30. (a) Design [illegible] binary u[illegible]unter.
    (b) Realize [illegible] flip flop usi[illegible]d T flip flops.

Design a full subtractor using half subtractors.

31. (i) Design a full adder using NOR gates.
    (ii) Design a 3*8 decoder and explain its operation as a minterm generator.

32. Explain with the suitable example how a multiplexer is used to implement the Boolean function.

(i) Design a full subtractor and implement it using logic gates.

(ii) Design a full adder using two half adders and an OR gate.

UNIT - 4

PROCESSING AND PIPIELINING

4.1 INSTRUCTION SET ARCHITECTURE (ISA)

★ Instruction Set Architecture (ISA) is the interface between hardware and software.

★ It defines the set of machine instructions a processor can execute, data types, registers, addressing modes, memory structure, and I/O mechanisms.

★ ISA acts like a contract between the hardware and the programmer/compiler.

★ ISA simplifies compiler design, improves execution efficiency, and helps achieve architectural scalability.

4.1.1 *RISC Architecture (Reduced Instruction Set Computer)*

★ These systems are based on the principle of **simplifying the instruction set** so that each instruction can execute within **a single clock cycle**.

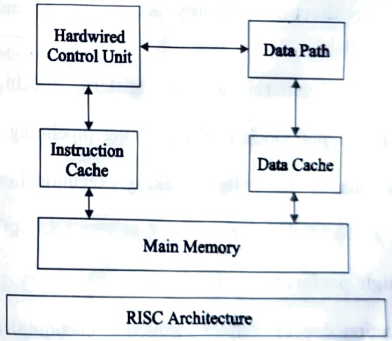


Fig. 4.1 Reduced Instruction Set Architecture

- ★ The architecture focuses on hardware efficiency and faster execution.
- ★ RISC simplifies processor design by using a small, uniform set of instructions.
- ★ Each instruction performs a basic operation (e.g., load, compute, store) and is designed to execute in a single clock cycle, enabling efficient pipelining and simpler hardware.

Characteristics of RISC

- ★ Simpler instruction, hence simple instruction decoding.
- ★ Instruction comes in the form of one word.
- ★ An instruction takes a single clock cycle to get executed.
- ★ More general-purpose registers for register-to-register operations.
- ★ Simple Addressing Modes.
- ★ Optimized for pipelining due to uniform instruction size and simplicity.

Key Features of RISC

- ★ **Simple and limited instruction set** (less complex operations).
- ★ **Fixed instruction format** (generally of equal length).
- ★ **Load/Store architecture** (memory access allowed only through load and store instructions).
- ★ **Large number of general-purpose registers**, reducing memory access.
- ★ **One instruction per clock cycle**, enabling pipelining.
- ★ Control unit usually **hardwired**, making execution faster.

Advantages

- ★ Supports **high performance pipelining.**
- ★ Faster execution due to simpler instruction decoding.
- ★ Efficient for real-time and embedded systems.

Disadvantages

- ★ Requires **large program size** due to simple instructions.
- ★ Compiler must optimize instruction grouping and scheduling.

Examples

ARM processors, MIPS, SPARC, RISC-V, and Power PC.

4.1.2 CISC Architecture (Complex Instruction Set Computer)

- ★ It focuses on reducing the number of instructions in a program by supporting **complex instructions**
- ★ It can execute several low-level operations such as memory access, arithmetic, and conditional jumps within a single instruction.
- ★ CISC reduces the number of instructions a program needs by using a large set of complex, variable-length instructions.
- ★ A single instruction can perform multiple operations (e.g., load, compute, and store), which may take multiple clock cycles.

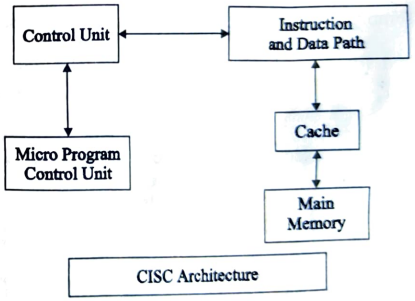


Fig. 4.2 Complex Instruction Set Architecture

Characteristics of CISC

- ★ Complex instruction, hence complex instruction decoding.

- ★ Instructions are larger than one-word size.
- ★ Instruction may take more than a single clock cycle to get executed.
- ★ Less number of general-purpose registers as operations get performed in memory itself.
- ★ Complex Addressing Modes.

Key Features of CISC

- ★ **Large and complex instruction set** (hundreds of instructions).
- ★ **Variable-length instructions**, making decoding more complicated.
- ★ Instructions can directly access memory.
- ★ Fewer registers; more memory-oriented processing.
- ★ Control unit usually **microprogrammable**, making it flexible but slower.

Advantages

- ★ Reduced program size due to complex instructions.
- ★ Easier assembly programming due to rich instruction set.
- ★ Better suited for general-purpose desktops and laptops.

Disadvantages

- ★ Slower instruction execution and difficult pipelining.
- ★ Costlier and more complex hardware design.

Examples

Intel x86 family, VAX, IBM 370.

4.1.3 Key Differences in RISC vs CISC

RISC and CISC are two different ways of designing computer processors.

- ★ RISC uses a small set of simple, fixed-size instructions designed to execute in a single clock cycle.

- ★ CISC includes a larger set of instructions, many of which are complex and can perform multiple operations (e.g., memory access and computation) in a single instruction.
- ★ CISC instructions often require multiple clock cycles.

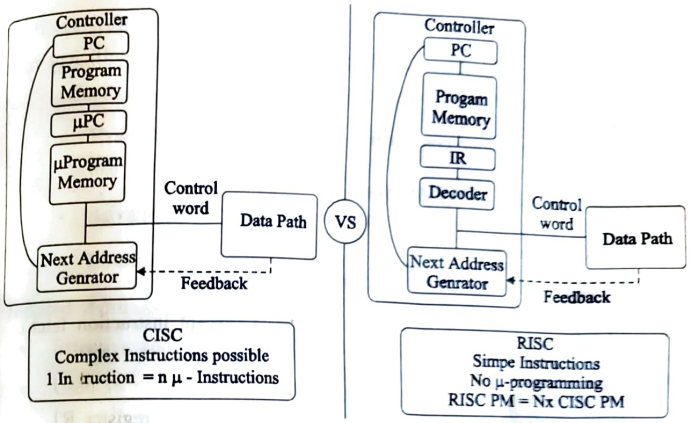


Fig. 4.3 RISC vs CISC

Table 4.1 Differences between RISC vs CISC

Feature	RISC	CISC
Instruction Size	Small, fixed	Large, variable
Execution Speed	Fast (usually 1 cycle)	Slow (multiple cycles)
Memory Access	Load/Store	Memory operations within instructions
Control Unit	Hardwired	Microprogrammed
Registers	Many	Few
Program Size	Larger	Smaller
Optimization	Compiler dependent	Hardware dependent

4.2 ADDRESSING MODES

★ Addressing modes specify **how the operand of an instruction is chosen** (or) **where the data resides** during program execution.

★ Operands may be located in registers, memory (or) be part of the instruction itself.

★ Addressing modes increase programming flexibility, allow efficient memory utilization, support pointers, indexing, and arrays.

★ It alsoenables better code optimization by the compiler.

4.2.1 *Common Addressing Modes*

1. Immediate Addressing

★ Operand is directly present within the instruction itself.

★ No memory access is needed to fetch data (except instruction fetch).

★ Used for constants during execution.

- **Example:** MOV R1, #5 → Load constant 5 into register R1.
- **Advantage:** Fast execution because no extra memory access.
- **Limitation:** Operand value is fixed and limited by instruction size.

2. Register Addressing

★ Operand is located in a **CPU register.**

★ Instruction specifies register name, not memory location.

- **Example:** ADD R1, R2 → Add contents of R2 with R1.
- **Advantage:** Very fast since registers are inside CPU.
- **Limitation:** Number of registers is limited.

3. Direct Addressing (Absolute Addressing)

★ Instruction contains the **full memory address** of the operand.

- **Example:** LOAD R1, 1000 → Load value from memory location 1000.
- **Advantage:** Simple to use.
- **Limitation:** Address must fit inside the instruction; less flexible for dynamic programs.

4. Indirect Addressing

★ Instruction gives the **address of a memory location** that contains the **effective address (actual data location).**

- **Example:** LOAD R1, (R2) → R2 stores an address; data fetched from that address.
- **Advantage:** Enables pointer-based programming.
- **Limitation:** Requires extra memory access.

5. Indexed Addressing

★ Effective address = **Base Address + Index.**

★ Used in accessing arrays and tables.

- **Example:** LOAD R1, 1000(R2) → Address = 1000 + contents of R2.
- **Advantage:** Ideal for array traversal.
- **Limitation:** Requires additional hardware for index calculations.

6. Base Register Addressing

★ Effective Address = **Content of Base Register + Offset.**

★ Base register points to a memory segment; offset specifies displacement.

- **Example:** LOAD R1, 20(R3) → Address = 20 + contents of R3.
- **Advantage:** Supports relocatable and modular programming.
- **Used in:** Operating systems, memory management.

7. Relative Addressing

★ Effective Address = **Program Counter (PC) + Offset.**

★ Used in branch and jump instructions.

- **Example:** JMP LABEL → Offset added to PC.
- **Advantage:** Useful for dynamic code movement, position-independent code.
- **Used in:** Compilers and OS memory loading.

8. Register Indirect Addressing

★ Register contains memory address; operand fetched from that address.

- **Example:** MOV R1, (R3) → Fetch data from memory pointed by R3.
- **Advantage:** Fast pointer referencing (no explicit memory address).
- **Common in:** Linked lists, stacks, dynamic memory access.

Table 4.2: Summary of Addressing Mode

Addressing Mode	Operand Location	Example
Immediate	Inside instruction	MOV R1, #10
Register	CPU register	ADD R1, R2
Direct	Memory address given	LOAD R1, 2000
Indirect	Address stored in memory/register	LOAD R1, (R2)
Indexed	Base + Index	LOAD R1, 1000(R2)
Base Register	Base register + Offset	LOAD R1, 20(R3)
Relative	PC + Offset	JMP LABEL

Applications of Addressing Modes

★ **Indexed addressing** → Array and matrix computations.

★ **Relative addressing** → efficient branching and looping.

★ **Indirect and register indirect addressing** → Pointer-based data structures (trees, linked lists).

★ **Immediate addressing** → Constant initialization and small arithmetic tasks.

4.3 CONTROL UNIT IN A CPU

★ The Control Unit (CU) is the "brain inside the brain" of a computer processor.

★ ALU performs arithmetic and logic operations, it is the CU that decides **which operation must be performed, when it should be executed, and which data should be used.**

★ It accomplishes this through a set of control signals that coordinate data movement among registers, activate ALU functions, manage program counters, control memory access, and regulate instruction decoding.

★ Without a CU, the CPU would not have ordered instruction execution, leading to random and erroneous results.

★ Therefore, the CU directly impacts the **performance, efficiency, power consumption, and functional capabilities of a computer system.**

4.3.1 Operation of Control Unit

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.

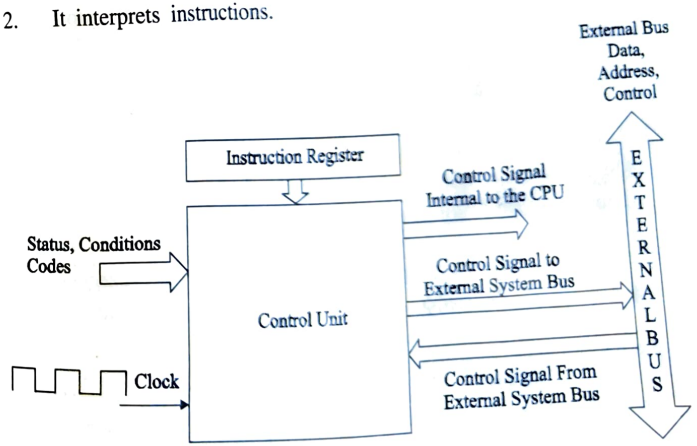


Fig. 4.4 Control Unit

4.7.3 Pipeline Throughput and Latency

Latency

- ★ Latency is the time taken by a single instruction to pass through the entire pipeline.
- ★ Since there are 5 stages, latency = 5 clock cycles.

Throughput

- ★ Throughput is the number of instructions completed per unit time.
- ★ After the pipeline fills, throughput = 1 instruction per cycle.
- ★ Pipelining improves throughput, not latency.

Pipeline Depth and Performance

Increasing the number of pipeline stages (i.e., deep pipelines):

- ★ Reduces stage delays → allows higher clock speeds.
- ★ But increases complexity due to hazards and need for forwarding, stall logic.
- ★ Example:

1. Intel Pentium 4 had a 20+ stage pipeline for high frequency performance.

Too deep pipelines can degrade performance due to frequent branch mispredictions.

4.8 PIPELINE HAZARDS

- ★ In a pipelined processor, multiple instructions execute simultaneously without interference. However, in real-world systems, various limitations and dependencies cause delays (or) conflicts during pipeline execution. These interruptions are known as **pipeline hazards**.
- ★ Hazards reduce the efficiency of pipelining by forcing the processor to wait (or) insert idle cycles called *pipeline stalls* (or) *bubbles*.

- ★ A pipeline hazard is a situation that prevents the next instruction in the pipeline from executing in the expected clock cycle. As a result, one (or) more pipeline stages become idle, leading to reduced throughput.
- ★ A pipeline hazard is any condition that causes a pipeline to stall (or) operate inefficiently, disrupting the smooth flow of instruction execution. Hazards reduce overall performance by introducing extra clock cycles, called 'stalls' (or) 'bubbles', into the pipeline.

Effects of Hazards

- ★ Introduce delays in instruction execution.
- ★ Decrease overall speedup of pipelined architecture.
- ★ Increase hardware complexity for hazard management.
- ★ May require additional techniques like forwarding, branch prediction, and stalling.

Types of Pipeline Hazards

Pipeline hazards are broadly classified into three types:

1. Structural Hazards
2. Data Hazards
3. Control Hazards

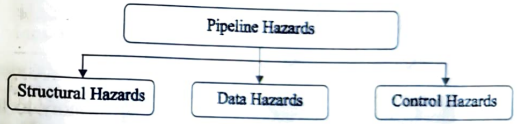


Fig. 4.11

4.8.1 Structural Hazards (Resource Conflicts)

A structural hazard, also called a resource conflict, occurs when two (or) more instructions require access to the same hardware resource simultaneously, and the hardware cannot support the required parallel access.

Structural hazards occur when multiple instructions compete for the same hardware resources at the same time. This happens when processor resources are insufficient to support overlapping execution of instructions.

Example

If a single memory unit is used for both **instruction fetch (IF)** and **data access (MEM)**, then:

- ★ One instruction might be fetching an instruction.
- ★ Another instruction might be reading/writing data simultaneously.

This leads to a conflict, as both need the same memory unit at the same time.

Example:

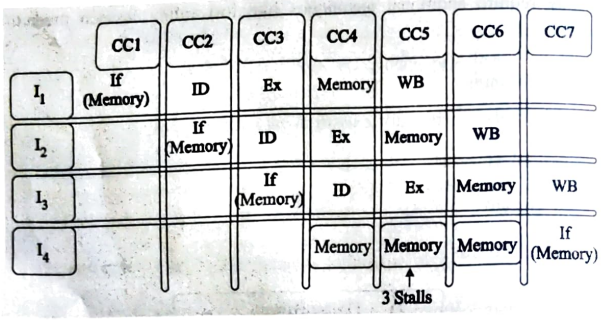


Fig 4.12

- ★ In clock cycle 4, suppose instruction I1 is accessing memory to load data, and instruction I4 is trying to **fetch** its instruction from memory at the same time.
- ★ Since both operations use the same memory module, they cannot proceed concurrently.
- ★ As a result, one of them (say I4) is stalled until the resource becomes free, creating a *bubble* in the pipeline.

Common resources involved in conflicts:

- ★ ALU (or) functional units
- ★ Memory
- ★ Registers
- ★ Buses

Solution to Structural Hazards:

To minimize such stalls, one common technique is **Resource Duplication** (or) **Memory Split**, where:

- ★ **Instruction Memory (Code Memory)** stores program instructions.
- ★ **Data Memory** stores program data.

This separation allows simultaneous access—one stage can fetch instructions while another accesses data—thus avoiding memory-related structural dependencies.

RISC processors commonly solve structural hazards by using **separate instruction and data memories**.

Table 4.5

Technique	Description
Resource Duplication	Using separate instruction and data memory (Harvard architecture)
Pipelined Memory Units	Allow multiple accesses in different phases
Cache Usage	Instruction and data caches reduce access congestion

4.8.2 Data Hazards (Operand/Data Dependencies)

- ★ Data hazards occur when an instruction depends on the result of a previous instruction that has not yet completed its pipeline execution.

Types of Data Dependencies

- ★ **True Dependency (Read after Write - RAW):** A later instruction needs a value that an earlier instruction writes.

★ **Anti-dependency (Write after Read - WAR):** A later instruction writes to a location that an earlier instruction is reading.

★ **Output Dependency (Write after Write - WAW):** Two instructions write to the same destination register (or) memory location.

Example

Consider these instructions:

I1: ADD R1, R2, R3

I2: SUB R4, R1, R5 ← depends on R1 (result of I1)

I2 needs the output of I1, but the value may not be ready when I2 reaches EX stage.

Solutionsfor Data Hazards

Technique	Description
Pipeline Stalling	Temporarily stops pipeline until data is available
Data Forwarding /Bypassing	ALU output is directly passed to dependent instruction without waiting for WB
Compiler Scheduling	Reorders instructions to minimize data hazards

4.8.3 Control Hazards (Branch and Jump Uncertainty)

★ Control hazards occur due to changes in program flow caused by branch, **jump, and interrupt** instructions. The processor cannot fetch the correct next instruction until the branch decision is known.

Example

BEQ R1, R2, Label

ADD R3, R4, R5 ← next instruction unknownuntil branch resolves

If branch is taken, the next instruction is at Label. If not taken, the next instruction follows sequentially. The pipeline becomes uncertain.

Techniques to Handle Control Hazards

Method	Description
Pipeline Flushing	Any wrong instructions fetched are discarded
Branch Prediction	CPU predicts the branch direction (taken/not taken)
Delayed Branching	Compiler places useful instruction in the branch delay slot
Speculative Execution	Executes predicted path; discards if incorrect

Branch Prediction Types

★ Static Prediction (e.g., always assume branch not taken)

★ Dynamic Prediction using branch history buffers and predictors

Modern CPUs use **advanced dynamic branch predictors** (e.g., Two-level, Tournament Predictors) to minimize pipeline penalties.

4.8.4 Penalty Due to Hazards

Hazards decrease pipeline performance. The effective speedup considering stalls is:

Effective Speedup = Ideal Speedup / (1 + Stall Penalty)

Hazard Type	Main Cause	Typical Solution
Structural	Resource conflict	Resource duplication
Data	Operand dependency	Forwarding, stalling
Control	Branch uncertainty	Prediction, speculations

4.9 INSTRUCTION LEVEL PARALLELISM (ILP)

★ Instruction-Level Parallelism (ILP) is the capability of a processor to execute multiple instructions at the same time by overlapping their execution to increase performance.

3. It controls data flow inside the processor.
4. It receives external instructions (or) commands to which it converts to sequence of control signals.
5. It controls many execution units (i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

4.3.2 Types of Control Unit

There are two types of control units:

1. Hardwired control and
2. Micro programmed control.

4.4 HARDWIRED CONTROL UNIT

- ★ A Hardwired Control Unit uses a network of fixed digital circuits such as decoders, counters, multiplexers, and combinational logic to produce control signals.
- ★ Each step in an instruction's execution is mapped into hardware.
- ★ For instance, when an instruction is fetched, the control unit's logic will automatically trigger memory read signals, increment the program counter, and load the instruction into the instruction register.
- ★ These operations occur because fixed logic circuits are permanently wired to generate specific outputs for each instruction opcode and clock pulse.

The internal structure typically includes:

- ★ **Instruction Decoder** – Identifies which operation needs to be performed.
- ★ **Sequencing Logic Circuit** – Controls the step-by-step progression of execution cycles.

- ★ **State Registers** – Store the current state of execution.
- ★ **Control Signal Generator** – Produces specific control outputs based on state, instruction, and clock.

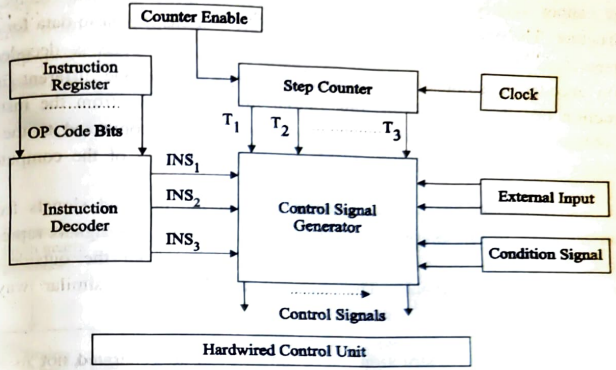


Fig. 4.5 Hardwired Control Unit

Characteristics:

- ★ Since control logic is directly implemented in circuits, the system behaves like a deterministic state machine where each instruction triggers a predefined series of events.
- ★ The system has **minimal latency**, since the CU does not need to fetch microinstructions.
- ★ However, the logic becomes exceedingly complex when supporting **large instruction sets, multiple addressing modes (or) advanced features such as memory protection and pipelining**.

4.4.1 Block Diagramof Hardwired Control Unit

In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we cannot modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode. As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for execution units of the computer.

This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupts signals. The matrices are built in a similar way as a programmable logic arrays.

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit. A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generatedby the timing unit based on the rectangular patterns usually supplied by the quartz generator.

When a new instruction arrives at the control unit, the control units are in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computerremain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state. This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing).

The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle. The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

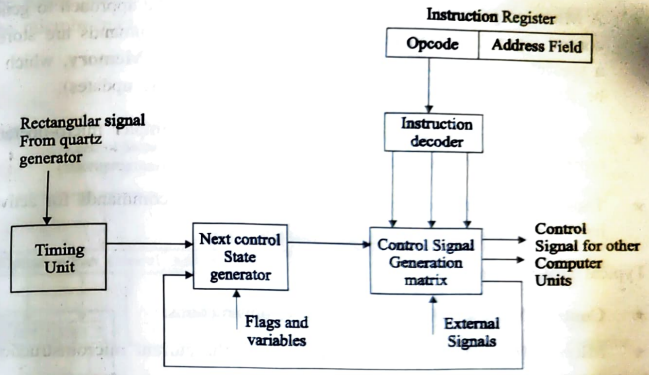


Fig. 4.6 Block Diagram of Hardwired control

Advantages:

- ★ **Blazing Fast Execution:** No microcode access; direct hardware control.
- ★ **Power Efficient:** Fewer memory accesses for control instructions.
- ★ **Ideal for RISC Models:** Simple instructions benefit from direct wired control, enabling high-frequency execution and deep pipelining.

Limitations:

- ★ **Difficult to Extend:** Adding a new instruction (or) addressing mode requires redesigning logic circuits.

- ★ **Complexity Explodes with CISC Features:** Large instruction sets make wiring impractical.
- ★ **Hardware Debugging Complexity:** If error occurs, entire chip redesign may be required.

4.5 MICRO PROGRAMMED CONTROL UNIT

- ★ A Micro programmed Control Unit uses a software-like approach to generate control signals. Instead of hardware logic, control commands are stored in a special memory called **Control Store** (or) **Control Memory**, which may be ROM, EPROM (or) RAM (for writable microcode updates).
- ★ Each instruction is broken down into a series of simpler micro-operations, represented in **microinstructions**.
- ★ These microinstructions resemble low-level control commands for activating hardware components.

Typical structure includes:

- ★ **Control Memory (CM)** – Stores microinstructions.
- ★ **Microinstruction Register (MIR)** – Holds the current microinstruction for execution.
- ★ **Microprogram Counter (Micro-PC)** – Determines the next microinstruction.
- ★ **Sequencer** – Branches microinstruction flow based on conditions.
- ★ **Control Signal Decoder** – Interprets microinstruction bits into hardware signals.

4.5.1 *Block Diagram of Micro programmed control*

The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution. In micro programmed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

1. With a single-level control store

In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a micro program that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoders, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction micro program and a control field used to control activities of the microinstruction address generator.

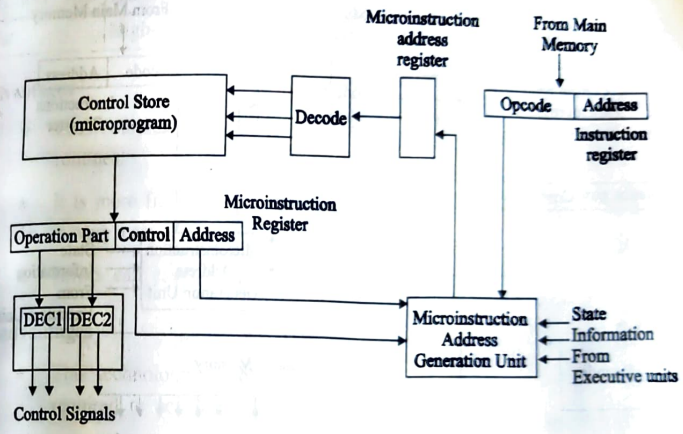


Fig. 4.7 Micro programmed control with a single-level control store

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

2. With a two-level control store

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

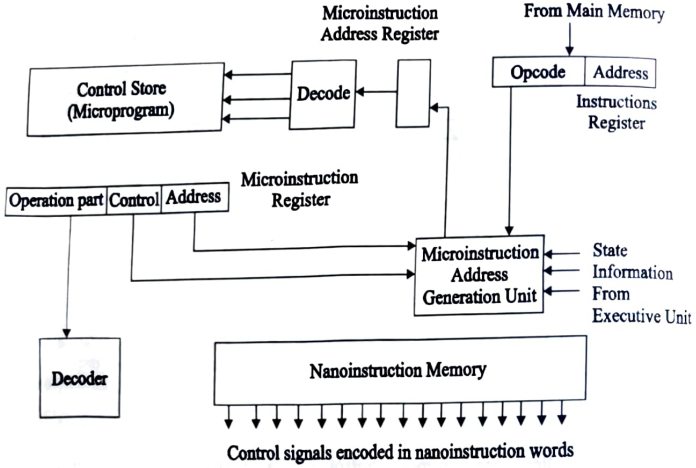


Fig. 4.8 Micro programmed control with a two-level control store

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

4.5.2 Types of Microinstruction Organization

Microinstructions can be organized in two styles:

Table 4.3 comparison of Microinstruction Organization

Feature	Horizontal Microprogramming	Vertical Microprogramming
Width	Wider (contains many control bits)	Narrow (encoded fields)
Decoding	Not required	Required
Speed	Very Fast	Slower
Memory Size	Larger	Smaller
Use	High-end CPUs	Low-cost systems

Advantages

- ★ Less complex to design because micro-program is implement using software routines.
- ★ It is more flexible because design modification, correction, and enhancement is very easy
- ★ Errors can easily remove.

Disadvantages

- ★ This technology is slower than the hardwired control unit because time is required to access the micro-instructions from control memory is larger.
- ★ It is more expensive than hardwired due to use of ROM.
- ★ Required more Chip area as compare to hardwire

4.5.3 Comparison of Hardwired and Micro programmed control

Table 4.4 Comparison of Hardwired and Micro programmed control

Aspect	Hardwired Control	Microprogrammed Control
Implementation	Logic Circuits	Firmware-based Microcode
Speed	Faster	Slower
Flexibility	Poor	Excellent

Aspect	Hardwired Control	Microprogrammed Control
Cost	Low for simple designs	Higher due to control memory
Best For	RISC & embedded systems	CISC & complex architectures
Debugging	Difficult	Easier due to software-like structure
Instruction Modification	Requires hardware redesign	Modify microcode only

4.6 CONCEPTS OF PIPELINING

- ★ Pipelining is one of the most powerful performance enhancement techniques used in modern processors.
- ★ It improves CPU throughput by dividing the execution of instructions into separate stages and executing multiple instructions simultaneously.
- ★ Instead of completing one entire instruction before starting the next, the processor overlaps different phases of multiple instructions.
- ★ This is similar to an assembly line in a factory, where each worker performs only one specific task and passes the product to the next stage.

Definition

- ★ Pipelining is a technique in which the instruction execution process is split into sequential stages such as fetch, decode, execute, memory access, and write-back.
- ★ Each stage works independently and processes a different instruction at the same time.
- ★ Hence, pipelining increases the instruction throughput (number of instructions executed per unit time) without increasing the clock speed of the processor.
  - Improves performance by parallel execution of instruction components.
  - Reduces overall execution time of a program by overlapping operations.
  - Enhances system efficiency without requiring complex hardware changes.

4.6.1 Need for Pipelining

Traditional non-pipelined processors execute only one instruction at a time, causing:

- ★ Idle CPU resources.
- ★ Longer program execution time.
- ★ Underutilization of available hardware.

Pipelining maximizes hardware utilization by ensuring that all parts of the CPU remain active, thus increasing processing speed. It is especially beneficial in high-speed microprocessors and advanced computing architectures such as RISC, superscalar, and parallel processors.

4.6.2 Basic Pipeline Stages

A typical 5-stage pipeline used in RISC processors (e.g., MIPS) includes:

Here, multiple instructions are executed simultaneously in different stages.

Stage	Name	Description
IF	Instruction Fetch	Fetches the instruction from memory
ID	Instruction Decode	Decodes and interprets instruction
EX	Execute / ALU	Performs arithmetic/logic operations
MEM	Memory Access	Reads/Writes data from/to memory
WB	Write Back	Stores computed result into registers

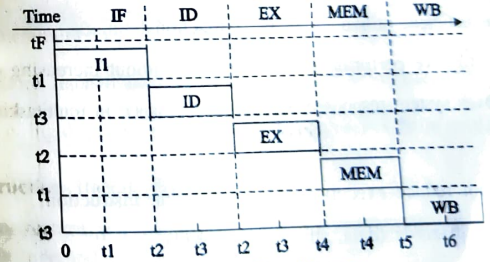


Fig. 4.9 Pipeline Diagram

- ★ Instead of running instructions strictly one after another, ILP allows a processor to find and execute independent instructions in parallel, reducing the total number of clock cycles needed to complete a program.
- ★ This is achieved through hardware and compiler techniques like pipelining and superscalar execution

Parallel Computing:

- ★ Parallel computing is the use of multiple processors to run an application (or) task at the same time. Through increased system calculation capacity, parallel computing enables quicker application processing and task execution.
- ★ Large Instructions are frequently broken down into independent, smaller, usually related pieces that can be processed all at once in this form of computing architecture.

Types of Parallel Computing:

1. Bit Level Parallelism
2. Instruction Level Parallelism
3. Task Parallelism

4.9.1 Instruction Level Parallelism (ILP) Architecture

Instruction Level Parallelism is achieved when multiple operations are performed in a single cycle, which is done by either executing them simultaneously (or) by utilizing gaps between two successive operations that are created due to the latencies. Now, the decision of when to execute an operation depends largely on the compiler rather than the hardware. However, the extent of the compiler's control depends on the type of ILP architecture where information regarding parallelism given by the compiler to hardware via the program varies.

4.9.2 Classification of ILP Architectures

The classification of ILP architectures can be done in the following ways -

- ★ Sequential Architecture: Here, the program is not expected to explicitly convey any information regarding parallelism to hardware, like superscalar architecture.
- ★ Dependence Architectures: Here, the program explicitly mentions information regarding dependencies between operations like dataflow architecture.
- ★ Independence Architecture: Here, programme m gives information regarding which operations are independent of each other so that they can be executed instead of the 'nops.

In order to apply ILP, the compiler and hardware must determine data dependencies, independent operations, and scheduling of these independent operations, assignment of functional units, and register to store data.

Advantages:

- ★ **Improved Performance**: ILP can significantly improve the performance of processors by allowing multiple instructions to be executed simultaneously (or) out-of-order. This can lead to faster program execution and better system throughput.
- ★ **Efficient Resource Utilization**: ILP can help to efficiently utilize processor resources by allowing multiple instructions to be executed at the same time. This can help to reduce resource wastage and increase efficiency.
- ★ **Reduced Instruction Dependency**: ILP can help to reduce the number of instruction dependencies, which can limit the amount of instruction-level parallelism that can be exploited. This can help to improve performance and reduce bottlenecks.
- ★ **Increased Throughput**: ILP can help to increase the overall throughput of processors by allowing multiple instructions to be executed simultaneously (or) out-of-order. This can help to improve the performance of multi-threaded applications and other parallel processing tasks.

Disadvantages:

Increased Complexity: Implementing ILP can be complex and requires additional hardware resources, which can increase the complexity and cost of processors.

- ★ **Instruction Overhead:** ILP can introduce additional instruction overhead, which can slow down the execution of some instructions and reduce performance.
- ★ **Data Dependency:** Data dependency can limit the amount of instruction-level parallelism that can be exploited. This can lead to lower performance and reduced throughput.
- ★ **Reduced Energy Efficiency:** ILP can reduce the energy efficiency of processors by requiring additional hardware resources and increasing instruction overhead. This can increase power consumption and result in higher energy costs.

4.10 PARALLEL PROCESSING CONCEPT

- ★ It defines the simultaneous execution of multiple tasks (or) calculations by using more than one processor to increase a computer's speed and efficiency.
- ★ It involves breaking down a large task into smaller subtasks that are processed concurrently by different processing units, such as CPUs (or) GPUs.
- ★ It is distinguished between parallel and serial operations by the type of registers used at the lowest level

1. Shift Register:
2. Floating point Operations

Working Process:

- ★ **Task division:** A complex problem is divided into smaller, independent subtasks.
- ★ **Simultaneous execution:** Multiple processors, which can be cores within a CPU (or) a specialized GPU, work on these subtasks at the same time.
- ★ **Assembly:** Once the subtasks are complete, the results are combined to provide the final solution.

Benefits:

- ★ **Increased speed:** By performing multiple operations at once, parallel processing significantly reduces computation time.
- ★ **Higher throughput:** More tasks can be completed in a given period.
- ★ **Improved efficiency:** It allows for the more efficient use of processing units, especially for complex computations.

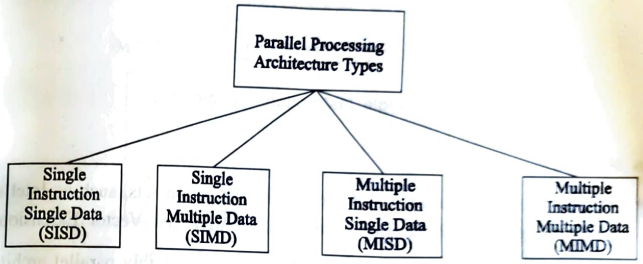


Fig. 4.13

4.10.1 Single Instruction Multiple Data (SIMD)

- ★ SIMD, which stands for Single Instruction Multiple Data, is a type of parallel processing where a single instruction operates on multiple data points simultaneously.
- ★ This architecture is efficient for tasks like image, video, and scientific processing that involve performing the same operation on large datasets

Working Process:

- ★ **Single instruction, multiple data:** A single instruction, such as "add," is issued, but it is applied to multiple pairs of data at the same time.
- ★ **Parallel data processing:** This allows for a significant speedup compared to a traditional scalar processor, which would have to perform the operation one data pair at a time.

14. Describe the architec... instructions and vector ...
15. Explain the concept of arr... processors.
16. Explain pipelining, hazards, and s... ith ne... (N...
17. Discuss in detail the types of parallel processing a... pplications. (Apr/May 2023)
18. With neat sketches, discuss the working of superscalar, vector, and array processors. (Nov/Dec 2022)
19. Write a short note on Instruction Execution
20. Explain the basic MIPS implementation with necessary multiplexers and controllines
21. Draw and explain the data path to implement instruction fetch and PC incrementoperations
22. Draw and explain the data path to implement arithmetic - logic instructions
23. Draw and explain the data path segment for computation of branch target address
24. Explain data path in detail
25. Draw and explain the simple data path with the control unit and explain theexecution of ALU instructions.
26. Draw and explain Hardwired control unit
27. Explain the concepts of microprogrammed unit
28. Discuss the basic concepts of pipelining and pipeline stages in MIPS Architecture
29. Discuss the various hazards that might arise in a pipeline
30. Discuss about pipelined data path and control.
31. What is a data hazard? How do you overcome it? And discuss its side effects?
32. Describe the techniques for handling control hazards in pipelining?
33. Explain dynamic branch prediction technique?

UNIT - 5

MEMORY

5.1 MEMORY

A physical device that stores data (or) information temporarily (or) permanently in it is called memory. Its a device where data is stored and processed. In common, a computer has primary and secondary memories. Auxiliary (secondary) memory stores data and programs for long-term storage (or) until the time a user wants to keep them in memory, while main memory stores instructions and data during programme execution; hence, any programme (or) file that is currently running (or) executing on a computer is stored in primary memory.

5.1.1 *Types of Computer Memory*

Computer memory comes in various types and serves different purposes.

1. **Primary Memory** (RAM - Random Access Memory) - Volatile memory loses its contents when the machine is turned off. RAM stores the data that is actively being used. During the booting process of a system, the operating system actively uses RAM and applications that are necessary to execute a file (or) a program. It speeds up CPU processing by providing fast data and instruction access.

2. **Secondary Memory** (Storage) - Secondary Memory is also known as permanent memory (or) non-volatile memory of a computer. Secondary memory retains data when the machine shuts down. Files, programmes, and the OS are stored there permanently. HDDs, SSDs, USB flash drives, and optical discs are non-volatile memory devices.

3. **Cache Memory** - Memory that is smaller and faster than RAM is called cache memory. It is placed closer to the CPU than the RAM. It saves data and instructions that are used a lot so that processing goes faster.

4. **Register Memory** - Register memory, which is also called processor registers (or) "registers," is the smallest and fastest type of computer memory that is directly

integrated into the CPU. Registers are small, fast storage units inside the CPU that are used to quickly store data that is being processed (or) instructions that are being run.

5.2 MEMORY HIERARCHY

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time.

The memory in a computer can be divided into five hierarchies based on the speed as well as use. The processor can move from one level to another based on its requirements. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three hierarchies are volatile memories which mean when there is no power, and then automatically they lose their stored data. Whereas the last two hierarchies are not volatile which means they store the data permanently.

A memory element is the set of storage devices which stores the binary data in the type of bits. In general, the storage of memory can be classified into two categories such as volatile as well as non- volatile.

5.2.1 Types of Memory Hierarchy

This Memory Hierarchy Design is divided into 2 main types:

- ★ **External Memory (or) Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e., peripheral storage devices which are accessible by the processor via an I/O Module.
- ★ **Internal Memory (or) Primary Memory:** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

5.2.2 Characteristics of Memory Hierarchy

One can infer these characteristics of a Memory Hierarchy Design from the figure given above:

1. Capacity

It refers to the total volume of data that a system's memory can store. The capacity increases moving from the top to the bottom in the Memory Hierarchy.

2. Access Time

It refers to the time interval present between the request for read/write and the data availability. The access time increases as we move from the top to the bottom in the Memory Hierarchy.

3. Performance

When a computer system was designed earlier without the Memory Hierarchy Design, the gap in speed increased between the given CPU registers and the Main Memory due to a large difference in the system's access time. It ultimately resulted in the system's lower performance, and thus, enhancement was required. Such a kind of enhancement was introduced in the form of Memory Hierarchy Design, and because of this, the system's performance increased. One of the primary ways to increase the performance of a system is minimising how much a memory hierarchy has to be done to manipulate data.

4. Cost per bit

The cost per bit increases as one moves from the bottom to the top in the Memory Hierarchy, i.e. External Memory is cheaper than Internal Memory.

5.2.3 Memory Hierarchy Design

The Computer memory hierarchy looks like a pyramid structure which is used to describe the differences among **memory types**. It separates the computer storage based on hierarchy.

Level 0: CPU registers

Level 1: Cache memory

Level 2: Main memory (or) primary memory

Level 3: Magnetic disks (or) secondary memory

Level 4: Optical disks

Level 5: Magnetic types (or) tertiary Memory

Memory Hierarchy Design

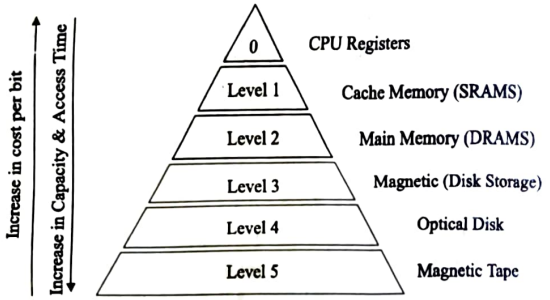


Fig. 5.1 Memory Hierarchy

1. Registers

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

2. Cache Memory

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

3. Main Memory

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

Types of Main Memory

(i) Static RAM

Static RAM stores the binary information in flip flops and information remains valid until power is supplied. Static RAM has a faster access time and is used in implementing cache memory.

(ii) Dynamic RAM

It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

4. Secondary Storage

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

5. Magnetic Disk:

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

6. Magnetic Tape:

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

Table 5.1: The memory levels in terms of size, access time, and bandwidth

Level	Register	Cache	Primary memory	Secondary memory
Bandwidth	4k to 32k MB/sec	800 to 5k MB/sec	400 to 2k MB/sec	4 to 32 MB/sec
Size	Less than 1KB	Less than 4MB	Less than 2 GB	Greater than 2 GB
Access time	2 to 5nsec	3 to 10 nsec	80 to 400 nsec	5ms
Managed by	Compiler	Hardware	Operating system	OS (or) user

5.2.4 Advantages of Memory Hierarchy

The need for a memory hierarchy includes the following.

- ★ Memory distributing is simple and economical
- ★ Removes external destruction
- ★ Data can be spread all over
- ★ Permits demand paging & pre-paging
- ★ Swapping will be more proficient

5.3 REGISTERS

A Register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing binary information of n-bits.

A register consists of a group of flip-flops and gates. The flip-flops hold the binary information and gates control when and how new information is transferred into a register. Various types of registers are available commercially. The simplest register is one that consists of only flip-flops with no external gates.

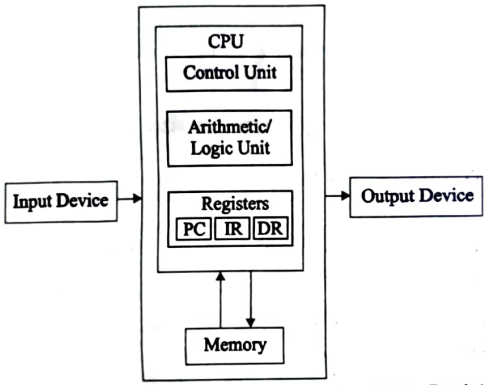


Fig. 5.2 Diagram of CPU that includes CU, ALU, Registers, and its relation to other components within embedded system

These days registers are also implemented as a register file.

Tiny, ultra-fast storage locations inside the CPU used to hold operands, addresses, and results of the current instructions.

A register is composed of multiple flip-flops, which are electronic circuits capable of storing a single bit of information, which is represented through binary data - either a 0 (or) a 1. By combining multiple flip-flops, registers can store larger binary values, such as bytes (or) words.

Registers also contain control logic circuitry, which allows it to coordinate the flow of data and instructions within the CPU. This can include operations such as decoding control signals, performing data manipulation like data loading, storing (or) arithmetic operations, and using mulitplexers to route data to a specific location within the register.

Together, flip-flops and control logic work in partnership within registers. Flip-flops provide the storage capacity, while control logic facilitates the coordination of data transfer, manipulation, and synchronization with other components of the CPU. This enables registers to store and process data efficiently during the execution of instructions.

The ALU is a fundamental component of the CPU responsible for performing arithmetic and logical operations. It can execute operations like addition, subtraction, AND, OR, and others. The ALU takes input from registers, performs the desired operation, and stores the result back into a register.

The CU directs and coordinates the operations of various components within the CPU. It interprets instructions and generates control signals to manage the flow of data between registers, the ALU, memory, and input/output devices.

The interaction between registers, ALU, and CU can be summarized in the following steps:

1. The CU fetches an instruction from memory and places it into the instruction register.
2. The CU decodes the instruction to determine the operation to be performed and identifies the registers involved.
3. The CU issues control signals to select the appropriate registers and routes the data to the ALU.

4. The ALU performs the arithmetic (or) logical operation on the data from the selected registers.
5. The result of the operation is stored back into a register, based on the CU's control signals.

5.3.1 Types of Registers

There are mainly two types of register,

1. General Purpose Register
2. Special Purpose Register

1. General Purpose Register

The General-purpose registers are mainly stored data. The general-purpose registers are,

(i) Accumulator
(ii) Data Register
(iii) Counter Register

2. Special Purpose Register

The Special purpose registers mainly to hold the instructions (or) lines (or) states of a program before execution. The special purpose registers are,

(i) Instruction Register
(ii) Program Counter

Accumulator (AC): Stores intermediate results of arithmetic and logic operations. It is one of the most frequently used registers in the CPU.

Program Counter (PC): Tracks the memory address of the next instruction to be executed. It increments automatically after each instruction unless a branch (or) jump occurs.

Instruction Register (IR): Temporarily holds the current instruction fetched from memory for execution.

Memory Address Register (MAR): Contains the address of the memory location to be accessed for reading (or) writing.

Memory Data Register (MDR): Holds the data to be written to (or) read from the memory location specified by the MAR.

General Purpose Registers (GPRs): Used for temporary storage of data during operations. They are labeled as R0, R1, R2, etc., and are accessible in assembly programming.

Stack Pointer (SP): Points to the top of the stack, which is used for managing function calls and local variables.

Flag Register: Also known as the status register, it contains flags that indicate the outcome of operations, such as Zero Flag, Carry Flag, Sign Flag, and Overflow Flag.

Input and Output Registers (INPR and OUTR): Handle data transfer between the CPU and input/output devices.

Temporary Register (TR): Stores intermediate data temporarily during processing.

5.3.2 Size of Register

The number and size of registers in a CPU are determined by the processor design and can have a significant impact on its performance and capabilities. Most modern computer processers include:

Registers come in various sizes, depending on the processor architecture:

- ★ **8-bit registers**: Handle 1 byte of data, suitable for simple operations.
- ★ **16-bit registers**: Store 2 bytes, used in older processors.
- ★ **32-bit registers**: Common in many processors, capable of handling larger data and more complex calculations.
- ★ **64-bit registers**: Found in modern processors, offering enhanced computational power and memory addressing.
- ★ Modern PCs today most often have 32-bit (or) 64-bit registers and are referred to as the 32-bit processors and 64-bit processors we often hear about. This indicates the size (or) width of the processor's registers and the amount of data the processor can handle in a single operation.
- ★ In some specialized processors (or) architectures, you may also find larger register sizes, such as 128-bit, 256-bit (or) even larger registers. These larger registers are often used for specific purposes like vector processing (or) cryptographic operations, where parallelism and large data sets are involved.

5.3.3 Purpose of Registers

Registers are used by computers for various purposes, including storing program instructions before they're executed (or) holding intermediate results from calculations so that their values can be retrieved later on if needed. They also help speed up processes by allowing processors to access frequently used values without having to retrieve them from main memory every time they need them.

5.4 CACHE MEMORY

Cache memory is a small, fast storage space within a computer. It holds duplicates of data from commonly accessed locations in the main memory. The CPU contains several separate caches that store both instructions and data.

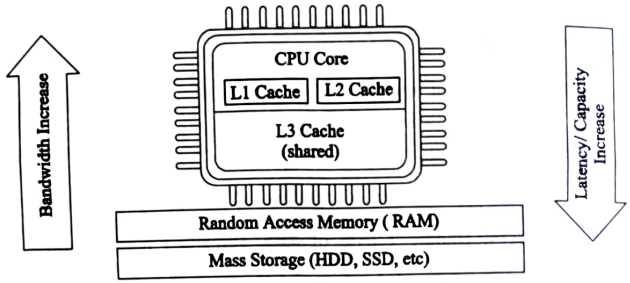


Fig. 5.3 Cache memory

- ★ The key function of cache memory is to reduce the average time needed to retrieve data from the main memory.
- ★ Cache's effectiveness relies on the principle of locality of reference, where recently accessed items (or) nearby items are more likely to be accessed again.
- ★ Cache memory stores data close to the CPU, which helps speed up processing. It's much faster than the main memory (RAM). When the CPU needs data, it checks the cache first. If the data is there, it's quickly accessed. If not, the CPU gets it from the slower main memory.

5.4.1 Working of Cache Memory

The working process of the cache,

1. **Fast but Small**: Cache is way faster than RAM but can only hold a small amount of data because it's limited in size.
2. **Checking the Cache First**: When the CPU needs data, it looks in the cache first because it's so quick. If the data is there (called a cache hit), the CPU grabs it and gets to work.
3. **Data Isn't in Cache**: If the data isn't in the cache (called a **cache miss**), the CPU looks in the slower RAM, gets the data, and copies it to the cache for next time.
4. **Speeding Things Up**: By keeping frequently used data in the cache, the CPU spends less time waiting for data from RAM. This makes your computer run faster.

5.4.2 Types of Cache Memory

- ★ **L1 (or) Level 1 Cache:** It is the first level of cache memory that is present inside the processor. It is present in a small amount inside every core of the processor separately. The size of this memory ranges from 2KB to 64 KB.
- ★ **L2 (or) Level 2 Cache:** It is the second level of cache memory that may present inside (or) outside the CPU. If not present inside the core, It can be shared between two cores depending upon the architecture and is connected to a processor with the high-speed bus. The size of memory ranges from 256 KB to 512 KB.
- ★ **L3 (or) Level 3 Cache:** It is the third level of cache memory that is present outside the CPU and is shared by all the cores of the CPU. Some high processors may have this cache. This cache is used to increase the performance of the L2 and L1 cache. The size of this memory ranges from 1 MB to 8MB.

5.4.3 Features of Cache Memory

- ★ Cache memory acts as a high-speed bridge between the CPU and RAM, storing frequently used data close to the CPU.
- ★ This proximity allows the CPU to access data more quickly, reducing the need to fetch information from slower main memory.
- ★ By minimizing wait times, cache memory significantly enhances CPU efficiency and overall system performance.

1. **Speed:** Faster than the main memory (RAM), which helps the CPU retrieve data more quickly.
2. **Proximity:** Located very close to the CPU, often on the CPU chip itself, reducing data access time.
3. **Function:** Temporarily holds data and instructions that the CPU is likely to use again soon, minimizing the need to access the slower main memory.

5.4.4 Application of Cache Memory

Here are some of the applications of Cache Memory.

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

5.4.5 Advantages

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

5.4.6 Disadvantages

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

5.5 MAIN MEMORY

- ★ The main memory is the fundamental storage unit in a computer system. It is associatively large and quick memory and saves programs and information during computer operations. The technology that makes the main memory work is based on semiconductor integrated circuits.
- ★ It is used to store computer data and instructions needed during the processing of data and output the results.

The memory unit that communicates directly within the CPU, Cache memory is called main memory. It is fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, majority part consists of RAM.

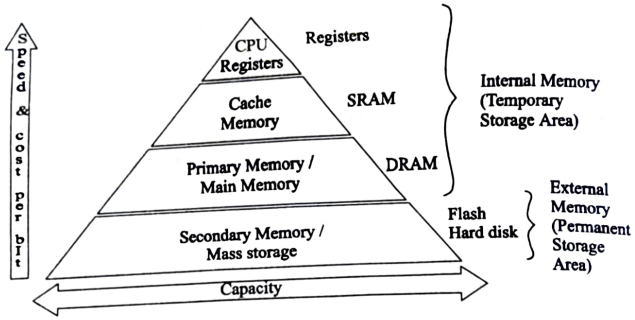


Fig. 5.4 Main memory

5.5.1 Types of Main memory

RAM technology is divided into two technologies: dynamic and static.

1. Dynamic RAM

- ★ A dynamic RAM (DRAM) is made with cells that store data as charge on capacitors.
- ★ The presence (or) absence of charge in a capacitor is interpreted as a bi nary 1 (or) 0. Because capacitors have a natural tendency to discharge, dynamic RAMs require periodic charge refreshing to maintain data storage.
- ★ The term dynamic refers to this tendency of the stored charge to leak away, even with power continuously applied.
- ★ Figure 5.5a is a typical DRAM structure for an individual cell that stores 1 bit. The address line is activated when the bit value from this cell is to be read (or) written.
- ★ The transistor acts as a switch that is closed (allowing current to flow) if a volt age is applied to the address line and open (no current flows) if no voltage is present on the address line.
- ★ For the write operation, a voltage signal is applied to the bit line; a high voltage represents 1, and a low voltage represents 0.

- ★ A signal is then applied to the address line, allowing a charge to be transferred to the capacitor.
- ★ For the read operation, when the address line is selected, the transistor turns on and the charge stored on the capacitor is fed out onto a bit line and to a sense amplifier.
- ★ The sense amplifier compares the capacitor voltage to a reference value and determines if the cell contains a logic 1 (or) a logic 0.
- ★ The readout from the cell discharges the capacitor, which must be restored to complete the operation.
- ★ Although the DRAM cell is used to store a single bit (0 (or) 1), it is essentially an analog device.
- ★ The capacitor can store any charge value within a range; a threshold value determines whether the charge is interpreted as 1 (or) 0.

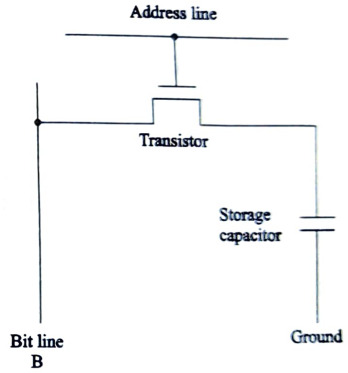


Fig. 5.5 Dynamic RAM (DRAM) cell

(i) Function of DRAM

The function of DRAM is that it is used for programming code by a computer processor to function. It is used in our PCs (Personal Computers).

(ii) Characteristics of DRAM

- ★ DRAM is used as the Main Memory inside the computer.
- ★ DRAM is known to be a fast memory but not as fast as SRAM.
- ★ DRAM is cheaper as compared to SRAM.
- ★ DRAM has a higher density (number of memory cells per unit area)
- ★ The power consumption by DRAM is more

2. Static RAM

- ★ In contrast, a static RAM (SRAM) is a digital device that uses the same logic elements used in the processor.
- ★ In a SRAM, binary values are stored using traditional flip-flop logic-gate configurations (see Chapter 20 for a description of flip-flops).
- ★ A static RAM will hold its data as long as power is supplied to it.
- ★ Figure 5.6 is a typical SRAM structure for an individual cell. Four transistors (T_1, T_2, T_3, and T_4) are cross connected in an arrangement that produces a stable logic state.
- ★ In logic state 1, point C_1 is high and point C_2 is low;in this state, T_1 and T_4 are off and T_2 and T_3 are on.1 In logic state 0, point C_1 is low and point C_2 is high; in this state, T_1 and T_4 are on and T_2 and T_3 are off.
- ★ Both states are stable as long as the direct current (dc) voltage is applied. Unlike the DRAM, no refresh is needed to retain data.
- ★ As in the DRAM, the SRAM address line is used to open (or) close a switch.
- ★ The address line controls two transistors (T_5 and T_6).When a signal is applied to this line, the two transistors are switched on, allowing a read (or) write operation.
- ★ For a write operation, the desired bit value is applied to line B, while its complement is applied to line.

- ★ This forces the four transistors (T_1, T_2, T_3, and T_4) into the proper state. For a read operation, the bit value is read from line B.

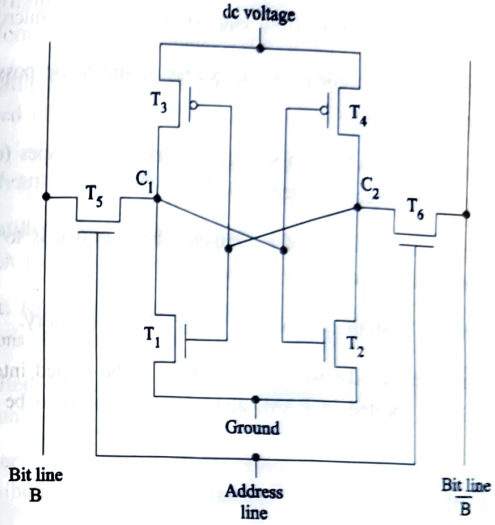


Fig. 5.6 Static RAM (SRAM) cell

(i) Function of SRAM

The function of SRAM is that it provides a direct interface with the Central Processing Unit at higher speeds.

(ii) Characteristics of SRAM

- ★ SRAM is used as the Cache memory inside the computer.
- ★ SRAM is known to be the fastest among all memories.
- ★ SRAM is costlier.
- ★ SRAM has a lower density (number of memory cells per unit area).
- ★ The power consumption of SRAM is less but when it is operated at higher frequencies, the power consumption of SRAM is compatible with DRAM.

5.6 RAM

- ★ The term **Random Access Memory** (or) RAM is typically used to refer to memory that is easily read from and written to by the microprocessor.
- ★ For a memory to be called random access, it should be possible to access any address at any time.
- ★ This differentiates RAM from storage devices such as tapes (or) hard drives where the data is accessed sequentially.
- ★ RAM is the main memory of a computer. Its objective is to store data and applications that are currently in use.
- ★ The **operating system** controls the usage of this memory.
- ★ It gives instructions like when the items are to be loaded into RAM, where they are to be located in RAM, and when they need to be removed from RAM.

5.6.1 Features of RAM

- ★ RAM is volatile, meaning that the data is erased when the device is turned off.
- ★ It is referred to as the primary memory of the computer, as it directly supports the CPU during operation.
- ★ RAM is relatively expensive because it allows for fast, direct access to data.
- ★ As the fastest type of memory, RAM serves as internal memory within the computer, enabling quick data retrieval.
- ★ The overall speed of the computer is greatly influenced by the amount of RAM. With less RAM, the computer takes longer to load and may slow down significantly.

5.6.2 Advantages of RAM

- ★ **Speed:** RAM is faster than other types of storage like ROM, hard drives (or) SSDs, allowing for quick access to data and smooth performance of applications.

- ★ **Multitasking:** More RAM allows a computer to handle multiple applications simultaneously without slowing down.
- ★ **Flexibility:** RAM can be easily upgraded, enhancing a computer's performance and extending its usability.
- ★ **Volatile Storage:** RAM automatically clears its data when the computer is turned off, reducing the risk of unwanted data accumulation.

5.6.3 Disadvantages of RAM

- ★ **Volatility:** Data stored in RAM is lost when the computer is turned off, which means important data must be saved to permanent storage.
- ★ **Cost:** RAM can be more expensive per gigabyte compared to other storage options like hard drives (or) SSDs.
- ★ **Limited Storage:** RAM has a limited capacity, so it cannot store large amounts of data permanently.
- ★ **Power Consumption:** RAM requires continuous power to retain data, contributing to the overall power consumption of the device.
- ★ **Physical Space:** Increasing RAM requires physical space in the computer, which might be limited to smaller devices like laptops and tablets.

5.6.4 CACHE vs RAM

Cache Memory	RAM (Random Access Memory),
Very close to CPU	Close to CPU, connected via memory bus
Stores frequently accessed data for faster CPU access	Main working memory for currently running programs
Extremely fast (nanoseconds)	Fast, but slower than cache (tens of nanoseconds)
Small (KB to few MB)	Large (GB to TB)
SRAM (faster, more expensive)	DRAM (slower, cost-effective)
Multi-level (L1, L2, L3)	Single level

Cache Memory	RAM (Random Access Memory),
Acts as a buffer between CPU and RAM	Stores data/instructions currently being processed
High	Lower
Limited	Larger, supports multiple applications

5.6.5 Difference between SRAM and DRAM

The below table lists some of the differences between SRAM and DRAM.

SRAM	DRAM
SRAM stands for Static Random Access Memory.	DRAM stands for Dynamic Random Access Memory.
Uses a flip-flop circuit to store data	Uses a capacitor and a transistor to store data
SRAM has a lower access time, so it is faster compared to DRAM.	DRAM has a higher access time, so it is slower than SRAM.
SRAM has long data life.	DRAM has short data life.
SRAM has a storage capacity of 1 MB to 16 MB in most cases.	DRAM, which is often found in tablets and smartphones, has a capacity of 1 GB to 2 GB
SRAM is costlier than DRAM.	DRAM costs less compared to SRAM.
SRAM provides faster speed of data read/write.	DRAM provides slower speed of data read/write.
SRAM requires a constant power supply, which means this type of memory consumes more power.	DRAM offers reduced power consumption due to the fact that the information is stored in the capacitor.
Good choice for applications that may be exposed to extreme temperatures.	Not suitable for such applications.
Due to complex internal circuitry, less storage is available compared to the same physical size of a DRAM memory chip.	Due to the small internal circuitry in the one-bit memory cell of DRAM, a large storage capacity is available.
SRAM has low packaging capacity.	DRAM has a high packaging density.

SRAM	DRAM
SRAM is used in cache memories.	DRAM is used in main memories.
SRAM does not require refresh time.	DRAM requires periodic refresh time.
SRAMs are used as cache memory in computer and other computing devices.	DRAMs are used as main memory in computer systems.

5.7 ROM

ROM stands for Read-Only Memory. It is a non-volatile memory was used to operate the system. As its name refers to read-only memory, we can only read the stored programs and data.

- ★ Information stored in ROM is permanent.
- ★ Information and programs are stored on ROM in binary format (0s and 1s).
- ★ It is used in the start-up process of the computer.

5.7.1 Internal Structure of ROM

- ★ The internal structure of a ROM consists of a decoder and a grid of OR gates, which together store and retrieve data.

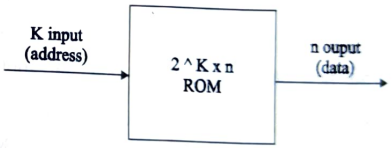


Fig. 5.7

- ★ The decoder takes a binary address input and activates a specific output line corresponding to that address.
- ★ These lines are connected to the inputs of the OR gates, which are programmed (or pre-wired) to produce the correct bit pattern for that address as the output.

The internal structure of ROM has two basic components:

- This information may have been included in the original interrupt signal (or) the processor may have to issue a request to the device that issued the interrupt to get a response that contains the needed information.

6. At this point, the program counter and PSW relating to the interrupted program have been saved on the system stack. However, there is other information that is considered part of the "state" of the executing program.

7. In particular, the contents of the processor registers need to be saved, because these registers may be used by the interrupt handler. So, all of these values, plus any other state information, need to be saved.

- Typically, the interrupt handler will begin by saving the contents of all registers on the stack. Fig. 6.7 a shows a simple example.

- In this case, a user program is interrupted after the instruction at location N.

- The contents of all of the registers plus the address of the next instruction (N + 1) are pushed on to the stack.

- The stack pointer is updated to point to the new top of stack, and the program counter is updated to point to the beginning of the interrupt service routine.

8. The interrupt handler next processes the interrupt. This includes an examination of status information relating to the I/O operation or other event that caused an interrupt. It may also involve sending additional commands or acknowledgments to the I/O device.

9. When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers (e.g., see Fig. 6.7b).

10. The final act is to restore the PSW and program counter values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.

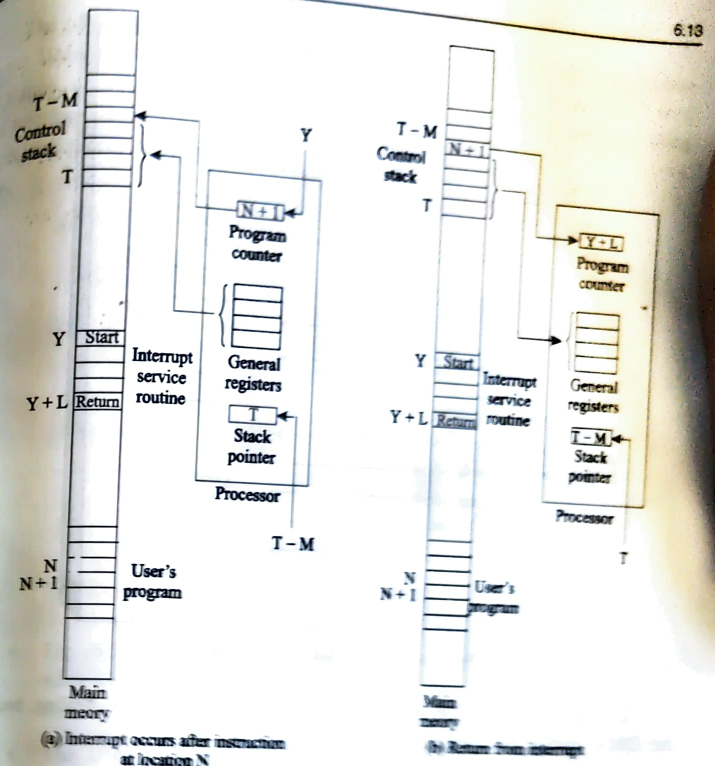


Fig. 6.7 Changes in Memory and Registers for an Interrupt

6.5 DIRECT MEMORY ACCESS (DMA)

DMA involves an additional module on the system bus. The DMA module (Fig. 6.8) is capable of mimicking the processor and, indeed, of taking over control of the system from the processor. It needs to do this to transfer data to and from memory over the system bus. For this purpose, the DMA module must use the bus only when the processor does not need it (or) it must force the processor to suspend operation temporarily.

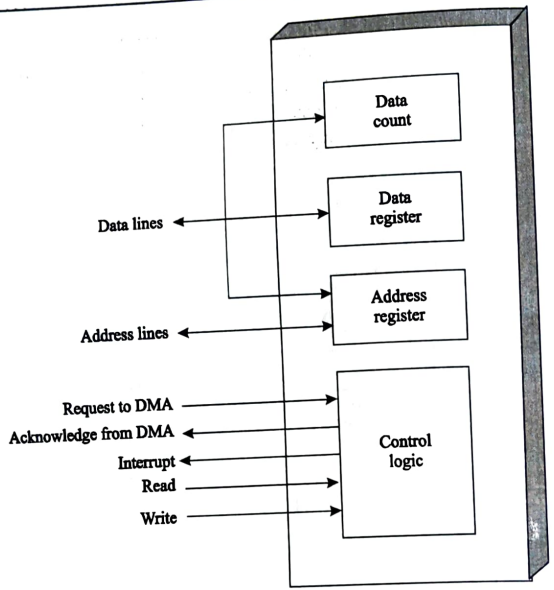


Fig. 6.8 Typical DMA Block Diagram

The latter technique is more common and is referred to as cycle stealing, because the DMA module in effect steals a bus cycle.

When the processor wishes to read (or) write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information:

- ★ Whether a read (or) write is requested, using the read (or) write control line between the processor and the DMA module.
- ★ The address of the I/O device involved, communicated on the data lines
- ★ The starting location in memory to read from (or) write to, communicated on the data lines and stored by the DMA module in its address register
- ★ The number of words to be read (or) written, again communicated via the data lines and stored in the data count register.

The processor then continues with other work. It has delegated this I/O operation to the DMA module. The DMA module transfers the entire block of data, one word at a time, directly to (or) from memory, without going through the processor. When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer (Fig. 6.9).

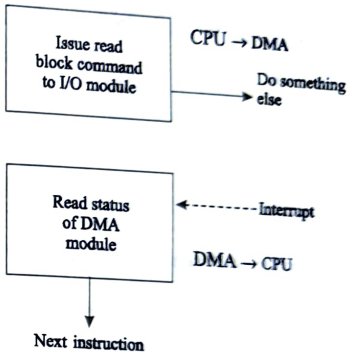


Fig. 6.9 Direct Memory Access

- ★ Fig. 6.10 shows where in the instruction cycle the processor may be suspended. In each case, the processor is suspended just before it needs to use the bus.
- ★ The DMA module then transfers one word and returns control to the processor.
- ★ Note that this is not an interrupt; the processor does not save a con text and do something else. Rather, the processor pauses for one bus cycle.
- ★ The overall effect is to cause the processor to execute more slowly.
- ★ Nevertheless, for a multiple-word I/O transfer, DMA is far more efficient than interrupt-driven (or) programmed I/O.

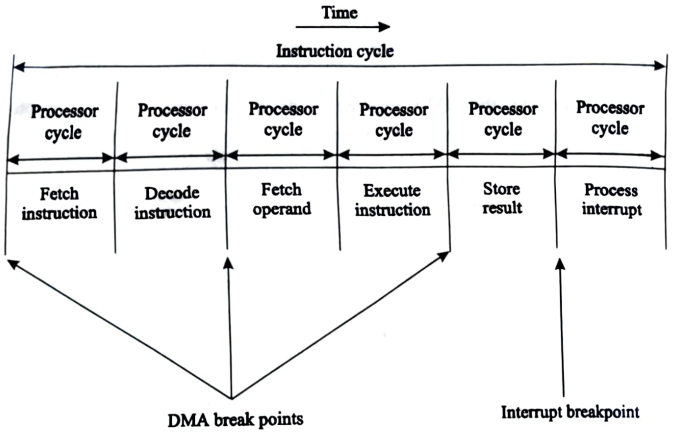
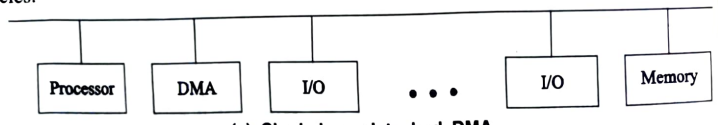
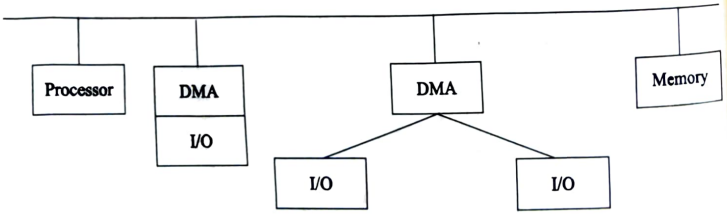


Fig. 6.10 DMA and Interrupt Breakpoints during an Instruction Cycle

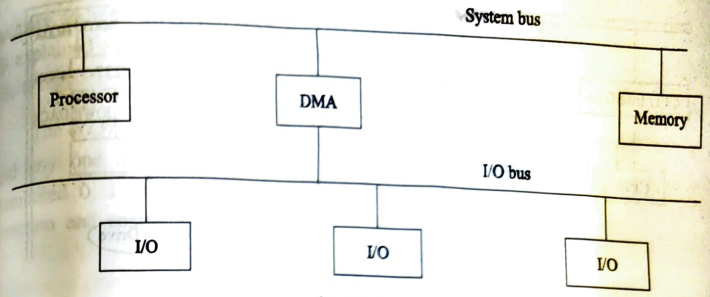
The DMA mechanism can be configured in a variety of ways. Some possibilities are shown in Fig. 6.11. In the first example, all modules share the same system bus. The DMA module, acting as a surrogate processor, uses programmed I/O to exchange data between memory and an I/O module through the DMA module. This configuration, while it may be inexpensive, is clearly inefficient. As with processor-controlled programmed I/O, each transfer of a word consumes two bus cycles.



(a) Single-bus, detached DMA



(b) Single-bus, integrated DMA-I/O



(c) I/O bus

Fig. 6.11 Alternative DMA Configurations

The number of required bus cycles can be cut substantially by integrating the DMA and I/O functions. As Fig. 6.11 (b) indicates, this means that there is a path between the DMA module and one (or) more I/O modules that does not include the system bus. The DMA logic may actually be a part of an I/O module (or) it may be a separate module that controls one (or) more I/O modules. This concept can be taken one step further by connecting I/O modules to the DMA module using an I/O bus (Fig. 6.11c). This reduces the number of I/O interfaces in the DMA module to one and provides for an easily expandable configuration. In both of these cases (Figures 6.11 (b) and (c)), the system bus that the DMA module shares with the processor and memory is used by the DMA module only to exchange data with memory. The exchange of data between the DMA and I/O modules takes place off the system bus.

6.5.1 *Intel 8237A DMA Controller*

The Intel 8237A DMA controller interfaces to the 80x86 family of processors and to DRAM memory to provide a DMA capability. Fig. 6.12 indicates the location of the DMA module. When the DMA module needs to use the system buses (data, address, and control) to transfer data, it sends a signal called HOLD to the processor. The processor responds with the HLDA (hold acknowledge) signal, indicating thatthe DMA module can use the buses.

If the DMA module is to transfer a block of data from memory to disk, it will do the following:

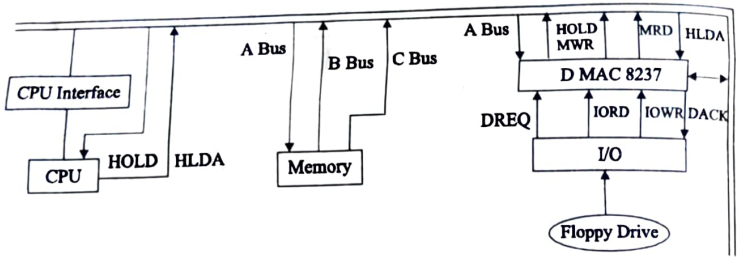


Fig. 6.12

1. The peripheral device (such as the disk controller) will request the service of DMA by pulling DREQ (DMA request) high.
2. The DMA will put a high on its HRQ (hold request), signaling the CPU through its HOLD pin that it needs to use the buses.
3. The CPU will finish the present bus cycle (not necessarily the present instruction) and respond to the DMA request by putting high on its HDLA (hold ac knowledge), thus telling the 8237 DMA that it can go ahead and use the buses to perform its task. HOLD must remain active high as long as DMA is per forming its task.
4. DMA will activate DACK (DMA acknowledge),which tells the peripheral de vice that it will start to transfer the data.
5. DMA starts to transfer the data from memory to peripheral by putting the ad dress of the first byte of the block on the address bus and activating MEMR, thereby reading the byte from memory into the data bus; it then activates IOW to write it to the peripheral. Then DMA decrements the counter and increments the address pointer and repeats this process until the count reaches zero and the task is finished.
6. After the DMA has finished its job it will deactivate HRQ, signaling the CPU that it can regain control over its buses.

While the DMA is using the buses to transfer data, the processor is idle. Similarly, when the processor is using the bus, the DMA is idle. The 8237 DMA is known as a fly-by DMA controller. This means that the data being moved from one location to another does not pass through the DMA chip and is not stored in the DMA chip. Therefore, the DMA can only transfer data between an I/O port and a

memory address, but not between two I/O ports (or) two memory locations. However, as explained subsequently, the DMA chip can perform a memory-to-memory transfer via a register.

The 8237 contains four DMA channels that can be programmed independently, and any one of the channels may be active at any moment. These channels are numbered 0, 1, 2, and 3. The 8237 has a set of five control/command registers to program and control DMA operation over one of its channels (Table 6.1):

Table 6.1 Intel 8237A Registers

Bit	Command	Status	Mode	Single Mask	All Mask
D0	Memory-to-memory E/D	Channel 0 has reached TC	Channel select	Select channel mask bit	Clear/set channel 0 mask bit
D1	Channel 0 address hold E/D	Channel 1 has reached TC			Clear/set channel 1 mask bit
D2	Controller E/D	Channel 2 has reached TC	Verify/write/read transfer	Clear/set mask bit	Clear/set channel 2 mask bit
D3	Normal/compressed timing	Channel 3 has reached TC		Not used	Clear/set channel 3 mask bit
D4	Fixed/rotating priority	Channel 0 request	Auto-initialization E/D		Not used
D5	Late/extended write selection	Channel 0 request	Address increment/decrement select		
D6	DREQ sense active high/low	Channel 0 request			
D7	DACK sense active high/low	Channel 0 request	Demand/single /block/cascade mode select		

E/D = enable/disable

TC = terminal count

★ **Command:** The processor loads this register to control the operation of the DMA.D0 enables a memory-to-memory transfer, in which channel 0 is used to transfer a byte into an 8237 temporary register and channel 1 is used to transfer the byte from the register to memory. When memory-to-memory is enabled, D1 can be used to disable increment/decrement on channel 0 so that a fixed value can be written into a block of memory. D2 enables (or) disables DMA.

★ ***Status:*** *The processor reads this register to determine DMA status. Bits D0-D3 are used to indicate if channels 0-3 have reached their TC (terminal count). Bits D4-D7 are used by the processor to determine if any channel has a DMA request pending.*

- **Mode:** The processor sets this register to determine the mode of operation of the DMA. Bits D0 and D1 are used to select a channel. The other bits select various operation modes for the selected channel. Bits D2 and D3 determine if the transfer is a from an I/O device to memory (write) (or) from memory to I/O (read) (or) a verify operation. If D4 is set, then the memory address register and the count register are reloaded with their original values at the end of a DMA data transfer. Bits D6 and D7 determine the way in which the 8237 is used. In single mode, a single byte of data is transferred. Block and demand modes are used for a block transfer, with the demand mode allowing for premature ending of the transfer. Cascade mode allows multiple 8237s to be cascaded to expand the number of channels to more than 4.

- **Single Mask:** The processor sets this register. Bits D0 and D1 select the channel. Bit D2 clears (or) sets the mask bit for that channel. It is through this register that the DREQ input of a specific channel can be masked (disabled) (or) unmasked (enabled). While the command register can be used to disable the whole DMA chip, the single mask register allows the programmer to disable (or) enable a specific channel.

- **All Mask:** This register is similar to the single mask register except that all four channels can be masked (or) unmasked with one write operation.

6.6 I/O DEVICES

In computing, input/output (or) I/O, refers to the communication between an information processing system (computer), and the outside world. Inputs are the signals (or) data received by the system, and outputs are the signals (or) data sent from it. I/O devices are used by a person (or other system) to communicate with a computer.

An **input/output (I/O) device** is any hardware that enables a human user (or) another system to communicate with a computer. As the name suggests, these devices can both **receive data (input)** from the user (or) another source and **deliver data**

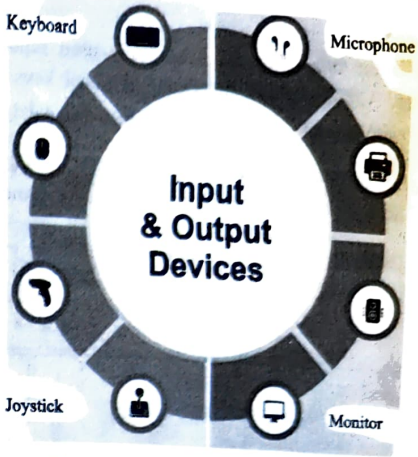


Fig. 6.13 Input and Output Devices

(output) from the computer. Essentially, an I/O device bridges the computer and external entities, facilitating seamless data exchange.

6.6.1 Input Devices

Input devices are the devices that are used to send signals to the computer for performing tasks. The receiver at the end is the CPU (Central Processing Unit), which works to send signals to the output devices. Some of the classifications of Input devices are:

(a) Keyboard Devices

(b) Pointing Devices

(c) Composite Devices

(d) Game Controller

(e) Visual Devices

(f) Audio Input Devices

Some of the input devices are described below.