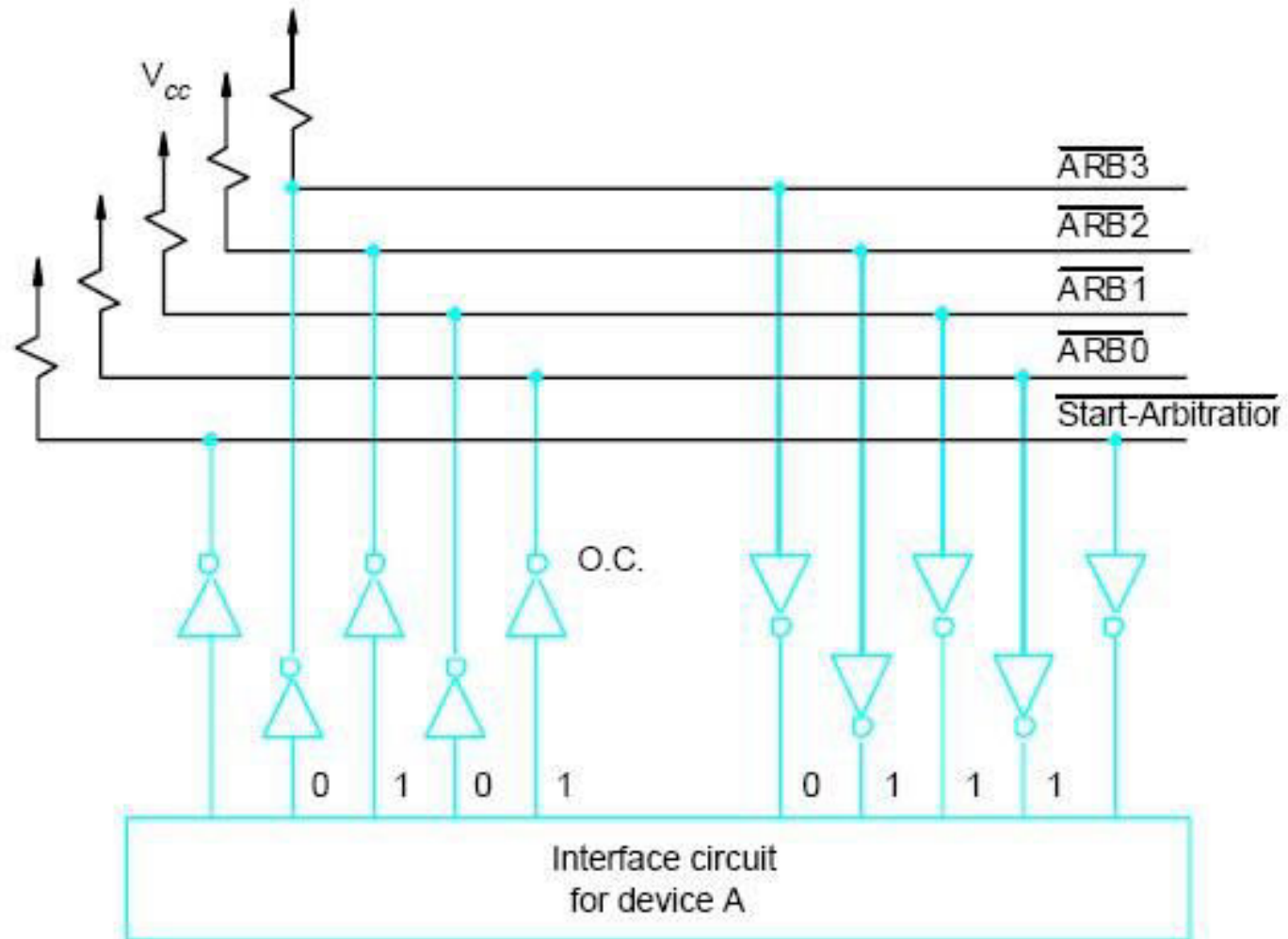


Distributed arbitration

- All devices waiting to use the bus share the responsibility of carrying out the arbitration process.
 - Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability.
- Each device is assigned a 4-bit ID number.
- All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start- Arbitration signal.
- **To request the bus a device:**
 - Asserts the Start-Arbitration signal.
 - Places its 4-bit ID number on the arbitration lines.

- The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.

Distributed arbitration



Distributed arbitration(Contd.,)

- Arbitration process:
 - *Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.*
 - *If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.*
 - *The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.*

Distributed arbitration (contd..)

- *Device A has the ID 5 and wants to request the bus:*
 - *Transmits the pattern 0101 on the arbitration lines.*
- *Device B has the ID 6 and wants to request the bus:*
 - *Transmits the pattern 0110 on the arbitration lines.*
- *Pattern that appears on the arbitration lines is the logical OR of the patterns:*
 - *Pattern 0111 appears on the arbitration lines.*

Arbitration process:

- *Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.*
- *If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.*
- *Device A compares its ID 5 with a pattern 0101 to pattern 0111.*
- *It detects a difference at bit position 0, as a result, it transmits a pattern 0100 on the arbitration lines.*
- *The pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110.*
- *This pattern is the same as the device ID of B, and hence B has won the arbitration.*

Buses

Buses

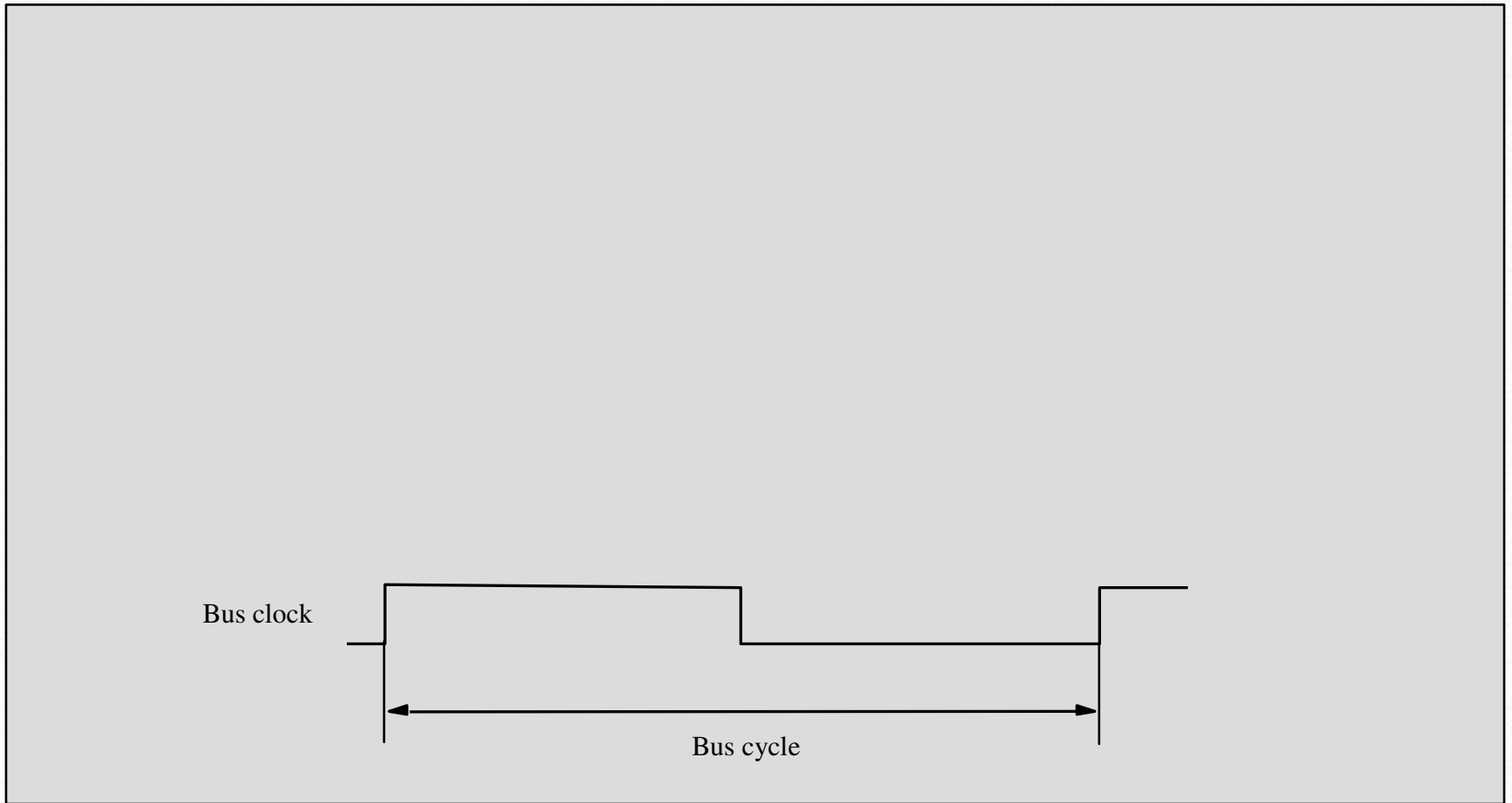
- Processor, main memory, and I/O devices are interconnected by means of a bus.
- Bus provides a communication path for the transfer of data.
 - Bus also includes lines to support interrupts and arbitration.
- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus, as to when to place information on the bus, when to assert control signals, etc.

Buses (contd..)

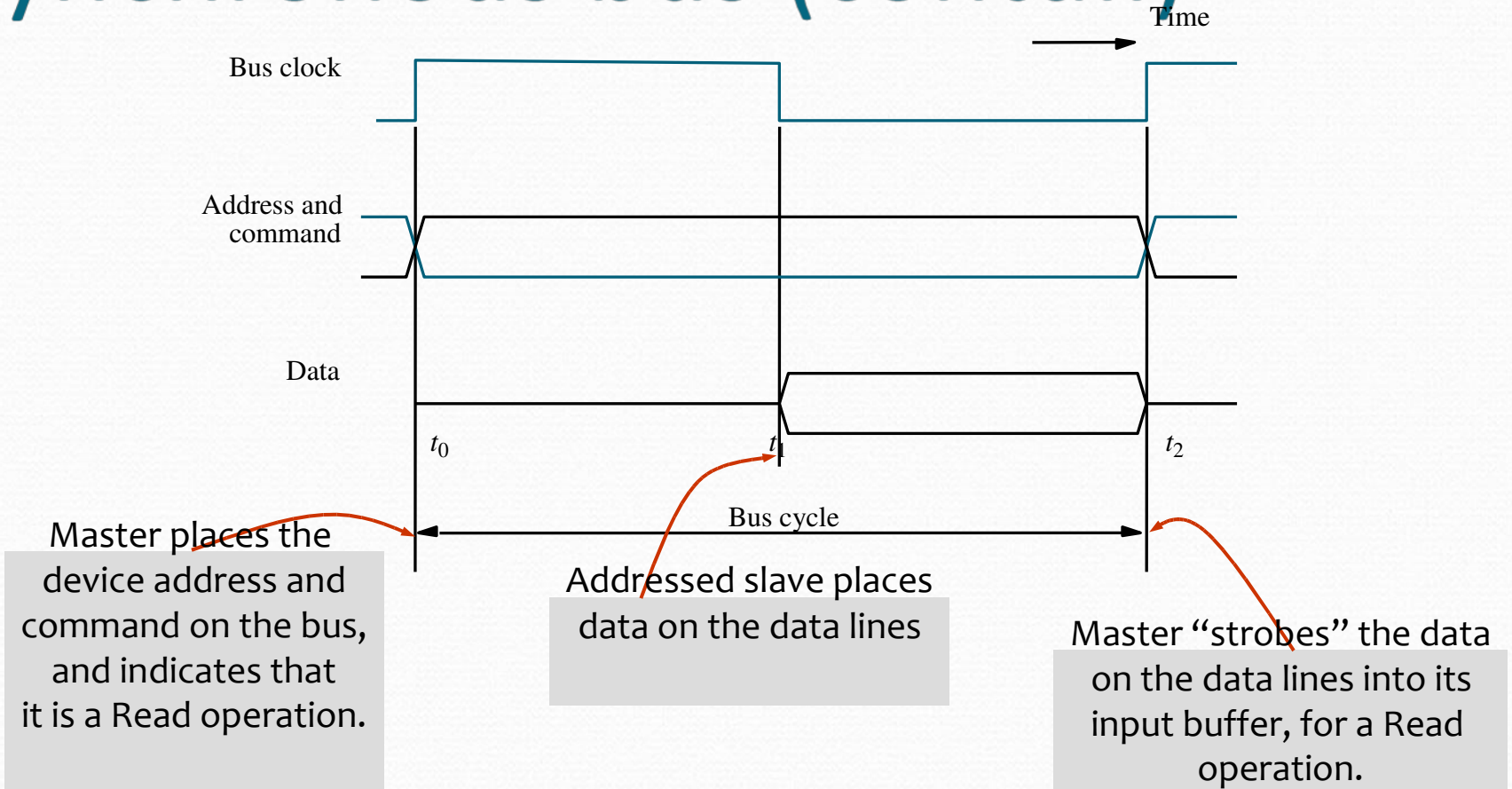
- Bus lines may be grouped into three types:
 - Data
 - Address
 - Control
- Control signals specify:
 - Whether it is a read or a write operation.
 - Required size of the data, when several operand sizes (byte, word, long word) are possible.
 - Timing information to indicate when the processor and I/O devices may place data or receive data from the bus.
- Schemes for timing of data transfers over a bus can be classified into:
 - Synchronous,

- Asynchronous.

Synchronous bus



Synchronous bus (contd..)



- *In case of a Write operation, the master places the data on the bus along with the address and commands at time t_0 .*
- *The slave strobes the data into its input buffer at time t_2 .*

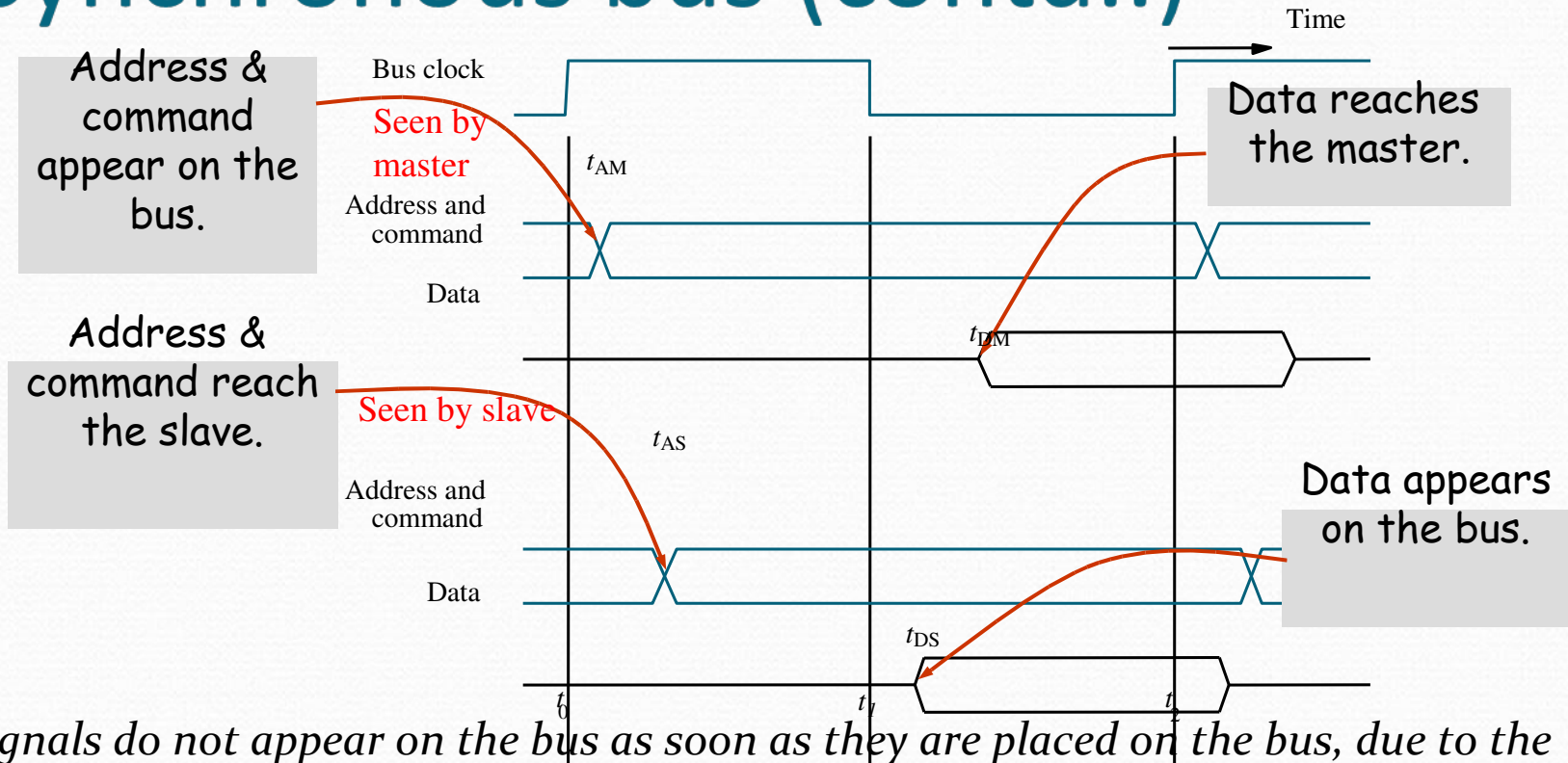
Synchronous bus (contd..)

- Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices:
 - This time depends on the physical and electrical characteristics of the bus.
- Also, all the devices have to be given enough time to decode the address and control signals, so that the addressed slave can place data on the bus.
- **Width of the pulse $t_1 - t_0$ depends on:**
 - Maximum propagation delay between two devices connected to the bus.
 - Time taken by all the devices to decode the address and control signals, so that the addressed slave can respond at time t_1 .

Synchronous bus (contd..)

- At the end of the clock cycle, at time t_2 , the master strobes the data on the data lines into its input buffer if it's a Read operation.
 - “Strobe” means to capture the values of the data and store them into a buffer.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.
- Width of the pulse $t_2 - t_1$ should be longer than:
 - Maximum propagation time of the bus plus
 - Set up time of the input buffer register of the master.

Synchronous bus (contd..)



- Signals do not appear on the bus as soon as they are placed on the bus, due to the propagation delay in the interface circuits.
- Signals reach the devices after a propagation delay which depends on the characteristics of the bus.
- Data must remain on the bus for some time after t_2 equal to the hold time of the buffer.

Synchronous bus (contd..)

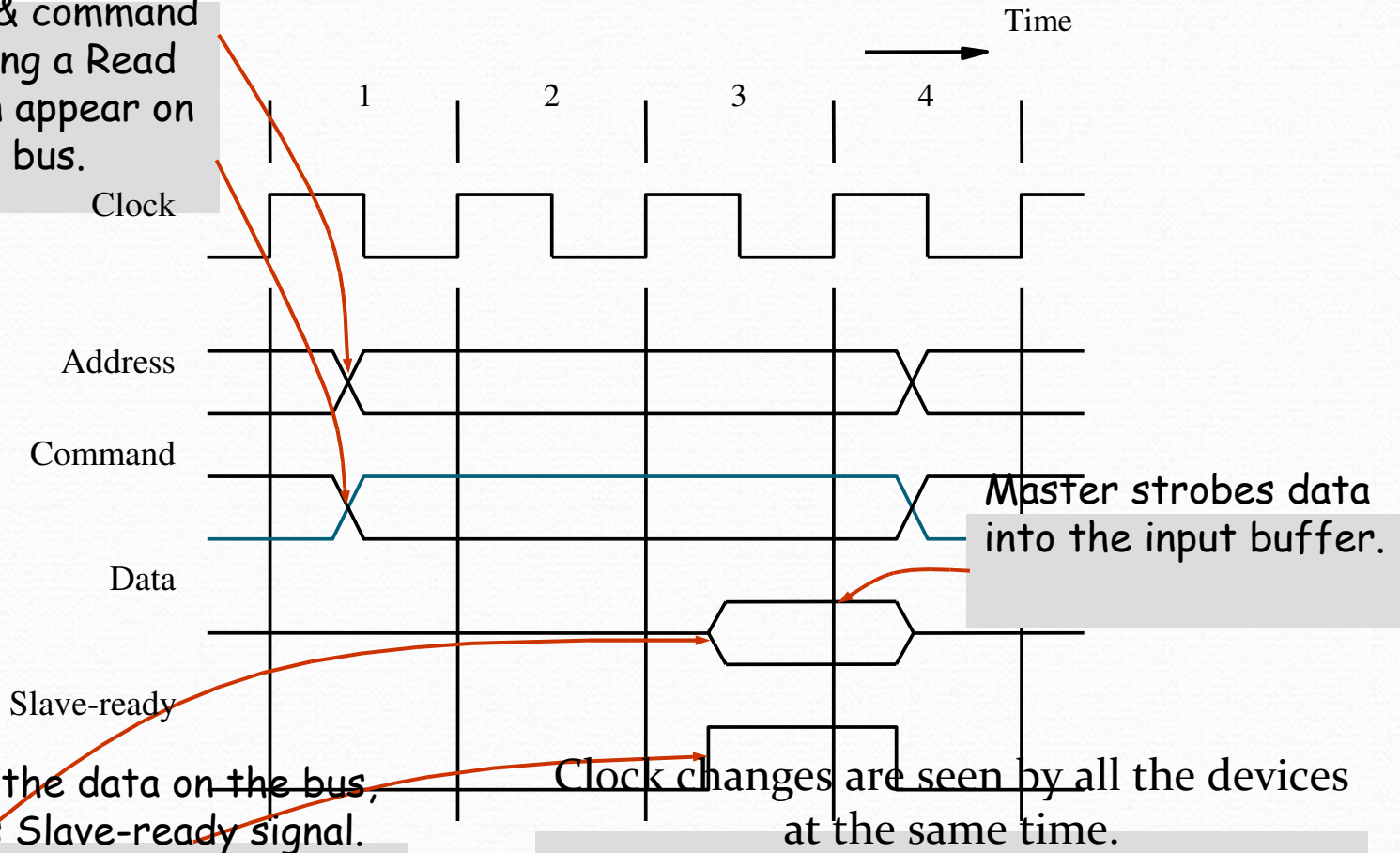
- Data transfer has to be completed within one clock cycle.
 - Clock period t_2 - to must be such that the longest propagation delay on the bus and the slowest device interface must be accommodated.
 - Forces all the devices to operate at the speed of the slowest device.
- Processor just assumes that the data are available at t_2 in case of a Read operation, or are read by the device in case of a Write operation.
 - What if the device is actually failed, and never really responded?

Synchronous bus (contd..)

- Most buses have control signals to represent a response from the slave.
- Control signals serve two purposes:
 - Inform the master that the slave has recognized the address, and is ready to participate in a data transfer operation.
 - Enable to adjust the duration of the data transfer operation based on the speed of the participating slaves.
- High-frequency bus clock is used:
 - Data transfer spans several clock cycles instead of just one clock cycle as in the earlier case.

Synchronous bus (contd..)

Address & command requesting a Read operation appear on the bus.



Slave places the data on the bus, and asserts Slave-ready signal.

Clock changes are seen by all the devices at the same time.

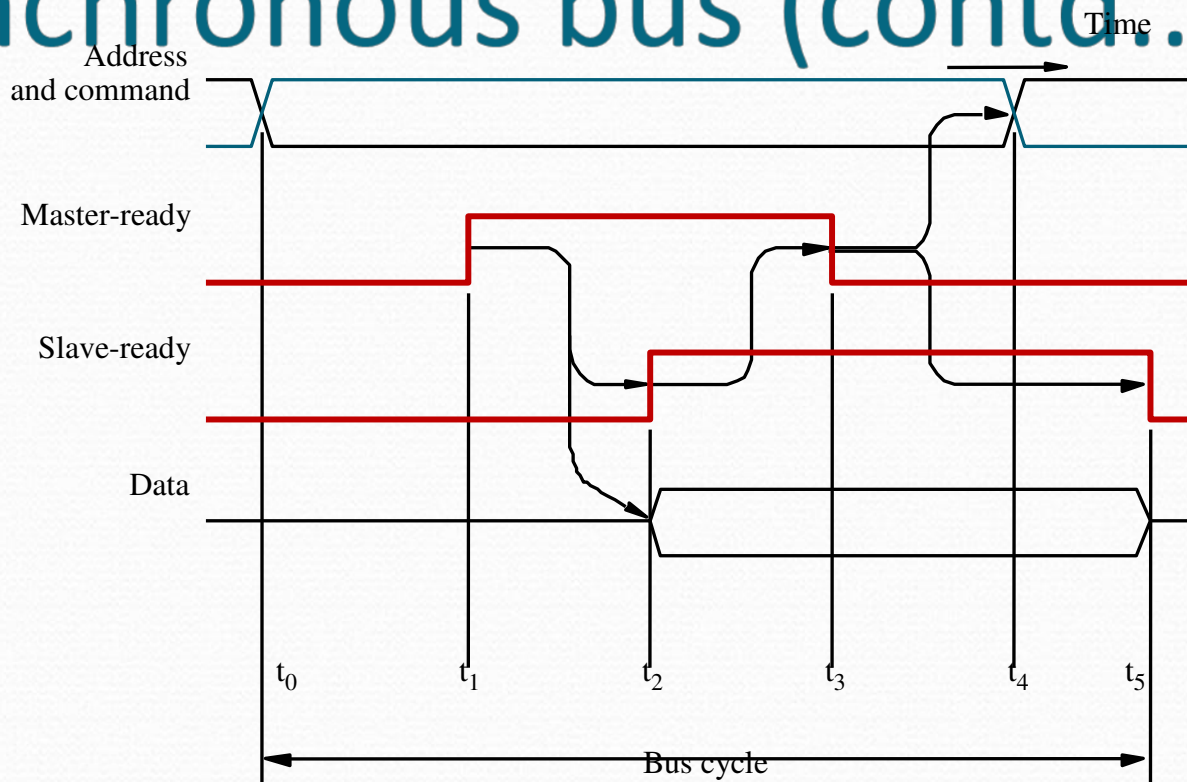
Asynchronous bus

- Data transfers on the bus is controlled by a handshake between the master and the slave.
- Common clock in the synchronous bus case is replaced by two timing control lines:
 - Master-ready,
 - Slave-ready.
- Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate in a data transfer.
- Slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to participate in a data transfer.

Asynchronous bus (contd..)

- Data transfer using the handshake protocol:
 - Master places the address and command information on the bus.
 - Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
 - All devices on the bus decode the address.
 - Address slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
 - Master removes all the signals from the bus, once Slave-ready is asserted.
 - If the operation is a Read operation, Master also strobcs the data into its input buffer.

Asynchronous bus (contd..)



t_0 - Master places the address and command information on the bus.

t_1 - Master asserts the Master-ready signal. Master-ready signal is asserted at t_1 instead of t_0 .

t_2 - Addressed slave places the data on the bus and asserts the Slave-ready signal.

t_3 - Slave-ready signal arrives at the master.

t_4 - Master removes the address and command information.

t_5 - Slave receives the transition of the Master-ready signal from 1 to 0. It removes the data and the Slave-ready signal from the bus.

Asynchronous vs. Synchronous bus

- **Advantages of asynchronous bus:**
 - Eliminates the need for synchronization between the sender and the receiver.
 - Can accommodate varying delays automatically, using the Slave-ready signal.
- **Disadvantages of asynchronous bus:**
 - Data transfer rate with full handshake is limited by two-round trip delays.
 - Data transfers using a synchronous bus involves only one round trip delay, and hence a synchronous bus can achieve faster rates.

Interface Circuits

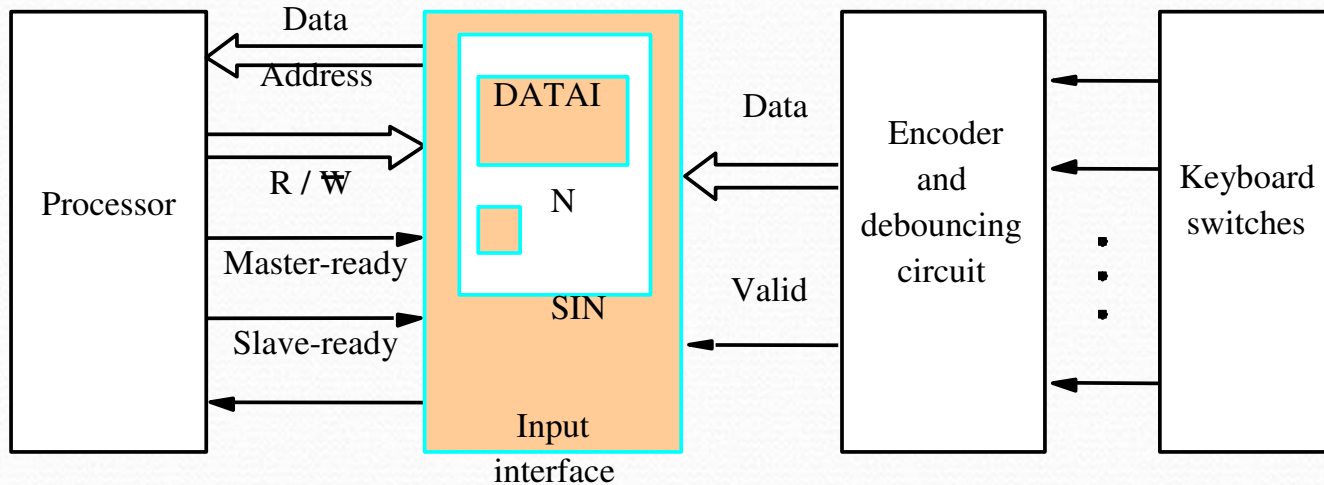
Interface circuits

- I/O interface consists of the circuitry required to connect an I/O device to a computer bus.
- Side of the interface which connects to the computer has bus signals for:
 - Address,
 - Data
 - Control
- Side of the interface which connects to the I/O device has:
 - Datapath and associated controls to transfer data between the interface and the I/O device.
 - This side is called as a “port”.
- Ports can be classified into two:
 - Parallel port,
 - Serial port.

Interface circuits (contd..)

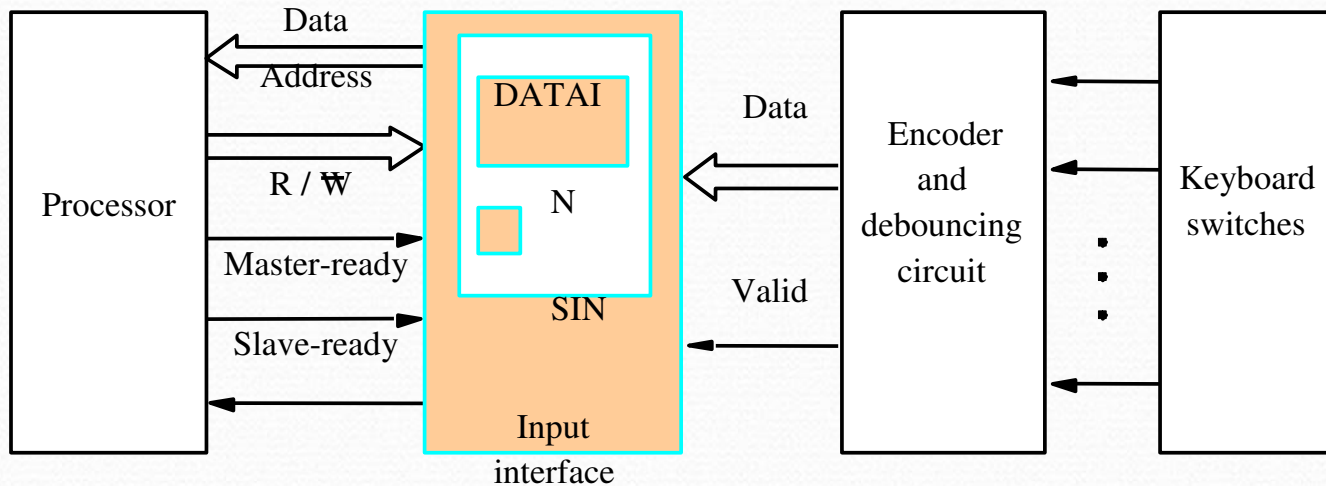
- Parallel port transfers data in the form of a number of bits, normally 8 or 16 to or from the device.
- Serial port transfers and receives data one bit at a time.
- Processor communicates with the bus in the same way, whether it is a parallel port or a serial port.
 - Conversion from the parallel to serial and vice versa takes place inside the interface circuit.

Parallel port



- *Keyboard is connected to a processor using a parallel port.*
- *Processor is 32-bits and uses memory-mapped I/O and the asynchronous bus protocol.*
- *On the processor side of the interface we have:*
 - *Data lines.*
 - *Address lines*
 - *Control or R/W line.*
 - *Master-ready signal and*
 - *Slave-ready signal.*

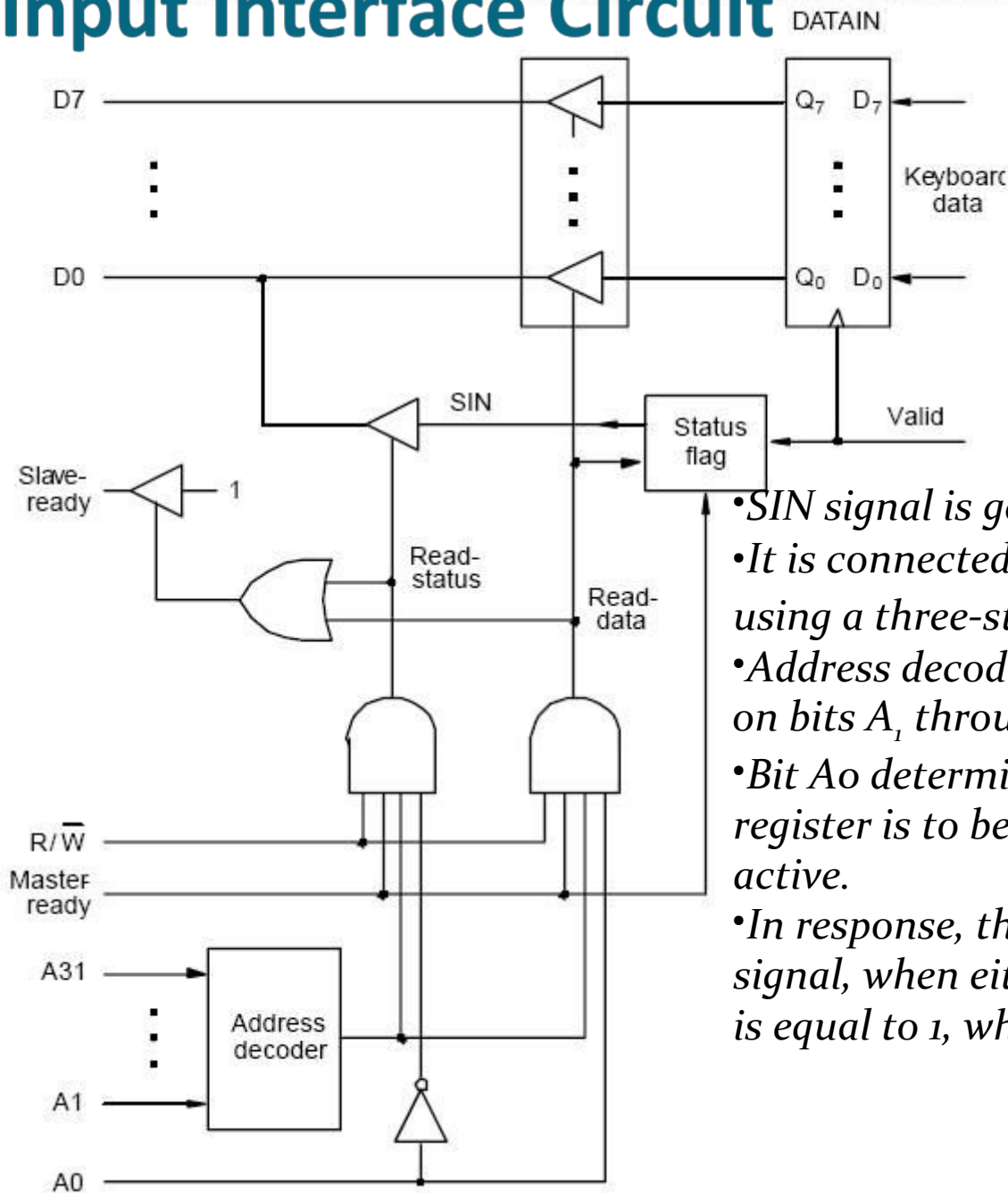
Parallel port (contd..)



- *On the keyboard side of the interface:*

- *Encoder circuit which generates a code for the key pressed.*
- *Debouncing circuit which eliminates the effect of a key bounce (a single key stroke may appear as multiple events to a processor).*
- *Data lines contain the code for the key.*
- *Valid line changes from 0 to 1 when the key is pressed. This causes the code to be loaded into DATAI and SIN to be set to 1.*

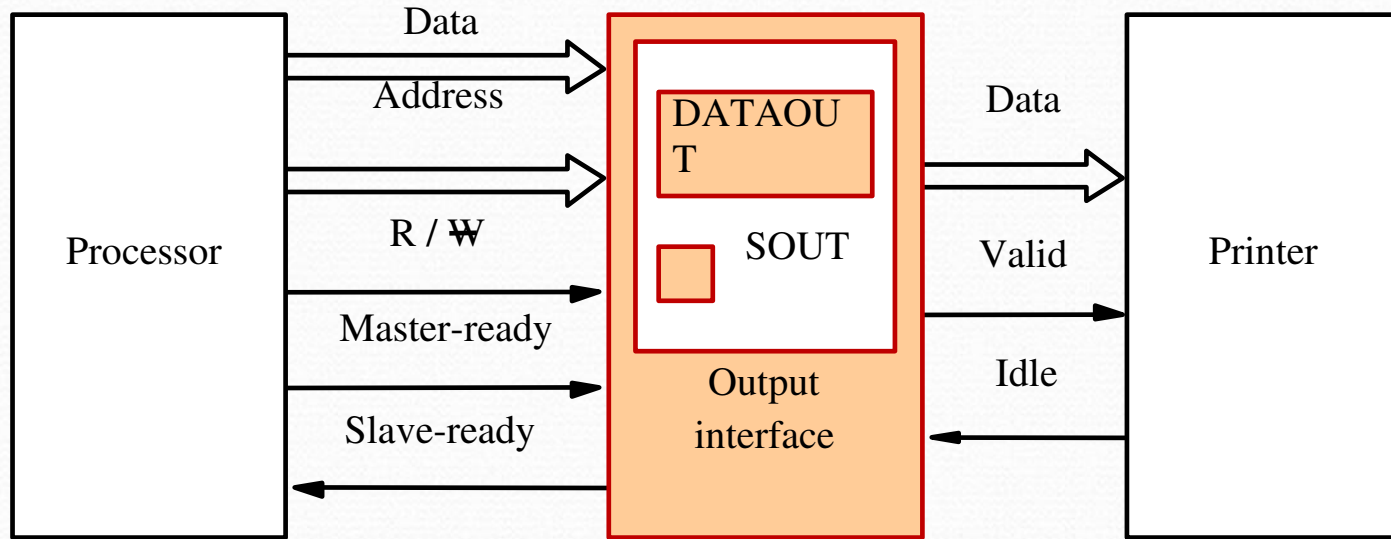
Input Interface Circuit



- Output lines of *DATAIN* are connected to the data lines of the bus by means of 3 state drivers
- Drivers are turned on when the processor issues a read signal and the address selects this register.

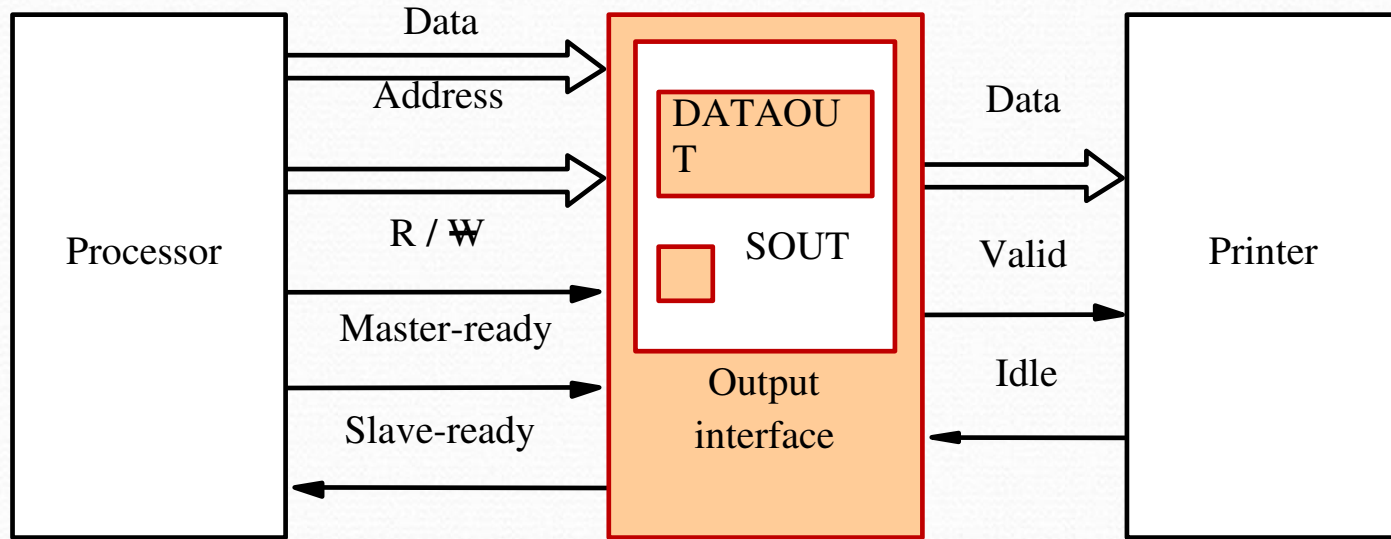
- *SIN* signal is generated using a status flag circuit.
- It is connected to line D_0 of the processor bus using a three-state driver.
- Address decoder selects the input interface based on bits A_1 through A_{31} .
- Bit A_0 determines whether the status or data register is to be read, when Master-ready is active.
- In response, the processor activates the Slave-ready signal, when either the Read-status or Read-data is equal to 1, which depends on line A_0 .

Parallel port (contd..)



- *Printer is connected to a processor using a parallel port.*
- *Processor is 32 bits, uses memory-mapped I/O and asynchronous bus protocol.*
- *On the processor side:*
 - *Data lines.*
 - *Address lines*
 - *Control or R/W line.*
 - *Master-ready signal and*
 - *Slave-ready signal.*

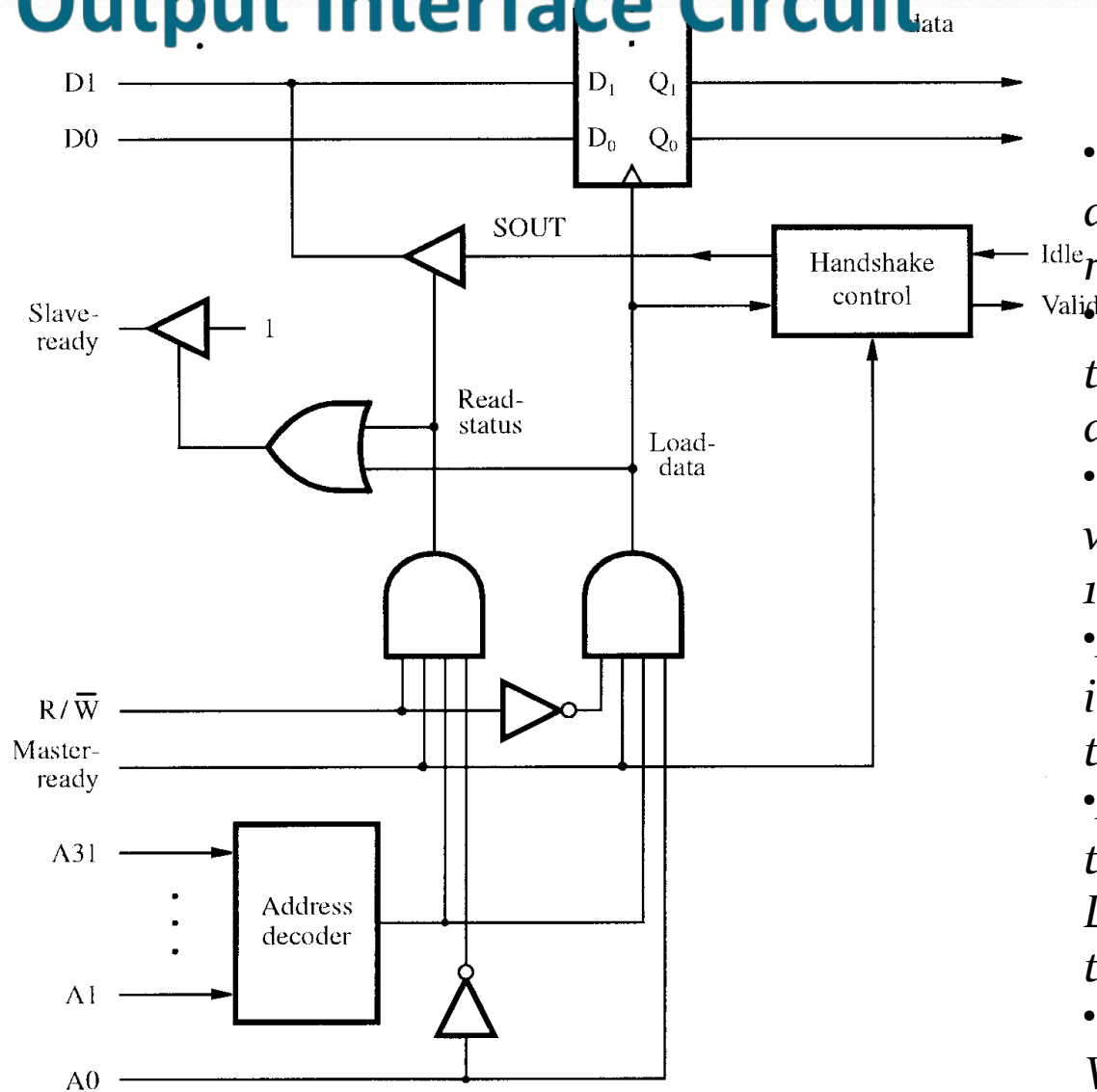
Parallel port (contd..)



•On the printer side:

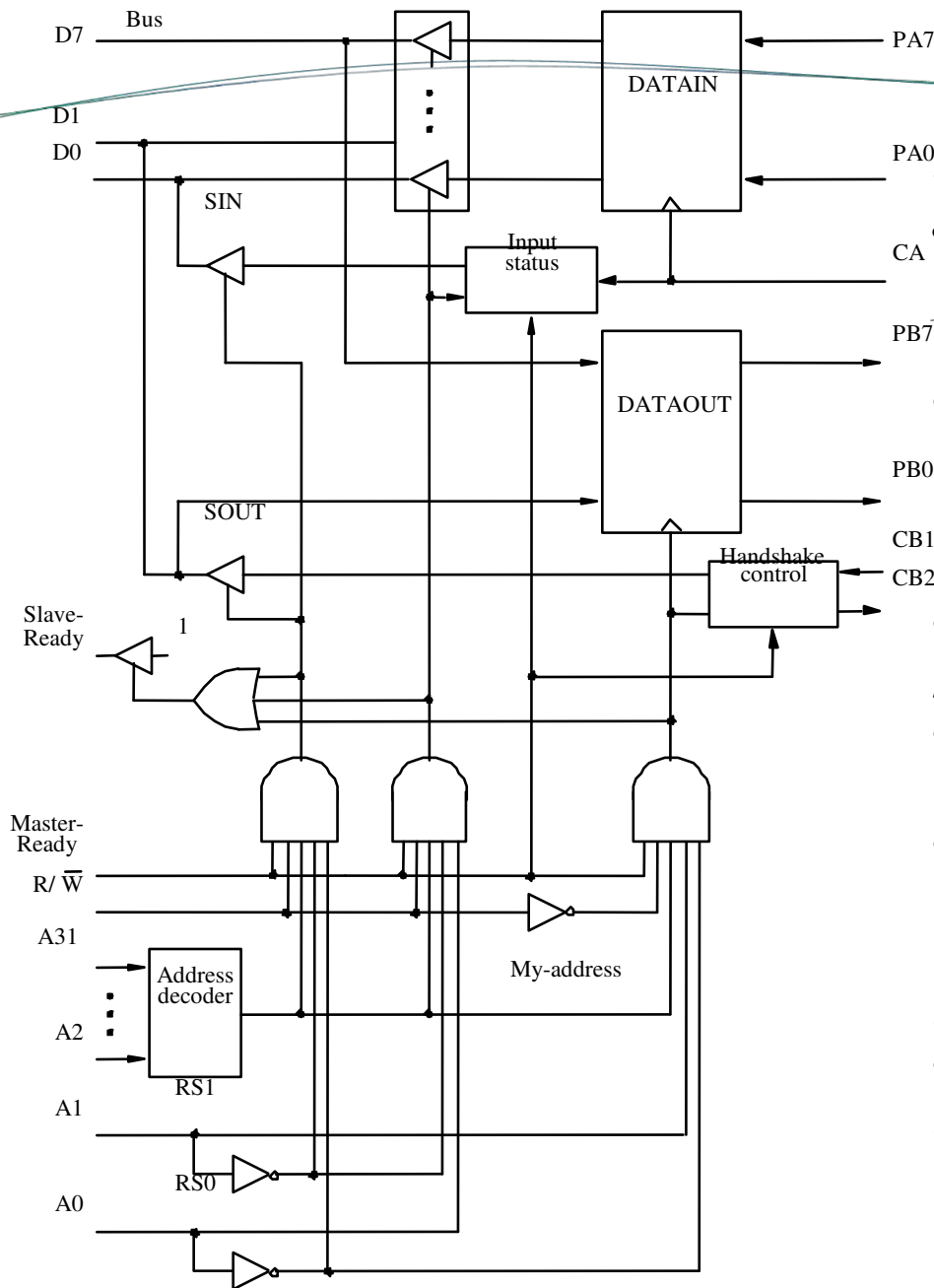
- Idle signal line which the printer asserts when it is ready to accept a character. This causes the SOUT flag to be set to 1.
- Processor places a new character into a DATAOUT register.
- Valid signal, asserted by the interface circuit when it places a new character on the data lines.

Output Interface Circuit

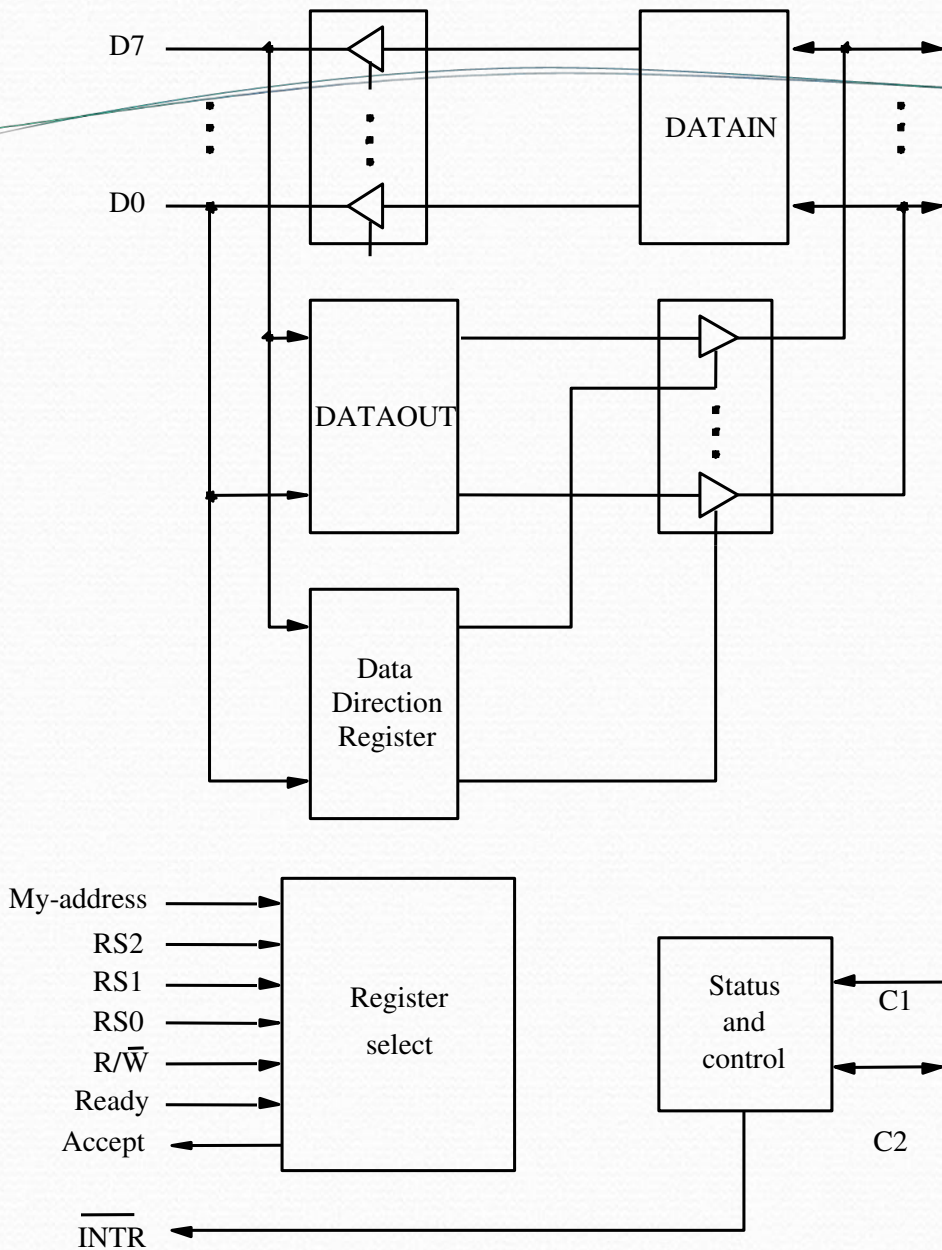


- Data lines of the processor bus are connected to the DATAOUT register of the interface.
- The status flag SOUT is connected to the data line D1 using a three-state driver.
- The three-state driver is turned on, when the control Read-status line is 1.
- Address decoder selects the output interface using address lines A1 through A31.
- Address line A0 determines whether the data is to be loaded into the DATAOUT register or status flag is to be read.
- If the Load-data line is 1, then the Valid line is set to 1.
- If the Idle line is 1, then the status flag SOUT is set to 1.

Figure 4.32. Output interface circuit.



- Combined I/O interface circuit.
- Address bits A2 through A31, that is 30 bits are used to select the overall interface.
- Address bits A1 through A0, that is, 2 bits select one of the three registers, namely, DATAIN, DATAOUT, and the status register.
- Status register contains the flags SIN and SOUT in bits 0 and 1.
- Data lines PA0 through PA7 connect the input device to the DATAIN register.
- DATAOUT register connects the data lines on the processor bus to lines PB0 through PB7 which connect to the output device.
- Separate input and output data lines for connection to an I/O device.

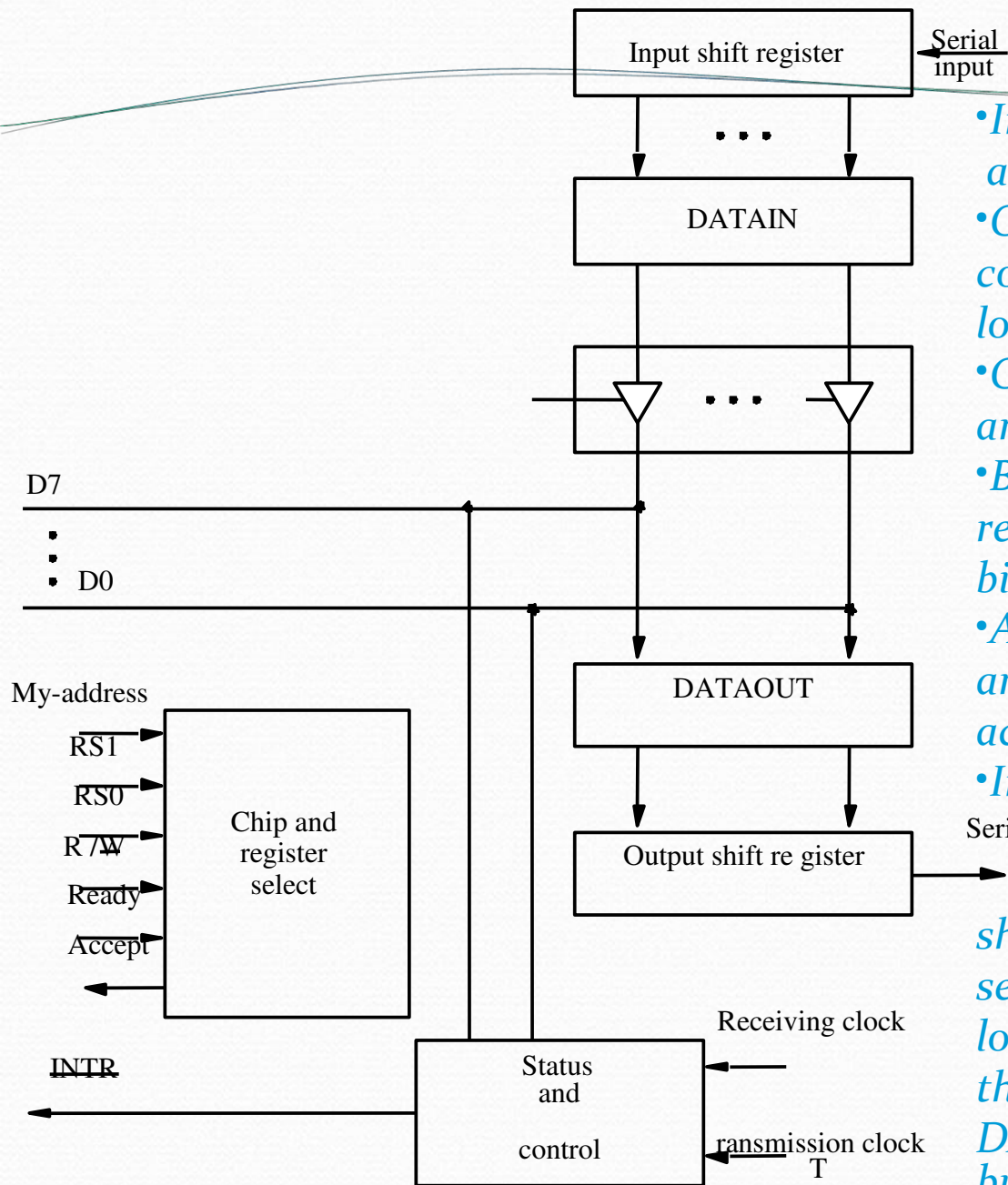


^{P.7} Data lines to I/O device are bidirectional.

- Data lines P₇ through P₀ can be used for both input, and output.
- In fact, some lines can be used for input & some for output depending on the pattern in the Data Direction Register (DDR).
- Processor places an 8-bit pattern into a DDR
- If a given bit position in the DDR is 1, the corresponding data line acts as an output line, otherwise it acts as an input line.
- C₁ and C₂ control the interaction between the interface circuit and the I/O devices.
- Ready and Accept lines are the handshake control lines on the processor bus side, and are connected to Master-ready & Slave-ready
- Input signal My-address is connected to the output of an address decoder.
- Three register select lines that allow up to 8 registers to be selected.

Serial port

- Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- Serial port communicates in a bit-serial fashion on the device side and bit parallel fashion on the bus side.
 - Transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.



- Input shift register accepts input one bit at a time from the I/O device.
- Once all the 8 bits are received, the contents of the input shift register are loaded in parallel into DATAIN register.
- Output data in the DATAOUT register are loaded into the output shift register.
- Bits are shifted out of the output shift register and sent out to the I/O device one bit at a time.
- As soon as data from the input shift reg. are loaded into DATAIN, it can start accepting another 8 bits of data.
- Input shift register and DATAIN register are both used at input so that the shift register can start receiving another set of 8 bits from the input device after loading the contents to DATAIN, before the processor reads the contents of DATAIN. This is called as double-buffering.

Serial port (contd..)

- Serial interfaces require fewer wires, and hence serial transmission is convenient for connecting devices that are physically distant from the computer.
- Speed of transmission of the data over a serial interface is known as the “bit rate”.
 - Bit rate depends on the nature of the devices connected.
- In order to accommodate devices with a range of speeds, a serial interface must be able to use a range of clock speeds.
- Several standard serial interfaces have been developed:
 - Universal Asynchronous Receiver Transmitter (UART) for low-speed serial devices.
 - RS-232-C for connection to communication links.

Standard I/O interfaces

- I/O device is connected to a computer using an interface circuit.
- Do we have to design a different interface for every combination of an I/O device and a computer?
- A practical approach is to develop standard interfaces and protocols.
- A personal computer has:
 - A motherboard which houses the processor chip, main memory and some I/O interfaces.
 - A few connectors into which additional interfaces can be plugged.
- Processor bus is defined by the signals on the processor chip.

- Devices which require high-speed connection to the processor are connected directly to this bus.

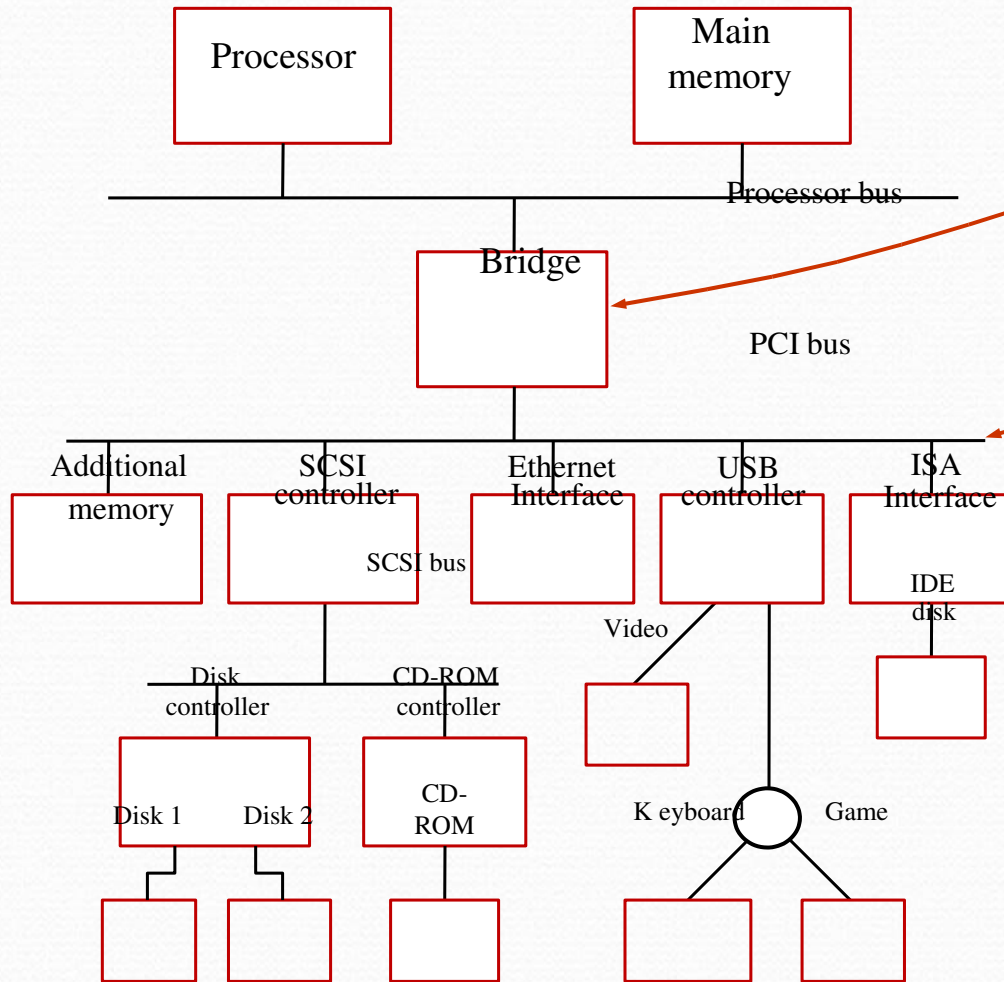
Standard I/O interfaces (contd..)

- Because of electrical reasons only a few devices can be connected directly to the processor bus.
- Motherboard usually provides another bus that can support more devices.
 - Processor bus and the other bus (called as expansion bus) are interconnected by a circuit called “bridge”.
 - Devices connected to the expansion bus experience a small delay in data transfers.
- Design of a processor bus is closely tied to the architecture of the processor.
 - No uniform standard can be defined.
- Expansion bus however can have uniform standard defined.

Standard I/O interfaces (contd..)

- A number of standards have been developed for the expansion bus.
 - Some have evolved by default.
 - For example, IBM's Industry Standard Architecture.
 - **Three widely used bus standards:**
 - PCI (Peripheral Component Interconnect)
 - SCSI (Small Computer System Interface)
 - USB (Universal Serial Bus)

Standard I/O interfaces (contd..)



Bridge circuit translates signals and protocols from processor bus to PCI bus.

Expansion bus on the motherboard

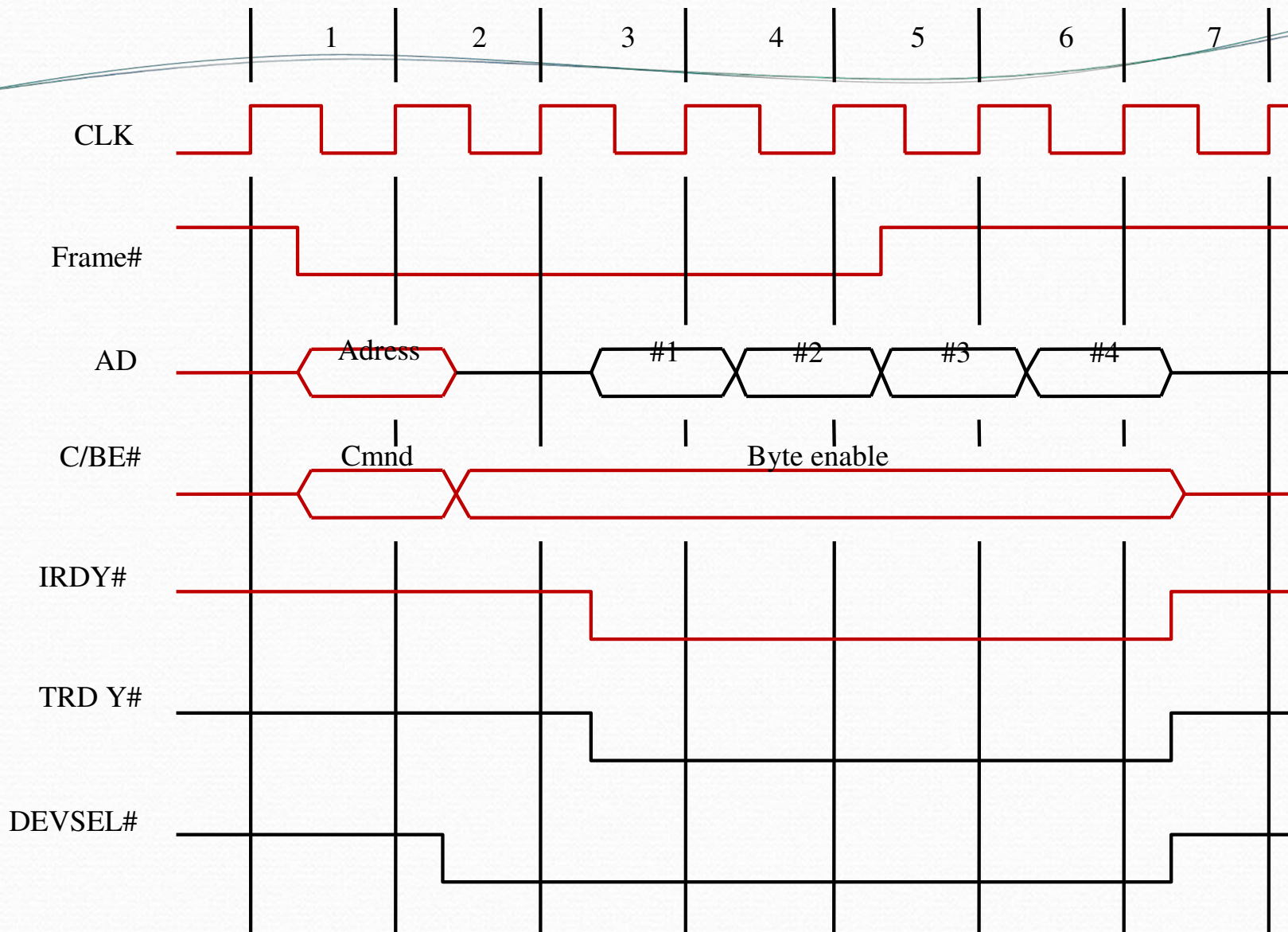
PCI Bus

- *Peripheral Component Interconnect*
- Introduced in 1992
- Low-cost bus
- Processor independent
- Plug-and-play capability
- In today's computers, most memory transfers involve a burst of data rather than just one word. The PCI is designed primarily to support this mode of operation.
- The bus supports three independent address spaces: memory, I/O, and configuration.
- we assumed that the master maintains the address information on the bus until data transfer is completed. But, the address is needed only long enough for the slave to be selected. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles. The result is a significant cost reduction.
- A master is called an initiator in PCI terminology. The addressed device that responds to read and write commands is called a target.

Data transfer signals on the PCI bus.

Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus). IRDY#,
TRDY#	Initiator-ready and Target-ready signals.
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction.

IDSEL# Initialization Device Select.



A read operation on the PCI bus

Device Configuration

- When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.
- PCI incorporates in each I/O device interface a small configuration ROM memory that stores information about that device.
- The configuration ROMs of all devices are accessible in the configuration address space. The PCI initialization software reads these ROMs and determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn about various device options and characteristics.
- Devices are assigned addresses during the initialization process.
- This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one.
- Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#
- **Electrical characteristics:**

- PCI bus has been defined for operation with either a 5 or 3.3 V power supply

SCSI Bus

- The acronym SCSI stands for Small Computer System Interface.
- It refers to a standard bus defined by the American National Standards Institute (ANSI) under the designation X3.131 .
- In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to 5 megabytes/s.
- The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years.
- SCSI-2 and SCSI-3 have been defined, and each has several options.
- Because of various options SCSI connector may have 50, 68 or

80 pins.

SCSI Bus (Contd.,)

- Devices connected to the SCSI bus are not part of the address space of the processor
- The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa.
- A packet may contain a block of data, commands from the processor to the device, or status information about the device.
- A controller connected to a SCSI bus is one of two types – an initiator or a target.
- An initiator has the ability to select a particular target and to send commands specifying the operations to be performed. The disk controller operates as a target. It carries out the commands it receives from the initiator.
- The initiator establishes a logical connection with the intended target.
- Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data.
- While a particular connection is suspended, other device can use the bus to transfer information.
- This ability to overlap data transfer requests is one of the key features of the SCSI

bus that leads to its high performance.

SCSI Bus (Contd.,)

- Data transfers on the SCSI bus are always controlled by the target controller.
- To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.
- Then the controller starts a data transfer operation to receive a command from the initiator.

SCSI Bus (Contd.,)

- Assume that processor needs to read block of data from a disk drive and that data are stored in disk sectors that are not contiguous.
- The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:
 1. The SCSI controller, acting as an initiator, contends for control of the bus.
 2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
 3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
 4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
 5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.

SCSI Bus (Contd.,)

1. The target transfers the contents of the data buffer to the initiator and then suspends the connection again
2. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfers, the logical connection between the two controllers is terminated.
3. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
4. The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been completed

Operation of SCSI bus from H/W point of view

Category	Name	Function
Data	-DB(0)to	Datalines:Carry onebyte of information
	-DB(7)	duringtheinformation transfer phase and iden tify deviceduringarbitration,selection and reselection phases
Phase	-DB(P)	Paritybit forthedatabus
	-BSY	Busy:Asserted when thebus isnotfree
	-SEL	Selection:Assertedduringselection and reselection
Information type	-C/D	Control/Data:Asserted duringtransfer of control information (command,status or message)

-MSG Message:indicates thattheinformation being
transferred is amessage

Table 4. The SCSI bus signals.

Table 4. The SCSI bus signals.(*cont.*)

Category	Name	Function
Handshake	- REQ	Request: Asserted by a target to request a data transfer cycle
	- ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	- I/O	Input/Output: Asserted to indicate an input operation (relative to the initiator)
Other	- ATN	Attention: Asserted by an initiator when it wishes to send a message to a target

–

uses all device controls to disconnect from the
bus and assume their start-up state

R

S

T

R

e

s

e

t

:

C

a

Main Phases involved

- Arbitration
 - A controller requests the bus by asserting BSY and by asserting it's associated data line
 - When BSY becomes active, all controllers that are requesting bus examine data lines
- Selection
 - Controller that won arbitration selects target by asserting SEL and data line of target. After that initiator releases BSY line.
 - Target responds by asserting BSY line
 - Target controller will have control on the bus from then
- Information Transfer
 - Handshaking signals are used between initiator and target
 - At the end target releases BSY line
- Reselection

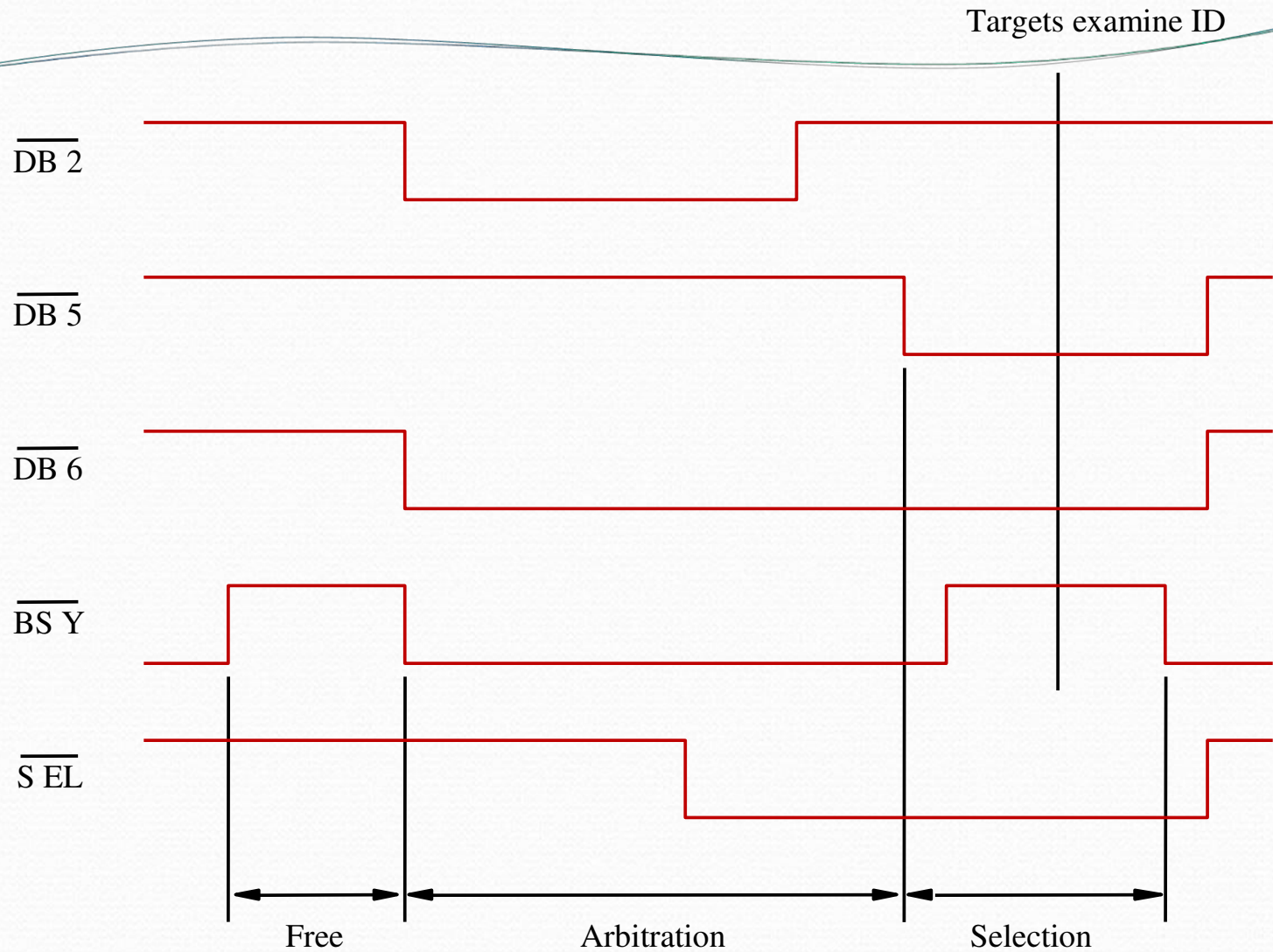
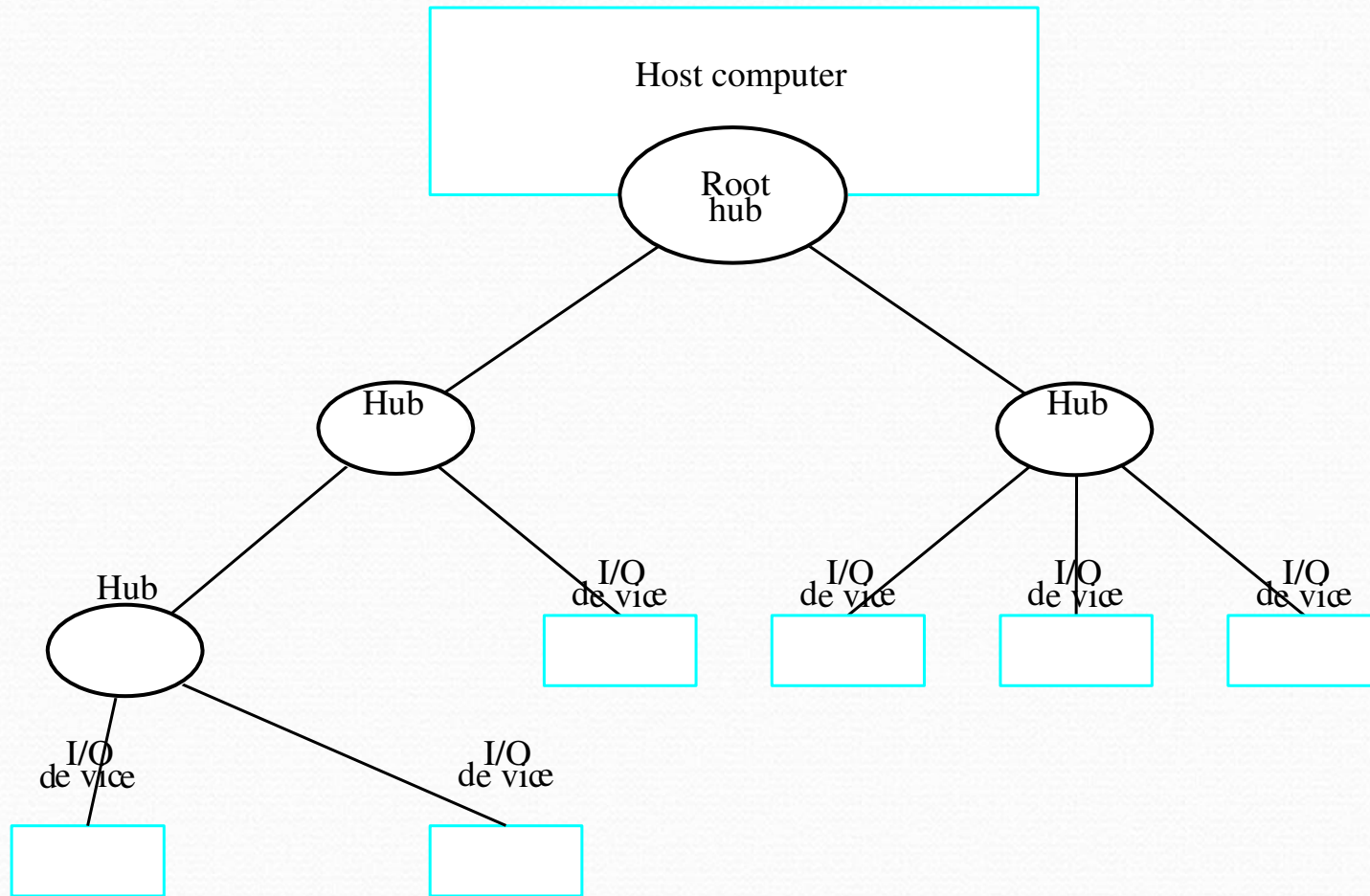


Figure 42. Arbitration and selection on the SCSI bus.
 Device 6 wins arbitration and selects device 2.

USB

- Universal Serial Bus (USB) is an industry standard developed through a collaborative effort of several computer and communication companies, including Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nortel Networks, and Philips.
- Speed
 - Low-speed(1.5 Mb/s)
 - Full-speed(12 Mb/s)
 - High-speed(480 Mb/s)
- Port Limitation
- Device Characteristics
- Plug-and-play

Universal Serial Bus tree structure



Universal Serial Bus tree structure

- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.
- Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root hub connects the entire tree to the host computer. The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker, or digital TV)
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message. However, a message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, the USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

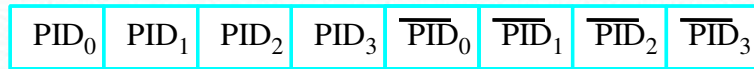
Addressing

- When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree.
- Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.
- A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily. When a device is first connected to a hub, or when it is powered on, it has the address 0. The hardware of the hub to which this device is connected is capable of detecting that the device has been connected, and it records this fact as part of its own status information. Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected.
- When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.

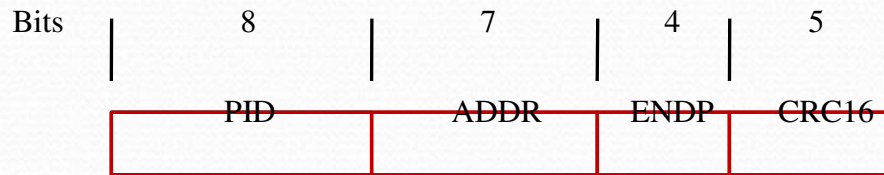
USB Protocols

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information. There are many types of packets that perform a variety of control functions.
- The information transferred on the USB can be divided into two broad categories: control and data.
 - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error.
 - Data packets carry information that is delivered to a device.
- A packet consists of one or more fields containing different kinds of information. The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.
- They are transmitted twice. The first time they are sent with their true values, and the second time with each bit complemented

- The four PID bits identify one of 16 different packet types. Some control packets, such as ACK (Acknowledge), consist only of the PID byte.

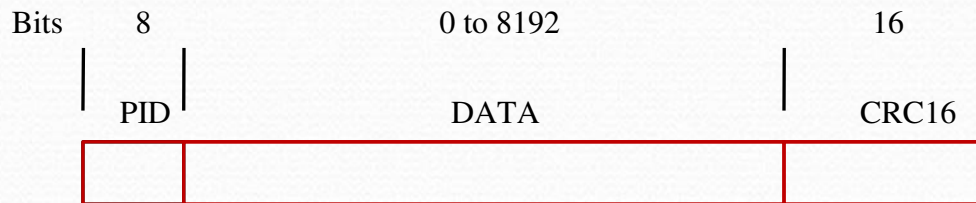


(a) Packet identifier field



(b) Token packet, IN or OUT

Control packets used for controlling data transfer operations are called token packets.



(c) Data packet

Figure 45. USB packet format.

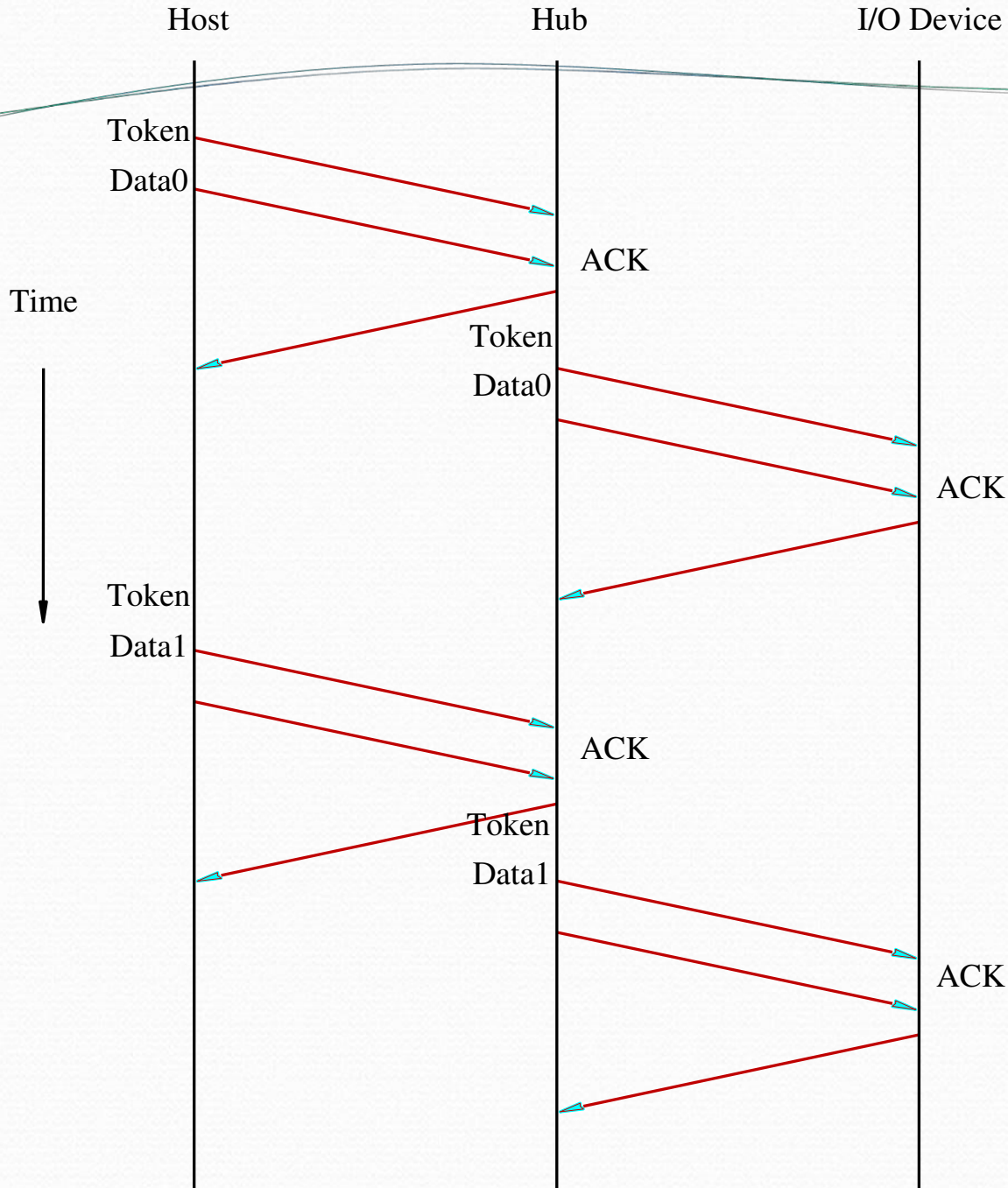


Figure: An output transfer

Isochronous Traffic on USB

- One of the key objectives of the USB is to support the transfer of isochronous data.
- Devices that generate or receive isochronous data require a time reference to control the sampling process.
- To provide this reference, transmission over the USB is divided into frames of equal length.
- A frame is 1ms long for low- and full-speed data.
- The root hub generates a Start of Frame control packet (SOF) precisely once every 1 ms to mark the beginning of a new frame.
- The arrival of an SOF packet at any device constitutes a regular clock signal that the device can use for its own purposes.
- To assist devices that may need longer periods of time, the SOF packet carries an 11-bit frame number.
- Following each SOF packet, the host carries out input and output transfers for isochronous devices.

- This means that each device will have an opportunity for an input or output transfer once every 1 ms.

Electrical Characteristics

- The cables used for USB connections consist of four wires.
- Two are used to carry power, +5V and Ground.
 - Thus, a hub or an I/O device may be powered directly from the bus, or it may have its own external power connection.
- The other two wires are used to carry data.
- Different signaling schemes are used for different speeds of transmission.

- At low speed, 1s and 0s are transmitted by sending a high voltage state (5V) on one or the other of the two signal wires. For high-speed links, differential transmission is used.