CAO

Components of Processor : IR – Instruction register.
PC – Program control
MAR – Holds address of location to be accessed.
MDR – Contains DATa to be written or read out.
GPR – Store operand temporarily

1. X → PC (address of instr^n is given to PC)
2. PC → MAR
3. Read (contents of memory loc^n by MAR read ⇒ MDR] ⇔ CU
4. PC ← PC+1 (next instr^n ptr)
5. MDR (contain instr^n currently) → IR & decode
6. LOC A → MAR
7. Read [by CU]
8. MDR (operand) → ALU
9. R_0 → ALU
10. CU → ADD(ALU) control signal sent to ALU.
11. ALU → Accumulator → (R_0)

X aDD LOCA, LOC B

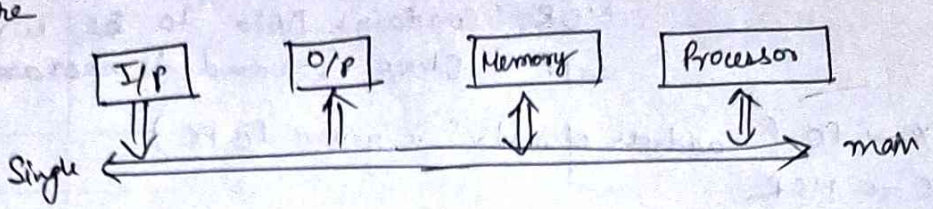
1. X → PC
2. PC → MAR → MDR (Inst^n)
3. Read → [MDR] ⇔ CU
4. PC ← PC+1
5. MDR → IR & decoded
6. Access operand
   LOCA → MAR
7. Read (by CU) (operand → MDR)
8. MDR (operand) → ALU
9. LOC B → MAR
10. Read (by CU) (operand → MDR)
11. MDR → ALU
12. CU → ADD(ALU)
13. ALU → Accumulator
14. Accumulator → MDR
15. Loc B → MAR
16. Write

Add LocA, R0, LocB Add elements stored at LocA & R0 & store at LocB.

Interrupt & Interrupt service routine

Bus structure

Single bus



Compile Use to convert HighLevel L to machine code.
Check for semantic & syntactic error in a program.

Buffer

Processor clock

Basic Performance eqn T – Processor time required to execute
N – No. of actual machine lang. inst.
S – average no. of basic step.
R – Clock rate

$$T = \frac{N \times S}{R}$$

How to improve T? Reduce value of T.

Ch-1 hardware

Machine Instructions

Signed & Unsigned no.

Big- Endian & Little - Endian

Big- Endian	Little - Endian
0 1 2 3	3 2 1 0
4 5 6 7	7 6 5 4

Bit & Byte addressable.

Memory Operation 1. Load (or Read / Fetch) 2nd. Store (Write)

Register transfer notation: Identify location by a symbolic name standing for its hardware binary address.

for memory location: LOC, PLACE, A, VAR2

" Processor registers: R0, R1

" I/O Registers: Datain, outstatus

Contents of a location are denoted by placing square brackets around the name of the location.

$R_1 \leftarrow [LOC]$ $R_3 \leftarrow [R1] + [R2]$

This type of notation is known as Register transfer Notation (RTN).

CPU organization

Single Accumulator

Assembly lang. Notation

To represent machine instruction & Program.

Move LOC, R1 $R_1 \leftarrow [LOC]$

Add R1, R2, R3 $R_3 \leftarrow [R1] + [R2]$

Basic Instruction Types

3 Address inst^n	ADD R1, R2, R3	;	$R_3 \leftarrow R_1 + R_2$
2 Address inst^n	ADD R1, R2	;	$R_2 \leftarrow R_1 + R_2$
1 Address inst^n	ADD A	;	$AC \leftarrow AC + M[A]$
0 address inst^n	ADD	;	$TOS \leftarrow TOS + (TOS-1)$

Top of stack

Load (A) → source. Contents of A copies to accumulator.

Store (A) → destination. Contents of Acumulator placed in A.

One & half address instruction

1 Address always refers to main mem & 1 in reg or both registers.

Evaluate C = A+B

3 address – ADD A, B, C

2 address – ADD A, B
MOVE B, C

1 address – LOAD A
ADD B
STORE C

Evaluate E = (A+B) * (C+D)

3 address
ADD A, B, R1
ADD C, D, R2
MUL R1, R2, E

2 address
ADD A, B
ADD C, D
MUL B, D
MOVE D, E

MOV A, R1
ADD B, R1
MOV C, R2
ADD D, R2
MUL R1, R2
MOV R2, E

LOAD A
ADD B
STORE R1
LOAD C
ADD D
MUL R1
STORE E

Complete it.

Instruction execution & straight-line sequencing

$$Z = [(A+B-C) * (C-D+E)] / (A * B-C)$$

```
MOV A, R1
ADD B, R1
MOV C, R2
ADD E, R2
SUB R1, C
SUB R2, D
MUL C, D
PUSH
```

```
PUSH C
PUSH A
PUSH B
POP C
MOV C, R1
```

Zero address

```
PUSH A
PUSH B
ADD
PUSH C
SUB
PUSH C
PUSH D
SUB
PUSH E
ADD   // C-D+E
MUL
PUSH A
PUSH B
MUL
PUSH C
SUB
DIV
POP Z
```

1 address

```
LOAD A      [A] → Acc
ADD B       [B] + [Acc] → Acc
Store X     [Acc] → [X]   A+B
Load C      [C] → Acc
SUB X       [X] - [C] → Acc
Store X     [Acc] → X
Load D      [D] → Acc
Sub C       [C] - [Acc] → Acc [C-D]
Add E       [E] + [Acc] → Acc [C-D+E)
MUL X       [X]* [Acc] → Acc [[ ) + ( ]]
Store X     [Acc] → X
Load A      [A] → Acc
MUL B
Store Y
Load C
Sub Y
Add E
Store Y
DIV X
```

```
DIV X
Store Z
```

2 address

```
MOVE B, Y
Add A, Y
Move C, X
Sub Y, X
Sub C, D
Add D, E
MUL X, E
MUL A, B
Sub B, C
DIV E, C
MOVE C, Z
```

Three address

```
ADD A, B, X
SUB X, C, X
SUB C, D, Y
ADD X, Y, X
MUL A, B, Y
SUB Y, C, Y
DIV X, Y, Z
```

* **Branching**: Consider task of adding list of n numbers.

Write program to perform.

$C = A_1 B_1 + A_2 B_2 + \dots + A_n B_n$

```
MOVE  n, R1      ⟶ nos added / terms.
MOVE  #1, i      ⟶ # Referring to a no/constant
                    i ⟶ index
Clear R0, R2
MOVE  Ai, R0
MUL   Bi, R0
ADD   R0, R2
INC   i
DEC   R1
R1>0  goto 1012
MOVE  R2, C
```

Complete it.

* **Condition codes**: Four used flag. N (negative), Z (zero), V (overflow), C (carry) ← set to 1 if arithmetic overflow occurs.

For ex:
Signed no. & unsigned no,
if get overflow while doing arithmetics.

Addressing Modes	Syntax	Addressing func:
Immediate	#value	Operand = value
Register	Ri	EA = Ri
Absolute (direct)	LOC	EA = LOC
Indirect	(Ri)	
Index	LOC	
Base with index	X(Ri)	
Base with index & offset	(Ri, Rj)	
Relative	X(Ri, Rj)	
Auto increment	X(PC)	
Auto decrement	(Ri)+	
	-(Ri)	

Indirect : Add (R1), R0

```
MOV     N, R1
MOV     #NUM1, R2
CLEAR   R0
ADD     (R2), R0
Add     #4, R2
decrement   R1
Branch>0    LOOP
MOVE    R0, SOM
```

Addressing Modes

① direct ② indirect ③ index mode ④ Immediate ⑤ Register

Relative address ⇒ Effective address = Program counter + Relative address

① Auto increment & Auto decrement modes.

OP code	Register notating	Memory location
MOV	R_1	LOC A
ADD	R_2	LOC B
INC		
BR.		

Program written in assembly language converts to set of machine inst.

* by assembler

Name	Mnemonic
LOAD	LD
Store	ST
MOVE	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Clear	CLR
Complement	COM
AND	AND
OR	OR.
Exclusive OR	XOR
clear carry	CLRC
Set carry	SETC
Complement Carry	COMC
Enable interrupt	EI
Disable interrupt	DI
Return	RET
Compare	CMP
Test	TST
Increment	INC
Decrement	DEC
Add	ADD
Subt.	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subt. with borrow	SUBB
Branch if zero	BZ
Branch if not zero	BNZ

Assembler Directives : The assembly lang (completely from PPT)

Stack & Queues

Subroutines / Nesting of Subroutines

Frame pointer

ALU

General Purpose register (GPR)

Special Registers → Memory address register (MAR)
→ Memory data register (MDR)

MAR → is used to read/hold the address

MDR → is used to read/hold/write data.

operation operand 1, operand 2 — New Teacher

Dest / Source ↙ SRC2

Dest/SRC1

Add LOCA, R0

① Get operand's value into ALU.

a) Register → ALU

b) MAR → MDR → ALU

Read operation

① Register
② Memory location

② Perform the operation

③ Store the results.

a) Reg: ALU → Register

b) Memory: ALU → MDR → (WB, Write Back) → MAR

operand 1 → Reg / Memory

Example

Add LOC A, R0

→ Instruction PC(2000)

PC → 2000 — Program counter

2000: Add LOCA, R0

MAR

MDR [2000] — MAR: 2000

Content inside address 2000.

IR → ⊕ — Add LOC A, R0

LOCA → MAR → (Read) MDR → ALU

R0 → ALU

ALU → MDR → (WB) MAR (LOCA)

PC = PC+1

Ex:- Add R0, R1

Add R0, LOCA (HW)

Add LOCA, LOC B (HW)

Add R0, R1 :-

PC → MAR → (read) MDR → IR → ⊕ → LOCA → ALU

R0 → ALU

ALU → MDR → (WB) MAR (R0) → PC = PC+1

* Number system Representation

$B = b_n b_{n-1} \ldots b_1 b_0$

+ve → 0
-ve → 1

→ 1's & 2's complement

-3 → +ve number (+3)

0 011

1100 1's complement

+1

1101 2's complement

$-3 =$ 1101

+5 → 0101

-5 → 1101

0101

1010 1's complement

+1

1011 2's complement → -5

* Arithmetic operations :-

Ex : Add +7 and -3.

0111 → +7

+ 1101 → -3

① 0100 → +4

Carry bit

ignore the carry signal.

If ans. lies within the range then accept it otherwise cond^n of overflow will occur.

* Rules in 2's complement Arithmetic :

① Addition Add two no's in their N bit representation & ignore carry bit signal from MSB.

If ans lies -2^{n-1} to $+2^{n-1}-1$ range then accept ans overflow condition occurs.

if $n=3$ -2^2 to $2^{n-1}-1$ $\quad n \rightarrow$ no. of bits.

-4 to 3

② Subtraction $x - y$

Add $x + (-y)$

↳ 1's complement → 2's complement

-2^{n-1} to $2^{n-1}-1$ accept ans otherwise reject.

Arithmetic operation using 2's complement

Q. Add +2 and +3

$$\begin{array}{r} 0010 \\ 0011 \\ \hline +5 = 0101 \end{array}$$

Add +4, −6

Subt, −3, −7 (HW)

$n = 4$

-2^{n-1} to $+2^{n-1}-1$

-2^3 to 2^3-1

-8 to 7

Q. $+4 \rightarrow 0100$

$+6 \rightarrow 0110 \rightarrow$ 2's complement $\rightarrow 1001$

$$\begin{array}{r} 1001 \\ +1 \\ \hline 1010 \end{array}$$

$+4 \rightarrow 0100$

$-6 \rightarrow 1010$

$$\begin{array}{r} 0100 \\ 1010 \\ \hline (-2)\ (1)110 \end{array} \rightarrow \text{2's complement} \quad \begin{array}{r} 0001 \\ +1 \\ \hline 0010 \end{array}$$

MSB is (1) so take 2's complement and put $\ominus$ ahead of this.

$+2$

So ans is -2

Q. Subtract $+2, -4$

0010

$+4 \rightarrow 0100 \Rightarrow$ 2's complement $\rightarrow 1011$

$$\begin{array}{r} 1011 \\ +1 \\ \hline 1100 \end{array}$$

= H.W.

Q Add $+2, -7$

0010

$+7 \rightarrow 0111 \rightarrow$ 2's complement $\rightarrow 1000$

$$\begin{array}{r} 1000 \\ +1 \\ \hline 1001 \end{array}$$

$+2 \Rightarrow 0010$

$-7 \Rightarrow 1001$

$$\begin{array}{r} 0010 \\ 1001 \\ \hline (1)0011 \end{array} \Rightarrow \begin{array}{r} 0100 \\ +1 \\ \hline 0101 \end{array}$$

$\ominus 0101 \Rightarrow 5 \quad \ominus 5$

When during arithmetic operation, if one of the operand is −ve then computed answer should follow 2's complement method for verification

Q. Add $-3, -7$

$0011 \quad 0111$

2's comp. $1101 \quad 1001$

$$\begin{array}{r} 1101 \\ 1001 \\ \hline (1)0110 \end{array} \rightarrow \sout{+6}$$

$\ominus$

$= -10$

We can't take 2's complement bcoz MSB is 0.

So we will say this is overflow condition.

Q. Add +3, +7.

```
  0011
  0111
  1010   overflow.
```

When 2 nos having same sign, 2's complement representation system might give incorrect results.

* Memory

Trillions / billions of storage cells used to store the bits.
Access bits in the form of words.

Data is usually accessed N-bit group called word length.

characters

8 bit	8 bit	8 bit	8 bit

ASCII

⇒ Byte addressable memory, WL = 0, 4, 8 ... each word = 4 Bytes.

⇒ K-bit address = 2^K memory location.

32 bit = 2^{32} Mem. Loc^n.

$= 2^{30} \cdot 2^2$

$= 4G$

$2^{10} = 1K$

$2^{20} = 1M$

$2^{30} = 1G$

$2^{40} = 1T$

Memory Representation

↳ Big Endian → Lower bit is used to denote MSB address.
↳ Little Endian → Lower bit is used to denote LSB address.

	W_0	W_1	W_2	W_3
0	0	4	8	12

0	12	8	4	0

25/8/25

Memory operations :-

↳ Load: transfer the contents of memory location to processor register.
↳ Store: transfer the contents of processor's register into memory location

e.g. ~~Op~~ M[A] ← R_1
Store M[A], R_4

Load R, M[A]

ALU
↓
MDR ⇉ Processor reg.
↑WR
MAR

* Instruction & instruction sequencing :-
  1) Data transfer b/w memory location and register.
  2) ALU operations on data.
  3) Program sequencing and control ↳ condition codes
  4) I/O transfers ↳ Branching instructions.

→ 1.1 Register transfer Notation
→ 1.2 Assembly lang. notation

instruction format.

① RTN :-
→ contents inside the register are denoted by placing [ ]

② Assembly lang Not^{n} :- Used to understand LL lang. like machine inst^{n}.

opcode	op1	op2

~~③ ALU operation~~

③ ALU operation

PC
↓
MAR
↓
MDR
↓
IR → operand 1
operand 2
Instruction Register

* In 3 address format, all the address are explicitly defined.
* Here the instruction format has 3 diff. which specify either memory or reg. location.

* 3 address Instruction format ,

opcode	operand 1	operand 2	operand 3 / mode
	M/Reg	M/R	M/R

* Two many bits are required

* 2-Address instruction format :-

opcode	operand 1	operand 2

Ex : Add A, B
M[A] ← M[A] + M[B]

* 1-Address instruction format :- It uses implied Accumulator register for data manipulations.
  ↓
  CPU will use its own acc. reg. so there is no need to specify the address.

e.g. Load A
Acc ← M[A]

* 0- address Instruction

⇒ No operands address

⇒ only opcode → Push / Pop

opcode

→ stack Organization

Disadv. ① we need to convert all inst. to postfix format.

② Requires stack arrangement.

Q. Evaluate the expression (A+B) * (C+D) using 3, 2, 1, 0 Inst^n format.

3 address inst^n

Add R_1, M[A], M[B] → (Req. 1 clock cycle Register computation)

Add R_2, M[C], M[D]

Mul R_3, R_1, R_2

opcode / operand 1 / op. 2 / op. 3

↓ M/R ↓ M/R ↓ M/R

Memory

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

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

$R_3 \leftarrow R_1 * R_2$

Add Z, A, B → (2 clock cycles)

(Memory cycle computation)

Q. (A+B) * (C+D)

2 address instruction format.

~~Add A, B~~

~~Add Mul C, D~~

~~Mul A, C~~

Load R_1, A — $R_1 \leftarrow M[A]$

Add R_1, B — $R_1 \leftarrow [R_1] + M[B]$

Load R_2, C — $R_2 \leftarrow M[C]$

Add R_2, D — $R_2 \leftarrow [R_2] + M[D]$

MUL R_1, R_2 — $R_1 \leftarrow [R_1] + [R_2]$

MOV X, R_1 — $M[X] \leftarrow R_1$

1 address instruction format.

Load A	Acc ← M[A]
Add B	Acc ← [Acc] + M[B]
MOV X	M[X] ← (Acc)
Load C	Acc ← M[C]
Add D	Acc ← [Acc] + M[D]
Mul X	M[X] ← Acc

In stack organisation Push & Pop ~~also~~ will be called along with the address of memory. However in zero add inst, ADD, MUL, SUB will not carry any address.

Push A	TOS ← M[A]
Push B	TOS ← M[B]
Add	TOS ← (A+B)
Push C	TOS ← M[C]
Push D	TOS ← M[D]
Add	TOS ← (C+D)
Mul	TOS ← (A+B) * (C+D)

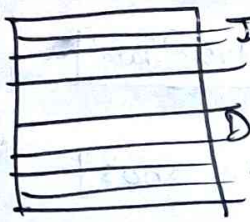
7/9/25 * Instruction Sequencing (

Add C, A, B

↳ mov R1, A — R1 ← M[A]

Add R1, B — R1 ← [R1] + M[B]

mov C, R1 — M[C] ← [R1]



32-bit → 2^{32} Memory loc.

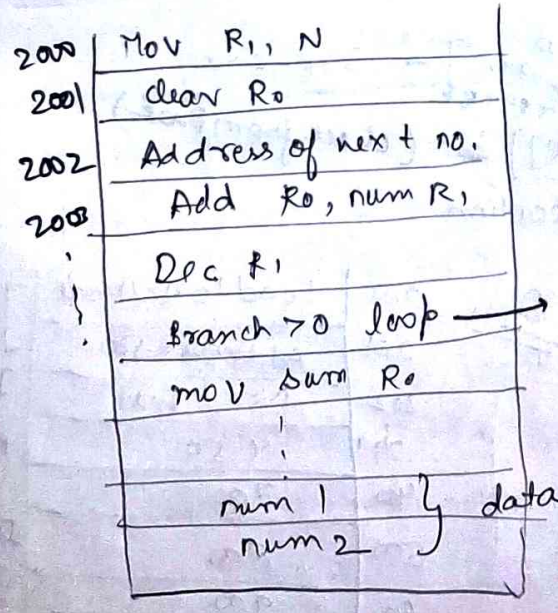
Byte addressable memory

$\frac{32}{8} = 4B$

each word length is 4B.

* Branch instruction :- This type of inst. loads a new address into the program counter. as a result, the processor fetches & executes the instruction at the new address called the branch target.

A condition branch instruction causes a branch only if a specific cond^n is specified. The instruction immediately stop in case of condition failure.



Addressing Modes :-

Representation in which location of an operand is specified within inst^n.

Important terms :- i) displacement → 8/16 bit value
ii) Index Register (SI, DI)
iii) Base register (BX)

① Implied AM → CLC, CMC operand specified in Inst^n itself.
② Immediate AM → data is stored as 8 bit / 16 bit displacement
③ Register direct AM → EA = [R1]
④ Register indirect AM → EA = [[R1]]
⑤ Direct AM → Acc ← m[A]
⑥ Indirect AM → [[ ]]
⑦ Relative AM → [PC] + Address field of the instruction.
⑧ Index AM
⑨ Auto-increment AM
⑩ Auto-decrement AM
⑪ Base-register AM

PC = 2000

opcode	Address/data

= 2000 + 3

EA = [2000 + 3] = [2003]

→ Program relocation
→ Calculation of BTA (Branch target address)

⑧ Index Addressing mode
→ Index Register used to cal EA
EA = [(Address of inst.) + [IR]]

⑨ Auto-increment AM :- $R_{auto} = [R_{auto}] + step\text{-}size$
→ used to perform loops.
↳ depend on the size of operand

⑩ Auto-decrement AM :- $R_{auto} = [R_{auto}] - step\text{-}size$

⑪ Base register AM ⇒ EA = [(Address part) + [PC] + [BR]] ← (check from Book)
↳ array relocation

Q.

MOV	M[500]	Mode

PC =
R1 = 400
IR = 100
stepsize = 1

Q.

200	Load to Ac/Mode	I1
201	Address = 500	
202	Next Inst^n	I2
399	4-50	PC = 200 / 202
400	700	R1 = 400
500	800	IR = 100
600	900	Step Size = 1
702	325	
800	300	

AM		EA	Content	
~~AM~~				
Direct	(1)	M[500]	800	1
Immediate	(0)	—	500	0
Indirect	(2)	[[500]]	300	2
Relative	(0)	[702]	325	1 → EA = [[PC]+500] = [202+500] = [702]
Index	(1)	[600]	900	1 → [100+500]
Register direct	(0)	$[R_1]$	400	0
Register Indirect	(1)	$[[R_1]]$	700	1
Auto increment	(1)	[[R]]+1	701	1 [R_{auto}] + step size
Auto Decrement	(1)	[[R]]-1	699	1

Relative, Index

How many memory references? 1, 0, 2, 0,

Consider a 3-word machine instruction, identify the no. of cycle needed during the execution of the instruction.

A [R_0]

EA = [[IR] + Address field of the inst^n]

EA = [[R_0] + Address] → 1 memory reference is required.

EA = @B = indirect. M[[B]] → 2 memory reference are required.

Add A[R_0], @B

→ 1 → 1 → 2

A [R_0] ← A [R_0] + @ B

Total = 1+1+2 = 4 Memory ref. are required.

Q. Memory locations 1000, 1001 & 1020 have data values 18, 1, & 16 consider following instructions:-

MOVI R_3, 1	Move immediate
Load R_d, 1000(R_s)	load from memory (AM → Index Addressing)
Add I R_d, 1000	Add Immediate
Store I 0(R_d), 20	Store immediat

202

what will be the value in location 1001 after execution?

R_s [1]

R_d ← [1000 + [R_s]]

R_4 ← [1000 + 1]

R_d ← [1001]

$[0+(R_d)] \leftarrow 20$

$[0+100] \leftarrow 20$

Q2. Consider the following memory values & one address machine with accumulator. Calculate the o/p of each instruction.

20	40
30	50
40	60
50	70

Inst^n		o/p
Load	Immediate 20	20
Load	Direct 20	40
Load	indirect @ 20	60
Load	I 30	30
Load	D 30	50
Load	@ 30	70

HW Q3. Consider a RISC machine where each instruction is exactly 4 bytes long. Here, conditional branch instruction with PC relative addressing mode is used. Offset is specified in bytes for the calculation of target location of the branch instruction. Consider the following sequence of instruction

Is. no.	Inst^n
i	Add R_2, R_3, R_4
i+1	Sub R_5, R_6, R_7
i+2	cmp R_1, R_9, R_{10}
i+3	beq. R_1, offset

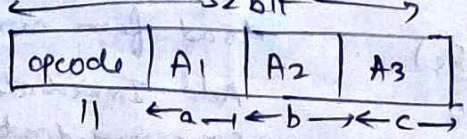
If the target branch inst^n is i, find the value of offset.

EA = [PC] + offset

BTA : i

100	i
104	i+1
108	i+2
112	i+3
116	i+4

* Expanded opcode technique :-



3-address instruction

of 3 AI = x

of 2 AI = y

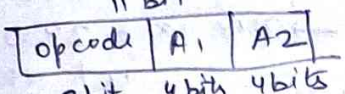
of 1 AI = z

2 AI possible = $(2^{11} - x).2^a$

1 Address inst^n possible $((2^n - x)2^a + -y)2^b$

0 AI possible - $(((2^n - x)2^a - y)2^b - z)2^c$

Q. In an 11 bit comp. inst^n format, the size of address field is 4 bits. The comp. uses expanding opcode technique & contains 5 - two address instruction. & 32 1 address instruction. Calculate the no. of zero address instruction that it can support?



In Question info about 2 AI is given. Hence assume 2 AI.

5 → 2 AI

32 → 1 AI

1 AI possible in the given inst^n format $= ((2^3 - 5)2^4)$

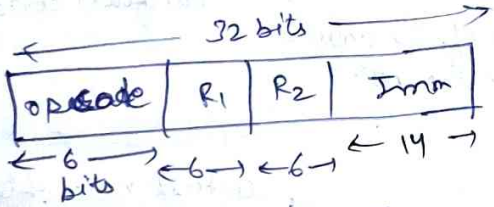
$= 48$

0 AI " " $x = 5$ " " " $\Rightarrow ((2^3 - 5)2^4 - 32)2^4$

$Y = 32$ $16 \times 2^4 = 256$

★ Practice gate exam Question based on this topic.

Q A machine has 32 bit architecture with 1 word long instruction. It has 64 registers each of which are 32 bits long. It needs to support 45 instructions. An immediate operand is also there along with 2 register operands. Assuming immediate operand is an unsigned integer, then find out the maximum of the immediate operand.



$\log_2 45 \simeq 6$

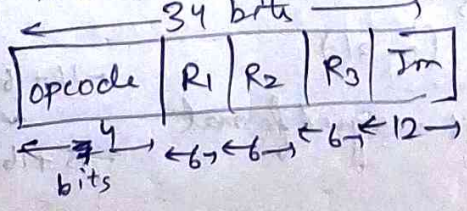
$\log_2 64 = 6$

$R_1 = R_2$ bit

$32 - (6 \times 3) = 14$

$2^{14} - 1 = 16383$ Ans

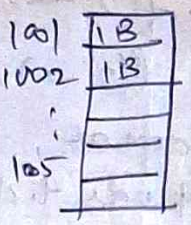
Q Consider a processor with 64 registers & instruction set of size 12. Each inst^n has opcode, 2 registers source, 1 destination register & 12 bit immediate value. Each inst is stored in byte aligned fashion. If a program has 100 instruction. The amount of memory consumed by the program in bytes ------?



~~$\log_2 100 = 7$~~

Opcode $\log_2 12 \simeq 4$

$\log_2 64 = 6$



⇒ 5 Byte will be used for storing

⇒ 1 inst^n → 5 Bytes

⇒ 100 inst^n → 500 Bytes

⇒ If asked in bit → 34 × 100 = 3400 bits (not 4K bits) — Imp.

HW.

Q. A CPU is designed to have 58 - 3 add inst^n., the CPU is able to address a max. of 60 memory location. The length of machine code is same for all the instructions

1) Determine the no. of bits allocated for each address field.

2) " min no. of bits allocated for the opcode field of 3 address instruction.

3) " min no. of bits allocated for the opcode of 2 add inst^n.

** number representation → Sign magnitude

→ IEEE floating point representation.

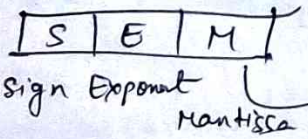
Sign → S → 0 (+ve) | 1 (-ve)

Exponent — no's of 0's power in the no.

-2^{n-1} to $2^{n-1}-1$

overflow condition

* Standard format



Mantissa → actual no. after decimal, in binary

normalised Always.

Implicit Explicit

(i) 0.001

0.1×10^{-2} (left shift of no.)

(ii) 100.1

$0.1001 \times 2^{+3}$ (Right side shift of No)

Exponent in bias form.

Range: -2^{5-1} to $2^{5-1}-1$

-2^4 to 2^4-1

-16 to 15 (0-31)

bias

-16+16 = 0

-15+16 = 1

15+16 = 31

(0-31)

32 bit unsigned format

* If K bits are allocated in E, then bias formula is 2^{K-1}.

* Mantissa → contain fraction part in binary but in normalised form.

101.11 → Explicit (By default) $0.10111 \times 2^{+3}$ $E = e + bias$, $M = 10111$

→ Implicit $1.X \times 2^{+3}$ (X → anything 0/1)

→ In the explicit mode, after the point the first bit should be 1.

→ " " Implicit " , the first bit before the point should be 1 & after the point it can be anything (0/1)

$101.11 \longrightarrow 1.0111 \times 2^{+2}$ $E = 2 + bias$

$M = 0111$ Mantissa

S	E	M
0	3+bias	10111
0	2+bias	0111

$= (-1)^S * 0.M * 2^{E-bias}$

$= (-1)^S * 1.M * 2^{E-bias}$

Q. $(19.25)_{10}$

16 bit

S	E	M
1	6	9

$(10011.01)_2$

→ $M = 0.100|101 \times 2^{+5}$

bias $= 2^{K-1} = 2^{6-1} = 32$

$E = e + bias$

$= 5 + 32 = (37)_{10}$

$E = (100101)_2$

Exponent: binary.

S	E	M
0	100101	100	10100
1	6	9

---

* Addition/ Subtraction of floating numbers :-

(i) check for 0's.

(ii) align with mantissa

(iii) Perform operation

(iv) Normalise the results.

Ex:- Perform addition.

0.583123×10^{3} $\times 10^{-4}$

0.123000×10^{-1} $\times 10^{4} \rightarrow 10^{3}$

0.583123×10^{3} 0.230000×10^{-1}

0.000012×10^{3} 0.123000×10^{-1}

0.583135×10^{3} 0.353000×10^{-1} ✓

1) Less loss of bits chosen during shifting of bits

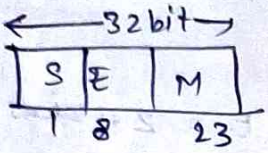
i) In addition, if there is overflow bit
→ Point should be left shift.

$$\begin{array}{r} 0.534 \times 10^3 \\ 0.712 \times 10^3 \\ \hline 1.246 \times 10^3 \\ 0.124 \times 10^4 \end{array}$$

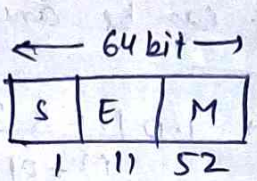
2) In subtraction, if there is underflow bit.
Point should be right shift

$$\begin{array}{r} 0.534 \times 10^3 \\ -\ 0.521 \times 10^3 \\ \hline 0.013 \times 10^3 \end{array}$$

IEEE-754 standard → (Implicit mode only)



Single precision
bias = 127



double precision
bias = 1023

	S	E	M	Number
Special Cases	0	000-000	000-000	+0
	1	000-000	000-000	-0
	0	111-111	000-000	+∞
	1	111-111	000-000	-∞
	0/1	111---111	M≠0	not a number
	0/1	000--000	M≠0	denormalised no.
	0/1	E≠00-00 11--11	M=XXX	number (Implicit mode)

not a number
E = 111 - -111

range : 0 to 255

E = e + bias
e + 127
= 128 + 127
E = 255

255 can't be achieved because e has to be 128 as per IEEE standard.

(e: range : -2^{8-1} to $2^{8-1}-1$
-2^7 to 2^7-1
-128 to 127)

* Denormalised number :- This is very small number which can't be normalised.

We can't store -ve negative bcoz only unsigned integers are allowed. Therefore, IEEE 754 standard says, for a very small number normalise it till ±126.

Implicit Mode

0.000-1111

Case 1

1.1×2^{-126}

$M = 1$

$e = -126$

$E = e + \text{bias}$

$= -126 + 127$

$= 1$

$(-1)^S \times 1.M \times 2^{-126}$

(normalised

Case 2

0.11×2^{-126}

$E = (00000000)_2$

$(-1)^S \times 0.M \times 2^{-126}$

(Denormalised number)

(if unmentioned, use single precision)

Q. $\underset{S}{0}\ \underset{E}{00000000}\ \underset{M}{1100\text{——}000}$ into IEEE-754 standard.

$M = 1100\text{——}000$

$E = 0$

$= (-1)^S * 1.1100\ldots * 2^e$

$1.1100\ldots * 2^{-127}$

$=$

$0 + \text{bias} = 127$

$E = e + \text{bias}$

$e = 0 - 127$

$e = -127$

Q. ~~Solve~~ Store the given no. in IEEE754 standard format in single precision.

$(-14.25)_{10} = (1110.01)_2$ → Implicit mode

$\Rightarrow 1.1101 \times 2^3$

$\Rightarrow M = 11001 \quad e = 3 \quad \text{bias} = 127$

$\Rightarrow E = e + \text{bias} = 130 = (10000010)_2$

S	E	M
1	1000 0010	1100 1

① ⑧ ㉓

$\underset{C}{1100}\ \underset{1}{0001}\ \underset{6}{0110}\ \underset{4}{0100}\ \underset{0}{000}\ 000$

$= C16400\ldots$

Q. from given IEEE754 rep, find the no. in decimal.

$\underset{S}{0}\ \underset{E}{1000\ 0011}\ \underset{M}{1100\ldots00}$

1	8	23
S	E	M

$1.1100\text{——}00 \times 2^e$

$1.11000\text{——}00 \times 2^4$

$(11100)_2 = (+28)_{10}$ ⟶ sign bit = 0

$e = E - \text{bias}$

$e = 131 - 127 = 4$

Q: $(1.0)_{10}$ into IEEE 754 format

S=0 ↳ 1.0×2^0 E = e + bias = 0+127 = $(01111111)_2$

0	011 11111	00000...

E = $(127)_{10}$

⇒ (3 F 8 0 0 0 0 0)$_H$

Q What can be max value which can be represented in IEEE 754.?

Q. What can be min value which can be represented in IEEE 754?

Q. 0 1 000 0000 | 11000.... 00

S ← E → | ← M →

M = 11000....

⇒ $1.11000... \times 2^e$

⇒ $1.11000... \times 2^1$

⇒ $(11.100...)_2$

⇒ $(3.5)_{10}$ Ans

E = $(10000000)_2$

= $(128)_{10}$

e = E − bias = 128 − 127 = 1