What is 8086 Architecture?

- **Contents at a glance:**

✓ Architecture of 8086 microprocessor

✓ Register organization

✓ 8086 flag register and its functions

✓ Addressing modes of 8086

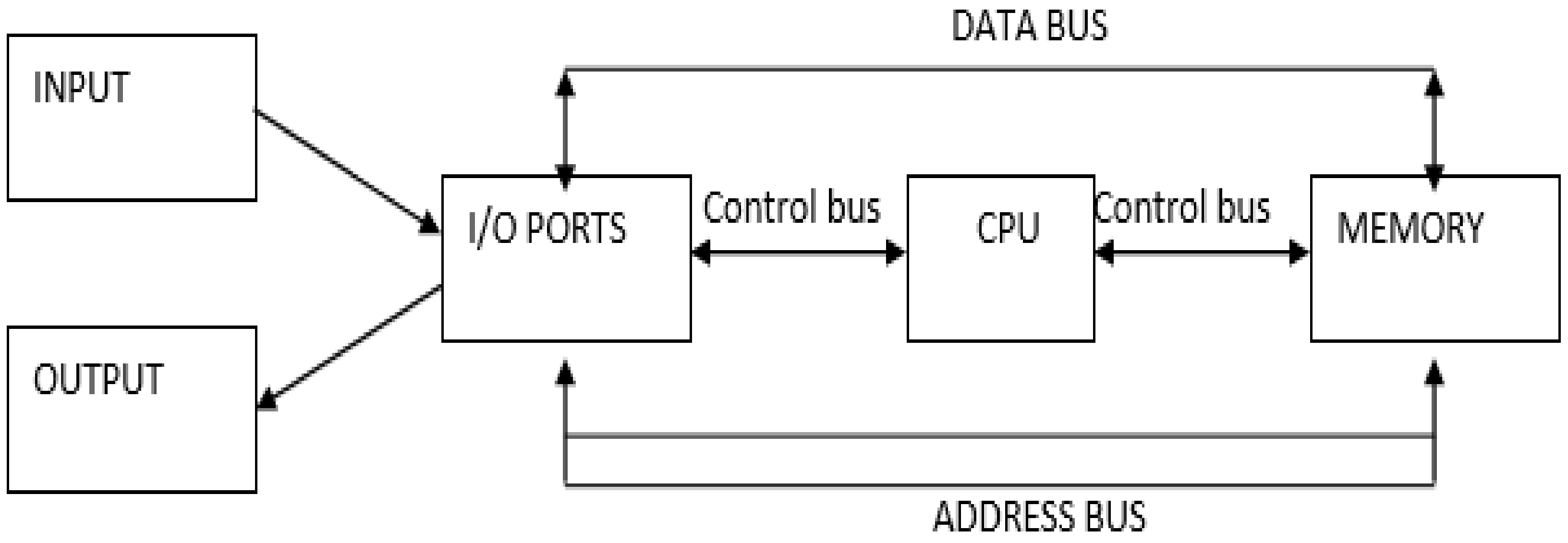
✓ Pin diagram of 8086

✓ Minimum mode & Maximum mode system operation

✓ Timing diagrams

OVERVIEW OF A SIMPLE MICRO COMPUTER:

CPU: The central processing unit or CPU controls the operation of the computer. It fetches binary-coded instruction of the computer. It fetches binary-coded instructions from memory, decodes the instructions into a series of simple actions, and carries out these actions. The CPU contains an arithmetic logic unit, or ALU. Which can perform
add, subtract, OR, AND, invert, or exclusive-OR operations on binary words when instructed to do so.



What is a Microprocessor?

- **The word comes from the combination micro and processor**.

  – Processor means a device that processes numbers, specifically binary numbers, 0's and 1's.

  – Micro is a new addition.

  – In the late 1960's, processors were built using discrete elements.

  – These devices performed the required operation, but were too large and too slow.

  – In the early 1970's the microchip was invented. All of the components that made up the processor

  were now placed on a single piece of silicon. The size became several thousand times smaller and the

  speed became several hundred times faster.

Definition of the Microprocessor

The microprocessor is a *programmable device* that *takes in numbers*, performs on them *arithmetic or logical operations* according to the *program stored in memory* and then *produces* other numbers as a result.

Lets expand each of the underlined words:

Programmable device:

- The microprocessor can perform different sets of operations on the data it receives depending on the sequence of *instructions* supplied in the given program.
- By changing the program, the microprocessor manipulates the data in different ways.

Instructions:

- Each microprocessor is designed to execute a specific group of operations. This group of operations is called an instruction set. This instruction set defines what the microprocessor can and cannot do.

History of Microprocessors:

Processor	No. of bits	Clock speed (Hz)	Year of introduction
4004	4	740K	1971
8008	8	500K	1972
8080	8	2M	1974
8085	8	3M	1976
8086	16	5, 8 or 10M	1978
8088	16	5, 8 or 10M	1979
80186	16	6M	1982
80286	16	8M	1982
80386	32	16 to 33M	1986
80486	32	16 to 100M	1989
Pentium	32	66M	1993
Pentium II	32	233 to 500M	1997
Pentium III	32	500M to 1.4G	1999
Pentium IV	32	1.3 to 3.8G	2000
Dual core	32	1.2 to 3 G	2006
Core 2 Duo	64	1.2 to 3G	2006

8086 Microprocessor features:

- It is 16-bit microprocessor
- It has a 16-bit data bus, so it can read data from or write data to memory and ports either 16-bit or 8-bit at a time.
- It has 20 bit address bus and can access up to 2^{20} memory locations (1 MB).
- It can support up to 64K I/O ports
- It provides 14, 16-bit registers
- It has multiplexed address and data bus AD_0-AD_{15} & A_{16}-A_{19}
- It requires single phase clock with 33% duty cycle to provide internal timing.
- Prefetches up to 6 instruction bytes from memory and queues them in order to speed up the processing.
- 8086 supports 2 modes of operation
  - Minimum mode
  - Maximum mode

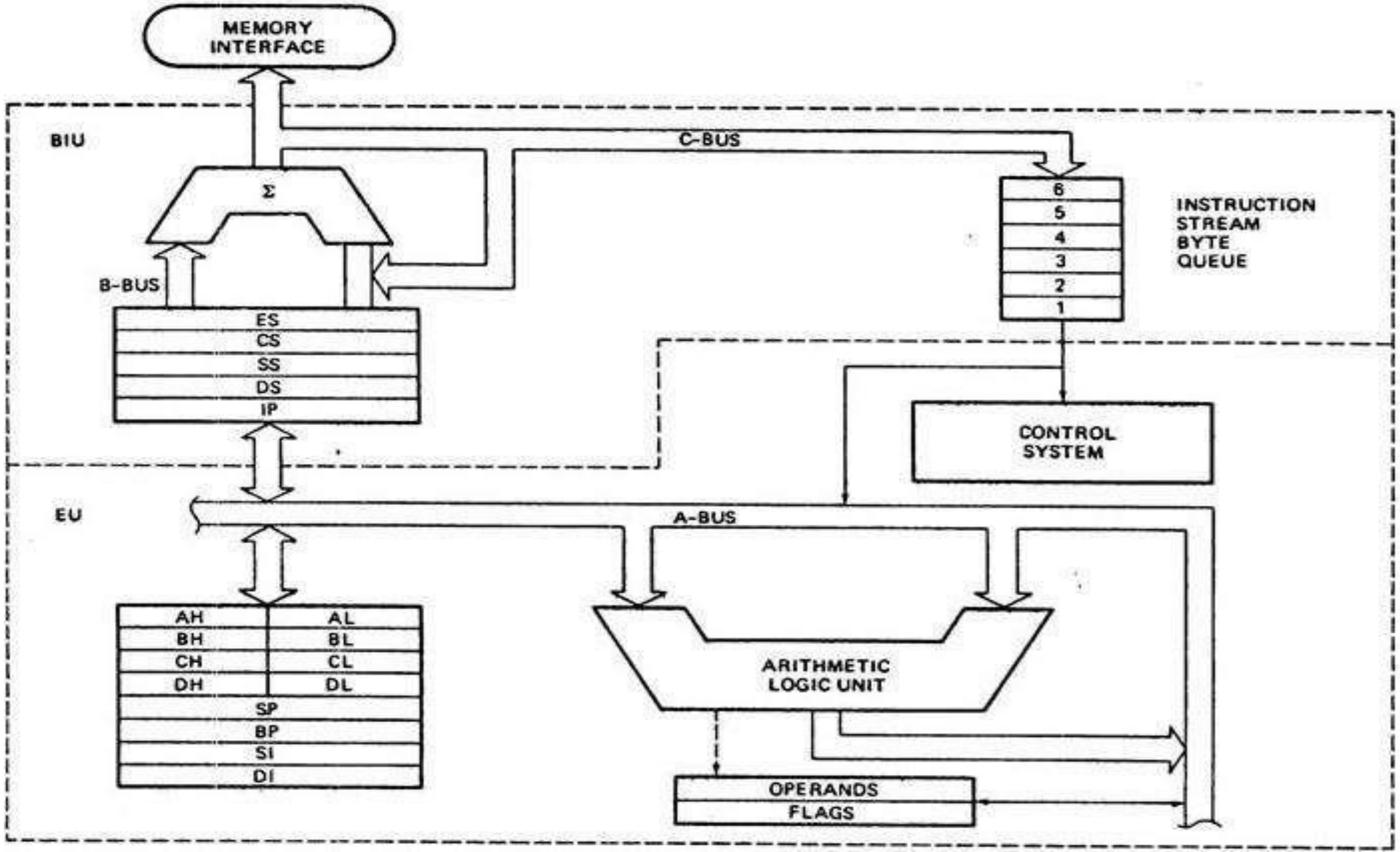
Architecture of 8086 microprocessor:

As shown in the below figure, the 8086 CPU is divided into two independent functional parts

1).Bus Interface Unit(BIU)

2).Execution Unit(EU)

Dividing the work between these two units' speeds up processing.



Architecture of 8086

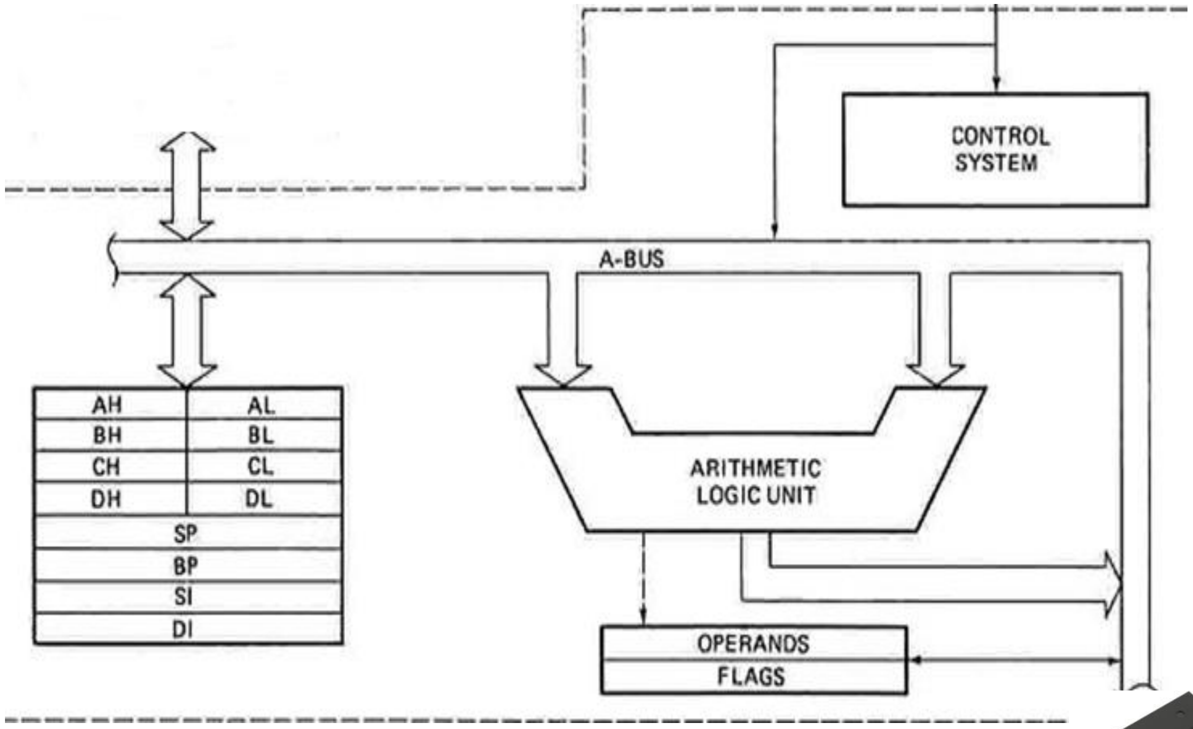
- *The 8086 CPU logic has been partitioned into two functional units namely*
  - ***Bus Interface Unit (BIU)***
  - ***Execution Unit (EU).***
- *The major reason for this separation is to increase the processing speed of the processor.*

- *The BIU sends out addresses, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.*

- *In other words, the BIU handles all transfers of data and addresses on the buses for the execution unit.*

- *Execution Unit (EU) tells the BIU where to fetch instructions or data from, decodes instructions, and execute instructions.*

- The ***Execution Unit (EU)*** has
  - *Control unit*
  - *Instruction decoder*
  - *Arithmetic and Logical Unit (ALU)*
  - *General registers*
  - *Pointers*
  - *Index registers*
  - *Flag register*



Execution Unit (EU)

- ***Control unit*** ***is responsible for the co-ordination of all other units of the processor***
- ***ALU*** ***performs various arithmetic and logical operations over the data***
- ***The*** ***Instruction Decoder*** ***translates the instructions fetched from the memory into a series of actions that are carried out by the EU***

<u>Architecture of 8086 microprocessor:</u>

- The main functions of EU are:
  - Decoding of Instructions
  - Execution of instructions
    - Steps
      - EU extracts instructions from top of queue in BIU
      - Decode the instructions
      - Generates operands if necessary
      - Passes operands to BIU & requests it to perform read or write bus cycles to memory or I/O
      - Perform the operation specified by the instruction on operands

Register organization:

- 8086 has a powerful set of registers known as ***general purpose registers*** and ***special purpose registers****.*
- All of them are 16-bit registers.
- ***General purpose registers:***
  - These registers can be used as either 8-bit registers or 16-bit registers.
  - They may be either used for holding data, variables and intermediate results temporarily or for other purposes like a counter or for storing offset address for some particular addressing modes etc.
- ***Special purpose registers:***
  - These registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.
- **The 8086 registers are classified into the following types:**
  - General Data Registers
  - Segment Registers
  - Pointers and Index Registers
  - Flag Register

General Data Registers:

The registers *AX, BX, CX* and *DX* are the general purpose 16-bit registers.

AX is used as 16-bit accumulator. The lower 8-bit is designated as *AL* and higher 8-bit is designated as *AH*.

AL can be used as an 8-bit accumulator for 8-bit operation.

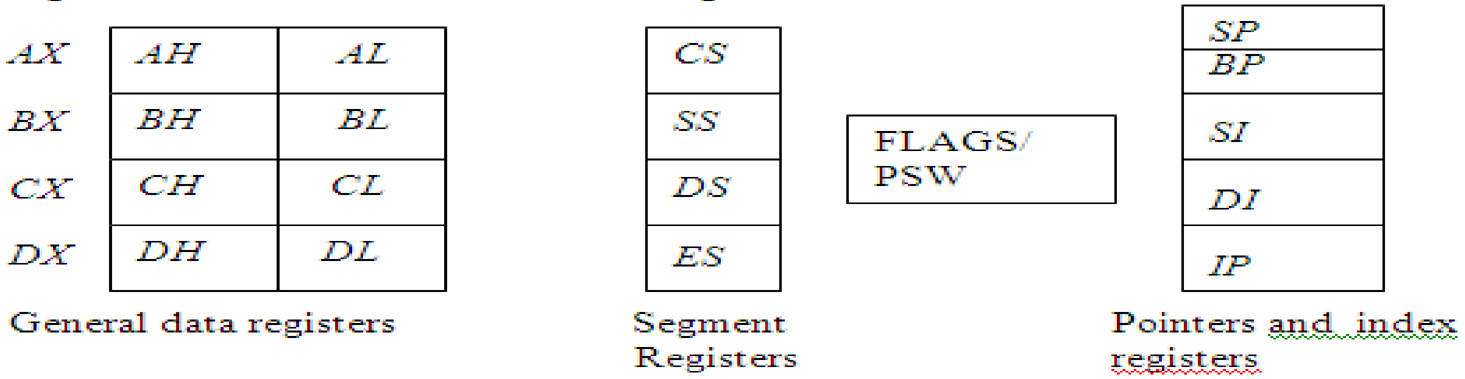
All data register can be used as either 16 bit or 8 bit. *BX* is a 16 bit register, but *BL* indicates the lower 8-bit of *BX* and *BH* indicates the higher 8-bit of *BX*.

The register *BX* is used as offset storage for forming physical address in case of certain addressing modes. The register *CX* is used default counter in case of string and loop instructions.

DX register is a general purpose register which may be used as an implicit operand or

destination in case of a few instructions.

The register set of 8086 can be categorized into 4 different groups. The register organization of 8086 is shown in the figure.



Register organization of 8086

Segment Registers:

There are 4 segment registers. They are:

- Code Segment Register(CS)
- Data Segment Register(DS)
- Extra Segment Register(ES)
- Stack Segment Register(SS)

- The 8086 architecture uses the concept of **segmented memory**. 8086 able to address a memory capacity of 1 megabyte and it is byte organized. This 1 megabyte memory is divided into 16 logical segments. Each segment contains 64 kbytes of memory.
- Code segment register (CS): is used for addressing memory location in the code segment of the memory, where the executable program is stored.
- Data segment register (DS): points to the data segment of the memory where the data is stored.
- Extra Segment Register (ES) : also refers to a segment in the memory which is another data segment in the memory.
- Stack Segment Register (SS): is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.
- While addressing any location in the memory bank, the **physical address** is calculated from two parts:
- *Physical address= segment address + offset address*
- The first is segment address, the segment registers contain 16-bit segment base addresses, related to different segment.
- The second part is the offset value in that segment.

Pointers and Index Registers:

The index and pointer registers are given below:

- IP—Instruction pointer-store memory location of next instruction to be executed
- BP—Base pointer
- SP—Stack pointer
- SI—Source index
- DI—Destination index

- The pointers registers contain offset within the particular segments.
  - The pointer register *IP* contains offset within the code segment.
  - The pointer register *BP* contains offset within the data segment.
  - Thee pointer register *SP* contains offset within the stack segment.
- The index registers are used as general purpose registers as well as for offset storage in case of indexed, base indexed and relative base indexed addressing modes.
- The register *SI* is used to store the offset of source data in data segment.
- The register *DI* is used to store the offset of destination in data or extra segment.
- The index registers are particularly useful for string manipulation.

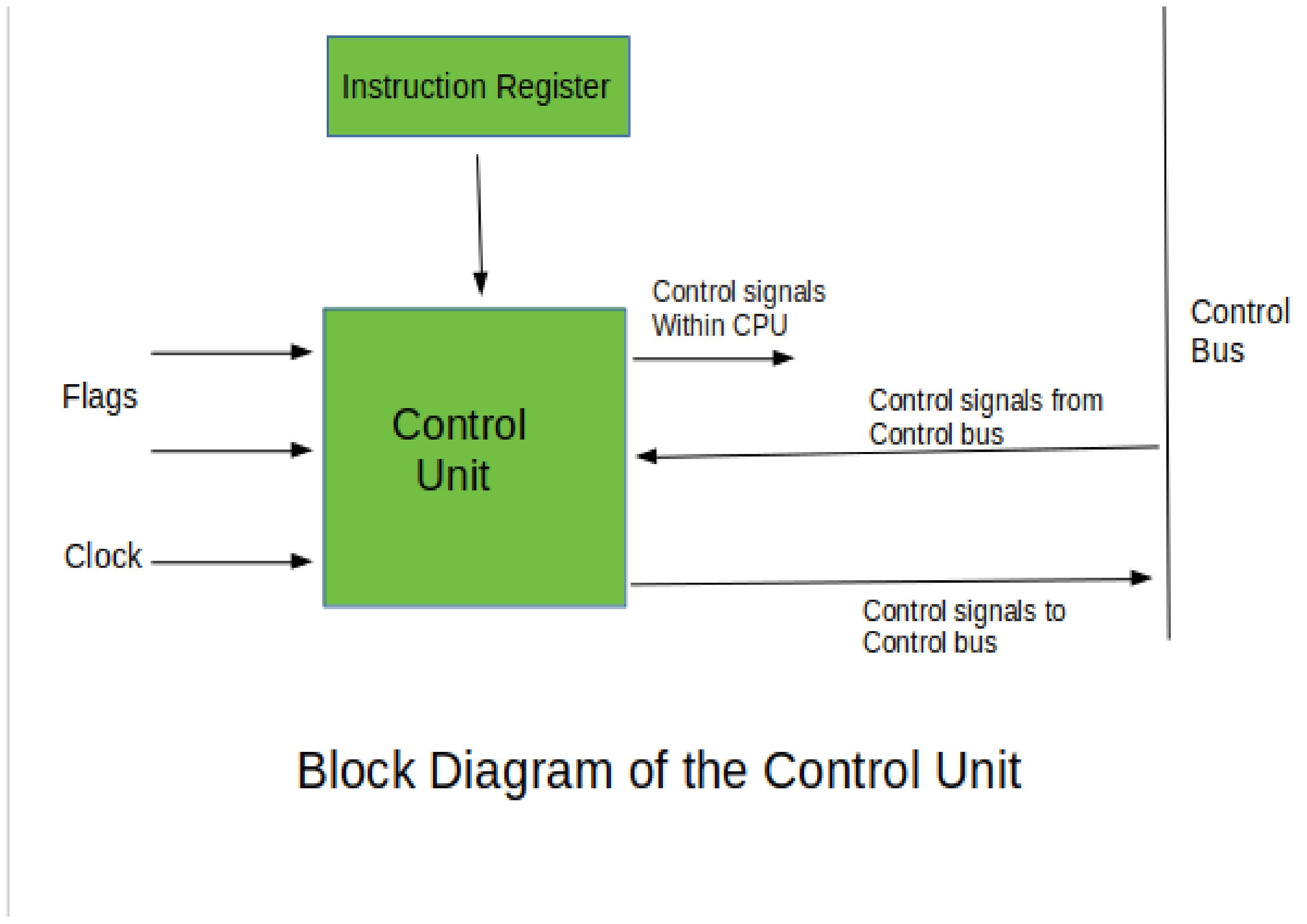
Introduction of Control Unit and its Design

- **Control Unit** is the part of the computer's central processing unit (CPU), which directs the operation of the processor.
- It was included as part of the Von Neumann Architecture by John von Neumann.
- It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor.
- It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

Introduction of Control Unit and its Design

- A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor.
- The computer's processor then tells the attached hardware what operations to perform.
- The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer.
- Examples of devices that require a CU are:
- **1) Control Processing Units(CPUs)**
- **2) Graphics Processing Units(GPUs**)

Introduction of Control Unit and its Design


Instruction Register
Control Unit
Flags
Clock
Control signals Within CPU
Control signals from Control bus
Control signals to Control bus
Control Bus

Block Diagram of the Control Unit

Introduction of Control Unit and its Design

- **Functions of the Control Unit –**
- It coordinates the sequence of data movements into, out of, and between a processor’s many sub-units.
- It interprets instructions.
- It controls data flow inside the processor.
- It receives external instructions or commands to which it converts to sequence of control signals.
- It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
- It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

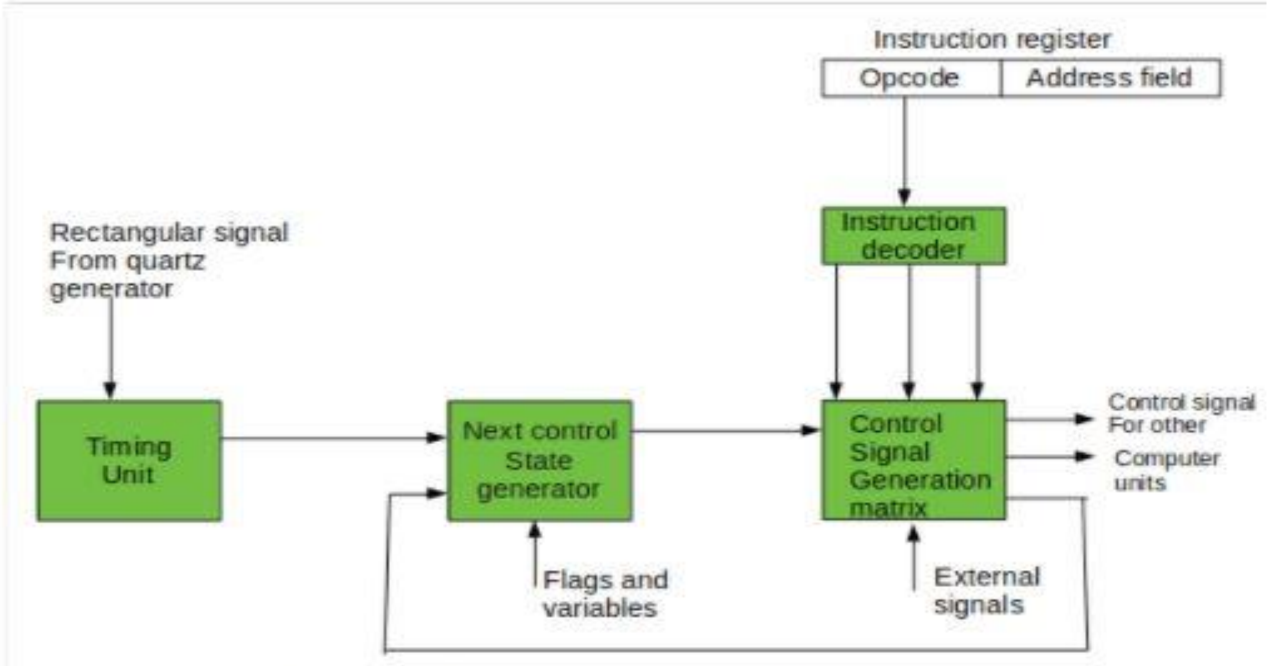
Introduction of Control Unit and its Design

- **Types of Control Unit –**
  There are two types of control units:
  1) Hardwired control unit and
  2) Microprogrammable control unit.

  1) Hardwired Control Unit :
  In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure.
- The operation code of an instruction contains the basic data for control signal generation.
- In the instruction decoder, the operation code is decoded.
- The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.

Introduction of Control Unit and its Design

- Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle.


Instruction register
Opcode
Address field
Instruction decoder
Rectangular signal From quartz generator
Timing Unit
Next control State generator
Control Signal Generation matrix
Control signal For other Computer units
Flags and variables
External signals

Block diagram of a hardwired control unit of a computer

Introduction of Control Unit and its Design

- A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix.
- This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator.
- When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching.
- Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered.
- A change of any of the earlier mentioned signals stimulates the change of the control unit state.
- This causes that a new respective input is generated for the control signal generator matrix.

Introduction of Control Unit and its Design

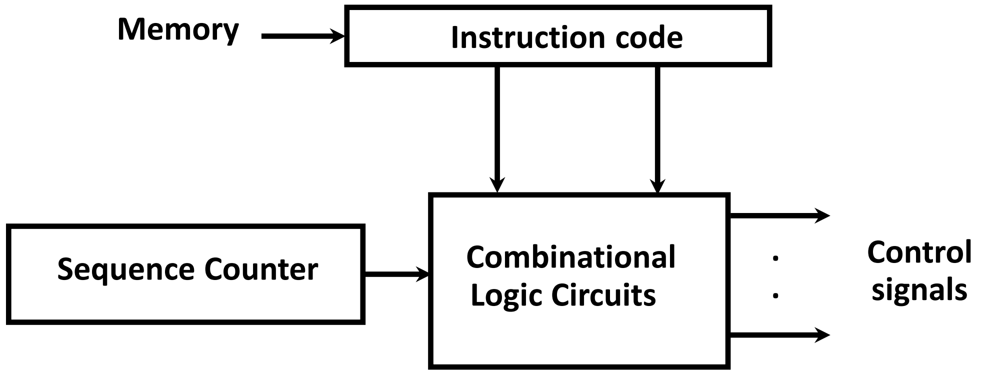
- When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing).
- The last states in the cycle are control states that commence fetching the next instruction of the program:
- sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer.
- When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

Introduction of Control Unit and its Design

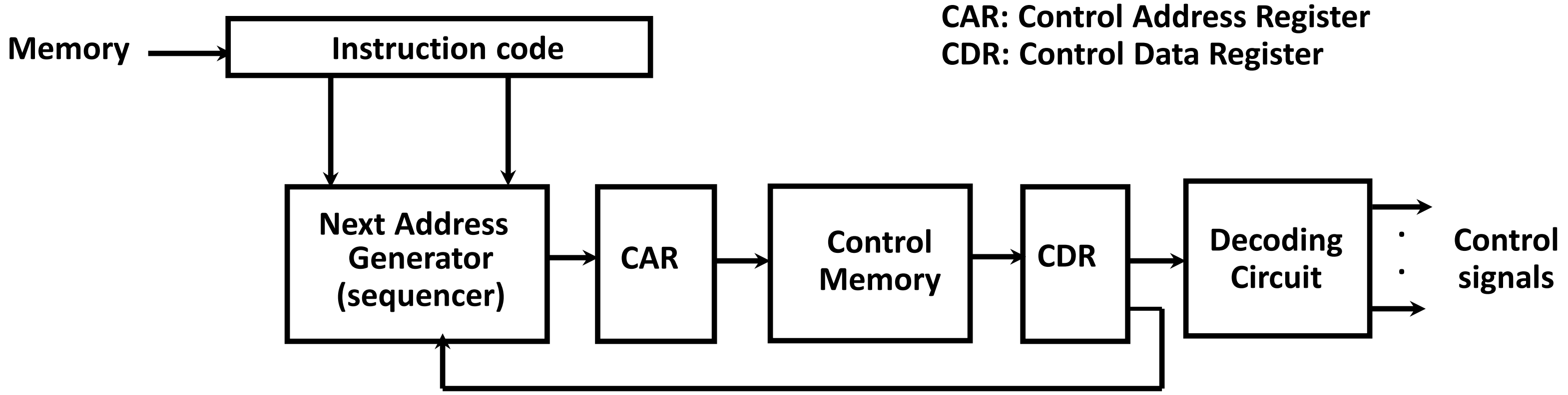
- **Microprogrammable control unit :**
- The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.
- In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way.

Control Unit Implementation

- Hardwired



- Microprogrammed



CAR: Control Address Register
CDR: Control Data Register

Microprogrammed Control Unit

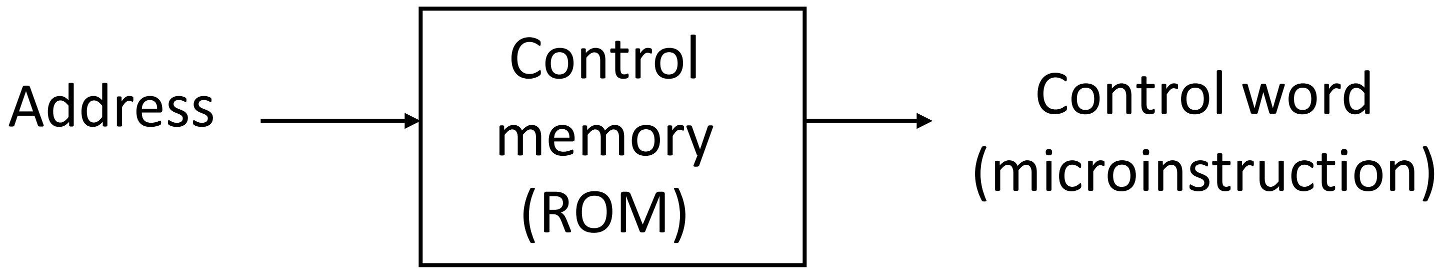
- Control signals
  - Group of bits used to select paths in multiplexers, decoders, arithmetic logic units
- Control variables
  - Binary variables specify microoperations
    - Certain microoperations initiated while others idle
- Control word
  - String of 1’s and 0’s represent control variables

Microprogrammed Control Unit

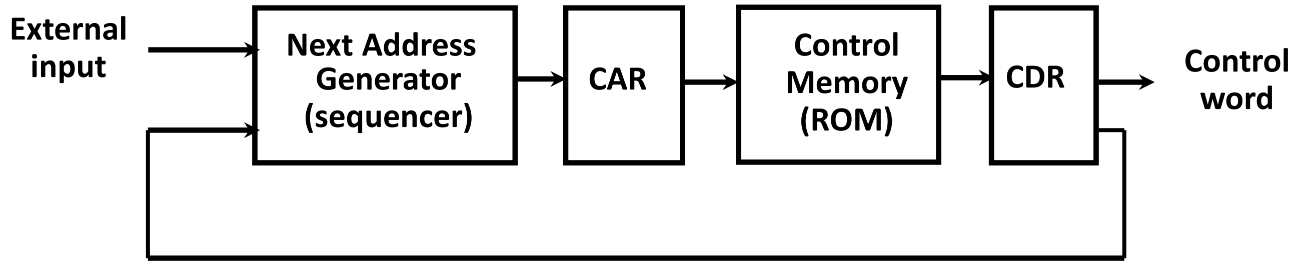
- Control memory
  - Memory contains control words
- Microinstructions
  - Control words stored in control memory
  - Specify control signals for execution of microoperations
- Microprogram
  - Sequence of microinstructions

Control Memory

- Read-only memory (ROM)
- Content of word in ROM at given address specifies microinstruction
- Each computer instruction initiates series of microinstructions (microprogram) in control memory
- These microinstructions generate microoperations to
  - Fetch instruction from main memory
  - Evaluate effective address
  - Execute operation specified by instruction
  - Return control to fetch phase for next instruction



Microprogrammed Control Organization



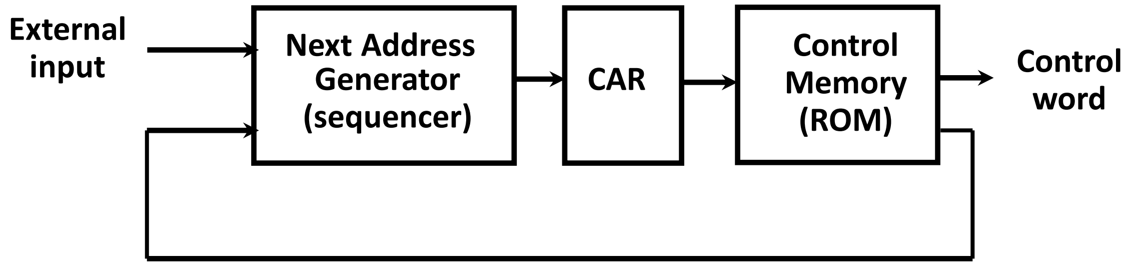
- Control memory
  - Contains microprograms (set of microinstructions)
  - Microinstruction contains
    - Bits initiate microoperations
    - Bits determine address of next microinstruction
- Control address register (CAR)
  - Specifies address of next microinstruction

Microprogrammed Control Organization

- Next address generator (microprogram sequencer)
  - Determines address sequence for control memory
- Microprogram sequencer functions
  - Increment CAR by one
  - Transfer external address into CAR
  - Load initial address into CAR to start control operations

Microprogrammed Control Organization

- Control data register (CDR)- or pipeline register
  - Holds microinstruction read from control memory
  - Allows execution of microoperations specified by control word simultaneously with generation of next microinstruction
- Control unit can operate without CDR



Peripheral Devices

- The input-output subsystem of a computer, referred to as 110, provides an efficient mode of communication between the central system and the outside environment.
- Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded and displayed for the user.
- A computer serves no useful purpose without the ability
- to receive information from an outside source and to transmit results in a meaningful form.
- The most familiar means of entering information into a computer is through a typewrite"' like keyboard that allows a person to enter alphanumeric information directly.

Peripheral Devices

- Every time a key is depressed, the terminal sends a binary
- coded character to the computer.
- The fastest possible speed for entering Information this way depends on the person's typing speed.
- On the other hand, the central processing unit is an extremely fast device capable of performing operations at very high speed.
- When input information is transferred to the processor via a slow keyboard, the processor will be idle most of the time while waiting for the information to arrive.

Peripheral Devices

- Devices that are under the direct control of the computer are said to be connected on-line.
- These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be part of the total computer system.
- **Input or output devices attached to the computer are also called peripherals** .
- Among the most common peripherals are keyboards, display units, and printers.
- Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.
- Peripherals are electromechanical and electromagnetic devices of some complexity.

Peripheral Devices

Monitor and Key board:

- Video monitors are the most commonly used peripherals.
- They consist of a keyboard as the input device and a display unit as the output device.
- There are different types of video monitors, but the most popular use a cathode ray tube **(CRT).**
- The CRT contains an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube.
- The beam can be deflected horizontally and vertically.
- A characteristic feature of display devices is a **cursor** that marks the position in the screen where the next character will be inserted.
- The cursor can be moved to any position in the screen, to a single character, the beginning of a word, or to any line.
- Edit keys add or delete information based on the cursor position.
- The display terminal can operate in a single-character mode where all characters entered on the screen through the **keyboard** are transmitted to the computer simultaneously.

Peripheral Devices

Printer:

- Printers provide a permanent record on paper of computer output data or text.
- There are three basic types of character printers:

  1) daisywheel, 2) dot matrix 3)laser printers.
- The **daisywheel printer** contains a wheel with the characters placed along the circumference.
- To print a character, the wheel rotates to the proper position and an energized magnet then presses the letter against the ribbon.
- The **dot matrix printer** contains a set of dots along the printing mechanism.
- For example, a 5 x 7 dot matrix printer that prints 80 characters per line has seven horizontal lines, each consisting of 5 x 80 = 400 dots.
- Each dot can be printed or not, depending on the specific characters that are printed on the line.
- The **laser printer** uses a rotating photographic drum magnetic tape magnetic disk that is used to imprint the character images.
- The pattern is then transferred onto paper in the same manner as a copying machine.

Peripheral Devices

- **Magnetic tape:** Magnetic tapes are used mostly for storing files of data:
- for example, a company's payroll record.
- Access is sequential and consists of records that can be accessed one after another as the tape moves along a stationary read-write mechanism.
- It is one of the cheapest and slowest methods for storage and has the advantage that tapes can be removed when not in use.
- **Magnetic disks** :
- Magnetic disks have high-speed rotational surfaces coated with magnetic material.
- Access is achieved by moving a read-write mechanism to a track in the magnetized surface.
- Disks are used mostly for bulk storage of programs and data.

Input-Output Interface

- Input-output interface provides a method for transferring information between internal storage and external I/0 devices.
- Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.
- The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.
- The major differences are:

1) Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices.

Therefore, a conversion of signal values may be required.

2). The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.

3). Data codes and formats in peripherals differ from the word format in the CPU and memory.

4). The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

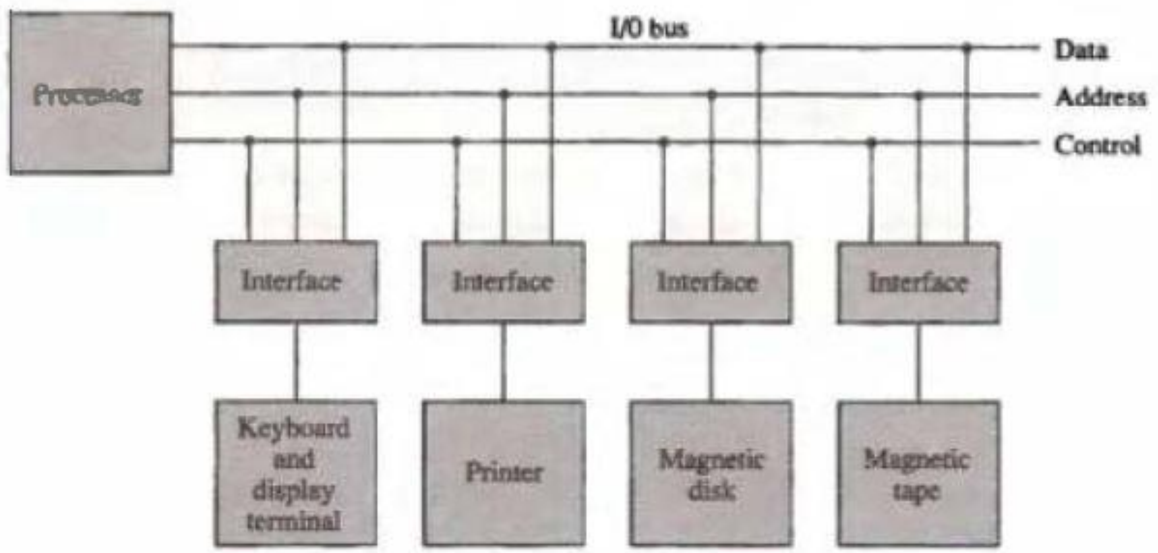
Input-Output Interface

- **Interface:** To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers.
- These components are called interface units because they interface between the processor bus and the peripheral device.
- In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral

I/O Bus and Interface Modules

- A typical communication link between the processor and several peripherals.
- The 1/0 bus consists of data lines, address lines, and control lines.

Figure 11.1 Connection of I/O bus to input-output devices



I/O Bus and Interface Modules

- The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit.
- The interpretation of the command depends on the peripheral that the processor is addressing.
- There are four types of commands that an interface may receive.
- They are classified as **control, status, data output, and data input.**
- **A control command:** is issued to activate the peripheral and to inform it what to do.
- **For example,** a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.

I/O Bus and Interface Modules

- **A status command:** is used to test various status conditions in the interface and the peripheral.
- **For example**, the computer may wish to check the status of the peripheral before a transfer is initiated.
- During the transfer, one or more errors may occur which are detected by the interface.
- **A data output command :**causes the interface to respond by transferring data from the bus into one of its registers. Consider an example with a tape unit.
- The computer starts the tape moving by issuing a control command.
- The processor then monitors the status of the tape by means of a status command.
- When the tape is in the correct position, the processor issues a data output command.
- **The data input command :** is the opposite of the data output.
- In this case the interface receives an item of data from the peripheral and places it in its buffer register.
- The processor checks if data are available by means of a status command and then issues a data input command.
- The interface places the data onthe data lines, where they are accepted by the processor.

I/O versus Memory Bus

- The processor must communicate with the memory unit.
- Like the i/o bus, the memory bus contains data, address, and read/write control lines.
- There are three ways that computer buses can be used to communicate with memory and I/O.

    1. Use two separate buses, one for memory and the other for I/O.

    2. Use one common bus for both memory and I/O but have separate control lines for each.

    3. Use one common bus for memory and i/o with common control lines.

Isolated versus Memory-Mapped I/O

- Many computers use one common bus to transfer information between memory or i/o and the CPU.
- The distinction between a memory transfer and i/o transfer is made through separate read and write lines.
- The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines.
- The i/o read and i/o write control lines are enabled during an i/o transfer.
- The memory read and memory write control lines are enabled during a memory transfer.
- This configuration isolates all i/o interface addresses from the addresses assigned to memory and is referred to as the isolated i/o method for assigning addresses in a common bus.
- The isolated i/o method isolates memory and i/o addresses so that memory address values are not affected by interface address assignment since each has its own address space.

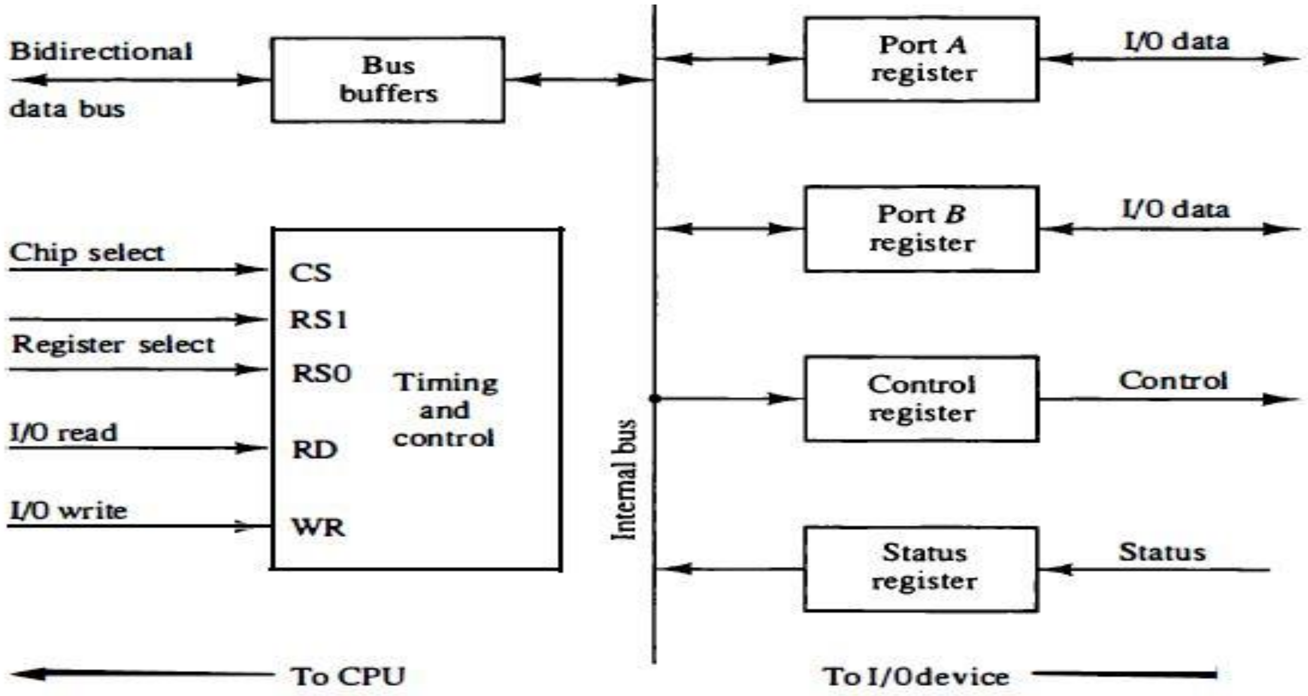
Isolated versus Memory-Mapped I/O

- The other alternative is to use the same address space for both memory and i/o.
- This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses.
- This configuration is referred to as **memory-mapped i/o.**
- The computer treats an interface register as being part of the memory system.
- The assigned addresses for interface registers cannot be used for memory words, which reduces the memory address range available.

Example of I/O Interface

- It consists of two data registers called ports, a control register, a status register , bus buffers, and timing and control circuits.
- The interface communicates with the CPU through the data bus.
- The chip select and register select inputs determine the address assigned to the interface.
- The I/0 read and write are two control lines that specify an input or output, respectively.
- The four registers communicate directly with the I/0 device attached to the interface.

Example of I/O Interface



CS	RS1	RS0	Register selected
0	×	×	None: data bus in high-impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

Program-Controlled I/O

- The example described above illustrates program controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor polls the devices
- There are two other commonly used mechanisms for implementing I/O operations**: interrupts** and **direct memory access** ⍰
- **Interrupts:** synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation ⍰
- **Direct memory access:** it involves having the device interface transfer data directly to or from the memory

Interrupts

- To avoid the processor being not performing any useful computation, a hardware signal called an interrupt to the processor can do it. At least one of the bus control lines, called an **interrupt-request** line, is usually dedicated for this purpose
- An **interrupt-service** routine usually is needed and is executed when an interrupt request is issued
- ¾On the other hand, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. An **interrupt-acknowledge** signal serves this function

Modes of Transfer

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

  1. Programmed I/O

  2. Interrupt-initiated I/O

  3. Direct memory access (DMA)

Programmed I/O

- Programmed i/o operations are the result of i/o instructions written in the computer program.
- Each data item transfer is initiated by an instruction in the program.
- Usually, the transfer is to and from a CPU register and peripheral.
- Other instructions are needed to transfer the data to and from CPU and memory.
- Transferring data under program control requires constant monitoring of the peripheral by the CPU.
- Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

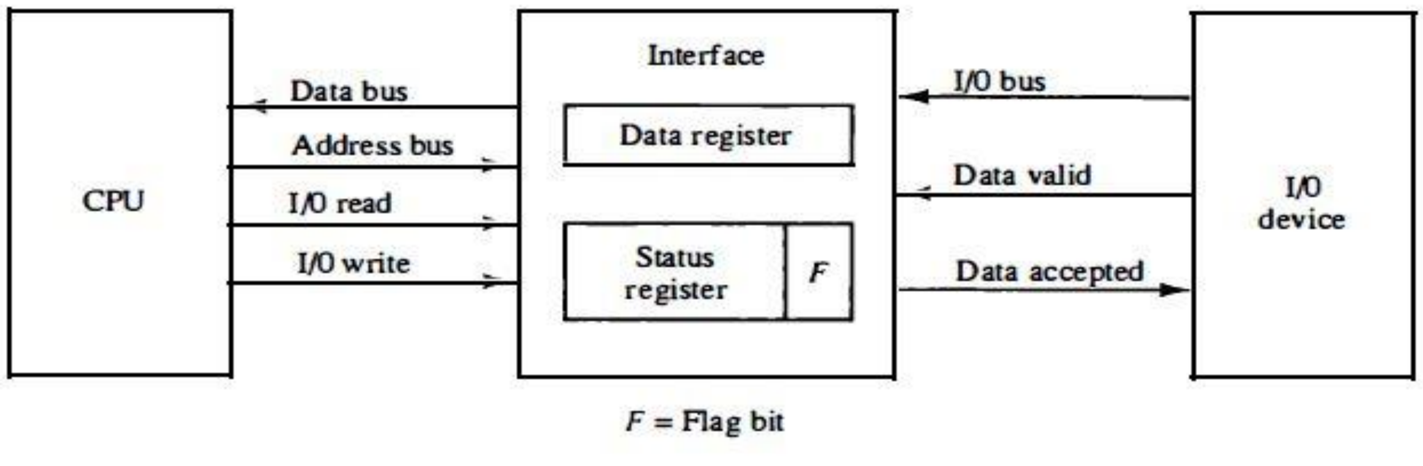
Programmed I/O

- In the programmed i/o method, the i/o device does not have direct access to memory.
- A transfer from an i/o device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.
- Other instructions may be needed to verify that the data are available from the device and to count the numbers of words transferred.

Programmed I/O

- 1. Read the status register.
- 2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
- 3. Read the data register.

Figure 11-10 Data transfer from I/O device to CPU.



Programmed I/O

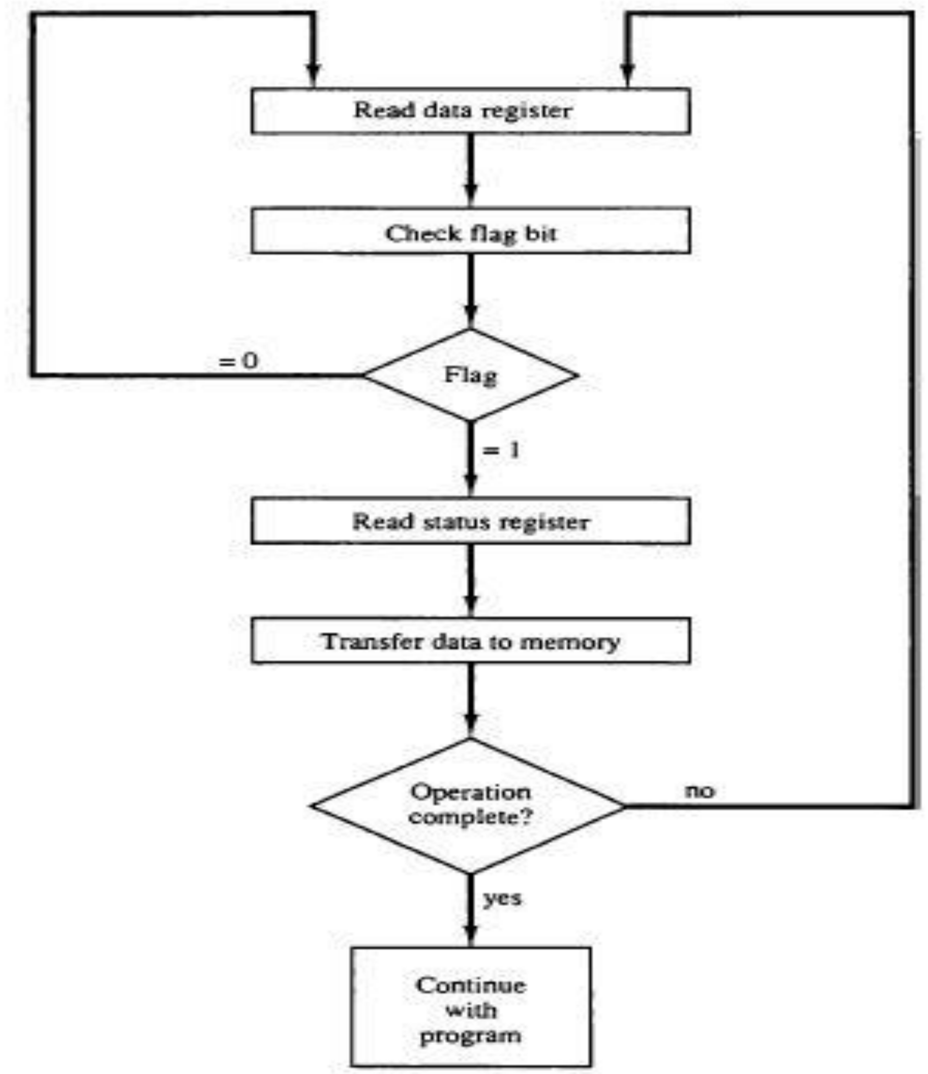


Figure 11-11 Flowchart for CPU program to input data

Interrupt-Initiated I/O

- In the programmed 1/0 method, the CPU stays in a program loop until the 110 unit indicates that it is ready for data transfer.
- This is a time-consuming process since it keeps the processor busy needlessly.
- It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.
- In the meantime the CPU can proceed to execute another program.
- The interface meanwhile keeps monitoring the device.
- When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer.
- Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the i/o transfer, and then returns to the task it was originally performing.

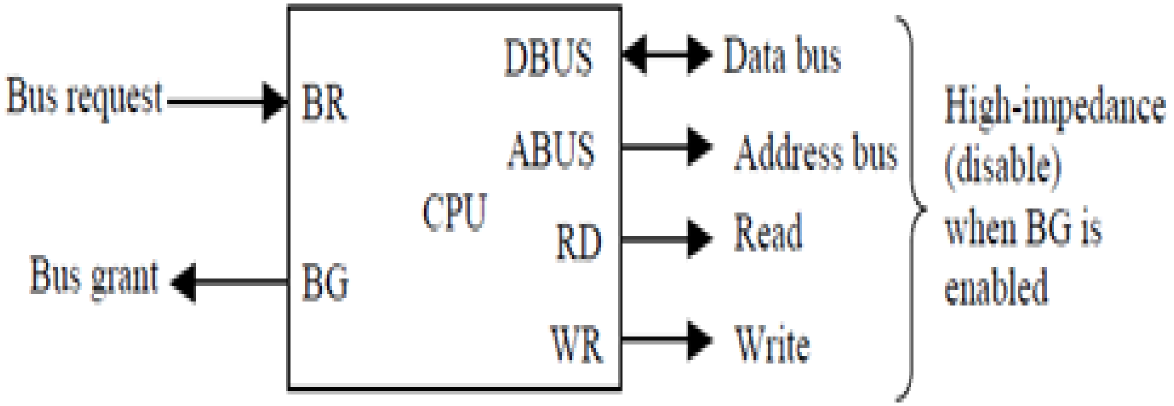
Interrupt-Initiated I/O

Vectored interrupt and Nonvectored interrupt:

- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required i/o transfer.
- The way that the processor chooses the branch address of the service routine varies from one unit to another. In principle, there are two methods for accomplishing this.
- One is called **vectored interrupt** and the other, **nonvectored interrupt**.
- In a non vectored interrupt, the branch address is assigned to a fixed location in memory.
- In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.
- In some computers the interrupt vector is the first address of the VO service routine.

Direct Memory Access (DMA)

- **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU.
- Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU.
- This type of data transfer technique is known as DMA or direct memory access.
- During DMA the CPU is idle and it has no control over the memory buses.
- The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.



Direct Memory Access (DMA)

- **Bus Request :** It is used by the DMA controller to request the CPU to relinquish the control of the buses.
- **Bus Grant :** It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Direct Memory Access (DMA)

- The CPU activates the Bus grant (BG) output to inform the external DMA that the buses are in the high -impedance state.
- The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention.
- When the DMA terminates the transfer, it disables the bus request line.
- The CPU disables the bus grant, takes control of the buses, and returns to its normal operation.

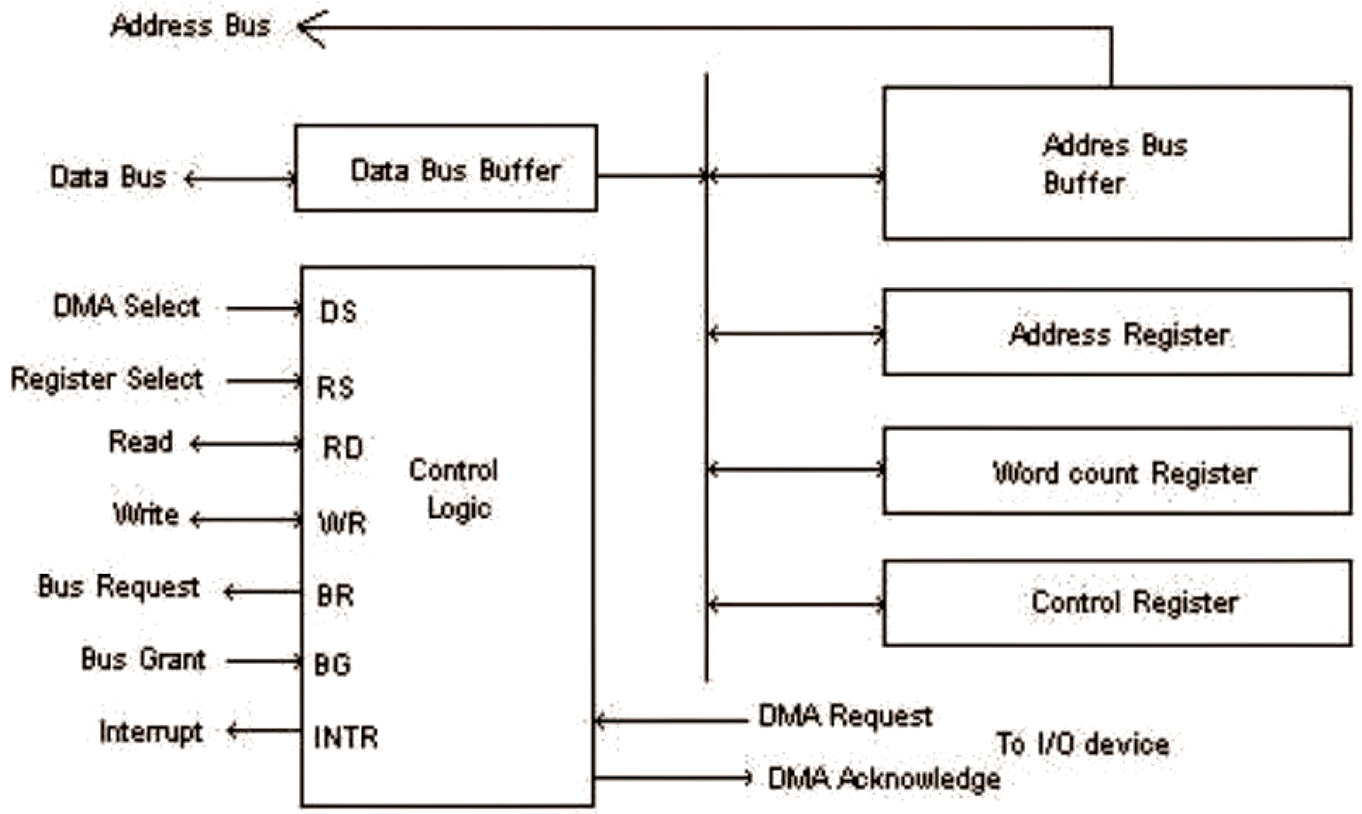
Direct Memory Access (DMA)

- When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways.
- **Burst transfer**
- **Cycle stealing**
- In DMA burst transfer, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses.
- This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred.
- An alternative technique called cycle stealing allows the DMA controller to transfer one data word at a time after which it must return control of the buses to the CPU.
- The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

DMA CONTROLLER

- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device.
- Below Figure shows the block diagram of a typical DMA controller.
- The unit communicates with the CPU via the data bus and control lines.
- The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs.
- The RD (read) and WR (write) inputs are bidirectional.
- **When the BG (bus grant) input is 0**, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.
- **When BG = 1**, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.

DMA CONTROLLER



DMA CONTROLLER

- The DMA controller has three registers: an address register, a word count register, and a control register.

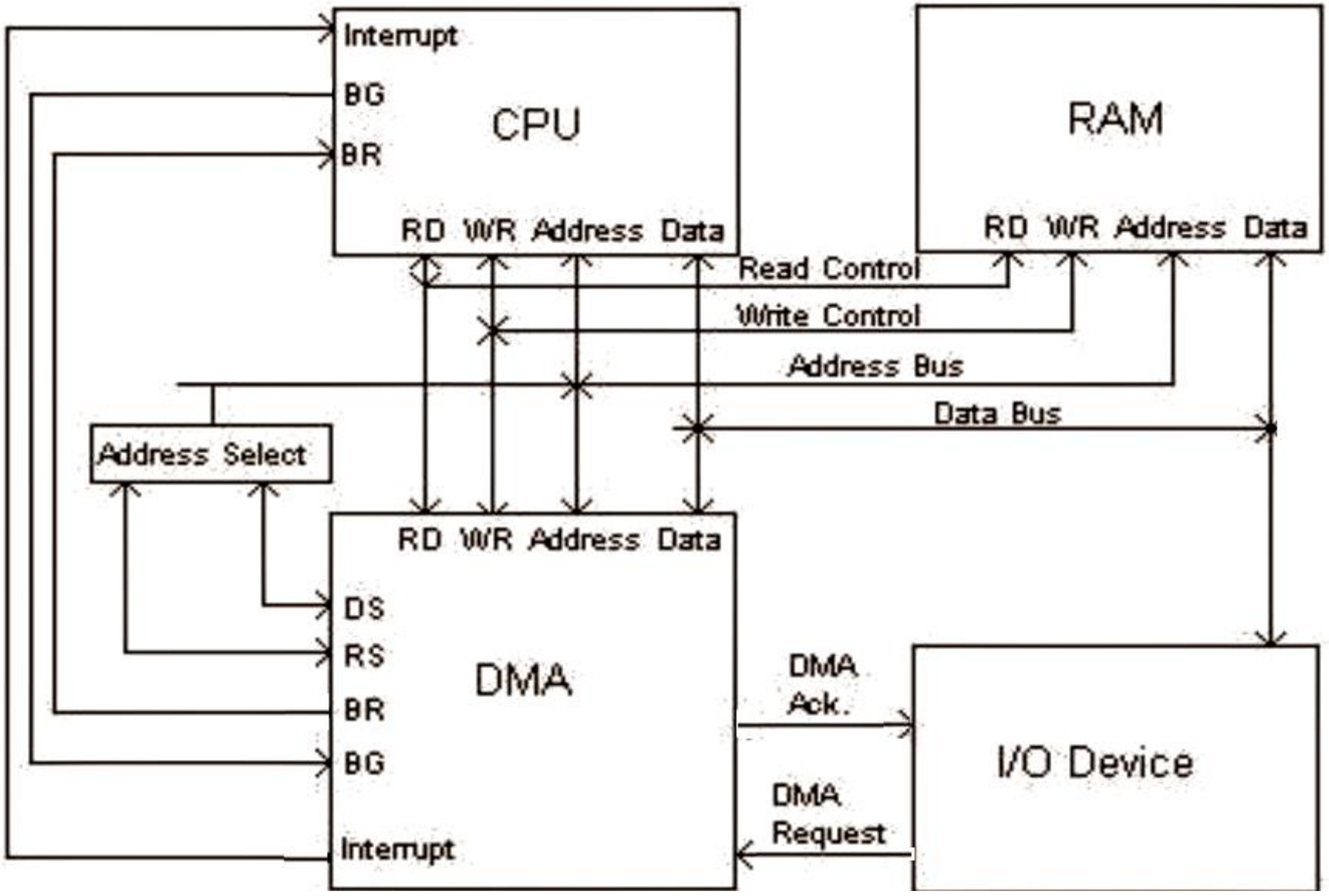
1. **The address register** contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.
2. **The word count register** is incremented after each word that is transferred to memory. The word count register holds the number of words to be transferred. This register is decremented by one after each word transfer and internally tested for zero.
3. **The control register** specifies the mode of transfer. All registers in the DMA appear to the CPU as I/O interface registers. Thus the CPU can read from or write into the DMA registers under program control via the data bus.

DMA TRANSFER

The position of the DMA controller among the other components in a computer system is illustrated in below Fig. The CPU communicates with the DMA through the address and data buses as with any interface unit.

1. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.
2. When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses.
3. The CPU responds with its BG line, informing the DMA that its buses are disabled.
4. The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device. Note that the RD and WR lines in the DMA controller are bidirectional.
5. When the peripheral device receives a DMA acknowledge, it puts a word in the data us (for write) or receives a word from the data bus (for read). Thus the DMA controls the read or write operations and supplies the address for the memory.

DMA TRANSFER



DMA TRANSFER

- For each word that is transferred, the DMA increments its address registers and decrements its word count register. If the word count does not reach zero, the DMA checks the request line coming from the peripheral. A second transfer is then initiated, and the process continues until the entire block is transferred.
- It the word count register reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of an interrupt.
- DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory. It is also useful for updating the display in an interactive terminal.

Privileged and Non-Privileged

- **What are Privileged Instructions?** *The Instructions that can run only in Kernel Mode are called Privileged Instructions.*
- **Privileged Instructions possess the following characteristics** :
- Various examples of Privileged Instructions include:
- I/O instructions and Halt instructions
- Turn off all Interrupts
- Set the Timer
- Context Switching
- Clear the Memory or Remove a process from the Memory
- Modify entries in Device-status table

Privileged and Non-Privileged

- <u>**What are Non-Privileged Instructions?**</u> *The Instructions that can run only in User Mode are called Non-Privileged Instructions.*
- Various examples of Non-Privileged Instructions include:
- Reading the status of Processor
- Reading the System Time
- Generate any Trap Instruction
- Sending the final printout of Printer

Software Interrupts

Given that the operating system prevents unprivileged code from directly accessing system resources, **how do applications gain access to these protected resources?**

- "Hey, over here, I need to read some data from disk!"
- "I could really use some more memory. Please?“
- **CPUs provide a special instruction (syscall on the MIPS) that generates a software (or synthetic) interrupt**.
- Software interrupts provide a mechanism for user code to indicate that it needs help from the kernel.

Software Interrupts

Rest of the interrupt handling path is unchanged. The CPU:

1. **enters privileged mode**,
2. **records state** necessary to process the interrupt,
3. **jumps to a pre-determined memory location** and begins executing instructions

Software Interrupts

To access the kernel system call interface an application:

- arranges arguments to the system call in an agreed-on place where the kernel can find them, typically in registers or on its stack,
- loads a number identifying the system call it wants the kernel to perform into a pre-determined register, and
- executes the syscall instruction.

Software Exceptions

- An **software exception** indicates that code running on the CPU has created a situation that the processor needs help to address.
- Can you think of examples of software exceptions?
- Divide by zero—probably kills the process.
- Attempt to use a privileged instruction—also probably kills the process.
- Attempt to use a virtual address that the CPU does not know how to translate—a common exception handled transparently as part of virtual memory management.

Software Exceptions vs Software Interrupts

- **Interrupts** are **voluntary:**

  Think of the CPU as saying to the kernel: "The /bin/true process would like your assistance."

- **Exceptions** are **non-voluntary:**

  Think of the CPU as saying to the kernel: "I need some help with this /bin/false process. It just tried to divide by zero and I think it needs to be terminated."

Definition of Process

- A program in execution – A process has its own address space consisting of:
  - Text region: – Stores the code that the processor executes
  - Data region: – Stores variables and dynamically allocated memory
  - Stack region: – Stores instructions and local variables for active procedure calls

Process States: Life Cycle of a Process

- A process moves through a series of discrete process states: – Running state
  - The process is executing on a processor – Ready state
  - The process could execute on a processor if one were available – Blocked state
  - The process is waiting for some event to happen before it can proceed
  - The OS maintains a ready list and a blocked list to store references to processes not running

Process Management

- Operating systems provide fundamental services to processes including: –
- Creating processes
- Destroying processes
- Suspending processes
- Resuming processes
- Changing a process’s priority
- Blocking processes
- Waking up processes
- Dispatching processes
- Interprocess communication (IPC)

Process States and State Transitions

- **Process states:** –

1. The act of assigning a processor to the first process on the ready list is called dispatching
2. The OS may use an interval timer to allow a process to run for a specific time interval or quantum
3. Cooperative multitasking lets each process run to completion

- **State Transitions:** –

  At this point, there are four possible state transitions

1) When a process is dispatched, it transitions from ready to running .
2) When the quantum expires, it transitions from running to ready.
3) When a process blocks, it transitions from running to blocked.
4) When the event occurs, it transitions from blocked to ready.

Universal Serial Bus (USB)

- **The USB has been designed to meet several key objectives **

1. Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer ‹
2. Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections ‹
3. Enhance user convenience through a “plug-and-play” mode of operation
4. A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements
5. To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure