Monday

ARITHMATIC

R10 Page No. 46
Date 23 / 09 / 2019

* Addition and subtraction of signed numbers: The arithmatic operations are performed and built using logical gates

x_i	y_i	c_i (Carry-in)	s_i	Carryout (c_{i+1})
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

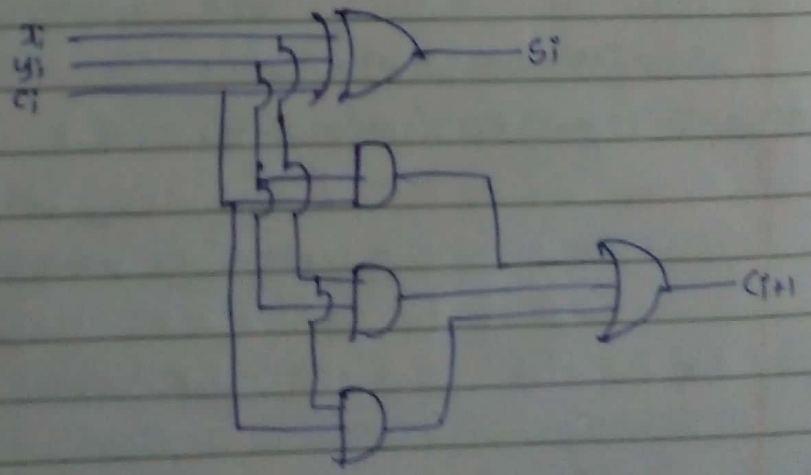
$s_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i$

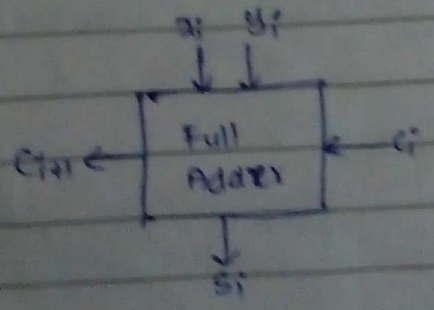
$c_{i+1} = \bar{x}_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i$

$c_{i+1} = y_i c_i (\bar{x}_i + x_i) + x_i (\bar{y}_i c_i + y_i \bar{c}_i)$

$c_{i+1} = y_i c_i + x_i y_i + x_i c_i = x_i y_i + c_i (x_i + y_i)$

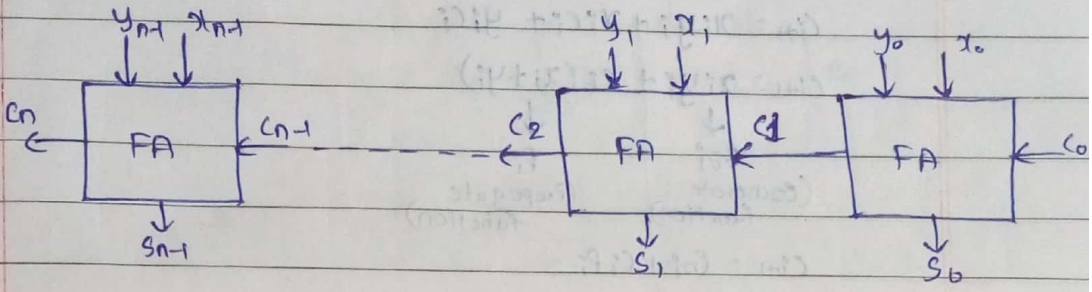
$s_i = x_i \oplus y_i \oplus c_i$



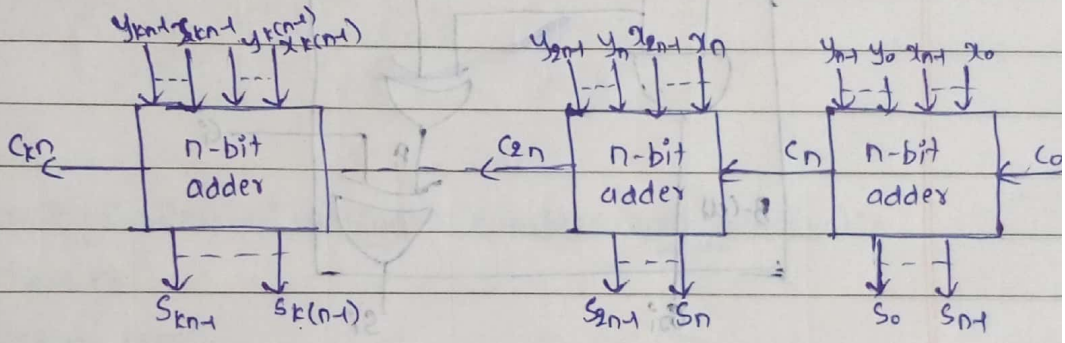


* n-bit ripple carry adder: A cascaded connection of n full adders to build add a bit x and y numbers since the

carry must propogate or ripple through these cascaded adders. It is called n-bit ripple carry adder.

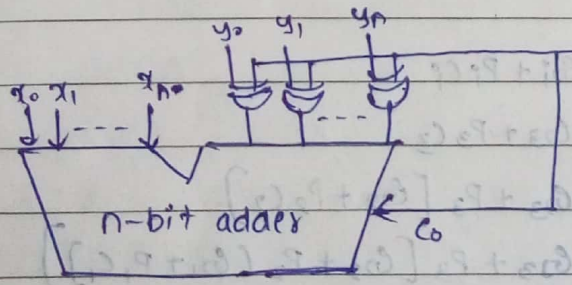


* Cascaded k n-bit adders:



Wednesday 25-09-2019

* Binary addition and subtraction logic network:



x	y	x⊕y
0	0	0
0	1	1
1	0	1
1	1	0

When Add/Sub = 0 → Addition.
∴ y input → unchanged - with XOR gates.
When Add/Sub = 1 → Subtraction.
y ⊕ Add/Sub → 1's complement
$C_0 = 1$

* Design of fast adders:

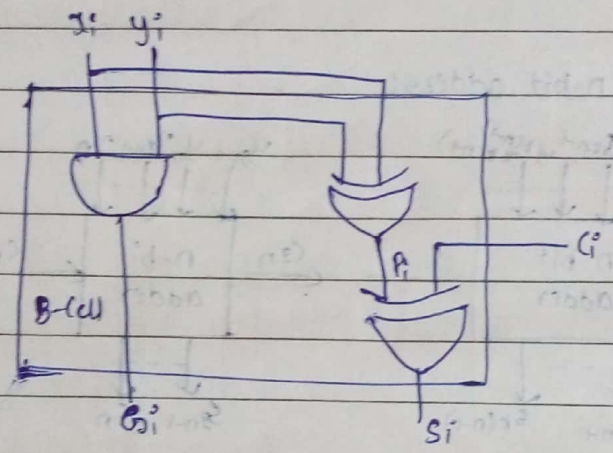
$s_i = x_i \oplus y_i \oplus c_i$

$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$

$c_{i+1} = x_i y_i + c_i (x_i + y_i)$

g_i (Generate function) $\quad p_i$ (Propagate function)

$c_{i+1} = g_i + c_i p_i$



*Carry look ahead adder:

Design and derive equation for 4-bit carry look ahead adder.

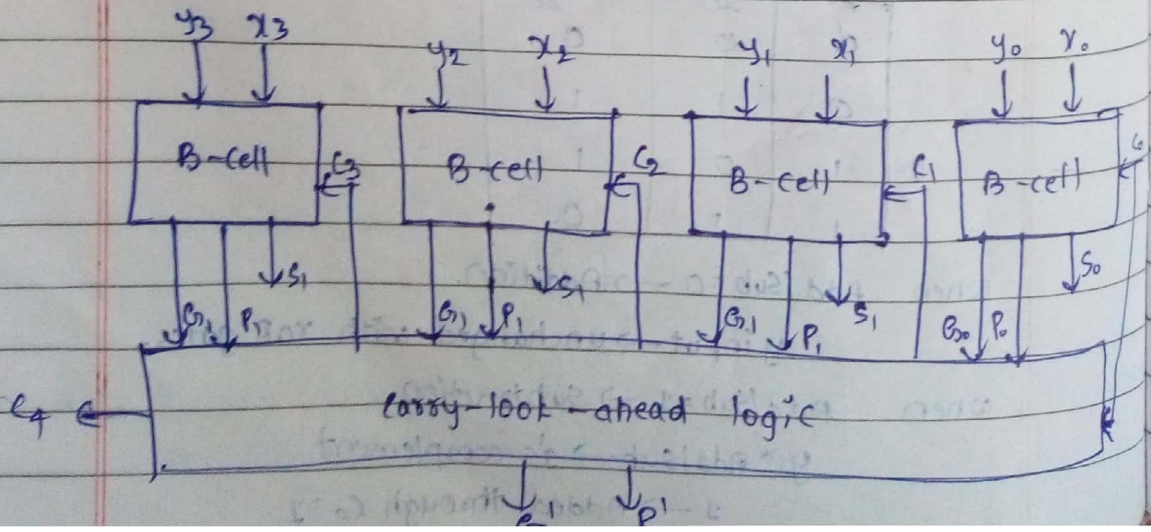
$c_{i+1} = g_i + p_i c_i$

$c_4 = g_3 + p_3 c_3$

$c_4 = g_3 + p_3 [g_2 + p_2 c_2]$

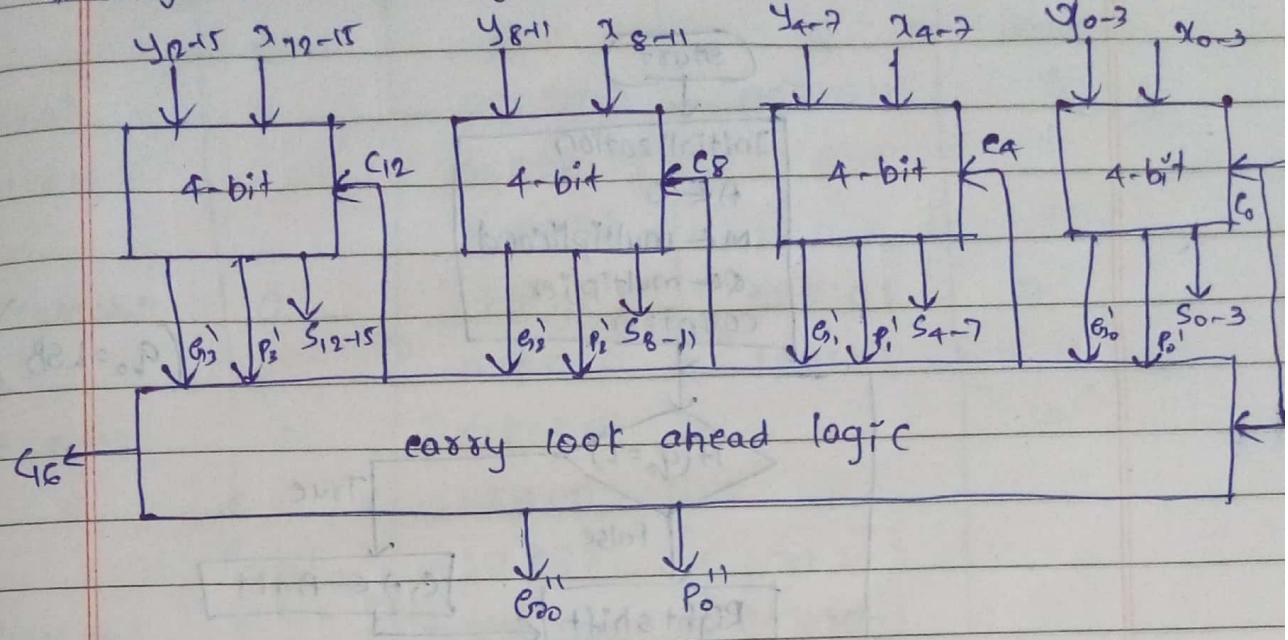
$c_4 = g_3 + p_3 [g_2 + p_2 (g_1 + p_1 c_1)]$

$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$



Date / /

* Design a 16-bit carry lookahead adder using 4-bit adders



* Multiplication of unsigned numbers (only positive):

1) 12 x 13

12 - 1100

13 - 1101

1100 * 1101

1100

0000

1100

1100

10011100

2) 13 x 15

13 - 1101

15 - 1111

1101 x 1111

1

1101

1 0 1.0 1 (∵ 4 = 100)

1

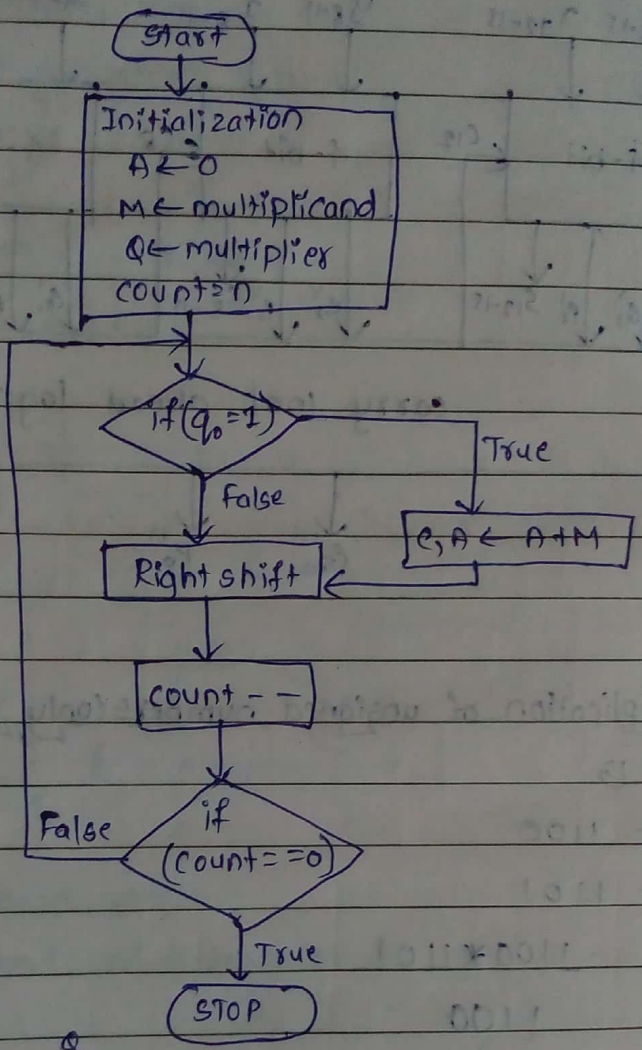
1101

1 0 1101

11000011

Thursday

RIO Page No. 50
Date 26 / 09 / 2019



*. 13 * 11 ⇒ 1101 * 1011
(M = 1101, Q = 1011)

	C	A	Q	Count
initial	0	0000	1011	4
A = A+M		0000		
		1101		
Right shift	0	1101	1011	3
	0	0110	1101	
A = A+M	1	0110		2
		1101		
Right shift	1	0011	1011	
		0001	1101	1
Right shift	0	0100	1111	1
		1000	1111	

*. 33 * 5 ⇒ 10000 (M) * 101 (Q)

	C	A	Q	Count
initial	0	000000	000101	5
A = A + M		000000		
		100001		
Right shift	0	100001	000101	5
		010000	100001	
A = A + M	0	010000		4
		100001		
	0	110001	100001	
Right shift		011000	110001	
A = A + M	0	011000		
		100001	110001	3
Right shift	0	111001	100001	
		011100	100001	
A = A + M	0	011100		2
		100001		
Right shift		111101	100001	
		011110	111101	
A = A + M	0	011110		1
		100001		
		111111		
		011111	111111	

Friday

Page No. 51
Date 27/09/2019

**** Division operation: There are 2 ways of performing division operation.

1) Restoring division.

2) Non-restoring division.

1) Restoring division:

$$\frac{12}{3} = \frac{\text{Divident [Q]}}{\text{Divisor [M]}}$$

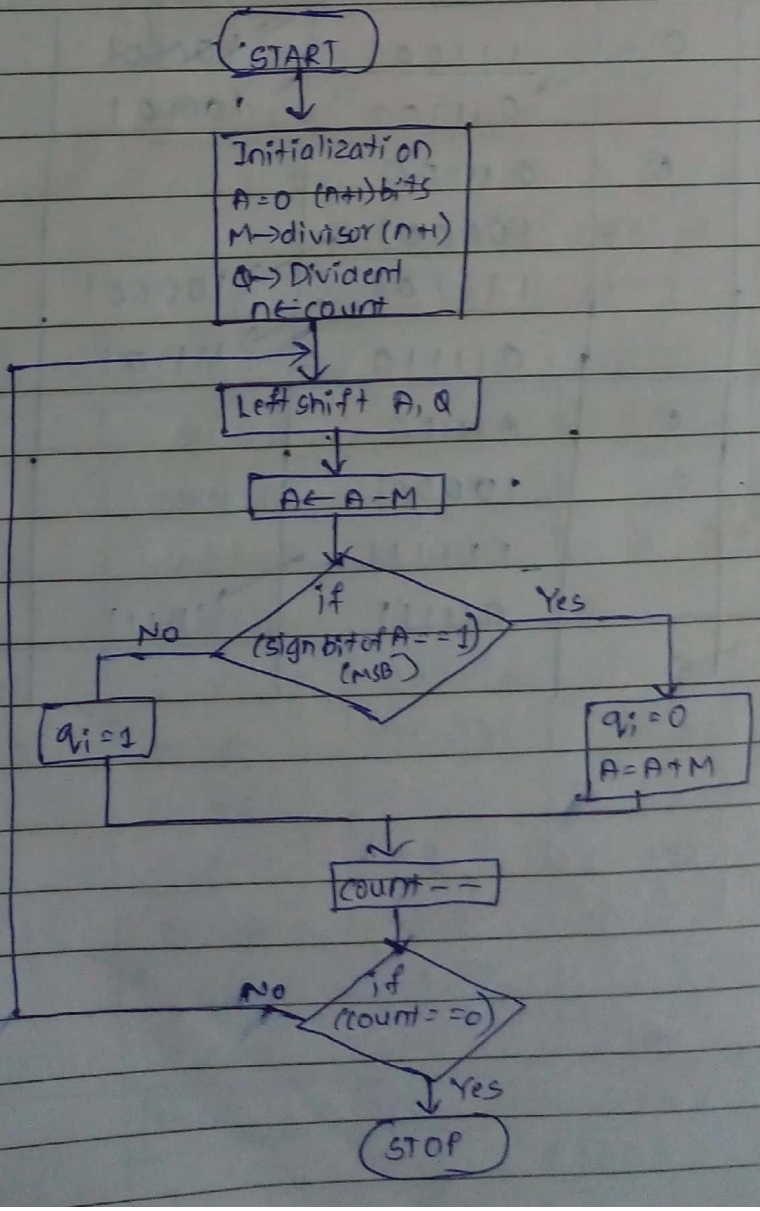
Result = A [Quotient Q]

Remainder = A

Initially, A will be initialized with [n+1] bits.

M → [n+1] bits

Q → n bits



* 10/3 → Q = 1010 M = 0011 A = 0

Operation	A	Q	Count
Initial	00000	1010	4
Left shift	00001	010□	3
	11101	010□	
$q_i = 0$	11110	010□	
A+M	11110		
	00011		
	00001	0100	
Left shift	00010	100□	2
	11101		
$q_i = 0$	11111	100□	
A+M	11111		
	00011		
	00010	1000	
Left shift	00101	000□	1
	11101		
$q_i = 0$	00010	000□	
A+M	00010		
	00011		
	00101		
Left shift	01010	001□	0
	00001	0011	
	Remainder	Quotient	

A − M
= A + 2's(M)

M = 00011
00011
11100
11101
↓
2's(M)

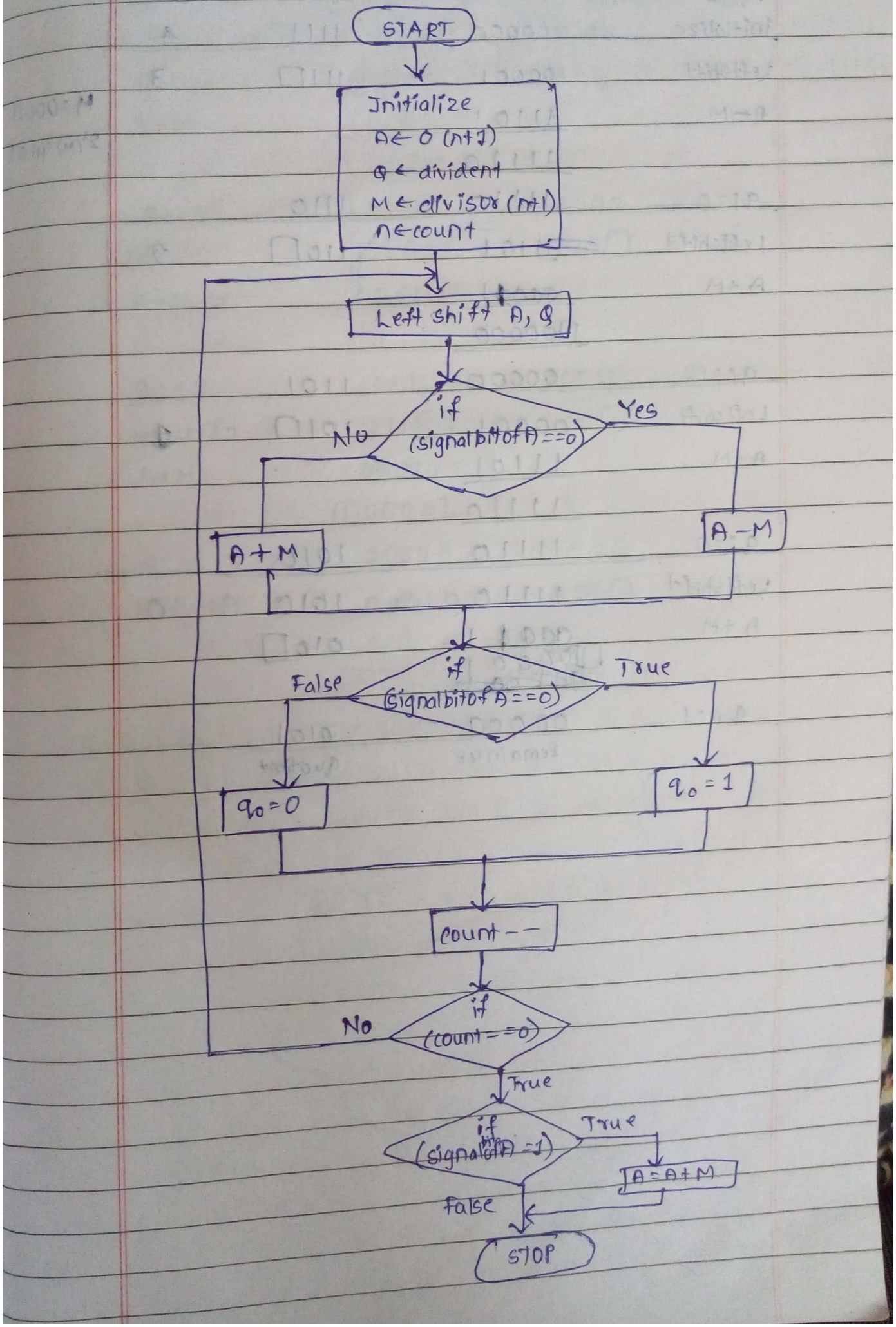
*. 15/9 ⇒ Q = 1111 M = 1001 A = 0000

Operation	A	Q	count	
Initial	00000	1111	4	M = 01001
Left shift	00001	111□	3	2's M = 10111
	10111			
$q_i = 0$	10111	111[0]		
A = A+M	10111			
	01001			
	[1]00000	1110		
Left shift	00001	110□	2	
	10111			
	11000			
$q_i = 0$	11000	1100		
A = A+M	01001			
	[X]00001	1100		
Left shift	00011	100□	1	
	10111			
$q_i = 0$	11010	1000		
A = A+M	11010			
	01001			
	[X]00011	1000		
Left shift	00111	000□	0	

00110 | 0001

Remainder Quotient

2) Non-restoring division:



* 15/3 , 15 - 1111 , 3 - 0011 n = 4

Operation	A(n+1)	Q(n)	Count
Initialize	00000	1111	4
Leftshift	00001	111□	3
A-M	11101		
	11110		
$q_i = 0$	11110	1110	
Leftshift	11101	110□	2
A+M	00001		
	100000		
$q_i = 1$	00000	1101	
Leftshift	00001	101□	1
A-M	11101		
	11110		
$q_i = 0$	11110	1010	
Leftshift	11110	1010	0
A+M	00001	010□	
	00000		
$q_i = 1$	00000	0101	
	Remainder	Quotient	

M = 0011
2's(M) = 1101

* 8/3 , 8 - 1000 , 3 - 0011 n = 4

operation	A(n+1)	Q(n)	count	
initialize	00000	1000	4	M - 0011
leftshift	00001	000□	3	2's - 11101
A - M	11101			
	11110			
$q_0 = 0$	11110	0000		
leftshift	11100	000□	2	
A + M	00011			
	11111			
$q_1 = 0$	11111	0000		
leftshift	1111[0]	000□	1	
A + M	00011			
	[1]00001			
$q_2 = 1$	00001	0001		
leftshift	00010	001□	0	

***** * Booth's algorithm: Multiplication of signed numbers.

It is a powerful algorithm used for multiplication of signed numbers. It works by recoding the multiplier. Before recoding, and extension of a LSB bit with 0 has to be considered for a multiplier.

Bi	Bi-1	Recording-bit
0	0	0
0	1	+1
1	0	-1
1	1	0

Ex for decoding: 1) 10101101

Add 0 at end ⇒ 10101101 0

+1 -1 +1 -1 0 +1 -1 0

-1 +1 -1 +1 0 -1 +1 -1

2) 1110110101

Add 0 at LSB ⇒ 1 1 1 0 1 1 0 1 0 1 0

0 0 -1 +1 0 -1 +1 -1 +1 -1

ALGORITHM: ~~Add~~

Steps: 1) Add 0 at MSB for both multiplicant & multiplier.

2) Perform 2's complement of multiplicant and multiplier which will be required for processing of multiplication.

3) Perform 2's complement of given numbers whenever it is multiplied with negative value.

4) Perform decoding of multiplier along with extra zero bit at LSB.

Thursday 03-10-2019

5) ~~If the multiplier bit (qi = 1), substitute~~

If $q_i = 0$, multiply 0 with multiplicant

If $q_i = 1$, multiply 1 with multiplicant.

If $q_i = -1$, multiply -1 with multiplicant = 2's (M).

6) Result should be of 2n bits.

1) 13*12

13 - M - Multiplicant - 11013

12 - Q - Multiplier - 11010

Add 0 at MSB - M - 011010

Q - 011010

Recoding of Q - 011000 0

0 1 1 0 0 0 0

1 0 -1 0 0 0

011010 * +10-100

Result is 20bits 0000000000

0000000000

000100111 ——→ 2's of M (∵ -1 * M = 2' M

0000000

001101

001001111 00

2) -20*18

Q - 18 - 10010

M - 20 - 10100

Add 0 at MSB - M - 010100 —2's→ 101100

Q - 010010

Recoding of Q - 101100 0

1 0 1 1 0 0 0

-1 1 0 -1 0 0

101100 * +1 -1 0 +1 -1 0 0

000000000000

000000000000

000000101100

000010110 0

000000000

000101100

010110 0

-0 1 1 1 0 0 1 1 0 0 0

3) 25 * -8

M - 011001

Q - 001000

Recoding Q → 0 1 1 0 0 1 0
+1 0 -1 0 +1 -1

011001 * +1 0 -1 0 +1 -1

000000100111

000000011001

000000000000

000100111

000000000

001100 1

0111000111000

4) -32 * -12

M - 01000000 2's 11000000

Q - 00001100 2's 11110100

Recoding Q - 1 1 1 1 0 1 0 0
0 0 -1 +1 -1 0

11000000 * 0 0 -1 +1 -1 0

0000000000000000

0000001000000

0000011000000

0000100000000

0000000000000

0000000000

0000000000000

* Bit pair recoding:

Bi+1	Bi	Bi-1	Recoding
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

Step: 1) Add 0 at MSB bit position of both M and Q.
2) If M or Q are negative, convert it into 2's complement
3) To recode the multiplier add 0 at LSB and the sign extension of 1bit at MSB.
4) Multiply M with the recoded bits

$q_i = 0 \rightarrow 0 * M$

$q_i = 1 \rightarrow 1 * M$

$q_i = -1 \rightarrow 1 * 2's(M)$

$q_i = +2 \rightarrow (1 * M) + Leftshift$

$q_i = -2 \rightarrow (1 * 2's(M)) + Leftshift$

Example for recoding:

1) 1 0 1 0 0 1 0

Sign-extension, LSB

1 1 1 0 1 0 0 1 0

0 -1 -2 +1

2) 0 1 0 1 0 1 1 0

Sign-extension, LSB

0 0 1 0 1 0 1 1 0

+1 -1 -1 -1

1) 13*6

M - 13 - 01101

Q - 6 - 00110

Q - 0 0 1 1 0

011010 * 0 +2 -2

111111001 1

lets shift ↙ 1111110011 0

Left shift 0000001 101 * * (Skip 2 parts)

~~0001101***~~

0000000 ****

0001001110

2) 22 * (-18)

Q - 010010

M - 010110

2s (Q) = 10 111 0 0

0 -1 0 -2

010110 * 0 -1 0 -2

0101100

3) -15 * -9

M - 01111 $\xrightarrow{2's}$ 10000 $\xrightarrow{+1}$ 10001

Q - 01001 $\xrightarrow{1's}$ 10110 $\xrightarrow{+1}$ 10111

Q = 1 1 0 1 1 0

-1 +2 -1

10001 * -1 +2 -1

0000001111

1 1 1 1 0 0 1 0 * *

1 1 1 0 0 1 * * * *

1 1 0 1 1 0 0 1 1 1

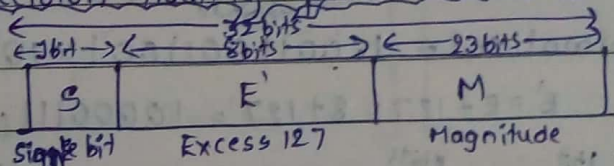
* Representation of floating point numbers: Floating pointing numbers are represented by mantissa e exponent.

$12.53 * 10^6 \Rightarrow 12.53 e 6$

There are 2 ways of representing

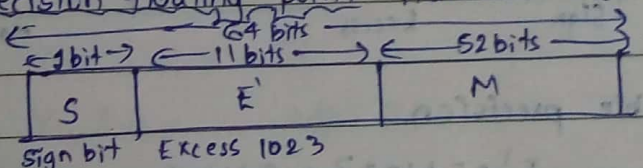
1) single precision floating point numbers
2) Double precision floating point numbers

1) single precision floating point numbers (32 bits)



Unsigned: $0 \le E' \le 255$ Signed: $-126 \le E' \le 127$

2) Double precision floating point numbers: (64 bits)



Unsigned: $0 \le E' \le 2047$

Signed: $-1022 \le E' \le 1023$

1) Represent the following numbers into single and double precision.

(i) 1258.52

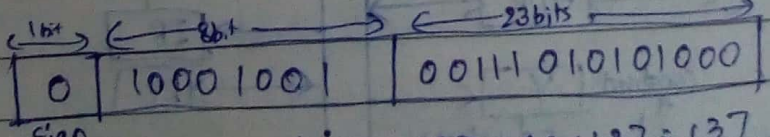
single presicion.

1258 → 10011101010.1000

Unnormalized ← 10011101010.1000

Normalized ← $1.0011101010101000 \times 2^{10}$ (Magnitude)

0.52 × 2
1.04
0.04 × 2
0.08
0.08 × 2
0.16

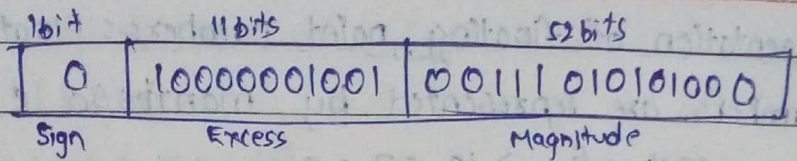


$E' = E + 127 = 10 + 127 = 137$

$137_{(10)} = 10001001_{(2)}$

Double precision.

$E' = E + 1023 = 10 + 1023 = 1033 = 10000001001_{(2)}$



(ii) - 307.1875.

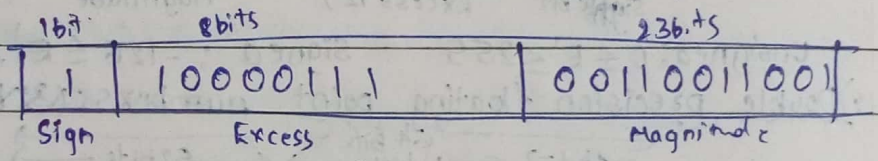
Single precision.

$307.1875 = 100110011.001$

Unnormalized - 100110011.001

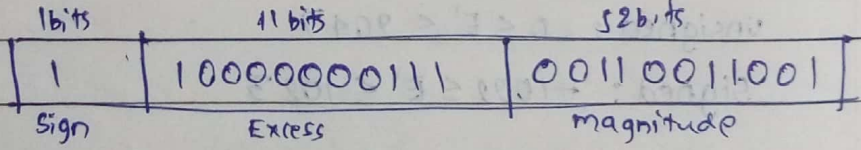
Normalized - $1.00110011001 \times 10^8$ (magnitude)

$E' = E + 127 = 8 + 127 = 10000111_{(2)}$



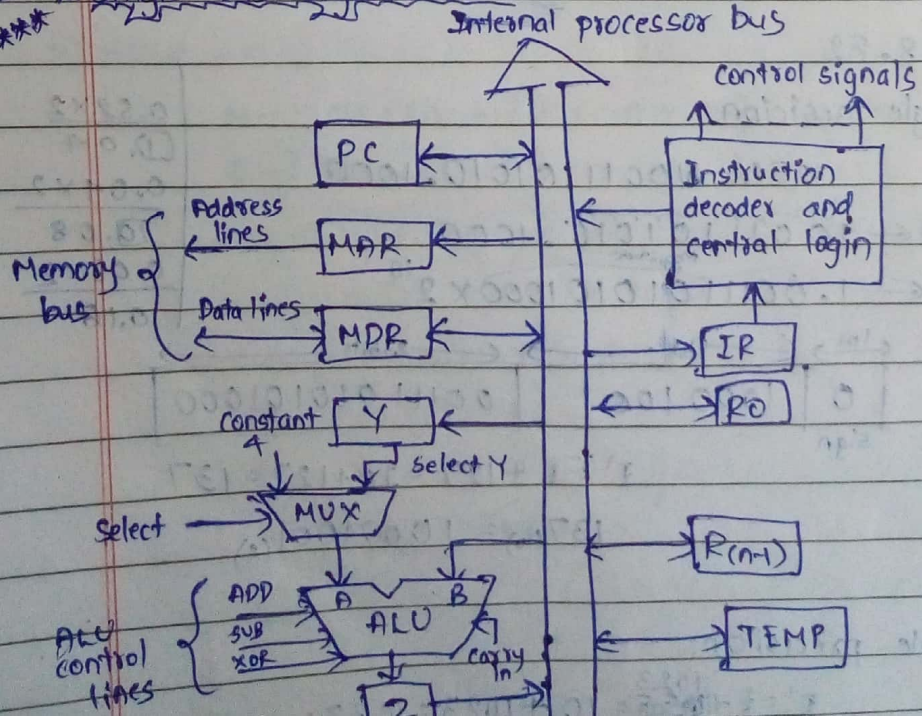
Double precision

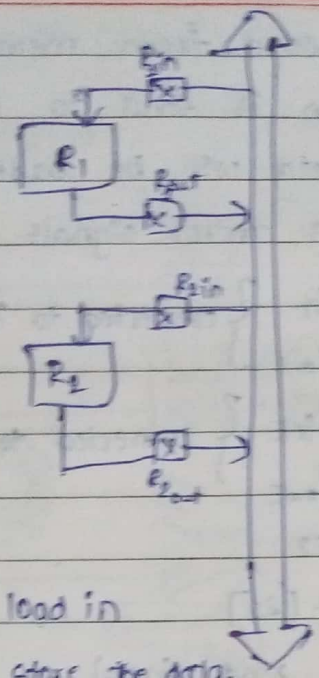
$E' = E + 1023 = 8 + 1023 = 1031 = 10000000111_{(2)}$



Thursday
10-10-2019
*****

* Single bus organization:





move R_1, R_2

1) Enable R_{1in} and load in

2) Enable R_{2in} and store the data.

* Write the control sentence instruction for Add R_2, R_3.

1) R_{2out}, Y_{in}

2) Select Y, R_{3out}, Add, Z_{in}

3) Z_{out}, R_{3in}.

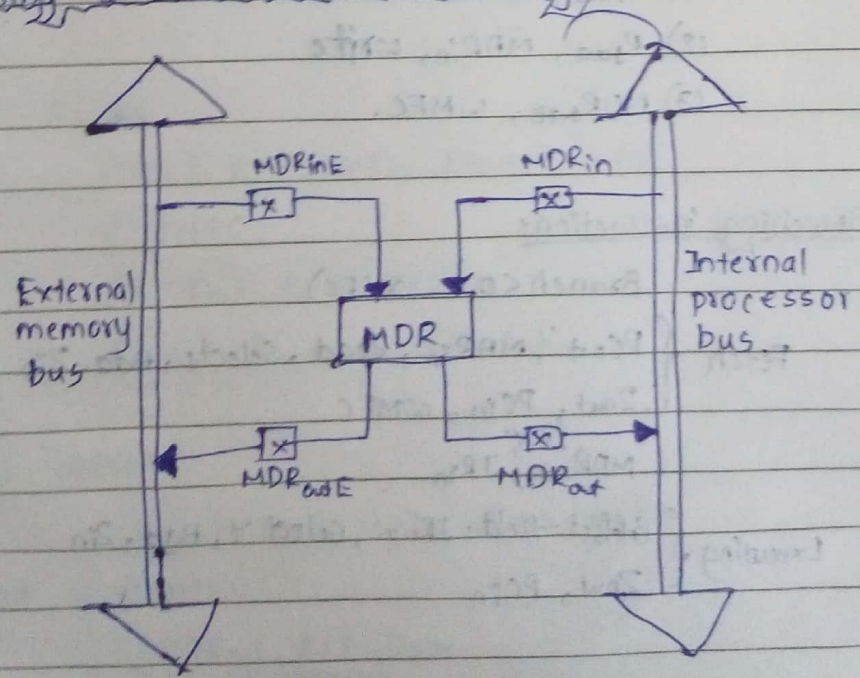
* mul R_4, R_5, R_6

1) R_5, R_6

2) R_4, R_6

Wednesday
30-10-2019

* Fetching a word from the memory:



To fetch a word from memory, the address of memory location is send to MAR, MAR initiates Read signal. After WFMC data will be loaded to MDR.

MDR has 4 control signals

1) MDR_{in} } connected to internal bus.
2) MDR_{out} }
3) MDR_{inE} } connected to external memory bus.
4) MDR_{outE} }

*. move (R1), R2

(1) MAR ← [R1]
(2) Initiate read signal.
(3) Data ~~WBMFC~~
(4) Data → loaded to MDR
(5) R2 ← [MDR].

(1) R_{1out}, MAR_{in}, Read
(2) MDR_{inE}, WMFC.
(3) MDR_{out}, R_{2in}.

*. move R_1, (R_2)

(1) R_{2out}, MAR_{in}
(2) R_{1out}, MAR_{in}, write
(3) MDR_{outE}, WMFC.

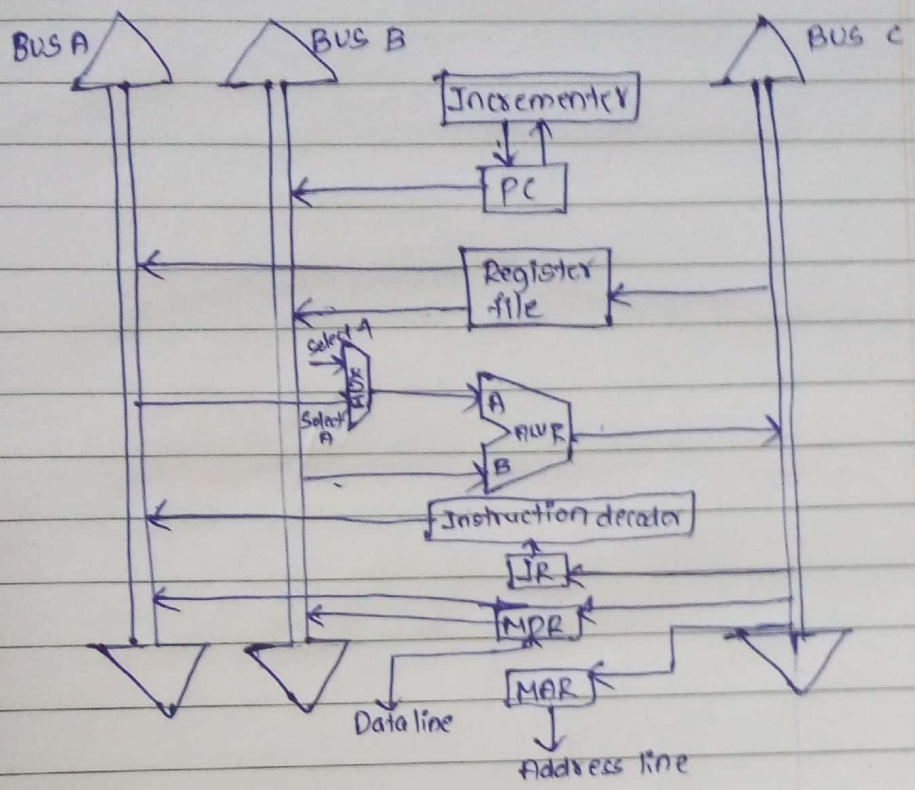
*. Branching instruction:

Branch<0 → X(PC)

Fetch { PCout, MAR_{in}, Read, Select4, Add, Z_{in}
Zout, PC_{in}, WMFC
MDR_{out}, IR_{in} (Instruction)

Executing { Offset-field-IRout, Select Y, Add, Z_{in}
Zout, PC_{in}

** Multiple BUS organization (3 bus):



Add R_1, R_2, R_3 $_{R=B}$

Fetch
1) PC_{out}, MAR$_{in}$, Read, Inc PC
2) WMFC
3) MDR$_{out}$, R=B, IR$_{in}$

4) R_{1out}, R_{2out} B, Select A, Add, R_{3in}

Sub R_1, R_2

1) PC_{out}, R=B, MAR$_{in}$, Read, Inc PC.
2) WMFC.
3) MDR$_{out}$, R=B, IR$_{in}$
4) R_{1out}, R_{2out}, Select A, Sub.

Add (R1), R2

Fetch
1) PC_{out}, R=B, MAR$_{in}$, Read, Inc PC
2) WMFC.
3) MDR$_{out}$, R=B, IR$_{in}$

Execute
4) R_{1out}, R=B, MAR$_{in}$, Read, Inc PC
5) WMFC
6) R_{2out}, Select A, Add R_{3in}