Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (MP)

Think Excellence. Live Excellence.

Shri Vaishnav Institute of Information Technology

Department of Information Technology

Lecture Notes

COMPUTER SYSTEM ORGANIZATION

(BTIT205)

Subject Teacher

Er. Gaurav Shrivastava

B.E. (CSE), M.E. (IT)
Asst. Professor (IT Department)
IT- First Year Coordinator
SVITS-SVVV, Indore

Syllabus

Unit-I: Computer Basics and CPU : Von Newman model, various subsystems, CPU, Memory, I/O, System Bus, CPU and Memory registers, Program Counter, Accumulator, Instruction register, Micro operations, Register Transfer Language, Instruction Fetch, decode and execution, data movement and manipulation, Instruction formats and addressing modes of basic computer. 8085 microprocessor organization

Unit-II Control Unit Organization: Hardwired control unit, Micro and nano programmed control unit, Control Memory, Address Sequencing, Micro Instruction formats, Micro program sequencer, Microprogramming, Arithmetic and Logic Unit: Arithmetic Processor, Addition, subtraction, multiplication and division, Floating point and decimal arithmetic and arithmetic units, design of arithmetic unit.

Unit-III Input Output Organization: Modes of data transfer – program controlled, interrupt driven and direct memory access, Interrupt structures, I/O Interface, Asynchronous Data Transfer, I/O processor, 8085 I/O structure, 8085 instruction set and basic programming. Data Transfer – Serial / parallel, synchronous/asynchronous, simplex,/half duplex and full duplex.

Unit-IV Memory organization: Memory Maps, Memory Hierarchy, Cache Memory - Organization and mappings. Associative Memory, Virtual Memory, Memory Management Hardware.

Unit-V Multiprocessors: Pipeline and Vector processing, Instruction and arithmetic pipelines, Vector and array processors, Interconnection structure and inter-processor communication.

References:
- Morris Mano: Computer System Architecture, PHI.
- Gaonkar: Microprocessor Architecture, Programming, Applications with 8085; Penram Int.
- William Stallings: Computer Organization and Architecture, PHI
- Carter; Computer Architecture (Schaum); TMH
- Carl Hamacher: Computer Organization, TMH
- Tanenbaum: Structured Computer Organization, Pearson Education

Unit-I:
Computer Basics and CPU

1. Computer Types

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

List of instructions are called programs & internal storage is called computer memory.

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

2. Functional unit

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

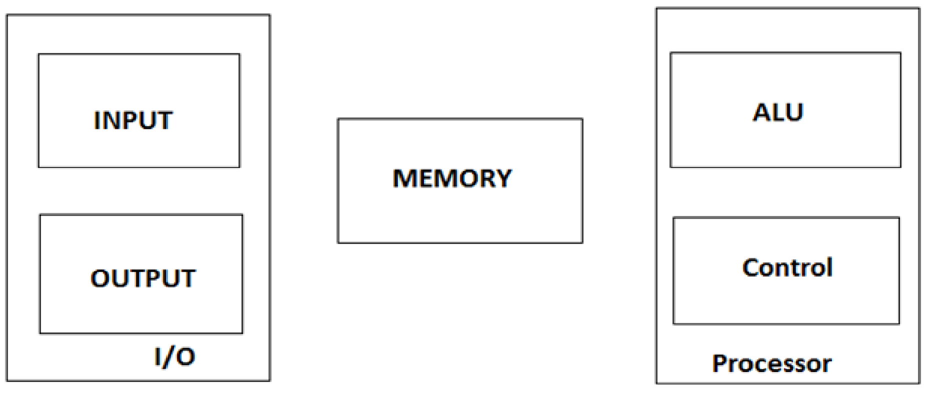


Fig : Functional units of computer

Input device accepts the coded information as source program i.e. high level language. This is either stored in the memory or immediately used by the processor to perform the desired operations. The program stored in the memory determines the processing steps. Basically the computer converts one source program to an object program. I.e. into machine language

Finally the results are sent to the outside world through output device. All of these actions are coordinated by the control unit.

Input unit: -
The source program/high level languages program/coded information/simply data is fed to a computer through input devices keyboard is a most common type. Whenever a key is pressed, one corresponding word or number is translated into its equivalent binary code over a cable & fed either to memory or processor.

Joysticks, trackballs, mouse, scanners etc. are other input devices.

Memory unit: -
Its function into store programs and data. It is basically to two types

I. Primary memory
II. Secondary memory

I. Primary memory: - Is the one exclusively associated with the processor and operates at the electronics speeds programs must be stored in this memory while they are being executed. The memory contains a large number of semiconductors storage cells. Each capable of storing one bit of information. These are processed in a group of fixed site called word.

To provide easy access to a word in memory, a distinct address is associated with each word location. Addresses are numbers that identify memory location.

Number of bits in each word is called word length of the computer. Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the control of processor.

Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random-access memory (RAM).

The time required to access one word in called memory access time. Memory which is only readable by the user and contents of which can't be altered is called read only memory (ROM) it contains operating system.

Caches are the small fast RAM units, which are coupled with the processor and are aften contained on the same IC chip to achieve high performance. Although primary storage is essential it tends to be expensive.

II. Secondary memory: - Is used where large amounts of data & programs have to be stored, particularly information that is accessed infrequently.

Examples: - Magnetic disks & tapes, optical disks (ie CD-ROM's), floppies etc.,

Arithmetic logic unit (ALU):-

Most of the computer operators are executed in ALU of the processor like addition, subtraction, division, multiplication, etc. the operands are brought into the ALU from memory and stored in high speed storage elements called register. Then according to the instructions the operation is performed in the required sequence.

The control and the ALU are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as key boards, displays, magnetic and optical disks, sensors and other mechanical controllers.

Output unit:-

These actually are the counterparts of input unit. Its basic function is to send the processed results to the outside world.

Examples: - Printer, speakers, monitor etc.

Control unit:-

It effectively is the nerve centre that sends signals to other units and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.

3. The von Neumann Computer Model

- Von Neumann computer systems contain three main building blocks:
    - the central processing unit (CPU),
    - memory,
    - Input/output devices (I/O).
- These three components are connected together using the *system bus*.
- The most prominent items within the CPU are the registers: they can be manipulated directly by a computer program.

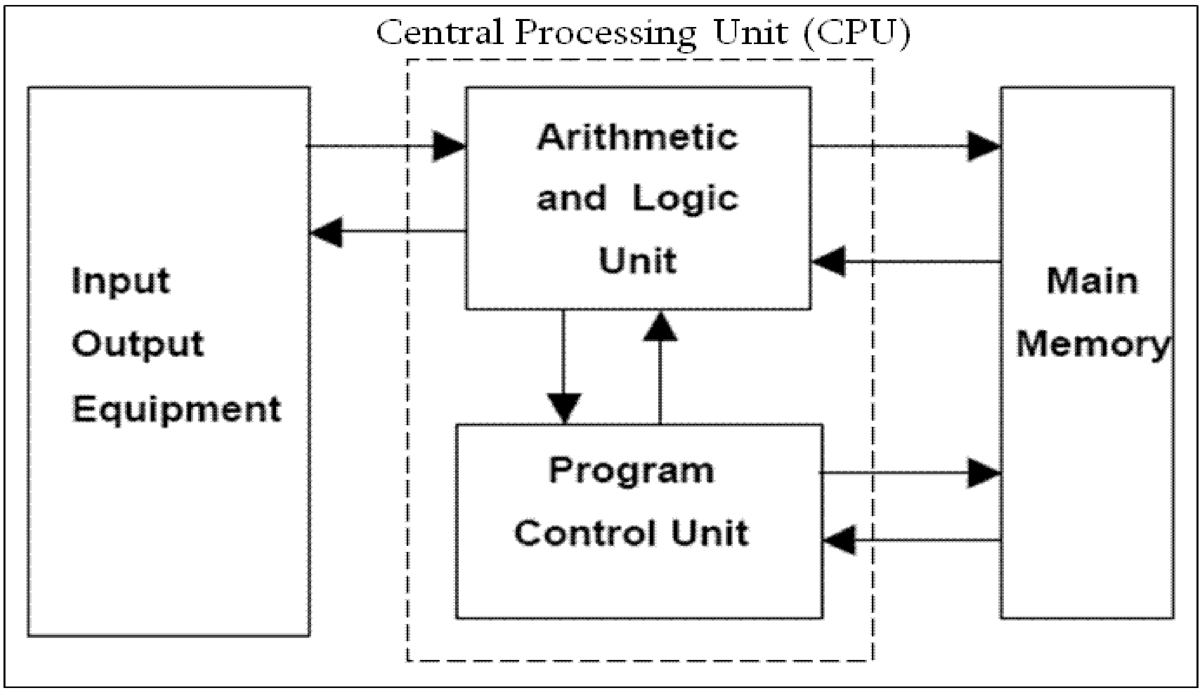
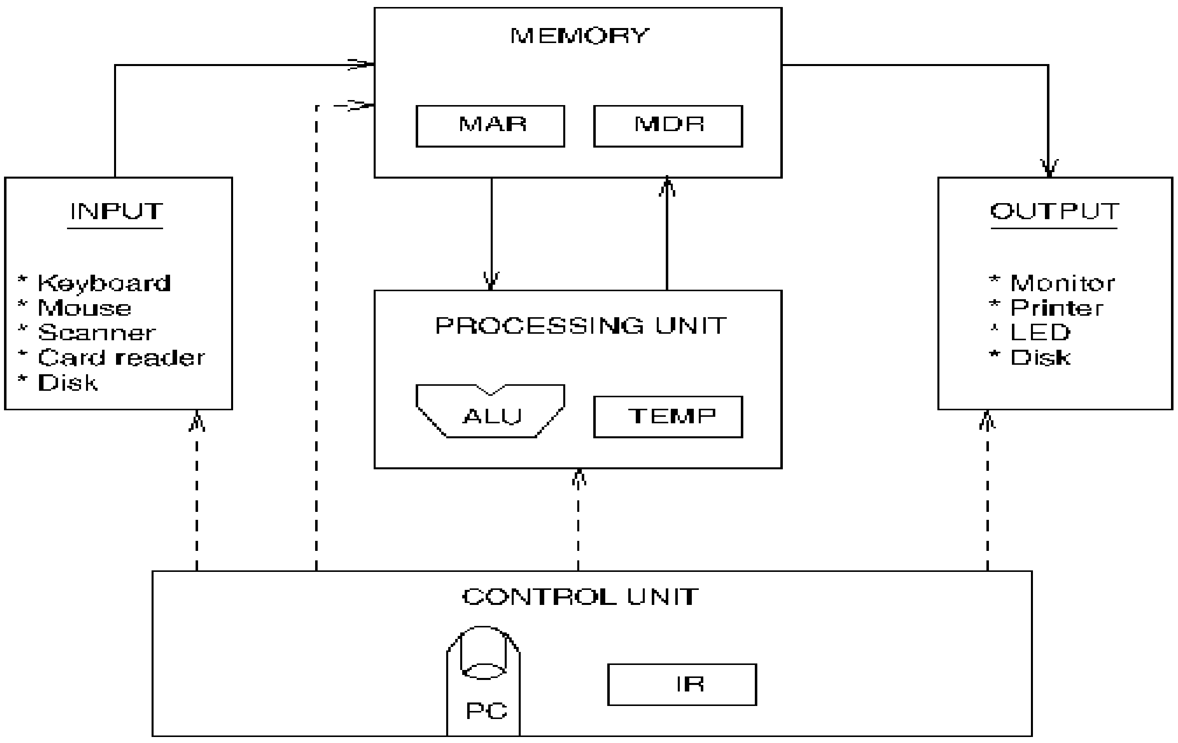


Figure : General structure of Von Neumann Architecture

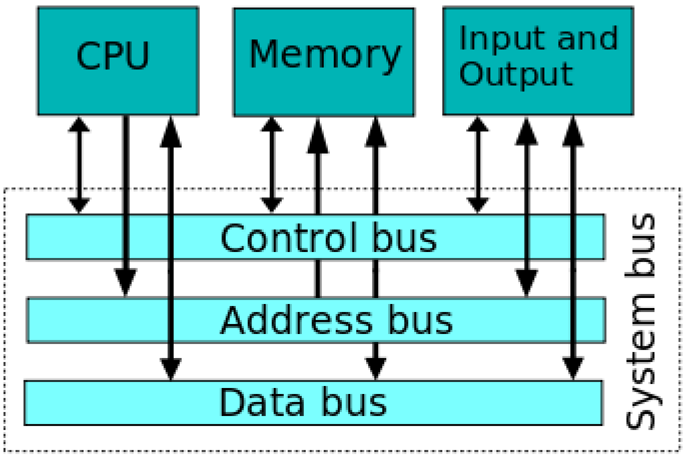
Components of the Von Neumann Model

1. **Memory**: Storage of information (data/program)
2. **Processing Unit**: Computation/Processing of Information
3. **Input**: Means of getting information into the computer. e.g. keyboard, mouse
4. **Output**: Means of getting information out of the computer. e.g. printer, monitor
5. **Control Unit**: Makes sure that all the other parts perform their tasks correctly and at the correct time.



4. System Bus

The system bus connects the CPU with the main memory and, in some systems, with the level 2 (L2) cache. Other buses, such as the IO buses, branch off from the system bus to provide a communication channel between the CPU and the other peripherals.



The system bus combines the functions of the three main buses, which are as follows:

- The control bus carries the control, timing and coordination signals to manage the various functions across the system.
- The address bus is used to specify memory locations for the data being transferred.
- The data bus, which is a bidirectional path, carries the actual data between the processor, the memory and the peripherals.

5. CPU and Memory Registers,

CPU Register:
A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor.

A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify that the contents of two defined registers be added together and then placed in a specified register. A register must be large enough to hold an instruction - for example, in a 64-bit computer; a register must be 64 bits in length. In some computer designs, there are smaller registers - for example, half-registers - for shorter instructions. Depending on the processor design and language rules, registers may be numbered or have arbitrary names.

Memory Registers:

Register are used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU, there are various types of Registers those are used for various purpose. Among of the some Mostly used Registers named as AC or Accumulator, Data Register or DR, the AR or Address Register, program counter (PC), Memory Data Register (MDR) ,Index register, Memory *Buffer Register*.

These Registers are used for performing the various Operations. While we are working on the System then these Registers are used by the CPU for Performing the Operations. When We Gives Some Input to the System then the Input will be Stored into the Registers and When the System will gives us the Results after Processing then the Result will also be from the Registers.

So that they are used by the CPU for Processing the Data which is given by the User. Registers Perform:-

1) Fetch: The Fetch Operation is used for taking the instructions those are given by the user and the Instructions those are stored into the Main Memory will be fetch by using Registers.

2) Decode: The Decode Operation is used for interpreting the Instructions means the Instructions are decoded means the CPU will find out which Operation is to be performed on the Instructions.

3) Execute: The Execute Operation is performed by the CPU. And Results those are produced by the CPU are then Stored into the Memory and after that they are displayed on the user Screen.

Types of Registers are as Followings

Memory Address Register (MAR)

This register holds the memory addresses of data and instructions. This register is used to access data and instructions from memory during the execution phase of an instruction. Suppose CPU wants to store some data in the memory or to read the data from the memory. It places the address of the-required memory location in the MAR.

Program Counter

The program counter (PC), commonly called the instruction pointer (IP) in Intel x86 microprocessors, and sometimes called the instruction address register, or just part of the instruction sequencer in some computers, is a processor register

It is a 16 bit special function register in the 8085 microprocessor. It keeps track of the next memory address of the instruction that is to be executed once the execution of the current instruction is completed. In other words, it holds the address of the memory location of the next instruction when the current instruction is executed by the microprocessor.

Accumulator Register

This Register is used for storing the Results those are produced by the System. When the CPU will generate Some Results after the Processing then all the Results will be Stored into the AC Register.

Memory Data Register (MDR)

MDR is the register of a computer's control unit that contains the data to be stored in the computer storage (e.g. RAM), or the data after a fetch from the computer storage. It acts like a buffer and holds anything that is copied from the memory ready for the processor to use it. MDR hold the information before it goes to the decoder.

MDR which contains the data to be written into or readout of the addressed location. For example, to retrieve the contents of cell 123, we would load the value 123 (in binary, of course) into the MAR and perform a fetch operation. When the operation is done, a copy of the contents of cell 123 would be in the MDR. To store the value 98 into cell 4, we load a 4 into the MAR and a 98 into the MDR and perform a store. When the operation is completed the contents of cell 4 will have been set to 98, by discarding whatever was there previously.

The MDR is a two-way register. When data is fetched from memory and placed into the MDR, it is written to in one direction. When there is a write instruction, the data to be written is placed into the MDR from another CPU register, which then puts the data into memory.

The Memory Data Register is half of a minimal interface between a micro program and computer storage, the other half is a memory address register.

- ***Index Register***

A hardware element which holds a number that can be added to (or, in some cases, subtracted from) the address portion of a computer instruction to form an effective address. Also known as base register. An index register in a computer's CPU is a processor register used for modifying operand addresses during the run of a program.

- ***Memory Buffer Register***

MBR stand for *Memory Buffer Register*. This register holds the contents of data or instruction read from, or written in memory. It means that this register is used to store data/instruction coming from the memory or going to the memory.

- ***Data Register***

A register used in microcomputers to temporarily store data being transmitted to or from a peripheral device.

6. Microoperations

In computer central processing units, micro-operations (also known as a micro-ops) are detailed low-level instructions used in some designs to implement complex machine instructions (sometimes termed macro-instructions in this context).

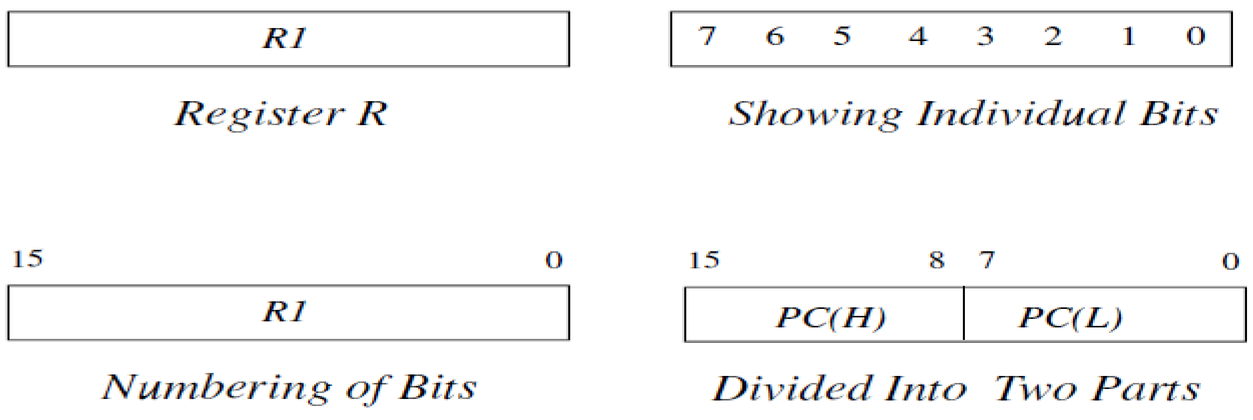
Usually, micro-operations perform basic operations on data stored in one or more registers, including transferring data between registers or between registers and external buses of the central processing unit (CPU), and performing arithmetic or logical operations on registers. In a typical fetch-decode-execute cycle, each step of a macro-instruction is decomposed during its execution so the CPU determines and steps through a series of micro-operations. The execution of micro-operations is performed under control of the CPU's control unit, which decides on their execution while performing various optimizations such as reordering, fusion and caching.

7. Register Transfer Language

• A register transfer language is a notation used to describe the microoperation transfers between registers.
• It is a system for expressing in symbolic form the microoperation sequences among register that are used to implement machine-language instructions.

Registers are denoted by capital letters and are sometimes followed by numerals, e.g.,
– MAR – Memory Address Register (holds addresses for the memory unit)
– PC – Program Counter (holds the next instruction’s address)
– IR – Instruction Register (holds the instruction being executed)
– R1 – Register 1 (a CPU register)
• We can indicate individual bits by placing them in parentheses, e.g., PC (8-15), R2 (5), etc.

Block Diagrams of Registers



Register Transfer Language Instructions

- Register Transfer $R2 \leftarrow R1$
- Simultaneous Transfer

  $R2 \leftarrow R1, R1 \leftarrow R2$
- Conditional Transfer (***Control Function***)

  $P: R2 \leftarrow R1$

  or

  If ($P = 1$) Then $R2 \leftarrow R1$
- Conditional, Simultaneous Transfer

  $T: R2 \leftarrow R1, R1 \leftarrow R2$

Basic Symbols for Register Transfer

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow $\rightarrow$	Denotes Transfer of information	$R2 \leftarrow R1$
Comma ,	Separates 2 microoperations	$R2 \leftarrow R1, R1 \leftarrow R1$

8. Instruction cycle:

An instruction cycle (sometimes called a fetch–decode–execute cycle) is the basic operational process of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction dictates, and carries out those actions. This cycle is repeated continuously by a computer's central processing unit (CPU), from boot-up to when the computer is shut down.

In simpler CPUs the instruction cycle is executed sequentially, each instruction being processed before the next one is started. In most modern CPUs the instruction cycles are instead executed concurrently, and often in parallel, through an instruction pipeline: the next instruction starts being processed before the previous instruction has finished, which is possible because the cycle is broken up into separate steps.

Components

Program counter (PC)

An incrementing counter that keeps track of the memory address of the instruction that is to be executed next or in other words, holds the address of the instruction to be executed next.

Memory address register (MAR)

Holds the address of a block of memory for reading from or writing to.

Memory data register (MDR)

A two-way register that holds data fetched from memory (and ready for the CPU to process) or data waiting to be stored in memory. (This is also known as the memory buffer register (MBR).)

Instruction register (IR)

A temporary holding ground for the instruction that has just been fetched from memory.

Control unit (CU)

Decodes the program instruction in the IR, selecting machine resources, such as a data source register and a particular arithmetic operation, and coordinates activation of those resources.

Arithmetic logic unit (ALU)

Performs mathematical and logical operations.

Accumulator: A**ccumulator** is a register in which intermediate arithmetic and logic results are stored.

Steps

Each computer's CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

1. **Fetch the instruction**: The next instruction is fetched from the memory address that is currently stored in the program counter (PC), and stored in the instruction register (IR). At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.
2. **Decode the instruction**: During this cycle the encoded instruction present in the IR (instruction register) is interpreted by the decoder.
3. **Read the effective address**: In case of a memory instruction (direct or indirect) the execution phase will be in the next clock pulse. If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers (Clock Pulse: T_3). If the instruction is direct, nothing is done at this clock pulse. If this is an I/O instruction or a Register instruction, the operation is performed (executed) at clock Pulse.
4. **Execute the instruction**: The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU. The result generated by the operation is stored in the main memory, or sent to an output device. Based on the condition of any feedback from the ALU, Program Counter may be updated to a different address from which the next instruction will be fetched.

The cycle is then repeated.

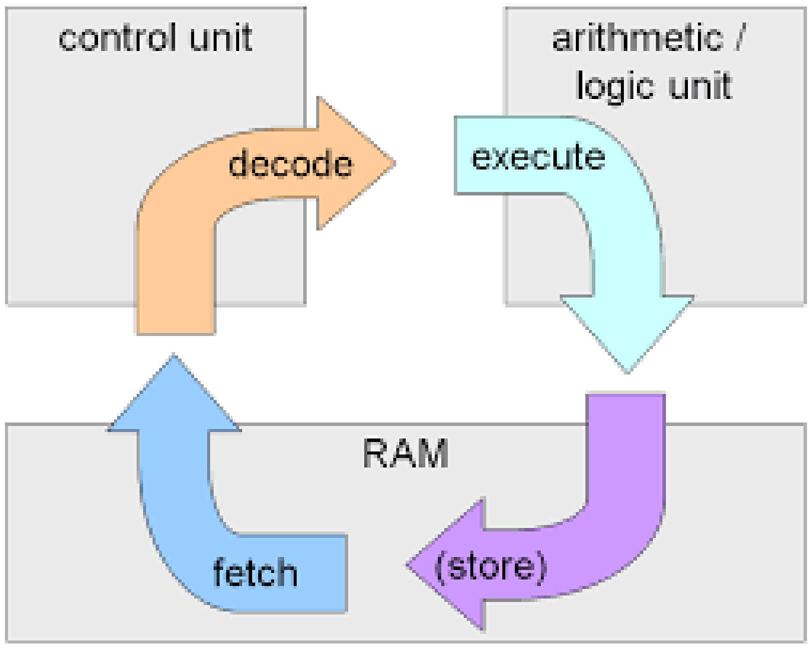


Fig: CPU Cycle

9. Internal Communication:-

CPU of the computer system communicates with the memory and the I/O devices in order to transfer data between them. However, the method of communication of the CPU with memory and I/O devices is different. The CPU may communicate with the memory either directly or through the cache memory. However, the communication between the CPU and I/O devices is usually implemented with the help of interfaces. Therefore, the internal communication of a processor in the computer system can be divided into two major categories:

- Processor to memory communication
- Processor to I/O devices communication

1.5.1 Processor to Memory Communication

The direct communication between the processor and memory of the computer system is implemented with the help of two registers, Memory Address Register (MAR) and Memory Buffer Register (MBR). Figure 1.6 shows the communication between the processor and the memory of the computer system.

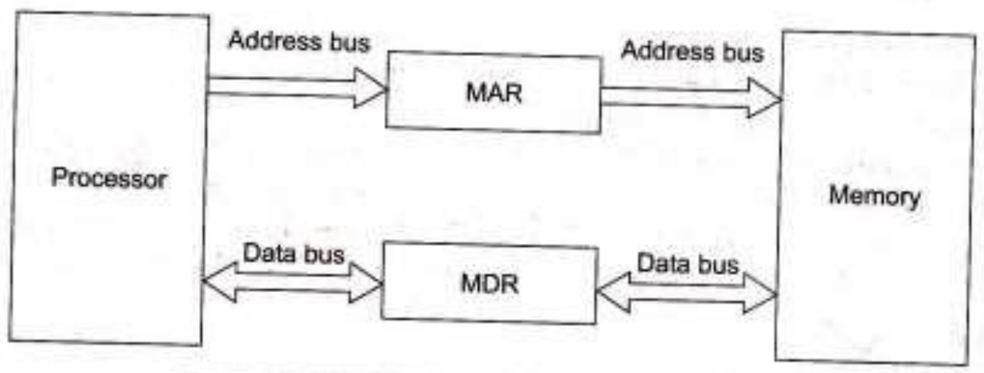


Fig. 1.6 ⇔ *Processor to memory communication*

Memory Read Operation :-

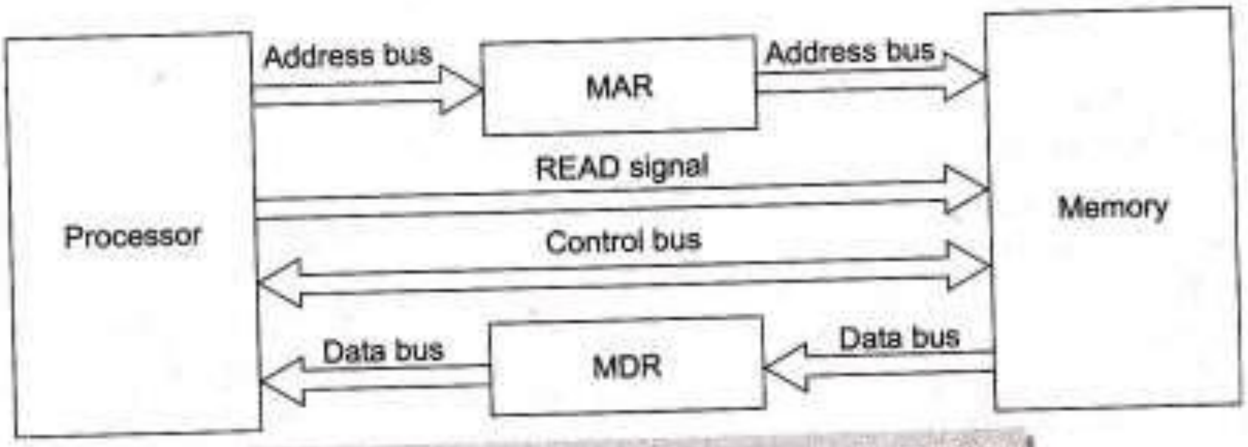


Fig. 1.7 ⇔ *Illustrating the memory read operation*

1. First, the processor loads the address of the memory location from where data is to be read into the MAR register, using the address bus.
2. After loading the address of the memory location, the processor issues the READ control signal through the control bus. The control bus is used to carry the commands issued by the processor, and the status signals are generated by the various devices in response to these commands.
3. After receiving the READ control signal, the memory loads the data into the MDR register from the location specified in the MAR register, using the data bus.
4. Finally, the data is transferred to the processor.

Memory Write Operation:-

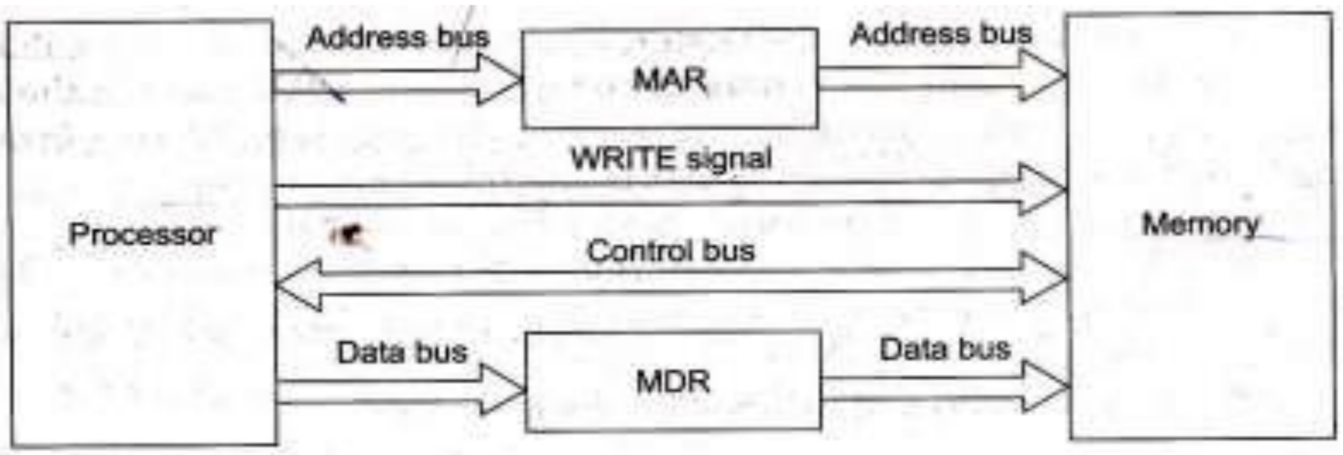


Fig. 1.8 ⇔ *Illustrating memory write operation*

The processor performs the following steps for writing the data at the desired memory location in the computer system:

1. First, the processor loads the address of the memory location where data is to be written in the MAR register, using the address bus.
2. After loading the address of the memory location, the processor loads the desired data in the MDR register, using the data bus.
3. After this, the processor issues the WRITE control signal to the memory, using the control bus.
4. Finally, the memory stores the data loaded in the MDR register at the desired memory location.

1.5.2 Processor to I/O Devices Communication

The communication between I/O devices and processor of the computer system is implemented using an *interface unit*. In a computer system, data is transferred from an input device to the processor and from the processor to an output device. Each input and output device in the computer system is provided with a controller, called *device controller*. The device controller is used to manage the working of various peripheral devices. The processor actually communicates with the device controllers of the various I/O devices for performing the I/O operations.

Figure 1.9 illustrates how the communication between the processor and the I/O devices of the computer system is implemented. The interface unit acts as an intermediary between the processor and the device controllers of various peripheral devices in the computer system. The basic function of the interface unit is to accept the control commands from the processor and interpret the commands so that they can be easily understood by the device controllers for carrying out necessary operations. Therefore, we can say that the interface unit is responsible for controlling the input and output operations between the processor and the I/O devices. The interface unit contains data register and status register. The data register is used to store the data to be transferred, either to the processor or to an output device. The status register is

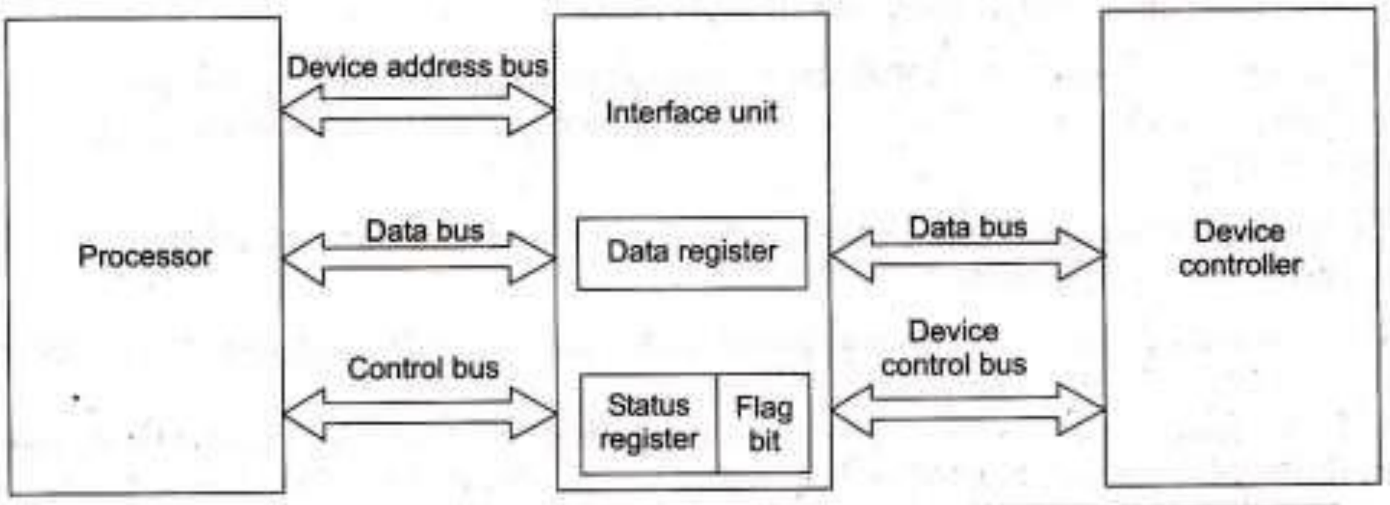


Fig. 1.9 ⇔ *Illustrating the communication process between the processor and I/O devices*

used to indicate the status of the data register, i.e., whether it is currently holding the data or not. If the data register is holding the data to be transferred, the flag bit of the status register is set to one. The processor to I/O devices communication involves two important operations, i.e., I/O read and I/O write. The I/O read operation helps the processor to read the data from an input device.

Figure 1.10 illustrates how the data is transferred from an input device to the processor of the computer system. The steps performed while transferring the data from an input device to the processor are:

1. The data to be transferred is placed on the data bus by the input device, which transfers single byte of data at a time.
2. The input device then issues the data valid signal through the device control bus to the data register, indicating that the data is available on the data bus.
3. When the data register of the interface unit accepts the data, it issues a data accepted signal through the device control bus as an acknowledgement to the input device, indicating that the data has been received. The input device then disables the data valid signal.
4. As the data register now holds the data, the F or the flag bit of the status register is set to 1.
5. The processor now issues an I/O read signal to the data register in the interface unit.
6. The data register then places the data on the data bus connected to the processor of the computer system. After receiving the data, the processor sends an appropriate acknowledgement signal to the input device, indicating that the data has been received.

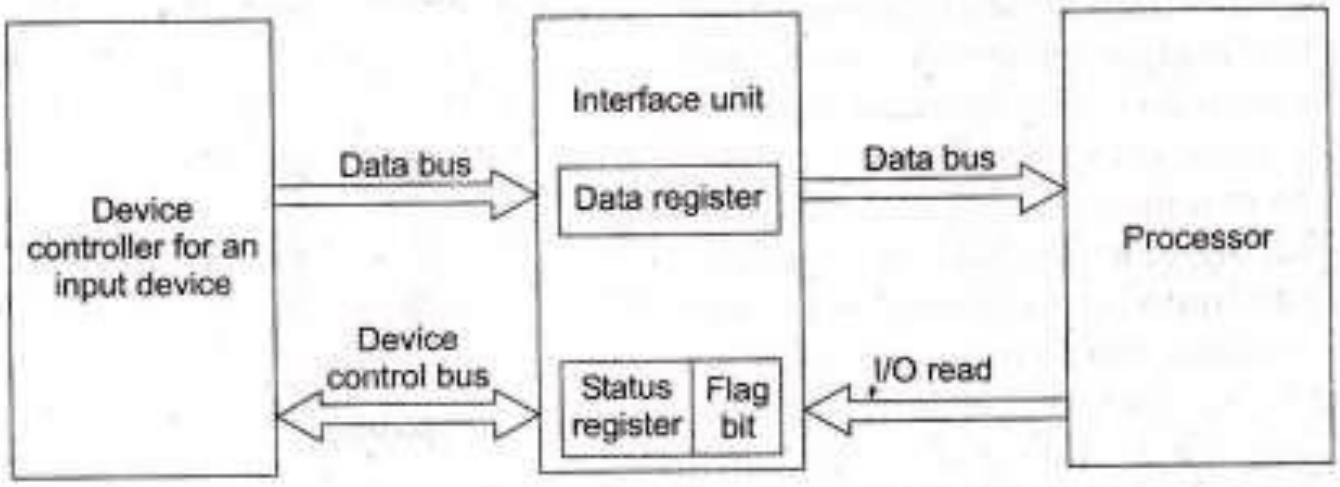


Fig. 1.10 ⇔ *Illustrating the I/O read operation*

The I/O write operation helps the processor to write the data to an output device. Figure 1.11 illustrates how the data is transferred from the processor to an output device. The steps performed while transferring the data from the processor to the output device are:

1. The processor places the data that needs to be transferred on the data bus connected to the data register of the interface unit.
2. The CPU also places the address of the output device on the device address bus.
3. After placing the address and data on the appropriate buses, CPU issues the I/O write signal, which writes the data on the data register. The flag bit in the interface unit is set to 1, indicating that the data register now holds the data.
4. The data register of the interface unit issues a data accepted signal through the control bus to the processor, indicating that the data has been received.
5. The interface unit then places the data stored in the data register on to the data bus connected to the device controller of the output device.
6. The output device then receives the data and sends an acknowledgement signal to the processor of the computer system through the interface unit, indicating that the desired data has been received.

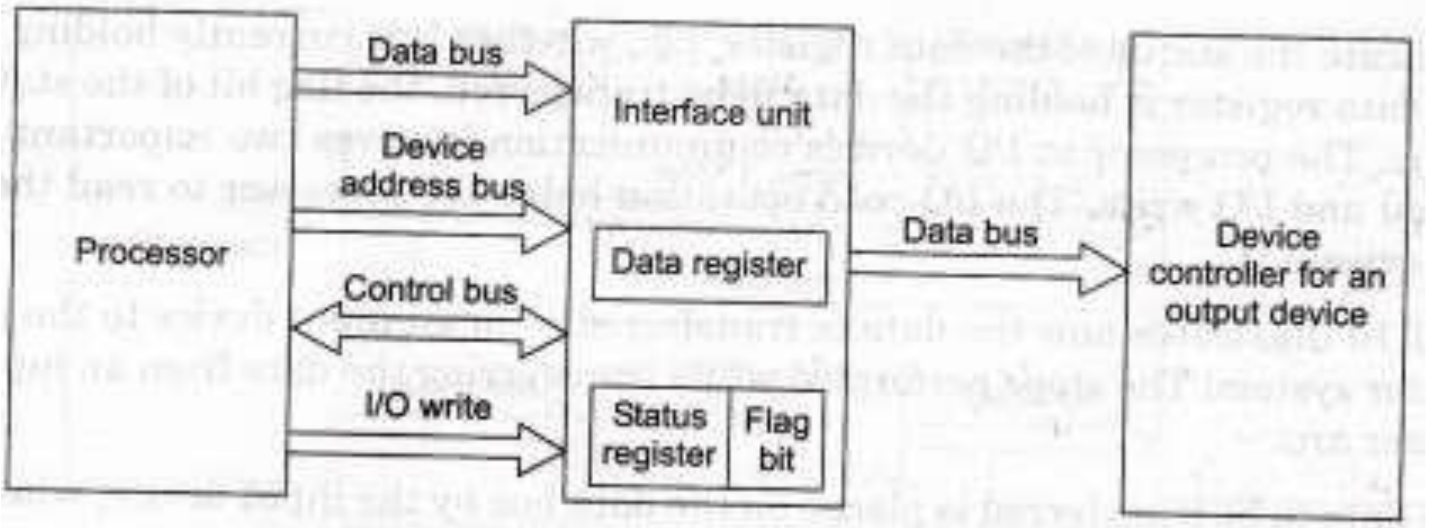
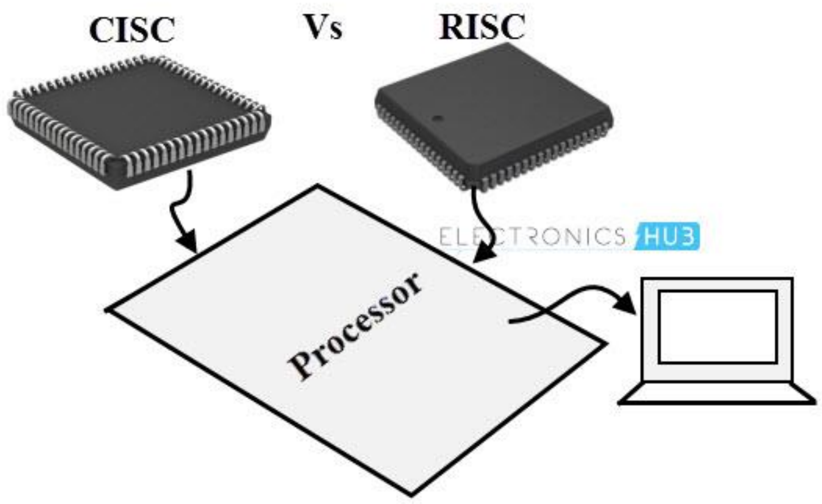


Fig. 1.11 ⇔ *Illustrating I/O write operation*

10. Instruction Set:-

The instruction set, also called instruction set architecture (ISA), an instruction set is a group of commands for a CPU in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations.
All CPUs have instruction sets that enable commands to the processor directing the CPU to switch the relevant transistors. Some instructions are simple read, write and move commands that direct data to different hardware.

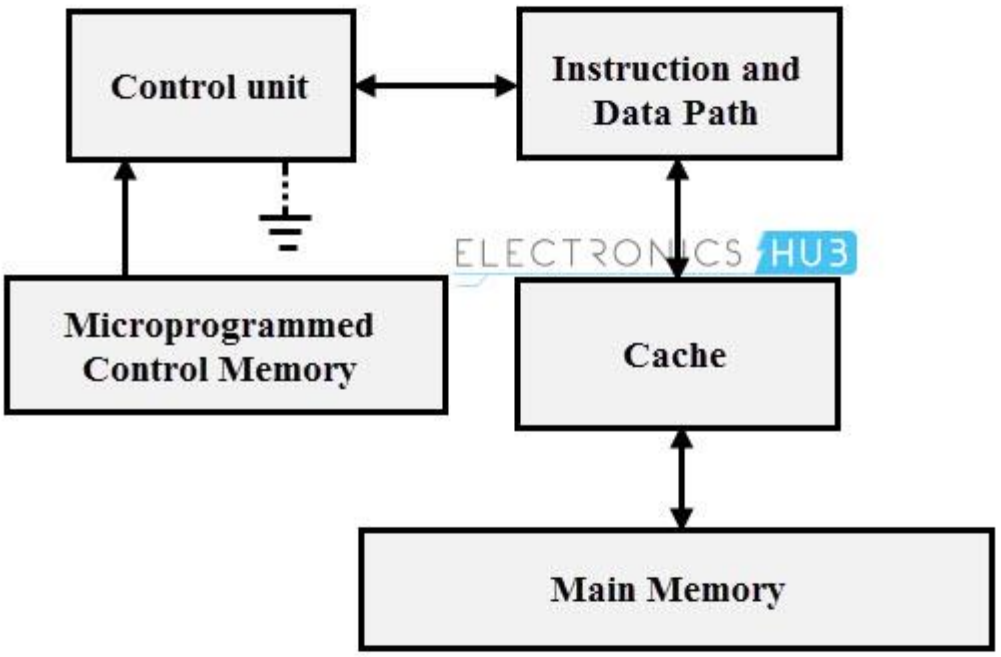
The two major instruction sets architectures are
1) CISC (Complex Instruction Set Computing)
2) RISC (Reduced Instruction Set Computing)



10.1 CISC Architecture

In the early days machines were programmed in assembly language and the memory access is also slow. To calculate complex arithmetic operations, compilers have to create long sequence of machine code.

This made the designers to build an architecture , which access memory less frequently and reduce burden to compiler. Thus this lead to very power full but complex instruction set.



CISC architectures directly use the memory, instead of a register file. The above figure shows the architecture of CISC with micro programmed control and cache memory.
This architecture uses cache memory for holding both data and instructions. Thus, they share the same path for both instructions and data.
CISC has instructions with variable length format. Thus, the number of clock cycles required to execute the instructions may be varied.
Instructions in CISC are executed by micro program which has sequence of microinstructions.

Advantages of CISC Architecture

- Microprogramming is easy to implement and much less expensive than hard wiring a control unit.
- It is easy to add new commands into the chip without changing the structure of the instruction set as the architecture uses general-purpose hardware to carry out commands.
- This architecture makes the efficient use of main memory since the complexity (or more capability) of instruction allows to use less number of instructions to achieve a given task.
- The compiler need not be very complicated, as the micro program instruction sets can be written to match the constructs of high level languages.

Disadvantages of CISC Architecture

- A new or succeeding versions of CISC processors consists early generation processors in their subsets (succeeding version). Therefore, chip hardware and instruction set became complex with each generation of the processor.

- The overall performance of the machine is reduced because of slower clock speed.
- The complexity of hardware and on-chip software included in CISC design to perform many functions.

Examples of CISC processor

1. IBM 370/168
2. Intel 80486
3. VAX 11/780

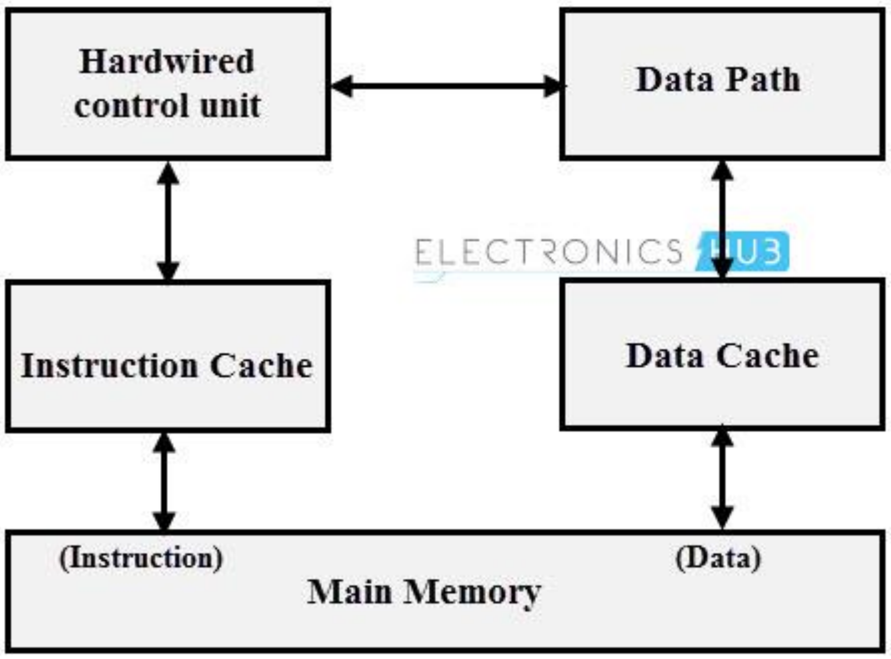
10.2 RISC (Reduced Instruction Set Computer) Architecture

Although CISC reduces usage of memory and compiler, it requires more complex hardware to implement the complex instructions.

In RISC architecture, the instruction set of processor is simplified to reduce the execution time. It uses small and highly optimized set of instructions which are generally register to register operations.

The speed of the execution is increased by using smaller number of instructions .This uses pipeline technique for execution of any instruction.

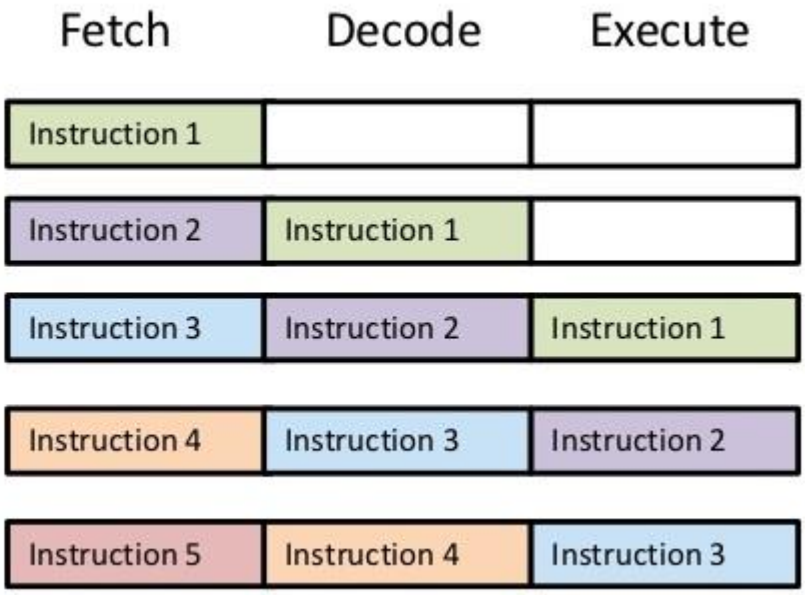
The figure shown below is the architecture of RISC processor, which uses separate instruction and data caches and their access paths also different. There is one instruction per machine cycle in RISC processor.



The pipelining technique allows the processor to work on different steps of instruction like fetch, decode and execute instructions at the same time. Below is image showing execution of instructions in pipelining technique.

Generally, execution of second instruction is started, only after the completion of the first instruction. But in pipeline technique, each instruction is executed in number of stages simultaneously.

When the first stage of first instruction is completed, next instruction is enters into the fist stage. This process continuous until all the instructions are executed.



Advantages of RISC Architecture

- The performance of RISC processors is often two to four times than that of CISC processors because of simplified instruction set.
- This architecture uses less chip space due to reduced instruction set. This makes to place extra functions like floating point arithmetic units or memory management units on the same chip.
- The per-chip cost is reduced by this architecture that uses smaller chips consisting of more components on a single silicon wafer.
- RISC processors can be designed more quickly than CISC processors due to its simple architecture.
- The execution of instructions in RISC processors is high due to the use of many registers for holding and passing the instructions as compared to CISC processors.

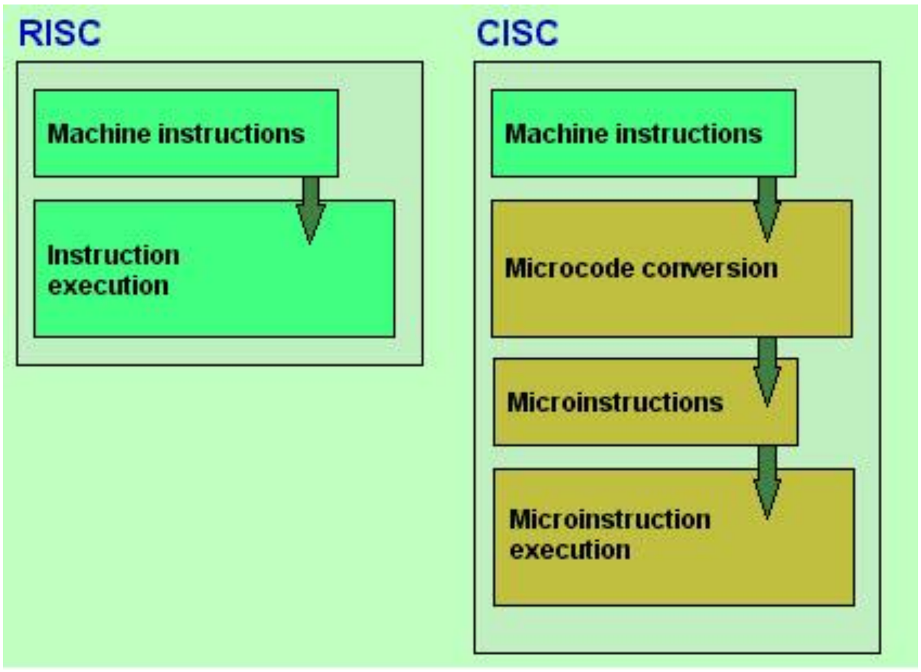
Disadvantages of RISC Architecture

- The performance of a RISC processor depends on the code that is being executed. The processor spends much time waiting for first instruction result before it proceeds with next subsequent instruction, when a compiler makes a poor job of scheduling instruction execution.

- RISC processors require very fast memory systems to feed various instructions. Typically, a large memory cache is provided on the chip in most RISC based systems.

Examples of RISC processors

This architecture includes alpha, AVR, ARM, PIC, PA-RISC, and power architecture.



Understand RISC & CISC architecture with example

Let we take an example of multiplying two numbers

A = A * B; <<<======this is C statement

The CISC Approach: - The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding & executing a series of operations, this is where our CISC architecture introduced.

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

1. Loads the two values into separate registers
2. Multiplies the operands in the execution unit
3. And finally third, stores the product in the appropriate register.

Thus, the entire task of multiplying two numbers can be completed with one instruction:

MULT A, B <<<======this is assembly statement

MULT is what is known as a “complex instruction.” It operates directly on the computer’s memory banks and does not require the programmer to explicitly call any loading or storing functions.

The RISC Approach: - RISC processors only use simple instructions that can be executed within one clock cycle. Thus, the “MULT” command described above could be divided into three separate commands:

1. “LOAD” which moves data from the memory bank to a register
2. “PROD” which finds the product of two operands located within the registers
3. “STORE” which moves data from a register to the memory banks.

In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

```
LOAD R1, A        <<<======this is assembly statement
LOAD R2,B         <<<======this is assembly statement
PROD A, B         <<<======this is assembly statement
STORE R3, A       <<<======this is assembly statement
```

At first, this may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement into code of this form.

Which one is better?

We cannot differentiate RISC and CISC technology because both are suitable at its specific application. What counts is how fast a chip can execute the instructions it is given and how well it runs existing software. Today, both RISC and CISC manufacturers are doing everything to get an edge on the competition.

Comparison between RISC and CISC:

	RISC	**CISC**
Acronym	It stands for ‘Reduced Instruction Set Computer’.	It stands for ‘Complex Instruction Set Computer’.
Definition	The RISC processors have a smaller set of instructions with few addressing nodes.	The CISC processors have a larger set of instructions with many addressing nodes.
Memory unit	It has no memory unit and uses a separate hardware to implement instructions.	It has a memory unit to implement complex instructions.

Program	It has a hard-wired unit of programming.	It has a micro-programming unit.
Design	It is a complex complier design.	It is an easy complier design.
Calculations	The calculations are faster and precise.	The calculations are slow and precise.
Decoding	Decoding of instructions is simple.	Decoding of instructions is complex.
Time	Execution time is very less.	Execution time is very high.
External memory	It does not require external memory for calculations.	It requires external memory for calculations.
Pipelining	Pipelining does function correctly.	Pipelining does not function correctly.
Stalling	Stalling is mostly reduced in processors.	The processors often stall.
Code expansion	Code expansion can be a problem.	Code expansion is not a problem.
Disc space	The space is saved.	The space is wasted.
Applications	Used in high end applications such as video processing, telecommunications and image processing.	Used in low end applications such as security systems, home automations, etc.

<u>11. Data-transfer instructions</u>

Following is the table showing the list of Data-transfer instructions with their meanings.

Opcode	Operand	Meaning	Explanation
MOV	Rd, Sc M, Sc Dt, M	Copy from the source (Sc) to the destination(Dt)	This instruction copies the contents of the source register into the destination register without any alteration. Example − MOV K, L
MVI	Rd, data M, data	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. Example − MVI K, 55L
LDA	16-bit address	Load the accumulator	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. Example − LDA 2034K
LDAX	B/D Reg. pair	Load the accumulator indirect	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. Example − LDAX K
LXI	Reg. pair, 16-	Load the register pair	The instruction loads 16-bit data in the register

	bit data	immediate	pair designated in the register or the memory. Example − LXI K, 3225L
LHLD	16-bit address	Load H and L registers direct	The instruction copies the contents of the memory location pointed out by the address into register L and copies the contents of the next memory location into register H. Example − LHLD 3225K
STA	16-bit address	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example − STA 325K
STAX	16-bit address	Store the accumulator indirect	The contents of the accumulator are copied into the memory location specified by the contents of the operand. Example − STAX K
SHLD	16-bit address	Store H and L registers direct	The contents of register L are stored in the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example − SHLD 3225K
XCHG	None	Exchange H and L with D and E	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example − XCHG
SPHL	None	Copy H and L registers to the stack pointer	The instruction loads the contents of the H and L registers into the stack pointer register. The contents of the H register provide the high-order address and the contents of the L register provide the low-order address. Example − SPHL
XTHL	None	Exchange H and L with top of stack	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1). Example − XTHL
PUSH	Reg. pair	Push the register pair onto the stack	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example − PUSH K
POP	Reg. pair	Pop off stack to the	The contents of the memory location pointed out

		register pair	by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example − POPK
OUT	8-bit port address	Output the data from the accumulator to a port with 8bit address	The contents of the accumulator are copied into the I/O port specified by the operand. Example − OUT K9L
IN	8-bit port address	Input data to accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example − IN5KL

12. Instruction Format:-

The most common fields found in instruction format are:-
(1) An operation code field that specified the operation to be performed
(2) An address field that designates a memory address or a processor registers.
(3) A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address field in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organization.

(1) Single Accumulator organization ADD X AC ® AC + M [×]
(2) General Register Organization ADD R1, R2, R3 R ® R2 + R3
(3) Stack Organization PUSH X

Three address Instruction
Computer with three addresses instruction format can use each address field to specify either processor register are memory operand.
ADD R1, A, B A1 ® M [A] + M [B]
ADD R2, C, D R2 ® M [C] + M [B] X = (A + B) * (C + A)
MUL X, R1, R2 M [X] R1 * R2

The advantage of the three address formats is that it results in short program when evaluating arithmetic expression. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two Address Instructions

Most common in commercial computers. Each address field can specify either a processes register on a memory word.

```
MOV     R1, A       R1 ® M [A]
ADD     R1, B       R1 ® R1 + M [B]
MOV     R2, C       R2 ® M [C]          X = (A + B) * ( C + D)
ADD     R2, D       R2 ® R2 + M [D]
MUL     R1, R2      R1 ® R1 * R2
MOV     X1 R1       M [X] ® R1
```

One Address instruction

It used an implied accumulator (AC) register for all data manipulation. For multiplication/division, there is a need for a second register.

```
LOAD    A           AC ® M [A]
ADD     B           AC ® AC + M [B]
STORE T             M [T] ® AC          X = (A +B) × (C + A)
```

All operations are done between the AC register and a memory operand. It’s the address of a temporary memory location required for storing the intermediate result.

```
LOAD    C           AC ® M (C)
ADD     D           AC ® AC + M (D)
ML      T           AC ® AC + M (T)
STORE  X            M [×]® AC
```

Zero – Address Instruction

A stack organized computer does not use an address field for the instruction ADD and MUL. The PUSH & POP instruction, however, need an address field to specify the operand that communicates with the stack (TOS ® top of the stack)

```
PUSH      A       TOS ® A
PUSH      B       TOS ® B
ADD               TOS ®  (A + B)
PUSH      C       TOS ® C
PUSH      D       TOS ® D
ADD               TOS ® (C + D)
MUL               TOS ® (C + D) * (A + B)
POP       X       M [X] TOS
```

13. Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer register as memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction between the operand is activity referenced. Computer use addressing mode technique for the purpose of accommodating one or both of the following provisions.

(1) To give programming versatility to the uses by providing such facilities as pointer to memory, counters for top control, indexing of data, and program relocation.
(2) To reduce the number of bits in the addressing fields of the instruction.

Addressing Modes: The most common addressing techniques are:

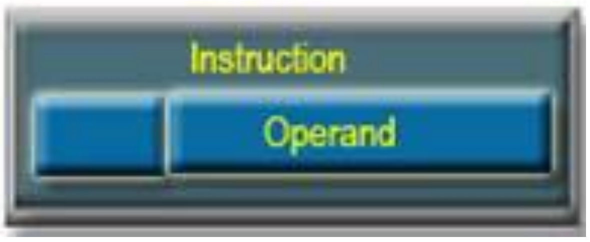
1. Immediate
2. Direct
3. Indirect
4. Register
5. Register Indirect

All computer architectures provide more than one of these addressing modes. The question arises as to how the control unit can determine which addressing mode is being used in a particular instruction. Several approaches are used. Often, different opcodes will use different addressing modes. Also, one or more bits in the instruction format can be used as a mode field. The value of the mode field determines which addressing mode is to be used. What is the interpretation of effective address. In a system without virtual memory, the effective address will be either a main memory address or a register. In a virtual memory system, the effective address is a virtual address or a register. The actual mapping to a physical address is a function of the paging mechanism and is invisible to the programmer.

Opcode	Mode	Address

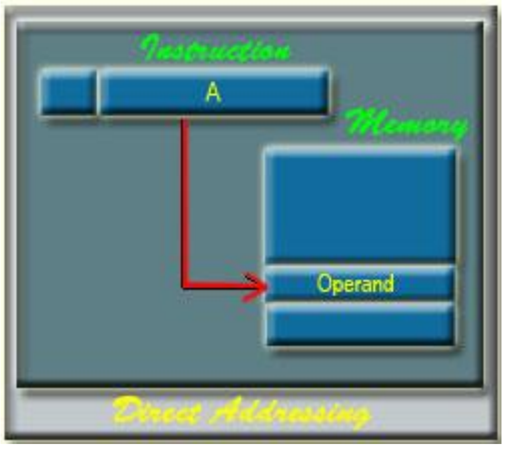
1. Immediate Addressing:
The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:
OPERAND = A

This mode can be used to define and use constants or set initial values of variables. The advantage of immediate addressing is that no memory reference other than the instruction fetch is required to obtain the operand. The disadvantage is that the size of the number is restricted to the size of the address field, which, in most instruction sets, is small compared with the world length.



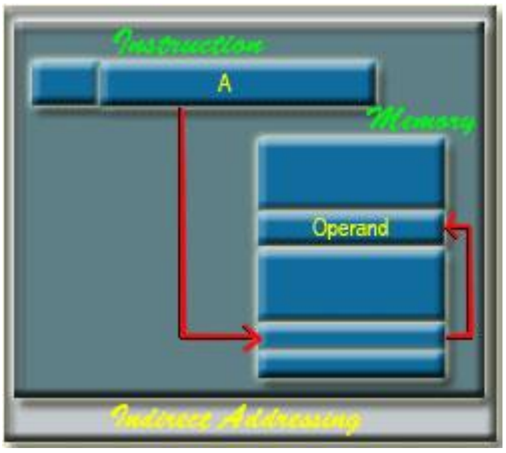
2. Direct Addressing:
A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand:

EA = A
It requires only one memory reference and no special calculation.



3. Indirect Addressing:
With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing:

EA = (A)

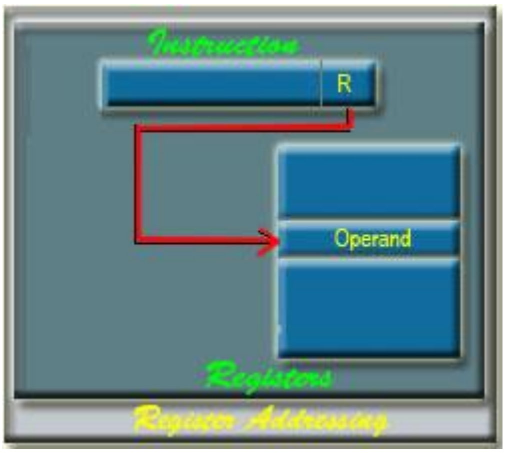


4. Register Addressing:

Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address:

$$EA = R$$

The advantages of register addressing are that only a small address field is needed in the instruction and no memory reference is required. The disadvantage of register addressing is that the address space is very limited.



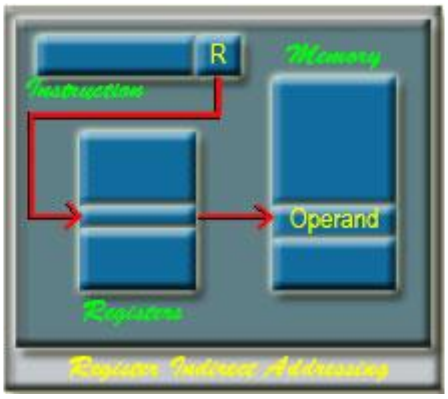
The exact register location of the operand in case of Register Addressing
Mode is shown in the Figure 34.4. Here, 'R' indicates a register where the operand is present.

5. Register Indirect Addressing:

Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location. It requires only one memory reference and no special calculation.

EA = (R)

Register indirect addressing uses one less memory reference than indirect addressing. Because, the first information is available in a register which is nothing but a memory address. From that memory location, we use to get the data or information. In general, register access is much more faster than the memory access.

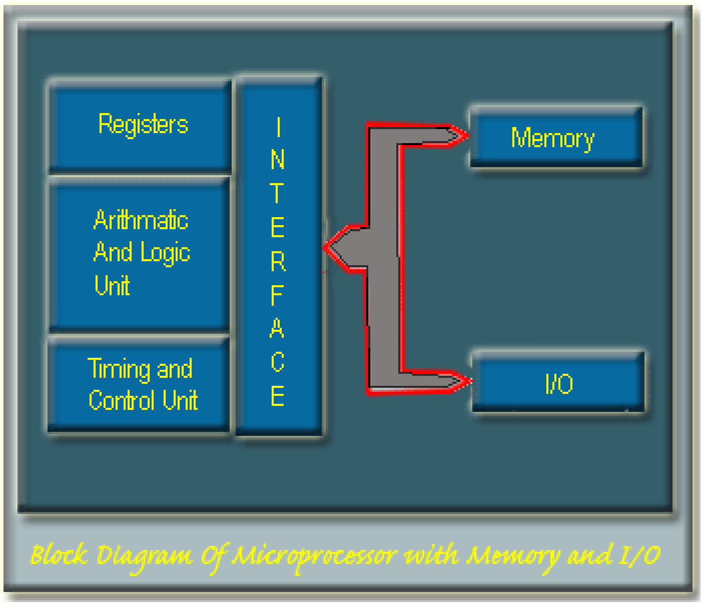


14. 8085 microprocessor organization

The microprocessors that are available today came with a wide variety of capabilities and architectural features. All of them, regardless of their diversity, are provided with at least the following functional components, which form the central processing unit (CPU) of a classical computer.

a) Register Section: A set of registers for temporary storage of instructions, data and address of data.
b) Arithmetic and logic unit: Hardware for performing primitive arithmetic and logical operations.
c) Interface Section: Input and output lines through which the microprocessor communicates with the outside world.
d) Timing and control Section: Hardware for coordinating and controlling the activities of the various sections within the microprocessor and other devices connected to the interface section.

The block diagram of the microprocessor along with the memory and input / output (I/O) devices is shown in the figure.


Registers
Arithmatic And Logic Unit
Timing and Control Unit
INTERFACE
Memory
I/O
Block Diagram Of Microprocessor with Memory and I/O

1 Intel Microprocessors:

Intel 4004 is the first 4-bit microprocessor introduced by Intel in 1971. After that Intel introduced its first 8-bit microprocessor 8088 in 1972.
These microprocessors could not last long as general-purpose microprocessors due to their design and performance limitations.
In 1974, Intel introduced the first general purpose 8-bit microprocessor 8080 and this is the first step of Intel towards the development of advanced microprocessor.
After 8080, Intel launched microprocessor 8085 with a few more features added to its architecture, and it is considered to be the first functionally complete microprocessor.
The main limitations of the 8-bit microprocessors were their low speed, low memory capacity, limited number of general purpose registers and a less powerful instruction set .
To overcome these limitations Intel moves from 8-bit microprocessor to 16-bit microprocessor.
In the family of 16-bit microprocessors, Intel's 8086 was the first one introduced in 1978 .
8086 microprocessor has a much powerful instruction set along with the architectural developments, which imparted substantial programming flexibility and improvement over the 8-bit microprocessor.

Microprocessor Intel 8085:

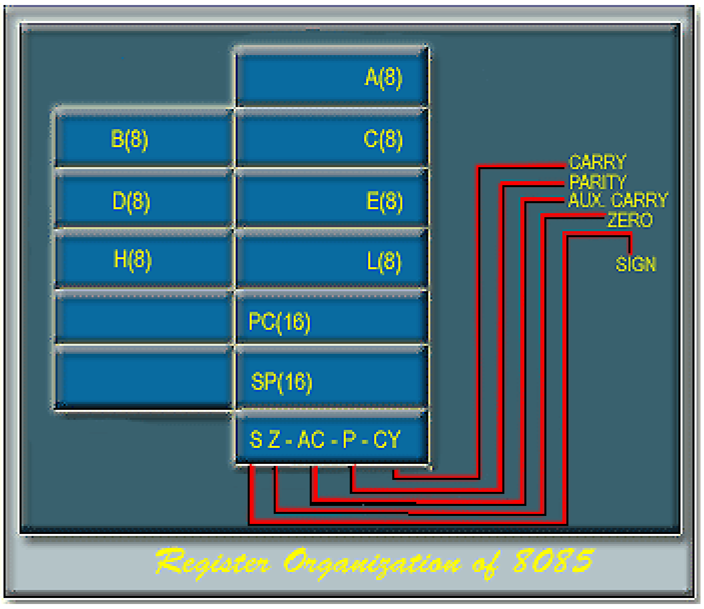
Intel 8085 is the first popular microprocessor used by many vendors . Due to its simple architecture and organization, it is easy to understand the working principle of a microprocessor .

Register in the Intel 8085:

The programmable registers of 8085 are as follows –

- One 8-bit accumulator A .
- Six 8-bit general purpose register (GPR's)
  B, C, D , E , H and L .
- The GPR's are also accessible as three 16-bit register pairs BC, DE and HL .
- There is a 16-bit program counter(PC) , one 16-bit stack pointer(SP) and 8-bit flag register . Out of 8 bits of the flag register , only 5 bits are in use .

The programmable registers of the 8085 are shown in the figure –



Apart from these programmable registers , some other registers are also available which are not accessible to the programmer . These registers include –

- Instruction Register (IR)
- Memory address and data buffers (MAR & MDR)
- Temporary register for ALU use .

ALU of 8085:

The 8-bit parallel ALU of 8085 is capable of performing the following operations –
Arithmetic: Addition , Subtraction , Increment , Decrement , Compare .
Logical : AND , OR , EXOR , NOT , SHIFT / ROTATE , CLEAR .

Because of limited chip area , complex operations like multiplication , division , etc are not available , in earlier processors like 8085 .

The operations performed on binary 2's complement data .
The five flag bits give the status of the microprocessor after and ALU operation .

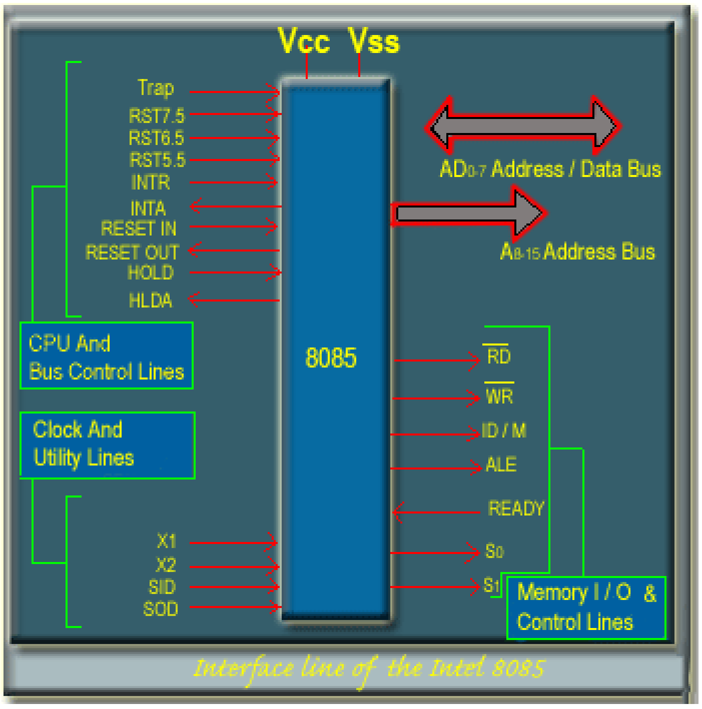
The carry (C) flag bit is set if the parity of the accumulator is even .
The Auxiliary Carry (AC) flag bit indicates overflow out of bit –3 (lower nibble) in the same manner , as the C-flag indicates the overflow out of the bit-7.

The Zero (Z) flag bit is set if the content of the accumulator after any ALU operations is zero .
The Sign(S) flag bit is set to the condition of bit-7 of the accumulator as per the sign of the contents of the accumulator(positive or negative) .

The Interface Section:

Microprocessor chips are equipped with a number of pins for communication with the outside world . This is known as the system bus .

The interface lines of the Intel 8085 microprocessor are shown in the figure –



Address and Data Bus
The AD0- AD7 lines are used as lower order 8-bit address bus and data bus , in time division multiplexed manner .
The A8-A15 lines are used for higher order 8-bit of address 8-bit of address bus.
There are seven memory and I/O control lines –

RD : indicates a READ operation when the signal is LOW .

WR : indicates a WRITE operation when the signal is LOW .

IO/M : indicates memory access for LOW and I/O access for HIGH .

ALE : ALE is an address latch enable signal , this signal is HIGH when address information is present in AD0-AD-7 . The falling edge of ALU can be used to latch the address into an external buffer to de-multiples the address bus .

READY : READY line is used for communication with slow memory and I/O devices .

S0 and S1 : The status of the system bus is difined by the S0 and S1 lines as follows –

S1	S0	Operation Specified
0	0	Halt
0	1	Memory or I/O WRITE
1	0	Memory or I/O READ
1	1	Instruction Fetch

There are ten lines associated with CPU and bus control –

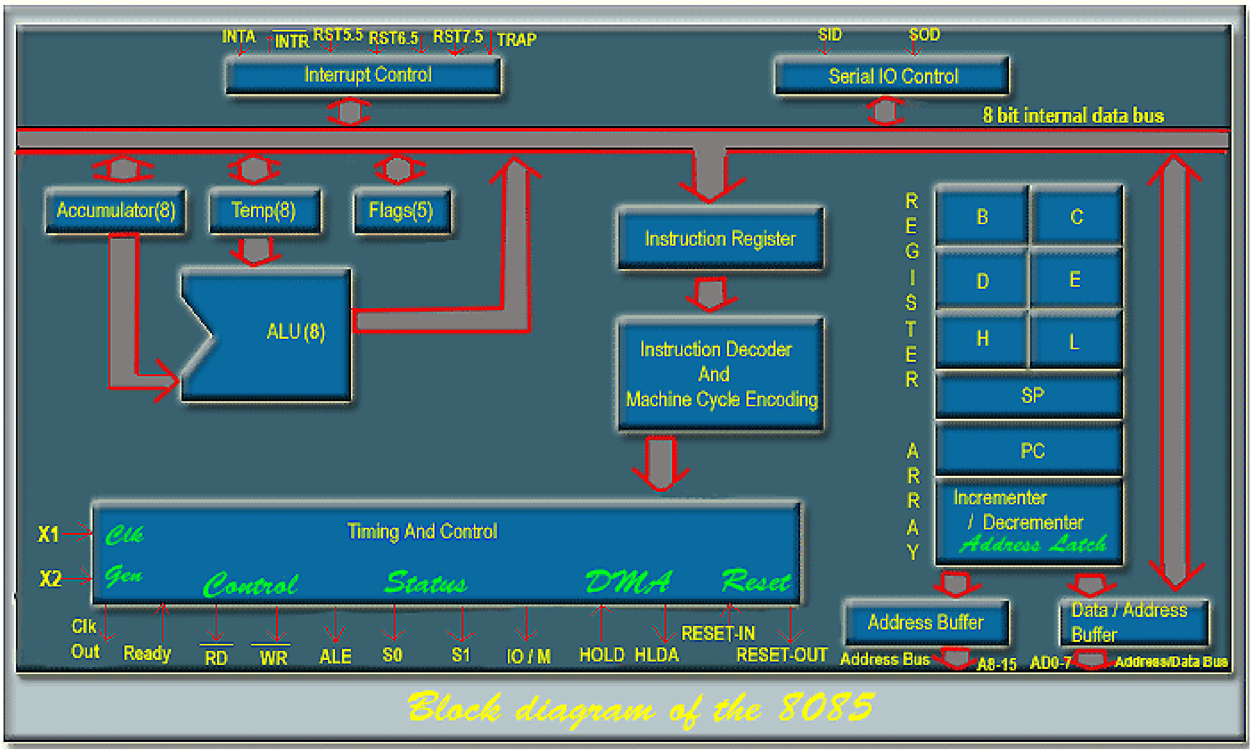
- TRAP , RST7.5 , RST6.5 , RST5.5 and INTR to acknowledge the INTA output
- RESET IN : This is the reset input signal to the 8085.
- RESET OUT : The 8085 generates the RESET-OUT signal in response to RESET-IN signal , which can be used as a system reset signal .
- HOLD : HOLD signal is used for DMA signal .
- HLDA : HLDA signal is used for DMA GRANT .

Clock and Utility Lines:

X1 and X2 : X1 and X2 are provided to connect a crystal or a RC network for generating the clock internal to the chip .

SID : input line for serial data communication
SOD : output line for serial data communication
V_{CC} and V_{SS} : Power supply .

The block diagram of the Intel 8085 is shown in the figure –


INTA
INTR
RST5.5
RST6.5
RST7.5
TRAP
Interrupt Control
SID
SOD
Serial IO Control
8 bit internal data bus
Accumulator(8)
Temp(8)
Flags(5)
ALU (8)
Instruction Register
Instruction Decoder And Machine Cycle Encoding
REGISTER ARRAY
B
C
D
E
H
L
SP
PC
Incrementer / Decrementer Address Latch
X1
X2
Clk Gen
Timing And Control
Control
Status
DMA
Reset
Clk Out
Ready
RD
WR
ALE
S0
S1
IO / M
HOLD
HLDA
RESET-IN
RESET-OUT
Address Buffer
Data / Address Buffer
Address Bus
A8-15
AD0-7
Address/Data Bus
Block diagram of the 8085

Unit-II

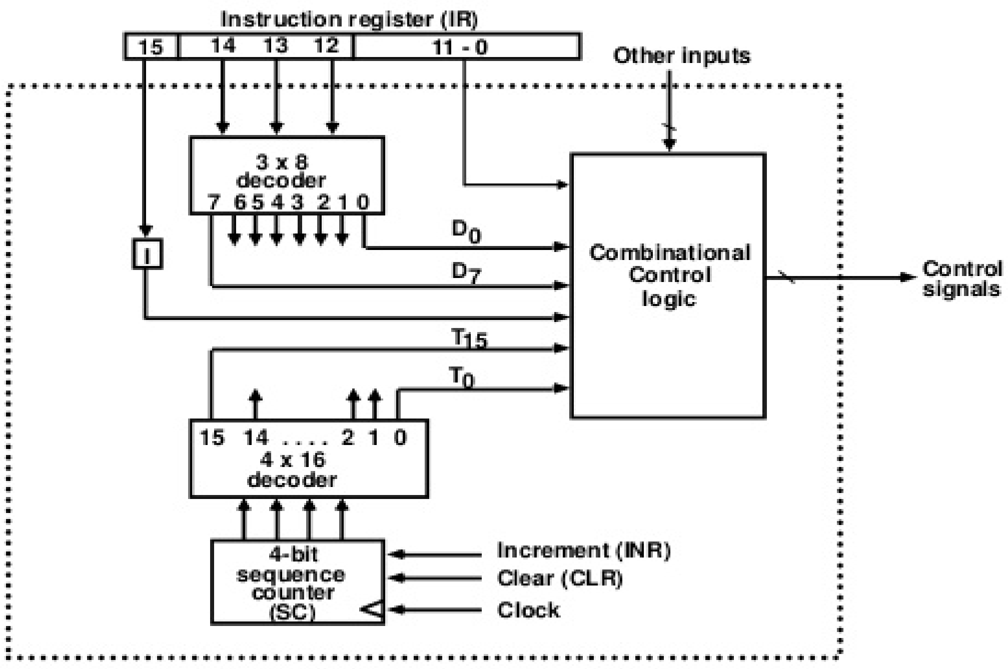
Control Unit Organization:

1. Control Unit:-

The **control unit** (CU) is a component of a computer's central processing unit (CPU) that directs the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.

It directs the operation of the other units by providing timing and control signals. Most computer resources are managed by the CU. It directs the flow of data between the CPU and the other devices. John von Neumann included the control unit as part of the von Neumann architecture in modern computer designs; the control unit is typically an internal part of the CPU with its overall role and operation unchanged since its introduction

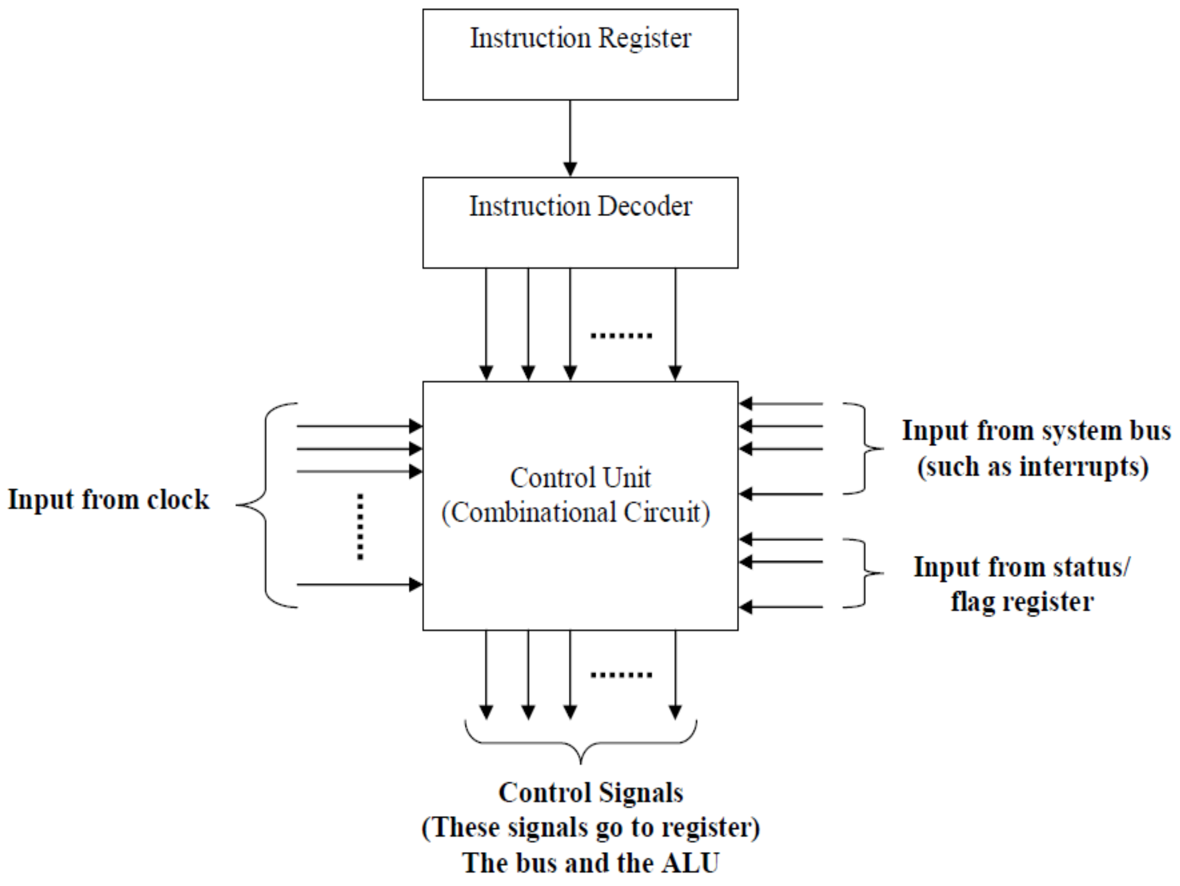
Control unit of Basic Computer



The block diagram of the control unit is shows. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The position of this register in the common bus system is indicated. The instruction register is divided into three parts: the I bit, the operation code, and bits 0 through 11. The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

2. Hardwired control unit:

Hardwired and Microprogrammed Control For each instruction, the control unit causes the CPU to execute a sequence of steps correctly. In reality, there must be control signals to assert lines on various digital components to make things happen. For example, when we perform an Add instruction in assembly language, we assume the addition takes place because the control signals for the ALU are set to "add" and the result is put into the AC. The ALU has various control lines that determine which operation to perform. The question we need to answer is, "How do these control lines actually become asserted?" We can take one of two approaches to ensure control lines are set properly. The first approach is to physically connect all of the control lines to the actual machine instructions. The instructions are divided up into fields, and different bits in the instruction are combined through various digital logic components to drive the control lines. This is called hardwired control, and is illustrated in figure

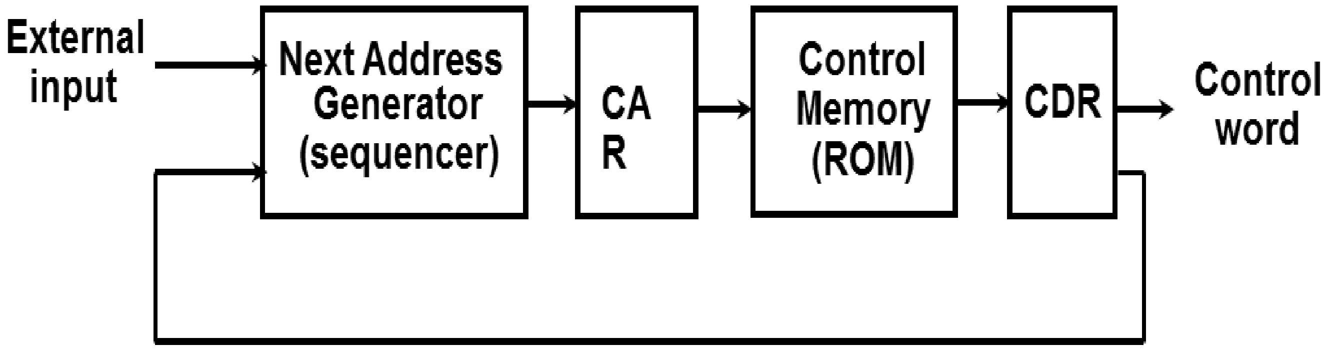


The control unit is implemented using hardware (for example: NAND gates, flip-flops, and counters).We need a special digital circuit that uses , as inputs, the bits from the Opcode field in our instructions, bits from the flag (or status) register, signals from the bus, and signals from the clock. It should produce, as outputs, the control signals to drive the various components in the computer.

3. Microprogramming control unit:

Microprogramming is a second alternative for designing control unit of digital computer (uses software for control). A control unit whose binary control variables are stored in memory is called a microprogrammed control unit. The control variables at any given time can be represented by a string of 1's and 0's called a control word (which can be programmed to perform various operations on the component of the system). Each word in control memory contains within it a microinstruction. The microinstruction specifies one or more microoperatiotins for the system. A sequence of microinstructions constitutes a microprogram. A memory that is part of a control unit is referred to as a control memory.

The general configuration of a microprogrammed control unit is demonstrated in the block diagram of Figure . The control memory is assumed to be a ROM, within which all control information is permanently stored.



Sequencer : Used to generate the address of the next microinstruction to be retrieved from the control memory.
Control Address Register : (CAR) Holds the address generated by the sequence; provides address inputs to the control memory
Control Memory : (CM) Usually a ROM; holds the control words which make up the microprogram for the MCU
Control Data Register : (CDR) Holds the control word being retrieved; used to generate/propogate control function values to the MCU

The "starting address generator " block is responsible for loading the starting address of the microprogram into the PC everytime a new instruction is loaded in the IR.
Each microinstruction basically provides the required control signal at that time step. The microprogram counter ensures that the control signal will be delivered to the various parts of the CPU in correct sequence.

4. Difference Between Hardwired control unit & Microprogrammed control unit

Hardwired control unit	**Microprogrammed control unit**
1) Speed is fast.	1) Speed is slow.
2) More costlier.	2) Cheaper.
3) Occurrence of error is more	3) Occurrence of error is less
4) Control functions implemented in hardware	4) Control functions implemented in software
5) Not flexible to accommodate new system specification or new instruction redesign is required.	5) More flexible to accommodate new system specification or new instructions.
6) Difficult to handle complex instruction sets.	6) Easier to handle complex instruction sets
7) Complicated design process.	7) Orderly, systematic and simple design process.
8) Complex decoding and sequencing logic.	8) Easier decoding and sequencing logic
9) More chip area.	9) Less chip area.
10) Application: Mostly RISC microprocessor.	10) Application: Mainframes some microprocessors

5. Advantage and disadvantage of Hardwired & Microprogrammed control unit

Advantages of hardwired control unit

1. Faster than micro- programmed control unit.
2. Can be optimized to produce fast mode of operation.

Disadvantages of hardwired control

1. Instruction set control logic are directly
2. Require change in wiring if designed has to be controlled.

Advantages of micro-programmed control unit

1. Simplifies design of CU.
2. Cheaper
3. Less error prone to implement.

Disadvantage of micro-programmed control unit

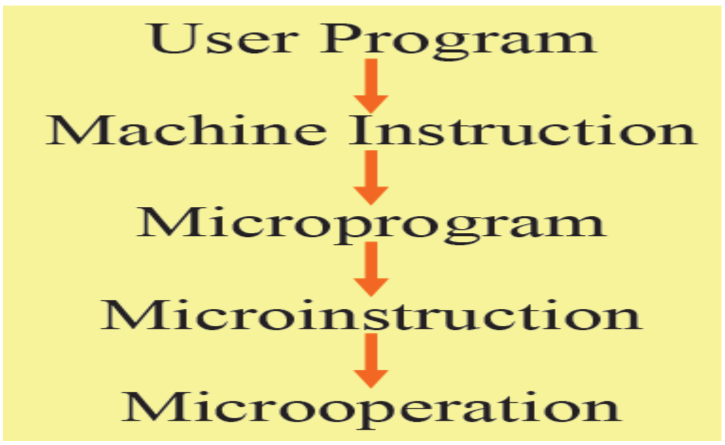
1. Slower compared to hardwired control unit.

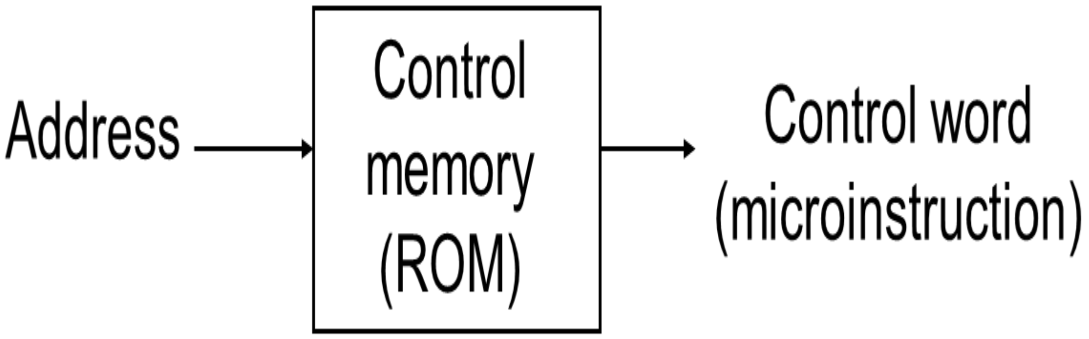
6. Control Memory,

A control memory is part of a control unit:

Computer Memory (*employs a microprogrammed control unit*)

- Main Memory : for storing user program (*Machine instruction/data*)
- Control Memory : for storing microprogram (*Microinstruction)*





- 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

7. Address Sequencing,

Each machine instruction is executed through the application of a sequence of microinstructions. Clearly, we must be able to sequence these; the collection of microinstructions which implements a particular machine instruction is called a *routine*.

The MCU typically determines the address of the first microinstruction which implements a machine instruction based on that instruction's opcode. Upon machine power-up, the CAR should contain the address of the first microinstruction to be executed.

The MCU must be able to execute microinstructions sequentially (e.g., within routines), but must also be able to ``branch" to other microinstructions as required; hence, the need for a sequencer.

The microinstructions executed in sequence can be found sequentially in the CM, or can be found by branching to another location within the CM. Sequential retrieval of microinstructions can be done by simply incrementing the current CAR contents; branching requires determining the desired CW address, and loading that into the CAR.

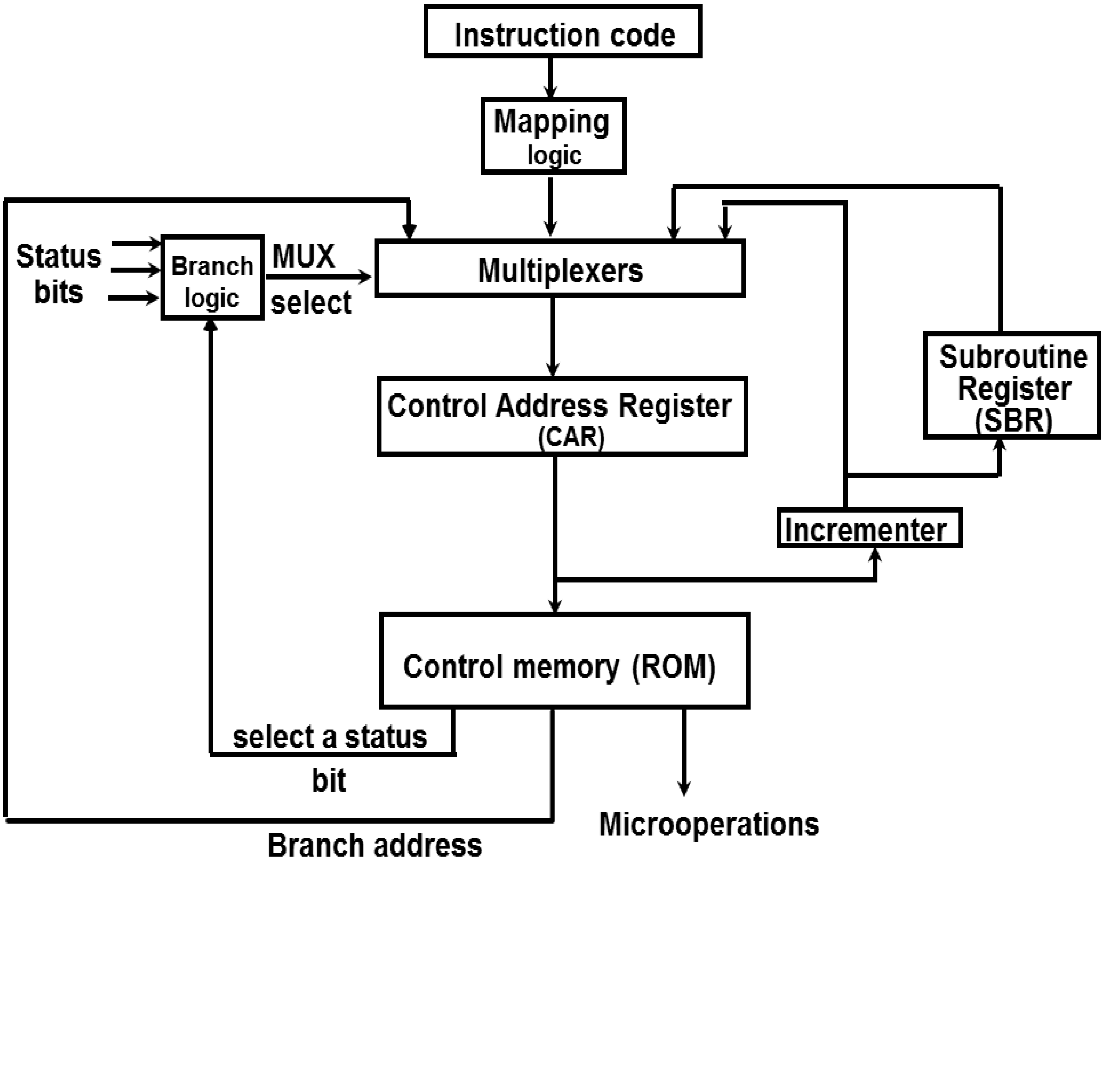
- CAR: Control Address Register
- Control ROM: Control memory (CM); holds CWs
- Opcode: Opcode field from machine instruction
- Mapping Logic: hardware which maps opcode into microinstruction address
- branch logic: Determines how the next CAR value will be determined from all the various possibilities
- Multiplexors: implements choice of branch logic for next CAR value
- Incrementer: generates CAR + 1 as a possible next CAR value
- SBR: used to hold return address for subroutine-call branch operations

Conditional branches are necessary in the microprogram. We must be able to perform some sequences of micro-ops only when certain situations or conditions exist (e.g., for conditional branching at the machine instruction level); to implement these, we need to be able to conditional execute or avoid certain microinstructions within routines.

Subroutine branches are helpful to have at the microprogram level. Many routines contain identical sequences of microinstructions; putting them into subroutines allows those routines to be shorter, thus saving memory.

Mapping of opcodes to microinstruction addresses can be done very simply. When the CM is designed, a ``required" length is determine for the machine instruction routines (i.e., the length of the longest one). This is rounded up to the next power of 2, yielding a value ***k*** such that ***2 k*** microinstructions will be sufficient to implement any routine.

The first instruction of each routine will be located in the CM at multiples of this ``required" length. Say this is ***N***. The first routine is at 0; the next, at ***N***; the next, at ***2*N***; etc. This can be accomplished very easily. For instance, with a four-bit opcode and routine length of four microinstructions, ***k*** is two; generate the microinstruction address by appending two zero bits to the opcode:



8. Micro-instruction Format

For the control memory, the microinstruction format is as shown:

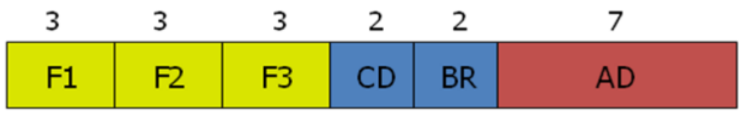


Fig: Microinstruction Fields

The micro-instruction is divided into four parts:

The three fields are microoperation field (F1, F2, and F3).
The CD field selects status bit conditions which are condition for branching
The BR field specifies the type of branch which is branch field
The AD field contains a branch address which is address field

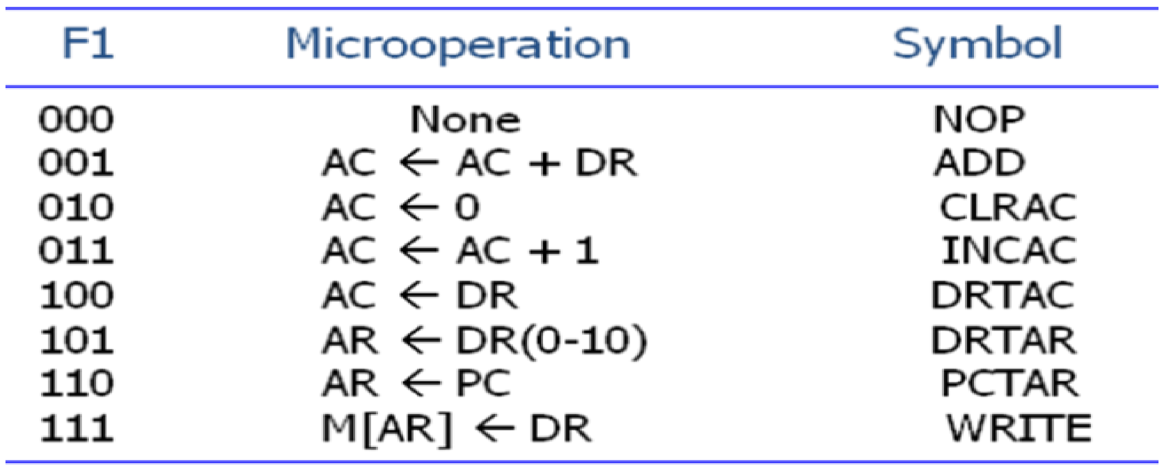
9. Micro operations:

For specification of seven different micro operations, the three bits in each field are encoded.

We cannot specify two or more micro-operations that get conflicted

e.g. 010 001 000

Each micro-operation is defined with a register transfer statement and is assigned a symbol for use in a symbolic micro program.



F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC V DR	OR
011	AC ← AC $\wedge$DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC $\oplus$ DR	XOR
010	AC ← $\overline{AC}$	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC +1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

The following table shows Symbols and Binary code for Micro-instruction Fields:

CD	Condition	Symbol	Comments
00	Always = 1	U	Uncondition branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of *AC*
11	AC = 0	Z	Zero value in *AC*

BR	Symbol	Function

00	JMP	CAR ← AD if condition = 1
		CAR ← CAR +1 if condition = 0

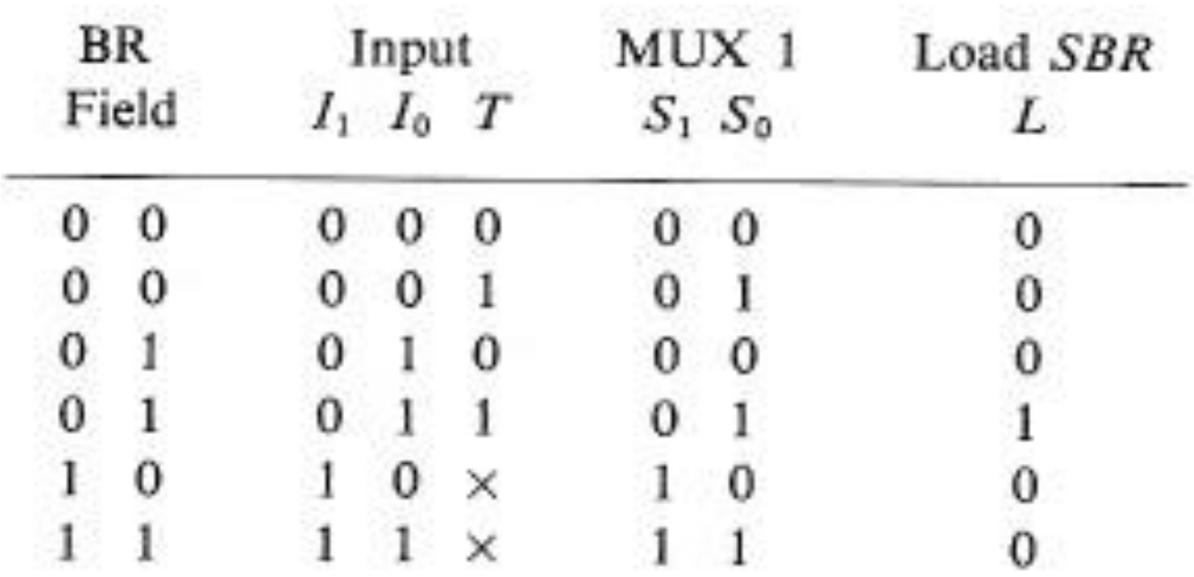
01	CALL	CAR ← AD, SBR ← CAR +1 if condition = 1
		CAR ← CAR +1 if condition = 0

10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0, 1, 6) ← 0

10.Micro program sequencer,

Basic components of a microprogrammed control unit are control memory and the circuits that select the next address. This address selection part is called a microprogram sequencer. The purpose of microprogram sequencer is to load CAR so that microinstruction may be read and executed. Commercial sequencers include within the unit an internal register stack to store addresses during microprogram looping and subroutine calls.

Internal structure of a typical microprogram sequencer is shown below in the diagram. It consists of input logic circuit having following truth table:



BR	Input			MUX 1		Load *SBR*
Field	I_1	I_0	T	S_1	S_0	L
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	×	1	0	0
1 1	1	1	×	1	1	0

Fig: Input logic truth for microprogram sequencer

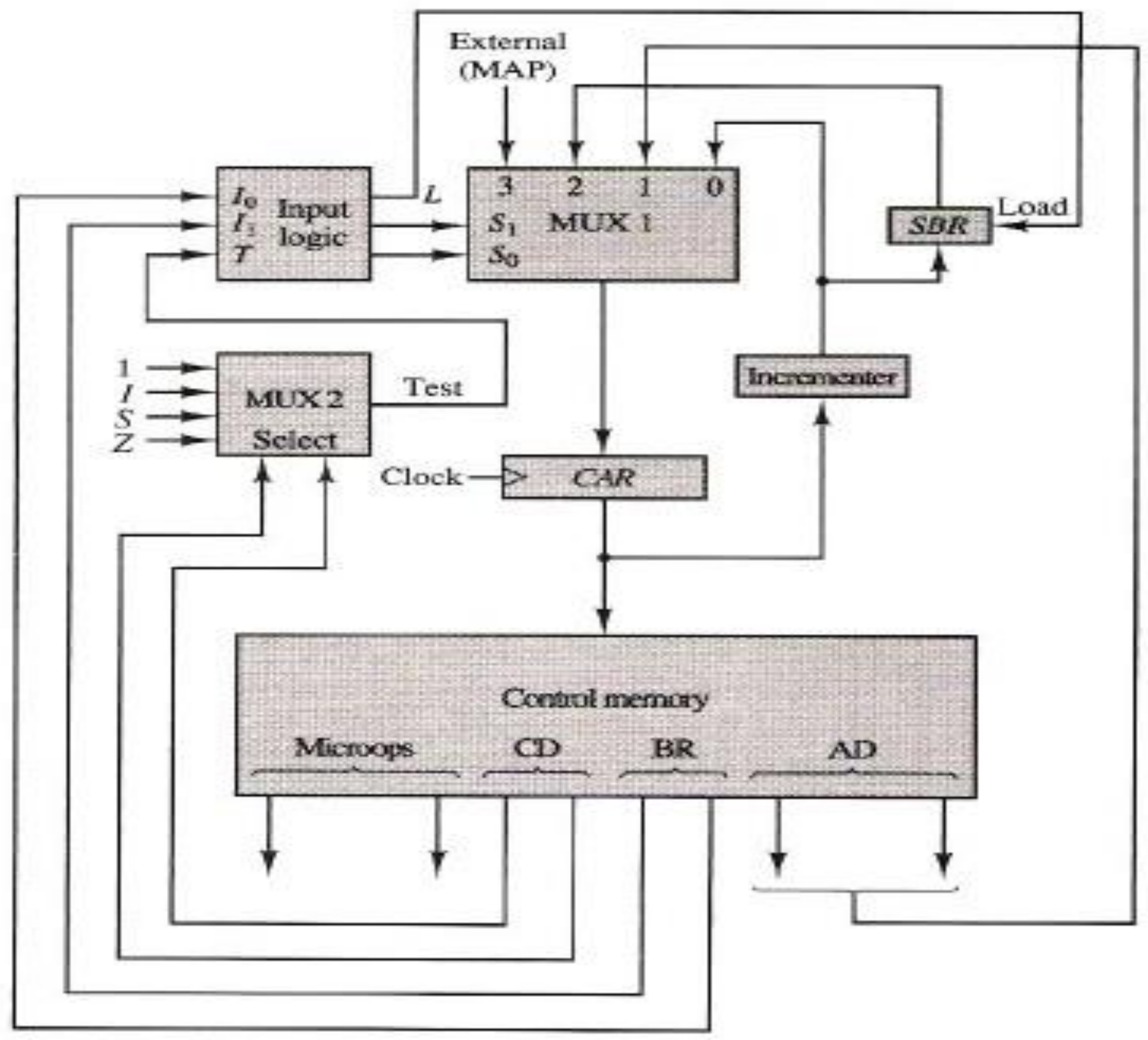


Fig. microprogram sequencer for control memory

- MUX1 selects address from one of four sources of and routes it into CAR.
- MUX2 tests the value of selected status bit and result is applied to input logic circuit.
- Output of CAR provides the address for the control memory.
- Input logic circuit has 3 inputs I0, I1 and T and 3 outputs S0, S1 and L. Variables S0 and S1 select one of the source address for CAR. L enables load input of SBR.
- e.g: when S1S0=10, MUX input number 2 is selected and establishes a transfer path from SBR to CAR.

11.Microprogramming,

Microprogramming is a technique to implement the control logic necessary to execute instructions within a processor. It is based on the general idea of fetching low-level microinstructions from a control store and deriving the appropriate control signals to be active for a single clock cycle, as well as microprogram sequencing information, from each microinstruction. Although hybrid techniques exist, microprogramming is generally contrasted with hardwired implementation techniques.

or

Microprogramming is a systematic technique for implementing the control logic of a computer's central processing unit. It is a form of stored-program logic that substitutes for hardwired control circuitry. The central processing unit in a computer system is composed of a data path and a control unit.

13. Arithmetic and Logic Unit:

Short for **Arithmetic Logic Unit**, the **ALU** is a complex digital circuit; one of many components within a computer's central processing unit. It performs both bitwise and mathematical operations on binary numbers and is the last component to perform calculations in the processor. The ALU uses to operands and code that tells it which operations to perform for input data. After the information has been processed by the ALU, it is sent to the computer's memory.

Multiple Arithmetic Logic Units can be found in CPUs, GPUs and FPUs. In some computer processors, the ALU is divided into an AU and LU. The AU performs the arithmetic operations, and the LU performs the logical operations.

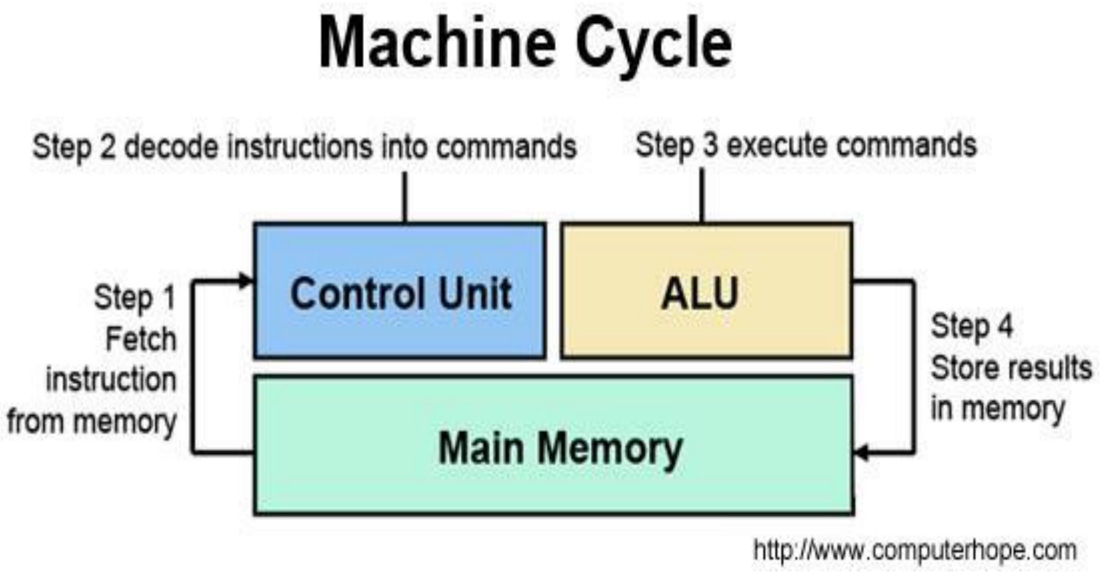

Machine Cycle
Step 2 decode instructions into commands
Step 3 execute commands
Step 1 Fetch instruction from memory
Control Unit
ALU
Step 4 Store results in memory
Main Memory
http://www.computerhope.com

Fig: ALU is responsible to perform the operation in the computer.

The basic operations are implemented in hardware level. ALU is having collection of two types of operations:

- Arithmetic operations
- Logical operations

Consider an ALU having 4 arithmetic operations and 4 logical operations.

To identify any one of these four logical operations or four arithmetic operations, two control lines are needed. Also to identify the any one of these two groups- arithmetic or logical, another control line is needed. So, with the help of three control lines, any one of these eight operations can be identified.
Consider an ALU is having four arithmetic operations. Addition, subtraction, multiplication and division. Also consider that the ALU is having four logical operations: OR, AND, NOT & EX-OR.

We need three control lines to identify any one of these operations. The input combination of these control lines are shown below:

Control line C_2 is used to identify the group: logical or arithmetic, ie
$C_2 = 0$: Arithmetic operation $C_2 = 1$: logical operation.

Control lines C_0 and C_1 are used to identify any one of the four operations in a group. One possible combination is given here.

C_1	C_0	Arithmetic $(C_2 = 0)$	Logical $(C_2 = 1)$
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	EX-OR

A 3×8 decode is used is used to decode the instruction. The block diagram of the ALU is shown in the figure.

The ALU has got two input registers named as *A* and *B* and one output storage register, named as *C*. If performs the operation as:

$$C = A \text{or } B$$

The input data are stored in *A* and *B*, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register *C*.
As for example, if the contents of controls lines are, 000, then the operation decoder enables the addition operation and in terms it activates the adder circuit and the addition

operation is performed on the data that are available in storage register *A* and *B*. After the completion of the operation, the result is stored in register *C*.

We should have some hardware implementations for basic operations. These basic operations can be used to implement some complicated operations which are not feasible to implement directly in hardware.

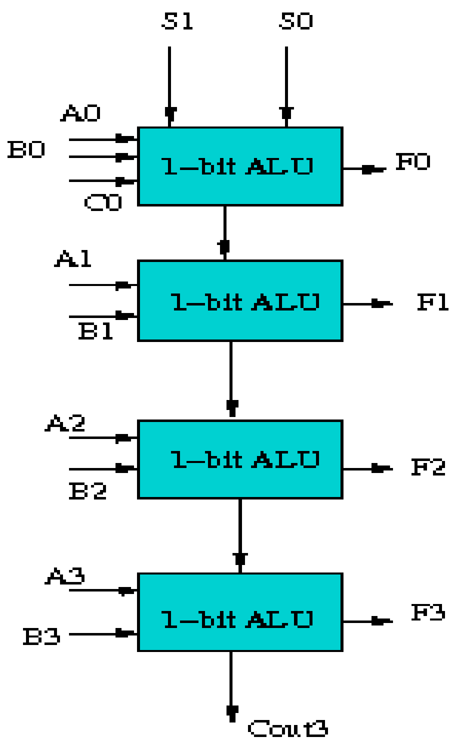
These are several logic gates exists in digital logic circuit. These logic gates can be used to implement the logical operation. Some of the common logic gates are mentioned here.

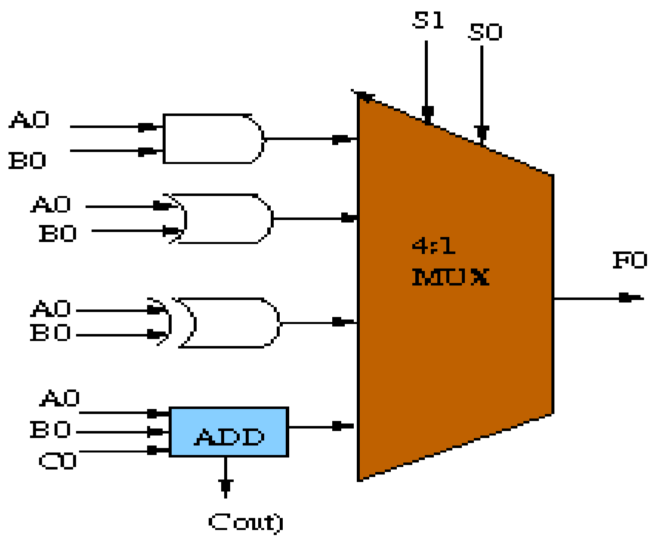
- AND gate: The output is high if both the 0-inputs are high.
- OR gate: The output is high if any one of the input is high.
- EX-OR gate: The output is high if either of the input is high.

If we want to construct a circuit which will perform the AND operation on two 4-bit number, the implementation of the 4-bit AND operation is shown in the figure.

Design of ALU:

ALU or Arithmetic Logical Unit is a digital circuit to do arithmetic operations like addition, subtraction, division, multiplication and logical operations like and, or, xor, nand, nor etc. A simple block diagram of a 4 bit ALU for operations and, or, xor and Add is shown here:





The 4-bit ALU block is combined using 4 1-bit ALU block

Design Issues:

The circuit functionality of a 1 bit ALU is shown here, depending upon the control signal S_1 and S_0 the circuit operates as follows:

for Control signal $\mathbf{S_1 = 0}$, $\mathbf{S_0}$ = 0, the output is **A And B**,

for Control signal $\mathbf{S_1 = 0}$, S_0 = 1, the output is **A Or B**,

for Control signal $\mathbf{S_1 = 1}$, S_0 = 0, the output is **A Xor B**,

for Control signal $\mathbf{S_1 = 1}$, S_0 = 1, the output is **A Add B**.

The truth table for 16-bit ALU with capabilities similar to 74181 is shown here:

Required functionality of ALU (inputs and outputs are active high)
Mode Select $\mathbf{F_n}$ for active HIGH operands
Inputs Logic Arithmetic (note 2)

S_3	S_2	S_1	S_0	(M = H)	(M = L) (Cn=L)
L	L	L	L	A'	A
L	L	L	H	A'+B'	A+B
L	L	H	L	A'B	A+B'
L	L	H	H	Logic 0	minus 1
L	H	L	L	(AB)'	A plus AB'
L	H	L	H	B'	(A + B) plus AB'
L	H	H	L	A ⊕ B	A minus B minus 1
L	H	H	H	AB'	AB minus 1
H	L	L	L	A'+B	A plus AB
H	L	L	H	(A ⊕ B)'	A plus B
H	L	H	L	B	(A + B') plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	Logic 1	A plus A (Note 1)
H	H	L	H	A+B'	(A + B) plus A
H	H	H	L	A+B	(A + B') plus A
H	H	H	H	A	A minus 1

=====================

Unit-III
Input Output Organization

1. I/O Subsystem:

Data transfer to and from peripherals may be handled in one of three possible modes:
a) Programmed I/O
b) Interrupt-initiated I/O
c) Direct memory access (DMA)

a) Programmed I/O

Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device.

The CPU issues a command then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as Polling. As a result, the level of the performance of the entire system is severely degraded.

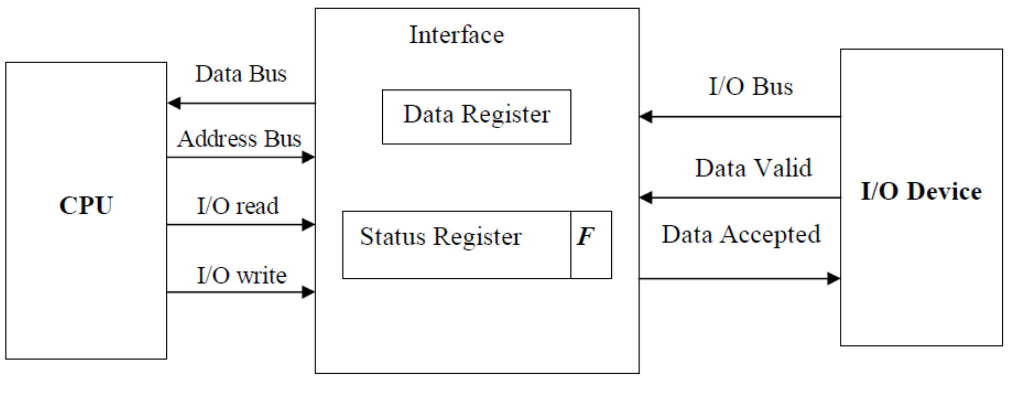
Programmed I/O basically works in these ways:

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

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. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

If the speed of an I/O device is in the right range, neither too fast for the processor to read and write the signalling bits nor too slow for the processor to wait for its activity, this form of signalling may be sufficient.

An example of data transfer from an I/O device through an interface into the CPU is shown in Figure bellow.



F= Flag Bit

Fig. (1) Data transfer from I/O to CPU

By device
When a byte of data is available, the device places it in the I/O bus and enables its data valid line.

By interface
Accepts the byte into its data register and enables the data accepted line. Sets F.

By device
Can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface. This is according to the handshaking procedure established

By program
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.

The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed.

By interface
Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the Next data byte.

Each byte is read into a CPU register and then transferred to memory with a store instruction. The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously .The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

b) Interrupt-Initiated I/O

An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the
flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.

For each interrupt there is service routine, service routine address must be known by CPU to branch to it.

The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

Below are the basic operations of Interrupt:

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

c) Direct Memory Access (DMA)

The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called direct memory access (DMA). During DMA transfer, the CPU is idle and has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals. Figure (1) shows two control signals in the CPU that facilitate the DMA transfer. The bus request (BR) input is used by the DMA controller to request the CPU to relinquish control of the buses. 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. When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways.

*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.

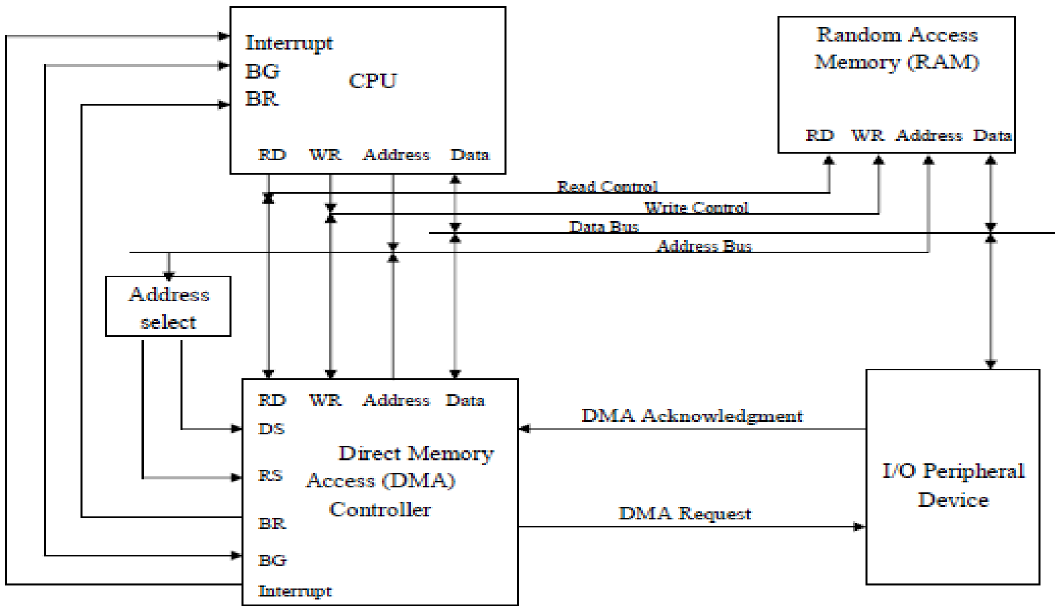


Figure (2) DMA transfer in computer system

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. In addition, it needs an **address register** (contains an address to specify the desired location in memory, the address register is incremented after each word that is transferred to memory), a **word count register** (specifies the number of words that must be transferred), a set of **address lines,** and **control register**. The address register and address lines are used for direct communication with the memory.
The position of the DMA controller among the other components in a computer system is illustrated in Figure (1). The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS (DMA select) and RS (register select) lines.
When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing the DMA that its buses are disabled. The CPU initializes the DMA by sending the following information through the data bus:

1. The starting address of the memory block where data are available (for read) or where data are to be stored (for write)
2. The word count, which is the number of words in the memory block
3. Control to specify the mode of transfer such as read or write
4. A control to start the DMA transfer

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** the RD and WR lines in the DMA Controllers are bidirectional. The direction of transfer depends on the status of the BG line. When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG = 1, the RD and WR are output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.

When the peripheral device receives a DMA acknowledge, it puts a word in the data bus (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. The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.

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. For a high-speed device, the line will be active, as soon as the previous transfer is completed. A second transfer is then initiated, and the process continues until the entire block is transferred. If the peripheral speed is slower, the DMA request line may come somewhat later. In this case the DMA disable the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses again.

If 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. When the CPU responds to the interrupt, it reads the content of the word count register. The zero value of this register indicates that all the words were transferred successfully.
DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory.

2. Interrupt structures,

- Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.
- Mainly in the microprocessor based system the interrupts are used for data transfer between the Peripheral and the microprocessor.
- If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.
- The vectored address of particular interrupt is stored in program counter.
- The processor executes an interrupt service routine (ISR) addressed in program counter.
- It returned to main program by RET instruction.

2.1 Types of Interrupts

2.1.1 Hardware Interrupts: If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are

1. **Maskable Interrupt**: Interrupt signals which can be ignored by processor, while processor doing crucial job it will ignore these kind of interrupts and continue in its state

2. **Non Maskable Interrupt**: These Interrupt signals will be not ignored and given attention all the time

2.1.2 Software Interrupts: Software interrupt can also divided in to two types. They are

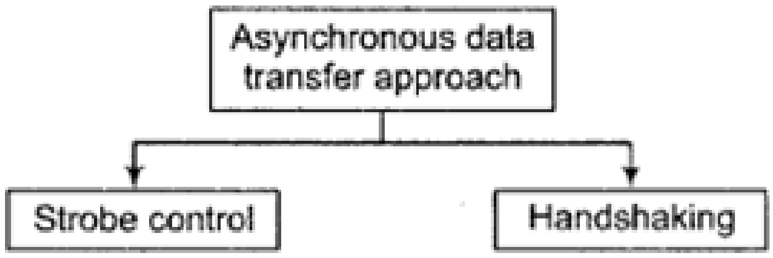
1. **Normal Interrupts:** the interrupts which are caused by the software instructions are called software instructions.
2. **Exception:** unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called an exception.

3. I/O Interface,

- The I/O system 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 end results obtained from computations must be displayed for the user. The most familiar means of entering information into a computer is through a type writer-like keyboard. On the other hand the central processing unit is an extremely fast device capable of performing operations at very high speed.
- To use a computer efficiently, a large amount of programs and data must be prepared in advance and transmitted into a storage medium such as magnetic tapes or disks. The information in the disk is then transferred into a high-speed storage, such as disks.
- Input or output devices attached to the computer are called the peripheral devices. The most common peripherals are keyboards, display units and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.

4. Asynchronous Data Transfer

The two units such as CPU and I/O interface are designed independently of each other. If the registers in the interface does not have a common clock (global clock) with the CPU registers, then the transfer between the two units is said to be asynchronous.

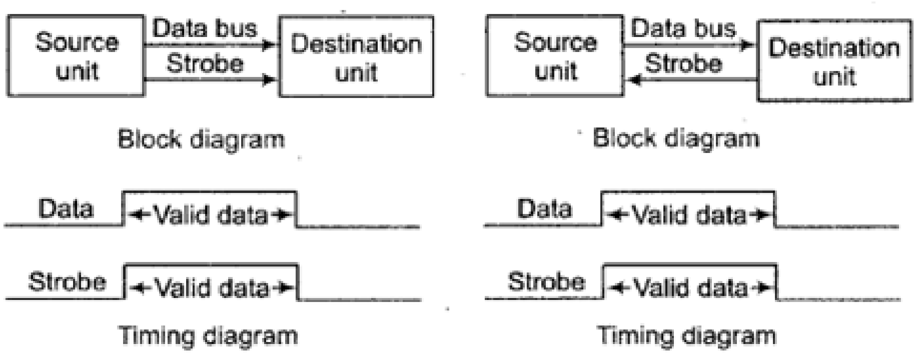


Classification of asynchronous data transfer approach

- The asynchronous data transfer requires the control signals that are being transmitted between the communicating units to indicate the time at which data is being transmitted.

a) Strobe Control

- Strobe is a pulse signal supplied by one unit to another unit to indicate the time at which data is being transmitted.



Source initiated strobe for data transfer

Destination initiated for data transfer

Strobe may be activated by either the source or the destination unit.

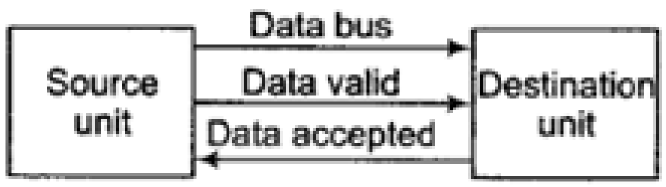
The strobe pulse is controlled by the clock pulses in the CPU. The data bus carries the binary information from source unit to the destination unit. In source initiated strobe for data transfer, the strobe is a single line that informs the destination unit when a valid data word is available in the bus.

But in destination initiated for data transfer it informs the source to provide the data. Then source unit places the data on the data bus.

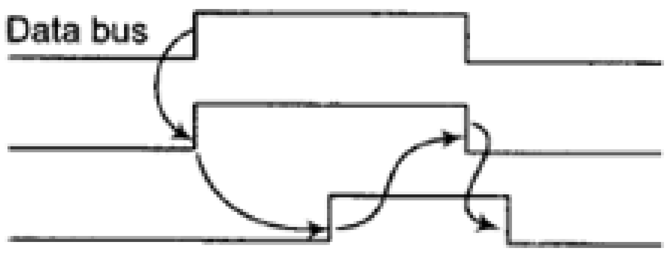
b) Handshaking

The disadvantage of the strobe method is that the source unit has no information whether the destination unit has actually received the data item, if the source unit initiates the transfer. But if the destination unit initiates the transfer it has no way of knowing whether the source unit has actually placed the data on the bus. The handshake method solves this problem.

The basic approach of handshaking is as follows. In handshaking method, there are two control signals unlike strobe control method. One control signal is in the same direction as the data flow in the bus from the source to the destination. This signal is used to inform the destination unit whether there are valid data in the bus. The second control signal is in the other direction from the destination to the source. It is used to inform the source whether it can accept data.



Block diagram of handshaking



Timing diagram

4.1 Synchronous Data Transfer

In synchronous data transfer a global or shared clock is provided to both sender and receiver. The sender and receiver work simultaneously.

5. I/O processor

The input/output processor or I/O processor is a processor that is separate from the main processor or CPU designed to handle only input/output processes for a device or the computer.

The concept of I/O processor is an extension of the concept of DMA. The I/O processor can execute specialised I/O program residing in the memory without intervention of the CPU. Thus, CPU only needs to specify a sequence of I/O activity to I/O processor. The I/O processor then executes the necessary I/O instructions which are required for the task; and interrupts the CPU only after the entire sequence of I/O activity as specified by CPU have been completed. An advanced I/O processor can have its own memory, enabling a large set of I/O devices to be controlled without much involvement from the CPU. Thus, an I/O processor has the additional ability to execute I/O instructions which provide it a complete control on I/O operations. Thus, I/O processors are much more powerful than DMA which provides only a limited control of I/O device. For example, if an I/O device is busy then DMA will only interrupt the CPU and will inform the CPU again when the device is free while I/O device and once it has found to be free go ahead with I/O and when I/O finishes, communicate it to the CPU. The I/O processor is termed as channel in IMB machines.

In computer systems which have IOPs the CPU normally do not execute I/O data transfer instructions. I/O instructions are stored in memory and are executed by IOPs. The IOP can be provided with the direct access to the memory and can control the system bus. An IOP can execute a sequence of data transfer instructions involving different memory regions and different devices without intervention of the CPU.

6. 8085 I/O structure

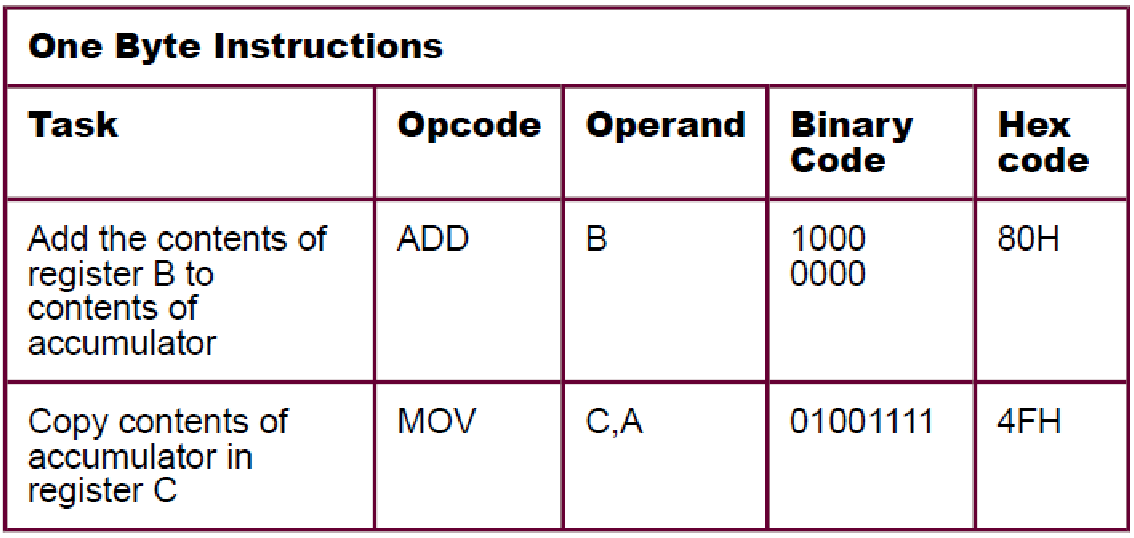
The 8085 supports up to 256 input/output (I/O) ports, accessed via dedicated Input/output instructions—taking port addresses as operands. This I/O mapping scheme is regarded as an advantage, as it frees up the processor's limited address space. The IN and OUT instructions are used to read and write I/O port data. In an I/O bus cycle, the 8-bit I/O address is output by the CPU on both the lower and upper halves of the 16-bit address bus

7. 8085 instruction set and basic programming.

7.1 8085 Instruction Format

8085 instructions are classified into following three groups of instructions:
• One-word or 1-byte instructions
• Two-word or 2-byte instructions
• Three-word or 3-byte instructions

Instruction: It is a command given to the microprocessor to perform given task on specified data. Each instruction has two parts viz. task to be performed known as operation code or **opcode** and second is the data to be operated upon known as **operand**. The Operand can be used in many different ways e.g. 8 bit data or 16 bit data or internal register or memory location or 8 bit or 16 bit address.



One Byte Instructions				
Task	**Opcode**	**Operand**	**Binary Code**	**Hex code**
Add the contents of register B to contents of accumulator	ADD	B	1000 0000	80H
Copy contents of accumulator in register C	MOV	C,A	01001111	4FH

Two Byte Instructions				
Task	**Opcode**	**Operand**	**Binary Code**	**Hex code**
Load 8 bit data byte in the accumulator	MVI	A, data	0011 1110, data	3E, data

If the data byte is stored in the 32H which need to be moved in the accumulator then the instruction can be written as follows: MVI A, 32H Hex code is : 3E 32H

Three Byte Instructions				
Task	**Opcode**	**Operand**	**Binary Code**	**Hex code**
Transfer the program sequence to memory location 2085H.	JMP	2085H	11000011 10000101 00100000	C3 85 20

8. Data Transfer

8.1 Serial / parallel

Serial versus Parallel Data Transfer

Information flows through the computer in many ways. The CPU is the central point for most information. When you start a program, the CPU instructs the storage device to load the program into RAM. When you create data and print it, the CPU instructs the printer to output the data.

Because of the different types of devices that send and receive information, two major types of data transfers take place within a computer: parallel and serial. These terms are used frequently, but if you're not familiar with the differences between them, check out

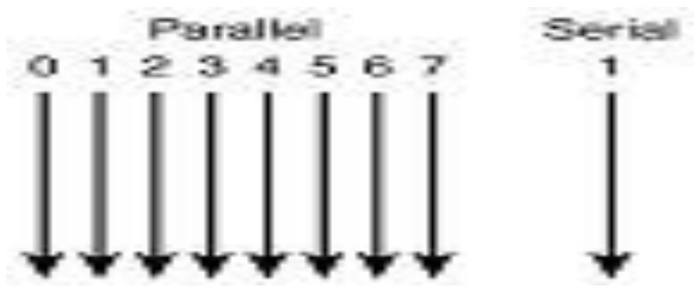


Figure: Parallel data transfers move data 8 bits at a time, whereas serial data transfers move 1 bit at a time.

8.1.1 Parallel Data Transfer

Parallel transfers use multiple "lanes" for data and programs, and in keeping with the 8 bits = 1 byte nature of computer information, most parallel transfers use multiples of 8. Parallel transfers take place between the following devices:

- CPU and RAM
- CPU and interface cards (see Chapter 8)
- **LPT** (printer) port and parallel printer
- **SCSI** port and SCSI devices
- **ATA/IDE** host adapter and ATA/IDE drives
- RAM and interface cards (either via the CPU or directly with DMA)

Why are parallel transfers so popular?

- Multiple bits of information are sent at the same time.
- At identical clock speeds, parallel transfers are faster than serial transfers because more data is being transferred.

However, parallel transfers also have problems:

- Many wires or traces (wire-like connections on the motherboard or expansion cards) are needed, leading to interference concerns and thick, expensive cables.
- Excessively long parallel cables or traces can cause data to arrive at different times. This is referred to as *signal skew*.

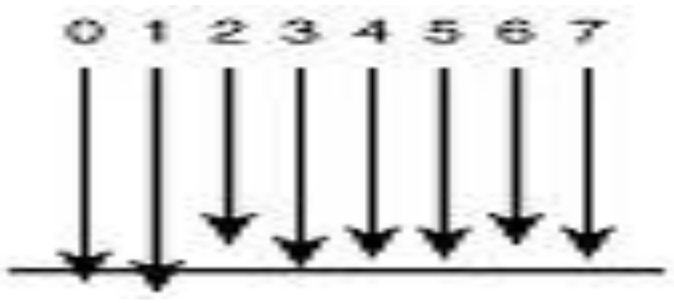


Figure : Parallel cables that are too long can cause signal skew, allowing the parallel signals to become "out of step" with each other.

- Differences in voltage between wires or traces can cause *jitter*.

As a result of these problems some compromises have had to be included in computer and system design:

- Short maximum lengths for parallel, ATA/IDE, and SCSI cables
- Dual-speed motherboards (running the CPU internally at much faster speeds than the motherboard or memory)

Fortunately, there is a second way to transmit information: serial transfers.

8.1.2 Serial Data Transfer

A **serial** transfer uses a single "lane" in the computer for information transfers. This sounds like a recipe for slowdowns, but it all depends on how fast the speed limit is on the "data highway."

The following ports and devices in the computer use serial transfers:

- Serial (also called **RS-232** or COM) ports and devices
- **USB** (Universal Serial Bus) 1.1 and 2.0 ports and devices
- Modems (which can be internal devices or can connect to serial or USB ports)
- **IEEE-1394** (**FireWire**, i.Link) ports and devices
- **Serial ATA** (SATA) host adapters and drives

Serial transfers have the following characteristics:

- One bit at a time is transferred to the device.
- Transmission speeds can vary greatly, depending on the sender and receiver.
- Very few connections are needed in the cable and ports (one transmit, one receive, and a few control and ground wires).
- Cable lengths can be longer with serial devices. For example, an UltraDMA/66 ATA/IDE cable can be only 18 inches long for reliable data transmission, whereas a Serial ATA cable can be almost twice as long.

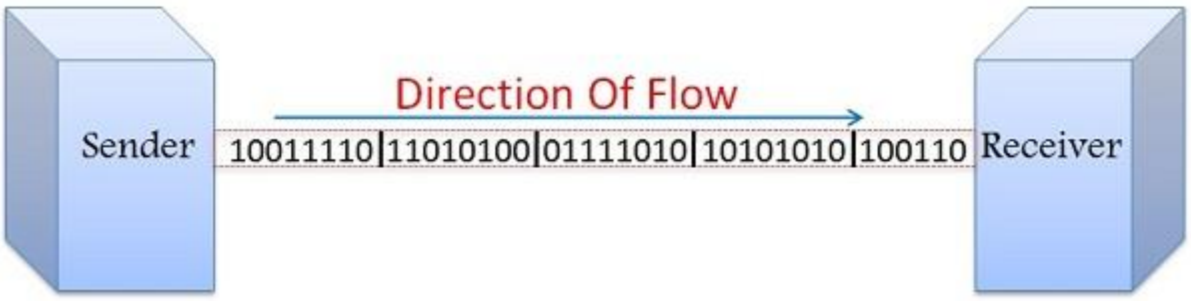
Although RS-232 serial ports are slow, newer types of serial devices are as fast or faster than parallel devices. The extra speed is possible because serial transfers don't have to worry about interference or other problems caused by running so many data lines together.

For more information about serial, parallel, USB, and IEEE-1394 ports, see Chapter 8, "Input/Output Devices and Cables." For more information about RAM, see Chapter 7, "RAM." For more information about ATA/IDE, Serial ATA, and SCSI,

8.2 Synchronous & Asynchronous Transmission

8.2.1 Synchronous Transmission

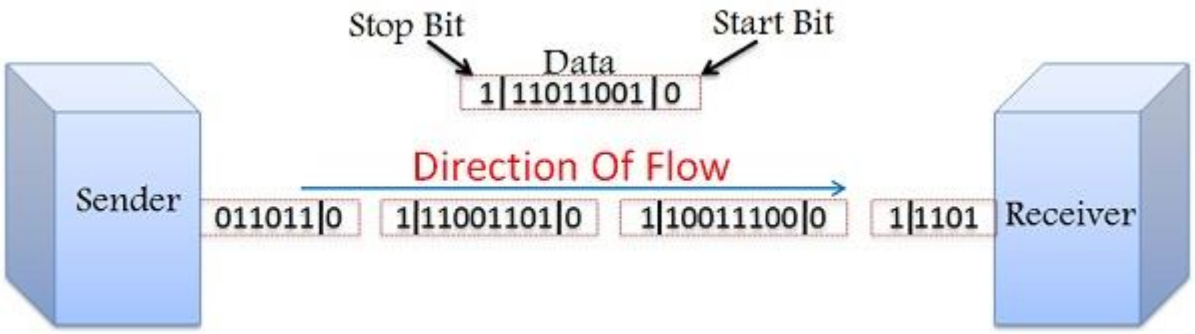
In Synchronous Transmission, data flows in a full duplex mode in the form of blocks or frames. Synchronization between the sender and receiver is necessary so that the sender know where the new byte starts (since there is no gap between the data).



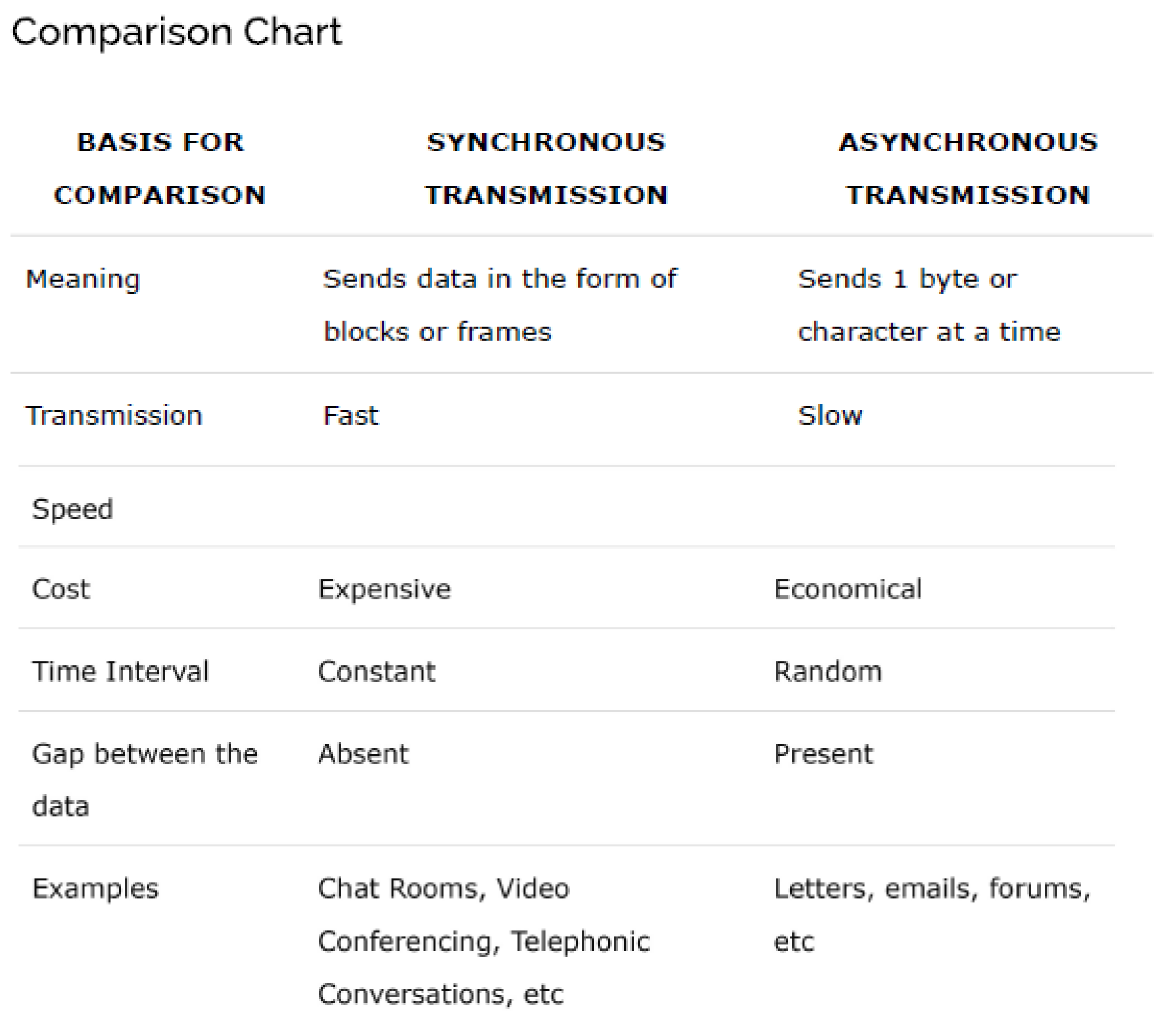
Synchronous Transmission is efficient, reliable and is used for transferring a large amount of data. It provides real-time communication between connected devices. Chat Rooms, Video Conferencing, telephonic conversations, as well as face to face interactions, are some of the examples of Synchronous Transmission.

8.2.2 Asynchronous Transmission

In Asynchronous Transmission data flows in a half duplex mode, 1 byte or a character at a time. It transmits the data in a continuous stream of bytes. In general, the size of a character sent is 8 bits to which a parity bit is added i.e. a start and a stop bit that gives the total of 10 bits. It does not require a clock for synchronization; rather it uses the parity bits to tell the receiver how to interpret the data.



It is simple, fast, and economical and does not require a 2-way communication. Letters, emails, forums, televisions and radios are some of the examples of Asynchronous Transmission.



Comparison Chart

BASIS FOR COMPARISON	SYNCHRONOUS TRANSMISSION	ASYNCHRONOUS TRANSMISSION
Meaning	Sends data in the form of blocks or frames	Sends 1 byte or character at a time
Transmission	Fast	Slow
Speed		
Cost	Expensive	Economical
Time Interval	Constant	Random
Gap between the data	Absent	Present
Examples	Chat Rooms, Video Conferencing, Telephonic Conversations, etc	Letters, emails, forums, etc

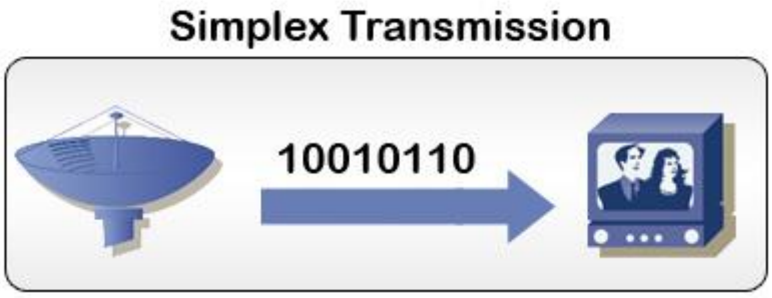
Key Differences between Synchronous and Asynchronous Transmission

1. In Synchronous Transmission data is transferred in the form of frames on the other hand in Asynchronous Transmission data is transmitted 1 byte at a time.
2. Synchronous Transmission requires a clock signal between the sender and receiver so as to inform the receiver about the new byte. Whereas, in Asynchronous Transmission sender and receiver does not require a clock signal as the data sent here has a parity bit attached to it which indicates the start of the new byte.
3. Data transfer rate of Asynchronous Transmission is slower than that of Synchronous Transmission.
4. Asynchronous Transmission is simple and economic whereas, Synchronous Transmission is complex and expensive.
5. Synchronous Transmission is efficient and has lower overhead as compared to the Asynchronous Transmission.

8.3 Simplex Half Duplex Full Duplex Data Transfer

8.3.1 Simplex

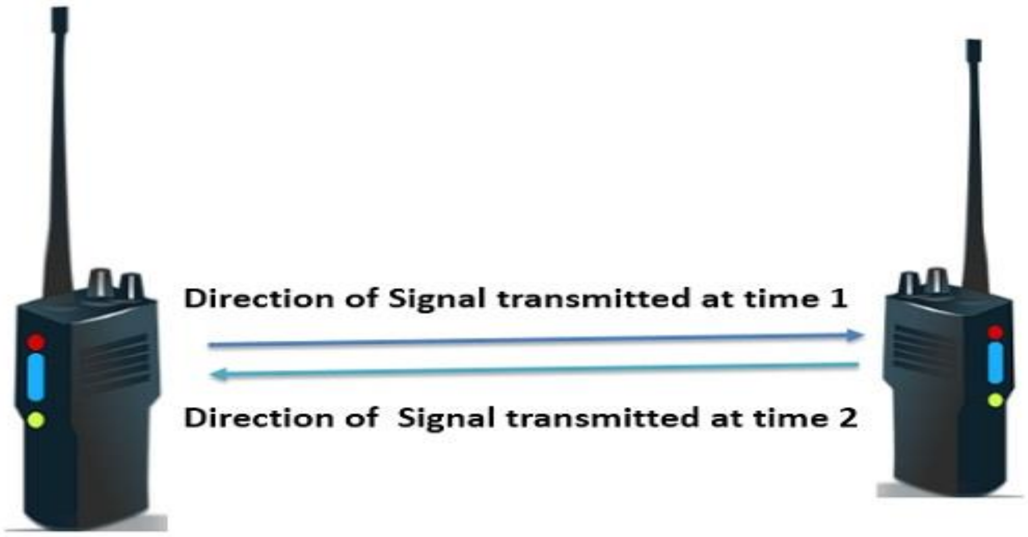
In a **simplex transmission mode**, the communication between sender and receiver occur only in one direction. That means only the sender can transmit the data, and receiver can only receive the data. The receiver cannot reply in reverse to the sender. Simplex is like a one-way road in which the traffic travels only in one direction, no vehicle from opposite direction is allowed to enter. The entire channel capacity is only utilized by the sender.



You can better understand the simplex transmission mode with an example of keyboard and monitor. The Keyboard can only transmit the input to the monitor, and the monitor can only receive the input and display it on the screen. The monitor cannot transmit any information back to the keyboard.

8.3.2 Definition of Half Duplex

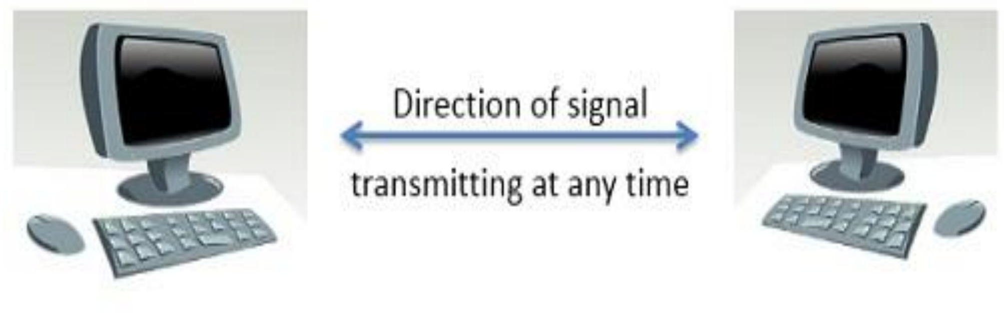
In a **half-duplex transmission mode**, the communication between sender and receiver occurs in both the directions but, one at a time. The sender and receiver both can transmit and receive the information but, only one is allowed to transmit at a time. Half duplex is still a one way road, in which a vehicle traveling in opposite direction of the traffic has to wait till the road is empty. The entire channel capacity is utilized by the transmitter, transmitting at that particular time.


Direction of Signal transmitted at time 1
Direction of Signal transmitted at time 2

Half duplex can be understood with an example of walkie-talkies. As the speaker at both the end of walkie-talkies can speak but they have to speak one by one. Both cannot speak simultaneously.

8.3.3 Definition of Full Duplex

In a **full duplex transmission mode**, the communication between sender and receiver can occur simultaneously. Sender and receiver both can transmit and receive simultaneously at the same time. The full duplex transmission mode is like a two way road in which traffic can flow in both the direction at the same time. The entire capacity of the channel is shared by both the transmitted signal traveling in opposite direction. Sharing of the channel capacity can be achieved in two different ways. First, either you physically separate the link in two parts one for sending and other for receiving. Second, or you let the capacity of a channel to be shared by the two signals traveling in opposite direction.

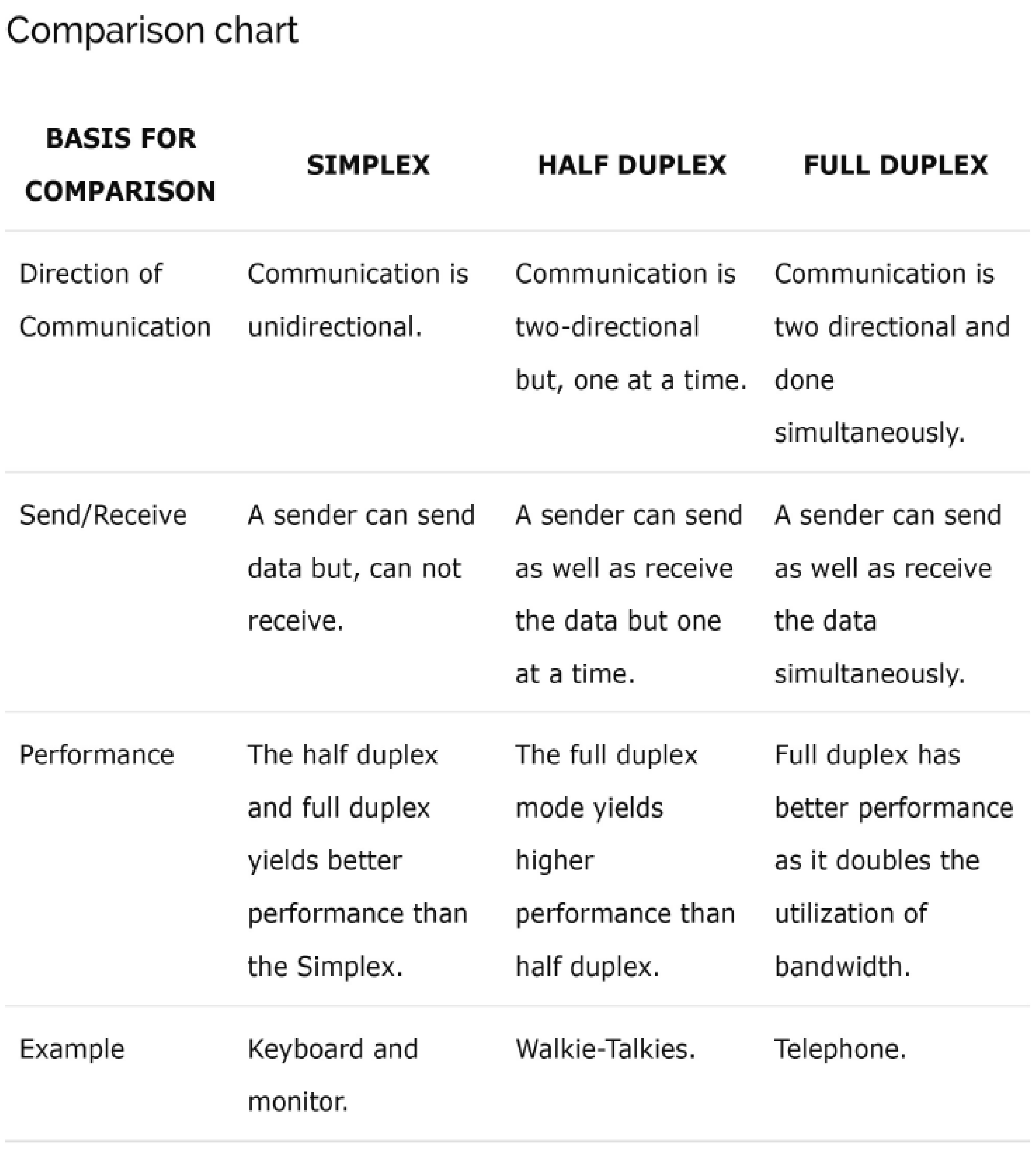


Full duplex can be understood best, with an example of a telephone. When two people communicate over a telephone both are free to speak and listen at the same time.

Key Differences Between Simplex, Half Duplex and Full Duplex

1. In a Simplex mode of transmission, the signal can be sent only in one direction; hence, it is unidirectional. On the other hand, in half duplex, both the sender and receiver can transmit the signal but, only one at a time, whereas, in full duplex, the sender and receiver can transmit the signal simultaneously at the same time.
2. In a simplex mode of transmission, only one of the two devices on the link can transmit the signal, and the other can only receive but cannot send back the signal in reverse. In a half-duplex mode, both the devices connected on the link can transmit the signal but only one device can transmit at a time. In a full-duplex mode, both the device on the link can transmit simultaneously.

3. The performance of full duplex is better than half duplex and simplex because it better utilizes the bandwidth, as compared to half duplex and simplex.
4. If we take the example of keyboard and monitor, it is observed that keyboard inputs the command and monitor displays it, monitor never replies back to the keyboard; hence, it is an example of the simplex transmission mode. In a walkie-talkie, only one person can communicate at a time so; it represents an example of half duplex mode of transmission. In a telephone, both the person on the either side of a telephone can communicate parallelly at the same time; hence, it represents an example of a full-duplex mode of transmission.



Comparison chart

BASIS FOR COMPARISON	SIMPLEX	HALF DUPLEX	FULL DUPLEX
Direction of Communication	Communication is unidirectional.	Communication is two-directional but, one at a time.	Communication is two directional and done simultaneously.
Send/Receive	A sender can send data but, can not receive.	A sender can send as well as receive the data but one at a time.	A sender can send as well as receive the data simultaneously.
Performance	The half duplex and full duplex yields better performance than the Simplex.	The full duplex mode yields higher performance than half duplex.	Full duplex has better performance as it doubles the utilization of bandwidth.
Example	Keyboard and monitor.	Walkie-Talkies.	Telephone.

Unit-IV
Memory organization

1. Memory Maps

A memory map is a massive table, in effect a database that comprises complete information about how the memory is structured in a computer system. A memory map works something like a gigantic office organizer. In the map, each computer file has a unique memory address reserved especially for it, so that no other data can inadvertently overwrite or corrupt it.

In order for a computer to function properly, its OS (operating system) must always be able to access the right parts of its memory at the right times. When a computer first boots up (starts), the memory map tells the OS how much memory is available. As the computer runs, the memory map ensures that data is always written to, and read from, the proper places. The memory map also ensures that the computer's debuggers can resolve memory addresses to actual stored data.

If there were no memory map, or if an existing memory map got corrupted, the OS might (and probably would) write data to, and read data from, the wrong places. As a result, when data was read, it would not always pertain to the appropriate files or application programs. The problem would likely start out small and unnoticeable, worsen with time, and become apparent only after considerable damage had been done to stored data and programs. In the end, some or all of the applications would fail to run, and many critical data files would be ruined.

2. Memory Hierarchy

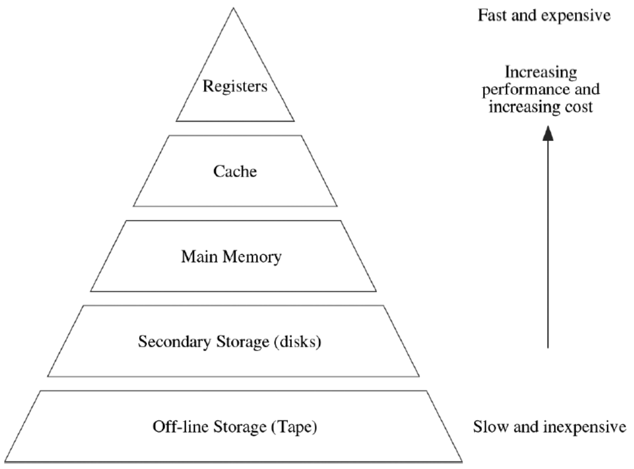
In computer architecture the memory hierarchy is a concept used to discuss performance issues in computer architectural design, algorithm predictions, and lower level programming constructs involving locality of reference. The memory hierarchy in computer storage separates each of its levels based on response time. Since response time, complexity, and capacity are related, the levels may also be distinguished by their performance and controlling technologies.

Designing for high performance requires considering the restrictions of the memory hierarchy, i.e. the size and capabilities of each component. Each of the various components can be viewed as part of a hierarchy of memories (m1,m2,...,mn) in which each member mi is typically smaller and faster than the next highest member mi+1 of the hierarchy. To limit waiting by higher levels, a lower level will respond by filling a buffer and then signalling to activate the transfer.

There are four major storage levels.

1. Internal – Processor registers and cache.
2. Main – the system RAM and controller cards.
3. On-line mass storage – Secondary storage.
4. Off-line bulk storage – Tertiary and Off-line storage.

This is a general memory hierarchy structuring. Many other structures are useful. For example, a paging algorithm may be considered as a level for virtual memory when designing a computer architecture, and one can include a level of near line storage between online and offline storage



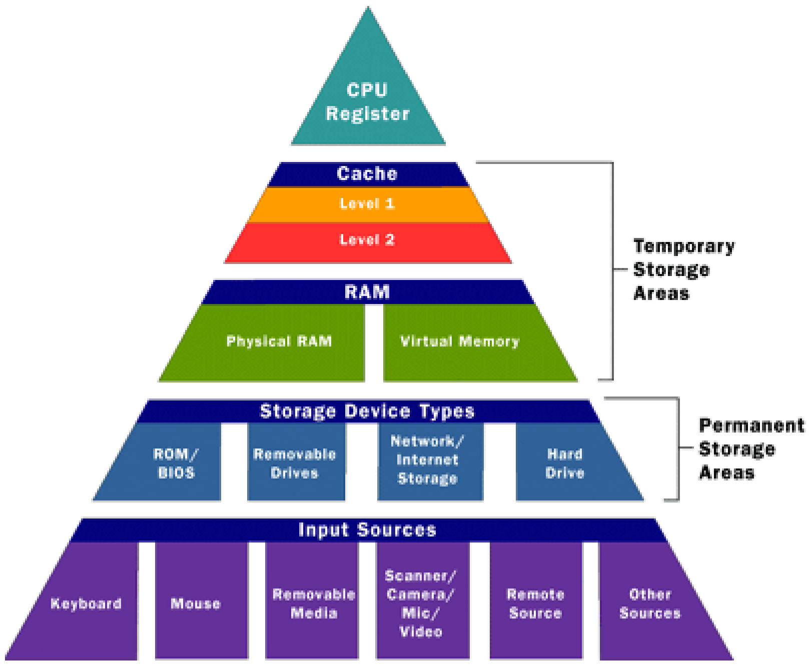
Memory type	Access time	Cost/MB	Typical amount used	Typical cost
Registers	0.5 ns	High	2 KB	–
Cache	5–20 ns	$80	2 MB	$160
Main memory	40–80ns	$0.40	512 MB	$205
Disk memory	5 ms	$0.005	40 GB	$200

3. Cache Memory

A Cache (Pronounced as “cash”) is a small and very fast temporary storage memory. It is designed to speed up the transfer of data and instructions. It is located inside or close to the CPU chip. It is faster than RAM and the data/instructions that are most recently or most frequently used by CPU are stored in cache.

The data and instructions are retrieved from RAM when CPU uses them for the first time. A copy of that data or instructions is stored in cache. The next time the CPU needs that data or instructions, it first looks in cache. If the required data is found there, it is retrieved from cache memory instead of main memory. It speeds up the working of CPU.

3.1 Types/Levels of Cache Memory



The following are the deferent levels of Cache Memory.

A CPU cache is a smaller faster memory used by the central processing unit (CPU) of a computer to reduce the average time to access memory. L1 (Level 1), L2 (Level 2), cache are some specialized memory which work hand in hand to improve computer performance.

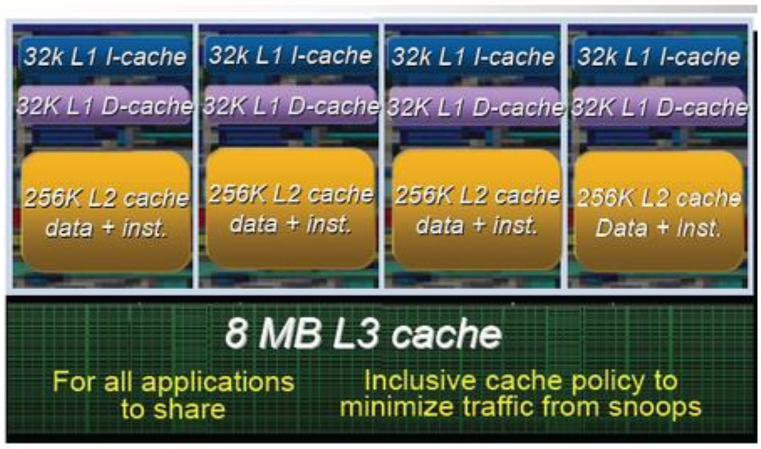
When a request is made to the system, CPU has some set of instructions to execute, which it fetches from the RAM. Thus to cut down delay, CPU maintains a cache with some data which it anticipates it will be needed.

- **L1 Cache**

L1 cache (also known as primary cache or Level 1 cache) is the top most cache in the hierarchy of cache levels of a CPU. It is the fastest cache in the hierarchy. It has a smaller size and a smaller delay (zero wait-state) because it is usually built in to the chip. SRAM (Static Random Access Memory) is used for the implementation of L1.

- **L2 Cache**

L2 cache (also known as secondary cache or Level 2 cache) is the cache that is next to L1 in the cache hierarchy. L2 is usually accessed only if the data looking for is not found in L1. L2 is usually used to bridge the gap between the performance of the processor and the memory. L2 is typically implemented using a DRAM (Dynamic Random Access Memory). Most times, L2 is soldered on to the motherboard very close to the chip (but not on the chip itself), but some processors like Pentium Pro deviated from this standard.


32k L1 I-cache
32K L1 D-cache
256K L2 cache data + inst.
32k L1 I-cache
32K L1 D-cache
256K L2 cache data + inst.
32k L1 I-cache
32K L1 D-cache
256K L2 cache data + inst.
32k L1 I-cache
32K L1 D-cache
256K L2 cache Data + Inst.
8 MB L3 cache
For all applications to share
Inclusive cache policy to minimize traffic from snoops

4. Cache memory organization and mappings

The three different types of mapping used for the purpose of cache memory are as follow, Associative mapping, Direct mapping and Set-Associative mapping.

- **Associative mapping:** In this type of mapping the associative memory is used to **store content and addresses both of the memory word**. This enables the placement of the any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.

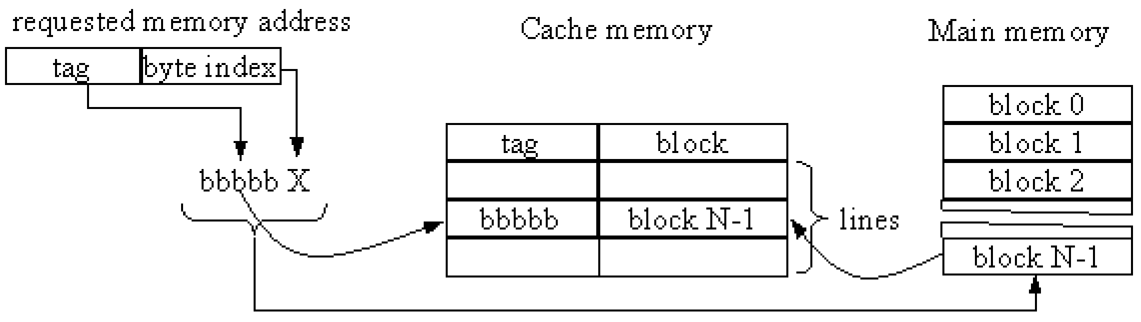
- **Direct mapping:** In direct mapping the RAM is made use of to store data and some is stored in the cache. An address space is split into two parts index field and tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

- **Set-associative mapping:** This form of mapping is a modified form of the direct mapping where the disadvantage of direct mapping is removed. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address.

4.1 Associative mapping

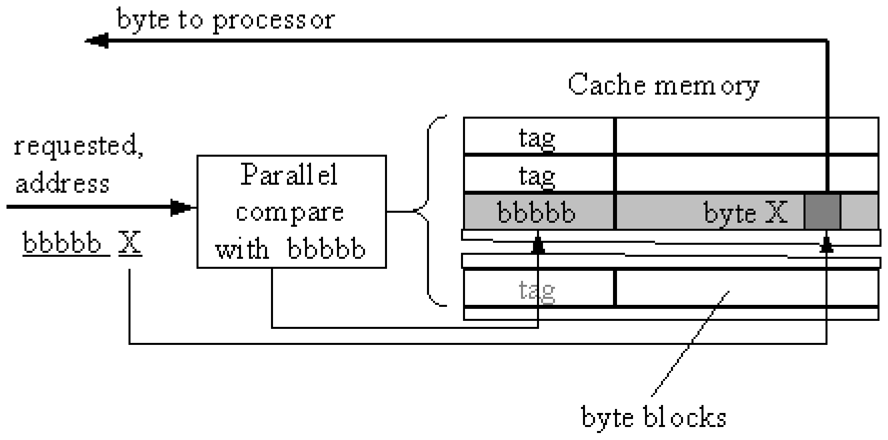
With the associative mapping of the contents of cache memory, the address of a word in the main memory is divided into two parts: the tag and the byte index (offset). Information is fetched into the cache in blocks. The byte index determines the location of the byte in the block whose address is generated from the tag bits, which are extended by zeros in the index part (it corresponds to the address of the first byte in the block. In the number of bits in the byte index is n then the size of the block is a power of 2 with the exponent n. The cache is divided into lines. In each line one block can be written together with its tag and usually some control bits. It is shown in the figure below.

When a block is fetched into the cache (on miss), the block is written in an arbitrary free line. If there is no free line, one block of information is removed from the cache to liberate one line. The block to be removed is determined according to a selected strategy, for example the least used block can be selected. To support the block selection, each access to a block residing in the cache, is registered by changing the control bits in the line the block occupies.



Information organization in cache with associative mapping

The principle of the read operation in cache memory is shown below. The requested address contains the tag (bbbbb) and the byte index in the block (X). The tag is compared in parallel with all tags written down in all lines. If a tag match is found in a line, we have a hit and the line contains the requested information block. Then, based on the byte index, the requested byte is selected in the block and read out into the processor. If none of the lines contains the requested tag, the requested block does not reside in the cache. The missing block is next fetched from the main memory or an upper level cache memory.

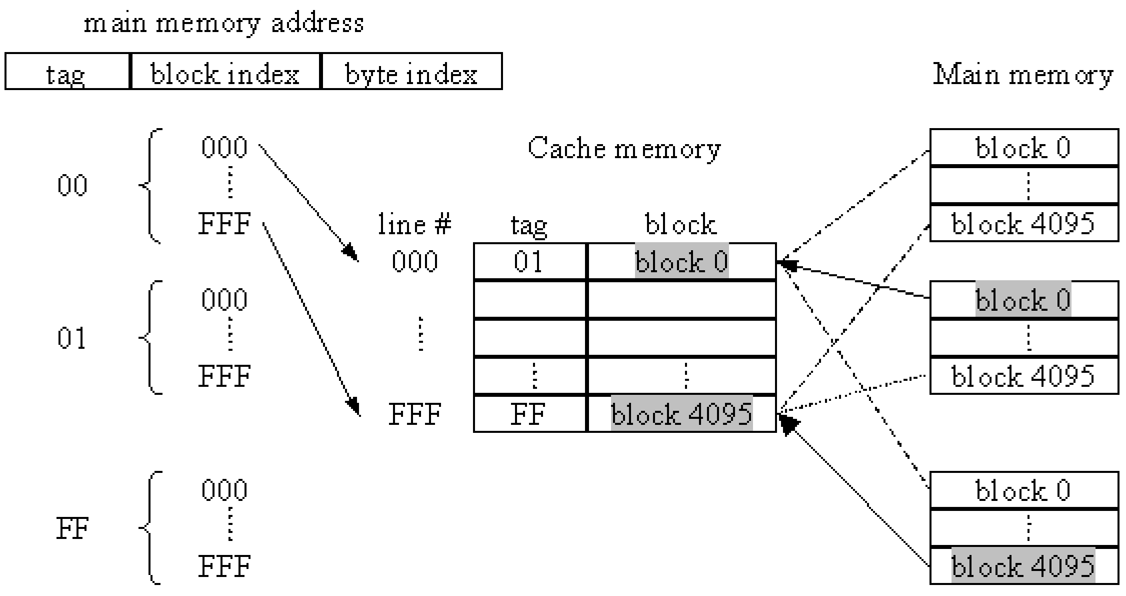


Read of a byte on hit in a cache with associative mapping

The functioning of a cache with associative mapping is based on the associative access to memory. The requested data are found by a parallel comparison of the requested tag with tags registered in cache lines. For a big number of lines, the comparator unit is very large and costly. Therefore, the associative mapping is applied in cache memories of a limited sizes (i.e. containing not too many lines).

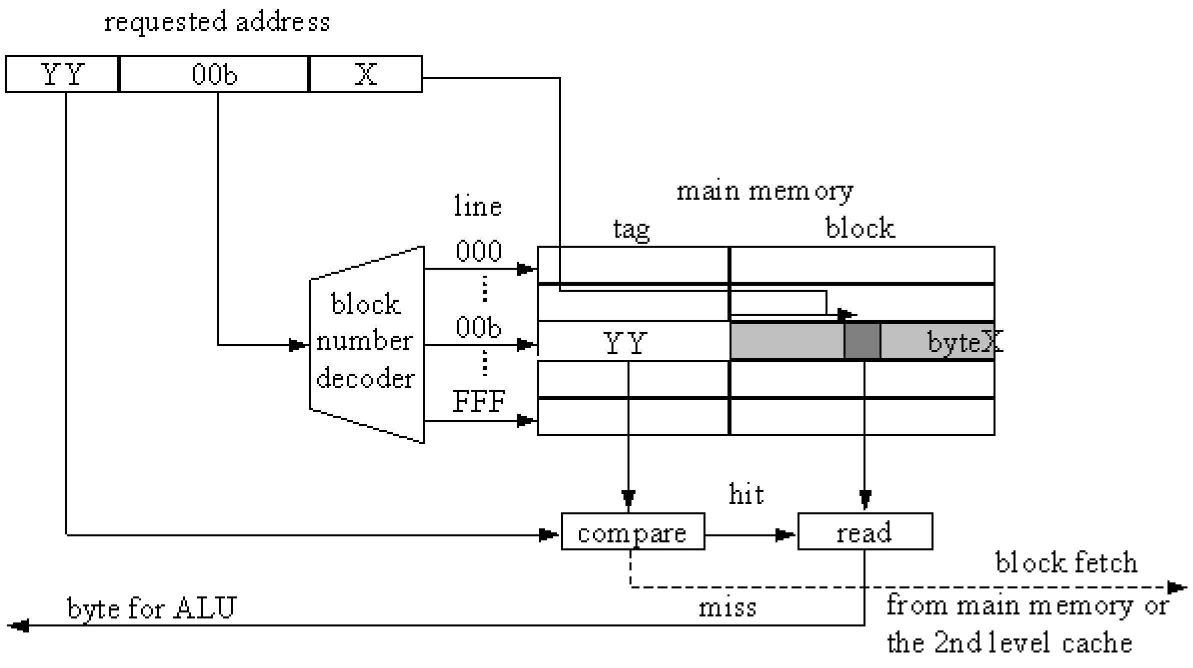
4.2. Cache memory with direct mapping

The name of this mapping comes from the direct mapping of data blocks into cache lines. With the direct mapping, the main memory address is divided into three parts: a tag, a block index and a byte index. In a given cache line, only such blocks can be written, whose block indices are equal to the line number. Together with a block, the tag of its address is stored. It is easy to see that each block number matches only one line in the cache.



The readout principle in cache with direct mapping is shown below. The block index (middle part of the address) is decoded in a decoder, which selects lines in the cache. In a selected line, the tag is compared with the requested one. If a match is found, the block residing in the line is exactly that which has been requested, since in the main memory there are no two blocks with the same block indices and tags.

We have a hit in this case and the requested byte is read in the block. If there was no tag match, it means that either there is no block yet in the line or the residing block is different to the requested one. In both cases, the requested block is fetched from the main memory or the upper level cache. Together with the fetched block, its tag is stored in the cache line.

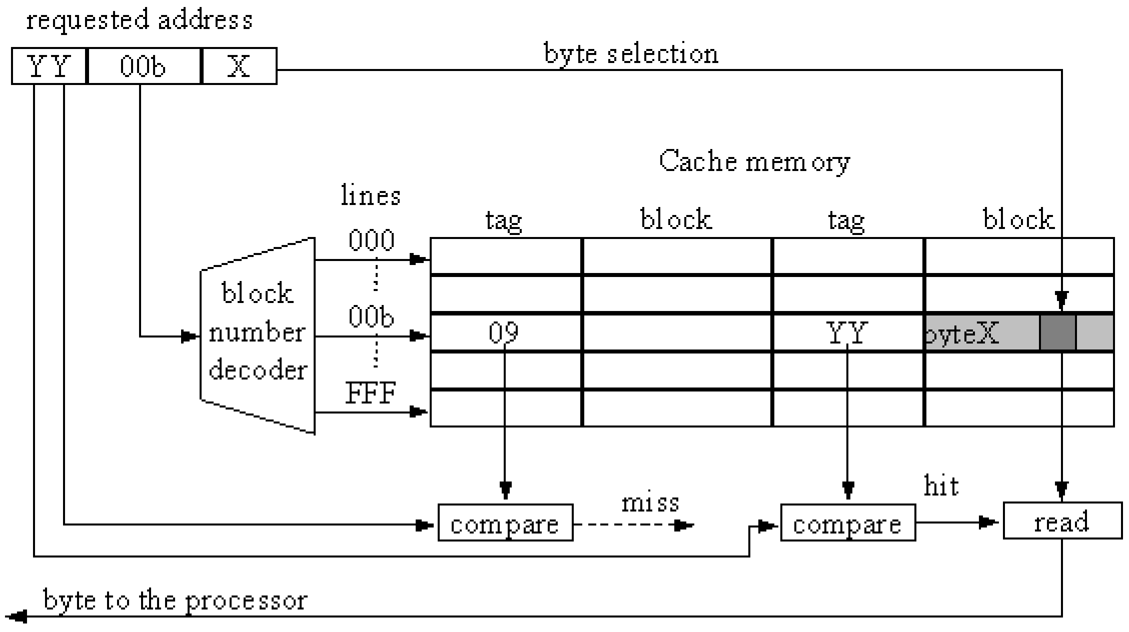


Read of a byte in a cache with direct mapping

With direct mapping, all blocks with the same index have to be written into the same cache line. It can cause frequent block swapping in cache lines, since only one block can reside in a line at a time. It is called block thrashing in the cache. For large data structures used in programs, this phenomenon can substantially decrease the efficiency of cache space use. The solution shown in next section eliminates this drawback.

4.3. Cache memory with set associative mapping

With this mapping, the main memory address is structured as in the previous case. We have there a tag, a block index and a byte index. The block into line mapping is the same as for the direct mapping. But in a set associative mapping many blocks with different tags can be written down into the same line (a set of blocks). Access to blocks written down in a line is done using the associative access principle, i.e. by comparing the requested tag with all tags stored in the selected line. From both mentioned features, the name of this mapping is derived. The figure below shows operations during a read from a cache of this type.



Read of a byte in a cache with set associative mapping

First, the block index of the requested address is used to select a line in a cache. Next, comparator circuits compare the requested tag with all stored in the line. On match, the requested byte is fetched from the selected block and sent to the processor. On miss (no match), the requested block is fetched from the main memory or the upper level cache. The new block is stored in a free block slot in the line or in the slot liberated by a block sent back to the main memory (or the upper level cache). To select a block to be removed, different strategies can be applied. The most popular is the LRU (least-recently used) strategy, where the block not used for the longest time is removed. Other strategies are: FIFO (first-in-first-out) strategy - the block that is stored during the longest time is selected or LFU (least-frequently used) strategy where the least frequently modified block is selected. To implement these strategies, some status fields are maintained associated with the tags of blocks.

Due to the set associative mapping, block thrashing in cache is eliminated to the large degree. The number of blocks written down in the same cache line is from 2 to 6 with the block size of 8 to 64 bytes.

5. Associative Memory

A type of computer memory from which items may be retrieved by matching some part of their content, rather than by specifying their ADDRESS (hence also called associative or content-addressable memory.) Associative memory is much slower than RAM, and is rarely encountered in mainstream computer designs.

For example, that serves as an identifying tag. Associative memory is used in multilevel memory systems, in which a small fast memory such as a cache may hold copies of some blocks of a larger memory for rapid access.
To retrieve a word from associative memory, a search key (or descriptor) must be presented that represents particular values of all or some of the bits of the word. This key is compared in parallel with the corresponding lock or tag bits of all stored words, and all words matching this key are signalled to be available.
Associative memory is expensive to implement as integrated circuitry.

6. Virtual Memory

Virtual memory acts as a cache between main memory and secondary memory. Data is fetched in advance from the secondary memory (hard disk) into the main memory so that data is already available in the main memory when needed. The benefit is that the large access delays in reading data from hard disk are avoided. Pages are formulated in the secondary memory and brought into the main memory. This process is managed both in hardware (Memory Management Unit) and the software (The operating systems is responsible for managing the memory resources).The block diagram shown (Book Ch.7, Section 7.6, and figure 7.37) specifies how the data interchange takes place between cache, main memory and the disk. The Memory Management unit (MMU) is located between the CPU and the physical memory. Each memory reference issued by the CPU is translated from the logical address space to the physical address space, guided by operating system controlled mapping tables. As address translation is done for each memory reference, it must be performed by the hardware to speed up the process. The operating system is invoked to update the associated mapping tables

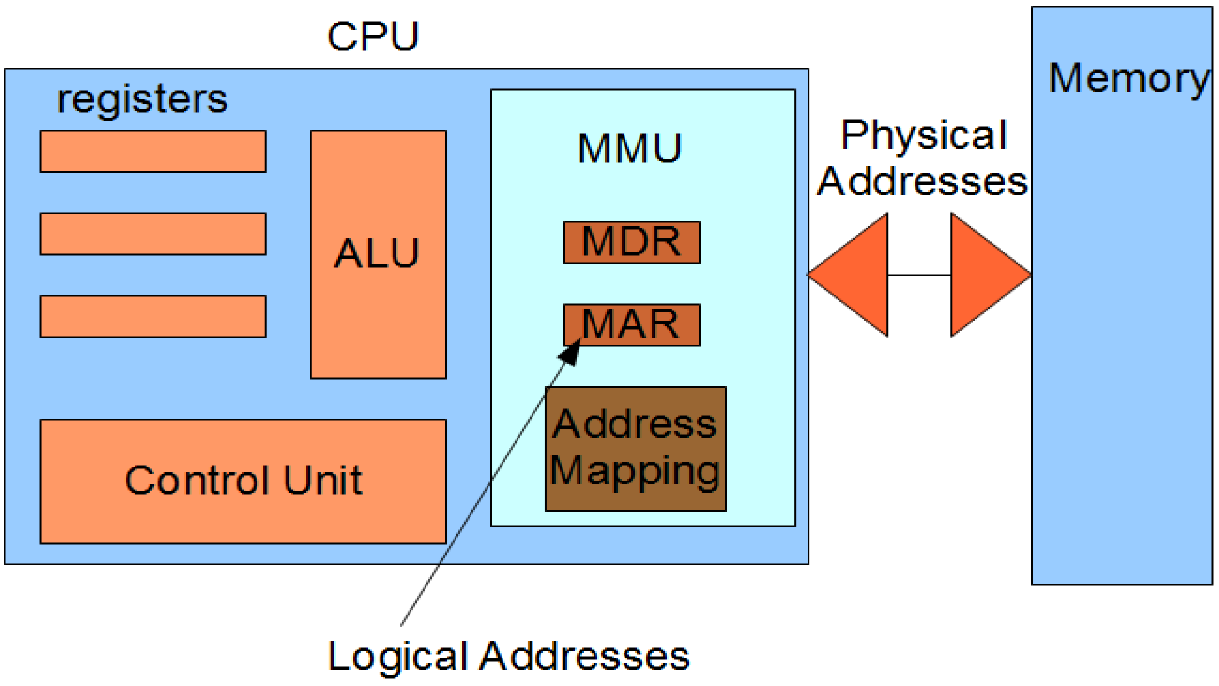
Memory Management and Address Translation the CPU generates the logical address. During program execution, effective address is generated which is an input to the MMU, which generates the virtual address. The virtual address is divided into two fields. First field represents the page number and the second field is the word field. In the next step, the MMU translates the virtual address into the physical address which indicates the location in the physical memory.

6.1 Advantages of Virtual Memory

- Simplified addressing scheme: the programmer does not need to bother about the exact locations of variables/instructions in the physical memory. It is taken care of by the operating system.
- For a programmer, a large virtual memory will be available, even for a limited physical memory.
- Simplified access control

7. Memory Management Hardware

As a program runs, the memory addresses that it uses to reference its data are the logical address. The real time translation to the physical address is performed in hardware by the CPU's Memory Management Unit (MMU). The MMU has two special registers that are accessed by the CPU's control unit. A data to be sent to main memory or retrieved from memory is stored in the Memory Data Register (MDR). The desired logical memory address is stored in the Memory Address Register (MAR). The address translation is also called address binding and uses a memory map that is programmed by the operating system.



Before memory addresses are loaded on to the system bus, they are translated to physical addresses by the MMU.

Unit-V
Multiprocessors

1. Multiprocessors:-

A multiprocessor is a computer system with two or more central processing units (CPUs), with each one sharing the common main memory as well as the peripherals. This helps in simultaneous processing of programs.

The key objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

A good illustration of a multiprocessor is a single central tower attached to two computer systems. A multiprocessor is regarded as a means to improve computing speeds, performance and cost-effectiveness, as well as to provide enhanced availability and reliability.

Different ways of using a multiprocessor include:

- As a uniprocessor, such as single instruction, single data (SISD)
- Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data (MIMD)
- A single series of instructions in various perspectives, such as single instruction, multiple data (SIMD), which is usually used for vector processing
- Multiple series of instructions in a single perspective, such as multiple instruction, single data (MISD), which is used for redundancy in failsafe systems and, occasionally, for describing hyper-threading or pipelined processors

Benefits of using a multiprocessor include:

- Enhanced performance
- Multiple applications
- Multiple users
- Multi-tasking inside an application
- High throughput and/or responsiveness
- Hardware sharing among CPUs

Communication architecture of a multiprocessor:

- Message Passing
  - Independent address space for every processor
  - Processor communication by means of message passing
  - Processors include private memories
  - Concentrates attention on high-priced, non-local operations
- Shared Memory
  - Processor communication is done by means of a shared address space
  - Processor communication is done by means of shared memory read/write
  - Convenient on small-scale devices
  - Lower latency
  - Non-uniform memory access (NUMA) or symmetric multiprocessing (SMP)

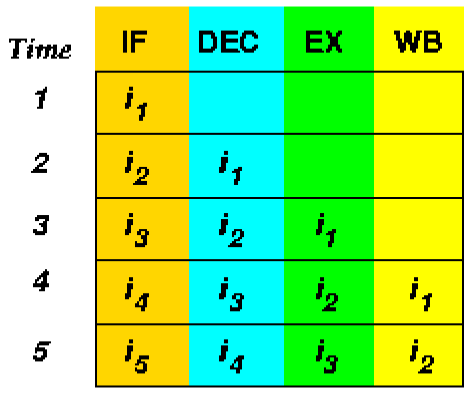
2. Pipelining:

In order to increase the instruction throughput, high performance processors make extensive use of a technique called **pipelining**. A pipelined processor doesn't wait until the result from a previous operation has been written back into the register files or main memory - it fetches and starts to execute the next instruction as soon as it has fetched the first one and dispatched it to the instruction register. When the simple processor described on the previous pages is performing an add instruction, there is no need for it to wait for the add operation to complete before it starts fetching the next instruction. So a pipelined processor will start fetching the next instruction from memory as soon as it has latched the current instruction in the instruction register.

Thus a **pipelined processor** has a pipeline containing a number of stages (4 or 5 was a common number in early RISC processors) arranged so that a new instruction is latched into its input register as the results calculated in this stage are latched into the input register of the following stage. This means that there will be a number of instructions (equal to the number of pipeline stages in the best case) "active" in the processor at any one time.

In a typical early RISC processor (*eg* MIPS R3000), there would be four pipeline stages

IF	**Instruction Fetch** Fetch the instruction from memory
DEC	**Decode and Operand Fetch** Decode it and fetch operands from the register file
EX	**Execute** Execute the instruction in the ALU
WB	**WriteBack** Write the result back in to a register



Time	IF	DEC	EX	WB
1	i_1			
2	i_2	i_1		
3	i_3	i_2	i_1	
4	i_4	i_3	i_2	i_1
5	i_5	i_4	i_3	i_2

Population of the pipeline at each clock cycle: i_1, i_2, ... are successive instructions in the instruction stream.

With an ***n***-stage pipeline, after ***n-1*** clock cycles, the pipeline will become full and an instruction completes on every clock cycle, instead of every ***n*** clock cycles. This effectively speeds up the processor by a factor of ***n***.

2.1 Advantages of Pipelining:

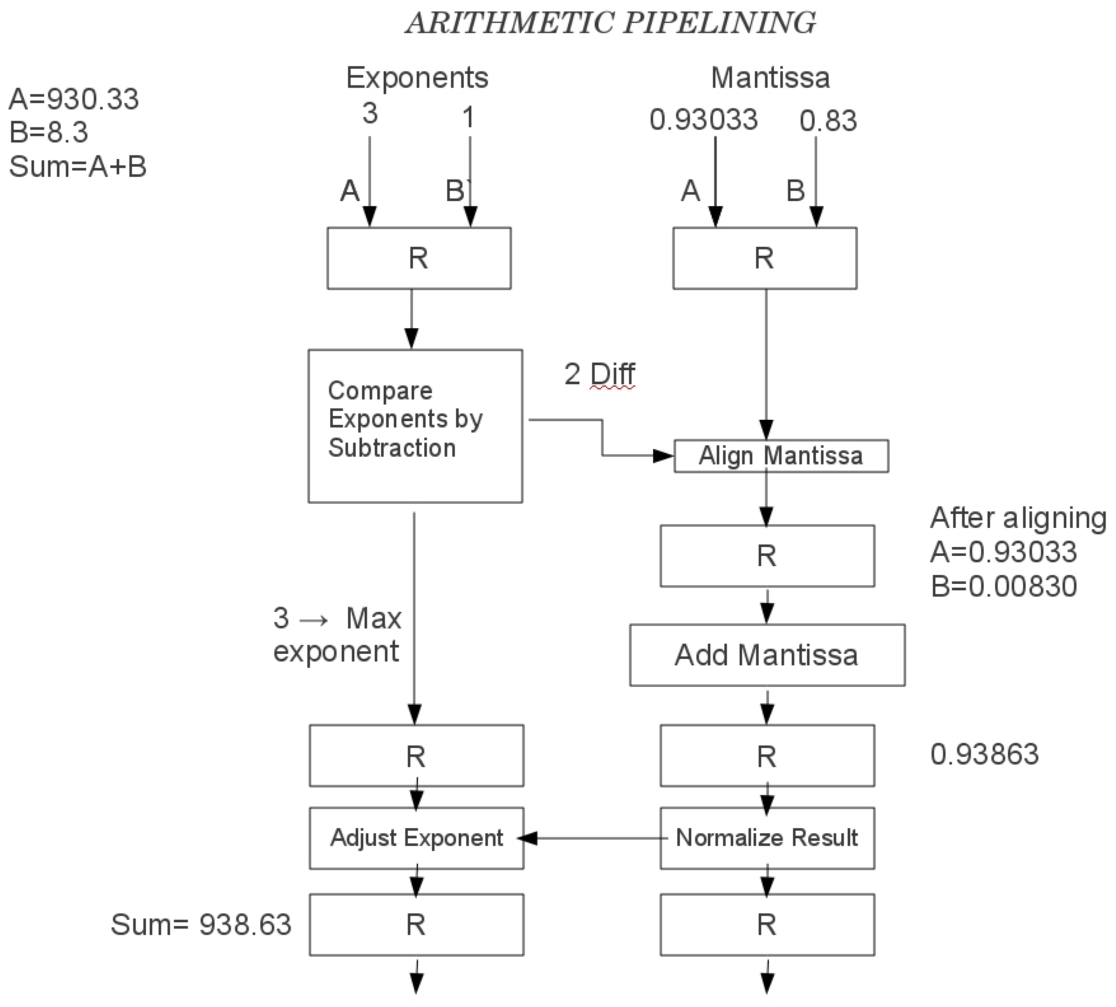
1. The cycle time of the processor is reduced; increasing the instruction throughput. Pipelining doesn't reduce the time it takes to complete an instruction; instead it increases the number of instructions that can be processed simultaneously ("at once") and reduces the delay between completed instructions (called 'throughput').
2. The more pipeline stages a processor has, the more instructions it can process "at once" and the less of a delay there is between completed instructions. Every predominant general purpose microprocessor manufactured today uses at least 2 stages of pipeline up to 30 or 40 stages.
3. If pipelining is used, the CPU Arithmetic logic unit can be designed faster, but more complex.
4. Pipelining in theory increases performance over an un-pipelined core by a factor of the number of stages (assuming the clock frequency also increases by the same factor) and the code is ideal for pipeline execution.

5. Pipelined CPUs generally work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, RAMs work at a low frequencies compared to CPUs frequencies) increasing computers overall performance.

3. Vector Processing:

Normal computational systems are not enough in some special processing requirements. Such as, in Special processing systems like artificial intelligence systems and some weather forecasting systems, terrain analysis, the normal systems are not sufficient. In such systems the data processing will be involving on very high amount of data; we can classify the large data as very big arrays. Now if we want to process this data, naturally we will need new methods of data processing. The vectors are considered as the large one dimensional array of data. The term vector processing involves the data processing on the vectors of such large data.

4. Arithmetic Pipeline:



The above diagram represents the implementation of arithmetic pipeline in the area of floating point Arithmetic operations. In the diagram, we can see that two numbers A and B are added together. Now the values of A and B are not normalized, therefore we must normalize them before start to do any operations. The first thing is we have to fetch the values of A and B into the registers. Here R denote a set of registers. After that the values of A and B are normalized, therefore the values of the exponents will be compared in the comparator. After that the alignment of mantissa will be taking place. Finally, we will be performing addition, since an addition is happening in the adder circuit. The source registers will be free and the second set of values can be brought. Likewise when the normalizing of the result is taking place, addition of the new values will be added in the adder circuit and when addition is going on, the new data values will be brought into the registers in the start of the implementation. We can see how the addition is being performed in the diagram.

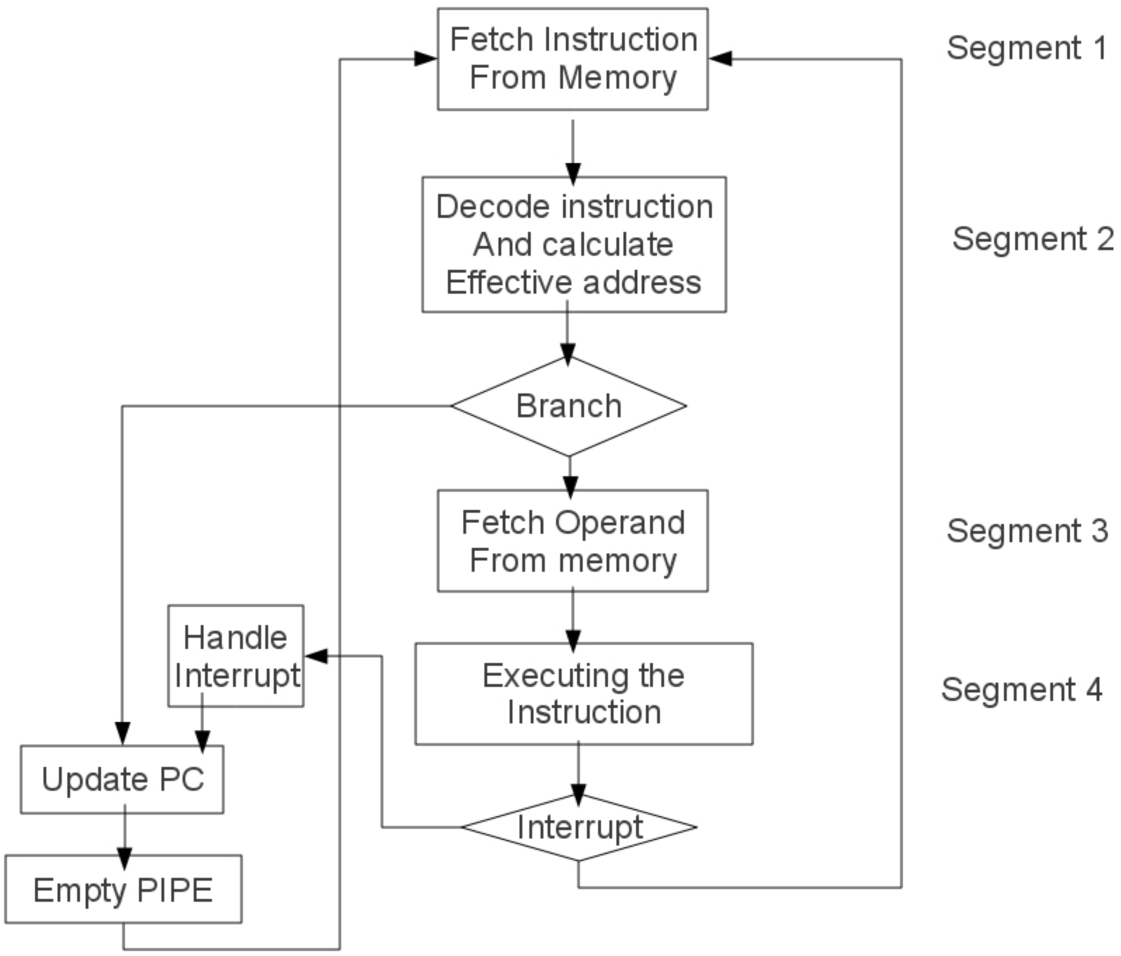
5. Instruction Pipeline:

Pipelining concept is not only limited to the data stream, but can also be applied on the instruction stream. The instruction pipeline execution will be like the queue execution. In the queue the data that is entered first, will be the data first retrieved. Therefore when an instruction is first coming, the instruction will be placed in the queue and will be executed in the system. Finally the results will be passing on to the next instruction in the queue. This scenario is called as Instruction pipelining. The instruction cycle is given below

- Fetch the instruction from the memory
- Decode the instruction
- calculate the effective address
- Fetch the operands from the memory
- Execute the instruction
- Store the result in the proper place.

In a computer system each and every instruction need not necessary to execute all the above phases. In a Register addressing mode, there is no need of the effective address calculation. Below is the example of the four segment instruction pipeline.

Instruction pipelining


Fetch Instruction From Memory
Segment 1
Decode instruction And calculate Effective address
Segment 2
Branch
Fetch Operand From memory
Segment 3
Handle Interrupt
Executing the Instruction
Segment 4
Update PC
Interrupt
Empty PIPE

In the above diagram we can see that the instruction which is first executing has to be fetched from the memory, there after we are decoding the instruction and we are calculating the effective address. Now we have two ways to execute the instruction. Suppose we are using a normal instruction like ADD, then the operands for that instruction will be fetched and the instruction will be executed. Suppose we are executing an instruction such as Fetch command. The fetch command itself has internally three more commands which are like ACTDR, ARTDR etc.., therefore we have to jump to that particular location to execute the command, so we are using the branch operation. So in a branch operation, again other instructions will be executed. That means we will be updating the PC value such that the instruction can be executed. Suppose we are fetching the operands to perform the original operation such as ADD, we need to fetch the data. The data can be fetched in two ways, either from the main memory or else from an input output devices. Therefore in order to use the input output devices, the devices must generate the interrupts which should be handled by the CPU. Therefore the handling of interrupts is also a kind of program execution. Therefore we again have to start from the starting of the program and execute the interrupt cycle.

Timing of Instruction Pipeline

FI → Fetch Instruction

DA → Decode instruction and Fetch Effective Address

FO → Fetch Operand

EX → Execute the Instruction

Step	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
3			FI	DA	FO	EX							
4				FI	-	-	FI	DA	FO	EX			
5					-	-	-	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

The different instruction cycles are given below:

• FI → FI is a segment that fetches an instruction

• DA → DA is a segment that decodes the instruction and identifies the effective address.

• FO → FO is a segment that fetches the operand.

• EX → EX is a segment that executes the instruction with the operand.

6. Interconnection Structures

- Various components of the multiprocessor system are CPU, IOP, and a memory unit.
- There are different physical configurations between interconnected components.
- The physical configuration depends on the number of transfer paths that are available.between the processors & memory in a shared memory system and among the processing elements in a loosely coupled system.

Various physical forms:

1. Time-shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube system

6.1 Time-shared common bus

- Consists of a number of processors connected through a common path to a memory unit.
- Part of the local memory may be designed as a *cache memory* attached to the CPU

Disadvantages:

- Only one processor can communicate with the memory or another processor at any given time.
- The total transfer rate within the system is limited by the speed of the single path

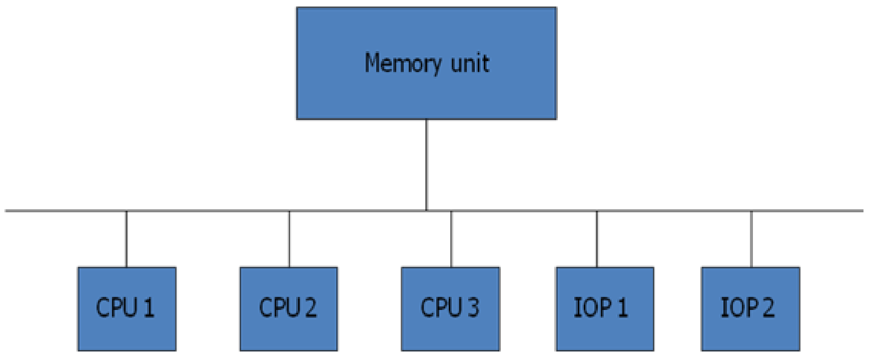


Fig: Time shared common bus organization

6.2 Multiport memory

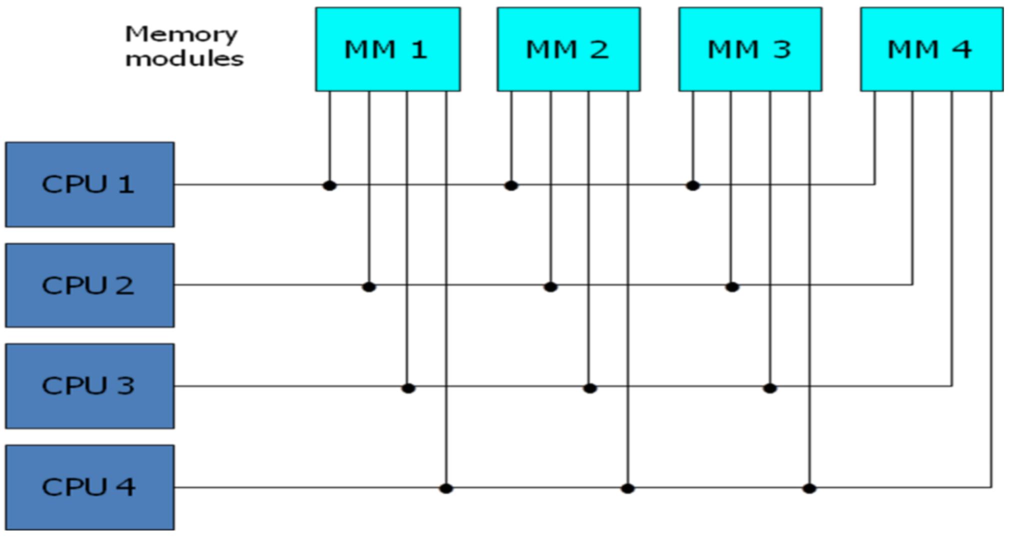


Fig: Multiport memory organization

- In the multiport memory system, different memory module and CPUs have separate buses.
- The module has internal control logic to determine port which will access to memory at any given time.
- Priorities are assigned to each memory port to resolve memory access conflicts.

Advantages:

Because of the multiple paths high transfer rate can be achieved.

Disadvantages:

It requires expensive memory control logic and a large number of cables and connections.

6.3 Crossbar switch

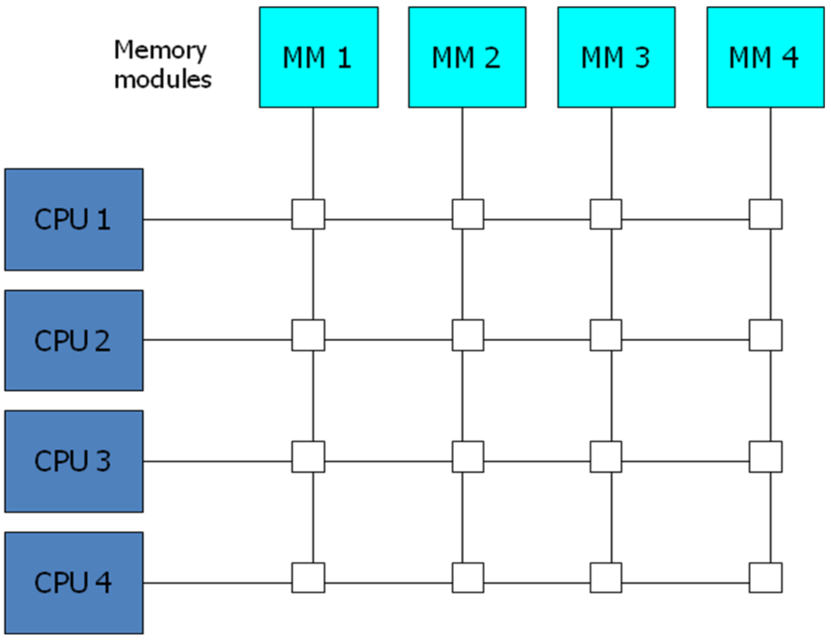


Fig: Crossbar Switch

- Consists of a various number of *crosspoints* that are present at intersections between processor buses and memory module paths.
- A switch determines the path from a processor to a memory module.

Advantages:

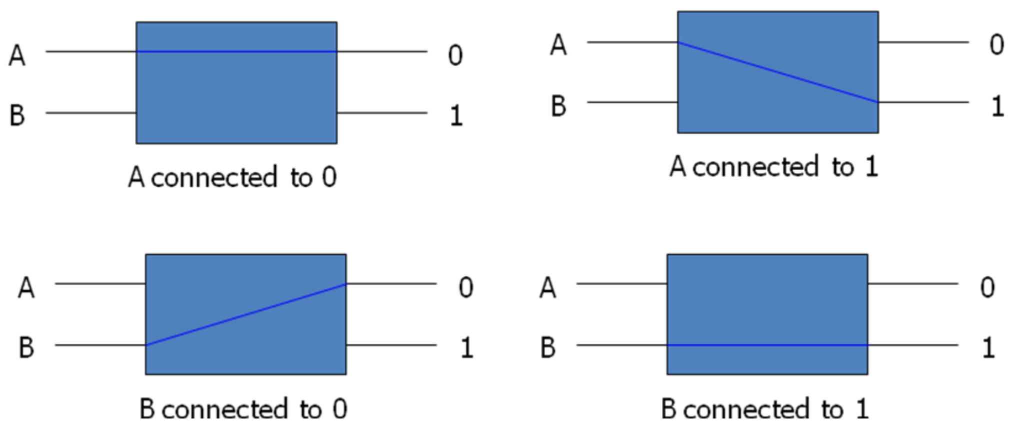
Supports simultaneous transfers from all memory modules

Disadvantages:

The hardware required to implement the switch can be very large and complex.

6.4 Multistage switching network

The basic components are a two-input, two-output interchange switch:



One topology is the omega switching network shown in fig:

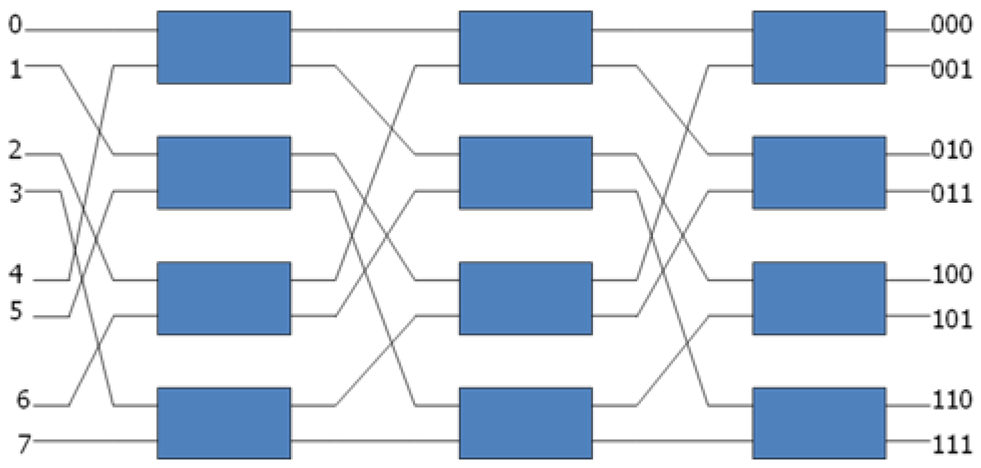


Fig: 8 x 8 Omega Switching Network

- Some request patterns cannot be connected simultaneously. i.e., any two sources cannot be connected simultaneously to destination 000 and 001
- The source is a processor and the destination is a memory module,in a tightly coupled multiprocessor system,

- Set up the path à transfer the address into memory à transfer the data
- Both the source and destination are processing elements,in a loosely coupled multiprocessor system,

6.5 Hypercube system

- The hypercube or binary n-cube multiprocessor structure is a loosely coupled system which is composed of N=2^n processors interconnected in an n-dimensional binary cube.
- Routing messages through an *n*-cube structure may take from one to *n* links from a source to a destination node.
- It consists of 128(here*n*=7) microcomputers, where each node consists of a CPU, a floating point processor, local memory, and serial communication interface units.

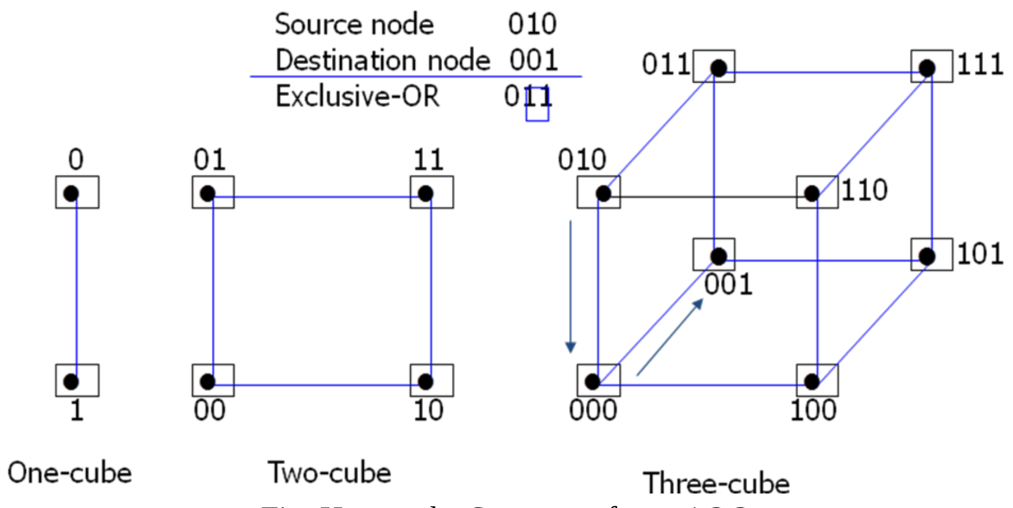


Fig: Hypercube Structures for n=1,2,3

7. **Inter-process communication**.

Inter-process communication (IPC) is a mechanism that allows the exchange of data between processes. By providing a user with a set of programming interfaces, IPC helps a programmer organize the activities among different processes. IPC allows one application to control another application, thereby enabling data sharing without interference.

IPC enables data communication by allowing processes to use segments, semaphores, and other methods to share memory and information. IPC facilitates efficient message transfer between processes. The idea of IPC is based on Task Control Architecture (TCA). It is a flexible technique that can send and receive variable length arrays, data structures, and lists. It has the capability of using publish/subscribe and client/server data-transfer paradigms while supporting a wide range of operating systems and languages.

The IPC mechanism can be classified into pipes, first in, first out (FIFO), and shared memory. Pipes were introduced in the UNIX operating system. In this mechanism, the data flow is unidirectional. A pipe can be imagined as a hose pipe in which the data enters through one end and flows out from the other end. A pipe is generally created by invoking the pipe system call, which in turn generates a pair of file descriptors. Descriptors are usually created to point to a pipe node. One of the main features of pipes is that the data flowing through a pipe is transient, which means data can be read from the read descriptor only once. If the data is written into the write descriptor, the data can be read only in the order in which the data was written.

The working principle of FIFO is very similar to that of pipes. The data flow in FIFO is unidirectional and is identified by access points. The difference between the two is that FIFO is identified by an access point, which is a file within the file system, whereas pipes are identified by an access point.

======================