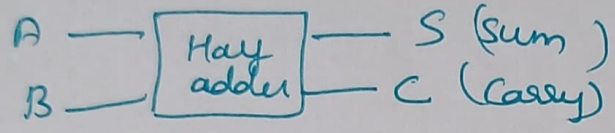
KRISHNASWAMY EDUCATIONAL TRUST, Chennai.
Class..... Sec..... Reg. No. .....

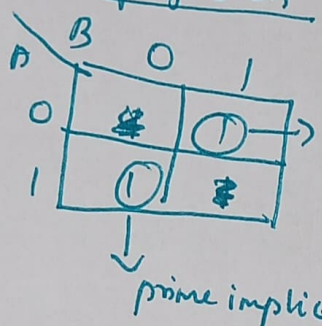
Designing of half adder

* Half adder is a combinational logic circuit designed to add two single bit numbers
* It contains 2 inputs & two outputs (sum & carry)



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Kmap of sum

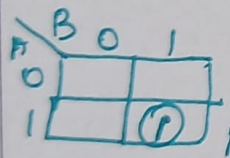


prime implicant $= \bar{A}B$

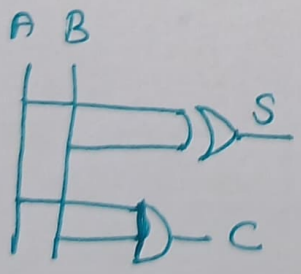
prime implicant $= A\bar{B}$

$A \oplus B$

K-map of carry



prime implicant $= AB$



Designing of Full Adder

Truth Table.

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A 00
B 01
C 01
0

To design a logic circuit we need a boolean expression
To design any boolean expression, we use a K Map.

3 variable K Map as there are 3 input variables

no. of cells for a 3 variable K map is 8.

A \ BCi	00	01	11	10
0		1		1
1	1		1	

K map for sum

need to place values in K map where sum is 1 in the truth table.

if Total diagonal is covered, then XOR of variables is taken

$A \oplus B \oplus C_{in}$

K Map

3 variable for Cout

A \ BCin	00	01	11	10
0			1	1
1		1	1	1

no quarter possible

if doubles are possible → yes

3 doubles are possible

1 vertical grouping

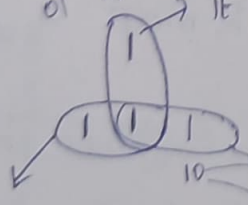
& 2 horizontal grouping

we got 3 prime implicants

by adding these 3 prime implicants, we get the boolean exp for Cout

→ [if 11 → BCin, if 01 → only Cin, if 10 → only B]

01 11 → it is covering only top variables, so BCin



10 → only B in topwise & A in sidewise.

sidewise & topwise

~~BCin~~

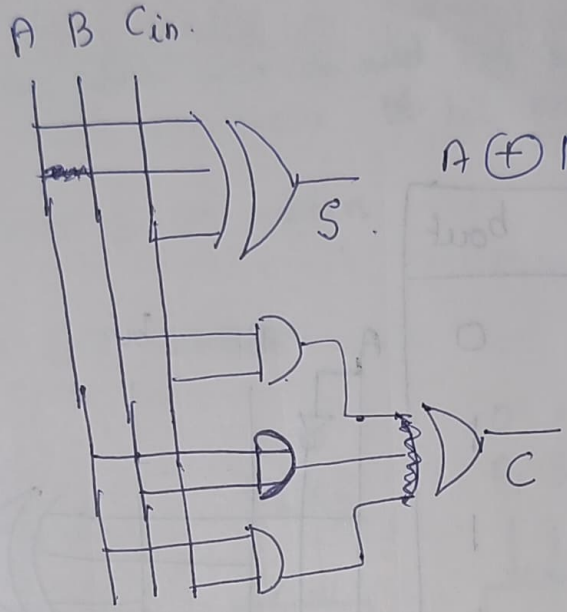
ACin.

10

BCin.

↑↑ ~~one~~ ↑ in B only so B is selected

So BCin + AB + ACin

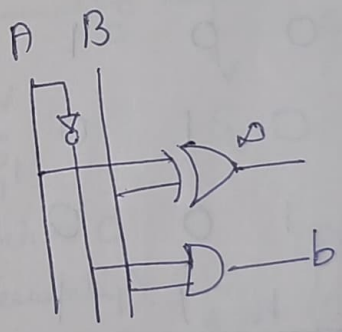


$A \oplus B \oplus C_{in}$

$BC_{in} + AB + AC_{in}$.

Half subtractor

A	B	D	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



A \ B	0	1
0		1
1	1	

$D = A \oplus B$

K-map of D.

A \ B	0	1
0		1
1		

one ~~only~~ & only prime implicant

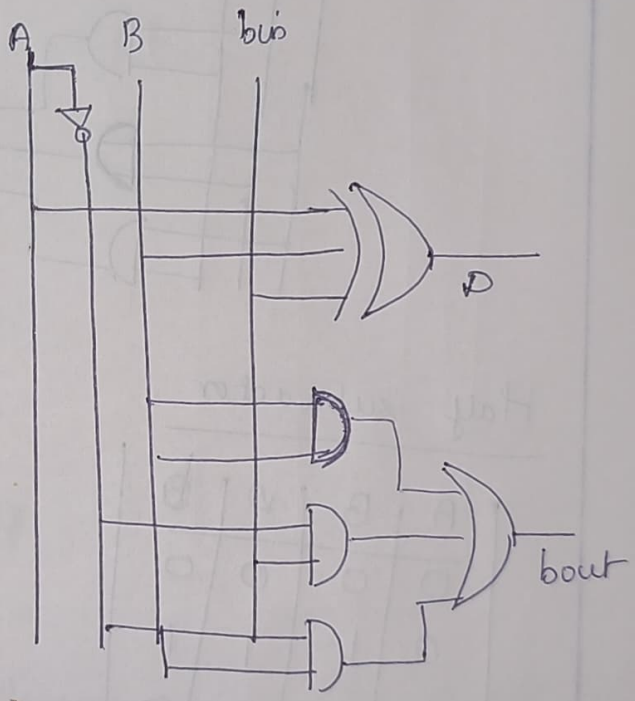
K-map of b. $= \bar{A}B$

b = Sidewise it is 0 which means complement of A which is $\bar{A}$

Full subtractor

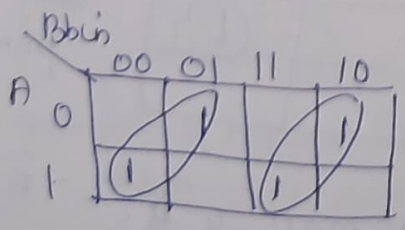
Truth table

A	B	b_{in}	D	b_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



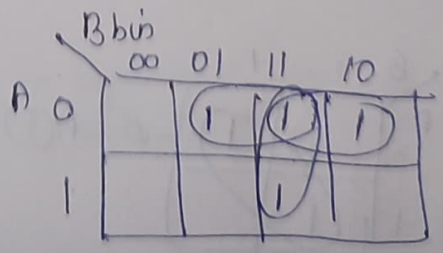
K Map for D $\quad 2^3 = 8$ cells.

3 input variables



$D = A \oplus B \oplus b_{in}$

K Map for bout



$Bb_{in} + \bar{A}b_{in} + \bar{A}B$

only doubles are possible

3 doubles

LSB - Least Significant Bit Adder

Ripple Carry Adder or parallel adder

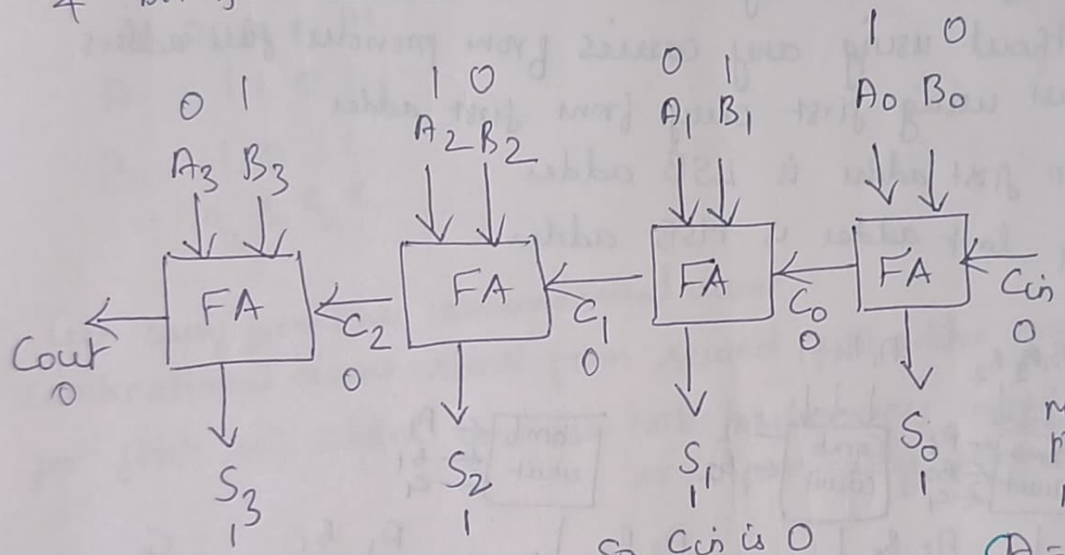
It is a binary adder used to add two N-bit binary numbers

To add multiple bit numbers is called N-bit parallel adder

Cascade of full adders

Consider

4 bit full adders



Initially there is no carry. So C_{in} is 0 (unless it is mentioned deliberately).

Most primary bits MSB — Least primary bits LSB.

$A_3 A_2 A_1 A_0$

A = 0 1 0 1

B = 1 0 1 0

$S_3 S_2 S_1 S_0$

1 1 1 1

2 4 bit binary number

For a full adder of 3 inputs A, B and C_{in}, XOR of three bits is the odd number of XOR

[XOR gate is odd one detector]

1, 0, 0

A_0 B_0 C_0

$\therefore S_0 = 1$

add one.

$$Sum = A \oplus B \oplus C$$

$$Carry = AB + BC + CA$$

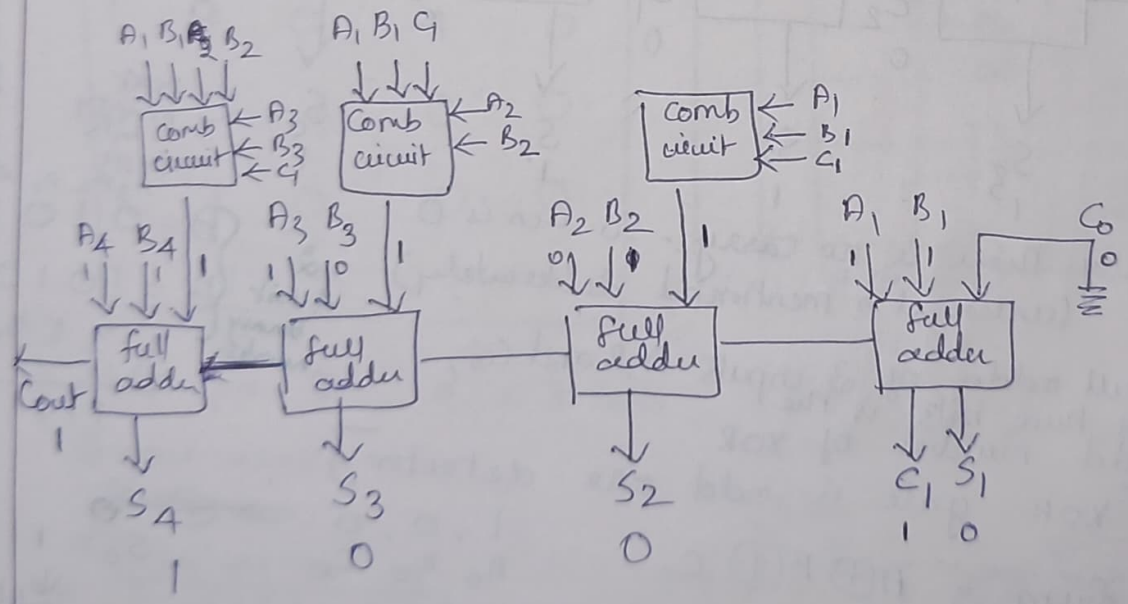
$AB+BC+CA$ = Carry

$1\times0 + 0\times0 + 0\times1 = 0$

Result of addition A B

Carry look ahead Adder

designed from 4 full adders
4 bit addition
→ using N-bit parallel adder, the delay is very high. To reduce this delay we go for carry look ahead adder
without using any carries from previous full adders
just using first carry from first adder
so first adder is LSB adder
& last adder is MSB adder



$0 \oplus 1$ is always $\underline{1}$.

$S = A_i \oplus B_i \oplus C_{in}$

$C_{i+1} = P_i C_i + G_i$

where $P_i = A_i \oplus B_i$

$G_i = A_i B_i$

$P_i \Rightarrow$ Carry propagate

$G_i \Rightarrow$ Carry generator

	A_4	A_3	A_2	A_1
A	1	1	0	1
B	1	0	1	1
	B_4	B_3	B_2	B_1

Get carry from the combinational circuit

Combinational circuit starts from second full adder onwards.

for first full adder we are not having any input or input carry flag is zero

S = XOR values of inputs

$S1$ = 1 XOR 1 XOR 0 (odd one is 0) So $S1 = 0$.

$C_1 = AB + BC + CA$

$= 1 + 0 + 0$ so $C_1 = 1$

$C_2 = P_1 C_1 + G_1$

$P_1 = A_1 \oplus B_1 \Rightarrow 1 \oplus 1 = 0$

$G_1 = A_1 B_1 \Rightarrow 1 \cdot 1 = 1$

$C_2 = 0 \cdot 1 + 1 = \underline{1} \Rightarrow$ combinational circuit

$= 0 + 1$

$S_2 = 0$ $(0 \oplus 1 \oplus 1)$

$C_3 = P_2 C_2 + G_2 = P_2 (P_1 C_1 + G_1) + G_2 = P_1 P_2 C_1 + P_2 G_1 + G_2$

$= 0 + 1 \cdot 1 + 0$

$= 0 + 1 + 0$

$= \underline{1}$.

~~$P_2 = A_2 \oplus B_2 \Rightarrow 0 \oplus 1 =$~~

~~$P_2 =$~~

so $C_3 = 1$

$$C_4 = P_3 C_3 + G_3$$
$$= P_3 (P_1 P_2 C_1 + P_2 G_1 + G_2) + G_3$$
$$= P_1 P_2 C_1 P_3 + G_1 P_2 P_3 + G_2 P_3 + G_3$$
$$= 0 \quad + 1 \cdot 1 \cdot 1 \quad +$$

$$C_4 = 1$$

$$S_4 = A_4 \oplus B_4 \oplus C_4$$
$$= 1 \quad 1 \quad 1$$

$$S_4 = 1$$

$$\text{Carry} = AB + BC + CA$$
$$= 1$$

In all the cases we are not depending on the previous carry we just depend upon the first carry which is generated by first full adder

<u>Cross checking</u>

$$\begin{array}{lcccc} A & 1 & 1 & 0 & 1 \\ B & 1 & 0 & 1 & 1 \\ & 1 & 1 & 1 & \\ \hline 1 & 1 & 0 & 0 & 0 \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ & S_4 & S_3 & S_2 & S_1 \end{array}$$

Another bit $\rightarrow C_{out}$

Binary multiplication

Rules
$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

```
  1011   (11)
   111   (7)
  ----
  1011
 1011 .
1011
------
1001101
```

	sum	carry
1+1	0	1
1+1+1	1	1
1+1+1+1	0	1+1 → 1

1+1 = 1 with carry 1

3

decimal

1000110.1011
64 32 16 8 4 2 1 -1 -2 -3 -4

$64+4+2 = 70$

$1\times2^{-1}+1\times2^{-3}+1\times2^{-4}$

$70+\frac{1}{2}+\frac{1}{8}+\frac{1}{16}$

$70+0.5+0.125+0.0625$
$=70.6875$

1001101
64 32 16 8 4 2 1

$64+8+4+1 = 77$

Shift and add multiplication

$X \times Y \longrightarrow$ multiplier
$X \downarrow$ multiplicand

M = multiplicand ⇒ 1101

C = 0

13 × 14 = 182
~~1011010~~
~~128 64 32 16 8 4 2 1~~
10110
64 32 16 8 4 2 1

```
  1101   13
  1110   14
  ----
  0000
 1101x
1101xx
1101xxx
--------
10110110
```

Right shift CAB
C A B

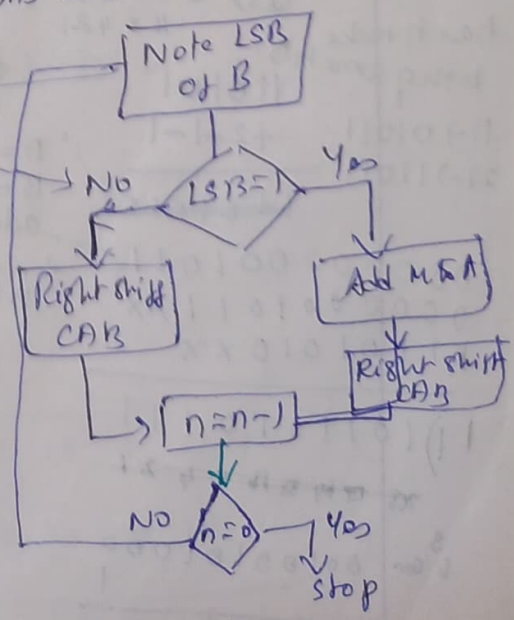
A = accumulator
= 0000 – initialize as it is four bits.

B – multiplier.

n = 4
↳ no. of bits in B

C	A	B	
0	0000	1110	discard, n = 3
0	0000	0111	
0	1101	0111	discard, n = 2
	0110	1011	
1	0011	1011	discard, n = 1
0	1001	1101	
1	0110	1101	discard, n = 0
0	1011	0110	

matches the addition value.



Add M to A

```
 1101
 0000
 ----
 1101

 1101 M
 0110 A
 ------
 10011
```

```
 1101
 1001
 ----
 0110
```

This technique guarantees that the maximum number of summands (versions of the multiplicand) that must be added is $n/2$ for n bit operands

Bit Pair Recoding of multipliers

comes under Fast multiplication

Booths algorithm is best for handling signed & unsigned numbers
Same is the case with bit pairs & carry save
but this handles both positive & negative numbers equally

in booth & bit pair we do recoding the multiplier
but in booth for recoding we will examine 2 bits (for recoding)
in bit pair " 3 bits & that's y it is fast & the steps is less

3 bits $i+1, i, i-1$

$i+1$	i	$i-1$	Bit pair recoded
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	0

$2\overline{|27}$
$2\overline{|13}$ −1
$2\overline{|6}$ −1
$2\overline{|3}$ −0
1 −1

Eg in booth we examine right to left for recoding but in bit pair left to right

0 1 0 0 1 0 $\boxed{0}$ implied zero

+1 +1 −2

write straight to i bit

11 (4 bits)
1011

27 eg (−11) 1 1 0 1 0 1 → Multiplicand → 1's 001010 → 001011 (8 2 1) = 11

11011 (5 bits) (27) 0 1 1 0 1 1 (16 8 4 2 1) → Multiplier → Recode

0 1 1 0 1 1 $\boxed{0}$
+2 −1 −1

have to make both of same bits

11 → 01011
27 → 11011

110101
+2 −1 −1

for −11, take 2's compliment of 11
01011 → 10101

∴ A = 10101
B = 11011
add signed bit → if neg add 1 if pos 0
∴ A = 110101
B = 011011
27 × −11 = −297

−1 → 2's compliment of multiplicant
+2 → Multiplicant × 2 or (10)
−2 → 2's compliment of multiplicant × 2 or × (10)

~~1 1 0 1 0 1~~

000000001011
00000001011 xx
11101010 xx

11 1 0 1 1 0 1 0 1 1 1
~~256 128 64 32 16 8 4 2 1~~

2's com 0 0 1 0 0 1 0 1 0 0 0
1

0 0 1 0 0 1 0 1 0 0 1
256 128 64 32 16 8 4 2 1 = 256 + 32 + 8 + 1
= 297

110101
1's 001010
2's 001011 (−1)

+2 110101 × 10
= 1101010

7 Reduces the time needed to add the summands

KRISHNASWAMY EDUCATIONAL TRUST, Chennai.

Class..... Sec..... Reg. No. .....

CARRY SAVE ADDITION of summands M Q 45 x 63 = 2835

```
  1 0 1 1 0 1
  1 1 1 1 1 1
-------------
  1 0 1 1 0 1      A
1 0 1 1 0 1        B
1 0 1 1 0 1        C
-------------
1 1 0 0 0 0 1 1    S1
0 0 1 1 1 1 0 0    C1
-------------
  1 0 1 1 0 1      D
1 0 1 1 0 1        E
1 0 1 1 0 1        F
-------------
1 1 0 0 0 0 1 1    S2
0 0 1 1 1 1 0 0    C2
```

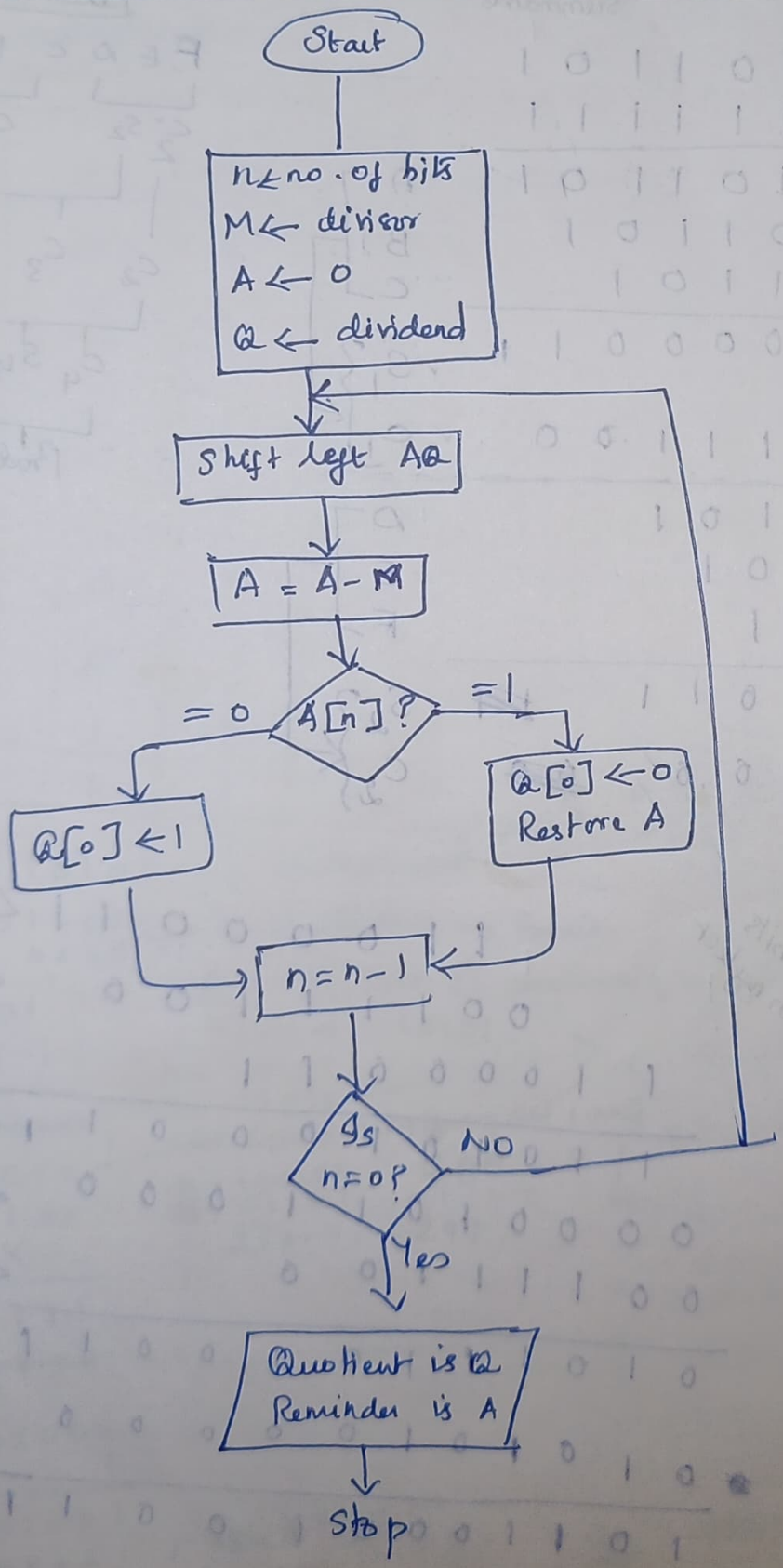
Roadmap

```
F E D C B A
 \ /   \ /   Level 1 CSA
C2 S2 C1 S1
 |   \ | /   Level 2 CSA
 C2  C3 S3
 \ /  \ /    Level 3 CSA
 C4   S4
   \ /       Final addition
   Product.
```

8 + 6 bits = 12 bits
after 12 bits
we can neglect

```
1 1 0 0 0 0 1 1   ← S1
0 0 1 1 1 1 0 0   ← C1
1 1 0 0 0 0 1 1   ← S2
-----------------
1 1 0 1 0 1 0 0 0 1 1   ← S3
0 0 0 0 1 0 1 1 0 0 0   ← C3
0 0 1 1 1 1 0 0         ← C2
-----------------
0 1 0 1 1 1 0 1 0 0 1 1   S4
0 1 0 1 0 1 0 0 0 0 0     C4
-----------------
1 0 1 1 0 0 0 1 0 0 1 1
```

Restoring division algorithm for unsigned integer



KRISHNASWAMY EDUCATIONAL TRUST, Chennai.

Class..... Sec..... Reg. No. .....

2 | 11
2 | 5 – 1
2 | 2 – 1
1 – 0

2 | 3
1 – 1

Tracing table

11 / 3 ⇒ 3 Quotient &
dividend divisor 2 Remainder

11 ⇒ 1011
3 ⇒ 00011

n	M	A	Q	Action/operation
4	00011	00000	1011	initialization
	register = (n+1) bits	is lost		
		00001	011?	SLAQ
		11110	011?	A = A – M
3		00001	0110	Q[0] ← 0 Restore A
		00010	110?	SLAQ
		11111	110?	A = A – M
2		00010	1100	Q[0] ← 0 Restore A
		00101	100?	SL AQ
		00010	100?	A = A – M
		00010	1001	[0] ← 1
1		00101	001?	SL AQ
		00010	001?	A = A – M
		00010	0011	Q[0] ← 1
		2	3	
		2		

A[n] means MSB of the register

00111
011) 10110
011
101
011
100
011
001 → R

Shift left AQ

MSB in A will be lost
will be vacant

LSB in Q ~~is lost~~

– M

2's complement of M
00011
11100 1's compli
1 +1
11101

00001
11101
11110

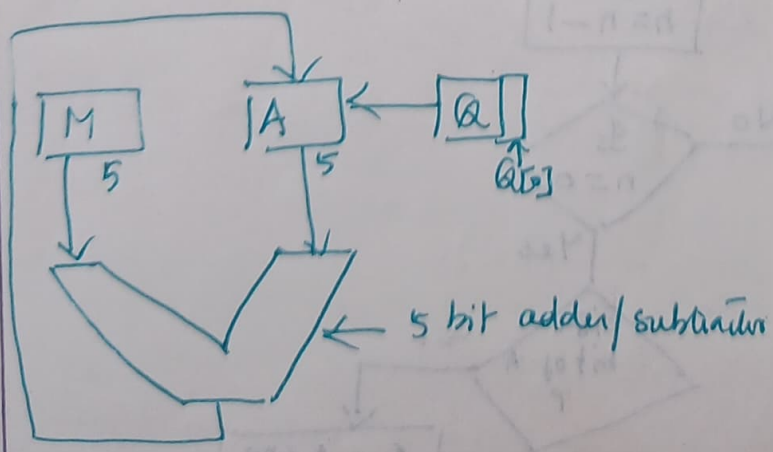
00010
11101
11111

00101
11101
100010

6th bit is discarded.

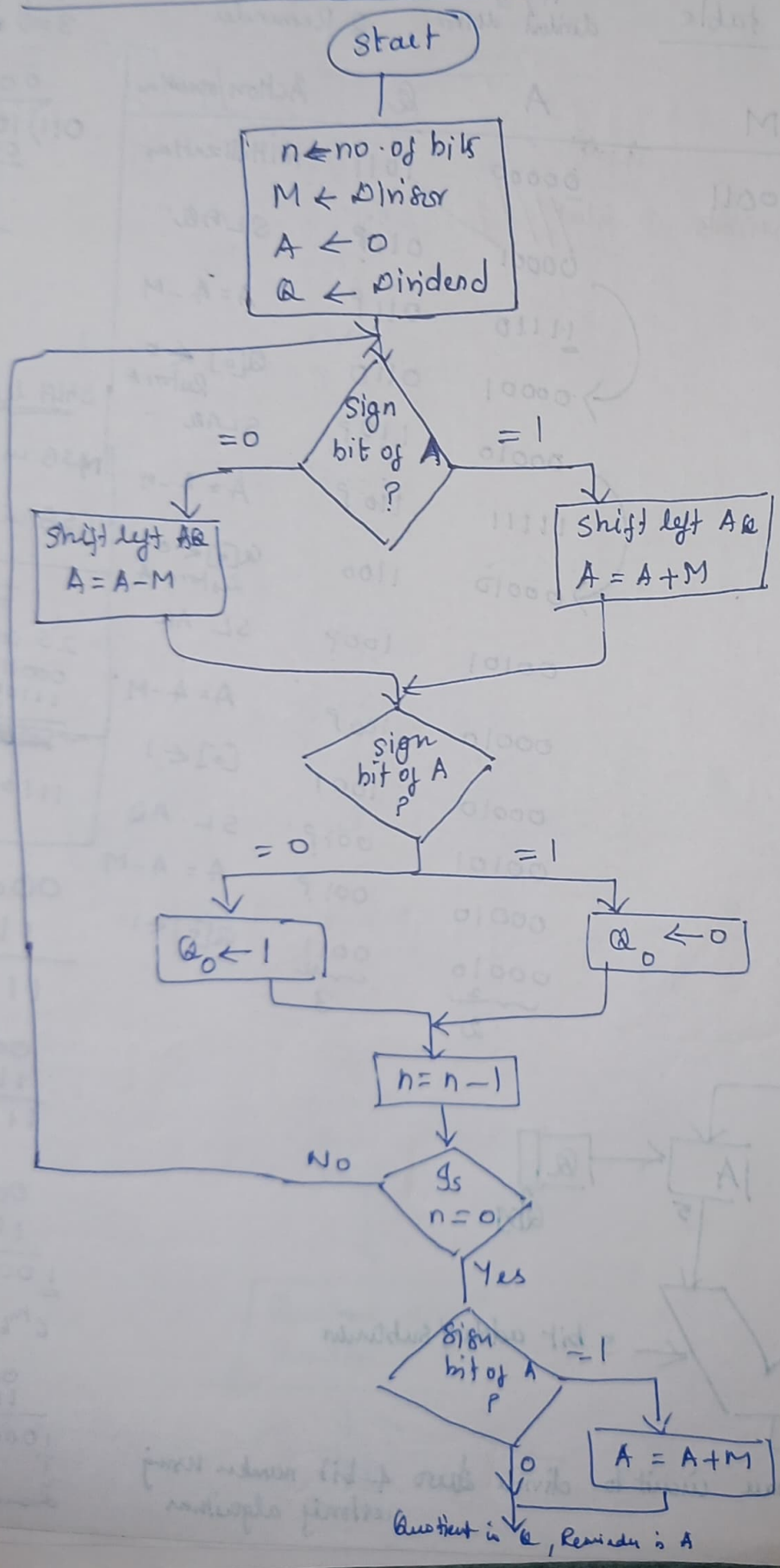
00101
11101
100010

discarded



Hardware circuit to divide two 4-bit numbers using restoring algorithm

Non restoring division for unsigned integer.



KRISHNASWAMY EDUCATIONAL TRUST, Chennai.

Class..... Sec..... Reg. No. .....

11 dividend / 3 divisor = 3 (Quotient) 2 (Remainder)

n	M	A	Q	Action/operation
4	00011 (n+1) bits	00000	1011	Initialization
		is lost 00001	011?	SL AQ
		11110	011?	A = A − M
		11110	0110	$Q_0 \leftarrow 0$
		11100	110?	SL AQ
3		11111	110?	A = A + M
		11111	1100	$Q_0 \leftarrow 0$
2		11111	100?	SL AQ
		00010	100?	A = A + M
		00010	1001	$Q_0 \leftarrow 1$
1		00101	001?	SL AQ
		00010	001?	A = A − M
		00010	0011	$Q_0 \leftarrow 1$
		Remainder	Quotient	

2 | 11
2 | 5 − 1
2 | 2 − 1
1 − 0

00011
11100
1
−M 11101

00001
11101
11110

00101
11101
100010

IEEE 754 Floating point (Education 4u)

It has three parts sign & Mantissa, ~~Base~~ biased, Exponent, normalized mantissa binary pt floats to the right of most sig 1 and an exponent is used

floating pt representation $\pm M \times B^{E}$

Single precision format: 32 bit

31	30 … 23	0
1 bit	8 bits	23 bits
Sign	exponent	Mantissa

$E' = E + 127$

Double precision format: 64 bit

64	63 … 51	0
1	11	52
S	E'	M

$E' = E + 1023$

eg: 1259.125

convert decimal to binary

Step (i) ~~Binary~~

$(1259)_{10} \Rightarrow (10011100011)_2$

$(0.125)_{10}$

$0.125 \times 2 \Rightarrow 0.250 \Rightarrow 0$ (integer part)

$0.25 \times 2 \Rightarrow 0.5 \Rightarrow 0$

$0.5 \times 2 \Rightarrow 1.0 \Rightarrow 1$

$0.0 \times 2 \Rightarrow 0 \Rightarrow 0$

2	1259
2	629 – 1
2	312 – 1
2	156 – 0
2	78 – 0
2	39 – 0
2	19 – 1
2	9 – 1
2	4 – 1
2	2 – 0
	1 – 0

0	10010011	00111000011001…0

$(10011100011.001)_2$ (decimal is converted into binary)

Step 2: Normalize the number

for single precision formula is $(1.N)2^{E-127}$

for double precision $(1.N)2^{E-1023}$

$(1.00111000011.001)_2$

~~1.00111000011.001~~

1.00111000011001

2	137
2	68 – 1
2	34 – 0
2	17 – 0
2	8 – 1
2	4 – 0
2	2 – 0
	1 – 0

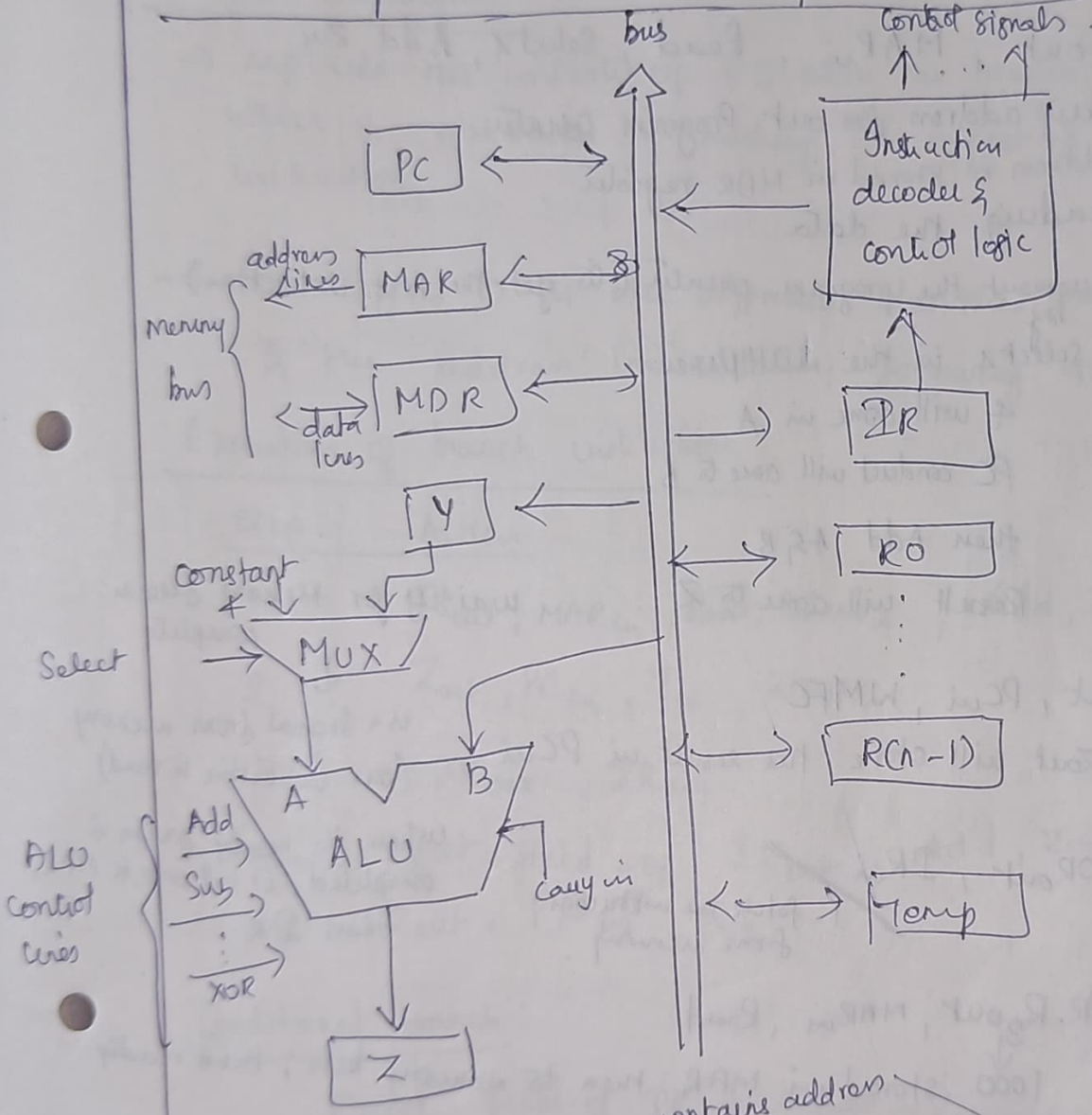
Step 3 : single precision format

$[1.N]2^{E-127} = \boxed{1.00111000011001} \times 2^{10} \Rightarrow E - 127 = 10$

$E = 137 = 10001001$

N Mantissa includes the sign bit. 0 is a signed bit

Control sequence to execute a complete instruction



Program Counter
Stores the address of the next instruction to be executed

MAR – contains address register

MDR – Register

IR – Instruction register

$R_0 \ldots R_{n-1}$ – Registers

Temp – Temporary register

Y register

MUX – Multiplexer one input constant & another input is o/p of Y register

o/p of ALU is stored in Z register

Address	Content
1016	Add (R3), R1
1000	20

Add (R3) R1 — R_3 – 1000 (contains address) — R_1 – 30

(parenthesis indicates pointer)

1. Fetch the instruction from memory – IR (current instruction are stored in IR register)
2. Fetch the first operand from memory.
   20 should add with 30
3. perform addition operation
4. Load the result into R1

How to write control sequence for that

1. PCout , MAR_{in} , Read , Select 4, Add, Z_{in}

taking address ~~for~~ out Program Counter
address is stored in MAR register
reading the data
increment the program counter (to get the next instruction)
by
Select 4 in the multiplexer
4 will come in A
PC conduct will come to B
then Add A & B
Result will come to Z

waiting for Memory function complete
↓
is a signal from memory (our instruction is read)

2. Zout , PC_{in} , WMFC

Zout will store the result in PC_{in}

3. MDR_{out} , IR_{in}

1. fetch the instruction from memory.

when the reading process is completed, it will come to MDR, then IR

4. R_3 out , MAR_{in} , Read

1000 stored in MAR then to memory bus, then reading

5. R_1 out , Y_{in} , WMFC

reading R_1 30 → receiving

6. MDR_{out} , Select Y , Add , Z_{in} → then it is stored in Z

↓ contains first operand

↓ contains second operand

↳ adding

7. Zout , R_{in} , End

Branch instruction

→ replaces the contents of PC with the branch target address, which is obtained by adding an offset X given in branch instruction

→ The offset X is the difference b/w the branch target address & the address immediately following the branch instruction

Execution of branch instructions:

Step	Action
1	PC_{out}, MAR_{in}, Read, $Select_4$, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End.

Conditional branch:

Offset field of IR_{out}, Add, Z_{in}, if N = 0 then End

If N = 0, the processor returns to step 1 immediately after step 4

If N = 1, step 5 is performed to load a new value in PC

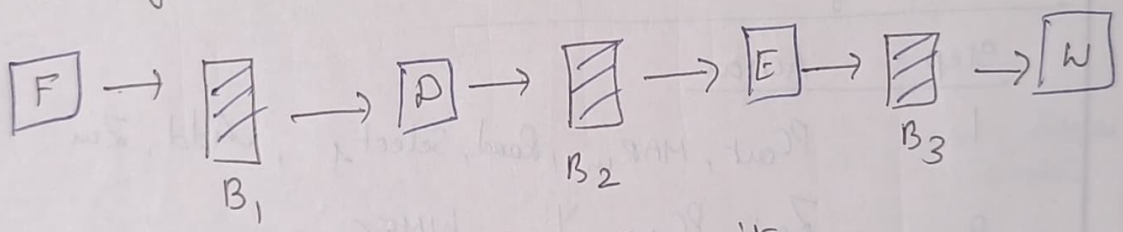
KRISHNASWAMY EDUCATIONAL TRUST, Chennai
Class ..... Sec ..... Reg. No. .....

Pipelining : Execution of machine instruction concurrently

instruction is divided into multiple stages

1) Fetch instruction from main memory/ cache
2) Decode instruction & fetch operands
3) Execute operation specified in instruction
4) Write the result back into register/memory

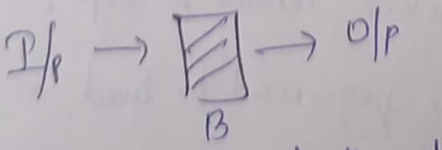
H/w orgn for a 4-stage pipeline



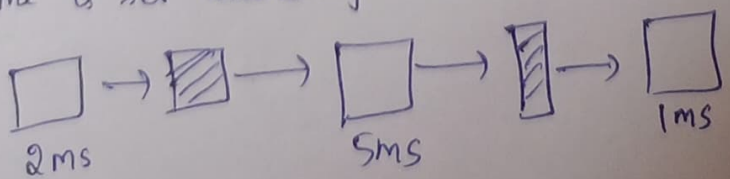
F, D, E, W are four distinct h/w units which can perform their operations independantly
↓
without interfering with one another

B_1 B_2 B_3 are buffers which holds the intermediate results generated by the predecessor units

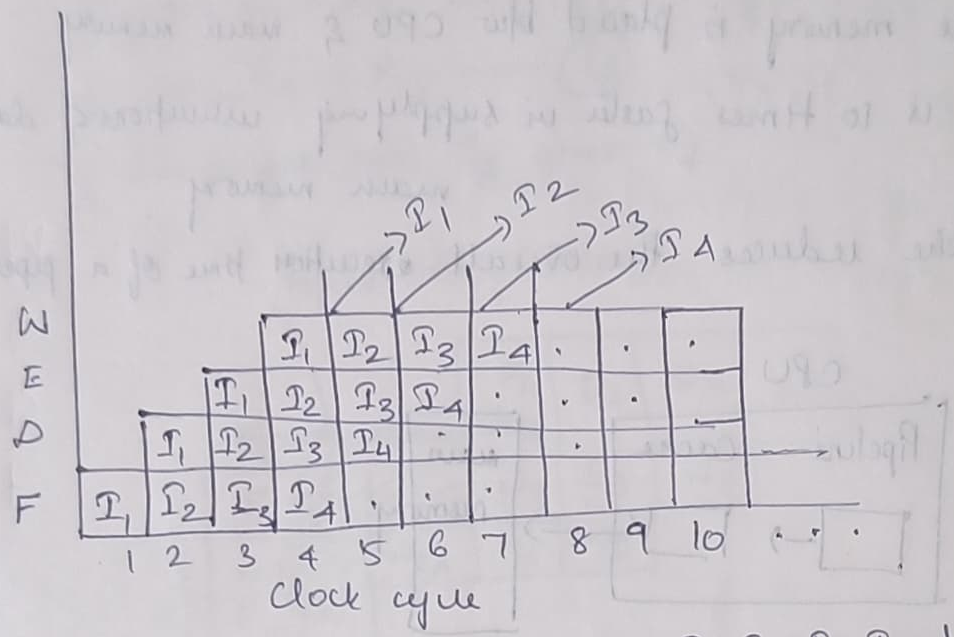
Pipeline operates in cycles. (or) in each cycle each buffer gives its output to the next unit.



Cycle time is set according to the time taken by the slowest H/w unit



cycle time is 5ms



In a pipeline system 4 instructions I_1 I_2 I_3 I_4 take 7 cycles to complete

In case of <u>non-pipeline system</u>

F_1	D_1	E_1	W_1	F_2	D_2	E_2	W_2	F_3	D_3	E_3	W_3	F_4	D_4	E_4	W_4
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

It takes 16 cycles to complete

So efficiency of a pipeline system is $7/16 = 0.4375$

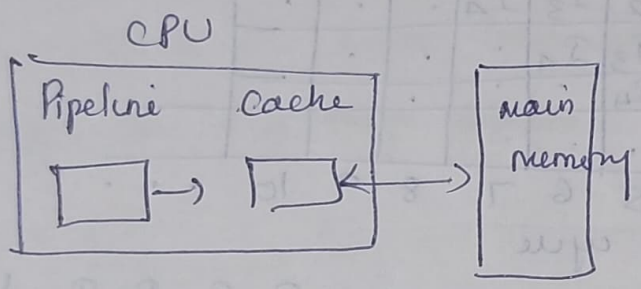
which means pipeline system is 44% more efficient than non pipeline sys

Role of Cache memory

Cache memory is placed b/w CPU & main memory

It is 10 times faster in supplying instructions & data than main memory.

Cache reduces the overall execution time of a pipeline system



Data hazards

It is a problem in a pipeline system which causes a stall the pipeline for one or more than one cycle

Data hazards arises when one/more than one of the operand is/are not available for the complete execution of an instruction

(for eg) $A \leftarrow 3 + A$ — I_1

$B \leftarrow 4 \times A$ — I_2

I_2 cannot generate correct result until I_1 is completed

So due to I_1, the execution of I_2 gets delayed (Stalling of pipeline)

$A \leftarrow 5 \times C$ — I_3

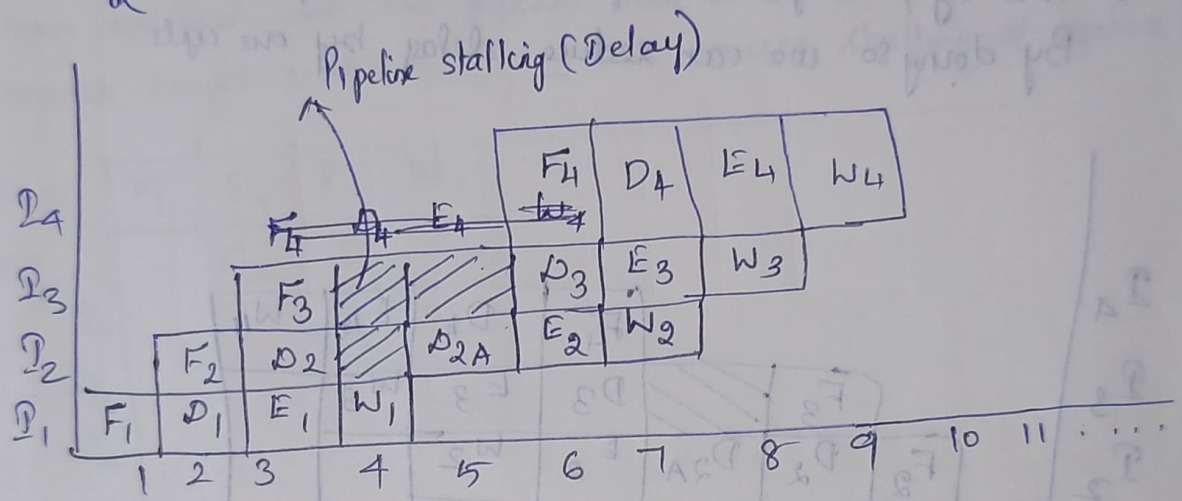
$B \leftarrow 20 + C$ — I_4

I_3, I_4 can run concurrently

$R_4 \leftarrow R_2 \times R_3$ — I_1

$R_6 \leftarrow R_4 + R_5$ — I_2

I_2 has to wait until I_1 gives R_4



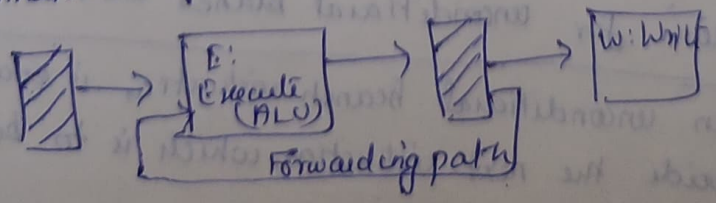
In the diagram,

I_2 has to wait until I_1 writes the result in R_4 register

the decoding stage of I_2 is waiting for the completion of writing of I_1

In I_3, since the decoding stage is busy with I_2, I_3 has to wait for its decoding, until decoding stage is freed by I_2

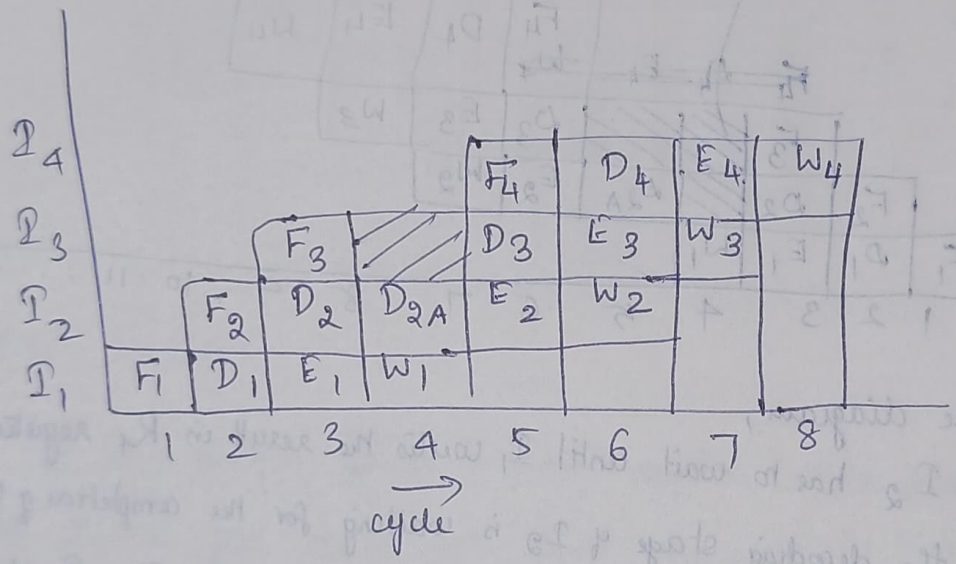
<u>Operand forwarding</u> : To deal with data hazards, operand forwarding can be used. In this approach, the results are directly forwarded to the place where they are needed without waiting to get them stored in destination registers



$R_4 \leftarrow R_2 \times R_3$ — I_1

$R_5 \quad R_4 + R_5$ — I_2

In this eg, implement operand forwarding.
By doing so we can reduce delay by one cycle



Instruction Hazards

Instruction fetch unit is supposed to supply instruction to decode unit but due to CACHE miss or due to BRANCH instruction, there may be some delay.

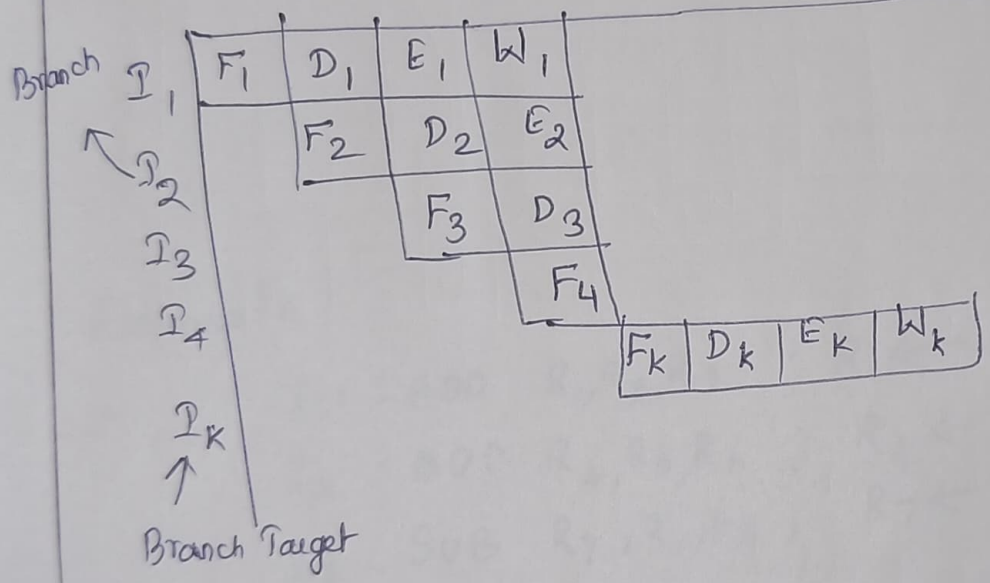
Branch instruction may be of 2 types
a) unconditional b) conditional

Effects & solution for unconditional branch instructions

When an unconditional branch instruction is encountered, it becomes essential to decide the next instruction, which is to be executed as BRANCH TARGET.

Normally it is decided by the execution unit

If the branch target instruction decided by the execution unit is not the next instruction, then the instructions that have been entered into the pipeline has to be removed, in order to load branch target instruction, which causes stalling of pipeline (delay)



Example:

I_1 : ADD R_1, R_2, R_3 ; $R_1 \leftarrow R_2 + R_3$

I_2 : ADD R_4, R_5, R_6 ; $R_4 \leftarrow R_5 + R_6$

I_3 : SUB R_7, R_1, R_5 ; $R_7 \leftarrow R_1 - R_5$

	1	2	3	4	5	6
IF	I_1	I_2	I_3			
ID		ADD R_2, R_3	ADD R_5, R_6	SUB R_1, R_5		
Ex			R_2+R_3	R_5+R_6	R_1-R_5	
WB				$R_1 \leftarrow$ Result	$R_4 \leftarrow$ Result	

R_1 is needed in both places

R_1 used by 2 different instruction for different purpose in the same clock cycle

Solution → Delay the Pipe (Pipeline Bubbling, Stalling)

Binary Arithmetic

Binary addition
Subtraction
Multiplication
Division.

Binary addition (Rule for binary addition)

Truth Table

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\boxed{1+1+1 = 1 \text{ carry } 1}$

1001 + 1011

$$\begin{array}{r} 1001 \\ 1011 \\ \hline 10100 \end{array}$$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0 = 20$

1001 → 9 (8 4 2 1)
1011 → 11
20

ex: 1) 2+3
2) 01001 + 10110

Binary Subtraction

Rules – Truth table

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Subtract $(0011)_2$ from $(1100)_2$

```
1100 – 8421 – 12
0011        – 3
----------
1001 ———→ 9
```

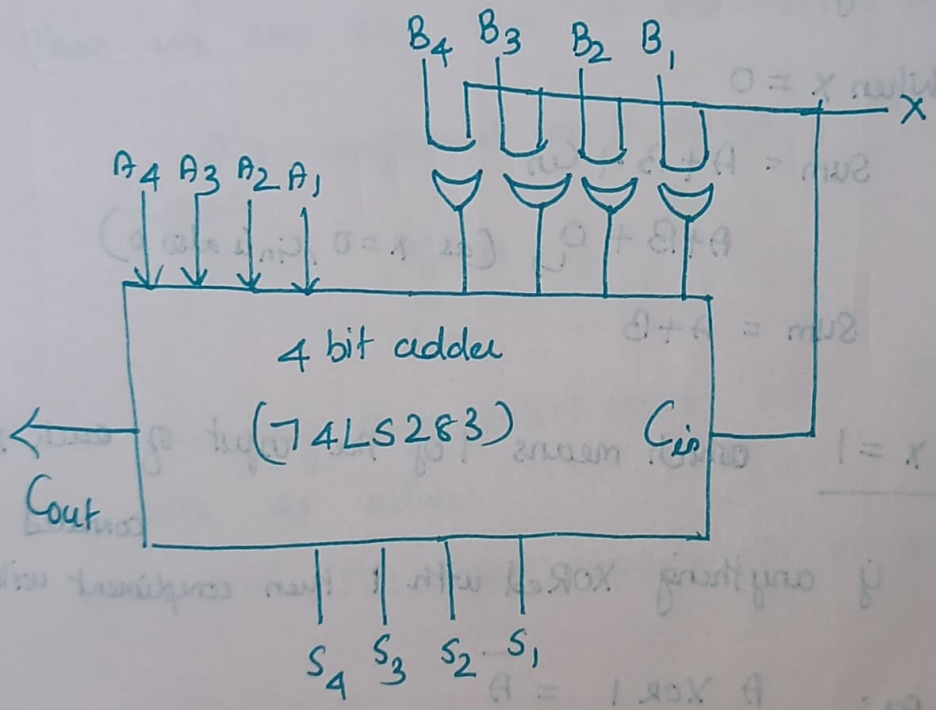
In this the 3 LSB is give the 1 to the 1st LSB so the borrow comes to the middle value

```
0011010
0011 00→

  0011010 → 26
  0001100 → 12
  -------
  0001110 → 14
```

4 Bit Parallel adder/subtractor

adder cum subtractor circuit
We can add/subtract 4 bit numbers



This 74LS283 IC acts as adder cum subtractor when it is connected to the 4 XOR gates

2 input XOR gate we have taken
1 input is coming from bits of a second number
2nd input is given to a common input X (ie) Carry in

when X = 0 (means providing 0 to all XOR gates as 1 input
0 XOR with anything means same thing we get (ie) $0 \oplus A = A$
$0 \oplus B = B$

$\therefore \quad X \oplus B_4 = B_4$

$X \oplus B_3 = B_3$

$X \oplus B_2 = B_2$

$X \oplus B_1 = B_1$

By using 4 bit adder $A+B+C$

When $X = 0$

Sum $= A + B + C_{in}$

$A + B + 0$ (as $X = 0$, C_{in} is also 0)

Sum $= A + B$

<u>When $X = 1$</u> which means 1 of the input of each XOR gate is connected to $X = 1$

if anything XORed with 1 then compliment will come

eg: $A \text{ XOR } 1 = \bar{A}$

$1 \text{ XOR } 1 = 0$

solve this, we are getting compliment of input variable here

So when 1 of the input is 1 is XOR gate, we get

$\bar{B}_4 \quad \bar{B}_3 \quad \bar{B}_2 \quad \bar{B}_1$

when $X = 1$

$A + (1$'s compliment of $B) + C_{in}$

$X = 1$ is this case

= A + 1's compliment of B + 1

= A + 2's compliment of B

We are doing this procedure in 2's compliment subtraction

we will find 2's compliment of subtrahend and add it to minuend

Then we can say this as subtraction process

2's compliment of B means $-B$

when

$\therefore$ $X = 1$, o/p = $A - B$
or
Sum

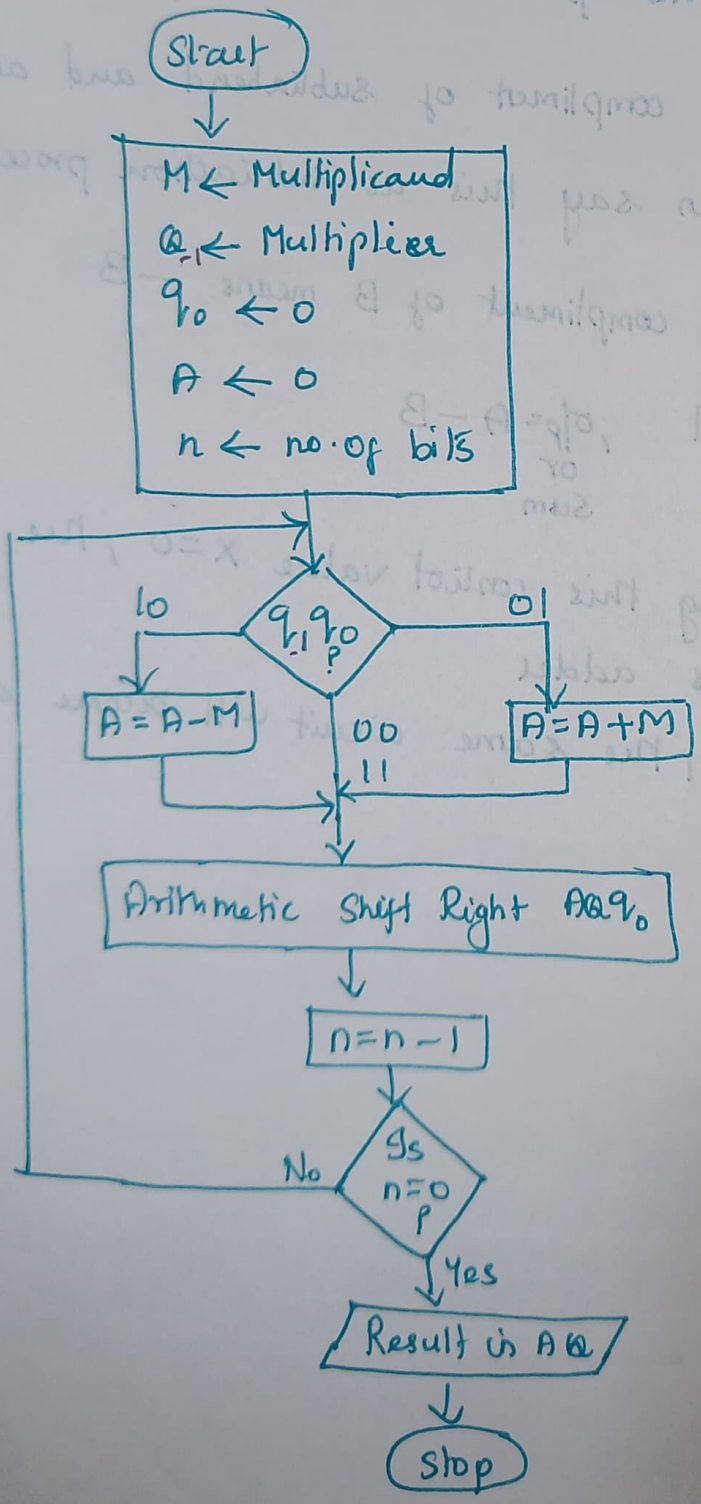
Just by changing this control value $X = 0$, the same circuit will act as adder

when $x = 1$, the same circuit will become as subtractor

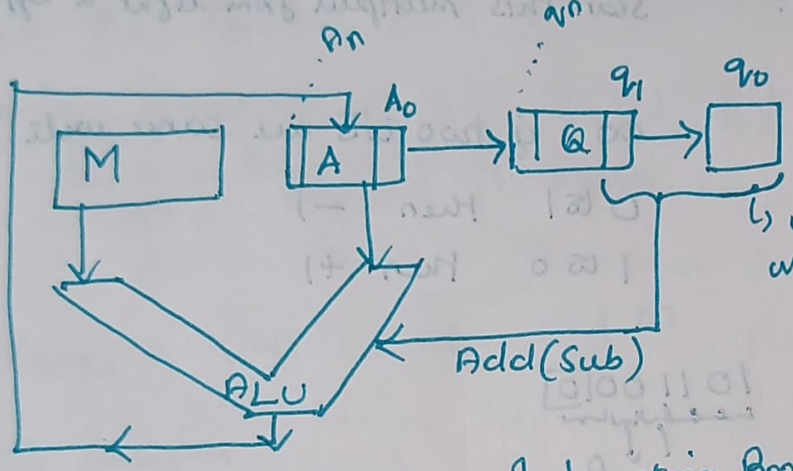
Signed Multiplication : Booth's Algorithm

Eg: $(-7) \times (+3) = -21$

Multiplicand → (-7); Multiplier → $(+3)$; product → -21



M, A, Q are n bits
q_0 1 bit



↳ will decide whether addition or subtraction

M register

A Q q_0 would be shifted to right hand side

Hardware Structure Implementing Booth's algorithm

$M = -7_{10}$
$= 2$'s compliment of 0111_2
$= 1000+1$
$= 1001_2$
$-M = 0111_2$

Tracing Table

n	A	Q (q_4 q_3 q_2 q_1)	q_0	Action / Comment
4	0000	0011	0	Initialisation
	0111	0011	0	A = A − M
3	0011	1001	1	ASR AQq_0
	0001	1100	1	ASR AQq_0
2	1010	1100	1	A = A + M
	1101	0110	0	ASR AQq_0
1	1110	1011	0	ASR AQq_0
0				

$$\begin{array}{r} 0001 - A \\ +\ 1001 - M \\ \hline 1010 \end{array}$$

2's compliment: 1's compliment of 1110 1011
$= 0001\ 0100$
$+\ 1$
$0001\ 0101$
$\Downarrow$ $(-21)_{10}$

~~$-(21)_{10}$~~

Sign is 1 so value will be (−)

→ To get its magnitude I shall go for its 2's compliment calculation

But in case if the first bit is zero, I can go the decimal equivalent of the binary number

Booth's algorithm: Signed multiplication (Recoded)

Multiplier : 101100 $\boxed{0}$ implied zero

Recoded multiplier : scan this multiplier from right to left

00 if two bits are same write 0

0 to 1 then −1

1 to 0 then +1

101100$\boxed{0}$

−1 +1 0 −1 0 0

(eg) Find recoded multiplier for 011001

011001 0 implied zero

recoded multiplier = +1 0 −1 0 +1 −1

Multiply : multiplier : 001100 multiplicand : 010011

001100 $\boxed{0}$ implied zero

0 +1 0 −1 0 0 (recoded multiplier)

2's compliment of multiplicand = 101101

(−1) we have to write the 2's compliment of multiplicand

```
                  0 1 0 0 1 1      6 bits
                  0 +1 0 -1 0 0    6 bits
                                   so product should be 12 bits
  0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0
  1 1 1 1 1 0 1 1 0 1
  0 0 0 0 0 0 0 0 ✓
  0 0 0 1 0 0 1 1
  0 0 0 0 0 0 0
  ---------------------------------
  0 0 0 1 1 1 0 0 1 0 0
```

extend the remaining with the signed bit

multiply by +1 write actual multiplicand

extending the signed bit

negative numbers representing 2's compliment form

positive result

ignore the carry in 2's compliment form

So

IEEE 754 Floating point Representation.

Single Precision 32 bits

1	8	23 bits
sign	Exponent	Mantissa
s	e	m

$$X_{FP32} = (-1)^s \times 2^{e-127} \times 1.m$$

Double Precision

64 bits

1	11	52
sign	Exponent	Mantissa

$$X_{FP64} = (-1)^s \times 2^{e-1023} \times 1.m$$

What is the decimal equivalent of the floating number 40400000H

each hexadecimal digit is 4 bits 1 nibble

8 hexadecimal digit is 32 bits 4 nibbles

so it is single precision

40400000H = 0100 0000 0100 0000 0000 0000 0000

sign | Exponent | mantissa

$s = 0$

$e = 2^7 = 128$

$m = 0.5$

$$X_{FP32} = (-1)^0 \times 2^{128-127} \times 1.5$$

$$= 2 \times 1.5 = 3_{10}$$

IEEE 754 documentation there are 4 special cases

e	m	inference
255	0	NaN (not-a-number) (eg. error - divide by zero → sqrt of a negative no.)
255	≠0	Infinite number
0	0	then the number can be expressed as $x = (-1)^s \times 2^{-126} \times (0.m)$
0	≠0	0, $(-1)^s \times 0$, Again +0 and −0 are possible

Express 10_{10} in IEEE 754 Single precision floating point representation

$$10 = (-1)^0 \times 2^{130-127} \times 1.25$$

$$= 0 \,|\, 100\ 0001\ 0 \,|\, 010\ 0000\ 0000\ 0000\ 0000\ 0000$$

2^3

$10 = 8 \times 1.25$

$130 = 128 + 2$

$= 2^7 + 2$

4 1 2 0 0 0 0 0 H

Evaluate the floating point number 40A00000H in decimal form

40A00000H

$$0 \,|\, 100\ 0000\ 1 \,|\, 010\ 0000\ 0000\ 0000\ 0000\ 0000$$

s e m

$$X = (-1)^0 \times 2^{129-127} \times 1.25$$

$$= 1 \times 2^2 \times 1.25$$

$$4 \times 1.25 = 5_{10}$$