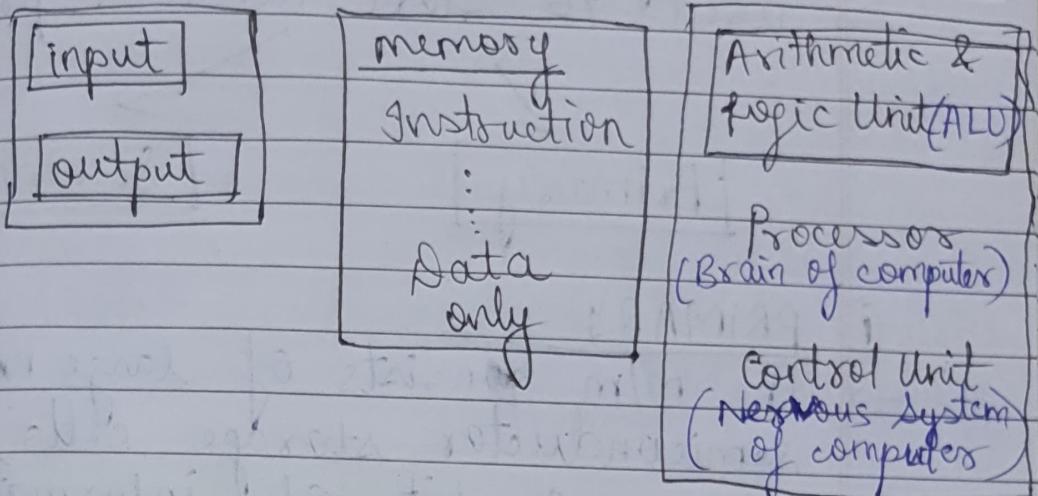


FUNCTIONAL UNITS OF COMPUTER



(Control Unit generates control signal)

- ① Examples of I/P unit are:-
- Keyboard
 - Mouse
 - Joystick

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

Memory Unit

→ used to store data & program

Primary

Secondary

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)

② [CACHE M/m]: Small, fast RAM units are called "cache m/m".

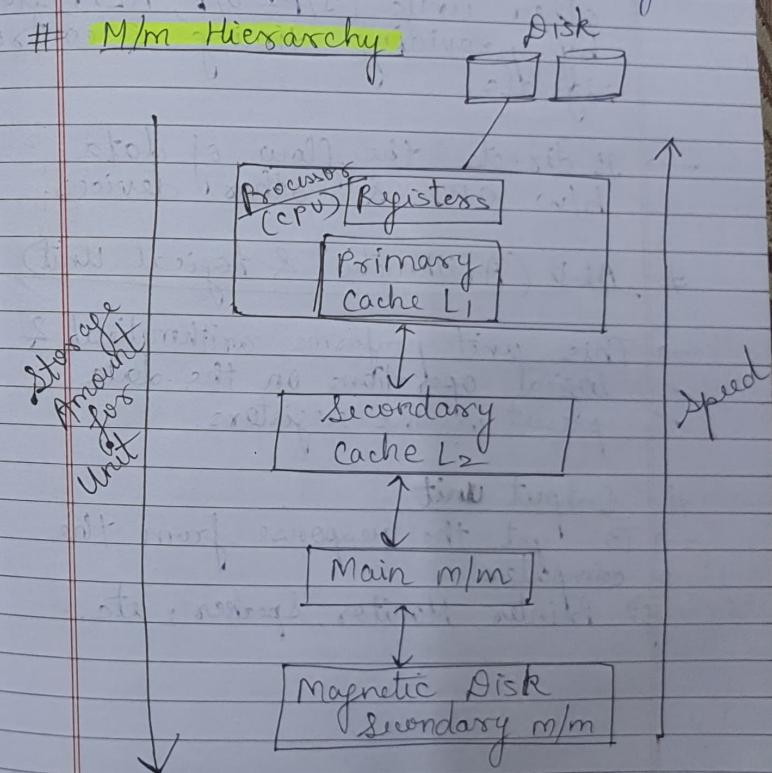
③ Largest & slowest unit is main m/m.

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

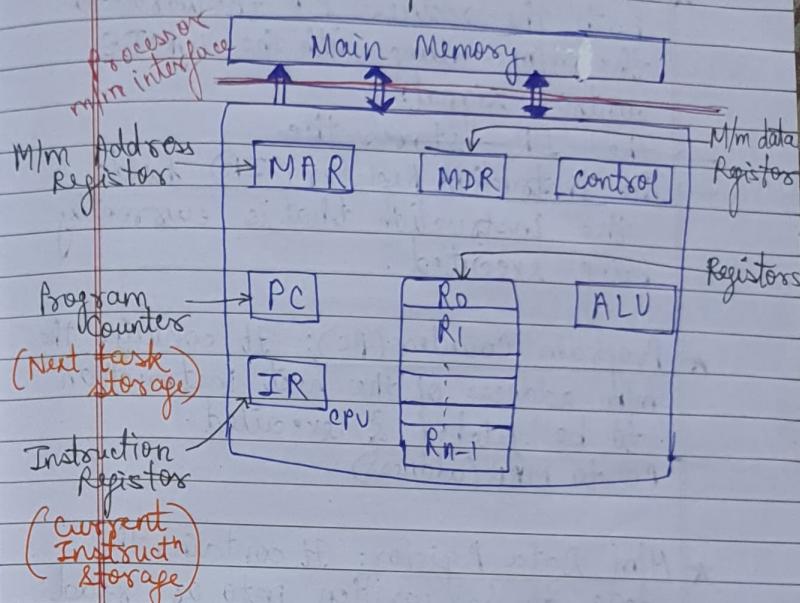
- Instructs the m/m, logical unit, I/P-O/P unit of computer on how to respond to the program's instruction.
- It directs the operations of other units (I/P-O/P, ALU, etc) by providing timing & control signals.
- It directs the flow of data b/w CPU & other devices.

ALU (Arithmetic & Logical Unit)

- This unit performs arithmetical & logical operations on the data present in the registers.

Output Unit

- To get the response from the computer.
- (i) Printer, Monitor, Speakers, 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 registers.

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

→ The JR stores the

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

* Program Counter (PC): It contains the m/n 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.

→ Instruction execute steps :-

PC

MAR

MM ← cu

MDR

IR

MAR

MM

MDR

PR

operation

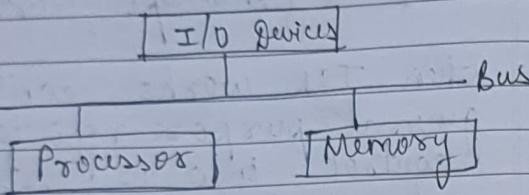
ALU

Address to locate data
read control signal

cu → send read
control signal.

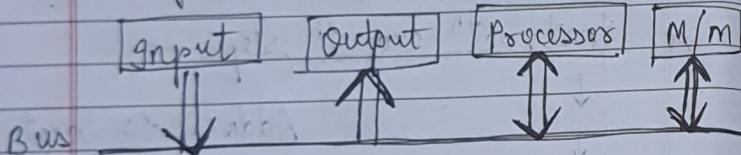
Bus Structure

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



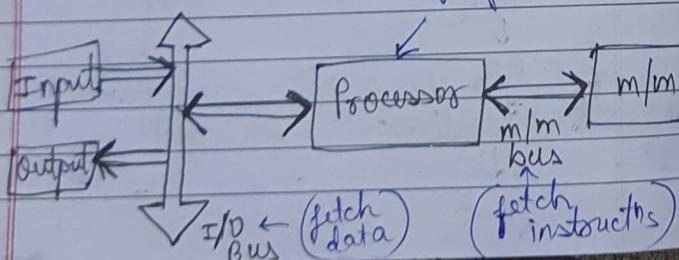
Type of Buses in Computer

✓ (I) Single Bus Structure



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

(II) Double Bus Structure very fast speed



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

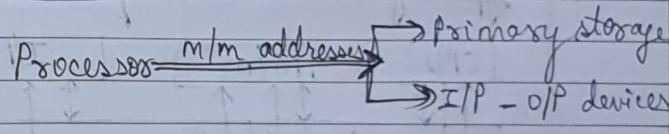
✓ (III) 3-bus structure

→ we have 3 buses:-

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

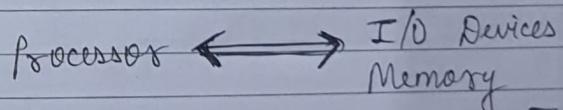
① ADDRESS BUS: It carries m/m addresses from the processor to other components such as primary storage & I/P-O/P devices.

→ Uni-directional.



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

→ Bi-directional

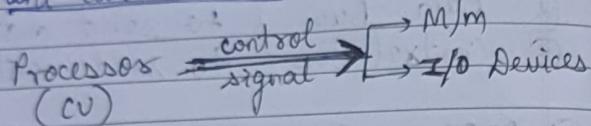


Read → m/m to processor
Write → processor to m/m

② CONTROL BUS: It carries control signals (generated from CPU present in the processor) from processor to other components (m/m & I/O devices).

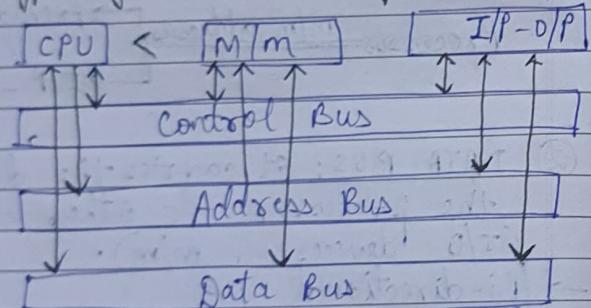
→ also carries the clock pulses.

→ Uni-directional



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

► Types of Buses in Computers (Diagram)



Memory Address & Operation

→ A word consists of 'n' no. of bits

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

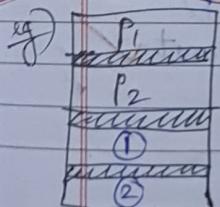
* Storage of 'n' words in m/m.

0	W0
1	W1
2	W2
3	:
4	:
5	:
6	Wn-1

$$n = 4 \text{ bytes}$$

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

* COMPACTON → used space ek jagah chla jaata hai aur unused space ek jagah (free space) chla jaata hai.



requires ① ② free space
P3 → for its storage so
cused space (cused space) gets one
side free space ek jagah
ek jagah hai to store P3 together

① Byte Addressability

→ To assign address to each byte
 (num bits in address देने के लिए बायट का दृष्टि करें)

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

0	W ₀
4	W ₁
8	W ₂
12	W ₃
...	...
W _{n-1}	W _{n-1}

Word alignment ✓

1	W ₀
5	W ₁
9	W ₂
13	W ₃
...	...
W _{n-1}	W _{n-1}

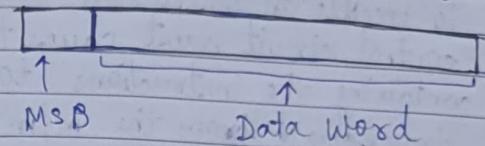
Word alignment X

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

ABHI	A	N	V	I	-	-	-
0	1	2	3	4	5	6	7

(multiple of 4)
 Word alignment ✓

② Word Representation in m/m



Big Endian & Little Endian Assignment

word =

MSB	a	b	c	d
-----	---	---	---	---

a	b	c	d
---	---	---	---

 (same)

0 1 2 3

d	c	b	a
---	---	---	---

 (reverse)

0 1 2 3

Big-Endian

little-Endian

- Big Endian Assignment: lower byte address is used to represent the MSB of the word!

- Little Endian Assignment: highest byte address is used to represent the MSB of the word.

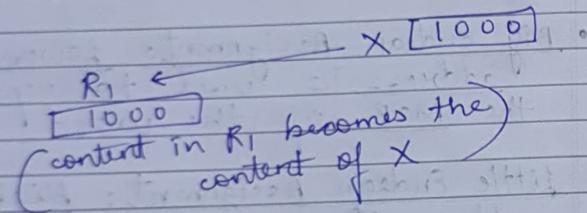
Memory Operations

→ 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.

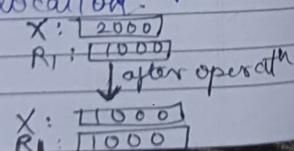
② LOAD R₁, X
R₁ ← M[X]



② STORE (Write) →

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

③ STORE X, R₁
M[X] ← R₁



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

Assembly lang. code:

Assemblers → M/c lang. code

MT

Assembly Language

→ M/c understands 0 and 1 BYT 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~~ OP code for the operation performed by the instrucⁿ (operaⁿ code)

ADD R₁, R₂
 ↓
 000

→ Instrucⁿ

Notations used to represent instrucⁿ

① Register Transfer Notation (RTN)

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

e.g. LOC, Place, X, DATAIN, DATAOUT
 (R₁, R₂, etc)
 (Registers)

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

e.g) R₁ ← main LOC at data transfer locn.
 R₁ ← [LOC]
 ↑ address ↑ content

e.g) R₁ ← R₂ at data transfer locn.
 R₁ ← [R₂]

e.g) R₁ ← (R₁, R₂) → done at data transfer locn.
 R₁ ← [R₁] + [R₂] → (A = 10
 B = 20
 A = A + B)

② Assembly Language Notation

e.g) ADD, [R₁, R₂, R₃] ← CPU registers

mnemonic meaning

R₁ ← R₂ + R₃ / R₃ ← R₁ + R₂

use only one of
 these 2 notations

(R₁ → destination
 R₂, R₃ → operands)

e.g) ADD R₁, X at data transfer locn.
 ↓ meaning m/m locn.

R₁ ← R₁ + M[X]

e.g) 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.

$$\text{eg) } X = (A+B) * (C+D)$$

$$\text{ADD } R_1, A, B$$

$$\text{ADD } R_2, C, D$$

$$\text{MUL } X, R_1, R_2$$

★ RTN Notation

$$R_1 \leftarrow M[A] + M[B]$$

$$R_2 \leftarrow M[C] + M[D]$$

$$M[X] \leftarrow R_1 \star R_2$$

- Advantages → short program
- Disadvantages → M/m size is more.

② 2- Address Instruction

MOV	R ₁	A
Opcode	Destination	Source1

$$\text{eg) } X = (A+B) * (C+D)$$

(R₁ & R₂ are empty in the starting)
~~MOV R₁, A~~ R₁ = R₁ + M[A]

~~ADD R₁, B~~

$$R_1 = R_1 + M[B] = M[A] + M[B]$$

$$R_2 = R_2 + M[C]$$

$$R_2 = R_2 + M[D] = M[C] + M[D]$$

$$R_1 = R_1 * R_2 = (A+B) * (C+D)$$

$$X = R_1 = (A+B) * (C+D)$$

- MOV → moves or transfers the operand 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' addresses in instruction \rightarrow 'n' addresses used in instruction

Date 25/8/23
Page 2

Date _____
Page 21

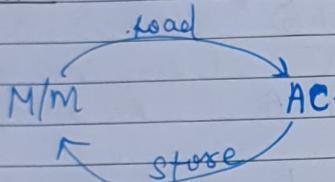
③ 1- Address Instruction

(ACCUMULATOR register)

- use an implied accumulator register (AC) for all data manipulation.
- we use "LOAD" & "STORE" operations.
- Here, AC works as 2nd register

i) LOAD \rightarrow loads data from m/m to AC

ii) STORE \rightarrow store AC result in m/m.



$$\text{eg) } X = (A + B) * (C + D)$$

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

$$\text{eg) } X = (A * B) + (C * D)$$

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

④ 0- Address Instruction

(STACK)

→ 0-Address is provided for computers that support 0-address instructions coz of the absence of an address field in the computational instructn.

→ 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 operations 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 well-formed polished notation (postfix notation).

STACK NOTATION

① Infix ($X = A + B$)

② Prefix ($+ AB$)

③ Postfix ($AB +$) \leftarrow mainly computer uses this

(read operand and read
change stack if operator apply logic
operator apply logic)

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

(Postfix)

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

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

PUSH A

PUSH B

ADD ($TOS \leftarrow M[A] + M[B]$)

PUSH C

PUSH D

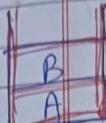
ADD ($TOS \leftarrow M[C] + M[D]$)

MUL ($TOS \leftarrow (A+B) * (C+D)$)

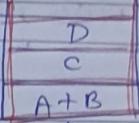
POP X ($M[X] \leftarrow (A+B) * (C+D)$)

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

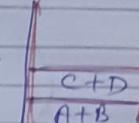
Pop A, B



Pop C, D

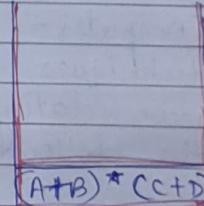


(Push A+B)



(Push C+D)

↓
Pop (A+B),
(C+D)



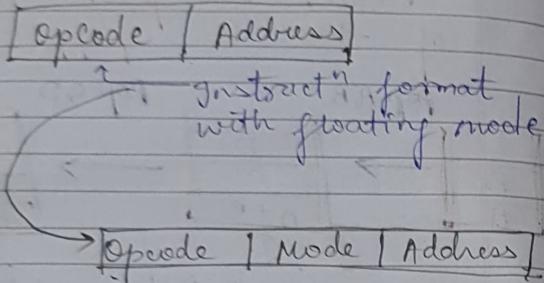
↑
push (A+B) * (C+D)

(m/m → memory)

Date 28/8/23
Page 29

Date _____
Page 25

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 accommodating one or both of the following provisions:-

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

② To reduce the no. of bits in the addressing field of the instruct's.

Addressing ① IMPLICIT / IMPLIED Mode

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

e.g.) 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

g) Add R₁, #3 → directly given
Increment in the value of operand R₁ by +3.

• Limitat'n → 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.

Stores address

$$0-2-1 = 8-1-7$$

Opcode	Mode	010	13
		← 3 bits	0-2-1 = 8-1-7
		0	50 (Binary)
		1	55 010 = 2
		2	60 Value at
		3	1 010 is
		4	2 60, hence
		5	3 it is used
		6	4
		7	5

- Adv → High speed opntrn as value is stored in register.

- Disadv → limited m/m space

④ Register Indirect Addressing

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

← 3 bits

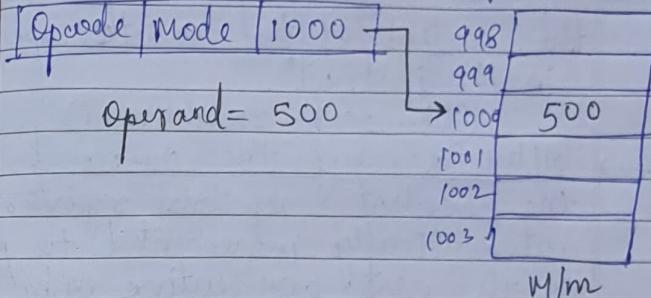
Opcode	Mode	010	000	001
		000	00	00
		001	00	010
		100	00	011
		(010)2	00	100
		(011)3	00	101
		(100)4	00	110
		(101)5	00	111
		(110)6	00	
		(111)7	00	

Registers

(it also stores address)

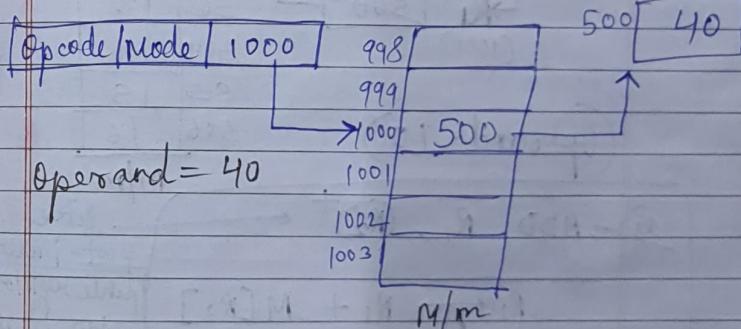
Direct Addressing Mode

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



Indirect Addressing Mode

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



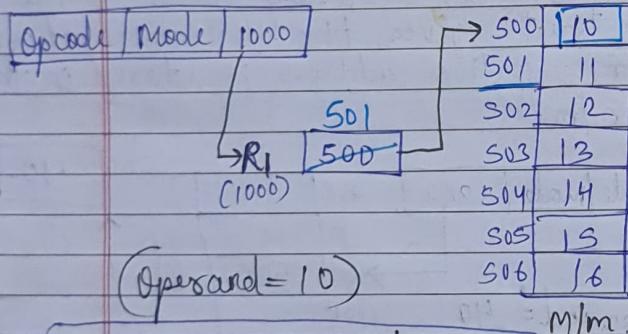
P.T.O. →

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) (Index)



eg) ADD $R_1, (R_2)^+$ ← like post-increment
 $R_1 \leftarrow R_1 + M[R_2]$
 $R_2 \leftarrow R_2 + 1$

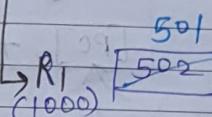
register R_2 in step 2 effective address increment ho jayega

(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)

Opcode / Mode | 1000



500	10
501	11
502	12
503	13
504	14
505	15
506	16
507	17

m/m

(Operand = 11)
 public effective address derement hoga
 R_1 aur phir use a operand fetch hoga

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

$$R_2 \leftarrow R_2 - 1$$

$$R_1 \leftarrow R_1 + M[R_2]$$

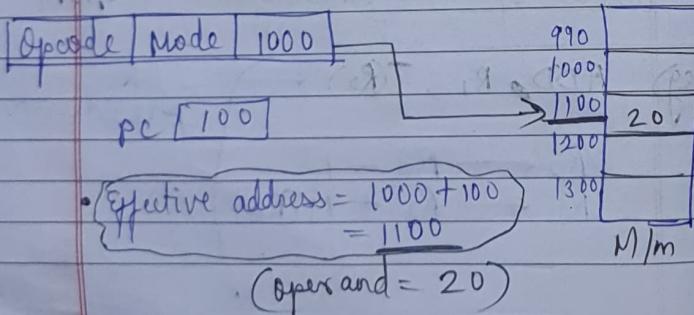
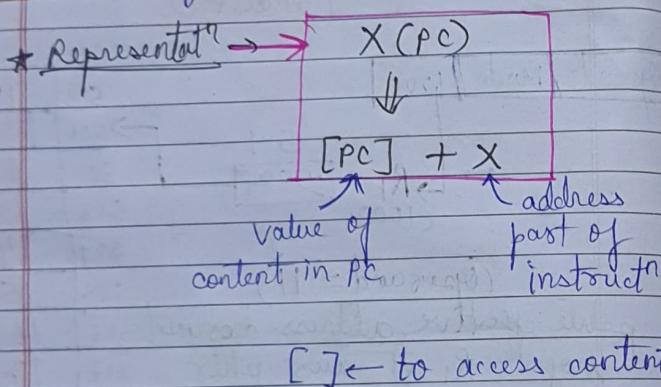
✓(8)

Relative Addressing Mode

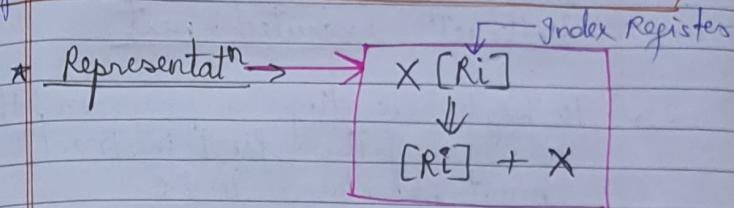
effective address is obtained by adding the content of the program counter to address part of the instruction.

- program counter: stores the address (P.C.) of the next instruction to be executed.

$$\boxed{\text{content of P.C.} + \text{Address part of instruction} = \text{effective address}}$$

Indexed Addressing Mode

$$\boxed{\text{Content of index register} + \text{Address part of instruction} = \text{effective address}}$$

Base Register Addressing Mode

$$\boxed{\text{content of base register} + \text{Address part of instruction} = \text{effective address}}$$

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)

Date _____

Page _____

32

Instructⁿ and Instructⁿ sequencing

→ A processor contains a register c/a PROGRAM COUNTER (PC) which holds the address of the instructⁿ executed next.

→ To begin executing a program, the address of its first instructⁿ must be placed in P.C.

→ Then the processor control circuit uses the info. in the P.C. to fetch & execute the instructⁿs one at a time, in the order of increasing addresses.

This is c/a. straight-line sequencing

eg)
$$\begin{aligned} C &= A + B \\ C &\leftarrow M[A] + M[B] \end{aligned}$$

i	MOV R0, A
i+4	ADD R0, B
i+8	MOV C, R0

Instructⁿ for program

A
B
C

Data for program

Date _____

Page _____

33

* Phases/Steps of Instruction cycle

- 1) Instructⁿ fetch from m/m
- 2) Instructⁿ decode
- 3) Operand fetch from m/m
- 4) Instructⁿ executⁿ
- 5) Write Back

→ Executing a given instructⁿ is a 2-phase procedure.

First Phase

Step - ①

→ The instructⁿ is fetched from m/m locatⁿ whose address is in the P.C.

→ The instructⁿ is placed in the instructⁿ register in the processor.

Second Phase

Steps
②, ③,
④, ⑤

Instructⁿ Executⁿ (I-E)

→ At the start of this phase, the instructⁿ is examined to determine which operatⁿ is to be performed. The specified operatⁿ is then performed by the processor.

→ This often involves fetching operands from the m/m or from processor registers, performing an arithmetic or logical operatⁿ & then storing the result in the destinatⁿ locatⁿ.

BRANCHING

Q Consider a task of adding a list of 'n' nos. $NUM_1, NUM_2, \dots, NUM_n$ & store in variable SUM .

Without loop

i	MOV R0, NUM1
i+4	ADD R0, NUM2
i+8	ADD R0, NUM3
⋮	⋮
i+4(n)	ADD R0, NUMn
	MOV SUM, R0

} instruction
for
program

NUM1

NUM2

NUM3

⋮

NUMn

SUM

} Data for
program

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

Through loop

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

MOV R1, N	Ro at 342 "0" value start at 0 Ro Instruction for program
MOV R0, #0	
Clears R0 ←	
Determine the address of next no. & add next no. to R0	
Decrement R1	
R1 > 0 loop	
MOV SUM, R0	

NUM1

NUM2

NUM3

⋮

NUMn

SUM

N

} Data for
program

CONDITION CODES

- The processor keeps track of info. about the results of various operatⁿs for use by subsequent conditional branch instrucⁿs.
- 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

$$\begin{array}{r} 254 \text{ (8 bits)} \\ + 255 \text{ (8 bits)} \\ \hline \end{array}$$

qbits → 509
(Overflow!)

Till 256 → 8 bits allowed
 (2^8)

Till 512 → 9 bits allowed
 (2^9)