# CSPC406-Computer Organization and Architecture

## Unit 1-Introduction

**Functional Units**

Types and Functional Unit of Computers

### 1.1 Computer types

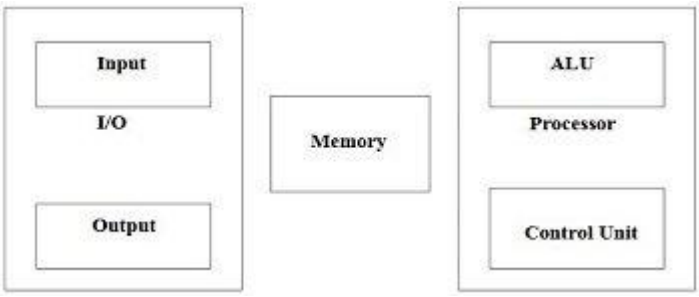
A computer can be defined as a fast electronic calculating machine that accepts the (data) digitized input information process it as per the list of internally stored instructions and produces the resulting information. List of instructions are called programs and internal storage is called computer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

The different types of computers are:

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

### 1.2 Functional unit

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




**Functional units of computer**

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

program. i.e. into machine language. Finally the results are sent to the outside world through output device. All of these actions are coordinated by the control unit.

Input unit: The source program/high level language program/coded information/simple data is fed to a computer through input devices; keyboard is a most common type. Whenever a key is pressed, one corresponding word or number is translated into its equivalent binary code over a cable and fed either to memory or process. Joysticks, trackballs, mouse, and scanners are other input devices.

Memory unit: Its function into store programs and data.
It is basically to two types
1.Primary memory
2.Secondary memory

**1. Primary memory:** Is the one exclusively associated with the processor and operates at the electronics speeds programs must be stored in this memory while they are being executed. The memory contains a large number of semiconductors storage cells.Each ALU Processor Control Unit capable of storing one bit of information. These are processed in a group of fixed site called word. To provide easy access to a word in memory, a distinct address is associated with each word location. Addresses are numbers that identify memory location. Number of bits in each word is called word length of the computer. Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the control of processor. Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called random-access memory (RAM). The time required to access one word in called memory access time. Memory which is only readable by the user and contents of which can't be altered is called read only memory (ROM) it contains operating system. Caches are the small fast RAM units, which are coupled with the processor and are often contained on the same IC chip to achieve high performance. Although primary storage is essential it tends to be expensive.

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

Examples:Magnetic disks & tapes, optical disks (ie CD-ROM's), floppies etc.,
Arithmetic logic unit (ALU): Most of the computer operators are executed in ALU of the processor like addition, subtraction, division, multiplication, etc. the operands are brought into the ALU from memory and stored in high speed storage elements called register.Then according to the instructions the operation is performed in the required sequence. The control and the ALU are may times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as keyboards, displays, magnetic and optical disks, sensors and other mechanical controllers. Output unit: - These actually are the counterparts of input unit. Its basic function is to send the processed results to the outside world.

Examples:-

Printer, speakers, monitor etc. Control unit:-It effectively is the nerve center that sends signals to other units and senses their states. The actual timing signals that govern the transfer of data between input unit, processor, memory and output unit are generated by the control unit.

**Basic Operational Concepts :**

A Computer has five functional in dependant units like Input Unit, Memory Unit, Arithmetic & Logic Unit, Output Unit, Control Unit.

**Input Unit :-**

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

**Memory Unit :-**

It stores programs as well as data and there are two types- Primary and Secondary Memory

Primary Memory is quite fast which works at electronic speed. Programs should be stored in memory before getting executed. Random Access Memory are those memory in which location can be accessed in a shorter period of time after specifying the address. Primary memory is essential but expensive so we went for secondary memory which is quite cheaper. It is used when large amount of data & programs are needed to store, particularly the information that we dont access very frequently. Ex- Magnetic Disks, Tapes

**Arithmetic & Logic Unit :-**

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

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

**Control Unit** :– It tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory

**Bus Structures and its Performance :**

Bus is a communication system that transfers data between components inside a computer, or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocols.[2]

Early computer buses were parallel electrical wires with multiple hardware connections, but the term is now used for any physical arrangement that provides the same logical function as a parallel electrical bus. Modern computer buses can use both parallel and bit serial connections, and can be wired in either a multidrop (electrical parallel) or daisy chain topology, or connected by switched hubs, as in the case of USB.Aspects of **performance**. **Computer performance metrics** (things to measure) include availability, response time, channel capacity, latency, completion time, service time, bandwidth, throughput, relative **efficiency**,

scalability, **performance** per watt, compression ratio, instruction path length and speed up.The most common **measure** of **CPU** speed is the clock speed, which is **measured** in MHz or GHz. One GHz equals 1,000 MHz, so a speed of 2.4 GHz could also be expressed as 2,400 MHz. The higher the clock speed, the more operations the **CPU** can execute per second.

## INSTRUCTIONS AND INSTRUCTION SEQUENCING

Four types of operations

1. Data transfer between memory and processor registers.
2. Arithmetic & logic operations on data
3. Program sequencing & control
4. I/O transfers.

### 1) Register transfer notations(RTN)

R3<–[R1]+[R2]

- Right hand side of RTN-denotes a value.
- Left hand side of RTN-name of a location.

### 2) Assembly language notations(ALN)

Add R1, R2, R3

- Adding contents of R1, R2 & place sum in R3.

### 3) Basic instruction types-4 types

- **Three address instructions**– Add A,B,C

A, B-source operands

C-destination operands

- **Two address instructions**-Add A,B

B <–[A] + [B]

- **One address instructions** –Add A

Add contents of A to accumulator & store sum back to accumulator.

- **Zero address instructions**

Instruction store operands in a structure called push down stack.

### 4) Instruction execution & straight line sequencing

- The processor control circuits use information in PC to fetch & execute instructions one at a time in order of increasing address.
- This is called straight line sequencing.
- Executing an instruction-2 phase procedures.

- 1st phase–**"instruction fetch"-**instruction is fetched from memory location whose address is in PC.
- This instruction is placed in instruction register in processor
- 2nd phase-**"instruction execute"-**instruction in IR is examined to determine which operation to be performed.

**5) Branching**

- Branch-type of instruction loads a new value into program counter.
- So processor fetches & executes instruction at this new address called "branch target"
- Conditional branch-causes a branch if a specified condition is satisfied.
- E.g. Branch>0 LOOP –conditional branch instruction .it executes only if it satisfies condition.

**6) Condition codes**

- Recording required information in individual bits called "condition code flags".
- These flags are grouped together in a special processor register called "condition code register" or "status register"
- Individual condition code flags-1 or 0.
- 4 commonly used flags.

1) N (negative)-set to 1 if result is –ve or else 0.

2) Z (zero)-set to 1 if result is 0, or else 0 .

3) V (overflow)-set to 1if arithmetic overflow occurs or else 0.

4) C(carry)-set to 1 if carry out results from operation or else 0

**Instruction Set :**

The **instruction set**, also called **ISA** (**instruction set architecture**), is part of a computer that pertains to programming, which is basically machine language. The instruction set provides commands to the processor, to tell it what it needs to do. The instruction set consists of addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O.

An example of an instruction set is the x86 instruction set, which is common to find on computers today. Different computer processors can use almost the same instruction set while still having very different internal design. Both the Intel Pentium and AMD Athlon

processors use nearly the same x86 instruction set. An instruction set can be built into the hardware of the processor, or it can be emulated in software, using an interpreter. The hardware design is more efficient and faster for running programs than the emulated software version.

**Examples of instruction set**

- **ADD** - Add two numbers together.
- **COMPARE** - Compare numbers.
- **IN** - Input information from a device, e.g., keyboard.
- **JUMP** - Jump to designated RAM address.
- **JUMP IF** - Conditional statement that jumps to a designated RAM address.
- **LOAD** - Load information from RAM to the CPU.
- **OUT** - Output information to device, e.g., monitor.
- **STORE** - Store information to RAM.

**Addressing Modes :**

The term ***addressing modes*** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand. Following are the main addressing modes that are used on various platforms and architectures.

**1) Immediate Mode**

The operand is an immediate value is stored explicitly in the instruction:

Example: SPIM ( opcode dest, source)

li $11, 3 // loads the immediate value of 3 into register $11

li $9, 8 // loads the immediate value of 8 into register $9

Example : (textbook uses instructions type like, opcode source, dest)

move #200, R0; // move immediate value 200 in register R0

**2) Index Mode**

The address of the operand is obtained by adding to the contents of the general register (called index register) a constant value. The number of the index register and

the constant value are included in the instruction code. Index Mode is used to access an array whose elements are in successive memory locations. The content of the instruction code, represents the starting address of the array and the value of the index register, and the index value of the current element. By incrementing or decrementing index register different element of the array can be accessed.

**Example: SPIM/SAL - Accessing Arrays**

```
.data
array1: .byte 1,2,3,4,5,6
.text
__start:
move $3, $0                 # $3 initialize index register with 0
add $3, $3,4                # compute the index value of the fifth element
sb $0, array1($3)   # array1[4]=0
                            # store byte 0 in the fifth element of the array
                            # index addressing mode
done
```

**3) Indirect Mode**

The effective address of the operand is the contents of a register or main memory location, location whose address appears in the instruction. Indirection is noted by placing the name of the register or the memory address given in the instruction in parentheses. The register or memory location that contains the address of the operand is a pointer. When an execution takes place in such mode, instruction may be told to go to a specific address. Once it's there, instead of finding an operand, it finds an address where the operand is located.

NOTE:

Two memory accesses are required in order to obtain the value of the operand (fetch operand address and fetch operand value).

**Example:** (textbook) ADD (A), R0

(address A is embedded in the instruction code and (A) is the operand address = pointer variable)

**Example: SPIM - simulating pointers and indirect register addressing**

The following "C" code:

```
int *alpha=0x00002004, q=5;
*alpha = q;
```

could be translated into the following assembly code:

```
alpha: .word 0x00002004 # alpha is and address variable # address value is
0x00002004
```

```
q: .word 5
....
lw $10,q          # load word value from address q in into $10
                  # $10 is 5
lw $11,alpha      # $11 gets the value 0x0002004
                  # this is similar with a load immediate address value
sw $10,($11)      # store value from register $10 at memory location
                  # whose address is given by the contents of register $11
                  # (store 5 at address 0x00002004)
```

**Example: SPIM/SAL - - array pointers and indirect register addressing**

```
.data
array1: .byte 1,2,3,4,5,6
.text
__start:
la $3, array1     # array1 is direct addressing mode
add $3, $3,4      # compute the address of the fifth element
sb $0, ($3)       # array1[4]=0 , byte accessing
                  # indirect addressing mode
done
```

**4) Absolute (Direct) Mode**

The address of the operand is embedded in the instruction code.

**Example: (SPIM)**

```
beta: .word 2000

lw $11, beta      # load word (32 -bit quantity) at address beta into register $11
                  # address of the word is embedded in the instruction code
                  # (register $11 will receive value 2000)
```

**5) Register Mode**

The name (the number) of the CPU register is embedded in the instruction. The register contains the value of the operand. The number of bits used to specify the register depends on the total number of registers from the processor set.

**Example (SPIM)**

```
add $14,$14,$13       # add contents of register $13 plus contents of
                        # register $14 and save the result in register $14
```

No memory access is required for the operand specified in register mode.

**6) Displacement Mode**

Similar to index mode, except instead of a index register a base register will be used. Base register contains a pointer to a memory location. An integer (constant) is also referred to as a displacement. The address of the operand is obtained by adding the contents of the base register plus the constant. The difference between index mode and displacement mode is in the number of bits used to represent the constant. When the constant is represented a number of bits to access the memory, then we have index mode. Index mode is more appropriate for array accessing; displacement mode is more appropriate for structure (records) accessing.

**Example: SPIM/SAL - Accessing fields in structures**

```
.data
student: .word 10000 #field code
.ascii "Smith" #field name
.byte # field test
.byte 80,80,90,100 # fields hw1,hw2,hw3,hw4
.text
__start:
la $3, student      # load address of the structure in $3
                    # $3 base register
add $17,$0,90       # value 90 in register $17
                    # displacement of field "test" is 9 bytes
                    #
sb $17, 9($3)       # store contents of register $17 in field "test"
                    # displacement addressing mode
done
```

**7) Autoincrement /Autodecrement Mode**

A special case of indirect register mode. The register whose number is included in the instruction code, contains the address of the operand. Autoincrement Mode = after operand addressing , the contents of the register is incremented. Decrement Mode = before operand addressing, the contents of the register is decrement.

**Example: SPIM/SAL - - simulating autoincrement/autodecrement addressing mode**

(**MIPS** has no autoincrement/autodecrement mode)

```
lw $3, array1($17)          #load in reg. $3 word at address array1($17)
addi $17,$17,4              #increment address (32-bit words) after accessing
                                #operand this can be re-written in a "autoincrement
like mode":
lw+ $3,array1($17)              # lw+ is not a real MIPS instruction
subi $17,$17,4                  # decrement address before accessing the operand
lw $3,array1($17)
```

**NOTE:** the above sequence can be re-rewritten proposing an "autodecrement instruction", **not real** in MIPS architecture.

**RISC AND CISC :**

**Reduced Set Instruction Set Architecture (RISC) –**
The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data.

**Complex Instruction Set Architecture (CISC) –**
The main idea is to make hardware complex as a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it.

Both approaches try to increase the CPU performance

Earlier when programming was done using assembly language, a need was felt to make instruction do more task because programming in assembly was tedious and error prone due to which CISC architecture evolved but with uprise of high level language dependency on assembly reduced RISC architecture prevailed.

**Characteristic of RISC –**

1. Simpler instruction, hence simple instruction decoding.
2. Instruction come under size of one word.
3. Instruction take single clock cycle to get executed.
4. More number of general purpose register.
5. Simple Addressing Modes.
6. Less Data types.
7. Pipeling can be achieved.

**Characteristic of CISC –**

1. Complex instruction, hence complex instruction decoding.
2. Instruction are larger than one word size.
3. Instruction may take more than single clock cycle to get executed.
4. Less number of general purpose register as operation get performed in memory itself.
5. Complex Addressing Modes.
6. More Data types.

**Example –** Suppose we have to add two 8-bit number:

- **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
- **RISC approach:** Here programmer will write first load command to load data in registers then it will use suitable operator and then it will store result in desired location.

# Unit 2- Fundamental Concepts

**ALU Design:**

An **arithmetic logic unit** (**ALU**) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.

**Execution of a Complete Instruction :**

**The Fetch Cycle –**

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).

Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.

Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)

Step 3: The content of the MBR is moved to the instruction register(IR).

**The Indirect Cycles –**

Once an instruction is fetched, the next step is to fetch source operands. Source Operand is being fetched by indirect addressing( it can be fetched by any addressing mode, here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following micro-operations takes place:-

Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.

Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)

Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

**The Execute Cycle**
The other three cycles(Fetch, Indirect and Interrupt) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.
Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.

We begin with the IR containing the ADD instruction.
Step 1: The address portion of IR is loaded into the MAR.
Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.
Step 3: Now, the contents of R and MBR are added by the ALU.

**The Interrupt Cycle**:
At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The natare of this cycle varies greatly from one machine to another.
Lets take a sequence of micro-operation:-

Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.
Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.
PC is loaded with the address of the start of the interrupt-processing routine.
Step 3: MBR, containing the old value of PC, is stored in memory.

**Multiple Bus Organization :**

**Multiple bus organization** is primarily used in industrial systems. In this structure, various devices that have different transfer rates can be connected. At the same time, maximum throughput is maintained. Some benefits of **multiple bus** architecture are: * Allows more number of devices to be connected to the computer.

**Hardwired Control Unit –**
The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.
- RISC architecture is based on hardwired control unit

**Micro-programmed Control Unit –**

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

**Some Important Terms –**

1. **Control Word :** A control word is a word whose individual bits represent various control signals.
2. **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
3. **Micro-instruction :** Individual control words in this micro-routine are referred to as microinstructions.
4. **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
5. **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.

**Types of Micro-programmed Control Unit –** Based on the type of Control Word stored in the Control Memory (CM), it is classified into two types :

**1. Horizontal Micro-programmed control Unit :**
The control signals are represented in the decoded binary format that is 1 bit/CS. Example: If 53 Control signals are present in the processor than 53 bits are required. More than 1 control signal can be enabled at a time.

- It supports longer control word.
- It is used in parallel processing applications.
- It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.
- It requires no additional hardware(decoders). It means it is faster than Vertical Microprogrammed.
- It is more flexible than vertical microprogrammed

**2. Vertical Micro-programmed control Unit :**
The control signals re represented in the encoded binary format. For N control signals- Log2(N) bits are required.

- It supports shorter control words.
- It supports easy implementation of new conrol signals therefore it is more flexible.
- It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
- Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal microprogrammed.
- It is less flexible than horizontal but more flexible than that of hardwired control unit.

**Nano Programming :**

**Nano programming**: In, most microprogrammed processors, an instruction fetched from memory is interpreted by a micro **program** stored in a single control memory CM. ... They use second control memory called a **nano** control memory (nCM).

Nano programmed machine :

- Instance : Assume that a system is being designed with 200 control points and 2048 microinstructions
- Suppose that only 256 different combinations of control points are ever used
- A single-level control memory would need 2048x200=409,600 storage bits
- A nano programmed system would utilize
- Microstore of size 2048x8=16k
- Nanostore of size 256x200=51200
- Total size = 67,584 storage bits
- Nano programming has been used in many CISC microprocessors

# Unit -3

# Memory

**Semiconductor RAM and ROM:**

**RAM:**

**Random-access memory** is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code.

RAM is of two types −

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

## Static RAM (SRAM)

The word **static** indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis.

There is extra space in the matrix, hence SRAM uses more chips than DRAM for the same amount of storage space, making the manufacturing costs higher. SRAM is thus used as cache memory and has very fast access.

### Characteristic of Static RAM

- Long life
- No need to refresh
- Faster
- Used as cache memory
- Large size
- Expensive
- High power consumption

## Dynamic RAM (DRAM)

DRAM, unlike SRAM, must be continually **refreshed** in order to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory as it is cheap and small. All DRAMs are made up of memory cells, which are composed of one capacitor and one transistor.

**Characteristics of Dynamic RAM**

- Short data lifetime
- Needs to be refreshed continuously
- Slower as compared to SRAM
- Used as RAM
- Smaller in size
- Less expensive
- Less power consumption

**ROM:**

ROM stands for **Read Only Memory**. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as **bootstrap**. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.

**MROM (Masked ROM)**

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

**PROM (Programmable Read Only Memory)**

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

**EPROM (Erasable and Programmable Read Only Memory)**

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

**EEPROM (Electrically Erasable and Programmable Read Only Memory)**

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

**Advantages of ROM**

The advantages of ROM are as follows −

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing
- Contents are always known and can be verified

**Cache Memory :**

Cache memory is a chip-based computer component that makes retrieving data from the computer's memory more efficient. It acts as a temporary storage area that the computer's processor can retrieve data from easily. This temporary storage area, known as a cache, is more readily available to the processor than the computer's main memory source, typically some form of DRAM.

## Types of cache memory

Cache memory is fast and expensive. Traditionally, it is categorized as "levels" that describe its closeness and accessibility to the microprocessor. There are three general cache levels:

**L1 cache**, or primary cache, is extremely fast but relatively small, and is usually embedded in the processor chip as CPU cache.

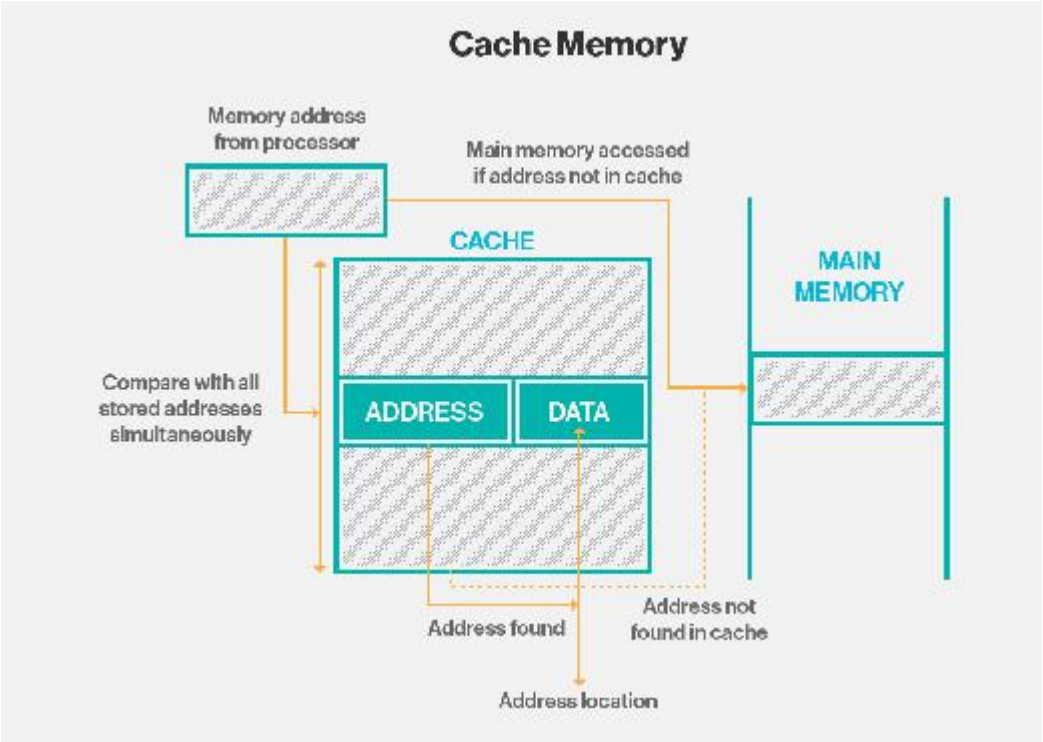
**L2 cache**, or secondary cache, is often more capacious than L1. L2 cache may be embedded on the CPU, or it can be on a separate chip or coprocessor and have a high-speed alternative system bus connecting the cache and CPU. That way it doesn't get slowed by traffic on the main system bus.

**Level 3 (L3) cache** is specialized memory developed to improve the performance of L1 and L2. L1 or L2 can be significantly faster than L3, though L3 is usually double the speed of DRAM. With multicore processors, each core can have dedicated L1 and L2 cache, but they can share an L3 cache. If an L3 cache references an instruction, it is usually elevated to a higher level of cache.

In the past, L1, L2 and L3 caches have been created using combined processor and motherboard components. Recently, the trend has been toward consolidating all three levels of memory caching on the CPU itself. That's why the primary means for

increasing cache size has begun to shift from the acquisition of a specific motherboard with different chipsets and bus architectures to buying a CPU with the right amount of integrated L1, L2 and L3 cache.

Contrary to popular belief, implementing flash or more dynamic RAM (DRAM) on a system won't increase cache memory. This can be confusing since the terms memory caching (hard disk buffering) and cache memory are often used interchangeably. Memory caching, using DRAM or flash to buffer disk reads, is meant to improve storage I/O by caching data that is frequently referenced in a buffer ahead of slower magnetic disk or tape. Cache memory, on the other hand, provides read buffering for the CPU.




**Cache memory mapping**

Caching configurations continue to evolve, but cache memory traditionally works under three different configurations:

- **Fully associative cache mapping** is similar to direct mapping in structure but allows a memory block to be mapped to any cache location rather than to a prespecified cache memory location as is the case with direct mapping.
- **Set associative cache mapping** can be viewed as a compromise between direct mapping and fully associative mapping in which each block is mapped to a subset of cache locations. It is sometimes called *N-way set associative mapping*,

which provides for a location in main memory to be cached to any of "N" locations in the L1 cache.

**Data writing policies**

Data can be written to memory using a variety of techniques, but the two main ones involving cache memory are:

- **Write-through.** Data is written to both the cache and main memory at the same time.
- **Write-back.** Data is only written to the cache initially. Data may then be written to main memory, but this does not need to happen and does not inhibit the interaction from taking place.

The way data is written to the cache impacts data consistency and efficiency. For example, when using write-through, more writing needs to happen, which causes latency upfront. When using write-back, operations may be more efficient, but data may not be consistent between the main and cache memories.

One way a computer determines data consistency is by examining the dirty bit in memory. The dirty bit is an extra bit included in memory blocks that indicates whether the information has been modified. If data reaches the processor's register file with an active dirty bit, it means that it is not up to date and there are more recent versions elsewhere. This scenario is more likely to happen in a write-back scenario, because the data is written to the two storage areas asynchronously.

**Specialization and functionality**

In addition to instruction and data caches, other caches are designed to provide specialized system functions. According to some definitions, the L3 cache's shared design makes it a specialized cache. Other definitions keep the instruction cache and the data cache separate and refer to each as a specialized cache.

Translation lookaside buffers (TLBs) are also specialized memory caches whose function is to record virtual address to physical address translations.

Still other caches are not, technically speaking, memory caches at all. Disk caches, for instance, can use DRAM or flash memory to provide data caching similar to what memory caches do with CPU instructions. If data is frequently accessed from the disk, it is cached into DRAM or flash-based silicon storage technology for faster access time and response.

Specialized caches are also available for applications such as web browsers, databases, network address binding and client-side Network File System protocol support. These types of caches might be distributed across multiple networked hosts to provide greater scalability or performance to an application that uses them.

**Locality**

The ability of cache memory to improve a computer's performance relies on the concept of locality of reference. Locality describes various situations that make a system more predictable. Cache memory takes advantage of these situations to create a pattern of memory access that it can rely upon.

There are several types of locality. Two key ones for cache are:

- **Temporal locality.** This is when the same resources are accessed repeatedly in a short amount of time.
- **Spatial locality.** This refers to accessing various data or resources that are near each other.

**Performance**

Cache memory is important because it improves the efficiency of data retrieval. It stores program instructions and data that are used repeatedly in the operation of programs or information that the CPU is likely to need next. The computer processor can access this information more quickly from the cache than from the main memory. Fast access to these instructions increases the overall speed of the program.

Aside from its main function of improving performance, cache memory is a valuable resource for *evaluating* a computer's overall performance. Users can do this by looking at cache's hit-to-miss ratio. Cache hits are instances in which the system successfully retrieves data from the cache. A cache miss is when the system looks for the data in the cache, can't find it, and looks somewhere else instead. In some cases, users can improve the hit-miss ratio by adjusting the cache memory block size -- the size of data units stored.

Improved performance and ability to monitor performance are not just about improving general convenience for the user. As technology advances and is increasingly relied upon in mission-critical scenarios, having speed and reliability becomes crucial. Even a few milliseconds of latency could potentially lead to enormous expenses, depending on the situation.

**Cache vs. main memory**

DRAM serves as a computer's main memory, performing calculations on data retrieved from storage. Both DRAM and cache memory are volatile memories that lose their contents when the power is turned off. DRAM is installed on the motherboard, and the CPU accesses it through a bus connection.

DRAM is usually about half as fast as L1, L2 or L3 cache memory, and much less expensive. It provides faster data access than flash storage, hard disk drives (HDD) and tape storage. It came into use in the last few decades to provide a place to store frequently accessed disk data to improve I/O performance.

DRAM must be refreshed every few milliseconds. Cache memory, which also is a type of random access memory, does not need to be refreshed. It is built directly into the CPU to give the processor the fastest possible access to memory locations and provides nanosecond speed access time to frequently referenced instructions and data. SRAM is faster than DRAM, but because it's a more complex chip, it's also more expensive to make.

**Cache vs. virtual memory**

A computer has a limited amount of DRAM and even less cache memory. When a large program or multiple programs are running, it's possible for memory to be fully used. To compensate for a shortage of physical memory, the computer's operating system (OS) can create virtual memory.

To do this, the OS temporarily transfers inactive data from DRAM to disk storage. This approach increases virtual address space by using active memory in DRAM and inactive memory in HDDs to form contiguous addresses that hold both an application and its data. Virtual memory lets a computer run larger programs or multiple programs simultaneously, and each program operates as though it has unlimited memory.

In order to copy virtual memory into physical memory, the OS divides memory into page files or swap files that contain a certain number of addresses. Those pages are stored on a disk and when they're needed, the OS copies them from the disk to main memory and translates the virtual memory address into a physical one. These translations are handled by a memory management unit (MMU).

**Virtual Memory :**

Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available.

Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.

The transformation of data from main memory to cache memory is called mapping. There are 3 main types of mapping:

- Associative Mapping
- Direct Mapping
- Set Associative Mapping

**Associative Mapping**

The associative memory stores both address and data. The address value of 15 bits is 5 digit octal numbers and data is of 12 bits word in 4 digit octal number. A CPU address of 15 bits is placed in argument register and the associative memory is searched for matching address.

**Direct Mapping**

The CPU address of 15 bits is divided into 2 fields. In this the 9 least significant bits constitute the **index** field and the remaining 6 bits constitute the **tag** field. The number of bits in index field is equal to the number of address bits required to access cache memory.

**Set Associative Mapping**

The disadvantage of direct mapping is that two words with same index address can't reside in cache memory at the same time. This problem can be overcome by set associative mapping.

In this we can store two or more words of memory under the same index address. Each data word is stored together with its tag and this forms a set.

**Memory Management Requirements:**

These Requirements of memory management are:

**1)Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We

may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.

2. **Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of relocation requirement increases the difficulty of satisfying the protection requirement.

Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.

**3. Sharing –** A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.

For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

**4.Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

Modules are written and compiled independently and all the references from one module to another module are resolved by `the system at run time.
Different modules are provided with different degrees of protection.
There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.

6. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system

concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.

In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

## Associative Memory

***Associative memory*** searches stored data only by the data value itself rather by an address. This type of search helps in reducing the search time by a large extent.

> *When data is accessed by data content rather than data address, then the memory is refered to as **associative memory** or **content addressable memory**.*

**How associative memory works?**
Given below is a series of steps that depicts working of associative memory:

- Data is stored at the very first empty location found in memory.
- In associative memory when data is stored at a particular location then no address is stored along with it.
- When the stored data need to be searched then only the key (i.e. data or part of data) is provided.
- A sequential search is performed in the memory using the specified key to find out the matching key from the memory.
- If the data content is found then it is set for the next reading by the memory.

**Associative memory organization**
The associative memory hardware structure consists of memory array, logic for m words with n bits per word, and several registers like input resgister, mask register, select register and output register.
The block diagram showing organization of associative memory is shown below:
image

**Functions of the registers** used in associative memory is given below:

- **Input Register (I)** hold the data that is to be written into the associative memory. It is also used to hold the data that is to be searched for. At a particular time it can hold a data containing one word of length (say n).
- **Mask Register (M)** is used to provide a mask for choosing a key or particular field in the input register's word. Since input register can hold a data of one word of length n so the maximum length of mask register can be n.
- **Select Register (S)** contains m bits, one for each memory words. When input data in I register is compared to key in m register and match is found then that particular bit is set in select register.
- **Output Register (Y)** contains the matched data word that is retrieved from associative memory.

**Advantages of associative memory**

*Advantages of associative memory* is given below:

- Associative memory searching process is fast.
- Associative memory is suitable for parallel searches.

**Disadvantages of associative memory**

*Disadvantages of associative memory* is given below:

- Associative memory is expensive than RAM

**Secondary storage devices** :

It provide a way for the computer to store information on a permanent basis. The three primary categories for storage devices are magnetic storage, optical storage and solid state storage. Examples of these are hard drives, CDs and DVDs, and flash drives.

# Unit 4

# I/O Devices

**Accessing I/O Devices :**

I/O devices accessed through I/O interface. Requirements for I/O interface: – CPU communication – Device communication – Data buffering – Control and timing – Error detection. 3. CPU Communication:• Processor sends commands to the I/O system which are generally the control signals on the control bus

**Programmed I/O :**

Programmable I/O is one of the I/O technique other than the interrupt-driven I/O and direct memory access (DMA). The programmed I/O was the most simple type of I/O technique for the exchanges of data or any types of communication between the processor and the external devices. With programmed I/O, data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time. The overall operation of the programmed I/O can be summaries as follow:

1. The processor is executing a program and encounters an instruction relating to I/O operation.
2. The processor then executes that instruction by issuing a command to the appropriate I/O module.
3. The I/O module will perform the requested action based on the I/O command issued by the processor (READ/WRITE) and set the appropriate bits in the I/O status register.
4. The processor will periodically check the status of the I/O module until it find that the operation is complete.

**Programmed I/O Mode Input Data Transfer :**

1. Each input is read after first testing whether the device is ready with the input (a state reflected by a bit in a status register).
2. The program waits for the ready status by repeatedly testing the status bit and till all targeted bytes are read from the input device.
3. The program is in busy (non-waiting) state only after the device gets ready else in wait state.

**Programmed I/O Mode Output Data Transfer:**

1. Each output written after first testing whether the device is ready to accept the byte at its output register or output buffer is empty.
2. The program waits for the ready status by repeatedly testing the status bit(s) and till all the targeted bytes are written to the device.
3. The program in busy (non-waiting) state only after the device gets ready else wait state.

**I/O Commands :**

To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command. There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:

- **Control**: Used to activate a peripheral and tell it what to do. For example, a magnetic-tape unit may be instructed to rewind or to move forward one record. These commands are tailored to the particular type of peripheral device.
- **Test**: Used to test various status conditions associated with an I/O module and its peripherals. The processor will want to know that the peripheral of interest is powered on and available for use. It will also want to know if the most recent I/O operation is completed and if any errors occurred.
- **Read**: Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer. The processor can then obtain the data item by requesting that the I/O module place it on the data bus.
- **Write**: Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

**Interrupts**

Interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high priority process requiring interruption of the current working process. In I/O devices one of the bus control lines is dedicated for this purpose and is called the *Interrupt Service Routine (ISR)*.

When a device raises an interrupt at lets say process i, the processor first completes the execution of instruction i. Then it loads the Program Counter (PC) with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process i+1.

While the processor is handling the interrupts, it must inform the device that its request has been recognized so that it stops sending the interrupt request signal. Also, saving the registers so that the interrupted process can be restored in the future, increases the delay between the time an interrupt is received and the start of the execution of the ISR. This is called Interrupt Lattency.

**Hardware Interrupts:**
In a hardware interrupt, all the devices are connected to the Interrupt Request Line. A single request line is used for all the n devices. To request an interrupt, a device closes its associated switch. When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

**Sequence of events involved in handling an IRQ:**

1. Devices raise an IRQ.
2. Processor interrupts the program currently being executed.
3. Device is informed that its request has been recognized and the device deactivates the request signal.
4. The requested action is performed.
5. Interrupt is enabled and the interrupted program is resumed.

**Handling Multiple Devices:**
When more than one device raises an interrupt request signal, then additional information is needed to decide which which device to be considered first. The following methods are used to decide which device to select: Polling, Vectored Interrupts, and Interrupt Nesting. These are explained as following below.

1. **Polling:**
   In polling, the first device encountered with with IRQ bit set is the device that is to be serviced irst. Appropriate ISR is called to service the same. It is easy to implement but a lot of time is wasted by interrogating the IRQ bit of all devices.
2. **Vectored Interrupts:**
   In vectored interrupts, a device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus. This enables the processor to identify the device that generated the interrupt. The special code can be the starting address of the ISR or where the ISR is located in memory, and is called the interrupt vector.
3. **Interrupt Nesting:**
   In this method, I/O device is organized in a priority structure. Therefore, interrupt request from a higher priority device is recognized where as request from a lower priority device is not. To implement this each process/device (even the processor). Processor accepts interrupts only from devices/processes having priority more than it.

**Direct Memory Access :**

Direct Memory Access (DMA) transfers the block of data between the memory and peripheral devices of the system, without the participation of the processor. The unit that controls the activity of accessing memory directly is called a DMA controller.

The processor relinquishes the system bus for a few clock cycles. So, the DMA controller can accomplish the task of data transfer via the system bus. In this section,

we will study in brief about DMA, DMA controller, registers, advantages and disadvantages.

**What is DMA and Why it is used?**

Direct memory access (DMA) is a **mode of data transfer** between the memory and I/O devices. This happens **without the involvement** of the processor. We have two other methods of data transfer, **programmed I/O** and **Interrupt driven I/O**. Let's revise each and get acknowledge with their drawbacks.

In **programmed I/O**, the processor keeps on scanning whether any device is ready for data transfer. If an I/O device is ready, the processor **fully dedicates** itself in transferring the data between I/O and memory. It transfers data at a **high rate, but it can't get involved in any other activity** during data transfer. This is the major **drawback** of programmed I/O.

In **Interrupt driven I/O**, whenever the device is ready for data transfer, then it raises an **interrupt to processor**. Processor completes executing its ongoing instruction and saves its current state. It then switches to data transfer which causes a **delay**. Here, the processor doesn't keep scanning for peripherals ready for data transfer. But, it is **fully involved** in the data transfer process. So, it is also not an effective way of data transfer.

The above two modes of data transfer are not useful for transferring a large block of data. But, the DMA controller completes this task at a faster rate and is also effective for transfer of large data block.

The DMA controller transfers the data in three modes:

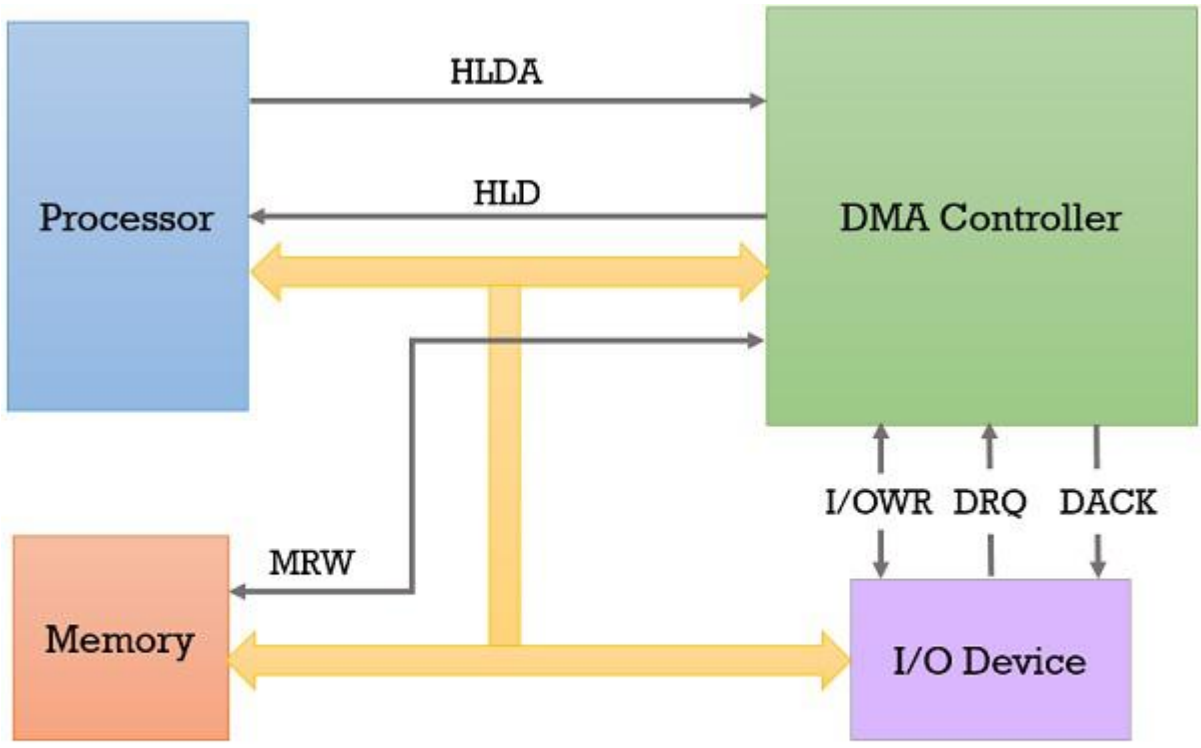
1. **Burst Mode:** Here, once the DMA controller gains the charge of the system bus, then it releases the system bus only after **completion** of data transfer. Till then the CPU has to wait for the system buses.
2. **Cycle Stealing Mode:** In this mode, the DMA controller **forces** the CPU to stop its operation and **relinquish the control over the bus** for a **short term** to DMA controller. After the **transfer of every byte**, the DMA controller **releases** the **bus** and then again requests for the system bus. In this way, the DMA controller steals the clock cycle for transferring every byte.
3. **Transparent Mode:** Here, the DMA controller takes the charge of system bus only if the **processor does not require the system bus**.

**Direct Memory Access Controller & it's Working**

DMA controller is a **hardware unit** that allows I/O devices to access memory directly without the participation of the processor. Here, we will discuss the working of the DMA controller. Below we have the diagram of DMA controller that explains its working:

1. Whenever an I/O device wants to transfer the data to or from memory, it sends the DMA request (**DRQ**) to the DMA controller. DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it the Hold request (**HLD**).

2. CPU receives the Hold request (HLD) from DMA controller and relinquishes the bus and sends the Hold acknowledgement (**HLDA**) to DMA controller.
3. After receiving the Hold acknowledgement (HLDA), DMA controller acknowledges I/O device **(DACK)** that the data transfer can be performed and DMA controller takes the charge of the system bus and transfers the data to or from memory.
4. When the data transfer is accomplished, the DMA raise an **interrupt** to let know the processor that the task of data transfer is finished and the processor can take control over the bus again and start processing where it has left.




DMA Controller Data Transfer

**Direct Memory Access Diagram**

After exploring the working of DMA controller, let us discuss the block diagram of the DMA controller. Below we have a block diagram of DMA controller.

Whenever a processor is requested to read or write a block of data, i.e. transfer a block of data, it instructs the DMA controller by sending the following information.

1. The first information is whether the data has to be read from memory or the data has to be written to the memory. It passes this information via **read or write control lines** that is between the processor and DMA controllers **control logic unit**.
2. The processor also provides the **starting address** of/ for the data block in the memory, from where the data block in memory has to be read or where the data block has to be written in memory. DMA controller stores this in its **address register**. It is also called the **starting address register**.
3. The processor also sends the **word count**, i.e. how many words are to be read or written. It stores this information in the **data count** or the **word count** register.
4. The most important is the **address of I/O device** that wants to read or write data. This information is stored in the **data register.**

**Direct Memory Access Advantages and Disadvantages**

**Advantages:**

1. Transferring the data without the involvement of the processor will **speed up** the read-write task.
2. DMA **reduces the clock cycle** requires to read or write a block of data.
3. Implementing DMA also **reduces the overhead** of the processor.

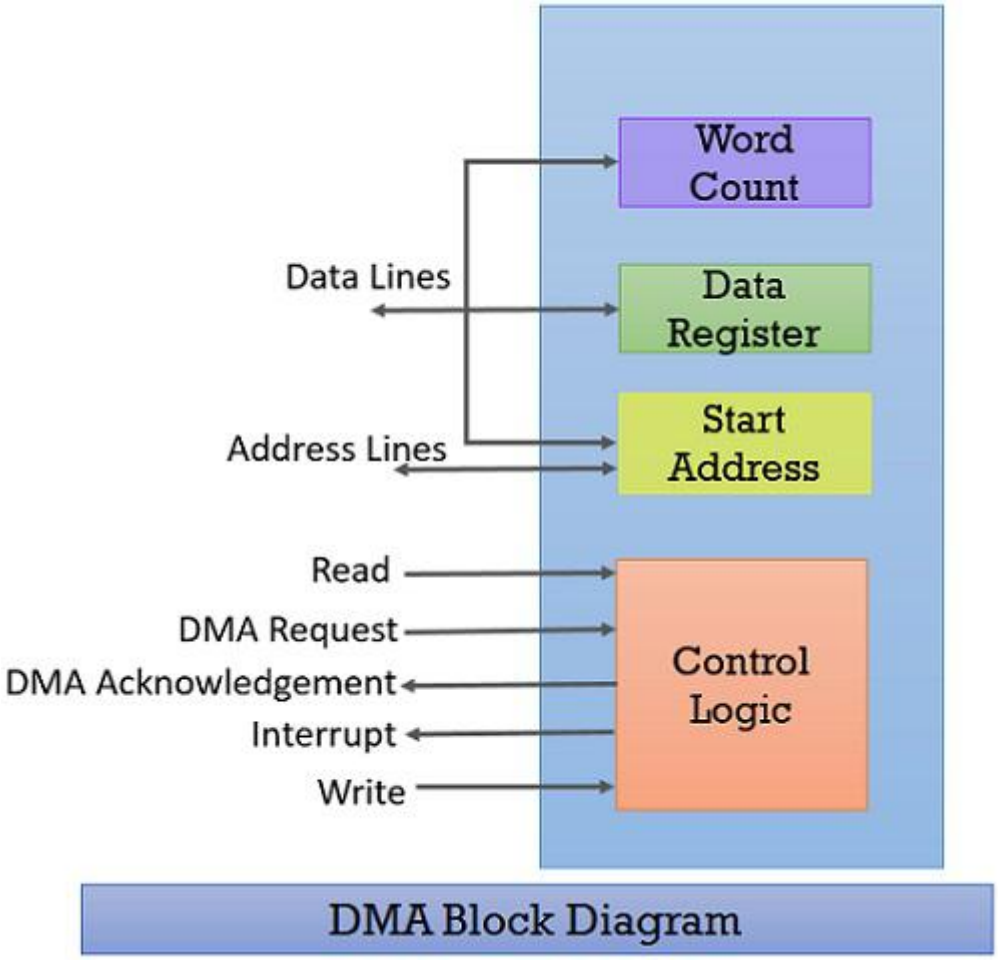
**Disadvantages**

1. As it is a hardware unit, it would **cost** to implement a DMA controller in the system.
2. Cache **coherence** problem can occur while using DMA controller.

**Key Takeaways**

- DMA is an abbreviation of **direct memory access**.
- DMA is a **method of data transfer** between main memory and peripheral devices.
- The hardware unit that controls the DMA transfer is a **DMA controller**.
- DMA controller transfers the data to and from memory **without the participation of the processor**.
- The processor provides the **start address** and the **word count** of the data block which is transferred to or from memory to the DMA controller and **frees the bus for DMA controller** to transfer the block of data.
- DMA controller transfers the data block at the **faster rate** as data is directly accessed by I/O devices and is not required to pass through the processor which save the clock cycles.
- DMA controller transfers the block of data to and from memory in three modes **burst mode**, **cycle steal mode** and **transparent mode**.
- DMA can be configured in various ways it can be a part of individual I/O devices, or all the peripherals attached to the system may share the same DMA controller.

Thus the DMA controller is a convenient mode of data transfer. It is preferred over the programmed I/O and Interrupt-driven I/O mode of data transfer.




**Buses :**

A bus is a subsystem that is used to connect computer components and transfer data between them. For example, an internal bus connects computer internals to the motherboard.

A bus may be parallel or serial. Parallel buses transmit data across multiple wires. Serial buses transmit data in bit-serial format.

A bus was originally an electrical parallel structure with conductors connected with identical or similar CPU pins, such as a 32-bit bus with 32 wires and 32 pins. The earliest buses, often termed electrical power buses or bus bars, were wire collections that connected peripheral devices and memory, with one bus designated for peripheral devices and another bus for memory. Each bus included separate instructions and distinct protocols and timing.

Parallel bus standards include advanced technology attachment (ATA) or small computer system interface (SCSI) for printer or hard drive devices. Serial bus standards include universal serial bus (USB), FireWire or serial ATA with a daisy-chain topology or hub design for devices, keyboards or modem devices.

Computer bus types are as follows:

- System Bus: A parallel bus that simultaneously transfers data in 8-, 16-, or 32-bit channels and is the primary pathway between the CPU and memory.
- Internal Bus: Connects a local device, like internal CPU memory.
- External Bus: Connects peripheral devices to the motherboard, such as scanners or disk drives.
- Expansion Bus: Allows expansion boards to access the CPU and RAM.
- Frontside Bus: Main computer bus that determines data transfer rate speed and is the primary data transfer path between the CPU, RAM and other motherboard devices.
- Backside Bus: Transfers secondary cache (L2 cache) data at faster speeds, allowing more efficient CPU operations.

**Interface circuits :**

**Interface circuits**: An I/O **interface** consists of the circuitry required to connect an I/O device to a **computer** bus. On the side of the **interface** we have the bus signals for address, data and control.

**Standard I/O Interfaces:**

**PCI BUS :**

The power and speed of computer components has increased at a steady rate since desktop computers were first developed decades ago. Software makers create new applications capable of utilizing the latest advances in processor speed and hard drive capacity, while hardware makers rush to improve components and design new technologies to keep up with the demands of high-end software.

There's one element, however, that often escapes notice - the **bus**. Essentially, a bus is a channel or path between the components in a computer. Having a high-speed bus is as important as having a good transmission in a car. If you have a 700-horsepower engine combined with a cheap transmission, you can't get all that power to the road. There are many different types of buses.

The idea of a bus is simple -- it lets you connect components to the computer's processor. Some of the components that you might want to connect include hard disks, memory, sound systems, video systems and so on. For example, to see what your computer is doing, you normally use a CRT or LCD screen. You need special hardware to drive the screen, so the screen is driven by a graphics card. A graphics card is a small printed circuit board designed to plug into the bus. The graphics card talks to the processor using the computer's bus as a communication path.

The advantage of a bus is that it makes parts more interchangeable. If you want to get a better graphics card, you simply unplug the old card from the bus and plug in a

new one. If you want two monitors on your computer, you plug two graphics cards into the bus. And so on.

In this article, you will learn about some of those buses. We will concentrate on the bus known as the Peripheral Component Interconnect (PCI). We'll talk about what PCI is, how it operates and how it is used, and we'll look into the future of bus technology.

**SCSI Bus :**

A small computer systems interface (SCSI) is a standard interface for connecting peripheral devices to a PC. Depending on the standard, generally it can connect up to 16 peripheral devices using a single bus including one host adapter. SCSI is used to increase performance, deliver faster data transfer transmission and provide larger expansion for devices such as CD-ROM drives, scanners, DVD drives and CD writers. SCSI is also frequently used with RAID, servers, high-performance PCs and storage area networks SCSI has a controller in charge of transferring data between the devices and the SCSI bus. It is either embedded on the motherboard or a host adapter is inserted into an expansion slot on the motherboard. The controller also contains SCSI basic input/output system, which is a small chip providing the required software to access and control devices. Each device on a parallel SCSI bus must be assigned a number between 0 and 7 on a narrow bus or 0 and 15 on a wider bus. This number is called an SCSI ID. Newer serial SCSI IDs such as serialattached SCSI (SAS) use an automatic process assigning a 7-bit number with the use of serial storage architecture initiators.

**USB Bus:**

A Universal Serial Bus (USB) is a common interface that enables communication between devices and a host controller such as a personal computer (PC). It connects peripheral devices such as digital cameras, mice, keyboards, printers, scanners, media devices, external hard drives and flash drives. Because of its wide variety of uses, including support for electrical power, the USB has replaced a wide range of interfaces like the parallel and serial port.

A USB is intended to enhance plug-and-play and allow hot swapping. Plug-and-play enables the operating system (OS) to spontaneously configure and discover a new peripheral device without having to restart the computer. As well, hot swapping allows removal and replacement of a new peripheral without having to reboot.

Although there are several types of USB connectors, the majority of USB cables are one of two types, type A and type B. The USB 2.0 standard is type A; it has a flat rectangle interface that inserts into a hub or USB host which transmits data and supplies power. A keyboard or mouse are common examples of a type A USB connector. A type B USB connector is square with slanted exterior corners. It is connected to an upstream port that uses a removable cable such as a printer. The type B connector also transmits data and supplies power. Some type B connectors do not have a data connection and are used only as a power connection.

The USB was co-invented and established by Ajay Bhatt, a computer architect who had been working for Intel. In 1994 seven companies that included Intel, Compaq, Microsoft, IBM, Digital Equipment Corporation (DEC), Nortel and NEC Corporation started the development of the USB. Their objective was to make it easier to connect peripheral devices to a PC and eliminate the mass amount of connectors. Factors involved included: creating larger bandwidths, streamlining software configurations and solving utilization problems for current interfaces.

**I/O Devices and Processors :**

The **DMA mode** of data transfer reduces CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices whose speeds are much slower as compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rises to the development of special purpose processor called **Input-Output Processor (IOP) or IO channel**.

The Input Output Processor (IOP) is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O – related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The main memory unit takes the pivotal role. It communicates with processor by the means of DMA.

The Input Output Processor is a specialized processor which loads and stores data into memory along with the execution of I/O instructions. It acts as an interface between system and devices. It involves a sequence of events to executing I/O operations and then store the results into the memory.

**Advantages –**

- The I/O devices can directly access the main memory without the intervention by the processor in I/O processor based systems.
- It is used to address the problems that are arises in Direct memory access method.

# Unit -5
# Parallel Processing

## Concept of parallel processing:

Parallel processing is a method of simultaneously breaking up and running program tasks on multiple microprocessors, thereby reducing processing time. Parallel processing may be accomplished via a computer with two or more processors or via a computer network. Parallel processing is also called parallel computing

## Pipeling:

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing. Pipelining is a technique where multiple instructions are overlapped during execution

**Types of Pipelining**

In 1977 Handler and Ramamoorthy classified pipeline processors depending on their functionality.

### 1. Arithmetic Pipelining

It is designed to perform high-speed floating-point addition, multiplication and division. Here, the multiple arithmetic logic units are built in the system to perform the parallel arithmetic computation in various data format. Examples of the arithmetic pipelined processor are Star-100, TI-ASC, Cray-1, Cyber-205.

### 2. Instruction Pipelining

Here, the number of instruction are pipelined and the execution of current instruction is overlapped by the execution of the subsequent instruction. It is also called **instruction lookahead**.

### 3. Processor Pipelining

Here, the processors are pipelined to process the **same data stream**. The data stream is processed by the first processor and the result is stored in the memory block. The result in the memory block is accessed by the second processor. The second processor reprocesses the result obtained by the first processor and the passes the refined result to the third processor and so on.

### 4. Unifunction Vs. Multifunction Pipelining

The pipeline performing the precise function every time is unifunctional pipeline. On the other hand, the pipeline performing multiple functions at a different time or multiple functions at the same time is multifunction pipeline.

### 5. Static vs Dynamic Pipelining

The static pipeline performs a fixed-function each time. The static pipeline is unifunctional. The static pipeline executes the same type of instructions continuously. Frequent change in the type of instruction may vary the performance of the pipelining.

Dynamic pipeline performs several functions simultaneously. It is a multifunction pipelining.

**6. Scalar vs Vector Pipelining**

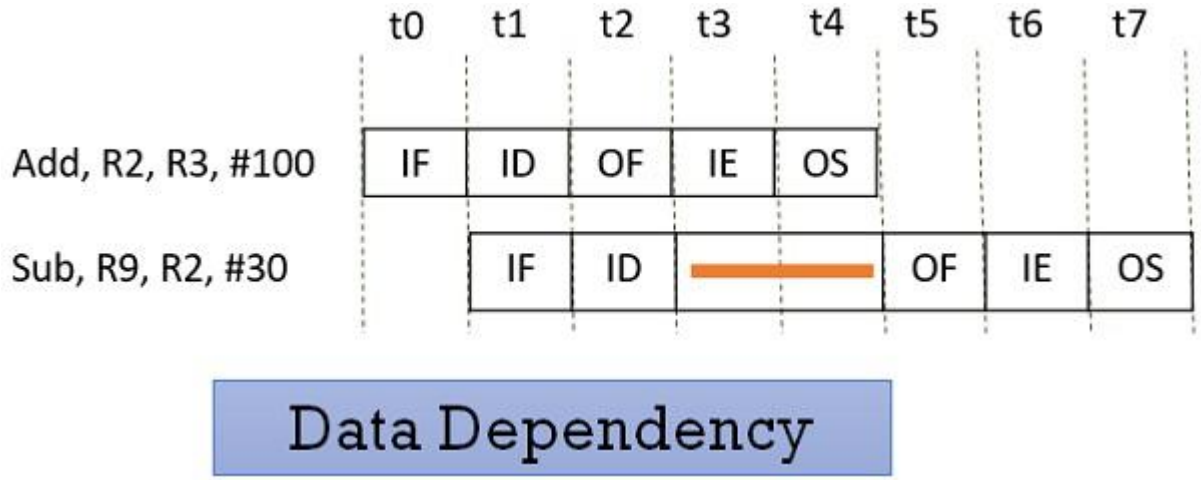
Scalar pipelining processes the instructions with scalar operands. The vector pipeline processes the instruction with vector operands.

**Pipelining Hazards**

Whenever a pipeline has to stall due to some reason it is called pipeline hazards. Below we have discussed four pipelining hazards.

**1. Data Dependency**

Consider the following two instructions and their pipeline execution:




In the figure above, you can see that result of the **Add** instruction is stored in the register **R2** and we know that the final result is stored at the end of the execution of the instruction which will happen at the clock cycle **t4**.

But the **Sub** instruction need the value of the register **R2** at the cycle **t3**. So the Sub instruction has to **stall** two clock cycles. If it doesn't stall it will generate an incorrect result. Thus depending of one instruction on other instruction for data is **data dependency**.

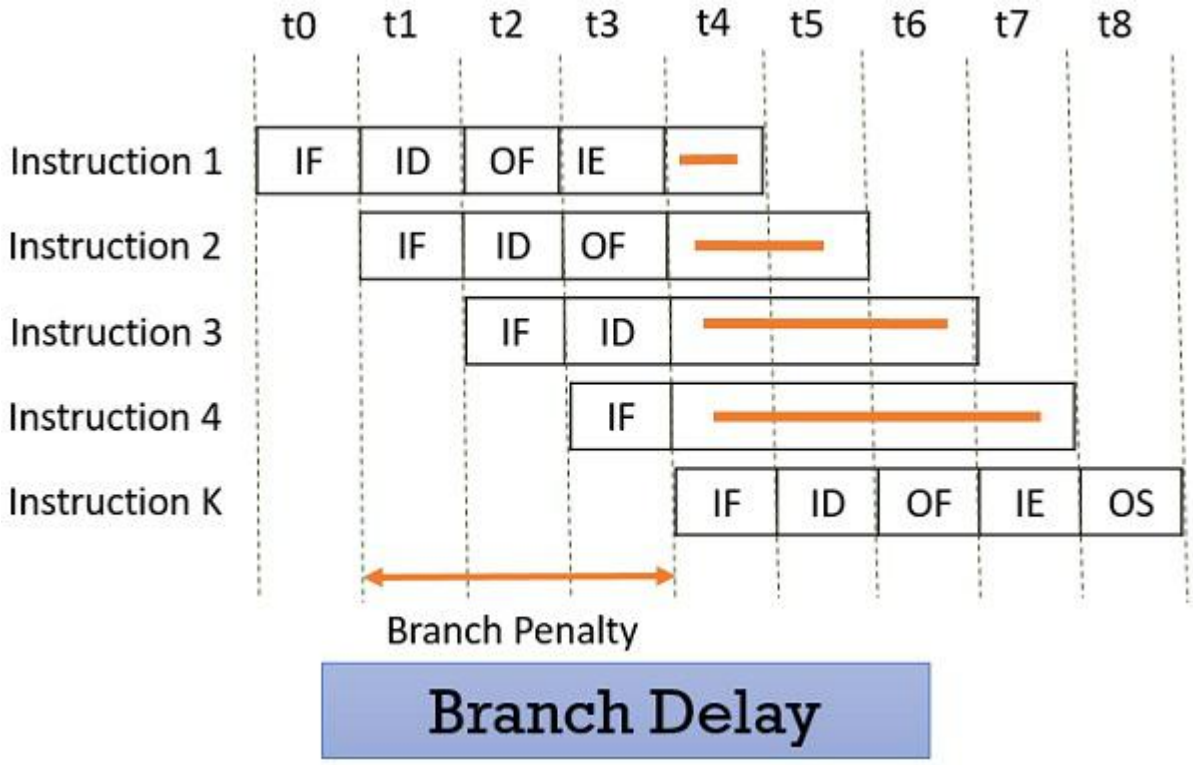
**2. Memory Delay**

When an instruction or data is required, it is first searched in the cache memory if not found then it is a **cache miss**. The data is further searched in the memory which may take ten or more cycles. So, for that number of cycle the pipeline has to stall and this is a **memory delay** hazard. The cache miss, also results in the delay of all the subsequent instructions.

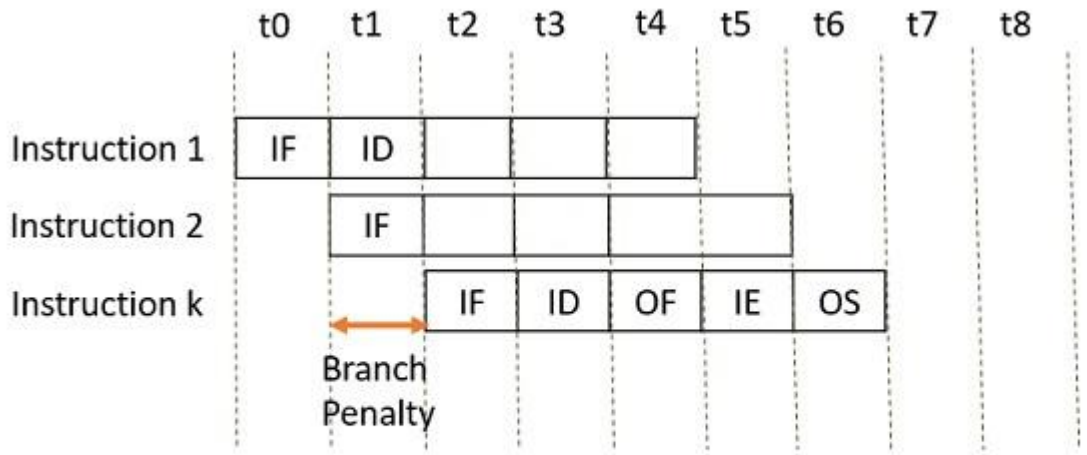
**3. Branch Delay**

Suppose the four instructions are pipelined I1, I2, I3, I4 in a sequence. The instruction I1 is a branch instruction and its target instruction is Ik. Now, processing starts and instruction I1 is fetched, decoded and the target address is computed at the 4th stage in cycle t3.

But till then the instructions I2, I3, I4 are fetched in cycle 1, 2 & 3 before the target branch address is computed. As I1 is found to be a branch instruction, the instructions I2, I3, I4 has to be discarded because the instruction Ik has to be processed next to I1. So, this delay of three cycles 1, 2, 3 is a **branch delay**.




Prefetching the target branch address will reduce the branch delay. Like if the target branch is identified at the decode stage then the branch delay will reduce to 1 clock cycle.




**Branch Delay when Target Branch is Determined at Decode stage**

**4. Resource Limitation**

If the two instructions request for accessing the same resource in the same clock cycle, then one of the instruction has to stall and let the other instruction to use the resource. This stalling is due to **resource limitation**. However, it can be prevented by adding more hardware.

**Advantages**

1. Pipelining improves the throughput of the system.
2. In every clock cycle, a new instruction finishes its execution.
3. Allow multiple instructions to be executed concurrently.

**Key Takeaways**

- Pipelining divides the instruction in 5 stages instruction fetch, instruction decode, operand fetch, instruction execution and operand store.
- The pipeline allows the execution of multiple instructions concurrently with the limitation that no two instructions would be executed at the **same stage** in the **same clock cycle**.
- All the stages must process at equal speed else the slowest stage would become the bottleneck.
- Whenever a pipeline has to stall for any reason it is a pipeline hazard.

This is all about pipelining. So, basically the pipelining is used to improve the performance of the system by improving its efficiency.

**Forms Of Parallel Processing :**

There are multiple types of parallel processing, two of the most commonly used types include SIMD and MIMD. SIMD, or single instruction multiple data, is a form of parallel processing in which a computer will have two or more processors follow the same instruction set while each processor handles different data.

**Interconnection networks :**

Interconnection networks, also called multi-stage interconnection networks (or MINs), are high-speed computer networks. They're connections between nodes where each node can be a single processor or a group of processors or memory modules. ... The pattern in which the nodes are connected to each other is known as topology.

**Data Hazards :**

Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing instructions on the unpipelined machine. All the instructions after the ADD use the result of the ADD instruction (in R1).

Data hazards occur when instructions that exhibit data dependence modify data in different stages of a pipeline. Ignoring potential data hazards can result in race conditions (also termed race hazards). There are three situations in which a data hazard can occur:

1. read after write (RAW), a *true dependency*
2. write after read (WAR), an *anti-dependency*
3. write after write (WAW), an *output dependency*

Consider two instructions i1 and i2, with i1 occurring before i2 in program order.

**Read after write**

(i2 tries to read a source before i1 writes to it) A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved. This can occur because even though an instruction is executed after a prior instruction, the prior instruction has been processed only partly through the pipeline.

**Example**

For example:

```
i1. R2 <- R5 + R3
i2. R4 <- R2 + R3
```

**Write after read (WAR)**

(i2 tries to write a destination before it is read by i1) A write after read (WAR) data hazard represents a problem with concurrent execution.

**Example**

For example:

```
i1. R4 <- R1 + R5
i2. R5 <- R1 + R2
```

In any situation with a chance that i2 may finish before i1 (i.e., with concurrent execution), it must be ensured that the result of register R5 is not stored before i1 has had a chance to fetch the operands.

**Write after write**

(i2 tries to write an operand before it is written by i1) A write after write (WAW) data hazard may occur in a concurrent execution environment.

**Example**

For example:

```
i1. R2 <- R4 + R7
i2. R2 <- R1 + R3
```

The write back (WB) of i2 must be delayed until i1 finishes executing.

**Structural hazards:**

A structural hazard occurs when two (or more) instructions that are already in pipeline need the same resource. The result is that instruction must be executed in series rather than parallel for a portion of pipeline. Structural hazards are sometime referred to as resource hazards.

Example: A situation in which multiple instructions are ready to enter the execute instruction phase and there is a single ALU (Arithmetic Logic Unit). One solution to such resource hazard is to increase available resources, such as having multiple ports into main memory and multiple ALU (Arithmetic Logic Unit) units.

**Control hazards (branch hazards or instruction hazards)**

Control hazard occurs when the pipeline makes wrong decisions on branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded. The term branch hazard also refers to a control hazard.

**Influence on instruction sets :**

**Instruction pipelining** is a technique for implementing instruction-level parallelism within a single processor. Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps (the eponymous "pipeline") performed by different processor units with different parts of instructions processed in parallel.

**Design Considerations :**

**Speed**

Pipelining keeps all portions of the processor occupied and increases the amount of useful work the processor can do in a given time. Pipelining typically reduces the processor's cycle time and increases the throughput of instructions. The speed advantage is diminished to the extent that execution encounters hazards that

require execution to slow below its ideal rate. A non-pipelined processor executes only a single instruction at a time. The start of the next instruction is delayed not based on hazards but unconditionally.

A pipelined processor's need to organize all its work into modular steps may require the duplication of registers, which increases the latency of some instructions.

**Economy**

By making each dependent step simpler, pipelining can enable complex operations more economically than adding complex circuitry, such as for numerical calculations. However, a processor that declines to pursue increased speed with pipelining may be simpler and cheaper to manufacture.

**Predictability**

Compared to environments where the programmer needs to avoid or work around hazards, use of a non-pipelined processor may make it easier to program and to train programmers. The non-pipelined processor also makes it easier to predict the exact timing of a given sequence of instructions.

**Data path and control Considerations:**

A **data path** (also written as **datapath**) is a set of functional units that carry out **data** processing operations. Datapaths, along with a **control** unit, make up the CPU (central processing unit) of a **computer** system. A larger **data path** can also be created by joining more than one together using multiplexers

**Performance Considerations:**

Imbalance among pipeline stages. Imbalance among the pipe stages reduces performance since the clock can run no faster than the time needed for the slowest pipeline stage; Pipeline overhead. Pipeline overhead arises from the combination of pipeline register delay (setup time plus propagation delay) and clock skew.

**Exception Handling:**

Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often disrupting the normal flow of program execution.