ISA (32 bit machine)

	Opcode	Function	Example
1)	IN	Input from user	IN X
2)	OUT	Output	OUT X
3)	JMP	Jump to	JMP LABEL
4)	HLT	Halt the program	HLT
5)	JNE	Jump if not equal	JNE LABEL
6)	JEQ	Jump if equals	JEQ LABEL
7)	CMP	compare	CMP R_1, R_2
8)	JLT	Jump if less then	JLT LABEL
9)	JGT	Jump if greater then	JGT LABEL
10)	ADD	add value	ADD R_1 ; $R_2 \Rightarrow R_2 = R_1 + R_2$
11)	SUB	Subtract values	SUB $R_1, R_2 \Rightarrow R_2 = -R_1 + R_2$
12)	MUL	Multiply value	MUL $R1, R_2 \Rightarrow R_2 = R_1 * R_2$
13)	DIV	Divide value	DIV $R_1, R_2 \Rightarrow R_2 = R_2 / R_1$
14)	INC	add value by 1	INC $R_1 \Rightarrow R_1 = R_1 + 1$
15)	DEC	subtract value by 1	DEC $R_1 \Rightarrow R_1 = R_1 - 1$
16)	OR	logical OR	OR $R_1, R_2 \Rightarrow R_2 = R_1$ OR R_2
17)	AND	logical AND	AND $R_1, R_2 \Rightarrow R_2 = R_1$ AND R_2
18)	XOR	exclusive OR	XOR $R_1, R_2 \Rightarrow R_2 = R_1$ XOR R_2
19)	NOT	1's complement	NOT $R_1 \Rightarrow R_1 =$ NOT R_1
20)	SWP	exchange	SWP R_1, R_2
21)	POP	POP from stack	POP R_1
22)	PUSH	PUSH into stack	PUSH R_1
23)	RET	RETURN from Procedure	RET
24)	TEST	Boolean AND	TEST R_1
25)	SHR	logical right shift	SHR \$1, R2
26)	SAL	logical shift left	SAL \$1, R2

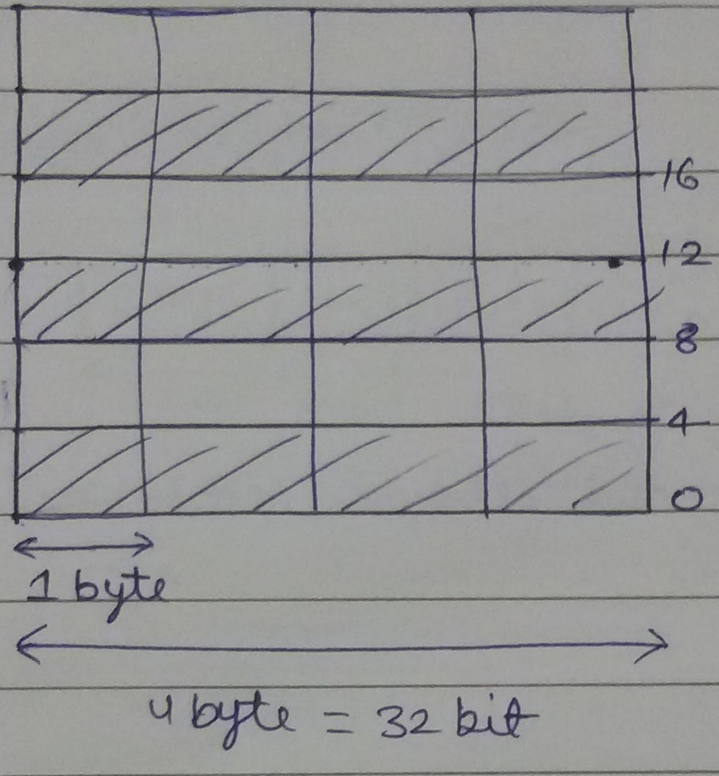
~~27) LEAVE~~ ~~undo stack frame~~ ~~LEAVE~~

28) MOV mov value MOV R1 R2

2) Memory model :

We are using alligned memory model.

→ Because it is structured memory model and helps us to retrive data fastly.



3) Registers :-

- special Purpose Register
  - stack Pointer(SP), Base Pointer(BP), PC.
  - Flags - zero, carry, sign (3 register)

- General purpose Registers : 20 ($R_0 - R_{19}$)
  - 5 bits for addresses

R_0 = 00000
R_1 = 00001
⋮
R_9 = 01001
⋮
R_{19} = 10011

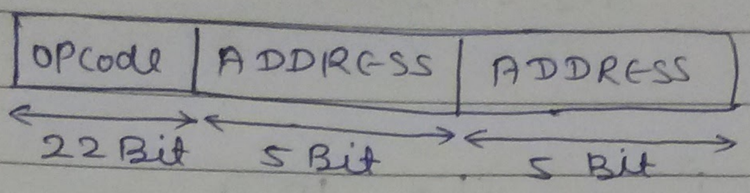
4) Instruction Design :

2a) opcode length (in bit) = variable length according to address instruction

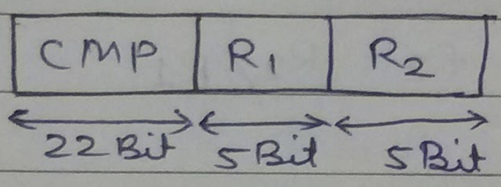
address length = 5 bit (for each register)

Instruction format :-

1) One opcode & 2 address instruction :

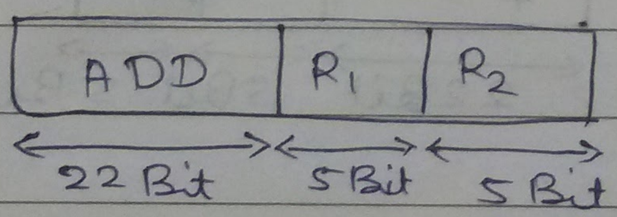


i) CMP:



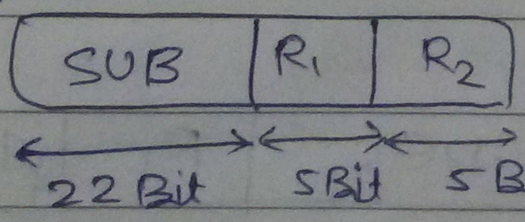
This instruction compares two values: R_1 and R_2.

ii) ADD :



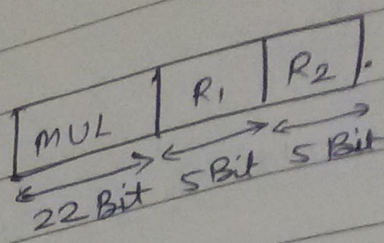
Ex. $R_2 = R_1 + R_2$

iii) SUB :



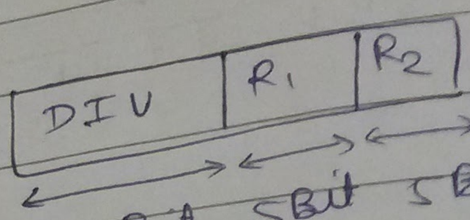
Ex. $R_2 = R_1 + R_2$

(iv) MUL :



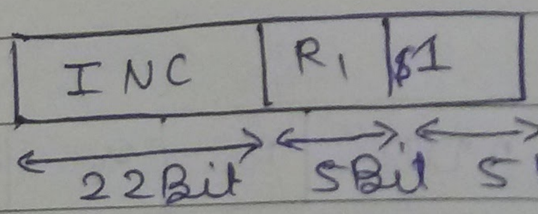
$R_2 = R_2 * R_1$

(v) DIV :



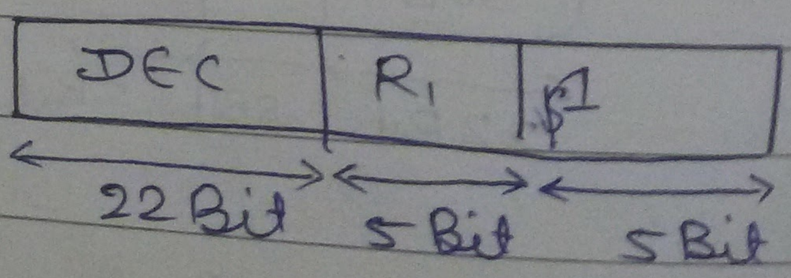
$R_2 = R_2 / R1$

(vi) INC :



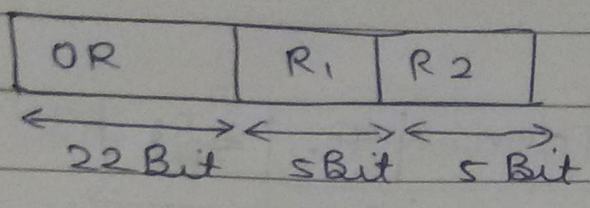
$R_1 = R_1 + 1$

(vii) DEC :



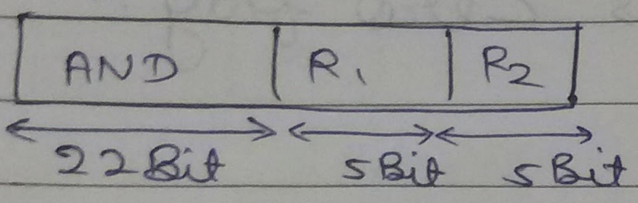
$R_1 = R_1 - 1$

(viii) OR : Logical OR



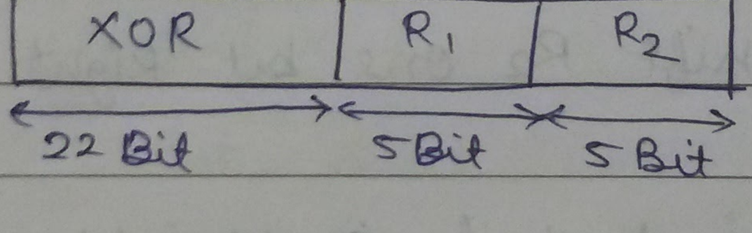
$R_2 = R_1$ OR R_2

(ix) AND: Logical AND



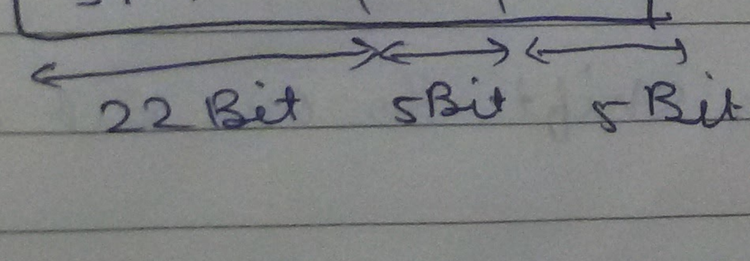
$R_2 = R_1$ AND R_2

(x) XOR : xclusive OR



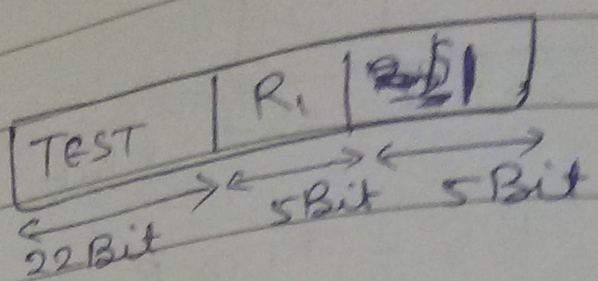
$R_2 = R_1$ XOR R_2

(xi) SWP : exchange



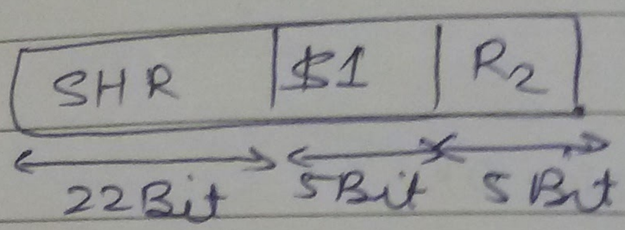
exchange values of R_1 and R_2

(xii) TEST : [redacted]



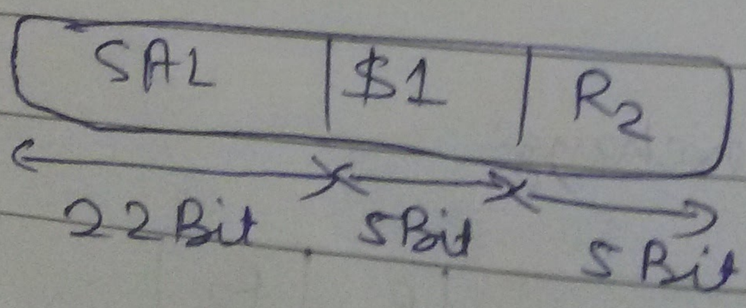
Test R1, $1
Generates zero flag.

(xiii) SHR : logical right shift



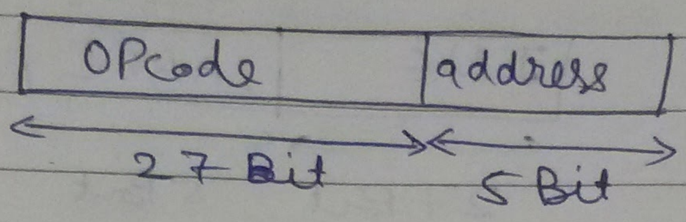
Shift R2 one bit Right.

(14) SAL : logical shift left

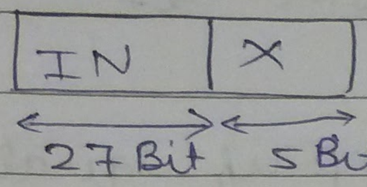


Shift R2 one bit left

(2) One Opcode & 1 address Instruction :-

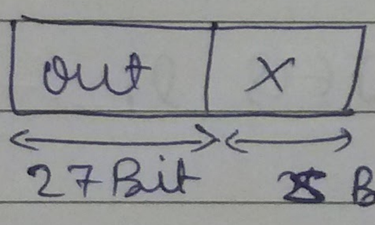


(1) IN : Input



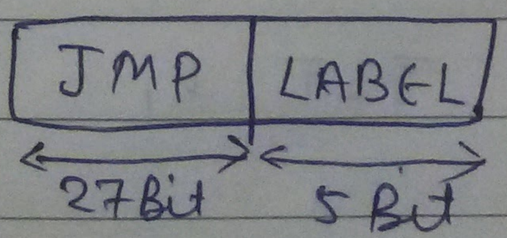
Ex. IN X IN Reg0, Portadd

(2) OUT : Output



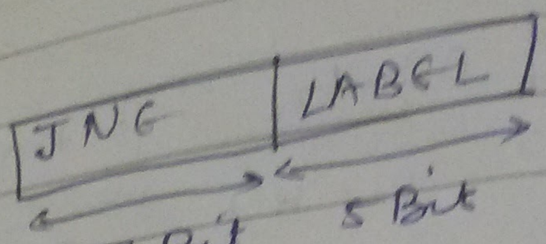
Ex. out X

(3) JMP : Jump to



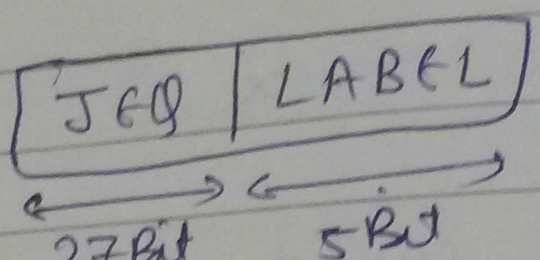
Ex. JMP Hello

(4) JNE : Jump if not equal



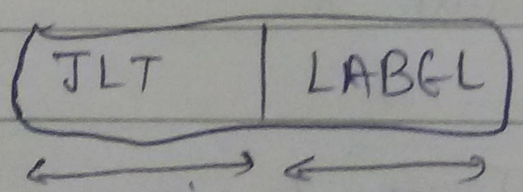
Ex JNE l1

(5) JEQ : Jump if equals



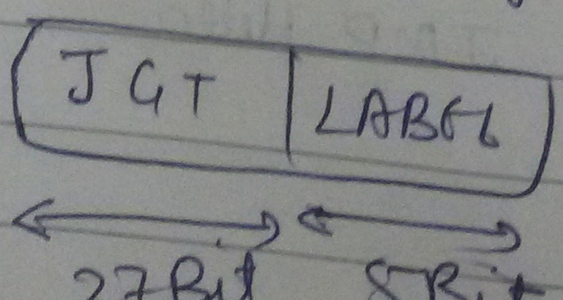
Ex. JEQ l1

(6) JLT : Jump if less then



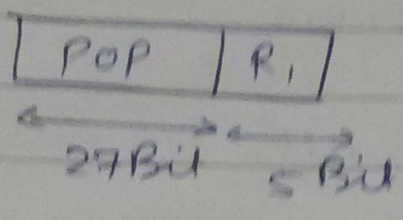
Ex. JLT l1

(7) JGT : Jump if greater then



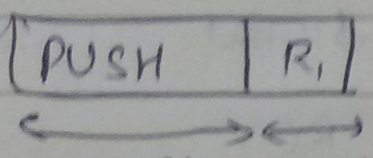
Ex JG7 R1

(8) POP : Pop from the stack.



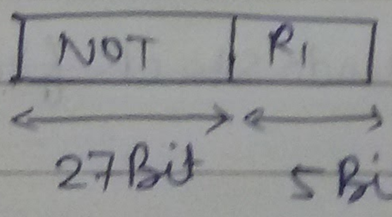
→ it POP R1 from the stack

(9) PUSH : Push into the stack



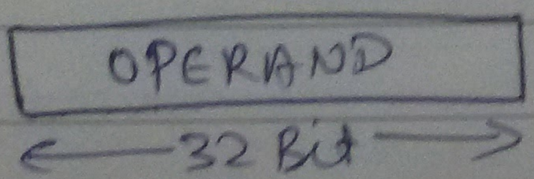
→ it R1 into the stack

(10) NOT : 1's complement

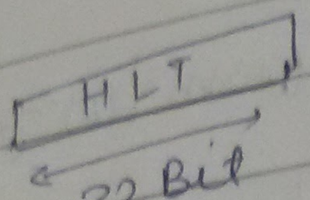


→ it gives R1's 1's complement

(3) One operand & Zero address instruction:

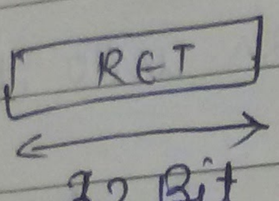


1) HLT : Halt the Program



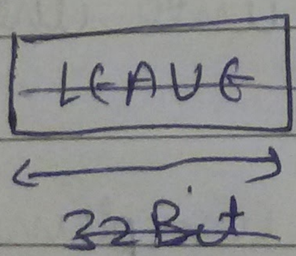
Ex Hlt

2) RET : Return from Procedure



Ex RET

3) LEAVE :-



→ Undo Stack frame

Opcode & Its Binary Expansion :-

CMP ⟶ 0 ---- 0 00
(20 zeros)

ADD ⟶ 0 ---- 0 01
(20 zeros)

SUB ⟶ 0 ---- 0 10
(20 zeros)

MUL ⟶ 0 ---- 0 11

DIV ⟶ 0 ---- 0100
(19 zeros)

INC ⟶ 0 ---- 0 101
(19)

DEC ⟶ 0 ---- 0110
(19)

OR ⟶ 0 ---- 0111
(19)

AND ⟶ 0 ---- 01000
(18)

XOR ⟶ 0 ---- 01001
(18)

SWP ⟶ 0 ---- 01010
(18)

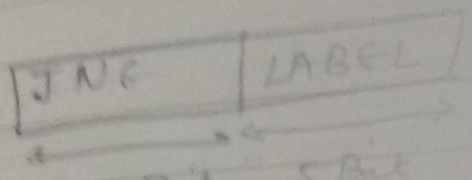
TEST ⟶ 0 ---- 01011
(18)

SHR ⟶ 0 ---- 01100
(18)

SAL ⟶ 0 ---- 01101
(18)

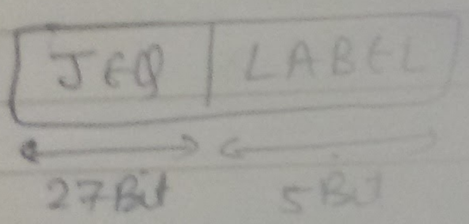
22 Bit length

(5) JNE : Jump if not equal



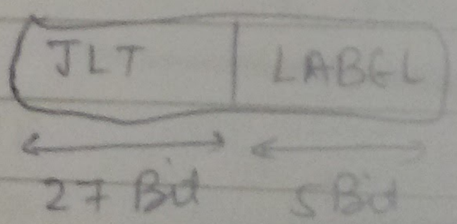
Ex JNE l1

(6) JEQ : Jump if equal



Ex. JEQ l1

(6) JLT : Jump if less



Ex. JLT l1

(7) JGT : Jump if greater

MOV → 0-------01110 } 22 Bit length

IN → 0-------0111100000

OUT → 0-------0111100001

JMP → 0-------0111100010

JNE → 0-------0111100011

JEQ → 0-------0111100100

JLT → 0-------0111100101

JGT → 0-------0111100110

POP → 0-------0111100111

PUSH → 0-------0111101000

NOT → 0-------0 1111 01001

HLT → 0-------0 1111 01010 00000

RET → 0-------0 1111 01010 00001

~~LEAVE~~ → ~~0-------0 1111 01010 00010~~

27 Bit length

7)

b)

c)

d)

e)

8)

a)

b)

7) Addressing Mode :-

a) Direct Addressing Mode →
Eg. lea (load effective address)

b) Immediate Addressing Mode →
Eg. MOV $23, R02

c) Register Addressing mode
Eg → Sub R04, R05
R05 = R05 - R04

d) Register indirect Addressing Mode →
Eg → [R05], R06

e) Based Indexed Addressing Mode
MOV R5, [R1+R06]
Index
MOV R1+R06], R5

8) Instruction Type →

a) Data Movement :
Reg → Reg
Reg → Mem
Mem → Reg

b) Dyadic Operations :
MOV E1 R2

c) Monadic operation
Operation which require one operand
Ex → SAR R1

d) Conditional type
Ex → JNE L0

e) Produre Call Instruction
Ex → CALL GCD

g) Flow of Control Handling →
Flow of control in our program is distri
by operation like JNE, CALL, RET etc.