a) Consider a CPU's base CPI = 1.2, processor speed = 2 GHz, main memory access time = 100 ns, first level cache miss rate per instruction = 10%, cost of second level cache miss = 25 cycle, and global miss rate with second level cache = 4%. What will be the effective CPI if-

i) just primary cache is used and

ii) With the second level cache implemented?

i)

miss penalty = $\frac{100}{0.5}$

= 200 cycle

Effective CPI = 1.2 + (0.10 × 200)

= 21.2

clock cycle = $\frac{1}{2 \times 10^9}$

= $\frac{1}{2 \times 10^9} \times 10^9$ ns

= 0.5 ns/cycle

ii)

L1 miss L2 hit, 10% - 4% = 6% = 0.06

Stall cycle from (L1 miss, L2 hit)

$= 0.06 \times 25$ cycle

$= 1.5$ CPI

L1 miss L2 miss, 4% = 0.04

Stall cycle from (L1 miss, L2 miss)

$= 0.04 \times 25$

$= 9$ CPI

∴ Effective CPI = 1.2 + 1.5 + 9

$= 11.7$

Multi level cache example

Given,

CPU base CPI = 1

clock rate = 4 GHz

miss rate/instruction = 2%

main memory access = 100 ns

we know, with just primary (level 1) cache

miss rate = 2%.

$$\text{clock rate} = \frac{1}{\text{clock cycle time}}$$

$$\therefore \text{clock cycle time} = \frac{1}{\text{clock rate}}$$

$$= \frac{1}{4 \times 10^9} \text{ s}$$

$$= \frac{1}{4 \times 10^9} \times 10^9 \text{ ns}$$

$$= 0.25 \text{ ns/cycle}$$

$$\text{miss penalty} = \frac{\text{Access time}}{\text{clock cycle}} = \frac{100 \text{ ns}}{0.25 \text{ ns/cycle}}$$

$$= 400 \text{ cycle}$$

$$\text{Effective CPI} = 1 + (0.02 \times 400) = 9$$

Given,

I-cache miss rate = 2 %

D- " " = 4%

Miss penalty = 100 cycles

Base CPI (ideal cache) = 2

Load & stores are 36% of instructions

We know,

miss cycle per instruction,

I-cache = $0.02 \times 100 = 2$

D-cache = $0.04 \times 0.36 \times 100 = 1.44$

Actual CPI = Base CPI + I-cache + D-cache

$= 2 + 2 + 1.44$

$= 5.44$

Ideal CPU / Speed UP = $\frac{5.44}{2} = 2.72$ times faster.

b) Differentiate among programmed I/O, interrupt driven I/O, and direct memory access (DMA). Explain operation of direct memory access (DMA).

Ans: Three techniques are possible for I/O operations.

Programmed I/O: In programmed I/O the cpu handles the entire I/O operations and waits for it to finish. It is least efficient. cpu time is wasted waiting. Data transfer done by CPU. CPU is fully involved during the whole process.

Interrupt driven I/O: CPU starts I/O and continues other task; gets interrupted when done. It is better than programmed I/O because CPU can do other work during I/O. Data transfer

done by cpu after interrupt. cpu involved only at start and end.

Direct Memory Access (DMA): In DMA the cpu starts I/O, then DMA controller handles the data transfer. It is most efficient. cpu is free while DMA transfers data. data transfer done by DMA controller directly between memory and I/O. cpu involved only to start and receive final signal (interrupt).

operation of direct memory access (DMA):

DMA allows data to be transferred between memory and I/O devices without cpu involvement. The cpu initiates the process by providing the DMA controller with operation type, addresses and data size. The DMA controller then takes control of the bus and perform

the transfer. After completion, it interrupts the CPU to signal the end. This improves system performance by reducing CPU workload, especially for large transfers.

b) What is an interrupt? Draw the state diagram of a typical instruction cycle with interrupts.

Ans: An interrupt is a mechanism by which other modules (like I/O device or memory) can interrupt the normal processing of the processor. Interrupts are implemented to improve processing efficiency. This is particularly important because most external devices operate much slower than the processor. Without interrupts, the processor would have to pause and remain idle while waiting for slower devices (like printer) to complete an operation, which is a wasteful use of its processing time.

Classes of interrupts:

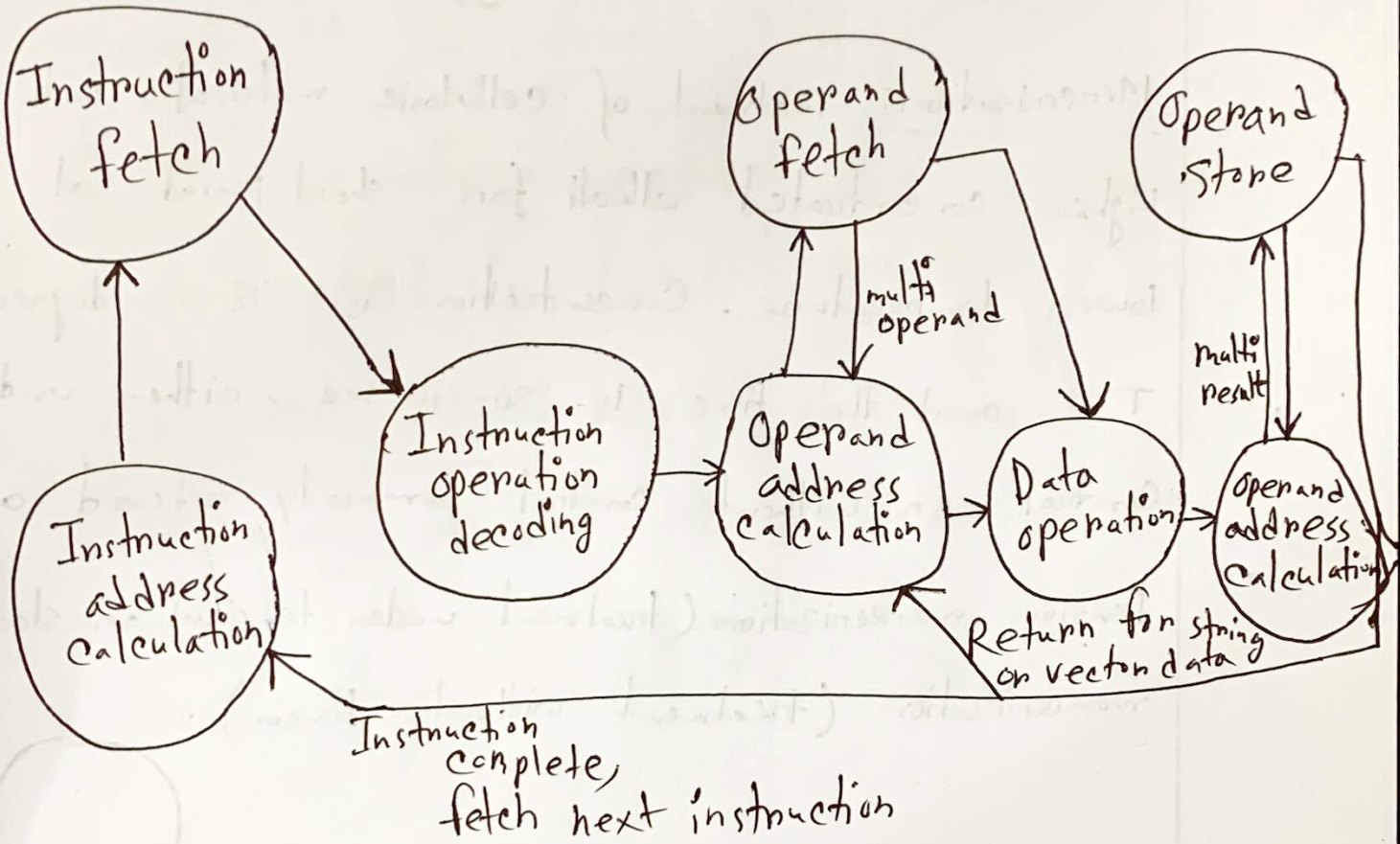
program: Generated by a condition that occurs as a result of an instruction execution.
Example: Arithemetic overflow, division by zero.

Timer: Generated by a timer with in the processor. This allows the operating system to perform certain functions on a regular basis.

I/O: Generated by an I/O controller to signal the normal completion of an operation or to signal a variety of error condition.

Hardware failure: Generated by a failure such as power failure or memory parity error.

Instruction cycle State Diagram



Instruction address calculation: Determine the address of the next instruction to be executed.

Instruction fetch: Read instruction from its memory location into the processor.

Instruction Operation decoding: Analyze instruction to determine type of operation to be performed and operand to be used.

Operand fetch: Fetch the operand from memory or read it in from I/O.

Data Operation: Perform the operation indicated in the instruction.

Operand Store: Write the result into memory or out to I/O.

2. a) Explain the mechanisms to handle multiple interrupts.

b) Using a block diagram, explain a typical instruction cycle with interrupts.

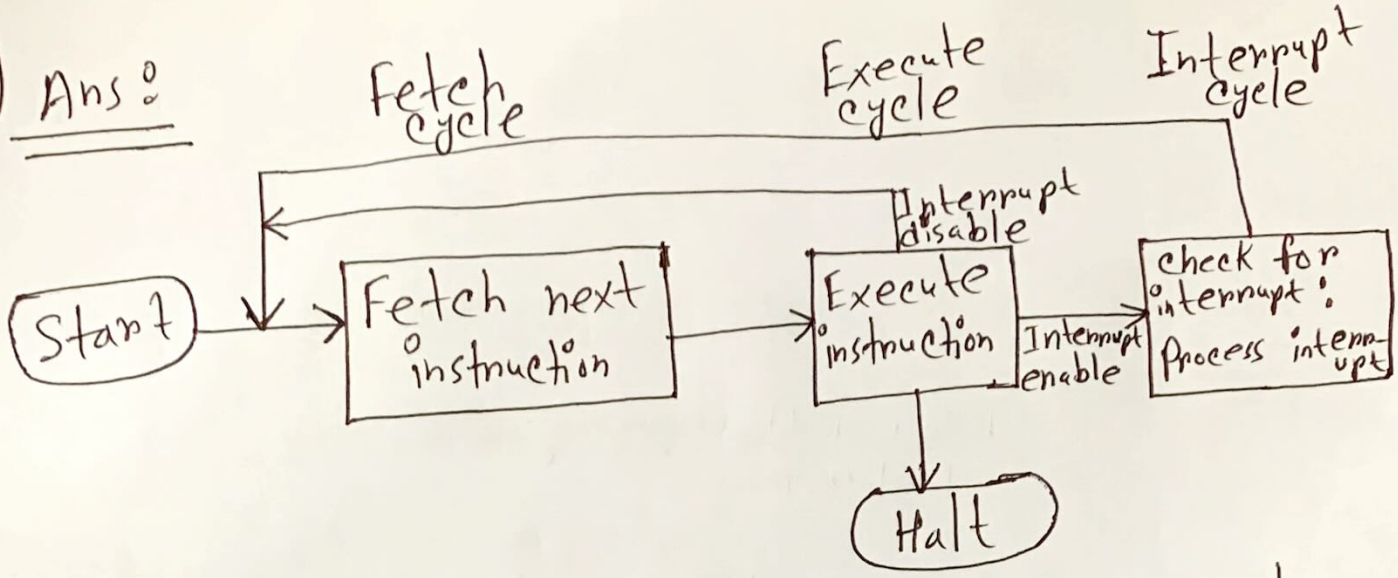
c) Differentiate between centralized and distributed bus arbitration.

a) Ans: Two mechanisms help manage and organize multiple interrupt requests efficiently.

1) Disable interrupts: When the CPU is handling one interrupts, it can temporary disable other interrupts. this ensures sequence (one by one) interrupts handling.

2) Interrupt priority level: Each interrupt can be assigned a priority. A higher priority interrupt can interrupt a lower priority interrupts. This helps handle more important task first.

b) Ans:



Instruction cycle with Interrupts

Explain:

1. Start: The process begins here.
2. Fetch cycle: Fetch next instruction: The process retrives the next from the memory.
3. Execute cycle: Execute instruction: The process performs the action required by the fetch instructions.
4. Interrupt check & Handle: After executing an instruction, the Interrupts Disable/Enable status is considers.

check for interrupts; Proces interrupts:

The process check if any interrupts signals are pending.

* if no interrupts are pending, the processor loops back to the fetch cycle.
* if an interrupts is pending, the processor initiates the interrupt cycle.

5. Halt: The process can eventually reach a halt state.

Interrupt cycle: If an interrupts is pending:

→ The processor suspends execution of the current program.

→ It saves its content. This means saving the address of the next instruction that was supposed to be executed.

→ It now sets the program counter to the starting address of an interrupt handler routine. This routine is

specific piece of code designed to deal with the interrupt.

→ The processor then process the interrupt by executing the interrupt handler routine.

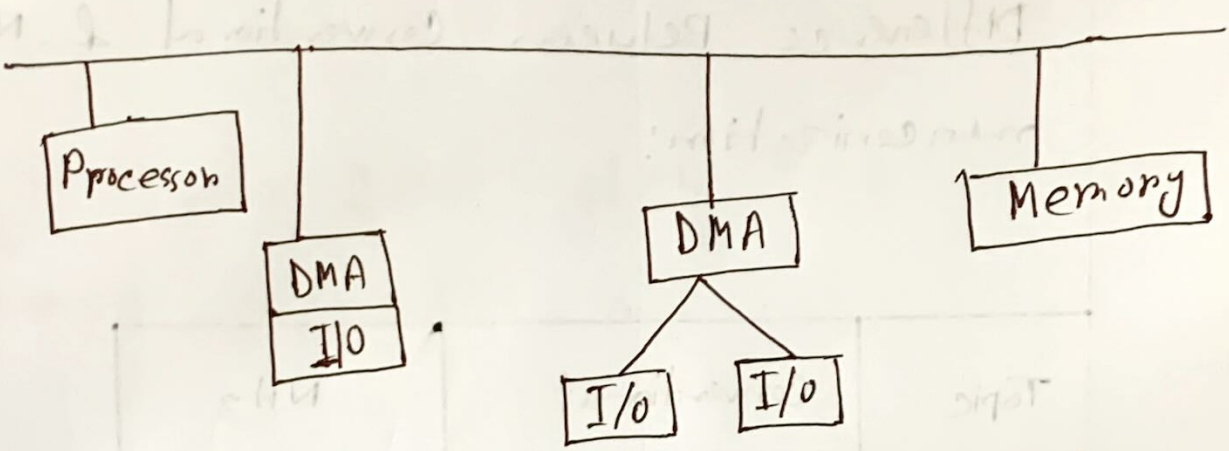
→ When the interrupt handler is completed, the processor resumes execution of the user program at the point of Interruption using saved context..

c]

Centralized Arbitration	Distributed Arbitration
A single device (like a bus controller or CPU) controls who gets access to the bus	No single controller; all devices participate in deciding who gets access.
Simple to implement	More complex because each module needs its own logic
Usually faster because decisions are made by one device	Slower due to communication between modules.

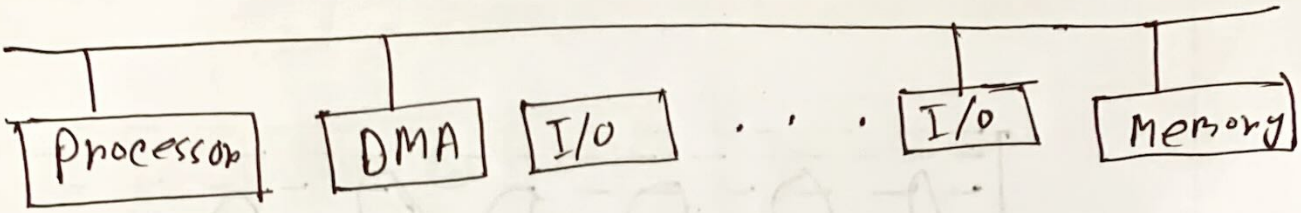
A centralized system offers simplicity and efficiency but can create bottlenecks if the arbiter fails. Distributed arbitration increases system resilience and scalability but requires more complex coordination among modules.

2. a) Draw and explain the single bus, integrated DMA-I/O configuration.

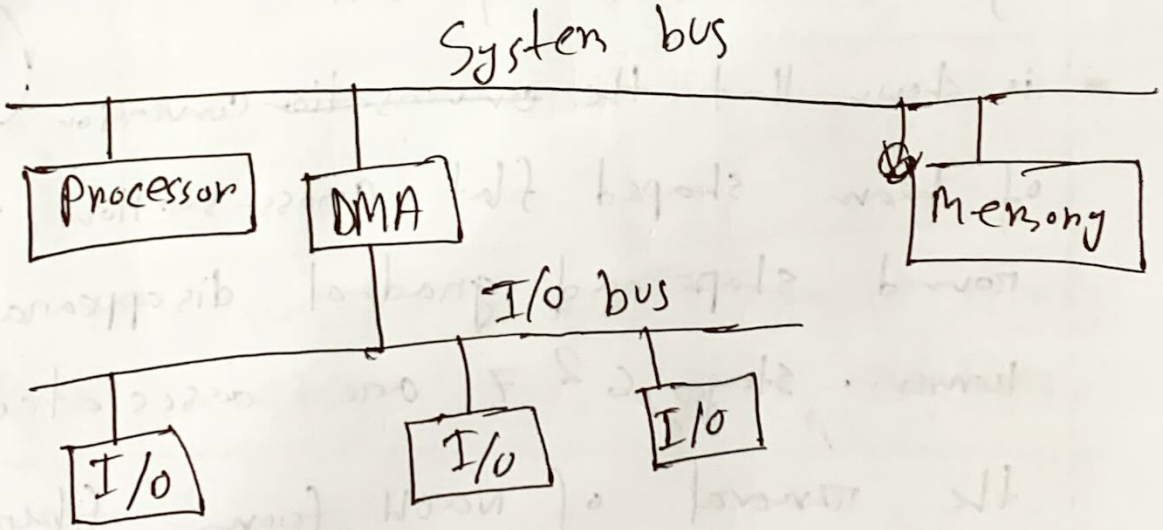


⇒ This configuration features a single system bus that is shared by the processor(CPU), memory, and I/O devices. The direct memory access (DMA) capability is integrated, often meaning the DMA logic is part of the I/O interface or a dedicated DMA controller that manages I/O operations.

* Each transfer uses bus once: DMA to memory
* CPU suspended once.



* I/O to DMA then DMA to memory
* CPU suspended twice

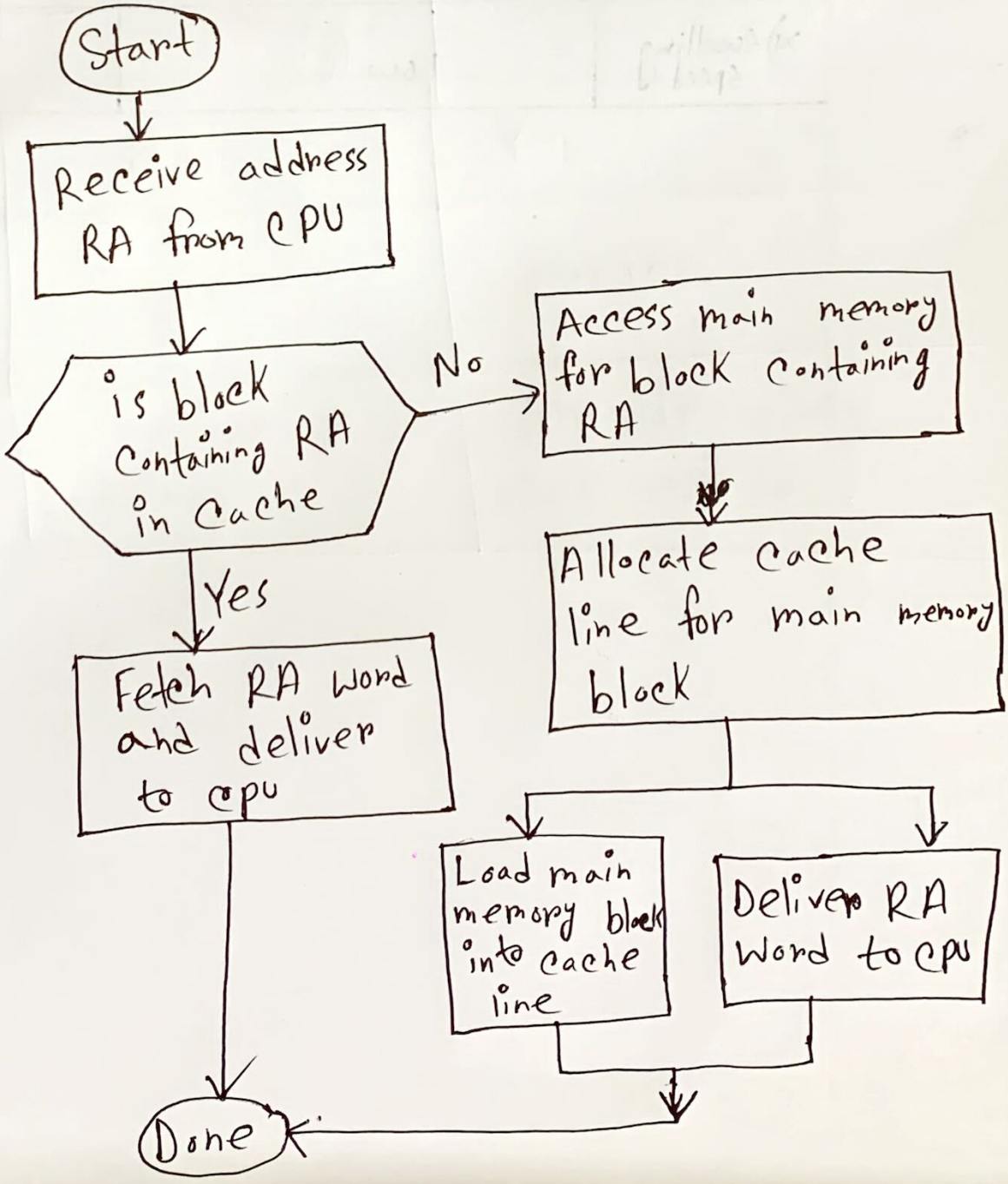


* Each transfer uses bus once: DMA to memory

* CPU suspended once.

b) Explain typical cache read operation with a flowchart. Differentiate between Write-back and Write-through strategies during cache Write.

Ans: Cache Read operation flowchart and explanation given below:



Differentiate between Write-back and Write-through.

Feature	Write-Through	Write-Back
Write on Hit	update both cache and memory	Update cache only
Write on miss	Write Allocate or No Write allocate	Always Write allocate
Memory traffic	Higher (Writes go to memory)	Lower (Writes go to memory only on eviction)
Dirty bit used?	No	Yes
Implementation Complexity	Simpler	Harder
Use of Write buffer	To reduce Write latency	To handle dirty block evictions.