Lecture-2

Introduction to CAD

Digital computer: A machine that can solve problems for people by carrying out instructions given to it.

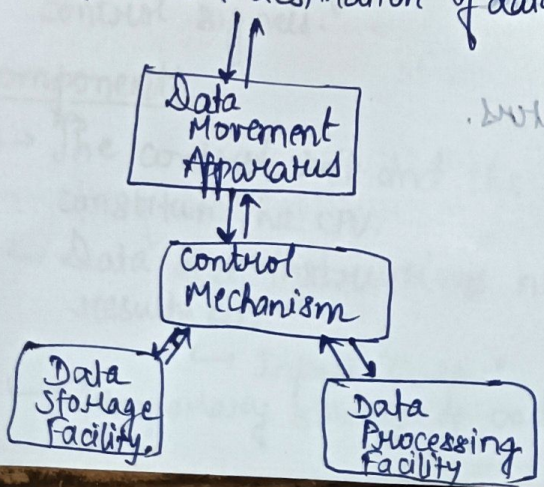
Computer Architecture

- The set of data types, operations and features at each level
- Deals with those aspects that are visible to programmers
  - Memory, control unit, ALO, I/O
  - Architecture and organization are essentially the same.
- Functional operation of the individual hardware units in a computer system, the flow of information among, and control of those units.
- Computer architecture and organization deals with structure and function of computers.
  - Structure is the way in which components relate to each other.
  - Function is the operation of individual components as part of the structure.

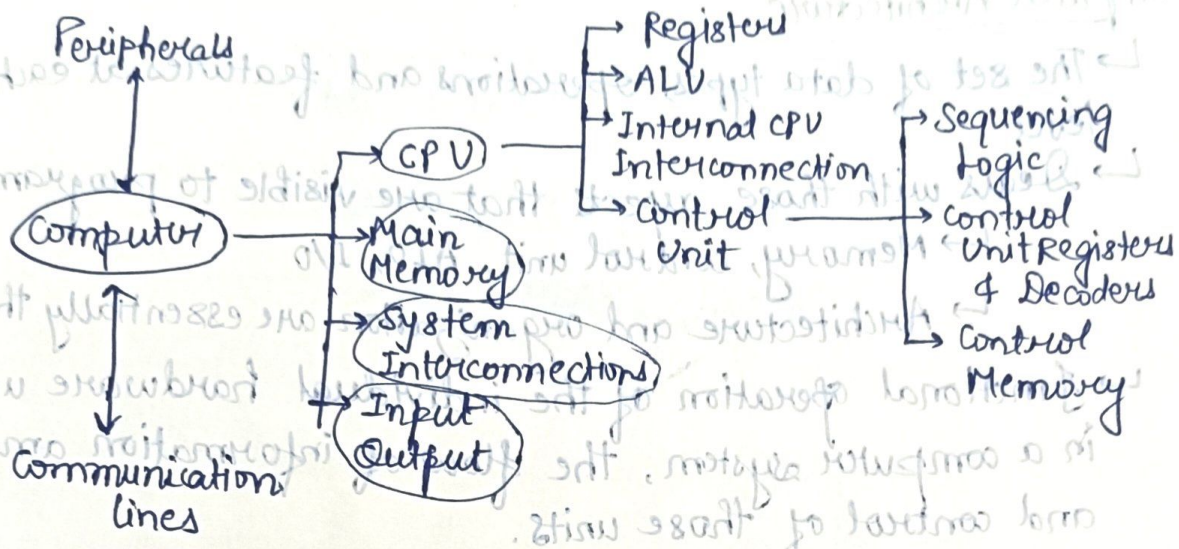
Functions of Computer:

- Data processing
- Data storage
- Data movement
- Control

Functional view: Operating environment (source & destination of data)



Structure of a Computer

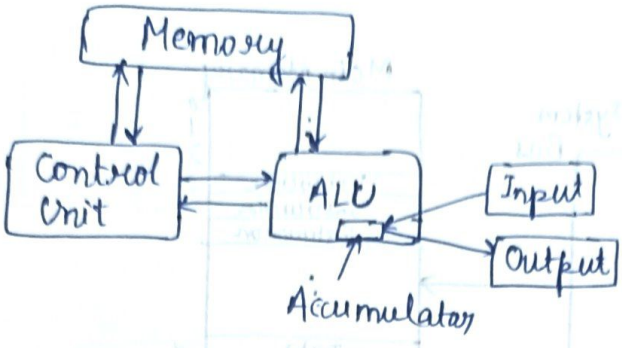


Computer Generations

- Zeroth Generation
  - ↳ Mechanical computers (1642–1945)
- First Generation
  - ↳ Vacuum Tubes (1945–1955)
- Second Generation
  - ↳ Transistors (1955–1965)
- Third Generation
  - ↳ Integrated circuits (1965–1980)
- Fourth Generation
  - ↳ Very Large Scale Integration (1980–?)
- Fifth Generation
  - ↳ Artificial Intelligence
- Sixth Generation
  - ↳ Quantum computers.

Lecture-3

Von Neumann Machine:



Lecture-5

Moore's law

- Increased density of components on chip
- No. of transistors on a chip will double every year.
- Cost of a chip has remained almost unchanged.
- Higher packing density means shorter electrical paths, giving higher performance.
  - Smaller size gives increased flexibility.
  - Reduced power and cooling requirements.
  - Fewer interconnections increases reliability.

Program

- A sequence of steps
- For each step, an arithmetic or logical operation is done.
- For each operation, a different set of control signals is needed.

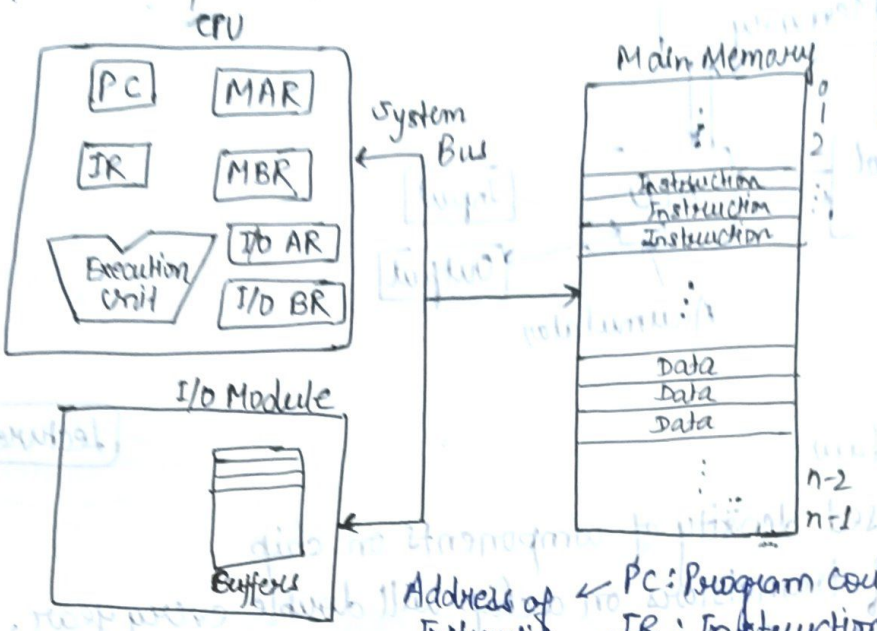
Control Unit

- For each operation a unique code is provided (ADD, MOVE, etc.).
- A hardware segment accepts the code and issues the control signals.

Components

- The control unit and the Arithmetic and logic unit constitute the CPU.
- Data and instructions need to get into the system and results out
  - Input/Output
- Temporary storage of code & results is needed
  - Main memory.

Computer Components



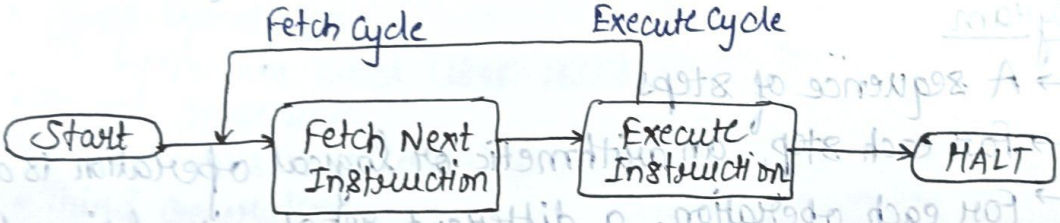
Address of Instruction

Instruction being executed

PC: Program counter
IR: Instruction Register
MAR: Memory Address Register
MBR: Memory Buffer Register
I/O AR: Input/output Add. Register
I/O BR: Input/output Buffer Register

Instruction Cycle

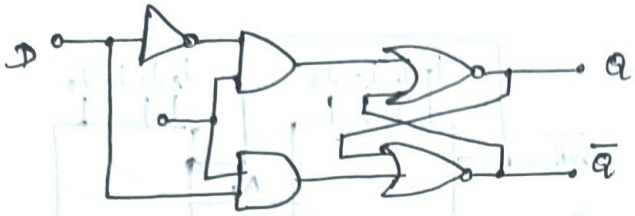
- Fetch
- Execute



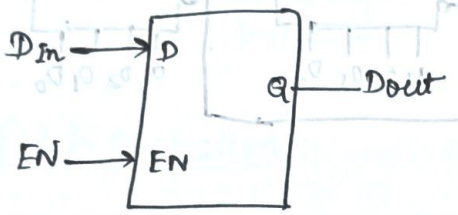
Fetch cycle:

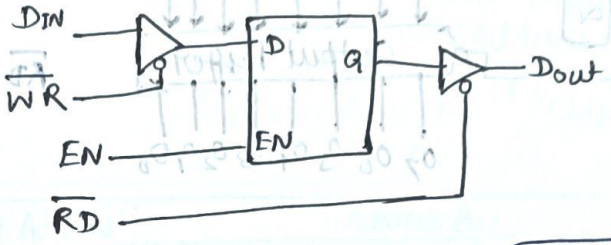
- PC holds address of next instruction to fetch.
- Processor fetches instruction from memory location pointed to by PC.
- Increment PC
  - Unless told otherwise.
- Instruction loaded into IR.
- Processor interprets instruction and performs required actions.

A single bit memory using D Latch:



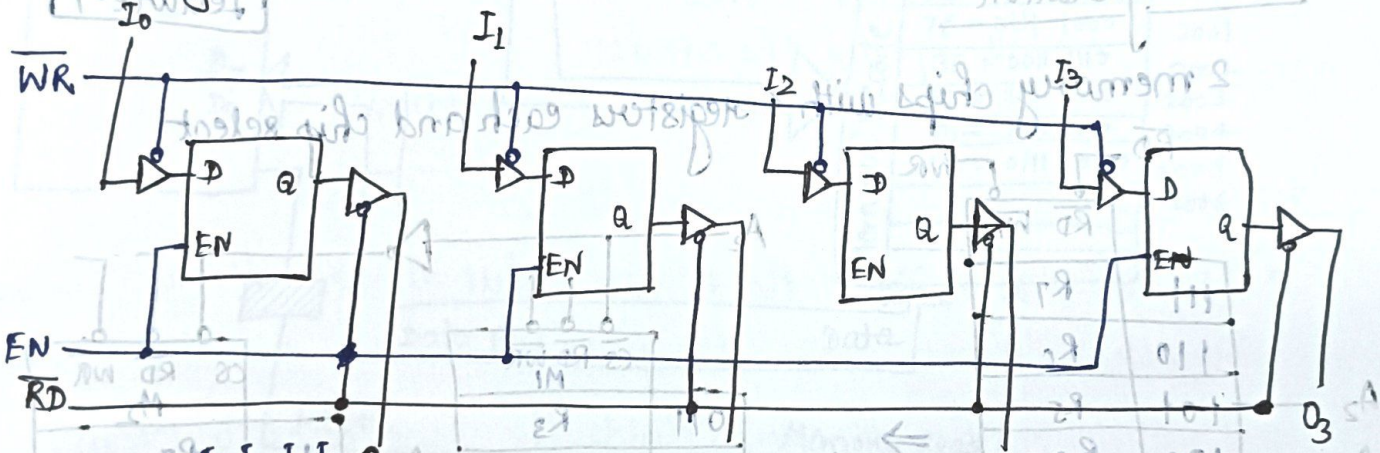
D	S	R	Q	State
	0	0	Previous state	No change
0	0	1	0	Reset
1	1	0	1	set
	1	1	?	Forbidden

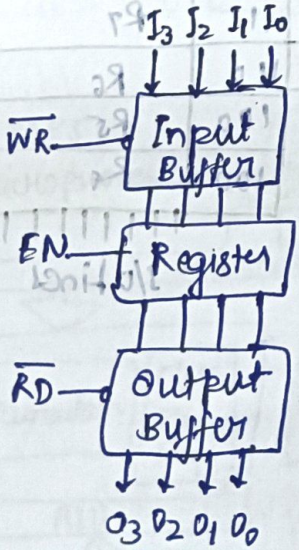


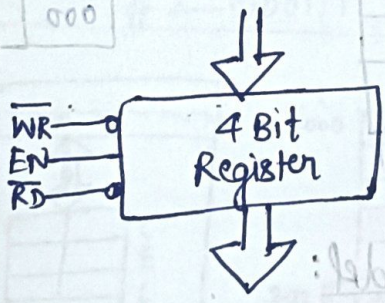


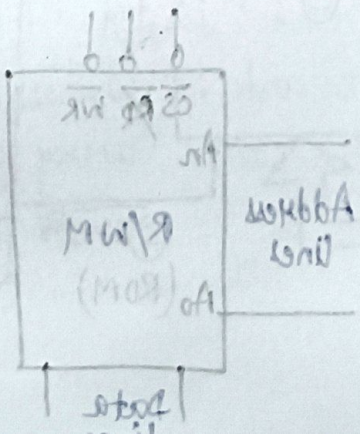
Lecture - 6

4 Bit Register

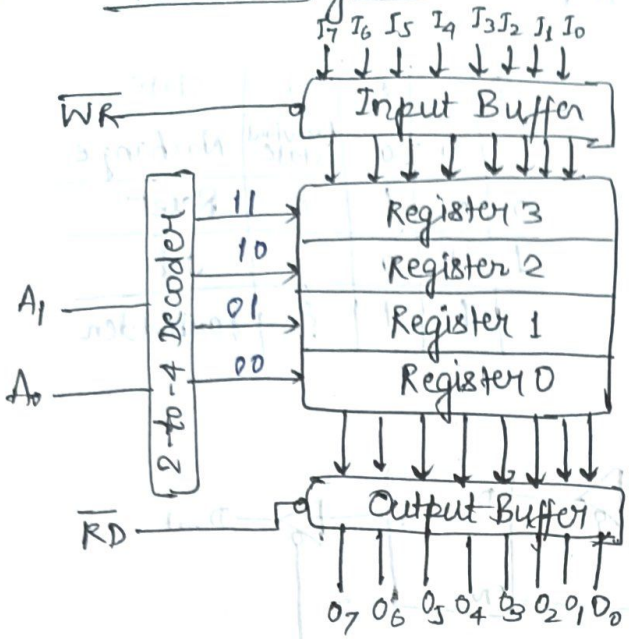


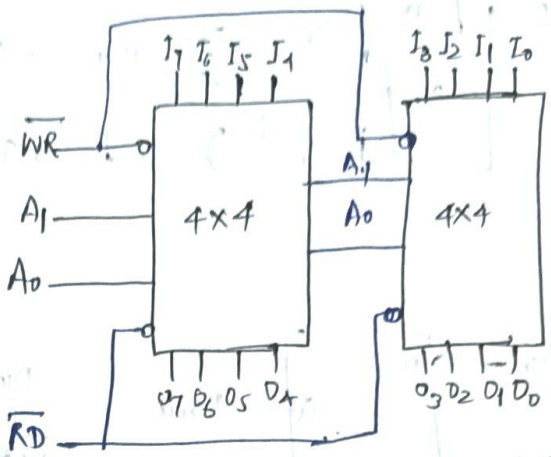






4x8-Bit Register

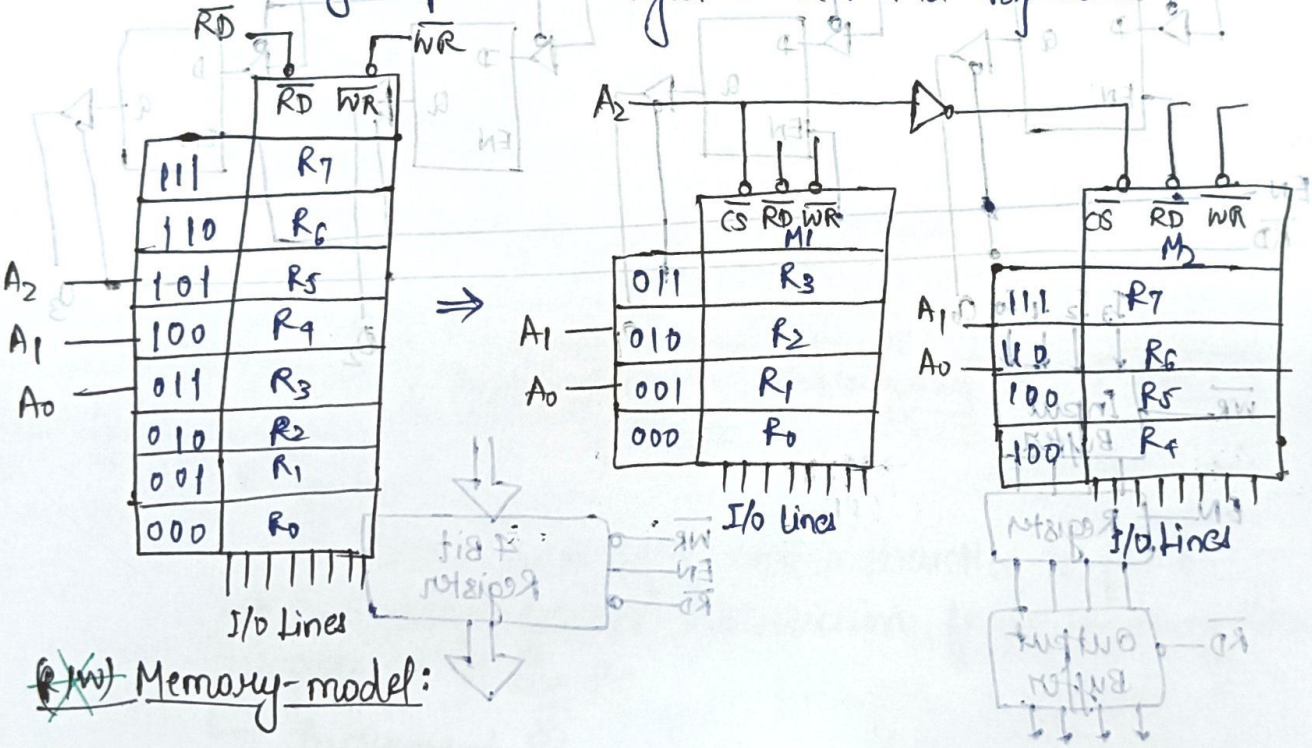




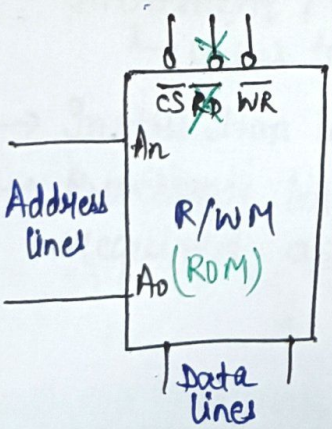
Fetch Operation

Lecture-7

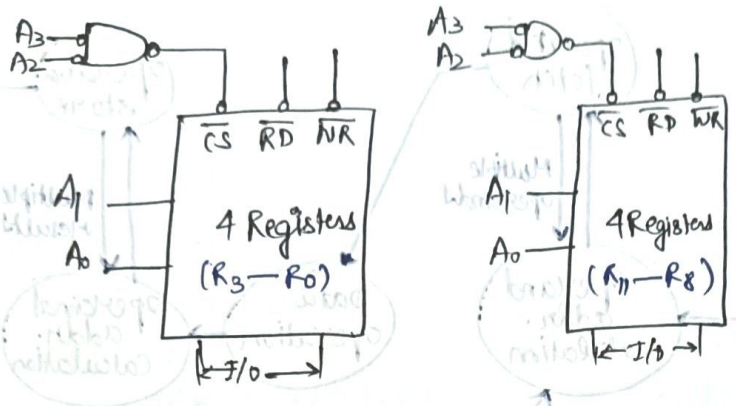
2 memory chips with 4 registers each and chip select



~~R/W~~ Memory-model:



Addressing 8 registers with 4 address lines :

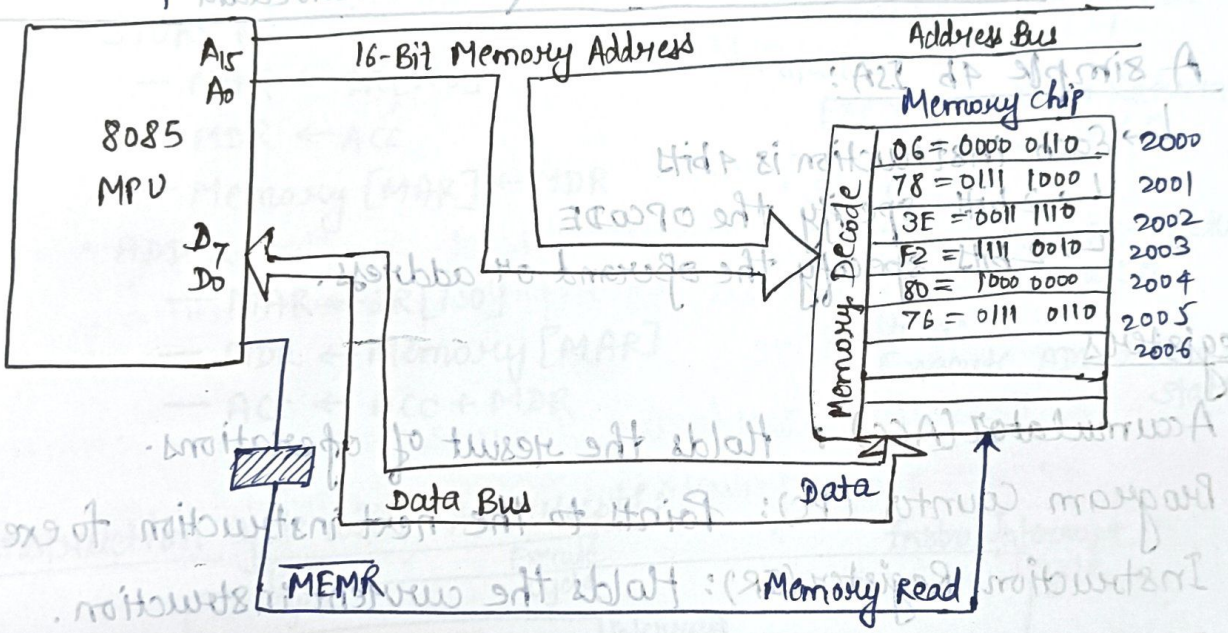


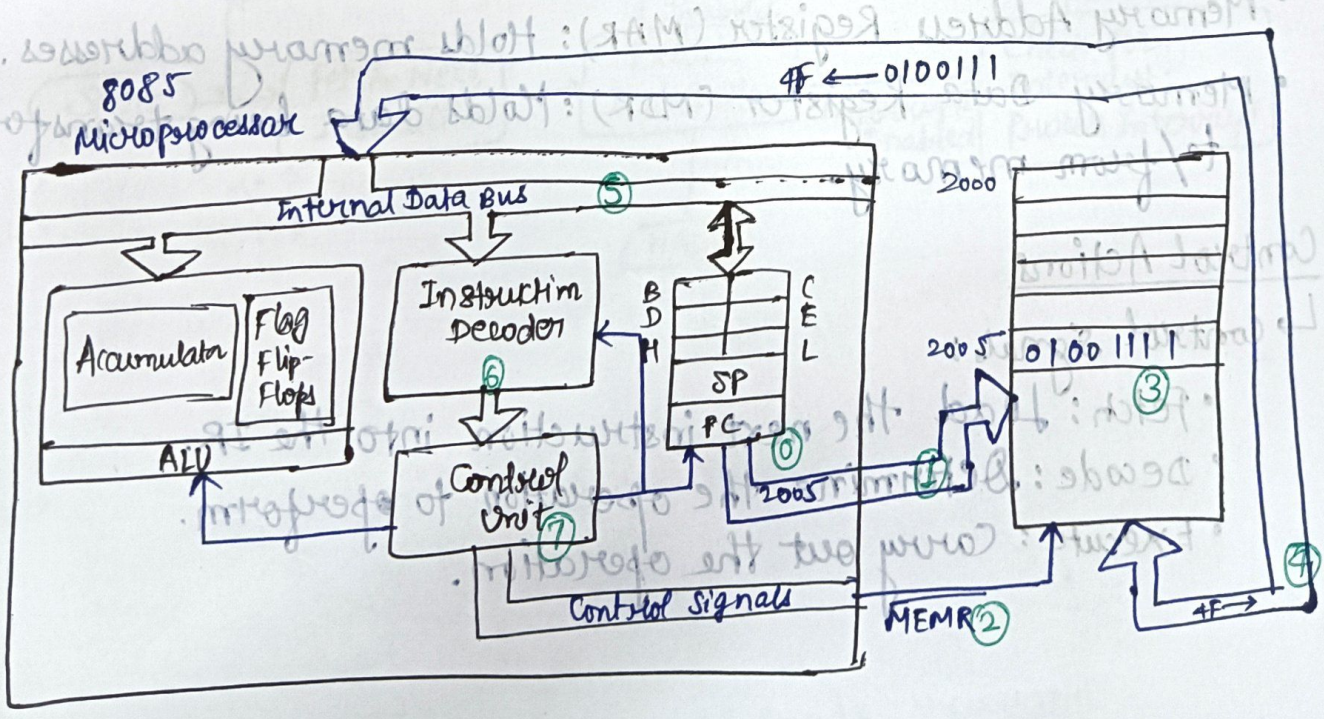
1000 — 1011
(0x20 — 0x23)
↳ (Address range supported by chip-1)

0x04 — 0x1F
↳ (Address range not used)

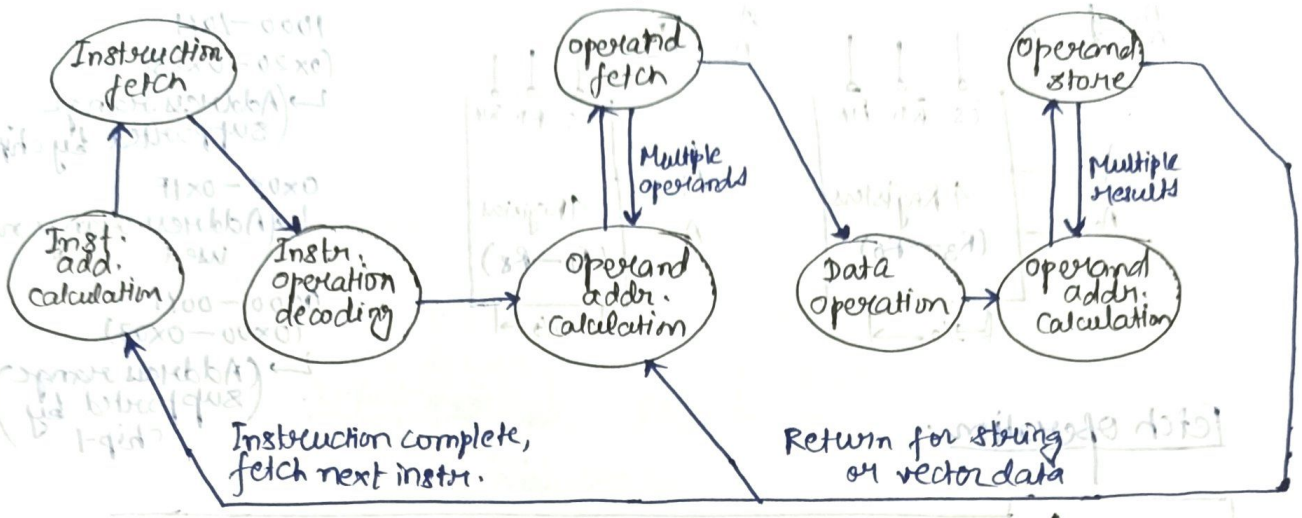
0000 — 0011
(0x00 — 0x03)
↳ (Address range supported by chip-1)

Fetch Operation :





Instruction Cycle State Diagram



A simple 4b ISA:

↳ Each instruction is 4 bits
  ↳ 2 bits specify the OPCODE
  ↳ 2 bits specify the operand or address.

Registers

- Accumulator (ACC): Holds the result of operations.
- Program Counter (PC): Points to the next instruction to execute.
- Instruction Register (IR): Holds the current instruction.
- Memory Address Register (MAR): Holds memory addresses.
- Memory Data Register (MDR): Holds data being transferred to/from memory.

Control Actions

↳ Control signals:
- Fetch: Load the next instruction into the IR.
- Decode: Determine the operation to perform.
- Execute: Carry out the operation.

Fetch cycle:

① MAR ← PC
② MDR ← Memory [MAR]
③ IR ← MDR
④ PC ← PC+1

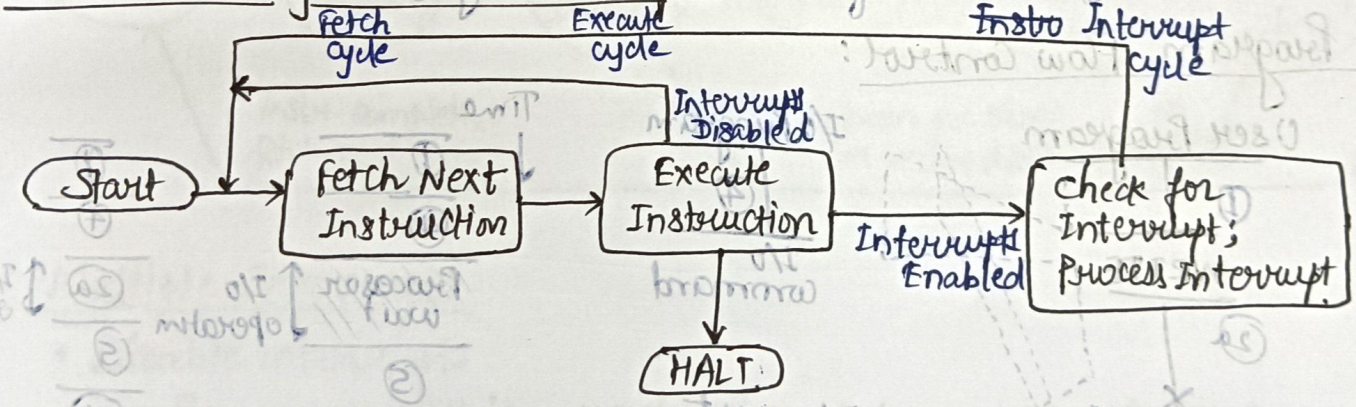
Decode & Execute cycle:

- NOP : Do nothing
- LOAD R:
  - MAR ← IR [1:0]
  - MDR ← Memory [MAR]
  - ACC ← MDR
- STORE R:
  - MAR ← IR [1:0]
  - MDR ← ACC
  - Memory [MAR] ← MDR
- ADD R:
  - MAR ← IR [1:0]
  - MDR ← Memory [MAR]
  - ACC ← ACC + MDR

State Transition Diagram:

- State 0:
  Fetch instr. → Go to state 1
- State 1:
  Decode instr. → Go to the appropriate execution state based on the opcode.
- State 2:
  Execute LOAD → Return to state 0.
- State 3:
  Execute STORE → Return to state 0.
- State 4:
  Execute ADD → Return to state 0.

Instruction cycle with Interrupts:



INTERRUPTS

↳ Mechanism by which other modules (e.g., I/O) may interrupt normal sequence of processing.

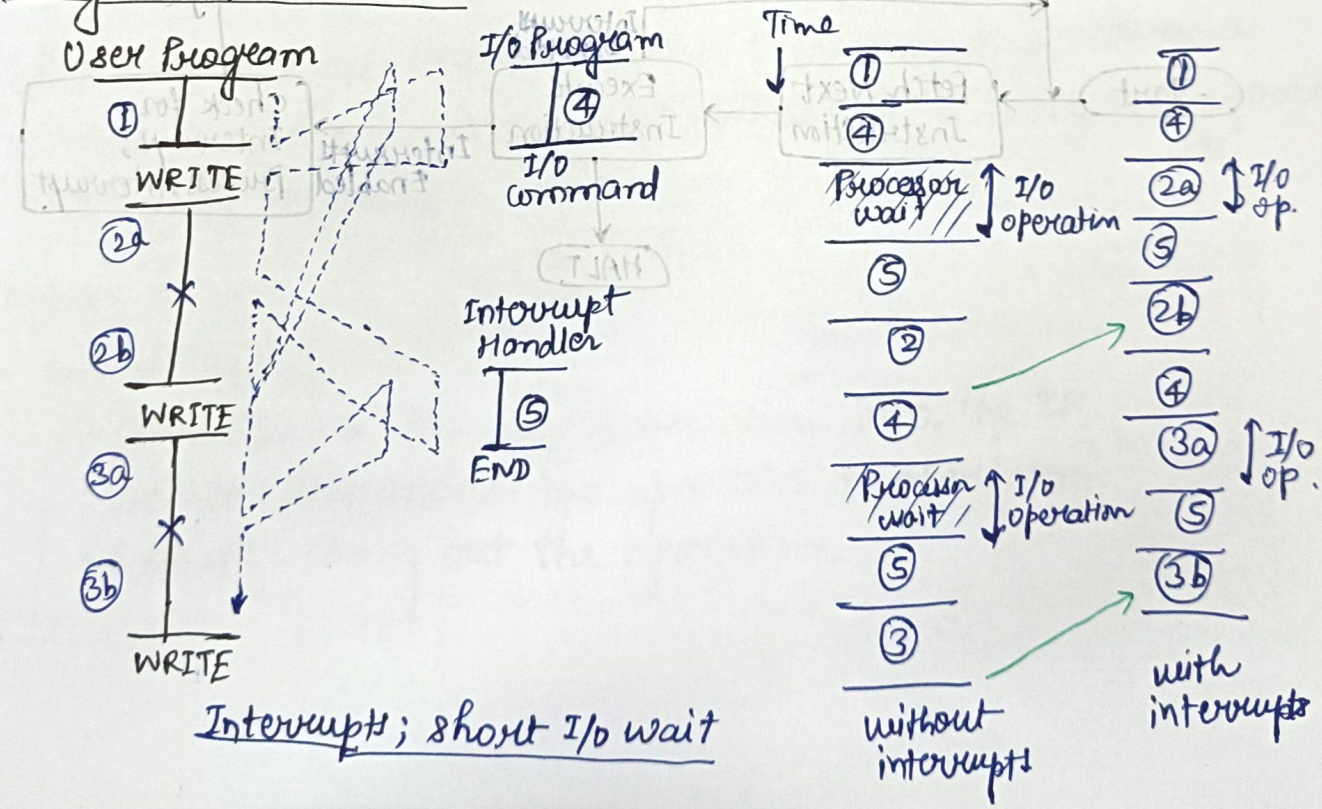
↳ Program
  ↳ e.g., overflow, division by zero
  ↳ Attempt to execute an illegal instruction.
  ↳ Referring outside the allowed memory space.

↳ Timer
  ↳ To do something at regular intervals.
  ↳ Generated by internal processor timer.
  ↳ Used in pre-emptive multi-tasking.

↳ I/O
  ↳ from I/O controller.
  ↳ Intimating the computer about completion of tasks, arrival of signals, or reporting error conditions.

↳ Hardware failure
  ↳ Reporting hardware failures or similar critical events.
  ↳ Eg., Power failure or memory parity error.

Program Flow Control:



Interrupts; short I/O wait

Interrupt Cycle

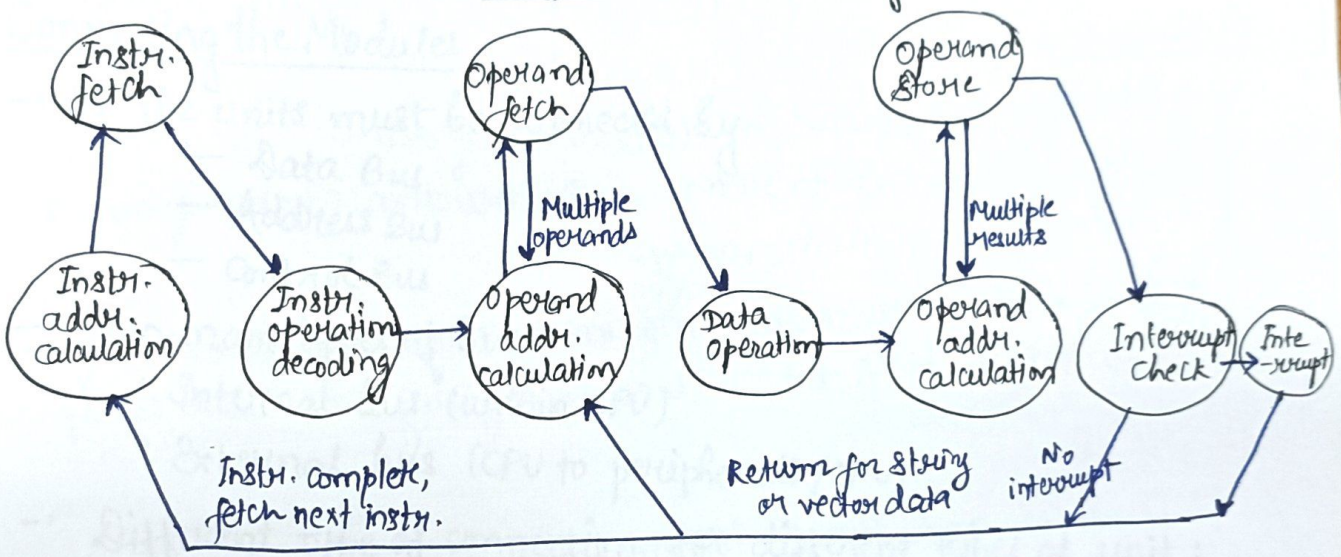
↳ Added to instruction cycle.

→ Processor checks for interrupt
  ↳ Indicated by an interrupt signal

→ If no interrupt, fetch next instruction.

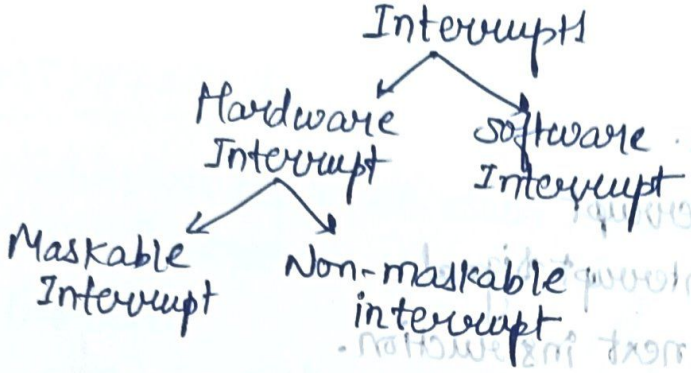
→ If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to set address of interrupt handler routine
  - Process interrupt.
  - Restore context and continue interrupted program.

Instruction Cycle (with Interrupts) — State Diagram

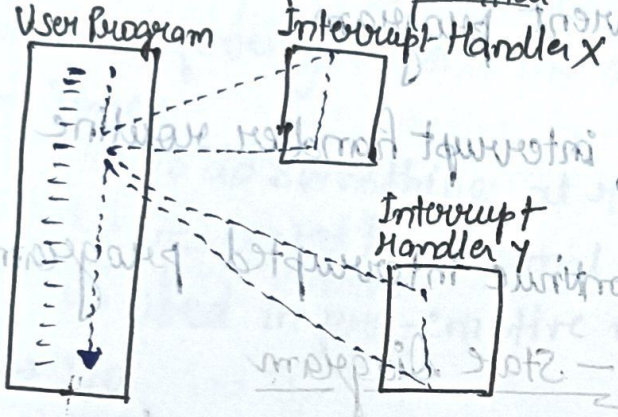


Multiple Interrupts

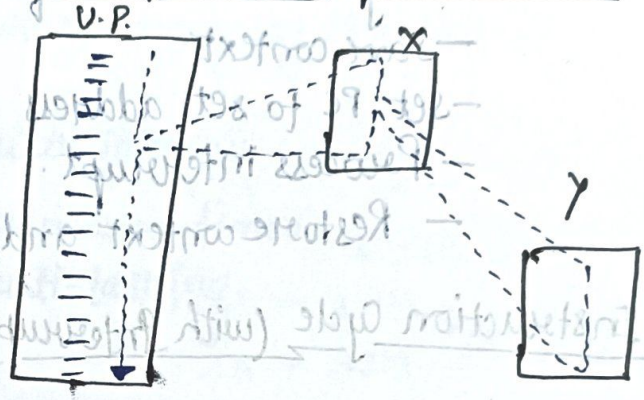
- Disable interrupts
  ↳ Processor will ignore further interrupts whilst processing one interrupt.
  ↳ Interrupts remain pending and are checked after first interrupt has been processed.
  ↳ Interrupts handled in sequence as they occur.
- Define priorities
  ↳ Low priority interrupts can be interrupted by higher priority interrupts.
  ↳ When higher priority interrupt has been processed, processor returns to previous interrupt.

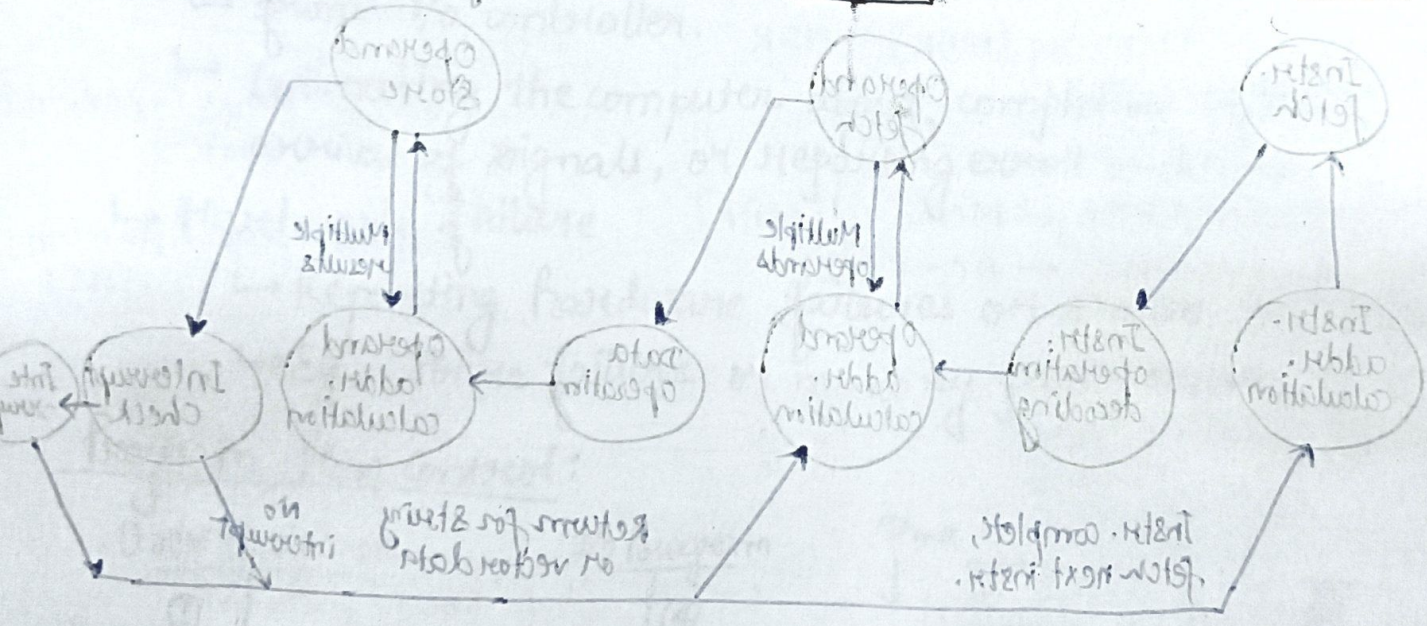


Multiple Interrupts - Sequential



Multiple Interrupts - Nested





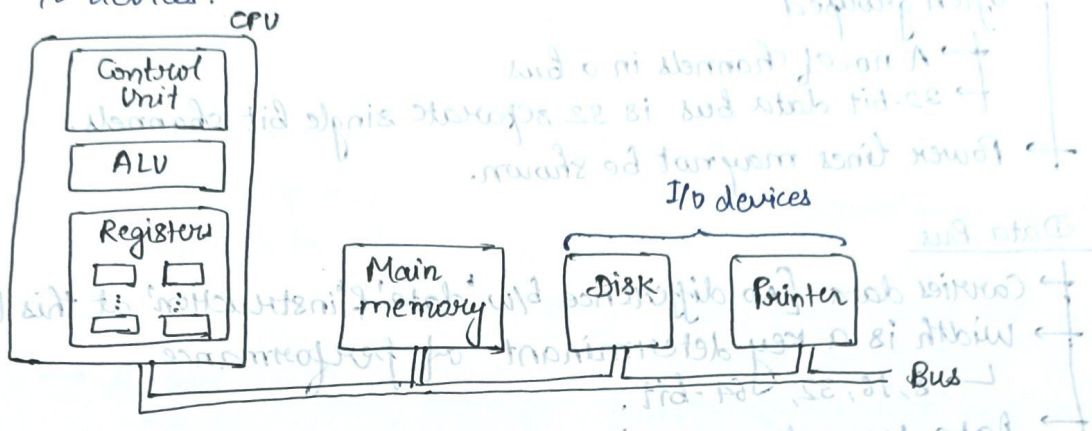
Multiple Interrupts

- Disable interrupts
  - ↳ Processor will ignore further interrupts while processing one interrupt.
  - ↳ Interrupts remain pending and are checked after first interrupt has been processed.
  - ↳ Interrupts handled in sequence as they occur.
- Define priorities
  - ↳ Low priority interrupts can be interrupted by higher

Lecture-10

PROCESSORS AND INTERCONNECTIONS

- The organization of a simple computer with one CPU and two I/O devices.

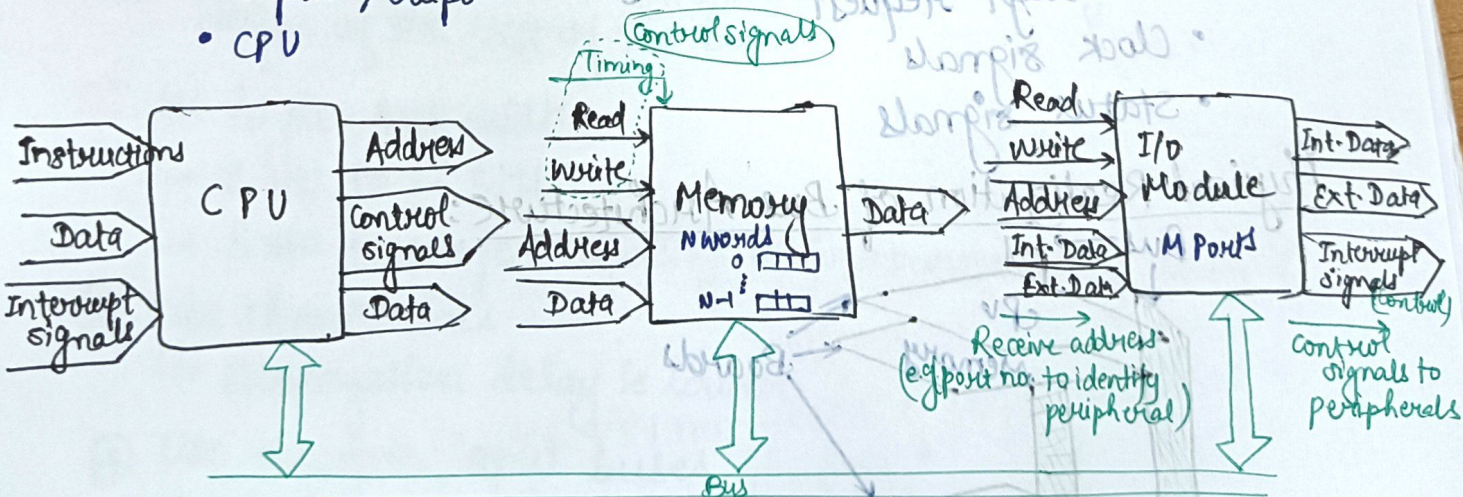


Connecting the Modules

→ All the units must be connected by
- Data Bus
- Address Bus
- Control Bus

→ Two main types of bus:
- Internal bus (within CPU)
- External bus (CPU to peripherals / CPU)

→ Different type of connection for different types of unit:
- Memory
- Input / Output
- CPU



Bus → Parallel lines on circuit boards
→ Ribbon cables
→ set of wires
→ Strip connectors on mother boards (e.g., PCI)

Bus

- A communication pathway connecting two or more devices.
- Usually broadcast
- often grouped
  - A no. of channels in a bus
  - 32-bit data bus is 32 separate single bit channels.
- Power lines may not be shown.

Data Bus

- Carries data [No difference b/w 'data' & 'instruction' at this level]
- Width is a key determinant of performance.
  - 8, 16, 32, 64-bit.
- Data transfer capacity is determined by the Databus.

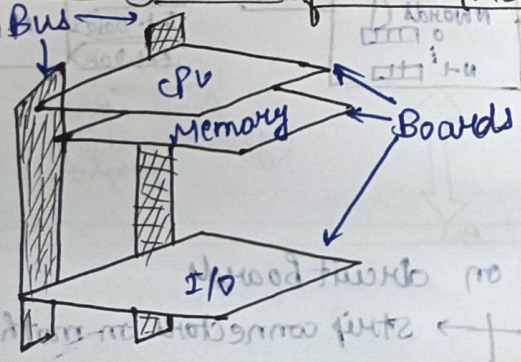
Address Bus

- Identify the source or destination of data.
  e.g., CPU needs to read an instruction (data) from a given location in memory.
- Bus width determines maximum memory capacity of system.
  e.g., 8080 has 16-bit address bus giving 64K address space.

Control Bus

- Control and timing information
  - Memory read/write signal
  - Interrupt request
  - Clock signals
  - Status signals

Physical Realization of Bus Architecture:



Single Bus Problems

- Lot of devices (I/O and memory) on one bus leads to
  - Propagation delay
- Long data paths mean that co-ordination of bus use can adversely affect performance.
- Clock period for a 1 GHz clock frequency = 1/1 GHz = 1 ns
  - 1m long bus can cause 5 ns delay @ 2×10^8 m/s signal speed
  - If bus length is 5cm, pd = 0.25 ns
    - enough to delay a full clock for a 4 GHz processor clock.
- Aggregate data transfer capacity approaches bus capacity.

Data Transfer Capacity of Data Bus

- Capacity limit : Max data transfer capacity of a bus.
- Data transfer capacity of bus : $D = W \times (H/C)$
  - W : width of bus (word length) [W bits]
  - H : clock speed of bus [H Hz]
  - C : cycles required per word cycle [C cycle]

Increasing Bus capacity

1. Increase clock frequency
   - Data transfer capacity can be increased proportionally.
   - cannot be increased due to propagation delays, as well as design of the rest of the bus.
2. Use higher bus width
   - 8 bits to 16 bits
   - Data transfer capacity is proportionately doubled.
3. Use shorter bus
   - Propagation delay is lower.
4. Use multiple / split buses
   - Each bus is shorter → Low propagation delay.
   - collective capacity is higher → Traffic localization.

Bus Design Elements

① Bus classification into various types.

② Based on the character of serial/parallel.
- ↳ Bus can use one line (serial) or multiple lines (parallel).
  - ↳ USB is a serial bus.
  - ↳ PCI is a parallel bus.

③ Based on information carried in a bus.
- ↳ Address, Data or Control Bus.

④ Based on multiplexing character.
- ↳ Dedicated vs. shared (or multiplexed)

⑤ Based on arbitration requirement.
- ↳ Centralized or Distributed (arbitration)

⑥ Based on synchronization requirement
- ↳ Synchronous or Asynchronous

⑦ Based on Data transfer type permitted.
- ↳ Read, write
- ↳ Read-modify-write
- ↳ Read-after-write
- ↳ Block data transfer.

Lecture-11

Dedicated
- ↳ Permanently assigned to one function or to a subsystem.
- ↳ Separate data and address lines.

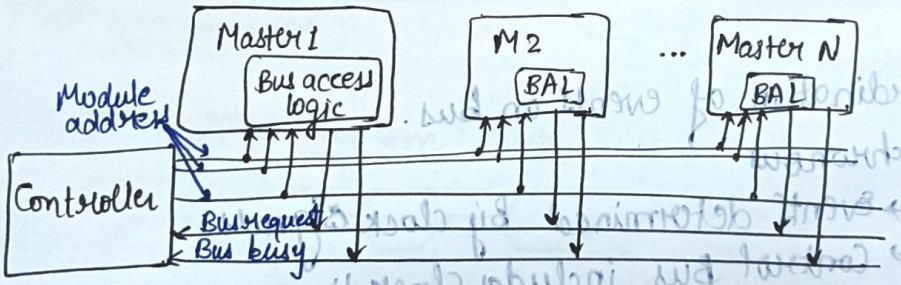
Multiplexed
- ↳ Time multiplexed
- ↳ Shared lines
- ↳ Address line or data valid control line
- ↳ Advantage: Fewer lines and pins
- ↳ Disadvantages:
  - More complex control
  - Performance limit.

Bus Arbitration

- Only one module may control bus at one time.
- More than one module controlling the bus
  - eg., CPU and DMA controller.
- Arbitration helps in choosing the module (bus master).
- Arbitration
  - Decision on who gets the bus next.
  - Who will be the bus master (who holds the bus)?
    - Bus master can initiate a data transfer (eg., read or write) with some other device.
  - May be centralised or distributed.

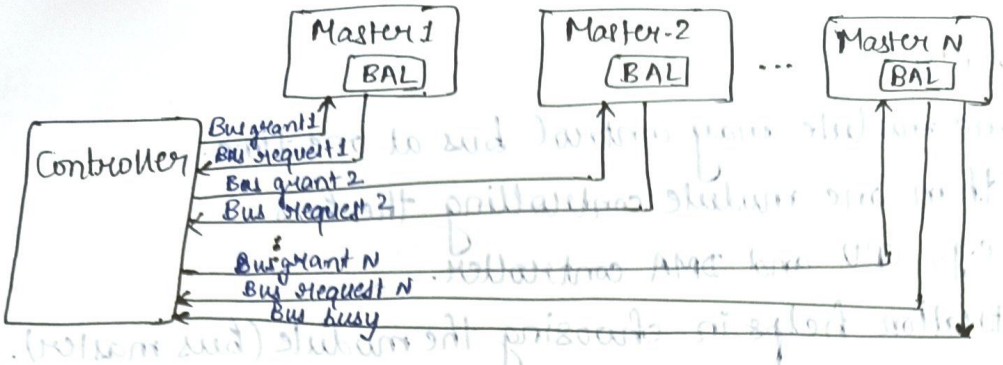
Centralised Arbitration:

- Single hardware device controlling bus access.
  - Bus controller
  - Arbiter
  - Special arbiter chip collects requests, decides.
  - Ensures fairness, but arbiter chip may itself become bottleneck.
- May be part of CPU or separate.
- Polling vs. individually addressed.



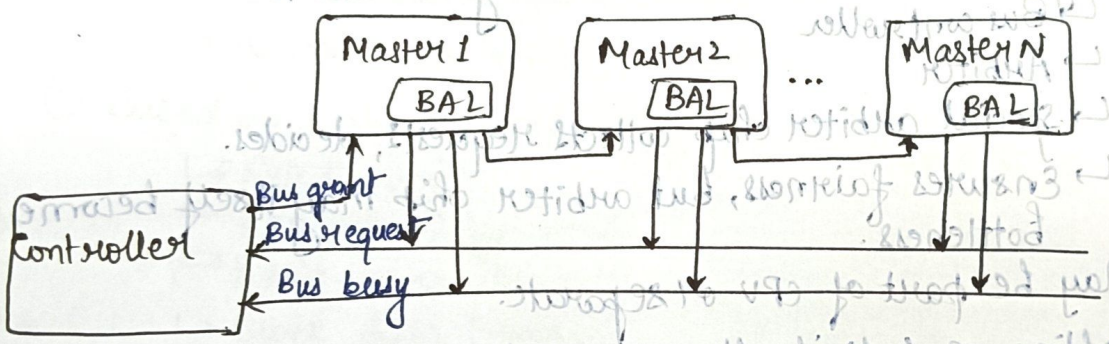
Distributed Arbitration:

- Each module may claim the bus.
- Control logic on all modules.
- Back off and retry if not the highest priority request.
- No bottlenecks and fair, but needs a lot of control lines.



Daisy-chain

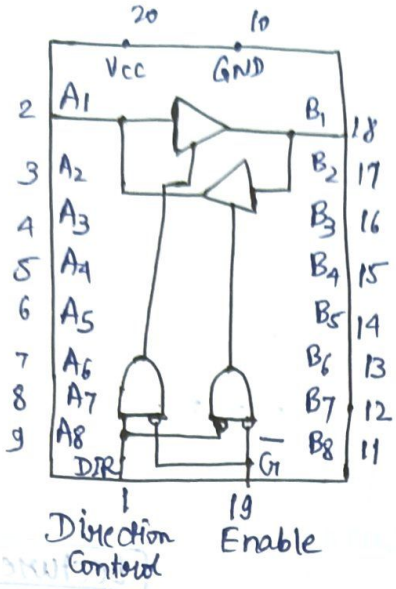
↳ Devices connect to bus in a certain predefined priority order.
↳ High-priority devices interrupt/deny requests by low-priority ones.
↳ Simple, bus slow and can't ensure fairness.
↳ A small centralized arbiter to service a large no. of devices that use a shared resource at a lower cost.



Timing

↳ Coordination of events on bus.
↳ Synchronous
  ↳ Events determined by clock signals.
  ↳ Control Bus includes clock line.
  ↳ A single 1-0 is a bus cycle.
  ↳ All devices can read clock line.
  ↳ Usually sync on leading edge.
  ↳ Usually a single cycle for an event.

Bi-directional Buffer

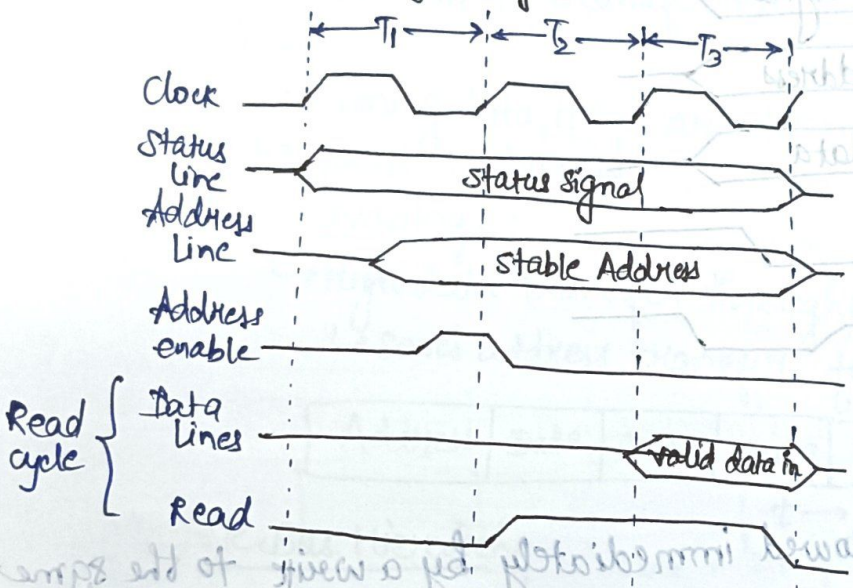


(74LS245)

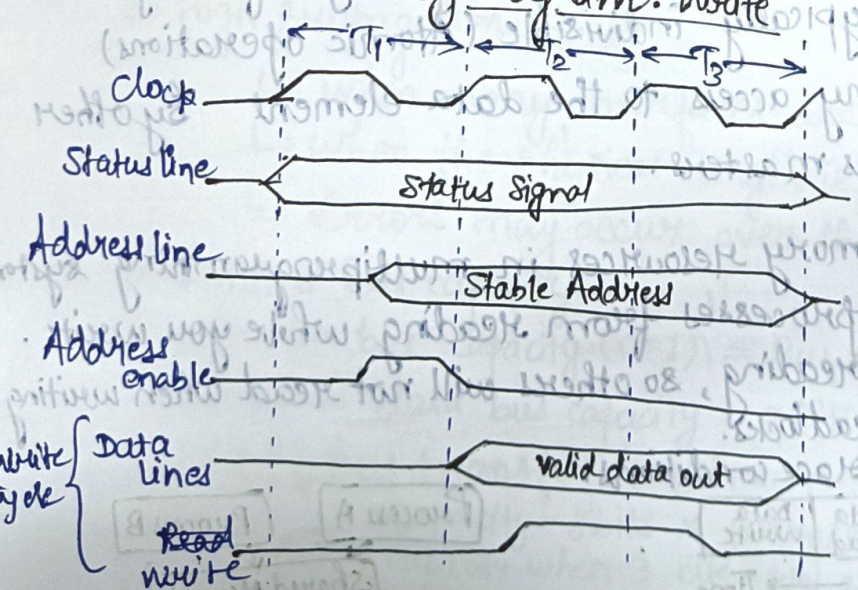
Function Table :

Enable $\overline{G_1}$	Dirn control DIR	Operation
Low level → L	L	B Data to A Bus
High level → L	H	A Data to B Bus
H	X ↑ Irrelevant	Isolation

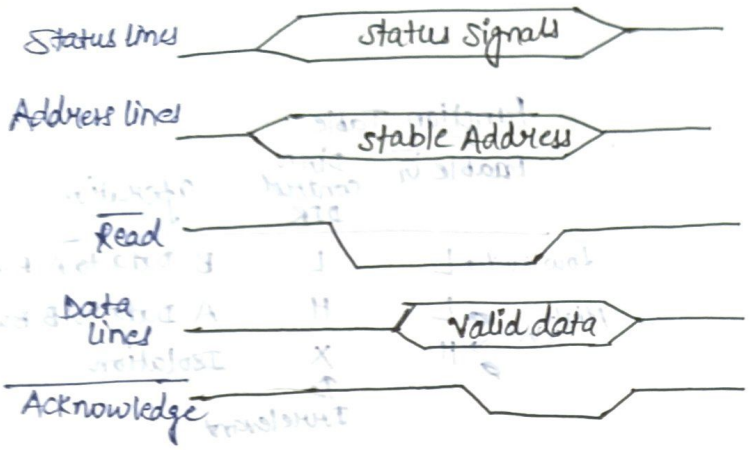
Synchronous Timing Diagram : Read



Synchronous Timing Diagram : Write

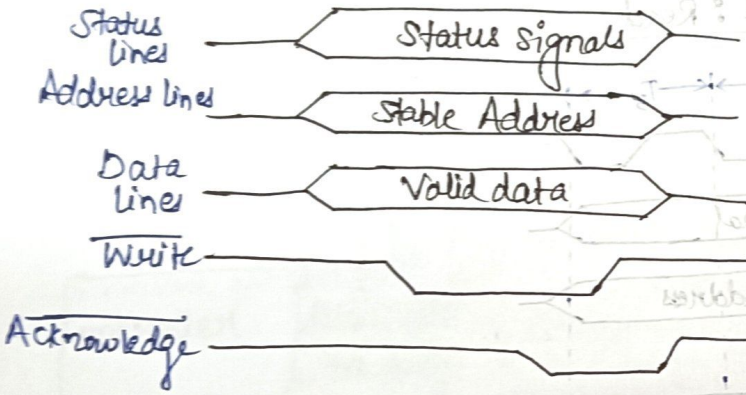


Asynchronous Timing - Read Diagram



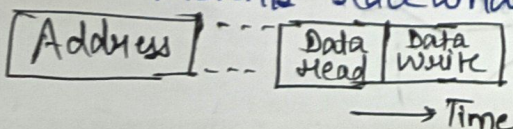
Lecture-13

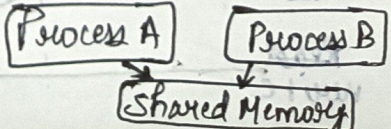
Asynchronous Timing - Write Diagram



Read-Modify-Write

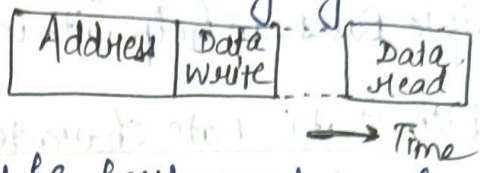
↳ Simply a read followed immediately by a write to the same address.
↳ Address is only broadcast once at the beginning of the operation.
↳ Whole operation is typically indivisible. (Atomic operations)
  ↳ To prevent any access to the data element by other potential bus masters.
↳ Protects shared memory resources in multiprogramming system
  ↳ Prevents other processes from reading while you write.
    ↳ Lock it for reading, so others will not read when writing.
    ↳ Prevents deadlocks.
    ↳ Prevents race conditions.





Read-After Write Bus

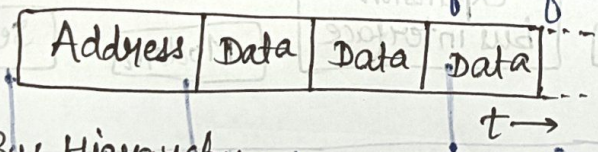
↳ An indivisible / atomic operation.
↳ Address is only given once.
↳ Consisting of a write followed immediately by a read from the same address.



Main purpose:
↳ The read operation may be performed for checking purposes.
↳ Whether the data is written correctly.
↳ Memory cell errors can be present.
↳ Data Read must be same as Data written.

Block Data Transfer Bus

↳ One address cycle is followed by N data cycles.
↳ First data item is transferred to or from the specified address.
↳ The remaining data items are transferred to or from subsequent addresses.
↳ **Main advantage:**
↳ High data transfer throughput
↳ saves address transfer for ~~ses~~ subsequent reads/writes.



Multiple Bus Hierarchy

↳ Single buses face many issues.

① High propagation delay
↳ when many interfaces are connected.
↳ when the physical length is high.
↳ Errors may occur over the bus.

② Limited bus capacity
↳ Max bus capacity (C=1) = Bus width × clock speed
↳ 8085 data bus capacity = 8 bits × 3 MHz = 24 Mbps
↳ If bus transfer requires K clock cycles.
↳ capacity = 8 bits × (3 MHz / K)
∴ ↳ capacity when 3 clk cycles are required for a data transfer: 8 Mbps

↳ I/O or memory devices faster than these cannot be supported.

③ Increasing the capacity
  ↳ Increase bus width
  ↳ Increase clock frequency.

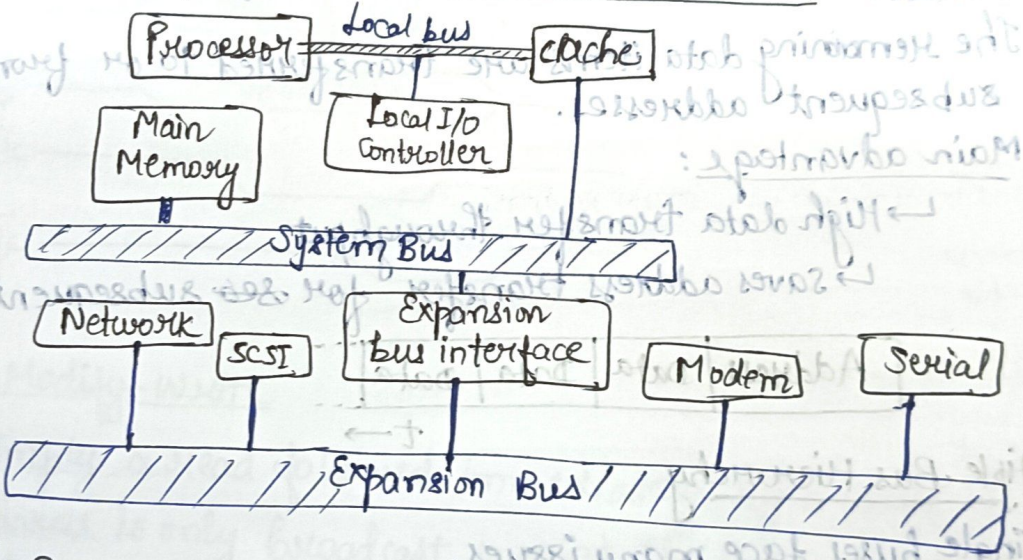
④ Multiple Buses can help with increasing capacity.

Bandwidth/Data Rate characteristics of I/O devices

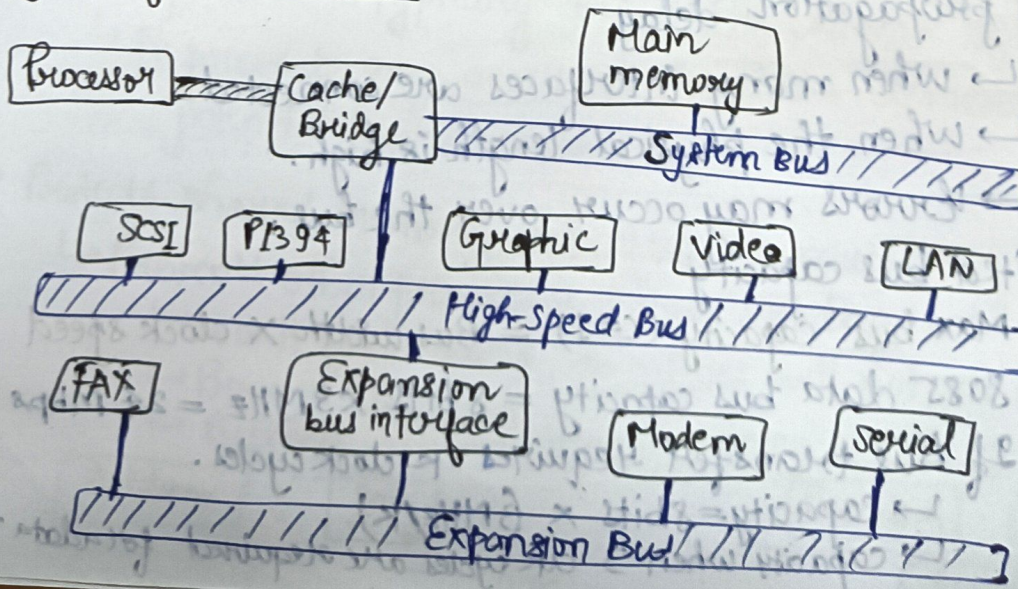
① Primary characteristic : Data rate or B/W.

② Contributing factors:
  ↳ Partner: Humans have slower input/output data rates than machines.
  ↳ Input or output or both (input/output).

An Eg. of Multiple Bus Architecture (with cache)



High Performance Bus



Lecture-14

Some Popular Bus Standards

- Industry Standards Architecture (ISA) Bus
  - ↳ Extended to Extended ISA (EISA) bus. (16.7 MB/s)
- PCI Bus
  - ↳ Limited data rate
    - ↳ 135 MB/s required for video interface
- USB Bus
- PCMCIA (Personal Computer Memory Card International Association) Bus
  - ↳ Express card Bus
- Firewire (IEEE 1394)
- Serial ATA (Serial AT Attachment)
- SCSI (Small Computer System Interface).

PCI (Peripheral Component Interconnection) Bus → 133 MB/s (original)

↓

PCI 2.1 → 528 MB/sec.

MEMORY

↳ The internal or external module of CPU or I/O subsystem in a computer system where data is stored by a computer temporarily or permanently.

Memory characteristics:

① Location
- CPU
- Internal
- External

② Capacity
- ↳ word size [natural unit of org.]
- ↳ No. of words (or bytes).

③ Unit of transfer
- ↳ Internal → usually governed by data bus width.
- ↳ External → usually a block which is much larger than a word.
- ↳ Addressable unit
  - ↳ Smallest location which can be uniquely addressed.
  - ↳ word internally.
  - ↳ Cluster of Memory disks.

④ Access Method ⑤ Physical type ⑥ Organisation.
⑦ Performance ⑧ Physical characteristics

Access Methods

① Sequential
- ↳ Start at the beginning and read through in order.
- ↳ Access time depends on location of data and previous location.
- ↳ High volume storage
- ↳ Archival storage
- ↳ Low cost of storage (e.g., tape).

② Direct
- ↳ Individual blocks have unique address.
- ↳ Access is by jumping to vicinity plus sequential search.
- ↳ Access time depends on current and previous location (e.g., disk).

③ Random
- ↳ Individual addresses identify locations exactly.
- ↳ Access time is independent of location or previous access (e.g., RAM).

④ Associative
- ↳ Data is located by a comparison with contents of a portion of the store.
- ↳ Access time is independent of location or previous access (e.g, cache).

Lecture-15

Performance

① Access time
- ↳ time b/w presenting the address and getting the valid data.

② Memory cycle time
- ↳ time may be required for the memory to "recover" before next access.
- ↳ cycle time is access + recovery.

③ Transfer Rate : Rate at which data can be moved

For Non-Random access memory :

$T_N = T_A + n/R$, T_N: av. time to read or write N bits
T_A : av. access time
n : no. of bits
R : transfer rate in bps

For Random-access memory :

$T_N = \frac{N}{\text{cycle time}}$

Physical Types

① Semiconductor — RAM
② Magnetic — Disk & Tape
③ Optical — CD & DVD
④ Others — Hologram, Quantum States

Physical Characteristics

① Volatility
↳ Data loses when power supply is removed.
↳ Magnetic surface memories are non-volatile (decay possibility exists)
↳ Semi-conductor memories
↳ volatile
↳ Non-volatile
↳ Flash memory uses an electrically erasable programmable read-only (EEPROM)

② Erasable
↳ Non-erasable memory can only be destroyed by physical elimination.

③ Decay
↳ Slow decline of the stored contents over a period of time.

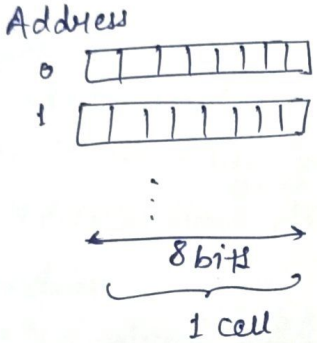
④ Bit-rotting
↳ A new term identified for representing the loss of world's information stored in various memory media.
↳ Eg. no hardware readers available ↳ Data format is obsolete.
↳ Data format is not known
↳ Encoding is not know

Memory Organization

↳ Physical arrangement of bits into words.
  ↳ Not always obvious.
  ↳ Big endian vs. little endian.

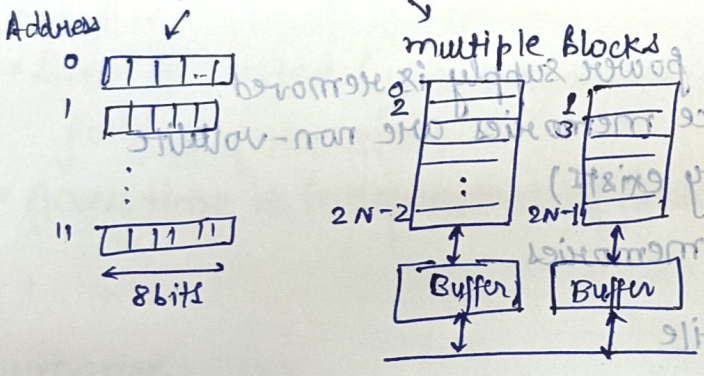
↳ Dilemma of a memory system designer

access time ↑ (fast) | cost per bit ↑ | capacity ↓ → Tradeoff



↳ Organizing words within memory:

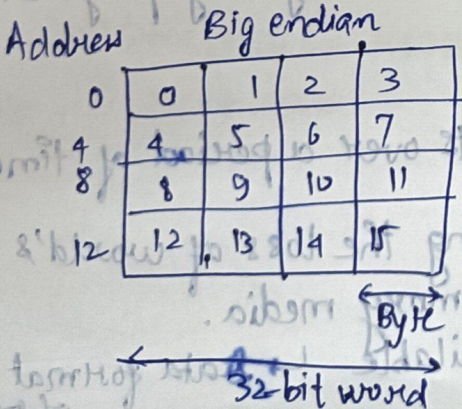
Blocks vs interleaved memory → multiple Blocks

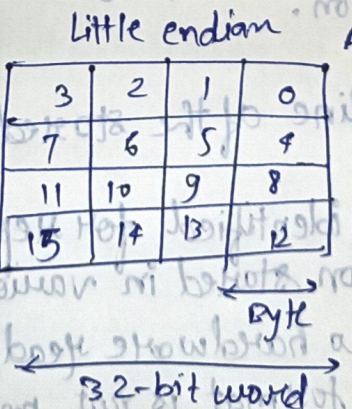


Byte ordering

ⓐ Big Endian memory — SPARC, IBM → MSB stored at lowest memory address

ⓑ Little Endian memory — Intel family → LSB stored at lowest memory address.





Memory Hierarchy

① Registers
  ↳ In CPU

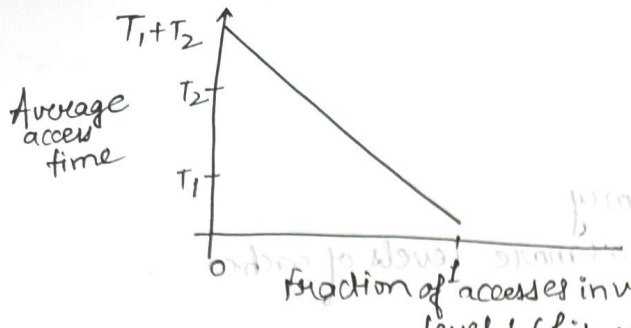
② Internal or Main memory
  ↳ May include one or more levels of cache
  ↳ "RAM"

③ External memory
  ↳ Backing store

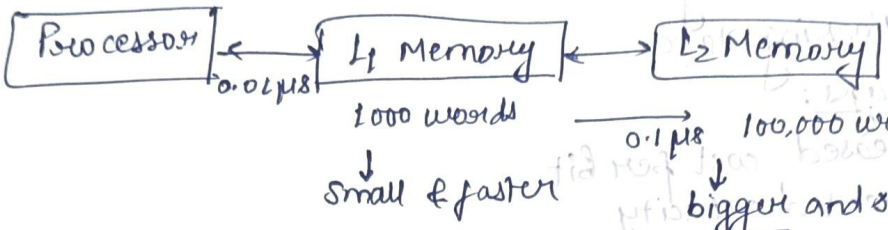
↳ Advantages:
- Decreased cost per bit
- Increased capacity
- Decreased frequency of access of memory by the processor.
- Average decreased access time.

↳ Disadvantages:
- Increased access time for down the hierarchy.
- Increased hardware complexity.

Eg. Suppose that the processor has access to two levels of memory. Level 1 contains 1000 words and has an access time of 0.01 μs; level 2 contains 100,000 words and has an access time of 0.1 μs. Assume that if a word has to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, the the word is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the time required for the processor to determine whether the word is in level 1 or level 2. Figure shows the general shape of the curve that covers this situation; shows the average access time to a two-level memory as a function of the hit ratio H, where H is defined as the fraction of all memory accesses that are found in the faster memory (e.g., the cache). T_1 is the access time to level 1, T_2 to level 2. As can be seen, for high %age of level 1 access, the average total access time is much closer to that of level 1 than that of level 2.



Eg:



Suppose 95% of the memory accesses are found in the cache. Then, avg. time to access word :

$(0.95)(0.01\ \mu s) + (0.05)(0.01 + 0.1)\ \mu s$

$= 0.095 + 0.0055$

$= 0.015\ \mu s$ ⇝ much closer to 0.01 μs than to 0.1 μs, as desired.

If Hit Ratio is 50% :

$(0.5)\ 0.01\ \mu s + (0.5)(0.1\ \mu s + 0.01\ \mu s)$

$= 0.06\ \mu s$

Hierarchy List :

- Registers
- L_1 cache
- L_2 cache
- L_3 cache
- Main memory
- Disk cache
- Disk storage
- Optical storage
- Tape storage

Locality of Reference

↳ During the course of the execution of a program, memory references tend to cluster.

↳ Eg loops

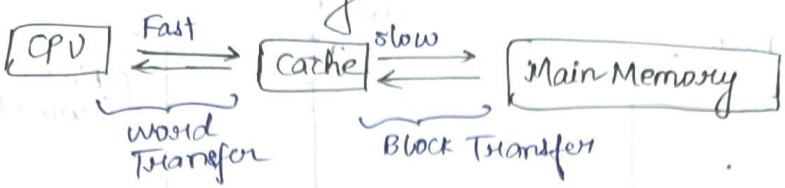
Cache → Small amount of fast memory

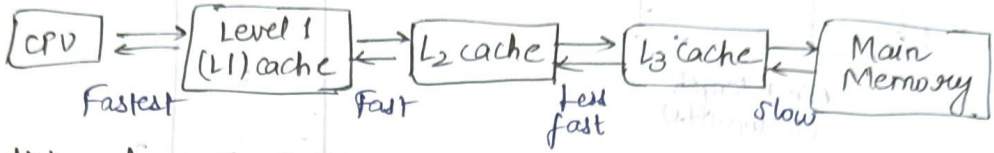
↳ Sits b/w normal main memory and CPU

↳ May be located on CPU chip or module.

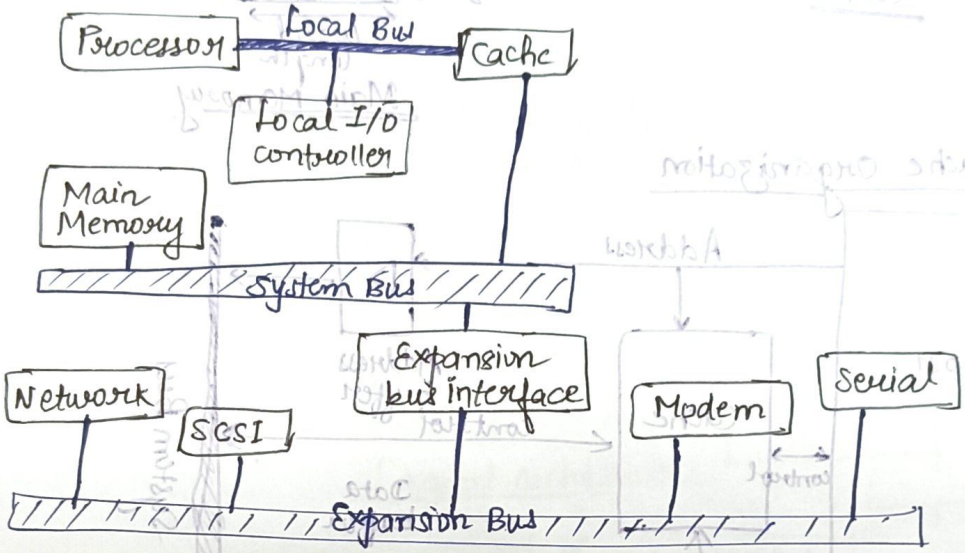
Lecture-16

Cache and Main Memory





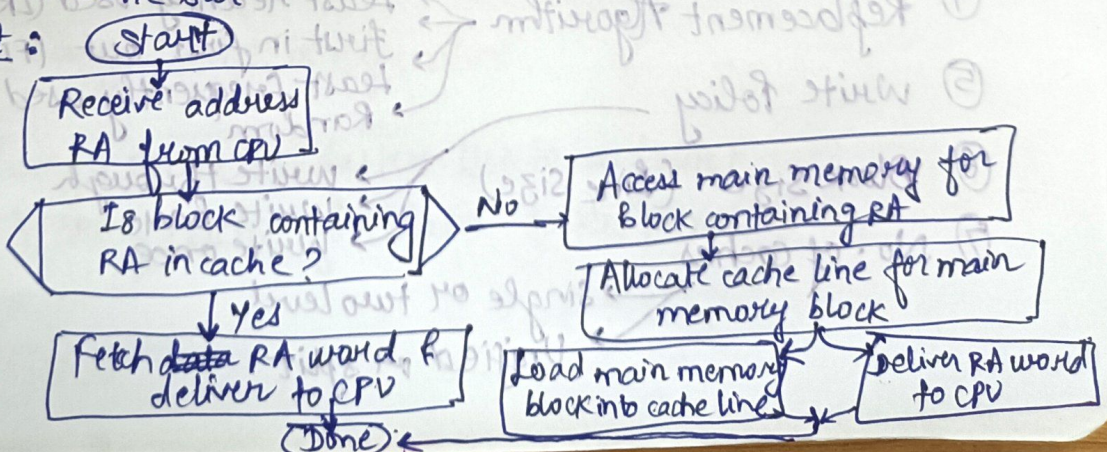
Traditional ~~Cache~~ (ISA) (with cache):



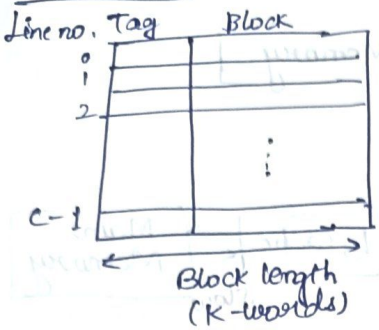
Cache Operation

→ CPU requests contents of memory location.
→ Check cache for this data.
→ If present, get from cache (fast)
→ If not present, read required block from main memory to cache, then deliver from cache to CPU.
→ Cache includes tags to identify which block of main memory is in each cache slot.

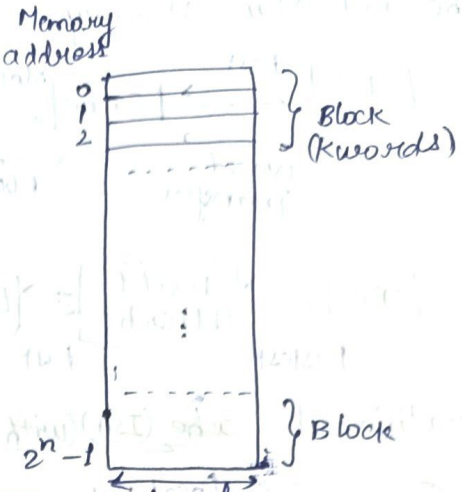
Flowchart:



Structure:

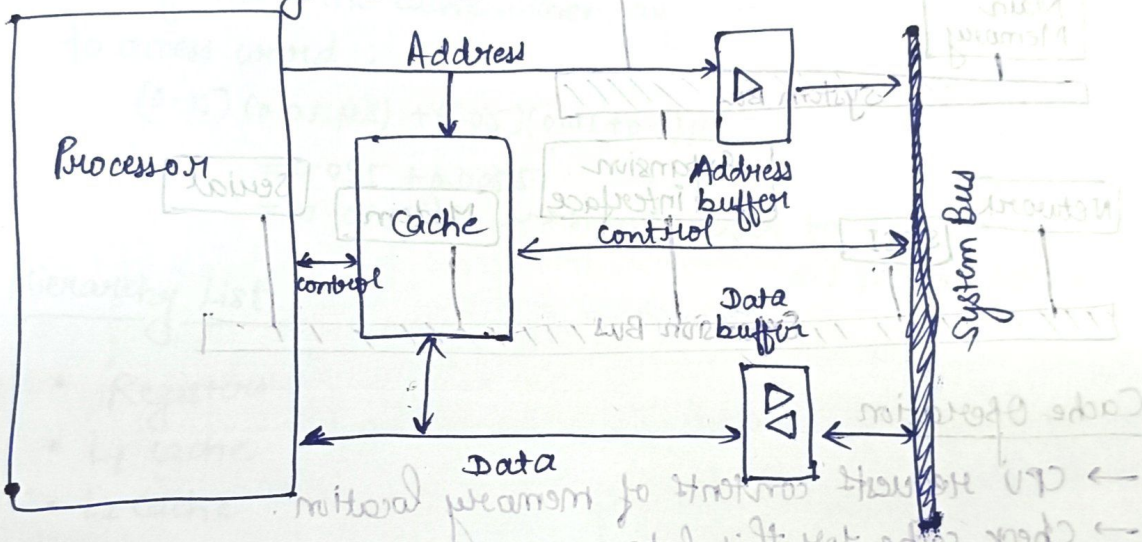


Cache



Main Memory

Typical Cache Organization

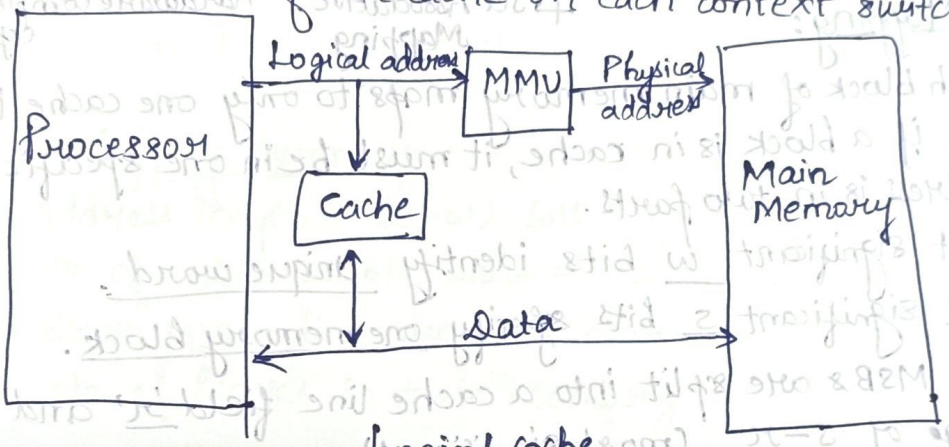


Cache Design

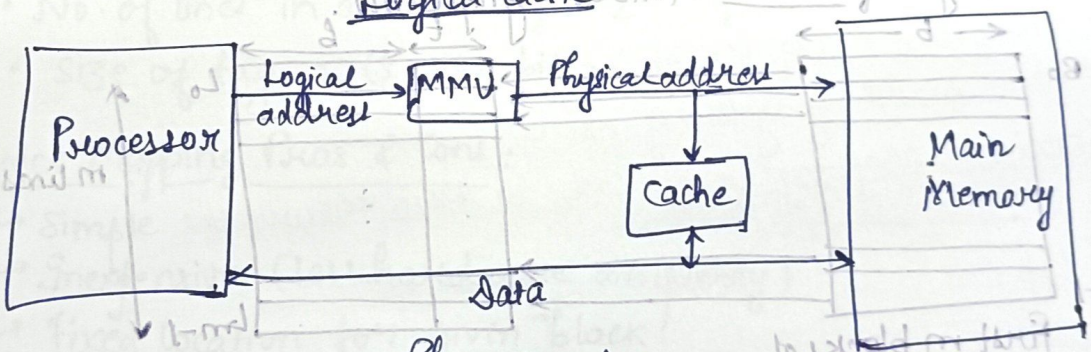
1. Addressing → Logical, Physical
2. Size
3. Mapping function → Direct, Associative, Set Associative
4. Replacement Algorithm → Least recently used (LRU), First in first out (FIFO), Least frequently used (LFU), Random
5. Write Policy → Write through, Write back, Write once
6. Block Size (Line Size)
7. No. of caches → Single or two level, Unified or split

Cache Addressing (Logical and Physical)

- Where does cache sit?
  - ↳ B/w processor and virtual memory management unit (MMU)
  - ↳ B/w MMU and main memory.
- Logical cache (virtual cache) stores data using virtual addresses.
  - ↳ Processor accesses cache directly, not through physical cache.
  - ↳ Cache access faster, before MMU address translation.
  - ↳ virtual addresses use same address space for different applications.
    - ↳ Must flush cache on each context switch.



Logical cache



Physical cache

- Physical cache stores data using main memory physical addresses.

Cache Size does matter

- Cost → Large cache is expensive.
- Speed
  - ↳ Large cache is faster (upto a point)
    - ↳ checking cache for data takes time.

Mapping Function

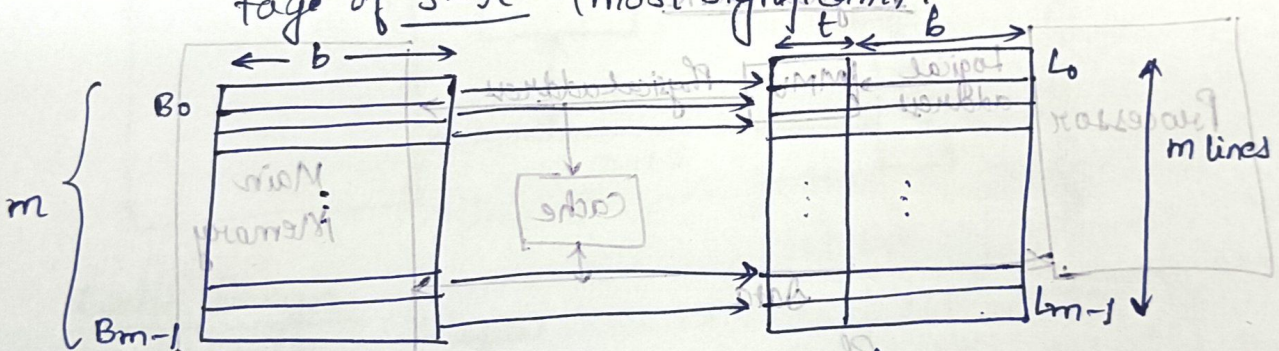
↳ Defines the method of mapping b/w main memory address & cache location.

Eg.,
- cache of 64 KBytes
- cache block of 4 bytes
  - ↳ cache is 16K (2^{14}) lines of 4 bytes
- 16 MB main memory
- 24 bit address ($2^{24} = 16M$)

↳ Direct Mapping ↔ simple / High miss rates / high penalty
↳ (Fully) Associative Mapping ↔ Improves hit rate / Hardware complexity / expensive
↳ set Associative Mapping

Direct Mapping:

↳ Each block of main memory maps to only one cache line, i.e., if a block is in cache, it must be in one specific place.
↳ Address is in two parts.
↳ Least significant w bits identify unique word.
↳ Most significant s bits specify one memory block.
↳ The MSBs are split into a cache line field 'r' and a tag of $s-r$ (most significant).

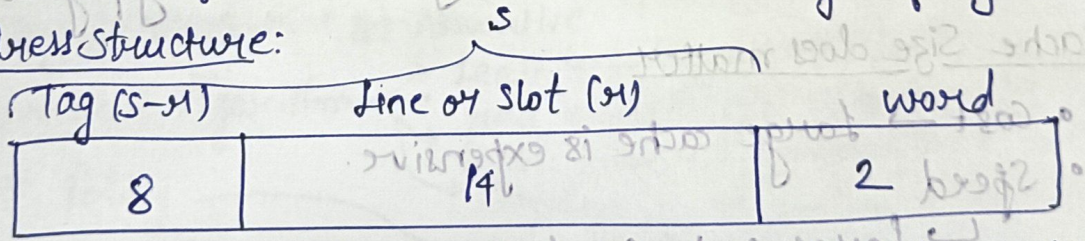


First m blocks of main memory (equal to size of cache)

Cache memory

b: length of block in bits
t: length of tag in bits.

Address Structure:



→ 24 bit address
→ 2 bit word identifier (4 byte block)
→ 22 bit block identifier
  ↳ 8 bit tag (= 22-14)
  ↳ 14 bit slot or line

→ No 2 blocks in the same line have same tag.
→ check contents of cache by finding line & checking tag.

$i = j \text{ modulo } m$

- i → Cache line no.
- j → main memory block no.
- m → no. of lines in the cache

Cache line Table:

cache line	Main memory blocks held
0	$0, m, 2m, 3m, \ldots, 2^s - m$
1	$1, m+1, 2m+1, \ldots, 2^s - m + 1$
⋮	
$m-1$	$m-1, 2m-1, 3m-1, \ldots, 2^s - 1$

Lecture-17

Direct Mapping Summary:

- Address length = $(s+w)$ bits
- No. of addressable units = $2^{(s+w)}$ words or bytes
- Block size = line size = 2^w words or bytes
- No. of blocks in main memory = $2^{(s+w)} / 2^w = 2^s$
- No. of lines in cache = $m = 2^r$
- Size of tag = $(s-r)$ bits.

Direct Mapping Pros & Cons:

↳ Simple

↳ Inexpensive (less hardware complexity)

↳ Fixed location for given block

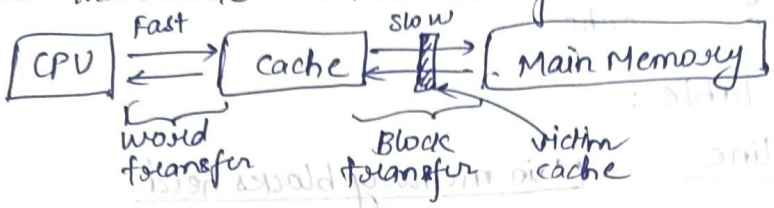
↳ If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high.

↳ Repeatedly cache miss can occur due to the execution of the same blocks of code.

↳ Cache trash.

Victim Cache

↳ smaller than cache with similar speed as the cache.

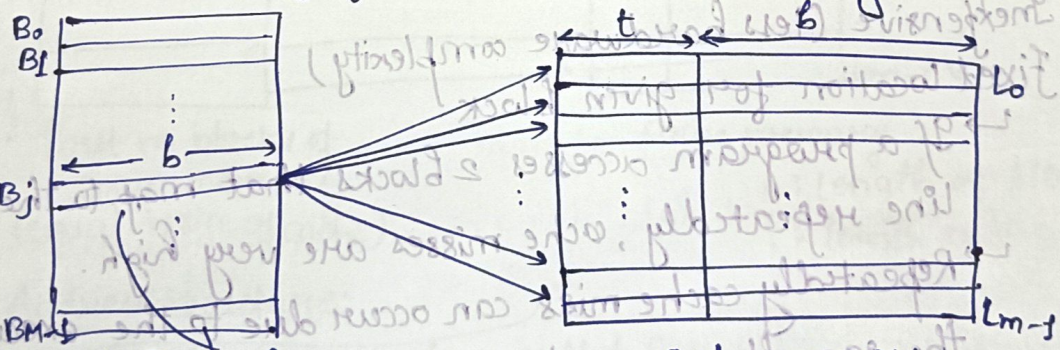


↳ Lower Miss Penalty
↳ Remember what was discarded
↳ Already fetched
↳ Use again with little penalty.
↳ Fully associative
↳ 4 to 16 cache lines
↳ B/w direct mapped L_1 cache and next memory level.

Associative Mapping

↳ A main memory block can load into any line of cache.
↳ Memory address is interpreted as tag and word.
↳ Tag uniquely identifies block of memory.
↳ Every line's tag is examined for a match.
↳ cache searching gets expensive.

Associative Mapping from cache to Main Memory:



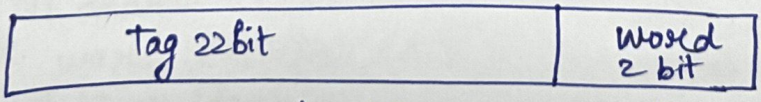
First m blocks of main memory (equal to size of cache)

one block of main memory

Cache memory

b: length of block in bits
t: length of tag in bits

Address structure:



→ 22 bit tag stored with each 32 bit block of data.
→ Compare tag field with tag entry in cache to check for hit.

→ Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block.

Summary:

- Address length = (s+w) bits
- No. of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^w words or bytes
- No. of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s$
- No. of lines in cache = undetermined
- Size of tag = s bits.

Set Associative Mapping

↳ Cache is divided into a no. of sets.
↳ Each set contains a no. of lines.
↳ A given block maps to any line in a given set.
e.g., Block B can be in any line of set i.
↳ Eg., 2 lines per set (2 way)
↳ 2-way associative mapping
↳ A given block can be in one of 2 lines in only one set.

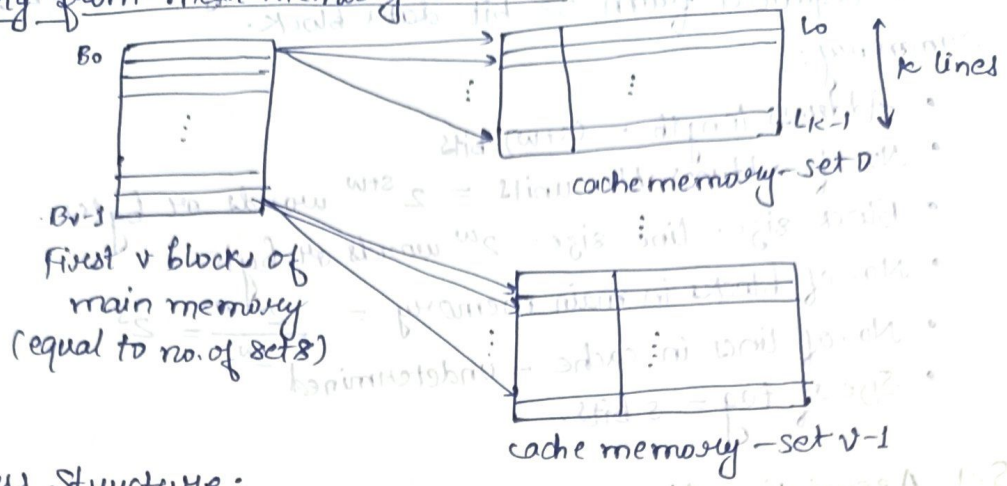
→ $m = v \times k$
→ $i = j$ modulo v

i : cache set no.
j : main memory block no.
m : no. of lines in the cache
v : no. of sets
k : no. of lines in each set
(k-way associative mapping)

→ Block Bj can be mapped to any of the lines of set j.

Eg. 13-bit set no.
Block no. in main memory is modulo 2^{13}
$i = j$ modulo v
$v = 2^{13}$
→ 000000, 00A000, 00B000, 00C000 ... map to same set.

Mapping from main memory to cache:



Address Structure:

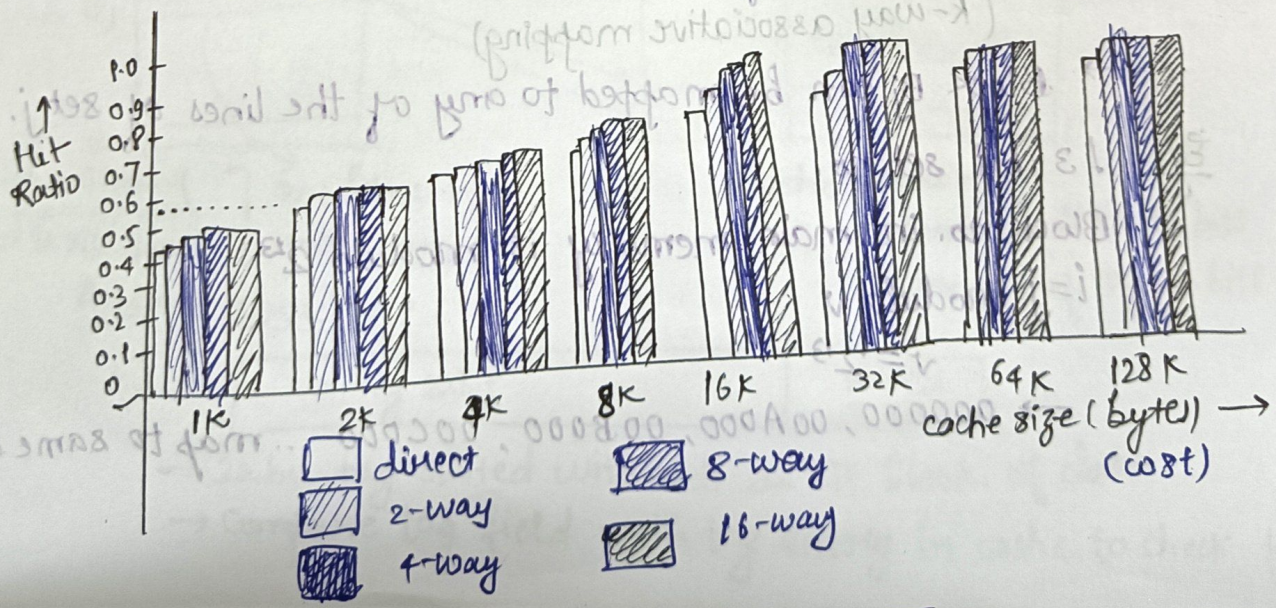
Tag 9 bit	Set 13 bit	2 bit word

→ Use set field to determine cache set to look in.
→ Compare tag field to see if we have a hit.

Summary:

- Address length = (s+w) bits
- No. of addressable units = 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
- No. of blocks in main memory = 2^d
- No. of lines in set = k
- No. of sets = $v = 2^d$
- No. of lines in cache = $kv = k * 2^d$
- No. of tag = (s-d) bits.

Lecture-18



Direct & set Associative cache Performance Differences:

→ Significant upto at least 64 KB for 2-way
→ Difference b/w 2-way and 4-way at 4KB << 4KB to 8KB.
→ Cache complexity increases with associativity.
→ Not justified against increasing cache to 8KB or 16KB.
→ Above 32 KB gives no improvement.

Locality of Reference in cache Design

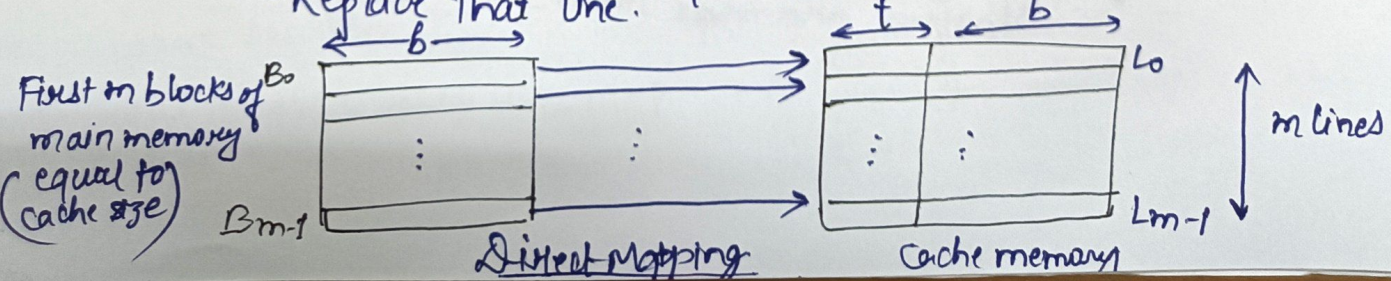
- Spatial locality
  - ↳ Memory addresses numerically similar to a recently accessed memory location are likely to be accessed in the near future.
  - ↳ Cache exploit this property.
    - ↳ by bringing in more data (block size or line size) than have been requested.
      - ↳ future requests can be anticipated.
- Temporal locality
  - ↳ occurs when recently accessed memory locations are likely accessed again.
  - ↳ Eg: • memory locations near the top of the stack.
    • instructions inside a loop.
  - ↳ Temporal locality is exploited in cache designs primarily by the choice of what to discard on a cache miss.

Replacement Algorithm

↳ Replacing the contents of the cache memory which is full when a new item is brought in.

Replacement Agorithms in Direct Mapping caches:

↳ Mapping leaves no choice for the data block to be stored.
  ↳ Each block only maps to one unique line.
  ↳ Replace that line.



Replacement Algorithms and Associative & set Associative caches:

↳ Replacement algorithms are relevant for Associative and set-Associative Mapped caches.

↳ Algorithms are implemented in hardware
  ↳ Speed of operation is very critical.

Eg. In 2 way set associative
  ↳ which of the 2 block is LRU?

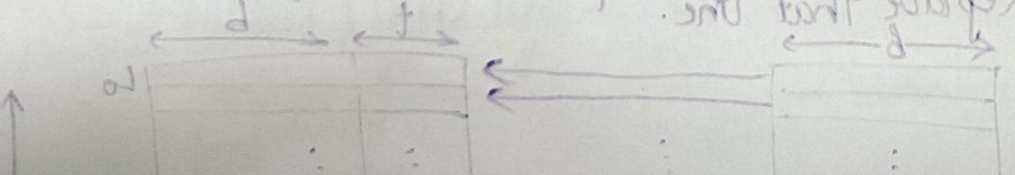
→ There are main algorithms:
- Least Recently used (LRU)
- First In First Out (FIFO)
- Most Frequently Used (MFU) / Least Frequently Used
- Random

Least Recently Used (LRU)

↳ Maintain the information about the usage of cache lines.

↳ A new memory request facing a cache miss results in memory access.
  ↳ The least recently used item is replaced.
  ↳ Most popular algorithms in cache replacement.

↳ How to keep track of the usage
  ↳ Easy to be implemented in a fully-associative cache.
  ↳ 1/0 bit for 2-way set associative caches (1 for the recently accessed item and 0 for the other item) and replace the older item.
  ↳ Maintains a separate list of indexes to all the lines in the cache.
  ↳ When an item is accessed, it is moved to the front of the list.
  ↳ Item from the end of the list is removed when a new item is cached.



First In First Out (FIFO)

- Replace block that has been in cache longest.
- First item stored in the cache is moved out when the cache is full and a new item is to be taken into the cache.
- Easy to implement using a circular or round-robin buffer.

Least Frequently Used

- Replace block which has had fewest hits.
- Maintain a counter against each cache line.
- Increment the counter with each reference.
- Replace the item with lowest count with new data item.
  - Reset the counter.

Random Replacement

- Easy to implement.
- Faster processing
- Performance is not very poor in comparison to the rest of the algorithms.

Cache Write Policy

- Writing data into memory / cache faces some issues.
- Processor modifies the data and writes to cache/memory.
- Data is first written into the cache.
  - It must also be updated on main memory.
- Cache content modified by the processor shall be written onto main memory.
  - Must not overwrite a cache block unless main memory is up to date.
- Data inconsistency can be resulted due to a no. of issues.
  - Multiple CPUs may have individual caches.
  - I/O may address main memory directly which may updated the main memory contents.

Cache Write Policy

↳ Must not overwrite a cache block unless main memory is up to data.

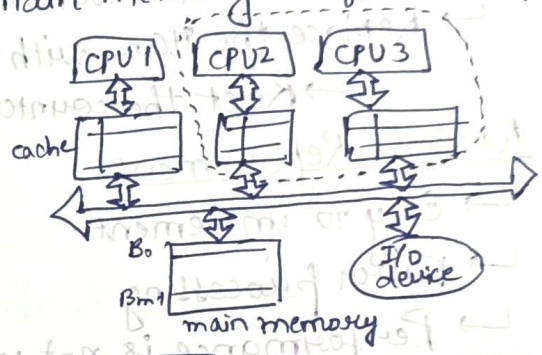
↳ Even if
- Multiple CPUs present and they may have individual caches or
- I/O may address main memory directly.

Write Through

↳ All writes go to main memory as well as cache.
↳ Main memory is always updated.

↳ Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to data.

↳ Lots of traffic
↳ Slows down writes.



Write Back

↳ Updates initially made in cache only.

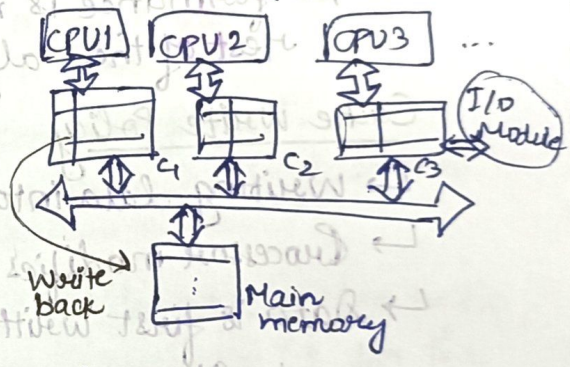
↳ Update bit for cache slot is set when update occurs.

↳ If block is to be replaced, write to main memory only if update bit is set.

↳ Other caches get out of sync.

↳ I/O must access main memory through cache.
↳ Complex circuitry required.

↳ Memory writes
↳ Normal applications : only 15% (85% for READ)
↳ HPC applications : 33–50%.



Example: Write back vs. Write Through ?

- Cache with line size :
  - 32 bytes (8 words of 4 bytes each)
  - Assuming each word gets a write at least once.
  - Main memory data transfer rate : word (4 bytes) in 30 ns.

- Write through case:
  - No. of word writes : 32/4 = 8 word writes
  - No. of write back line requires : 8 [Each word writes takes one]
  - Each word write takes 30 ns.
  - Total write time required : 30ns × 8 words × 8 writes = 15·36 ns
- write Back case:
  - No. of word writes : 32/4 = 8 writes
  - No. of write back of line requires : 1
  - Each line update takes 30 ns.
  - Total write time required : 30 ns × 8 = 240 ns.

Shared Main Memory Systems

→ When main memory is shared
  ↳ By multiple processor-cache systems:
  ↳ Cache coherency is a major issue.
    ↳ If one cache line is altered, main memory and other caches invalidate the data.
    ↳ Present in write-through-cache also.

→ Method for cache coherency
  ↳ Bus watching with write through
    ↳ Each cache controller monitors the address bus.
    ↳ If any other cache controller writes to a main memory block, the other caches invalidates the cache data (if present).
    ↳ Useful for write-through caches.

Hardware transparency

Non cacheable memory.
  ↳ only a portion of the memory is shared among the processors [designated as "non-cacheable" memory].
  ↳ All access to this area face miss.

Hardware transparency:
  ↳ Additional hardware circuitry is used to update all caches whenever there is a write to main memory through cache.

Line Size

→ Retrieve not only desired word but a no of adjacent words as well.

→ Increased block size will increase hit ratio at first
  - the principle of locality

→ Hit ratio ↓↓ as block size becomes even bigger
  - Probability of using newly fetched info. becomes less than probability of reusing replaced.

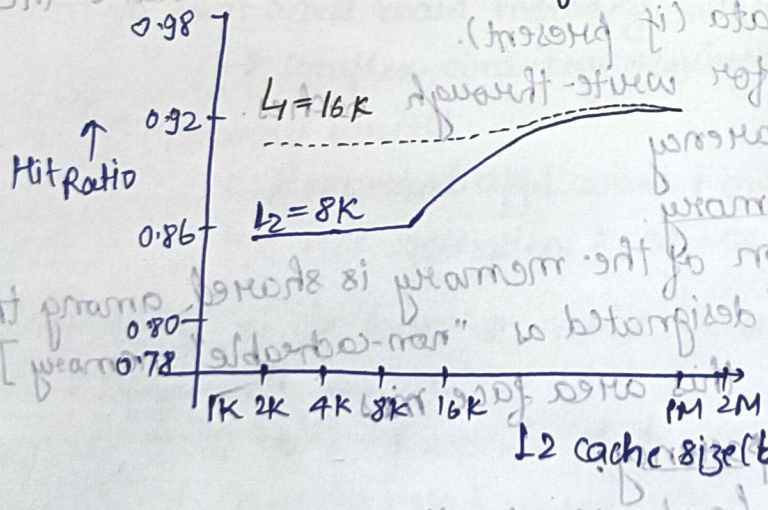
→ Larger blocks
  - No. of blocks that fit in cache ↓
  - Data overwritten shortly after being fetched.
  - Each additional word is less local so less likely to be needed.

→ No definitive optimum value (8 to 64 bytes – reasonable)

Multilevel caches

→ High logic density enables caches on chip
  - Faster than bus access
  - Frees bus for other transfers

→ Common to use both one and off chip cache.
  - L_1 on chip, L_2 off chip in static RAM.
    ↳ access much faster than DRAM or ROM
    ↳ often uses separate datapath.



Unified and Split caches

→ Unified cache → one cache for data and instructions.

→ Split cache
  ↳ Two separate caches
  ↳ one for data & one for instructions.
  ↳ Advantage: Eliminates cache "contention" b/w instruction fetch/decode unit and execution unit
    ↳ Important in pipelining.

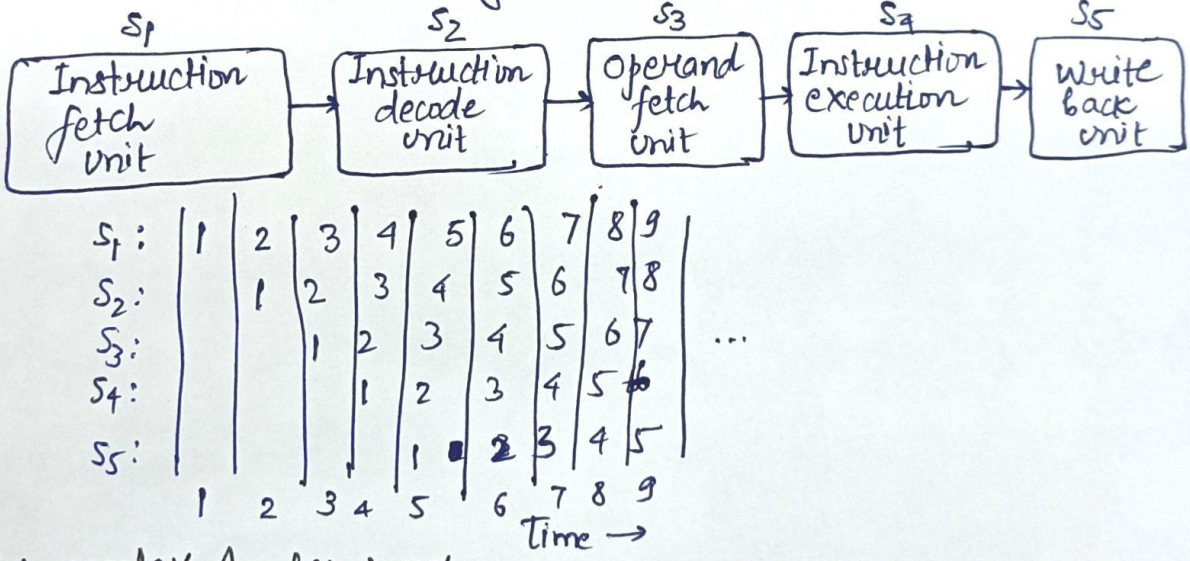
→ Unified cache advantage:
  ↳ High hit rate
    ↳ Balances load of instruction and data fetch.
    ↳ only one cache to design and implement.

Lecture-20

Pipelining

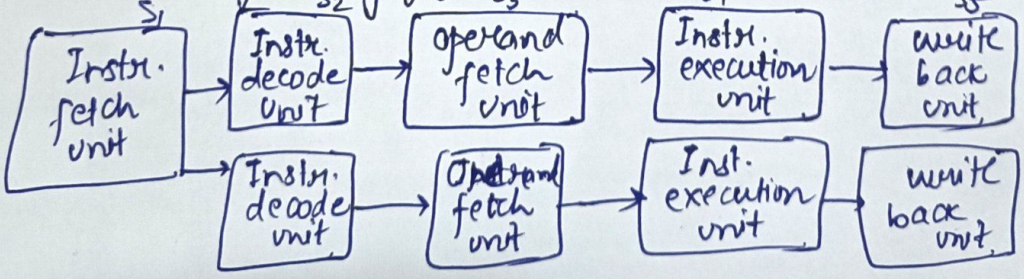
A five-stage pipeline:

↳ The state of each stage as a function of time.

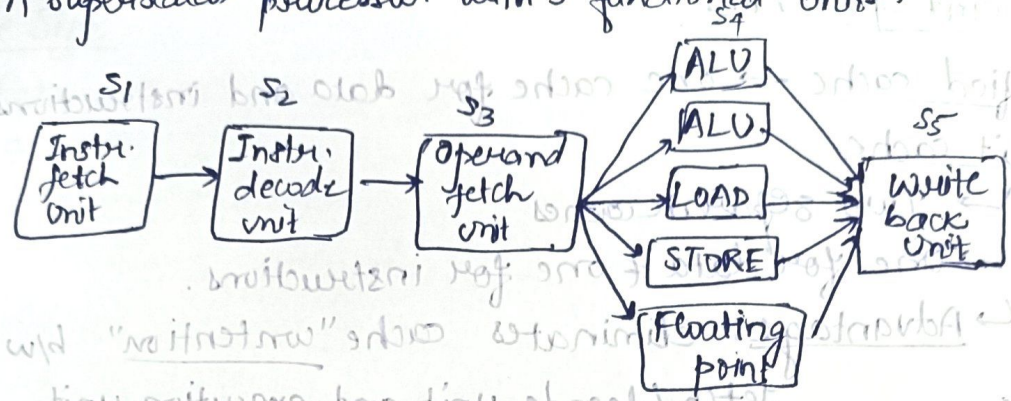


Superscalar Architectures

↳ Dual five-stage pipelines with a common instruction fetch unit.



→ A superscalar processor with 5 functional units :

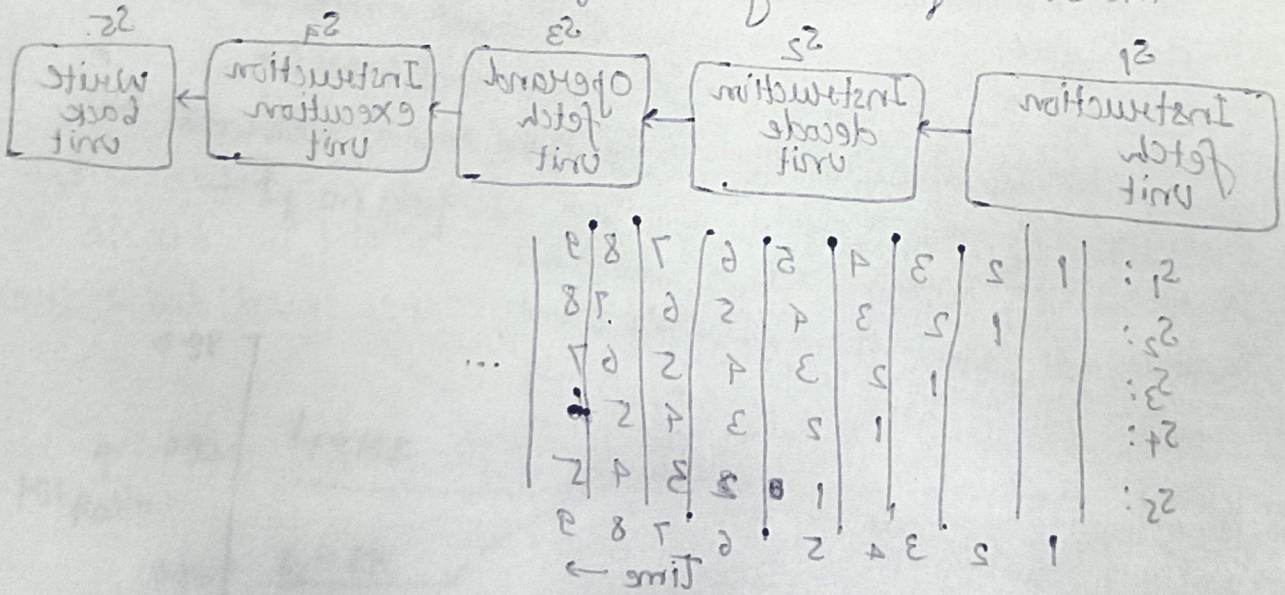


Lecture-20

Pipelining

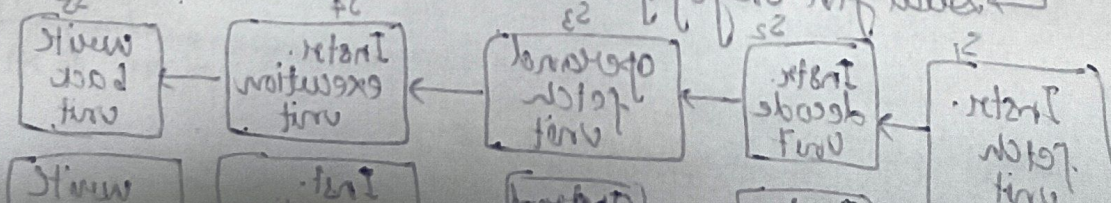
A five-stage pipeline :

→ The space-time diagram of each stage as a function of time.



Superscalar Architecture

→ Dual five-stage pipeline with a common instruction fetch unit

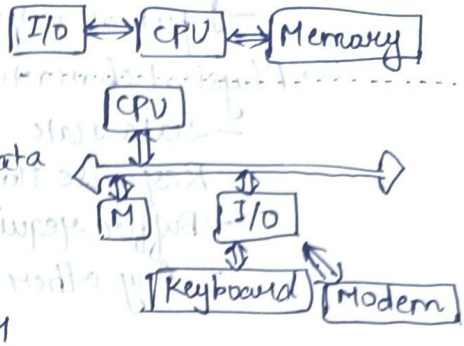


I/O SUBSYSTEM

Input/Output Problems

↳ Challanges in interfacing with I/O

- Wide variety of peripherals
  - ↳ Delivering different amounts of data
  - ↳ at different speeds
  - ↳ in different formats
- Most are slower than CPU and RAM
  - ↳ Some may be even faster (eg, optical fiber links)
- Need I/O modules that can handle the diversity



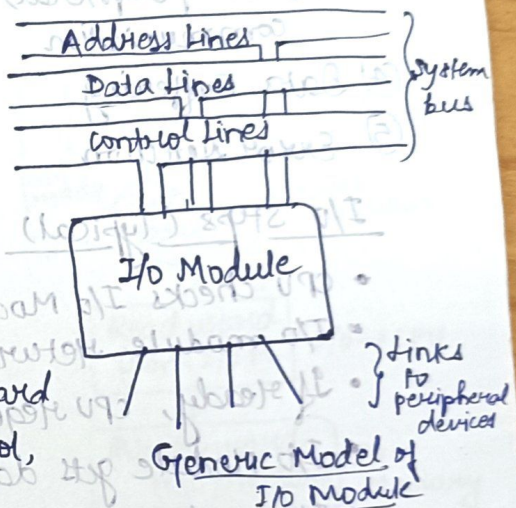
Lecture-21

Input/Output Modules

↳ Main functions:

↳ Interface to CPU and Memory

↳ Interface to one or more peripherals.



Generic Model of I/O Module

Diversity of external Devices

- Human Readable → Screen, printer, keyboard
- Machine Readable → Monitoring and control, Sensor data
- Communication → Modem, Network Interface Card (NIC)
- Based on Data Flow direction:
  - ↳ Input devices: CPU ← PD
  - ↳ Output devices: CPU → PD
  - ↳ Input & Output Devices: Communication interfaces CPU ↔ PD

Bandwidth/Data Rate Characteristics

→ Major types of I/O devices:

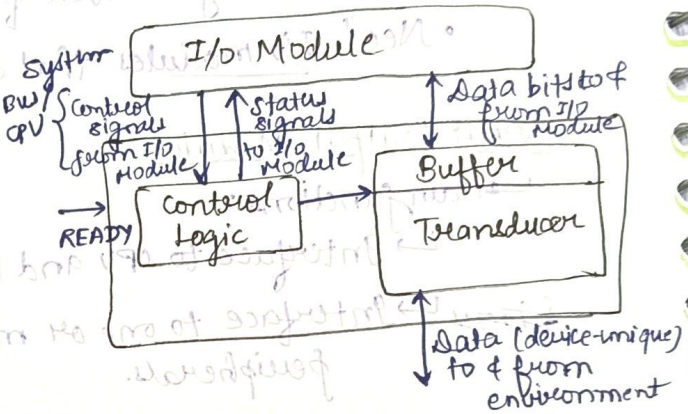
- Human Readable
- Machine Readable
- Communications

→ Contributing factors:
- Partner: Humans have slower input/output data rates than machines.
- Input or output or both.

→ Physical characteristic:
- Data rate
- Response time (delay characteristics)
- Buffer requirement
- Any other special requirements.

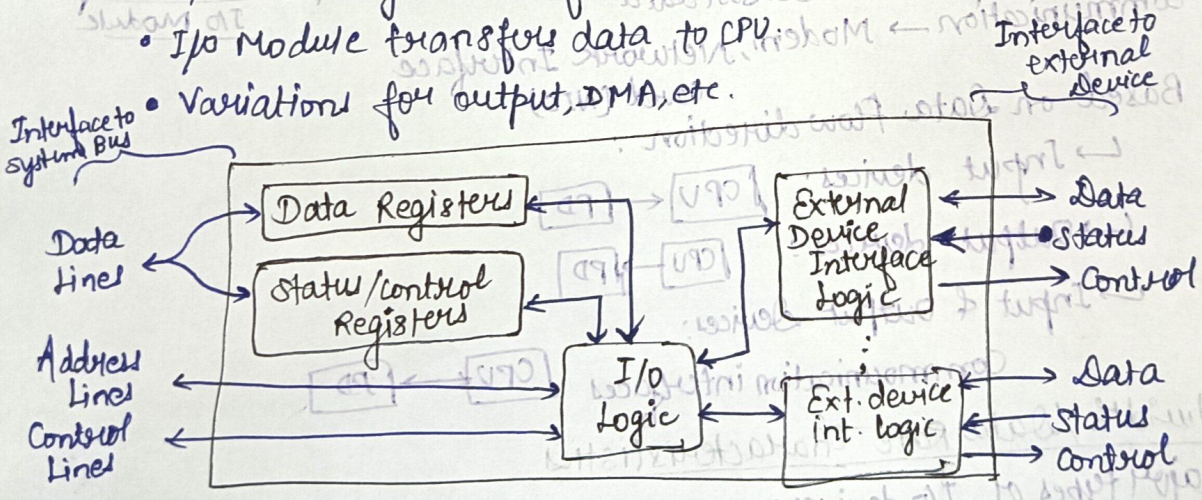
Functions of I/O Modules

① Control and Timing
② CPU communication
③ Device (peripheral) communication.
④ Data Buffering
⑤ Error Detection



I/O Steps (Typical)

- CPU checks I/O Module device status.
- I/O module returns status.
- If ready, CPU requests data transfer.
- I/O Module gets data from device.
- I/O Module transfers data to CPU.
- Variations for output, DMA, etc.



I/O Module Decisions

- Hide or reveal device properties to CPU.
- Support multiple or single device.
- Control device functions or leave for CPU
- Also OS decisions
  - ↳ e.g., Unix treats everything it can as a file.

Input Output Techniques

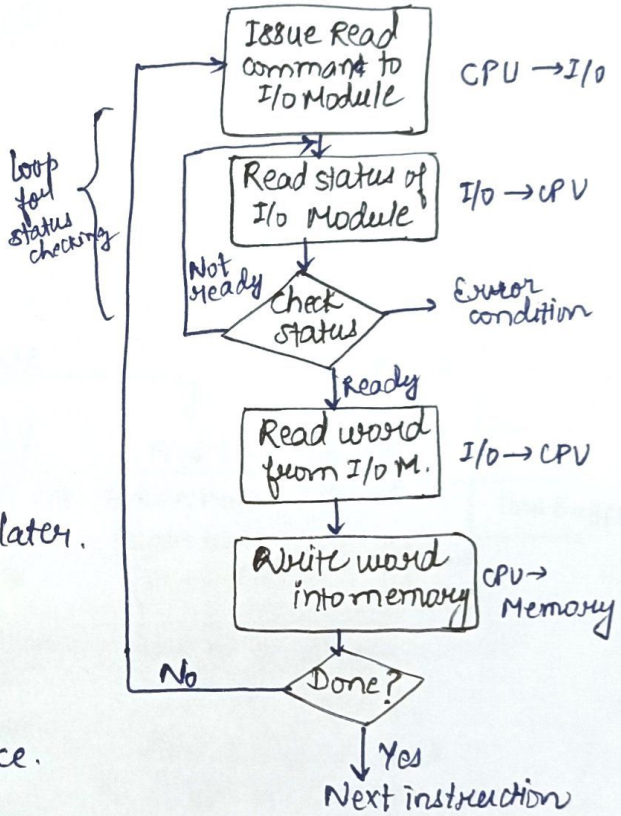
- Programmed I/O operation
- Interrupt driven I/O operation
- Direct Memory Access (DMA) based I/O operation.

Programmed I/O

→ CPU has direct control over I/O.
- Sensing status
- Read/write commands
- Transferring data

→ CPU waits for I/O module to complete operation.
→ Wastes CPU time.

→ CPU requests I/O operation.
→ I/O module performs operation.
→ I/O module sets status bits.
→ CPU checks status bit periodically.
→ I/O module does not inform CPU directly.
→ I/O module does not interrupt CPU.
→ CPU may wait or come back later.



I/O Commands

→ CPU issues addresses
- Identifies module and device.

→ CPU issues command
- Control – telling module what to do. (e.g., spin up disk)
- Test – check status (e.g., power?, Error?)
- Read/write
  - Module transfers data via buffer from/to device.

Addressing I/O Devices

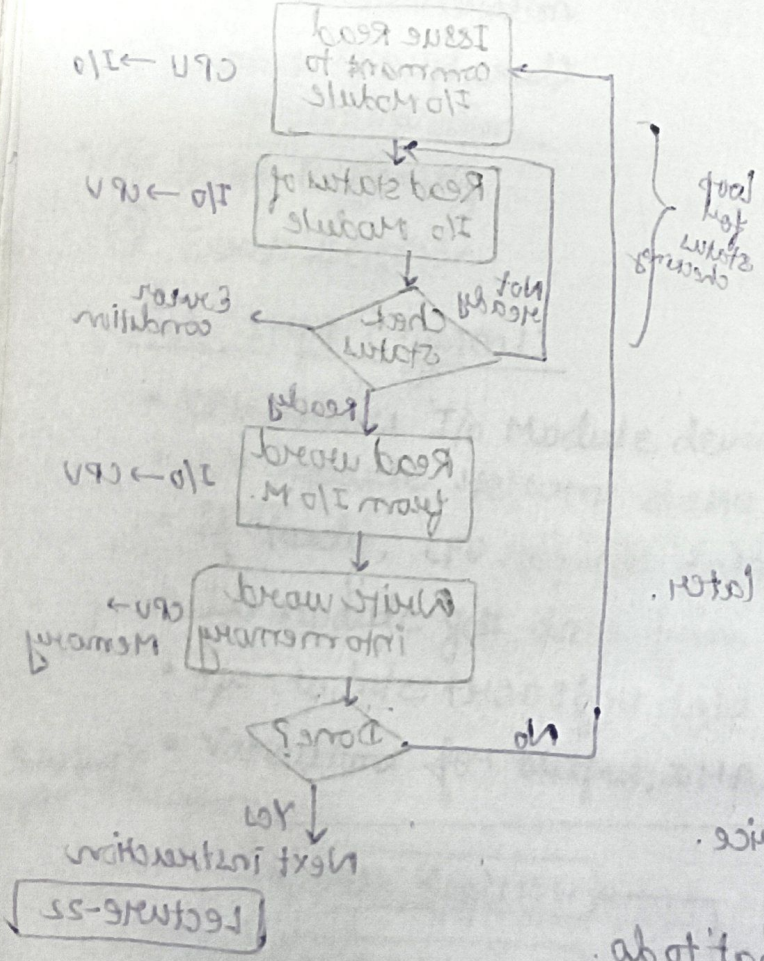
→ Under programmed I/O data transfer is very like memory access (CPU viewpoint).
→ Each device given unique identifier.
→ Each commands contain identifier (address).

Lecture-22

I/O Mapping

→ Memory mapped I/O
  ↳ Devices and memory share an address space.
  ↳ I/O looks just like memory read/write.
  ↳ No special commands for I/O
    ↳ Large selection of memory access commands available.

→ Isolated I/O or Peripheral I/O or I/O mapped I/O
  ↳ Separate address spaces.
  ↳ Need I/O or memory select lines.
  ↳ Special commands for I/O
    ↳ Limited set



Lecture-23

General Purpose Programmable Peripheral I/O

→ 8255A : Chip for GPPP I/O (PPI)

↳ can be used for any other processor (8085, 8086, ...).

↳ Multiple types of I/O can be used: Simple and interrupt driven.

↳ 24 pins for I/O

↳ Port A } two 8 bit parallel ports
↳ Port B }

↳ Port C } two 4 bits parall ports
↳ Port C_upper
↳ Port C_lower

↳ Port functions, configurations and purpose can be programmed by writing a control word.

8255A : Ports and control word

Control word

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

[0/1]

BSR Mode (Bit set/Reset)

I/O Mode

Port Port C

Non effect on I/O mode

Mode 0
Simple I/O for ports A, B & C

Mode 1
Handshake I/O for port A and/or B

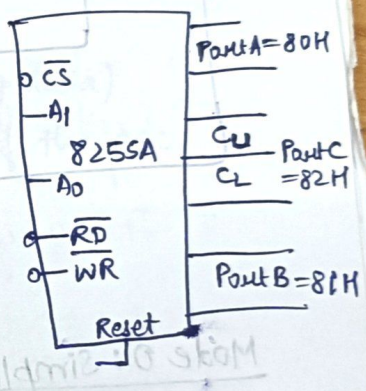
Port C bits are used for handshake

Interrupt-driven I/O

Mode 2
Bidirectional data bus for port A

Port B: either in Mode 0 or 1

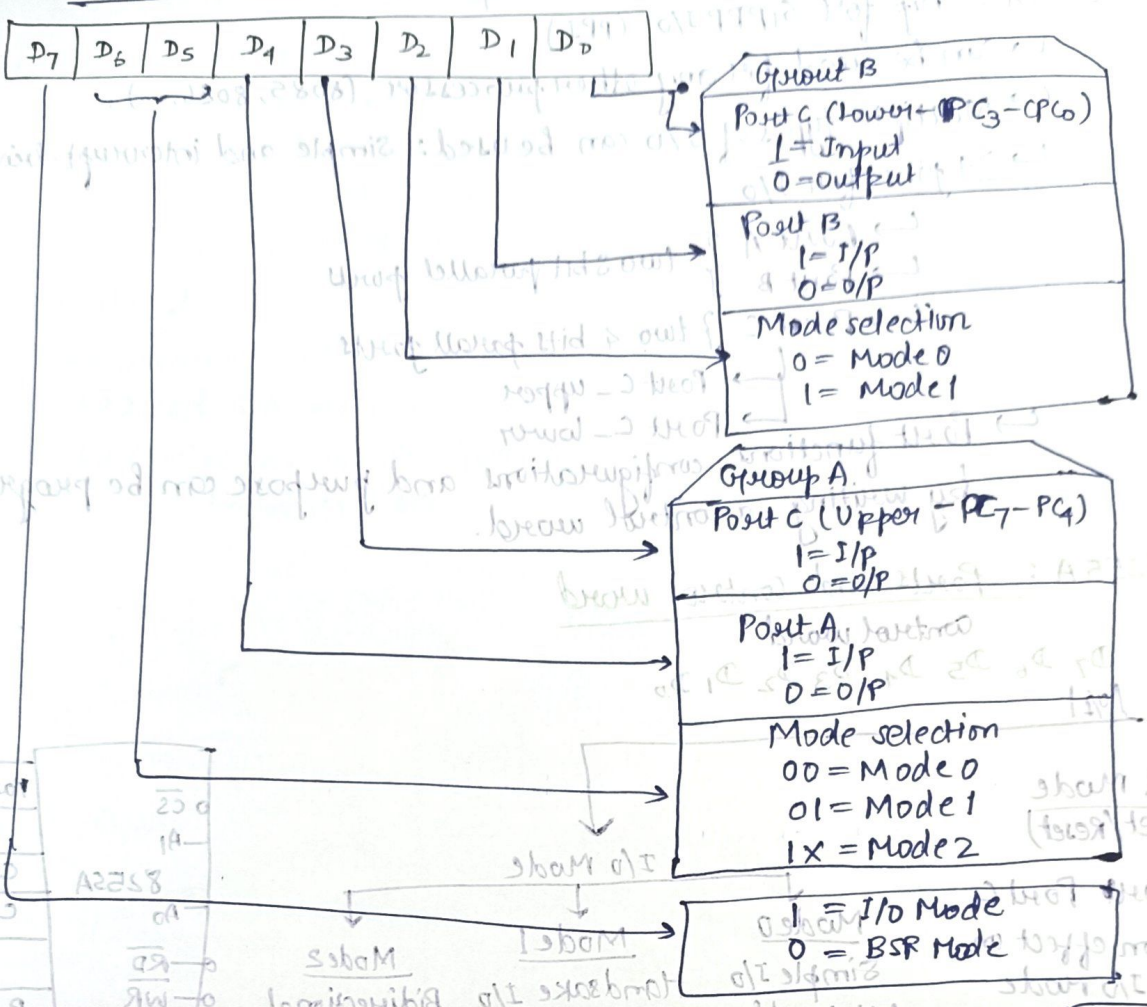
Port C bits are used for handshake.



A_7 A_6 A_5 A_4 A_3 A_2
1 0 0 0 0 0

$\overline{CS}$	A_1	A_0	Selected
0	0	0	Port A (80H)
0	0	1	Port B (81H)
0	1	0	Port C (82H)
0	1	1	control Register (83H)
1	X	X	8255A is not selected.

Control word

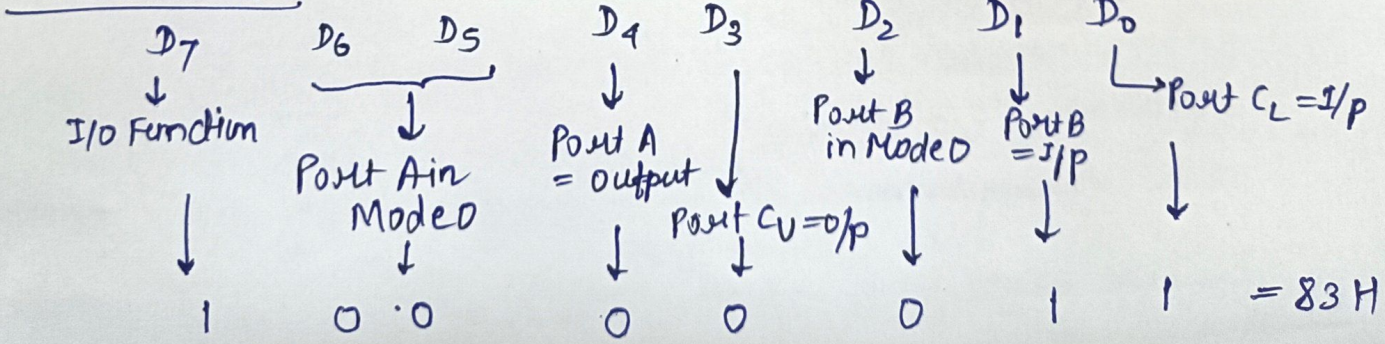


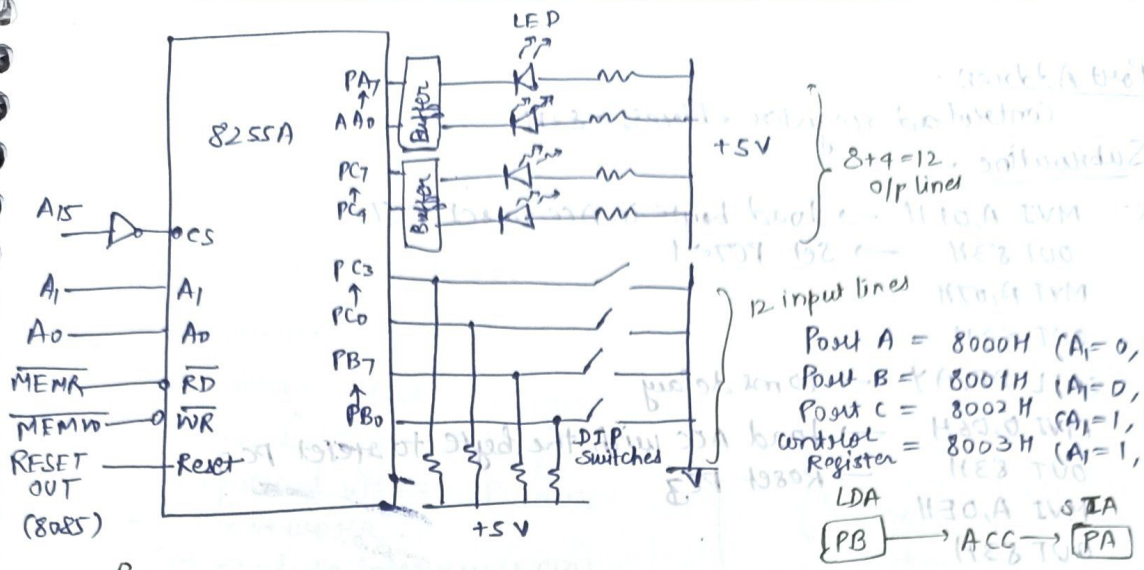
Lecture-24

Mode 0: Simple Input/Output

→ Ports A and B can be used as two simple 8-bit I/O ports
→ Port C as two 4-bit I/O ports
  ↳ Each sub-ports can be programmed separately.
→ Outputs are latched (buffered)
→ Inputs are not latched.
→ Ports do not interrupt or handsake capability.

Control word:





Program:

```
MVI A, 83H   → Load ACC with control word
STA 8003H    → Write word in the control register to initialize the ports
LDA 8001H    → Read switches at port B
STA 8000H    → Display the reading at port A
LDA 8002H    → Read switches at port C
ANI 0FH      → Mask the upper 4 bits of PC (not i/p data)
RLC          → Rotate & place data in upper half of the ACC.
RLC
RLC
RLC
STA 8002H    → Display data at port C
HLT
```

BSR Mode:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	x	x	x	Bit select			S/R

- D_7 = 0 → BSR Mode
- D_6, D_5, D_4 = Not used, generally set = 0
- D_3, D_2, D_1: 000 = Bit 0, 001 = Bit 1, ..., 111 = Bit 7
- D_0: Set = 1, Reset = 0

Eg. Write a BSR control word subroutine to set bits PC_7 and PC_3 and reset them after 10 ms.

BSR control words

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
To set bit PC7	0	0	0	0	1	1	1	1	= 0FH
To reset bit PC7	0	0	0	0	1	1	1	0	= 0EH
To set bit PC3	0	0	0	0	0	1	1	1	= 07H
To reset bit PC3	0	0	0	0	0	1	1	0	= 06H

Port Address:

Control ~~ad~~ register address = 83H

Subroutine:

```
BSR:  MVI A,0FH  → Load byte in ACC to set PC7
      OUT 83H    → Set PC7 = 1
      MVI A,07H
      OUT 83H
      CALL DELAY → 10 ms delay
      MVI 0,06H  → Load ACC with the byte to reset PC3
      OUT 83H    → Reset PC3
      MVI A,0EH
      OUT 83H
      RET
```

Lecture-25

Mode 1

↳ Ports A and B ~~are~~ as 8 bit I/O ports
→ Port C lines are used for handshaking for Ports A & B.
→ I/P and O/P are latched.
→ Interrupt logic (handshake is supported).

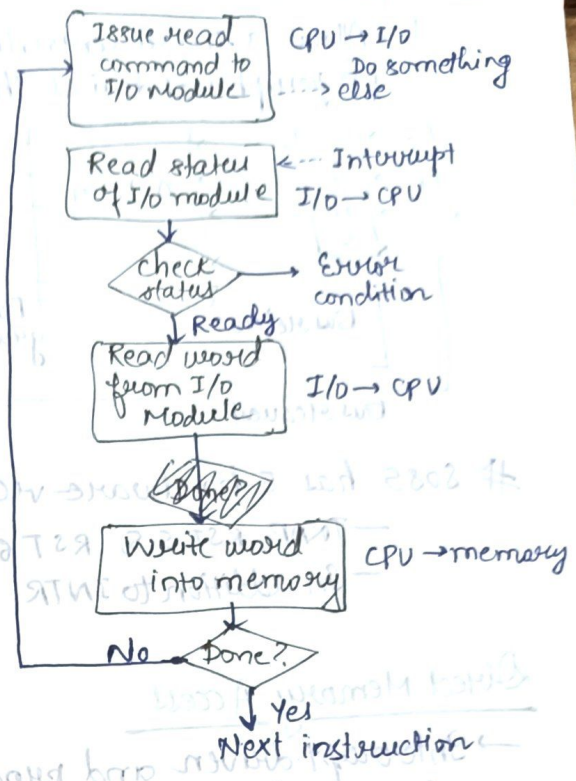
Mode 2 : Bi-directional data transfer

→ For data transfer b/w two computers, storage controllers.
→ Port A
  ↳ can be configured as bi-directional port
  ↳ Uses 5 lines from Port C for handshaking.
→ Port B
  ↳ Either in Mode 0 or Mode 1.
→ Port C
  ↳ 5 lines are for Port A handshaking.
  ↳ 3 lines can be used as simple I/O or handshake for port B.

Interrupt Driven I/O

→ CPU overcomes CPU waiting.
→ No repeated CPU checking of device.
→ I/O module interrupts when ready.

Basic operation:
→ CPU issues read command.
→ I/O module gets data from peripheral whilst CPU does other work.
→ I/O module interrupts CPU.
→ CPU requests data.
→ I/O module transfers data.



Design Issues:
→ Identifying the module issuing the interrupt.
→ Dealing with multiple interrupts.
(ie, an interrupt handler being interrupted)
→ Each interrupt line has a priority.
→ Higher priority lines can interrupt lower priority lines.
→ If bus mastering, only current master can interrupt.
→ Different lines for each module.
↳ PC
↳ Limits no. of devices.
→ Software poll.
↳ CPU asks each module in turn.
↳ Slow
→ Daisy chain or Hardware poll
↳ Interrupt Acknowledge sent down a chain.
↳ Module responsible places vector on bus.
↳ CPU uses vector to identify handler routine.
→ Bus Master
↳ Module must claim the bus before it can raise interrupt.
↳ Eg., PCI & SCSI

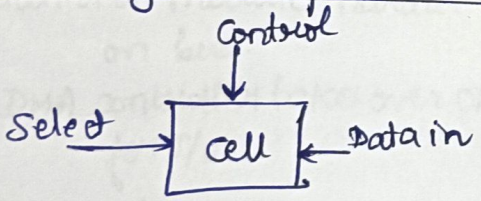
Internal Memory

→ Internal Memory
  ↳ Cache/Main memory
→ External Memory
  ↳ External storage devices
  ↳ HDDs / Disks / TAPE Devices Drives.

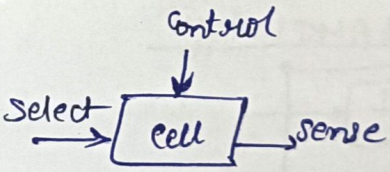
Semiconductor Memory Types :

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks (Chip layout, high volume)	Non-volatile
Programmable ROM (PROM)				
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level (exposure for 10-20m)	Electrically	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash Memory		Electrically, block-level		

Memory-cell operation (1bit) :



Write



Read

RAM
- → Read/write
- → Volatile
- → Temporary storage
- → Static or dynamic

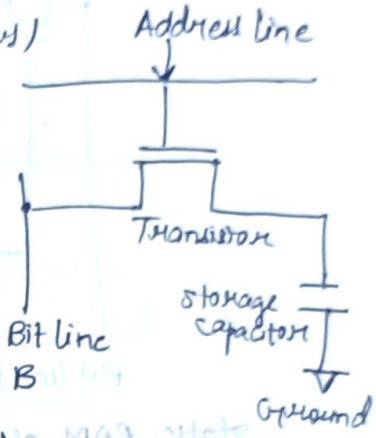
Dynamic RAM
- ↳ Bit stored as charge in capacitors
- ↳ Charges leak
- ↳ Need refreshing even when powered
- ↳ Simpler construction.

↳ Smaller area per bit
↳ Less expensive
↳ Need refresh circuits
↳ Slower
↳ Main memory
↳ Essentially analog
↳ Level of charge determines value.

DRAM operation:

→ Address line active when bit ~~is selected~~ read or written.
↳ Transistor switch closed (current flows)

→ Write:
↳ Voltage to bit line (High→1, Low→0)
↳ Then signal address line
↳ Transfers charge to capacitor.



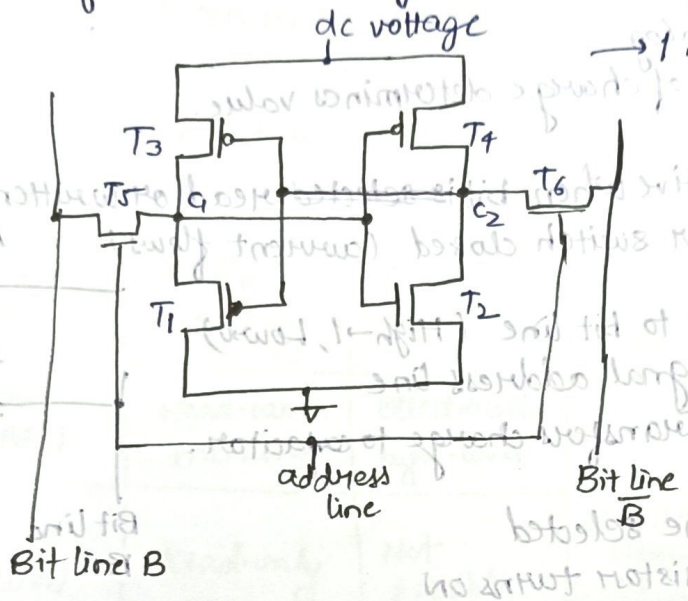
→ Read:
↳ Address line selected
↳ Transistor turns ON
→ Charge from capacitor fed via bit line to sense amplifier
↳ Compares with ref. value to determine 0 or 1.
→ Capacitor charge must be restored.

DRAM organisation: Many ways
↳ 16 Mb chip → organised as 1M of 16 bit words.
↳ as 2048 × 2048 × 4 bit array
↳ Reduces no. of address pins.

Refreshing:
↳ Refresh circuit included on chip.
↳ Disable chip.
↳ Count through rows.
↳ Read & write back.
↳ Takes time
↳ Slows down apparent performance.

Static RAM
↳ Bits stored as on/off switches.
↳ Transistors used for bi-stable multi-vibrators.
↳ Digital → uses flip flops.
↳ No charges to leak.
↳ No refreshing needed when powered.
↳ Does not need refresh circuits.
↳ More complex construction.

→ Larger size/area per bit.
→ More expensive.
→ ~~Faster~~ Faster.
→ Suitable for cache memory.



→ 1 bit Memory cell.

Static RAM operation:

→ Transistor arrangement gives logic state.
→ State 1 : C_1 high, C_2 low
T_1, T_4 OFF ; T_2, T_3 ON
→ State 0: C_2 high, C_1 low
T_2, T_3 OFF ; T_1, T_4 ON
→ Address line transistors T_4 and T_6 are switch.
→ Write – apply value to B & complement to B
→ Read – value is on line B.

ROM

↳ Permanent storage
↳ Non-volatile
↳ Microprogramming
↳ Library subroutines
↳ System programs (BIOS)
↳ Function tables.

External Memory

Lecture-28

Types:

→ Magnetic Disk (HDD, Floppy Drive)
  ↳ RAID (Reduendant Array of Inexpensive/Independent Disks)
  ↳ Removable
→ Optical
  ↳ CD-ROM
  ↳ CD-Recoordable (CD-R)
  ↳ CR-R/W
  ↳ DVD
→ Magnetic Tape ⇝ Archival storage of large volume of data.

Magnetic Disk

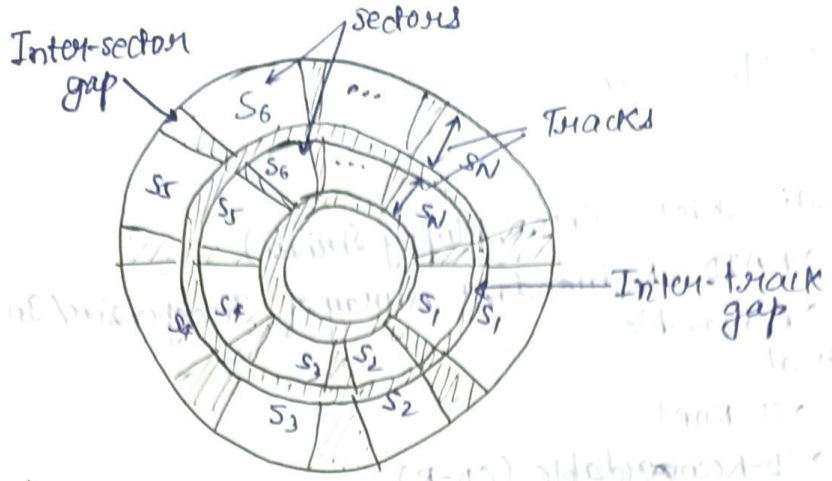
↳ Disk substrate coated with magnetizable material (Iron oxide).
↳ Substrate used to be aluminum [Now glass].

Read-Write Mechanisms

↳ Recording & retrieval via conductive coil called head
  ↳ May be single or separate for read/write.
↳ Write: current through coil produces magnetic field
  ↳ Pulses sent to head
  ↳ Magnetic pattern recorded on surface below.
↳ Read (traditional)
  ↳ Mag. field moving relative to coil produces current
  ↳ same coil for read/write.
↳ (Contemporary)
  ↳ Separate read head, close to write head.
  ↳ Higher storage density and speed.
  ↳ Partially shielded magneto resistive (MR) sensor.

Data Organization and Formatting

→ concentric rings or tracks.
  ↳ Gaps b/w tracks (reduce to increase capacity).
  ↳ Same no. of bits per track (variable packing density).
  ↳ constant angular velocity.
→ Tracks divided into sectors.
→ Min^m block size is one sector.
→ May have more than one sector per block.



Disk velocity

↳ Bit near centre of rotating disk passes fixed point slower than bit on outside of disk.

↳ Increase spacing b/w bits in different tracks.

↳ Rotate disk at constant angular velocity.

↳ Individual tracks and sectors addressable.

↳ Move head to given track and wait for given sector.

↳ Waste of space on outer tracks

↳ Lower data density.

↳ Can use zones to increase capacity [fixed bits per track]

Characteristics

① Fixed Head (rare)
↳ One R/W per track
↳ Head mounted on fixed ridged arm.

Movable head
↳ One R/W per side
↳ Mounted on a movable arm.

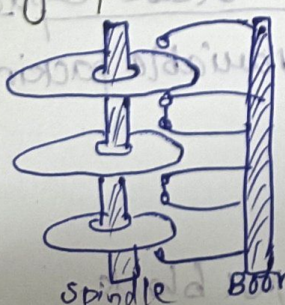
② Removable disk
↳ Can be replaced
↳ Easy data transfer

Fixed
↳ Permanently mounted in the drive

③ Single sided

Double sided
↳ Usually

④ Single platter



Multiple platter
↳ One head per side (joined & aligned)
↳ Aligned tracks on each platter form cylinders
↳ Data is striped by cylinder
↳ Reduces head movement
↳ Increases speed (transfer rate)

⑤ Head Mechanism

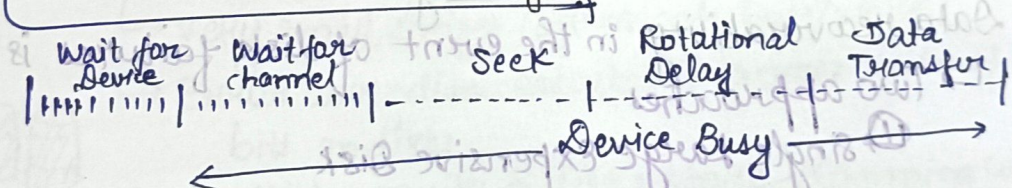
- Contact (Floppy)
  - Head physically touches platter/media
  - Slow access
  - Low density of storage
  - Possibility of damage to the media
  - High wear and tear.
- Fixed Gap ⇝ not very common
  - Maintains gap b/w head and media (as small as possible)
  - Complex to maintain the gap
  - Speed of access is faster.
- Flying (Winchester)
  - Head rests on media when no rotation.
  - when media/disc rotates at high speed, head is aerodynamically lifted and a gap is maintained.
  - Common in Hard disk drives.

Speed of Data Transfer

→ Seek time: Moving head to correct track

→ (Rotational) latency: Waiting for data to rotate under head.

→ **Access time = Seek + Latency**



→ Seek time
- Initial start up time
- Time taken to traverse the tracks that have to be crossed.
- Settling time.
- Typically, <10ms for HDDs.
- Mostly contributed by mechanism design.

→ Rotational delay: 3600 RPM to 20000 RPM (speed)
- R.D ~ 1.5ms (average)

Transfer time, $T = \frac{B}{rN}$, B: no. of bytes to read
N: no. of bytes on a track
r: rotational speed in RPS.

→ Access time = Av. seek time + Rotational delay + Data Transfer time

↳ $T_a = T_s + T_r + \text{Transfer time}$

$= T_s + \frac{1}{2r} + \frac{b}{rN}$

Lecture-29

Solid State Drives

↳ NAND flash memory-based drives.
  ↳ High voltage is able change the configuration of a floating-gate transistor.
  ↳ State of the transistor interpreted as binary data.
↳ More resilient against physical damage
↳ Greatly reduced power consumption (no mechanical, moving parts).
↳ Much faster than hard drives
  ↳ No penalty for random access
    ↳ Each flash cell can be addressed directly.
    ↳ No need to rotate or seek.
  ↳ Extremely high throughput.

Performance of External Memory

↳ Data recoverability in the event of disk failures is critical.
  ↳ Two approaches
    (1) Single Large Expensive Disk
      ↳ High capacity, high transfer rate
        ↳ Low access time
    (2) Redundant Array of Inexpensive (Independent) Disks [RAID]
      ↳ Stored data is striped among several physical drives
        ↳ Redundant disks provide higher reliability, robustness and fault tolerance.
          ↳ High data recoverability in the event of disk failures
            ↳ OS sees a single logical drive.

RAID

↳ Not a hierarchy

↳ Set of physical disks viewed as single logical drive by O/s.

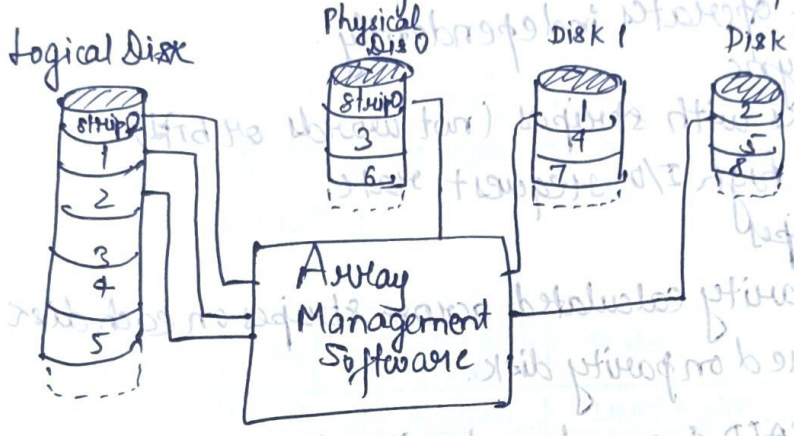
↳ Data distributed across physical drives [by striping].

↳ Can use redundant capacity to store parity information.

RAID 0 → No redundancy

→ Data stripped across all disks [Round Robin striping]

→ Increase speed (disks seek in parallel)



N Disks × C Bytes = NC

RAID 1 → Mirrored Disks

↳ 2 copies of each strip on separate disks)

↳ Read from either, write to both.

↳ Simple recovery, expensive.

RAID 2 → Disks are synchronised

↳ very small stripes (often single byte/word)

↳ Error correction calculated across corresponding bits on disks.

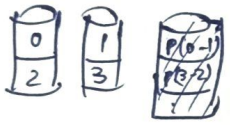
↳ Multiple parity disks store Hamming code error correction in corresponding positions.

↳ Lots of redundancy, expensive, not used.

RAID 3 → Similar to RAID 2
↳ only one redundant disk, no matter how large the array.
↳ simple parity bit for each set of corresponding bits.
↳ very high transfer rate.

Recovery: $x_p = x_1 + x_2 + x_3$ [0, 1, 0]
$\Rightarrow x_p = 0+1+0 = 1 \equiv$ Even parity
x_2 is damaged $\Rightarrow x_2 = x_p + x_1 + x_3$
$= 1+0+0 = 1$ → Recovered.

RAID 4 → Each disk operates independently.
↳ Not sync
↳ works with stripes (not words or bits)
↳ Good for high I/O request rate.
↳ Large stripes
↳ Bit-byte parity calculated across stripes on each disk
↳ Parity stored on parity disk.



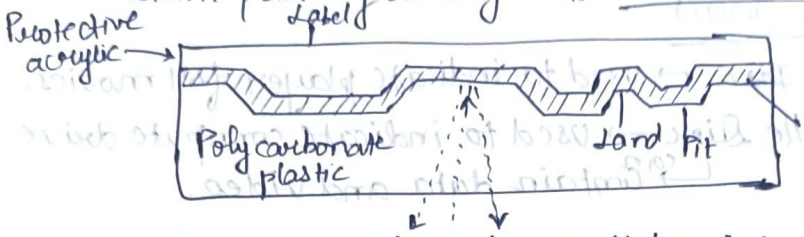
RAID 5 → Similar to RAID 4; reduces parity disk bottleneck
↳ Avoids RAID 4 bottleneck at parity disk.
↳ Commonly used in network servers.
↳ Complex recovery.

RAID 6 → Two parity calculations
↳ Stored in separate blocks on different disks.
↳ User requirement of N disks needs N+2.
↳ High data availability.

Lecture-30

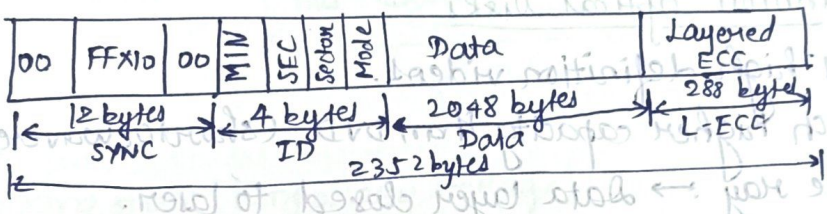
Optical Storage: CD-ROM

- Polycarbonate coated with highly reflective coat, usually Al.
- Data stored as pits
- Read by reflecting laser.
- Constant packing density and linear velocity



- Audio is single speed (1.2 m/s).
- Other speeds are quoted as multiple (e.g., 24X) [Max^m speed].

CD-ROM format:



Mode 0 = blank data field
Mode 1 = 2048 byte data + error correction
Mode 2 = 2336 byte data.

Random Access on CD-ROM

- Difficult; Move head to rough position
- Set correct speed
- ~~Dodo~~ Read address & data.
- Adjust to required location (takes time).

Advantages:

- Large capacity
- Easy to mass produce
- Removable
- Robust

Disadvantages:

- Expensive for small runs
- Slow
- Read-only

CD-Recordable (CD-R)

- Write once Read Many (WORM)
- Was affordable
- Compatible with CD-ROM drives.

CD-RW

- Erasable (50K-100K times rewritable)
- Getting cheaper
- Mostly CD-ROM drive compatible
- Phase change → 2 different reflectivities in different phase states.

Digital Video Disk (DVD)

- Digital Video Disk. → used to indicate player for movies.
- Digital Versatile Disk → used to indicate compute drive.
  - can contain data and video.
- Multi-layer
- very high capacity (4.7G per layer)
- Movies carry regional coding.
- writable.

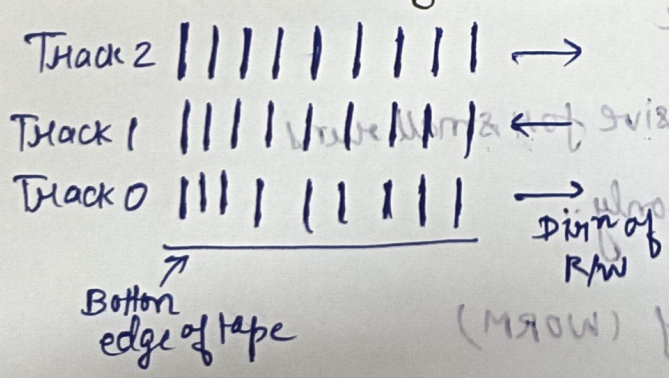
High Definition Optical Disks

- For high definition videos.
- Much higher capacity than DVD (shorter wavelength laser).
- Blue ray → Data layer closed to laser.
- Available read-only, recordable once, re-recordable.

Magnetic Tape

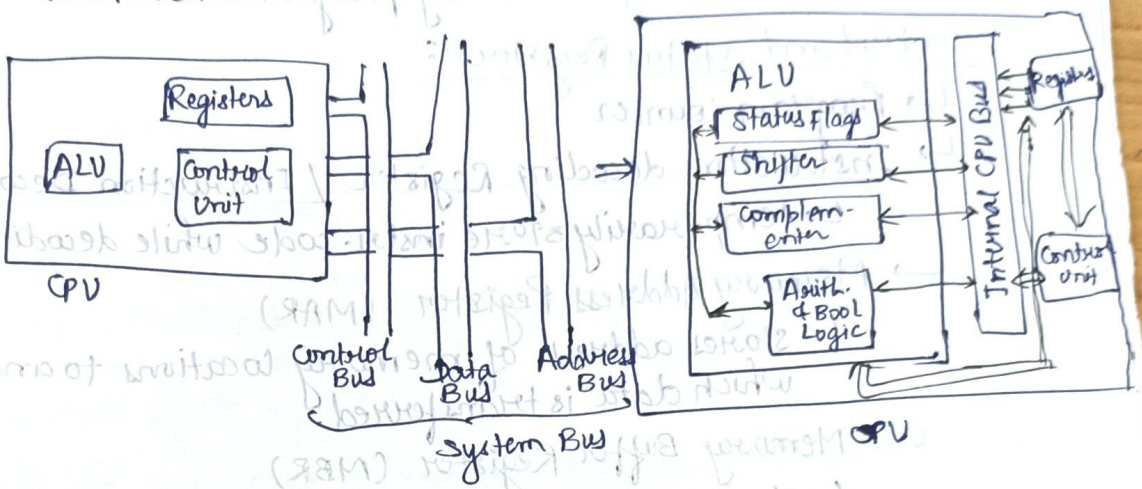
- Serial access
- Slow
- very high storage capacity
- very cheap
- Backup and archive.
- Linear Tape-Open (LTO) Tape Drives.

Serpentine reading and writing of Tape drives:



OPERATION OF A CPU

- Fetch instructions
- Interpret instructions
- Fetch data (Immediate)
- Process data
- Write data



Registers

- Working space (temporary storage)
- Top level of memory hierarchy.

User visible Registers

- General Purpose
- Data
- Address
- Condition codes (Flags)

General Purpose Registers:

- Maybe true GP or restricted
- Used for data or addressing
  - Accumulator
  - segment, offset (e.g. HL pair)
- Design:
  - GP: Increase flexibility & programmer options
    - Increase instr. size & complexity
  - Specialised → Smaller (faster) instr.
    - Less flexible
  - B/w 8–32
    - Fewer → more memory ref.
    - More → reduce mem. ref.
      - Takes processor real estate.

↳ Large enough to hold full address and full word.
↳ often possible to combine two data registers.

Condition Code Registers:
↳ Set of individual bits (e.g., result of last operation was zero)
↳ can be read (implicitly) by programs (e.g., jump if zero)
↳ cannot (usually) be set by programs.

Control and Status Registers:
↳ Program counter
↳ Instruction decoding Register / Instruction Decoder (IR)
↳ temporarily stores instr. code while decoding, executing.
↳ Memory Address Register (MAR)
↳ stores address of memory locations to and from which data is transferred.
↳ Memory Buffer Register (MBR)
↳ stores data being transferred to and from immediate access storage (memory).

Program Status Word → A set of bits
↳ Includes condition codes
↳ Sign of last result
↳ Zero
↳ Carry
↳ Equal
↳ Overflow
↳ Interrupt enable/disable
↳ Supervisor

Data Flow (Instr. Fetch):
↳ PC contains address of next instr.
↳ Address moved to MAR.
↳ Address placed on address bus.
↳ Control unit requests memory read.
↳ Result placed on data bus, copied to MBR, then to IR.
↳ Meanwhile, PC = PC + 1;

Data Flow (Data Fetch):

↳ IR is examined.
↳ If indirect addressing, indirect cycle is performed.
  ↳ Right most N bits of MBR transferred to MAR.
  ↳ Control unit requests memory read.
  ↳ Result (result of operand) moved to MBR.

Data Flow (Execute):

May include:
- ↳ Memory read/write
- ↳ Input/Output
- ↳ Register transfers
- ↳ ALU operations.

Data Flow (Interrupt):

↳ Simple, Predictable.
↳ Current PC saved to allow resumption after interrupt.
↳ Contents of PC copied to MBR
↳ Special memory location (eg, stack pointer) loaded to MAR.
↳ MBR written to memory.
↳ PC loaded with address of interrupt handling routine.
↳ Next instr. can be fetched.

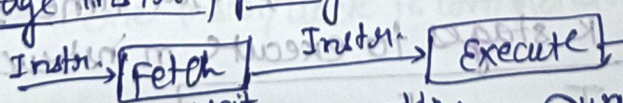
Lecture - 32

Prefetch

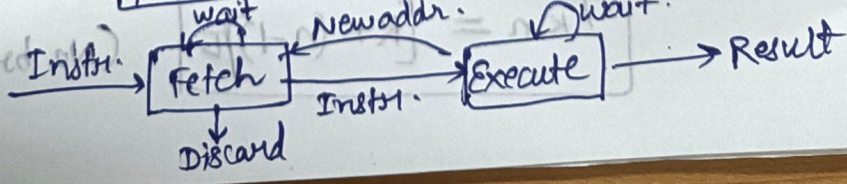
↳ Fetch accessing main memory.
↳ ~~Execution~~ can fetch next instruction during execution of current instruction.
↳ Improved performance
  ↳ But not doubled.
  ↳ Add more stages to improve performance.

Pipelining

↳ Overlap operations.
↳ Two stage instr. pipelining:



Expanded view:



→ Operations:

① Fetch Instruction (FI) → Read into buffer.
② Decode Instruction (DI) → Determine opcode & operand specifiers.
③ Calculate operands (CO) → Calculate effective addr. of each source operand.
④ Fetch operands (FO) → From memory; need not from registers.
⑤ Execute Instruction (EI) → Perform indicated operation & store the result, if any, to specified destination.
⑥ Write operand (WO) → Store result in memory.

Timing Diagram

Time →

Instn.	1	2	3	4	5	6	7	8	9	10	11	12
1	FI	DI	CO	FO	EI	WO						
2		FI	DI	CO	FO	EI	WO					
3			FI	DI	CO	FO	EI	WO				
4				FI	DI	CO	FO	EI	WO			
5					FI	DI	CO	FO	EI	WO		
6						FI	DI	CO	FO	EI	WO	
7							FI	DI	CO	FO	EI	WO

7 instn. in 12 clock cycles
⇓
12/7 clock cycles per instn.
⇓
compare this to $7 \times 6 = 42$ cycles.

Pipelining performance and ~~clock~~ cycle time

cycle time, $$\tau = \max_i [\tau_i] + d \quad [1 \le i \le k]$$
$$= \tau_m + d$$

τ_i → time delay in circuitary in i^{th} stage of pipeline.
τ_m → max stage delay.
k → no. of stages
d → time delay in latch [to advance signal to next stage].
↳ equivalent to clock pulse

$$\tau_m \gg d$$

For n instn. with no branches, total time required for a pipeline with k stages to execute n instructions,

$$T_{k,n} = [k + (n-1)]\tau \quad \text{(No branching)}$$

For no pipeling, instruction cycle time = KT

Speed up factor for instr. pipeline, $S_K = \frac{T_{1,n}}{T_{K,n}} = \frac{nKT}{[K+(n-1)]T} = \boxed{\frac{nK}{K+(n-1)}}$

Eg A pipelined processor has a clock rate of 2.5 GHz and executes a program with 1.5 M instr. The pipeline has 5 stages, and instr. are issued at a rate of one clock per clock cycle. Ignore penalities due to branch instr. and out-of-sequence executions.

(a) What is the speedup of this processor for this program compared to a non-pipelined processor?

(b) What is throughput (in MIPS) of the pipelined processor?

Soln: Since large no. of instr., we can exclude initial filling as well as end finishing up.

(a) So, speedup factor is 5.

(b) 1 instr. in 1 clock cycle $\Rightarrow$ MIPS = 2500 MIPS.

Eg A non-pipelined processor has a clock rate of 2.5 GHz and an average CPI (clocks per instr.) of 4. An upgrade to the processor introduces a 5-stage pipeline. However, due to internal pipeline delays, such as latch delay, the clock rate of the new processor has to be reduced to 2 GHz.

(a) What is the speedup achieved for a typical program?

(b) What is the MIPS rate for each processor?

Soln: (a) $K=5, n=100$ (say)

Speedup, $S = \frac{nK}{K+(n-1)} = \frac{500}{104} = 4.8$

However, the pipelined 2 GHz CPU will have a reduced speed of a factor of 0.8 to the unpipelined 2.5 GHz CPU.

So, overall speedup $= 4.8 \times 0.8 = 3.8$

(b) Non-pipelined: Each instr. takes 4 clocks. So, MIPS $= \frac{2500 \text{ MHz}}{4} = 625$ MIPS.

Pipelined: One instr/clock is completed. So, MIPS = 2000 MIPS.

Lecture-33

~~Consider~~

Pipelining with Branching

Consider an instr. sequence of length n that is streaming through the instruction pipeline.

p: probability of encountering a conditional or unconditional branch instruction.

q: probability that execution of a branch instr. causes a jump to a non-consecutive address.

No. of instr. causing branching = npq

No. of instr. not causing branching = (1−pq) n

Non-branching case: Time taken by a pipelined system of K stages, n instructions and cycle time τ,

$$T_{K,n} = [K + (n-1)]\tau$$

Branching case:

$$T_K = npq\,nK\tau + (1-pq)\,[K + (n-1)]\tau$$

Speedup (non-branching):

$$S_K = \frac{T_{1,n}}{T_{K,n}} = \frac{nK\tau}{[K+(n-1)]\tau} = \frac{nK}{K+(n-1)}$$

(Branching case) speedup:

$$S_K = \frac{T_1}{T_K} = \frac{nK\tau}{(pq)\,nK\tau + (1-qp)[K+(n-1)]\tau} = \frac{nK}{pq\,nK + (1-pq)[K+(n-1)]}$$

Pipeline Hazards

↳ Conditions in a pipelined machine that impede the execution of a subsequent instruction in a cycle.

↳ Types:
- Resource
- Data
- Control

Resource Hazards / Structural Hazard

↳ Two (or more) instructions in pipeline need same resource.
↳ Executed in serial rather than parallel for part in pipeline.

Eg. Instr. fetches & data reads and writes performed at once.
↳ Single ALU ↳ Idle for one cycle (fetch) (single main memory port)
↳ Solutions: Increase available resources
↳ Multiple main memory ports
↳ Multiple ALUs.

Data Hazards

↳ Conflict in access of an operand location.

Eg. 2 instr. to be executed ~~in sequence~~ (in pipeline), operand value could be updated so as to produce different result from strict sequential execution.

↳ Pipeline stall for 2 clock cycles.

Types:

① Read after write (RAW) or true dependency
↳ If an instr. modifies a register or memory location; succeeding inst. read data from that location; hazard if read takes place before write complete.

② Write after read (WAR) or anti-dependency
↳ An instr. reads a location; succeeding instr. writes there; hazard if write completes before read takes place.

③ Write after write (WAW) or output dependency
↳ Two instr. both write to same location; hazard if writes take place in reverse order.

Lecture 34-35

Control Hazard / Branch Hazard

↳ Pipeline makes wrong decision on branch prediction.
↳ Brings instr. into pipeline that must subsequently be discarded.

Dealing with Branches

① Multiple streams → 2 pipelines; prefetch each branch into separate pipeline.
② Prefetch Branch Target ↳ Leads to contention delay for access to the registers and to memory.
③ Loop Buffer
④ Branch Prediction
⑤ Delayed Branching.

Loop Buffer

↳ Contains the N most recently fetched instructions in sequence.
↳ Fetch check buffer before fetching from memory.
↳ Similar to cache.
↳ No additional memory access time.
↳ Well suited to dealing with loops or iterations.

Branch Prediction → Static / Dynamic

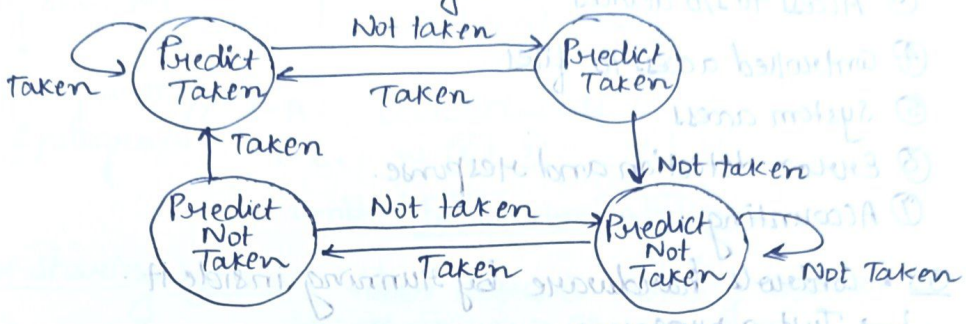
↳ ① Static Branch Prediction:
↳ Not using the execution history as well as the results of execution (use a predefined logic).
↳ Predict Never taken → assume branch does not happen.
↳ Always fetch next instr.
↳ Predict Always taken → assume branch always happens.
↳ Always fetch target instr.
↳ 50% chance for branching.
↳ Predict by Opcode → Some instr. are more likely to result in a jump than others.
↳ Can get upto 75% success.

② Dynamic Branch Prediction:
↳ History of conditional branches in the program was considered.
↳ Taken / Not taken switch → one or more bits can be associated with each conditional branch instr. that reflects the recent history of the instr.
↳ Based on previous history.
↳ Good for loops.

(Refined by 2-level or correlation-based branch history)

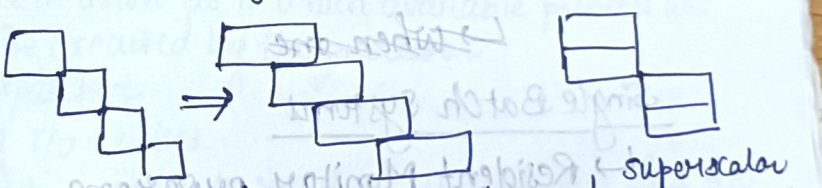
↳ Correlation-Based → In more complex structures, branch direction correlates with that of related branches.
↳ Use recent branch history as well.
↳ Two bit-branch prediction to record result of last 2 instances of the execution of the associated instruction.

Branch Prediction state Diagram:



Delayed Branching → Do not jump until you have to.
↳ Rearrange instructions without Data hazard.
↳ The instr. location immediately following the branch is refered to as delay slot.

Superpipelining
↳ Many pipeline stages perform tasks that require less than half a clock cycle.
Thus, a doubled internal clock speed allows the performance of 2 tasks in one external clock cycle.
↳ Functions performed in each stage can be split into 2 non-overlapping parts.
↳ Degree 2.



Superscalar → Depends on the ability to execute multiple instructions in parallel.
Instruction-level parallelism: Degree to which, on average, the instruction of a program can be executed in parallel.

Lecture-36

OPERATING SYSTEM

OS services:
① Program creation
② Program execution
③ Access to I/o devices
④ Controlled access to files
⑤ System access
⑥ Error detection and response.
⑦ Accounting

OS: Controls hardware by running inside it.
↳ Just a program.

Types:
① Interactive: User directly interacts with the hardware
② Batch: A user's program is batched other user's.
↳ Submitted by an operator.
③ Single ~~Batch~~ Program (Uni-programming):
↳ Processor is used for executing one program at a time.
④ Multi-programming (Multi tasking):
↳ several programs are loaded to the memory.
↳ Processor is made to switch b/w them rapidly.
↳ ~~When one~~

Single Batch Systems
↳ Resident Monitor program
↳ User submit jobs to operator.
↳ Operator batches jobs.
↳ Monitor controls sequence of events to process batch.
↳ when one job is finished, control returns to Monitor which reads next job.
↳ Monitor handles scheduling.
Job control language: Instructions to monitor
↳ usually denoted by $.
↳ Features: Monitor Protection
↳ Timer → To prevent a job monopolizing the system.

↳ Privileged instructions (e.g., I/O) → only executed by Monitor.
↳ Interrupts → Allows for relinquishing and regaining control.

Multi-programmed Batch Systems:

↳ I/O devices are very slow.
↳ when one program is waiting for I/O, another can use the CPU.

Uniprogramming: [Run] wait [Run] wait
Time →

Multiprogramming (2 programs):
P-A: [R] W [R] W
P-B: W [R] W [R] W
Combined: [A][B] W [A][B] W
Time →

Time sharing systems: Allows users to directly interact with the computer (i.e., interactive)
↳ Eg. multi-programming.

Scheduling

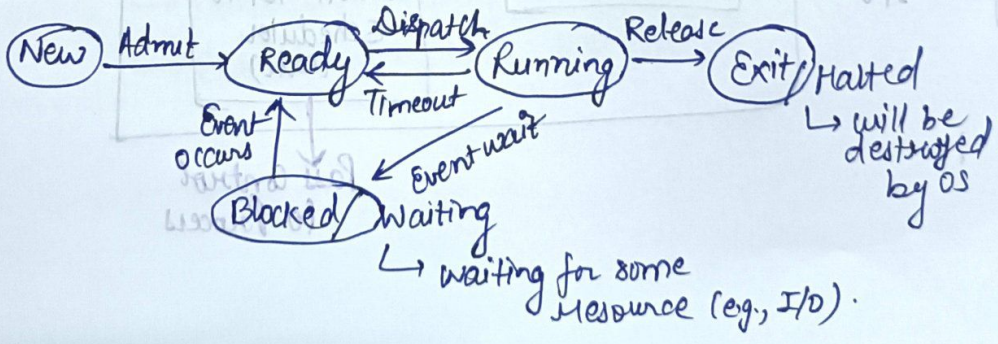
→ Process: A program in execution.
↳ (In multi-pro.) Providing the sequence of execution among the processes / programs

→ Long term → Decision to add to the pool of processes to be executed.
→ Medium term → Decision to add to the no. of processes that are partially or fully in main memory
↳ Usually based on need to manage multi-pro.
→ short term
→ I/O.

→ The decision as to which available process will be executed by the processor
↳ Dispatcher

→ Scheduling the use of I/O devices
↳ The decision as to which process's pending I/O request shall be handled by an available I/O device.

5-stage Process Model:



Process Control Block (PCB)

↳ OS maintains info about the state of the process and other info necessary for process execution.

↳ PCB →
- Identifier
- State
- Priority
- Program Counter
- Memory Pointers
- Context Data
- I/O status
- Accounting info

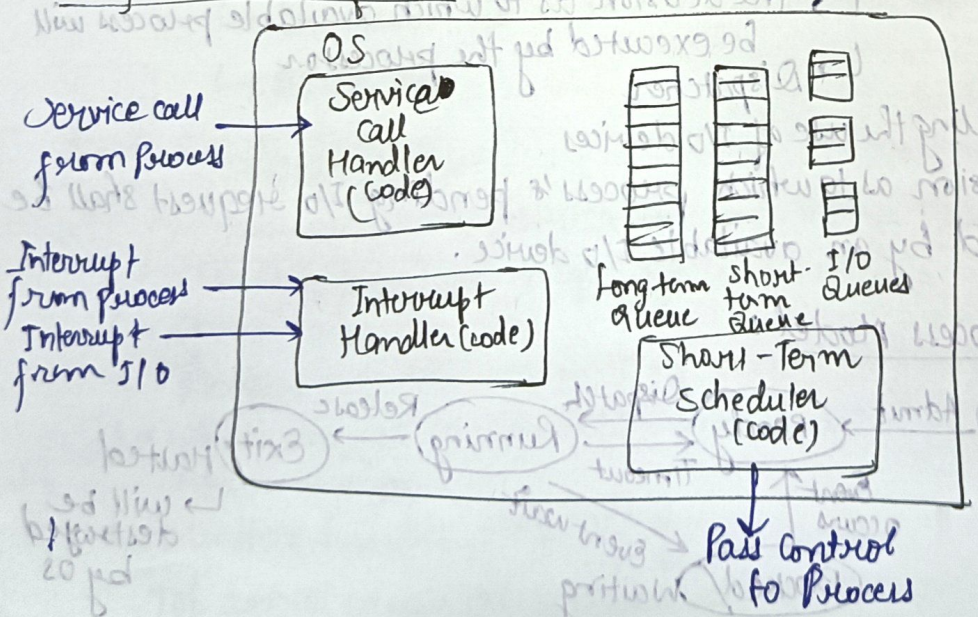
→ OS gets back control when:

① Process A issues a service call (eg., I/O request) to the OS.
↳ execution suspendend until the call is satisfied by OS.

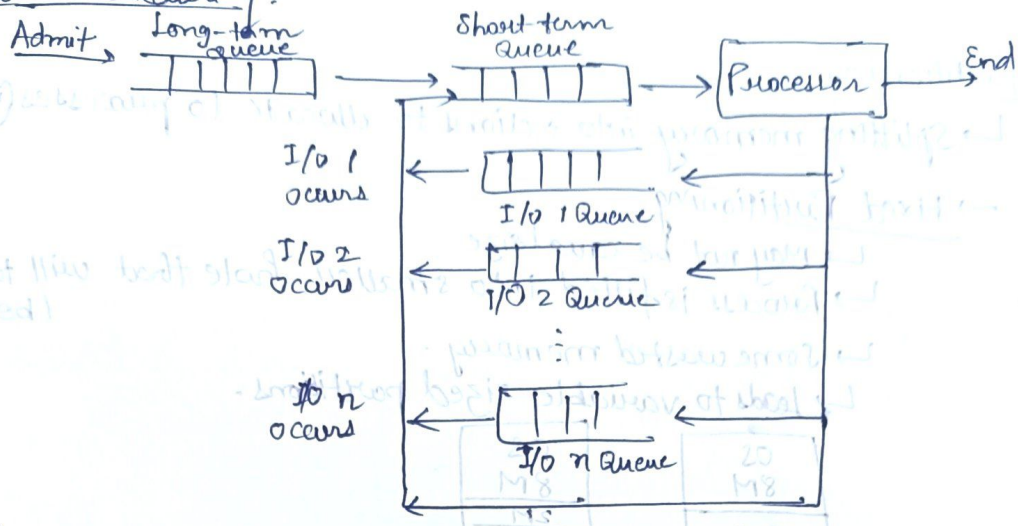
② Process A caused an interrupt (hardware-generated signal) to processor.
↳ Processor ceases to execute A and transfers to the interupt handler in OS.
Eg: Timeout, to prevent a process from monopolizing the processor.

③ Some event unrelated to process A that requires attention causes an interrupt.
↳ Eg. completion of an I/O operation.

Key-Elements of OS:



Process scheduling:



Memory Management:

→ Uni-program: Memory split into 2 ↔ one for OS (monitor)
one for currently executing program.

→ Multi-program: "User" part is sub-divided and shared among active processes.

Swapping

Problem: I/O is so-slow compared to CPU that even in multi-pro. system, CPU can be idle most of the time.

↳ **Solutions:** → Increase main memory ↔ Expensive
Leads to larger problems
→ Swapping.

Swapping:

↳ Long term queue of processes stored on disk.
↳ Processes swapped in as space becomes available.
↳ As a process completes it is moved out of main memory.
↳ If none of the processes in memory are ready (i.e., all I/O blocked)
↳ Swap out a blocked process to immediate queue.
↳ Swap in a ready or a new process.
↳ But swapping is a I/O process.

Partitioning

↳ Splitting memory into sections to allocate to processes (including OS).

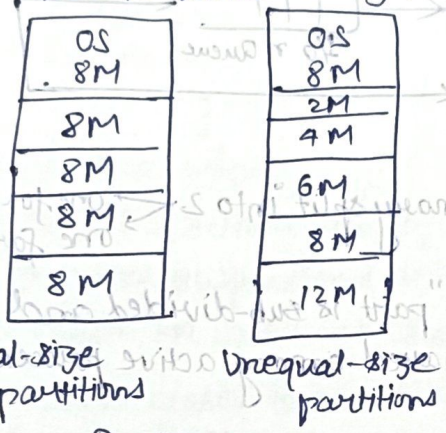
→ **Fixed Partitioning**

↳ May not be equal size.

↳ Process is fitted into smallest hole that will take it (best fit).

↳ Some wasted memory.

↳ Leads to variable sized partitions.



Equal-size partitions

Unequal-size partitions

Variable Size Partitions

↳ Allocate exactly the required memory to a process.

↳ Leads to a hole at the end of memory → too small to use → less waste.

↳ When all process are blocked, swap out a process and bring in another.

↳ New process may be smaller → Another hole.

↳ Eventually have lots of holes (fragmentation).

↳ Solution:

Coalesce: Join adjacent holes into one large hole

Compaction / de-fragmentation: From time to time, go through memory and move all hole into one free block.

~~Relocat~~

Relocation

- No guarantee that process will load into same place in memory.
- Instructions contain addresses
  - Locations of data
  - Addresses for instructions (branching)
- Logical address: Relative to beginning of program.
- Physical address: Actual location in memory (this time).
- Automatic conversion using base address.

Paging

- Split memory into equal sized, small chunks — page frames.
- Split programs (processes) into equal sized small chunks — pages.
- Allocate the required no. of page frames to a process.
- OS maintains list of free frames.
- A process does not require contiguous page frames.
- Use page table to keep track.

Virtual Memory

- Demand paging
  - Do not require all pages of a process in memory.
  - Bring in pages as required.
- Page fault
  - Required page is not in memory.
  - OS must swap in required page.
  - May need to swap out a page to make space.
  - Select page to throw out based on recent history.

Thrashing

- Too many processes in too little memory.
- OS suspends all its time swapping.
- Little or no real work is done.
- Disk light is on all the time.

Solutions:
- Good page replacement algorithms
- Reduce no. of processes running
- Fit more memory.

Lecture 37

Segmentation → visible to programmer.

↳ Paging is (usually) not visible to the programmer.

↳ Usually different segments allocated to program and data.

↳ May be a no. of program and data segments.

↳ **Advantages**: Simplifies handling of growing data structures.

↳ Allows programs to be altered and recompiled independently, without re-linking and re-loading.

↳ Lends itself to sharing among processes.

↳ Lends itself to protection.

↳ Some systems combine segmentation with paging.