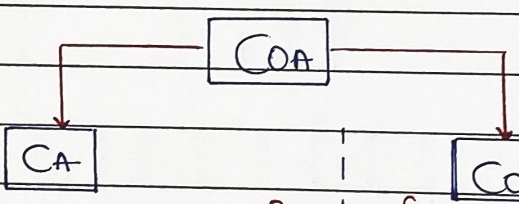
COMPUTER ORGANIZATION & ARCHICTECTURE.

Page No.
Date. 15/09/23

Computer Generation:

	1st	2nd	3rd	4th and 5th
	1942-1955	1955-1964	1965-1974	1974-Present
Components	Vacuum Tube	Transistor	I.C (Integrand Chip)	VLSI & ULSI.
Languages.	Machine language	Assembly language	H.L.L., OOPs RDBMS	OOPs, RDBMS, ML, AI.

* The First Digital Computer was "ENIAC" in 1943. (Electronic Numerical and Integrator Computer).



{Computer Architecture}

→ It deals with:
* Instruction
* Addressing mode
* ALU
* Pipeline (Internal Design)

→ Computer Architecture is a attribute which is visible to the programmer.

{Computer Organisation}

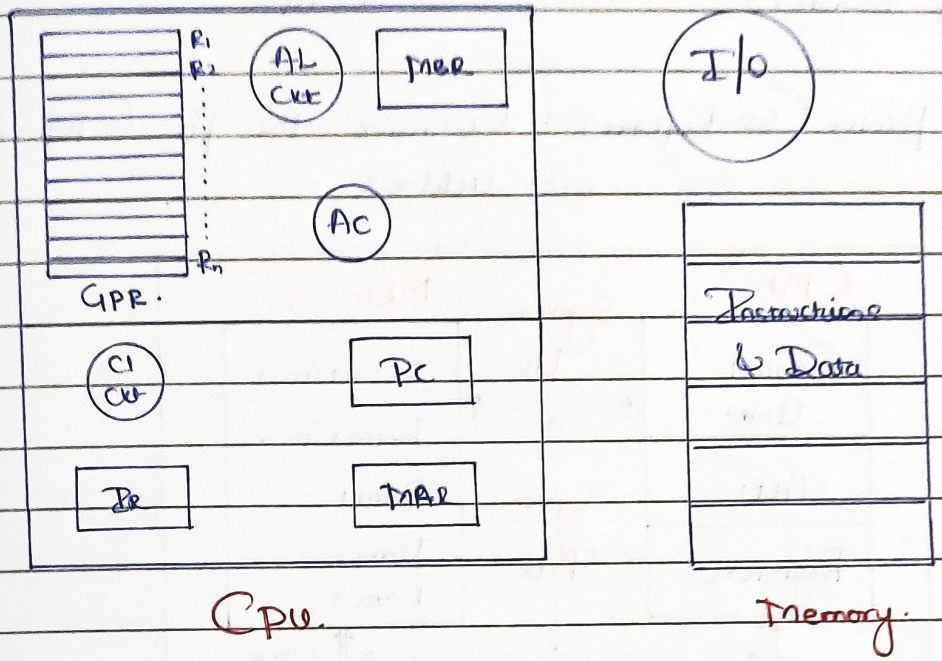
→ It deals with:
How various memory & I/O interact with System (Memory organisation & I/O Organisation).

→ It also deals with how features will be implemented.

Note: Intel X86 has same architecture but different organisation.

Page No.
Date. / /

Components of a Computer:



1. **Program Counter : [PC] :** It contains starting address of the next instruction to be executed (fetched).

2. **Memory Address Register [MAR] :** MAR contains the memory address used for either read or write operation.

3. **DR/MDR/MBR (Memory Buffer Register) :** Hold the instruction or data.

4. **IR (Instruction Register) :** It contains the instruction which is currently executed by the CPU.

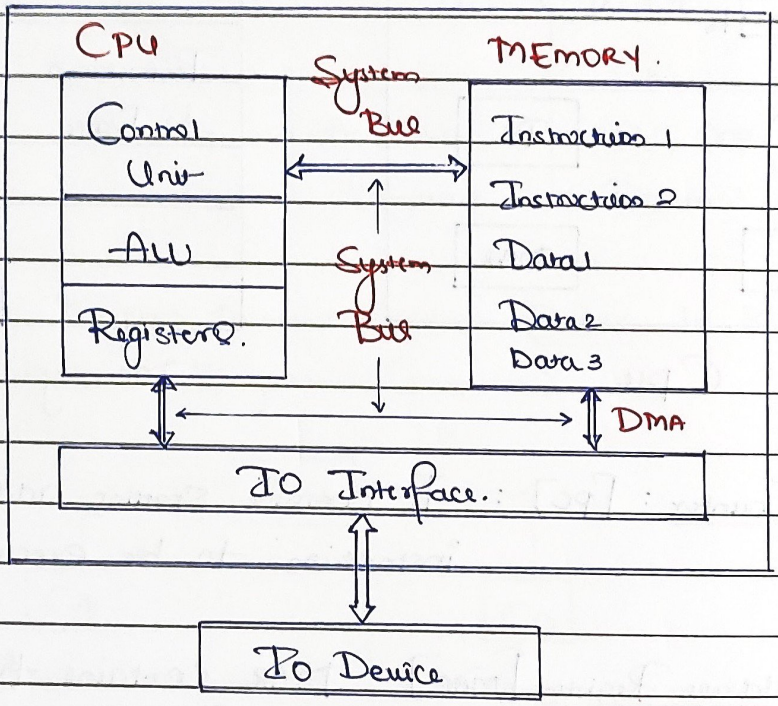
Note:
* Instruction is stored in IR because instruction format is predefined in IR.

5. **AC (Accumulator) :** It contains the temporary result of ALU operation or first operand (data) of ALU operation.

6. **GPR (General Purpose Register) :** It is used for processing the data.

⑧ PSW (Program Status Register) : It store the status of the ALU / Flag Register result.

⑨ Stack Pointer (SP Register) : Contains the TOS (TOP of the Stack) address.



Instruction Cycle:

* The process required for each instruction execution.
or
* The instruction cycle describe the execution sequence of the instruction.

Instruction Cycle contains two sub cycles:

1. Fetch Cycle: To fetch (to bring) the instructions from memory to CPU and at the end of fetch cycle PC (program counter) is incremented, now PC will denote next instruction starting address.

2. Execute Cycle: To process (to execute) the fetched instruction.
   (i) Decode (ii) Execute

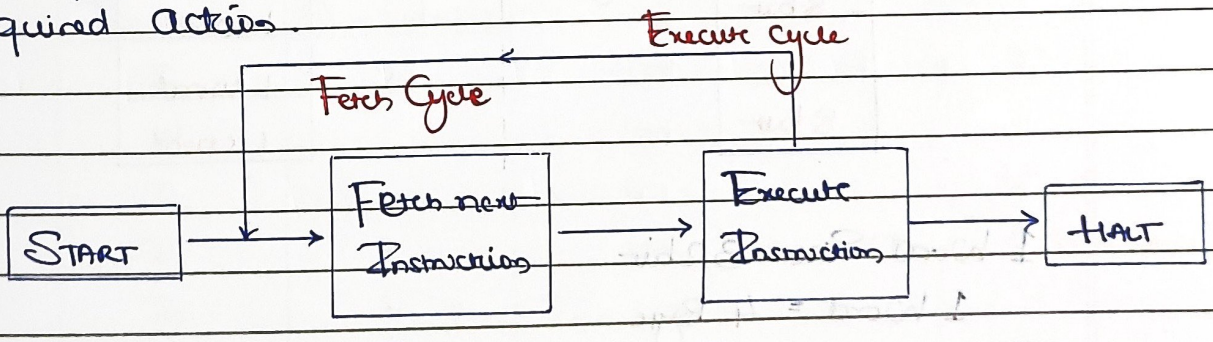
Page No.
Date. / /

Steps in Instruction Cycle:

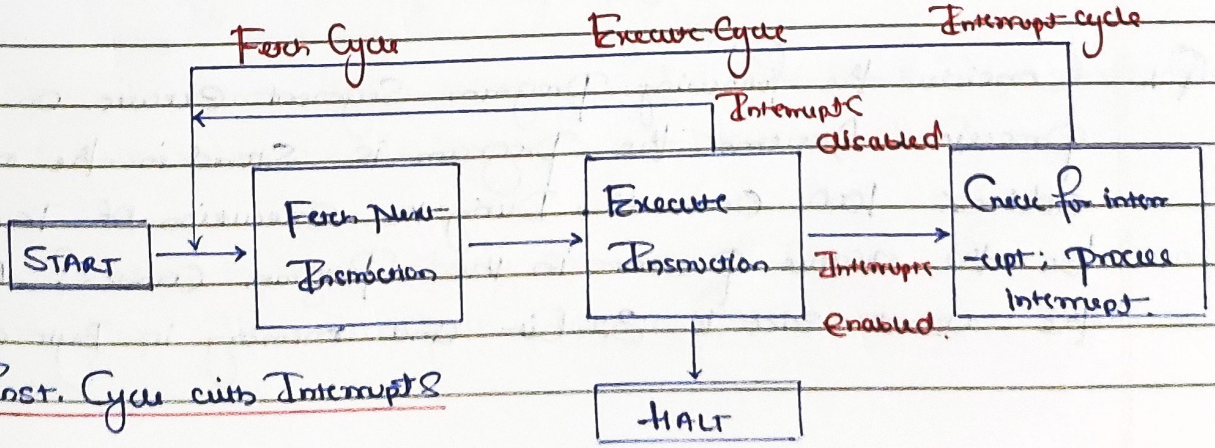
1. IAC (Instruction Address Calculation) } Fetch Cycle
2. IF (Instruction Fetch)
3. Decoding (Analysis of instruction)
4. OAC (Operand address Calculation) } Execute Cycle
5. OF (Operand Fetch)
6. DP (Data Processing)
7. Result Storage.

Fetch Cycle:

* At the beginning of each instruction cycle the processor fetches an instruction from memory
* The Program Counter holds the address of the instruction to be fetched next.
* The processor increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.
* The fetched instruction is loaded into the instruction register.
* The Processor interprets the instruction and Performs the required actions.



Basic Instruction Cycle.

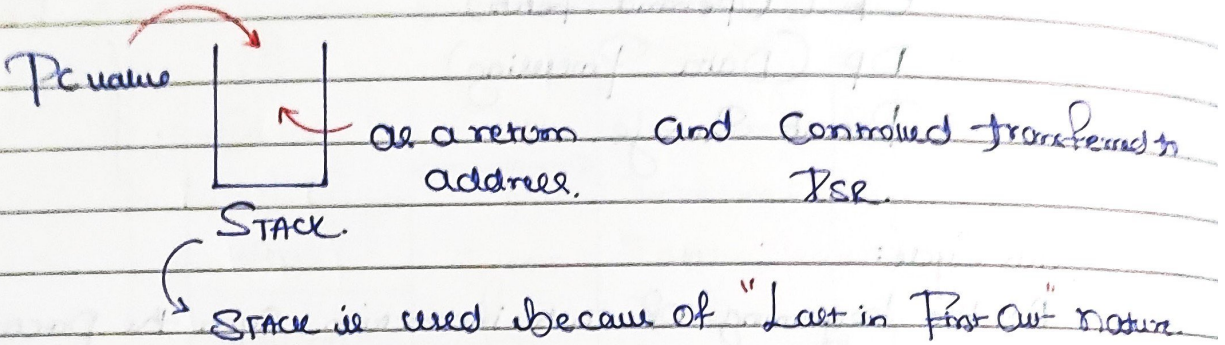


Inst. Cycle with Interrupts

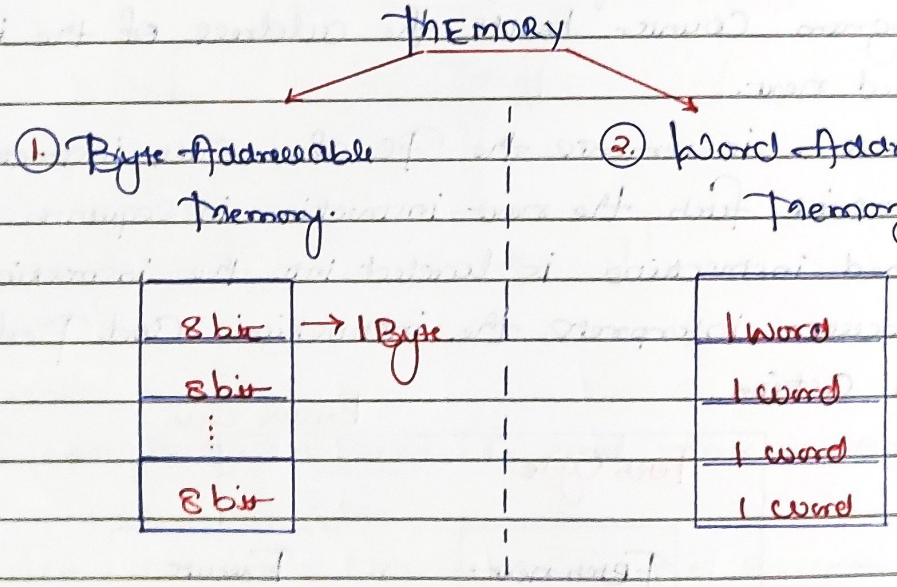
Page No.
Date: / /

Note: *When interrupt occur then after completion of current instruction execution, interrupt will be serviced.

*When interrupt occur, it push the PC value into the stack as a return address and controlled transferred to ISR (Interrupt Sub Routine).



STACK is used because of "Last in First Out" nature.



1 word Size = 32 bit

1 Word = 4 Byte

I1 : 1 word = 4 Byte.

Q1. Consider the following program segment execute on hypothetical processor. Assume the program is stored in the memory address 1000 onwards. During the execution of I6 what could be the value present in the Program Counter? Assume the word size is 32 bit and memory is Byte addressable?

Page No.
Date. / /

Ans.

Instruction	Size (words)
I_1	2
I_2	1
I_3	1
I_4	3
I_5	1
I_6	2
I_7	1.

1 Word = 32 bit
= 4 Byte
2 word = 2 × 4 = 8 Byte

1000 — 1007
1008 — 1011
1012 — 1015
1016 — 1027
1028 — 1037
I_6 — 1032 — 1039
1040 — 1043

Byte addressable

Stored in PC during exe. I_6

∴ Value present in PC during execution of I_6 = 1040

Q2. Consider the following program segment execute on Hypothetical processor. Assume that word size is 32 bit and memory is word addressable. The program is stored in the memory at address 1000 onwards. During the execution of I_5 what would be the value present in the Program Counter?

Ans

Instruction	Size (words)
I_1	2
I_2	1
I_3	1
I_4	3
I_5	1
I_6	2
I_7	1.

1000 — 1001.
1002
1003
1004, 1005, 1006
I_5 — 1007
1008 — 1009
1010

Word Addressable

→ PC will store 1008 (address of new instruction i.e I_6)

PC will store I_6 address during execution of I_5.

* If interrupt occure during execution of I_5 or after execution of I_5 then PC will push I_6 value (1008) into stack as return address.

Page No.
Date. / /

Q3. Consider the following program segment for a hypothetical CPU having three user registers R_1, R_2 and R_3.

	Instruction	Operation	Instruction (words)	
I_1	MOV R_1, 5000	$R_1 \leftarrow$ Memory[5000]	2	1000 – 1001
I_2	MOV R_2, (R_1)	$R_2 \leftarrow$ Memory[(R_1)]	1	1002
I_3	ADD R_2, R_3	$R_2 \leftarrow R_2 + R_3$	1	1003
I_4	MOV 6000, R_2	Memory(6000) $\leftarrow R_2$	2	1004 – 1005
I_5	HALT	Machine Halt	1.	1006

Consider that the memory is word addressable with size 32 bits and the program has been loaded starting from memory location 1000. If an interrupt occurs during ADD instruction, what will be the return address pushed into stack?

Ans Interrupt occurs at Add(I_3), ∴ PC will store I_4^{th} address
⇒ 1004

Q4. Consider the following program segment for a hypothetical CPU having three user registers R_1, R_2, R_3.

	Instruction	Operation	Instruction (words)	
I_1	MOV R_1, 5000	$R_1 \leftarrow$ Memory[5000]	2	1000 – 1007
I_2	MOV R_2, (R_1)	$R_2 \leftarrow$ Memory[(R_1)]	1	1008 – 1011
I_3	ADD R_2, R_3	$R_2 \leftarrow R_2 + R_3$	1	1012 – 1015
I_4	MOV 6000, R_2	Memory[6000] $\leftarrow R_2$	2	1016 – 1023
I_5	HALT	Machine Halt	1	1024 – 1027

Consider that the memory is Byte addressable with size 32 bits and the program has been loaded starting from memory location 1000. If an interrupt occurs while the CPU has been halted after executing "MOV 6000, R_2" instruction the return address saved in the stack will be

Ans MOV 6000, R_2 = I_4 instruction interrupted, so I_5^{th} instruction address will be stored in stack ⇒ 1024

Page No.
Date. 23/09/23

Q5. Consider the following Program Segment for hypothetical CPU

Instruction	Meaning	Instruction Size (words)	
I_1 MOV R0, 3000	R0 ← M[2000]	3	3×2 + 4 = 13
I_2 MOV R1, 3000	R1 ← M[3000]	3	3×2 + 4 = 13
I_3 ADD R0, R1	R0 ← R0 + R1	1	1×3 + 6 = 9
I_4 MOV 6000, R0	M[6000] ← R0	3	3×2 + 4 = 13
I_5 HALT	Machine Halt	1	1×3 = 3

51 cycles

Let the clock cycle required for various operations be as follows: Instruction Fetch & Decode : 3 clock cycle per word
ADD with both operand and stored in register : 6 clock cycle. Register to/from memory transfer : 4 clock cycle
The total number of clock cycle required to execute the program is 51 cycle.

Q6. Consider the following Program Segment for a hypothetical CPU having three User registers R_1, R_2 and R_3

Instruction	Operation	Instruction Size	
MOV R_1, 5000	R_1 ← Memory[5000]	2	2×2+3 = 7
MOV R_2, (R_1)	R_2 ← Memory[(R_1)]	1	1×2+3 = 5
ADD R_2, R_3	R_2 ← R_2+R_3	1	1×2+1 = 3
MOV 6000, R_2	Memory[6000] ← R_2	2	2×2+3 = 7
HALT	Machine Halt	1	1×2+0 = 2

24 cycles.

Let the clock cycles required for various operations be as follows:
Register to/from memory transfer: 3 clock cycles
ADD with both operand in register: 1 clock cycle.
Instruction fetch and decode: 2 clock cycle per word
Total no of clock cycles required to execute this program 24

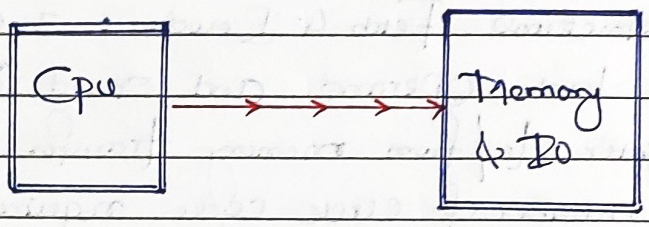
System Bus:

System bus is a collection of lines which are used to provide the communication between major components of the computer (Memory, I/O, CPU)

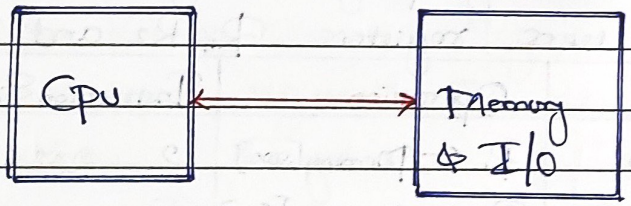
System Bus Contain 3 types of bus/lines

1. Address line / Address bus
2. Data line / Data bus
3. Control line / Control bus.

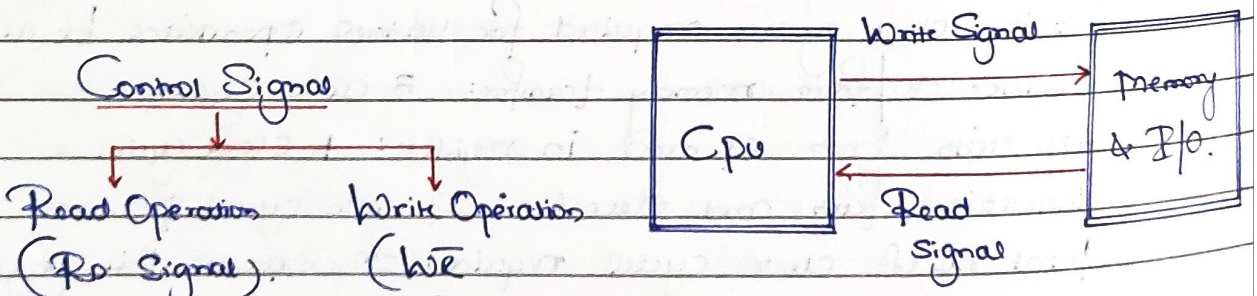
1. **Address line**: * Address line are used to carry the address towards memory and I/O.
* Address line are unidirectional.



2. **Data Line**: Data line are used to carry the data (Binary sequence). Data line are bi-directional.



3. **Control Line**: Control line are used to control the carry signal. Control lines are individually unidirectional and collectively bidirectional.

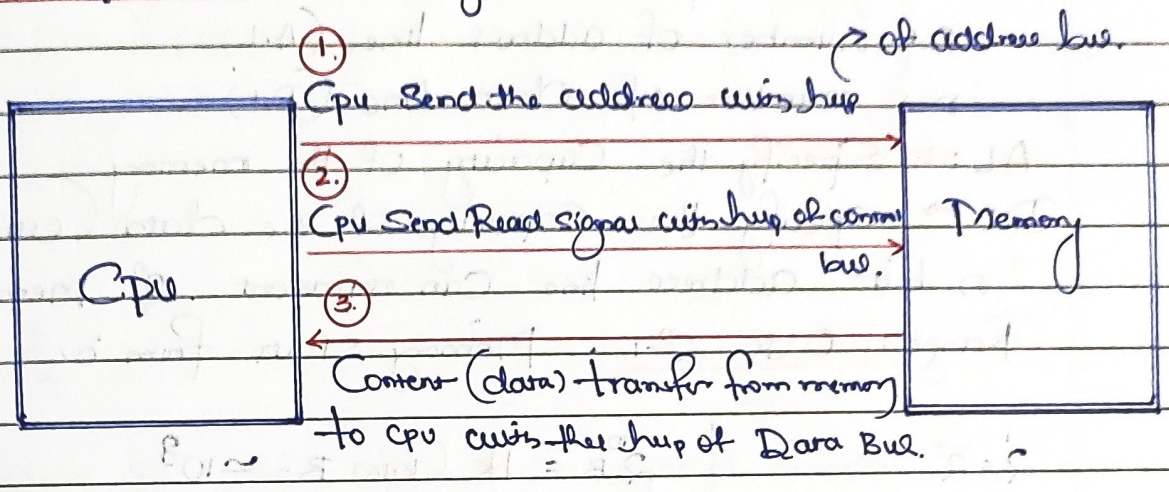


Memory Read: Memory to Cpu
Memory Write: Cpu to Memory

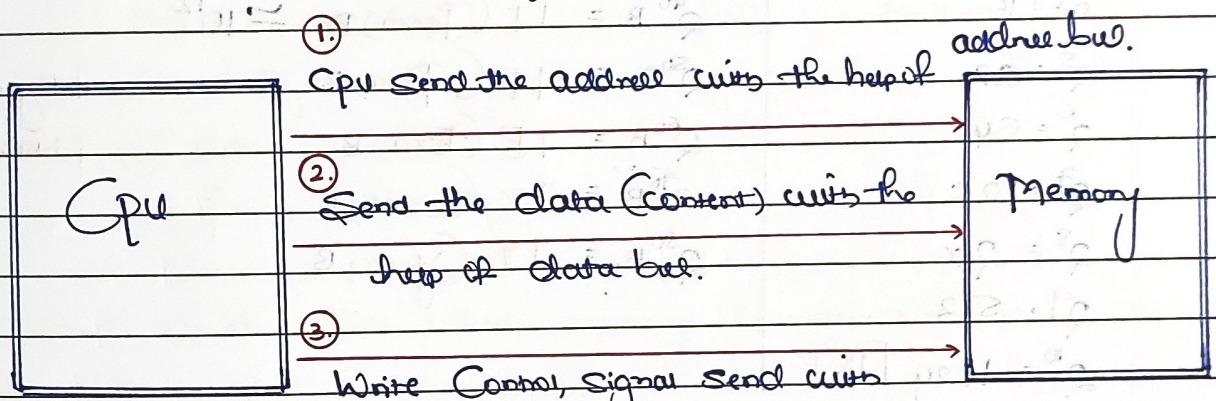
LOAD: Memory Read (LD)
STORE: Memory Write (ST)

Page No.
Date. / /

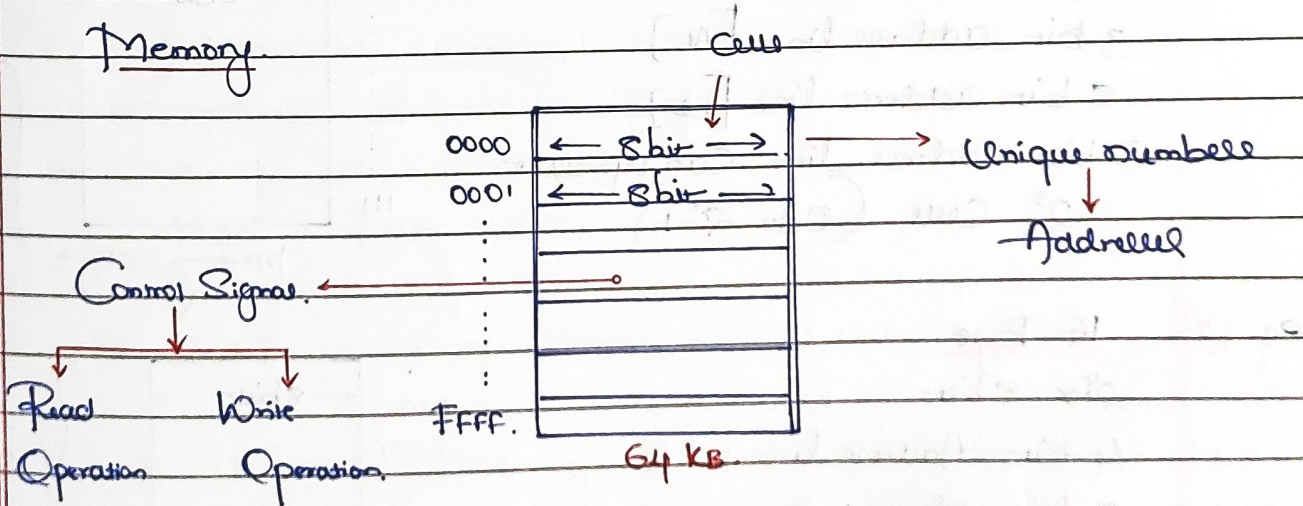
Read Operation: (Memory Read).



Write Operation: (Memory Write)



Memory.



* Memory is organised into equal parts, each part is called cell
* Each cell is identified by a unique number called as address.

→ Memory is represented as $2^n \times m$.

n: number of address line (A.L)

m: number of data line (D.L)

AL → Specify the capacity of the memory

DL → Specify the capacity of the data (cell size)

* n bit address line can represent 2^n memory cells

Range: 0 to $2^n - 1$. Memory starts from 0.

$2^1 = 2$	$2^{10}B$ = 1K (kilo) B $\simeq 10^3$
$2^2 = 4$	$2^{20}B$ = 1M (mega) B $\simeq 10^6$
$2^3 = 8$	$2^{30}B$ = 1G (Giga) B $\simeq 10^9$
$2^4 = 16$	$2^{40}B$ = 1T (Tera) B $\simeq 10^{12}$
$2^5 = 32$	$2^{50}B$ = 1P (Peta) B
$2^6 = 64$	$2^{60}B$ = 1E (Exa) B
$2^7 = 128$	$2^{70}B$ = 1Z (Zetta) B
$2^8 = 256$	$2^{80}B$ = 1Y (Yotta) B
$2^9 = 512$	
$2^{10} = 1024$ [1K]	

1 Byte = 8 bit

1 Nibble = 4 bit

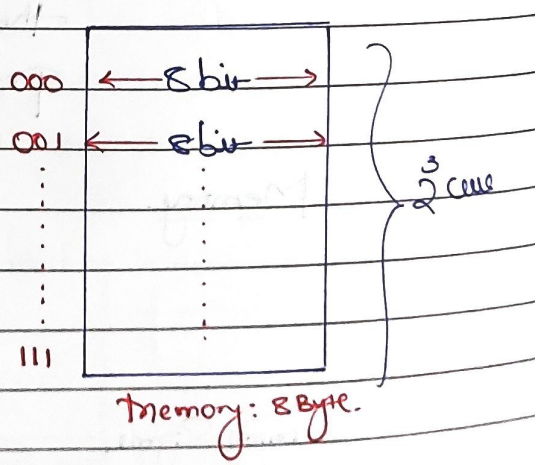
eg ① 8 Byte

$2^3 \times 8$ bit

3 bit address line [AL]

8 bit address line [DL]

3 bit address line can represent 2^3 cells (0 to $2^3 - 1$)



eg. ② 16 Byte

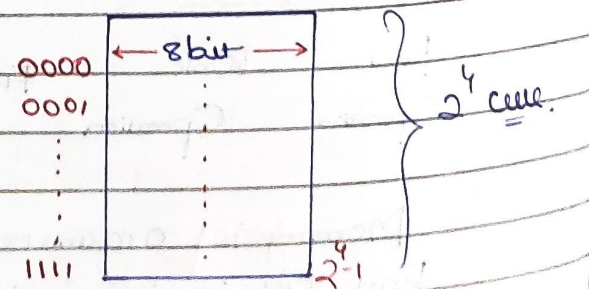
$2^4 \times 8$ bit

4 bit Address line

8 bit data line

4 bit A.L can represent

2^4 memory cells (0 to $2^4 - 1$)



Page No.
Date. / /

Q1. The capacity of a memory Unit is defined by the number of word multiplied by number of bit/word. How many seperate address and data lines needed for memory of 64K x 16?

Ans 64K x 16

$2^6 \cdot 2^{10} \times 16 = 2^{16} \times 16$

∴ 16 Bit Address and 16 Bit data lines

Q2. Consider a System which has 1024 K words. Each word has a size of 32 bit then what is the Capacity of memory in MB (Mega Byte) 4 MB

1024 K words

$2^{10} \times 2^{10} \times 2^2$ Byte

2^{22} Byte $= 2^2 \times 2^{20}$

$= 4$ Mega Byte

1 Word = 32 bits

⇒ 4 Bytes

→ Based on the Cell Size memory configuration is divided into 2 types:

1. Byte addressable memory
2. Word addressable memory

Byte Addressable Memory: When the Cell Size is '8bit' then corresponding address is called Byte addressable memory.

(2^2) 4 x 8: 2 bit address

(2^4) 16 x 8: 4 bit address

(2^6) 64 x 8: 6 bit address.

→ $2^n \times 8$: 'n' bit address. (Cell Size is 8 bit)

Word Addressable Memory: When the Cell Size is given in the form of words (word length) then corresponding address is word addressable memory

$2^n \times w$: n bit address (each Cell Size must be a word)

Microprocessor	Byte-Addressable [B]	Word-Addressable
8 bit Processor	8 bit	8 bit
16 bit Processor	8 bit	16 bit
32 bit Processor	8 bit	32 bit
	8 bit	
n bit Processor	8 bit	n bit

(Word = Word length)

Unique meaning ⇓ No ambiguity.

multiple meaning ⇓ Ambiguity.

Note:

* Default configuration in memory is Byte addressable so in the memory data always Stored Byte wise.
* Default data type in the CPU is word so operations are performed on a CPU based on word format.
* So to Synchronise the memory data type (Byte) with a CPU data type (word), memory interfacing will be adjusted by the designer according to the word length of the CPU to access the data from memory to CPU in the form of words.

eg: 16 bit processor
Word length = 16 bit
Operations are performed on 16 bit data.

Q1. The Capacity of a memory unit is defined by the number of word multiplied by the number of bit/word. How many seperate addresses and data lines are needed for memory of 64K x 16?

Ans $2^6 \times 2^{10} \leftarrow 64K \times 16$

$2^{16} \times 16$ ∴ 16 bit address line

16 bit data line

Q2. The Capacity of a memory unit is defined by the number of

Page No.
Date. / /

word multiplied by the number of bit/word. How many seperate addresses and data lines are needed for a memory of 4K x 16?

Ans $4K \times 16 \Rightarrow 2^{12} \times 16$

$\therefore$ 12 bit address line

16 bit data line.

Q3. A processor can support a maximum of 4GB memory, where the memory is word addressable (a word consist of 2 bytes). The size of the address bus is of the processor is atleast 31 bits.

Ans 4GB

$2^6 \times 2$ Byte $\quad \frac{4GB}{2B}$ words

Memory: word addressable

1 Word: 2 Byte

2 Byte => 1 Word

2^6 Words

2×2^{30} Words

2^{31} Word $\Longrightarrow$ 31 bit

Q4 Consider a 32bit hypothetical processor which support 128 MByte memory. System is enhanced (new design) with a word addressable memory. Then how many address lines are required in the new System?

Ans. 128 MB $\Longrightarrow$ 32m x 4Byte

32m Words

2^{25} Word $\Longrightarrow$ 25 bit Address lines

Byte Addressable

2^{27} Byte (128 MB)

$\Downarrow$

$2^{25} \times 4$ Byte

2^{25} Words or 32m words

Word Addressable

2^{25} Words (32 m words)

1 Word = 4 Byte

$2^{25} \times 4$ Byte

2^{27} Byte.

128 MByte

Page No.
Date. 25/09/23

Qs. Consider a 64 bit hypothetical processor which support 2G word memory. System is enhanced (new design) with a Byte addressable memory. Then how many extra address lines are required in the new system?

Ans

Old design:	New design: Byte addressable
2G Words	1 Word = 64 bit = 8 Byte
$2^1 \cdot 2^{30}$ Word	2G words → 2G × 8 Byte
2^{31} Words	16 G Byte = $2^4 \cdot 2^{30}$ B
Address = 31 bit	= 2^{34} Byte
	Address = 34 bit

∴ Extra = 34 − 31
= 3 bit

Pins: Processor contain set of hardware pins to perform operations
1. Active low pin
2. Active high pin
3. Dual pin
4. Time multiplexed pin

(1) Active Pin: This pin is enabled when the input is '0' or clock pulse is in low state. It is denoted as pin name
eg: $\overline{RD}$, $\overline{WR}$ etc.

(2) Active high pin: This pin is enabled when the input is '1' or clock pulse is in high state.
eg: INTR, HLDA, ALE

(3) Dual pins: eg: M/$\overline{IO}$
1: then memory operation (Read or write)
0: then IO operation (IO Read or IO write)

Page No.
Date. / /

4. Time Multiplexed Pins: This pin carries the multiple meaning but one only at a time. Address pins are time multiplexed with data pin to carry the data and address. (But only one at a time).

eg: In 8085 Processor

$AD_0 - AD_7$

Address Data.

eg: In 8086 Processor

$AD_0 - AD_{15}$

Address Data

In 8085 $AD_0 - AD_7$: Address pins are time multiplexed with data pin to carry the data but to only once at a time.

ALE (Address latch enabled).

→ 1 Time multiplexed pin carries the address

→ 0 Time multiplexed Pin carries the data

* The advantage is number of hardware pins reduced.

System Bus:

1. Address Line: Address lines are used to carry the address towards memory and I/O (unidirectional).
Based on the address line we can determine capacity of the memory.

eg: In 8085 processor

$AD_0 - AD_7$ and $A_8 - A_{15}$

Address = 16 bit.

$= 2^{16}$ cells, 64K cells

= 64 KB

eg: In 8086 processor

$AD_0 - AD_{15}$ & $A_{16} - A_{19}$

Address = 20 bit.

$= 2^{20}$ cells

= 1M cells.

2. Data Line: Data line are used to carry the data (bidirectional).
Based on the data line we can determine the word length of the Cpu. (processor).

Page No.
Date. / /

The performance of the processor is measured by word length of the CPU.

eg: In 8085
- AD_0 - AD_7
- Word length = 8 bit
- Operation performed on 8 bit data format.

eg: In 8086
- AD_0 - AD_{15}
- Word length = 16 bit
- Operation performed on 16 bit data format.

3. Control Lines: → Carries the control signal
   → Unidirectional (individually)

Q1. 64K × 8
- 64K → memory cells. (2^{16} cells)
- 8 → each cell size / size of each word

∴ 16 bit required to represent any of the cells.
Width of address bus = $\lceil \log_2 \text{Memory Size} \rceil = \lceil \log_2 64K \rceil$ = 16 bit
Width of data bus = 8 bit.

Q2. 4G × 32
Width of address Bus = 32 bit
Width of data bus = 32 bit.

32 bit Processor
- Data line = 32 bit
- Data bus = 32 bit
- Word Size = 32 bit
- ALU = 32 bit
- AC = 32 bit
- MBR / MDR = 32 bit
- Registers = 32 bit

Page No.
Date. / /

Computer: Computer is a computational device used to process the data under the control of program.

Program: Sequence (set) of Instructions along with data.

Program → Instructions
Program → Data

Instructions: It is a binary sequence (code) which is designed inside the processor to perform the actions/operations.

Or

Instruction is a set (sequence) of binary bits to instruct the computer to perform the operations.

eg: Assume in processor.
0110 → Add
1011 → Sub

Data: Data is a binary sequence bind with the value (data format BCD, Hex, etc).

Architecture
① Von Neumann Architecture
② Harvard Architecture

* Computer works on "Stored Program Concept".

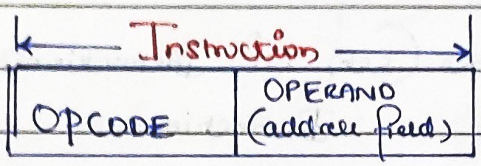
Von Neumann Architecture (Stored Program Concept)

* Main memory contain the instructions and data.
* ALU operating on Binary data
* Control unit interpreting the instructions from memory and executing.
* Input/Output equipment operated by control unit.

Page No.
Date. 25/09/21

MACHINE INSTRUCTION & ADDRESSING MODES

Instruction Format



OPCODE : Operational code → Type of Operation.

OPERAND : Data or Address of Operand.

eg: * 2 bit OPCODE can perform $2^2 = 4$ operation.
3 bit OPCODE can perform $2^3 = 8$ operation

> n bit OPCODE can perform 2^n operation.

If memory size is given then we can calculate address field size.

eg: if memory = 1MByte ⇒ 20 bit Address Line.

Q1. If instruction size is 16 bit & memory is 1K Byte then how many number of operation supported by the system?

Ans. Instruction 16 bit

OPCODE	A.F
6 bit	10 bit

Memory = 1K Byte
$= 2^{10}$ Byte
∴ AF = 10 bit

If opcode = 6 bit
then total # operations = 2^6 = 64 operations

n bit address field

- Word-Addressable: 2^n Words
- Byte-Addressable: 2^n Byte

Page No.
Date. / /

Important Points:

* If n bit Opcode can perform 2^n Operations.
* If N operations then Opcode = $\lceil \log_2 N \rceil$ bits.
* If n bit address line can represent memory of capacity 2^n.
* If memory capacity of m B/w then address field = $\lceil \log_2 m \rceil$ bits.

Q1. If opcode is 6 bit then total # Operations?

Ans Opcode = 6 bit, total # of Operations = 2^6 = 64.

Q2. If 100 Operation supported by the System then Size of opcode?

Ans 100 Operation $\simeq 2^7$.

Opcode = 7 bit.

Q3. If memory is 256 KB then Size of the address field?

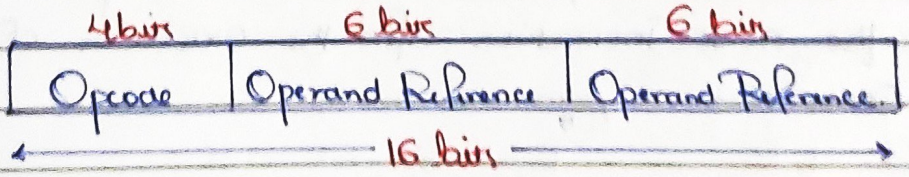
Ans 256 KB = 2^{18} Byte ∴ Address = 18 bit.

Q4. If A.F is 29 bit then Size of memory?

Ans A.F = 29 bit, Memory = 2^{29} Byte => 512 MByte.

Instruction Representation:

* Within the Computer each instruction is represented by a sequence of bits.
* The instruction bits divided into fields, corresponding to the constituent elements of the instruction.



* Opcodes are represented by abbreviations called mnemonics.

eg: ADD — Add
SUB — Subtract.

MUL — Multiply
DIV — Divide
LOAD — Load from memory the data.
STORE — Store data to memory.

* Operands are also represented symbolically
* Each Symbolic Opcode has a fixed binary representation.

Machine Instruction Characteristics:

* The operations of the processor is determined by the instructions it executes, refered to as machine instructions or computer instructions.
* The collection of different instructions that the processor can execute is refered to as the processor's instruction Set.

Elements of a machine Instruction:

Operation code (Opcode): Specifies the operation to be performed. The operation is specified by a binary code, known as the Operation Code, or Opcode.

Source Operand Refrence: The Operation may include one or more source operands, that is Operands that are inputs for the Operation.

Result Operand Refrence: The Operation may produce a result.

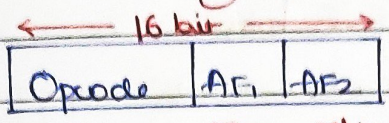
Next Instruction Reference: This tells the processor where to fetch the next instruction after the execution of this instruction is Complete.

Q1. Consider a hypothetical processor which support 128 byte memory and instruction length is 16 bit.

(i) If 2AF is used then how many total number of operation supported?

Page No.
Date. / /

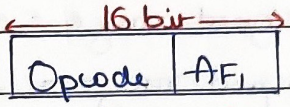
Ans 128 Byte $\Rightarrow 2^7$ Byte $\Rightarrow$ AF = 7 bit
Instruction length = 16 bit



if Opcode = 2 bit
Total # Operations = 2^2
= 4 Operations

(ii) If 1AF (address field) is used then how many total number of Operations Support formulated?

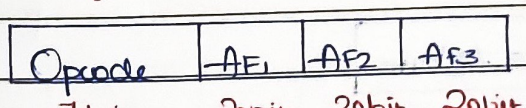
Ans 1AF / 1AI



Total # of Operations = 2^9
= 512 Operations

Q2 A Hypothetical processor support 100 different operations and 3 address memory field (same size). Instruction is stored in 1MB memory. Then what is the length of the instructions?

Ans Memory = 1MB (2^{20}B), AF = 20 bit.
100 Operation = Opcode = 7bit
or $\lceil \log_2 100 \rceil = 2^7 = 100$



Instruction length = 7 + 20 x 3
= 67 bit

Storage

Registers (fastest)

Memory → Cache memory
→ Main memory / primary memory
→ Secondary / Auxiliary memory

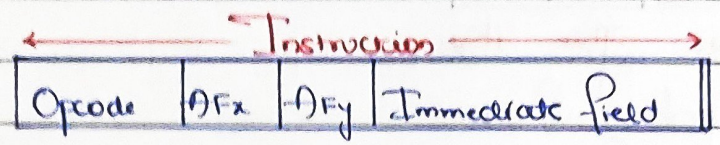
Fastest : Register > Cm. > M.m. > Sm
Size : S.m > M.m > C.m > Register.

Register → Reg. A.F
Memory → Mem A.F

Page No.
Date. / /

Q1. Memory 4MB = 2^{22} Byte → Memory-AF = 22 bits.

Q2. 50 Registers = 2^6 → Reg-AF = 6 bits.



Sometimes instruction having immediate field also along with AF.

→ If immediate field = n bit

n bit immediate Unsigned range = 0 to $2^n - 1$

n bit immediate Signed range = -2^{n-1} to $+(2^{n-1}-1)$

eg: Assume: if immediate field is 7 bit.

Unsigned = 0 to $2^n - 1$ = 0 to $2^7 - 1$ = 0 to 127

Signed = $-(2^{n-1})$ to $+(2^{n-1}-1)$ = −64 to +63.

Instruction Set Size of 10

↓

10 Different type of Operation/instructions performed

↓

Opcode = 4 bit

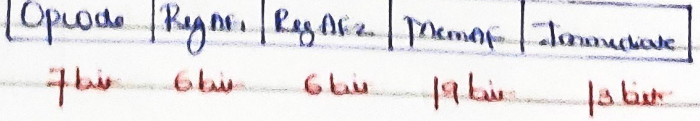
Q1. Consider a hypothetical CPU which supports 110 instructions 50 registers and 512 KB memory space. Instructions contain 2 register operands, memory operands and 13 bit immediate constant fields. Program contain 300 instructions Memory storage space required in Bytes to store the Program is 2100 Byte

Ans. 110 Instructions/Operations = Opcode = 7 bit

50 Register = Reg-AF = 6 bit

512 KB Memory = Mem-AF = 19 bit

2^{19} B.



Page No.
Date. / /

Instruction Size (length) = 7+6+6+19+13 = 51 bit

Instruction Size = $\lceil \frac{51 \text{ bit}}{8 \text{ bit}} \rceil$ Byte = 7 Byte

Program Contain 300 instructions

Program Size = 300 × 7 Byte = 2100 Byte

Q2. Consider a processor which contain the following pin structure

$A_{00} - A_{023}$, $A_{24} - A_{39}$.

Processor Contain 250 Register instruction is designed with 4 fields i.e OPCODE, register address, memory address and 16 bit immediate field

Processor Support 180 instruction (Operations). A program Contain 400 instruction then how much Space is required for the program

(i) in Byte?

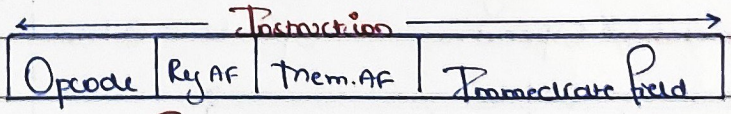
(ii) in Words?

$A_{00} - A_{23}$, $A_{24} - A_{39}$ = Memory address = $A_0 - A_{39}$ = 40 bit

Dataline $A_{00} - A_{23}$ = 24 bit

250 Register = Reg AF = 8 bit

180 Operation/instruction ⇒ Opcode = 8 bit



Instruction length (Size) = 8 + 8 + 40 + 16 = 72 bit

1 Instruction Size = $\frac{72 \text{ bit}}{8 \text{ bit}}$ = 9 Byte

Program Contain = 400 Instruction.

So program size = 9 × 400 = 3600 Byte

Program Size in Word:

1 Word Size = 24 bit = 3B

Program Size = 3600 Byte, in Word = $\frac{3600 \text{ Byte}}{3 \text{ Byte}}$ Word

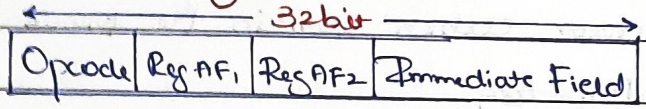
= 1200 Words

Page No.
Date. / /

Q3. 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

Ans. 40 Instruction → Opcode = 6 bit

24 Register → Reg-AF = 5 bit



Immediate field = 32 − (6+5+5)
= 16 bit

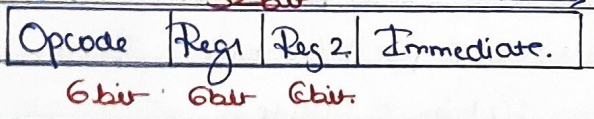
Q4. A machine has a 32 bit architecture, with 1 word long instructions. It has 64 registers, each of which is 32 bit long. It needs to support 45 instructions, which have an immediate operand in addition to two register operands. Assuming that the immediate operand is an unsigned integer the maximum value of the immediate operand is _____.

Ans. 1 Word long instruction

∴ 1 Instruction Size = 32 bit

64 Register = 6 Reg AF = 6 bit

45 Instructions / Operations ⇒ Opcode = 6 bit



Immediate field = 32 − (6+6+6)
= 14 bit

Unsigned Immediate field
Maximum = 0 to $2^{14}-1$
= 0 to 16383
∴ 16383

Q5. Consider a processor with 64 registers and 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)

Page No.
Date. / /

Consumed by the program text is

Opcode	Reg1	Reg Adr2	Reg Adr3	Imm. F
4 bit	6 bit	6 bit	6 bit	12 bit

Instruction Size = 4+6+6+6+12 = 34 bits.

1 Instruction Size = 5 Byte

Program contains = 100 Instr.

Program size = 100 × 5B

= 500 Byte

Instruction Set Architecture Classification:

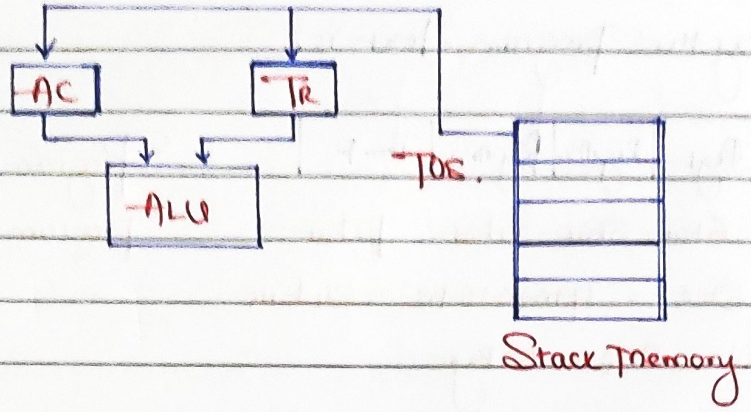
1. Stack organisation
2. Single accumulator
3. General register organisation.

→ CPU Organisation is classified into 3 types based on the availability of ALU Operand (Data). (AF: Address Fields or AI: Address Instruction).

1. Stack CPU (0AF)
2. Accumulator CPU (1AF)
3. General purpose Register Organisation
   (i) Reg-Memory reference CPU (2AF)
   (ii) Reg-Reg reference CPU. (3AF)

Stack Organisation: [Stack CPU]

* In the Stack based organisation ALU operations are performed only on Stack data.
* So both the Operand (Data) must be required in the Stack and after processing result after processing is also stored in the stack.
* Stack is part of memory which is initialised by (Stack pointer) SP register.
* In stack insert and delete operation are performed at the same level/end called TOS (Top of Stack) so its become default location.



Push → Insert } Stack Specific
Pop → Delete }

Load → Memory Read } Memory Specific
Store → Memory Write }

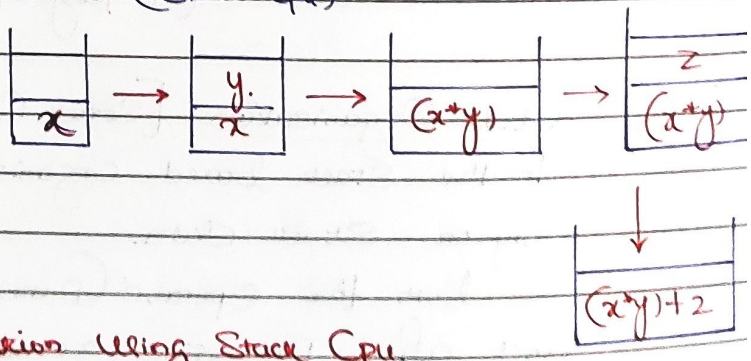
In → I/O Read } I/O Specific.
Out → I/O Write }

Mov → General with all combinations.

eg: (x+y)+2. How many machine instruction are required to execute using stack based organisation (stack-cpu)?

→ I1 : Push x
I2 : Push y
I3 : Mul
I4 : Push 2
I5 : Add.

5 Machine Instruction using Stack Cpu.



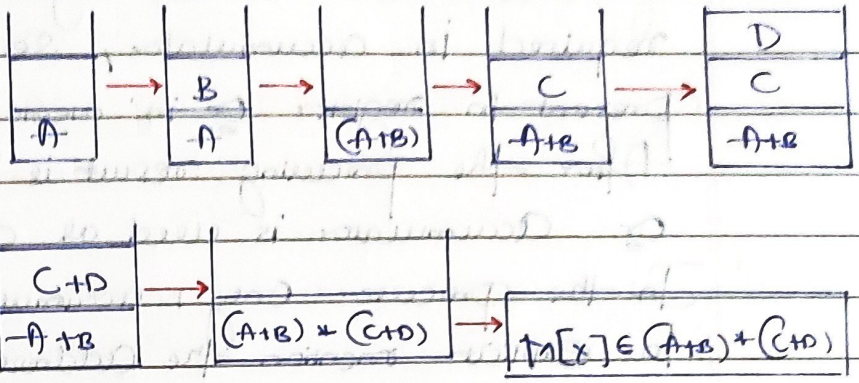
Note: In the Stack-Cpu, only ALU Operation are OAI & data transfer instruction (Push & pop) are not OAI (zero address instruction).

Page No.
Date. / /

Q1. How many machine instructions are required using Stack-Cpu?

$X = (A+B) * (C+D)$

I_1: Push A
I_2: Push B
I_3: Add
I_4: Push C
I_5: Push D
I_6: Add
I_7: Mul
I_8: Pop X.



8 Machine Instructions using Stack-Cpu.

Q2 Consider a 32 bit hypothetical processor which use stack Cpu. which support 1 word Opcode and 24 bit address. following statement is execued on a Stack-Cpu

$X = (A+B) \times (C+D)$

Ans Word Size = 32 bit
Opcode = 1 word = 32 bit ≃ 4 Byte
Address = 24 bit ≃ 3 Byte

Q3 How much memory is required for the program in Byte?

$X = (A+B) * (C+D)$

I_1: Push A → 4B + 3B = 7 Byte
I_2: Push B → 4B + 3B = 7 Byte
I_3: Add → 4B = 4 Byte
I_4: Push C → 4B + 3B = 7 Byte
I_5: Push D → 4B + 3B = 7 Byte
I_6: Add → 4B = 4 Byte
I_7: Mul → 4B = 4 Byte
I_8: Pop X. → 4B + 3B = 7 Byte

47 Byte.

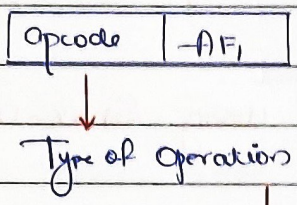
Q. What is the status at the end of the Program execution?
→ Empty, because result is stored in M[X]

Page No.
Date. / /

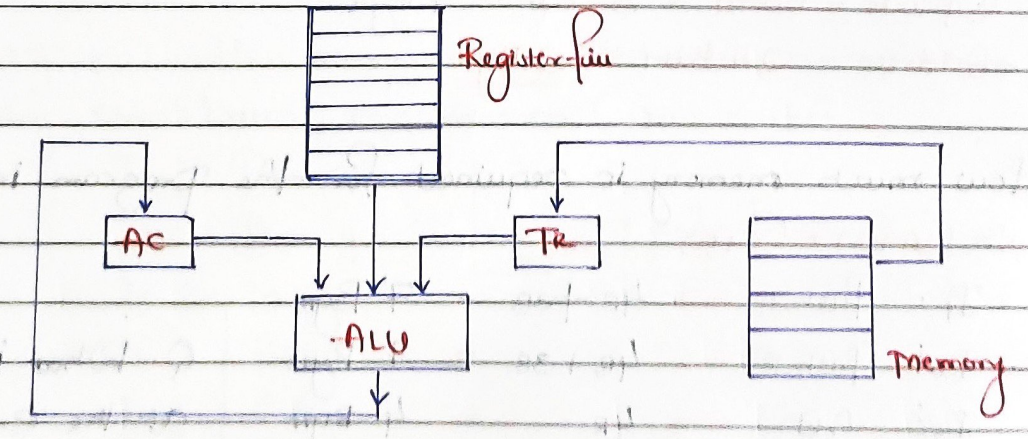
Single Accumulator Organisation:

- In Accumulator based Organisation first ALU Operand are required in accumulator, second ALU Operand (DATA) either present in register or in memory.
- After the processing result is also stored in the accumulator or Accumulator is used as a destination.
- In the processor only 1 accumulator is present, so not needed to explicit mention the address.

Compatible format.



	Destination	Source	Source 2
(i) ALU Operation	AC	AC	Reg/Memory
(ii) Data transfer Operations.	AC	Memory	(Memory Read)
	Memory	AC	(Memory Write)



Register File: The number of registers supported by the processor is denoted (called) register file.

eg: Register file = 16

Contain 16 register file.

Page No.
Date. / /

Q1. (A+B) — A and B are the variable in the memory. How many machine Instructions are required to execute using Ac-Cpu?

Ans. I_1: Load A; $AC \leftarrow M[A]$

I_2: Add B; $AC \leftarrow AC + M[B]$

2 machine Instructions using Ac-Cpu.

Q2 (x*y)+z How many machine instructions required using Ac-cpu?

Ans. I_1: Load x; $AC \leftarrow M[x]$

I_2: Mul Y; $AC \leftarrow AC * M[y]$ $AC \leftarrow x*y$

I_3: Add z; $AC \leftarrow AC + M[z]$ $AC \leftarrow [x*y]+z$

3 Machine Instructions using Ac-Cpu.

Q3 $X = (A+B)*(C+D)$ A, B, C, D and X are variables in the memory.

(i) How many machine instructions required to execute using Ac-Cpu? → 7

(ii) How many Spills (memory Spills) are required? → 1

Ans I_1: Load A; $AC \leftarrow M[A]$

I_2: Add B; $AC \leftarrow AC + M[B]$

I_3: Store T; $M[T] \leftarrow AC$

I_4: Load C; $AC \leftarrow M[C]$

I_5: Add D; $AC \leftarrow AC + M[D]$

I_6: Mul T; $AC \leftarrow AC + M[T]$

I_7: Store X; $M[X] \leftarrow AC$

7 m/c Instructions using Ac-Cpu

$M[T] = (Ans)$

(Memory Spill = 1)

General Register Organisation:

Based on the number of register (Register file size) supported by the processor this architecture is divided into two types:

① Register - memory Reference (Reg-Memory Cpu)

② Register - Memory Register Reference (Reg-Reg Cpu)

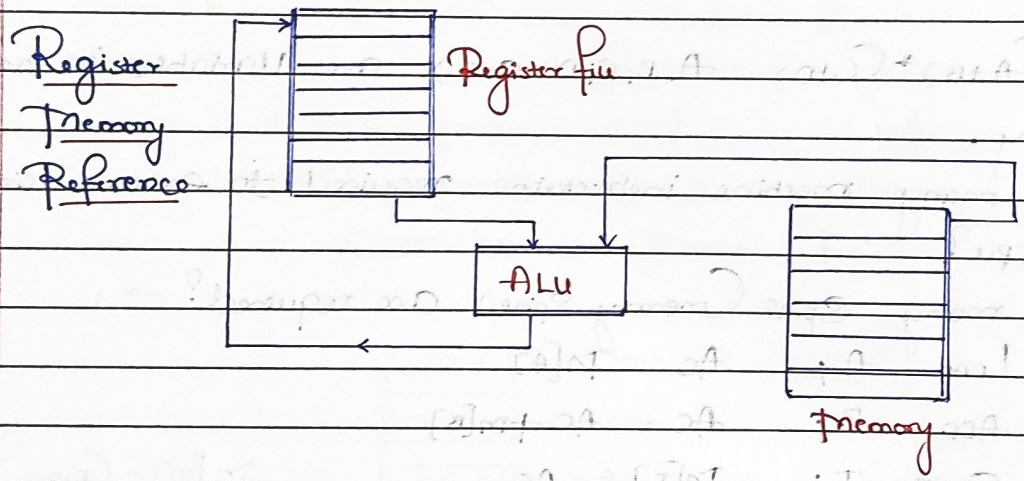
Page No.
Date. 06/09/23

1. **Register – Memory Reference** : This architecture supports less number of registers so register file size is small.

Compatible format:

Opcode	AF1	AF2

	Destination	Source1	Source2	
ALU Operation	Reg(s1)	Reg	Reg/Memory	
Data Transfer Operation	Reg	Mem		MOV
	Memory	Reg		
	Reg	Reg		



eg: (A+B) Using Register memory reference : How many m/c instructions are required?

I1 : MOV R0 A; R0 ← M[A]

I2 : ADD R0 B: R0 ← R0 + M[B]

2 Machine Instructions (using Register Memory Reference).

eg: (x*y)+z ; using Register-Memory reference CPU?

I1 : MOV R0 X; R0 ← M[x]

I2 : MUL R0 Y; R0 ← R0 * M[Y]

I3 : ADD R0 Z; R0 ← R0 + M[Z]

3 Machine Instructions

Page No.
Date. / /

eg: $X = (A+B) * (C+D)$ — A, B, C, D, E are variables in memory, then how many machine instructions are required using Reg-memory reference?

I_1 : MOV R0 A;
I_2 : ADD R0 B;
I_3 : MOV R1 C;
I_4 : ADD R1 D;
I_5 : MUL R0 R1;
I_6 : MOV X R0;

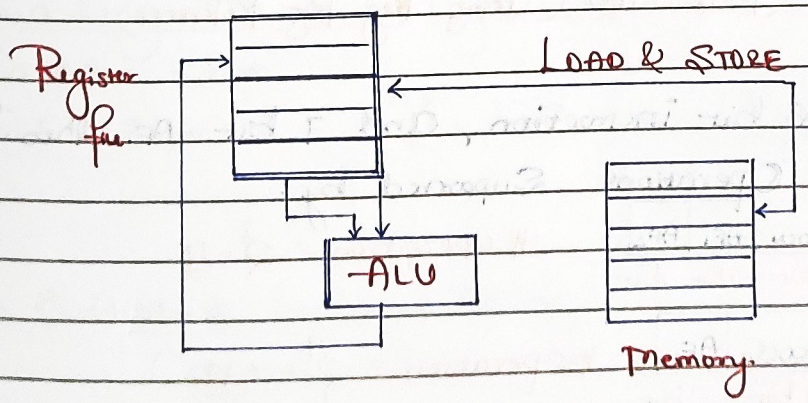
6 M/C Instructions using Reg-Memory reference.

② Register - Register Reference : This architecture : Supports more number of registers.

Compatible format:

Opcode	AF1	AF2	AF3
ALU Operation	Destination Reg	Source1 Reg	Source2 Reg

* In this organisation ALU Operand always required in registers.



eg: (A+B) using Reg-Reg reference CPU?

I_1: LOAD R0 A; R0 ← M[A]
I_2: LOAD R1 B; R1 ← M[B]
I_3: ADD R2 R0 R1; R2 ← R0 + R1

3 Machine Instructions using Reg-Reg Reference.

Page No.
Date. / /

eg: (x*y) + z Using reg-reg reference?

I1 : LOAD R0 X;
I2 : LOAD R1 Y;
I3 : MUL R2 R0 R1;
I4 : LOAD R3 Z;
I5 : ADD R4 R2 R3;

5 M/C Instructions using Reg-Reg reference

eg: X = (A+B) * (C-D) → Here A, B, C, D and X are variable in memory
→ How many machine instructions are required using Reg-Reg CPU?

I1 : LOAD R0 A;
I2 : LOAD R1 B;
I3 : ADD R2 R0 R1;
I4 : LOAD R3 C;
I5 : LOAD R4 D;
I6 : ADD R5 R3 R4
I7 : MUL R6 R2 R5
I8 : STORE X R6

→ How many minimum no. of registers required?
3 Minimum Reg required.

8 M/C Instructions (Using Reg-Reg Reference)

Q1. Consider a 16 bit instruction, and 7 bit OF → How how many total no. of Operations Supported by:

2AF → | Opcode | AF1 | AF2 | → # operations = $2^9 = 4$
9 bit 7 bit 7 bit.

1AF → | Opcode | AF1 | → # Operations = $2^7 = 512$
9 bit 7 bit

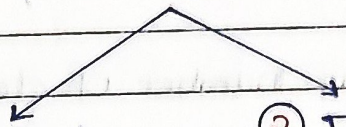
0AF → | Opcode | → # operations = $2^{16} = 64K$
16 bit.

→ 3AF / 3AI | Opcode | AF1 | AF2 | AF3 | → (It can support upto 3AF / 3AI it can support 2AF, 1AF, 0AF)

Page No.
Date: / /

Expand Opcode Technique:

Expand opcode length is required in the fixed length instruction supported CPU design to implement the various instructions with different formulae (formats).



① Variable length instructions (Fixed length opcode)

② Fixed length instructions. (Variable length opcode)

Variable Length Instruction Supported CPU Design

Opcode = 8 bit

Address field = 8 bit

(i) 1 Address Instruction design:

Opcode	Address	= 16 bit
8 bit	8 bit	

(ii) 0 Address Instruction Design:

Opcode	= 8 bit. (One need of expand opcode technique)
8 bit	

Fixed Length Instruction Cpu Supported CPU Design

Opcode = 8 bit

A.F = 8 bit

(i) 1 Address Instruction Design:

Opcode	Address	= 16 bit.
8 bit	8 bit	

(ii) 0 Address Instruction Design:

Opcode	= 16 bit. (There Expand Opcode technique required)
16 bit	

Expand Opcode Technique: We start from primitive instructions.

① Primitive Instruction (lowest bit in opcode)

② Derived Instruction (higher opcode bit)

③ Further derived instruction (highest opcode bit).

Page No.
Date. / /

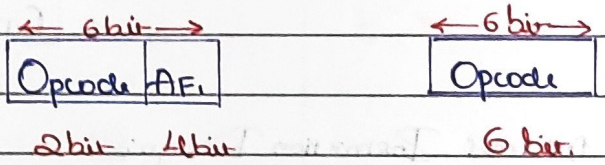
Expand Opcode Technique:

Step 1: Identify the primitive instruction in the CPU.
Step 2: Calculate the total number of possible operation.
Step 3: Identify the free opcode after allocating the existing instruction.
Step 4: Calculate the number of derived instruction possible by multiply

Free opcode $\times 2^{\text{Increment bit in opcode}}$

Q1. Consider a processor which support 6 bit instruction and 4 bit address field. If there exist 2 one address instruction then how many 0 address instruction can be formulated?

Ans Instruction length = 6 bit
AF = 4 bit



Step 1: Primitive Inst. Derived Inst.

Step 2: Total no. of operation/instruction in AF = $2^2 = 4$

Step 3: Number of free opcode after allocating AF = 4 - 2 = 2 Free

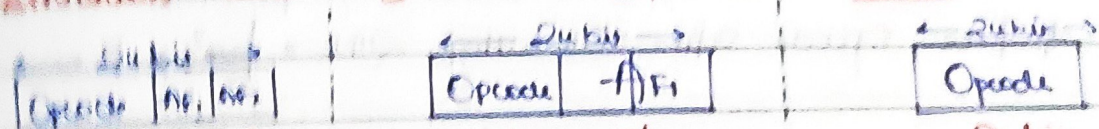
Step 4: Total # operation in OAF = Free opcode $\times 2^{\text{Inc. bit in opcod}}$
$= 2 \times 2^{6-2}$
$= 32$

Q2. Consider a processor which contain 8 bit word and 256 word memory. It support 3 word instruction. If there exist 254 2-address instruction and 256 1-address instruction then how many 0-address instruction can be formulated?

Ans 1 Word = 8 bit = 1 Byte
Memory = 256 Word = 256 × 1 Byte = 256 Byte

Page No.
Date / /

Memory = 256 Byte [2^8 B] → Add = 8 bit
Instruction size = 8 word → 8 × 8 bit = 24 bit



Primitive Inst. | Derived Instr. | Further Derived Inst

(I) Primitive:
- Total no of operation in 2AF (prime inst.) = 2^8 = 256 Oper. Inst.
Given 2AF/2AI = 254
Free Opcode after allocating 2AF = 256 − 254 = 2

(II) Derived:
- Total # operations = Free opcode × 2^(Increment bit in opcode)
In 1AF = $2 \times 2^{16-8}$
= 512 Operations
Given 1AI (AF) = 256
Free Opcode after allocating 1AF = 512 − 256 = 256

(III) Further Derived:
- Total # Operations = Free opcode × 2^(Increment bit in opcode)
in 0AF = $256 \times 2^{24-16}$
= 256×2^8
= 2^{16} or 64K

(Ex) Consider a processor with 16 bit instruction. Processor has 16 registers and supports 2 address instruction and 1 address instructions. If processor supports 256 1 address instruction then number of 2 address instruction are.

Sol: No. of registers = 16 = 4 bit, Instruction size = 16 bit

2AF: | OP (8 bit) | Add1 (4) | Add2 (4) |
1AF: | OP (12) | Add (4) |

Page No.
Date. / /

Primitive: Total no. of operation in 2AF = 2^8

Assume 2AI/2AF Instruction/Operation = x

Free Opcode after allocating 2AI = $[2^8 - x]$

Operation in 1AF/1AI = Free Opcode × 2 (Increment bit in opcode)

$= (2^8 - x) \times 2^{12-8}$

$2^8 = (2^8 - x) \times 2^4 \Rightarrow 2^8 / 2^4 = 2^8 - x$

$2^4 = 2^8 - x$

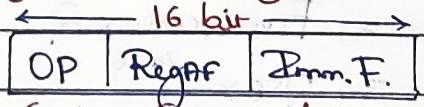
$16 = 256 - x$

$x = 256 - 16 \Rightarrow 240$

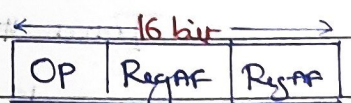
Q4. A processor has 64 registers and uses 16 bit instruction format. It has two types of instructions: I-type and R-type. Each I-type instruction contains an opcode, a register name, and a 4 bit immediate value. Each R-type instruction contains an opcode and two register names. If there are 8 distinct I-type opcodes, then the maximum number of distinct R-type opcodes is 14

Ans 64 Register ⇒ Reg AF = 6 bit. Instruction Size = 16 bit

I type:



R type:



16 − (6+4) = 6 bit

16 − (6+6) = 4 bit

Total # Operation in R-type = 2^4

Assume R-type instruction = x

Number of Free Opcode after allocating R type = $2^4 - x$.

operations in I-type = $(2^4 - x) \times 2^{6-4} \Rightarrow 8 = (2^4 - x) \times 2^2$

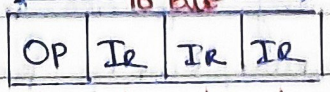
$8 = (2^4 - x) \times 4 \Rightarrow x = 16 - 2 \Rightarrow 14$

Page No.
Date: / /

Qs. A processor has 16 registers (R0, R1 ... R15) and 64 floating point registers (F0 ... F63). It uses a 2 byte instruction format. There are four categories of instructions: Type-1, Type-2, Type-3, Type-4.
Type-1 Category consists of 4 instructions, each with three integer operands (3Rs) Type-2 Category consist of eight instructions, each with two floating point register operands (2Fs). Type-3 Category consist of fourteen instructions, each with one integer operand and one floating point register operand (1R + 1F). Type-4 Category consists of N-instructions, each with a floating point register operand (Fs). The maximum value of N is ____

Ans 16 Registers ⇒ IR Registers = 4bit
64 Floating registers: FR = 6 bit
Instructions = 2 Byte = 2×8 = 16 bit

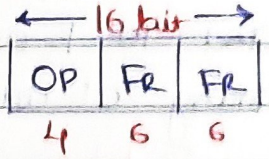
Type-1:



Primitive:
Total # operations = $2^4 = 16$
Free in Type-1 = 16 − 4 = 12.

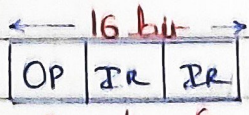
Type-2:

Primitive:



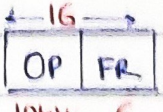
Total # Operations = $12 \times 2^{4-4}$
= 12
Given = 8
Free = 12 − 8 = 4

Type-3:



Derived:
Operations = $4 \times 2^{6-4}$
= $4 \times 2^2 = 16$
Given = 14
Free = 16 − 14 = 02

Type 4: Further Derived:

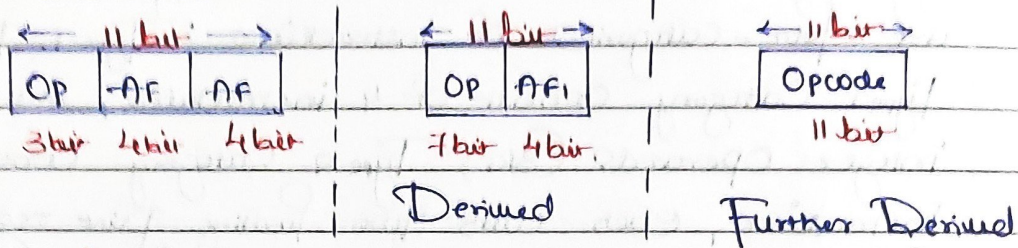


Operations = $2 \times 2^{10-6} = 2 \times 2^4$
= 32

Q6 Consider a processor with 11 bit instruction. The size of address field is 4 bit. The computer uses expanding opcode technique and has '5' two (2) address instructions and '32' one (1) address

instruction. Then the number of zero address instruction it can support ie

Ans



$2^n = 5 \times 2^4 \times 2^4 + 32 \times 2^4 + x$

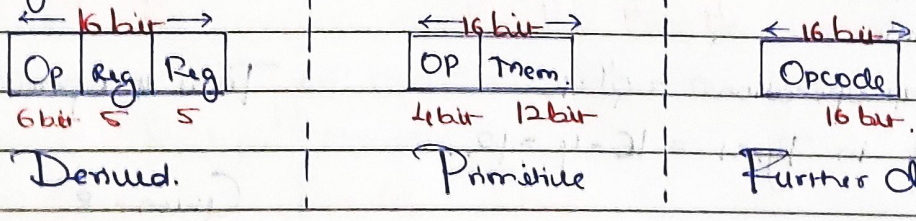
$2^n = 5 \times 2^8 + 2^9 + x$

$2^n = 2^8 (5+2) + x$

$x = 256$

Q7. Consider a 16 bit hypothetical processor which support 1 word long instruction. Processor has 30 registers and 4KB of memory size. If there exist '1' 2 address register reference instruction and '10' 1 address memory reference instruction. Then how many '0' address instruction can be formulated/supported?

Ans



Primitive: Total # Operations $= 2^4 = 16$

Given memory reference = 10

Free = 16 - 10 = 6

Derived:

Total # operations $= 6 \times 2^{6-4} = 6 \times 2^4 = 24$

Given = 11

Free Opcode = 24 - 11 = 13.

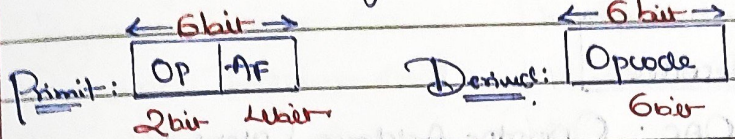
One = $13 \times 2^{6-6}$

$= 13 \times 2^{10} = 13K$

Page No.
Date. / /

Q8. Consider a processor which support 6 bit instructions and 4 bit address field. If there exist 2 one address instructions then how many 0 address instructions can be formulated?

Ans.



operation in AF/AF = $2^2 = 4$

Given AF = 2

Free = 4 - 2 = 2

Total # operations in OAF

$= 2 \times 2^{6-2}$

$= 2 \times 2^4 = 32$

Addressing Modes

* Addressing is a technique used to calculate the effective address (EA)

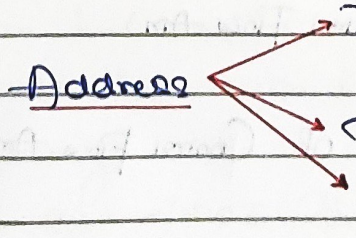
Or

* Addressing mode show the way where the required object is present

Or

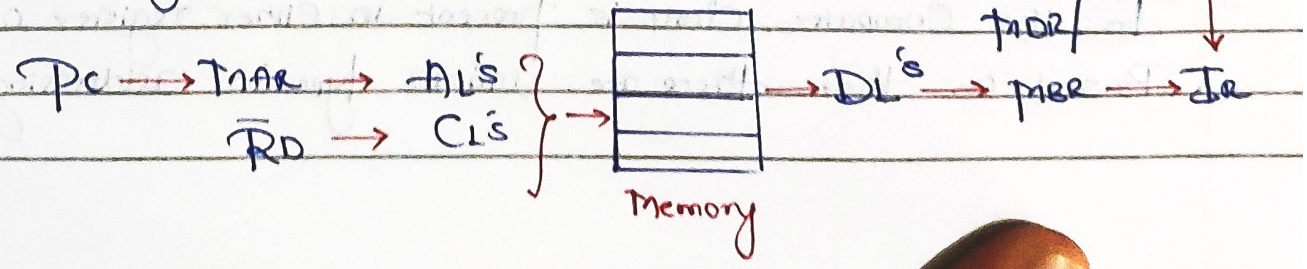
* Addressing mode show the way how to get operand.

* Effective Address [EA] is the actual address of the object.
* Object may be instruction or data
* The output of addressing mode is effective address



When Am?

Fetch Cycle: Memory to CPU [IR]



Page No.
Date. / /

Fetch Cycle:

IR | Opcode | mode | AF

Execute Cycle

Decoder

→ Decode — OAC: Operator Address Calculation

→ Execute — OF: Operand Fetch

DP & Rs: Data Processing & result Storage

Why Am?

Opcode | Address

→ Register (Direct or Indirect)
→ Memory (Direct or Indirect)
→ Constant (Value)

Opcode | mode field | AF

→ Immediate Constant
→ Register (Direct / Indirect)
→ Memory (Direct / Indirect)

* mode field helps you to get Operand.

Addressing Modes: Addressing mode show the way where the required object is present or location of required object.

1. Data Centric Am (Sequential Control Flow Am)
   → Focus on 'DATA'
2. Instruction Centric Am (Transfer of Control Flow Am)
   → Focus on 'Instruction'

* Am in the instruction is implemented with the help of 'mode field'.
* In the Computer, data is present in either register or memory. Based on that there are various type in addressing mode.

Page No.
Date: 21/01/23

1. Immediate Am
2. Direct / Absolute Am
3. Memory Indirect Am
4. Register Direct Am
5. Register Indirect Am
6. PC Relative Am
7. Based Register Am
8. Indexed Register Am
9. Implied / Implicit Am
10. Auto Decrement Am
11. Auto Increment Am
12.

Symbols of Am.

I or # ⟶ Immediate Am
[ ] ⟶ Direct Am
[( )] or @ ⟶ Indirect Am
Reg name ⟶ Register Am
Index Reg Name ⟶ Indexed Reg. Am

① Immediate Am: In this addressing mode Operand are present in the address field of the instruction or In this Am operand are present in the instruction itself.

Ri: | Opcode | Addrss | → 'Data'

Immediate Am are used to access the Constant and initialise the register with Constant value

Immediate Am has some limitations:

* Immediate Am can never be used as a destination address. It can be used only as source address, because destination must be required storage and constant does not have any storage.

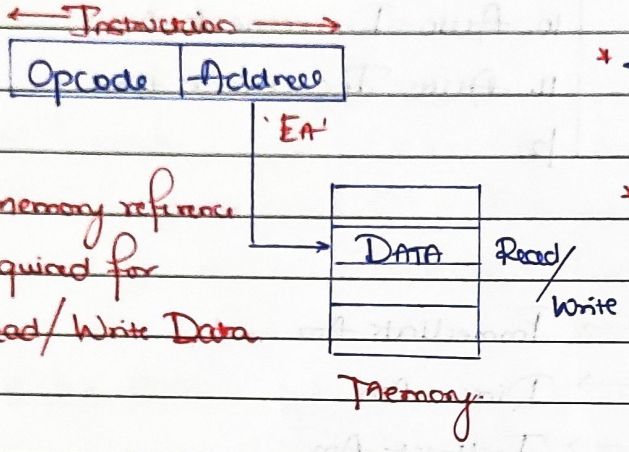
eg: ADDI R1 5100 ✓
R1 ← R1 + 500

eg: ADDI 500 R1 ✗
500 ← 500 + R1

* The range of constant is limited by address field.
n bit Immediate field Unsigned range = 0 to $2^n - 1$
Signed range = -2^{n-1} to $+2^{n-1} - 1$.

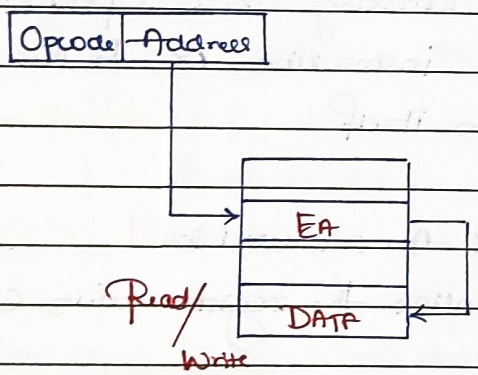
Page No.
Date. / /

(2) Memory Direct or Absolute Am : In this addressing mode operands are present in the memory and address field of the instruction contain the effective address.



* This Am is used to access the variable
* 1 Memory reference required for read/write data

(3) Memory Indirect Am : In this Am operand are present in the memory, effective address is also present in memory, instruction contain address of effective address.



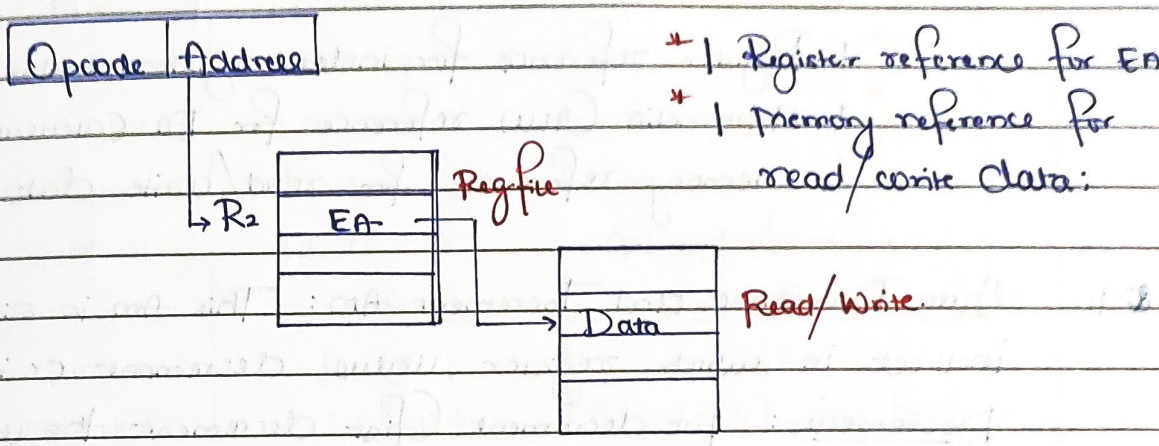
* Indirect Am are used to access the pointers
* 2 Memory reference is required in memory indirect Am.

(4) Register Direct Am : This Am is similar to memory direct Am but in this Am operand are present in the registers instead of memory. Or

In this Am operand are present in the register that register address (register name) maintained in the address field of the instruction.

* 1 Register reference required for read/write data.

⑤ Register Indirect Am: In this Am operand are present in the memory and effective address is stored in register.



* 1 Register reference for EA
* 1 Memory reference for read/write data.

Q. Why register indirect Am used instead of memory indirect Am?

Ans. * To shorten the instruction length (size)
* Accessing the register is very fast compared to memory.

eg: If memory = 16 GB then memory AF = 34 bit then instruction size large but in the processor we have 32 or 64 register so we need 5 or 6 bit respectively.
So instruction size reduced and faster access.

Displacement Am

⑥ PC Relative Am.

EA = Current + A.F
PC value (offset)

or

EA = Current + Relative
PC value value

⑦ Base Register Am.

EA = Base Register
Value
+
AF (offset)

⑧ Index Register Am.

EA = Index reg value
+
Offset (AF)

↓

Array implementation

Page No.
Date. / /

Index Am → (Array implementation)

EA = Index Reg value + AF (Offset)

* 1 Register reference for index register value
* 1 Arithmetic (ALU) reference for EA calculation
* 1 memory reference for read/write data.

(9), (10) Auto Decrement and Increment Am: This Am is similar to register indirect in which register value decrement or increment.

Decrement: Pre decrement (first decrement the value (in register) then access the data)

Increment: Post increment (first access the value/data then increment in register value).

(11) Implied/Implicit Am: In this Am operand (data info) are present in the opcode itself.

OPCODE → Type of Operation, Data information

* Stack based organisation used in implicit/implied Am

Note:
Constant → Immediate Am
Variable → Direct Am
Pointer → Indirect Am
Array → Index Am
Relocation → Base Reg Am

Q1. In which of the following addressing modes, operand is Not a part of the instruction?

Ans Direct
Indirect
Register.

Page No.
Date. 28/09/23

Eight addressing modes for the load instructions:

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate Operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Auto increment.	LD (R1)+	AC ← M[R1] R1 ← R1+1

Q2. Consider a 16 bit hypothetical Processor which Support 1 address instruction design with various addressing modes. It contain 8 bit opcode. It supports 1 word instructions stored in the memory with a starting address of $(745)_{10}$ onwards. Address field value of the instructions is 222. Register R1 contain 111 memory content of [222] is 155. Which of the following is/are correct about effective address of various Addressing Mode (Am)?

I In the immediate Am effective Address is 745

II In the immediate Am effective Address is 746

III In the memory indirect Am effective address is 155

IV In the index Am effective address is 333 (R1 as index register).

Ans 16 bit Processor

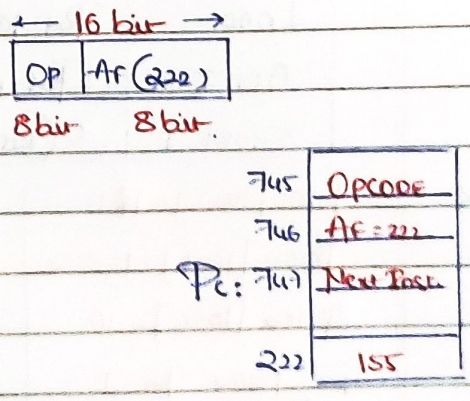
Instruction Size = 1 word = 16 bit

Register (R_1) = 111.

Immediate Am: EA = 746

Memory indirect EA = 155

Index Am EA = Index Reg + AF = 111 + 222 = 333



Page No.
Date. / /

Q3. Match the following pairs:

X. Indirect addressing	1. Loops
Y. Immediate addressing	2. Pointers
Z. Auto decrement addressing	3. Constants

Q4. If we use internal data forwarding to speed up the performance of a CPU (R_1, R_2 and R_3 are registers and $M[100]$ is a memory reference), then the sequence of operations

$R_1 \rightarrow M[100]$

$M[100] \rightarrow R_2$

$M[100] \rightarrow R_3$ Can be replaced by

Ans $R_1 \rightarrow R_2$ R_2, R_3 and $M[100]$ Correct updation by R_1 Content

$R_1 \rightarrow R_3$

$R_1 \rightarrow M[100]$

Q5. Match the following:

A. A[I] = B[j];	1. Indirect addressing
B. while [*A++];	2. Indexed addressing
C. int temp = *x;	3. Auto increment

Q6. The memory locations 1000, 1001, and 1020 have data values 18, 1 and 16 respectively before the following program is executed.

MOV	R5, 1	Move immediate
LOAD	Rd, 1000 (R5)	Load from memory
ADDI	Rd, 1000	Add immediate
STORE I	0(Rd), 20	Store immediate

Ans MOV [1000] = 18

MOV [1001] = 1

MOV [1020] = 16

I_1 : MOV R5, 1

R5 = 1

I_2 : Load Rd, 1000 (R5)

Rd ← M[1000 + R5]

Rd ← M[1000 + 1]

Rd = 1

I_3 : ADDI Rd, 1000

Rd ← Rd + 1000

Rd = 1001

Page No.
Date. / /

I4 = Store I, 0(Rd), 20
M[0+Rd] ← 20 ∴ Memory location 1001 has value 20
M[1001] ← 20

Q7 The absolute addressing mode:
Ans The address of the operand is inside the instruction.

Q8 A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter?

Ans 24 bit instruction = 3 Byte

$300 + 3x = 600$ ∴ 600

$300x = 3x$,

$x = \frac{300}{3} = 100$

Q Why Auto increment, decrement is used?
Ans When we want to access or perform some operation in multiple locations (assume 100 location) this different instructions are required.
But with the help of auto decrement or auto increment Am it is possible with only one instruction we can perform same operation on multiple locations.

Note:
* Index addressing mode: This Am is used to access specific (random) element of the array.

A[n] = Starting address + n * Size of datatype → Index value
(base address)

* Index value stored in index register
* Starting address present in the address field of the instruction.

EA = Index Rg Value + AF

Page No.
Date. / /

Index Register

(1.) At the CPU design time one special register made as a index register then in this they implicit access the index value from the index register.
(Wastage of one register because index register use very less)

(2.) In most of the cases one general purpose register made (designated) as index register. & mentioned that register name as index register eg: R_0 as a index register. We can use that register as general purpose register also.

Index Addressing Modes:

1. Index Am
2. Index based Am / Base Index Am
3. Base index with displacement Am
4. Index indirect Am
5. Auto indexed Am

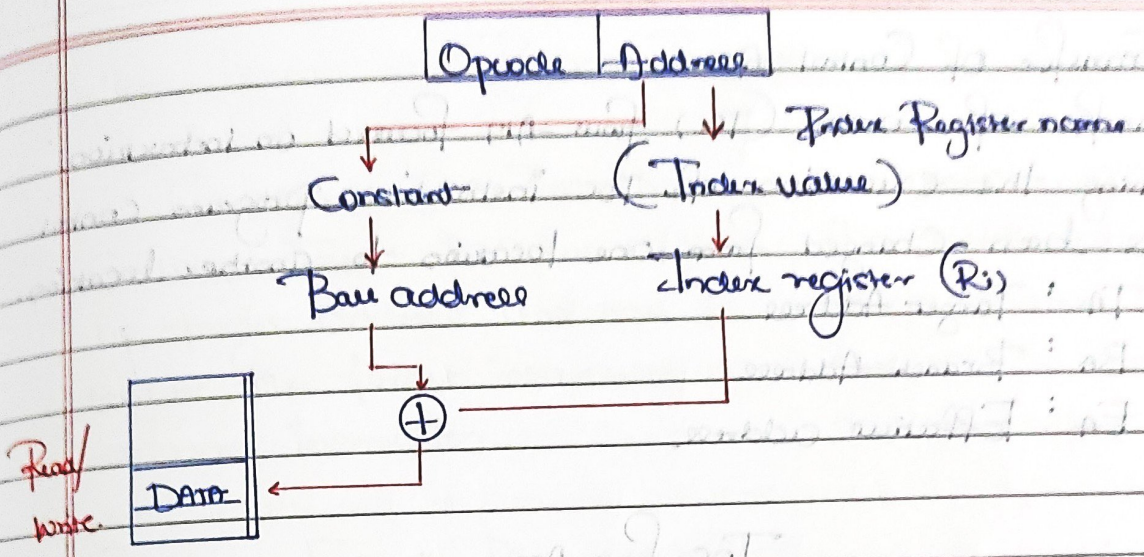
(1.) **Indexed Am**: Index Am is used to access the random element of array

Base Address	Index Value.

Base Address → Starting address of the instruction → Present in the address field of the instruction as a constant.

Index Value. → Position of array element → Index Register

* 1 Reg reference for index value
* 1 ALU reference for EA & 1 Memory-ref for read data.



② Base Index Am: Sometime CPU contain Seperate base register and index register to store the base value and index value respectively.

eg: 8086 up

Bx reg ⇒ Base register

SI/DI : Index register (Source index / destination index)

MOV Bx, #1000

MOV SI, #20

MOV Ax [Bx][SI]

Ax ← [Bx+SI]

Ax ← M[1000+20]

③ Base Index with displacement:

MOV Ax 50([Bx][SI])

Ax ← M[(Bx)+(SI)+50]

④ Indirect Index Am.

1 Reg ref for index value

1 ALU ref for EA

1 Memory reference for Read Data.