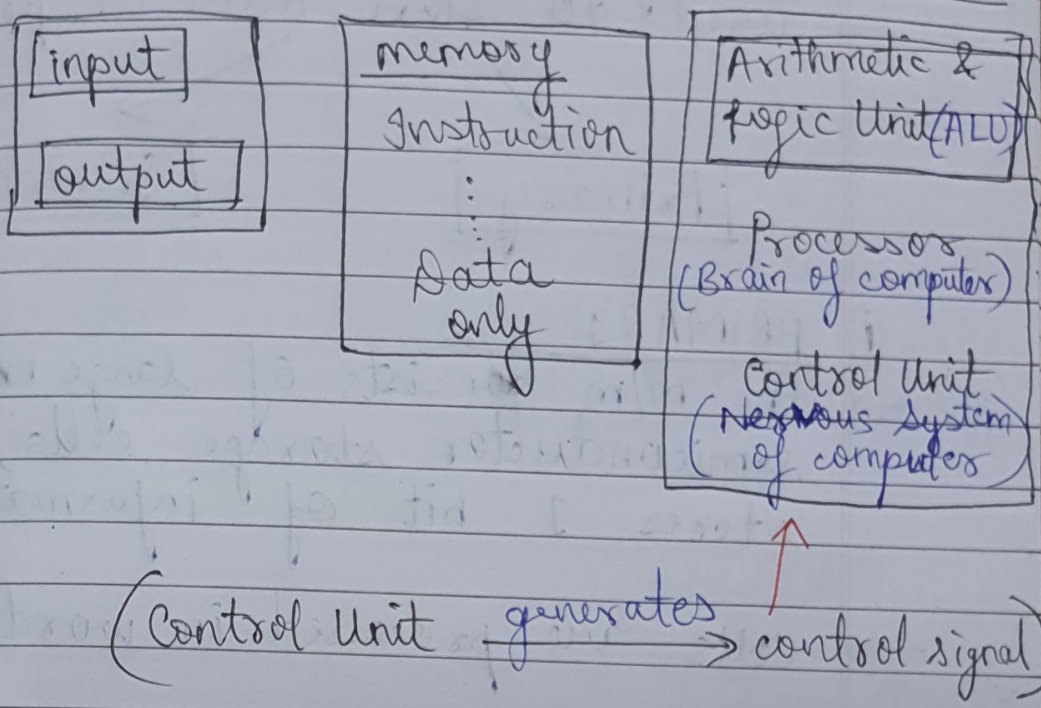
UNIT-2

Date 1/8/23
Page 1

FUNCTIONAL UNITS OF COMPUTER



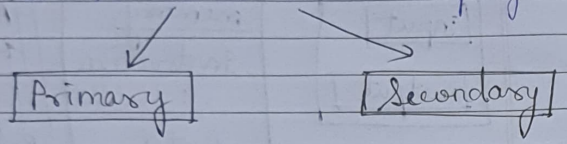
(Control Unit → generates → control signal)

- Examples of I/P unit are :→
  - Keyboard
  - Mouse
  - Joystick

[Keyboard] : When a key is pressed, the corresponding letter/digit/symbol is automatically translated into it's binary code & transmitted over the cable to either memory or processor

Memory Unit

→ used to store data & program



① PRIMARY

→ The m/m consists of large no. of semiconductor storage cells, each stores 1 bit of information.

→ Cells are processed in words.

→ fast m/m operated at electronic speed.

→ The m/m in which any location is reached in a short, fixed amt. of time is c/a Random Access Memory (RAM)

⊙ [CACHE M/M]: Small, fast RAM units are called "cache m/m"

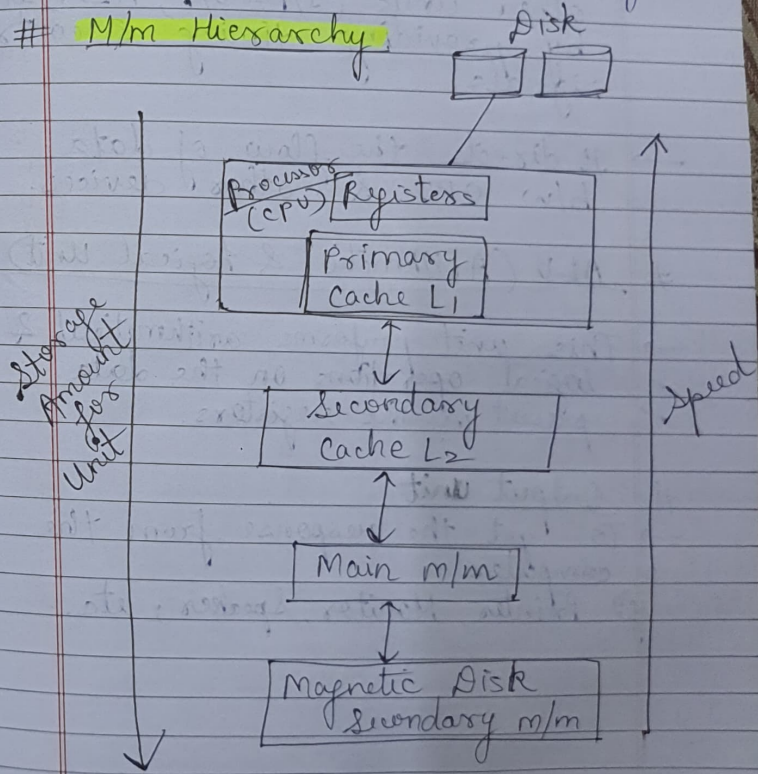
~~⊙~~ largest & slowest unit is main m/m.

② SECONDARY

→ The program & data is stored in secondary m/m.

→ When we need to execute that program, the program transfers to m/m & the corresponding data is transferred to CPU registers.

M/m Hierarchy



Control Unit

→ Instructs the m/m, logical unit, I/P-O/P unit of computer on how to respond to the program's instruction.

→ It directs the operations of other units (I/P-O/P, ALU, etc) by providing timing & control signals.

→ It directs the flow of data b/w CPU & other devices.

ALU (Arithmetic & Logical Unit)

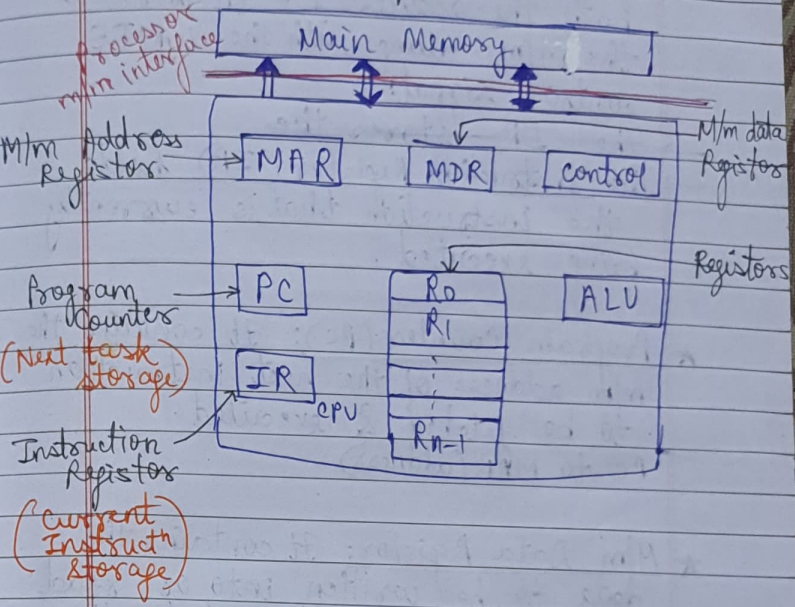
→ This unit performs arithmetical & logical operations on the data present in the registors.

Output Unit

→ To get the response from the computer.

eg) Printer, Monitor, Speaker, etc.

Operational Concepts



→ The Processor m/m interface manages the transfer of data b/w the main m/m & the processor.

★ M/m read: The interface sends the address of the word to the m/m along with the READ control signal.

→ The interface waits for the word to be retrieved & then transfers it to appropriate CPU registor.

★ M/m write: The interface transfers both the address & the word to the m/m along with the WRITE control signal.

~~→ The IR stores the~~

→ The Instruction Register (IR) holds the instruction that is currently being executed.

★ Program Counter (PC): It contains the m/m address of the next instruction to be fetched & executed.
PC to MAR (address)

★ M/m Data Register: It contains the data to be written into or read out from the addressed location. (received data) send to IR.

★ MAR: contains the ~~data~~ address of the data to be accessed.
(MAR to m/m via interface)

→ The execution of a program may be preempted if some device require urgent servicing.
To deal with these situations immediately, the normal execution of the program is interrupted, for this the device raise an interrupt signal.

→ An INTERRUPT is a request from an I/P / O/P device for service by the processor. The processor provides the requested service by executing an appropriate ISR (Interrupt service Routine)

→ When ISR works, content of PC, general purpose register & some control info is stored in m/m. When ISR complete, the internal state of the processor is restored so that the interrupted program may continue.

★ Instruct^{n} execut^{n} steps :→

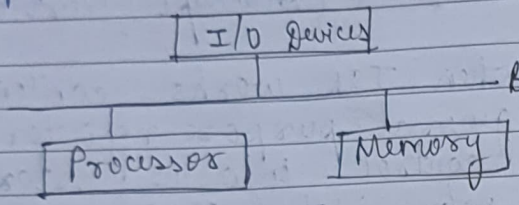
PC
↓
MAR
↓
MM ← CU
↓
MDR
↓
IR
↓
MAR
↓
MM
↓
MDR
↓
PR
↓
operation
↓
ALU

Address to locate data
read control signal

CU ⇒ send, read control signal.

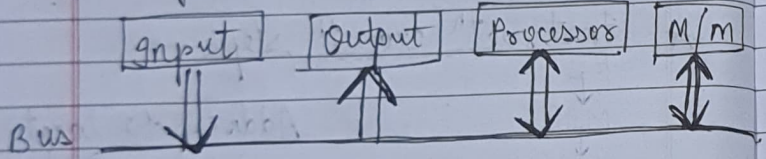
Bus Structure

→ Bus is an interconnection network used to transfer data among processor, m/m, I/P - O/P devices.



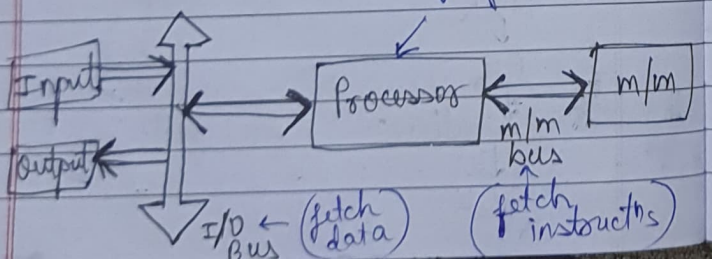
Types of Buses in Computer

(I) Single Bus Structure



→ One common bus is used to communicate b/w peripherals & micro processor.

(II) Double Bus structure



→ One bus is used to fetch instructns & other bus is used to fetch data.

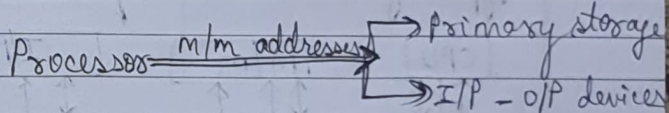
(III) 3- bus structure

→ we have 3 buses:-

1. Address Bus (to send address)
2. Data Bus (to send data)
3. Control Bus (to send control info.)

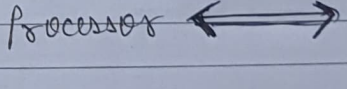
① ADDRESS BUS: It carries m/m addresses from the processor to other components such as primary storage & I/P - O/P devices.

→ Uni- directional.



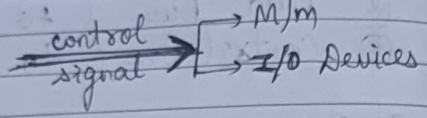
② DATA BUS: It carries the data b/w the processor & other components (I/O devices & m/m).

→ Bi-directional



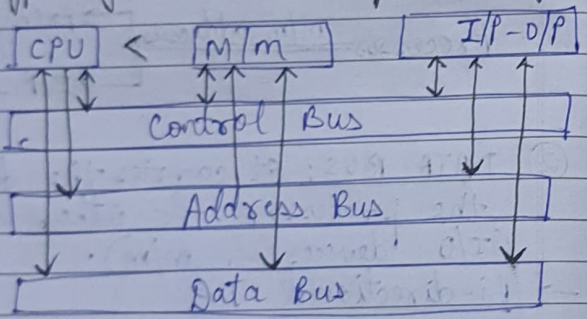
Read → m/m to processor
Write → Processor to m/m

③ CONTROL BUS: It carries control signals (generated from CU present in the processor) from processor to other components (m/m & I/O devices)
→ also carries the clock pulses.
→ Uni-directional



eg) M/m READ, M/m WRITE,
I/O READ, I/O WRITE

▶ Types of Buses in Computer (Diagram)

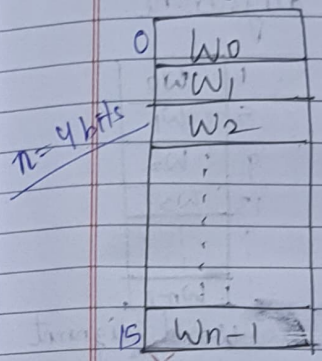


Memory Address & Operation

→ A word consists of 'n' no. of bits

1 word = n bits = $\frac{n}{8}$ bytes

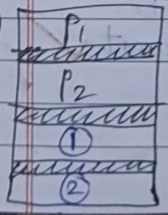
★ Storage of 'n' words in m/m



- 'k' bit address to access m/m = 2^k
- Range = 0 to $2^k - 1$

★ COMPACTION → Used space ek jagah chla jaata hai aur unused space ek jagah (free space इकट्ठा हो जाता है)

eg)

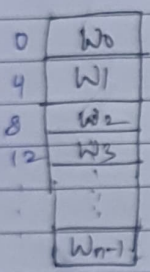


requires ① & ② free space → for it's storage, so (used space) gets one side aur free space ek saath aa jaata hai to store P3 together

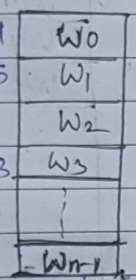
Byte Addressibility

→ To assign address to each byte
(Hum bits को address देने को बजाय byte को देते है)

Word Alignment → If address is a multiple of no. of bits in words stored in m/m, then it is c/a word alignment.

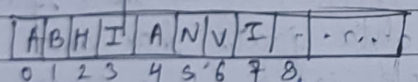


Word alignment ✓



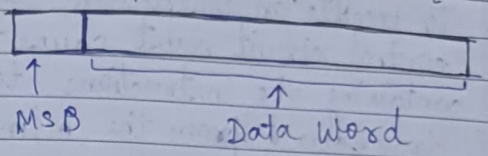
Word alignment ✗

ABHI
ANVI
DENO
MANU
RAMU
(No. of bits in all these words = 4)

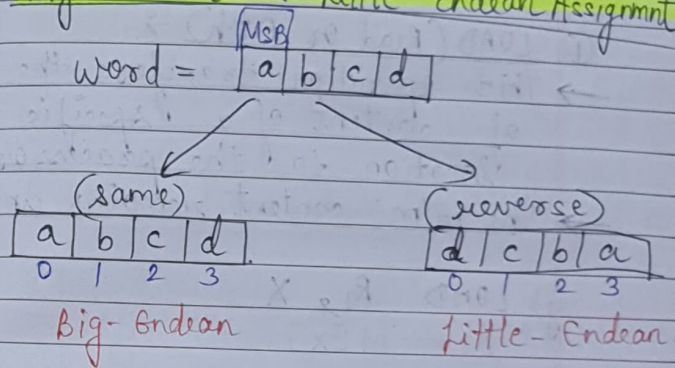


(multiple of 4)
Word alignment ✓

Word Representation in m/m



Big Endean & Little Endean Assignment



- **Big Endean Assignment:** Lower byte address is used to represent the MSB of the word.
- **Little Endean Assignment:** Highest byte address is used to represent the MSB of the word.

Date 9/8/23
Page 4

Memory Operations

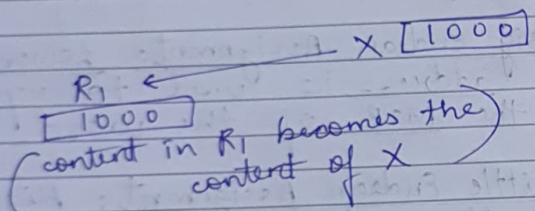
→ To execute an instruction, the processor control circuit must cause the word containing the instructions to be transferred from the m/m to the processor.

① LOAD (Read or Fetch) →

→ This operation transfers the copy of content of a specific m/m location to the processor.

→ The m/m content remains unchanged.

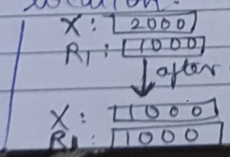
eg) LOAD R_1, X
$R_1 \leftarrow M[X]$



② STORE (Write) →

→ Transfers an item of info. from the m/m to a specified m/m location, destroying former content of that location.

eg) STORE X, R_1
$M[X] \leftarrow R_1$



Date 22/8/23
Page 5

Assemblers: used to convert assembly lang. code to m/c code.

Assembly lang. code → Assembler → M/c lang. code

(11:)

Assembly Language

→ M/c understands 0 and 1 BUT we can't write code in 0 and 1 form, rather we use normal words like MOV, ADD, LOAD, STORE

→ When we write programs for a specific computer, the normal words needs to be replaced by acronyms c/a MNEMONICS.

- MOVE → MOV
- Increment → INC

→ A complete set of symbolic names & tools for their use constitutes a programming lang. c/a ASSEMBLY language.

→ Each mnemonic represents the binary pattern or OP code for the operation performed by the instruction. (Operation code)

ADD R_1, R_2 → Instruction

000

Notations used to represent instructions

① Register Transfer Notation (RTN)

→ In this notation, locations are identified by symbolic names which represent their hardware binary address.

eg → LOC, Place, X, DATAIN, DATAOUT, R_1, R_2, etc
(registers)

→ Content of a location is denoted by placing a square bracket around the name of the location.

eg) R_1 mein LOC ka data transfer krna.

$R_1 \leftarrow [LOC]$

R_1 : address, [LOC] : content

eg) R_1 mein R_2 ka data transfer krna

$R_1 \leftarrow [R_2]$

eg) R_1 mein (R_1, R_2) jodna ka data transfer krna

$R_1 \leftarrow [R_1] + [R_2]$ → (A = 10, B = 20, A = A + B)

② Assembly Language Notation

eg) ADD R_1, R_2, R_3 ← CPU registers

ADD : mnemonic

meaning

$R_1 \leftarrow R_2 + R_3$ / $R_3 \leftarrow R_1 + R_2$

use only one of these 2 notations

(R_1 → destination; R_2, R_3 → operands)

eg) ADD R_1, X

meaning

$R_1 \leftarrow R_1 + M[X]$

M/m location

eg) PUSH X

↳ Push the word at location X on the top of the stack

Types of Instructions

→ There are various types of instructions Based on no. of operand addresses we have 4 types of Instructions

1) 3-Address Instruction
2) 2-Address Instruction
3) 1-Address Instruction
4) 0-Address Instruction

① 3-Address Instruction

Opcode	Destination	Source1	Source2
ADD	C	A	B

↑
(C = A+B)

→ 3 addresses are involved.

• If 'k' bits are used to specify a m/m address then instruction needs 3*k bits to specify the operand addresses.

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

```
ADD R1, A, B
ADD R2, C, D
MUL X, R1, R2
```


★ RTN Notation

$R_1 \leftarrow M[A] + M[B]$
$R_2 \leftarrow M[C] + M[D]$
$M[X] \leftarrow R_1 * R_2$

- Advantages → Short program
- Disadvantages → M/m size is more

② 2-Address Instruction

MOV	R1	A
Opcode	Destination	Source 1

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

(R1 & R2 are empty in the starting)

MOV R1, A	R1 = R1 + M[A]
~~MOV~~	
ADD R1, B	R1 = R1 + M[B] = M[A]+M[B]
MOV R2, C	R2 = R2 + M[C]
ADD R2, D	R2 = R2 + M[D] = M[C]+M[D]
MUL R1, R2	R1 = R1 * R2 = (A+B) * (C+D)
MOV X, R1	X = R1 = (A+B) * (C+D)

- MOV → moves or transfers the operands to & from m/m & processor registers.

→ The first symbol is assumed to be both source & destination where the result of the operation is to be transferred.

'n' address instruct^{n} → 'n' addresses used in instruct^{n}

Date 25/8/23
Page 20

③ 1- Address Instruction

(ACCUMULATOR Register)

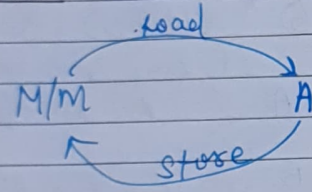
→ use an implied accumulator register (AC) for all data manipulation.

→ We use "LOAD" & "STORE" operat^{n}s.

→ Here, AC works as 2nd register

(i) LOAD → Loads data from m/m to A.C.

(ii) STORE → Store A.C. result in m/m.



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

LOAD A	AC ← M[A]
ADD B	AC ← AC + M[B]
STORE T	M[T] ← AC
LOAD C	AC ← M[C]
ADD D	AC ← AC + M[D]
MUL T	AC ← AC * M[T]
STORE X	M[X] ← AC

Date
Page 21

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

LOAD A	AC ← M[A]
MUL B	AC ← AC * M[B]
STORE T	M[T] ← AC
LOAD C	AC ← M[C]
MUL D	AC ← AC * M[D]
ADD T	AC ← AC + M[T]
STORE X	M[X] ← AC

④ 0- Address Instruction

(STACK)

→ 0- Address is provided for computers that support 0- address instruct^{n}s coz of the absence of an address field in the computational instruct^{n}.

→ A stack-organised computer doesn't use an address field for the instructions like ADD & MUL.

→ However, an address field is reqd. for PUSH and POP operat^{n}s to specify the operand & communicate with the stack.

PUSH, POP have address
ADD, MUL have no address.

NOTE: → Before evaluating the expression, we need to convert the expression in reverse-polished notation (postfix notation).

STACK Notations

1. Infix (X = A+B)
2. Prefix (+AB)
3. Postfix (AB+) ← mainly computer uses this

(Take operand read, change stack if this operator apply then)

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

↓ (postfix)

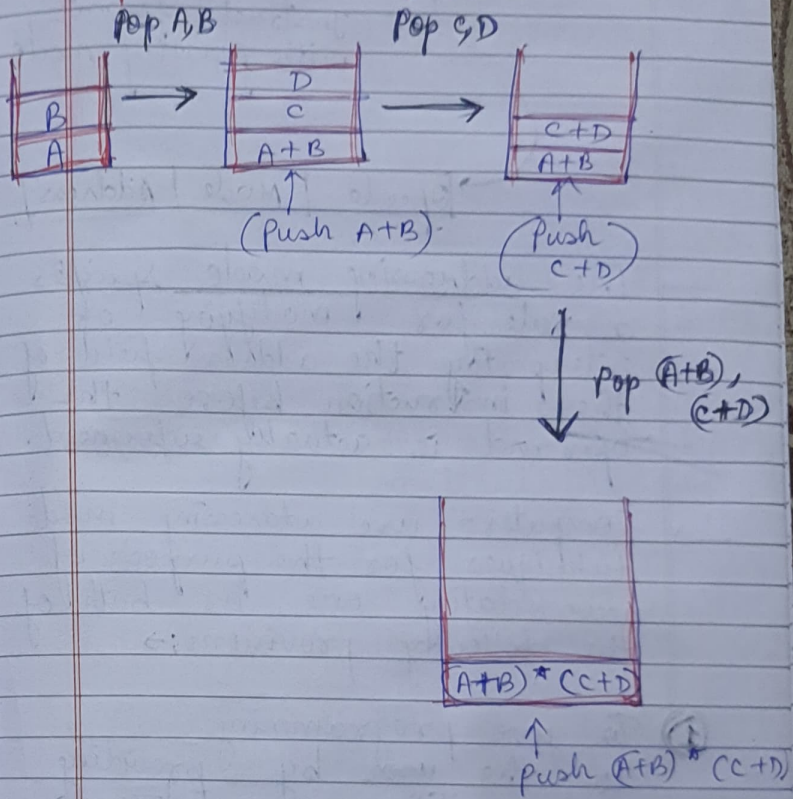
XAB* CD* + =

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

PUSH A
PUSH B
ADD (TOS ← M[A] + M[B])
PUSH C
PUSH D
ADD (TOS ← M[C] + M[D])
MUL (TOS ← (A+B) * (C+D))
POP X (M[X] ← (A+B) * (C+D))

Top of stack

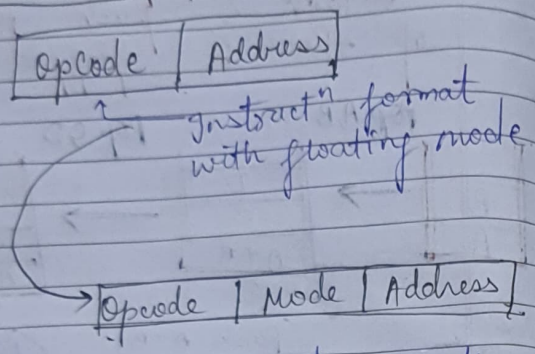
→ By default, pop 2 values from stack & push ADD, MUL, etc of those 2 popped values.



(m/m → memory)

Date 28/8/23
Page 24

ADDRESSING MODES



→ The addressing mode specifies a rule for modifying or interpreting the address field of the instruction before the operand is actually referenced.

→ Computers use addressing mode techniques for the purpose of accomodating one or both of the following provisions;→

(1) To give programming to the users by providing such facilities as pointers to m/m, control for loop control, indexing of data & program

Date
Page 25

(ii) To reduce the no. of bits in the addressing field of the instruct"s.

(1) IMPLICIT / IMPLIED Addressing Mode

→ In this mode, the operand is specified implicitly in the instruction itself.

eg) INCA → Increment Accumulator
CRC → clear Carry Flag
CLA → Complement Accumulator

(2) IMMEDIATE Addressing Mode

→ In this mode, the operand is directly provided as constant

→ used to denote constant value

eg) Add R1, #3 ← directly given
Increment in the value of operand R1 by +3

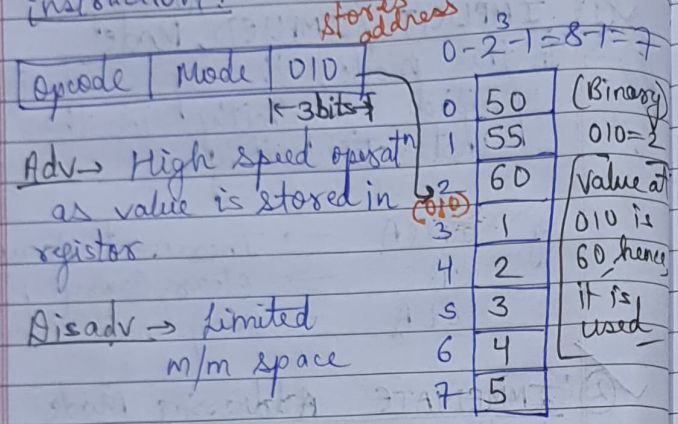
- Limitat" → limited m/m space
- Advantage → No need to access m/m

P.T.O →

Direct Addressing

3) Register Mode.

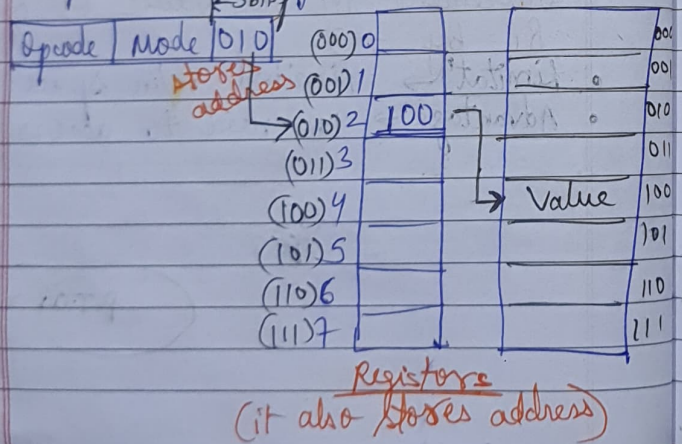
→ Operand is present in the register specified in address field of the instruction.



- Adv → High speed operatn as value is stored in register.
- Disadv → limited m/m space

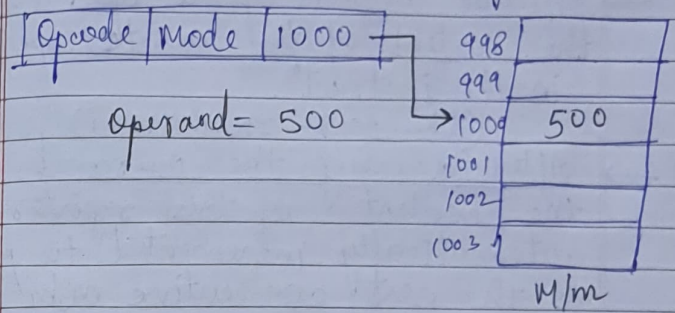
4) Register Indirect Addressing Mode

→ Register contains the address of the operand rather than the operand itself.



5) Direct Addressing Mode

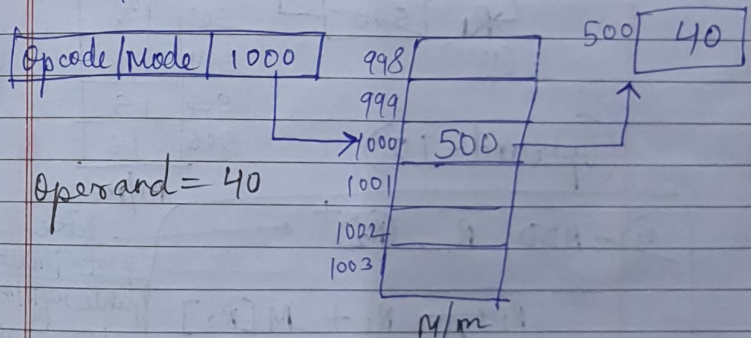
→ Actual address of m/m is given in the instructn itself.



29/8/23

6) Indirect Addressing Mode

→ In this mode, addressing field of the instructn gives the address where the effective address is stored in the m/m.



P.T.O.→

7) Auto-Indexed:

(a) Increment Addressing Mode

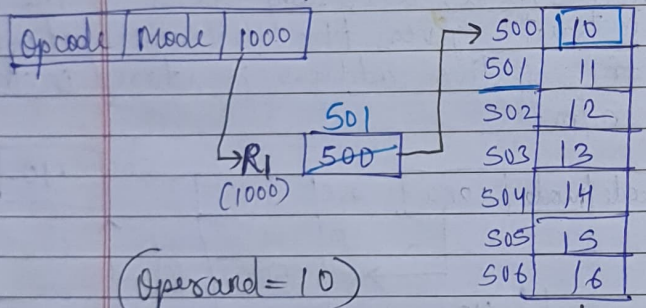
→ effective address of the operand is the content of the register specified in the instruction.

(like POST-INCREMENT)

→ After accessing the operand, the content of this register is automatically incremented to point to the next consecutive m/m location.

Notation → $(R_1)^+$

(Indirect)



(Operand = 10)

eg) ADD R_1, $(R_2)^+$ ← like post-increment

$R_1 \leftarrow R_1 + M[R_2]$

$R_2 \leftarrow R_2 + 1$

register ki jo effective address increment ho jayega

pehle register ki jo effective address hoga uski value use hogi aur phir

(b) Decrement Addressing Mode

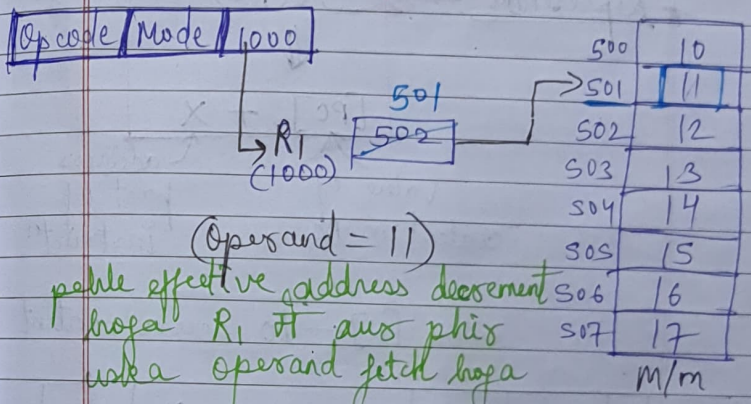
→ effective address of the operand is the content of the register specified in the instruction.

(like PRE-DECREMENT)

→ Before accessing the operand, the content of this register is automatically decremented to point to the previous consecutive m/m location.

Notation → $-(R_1)$

(Indirect)



(Operand = 11)

pehle effective address decrement hoga R_1 ka aur phir uska operand fetch hoga

eg) ADD R_1, $-(R_2)$ ← like pre-decrement

$R_2 \leftarrow R_2 - 1$

$R_1 \leftarrow R_1 + M[R_2]$

(8) Relative Addressing Mode

→ effective address is obtained by adding the content of the program counter to address part of the instruct^n

- **Program Counter (P.C.)**: stores the address of the next instruct^n to be executed.

Content of P.C.	+	Address part of instruct^n	=	effective address

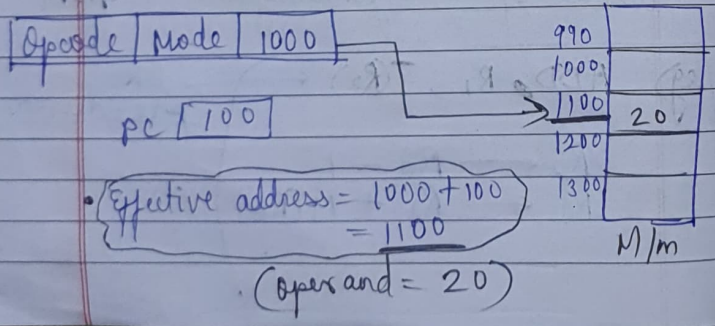
★ Representat^n → X(PC)

⇓

[PC] + X

- [PC]: value of content in PC
- X: address part of instruct^n

[ ] ← to access content



(9) Indexed Addressing Mode

Content of index register	+	Address part of instruct^n	=	effective Address

★ Representat^n → X[Ri] (Ri: Index Register)

⇓

[Ri] + X

(10) Base Register Addressing Mode

Content of base register	+	Address part of instruct^n	=	effective Address

(11) Branching

```
1)  i=1
2)  if (i<=10){goto 3} else {goto 10}
3)  print( )
4)  ,,
5)  ,,
6)  ,,
7)  ,,
8)  i++
9)  goto 2
10) exit
```

statements

(c/a → called as)

Instruct^{n} and Instruct^{n} Sequencing

→ A processor contains a register c/a PROGRAM COUNTER (PC) which holds the address of the instruct^{n} executed next.

→ To begin executing a program, the address of it's first instruct^{n} must be placed in P.C.

→ Then the processor control circuit use the info. in the P.C. to fetch & execute the instruct^{n}s, **one** at a time, in the order of increasing addresses.
This is c/a Straight-line sequencing

eg) C = A + B
C ← M[A] + M[B]

i	MOV R0, A	Instruct^{n}s for program
i+4	ADD R0, B	
i+8	MOV C, R0	
A		Data for program
B		
C		

☆ Phases/steps of Instruction cycle

1) Instruct^{n} fetch from m/m
2) Instruct^{n} decode
3) Operand fetch from m/m
4) Instruct^{n} execut^{n}
5) Write Back

→ Executing a given instruct^{n} is a 2-PHASE procedure.

First Phase) Instruct^{n} Fetch (IF)

step ①
→ The instruct^{n} is fetched from m/m locat^{n} whose address is in the P.C.

→ The instruct^{n} is placed in the instruct^{n} register in the processor

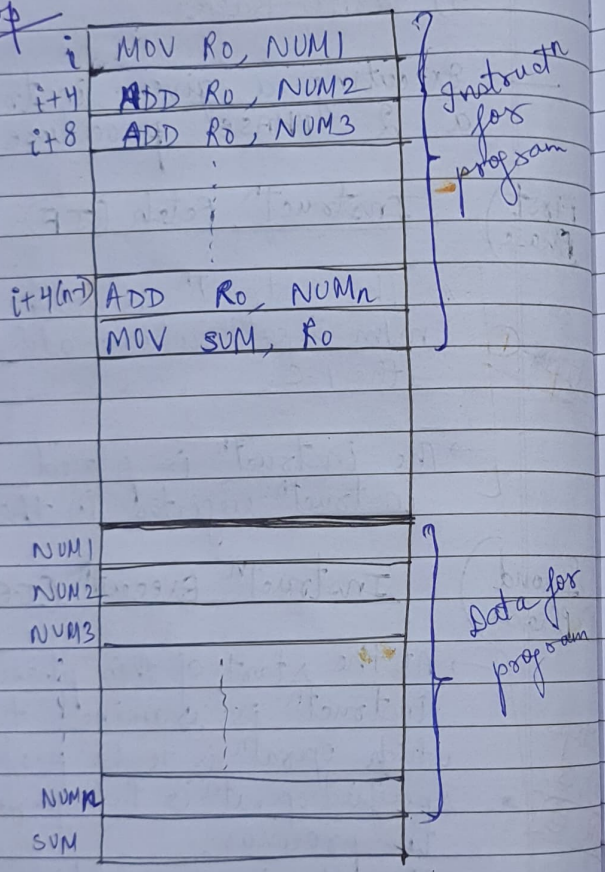
Second Phase) Instruct^{n} Execut^{n} (IE)

Steps ②, ③, ④, ⑤
→ At the start of this phase, the instruct^{n} is examined to determine which operat^{n} is to be performed. The specified operat^{n} is then performed by the processor.
→ This often involves fetching operands from the m/m or from processor registers, performing an arithmetic or logic operat^{n} & then storing the result in the destinat^{n} locat^{n}.

BRANCHING

Q Consider a task of adding a list of 'n' no's NUM1, NUM2, ... NUMn & store in variable SUM.

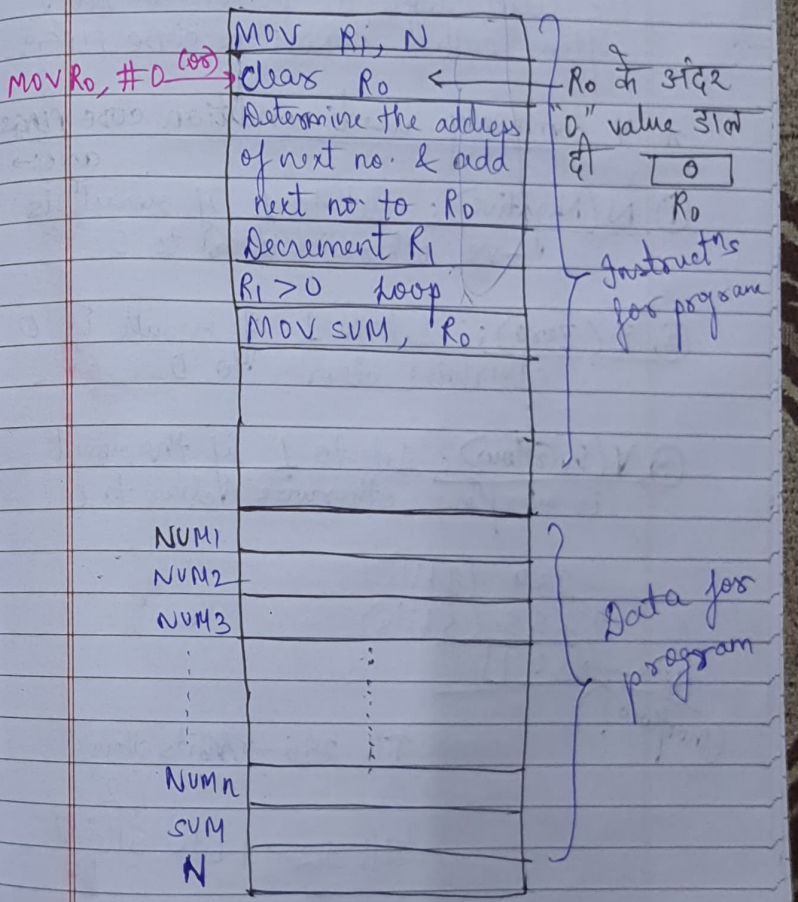
Without loop



→ The loop is a straight-line sequence of instructns executed as many times as needed.

Through loop

N → loop counter (total no's to be added)



CONDITION CODES

→ The processor keeps track of info. about the results of various operations for use by subsequent conditional branch instructions.

→ This is accomplished by recording the reqd. info. in individual bits, often called CONDITION CODE FLAGS.

★ 4 commonly used condition CODE FLAGS are:-

① N (Negative): Set to 1 if result is -ve otherwise clear to 0.

② Z (Zero): Set to 1 if result is 0 otherwise clear to 0

③ V (Overflow): Set to 1 if the result is overflow otherwise clear to 0.

```
  254 (8 bits)
+ 255 (8 bits)
-------------
  509
```

9 bits → (overflow!)

Till 256 → 8 bits allowed (2^8)

Till 512 → 9 bits allowed (2^9)