COMPUTER ARCHITECTURE AND ORGANIZATION.

UNIT-1

Presented by
Dr.K.RAJKAMAL,
Assoc.Prof,
Dept of ECE,
KHIT.

		L	T	P	Cr.
B.Tech. (5_Sem.)	COMPUTER ARCHITECTURE AND ORGANIZATION (Theory)	3	0	0	3

Pre-requisites: Digital Logic Design

Course Educational Objective:

1. To understand the architecture of a modern computer with its various processing units. Also the Performance measurement of the computer system.
2. To understand the memory management system of computer.
3. To understand the various instructions, addressing modes
4. To understand the concept of I/O organization

Course Outcomes:

At the end of the course, the student will be able to
CO1: Understand the architecture of modern computer.
CO2: Analyze the Performance of a computer using performance equation
CO3: Understanding of different instruction types.
CO4: Calculate the effective address of an operand by addressing modes
CO5: Understand the concepts of I/O Organization and Memory systems.

UNIT – I

Basic Structure Of Computers: Functional unit, Basic Operational concepts, Bus structures, System Software, Performance, The history of computer development.

Machine Instruction and Programs:

Instruction and Instruction Sequencing: Register Transfer Notation, Assembly Language Notation, Basic Instruction Types,

UNIT – II

Addressing Modes, Basic Input/output Operations, The role of Stacks and Queues in computer programming equation. Component of Instructions: Logic Instructions, shift and Rotate Instructions

Type of Instructions: Arithmetic and Logic Instructions, Branch Instructions, Addressing Modes, Input/output Operations

UNIT – III

INPUT/OUTPUT ORGANIZATION: Accessing I/O Devices, Interrupts: Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Direct Memory Access, Buses: Synchronous Bus, Asynchronous Bus, Interface Circuits, Standard I/O Interface: Peripheral Component Interconnect (PCI) Bus, Universal Serial Bus (USB)

UNIT – IV

The MEMORY SYSTEMS: Basic memory circuits, Memory System Consideration, Read-Only Memory: ROM, PROM, EPROM, EEPROM, Flash Memory, Introduction about Cache Memories

Secondary Storage: Magnetic Hard Disks, Optical Disks,

UNIT – V

Processing Unit: Fundamental Concepts: Register Transfers, Performing an Arithmetic Or Logic Operation, Fetching a Word from Memory, Execution of Complete Instruction, Hardwired Control,

TEXT BOOKS

1. Computer Organization, Carl Hamacher, ZvonksVranesic, SafeaZaky, 5thEdition, McGrawHill, 2011.
2. Computer Architecture and Organization, John P. Hayes, 3rd Edition, McGrawHill, 2002.

REFERENCE

1. Computer Organization and Architecture – William Stallings Sixth Edition, Pearson/PHI
2. Structured Computer Organization – Andrew S. Tanenbaum, 4th Edition PHI/Pearson, 2012.
3. Fundamentals or Computer Organization and Design, - Sivaraama Dandamudi Springer Int.Edition, 2003.
4. “Computer Organization and Design: The Hardware/Software Interface” by David A. Patterson and John L.Hennessy, 1998.
5. J .P. Hayes, "Computer Architecture and Organization", McGraw-Hill,1998.

Architecture

- **Structure and behavior of the computer as seen by the user.**
  - those properties, which directly affect the logical working of a program;
  - the attributes, which are apparent to a programmer

Examples: instruction set and formats, techniques for addressing memory, number of bits used to represent data

Organization

- Organization: interconnection of operational units for realizing the architectural specifications
  - Determination of which hardware should be used and
  - how the parts should be connected together

BASIC STRUCTURE OF COMPUTERS

- **Functional Units**

Functional Units

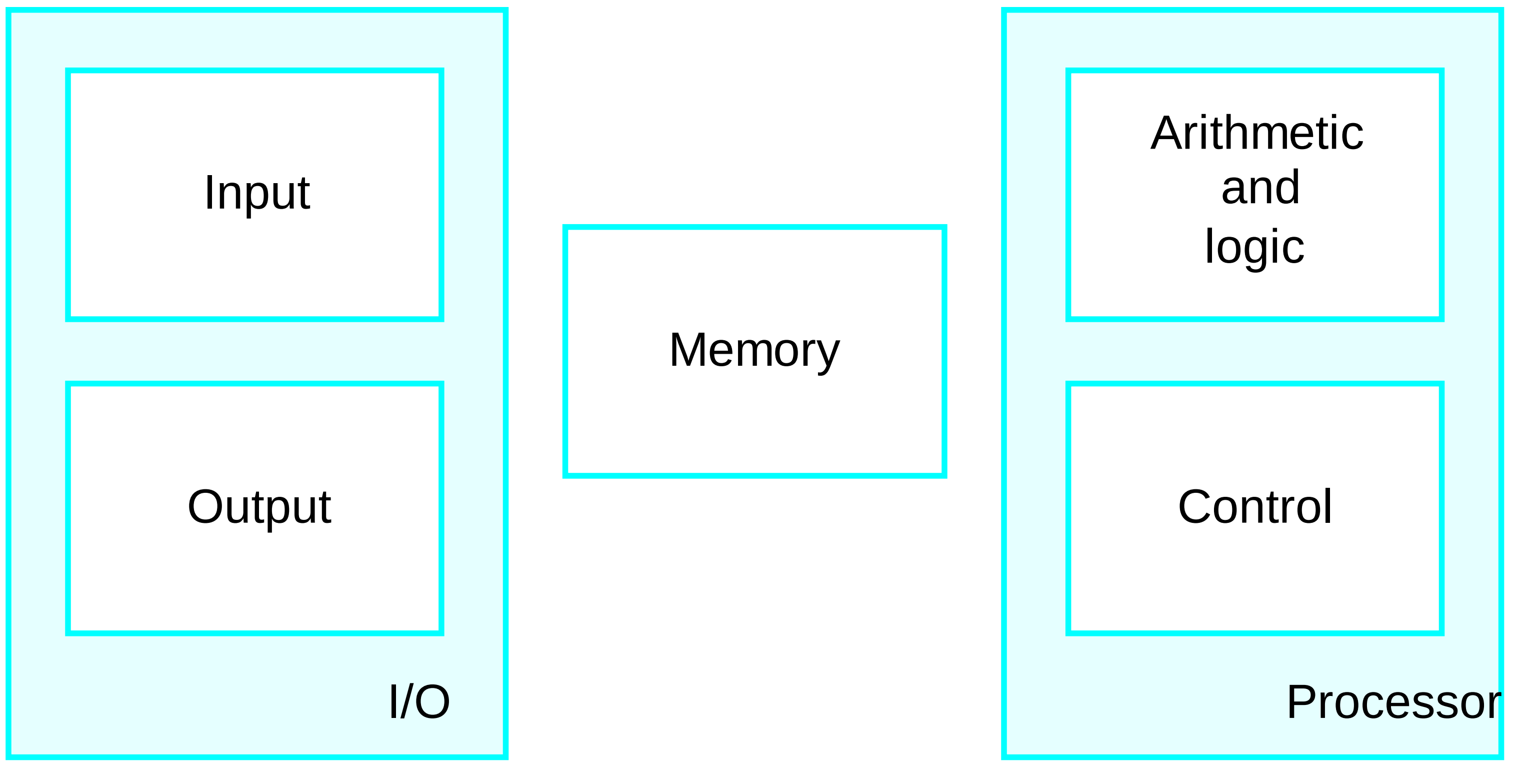


Figure 1.1. Basic functional units of a computer.

Information Handled by a Computer

- Instructions/machine instructions
  - ➢ Govern the transfer of information within a computer as well as between the computer and its I/O devices
  - ➢ Specify the arithmetic and logic operations to be performed
  - ➢ Program
- Data
  - ➢ Used as operands by the instructions
  - ➢ Source program
- Encoded in binary code – 0 and 1

Memory Unit

- Store programs and data
- Two classes of storage

➢ Primary storage

❖ Fast

❖ Programs must be stored in memory while they are being executed

❖ Large number of semiconductor storage cells

❖ Processed in words

❖ Address

❖ RAM and memory access time

❖ Memory hierarchy – cache, main memory

➢ Secondary storage – larger and cheaper

Arithmetic and Logic Unit (ALU)

- Most computer operations are executed in ALU of the processor.
- Load the operands into memory – bring them to the processor – perform operation in ALU – store the result back to memory or retain in the processor.
- Registers
- Fast control of ALU

Control Unit

- All computer operations are controlled by the control unit.
- The timing signals that govern the I/O transfers are also generated by the control unit.
- Control unit is usually distributed throughout the machine instead of standing alone.
- Operations of a computer:

➢ Accept information in the form of programs and data through an input unit and store it in the memory

➢ Fetch the information stored in the memory, under program control, into an ALU, where the information is processed

➢ Output the processed information through an output unit

➢ Control all activities inside the machine through a control unit

BASIC OPERATIONAL CONCEPTS OF COMPUTER

→To perform a given task an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be stored are also stored in the memory.

Examples: - Add LOCA, R0

→This instruction adds the operand at memory location LOCA, to operand in register R0 & places the sum into register. This instruction requires the performance of several steps,

1. First the instruction is fetched from the memory into the processor.
2. The operand at LOCA is fetched and added to the contents of R0
3. Finally the resulting sum is stored in the register R0

BASIC OPERATIONAL CONCEPTS OF COMPUTER

→The preceding add instruction combines a memory access operation with an ALU Operations. In some other type of computers, these two types of operations are performed by separate instructions for performance reasons.

```
Load LOCA, R1
Add R1, R0
```

→ Transfers between the memory and the processor 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 are then transferred to or from the memory

Connection Between the Processor and the Memory

Registers

- Instruction register (IR)
- Program counter (PC)
- General-purpose register (R_0 – R_{n-1})
- Memory address register (MAR)
- Memory data register (MDR)

COMPUTER

→The fig shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

→**The instruction register (IR):-** Holds the instructions that is currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

→**The program counter PC:-**

→This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.

→Besides IR and PC, there are n-general purpose registers R0 through Rn-1.

BASIC OPERATIONAL CONCEPTS OF COMPUTER

The other two registers which facilitate communication with memory are: -

1. **MAR – (Memory Address Register):-** It holds the address of the location to be accessed.
2. **MDR – (Memory Data Register):-** It contains the data to be written into or read out of the address location.

Operating steps are

1. Programs reside in the memory & usually get these through the I/P unit.
2. Execution of the program starts when the PC is set to point at the first instruction of the program.
3. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.

BASIC OPERATIONAL CONCEPTS OF COMPUTER

4. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
5. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
6. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
7. An operand in the memory is fetched by sending its address to MAR & Initiating a read cycle.
8. When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
9. After one or two such repeated cycles, the ALU can perform the desired operation.

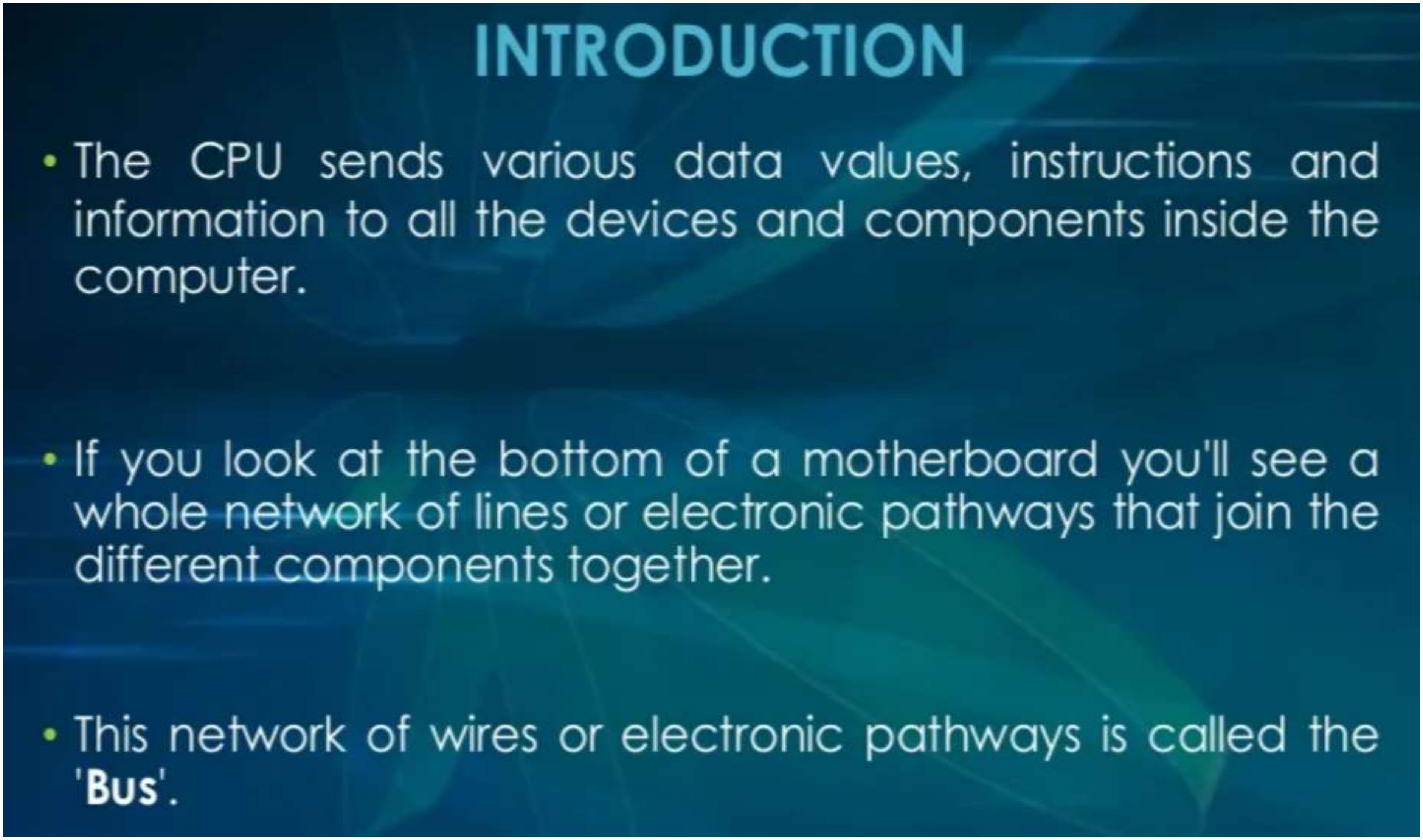
BASIC OPERATIONAL CONCEPTS OF COMPUTER

10. If the result of this operation is to be stored in the memory, the result is sent to MDR.
11. Address of location where the result is stored is sent to MAR & a write cycle is initiated.
12. The contents of PC are incremented so that PC points to the next instruction that is to be executed.

→Normal execution of a program may be preempted (temporarily interrupted) if some devices require urgent servicing, to do this one device raises an Interrupt signal.

→An interrupt is a request signal from an I/O device for service by the processor. The processor provides the requested service by executing an appropriate interrupt service routine.

BUS STRUCTURES

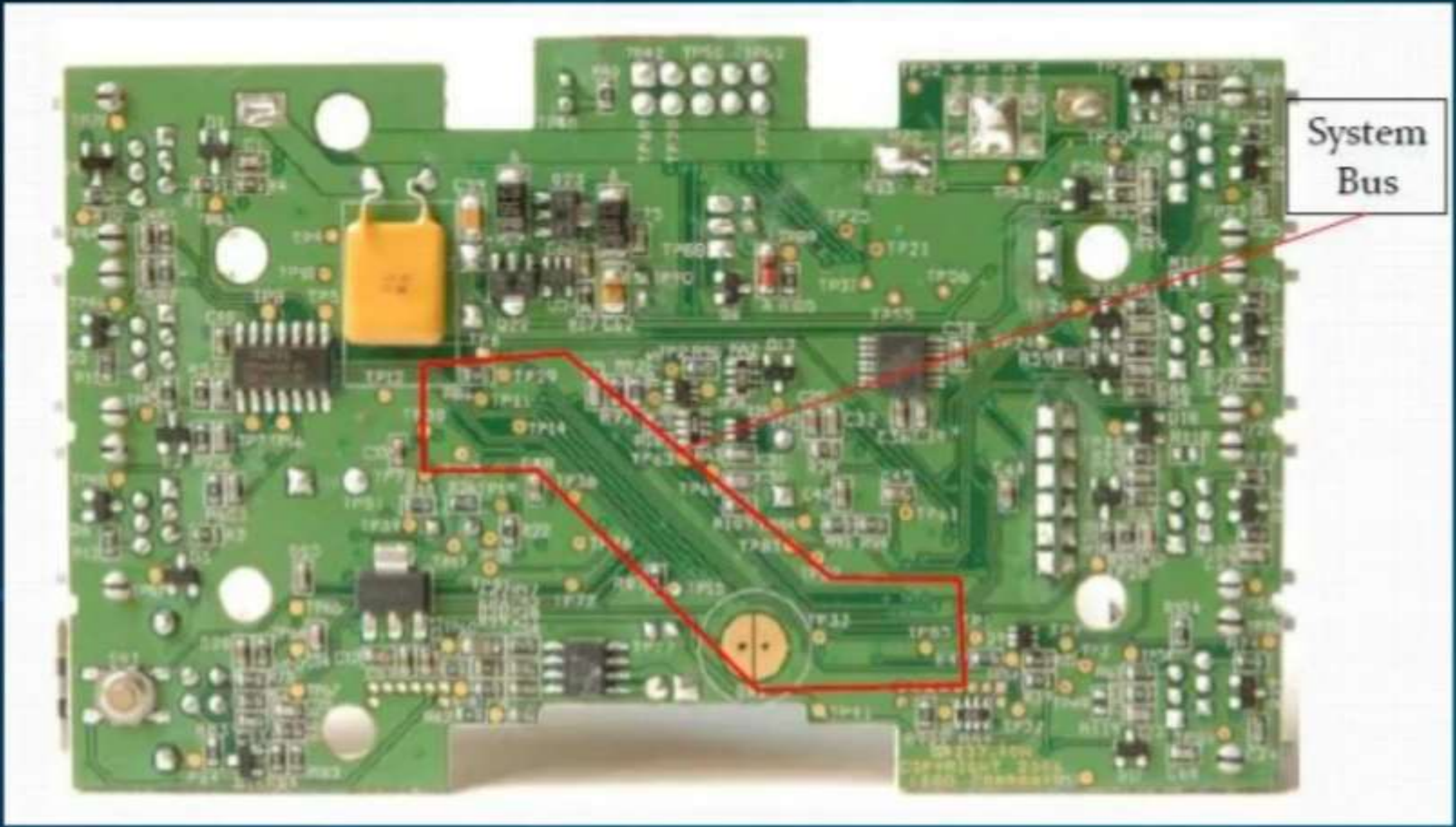


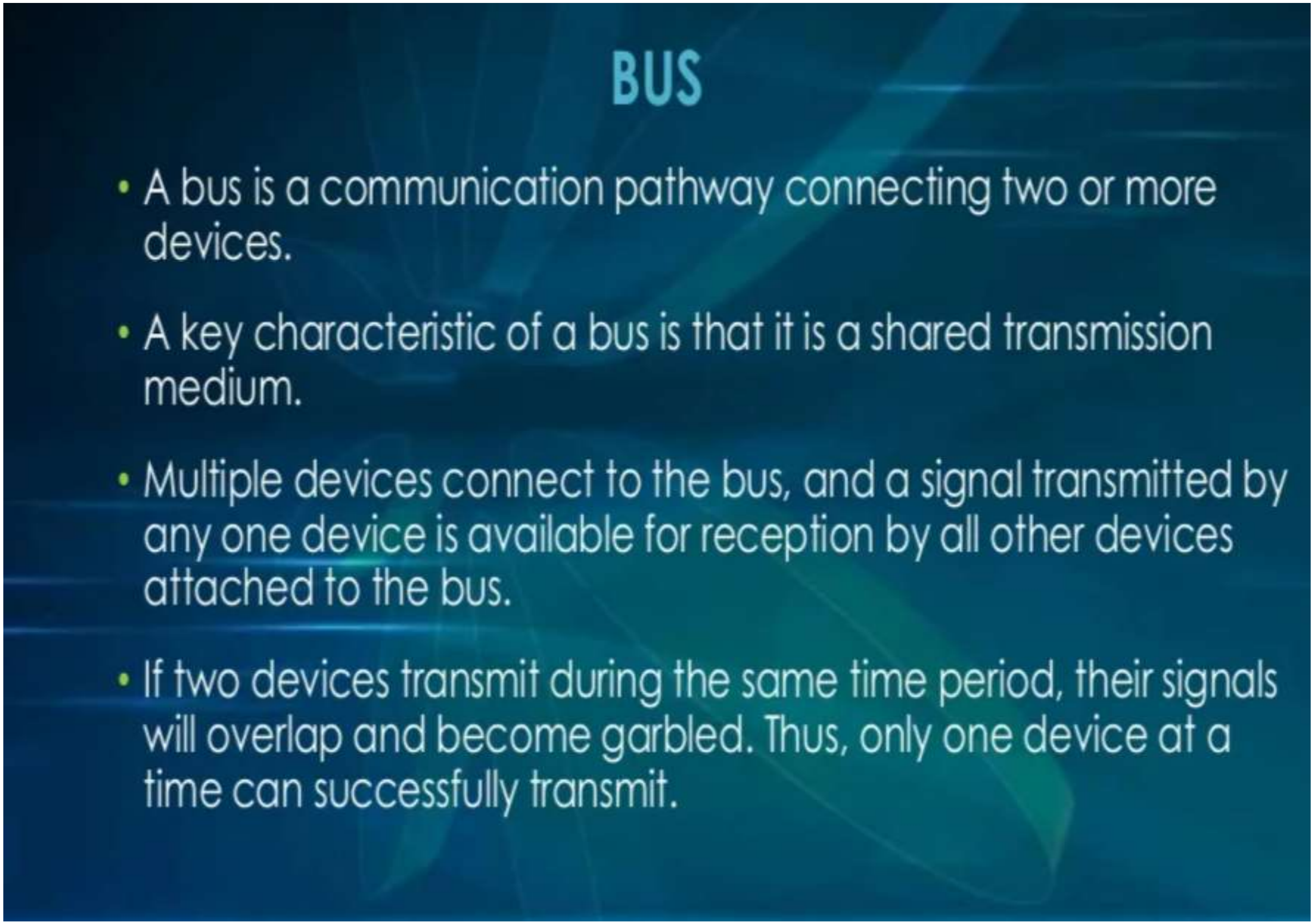
INTRODUCTION

- The CPU sends various data values, instructions and information to all the devices and components inside the computer.
- If you look at the bottom of a motherboard you'll see a whole network of lines or electronic pathways that join the different components together.
- This network of wires or electronic pathways is called the '**Bus**'.

INTRODUCTION (cont'd)

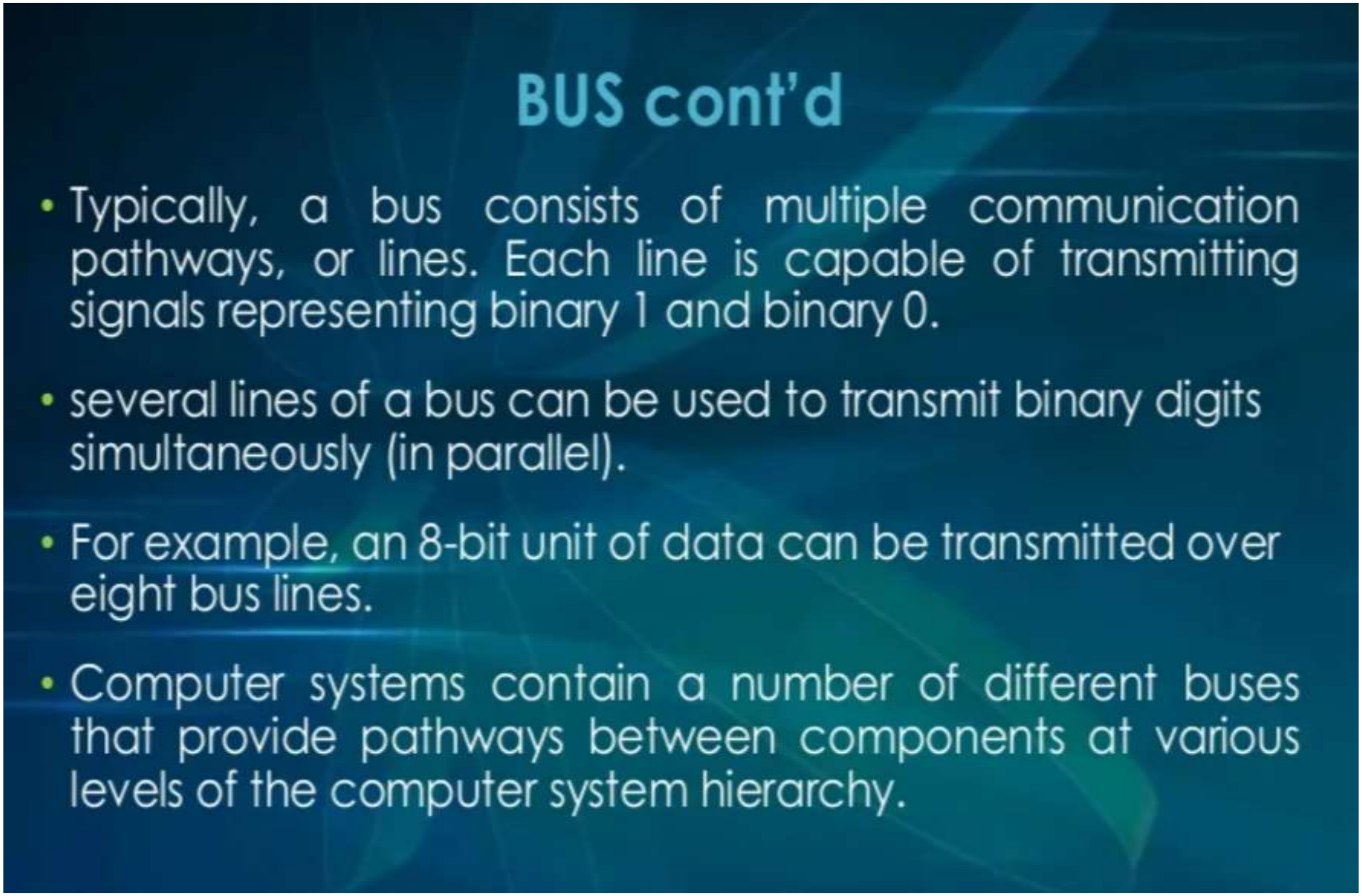
- Bottom of motherboard





BUS

- A bus is a communication pathway connecting two or more devices.
- A key characteristic of a bus is that it is a shared transmission medium.
- Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.



BUS cont'd

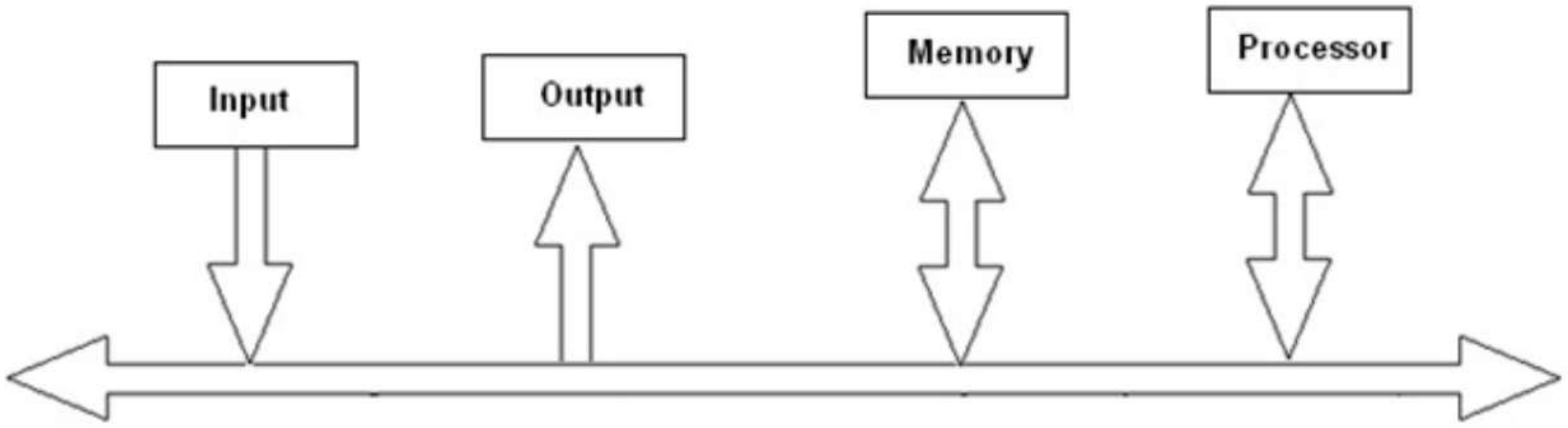
- Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0.
- several lines of a bus can be used to transmit binary digits simultaneously (in parallel).
- For example, an 8-bit unit of data can be transmitted over eight bus lines.
- Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy.

Computer Organization, Bus Structure

- Bus Structure
- A communication pathway connecting two or more devices
- When a word of data is transferred between units all its bits are transferred in parallel
- ie. The bits are transferred simultaneously over many lines, one bit per line .
- A group of lines that serves as a connecting path for several devices is called a bus
- A group of lines connected to different devices is called bus .
- ▪ Bus can carry data and control signals
- ▪ Single bus – simple way to interconnect functional units .

Computer Organization, Bus Structure

- Single Bus Structure



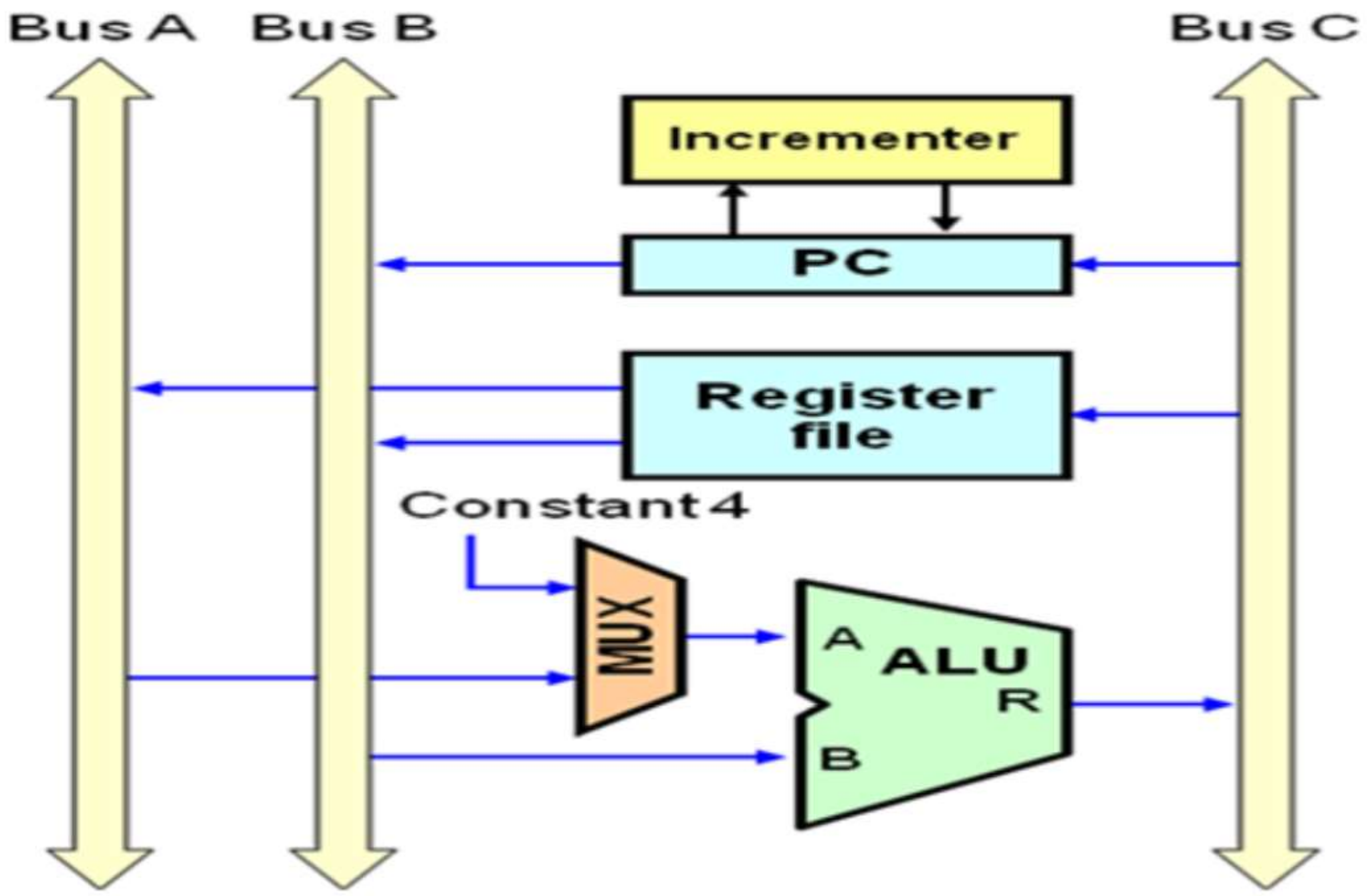
Computer Organization, Bus Structure

- All units are connected to this bus
- Bus can used only for one transfer at a time
- So only 2 units can actively use the bus at any given time.
- Only one device at a time can successfully transmit .
- Single bus – low cost and It is flexible for attaching diff devices
- By using multiple bus simultaneously can transfer more than one data at a time
- This leads to increase the performance of the system.
- So the bus can carry several data at a time in parallel such as
- Power
- Instructions
- Data
- Addresses
- Commands

Computer Organization, Bus Structure

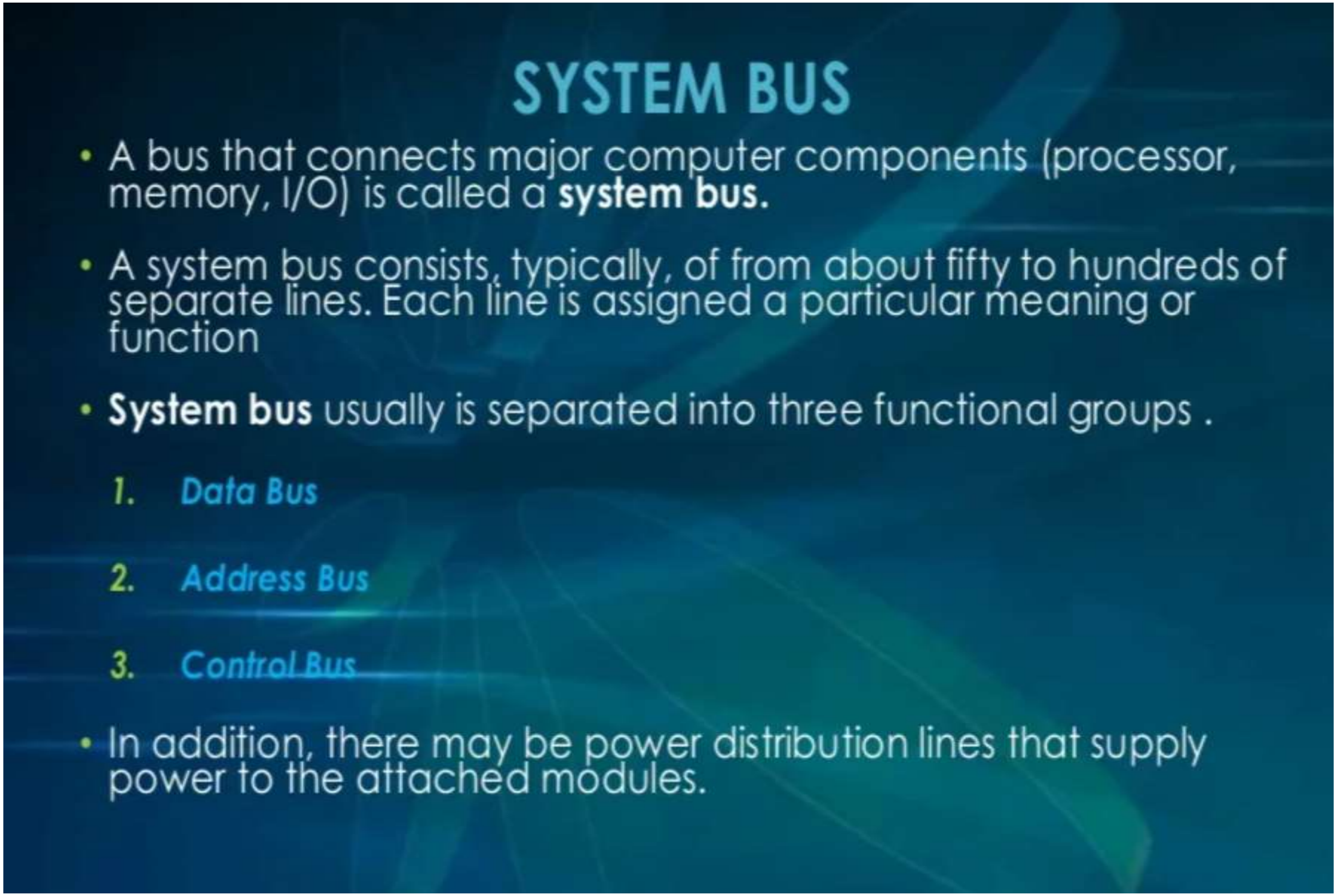
- Multiple-Bus Organization
- The following Figure shows a three-bus structure.
- All registers are combined into a single block called register file with three ports: 2 outputs allowing 2 registers to be accessed simultaneously.
- Buses A and B are used to transfer source operands to the A and B inputs of ALU, and result transferred to destination over bus C.

Computer Organization, Bus Structure



Computer Organization, Bus Structure

- Diff units in a system having diff speed
- Keyboard
- Printer are relatively slow
- some units are very fast all these devices communicate with each other over the same bus
- in order to communicate all types of units smoothly include buffer registers with the devices to hold the data during transfer.
- ▪ eg transfer of data between processor and printer
- Processor sent data over the bus to printer buffer
- Once the buffer is loaded the printer can start printing With out the interaction of the processor
- The bus and processor are now free to do another work.



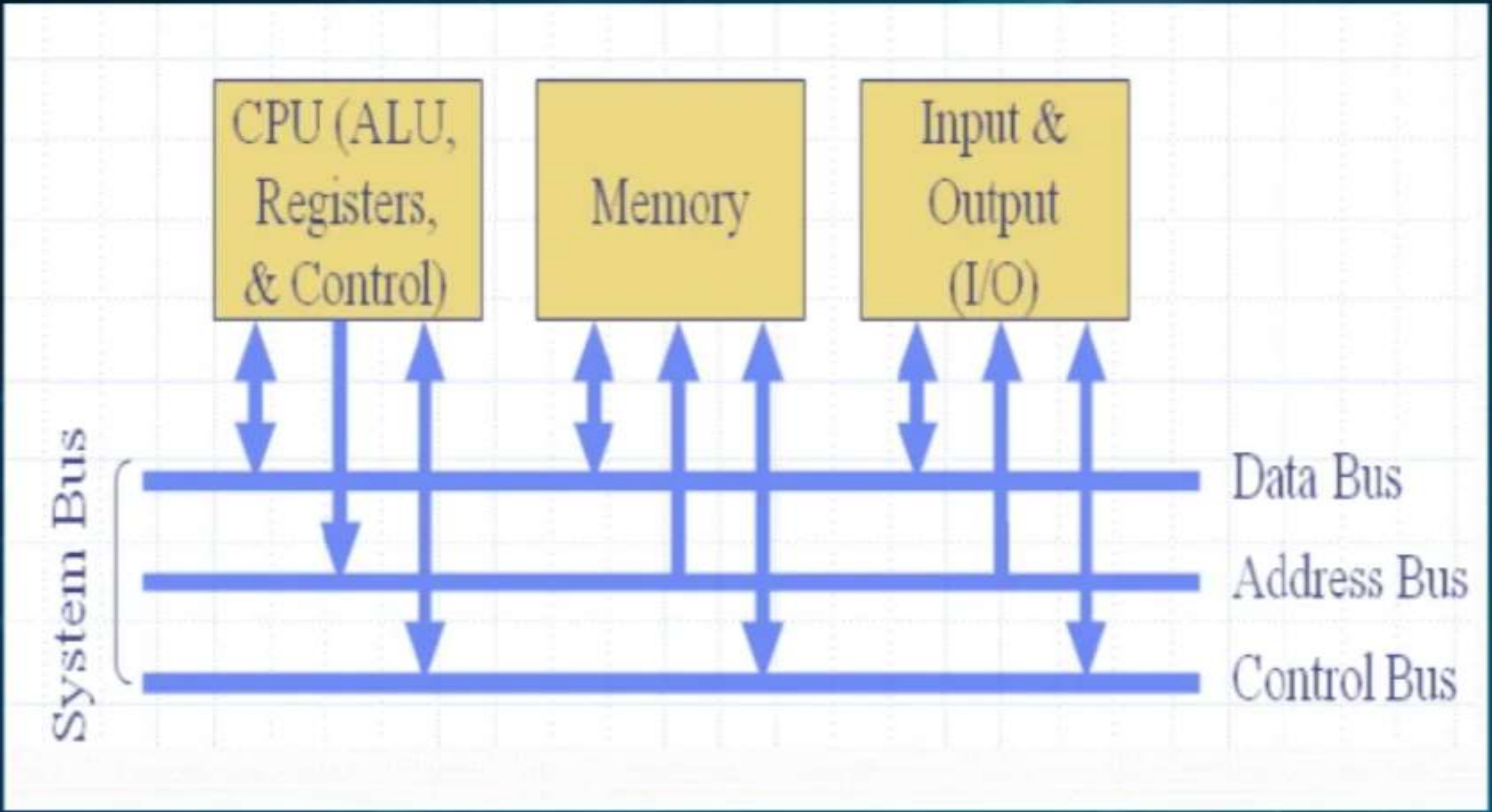
SYSTEM BUS

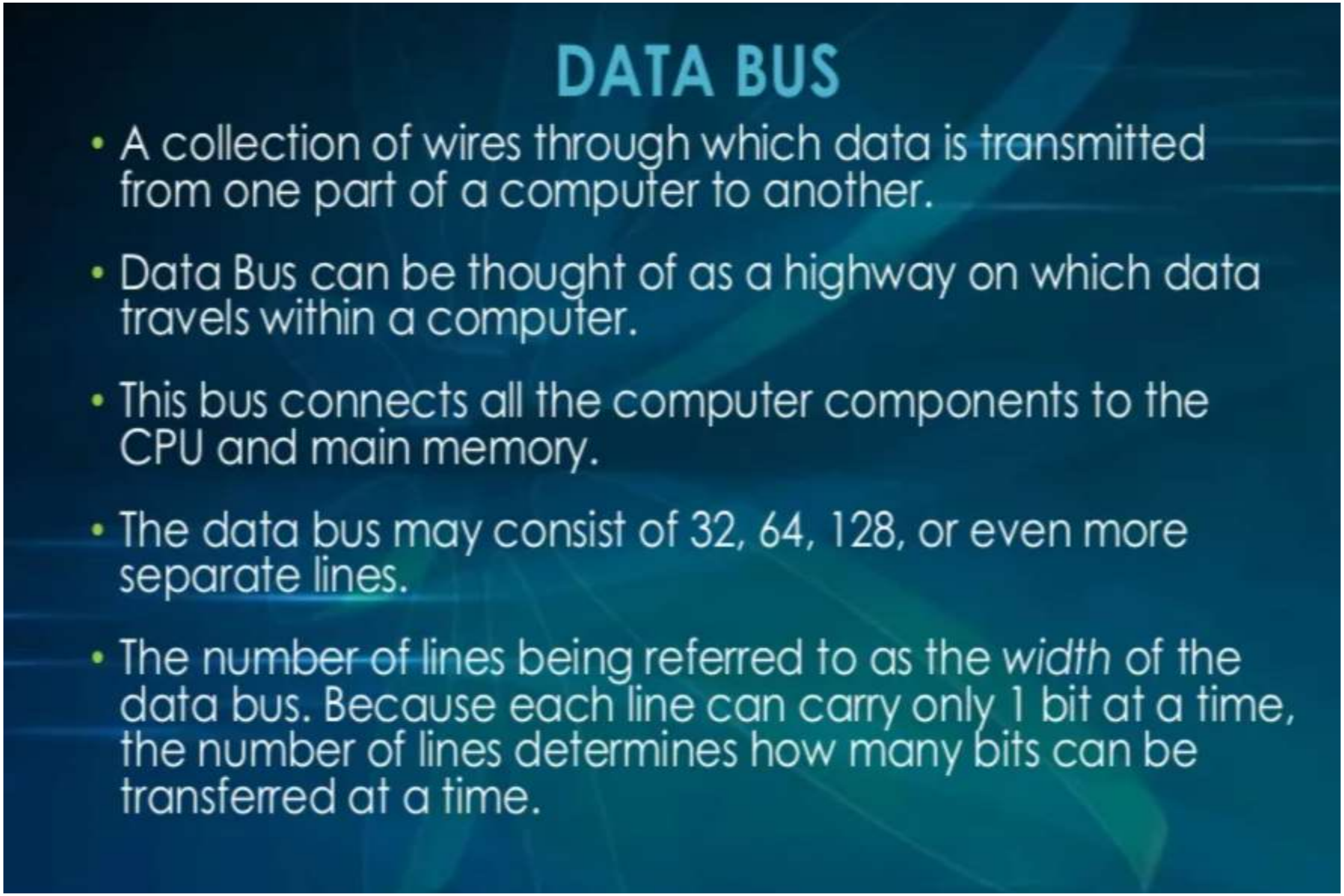
- A bus that connects major computer components (processor, memory, I/O) is called a **system bus.**
- A system bus consists, typically, of from about fifty to hundreds of separate lines. Each line is assigned a particular meaning or function
- **System bus** usually is separated into three functional groups .

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

- In addition, there may be power distribution lines that supply power to the attached modules.

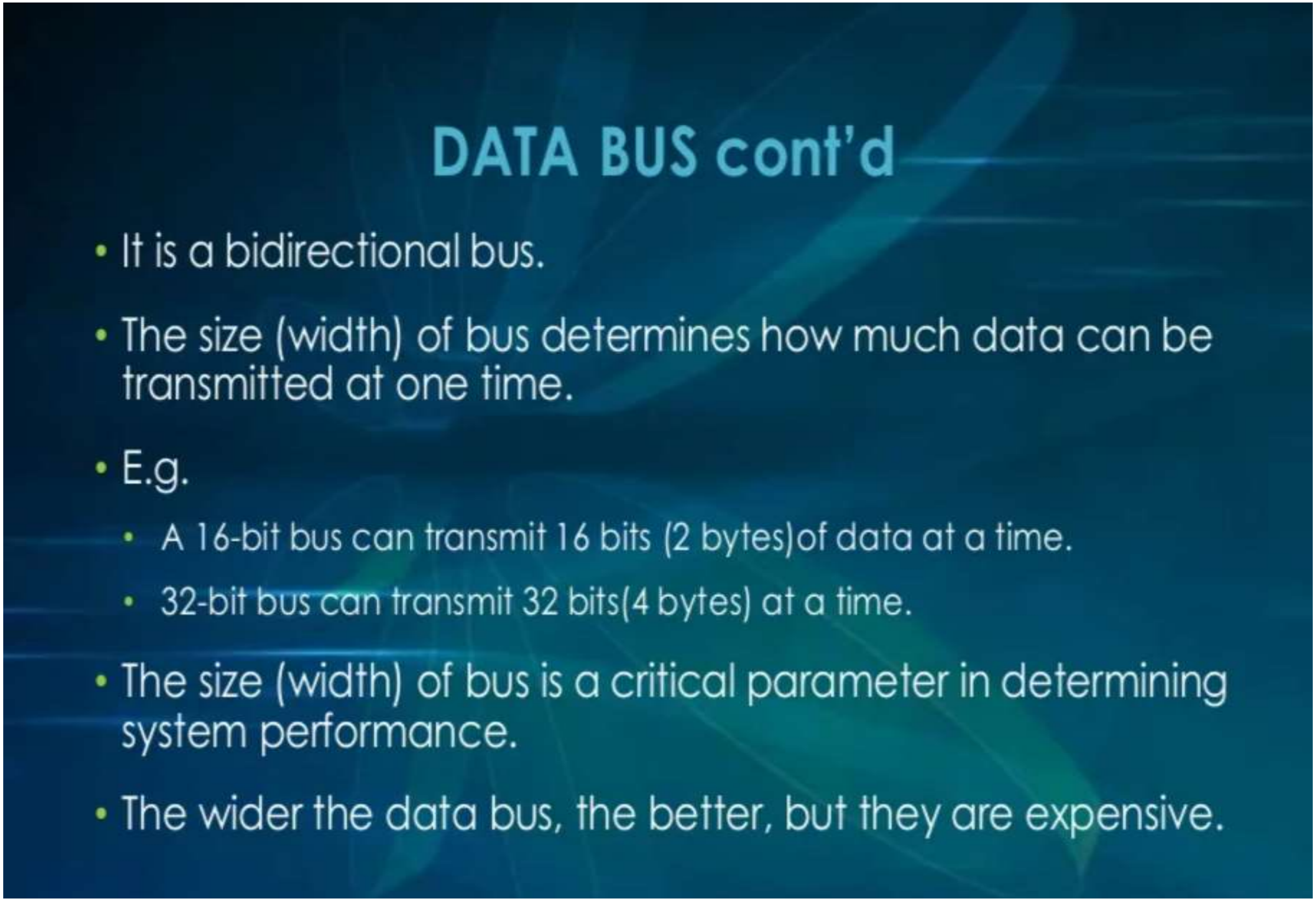
SYSTEM BUS MODEL





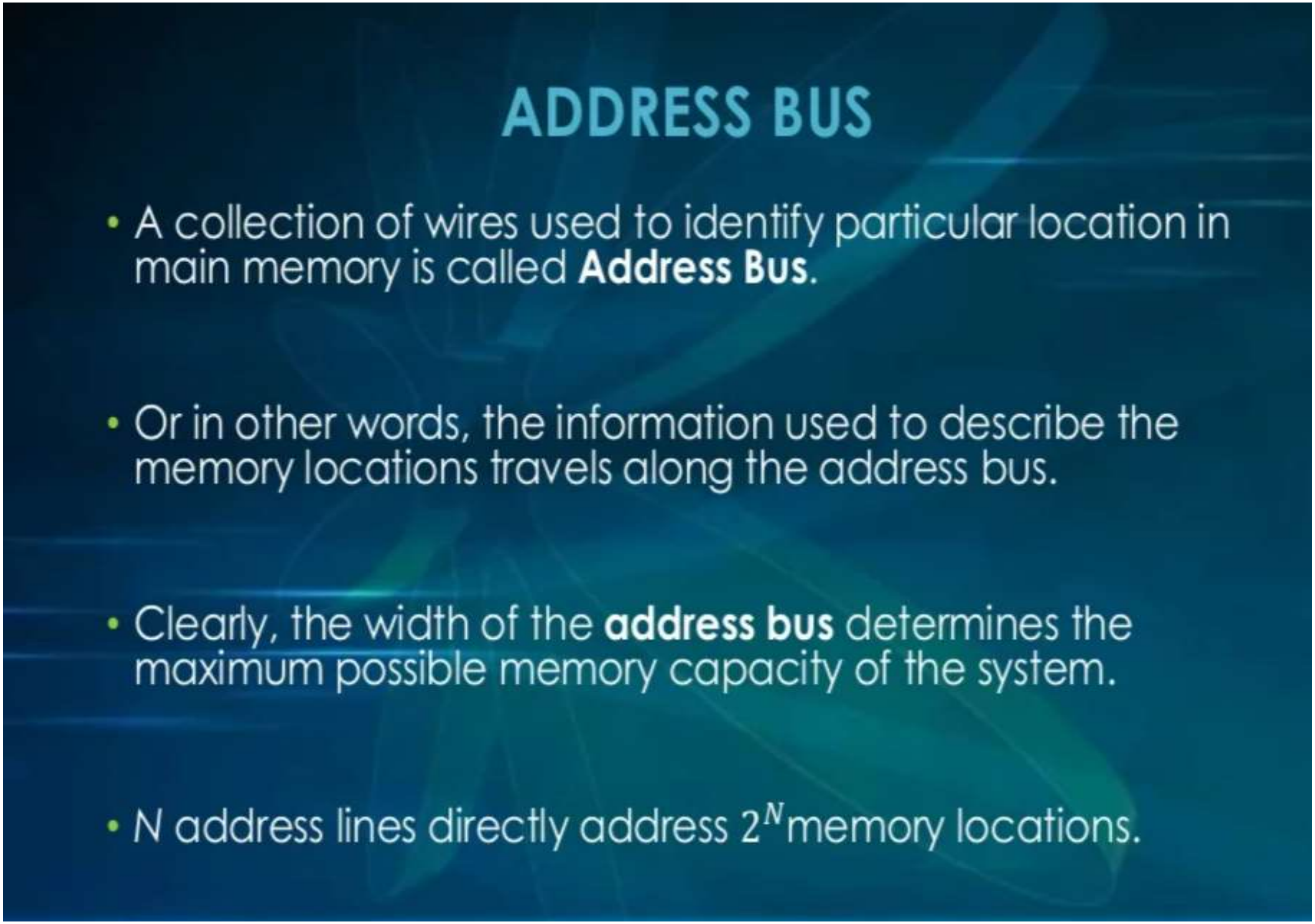
DATA BUS

- A collection of wires through which data is transmitted from one part of a computer to another.
- Data Bus can be thought of as a highway on which data travels within a computer.
- This bus connects all the computer components to the CPU and main memory.
- The data bus may consist of 32, 64, 128, or even more separate lines.
- The number of lines being referred to as the *width* of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time.



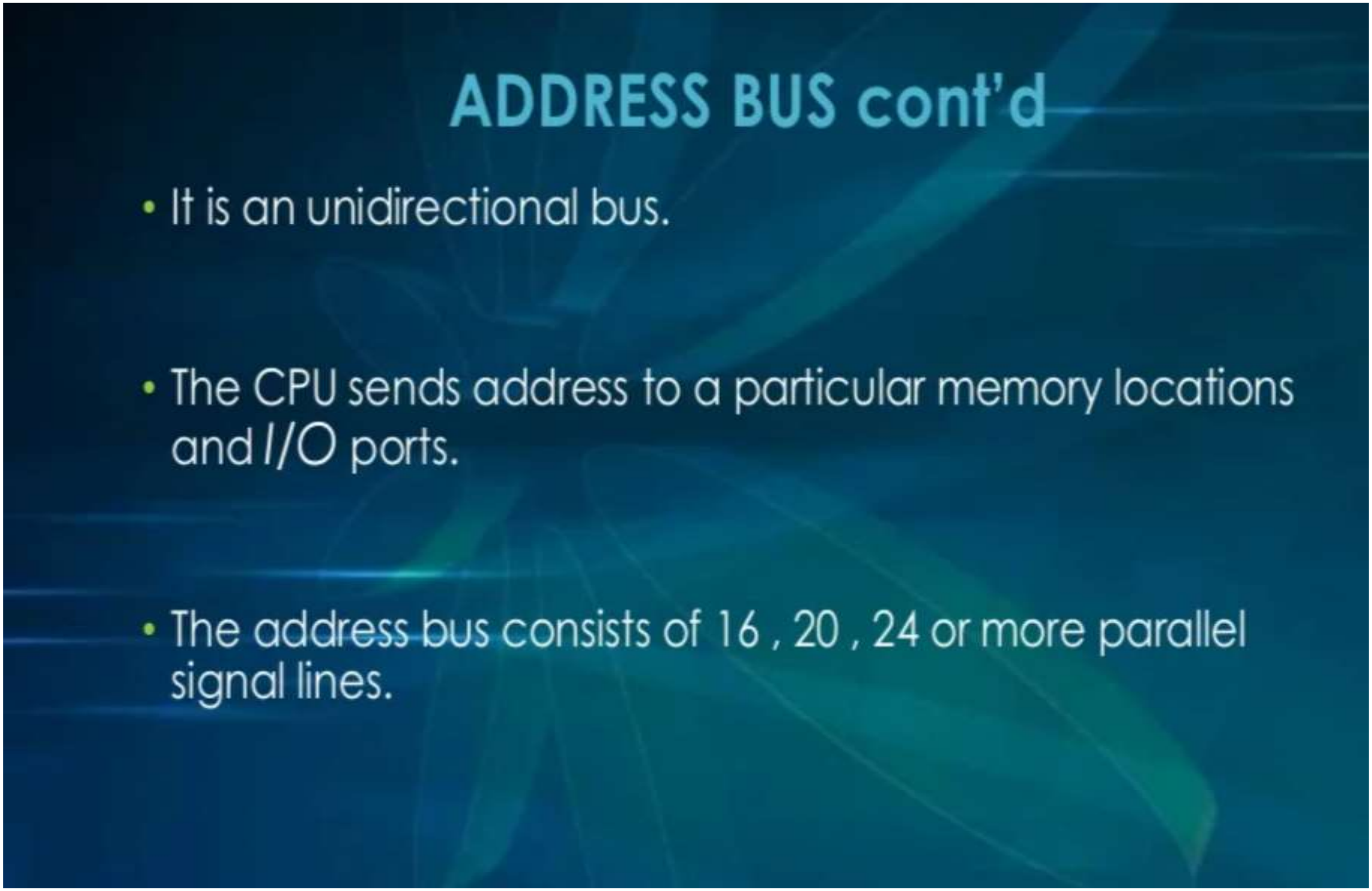
DATA BUS cont'd

- It is a bidirectional bus.
- The size (width) of bus determines how much data can be transmitted at one time.
- E.g.
  - A 16-bit bus can transmit 16 bits (2 bytes)of data at a time.
  - 32-bit bus can transmit 32 bits(4 bytes) at a time.
- The size (width) of bus is a critical parameter in determining system performance.
- The wider the data bus, the better, but they are expensive.



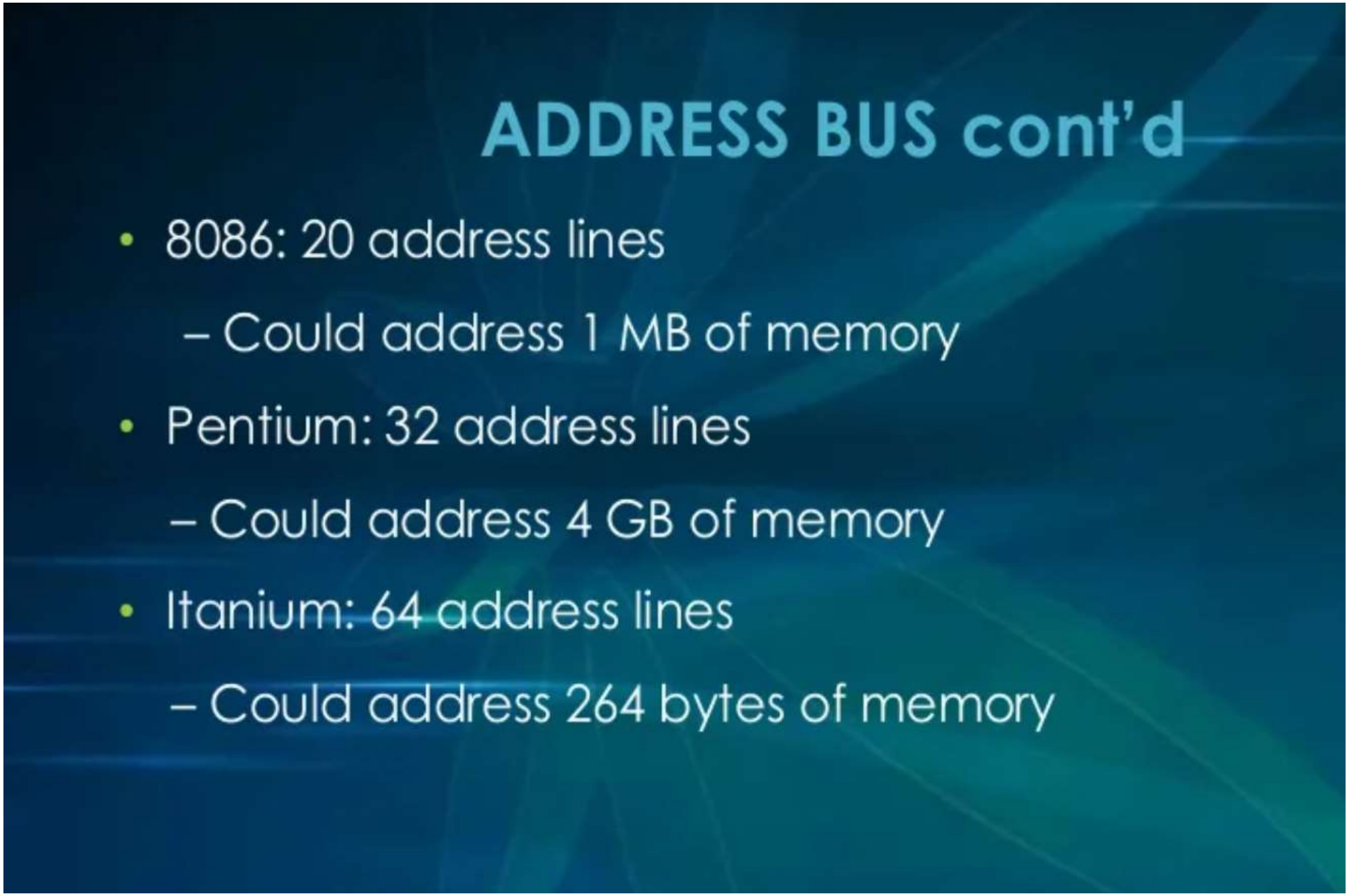
ADDRESS BUS

- A collection of wires used to identify particular location in main memory is called **Address Bus**.
- Or in other words, the information used to describe the memory locations travels along the address bus.
- Clearly, the width of the **address bus** determines the maximum possible memory capacity of the system.
- *N* address lines directly address 2^N memory locations.



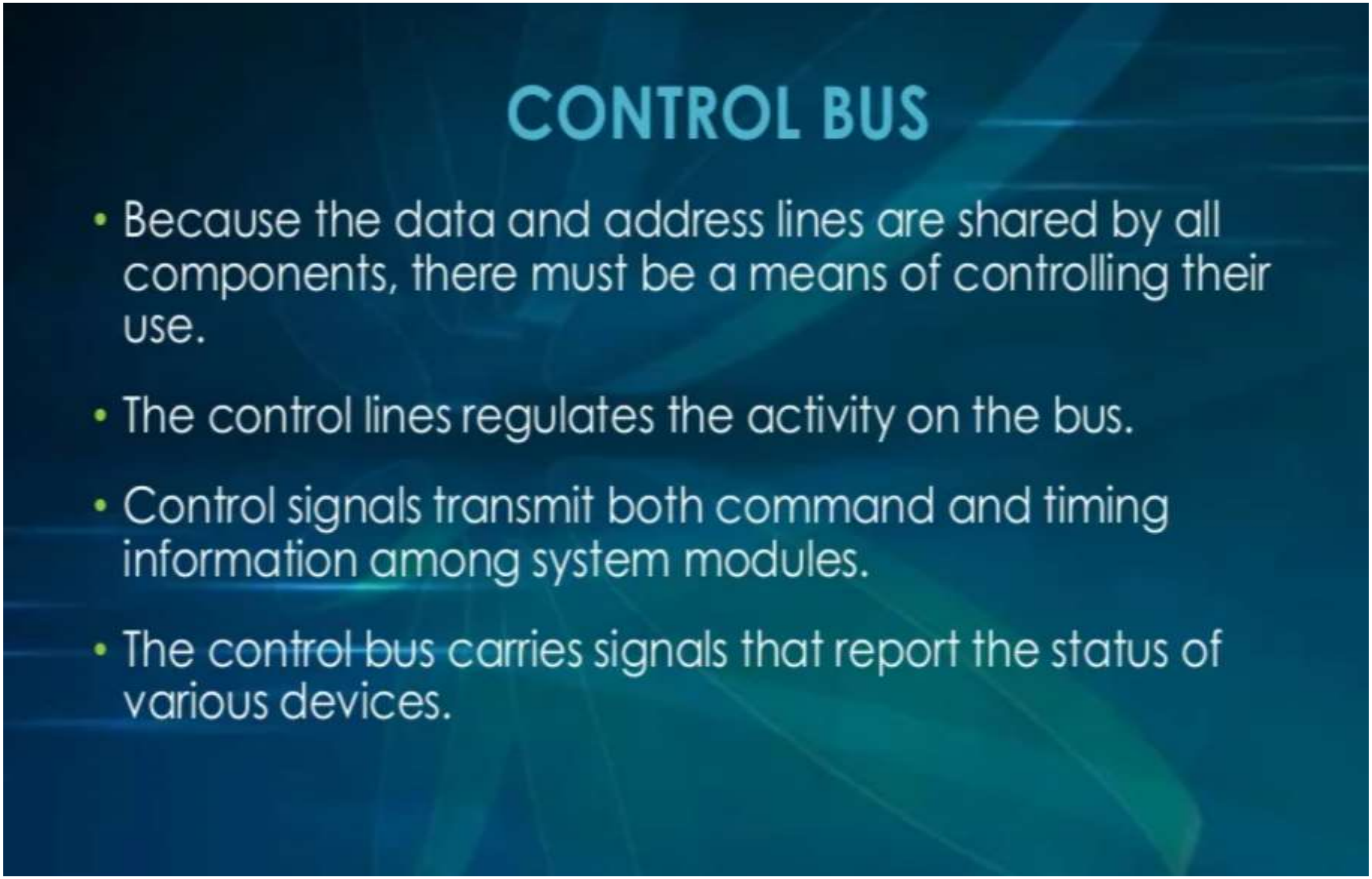
ADDRESS BUS cont'd

- It is an unidirectional bus.
- The CPU sends address to a particular memory locations and I/O ports.
- The address bus consists of 16 , 20 , 24 or more parallel signal lines.



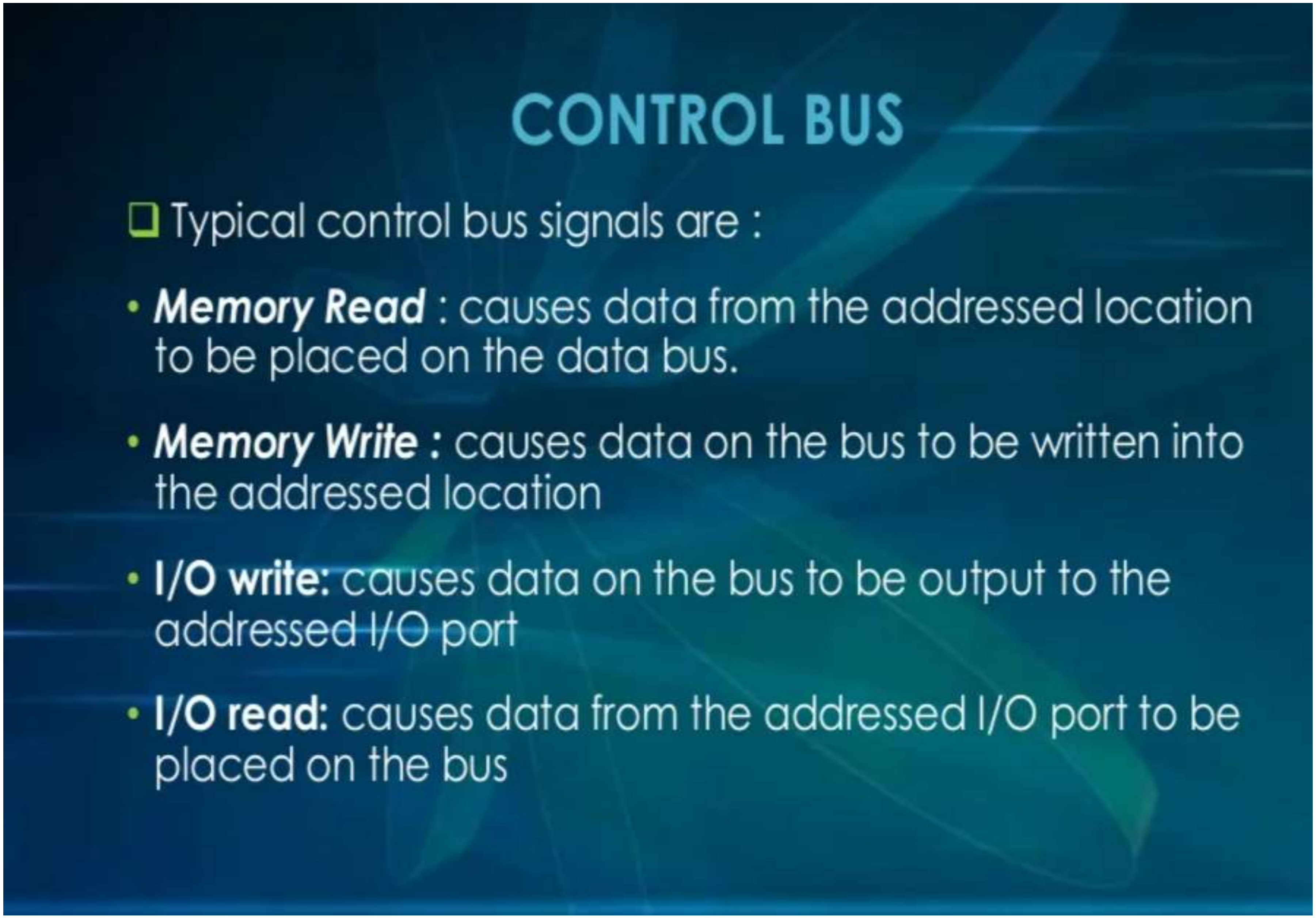
ADDRESS BUS cont'd

- 8086: 20 address lines
  - Could address 1 MB of memory
- Pentium: 32 address lines
  - Could address 4 GB of memory
- Itanium: 64 address lines
  - Could address 264 bytes of memory



CONTROL BUS

- Because the data and address lines are shared by all components, there must be a means of controlling their use.
- The control lines regulates the activity on the bus.
- Control signals transmit both command and timing information among system modules.
- The control bus carries signals that report the status of various devices.



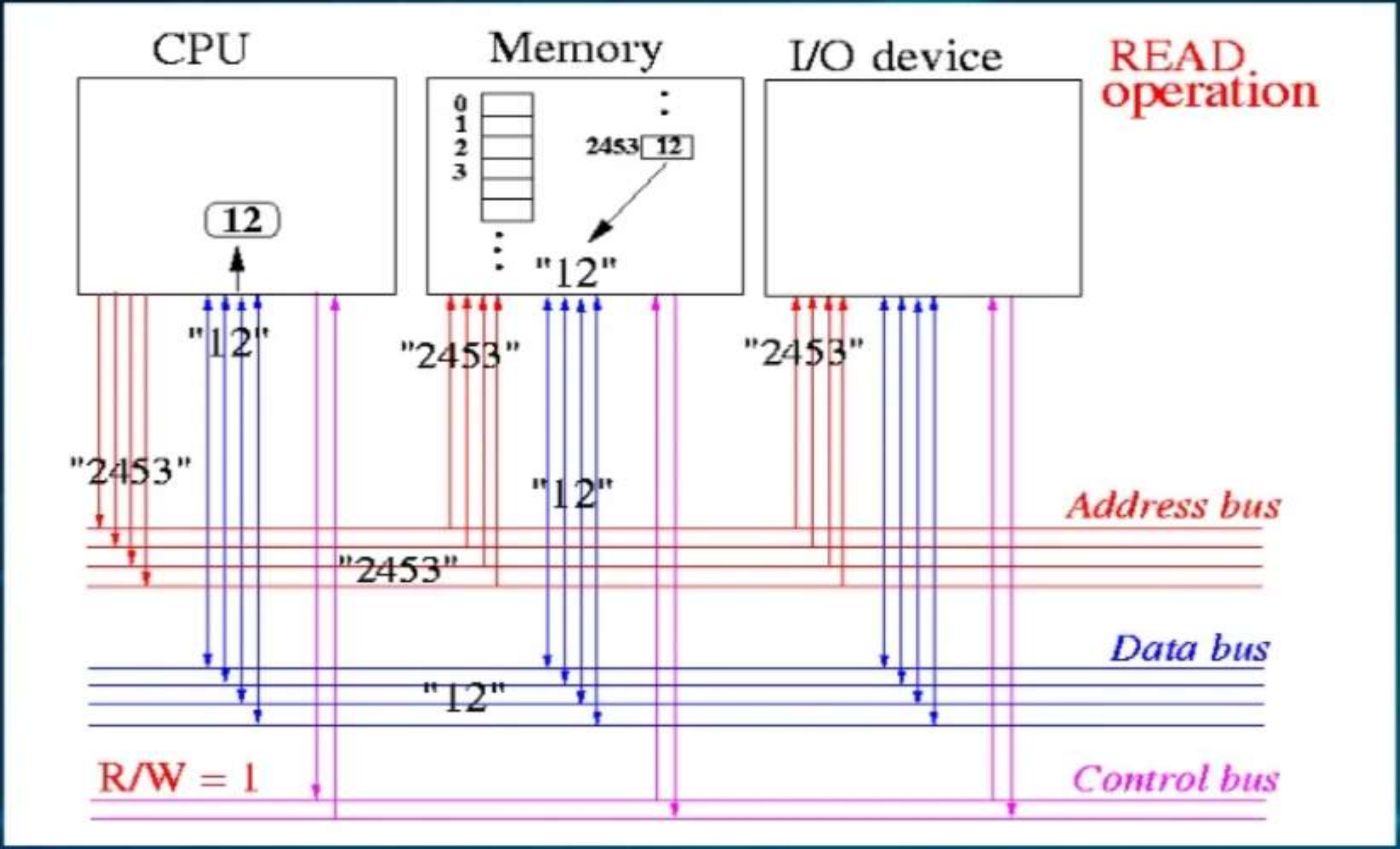
CONTROL BUS

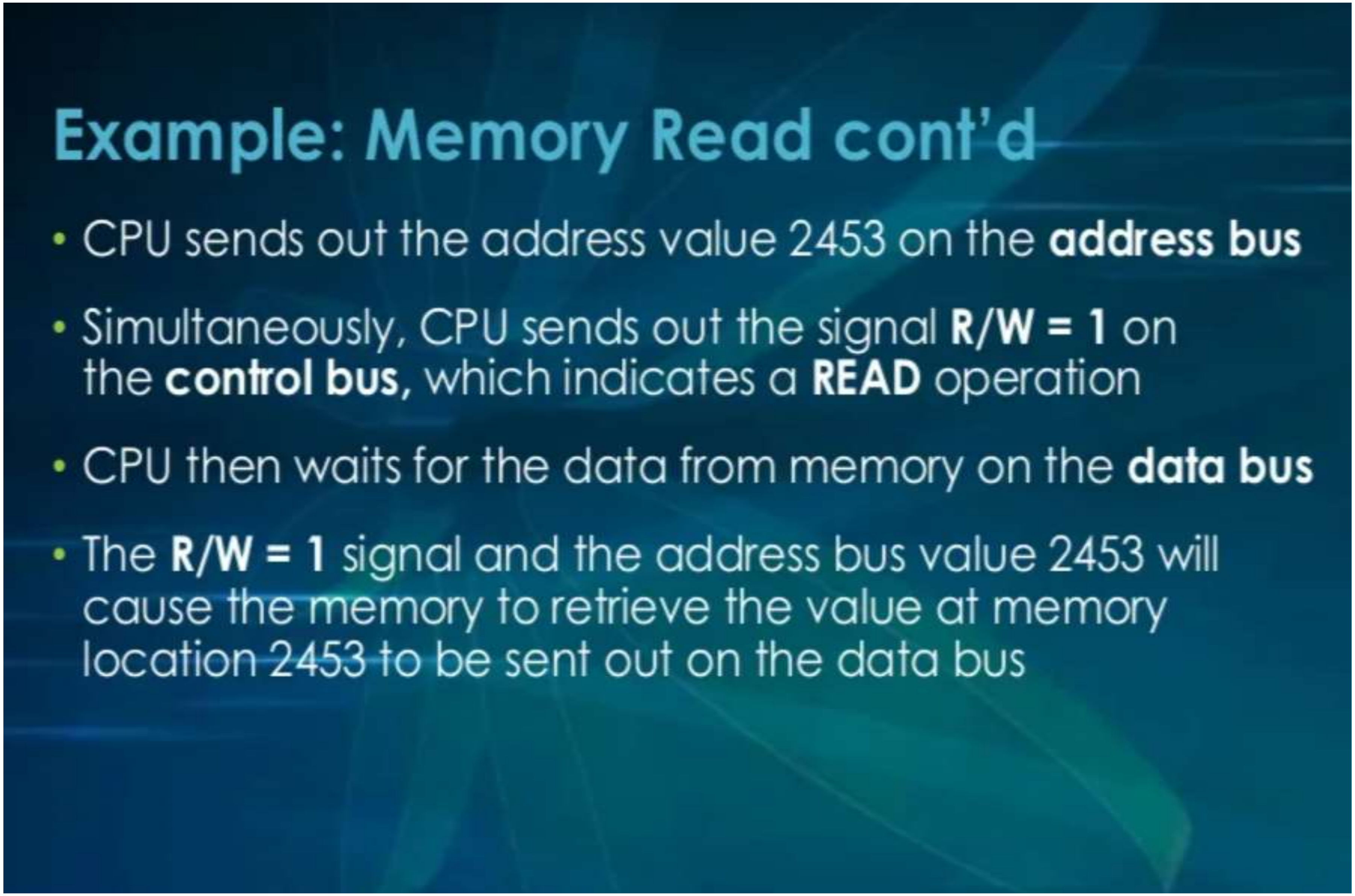
❑ Typical control bus signals are :

- ***Memory Read*** : causes data from the addressed location to be placed on the data bus.
- ***Memory Write* :** causes data on the bus to be written into the addressed location
- **I/O write:** causes data on the bus to be output to the addressed I/O port
- **I/O read:** causes data from the addressed I/O port to be placed on the bus

Example : Memory Read

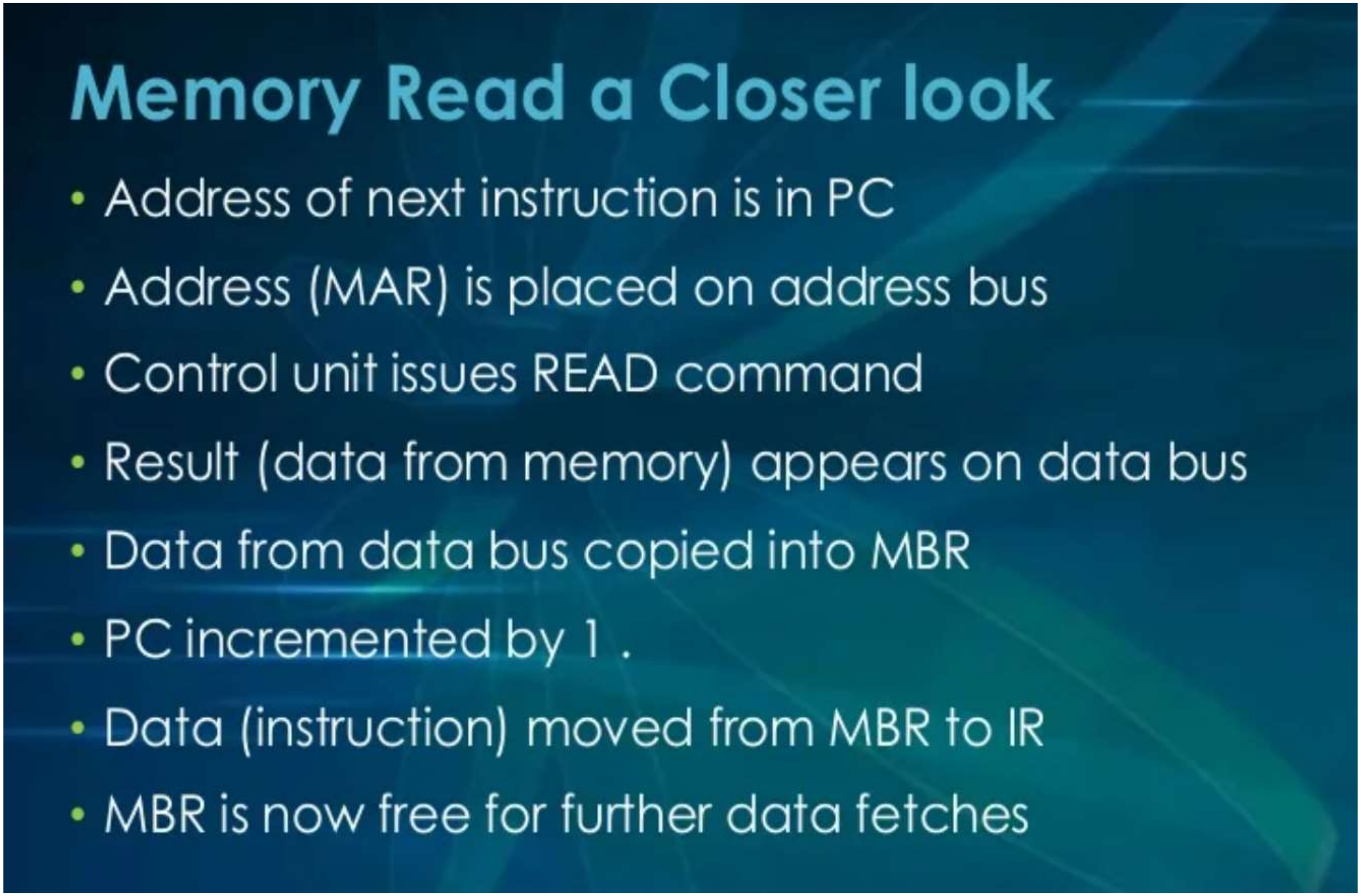
- The following figure shows how the CPU reads the value 12 from the memory location 2453:





Example: Memory Read cont'd

- CPU sends out the address value 2453 on the **address bus**
- Simultaneously, CPU sends out the signal **R/W = 1** on the **control bus,** which indicates a **READ** operation
- CPU then waits for the data from memory on the **data bus**
- The **R/W = 1** signal and the address bus value 2453 will cause the memory to retrieve the value at memory location 2453 to be sent out on the data bus

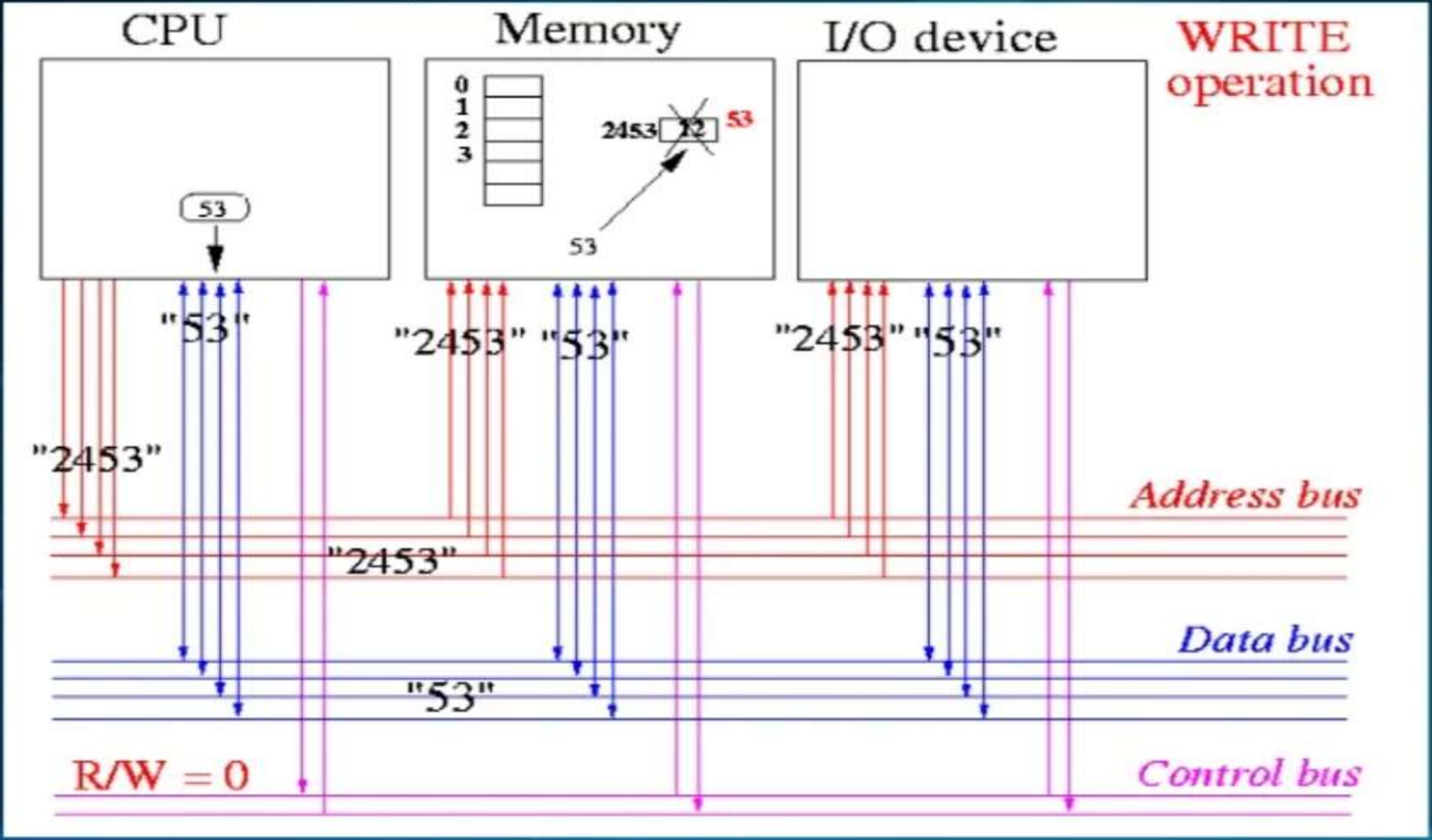


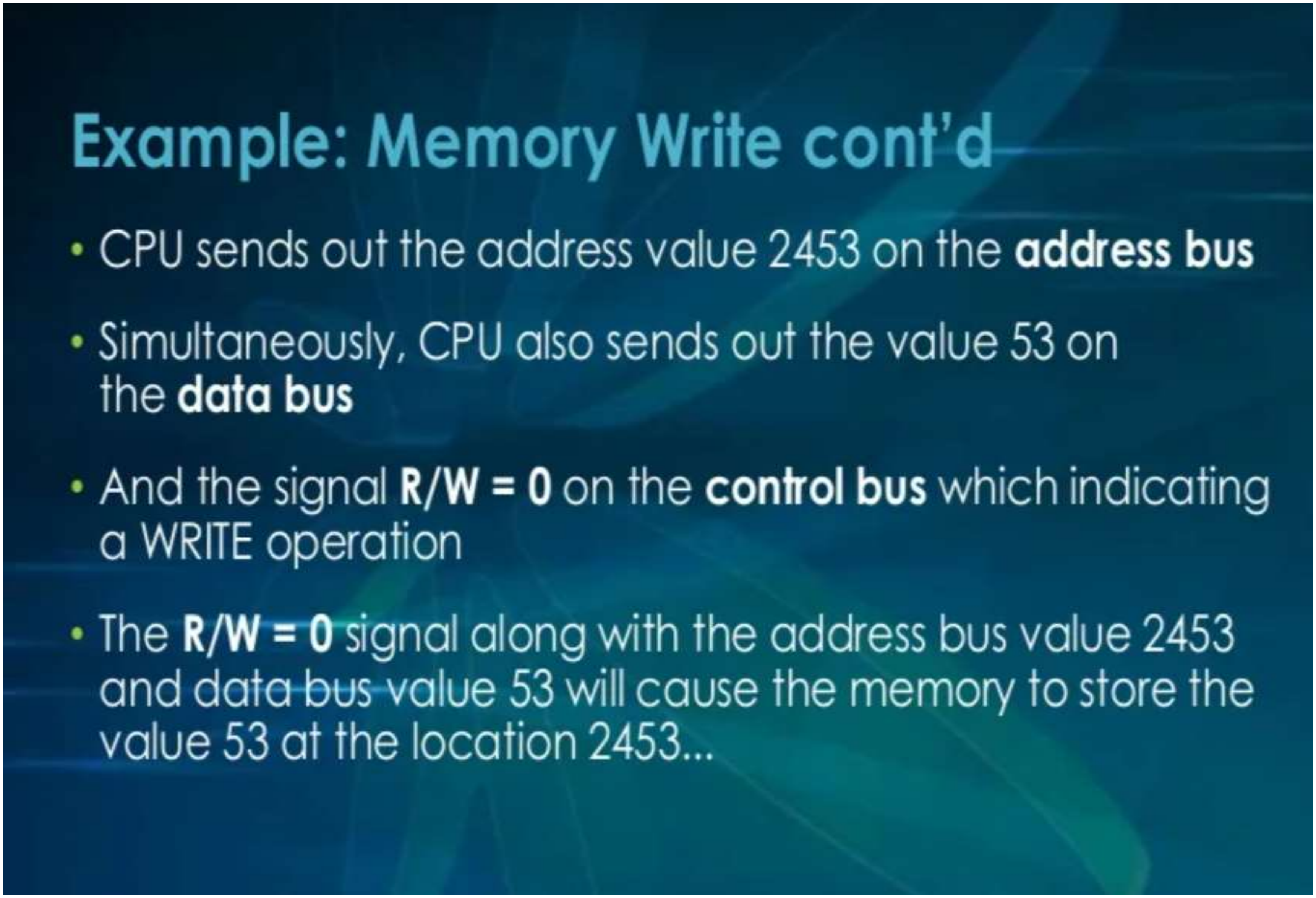
Memory Read a Closer look

- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 .
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches

Example: Memory Write

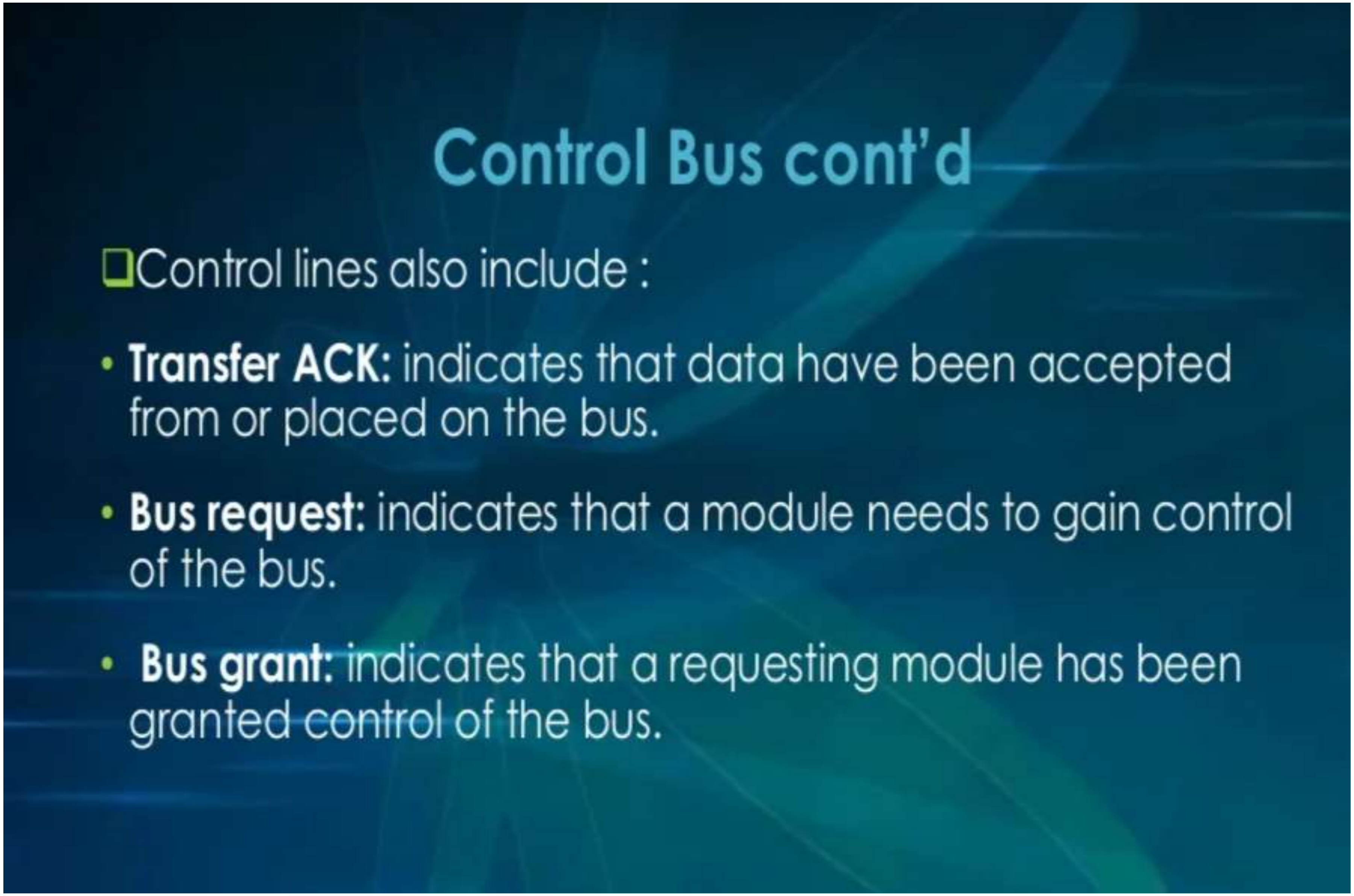
- The following figure shows how the CPU writes the value 53 from the memory location 2453:





Example: Memory Write cont'd

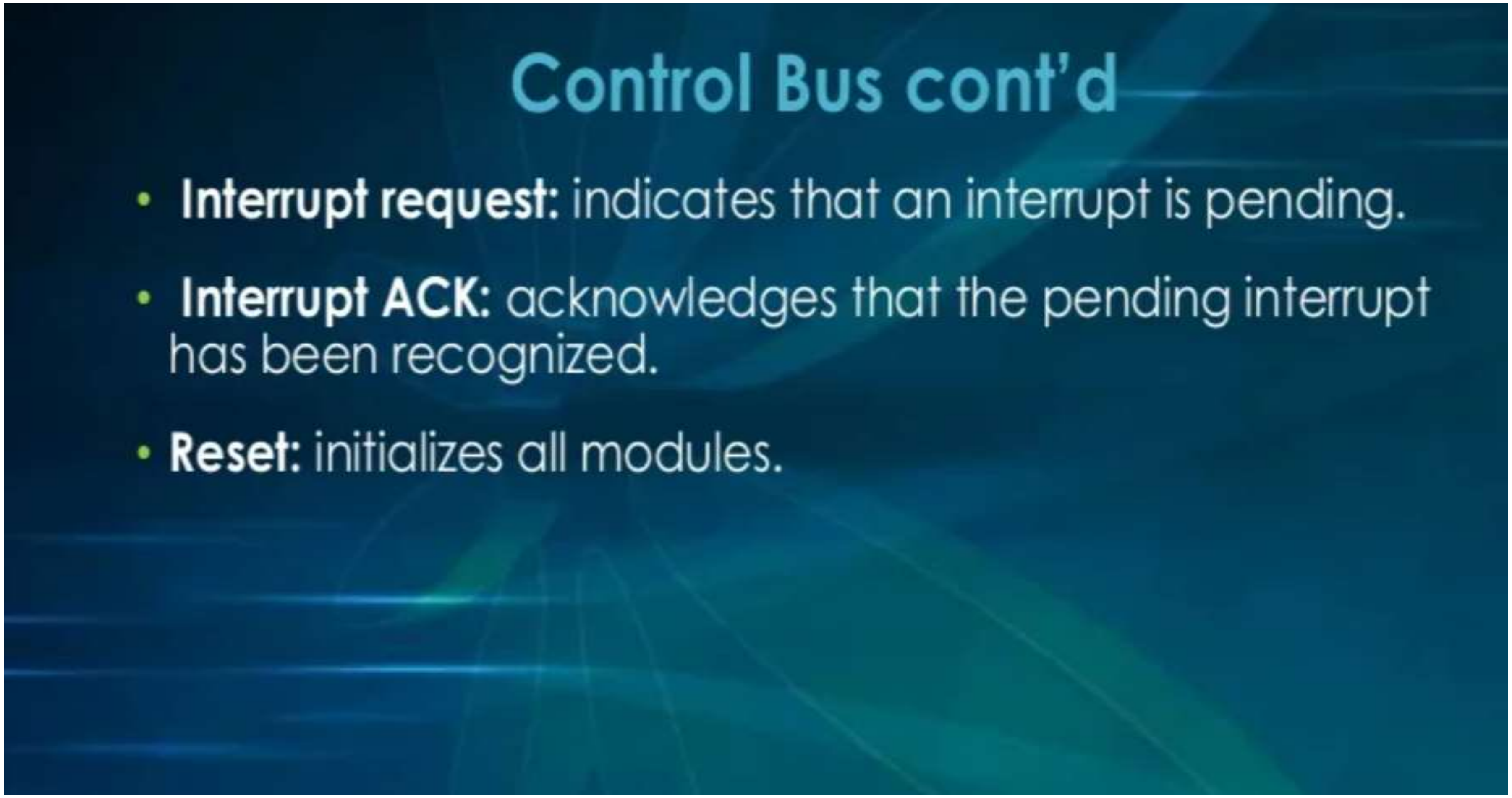
- CPU sends out the address value 2453 on the **address bus**
- Simultaneously, CPU also sends out the value 53 on the **data bus**
- And the signal **R/W = 0** on the **control bus** which indicating a WRITE operation
- The **R/W = 0** signal along with the address bus value 2453 and data bus value 53 will cause the memory to store the value 53 at the location 2453...



Control Bus cont'd

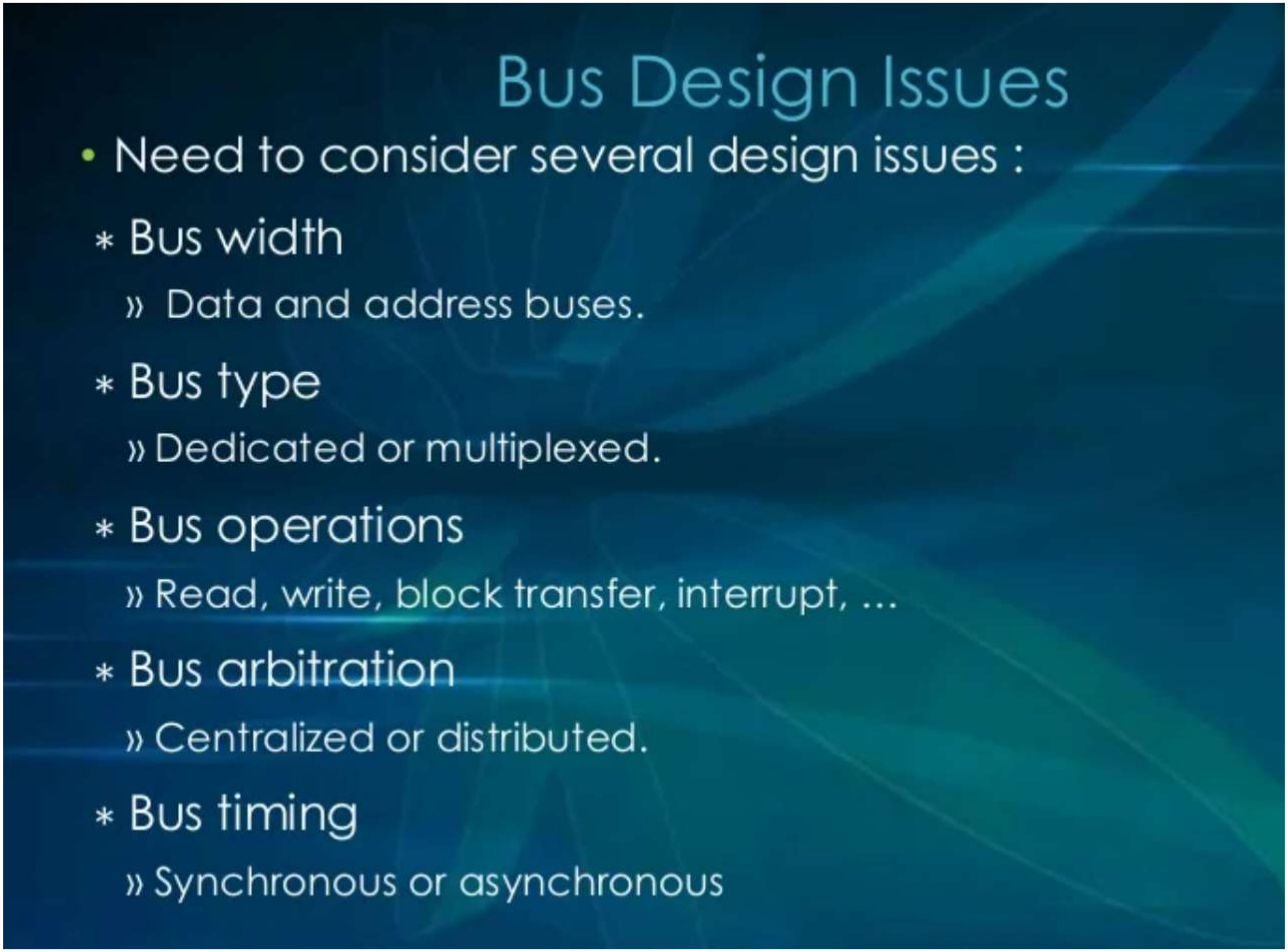
❑Control lines also include :

- **Transfer ACK:** indicates that data have been accepted from or placed on the bus.
- **Bus request:** indicates that a module needs to gain control of the bus.
- **Bus grant:** indicates that a requesting module has been granted control of the bus.



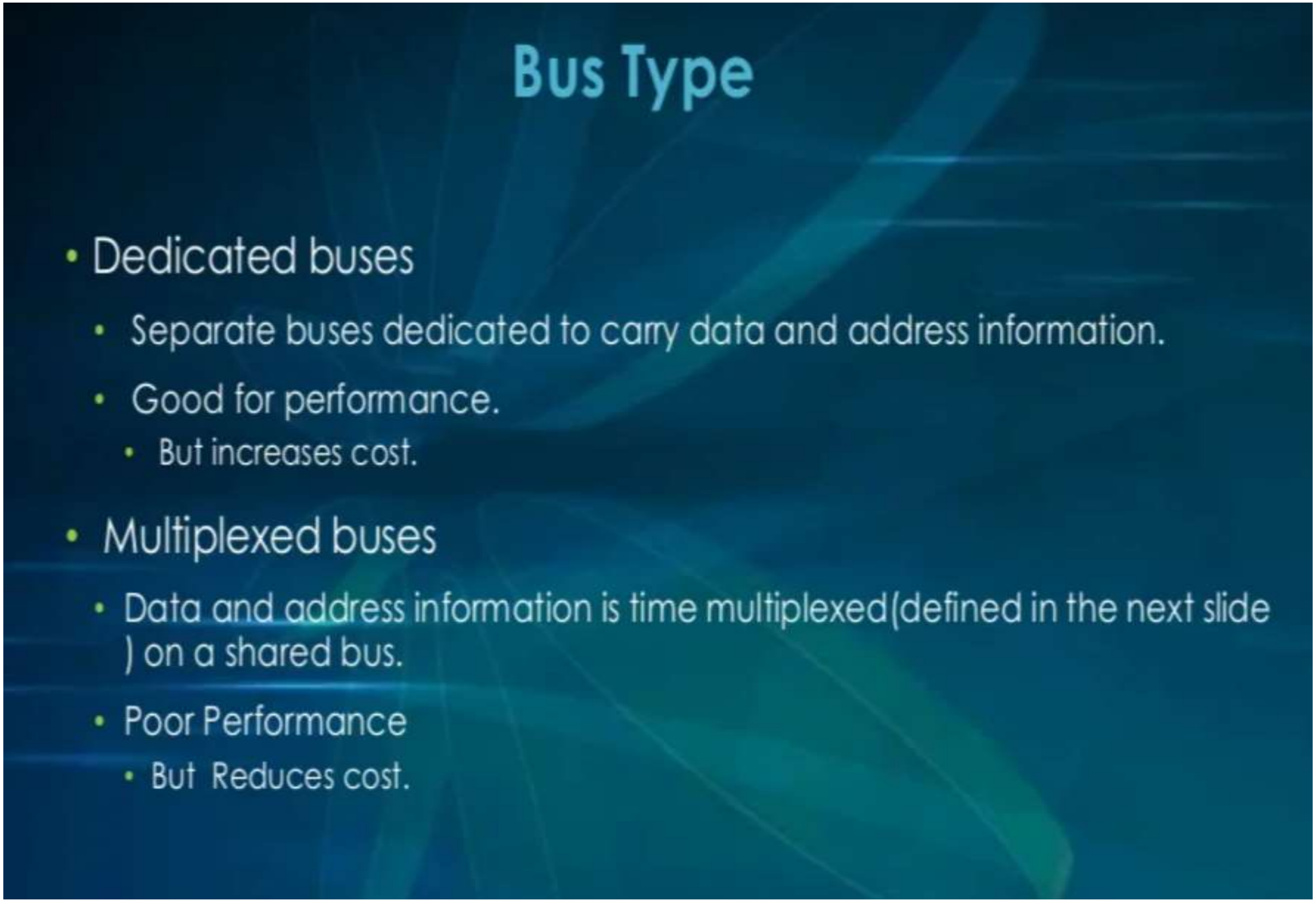
Control Bus cont'd

- **Interrupt request:** indicates that an interrupt is pending.
- **Interrupt ACK:** acknowledges that the pending interrupt has been recognized.
- **Reset:** initializes all modules.



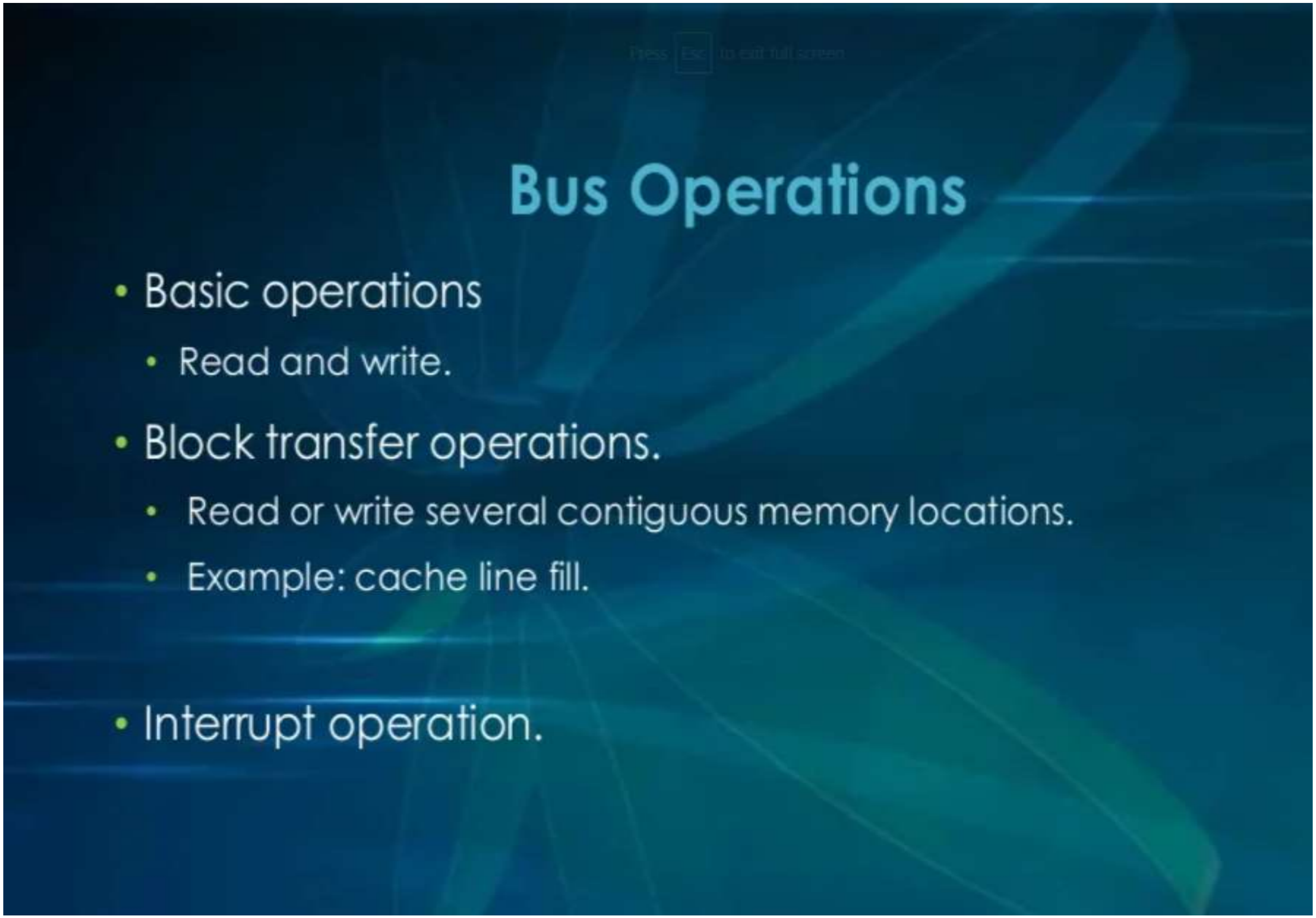
Bus Design Issues

- Need to consider several design issues :
  - ∗ Bus width
    - » Data and address buses.
  - ∗ Bus type
    - » Dedicated or multiplexed.
  - ∗ Bus operations
    - » Read, write, block transfer, interrupt, …
  - ∗ Bus arbitration
    - » Centralized or distributed.
  - ∗ Bus timing
    - » Synchronous or asynchronous



Bus Type

- Dedicated buses
  - Separate buses dedicated to carry data and address information.
  - Good for performance.
    - But increases cost.
- Multiplexed buses
  - Data and address information is time multiplexed(defined in the next slide) on a shared bus.
  - Poor Performance
    - But Reduces cost.

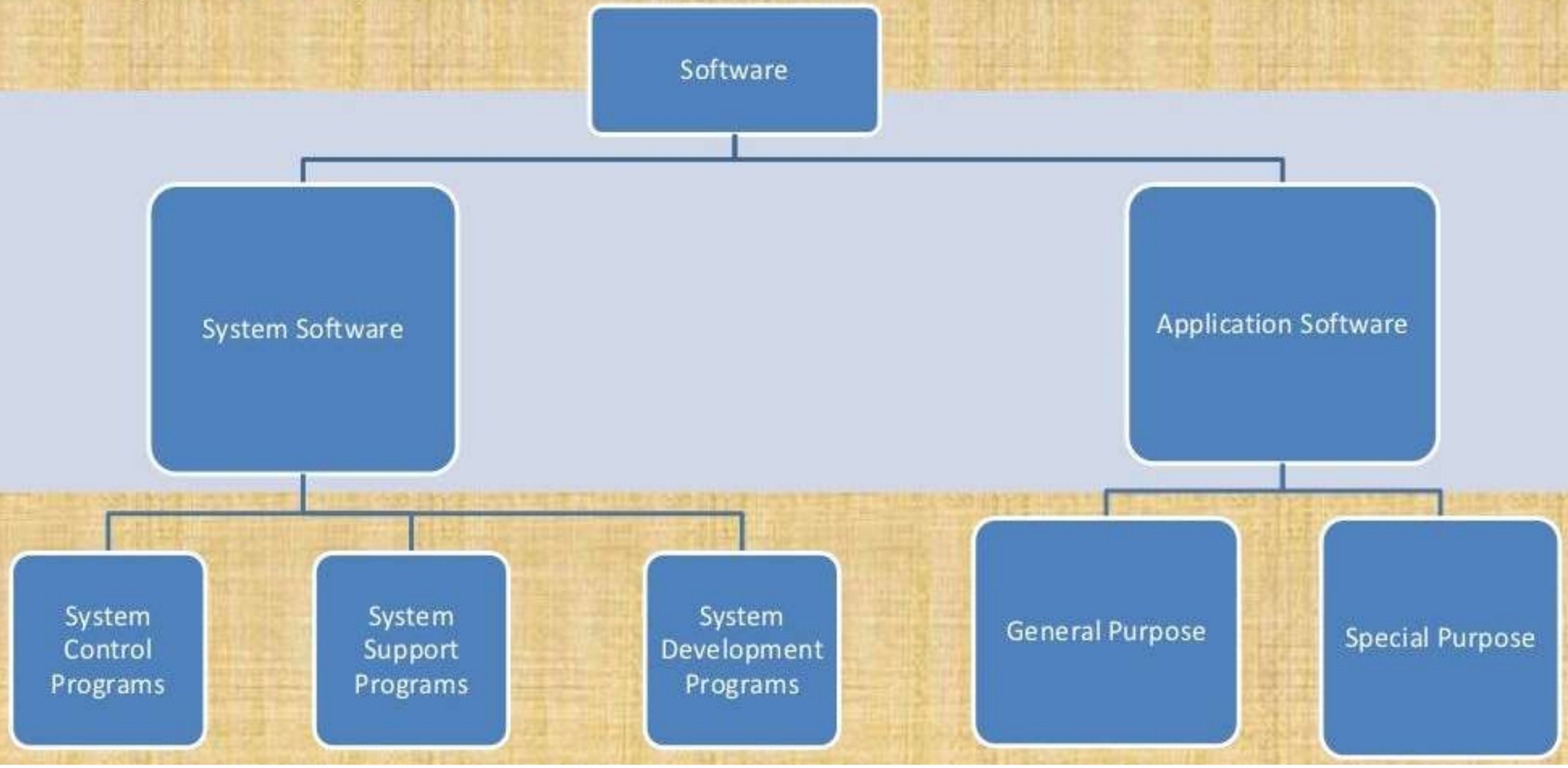


Bus Operations

- Basic operations
  - Read and write.
- Block transfer operations.
  - Read or write several contiguous memory locations.
  - Example: cache line fill.
- Interrupt operation.

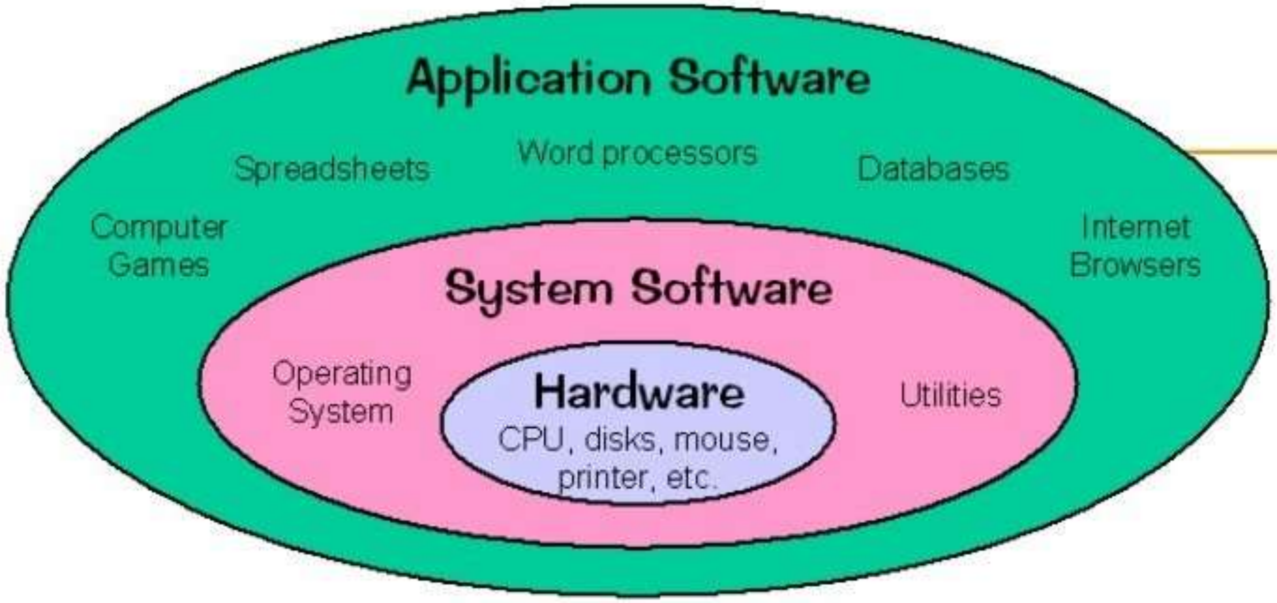
SOFTWARE

- Software is a set of computer programs which are designed and developed to perform specific task desired by the user or by the computer itself.



Types of Software

- System Software
- Application Software



Software ?

- **Software**

Computer Instructions or data, anything that can be stored electronically is **Software**.

- **Examples:-**
- Ms word, excel, power point, spread sheets, library management system etc.

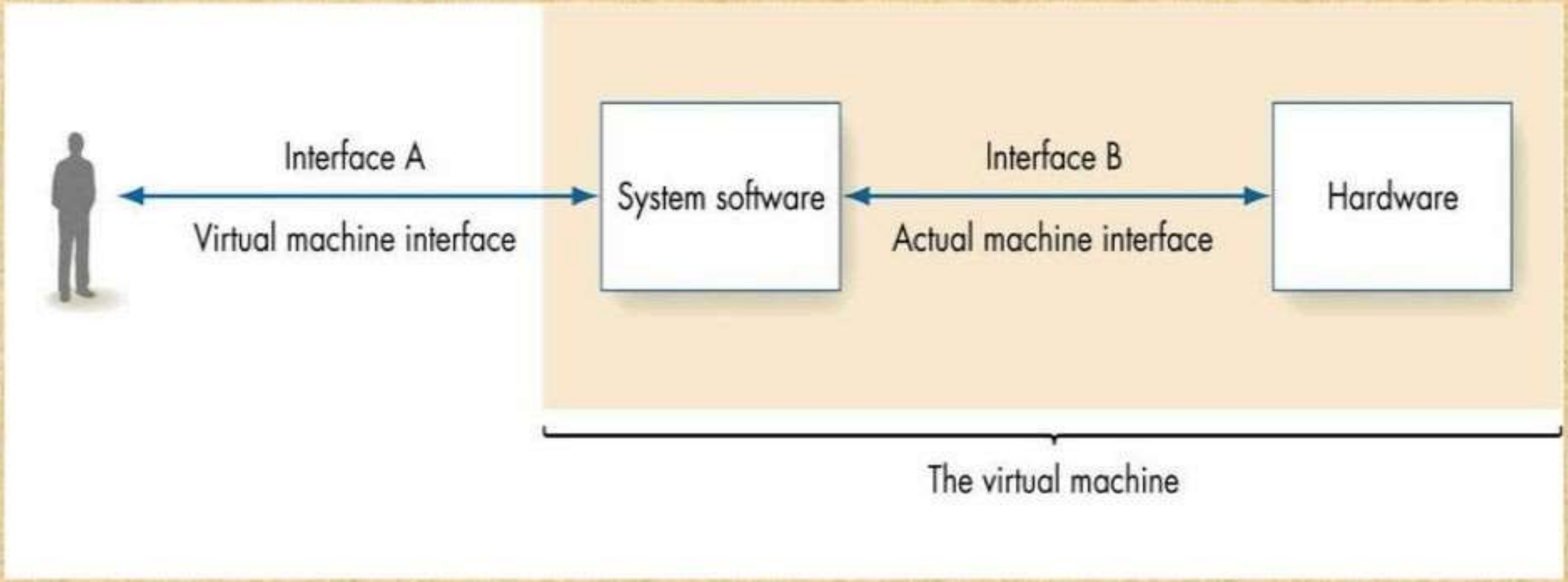
System Software

- The system software is collection of programs designed to operate, control and extend the processing capabilities of the computer itself.

- These are generally prepared by computer manufacturers.

- These software perform a variety of functions like file editing, storage management, resource accounting, I/O management, etc.

Role of System Software



Types of System Software

1. System Control Programs :

   They control the execution of programs, manage the storage and processing resources of the computer and perform other management and monitoring functions. e.g., OS

2. System Support Programs :

   They provide routine service functions to other computer programs and computer users. e.g., Utility Programs

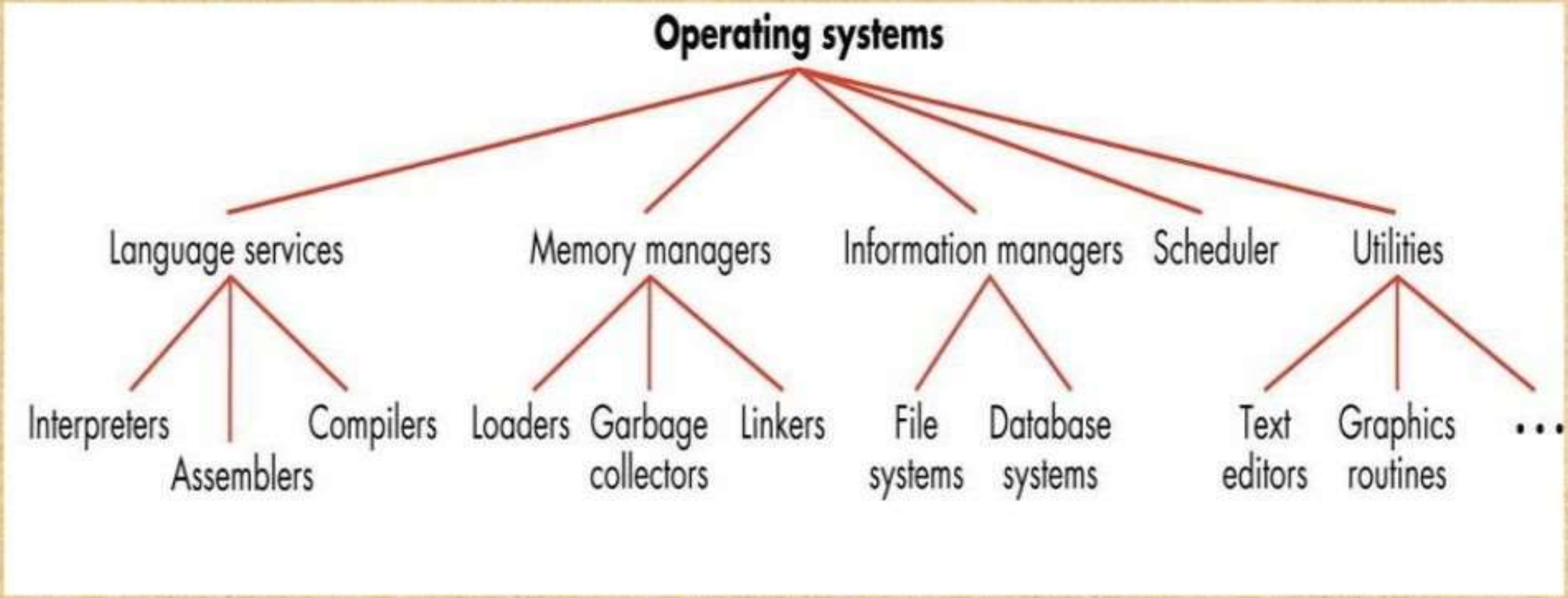
3. System Development Programs :

   They assist in the creation of publication programs.

   e.g., Language translators like interpreters, compilers and assemblers

System Control Programs-OS

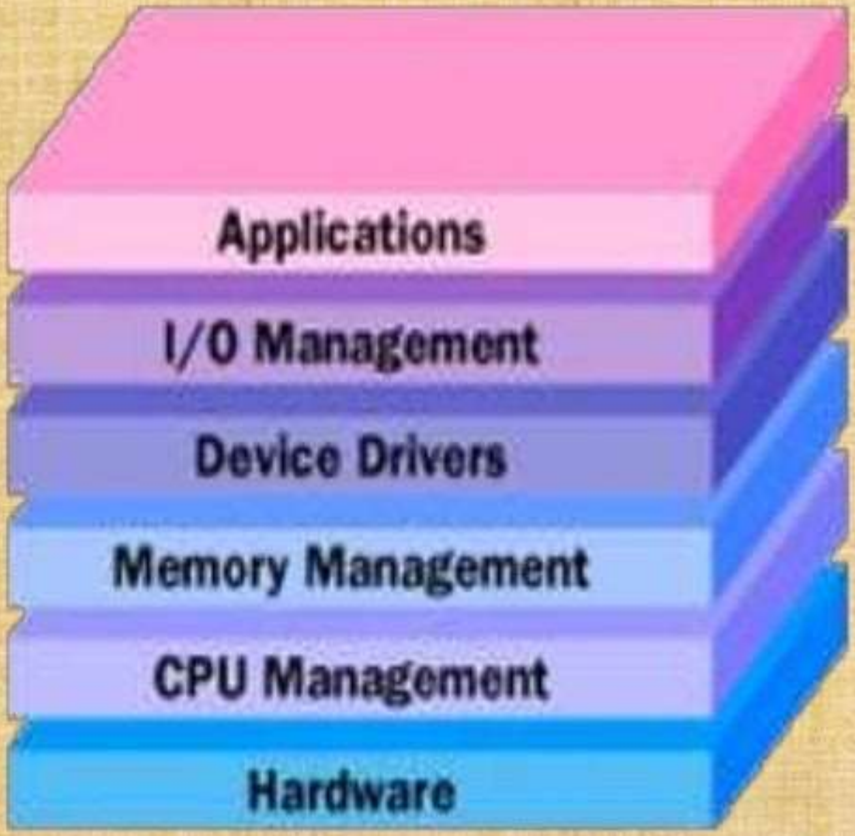
An operating system is an integrated set of specialized programs that are used to manage overall resources of and operations of the computer.



Operating System contd…

Main functions-

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job Accounting



Operating System

- An operating system is software which manages computer hardware and software resources. It also provides common services to computer programs.
- The operating system is an essential component of the system software in a computer system. Application programs are dependent on operating system to function

Windows

Linux

Mac OS

FreeBSD

redhat

solaris

Sun Cobalt

Utilities

- Utilities software is system software which is manufactured to help ,analyze, configure, optimize or to maintain a computer.

- It also helps in maintenance and problem solving of a computer.

Common types of utility programs

- Hardware utilities
- Virus-detection and recovery utilities
- File-compression utilities
- Spam and pop-up blocker utilities
- Backup
- Uninstall

System Development Programs-Language Translators

Language translators are also called language processors.

Main functions are :

- Translate high level language to low level language.
- Check for and identity syntax errors that may be present in the program being installed.

There are **3 types of translator programs-**

1. Assembler
2. Interpreter
3. Compiler

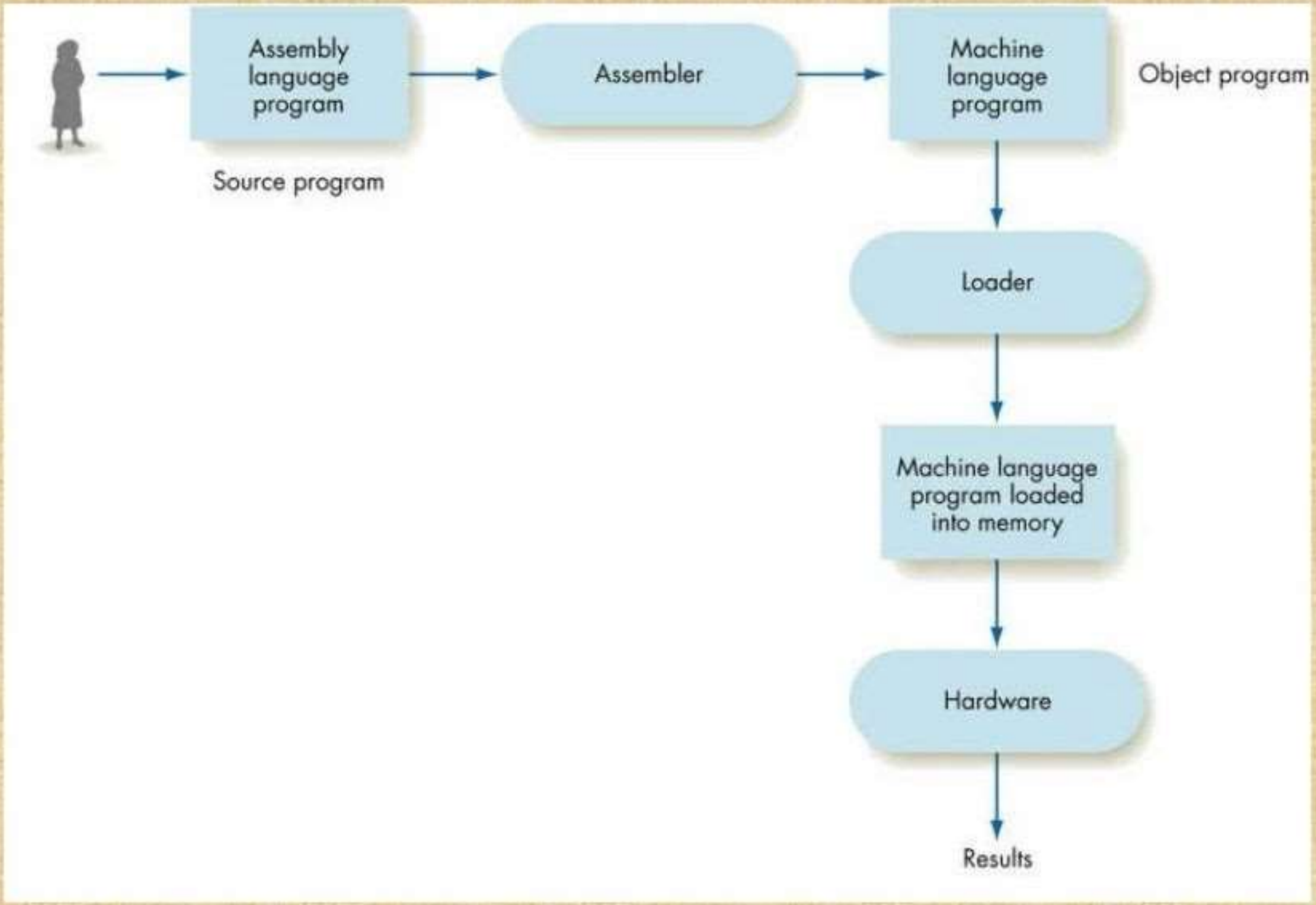
Assembler

- Translates a source program into a corresponding object program.

Assembler tasks

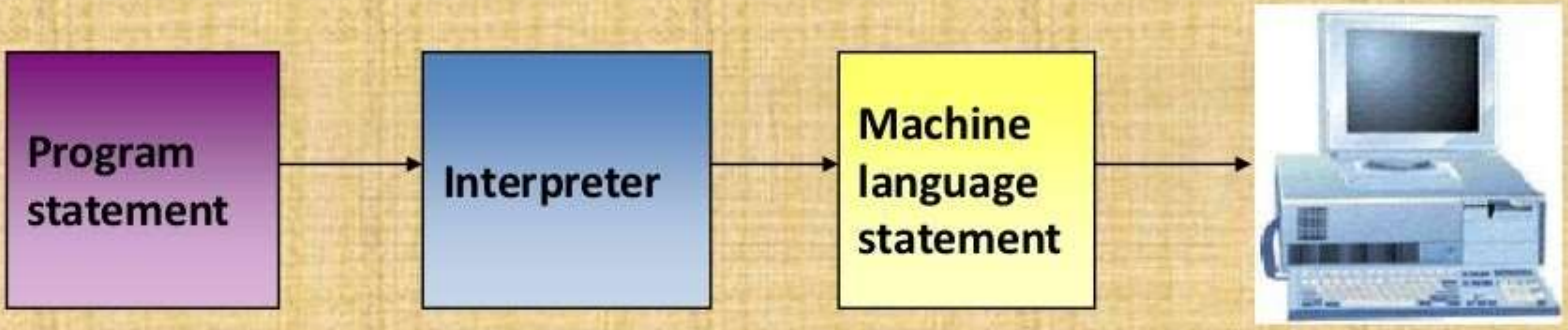
- Convert symbolic op codes to binary
- Convert symbolic addresses to binary
- Perform assembler services requested by the pseudo-ops
- Put translated instructions into a file for future use

How an Assembler works…



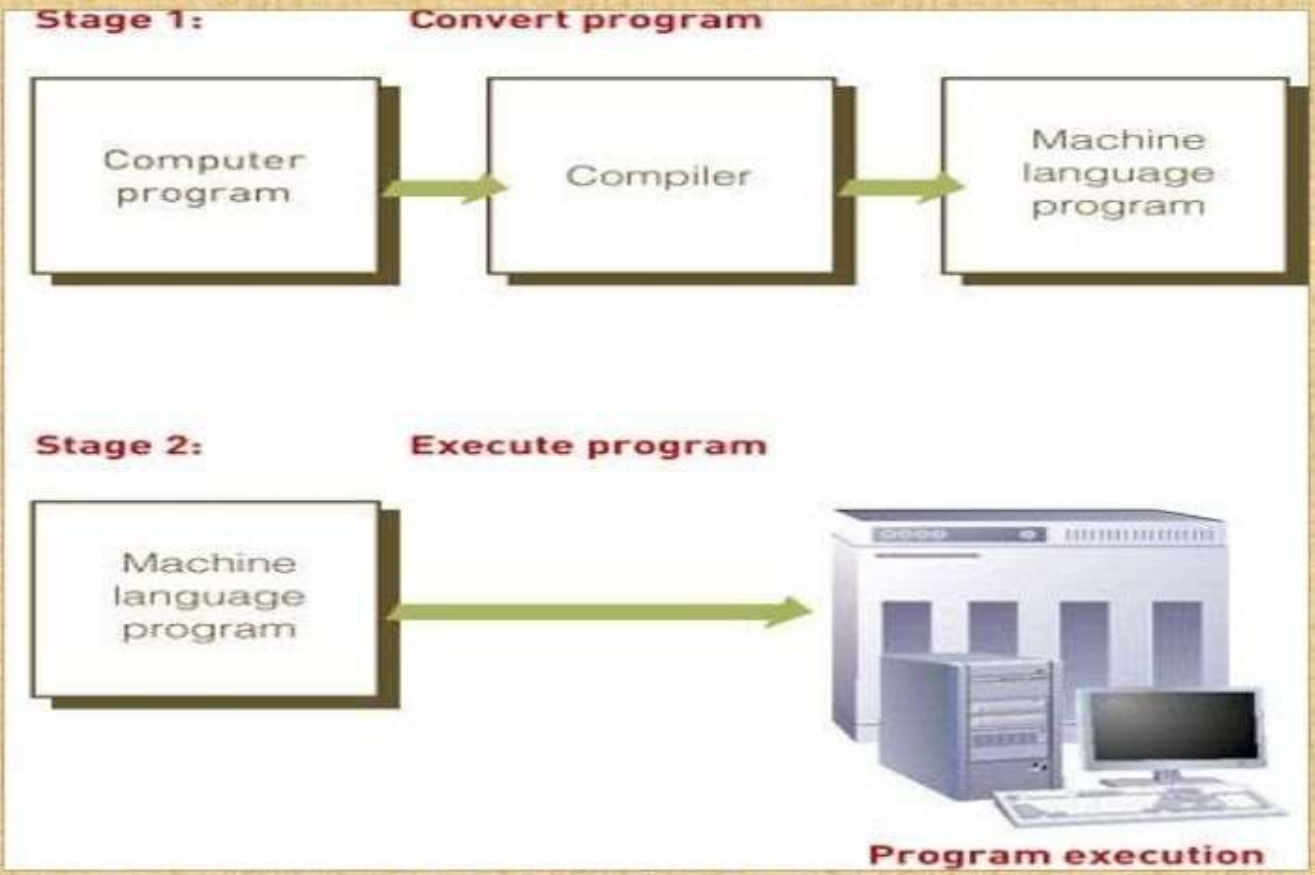
Interpreter

- A language translator that translates one program statement at a time into machine code.



Compiler

• A language translator that converts a complete program into machine language to produce a program that the computer can process in its entirely



Device Drivers

- Device driver is actually a communication device between device and computer
- It loads every time in memory
- When a new device is added the driver should be installed in order to run the program

Features of System Software

- Close to system
- Fast in speed
- Difficult to design
- Difficult to understand
- Less interactive
- Smaller in size
- Difficult to manipulate
- Generally written in low level language

Application Software:

Application Software includes programs that do real work for user.

Example:

Payroll systems, Inventory Control, Manage student database, Word Processor, Spreadsheet and Database Management System etc.,

Application Software

- Actually the application software consists of programs that are designed to make users more comfortable or productive to assist personal tasks
- The application software is present on computer hard disk
- Application software can also be stored on CDs, DVDs, and flash or keychain storage devices

Application Software

- Categories of Application Software
- Types of Application
- Forms of Application Software

Categories of Application Software

- Business Software
- Graphic & Multimedia
- Home / Personal / Education
- Communication

Types of Application Software

- Proprietary
- In-house
- Contract
- Off-the-shelf
- Customized package

Forms of Application Software

- Packaged Software
- Custom Software
- Web Application
- Open Source
- Shareware
- Freeware
- Public-domain Software

Form Of Application Software

- **Package software**

  A software which is sold in a bundle due to similar function of programs.

- **Example**

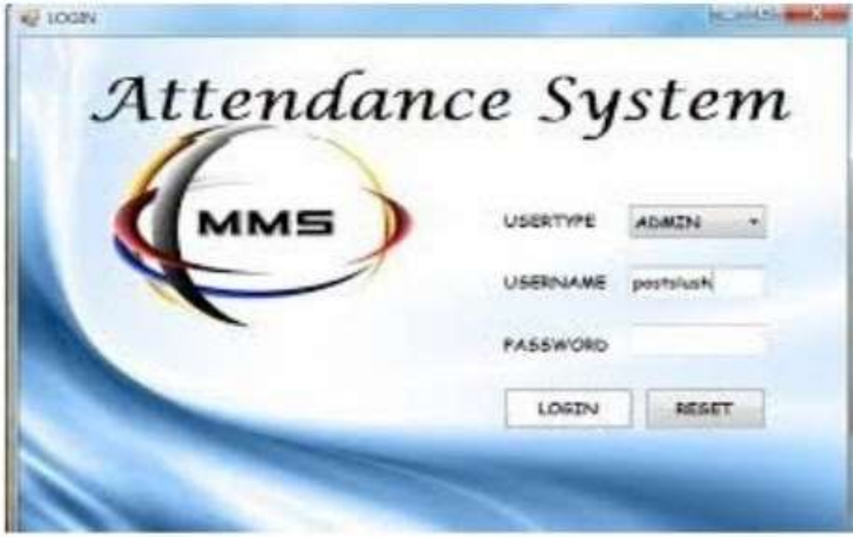
  - Microsoft office, windows Cd

Custom Software

This is software which is specially made for an organization as per their requirement.

- **Example**

I. Attendance system
II. Security code system.



USERTYPE ADMIN
USERNAME postslush
PASSWORD

Web Application

A web application is any application that uses a web browser.

Example

I. Google docs

II. Drop Box

Open source software

- Open source software is made available to every one and can be change, modify and distribute to public without any notification.
- **Example**

I. Linux

II. Moodle

III. Wordpress

IV. Drupal

Free ware

- Free ware is the software that is freely available to public but author has a copy right, means that you can only use it ,not sell it.
- **Example**

1. Antivirus

Public domain software

- Public domain software is totally free and it is not copyrighted plus it have no restriction

Example

I. SQlite
II. Blast
III. I2P

Computer Performance

Performance

- The most important measure of a computer is how quickly it can execute programs.
- Three factors affect performance:
  - Hardware design
  - Instruction set
  - Compiler

Performance

- Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.

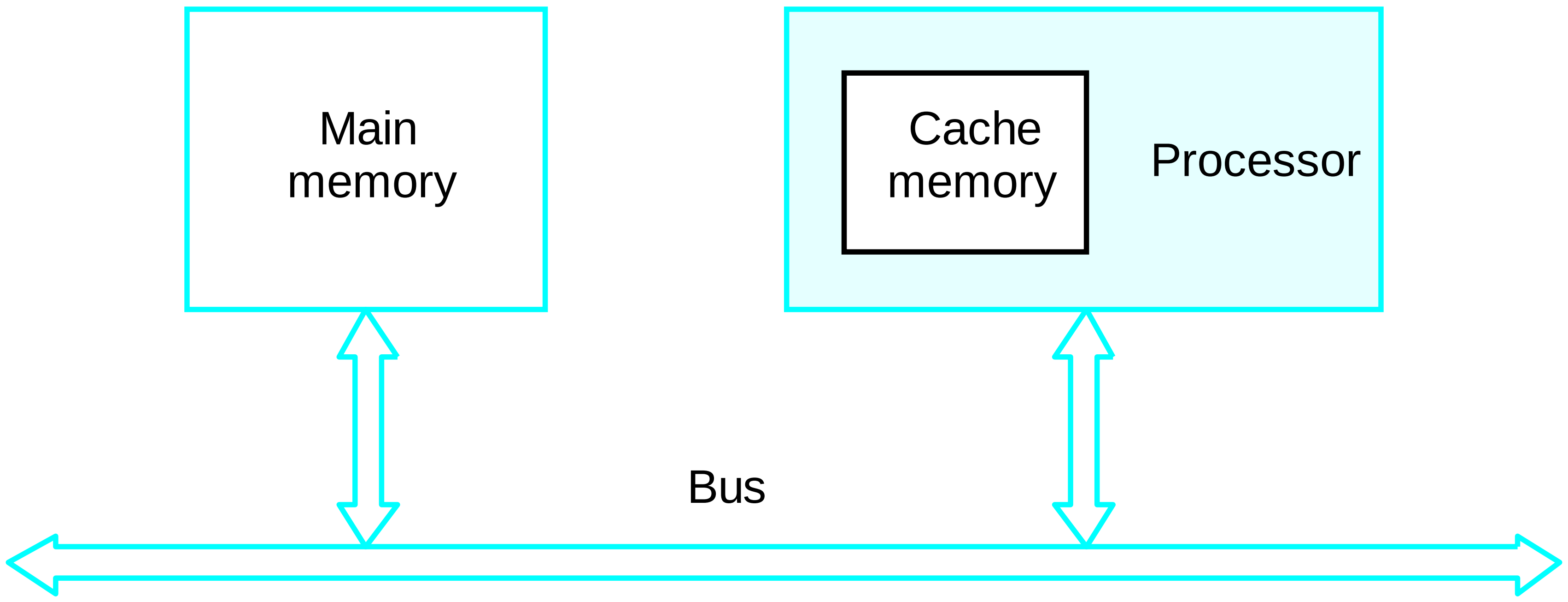


Figure 1.5. The processor cache.

Performance

- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip.
- Speed
- Cost
- Memory management

Processor Clock

- Clock, clock cycle, and clock rate
- The execution of each instruction is divided into several steps, each of which completes in one clock cycle.
- Hertz – cycles per second

Basic Performance Equation

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other

$$T = \frac{N \times S}{R}$$

How to improve T?

Pipeline and Superscalar Operation

- Instructions are not necessarily executed one after another.
- The value of S doesn’t have to be the number of clock cycles to execute one instruction.
- Pipelining – overlapping the execution of successive instructions.
- Add R1, R2, R3
- Superscalar operation – multiple instruction pipelines are implemented in the processor.
- Goal – reduce S (could become <1!)

Clock Rate

- Increase clock rate
  - Improve the integrated-circuit (IC) technology to make the circuits faster
  - Reduce the amount of processing done in one basic step (however, this may increase the number of basic steps needed)
- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor’s operation equally except the time to access the main memory.

CISC and RISC

- Tradeoff between N and S
- A key consideration is the use of pipelining
  - S is close to 1 even though the number of basic steps per instruction may be considerably larger
  - It is much easier to implement efficient pipelining in processor with simple instruction sets
- Reduced Instruction Set Computers (RISC)
- Complex Instruction Set Computers (CISC)

Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal – reduce N×S
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.

Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

$$SPEC\ rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$SPEC\ rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Why Study Performance?

- Make intelligent design choices
- See through the marketing hype
- Key to understanding underlying computer organization
  - Why is some hardware faster than others for different programs?
  - What factors of system performance are hardware related? (e.g., Do we need a new machine, or a new operating system?)

Computer performance

Computer performance is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used.

Computer performance

Depending on the context, good computer performance may involve one or more of the following:

- Short response time for a given piece of work
- High throughput (rate of processing work)
- Low utilization of computing resource(s)
- High availability of the computing system or application
- Fast (or highly compact) data compression and decompression
- High bandwidth / short data transmission time

Computer vs H/W Performance

- Latency/Response Time (clocks from input to corresponding output)
  - —How long does it take for my program to run?
  - —How long must I wait after typing return for the result?
- Throughput (How many results per clock)
  - —How many results can be processed per second?
  - —What is the average execution rate of my program?
  - —How much work is getting done?

If we upgrade a machine with a new processor what do we improve?

Response Time/Latency

If we add a new machine to the lab what do we increase?

Throughput

Design Tradeoffs

- **Maximum Performance: measured by the numbers of instructions executed per**

  Second
- **Minimum Cost: measured by the size of the circuit.**
- **Best Performance/Price: measured by the ratio of MIPS to size. In powersensitive applications MIPS/Watt is important too.**

Aspect of software quality

Computer software performance, particularly software application response time, is an aspect of software quality that is important in human–computer interactions.

Performance Equation

The total amount of time (t) required to execute a particular benchmark program is

$t = N * C/f,$ or equivalently

$P = I * f/N$

where

- $P = 1/t$ is "the performance" in terms of time-to-execute
- N is the number of instructions actually executed (the instruction path length).
- f is the clock frequency in cycles per second.
- C= is the average cycles per instruction (CPI) for this benchmark.
- I= is the average instructions per cycle (IPC) for this benchmark.

Performance Equation

An another performance equation- The equation, which is fundamental to measuring computer performance is :

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

where the time per program is the required CPU time.

Comparing the performance of two systems

In comparing the performance of two systems we measure the time that it takes for each system to perform the same amount of work. If the same program is run on two systems, System A and System B, System A is *n times as fast* as System B if:

$$\frac{\text{running time on system B}}{\text{running time on system A}} = n.$$

Comparing the performance of two systems

System A is x% faster than System B if:

$$\left(\frac{\text{running time on system B}}{\text{running time on system A}} - 1\right) \times 100 = x.$$

These formulas are useful in comparing the average performance of one system with the average performance of another.

Execution Time

- **Elapsed Time/Wall Clock Time**

  counts everything *(disk and memory accesses, I/O , etc.)*

  a useful number, but often not good for comparison purposes

- **CPU time**

  Doesn't include I/O or time spent running other programs can be broken up into system time, and user time

- **Our focus: user CPU time**

  Time spent executing actual instructions of "our" program

Computer Performance Measure

Millions of Instructions per Second

Frequency in MHz

$$\mathrm{MIPS} = \frac{\text{clocks/sec}}{\text{AVE(clocks/instruction)}}$$

CPI (Average Clocks Per Instruction)

How to Improve Performance?

$$\frac{\text{seconds}}{\text{program}} = \boxed{\frac{\text{cycles}}{\text{program}}} \times \boxed{\frac{\text{seconds}}{\text{cycle}}} \qquad \text{MIPS} = \frac{\boxed{\text{Freq}}}{\boxed{\text{CPI}}}$$

So, to improve performance (everything else being equal) you can either

Decrease the # of required cycles for a program,or(improve ISA/Compiler)

Decrease the clock cycle time or, said another way,

Increase the clock rate. (reduce propagation delays or use pipelining)

Decrease the CPI (average clocks per instruction) (new H/W)

THE HISTORY OF COMPUTER DEVELOPMENT

- A long time ago, human are using their fingers, stones etc to do calculation.
- At the same time, they are trying to create an apparatus that could facilitate the calculation process.
- After a few trial, finally the complex and advance calculation system has been produced and it is known as a computer.

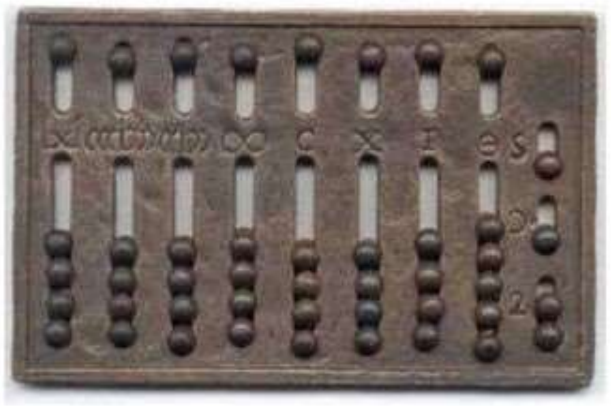
- The History & Evolution Of Computer Basically, the history of computer development is divided into 2 parts :

before 1940 & after 1940 .

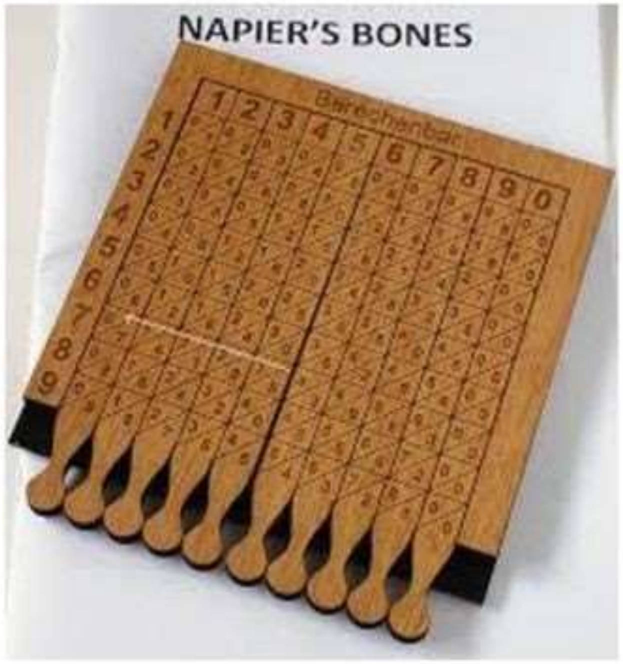
BEFORE 1940

Abacus Counting Device

- Created on 3000 B.D. at Babylonia.
- Was the first mechanical counting device in the world.
- Able to execute addition and subtraction operation .



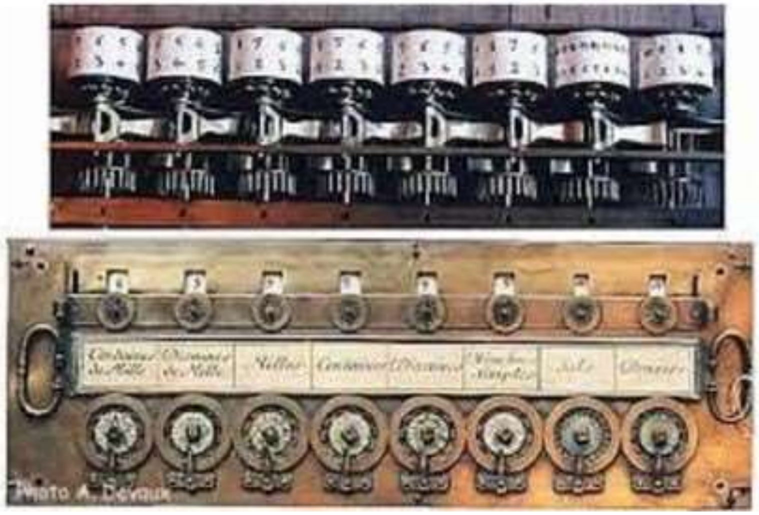
John Napier's Bone



- Created on 1614 by John Napier.
- Facilitate multiplication and division processes – faster & easier.
- The first logarithm table has been created.

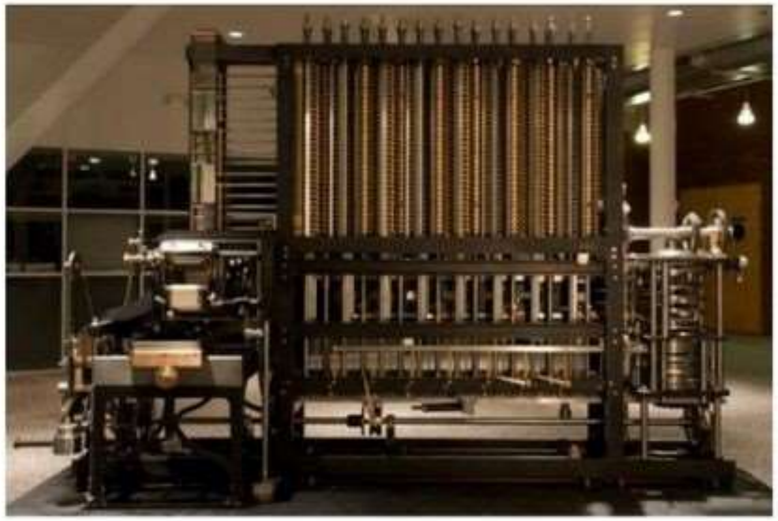
Pascaline Machine

- Created on 1642 by Braise Pascal.
- Was the first mechanical machine or calculator in the world.
- Able to execute addition and subtraction processes.



Babbage Differentiation Machine

- Created by Charles Babbage on 1821.
- Was the first mechanical machine which is used the steam power.
- Able to do a calculation and printing the output automatically.



Babbage Analytical Engine

- It has five (5) main parts :
  - Input unit
  - Output unit
  - Processing Unit
  - Control unit
  - Memory unit
- His invention has became a theory model for today's computer technology. Because of that, Charles Babbage has been known as The Ancestor of A Modern Computer

1834
Babbage's Analytical Engine designed but never built, with similar ideas to the modern computer: sections for input, processing and output

After 1940

Evolution

First Generation

- MARK 1
- ENIAC
- Von Neumann Machine
- UNIVAC
- IBM

Second Generation

- Transistors

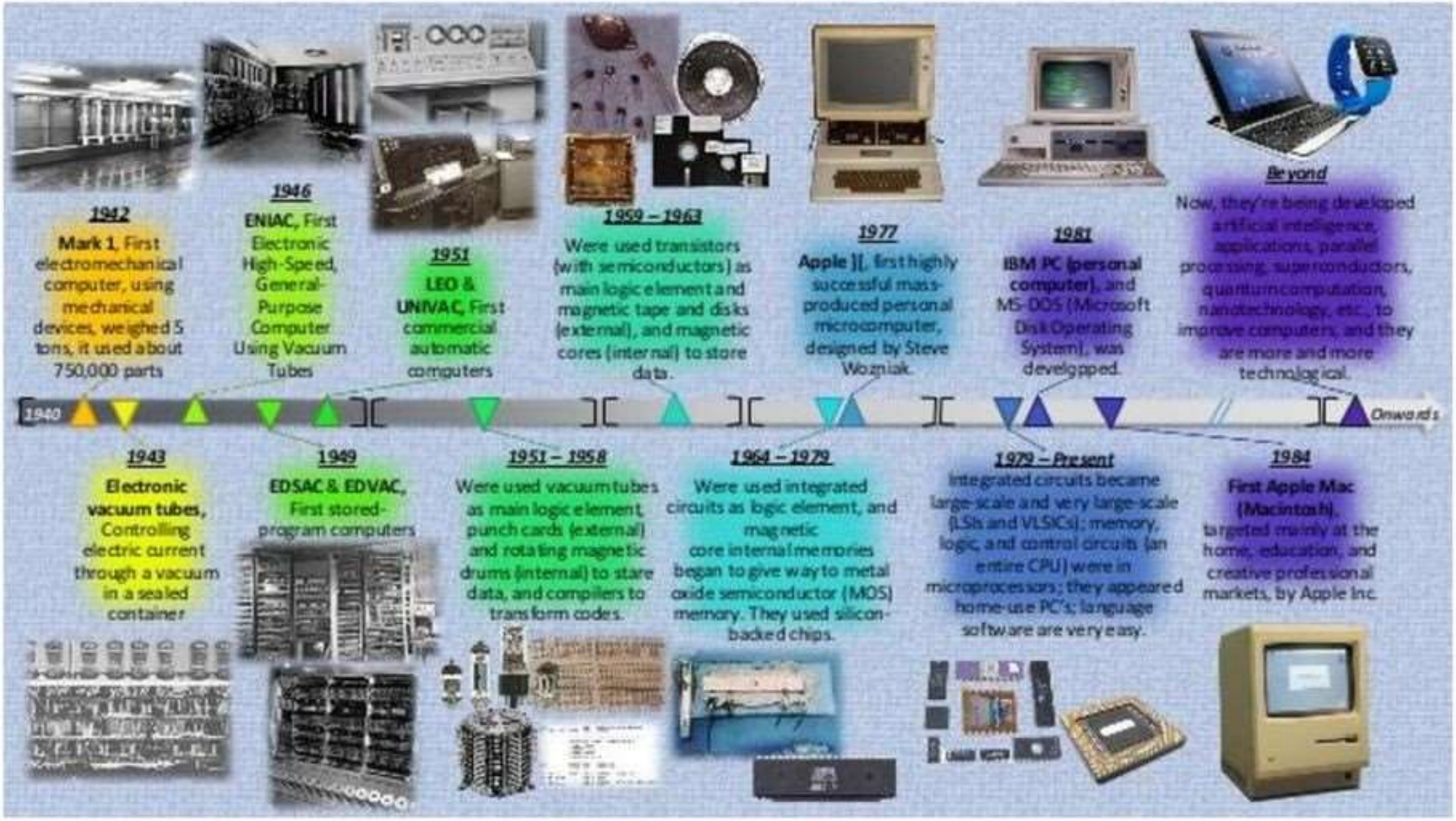
Third Generation

- Integrated Circuit

Later Generation

- Microprocessor

COMPUTER SYSTEM



1942
Mark 1, First electromechanical computer, using mechanical devices, weighed 5 tons, it used about 750,000 parts

1946
ENIAC, First Electronic High-Speed, General-Purpose Computer Using Vacuum Tubes

1951
LEO & UNIVAC, First commercial automatic computers

1959 – 1963
Were used transistors (with semiconductors) as main logic element and magnetic tape and disks (external), and magnetic cores (internal) to store data.

1977
Apple][, first highly successful mass-produced personal microcomputer, designed by Steve Wozniak.

1981
IBM PC (personal computer), and MS-DOS (Microsoft Disk Operating System), was developped.

Beyond
Now, they're being developed artificial intelligence, applications, parallel processing, superconductors, quantum computation, nanotechnology, etc., to improve computers, and they are more and more technological.

1943
Electronic vacuum tubes, Controlling electric current through a vacuum in a sealed container

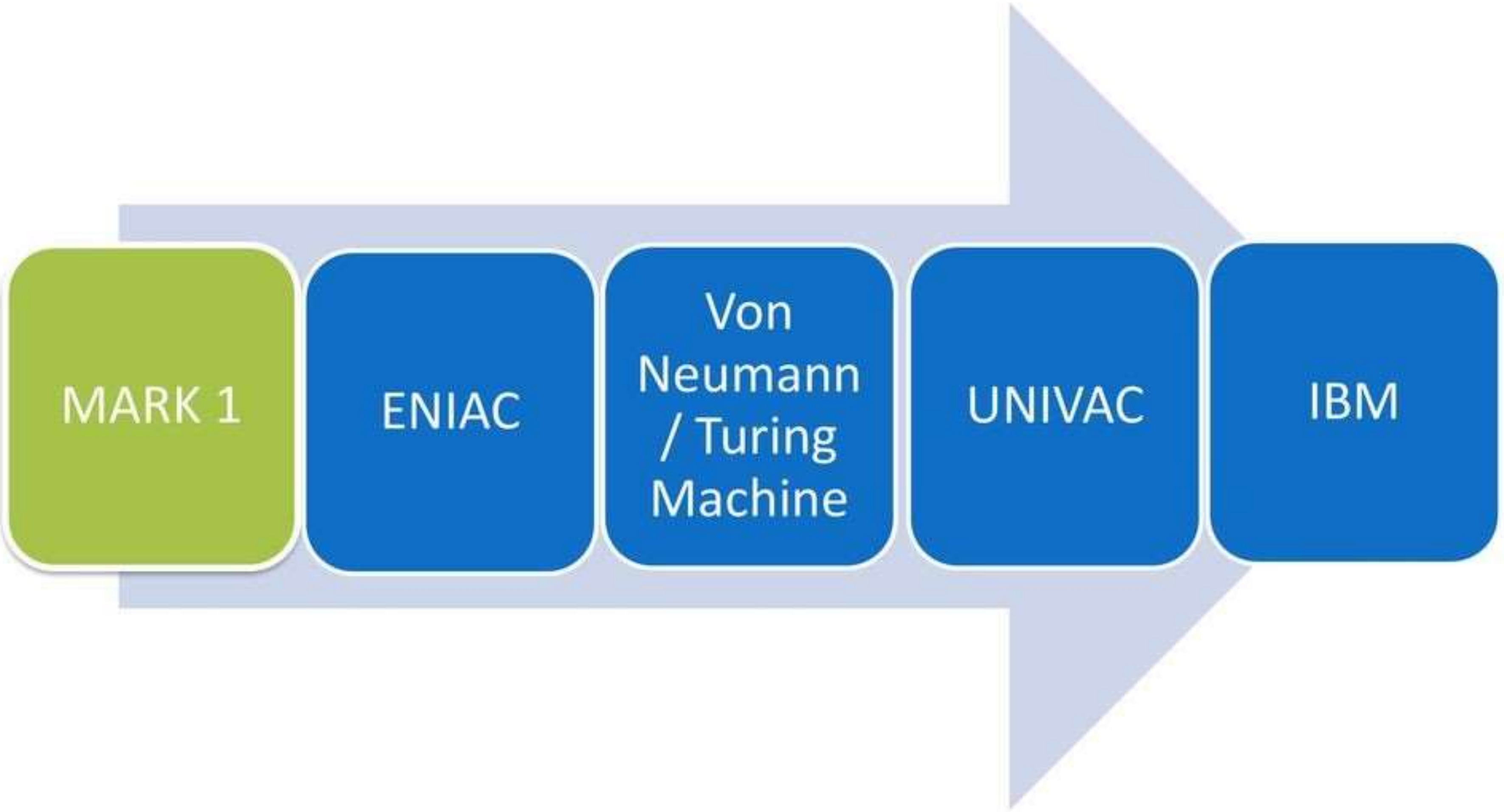
1949
EDSAC & EDVAC, First stored-program computers

1951 – 1958
Were used vacuum tubes as main logic element, punch cards (external) and rotating magnetic drums (internal) to store data, and compilers to transform codes.

1964 – 1979
Were used integrated circuits as logic element, and magnetic core internal memories began to give way to metal oxide semiconductor (MOS) memory. They used silicon-backed chips.

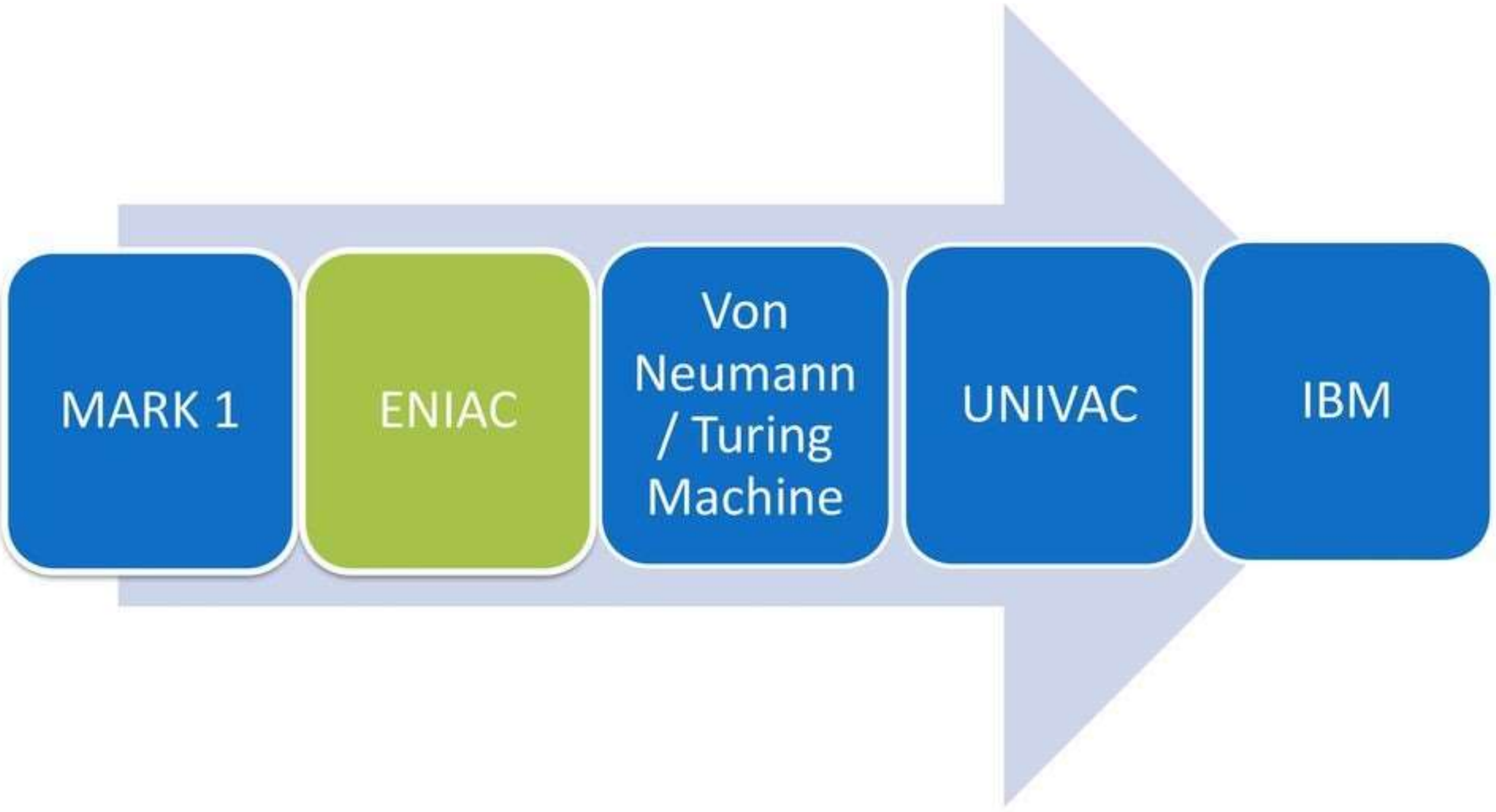
1979 – Present
Integrated circuits became large-scale and very large-scale (LSIs and VLSICs); memory, logic, and control circuits (an entire CPU) were in microprocessors; they appeared home-use PC's; language software are very easy.

1984
First Apple Mac (Macintosh), targeted mainly at the home, education, and creative professional markets, by Apple Inc.


MARK 1
ENIAC
Von Neumann / Turing Machine
UNIVAC
IBM

Mark 1

- Created on 1941 by Dr. Howard Aikern in conjunction with IBM.
- Was the first electro-mechanical computer.
- Size : 55 feet long, 8 feet height and connected with 800 km of wire.


MARK 1
ENIAC
Von Neumann / Turing Machine
UNIVAC
IBM

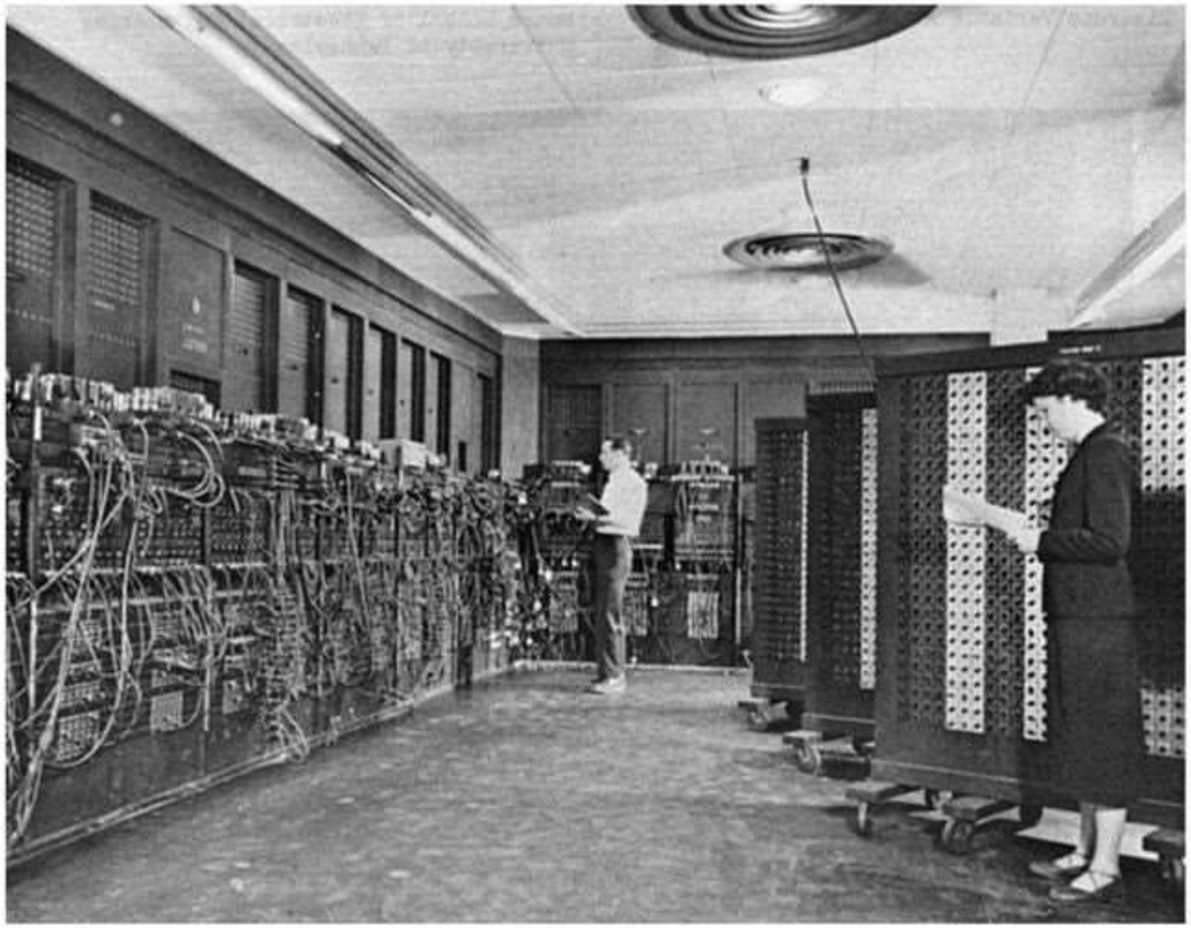
ENIAC

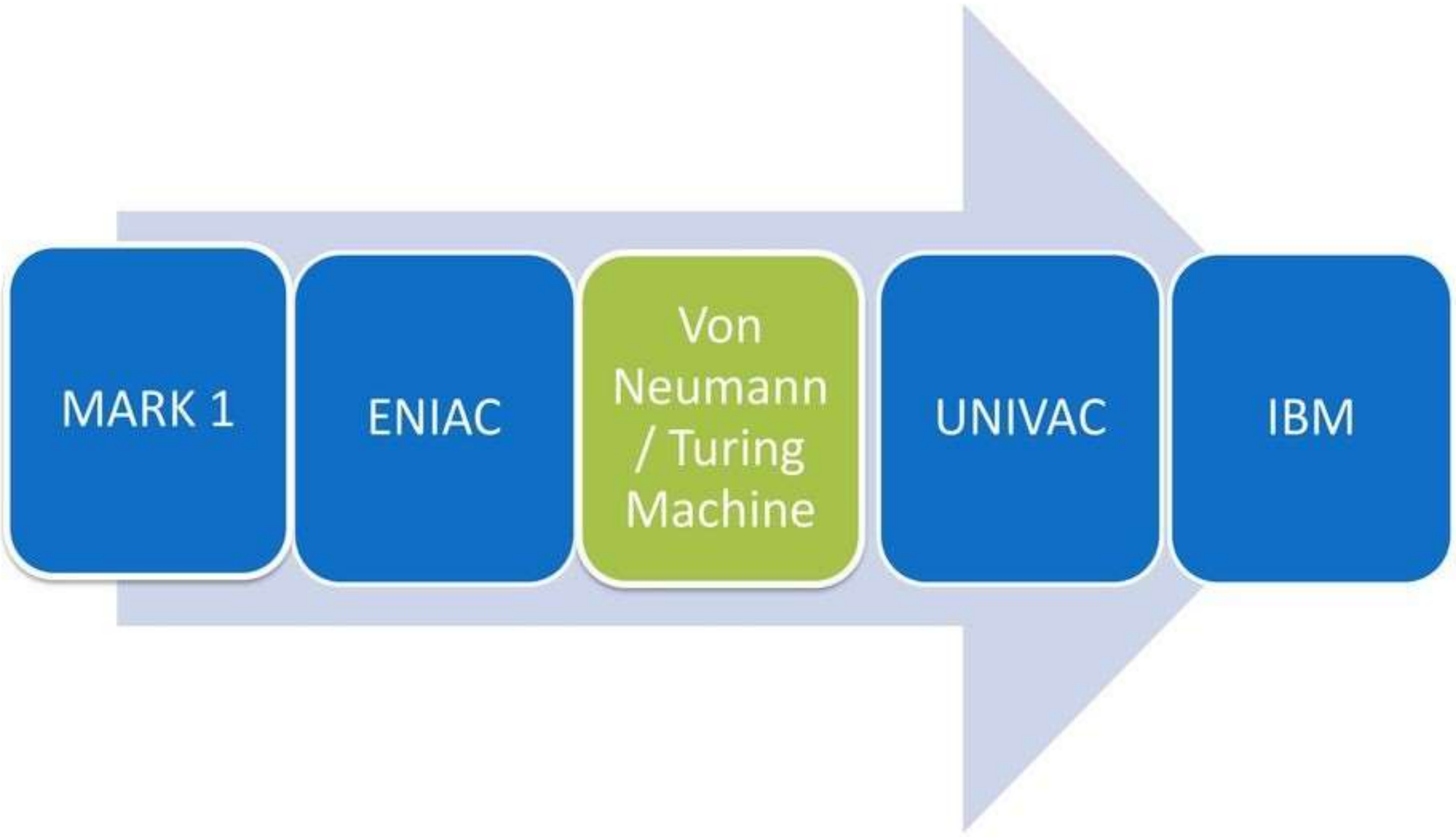
- Electronic Numerical Integrator And Computer
- Eckert and Mauchly
- University of Pennsylvania
- Trajectory tables for weapons
- Started 1943
- Finished 1946
  - ENIAC was created to help with the war effort against German forces.Used until 1955

ENIAC

- Decimal (not binary)
- 20 accumulators of 10 digits
- Programmed manually by switches
- 18,000 vacuum tubes
- 30 tons
- 15,000 square feet
- 140 kW power consumption
- 5,000 additions per second
- 1000 times faster than Mark 1.

COMPUTER SYSTEM




MARK 1
ENIAC
Von Neumann / Turing Machine
UNIVAC
IBM

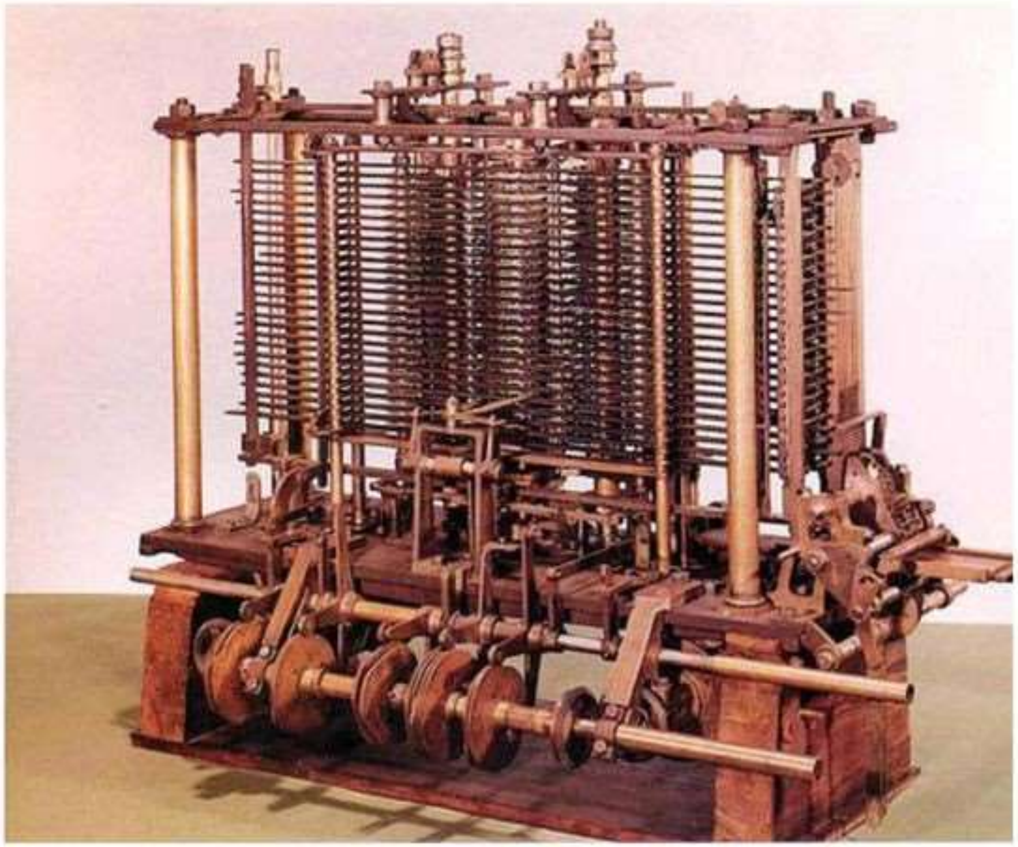
Stored-program concept

- The task of entering and altering program is tedious for ENIAC. Suppose a program could be represented in a form suitable for storing in memory alongside data. Then computer could get its instruction by reading them from memory and program could be set or altered by settings the values of a portion of memory. This idea is known as stored-program concept an developed by Von Neumann referred to IAS computer.

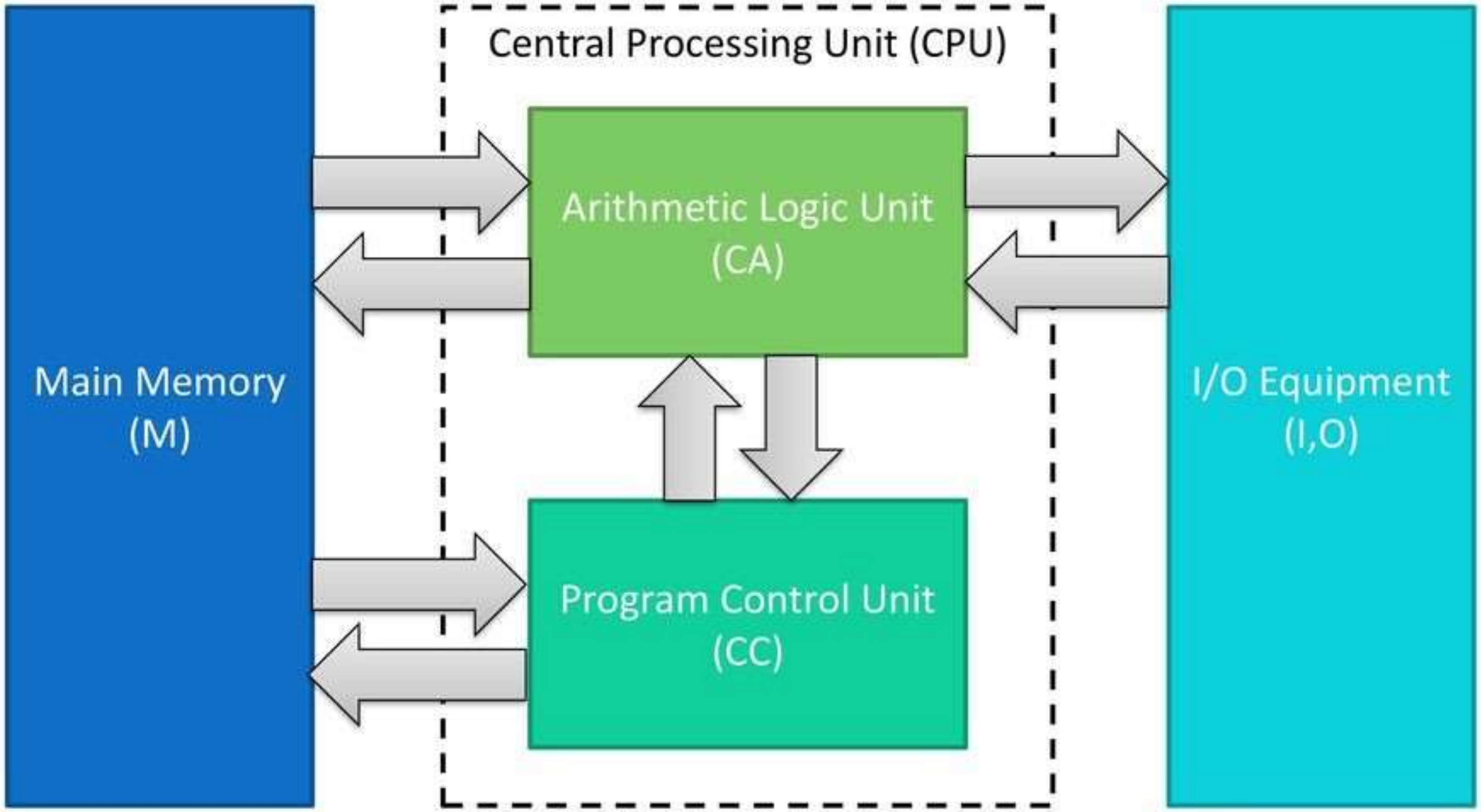
Von Neumann / Turing Machine

- Stored Program concept
  - Main memory storing programs and data
  - ALU operating on binary data
  - Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit

Von Neumann / Turing Machine - Example



Von Neumann Machine - Structure



Von Neumann earlier proposal

- First – since the device is primarily a computer, it will have to perform addition, subtraction, multiplication or addition therefore it need special organs to do it, therefore come the CA
- Second- The logical control of the device that is the proper sequencing of its operation, carried out by central control organ, therefore come the second part, CC

Von Neumann earlier proposal

- Third – Any device that is to carried out long and complicated operation need considerable memory, therefore come the third specific part of the device, M
- These three specific part called CC, CA and M correspond to the associative neurons nervous system

- Fourth- The device need an organ to transfer from R to specific part of C or M, these organ form its input called, I.
- Fifth – The device must have an organ to transfer from C or M to R outside specific medium. These organ form O, output.

Von Neumann / Turing Machine (2)

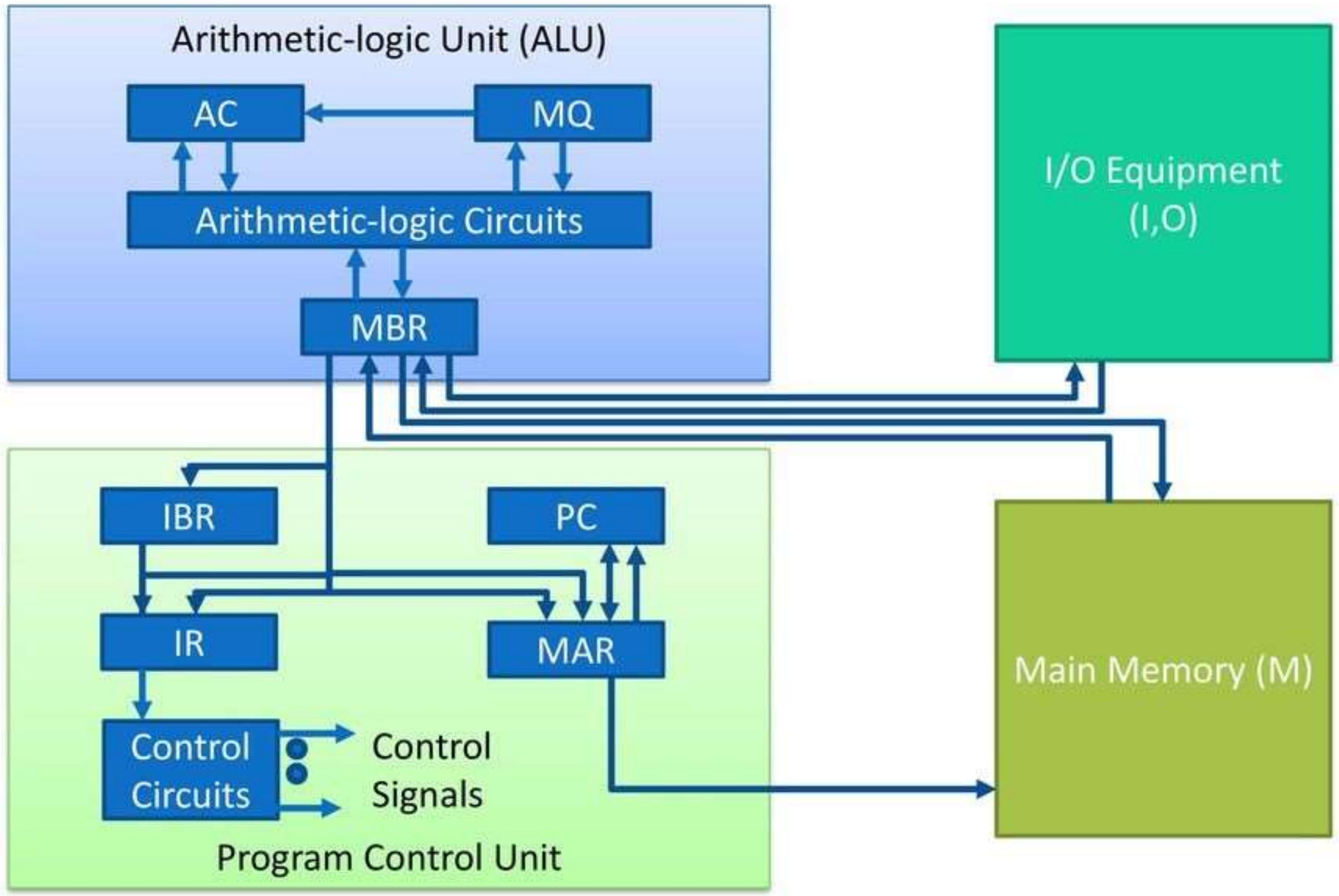
- Princeton Institute for Advanced Studies
  - IAS
- Completed 1952

IAS

- 1000 x 40 bit words
  - Binary number
  - 2 x 20 bit instructions
- Set of registers (storage in CPU)
  - Memory Buffer Register – contains word to be stored/received from in memory or sent to i/o unit.
  - Memory Address Register – specifies the address in memory of the word to be written from or read into MBR.
  - Instruction Register - contains 8-bit operation code instruction being executed.
  - Instruction Buffer Register – to hold temporarily the instruction
  - Program Counter – contain address of the next instruction.
  - Accumulator
  - Multiplier Quotient

} hold temporarily operands and result of ALU operation.

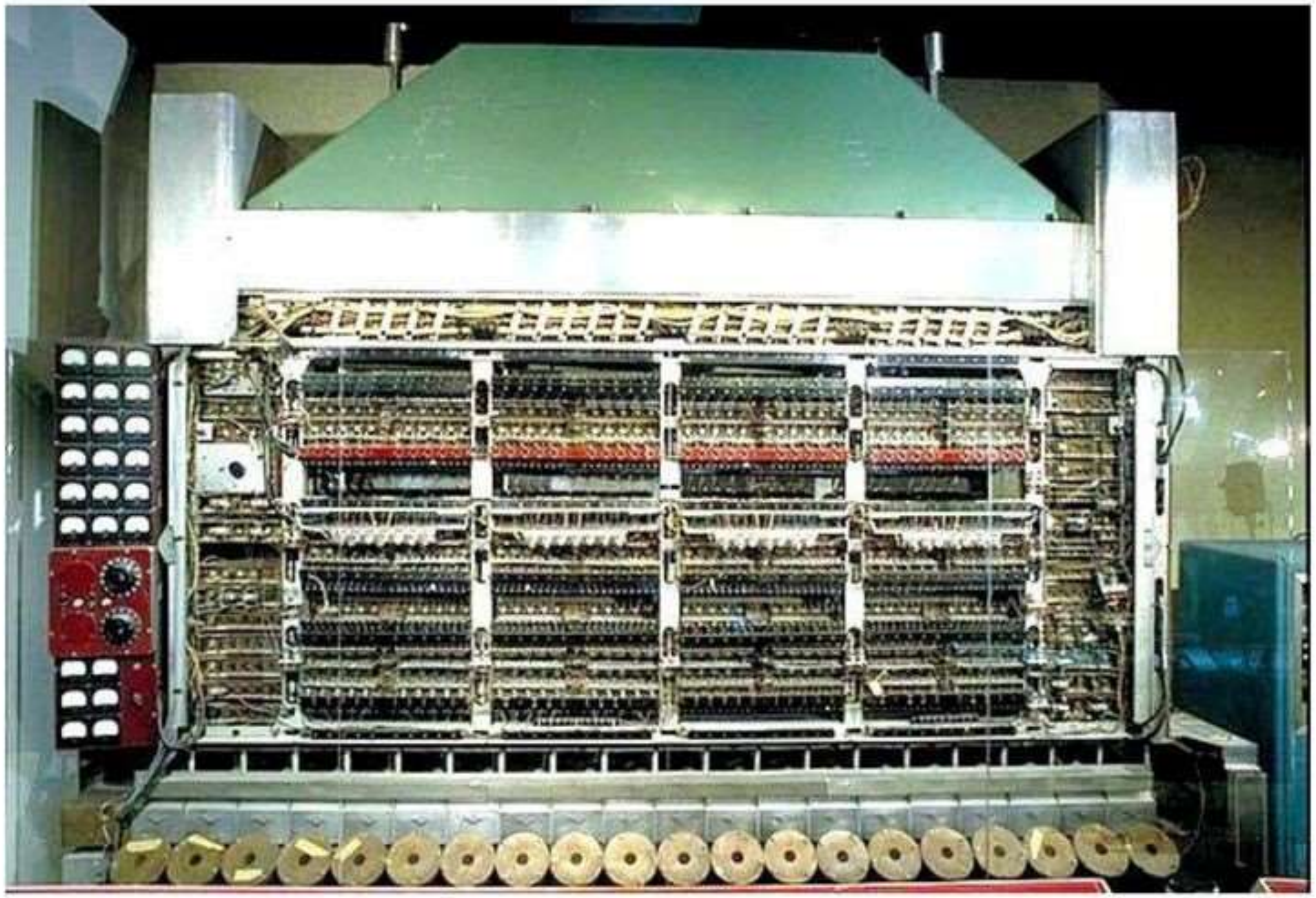
IAS – Structure

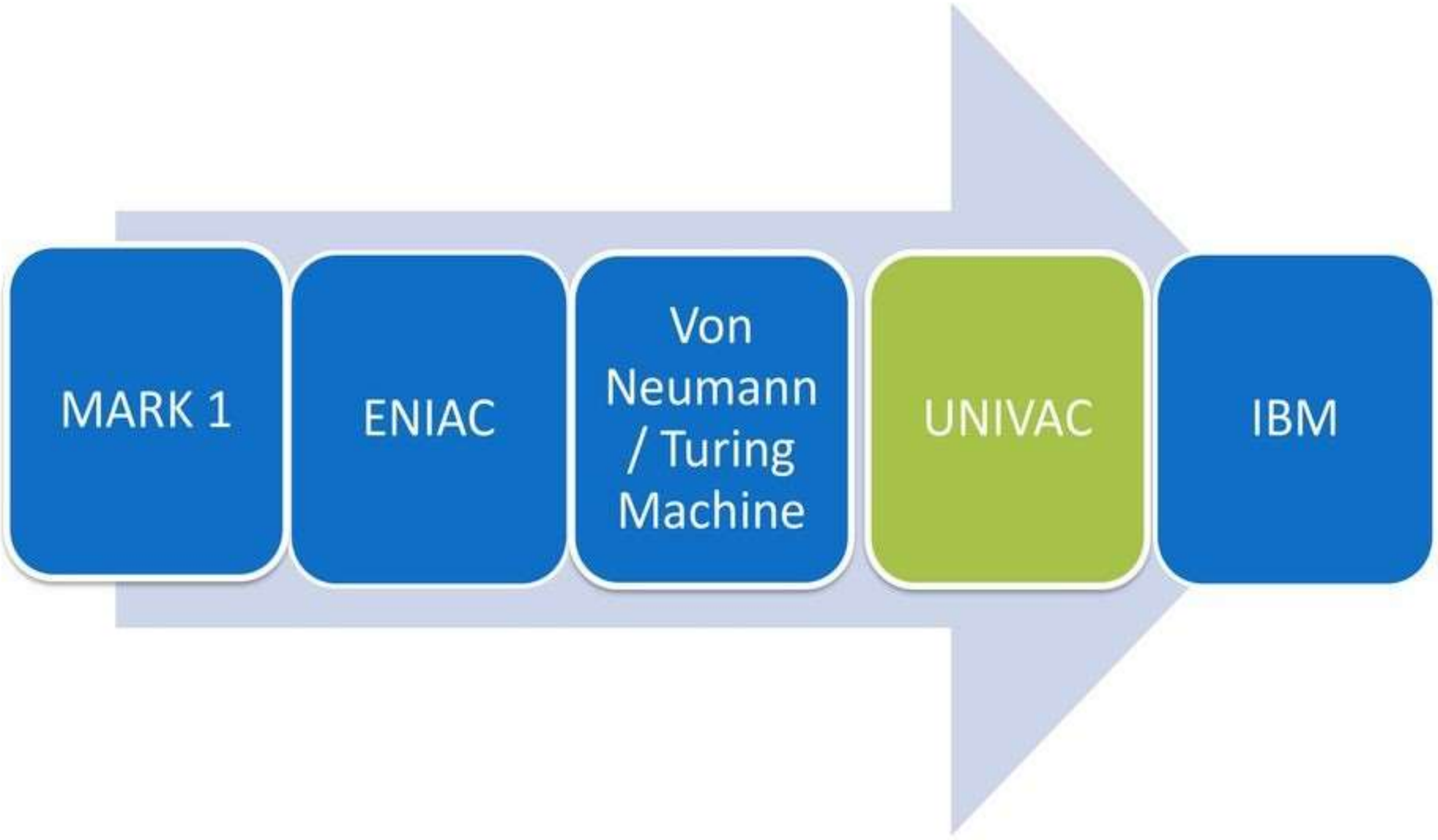


IAS operation

- The opcode of the next instruction is loaded into the IR and the address portion is loaded into MAR. The instruction may by be taken from IBR or it can be taken from memory by loading a word into MBR, then down to IBR, IR and MAR.
- Once in the IR, the control circuitry interprets the opcode and execute the instruction by sending the right signal to moved the data or an operation to be performed by the ALU.

IAS Computer - Example




MARK 1
ENIAC
Von Neumann / Turing Machine
UNIVAC
IBM

Universal Automatic Computer (UNIVAC)

1947
UNIVAC I
Eckert-Mauchly Formed Computer Corporation

(to manufacture computer commercially)

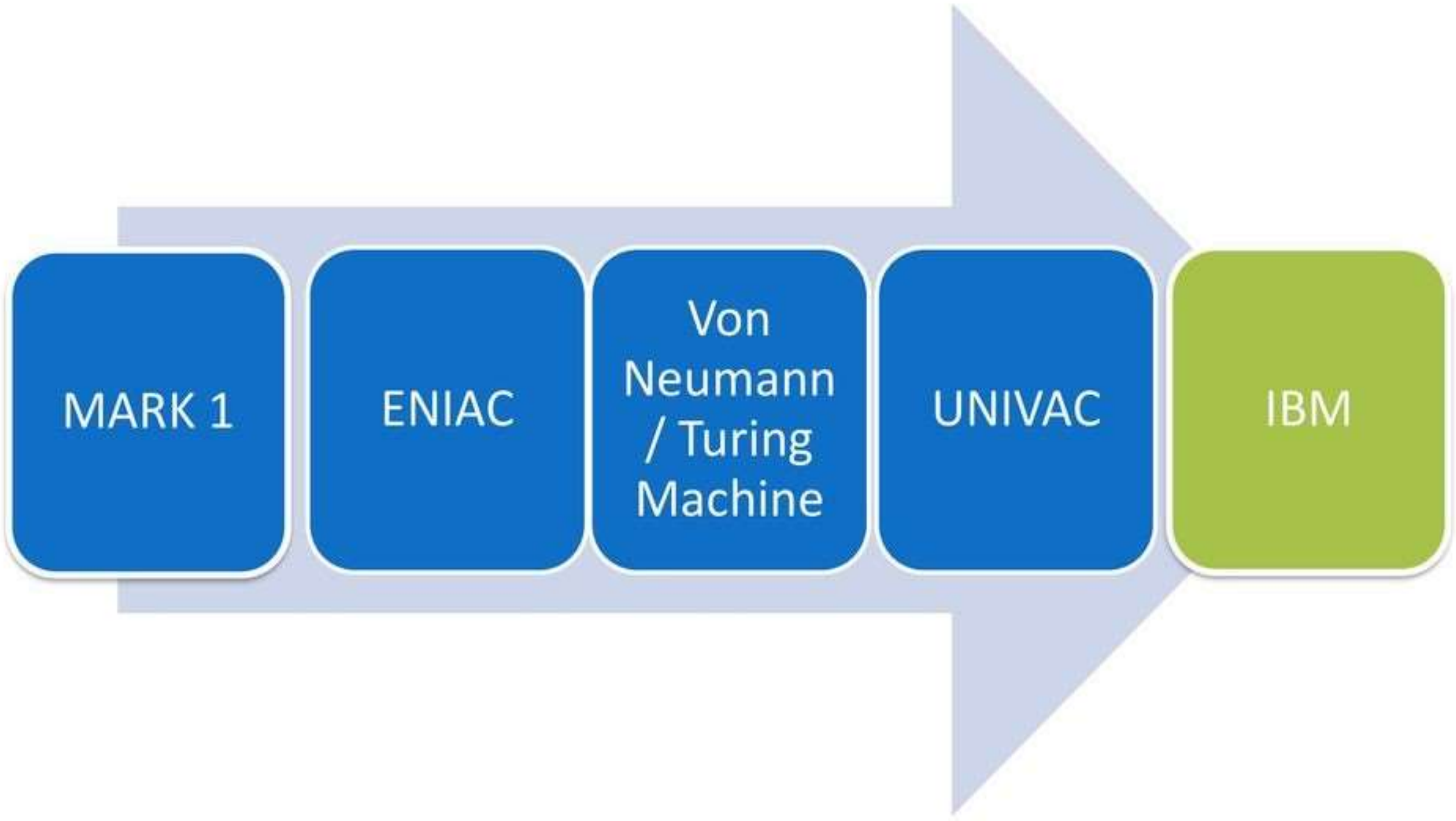
Late 1950
UNIVAC II
Part of Sperry-Rand Corporation

- Faster & more memory

UNIVAC

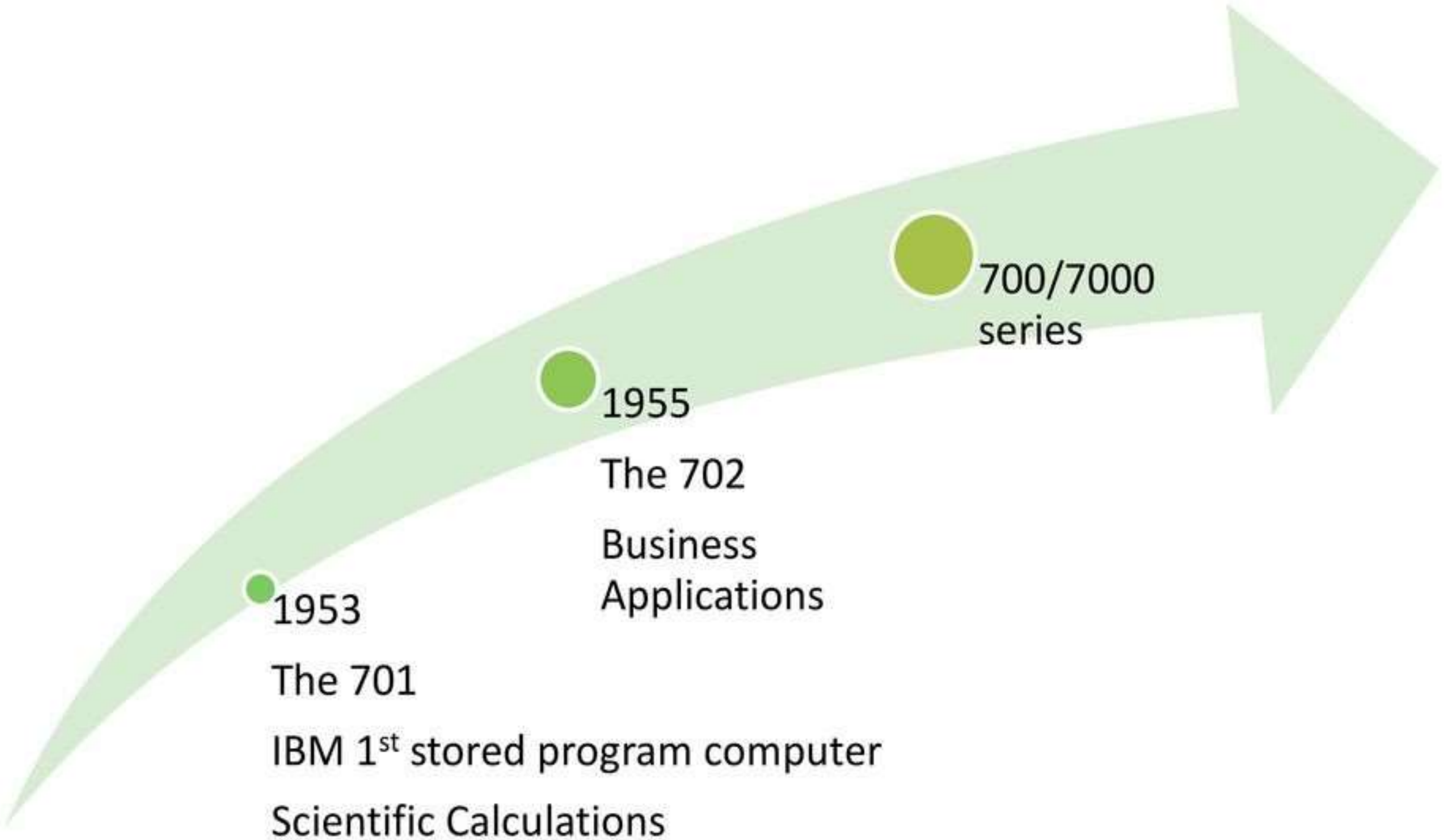
- UNIVAC I – the first successful commercial computer. Used for scientific and commercial application ie, matrix algebraic computation, statistical problem, premium billings, or life insurance company and logistic problem.
- UNIVAC II – greater memory capacity and higher performance

UNIVAC - Example


MARK 1
ENIAC
Von Neumann / Turing Machine
UNIVAC
IBM

Second Generation Machine

IBM



1953

The 701

IBM 1st stored program computer

Scientific Calculations

1955

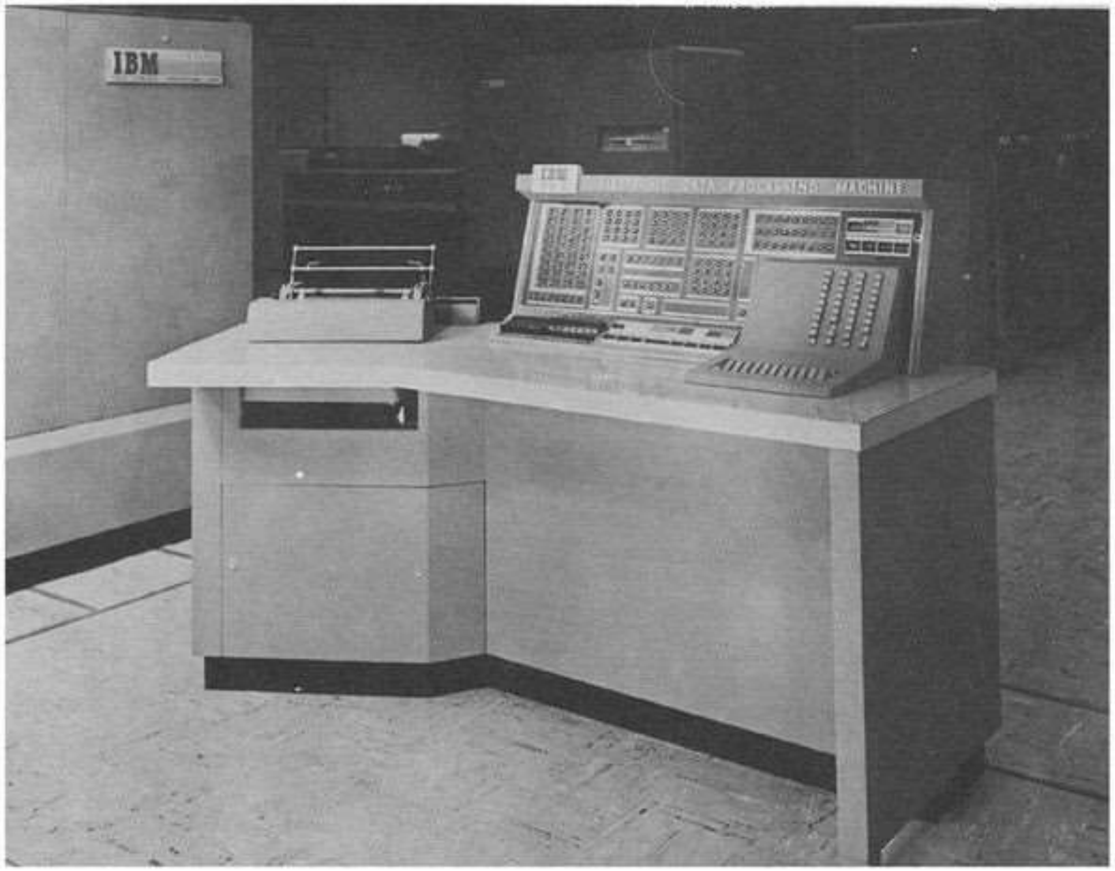
The 702

Business Applications

700/7000 series

IBM 701

IBM 702



IBM 700/7000

Transistors

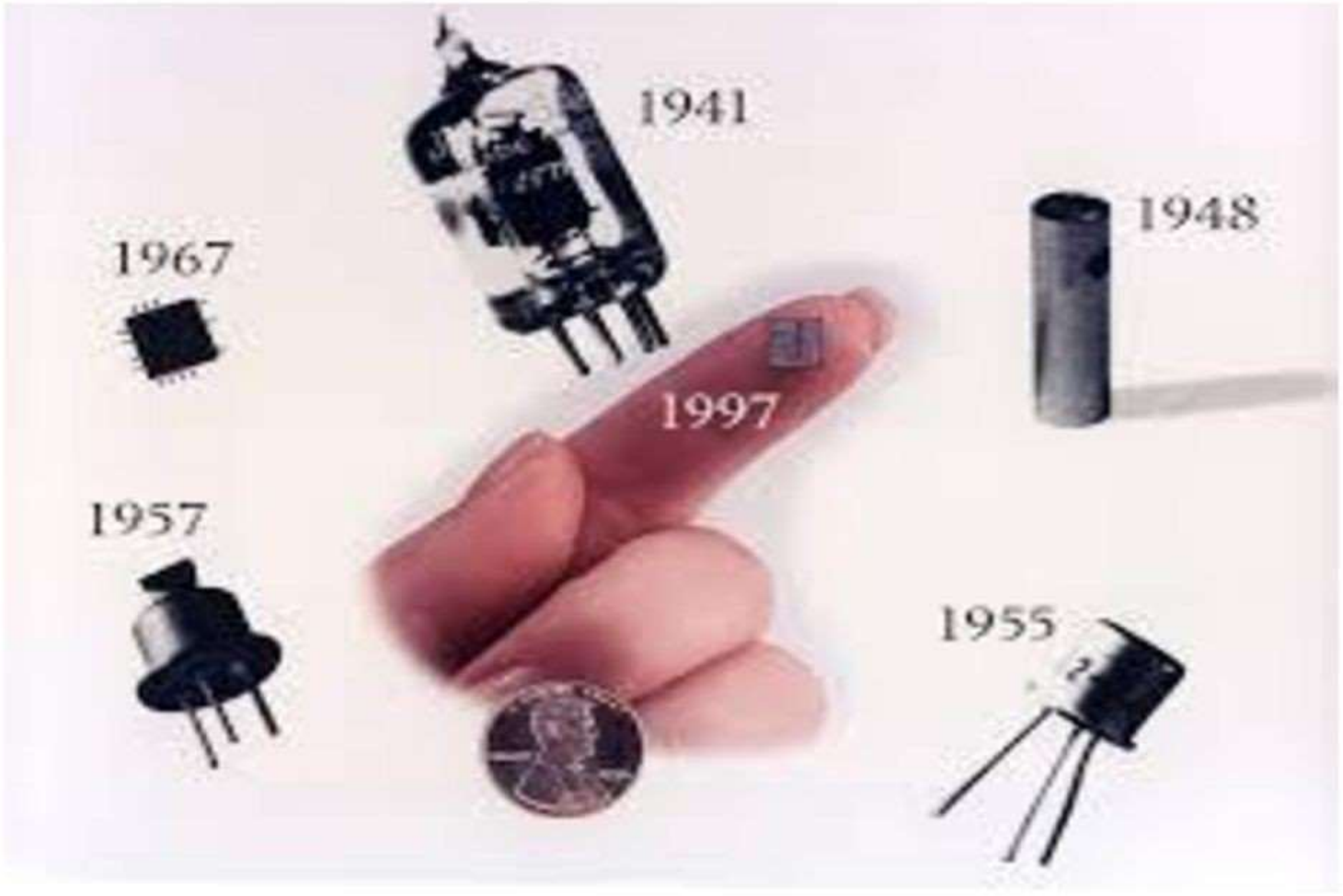
- Made from Silicon (Sand)
- Invented 1947 at Bell Labs
- William Shockley et al.
- Replaced vacuum tubes: wires, metal plates, glass capsule and vacuum.
- Solid State device made from silicon.

Advantages of Transistors

- Smaller
- Cheaper
- Less heat dissipation

Transistors Based Computers

- Second generation machines
- NCR & RCA produced small transistor machines
- IBM 7000
- Digital Equipment Corporation(DEC) - 1957
  - Produced PDP-1 – first mini computer phenomenon.


1941
1948
1967
1997
1957
1955

Third Generation Machine

Integrated Circuit/Microelectronics

- Literally - "small electronics"
- Transistors were replaced by integrated circuits(IC)
- One IC could replace hundreds of transistors
- This made computers even smaller and faster.

Integrated circuit

- In 1958, came an achievement that revolutionized electronics and started the era of microelectronic, the invention of integrated circuit, defined the third generation of computer.
- Initially only a few gates and memory cells, could be reliably manufactured together. As time went on, it become possible to pack more and more component on the same chip

- The cost of chip remained unchanged during the period growth of density. This means the cost of memory circuitry has fallen at a dramatic rate
- Because logic and memory elements are placed closer together on more densely packed chips, the electrical path length is shortened, increasing operating speed.

- The computer become smaller
- Reduction in power and cooling requirement
- The interconnection of integrated circuit are more reliable than solder connection.

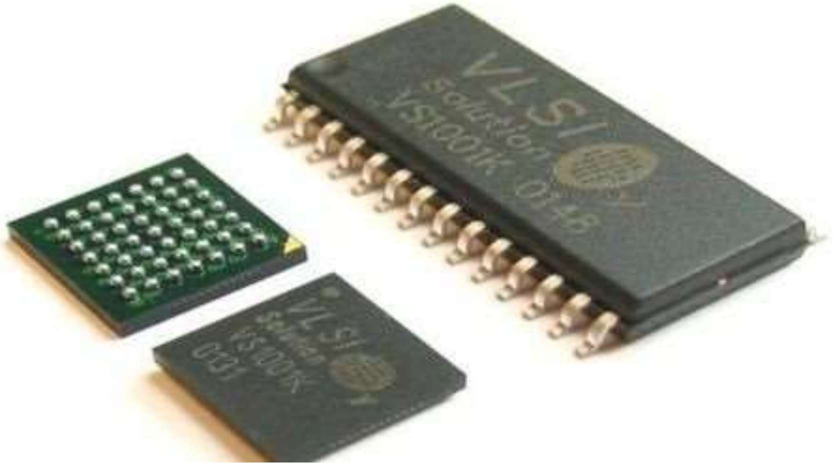
Later Generation Computers

Later Generation Computers

- In 1970 the Intel Corporation invented the Microprocessor: an entire CPU on one chip
- This led to microcomputers-computers on a desk

Later Generation Computers

- This transformation was a result of the invention of the ***microprocessor***.
- A microprocessor (uP) is a computer that is fabricated on an integrated circuit (IC).
- Computers had been around for 20 years before the first microprocessor was developed at ***Intel*** in 1971.



Microprocessor

- More than 1000 component can be placed on a single integrated chip. VLSI achieved more than 10000 component on single chip.
- Just as density of element of memory chips has continue to rise, so has the density of elements on processor chips. As time went on, more and more elements were placed on each chip, so that fewer and fewer chips were needed to construct. a single computer processor. A breakthrough is achieved on 1971.

Intel

Year	Computer Name	Description
1971	4004	• First microprocessor • All CPU components on a single chip • 4 bit
1972	8008	• 8 bit • Both designed for specific applications
1974	8080	• Intel's first general purpose microprocessor

Chapter 2. Machine Instructions and Programs

Objectives

- Machine instructions and program execution, including branching and subroutine call and return operations.
- Number representation and addition/subtraction in the 2’s-complement system.
- Addressing methods for accessing register and memory operands.
- Assembly language for representing machine instructions, data, and programs.
- Program-controlled Input/Output operations.

Instruction and Instruction Sequencing

“Must-Perform” Operations

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location (R1←[LOC], R3 ←[R1]+[R2])
- Register Transfer Notation (RTN)

Assembly Language Notation

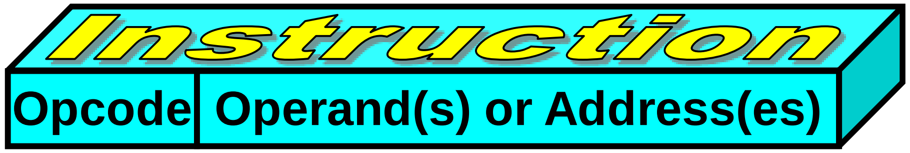
- Represent machine instructions and programs.
- Move LOC, R1 = R1←[LOC]
- Add R1, R2, R3 = R3 ←[R1]+[R2]

CPU Organization

- Single Accumulator
  - Result usually goes to the Accumulator
  - Accumulator has to be saved to memory quite often
- General Register
  - Registers hold operands thus reduce memory traffic
  - Register bookkeeping
- Stack
  - Operands and result are always in the stack

Instruction Formats

- Three-Address Instructions
  - ADD R1, R2, R3 — $R1 \leftarrow R2 + R3$
- Two-Address Instructions
  - ADD R1, R2 — $R1 \leftarrow R1 + R2$
- One-Address Instructions
  - ADD M — $AC \leftarrow AC + M[AR]$
- Zero-Address Instructions
  - ADD — $TOS \leftarrow TOS + (TOS - 1)$
- RISC Instructions
  - Lots of registers. Memory is restricted to Load & Store



Instruction Formats

Example: Evaluate (A+B) $*$ (C+D)

- Three-Address
  1. ADD R1, A, B ; R1 $\leftarrow$ M[A] + M[B]
  2. ADD R2, C, D ; R2 $\leftarrow$ M[C] + M[D]
  3. MUL X, R1, R2 ; M[X] $\leftarrow$ R1 $*$ R2

Instruction Formats

Example: Evaluate (A+B) ∗ (C+D)

- Two-Address

```
1.  MOV   R1, A          ; R1 ← M[A]
2.  ADD   R1, B          ; R1 ← R1 + M[B]
3.  MOV   R2, C          ; R2 ← M[C]
4.  ADD   R2, D          ; R2 ← R2 + M[D]
5.  MUL   R1, R2         ; R1 ← R1 ∗ R2
6.  MOV   X, R1          ; M[X] ← R1
```

Instruction Formats

Example: Evaluate $(A+B) * (C+D)$

- One-Address

1. LOAD A	; $AC \leftarrow M[A]$
2. ADD B	; $AC \leftarrow AC + M[B]$
3. STORE T	; $M[T] \leftarrow AC$
4. LOAD C	; $AC \leftarrow M[C]$
5. ADD D	; $AC \leftarrow AC + M[D]$
6. MUL T	; $AC \leftarrow AC * M[T]$
7. STORE X	; $M[X] \leftarrow AC$

Instruction Formats

Example: Evaluate (A+B) ∗ (C+D)

- Zero-Address

1.	PUSH A	; TOS ← A
2.	PUSH B	; TOS ← B
3.	ADD	; TOS ← (A + B)
4.	PUSH C	; TOS ← C
5.	PUSH D	; TOS ← D
6.	ADD	; TOS ← (C + D)
7.	MUL	; TOS ← (C+D)∗(A+B)
8.	POP X	; M[X] ← TOS

Instruction Formats

Example: Evaluate (A+B) $*$ (C+D)

- RISC

```
1. LOAD   R1, A           ; R1 ← M[A]
2. LOAD   R2, B           ; R2 ← M[B]
3. LOAD   R3, C           ; R3 ← M[C]
4. LOAD   R4, D           ; R4 ← M[D]
5. ADD    R1, R1, R2      ; R1 ← R1 + R2
6. ADD    R3, R3, R4      ; R3 ← R3 + R4
7. MUL    R1, R1, R3      ; R1 ← R1 * R3
8. STOREX, R1             ; M[X] ← R1
```

Using Registers

- Registers are faster
- Shorter instructions
  - The number of registers is smaller (e.g. 32 registers need 5 bits)
- Potential speedup
- Minimize the frequency with which data is moved back and forth between the memory and processor registers.

Instruction Execution and Straight-Line Sequencing

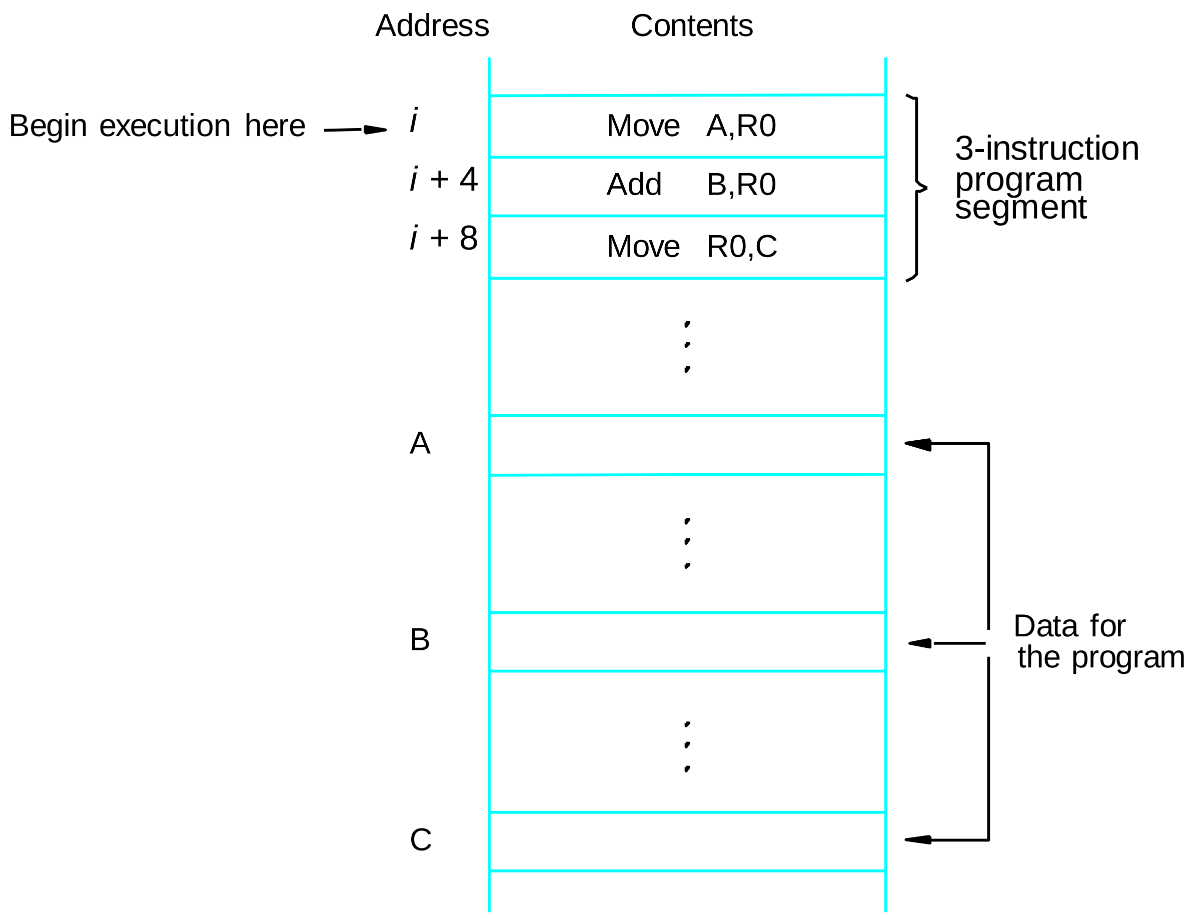


Figure 2.8. A program for C ← [A] + [B].

Assumptions:

- One memory operand per instruction
- 32-bit word length
- Memory is byte addressable
- Full memory address can be directly specified in a single-word instruction

Two-phase procedure

-Instruction fetch

-Instruction execute

Page 43

Branching

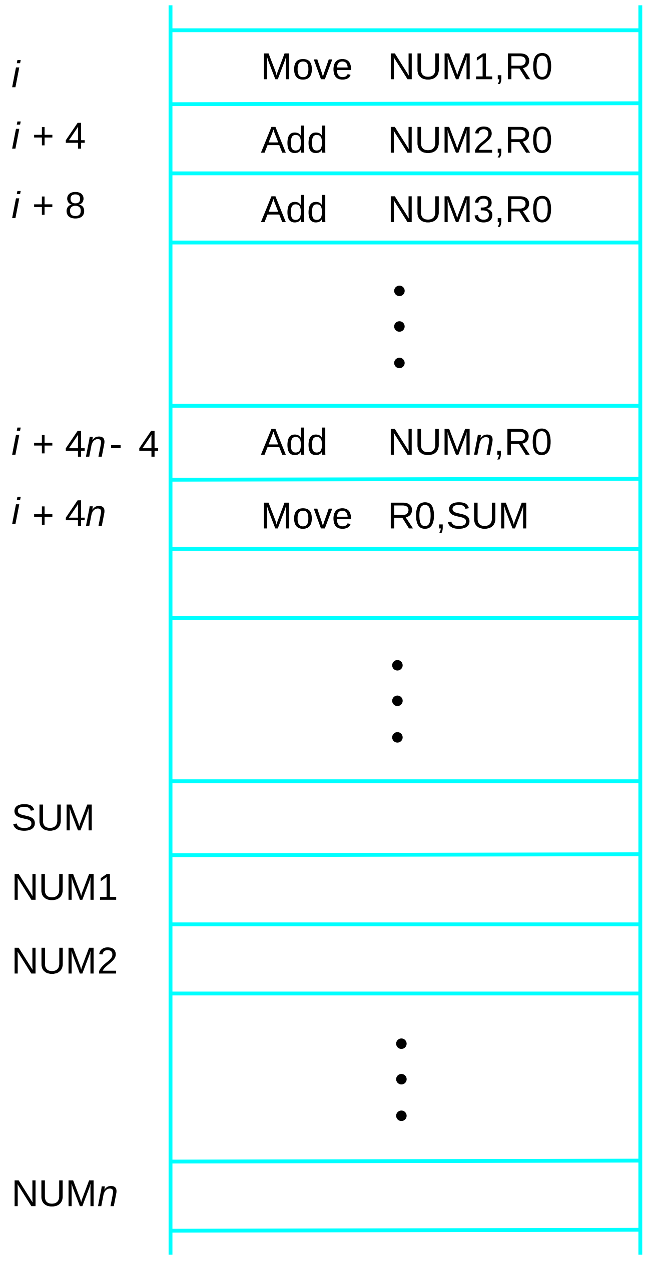


Figure 2.9. A straight-line program for adding *n* numbers.

Branching

Branch target

Conditional branch

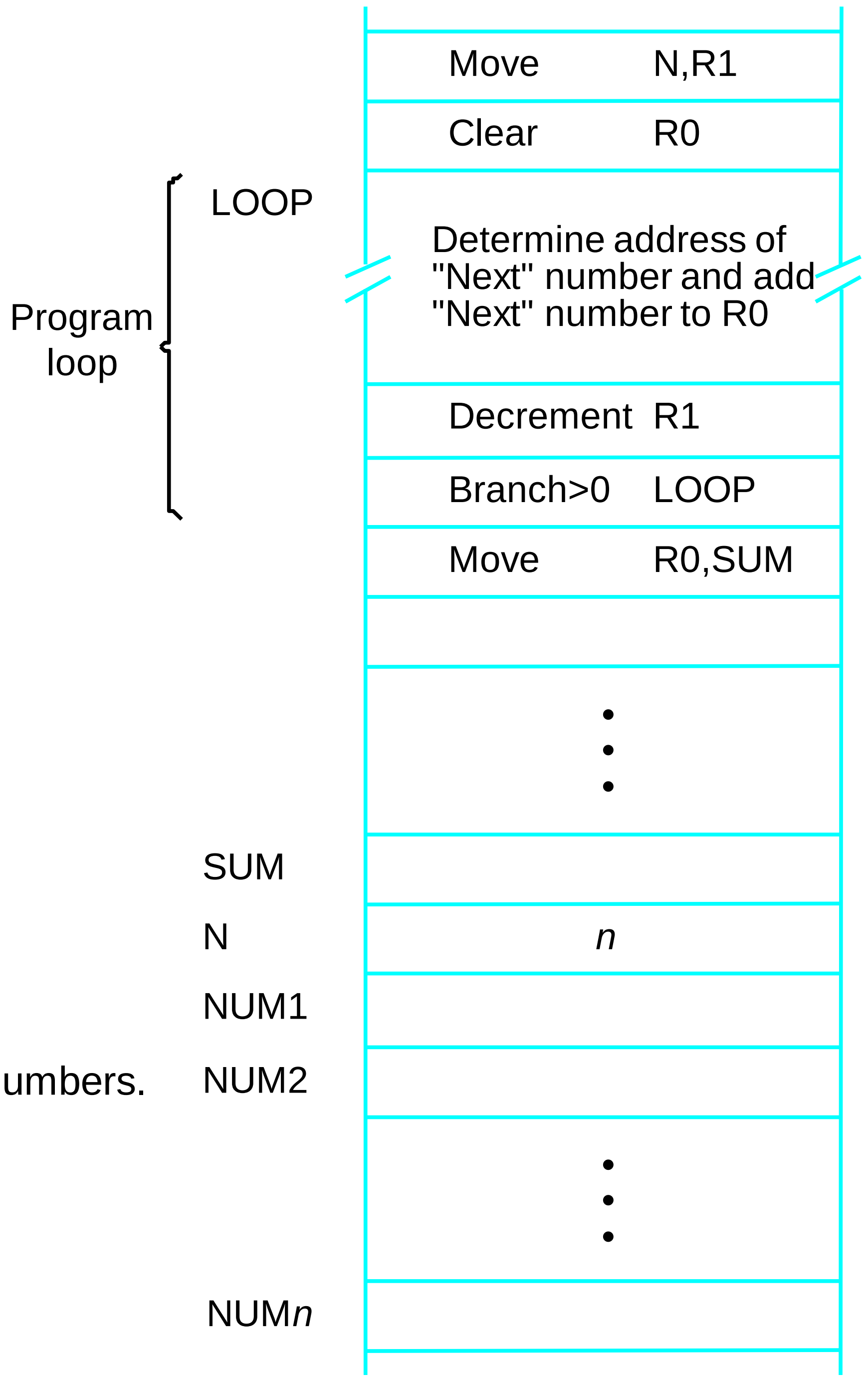


Figure 2.10. Using a loop to add *n* numbers.

Condition Codes

- Condition code flags
- Condition code register / status register
- N (negative)
- Z (zero)
- V (overflow)
- C (carry)
- Different instructions affect different flags

Conditional Branch Instructions

- Example:
  - A: 1 1 1 1 0 0 0 0
  - B: 0 0 0 1 0 1 0 0

```
A:      1 1 1 1 0 0 0 0
+(−B):  1 1 1 0 1 1 0 0
-----------------------
        1 1 0 1 1 1 0 0
```

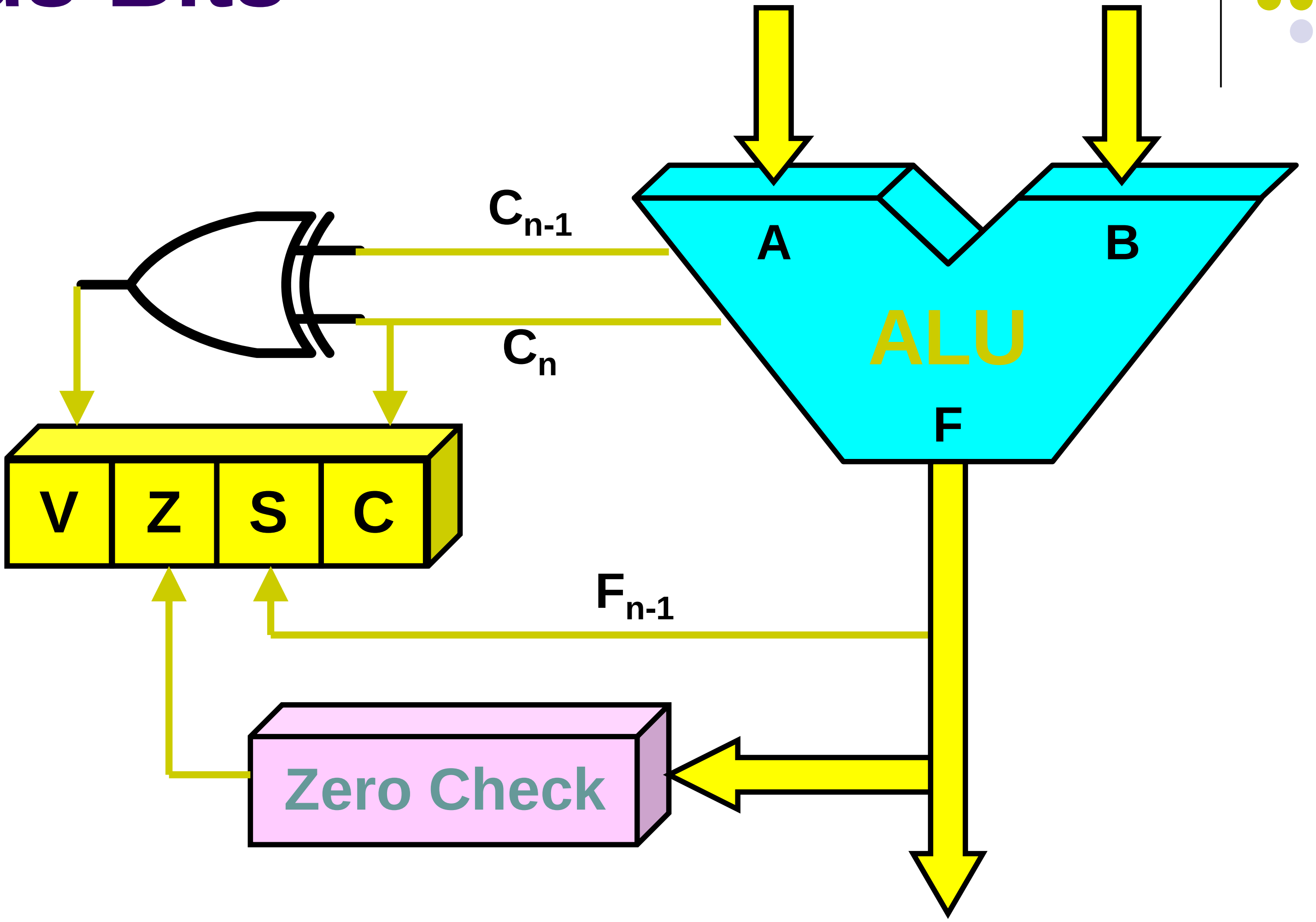
C = 1

Z = 0

S = 1

V = 0

Status Bits



THANK YOU

COMPUTER ARCHITECTURE AND ORGANIZATION.

UNIT-II

Presented by
Dr.K.RAJKAMAL,
Assoc.Prof,
Dept of ECE,
KHIT.

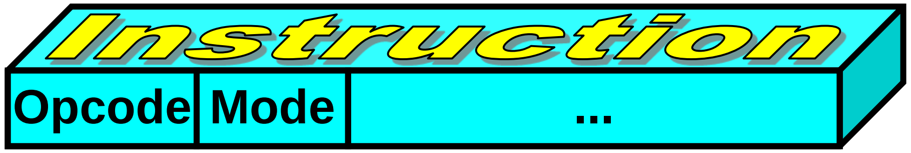
UNIT-II

Addressing Modes

Generating Memory Addresses

- How to specify the address of branch target?
- Can we give the memory operand address directly in a single Add instruction in the loop?
- Use a register to hold the address of NUM1; then increment by 4 on each pass through the loop.

Addressing Modes

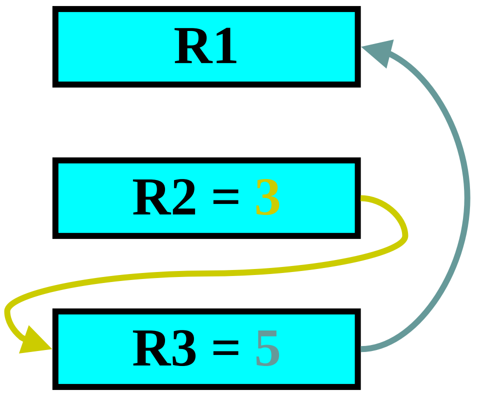


- Implied
  - AC is implied in "ADD M[AR]" in "One-Address" instr.
  - TOS is implied in "ADD" in "Zero-Address" instr.
- Immediate
  - The use of a constant in "MOV R1, 5", i.e. R1 ← 5
- Register
  - Indicate which register holds the operand

Addressing Modes

- Register Indirect
  - Indicate the register that holds the number of the register that holds the operand

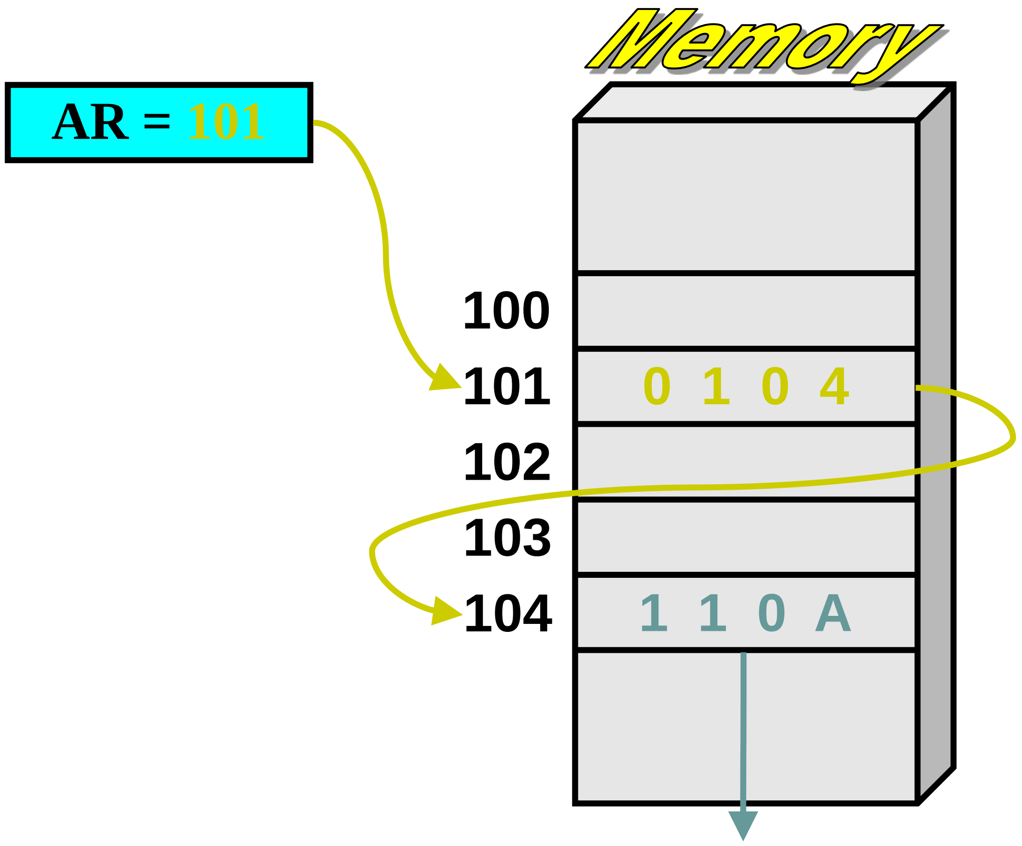
    MOV R1, (R2)



- Autoincrement / Autodecrement
  - Access & update in 1 instr.
- Direct Address
  - Use the given address to access a memory location

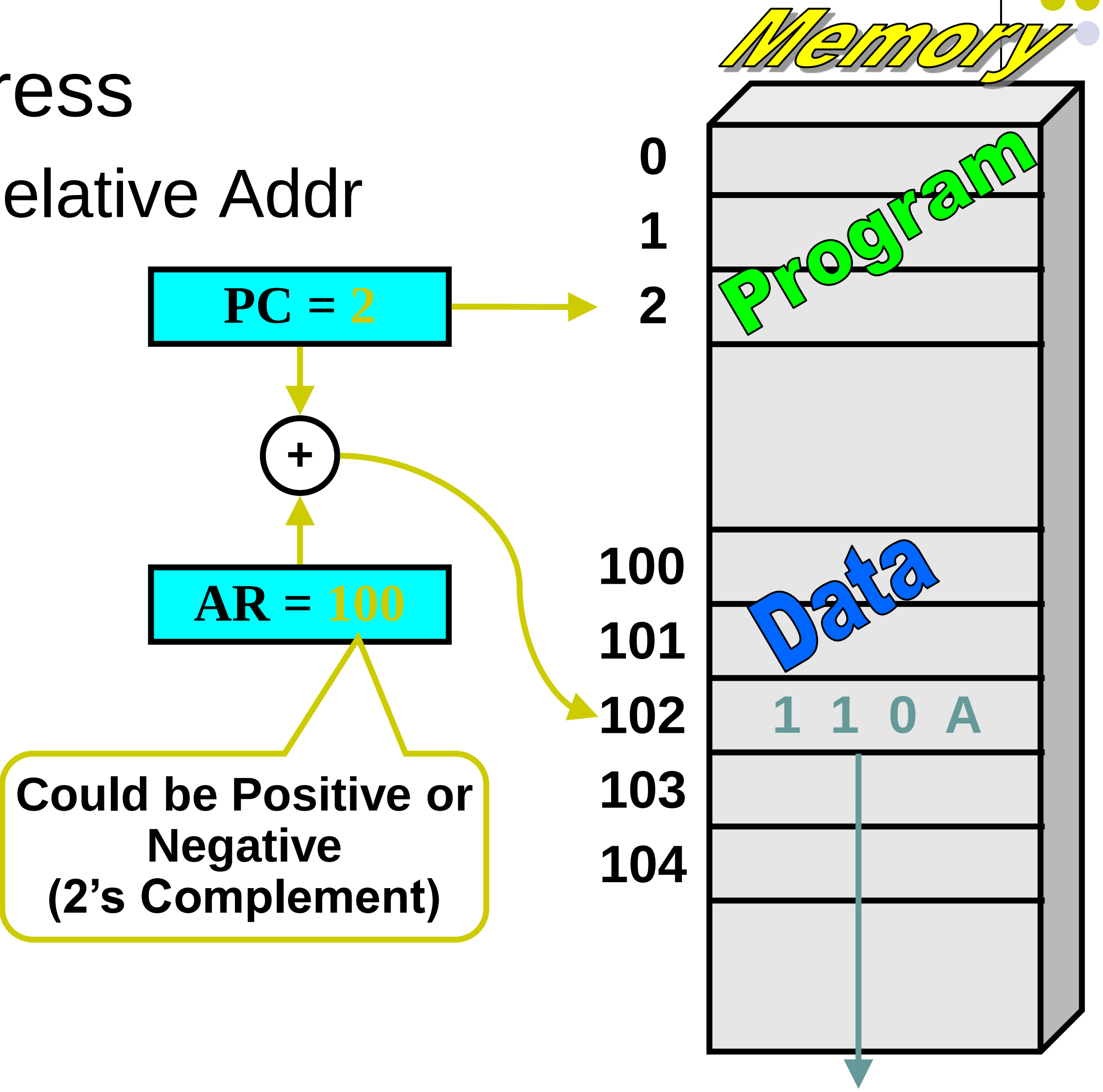
Addressing Modes

- Indirect Address
    - Indicate the memory location that holds the address of the memory location that holds the data



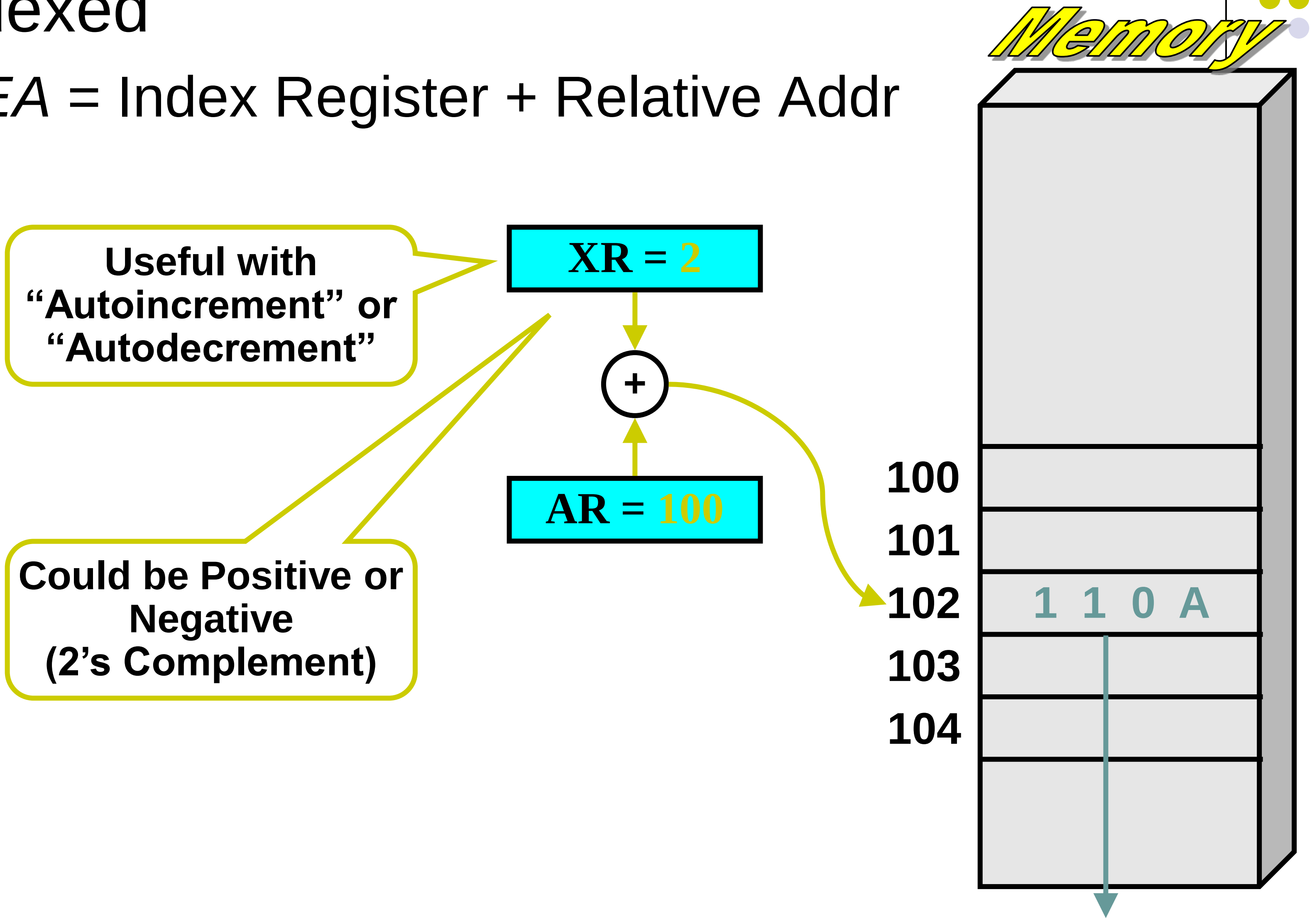
Addressing Modes

- Relative Address
  - *EA* = PC + Relative Addr



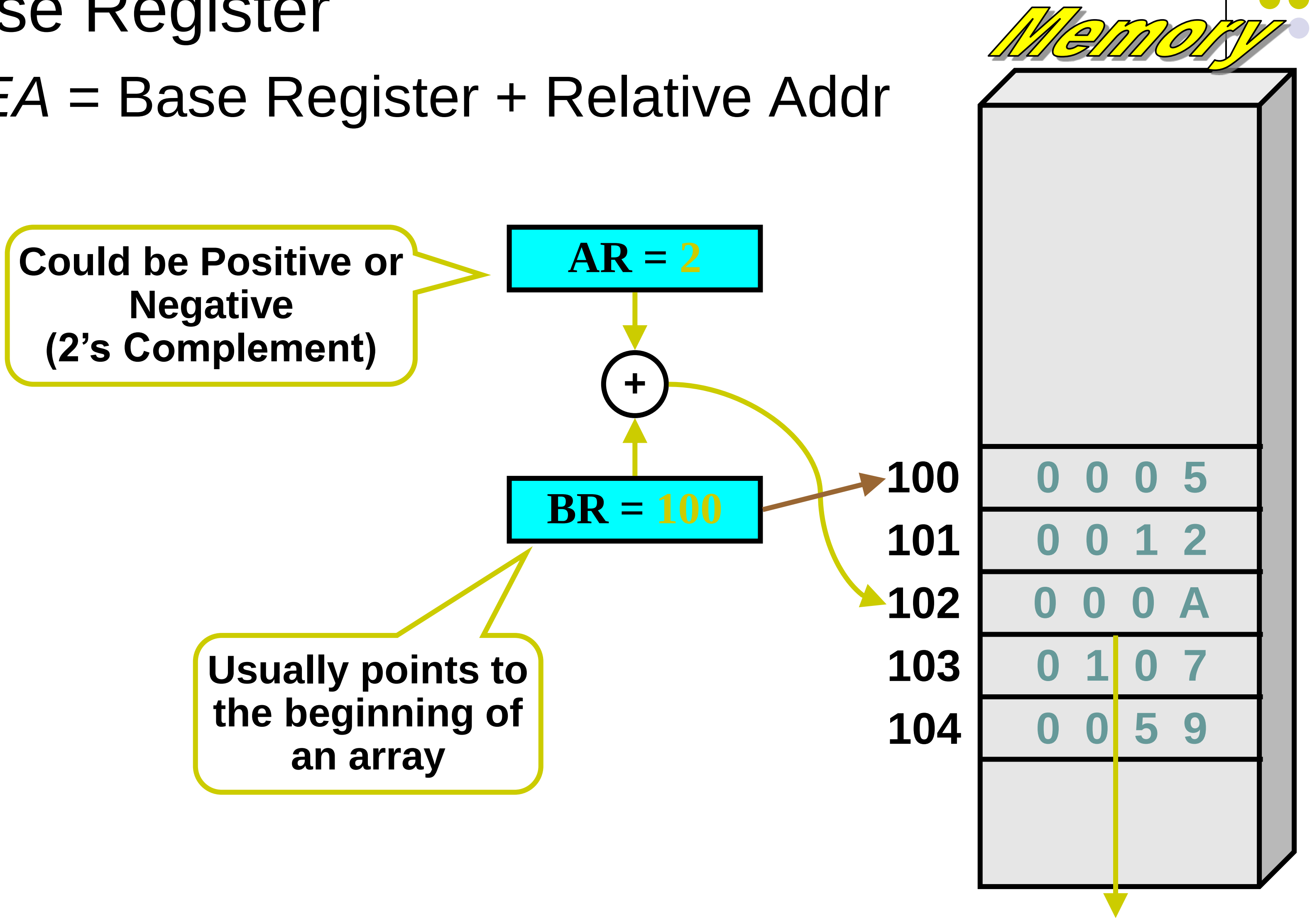
Addressing Modes

- Indexed
  - *EA* = Index Register + Relative Addr


Memory
Useful with "Autoincrement" or "Autodecrement"
Could be Positive or Negative (2's Complement)
XR = 2
AR = 100
+
100
101
102
103
104
1 1 0 A

Addressing Modes

- Base Register
  - *EA* = Base Register + Relative Addr


Memory
Could be Positive or Negative (2's Complement)
AR = 2
+
BR = 100
Usually points to the beginning of an array
100 0 0 0 5
101 0 0 1 2
102 0 0 0 A
103 0 1 0 7
104 0 0 5 9

Addressing Modes

- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

Name	Assem bler syn tax	Addressing function
Immediate	#V alue	Op erand = Value
Register	R *i*	EA = R *i*
Absolute (Direct)	LOC	EA = LOC

Indirect	(R *i*)
	(LOC)
		EA = [R *i*]
		EA = [LOC]

Index	X(R *i*)	EA = [R *i*] + X
Base with index	(R *i* ,R *j*)	EA = [R *i*] + [R *j*]
Base with index and offset	X(R *i* ,R *j*)	EA = [R *i*] + [R *j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincremen t	(R *i*)+	EA = [R *i*] ; Incremen t R *i*
Autodecrement	−(R *i*)	Decremen t R *i* ; EA = [R *i*]

Indexing and Arrays

- Index mode – the effective address of the operand is generated by adding a constant value to the contents of a register.
- Index register
- $X(R_i)$: $EA = X + [R_i]$
- The constant X may be given either as an explicit number or as a symbolic name representing a numerical value.
- If X is shorter than a word, sign-extension is needed.

Indexing and Arrays

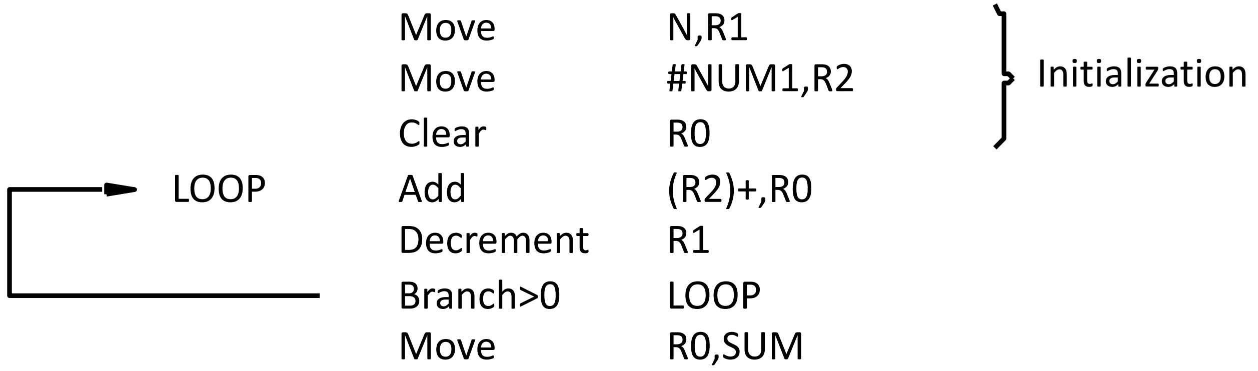
- In general, the Index mode facilitates access to an operand whose location is defined relative to a reference point within the data structure in which the operand appears.
- Several variations:
  (R_i, R_j): $EA = [R_i] + [R_j]$
  $X(R_i, R_j)$: $EA = X + [R_i] + [R_j]$

Relative Addressing

- Relative mode – the effective address is determined by the Index mode using the program counter in place of the general-purpose register.
- X(PC) – note that X is a signed number
- Branch>0 LOOP
- This location is computed by specifying it as an offset from the current value of PC.
- Branch target may be either before or after the branch instruction, the offset is given as a singed num.

Additional Modes

- Autoincrement mode – the effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
- $(R_i)+$. The increment is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands.
- Autodecrement mode: $-(R_i)$ – decrement first



```
                Move          N,R1        }
                Move          #NUM1,R2    }  Initialization
                Clear         R0          }
      LOOP      Add           (R2)+,R0
                Decrement     R1
                Branch>0      LOOP
                Move          R0,SUM
```

Figure 2.16. The Autoincrement addressing mode used in the program of Figure 2.12.

Basic Input / Output Operations

I/O

- The data on which the instructions operate are not necessarily already stored in memory.
- Data need to be transferred between processor and outside world (disk, keyboard, etc.)
- I/O operations are essential, the way they are performed can have a significant effect on the performance of the computer.

Program-Controlled I/O Example

- Read in character input from a keyboard and produce character output on a display screen.
  - Rate of data transfer (keyboard, display, processor)
  - Difference in speed between processor and I/O device creates the need for mechanisms to synchronize the transfer of data.
  - A solution: on output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character. Input is sent from the keyboard in a similar way.

Program-Controlled I/O Example

- Registers
- Flags
- Device interface

Program-Controlled I/O Example

- Machine instructions that can check the state of the status flags and transfer data:

```
READWAIT  Branch to READWAIT if SIN = 0
          Input from DATAIN to R1

WRITEWAIT Branch to WRITEWAIT if SOUT = 0
          Output from R1 to DATAOUT
```

Program-Controlled I/O Example

- Memory-Mapped I/O – some memory address values are used to refer to peripheral device buffer registers. No special instructions are needed. Also use device status registers.

```
READWAIT  Testbit   #3, INSTATUS
          Branch=0  READWAIT
          MoveByte  DATAIN, R1
```

Program-Controlled I/O Example

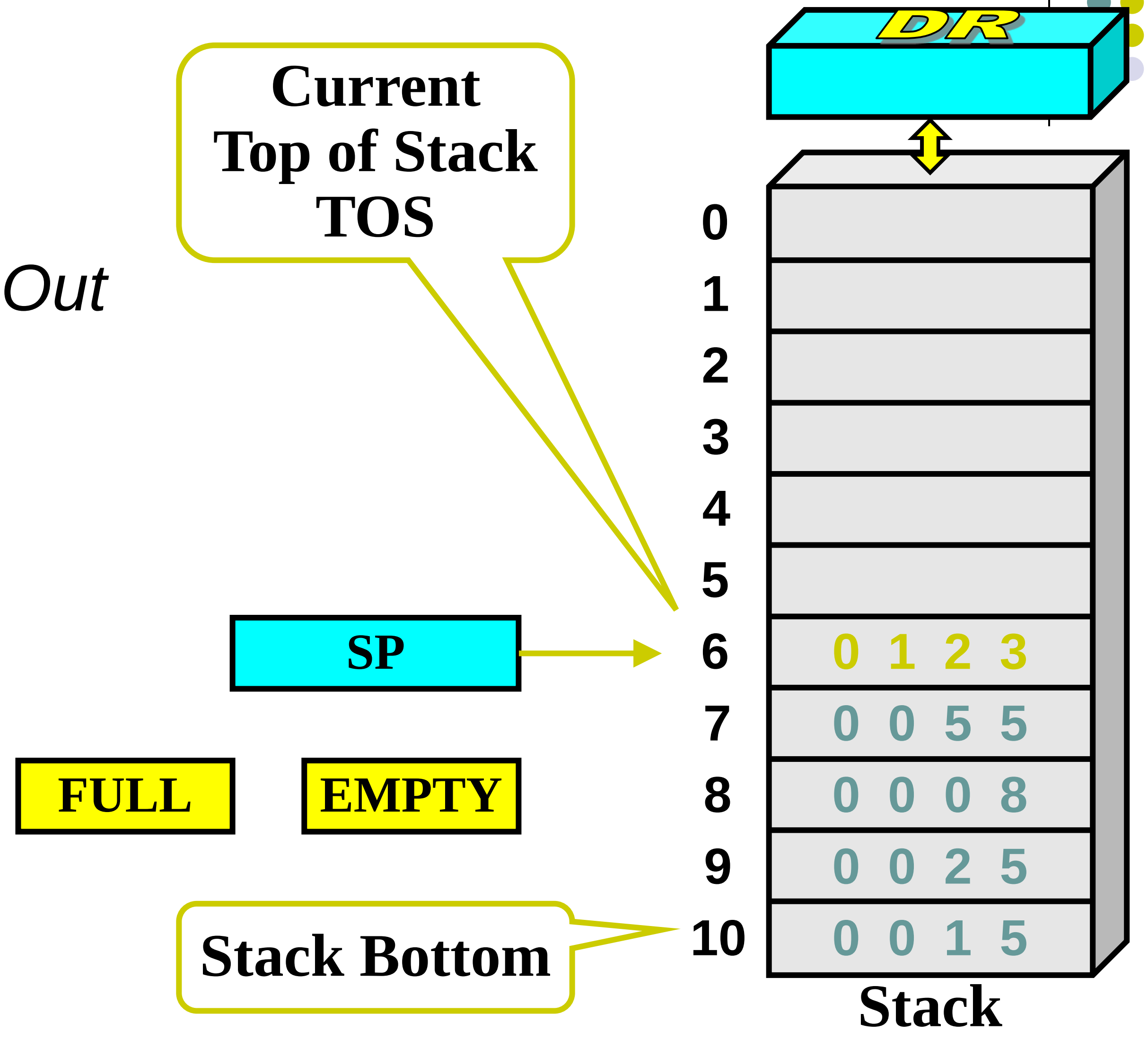
- Assumption – the initial state of SIN is 0 and the initial state of SOUT is 1.
- Any drawback of this mechanism in terms of efficiency?
  - Two wait loops→processor execution time is wasted
- Alternate solution?
  - Interrupt

Stacks & Queues

Stack Organization

- LIFO

  Last In First Out



Stack Organization

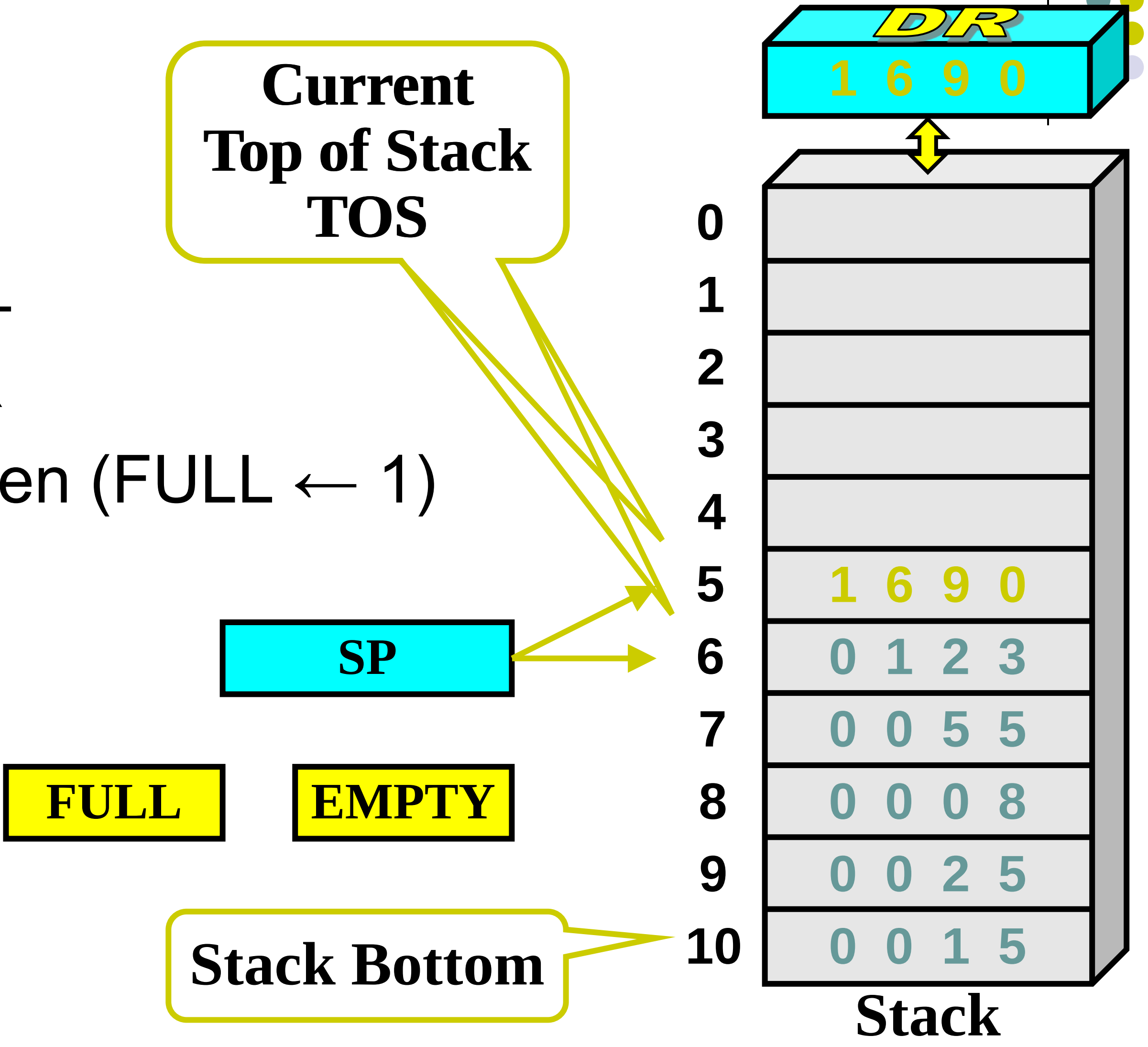
- PUSH

  SP ← SP – 1

  M[SP] ← DR

  If (SP = 0) then (FULL ← 1)

  EMPTY ← 0



Stack Organization

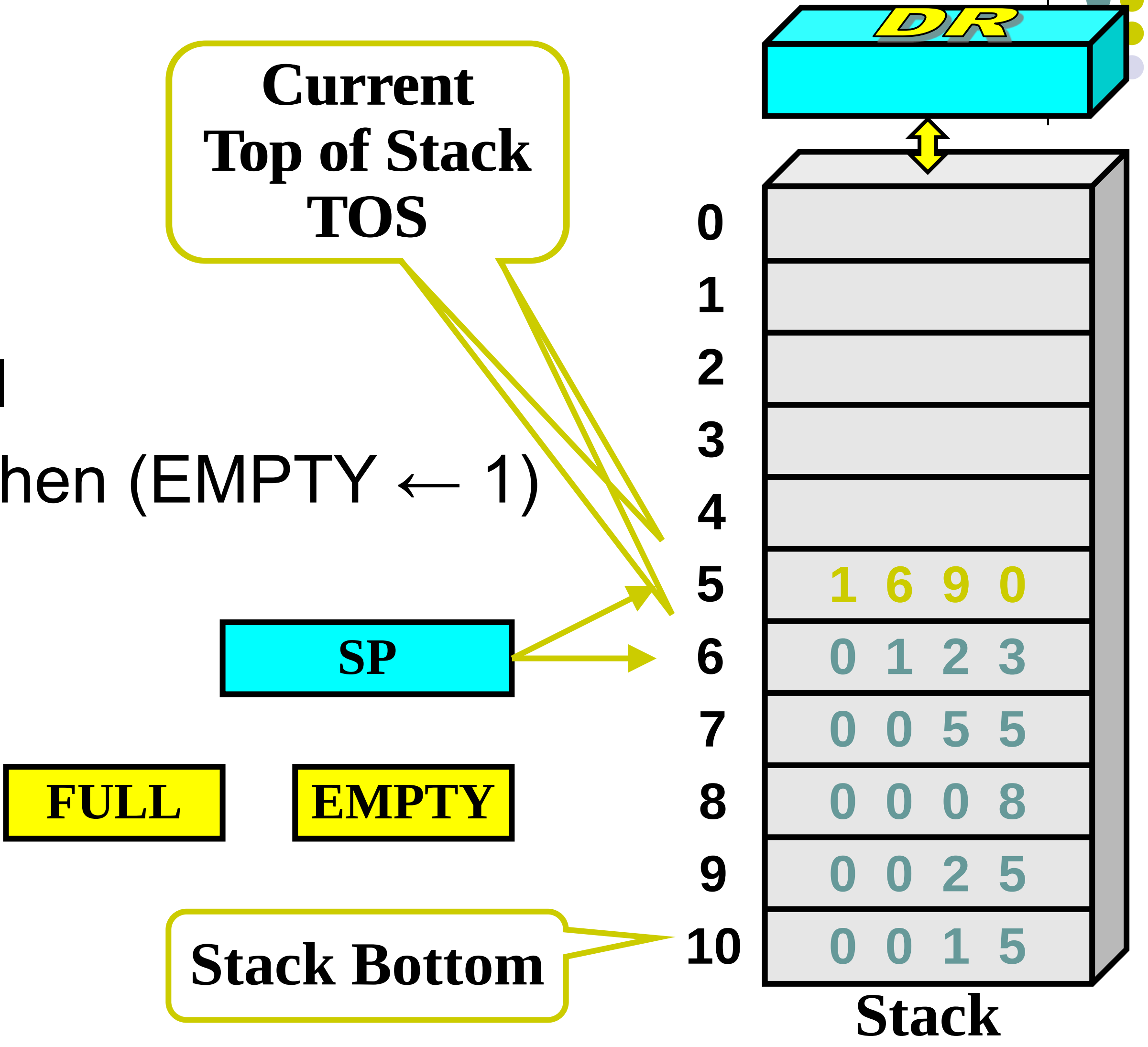
- POP

  DR ← M[SP]

  SP ← SP + 1

  If (SP = 11) then (EMPTY ← 1)

  FULL ← 0



Stack Organization

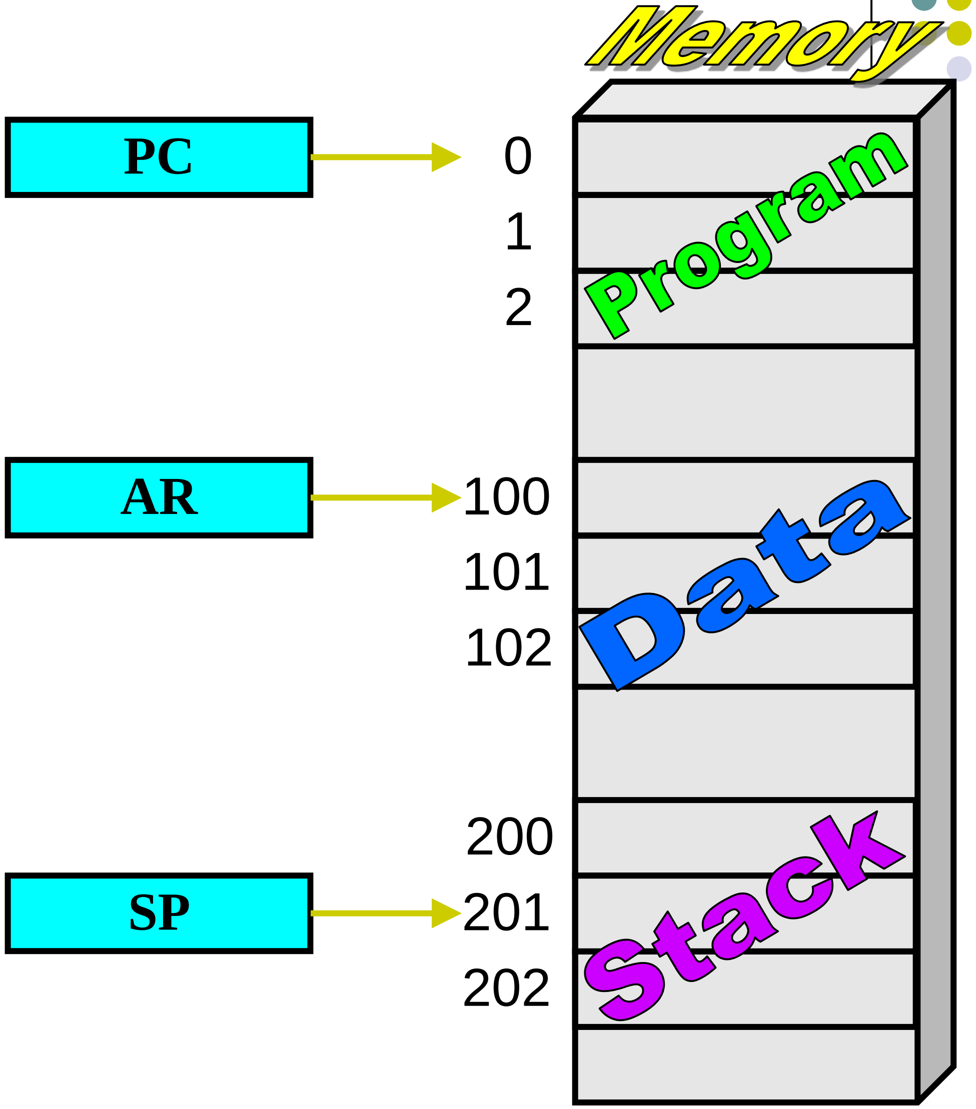
- Memory Stack
  - PUSH

    SP ← SP – 1

    M[SP] ← DR
  - POP

    DR ← M[SP]

    SP ← SP + 1



Reverse Polish Notation

- Infix Notation

  A + *B*

- Prefix or Polish Notation

  + *A* *B*

- Postfix or Reverse Polish Notation (RPN)

  A *B* +

$$A * B + C * D \xrightarrow{\text{RPN}} A\ B * C\ D * +$$

(2) (4) * (3) (3) * +

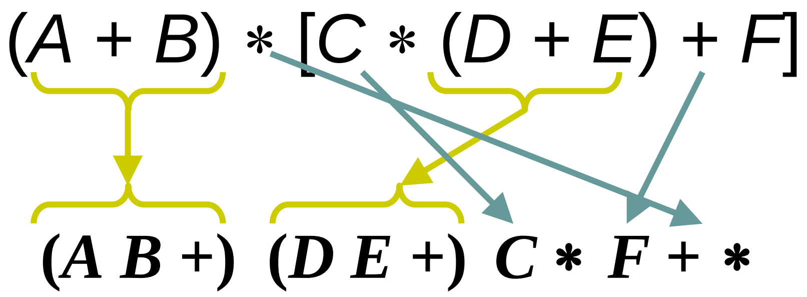
(8) (3) (3) * +

(8) (9) +

17

Reverse Polish Notation

- Example



$(A + B) * [C * (D + E) + F]$

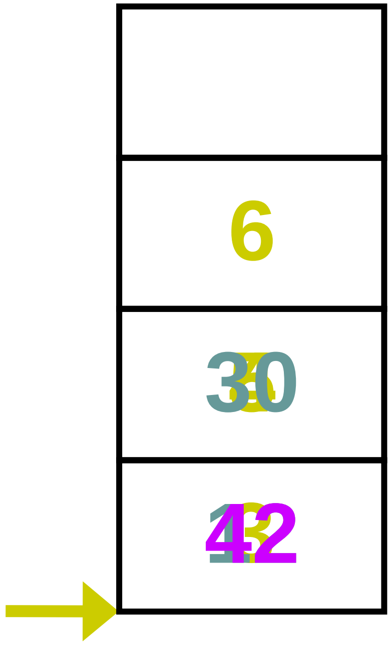
$(A\ B\ +)\ (D\ E\ +)\ C\ *\ F\ +\ *$

Reverse Polish Notation

- Stack Operation

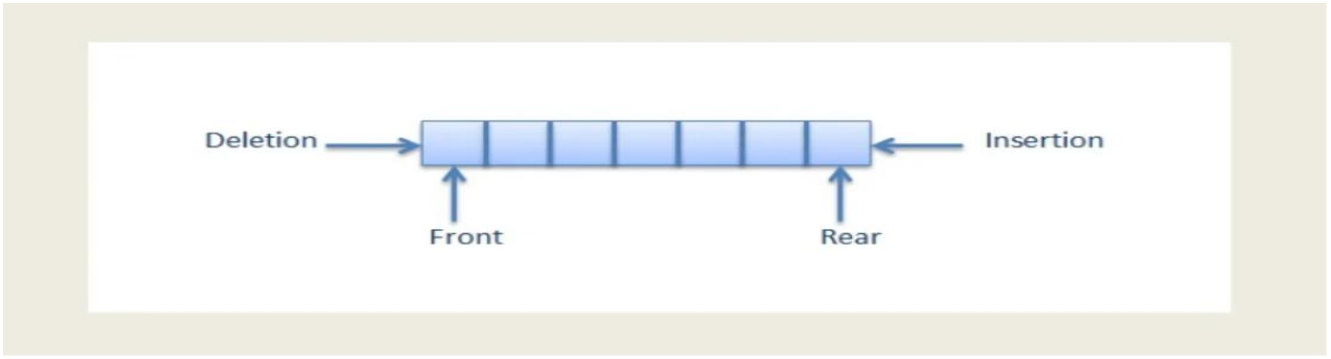
  (3) (4) * (5) (6) * +

PUSH	**3**
PUSH	**4**
MULT	
PUSH	**5**
PUSH	**6**
MULT	
ADD	



Queues

- Queue is an ADT data structure similar to stack, except that the first item to be inserted is the first one to be removed.
- This mechanism is called First-In-First-Out (FIFO).
- Placing an item in a queue is called “insertion or enqueue”, which is done at the end of the queue called “rear”.
- Removing an item from a queue is called “deletion or dequeue”, which is done at the other end of the queue called “front”.
- Some of the applications are : printer queue, keystroke queue, etc.



The Queue Operation

Placing an item in a queue is called "insertion or enqueue", which is done at the end of the queue called "rear".

Operations On A Queue

1. To insert an element in queue
2. Delete an element from queue

Algorithm QINSERT (ITEM)

```
1.If (rear = maxsize-1 )
        print ("queue overflow") and return
2.Else
    rear = rear + 1
     Queue [rear] = item
```

Algorithm QDELETE ()

```
1.If  (front =rear)
        print "queue empty" and return
2. Else
        Front = front + 1
        item = queue [front];
        Return item
```

Queue Applications

- Real life examples
  - ✓Waiting in line
  - ✓Waiting on hold for tech support
- Applications related to Computer Science
  - ✓Round robin scheduling
  - ✓Job scheduling (FIFO Scheduling)
  - ✓Key board buffer

3 states of the queue

1. Queue is empty

   FRONT=REAR

2. Queue is full

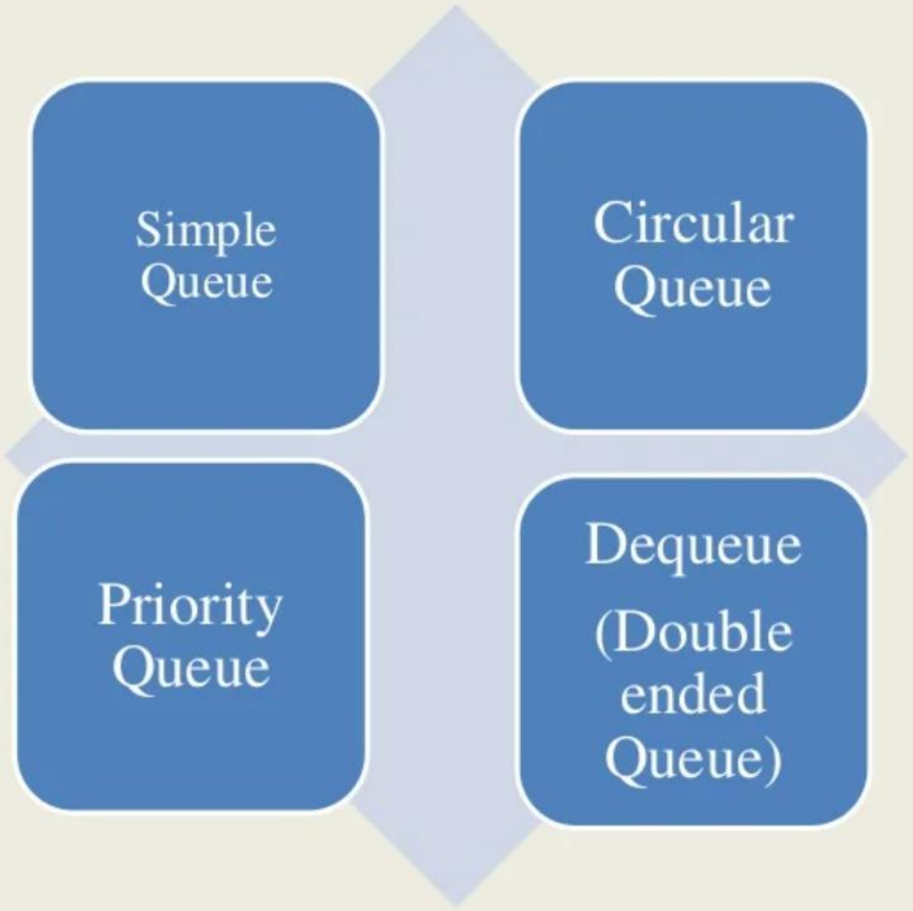
   REAR=N

3. Queue contains element >=1

   FRONT<REAR

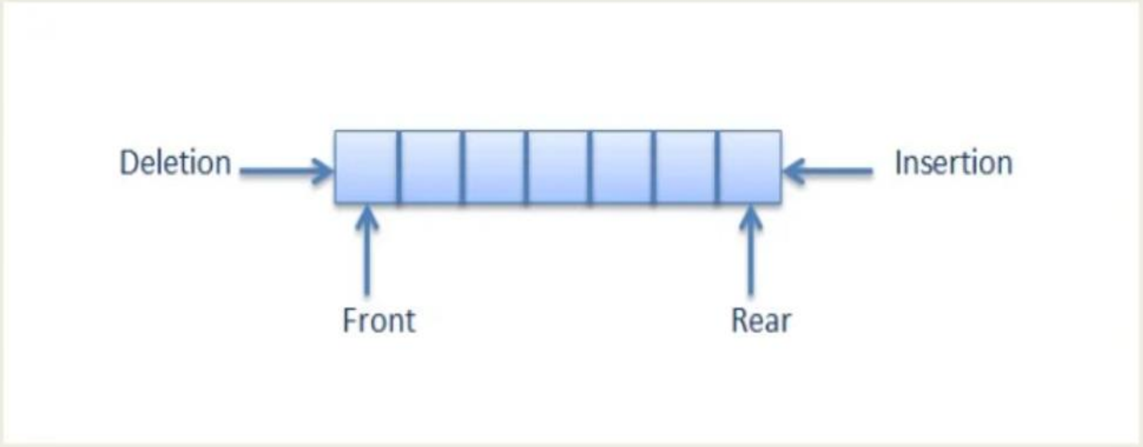
   NO. OF ELEMENT=REAR-FRONT+1

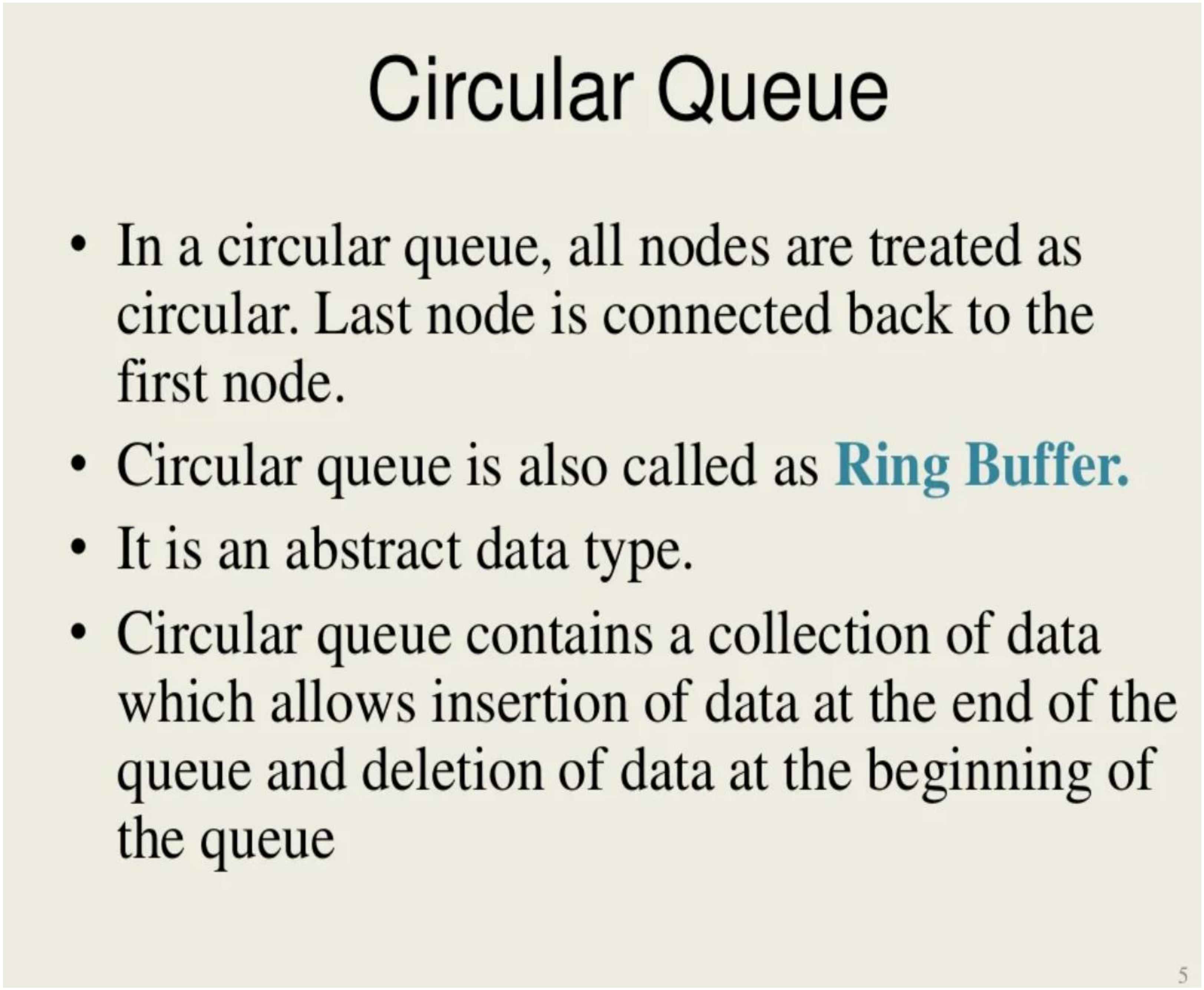
Type of queue



Simple Queue

- Simple queue defines the simple operation of queue in which insertion occurs at the rear of the list and deletion occurs at the front of the list.





Circular Queue

- In a circular queue, all nodes are treated as circular. Last node is connected back to the first node.
- Circular queue is also called as **Ring Buffer.**
- It is an abstract data type.
- Circular queue contains a collection of data which allows insertion of data at the end of the queue and deletion of data at the beginning of the queue

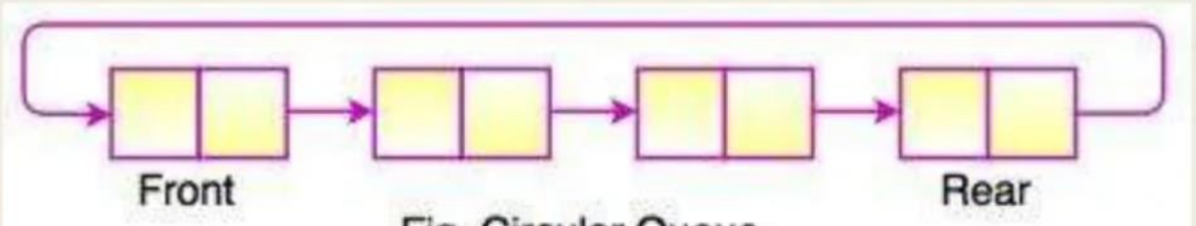


Fig. Circular Queue

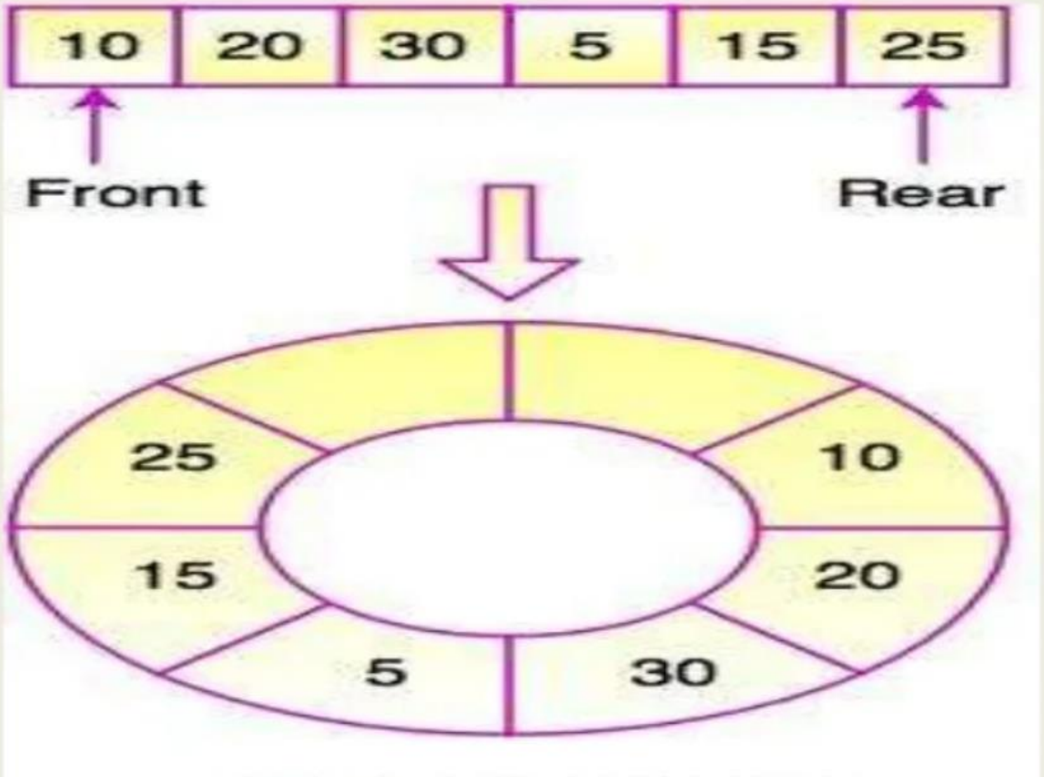
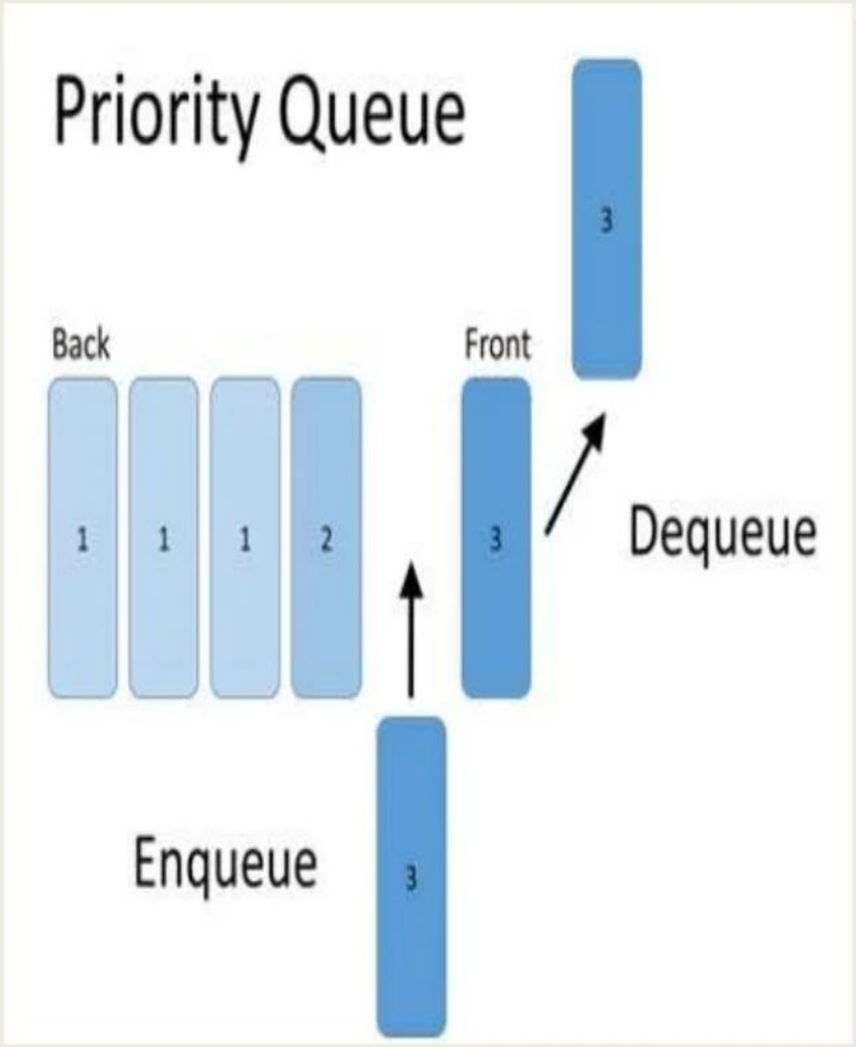


Fig. Circular Queue

Priority Queue

- Priority queue contains data items which have some preset priority. While removing an element from a priority queue, the data item with the highest priority is removed first.
- In a priority queue, insertion is performed in the order of arrival and deletion is performed based on the priority.



Dequeue (Double ended Queue)

- In Double Ended Queue, insert and delete operation can be occur at both ends that is front and rear of the queue

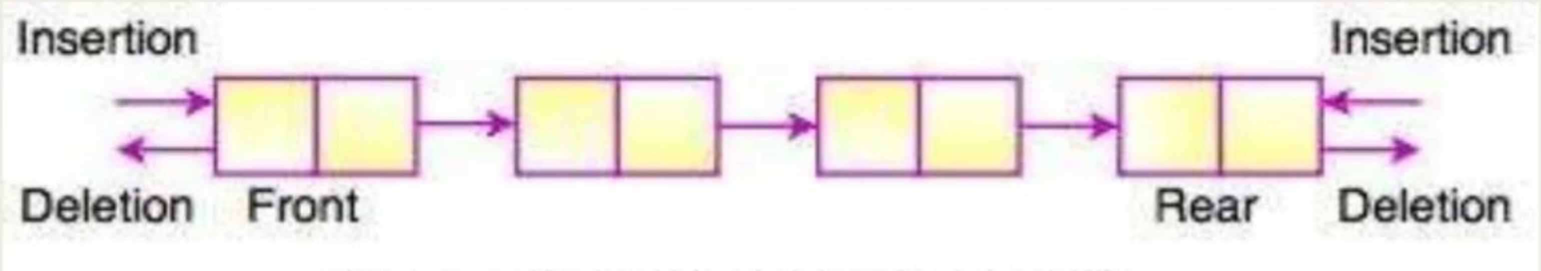
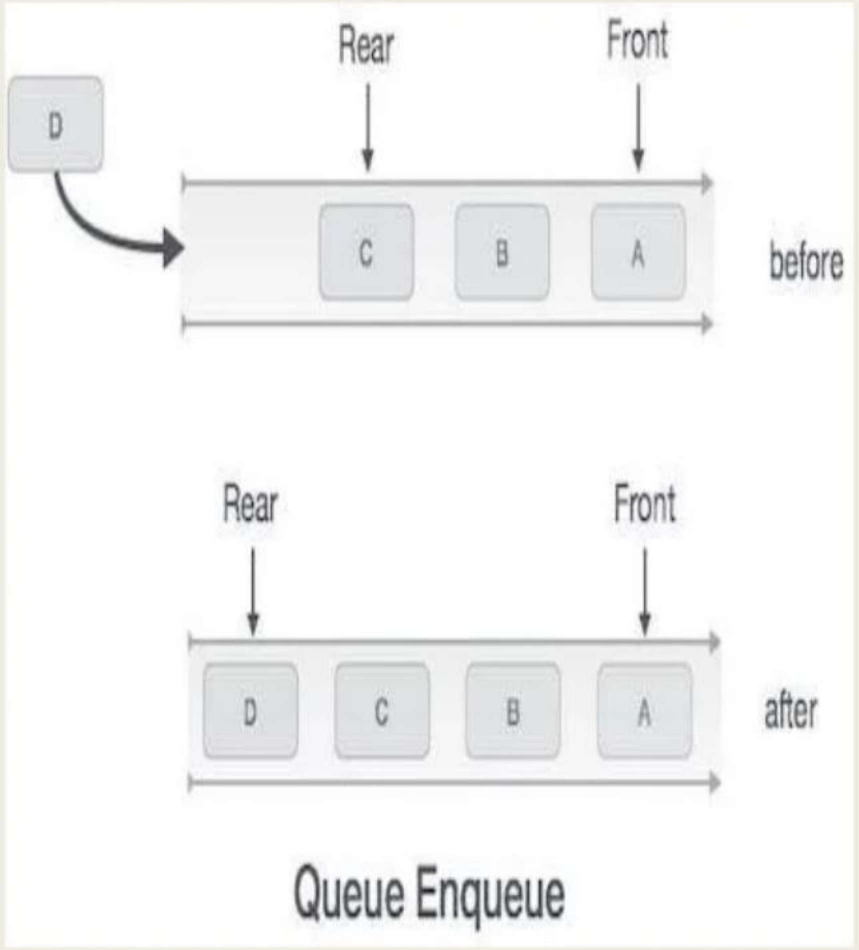


Fig. Double Ended Queue (Dequeue)

Enqueue Operation

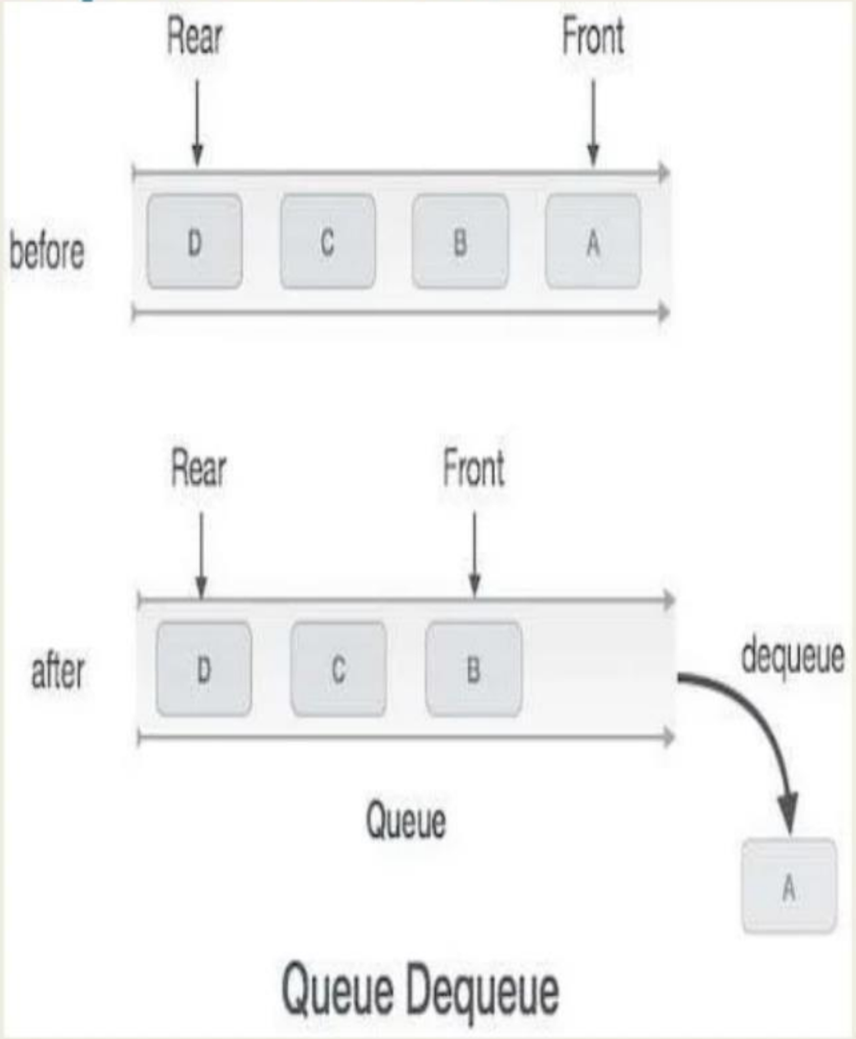
- **Step 1** – Check if the queue is full.
- **Step 2** – If the queue is full, produce overflow error and exit.
- **Step 3** – If the queue is not full, increment **rear** pointer to point the next empty space.
- **Step 4** – Add data element to the queue location, where the rear is pointing.
- **Step 5** – return success.



Queue Enqueue

Dequeue Operation

- **Step 1** – Check if the queue is empty.
- **Step 2** – If the queue is empty, produce underflow error and exit.
- **Step 3** – If the queue is not empty, access the data where **front** is pointing.
- **Step 4** – Increment **front** pointer to point to the next available data element.
- **Step 5** – Return success.



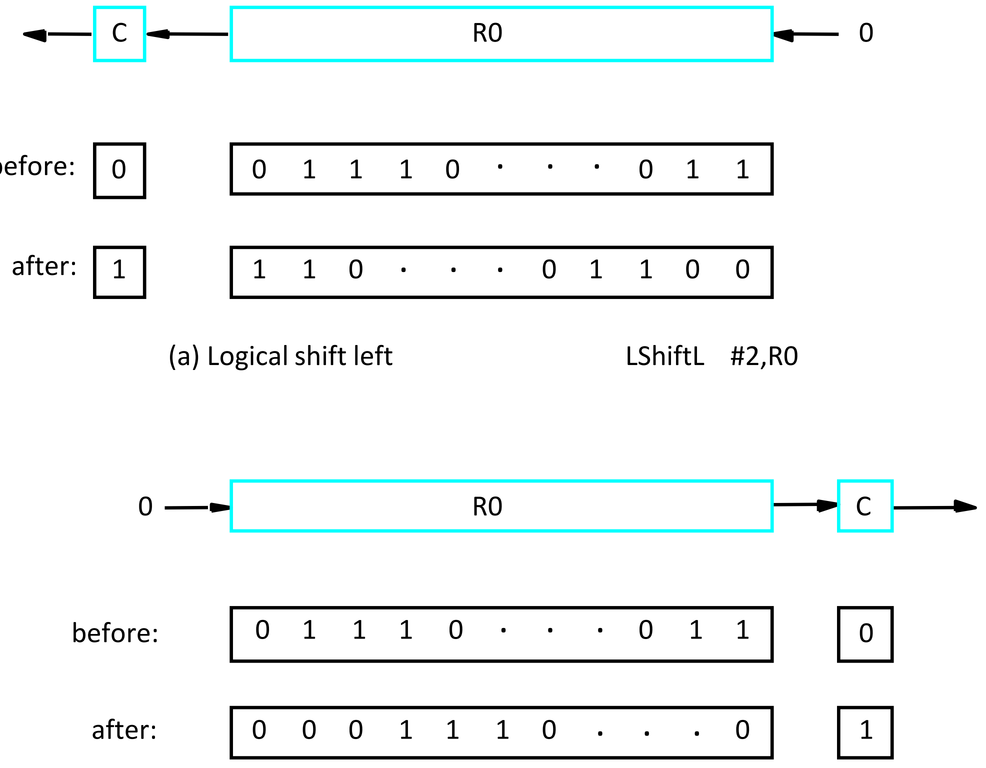
Component of Instructions

Logic
Instructions,
shift and Rotate
Instructions

Additional Instructions

Logical Shifts

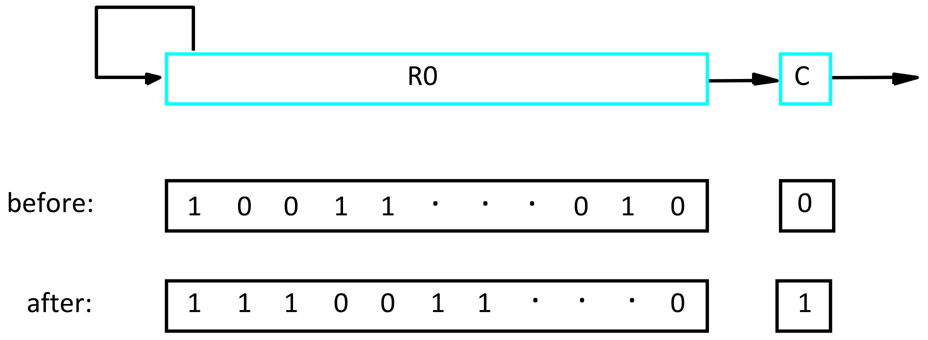
- Logical shift – shifting left (LShiftL) and shifting right (LShiftR)



(a) Logical shift left LShiftL #2,R0

(b) Logical shift r ight LShiftR #2,R0

Arithmetic Shifts



(c) Ar ithmetic shift right AShiftR #2,R0

Rotate

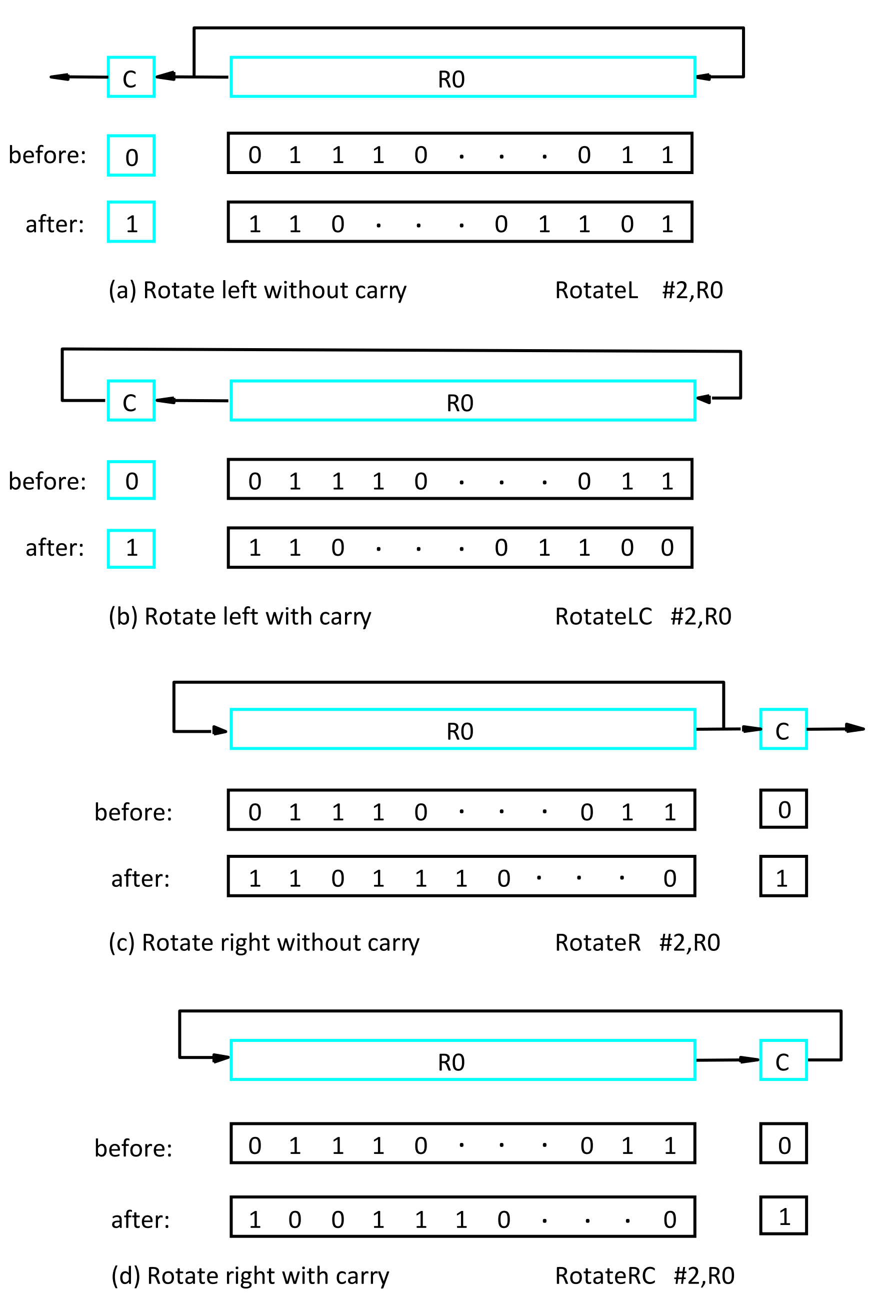


Figure 2.32. Rotate instructions.

Rotate

- Each bit in the accumulator can be shifted either left or right to the next position.

Opcode	Operand	Description
RLC	None	Rotate accumulator left

➢Each binary bit of the accumulator is rotated left by one position.

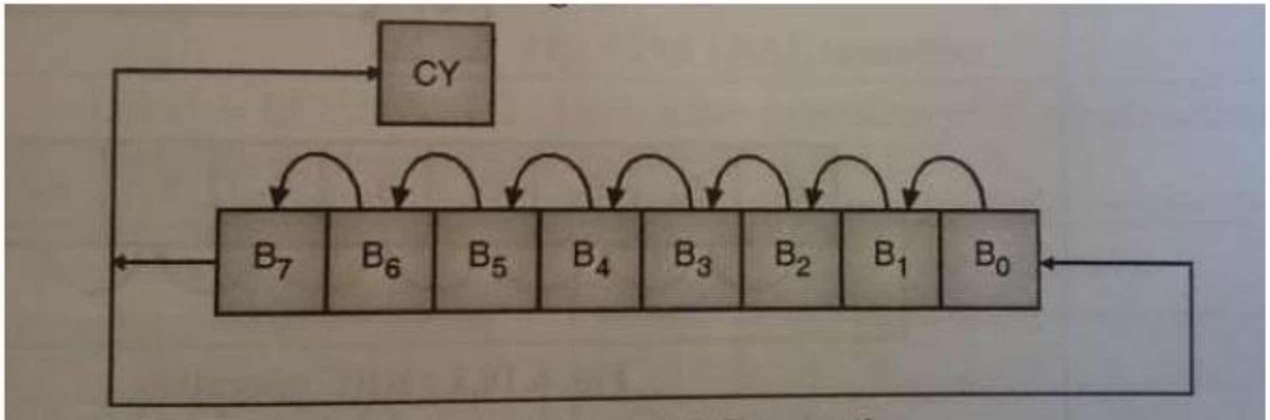


Fig. 6.17.1 : RLC operation

- Bit D7 is placed in the position of D0 as well as in the Carry flag.
- CY is modified according to bit D7.
- S, Z, P, AC are not affected.

Example: RLC.

Opcode	Operand	Description
RRC	None	Rotate accumulator right

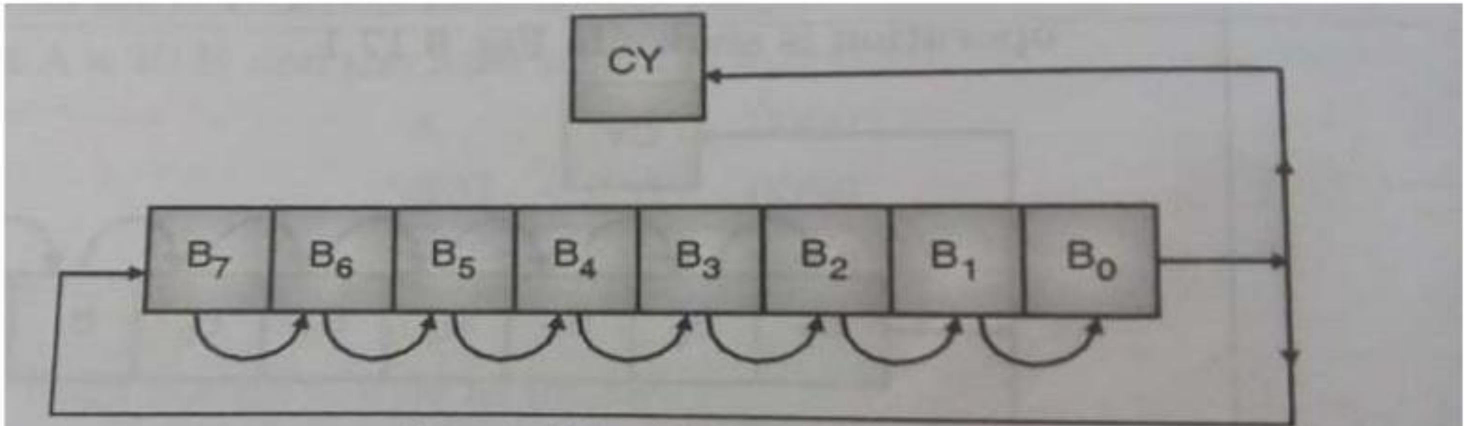


Fig. 6.18.1 : RRC operation

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

Example: RRC.

Opcode	Operand	Description
RAL	None	Rotate accumulator left through carry

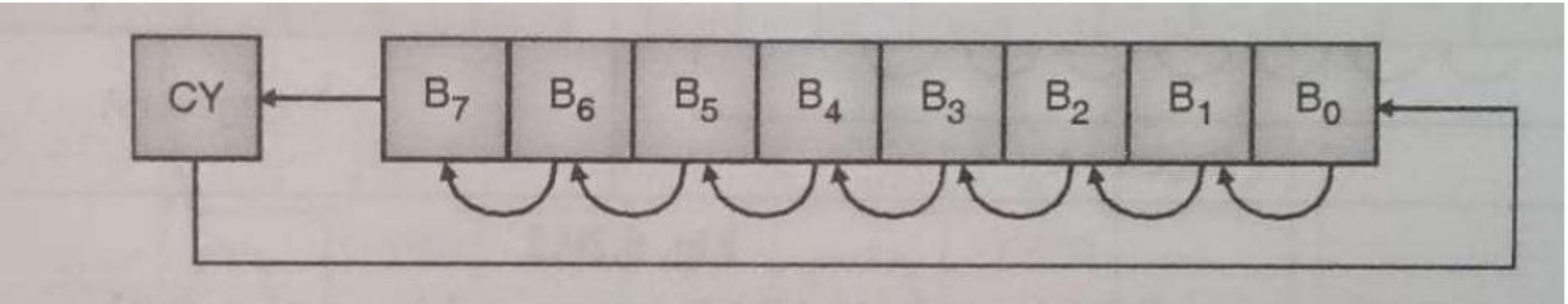


Fig. 6.19.1 : RAL operation

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

Example: RAL.

Opcode	Operand	Description
RAR	None	Rotate accumulator right through carry

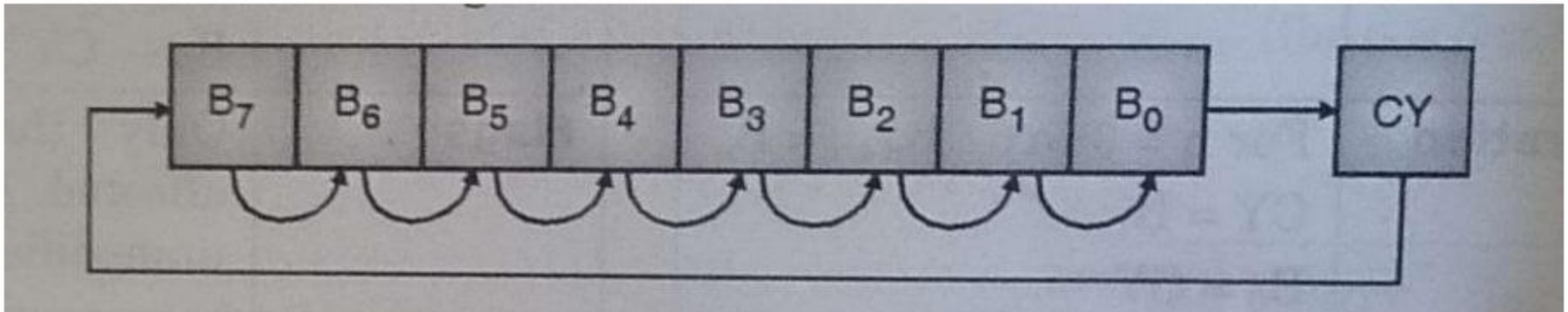


Fig. 6.20.1 : RAR operation

- ➢Each binary bit of the accumulator is rotated right by one position through the Carry flag.
- ➢ Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7.
- ➢CY is modified according to bit D0.
- ➢S, Z, P, AC are not affected.

Example: RAR.

Complement

- The contents of accumulator can be complemented.
- Each 0 is replaced by 1 and each 1 is replaced by 0.

Opcode	Operand	Description
CMA	None	Complement accumulator

- The contents of the accumulator are complemented.
- No flags are affected.

Example: CMA.

Opcode	Operand	Description
CMC	None	Complement carry

- The Carry flag is complemented.
- No other flags are affected.
- **Example: CMC.**

Opcode	Operand	Description
STC	None	Set carry

- The Carry flag is set to 1.
- No other flags are affected.
- **Example: STC.**

Multiplication and Division

- Not very popular (especially division)
- Multiply R_i, R_j
  $R_j \leftarrow [R_i] \times [R_j]$
- 2n-bit product case: high-order half in R(j+1)
- Divide R_i, R_j
  $R_j \leftarrow [R_i] / [R_j]$
  Quotient is in Rj, remainder may be placed in R(j+1)

Type of Instructions

Arithmetic and Logic Instructions, Branch Instructions

Types of Instructions

- Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data value is not modified

Data Transfer Instructions

Mode	Assembly	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC+ADR]$
Immediate operand	LD #NBR	$AC \leftarrow \mathrm{NBR}$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR+XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1],\ R1 \leftarrow R1+1$

Data Manipulation Instructions

- Arithmetic
- Logical & Bit Manipulation
- Shift

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate	NEG

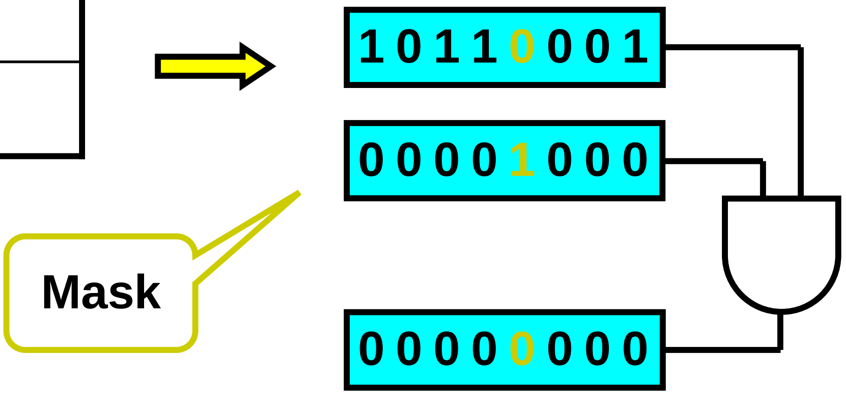
Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (Subtract)	CMP
Test (AND)	TST

Subtract A – B but don't store the result



Conditional Branch Instructions

Mnemonic	Branch Condition	Tested Condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0

- **Thank you**

UNIT-III

INPUT/OUTPUT ORGANIZATION

I/O Organization

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment.

Peripheral Devices

An external device connected to an I/O module Provide a means of exchanging data between the external device environment and the computer.

Attach to the computer by a link to an I/O module

The link is used to exchange control, status, and data between the I/O module and the external device.

Peripheral Devices

The most common input output devices are:

i) Monitor

ii) Keyboard

iii) Mouse

iv) Printer

v) Magnetic tapes

The devices that are under the direct control of the computer are said to be connected online.

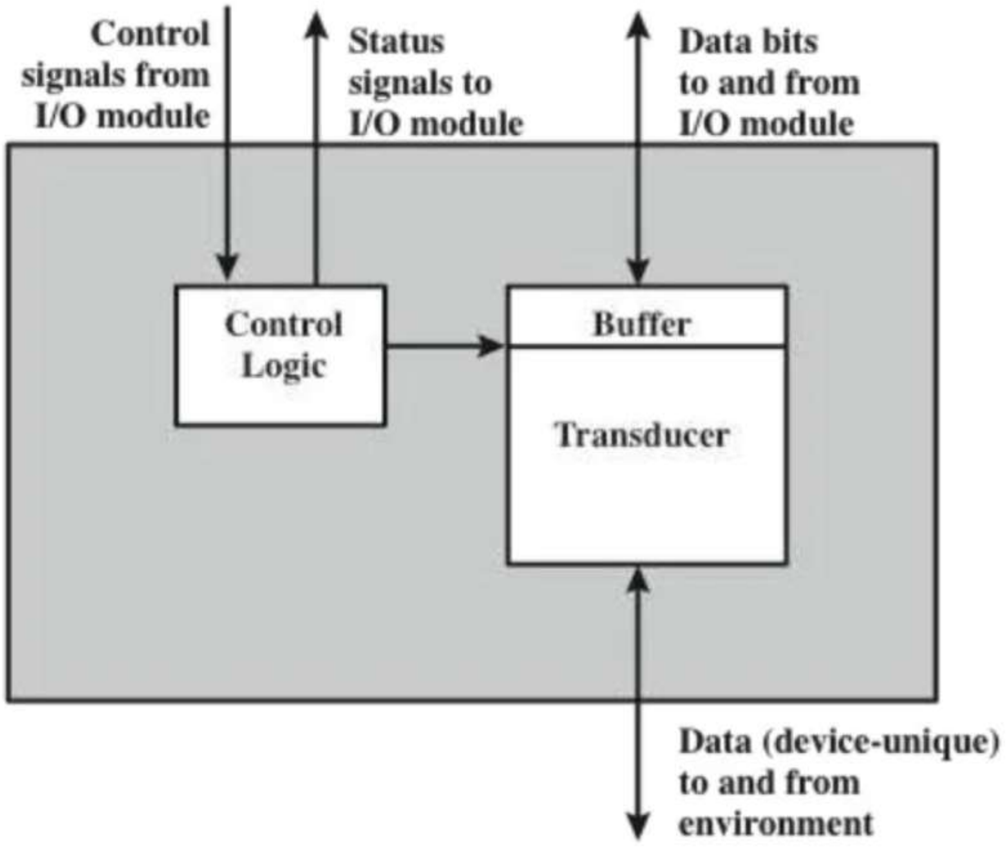

Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique) to and from environment

Figure 7.2 Block Diagram of an External Device

I/O Module Structure

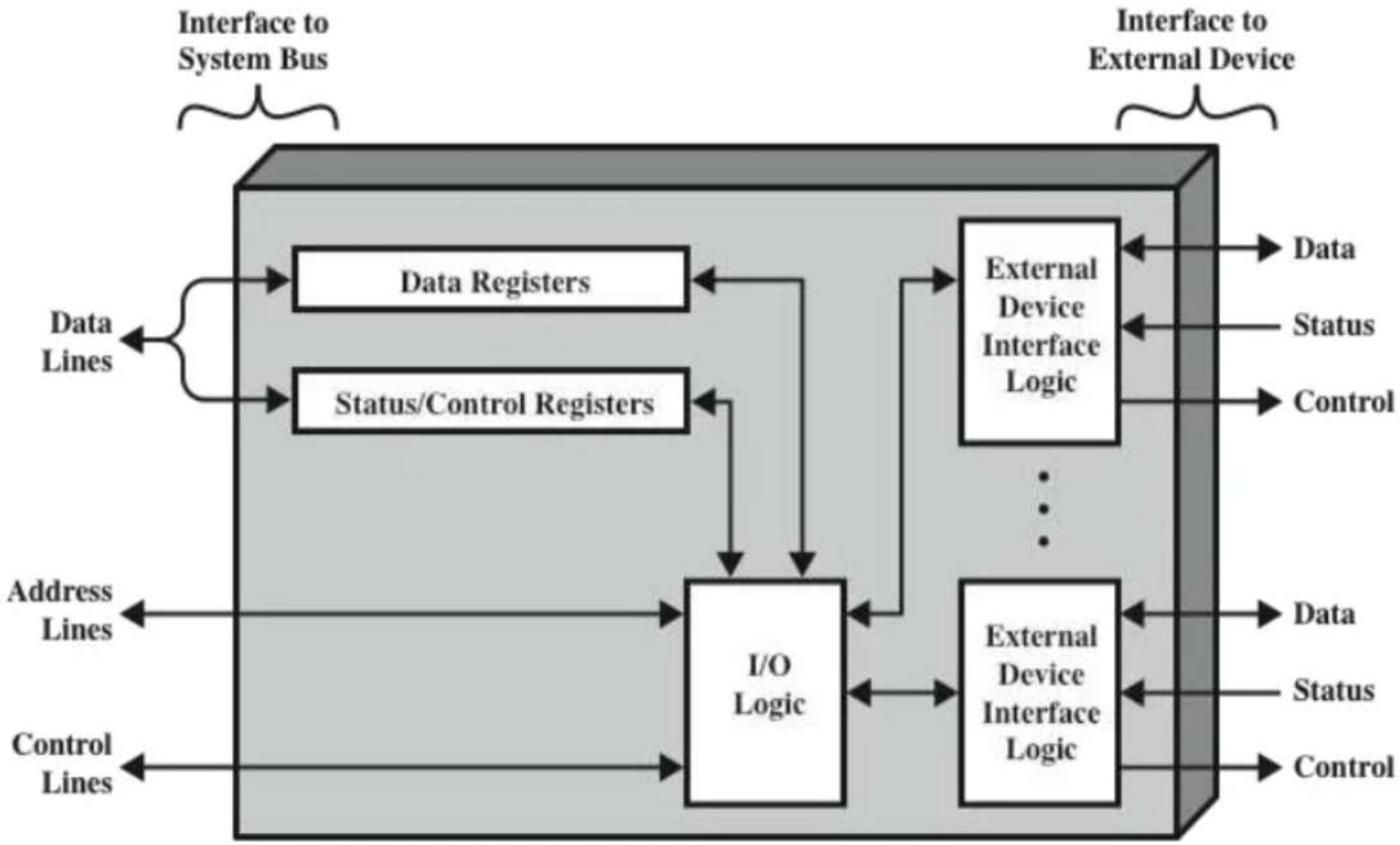
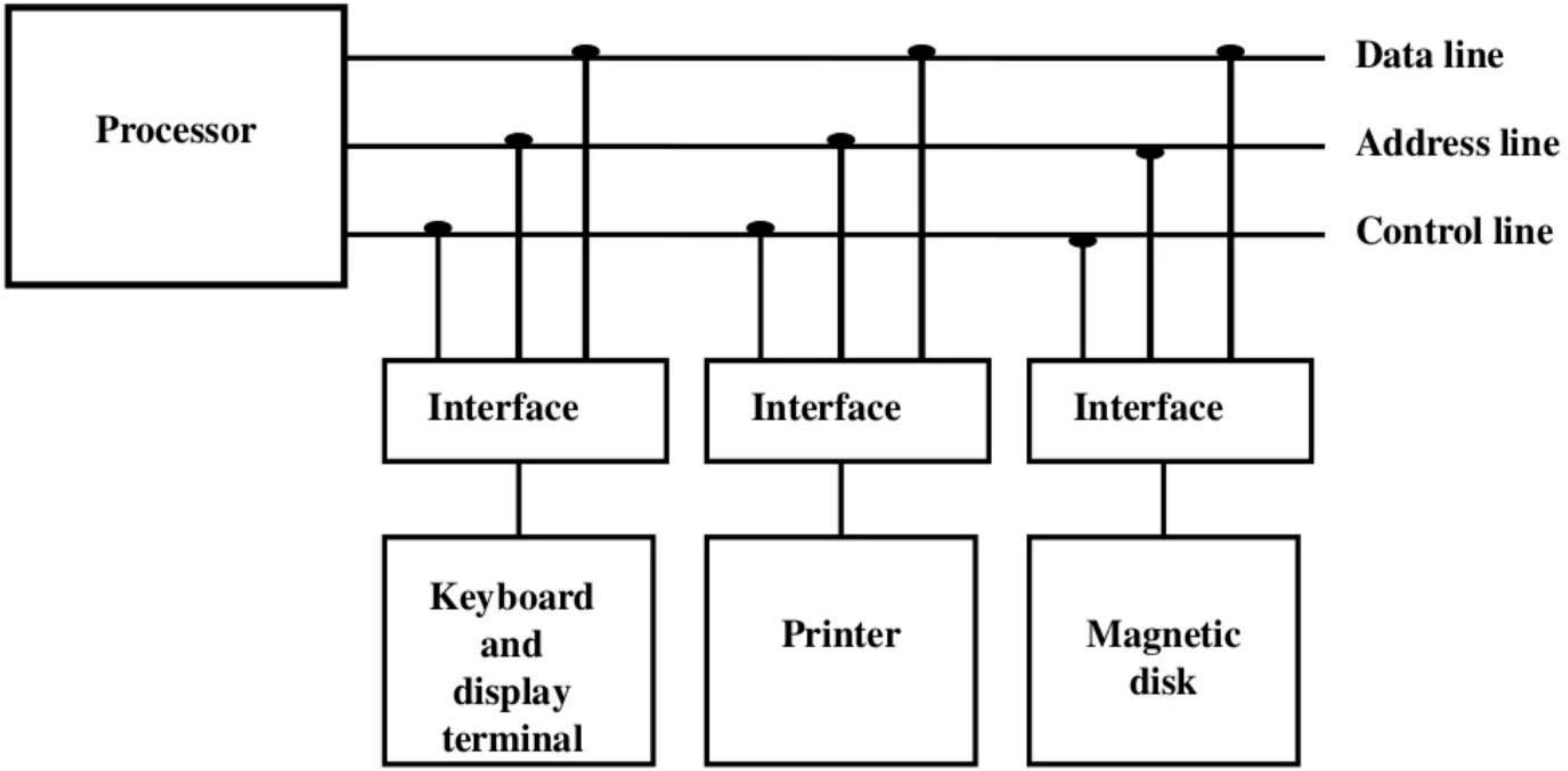


Figure 7.3 Block Diagram of an I/O Module

I/O BUS and Interface Module

It defines the typical link between the processor and several peripherals. The I/O Bus consists of **data lines, address lines** and **control lines**.

I/O BUS and Interface Module



Connection of I/O bus to input-output devices

I/O Commands

There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:

Control

- used to activate a peripheral and tell it what to do.

Test

- used to test various status conditions associated with an I/O module and its peripherals.

Read

- causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer.

Write

- causes the I/O module to take an item of data from the data bus and subsequently **transmit that data item to the peripheral.**

Accessing I/O Devices

- I/O devices accessed through I/O interface.
- Requirements for I/O interface:
  - CPU communication
  - Device communication
  - Data buffering
  - Control and timing
  - Error detection.

CPU Communication:

- **Processor sends commands to the I/O system which are generally the control signals on the control bus.**
- **Exchange of data between the processor and the I/O interface over the data bus.**
- **Check whether the devices are ready or not.**

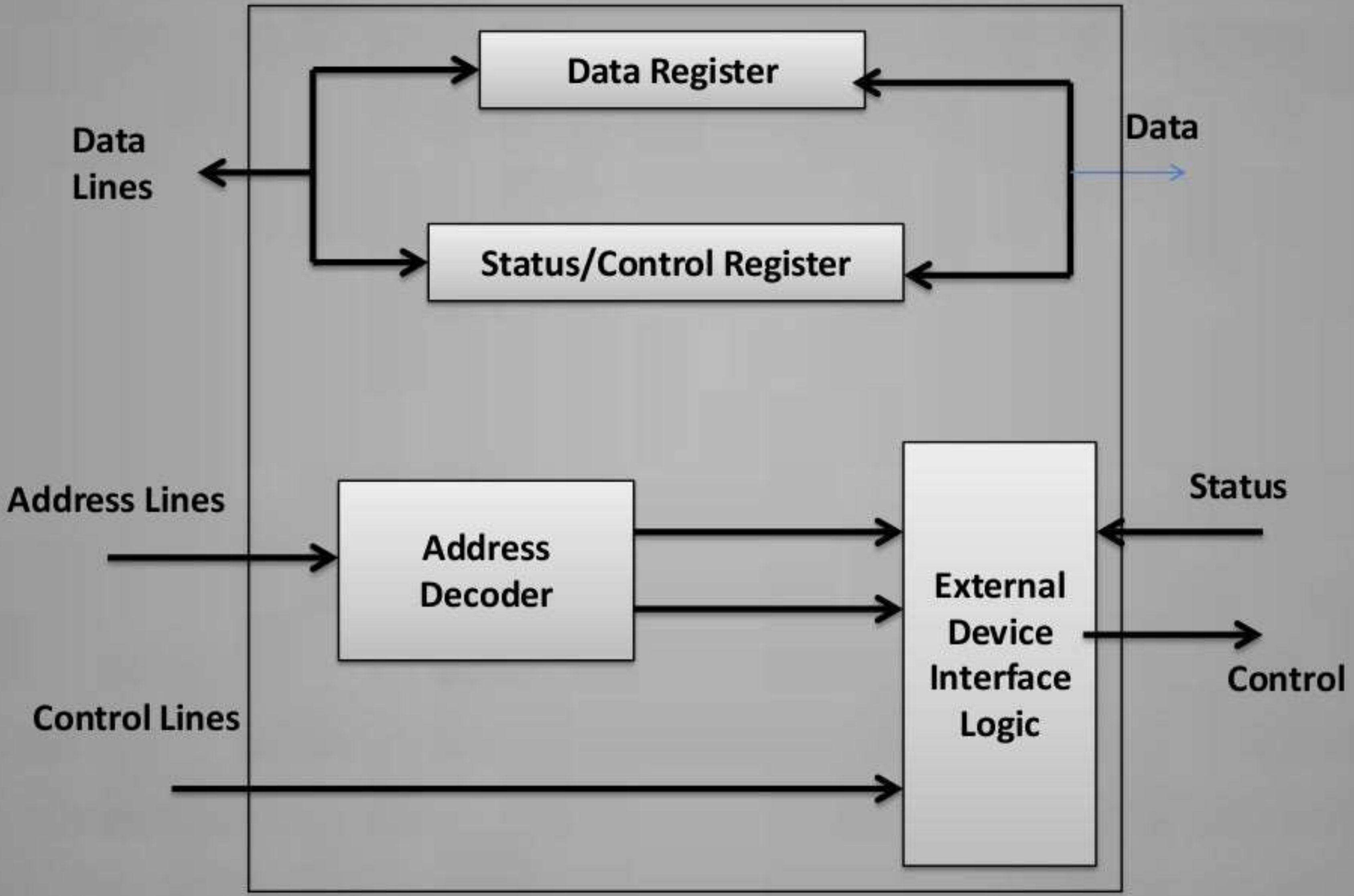
Data Buffering:

- Data transfer rate is too high .
- Data from processor and memory are sent to an I/O interface, buffered and then sent to the peripheral device at its data rate.

Error Detection:

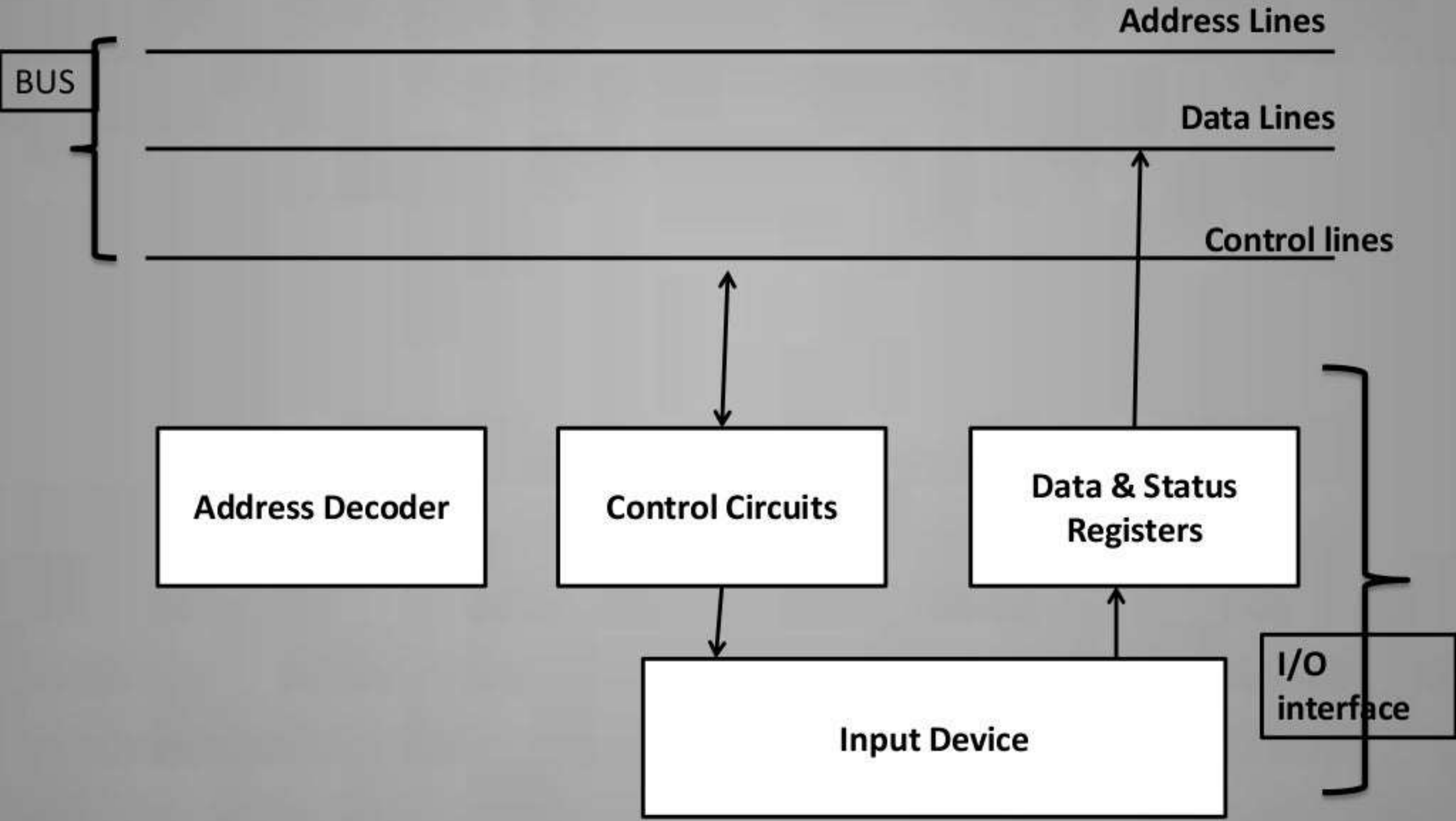
- I/O interface is responsible for error detection
- Used to report errors to the processor.
- Types of errors:
  - Mechanical, electrical malfunctions, bad disk track, unintentional changes.

I/O interface Block diagram

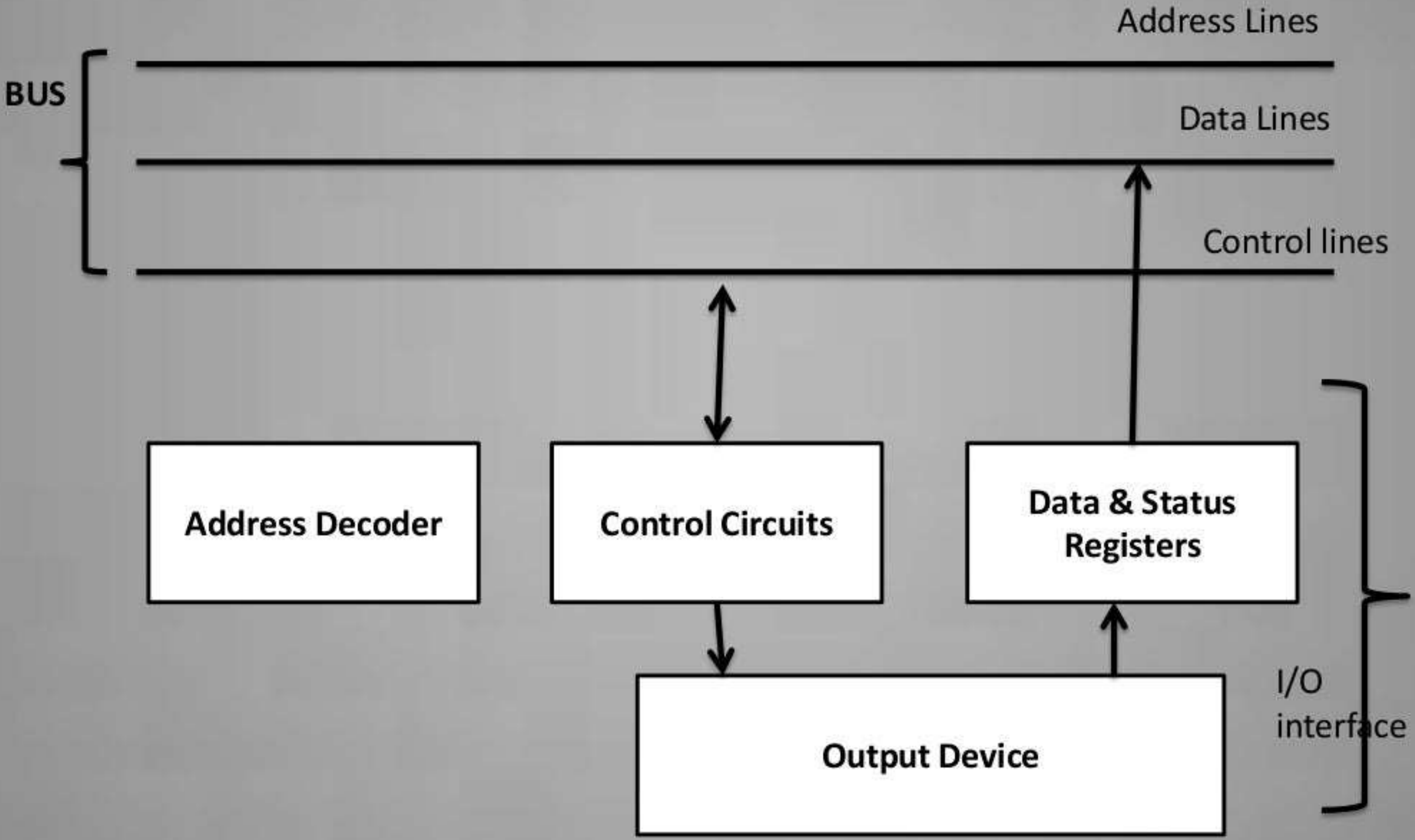


- **Data Register:** holds the data being transferred to or from the processor.
- **Status/Control Register:** contains information relevant to the operation.
- **Data and status/control registers:** are connected to the data bus.
- **Address decoder**: enables the device to recognize its address.

I/O interface for Input Device



I/O interface for Output Device



I/O interface Techniques

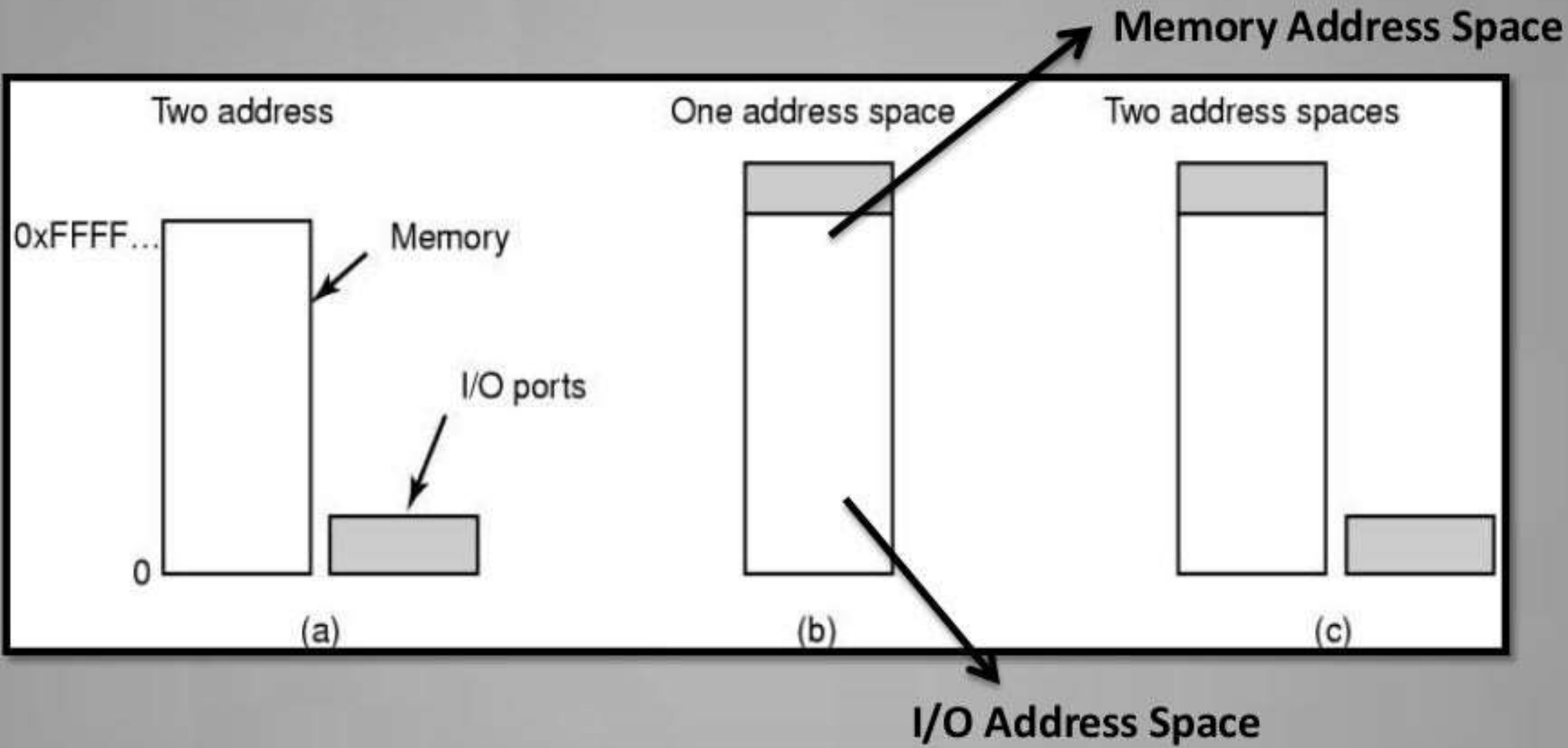
I/O Ports

- 4 registers - status, control, data-in, data-out
  - **Status** - states whether the current command is completed, byte is available, device has an error, etc
  - **Control** - host determines to start a command or change the mode of a device
  - **Data-in** - host reads to get input
  - **Data-out** - host writes to send output
- Size of registers - 1 to 4 bytes

I/O devices can be interfaced to a computer system I/O in 2 ways:

- **Memory Mapped I/O**
- **I/O mapped I/O**

Memory-Mapped I/O (1)



(a) Separate I/O and memory space

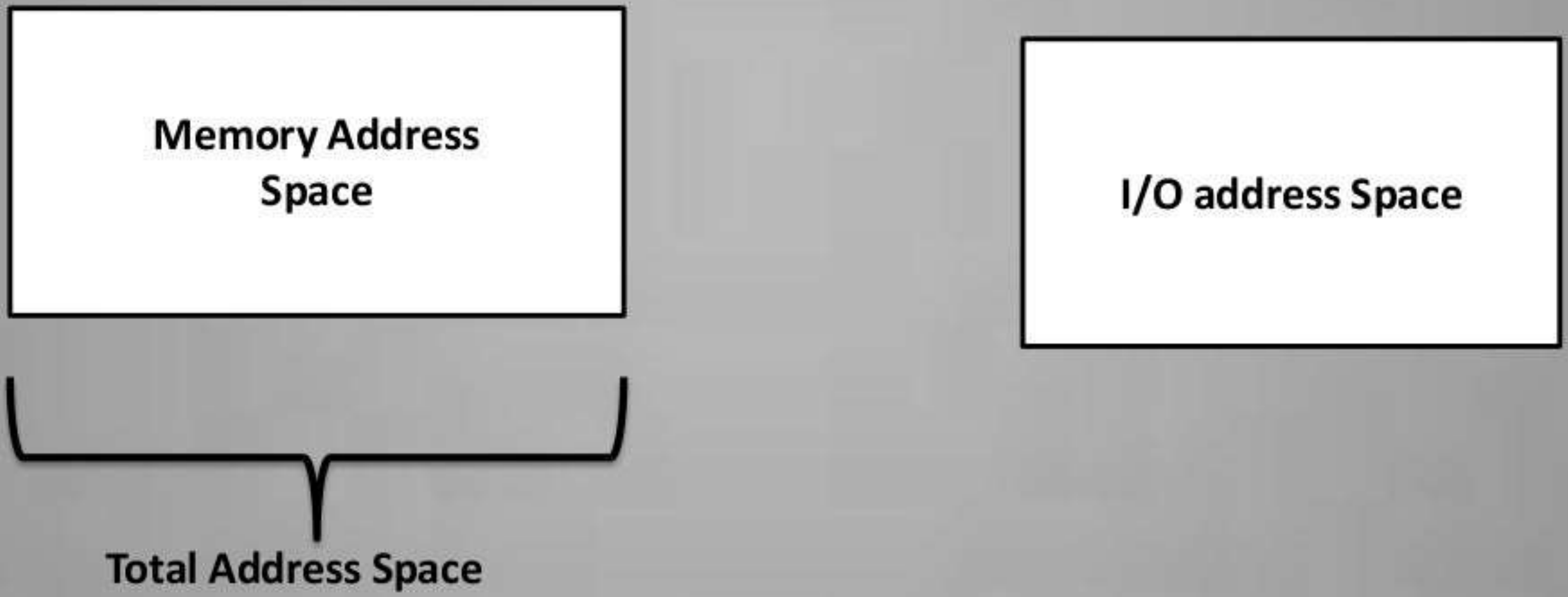
(b) Memory-mapped I/O

(c) Hybrid

Memory Mapped I/O

- **No need of special I/O instructions.**
- **Memory related instructions are used for I/O related operations.**

I/O Mapped I/O



I/O Mapped I/O

- If we want to reduce the memory address space, we allot a different I/O address space, apart from total memory space.

➢ Memory related instructions do not work here
➢ Processor use these mode only for I/O Read, I/O Write.

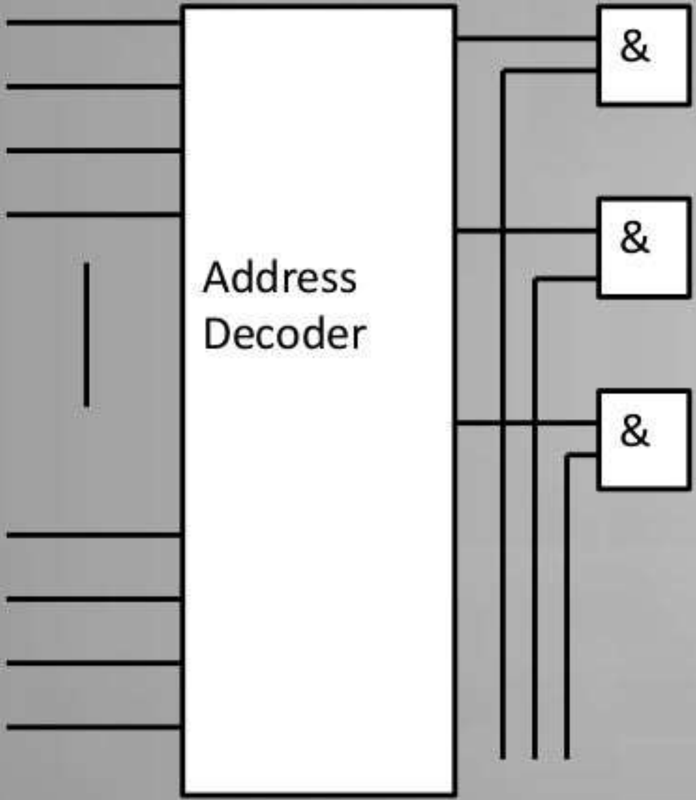
Difference between Memory Mapped I/O & I/O mapped I/O

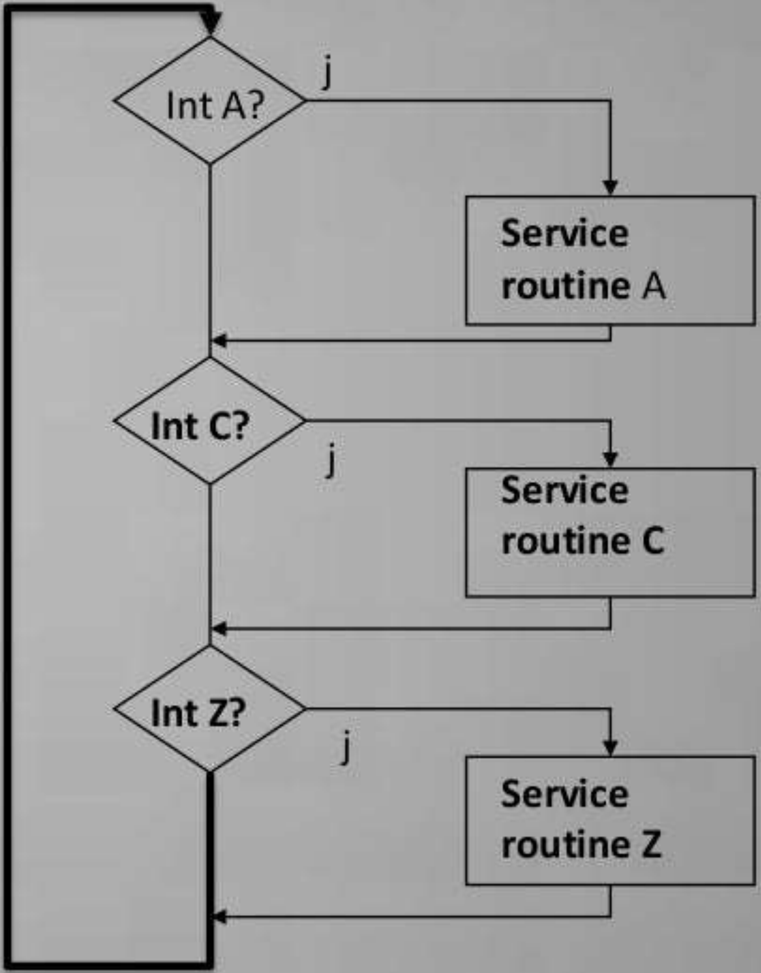
Memory Mapped I/O	I/O Mapped I/O
Memory & I/O share the entire address range of processor	Processor provides separate address range for memory & I/O
Processor provides more address lines for accessing memory	Less address lines for accessing I/O
More Decoding is required	Less decoding is required
Memory control signals used to control Read & Write I/O operations	I/O control signals are used to control Read & Write I/O operations

Programmed I/O

- I/O operation means
  - A data transfer between an I/O device & memory or
  - Between I/O device & Processor.
- If any I/O operations are completely controlled by processor, then the system is said to be using " Programmed I/O"
  - Processor has to check I/O system periodically until the operation completes → "POLLING"
  - Microprocessor has to check if any device need service.

Programmed I/O





Priority:

- The Routines assigns priority to the different I/O devices
- Port A is always checked 1st.
- Then Port B
- Then Port C
- Order may change by changing routine.

INTERRUPTS

INTRODUCTION

- **INTERRUPT** meaning to break the sequence of operations.
- While the processor is executing a program an 'interrupt' breaks the sequence of execution of that program and start execution of another program.

INTERRUPT

Interrupt is a signal from a device attached to a computer or from a program within the computer that requires the operating system to figure out what to do next.

Need of Interrupts?

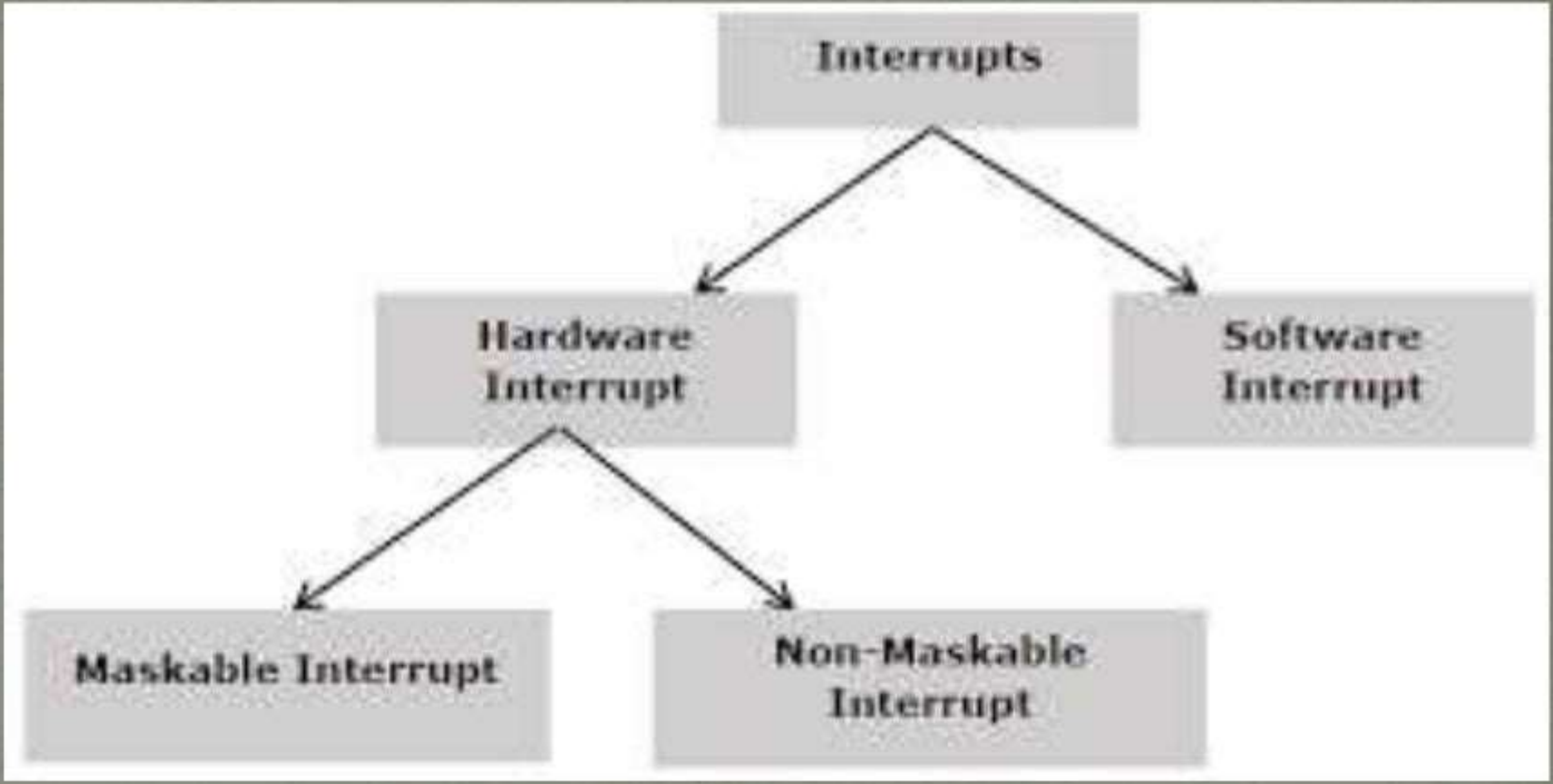
- Devices and programs occasionally need CPU service but we can’ t predict when.
- So for the interaction with CPU each device or a program is allowed to give interrupt so that it can be used as a signal to the processor.
- Need a way for CPU to find out devices/programs need attention

Types of Interrupts

Generally Interrupts are of Two types.

1) Hardware Interrupts
2) Software Interrupts

Types


Interrupts
Hardware Interrupt
Software Interrupt
Maskable Interrupt
Non-Maskable Interrupt

Hardware Interrupt

If the signal for the processor is from external device or hardware it is called hardware interrupts.

Example:

Keystroke(pressing of key on keyboard) and mouse movements cause hardware interrupts.

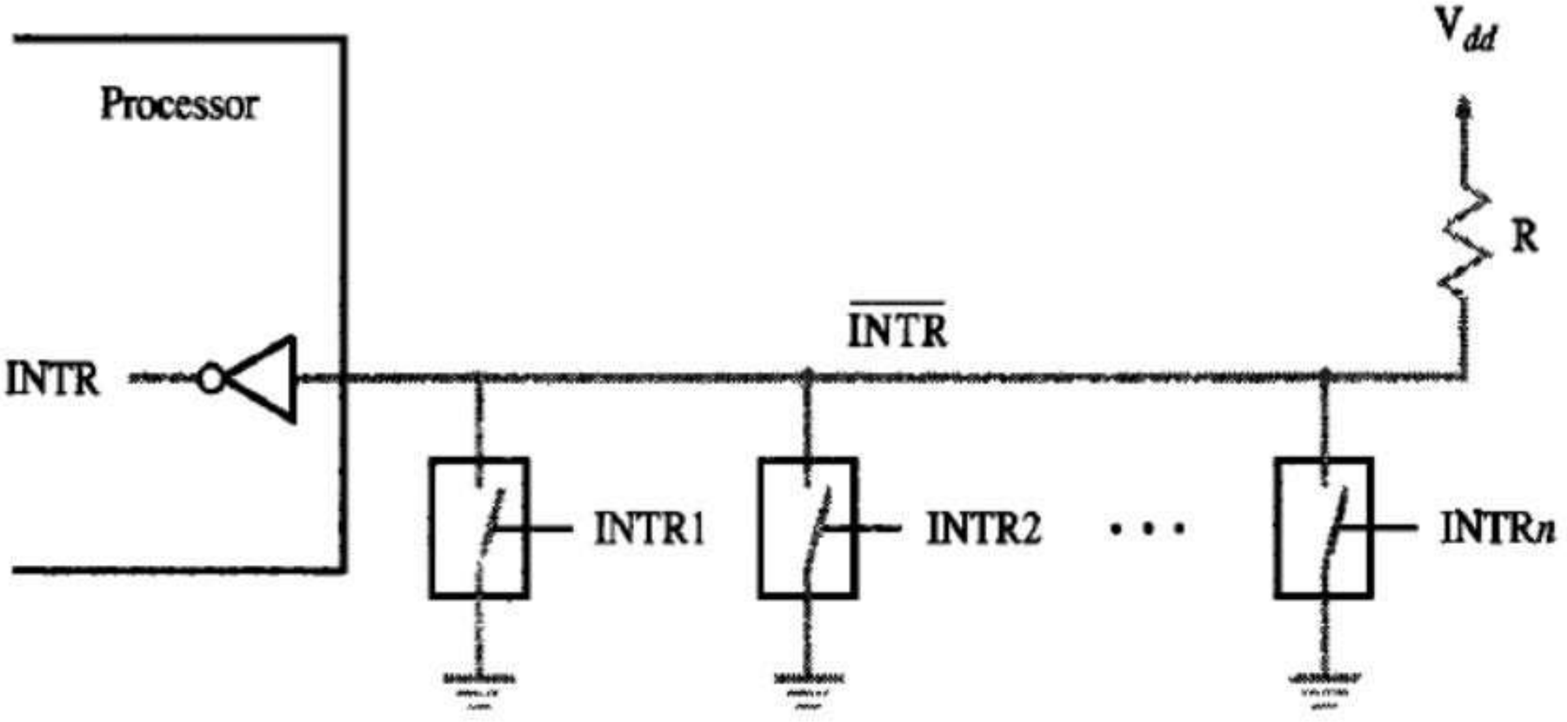
Hardware Interrupt

- ***Mask-able Interrupt*:**

  The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.

- **Non Mask-able Interrupt:**

  The hardware which cannot be delayed and should process by the processor immediately.


Processor
INTR
$\overline{\text{INTR}}$
V_{dd}
R
INTR1
INTR2
INTRn

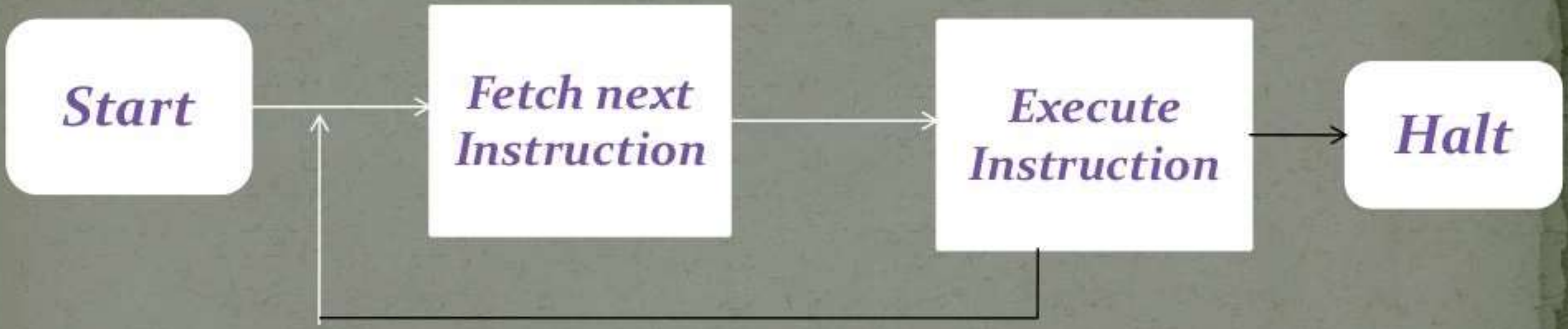
Instruction Cycle

- An instruction cycle includes
  - Fetch
  - Decode
  - Execute
- It is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the central processing unit (CPU).

Instruction Cycle and Interrupts

❖ Without Interrupts

- Instruction fetch
- Instruction execute



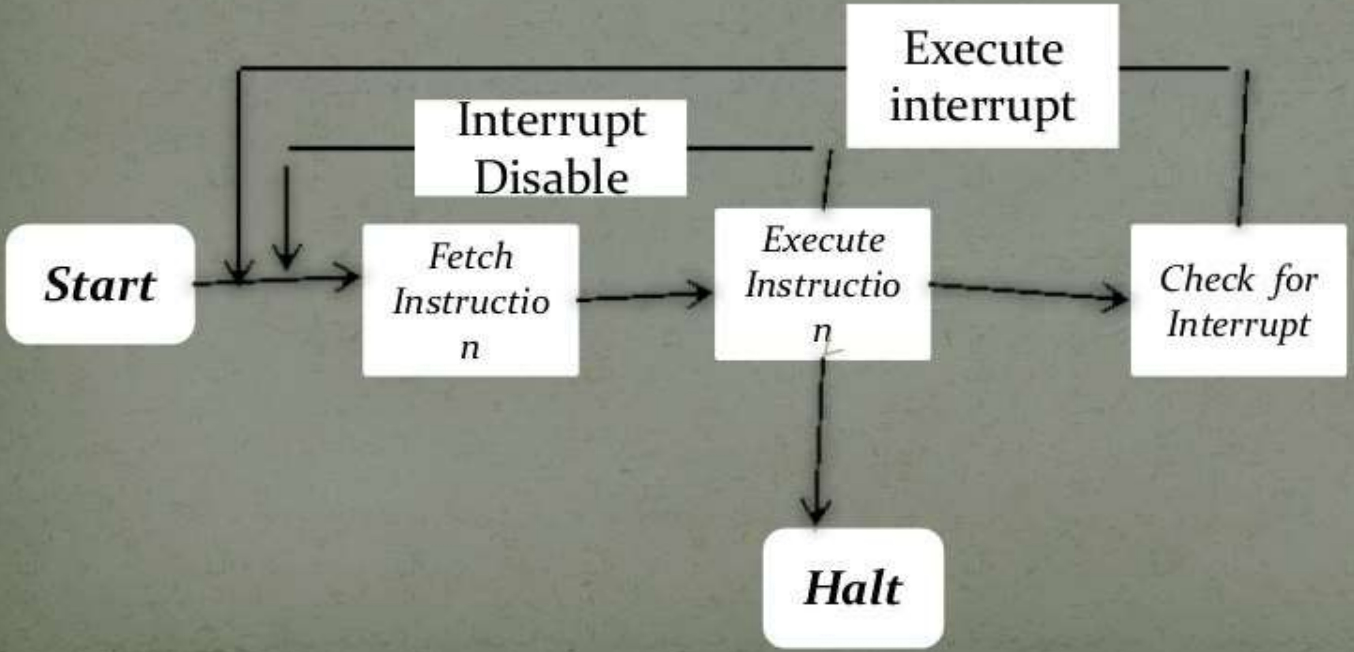
Instruction cycle and Interrupts

- **With Interrupt**

Instruction Fetch

Instruction Execute

Check the Interrupt



Classes of Interrupt

- ***Program***

  Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space

- ***Timer***

  Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.

- ***I/O***

  Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.

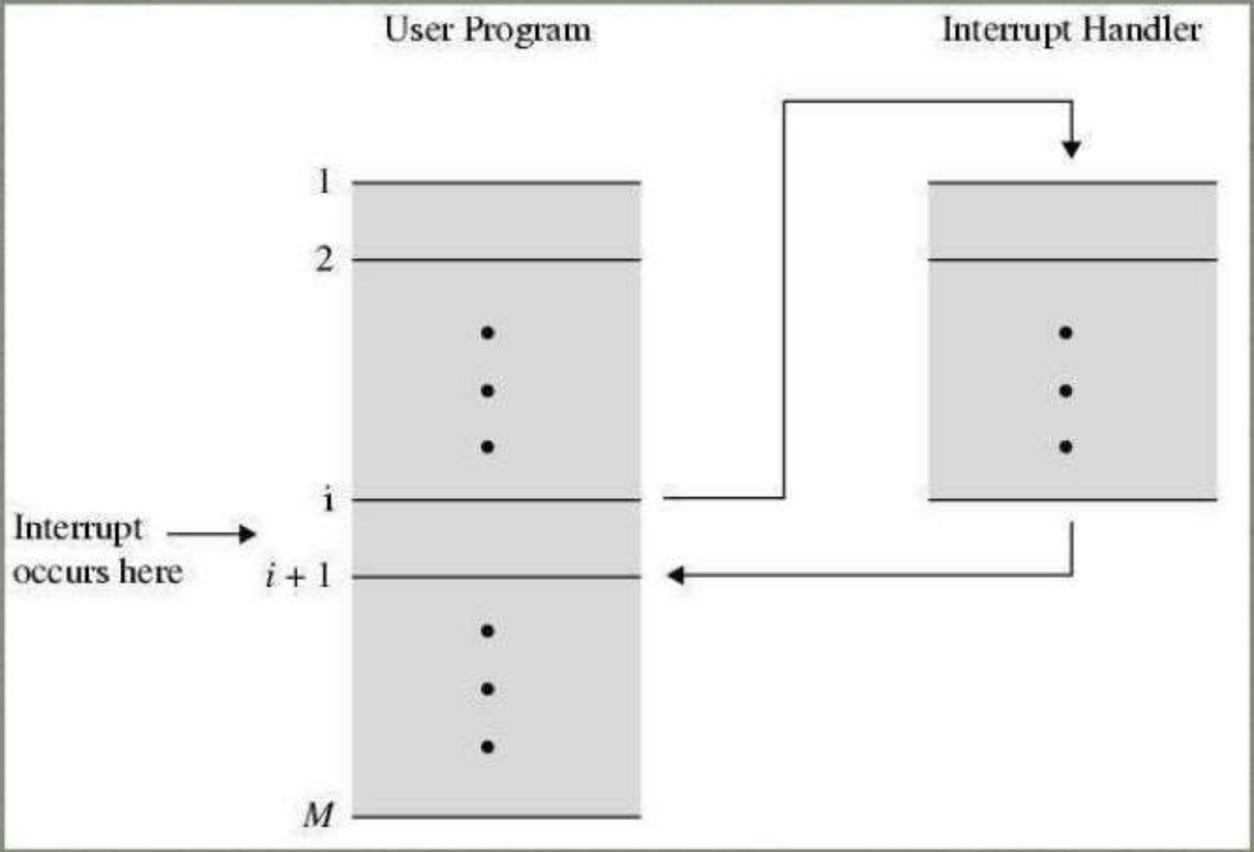
- Hardware Failure

  Generated by a failure, such as power failure or memory parity error.

How Interrupts are handled?

- ➢ If there is an interrupt in instruction cycle then
- It will trigger the interrupt handler.
- The handler will stop the present instruction which is processing and save its state in a PC register and load the state of the interrupt program .
- OS serves the interrupt from Interrupt Vector Table through Interrupt Service Routine(Interrupt handler).
- After processing the interrupt, interrupt handler will load the instruction of previous saved process from the register, process will start its processing where it's left. This saving the old instruction processing configuration and loading the new interrupt configuration is also called as context switching.

Interrupt Handling



Multiple Interrupts

An interrupt event that can occur while the processor is handling a previous interrupt.

- ✓ ***Disable Interrupt:***

  Processor Will ignore further interrupts while handling one interrupt (Sequential Interrupt)

- ✓ ***Define Priorities:***

  Low priority interrupt interrupted by higher priority interrupts.

1.Enabling and disabling interrupts

- A processor has the facility to enable and disable interrupts as desired.
- When a device request the interrupt during the processor service for another interrupt, the result cause the processor enter into the infinite loop.
- This can be handled by the following 2 ways:

➢ The processor ignore the interrupt request line(INTR) until the Interrupt Service Routine(ISR) is completed.

➢ This can be done by using interrupt-Disable as first instruction and interrupt-Enable as the last instruction.

- The second option is processor automatically disable interrupts before starting the execution of the ISR.
- The status register PS stored in the stack with PC value.
- The processor set this register bit 1 when the interrupt accept and when a return instruction is executed, the contents of the PS are cleared (0)and stored in the stack again.

2.Handling Multiple Devices

- When the number of devices initiating interrupts.
- For example, device X may request an interrupt while an interrupt caused by device Y is being serviced.
- Hence all the device using the common interrupt line.
- Additional information require to identify the device that activated the request.
- When the two devices activated the line at the same time, we must break up the tie and chose one the device request among two. Some scheme should be used by the processor.

2.1Polling scheme

- The device that raises the interrupt will set one of the bit (IRQ) in status register to 1.
- The processor will poll the devices to find which raised an interrupt first.

Disadvantage:

- Time spend in interrogating the IRQ bits of the devices that may not be requesting any service.

2.2Vectored interrupts

- To reduce the time involved in the polling scheme, a device requesting an interrupt may identify itself directly to the processor.
- A device can send a special code to the processor over the bus. The code is used to identify the device.
- If the interrupt produces a CALL to a predetermined memory location, which is the starting address of ISR, then that address is called vectored address and such interrupts are called vectored interrupts.

Introduction of Direct Memory Access (DMA)

"DIRECT MEMORY ACCESS (**DMA**) IS A FEATURE OF COMPUTERIZED SYSTEMS THAT ALLOWS CERTAIN HARDWARE SUBSYSTEMS TO ACCESS MAIN SYSTEM MEMORY INDEPENDENTLY OF THE CENTRAL PROCESSING UNIT (CPU)."

–WIKIPEDIA

"Direct memory access (DMA) is a means of having a peripheral device control a processor's memory bus directly."

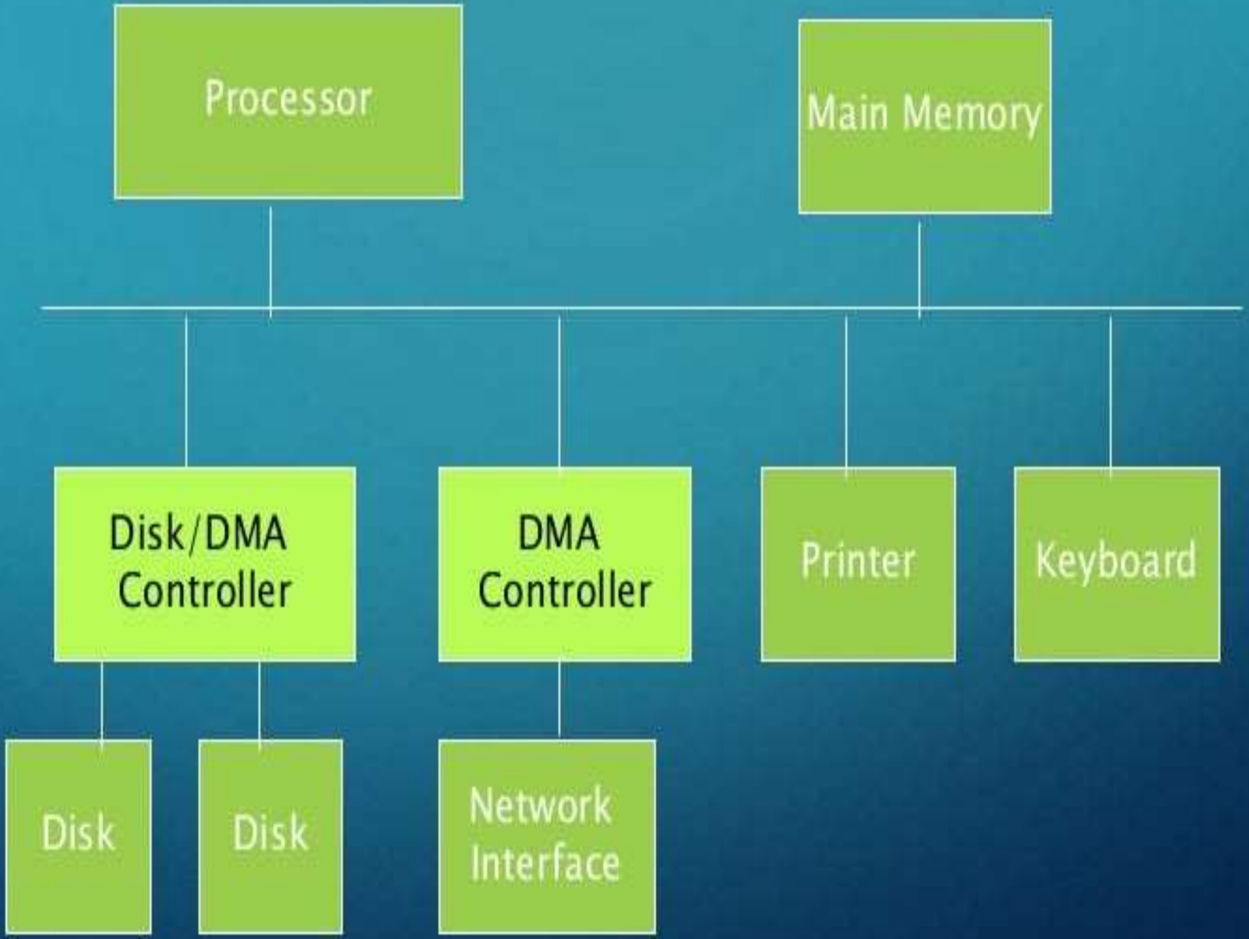
What is DMA?

- The direct memory access (DMA) I/O technique provides direct access to the memory while the microprocessor is temporarily disabled.
- I/O devices are connected to system bus via a special interference circuit known as "DMA Controller".
- In DMA, both CPU and DMA controller have access to main memory via a shared system bus having data, address and control lines.
- It is sometimes referred to as a channel. In an alternate configuration, the DMA controller may be incorporated directly into the I/O device.

What is DMA?

- A DMA controller temporarily borrows the address bus, data bus, and control bus from the microprocessor and transfers the data bytes directly between an I/O port and a series of memory locations.
- The DMA transfer is also used to do high-speed memory-to memory transfers.
- DMA transfer can be done in tow ways. A) DMA Transfer Blocks. B) Cycle Stealing

USE OF DMA CONTROLLERS IN A COMPUTER SYSTEM



Why DMA?

- An important aspect of governing the Computer System performance is the transfer of data between memory and I/O devices.
- The operation involves loading programs or data files from disk into memory, saving file on disk, and accessing virtual memory pages on any secondary storage medium.
- The process would be much quicker if we could bypass the CPU & transfer data directly from the I/O device to memory. Direct Memory Access does exactly that.
- During DMA transfer, the CPU is idle and no control of the memory buses.

DMA VS. NO DMA

Without DMA	With DMA
When the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work.	The CPU initiates the transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller when the operation is done.

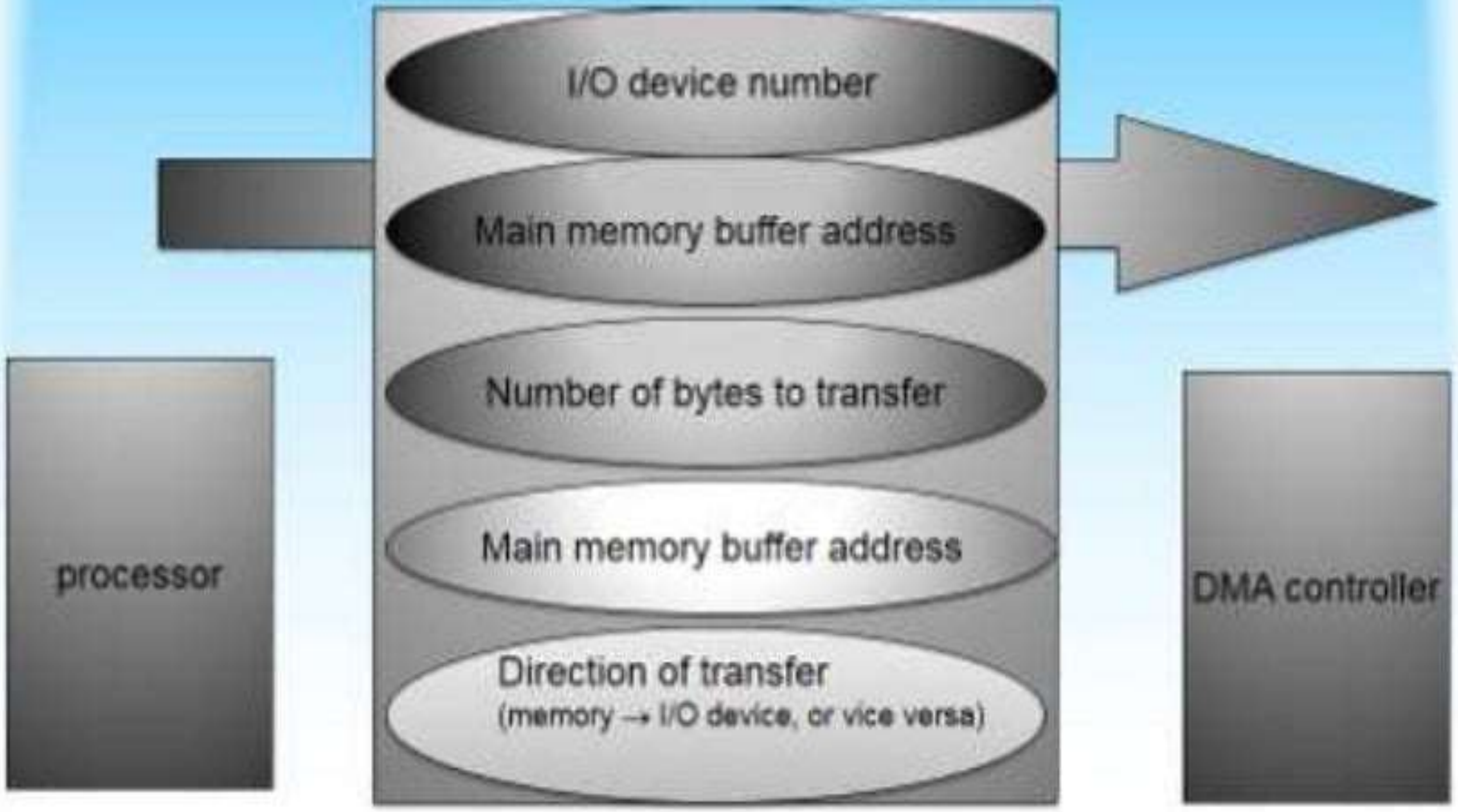
This feature is useful any time the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer.

DMA INITIALIZATION

DMA controllers require initialization by software. Typical setup parameters include the base address of the source area, the base address of the destination area, the length of the block, and whether the DMA controller should generate a processor interrupt once the block transfer is complete.

Example: Reading from an I/O device

Processor gives details to the DMA controller


I/O device number
Main memory buffer address
Number of bytes to transfer
Main memory buffer address
Direction of transfer
(memory → I/O device, or vice versa)
processor
DMA controller

TYPES OF DMA TRANSFER:

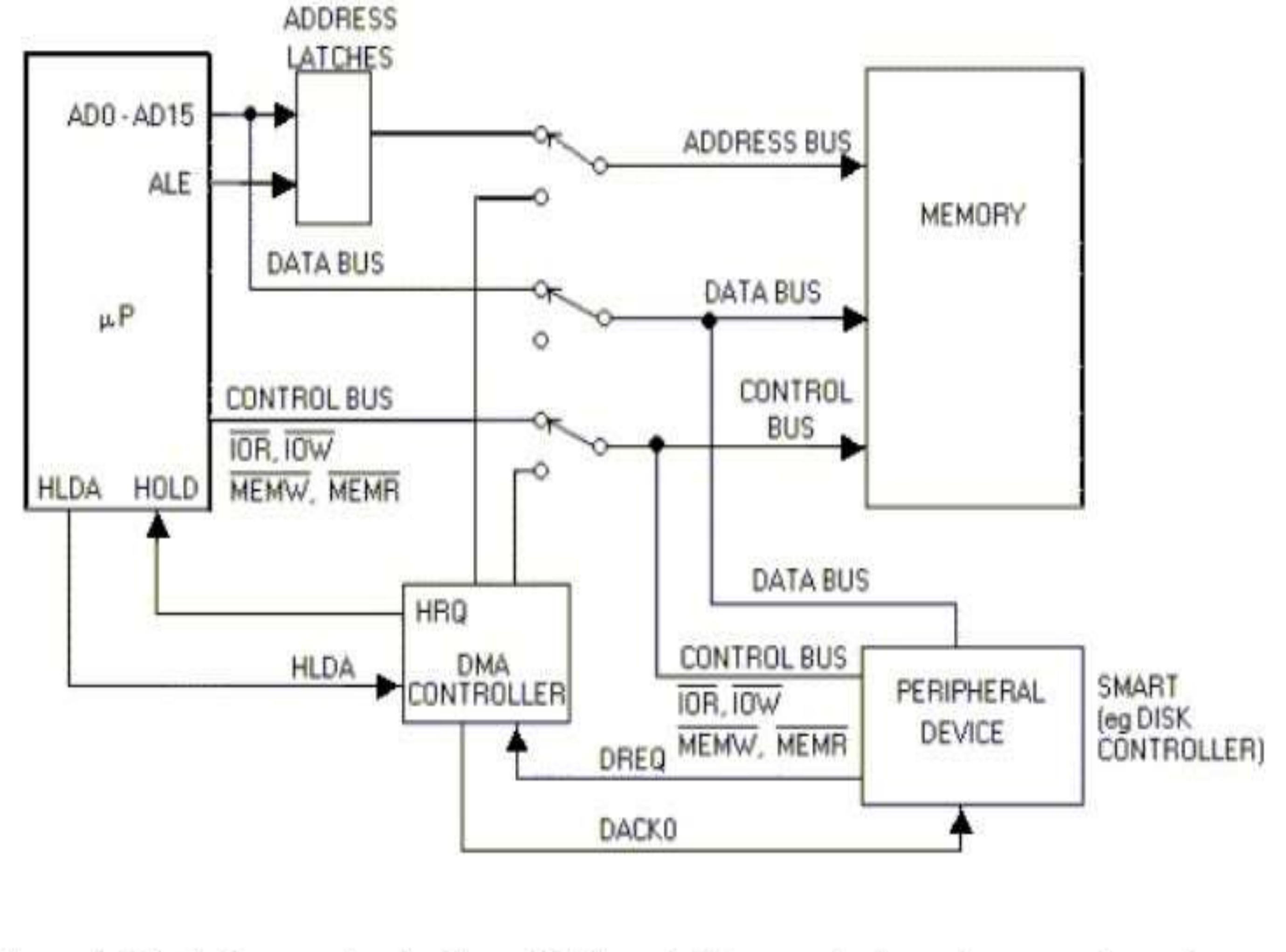
1. DMA TRANSFER BLOCK

- In this DMA mode , DMA controller is master of memory bus.
- This mode is needed by the secondary memory like disk drives, that have data transmission and are no to be stopped or slowed without any loss of data transfer of blocks.
- Block DMA transfer supports faster I/O data transfer rates but the CPU remains inactive for relatively long period by tieing up the system bus.

2.CYCLE STEALING

- In this method, system allows DMA controller to use system bus to transfer one word, after which it should return back control of bus to CPU.
- This method reduces maximum I/O transfer rates
- It also reduces interference of DMA controller in CPU memory access
- It is completely eliminated by designing DMA interface so that system bus cycles are stolen only when CPU is not actually using system bus.
- This is also called as Transparent DMA

DMA DATA TRANSFER: BLOCK DIAGRAM



ADDRESS LATCHES
AD0 - AD15
ALE
μP
DATA BUS
ADDRESS BUS
MEMORY
DATA BUS
CONTROL BUS
$\overline{IOR}$, $\overline{IOW}$
$\overline{MEMW}$, $\overline{MEMR}$
CONTROL BUS
HLDA
HOLD
DATA BUS
HRQ
DMA CONTROLLER
HLDA
CONTROL BUS
$\overline{IOR}$, $\overline{IOW}$
$\overline{MEMW}$, $\overline{MEMR}$
PERIPHERAL DEVICE
SMART (eg DISK CONTROLLER)
DREQ
DACK0

Figure 2: Block diagram showing how a DMA controller operates in a microcomputer system

DATA TRANSFER: BLOCK COMPONENTS

- Two control signals are used to request and acknowledge a DMA transfer .
- The HOLD signal is a bus request (**BR**) signal which asks the microprocessor to release control of the buses after the current bus cycle.
- The HLDA signal is a bus grant (**BG**) signal which indicates that the microprocessor has indeed released control of its buses by placing the buses at their high-impedance states.
- DREQi (DMA request): Used to request a DMA transfer for a particular DMA channel.
- DACKi (DMA channel acknowledge): Acknowledges a channel DMA

INTERNAL CONFIGURATION OF DMA CONTROLLER

The DMA controller includes several internal registers :-

- The DMA Address Register contains the memory address to be used in the data transfer. The CPU treats this signal as one or more output ports.

- The DMA Count Register, also called Word Count Register, contains the number of bytes of data to be transferred. Like the DMA address register, it too is treated as an O/P port (with a different address) by the CPU.

- The DMA Control Register accepts commands from the CPU. It is also treated as an O/P port by the CPU.

- The DMA Data Register, are used to store intermediate data values

DMA DATA TRANSFER

Data transfer technique directly between memory and I/O device

Steps:

1. I/O device asserts DREQ signal.
2. DMA controller sends HOLD signal to CPU
3. CPU sends HLDA back to DMA controller
4. DMA controller give DMA acknowledgment back to I/O.

DMA DATA TRANSFER

5. DMA controller places memory address on address bus

6. DMA controller updates memory address register and word counter register

7. When DC register becomes zero ,DMA controller sets HOLD=0

8. Data transfer process terminates and processor regain control of system

bus.

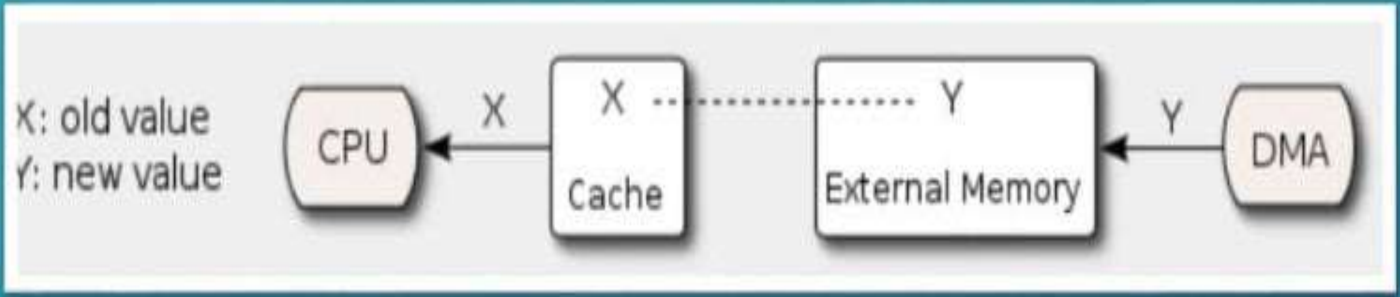
ADVANTAGES OF DMA

- DMA allows a peripheral device to read from/write to memory without going through the CPU
- DMA allows for faster processing since the processor can be working on something else while the peripheral can be populating memory.

DISADVANTAGES OF DMA

- DMA transfer requires a DMA controller to carry out the operation, hence cost of the system increases.
- Cache Coherence problems.

CACHE COHERENCE PROBLEM



Cache coherency refers to the inconsistency of data stored in local caches of a shared resource and data stored in memory.

Buses-1

- The bus protocol determines when a device may place information on the bus, when it may load the data on the bus into one of its registers, and so on.
- These rules are implemented by control signals that indicate what and when actions are to be taken.
- **Master or initiator- Device which initiate data transfers**
- **Slave or target- Device addressed by master**

Buses-2 Synchronous Bus

- All devices derive timing information from a control line called the *bus clock*
- The timing diagram shows an **idealized representation** of the actions that take place on the bus lines.

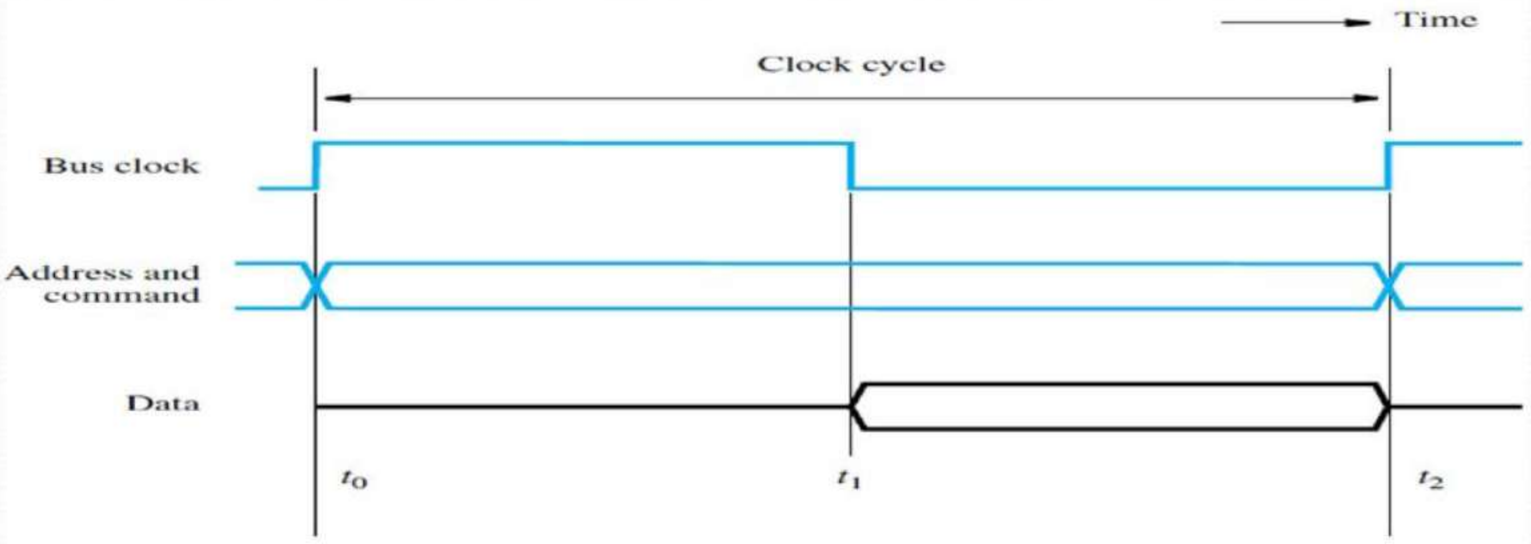


Figure 7.3 Timing of an input transfer on a synchronous bus.

Buses-3 Synchronous Bus

- The sequence of signal events during an input (Read) operation.
- At time ***t*o**, the master places the device address on the address lines and sends a command on the control lines indicating a Read operation.
- Information travels over the bus.
- The clock pulse width, ***t*1** – ***t*o**, must be longer than the maximum propagation delay over the bus. Also, it must be long enough to allow all devices to decode the address and control signals, so that the addressed device (the slave) can respond at time ***t*1** by placing the requested input data on the data lines.
- At the end of the clock cycle, at time ***t*2**, the master loads the data on the data lines into one of its registers.
- To be loaded correctly into a register, data must be available for a period greater than the setup time of the register.
- Hence, the period ***t*2** – ***t*1** must be greater than the maximum propagation time on the bus plus the setup time of the master's register.

Buses-4 Synchronous Bus

- The **exact times at** which signals change state are somewhat different from **idealized representation**, because of propagation delays on bus wires and in the circuits of the devices.

Buses-5 Synchronous Bus

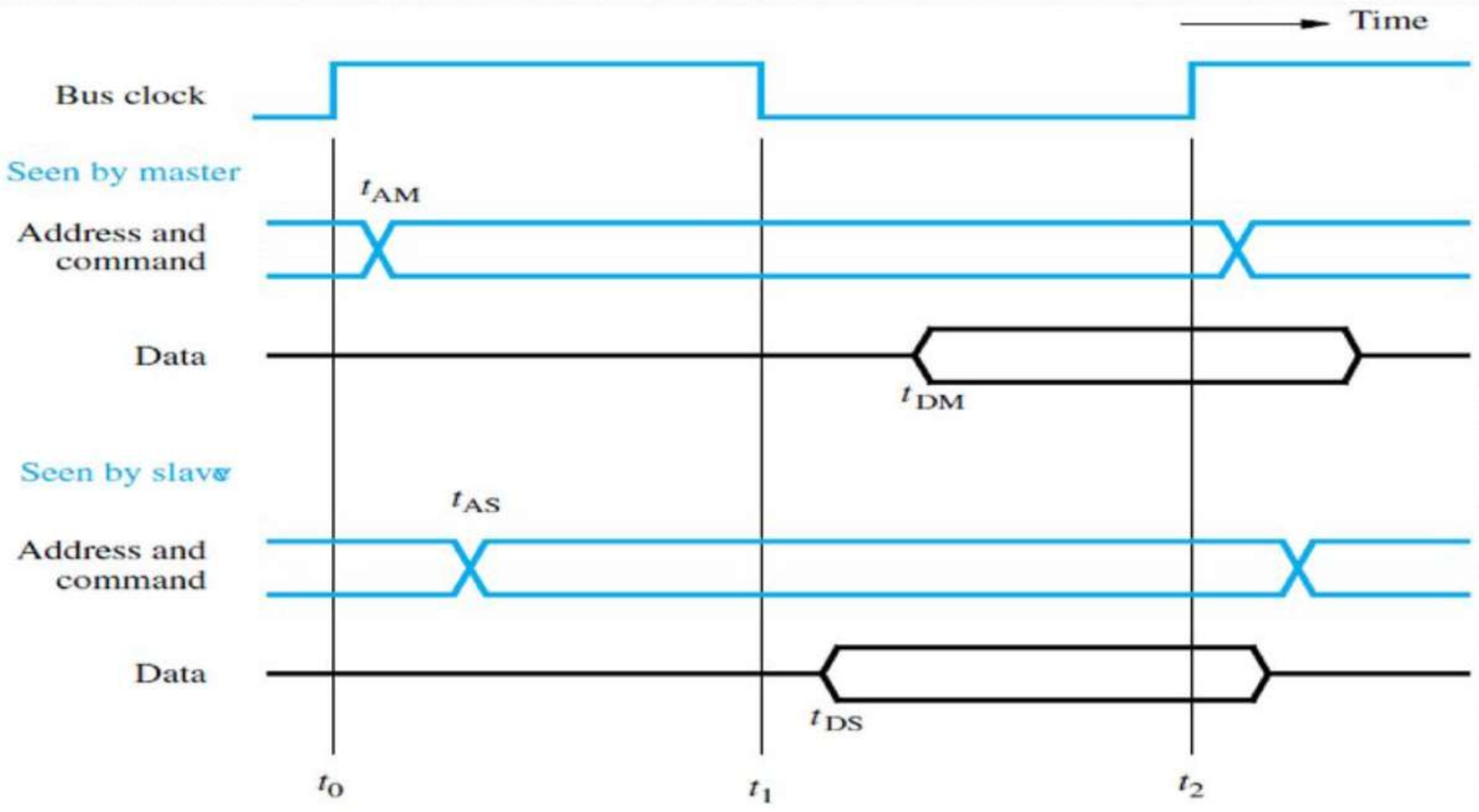


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

Buses-6 Synchronous Bus

- The master sends the address and command signals on the rising edge of the clock at the beginning of the clock cycle (**at** $\boldsymbol{t_0}$). However, these signals do not actually appear on the bus until $\boldsymbol{t}_{\mathrm{AM}}$, largely due to the delay in the electronic circuit output from the master to the bus lines.
- A short while later, at $\boldsymbol{t}_{\mathrm{AS}}$, the signals reach the slave. The slave decodes the address, and at $\boldsymbol{t_1}$ sends the requested data.
- Here again, the data signals do not appear on the bus until $\boldsymbol{t}_{\mathrm{DS}}$.
- They travel toward the master and arrive at $\boldsymbol{t}_{\mathrm{DM}}$. At $\boldsymbol{t_2}$, the master loads the data into its register.
- Hence the period $\boldsymbol{t_2} - \boldsymbol{t}_{\mathrm{DM}}$ must be greater than the setup time of that register. The data must continue to be valid after $\boldsymbol{t_2}$ for a period equal to the hold time requirement of the register.

Buses-7 Synchronous Bus Multiple-Cycle Data Transfer

- However, it has some limitations.
- Because a transfer has to be completed within one clock cycle, the clock period, $t_2 - t_0$, must be chosen to accommodate the longest delays on the bus and the slowest device interface.
- This forces all devices to operate at the speed of the slowest device.
- To overcome these limitations, most buses incorporate control signals that represent a **response from the device**. These signals inform the master that the slave has recognized its address and that it is ready to participate in a data transfer operation. They also make it possible to adjust the duration of the data transfer period to match the response speeds of different devices.
- This is often accomplished by allowing a complete data transfer operation to span several clock cycles. Then, the number of clock cycles involved can vary from one device to another.

Buses-8 Synchronous Bus Multiple-Cycle Data Transfer

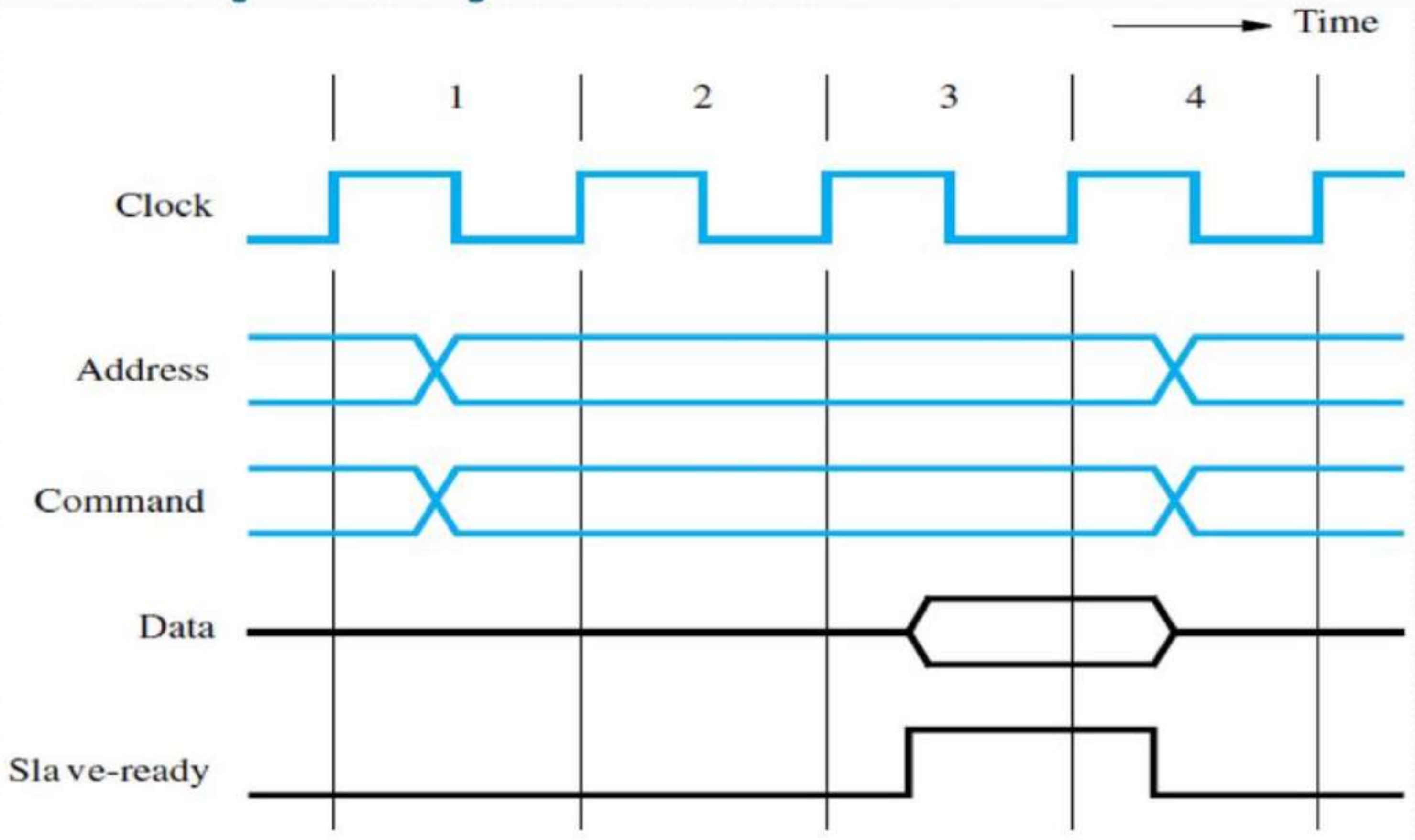


Figure 7.5 An input transfer using multiple clock cycles.

Buses-9 Synchronous Bus Multiple-Cycle Data Transfer

- During clock cycle 1, the master sends address and command information on the bus, requesting a Read operation.
- The slave receives this information and decodes it. It begins to access the requested data on the active edge of the clock at the beginning of clock cycle 2.
- Due to the delay involved in getting the data, the slave cannot respond immediately. The data become ready and are placed on the bus during clock cycle 3.
- The slave asserts a control signal called Slave-ready at the same time.
- The master, which has been waiting for this signal, loads the data into its register at the end of the clock cycle.
- The slave removes its data signals from the bus and returns its Slave-ready signal to the low level at the end of cycle 3.
- The bus transfer operation is now complete, and the master may send new address and command signals to start a new transfer in clock cycle 4.
- If the addressed device does not respond at all, the master waits for some predefined maximum number of clock cycles, then aborts the operation.

Buses-10 Asynchronous Bus

- An alternative scheme for controlling data transfers on a bus is based on the use of a *handshake* protocol between the master and the slave.
- A handshake is an exchange of command and response signals between the master and the slave.

Buses-10 Asynchronous Bus

- The master places the address and command information on the bus. Then it indicates to all devices that it has done so by activating the Master-ready line.
- This causes all devices to decode the address. The selected slave performs the required operation and informs the processor that it has done so by activating the Slave-ready line.
- The master waits for Slave-ready to become asserted before it removes its signals from the bus.
- In the case of a Read operation, it also loads the data into one of its registers.

Buses-10 Asynchronous Bus

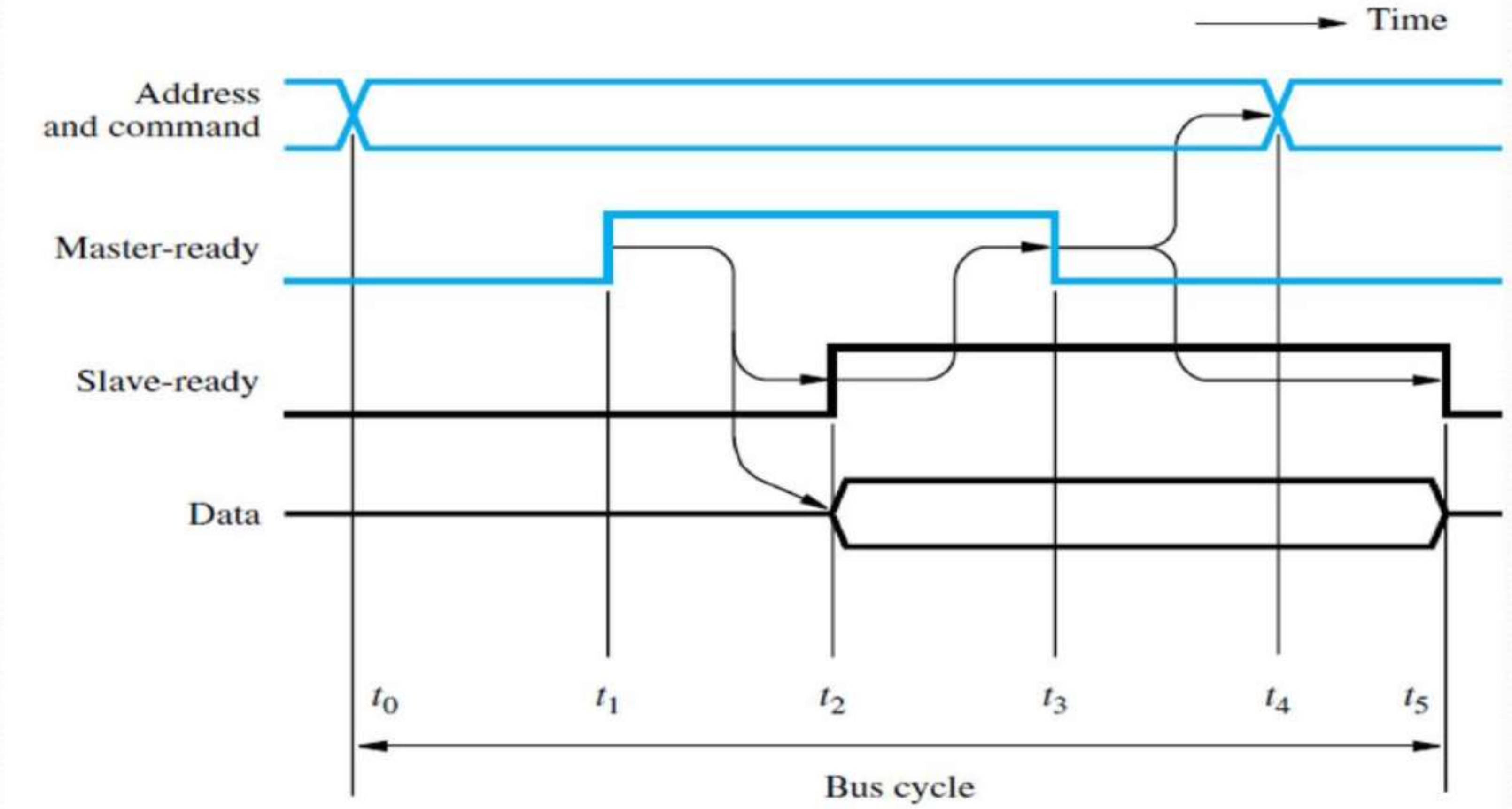


Figure 7.6 Handshake control of data transfer during an input operation.

Buses-10 Asynchronous Bus

- t_0—The master places the address and command information on the bus, and all devices on the bus decode this information.
- t_1—The master sets the Master-ready line to 1 to inform the devices that the address and command information is ready. Sufficient time should be allowed for the device interface circuitry to decode the address. The delay needed can be included in the period $t_1 - t_0$.
- t_2—The selected slave, having decoded the address and command information, performs the required input operation by placing its data on the data lines. At the same time, it sets the Slave-ready signal to 1. If extra delays are introduced by the interface circuitry before it places the data on the bus, the slave must delay the Slave-ready signal accordingly. The period $t_2 - t_1$ depends on the distance between the master and the slave and on the delays introduced by the slave's circuitry.

Buses-11 Asynchronous Bus

- *t*3—The Slave-ready signal arrives at the master, indicating that the input data are available on the bus. After a delay to the master loads the data into its register. Then, it drops the Master-ready signal, indicating that it has received the data.
- *t*4—The master removes the address and command information from the bus. The delay between *t*3 and *t*4 is again intended to allow for bus skew. Erroneous addressing may take place if the address, as seen by some device on the bus, starts to change while the Master-ready signal is still equal to 1.
- *t*5—When the device interface receives the 1-to-0 transition of the Master-ready signal, it removes the data and the Slave-ready signal from the bus. This completes the input transfer.

Buses-12 Asynchronous Bus

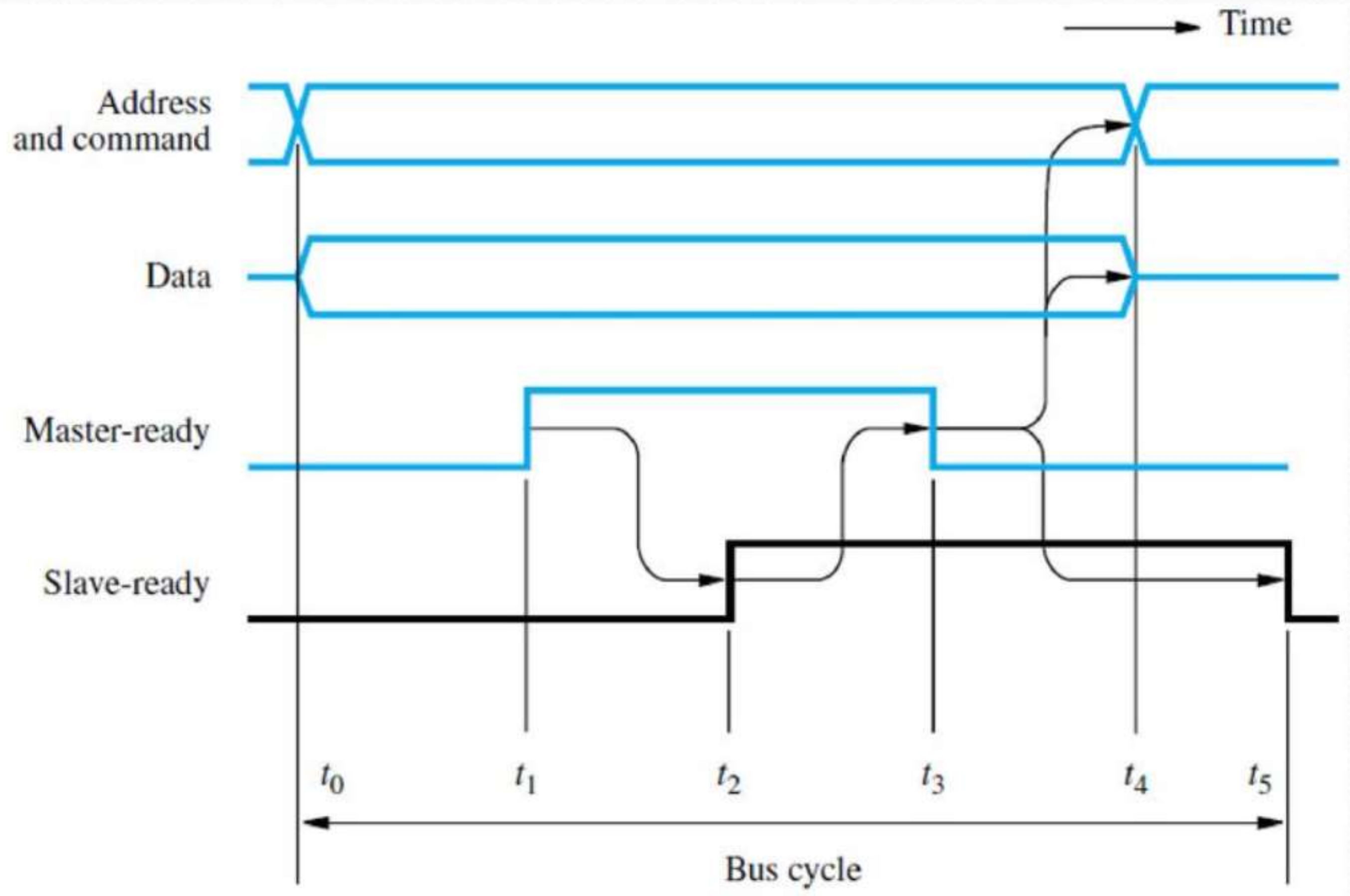


Figure 7.7 Handshake control of data transfer during an output operation.

Interface Circuits

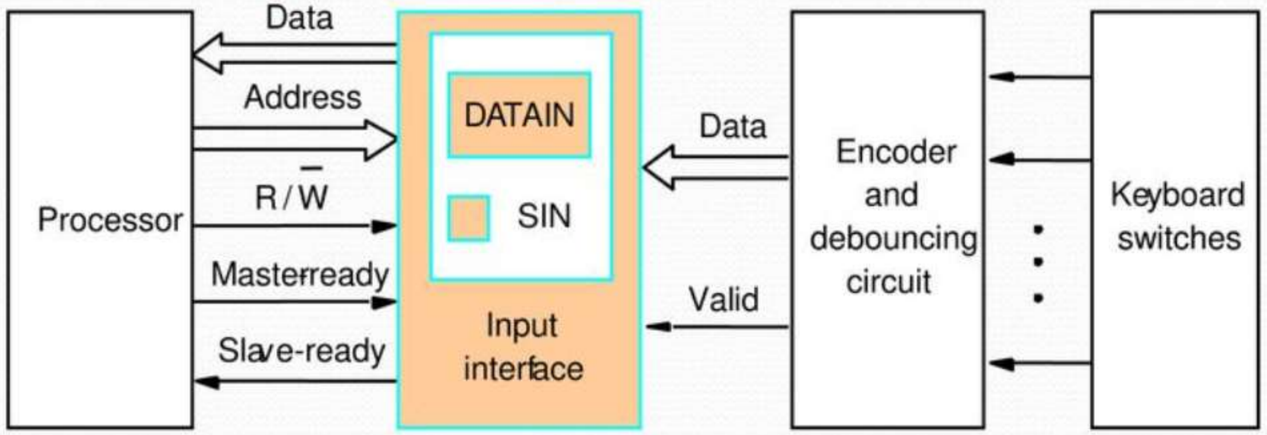
Interface circuits-1

- 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-2

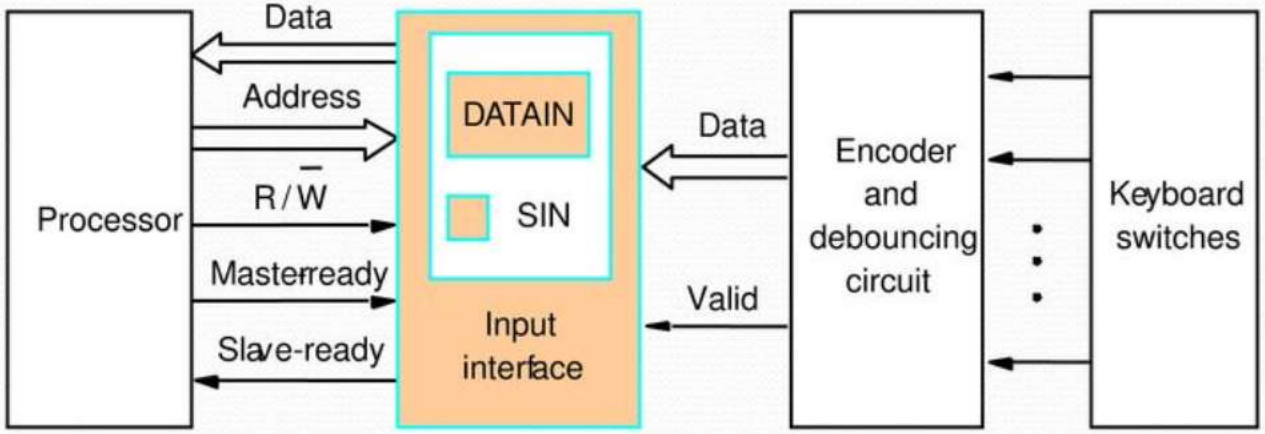
- Parallel port transfers data in the form of a number of bits, normally 8 or 16 to or from the device.
- Serial port transfers and receives data one bit at a time.
- Processor communicates with the bus in the same way, whether it is a parallel port or a serial port.
    - Conversion from the parallel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

Parallel port



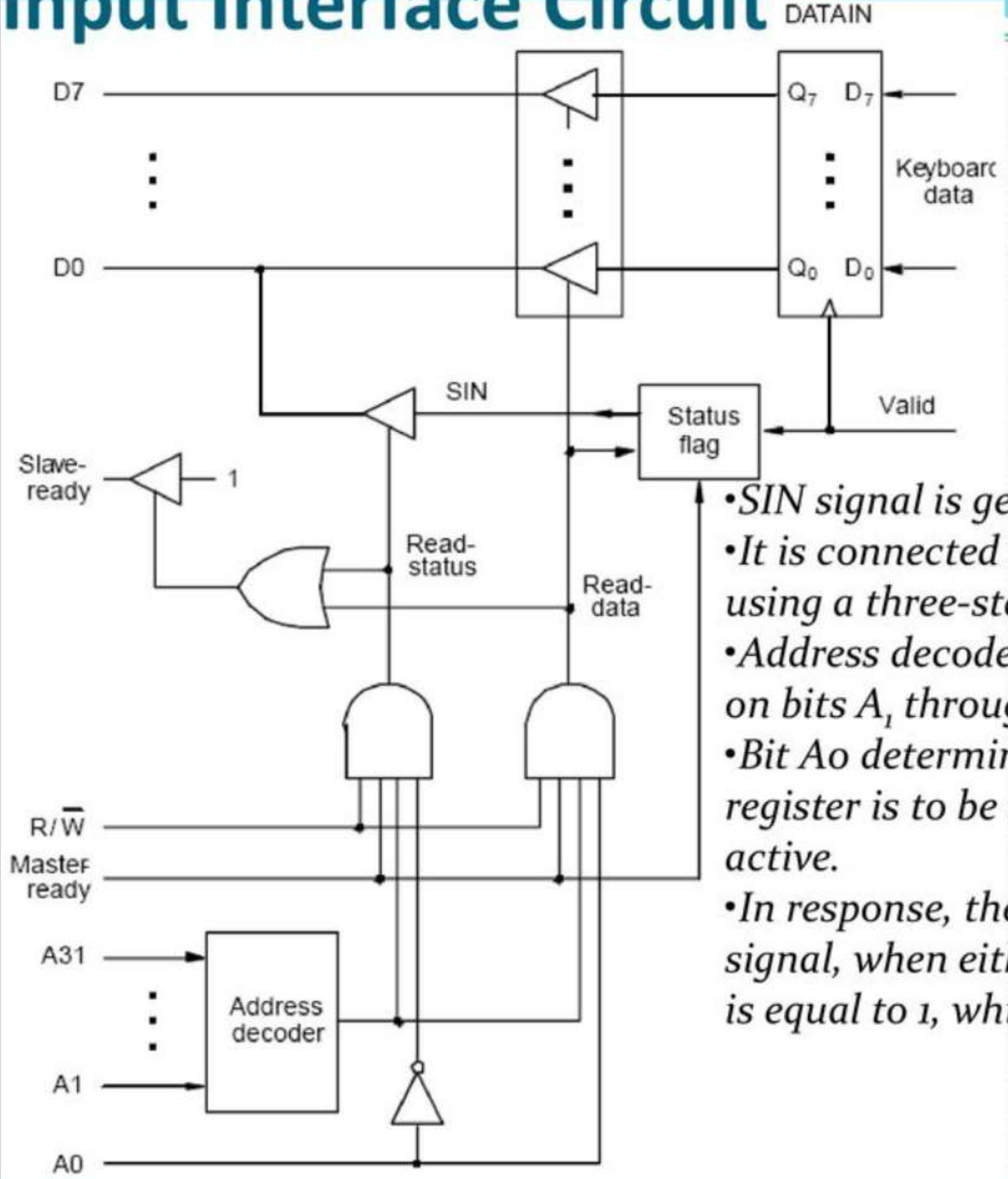
- *Keyboard is connected to a processor using a parallel port.*
- *Processor is 32-bits and uses memory-mapped I/O and the asynchronous bus protocol.*
- *On the processor side of the interface we have:*
  - *Data lines.*
  - *Address lines*
  - *Control or R/W line.*
  - *Master-ready signal and*
  - *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 DATAIN and SIN to be set to 1.*

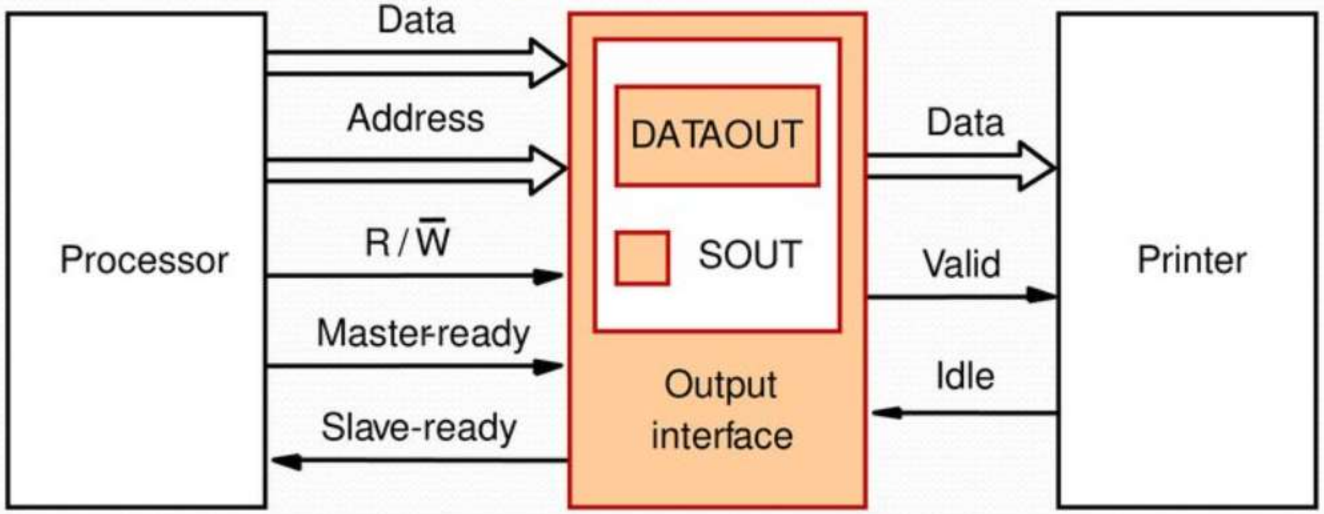
Input Interface Circuit



•Output lines of DATAIN are 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.

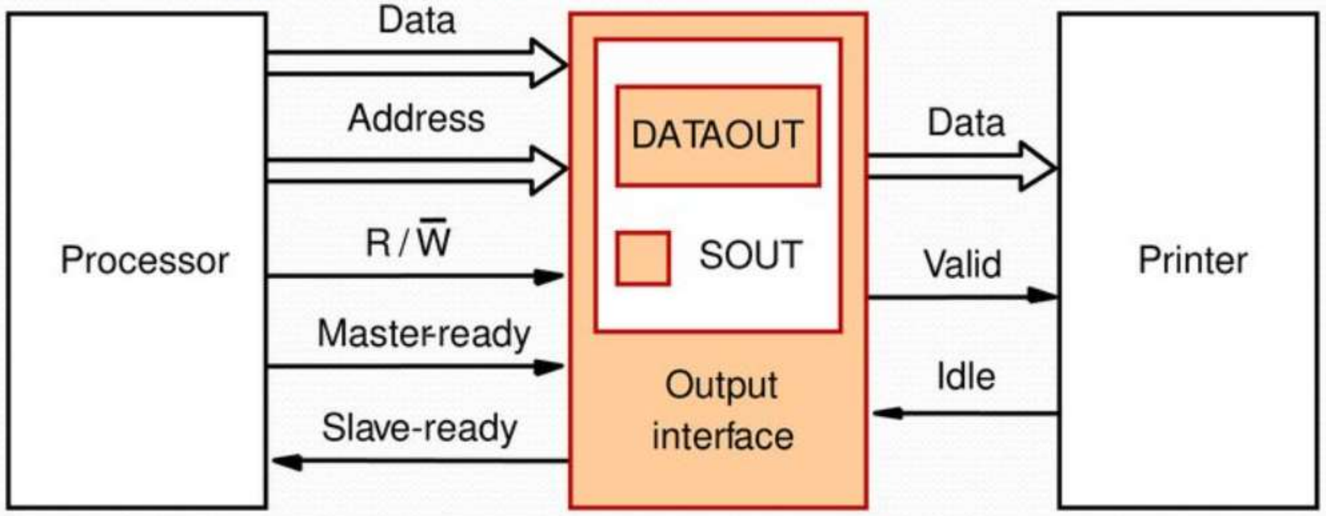
•SIN signal is generated using a status flag circuit.
•It is connected to line D_0 of the processor bus using a three-state driver.
•Address decoder selects the input interface based on bits A_1 through A_{31}.
•Bit Ao determines whether the status or data register is to be read, when Master-ready is 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.*

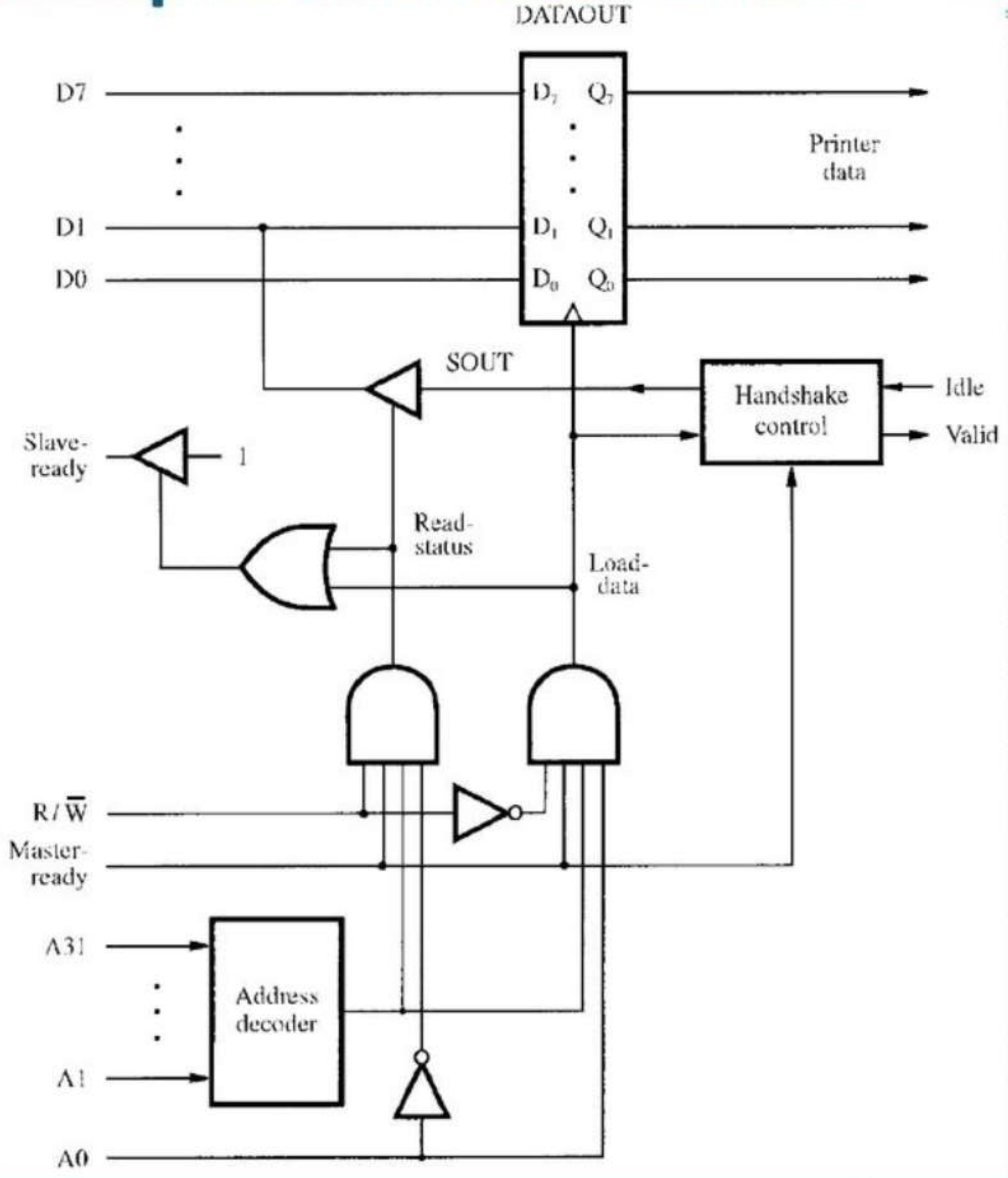
Parallel port (contd..)



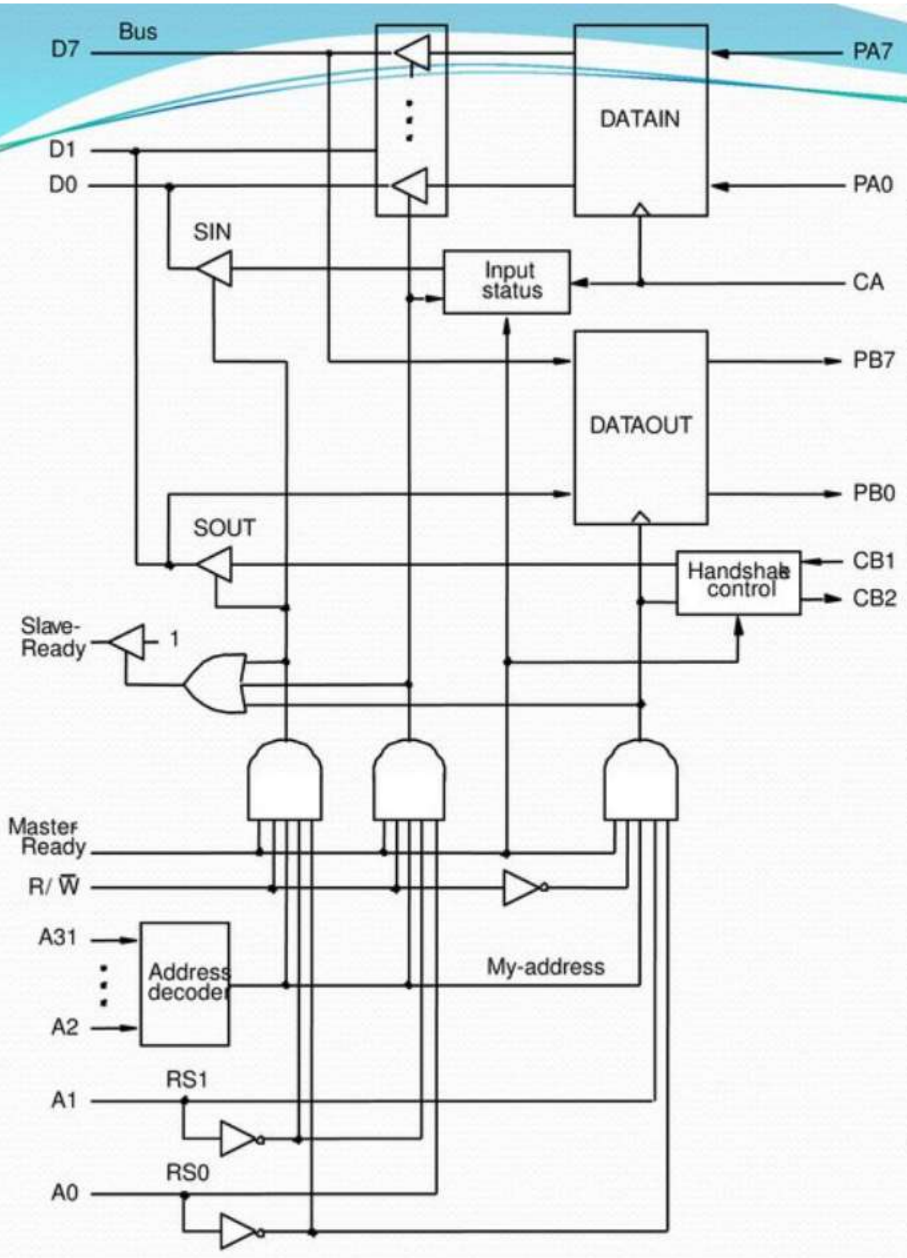
•*On the printer side:*

- *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.*

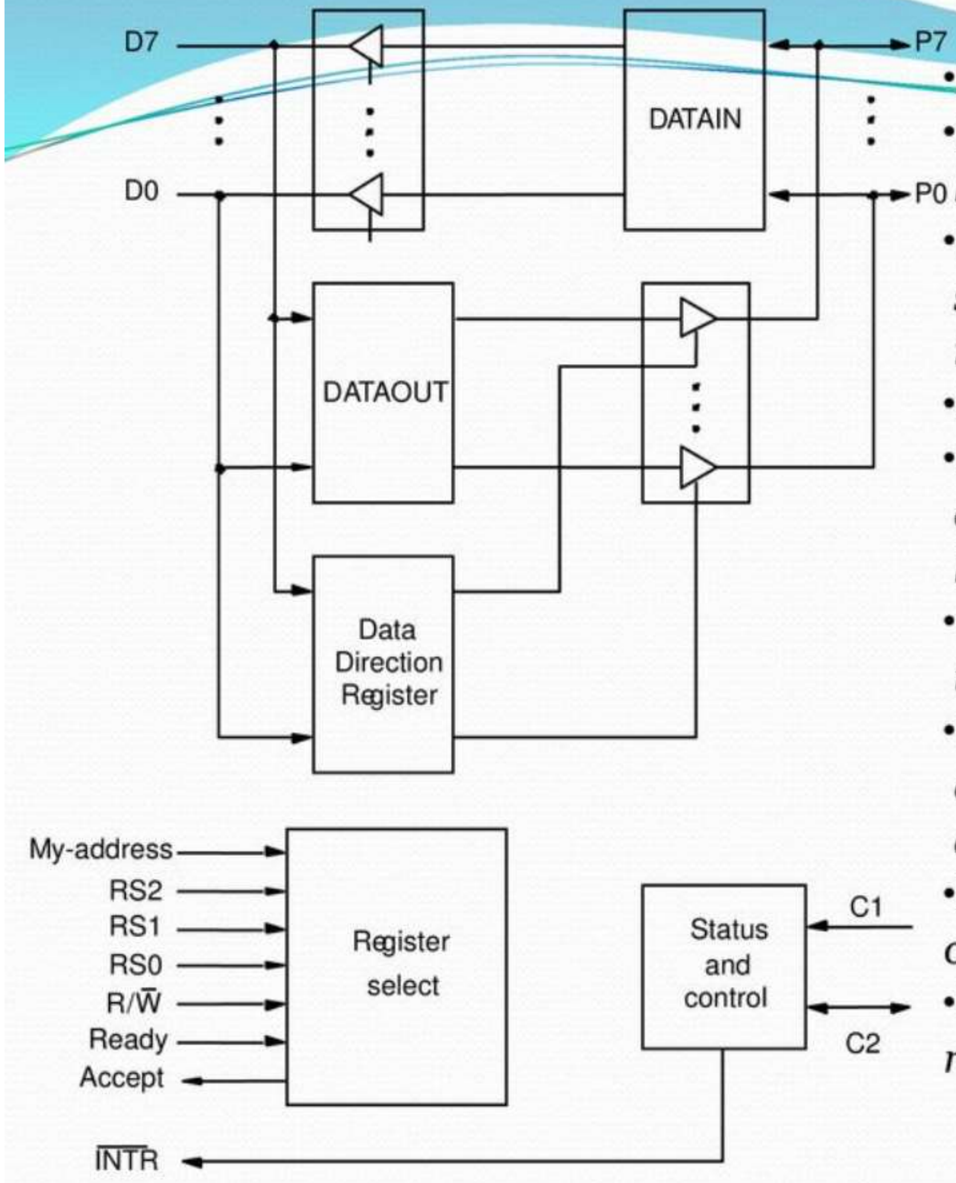
Output Interface Circuit



•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.



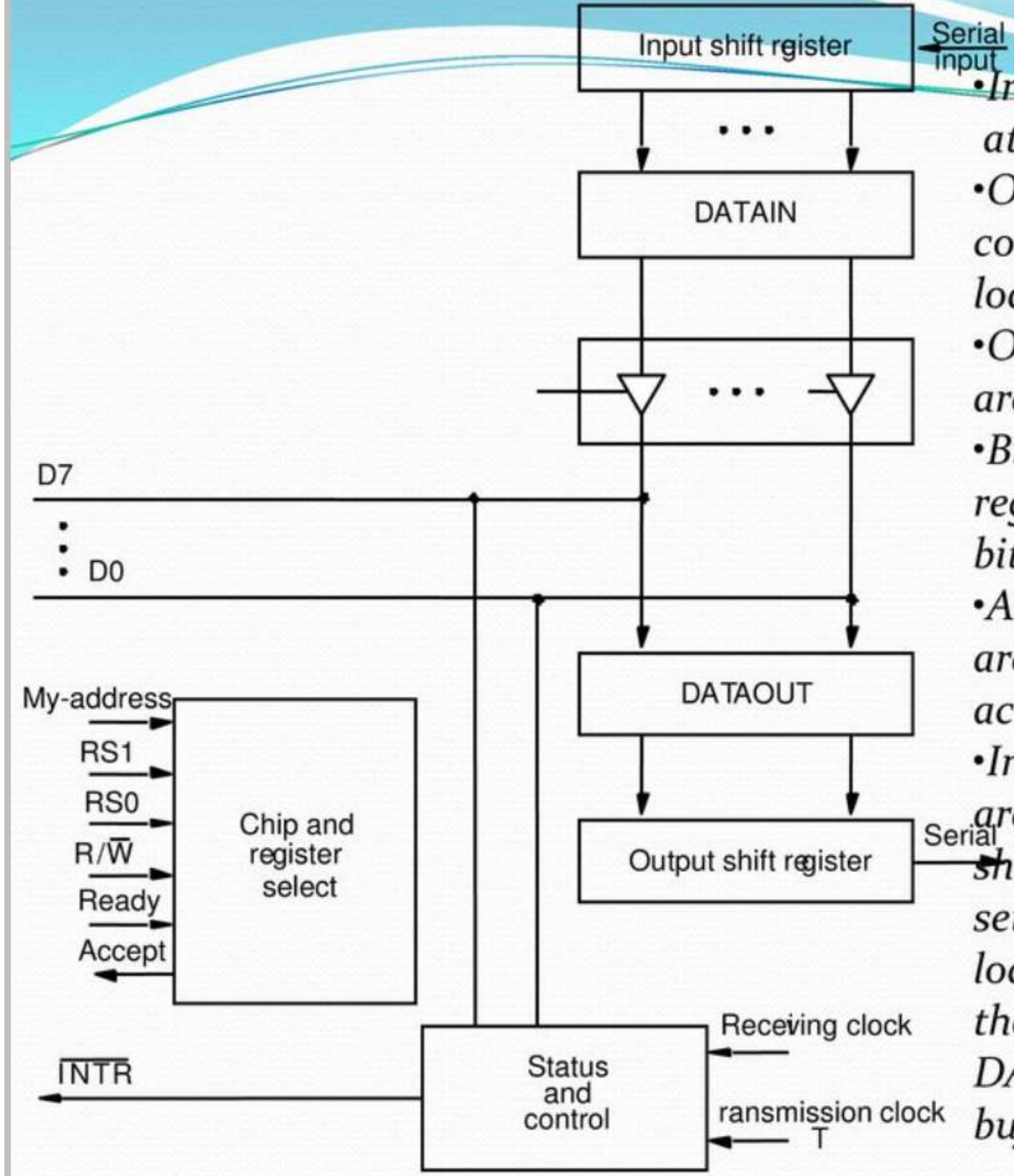
- *Combined I/O interface circuit.*
- *Address bits A2 through A31, that is 30 bits are used to select the overall interface.*
- *Address bits A1 through A0, that is, 2 bits select one of the three registers, namely, DATAIN, DATAOUT, and the status register.*
- *Status register contains the flags SIN 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.*



- *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 registers are both used at input so that the input 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.

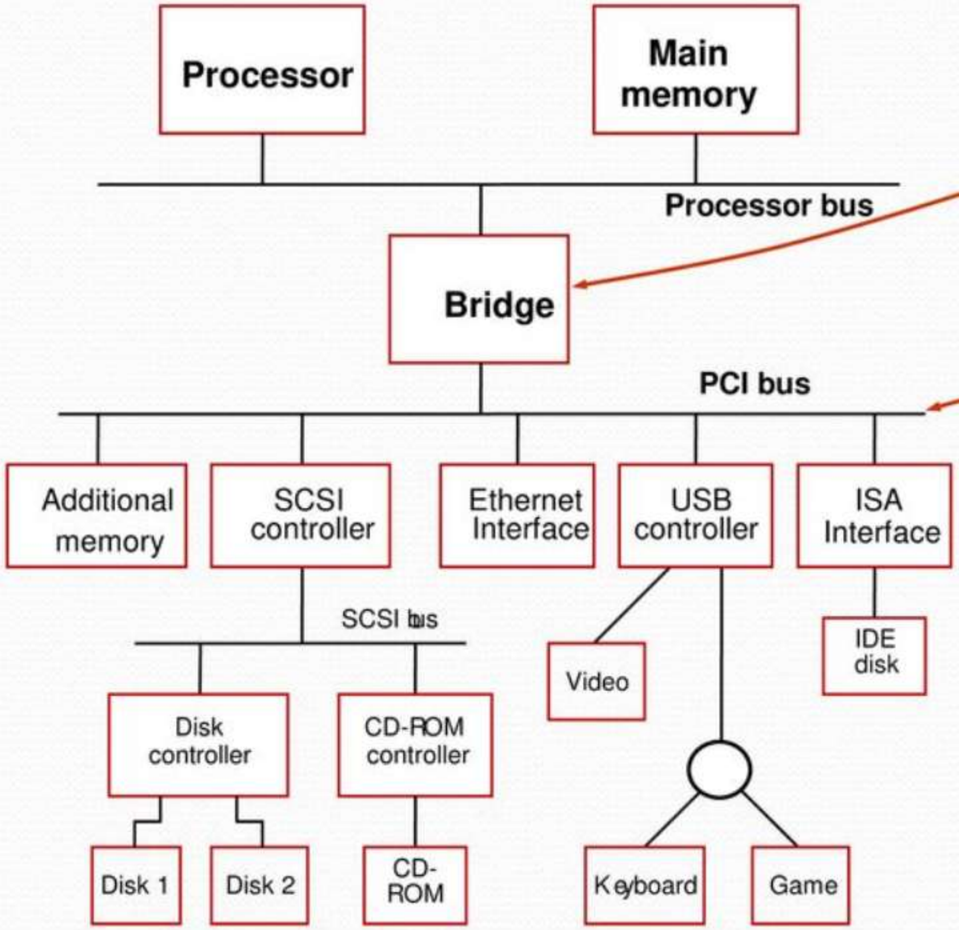
Standard I/O interfaces (contd..)

- 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..)

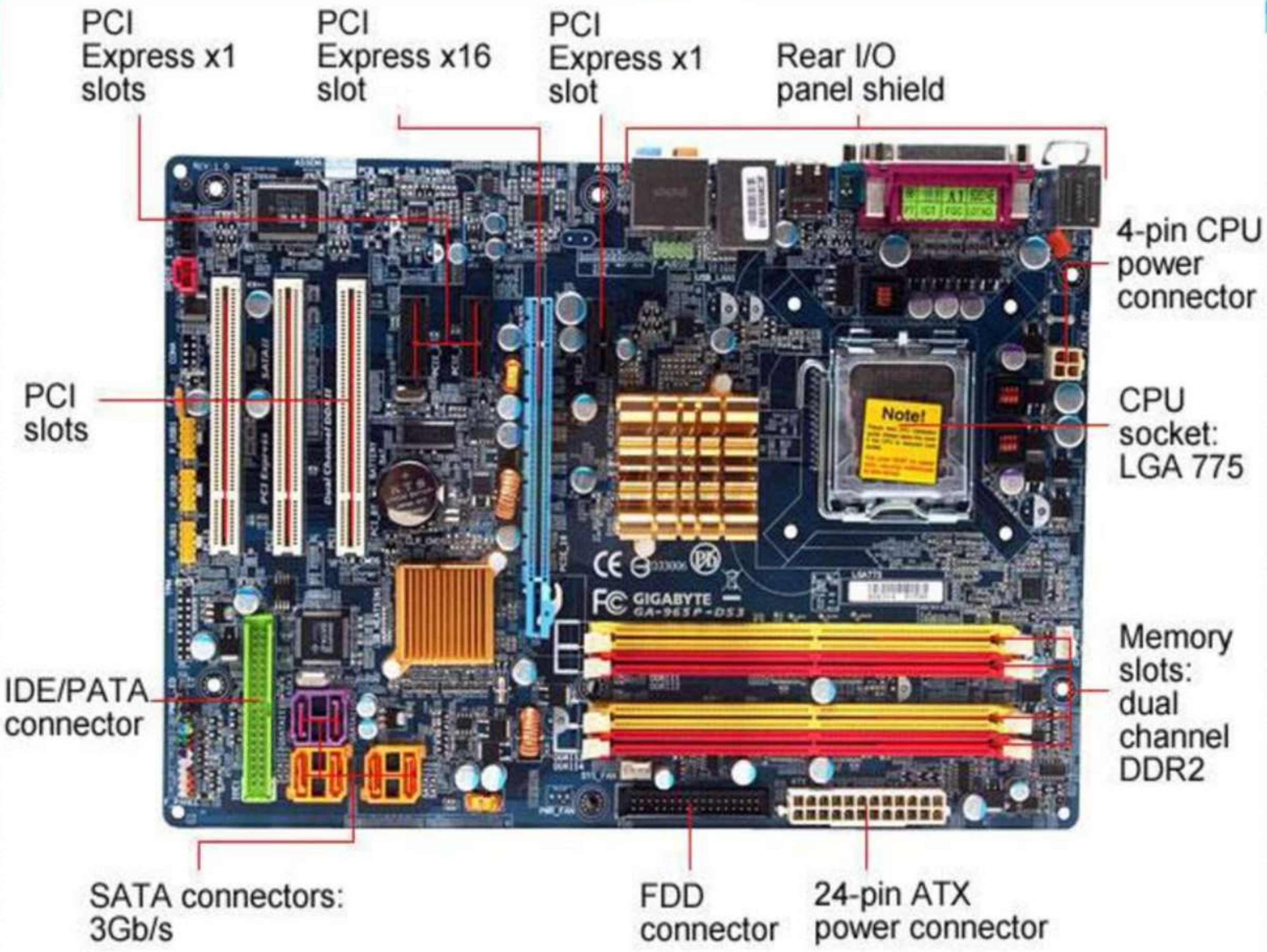
- A number of standards have been developed for the expansion bus.
  - Some have evolved by default.
  - For example, IBM's Industry Standard Architecture.
- Three widely used bus standards:
  - PCI (Peripheral Component Interconnect)
  - SCSI (Small Computer System Interface)
  - USB (Universal Serial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

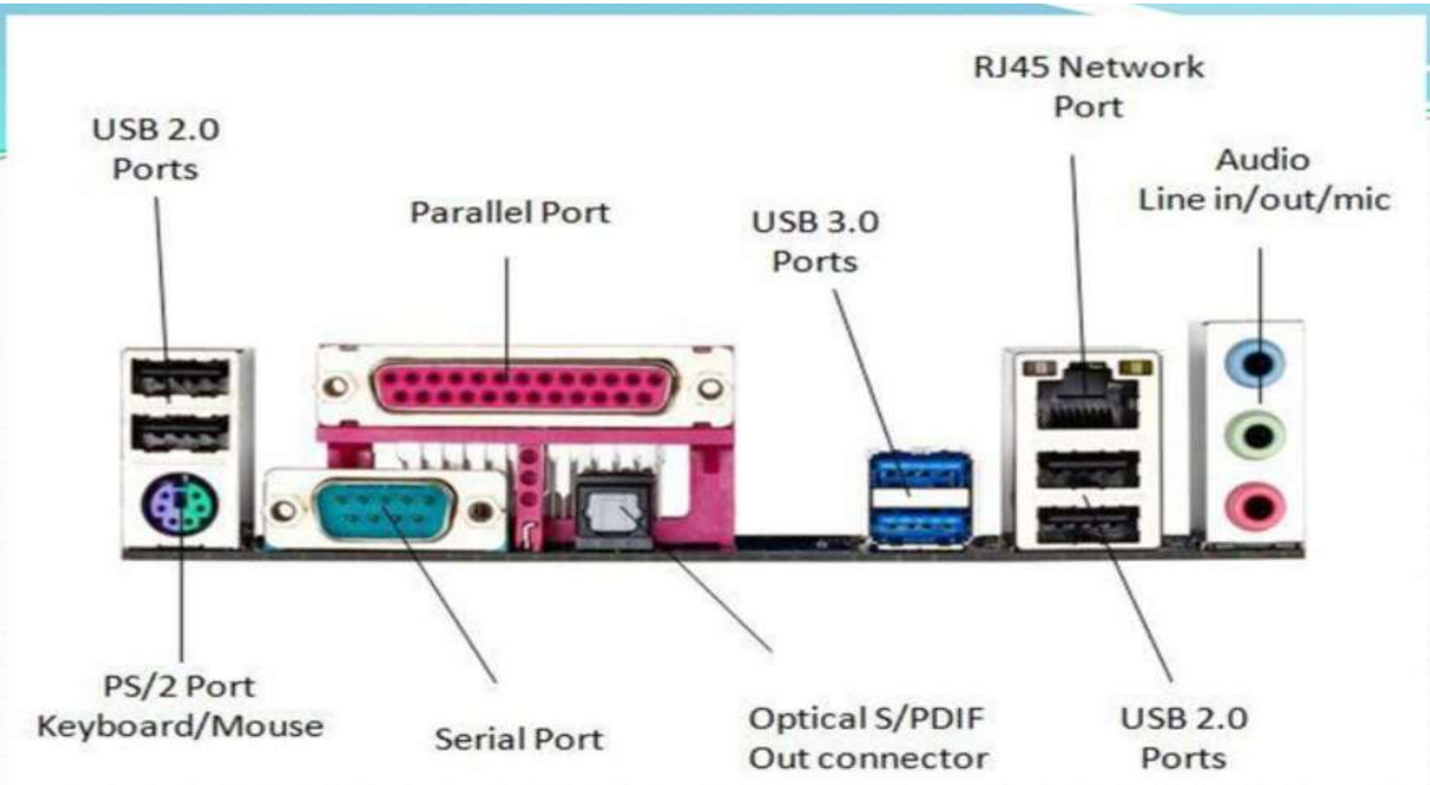
Standard I/O interfaces (contd..)

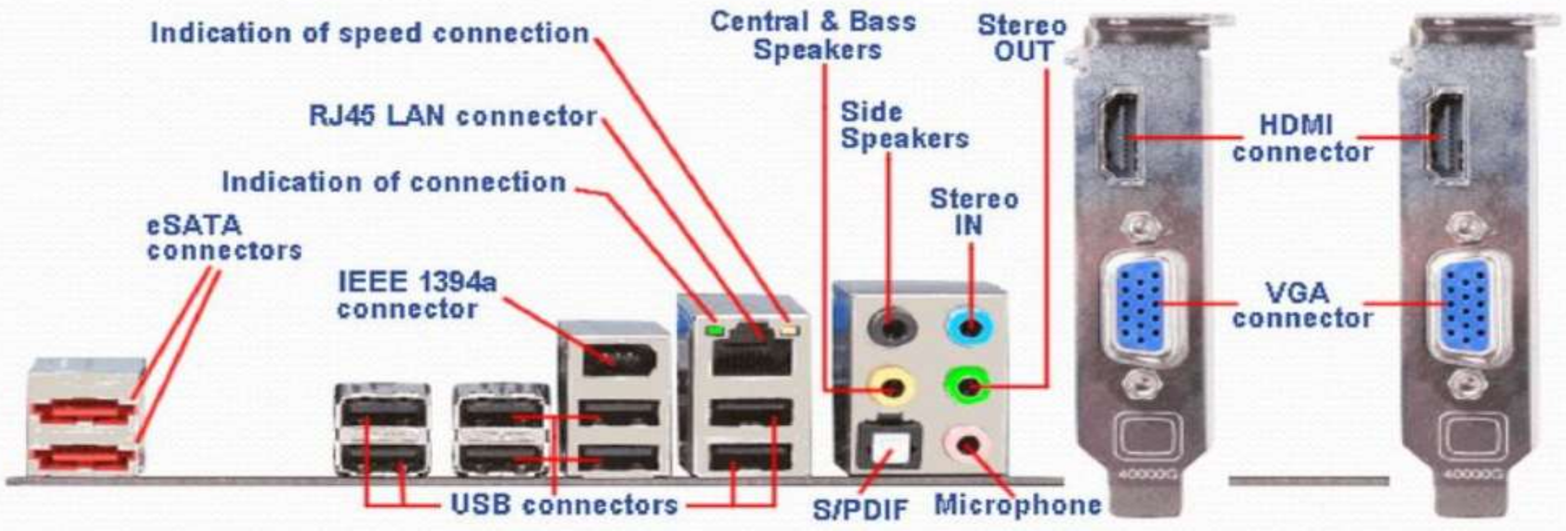


Bridge circuit translates signals and protocols from processor bus to PCI bus.

Expansion bus on the motherboard

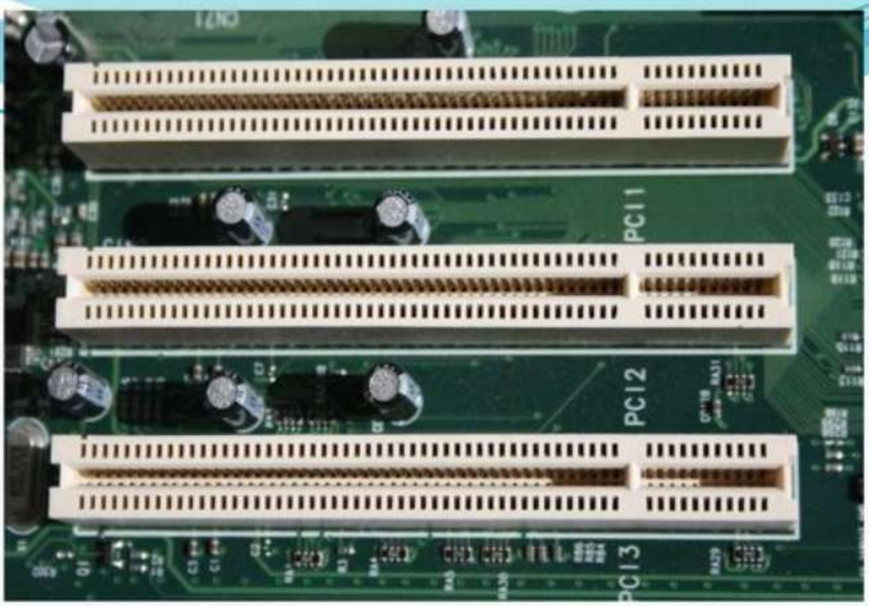

PCI Express x1 slots
PCI Express x16 slot
PCI Express x1 slot
Rear I/O panel shield
4-pin CPU power connector
PCI slots
CPU socket: LGA 775
IDE/PATA connector
Memory slots: dual channel DDR2
SATA connectors: 3Gb/s
FDD connector
24-pin ATX power connector


USB 2.0 Ports
Parallel Port
USB 3.0 Ports
RJ45 Network Port
Audio Line in/out/mic
PS/2 Port Keyboard/Mouse
Serial Port
Optical S/PDIF Out connector
USB 2.0 Ports


Indication of speed connection
RJ45 LAN connector
Indication of connection
eSATA connectors
IEEE 1394a connector
Central & Bass Speakers
Side Speakers
Stereo OUT
Stereo IN
HDMI connector
VGA connector
USB connectors
S/PDIF
Microphone

PCI BUS
(PERIPHERAL COMPONENT INTERCONNECT)

PCI Bus

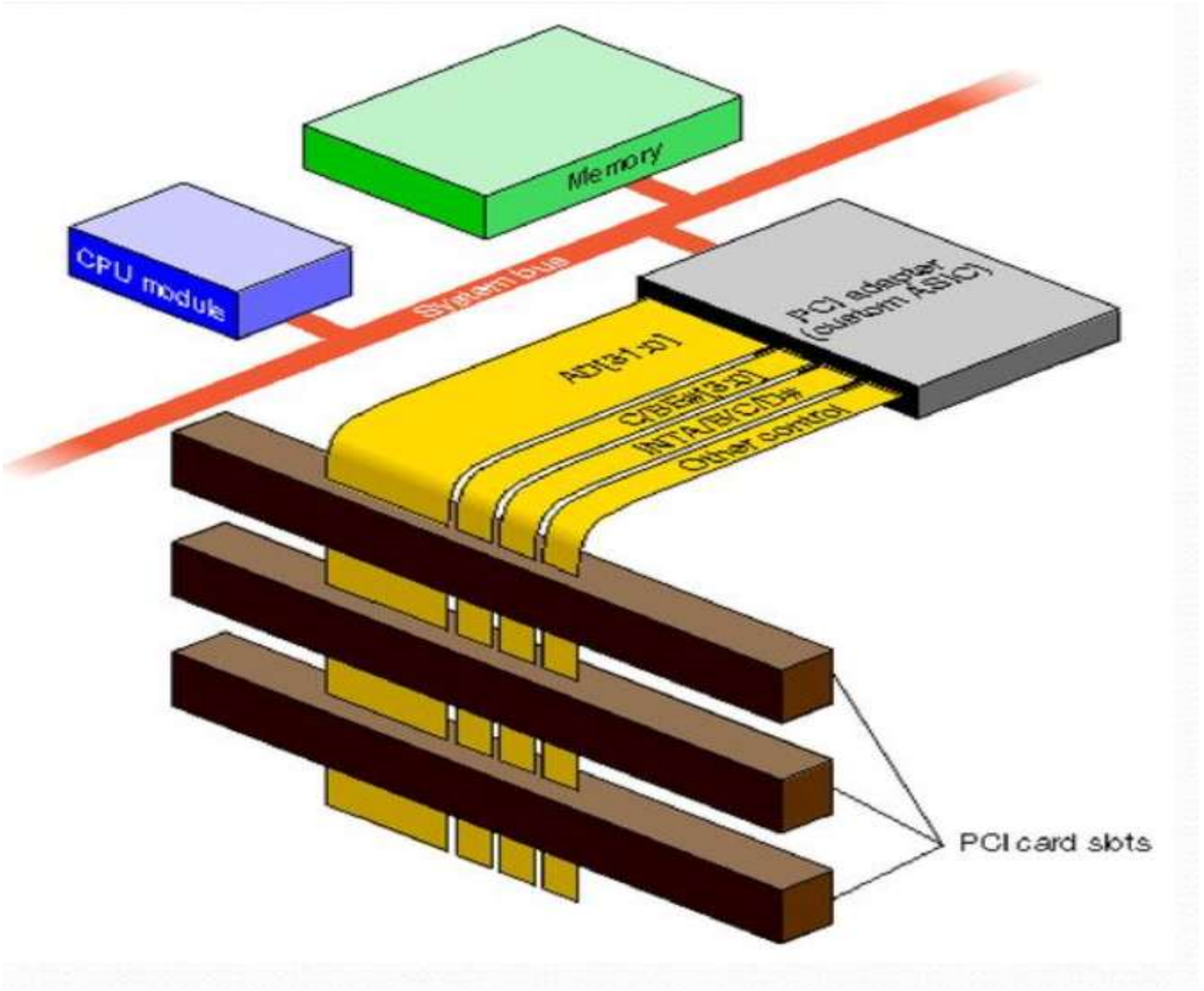


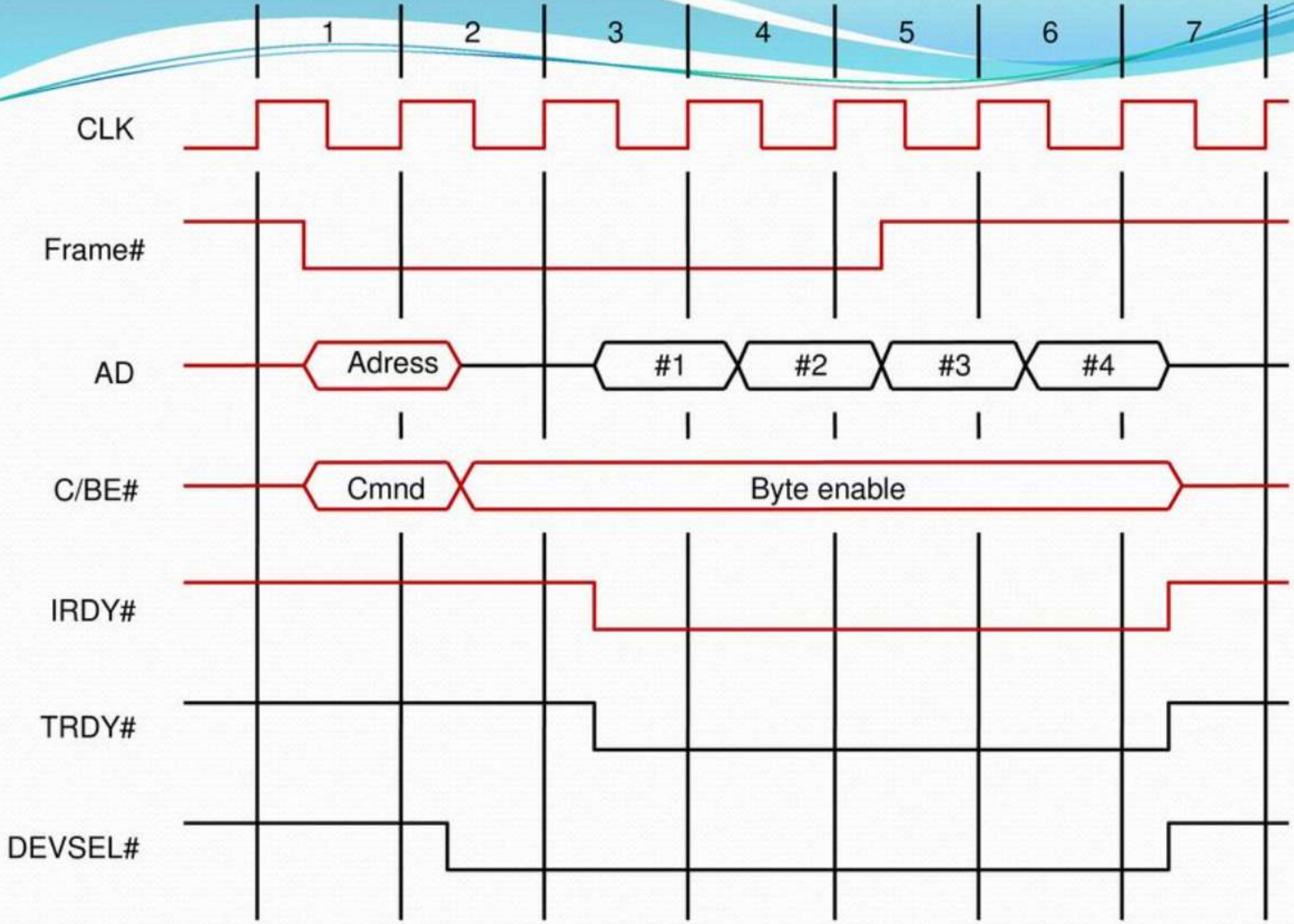
- *Peripheral Component Interconnect*
- Introduced in 1992
- Low-cost bus
- Processor independent
- Plug-and-play capability
- In today's computers, most memory transfers involve a burst of data rather than just one word. The PCI is designed primarily to support this mode of operation.
- The bus supports 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.

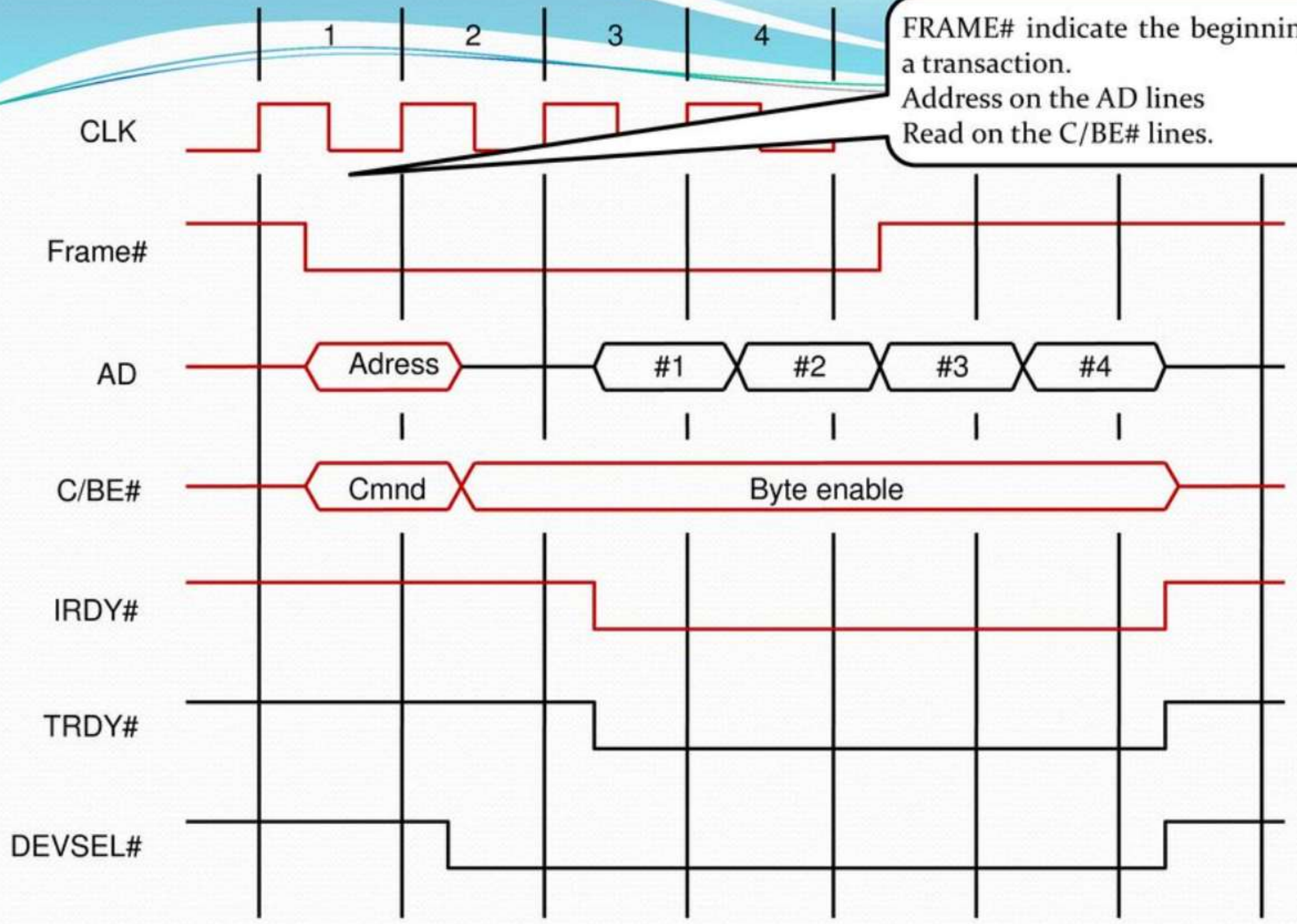
Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus).
IRD Y#, TRD Y#	Initiator-ready and Target-ready signals.
DEVSEL#	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 signal whose name ends with the symbol # is asserted when in the low voltage state.


Memory
CPU module
System bus
PCI adapter (custom ASIC)
AD[31:0]
C/BE#[3:0]
INTA/B/C/D#
Other control
PCI card slots

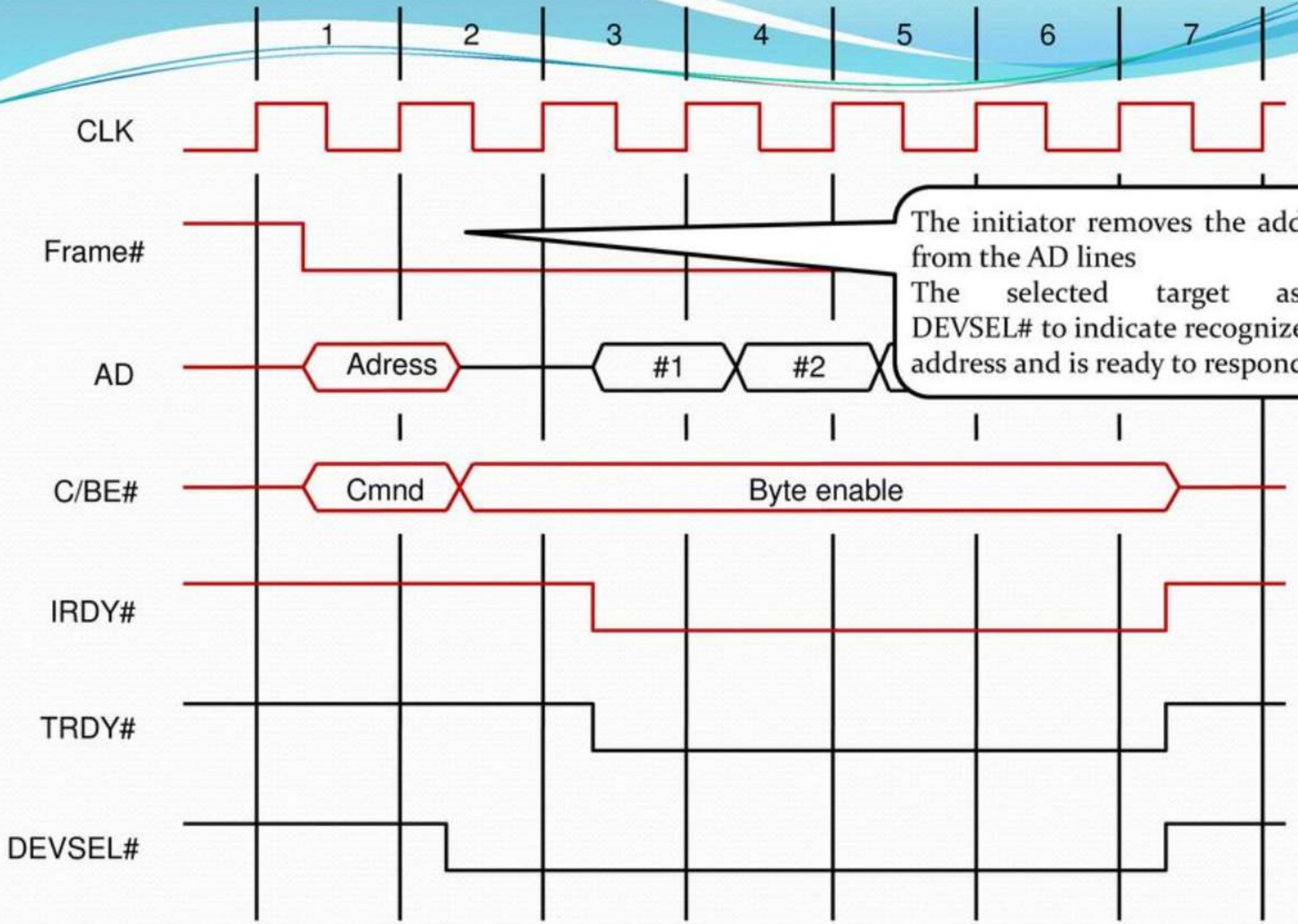


A read operation on the PCI bus



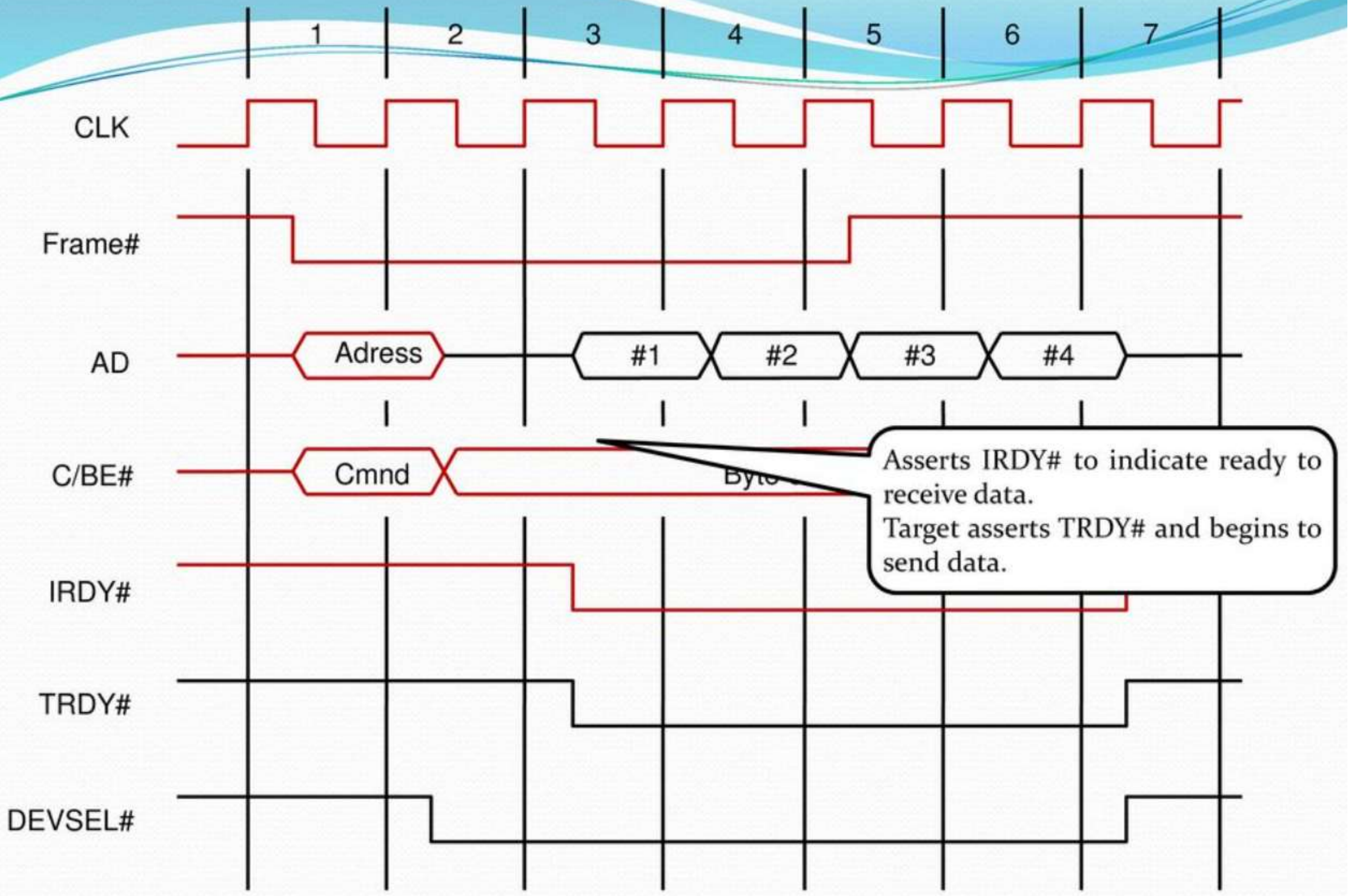
FRAME# indicate the beginning of a transaction.
Address on the AD lines
Read on the C/BE# lines.

A read operation on the PCI bus



The initiator removes the address, from the AD lines
The selected target asserts DEVSEL# to indicate recognized its address and is ready to respond

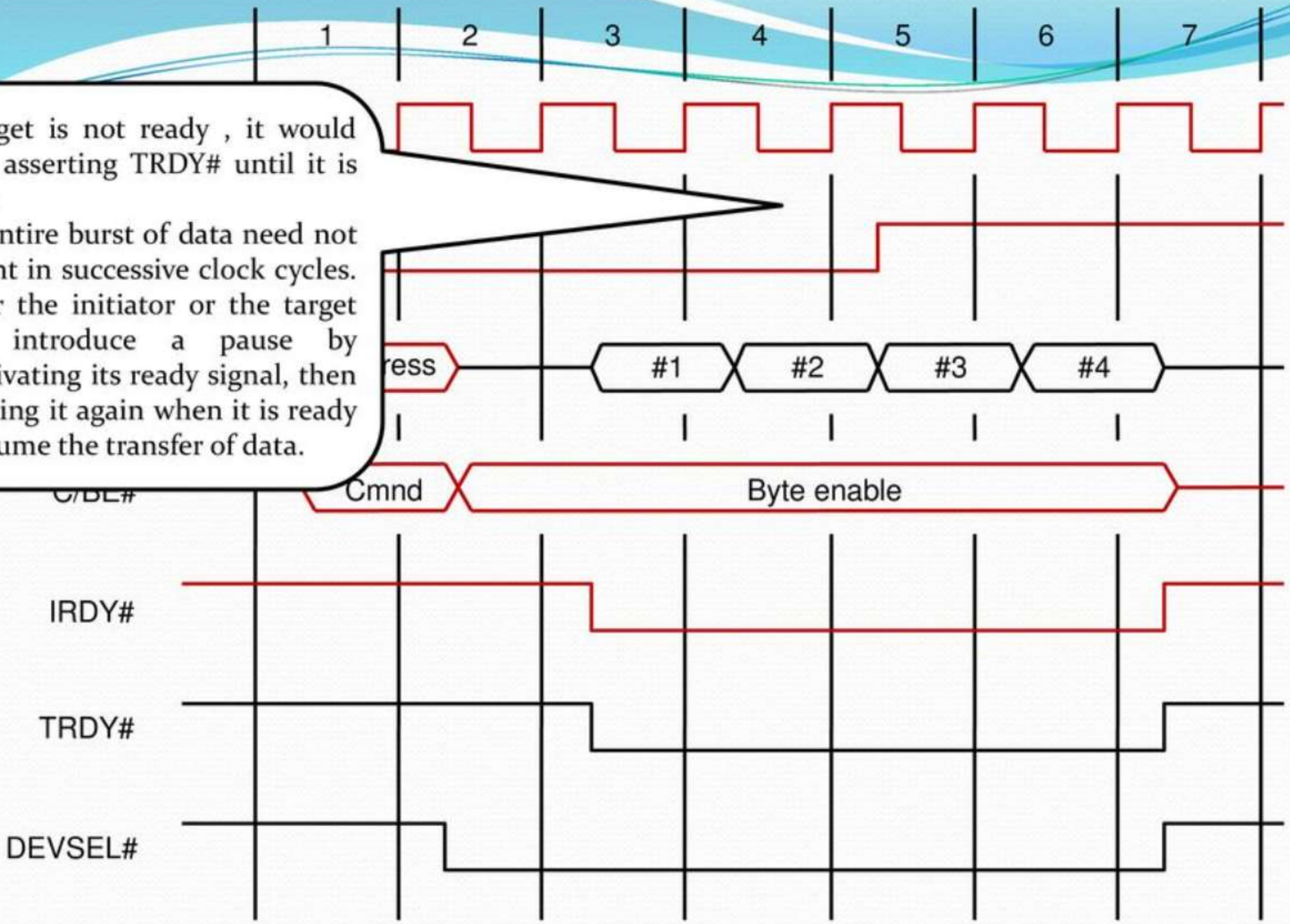
A read operation on the PCI bus



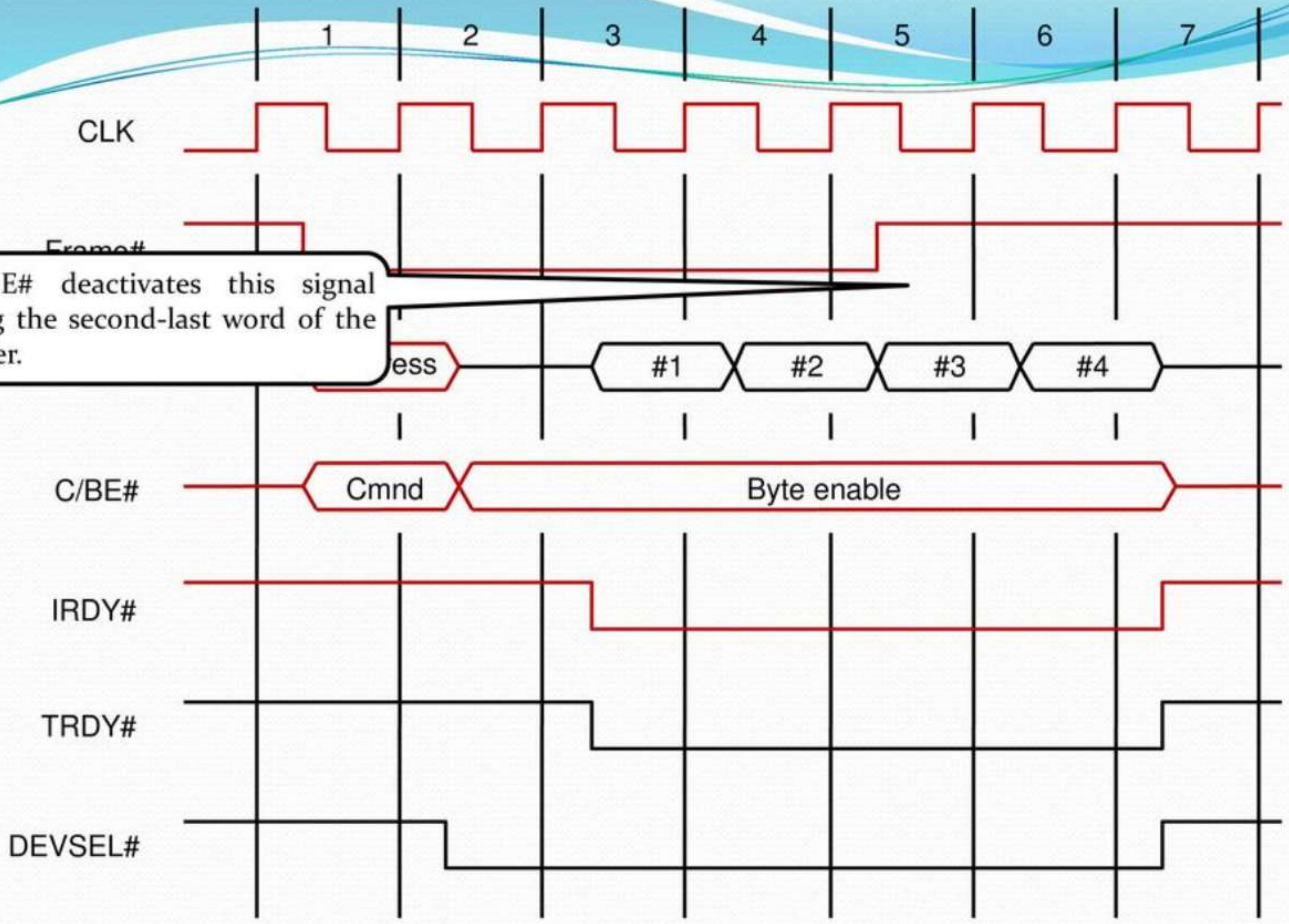
Asserts IRDY# to indicate ready to receive data.
Target asserts TRDY# and begins to send data.

A read operation on the PCI bus

If target is not ready , it would delay asserting TRDY# until it is ready.
The entire burst of data need not be sent in successive clock cycles. Either the initiator or the target may introduce a pause by deactivating its ready signal, then asserting it again when it is ready to resume the transfer of data.

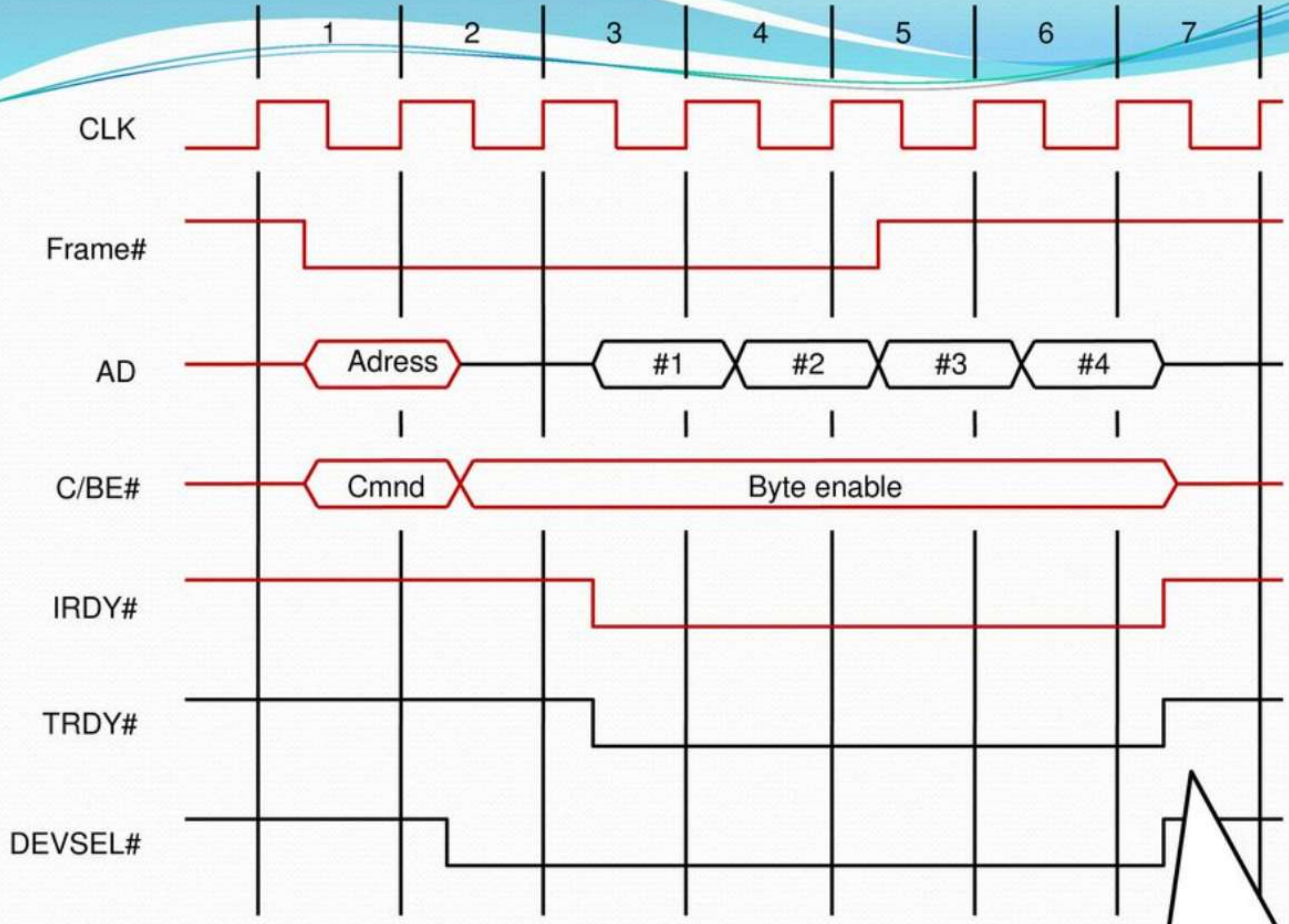


A read operation on the PCI bus



FRAME# deactivates this signal during the second-last word of the transfer.

A read operation on the PCI bus



Target sends one more word then stops. After sending the fourth word, the target deactivates TRDY# and DEVSEL# and disconnects its drivers on the AD lines.

A read operation on the PCI

Device Configuration

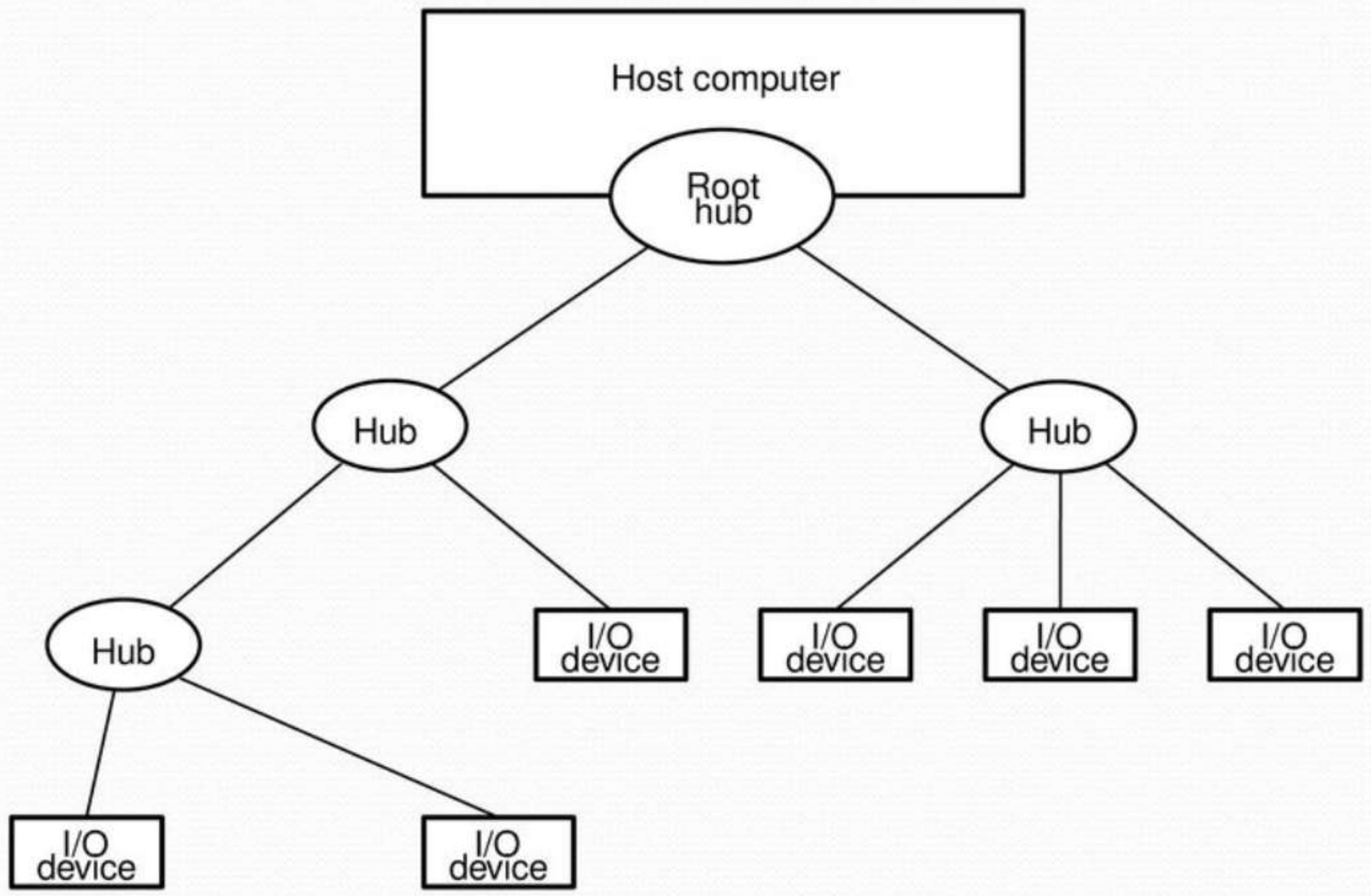
- When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.
- PCI incorporates in each I/O device interface a small configuration ROM memory that stores information about that device.
- The configuration ROMs of 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 bout 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

USB
(UNIVERSIAL SERIAL BUS)

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

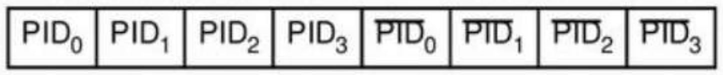
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.
- Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root 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

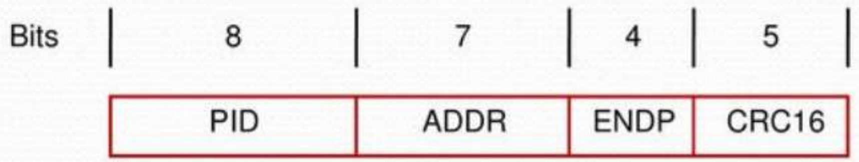
- When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the 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.

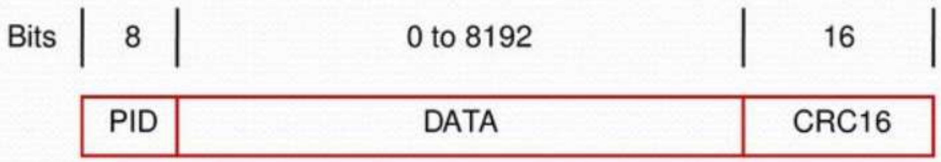


(a) Packet identifier field



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

(b) Token packet, IN or OUT



(c) Data packet

Figure 45. USB packet format.

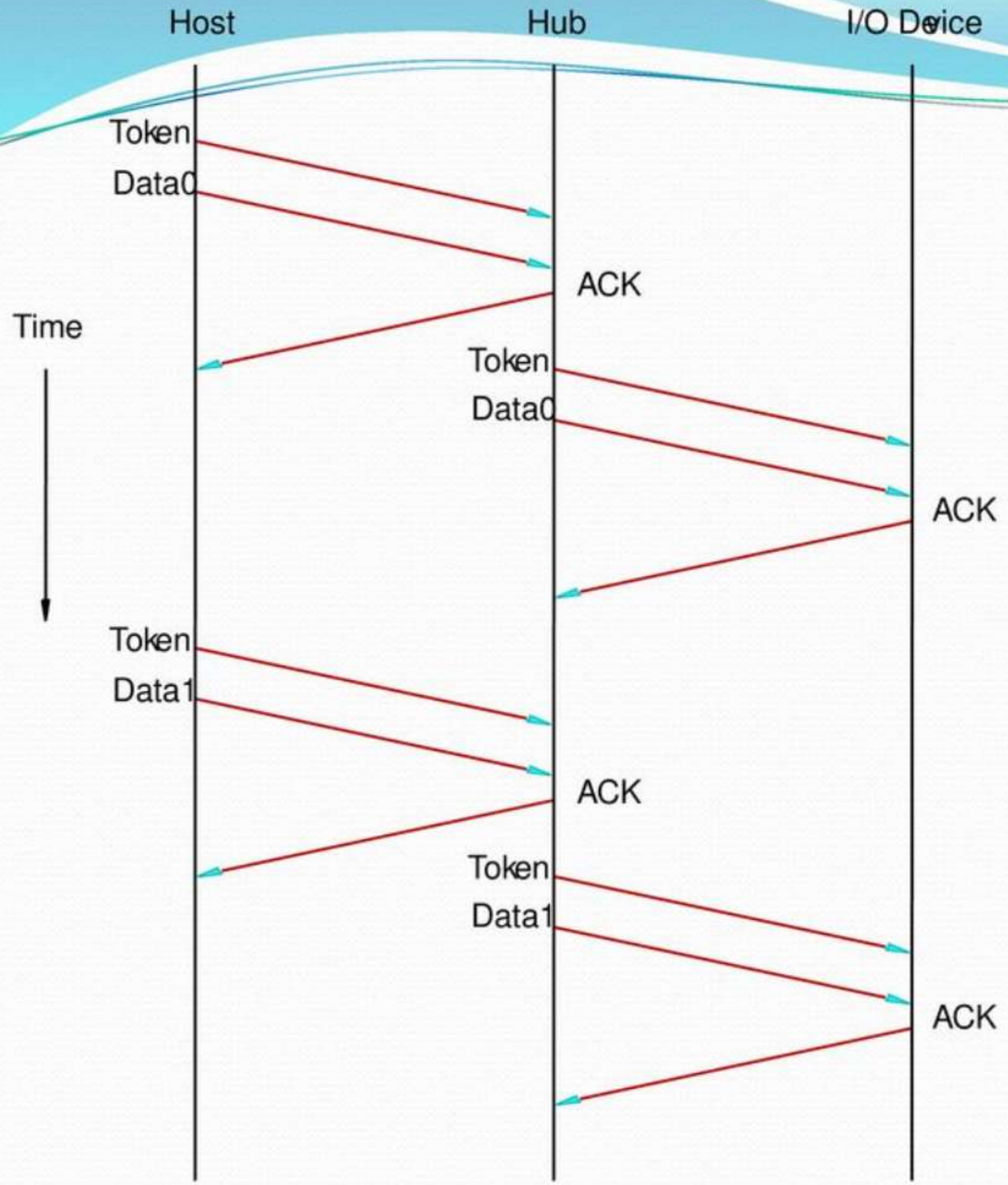
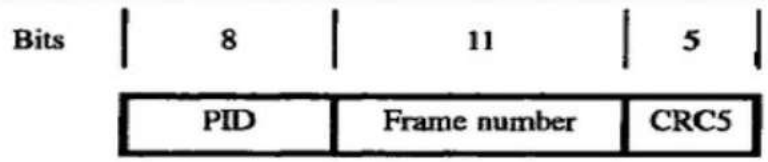


Figure: An output transfer

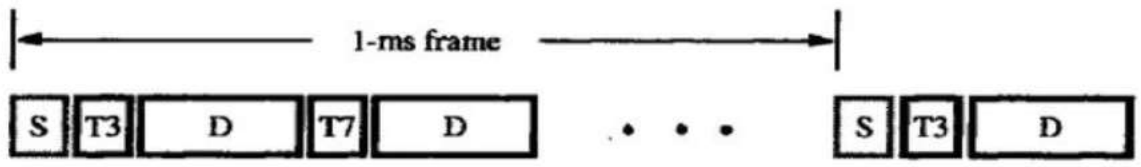
Isochronous Traffic on USB

- One of the key objectives of the USB is to support the transfer of isochronous data.
- Devices that generates or receives 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.

USB Frames



(a) SOF Packet



S — Start-of-frame packet

T*n*— Token packet, address = *n*

D — Data packet

A — ACK packet

(b) Frame example

Figure 4.47 USB frames.

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 o the two signal wires. For high-speed links, differential transmission is used.

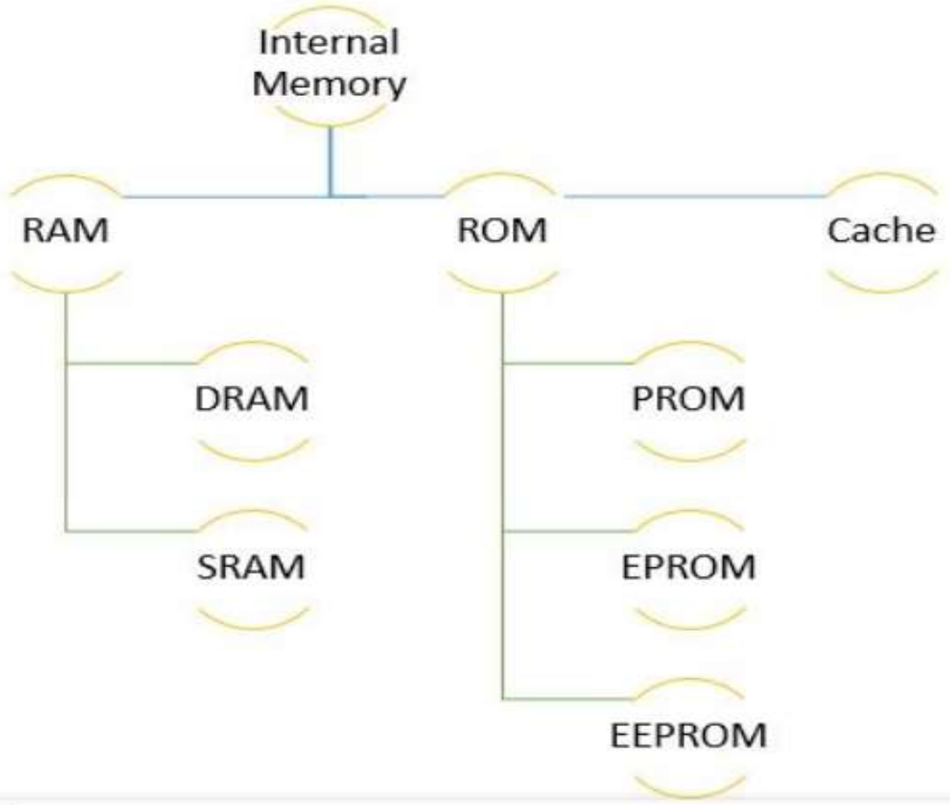
END

UNIT-IV

THE MEMORY SYSTEMS

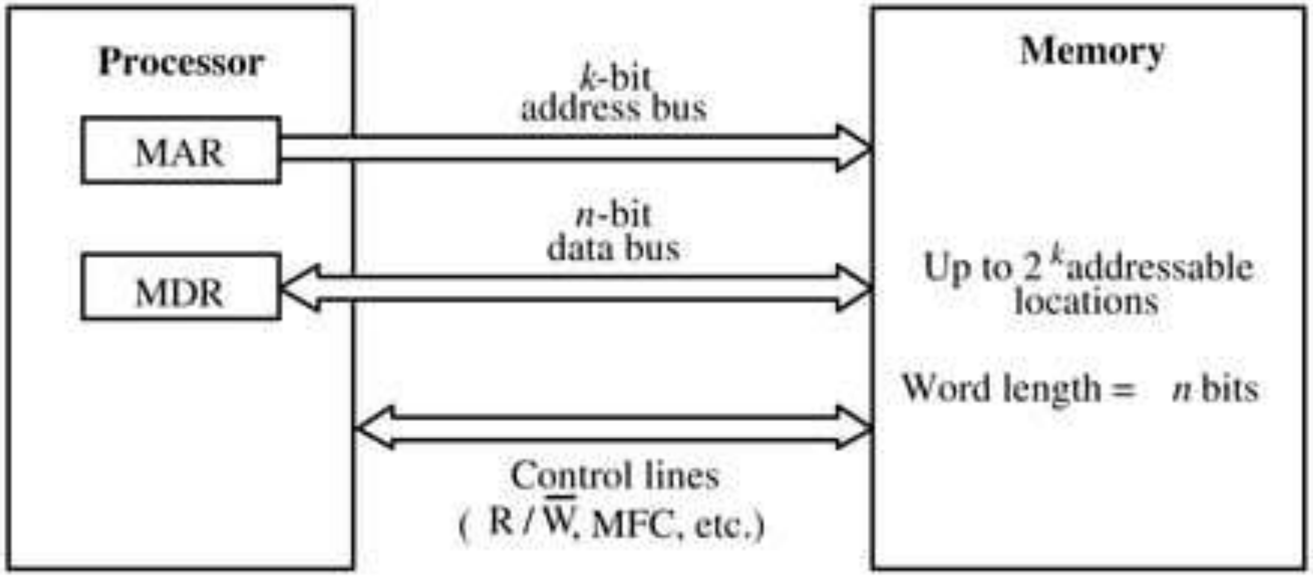
Types of Internal Memory

The internal memory of a computer can be classified as RAM, ROM, and cache memory.



Some basic concepts

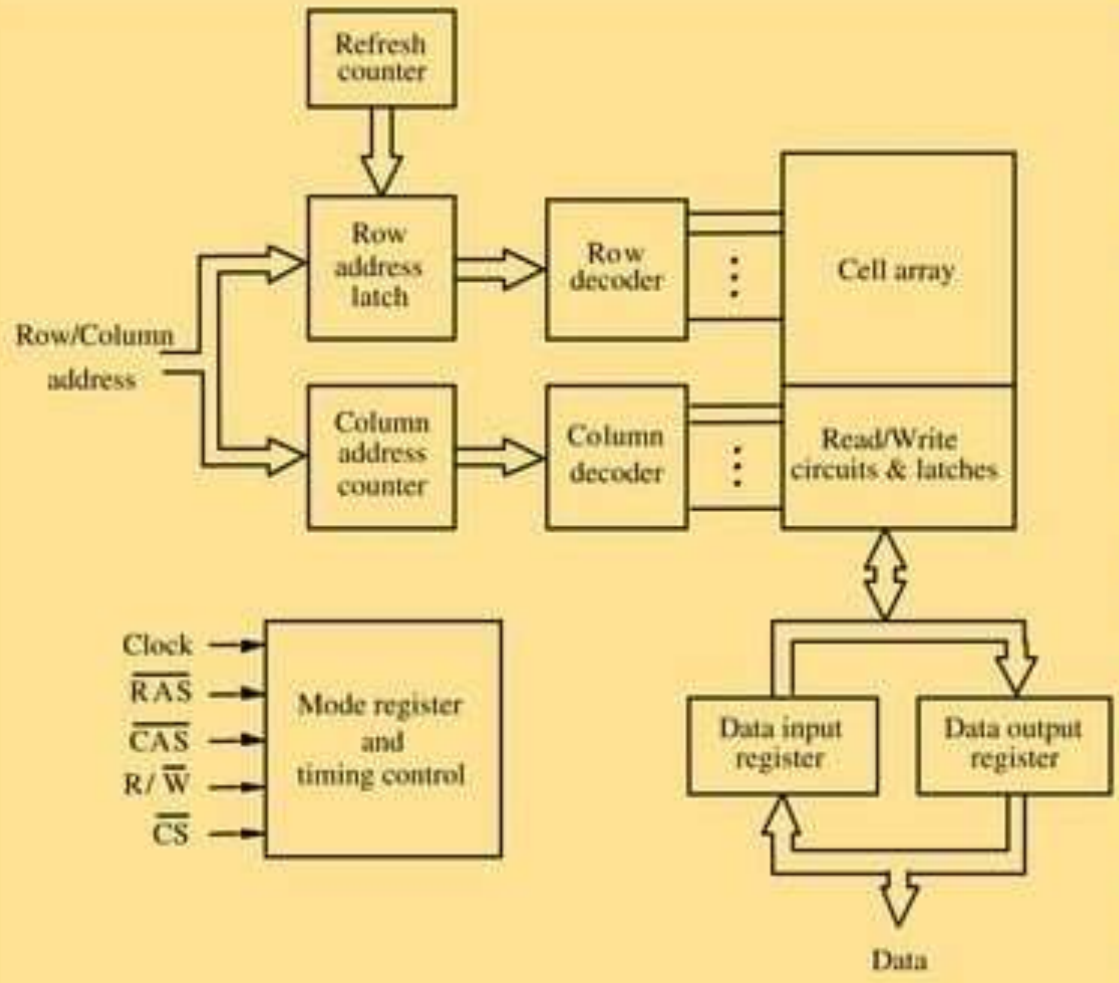
- Maximum size of the Main Memory
- byte-addressable
- CPU-Main Memory Connection



Some basic concepts(Contd.,)

- Measures for the speed of a memory:
  - memory access time. - Time that lapses b/w initiation n completion of operation
  - memory cycle time. – Minimum time delay b/w initiation of 2 successive memory operation
- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.
- Several techniques to increase the effective size and speed of the memory:
  - Cache memory (to increase the effective speed).
  - Virtual memory (to increase the effective size).

Synchronous DRAMs



- *DRAMs whose operation is directly synchronized with a clock signal*
- *The outputs of the sense circuits are connected to a latch.*
- *During a Read operation, the contents of the cells in a row are loaded onto the latches.*
- *During a refresh operation, the contents of the cells are refreshed without changing the contents of the latches.*
- *Data held in the latches correspond to the selected columns are transferred to the output.*
- *For a burst mode of operation, successive columns are selected using column address counter and clock. CAS signal need not be generated externally. A new data is placed during raising edge of the clock*

Synchronous DRAMs

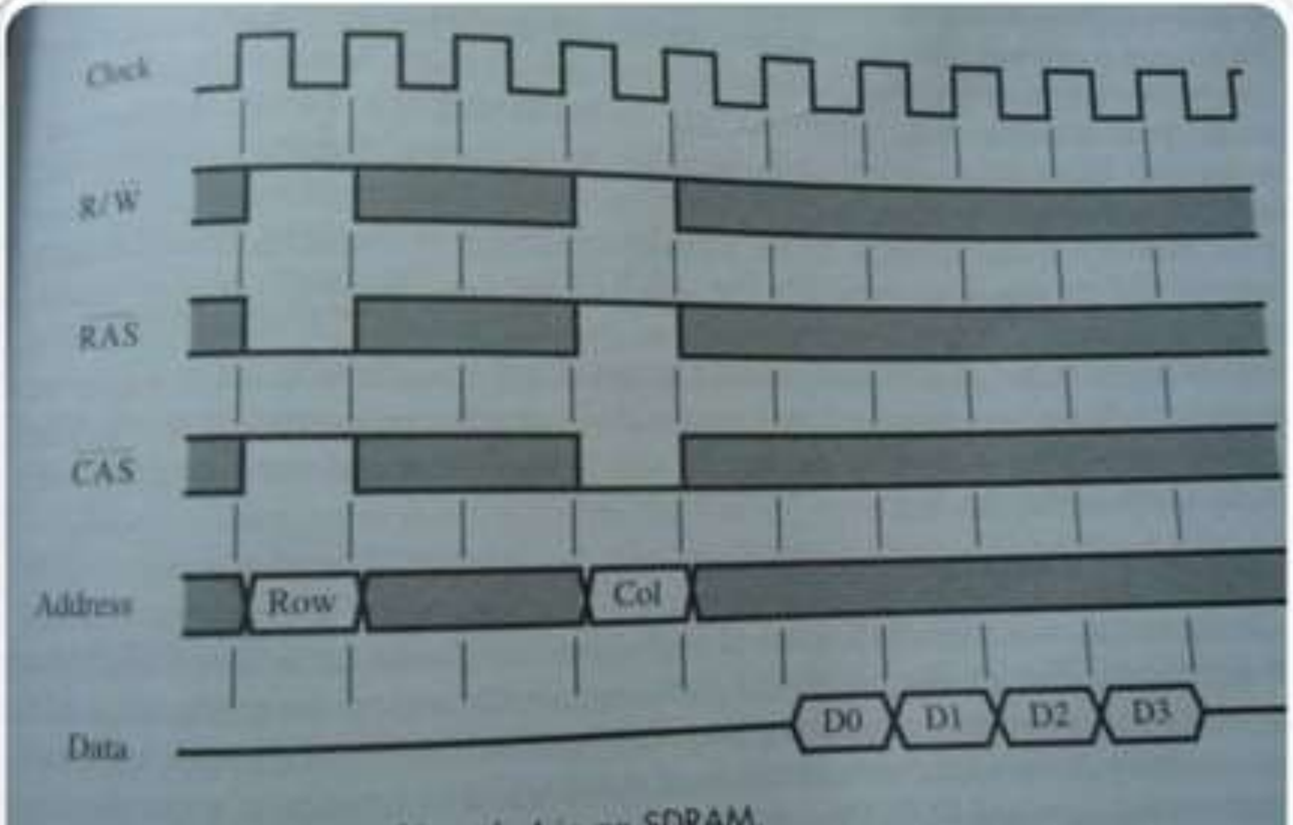
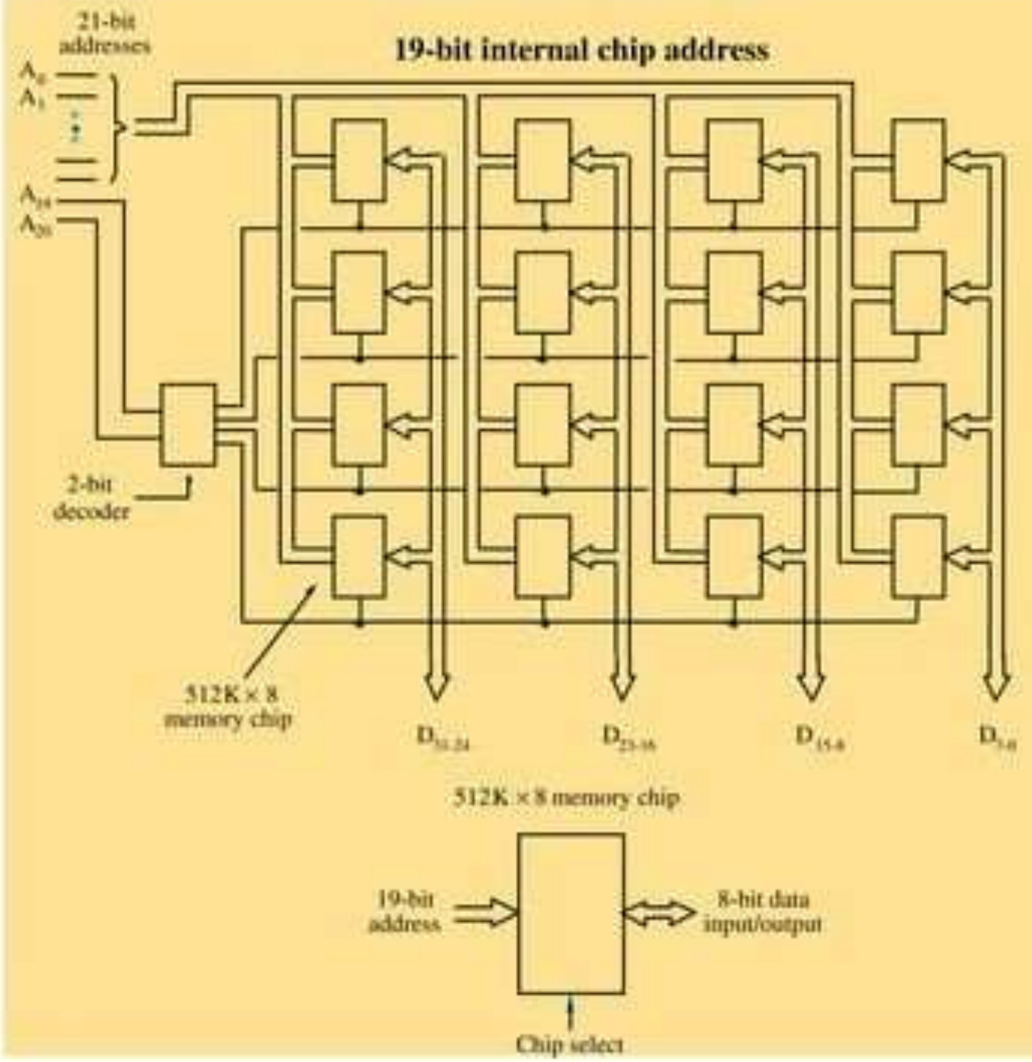


Figure 5.9 Burst read of length 4 in an SDRAM.

Synchronous DRAMs

- The timing diagram shows a timing diagram for a typical burst read of length 4
- First, the row address is latched under the control of RAS
- Memory takes 2 to 3 clock cycles to activate the selected row
- The column address is latched under the control of CAS signal
- After a delay of 1 clock cycle, the first set of data bits is placed in data lines
- The SDRAM automatically increments the column address to access the next three sets of bits in the selected row, which are placed on the data lines in the next 3 clock cycles
- SDRAMs have built-in refresh circuitry
- A part of the circuitry is a refresh counter, which provides the address of the rows that are selected for refreshing
- In a typical SDRAM, each row is refreshed at least every 64 ms
- Commercial SDRAMs can be used with clock speeds above 100 MHz

Static memories



- *Implement a memory unit of 2M words of 32 bits each.*
- *Use 512x8 static memory chips.*
- *Each column consists of 4 chips.*
- *Each chip implements one byte position.*
- *A chip is selected by setting its*
- *chip select control line to 1.*
- *Selected chip places its data on the data output line, outputs of other chips are in high impedance state.*
- *21 bits to address a 32-bit word.*
- *High order 2 bits are needed to select the row, by activating the four Chip Select signals.*
- *19 bits are used to access specific byte locations inside the selected chip.*

Dynamic memories

- Large dynamic memory systems can be implemented using DRAM chips in a similar way to static memory systems.
- Placing large memory systems directly on the motherboard will occupy a large amount of space.
  - Also, this arrangement is inflexible since the memory system cannot be expanded easily.
- Packaging considerations have led to the development of larger memory units known as SIMMs (Single In-line Memory Modules) and DIMMs (Dual In-line Memory Modules).
- Memory modules are an assembly of memory chips on a small board that plugs vertically onto a single socket on the motherboard.
  - Occupy less space on the motherboard.
  - Allows for easy expansion by replacement.

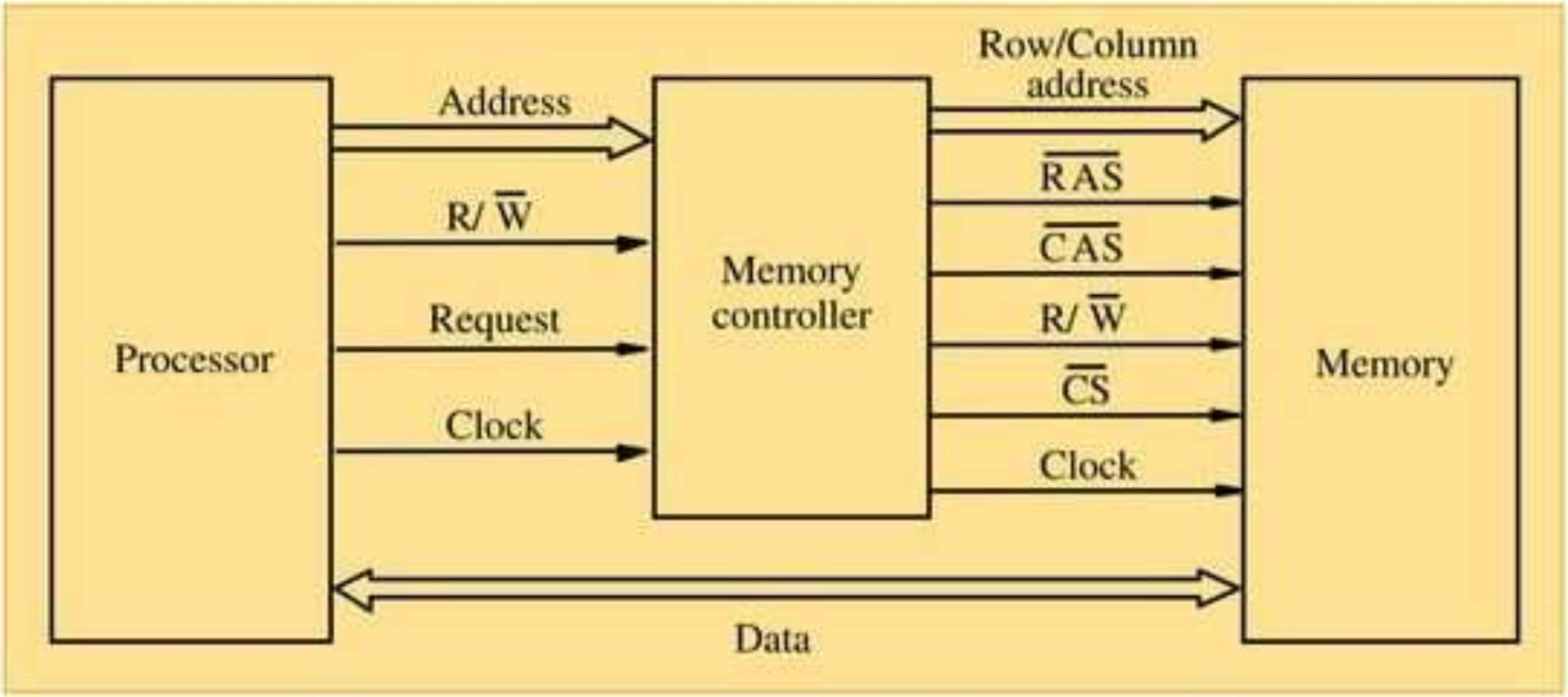
MEMORY SYSTEM CONSIDERATIONS

- The choice of RAM chip for the given application depends on several factors
- Foremost among these factors are the cost, speed, power dissipation, and size of the chip
- SRAM is only used when very fast operation is the primary requirement
- Their cost and size are adversely affected by the complexity of the circuit that realizes the basic cells
- They are used mainly in cache memory
- DRAM are the predominant choice for implementing computer main memories
- The high densities achievable in these chips make large memories economically feasible

Memory controller

- Recall that in a dynamic memory chip, to reduce the number of pins, multiplexed addresses are used.
- Address is divided into two parts:
  - High-order address bits select a row in the array.
  - They are provided first, and latched using RAS signal.
  - Low-order address bits select a column in the row.
  - They are provided later, and latched using CAS signal.
- However, a processor issues all address bits at the same time.
- In order to achieve the multiplexing, memory controller circuit is inserted between the processor and memory.

Memory controller (contd..)



Rambus Memory

- A very wide bus is expensive and requires a lot of space on the motherboard
- Alternative approach is to implement a narrow bus that is much faster
- This bus is called as the Rambus
- Key feature is the fast signaling method used to transfer information between chips
- It allows a clock frequency of 400 MHz
- These chips use cell array based on DRAM technologies
- Multiple banks of cell arrays used to access more than one word at a time
- Original design of Rambus is to provide 9 data lines and number of control and power supply lines
- It competes directly with DDR SDRAM technology

ROM

Read Only Memory

ROM
Read Only Memory

What is ROM?

Read-only memory (ROM) is a type of storage medium that permanently stores data on personal computers (PCs) and other electronic devices.

- It is type of **internal memory**.
- The data and instructions in ROM are stored by the manufacturer at the time of its manufacturing. This data and programs **cannot be changed or deleted** after wards.
- The data or instructions stored **in ROM can only be read but new data or instructions cannot be written into it.** This is the reason why it is called Read Only Memory.

Introduction

- **Read-only memory** (**ROM**) is a class of storage medium used in computers and other electronic devices. Data stored in ROM cannot be modified, or can be modified only slowly or with difficulty.
- Read Only Memories (ROM) or Programmable Read Only Memories (PROM) have:
  N input lines,
  M output lines, and
  2^N decoded minterms.
- Fixed AND array with 2^N outputs implementing all N-literal minterms.

- Programmable OR Array with M outputs lines to form up to M sum of minterm expressions.
- A program for a ROM or PROM is simply a multiple-output truth table
- If a 1 entry, a connection is made to the corresponding minterm for the corresponding output.
- If a 0, no connection is made

Can be viewed as a *memory* with the inputs as *addresses* of *data* (output values), hence ROM or PROM names!

Types of ROM

There are **five** basic ROM types:

ROM - Read Only Memory

PROM - Programmable Read Only Memory

EPROM - Erasable Programmable Read Only Memory

EEPROM - Electrically Erasable Programmable Read Only Memory

Flash EEPROM Memory

Why ROM is non-volatile?

ROM stores data and instructions **permanently.**

- When the power is turned off, the instructions stored in ROM are not lost. That is the reason ROM is called *non-volatile* memory.

Read-Only-Memory

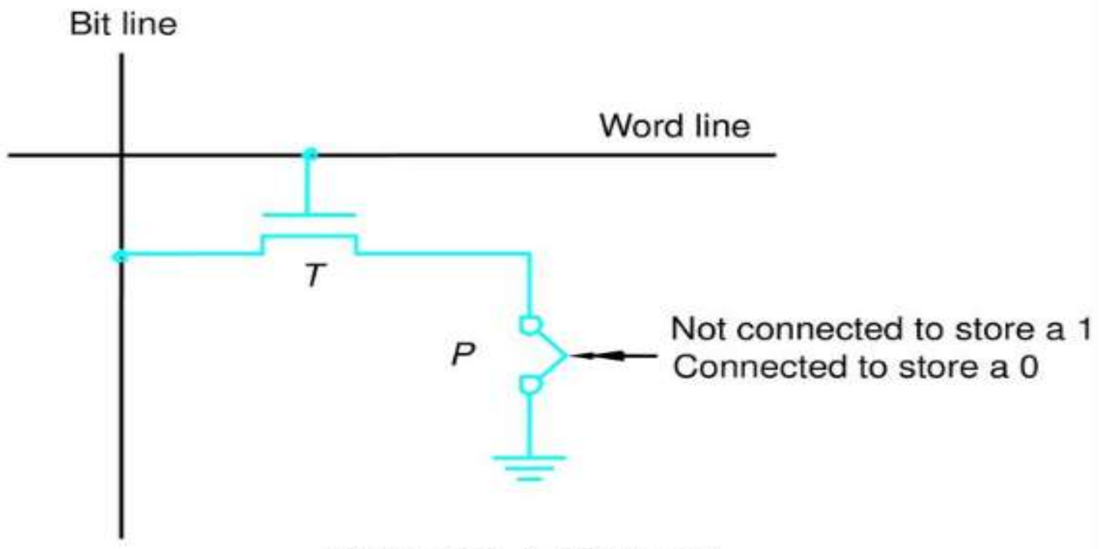


Figure 12. A ROM cell.

Read only memory (ROM)

- ✓ROM holds programs and data **permanently** even when computer is switched off
- ✓Data can be read by the CPU in any order so ROM is also **direct access**
- ✓The contents of ROM are fixed at the time of manufacture
- ✓Stores a program called the **bootstrap loader** that helps start up the computer
- ✓Access time of between 10 and 50 nanoseconds

ROMS VS. RAMS

There are some important differences between ROM and RAM.

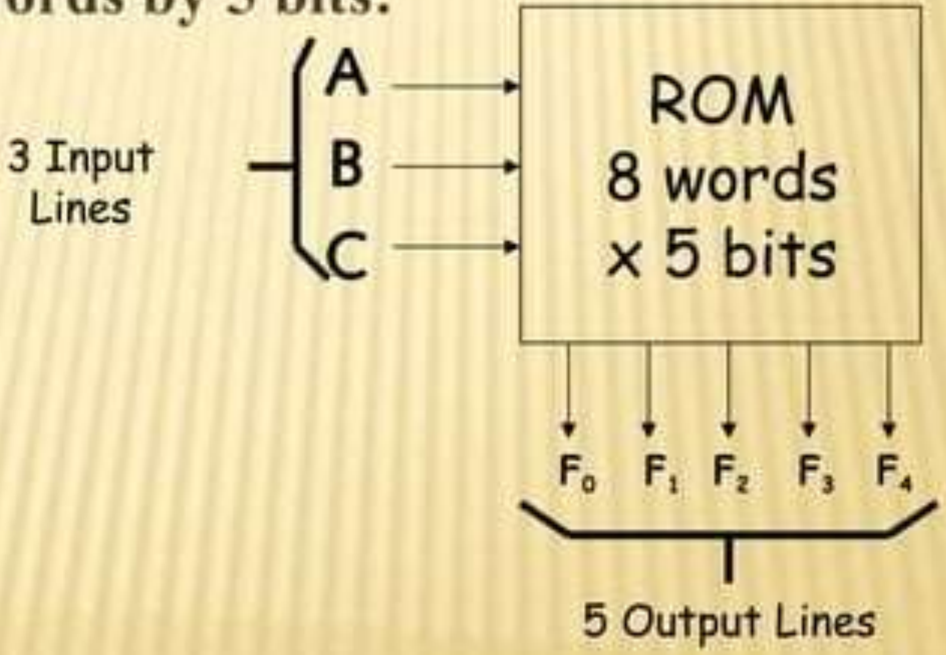
- ROMs are "non-volatile"—data is preserved even without power. On the other hand, RAM contents disappear once power is lost.
- ROMs require special (and slower) techniques for writing, so they're considered to be "read-only" devices.

Some newer types of ROMs do allow for easier writing, although the speeds still don't compare with regular RAMs.

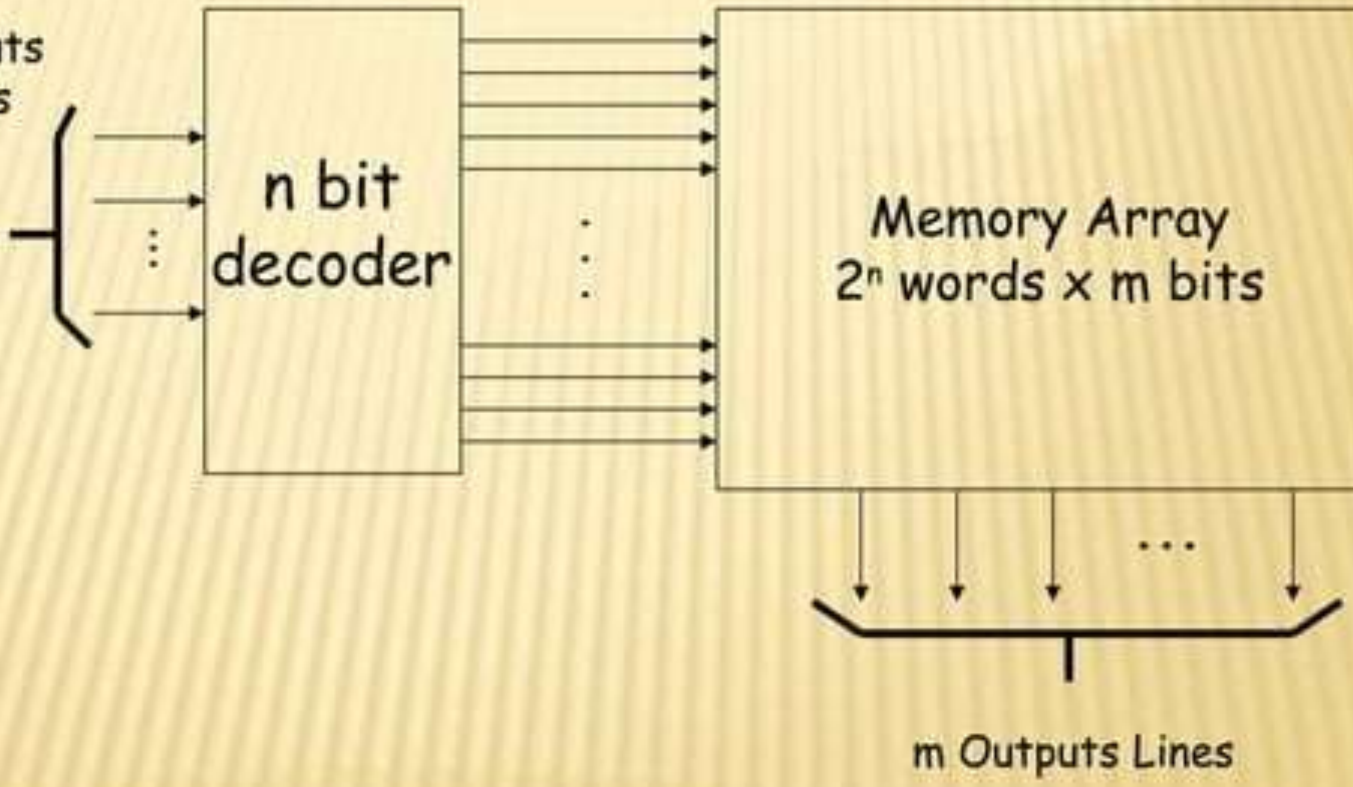
- MP3 players, digital cameras and other toys use CompactFlash, Secure Digital, or Memory Stick cards for non-volatile storage.
- Many devices allow you to upgrade programs stored in "flash ROM."

READ-ONLY MEMORY (ROM)

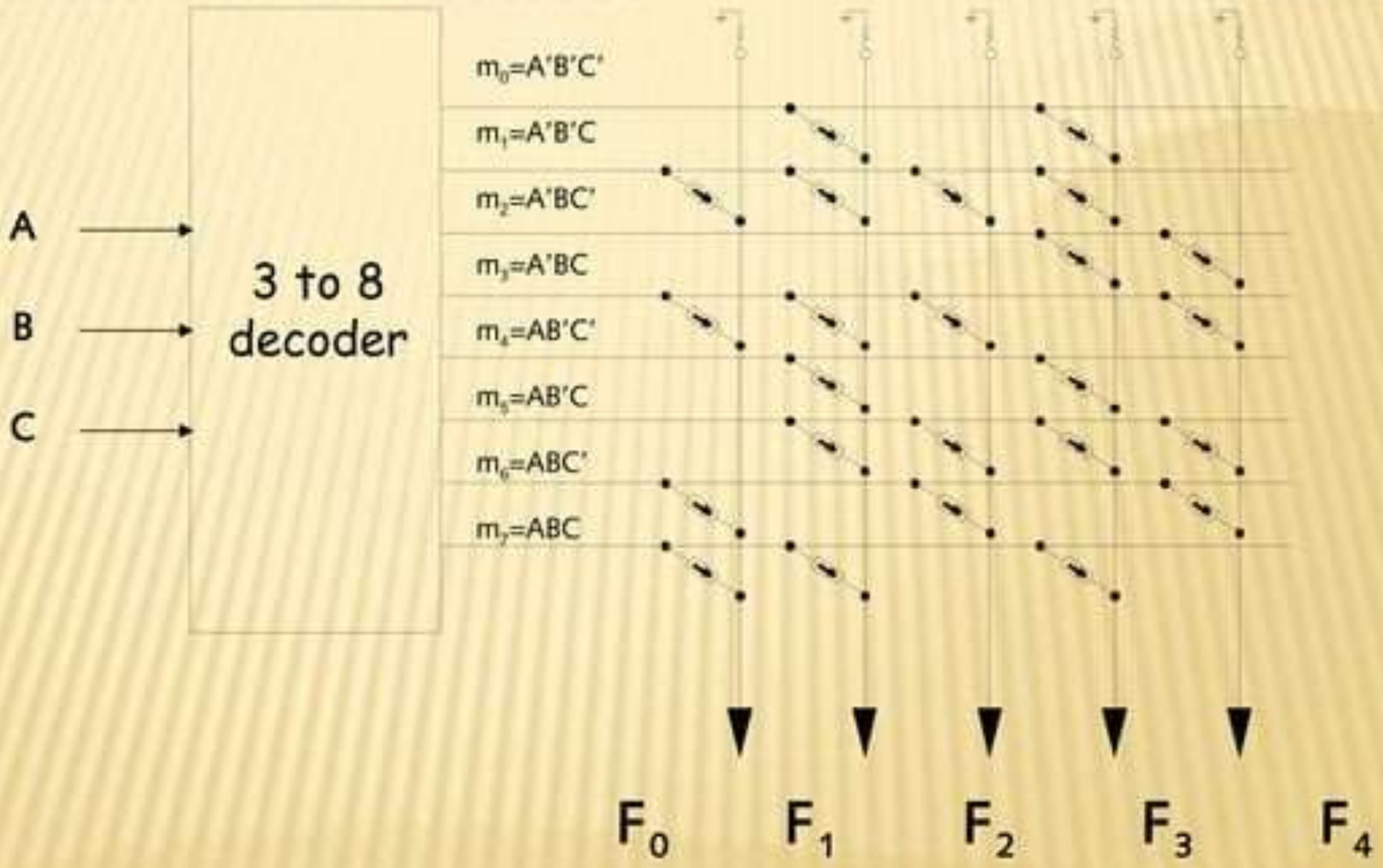
- **N input bits**
- **2^N words by M bits**
- **Implement M arbitrary functions of N variables**
  - **Example 8 words by 5 bits:**



ROM INTERNAL STRUCTURE



ROM MEMORY ARRAY



PROM

- PROM stands for Programmable Read Only Memory.
- This form of ROM is **initially blank**. The user or manufacturer can write data/program on it by using special devices.
- However, **once the program or data is written in PROM chip, it cannot be changed**.
- If there is an error in writing instructions or data in PROM, the error cannot be erased rather PROM chip becomes unusable.

EPROM

- EPROM stands for **Erasable Programmable Read Only Memory.**
- This form of ROM is **also initially blank**.
- The user or manufacturer **can write program or data** on it by using special devices.
- **Unlike PROM, the data written in EPROM chip can be erased by using special devices and ultraviolet rays.**
- **New data can also be added**.
- **When EPROM is in use, its contents can only be read.**

EEPROM

- EEPROM stands for **Electrically Erasable Programmable Read Only Memory.**
- **This kind of ROM can be written or changed with the help of electrical devices.**
- So data stored in this type of ROM chip can be easily modified.

Flash Memory

- **Flash memory** is an electronic non-volatile computer storage medium that can be **electrically erased and reprogrammed**. Introduced by Toshiba in 1984, **flash memory** was developed from EEPROM (electrically erasable programmable read-only memory).

The Memory System

Cache Memories

What is Cache Memory?

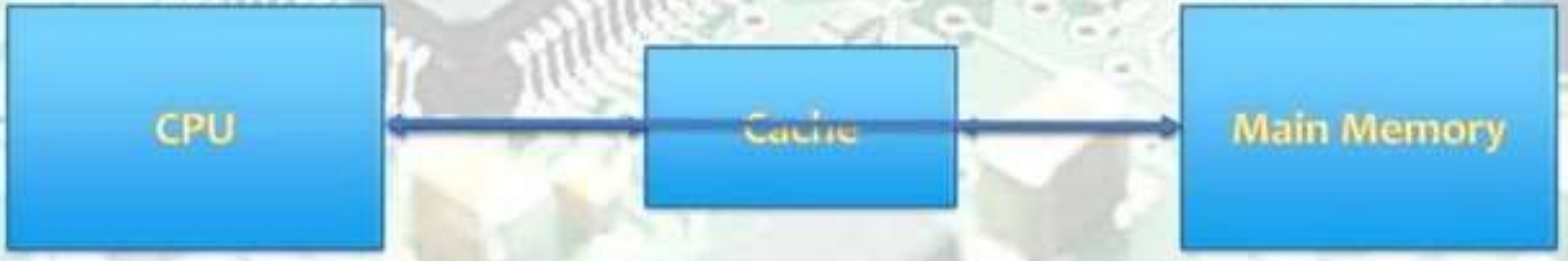
- Cache memory is a small, high-speed RAM buffer located between the CPU and main memory.

- Cache memory holds a copy of the instructions (instruction cache) or data (operand or data cache) currently being used by the CPU.

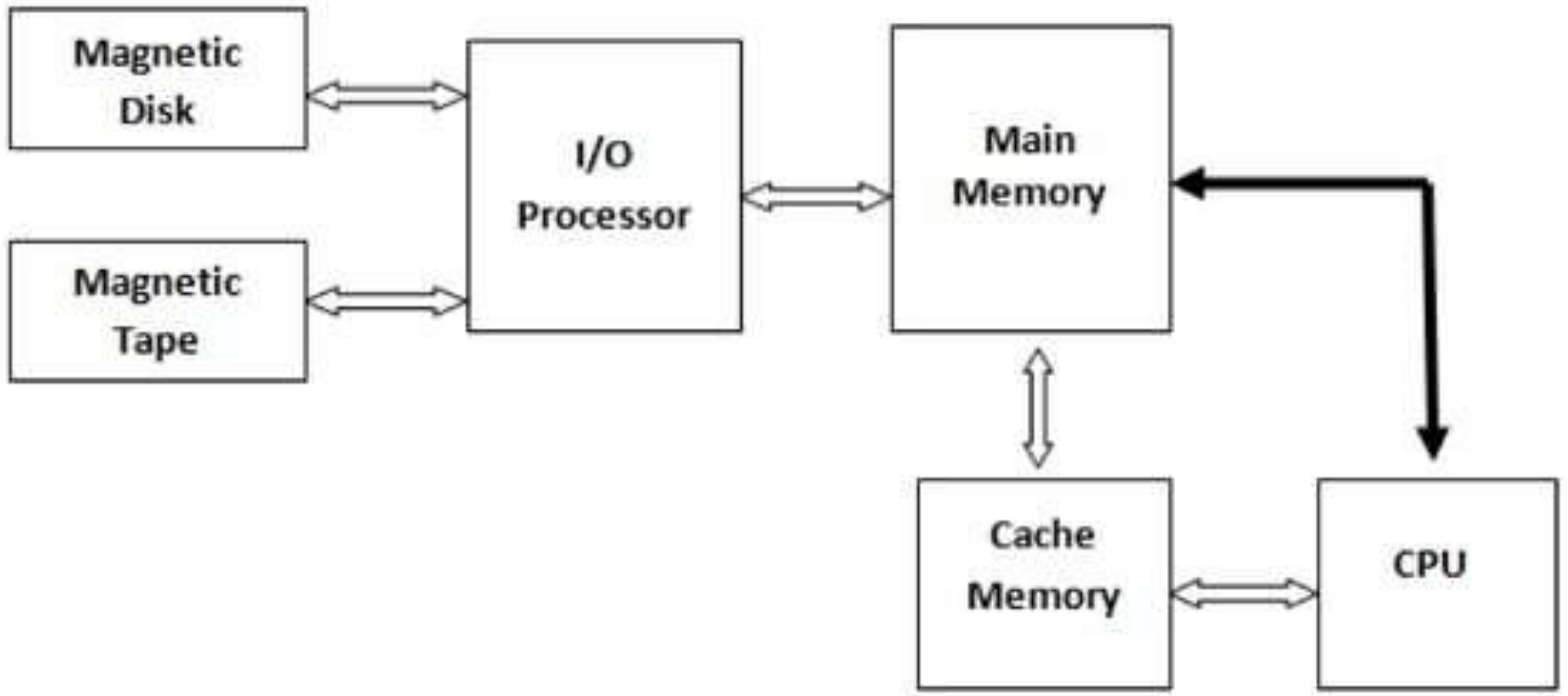
- The main purpose of a cache is to accelerate your computer while keeping the price of the computer low.

CACHE MEMORY

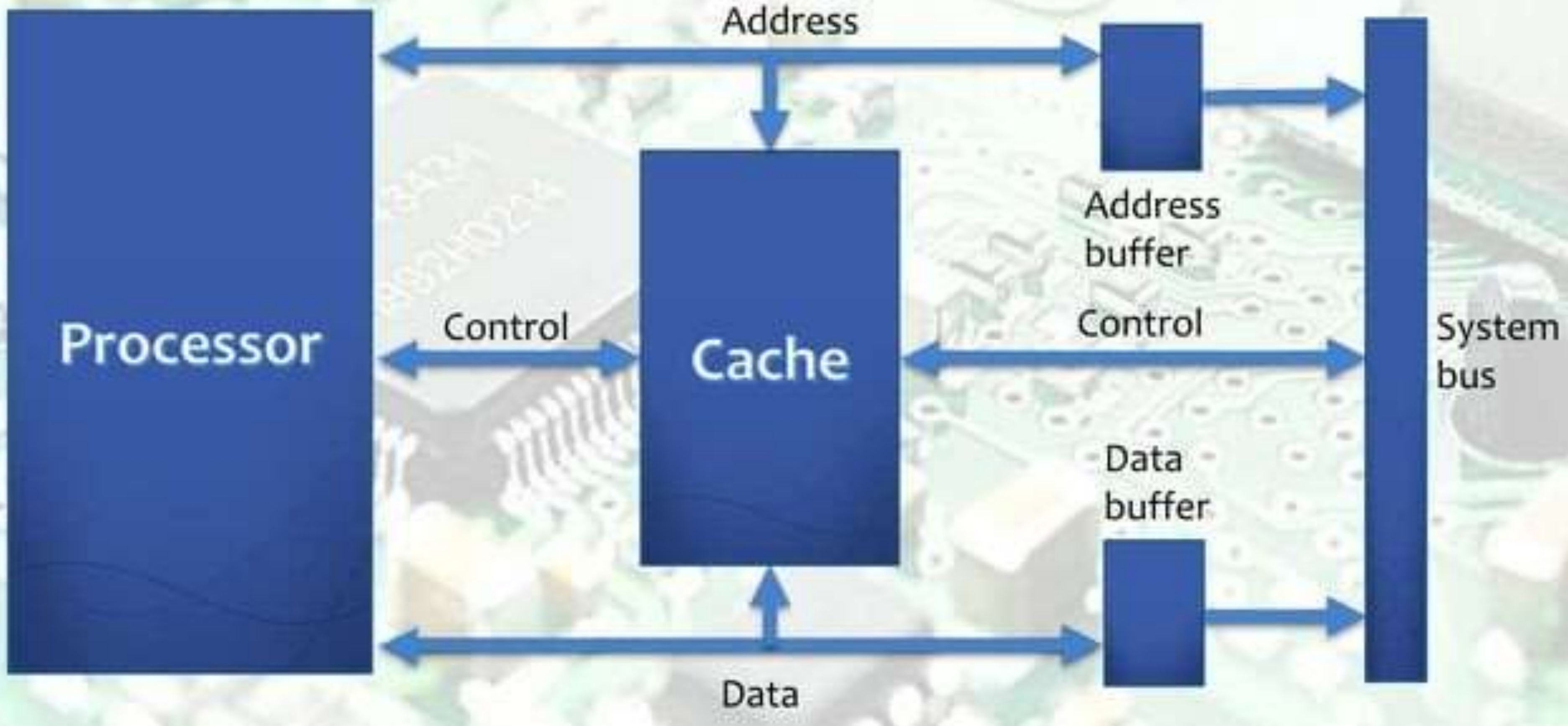
- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module



Placement of Cache in computer

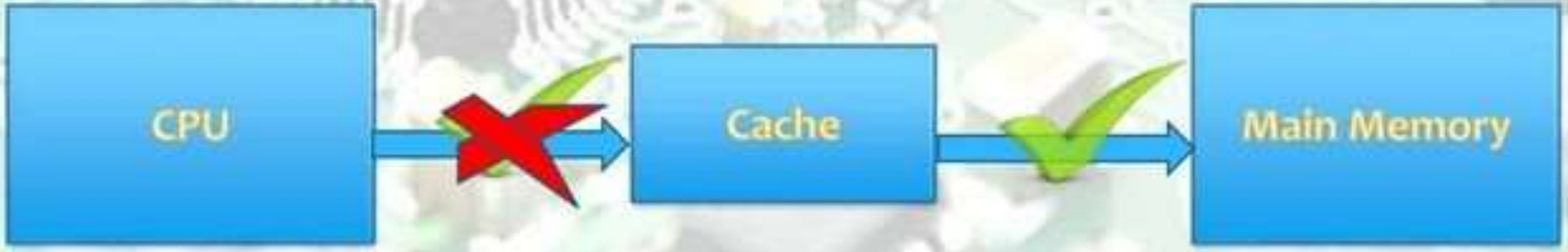


Cache Organization



Working Of Cache Memory

- The CPU initially looks in the Cache for the data it needs
- If the data is there, it will retrieve it and process it
- If the data is not there, then the CPU accesses the system memory and then puts a copy of the new data in the cache before processing it
- Next time if the CPU needs to access the same data again, it will just retrieve the data from the Cache instead of going through the whole loading process again



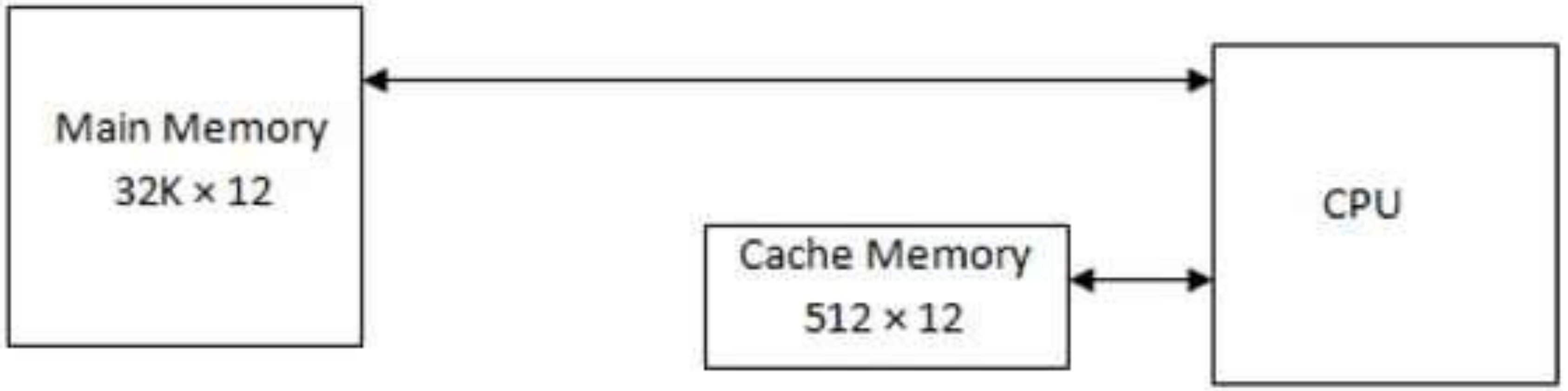
Hit Ratio

- The ratio of the total number of hits divided by the total CPU accesses to memory (i.e. hits plus misses) is called *Hit Ratio.*

- **Hit Ratio = Total Number of Hits / (Total Number of Hits + Total Number of Miss)**

Example

A system with 512 x 12 cache and 32 K x 12 of main memory.



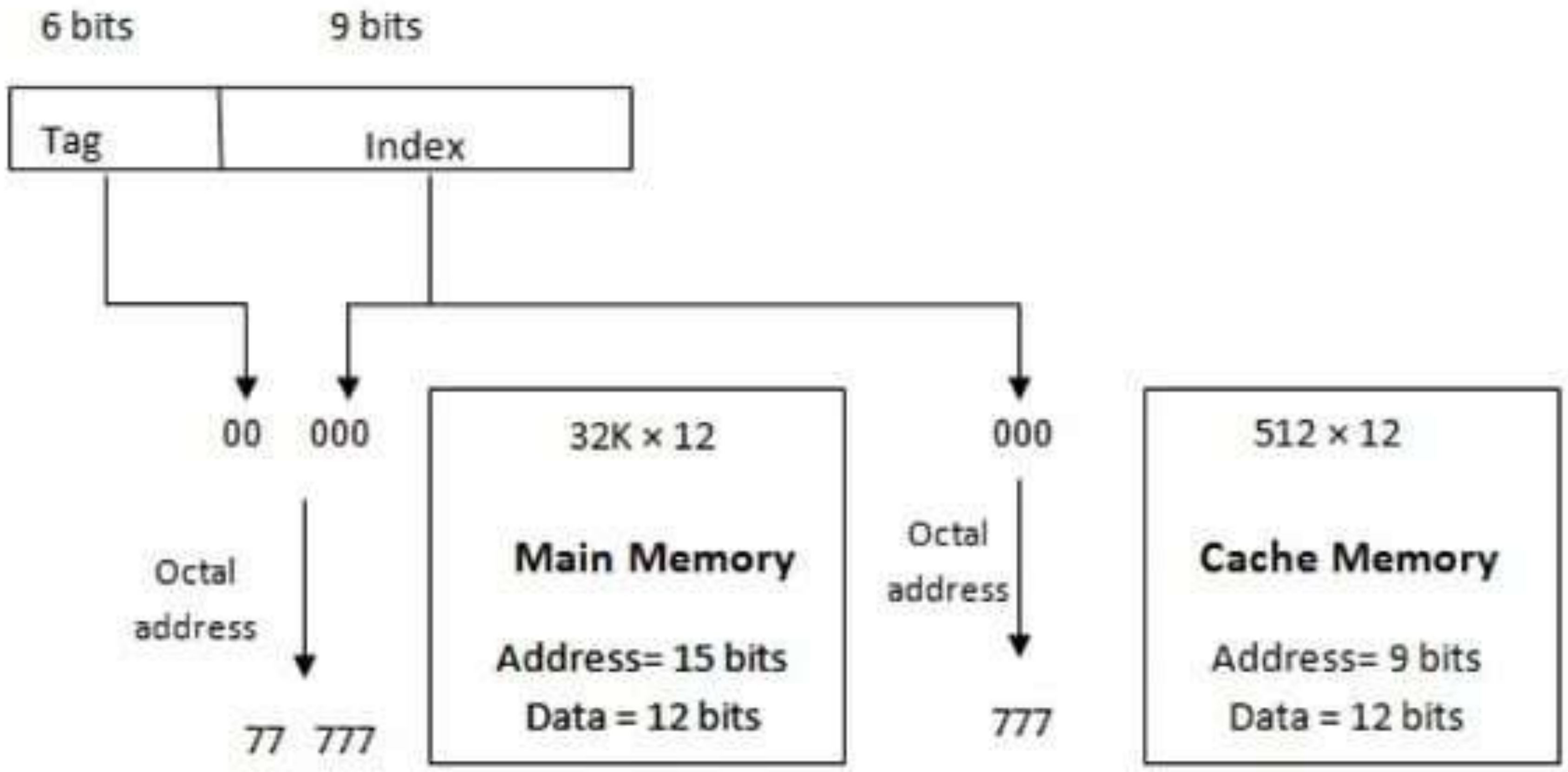
Types of Cache Mapping

1. Direct Mapping
2. Associative Mapping
3. Set Associative Mapping

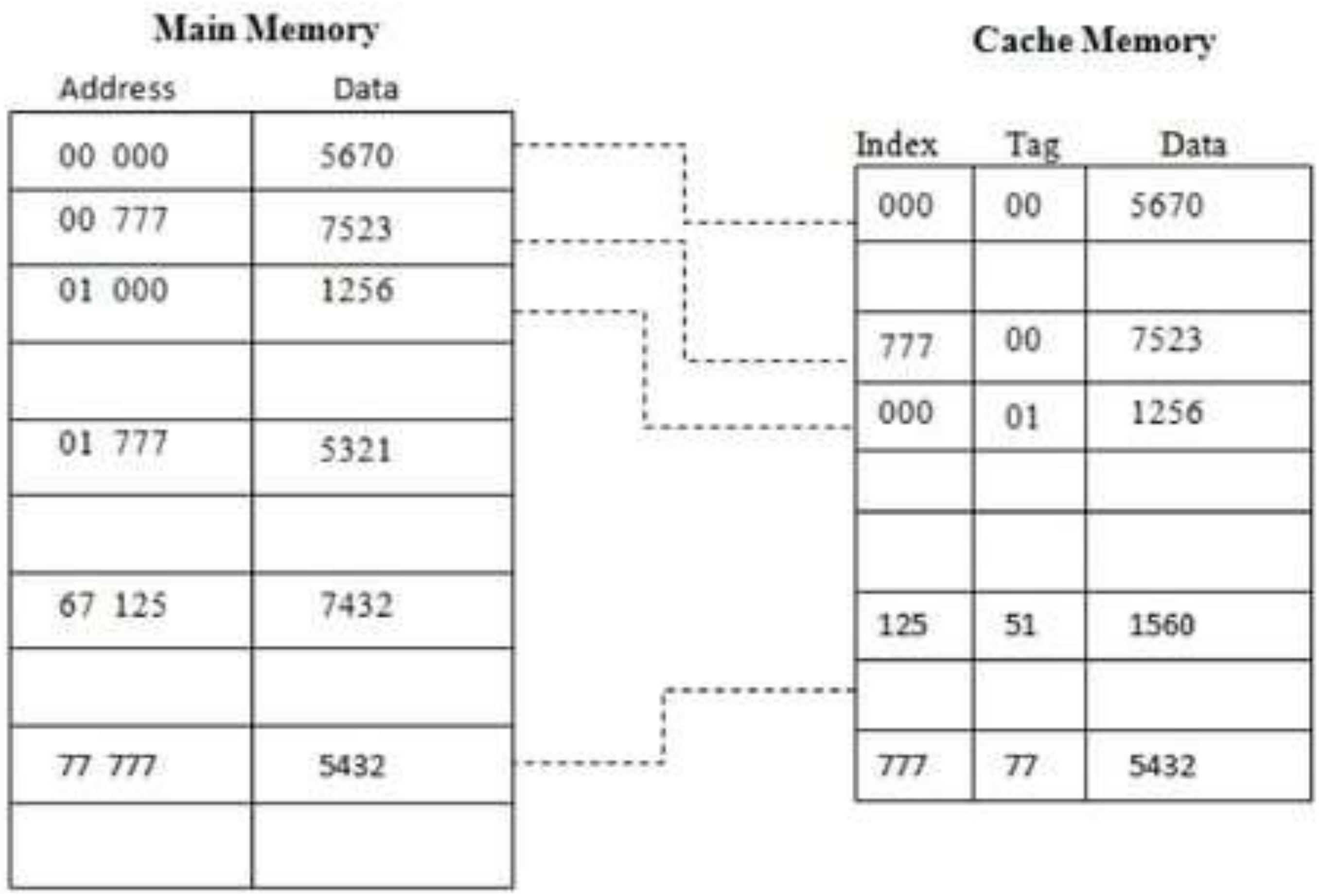
1. Direct Mapping

- The direct mapping technique is simple and inexpensive to implement.
- When the CPU wants to access data from memory, it places a address. The index field of CPU address is used to access address.
- The tag field of CPU address is compared with the associated tag in the word read from the cache.
- If the tag-bits of CPU address is matched with the tag-bits of cache, then there is a *hit* and the required data word is read from cache.
- If there is no match, then there is a *miss* and the required data word is stored in main memory. It is then transferred from main memory to cache memory with the new tag.

1. Direct Mapping



1. Direct Mapping



Main Memory

Address	Data
00 000	5670
00 777	7523
01 000	1256
01 777	5321
67 125	7432
77 777	5432

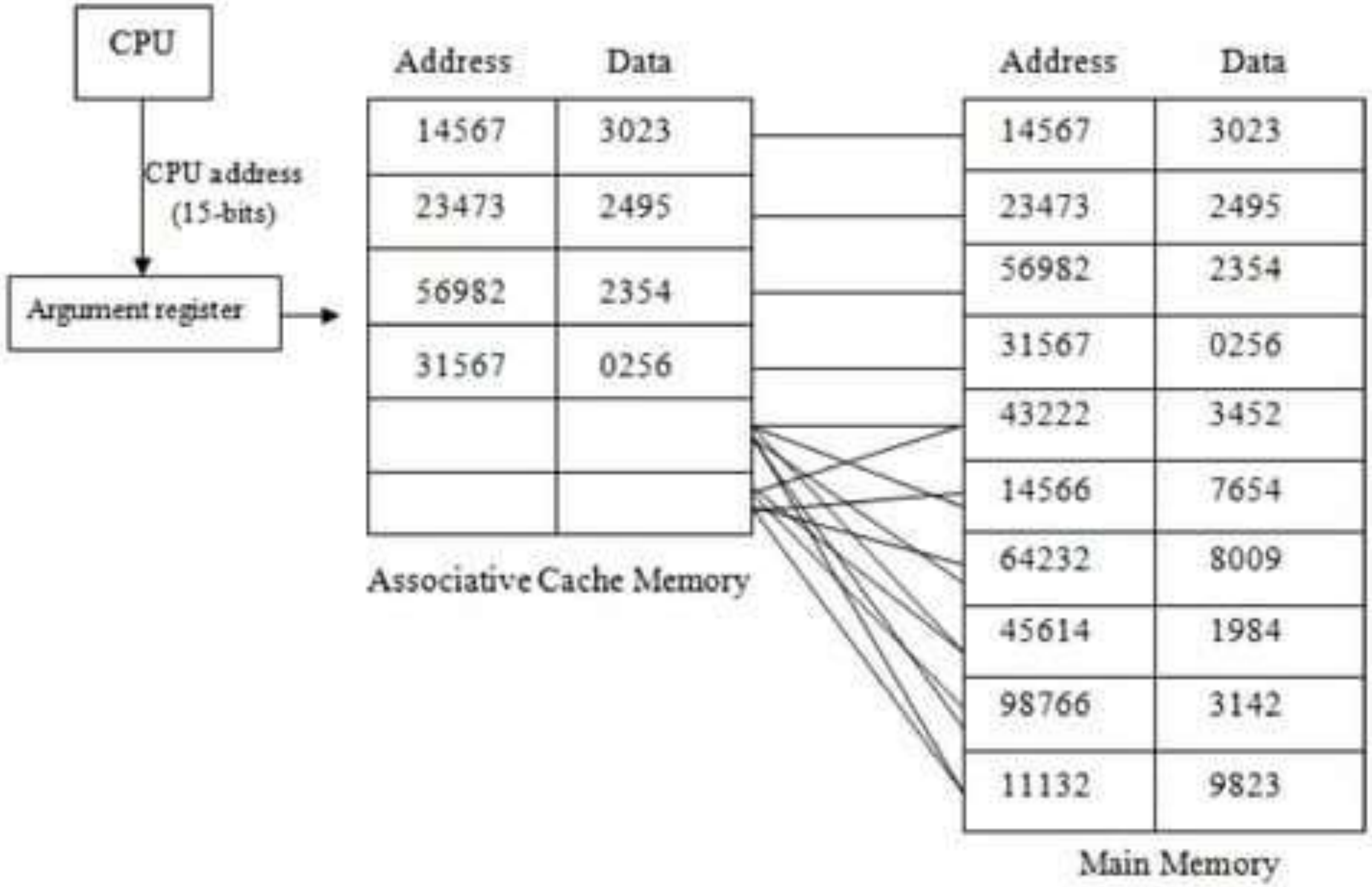
Cache Memory

Index	Tag	Data
000	00	5670
777	00	7523
000	01	1256
125	51	1560
777	77	5432

2. Associative Mapping

- An associative mapping uses an associative memory.
- This memory is being accessed using its contents.
- Each line of cache memory will accommodate the address (main memory) and the contents of that address from the main memory.
- That is why this memory is also called Content Addressable Memory (CAM). It allows each block of main memory to be stored in the cache.

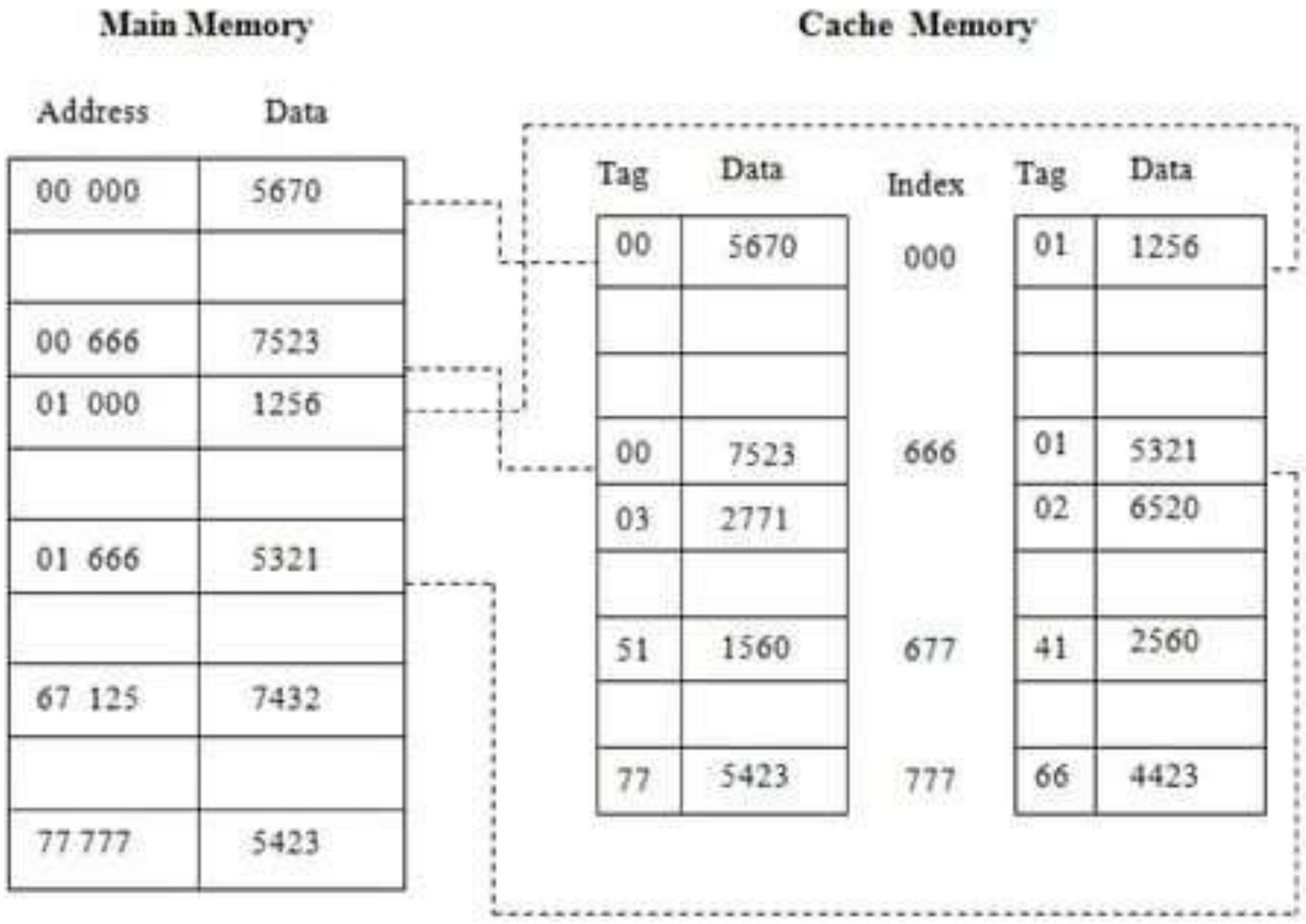
2. Associative Mapping



3. Set Associative Mapping

- That is the easy control of the direct mapping cache and the more flexible mapping of the fully associative cache.
- In set associative mapping, each cache location can have more than one pair of tag + data items.
- That is more than one pair of tag and data are residing at the same location of cache memory. If one cache location is holding two pair of tag + data items, that is called *2-way set associative mapping*.

3. Two-Way Set Associative Mapping



Replacement Algorithms of Cache Memory

- Replacement algorithms are used when there are no available space in a cache in which to place a data. Four of the most common cache replacement algorithms are described below:
- ***Least Recently Used (LRU):***
  - The LRU algorithm selects for replacement the item that has been least recently used by the CPU.
- ***First-In-First-Out (FIFO):***
  - The FIFO algorithm selects for replacement the item that has been in the cache from the longest time.
- ***Least Frequently Used (LRU):***
  - The LRU algorithm selects for replacement the item that has been least frequently used by the CPU.
- ***Random:***
  - The random algorithm selects for replacement the item randomly.

Writing into Cache

- When memory write operations are performed, CPU first writes into the cache memory. These modifications made by CPU during a write operations, on the data saved in cache, need to be written back to main memory or to auxiliary memory.

- These two popular cache write policies (schemes) are:
  - *Write-Through*
  - *Write-Back*

Write-Through

- In a write through cache, the main memory is updated each time the CPU writes into cache.

- The advantage of the write-through cache is that the main memory always contains the same data as the cache contains.

- This characteristic is desirable in a system which uses direct memory access scheme of data transfer. The I/O devices communicating through DMA receive the most recent data.

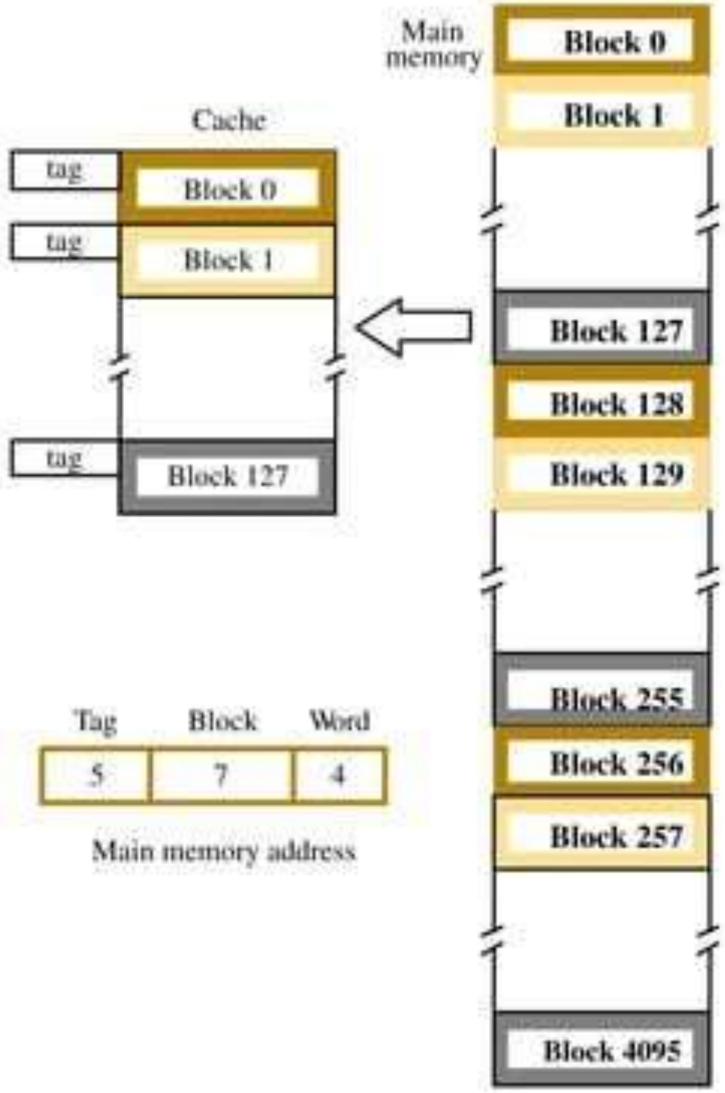
Write-Back

- In a write back scheme, only the cache memory is updated during a write operation.

- The updated locations in the cache memory are marked by a flag so that later on, when the word is removed from the cache, it is copied into the main memory.

- The words are removed from the cache time to time to make room for a new block of words.

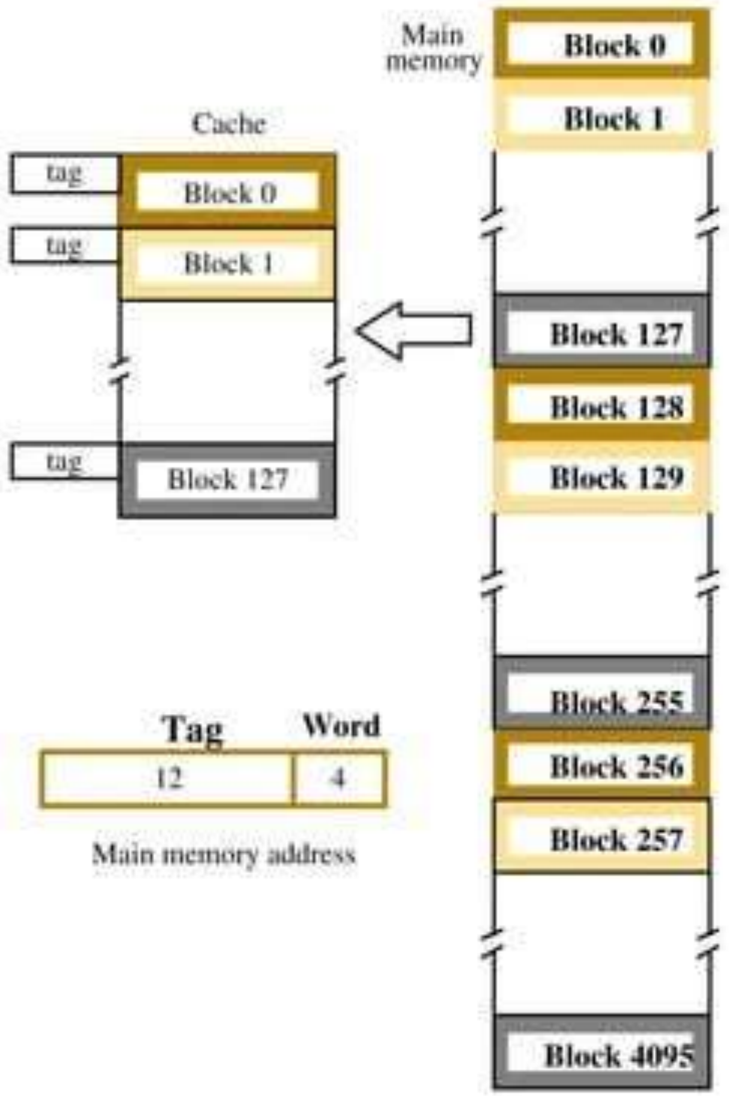
Direct mapping



•Block j of the main memory maps to j modulo 128 of the cache. 0 maps to 0, 129 maps to 1.
•More than one memory block is mapped onto the same position in the cache.
•May lead to contention for cache blocks even if the cache is not full.
•Resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.
•Memory address is divided into three fields:
- *Low order 4 bits determine one of the 16 words in a block.*
- *When a new block is brought into the cache, the the next 7 bits determine which cache block this new block is placed in.*
- *High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.*

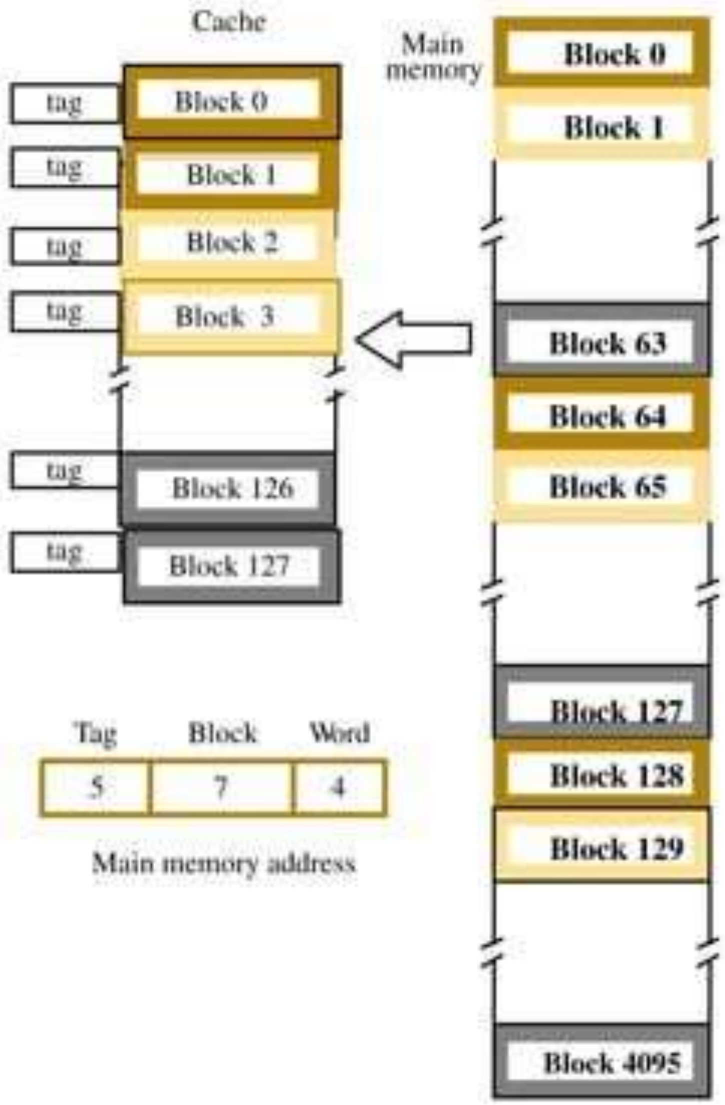
•Simple to implement but not very flexible.

Associative mapping



- *Main memory block can be placed into any cache position.*
- *Memory address is divided into two fields:*
  - *Low order 4 bits identify the word within a block.*
  - *High order 12 bits or tag bits identify a memory block when it is resident in the cache.*
- *Flexible, and uses cache space efficiently.*
- *Replacement algorithms can be used to replace an existing block in the cache when the cache is full.*
- *Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.*

Set-Associative mapping

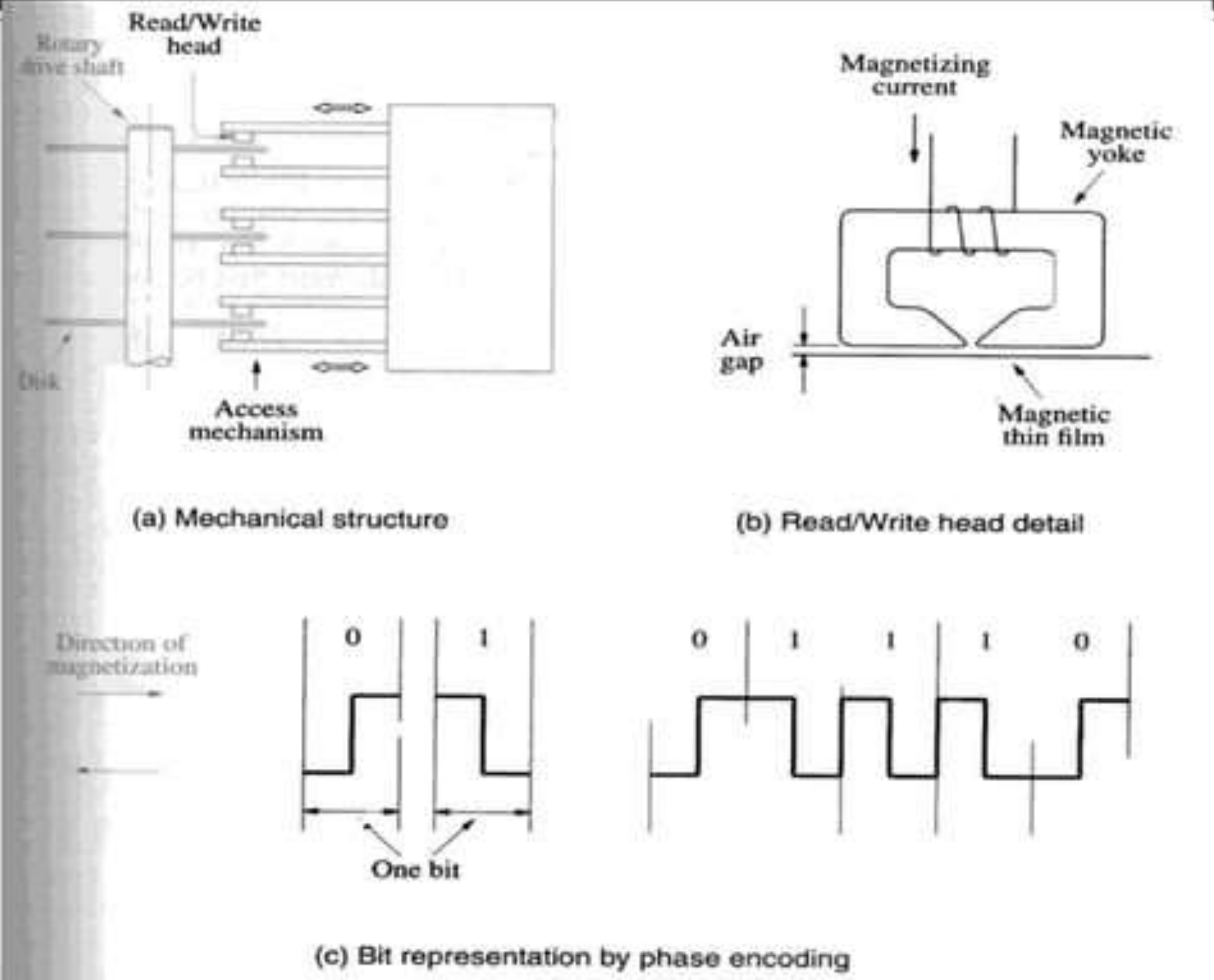


- *Blocks of cache are grouped into sets.*
- *Mapping function allows a block of the main memory to reside in any block of a specific set.*
- *Divide the cache into 64 sets, with two blocks per set.*
- *Memory block 0, 64, 128 etc. map to block 0, and they can occupy either of the two positions.*
- *Memory address is divided into three fields:*
  - *6 bit field determines the set number.*
  - *High order 6 bit fields are compared to the tag fields of the two blocks in a set.*
- *Set-associative mapping combination of direct and associative mapping.*
- *Number of blocks per set is a design parameter.*
  - *One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).*
  - *Other extreme is to have one block per set, is the same as direct mapping.*

The Memory System

Secondary Storage

Magnetic Hard Disks



Disk

Disk drive

Disk controller

Organization of Data on a Disk

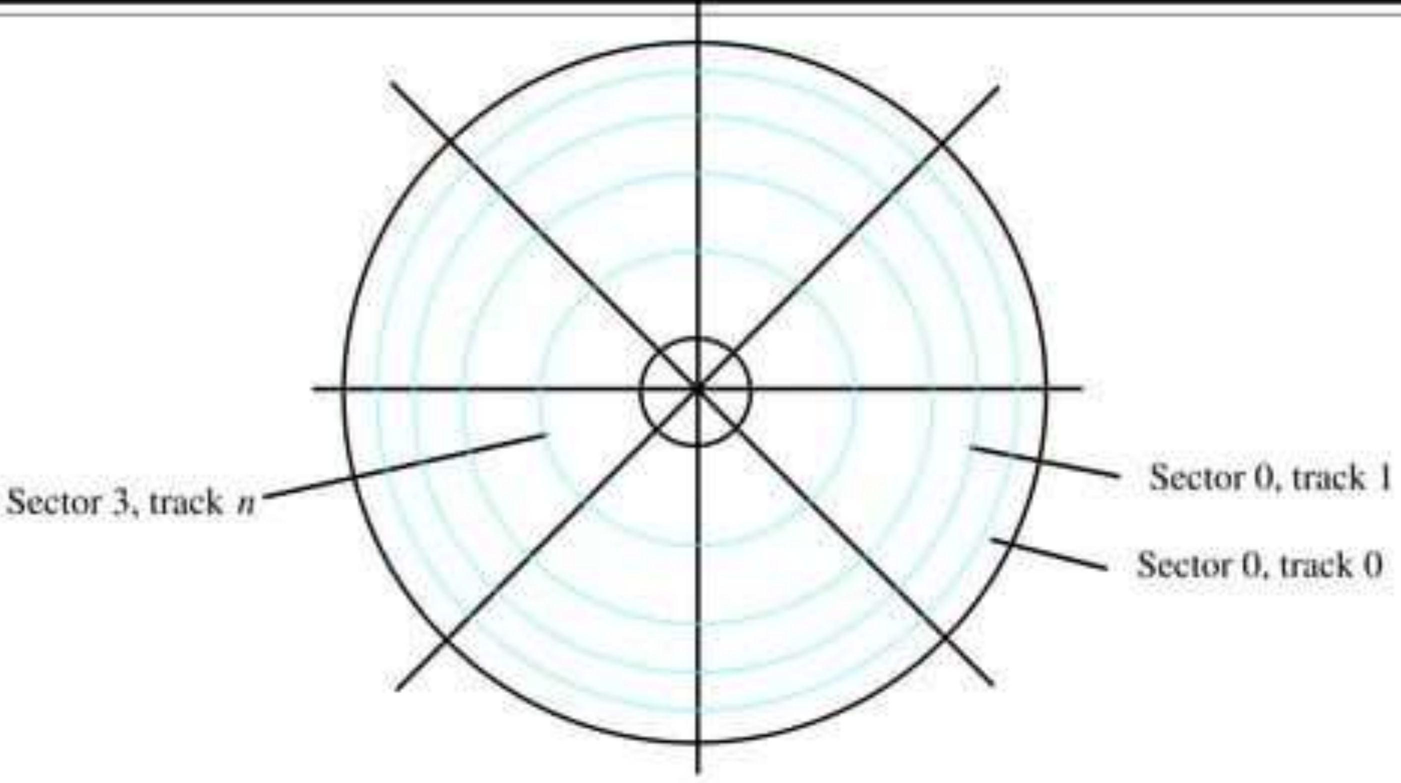


Figure 5.30. Organization of one surface of a disk.

Access Data on a Disk

- Sector header
- Following the data, there is an error-correction code (ECC).
- Formatting process
- Difference between inner tracks and outer tracks
- Access time – seek time / rotational delay (latency time)
- Data buffer/cache

Disk Controller

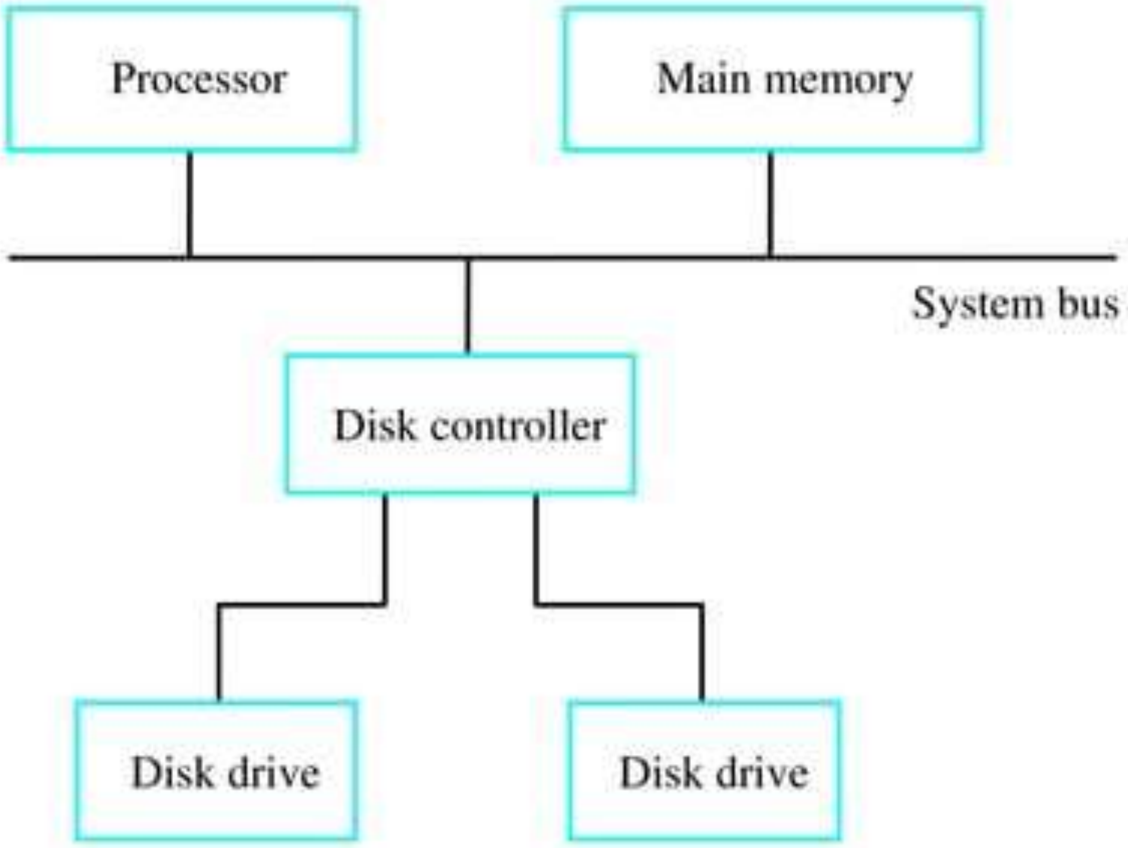


Figure 5.31. Disks connected to the system bus.

Disk Controller

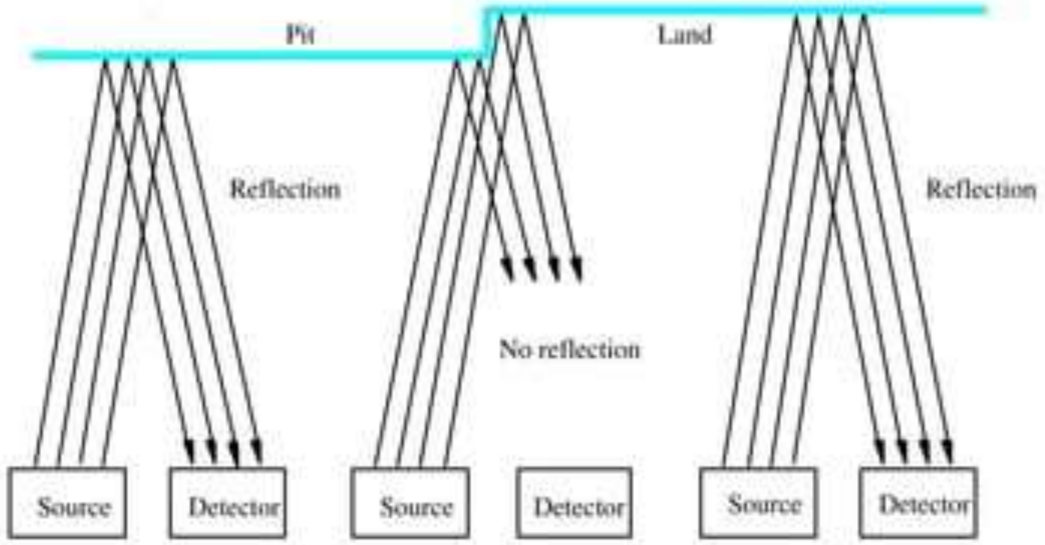
- Seek
- Read
- Write
- Error checking

RAID Disk Arrays

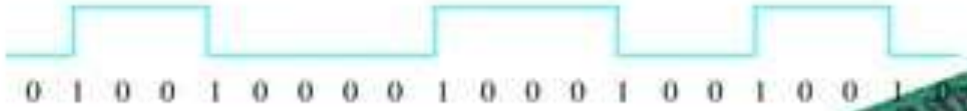
- Redundant Array of Inexpensive Disks
- Using multiple disks makes it cheaper for huge storage, and also possible to improve the reliability of the overall system.
- RAID0 – data striping
- RAID1 – identical copies of data on two disks
- RAID2, 3, 4 – increased reliability
- RAID5 – parity-based error-recovery

Optical Disks

(a) Cross-section

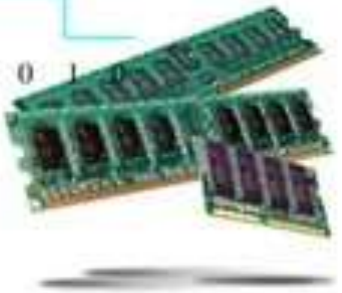


(b) Transition from pit to land



(c) Stored binary pattern

Figure 5.32. Optical disk.



Optical Disks

- CD-ROM
- CD-Recordable (CD-R)
- CD-ReWritable (CD-RW)
- DVD
- DVD-RAM

Magnetic Tape Systems

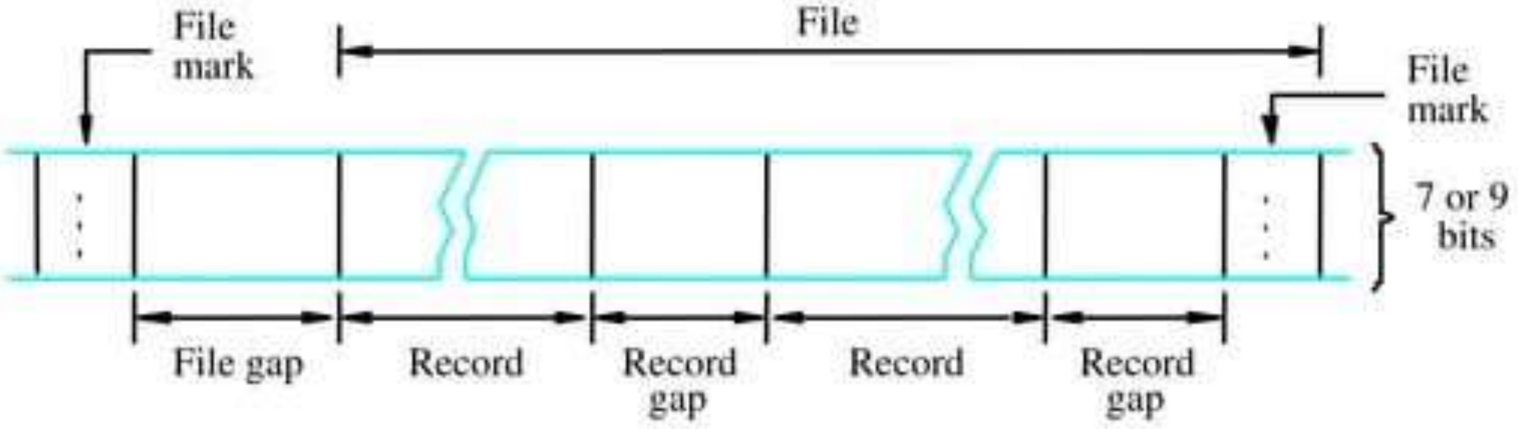


Figure 5.33. Organization of data on magnetic tape.

END

UNIT- V

PROCESSING UNIT

Overview

- Instruction Set Processor (ISP)
- Central Processing Unit (CPU)
- A typical computing task consists of a series of steps specified by a sequence of machine instructions that constitute a program.
- An instruction is executed by carrying out a sequence of more rudimentary operations.

Fundamental Concepts

Fundamental Concepts

- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)

Executing an Instruction

- Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

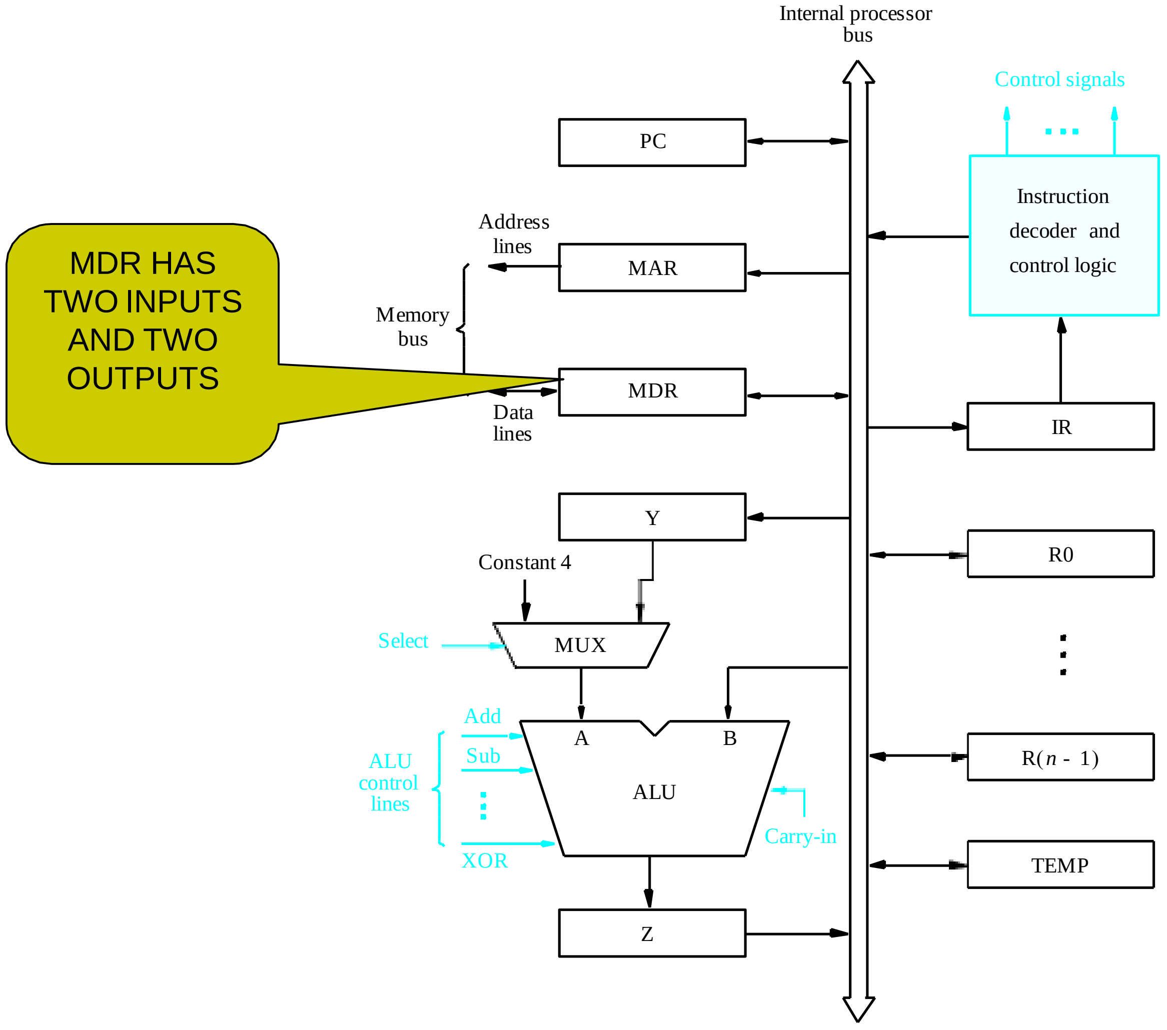
$$IR \leftarrow [[PC]]$$

- Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

$$PC \leftarrow [PC] + 4$$

- Carry out the actions specified by the instruction in the IR (execution phase).

Processor Organization



MDR HAS TWO INPUTS AND TWO OUTPUTS

Datapath

Figure 7.1. Single-bus organization of the datapath inside a processor.

Executing an Instruction

- Transfer a word of data from one processor register to another or to the ALU.
- Perform an arithmetic or a logic operation and store the result in a processor register.
- Fetch the contents of a given memory location and load them into a processor register.
- Store a word of data from a processor register into a given memory location.

Register Transfers

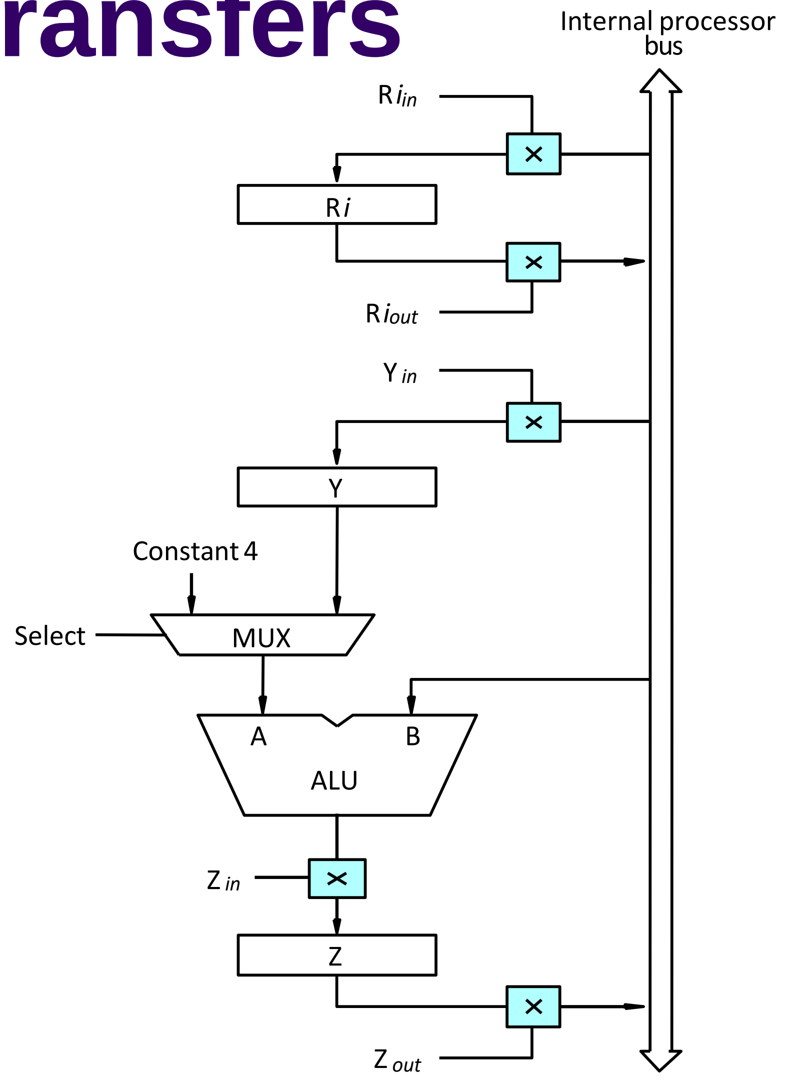


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Register Transfers

- All operations and data transfers are controlled by the processor clock.

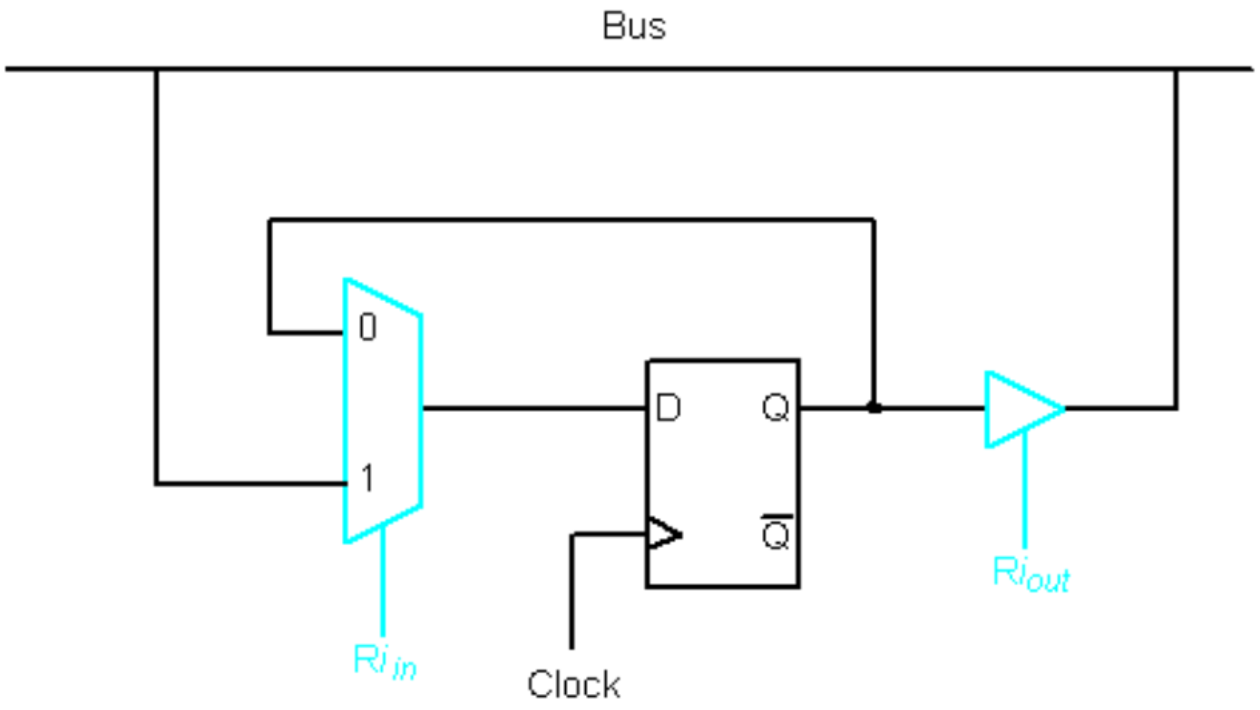


Figure 7.3. Input and output gating for one register bit.

Performing an Arithmetic or Logic Operation

- The ALU is a combinational circuit that has no internal storage.
- ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?
    1. R1out, Yin
    2. R2out, SelectY, Add, Zin
    3. Zout, R3in

Fetching a Word from Memory

- Address into MAR; issue Read operation; data into MDR.

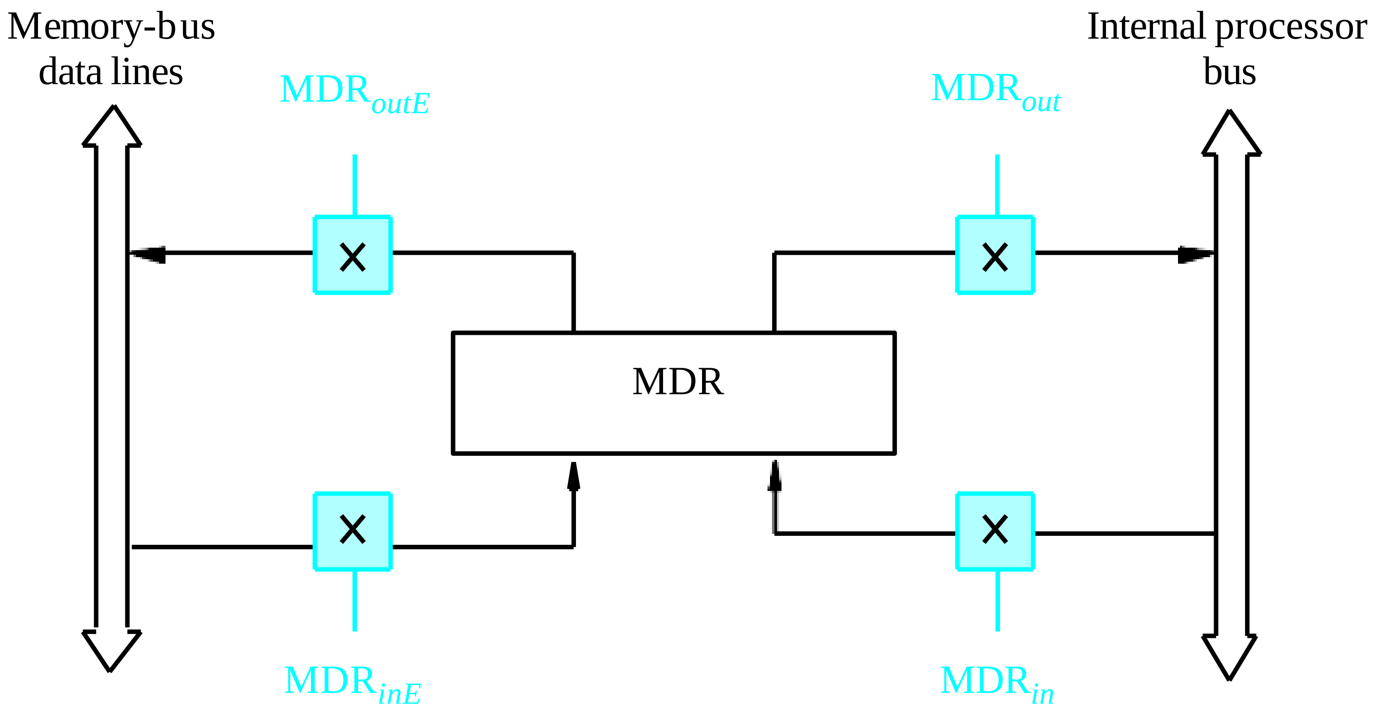


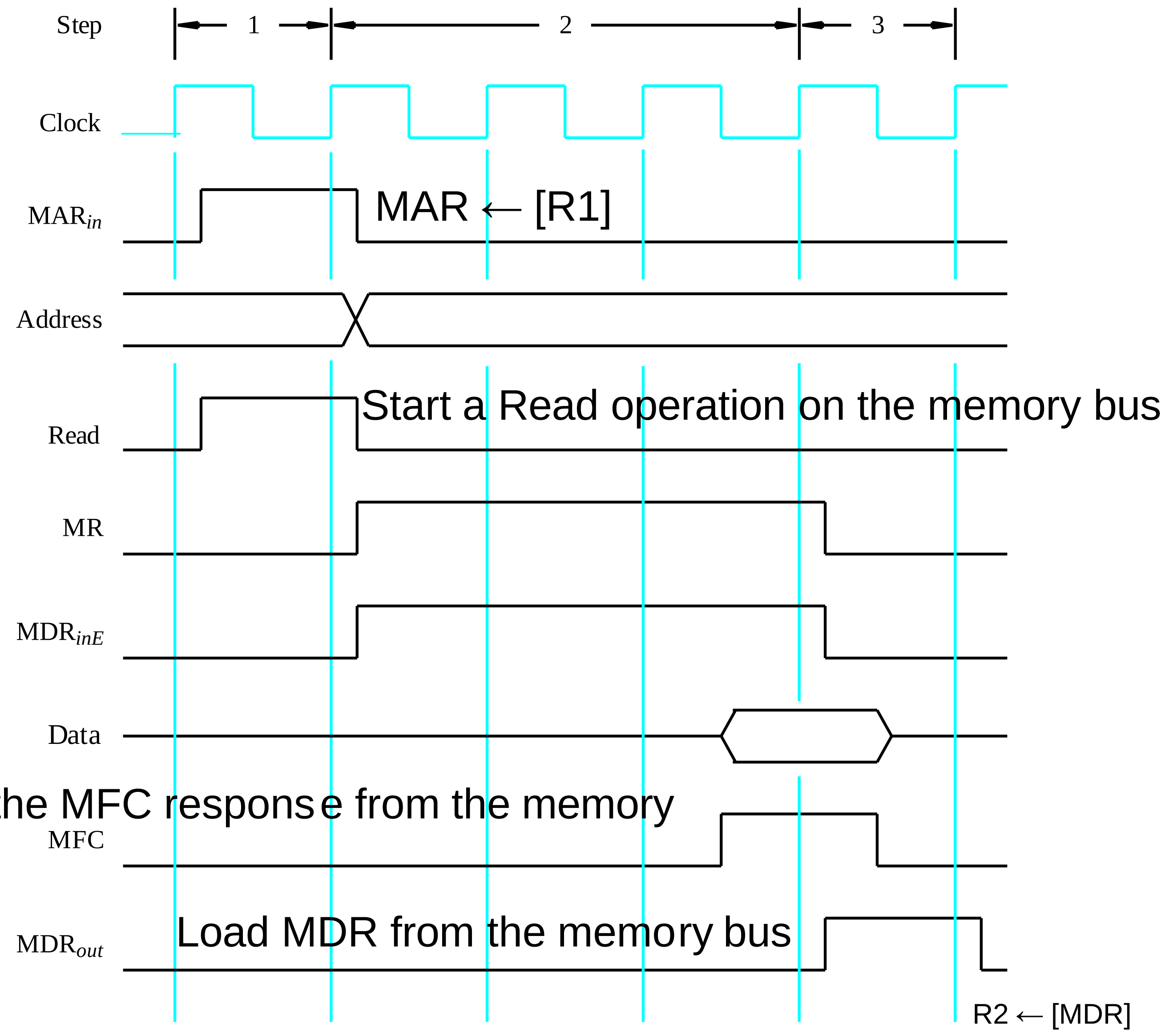
Figure 7.4. Connection and control signals for register MDR.

Fetching a Word from Memory

- The response time of each memory access varies (cache miss, memory-mapped I/O,…).
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).
- Move (R1), R2
  - MAR ← [R1]
  - Start a Read operation on the memory bus
  - Wait for the MFC response from the memory
  - Load MDR from the memory bus
  - R2 ← [MDR]

Timing

Assume MAR
is always available
on the address lines
of the memory bus.



Step
1
2
3
Clock
MAR_{in}
MAR ← [R1]
Address
Read
Start a Read operation on the memory bus
MR
MDR_{inE}
Data
Wait for the MFC response from the memory
MFC
MDR_{out}
Load MDR from the memory bus
R2 ← [MDR]

Figure 7.5. Timing of a memory Read operation.

Execution of a Complete Instruction

- Add (R3), R1
- Fetch the instruction
- Fetch the first operand (the contents of the memory location pointed to by R3)
- Perform the addition
- Load the result into R1

Architecture

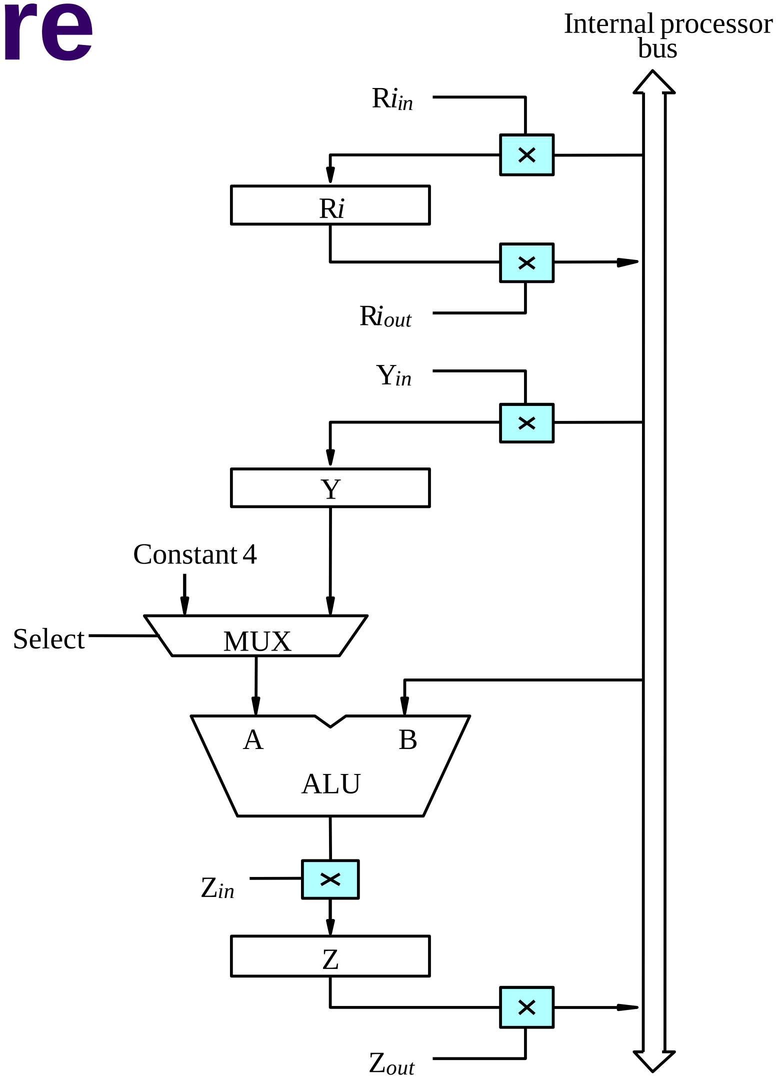


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Execution of a Complete Instruction

Add (R3), R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Figure 7.6. Control sequencefor execution of the instruction Add (R3),R1.

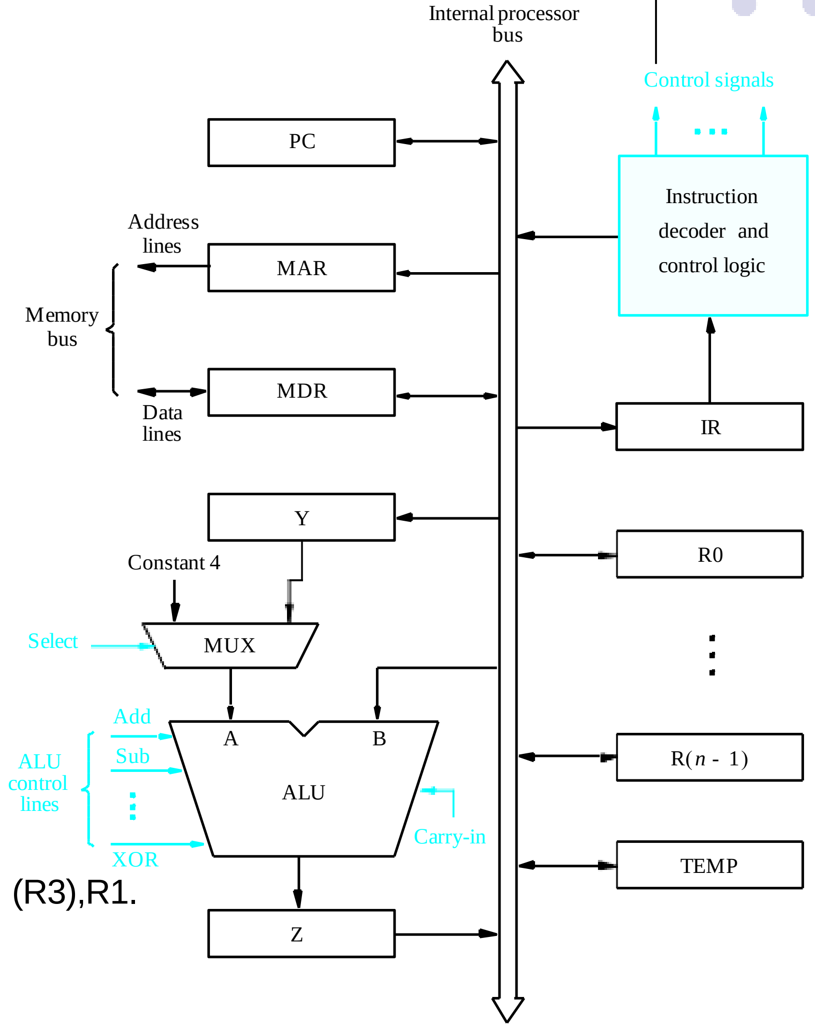


Figure 7.1. Single-bus organization of the datapath inside a processor.

Execution of Branch Instructions

- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.
- Conditional branch

Execution of Branch Instructions

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMF C
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 7.7. Control sequence for an unconditional branch instruction.

Multiple-Bus Organization

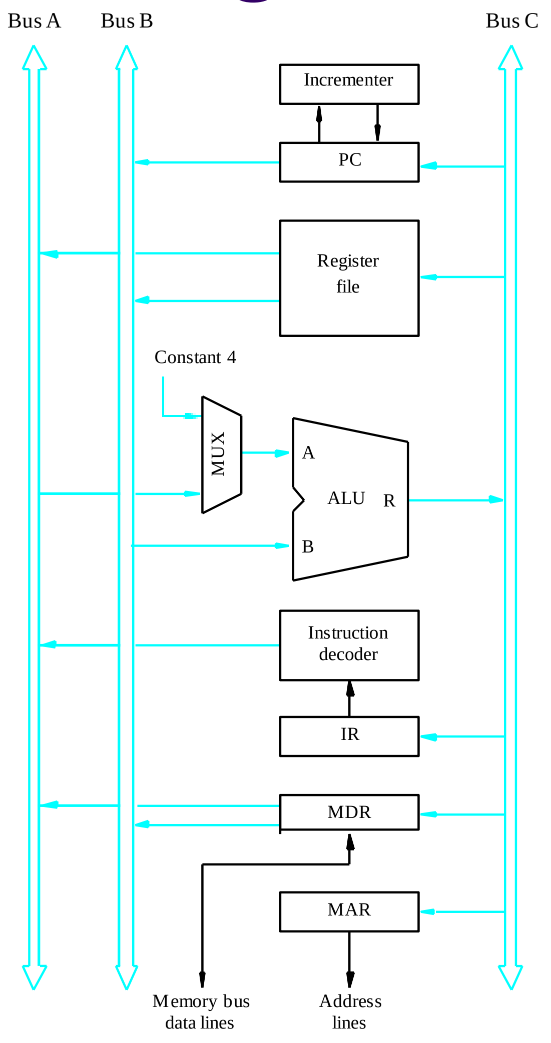


Figure 7.8. Three-bus organization of the datapath.

Multiple-Bus Organization

- Add R4, R5, R6

Step	**Action**
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMF C
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Figure 7.9. Control sequence for the instruction. Add R4,R5,R6, for the three-bus organization in Figure 7.8.

Hardwired Control

Overview

- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
- Two categories: hardwired control and microprogrammed control
- Hardwired system can operate at high speed; but with little flexibility.

Control Unit Organization

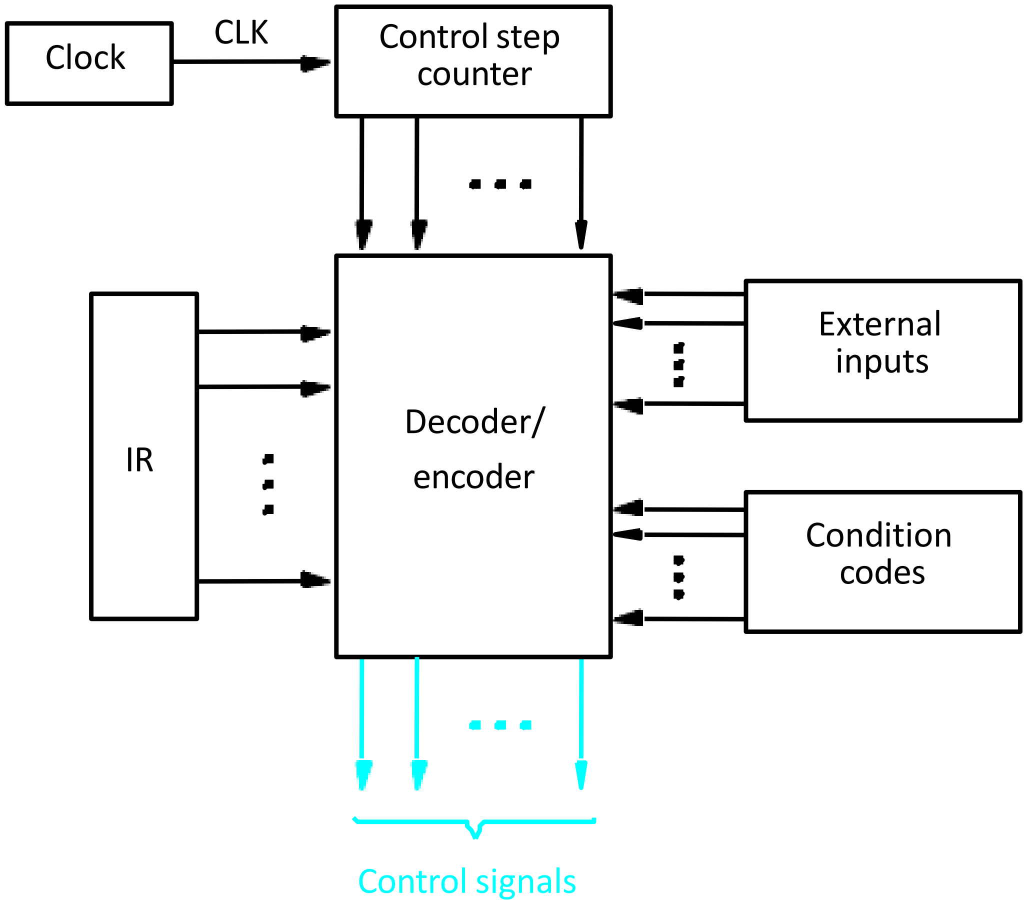


Figure 7.10. Control unit organization.

Detailed Block Description

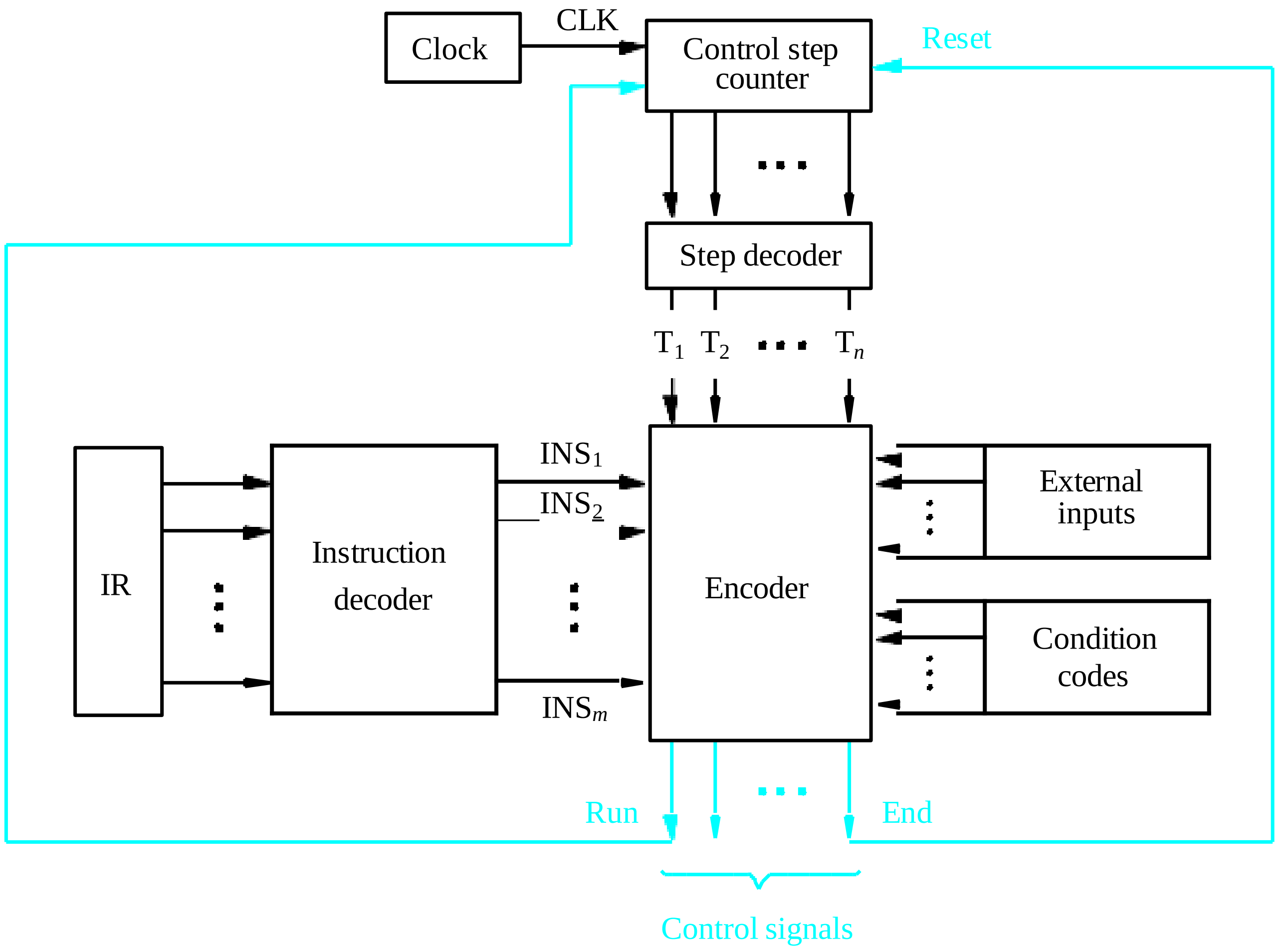


Figure 7.11. Separation of the decoding and encoding functions.

Generating Z_{in}

- $Z_{in} = T_1 + T_6 \cdot \text{ADD} + T_4 \cdot \text{BR} + \ldots$

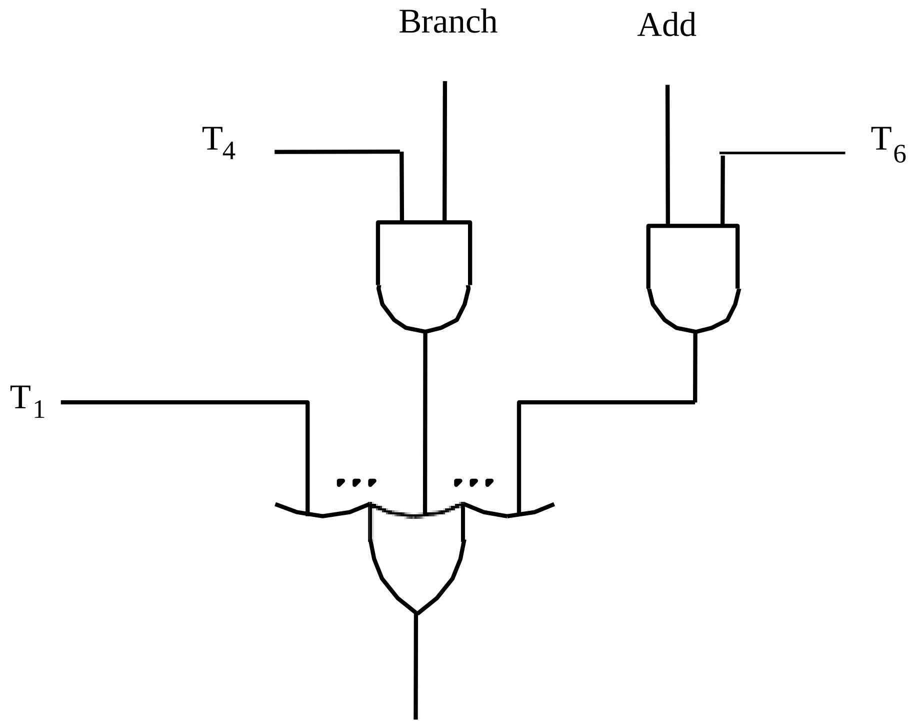


Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

- $End = T_7 \cdot ADD + T_5 \cdot BR + (T_5 \cdot N + T_4 \cdot \overline{N}) \cdot BRN + \ldots$

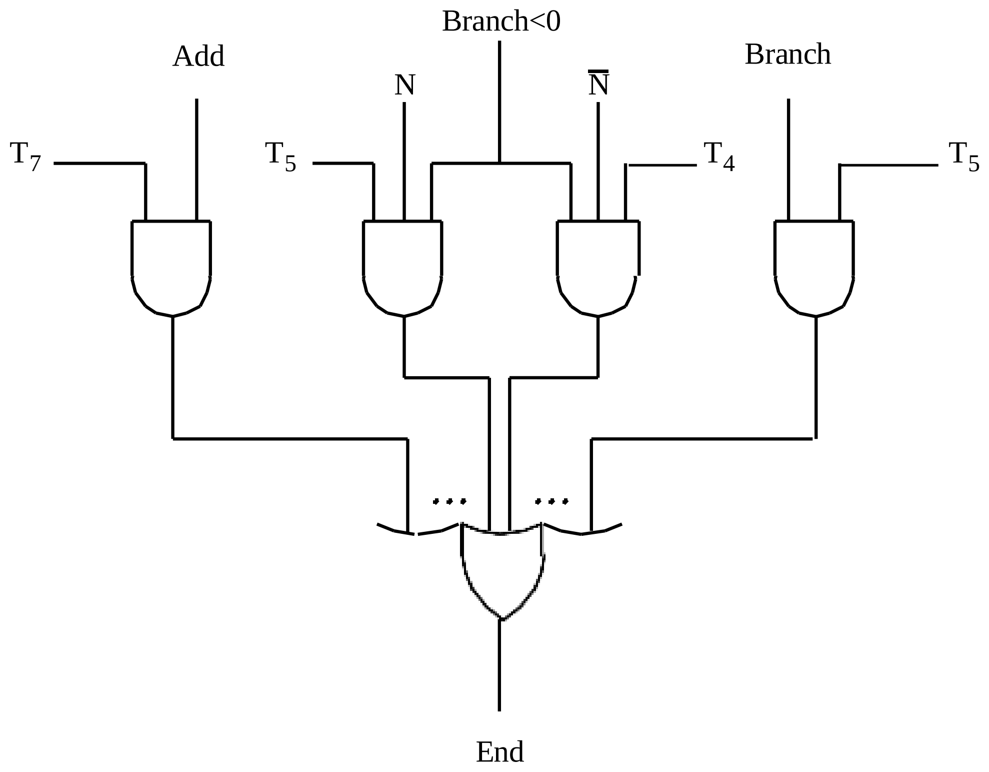


Figure 7.13. Generation of the End control signal.

A Complete Processor

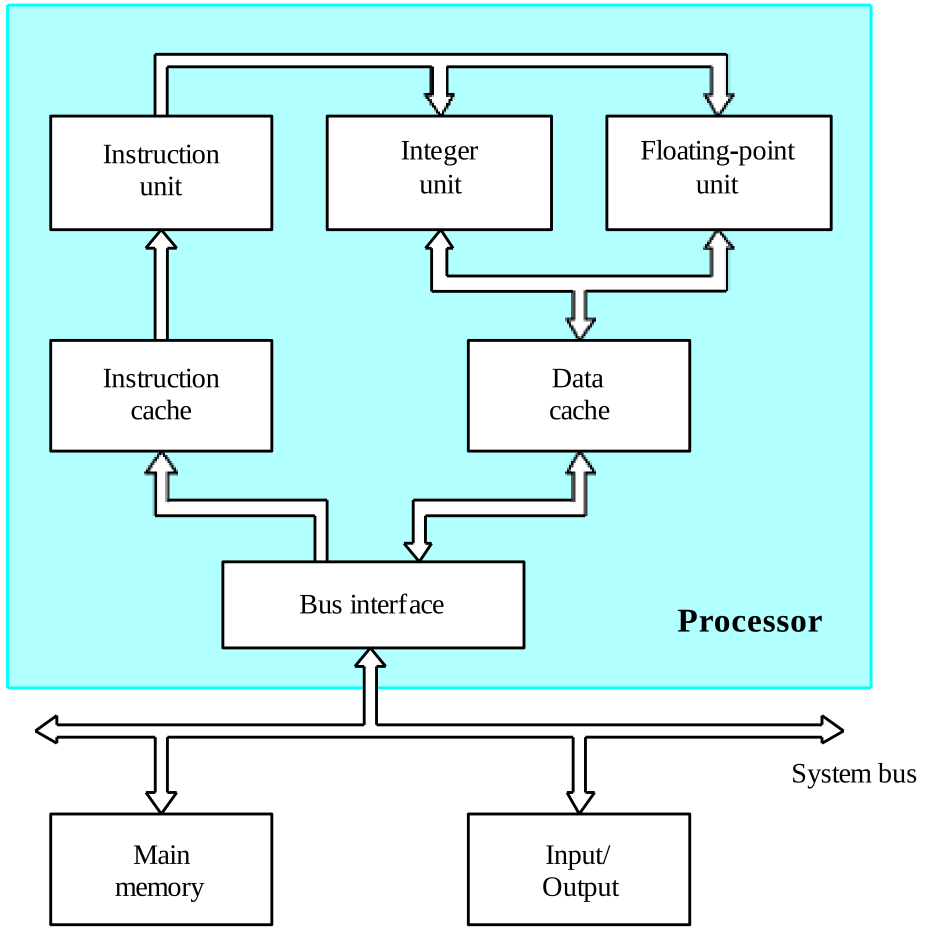


Figure 7.14. Block diagram of a complete processor.

END