Module 1

Functional Block of Computer

 # CPU, Memory, input output Subsystem,

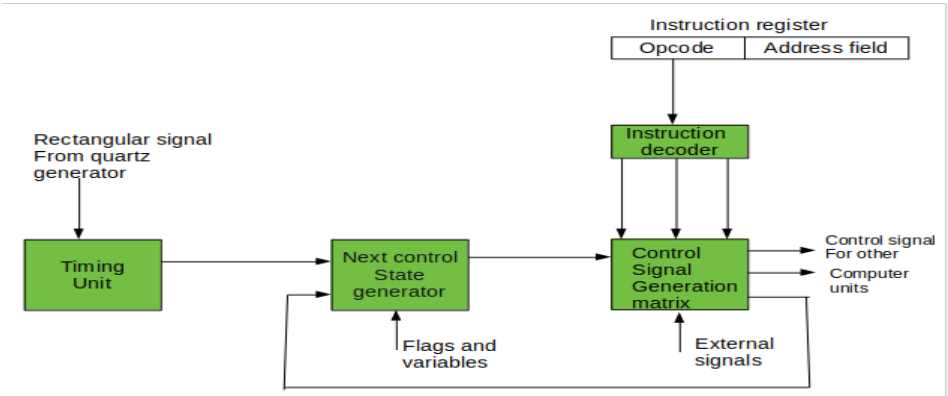
A computer is a complex system that performs various tasks through the coordination of different components. Here are the fundamental functional blocks of a typical computer system:

1. **Central Processing Unit (CPU):** The CPU is the brain of the computer. It executes instructions stored in the computer's memory, performs calculations, and manages data flow within the system.
2. **Memory (RAM):** Random Access Memory (RAM) is used for temporary data storage that the CPU can quickly access. It stores data and instructions that are actively being used by the CPU, providing fast read and write access.
3. **Storage:** This includes both primary storage (like Hard Disk Drives (HDDs) or Solid-State Drives (SSDs)) and secondary storage (external drives, optical discs, etc.). Primary storage is used for long-term data storage, while secondary storage is often used for backups and additional data.
4. **Motherboard:** The motherboard is the main circuit board that connects and facilitates communication between various components, such as the CPU, memory, storage devices, and peripherals.
5. **Input Devices:** These are devices that allow users to input data into the computer, such as keyboards, mice, touchpads, and other pointing devices.
6. **Output Devices:** These devices display or output information from the computer to the user, such as monitors, printers, and speakers.
7. **Power Supply Unit (PSU):** The power supply unit converts electrical power from an outlet into a form that the computer's components can use.
8. **Graphics Processing Unit (GPU):** The GPU is responsible for rendering graphics and images. It is crucial for tasks like gaming, video editing, and graphical design.
9. **Network Interface Card (NIC):** This component allows the computer to connect to a network, enabling communication with other devices.
10. **Peripheral Devices:** Additional devices that can be connected to the computer, such as printers, scanners, external drives, and more.
11. **System Bus:** It is a communication pathway that connects the CPU to other components on the motherboard, allowing data transfer between them.

Control Unit :- subcomponent of a central processing unit (CPU) that manages a computer's operations. The control unit fetches instructions from the CPU's memory, represented in bits, and translates those instructions into control signals in the form of pulses of electricity or light. There are two types of control units: **hardwired and microprogrammed.**

Hardwired

A hardwired control is a method of generating control signals with the help of Finite State Machines (FSM). It's made in the form of a sequential logic circuit by physically connecting components such as flip-flops, gates, and drums that result in the finished circuit. As a result, it's known as a hardwired controller.

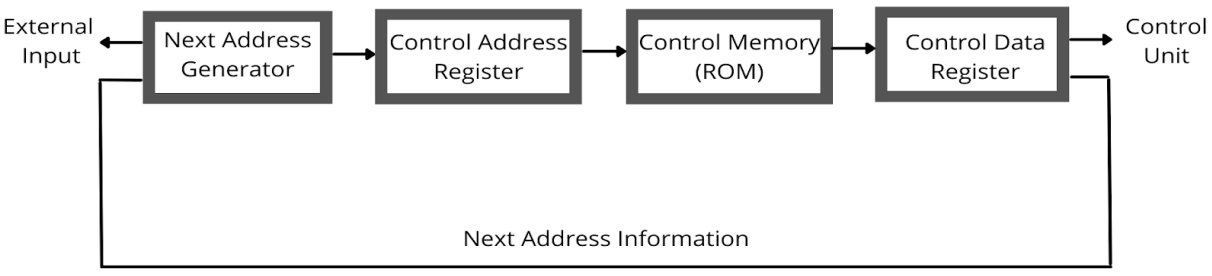


Block diagram of a hardwired control unit of a computer

s

Microprogrammed.

The programming approach is used to implement a microprogrammed control unit. A program made up of microinstructions is used to carry out a series of micro-operations. The control unit's control memory stores a microprogram composed of microinstructions. The creation of a set of control signals is dependent on the execution of a microinstruction.



 # Instruction set architecture (ISA) of a CPU register:

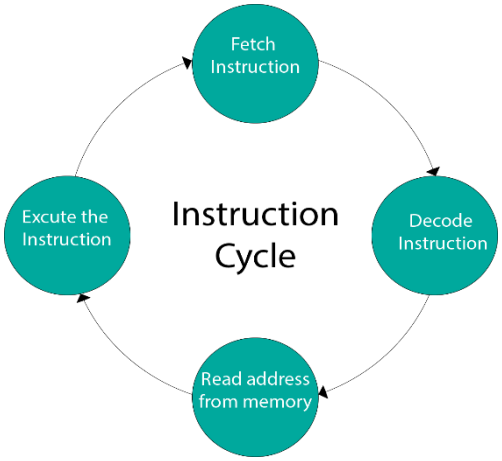
Instruction Set Architecture (ISA) is a set of rules and specifications that define the software interface to a CPU. It encompasses the instructions that a CPU can execute, the format of those instructions, the memory addressing modes, and the register architecture. Registers are small, fast storage locations within the CPU that hold data temporarily during program execution. The ISA defines the number of registers, their types, and their roles in the instruction set.

The register architecture in an ISA typically includes several types of registers, each serving a specific purpose:

1. **Data Registers (General-Purpose Registers):** These registers store data that the CPU uses during arithmetic and logic operations. They are general-purpose and can hold operands and intermediate results.
2. **Address Registers:** These registers store memory addresses or offsets. They are used in memory access operations, such as loading data from or storing data to memory.
3. **Status Registers (Flags):** These registers hold status information about the results of operations. Common flags include zero flag, carry flag, sign flag, and overflow flag. They help in making decisions based on the outcome of operations.
4. **Instruction Pointer (Program Counter):** This register keeps track of the memory address of the next instruction to be fetched and executed.
5. **Stack Pointer:** This register points to the top of the stack in memory. It is crucial for managing the program's call stack.
6. **Index Registers:** These registers are used for indexed addressing modes, where an operand's address is calculated by adding an index to a base address.

 # Instruction Execution Cycle

The Instruction Execution Cycle, also known as the instruction cycle or machine cycle, is the series of steps that a central processing unit (CPU) follows to fetch, decode, and execute an instruction from memory. It is a fundamental concept in computer architecture and is essential for understanding how a CPU processes instructions. The instruction cycle typically consists of the following stages:



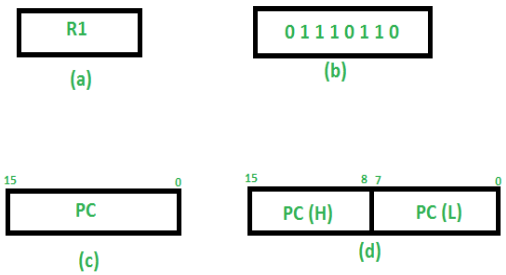
1. **Fetch (IF - Instruction Fetch):**
    - The CPU fetches the next instruction from the memory.
    - The address of the instruction to be fetched is provided by the program counter (PC).
2. **Decode (ID - Instruction Decode):**
    - The fetched instruction is decoded to determine the operation to be performed and the operands involved.
    - The opcode (operation code) is identified.
3. **Execute (EX - Execution):**
    - The CPU performs the operation specified by the decoded instruction.
    - This may involve arithmetic and logic operations, data movement, or control transfer.
4. **Memory Access (MA - Memory Access):**
    - If the instruction involves accessing memory (e.g., loading or storing data), the CPU performs the necessary read or write operations.
5. **Write Back (WB - Write Back):**
    - The results of the executed instruction are written back to the appropriate registers or memory locations.
    - This stage is particularly relevant for instructions that produce a result that needs to be stored.

RTL Interpretation of instructions

In symbolic notation, it is used to describe the micro-operations transfer among registers. It is a kind of intermediate representation (IR) that is very close to assembly language, such as that which is used in a compiler. The term “Register Transfer” can perform micro-operations and transfer the result of operation to the same or other register.

Micro-operations:
The operation executed on the data store in registers are called micro-operations. They are detailed low-level instructions used in some designs to implement complex machine instructions.
Register Transfer:
The information transformed from one register to another register is represented in symbolic form by replacement operator is called Register Transfer.
Replacement Operator :
In the statement, R2 <- R1, **<-** acts as a replacement operator. This statement defines the transfer of content of register R1 into register R2.
here are various methods of RTL –

1. General way of representing a register is by the name of the register enclosed in a rectangular box as shown in (a).
2. Register is numbered in a sequence of 0 to (n-1) as shown in (b).
3. The numbering of bits in a register can be marked on the top of the box as shown in (c).
4. A 16-bit register PC is divided into 2 parts- Bits (0 to 7) are assigned with lower byte of 16-bit address and bits (8 to 15) are assigned with higher bytes of 16-bit address as shown in (d).

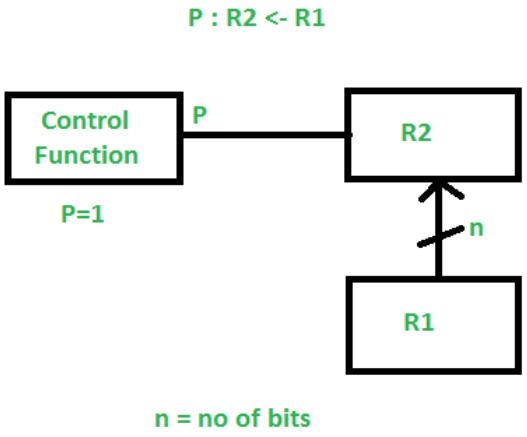


Letters and Numbers	Denotes a Register	MAR, R1, R2
( )	Denotes a part of register	R1(8-bit) R1(0-7)
<-	Denotes a transfer of information	R2 <- R1
,	Specify two micro-operations of Register Transfer	R1 <- R2 R2 <- R1
:	Denotes conditional operations	P : R2 <- R1 if P=1
Naming Operator (:=)	Denotes another name for an already existing register/alias	Ra := R

Register Transfer Operations:
The operation performed on the data stored in the registers are referred to as register transfer operations.

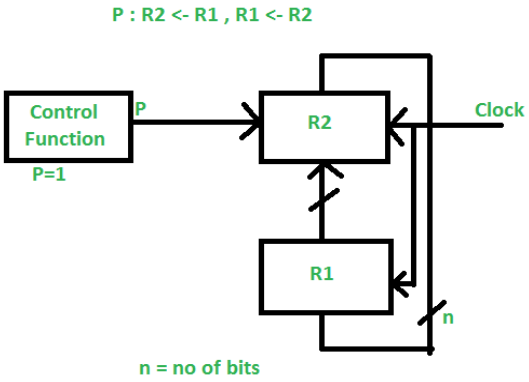
There are different types of register transfer operations:

1. Simple Transfer – R2 <- R1
The content of R1 are copied into R2 without affecting the content of R1. It is an unconditional type of transfer operation.
2. Conditional Transfer –



It indicates that if P=1, then the content of R1 is transferred to R2. It is a unidirectional operation.

3. Simultaneous Operations –
If 2 or more operations are to occur simultaneously then they are separated with comma (,).

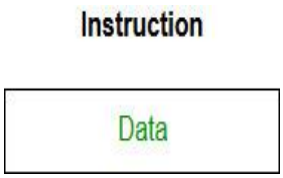


If the control function P=1, then load the content of R1 into R2 and at the same clock load the content of R2 into R1

 # ADDRESING MODES

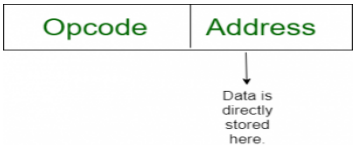
The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

1. **Implied or Implicit mode**: In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction. Zero address instruction are designed with implied addressing mode.



Example: CLC (used to reset Carry flag to 0)

2. **Immediate addressing mode** (symbol #): In this mode data is present in address field of instruction. Designed like one address instruction format.
Note: Limitation in the immediate mode is that the range of constants are restricted by size of address field.



Example: MVI A, AOH

AOH Generally transfers or copy the data in accumulator (A)

3. **Register mode:** In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction. *Here one register reference is required to access the data.*

Example: ADD B

A ← A+B

4. **Indirect mode**: In this addressing the operand's offset is placed in any one of the registers as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction. *Here two register reference is required to access the data.*

Example : MOV A, M (M stands for Memory Pointer)

5. **Direct addressing/ Absolute addressing Mode (symbol [ ]):** The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element. In this addressing mode the 16 bit effective address of the data is the part of the instruction. *Here only one memory reference operation is required to access the data.*

Example: LDA,C2OOH

LD: Load, A : Accumulator, C2OOH : Memory Address

(The data on this address load the content on accumulator)

Advantages of Addressing Modes

1. To give programmers to facilities such as Pointers, counters for loop controls, indexing of data and program relocation.
2. To reduce the number bits in the addressing field of the Instruction.

 # INSTRUCTION SET & CASE STUDY- INSTRUCTION SET OF 8085 PROCESSOR

INSTRUCTION SET: An instruction set is a group of commands for a central processing unit (CPU) in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations.

All CPUs have instruction sets that enable commands directing the CPU to switch the relevant transistors. The instructions tell the CPU to perform tasks. Some instructions are simple read, write and move commands that direct data to different hardware elements.

The instructions are made up of a specific number of bits. For instance, The CPU's instructions might be 8 bits, where the first 4 bits make up the operation code that tells the computer what to do. The next 4 bits are the operand, which tells the computer the data that should be used.

The length of an instruction set can vary from as few as 4 bits to many hundreds. Different instructions in some instruction set architectures (ISAs) have different lengths. Other ISAs have fixed-length instructions.

CASE STUDY- INSTRUCTION SET OF 8085 PROCESSOR

Instruction Set of 8085

- **Instruction**-An instruction is a binary code designed inside a microprocessor to perform a specific function.
- 8085 has 246 instructions.
- Each instruction is represented by an 8-bit binary value.

Classification of Instruction Set

- Data Transfer Instruction
- Arithmetic Instructions
- Logical Instructions
- Branching Instructions
- I/O & Machine Control Instructions

Data Transfer Instructions

- These instructions copy data from source to destination.
- These instructions move data between registers, or between memory and registers.
- While copying, the contents of source are not modified.

Data Transfer Instructions

- ***MOV Rd, Rs***- (Move the content of one register to another)
- This instruction copies the contents of the source register into the destination register.
- Example: MOV B, C, MOV A,B

Data Transfer Instructions

- ***MOV R, M -***(Move the content of memory to register)
- If one of the operands is a memory location, its location is specified by the contents of the HL registers.
- Example: MOV B, M, MOV A,M

Data Transfer Instructions

- ***MOV M, R -***(Move the content of register to memory)
- If one of the operands is a memory location, its location is specified by the contents of the HL registers.
- Example: MOV M, A MOV M,B

Data Transfer Instructions

- ***MVI R, Data-*** (Move data immediately in to the register)
- The 8-bit data is stored in the destination register
- Example: MVI B, 68H MVI C, 42H

Data Transfer Instructions

- ***MVI M, Data-*** (Move data immediately in to the memory)
- If the operand is a memory location, its location is specified by the contents of the H-L registers.
- Example: MVI M, 48H

Data Transfer Instructions

- ***LXI R_p Data(16 bit)***- (Load register pair immediate)
- This instruction loads 16-bit data in the register pair.
- Example: LXI H, 7084 H

Data Transfer Instructions

- ***LDA 16 bit address-*** (Load Accumulator directly)
- The contents of a memory location, specified by a 16- bit address in the operand, are copied to the accumulator.
- Example: LDA 3048 H

Data Transfer Instructions

- ***STA 16-bit address*** -(Store accumulator to given address)
- The contents of accumulator are copied into the memory location specified by the operand.
- Example: STA 6000 H

Data Transfer Instructions

- ***LHLD 16-bit address*** -(Load H-L registers direct)
- This instruction copies the contents of memory location pointed out by 16-bit address into register L.
- It copies the contents of next memory location into register H.
- Example: LHLD 2030 H

Data Transfer Instructions

- ***SHLD 16-bit address*** —(Store H-L registers direct)
- The contents of register L are stored into memory location specified by the 16-bit address.
- The contents of register H are stored into the next memory location.
- Example: SHLD 3250 H

Data Transfer Instructions

- **LDAX R_P** —(Only BC and DE register pair), (Load accumulator indirect)
- This instruction copies the contents of that memory location into the accumulator.
- Example: LDAX B

Data Transfer Instructions

- ***STAX R_P*** -(Store accumulator indirect)
- The contents of accumulator are copied into the memory location specified by the contents of the register pair.
- Example: STAX D

Data Transfer Instructions

- ***XCHG*** -(Exchange the content of H-L pair with D-E pair).
- The contents of register H are exchanged with the contents of register D.
- The contents of register L are exchanged with the contents of register E.
- Example: XCHG

Data Transfer Instructions

- ***SPHL*** -(Copy H-L pair to the Stack Pointer (SP)
- This instruction loads the contents of H-L pair into SP.
- Example: SPHL

Data Transfer Instructions

- ***XTHL*** –(Exchange H–L with top of stack)
- The contents of L register are exchanged with the location pointed out by the contents of the SP.
- The contents of H register are exchanged with the next location (SP + 1).
- Example: XTHL