PRE – REQUISITES

The power of 2 –

Unit	Time	Bit or Byte
K (Kilo)	10^3	2^{10}
M (Mega)	10^6	2^{20}
G (Giga)	10^9	2^{30}
T (Tera)	10^{12}	2^{40}

Computer Architecture – Conceptual Design and Fundamental Operational Structure. The architecture design starts with a detailed CPU design (instructions and pin – diagram).

Computer Organization – It is the implementation of architecture. Deals with the physical devices and interconnections. It aims to improve performance.

Computer Architecture	Computer Organization
• CPU Design	• I/O Organization
• Instructions	• Memory Organization
• Addressing modes	• Performance Improvement
• Data format	

DATA FORMATS

Data in System is represented in binary.

- Numbers
  - Fixed Point
    - Signed
      - 1's comp
      - 2's comp
      - Signed – magnitude
    - Unsigned
  - Floating Point
- Characters
  - American Standard Code for Information Interchange (ASCII)
  - Extended Binary Coded Decimal Interchange Code (EBCDIC)

COMPONENTS OF A COMPUTER

There are many components of a computer –

Central Processing Unit

This is the brain of the computer and it executes instructions. It has two components –

- Control Unit – This gives the commands and instructions to be performed
- Arithmetic Logic Unit – This blindly performs the operations provided by the control unit.

Memory

This is where the data is store. It is of two types –

- Primary (Main) – RAM/ROM
- Secondary (Auxiliary) – HDD/SSD

The Memory is divided into small sized data cells and each cell has a unique address.

I/O Devices

These are the ports to exchange data from the environment. These are of three types –

- Input devices – Mic, Keyboard, Camera etc.
- Output devices – Screen, speaker, printer etc.
- Storage – Pen drives, Floppy, CD etc.

System Buses

These are a group of communication lines that are used to enable in – system communication. There are different buses reserved for different types of data –

- Address Buses – Transmits address related information. This bus is uni-directional and it sends address information from CPU to Memory.
- Data Buses – Carries the data from different components. It is a bi-directional bus.
- Control Buses – Carries control instructions. It is also bi – directional.

CPU REGISTERS

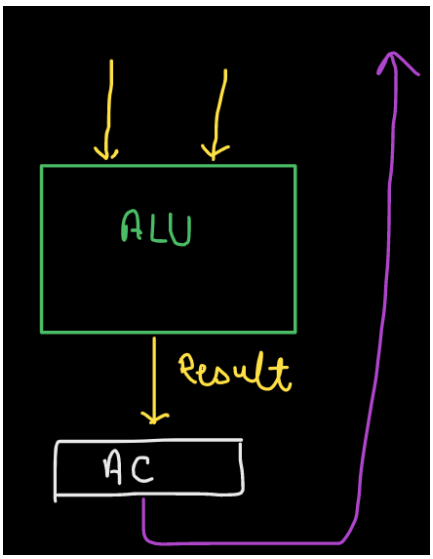
CPU performs operations on data and this data is stored in registers. There are two main types of CPU registers –

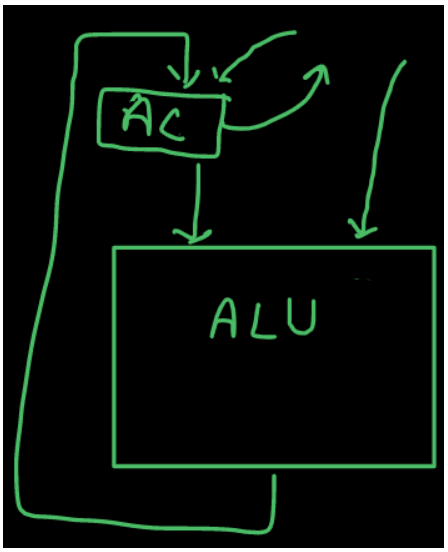
- **General Purpose Registers (GPRs**) – These store the general data.

- **Special Purpose Registers** – These store special data for special operations. Every special purpose register is used for a set function. For eg – Accumulators, Program Counter, Instruction Register, Program Status Word etc.

ACCUMULATOR

This is used to store the output of an ALU and sometimes one of the operands for the ALU. The result of the ALU is stored in accumulator and then the rest of the systems use the value in the accumulator to perform other operations. There is only 1 accumulator in a computer system.

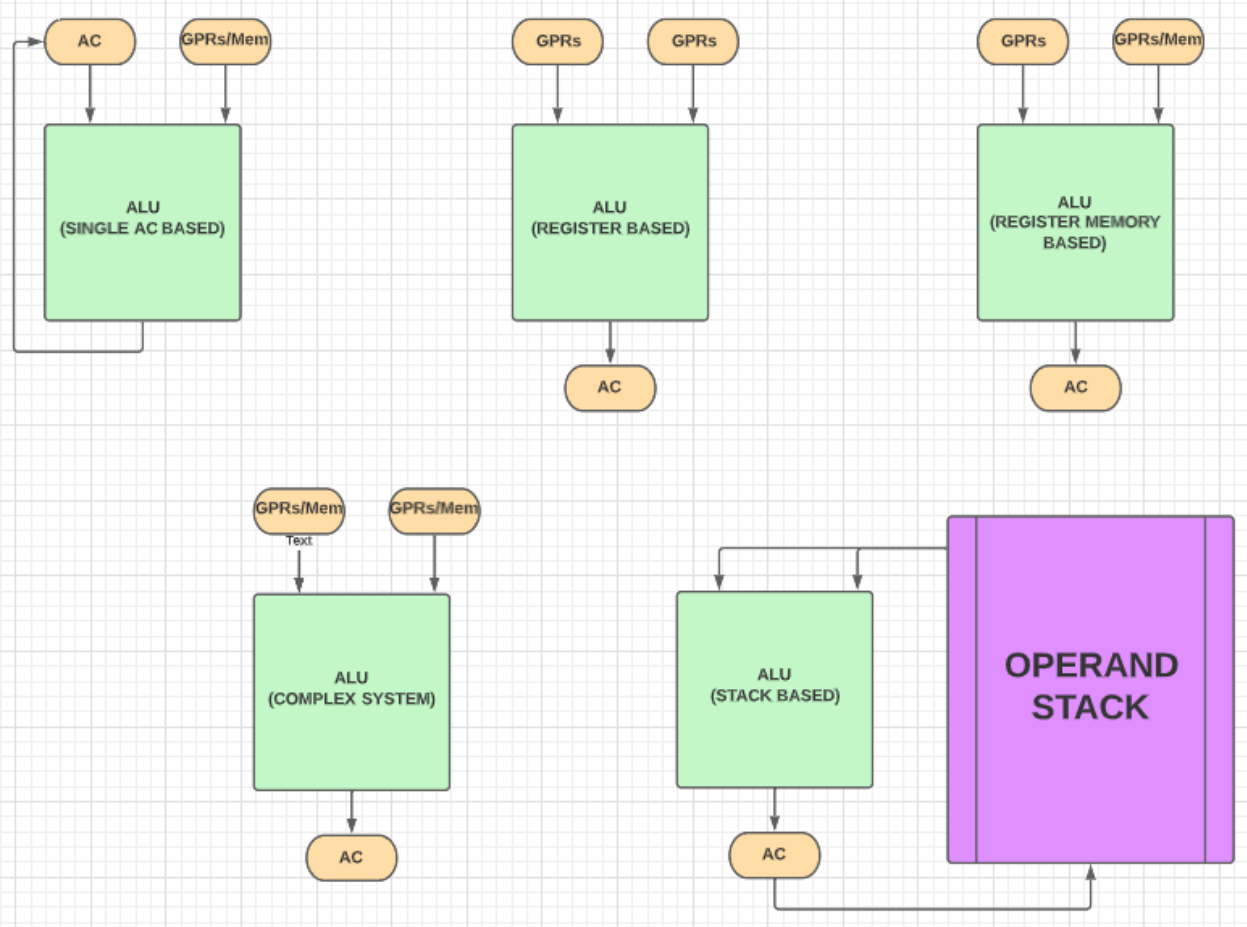




Two different types of architectures – AC stores the result of the ALU and AC stores the operand as well.

TYPES OF ARCHITECTURES BASED ON ALU INPUT

TYPE	**ALU Operand 1**	**ALU Operand 2**	**ALU Output**
Single AC Based	AC	GPRs/Mem	AC
Register Based	GPRs	GPRs	AC
Register – Memory Based	GPRs	GPRs/Mem	AC
Complex System	GPRs/Mem	GPRs/Mem	AC
Stack Based (**NOT IN USE**)	Operand Stack	Operand Stack	AC



TYPE OF ARCHITECTURES BASED ON INPUT SIZE

- **32-bit CPU architecture –** The inputs are of size 32 – bits.
- **64-bit CPU architecture –** The inputs are of size 64 – bits.

NOTE – The CPU architecture is based on the **CPU word size** which is independent of **Memory word size.** So even if we use 32 – bit architecture and the size of each memory cell is just 4 bits, the system would work as these two word sizes are independent of each other.

PROGRAM COUNTER

It stores the address of the next instruction to be executed.

INSTRUCTION REGISTER

It stores the instruction that is currently being executed.

STACK POINTER

The computer memory is stored as a stack (LIFO) structure and this pointer points to the top of the stack. It stores the address of the stack top.

FLAG/STATUS REGISTER

Stores the current state of the ALU result. It stores the follow information about the result –

- Sign of the result (1 bit)
- If the result if zero or non – zero (1 bit)
- If the result has a carry out (1 bit)

There are more flags as well and it depends on the MP/MC used.

DATA REGISTER (MDR)

Used to send or receive data from memory.

ADDRESS REGISTER (MAR)

It is used to send the address to the memory.

PROGRAM STATUS WORD

This is a register of size 1 word that stores the status of the current execution state. It stores info like parity flag, carry flag, overflow flag etc.

CPU READ OPERATION

1. CPU sends address to the memory via the address bus.
2. CPU enables the read control signal via the control bus.
3. Memory reads the content at the address mentioned.
4. Memory sends the data to the CPU via the data bus.

CPU WRITE OPERATION

1. CPU sends the address to the memory via the address bus.
2. CPU sends the data to the memory via the data bus.
3. CPU enables the write control signal via the control bus.
4. Memory writes the data to the mentioned address.

TYPE OF MEMORIES BASED ON ADRESSING MODES

The memory can be classified as two types –

- **Byte addressable (default) –** Every address can store 1 byte of data
- **Word addressable –** Every address can store 1 word of data. Size of word can vary.

NOTE

By default, the memory is considered to be byte addressable. If the question states the memory is word addressable, then the word size needs to be specified. Also, the **CPU word size** and the **Memory word size** need **NOT be the same.**

QUESTION

A CPU has 4 bytes instructions. A program (Instructions I1 to I200) starts at address 500 (in decimal). Find the address of following instructions:
1. I1
2. I5
3. I120

ANSWER

By default, we assume that this is a byte – addressable memory. This means that every instruction takes 4 cells/ 4 addresses. In that case, we already have **I1 = 500.** Now, for the In instruction, the address will be –

$$Address\ of\ In = [(n-1) * 4] + 500$$

Therefore, we have **I5 = 516** and **I120 = 976.**

Now, let us modify the question so that the memory is word addressable and the size of word is 4 Bytes and then 2 Bytes. In that case, we have

SIZE OF WORD	I1	I5	I120
4 BYTES	500	504	619
2 BYTES	500	508	738

QUESTION

A CPU has 4 bytes instructions. A program (Instructions I1 to I200) starts at address 500 (in decimal). What should be the PC value when instruction I6 will be executing in CPU?

ANSWER

The Program Counter stores the value of the next instruction. Therefore, when I6 is executing, the PC will have the address of I7. Since it is not explicitly mentioned, this is a byte – addressed memory. Therefore, **PC = I7 = 524.**

NOTE – In the above cases, let us assume that the memory was word – addressable and the word size was 8 bytes. In that case, the Memory will still only store 1 instruction per address. Multiple instructions can't be stored in a single address. Therefore, in this case, 4 Bytes of memory is waster per address.

QUESTION

Consider a memory of size 256GB. Calculate the number of cells and the minimum size of the address required if the memory is –

I. Byte Addressable
II. Word Addressable where the word size is 4B
III. Word Addressable where the word size is 16B

ANSWER

We have,

$$256GB = 2^8 * 2^{30} = 2^{38}\ bytes$$

Now, we get the following solution

QUESTION	NUMBER OF CELLS	MIN ADDRESS SIZE
Byte Addressable	$N = \frac{2^{38}}{1} = 2^{38}$	$n = \log_2 2^{38} = 38$
Word Addressable where the word size is 4B	2^{36}	36
Word Addressable where the word size is 16B	2^{34}	34

MICRO – OPERATIONS

These are the operations are executed on the values stored in the registers in the CPU. Every ALU operation is divided into a series of micro – ops. These micro-ops are represented using **Register Transfer Language (RTL).**

OPERATION	SYMBOL	EXPLANATION
Register Transfer	$R2 \leftarrow R1$	Value of R1 is copied to R2
Comma	$R2 \leftarrow R1\ ,\qquad R3 \leftarrow R4$	Value of R1 is copied to R2 and value of R4 is copied to R3 **simultaneously**

Memory Read	$DR \leftarrow M[Addr.]$	Transfer value from Memory (M) at address Addr to Data Register (DR).
Memory Write	$DR \rightarrow M[Addr.]$	Transfer value from Data Register (DR) to Memory (M) at address Addr.

QUESTION

Consider the following program segment. Here R1 and R2 are the general purpose register. Assume that the content of memory location 2000 is 41. All numbers are in decimal. After the execution of this program the value of memory location 2000 is?

Instructions	Operations
MOV R1, #5	R1 ← #5
MOV R2, (2000)	R2 ← M[2000]
SUB R2, R1	R2 ← R2 – R1
MOV (2000), R2	M[2000] ← R2
HALT	Stop

ANSWER

R1 = 5

R2 = M[2000] = 41

R2 = R2 – R1 = 41 – 5 = **36**

Now, value in R2 is moved back into memory space 2000. So, the memory space 2000 now has value 36.

INSTRUCTION

It is a group of bits that instructs the CPU to perform certain operation. The instruction is divided into two parts –

- The first group of bits specify the operation to be performed. These bits are called **Operation Code** or **OP-CODE.**
- The next group of bits specify the operands on which this operation needs to be performed.

$$OP - Code\ Size = \log_2(Number\ of\ distinct\ CPU\ operations)$$

INSTRUCTION SET ARCHITECTURE (ISA)

ISA is the collection of all instructions supported by the CPU. Therefore, we have

$$OP - Code\ Size = \log_2(Size\ of\ ISA)$$

TYPES OF INSTRUCTIONS BASED ON NUMBER OF OPERANDS

TYPE	NUMBER OF OPERANDS	CAN ALSO SUPPORT
3 – Address Instruction	3	2/1/0 Address Instructions
2 – Address Instruction	2	1/0 Address Instructions
1 – Address Instruction	1	0 Address Instructions
0 – Address Instruction	0	NA

QUESTION

Consider a digital computer which supports only 2-address instructions each with 14-bits. If address length is 5-bits then maximum and minimum how many instructions the system can support?

ANSWER

Every instruction is made up of 14 bits out of which we have 1 OP-Code and 2 operands. It is given that the address length aka operand address size is 5 bits each. Therefore, we have 10 bits of operand information. Remaining, we have 4 bits which can specify a **maximum of 16 operations** and a **minimum of 1 operation.**

QUESTION

A processor has 50 distinct instructions and 16 general purpose registers. Each instruction in system has one opcode field, 2 register operand field and a 10-bits memory address field. The length of the instruction is ____ bits?

ANSWER

$$OP - Code\ size = ceil(\log_2 50) = \mathbf{6}$$

$$Register\ (or)\ Operand\ Address\ Size = ceil(\log_2 16) = \mathbf{4}$$

$$Number\ of\ operands = \mathbf{2}$$

$$Memory\ Address\ Field = \mathbf{10}$$

Finally, we have

$$Size\ of\ instruction = 6 + (2 * 4) + 10 = \mathbf{24}$$

NOTE – The CPU systems can either have **fixed – length instructions** or **variable – length instructions.** The names are self – explanatory. In the fixed – length system, the CPU design is easy but there is wastage of memory as instruction size may not be exactly equal to the memory cell size. On the other hand, variable length system solves this issue with varying instruction size. The CPU design however is more complex.

For fixed length instructions, the op-code size will vary. While for variable length instructions, the op-code size will remain constant.

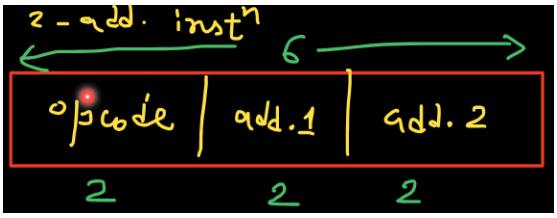
In practical cases, we can have a system that can support multiple address length systems. Let us take the question below –

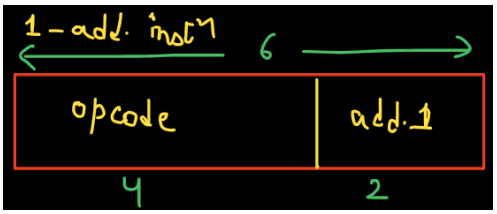
QUESTION

Consider a computer which supports only 2-address and 1-address instructions. Each instruction is of 6-bits and each address is of 2-bits. If there are 3 2-address instructions supported by the system then maximum number of 1-address instructions supported by system is?

ANSWER

Here, we have both 2 – address and 1 – address instructions present. According to the question, these are represented as –





In the above case, we are using a **fixed – length instruction** system since the size of the instruction remains 6 bits. Thus, the **op-code size is variable.** Now, let us assume the incoming instruction is something like **101100.** In this case, we can have –

TYPE	OP – CODE	ADDR 1	ADDR 2
2 – Address	10	11	00
1 - Address	1011	00	-

Thus, the CPU is in a state of confusion. It is not sure whether to treat this as a 2 – address or 1 – address instruction. In these cases, the CPU follows the procedure detailed below –

- First, the CPU finds out the smallest OP – Code size. In our case, the smallest OP – Code size is 2 bits.
- Then, the CPU checks the first 2 bits. If the system has an OP code corresponding to the 2 bits, then the CPU will treat this as a 2 – address instruction and go ahead.
- In case the 2 bits are not a valid OP – Code, then it proceeds to check the first 4 bits. If these bits are a valid OP – Code, then the CPU will treat this as a 1 – address instruction and go ahead.

Now, in our question, it is mentioned that there are **3 2 – address instructions** supported. Thus, out of the 4 OP – Codes available for 2 – address instructions only 3 are being used. Let us assume the system is using **00, 01 and 10 as valid 2 – address op – codes**. In that case, we have –

FIRST 4 BITS	VALID OP CODE	X – ADDRESS INSTRUCTION
0000	00	2
0001	00	2
0010	00	2
0011	00	2
0100	01	2
0101	01	2
0110	01	2
0111	01	2
1000	10	2
1001	10	2
1010	10	2
1011	10	2
1100	1100	1
1101	1101	1
1110	1110	1
1111	1111	1

In short, we have **3** possible **2 – address instructions** and **4** possible **1 – address instructions.**

QUESTION

Take the same question as the previous one. However, we have not mentioned the number of 2 – address instructions supported. How many 1 – address instructions are possible?

ANSWER

In this case, we don't have the number of 2 – address instructions supported. So, we assume the most basic case and take that at least 1 2 – address instruction is supported. Hence, there is a single 2 bit op – code used by 2 – address instructions.

Therefore, for 1 – address instruction op – codes, we have –

- First 2 bits = 3 possible combinations
- Next 2 bits = 4 possible combinations

Therefore, total number of 1 – address instructions supported = 4*3 = **12.**

QUESTION

Consider a system with 32-bit instructions and 12-bit addresses. If there are 254 2-address instructions and 8000 1-address instructions then maximum how many 0-address instructions can be formulated?

ANSWER

For 2 – address instructions

$$OP - Code\ length = 32 - (2 * 12) = 8\ bits$$

$$Possible\ 2 - address\ instructions = \ 2^8 = 256$$

Given that the system supports 254 2 – address instructions. Therefore,

$$Un - used\ combinations = \mathbf{2}$$

For 1 – address instructions

$$OP - Code\ length = 32 - 12 = 20\ bits$$

$$Possible\ combinations\ for\ 1^{st}\ 8\ bits = 2$$

$$Total\ possible\ 1 - address\ instructions = 2 * 2^{20-8} = 8192$$

Given that the system supports 8000 1 – address instructions. Therefore,

$$Un - usesd\ combinations = \mathbf{192}$$

For 0 – address instructions

$$OP - Code\ length = 32 - 0 = 32\ bits$$

$$Possible\ combinations\ for\ 1^{st}\ 20\ bits = 192$$

$$Total\ possible\ 1 - address\ instructions = 192 * 2^{32-20} = \mathbf{786432}$$

QUESTION

Consider there are 3 types of instructions in system:
1. Register Operand instructions: One opcode and 2 registers
2. Memory Operand instructions: One opcode, 1 register and 1 memory address
3. Immediate Operand Instructions: One opcode, 1 register and 1 immediate operand

Number of registers = 64
Number of bits in immediate operand = 10-bits
Memory size = 512Mbytes (byte addressable)

Total Instructions:
1. Reg Operand type: 10
2. Memory Operand type : 12
3. immediate Operand type : 4

Maximum and Minimum instruction length are?

ANSWER

We have,

$$Register\ Addr.Size = \log_2 64 = 6b$$

$$Memory\ Addr.Size = \log_2 512M = \log_2(2^9 * 2^{20}) = 29b$$

In this case, we need to find the minimum and maximum instruction length. Therefore, we have a **variable – length instruction** case. As discussed before, for variable – length instruction cases, we have a **fixed op – code length.**

$$Total\ number\ of\ instructions = 10 + 12 + 4 = 26$$

Therefore,

$$Size\ of\ op - code = \text{ceil}(\log_2 26) = 5b$$

Thus, the minimum and maximum instruction sizes are 17 and 40 bits respectively.

QUESTION

- In a simplified computer the instructions are:

$OP\ R_i, R_j$	- Performs $R_i\ Op\ R_j$ and stores the result in R_j
$OP\ m, R_i$	- Performs *val* $Op\ R_i$ and stores the result in R_i *val* denotes the content of memory location m
$MOV\ m, R_i$	- Moves the content of memory location m to register R_i
$MOV\ R_i, m$	- Moves the content of register R_i to memory location m

The computer has only two registers and OP is either ADD or SUB. Consider the following basic block:

```
t1 = a + b
t2 = c + d
t3 = e – t2
t4 = t1 – t3
```

Assume that all operands are initially in memory. The final value of the computation should be in memory. What is the minimum number of MOV instructions in the code generated for this basic block?

ANSWER

The basic idea here is that the operation OP is performed only on register values. Therefore, we need to move the values into a register and then perform the operation and then send the result back to memory.

Now, one thing to note is that in the question, it is said that the operation can be performed between two registers or between a memory element and a register, but not between memory elements. Plus, the final value needs to be stored in a memory location. Therefore, the series of operation becomes –

- $MOV\ b, R1$
- $ADD\ a, R1$
- $MOV\ d, R2$
- $ADD\ c, R2$
- $SUB\ e, R2$
- $SUB\ R1, R2$
- $MOV\ R2, x$

Therefore, we need **3 move operations.**

REGISTER SPILL

In a CPU architecture, we have a limited number of registers. Thus, when performing complex operations, it becomes difficult for the registers to store intermediate values. Hence, the registers move the intermediate values to some memory locations to clear up space. This is called register spill.

Question

Consider a register-memory architecture system (2-address instructions). For this system the following intermediate code is going to be converted in machine code. Minimum how many registers are required in system so that the code can run without register spill?
Consider first operand is always the register operand and it's the destination for operation too.

```
t1 = X + Y
t2 = t1 – Z
t3 = t1 + t2
t4 = M + t3
```

Assume X, Y, Z and M are memory operands

Answer

The flow of operations will be as follows –

$$X \rightarrow R1$$

$$R1 + Y \rightarrow R1$$

$$R1 \rightarrow R2$$

$$R1 - Z \rightarrow R1$$

$$R2 + R1 \rightarrow R2$$

$$M \rightarrow R1$$

$$R2 + R1 \rightarrow R2$$

Hence, we can see that with just 2 registers, we were able to compute without register spill.

Question

Take the same question above but perform the computation for an Accumulator Based architecture.

Answer

As we know, in an accumulator based architecture, the first operand is supposed to be the Accumulator value and the second operand can come from either GPRs or the memory. Keeping this in mind, we can write –

$$X \rightarrow AC$$

$$AC + Y \rightarrow AC\ (T1)$$

Since we need to preserve the value of T1, we store it in a register.

$$AC \rightarrow R1$$

$$AC - Z \rightarrow AC\ (T2)$$

In the next operation, T2 is the second operand. Since the accumulator can't be second operand, we store its value in another register and then use that instead. At the same time, the first operand needs to be T1 and therefore, we assign the value in R1 back to AC.

$$AC \rightarrow R2$$

$$R1 \rightarrow AC$$

$$AC + R2 \rightarrow AC$$

$$AC + M \rightarrow AC$$

Thus, we require **2 registers** to perform this operation without register spill.

EFFECTIVE ADDRESS

It is the address of an operand in a computation – type instruction. The address used in any instruction for computation is called an **effective address.**

BRANCH INSTRUCTION

It is basically a conditional instruction (for eg, if – else statement). There are three parts of a branch instruction –

- **Condition –** The condition that is checked by the instruction to determine the next step.
- **Branch taken –** This is the step taken in case the condition is true. The program jumps to the **target instruction** in this case.
- **Branch not taken –** This is the step taken in case the condition is false. The program resumes sequential operation in this case.

This is the case for conditional branch instruction. On the other hand, an unconditional branch instruction jumps to the target address without checking any condition. Statements like **goto** are examples of unconditional branch instructions and they jump to the target address every time.

NOTE – Effective address can be also explained as the target address in case of branch instructions.

TYPE OF INSTRUCTION	EFFECTIVE ADDRESS
Computation	Operand Address
Branch	Target Address

INSTRUCTION CYCLE

It is the whole process cycle for an instruction. It consists of the following phases –

- **Instruction Fetch* –** The CPU looks at the address of the Program Counter and then locates the instruction that needs to be executed. It then stores that instruction in the Instruction Register.
- **Instruction Decode* –** Now, the CPU looks at the op – code part of the instruction in the Instruction Register to decode the kind of operation that instruction is performing.
- **Effective Address Calculation –** The CPU then decodes the address part of the instruction to find the effective addresses and get the operand locations in the memory.
- **Operand Fetch –** The CPU then fetches the operands at the effective addresses and then stores them in the CPU GPRs.
- **Execution* –** This is the stage where the actual instruction is executed on the GPRs with the operand information by the ALU. The result of the operation is stored in the ACC.
- **Write Back Result –** The result from the GPRs is then written back into the memory.

Therefore, the execution of each instruction requires these six phases. Now, based on these phases, we have 3 cycles –

CYCLE	PHASES INVOLVED
Fetch	Instruction Fetch
Execution	Instruction Decode to Write Back Result
Instruction	All six phases

NOTE – The star – marked phases are mandatory for all instructions. However, the other phases are optional since instructions like goto or other branch instructions don't take any operands and also don't write back results.

BRANCH INSTRUCTION EXECUTION

Step 1 – Instruction Fetch

CPU brings the instruction from the address as mentioned in Program Counter and then copies it to the Instruction Register.

Step 2 – Instruction Decode

The CPU decodes the op – code and realizes that the instruction is a branch instruction.

Step 3 – Target Address Calculation

The CPU uses the remaining bits in the instruction and calculates the target address.

Step 4 – Operand Fetch

NOT REQUIRED

Step 5 – Execution

If the branch instruction is conditional, the CPU checks the condition. If the condition is true or if the instruction is unconditional, then the PC is updated with the target address. In case the condition is false, then the PC retains the address of the next instruction.

Step 6 – Write back result

NOT REQUIRED

BRANCH INSTRUCTION vs FUNCTION CALL

In a branch statement, there is no return. Basically, if there is a jump in the branch instruction, the control changes to the target address and doesn't get returned unless there is another branch instruction which sends it back.

On the other hand, the function call changes the control to the function code and then returns the control back to the place where the function was called always. Here is the basic process of a function call –

- When a function call is encountered, the PC value (which stores the next instruction) is stored on the stack
- Then the PC gets updated to point to the function code.
- Once the function gets executed, the PC value gets updated from the stack so that the normal flow can be resumed.

Basically, **function calls retain the PC value but branch instructions don't.**

Return From Function – This instruction copies the value from the stack to the PC to resume normal flow.

ADDRESSING MODES

The addressing modes are fields in the instructions which inform the CPU as to how the addresses provided in the instruction need to be used. These modes specify how and where the operands are accessed from – register, memory, stack etc.

ADDRESSING MODE	FUNCTION
Implied	The op – code in this case defines both the operation and the operands to be used. For example, if we need to perform operation on special purpose registers like accumulator, we don't need the operand and register address to be specified.
Immediate	The operand data is passed in the address field. So if the address field is 10 and the mode is immediate, then the operand value is 10. However, if the mode is not immediate, then the operand value is the value at memory address 10. This mode is usually used for variable initialization.
Direct/Absolute	The address part contains the effective address. The operand is pulled from the memory.
Indirect	The address part contains the address of the effective address. Basically, if we have a case where the address part has value 10. Then the instruction goes to memory address 10. Let us say the value there is 20. Then the memory address 20 has the operand value. This is useful when implementing pointers.
Register	The address field specifies the register address which contains the operand value.
Register Indirect	The address field specifies the register address which contains the effective address. This is used because length of register address is much smaller than the memory address and is therefore used to reduce instruction length.
Auto – increment/Auto – Decrement	It is a variation of the Register Indirect Mode to access a block of memory sequentially. It accesses the start address and then increments/decrements the address to access the next data in sequence. The increment/decrement value is the size of data. So if the data is 4B long, the each increment/decrement will be 4B long so that the next data is accessed after every increment/decrement.
Indexed	This is used to access an array element. In this mode, effective address is the sum of the base address of the array and the index of the element stored in the index register. If the indexed register is a special purpose register, then there is no need to mention its address in the instruction. Else, we need to mention the address.
PC – Relative/Position – Independent	Address part of the instruction is added with the PC (Program Counter) value to get the effective address. It is used in branch instructions. The address in this case is called **offset.** EA = PC value + offset. This is an intra – segment addressing mode.
Base Register	Address part of the instruction is added to the base register value to get the effective address. This is used for inter – segment branching.

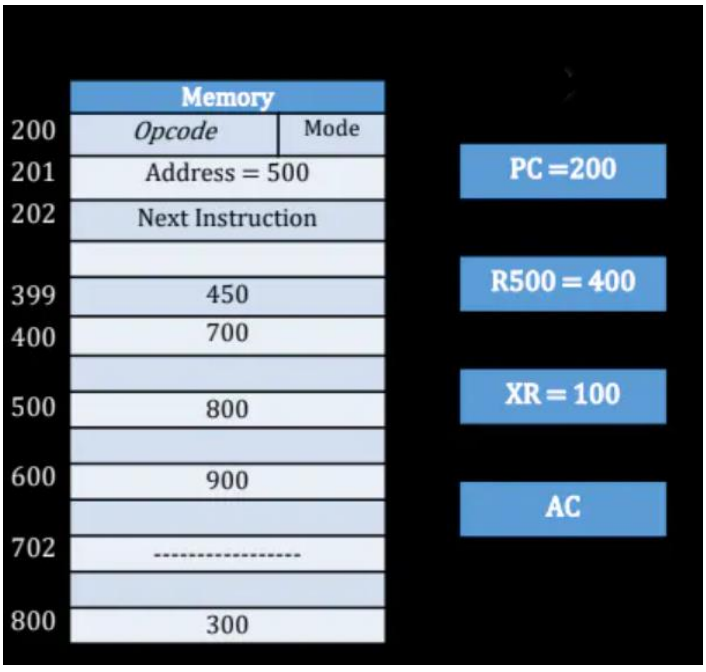
NOTE – PC – Relative and Base Register addressing mode support **relocation** and if the base instruction is changed in memory, these two modes can handle that scenario since they use offset rather than the memory address. Also, these are the only two modes that return the **target address** and the rest of the modes return the **effective address.** The reason for this is that the two modes are used for branching.

TYPES OF ADDRESSING MODES

Based on the effective address/target address calculation, there are mainly two types of addressing modes –

- **Computable –** In this case the effective address calculation requires some computation to be performed
  - Auto – Increment/Auto – Decrement
  - Indexed
  - PC – Relative
  - Base Register
- **Non – Computable –** In this case, the effective address calculation doesn't require any computation.
  - Implied
  - Immediate
  - Direct
  - Indirect
  - Register
  - Register – Indirect

QUESTION



Given this arrangement, fill the table below –

Mode	Effective Address	Operand
1. Immediate Mode		
2. Direct Mode		
3. Indirect Mode		
4. Register Mode		
5. Register Indirect Mode		
6. Autodecrement Mode		
7. Indexed Mode		
8. PC- Relative Mode		

Given,

PC => Program Counter

R500 => GRPs

XR => Index Register

Ac => Accumulator

ANSWER

MODE	EFFECTIVE ADDRESS	OPERAND
Immediate	201	500
Direct	500	800
Indirect	800	300
Register	NA	400
Register Indirect	400	700
Auto – decrement	399	450
Indexed	600	900
PC – Relative	702	------

NOTE – In assembly language, op – code is also referred to as **mnemonic.**

CPU RELATED TERMS

- **CPU Cycle time –** The time taken by CPU to perform the smallest/most basic micro-operation.
- **CPU Clock Rate –** It is the inverse of CPU cycle time
- **Cycles Per Instruction (CPI) –** Since the cycle is the time taken to execute the most basic micro – operation, the instructions require multiple cycles to execute. CPI is the number of cycles required for the execution of an instruction. CPI is different for different types of instructions.
- **Execution Time –** The time taken to execute an instruction. It is the product of CPI and CPU Cycle Time.

$$(Execution\ Time)_{AVG} = (CPI)_{AVG} * Cycle\ Time$$

- **Million Instructions Per Second (MIPS) –** This is the number of instructions that CPU can execute in 1 second divided by 1 million. If the CPU is execution 20 lakh instructions per second, then MIPS = 2.

Let us say we have n instructions being executed in total. Then, we get

$$MIPS = \frac{No.\,of\ instructions\ executed}{Total\ time * 10^6} = \frac{n}{n * Execution\ Time * 10^6} = \frac{CPU\ Clock\ Rate}{(CPI)_{AVG} * 10^6}$$

QUESTION

Consider computing the overall CPI for a machine A for which the following performance measures were recorded when executing a set of benchmark programs. Assume that the clock rate of the CPU is 200 MHz

Instruction Category	Number of Instructions	No. of cycles per Instruction
ALU	48	1
Load & Store	10	3
Branch	39	4
Other	3	5

Calculate the Average CPI and MIPS.

ANSWER

$$Total\ cycles\ required = (48 * 1) + (10 * 3) + (39 * 4) + (3 * 5) = 249$$

$$Total\ number\ of\ instructions = 48 + 10 + 39 + 3 = 100$$

Therefore,

$$CPI_{AVG} = \frac{249}{100} = \mathbf{2.49\ cycles\ per\ instruction}$$

$$Total\ time\ taken\ to\ execute\ 1\ instruction = 2.49 * \frac{1}{200 * 10^6} = 12.45ns$$

$$Instructions\ exectured\ in\ 1\ second = 80321285.14 = \mathbf{80.32MIPS}$$

ALU INPUTS AND OUTPUT

- **INPUTS**

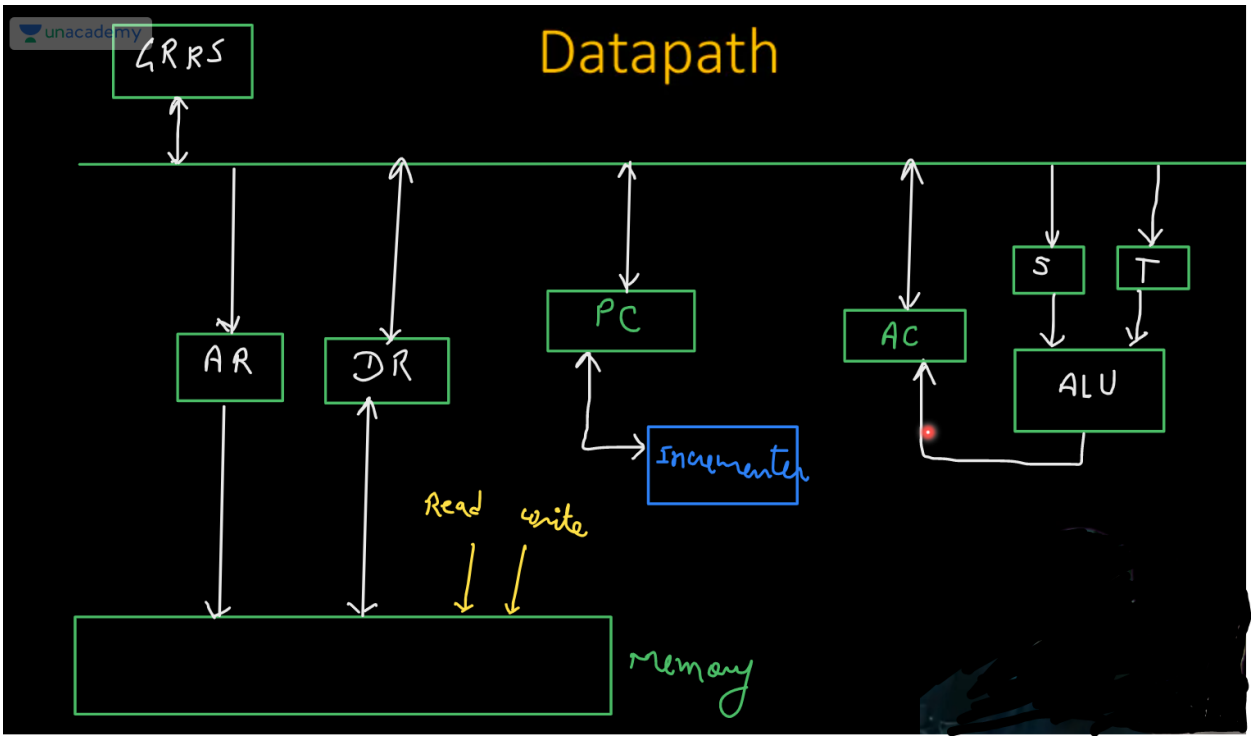
- Operand 1
- Operand 2
- Function code (this is used to specify the operation to be performed)

- **OUTPUTS**
  - Result (to accumulator)
  - Status (to Status Register)

NOTE – The instruction receives the type of operation using op – code. Once the op – code is decoded, the control unit sends the appropriate function signal to the ALU.

DATAPATH

This is the collection of functional units such as arithmetic units, logic units, multipliers etc. It performs data processing operations.



The above is a basic data bus for any CPU system. Let us understand how the Instruction Fetch is going to happen –

1. The address of the next instruction is stored in PC. Thus, we need to first send the data in PC to AR. To do so, we have two signals for every register which come from the control unit – **IN** and **OUT.** This way, the control unit can control the sender and receiver of the data.
   a. The PC counter receives the OUT signal and sends the value to the Internal Bus
   b. Then the AR receives the IN signal and reads the value in the Internal Bus

2. Next, the AR sends the data to the memory. The AR receives OUT signal from the Control Unit and the memory receives the READ signal from the Control Unit to read the data.
3. The Data from Memory now needs to go to the DR. For that, DR IN signal is enabled.
4. Finally, we need to send the data from DR to the Instruction Register (IR). So, the Control Unit enables DR OUT signal and IR IN signals.
5. Finally, the incrementor increments the PC value. This operation can happen in parallel with the previous step since the incrementor is independent of the internal bus.

NOTE – Each micro-operation takes 1 CPU cycle. For memory access, first we push the address to be accessed to AR. Then, we push the data to be pushed into DR and then finally we do $DR \rightarrow M[AR]$. Hence, a single memory access and data push took 3 micro-operations and 3 CPU cycles.

CONTROL UNIT

It generates control signals and sends those signals to the components. Based on these signals, the operations are performed by the components. To identify the different control signals, each of the control signals is given a unique name called the **control variable.** The collection of the control signal values generated by the CPU at a time is called a **control word.**

CONTROL UNIT ORGANIZATION

This determines how the Control Unit is able to generate the different control words/ signals. There are two main types of organizations –

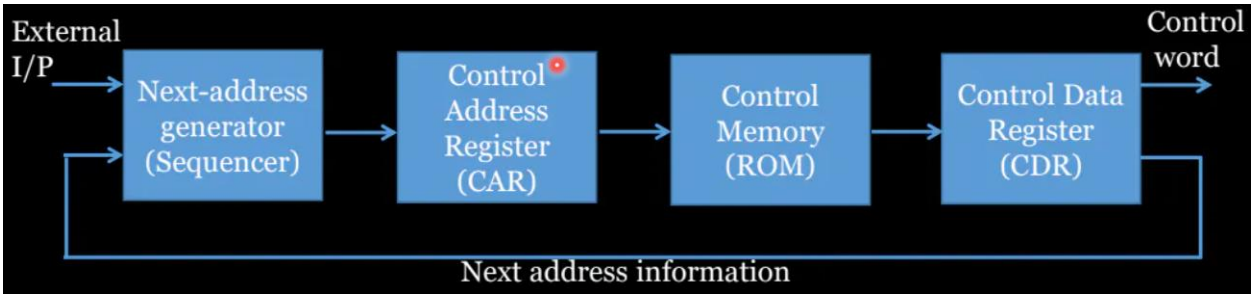
- **Hardwired CU –** Here the control logic is implemented by using physical components like Gates, FF, Decoders etc. This is a faster operating arrangement but makes things difficult to change as the wires are physically placed between components. Since changing the circuit here is very difficult, this arrangement is used in simpler systems.
- **Microprogrammed CU –** Here the control logic is implemented using micro – programs. All possible control words are stored in the memory (control memory) and fetched whenever needed. It is easier to change the logic here but the processing is slower as compared to the Hardwired CU. However, this can be easily used to implement more complex systems.

In a microprogrammed CU, there exists a device called the **next address generator** aka **sequencer.** This device will take the instruction to be executed and then point to the address of the first control word for that instruction. Each control word message in the control memory has the control word and the address information of the next control word.

NOTE – The address of the next instruction is stored in the PC. Every instruction is executed by multiple control words in sequence. Each control word in the memory also stores the information for the next control word.

CONTROL WORD SEQUENCING

- First, the External Input is provided to the Sequencer which generates the address to the first control word for that instruction.
- Once done, it stores the address in the **Control Address Register (CAR).**
- With this address, the control memory is accessed and the data in the cell is sent to the **Control Data Register (CDR).**
- The CDR has the control word and the next address information. The control word is sent to the components while the next address information is passed back to the sequencer for accessing the next control word.



NOTE – Control Memory is a ROM memory since it can't be altered.

$$Control\ Word + Next\ Address\ Info = \boldsymbol{Micro - operation}$$

TYPES OF MICROPROGRAMMED CU

The Microprogrammed CU can be of two types –

- **Horizontal –** Every control signal gets a bit in the control word. The length of the control word is longer. This has faster speed though.
- **Vertical –** The control signals are divided into **mutually exclusive** groups in such a way that only 1 signal is active at a time from each group. Each group info is stored in encoded form. For example, if we have 8 signals in one group, then we can encode them into 3 bits. However, this means that only 1 signal will be selected/active at a time. This reduces the size of control word. This is obviously slower.

For vertical CU,

$$Max\ number\ of\ signals\ active\ at\ a\ time = Number\ of\ Groups$$

HORIZONTAL	VERTICAL
Larger Control Word Size	Smaller Control Word Size
Faster	Slower
No Decoder Required	Decoder Required
Higher degree of parallelism	Lower degree of parallelism

RISC vs CISC

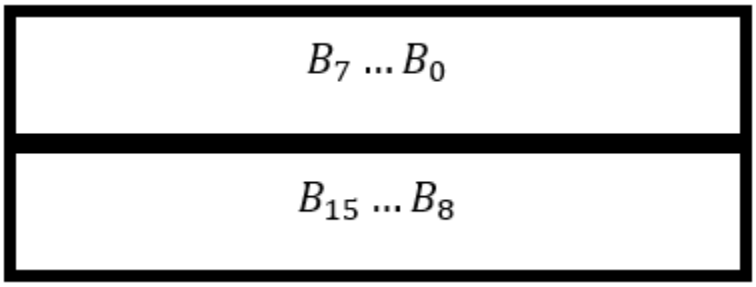
S. No.	RISC (Reduced Instruction-Set Computer)	CISC (Complex Instruction-Set Computer)
1.	Less Number of Instructions Supported	More Number of Instructions
2.	Fixed Length Instructions	Variable Length Instructions
3.	Simple Instructions	Complex Instructions
4.	Simple and less number of addressing Modes	Complex and More number of addressing Modes
5.	Easy to implement using hardwired control unit	Difficult to implement using hardwired control unit
6.	One Cycle per instruction	More than one cycle per instruction
7.	Register-to-Register arithmetic operation only	Register-to-Memory & Memory-to-Register arithmetic operations possible
8.	More Number of Registers	Less Number of Registers

BYTE ORDERING

There are cases in a memory where the data is stored in multiple locations across the memory. For example, let us say we have a byte addressable memory and an instruction that is 2B (16b) long. In that case, we need two memory locations to store that data. Based on how this is stored, we can have two architectural systems –

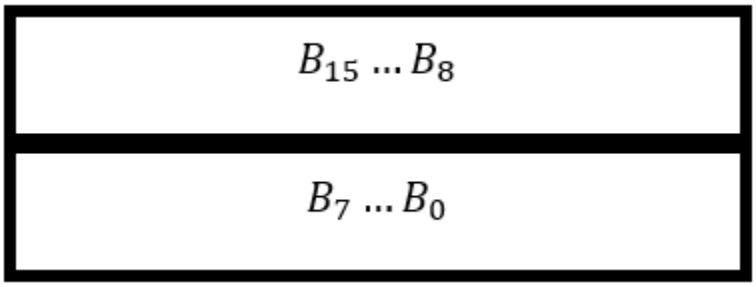
Little Endian

The LSBs are stored first and then the MSBs.



Big Endian

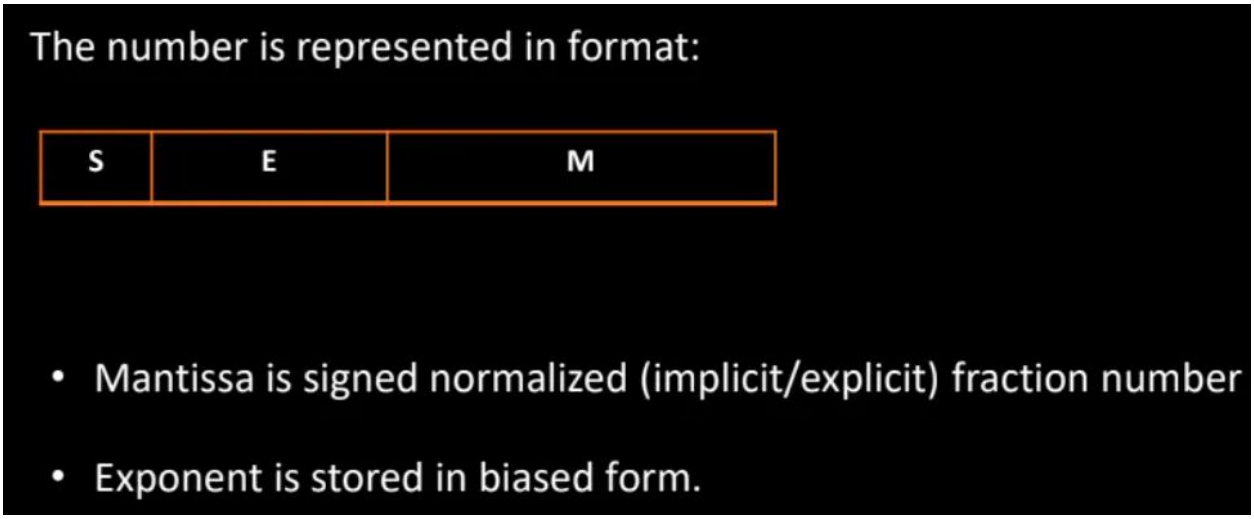
The MSBs are stored first and then the LSBs



FIXED POINT NUMBERS

In this case, there is a fixed number of bits that are to be used to represent the mantissa (before decimal point) and exponent (after decimal point). So, if we have an 8 – bit storage space, then fixed point representation includes fixing the first 5 bits (example) for mantissa and the last 3 bits for exponent. However, this reduces the range of numbers we can represent. In the example above, the max value we can represent using 8 – bit space is only 31.875. Therefore, we go for floating point representation.

FLOATING POINT NUMBERS



Exponent and Bias

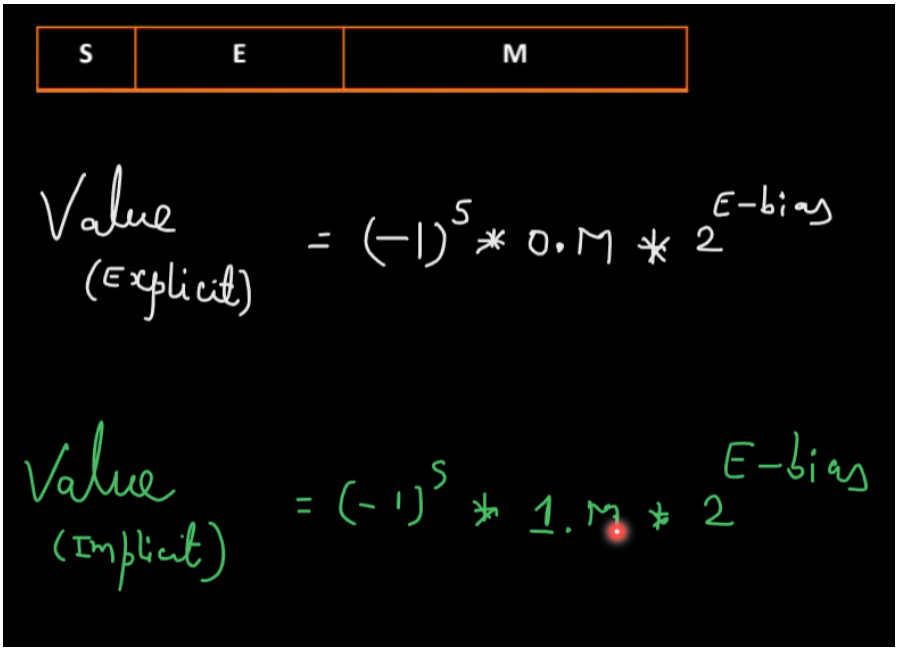
What is the biased form for the exponent? Let us assume the size of the exponent storage is 5 bits. That means, the exponent can take a value from -16 to +15 (2's complement). However, we don't want to store the exponent as a negative number but rather want it to be a positive number. For that, we add a **bias** to the exponent values. Now this bias in our example will be +16. Therefore, if we add a bias to the exponent

values, the range of exponent changes to 0 – 31 instead of the previous one which was -16 - +15. In general, if we have n bits for exponent, then the bias is 2^{n-1}.

Mantissa and Normalization

One way to store the Mantissa is to make the decimal point appear before the first 1. So if we have a number as 101.11 or 0.00101, then with this method the two values will be saved as 0.10111 and 0.101. Therefore, the mantissa for both cases will be 10111 and 101 respectively. This is called **Explicit Normalization.** If not mentioned in the question, take the default normalization as Explicit.

On the other hand, we have something called an **Implicit Normalization** in which the decimal point must be placed after the first 1. So, if we take the above example, the number will be 1.0111 and 1.01 and the mantissa will be 0111 and 01 respectively.



QUESTION

Consider a 16-bit register used to store floating point numbers. The mantissa is explicitly normalized signed fraction number. Exponent is represented in excess-32 form.

1. What is the minimum possible value of mantissa?
2. What is the maximum possible value of mantissa?
3. What is the minimum possible value of exponent?
4. What is the maximum possible value of exponent?
5. What is the positive minimum possible value represented?
6. What is the maximum possible value represented?

ANSWER

In this case, we have

$$Size\ of\ Sign\ bit = 1$$

$$Size\ of\ exponent = 6\ (since\ we\ add\ a\ bias\ of\ 32, all\ exponent\ values\ will\ be\ greater\ than\ 32)$$

$$Size\ of\ mantissa = 16 - (6 + 1) = 9$$

Now, we are using explicitly normalized mantissa. So the first bit after the decimal must be a 1. Therefore, the minimum possible value is **100000000 (256).** Similarly, the maximum value will be **111111111 (1023).**

The bias in the exponent is added to make sure all exponent values are positive numbers. Therefore, the minimum value of exponent will be **000000 (0)** and similarly, the maximum value will be **111111 (127).**

Minimum Positive Value

$$S = 0$$

$$E = 000000 = -32$$

$$M = 1000000000 = 0.1$$

Therefore, minimum value = $0.1 * 2^{-32} = \mathbf{2^{-33}}$

Maximum Positive Value

$$S = 0$$

$$E = 111111 = +31$$

$$M = 111111111 = 0.111111111$$

Therefore, maximum value = $0.111111111 * 2^{31} = 111111111.0 * 2^{-9} * 2^{32} = \mathbf{(2^9 - 1) * 2^{22}}$.

Now, in the above question if we change it to implicit normalization, we get –

a) 000000000
b) 111111111
c) 000000
d) 111111
e) 2^{-32}
f) $(2^{10} - 1) * 2^{22}$

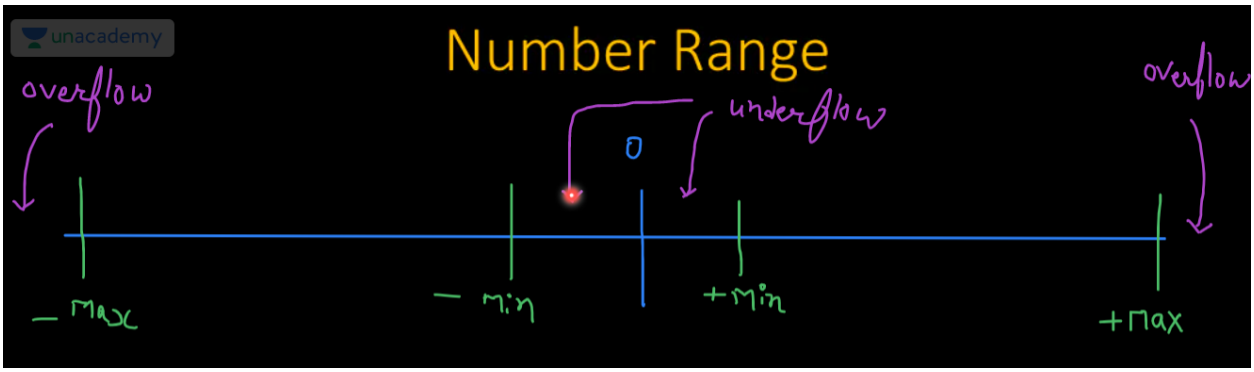
OVERFLOW AND UNDERFLOW

In the number line, we have five main values –

1. The maximum positive value
2. The minimum positive value
3. ZERO
4. The maximum negative value
5. The minimum negative value

Overflow – If the value is larger than the maximum positive value or lower than the minimum negative value.

Underflow – If the value is between the minimum positive value and the maximum negative value.



NOTE – In explicit normalization, there should be a 1 after the decimal point. On the other hand in the implicit normalization, there should be a 1 before the decimal point. Therefore, we **can't store zero using this model.**

NOTE – If we increase the number of bits in exponent, then we get a larger range. However, if we increase the number of bits in mantissa, then we get better precision/accuracy.

IEEE-754 FLOATING POINT REPRESENTATION

Since we can't store zero or small values (underflow) using the above model, we moved on to IEEE-754 model. This representation is of two types –

- Single Precision
  - Sign bit = 1
  - Exponent = 8 ; Bias = +127
  - Mantissa = 23
  - Total = 32
- Double Precision
  - Sign bit = 1
  - Exponent = 11 ; Bias = +1023
  - Mantissa = 52
  - Total = 64

In both the above representations, we can observe that the bias is 1 lesser than the expected value. This is because Exponent all 0's and all 1's are reserved for special numbers and any normal number will never have these exponents.

For the IEEE-754 representation (single precision), there are a few special cases –

SIGN BIT	EXPONENT	MANTISSA	VALUE
0	000…000	000…000	+0 (Min positive value)
1	000…000	000…000	-0 (Max negative value)
0	01111111 (127)	000…000	+1
1	01111111 (127)	000…000	-1
0	111…111	000…000	$+\infty$ (Max value possible)
1	111…111	000…000	$-\infty$ (Min value possible)
0/1	111…111	Mantissa ≠ 0	INVALID NUMBER
0/1	000…000	Mantissa ≠ 0	DE – NORMALIZED NUMBER
0/1	E ≠ 000…000 (or) 111…111	Mantissa = XXX…XXX	IMPLICIT NORMALIZATION
0	11111110 (254)	111…111	$(2^{24}-1)*2^{104}$ (Max normalized positive value)
0	00000001 (1)	000…000	2^{-126} (Min normalized positive value)
0	00000000 (0)	000…001	2^{-149} (Min de-normalized positive value)
0	00000000 (0)	111…111	$(2^{23}-1)*2^{-149}$ (Max de-normalized value)

NOTE – IEEE – 754 only has IMPLICIT NORMALIZATION. There is no explicit normalization for this format.

DENORMALIZED NUMBER

Using implicit normalization in IEEE-754, the lowest possible value of exponent is 000…001 since all 0's is reserved for special numbers. Therefore, the lowest possible exponent we can store is $2^{1-127} = 2^{-126}$. If we want to store any number smaller than this (the underflow condition in previous convention), then we make use of **denormalized number.**

From the table above, we can store the denormalized number by storing the exponent as all 0's. This is true because the denormalized values are smaller than the minimum positive number which have 000…001 as the exponent.

QUESTION

The value of a float type variable is represented using the single- precision 32-bit floating point format IEEE-754 standard that uses 1bit for sign, 8 bits for biased exponent and 23 bits for mantissa. A float type variable X is assigned the decimal value of −27.625. The representation of X in hexadecimal notation is?

ANSWER

$$Sign\ bit = 0$$

$$Number\ (implicit) = 1.1011101 * 2^4$$

$$Exponent = 4 + 127 = 129$$

Therefore, the number becomes $(\boldsymbol{C1DD0000})_{\boldsymbol{16}}$

PERIPHERAL DEVICE

These are externally connected to the CPU (apart from main memory). These are generally used to either give the CPU input or take the output from the CPU and therefore are normally called **I/O Devices.** The I/O devices can't communicate directly with CPU since they use different signals and configurations. Also, CPU are electronic devices while I/O devices are electro-mechanical devices. To connect CPU and I/O devices, there is a need for **I/O Interface.**

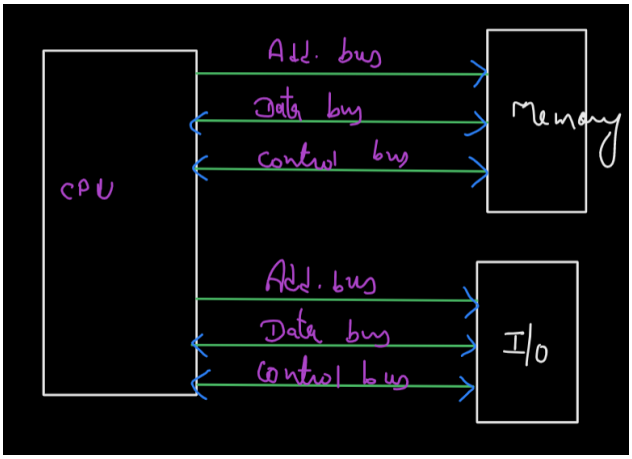
1. Peripherals are electromechanical or electromagnetic devices; and their manner of operation is different from the operation of the CPU and memory. Which are electronic devices. So conversion of signal required.
2. The data transfer rate of peripherals is usually slow. So synchronization is required.
3. Data codes and format in peripherals differ from the word format in the CPU and memory. So conversion of formats is required.
4. The operating modes of peripherals are different from each other and each must be controlled so a peripheral does not disturb the operation of other peripherals.

NOTE – From this point on, it is assumed that an I/O interface is present when CPU is being connected to I/O devices even when it is not explicitly mentioned.

I/O DEVICES AND MEMORY ORGANIZATION

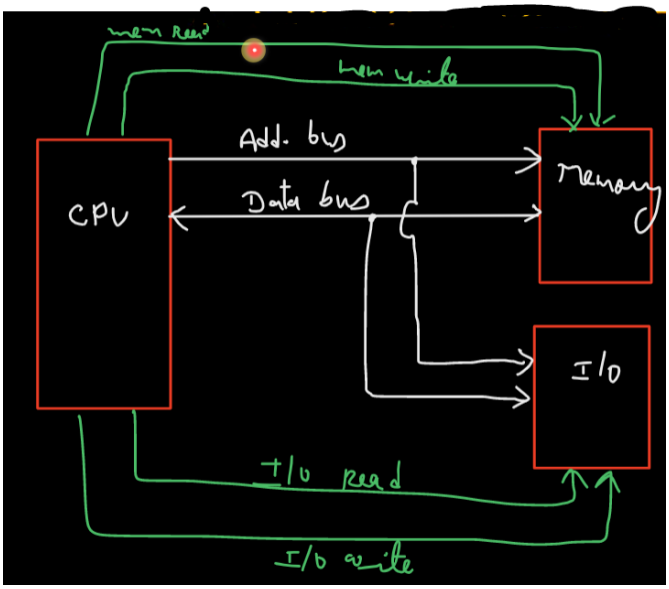
Since the CPU needs to connect to the I/O devices and the Memory Bus, it needs some organization. For this, there are three choices –

Separate Buses for Memory and I/O



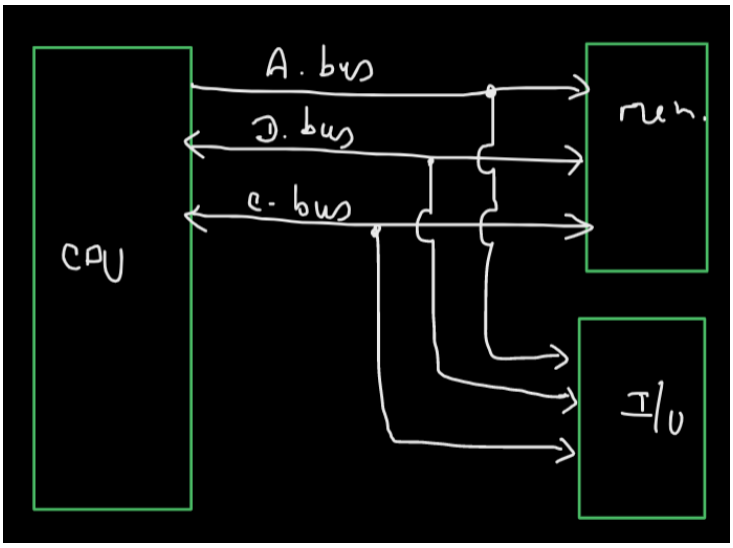
This is simpler to understand but it is costly and also there is no direct communication between memory and I/O devices.

Common Data and Address Bus (I/O – mapped – I/O aka Port – mapped – I/O)



This organization is cheaper and also enables CPU, Mem and I/O to communicate with each other. Here, the selection between Mem and I/O will be done based on control signals. Each cell in mem has a unique address while each I/O device has a unique address.

Common Data, Address and Control Bus (Memory – mapped – I/O)



Let us assume the addresses are of n bits length. Then, we get –

$$Size\ of\ memory = 2^n$$

$$Number\ of\ IO\ devices = k$$

$$Number\ of\ memory\ addresses = 2^n - k$$

Basically, each I/O device gets a unique address and the memory uses the remaining addresses. Some of the memory cells are not used because the address of those cells is used by I/O devices. That is why this is called **memory mapped I/O.** This also has some memory wastage.

Since the I/O devices have different address, the control bus and data bus send the same data to memory and I/O devices. The address determines if the I/O or Memory uses the control and data signals being sent.

MEMORY MAPPED IO	**IO MAPPED IO**
I/O devices don't have their own address space.	I/O devices have their own address space.
Memory space wasted to accommodate I/O	No memory wastage
Memory access and instruction modes are also used for I/O	I/O access and Memory access or instructions are different.
More number of instructions and modes for I/O access required.	Less number of instructions and modes for I/O access required.
CPU uses address to distinguish between memory and I/O	CPU uses control signals to distinguish between memory and I/O
Can accommodate more I/O devices	Lesser I/O devices can be accommodated

ANSYNCHRONOUS DATA TRANSFER

Synchronous data transfer is not ideal for systems where the devices require different clock cycles for operation. In this case, the devices are allowed to operate on individual clock cycles and will only require

synchronous data transfer when devices interact with each other. This system is called the asynchronous data transfer.

START AND STOP BITS

When transferring data from source to destination over a bus, the destination needs to know when a particular character ends and a new character is being sent. To ensure there is a demarcation between characters being sent, each character has a START and a STOP bit (or signal) to indicate the beginning and end of the character being sent. This is used for **synchronization of characters.**

NOTE – We use a metric called line efficiency which is defined as –

$$line\ efficiency = \frac{Number\ of\ bits\ received}{Number\ of\ bits\ sent}$$

SYNCHRONIZATION BITS

Another way to perform the synchronization is to have a bunch of bits called the synchronous bits being sent in the beginning and the end of the message. So here, the data stream looks like –

$$Data\ Stream = Sync\ bits + Character\ stream + Sync\ bits$$

Question

How many 8-bit characters can be transmitted per second over 9600 baud serial communication link using a parity synchronous mode of transmission with 1 start bit, 8 data bits, 2 stop bits and 1 parity bit?

Answer

$$Number\ of\ bits\ per\ second = 9600$$

$$Number\ of\ bits\ in\ 1\ character\ message = 1 + 8 + 2 + 1 = 12b$$

$$Number\ of\ characters\ per\ second = \frac{9600}{12} = 800bps$$

MODES OF TRANSFER

There are different modes of data transfer between CPU and I/O –

Programmed I/O aka Program Controlled I/O

In this, the CPU individually contacts the I/O devices if they need to transfer data. This is called Programmed I/O since the CPU has an in – built program to contact the I/O devices one – by – one. This

is super slow since if we have 20 I/O devices and the CPU starts asking each of them individually, then it would waste a lot of time.

- There is no any provision through which IO can inform to CPU about data transfer
- IO sets its own status and waits
- CPU runs program periodically and checks the status of each device one-by-one
- If any device has its status set then CPU performs data transfer for it.

$$Time\ taken = Time\ taken\ to\ read\ status\ reg\ from\ IO + TIme\ taken\ to\ check\ the\ flag\ bit\ in\ CPU + Data\ transfer$$

Interrupt Initiated (or Driven) I/O

The I/O devices here initiate an interrupt signal to ask the CPU to pause the current execution and then serve the needs of the I/O devices. This is faster since the transfer is done as and when the I/O device needs the CPU.

- IO device has a provision (Interrupt Signal) to inform to CPU about communication.
- When CPU receives interrupt:
  - It completes execution of current instruction
  - Saves the status (PC, PSW etc.) of current process onto the stack
  - Branches to service the interrupt
  - Resumes the previous process by taking out the values from stack

Direct Memory Access (DMA)

Both of the above cases do not accommodate the case where the I/O devices need to communicate with the Memory directly. For the case where Memory and I/O devices need to have direct communication, the DMA was introduced.

Question

Consider a device which operates with 50KBPS operating speed. The device is operating on program control mode of IO and it has to transfer data of 10B from it. The data is transferred byte wise. Size of status register is 2Bytes. Total time needed to perform the data transfer is _____ microseconds?
(Note: The status is checked only once, in the beginning)

Answer

$$Total\ time = (Time\ to\ access\ 2B\ status\ register) + (Time\ to\ transfer\ 10B\ data)$$

$$Time\ to\ transfer\ or\ access\ 1B = \frac{1}{50K} = 20\mu s$$

Therefore,

$$Total\ time = (2 * 20\mu s) + (10 * 20\mu s) = \mathbf{240}\boldsymbol{\mu s}$$

INTERRUPT SERVICE ROUTINE (ISR)

The function or routine whose execution services the interrupt.

VECTORED AND NON – VECTORED INTERRUPTS

When the interrupt gives the address of the ISR in the memory, then the interrupt is called a vectored interrupt. Else, in the non – vectored case, the CPU runs the default service for any interrupt and that in turn helps the CPU get the correct ISR.

MASKABLE AND NON – MASKABLE INTERRUPTS

Maskable interrupts are not high priority and these can be ignored by CPU. On the other hand, non – maskable interrupts always need to be serviced by the CPU.

INTERNAL AND EXTERNAL INTERRUPTS

When a device generates an interrupt, then it is called an External Interrupt. On the other hand, if the CPU generates an interrupt due to some unexpected error during execution, it is called an Internal Interrupt.

Question

The following are some events that occur after a device controller issues an interrupt while process L is under execution.

- P. The processor pushes the process status of L onto the control stack
- Q. The processor finishes the execution of the current instruction
- R. The processor executes the interrupt service routine
- S. The processor pops the process status of L from the control stack
- T. The processor loads the new PC value based on the interrupt

Which of the following is the correct order in which the events above occur?

A. QPTRS
B. PTRSQ
C. TRPQS
D. QTPRS

Answer

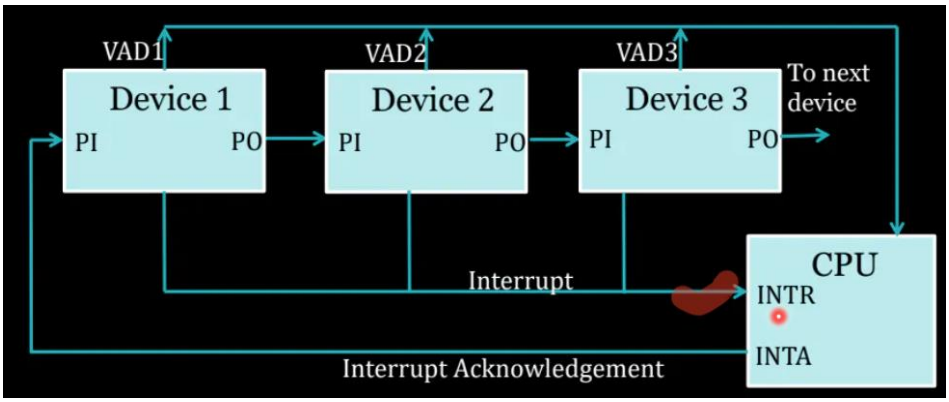
Option A

SIMULTANEOUS INTERRUPTS

When multiple devices generate interrupts simultaneously, then the interrupts are ranked based on device priorities. The highest priority device's interrupts are serviced first. This is called **Priority Based Interrupt Handling.** They can be multiple ways to implement this –

- Software Solution – CPU runs a program when it receives an interrupt and the program determines the priorities and order in which the interrupts need to be handled.
- Hardware Solution
    - Serial (Daisy Chaining)
    - Parallel

DAISY CHAINING



Here, the devices are connected in serial. All the devices have the same interrupt line such that if any of the devices send an interrupt, then the line is activated. Once the interrupt is received, the CPU returns an acknowledgement which is sent ins a serial manner to the different devices. In short, Device 1 receives the acknowledgement first, then device 2 and so on. The first device that has sent the interrupt and received an acknowledgement will send its address to the CPU so that the ISR begins.

TIME REQURIED TO SERVE I/O INTERRUPT

$$Total\ time\ taken = Interrupt\ Overhead\ time + Interrupt\ service\ time$$

The Interrupt Overhead time is the time needed to determine the Interrupt priority, generate acknowledgement and other activities before the interrupt begins servicing.

Question

A device with data transfer rate 20 KB/sec is connected to a CPU. Data is transferred byte-wise. Let the interrupt overhead be 10 microsecond.

1. Total time required in programmed IO for 10 bytes data transfer?
2. Total time required in interrupt IO for 10 bytes data transfer?
3. What is the minimum performance gain of operating the device under interrupt mode over operating it under program controlled mode?

Answer

For Programmed IO,

$$Total\ time = Status\ check\ time\ (in\ status\ register) + Data\ Transfer\ Time$$

Since the data is sent out byte – wise, the status register size would be 1B. Therefore,

$$Status\ check\ time = \frac{1}{20K} = 50\mu s$$

$$Data\ transfer\ time = 10 * 50 = 500\mu s$$

Therefore,

$$Total\ time = 550\mu s$$

For Interrupt IO

$$Total\ time = Overhead\ time + Data\ Transfer\ Time$$

Thus,

$$Toal\ time = 10 + 500 = 510\mu s$$

Performance Gain,

$$Perf\ gain = \frac{550 - 510}{510} * 100\% = \mathbf{7.8}\%$$

Direct Memory Access (DMA)

This is a data transfer mode to transfer data from memory to I/O without CPU intervention. It needs a specific hardware called DMA Controller. It is a step – by – step process –

1. First the I/O device will send a DMA request to the DMAC
2. DMAC then sends a signal to CPU to give it the permission to manage the system bus for data transfer. This signal is called the Bus Request (HOLD).
3. The CPU then sends the starting memory address and the size of data that needs to be transferred between the memory and I/O. This is called DMA Initialization.
4. Next, the CPU returns the Bus Grant (HOLD ack) to the DMAC back.
5. Next, the DMAC will send a DMA Ack back to the I/O device indicating that the bus request has been granted.

6. Finally, the DMAC allows the data transfer between I/O and memory.

NOTE – When the system bus is in use by DMAC, the CPU can't use the system bus to transfer data at the same time. Instead, the CPU can use other processes completed which don't require the system buses. However, the DMA transfer should not take a long time since CPU shouldn't be blocked for a long time.

MODES OF DMA TRANSFER

These modes dictate the time duration for which the CPU will remain blocked and DMAC uses the system bus. There are three main modes –

- **Burst Mode –** This mode allows the DMAC to transfer a burst of data (a block) before the CPU takes the control back.
- **Cycle Stealing –** This is used in cases where the internal processing of the I/O devices is way slower when compared to the CPU. Thus, while the I/O device performs the internal processing to get the data ready, the CPU will be using the system bus. Once the data is made ready by the I/O device, the DMAC steals the bus from the CPU for one memory cycle to finish data transfer.
- **Interleaving DMA –** In this case, the CPU will give the control of the system bus to the DMAC only when the CPU doesn't need it. Even if the I/O data is ready to be transferred but CPU needs the system bus, then the DMAC doesn't get the system bus for DMA. In this case, the amount of time CPU is blocked is approximately zero.

NOTE – Let us assume that I/O device takes 60s to prep the data and the memory cycle of 5s is enough to transfer the data. So, the CPU will be blocked for 5s. However, while the data is being transferred, the I/O starts prepping the next data block as well. In that case, by the time data transfer finishes, the I/O will be left with just 55s of prep time. This is called **overlapping** and is observed in Cycle Stealing mode. It is NOT observed in Burst Mode.

$$Time\ taken\ by\ IO\ to\ prep\ data = t_x$$

$$Time\ taken\ to\ transfer\ the\ data = t_y$$

For Burst Mode

$$\%\ of\ time\ CPU\ is\ blocked = \frac{t_y}{t_x + t_y} * 100$$

For Cycle Stealing Mode

$$\%\ of\ time\ CPU\ is\ blocked = \frac{t_y}{t_x} * 100$$

NOTE – As we can see that DMAC and CPU can't use the bus at the same time, this begs the question as to why DMA is needed. DMA is needed because the CPU doesn't need to waste time and instructions to change its state for I/O operations. It can just let DMA handle it. Also, since CPU is not changing its state, the DMA can be done even if the CPU is in the middle of instruction execution (unlike ISR).

Question

Consider a device operating on 1MBPS speed and transferring the data to memory using cycle stealing mode of DMA. If it takes 2 microseconds to transfer 16 bytes data to memory when it is ready/prepared. Then percentage of time CPU is blocked due to DMA is?

Answer

In this case,

$$t_y = 2\mu s$$

$$t_x = \frac{16B}{1MBps} = 16\mu s$$

Since we are using cycle stealing mode, we have

$$\%\ of\ CPU\ used = \frac{t_y}{t_x} * 100 = \frac{2}{16} * 100 = \mathbf{12.5}\%$$

Question

A hard disk with a transfer rate of 10 Mbytes/ second is constantly transferring data to memory using DMA. The processor runs at 600 MHz, and takes 300 and 900 clock cycles to initiate and complete DMA transfer respectively. If the size of the transfer is 20 Kbytes, what is the percentage of processor time consumed for the transfer operation ?

(A) 5.0%
(B) 1.0%
(C) 0.5%
(D) 0.1%

Answer

The sequence of operation for DMA is –

$$Total\ operation = Initialization\ (300\ cycles) + Data\ transfer + Completion\ (900\ cycles)$$

$$Initialization\ time = \frac{300}{600M} = 0.5\mu s$$

$$Completion\ time = \frac{900}{600M} = 1.5\mu s$$

$$Data\ transfer\ time = \frac{20K}{10M} = 2ms = 2000\mu s$$

Therefore, we have

$$Time\ CPU\ consumed = Initialization + Completion = 2\mu s$$

$$Total\ time = 0.5\mu + 1.5\mu + 2000\mu s = 2002\mu s$$

Finally,

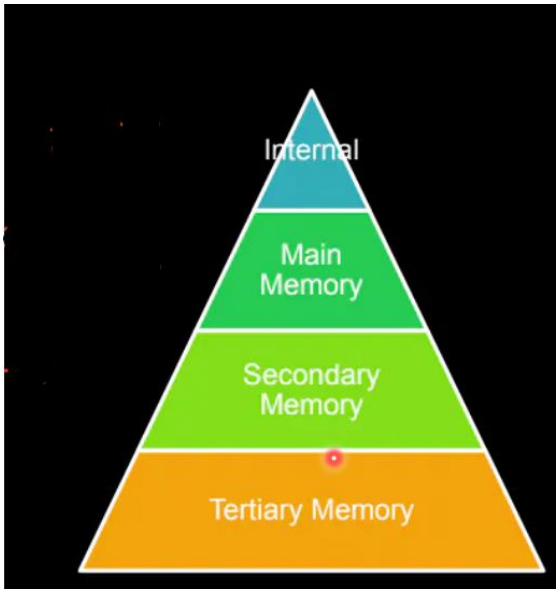
$$\%\ of\ CPU\ time\ consumed = \frac{2}{2002} * 100 = \mathbf{0,1}\%$$

MEMORY ORGANIZATION

There exists a memory hierarchy in a system for mainly two reasons –

- To maximize access speed
- To lower per bit storage cost

Access time and per bit storage cost are inversely related.



- **Internal –** Registers and Cache

- **Main Memory –** RAM and ROM. Used to store current running programs and operand values.
- **Secondary Memory –** HDD/SSD
- **Tertiary Memory –** Tapes

In the above diagram –

FIELD	UP – TO – DOWN DIRECTION
Size	Increases
Access Speed	Decreases
Access Time	Increases
Per bit cost	Decreases
Freq of access by CPU	Decreases

MEMORY PRESENTATION

The memory is divided into equal sized cells. Total size is the product of number of cells and the size of 1 cell. The memory is represented as follows –

$$Memory\ Representation = Number\ of\ cells\ \boldsymbol{X}\ Cell\ Capacity$$

Every cell has a unique address to prevent confusion for the CPU. Therefore,

$$Number\ of\ cells = Number\ of\ addresses$$

$$Size\ of\ address = \log_2(Number\ of\ addresses)$$

- **Byte addressable –** Size of 1 cell is 1B
- **Word addressable –** Size of 1 cell is 1 word. Size of word (in bytes) varies from system to system.

For byte addressable,

$$128k\ X\ 8 - bits \equiv 128k\ X\ 1B \equiv 128kB$$

Question

A processor can support a maximum memory of 4 GB, where the memory is word addressable (a word consists of two bytes). The size of the address bus of the processor is at least _____ bits?

Answer

$$Memory\ size = 4GB = 2^{32}B$$

$$Size\ of\ 1\ cell = 2B$$

$$Number\ of\ cells = \frac{2^{32}}{2} = 2^{31}$$

$$Size\ of\ address\ bus = \boldsymbol{31\ bits}$$

Memory Cycle – The time taken to perform read/write on one cell.

MEMORY ADDRESS DECODER

Let us take a case where the number of cells in the memory is n. Then the address bus will be of $\log_2 n = k(say)$ long. Then, the Memory Address Decoder, which is a $k\ X\ n$ decoder will take in the address bits and then select the cell address to send to the CPU. For example, if the memory size if 128KB and the memory is byte addressable, then the decoder needed would be **17 X 128K.**

Question

If there are m input lines n output lines for a decoder that is used to uniquely address a byte addressable 1 KB RAM, then the minimum value of m+n is _____?

Answer

$$Value\ of\ m = 10$$

$$Value\ of\ n = 1024$$

Therefore,

$$m + n = \mathbf{1034}$$

READ – ONLY MEMORY (ROM)

This is a non – volatile memory and is read only. This means that even if the system is shut down, the data in ROM doesn't get wiped. Once the system starts, the following steps occur –

- Before system is turned ON, the RAM is empty, ROM has a set of programs and the OS is in the secondary memory (C drive)
- Once system turns ON, the CPU checks the programs in the ROM and executes them. Out of these programs, the first ones to execute check the various contents and the memory connected to the system. This is called Power On Self Test (POST).
- Next, the ROM executes the program that copies the OS to the RAM. This is called Booting and the programs are called Bootstrap programs aka Loader.
- Once the system has booted, the CPU can now execute the user programs from the RAM and the system is ready to be used.

READ ONLY MEMORY (RAM)

This is a volatile memory as this memory is wiped clean when the system is switched off. The OS is copied to RAM because then only we will be able to make modifications and updates. In the task manager, if we look at the processes running, then we can get an idea as to what all is copied and running from the RAM. That is why we need larger sized RAMs.

TYPES OF RAM

STATIC RAM (SRAM)	DYNAMIC RAM (DRAM)
Made up of FF	Made up of capacitors
No refresh or recharge required.	Since this uses capacitors, this requires period refresh.
Faster Read/Write cycles	It is slow for Read/Write due to periodic refresh
Expensive	Cheap
As these are fast but expensive, they are used for cache implementation	These are used for main memory as they are slower but cheaper
Less Static Power consumption	More static power consumption
More operational power consumption	Less operational power consumption

For DRAM, there is a refresh required. Therefore, the operation starts with a Read/Write cycle and then a refresh occurs. This keeps on repeating. The addition of 1 Read/Write cycle and the refresh time is called the **Refresh Period.**

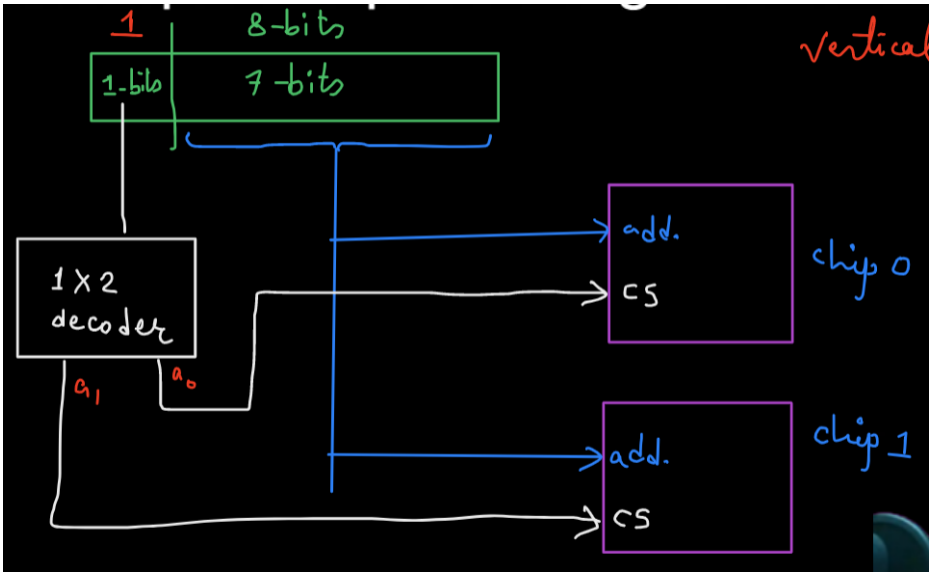
NOTE – Let us take two numbers of size n bits each. Then, we have

- **Max Sum size –** n+1 bits
- **Max Product size -** $2n$ bits
- **Addition table size -** $2^{2n}\ X\ (n+1)$ bits
- **Multiplication table size -** $2^{2n}\ X\ 2n$ bits

CHIP SELECT

Normally, we have multiple chips (RAM or ROM) in a system and therefore, we need a signal for the CPU to select a chip. Let us assume we need to create a memory of size 256 X 8 bits. However, we have chips of size 128 X 8 bits. In this case, we would need **2 chips** to get the memory requirement.

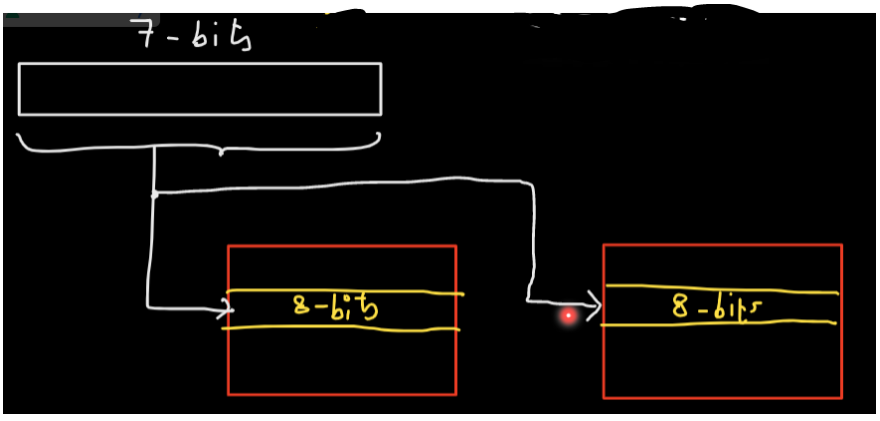
Now, how to arrange these chips? The CPU will send out 8 bit addresses since the total memory is 256 x 8 bits. However, each chip can only take 7 bit addresses. Therefore, the MSB is reserved for the **chip select** bit. If the MSB is 0, the next 7 bits are an address in CHIP 1 and if the MSB is 1, the next 7 bits are an address in CHIP 2. This can be implemented as shown below –



The MSB is passed to a decoder and the output of the decoder goes to the Chip select for the 2 chips. This is called the **Vertical Arrangement.**

Now, let us change the question a bit. Let us assume we want a 512 X 8 bit memory using 128 X 8 bit chips. In this case, the first 2 bits of the 9 – bit address provided by the CPU will be passed through a 2 X 4 decoder to get the 4 chip select signals for the 4 128 X 8 bit chips.

Let us now assume we need a 128 X 16 bit memory using 128 X 8 bit chips. In this case, the scenario will be different. Here, we are not changing the size of address bits and the CPU will provide 7 bit addresses to both the chips. However, the chips will be returning the data simultaneously. Each chip will provide 8 bit data which will be combined to form the 16 bit data CPU needed.



This is called **Horizontal Arrangement.** Therefore, we can write the following –

*If we want more number of addresses, we use **vertical arrangement.** If we want more bits per address, then we use **horizontal arrangement.** If we want more addresses and more bits per address, we use a combination of both which is also called a **hybrid arrangement**.*

DRAM REFRESH

In a DRAM chip, the cells are arranged in a grid fashion to save space. That means that there are rows and columns of cells. When the DRAM performs a refresh operation, **one single row of cells is refreshed.** Therefore,

$$Refresh\ time\ of\ chip = Refresh\ time\ of\ a\ single\ row * Number\ of\ rows$$

Question

Consider a DRAM which can be refreshed in 10ns. The refresh period is 0.05 microseconds.

1. % of time taken in refresh?
2. % of time remaining for read write is?

Answer

$$Refresh\ Period = 50ns$$

$$Refresh\ time = 10ns$$

Thus,

$$Read\ or\ Write\ time = 50 - 10 = 40ns$$

Now,

$$\%\ of\ time\ taken\ for\ refresh = \frac{10}{50} = \mathbf{20}\%$$

$$\%\ of\ time\ taken\ for\ read\ or\ write = \mathbf{80}\%$$

NOTE – If there are multiple DRAM chips, the refresh for each happens in parallel. So if the refresh for 1 chip takes 10ns and there are 5K chips in the memory, then the refresh time for the whole memory (all the chips) is still 10ns.

ASSOCIATIVE MEMORY

This is also known as Content Addressable Memory. In this case, the cells don't have addresses. Instead, the information in each cell is divided into two parts –

- The first part is the content that CPU is aware about.
- The second part is the content that CPU needs to get

In this case, the CPU starts looking for the content it knows about. This content is searched for in the memory **parallelly** and when there is a match, the rest of the information in the cell is sent back to the CPU. Since there is parallel matching, the memory is extremely fast. However, the hardware requirement is quite complex and this is also very expensive.

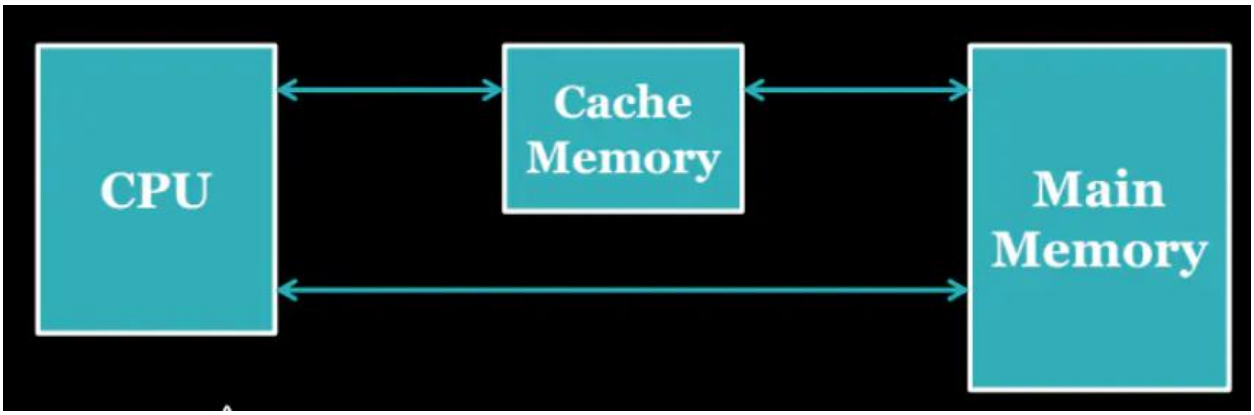
LOCALITY OF REFERENCE

If CPU has requested one address for memory access, then that particular address or near by address will be accessed soon. For example, let us assume address 3000 was accessed by the CPU. Then it is quite likely that either 3000 will be accessed or addresses near 3000 will be accessed next. This property is called **Locality of Reference.**

If the same addressed is accessed again by the CPU, then it is called **temporal LOR** and if the CPU accesses a nearby address, then it is called **Spatial LOR.**

CACHE MEMORY

Now we know that CPU only needs a specific locality instead of the entire memory. For this, the locality is stored in a fast speed memory called **cache memory.** Cache memory consists of the most frequently accessed localities by the CPU. This is done to improve the speed of the CPU access.



- **Cache Hit –** If the required content by the CPU is present in the cache.
- **Cache Miss –** If the required content by the CPU is NOT present in the cache
- **Hit Ratio –** It is the ratio of the number of cache hit to the total number of memory accesses done.
- **Miss Ratio –** 1 – Hit ratio.

When there is a Cache Miss, the CPU accesses the Main Memory to get the required data. Then this required data and its locality is copied to cache memory in anticipation that the CPU will have to access the locality again.

AVG MEMORY ACCESS TIME

$$T_{avg} = (H * T_H) + [(1 - H) * T_M] \qquad Eq\ (1)$$

Where,

$$T_{avg} \rightarrow Avg\ access\ time$$

$$H \rightarrow Hit\ Ratio$$

$$T_H \rightarrow Time\ required\ for\ hit$$

$$T_M \rightarrow Time\ required\ for\ miss$$

TYPES OF CACHE ACCESS

Simultaneous Access

In this case, the cache and memory access requests are sent simultaneously by the CPU. Thus,

$$T_H = Cache\ memory\ access\ time\ (t_{CM})$$

$$T_M = Main\ memory\ access\ time\ (t_{MM})$$

$$T_{avg} = (H * t_{cm}) + [(1 - H) * t_{MM}] \qquad Eq\ (2)$$

Hierarchical Access

The cache is accessed first. In case the cache is a miss, then the main memory is accessed.

$$T_H = t_{CM}$$

$$T_M = t_{MM} + t_{CM}$$

$$T_{avg} = (H * t_{CM}) + [(1 - H) * (t_{MM} + t_{CM})] = t_{CM} + [(1 - H) * t_{MM}] \qquad Eq\ (3)$$

NOTE - $t_{CM} \leq T_{avg} \leq t_{MM}$

WHEN TO USE WHICH FORMULA

Follow the given checklist to determine the type of cache architecture –

1. Check if cache and memory access times are given or not.
    a. If not, then use Eq (1)
    b. Else, check if the words **hierarchical** or **level** are used in the question.
        i. If not, then use Eq (2)
        ii. Else, use Eq (3)

Question

Assume that for a certain processor, a read request takes 200 nanoseconds on a cache miss and 25 nanoseconds on a cache hit. Suppose while running a program, it was observed that 60% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is_____?

Answer

Use Eq (1) to get

$$T_{avg} = (0{,}6 * 25n) + (0.4 * 200n) = \mathbf{95}\boldsymbol{ns}$$

NOTE – There can be questions where they mention that Locality of Reference is used. In that case, the main memory access time is replaced by memory block access time. Thus, for all formulae, just replace t_{MM} with t_{block}.

CACHE WRITE AND WRITE PROPAGATION

Cache memory has a duplicate copy of the main memory data. When the cache interacts with the CPU, the CPU performs read or write operation on the cache data. In case of write operation, the cache data changes. Since the cache is a copy of the main memory, this change needs to be visible in the main memory also. This is the **Cache Write problem.**

Basically, we need to make sure that the updates to the cache are being propagated and reflected in the main memory as well. For this, we use two methods –

Write Through

The cache and CPU are updated simultaneously. In this case, the cache is not exactly useful because every time the cache is accessed for write, the main memory is also accessed, thus making this time consuming. However, the advantage for this system is that the data remains consistent and there are no discrepancies.

$$\left(T_{avg}\right)_{read} = (H * t_{CM}) + [(1 - H) * t_{MM}]$$

$$\left(T_{avg}\right)_{write} = \max(t_{MM}, t_{CM}) \equiv t_{MM}$$

Write Back

The cache in this case keeps getting updated but the updates are not sent to the CPU. Only when the cache block is being replaced to make space for a new block in the cache, the data is written back to the CPU. So, the memory is accessed once itself and thus is a time saving system. However, there is inconsistency when it comes to data in cache and main memory.

In both the above processes, the read operation remains the same. If it is a cache hit, the cache is accessed. Else, the main memory is accessed.

Question

A system has a write through cache with access time of 100ns and hit ratio of 90%. The main memory access time is 1000ns. The 70% of memory references are for read operations.

1. Average memory access time for read operations only
2. Average memory access time for write operations only
3. Average memory access time for read-write operations both
4. Effective Hit ratio

Answer

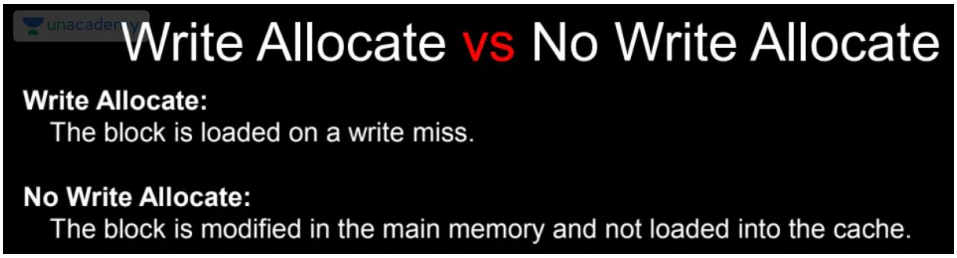
$$(T_{avg})_{read} = (0.9 * 100) + (0.2 * 1000) = \mathbf{190}\boldsymbol{ns}$$

$$(T_{avg})_{write} = \max(t_{MM}, t_{CM}) = \mathbf{1000}\boldsymbol{ns}$$

$$T_{avg} = (0.7 * 190) + (0.3 * 1000) = \mathbf{433}\boldsymbol{ns}$$

$$Effective\ Hit\ rate = (\%\ of\ read * Read\ hit\ rate) + (\%\ of\ write * Write\ hit\ rate)$$
$$= (0.7 * 0.9) + (0.3 * 0[since\ write\ through\ has\ no\ cache\ hit]) = \mathbf{0.63}$$

The average memory access time for read and write in the above question is correct only when the **read is simultaneous and not hierarchical.**



WRITE OP COMBO		HIT	MISS
Write Through	No Write Allocate	Perform write in cache and MM in simultaneous fashion.	Perform write in MM and do not bring the MM block to cache
Write Through	Write Allocate	Perform write in cache and MM in simultaneous fashion.	Perform write in MM and copy the block to cache
Write Back	No Write Allocate	Perform write in cache and write to MM only during replacement	Perform write in MM and do not bring the MM block to cache
Write Back	Write Allocate	Perform write in cache and write to MM only during replacement	Bring the block from MM to cache and then access cache to update the data.

NOTE – Write Through – No Write Allocation and Write Back – Write Allocate are the two configurations with maximum performance advantage.

CACHE MAPPING

CPU generates a main memory address to access data and then the cache checks if the content is present in it. However, how does cache know what is the data CPU requires since CPU has provided a main memory address. For this, there is a **mapping** between main memory and cache memory addresses. So, if we get the main memory, we can get the corresponding cache memory cell. This mapping is based on a **Hash Key Data Structure.**

There are mainly three types of cache mapping –

- Direct Mapping
- Set Associative Mapping
- Fully Associative Mapping

Direct Mapping

In this case, there is a direct logic to determine which block numbers from main memory are assigned to which block numbers in cache memory. Let us assume there are 10 blocks in cache (0-9) and 100 blocks in main mem (00-99). In this case, we can have the logic like –

$$Cache\ mem\ add = (Main\ mem\ add)\%10$$

Thus, blocks 00, 10, 20, 30, … 90 will be assigned to cache block 0 and so on for the other blocks. So if the CPU wishes to access main mem address 52, it will first access the cache memory block 2. If the cache hit, then no issues. Else, the CPU proceeds to access the main mem block 52 and then copy that block to cache mem block 2.

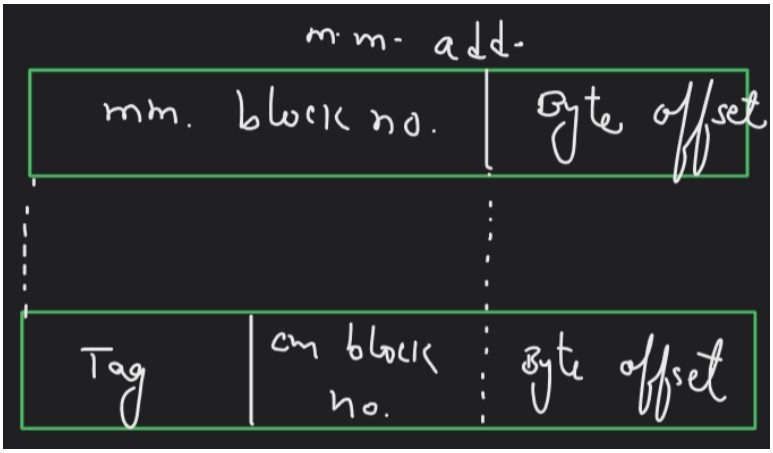
In addition to the data from main mem, the cache block also has a set of **Tag Bits** which provide the information on which block from main mem is stored in the respective cache block. So for example, in cache block 02 from our previous example, it is clear that the any of the main mem blocks from 02 to 92 can have their data in cache block 02. This is where tag bits come in to indicate which out of the 10 possible maim mem blocks' data is stored in cache block 02.

However, there is a direct disadvantage here. In this arrangement, blocks 02, 12, 22, … , 92 can't be in the cache at the same time. Only one block out of these should be present in the cache. Thus, there is a direct contention and consecutive access for blocks 12 and 22 (for example) will cause cache miss. Let us take the example here –

- **Step 1 –** The cache is initially empty. CPU requests address 12 from main mem. Cache is empty so cache miss occurs. Data from main mem block 12 is added to cache block 02 and tag is updated to 1.

- **Step 2 –** The CPU requests main mem block 22. Cache block 02 is checked. The data is present but the tag is 1 instead of 2. Therefore, the cache miss occurs and the data in cache block 02 is replaced by main mem block 22 and the tag is updated to 2.

This is a clear disadvantage of Direct Mapping. The whole address looks like –



The byte offset in the end is the byte required from the block. The CPU requests a single byte normally from a block and since the blocks are made of multiple bytes, the offset tells which byte needs to be accessed inside the block.

NOTE - In Direct mapping, the size of index is the size of the Cache mem block. First, we check the Cache memory block number in the main memory. If that block exists, then the tag bits are checked. If the tag bits match, then it is a hit. Else, it is a miss.

Also, we have –

$$Tag\ \ bits = \log_2\left(\frac{Main\ mem\ size}{Cache\ mem\ size}\right)$$

CACHE CONTROLLER

This is present in the cache chip with the cache memory. It has a small memory inside it which stores the tag information which is called the **metadata.** The metadata not only stores the tag bits, but also stores some additional information about the data. In short,

$$Cache\ controller = Cache\ Hit\ or\ Miss\ checker + Metadata\ storage$$

$$Metadata\ size = No\ of\ blocks\ in\ cache * Tag\ bits$$

CACHE INITIALIZATION

Cache mem is like RAM and it is volatile in nature. In short, when the Computer system is turned ON, the tags and info in the cache mem is **garbage** content. However, there is a chance that the CPU will try to

access the cache and by mistake, it might access the garbage value. So there is a need to tell the CPU that the value in the cache is a garbage value. This is where the **valid/invalid bit** system comes into play.

There is an additional bit apart from the tag and cache block data which is called the valid/invalid bit which is set to 0 when the cache block has garbage value. Else, it is set to 1. Thus, we have –

$$Metadata\ size = No\ of\ blocks\ in\ cache * (Tag\ bits + Extra\ bits)$$

Now, let us assume the cache is implemented using the write through architecture. In that case, there will never be any discrepancies between the cache mem and the main mem. However, when we use the write back architecture, there would be a write back needed. So for this case, we have a **dirty/modified** bit. When this bit is set, the write back is required. Else, write back is not required.

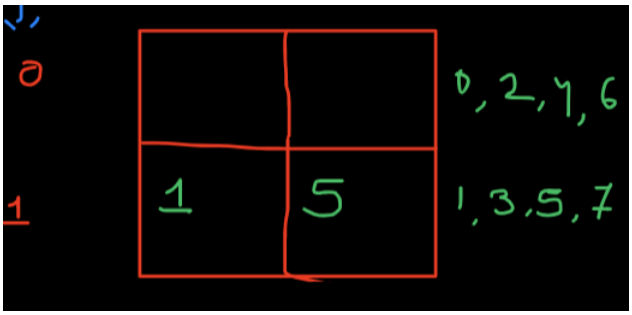
SET ASSOCIATIVE MAPPING

This is a solution to the Direct Mapping problem where there are multiple misses in cache as 1 block can accommodate only 1 possible main mem block. In this mapping, there are multiple blocks on a single index. Thus, the number of indexes needed have been reduced and these indices are called **sets.**

Let us assume a main mem has 8 blocks and the Direct Mapping cache has 4 blocks. In that case, we can have –

DIRECT MAP CACHE BLOCK	MAIN MEM BLOCK
1	1, 5
2	2, 6
3	3, 7
4	4, 8

Instead of this, we can change the mapping to a set associative mapping as shown below –



Now, we have only 2 indices but we accommodate 2 blocks on each index. Now, the arrangement becomes –

SET ASSOC. MAP CACHE BLOCK	BLOCKS AVAILABLE	MAIN MEM BLOCK
1	1, 2	1, 3, 5, 7
2	3, 4	2, 4, 6, 8

Now, in both the cases, let assume that the CPU requests blocks 1 and 5 repeatedly –

$$CPU\ request \rightarrow 1\ 5\ 1\ 5\ 1\ 5\ 1\ 5\ \ldots$$

For Direct Mapping

The first cycle is a miss and block 1 from main mem is added to block 1 of cache mem. In the next cycle, the cache is a miss and the block 5 of main mem is added to block 1 of cache mem. This cycle then continues and the cache miss occurs repeatedly.

For Set Associative Mapping

The first cycle is a miss and block 1 is added to set 1 block 1. The second cycle is also a miss and the block 5 from main mem is added to set 1 block 2. Now, the subsequent requests are all a hit.

In Set Associative mapping, the main mem address is given by –

$$Main\ mem\ add = Tag\ bits + Cache\ mem\ \boldsymbol{set\ number}\ (\boldsymbol{Set\ offset}) + Byte\ Offset$$

Question

A computer has a 512Kbyte, 4-way set associative, write back data cache with block size of 16 Bytes. The processor sends 34 bit addresses to the cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit

1. The number of bits in the tag field of an address is
2. The size of the cache tag directory is

Answer

$$Byte\ offset = \log_2 16 = 4b$$

$$Set\ offset = \log_2\left(\frac{2^{19}[512K]}{2^2[associativity] * 2^4[size\ of\ 1\ block]}\right) = 13b$$

Therefore,

$$Tag\ bits = 34 - 13 - 4 = 17b$$

Now,

$$Tag\ Directory\ Size = (no\ of\ blocks) * (Tag\ bits + Additional\ bits) = 2^{15} * (17 + 4) = 672Kb$$

CACHE HIT/MISS IN SET ASSOCIATIVE MAPPING

- First, the tag bit from the main memory instruction is extracted.
- Now, for each set we have multiple blocks. Each block will have a separate tag. Therefore, if we have a N – way associative mapping, then we extract N tags from the set.
- Now, we employ N comparators and compare each of the N tags with the tag bits in the main memory address.
- The results of these comparators are fed to an OR gate to get the final output. If any one of the comparators returns true, then we have a hit. Else, we have a miss.

NOTE – When we are changing from direct mapping to k-way associative mapping, then –

$$(No\ of\ tag\ bits)_{k-ass} = (No\ of\ tag\ bits)_{dir} + \log_2 k$$

$$(Set\ offset)_{k-ass} = (No\ of\ block\ bits)_{dir} - \log_2 k$$

FULLY ASSOCIATIVE MAPPING

For a cache with k blocks, the maximum associativity that we can achieve is k-way associative. In that case, the entire cache has just 1 set and all blocks are mapped to that 1 set. Therefore, here set offset field is not needed.

BLOCK REPLACEMENT

When we use a direct mapped cache, we don't need to have a replacement policy. This is because there is only one possible block that needs to be replaced. However, when we use k – associative mapping, we have k possible blocks that can replace the block in cache. Therefore, we need some sort of policy for block replacement. There are three possible replacement policies –

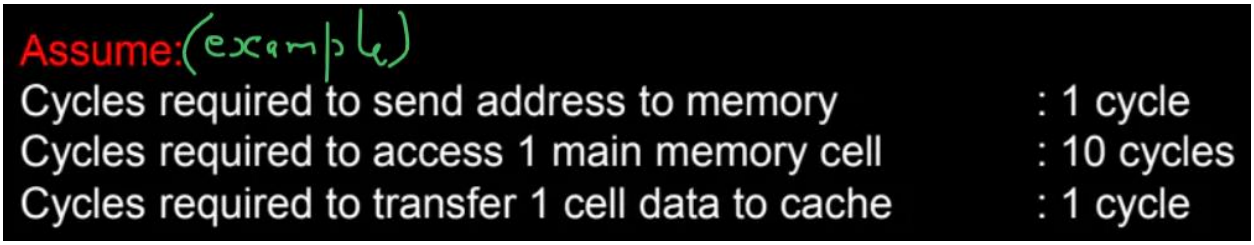
- **First-in First-out (FIFO) –** The first entry in the cache aka the oldest entry in the cache is replaced.
- **Least Recently Used (LRU) –** The cache block which was least used in the past is replaced.
- **Optimal –** The cache replaces the block that might not be frequently used in the near future.

MISS PENALTY

Miss penalty is the amount of time taken to copy a block of data from the main mem to the cache mem when there is a cache miss. This is not included in avg mem access time until and unless it is explicitly specified. During the miss cycle, the steps followed are –

- The CPU sends a single instruction to the main mem.
- The block data is copied to the cache mem from the main mem
    - Every block has multiple bytes of data
    - This means that multiple times the main mem needs to be accessed.
    - Then every time main mem is accessed, a byte is copied to the cache mem.

For example,



Let us assume a block has 4 memory cells. Then,

$$Miss\ penalty = 1 + (4 * 10) + (4 * 1) = \mathbf{45\ cycles}$$

Now suppose in the above example, the memory is word addressable, and each memory cell has 2B of data. Then only 2 access cycles are needed to transfer 4B of data. Hence,

$$Miss\ Penalty = 1 + (2 * 10) + (2 * 1) = \mathbf{23\ cycles}$$

TYPES OF CACHE MISS

There are three main types of cache miss –

Cold or Compulsory Miss

This is when the CPU is referencing a cache block for the first time. This miss is unavoidable. One way to reduce the number of cold misses is to increase the block size, thus reducing the number of blocks and also the number of cold misses.

Capacity Miss

This type of miss occurs when the cache memory is full and therefore a block replacement is needed. The way to reduce capacity miss is to increase the size of the cache memory.

Conflict Miss

This type of miss occurs when the cache set is full and there is a tag mismatch. To reduce the conflict miss, we can increase the associativity.

The priority of misses is as follows –

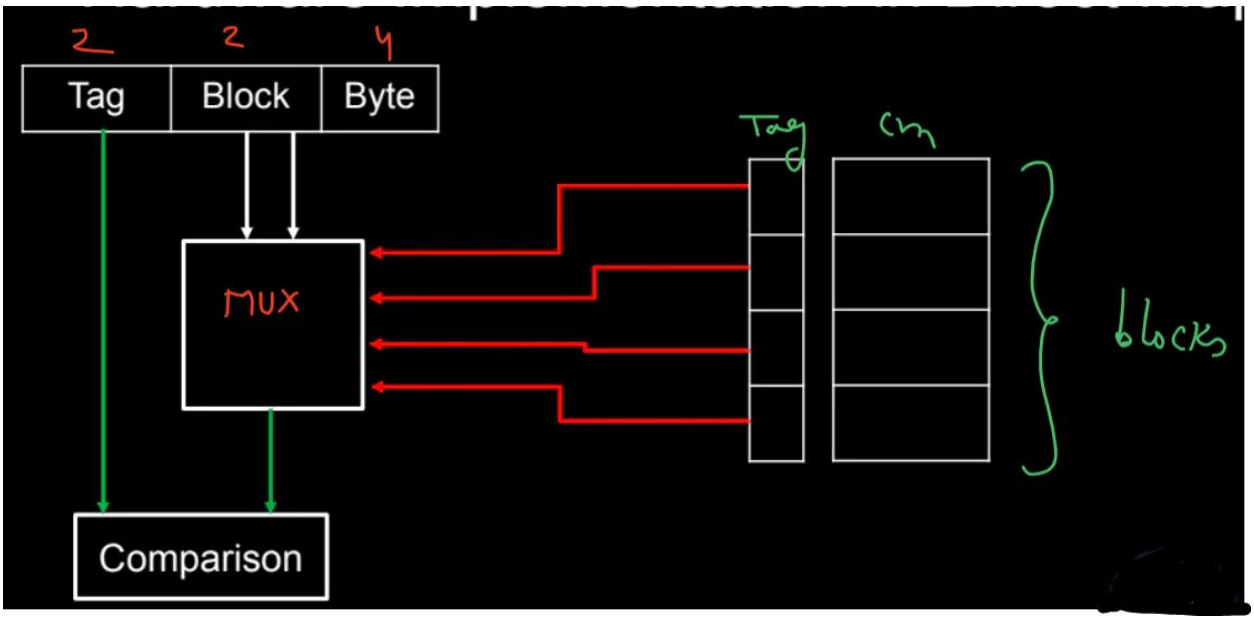
$$Cold\ Miss > Capacity\ Miss > Conflict\ Miss$$

This means that if a cold miss occurs, then we don't need to check for capacity or conflict misses.

NOTE – In fully associative, we have only one set. This means that if the set is full, then the entire cache is full. Therefore, in this case, the conflict miss doesn't exist and we can either have a cold or capacity miss.

HARWARE IMPLEMENTATION OF HIT/MISS – DIRECT MAPPING

In Direct mapping, Hit/Miss hardware is shown below –

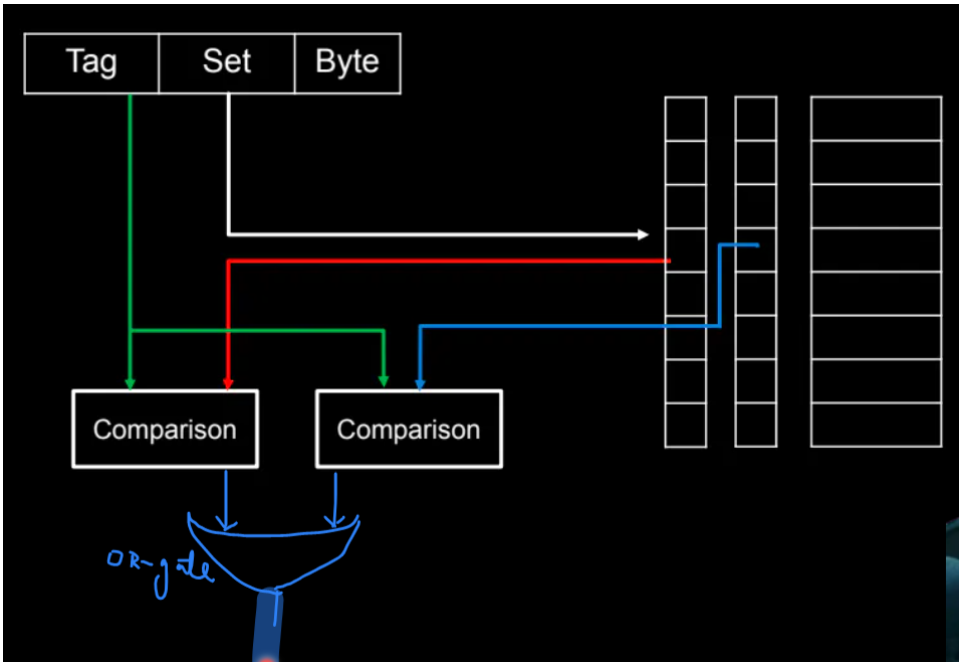


In this case, first the block bits from the memory address are sent to the MUX as select lines. The MUX inputs are the tag bits from all the blocks. Based on the select lines, the respective block tag bits are sent to the comparator. The second input to the comparator are the tag bits in the memory address. In case the comparator returns a 0, then it is a match and therefore, it is a hit. Else, it is a miss.

One thing to remember is that a MUX can only take 1 bit as an input line. Therefore, multiple MUXs are needed for extracting all the tag bits information. This entire hardware also takes some time to get the Hit or Miss information. This time is called **Hit Latency.**

$$Hit\ Latency = 1\ MUX\ delay\ for\ tag\ selection + 1\ comparator\ delay$$

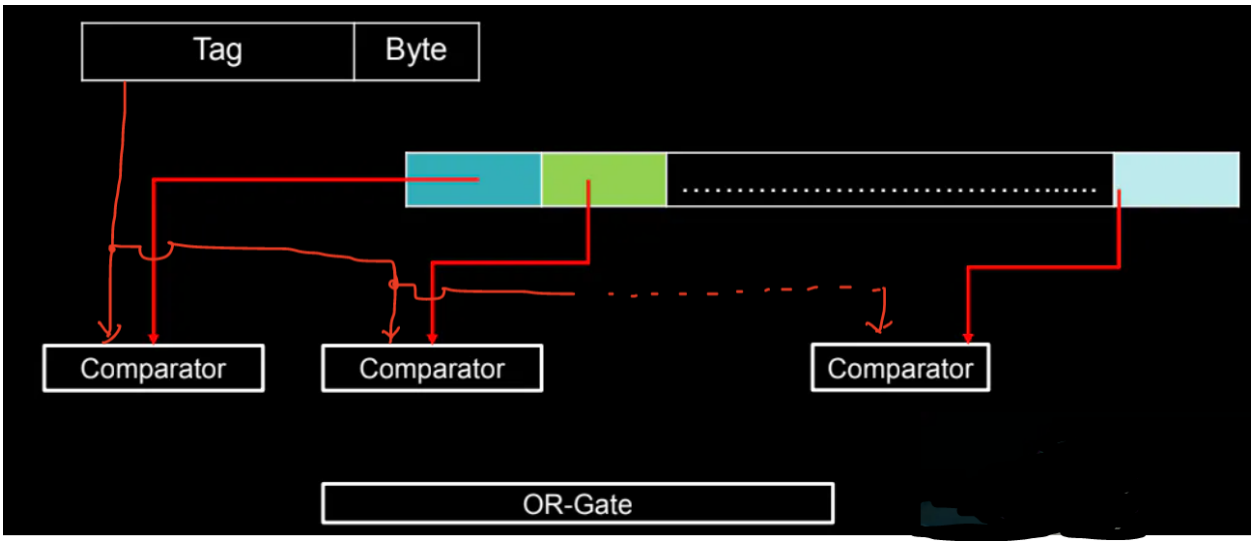
HARWARE IMPLEMENTATION OF HIT/MISS – SET ASSOCIATIVE MAPPING



The arrangement here is similar to the direct mapping cache. However, whenever we access a set, we are accessing a bunch of blocks and each block has its own tag. Thus, we need multiple comparators to check the tag hit/miss. If either of these tag bits are matching, then it is a cache hit. Thus, we send the outputs of the comparators to a OR gate to get the final hit/miss. Here, we can calculate the hit latency as –

$$Hit\ latency = MUX\ delay + Comparator\ delay + OR\ gate\ delay$$

HARWARE IMPLEMENTATION OF HIT/MISS – FULLY ASSOCIATIVE MAPPING

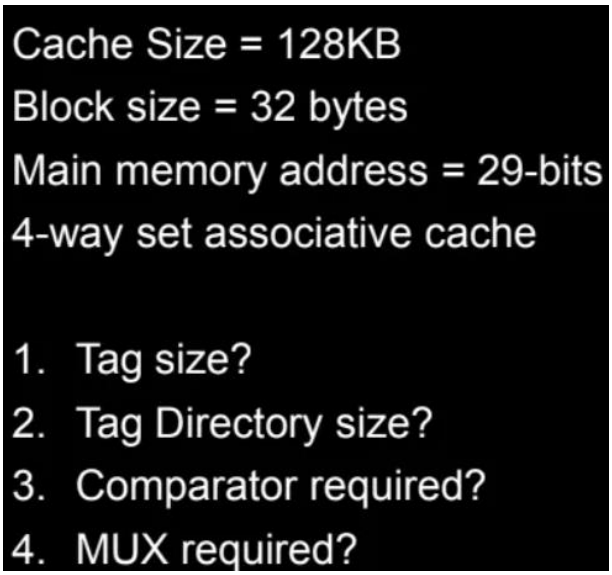


In this case, there is only 1 set and all the blocks are present in that set. Thus, we need to check the tag bits of all the blocks. Hence, there is no need for a MUX since we are performing tag selection. All the tag

bits are sent to the comparators and then the outputs of the comparators are sent to the OR gate to get the final hit/miss value. Here, we have

$$Hit\ latency = Comparator\ Delay + OR\ gate\ delay$$

Question



Answer

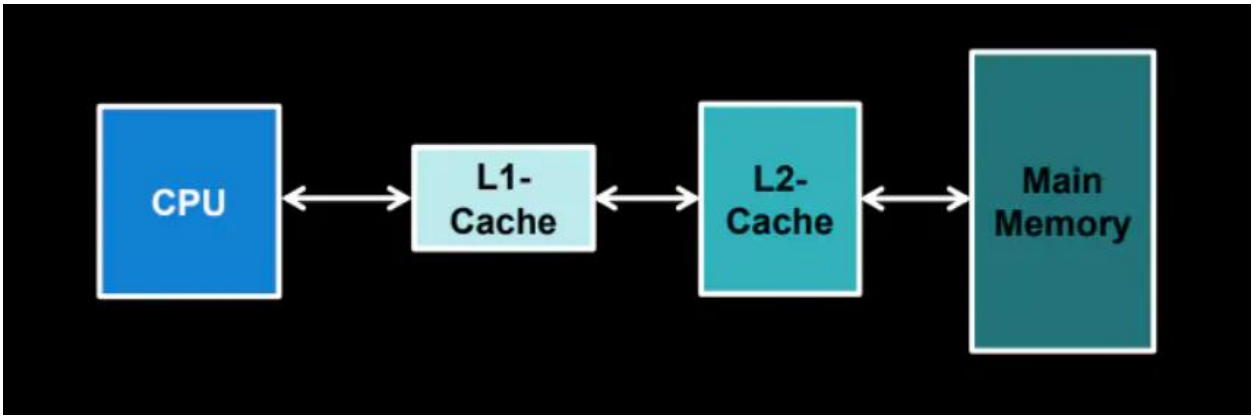
1. 14 bits
2. $14 * 2^{12}$ bits
3. 4 (k = 4)
4. 4*14 = 56 MUX

GOALS OF USING CACHE

- Minimize access time
- Maximize Hit rate
- Minimize miss penalty

Now, if we want to reduce the access time, we would need a smaller cache so that it is faster to access it. On the other hand, we would require a large cache (with more data) to have a larger hit rate. There is a clear contradiction. This is solved using a multi – level cache.

MULTI – LEVEL CACHE



This way, we have 2 cache – one small and one larger. That way, we can increase the hit ratio while also reducing the overall access time. Here, let us assume –

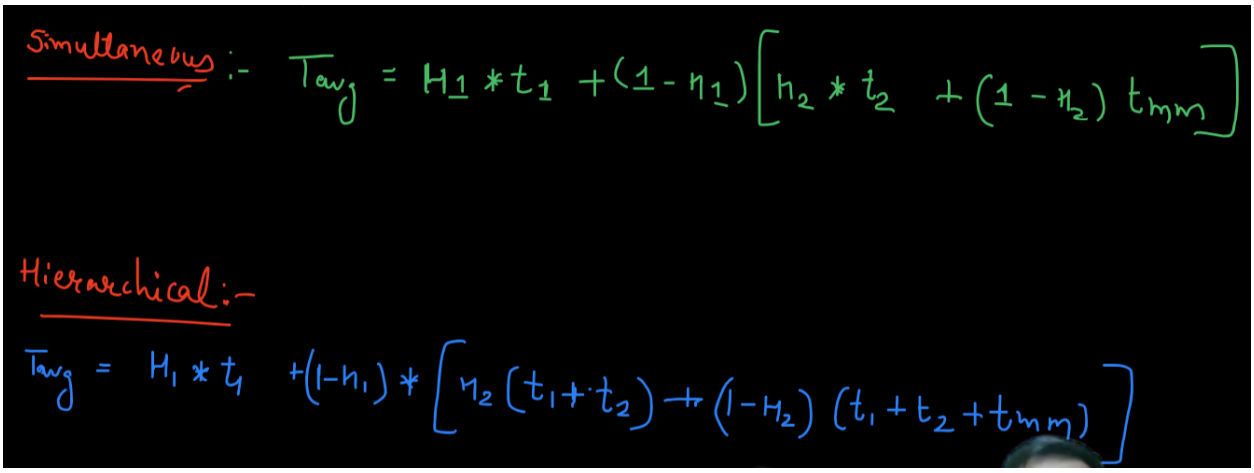
- $Access\ time\ for\ L1 = t_1$
- $Access\ time\ for\ L2 = t_2$
- $Access\ time\ for\ Main\ mem = t_{mm}$
- $Hit\ ratio\ of\ L1 = h_1$
- $Hit\ ratio\ of\ L2 = h_2$
- $Hit\ ratio\ of\ main\ mem = h_{mm} = 1$

Then, we can say that –

$$t_1 < t_2 < t_{mm}$$

$$h_1 < h_2 < h_{mm}$$

To get the average access time, we have again 2 types of access –



$$Probability\ of\ access\ of\ L1 = P(L1) = H1$$

$$Probability\ of\ access\ of\ L2 = P(L2) = H2 * (1 - H1)$$

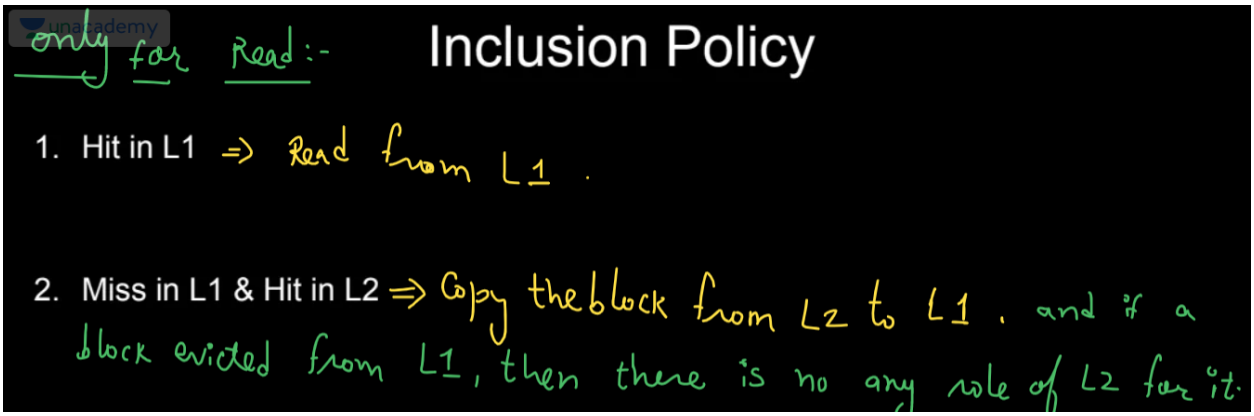
$$Probability\ of\ access\ of\ Main\ mem = P(MM) = (1 - H1)(1 - H2)$$

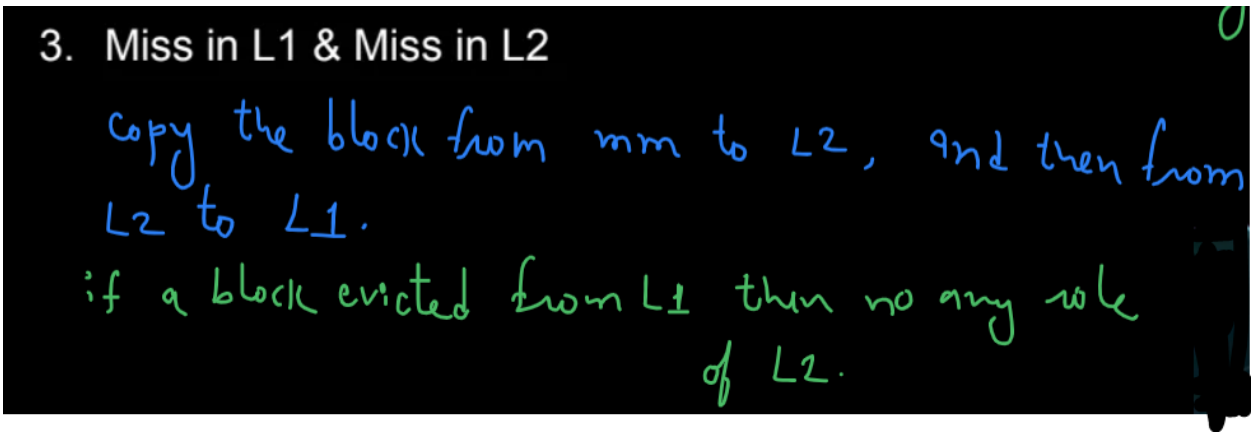
Hence,

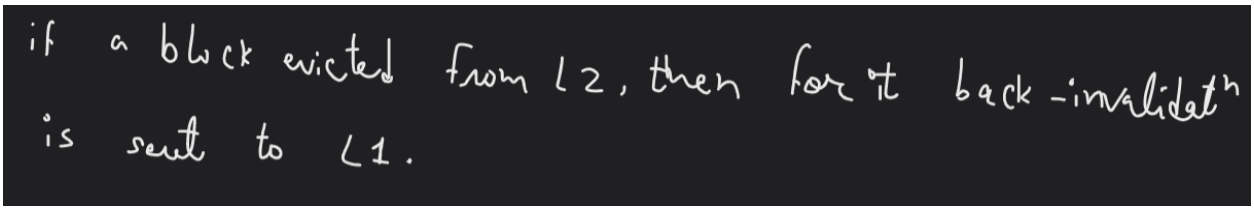
$$T_{avg} = P(L1) * t_1 + P(L2) * (t_1 + t_2) + P(MM) * (t_1 + t_2 + t_3)$$

Inclusion Policy

The contents present in L1 must also be present in L2 aka L2 must be **inclusive** of L1. However, if a block is present in L1 and also in L2, it is not necessary that these 2 blocks have the same value in them. In case the values are the same, then it is called a Value Inclusive Policy.

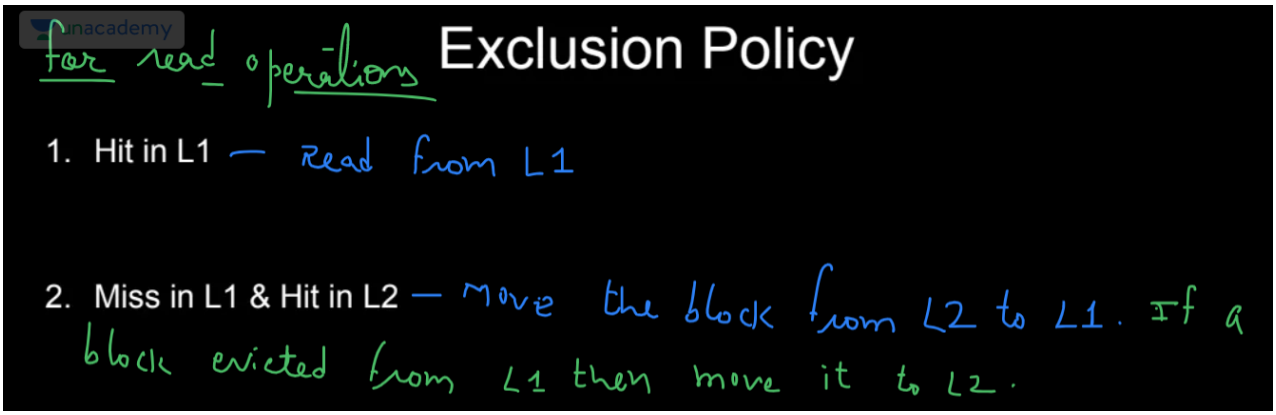


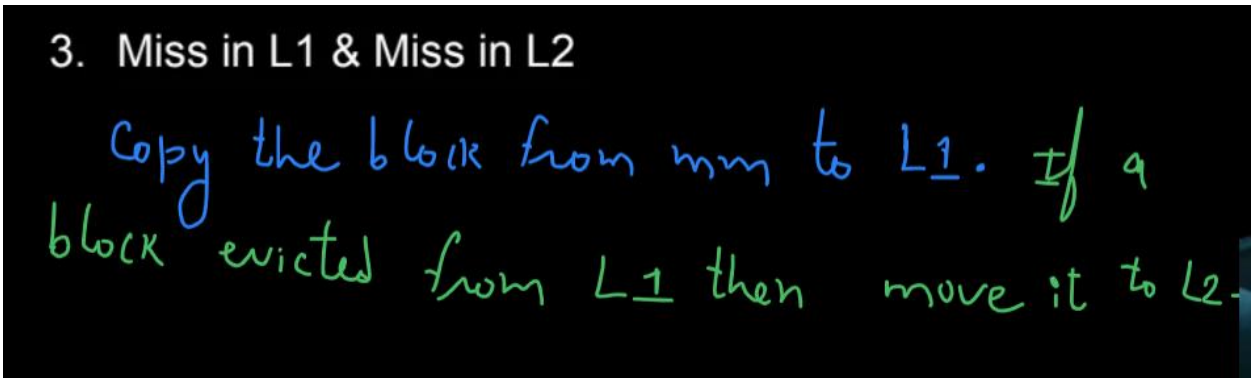




Exclusion Policy

On the contrary, the content of L1 should not be necessarily present in L2.

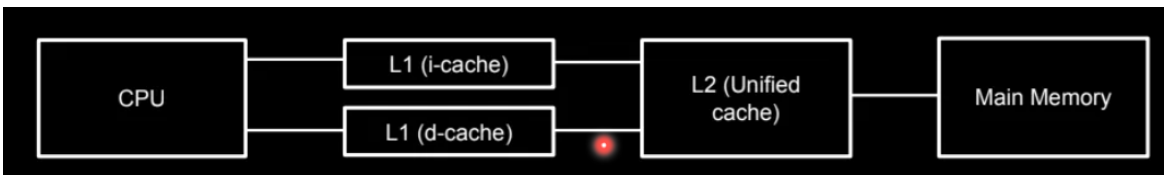




CACHE COHERENCE PROBLEM

This is the problem that occurs when a **shared memory system** is used. In this system, we have multiple processors, cache mems but one single main mem. Then, when the cache access the same data from the main mem, there is a chance of inconsistency which is called the **cache coherence problem.**

DUAL CACHE

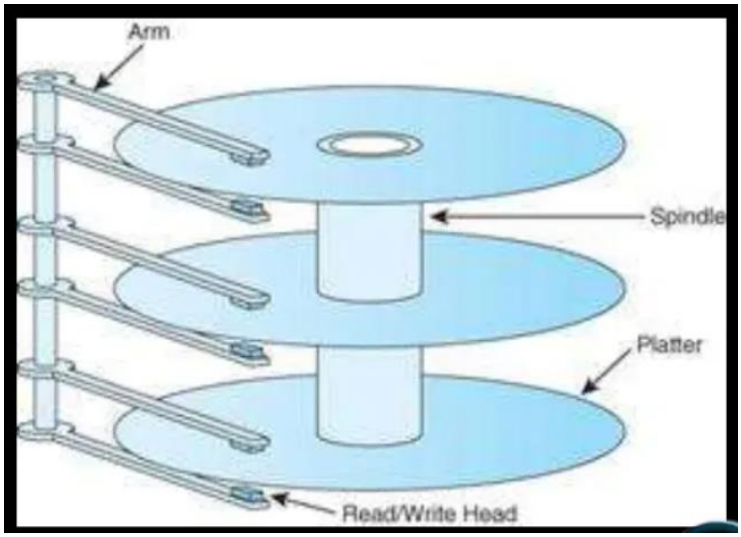


In this case, the first level of cache is split into two cache – one to store data and the other to store instructions. This will reduce the cache size for L1 and thus increases the access speed.

MAGNETIC DISK

This is an external memory and is treated as an I/O device. It is built of multiple CD like disks which are called **platters.** Each platter has a top and bottom surface and thus it is said that each platter has **2 recording surfaces.** Each surface works like a gramophone disk where the disk rotates and a pointer selects the part to be played.

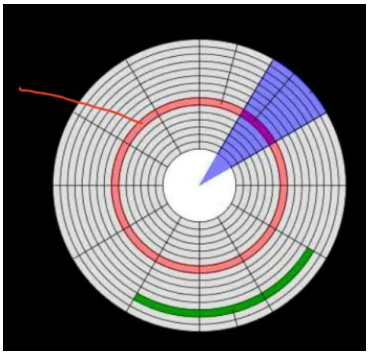
In a similar fashion, all the platters in the disk are connected by a **spindle** which spins and a **pointer/read-write head** points to the area in the memory where the read/write operation needs to be performed. Each surface has its own read-write head and these read-write heads are supported by **arms** which connect to the **arm assembly.**



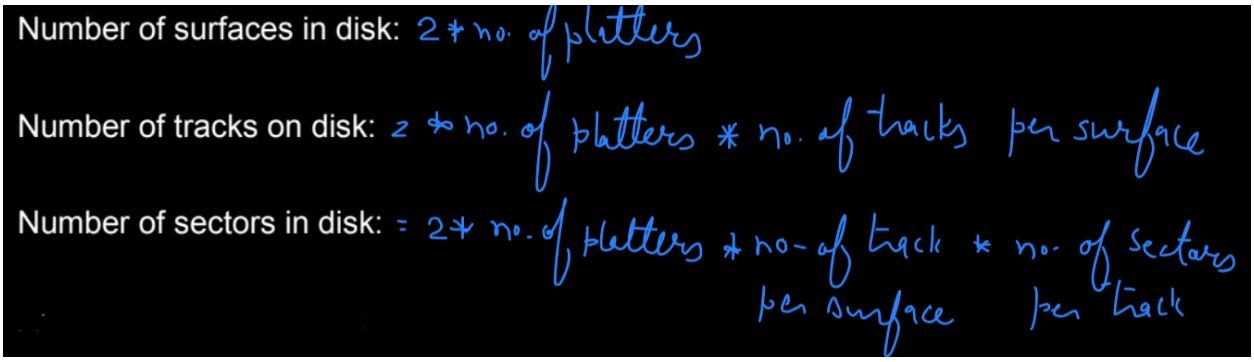
The arms are responsible to change the radius of the circle that the read-write head traces over the surface and this changes the memory part to be accessed. In this disk, the movement of arms and the platters is mechanical and therefore, the memory access is slow af.

NOTE – Read/write occurs on a **single surface at a time.**

Every surface has **concentric circular tracks** on which the read-write head traces.



- **Red –** A single track
- **Red part in the Blue –** A sector of the surface. Sector is smallest addressable unit.
- **Green –** A cluster of sectors of a single track



SECTOR CAPACITY

The capacity of a structure depends on the disk organization method –

- Fixed sector storage capacity (In syllabus)
- Variable sector storage capacity (NOT IN SYLLABUS)

We always consider a Fixed sector storage capacity organization. This means that despite the sector area, the number of bytes are the same. This means that the sector has a **variable storage density.**

DISK ACCESS TIME

Time taken to read/write data in one sector.

$$Disk\ access\ time = Seek\ time + Rotational\ Latency + Transfer\ time$$

Seek Time: Time required to position the arm over the desired track
Rotational Latency: time required to rotate desired sector under R/W head
Transfer Time: Time required to read or write 1 sector

Question

A disk has each track capacity of 2MB and it takes 20msec for 1 rotation. The transfer rate of the disk is?

Answer

It takes 20ms to transfer 2MB of data. Thus, the disk would transfer 100MB is 1 sec. Hence, rate – **100MBps.**

MULTIPLE SECTOR TRANSFER

Since we are transferring multiple sectors, we can assume that these sectors are present on the same track. Let us assume we need to transfer n such sectors. Then, we have

$$Disk\ access\ time = Seek\ time + Rotational\ Latency + (n * Sector\ transfer\ time)$$

On the other hand, if these sectors are not consecutive, then we need to calculate the disk access time for each sector and then add them up.

CYLINDER

One thing to remember is that all read/write heads move simultaneously. So if 1 read/write head is pointing to 1 track on a surface, all the other read/write heads will also be pointing to that track on different surfaces. So instead of storing the data track wise, the system stores data on the same track number at different surfaces. This way, we don't need seek time as the read/write heads are already pointing to the right track.

In short, all the tracks across surfaces that **have the same radius** are accessed first while storing data. These group of tracks that have the same radius make a **cylinder.** Thus, the disk stores the data cylinder wise to save seek time.

$$No\ of\ cylinders\ in\ disk = No\ of\ tracks\ per\ surface$$

DISK ADDRESSING

In a disk, every sector gets a unique address. To identify a sector, we first need to identify a cylinder, then identify the surface and then identify the sector.

Question

A hard disk has 16 sectors per track, 4 platters each with 2 recording surfaces and 32 cylinders. The address of a sector is given as a triple ⟨c,h,s⟩, where c is the cylinder number, h is the surface number and s is the sector number. Thus, the 0th sector is addressed as ⟨0,0,0⟩, the 1st sector as ⟨0,0,1⟩, and so on.

The address ⟨12, 6 ,12⟩ corresponds to sector number?

Answer

$$Total\ no\ of\ surfaces = 8$$

$$Total\ no\ of\ tracks\ per\ surface = 32$$

$$Total\ no\ of\ sectors\ per\ surface = 32 * 16 = 512$$

Therefore, we have

$$Sector\ No = (c * No\ of\ sectors\ in\ 1\ cylinder) + (h * No\ of\ sectors\ in\ 1\ track) + s$$

$$Sector\ No = 1644$$

PARALLEL PROCESSING

This is the simultaneous data processing. It is of three types –

- Vector processing (NOT IN SYLLABUS)
- Array processing (NOT IN SYLLABUS)
- Pipelining

FLYNN'S CLASSIFICATION OF COMPUTERS

- **Single Instruction stream Single Data stream (SISD) –** It addresses the next instruction only when the current instruction finishes execution.
- **SIMD –** Single instruction is fetched per cycle, however these instructions can be executed parallelly. Pipeline processors follow this system.
- **MISD –** Multiple instructions are fetched per cycle but only one instruction is addressed at a time. This is only a theoretical model as this system makes no sense since we are still addressing only single instruction at a time.
- **MIMD –** Here, multiple instructions are fetched in a cycle and multiple instructions are addressed at a time. This is a super scalar computer where we have multiple pipelines.

NOTE – The Flynn's classification is applicable for **Single Processor systems.**

PIPELINE PROCESSING

Pipeline processing is useful when the same processes are applied over and over again on multiple inputs. To enable parallel processing, we first, every **task** is divided into various **sub – operations.** Each sub – operation is sent to a specific hardware (aka **segment**) for processing. These segments can work parallelly.

Example

Let us take the operation as –

$$A_i * B_i + C_i \qquad for\ i\ \epsilon\ [1,6]$$

Now, we can divide the whole operation into sub – operations as shown below –

Sub-operations:

seg1: $R1 \leftarrow A_i,\ R2 \leftarrow B_i$

seg2: $R3 \leftarrow R1 * R2,\ R4 \leftarrow C_i$

seg3: $R5 \leftarrow R3 + R4$

We can track the operations as follows –

Clock pulse number	Segment 1 R1	Segment 1 R2	Segment 2 R3	Segment 2 R4	Segment 3 R5
1	A_1	B_1			
2	A_2	B_2	$A_1 * B_1$	C_1	
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	-	-	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8					$A_6 * B_6 + C_6$

In this case, we can see that by the end of 8 cycles, we have performed the operation. Now when we compare this to a normal sequential operator, we can see that one operation takes **3 cycles** for operation. Thus, the total process will take **18 cycles** to complete.

PIPELINE CYCLE TIME

Since in a pipeline we have multiple segments operating parallelly, we need to find a cycle time that allows all the segments to finish operation. The pipeline cycle time is the minimum possible time in which all the segments can perform their respective sub – operations. Basically –

$$Pipeline\ Cycle\ time = \max\ (Segment\ operation\ times)$$

Let us assume that the Pipeline cycle time is t_p and we are using k segment pipeline to perform n operations. Then, we have –

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

$$Total\ no\ of\ cycles\ taken = k + n - 1$$

Now, if we consider a sequential system for the same process with a cycle time of t_n, then

$$Total\ time\ taken = n * t_n$$

We can also calculate the Speed up factor as –

$$S = \frac{n * t_n}{(k + n - 1) * t_p}$$

NOTE – When the number of tasks n increases, the difference speed up factor increases exponentially as sequential system time taken increases exponentially. When $n \gg k$, then we have –

$$S_{ideal} = \frac{t_n}{t_p}$$

Now, there can also be a case where $k * t_p = t_n$ and $n = 1$. In that case, we have

$$S_{ideal} = k$$

Also, **Max possible speed up = k.**

Question

A non-pipeline system takes 50 ns to process a task. The same task can be processed in a six-segment pipeline with a clock cycle of 10ns.

1. Determine the speedup ratio of the pipeline for 100 tasks.
2. What is the maximum speedup that can be achieved?

Answer

The speedup for 100 tasks –

$$S = \frac{100 * 50n}{(100 + 6 - 1) * 10n} = \mathbf{4.76}$$

Max speed up,

$$S_{ideal} = \frac{50n}{10n} = \mathbf{5}$$

SYNCHRONOUS PIPELINES

As seen previously, the different segments in the pipeline have different delays. Thus, normally additional buffers are added to the segments to ensure synchronous operation. In this case,

$$Pipeline\ Cycle\ time = \max(Segment\ delays) + Buffer\ delay$$

One thing to note here is that these buffers are not needed in the sequential operation and thus, the buffer delay doesn't affect the sequential operation.

NOTE – To improve the speed up for pipeline structure, we can try to have segments with same delays.

LATENCY AND THROUGHPUT

The time passed before the next input enters the system is called **Latency.** For non – pipeline system, the new input comes only after the current instruction finishes execution. Thus,

$$Latency = t_n$$

As for the pipeline system,

$$Latency = t_p$$

On the other hand, the number of operations/jobs per unit time is called the **Throughput.** In short,

$$Throughput = \frac{Total\ number\ of\ operations}{Time\ taken\ to\ complete\ all\ operations}$$

For non – pipeline system,

$$Throughput = \frac{n}{n * t_n} = \frac{1}{t_n}$$

For pipeline system,

$$Throughput = \frac{n}{(n + k - 1) * t_p}$$

In Ideal case, we have

$$Throughput_{ideal} = \frac{1}{t_p}$$

INSTRUCTION PIPELINE

We know that every instruction requires a basic set of steps to execute like instruction fetch, instruction decode, operand fetch etc. This means that for instructions, the system needs to perform the same process again and again. This is the ideal case for pipelining. When the pipeline is implemented in an instruction cycle, it is called an **Instruction Pipeline.**

In this pipeline, we first divide the whole process into sub – processes and assign the sub – processes to a segment each. Let us take a 5 – segment pipeline as shown below –

Now, the pipeline will function as shown below –

INS/CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1	IF	ID	OF	EX	WB										
I2		IF	ID	OF	EX										
I3			IF	ID	OF										
I4				IF	ID										
I5					IF										
I6															
I7															
I8															
I9															

This arrangement seems fine up until now, but there is an inherent problem here. Let us assume that I4 is a BRANCH/JUMP instruction. Thus, in Cycle 5, the system knows that I4 is a branch instruction. However, only after I4 executes will the system know whether I4 will jump the control to another instruction or will the condition return false causing I5 to be executed next. In short, when jump/branch instruction comes, the system doesn't know which instruction to execute next until the branch instruction finishes execution. Thus, the next steps in the pipeline are as follows –

INS/CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1	IF	ID	OF	EX	WB										
I2		IF	ID	OF	EX	WB									
I3			IF	ID	OF	EX	WB								
I4				IF	ID	OF	EX								
I5					IF	-	-								
I6															
I7															
I8															
I9															

At this point, I4 has finished execution and we know the next instruction to be executed after I4. Let us assume that the condition I4 checked returned True and it advised to jump to I6. Then the final pipeline looks like –

INS/CLK	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1	IF	ID	OF	EX	WB										
I2		IF	ID	OF	EX	WB									
I3			IF	ID	OF	EX	WB								
I4				IF	ID	OF	EX	WB							
I5					IF	-	-								
I6								IF	ID	OF	EX	WB			
I7									IF	ID	OF	EX	WB		
I8										IF	ID	OF	EX	WB	
I9											IF	ID	OF	EX	WB

Here,

$$No\ of\ instructions\ executed = 8$$

$$No\ of\ cycles\ normally\ needed = 8 + 5 - 1 = 12$$

$$Cycles\ taken = 15$$

Therefore, we have 3 cycles extra needed to finish the execution due to a jump instruction. These extra cycles are called **Stall Cycles.** In general, if we get the target address after i^{th} stage (in our case 4th stage which is Execution stage), then the number of stall cycles will be $i - 1$ (in our case 3).

PIPELINE HAZARDS

These are situations which prevent the next instruction from being executed during its designated clock cycle. There are three types of hazards possible –

- **Structural Hazards/Resource Conflict –** This occurs when two or more segments try to access the same resource at the same time. In this case, one of the instructions has to wait for the other to finish using the resource.
- **Data Hazards/Data Dependency –** In this case, the result of an instruction acts as an input to the next. This means that until and unless the previous instruction finishes execution and writes back the result, the current instruction can't execute.
- **Control Hazards/Branch Difficulty –** This occurs when there is a branch instruction. The execution phase of the branch instruction gives the information on which instruction to work on next.

DELAYED LOAD

This is a software solution provided by the compiler to overcome the Data Dependency problem. In this, the compiler purposely delays the operand fetch for the dependent instruction. Within this delay between the two dependent instructions, the compiler tries to fit in some of the independent instructions to prevent a loss of time and resources. Suppose there is no independent instruction that can be executed between the two dependent instructions, then the compiler will add a **No Operation (NOOP)** instruction instead.

HARDWARE INTERLOCK

This is a hardware solution where the hardware segments are locked for the dependent instructions when the parent instruction is operating. This way, there is no data dependency problem however there is an addition of stall cycles.

$$Total\ no\ of\ cycles\ with\ hazard = Total\ no\ of\ cycle\ without\ hazard + (stall\ cycle\ per\ instruction * No\ of\ dependent\ Instructions)$$

OPERAND FORWARDING

This is the other hardware solution which was designed mainly to prevent the stall cycles as seen in Hardware Interlock system. Here, the first instruction undergoes normal operation and the second instruction (dependent one) is undergoing a special operation.

Previously, the second instruction would stall after Instruction Decode and wait for the first instruction to provide the output. However, here the second instruction does not stall. It performs operand fetch where it fetches the operands that are not dependent. By this time, the first instruction finishes execution and stores the result in the ALU accumulator.

Now, the second instruction uses the independent operand and the ALU accumulator value to perform the execution. So, there is no stall cycles here. Instead of waiting for the dependent operand to reach the memory for fetching, the second instruction simply uses it from the ALU itself.

DELAYED BRANCH

This is a software solution for the Control Hazard. Here, the compiler adds a few independent or NOOP instructions after the branch instructions. This allows the output of the branch instruction to be computed before determining the next instruction to be executed.

BRANCH PREDICTION

This is a hardware solution for the Control Hazard. As the name suggests, the hardware in this case is smart enough to predict whether the output of the branch instruction returns true or false with almost 85% accuracy. In case the prediction is correct, this solution saves time. However, before prediction, the CPU stores the state before the prediction process in separate registers. Thus, if the prediction is wrong, the CPU restores the state using those register values.

Branch prediction is of two types –

- **Static –** In this case, the hardware always assumes that the output of the branch instruction is true or false. Like always true or always false.
- **Dynamic –** In this case, the hardware studies the program state and previous states and predicts whether the output will be true or false.

DATA HAZARD CLASSIFICATION

Assume that there are 2 instructions i and j where i is executed before j. Then there are four types of data hazards –

HAZARD TYPE	IDEAL OPERATION	HAZARDOUS OPERATION	ALTERNATE NAME	HARDWARE SOLUTIONS
Read After Write (RAW)	First i changes the data and then j reads it	j reads the data before i can change it.	True Data Dependency	Operand Forwarding
Write After Write (WAW)	First i writes a data in an operand and then j writes on the same operand	j writes over the operand before i writes it.	False Data Dependency	Register Renaming (store output for each instruction in different registers)
Write After Read (WAR)	First i reads the register value and then j changes the register values	j writes the register value and then i reads the incorrect value	False/Anti – Dependency	Register Renaming

Question

Consider a pipelined processor with the following four stages:

IF: Instruction Fetch ID: Instruction Decode and Operand Fetch

EX: Execute WB: Write Back

The IF, ID, and WB stages take one clock cycle each to complete the operation. The number of clock cycles for the EX stage depends on the instruction. The ADD and SUB instructions need 1 clock cycle and the MUL instruction needs 3 clock cycle in the EX stage. Operand forwarding is used in the pipelined processor. What is the number of clock cycles taken to complete the following sequence of instructions?

```
R2←R1+R0
R4←R3*R2
R6←R5-R4
```

Answer

INS/CLK	1	2	3	4	5	6	7	8
I1	IF	ID	EX	WB				
I2		IF	ID	EX	EX	EX	WB	
I3			IF	ID	-	-	EX	WB

As per operand forwarding, there should be no **stall cycle** due to data hazard. At the same time, since there are no branch type instructions, we are not facing any **control hazard** as well. The only hazard here is the structural hazard as the execution hardware will be busy in the 2nd instruction and will not be available to the 3rd instruction till 7th cycle.

Therefore, total number of cycles = **8 cycles.**

Question

The instruction pipeline of a RISC processor has the following stages: Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO) and Writeback (WB), The IF, ID, OF and WB stages take 1 clock cycle each for every instruction. Consider a sequence of 100 instructions. In the PO stage, 40 instructions take 3 clock cycles each, 35 instructions take 2 clock cycles each, and the remaining 25 instructions take 1 clock cycle each. Assume that there are no data hazards and no control hazards.
The number of clock cycles required for completion of execution of the sequence of instruction is _____?

Answer

$$No\ of\ cycles\ without\ hazard = 100 + 5 - 1 = 104$$

40 instructions take 3 cycles for each execution. That means, each instruction will cause 2 stall cycles. The 35 instructions take 2 cycles for each execution i.e. each instruction will cause 1 stall cycle. The rest of the 25 instructions will take a single cycle, so no stall cycles will be needed. Hence,

$$No\ of\ stall\ cycles = (40 * 2) + (35 * 1) = 115$$

Therefore, total number of cycles will be **219 cycles.**

NOTE

In the above question, let us assume that Operand Forwarding is not used and we will have data dependency. In that case, the next instruction's OF can only happen after the previous instruction finishes its WB. Since the WB is the 5th stage and the OF is the 3rd stage, each instruction will have **5-3 = 2 stall cycles** added due to **data dependency.** Hence, the total number of cycles will become **419 (**assuming each instruction is dependent on the previous instruction).

CPI CALCULATION IN PIPELINE

We know,

$$CPI = \frac{No\ of\ cycles\ in\ total}{No\ of\ instructions} = \frac{k + n - 1}{n}$$

Now, this is in case there are no stall cycles and hazards. In case there are additional stall cycles, then the CPI changes to –

$$CPI = \frac{k + n - 1 + x}{n} \qquad where\ x \rightarrow No\ of\ stall\ cycles$$

In ideal conditions, $k = 1$. Thus, we have

$$(CPI)_{ideal} = \frac{n + x}{n} = 1 + \frac{x}{n}$$

Question

Consider a 5-stage pipeline which is executing a program of 1000 instructions. Among all instructions 200 instructions cause 2 stall cycles each.

1. Calculate CPI of pipeline?
2. If pipeline cycle time is 3ns then what is average instruction execution time?
3. Calculate CPI of pipeline in ideal conditions?
4. If pipeline cycle time is 3ns then what is average instruction execution time in ideal conditions?

Answer

Given,

$$k = 5$$

$$n = 1000$$

$$x = 200 * 2 = 400$$

Therefore,

$$CPI = \frac{5 + 1000 - 1 + 400}{1000} = \mathbf{1.404}$$

Now, it is given that

$$t = 3ns$$

Therefore,

$$Execution\ time\ of\ 1\ instruction = CPI * t = \mathbf{4.212ns}$$

Under ideal conditions,

$$(CPI)_{ideal} = 1 + \frac{400}{1000} = \mathbf{1.4}$$

Also,

$$Avg\ time\ for\ 1\ instruction\ execution = 1.4 * 3 = \mathbf{4.2ns}$$

CLASSIC RISC PIPELINE

This is similar to the general pipeline structure and has 5 stages as shown below –

- **Instruction Fetch**
- **Instruction Decode** and Register Read (aka Operand Fetch). For branch instructions, the target address calculation, condition evaluation and Program Counter value update occurs in this stage itself
- **Execution**
- **Memory** is accessed for load or store operations.
- **Write Back** result of ALU operation or load operation

Question

Consider the following program segment. Assume $R1$ and $R2$ are GPRs and the initial value at memory location 1000 is **5.** Find the final value in $R2$ at the end of the program execution.

	Instructions	Operations
	MOV R1, (1000)	R1 ← M[1000]
	MOV R2, #10	R2 ← #10
LOOP:	ADD R2, R1	R2 ← R2 + R1
	DEC R1	R1 ← R1 – 1
	BNZ LOOP	Branch on not zero
	HALT	Stop

Answer

Initially, $R1 = 5$ and $R2 = 10$. The program then loops the rest of the statements till the zero flag in the Status Register in the ALU is not set. The status register sets after each operation.

ITERATION	$R2$	$R1$
1	15	4
2	19	3
3	22	2
4	24	1
5	25	**0**

Once $R1$ is assigned to zero, the zero flag will be set. Hence, the program breaks and stops. Therefore, the final value of $R2$ is **25.**

Question

Consider 5 instructions are shown below –

Instruction	Size in words
i	2
i+1	1
i+2	2
i+3	3
i+4	2

The program is loaded from address **5000** and the memory is **word – addressable.** What are the possible PC values after the execution of $i + 3$ instruction?

Answer

This is actually an **MSQ** since we can have branch statements to jump between instructions. Hence, there can be a few possible ways to finish $i + 3$ –

- $i \rightarrow i+1 \rightarrow i+2 \rightarrow i+3$
- $i \rightarrow i+3$
- $i \rightarrow i+1 \rightarrow i+3$
- $i+3$

Using the above sequence, we can see that the possible PC values can be – **5008, 5002, 5003, 5000.**

Question

Which of the following statements is/are not wrong?

a) 2 micro-operations can be performed simultaneously
b) Only 1 micro-operation is performed at a time always
c) Memory read and Memory write both can be performed simultaneously
d) None

Answer

a. 2 micro-operations **CAN** be performed simultaneously. This is represented by the comma operator.
b. Since a) is correct, b) is **WRONG.**
c. In Memory read and write, the direction of data transfer is opposite. So, if we perform them simultaneously, there will be collisions. Hence, write and read is **NOT** performed simultaneously.

Question

Suppose, a system has 22 bit instructions and there are 100 instructions. How many memory blocks will this require?

Answer

Assuming byte addressable, each block will have 1 byte or 8 bits. Hence, each instruction will require 3 blocks (8 + 8 + 6). Therefore, 100 instructions will take **300 blocks.**

Question

Assume a system has a total of 32 instructions each of 24 – bits size. There are 32 GPRs and an instruction consists of an op-code, 2 register addresses and immediate operand. If the immediate operand is an unsigned integer, what is the minimum and maximum value it can take?

Answer

Since we have 32 instructions, the op-code will be of **5 – bits size.** At the same time, we also have 32 GPRs. Hence, the size of the register address will be of **5 – bits size.** There, we have –

$$24b = 5b + 5b + 5b + Immediate\ operand\ size$$

This gives us that the size of the immediate operand is **9 – bits.** It is given that the immediate operand is an unsigned integer. Hence, the value can vary from 000000000 to 111111111. Therefore, the value varies from **0 to 511**. We can do a similar analysis for signed integers as well –

SIGNED REPRESENTATION	MIN VALUE	MAX VALUE
Signed magnitude	-255	+255
1's complement	-255	+255
2's complement	-256	+255

Question

There is system which uses 20 bits instructions and 8-bits addresses. It supports 2-address and 1-address instructions both. Suppose there are x 2-address instructions, and based on that maximum & minimum 1-address instructions can be *a*, *b* respectively. Then the maximum value of *a* – *b* is?

Answer

The minimum number of 1 – address instructions has to be **1.** It can't be 0 since we know that 1-address instructions are supported. At the same time, to get the maximum number of 1 – address instructions, we need to have the minimum number of 2 – address instructions i.e. 1.

Therefore,

$$Max\ no\ of\ 1 - add\ ins. = (2^{20-16} - 1) * 2^8 = 3840$$

Therefore,

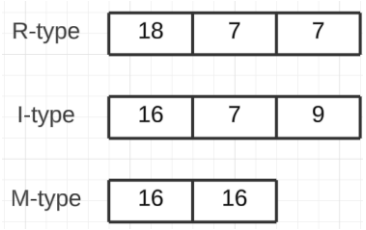
$$a - b = 3840 - 1 = \mathbf{3839}$$

Question

A CPU has 128 registers, 64KB byte addressable memory and 4Bytes instructions. The CPU supports 3 types of instructions: R-type, I-type and M-type. Each R-type instruction contains an opcode and 2 register names. Each I-type Instruction contains an opcode, a register name and a 9-bit immediate value. Each M-type instruction contains an opcode and a memory address. If there are 3072 R-type instructions, 252 I-type instructions then maximum how many M-type instructions the CPU can support?

Answer

From the question, we can see that instruction size is 32bits. Using the information, we can create the instruction format for all three types –



Max number of I – type instructions is $2^{16} = 65536$. It is given that there are 252 I – type instructions. Hence, we get –

$$Max\ no\ of\ M - type = (65536 - 252) * 2^0 = 65284$$

Now, let us assume that there are x number of M – type instructions. Then, we get –

$$No\ of\ R - type = (65284 - x) * 2^2 = 3072$$

Finally, we get

$$\boldsymbol{x = 64516}$$

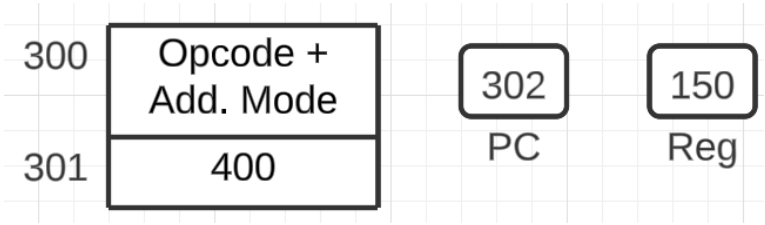
Question

An instruction is stored at Location 300 with its address field at location 301. The address field has the value 400. A processor register contains the number 150. Evaluate the effective address, if addressing mode is:

1. Direct
2. Immediate
3. Relative
4. Register Indirect

Answer

The memory organization looks as shown below –



Now, we get a clear picture on how to answer the question –

1. **Direct –** 400
2. **Immediate –** 301
3. **Relative –** 302 (PC) + 400 (offset) = 702
4. **Register Indirect –** 150 (effective address)

Question

A relative branch mode type instruction is stored in memory at address 300. The branch is made to an address 450.

1. What should be the value of relative address field of the instruction?
2. Determine the value of PC before instruction fetch, after the fetch and after execution phase?

Answer

In this case, we have –

- PC value = 301
- Target/Effective Address = 450

We know that for PC relative mode,

$$Target\ Address = PC + Offset$$

Hence, offset will be **149** which will be stored in the address field of the instruction. This is how the PC value will change –

- Before instruction fetch – 300
- After instruction fetch/During decode – 301
- After execution – 450

Question

Consider a three word machine instruction

```
ADD A[R0], @ B
```

The first operand (destination) "A [R0]" uses indexed addressing mode with R0 as the index register. The second operand (source) "@ B" uses indirect addressing mode. A and B are memory addresses residing at the second and the third words, respectively. The first word of the instruction specifies the opcode, the index register designation and the source and destination addressing modes. During execution of ADD instruction, the two operands are added and stored in the destination (first operand).

The number of memory cycles needed during the execution cycle of the instruction is

Answer

We have the following steps to perform in this operation –

- Find the operand A
- Find the operand B
- Add them
- Store the result back in memory

In the case of A, we are using Base indexed addressing mode. It is also mentioned that in the instruction itself they have mentioned the base and the offset(index) information. Hence, we can simply calculate the address and then have **1 memory access** to get the operand value.

As for B, we are using Indirect addressing. This means that we first access the memory location 1 pointed to by the address value in the instruction. Then we access the memory location 2 pointed to by the value in memory location 1. Hence, we have **2 memory access** to get the operand value.

The addition doesn't need any memory access but we do require **1 memory access** to store the data back into the memory.

Therefore, total memory cycles needed is **4 cycles.**

Question

Consider a 6-words instruction, which is of the following type:

Opcode	Mode1	Mode2	Address1	Address2

The first operand (destination) uses register indirect mode and second operand uses indirect mode. Assume each operand is of size 2 words, each address is of 2 words and main memory takes 50ns for 1 byte access. Further assume that the opcode denotes addition operation which copies result of addition of 2 operands.

Find the time required in the fetch cycle, execution cycle and the total instruction cycle. Assume two conditions –

- 1 word = 1B
- 1 word = 4B

Answer

1 word = 1B

During instruction fetch, we need to get the 6 word instruction. Hence, we need to access 6B data from memory which will take **300ns.**

Next, we start the execution cycle. The decode doesn't need any memory access. After decode, we need to perform operand fetch. We see that the first operand needs to be accessed using register indirect mode. In this case, we need to access the memory pointed to by the register value. Hence, there is a single memory access itself. Since the operand is of 2 words or 2B, the time taken will be **100ns.**

For the second operand, we perform indirect accessing. Hence, we first access a memory location and then we get the operand value at the next memory access. Therefore, we access a total of **4 words or 4B** which will take **200ns** of time.

Next, we are performing addition as per the instruction execution. That doesn't take any memory access. Finally, we need to write back the results. From the question, we can see that the destination is the first operand itself. Hence, we need to write back **2B** into the first operand. This will take another **100ns.**

Hence, we have –

$$Instruction\ Fetch = 300ns$$

$$Execution\ cycle = 400ns$$

$$Total\ instruction\ cycle = 700ns$$

1 word = 4B

We follow the same process for this case as well, just change 1 word to 4B. This gives us –

$$Instruction\ Fetch = 1200ns$$

$$Execution\ cycle = 1600ns$$

$$Total\ instruction\ cycle = 2800ns$$

Question

A hardwired CPU uses 10 control signals S1 to S10, in various time steps T1 to T5, to implement 4 instructions I1 to I4 as shown below:

	T1	T2	T3	T4	T5
I1	S1, S3, S5	S2, S4, S6	S1, S7	S10	S3, S8
I2	S1, S3, S5	S8, S9, S10	S5, S6, S7	S6	S10
I3	S1, S3, S5	S7, S8, S10	S2, S6, S9	S10	S1, S3
I4	S1, S3, S5	S2, S6, S7	S5, S10	S6, S9	S10

Which of the following pairs of expressions represent the circuit for generating control signals S5 and S10 respectively?

Answer

$$S5 = T1 + T3 * (I2 + I4)$$

$$S10 = T2(I2 + I3) + (T3 * I4) + T4(I1 + I3) + T5(I2 + I4)$$

Question

Consider a CPU which supports 64 distinct instructions. To execute a single instruction, there is a need to perform 16 micro – operations in sequence. The control memory of the micro – programmed CU has the following micro-instruction format –

- Control signals (120 bits)
- Mux Select (3 bits)
- Next address.

What is the size of the control memory?

Answer

As we know, in the control memory each address houses 1 micro – instruction. We know –

$$No\ of\ micro - instructions = 64 * 16 = 2^{10}$$

Hence, the control memory will have an address of size 10 bits. Therefore,

$$Size\ of\ \mathbf{1\ row}\ of\ control\ memory = 120 + 3 + 10 = \mathbf{133\ bits}$$

Hence, size of control memory will be –

$$Size = 2^{10} * 133b = \mathbf{133Kb}$$

Question

Design of a vertical microprogrammed control unit requires to generate 40 signals. Out of first 34 those only 3 signals can be active at a time. And for remaining 6, anyone can be active anytime. The microinstruction of the control unit stores control signal information along with 3-bit mux select and 10-bits address field. The size of control memory required is?

Answer

As we can see, the first 34 signals can have 3 signals active at a time. Hence, we can create 3 groups of 34 signals each where in each group we will have 1 signal active. For the rest of the 6 bits, any of them can be active. So we will need 6 bits for that. Hence, the total becomes –

$$Total\ bits = (3 * 6) + 6 + 3 + 10 = \mathbf{37b}$$

Since we have 10 bit addresses, there are a total of 2^{10} memory slots. Hence,

$$Memory\ size = \ 2^{10} * 37 = \mathbf{37Kb}$$

Question

Let us assume the system interrupt overhead time is $1\mu s$. Now, the performance gain of the interrupt based I/O as compared to Programmed based I/O is 5. Consider the data transfer rate and the interrupt service time as negligible. What is the I/O speed?

Answer

We know,

$$Performance\ Gain = \frac{Time\ for\ programmed\ I/O}{Time\ for\ interrupt\ I/O} = 5$$

Now, we also know –

$$Interrupt\ time = Interrupt\ Overhead + ISR = 1\mu s + 0 = \mathbf{1\mu s}$$

Therefore,

$$Time\ for\ programmed\ IO = 5\mu s$$

We also know that,

$$Time\ for\ programmed\ IO = Time\ to\ check\ status\ word + Data\ transfer\ time$$
$$= Time\ to\ check\ status\ word$$

As it is not explicitly mentioned, we assume that the status word is of size **1B.** Hence, we can see that –

$$Time\ to\ check\ 1B = 5\mu s$$

Hence, the I/O speed becomes $\mathbf{200KBps}$

Question

The DMA controller has data count register of size 8-bits. The memory is byte addressable. The maximum number of bytes the DMA can transfer to memory at a time without giving the control of the buses back to CPU?

Answer

The data count register is 8 – bits long. Hence, the maximum value it can store is **255.** Thus, it can count up to a max value of 255. This means, it can transfer **255 bytes.**

Question

Which of the following is/are true?

1. Data format used in IO devices may differ from CPUs format
2. IO devices are slower than CPU
3. IO devices are slower than main memory

Answer

All 3 statements are correct.

Question

Which of the following functionalities are provided by IO processor?

1. IO interfacing
2. DMA controller
3. IO Instruction execution

Answer

All 3 statements are correct.

Question

Which of the following is/are true?

a) Any IO operation can be performed always after current instruction execution in CPU
b) DMA controller can start data transfer immediately after getting DMA request from IO
c) CPU generates memory read/write signal for DMA transfer
d) CPU generates Interrupt acknowledgement immediately after receiving the interrupt always.

Answer

None of the above statements are true.

Question

Which of the following is/are not false?

a) DMA controller is a special purpose processor
b) CPU can perform some operation during DMA transfer
c) CPU can enable DMA hold acknowledgement after instruction fetch cycle
d) DMA transfer stops when starting address becomes zero

Answer

Options a), b) and c) are **TRUE.**

Question

Statement 1: Static RAM memory devices retain data for as long as power is supplied

Statement 2: Static RAM is used when the size of read/write memory required is large

Answer

As per the properties of the RAM, the static RAM will hold the value as long as the power is supplied. However, SRAMs are costly (but fast). Hence, they are not used in large memory designs and therefore the second statement is **FALSE.**

Question

The memory access time is 1 nanosecond for a read operation with a hit in cache, 5 nanoseconds for a read operation with a miss in cache, 2 nanoseconds for a write operation with a hit in cache and 10 nanoseconds for a write operation with a miss in cache. Execution of a sequence of instructions involves 100 instruction fetch operations, 60 memory operand read operations and 40 memory operand write operations. The cache hit-ratio is 0.9. The average memory access time (in nanoseconds) in executing the sequence of instructions is?

Answer

From cache equations, we have –

$$\left(T_{avg}\right)_{read} = (0.9 * 1) + (0.1 * 5) = 1.4ns$$

$$\left(T_{avg}\right)_{write} = (0.9 * 2) + (0.1 * 10) = 2.8ns$$

As per the question,

$$Number\ of\ Read\ ops = 100 + 60 = 160$$

$$Number\ of\ Write\ ops = 40$$

Hence, we have 80% of read ops and 20% of write ops. Therefore, we get –

$$T_{avg} = (0.8 * 1.4) + (0.2 * 2.8) = \mathbf{1.68\ ns}$$

Question

Assume there is a Direct mapped cache of size 256B and has a block size of 16B. If the memory address is of 13 – bits, then find the size of the tag field.

Answer

We know that –

$$Memory\ address = Tag\ bits + Cache\ mem\ bits + Bit\ offset$$

Since the block size of the cache is 16B, the Bit Offset will be **4 – bits.** At the same time,

$$Cache\ mem\ bits = \log_2(No\ of\ cache\ blocks) = \log_2\left(\frac{256}{16}\right) = \mathbf{4 - bits}$$

Hence, we get –

$$Tag\ bits = 13 - 4 - 4 = \mathbf{5 - bits}$$

Question

Consider a direct mapped cache of size 32KB. The CPU generates 32 bit addresses. The number of tag bits in main memory address are?

Answer

Let us assume that the byte offset be x bits. Hence, we get –

$$Cache\ mem\ bits = \log_2\left(\frac{32KB}{2^x}\right) = 15 - x$$

Hence, we can write the following equation –

$$32 = Tag\ bits + (15 - x) + x$$

$$\mathbf{Tag\ bits = 17b}$$

Question

Assume,

- Block size = 16B
- Cache size = 128KB

- Size of main memory address = 34-bits
- 1 valid and 1 modified bit present
- Direct mapped cache

Find the following –

- Byte offset
- Bits in cache block
- Bit in Tag
- Tag Directory size

Answer

$$Byte\ offset = \log_2 16 = \mathbf{4b}$$

$$Cache\ block\ bits = \log_2\left(\frac{128K}{16}\right) = \mathbf{13b}$$

$$Bits\ in\ Tag = 34 - 13 - 4 = \mathbf{17b}$$

$$Tag\ directory\ size = 2^{13} * (17 + 2) = \mathbf{152Kb} = \mathbf{19KB}$$

Question

The width of the physical address on a machine is 36 bits. The width of the tag field in a 256 KB 8-way set associative cache is _____ bits ?

Answer

Let us first calculate the tag bits if the memory was a direct cache. Suppose the block offset was x bits. Then,

$$Block\ offset = x$$

$$Cache\ mem\ size = \log_2\left(\frac{2^{18}}{2^x}\right) = 18 - x$$

Hence,

$$(Tag\ bits)_{direct} = 36 - (18 - x) - x = \mathbf{18}$$

Now, we know for a k-associate cache memory,

$$(Tag\ bits)_{k-ass} = (Tag\ bits)_{direct} + \log_2 k$$

Hence,

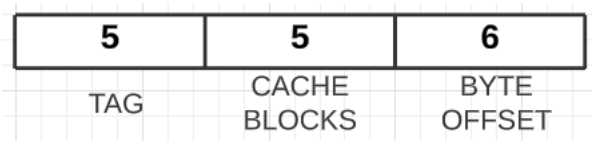
$$(Tag\ bits)_{8-ass} = 18 + \log_2 8 = \mathbf{21\ bits}$$

Question

Consider a machine with a byte addressable main memory of 2^{16} bytes. Assume that a direct mapped data cache consisting of 32 lines of 64 bytes each is used in the system. A 50 × 50 two-dimensional array of bytes is stored in the main memory starting from memory location 1100H. Assume that the data cache is initially empty. The complete array is accessed twice. Assume that the contents of the data cache do not change in between the two accesses.
How many data cache misses will occur in total?

Answer

From the given information, we can find that the cache has the following address split –



Now, we know that the array address starts at address 1100H.

$$(1100)_{16} = (0001000100000000)_2$$

Using the above address, we can see that the address 1100H is present in **Block 68 of main memory** (tag + cache block) and is present in **Block 4 of cache memory.** Each memory block can store 64 elements. As there are 2500 elements in the array, we will have to map **40 blocks** in total. Initially, the cache is empty and every element will have a **Cold Miss.**

0	96
1	97
2	98
3	99
4	**~~68,~~ 100**
5	**~~69~~, 101**
6	**~~70~~, 102**
7	**~~71~~, 103**
8	**~~72~~, 104**
9	**~~73~~, 105**
10	**~~74~~, 106**
11	**~~75~~, 107**
12	**~~76~~, 108**
13	77
14	78
15	79

16	80
17	81
18	82
19	83
20	84
21	85
22	86
23	87
24	88
25	89
26	90
27	91
28	92
29	93
30	94
31	95

Hence, in the 1st iteration we have a total of **40 cold misses.** Now, we perform the 2nd iteration. For the 2nd iteration, the blocks that will have a miss will be 68 – 76 and 100 – 108. Therefore, total number of misses will be 40+8+8 = **56 misses.**

Question

An access sequence of cache block address is of length N and contains n unique block addresses. The number of unique block addresses between two consecutive accesses to the same block address is bounded above by k. What is the miss ration is the access sequence is passed through a cache of associativity A >= k exercising least-recently used replacement policy.

(A) n/N

(B) 1/N

(C) 1/A

(D) k/n

Answer

The question states that there are N total memory accesses. Out of these, n are unique accesses. Hence, there will be $\boldsymbol{n}$ **cold misses.** Now, we have $N - n$ repeated accesses as well. Here, is where k comes into play.

The question also states that between 2 repeated accesses, the number of unique accesses will be **strictly lesser than $\boldsymbol{k}$.** This means that the for a cache with associativity $A \geq k$, the unique elements

between 2 repeated accesses can be stored in one single set. Hence, for **repeated accesses, there will be no replacement.**

Therefore,

$$Miss\ ratio = \frac{Total\ misses}{Number\ of\ accesses} = \frac{n}{N}$$

Question

Consider a disk with 16 platters, 2 surfaces per platter, 2K tracks per surface, 4K sectors per track and 4096 Bytes per sector. Disk rotates with 6000 rpm. Seek time is 5ms.
(a) Find capacity of disk
(b) Number of bits required for addressing the disk?
(c) Find disk access time?
(d) Find the disk transfer rate?

Answer

$$Capacity\ of\ disk = 16 * 2 * 2k * 4k * 4096 = 2^{40}B = \mathbf{1TB}$$

Since the sector is the smallest addressable element, we need to basically find the number of bits in the sector address.

$$No\ of\ sectors = 16 * 2 * 2k * 4k = 2^{28}$$

Hence, the disk requires **28 – bits** for addressing.

$$Disk\ access\ time = Seek\ time + Rotational\ Latency + Transfer\ time$$

Since the rotational latency is not given, we calculate and use the average rotational latency which is the time taken to rotate by **half the track.** We know that the disk spins at **6000 rpm.** Hence, the disk takes **10ms** to perform 1 revolution. Also, transfer time is the time taken to traverse a single sector. Hence, we get –

$$Disk\ access\ time = 5ms + \frac{10ms}{2} + \frac{10ms}{4K} = \mathbf{10.0025\ ms}$$

Finally, the disk transfer rate is the data the disk is able to send in 1 second. In one track, we have **16MB** of data and it takes 10ms to get that data. Hence, the disk can send/transfer the data at a rate of **1.6GBps**

Question

Consider a disk with 16 platters, 2 surfaces per platter, 1K tracks per surface, 2K sectors per track and 2048 Bytes per sector. Disk rotates with 3000 rpm. Seek time is 10ms.
If the disk is used in cycle stealing mode of DMA, such that whenever 64-bits word is available, it will be transferred in 16ns. What is the % of time CPU is blocked?

Answer

Since the disk rotates at 3000rpm, 1 revolution will take **20ms.** In other words, **4MB** of data (one full track) will be parsed in 20ms. Hence, 64-bits or **8B** of data will be parsed in –

$$Time\ to\ parse\ 8B\ of data = \frac{20 * 10^{-3} * 8}{4 * 10^{6}} = \mathbf{40}\boldsymbol{ns}$$

Using DMA, we can send the same 8B of data in **16ns** of time. Hence,

$$\%\ of\ time\ CPU\ is\ blocked = \frac{t_y}{t_x} = \frac{16}{40} = \mathbf{40}\%$$

Question

Consider a CPU which supports fixed length instructions. For this CPU a compiler generates 175 instructions for a user program. All the instructions are stored in the memory in byte aligned fashion, occupying a total of 525 bytes space in memory. The minimum possible length of one instruction for the CPU is _____ bits?

Answer

If 175 instructions are stored in 525 bytes of space, then each instruction occupies **3 bytes of space.** Now, they have asked the minimum possible length of the instruction. That will be when the instruction occupies the first 2 bytes completely but occupies just 1 bit in the 3rd byte space. Hence,

$$Min\ size\ of\ instruction = 8b + 8b + 1b = \mathbf{17}\ \boldsymbol{bits}$$

Question

Consider a program which contains 50 instructions I1, I2 , I3 ….. I50. Further consider a 5-stage pipeline with stages as: Instruction Fetch, Decode, Operand Fetch, Execution and Write-Back. The program contains only 1 branch instruction which is instruction I5 and its target is instruction I48. If during the execution of the program the branch is taken then number of cycles required to execute this program in the given pipeline is _____ ?

Answer

The instruction sequence will be as follows –

$$I1 \rightarrow I2 \rightarrow I3 \rightarrow I4 \rightarrow I5 \rightarrow I48 \rightarrow I49 \rightarrow I50$$

Hence, we have a total of **8 instructions.** For these instructions, we would require **12 cycles** ideally. However, we have 1 branch instruction whose target instruction will be found in the execution stage (4th stage). Hence, we will have an additional **3 stall cycles.** Therefore, the execution takes a total of **15 cycles.**

Question

Which of the following is/are true of the auto-increment addressing mode?

I. It is useful in creating self-relocating code
II. If it is included in an Instruction Set Architecture, then an additional ALU is required for effective address calculation
III. The amount of increment depends on the size of the data item accessed

A. I only B. II only C. III only D. II and III only

Answer

In this case, **Statement III** is obviously correct since the increment value will be the same as the word size of the memory. That means, we can either have Option C or Option D. Now, as per **Statement II,** it states that the auto – increment requires an additional ALU. Actually, the auto – increment requires an addition operation **but no need for a whole new ALU.** The increment will be accomplished using the ALU already present. Hence, **Statement II is FALSE.** Therefore, **Option C** is the correct answer.

Question

Consider the C struct defined below:

```
struct data {
    int marks [100];
    char grade;
    int cnumber;
};
struct data student;
```

The base address of student is available in register $R1$. The field student.grade can be accessed efficiently using:

A. Post-increment addressing mode, $(R1)+$
B. Pre-decrement addressing mode, $-(R1)$
C. Register direct addressing mode, $R1$
D. Index addressing mode, $X(R1)$, where X is an offset represented in $2's$ complement $16 - bit$ representation

Answer

Since we are using structure, we are creating a new data type as seen from our **Programming and DS** lectures. Since **student** is a variable of the type struct data, this means that the marks, grade and cnumber will be stored consecutively. Hence,

$$student.grade = Base\ address\ of\ R1 + (100 * size\ of\ int)$$

Basically, we are addressing by taking the base address of R1 and then adding an offset to it. **Hence, Option D is the most efficient way to access.**

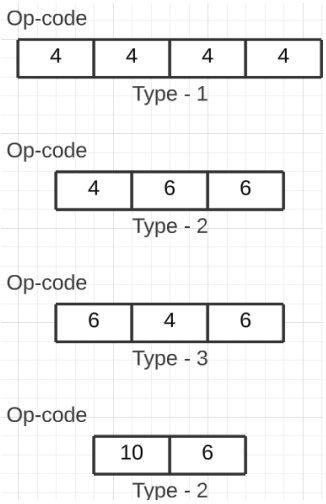
Question

A processor has 16 integer registers $(R0, R1, \ldots, R15)$ and 64 floating point registers $(F0, F1, \ldots, F63)$. It uses a $2 - byte$ instruction format. There are four categories of instructions: $Type - 1, Type - 2, Type - 3,$ and $Type - 4$. $Type - 1$ category consists of four instructions, each with 3 integer register operands $(3Rs)$. $Type - 2$ category consists of eight instructions, each with 2 floating point register operands $(2Fs)$. $Type - 3$ category consists of fourteen instructions, each with one integer register operand and one floating point register operand $(1R + 1F)$. $Type - 4$ category consists of N instructions, each with a floating point register operand $(1F)$.

The maximum value of N is _____

Answer

From the question, we can find the instruction formats as follows –



Total number of Type – 1 instructions possible = Total number of Type – 2 instructions possible = **16.** As per the question, there are 4 Type – 1 and 8 Type – 2 instructions. Hence, there are still **4 combinations** for the 1st 4 bits possible.

Thus, total number of Type – 3 instructions = $4 * 2^2 = \mathbf{16.}$ As per the question, there are 14 Type – 3 instructions. Hence, there are still **2 combinations** for the 1st 6 bits possible. Thus, total number of Type – 4 instructions possible = $2 * 2^4 = \mathbf{32}$ which is the answer!

Question

The main difference(s) between a CISC and a RISC processor is/are that a RISC processor typically

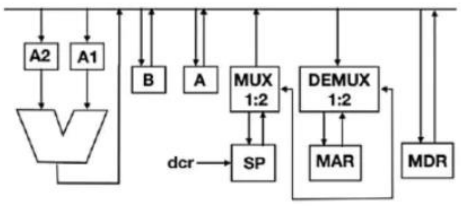
A. has fewer instructions
B. has fewer addressing modes
C. has more registers
D. is easier to implement using hard-wired logic

Answer

All the options are correct.

Question

Consider the following data path of a simple non-pipelined CPU. The registers A, B, A_1, A_2, MDR, the bus and the ALU are 8-*bit* wide. SP and MAR are 16-*bit* registers. The MUX is of size $8 \times (2:1)$ and the DEMUX is of size $8 \times (1:2)$. Each memory operation takes 2 CPU clock cycles and uses MAR (Memory Address Register) and MDR (Memory Date Register). SP can be decremented locally.



The CPU instruction "push r" where, $r = A$ or B has the specification

- $M[SP] \leftarrow r$
- $SP \leftarrow SP - 1$

How many CPU clock cycles are required to execute the "push r" instruction?

A. 2 B. 3 C. 4 D. 5

Answer

It is given that r can either be A or B. For our sake, let us assume $r = A$. The whole transfer will be done in the following fashion –

$$MDR \leftarrow A$$

$$MAR \leftarrow SP$$

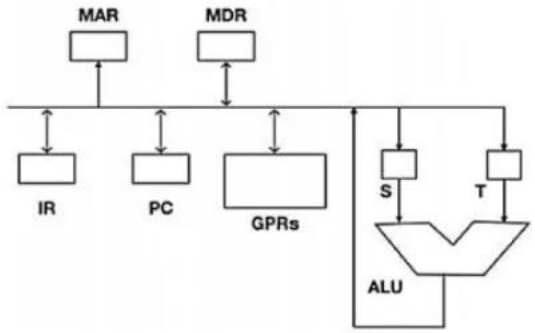
$$M[MAR] \leftarrow A \quad ; \quad SP \leftarrow SP - 1$$

In the question, we know that A and MDR are 8 – bits long while MAR and SP are 16 – bits long. Since the bus is 8 – bits long, **it takes 1 clock cycle to transfer 8 – bits of data and 2 clock cycles to transfer 16 – bits of data.** Also, the decrement of SP happens locally and therefore doesn't require any additional cycles. Therefore,

$$Total\ number\ of\ cycles = 1 + 2 + 2 = \mathbf{5\ cycles}$$

Question

Consider the following data path of a CPU.



The ALU, the bus and all the registers in the data path are of identical size. All operations including incrementation of the PC and the GPRs are to be carried out in the ALU. Two clock cycles are needed for memory read operation – the first one for loading address in the MAR and the next one for loading data from the memory bus into the MDR.

The instruction "add R0, R1" has the register transfer interpretation R0 <= R0 + R1. The minimum number of clock cycles needed for execution cycle of this instruction is:

A. 2 B. 3 C. 4 D. 5

Answer

In this case, we have the following process to be followed –

- First, we need to perform instruction fetch
  - Send PC to MAR
  - Get the memory value at MAR and send it to MDR

    - Send MDR to IR
- Next, we need to increment PC
    - Send PC to ALU input
    - Add and send the output back to PC
- After that, we perform the addition operation
    - Send R0 to 1st input of ALU
    - Send R1 to the 2nd input of ALU
    - Add and send the result back to R0

Each of the sub – step takes 1 clock cycle. So, if we include the instruction fetch as well, then we get a total of **8 cycles.** However, the last line of the question states that we only need to look at the Execution cycle…which means we don't need to worry about instruction fetch and PC incrementation. Hence, the answer here will be **3 cycles.**

Question

Consider a CPU where all the instructions require 7 clock cycles to complete execution. There are 140 instructions in the instruction set. It is found that 125 control signals are needed to be generated by the control unit. While designing the horizontal microprogrammed control unit, single address field format is used for branch control logic. What is the minimum size of the control word and control address register?

A. $125, 7$ B. $125, 10$ C. $135, 9$ D. $135, 10$

Answer

There are 140 instructions and given that each instruction requires 7 micro – instructions (cycles) to execute. Hence,

$$Size\ of\ address = ceil(\log_2(140 * 7)) = \mathbf{10\ bits}$$

Now, since this is a horizontal control unit, we will have 125 bits for all the control signals. At the same time, the control unit uses single address field format aka it has the control signal bits and an address for branching instruction. Hence,

$$Size\ of\ control\ unit = 125 + 10 = \mathbf{135\ bits}$$

Hence, **Option D** is the answer.

Question

The following code segment is executed on a processor which allows only register operands in its instructions. Each instruction can have atmost two source operands and one destination operand. Assume that all variables are dead after this code segment.

```
c = a + b;
d = c * a;
e = c + a;
x = c * c;
if (x > a) {
   y = a * a;
}
else {
  d = d * d;
  e = e * e;
}
```

Suppose the instruction set architecture of the processor has only two registers. The only allowed compiler optimization is code motion, which moves statements from one place to another while preserving correctness. What is the minimum number of spills to memory in the compiled code?

Answer

First step is to optimize the above program and reduce the scope of the variables. This can be done as follows –

```
c = a + b;
x = c * c;
if (x > a) {
    y = a * a;
}
else {
    d = c * a;
    e = c + a;
    d = d * d;
    e = e * e;
}
```

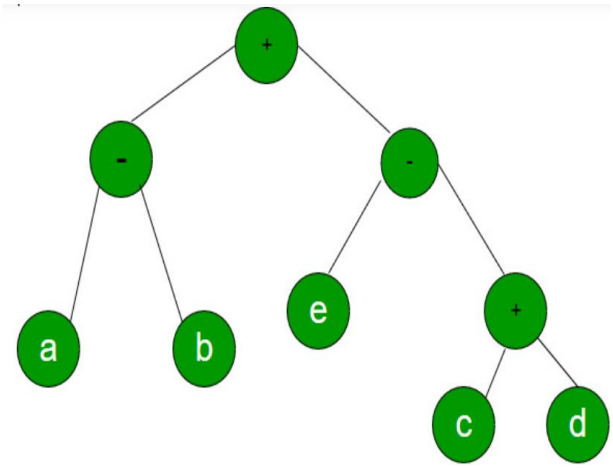
The assembly code for the if part will now become something like this –

```
MOV R1, a
MOV R2, b
ADD R2, R1, R2 //R2 has value of C
MOV Mem(A), R2 //Move c to a memory location A
MUL R2, R2, R2 //R2 has value of x

CMP R2, R1
JG if_part //Jump if greater than to if part

if_part :
MUL R1, R1, R1 //R1 has the value of y
```

Hence, in the best case scenario, the minimum number of register spills will be **1.**

Question

Consider evaluating the following expression tree on a machine with load-store architecture in which memory can be accessed only through load and store instructions. The variables a, b, c, d and e initially stored in memory. The binary operators used in this expression tree can be evaluate by the machine only when the operands are in registers. The instructions produce results only in a register. If no intermediate results can be stored in memory, what is the minimum number of registers needed to evaluate this expression?



Answer

R1←c, R2←d, R2←R1+R2, R1←e, R2←R1-R2
Now to calculate the rest of the expression we must load a and b into the registers but we need the
content of R2 later.
So we must use another Register.
R1←a, R3←b, R1←R1-R3, R1←R1+R2

Hence we need at least **3 registers.**

Question

Explain about Return From Exception instruction.

Answer

RFE (Return From Exception) is a privileged trap instruction that is executed when exception occurs, so an exception is not allowed to execute. In computer architecture for a general purpose processor, an exception can be defined as an abrupt transfer of control to the operating system. Exceptions are broadly classified into 3 main categories: a. Interrupt: it is mainly caused due to I/O device. b. Trap: It is caused by the program making a syscall. c. Fault: It is accidentally caused by the program that is under execution such as(a divide by zero, or null pointer exception etc). The processor's fetch instruction unit makes a poll for the interrupts. If it finds something unusual happening in the machine operation it inserts an interrupt pseudo-instruction in the pipeline in place of the normal instruction. Then going through the pipeline it starts handling the interrupts. The operating system explicitly makes a transition from kernel mode to user mode, generally at the end of an interrupt handle pr kernel call by using a privileged instruction RFE(Return From Exception) instruction.

Question

Which of the following are NOT true in a pipelined processor?

```
I.  Bypassing can handle all RAW hazards.
II. Register renaming can eliminate all register
    carried WAR hazards.
III. Control hazard penalties can be eliminated by
     dynamic branch prediction.
```

Answer

I - False, Bypassing can't handle all RAW hazard, consider when any instruction depends on the result of LOAD instruction, now LOAD updates register value at Memory Access Stage (MA), so data will not be available directly on Execute stage.

II - True, register renaming can eliminate all WAR Hazard.

III- False, It cannot completely eliminate, though it can reduce Control Hazard Penalties.

Question

The use of multiple register windows with overlap causes a reduction in the number of memory accesses for

```
I. Function locals and parameters
II. Register saves and restores
III. Instruction fetches
```

Answer

I is true as by using multiple register windows, we eliminate the need to access the variable values again and again from the memory. Rather, we store them in the registers.

II is false as register saves and restores would still be required for each and every variable.

III is also false as instruction fetch is not affected by memory access using multiple register windows.

QUESTION BANK

Question 1

Q Consider a computer system with a byte-addressable primary memory of size 2^{32} bytes. Assume the computer system has a direct-mapped cache of size 32 KB (1 KB = 2^{10} bytes), and each cache block is of size 64 bytes. The size of the tag field is _____ bits. **(GATE 2021)**

(a) 17 **(b)** 18 **(c)** 15 **(d)** 9

Question 2

Q Consider a machine with byte addressable memory of 2^{32} bytes divided into blocks of size 32 bytes. Assume a direct mapped cache having 512 cache lines is used with this machine. The size of tag field in bits is _____ **(Gate-2017) (2 Marks)**

(A) 12 **(B)** 16 **(C)** 18 **(D)** 24

Question 3

Q Consider a direct mapped cache of size 32 KB with block size 32 bytes. The CPU generates 32 bit addresses. The number of bits needed for cache indexing and the number of tag bits are respectively **(Gate-2005) (2 Marks)**

(A) 10, 17 **(B)** 10, 22 **(C)** 15, 17 **(D)** 5, 17

Question 4

Q Consider a machine with a byte addressable main memory of 2^{20} bytes, block size of 16 bytes and a direct mapped cache having 2^{12} cache lines. Let the addresses of two consecutive bytes in main memory be $(E201F)_{16}$ and $(E2020)_{16}$. What are the tag and cache line address (in hex) for main memory address $(E201F)_{16}$? **(Gate-2015) (1 Marks)**

(A) E, 201 **(B)** F, 201 **(C)** E, E20 **(D)** 2, 01F

Question 5

Q Consider the main memory size is of 128 KB, the cache size is of 16 KB, the block size is of 256 B, the set size is 2. Find Tag.

Question 6

Q Consider a set-associative cache of size 2KB (1KB=2^10 bytes) with cache block size of 64 bytes. Assume that the cache is byte-addressable and a 32 -bit address is used for accessing the cache. If the width of the tag field is 22 bits, the associativity of the cache is _____ . **(GATE 2021) (1 MARKS)**

Question 7

Q A computer system with a word length of 32 bits has a 16 MB byte- addressable main memory and a 64 KB, 4-way set associative cache memory with a block size of 256 bytes. Consider the following four physical addresses represented in hexadecimal notation. **(Gate-2020) (2 Marks)**

A_1 = 0x42C8A4, A_2 = 0x546888, A_3 = 0x6A289C, A_4 = 0x5E4880

(A) A_1 and A_4 are mapped to different cache sets.
(B) A_2 and A_3 are mapped to the same cache set.
(C) A_3 and A_4 are mapped to the same cache set.
(D) A_1 and A_3 are mapped to the same cache set.

Question 8

Q The size of the physical address space of a processor is 2^P bytes. The word length is 2^W bytes. The capacity of cache memory is 2^N bytes. The size of each cache block is 2^M words. For a K-way set-associative cache memory, the length (in number of bits) of the tag field is **(Gate-2018) (2 Marks)**

(A) $P - N - \log_2 K$
(B) $P - N + \log_2 K$
(C) $P - N - M - W - \log_2 K$
(D) $P - N - M - W + \log_2 K$

Question 9

Q A cache memory unit with capacity of N words and block size of B words is to be designed. If it is designed as direct mapped cache, the length of the TAG field is 10 bits. If the cache unit is now designed as a 16-way set-associative cache, the length of the TAG field is _____ bits. **(Gate-2017) (1 Marks)**

Question 10

Q The width of the physical address on a machine is 40 bits. The width of the tag field in a 512 KB 8-way set associative cache is _____ bits **(Gate-2016) (2 Marks)**

Question 11

Q A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. The number of bits for the TAG field is _____ **(Gate-2014) (1 Marks)**

Question 12

Q In a k-way set associative cache, the cache is divided into v sets, each of which consists of k lines. The lines of a set are placed in sequence one after another. The lines in set s are sequenced before the lines in set (s+1). The main memory blocks are numbered 0 onwards. The main memory block numbered j must be mapped to any one of the cache lines from. **(Gate-2013) (1 Marks)**

(A) (j mod v) * k to (j mod v) * k + (k-1)
(B) (j mod v) to (j mod v) + (k-1)
(C) (j mod k) to (j mod k) + (v-1)
(D) (j mod k) * v to (j mod k) * v + (v-1)

Question 13

Q A computer has a 256 KByte, 4-way set associative, write back data cache with block size of 32 Bytes. The processor sends 32-bit addresses to the cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. The number of bits in the tag field of an address is **(Gate-2012) (2 Marks)**

(A) 11 **(B)** 14 **(C)** 16 **(D)** 27

Question 14

Q An 8KB direct-mapped write-back cache is organized as multiple blocks, each of size 32-bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following.

1 Valid bit

1 Modified bit

As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache? **(Gate-2011) (2 Marks)**

(A) 4864 bits **(B)** 6144 bits **(C)** 6656 bits **(D)** 5376 bits

---

Question 15

Q Consider a computer with a 4-ways set-associative mapped cache of the following characteristics: a total of 1 MB of main memory, a word size of 1 byte, a block size of 128 words and a cache size of 8 KB. The number of bits in the TAG, SET and WORD fields, respectively are: **(Gate-2008) (2 Marks)**

(A) 7, 6, 7 **(B)** 8, 5, 7 **(C)** 8, 6, 6 **(D)** 9, 4, 7

Question 16

Q Consider a computer with a 4-ways set-associative mapped cache of the following characteristics: a total of 1 MB of main memory, a word size of 1 byte, a block size of 128 words and a cache size of 8 KB. While accessing the memory location 0C795H by the CPU, the contents of the TAG field of the corresponding cache line is: **(Gate-2008) (2 Marks)**

a) 000011000 **b)** 110001111 **c)** 00011000 **d)** 110010101

Question 17

Q Consider a 4-way set associative cache consisting of 128 lines with a line size of 64 words. The CPU generates a 20-bit address of a word in main memory. The number of bits in the TAG, LINE and WORD fields are respectively: **(Gate-2007) (1 Marks)**

(A) 9,6,5 **(B)** 7, 7, 6 **(C)** 7, 5, 8 **(D)** 9, 5, 6

Question 18

Q A computer system has a level-1 instruction cache (1-cache), a level-1 data cache (D-cache) and a level-2 cache (L2-cache) with the following specifications:
The length of the physical address of a word in the main memory is 30 bits. The capacity of the tag memory in the I-cache, D-cache and L2-cache is, respectively,
(Gate-2006) (2 Marks)
(A) 1 K x 18-bit, 1 K x 19-bit, 4 K x 16-bit
(B) 1 K x 16-bit, 1 K x 19-bit, 4 K x 18-bit
(C) 1 K x 16-bit, 512 x 18-bit, 1 K x 16-bit
(D) 1 K x 18-bit, 512 x 18-bit, 1 K x 18-bit

	Capacity	Mapping Method	Block size
I-cache	4K words	Direct mapping	4 Words
D-cache	4K words	2-way set associative mapping	4 Words
L2-cache	64K words	4-way set associative mapping	16 Words

Question 19

Q The main memory of a computer has 2 cm blocks while the cache has 2c blocks. If the cache uses the set associative mapping scheme with 2 blocks per set, then the block k of main memory maps to the set: **(Gate-1999) (1 Marks)**
(A) (k mod m)of the cache
(B) (k mod c)of the cache
(C) (k mod 2c)of the cache
(D) (k mod 2cm)of the cache

Question 20

Q Consider a small two-way set-associative cache memory, consisting of four blocks. For choosing the block to be replaced, use the least recently used (LRU) scheme. The number of cache misses for the following sequence of block addresses is 8, 12, 0, 12, 8 **(Gate-2004) (2 Marks)**
(A) 2 **(B)** 3 **(C)** 4 **(D)** 5

Question 21

Q Consider a Direct Mapped Cache with 8 cache blocks (numbered 0-7). If the memory block requests are in the following order 3, 5, 2, 8, 0, 63, 9,16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82,17, 24. Which of the following memory blocks will not be in the cache at the end of the sequence? **(GATE-2007) (2 Marks)**
(A) 3 **(B)** 18 **(C)** 20 **(D)** 30

3, 5, 2, 8, 0, 63, 9,16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82,17, 24

Question 22

Q Consider a 4-way set associative cache (initially empty) with total 16 cache blocks. The main memory consists of 256 blocks and the request for memory blocks is in the following order: 0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155. Which one of the following memory blocks will NOT be in cache if LRU replacement policy is used? **(Gate-2009) (2 Marks)**
(A) 3 **(B)** 8 **(C)** 129 **(D)** 216

Question 23

Q Consider a 2-way set associative cache with 256 blocks and uses LRU replacement, Initially the cache is empty. Conflict misses are those misses which occur due the contention of multiple blocks for the same cache set. Compulsory misses occur due to first time access to the block. The following sequence of accesses to memory blocks
(0, 128, 256, 128, 0, 128, 256, 128, 1, 129, 257, 129, 1, 129, 257, 129) is repeated 10 times. The number of conflict misses experienced by the cache is _____. **(Gate-2017) (2 Marks)**

Question 24

Q Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 5 nanoseconds on a cache hit. Suppose while running a program, it was observed that 80% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is_____. **(GATE-2015) (2 Marks)**

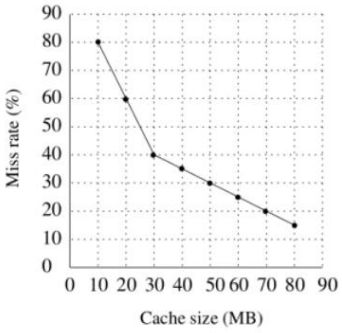
Question 25

Q Consider a system with 2 level cache. Access times of Level 1 cache, Level 2 cache and main memory are 0.5 ns, 5 ns and 100 ns respectively. The hit rates of Level 1 and Level 2 caches are 0.7 and 0.8 respectively. What is the average access time of the system ignoring the search time within the cache? **(NET-DEC-2018)**

a) 35.20 ns **b)** 7.55 ns **c)** 20.75 ns **d)** 24.35 ns

Question 26

Q A file system uses an in-memory cache to cache disk blocks. The miss rate of the cache is shown in the figure. The latency to read a block from the cache is 1 ms and to read a block from the disk is 10 ms. Assume that the cost of checking whether a block exists in the cache is negligible. Available cache sizes are in multiples of 10 MB.



The smallest cache size required to ensure an average read latency of less than 6 ms is _____ MB. **(Gate-2016) (2 Marks)**

Question 27

Q Consider a two-level cache hierarchy L_1 and L_2 caches. An application incurs 1.4 memory accesses per instruction on average. For this application, the miss rate of L_1 cache 0.1, the L_2 cache experience on average. 7 misses per 1000 instructions. The miss rate of L_2 expressed correct to two decimal places is _____. **(Gate-2017) (1 Marks)**

Question 28

Q The read access times and the hit ratios for different caches in a memory hierarchy are as given below:

Cache	Read access time (in nanoseconds)	Hit ratio
I-cache	2	0.8
D-cache	2	0.9
L2-cache	8	0.9

The read access time of main memory in 90 nanoseconds. Assume that the caches use the referred-word-first read policy and the writeback policy. Assume that all the caches are direct mapped caches. Assume that the dirty bit is always 0 for all the blocks in the caches. In execution of a program, 60% of memory reads are for instruction fetch and 40% are for memory operand fetch. The average read access time in nanoseconds (up to 2 decimal places) is _____ **(Gate-2017) (2 Marks)**

Question 29

Q The memory access time is 1 nanosecond for a read operation with a hit in cache, 5 nanoseconds for a read operation with a miss in cache, 2 nanoseconds for a write operation with a hit in cache and 10 nanoseconds for a write operation with a miss in cache. Execution of a sequence of instructions involves 100 instruction fetch operations, 60 memory operand read operations and 40 memory operands write operations. The cache hit-ratio is 0.9. The average memory access time (in nanoseconds) in executing the sequence of instructions is _____. **(Gate-2014) (2 Marks)**

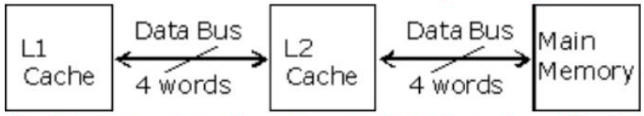
Question 30

Q Consider a system with 2 level caches. Access times of $Level_1$ cache, $Level_2$ cache and main memory are 1 ns, 10ns, and 500 ns, respectively. The hit rates of $Level_1$ and $Level_2$ caches are 0.8 and 0.9, respectively. What is the average access time of the system ignoring the search time within the cache? **(Gate-2004) (1 Marks)**

(A) 13.0 ns **(B)** 12.8 ns **(C)** 12.6 ns **(D)** 12.4 ns

Question 31

Q A computer system has an L_1 cache, an L_2 cache, and a main memory unit connected as shown below. The block size in L_1 cache is 4 words. The block size in L_2 cache is 16 words. The memory access times are 2 nanoseconds. 20 nanoseconds and 200 nanoseconds for L_1 cache, L_2 cache and main memory unit respectively.

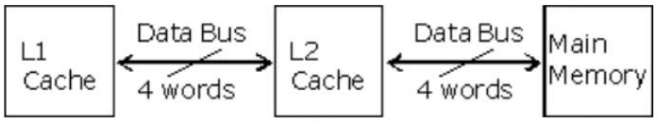


When there is a miss in L_1 cache and a hit in L_2 cache, a block is transferred from L_2 cache to L_1 cache. What is the time taken for this transfer? **(Gate-2010) (2 Marks)**

(A) 2 nanoseconds **(B)** 20 nanoseconds
(C) 22 nanoseconds **(D)** 88 nanoseconds

Question 32

Q A computer system has an L_1 cache, an L_2 cache, and a main memory unit connected as shown below. The block size in L_1 cache is 4 words. The block size in L_2 cache is 16 words. The memory access times are 2 nanoseconds. 20 nanoseconds and 200 nanoseconds for L_1 cache, L_2 cache and main memory unit respectively.



Q When there is a miss in both L_1 cache and L_2 cache, first a block is transferred from main memory to L_2 cache, and then a block is transferred from L_2 cache to L_1 cache. What is the total time taken for these transfers? **(Gate-2010) (2 Marks)**

(A) 222 nanoseconds
(B) 888 nanoseconds
(C) 902 nanoseconds
(D) 968 nanoseconds

Question 33

Q Let W_B and W_T be two set associative cache organizations that use LRU algorithm for cache block replacement. W_B is a write back cache and W_T is a write through cache. Which of the following statements is/are FALSE? **(GATE 2022) (1 MARKS)**

(A) Each cache block in W_B and W_T has a dirty bit.

(B) Every write hit in W_B leads to a data transfer from cache to main memory.

(C) Eviction of a block from W_T will not lead to data transfer from cache to main memory.

(D) A read miss in W_B will never lead to eviction of a dirty block from W_B.

Question 34

Q Consider a main memory system that consists of 8 memory modules attached to the system bus, which is one word wide. When a write request is made, the bus is occupied for 100 nanoseconds (ns) by the data, address, and control signals. During the same 100 ns, and for 500 ns thereafter, the addressed memory module executes one cycle accepting and storing the data. The (internal) operation of different memory modules may overlap in time, but only one request can be on the bus at any time. The maximum number of stores (of one word each) that can be initiated in 1 millisecond is _____ **(Gate-2014) (2 Marks)**

(A) 1,000 **(B)** 10,000 **(C)** 1,00,000 **(D)** 100

Question 35

Q A CPU generally handles an interrupt by executing an interrupt service routine:
(GATE-2009) (1 Marks)

a) As soon as an interrupt is raised.
b) By checking the interrupt register at the end of fetch cycle.
c) By checking the interrupt register after finishing the execution of the current instruction.
d) By checking the interrupt register at fixed time intervals.

Question 36

Q For the daisy chain scheme of connecting I/O devices, which of the following statements is true? **(GATE-1996) (1 Marks)**

a) It gives non-uniform priority to various devices
b) It gives uniform priority to all devices
c) It is only useful for connecting slow devices to a processor device
d) It requires a separate interrupt pin on the processor for each device

---

Question 37

Q The following are some events that occur after a device controller issues an interrupt while process L is under execution. **(GATE-2018) (2 Marks)**
(P) The processor pushes the process status of L onto the control stack.
(Q) The processor finishes the execution of the current instruction.
(R) The processor executes the interrupt service routine.
(S) The processor pops the process status of L from the control stack.
(T) The processor loads the new PC value based on the interrupt.

(A) QPTRS

(B) PTRSQ

(C) TRPQS

(D)QTPRS

Question 38

Q On a non-pipelined sequential processor, a program segment, which is a part of the interrupt service routine, is given to transfer 500 bytes from an I/O device to memory.

```
Initialize the address register
Initialize the count to 500
LOOP: Load a byte from device
      Store in memory at address given by address register
      Increment the address register
      Decrement the count
      If count != 0 go to LOOP
```

Assume that each statement in this program is equivalent to machine instruction which takes one clock cycle to execute if it is a non-load/store instruction. The load-store instructions take two clock cycles to execute. The designer of the system also has an alternate approach of using DMA controller to implement the same transfer. The DMA controller requires 20 clock cycles for initialization and other overheads. Each DMA transfer cycle takes two clock cycles to transfer one byte of data from the device to the memory. What is the approximate speedup when the DMA controller-based design is used in place of the interrupt driven program-based input-output? **(GATE-2011) (2 Marks)**
(A) 3.4 **(B)** 4.4 **(C)** 5.1 **(D)** 6.7

Question 39

Q Consider a disk pack with 16 surfaces, 128 tracks per surface and 256 sectors per track. 512 bytes of data are stored in a bit serial manner in a sector. The capacity of the disk pack and the number of bits required to specify a particular sector in the disk are respectively: **(GATE-2007) (1 Marks)**
(A) 256 Mbyte, 19 bits **(B)** 256 Mbyte, 28 bits
(C) 512 Mbyte, 20 bits **(D)** 64 Gbyte, 28 bit

Question 40

Q For a magnetic disk with concentric circular tracks, the seek latency is not linearly proportional to the seek distance due to **(GATE-2008) (2 Marks)**

(A) non-uniform distribution of requests
(B) arm starting and stopping inertia
(C) higher capacity of tracks on the periphery of the platter
(D) use of unfair arm scheduling policies

Question 41

Q Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of 50 × 10^6 bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is _____ **(GATE-2015) (2 Marks)**

Question 42

Q Consider a disk pack with a seek time of 4 milliseconds and rotational speed of 10000 rotations per minute (RPM). It has 600 sectors per track and each sector can store 512 bytes of data. Consider a file stored in the disk. The file contains 2000 sectors. Assume that every sector access necessitates a seek, and the average rotational latency for accessing each sector is half of the time for one complete rotation. The total time (in milliseconds) needed to read the entire file is _____. **(GATE-2015) (1 Marks)**

Question 43

Q A hard disk system has the following parameters: **(GATE-2007) (2 Marks)**
Number of tracks = 500
Number of sectors / track = 100
Number of bytes / sector = 500
Time taken by the head to move from one track to adjacent track = 1 ms, Rotation speed = 600 rpm. What is the average time taken for transferring 250 bytes from the disk?

(A) 300.5 ms **(B)** 255.5 ms **(C)** 255.0 ms **(D)** 300.0 ms

Question 44

Q A hard disk has 63 sectors per track, 10 platters each with 2 recording surfaces and 1000 cylinders. The address of a sector is given as a triple (c, h, s), where c is the cylinder number, h is the surface number and s is the sector number. Thus, the 0th sector is addressed as (0, 0, 0), the 1st sector as (0, 0, 1), and so on The address <400,16,29> corresponds to sector number: **(GATE-2009) (2 Marks)**

(A) 505035 **(B)** 505036 **(C)** 505037 **(D)** 505038

Question 45

Q Consider a hard disk with 16 recording surfaces (0-15) having 16384 cylinders (0-16383) and each cylinder contains 64 sectors (0-63). Data storage capacity in each sector is 512 bytes. Data are organized cylinder-wise and the addressing format is <cylinder no., surface no., sector no.>. A file of size 42797 KB is stored in the disk and the starting disk location of the file is <1200, 9, 40>. What is the cylinder number of the last sector of the file, if it is stored in a contiguous manner? **(GATE-2013) (1 Marks)**

(A) 1281 **(B)** 1282 **(C)** 1283 **(D)** 1284

Question 46

Q An application loads 100 libraries at start-up. Loading each library requires exactly one disk access. The seek time of the disk to a random location is given as 10 milli second. Rotational speed of disk is 6000 rpm. If all 100 libraries are loaded from random locations on the disk, how long does it take to load all libraries? (The time to transfer data from the disk block once the head has been positioned at the start of the block may be neglected) **(GATE-2011) (2 Marks)**

(A) 0.50 s **(B)** 1.50 s **(C)** 1.25 s **(D)** 1.00 s

Question 47

Q Consider a computer system with DMA support. The DMA module is transferring one 8-bit character in one CPU cycle from a device to memory through cycle stealing at regular intervals. Consider a 2 MHz processor. If 0.5% processor cycles are used for DMA, the data transfer rate of the device is _____ bits per second.**(GATE 2021) (1 MARKS)**

Question 48

Q The size of the data count register of a DMA controller is 16 bits. The processor needs to transfer a file of 29,154 kilobytes from disk to main memory. The memory is byte addressable. The minimum number of times the DMA controller needs to get the control of the system bus from the processor to transfer the file from the disk to main memory is _____ **(GATE-2016) (2 Marks)**

Question 49

Q A non-pipelined system takes 30ns to process a task. The same task can be processed in a four-segment pipeline with a clock cycle of 10ns . Determine the speed up of the pipeline for 100 tasks. **(NET 2019 DEC)**

(A) 3 **(B)** 4 **(C)** 3.91 **(D)** 2.91

Question 50

Q Consider a 4 stage pipeline processor. The number of cycles needed by the four instructions I_1, I_2, I_3, I_4 in stages S_1, S_2, S_3, S_4 is shown below: **(Gate-2009) (2 Marks)**

```
For (I = 1; I <= 2; i++)
{
        I1
        I2
        I3
        I4
}
```

	F	D	E	WB
I_1	2	1	1	1
I_2	1	3	2	2
I_3	2	1	1	3
I_4	1	2	2	2

a) 16 **b)** 23 **c)** 28 **d)** 30

Question 51

Q A 4-stage pipeline has the stage delays as 150, 120, 160 and 140 nanoseconds respectively. Registers that are used between the stages have a delay of 5 nanoseconds each. Assuming constant clocking rate, the total time taken to process 1000 data items on this pipeline will be **(Gate-2004) (2 Marks)**

(A) 120.4 microseconds

(C) 165.5 microseconds

(B) 160.5 microseconds

(D) 590.0 microseconds

Question 52

Q A non-pipelined single cycle processor operating at 100 MHz is converted into a synchronous pipelined processor with five stages requiring 2.5 nsec, 1.5 nsec, 2 nsec, 1.5 nsec and 2.5 nsec, respectively. The delay of the latches is 0.5 nsec. The speedup of the pipeline processor for a large number of instructions is **(Gate-2008) (2 Marks)**

(A) 4.5

(C) 3.33

(B) 4.0

(D) 3.0

Question 53

Q We have two designs D_1 and D_2 for a synchronous pipeline processor. D_1 has 5 pipeline stages with execution times of 3 nsec, 2 nsec, 4 nsec, 2 nsec and 3 nsec while the design D_2 has 8 pipeline stages each with 2 nsec execution time How much time can be saved using design D_2 over design D_1 for executing 100 instructions? **(Gate-2005) (2 Marks)**

(A) 214 nsec

(C) 86 nsec

(B) 202 nsec

(D) 200 nsec

Question 54

Q Instruction execution in a processor is divided into 5 stages. Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Execute (EX), and Write Back (WB), These stages take 5,4,20, 10 and 3 nanoseconds (ns) respectively.
A pipelined implementation of the processor requires buffering between each pair of consecutive stages with a delay of 2 ns.
Two pipelined implementations of the processor are contemplated:
(i) a naïve pipeline implementation (NP) with 5 stages and
(ii) an efficient pipeline (EP) where the OF stage id divided into stages OF_1 and OF_2 with execution times of 12 ns and 8 ns respectively.
The speedup (correct to two decimals places) achieved by EP over NP in executing 20 independent instructions with no hazards is _____. **(Gate-2017) (2-marks)**

Question 55

Q The stage delays in a 4-stage pipeline are 800, 500, 400 and 300 picoseconds. The first stage (with delay 800 picoseconds) is replaced with a functionally equivalent design involving two stages with respective delays 600 and 350 picoseconds. The throughput increase of the pipeline is _____ percent. **(Gate-2016) (2 Marks)**

Question 56

Q Consider a non-pipelined processor with a clock rate of 2.5 gigahertz and average cycles per instruction of four. The same processor is upgraded to a pipelined processor with five stages; but due to the internal pipeline delay, the clock speed is reduced to 2 gigahertz. Assume that there are no stalls in the pipeline. The speed up achieved in this pipelined processor is _____. **(Gate-2015)(2-marks)**

Question 57

Q Consider a 3 GHz (gigahertz) processor with a three-stage pipeline and stage latencies τ_1, τ_2 and τ_3 such that $\tau 1 = (3\ \tau_2)\ /4 = 2\tau_3$. If the longest pipeline stage is split into two pipeline stages of equal latency, the new frequency is _____ GHz, ignoring delays in the pipeline registers. **(Gate-2016) (2 Marks)**

Question 58

Q A five-stage pipeline has stage delays of 150,120,150,160 and 140 nanoseconds. The registers that are used between the pipeline stages have a delay of 5 nanoseconds each. The total time to execute 100 independent instructions on this pipeline, assuming there are no pipeline stalls, is _____ nanoseconds.**(GATE 2021) (2 MARKS)**

Question 59

Q Consider a 6-stage instruction pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6-stage pipeline, the speedup achieved with respect to non-pipelined execution, if 25% of the instructions incur 2 pipeline stall cycles is **(Gate-2014) (2marks)**

Question 60

Q An instruction pipeline has five stages, namely, instruction fetch (IF), instruction decode and register fetch (ID/RF), instruction execution (EX), memory access (MEM), and register writeback (WB) with stage latencies 1 ns, 2.2 ns, 2 ns, 1 ns, and 0.75 ns, respectively (ns stands for nanoseconds).
To gain in terms of frequency, the designers have decided to split the ID/RF stage into three stages (ID, RF1, RF2) each of latency 2.2/3 ns.

Also, the EX stage is split into two stages (EX1, EX2) each of latency 1 ns. The new design has a total of eight pipeline stages. A program has 20% branch instructions which execute in the EX stage and produce the next instruction pointer at the end of the EX stage in the old design and at the end of the EX2 stage in the new design. The IF stage stalls after fetching a branch instruction until the next instruction pointer is computed. All instructions other than the branch instruction have an average CPI of one in both the designs. The execution times of this program on the old and the new design are P and Q nanoseconds, respectively. The value of P/Q is _____. **(Gate-2014, 2marks)** ?

Question 61

Q Consider an instruction pipeline with five stages without any branch prediction: Fetch Instruction (FI), Decode Instruction (DI), Fetch Operand (FO), Execute Instruction (EI) and Write Operand (WO). The stage delays for FI, DI, FO, EI and WO are 5 ns, 7 ns, 10 ns, 8 ns and 6 ns, respectively. There are intermediate storage buffers after each stage and the delay of each buffer is 1 ns. A program consisting of 12 instructions I1, I2, I3, ..., I12 is executed in this pipelined processor. Instruction I4 is the only branch instruction and its branch target is I9. If the branch is taken during the execution of this program, the time (in ns) needed to complete the program is **(Gate-2013) (2 Marks)**

(A) 132

(B) 165

(C) 176

(D) 328

Question 62

Q A 5-stage pipelined processor has Instruction Fetch (IF), Instruction Decode (ID), Operand Fetch (OF), Perform Operation (PO)and Write Operand (WO)stages. The IF, ID, OF and WO stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD and SUB instructions,3 clock cycles for MUL instruction, and 6 clock cycles for DIV instruction respectively. Operand forwarding is used in the pipeline. What is the number of clock cycles needed to execute the following sequence of instructions? **(Gate-2010) (2 Marks)**

Instruction	Meaning of instruction
I_0: MUL R_2, R_0, R_1	$R_2 \leftarrow R_0 * R_1$
I_1: DIV R_5, R_3, R_4	$R_5 \leftarrow R_3 / R_4$
I_2: ADD R_2, R_5, R_2	$R_2 \leftarrow R_5 + R_2$
I_3: SUB R_5, R_2, R_6	$R_5 \leftarrow R_2 - R_6$

(A) 13 **(B)** 15 **(C)** 17 **(D)** 19

Question 63

Q Consider the sequence of machine instructions given below: **(Gate-2015) (2 Marks)**

MUL R_5, R_0, R_1
DIV R_6, R_2, R_3
ADD R_7, R_5, R_6
SUB R_8, R_7, R_4

In the above sequence, R_0 to R_8 are general purpose registers. In the instructions shown, the first register stores the result of the operation performed on the second and the third registers. This sequence of instructions is to be executed in a pipelined instruction processor with the following 4 stages:
(1) Instruction Fetch and Decode (IF),
(2) Operand Fetch (OF),
(3) Perform Operation (PO) and
(4) Write back the Result (WB).
The IF, OF and WB stages take 1 clock cycle each for any instruction. The PO stage takes 1 clock cycle for ADD or SUB instruction, 3 clock cycles for MUL instruction and 5 clock cycles for DIV instruction.
The pipelined processor uses operand forwarding from the PO stage to the OF stage. The number of clock cycles taken for the execution of the above sequence of instructions is _____

Question 64

Q Consider a pipelined processor with the following four stages: **(Gate-2007) (2 Marks)**
IF: Instruction Fetch
ID: Instruction Decode and Operand Fetch
EX: Execute
WB: Write Back
The IF, ID and WB stages take one clock cycle each to complete the operation. The number of clock cycles for the EX stage depends on the instruction. The ADD and SUB instructions need 1 clock cycle and the MUL instruction needs 3 clock cycles in the EX stage. Operand forwarding is used in the pipelined processor. What is the number of clock cycles taken to complete the following sequence of instructions?
ADD R_2, R_1, R_0 $R_2 <- R_0 + R_1$
MUL R_4, R_3, R_2 $R_4 <- R_3 * R_2$
SUB R_6, R_5, R_4 $R_6 <- R_5 - R_4$
(A) 7 **(B)** 8 **(C)** 10 **(D)** 14

Question 65

Q Consider a pipelined processor with 5 stages, Instruction Fetch(IF), Instruction Decode(ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Each stage of the pipeline, except the EX stage, takes one cycle. Assume that the ID stage merely decodes the instruction and the register read is performed in the EX stage. The EX stage takes one cycle for ADD instruction and the register read is performed in the EX stage, The EX stage takes one cycle for ADD instruction and two cycles for MUL instruction. Ignore pipeline register latencies. Consider the following sequence of 8 instructions:
ADD, MUL, ADD, MUL, ADD, MUL, ADD, MUL
Assume that every MUL instruction is data-dependent on the ADD instruction just before it and every ADD instruction (except the first ADD) is data-dependent on the MUL instruction just before it. The speedup defined as follows.
Speedup = (Execution time without operand forwarding) / (Execution time with operand forwarding)
The Speedup achieved in executing the given instruction sequence on the pipelined processor (rounded to 2 decimal places) is _____
.(GATE 2021) (2 MARKS)

Question 66

Q A computer uses a memory unit of 512K words of 32 bits each. A binary instruction code is stored in one word of the memory. The instruction has four parts: an addressing mode field to specify one of the two-addressing mode (direct and indirect), an operation code, a register code part to specify one of the 256 registers and an address part. How many bits are there in addressing mode part, opcode part, register code part and the address part? **(NET 2019 DEC)**

(A) 1, 3, 9, 19

(B) 1, 4, 9, 18

(C) 1, 4, 8, 19

(D) 1, 3, 8, 20

Question 67

Q A Computer uses a memory unit with 256K word of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code and a register code part to specify one of 64 registers and an address part. How many bits are there in operation code, the register code part and the address part? **(NET-DEC- 2018)**

a) 7, 7, 18

c) 7, 6, 18

b) 18, 7, 7

d) 6, 7, 18

Question 68

Q Consider a processor with 64 registers and an instruction set of size twelve. Each instruction has five distinct fields, namely, opcode, two source register identifiers, one destination register identifier, and a twelve-bit immediate value. Each instruction must be stored in memory in a byte-aligned fashion. If a program has 100 instructions, the amount of memory (in bytes) consumed by the program text is _____ **(Gate-2016) (2 Marks)**

Question 69

Q A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter (all values in decimal)? **(Gate-2006) (1 Marks)**

(A) 400

(C) 600

(B) 500

(D) 700

Question 70

Q A processor has 16 integer registers (R_0, R_1, ..., R_{15}) and 64 floating point registers (F_0, F_1, ..., F_{63}). It uses a 2-byte instruction format. There are four categories of instructions: Type-1, Type-2, Type-3, and Type 4. Type-1 category consists of four instructions, each with 3 integers register operands (3Rs). Type-2 category consists of eight instructions, each with 2 floating point register operands (2Fs). Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand (1R+1F). Type-4 category consists of N instructions, each with a floating-point register operand (1F). The maximum value of N is _____. **(Gate-2018) (2 Marks)**

Question 71

Q A processor has 40 distinct instructions and 24 general purpose registers. A 32-bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is _____. **(Gate-2016) (1 Marks)**

Question 72

Q In the absolute addressing mode **(Gate-2002) (2 Marks)**
(A) the operand is inside the instruction

(B) the address of the operand is inside the instruction

(C) the register containing address of the operand is specified inside the instruction

(D) the location of the operand is implicit

Question 73

Q In the indirect addressing scheme, the second part of an instruction contains: **(NET-DEC-2008)**
(A) the operand in decimal form
(B) the address of the location where the value of the operand is stored
(C) the address of the location where the address of the operand is stored
(D) the operand in an encoded form

Question 74

Q A machine has a 32-bit architecture, with 1-word long instructions. It has 64 registers, each of which is 32 bits long. It needs to support 45 instructions, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer, the maximum value of the immediate operand is _____. **(Gate-2014) (1 Marks)**

Question 75

Q In _____ addressing mode, the operands are stored in the memory. The address of the corresponding memory location is given in a register which is specified in the instruction. **(NET-Aug-2016)**

(A) Register direct
(B) Register indirect
(C) Base indexed
(D) Displacement

Question 76

Q The advantage of _____ is that it can reference memory without paying the price of having a full memory address in the instruction. **(NET-JUNE-2014)**

(A) Direct addressing

(C) Register addressing

(B) Indexed addressing

(D) Register Indirect addressing

Question 77

Q Consider a hypothetical processor with an instruction of type LW R_1, 20(R_2), which during execution reads a 32-bit word from memory and stores it in a 32-bit register R_1. The effective address of the memory location is obtained by the addition of a constant 20 and the contents of register R_2. Which of the following best reflects the addressing mode implemented by this instruction for operand in memory? **(Gate-2011) (1 Marks)**

(A) Immediate Addressing

(C) Register Indirect Scaled Addressing

(B) Register Addressing

(D) Base Indexed Addressing

Question 78

Q Which of the following addressing mode is best suited to access elements of an array of contiguous memory locations? **(NET-NOV-2017)**

(A) Indexed addressing mode

(C) Relative address mode

(B) Base Register addressing mode

(D) Displacement mode

Question 79

Q In which addressing mode, the effective address of the operand is generated by adding a constant value to the contents of register? **(NET-Dec-2012)**

(A) absolute mode

(C) indirect mode

(B) immediate mode

(D) index mode

Question 80

Q Which type of addressing mode, less number of memory references are required? **(NET-JULY-2019)**

a) Immediate

c) Register

b) Implied

d) Indexed

Question 81

Q The most appropriate matching for the following pairs **(Gate-2000) (1 Marks)**

X: Indirect addressing	1 : Loops
Y: Immediate addressing	2 : Pointers
Z: Auto decrement addressing	3: Constants

(A) X-3, Y-2, Z-1

(B) X-I, Y-3, Z-2

(C) X-2, Y-3, Z-1

(D) X-3, Y-I, Z-2

Question 82

Q Which of the following addressing modes are suitable for program relocation at run time? **(Gate-2004) (1 Marks)**

(i) Absolute addressing

(ii) Based addressing

(iii) Relative addressing

(iv) Indirect addressing

(A) (i) and (iv)

(B) (i) and (ii)

(C) (ii) and (iii)

(D) (i), (ii) and (iv)

Question 83

Q For computers based on three-address instruction formats, each address field can be used to specify which of the following: **(Gate-2015) (1 Marks)**

S_1: A memory operand

S_2: A processor register

S_3: An implied accumulator register

(A) Either S_1 or S_2

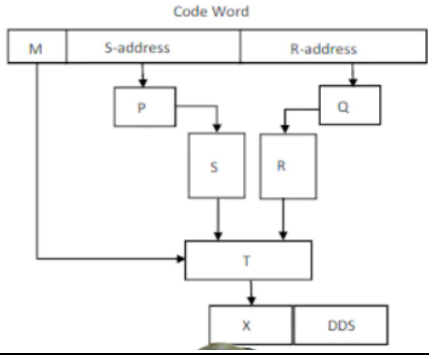
(B) Either S_2 or S_3

(C) Only S_2 and S_3

(D) All of S_1, S_2 and S_3

Question 84

Q. Consider a digital display system (DDS) shown in the figure that displays the contents of register X. A 16-bit code word is used to load a word in X, either from S or from R. S is a 1024-word memory segment and R is a 32-word register file. Based on the value of mode bit M, T selects an input word to load in X. P and Q interface with the corresponding bits in the code word to choose the addressed word.
Which one of the following represents the functionality of P, Q, and T? **(GATE 2022) (2 MARKS)**



(A) P is 10:1 multiplexer; Q is 5:1 multiplexer; T is 2:1 multiplexer

(B) P is 10:210 decoder; Q is 5:25 decoder; T is 2:1 encoder

(C) P is 10:210 decoder; Q is 5:25 decoder; T is 2:1 multiplexer

(D) P is 1:10 de-multiplexer; Q is 1:5 de-multiplexer; T is 2:1 multiplexer

---

Question 85

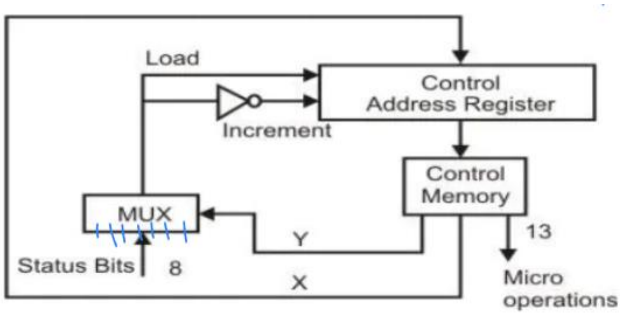
Q Consider a CPU where all the instructions require 7 clock cycles to complete execution. There are 140 instructions in the instruction set. It is found that 125 control signals are needed to be generated by the control unit. While designing the horizontal microprogrammed control unit, single address field format is used for branch control logic. What is the minimum size of the control word and control address register? (Gate-2008) (2 Marks)

(A) 125, 7 **(B)** 125, 10 **(C)** 135, 7 **(D)** 135, 10

Question 86

Q The microinstructions stored in the control memory of a processor have a width of 26 bits. Each microinstruction is divided into three fields: a micro-operation field of 13 bits, a next address field (X), and a MUX select field (Y). There are 8 status bits in the inputs of the MUX. How many bits are there in the X and Y fields, and what is the size of the control memory in number of words? **(GATE-2004) (2 Marks)**

(A) 10, 3, 1024 **(B)** 8, 5, 256 **(C)** 5, 8, 2048 **(D)** 10, 3, 512



ANSWER KEY

1	A								
2	C								
3	A								
4	A								
5	4								
6	2								
7	B								
8	B								
9	14								
10	24								
11	20								
12	A								
13	A								
14	D								
15	D								
16	A								
17	D								
18	A								
19	B								
20	C								
21	B								
22	D								
23	76								
24	14								
25	B								
26	30								
27	0.05								
28	4.72								
29	1.96								
30	C								
31	C								
32	C								
33	A,B,D								
34	B								
35	C								
36	A								
37	A								
38	A								
39	A								
40	B								
41	6.11								
42	14020								
43	D								
44	C								
45	D								
46	B								

47	80000								
48	456								
49	D								
50	B								
51	C								
52	C								
53	B								
54	1.508								
55	33.33%								
56	3.2								
57									
58	17160								
59	4								
60	1.54								
61	B								
62	B								
63	13								
64	B								
65	1.87								
66	C								
67	C								
68	500								
69	C								
70	32								
71	16								
72	B								
73	C								
74	16383								
75	B								
76	D								
77	D								
78	A								
79	D								
80	B								
81	C								
82	C								
83	A								
84	C								
85	D								
86	A								