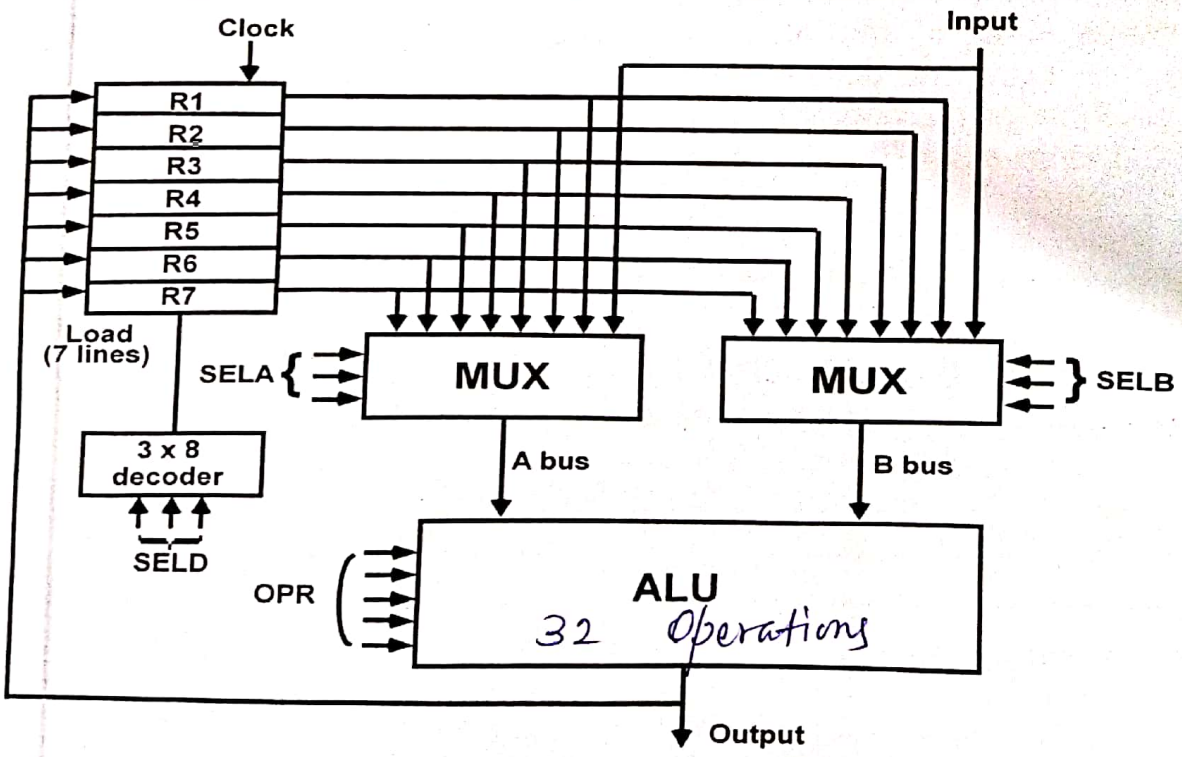
GENERAL REGISTER ORGANISATION: When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic and shift micro-operation in the processor.

A Bus organization for seven CPU registers:—

The output of each register is connected to two multiplexer (mux) to form the two buses A & B. The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses forms the input to a common ALU. The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed. The result of the micro-operation is available for output and also goes into the inputs of the registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer both between the data in the output bus and the inputs of the selected destination register.



The control unit that operates the CPU bus system directs the in flow through the registers and ALU by selecting the various comp the systems.

$$R_1 \rightarrow R_2 + R_3$$

(1) MUX A selection (SEC A): to place the content of R2 into bus
(2) MUX B selection (sec B): to place the content of R3 into bus B
(3)ALU operation selection (OPR): to provide the arithmetic additio
(4) Decoder destination selection (SEC D): to transfer the conte output bus into R_1

These form the control selection variables are generated in the co and must be available at the beginning of a clock cycle. The data two source registers propagate through the gates in the multiplexe ALU, to the output bus, and into the into of the destination reg during the clock cycle intervals.

Control Word:

There are 14 binary selection inputs in the units, and their combi specified a control word. It consists of four fields three fields cont bits each, and one field has five bits. The three bits of SEL A select register for the A input of the ALU. The three bits of SEL B select register for the B input of the ALU. The three bit of SEC D destination register using the decoder and its seven load outputs bits of OPR select one of the operations in the ALU. The 14-bit cor when applied to the selection inputs specify a particular micro-oper

Table: Encoding of Register selection fields.

Binary Code	SEL A	SEL B	SEL D
000	Input	Input	None
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

Table: Encoding of ALU operation

OPR & elect	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A-B	SUB
00110	Decrement A	DEC A
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of Micro-operation for the CPU Symbolic Designation

Micro Operation	SECA	SEC B	SEL D	OPR	Control Word SELA	SELB	SELD	OPR
$R_1 \rightarrow R_2 - R_3$	R_2	R_3	R_1	SUB	010	011	001	00101
$R_4 \rightarrow R_4 \vee R_5$	R_4	R	R_4	OR	100	101	100	01010
$R_6 \rightarrow R_6 + 1$	R_6	-	R_6	MCA	110	000	110	00001
$R_7 \rightarrow R_1$	R_1	-	R_7	TSFA	001	000	111	00000
Output $\leftarrow R_2$	R_2	-	None	TSFA	010	000	000	00000
Output $\leftarrow$ Input	Input	-	None	TSFA	000	000	000	00000
$R_4 \rightarrow$ SHL R4	R_4	-	R_4	SHLA	100	000	100	11000
$R_5 \rightarrow 0$	R_5	R_5	R_5	XOR	101	101	101	01100

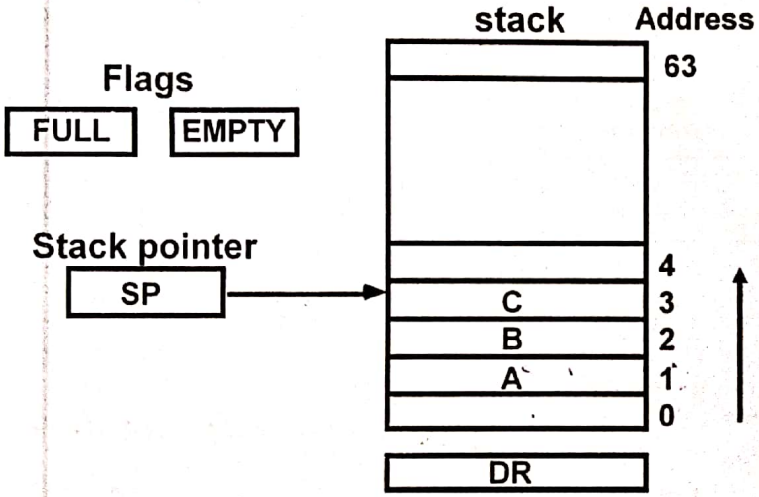
Stack Organization

A useful feature that is included in the CPU of most computers is a stack or last-in first out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation a stack can be companied to a stack of trays.

The stack in Digital Computer is essentially a memory unit with an address register that can count only (after an initial value is loaded into it.) The register that holds the address for the stack is called a Stack Pointer (SP) because its values always points at the top item in the stack.

The two operations – PUSH (insert)
- Pop (delete)

Register Stack:- A stack can be placed in a portion of a large memory as it can be organized as a collection of a finite number of memory words as register.



Initially SP = 0
EMPTY = 1
FULL = 0

63 : binary value of 63
111111

On adding +1
Discard 1000000

In a 64- word stack, the stack pointer contains 6 bits because $2^6 = 64$.

The one bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty. DR is the data register that holes the binary data to be written into on read out of the stack.

Initially, SP is decide to 0, EMTY is set to 1, FULL = 0, so that SP points to the word at address 0 and the stack is masked empty and not full.

PUSH	SP ← SP + 1	increment stack pointer
	M [SP] ← DR	unit item on top of the Stack
	If (SP = 0) then	(FULL ← 1) check it stack is full
	EMTY ← 0	mask the stack not empty.
POP	DR ← M[SP]	read item trans the top of stack
	SP ← SP –1	decrement SP

last SP = 0 : 000000
− 1
111111

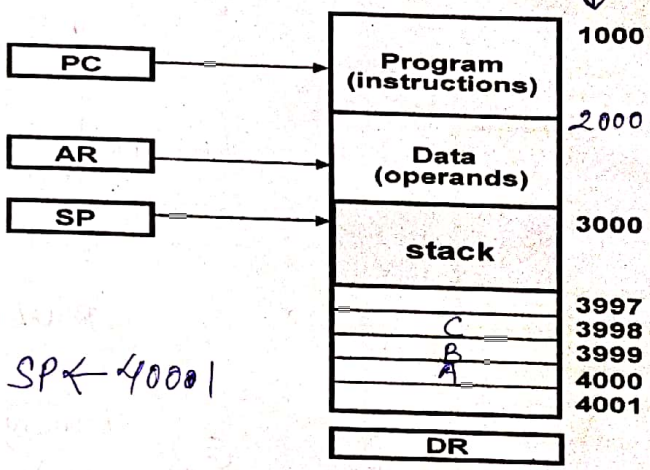
4
2
1
63

If(SP = 0) then (EMTY ← 1) check it stack is empty
FULL ← 0 mark the stack not full.

Memory stack: Memory with Program, Data, and Stack Segments
- A portion of memory is used as a stack with a processor register as a stack pointer
- PUSH: SP ← SP - 1
  M[SP] ← DR
- POP: DR ← M[SP]
  SP ← SP + 1
- Most computers do not provide hardware to check stack overflow (full stack) or underflow(empty stack)

Address

Push :-
SP ← SP - 1
M[SP] ← DR



Pop :
DR ← M[SP]
SP ← SP + 1

Initially, SP ← 4000 1

* Most Computer does not provide hardware to check Memory full stack or empty stack.

REVERSE POLISH NOTATION

Arithmetic Expressions: A + B

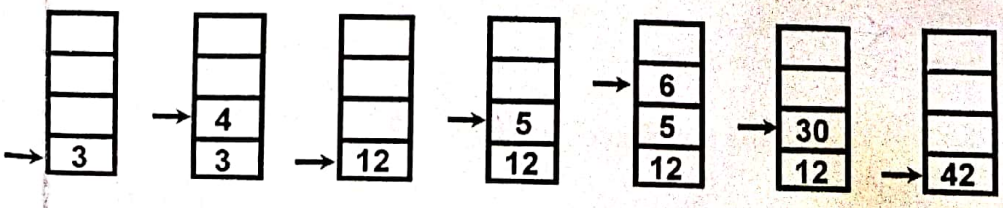
A + B Infix notation
+ A B Prefix or Polish notation
A B + Postfix or reverse Polish notation

- The reverse Polish notation is very suitable for stack manipulation

Evaluation of Arithmetic Expressions

Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



Instruction Formats:

The most common fields found in instruction format are:-

(1) An operation code field that specified the operation to be performed
(2) An address field that designates a memory address or a processor registers.
(3) A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address field in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organization.

(1) Single Accumulator organization ADD X AC ← AC + M [×]
(2) General Register Organization ADD R1, R2, R3 R1 ← R2 + R3
(3) Stack Organization PUSH X

Three address Instruction

Computer with three addresses instruction format can use each address field to specify either processor register are memory operand.

ADD R_1, A, B $A_1 \rightarrow M[A] + M[B]$
ADD R_2, C, D $R_2 \rightarrow M[C] + M[B]$ X = (A + B) * (C + A)
MUL X, R_1, R_2 M [X] $R_1 * R_2$

The advantage of the three address formats is that it results in short program when evaluating arithmetic expression. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two Address Instruction

Most common in commercial computers. Each address field can specify either a processes register on a memory word.

MOV	R_1, A	$R_1 \rightarrow M[A]$	
ADD	R_1, B	$R_1 \rightarrow R_1 + M[B]$	
MOV	R_2, C	$R_2 \rightarrow M[C]$	X = (A + B) * (C + D)
ADD	R_2, D	$R_2 \rightarrow R_2 + M[D]$	
MUL	R_1, R_2	$R_1 \rightarrow R_1 * R_2$	
MOV	X_1 R_1	$M[X] \rightarrow R_1$	

One Address instruction

It used an implied accumulator (AC) register for all data manipulation. For multiplication/division, there is a need for a second register.

LOAD	A	$AC \rightarrow M[A]$	
ADD	B	$AC \rightarrow AC + M[B]$	
STORE	T	$M[T] \rightarrow AC$	X = (A +B) × (C + A)
LOAD	C	$AC \rightarrow M(C)$	All operations are done between the AC
ADD	D	$AC \rightarrow AC + M(D)$	register and a memory operand. It's the
ML	T	$AC \rightarrow AC + M(T)$	address of a temporary memory location
STORE	X	$M[\times] \rightarrow AC$	required for storing the intermediate result.

Zero – Address Instruction

A stack organized computer does not use an address field for the instruction ADD and MUL. The PUSH & POP instruction, however, need an address field to specify the operand that communicates with the stack (TOS → top of the stack)

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

Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer register as memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction between the operand is activity referenced. Computer use addressing mode technique for the purpose of accommodating one or both of the following provisions.

(1) To give programming versatility to the uses by providing such facilities as pointer to memory, counters for top control, indexing of data, and program relocation.
(2) To reduce the number of bits in the addressing fields of the instruction.

The basic operation cycle of the computer

(1) Fetch the instruction from memory
(2) Decode the instruction
(3) Execute the instruction

Program Counter (PC) keeps track of the instruction in the program stored in memory. PC holds the address of the instruction to be executed next and in incremented each time an instruction is fetched from memory.

(1) Implied mode
(2) Immediate mode
(3) Register mode
(4) Register mode
(5) Register – indirect mode
(6) Auto increment/ Auto decrement mode
(7) Direct Address mode
(8) Indirect Address mode
(9) Relative Address mode
(10) Indexed Addressing mode
(11) Base Register Addressing mode

Opcode	Mode	Address

Instruction format with mode field

Relative Address: The content of PC is added to the address part of the instruction in order to obtain the effective address Ex :- PC = 825 + 1 + 24

Indexed addressing Mode:- The content of index register is added to the address part of the instruction in order to obtain the effective address.

Base Register Addressing Mode:-The content of a base register is added to the address bank of the instruction to obtain the effective address.

Indirect Address:-
Effective Address = Address part of instruction + content of CPU register

Numerical Example

Operand value.

Addressing Mode	Effective Address	Content of AC
Immediate Address Mode	201	500
Direct Address Mode	500	800
Indirect Address Mode	600	300
Register Mode (Direct)		400
Register Indirect Mode	400	700
Relative Address Mode	PC + addr part = 702	325
Indexed Address Mode	XR + addr part 100 + 500 = 600	900
Autoincrement Mode	400	700
Autodecrement Mode	399	450

R1 = 400

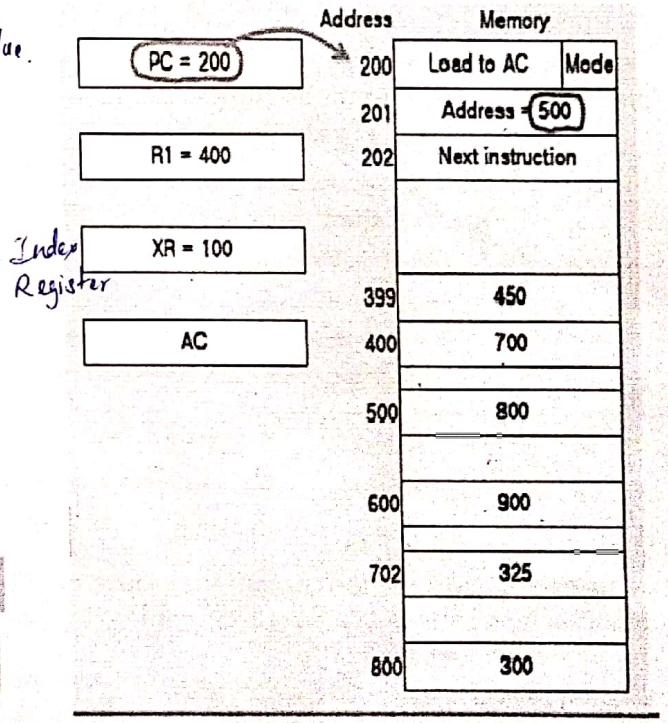
500 + 202 (PC)

500 + 100 (XR)

R1 = 400 (after)

R1 = 400 - 1 (prior)

but after execution of instruction is 401



Implicit IN
INC
~~DAD~~
~~JMP~~
DCR
RCC

X ~~Implied~~ Reg. Add. Mode

Immediate ADI 08H
ADD A ← A + 08H

ADD B
ADD A ← A + B (data)

Direct ADD 2000H
ADD A ← A + 2000 (data)

E.g. MVI A, 32H

Data Transfer & Manipulation

Computer provides an extensive set of instructions to give the user the flexibility to carryout various computational task. Most computer instruction can be classified into three categories.

(1) Data transfer instruction
(2) Data manipulation instruction
(3) Program control instruction

Data transfer instruction cause transferred data from one location to another without changing the binary instruction content. Data manipulation instructions are those that perform arithmetic logic, and shift operations. Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

(1) Data Transfer Instruction

Data transfer instruction move data from one place in the computer to another without changing the data content. The most common transfers are between memory and processes registers, between processes register & input or output, and between processes register themselves.

(Typical data transfer instruction)

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

(2) Data Manipulation Instruction

It performs operations on data and provides the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types.

(a) Arithmetic Instruction
(b) Logical bit manipulation Instruction
(c) Shift Instruction.

(a) Arithmetic Instruction

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	Add
Subtract	Sub
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Basses	SUBB
Negate (2's Complement)	NEG

(b) Logical & Bit Manipulation Instruction

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-Or	XOR
Clear Carry	CLRC
Set Carry	SETC
Complement Carry	COMC
Enable Interrupt	ET
Disable Interrupt	OI

(c) Shift Instruction

Instructions to shift the content of an operand are quite useful and one often provided in several variations. Shifts are operation in which the bits of a word are moved to the left or right. The bit-shifted in at the and of the word determines the type of shift used. Shift instruction may specify either logical shift, arithmetic shifts, or rotate type shifts.

Name	Mnemonic
Logical Shift right	SHR
Logical Shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate mgmt through carry	RORC
Rotate left through carry	ROLC

RISC & CISC ARCHITECTURE An important aspect of computer architecture is the design of the instruction set for the processor.

Characteristics of CISC architecture are:

(1) A large no. of instruction-typically from 100 to 250 instruction.
(2) Some instruction that perform specialized tasks and one used infrequently.
(3) A large variety of addressing modes - typically from 5 to 20 different modes.
(4) Variable length instruction format
(5) Instruction that manipulate operates in memory.

RISC CHARACTERSTICS

The concept of RISC arithmetic involves an attempt to reduce execution time by simplifying the instruction set of the computer.

The major characteristics of a RISC processes are:

(1) Relatively few instructions
(2) Relatively few addressing modes
(3) Memory access limited to load & store instruction
(4) All operations done within the registers of the CPU.

(5) Fixed-length, easily decoded instruction format
(6) Single-cycle instruction execution
(7) Handmaiden rather than micro programmed control.

More RISC Characteristics

- A relatively large numbers of registers in the processor unit.
- Efficient instruction pipeline
- Compiler support: provides efficient translation of high-level language Programs into machine language programs.

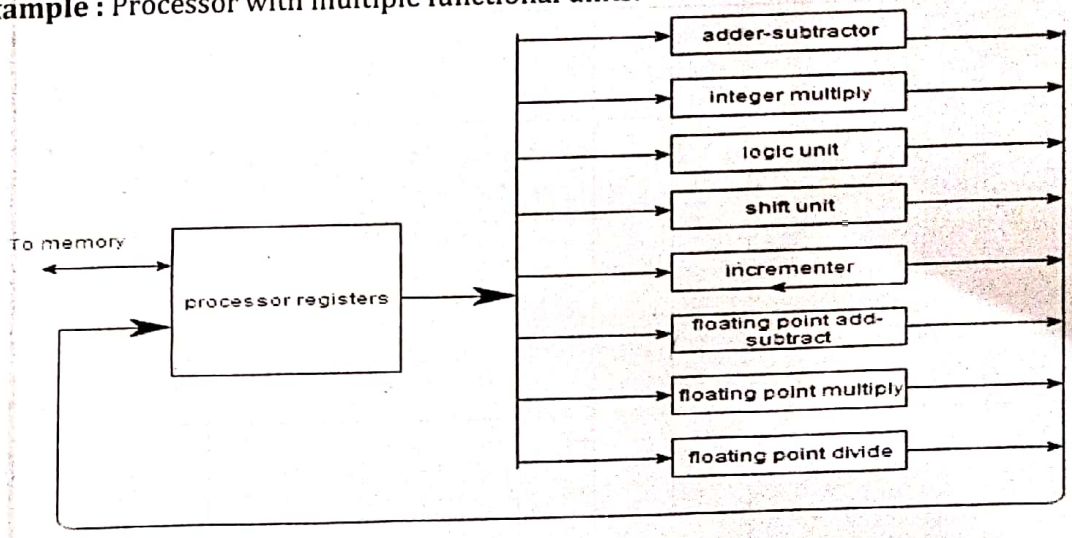
Advantages of RISC

- VLSI Realization
- Computing Speed
- Design Costs and Reliability
- High Level Language Support

Parallel Processing

Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneously data-processing tasks for the purpose of increasing the computational speed of the computer. Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time. The system may have two or more processor operating concurrently. The purpose of parallel processing is to speed up the computer processing capability and increase its throughput. The amount of hardware increases with parallel processing, hence the cost increases.

Example : Processor with multiple functional units.



PROCESSOR WITH MULTIPLE FUNCTIONAL UNITS

It shows one possible way of separating the execution unit into eight functional units operating in parallel. The operands in the registers are applied to one of the units depending on the operating specified by the instruction associated with the operand.

Addressing Modes

↓

(Operation code) opcode — rules for interpreting the add^{r} field of instruction.

10

① **Implied mode :-** (Implicit mode)

→ operands are specified implicitly

e.g. "Complement accumulator"

contain operand.

Eg:- HLT
CMA
RET

② **Immediate mode :-**

Opcode	Add^{r}

operand here.

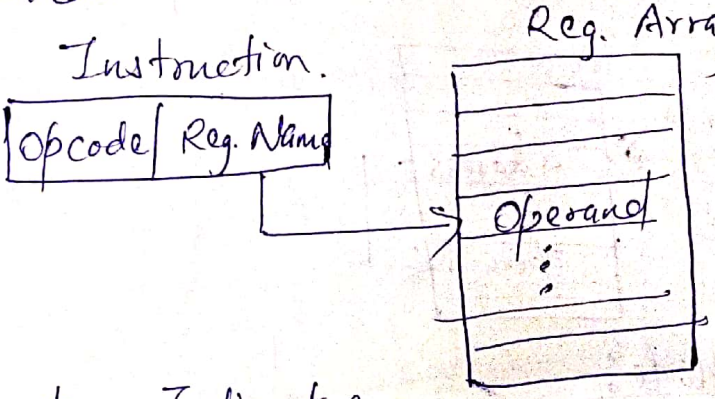
:: | Opcode | Operand |

there is no need to specify the address of operand.

Eg:- MVI A, 100
LDI 100,

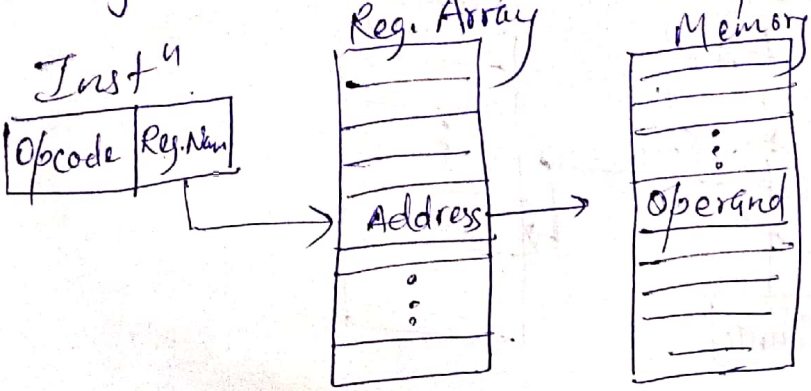
③ **Register Mode :-** (OR Register Direct mode)

→ operands are present in register



④ **Register Indirect :-**

→ register will contain an address of effective add^{r}, operand.

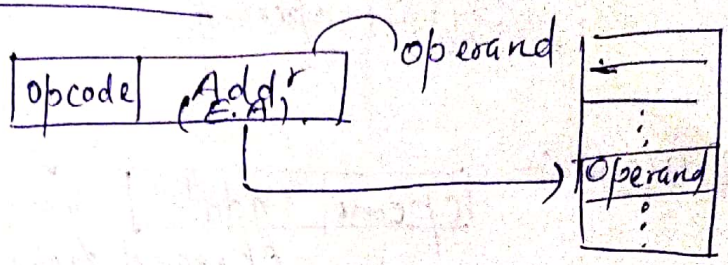


(5) Auto Increment / Auto decrement mode :-

The value of Reg. will be incremented by 1 but after the execution of instruction.

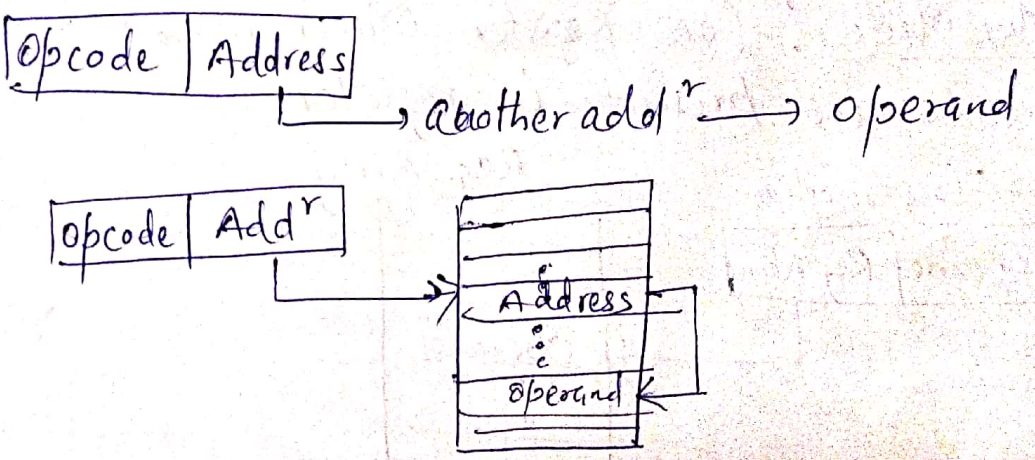
value of Reg. is decremented by 1 before the execution of ~~register~~ instruction.

(6) Direct add^r mode :-



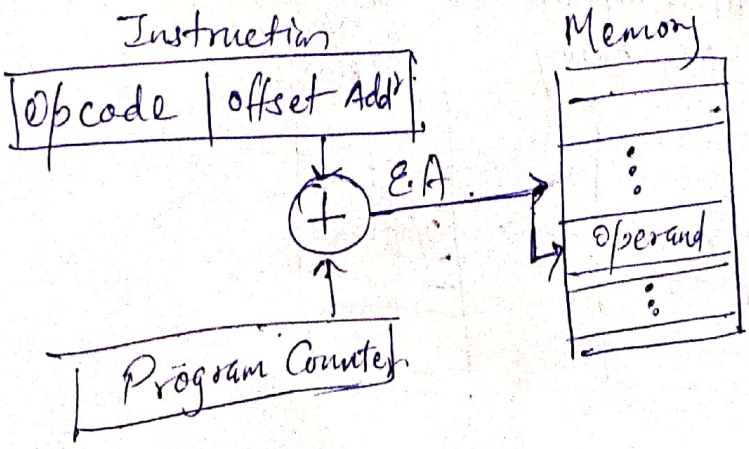
Eg:- LD R1, X

(7) Indirect add^r mode :-



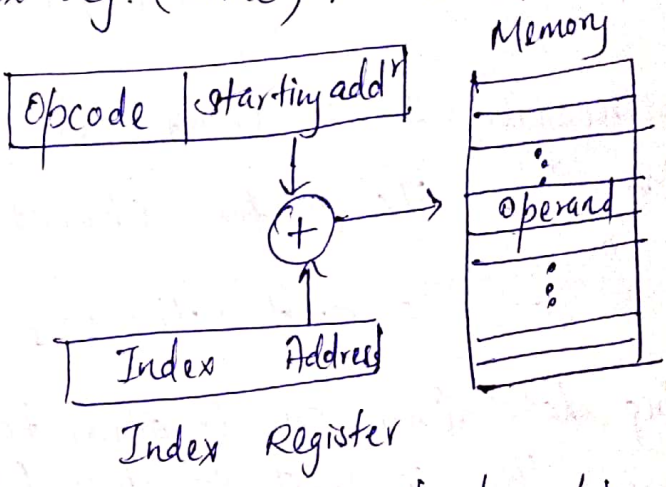
(8) Relative add^r Mode :-

PC + add^r field value of Inst^n = effective add^r.



(9) Indexed Addressing mode:-

Index reg. (value) + add^{r} field = E.A.



The add^{r} field of the instruction provides the starting add^{r} of a data array in memory. The distance b/w starting add^{r} & the address of operand is the index value stored in index register.

(10) Base add^{r} Mode:-

Value of Base reg. + add^{r} field = E.A.