Memory operations

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

$R_1 \leftarrow M[A]$

Load R_1, $M[A]$

ALU → MDR → WB

MAR

$M[A] \leftarrow [R_1]$

Store $M[A]$, R_1

Instruction and Ins^n sequencing:

i) Data transfer between memory location and register.

2) ALU operations on data

3) Program sequencing and control

4) I/O transfers.

- 1.1 Register Transfer Notation
- 1.2 Assembly lang. notation

Instruction format

conditional codes

Branching Instructions

1.1 RTN

→ contents inside the Register is denoted by placing [ ]

$$M/R \leftarrow [R_1]$$

1.2 Assembly lang. notation

→ used to understand low-level lang like Machine Instructions

Operation	Destination	Source
opcode	operand	operand

2) ALU operations on data.

Instruction format:

IR: variable size

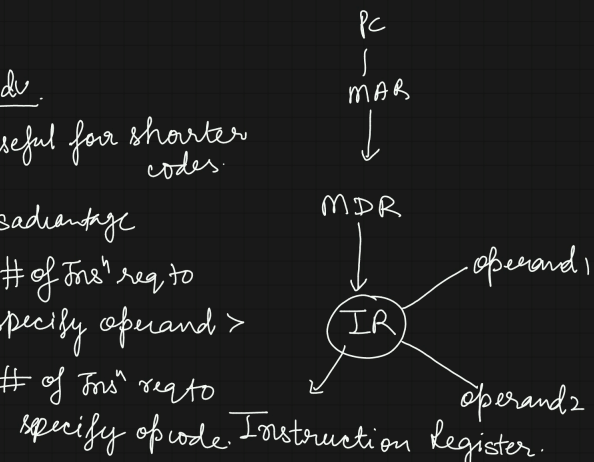
→ 3, 2, 1, 0 address Ins^n.

Adv.

1) useful for shorter codes.

disadvantage

1) # of Ins^n req to specify operand > # of Ins^n req to specify opcode.



3-Address Instruction format:

opcode	operand1	operand2	operand3	mode
	M/R	M/R	M/R	

Ex:

Add R_1, A, B

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

2- Address Ins" format

Ex:

Add A, B

A] ← M[A] + M[B]

Add A, B

1- Address Instruction format.

→ It uses implied Accumulator (operand1).
Register for data manipulation.

→ CPU will use its own Acc. Reg, so there is no need to specify the address.

Ex:

Load A

Acc ← M[A]

0- Address Ins"

→ no operand's address.

→ only opcode.

Example:

Push
Pop

→ Stack organization → Push, Pop.

Push Acc

→ Add

Push.

→ fast

→ less no. bits are required.

+ X 3, 2, 1

3 X 2 + 1

-7-

Q1. Evaluate the Expression $\underbrace{(A+B)}_{R_1} * \underbrace{(C+D)}_{R_2}$ using 3, 2, 1, 0 Ins^n format.

↓ Memory

Sol^n

3-address Ins^n.

Ins^n:

Add R_1, M[A], m[B]

Add R_2, m[c], m[D]

Mul R_3, R_1, R_2

notations

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

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

$R_3 \leftarrow R_1 * R_2$

| opcode | operand 1 | op. 2 | op. 3 |

m/R m/R m/R

Add, A, B

R_1 R_2

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

2-address Ins^n format

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	$M[X] \leftarrow [R_1]$

Add A, B

Mov

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

1 address Ins^n

2

Add A, B 100

1	Load A	$ACC \leftarrow M[A]$
2	Add B	$ACC \leftarrow [ACC] + M[B]$
3 →	Mov X	$M[X] \leftarrow [ACC]$
4 →	Load C	$ACC \leftarrow M[C]$
5	Add D	$ACC \leftarrow [ACC] + M[D]$
	Mul X	$M[X] \leftarrow ACC * M[X]$

Mul (Acc)

opcode	operand

↓

M/R

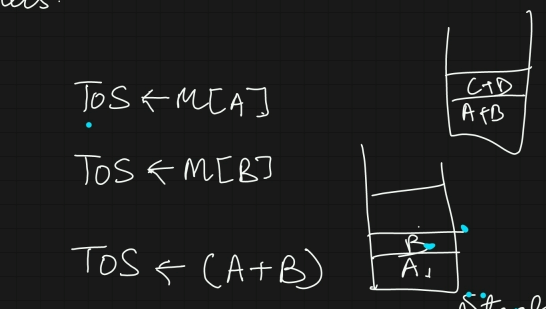
$ACC \leftarrow M[X] * [ACC]$

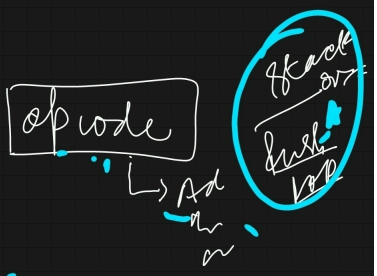
$(A+B)*(C+D)$

0-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)

Add Rep.





Mov (X), X

By using Pop operation

Pop X M[X] ← TOS

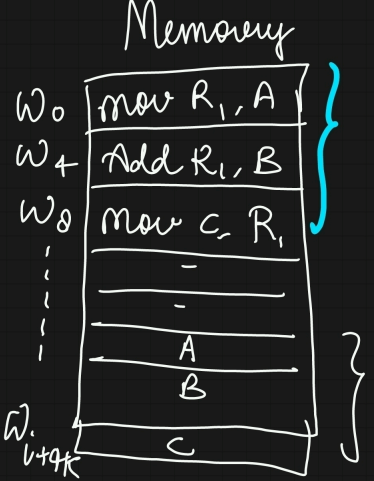
Instruction Sequencing.

Add C, A, B

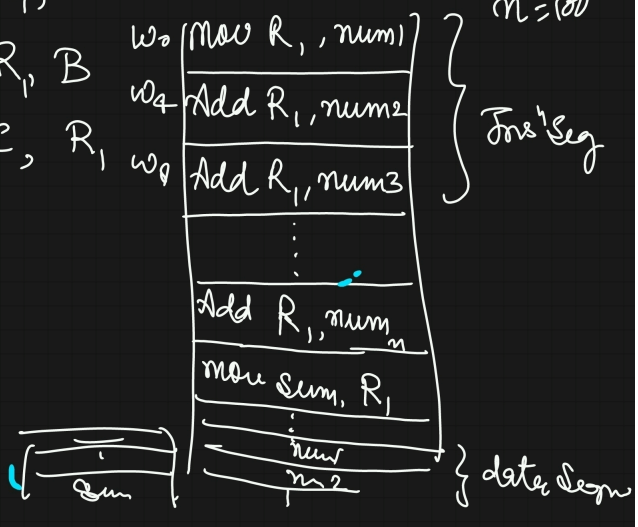
→ mov R_1, A

Add R_1, B

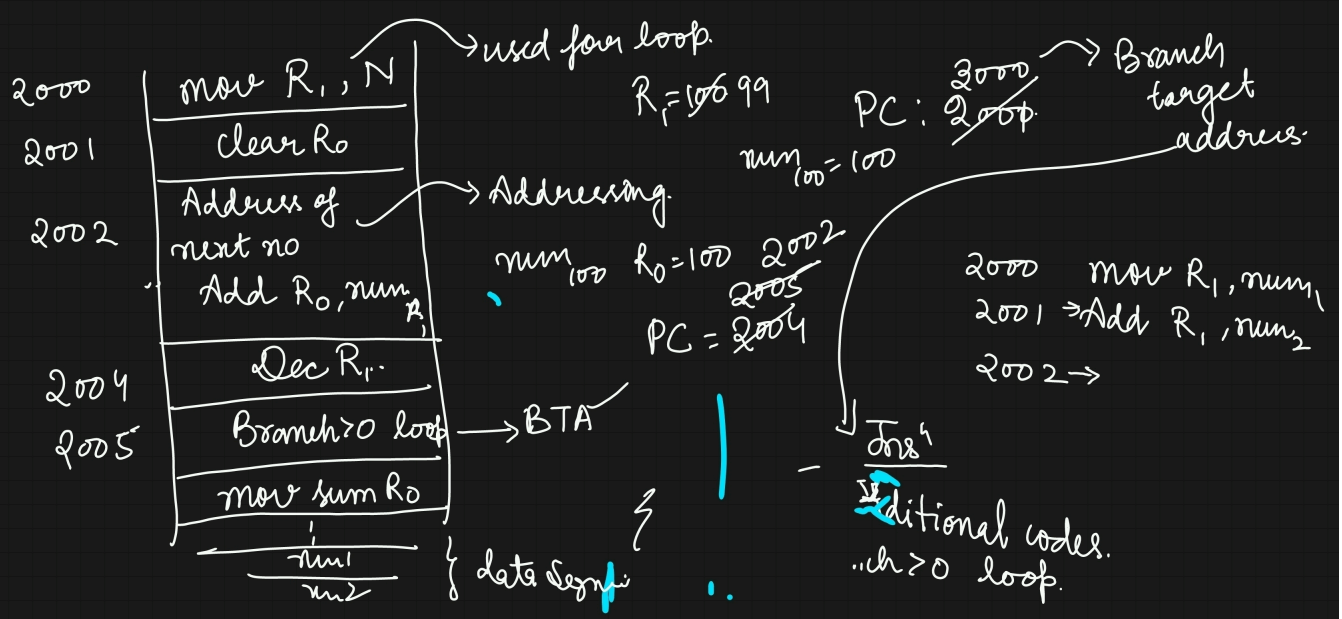
mov C, R_1



what if, there are n numbers for Sum.



Instⁿ Sequencing: Branch Instruction



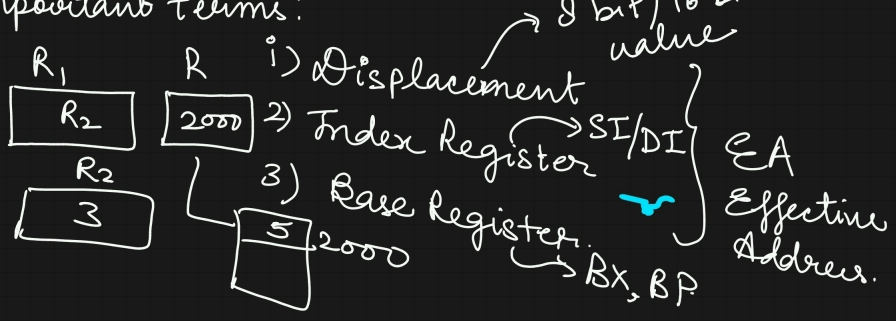
Addressing Modes

→ Representation in which location of an operand is specified with-in Instⁿ:

Important terms:

EA = [R1]

EA = [[R1]]



7) Relative AM

EA = [PC] + Address field of the Ins^{n}.

PC ⇒ 2000

= 2008 + (3)

EA = [2008 + 3]

= [2005]

EA =

→ Program Relocation.

→ Cal. of BTA

8) Index AM

→ Index Register used to cal EA

→ EA = [Address part + [IR]]

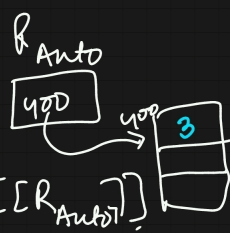
EA = [3002+2]

= [3004]

9) Auto-increment AM

→ used to perform loops.

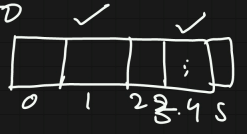
10) Auto-decrement



11) Base Register AM

→ EA = [Address part + [BR]]

→ array relocation



$R_{auto} = [[R_{auto}] + Step\_size$ → depend on the size of operand.

→ if R_{Auto} contains MA, then MR is required

Move	M[500]	mode

AM	EA	content
Direct	M[500]	800
Immediate	-	500
Indirect	[[500]]	300
Relative	[702]	325
Index	[600]	900
Register direct	$[R_1]$	400
" Indirect	[[R1]]	700
Auto Increment	[[R1]]+	701
Auto Decrement	[[R1]]-1	699

200	Load to Ac	Mode
201	Address = 500	
202	next Inst	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

I_1, I_2

EA = [[IR] + Address]

= [100 + 500] =

$[[R_{auto}]]$ + Step-size

= [400] + 1 = 701

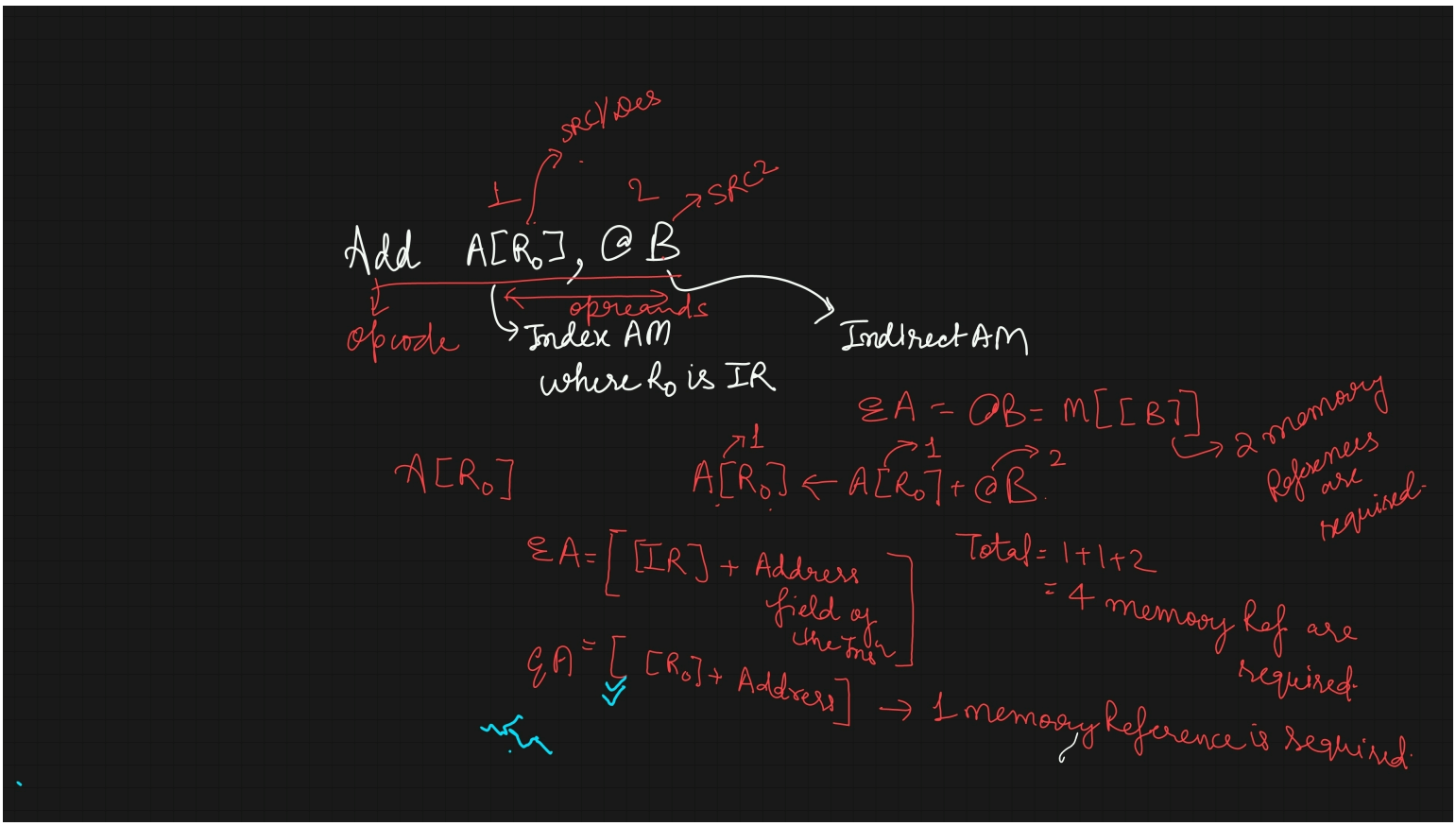
700 - 1 = 699

PC = ~~800~~ 202 R_1 400

R_1 = 400

IR = 100 [[400]]

Step-size = 1 [400]



Q. Memory locations 1000, 1001, and 1020 have data values 18, 1 and 16. consider the following Instructions:

	MovI	R_s, 1	Move immediate
A[R0]	Load	R_d, 1000(R_s)	Load from memory (AM → Index Addressing)
@B, @A	ADDI	R_d, 1000	Add Immediate
	StoreI	0(R_d), 20	Store Immediate

$R_d \leftarrow [R_d] + 1000$ what will be the value in location 1001 after execution?

1000	18
1001	20
1020	16

$R_s \leftarrow 1$

$R_d \leftarrow [1000 + [R_s]]$

$R_d \leftarrow [1000 + 1]$

$R_d \leftarrow [1001]$

$R_d \leftarrow 1$

$R_d \leftarrow 1 + 1000$

$R_d \leftarrow 1001$

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

$[0 + 1001] \leftarrow 20$

20	40
30	50
40	60
50	70

Ins^n	O/P
Load I 20	20
Load D 20	40
Load @20	60
Load I 30	30
Load D 30	50
Load @30	70

RISC

Is.no. | Ins^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

$EA = [PC] + offset$

$EA = 100$

$PC = 100$

$EA = [100 \quad 00]$

$EA = [PC]$

$EA = [PC] + offset$

$100 = 116 + offset$

$offset = 100 - 116$

$= -16$

BTA: i 100

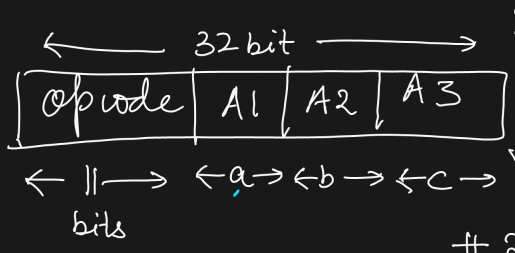
104	i+1
108	i+2
→112	i+3
116	i+4

$PC = 116$

$EA = 116 - 16$

$= 100$

Expanded opcode Technique:



3-address Instruction

of 3 AI = x

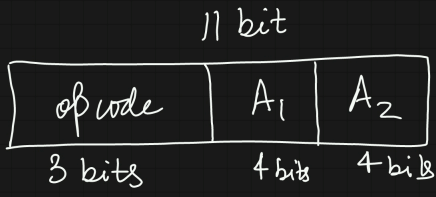
of 2 AI = y

of 1 AI = z

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

1 AI " $= ((2^{11} - x) \cdot 2^a - y) \cdot 2^b$

0 AI " $= (((2^{11} - x) \cdot 2^a - y) \cdot 2^b - z) \cdot 2^c$



1 AI possible from the given Inst^n.

format $= (2^3 - 5) \cdot 2^4$

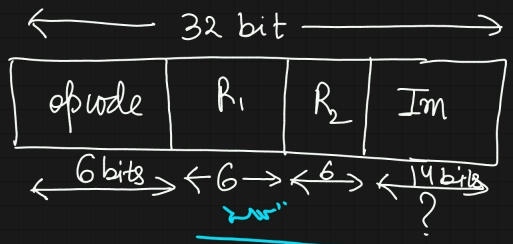
$= 48$

5 – 2 AI

32 → 1 AI

256 ? → 0 AI

gate exam Question Based on this topic



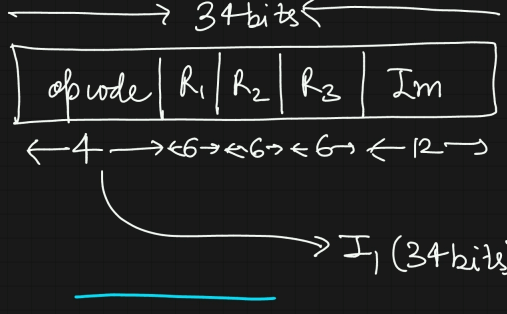
0 to $2^n - 1$

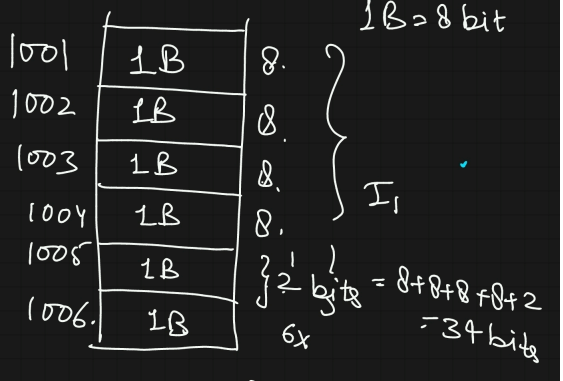
0 000 → +
001 → X
010 → –
⋮
111

$\therefore$ = 32 − (6+6+6)
= 32 − 18
= 14 bits

$\log_2 45 = \log 64$

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





1 Ins^n → 5 Bytes
100 " → 500 Bytes

= 5 Blocks
= 5 Bytes

number Representation

→ Sign and magnitude

→ IEEE floating point representation

→ 2's complement method

it allows large range of number using less no. of bits

1
10
100

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

→ overflow condition.

$n=8$

-2^{8-1} to $+2^{8-1}-1$

-2^7 to $+2^7-1$

↓ −128 to +127

Sign → S < 0 +ve, 1 -ve | Exponent: no. of 0's (power) in the num | Mantissa: actual no in binary

+128

-27-

Standard format

| S | E | M |

5 bits

E: Exponent in bias form.

$bias = 2^{K-1}$

$= 2^{5-1} = 2^4 = 16$

bias = 16 64

Excess 64 code

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

-2^4 to $+2^4-1$ (0-31)

−16 to +15

−16 + 16 = 0

↳ bias.

32-bit unsigned format

	bias
−16	16
−15	15
−14	14
1	
0	0

-28-

Mantissa

101.11

Explicit (By default) → $0.10111 \times 2^{(+3)}$; $e = +3$

$M = 10111$

$E = 3 + bias$

$E = e + bias$

$e = E - bias$

Implicit. $1.0111 \times 2^{+2}$ $e = +2$

$1.X$ → anything.

$M = 0111$

$E = 2 + bias$

$(19.65)_{10}$

0.1

S	E	M

0	3+bias	10111

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

0	2+bias	0111

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

S	E	m

Q $+(19.25)_{10}$

16 bits →

S	E	M
1	6	9

$(10011.01)_2$

$\rightarrow M: 0.100110 \times 2^5$

S	E	M
0	100101	100110100
1	6	9

$e = 5$

$M = 100110100$ (7 bits)

$E = e + bias$

$= 5 + 32$

$= 37$

$E = (100101)_2$

$bias = 2^{K-1}$

$= 2^{6-1}$

$= 2^5 = 32$

0100101100110100

Addition/Sub of floating numbers

1) check for 0's
2) align with Mantissa
3) Perform operation
4) Normalize the Result.

Example — perform addition

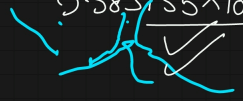
$0.583123 \times 10^3 \quad \times (10^4)$

$(0.123000 \times 10^{-1} \times 10^{+4} \rightarrow 10^3)$

10^3 — less loss of bits — 10^{-1} — positive forever

0.583123×10^3
0.000012×10^3
0.583135×10^3

0.230000×10^{-1}
0.123000×10^{-1}
0.353000×10^{-1}



-31-

1) In Addition, if there is overflow bit.
→ point should be left shift.

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

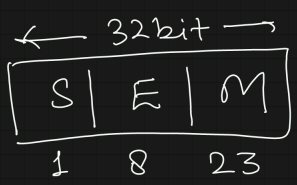
2) In Subtraction, if there is underflow bit.
→ Point should be right shift.

$$\begin{array}{r} 0.534 \times 10^3 \\ - \; 0.521 \times 10^3 \\ \hline 0.013 \times 10^3 \\ \hline 0.130 \times 10^2 \end{array}$$

+0 +∞
-0 −∞

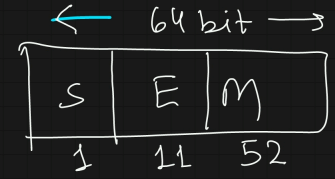
-32-

IEEE-754 Standard



Single precision.

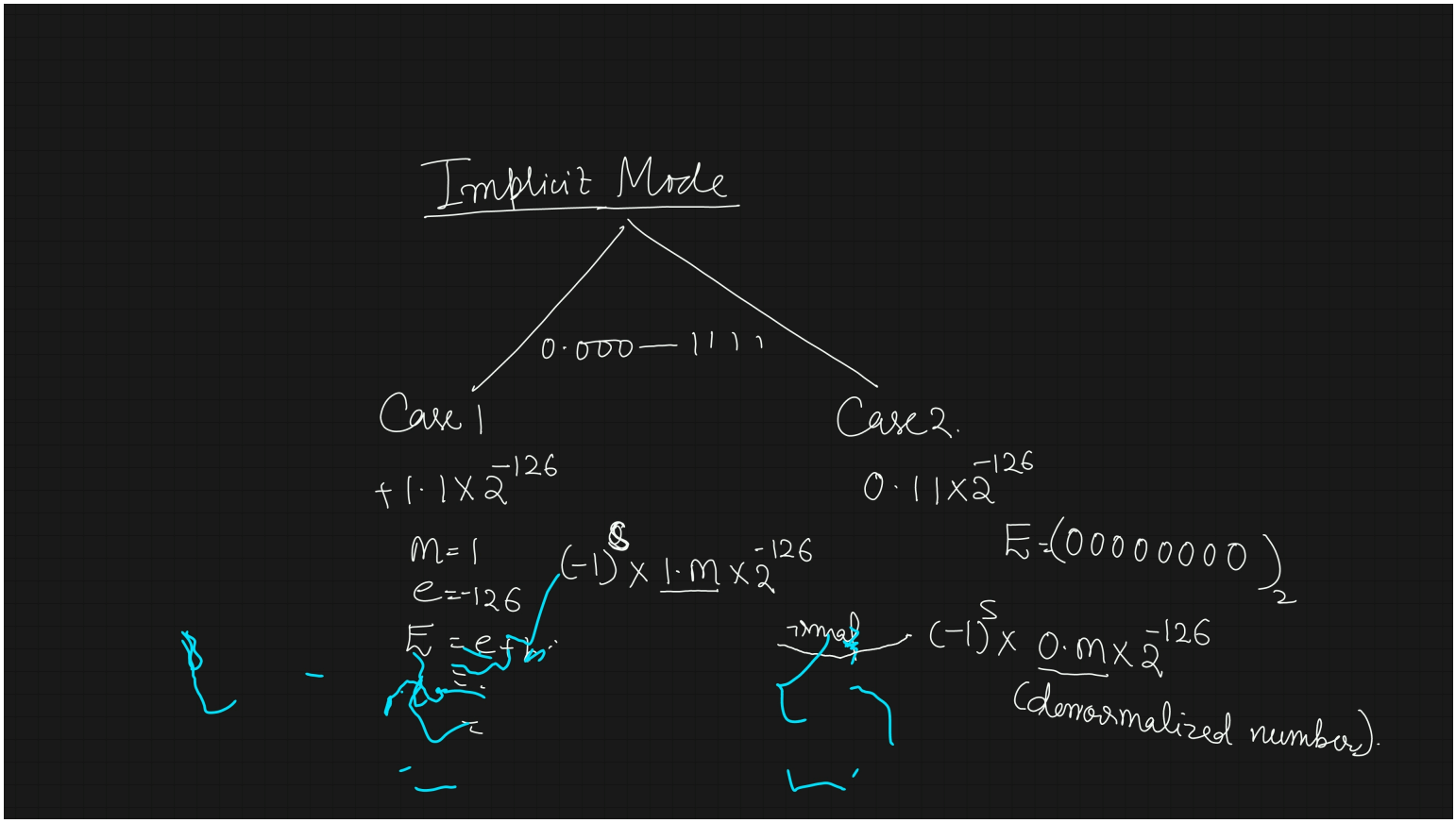
bias = 127

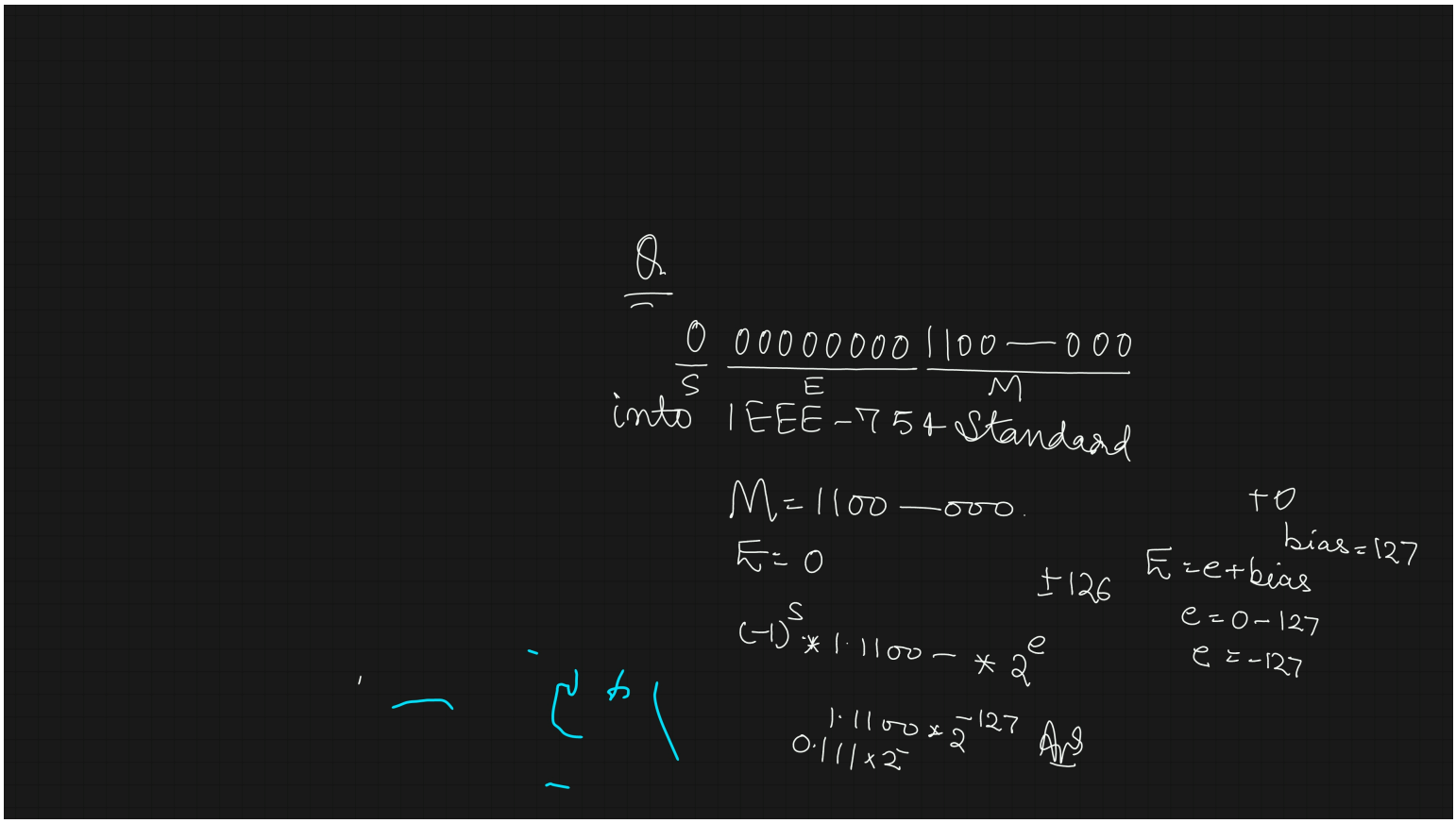


double Precision.

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	$+\infty$
	1	111 — 111	000 — 000	$-\infty$
	0/1	111 — 111	M ≠ 0	not a number
	0/1	000 — 000	M ≠ 0	denormalized numt
	0/1	E ≠ 00 — 00 11 — 11	M = XXX	number (Implicit mode)

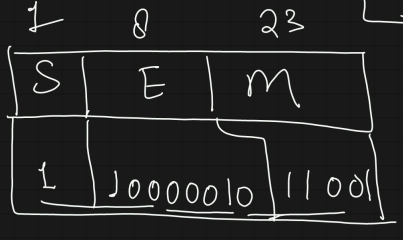

Implicit Mode
0.000—1111
Case 1
$+1.1\times2^{-126}$
$M=1$
$e=-126$
$E=e+b$
$(-1)^S\times1.M\times2^{-126}$
Case 2.
0.11×2^{-126}
$E=(00000000)_2$
$(-1)^S\times0.M\times2^{-126}$
(denormalized number).


Q.
0 | 00000000 | 1100—000
S | E | M
into IEEE-754 Standard
$M=1100—000$.
$E=0$
$(-1)^S*1.1100-*2^e$
1.1100×2^{-127}
$0.111\times2^{-}$
Ans
+0
bias=127
$E=e+bias$
$e=0-127$
$e=-127$
±126

-37-

$(-14.25)_{10}$

$(1110.01)_2$



Implicit mode

1.11001×2^3

$M = 11001$

$e = 3$

$E = e + \text{bias} = 3 + 127 = 130 = (10000010)_2$

1100 0001 0110 0100 000

C 1 6 4 000

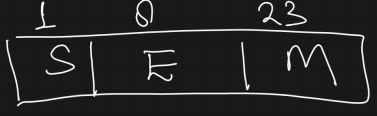
C16400000

-38-

given

0 10000011 1100---00

S | E | M



$1.1100-00 \times 2^{e}$

$1.110000-00 \times 2^{4}$

$(11100)_2 = (+28)_{10}$

$e = E - bias$

$e = 131 - 127$

$e = 4$

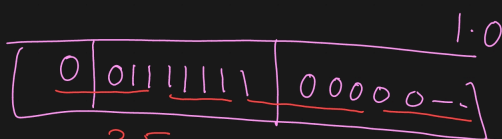
Q

0 10000000 11000---00

$(+3.5)_{10}$ Ans

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

1.0×2^{0}



3F800000

$E = e + bias$

$= 0 + 127$

$= (01111111)_2$