Instruction format

addressing
MOV A, B – register mode
MVI A, 67 – Immediate add. mode
LDA 4000 – Direct add. mode
LDAX D – Indirect add. mode.
Reg.

CLC – Implicit / Implied add mode

→ To address 8 registers 3 bits will be required

A	000
B	001
C	010
D	011
E	100
H	101
L	110

① MOV A, B → 1 byte inst^n
2 3 3 bits

② MVI A, 12 → 2 byte
1 byte 1 byte
(5 for, 3).
opcode reg

- Variable length instruction – size of instructions are varied
fixed length instruction – size is fixed.

1) Three address instruction
ex ADD R1, R2, R3
R1 ← R2 + R3

2) Two address instruction
ex ADD R1, R2
R1 ← R1 + R2

3) Single add. instruction ex APP R1 ∴ AC ← AC + R

→ If we want to perform operation
$X = A \times B + C \times C$
where, A, B, C, X all are memory addresses

– we want to perform it using three address instruction format.

① MUL T, A, B
② MUL Y, C, C
③ ADD X, T, Y

– Using two add. inst^n format,

① LOAD A (load into temp. (memory) variable)
② MUL T, B
③ LOAD C
④ MUL Y, C
⑤ ADD T, Y
⑥ STOR X

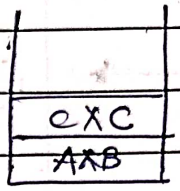
– Using one add. inst^n format, (Bydefault it is in Ac.)

① LOAD A
② MUL B
③ STOR T
④ LOAD C
⑤ MUL C
⑥ ~~STOR Y~~ ADD T
⑦ STOR X

DATE:
PAGE:

- Using zero add. inst^n format. (By default it is in stack)

① PUSH A
② PUSH B
③ MUL — top 2 operands are popped and multiplied & again pushed into stacks
④ PUSH C
⑤ PUSH C
⑥ MUL
⑦ ADD
⑧ POP X.



→ size of the opcode / instruction
three add. > two add. > one add > zero add.

→ size of the program
three add < two add. < one add. < zero add.

→ variable length opcode — more complex decoder is required.
— size of the code decreases

→ | opcode | operand |
n — 2^n
k — 2^k

If we want to increase the no of instructions

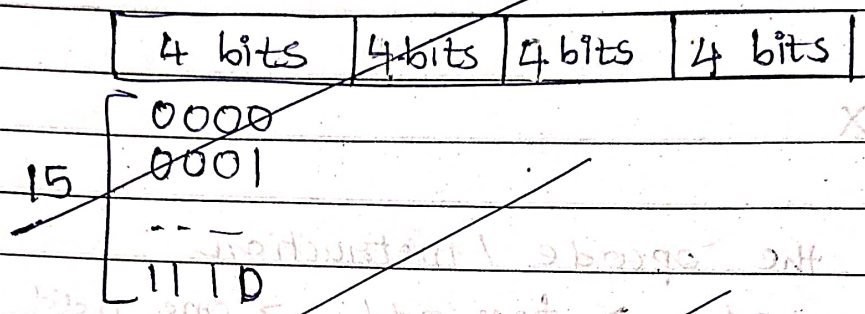
| opcode | operand |
2^{n+1} 2^{k-1}

— increase opcode size
decrease operand size.

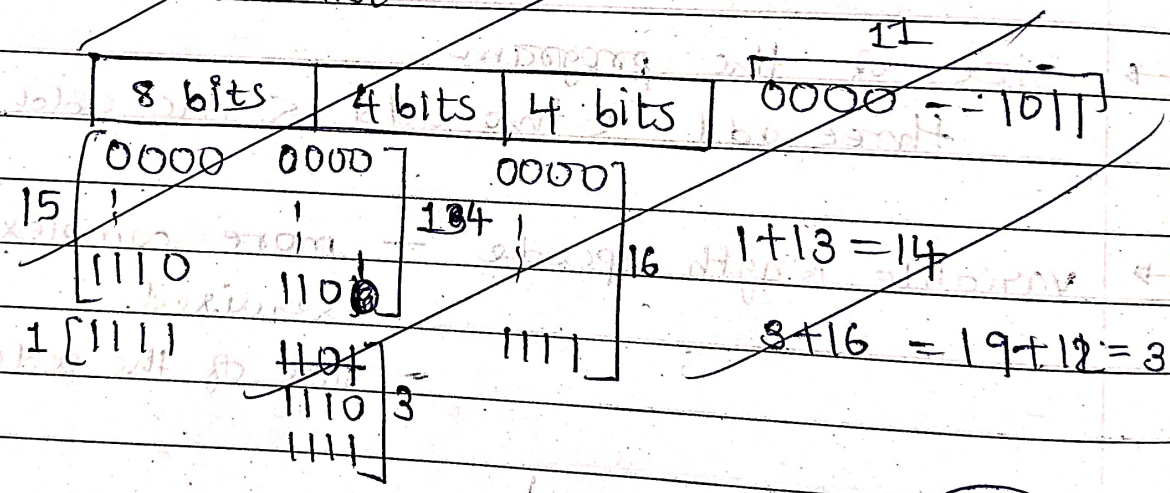
ex. MOV C,B
ADD B ← no need to specify the acc. add.
So, size of the operand can be decreased

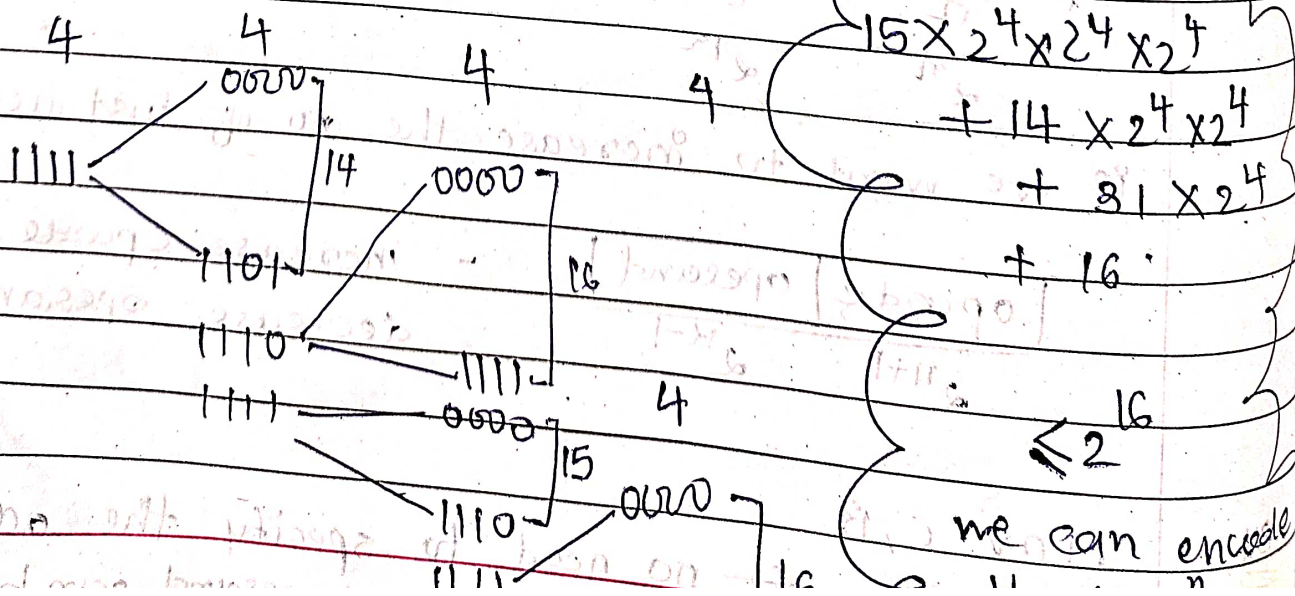
Ex. Consider a machine in which instructions are 16 bit long and add. are 4 bits long The requirement is ~~in~~ 15 3 add. instructions, 14 2 add. inst^{n}s, 31 single add. inst^{n}s & 16 zero add. inst^{n}s.

→ (1) 15 3 add. inst^{n}s.



(2) 14 2 add inst^{n}s





integer

Ex. A processor has 16 registers & 64 floating point registers. It uses 2 byte instruction format. There are 4 categories of inst^{n}s.

type	instruction	integer	floating
1	4	3	0
2	8	0	2
3	14	1	1
4	N	0	1

int = 4
float = 6

Find N.

→ $4 \times 2^4 \times 2^4 \times 2^4 + 8 \times 2^6 \times 2^6 + 14 \times 2^4 \times 2^6 + N \times 2^6 = 2^{16}$

$\Rightarrow 4 \times 2^{12} + 8 \times 2^{12} + 14 \times 2^{10} + N \times 2^6 = 2^{16}$

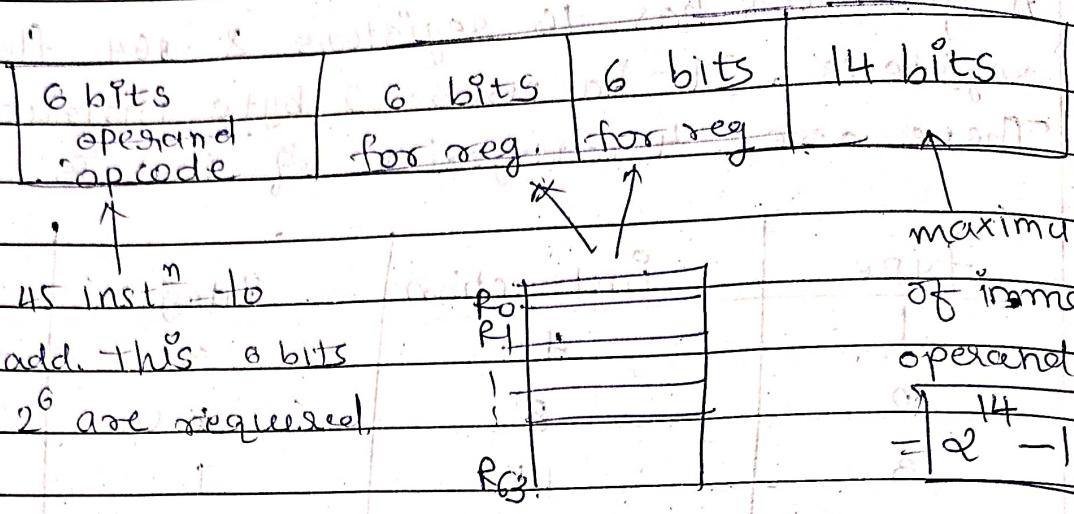
$\Rightarrow 4 \times 2^6 + 8 \times 2^6 + 14 \times 2^4 + N = 2^{10}$

$\Rightarrow 12 \times 2^6 + 14 \times 2^4 + N = 2^{10}$

$\Rightarrow \boxed{N = 32}$

Ex A machine has a 39 bit architecture with one word long inst^n. It has 64 registers each of which is 32 bit long. It needs to support 45 inst^n which has an immediate operand in addition to 2 reg operands. Assuming that immediate operand is an unsigned integer. Find the maximum value of immediate operand.

DATE:
PAGE:



→ add. of the operand = ~~addressing~~ effective address (E

Ex = For the instⁿ shown what values loaded in the accumulator for each addressing mode.

R1 = 800

800	900
900	500
1000	500
1100	600
1600	700

(i) Immediate

LOAD 800 (data)

effective address = 800

(ii) Direct

		EA	AC
LOAD 800 (data)	Immediate	601	800
LOAD 800 (add.)	Direct	800	900
LOAD 800	indirect	900	1000
LOAD 800	index	1600	

→ PC relative addressing mode

Data Path Design

M m_3 m_2 m_1 m_0 → Multiplicand

Q × q_3 q_2 q_1 q_0 → Multiplier.

m_3q_0 m_2q_0 m_1q_0 m_0q_0 → 0 0 0 0

m_3q_1 m_2q_1 m_1q_1 m_0q_1 → 1 0 1 1

0 0 0 0 →

1 0 1 1 →

1 1 0 1 1 1 0

Partial products. To store this 4 registers are required.

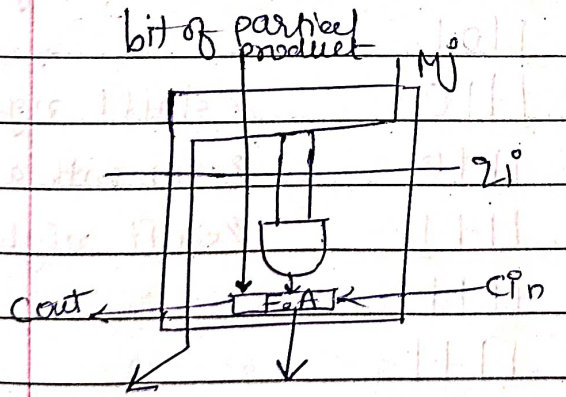
R = 0 0 0 0 0 0 0 0 0 using single register.

\+ 0 0 0 0

R = 0 0 0 0 0 0 0 0 0

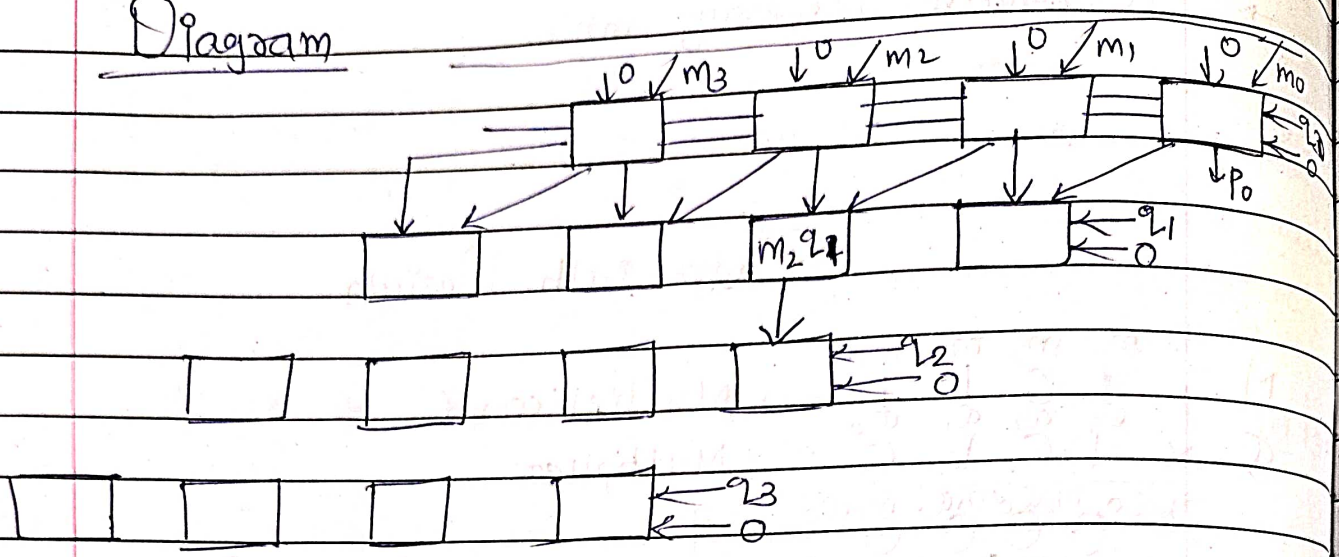
\+ 1 0 1 1

R = 0 0 0 1 0 1 1 0



DATE:
PAGE:

Diagram



→ 4 bit multiplication requires 16 (4^2) Blocks, more Complex more 16 bit multiplication. but speed is high.

Multiplicand = 1101
Multiplier = 1011

	C	Ac	Multiplier	
	0	0000	1011	→ if 1 added with ac.
I st step	0	1101	1011	
	0	0110	1101	// shift right
II nd step	1	0011	1101	
	0	1001	1110	// shift right
III rd step	0	1001	1110	// sum with ac. (0)
	0	0100	1111	// shift right
IV th step	1	0011	1111	
	0	1001	1111	

4 steps are required.

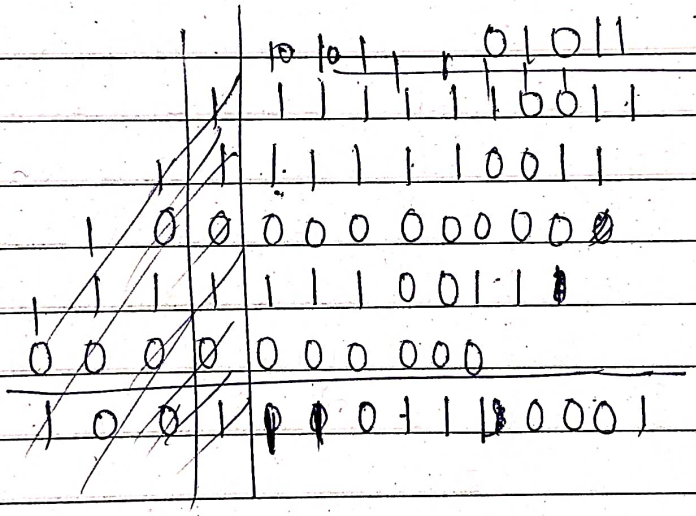
Ex. Multiplicand = 0111
Multiplier = 1000

C	AC	Multiplier
0	0000	1000
0	0000	0100
0	0000	0010
0	0000	0001
0	0111 + 1001	0001
0	0011	1000 → (56)10.

1000
0001
1001

(−13) × (+11).

10011
01011



00] Arithmetic shift right
11]
10 AC = AC − Mnd.]
01 AC = AC + Mnd]

↑ plus Arithmetic shift right

AC	MQR	Q	Mnd
00000	01011	0	10011
01101	01011	0	sub] → I
00110	10101	1	Asr.]
00011	01010	1	Asr.] → II
10110	01010	1	add.] → III
11011	00101	0	Asr.]
01000	00101	0	sub.] → IV
00100	00010	1	Asr.]

copy

AC	Mlr	Q.	Mnd.
10111	00010	1	sub.] — IV
11011	10001	0	Asr.]

Ans