GeeksforGeeks

GATE फर्स्ट

CSE

COA

SHORT NOTES

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

STAR MENTOR CS/DA

KHALEEL SIR
ALGORITHM & OS
29 YEARS OF TEACHING EXPERIENCE

CHANDAN SIR
DIGITAL LOGIC
GATE AIR 23 & 26 / EX-ISRO

SATISH SIR
DISCRETE MATHEMATICS
BE in IT from MUMBAI UNIVERSITY

MALLESHAM SIR
M.TECH FROM IIT BOMBAY
AIR – 114, 119, 210 in GATE
(CRACKED GATE 8 TIMES)
14+ YEARS EXPERIENCE

VIJAY SIR
DBMS & COA
M. TECH FROM NIT
14+ YEARS EXPERIENCE

PARTH SIR
DA
IIIT BANGALORE ALUMNUS
FORMER ASSISTANT PROFESSOR

SAKSHI MA'AM
ENGINEERING MATHEMATICS
IIT ROORKEE ALUMNUS

SHAILENDER SIR
C PROGRAMMING & DATA STRUCTURE
M.TECH in Computer Science
15+ YEARS EXPERIENCE

AVINASH SIR
APTITUDE
10+ YEARS OF TEACHING EXPERIENCE

AJAY SIR
PH.D. IN COMPUTER SCIENCE
12+ YEARS EXPERIENCE

1. Introduction to COA

1.1. Types of computers

- Embedded computers
- Personal computers: desktop computers, workstation computers, portable computers etc.
- Servers & Enterprise systems
- Supercomputers

1.2. Components of Computer

1.2.1. CPU (Central Processing Unit)

1.2.1.1. ALU (Arithmetic Logical Unit)

ALU performs the required micro-operations for executing the instructions.

1.2.1.2. Control Unit

The control unit supervises
the transfer of information among the registers and instructs the ALU as to which operation to
perform.

1.2.1.3. Registers

- Registers are small, high-speed storage locations within the CPU used for temporary storage, control, and data manipulation during program execution.
- They hold binary information and are crucial for efficient CPU operation.

1.2.1.3.1. Types of CPU Registers

01. PC

- The processor keeps track of the address of the memory location containing the next instruction to be fetched using the program counter.
- After fetching an instruction, the contents of the PC are updated to point to the next instruction in the sequence.

02. IR (Instruction Register)

Decodes fetched instruction with predefined format, which is:

1. Fetch the contents of the memory location pointed to by the PC, and load into IR. i.e. IR ← [(PC)]
2. Assuming that the memory is byte addressable & one word is 32 bit (4B), increment the contents of the PC by 4, so PC ← [PC] +4
3. Decode the inst^{n} to understand the operation & generate the control signals necessary to carry out the operation.

03. Accumulator

Temporary storage location for arithmetic & logical operations

04. MAR

Works with the memory bus to fetch/store data at a specific address.

05. MDR

a. Temporarily holds data being transferred to/from memory. Contents of MBR are directly connected to the data bus.
b. Acts as a buffer between CPU and memory.

1.2.2. Memory

- Main/Primary Memory
- Secondary Memory

1.2.3. I/O

1.2.3.1. Input Unit

The process of receiving data from an external source (like a user typing on a keyboard or a sensor reading data) and making it available to the computer's internal components for processing.

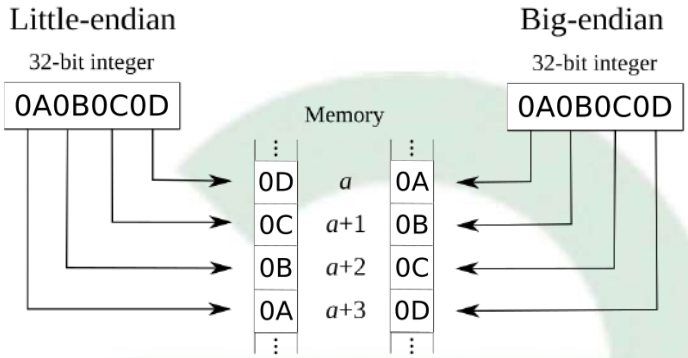
1.2.3.2. Output Unit

The process of sending data from the computer's internal components to an external device or system (like displaying

information on a monitor, printing to a printer, or sending data over a network).

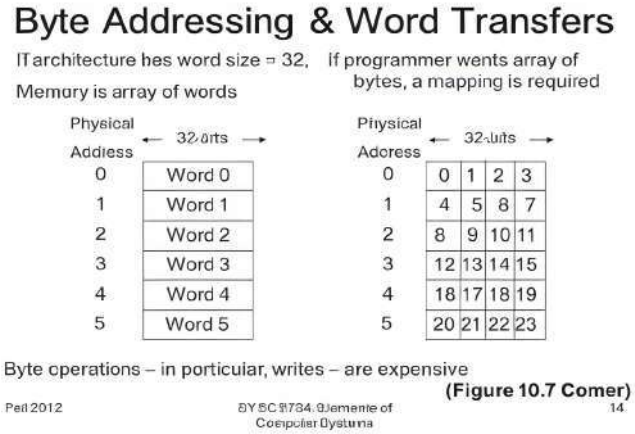
1.3. Big-endian & Little-endian Assignments

- The name big-endian is used when lower byte addresses are used for the more significant bytes.
- The name little-endian is used for where the lower byte addresses are used for the less significant bytes of the word.

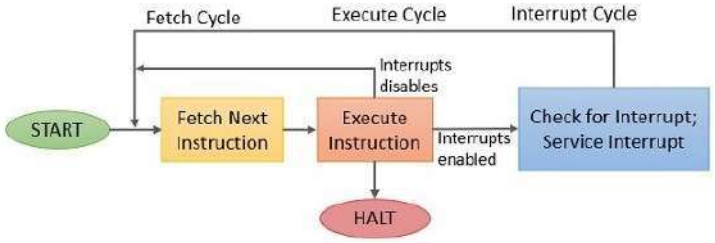


1.4. Memory Addressing

If the memory is byte addressable, then byte locations have addresses 0, 1, 2... and if the memory is word addressable and the length of each word is 32 bits, then successive words are located at addresses 0,4,8,12,...., with each word consisting of 2 bytes. (1B = 8 bits)



1.5. Instruction Cycle



Instruction Cycle with Interrupts

1.5.1. Instruction Fetch (IF)

- PC → MAR: Program Counter (PC) holds address of next instruction; transferred to Memory Address Register (MAR).
- Read Memory: Control unit issues a read; memory returns the instruction into Memory Buffer Register (MBR).
- MBR → IR: Instruction Buffer Register (IR) loads the instruction.
- PC ← PC + 1: PC is incremented to point to the following instruction.

1.5.2. Instruction Decode (ID)

- IR → Control Unit: Opcode field is sent to the control unit for interpretation.
- Operand Fetch: If needed, source operand addresses are loaded into MAR; memory or register file is accessed, placing data into temporary registers.
- Register File Access: Control signals select appropriate registers; operands are read into internal CPU registers.

1.5.3. Execute (EX)

- ALU Operation: Arithmetic/logic unit performs the operation specified (e.g., add, subtract, logical AND/OR, shift).
- Address Calculation: For memory-reference instructions, effective address is computed here.

1.5.4. Memory Access (MEM)

Data Read/Write:

- Load: If it's a load instruction, MAR→memory→MBR→destination register.
- Store: If it's a store instruction, source register→MBR→memory at MAR.

1.5.5. Write-Back (WB)

- Result → Register: ALU or MBR result is written back into the register file or PC (for branches/jumps).
- Flags Update: Condition codes (zero, carry, overflow) are updated if needed.

Note:

- The fetch-execute cycle repeats to execute next instructions until a halt instruction is executed.
- Halt (HLT) instruction stops the execution of further instructions until an interrupt or reset signal is received.
- While halted, the CPU may perform minimal operations like memory refresh to maintain system integrity.

2. Machine Instructions & Addressing Modes

2.1. Instructions Format

A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction. The most common fields found in instruction formats are:

1. An operation code field (opcode) that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

2.2. Types of Instructions

2.2.1. Three-Address Instructions

Each address field specifies either a register or an operand. The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.

```
X = (A + B) * (C + D)
ADD   R1, A, B     R1←M[A] + M[B]
ADD   R2, C, D     R2←M[C] + M[D]
MUL   X, R1, R2    M[X]←R1*R2
```

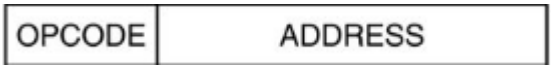
2.2.2. Two-Address Instructions

Each address field can specify either a register or a word. The program to evaluate X = (A + B) * (C + D) is as follows

```
MOV   R1, A     R1←M[A]
ADD   R1, B     R1←R1 + M[B]
MOV   R2, C     R2←M[C]
ADD   R2, D     R2←R2 + M[D]
MUL   R1,R2     R1←R1*R2
MOV   X, R1     M[X]←R1
```

Note: The MOV instruction moves or transfers the operands to and from memory and processor registers.

2.2.3. One-Address Instructions



One-address instructions use an implied accumulator (AC) register for all data manipulation. The program to evaluate X = (A + B) * (C + D) is

```
LOAD    A    AC ← M[A]
ADD     B    AC ← AC + M[B]
STORE   T    M[T] ← AC
LOAD    C    AC ← M[C]
ADD     D    AC ← AC + M[D]
MUL     T    AC ← AC * M[T]
STORE   X    M[X] ← AC
```

2.2.4. Zero-Address Instructions

Used in stack organised computers with PUSH & POP instructions. The following program shows how
X = (A + B) * (C + D) will be written

```
PUSH    A    TOS ← A
PUSH    B    TOS ← B
ADD          TOS ← (A + B)
PUSH    C    TOS ← C
PUSH    D    TOS ← D
ADD          TOS ← (C + D)
MUL          TOS ← (C + D) * (A + B)
POP     X    M[X] ← TOS
```

Note: To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into Reverse Polish notation.

2.2.5. RISC Instructions

It is restricted to LOAD & STORE instructions when interacting between memory & CPU. All other instructions like ADD, MUL are executed within the registers of the CPU without referring to memory. Following is a program to evaluate X = (A + B) * (C + D)

```
LOAD    R1, A        R1 ← M[A]
LOAD    R2, B        R2 ← M[B]
LOAD    R3, C        R3 ← M[C]
LOAD    R4, D        R4 ← M[D]
ADD     R1, R1, R2   R1 ← R1 + R2
ADD     R3, R3, R2   R3 ← R3 + R4
MUL     R1, R1, R3   R1 ← R1 * R3
STORE   X, R1        M[X] ← R1
```

2.3. Addressing Modes

The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.

Its types are:

- Implied Mode
- Register Indirect Mode
- Immediate Mode
- Autoincrement or Autodecrement Mode
- Direct Addressing Mode/Absolute Addressing Mode
- Relative Address Mode
- Indirect Address Mode
- Indexed Addressing Mode
- Register Mode
- Base Register Addressing Mode

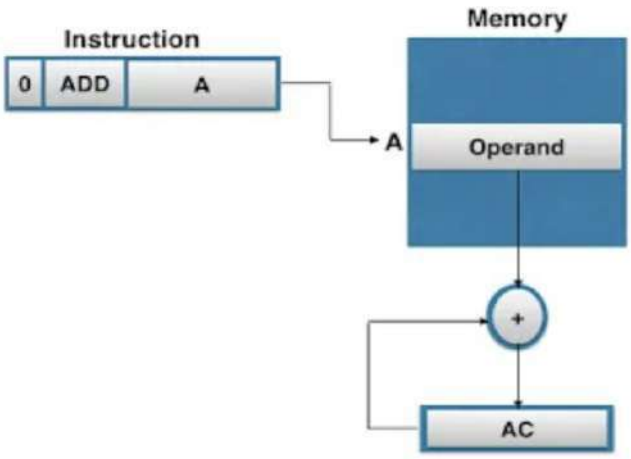
2.3.1. Implied Mode

In this mode the operands are specified implicitly in the definition of the instruction.
a. All register reference instructions that use an accumulator are implied-mode instructions.
b. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2.3.2. Immediate Mode

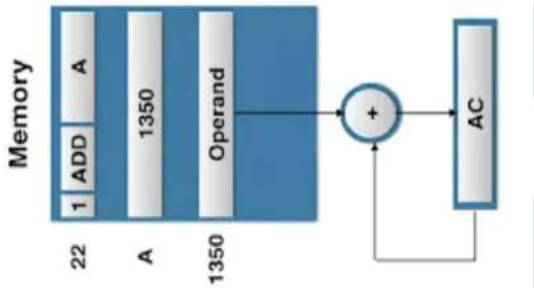
In this mode the operand is specified in the instruction itself. An immediate-mode instruction has an operand field rather than an address field.

2.3.3 Direct Address Mode/Absolute Address Mode



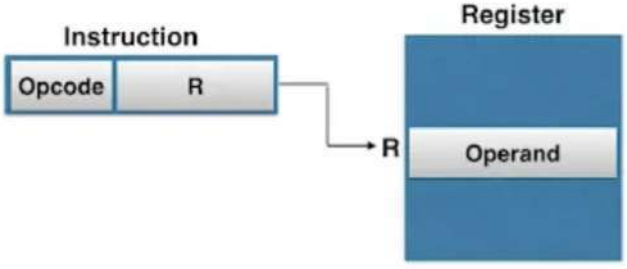
In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction. In a branch-type instruction the address field specifies the actual branch address.

2.3.4. Indirect Address Mode



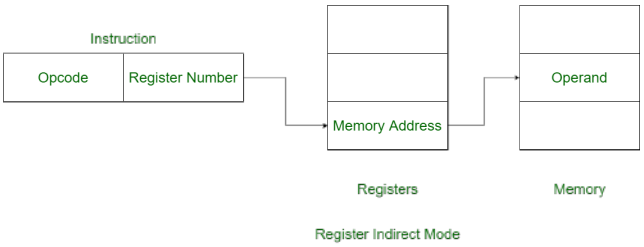
In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

2.3.5. Register Mode



In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

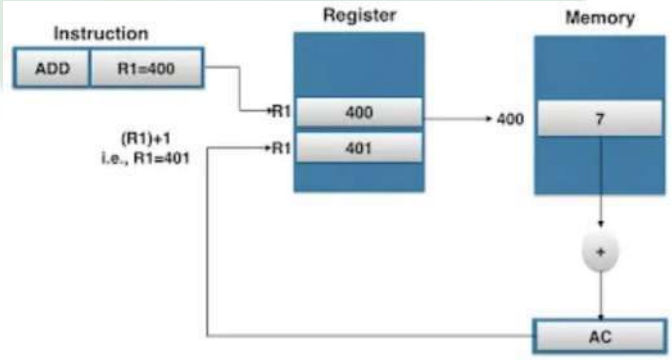
2.3.6. Register Indirect Mode



Register Indirect Mode

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than a memory address directly.

2.3.7. Autoincrement or Autodecrement Mode



This is similar to the register indirect mode except that the register is incremented or decremented after its value is used to access memory.

Note: The Effective Address (E.A.) is the memory location calculated based on the addressing mode specified in the instruction, i.e.
E.A. = address part of inst^n + content of CPU register

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

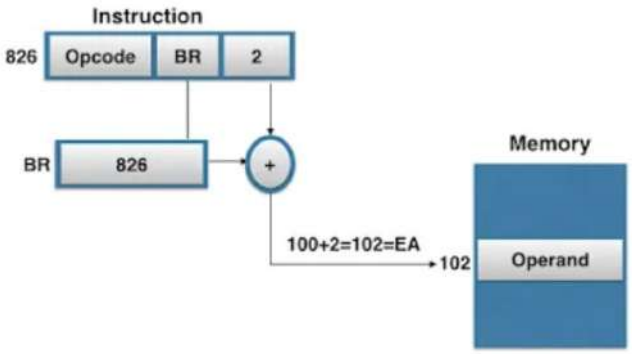
- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

2.3.8. Relative Address Mode

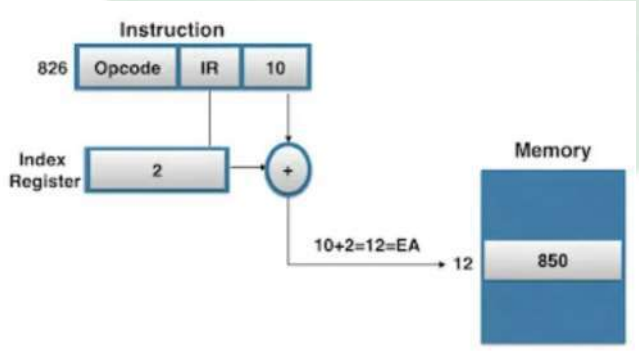


In this mode the content of the program counter is added to the address part to obtain E.A. The address part is a signed number (2's complement) which can be either positive or negative.

EA = Address Part (off set) + PC value

It results in a shorter address field since the relative address can be specified with a smaller number of bits compared to the entire memory address. It's generally used in Branch-Type instructions.

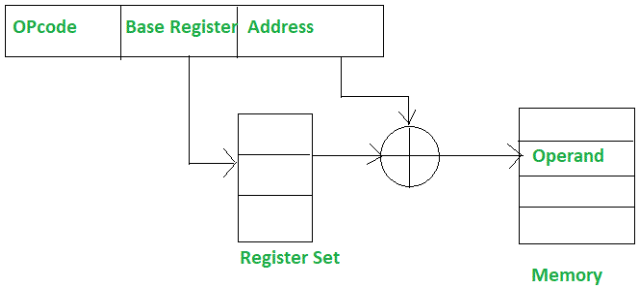
2.3.9. Indexed Addressing Mode



- In this mode the content of an index register is added to the address part to obtain E.A.
- The index register contains an index value.
- The address field of the instruction defines the beginning address of a data array in memory.

EA = Address Part (base address of data array) + Index register value (index value)

2.3.10. Base Register Addressing Mode



- In this mode the content of a base register is added to the address part of the instruction.
- The base register is assumed to hold the base address.
- The address field gives the displacement relative to this base address.

EA = Address Part (displacement/offset) + Base register value (Base address)

2.4. CISC vs RISC

CISC (Complex Instruction Set Computer)

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

RISC (Reduced Instruction Set Computer)

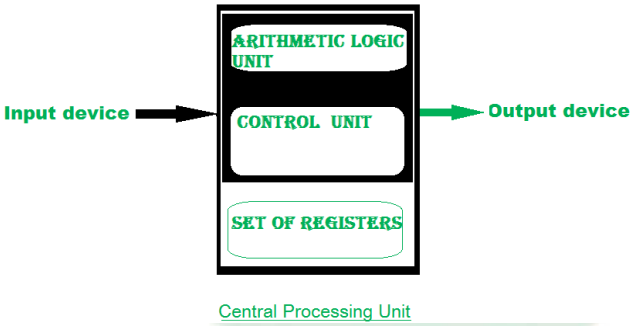
- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format

- Single-cycle instruction execution, CPI=1
- Hardwired rather than microprogrammed control
- A relatively large number of registers in the processor unit
- Use of overlapped register windows to speed-up procedure call and return.
- Efficient instruction pipeline, CPI=1
- Compiler support for efficient translation of high-level language programs into machine language programs.

3. ALU, Data-Path and Control Unit

3.1. ALU

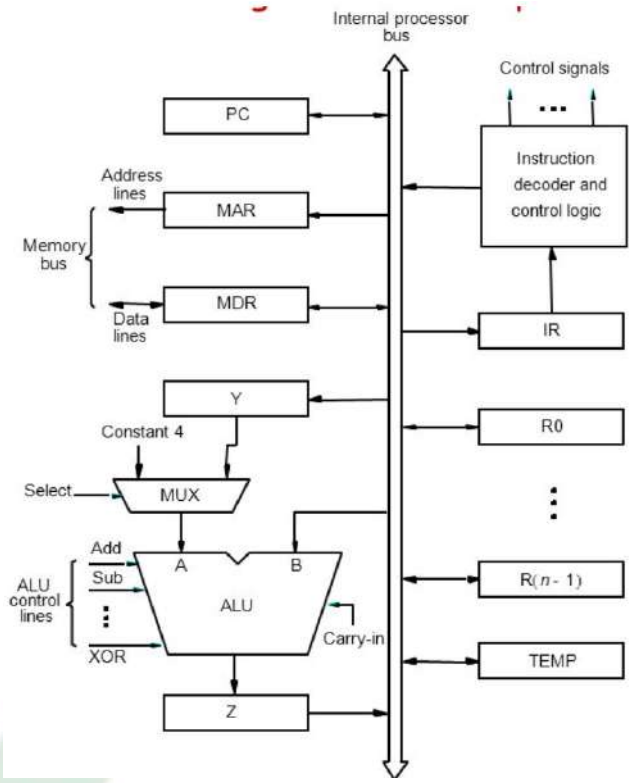
ALU is a digital circuit that provides arithmetic and logic operations. It is the fundamental building block of the CPU of a computer.



Central Processing Unit

3.2. Datapath

CPU has 2 sections: Data Section(Data Path)+ Control Section(Control Path)
Data Path = Registers + ALU + Interconnecting bus



The data and address lines of the external memory bus are shown above connected to the internal processor bus via the **memory data register** (MDR) and the **memory address register** (MAR) respectively.

3.2.1. Types of Datapath

a. One bus datapath

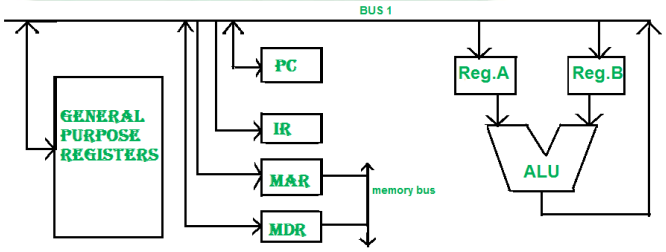


fig. One Bus Organisation

- Structure: A single internal bus connects all registers, ALU inputs, and memory data lines.
- Operation:
  - Only one data transfer or ALU operation can occur at any given clock cycle.
  - Uses multiplexers to select which register drives the bus and where the bus feeds.

- Advantages:
  - Minimal hardware (only one bus, fewer multiplexers).
  - Low cost.
- Disadvantages:
  - Poor parallelism → low throughput.
  - Longer instruction execution time due to sequential transfers.

b. Two bus datapath

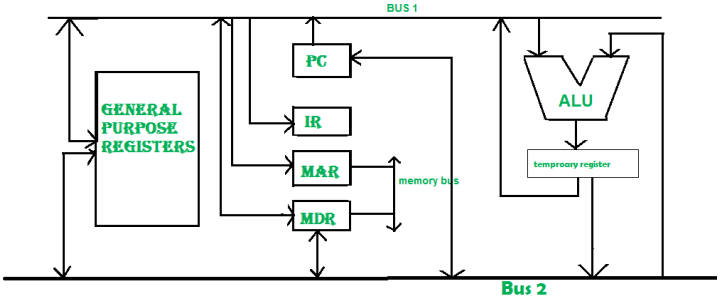


fig.TWO BUS ORGANISATION

- Structure:
  - Two internal buses (Bus A and Bus B) available for data movement.
  - ALU takes inputs from both buses and writes result back to one of them.
- Operation:
  - Can perform one register-to- register transfer and one ALU operation concurrently in the same cycle.
  - E.g., load register R□ → Bus A, register R□ → Bus B → ALU → result back to R_3 on Bus A.
- Advantages:
  - Better parallelism than single- bus → higher instruction throughput.
  - Still relatively simple compared to three-bus.
- Disadvantages:
  - Increased hardware cost (double buses, more multiplexers).
  - Still limited: only one ALU operation per cycle.

c. Three bus datapath

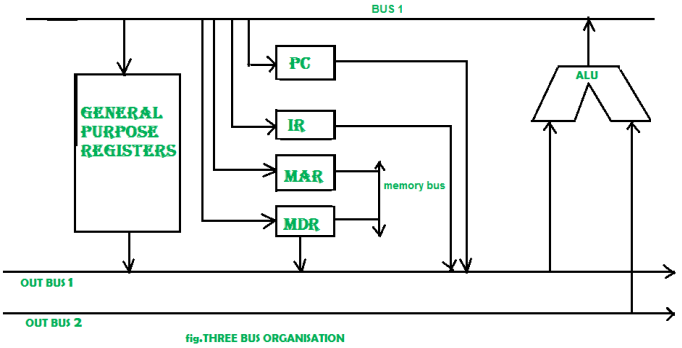


fig.THREE BUS ORGANISATION

- Structure:
  - Three distinct internal buses (Bus A, Bus B, Bus C).
  - Two source operands and one destination can be driven simultaneously.
- Operation:
  - Enables two reads and one write to registers in a single cycle.
  - E.g., read R□ → Bus A, R□ → Bus B → ALU → write result to R_3 via Bus C.
- Advantages:
  - Maximum data transfer parallelism for simple register-to- register and ALU ops.
  - Shortest instruction cycle times for register-based operations.
- Disadvantages:
  - Highest hardware overhead (three buses, extensive multiplexing).
  - Greater control complexity and cost.

3.3. Control Unit

a. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

b. The function of the control unit in a digital computer is to initiate sequences of micro-operations.

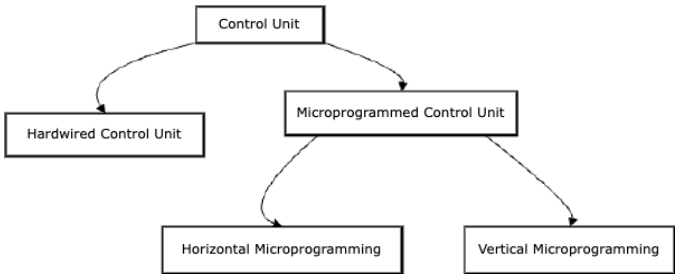


Fig. Types of Control Unit

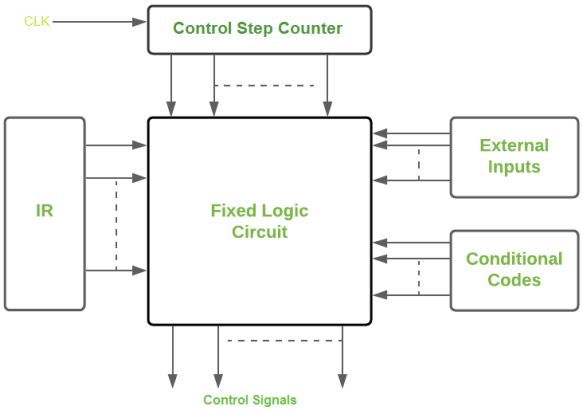
3.4.1. Microinstruction (Control Word)

Each word in control memory contains within it a microinstruction. The microinstruction specifies one or more microoperations for the system.

3.4.2. Microprogram

A sequence of microinstructions.
An instruction can be executed by performing one or more of the following operations in some
specified sequence:
a. Transfer a word of data from one processor register to another or to the ALU.
b. Perform an arithmetic or a logic operation and store the result in a processor register.
c. Fetch the contents of a given memory location and load them into a processor register.
d. Store a word of data from a processor register into a given memory location

3.4.3. Hardwired Control Unit



1. Implementation

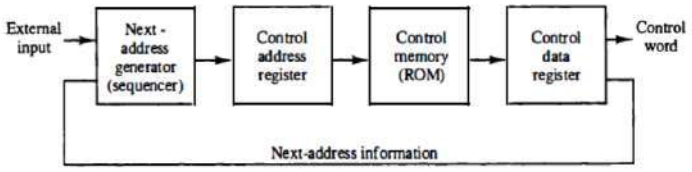
a. Fixed combinational and sequential logic (decoders, counters, gating circuits).
b. Control signals expressed as Boolean SOP (Sum-of-Products) functions of:

- ➔ Control step counter outputs (T_1, T_2,…).
- ➔ Instruction register bits (opcode).
- ➔ Condition codes & external flags (e.g. MFC, interrupt request).

2. Characteristics

- ➔ **Speed:** Very fast (single-cycle micro-operations).
- ➔ **Complexity:** Logic grows exponentially with ISA complexity.
- ➔ **Flexibility:** Difficult to modify or extend once designed.
- ➔ **Use Cases:** Simple RISC processors with limited instruction sets.

3.4.4. Microprogrammed Control Unit

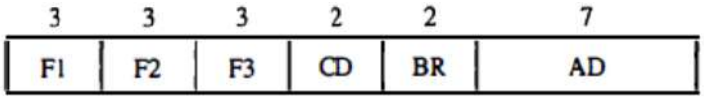


1. Implementation

a. Control signals generated by microinstructions stored in Control Memory (CM).

b. Each microinstruction (control word) encodes one or more control signals.
c. Sequencing via a Micro-Program Counter (μPC) & Address Sequencer.

2. Microinstruction Fields



F1, F2, F3 micro-operations fields, CD: Condition for branching, BR: Branch Field, AD: Address field.

3. Characteristics

➔ **Speed**: Slower than hardwired (multiple memory accesses).
➔ **Flexibility**: Easy to modify control sequences (update micro-program).
➔ **Complexity Handling**: Suited for complex ISAs (CISC) with many instructions.
➔ **Storage**: Requires ROM/RAM for control memory (2 K–10 K microinstructions).

3.4.4.1. Horizontal Microprogramming

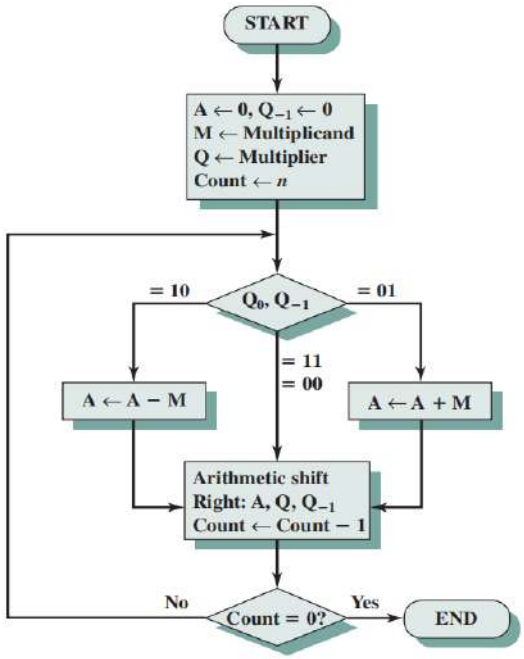
- Control Word: One bit per control signal → maximal parallelism.
- Word Width: Very wide (one bit × number of signals).
- Decoder: None (signals directly driven).

3.4.4.2. Vertical Microprogramming

- Control Word: Encoded fields (k-bits select 2^k signals).
- Word Width: Narrower, but requires decoders.
- Parallelism: Limited (typically one group executed per cycle).

3.5. Booth's Algorithm

3.5.1. Goal: Efficiently multiply two signed binary integers (two's-complement) with fewer add/subtract operations by encoding runs of 1's in the multiplier



3.5.2. Best Case and Worst Case Occurrence:

Best case is when there is a large block of consecutive 1's and 0's in the multipliers, so that there is minimum number of logical operations taking place, as in addition and subtraction.
Worst case is when there are pairs of alternate 0's and 1's, either 01 or 10 in the multipliers, so that maximum number of additions and subtractions are required.

3.5.3. Key Idea

➢ Examine pairs of bits of the multiplier (current LSB and an extra "previous" bit); decide whether to add, subtract, or do nothing with the multiplicand.
➢ Shift right each cycle, accumulating the partial product in a combined register.

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

3.5.4. Registers & Initialization

Register	Width	Initial Content
A	n-bit	0…0
Q	n-bit	Multiplier
M	n-bit	Multiplicand
Q_{-1}	1 bit	0
n	-	Number of bits in Q and M

ACC = [A (n bits) | Q (n bits) | Q_{-1}(1 bit)] starts as [0…0 | $Q_{initial}$ | 0]

3.5.5. Step-by- Step Algorithm

1. Initialize A = 0, Q = multiplier, Q_{-1} = 0, count = n.
2. Repeat until count = 0:

a. Examine (Q_0, Q□$_1$) and modify A per the decision rule.

b. Right-shift the triple [A | Q | Q$_{-1}$]:

i. New Q_{-1} ▯ old Q_0

ii. New Q ← old A_0 (least significant bit of A) … old Q□ … old Q□□□

iii. New A ← sign- extended shift of old A

3. Decrement count by 1.
4. Result is in [A | Q] (2n bits).

3.5.6. Example

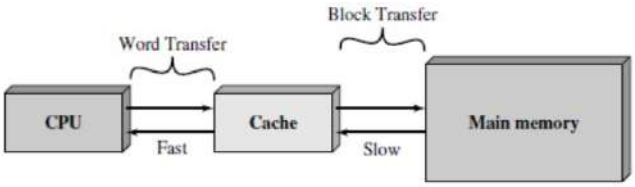
Multiply M = +3 (0011_2) by Q = −4 (1100_2) using 4-bit registers:

Cycle	A	Q	Q-1	Operation	Comment
Init	0000	1100	0	—	Initial values
1	0000	1100	0	(0,0): No op	Shift → 000011100
2	0000	1110	0	(0,0): No op	Shift → 000001110
3	0000	0111	0	(1,0): A ← A − M = 1101_2	Shift → 111000111
4	1110	0011	1	(1,1): No op	Shift → 111100011

∴ Final [A|Q] = 11110001_2 = −12□□, which is 3 × (−4).

4. Cache Memory Organisation

4.1. Introduction



- A small, fast SRAM buffer placed between the CPU and main memory.
- Holds copies of frequently accessed memory blocks (cache lines), exploiting temporal locality & spatial locality
- Goal: Reduce average memory access time by satisfying most requests from the cache rather than slower DRAM.
- CPU always generated MM address (even to access cache too)
- The performance of cache can be analysed with the following characteristics.
  - Cache size (Small in KB's)
  - Block or line size
  - No. of levels of cache
  - Cache mapping
  - Cache replacement policy
  - Cache updating scheme

4.2. Cache Organisation

➔ Cache Line (Block): Unit of transfer

➔ Fields in Address:

- ◆ Tag: Identifies which memory block is cached.
- ◆ Index: Selects a cache set or line.
- ◆ Offset: Chooses the byte/word within the cache line.

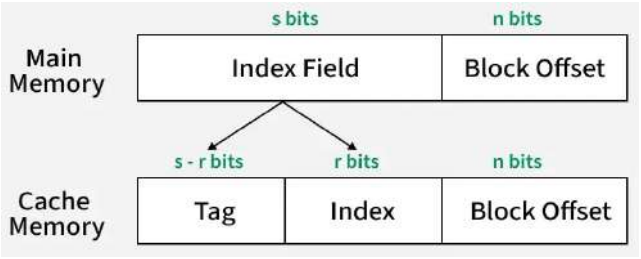
➔ Metadata:

- ◆ Valid Bit: Line contains valid data.
- ◆ Dirty Bit: Line has been written (for write-back).

Note:

1. CM block number = (MM block no.) modulo (no. of blocks/lines in cache)
2. No. of blocks = Cache size/Block size

4.3. Cache Mapping



4.3.1. Direct Mapping

Main Memory(MM) Address is

Tag	Cm block number	Byte offset

- Index in direct mapping = cm block number
- Tag in direct mapping = mm address - $\log_2$(cache size)
- MM block no. = Tag + cm block no.

Note:

1. Tag directory size (all mappings) = Number of blocks in cache * (tag + extra bits)
2. For a given cache size, block size and mm size: Tag is same (for byte and word addressable memory both)

4.3.2. Set Associative Mapping

Tag	Set offset	Byte offset

- Cm set number = (mm block no.) % no. of sets in cache
- Index in set associative mapping = Set offset
- Tag in K-way set associative mapping = mm address – $\log_2$(cache size) + $\log_2$K

4.3.3. Fully Associative Mapping

Tag	Byte offset

- Index in fully associative mapping = 0-bits
- Tag in fully associative = mm address – $\log_2$(block size)
- In fully associative mapping, tag = mm block no.

Note:

1. Size of tag is maximum in fully associative & minimum in direct mapping.
2. Size of index is minimum in fully associative & maximum in direct mapping.

4.4. Hardware Implementation

4.4.1. Direct Mapping

- ➢ Number of MUX for tag selection = Tag-bits
- ➢ Size of MUX for tag selection = Number of blocks : 1
- ➢ Number of comparators = 1
- ➢ Size of comparator = Tag-bits

4.4.2. K-way Set Associative Mapping

➢Number of MUX for tag selection = K * Tag-bits
➢Size of MUX for tag selection= Number of set : 1
➢Number of comparators = k
➢Size of comparator = Tag-bits
➢OR-gate = 1 (k-input OR gate)

4.4.3. Fully Associative Mapping

➢Number of comparators = Number of blocks in cache
➢Size of comparator = Tag-bits
➢OR-gate = 1 (number of blocks-input OR gate)

Note: Hit Latency Time
➢Direct mapping = MUX delay + comparator delay
➢Set associative mapping = MUX delay + comparator delay + OR-gate delay
➢Fully associative mapping = comparator delay + OR-gate delay

4.5. Types of misses in cache

4.5.1. Compulsory/Cold Miss

The very first access to a block that can't be in the cache, so the block must be brought into cache. These are also called first reference misses.

4.5.2. Capacity Miss

If the cache cannot contain all the blocks needed to during execution of a program. Capacity misses will occur because of blocks being discarded and later retrieved.

4.5.3. Conflict Miss

When multiple blocks compete for the same cache line (or set) under the chosen mapping, even though other lines are free.

Note:

1. To reduce conflict miss: increase associativity
2. To reduce cold miss: increase block size
3. To reduce capacity miss: increase cache size

4.6. Cache Replacement Policies

4.6.1. FIFO (First-In-First-Out)

It replaces the cache block having the longest time stamp with a new block.

4.6.2. LRU (Least Recently Used)

It replaces the cache block which is having less no. of references with the longest time stamp with a new block.

4.7. Cache Write Policy

4.7.1. Write Through VS Write Back

Feature	Write Through	Write Back
On Write Hit	Update both cache and main memory immediately.	Update only cache; mark line as dirty.
On Eviction	N/A (data already in memory)	If dirty, write entire block back to memory.
Memory Traffic	High (every write generates a memory write)	Lower (writes only on dirty-line evictions)
Data Consistency	Always consistent between cache & memory.	Memory stale until write-back occurs.
Hardware Needs	Simple; no dirty bits required.	Requires dirty bit per line and write-back logic.
Typical Use	L1 caches for simplicity & predictability	L2/L3 caches to reduce bus traffic

4.7.2. Write Miss Handling

4.7.2.1. Write Allocate

On a write miss, fetch the block into cache, then perform the write (marking it dirty under write-back).

4.7.2.2. No-Write-Allocate

On a write miss, bypass cache and write directly to main memory; cache remains unchanged.

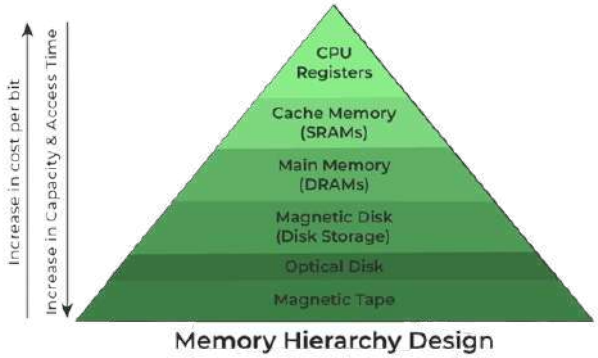
4.7.3. Write Buffering: Reduces CPU stalls on write operations by enqueuing the write in a write buffer.

4.7.4. Summary

Policy Combination	Pros	Cons
Write Through + No Allocate	Simple; consistent memory; no block pollution on writes.	High memory traffic; write misses still go to memory.
Write Through + Write Allocate	Subsequent reads of the block benefit from caching.	Still high memory traffic.
Write Back + Write Allocate	Lowest memory traffic; good for write-heavy workloads.	Complex; must track dirty lines; potential data staleness.
Write Back + No Allocate	Rarely used	Not practical; block writes aren't cached, so dirty bits unused.

5. Memory Organisation

5.1. Memory Hierarchy



Memory Hierarchy Design

5.2. Types of memory (Based on methods of accessing)

5.2.1. Sequential Access Memory

- Data is accessed in a fixed linear order.
- Example: Magnetic tape.
- Use Case: Archival storage (low cost, high capacity), not for random reads/writes.

5.2.2. Direct Access Memory

- Allows access to a record by first moving to a general area (track/sector), then sequentially to the exact record.
- Example: Hard disks, optical disks (CD/DVD).
- Use Case: File systems, databases;moderate access time, large capacity.

5.2.3. Random Access Memory (RAM)

- Uniform constant-time access to any location
- Each address has a dedicated physical path (wired) for immediate read/write.
- Examples:
  - DRAM (Dynamic RAM)
  - SRAM (Static RAM)
- Use Case: Primary/main memory, CPU caches (fastest at their level).

5.2.4. Associative (Content-Addressable) Memory

- Retrieves data by content rather than by specific address
- All words are compared simultaneously; matching word(s) are returned.
- Example: Translation Lookaside Buffer (TLB) in virtual memory.
- Use Case: Fast lookups (e.g., cache tags, TLB), where search key determines the fetch.

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

Note: Average Memory Access Time (AMAT) = Hit Time+Miss Rate×Miss Penalty

5.3. Difference b/w SRAM & DRAM

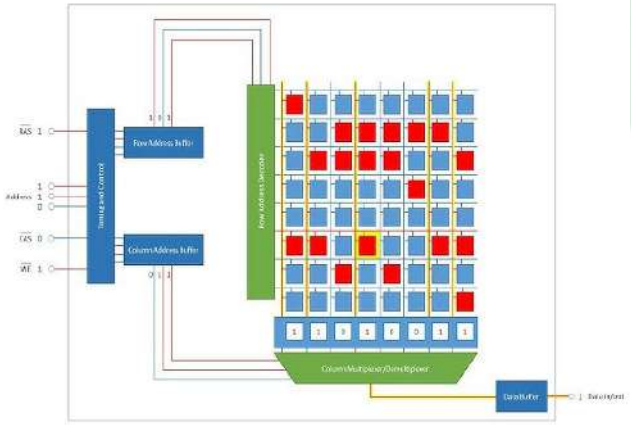
SRAM

Static
1. Implemented using flip-flops
2. No refresh required
3. Faster Read/Write
4. Used for Cache
5. Low Idle power consumption
6. High operational power consumption

DRAM

Dynamic
1. Implemented using capacitors
2. Periodic refresh is required
3. Slow Read/Write
4. Used for main memory
5. High Idle power consumption
6. Low operational power consumption

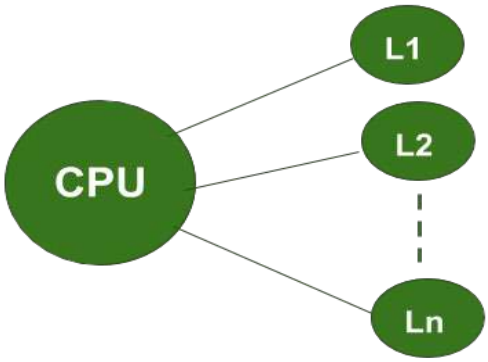
Note: DRAM consists of rows of cells & DRAM Refresh time = no. of rows of cells in DRAM * 1 cell refresh time



5.4. Types of Memory Access

5.4.1. Simultaneous Access

CPU can access the data simultaneously from all levels of memory



Consider n levels, then;

$T_{avg} = H_1*T_1 + (1-H_1)*H_2*T_2 +...+ (1-H_1)*(1-H_2)...(1-H_{n-1})*H_n*T_n$

(where H_n is Hit Ratio, T_n be access time for each level)

5.4.2. Hierarchical Access

Data comes from other levels to Level 1 then CPU gets its access as shown in the figure below.

Consider n levels, then;

$T_{avg}=T_1+(1-H_1)T_2+(1-H_1)(1-H_2)T_3+...+(1-H_1)*(1-H_2)...(1-H_{n-1})*T_n$

Note:

1. Memory Access Rate = 1/cycle time
2. Multiplication table for 2, n-bit unsigned number = 2^{2n} * 2n bits
3. Addition table for 2, n-bit unsigned number = 2^{2n} * (n + 1) bits

5.5. Locality Principles

5.5.1. Temporal Locality: Recently accessed data likely to be reused soon.

5.5.2. Spatial Locality: Data near recently accessed addresses likely to be accessed soon.

Note: Caches exploit both to deliver high hit rates.

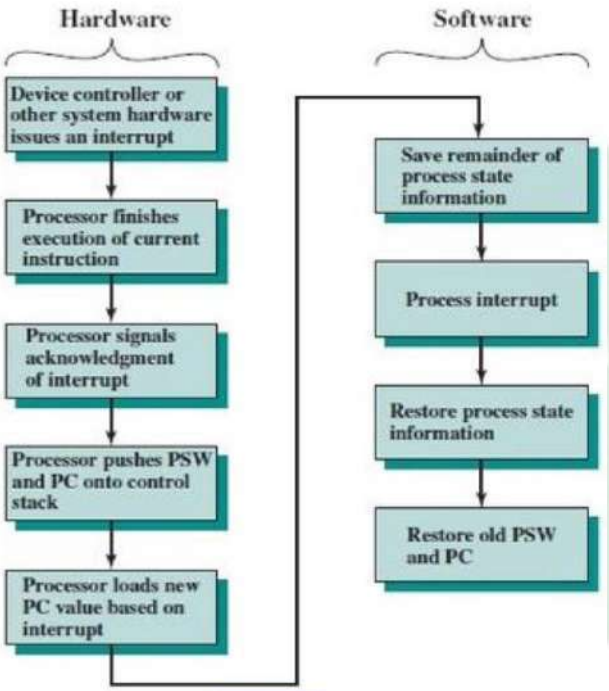
5.6. Memory representation

Eg: 2^{14} x 32 bits memory means,
2^{14} addresses and 32 bits wide word

5.7. Interrupts

5.7.1. Introduction

An interrupt is a signal that causes the CPU to temporarily halt the current execution and jump to a specific service routine (ISR i.e. Interrupt Service Routine) to handle an event (e.g., I/O completion, error, etc.).



Simple Interrupt Processing

5.7.2. Types of Interrupts

5.7.2.1. Vectored Interrupts

A unique ISR address is supplied (either directly or via vector table) to reduce the time involved in the polling process.

5.7.2.2. Non-Vectored Interrupts

CPU jumps to a general or fixed location (e.g., predefined interrupt handler), and software determines the source.

5.7.2.3. Maskable Interrupts

- Interrupts that can be disabled or ignored by the CPU using a special flag or instruction.
- Used for lower-priority or non-critical tasks.
- Eg. INTR in 8085, I/O completion.
- Controlled By: Interrupt Enable/Disable instructions (EI, DI)

5.7.2.4. Non-Maskable Interrupts

- Cannot be disabled by the CPU; always gets attention.
- Used for critical events (e.g., power failure, hardware fault).
- Eg. TRAP in 8085.
- Priority: Highest as it overrides all other interrupts.

Note: Difference b/w Interrupts and Exceptions

- Exceptions are caused by software executing instructions. Eg. a page fault, or an attempted write to read only page. An expected exception is 'trap', unexpected is a "fault".
- Interrupts are caused by hardware devices. Eg. device finishes I/O, timer fires.

6. Pipelining

6.1. What Is Pipelining?

- Pipelining is a technique of overlapping the execution of multiple instructions by dividing the processor's datapath into stages, each handling a part of the instruction.
- It increases instruction throughput without reducing the execution time of individual instructions.

6.2. Types of Pipelines

6.2.1. Linear Pipeline

Used for single specific function

6.2.2. Non-linear Pipeline

At a given stage, multiple parallel functional units or paths exist
rather than a single straight chain.

6.2.3. Synchronous Pipeline

On a common clock, all registers transfer data to next stages simultaneously.

6.2.4. Asynchronous Pipeline

6.3. Performance of Pipeline

6.3.1.

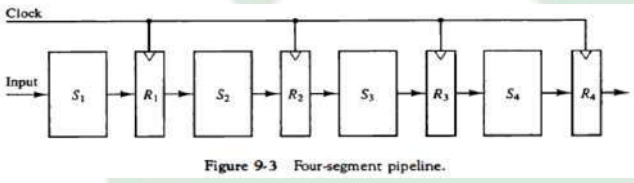
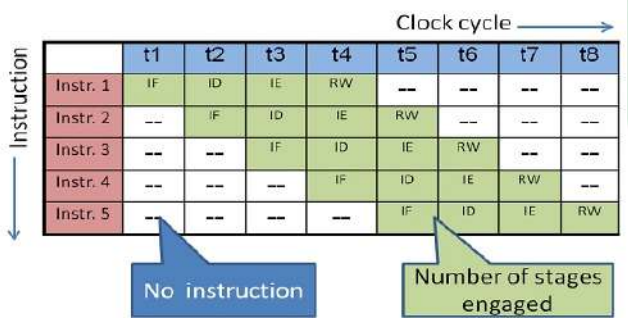


Figure 9-3 Four-segment pipeline.

Suppose 5 inst^n, T_1 through T_5, pass through a pipeline of four stages.



- Cycle 1: Stage 1 processes T_1
- Cycle 2: Stage 2 works on T_1 while Stage 1 begins T_2
- Cycle 3: Stage 3 takes on T_1, Stage 2 handles T_2, and Stage 1 starts T_3
- Cycle 4: Stage 4 completes T_1; meanwhile, earlier stages are busy with T_2, T_3, and T_4

After these four cycles, the pipeline is "full," and thereafter it finishes one task per cycle.

More generally, for a pipeline with k stages and clock period t□ running n instructions:

- First inst^n takes $k*t_p$ to traverse all stages.
- Remaining (n−1) inst^n each completes in one additional t□

Total time: $k*t_p + (n-1)*t_p = (k+n-1)*t_p$

Note:

1. To complete n tasks using a k-segment pipeline requires k + (n - 1) clock cycles
2. T_p = max(all segment/stage delays) + register delay

Note: Consider a nonpipeline unit that performs the same operation and takes time equal to t_n to complete each task. The total time required for n tasks is $n*t_n$, where t_n = sum of all segment delays

6.3.2. Speedup (S)

6.3.2.1. The speedup of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio, S = $\frac{□×□_□}{(□+□-1)×□_□}$

6.3.2.2. In ideal conditions i.e. n>>>k, we ignore (k-1), then S = $\frac{□_□}{□_□}$

6.3.2.3. If we assume that the time it takes to process a task is the same in the pipeline and non-pipeline circuits, we will have $t_n = k*t_p$, then S = k
In this scenario, maximum speedup is achieved.

6.4. Latency & Throughput

6.4.1. Latency

After how much time i/p is given to the system

- Pipeline Latency = t_p (∵ after every cycle, new i/p)

- Non-Pipeline Latency = t_n

6.4.2. Throughput

No. of tasks/instⁿ per unit time

- Pipeline: In ideal case = $1/t_p$

6.5. Instruction Pipelining

It overlaps the fetch-decode-execute phases of multiple instructions in a linear chain of stages, so each clock cycle a different instruction occupies each stage.

Eg. Consider a four stage pipeline as follows:

1. FI is the segment that fetches an instruction.
2. DA is the segment that decodes the instruction and calculates the effective address.
3. FO is the segment that fetches the operand.
4. EX is the segment that executes the instruction.

We assume the CPU has separate instruction and data memories, so it can do an instruction fetch (FI) and a data fetch (FO) at the same time.

At cycle 4 in our 4-stage pipeline:

- Stage EX is finishing Instruction 1
- Stage FO is loading the operand for Instruction 2
- Stage DA is decoding Instruction 3
- Stage FI is fetching Instruction 4

Now suppose Instruction 3 turns out to be a branch. As soon as it's decoded in DA (still in cycle 4), we stop feeding any new instⁿ from FI into DA until we know where the branch goes.

- If the branch is taken, we throw away what was in FI/DA and fetch the correct next instⁿ in cycle 7.
- If the branch is not taken, we keep and use the instⁿ that FI already fetched in cycle 4, and the pipeline keeps flowing as normal.

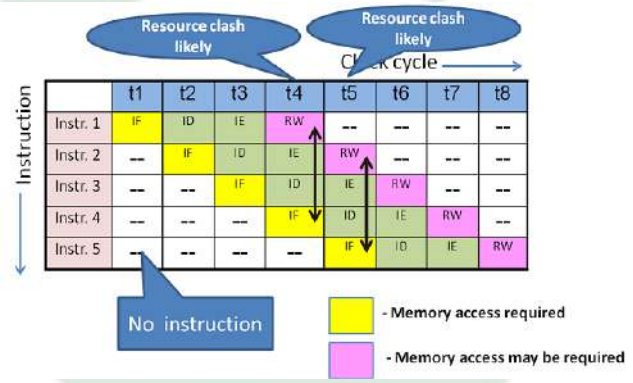
Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure 9-8 Timing of instruction pipeline.

6.6. Pipeline Hazards/Conflicts

6.6.1. Structural Hazard/Resource Conflict

☐☐



Caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction & data memories.

Solution:

1. Incure stall cycles
2. Increase no. of resources

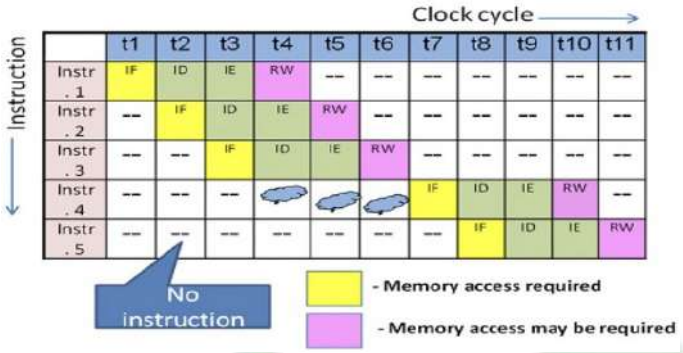
Stall cycles

Also called a pipeline bubble, a stall is a clock cycle in which no new instruction completes, inserted to resolve a hazard or resource conflict.

Solution of above figure:

Introduce bubble/stall cycles which stalls the pipeline as in figure besides.

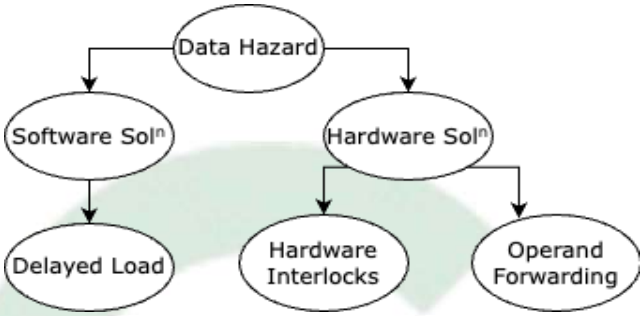
At t4, I4 is not allowed to proceed, rather delayed. It could have been allowed in t5, but again a clash with I2 RW. For the same reason, I4 is not allowed in t6 too. Finally, I4 could be allowed to proceed (stalled) in the pipe only at t7.



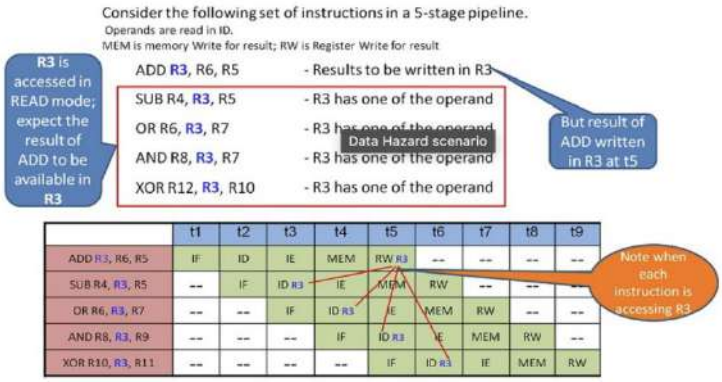
Solⁿ of Structural Hazard

Note:

- Stalls because of branch = i – 1 (if after i^{th} stage, the condition is evaluated)
- Even if branch is not taken, then too stalls are there due to branch instructions
- Result of branch condition evaluation is available after execution phase of branch instruction



6.6.2. Data Hazard/Data dependency



A data dependency occurs when one instruction needs data that is produced or used by another instruction,creating a potential conflict in a pipeline.

Eg.

Conflicts

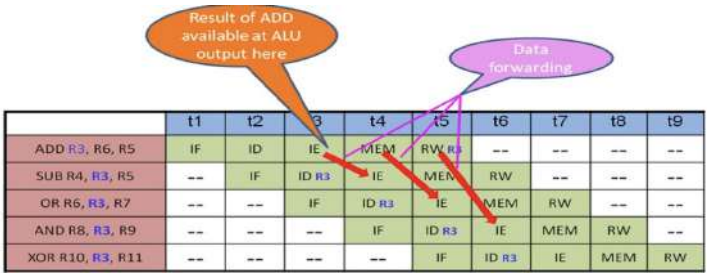
In the above case, ADD instruction writes the result into the register R3 in t5. If stalls are not introduced to delay the next SUB instruction, all three instructions would be using the wrong data from R3, which is earlier to ADD result. The program goes wrong! The possible solutions are:

6.6.2.1. Hardware Interlocks

Hardware Interlocks are built-in pipeline controls that detect hazards at runtime & automatically stall the pipeline to prevent incorrect execution.

6.6.2.2. Operand/Data Forwarding

Also called bypassing, is a hardware technique that routes results directly from one pipeline stage to an earlier stage that needs them, avoiding a stall.



6.6.2.3. Delayed load/No operation

Delayed Load is a compiler technique that avoids a pipeline stall after a load instruction

6.6.2.4. Data Hazard Classifications

Assume 2 inst^n i & j

i: $R_1 \leftarrow R_2 + R_3$

j: $R_5 \leftarrow R_1 + R_4$

If j reads a source before its written by i, hence j gets incorrect value

➢ WAW (Write After Write)

i: $R_1 \leftarrow R_2 + R_3$

j: $R_1 \leftarrow R_4 * R_5$

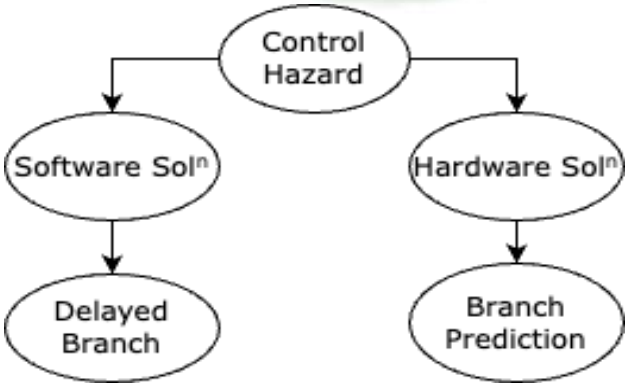
If j writes a destination before its written by i

➢ WAR (Write After Read)

i: $R_1 \leftarrow R_2 + R_3$

j: $R_2 \leftarrow R_9 + R_7$

6.6.3. Control hazard



6.6.3.1. Delayed Branch

A technique where the instruction immediately following a branch, "delay slot"

by reordering subsequent instructions or inserting a "no-op" slot before using the loaded data.

➢ RAW (Read After Write)

If j writes a destination before its read by i, hence i reads incorrect value

Note:

1. RAW is True Dependency while WAR & WAW are False Dependencies as they can be solved by register renaming.
2. WAR is known as Anti-dependency
3. WAW is known as Output Dependency
4. Operand forwarding and register renaming can not solve the memory access dependencies

is always executed, regardless of whether the branch is taken. The compiler fills this slot with a safe instruction (or a NOP) to hide the branch-decision penalty and improve pipeline throughput without extra hardware.

6.6.3.2. Branch Prediction

It is hardware logic that guesses the outcome of a conditional branch before it is resolved, allowing the pipeline to continue fetching along the predicted path and avoid stalls when the guess is correct.

7. Secondary Memory

7.1. Introduction

7.1.1. Platters & Surfaces

One or more circular platters coated with magnetic material; each surface has its own read/write head.

7.1.2. Track

A concentric circle on a platter surface. All tracks on different platters at the same radius form a cylinder i.e. No. of cylinders = No. of tracks

7.1.3. Sector

- Smallest addressable unit on a track (typically 512 B)
- Contains user data + control information (format overhead)

7.1.4. Gaps

7.1.4.1. Inter-Sector Gap: Empty region between adjacent sectors.

7.1.4.2. Inter-Track Gap: Empty region between adjacent tracks.

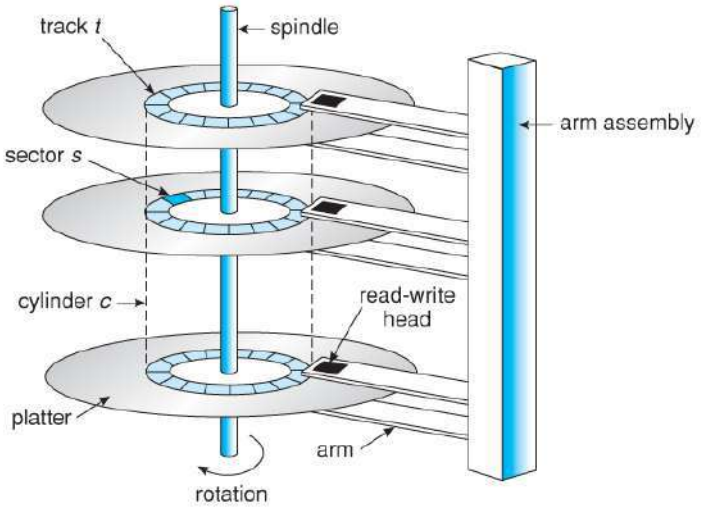


Fig. Disk Structure

7.2. Types of Disk Constructions

7.2.1. Constant Track Capacity (CTC)

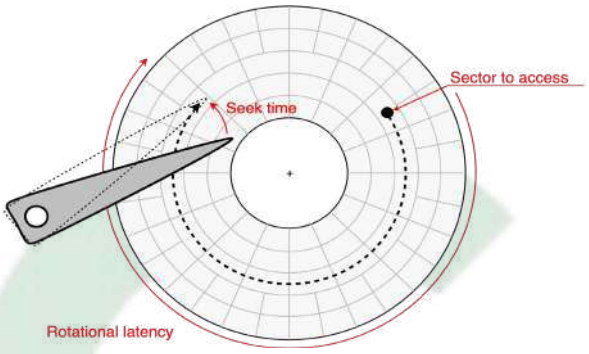
Variable recording density: Outer tracks hold more sectors than inner tracks.
Maintains constant angular velocity.

7.2.2. Variable Track Capacity (VTC)

Constant recording density: Same number of sectors per track; disk spins slower at outer tracks to equalize data rate.

Note: Each rotation of disk covers 1 track

7.3. Disk Performance



7.3.1. Seek Time (T_s)

Time to move the R/W head from its current track to the target track.

7.3.2. Rotational Latency (T_r)

Time for the platter to spin so the desired sector comes under the head.

T_r = (rotation time)/2

7.3.3. Transfer Time ($T_{transfer}$)

Time to actually read/write the bits in the sector once under the head.

7.3.4. Overhead Time ($T_{overhead}$)

Controller delays (e.g., command processing).

7.3.5. Disk Capacity

Disk Capacity = 2 * no. of platters * tracks per surface * sectors per track * sector capacity

Note:

1. $T_{avg} = T_s + T_r + T_{transfer} + T_{overhead}$(if given)
2. Sequentially stored N sector transfer time = Seek Time + Rotational Latency + N * (1 sector Transfer Time)
3. Randomly stored N sector transfer time = N * (Seek Time + Rotational Latency + 1 sector Transfer Time)

7.4. Disk Addressing

Disk addressing < c, h, s > , where c = cylinder number, h = surface number, s = sector number

- Sector number for given address = c * sectors per cylinder + h * sectors per track + s
- c = sector number / sectors per cylinder
- h = (sector number % sectors per cylinder) / sectors per track
- s = (sector number % sectors per cylinder) % sectors per track

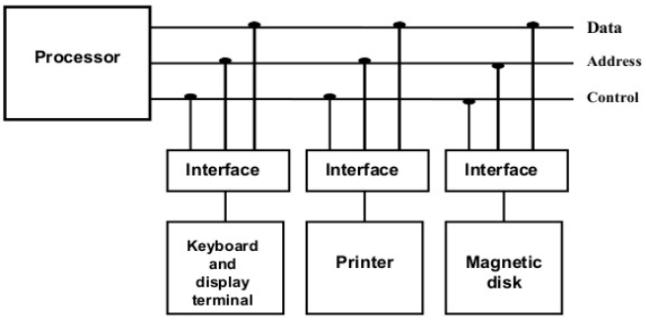
8. I/O Interface

8.1. Introduction

Input-output interface provides a method for transferring information between internal storage and external I/O devices.

There are 3 ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and IO but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.



Connection of I/O bus and input-output device

8.2. I/O processor (IOP)

➔ Same memory bus for both CPU & IOP

➔ IOP communicates with I/O devices through a separate I/O bus with its own address, data and control lines.

8.3. Isolated I/O

Common address space and different control lines

8.4. Memory mapped I/O

➔ take few addresses from memory address space

➔ same address space for both memory and I/O

Memory Mapped IO
1. Memory wastage
2. All Memory access instructions used for IO access also
3. No separate address space for IO
4. More Instructions for IO access
5. More addressing modes for IO access
6. More IO devices connected

Difference between Memory mapped I/O & Isolated I/O

8.5. I/O Interface

3.5.1. Programmed I/O
3.5.2. Interrupt driven I/O
3.5.3. Direct Memory Access (DMA)

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

8.5.1. Programmed I/O

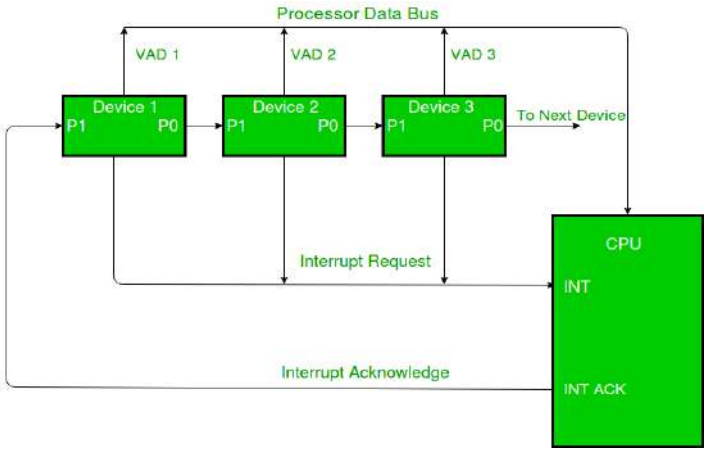
With programmed I/O, data is exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation. If the processor is faster than the I/O module, this is wasteful of processor time.
Time in programmed IO = time to check status + time to transfer data

8.5.2. Interrupt driven I/O

a. Interrupt I/O is a data transfer technique where the CPU is interrupted by the I/O device only when it is ready (e.g., after completing a data transfer or requiring service).
b. It eliminates constant polling of the device, thus increases CPU efficiency.
Time in Interrupt IO = interrupt overhead + time to service interrupt

8.5.2.1. Daisy– Chaining Priority (Serial– Priority Interrupt)

The system consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in first position followed by lower–priority devices.



8.5.3. Direct Memory Access

8.5.3.1. Definition & Purpose

- ➔ DMA is a mode of data transfer where a dedicated DMA controller moves data directly between I/O device and main memory without CPU intervention for each word.
- ➔ Frees the CPU from high-volume data transfers, improving overall system throughput.

8.5.3.2. DMA Controller

- ➔ Address Register: Holds the current memory address for transfer.
- ➔ Count Register: Holds the number of words/bytes left to transfer.
- ➔ Control Logic: Generates read/write and bus-request signals.
- ➔ Bus Arbitration Interface: Requests and releases the system bus from the CPU.

8.5.3.3. DMA Transfer Sequence

a. CPU Initialization:

- Loads DMA controller's Address and Count registers.
- Issues a "Start DMA" command.

b. Bus Request: DMA controller asserts Bus Request (BR); CPU grants via Bus Grant (BG).
c. Data Transfer:

- DMA takes control of the bus, reads from device → writes to memory (or vice versa) automatically.
- Address register increments/decrements; Count register decrements.

d. Completion: When Count = 0, DMA releases bus and raises an Interrupt to CPU indicating end of transfer.

8.5.3.4. DMA Modes

Mode	Description	CPU
Burst Mode	DMA transfers the entire block in one bus-hold period.	CPU paused for the burst
Cycle Stealing	DMA takes the bus for one transfer, then releases; later it will 'steal' memory cycle when CPU is idle.	CPU slowed slightly
Interleaving Mode	DMA transfers only when CPU is not using the bus.	Minimal interference

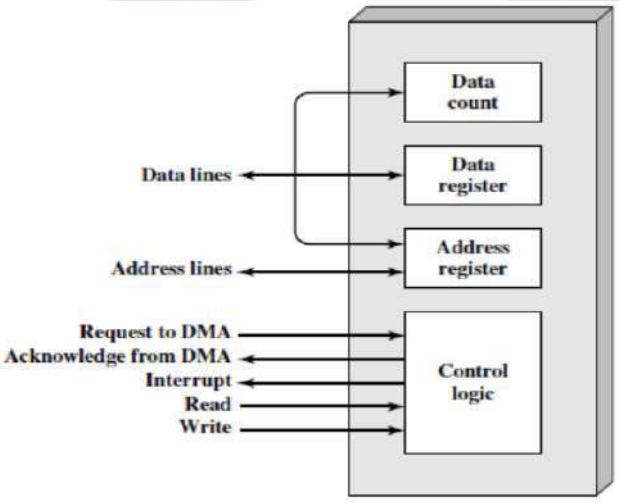


Figure 7.11 Typical DMA Block Diagram

Note:

1. % of time CPU blocked (burst mode)

$$\frac{transfer\ time\ to\ memory}{prepatation\ time + transfer\ to\ memory\ time} * 100\%$$

2. % of time CPU blocked (cycle stealing)

$$\frac{transfer\ time}{prepatation\ time} * 100\%$$

3. Max data transferred using DMA without CPUs intervention =

$2^x - 1$, x = bits in data count

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW