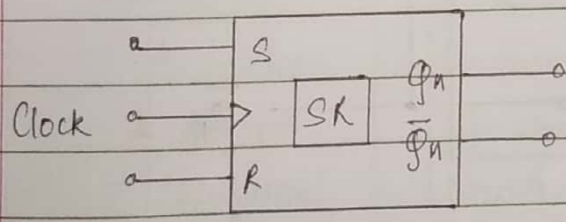
Date

Q1. Compare MUX and DEMUX.

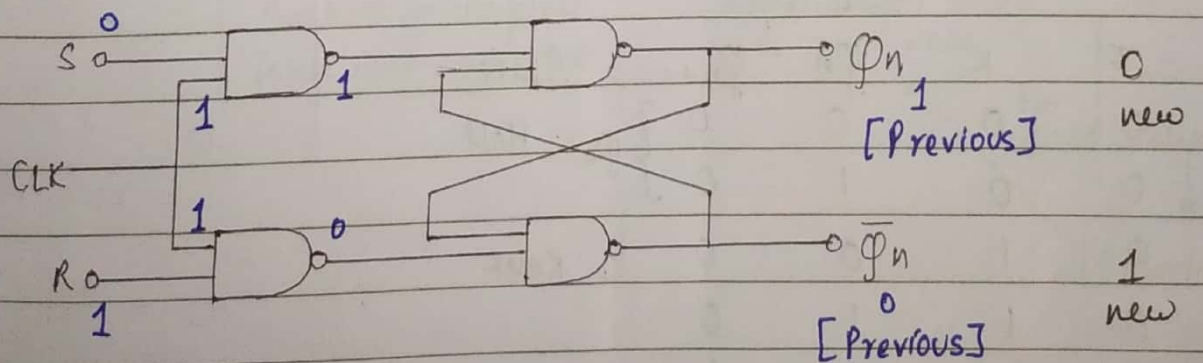
Multiplexer	Demultiplexer
1) A MUX is a combinational circuit that uses several data inputs to generate a single output.	1) A DEMUX is also a combinational circuit that uses single input that can be directed throughout several outputs.
2) A MUX is a data selector.	2) The DEMUX is a data distributor.
3) It is a digital switch.	3) It is a digital circuit.
4) The parallel to serial conversion is is used in the MUX.	4) The serial to parallel conversion is used in the DEMUX.
5) It doesn't use any extra gates while designing.	5) Additional gates are necessary while designing DEMUX.
6) The different types of multiplexers are 8-1 MUX, 16-1 MUX and 32-1 MUX.	6) The different types of DEMUX are 1-8 DEMUX, 1-16 DEMUX and 1-32 DEMUX.

Q2. Explain SR and JK flip flop using diagram and truth table.

→ SR - Set Reset Flip Flop

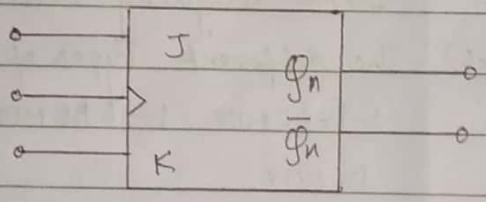


S	R	Q_n
0	0	Q_n → Hold
0	1	0 → Reset
1	0	1 → Set
1	1	X → Invalid

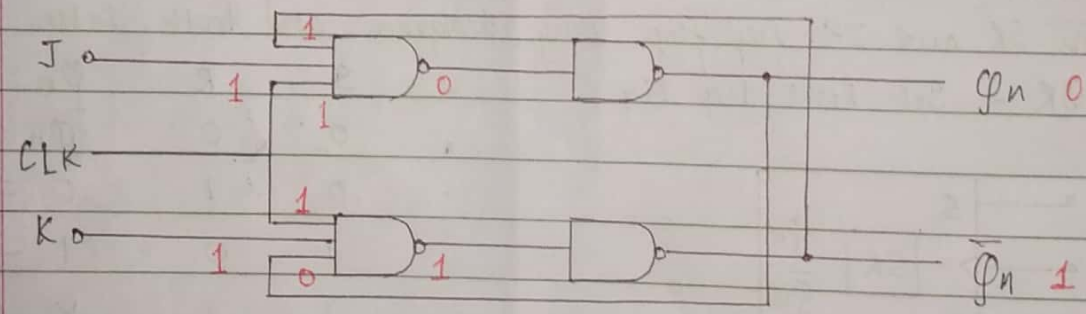


S	R	Q_n (Current Output)	Q_{n+1} (Next state Output)	State
0	0	0	0	Q_n Hold
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	X	Invalid
1	1	1	X	

JK Flip Flop



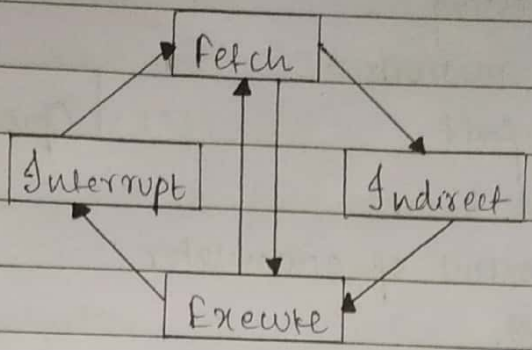
J	K	Q_{n+1}	
0	0	Q_n	Hold
0	1	0	Reset
1	0	1	Set
1	1	$\bar{Q}_n$	Invert



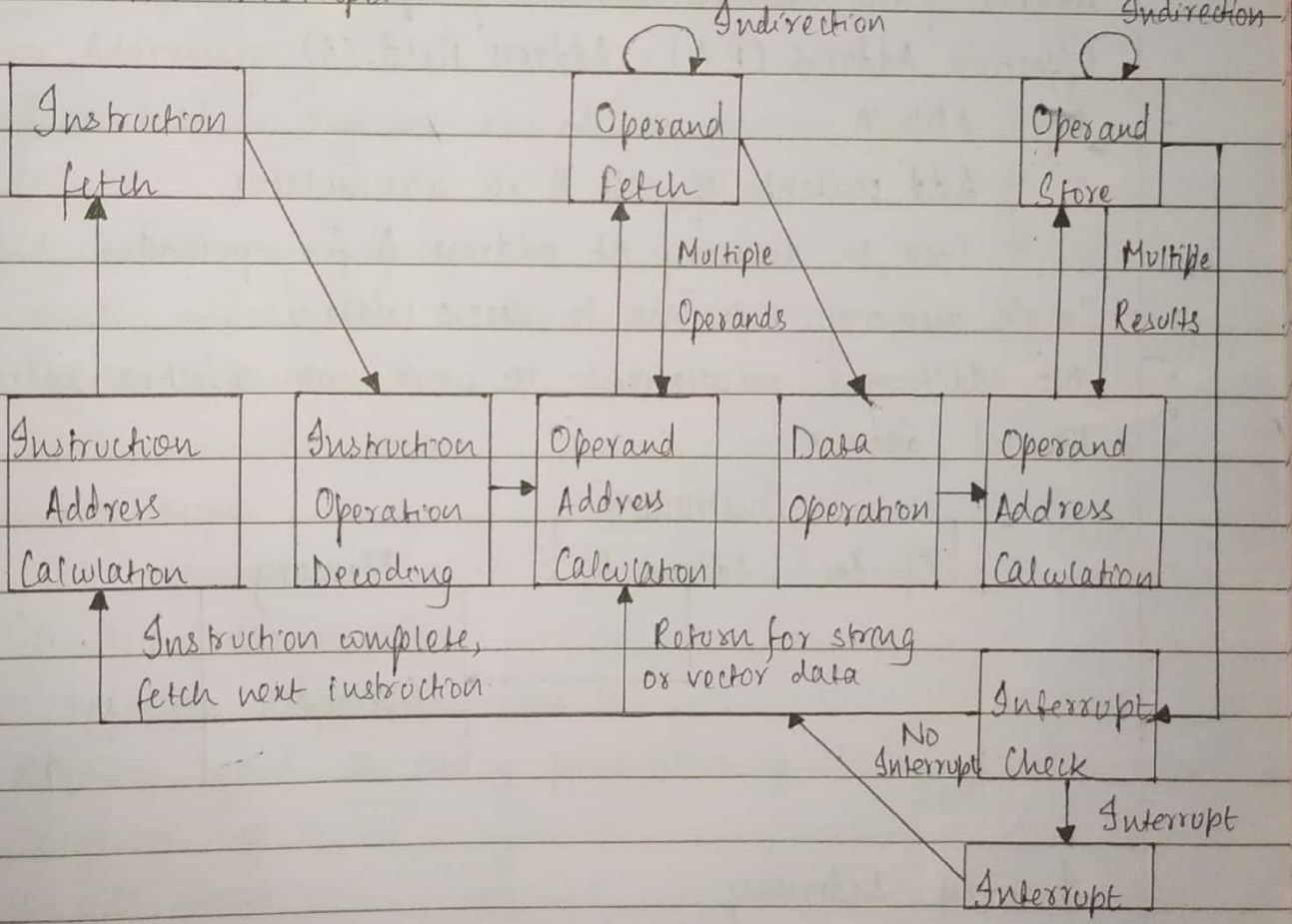
Previous state output

J	K	Q_n	Q_{n+1}	State
0	0	0	0	Q_n Hold
0	0	1	0	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	$\bar{Q}_n$
1	1	1	0	

Q5. Explain instruction cycle using block diagram.



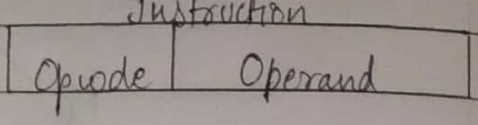
Fetch - Read the next instruction from memory into the processor.
Execute - Interpret the opcode and perform the indicated operation
Interrupt - If interrupts are enabled, and an interrupt has occured, save the current process state and service the interrupt.



Q4. Explain using necessary diagram addressing modes used in processor.

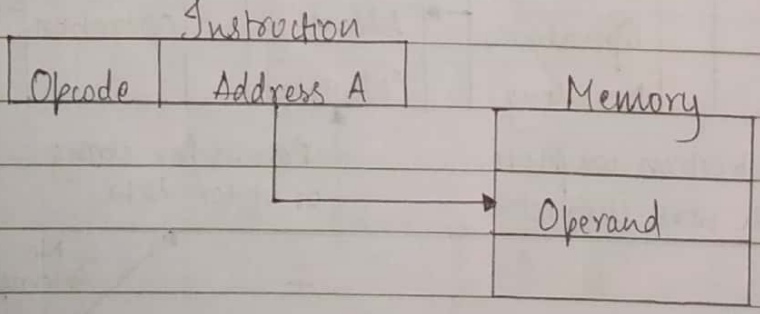
→ Immediate Addressing

- Operate is part of instruction.
- Operand = address field
- Eg - ADD 5
  - Add 5 to content of accumulator
  - 5 is operand.
- No memory reference to fetch data
- Fast
- Limited range



Direct Addressing

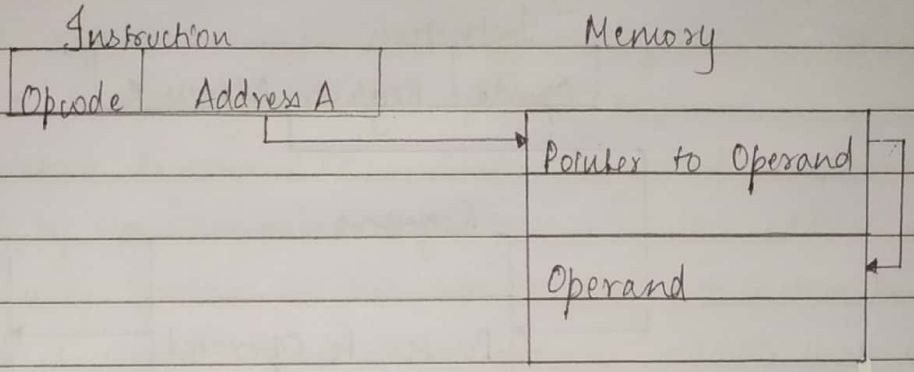
- Address field contains address of operand.
- Effective Address (EA) = Address field (A)
- Eg - ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand.
- Single memory reference to access data.
- No additional calculations to work out effective address.
- Limited address.



Indirect Addressing

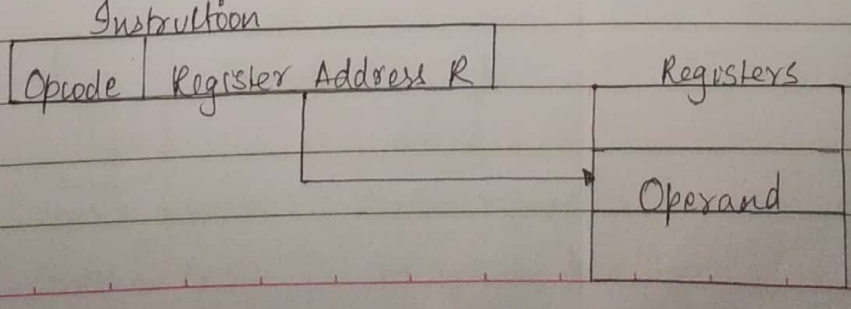
- Memory cell pointed to by address field contains the address of (pointer to) the operand.
- EA = (A)
  - Look in A, find address (A) and look there for operand
- Eg - ADD(A)
  - Add contents of cell pointed to by contents of A to accumulator.

- Large address space
- 2^n where n = word length
- May be nested, multilevel, cascaded
  - eg, EA = (((A)))
- Multiple memory accesses to find operand hence slower.



Register Addressing

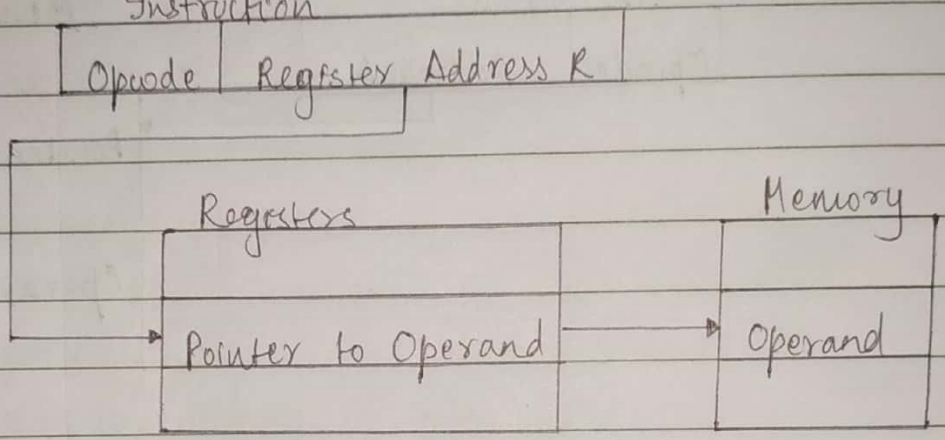
- Operand is held in register named in address field.
- EA = R
- Limited number of registers.
- Very small address field needed
  - Shorter instructions
  - Faster instruction fetch
- No memory access
- Very fast execution
- Very limited address space
- Multi registers helps performance
  - Requires good assembly programming or compiler writing.
  - register int a;
- Direct addressing



Date

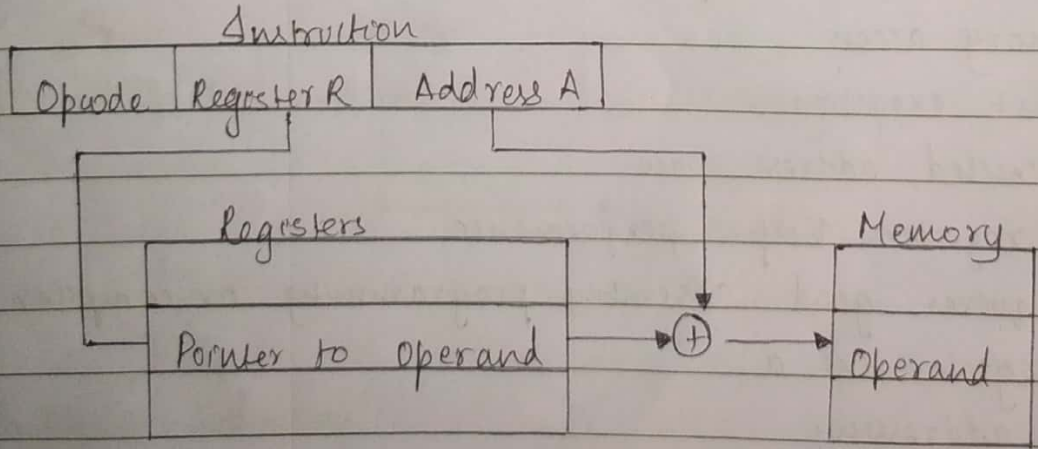
Register Indirect Addressing

- C.f. Indirect addressing
- EA = R
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory access than indirect addressing



Displacement Addressing

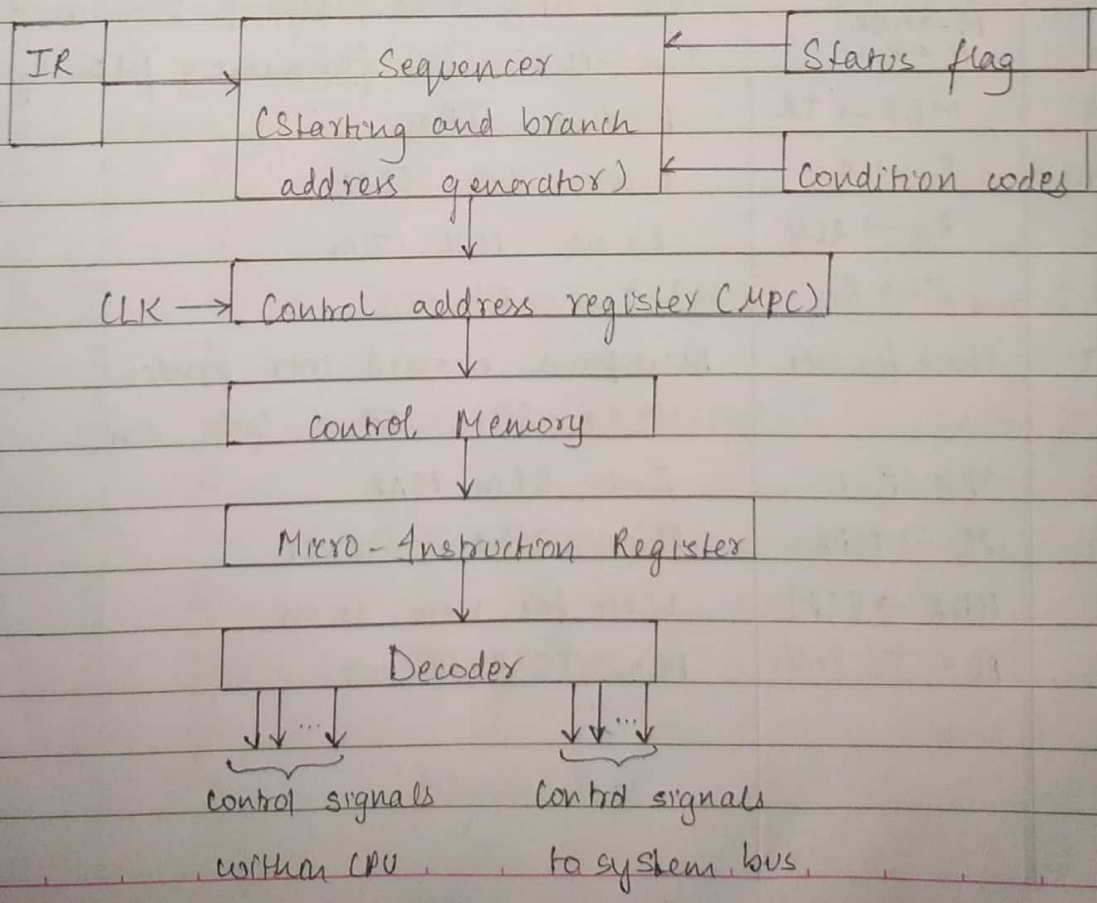
- EA = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement
  - or vice versa



Page No.
Date

Q5. Explain Microprogrammed Control Unit in detail.

→ • Micro programmed control unit generates control signals based on the microinstructions stored in a special memory called as the control memory.
• Each instructions points to a corresponding location in the control memory that loads the control signals in the control register.
• The control register is then read by a sequencing logic that issues the control signals in a proper sequence.
• The Instruction Register (IR), Status flag and condition codes are read by the sequencer that generates the address of the control memory location for the corresponding instruction in the IR.
• This address is stored in the Control address register that selects one of the locations in the control memory having the corresponding control signals.
• These control signals are given to the microinstruction register, decoded and then given to the individual components of the processor and the external devices.



Q6. Write microprogram to 1) ADD R_1, R_2 2) MUL R_1, R_2

1)

T-state	Operation	Microinstructions
T_1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add, Z_{in}
T_2	M → MBR	Z_{out}, PC_{in}, Wait for memory fetch cycle
	PC ← PC + 1	
T_3	MBR → IR	MBR_{out}, IR_{in}
T_4	R_1 → x	R_1 out, X in, CLRC
T_5	R_2 → ALU	R_2 out, ADD, Z_{in}
T_6	Z → R_1	Z_{out}, R_{1in}
T_7	Check for intr	Assumption enabled intr pending
		CLR X, SET C, SP_{out}, SUB, Z_{in}
T_8	SP ← SP − 1	Z_{out}, SP_{in}, MAR in
T_9	PC → MDR	PC_{out}, MDR in, WRITE
T_{10}	MDR → [SP]	Wait for mem access
T_{11}	PC ← IS Raddr	PC in IS Raddr out

2)

T-State	Operation	Microinstructions
T_1	PC → MAR	PC_{out}, MAR_{in}, Read, Clear y, Set C_{in}, Add Z_{in}
T_2	M → MBR PC ← PC + 1	Z_{out}, PC_{in}, Wait for memory fetch cycle
T_3	MBR → IR	MBR_{out}, IR_{in}
T_4	R_1 → x	R_1 out, X in, CLRC
T_5	R_2 → ALU	R_2 out, MUL, Z_{in}
T_6	Z → R_1	Z_{out}, R_{1in}
T_7	Check for intr	Assumption enabled intr pending
		CLR X, SET C, SP_{out}, SUB, Z_{in}
T_8	SP ← SP − 1	Z_{out}, SP_{in}, MAR
T_9	PC → MDR	PC_{out}, MDR in, WRITE
T_{10}	MDR → [SP]	Wait for mem access
T_{11}	PC ← IS Raddr	PC in IS Raddr out

Date

Q7. Write short note on 1) characteristics of memory

→ 1) Location : Internal - processor, mm, cache memory
External - Optical, Magnetic tapes.

2) Capacity : Number of words, bytes Internal - Bytes
External - Number of words

3) Unit of Transfer : Word, Block
- It is the no. of bits read out of or written into memory at a time.
- For internal memory the unit of transfer is equal to number of electrical lines into and out of the memory module (eg. word)
- for external memory unit of transfer is blocks.

4) Access Method : a) Sequential - eg. Tape
b) Random access - Eg. MM, cache
c) Direct Access (combination of random access and sequential access) Eg - Disk
d) Associative Access

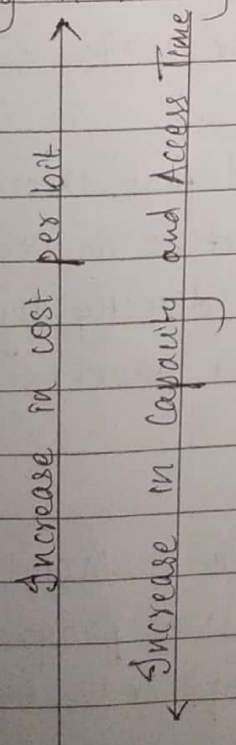
5) Performance : Access time, Memory cycle time, transfer rates.

6) Physical Type (Semiconductor, magnetic, Optical)

7) Physical Characteristics (Volatile, Non volatile)

8) Organisation (Erasable / Non erasable)

2) Memory Hierarchy



CPU Registers — Level 0

Cache Memory (SRAMs) — Level 1

Main Memory (DRAMs) — Level 2

Magnetic Disk (Disk Storage) — Level 3

Optical Disk
Magnetic Tape
} Level 4

c) Cache Coherence

- In a single CPU system two copies of same data, one in cache memory and another in MM may become different. This data inconsistency is called as cache coherence problem.
- There are four different approaches to prevent data inconsistency i.e. to protect cache coherency.

1) Bus watching (Snooping) - Each CPU cache modifies shared memory when shared variable gets update.
2) H/W Transparency - When processor routes through the cache then it routes through all remaining caches and MM.
3) Non-Cacheable Memory - MM is partitioned into cacheable and non cacheable memory.
4) Cache Flushing - To avoid data inconsistency, a cache flush writes any altered data into MM and caches in the system are flushed before a device writes to shared memory.

d) Types of Memory

1. CPU Registers - High speed registers for temporary storage of instruction and data. Capacity of register file is 32 words.
   - They form a register file for storing data as it is processed.
   - Each register can be accessed i.e. read or write within a single clock cycle.
2. Main (Primary) Memory - It is large and fast external memory stores data and programs that are active in use.
   - Storage locations in MM are addressed directly by the CPU load and store instruction. Access time of 5 or more clock cycles.
   - Load → memory to register
   - Storage → register to memory
3. Secondary Memory - This memory type is larger in capacity but slower than MM. It stores a system programs, large data files that is not continuously required by CPU.
   - Representative Technologies for secondary memory are magnetic hard disks and CDROMs.

- Capacity : many gigabites.
- Access time : Measured in milliseconds.

4. Cache - Cache capacity is less than the MM.
- Most computers have another level of IC memory - sometimes several such levels called cache memory.
- Access time of 1 to 3 cycles. Cache is much faster than MM.
- Caches are essential components of high-performance computer that aims to make CPI <= 1 (Cycles per instruction)

Q8. Explain different mapping techniques used in memory organisation.

→ a) Direct Mapping

Maps each block of main memory into only one possible cache line as : $i = j \bmod m$ where : i = cache line number ; m = no. of lines in the cache ; j = main memory block no. ;

Advantage : simple and inexpensive to implement

Disadvantage : There is a fixed cache location for any given block ; if a program happens to reference words repeatedly from two different blocks that map into the same line ; then the blocks will be continually swapped in the cache ; hit ratio will be low (a.k.a. thrashing).

Tag	Line Offset	Word Offset

b) Associative Mapping

Permits each block to be loaded into any cache line. There are only two fields in the address as tag and word. The tag uniquely identifies block of memory from where the line has been copied into the cache memory.

Advantage : Flexibility as to which block to replace when a new block is read into the cache ;

Disadvantage : Complex circuitry required to examine the tags of all cache lines in parallel.

Tag	Word Offset

c) Set Associative Mapping

Cache contains of a no. of sets, each consisting of a no. of lines.

Tag	Set Offset	Word Offset

$m = v \times k$ where: i = cache set no;

$i = j \bmod v$ j = MM block no;

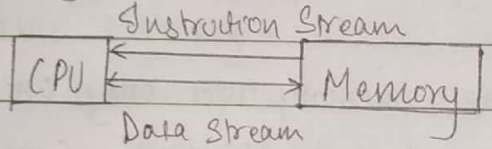
v = no. of sets; m = no. of lines in the cache;

k = no. of lines in each set.

Q9. Explain Flynn's Classification in detail.

→ Instruction Stream: In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established

Data Stream: The flow of operands between processor and memory bi-directionally is called as "Data stream".



Data Stream →

Instruction Stream	Single	Multiple
Single	SISD Uniprocessors	SIMD Pipelined vector processors
Multiple	MISD Systolic arrays	MIMD Multiprocessors Multi-core Multicomputers

1) Single Instruction Single Data stream (SISD) - It represents the organization of a single computer containing a control unit, a processor unit, a memory unit. Instructions are executed sequentially, and the system may or may not have internal II processing capabilities. Instructions are decoded by the control Unit and then the control Unit sends the instruction to the processing units for execution.

2) Multiple Instruction and Single Data stream (MISD) - In this Multiple Processing elements are organized under the control of multiple control unit. Each CU is handling one instructions stream and processed through its corresponding processing element.

But each processing element is processing only a Single Data Stream at a time. All processing elements are interacting with common shared memory for the organisation of the single data stream.

3) Single Instruction Multiple Data stream (SIMD) - It represents an organization that includes many processing units under the supervision of a common CU. Master instruction work on vector of operand.

- All the processors receive the same instruction from the control unit but operate on different items of data.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.
- Single instruction is applied to a multiple data item to produce the same output.

4) Multiple Instruction and Multiple Data stream (MIMD) -

i) Shared Memory MIMD architecture

- Creates a group of memory modules and processors.
- Any processor is able to directly access any memory module by means of an interconnection network.
- The group of memory modules outlines a universal address space that is shared between the processors.

ii) Distributed Memory MIMD architecture

- Clones the memory / processor pairs, known as a processing element (PE), and links them by using an interconnection network
- Each PE can communicate with others by sending messages.
- By providing every processor its own memory, the distributed memory architecture bypasses the downsides of the shared memory architecture. A processor may only access the memory that is directly connected to it.
- In case a processor requires data that resides in the remote processor memory, then the processor should send a message to the remote processor, requesting the required data.

Q10. Explain Amdahl's law.

→ This law is used for a fixed workload parallel system. There are systems that have fixed computational workload. Hence as the no. of processing elements or processors increase, the time required for execution must reduce.

The part that is parallelizable will require lesser and lesser time when the no. of processor increases. When there are two processors, the sequential time remains the same, while the parallelizable time requires half the time i.e. one unit time, as there are two processors.

Q11. Explain different types of pipeline hazards along with examples.

→ 1) Structural Hazard (Resource Conflicts)

- This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

Example:

Instruction / Cycle	1	2	3	4	5
I_1	IF (Mem)	ID	EX	Mem	WB
I_2		IF (Mem)	ID	EX	Mem
I_3			IF (Mem)	ID	EX
I_4				IF (Mem)	ID

In cycle 4, instructions I_1 and I_4 are trying to access same resource (Memory) which introduces a resource conflict. To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available. This wait will introduce stalls in the pipeline as shown before:

Cycle	1	2	3	4	5	6	7	8
I_1	IF (Mem)	ID	EX	Mem	WB			
I_2		IF (Mem)	ID	EX	Mem	WB		
I_3			IF (Mem)	ID	EX	Mem	WB	
I_4				–	–	–	IF (Mem)	

To minimize structural dependency stalls in the pipeline, we use a hardware mechanism called Renaming. According to renaming, we divide the memory onto two independent modules used to store the instruction and data separately called Code Memory (CM) and Data Memory (DM) respectively. CM will contain all the instructions and DM will contain all the operands that are required for the constructions.

2) Control Dependency (Branch Hazards)

- This type of dependency occurs during the transfer of control instructions such as CALL, JMP etc. On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline. Due to this, unwanted instructions are fed to the pipeline.
- Consider the following sequence of instructions in the program:

100 : I_1
101 : I_2 (JMP 250)
102 : I_3
⋮
250 : BI_1

Expected output : $I_1 \rightarrow I_2 \rightarrow BI_1$

Instruction/ Cycle	1	2	3	4	5	6
I_1	IF	ID	EX	MEM	WB	
I_2		IF	ID	EX	MEM	WB
			(PC: 250)			
I_3			IF	ID	EX	Mem
I_4				IF	ID	EX

Note: Generally, the target address of the JMP instruction is known after ID stage only.

Output Sequence : $I_1 \rightarrow I_2 \rightarrow$ Delay (stall) $\rightarrow BI_1$

As the delay slot performs no operation, this output sequence is equal to the expected output sequence. But this slot introduces stall in the pipeline.

Solution for control dependency Branch Prediction is the method through which stalls due to control dependency can be eliminated. In this at 1st stage prediction is done about which branch will be taken. For branch prediction Branch penalty is zero.

Branch Penalty: The no. of stalls introduced during the branch operations in the pipelined processor is known as branch penalty.

3) Data Dependency (Data Hazard)

- It occurs when there is a conflict in access of an operand location.
- Eg - Two instructions I_1 and I_2
- I_2 dependent on I_1

Consider A = 10

$I_1 = A \leftarrow A + 5$

$I_2 : B \leftarrow A \times 2$

3 types of data hazards

a) RAW hazard occurs when instruction J tries to read data before instruction I writes it.

Example: I: $R_2 \leftarrow R_1 + R_3$

J: $R_4 \leftarrow R_2 + R_3$

b) WAR hazard occurs when instruction J tries to write data before instruction I reads it.

Example: I: $R_2 \leftarrow R_1 + R_3$

J: $R_3 \leftarrow R_4 + R_5$

c) WAW hazard occurs when instruction J tries to write output before instruction I writes it.

Example: I: $R_2 \leftarrow R_1 + R_3$

J: $R_2 \leftarrow R_4 + R_5$

- WAR and WAW hazards occur during the out-of-order execution of the instructions.

Page No.
Date

Q12. Design 4 stage instruction pipeline with diagram.

Instruction / Cycle	1	2	3	4
I_1	IF (Mem)	ID	EX	WB
I_2		IF (Mem)	ID	EX
I_3			IF (Mem)	ID
I_4				IF (Mem)

Q13. Numericals based on Mapping Techniques