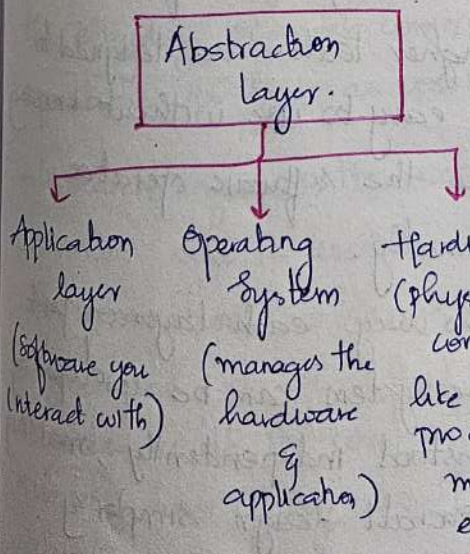
MODULE 03

- Computer Abstraction and technology refers to the way computers are designed and used by concealing complex details to make them easier to understand and work with.

Abstraction: It is simplifying things by focusing on what's important and ignoring unnecessary details.

Technology: This refers to the actual hardware and software that make computers work.



Abstraction simplifies the complexity of computer technology making it accessible for users and developers.

THE EIGHT DESIGN FEATURE

1. Design for Moore's Law.
2. Use Abstraction to simplify design.
3. Make the common case fast.
4. Performance via parallelism.
5. Performance via pipelining.
6. Performance via prediction.
7. Dependability via redundancy
8. Hierarchy of memories.

1. Design for Moore's Law.

moore's law states that the no of transistors on a computer chip doubles approximately every two years.

while the cost per transistor decreases. This means computer becomes faster, smaller & cheaper over time.

⇒ Transistors are tiny switches on a chip that process info.

- moore's law explains why technology improves rapidly
- modern devices like smartphones laptops & AI rely on this progress.
- moore's law is slowing down as we reach physical limitations of how small transistors can get. New tech (like quantum computing) may continue the trend

2) Use Abstraction to simplify design

- Use abstraction to simplify design means hiding the complex details of a system and focusing on only what is necessary at each level.
- This makes it easier to work on different parts of the system without overwhelming by unnecessary information.

Ex: In software design

- low level details like how the hardware works, are hidden behind simpler interfaces.
- Higher levels are designed to be easy to use without knowing how the software operates internally.
- This way each layer or part of the system can be developed & understood independently, making the overall design simpler & more efficient.

3) Make the common case fast.

focus on making the tasks that happen most often run as fast as possible. This improves efficiency and the overall experience

- It is a principle in design that focuses on optimizing the most frequent or typical tasks to make them quicker & more efficient.
- Instead of spending too much time improving rare or complex scenarios, the goal is to make the everyday or most common operations as fast as possible.
- In search engines, the most common case is users searching for information, so search engines are designed to give fast & accurate result for common queries, even if they don't always optimize for rare, complex searches.

4) Performance via parallelism.

(Ex: cake)

- Improving the speed of a system by breaking a task into smaller parts and executing those parts simultaneously rather than one after the other.
- Instead of doing one thing at a time, parallelism allows multiple tasks to be done at the same time.
- This speeds up the overall process and make systems perform better, especially for large or complex tasks.
- In computing it can be done by using multiple processors or core to handle different parts of a task simultaneously.
- Improves performance
- Efficient use of resources

Parallelism helps improve performance by breaking tasks into small parts and completing them all at once, making anything faster.

5) Performance via pipelining

- It is a technique used to improve the speed of a process by breaking it down into smaller stages where each stage works on different parts of the task at the same time.
- While one stage works on one part, other stage can work on different parts at the same time so the process runs faster.
- In a CPU instructions can be divided into multiple stages like fetching the instruction, decoding it, executing it & etc....
- While CPU is ~~fetching~~ executing one instruction, it can start fetching the next one, so it never stays idle
- faster processing
- Efficient use of time.

Pipelining improves performance by breaking a process into smaller steps and having each step run at the same time, so the task is completed faster.

6) Performance via Prediction

- It is a technique used to make systems faster by predicting what will happen next, and preparing it for advance.
- Instead of waiting for the next step the system guesses the next step and so starts working on it early.
- By preparing ahead of time, the system can avoid delays and complete tasks faster.

7) Branch prediction in CPU

- The cpu predicts which instruction will be executed next in a program, so it can start executing it before the decision is fully made.
- This reduces the waiting time that happens when the CPU is deciding what to do next.

8) Caching: Computer stores data they predict will be needed soon.

Eg: If you often visit a website your browser predicts you might visit it again soon and store some parts of that site to load faster.

* faster execution
* efficient resource use.

Prediction improves performance by guessing what will happen next & preparing for it early. So the system can respond faster & avoid delays.

9) Dependability via Redundancy.

- It is a technique used to make systems more reliable by having extra components or copies that can take over if something fails.
- Redundancy means adding extra parts to a system so if one part breaks or stops working the other part can step in & keep things running.
- This makes the system more dependable because it won't fail easily.

* Increased reliability
* Data protection (ensure data isn't lost if something breaks).

Redundancy makes system more dependable by having backup components or copies so they can continue to work even if something goes wrong.

8) Hierarchy of memories

memory in a computer is arranged in levels, from the fastest and smallest to the slowest and largest. The idea is to keep the most frequently used data in the fastest, smallest memory and move the less frequently used data to slower, larger, memories.

Slowest & largest memory, often used for backups or archives. (External drives, cloud storage, magnetic tapes).

LAYERS OF ABSTRACTION:

1) Hardware layer:

- This is the physical part of the computer such as CPU, memory and storage devices.
- It involves electrical circuits and low level operations.
- Abstraction

you no need to know how the electrical signals work to use a computer. as high layers abstract those details.

2) Operating System (OS)

- The OS sits above the Hardware layer and manages resources like memory, processes and I/O devices
- It abstracts away the complexity of interacting directly with hardware.

Abstraction:

Users and applications interact with the OS through easy to use interfaces without needing to understand the hardware.

3) Software Application

- These are the programs you use (web browser, etc..)
- They interact OS to perform tasks like opening files or displaying data.

Abstraction:

User's dont need to worry about how data is processed or stored, they just use the features of the application.

4) Programming Language.

- High level languages like Python, java etc.. are used to write software.
- They abstract away the machine code that runs directly on the hardware.

- Abstraction.

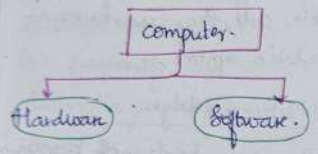
Programmers do not need to know the binary instructions or memory address to write code, they use human-readable syntax.

5) Networking Layer.

- This layer deals with transmission of data b/w systems over networks.
- The complexity of sending data across the internet is hidden behind simple operations like sending an email or browsing a website

COMPONENTS OF A COMPUTER

- A computer is an electronic device that processes information by following instructions and provides the result to the user

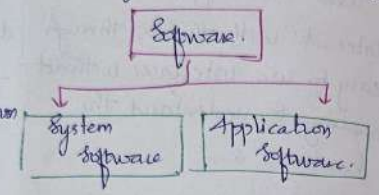


Hardware

Hardware refers to the physical parts of the computer that you can see and touch. These components are responsible for performing tasks based on instructions from the software
Eg: Monitor, CPU, keyboard, mouse.

Software.

Software consists of the instructions of programs that tell the hardware what to do. It acts as a bridge b/w user and hardware



System software :

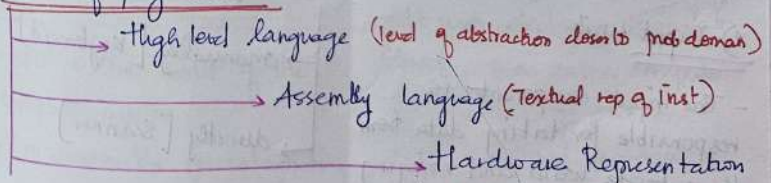
This software manages the computer's hardware and basic functions. It helps other software run on the computer.
Eg: OS.

Application Software.

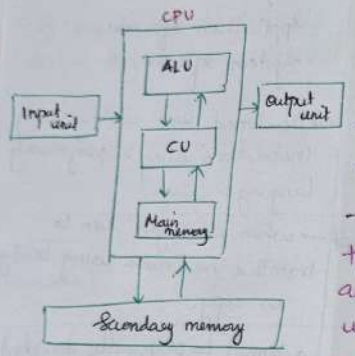
It is what allows users to do tasks like writing documents, watching videos or analyzing data on their computers. It is designed to help with specific activities, making it the part of the software that directly impacts users work or enjoyment
Eg: word processes, Browsers etc.

Hardware	Software
→ further divided into 4 categories • Input • Secondary • Output • Internal components	→ further divided into two main categories - Application Software - System software.
→ Developed using electronic & other material	→ Developed ~~using~~ writing using instructions using a programing language
→ when damaged can be replaced with new component	→ when damaged can be installed once more using backup or copy
→ physical in nature	→ cannot be physically touched
→ Hardware cannot be infected by virus	→ The software can be infected by virus
Eg: Hard drives, monitors, CPU, Scanners, printers, etc.	→ Windows 10, Adobe photoshope etc.

Levels of prog code



FIVE KEY COMPONENTS OF A COMPUTER



- The five classic components of a computer are, input, output, memory, datapath and control.
- In some cases the datapath and control components are combined and referred to as processor.

1) Input unit

- The input component is responsible for taking data from the outside world and bringing it into the computer.
  Ex: keyboard, mouse, scanner etc...
- Input devices allows the user to interact with the computer by entering data, such as typing, clicking or touching the screen.
- They are electromagnetic devices that accept data from the user and translate into machine understandable code which is understood by the computer.
- It is through the input unit that the user communicates with the computer. In this way it serves as a link b/w user and computer.
- A computer accepts data in two way
  - → manually [keyboard]
  - → directly [scanner]

2) Output Unit

- The output component displays or sends the data processed by the computer to the outside world
  ex: monitor, printer, speaker etc..
- output devices show or produce the results of the computer's processing such as showing a website, printing a document or playing a sound.
- The output generated by the CPU is in coded form and cannot be understood by the user. It translates the coded output from machine language to human understandable language.

3) Central processing unit (CPU)

- CPU is the heart of the computer system.
- It is responsible for interpreting and executing program instructions performing essential operations such as arithmetic, logic I/O, control tasks.

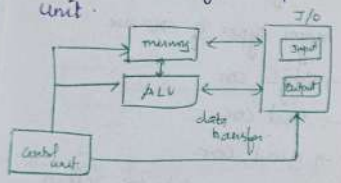
The common components of CPU are
- → ALU
- → CU
- → Registers.

major types of CPU are
- → single core
- → dual core
- → quad core.
- → Hexa core, Octa core & Deca core.

Control Unit

- It controls and coordinates all the tasks that allow the computer to function smoothly
- It uses electronic signals to direct the entire computer system to carry out or execute stored programs & instructions.

- It first instructs the input unit to transfer raw data and instruction to memory unit
- Sends to ALU for processing
- processed data is send back to memory unit for temporary storage
- Displays the result of processing to the user through output unit.

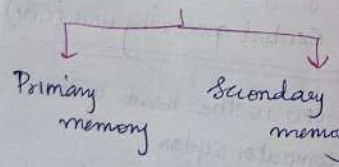


Arithmetic Logic Unit

- The arithmetic and logical unit is where actual processing on data takes place.
- it performs arithmetic and logical operations on the data.
- it also controls the speed of these operations.
- four basic mathematical operation.
(+, −, ×, & ÷)
Addition, sub, mul, div
- Three comparative or logical operation ie (>, <, =) to compare letters, numbers or special chars b/w data items.
- Logical Boolean operations such as 'AND', 'OR', 'NOT'
- These operations are carried out at high speed.

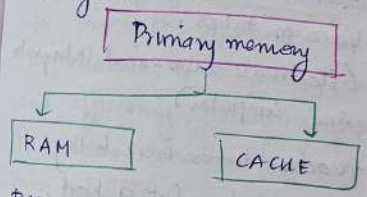
Memory Unit.

- The memory is where the programs are kept when they are running. It also contains the data needed by the running programs.



Primary memory

- it is a crucial part of a comp, that stores data and instruction that are actively being processed by the CPU.
→ it allows the CPU to quickly access the data it needs to perform tasks, making it faster than other types of memory such as secondary storage.



RAM:

This the main type of primary memory.
- volatile (loses all stored data when the power is turned off)
- Ram is used to store data that the CPU needs to access quickly.

Cache memory

- Smaller & faster type of memory.
- It stores frequently accessed data to speed up processing
- faster than RAM but smaller in size.

Imp bcoz
* Speed (faster, overall perfor ↑)
* Temporary storage. (temp hold data. once task comp, data is either saved (discarded).

Secondary Storage Unit

- Secondary memory also known as auxillary memory.
- It is a non-volatile memory
- Used to store data permanently unless erased by user or till the damage of the storage device.
- They are used to maintain a backup of important files and data.

- This memory is less expensive and has much larger storage capacity than primary memory.

Eg: Hard disk, pen drives, CD etc...

- Access speed is slower compared to primary memory.

- Secondary unit is used to store the data which is not needed immediately by the CPU.

- can store large capacity but less speed.

Technologies for Building Processors & memory

- The evolution of processors & memory technologies has been guided by innovations that have progressively improved computing performance, reduce, costs and enabled more complex application.

1. Vaccum Tubes (1st Gen)

- Early computers relied on vaccum tubes to perform logic operations. These tubes controlled the flow of electrons and acted as on/off switches to carry out digital computations.

ENIAC - The first comp to use vaccum tubes.
(Electronic numerical integrator and computer).

- Vaccum tubes were bulky, generated a lot of heat, consumed much power, & had a limited lifespan. They were quickly replaced by trans. for better performance.

2) Transistors (2nd Gen)

- foundation of modern electronic devices.
- They are made from semiconductor materials (usually silicon) and have atleast 3 terminals for connection of external circuit.
- Transistors are smaller, more reliable, require less power and generate less heat than vaccum tubes.
- They can switch circuits on & off enabling faster processing
- The shift from vaccum tubes to transistors marked a sig. improvement in computing power and efficiency, allowing comp to become smaller, faster & more reliable.

3) Integrated circuits (3rd Gen)

- Integrated chips replaced individual transistors. It combined multiple components such as transistors, resistors and capacitors & other circuits into a single chip.
- Integrated chips made computer much smaller & more reliable & energy efficient.

4) Very Large Scale Integration (VLSI) (4th Gen)

- refers to the integration of thousands of transistors & other components into a single chip.
- This technology made it possible to create microcomputer.
- Around 5000 transistors & circuit elements could be integrated into a single chip
- micro computers became more powerful, compact, reliable and affordable.
- The fourth gen → personal computer, making computer accessible to a broader audience and enabling a wide range of applications.

5) Ultra Large scale Integ (ULSI) (Vth gen)

- takes the principle of VLSI further by integrating millions of transistor onto a single chip.
- This technology is crucial for the development of advanced micro computers.
- The fifth generation of computer is characterised by parallel processing and artificial intelligence.
- ULSI lead to the creation of high performance processors capable of handling vast amounts of data & performing sophisticated operations quickly

PERFORMANCE

- Time is the measure of computer performance.
- The computer that performs the same amount of work in the least time is the fastest.
- Program execution time is measured in seconds per program.
- Throughput
  no of tasks ~~performed~~ completed per unit time
- Response time
  Total time required for the computer to complete a task.
- To maximise performance we want to minimise execution time. Thus we can relate performance and execution time for computer X

$$\text{performance}_X = \frac{1}{\text{Execution}_X}$$

- for two computers X & Y
  If performance of X is > performance of Y.
  Then execution time on Y is longer than on X.

$$\text{Performance}_x > \text{Performance}_y$$
$$\Downarrow$$
$$\text{Execution Time}_y > \text{ExecutionTime}_x$$

$$\frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution}_y}{\text{ExecutionTime}_x} = n$$

a) Computer X runs a program in 10 seconds and Y runs the same program in 15 seconds How much faster is X than Y

We know X is n times faster than Y if

$$\frac{\text{Per}_X}{\text{Per}_Y} = \frac{\text{Exe}_Y}{\text{Exe}_X} = n$$

$$\frac{\text{Exec}_Y}{\text{Exec}_X} = \frac{15}{10} = 1.5$$

∴ X is 1.5 times as fast as Y.
or Y is 1.5 times slower than X

clock period:
The length of each clock cycle. (clock frequency = 1 / clock period)

clock cycle:
is the total time period of a complete execution.

clock rate:
Inverse of clock period

CPU Execution time.
The actual time the CPU spends computing for specific task

$$\text{CPU Exec} = \text{CPU clock cycles} \times \text{clock cycle time}$$

$$\text{CPU Exec} = \frac{\text{CPU clock cycle}}{\text{clock rate}}$$

a) A program runs in 10 secs on comp A which has 2 GHz clock. A designer is trying to build a computer B to run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing comp B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target.

$\Longrightarrow$ Exec time$_A$ = 10s

Clk$_A$ = 2×10^9

Exec$_B$ = 6s

Clk = 1.2 x clk$_A$

(Refer the Pdf).

INSTRUCTIONS

- The words of a computer language is called Instructions.
- The vocabulary of commands understood by a given architecture is called Instruction set (Instruction set Architecture) →
- MIPS (microprocessor without Interlocked pipeline stages)

- Simplicity : MIPS uses a small, efficient set of instructions, making it easier for the processor to decode and execute them quickly.
- Efficiency : By using fewer instructions, it requires fewer cycles to perform tasks leading to higher efficiency.
- Scalability : The MIPS architecture is scalable which means it can be used in both small & large systems.
- has been used in various computing systems due to its simplicity, efficiency & scalability.

• ARM (Advanced RISC machine)

versions ARMv7, ARMv8.

- Power Efficiency :

ARM processor also known for being highly power efficient which makes them ideal for mobile devices like smartphones and tablets as well as Internet of things (IoT) devices.

- widespread use

ARM architecture is the most widely used processor architecture in mobile devices and low power - applications.

- Instruction Set :

The ARM processor have a simpler Instruction set. leading to faster execution & lower power consumption.

- ARM is found in most mobile devices, embedded systems

ASSEMBLY LANGUAGE (ASM)

- Assembly language is a type of low-level programming language that is intended to communicate directly with a computer's hardware.
- It requires knowledge of hardware details like registers, memory addressing and the Instruction set.

Registers : Small, fast storage locations within the CPU

Memory Addressing : How data is located in a memory

I/O operation : Handling how data input & output from devices (like, keyboard, displays etc)

Instruction Set

The set of operations supported by the CPU (add, subtract).

- An Assembly instruction typically consists of an operation / command / opcode followed by operands (the data or addresses that the operation will work with)

```
ADD  RA, RB, RC
     //RA = RB+RC
```

ADD : operation to add values.
RA, RB, RC : operands which are CPU registers.

* ADD R1, R2 R3

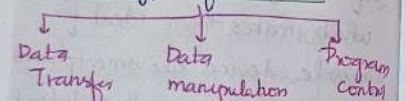
R1 → Destination ~~Add~~ reg.

R2, R3 → Source register containing the numbers to be Added.

Assembler

An assembler is a tool that translated human understandable code to object code that the computers CPU can execute. This translation is necessary because the CPU understands only binary machine instructions.

Types of Instruction



(instructions)

Data Transfer Instruction

Data transfer instructions are a type of assembly language instructions used to move data between different locations in a computer's memory, CPU registers or I/O devices.

1. MOV
2. LOAD
3. STORE
4. PUSH
5. POP
6. IN/OUT

I/O → peripheral devices.

Types of data Transfer Instruction.

1. Register to Register.

MOV R1, R2

(copies the value in R2 to R1)

2. Memory to Register

loads data from a specific memory location into CPU register

LOAD R1, 1000

(loads the value @ memory address 1000 → R1)

3. Register to memory

Stores the value from register to memory.

Store R1, 1000

(stores the value in R1 to memory address 1000)

4. Immediate - to - Register

moves a constant value (Immediate) directly to register

MOV, R1, #5

Stores the value 5 in Reg R1

5. Immediate to memory

moves a constant value directly into a specific memory location.

STORE #5, 1000

(Stores the value 5 into ~~1000~~ memory address 1000)

6. memory to memory

Transfers data directly from one memory to another

7. Input / output transfer.

Transfers data b/w I/O devices & CPU registers or memory

Data Manipulation Instructions.

Data manipulation instructions are essential in assembly language to process and transform data stored in registers or memory. These instructions can be broadly classified into Arithmetic, logical, shift and comparison/relational categories.

i) Arithmetic Instructions.

These instructions perform basic mathematical operations such as addition, subtraction, multiplication and division

Instruction : add

```
add $t0, $t1, $t2
```

$t0 = $t1 + $t2

$t1 contains 5, $t2 contains 3

Result : $t0 = 5+3 = 8

Instruction : sub

```
sub $t0, $t1, $t2
```

$t0 = $t1 - $t2

$t1 contains 7, $t2 contains 4

∴ $t0 = 7-4 = 3

(ii) Logical Instructions.

These performs bitwise operation such as AND, OR and XOR

Instruction : and

performs a logical AND operation b/w 2 registers.

```
and $t0, $t1, $t2
```

$t0 = $t1 & $t2

$t1 contains 1010 (binary)

$t2 contains 1100 (binary)

Result $t0 = 1000 (binary)

(iii) Shift Instructions.

These move bits to the left or right, filling with zeros or duplicating the most significant bit.

Instruction : sll

(Shift logical shift left)

→ Shifts all bits in a register to the left by a specified no of positions.

```
sll $t0, $t1, 2
```

$t0 = $t1 << 2

$t1 contains 0010 (binary)

After shifting left 2 positions

$t0 = 1000 (binary)

Instruction : srl

(Logical shift right)

→ Shifts all bits in a register to the right by a specified number of positions.

```
srl $t0, $t1, 2
```

$t0 = $t1 >> 2

$t1 contains 1000 (binary)

After shifting right 2 positions

$t0 = 0010 (binary)

Program Control Instructions..

Program control instructions enable the processes to change the sequence of instruction execution based on conditions. These instructions are used to implement loops, decision-making and function calls.

1) Unconditional branch instruction

→ These instructions cause the program to jump to a specified memory address unconditionally (no conditions are checked)

2) Conditional Branch Instruction

→ These instructions jump to a specified memory address only if a condition is true.

Eg: If values of $t1 and $t2 are equal jump to Label.

```
beq $t0,$t1, LABEL
```

(Branch if equal)

5) Procedure calls & Return Instructions.

These are used to jump to subroutines (functions) and return to the main program after the function completes.

→ Jal (jump & link)

→ Jr (jump Register).

4) Interrupt Instructions.

Interrupt Instructions handle hardware and software interrupts. These temporarily pause the normal execution flow to handle high-priority tasks.

Operations of the computer hardware.

~~Ex~~ MIPS :-

1. Data transfer

focuses on moving data without altering it (load, Store, MOV)

2. Arithmetic/logic

performs computations & logical manipulations. (ADD, sub)

3. Control operations :

manages program flow. (Branching, jumping)

4. Memory Access

handles interactions with RAM/ROM (Read, write)

5. I/O operations

deal with external devices

Input : Accept dat from ext device

Output : Send data to ext device.

6. Program Control operations

manage function calls procedure calls

7. Interrupt handling operations

→ Interrupts pause the current program to service a high-priority task.

- Software Interrupt :

Triggered by system call

- Hardware Interrupt

Triggered by hardware devices.

THREE UNDERLYING PRINCIPLES OF HARDWARE

- The principles of hardware design aim to balance performance, simplicity and efficiency. They guide the design of computer systems and architectures, ensuring that they meet perfo. requirements while maintaining cost & operational feasibility.

1) Simplicity favours Regularity.

- Simple, uniform designs are easier to implement, optimize & maintain.

(i) fixed no of operands.

- MIPS inst always use exactly 3 operands (2 source, 1 dest)

- This avoids complexity of supporting variable-length operands.

add a,b,c

a = b+c

(2) Breaking complex operation.

Statement like f = (g+h) - (i+j) is broken into multiple MIPS instructions.

add t0, g, h t0 = g+h

add t1, i, j t1 = i+j

sub f, t0, t1 f = t0 - t1

2) Smaller is faster

- Reducing the size of critical components such as registers, lead to a faster performance due to reduced signal travel distance and simpler circuitry

- If there are fewer components the CPU doesn't need to waste time managing &

and accessing them, this speeds up processing

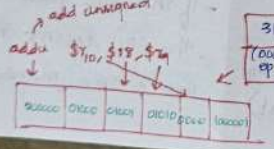
- processors like MIPS, registers are small so the CPU can work data quickly without the need for extra resources.

- The more compact the system is, the faster the CPU can access the data it needs, because the data doesn't have to travel long distances.

3) Good Design Demands Good compromises.

- The best design is where the most important goals are met while making the right compromise for the other factors.

(i) R-Type
(ii) I-type
(iii) j type



1. R-Type (Register - Register)

- R type registers are used for operations that involve only registers such as arithmetic & logical operations.

op	rs	rt	rd	Shamt	funct
6 bit	5bit	5bit	5bit	5bit	6bit

op - operation of the instruction called opcode

rs and rt - register source operands

rd - destination operand

Shamt - Shift amount (used in shift op)

funct : function field to determine exact operation

Ex: add $s0, $s1, $s2

Sum of $s1 and $2 are stored in $s0

31-26	25-21	20-16	15-11	10-6	5-0
(000000) op	rs	rt	rd	Shamt	funct

2. I-type (Immediate)

- used to perform operations that involve an immediate value along with one or two registers.

- In these instructions, one two register file locations are specified as well as a 16-bit immediate value which may be used as an operand or an address.

op	rs	rt	const/address
6bit	5bit	5bit	16bit

op - opcode

rs and rt - register operand destination specified by rt

- 16 bit address means a load word instruction, can load any word within a region of 2^{15} or 32,768 bytes of the address in the base register rs.

addi - add Immediate.

31-26	25-21	20-16	15-0
opcode	rs	rt	immed
6	5	5	Immediate (16)

↓
ALU (Immediate)

rs - operand A location in reg
rt - result dest location in registe
immed - operand B location in register file.

I-type Instruction format (Load or Store)

31-26	25-21	20-16	15-0
opcode	rs	rt	address

rs - base register address.
rt - source (store) a destination (load) location in reg file.
immed - offset address.

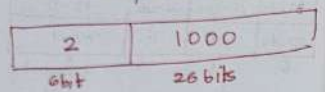
3) J-type (Jump)

used by jump instructions.

opcode	jump Instruction

↓
target address.

J 1000 represented as:



0 00
1 01
2 10
3 11

Instruction Set Architecture (ISA)

- It serves as a bridge b/w hardware and software and its design influences both hardware efficiency and software development.
- it should make hardware developers and software developers happy

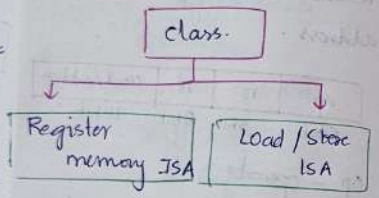
hardware (easy to implement efficiently)

Software (easy to generate good code).

There are seven dimensions of ISA

1) class of ISA
2) memory Addressing
3) Addressing modes
4) Types & Size of operands
5) operations
6) control flow instructions
7) Encoding.

1) class of ISA



Register memory ISA

- operations can be performed both on registers and memory.
- ie one operand for an operation can reside in memory and other can be in register.
- This allows the processor to work directly with both memory and register without requiring separate instructions for data movement b/w them.
- * direct operations b/w memory and registers are allowed.
- * it reduces the need for separate load / store instructions for each operation.
- * operations can involve data from memory and registers simultaneously.

Ex: IBM System/360
Intel x86

Add R1, 1000
(memory Address)
↑register.

Load - Store ISA

- memory is access is restricted to just load and store instructions.
- All other operations, including arithmetic and logic operations must take place b/w registers only
- * operations between registers only (no direct operations involving memory).
- * memory access is done only through load (load data from memory to register) and store (store data from registers to memory) instructions.

Ex: ARM (RISC)
MIPS (RISC)

```
LW R1, 0(R2)
# load word from memory address
  (R2 + 0) into reg R1
SW R1, 4(R2)
# Store word from Register
  R1 into memory address
  (R2 + 4)
ADD R3, R1, R2
  Perform arithmetic operation
  between Registers R1 and
  R2.
```

2) Memory Addressing (last pdf).

- memory addressing refers to the method by which a processor accesses memory locations.
- It describes how memory is organized and how the processor retrieves or stores data to/from these locations. Key components.

Direct memory Access (DMA) — memory Model

DMA

This allows the peripherals to directly access System memory without involving the CPU.
- This reduces the load on the processor and speeds up data transfer b/w device and memory.

memory model.

flat — Segmented

- flat memory model

all memory addresses are treated as a part of a continuous address space.

- Segmented memory model

memory is divided into segments (data, stack & heap et) each with its own address space.

Examples of memory model MIPS, x86

3) Addressing modes.

- Addressing modes specify how operands are located for operations.
- The operand could be in a register, memory or even part of the Instruction itself

There are different types of addressing modes & they are.

→ Immediate addressing mode

- The operand is a constant value directly embedded in the Instruction itself.

```
ADD R1, #5
```

operand 5, is the Immediate value to be added to the value in R1.

→ Register Addressing mode.

The operand is located in a register, The Instruction specifies which register to use.

```
ADD R1, R2
```

The value in register R2 is added to R1.

→ Direct Addressing mode.

The Instruction specifies the memory address directly where the operand is stored.

```
LOAD R1, 0X1000
```

The operand is located at the memory address 0X1000 and its value is loaded into register R1.

→ Indirect Addressing mode.

The Instruction specifies a reg that holds the address of the operand rather than the operand itself. @ or ()

```
LOAD R1, (R2)
```

Here the value of R2 is the memory address of the operand. The data at the address of R2 is loaded to R1.

→ Indexed Addressing mode

The operands address is computed by adding an offset to the value in a register.
→ useful while accessing array elements.

```
LOAD R1, (R2 +4)
```

The address of operand is determined by adding 4 to value in register R2.

6 → Base register Addressing Mode

Similar to Indexed addressing but here the base address is stored in a register and an offset is added to its base address.

LOAD R1, 8(R2)

The value R2 is treated as the base address. and 8 is added to it to get the final memory address of the operand.

→ Relative Addressing mode.

The operand's address is calculated by adding a constant value to the address of the next Instruction.

JMP 100

target address for jump is the address of next inst + (PC) + 100.

→ Stack Addressing mode

- The operand is located at the top of the stack.
- The stack pointer register (SP) is typically used to refer the current stack location

POP R1

4) Type & size of operands.

- This dimension specifies the types of data the processor can handle and their sizes
Eg: int, float point.
- Data types: Integer (4), float point, character (1 byte) string double - 8 byte
- word length: The size of word in processor. (Eg: 32-bit, 64 bit).
- Register size: The no of bits in a general purpose register (32 bits in MIPS)

5) Operations.

→ Data Transfer Inst.
→ Data manipulation Inst
→ Program sequencing & control inst.
→ In/output Instruction. (used for transfering info between the reg, memory and input/output devices)

6) Control flow Instructions.

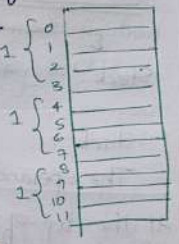
(same as program control Instructions)

7) Encoding in ISA.

Fixed length | Variable length

fixed length.

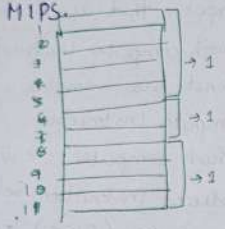
- Every computer Instruction is represented in same no of bytes.
- All ARM and MIPS inst are 4 bytes long. because each inst is of same length
- The bytes are required to increase to get the next Instruction is known.
- Computer that uses fixed length computer Instruction do not have complex computer Instruction.
- Such computer is called Reduced Instruction Set Computer (RISC) because the complexity of its Inst. set is reduced.



variable length

In variable length Encoding different computer Instructions are represented using different no of bytes.
- Variable length Instructions can take space less than fixed length Inst.

- So a program compiled for 8086 is usually smaller than the same program compiled for MIPS.



memory locations

(RISC vs CISC)
↳ watch.

TYPES OF ISA

Stack Accumulator GPR

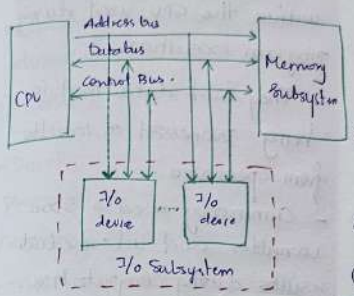
(1) Stack :
The operands are implicitly at the top of the stack

(2) Accumulator :
One operand is implicitly the accumulator.

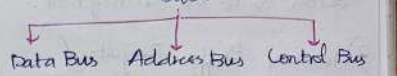
(3) General purpose register.
All operands and explicitly mentioned, they are either registers or

MODULE - 04

Basic Computer Organisation



- A Bus is a set of wires or pathways used to transfer, data addresses and control signals. between different parts of the system.
- A single bus is used to connect the CPU memory and other peripherals. However this bus can ~~transfer~~ ~~only~~ handle only one transfer at a time.



- Buses can be classified into
(I) Data Bus
(II) Address Bus
(III) Control Bus

(I) Data Bus.

Transfers Data between the CPU, memory and other peripherals

(II) Address Bus.

carries the memory address from the CPU to memory or I/O devices.

(III) Control Bus.

carries control signals that manage the operations of the CPU and other parts of the system.

Components of A CPU

(1) Memory Address register
holds the address of the memory to or from which

data is to be transferred.

2) Memory Buffer Register / Memory Data register.

- Holds the copy of the data that has been fetched from memory or is to be written to memory
- After memory address register (MAR) points the address, the MBR stores the data retrieved from that address or holds data to be written to memory.

3) Program counter (PC)

- Keeps track of the address of the next instruction to be fetched for execution.
- After the current instruction is fetched, the program counter is updated to point to the next instruction.

4) General purpose registers (R1, R2 etc)

- Temporary storage locations within the CPU used during program execution.
- They hold data that is being processed or results from operations.
- Commonly used to store variables and intermediate results during computation.

5) Instruction Register (IR)

- Holds the instruction that is currently being executed by the CPU.
- When an instruction is fetched from memory, it is placed in the IR for decoding & execution.

6) Arithmetic & logic unit (ALU)

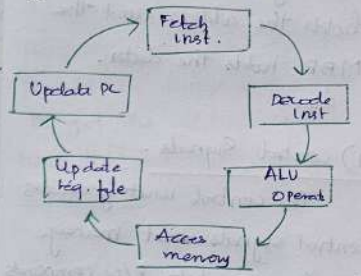
- Responsible for performing arithmetic and logical operations.
- ALU receives input from GPR or memory, performs the necessary computation and returns result.

7) Control Unit (CU)

- Directs operation of the processor by generating the necessary control signals.
- The CU coordinates data transfer, instructs the ALU to perform operations & manages memory read/write operations.
- It acts like the brain of the CPU ensuring that all the components work together in harmony.

The Executing An Instruction

The five fundamental phases of the Instruction cycle.



1. Fetching:

The Instruction is fetched from the memory and placed in the Instruction register (IR)

2. Decoding

The Instruction decoder (ID) decodes the Instruction in the IR to understand which operation is needed.

3. Executing the Instruction.

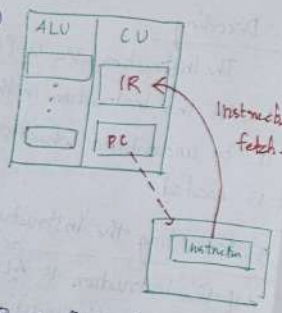
- If the Instruction is ALU operation or ALU performs the required operation using data from the GPR or memory.

- if the instruction involves memory access. the (MAR) holds the address and the MBR holds the data.

4) control signals.

The control unit generates control signals that manage the flow of data b/w components and control which operations are executed.

Execution.

① 

IR ← [[PC]]

PC ← [PC] + 4 (next counter point)

assuming (32 bit each containing 4 bit)

ie IR ← memory [PC]

The Instruction at memory location pointed to by PC is loaded into the IR.

- PC ← PC + 4

Program counter is incremented to point to the next instruction in memory.

② ⟹

for every execution of Instruction first two steps are always same

1) Send the program counter (PC) to the memory that contains the code and fetch the Instruction from the memory.

2) Read one or two registers. using field of Instruction to select the registers to read.

ie for a load word Inst. we need only one register but most other Instructions require two registers.

After these steps, the actions required to complete the Inst. depend on the Inst. class.

3) All Instruction class except jump (use ALU)

- memory reference Inst uses ALU [address calculation]
- arithmetic - logic Inst [operation execution]
- Branch Inst [comparison]

4) After using ALU (actions required to complete various Inst class differ.)

- memory - reference Inst need to access memory either to load or store data.
- Arithmetic - logical Inst write the data from the ALU back into a register.

- for branch Inst (change the next Inst address) based on the comparison otherwise the PC should be incremented by 4 to get the address of the next Instruction.

Examples

Add $s0, $s1, $s2

1. IR ← [[PC]]; PC ← [PC] + 4
2. Read s1 & s2
3. Perform Addition $s1 + $s2
4. Store $s0 ← $s1 + $s2

lw $ t0, 20 ($s0)

1. IR ← [[PC]]; PC ← [PC] + 4
2. Read $s0
3. Calculate the memory address, $s0 + 20
4. Access the memory & load the value to $t0
   t0 = memory [$s0 + 20]