COA Notes 4th semm - Computer organization and architecture

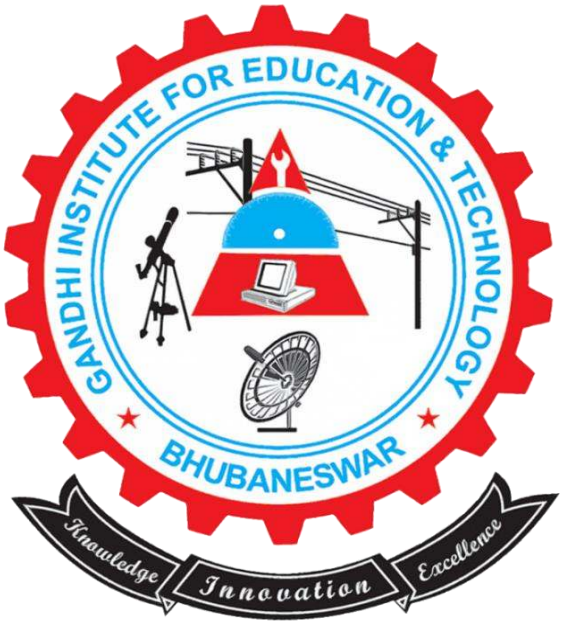
Computer science engineering (Gandhi Institute of Engineering and Technology University)

COMPUTER ORGANIZATION & ARCHITECTURE

DIGITAL NOTES

BY

Dona Chakraborty



BANIATANGI, BHUBNESWAR, KHORDHA – 752060

CONTENT

	10	software interrupts and exceptions
	11	Programs and processes-role of interrupts in process state transition
	12	I/O device interfaces – SCII, USB
4	**Pipelining**	
	1	Basic concepts of pipelining
	2	Throughput and speedup
	3	pipeline hazards
	4	Parallel Processors: Introduction to parallel processors
	5	Concurrent access to memory and cache coherency
	6	CPU Basics: Multiple CPUs
	7	Cores, and Hyper-Threading
	8	Introduction to Multiple-Processor Scheduling in Operating System.
5	**Memory organization**	
	1	Memory interleaving
	2	concept of hierarchical memory organization
	3	cache memory
	4	cache size vs. block size
	5	mapping function
	6	replacement algorithm
	7	write policies.

SYLLABUS

Module-I (08 Hrs.)

Functional blocks of a computer: CPU, memory, input-output subsystems, control unit. Instruction set architecture of a CPU-registers, instruction execution cycle, RTL interpretation of instructions, addressing modes, instruction execution cycle, RTL interpretation of instructions, addressing modes, instruction set. Case study – instruction sets of some common CPUs.

Module-II: (08 Hrs.)

Data representation: signed number representation, fixed and floating-point representations, character representation. Computer arithmetic – integer addition and subtraction, ripple carry adder, carry look-ahead adder, etc. multiplication – shift-and add, Booth multiplier, carry save multiplier, etc. Division restoring and non-restoring techniques, floating point arithmetic.

Module-III: (12 Hrs.)

Introduction to x86 architecture.

CPU control unit design: hardwired and micro-programmed design approaches, Case study – design of a simple hypothetical CPU.

Memory system design: semiconductor memory technologies, memory organization.

Peripheral devices and their characteristics: Input-Output subsystems, I/O device interface, I/O transfers=program controlled, interrupt driven and DMA, privileged and non-privileged instructions, software interrupts and exceptions, Programs and processes-role of interrupts in process state transitions, I/O device interfaces – SCII, USB

Module-IV: (07 Hrs.)

Pipelining: Basic concepts of pipelining, throughput and speedup, pipeline hazards.

Parallel Processors: Introduction to parallel processors, Concurrent access to memory and cache coherency CPU Basics: Multiple CPUs, Cores, and Hyper-Threading, Introduction to Multiple-Processor Scheduling in Operating System.

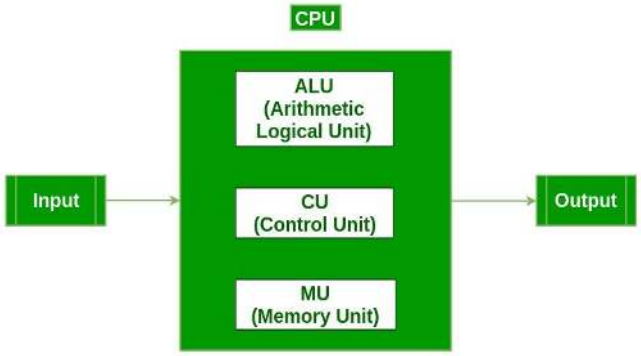
Module-V: (08 Hrs.)

Memory organization: Memory interleaving, concept of hierarchical memory organization, cache memory, cache size vs. block size, mapping function, replacement algorithm, write policies.

Module-I:

Functional blocks of a computer:

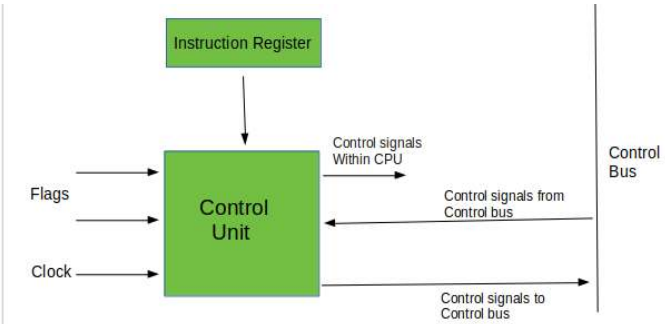
Computer: A computer is a combination of **hardware and software** resources which integrate together and provides various functionalities to the user. Hardware are the physical components of a computer like the processor, memory devices, monitor, keyboard etc. while software is the set of programs or instructions that are required by the hardware resources to function properly. There are a few basic components that aids the working-cycle of a computer i.e. the Input- Process- Output Cycle and these are called as the functional components of a computer. It needs certain input, processes that input and produces the desired output.



- **Input Unit :**The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.
- **Central Processing Unit (CPU) :** Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers
- **Arithmetic and Logic Unit (ALU) :** The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical

decisions involve comparison of two data items to see which one is larger or smaller or equal.

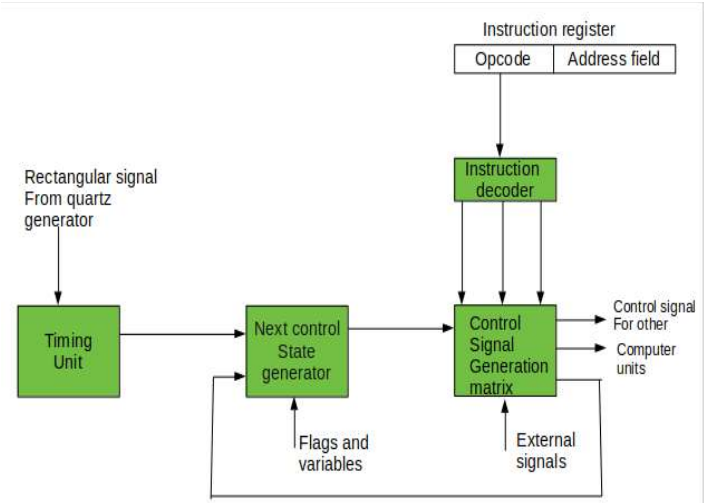
- **Control Unit :** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction, interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.



Block Diagram of the Control Unit

Types of Control Unit – There are two types of control units: Hardwired control unit and Microprogrammable control unit.

1. **Hardwired Control Unit –** In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we cannot modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode. As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for execution units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.

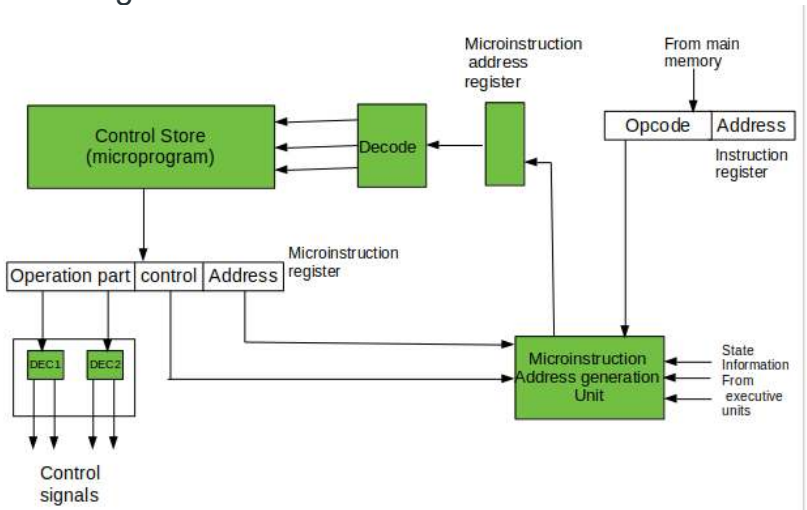


Block diagram of a hardwired control unit of a computer

1. Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit. A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state. This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle. The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction

that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

2. **Microprogrammable control unit –** The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution. In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.
   - **With a single-level control store:** In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator

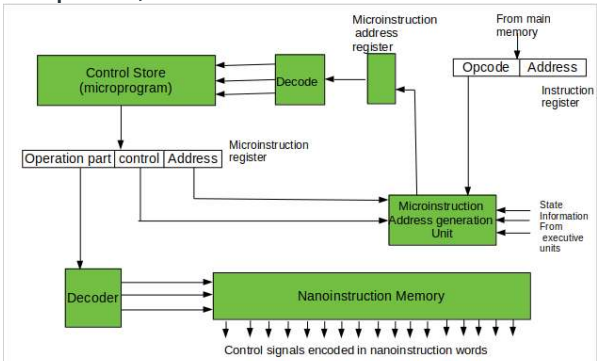


Microprogrammed control unit with a single level control store

The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the instruction of the given microprogram is the

microinstruction that fetches the next instruction from the main memory to the instruction register.

- **With a two-level control store:** In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions.



Microprogrammed control unit with a two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

Advantages of a well-designed Control Unit:

Efficient instruction execution: A well-designed control unit can execute instructions more efficiently by optimizing the instruction pipeline and minimizing the number of clock cycles required for each instruction.

Improved performance: A well-designed control unit can improve the performance of the CPU by increasing the clock speed, reducing the latency, and improving the throughput.

Support for complex instructions: A well-designed control unit can support complex instructions that require multiple operations, reducing the number of instructions required to execute a program.
Improved reliability: A well-designed control unit can improve the reliability of the CPU by detecting and correcting errors, such as memory errors and pipeline stalls.
Lower power consumption: A well-designed control unit can reduce power consumption by optimizing the use of resources, such as registers and memory, and reducing the number of clock cycles required for each instruction.

Disadvantages of a poorly-designed Control Unit:

Reduced performance: A poorly-designed control unit can reduce the performance of the CPU by introducing pipeline stalls, increasing the latency, and reducing the throughput.
Increased complexity: A poorly-designed control unit can increase the complexity of the CPU, making it harder to design, test, and maintain.
Higher power consumption: A poorly-designed control unit can increase power consumption by inefficiently using resources, such as registers and memory, and requiring more clock cycles for each instruction.
Reduced reliability: A poorly-designed control unit can reduce the reliability of the CPU by introducing errors, such as memory errors and pipeline stalls.
Limitations on instruction set: A poorly-designed control unit may limit the instruction set of the CPU, making it harder to execute complex instructions and limiting the functionality of the CPU.

- **Memory Registers :** A register is a temporary unit of memory in the CPU. These are used to store the data which is directly used by the processor. Registers can be of different sizes(16 bit, 32 bit, 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.

- **Memory :** Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. when a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as

RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM).

Types of computer memory

In general, memory can be divided into primary and secondary memory; moreover, there are numerous types of memory when discussing just primary memory. Some types of primary memory include the following

- **Cache memory:**This temporary storage area, known as a cache, is more readily available to the processor than the computer's main memory source. It is also called *CPU memory* because it is typically integrated directly into the CPU chip or placed on a separate chip with a bus interconnect with the CPU.
- **RAM:** The term is based on the fact that any storage location can be accessed directly by the processor.
- **Dynamic RAM:**DRAM is a type of semiconductor memory that is typically used by the data or program code needed by a computer processor to function.
- **Static RAM:** SRAM retains data bits in its memory for as long as power is supplied to it. Unlike DRAM, which stores bits in cells consisting of a capacitor and a transistor, SRAM does not have to be periodically refreshed.
- **Read-only memory:** ROM is a type of computer storage containing nonvolatile, permanent data that, normally, can only be read and not written to. ROM contains the programming that enables a computer to start up or regenerate each time it is turned on.
- **Programmable ROM:** PROM is ROM that can be modified once by a user. It enables a user to tailor a microcode program using a special machine called a *PROM programmer*.

- **Erasable PROM:** EPROM is programmable read-only memory PROM that can be erased and re-used. Erasure is caused by shining an intense ultraviolet light through a window designed into the memory chip.
- **Electrically erasable PROM:** EEPROM is a user-modifiable ROM that can be erased and reprogrammed repeatedly through the application of higher than normal electrical voltage. Unlike EPROM chips, EEPROMs do not need to be removed from the computer to be modified. However, an EEPROM chip must be erased and reprogrammed in its entirety, not selectively.
- **Virtual memory** : A memory management technique where secondary memory can be used as if it were a part of the main memory. Virtual memory uses hardware and software to enable a computer to compensate for physical memory shortages by temporarily transferring data from RAM to disk storage.

- **Output Unit :** The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, etc.

Instruction Set Architecture:

ISA describes the **design of a Computer** in terms of the **basic operations** it must support. The ISA is not concerned with the implementation-specific details of a computer. It is only concerned with the set or collection of basic operations the computer must support. For example, the AMD Athlon and the Core 2 Duo processors have entirely different implementations but they support more or less the same set of basic operations as defined in the x86 Instruction Set.

1. The ISA defines the **types of instructions** to be supported by the processor.
   Based on the type of operations they perform MIPS Instructions are classified into 3 types:
   - **Arithmetic/Logic Instructions:**
     These Instructions perform various Arithmetic & Logical operations on one or more operands.

- **Data Transfer Instructions:**
  These instructions are responsible for the transfer of instructions from memory to the processor registers and vice versa.
- **Branch and Jump Instructions:**
  These instructions are responsible for breaking the sequential flow of instructions and jumping to instructions at various other locations, this is necessary for the implementation of *functions* and *conditional statements*.

2. The ISA defines the **maximum length** of each type of instruction.
   Since the MIPS is a 32 bit ISA, each instruction must be accommodated within 32 bits.

3. The ISA defines the **Instruction Format** of each type of instruction.
   The **Instruction Format** determines how the entire instruction is encoded within 32 bits
   There are 3 types of Instruction Formats in the MIPS ISA:
   - R-Instruction Format
   - I-Instruction Format
   - J-Instruction Format

Registers: Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers. A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters). The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Following is the list of some of the most common registers used in a basic computer:

Register	Symbol	Number of bits	Function
Data register	DR	16	Holds memory operand
Address register	AR	12	Holds address for the memory

Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code
Program counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input register	INPR	8	Carries input character
Output register	OUTR	8	Carries output character

- The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- The Accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the Instruction register (IR).
- The Temporary Register (TR) is used for holding the temporary data during the processing.
- The Input Registers (IR) holds the input characters given by the user.
- The Output Registers (OR) holds the output after processing the input data.

Instruction Execution Cycle: The instruction cycle comprises three main stages and is also addressed as the fetch-decode-execute cycle or fetch-execute cycle because of the steps involved. The stages are as follows:

- **Fetch stage**
- **Decode stage**
- **Execute stage**

The following procedures are included in each instruction cycle:

- The CPU reads the effective address from memory if the instruction has an indirect address
- The register retrieves instructions from memory.
- It is capable of carrying out the command
- Decoding education is the primary task assigned

The loop continues carrying out the task repeatedly until a halt condition is met. Till then, the cycle keeps on restarting.

The code for Instruction Cycle determines which phase of the cycle it is in.

01: Indirect Cycle

10: Execute Cycle

11: Interrupt Cycle

00: Fetch Cycle

Cycle of Fetch

The address instruction to be executed in the order is stored on the program counter. The processor retrieves the education from the pointed memory. The register increments the PC and obtains the instruction's address. The instruction is then written to the instruction register. The processor reads the instruction and then executes it.

Cycle of Execution

The data transfer for implementation occurs in two ways, which are as follows:

The data is sent from the processor to memory or from memory to the processor. Processor-Input/ Output Data can be transferred to or from a peripheral device via a processor-I/O device transfer. The processor performs essential operations on the information during the execution cycle, and the control consistently requests a change in the data implementation sequence. These two methods are linked and complete the execution cycle.

RTL: (**Register Transfer Language)**

The Register Transfer Language **is the symbolic representation of notations used to specify the sequence of micro-operations.** In a computer system, data transfer takes place between processor registers and memory and between processor registers and

input-output systems. These data transfer can be represented by standard notations given below:

- Notations R0, R1, R2..., and so on represent processor registers.
- The addresses of memory locations are represented by names such as LOC, PLACE, MEM, etc.
- Input-output registers are represented by names such as DATA IN, DATA OUT and so on.
- The content of register or memory location is denoted by placing square brackets around the name of the register or memory location.

Instruction Set: Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

An instruction comprises of groups called fields. These fields include:

- The Operation code (Opcode) field which specifies the operation to be performed.
- The Address field which contains the location of the operand, i.e., register or memory location.
- The Mode field which specifies how the operand will be located.

A basic computer has three instruction code formats which are:

1. Memory - reference instruction
2. Register - reference instruction
3. Input-Output instruction

Memory - reference instruction:

(Opcode = 000 through 110)

In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

Register - reference instruction:

(Opcode = 111, I = 0)

The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction. A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register.

Addressing Modes:

Instructions that define the address of a definite memory location are known as memory reference instructions. The method in which a target address or effective address is recognized within the instruction is known as addressing mode.

The address field for instruction can be represented in two different ways are as follows −

- **Direct Addressing** − It uses the address of the operand.
- **Indirect Addressing** − It facilitates the address as a pointer to the operand.

The address of the operand or the target address is called the effective address.

Effective Address (EA) − It defines the address that can be executed as a target address for a branch type instruction or the address that can be used directly to create an operand for a computation type instruction, without creating any changes.

Opcodes

An opcode is a collection of bits that represents the basic operations including add, subtract, multiply, complement, and shift. The total number of operations provided

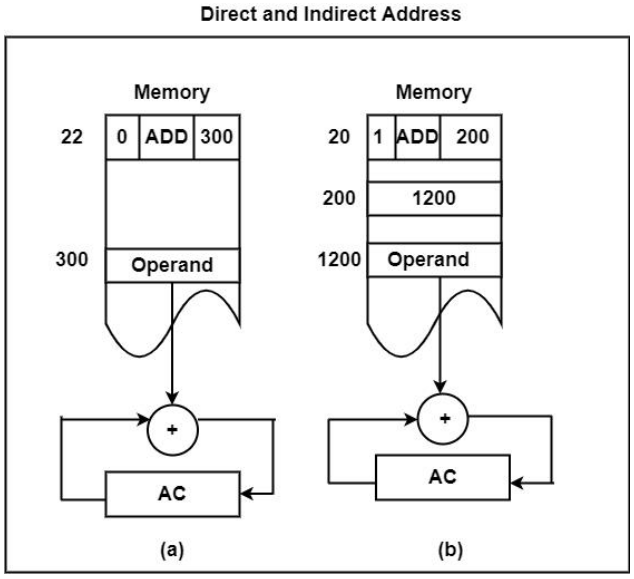
through the computer determines the number of bits needed for the opcode. The minimum bits accessible to the opcode should be n for 2n operations. These operations are implemented on information that is saved in processor registers or memory.

Address

The address is represented as the location where a specific instruction is constructed in the memory. The address bits of an instruction code is used as an operand and not as an address. In such methods, the instruction has an immediate operand. If the second part has an address, the instruction is referred to have a direct address.

There is another possibility in the second part including the address of the operand. This is referred to as an indirect address. In the instruction code, one bit can signify if the direct or indirect address is executed.

The figure shows a diagram showing direct and indirect addresses.



Instruction sets of some common CPUs: An instruction is a set of codes that the computer processor can understand. The code is usually in 1s and 0s, or machine language. It contains instructions or tasks that control the movement of bits and bytes within the processor.

Example of some instruction sets −

- **ADD** − Add two numbers together.
- **JUMP** − Jump to designated RAM address.
- **LOAD** − Load information from RAM to the CPU.

Types of Instruction Set:

Generally, there are two types of instruction set used in computers.

Reduced Instruction set Computer (RISC)

A number of computer designers recommended that computers use fewer instructions with simple constructs so that they can be executed much faster within the CPU without having to use memory as often. This type of computer is called a Reduced Instruction Set Computer.

The concept of RISC involves an attempt to reduce execution time by simplifying the instruction set of computers.

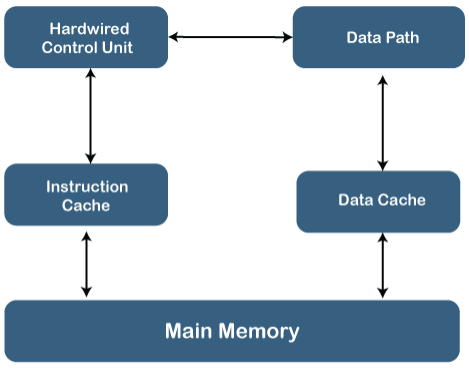
Characteristics of RISC

The characteristics of RISC are as follows −

- Relatively few instructions.
- Relatively few addressing modes.
- Memory access limited to load and store instructions.
- All operations done within the register of the CPU.
- Single-cycle instruction execution.
- Fixed length, easily decoded instruction format.
- Hardwired rather than micro ed control.

RISC Architecture

It is a highly customized set of instructions used in portable devices due to system reliability such as Apple iPod, mobiles/smartphones.



RISC Architecture

Complex Instruction Set Computer (CISC)

CISC is a computer where a single instruction can perform numerous low-level operations like a load from memory and a store from memory, etc. The CISC attempts

to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

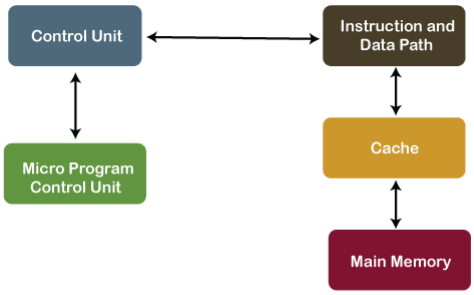
The design of an instruction set for a computer must take into consideration not only machine language constructs but also the requirements imposed on the use of high level programming languages.

The goal of CISC is to attempt to provide a single machine instruction for each statement that is written in a high level language.

Characteristics of CISC

The characteristics of CISC are as follows −

- A large number of instructions typically from 100 to 250 instructions.
- Some instructions that perform specialized tasks and are used infrequently.
- A large variety of addressing modes- typically from 5 to 20 different modes.
- Variable length instruction formats.
- Instructions that manipulate operands in memory.



CISC Architecture

.

RISC	CISC
It is a Reduced Instruction Set Computer.	It is a Complex Instruction Set Computer.
It emphasizes on software to optimize the instruction set.	It emphasizes on hardware to optimize the instruction set.
It is a hard wired unit of programming in the RISC Processor.	Microprogramming unit in CISC Processor.
It requires multiple register sets to store the instruction.	It requires a single register set to store the instruction.
RISC has simple decoding of instruction.	CISC has complex decoding of instruction.
Uses of the pipeline are simple in RISC.	Uses of the pipeline are difficult in CISC.

It uses a limited number of instruction that requires less time to execute the instructions.	It uses a large number of instruction that requires more time to execute the instructions.
It uses LOAD and STORE that are independent instructions in the register-to-register a program's interaction.	It uses LOAD and STORE instruction in the memory-to-memory interaction of a program.
RISC has more transistors on memory registers.	CISC has transistors to store complex instructions.
The execution time of RISC is very short.	The execution time of CISC is longer.
RISC architecture can be used with high-end applications like telecommunication, image processing, video processing, etc.	CISC architecture can be used with low-end applications like home automation, security system, etc.
It has fixed format instruction.	It has variable format instruction.
The program written for RISC architecture needs to take more space in memory.	Program written for CISC architecture tends to take less space in memory.
Example of RISC: ARM, PA-RISC, Power Architecture, Alpha, AVR, ARC and the SPARC.	Examples of CISC: VAX, Motorola 68000 family, System/360, AMD and the Intel x86 CPUs.

Module-II:

Data representation:

Signed number representation: A signed integer is an integer with a positive '+' or negative sign '-' associated with it. Since the computer only understands binary, it is necessary to represent these signed integers in binary form.
In binary, signed Integer can be represented in three ways:

1. Signed bit.
2. 1's Complement.
3. 2's Complement.

Signed bit Representation: In the signed integer representation method the following rules are followed:

1. The MSB (Most Significant Bit) represents the sign of the Integer.
2. Magnitude is represented by other bits other than MSB i.e. (n-1) bits where n is the no. of bits.
3. If the number is positive, MSB is 0 else 1.
4. The range of signed integer representation of an n-bit number is given as –(2^{n-1}-1) to (2)^{n-1}-1.

1's Complement representation of a signed integer:
In 1's complement representation the following rules are used:

1. For +ve numbers the representation rules are the same as signed integer representation.
2. For –ve numbers, we can follow any one of the two approaches:

- Write the +ve number in binary and take 1's complement of it.
- Write Unsigned representation of 2^n-1-X for –X.

2's Complement representation:
In 2's Complement representation the following rules are used:

1. For +ve numbers, the representation rules are the same as signed integer representation.
2. For –ve numbers, there are two different ways we can represent the number.

- Write an unsigned representation of 2^n-X for –X in n-bit representation.
- Write a representation of +X and take 2's Complement.

Fixed and Floating Point Representations:

Fixed-Point Representation −

This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field.

We can represent these numbers using:

- Signed representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 1's complement representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 2's complementation representation: range from $-(2^{(k-1)})$ to $(2^{(k-1)}-1)$, for k bits.

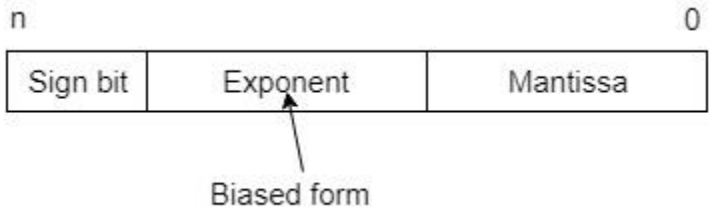
2's complementation representation is preferred in computer system because of unambiguous property and easier for arithmetic operations.

Floating-Point Representation −

This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two part: the first part represents a signed fixed point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be fraction or an integer. Floating -point is always interpreted to represent a number in the following form: Mxr^e.

Only the mantissa m and the exponent e are physically represented in the register (including their sign). A floating-point binary number is represented in a similar manner except that is uses base 2 for the exponent. A floating-point number is said to be normalized if the most significant digit of the mantissa is 1.



So, actual number is $(-1)^s(1+m)x2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number.

Note that signed integers and exponent are represented by either sign representation, or one's complement representation, or two's complement representation.

The floating point representation is more flexible. Any non-zero number can be represented in the normalized form of $\pm(1.b_1b_2b_3 \ldots)_2 x 2^n$ This is normalized form of a number x.

Character Representation:

Computers work in binary. As a result, all characters, whether they are letters, punctuation or digits, are stored as binary numbers. All of the characters that a computer can use are called a character set.

Two standards are in common use:

- American Standard Code for Information Interchange (ASCII)
- Unicode

ASCII uses seven bits, giving a character set of 128 characters. The characters are represented in a table called the ASCII table. The 128 characters include:

- 32 control codes (mainly to do with printing)
- 32 punctuation codes, symbols, and space
- 26 upper-case letters
- 26 lower-case letters
- numeric digits 0-9

We tend to say that the letter 'A' is the first letter of the alphabet, 'B' is the second and so on, all the way up to 'Z', which is the 26th letter. In ASCII, each character has its own assigned number.

Denary, binary and hexadecimal representations of ASCII characters

Character	Denary	Binary	Hexadecimal
A	65	1000001	41
Z	90	1011010	5A
a	97	1100001	61

Denary, binary and hexadecimal representations of ASCII characters

Character	Denary	Binary	Hexadecimal
z	122	1111010	7A
0	48	0110000	30
9	57	0111001	39
Space	32	0100000	20
!	33	0100001	21

'A' is represented by the denary number 65 (binary 1000001,hexadecimal 41), 'B' by 66 (binary 1000010, hexadecimal 42) and so on up to 'Z', which is represented by the denary number 90 (binary 1011010, hexadecimal 5A). Similarly, lower-case letters start at denary 97 (binary 1100001, hexadecimal 61) and end at denary 122 (binary 1111010, hexadecimal 7A).

When data is stored or transmitted, its ASCII or Unicode number is used, not the character itself.

For example, the word "Computer" would be represented as:

1000011 1101111 1101101 1110000 1110101 1110100 1100101 1110010

Computer arithmetic –

Integer addition and subtraction: There are eight conditions to consider while adding or subtracting signed numbers. These conditions are based on the operations implemented and the sign of the numbers.

The table displays the algorithm for addition and subtraction. The first column in the table displays these conditions. The other columns of the table define the actual operations to be implemented with the magnitude of numbers. The last column of the table is needed to avoid a negative zero. This defines that when two same numbers are subtracted, the output must not be - 0. It should consistently be +0.

In the table, the magnitude of the two numbers is defined by P and Q.

Addition and Subtraction of Signed Magnitude Numbers

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

As display in the table, the addition algorithm states that −

- When the signs of P and Q are equal, add the two magnitudes and connect the sign of P to the output.
- When the signs of P and Q are different, compare the magnitudes and subtract the smaller number from the greater number.
- The signs of the output have to be equal as P in case P > Q or the complement of the sign of P in case P < Q.
- When the two magnitudes are equal, subtract Q from P and modify the sign of the output to positive.

The subtraction algorithm states that −

- When the signs of P and Q are different, add the two magnitudes and connect the signs of P to the output.
- When the signs of P and Q are the same, compare the magnitudes and subtract the smaller number from the greater number.
- The signs of the output have to be equal as P in case P > Q or the complement of the sign of P in case P < Q.
- When the two magnitudes are equal, subtract Q from P and modify the sign of the output to positive.

Ripple Carry Adder: A structure of multiple full adders is cascaded in a manner to gives the results of the addition of an n bit binary sequence. This adder includes cascaded full adders in its structure so, the carry will be generated at every full adder stage in a ripple-carry adder circuit. These carry output at each full adder stage is forwarded to its next full adder and there applied as a carry input to it. This process continues up to its last full adder stage. So, each carry output bit is rippled to the next stage of a full adder. By this reason, it is named as “RIPPLE CARRY ADDER”. The

most important feature of it is to add the input bit sequences whether the sequence is 4 bit or 5 bit or any.

“One of the most important point to be considered in this carry adder is the final output is known only after the carry outputs are generated by each full adder stage and forwarded to its next stage. So there will be a delay to get the result with using of this carry adder”.

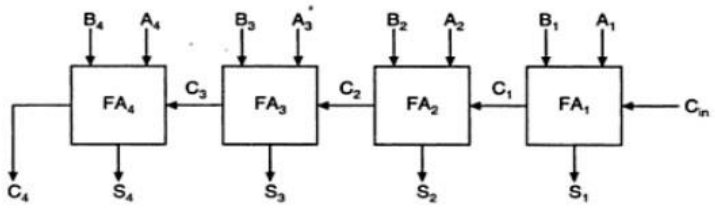
There are various types in ripple-carry adders. They are:

- 4-bit ripple-carry adder
- 8-bit ripple-carry adder
- 16-bit ripple-carry adder

First, we will start with 4-bit ripple-carry-adder and then 8 bit and 16-bit ripple-carry adders.

4-bit Ripple Carry Adder

The below diagram represents the 4-bit ripple-carry adder. In this adder, four full adders are connected in cascade. Co is the carry input bit and it is zero always. When this input carry ‘Co’ is applied to the two input sequences A1 A2 A3 A4 and B1 B2 B3 B4 then output represented with S1 S2 S3 S4 and output carry C4.

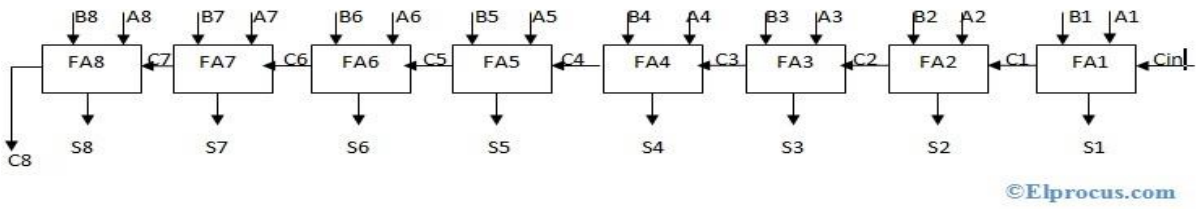


Working of 4-bit Ripple Carry Adder

- Let's take an example of two input sequences 0101 and 1010. These are representing the A4 A3 A2 A1 and B4 B3 B2 B1.
- As per this adder concept, input carry is 0.
- When Ao & Bo are applied at 1st full adder along with input carry 0.
- Here A1 =1 ; B1=0 ; Cin=0
- Sum (S1) and carry (C1) will be generated as per the Sum and Carry equations of this adder. As per its theory, the output equation for the Sum = A1⊕B1⊕Cin and Carry = A1B1⊕B1Cin⊕CinA1
- As per this equation, for 1st full adder S1 =1 and Carry output i.e., C1=0.
- Same like for next input bits A2 and B2, output S2 = 1 and C2 = 0. Here the important point is the second stage full adder gets input carry i.e., C1 which is the output carry of initial stage full adder.
- Like this will get the final output sequence (S4 S3 S2 S1) = (1 1 1 1) and Output carry C4 = 0
- This is the addition process for 4-bit input sequences when it's applied to this carry adder.

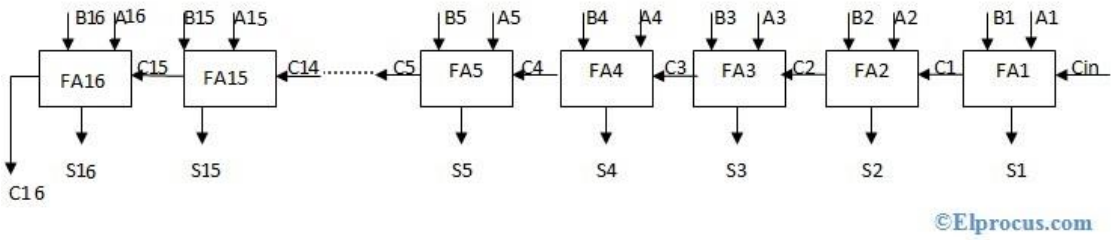
8-bit Ripple Carry Adder

- It consists of 8 full adders which are connected in cascaded form.
- Each full adder carry output is connected as an input carry to the next stage full adder.
- The input sequences are denoted by (A1 A2 A3 A4 A5 A6 A7 A8) and (B1 B2 B3 B4 B5 B6 B7 B8) and its relevant output sequence is denoted by (S1 S2 S3 S4 S5 S6 S7 S8).
- The addition process in an 8-bit ripple-carry-adder is the same principle which is used in a 4-bit ripple-carry-adder i.e., each bit from two input sequences are going to added along with input carry.
- This will use when the addition of two 8 bit binary digits sequence.



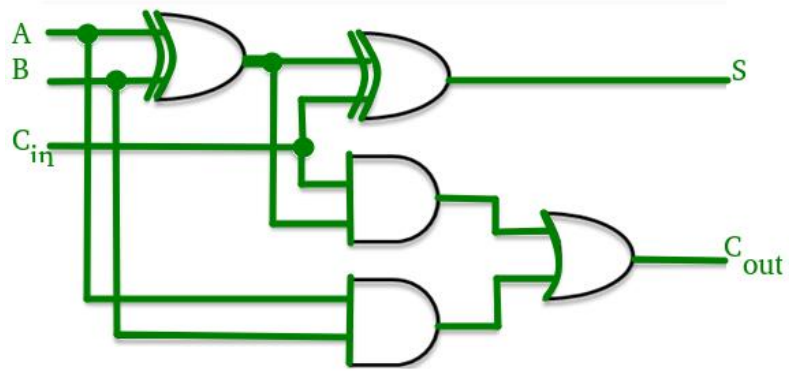
16-bit Ripple Carry Adder

- It consists of 16 full adders which are connected in cascaded form.
- Each full adder carry output is connected as an input carry to the next stage full adder.
- The input sequences are denoted by (A1 ….. A16) and (B1 …… B16) and its relevant output sequence is denoted by (S1 …….. S16).
- The addition process in a 16-bit ripple-carry-adder is the same principle which is used in a 4-bit ripple-carry adder i.e., each bit from two input sequences are going to add along with input carry.
- This will use when the addition of two 16 bit binary digits sequence.



Carry Look-Ahead Adder:

One widely used approach is to employ a carry look-ahead which solves this problem by calculating the carry signals in advance, based on the input signals. This type of adder circuit is called a carry look-ahead adder.
A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic. Let us discuss the design in detail.



A	B	C	C +1	Condition
0	0	0	0	No Carry Generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No Carry Propogate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry Generate
1	1	1	1	

Advantages and Disadvantages of Carry Look-Ahead Adder :
Advantages –

- The propagation delay is reduced.
- It provides the fastest addition logic.

Disadvantages –

- The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
- The circuit is costlier as it involves more number of hardware.

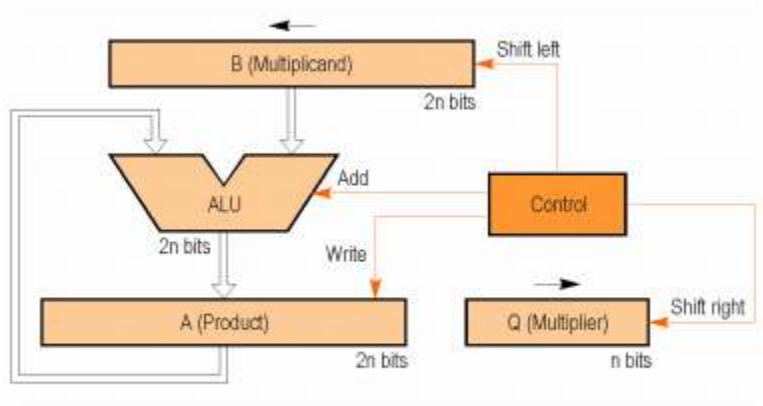
Multiplication–shift and Add:

- Shift-and-add multiplication is similar to the multiplication performed by paper and pencil. This method adds the multiplicand X to itself Y times, where Y denotes the multiplier. To multiply two numbers by paper and pencil, the algorithm is to take the digits of the multiplier one at a time from right to left, multiplying the multiplicand by a single digit of the multiplier and placing the intermediate product in the appropriate positions to the left of the earlier results.
- As an example, consider the multiplication of two unsigned 4-bit numbers, 8 (1000) and 9 (1001).

```
Multiplicand          1000 x
Multiplier            1001
                      1000
                     0000
                    0000
                   1000
Product            1001000
```

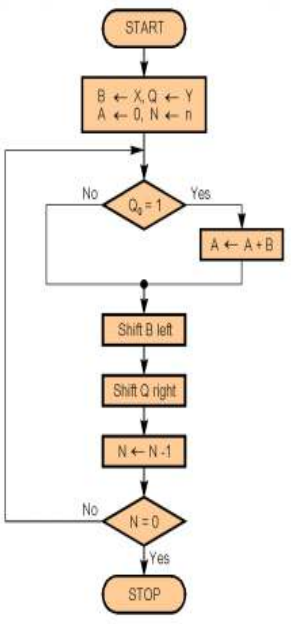
In the case of binary multiplication, since the digits are 0 and 1, each step of the multiplication is simple. If the multiplier digit is 1, a copy of the multiplicand (1 × multiplicand) is placed in the proper positions; if the multiplier digit is 0, a number of 0 digits (0 × multiplicand) are placed in the proper positions.

- Consider the multiplication of positive numbers. The first version of the multiplier circuit, which implements the shift-and-add multiplication method for two n-bit numbers, is shown in Figure



First version of the multiplier circuit.

- The 2n-bit product register (A) is initialized to 0. Since the basic algorithm shifts the multiplicand register (B) left one position each step to align the multiplicand with the sum being accumulated in the product register, we use a 2n-bit multiplicand register with the multiplicand placed in the right half of the register and with 0 in the left half.

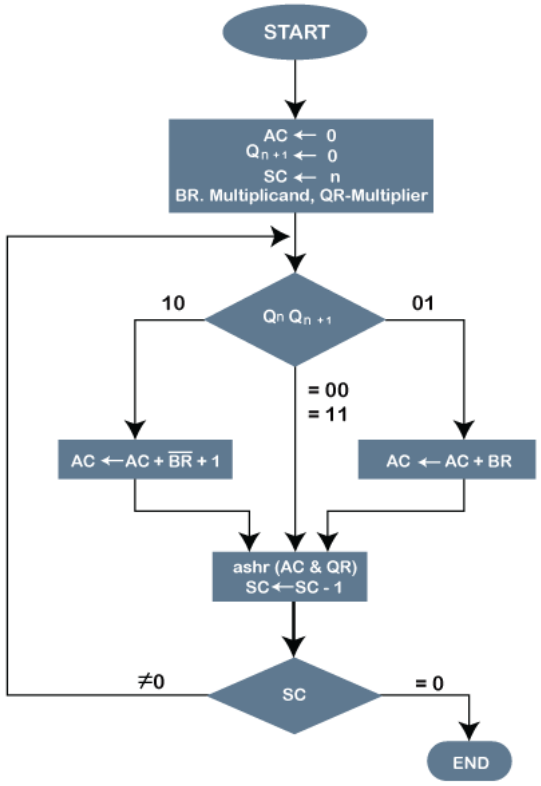


The first version of the multiplication algorithm.

- Figure shows the basic steps needed for the multiplication. The algorithm starts by loading the multiplicand into the B register, loading the multiplier into the Q register, and initializing the A register to 0. The counter N is initialized to n. The least significant bit of the multiplier register (Q0) determines whether the multiplicand is added to the product register. The left shift of the multiplicand has the effect of shifting the intermediate products to the left, just as when multiplying by paper and pencil. The right shift of the multiplier prepares the next bit of the multiplier to examine in the following iteration.

Booth's multiplication:

The booth algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement, respectively. It is also used to speed up the performance of the multiplication process. It is very efficient too. It works on the string bits 0's in the multiplier that requires no additional bit only shift the right-most string bits and a string of 1's in a multiplier bit weight 2^k to weight 2^m that can be considered as $\mathbf{2^{k+1} - 2^m}$.



In the above flowchart, initially, **AC** and $\mathbf{Q_{n+1}}$ bits are set to 0, and the **SC** is a sequence counter that represents the total bits set **n,** which is equal to the number of bits in the multiplier. There are **BR** that represent the **multiplicand bits,** and QR represents the **multiplier bits**. After that,

we encountered two bits of the multiplier as Q_n and Q_{n+1}, where Qn represents the last bit of QR, and Q_{n+1} represents the incremented bit of Qn by 1. Suppose two bits of the multiplier is equal to 10; it means that we have to subtract the multiplier from the partial product in the accumulator AC and then perform the arithmetic shift operation (ashr). If the two of the multipliers equal to 01, it means we need to perform the addition of the multiplicand to the partial product in accumulator AC and then perform the arithmetic shift operation (**ashr**), including $\mathbf{Q_{n+1}}$. The arithmetic shift operation is used in Booth's algorithm to shift AC and QR bits to the right by one and remains the sign bit in AC unchanged. And the sequence counter is continuously decremented till the computational loop is repeated, equal to the number of bits (n).

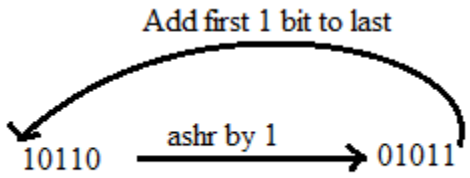
Working on the Booth Algorithm

1. Set the Multiplicand and Multiplier binary bits as M and Q, respectively.
2. Initially, we set the AC and Q_{n+1} registers value to 0.
3. SC represents the number of Multiplier bits (Q), and it is a sequence counter that is continuously decremented till equal to the number of bits (n) or reached to 0.
4. A Qn represents the last bit of the Q, and the Q_{n+1} shows the incremented bit of Qn by 1.
5. On each cycle of the booth algorithm, Q_n and Q_{n+1} bits will be checked on the following parameters as follows:
    i. When two bits Q_n and Q_{n+1} are 00 or 11, we simply perform the arithmetic shift right operation (ashr) to the partial product AC. And the bits of Qn and Q_{n+1} is incremented by 1 bit.
    ii. If the bits of Q_n and Q_{n+1} is shows to 01, the multiplicand bits (M) will be added to the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
    iii. If the bits of Q_n and Q_{n+1} is shows to 10, the multiplicand bits (M) will be subtracted from the AC (Accumulator register). After that, we perform the right shift operation to the AC and QR bits by 1.
6. The operation continuously works till we reached n - 1 bit in the booth algorithm.
7. Results of the Multiplication binary bits will be stored in the AC and QR registers.

There are two methods used in Booth's Algorithm:

1. RSC (Right Shift Circular)

It shifts the right-most bit of the binary number, and then it is added to the beginning of the binary bits.



2. RSA (Right Shift Arithmetic)

It adds the two binary bits and then shift the result to the right by 1-bit position.

Example: 0100 + 0110 => 1010, after adding the binary number shift each bit by 1 to the right and put the first bit of resultant to the beginning of the new bit.

Example: Multiply the two numbers 7 and 3 by using the Booth's multiplication algorithm.

Ans. Here we have two numbers, 7 and 3. First of all, we need to convert 7 and 3 into binary numbers like 7 = (0111) and 3 = (0011). Now set 7 (in binary 0111) as multiplicand (M) and 3 (in binary 0011) as a multiplier (Q). And SC (Sequence Count) represents the number of bits, and here we have 4 bits, so set the SC = 4. Also, it shows the number of iteration cycles of the booth's algorithms and then cycles run SC = SC - 1 time.

Q_n	Q_{n+1}	**M = (0111)** **M' + 1 = (1001) & Operation**	**AC**	**Q**	Q_{n+1}	**SC**
1	0	Initial	0000	0011	0	4
		Subtract (M' + 1)	1001			
			1001			

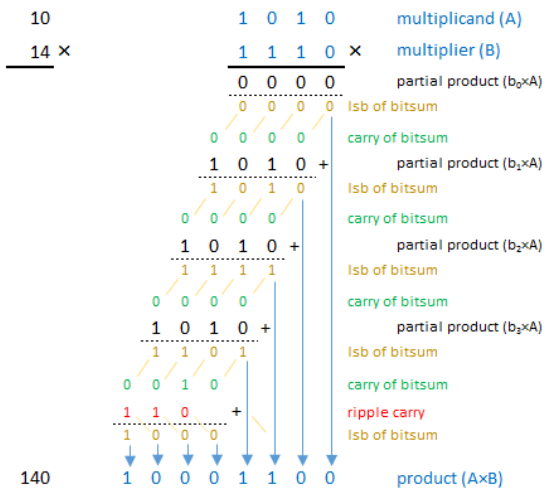
		Perform Arithmetic Right Shift operations (ashr)	1100	1001	1	3
1	1	Perform Arithmetic Right Shift operations (ashr)	1110	0100	1	2
0	**1**	Addition (A + M)	0111			
			0101	0100		
		Perform Arithmetic right shift operation	0010	1010	0	1
0	0	Perform Arithmetic right shift operation	**0001**	**0101**	0	0

The numerical example of the Booth's Multiplication Algorithm is 7 x 3 = 21 and the binary representation of 21 is 10101. Here, we get the resultant in binary 00010101. Now we convert it into decimal, as $(000010101)_{10}$ = 2*4 + 2*3 + 2*2 + 2*1 + 2*0 => 21.

Carry Save Multiplier:

An important advance in improving the speed of multipliers, pioneered by Wallace, is the use of carry save adders (CSA). Even though the building block is still the multiplying adder (`ma`), the topology of prevents a ripple carry by ensuring that, wherever possible, the carry-out signal propagates downward and not sideways.

The illustration below gives an example of this multiplication process.

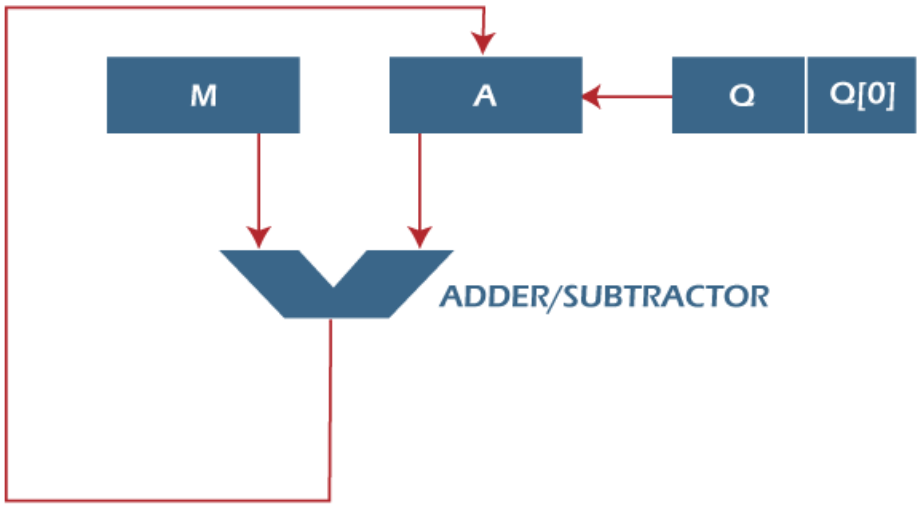


Inside a carry-save array multiplier

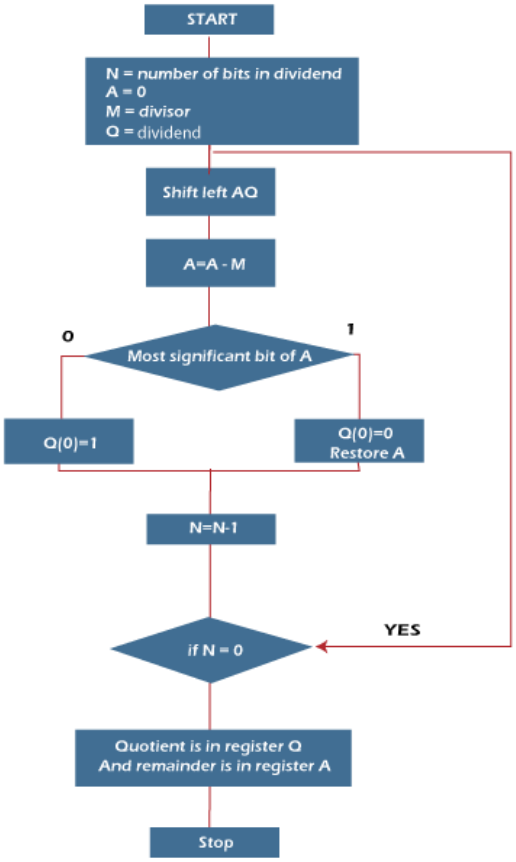
However, the topology is so that the carry-out from one adder is not connected to the carry-in of the next adder. Hence preventing a ripple carry.

Division Restoring Techniques:

In this section, we are going to perform restoring algorithm with the help of an unsigned integer. We are using restoring term because we know that the value of register A will be restored after each iteration. We will also try to solve this problem using the flow chart and apply bit operations.



Here, register Q is used to contain the quotient, and register A is used to contain the remainder. Here, the divisor will be loaded into the register M, and n-bit divided will be loaded into the register Q. 0 is the starting value of a register. The values of these types of registers are restored at the time of iteration. That's why it is known as restoring.



Now we will learn some steps of restoring division algorithm, which is described as follows:

Step 1: In this step, the corresponding value will be initialized to the registers, i.e., register A will contain value 0, register M will contain Divisor, register Q will contain Dividend, and N is used to specify the number of bits in dividend.

Step 2: In this step, register A and register Q will be treated as a single unit, and the value of both the registers will be shifted left.

Step 3: After that, the value of register M will be subtracted from register A. The result of subtraction will be stored in register A.

Step 4: Now, check the most significant bit of register A. If this bit of register A is 0, then the least significant bit of register Q will be set with a value 1. If the most significant bit of A is 1, then the least significant bit of register Q will be set to with value 0, and restore the value of A that means it will restore the value of register A before subtraction with M.

Step 5: After that, the value of N will be decremented. Here n is used as a counter.

Step 6: Now, if the value of N is 0, we will break the loop. Otherwise, we have to again go to step 2.

Step 7: This is the last step. In this step, the quotient is contained in the register Q, and the remainder is contained in register A.

For example:

In this example, we will perform a Division restoring algorithm.

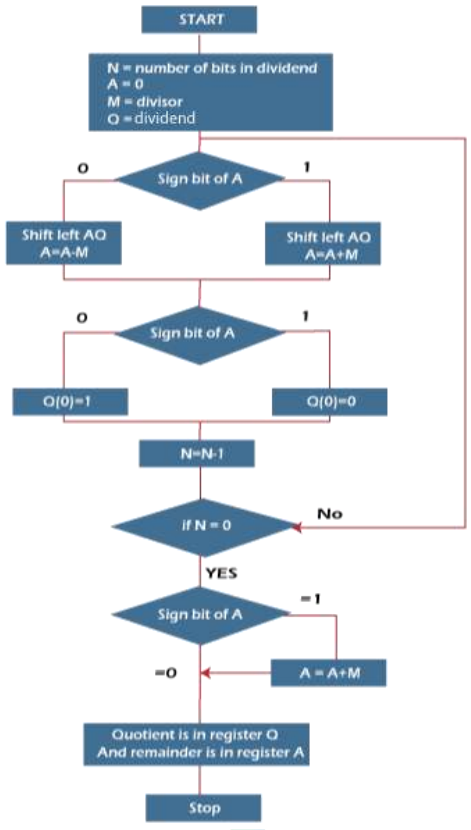
1. Dividend = 11
2. Divisor = 3

N	M	A	Q	Operation
4	00011	00000	1011	Initialize
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A = A - M
	00011	00001	0110	Q[0] = 0 And restore A
3	00011	00010	110_	Shift left AQ
	00011	11111	110_	A = A - M
	00011	00010	1100	Q[0] = 0
2	00011	00101	100_	Shift left AQ
	00011	00010	100_	A = A - M

	00011	00010	1001	Q[0] = 1
1	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
	00011	00010	0011	Q[0] = 1

Division non-restoring techniques:

The non-restoring division algorithm is more complex as compared to the restoring division algorithm. But when we implement this algorithm in hardware, it has an advantage, i.e., it contains only one decision and addition/subtraction per quotient bit. After performing the subtraction operation, there will not be any restoring steps. Due to this, the numbers of operations basically cut down up to half. Because of the less operation, the execution of this algorithm will be fast. This algorithm basically performs simple operations such as addition, subtraction. In this method, we will use the sign bit of register A. 0 is the starting value/bit of register A.



Now we will learn steps of the non-restoring division algorithm, which are described as follows:

Step 1: In this step, the corresponding value will be initialized to the registers, i.e., register A will contain value 0, register M will contain Divisor, register Q will contain Dividend, and N is used to specify the number of bits in dividend.

Step 2: In this step, we will check the sign bit of A.

Step 3: If this bit of register A is 1, then shift the value of AQ through left, and perform A = A + M. If this bit is 0, then shift the value of AQ into left and perform A = A - M. That means in case of 0, the 2's complement of M is added into register A, and the result is stored into A.

Step 4: Now, we will check the sign bit of A again.

Step 5: If this bit of register A is 1, then Q[0] will become 0. If this bit is 0, then Q[0] will become 1. Here Q[0] indicates the least significant bit of Q.

Step 6: After that, the value of N will be decremented. Here N is used as a counter.

Step 7: If the value of N = 0, then we will go to the next step. Otherwise, we have to again go to step 2.

Step 8: We will perform A = A + M if the sign bit of register A is 1.

Step 9: This is the last step. In this step, register A contains the remainder, and register Q contains the quotient.

For example:

In this example, we will perform a Non-Restoring Division algorithm with the help of an Unsigned integer.

1. Dividend = 11
2. Divisor = 3
3. -M = 11101

N	M	A	Q	Action

4	00011	00000	1011	Begin
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A = A - M
3	00011	11110	0110	Q[0] = 0
	00011	11100	110_	Shift left AQ
	00011	11111	110_	A = A + M
2	00011	11111	1100	Q[0] = 0
	00011	11111	100_	Shift left AQ
	00011	00010	100_	A = A +M
1	00011	00010	1001	Q[0] = 1
	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
0	00011	00010	0011	Q[0] = 1

So, register A contains the remainder 2, and register Q contains the quotient 3.

Module III:

Introduction to x86 architecture:

x86 is an Intel CPU architecture used to denote the microprocessor family released after the original 8086 processor. It was originated with the 16-bit 8086 processor in 1978, which denotes the microprocessor family on the basis of the Intel 8086 and 8088 microprocessors. Generally, X86 is the term for Intel processors that comprises of 286, 386, 486, and 586 processors. In modern times, the term "x86" is used to denote to any 32-bit processor that ensures backward compatibility for x86 instruction set architectures. Since the full name of the processors is 80286, 80386, 80486, and 80586; therefore, the term x86 is short for 80x86. Typically, the "80" is used to avoid redundancy.

Some of the highlights of the evolution of x86 architecture are:

1. **8080 –** It was the world's first general-purpose microprocessor. It was an 8-bit machine, with an 8-bit data path to memory. It was used in the first personal computer.
2. **8086 –** It was a 16-bit machine and was far more powerful than the previous one. It had a wider data path of 16-bits and larger registers along with an instruction cache or queue that prefetches a few instructions before they are executed. It is the first appearance of 8086 architecture. It has a real mode and an addressable memory of 1 MB.
3. **80286 –** It has an addressable memory of 16 MB instead of just 1 MB and contains two modes-real mode and first-generation 16-bit protected mode. It has a data transfer width of 16-bits and a programming model of 16-bits (16-bits general purpose registers and 16-bit addressing).
4. **80386 –** It was Intel's first 32-bit machine. Due to its 32-bit architecture, it was able to compete against the complexity and power of microcomputers and mainframes introduced just a few years earlier. It was the first processor to support multitasking and contained the 32-bit protected mode. It also implemented the concept of paging (permitted 32-bit virtual memory address to be translated into 32-bit physical memory address). It has an addressable physical memory of 4 GB and a data transfer width of 32 bits.
5. **80486 –** It introduced the concept of cache technology and instruction pipelining. It contained a write protect feature and offered a built-in math co-processor that offloaded complex math operations from the main CPU.
6. **Pentium –** The use of superscalar techniques was introduced as multiple instructions started executing in parallel. The page size extension (PSE) feature was added as a minor enhancement in paging.
7. **Pentium Pro –** It used register renaming, branch prediction, data flow analysis, speculative execution, and more pipeline stages. Advanced optimization techniques in microcode were also added along with level 2 cache. It implemented the second-generation address translation in which a 32-bit virtual address is translated into a 36-bit physical memory address.

8. **Pentium II –** It was able to process video, audio, and graphics data efficiently by incorporating Intel MMX technology (multimedia data set).
9. **Pentium III –** It contains SMD (streaming extensions) instructions (SSE) and supports 3D graphics software. It has a maximum CPU clock rate of 1.4 GHz and contained 70 new instructions.
10. **Pentium 4 –** It implements third-generation address translation that translates a 48-bit virtual memory address to a 48-bit physical memory address. It contains other floating point enhancements for multimedia.
11. **Core –** It is the first Intel microprocessor with dual-core which is the implementation of 2 processors on a single chip. There is an addition of Visualizing Technology.
12. **Core 2 –** It extends the architecture to 64 bits and core 2 Quad provides four processors on a single chip. The register set, as well as addressing modes, are 64 bits.

Comparison of major features of X-86 Family:

Microprocessor	**8086**	**80286**	**80386**	**80486**	**Pentium**
The data bus (bits)	16	16	32	32	64
Address bus (bits)	20	24	32	32	32
Operating Speed MHz	5 – 10	6 – 20	16 – 33	25- 50	50 – 100
Memory Capacity	1 MB	16 MB	4 GB	4 GB	4 GB
Memory Management	External	External	External	Internal	Internal
PC Type (IBM)	PC – XT	PC – AT	PC – AT	PC – AT	PC – AT

Main Co-Processor	External	External	External	Internal	Internal
Introduction	1978	1982	1985	1989	1993

Advantages:

Compatibility: One of the key advantages of the x86 architecture is its compatibility with older processors. This allows software developed for older processors to run on newer x86 processors without modification, which makes it easy to upgrade systems.
Performance: The evolution of x86 microprocessors has resulted in significant improvements in performance. Each new generation of processors has been faster and more efficient than the previous one, which has enabled the development of more advanced applications and technologies.
Versatility: The x86 architecture is used in a wide range of applications, from personal computers to servers, embedded systems, and mobile devices. This versatility has made it one of the most widely used processor architectures in the world.
Broad Industry Support: The x86 architecture is supported by a large ecosystem of hardware and software vendors. This broad industry support has helped to drive innovation and development, resulting in a range of products that are designed to work with x86 processors.

Disadvantages:

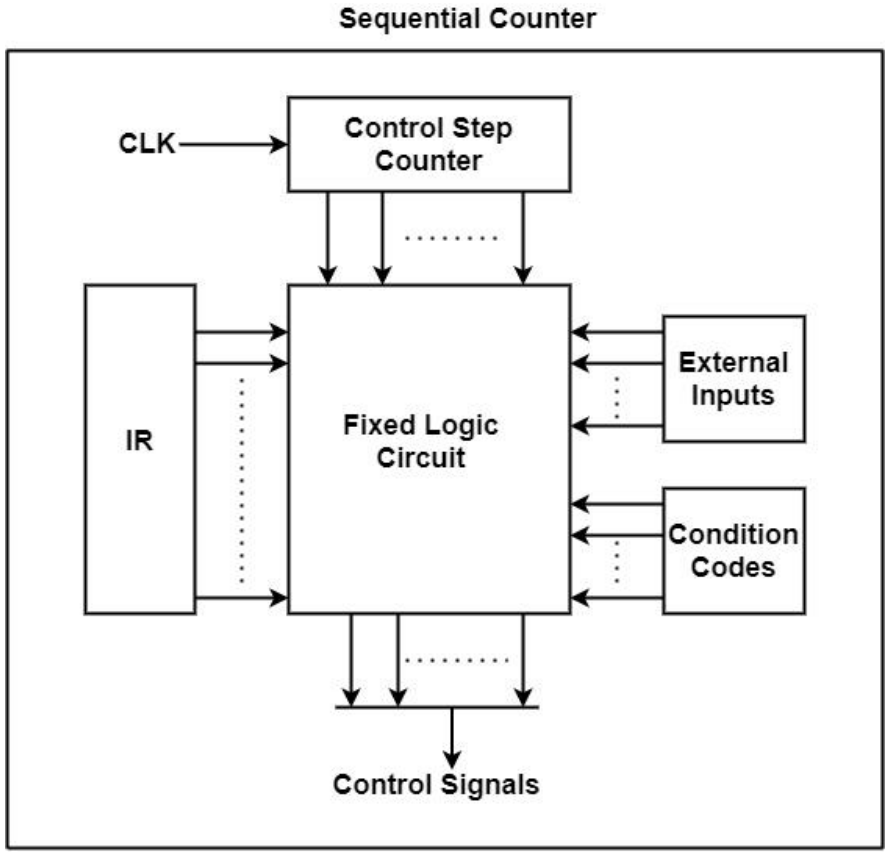
Complex Instruction Set: The x86 architecture has a complex instruction set, which makes it difficult to optimize code for performance. This complexity can also make it harder to debug software and hardware issues.
Power Consumption: The evolution of x86 microprocessors has led to a significant increase in power consumption. This has become a major issue in mobile devices, where battery life is critical.
Heat Dissipation: As x86 processors have become more powerful, they have also become hotter. This has led to the development of more sophisticated cooling systems, which can add to the cost and complexity of systems.
Cost: The x86 architecture is licensed by Intel, which can make it more expensive than other processor architectures that are available. This can be a significant barrier to entry for smaller hardware and software vendors.

CPU control unit design:

Hardwired Control Unit:

A hardwired control is a mechanism of producing control signals using Finite State Machines (FSM) appropriately. It is designed as a sequential logic circuit. The final circuit is constructed by physically connecting the components such as gates, flip flops, and drums. Hence, it is named a hardwired controller.

The figure shows a 2-bit sequence counter, which is used to develop control signals. The output obtained from these signals is decoded to generate the required signals in sequential order.



The hardwired control consists of a combinational circuit that outputs desired controls for decoding and encoding functions. The instruction that is loaded in the IR is decoded by the instruction decoder. If the IR is an 8-bit register, then the instruction decoder generates 2^8 (256) lines.

Inputs to the encoder are given from the instruction step decoder, external inputs, and condition codes. All these inputs are used and individual control signals are generated. The end signal is generated after all the instructions get executed. Furthermore, it results in the resetting of the control step counter, making it ready to generate the control step for the next instruction.

The major goal of implementing the hardwired control is to minimize the cost of the circuit and to achieve greater efficiency in the operation speed. Some of the methods that have come up for designing the hardwired control logic are as follows −

- **Sequence Counter Method** − This is the most convenient method employed to design the controller of moderate complexity.
- **Delay Element Method** − This method is dependent on the use of clocked delay elements for generating the sequence of control signals.
- **State Table Method** − This method involves the traditional algorithmic approach to design the Notes controller using the classical state table method.

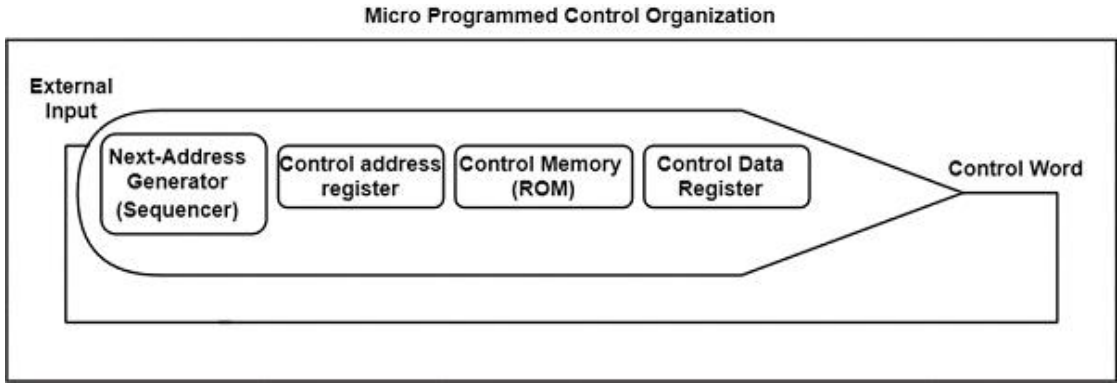
Microprogrammed Control Unit:

A control unit whose binary control values are saved as words in memory is called a microprogrammed control unit.

A controller results in the instructions to be implemented by constructing a definite collection of signals at each system clock beat. Each of these output signals generates one micro-operation including register transfer. Thus, the sets of control signals are generated definite micro-operations that can be saved in the memory.

Each bit that forms the microinstruction is linked to one control signal. When the bit is set, the control signal is active. When it is cleared the control signal turns inactive. These microinstructions in a sequence can be saved in the internal 'control' memory. The control unit of a microprogram-controlled computer is a computer inside a computer.

The following image shows the block diagram of a Microprogrammed Control organization.



There are the following steps followed by the microprogrammed control are −

- It can execute any instruction. The CPU should divide it down into a set of sequential operations. This set of operations are called microinstruction. The sequential micro-operations need the control signals to execute.

- Control signals saved in the ROM are created to execute the instructions on the data direction. These control signals can control the micro-operations concerned with a microinstruction that is to be performed at any time step.
- The address of the microinstruction is executed next is generated.
- The previous 2 steps are copied until all the microinstructions associated with the instruction in the set are executed.

The address that is supported to the control ROM originates from the micro counter register. The micro counter received its inputs from a multiplexer that chooses the output of an address ROM, a current address incrementer, and an address that is saved in the next address field of the current microinstruction.

Case study – design of a simple hypothetical CPU:

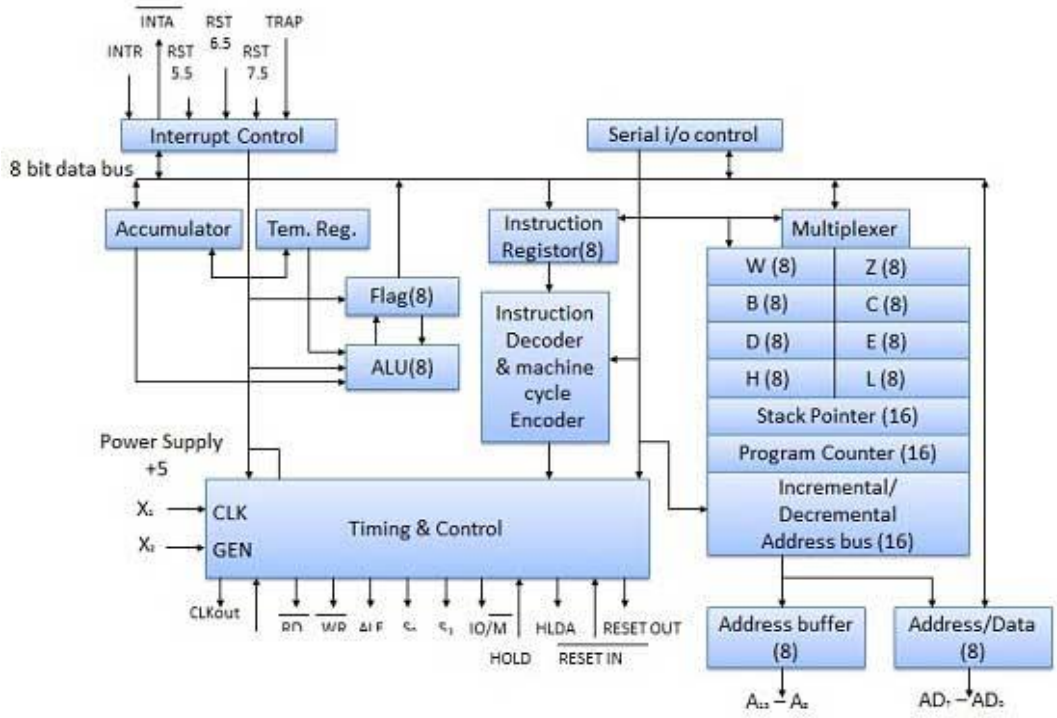
Microprocessing unit is synonymous to central processing unit, CPU used in traditional computer. Microprocessor (MPU) acts as a device or a group of devices which do the following tasks.

- communicate with peripherals devices
- provide timing signal
- direct data flow
- perform computer tasks as specified by the instructions in memory

8085 Microprocessor

The 8085 microprocessor is an 8-bit general purpose microprocessor which is capable to address 64k of memory. This processor has forty pins, requires +5 V single power supply and a 3-MHz single-phase clock.

Block Diagram



ALU

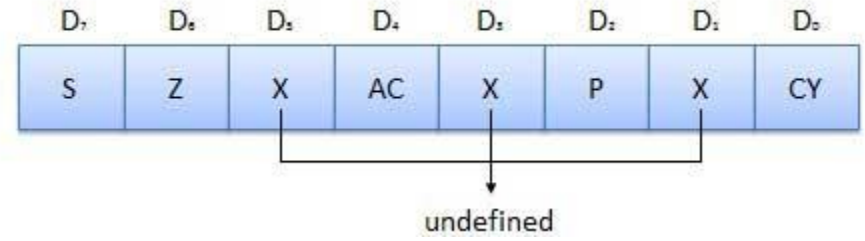
The ALU perform the computing function of microprocessor. It includes the accumulator, temporary register, arithmetic & logic circuit & and five flags. Result is stored in accumulator & flags.

Block Diagram

Accumulator

It is an 8-bit register that is part of ALU. This register is used to store 8-bit data & in performing arithmetic & logic operation. The result of operation is stored in accumulator.

Diagram



Flags

Flags are programmable. They can be used to store and transfer the data from the registers by using instruction. The ALU includes five flip-flops that are set and reset according to data condition in accumulator and other registers.

- **S (Sign) flag** − After the execution of an arithmetic operation, if bit D_7 of the result is 1, the sign flag is set. It is used to signed number. In a given byte, if D_7 is 1 means negative number. If it is zero means it is a positive number.
- **Z (Zero) flag** − The zero flag is set if ALU operation result is 0.
- **AC (Auxiliary Carry) flag** − In arithmetic operation, when carry is generated by digit D3 and passed on to digit D_4, the AC flag is set. This flag is used only internally BCD operation.
- **P (Parity) flag** − After arithmetic or logic operation, if result has even number of 1s, the flag is set. If it has odd number of 1s, flag is reset.
- **C (Carry) flag** − If arithmetic operation result is in a carry, the carry flag is set, otherwise it is reset.

Register section

It is basically a storage device and transfers data from registers by using instructions.

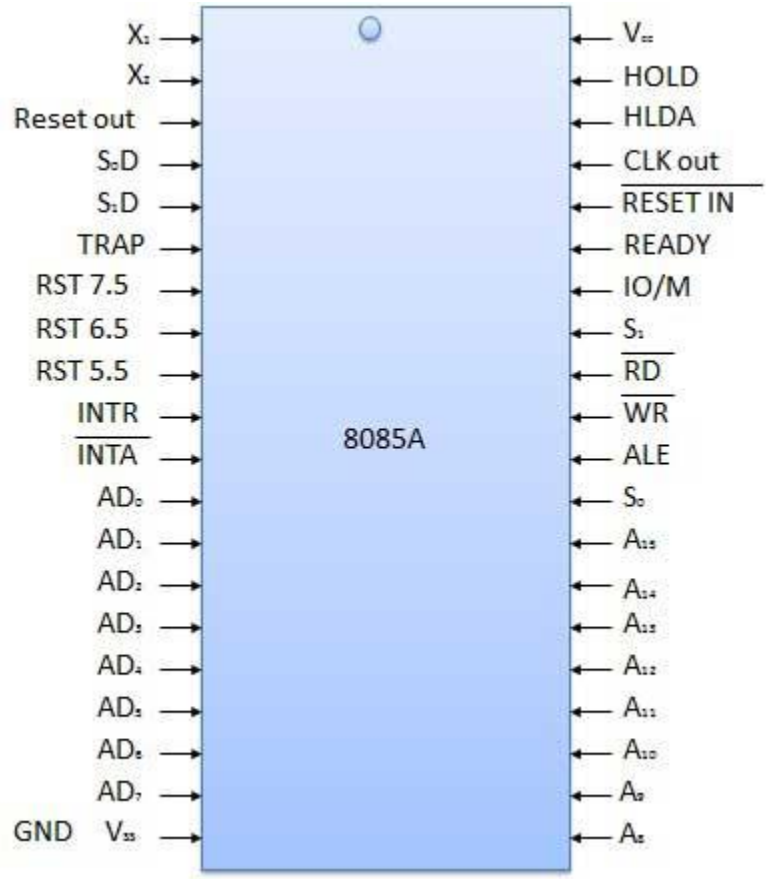
- **Stack Pointer (SP)** − The stack pointer is also a 16-bit register which is used as a memory pointer. It points to a memory location in Read/Write memory known as stack. In between execution of program, sometime data to be stored in stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.
- **Program Counter (PC)** − This 16-bit register deals with fourth operation to sequence the execution of instruction. This register is also a memory pointer. Memory location have 16-bit address. It is used to store the execution address. The function of the program counter is to point to memory address from which next byte is to be fetched.
- **Storage registers** − These registers store 8-bit data during a program execution. These registers are identified as B, C, D, E, H, L. They can be combined as register pair BC, DE and HL to perform some 16 bit operations.

Time and Control Section

This unit is responsible to synchronize Microprocessor operation as per the clock pulse and to generate the control signals which are necessary for smooth communication

between Microprocessor and peripherals devices. The RD bar and WR bar signals are synchronous pulses which indicates whether data is available on the data bus or not. The control unit is responsible to control the flow of data between microprocessor, memory and peripheral devices.

PIN diagram



All the signal can be classified into six groups

S.N.	Group	Description
1	**Address bus**	The 8085 microprocessor has 8 signal line, A_{15} - A_8 which are uni directional and used as a high order address bus.
2	**Data bus**	The signal line AD7 - AD0 are bi-directional for dual purpose. They are used as low order address bus as well as data bus.

3	**Control signal and Status signal**	Control Signal **RD bar** − It is a read control signal (active low). If it is active then memory read the data. **WR bar** − It is write control signal (active low). It is active when written into selected memory. Status signal **ALU (Address Latch Enable)** − When ALU is high. 8085 microprocessor use address bus. When ALU is low. 8085 microprocessor is use data bus. **IO/M bar** − This is a status signal used to differentiate between i/o and memory operations. When it is high, it indicate an i/o operation and when it is low, it indicate memory operation. **S_1 and S_0** − These status signals, similar to i/o and memory bar, can identify various operations, but they are rarely used in small system.
4	**Power supply and frequency signal**	**V_{cc}** − +5v power supply. **V_{ss}** − ground reference. **X, X** − A crystal is connected at these two pins. The frequency is internally divided by two operate system at 3-MHz, the crystal should have a frequency of 6-MHz. **CLK out** − This signal can be used as the system clock for other devices.
5	**Externally initiated signal**	**INTR (i/p)** − Interrupt request. **INTA bar (o/p)** − It is used as acknowledge interrupt. **TRAP (i/p)** − This is non maskable interrupt and has highest priority.

		HOLD (i/p) − It is used to hold the executing program. **HLDA (o/p)** − Hold acknowledge. **READY (i/p)** − This signal is used to delay the microprocessor read or write cycle until a slow responding peripheral is ready to accept or send data. **RESET IN bar** − When the signal on this pin goes low, the program counter is set to zero, the bus are tri-stated, & MPU is reset. **RESET OUT** − This signal indicate that MPU is being reset. The signal can be used to reset other devices. **RST 7.5, RST 6.5, RST 5.5 (Request interrupt)** − It is used to transfer the program control to specific memory location. They have higher priority than INTR interrupt.
6	**Serial I/O ports**	The 8085 microprocessor has two signals to implement the serial transmission serial input data and serial output data.

Instruction Format

Each instruction is represented by a sequence of bits within the computer. The instruction is divided into group of bits called field. The way instruction is expressed is known as instruction format. It is usually represented in the form of rectangular box. The instruction format may be of the following types.

Variable Instruction Formats

These are the instruction formats in which the instruction length varies on the basis of opcode & address specifiers. For Example, VAX instruction vary between 1 and 53 bytes while X86 instruction vary between 1 and 17 bytes.

Format

Advantage

These formats have good code density.

Drawback

These instruction formats are very difficult to decode and pipeline.

Fixed Instruction Formats

In this type of instruction format, all instructions are of same size. For Example, MIPS, Power PC, Alpha, ARM.

Format

Advantage

They are easy to decode & pipeline.

Drawback

They don't have good code density.

Memory system design:

semiconductor memory technologies:

A device for storing digital information that is fabricated by using integrated circuit technology is known as semiconductor memory. Also known as integrated-circuit memory, large-scale integrated memory, memory chip, semiconductor storage, transistor memory.

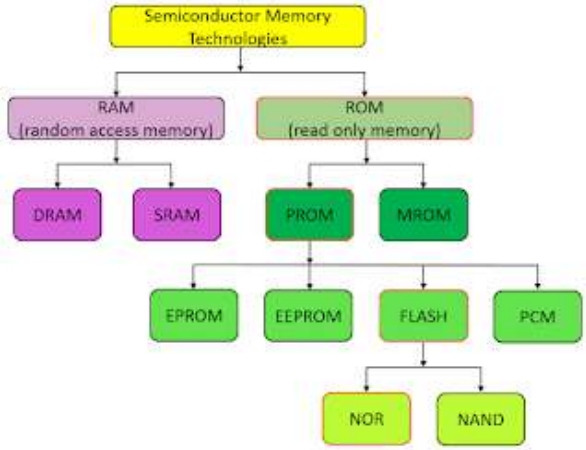
Semiconductor memory is the main memory element of a microcomputer-based system and is used to store program and data. The main memory elements are nothing but semiconductor devices that stores code and information permanently. The semiconductor memory is directly accessible by the microprocessor. And the access time of the data present in the primary memory must be compatible with the operating time of the microprocessor.

Thus semiconductor devices are preferred as primary memory. With the rapid growth in the requirement for semiconductor memories there have been a number of technologies and types of memory that have emerged. Names such as ROM, RAM, EPROM, EEPROM,etc.

Types of semiconductor memory

Electronic semiconductor memory technology can be split into two main types or categories, according to the way in which the memory operates :

1. RAM - ***Random Access Memory***
2. ROM - ***Read Only Memory***



There is a large variety of types of ROM and RAM that are available. These arise from the variety of applications and also the number of technologies available.

(i) Random Access Memory (RAM)

As the names suggest, the RAM or ***random access memory*** is a form of semiconductor memory technology that is used for reading and writing data in any order - in other words as it is required by the processor. It is used for such applications as the computer or processor memory where variables and other storage are required on a random basis. Data is stored and read many times to and from this type of memory.

Random access memory is used in huge quantities in computer applications as current day computing and processing technology requires large amounts of memory to enable them to handle the memory hungry applications used today. Many types of RAM including SDRAM with its DDR3, DDR4, and soon DDR5 variants are used in huge quantities.

DRAM

Dynamic RAM is a form of random access memory. DRAM uses a capacitor to store each bit of data, and the level of charge on each capacitor determines whether that bit is a logical 1 or 0. However these capacitors do not hold their charge indefinitely, and therefore the data needs to be refreshed periodically. As a result of this dynamic refreshing it gains its name of being a dynamic RAM.

DRAM is the form of semiconductor memory that is often used in equipment including personal computers and workstations where it forms the main RAM for the computer. The semiconductor devices are normally available as integrated circuits for use in PCB assembly in the form of surface mount devices or less frequently now as leaded components.

Disadvantages of DRAM

1. Complex manufacturing process
2. Data requires refreshing
3. More complex external circuitry required (read and refresh periodically)
4. Volatile memory
5. Relatively slow operational speed
6. Need to refresh the capacitor charge every once in two milliseconds.

SRAM

SRAM is stands for ***Static Random Access Memory***. This form of semiconductor memory gains its name from the fact that, unlike DRAM, the data does not need to be refreshed dynamically. These semiconductor devices are able to support faster read and write times than DRAM (typically 10 ns against 60 ns for DRAM), and in addition its cycle time is much shorter because it does not need to pause between accesses.

However they consume more power, they are less dense and more expensive than DRAM. As a result of this SRAM is normally used for caches, while DRAM is used as the main semiconductor memory technology.

(ii) Read Only Memory (ROM)

A ***ROM*** is a form of semiconductor memory technology used where the data is written once and then not changed. In view of this it is used where data needs to be stored permanently, even when the power is removed - many memory technologies lose the data once the power is removed. As a result, this type of semiconductor memory technology is widely used for storing programs and data that must survive when a computer or processor is powered down.

For example, the BIOS of a computer will be stored in ROM. As the name implies, data cannot be easily written to ROM. Depending on the technology used in the ROM, writing the data into the ROM initially may require special hardware. Although it is often possible to change the data, this gain requires special hardware to erase the data ready for new data to be written in.

PROM

This stands for ***Programmable Read Only Memory***. It is a semiconductor memory which can only have data written to it once, the data written to it is permanent. These memories are bought in a blank format and they are programmed using a special PROM programmer. Typically a PROM will consist of an array of fuseable links some of which are "blown" during the programming process to provide the required data pattern.

The PROM stores its data as a charge on a capacitor. There is a charge storage capacitor for each cell and this can be read repeatedly as required. However it is found that after many years the charge may leak away and the data may be lost. Nevertheless, this type of semiconductor memory used to be widely used in applications where a form of ROM was required, but where the data needed to be changed periodically, as in a development environment, or where quantities were low.

EPROM

This is an ***Erasable Programmable Read Only Memory***. This form of semiconductor memory can be programmed and then erased at a later time. This is normally achieved by exposing the silicon to ultraviolet light. To enable this to happen there is a circular window in the package of the EPROM to enable the light to reach the silicon of the chip. When the PROM is in use, this

window is normally covered by a label, especially when the data may need to be preserved for an extended period.

EEPROM

This is an ***Electrically Erasable Programmable Read Only Memory***. Data can be written to it and it can be erased using an electrical voltage. This is typically applied to an erase pin on the chip. Like other types of PROM, EEPROM retains the contents of the memory even when the power is turned off. Also like other types of ROM, EEPROM is not as fast as RAM. EEPROM memory cells are made from floating-gate MOSFETS (known as FGMOS).

Flash memory

Flash memory may be considered as a development of EEPROM technology. Data can be written to it and it can be erased, although only in blocks, but data can be read on an individual cell basis. To erase and re-program areas of the chip, programming voltages at levels that are available within electronic equipment are used. It is also non-volatile, and this makes it particularly useful. As a result Flash memory is widely used in many applications including memory cards for digital cameras, mobile phones, computer memory sticks and many other applications.

Flash memory stores data in an array of memory cells. The memory cells are made from floating-gate MOSFETS (known as FGMOS). These FG MOSFETs (or FGMOS in short) have the ability to store an electrical charge for extended periods of time (2 to 10 years) even without a connecting to a power supply.

Disadvantages of Flash Memory

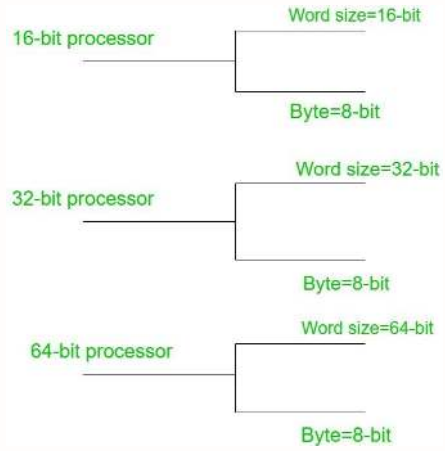
1. Higher cost per bit than hard drives
2. Slower than other forms of memory
3. Limited number of write / erase cycles
4. Data must be erased before new data can be written
5. Data typically erased and written in blocks

Memory organization:

The memory is organized in the form of a cell, each cell is able to be identified with a unique number called address. Each cell is able to recognize control signals such as “read” and “write”, generated by CPU when it wants to read or write address. Whenever CPU executes the program there is a need to transfer the instruction from the memory to CPU because the program is available in memory. To access the instruction CPU generates the memory request.

Memory Request:
Memory request contains the address along with the control signals. For Example, When inserting data into the stack, each block consumes memory (RAM) and the number of memory cells can be determined by the capacity of a memory chip.

Word Size: It is the maximum number of bits that a CPU can process at a time and it depends upon the processor. Word size is a fixed size piece of data handled as a unit by the instruction set or the hardware of a processor.



Word size varies as per the processor architectures because of generation and the present technology, it could be low as 4-bits or high as 64-bits depending on what a particular processor can handle. Word size is used for a number of concepts like Addresses, Registers, Fixed-point numbers, Floating-point numbers.

Peripheral devices and their characteristics:
Input-output subsystems:

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It

handles all the input- output operations of the computer system. Input or output devices that are connected to computer are called peripheral devices. There are three types of peripherals:

1. **Input peripherals** : Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.
2. **Output peripherals**: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc
3. **Input-Output peripherals**: Allows both input(from outised world to computer) as well as, output(from computer to the outside world). Example: Touch screen etc.

I/O device interface:

Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called **input-output interface units** because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

I/O transfers–program controlled:

Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.

Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU. It is time-consuming as it keeps the CPU busy needlessly.

Interrupt Driven I/O: To overcome the disadvantage of Programmed I/O,i.e., keeping CPU busy needlessly, Interrupt — Driven I/O is used. In this approach, when a peripheral sends an interrupt signal to the CPU whenever it is ready to transfer data. This indicates that the I/O data transfer is initiated by the external I/O device. The processor stops the execution of the current program & transfers the control to interrupt the service routine when interrupted. The interrupt service routine then performs the data transfer. After the completion of data transfer, it returns control to the main program to the point it was interrupted.

DMA(Direct Memory Access): DMA transfer is used for large data transfers. Here, a memory bus is used by the interface to transfer data in & out of a memory unit. The CPU provides starting address & number of bytes to be transferred to the interface to initiate the transfer, after that it proceeds to execute other tasks. DMA requests a memory cycle through the memory bus when the transfer is made. DMA transfers the data directly into the memory when the request is granted by the memory controller. To allow direct memory transfer(I/O), the CPU delays its memory access operation. So, DMA allows I/O devices to directly access memory with less intervention of the CPU.

Privileged and Non-privileged instructions:

In an operating system, instructions are divided into two categories: privileged and non-privileged instructions.

Privileged instructions are those that can only be executed by the operating system kernel or a privileged process, such as a device driver. These instructions typically perform operations that require direct access to hardware or other privileged resources, such as setting up memory mappings or accessing I/O devices. Privileged instructions are executed in kernel mode, which provides unrestricted access to the system resources.

Non-privileged instructions are those that can be executed by any process, including user-level processes. These instructions are typically used for performing computations, accessing user-level resources such as files and memory, and managing process control. Non-privileged instructions are executed in user mode, which provides limited access to system resources and ensures that processes cannot interfere with one another.

Software interrupts and exceptions:

Exceptions and interrupts are unexpected events which will disrupt the normal flow of execution of instruction(that is currently executing by processor). An exception is an unexpected event from within the processor. Interrupt is an unexpected event from outside the process.
Whenever an exception or interrupt occurs, the hardware starts executing the code that performs an action in response to the exception. This action may involve killing a process, outputting an error message, communicating with an external device, or horribly crashing the entire computer system by initiating a “Blue Screen of Death” and halting the CPU. The instructions responsible for this action reside in the operating system kernel, and the code that performs this action is called the interrupt handler code. Now, We can think of handler code as an operating system subroutine. Then, After the handler code is executed, it may be possible to continue execution after the instruction where the execution or interrupt occurred.

Exception and Interrupt Handling :
Whenever an exception or interrupt occurs, execution transition from user mode to kernel mode where the exception or interrupt is handled. In detail, the following steps must be taken to handle an exception or interrupts.
While entering the kernel, the context (values of all CPU registers) of the currently executing process must first be saved to memory. The kernel is now ready to handle the exception/interrupt.

1. Determine the cause of the exception/interrupt.
2. Handle the exception/interrupt.

When the exception/interrupt have been handled the kernel performs the following steps:

1. Select a process to restore and resume.
2. Restore the context of the selected process.
3. Resume execution of the selected process.

At any point in time, the values of all the registers in the CPU defines the context of the CPU. Another name used for CPU context is CPU state.

The exception/interrupt handler uses the same CPU as the currently executing process. When entering the exception/interrupt handler, the values in all CPU registers to be used by the exception/interrupt handler must be saved to memory. The saved register values can later restored before resuming execution of the process.

The handler may have been invoked for a number of reasons. The handler thus needs to determine the cause of the exception or interrupt. Information about what caused the exception or interrupt can be stored in dedicated registers or at predefined addresses in memory.

Next, the exception or interrupt needs to be serviced. For instance, if it was a keyboard interrupt, then the key code of the key press is obtained and stored somewhere or some other appropriate action is taken. If it was an arithmetic overflow exception, an error message may be printed or the program may be terminated.

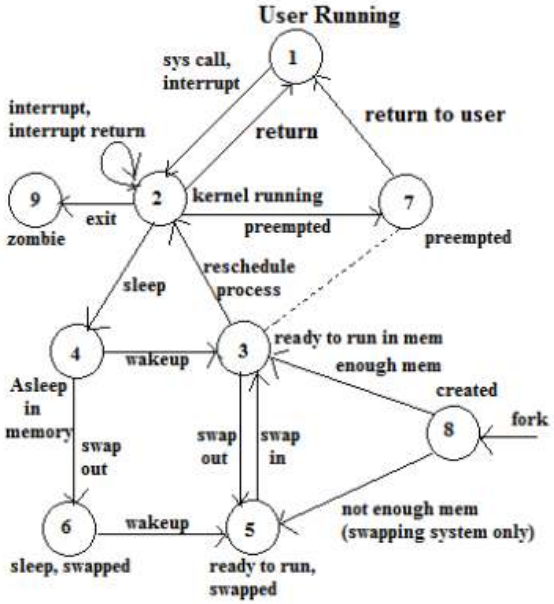
The exception/interrupt have now been handled and the kernel. The kernel may choose to resume the same process that was executing prior to handling the exception/interrupt or resume execution of any other process currently in memory.

The context of the CPU can now be restored for the chosen process by reading and restoring all register values from memory.

The process selected to be resumed must be resumed at the same point it was stopped. The address of this instruction was saved by the machine when the

interrupt occurred, so it is simply a matter of getting this address and make the CPU continue to execute at this address.

Programs and processes–role of interrupts in process state transitions:



The complete set of process states:

- Executing in user mode
- Executing in kernel mode
- Ready to run
- Sleeping in memory
- Ready to run, but in swap space (covered later)
- Sleeping in swap space
- Preempted (the process is returning from kernel to user mode, but the kernel preempts it and does a context switch to schedule another process. Very similar to state 3)
- Newly created. Not ready run, nor sleeping. (This is the start state for all processes except process 0)
- The process executed exit system call and is in the zombie state. The process no longer exists, but it leaves a record containing an exit code and some timing statistics for its parent process to collect. The zombie state is the final state of a process.

When the parent process executes the fork system call model and it moves into a state where it is ready to run (3 or 5) at that time the process enters the created state.

The scheduler will eventually pick the process and the process enters the 'kernel running' state where it completes its part of the fork system call. After the completion of

the system call, it may move to the 'user running'. When interrupts occur (such as system call), it again moves to the state 'kernel running'.

After completing the task of the interrupt the kernel may decide to schedule another process to execute, so the first process enters the state 'preempted'. The state preempted is actually same as the state ready to run in memory, but they are denoted separately to stress that a process executing in kernel mode can be preempted only when it is about to return to user mode. Consequently, the kernel could swap a process from the state preempted if necessary. Eventually, it will return to the 'user running' again.

When the system call is executed, it leaves the state user running and enters the state kernel running. If in kernel mode, the process needs to sleep for some reason (such as waiting for I/O), it enters the state asleep in memory. When the event on it which it has slept, happens, the interrupt handler awakens the process, and it enters the state ready to run in memory.

Suppose the system is executing many processes that do not set at the same time into main memory, then the swapper (process 0) swaps out a process to make space for another process that is in the state ready to run swapped. When forcefully takes from main memory, the process enters the state ready to run swapped. Finally, swapper chooses the process as most eligible to run and it re-enters the state ready to run in memory. And then when it is scheduled, it will enter the state kernel running. When a process completes and invokes exit system call, thus entering the states kernel running and finally, the zombie state.

Some state transitions can be managed by the users, but not all. User can create a process. But the user has no control over when a process transitions to sleeping in memory to sleeping in the swap, or ready to run in memory to ready to run in the swap, etc. A process can make a system call to transition itself to kernel running state. But it has no control over when it will return from kernel mode. Finally, a process can exit whenever it wants, but that is not the only reason for exit to be called.

Two kernel data structures describe the state of a process: the process table entry and the u-area as we have studied already. The process table contains information that should be accessible to the kernel and the u-area contains the information that should be accessible to the process only when it's running. The kernel allocates space for u-area only when creating a process. It does not need u-area for process table entries that do not have processes. For example, the process table entries that contain the information about the kernel context, do not have processes.

I/O device interfaces – SCII:

The basic interface for connecting peripheral devices to a PC is a small computer system interface. Based on the specification, it can typically **respond up to 16 external devices** using a single route, along with a host adapter. Small Computer System Interface is used to boost performance, deliver fast data transfer delivery and provide wider expansion for machines like CD-ROM drivers, scanners, DVD> drives

and CD writers. Small Computer System Interface is most commonly used for RAID, servers, highly efficient desktop computers, and storage area networks. The Small Computer System Interface has control, which is responsible for transmitting data across the Small Computer System Interface bus and the computers. It can be fixed on a **motherboard**, or one client adapter is installed through an extension on the computer's motherboard. The controller also incorporates a simple SCSI input/output system, which is a small chip that provides access and control equipment with the necessary software. The SCSI ID is his number. Using serial storage architecture initiators, new serial SCSI IDs such as serial attached SCSI use an automatic process which assigns a 7-bit number.

USB:

A USB is a common computer port, which shorts for Universal Serial Bus and allows communication between a computer and peripheral or other devices. It is the most common interface used in today's computers, which can be used to connect printers, scanners, keyboards, mice, game controllers, digital cameras, external hard drives and flash drives. The USB has replaced a wide range of interfaces like the parallel and serial port because it is used for a wide variety of uses as well as offers better support for electrical power. With a single USB port, up to 127 peripherals can be connected with the help of a few USB hubs, although that will need quite a bit of dexterity.

In modern times, to connect with the computer, there are many different USB devices. Some common are as follows:

- Keyboard
- Smartphone
- Tablet
- Webcams
- Keypad
- Microphone
- Mouse
- Joystick

In modern times, all computers contain at least one USB port in different locations. Below, a list is given that contains USB port locations on the devices that may help you out to find them.

- **Laptop computer:** A laptop computer may contain one to four ports on the left or right side, and some laptops have on the behind of the laptop computer.
- **Desktop computer:** Usually, a desktop computer has 2 to 4 USB ports in the front and 2 to 8 ports on the backside.
- **Tablet computer:** On the tablet, a USB connection is situated in the charging port and is sometimes USB-C and usually micro USB.
- **Smartphone:** In the form of micro USB or USB-C, a USB port is used for both data transfer and charging, similar to tablets on smartphones.

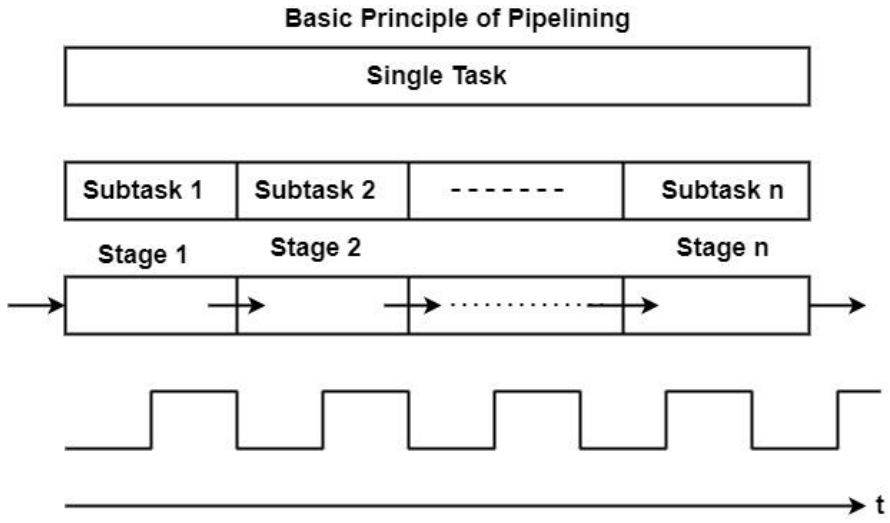
Module-IV:

Pipelining:

Pipelining defines the temporal overlapping of processing. Pipelines are emptiness greater than assembly lines in computing that can be used either for instruction processing or, in a more general method, for executing any complex operations. It can be used efficiently only for a sequence of the same task, much similar to assembly lines.

A basic pipeline processes a sequence of tasks, including instructions, as per the following principle of operation −

Each task is subdivided into multiple successive subtasks as shown in the figure. For instance, the execution of register-register instructions can be broken down into instruction fetch, decode, execute, and writeback.



A pipeline phase related to each subtask executes the needed operations.

A similar amount of time is accessible in each stage for implementing the needed subtask.

All pipeline stages work just as an assembly line that is, receiving their input generally from the previous stage and transferring their output to the next stage.

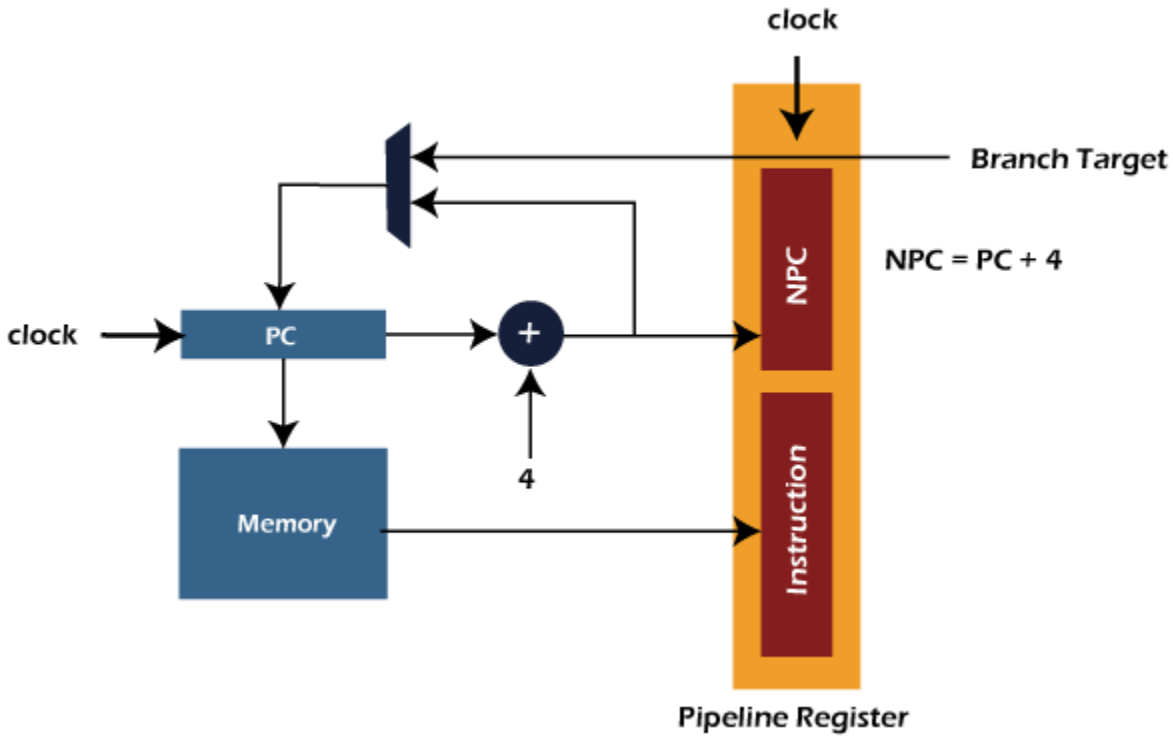
Finally, it can consider the basic pipeline operates clocked, in other words synchronously. This defines that each stage gets a new input at the beginning of the clock cycle, each stage has a single clock cycle available for implementing the needed operations, and each stage produces the result to the next stage by the starting of the subsequent clock cycle.

Throughput and Speedup:

In the RISC processor, we can execute all the instructions of RISC instruction set with the help of 5 instructions stages. The first stage is **instruction fetch**, where the instruction is fetched from memory. The second stage is **instruction decodes**, where instruction is decoded and register is read. The third stage is the **instruction execution**, where we calculate the address or execute the operation. The fourth stage is the **memory access stage**, where memory operands are accessed. The fifth stage is the **write back stage**, where the result writes back to the register. The detailed explanation of all these 5 stages of the RISC pipeline and their operations are described as follows:

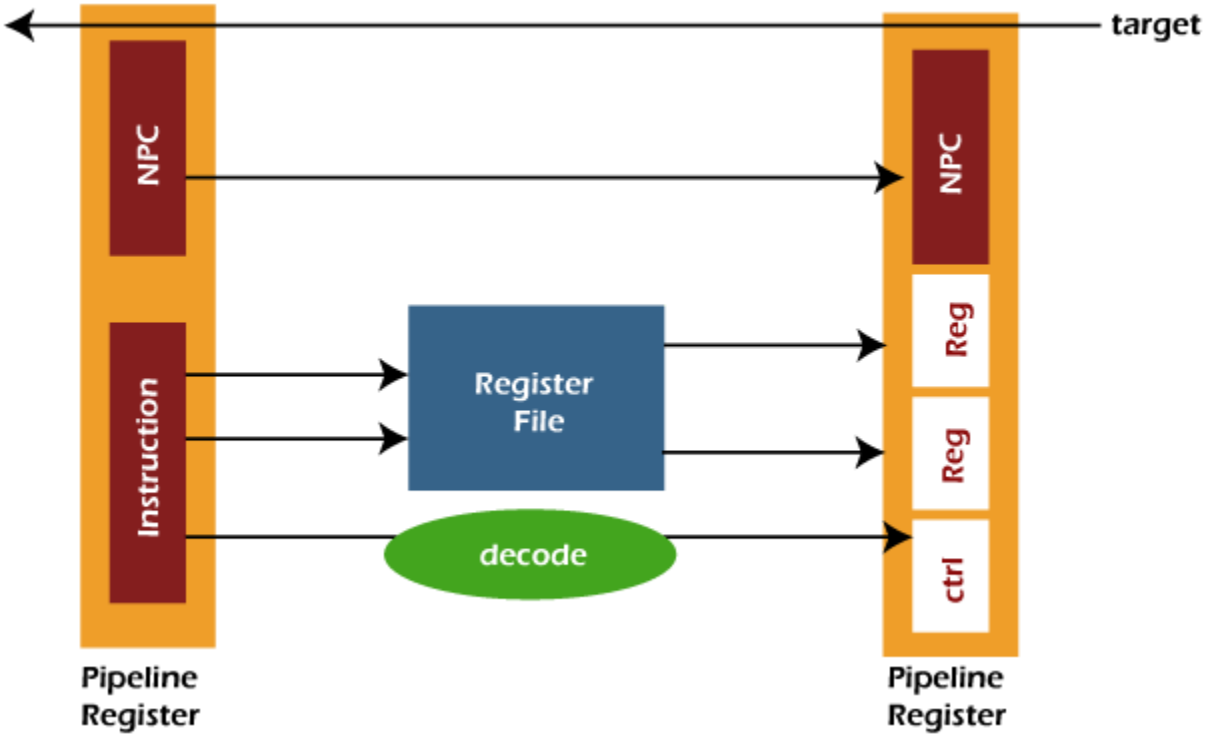
Stage 1:

Stage 1 is the **instruction fetch**. Here, an instruction is read from memory (I-Memory). In this stage, a program counter is used to contain the value of memory. With the help of incrementing the current PC, we are able to compute the NPC. The pipeline registers will be updated by writing the instruction into PR. The process of instruction fetch stage is described as follows:



Stage 2:

Stage 2 is the **instruction decodes stage**. Here instruction is decoded, and control signals are generated for the opcode bits. In this stage, the source operands are read from the register file. With the help of specifiers, the indexing of register file is done. The pipeline register will send the operands and immediate value to the next stage. It will also pass NPC and control signals to the next stage. The process of instruction decoder stage is described as follows:

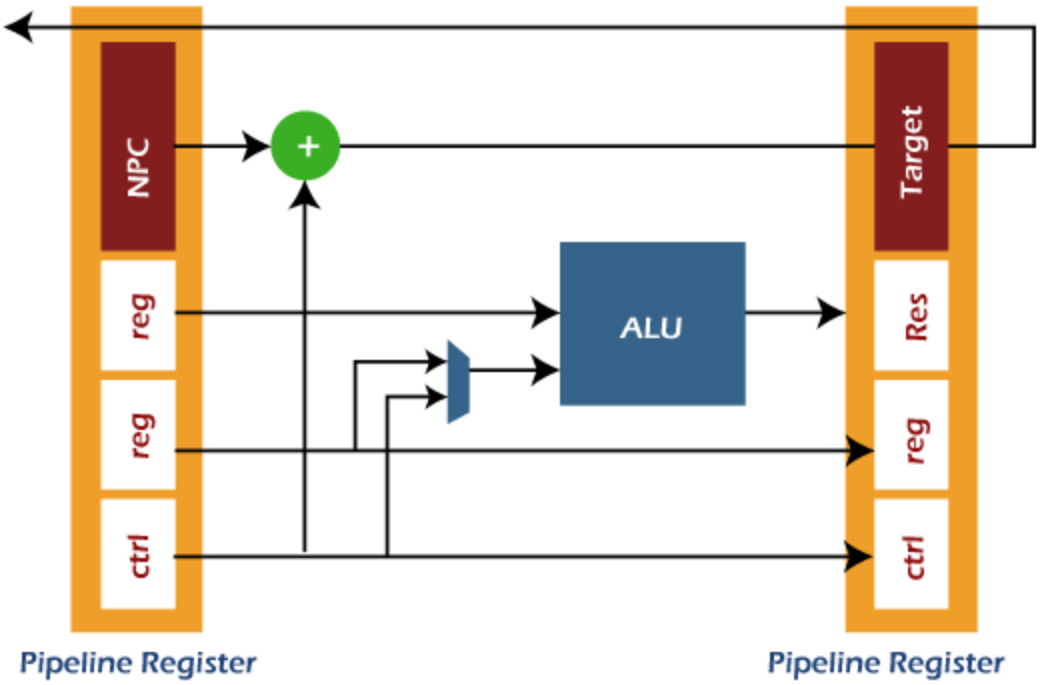


Stage 3:

Stage 3 is the **Instruction executes stage**. The ALU (arithmetical logical unit) operations are performed in this stage. It takes the two operands to perform the ALU operation. The first operand is used to contain the content of a register, and the second operand is used to contain either immediate value or a register. In this stage, the branch target can be computed with the help of following formula:

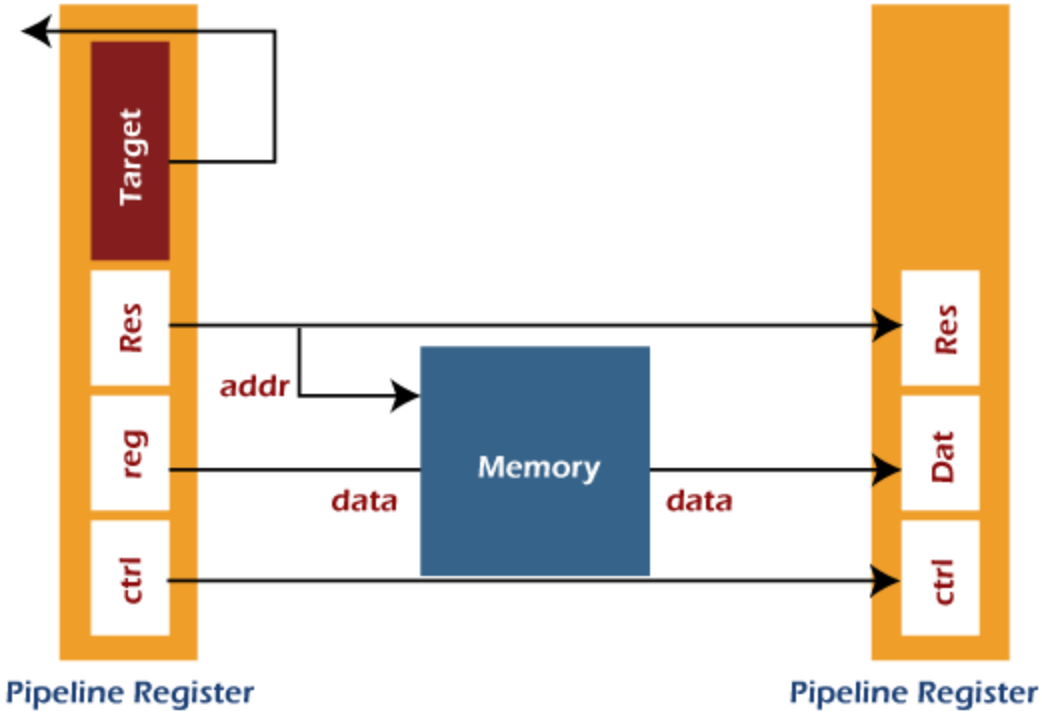
1. Target = NPC + immediate

The pipeline register (PR) will update the ALU result, branch target, control signals, and destination. The process of instruction execution is described as follows:



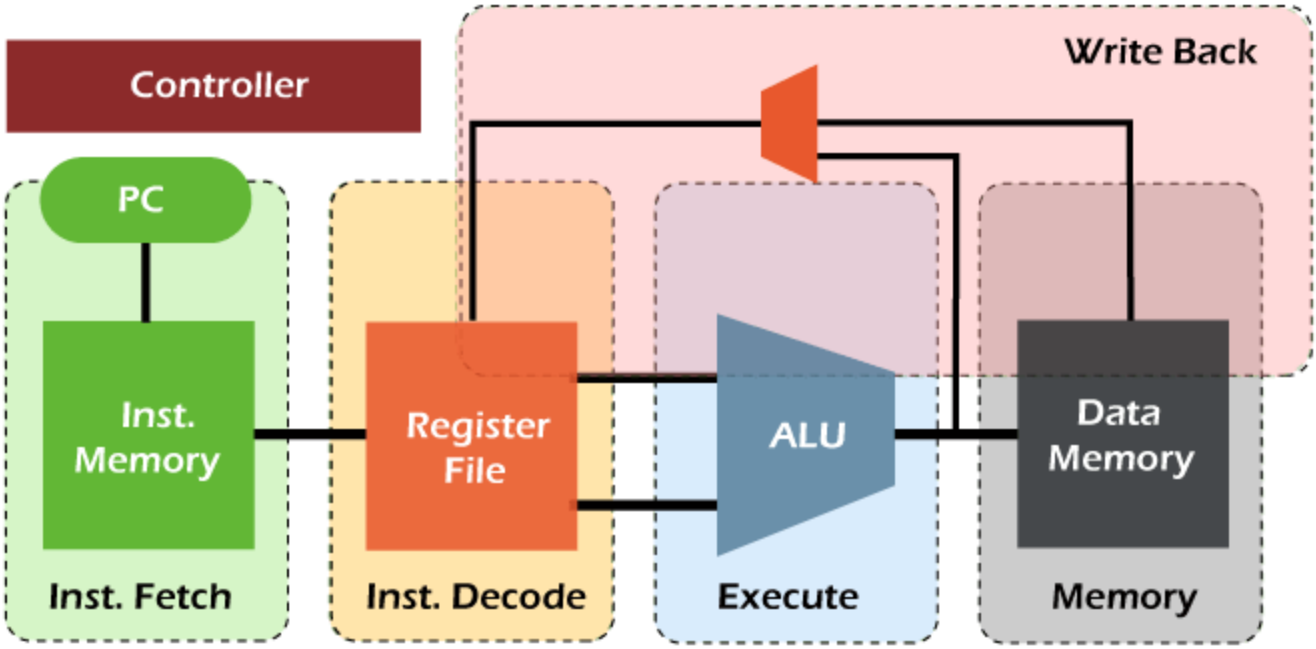
Stage 4:

Stage 4 is the **Memory Access stage**. Here, memory operands are able to read and write to/from memory, which exists in the instruction. The pipeline register (PR) will update the ALU result from execution, destination register, and loaded data from D-Memory. The process of memory access is described as follows:



Stage 5:

Stage 5 is the **Write Back stage**. Here, the fetched value is written back to the register, which exists in the instruction. This stage only needs one write part, which can either be used to write the loaded data into the register file or to write the result of ALU into the destination register.



Performance of Pipelined processor

Here we will assume a segment pipeline as 'k' and the clock cycle time 'Tp'. Suppose the pipeline processor needs to complete the 'n' number of tasks. Now the first instruction will come out from the pipeline by taking the 'k' cycle. On the other hand, 'n-1' instructions will take only '1' cycle each. That means the 'n-1' instruction will take the total 'n-1' cycle. In a pipeline processor, when we try to execute the 'n' instructions, then the time taken to do this is described as follows:

$$ET_{pipeline} = k + n - 1 \text{ cycle}$$
$$= (k + n - 1)\ Tp$$

In a non-pipeline processor, when we take the same case and try to execute the 'n' instructions, then the time taken to do this is described as follows:

$$ET_{non\text{-}pipeline} = n * k * Tp$$

So, when we perform 'n' tasks on the same processor, the speedup (S) of pipeline processor over non-pipelined processor is described as follows:

```
S = Performance of pipeline processor / Performance of Non-pipelined processor
```

As the execution time and the performance of process is inversely proportional to each other. So we have the following relation:

$$S = ET_{non\text{-}pipeline} / ET_{pipeline}$$

$$S = [n * k * Tp] / [(k + n - 1) * Tp]$$

$$S = [n * k] / [k + n - 1]$$

The following relation will contain if the n number of tasks is larger than 'k' that means n >>k.

$$S = n * k / n$$

$$S = k$$

Here K is used to indicate the number of stages in the pipeline.

$$\text{Also, Efficiency} = S / S_{max}$$

Here S is used to show the max speed up, and Smax is used to indicate the Maximum speed up.

$$\text{We know that } S_{max} = k$$

$$\text{So, Efficiency} = S / k$$

Now throughput can be described like this:

```
Throughput = Number of instruction / Total time to complete the instruction
```

$$\text{So throughput} = n / (k + n + 1) * Tp$$

Note: For the ideal pipeline processor, the value of Cycle per instruction (CPI) is 1.

Pipeline Conflicts

The performance of pipelines is affected by various factors. Some of the factors are described as follows:

Timing Variations

We know that the pipeline cannot take same amount of time for all the stages. The problem related to timing variation will occur if we are processing some instructions where different instructions need different operands and take different processing times.

Data Hazards

The problem of data hazards will occur when there is parallel execution of several instructions, and these instructions reference the same data. So we should be careful that the next instruction does not try to access the data which is already accessed by the current instruction. If this situation arises, it will generate the incorrect result.

Branching

We should know about the instruction before we try to fetch and execute the next instruction. Suppose there is a case in which the current instruction contains the conditional branch, and the result of this instruction will lead to the next instruction. In this case, we will not be able to know the next instruction as long as the current instruction is proceeding.

Interrupts

Because of the interrupts, the unwanted instruction will be set into the instruction stream. The execution of instruction is also affected by the interrupt.

Data dependency

The situation of data dependency will occur when the result of previous instruction will lead to the next instruction, and this result is not yet available.

Advantage of Pipelining

- The pipeline has the ability to increase the throughput of the system.
- We can use the pipeline in a modern processor. It is also used to reduce the cycle time of processor.
- It is used to make the system reliable.
- It is used to arrange the hardware so that it can perform more than one operation at once.

- Suppose there is an application that wants to repeat the same task with some other set of data in many time. In this case, this technique will be very efficient.

Disadvantage of Pipelining

- In this pipeline, the instruction latency is more.
- The process of designing a pipeline is very costly and complex because it contains additional hardware.

Pipeline hazards:

Pipeline hazards are conditions that can occur in a pipelined machine that impede the execution of a subsequent instruction in a particular cycle for a variety of reasons.

There are three kinds of hazards:

- Structural Hazards
- Data Hazards
- Control Hazards

There are many specific solutions to dependencies. The simplest is introducing a ***bubble*** which stalls the pipeline and reduces the throughput. The bubble makes the next instruction wait until the earlier instruction is done with.

Structural Hazards

Structural hazards arise due to hardware resource conflict amongst the instructions in the pipeline. A resource here could be the Memory, a Register in GPR or ALU. This resource conflict is said to occur when more than one instruction in the pipe is requiring access to the same resource in the same clock cycle. This is a situation that the hardware cannot handle all possible combinations in an overlapped pipelined execution.

Data Hazards

Data hazards in pipelining emerge when the execution of one instruction is dependent on the results of another instruction that is still being processed in the pipeline. The order of the READ or WRITE operations on the register is used to classify data threats into three groups.

Control Hazards

Branch hazards are caused by branch instructions and are known as control hazards in computer architecture. The flow of program/instruction execution is controlled by branch

instructions. Remember that conditional statements are used in higher-level languages for iterative loops and condition testing (correlate with while, for, and if case statements). These are converted into one of the BRANCH instruction variations. As a result, when the decision to execute one instruction is reliant on the result of another instruction, such as a conditional branch, which examines the condition's consequent value, a conditional hazard develops.

Parallel Processors:

Parallel processing can be described as a class of techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.

A parallel processing system can carry out simultaneous data-processing to achieve faster execution time. For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.

The primary purpose of parallel processing is to enhance the computer processing capability and increase its throughput, i.e. the amount of processing that can be accomplished during a given interval of time.

A parallel processing system can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. The data can be distributed among various multiple functional units.

Cache Coherence

A cache coherence issue results from the concurrent operation of several processors and the possibility that various caches may hold different versions of the identical memory block. The practice of cache coherence makes sure that alterations in the contents of associated operands are quickly transmitted across the system.

Cache coherence has three different levels:

- Each writing operation seems to happen instantly.
- Each operand's value changes are seen in every processor in precisely the same order.
- Non-coherent behavior results from many processors interpreting the same action in various ways.

Methods to resolve Cache Coherence

The two methods listed below can be used to resolve the cache coherence issue:

- Write Through
- Write Back

Write Through

The easiest and most popular method is to write through. Every memory write operation updates the main memory. If the word is present in the cache memory at the requested address, the cache memory is also updated simultaneously with the main memory.

The benefit of this approach is that the RAM and cache always hold the same information. In systems with direct memory access transfer, this quality is crucial. It makes sure the information in the main memory is up-to-date at all times so that a device interacting over DNA can access the most recent information.

Advantage - It provides the highest level of consistency.

Disadvantage - It requires a greater number of memory access.

Write Back

Only the catch location is changed during a write operation in this approach. When the word is withdrawn from the cache, the place is flagged, so it is replicated in the main memory. The right-back approach was developed because words may be updated numerous times while they are in the cache. However, as long as they are still there, it doesn't matter whether the copy that is stored in the main memory is outdated because requests for words are fulfilled from the cache.

An accurate copy must only be transferred back to the main memory when the word is separated from the cache. According to the analytical findings, between 10% and 30% of all memory references in a normal program are written into memory.

Advantage - A very small number of memory accesses and write operations.

Disadvantage - Inconsistency may occur in this approach.

The important terms related to the data or information stored in the cache as well as in the main memory are as follows:

- ***Modified*** - The modified term signifies that the data stored in the cache and main memory are different. This means the data in the cache has been modified, and the changes need to be reflected in the main memory.
- ***Exclusive*** - The exclusive term signifies that the data is clean, i.e., the cache and the main memory hold identical data.
- ***Shared*** - Shared refers to the fact that the cache value contains the most current data copy, which is then shared across the whole cache as well as main memory.
- ***Owned*** - The owned term indicates that the block is currently held by the cache and that it has acquired ownership of it, i.e., complete privileges to that specific block.
- ***Invalid*** - When a cache block is marked as invalid, it means that it needs to be fetched from another cache or main memory.

CPU Basics:Multiple CPUs, Cores, & Hyper-threading:

A multicore processor is an integrated circuit that has two or more processor cores attached for enhanced performance and reduced power consumption. These processors also enable more efficient simultaneous processing of multiple tasks, such as with parallel processing and multithreading.

In Hyper-Threading one core contains 1 or more threads and that threads again behave like logical CPU's. Because of it, several processes can run on different threads and improve the performance of the system.

Advantages

- **Multitasking:** If system supports Hyper-Threading then several processes can run simultaneously on the different threads of the CPU's core.
- **Resource Utilization:** Using Hyper-Threading resources can't be stay idle and utilized in great extent, because of several simultaneously running processes that can need resources. Because of this, the system's overall efficiency and performance increases.
- **Responsiveness:** Concurrent execution helps to decrease the response time for user.

Disadvantages

- If many processes are running simultaneously because of Hyper-Threading, and these processes compete for limited resources, such as cache memory then resources may get blocked.
- Because of many running processes, resources are shared among them, which can lead to increase the contention and delays in input/output of resources.

If Thread(s) per Core is greater than 1 then we can say that this system support the Hyper-Threading. Overall, Hyper-Threading improve the performance of multi-threaded applications on systems in modern computers via concurrent execution.

Introduction to Multiple-Processor Scheduling in Operating System:

Multiple processor scheduling or multiprocessor scheduling focuses on designing the system's scheduling function, which consists of more than one processor. Multiple CPUs share the load (load sharing) in multiprocessor scheduling so that various processes run simultaneously. In general, multiprocessor scheduling is complex as compared to single processor scheduling. In the multiprocessor scheduling, there are many processors, and they are identical, and we can run any process at any time.

The multiple CPUs in the system are in close communication, which shares a common bus, memory, and other peripheral devices. So we can say that the system is tightly coupled. These systems are used when we want to process a bulk amount of data, and these systems are mainly used in satellite, weather forecasting, etc.

There are cases when the processors are identical, i.e., homogenous, in terms of their functionality in multiple-processor scheduling. We can use any processor available to run any process in the queue.

Multiprocessor systems may be ***heterogeneous*** (different kinds of CPUs) or ***homogenous*** (the same CPU). There may be special scheduling constraints, such as devices connected via a private bus to only one CPU.

Approaches to Multiple Processor Scheduling

There are two approaches to multiple processor scheduling in the operating system: Symmetric Multiprocessing and Asymmetric Multiprocessing.

1. **Symmetric Multiprocessing:** It is used where each processor is ***self-scheduling***. All processes may be in a common ready queue, or each processor may have its private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.
2. **Asymmetric Multiprocessing:** It is used when all the scheduling decisions and I/O processing are handled by a single processor called the ***Master Server***. The other processors execute only the ***user code***. This is simple and reduces the need for data sharing, and this entire scenario is called Asymmetric Multiprocessing.

Processor Affinity

Processor Affinity means a process has an ***affinity*** for the processor on which it is currently running. When a process runs on a specific processor, there are certain effects on the cache memory. The data most recently accessed by the process populate the cache for the processor. As a result, successive memory access by the process is often satisfied in the cache memory.

Module-V:

Memory organization:

Memory interleaving: Memory Interleaving is less or More an Abstraction technique. Though it's a bit different from Abstraction. It is a Technique that divides memory into a number of modules such that Successive words in the address space are placed in the Different modules.

Why do we use Memory Interleaving? [Advantages]: Whenever Processor requests Data from the main memory. A block (chunk) of Data is Transferred to the cache and then to Processor. So whenever a cache miss occurs the Data is to be fetched from the main memory. But main memory is relatively slower than the cache. So to improve the access time of the main memory interleaving is used.

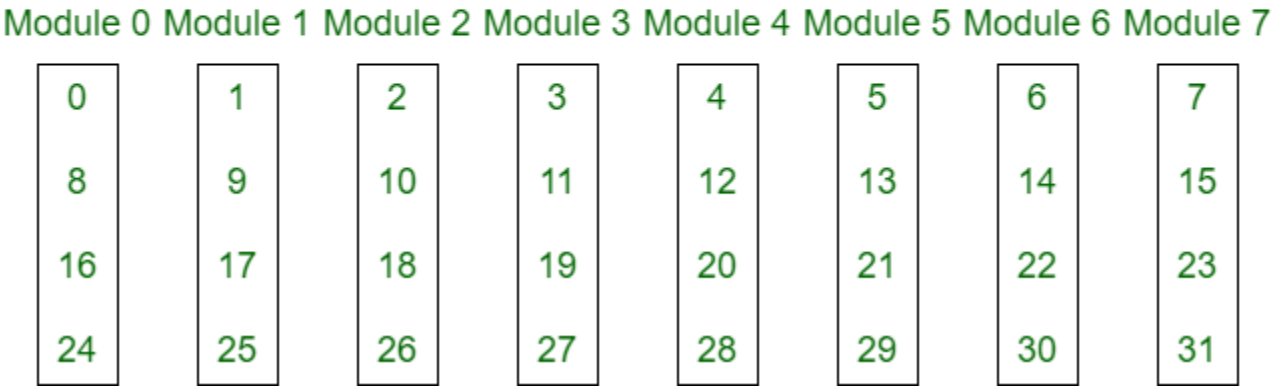
Classification of Memory Interleaving:
Memory interleaving is classified into two types:
1. High Order Interleaving –
In high-order interleaving, the most significant bits of the address select the memory chip. The least significant bits are sent as addresses to each chip. One problem is that consecutive addresses tend to be in the same chip. The maximum rate of data transfer is limited by the memory cycle time.
It is also known as Memory Banking.

Address bits	25-22	21-0
Use	Bank Select	Address to the chip

Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

2. Low Order Interleaving –
In low-order interleaving, the least significant bits select the memory bank (module). In this, consecutive memory addresses are in different memory modules. This allows memory access at much faster rates than allowed by the cycle time.

Address bits	25-4	3-0
Use	Address to the chip	Bank Select



Hierarchical memory organization:

Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.

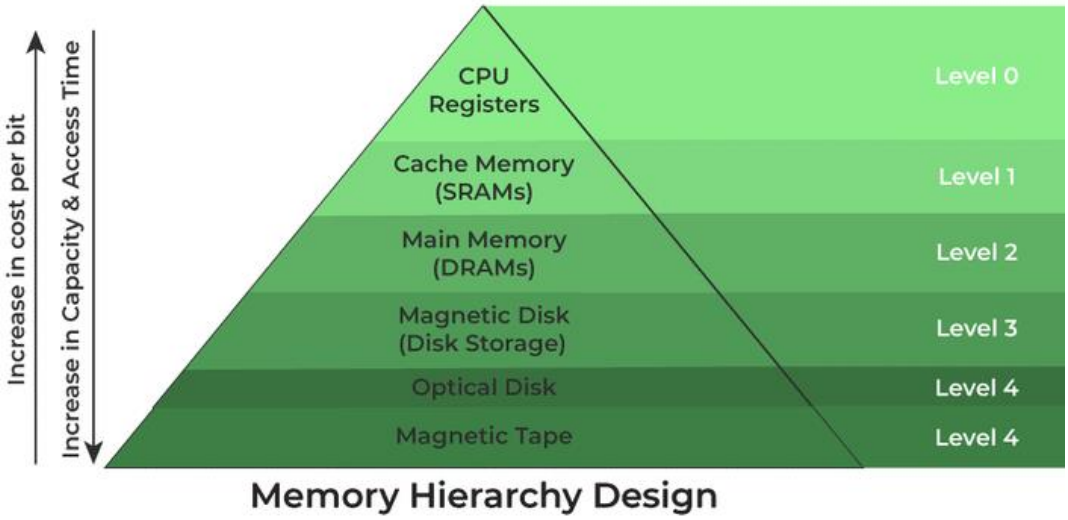
Memory Hierarchy is one of the most required things in Computer Memory as it helps in optimizing the memory available in the computer. There are multiple levels present in the memory, each one having a different size, different cost, etc. Some types of memory like cache, and main memory are faster as compared to other types of memory but they are having a little less size and are also costly whereas some memory has a little higher storage value, but they are a little slower. Accessing of data is not similar in all types of memory, some have faster access whereas some have slower access.

Types of Memory Hierarchy

This Memory Hierarchy Design is divided into 2 main types:

- **External Memory or Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via an I/O Module.

- **Internal Memory or Primary Memory:** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.



Memory Hierarchy Design

Memory Hierarchy Design

1. Registers

Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.

2. Cache Memory

Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

3. Main Memory

Main memory, also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.

Types of Main Memory

- **Static RAM:** Static RAM stores the binary information in flip flops and information remains valid until power is supplied. It has a faster access time and is used in implementing cache memory.
- **Dynamic RAM:** It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

4. Secondary Storage

Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD), is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

5. Magnetic Disk

Magnetic Disks are simply circular plates that are fabricated with either a metal or a plastic or a magnetized material. The Magnetic disks work at a high speed inside the computer and these are frequently used.

6. Magnetic Tape

Magnetic Tape is simply a magnetic recording device that is covered with a plastic film. It is generally used for the backup of data. In the case of a magnetic tape, the access time for a computer is a little slower and therefore, it requires some amount of time for accessing the strip.

Characteristics of Memory Hierarchy

- **Capacity:** It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
- **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
- **Performance:** Earlier when the computer system was designed without a Memory Hierarchy design, the speed gap increased between the CPU registers and Main Memory due to a large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of

the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
- **Cost Per Bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Advantages of Memory Hierarchy

- It helps in removing some destruction, and managing the memory in a better way.
- It helps in spreading the data all over the computer system.
- It saves the consumer's price and time.

System-Supported Memory Standards

According to the memory Hierarchy, the system-supported memory standards are defined below:

Level	1	2	3	4
Name	Register	Cache	Main Memory	Secondary Memory
Size	<1 KB	less than 16 MB	<16GB	>100 GB
Implementation	Multi-ports	On-chip/SRAM	DRAM (capacitor memory)	Magnetic
Access Time	0.25ns to 0.5ns	0.5 to 25ns	80ns to 250ns	50 lakh ns
Bandwidth	20000 to 1 lakh MB	5000 to 15000	1000 to 5000	20 to 150

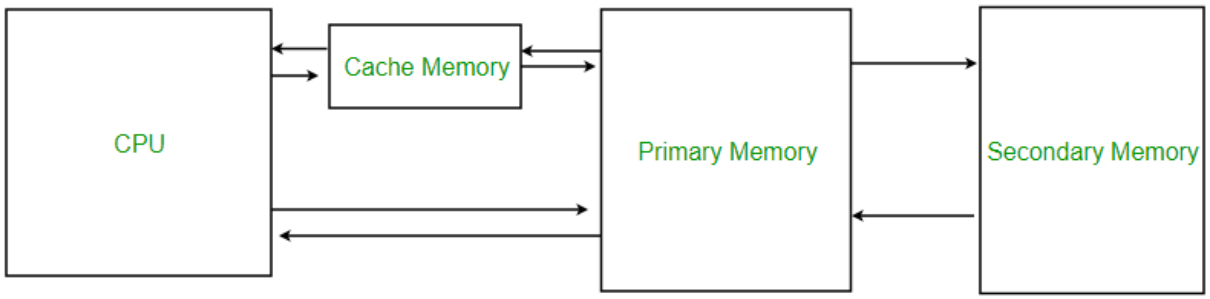
Level	1	2	3	4
Managed by	Compiler	Hardware	Operating System	Operating System
Backing Mechanism	From cache	from Main Memory	from Secondary Memory	from ie

Cache memory:

Cache Memory is a special very high-speed memory. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data. The most important use of cache memory is that it is used to reduce the average time to access data from the main memory.

Characteristics of Cache Memory

- Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.
- Cache Memory holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Cache memory is costlier than main memory or disk memory but more economical than CPU registers.
- Cache Memory is used to speed up and synchronize with a high-speed CPU.



Cache Memory

Levels of Memory

- **Level 1 or Register:** It is a type of memory in which data is stored and accepted that are immediately stored in the CPU. The most commonly used register is Accumulator, Program counter, Address Register, etc.
- **Level 2 or Cache memory:** It is the fastest memory that has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory:** It is the memory on which the computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory:** It is external memory that is not as fast as the main memory but data stays permanently in this memory.

Cache Performance

When the processor needs to read or write a location in the main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a Cache Hit has occurred and data is read from the cache.
- If the processor does not find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from the main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

```
Hit Ratio(H) = hit / (hit + miss) =  no. of hits/total accesses

Miss Ratio = miss / (hit + miss) =  no. of miss/total accesses = 1 -
hit ratio(H)
```

We can improve Cache performance using higher cache block size, and higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Mapping

There are three different types of mapping used for the purpose of cache memory which is as follows:

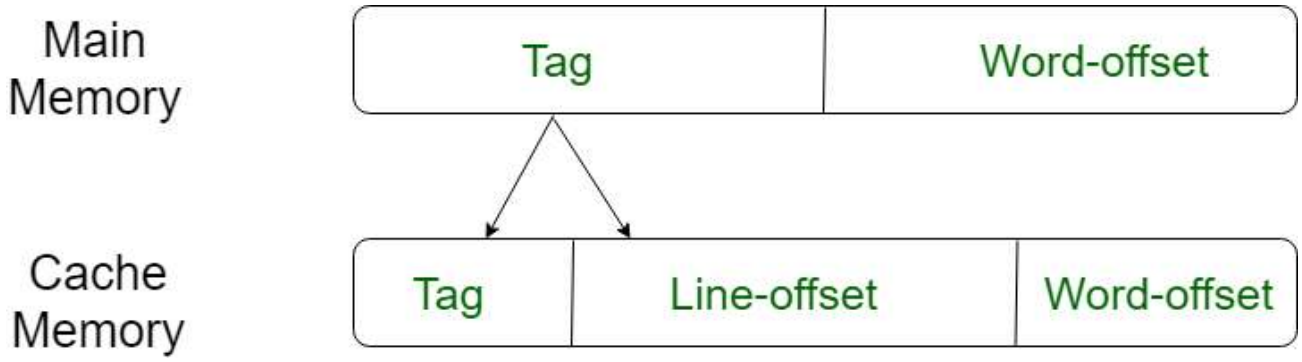
- Direct Mapping
- Associative Mapping
- Set-Associative Mapping

1. Direct Mapping

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is

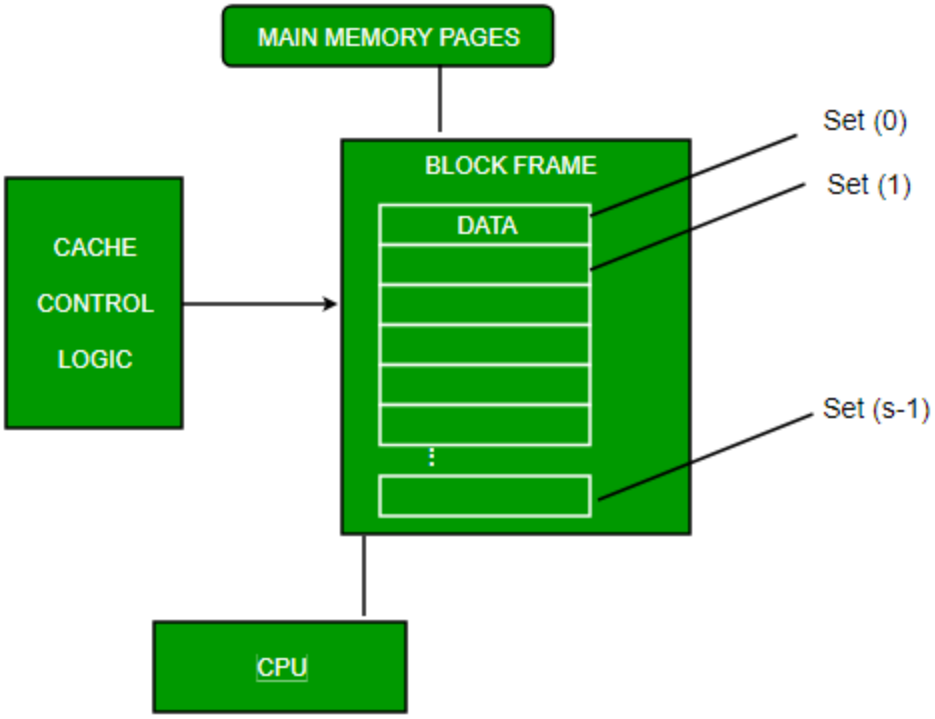
used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

```
i = j modulo m

where

i = cache line number

j = main memory block number

m = number of lines in the cache
```



Direct Mapping

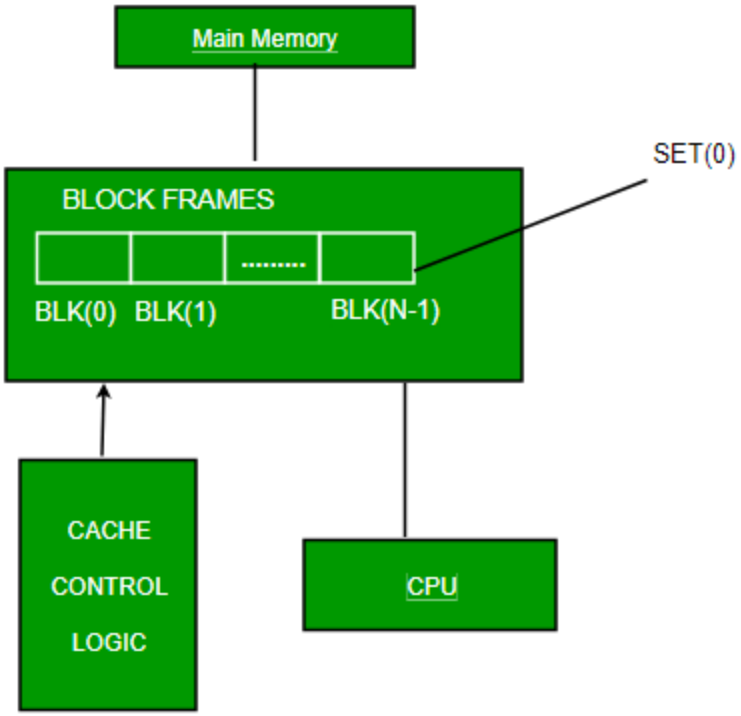
For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (the most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache. Line offset is index bits in the direct mapping.



Direct Mapping – Structure

2. Associative Mapping

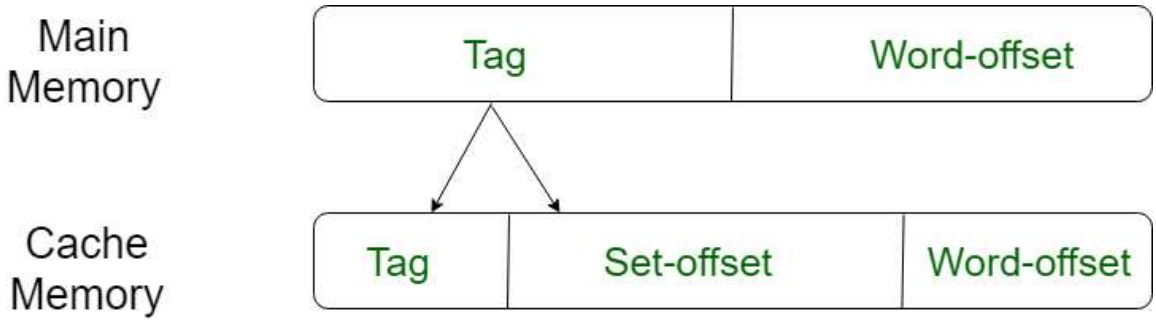
In this type of mapping, associative memory is used to store the content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and most flexible mapping form. In associative mapping, the index bits are zero.



Associative Mapping – Structure

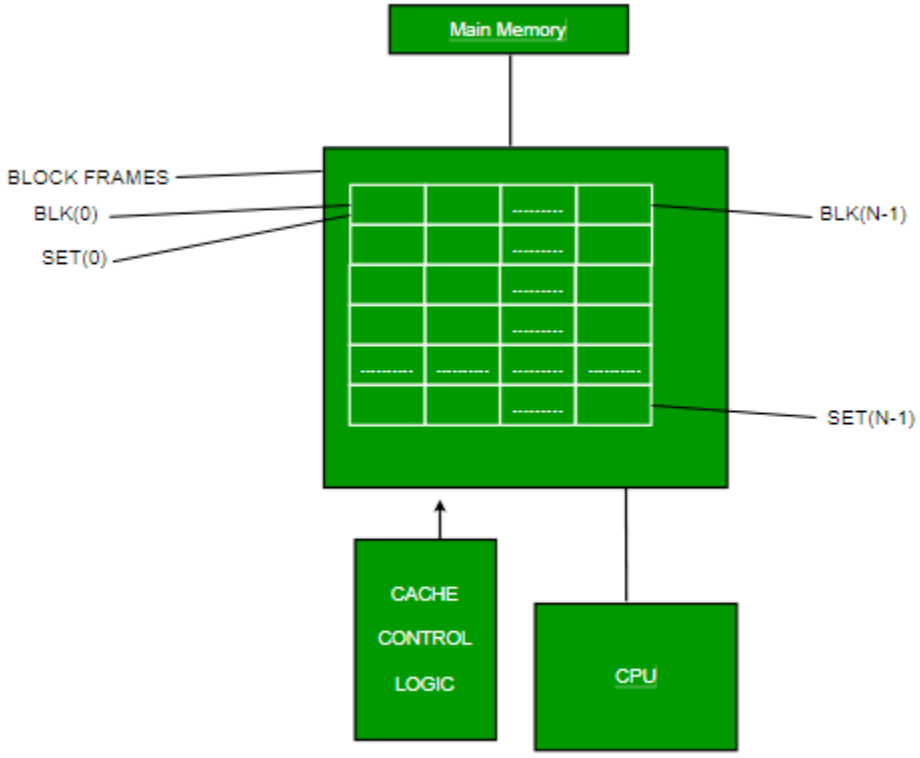
3. Set-Associative Mapping

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques. In set associative mapping the index bits are given by the set offset bits. In this case, the cache consists of a number of sets, each of which consists of a number of lines.



Relationships in the Set-Associative Mapping can be defined as:

```
m = v * k
i= j mod v
where
i = cache set number
j = main memory block number
v = number of sets
m = number of lines in the cache number of sets
k = number of lines in each set
```



Set-Associative Mapping – Structure

.

Application of Cache Memory

Here are some of the applications of Cache Memory.

1. **Primary Cache:** A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.

2. **Secondary Cache:** Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.
3. **Spatial Locality of Reference:** Spatial Locality of Reference says that there is a chance that the element will be present in close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
4. **Temporal Locality of Reference:** Temporal Locality of Reference uses the Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load the word in the main memory but the complete page fault will be loaded because the spatial locality of reference rule says that if you are referring to any word next word will be referred to in its register that's why we load complete page table so the complete block will be loaded.

Advantages of Cache Memory

- Cache Memory is faster in comparison to main memory and secondary memory.
- Programs stored by Cache Memory can be executed in less time.
- The data access time of Cache Memory is less than that of the main memory.
- Cache Memory stored data and instructions that are regularly used by the CPU, therefore it increases the performance of the CPU.

Disadvantages of Cache Memory

- Cache Memory is costlier than primary memory and secondary memory.
- Data is stored on a temporary basis in Cache Memory.
- Whenever the system is turned off, data and instructions stored in cache memory get destroyed.
- The high cost of cache memory increases the price of the Computer System.

cache size vs. block size:

1. **Cache Size:** It seems that moderately tiny caches will have a big impact on performance.
   `Cache size = Cache capacity`

2. **Block Size:** Block size is the unit of information changed between cache and main memory. As the block size will increase from terribly tiny to larger sizes, the hit magnitude relation can initially increase as a result of the principle of locality.the high chance that knowledge within the neck of the woods of a documented word square measure possible to be documented within the close to future. As the block size increases, a lot of helpful knowledge square measure brought into the cache. The hit magnitude

relation can begin to decrease, however, because the block becomes even larger and also the chance of victimization the new fetched knowledge becomes but the chance of reusing the information that ought to be abstracted of the cache to form area for the new block.

```
Block size= Cache block size = cache line size = line size
```

Mapping functions:

Mapping functions are a group of functions that could be applied successively to one or more lists of elements. The results of applying these functions to a list are placed in a new list and that new list is returned.

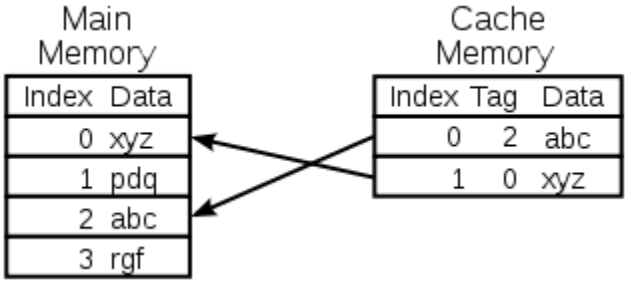
For example, the **mapcar** function processes successive elements of one or more lists.

The first argument of the mapcar function should be a function and the remaining arguments are the list(s) to which the function is applied.

The argument function is applied to the successive elements that results into a newly constructed list. If the argument lists are not equal in length, then the process of mapping stops upon reaching the end of the shortest list. The resulting list will have the same number of elements as the shortest input list.

Replacement algorithms:

Caching is the process of storing some data near where It's supposed to be used rather than accessing them from an expensive origin, every time a request comes in.



Cache replacement algorithms do just that. They decide which objects can stay and which objects should be evicted.

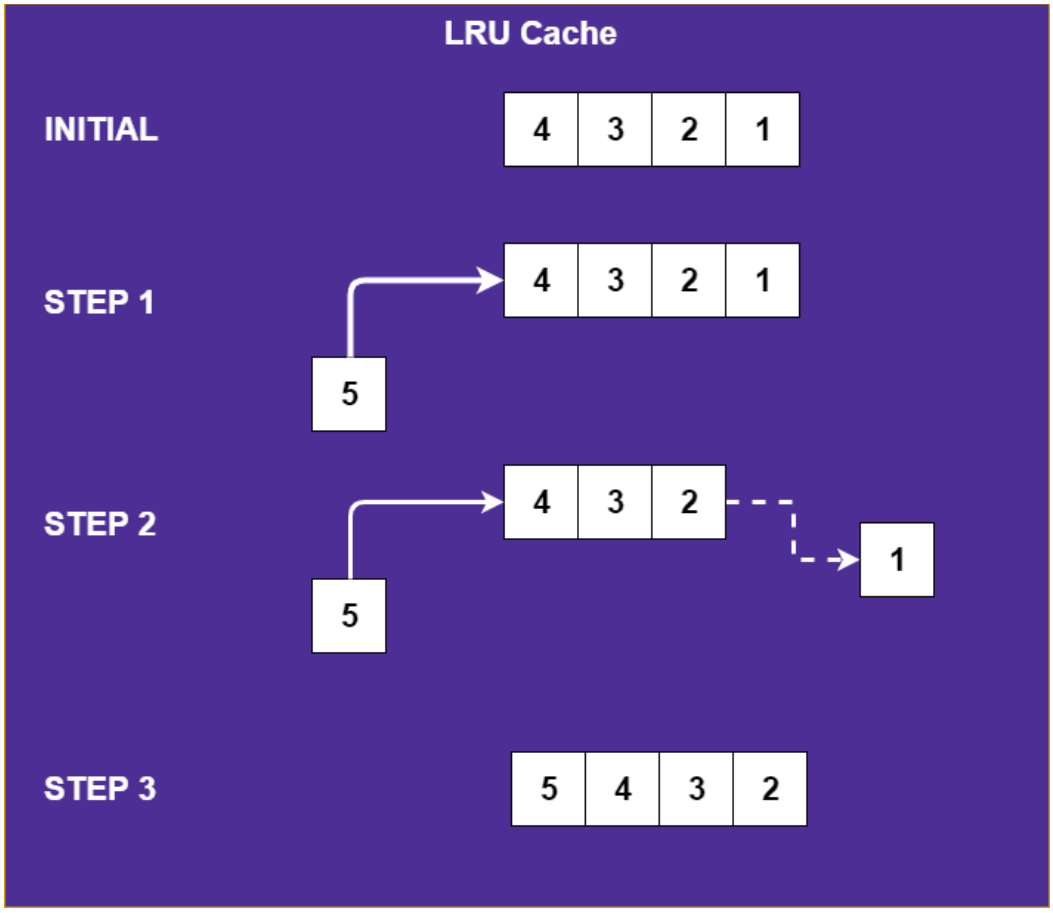
LRU

The least recently used (LRU) algorithm is one of the most famous cache replacement algorithms and for good reason!

As the name suggests, LRU keeps the least recently used objects at the top and evicts objects that haven't been used in a while if the list reaches the maximum capacity.

So it's simply an ordered list where objects are moved to the top every time they're accessed; pushing other objects down.

LRU is simple and providers a nice cache-hit rate for lots of use-cases.



LFU

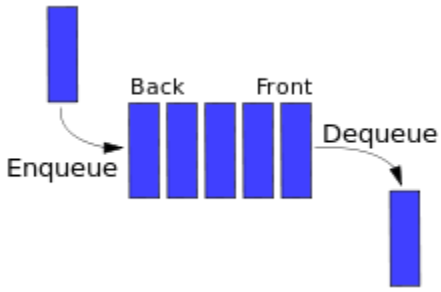
the least frequently used (LFU) algorithm works similarly to LRU except it keeps track of how many times an object was accessed instead of how recently it was accessed.

Each object has a counter that counts how many times it was accessed. When the list reaches the maximum capacity, objects with the lowest counters are evicted.

LFU has a famous problem. Imagine an object was repeatedly accessed for a short period only. Its counter increases by a magnitude compared to others so it's very hard to evict this object even if it's not accessed for a long time.

FIFO

FIFO (first-in-first-out) is also used as a cache replacement algorithm and behaves exactly as you would expect. Objects are added to the queue and are evicted with the same order. Even though it provides a simple and low-cost method to manage the cache but even the most used objects are eventually evicted when they're old enough.



Random Replacement (RR)

This algorithm randomly selects an object when it reaches maximum capacity. It has the benefit of not keeping any reference or history of objects and being very simple to implement at the same time.

This algorithm has been used in ARM processors and the famous Intel i860.

Write policies:

A cache's write policy is the behavior of a cache while performing a write operation. A cache's write policy plays a central part in all the variety of different characteristics exposed by the cache.

To make sure write operation is performed carefully, we can adopt two cache write methods:

1. **Write-through policy**
2. **Write-back policy**

Write-through policy

Write-through policy is the most commonly used methods of writing into the cache memory. In write-through method when the cache memory is updated simultaneously the main memory is also updated. Thus at any given time, the main memory contains the same data which is available in the cache memory.

It is to be noted that, write-through technique is a ***slow process*** as everytime it needs to access main memory.

Write-back policy

Write-back policy can also be used for cache writing.
During a write operation only the cache location is updated while following write-back method. When update in cache occurs then updated location is marked by a flag. The flag is known as modified or dirty bit.

When the word is replaced from the cache, it is written into main memory if its flag bit is set. The logic behind this technique is based on the fact that during a cache write operation, the word present in the cache may be accessed several times (temporal locality of reference). This method helps reduce the number of references to main memory.

The only ***limitation to write-back technique*** is that inconsistency may occur while adopting this technique due to two different copies of the same data, one in cache and other in main memory.