RAMAIAH Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Computer Organization and Architecture

Course Code: CS45

Credits: 3:1:0

UNIT 2

Term: March – June 2021

Faculty:

Chandrika P
Vandana S Sardar
Sowmya B J
Dr.Sangeetha V

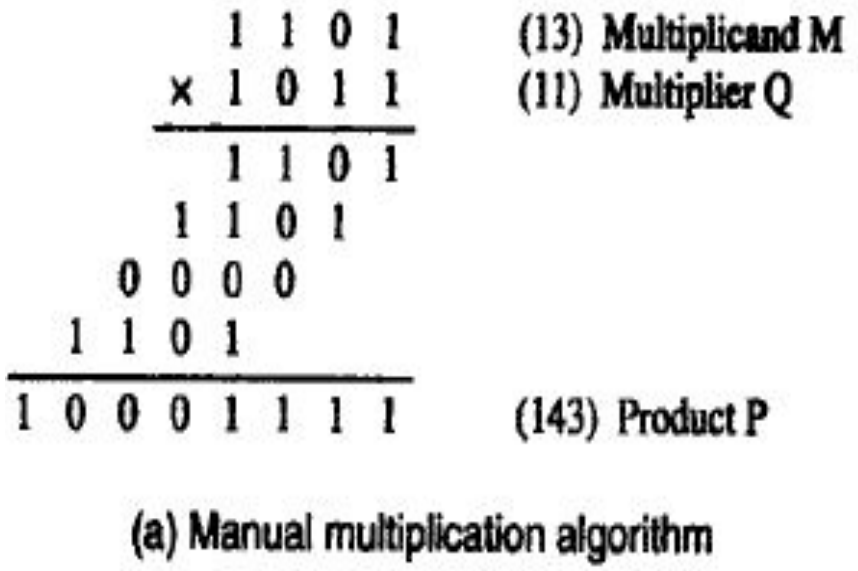
RAMAIAH
Institute of Technology

UNIT 2 – Arithmetic unit & Processing unit

- Multiplication of two Numbers
- A Signed Operand Multiplication
  - Booth algorithm
- Fast Multiplication
  - Bit pair recoding
  - Carry Save Addition(CSA)
- Integer Division
  - Restoring Division
  - No restoring Division
- IEEE standard for floating point numbers
- Fundamental concepts
- Execution of complete instruction
- Multiple bus organization
- Hardwired control
- Microprogrammed control

Multiplication of unsigned numbers

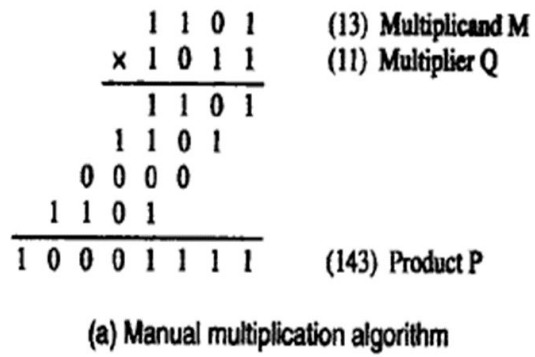
- Algorithm for multiplying unsigned number and positive signed numbers.
- The product of two n-digit number can be accommodated in 2n digits
- n=4 digits 2*4= 8 digits



(a) Manual multiplication algorithm

Multiplication of unsigned numbers

- Algorithm for multiplying unsigned number
  - If the multiplier bit is 1, the multiplicand is entered in the appropriate position to be added to the partial product.
  - If the multiplier bit is 0, then 0s are entered in the appropriate position to be added to the partial product.



(a) Manual multiplication algorithm

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

M					**1**	**1**	**0**	**1**
Q					1	0	1	1
P								

2	13	
2	6	1
2	3	0
	1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
					1	1	0	1
P								

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
					1	1	0	1
				1	1	0	1	
P								

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

M					**1**	**1**	**0**	**1**
Q					1	0	1	1
					1	1	0	1
				1	1	0	1	
			0	0	0	0		
P								

2	13	
2	6	1
2	3	0
	1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
					1	1	0	1
				1	1	0	1	
			0	0	0	0		
		1	1	0	1			
P								

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
					1	1	0	1
				1	1	0	1	
			0	0	0	0		
		1	1	0	1			
P								1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
					1	1	0	1
				1	1	0	1	
			0	0	0	0		
		1	1	0	1			
P							1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
					1	1	0	1
				1	1	0	1	
			0	0	0	0		
		1	1	0	1			
P						1	1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
				1	1	1	0	1
				1	1	0	1	
			0	0	0	0		
		1	1	0	1			
P					1	1	1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
				1	1	1	0	1
			1	1	1	0	1	
			0	0	0	0		
		1	1	0	1			
P				0	1	1	1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
				1	1	1	0	1
			1	1	1	0	1	
		1	0	0	0	0		
		1	1	0	1			
P			0	0	1	1	1	1

RAMAIAH
Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

2	13	
2	6	1
2	3	0
	1	1

M					1	1	0	1
Q					1	0	1	1
				1	1	1	0	1
			1	1	1	0	1	
		1	0	0	0	0		
		1	1	0	1			
P	1	0	0	0	1	1	1	1

RAMAIAH Institute of Technology

13 (Multiplicand M) 1101	*	11 (Multiplier Q) =143 1011

M					**1**	**1**	**0**	**1**
Q					1	0	1	1
				1	1	1	0	1
			1	1	1	0	1	
		1	0	0	0	0		
		1	1	0	1			
P	1	0	0	0	1	1	1	1

2	13	
2	6	1
2	3	0
	1	1

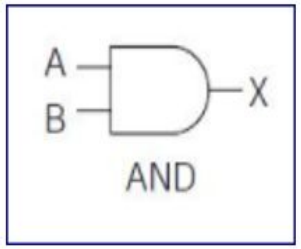
P	1	0	0	0	1	1	1	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128				8	4	2	1
128+8+4+2+1 = 143								

Implementation of Multiplication

1. Array multiplier
2. Sequential multiplier
3. Booth multiplier

Multiplication of Positive Numbers

Simple algorithm

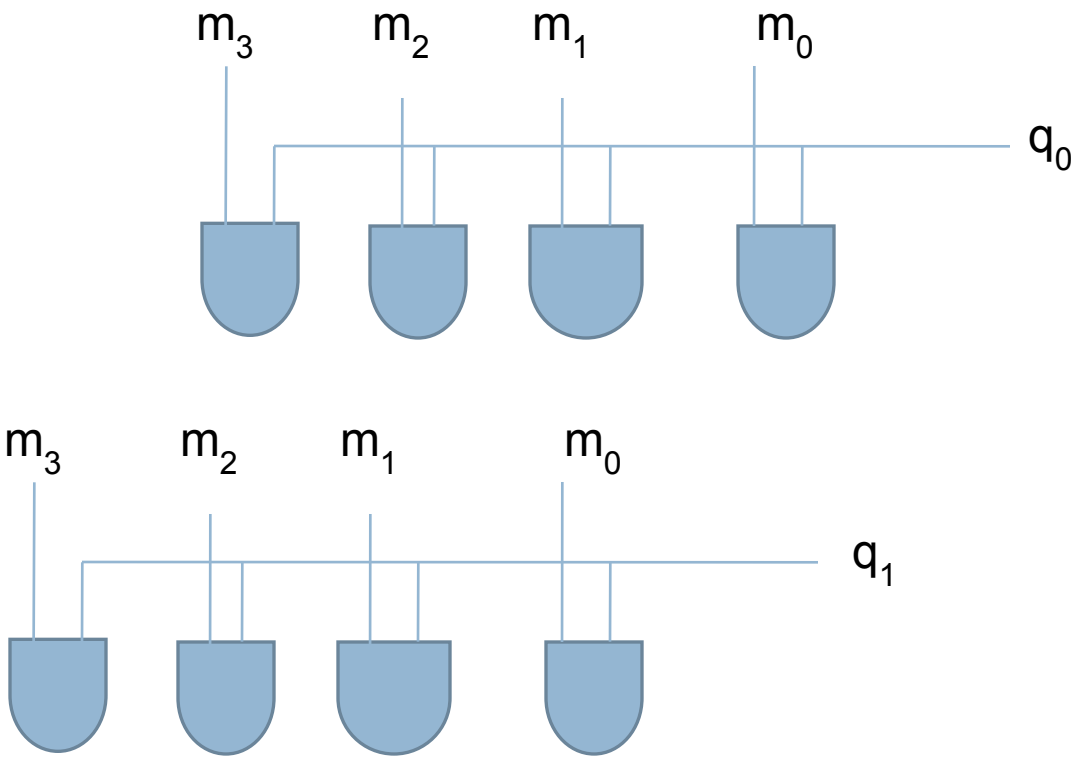


AND gate

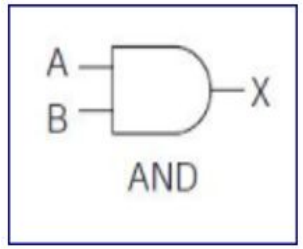
Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

```
                1  1  0  1        (13) Multiplicand M
             ×  1  0  1  1        (11) Multiplier Q
          ---------------
                1  1  0  1
             1  1  0  1
          0  0  0  0
       1  1  0  1
    ---------------------------
    1  0  0  0  1  1  1  1        (143) Product P
```

Multiplication of Positive Numbers



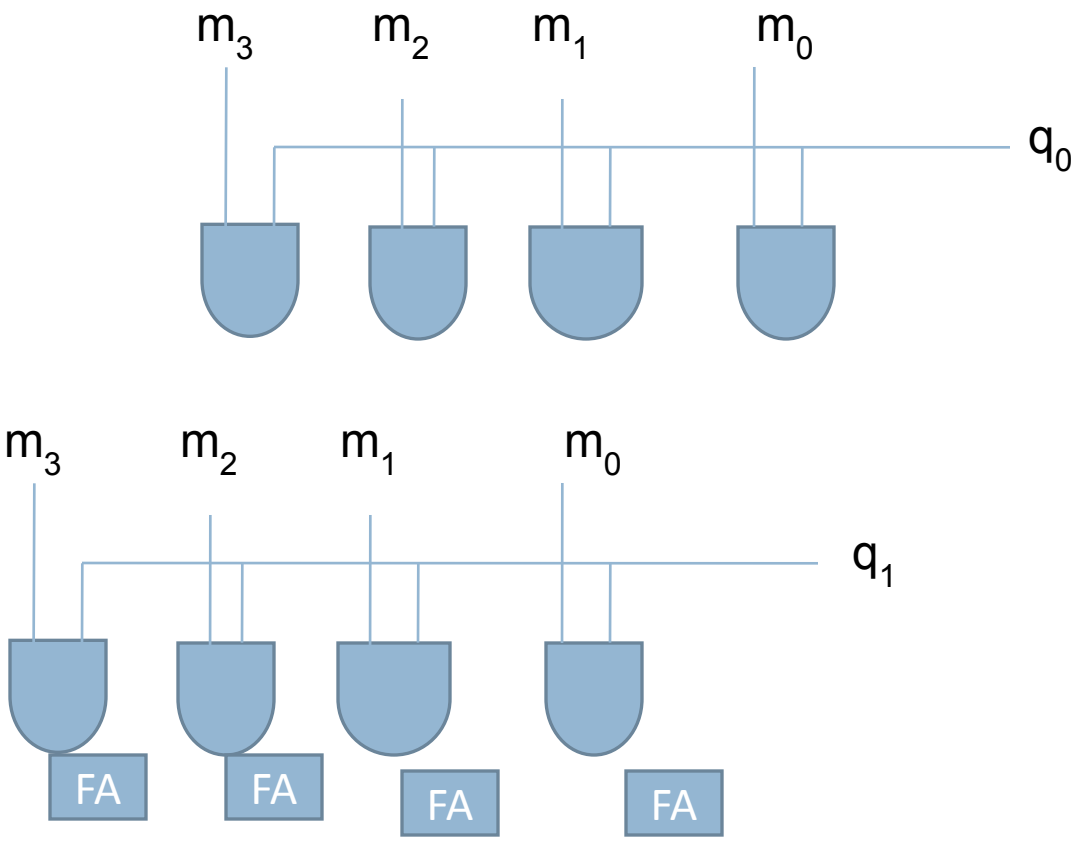
			1	1	0	1		(13) Multiplicand M
			1	0	1	1		(11) Multiplier Q
			1	1	0	1		
		1	1	0	1			
	0	0	0	0				
1	1	0	1					
1	0	0	0	1	1	1	1	(143) Product P



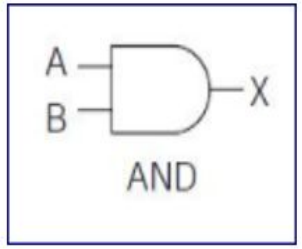
AND gate

Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

Multiplication of Positive Numbers




```
        1  1  0  1      (13) Multiplicand M
        1  0  1  1      (11) Multiplier Q
     ---------------
        1  1  0  1
     1  1  0  1
  ---------------------
  1  0  0  1  1  1
```



AND gate

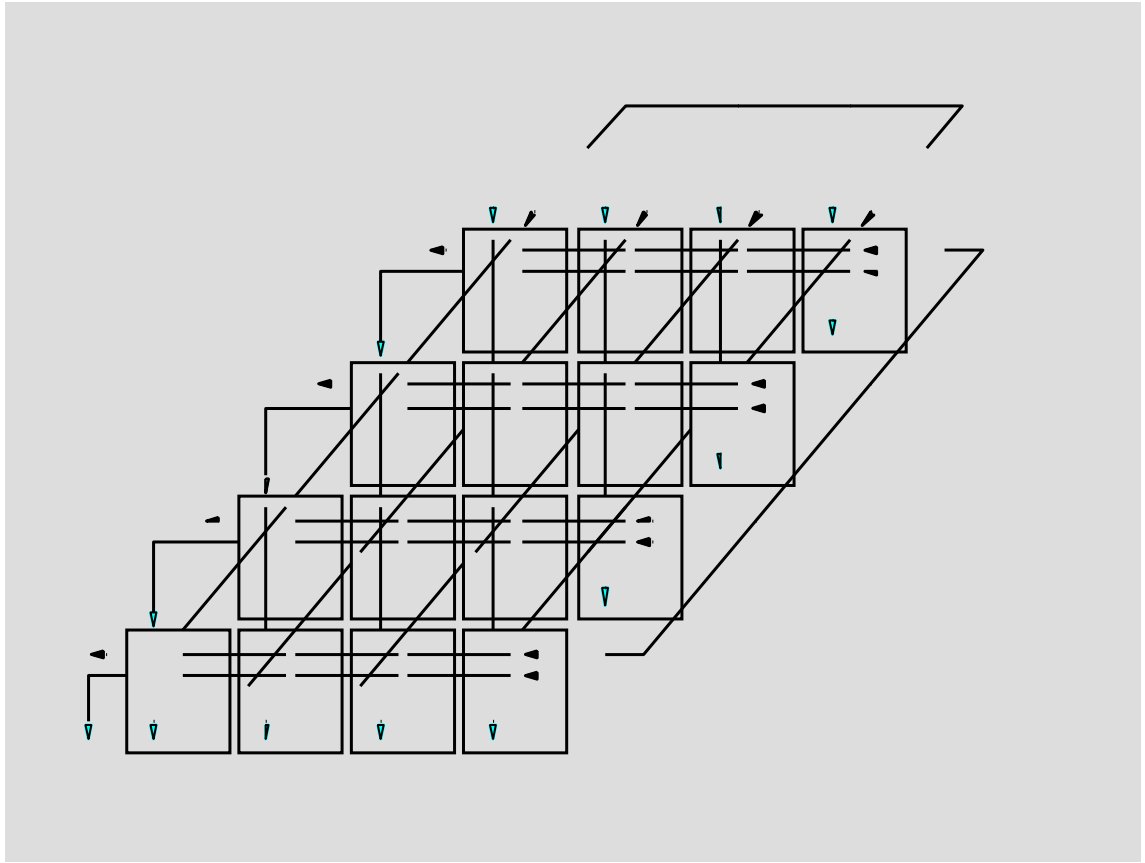
Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

RAMAIAH
Institute of Technology

Multiplication of Positive Numbers

Binary multiplication of positive operands can be implemented in a combinational 2D logic array

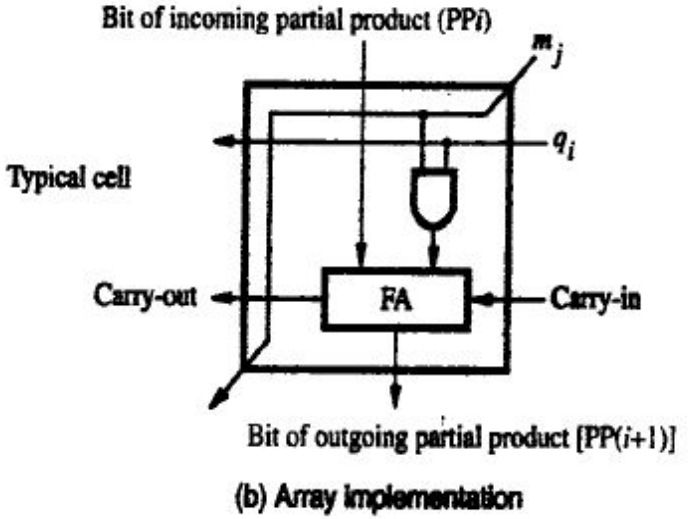
```
                1  1  0  1      (13) Multiplicand M
                1  0  1  1      (11) Multiplier Q
             ---------------
                1  1  0  1
             1  1  0  1
          0  0  0  0
       1  1  0  1
    ---------------------------
    1  0  0  0  1  1  1  1      (143) Product P
```



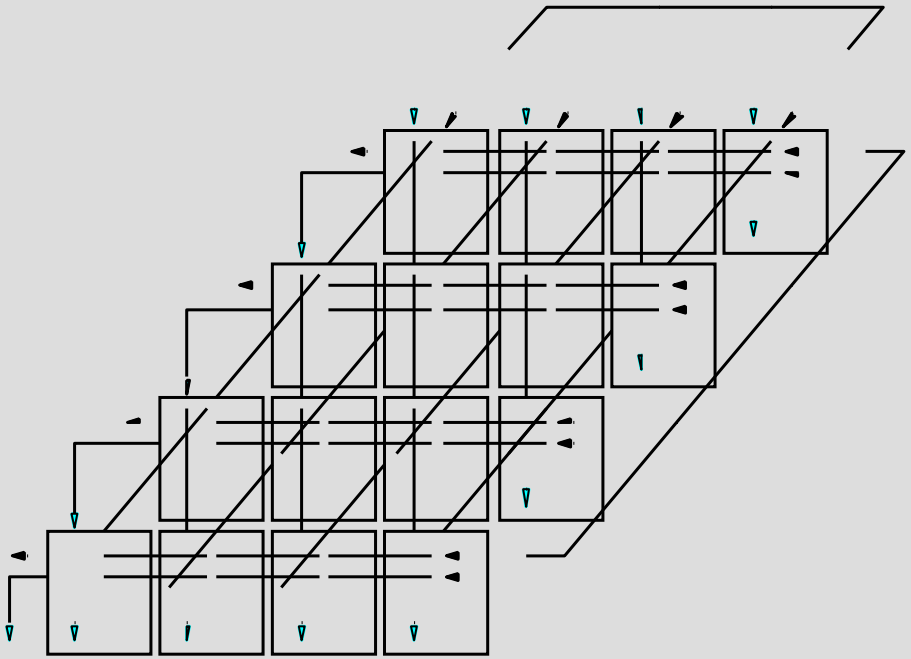
RAMAIAH
Institute of Technology

Implementation of Multiplication

The main component in each cell is a full adder

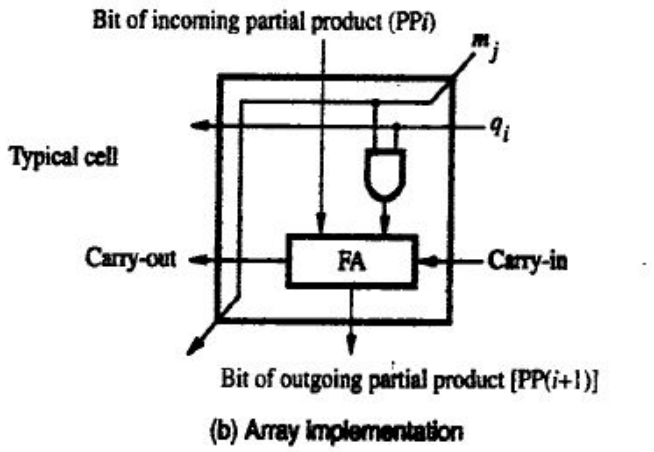


(b) Array implementation

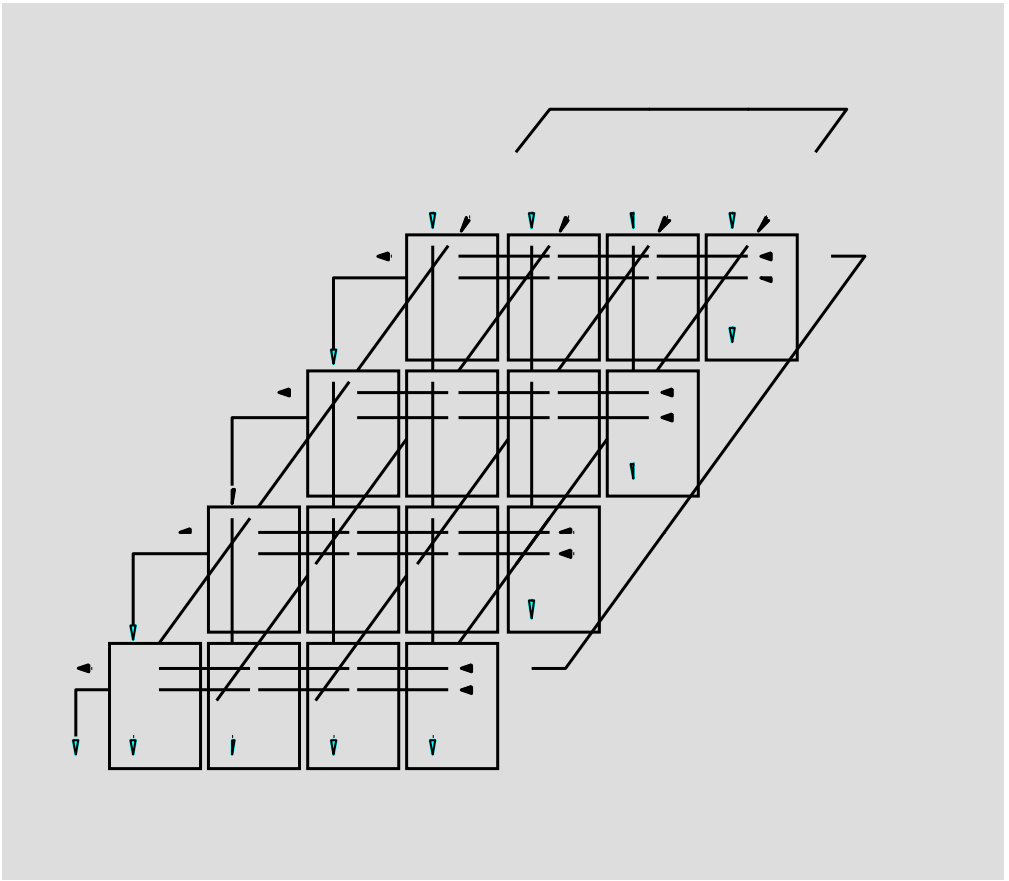


Implementation of Multiplication

The AND gate in each cell determines whether a multiplicand bit m_j, is added to the incoming partial- product bit, based on the value of the multiplier bit q_i



(b) Array implementation



Combinatorial array multiplier

Combinatorial array multipliers are:

- Extremely inefficient.
- Have a high gate count for multiplying numbers of practical size such as 32-bit or 64-bit numbers.
- Perform only one function, namely, unsigned integer product.

Improve gate efficiency by using a mixture of combinatorial array techniques and sequential techniques requiring less combinational logic.

Sequential multiplication

Recall the rule for generating partial products:

- If the ith bit of the multiplier is 1, add the appropriately shifted multiplicand to the current partial product.
- Multiplicand has been shifted left when added to the partial product.

However, adding a left-shifted multiplicand to an unshifted partial product is equivalent to adding an unshifted multiplicand to a right-shifted partial product.

Sequential multiplication procedure

1. Multiplicand(M) and Multiplier(Q) bits are loaded into register, C is initially set to 0, Register A is initially set to 0
2. 0^{th} Bit of Multiplier is checked
3. If 0^{th} bit = 1
   - Multiplicand and partial product is added
   - All bits of Q, A and C registers are shifted to right one bit
4. If 0^{th} bit = 0
   - No addition is performed, only shift operation is carried out

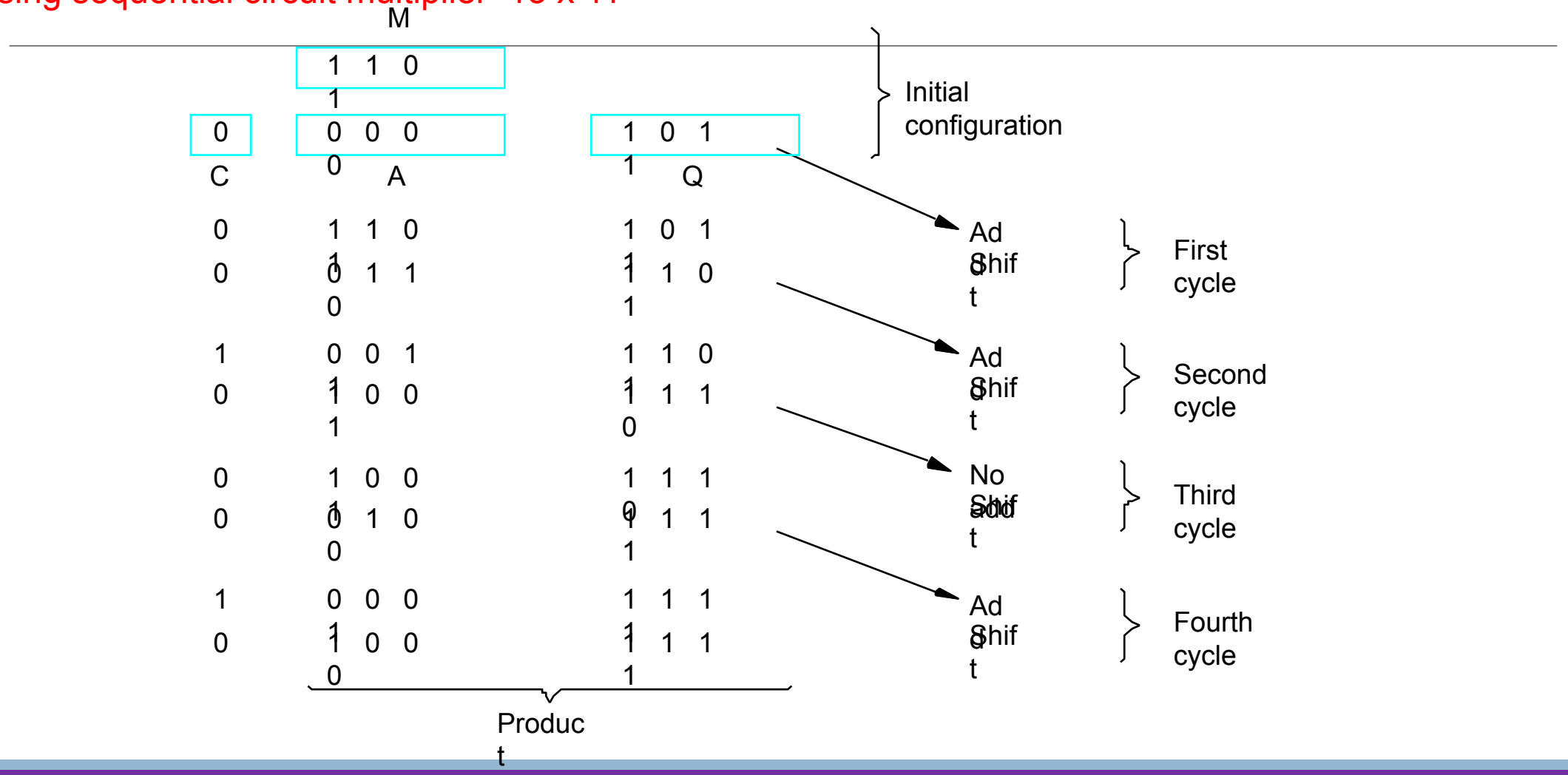
Step 2, 3 and 4 are repeated n times to get desired result in A and Q registers

RAMAIAH Institute of Technology

Sequential multiplication

Multiply using sequential circuit multiplier 13 x 11

				1	1	0	1	(13) Multiplicand M
				1	0	1	1	(11) Multiplier Q
1	0	0	0	1	1	1	1	(143) Product P



1 1 0 1 (13) Multiplicand M

1 0 1 1 (11) Multiplier Q

M | 1 | 1 | 0 | 1 |

1 0 0 0 1 1 1 1 (143) Product P

C	A				Q				
0	0	0	0	0	1	0	1	1	

1 1 0 1 (13) Multiplicand M

1 0 1 1 (11) Multiplier Q

1 0 0 0 1 1 1 1 (143) Product P

M | 1 | 1 | 0 | 1 |

C	A	Q	
0	0 0 0 0	1 0 1 1	
	1 1 0 1		ADD

	1	1	0	1	(13) Multiplicand M
	1	0	1	1	(11) Multiplier Q
1	0	0	0	1	1	1	1	(143) Product P

M

1	1	0	1

C	A				Q				
0	0	0	0	0	1	0	1	1	
0	1	1	0	1	1	0	1	1	ADD

	1	1	0	1	(13) Multiplicand M
	1	0	1	1	(11) Multiplier Q
1	0	0	0	1	1	1	1	(143) Product P

M

1	1	0	1

C	A				Q				
0	0	0	0	0	1	0	1	1	
0	1	1	0	1	1	0	1	1	ADD
								1	SHIFT

1 1 0 1 (13) Multiplicand M

1 0 1 1 (11) Multiplier Q

1 0 0 0 1 1 1 1 (143) Product P

M | 1 | 1 | 0 | 1 |

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
		0 1	SHIFT

	1	1	0	1 (13) Multiplicand M
	1	0	1	1 (11) Multiplier Q
1	0	0	0	1 1 1 1 (143) Product P

M

1	1	0	1

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
		1 0 1	SHIFT

	1	1	0	1 (13) Multiplicand M
	1	0	1	1 (11) Multiplier Q
1	0	0	0	1 1 1 1 (143) Product P

M

1	1	0	1

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
		1 1 0 1	SHIFT

	1	1	0	1 (13) Multiplicand M
	1	0	1	1 (11) Multiplier Q
1	0	0	0	1 1 1 1 (143) Product P

M

1	1	0	1

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
	0	1 1 0 1	SHIFT

	1	1	0	1 (13) Multiplicand M
	1	0	1	1 (11) Multiplier Q
1	0	0	0	1 1 1 1 (143) Product P

M

1	1	0	1

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
	1 0	1 1 0 1	SHIFT

1 1 0 1 (13) Multiplicand M

1 0 1 1 (11) Multiplier Q

1 0 0 0 1 1 1 1 (143) Product P

M | 1 | 1 | 0 | 1 |

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
	1 1 0	1 1 0 1	SHIFT

		1	1	0	1	(13) Multiplicand M
		1	0	1	1	(11) Multiplier Q
1	0	0	0	1	1	1	1	(143) Product P

M

1	1	0	1

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
	0 1 1 0	1 1 0 1	SHIFT

	1	1	0	1 (13) Multiplicand M
	1	0	1	1 (11) Multiplier Q
1	0	0	0	1 1 1 1 (143) Product P

M

1	1	0	1

C	A				Q				
0	0	0	0	0	1	0	1	1	
0	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**ADD**
0	0	1	1	0	1	1	0	1	SHIFT

RAMAIAH
Institute of Technology

		1	1	0	1	(13) Multiplicand M
		1	0	1	1	(11) Multiplier Q
1	0	0	0	1	1	1	1	(143) Product P

M

1	1	0	1

C	A	Q	
0	0 0 0 0	1 0 1 1	
0	1 1 0 1	1 0 1 1	ADD
0	0 1 1 0	1 1 0 1	SHIFT
1	0 0 1 1	1 1 0 1	ADD
0	1 0 0 1	1 1 1 0	SHIFT
0	0 1 0 0	1 1 1 0	NO ADD
1	0 0 0 1	1 1 1 1	SHIFT
0	1 0 0 0	1 1 1 1	ADD
		1 1 1 1	SHIFT

Examples

Multiply following numbers using sequential circuit multiplier

1. 24 x 16
2. 45 x 12

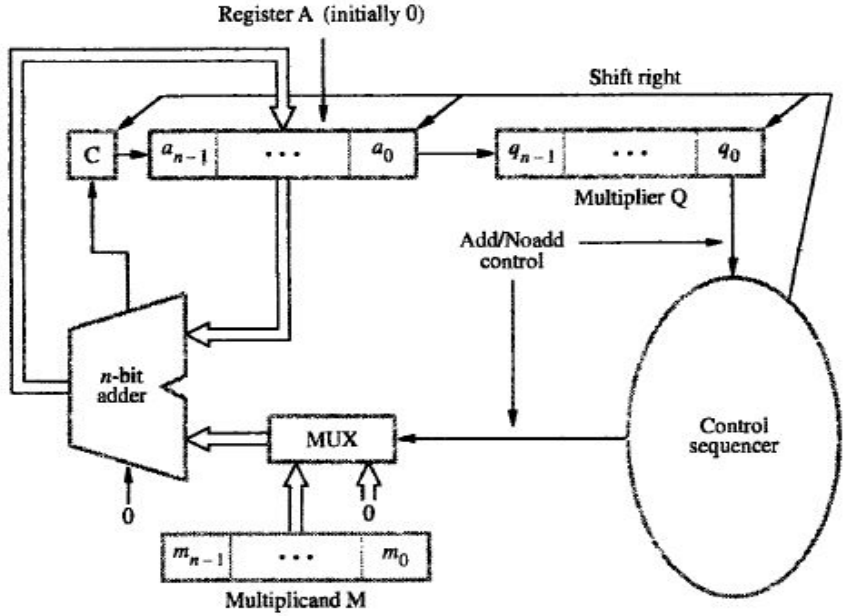
Illustrate a sequential circuit binary multiplier with a neat diagram.

Sequential Circuit Multiplier

Figure shows the Hardware arrangement for sequential multiplication

Procedure for multiplication:

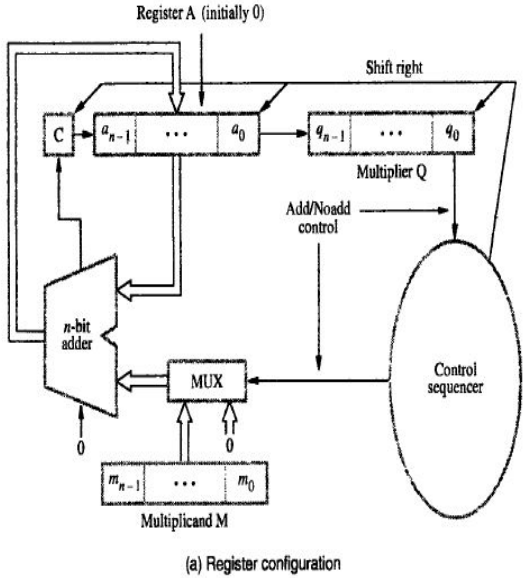
- Multiplier is loaded into register Q, Multiplicand is loaded into register M
- C & A are cleared to 0
- Registers A and Q combined hold PPi(partial product)
- Multiplier bit qi generates the signal Add/Noadd.
- Single n-bit adder is used n times



(a) Register configuration

Sequential Circuit Multiplier

- If q0=1, add M to A and store sum in A.
- The carry-out from the adder is stored in flip-flop C
- Then C, A and Q are shifted right one bit-position.
- If q0=0, no addition performed and C, A & Q are shifted right one bit-position.
- After n cycles, the high-order half of the product is held in register A and the low-order half is held in register Q.



(a) Register configuration

UNIT 2 – Arithmetic unit

Multiplication of two Numbers

A Signed Operand Multiplication
- Booth algorithm

Fast Multiplication
- Bit pair recoding
- Carry Save Addition(CSA)

Integer Division

IEEE standard for floating point numbers

Signed Multiplication

Multiplication of 2's complement signed operands, generating a double length product

- Case1 :

  Positive Multiplier and Negative multiplicand, When we add a negative multiplicand to a partial product , we must extend sign-bit value of the multiplicand to the left as far as the product will extend.

- Case2:

  For a negative multiplier, a straightforward solution is to form the 2's-complement of both the multiplier and the multiplicand and proceed as in the case of a positive multiplier.

A technique that works equally well for both negative and positive multipliers – Booth algorithm.

Signed Multiplication

Considering 2's-complement signed operands, what will happen to (-13)×(+11) if following the same method of unsigned multiplication?

					1	0	0	1	1	(-13)
					0	1	0	1	1	(+11)
1	1	1	1	1	1	0	0	1	1	
1	1	1	1	1	0	0	1	1		
0	0	0	0	0	0	0	0			
1	1	1	0	0	1	1				
0	0	0	0	0	0					
1	1	0	1	1	1	0	0	0	1	(-143)

Sign extension is shown in blue

Sign extension of negative multiplicand.

Booth Algorithm

Booths recoding scheme

Multiplier is scanned from right to left

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

Booth multiplier recoding table.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Booth Algorithm

Recode the multiplier 101100

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

Booth multiplier recoding table.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Booth Algorithm

1. Compute +13 x -6 using Booth algorithm

Booth multiplication with a negative multiplier.

```
    0  1  1  0  1   (+13)
X   1  1  0  1  0   (-6)
```

```
                    0  1  1  0  1
                    0 -1 + -1  0
                          1
0  0  0  0  0  0  0  0  0  0
1  1  1  1  1  0  0  1  1
0  0  0  0  1  1  0  1
1  1  1  0  0  1  1
0  0  0  0  0  0
-------------------------------
1  1  1  0  1  1  0  0  1  0   (-78)
```

Booth multiplier recoding table.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

RAMAIAH Institute of Technology

Booth Algorithm

2	13		2	6		
2	6	1	2	3	0	
2	3	0		1	1	
	1	1				

Booth multiplier recoding table.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

1
Represent in Binary form

M= +13			1	1	0	1
Q= -6			0	1	1	0

2
Take 2's complement of Negative value

M= +13			1	1	0	1
Q= -6			1	0	1	0

3
Add Sign bit

M= +13		0	1	1	0	1
Q= -6		1	1	0	1	0

-6 = 2's complement

0 1 1 0
10 0 1
1

1 0 1 0

4
Recode Multiplier

Q= -6	1	1	0	1	0	0
	0	-1	+1	-1	0	

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
							0	0	0	0	0
P											

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
P											

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
						1	0	0	1	1	
P											

+13 = 2's complement

0 1 1 0 1
1 0 0 1 0
1

1 0 0 1 1

RAMAIAH
Institute of Technology

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
P											

+13 = 2's complement

0 1 1 0 1
1 0 0 1 0
1

1 0 0 1 1

RAMAIAH
Institute of Technology

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
					0	1	1	0	1		
P											

RAMAIAH
Institute of Technology

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
		0	0	0	0	1	1	0	1		
P											

RAMAIAH
Institute of Technology

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
		0	0	0	0	1	1	0	1		
				1	0	0	1	1			
P											

+13 = 2's complement

0 1 1 0 1
1 0 0 1 0
1

1 0 0 1 1

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
		0	0	0	0	1	1	0	1		
		1	1	1	0	0	1	1			
P											

+13 = 2's complement

0 1 1 0 1
1 0 0 1 0
1

1 0 0 1 1

RAMAIAH
Institute of Technology

Booth Algorithm

M							0	1	1	0	1
Q							0	-1	1	-1	0
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
		0	0	0	0	1	1	0	1		
		1	1	1	0	0	1	1			
			0	0	0	0	0				
P											

RAMAIAH
Institute of Technology

Booth Algorithm

M							**0**	**1**	**1**	**0**	**1**
Q							**0**	**-1**	**1**	**-1**	**0**
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
		0	0	0	0	1	1	0	1		
		1	1	1	0	0	1	1			
		0	0	0	0	0	0				
P											

RAMAIAH Institute of Technology

Booth Algorithm

M							**0**	**1**	**1**	**0**	**1**
Q							**0**	**-1**	**1**	**-1**	**0**
		1	1	1	1	1	1	1			
		0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	0	0	1	1	
		0	0	0	0	1	1	0	1		
		1	1	1	0	0	1	1			
		0	0	0	0	0	0				
P	1	1	1	1	0	1	1	0	0	1	0

Carry ignore

1 + 1	**=**	**10**
1+1+1	**=**	**11**
1+1+1+1	**=**	**100**
1+1+1+1+1	**=**	**101**
1+1+1+1+1+1	**=**	**110**

RAMAIAH Institute of Technology

Booth Algorithm

2. Multiply each of the following pairs of signed 2's complement numbers using booth algorithm (Assume A is the Multiplicand and B is the Multiplier)
A = 010111 B = 110110

Q=	**1**	**1**	**0**	**1**	**1**	**0**	0
	0	-1	+1	0	-1	0	

Booth multiplier recoding table.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

A								**0**	**1**	**0**	**1**	**1**	**1**
B								**0**	**-1**	**+1**	**0**	**-1**	**0**
		0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	1	0	1	0	0	1	
		0	0	0	0	0	0	0	0	0	0		
		0	0	0	0	**1**	**0**	**1**	**1**	**1**			
		1	1	1	0	1	0	0	1				
		0	0	0	0	0	0	0					
P													

RAMAIAH Institute of Technology

Booth Algorithm

1 + 1	=	10
1+1+1	=	11
1+1+1+1	=	100
1+1+1+1+1	=	101
1+1+1+1+1+1	=	110

A								**0**	**1**	**0**	**1**	**1**	**1**
B								**0**	**-1**	**+1**	**0**	**-1**	**0**
		1	1	1	0	1	1	1					
		0	0	0	0	0	0	0	0	0	0	0	0
		1	1	1	1	1	1	0	1	0	0	1	
		0	0	0	0	0	0	0	0	0	0		
		0	0	0	0	1	0	1	1	1			
		1	1	1	0	1	0	0	1				
		0	0	0	0	0	0	0					
P	1	1	1	1	1	0	0	0	1	1	0	1	0

Carry ignore

Examples for Booth Algorithm

1. Compute 45 x -24 using Booth algorithm
2. Compute -13 x -20 using Booth algorithm
3. Computer -13 x +9 using Booth algorithm
4. Computer 14 x -8 using Booth algorithm
5. Multiply each of the following pairs of signed 2's complement numbers using booth algorithm (Assume A is the Multiplicand and B is the Multiplier)
   - A = 110011 B = 101100
   - A = 110101 B = 011011
   - A = 001111 B = 001111
   - A = 10101 B = 10101

UNIT 2 – Arithmetic unit

Multiplication of two Numbers

A Signed Operand Multiplication

- Booth algorithm

Fast Multiplication

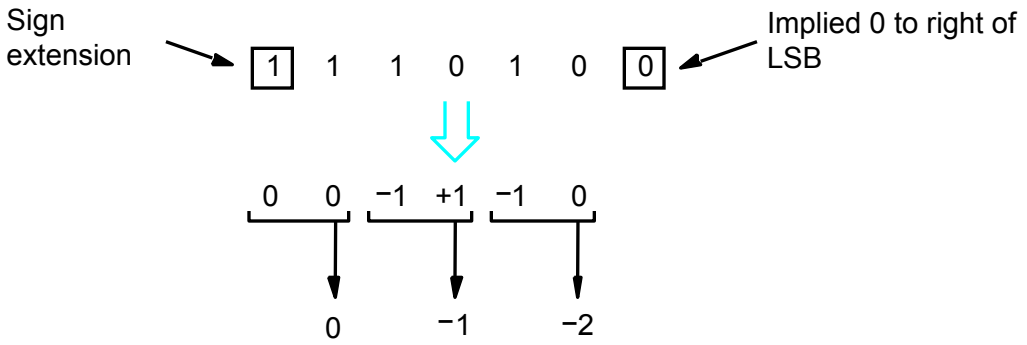
- Bit-Pair algorithm
- Carry Save Addition(CSA)

Integer Division

IEEE standard for floating point numbers

Bit-Pair algorithm

Bit-pair recoding halves the maximum number of summands (versions of the multiplicand).



(a) Example of bit-pair recoding derived from Booth recoding

Table of multiplicand selection decisions

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	-2×M
1	0	1	-1×M
1	1	0	-1×M
1	1	1	0×M

RAMAIAH
Institute of Technology

Bit-Pair algorithm

Recode the multiplier 101100

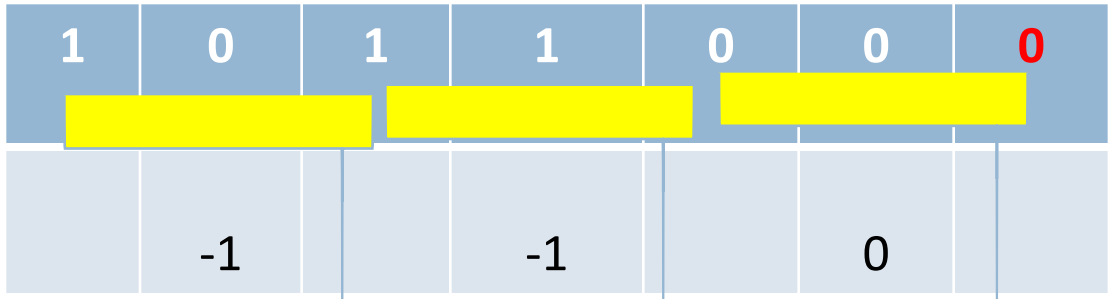


Table of multiplicand selection decisions

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	-2×M
1	0	1	-1×M
1	1	0	-1×M
1	1	1	0×M

RAMAIAH
Institute of Technology

Bit-Pair algorithm

Recode the multiplier 011011

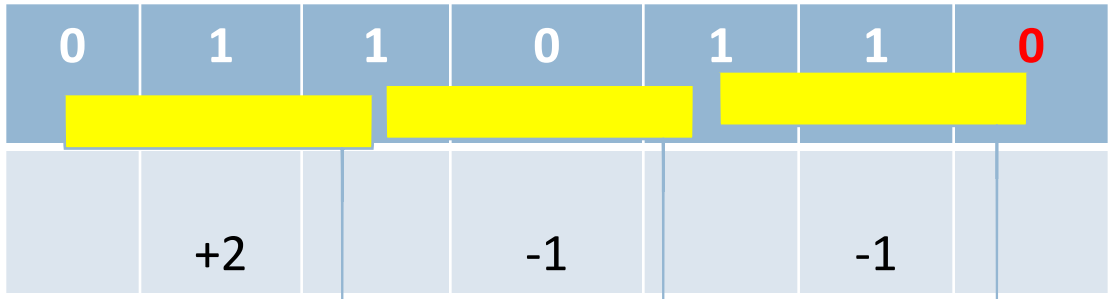


Table of multiplicand selection decisions

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	-2×M
1	0	1	-1×M
1	1	0	-1×M
1	1	1	0×M

RAMAIAH Institute of Technology

Bit-Pair algorithm

Recode the multiplier 1 1 0 1 0

Add sign extended bit

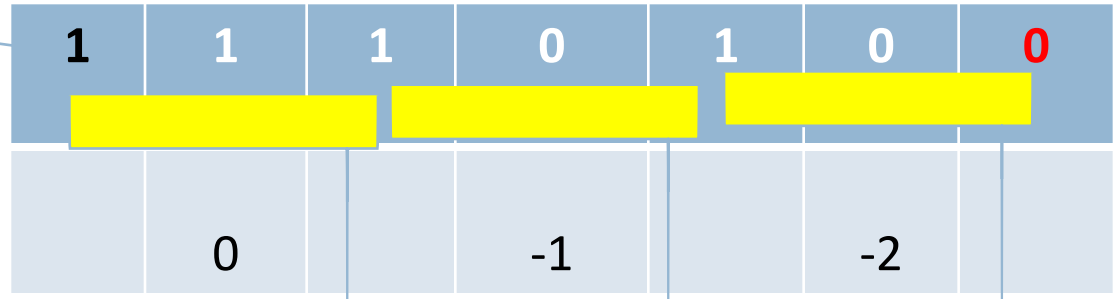
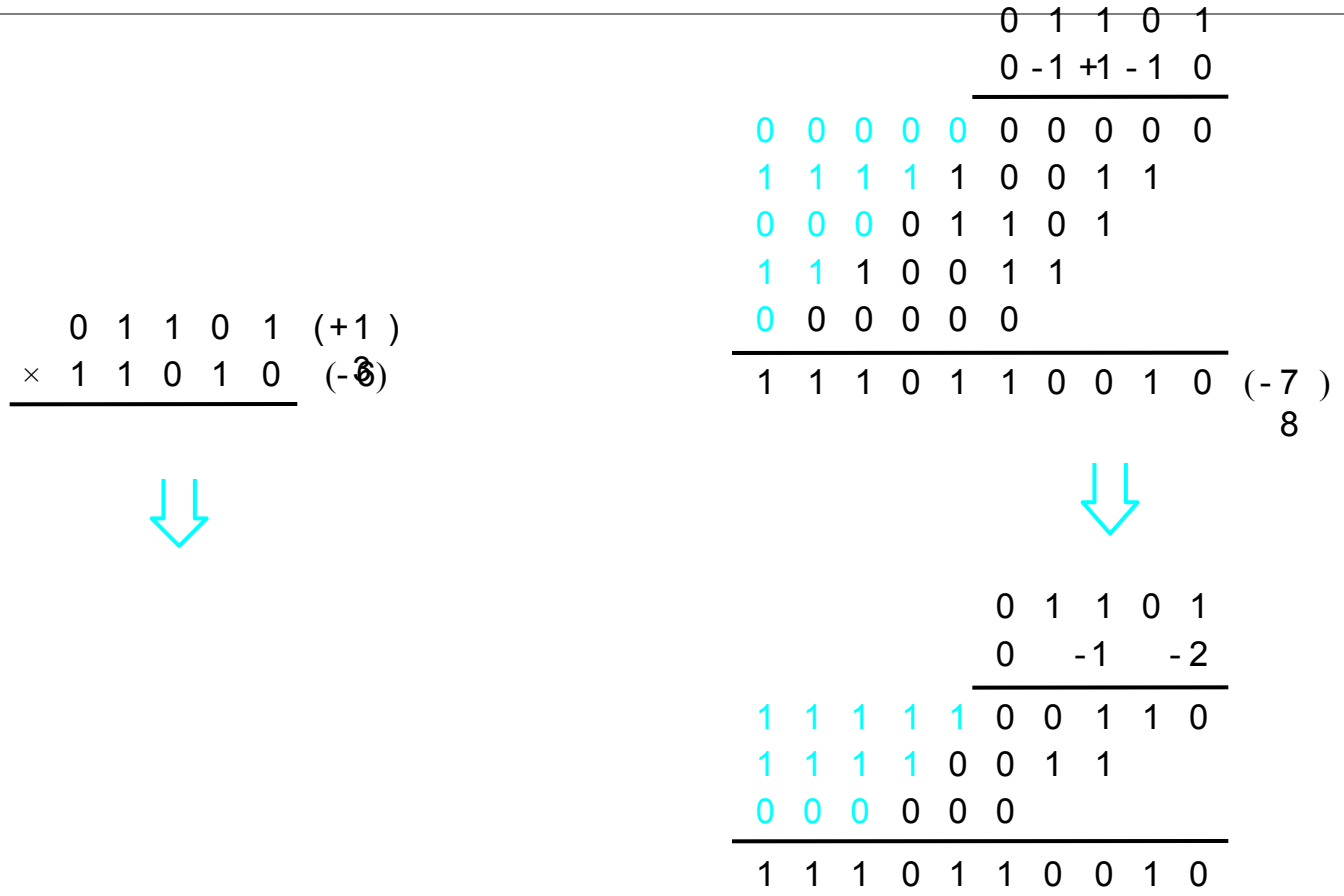


Table of multiplicand selection decisions

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	−2×M
1	0	1	−1×M
1	1	0	−1×M
1	1	1	0×M

RAMAIAH Institute of Technology

Bit-Pair algorithm



RAMAIAH Institute of Technology

Bit-Pair algorithm

2	13		2	6		
2	6	1	2	3	0	
2	3	0		1	1	
	1	1				

Table of multiplicand selection decisions

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0×M
0	0	1	+1×M
0	1	0	+1×M
0	1	1	+2×M
1	0	0	-2×M
1	0	1	-1×M
1	1	0	-1×M
1	1	1	0×M

1. Compute +13 x -6 using Bit Pair Recoding technique.

1 Represent in Binary form

M= +13			1	1	0	1
Q= -6			0	1	1	0

2 Take 2's complement of Negative value

M= +13			1	1	0	1
Q= -6			1	0	1	0

3 Add Sign bit

M= +13		0	1	1	0	1
Q= -6		1	1	0	1	0

-6 = 2's complement

0 1 1 0
10 0 1
1

1 0 1 0

4 Recode Multiplier

1	1	1	0	1	0	0
	0		-1		-2	

Bit-Pair algorithm

If Multiplier bit is -2, take 2's complement of Multiplicand and then multiply with 10

M= 0 1 1 0 1
1 0 0 1 0
1

1 0 0 1 1 X 1 0

0 0 0 0 0
1 0 0 1 1

1 0 0 1 1 0

M								0	1	1	0	1
Q								0		-1		-2
			1	1	1	1	1	0	0	1	1	0
P												

Bit-Pair algorithm

If Multiplier bit is -1, take 2's complement of Multiplicand

M= 0 1 1 0 1
1 0 0 1 0
1

1 0 0 1 1

M								0	1	1	0	1
Q								0		-1		-2
			1	1	1	1	1	0	0	1	1	0
			1	1	1	1	0	0	1	1		
P												

RAMAIAH
Institute of Technology

Bit-Pair algorithm

If Multiplier bit is 0, then 0 X M

M								0	1	1	0	1
Q								0		-1		-2
			1	1	1	1	1	0	0	1	1	0
			1	1	1	1	0	0	1	1		
			0	0	0	0	0	0				
P												

Bit-Pair algorithm

If Multiplier bit is 0, then 0 X M

M							0	1	1	0	1
Q							0		-1		-2
	1	1	1	1			1	1			
		1	1	1	1	1	0	0	1	1	0
		1	1	1	1	0	0	1	1		
		0	0	0	0	0	0				
P	1	1	1	1	0	1	1	0	0	1	0

Carry ignore

Bit-Pair algorithm

Compute -11 x +27 using Bit Pair Recoding technique.

1
Represent in Binary form

M= -11		0	1	0	1	1
Q= 27		1	1	0	1	1

2
Take 2's complement of Negative value

M= -11		1	0	1	0	1
Q= 27						

3
Add Sign bit

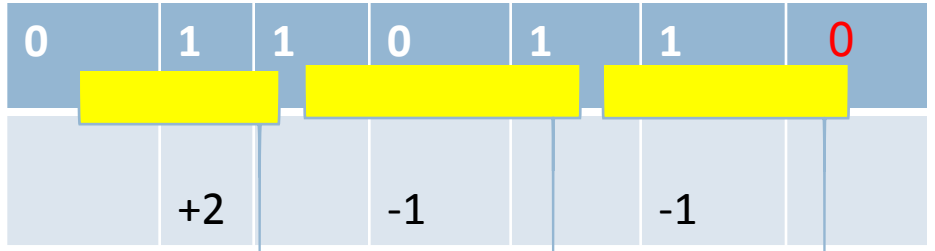
M= -11	1	1	0	1	0	1
Q= 27	0	1	1	0	1	1

-11 = 2's complement

0 1 0 1 1
1 0 1 0 0
1

1 0 1 0 1

4
Recode Multiplier



Bit-Pair algorithm

If Multiplier bit is -1, take 2's complement of Multiplicand

```
1 1 0 1 0 1
0 0 1 0 1 0
          1
-----------
0 0 1 0 1 1
```

M							1	1	0	1	0	1
Q								+2		-1		-1
							0	0	1	0	1	1
P												

Bit-Pair algorithm

If Multiplier bit is -1, take 2's complement of Multiplicand

```
1 1 0 1 0 1
0 0 1 0 1 0
          1
-----------
0 0 1 0 1 1
```

M							1	1	0	1	0	1
Q								+2		-1		-1
	0	0	0	0	0	0	0	0	1	0	1	1
P												

RAMAIAH
Institute of Technology

Bit-Pair algorithm

If Multiplier bit is -1, take 2's complement of Multiplicand

```
1 1 0 1 0 1
0 0 1 0 1 0
          1
-----------
0 0 1 0 1 1
```

M							1	1	0	1	0	1
Q								+2		-1		-1
	0	0	0	0	0	0	0	0	1	0	1	1
	0	0	0	0	0	0	1	0	1	1		
P												

Bit-Pair algorithm

If Multiplier bit is +2, multiply the multiplicand with 10

M= 1 1 0 1 0 1 X 1 0

```
      0 0 0 0 0 0
  1 1 0 1 0 1
-------------------
  1 1 0 1 0 1 0
```

M							1	1	0	1	0	1
Q								+2		-1		-1
	0	0	0	0	0	0	0	0	1	0	1	1
	0	0	0	0	0	0	1	0	1	1		
		1	1	0	1	0	1	0				
P												

Bit-Pair algorithm

If Multiplier bit is +2, multiply the multiplicand with 10

M= 1 1 0 1 0 1 X 1 0

```
      0 0 0 0 0 0
  1 1 0 1 0 1
-------------------
  1 1 0 1 0 1 0
```

M							1	1	0	1	0	1
Q								+2		-1		-1
	0	0	0	0	0	0	0	0	1	0	1	1
	0	0	0	0	0	0	1	0	1	1		
	1	1	1	0	1	0	1	0				
P												

Bit-Pair algorithm

If Multiplier bit is +2, multiply the multiplicand with 10

M= 1 1 0 1 0 1 X 1 0

0 0 0 0 0 0
1 1 0 1 0 1
1 1 0 1 0 1 0

M							1	1	0	1	0	1
Q								+2		-1		-1
						1		1				
	0	0	0	0	0	0	0	0	1	0	1	1
	0	0	0	0	0	0	1	0	1	1		
	1	1	1	0	1	0	1	0				
P	1	1	1	0	1	1	0	1	0	1	1	1

Examples for Bit-Pair Recoding Technique

1. Compute -87 X -35 using Bit Pair Recoding technique.
2. Write Booth multiplier recoding table and explain good, ordinary and worst-case multiplier with suitable examples.
3. Solve using Bit Pair Recoding technique

   i)-14 X -5

   ii) 12 X -4

   iii) 45 X -24

   iv) -32 X -15

4. Multiply each of the following pairs of signed 2's complement numbers using bit pairing of the multiplier (Assume A is the Multiplicand and B is the Multiplier)

   i) A=010111 B=110110

   ii) A=110011 B=101100.

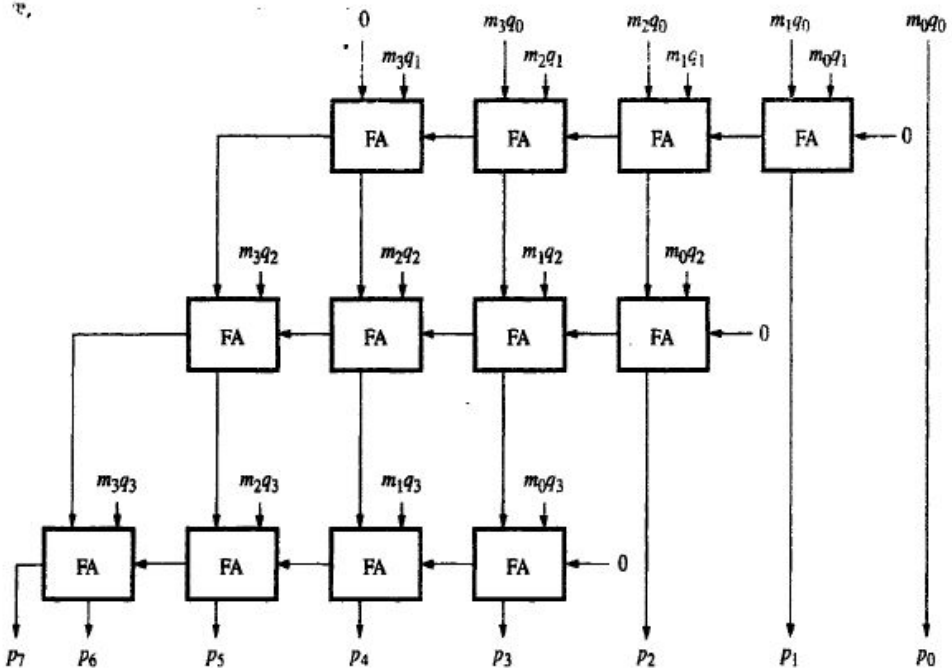
Carry-Save Addition(CSA) of Summands

Multiplication requires the addition of several summands

CSA speeds up the addition process

Consider the array for 4 x 4 multiplication

```
                1  1  0  1      (13) Multiplicand M
             ×  1  0  1  1      (11) Multiplier Q
          ---------------
                1  1  0  1
             1  1  0  1
          0  0  0  0
       1  1  0  1
    ---------------------
1  0  0  0  1  1  1  1      (143) Product P
```

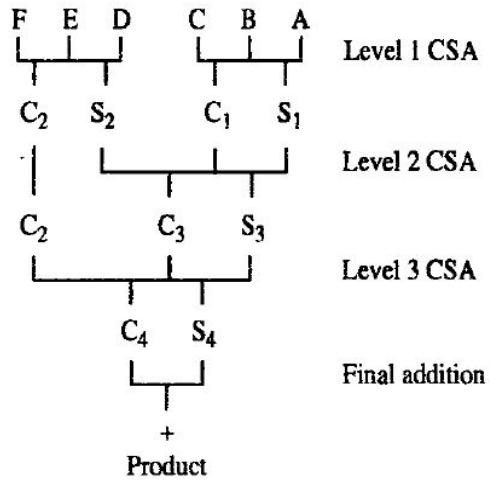


(a) Ripple-carry array

Carry-Save Addition of Summands

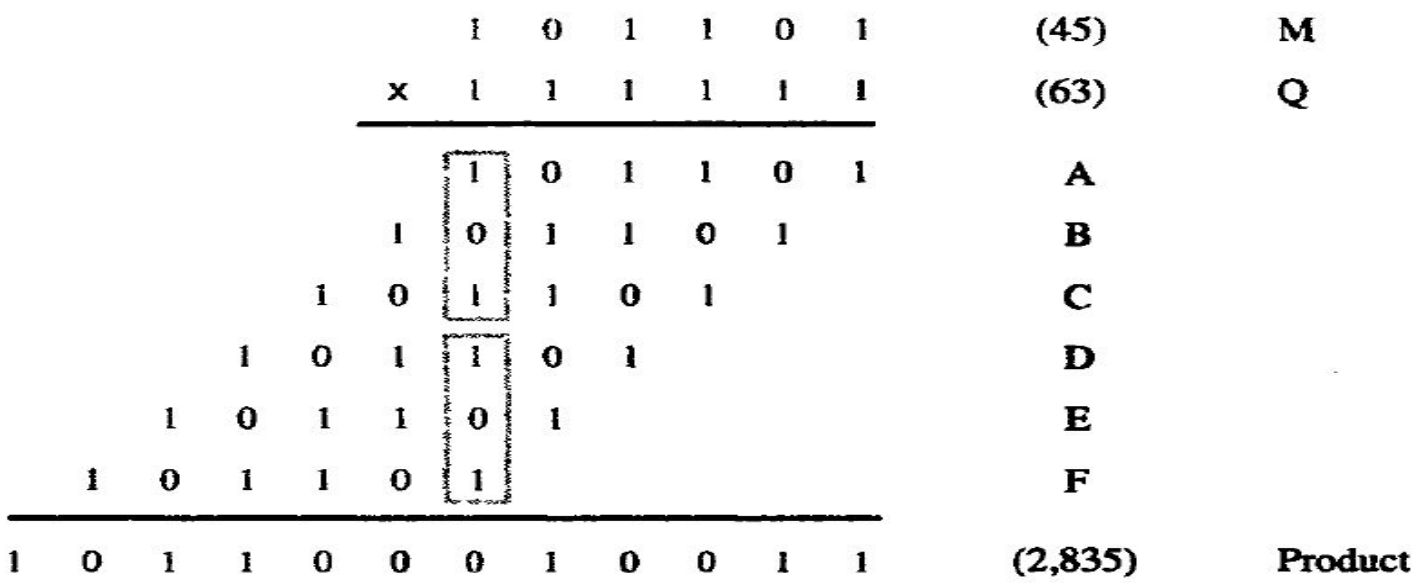
Consider the addition of many summands:

- Group the summands in threes and perform carry-save addition on each of these groups in parallel to generate a set of S and C vectors in one full-adder delay
- Group all of the S and C vectors into threes, and perform carry-save addition on them, generating a further set of S and C vectors in one more full-adder delay
- Continue with this process until there are only two vectors remaining
- They can be added in a RCA or CLA to produce the desired product



Schematic representation of the carry-save addition

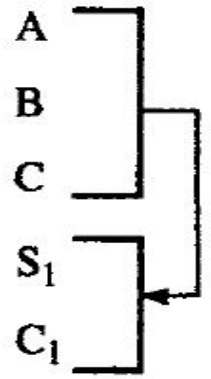
Carry-Save Addition(CSA) of Summands

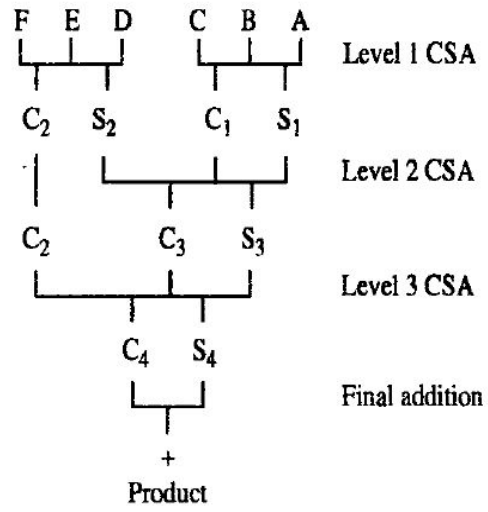


1 + 1	=	10
1+1+1	=	11
1+1+1+1	=	100
1+1+1+1+1	=	101
1+1+1+1+1+1	=	110

Carry-Save Addition(CSA) of Summands

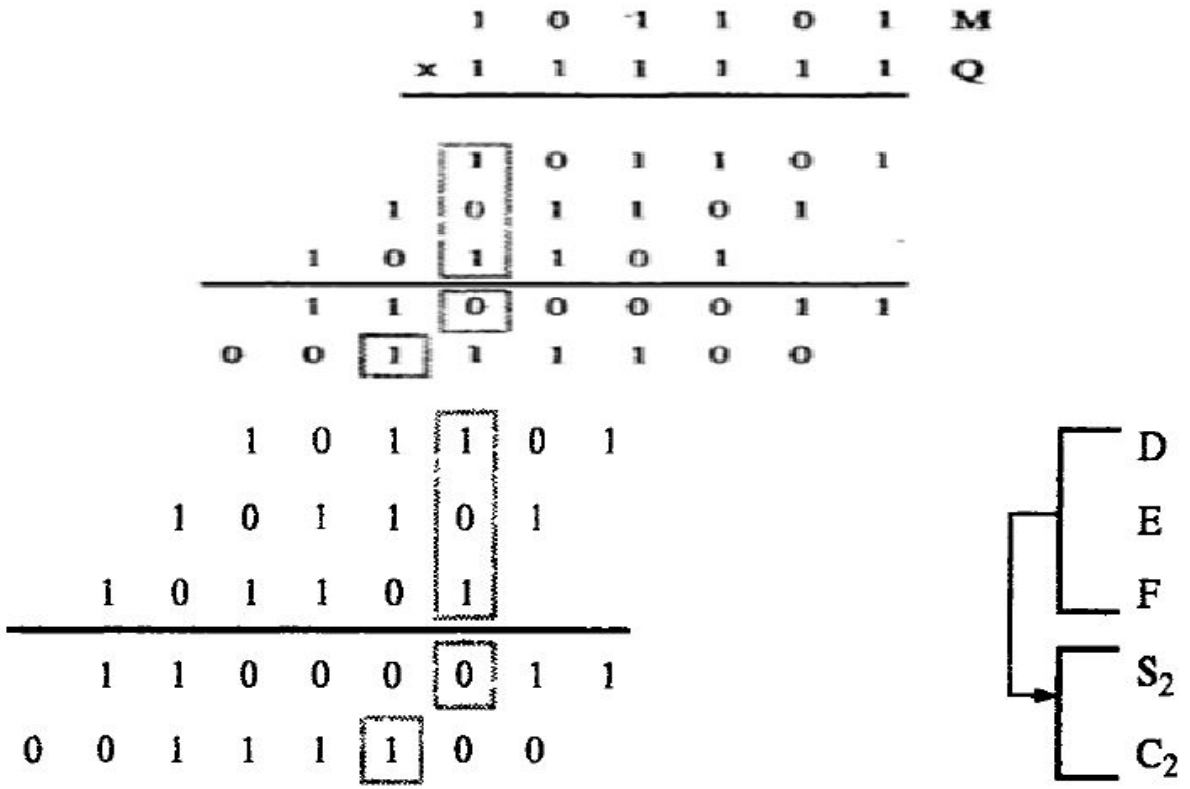
```
                1  0  1  1  0  1   M
              x 1  1  1  1  1  1   Q
            ---------------------
                1  0  1  1  0  1
             1  0  1  1  0  1
          1  0  1  1  0  1
       ---------------------------
       1  1  0  0  0  0  1  1
    0  0  1  1  1  1  0  0
```

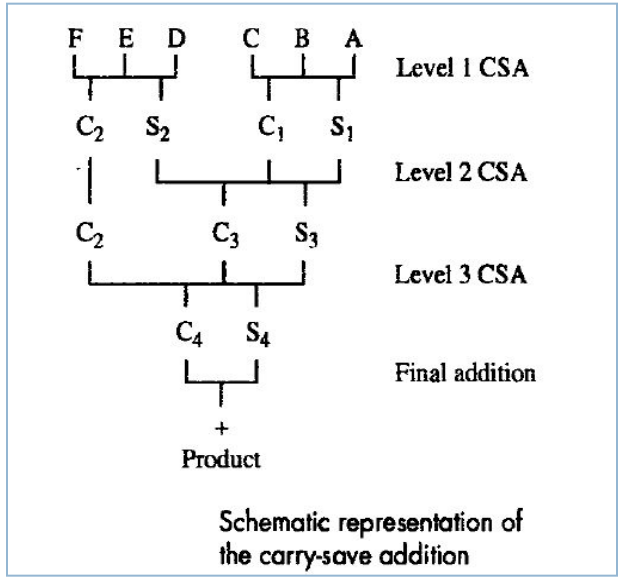




Schematic representation of the carry-save addition

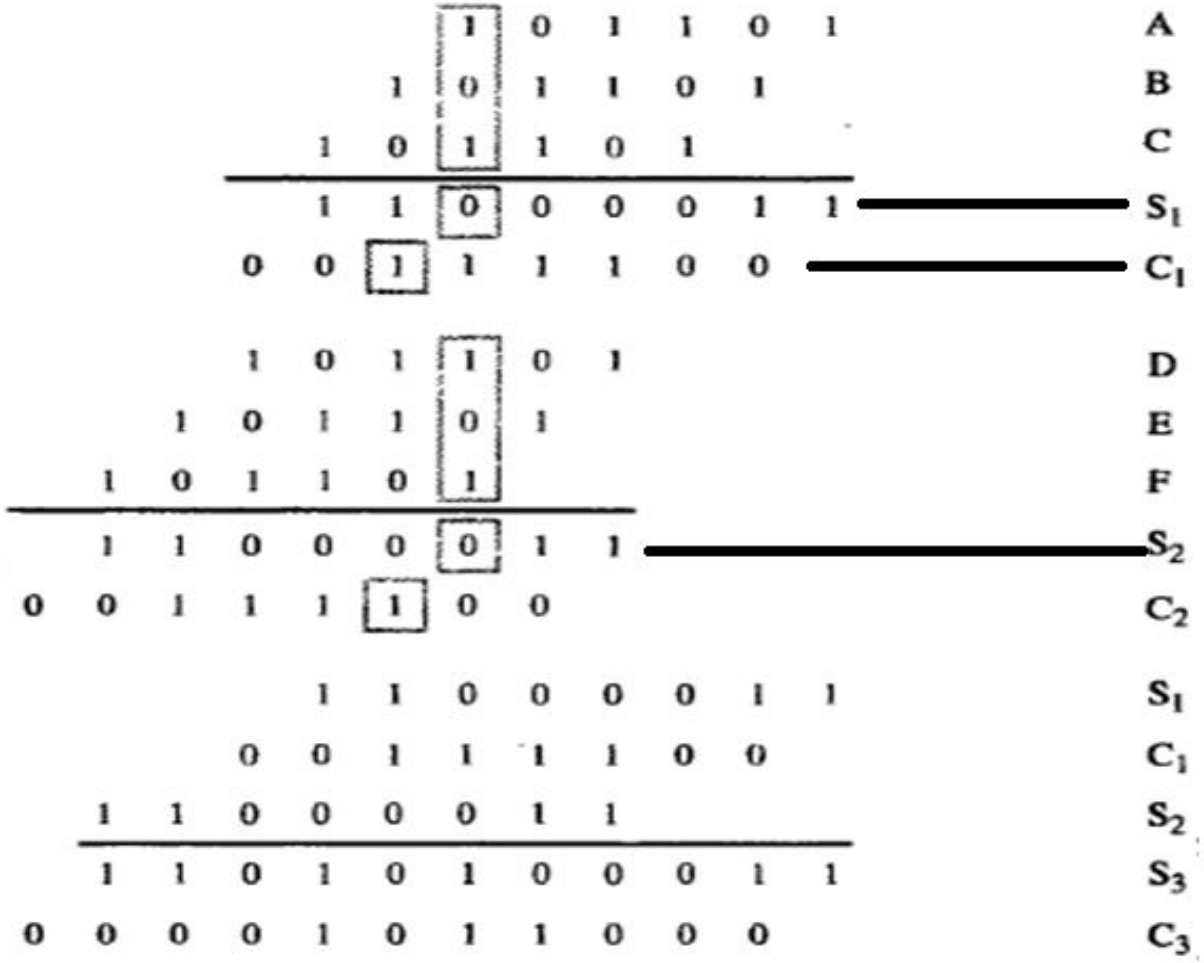
Carry-Save Addition(CSA) of Summands

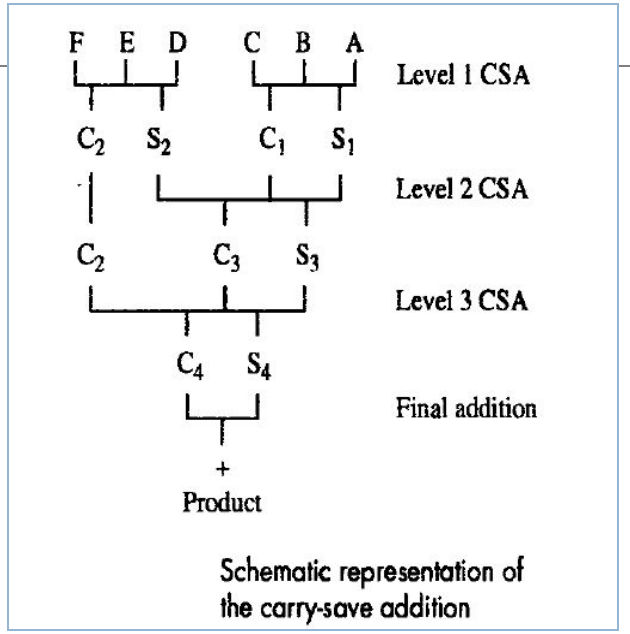




Schematic representation of the carry-save addition

Carry-Save Addition(CSA) of Summands

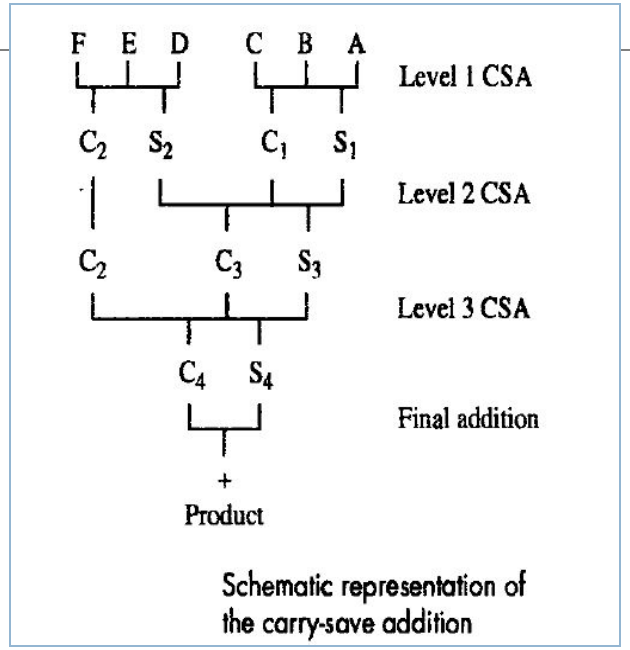




Schematic representation of the carry-save addition

Carry-Save Addition(CSA) of Summands

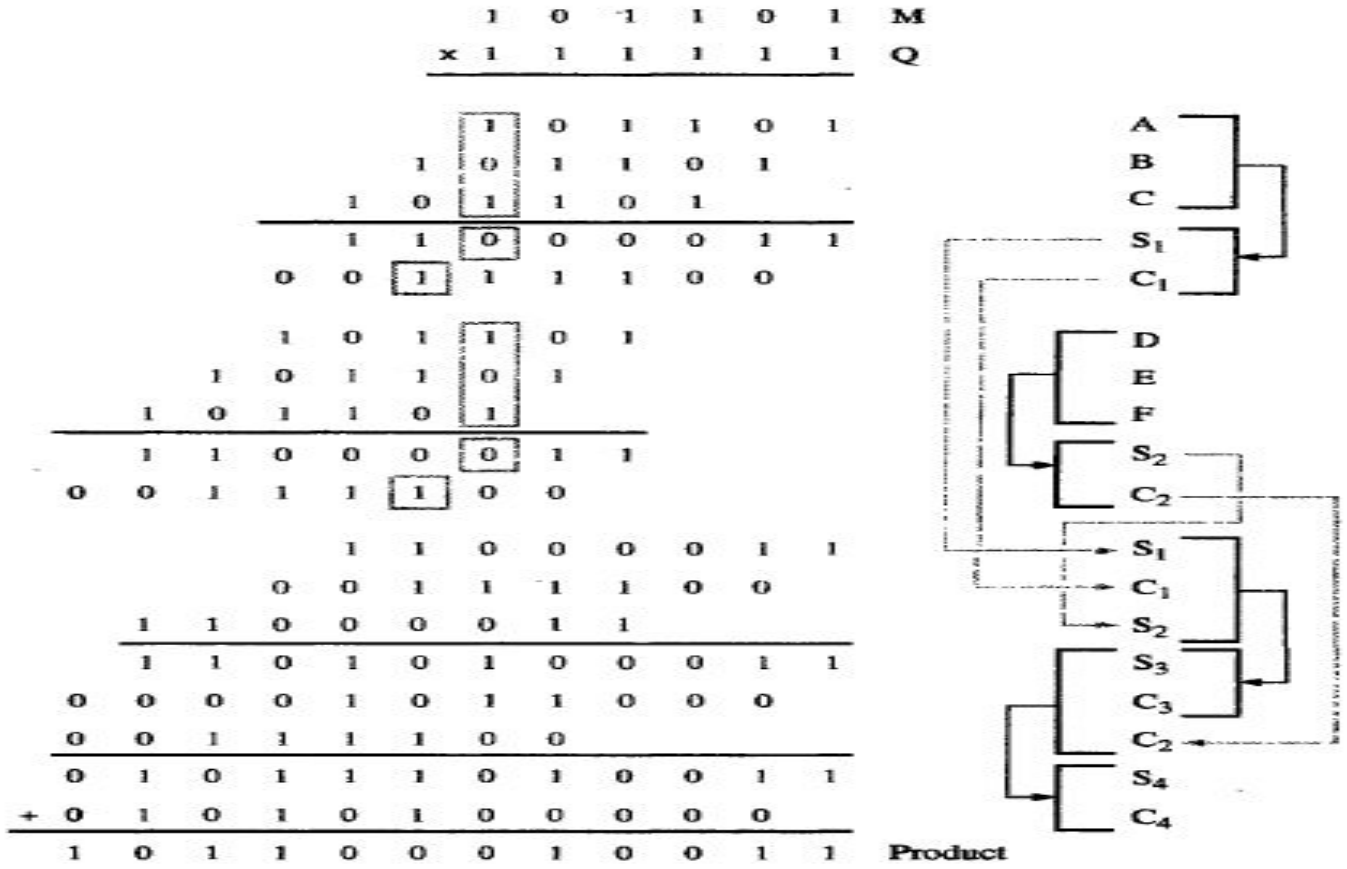
			1	0	1	1	0	1				D
		1	0	1	1	0	1					E
	1	0	1	1	0	1						F
	1	1	0	0	0	0	1	1				S_2
0	0	1	1	1	1	0	0					C_2
				1	1	0	0	0	0	1	1	S_1
			0	0	1	1	1	1	0	0		C_1
	1	1	0	0	0	0	1	1				S_2
	1	1	0	1	0	1	0	0	0	1	1	S_3
0	0	0	0	1	0	1	1	0	0	0		C_3
0	0	1	1	1	1	0	0					C_2
0	1	0	1	1	1	0	1	0	0	1	1	S_4
+ 0	1	0	1	0	1	0	0	0	0	0		C_4
1	0	1	1	0	0	0	1	0	0	1	1	Product

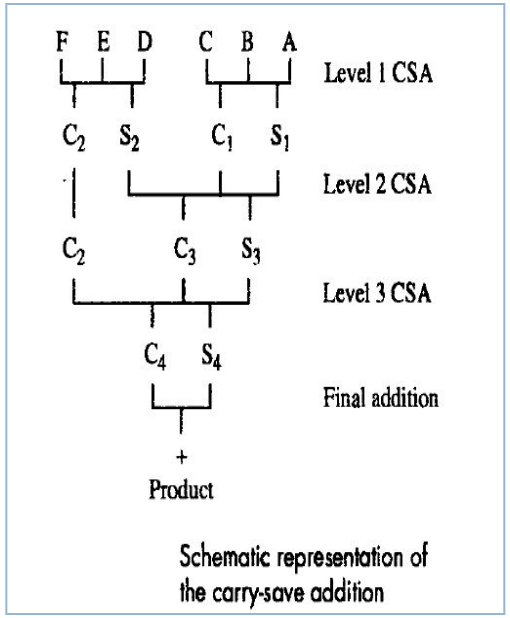


Schematic representation of the carry-save addition

RAMAIAH
Institute of Technology

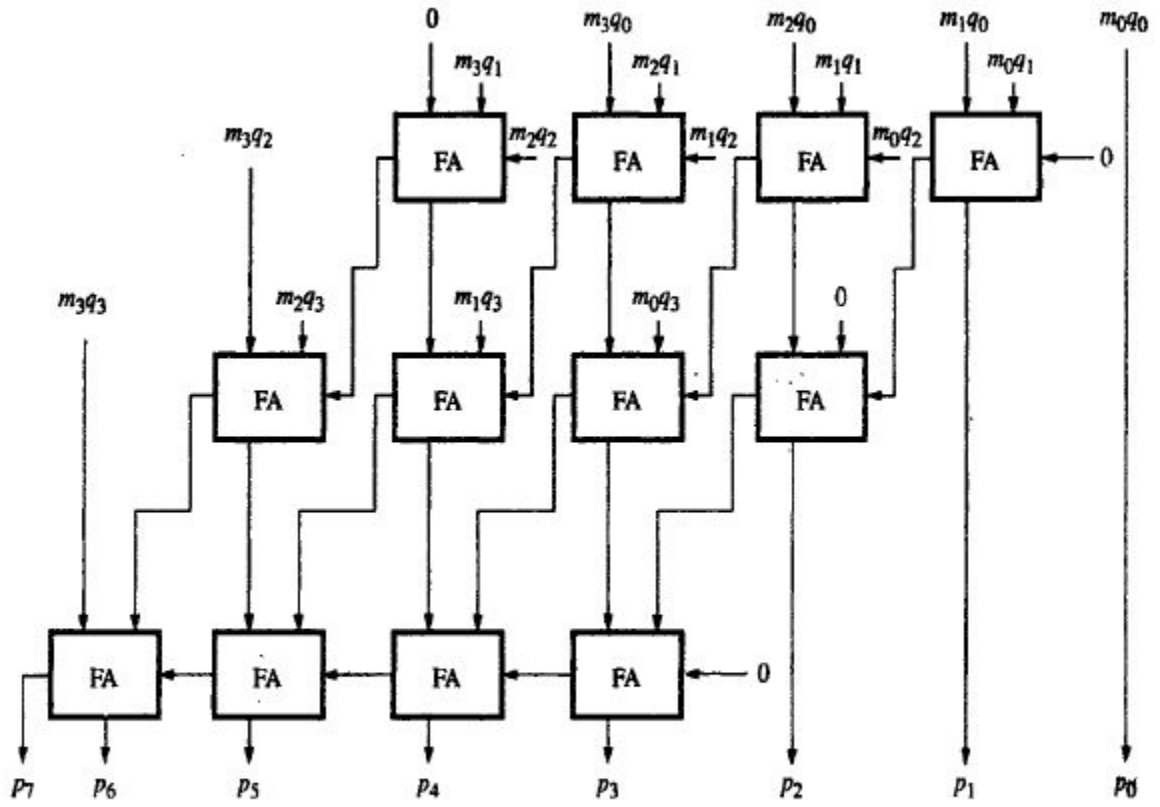
Carry-Save Addition(CSA) of Summands





Schematic representation of the carry-save addition

Carry-Save Addition(CSA) of Summands



(b) Carry-save array

Ripple-carry and carry-save arrays for the multiplication operation M x Q = P for 4-bit operands.

					1	0	1	1	0	1	(45)	M
				x	1	1	1	1	1	1	(63)	Q
					1	0	1	1	0	1		A
				1	0	1	1	0	1			B
			1	0	1	1	0	1				C
		1	0	1	1	0	1					D
	1	0	1	1	0	1						E
1	0	1	1	0	1							F
1	0	1	1	0	0	0	1	0	0	1	1	(2,835)	Product

Examples for CSA

1. Write the schematic representation of carry save addition and perform 47 x 60 using CSA
2. Illustrate the working of CSA with 45 and 63 as the Multiplicand and Multiplier respectively.

UNIT 2 – Arithmetic unit

Multiplication of two Numbers

A Signed Operand Multiplication

- Booth algorithm

Fast Multiplication

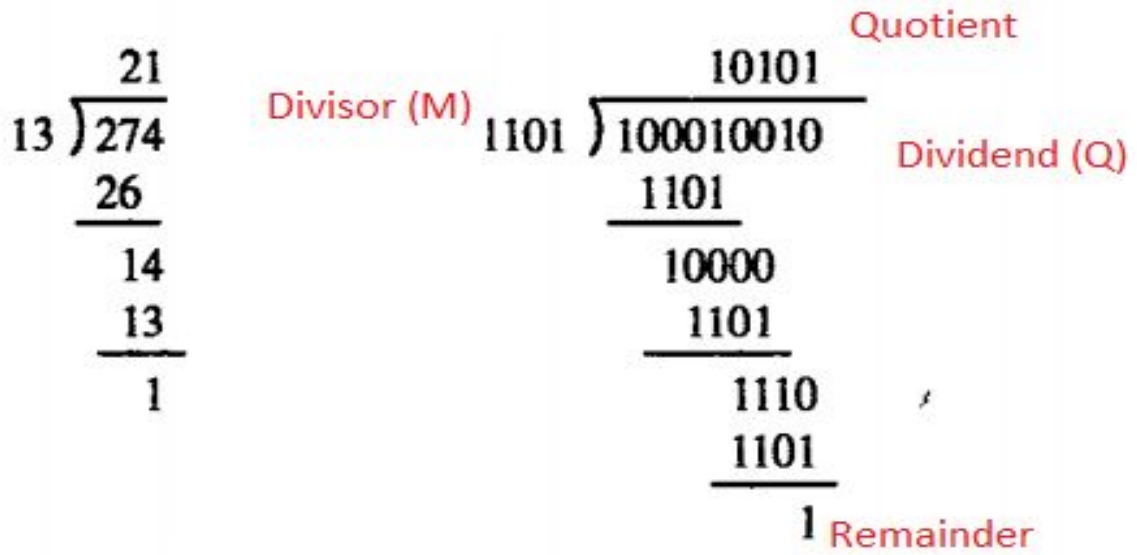
- Bit pair recoding
- Carry Save Addition(CSA)

Integer Division

- Restoring Division
- Nonrestoring Division

IEEE standard for floating point numbers

Integer Division



Longhand division examples.

Algorithm to perform restoring division

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

Count =4	A	Q		M
	0 0 0 0 0	1 0 1 0		0 0 0 1 1

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1

Count =4	A					Q				M					
	0	0	0	0	0	1	0	1	0		0	0	0	1	1
Shift	0	0	0	0	1	0	1	0							

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

Count =4	A	Q
	0 0 0 0 0	1 0 1 0
Shift	0 0 0 0 1	0 1 0
Subtract	1 1 1 0 1	

M | 0 | 0 | 0 | 1 | 1 |

2's complement of M

M = 0 0 0 1 1
1 1 1 0 0
1
1 1 1 0 1

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1

Count =4

	A					Q				M					
	0	0	0	0	0	1	0	1	0		0	0	0	1	1
Shift	0	0	0	0	1	0	1	0							
Subtract	1	1	1	0	1										
	1	1	1	1	0										

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1

Count =4

	A					Q				M				
	0	0	0	0	0	1	0	1	0		0	0	0	1	1
Shift	0	0	0	0	1	0	1	0							
Subtract	1	1	1	0	1										
Set q0	1	1	1	1	0										
						0	1	0	0						

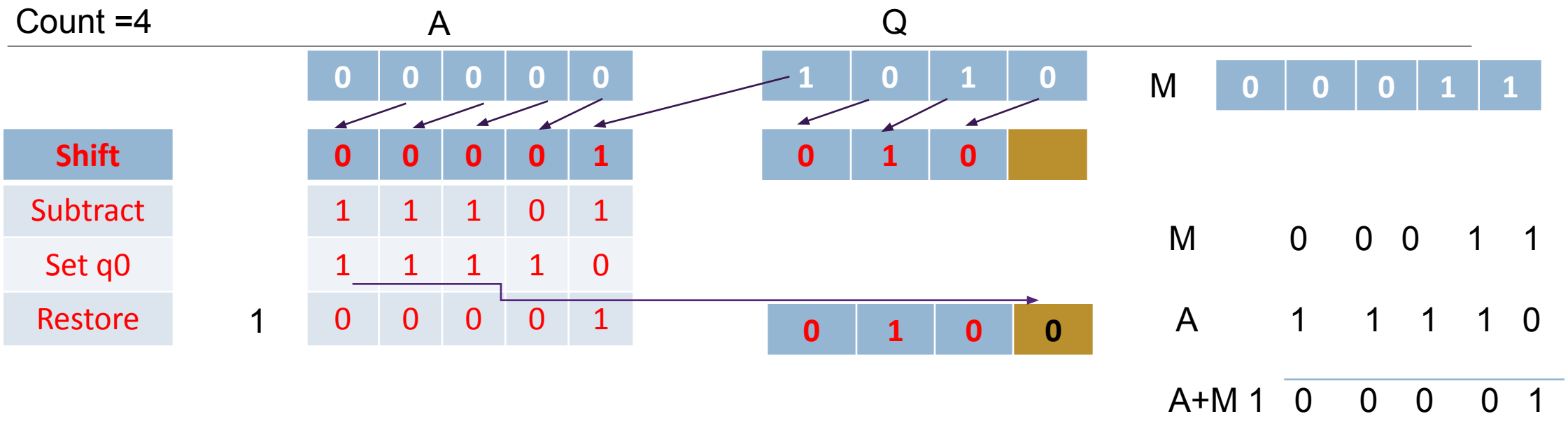
1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

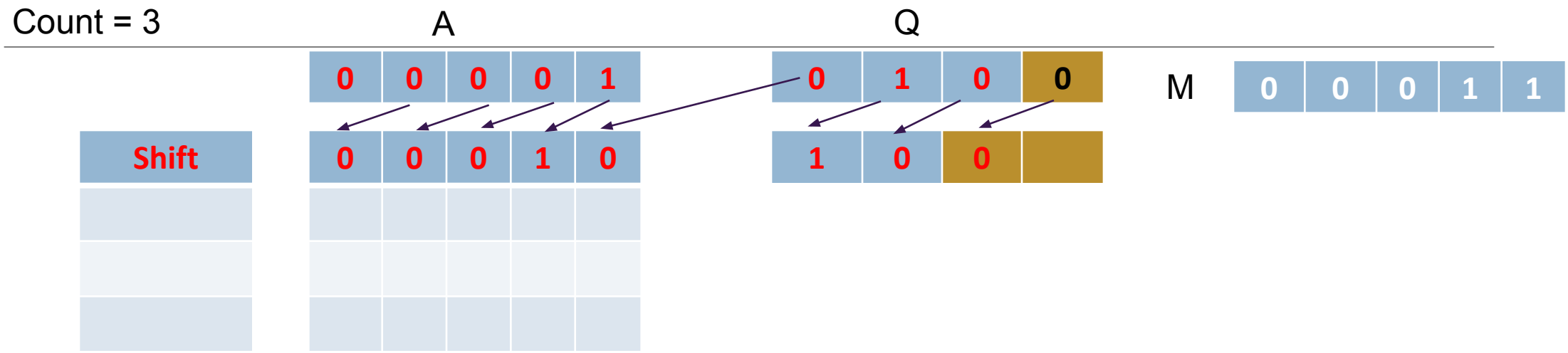


1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1



1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

Count =3	A	Q
	0 0 0 0 1	0 1 0 0
Shift	0 0 0 1 0	1 0 0
Subtract	1 1 1 0 1	

M | 0 | 0 | 0 | 1 | 1 |

2's complement of M

M = 0 0 0 1 1
1 1 1 0 0
1
1 1 1 0 1

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1

Count =3	A					Q				M					
	0	0	0	0	1	0	1	0	0		0	0	0	1	1
Shift	0	0	0	1	0	1	0	0							
Subtract	1	1	1	0	1										
	1	1	1	1	1										

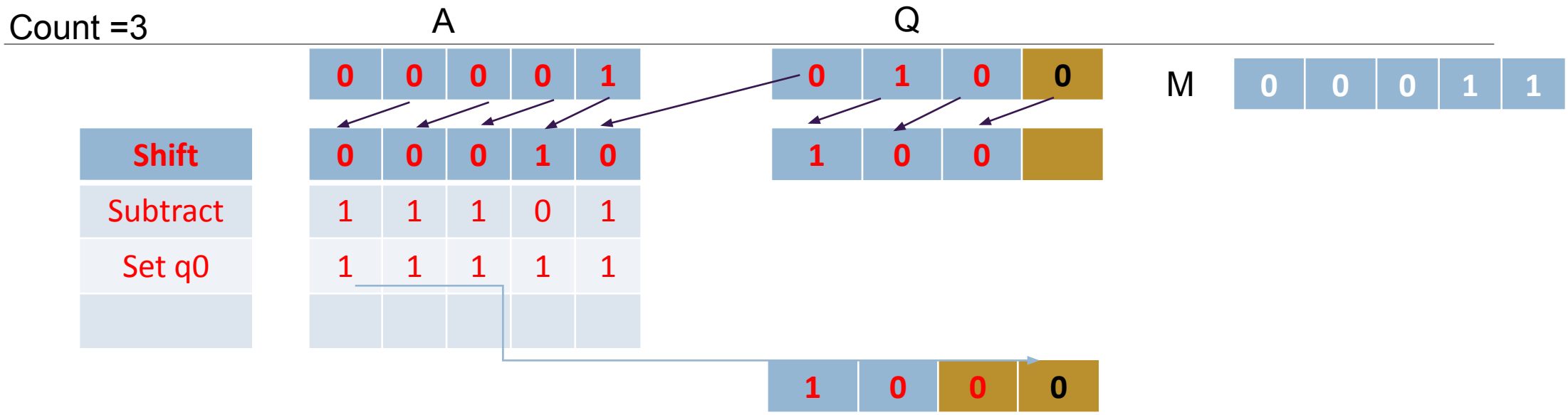
1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

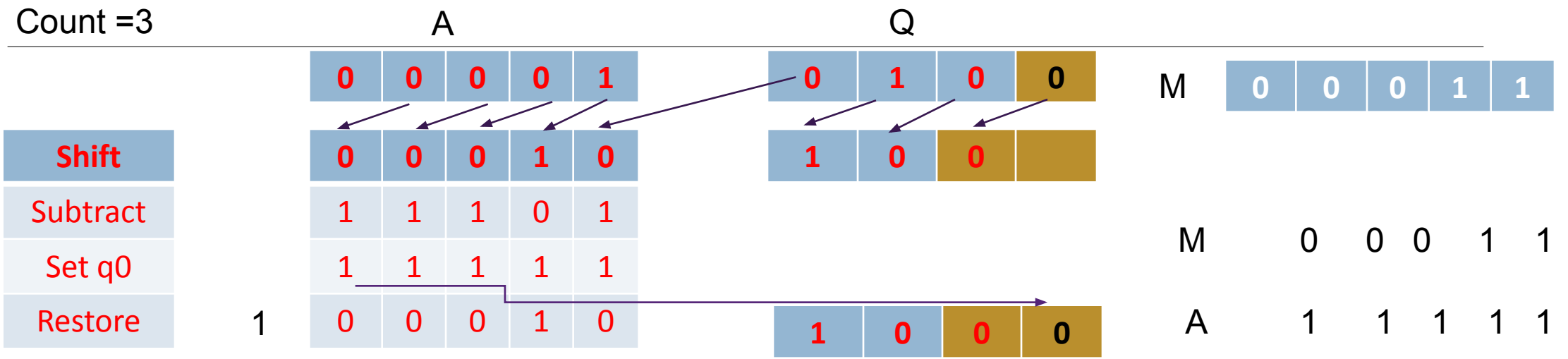


1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1

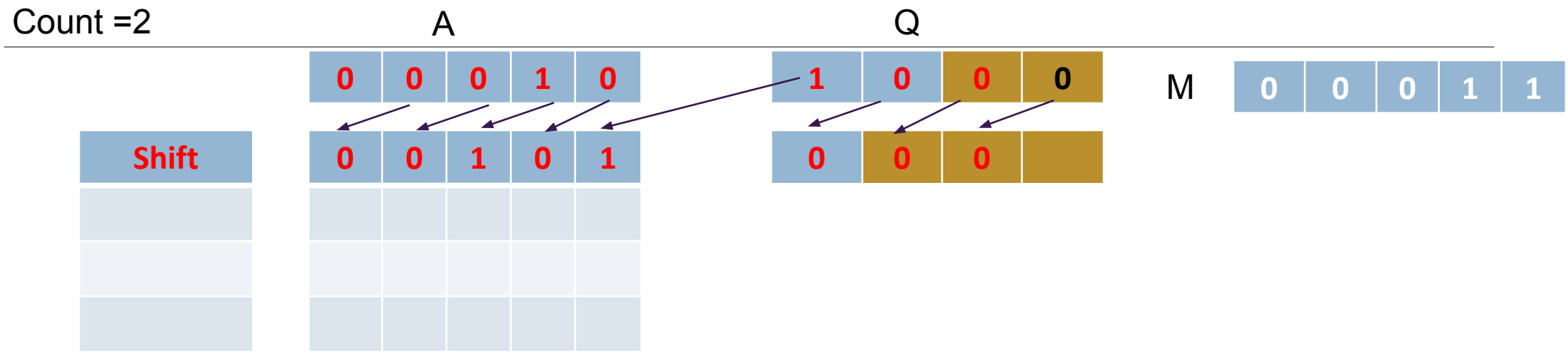


1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1



1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

Count =2

	A				Q			
	0	0	0	1	0	1	0	0	0
Shift	0	0	1	0	1	0	0	0	
Subtract	1	1	1	0	1				

M | 0 | 0 | 0 | 1 | 1 |

2's complement of M

M = 0 0 0 1 1
1 1 1 0 0
1
1 1 1 0 1

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1

Count =2

	A					Q				M					
	0	0	0	1	0	1	0	0	0		0	0	0	1	1
Shift	0	0	1	0	1	0	0	0							
Subtract	1	1	1	0	1										
	0	0	0	1	0										

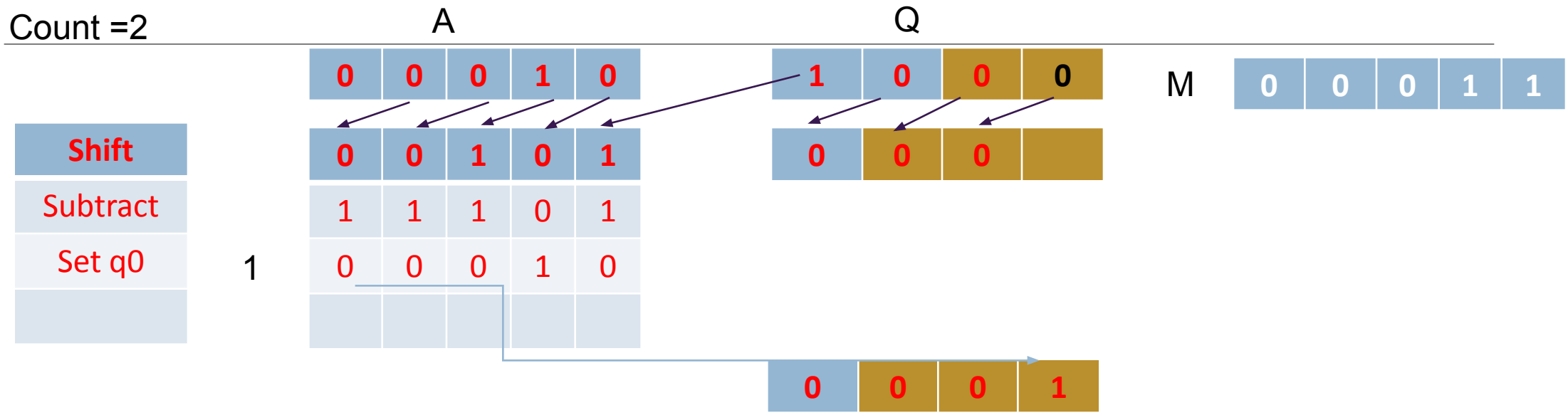
1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

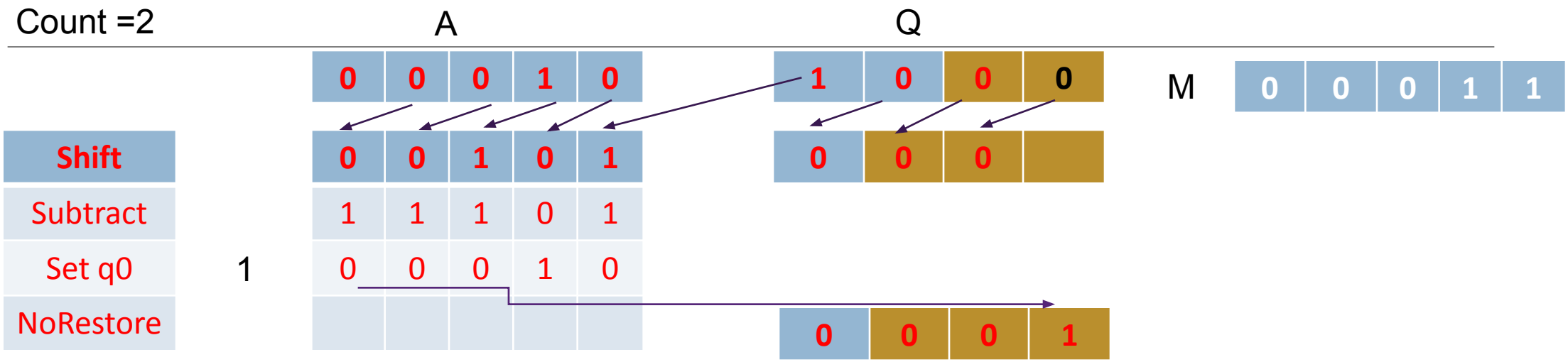


1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10	1	1	0	1
Divisor M=3	0	0	1	1



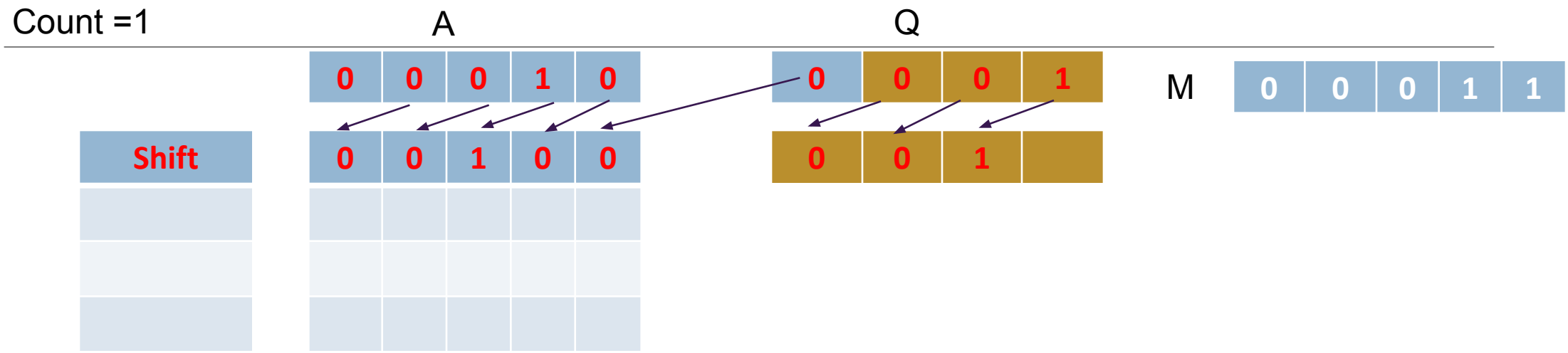
1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |



1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

Count =1

	A	Q
	0 0 0 1 0	0 0 0 1
Shift	0 0 1 0 0	0 0 1
Subtract	1 1 1 0 1	

M | 0 | 0 | 0 | 1 | 1 |

2's complement of M

M = 0 0 0 1 1
1 1 1 0 0
1
1 1 1 0 1

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |

Count =1

	A	Q	M
	0 0 0 1 0	0 0 0 1	0 0 0 1 1
Shift	0 0 1 0 0	0 0 1	
Subtract	1 1 1 0 1		
1	0 0 0 0 1		

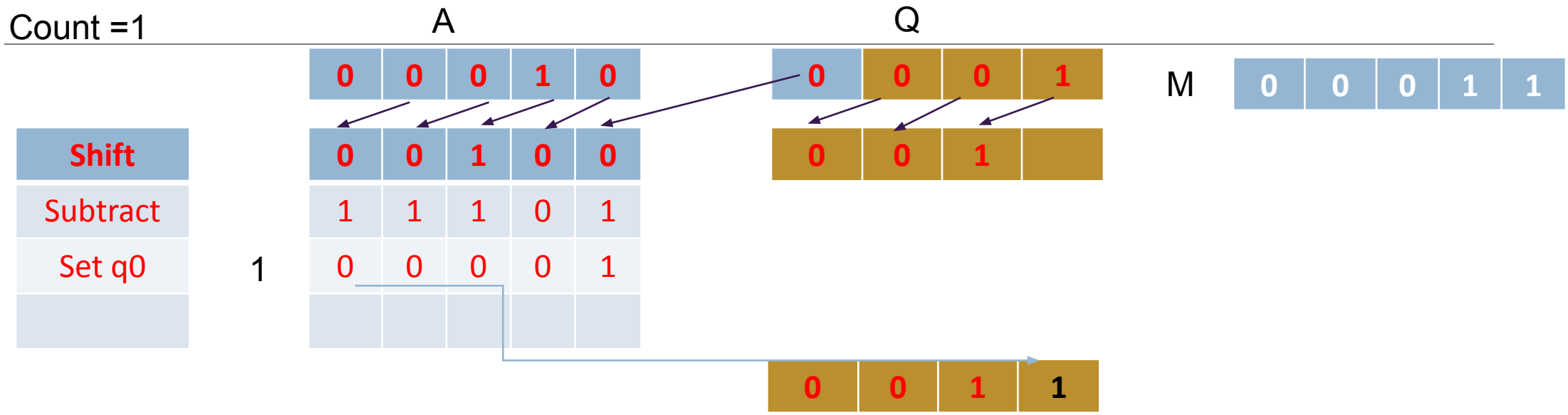
1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |



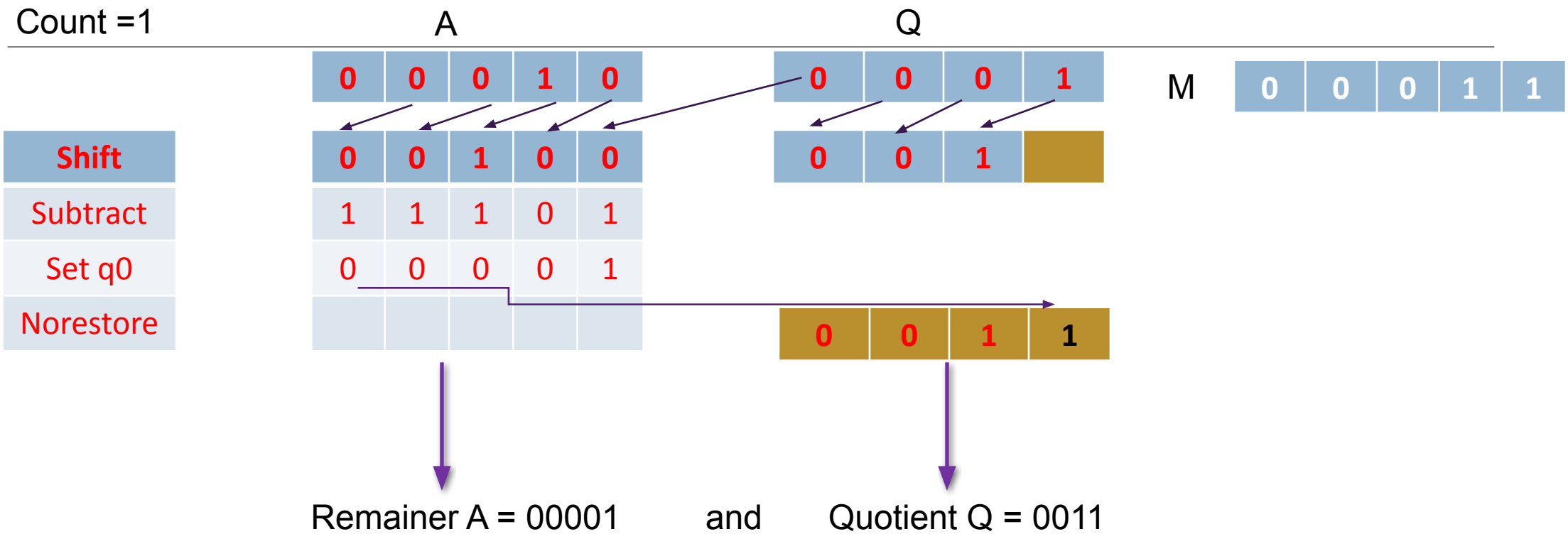
1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

RAMAIAH Institute of Technology

1. Compute 10/3 using restoring division

Dividend Q=10 | 1 | 1 | 0 | 1 |

Divisor M=3 | 0 | 0 | 1 | 1 |



RAMAIAH Institute of Technology

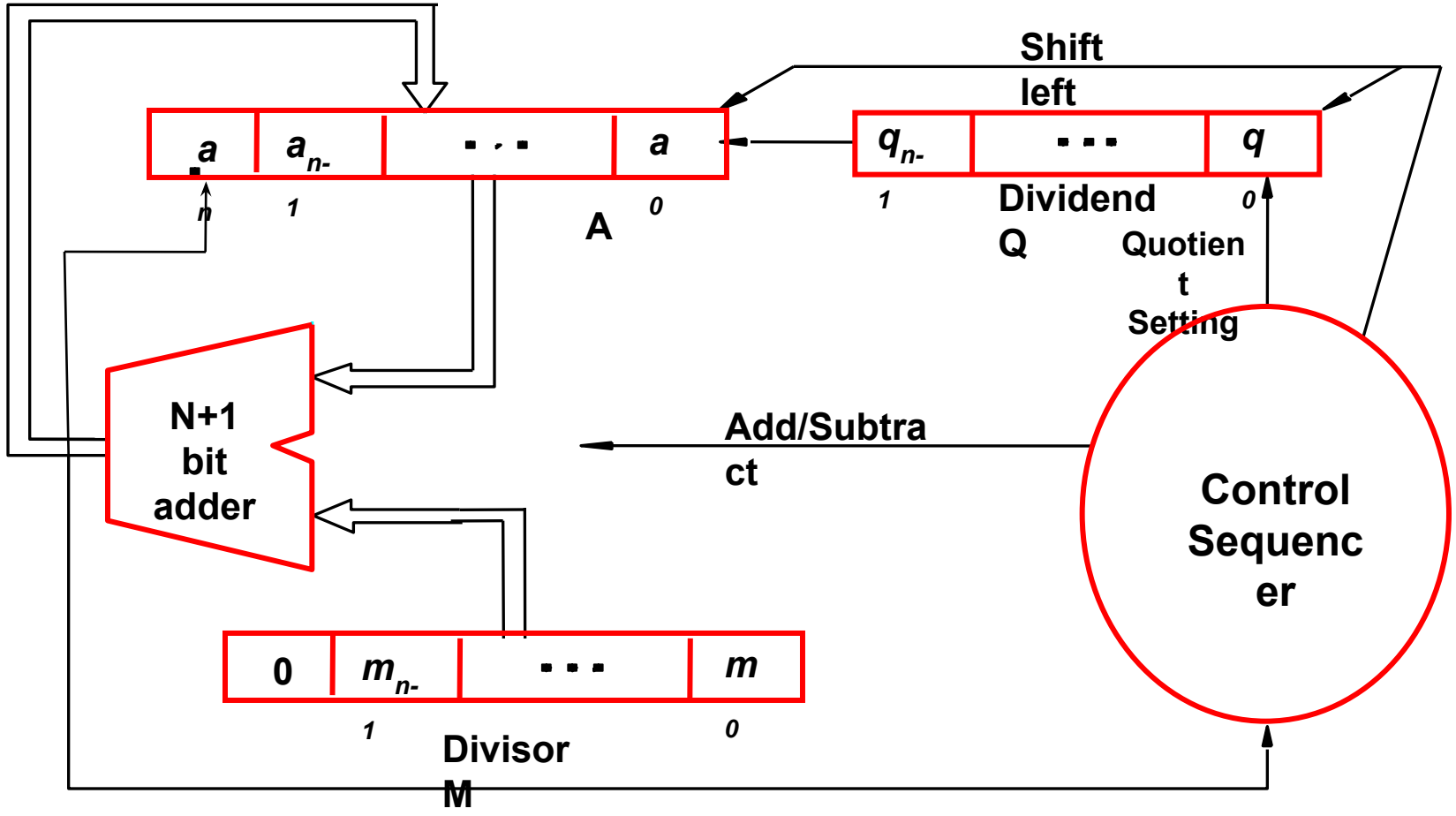
2. Compute 8/3 using restoring division

```
        1 0
    ------------
1 1 ) 1 0 0 0
      1 1
    -------
      1 0
```

Initially	0 0 0 0 0	1 0 0 0	
	0 0 0 1 1		
Shift	0 0 0 0 1	0 0 0 □	First cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0		
Restore	1 1		
	0 0 0 0 1	0 0 0 0	
Shift	0 0 0 1 0	0 0 0 □	Second cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1		
Restore	1 1		
	0 0 0 1 0	0 0 0 0	
Shift	0 0 1 0 0	0 0 0 □	Third cycle
Subtract	1 1 1 0 1		
Set q_0	0 0 0 0 1		
		0 0 0 1	
Shift	0 0 0 1 0	0 0 0 1	Fourth cycle
Subtract	1 1 1 0 1	0 0 1 □	
Set q_0	1 1 1 1 1		
Restore	1 1		
	0 0 0 1 0	0 0 1 0	
	Remainder	Quotient	

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. Subtract M from A, and place the answer back in A (A = A-M)
4. Observe the sign bit of A
   - If the sign bit of A is 1, set q_0 = 0 and add M back to A(A=A+M) restoration is done
   - If the sign bit of A is 0, set q_0 = 1 ,norestoration is done
5. Count = Count – 1
6. Repeat step 2,3,4,5 until count = 0

Restoring division - Circuit Arrangement



Examples for Restoring Division

1. Perform the following using restoring division algorithm.
   i)20/5 ii) 12/3
2. Give an Algorithm for Restoring division method.
3. Compute 22/5 using restoring division

Algorithm to perform Nonrestoring division

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. After left shift ,if the sign bit of A is 0 , then

Perform A=A-M	Then if Sign bit of A=1	Set Q_0=0
	Then if Sign bit of A =0	Set Q_0=1

else, If the sign bit of A is 1, then

Perform A=A+M	Then if Sign bit of A=1	Set Q_0=0
	Then if Sign bit of A =0	Set Q_0=1

4. Repeat step 2 and 3 , n times
5. If the sign bit of A is 1, add Divisor to A

RAMAIAH Institute of Technology

1. Compute 8/3 using nonrestoring division

$$\begin{array}{r} 1\ 0 \\ 1\ 1 \overline{)\ 1\ 0\ 0\ 0} \\ \underline{1\ 1} \\ 1\ 0 \end{array}$$

M = 0 0 0 1 1

1. Initialize Register A = n+1 bits with 0's and Register Q with Dividend bits and M with Divisor bits, Count = n
2. Shift A and Q left one binary position
3. After left shift ,if the sign bit of A is 0 , then

Perform A=A-M	Then if Sign bit of A=1	Set Q_0=0
	Then if Sign bit of A =0	Set Q_0=1

else, If the sign bit of A is 1, then

Perform A=A+M	Then if Sign bit of A=1	Set Q_0=0
	Then if Sign bit of A =0	Set Q_0=1

4. Repeat step 2 and 3 , n times
5. If the sign bit of A is 1, add Divisor to A

	A	Q	
Initially	0 0 0 0 0	1 0 0 0	
Shift	0 0 0 0 1	0 0 0 _	First cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0	0 0 0 0	
Shift	1 1 1 0 0	0 0 0 _	Second cycle
Add	0 0 0 1 1		
Set q_0	1 1 1 1 1	0 0 0 0	
Shift	1 1 1 1 0	0 0 0 _	Third cycle
Add	0 0 0 1 1		
Set q_0	0 0 0 0 1	0 0 0 1	
Shift	0 0 0 1 0	0 0 1 _	Fourth cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1	0 0 1 0	
		Quotient	

Add: 1 1 1 1 1 / 0 0 0 1 1 / 0 0 0 1 0 — Remainder — **Restore remainder**

M = 0 0 0 1 1 / 1 1 1 0 0 / 1 / 1 1 1 0 1

Examples for Nonrestoring division

1. State and apply non-restoring division algorithm to perform integer division 59/5.
2. Perform Nonrestoring division 10/3
3. Perform Nonrestoring division 11/5
4. Perform Nonrestoring division 23/5

RAMAIAH
Institute of Technology

UNIT 2 – Arithmetic unit

Multiplication of two Numbers

A Signed Operand Multiplication

- Booth algorithm

Fast Multiplication

- Bit pair recoding
- Carry Save Addition(CSA)

Integer Division

- Restoring Division
- Nonrestoring Division

IEEE standard for floating point numbers

IEEE standard for floating point numbers

- The IEEE Standard for Floating-Point Arithmetic is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**.
- The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability.
- IEEE Standard floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.

IEEE standard for floating point numbers

Datatype Representation

- Fixed Point number
- Floating Point number

Fixed Point number

- $(+7)_{10} = (0\ 111)_2$
- $(-7)_{10} = (100\ 1)_2$

```
0 1 1 1
1 0 0 0
      1
----------
1 0 0 1
```

IEEE standard for floating point numbers

Floating Point numbers -It has 3 parts

- Mantissa (M)
- Base (B)
- Exponent (E)

Scientific Notation $\pm M \times B^E$

Number	M	B	E
9×10^8	9	10	8
110×2^7	110	2	7
4364.784	4364784	10	-3

IEEE standard for floating point numbers

- Single Precision format

$\pm M \times B^E$

1.The Sign of Mantissa

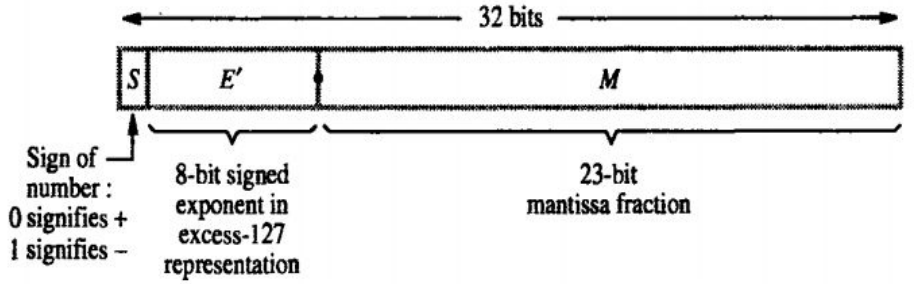
0 represents a positive number while 1 represents a negative number.

2.The Biased exponent

The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.

3.The Normalised Mantissa

The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

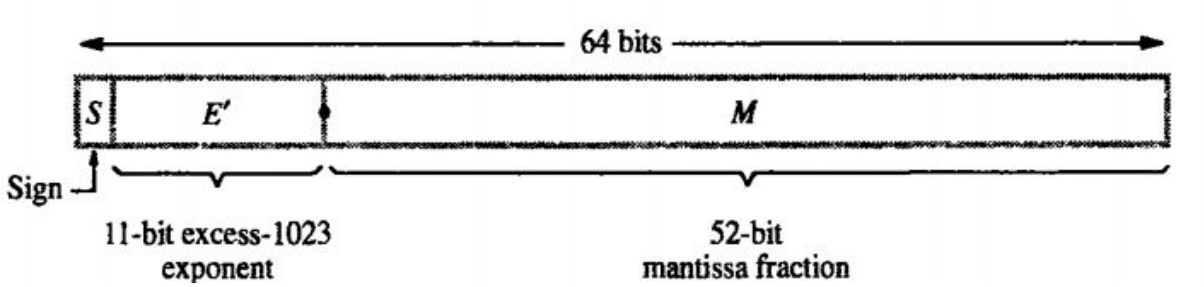


$$\text{Value represented} = \pm 1.M \times 2^{E' - 127}$$

IEEE standard for floating point numbers

- Double Precision format

$\pm M \times B^E$



- Normalize

$$\text{Value represented} = \pm 1.M \times 2^{E' - 1023}$$

Example

1.Represent $(1259.125)_{10}$ in single and double precision IEEE floating point format.

Solution

Step 1: Convert decimal to Binary form

Step 2: Normalize the number

Step 3: Represent in Single precision form

Step 4: Represent in double precision form

Example

1.Represent $(1259.125)_{10}$ in single and double precision IEEE floating point format.

Solution

Step 1: Convert decimal to Binary form

$(1259)_{10}$	10011101011		
$(0.125)_{10}$	0.125 x 2 = 0.25	0	001
	0.25 x 2 = 0.5	0	
	0.5 x 2 = 1	1	
$(1259.125)_{10}$ = (10011101011.001)			

Example

1.Represent $(1259.125)_{10}$ in single and double precision IEEE floating point format.

Solution

Step 2: Normalize the number

$(1259.125)_{10}$ = (10011101011.001)

$$\pm 1.M \times 2^{E'-127}$$

$$\pm 1.M \times 2^{E'-1023}$$

$\rfloor$ $1.0011101011001 \times 2^{10}$

Example

1.Represent $(1259.125)_{10}$ in single and double precision IEEE floating point format.

Solution

Step 3: Represent in Single precision form

$\pm 1.M \times 2^{E'-127}$ □ $1.0011101011001 \times 2^{10}$

$2^{E'-127}$ □ 2^{10}

E' - 127 = 10

0	10001001	00111010110010000000000
S-1bit	E' -8bits	M- 23bits

E' = 137 □ Binary form 10001001

Example

1.Represent $(1259.125)_{10}$ in single and double precision IEEE floating point format.

Solution

Step 4: Represent in Double precision form

□ $1.0011101011001 \times 2^{10}$

$\pm 1.M \times 2^{E' - 1023}$ □ 2^{10}

E' - 1023 = 10

0	10000001001	00111010110010000000
S-1bit	E' -11bits	M- 52bits

E' = 1033 □ Binary form 10000001001

Example

2. Compute the single precision IEEE 754 binary representation of the number $(-118.625)_{10}$.

Solution

Step 1: Convert decimal to Binary form

$(118)_{10}$	1110110		
$(0.625)_{10}$	0.625 x 2 = 1.25	1	101
	0.25 x 2 = 0.5	0	
	0.5 x 2 = 1	1	
$(118.625)_{10}$ = $(1110110.101)_2$			

Example

2.Compute the single precision IEEE 754 binary representation of the number $(-118.625)_{10}$.

Step 2: Normalize the number

$(118.625)_{10} = (1110110.101)_2$

$\pm 1.M \times 2^{E'-127}$ | 1.110110101×2^6

Example

2.Compute the single precision IEEE 754 binary representation of the number $(-118.625)_{10}$.

Step 3: Represent in Single precision form

$\pm 1.M \times 2^{E'-127}$ □ 1.110110101×2^6

$2^{E'-127}$ □ 2^6

E' - 127 = 6

1	10000101	11011010100...000000
S-1bit	E' -8bits	M- 23bits

E' = 133 □ Binary form 10000101

Example

3.Represent $(85.125)_{10}$ in single and double precision IEEE floating point format.

Solution

Step 1: Convert decimal to Binary form

$(85)_{10}$	**1010101**		
$(0.125)_{10}$	0.125 x 2 = 0.25	0	001
	0.25 x 2 = 0.5	0	
	0.5 x 2 = 1	1	
$(85.125)_{10}$ = (1010101.001)			

Example

3.Represent $(85.125)_{10}$ in single and double precision IEEE floating point format.

Step 2: Normalize the number

$(85.125)_{10}$ = (1010101.001)

$$\pm 1.M \times 2^{E'-127}$$

$$\pm 1.M \times 2^{E'-1023}$$

$$1.010101001 \times 2^{6}$$

10000101

RAMAIAH Institute of Technology

Example

3.Represent $(85.125)_{10}$ in single and double precision IEEE floating point format.

Step 3: Represent in Single precision form

□ 1.010101001×2^6

$\pm 1.M \times 2^{E'-127}$

$2^{E'-127}$

□ 2^6

= 6

0	10000101	01010100100000..0000
S-1bit	E' -8bits	M- 23bits

E' = 133 □ Binary form 10001001

Example

3.Represent $(85.125)_{10}$ in single and double precision IEEE floating point format.

Step 4: Represent in Double precision form

☐ 1.010101001×2^6

$\pm 1.M \times 2^{E' - 1023}$ ☐ 2^6

E' - 1023 = 6

0	10000000101	0101010010.....0000
S-1bit	E' -11bits	M- 52bits

E' = 1029 ☐ Binary form 10000000101

Example

4.Represent $(263.3)_{10}$ in single precision IEEE floating point format.

Solution

Step 1: Convert decimal to Binary form

$(263)_{10}$	100000111		
$(0.3)_{10}$	0.3 x 2 = 0.6	0	010011001….
	0.6 x 2 = 1.2	1	
	0.2 x 2 = 0.4	0	
	0.4 x 2 = 0.8	0	
	0.8 x 2 = 1.6	1	
$(263.3)_{10}$ = (100000111. **010011001….)**			

0.3 *2= 0.6 □ 0
0.6 *2= 1.2 □ 1
0.2 *2= 0.4□ 0
0.4 *2 = 0.8 □ 0
0.8 *2= 1.6 □ 1
0.6 *2 =1.2 □ 1
0.2 *2= 0.4□ 0
0.4 *2 = 0.8 □ 0
0.8 *2= 1.6 □ 1
0.6 *2 =1.2 □ 1
0.2 *2= 0.4□ 0
0.4 *2 = 0.8 □ 0
0.8 *2= 1.6 □ 1

Example

4.Represent $(263.3)_{10}$ in single precision IEEE floating point format.

Solution

Step 2: Normalize the number

$(263.3)_{10}$ = (100000111. 010011001….)

$$\pm 1.M \times 2^{E'-127}$$

$$= 1.00000111010011001\ldots\ldots \times 2^{8}$$

$$\pm 1.M \times 2^{E'-1023}$$

Example

4.Represent $(1259.125)_{10}$ in single precision IEEE floating point format.

Solution

Step 3: Represent in Single precision form

$\pm 1.M \times 2^{E'-127}$ ☐ 1.00000111010011001..... x 2^8

$2^{E'-127}$ ☐ 2^8

E' - 127 = 8

E' = 135 ☐ Binary form 10000111

0	10000111	00000111**010011001.....**
S-1bit	E' -8bits	M- 23bits

Examples

1.Use IEEE single and double precision formats to represent the following floating point numbers.

i) 120.75 ii) -875.45

2. Represent the following in binary using IEEE 754 Double precision format:

i) 12.0875 ii) 0.625.

3. Show the following numbers using IEEE double precision formats.

i)124.45 ii) -369.25.

4. Represent -307.1875 in single and double precision IEEE floating point numbers.

RAMAIAH
Institute of Technology

UNIT 2 – Arithmetic unit & Processing unit

- Multiplication of two Numbers
- A Signed Operand Multiplication
  - Booth algorithm
- Fast Multiplication
  - Bit pair recoding
  - Carry Save Addition(CSA)
- Integer Division
  - Restoring Division
  - No restoring Division
- IEEE standard for floating point numbers
- Fundamental concepts
- Execution of complete instruction
- Multiple bus organization
- Hardwired control
- Microprogrammed control

Fundamental concepts

Processor

- Data processing unit
- Control unit

Data processing unit

- Collection of functional unit capable of performing certain data operations on data

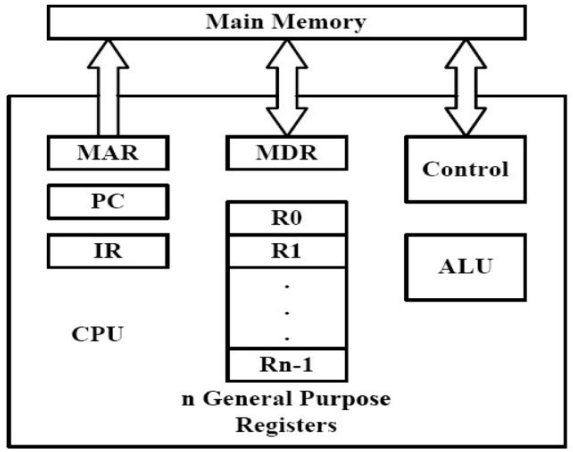
Control unit

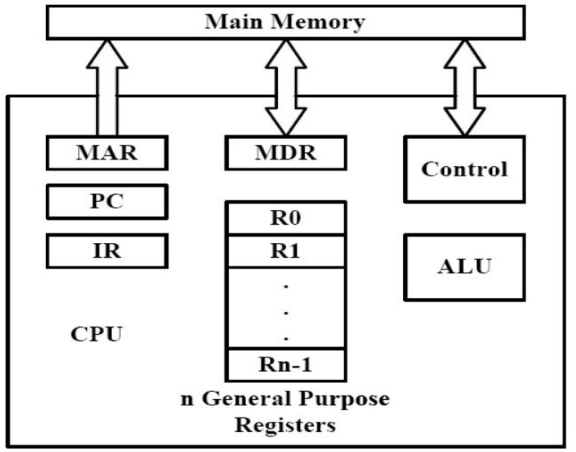
- Issues control signals to data processing part to perform operation on data

Fundamental Concepts

To execute a program

- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)





Fundamental Concepts

To Execute an Instruction

1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

$$IR \leftarrow [[PC]]$$

2. Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

$$PC \leftarrow [PC] + 4$$

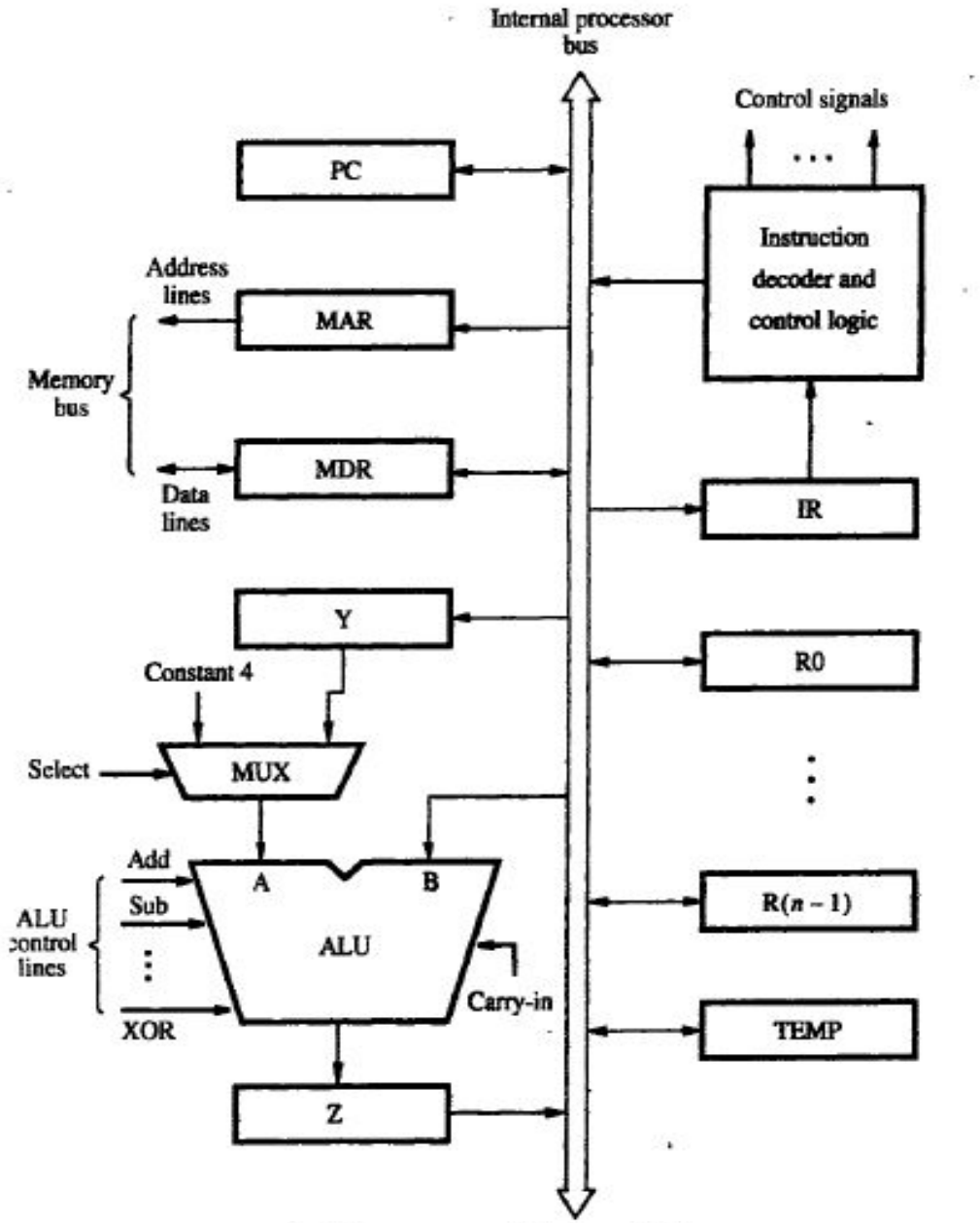
3. Carry out the actions specified by the instruction in the IR (execution phase).

Fundamental Concepts

SINGLE BUS ORGANIZATION

- Bus is communication system that transfers data between component inside a computer, or between computers.

- In Single bus organization, one data item can be transferred over the bus in a clock cycle

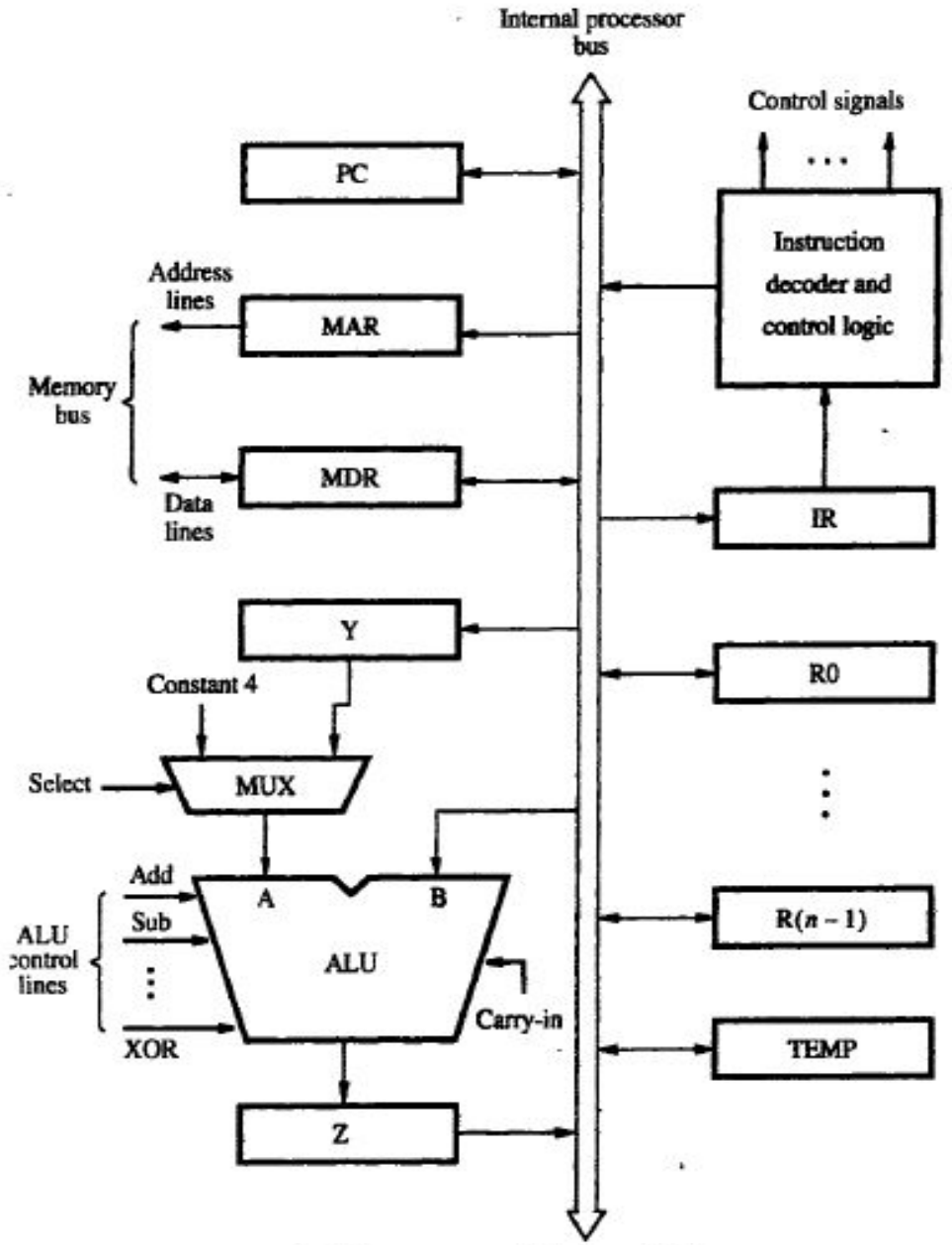


Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

- ALU and all the registers are interconnected via a single common bus
- The data and address lines of the external memory bus connected to the internal processor bus via the memory data register (MDR), and the memory address register(MAR) respectively.

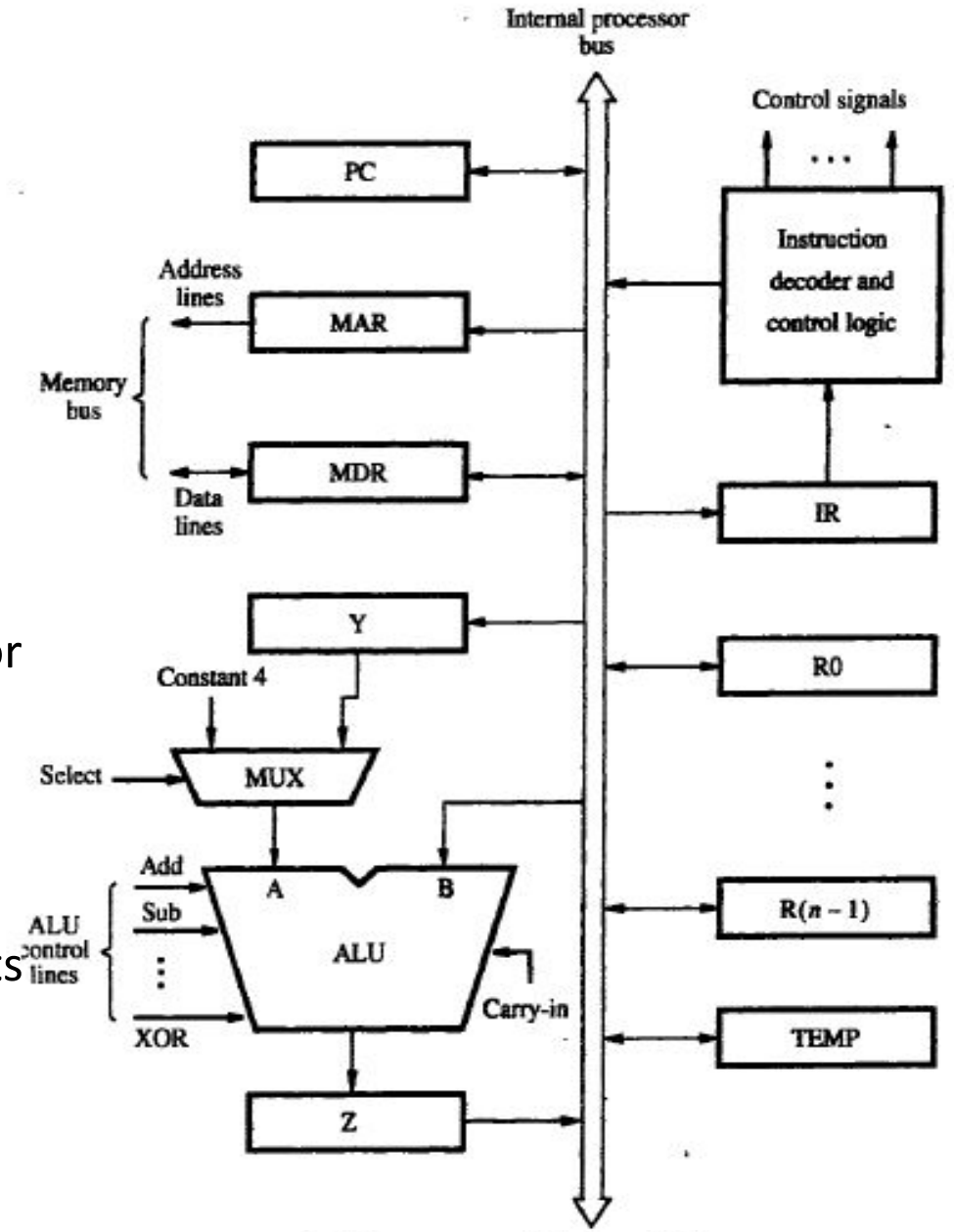


Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

- Register MDR has two inputs and two outputs.
- Data may be loaded into MDR either from the memory bus or from the internal processor bus.
- Data stored in MDR may be placed on either bus.
- The input of MAR is connected to the internal bus, and its output is connected to the external bus.

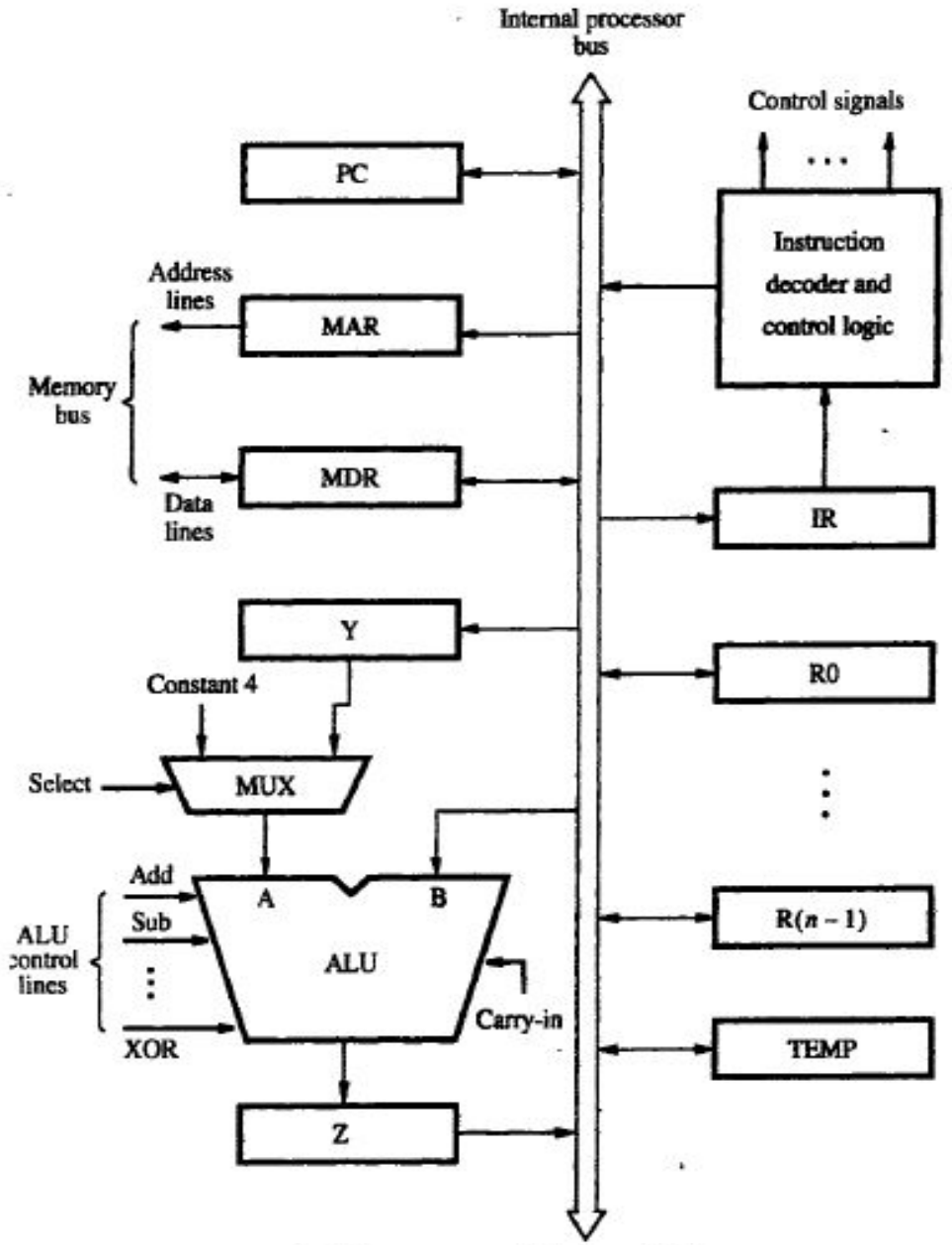


Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

- The control lines of the memory bus are connected to the instruction decoder and control logic.
- This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for increasing with the memory bus.



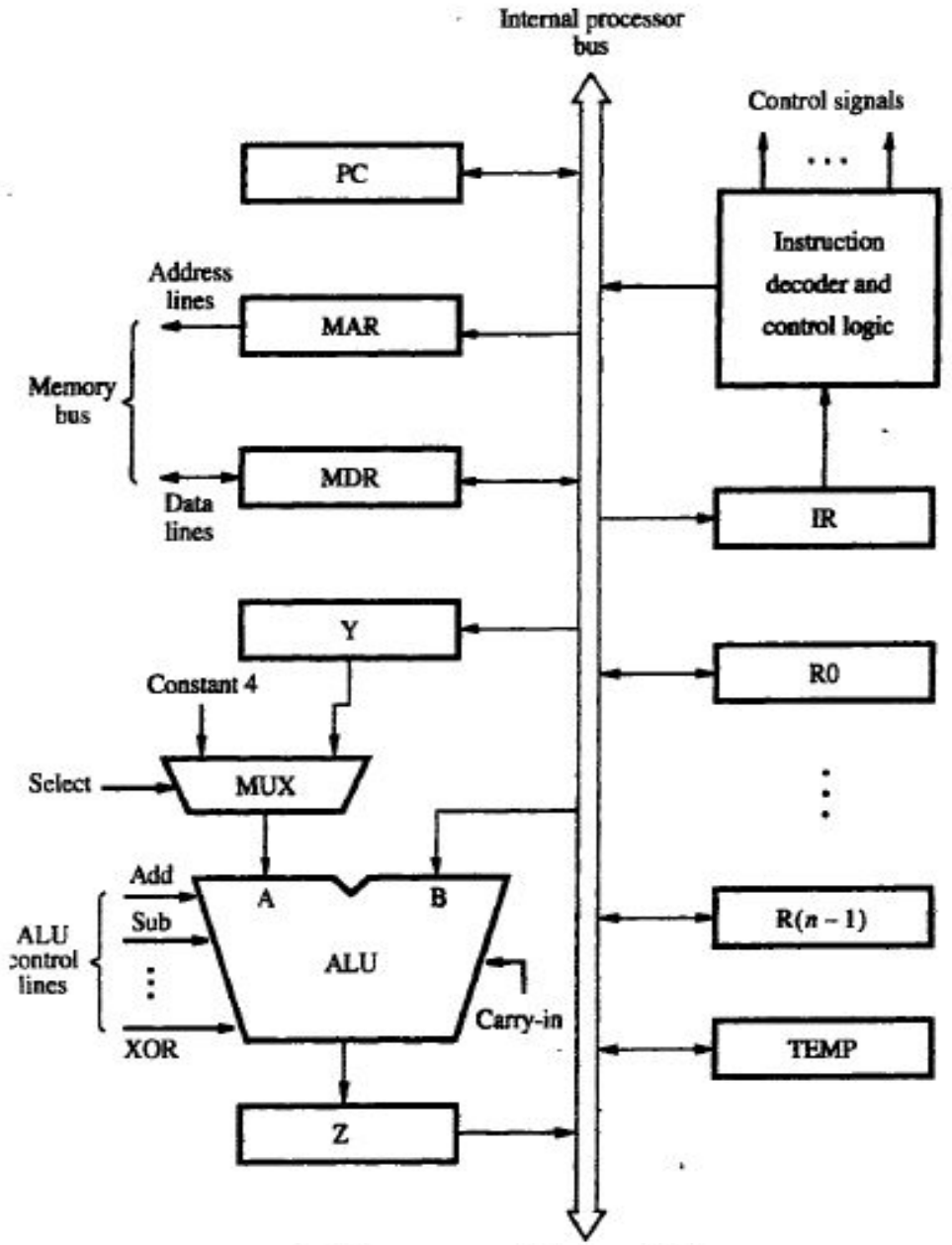
Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

- Register R0 through R(n-1) are the Processor Registers. The programmer can access these registers for general-purpose use.

- Only processor can access 3 registers Y, Z & Temp for temporary storage during program-execution. The programmer cannot access these 3 registers.

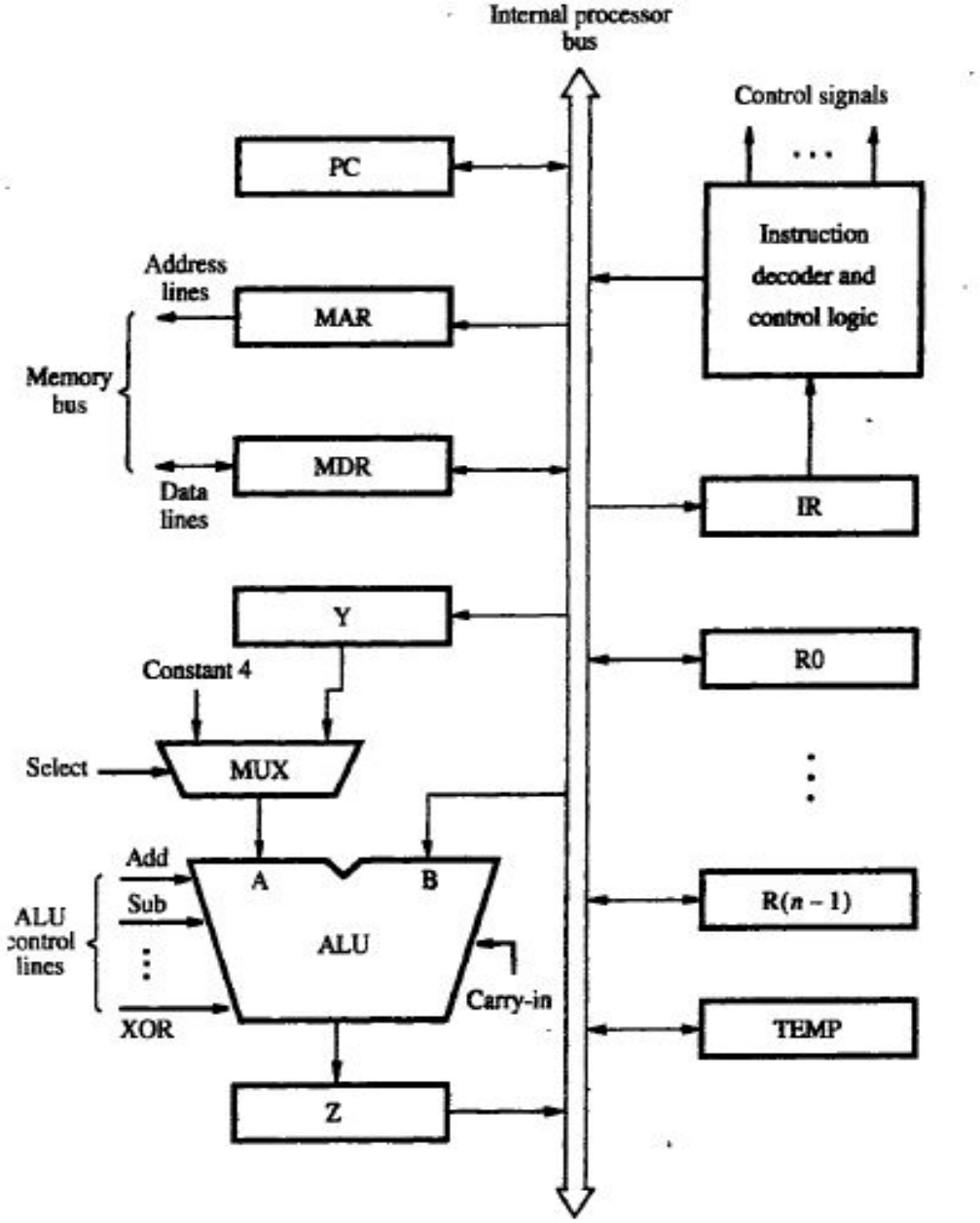


Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

- ALU
  - A input gets the operand from the output of the multiplexer (MUX).
  - B input gets the operand directly from the processor-bus.
- MUX is used to select one of the 2 inputs.
  MUX selects either output of Y or constant-value 4(which is used to increment PC content).

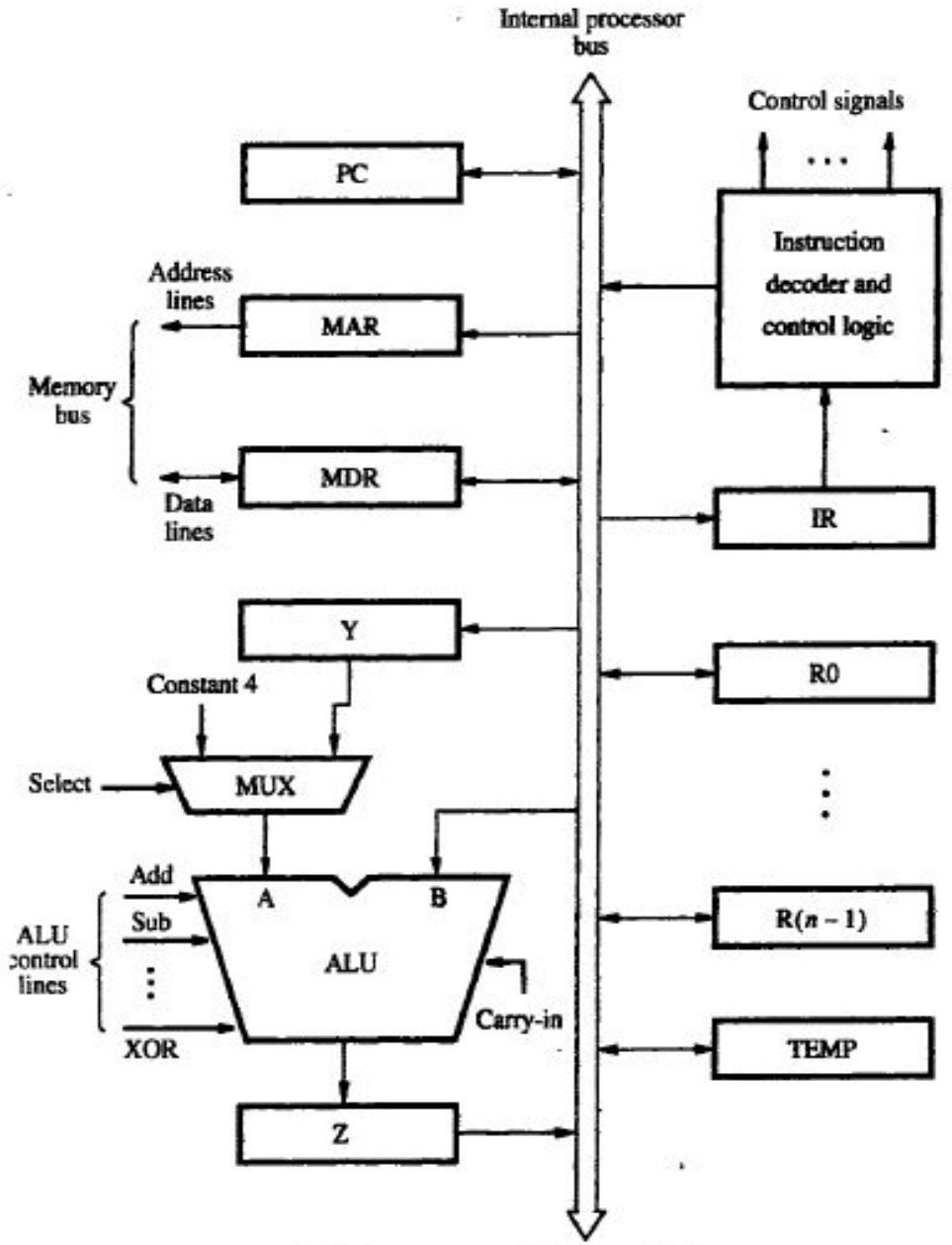


Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

- **Advantage :** Simplest and also this is very cheap to implement.
- **Disadvantage:** Only one data-word can be transferred over the bus in a clock cycle.
- **Solution:** Provide multiple internal-paths. Multiple paths allow several data-transfers to take place in parallel.



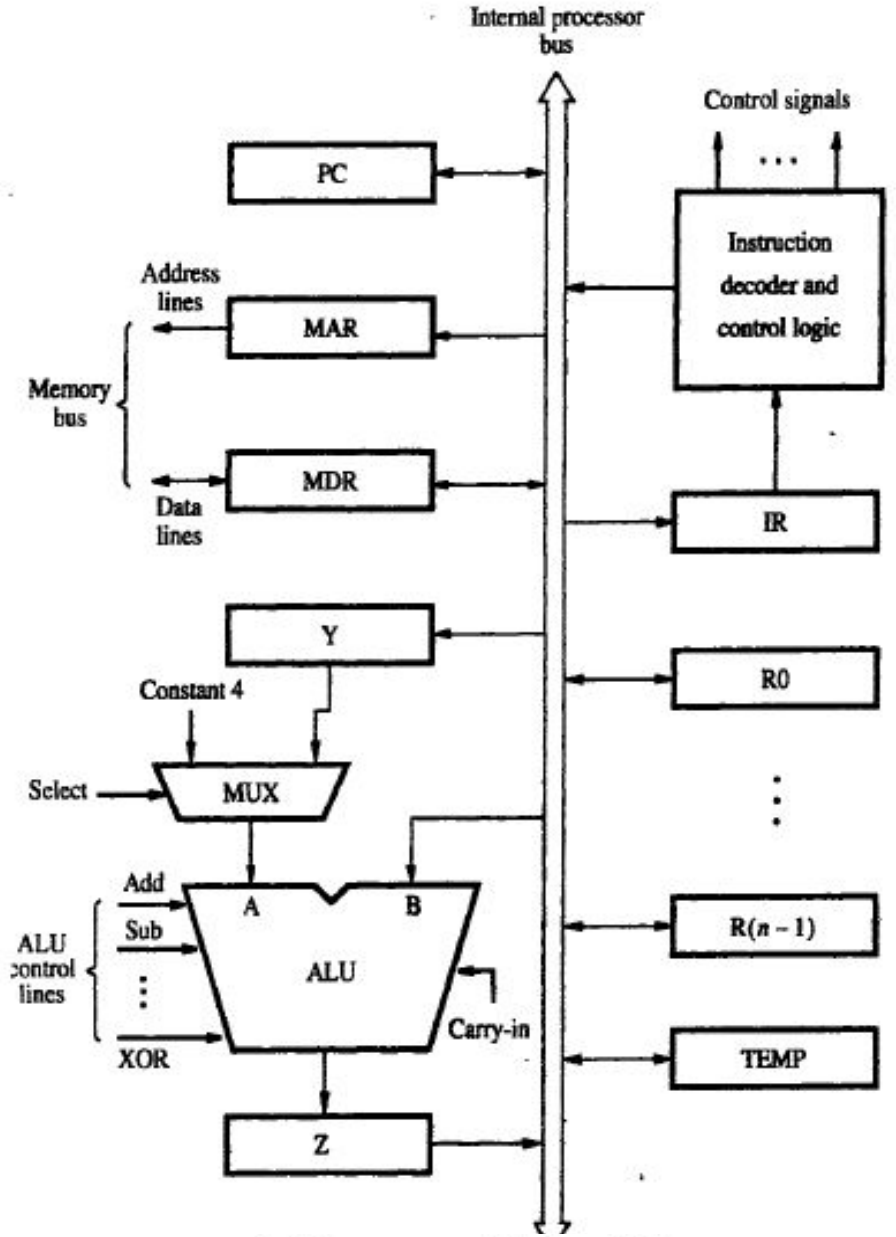
Single-bus organization of the datapath inside a processor.

Fundamental Concepts

SINGLE BUS ORGANIZATION

An instruction is executed by performing one or more of the following operations:

1. Transfer a word of data from one register to another or to the ALU.
2. Perform arithmetic or a logic operation and store the result in a register.
3. Fetch the contents of a given memory-location and load them into a register.
4. Store a word of data from a register into a given memory-location.



Single-bus organization of the datapath inside a processor.

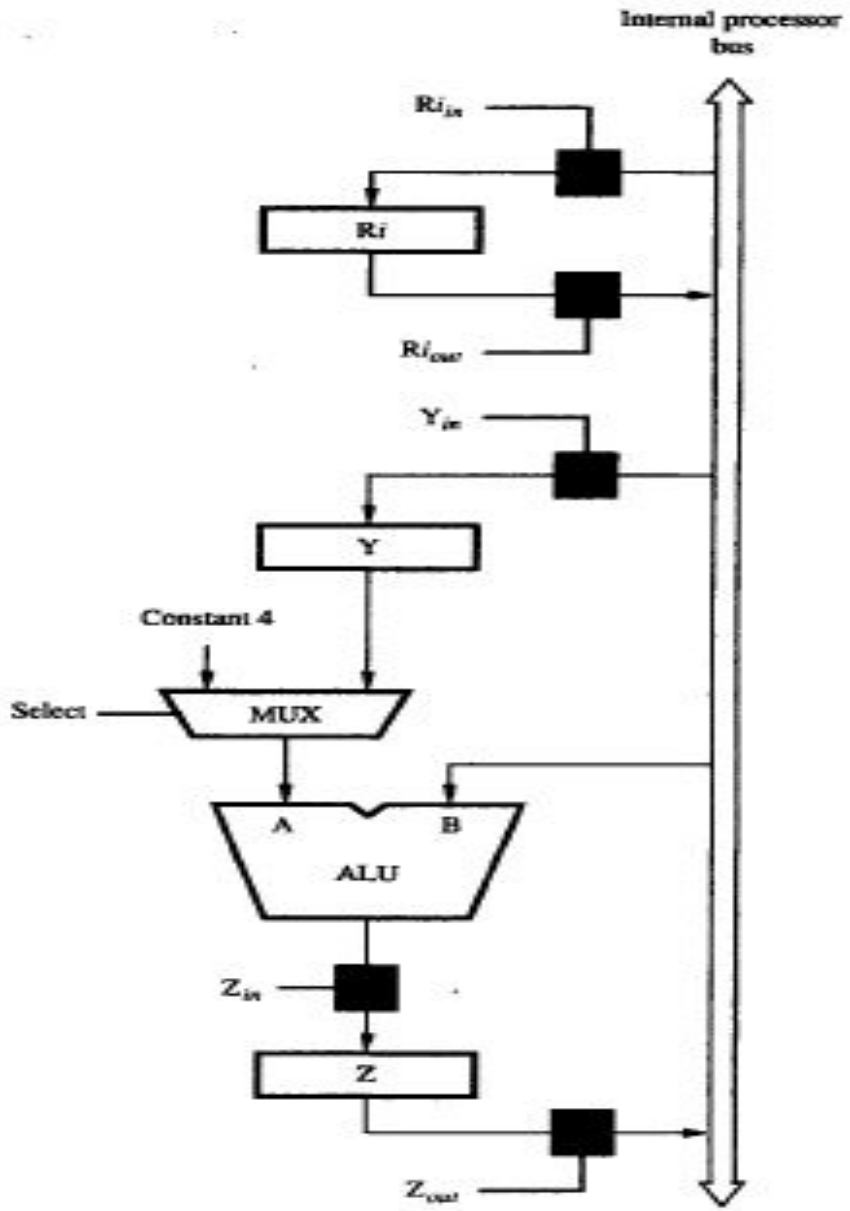
RAMAIAH
Institute of Technology

Fundamental Concepts

1. Register Transfers

Instruction execution involves a sequence of steps in which data are transferred from one register to another.

- For each register two control signals are used to place the contents of that register on the bus or to load the data on the bus into register as shown in figure.

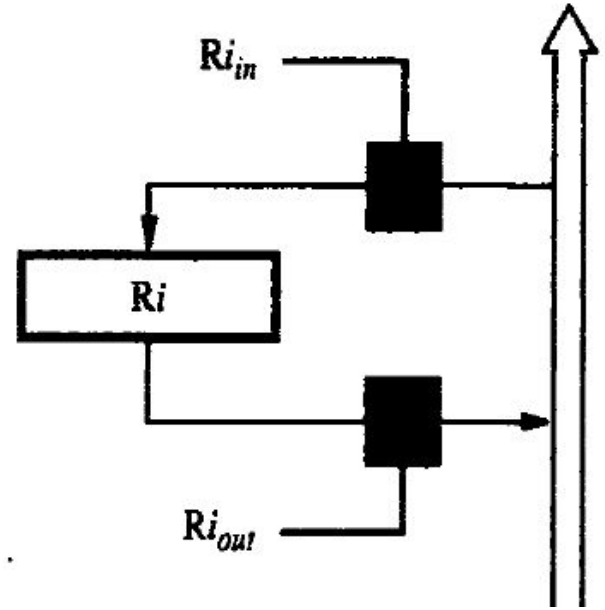


Input and output gating for the registers in Figure 7.1.

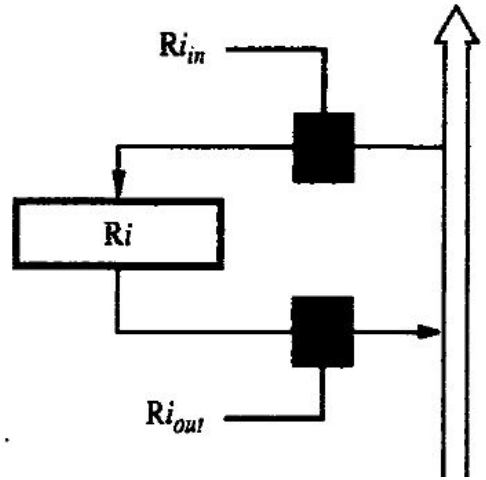
Fundamental Concepts

For each register two control signals are used

- Ri_{in} =1 ,load the data on the bus into register
- Ri_{out} =1, place the contents of that register on the bus
- Ri_{out} = 0, the bus can be used for transferring data from other registers.



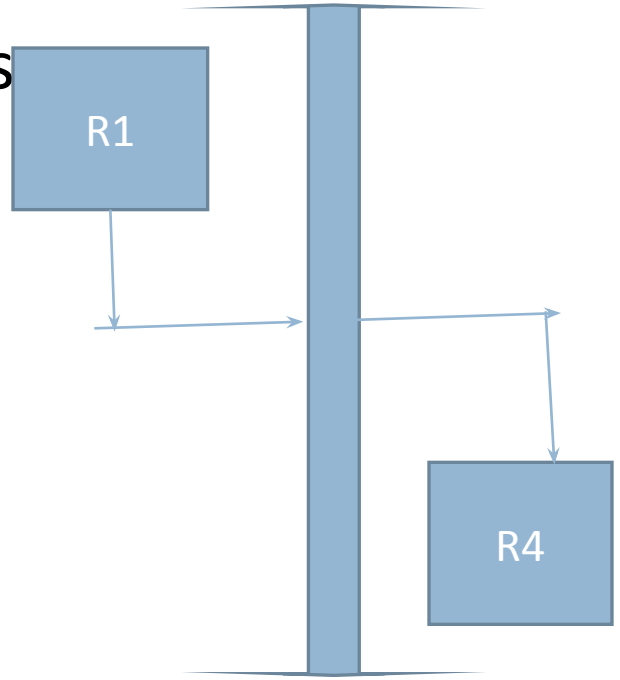
Fundamental Concepts



Suppose we wish to transfer the contents of register R1 to register R4.

1. Enable the output of registers R1 by setting R1out to 1. This places the contents of R1 on the processor bus.
2. Enable the input of register R4 by setting R4int to 1. This loads data from the processor bus into register R4.

All operations and data transfers with in the processor take place with in time periods defined by the processor clock.

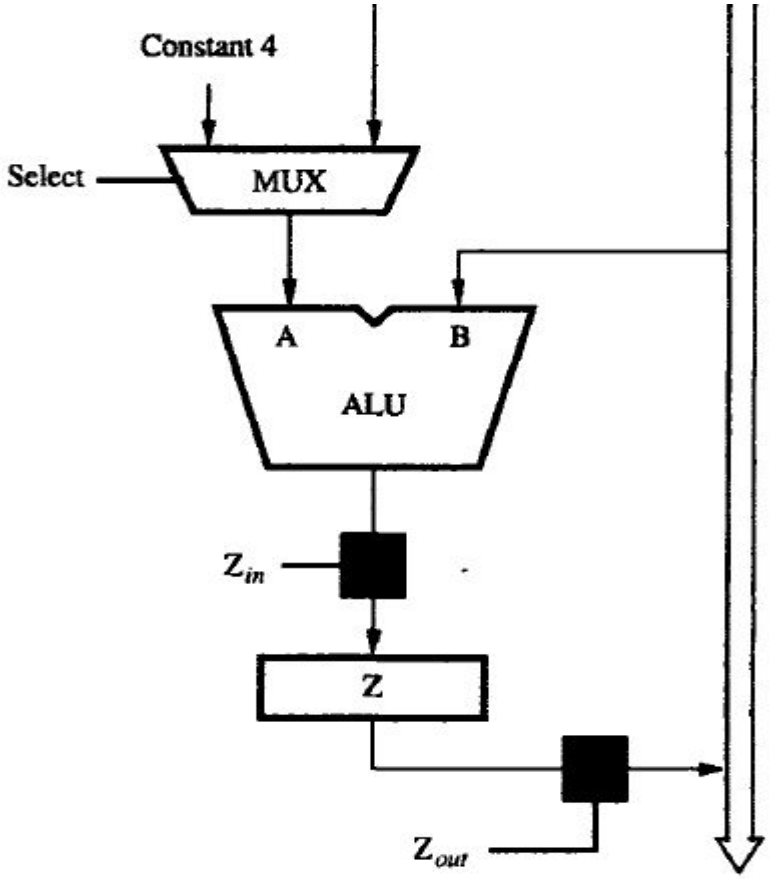


Fundamental Concepts

2.Performing an Arithmetic or Logic Operation

The ALU is a combinational circuit that has internal storage.

ALU gets the two operands from MUX and bus. result is temporarily stored in register Z.



Fundamental Concepts

2.Performing an Arithmetic or Logic Operation

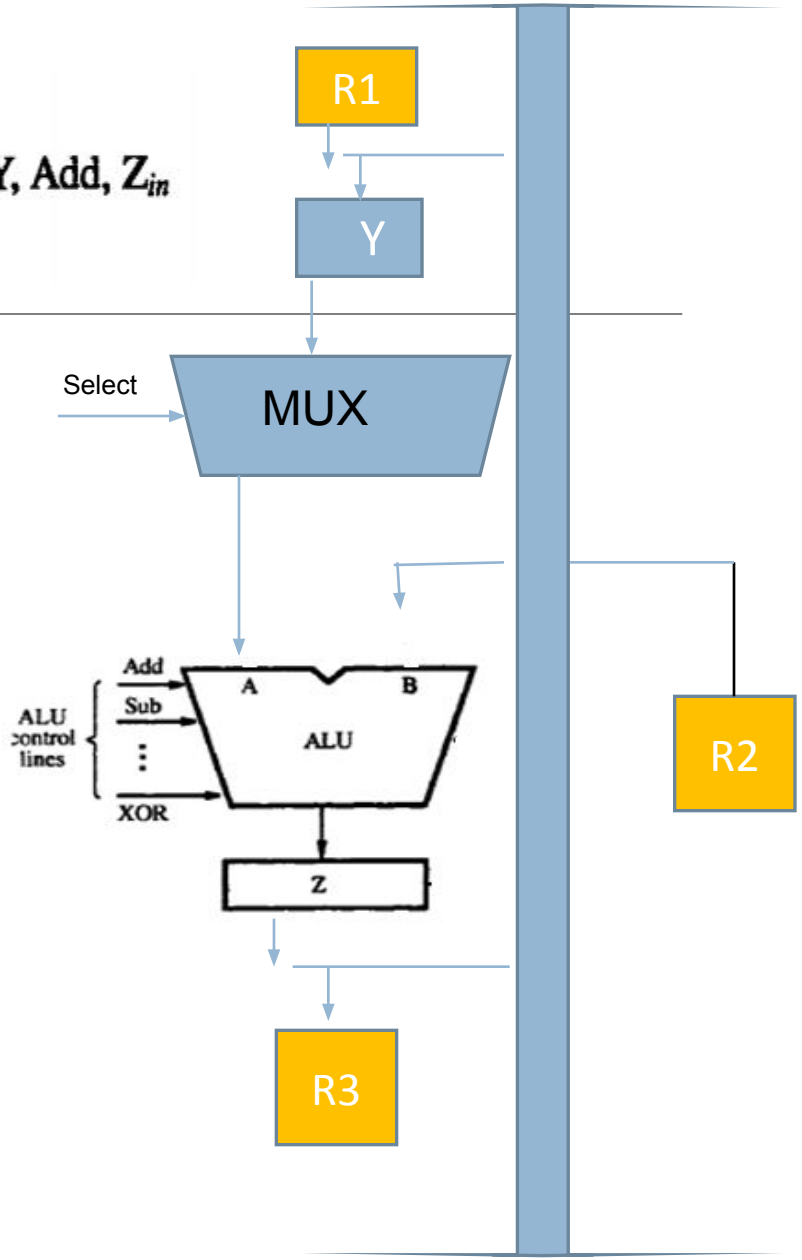
What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?

1. R1out, Yin
2. R2out, SelectY, Add, Zin
3. Zout, R3in

Fundamental Concepts

1. $R1_{out}, Y_{in}$
2. $R2_{out}$, SelectY, Add, Z_{in}
3. $Z_{out}, R3_{in}$

2.Performing an Arithmetic or Logic Operation



Step 1

Output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to Y.

Step 2

Multiplexer's select signal is set to Select Y, causing the multiplexer to gate the contents of register Y to input A of the ALU

Register R2 content is input to B

ADD line is set to 1, causing the output of the ALU to load the sum into register Z

Step 3

Content of register Z are transferred to the destination register R3.

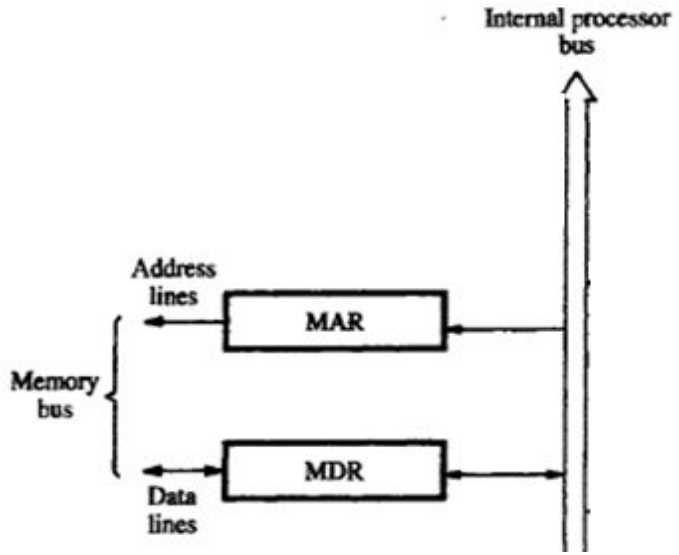
Fundamental Concepts

3.Fetching a Word from Memory

The processor has to specify the address of the memory location where this information is stored and request a Read operation.

This applies whether the information to be fetched represents an instruction in a program or an operand specified by an instruction.

The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus.



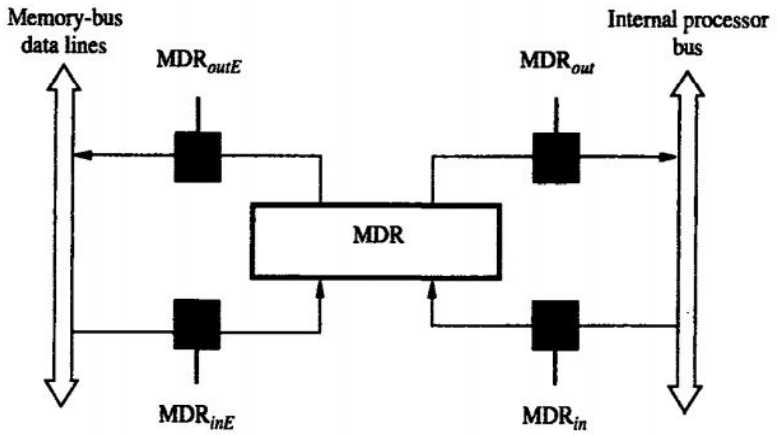
Fundamental Concepts

3.Fetching a Word from Memory

At the same time , the processor uses the control lines of the memory bus to indicate that a Read operation is needed.

When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.

Processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).



Connection and control signals for register MDR.

Fundamental Concepts

3.Fetching a Word from Memory

Example: <u>Move (R1), R2</u>

MAR ← [R1]

Start a Read operation on the memory bus Wait for the MFC response from the memory Load MDR from the memory bus

R2 ← [MDR]

Fundamental Concepts

4. Storing a word in Memory

The desired address is loaded into MAR.

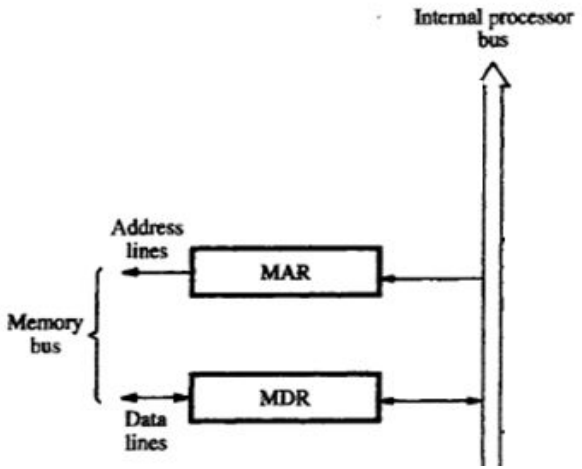
Then , the data to be written are loaded into MDR, and a write command is issued.

Example: **Move R2,(R1)** requires the following steps

1 $R1_{out}$,MAR_{in}

2.$R2_{out}$,MDR_{in},Write

3.MDR_{outE},WMFC



RAMAIAH
Institute of Technology

UNIT 2 – Arithmetic unit & Processing unit

- Fundamental concepts
- Execution of complete instruction
- Multiple bus organization
- Hardwired control
- Microprogrammed control

Execution of a Complete Instruction

Sequence of elementary operations required to execute one instruction using Single Bus organization

2 Phases:

1. Fetch Phase(Step 1 to Step 3)
2. Execution Phase(Step 4 to Step 7)

Example : Add (R3), R1

1. Fetch the instruction
2. Fetch the first operand (the contents of the memory location pointed to by R3)
3. Perform the addition
4. Load the result into R1

Execution of a Complete Instruction

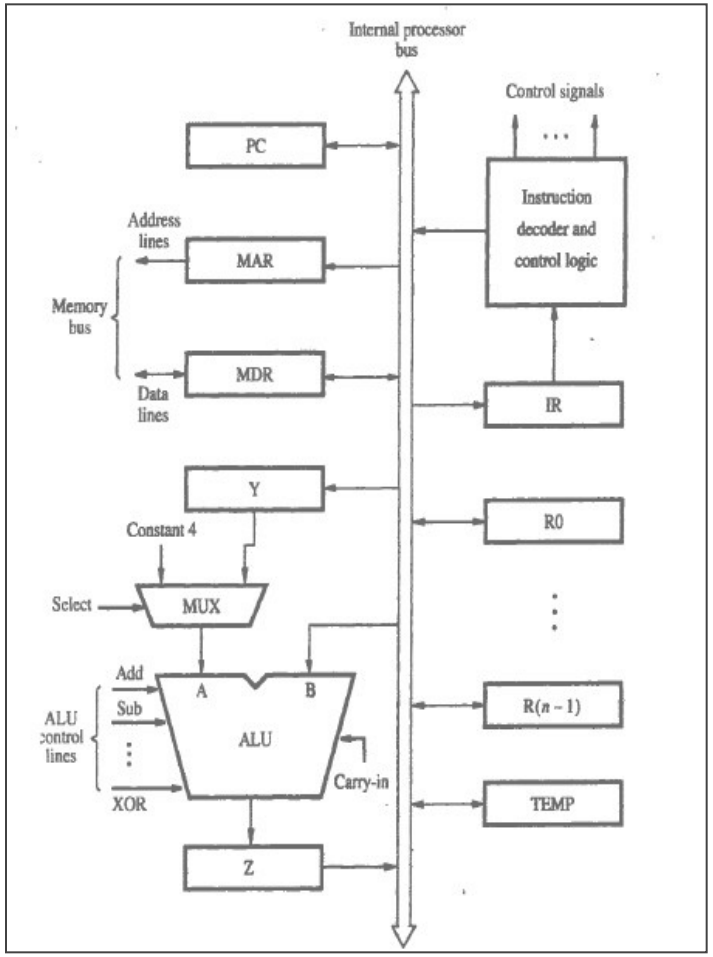
Step1: : Add (R3), R1

The instruction-fetch operation is initiated by

- Load contents of PC into MAR
- Send a Read request to memory
- The Select signal is set to Select4, which causes the Mux to select constant 4.
- This value is added to operand at input B (PC content)
- Result is stored in Z.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



Execution of a Complete Instruction

Step2:

- Updated value in Z is moved to PC.

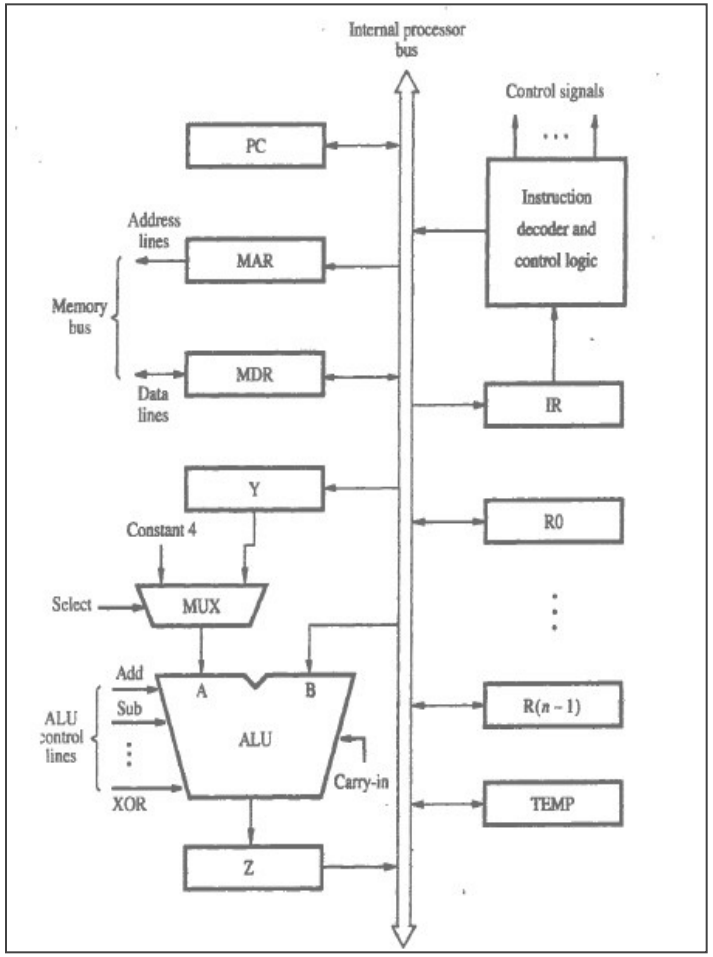
  This completes the PC increment operation

  and PC will now point to next instruction,

  While waiting for memory to respond

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



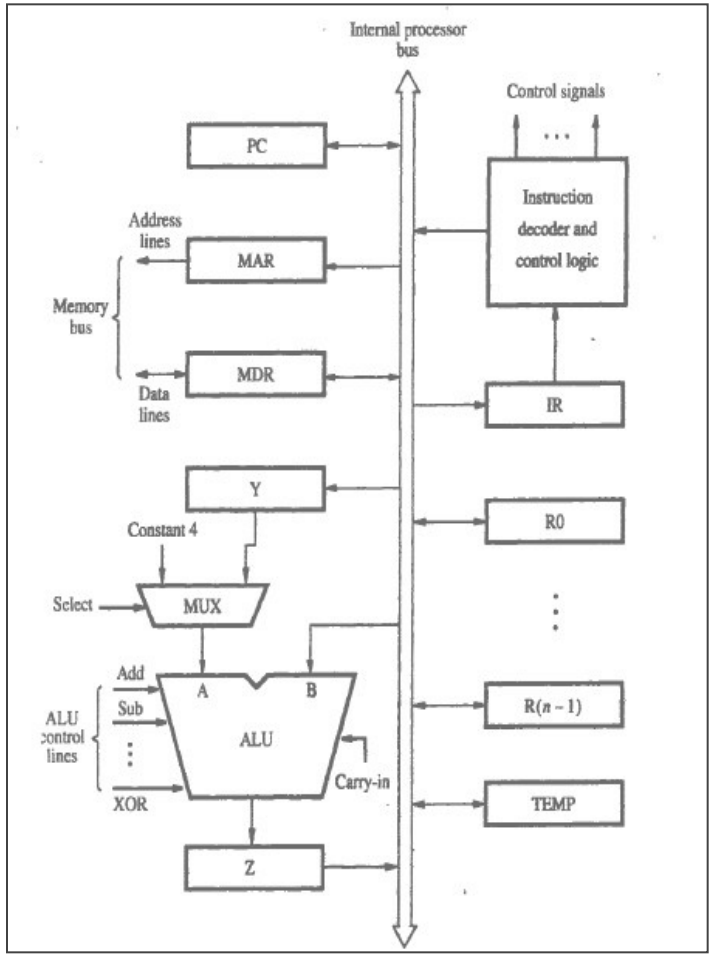
Execution of a Complete Instruction

Step3:

- Fetched instruction is moved into MDR and then to IR.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



Execution of a Complete Instruction

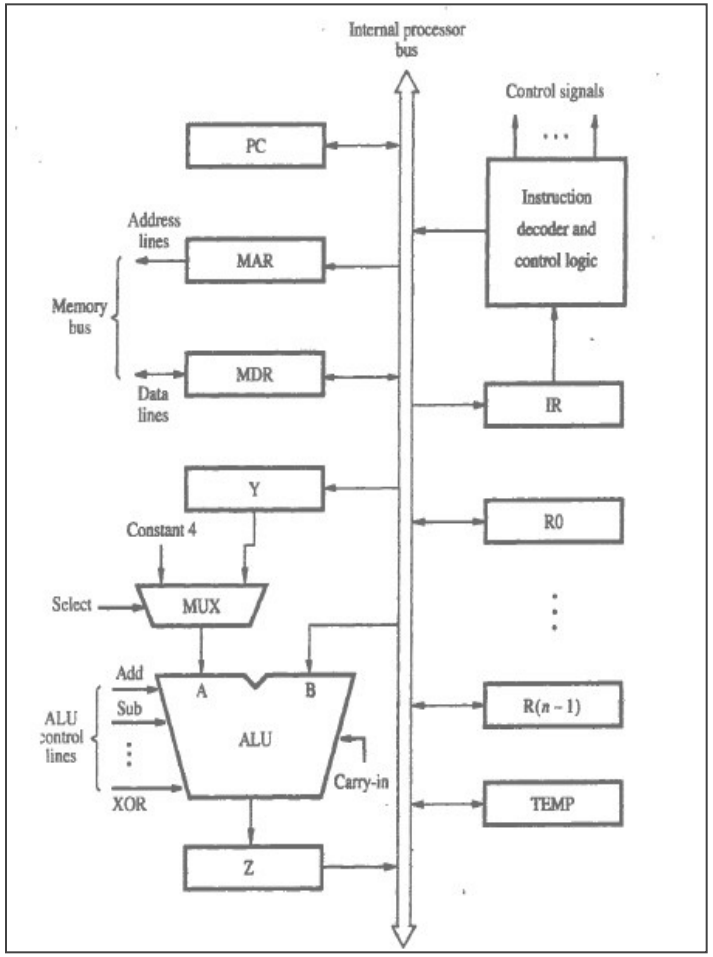
Step 4:

- Instruction decoder interprets the contents of the IR. This enables the control circuitry to activate the control-signals for steps 4 through 7.

- Contents of R3 are loaded into MAR & a memory read signal is issued.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



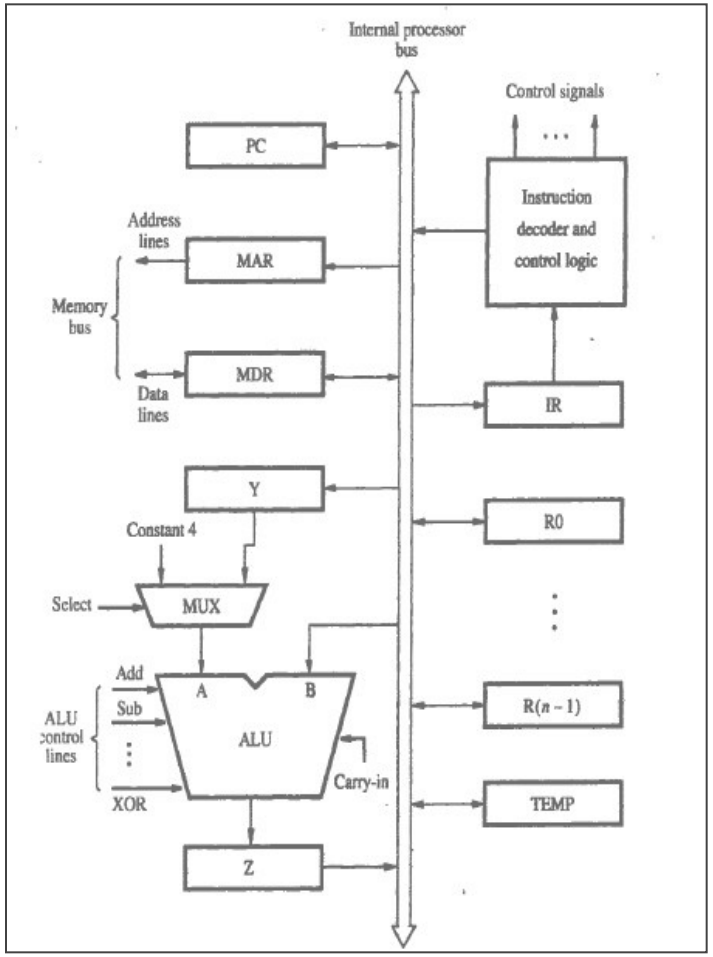
Execution of a Complete Instruction

Step 5:

Contents of R1 are transferred to Y to prepare for addition.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



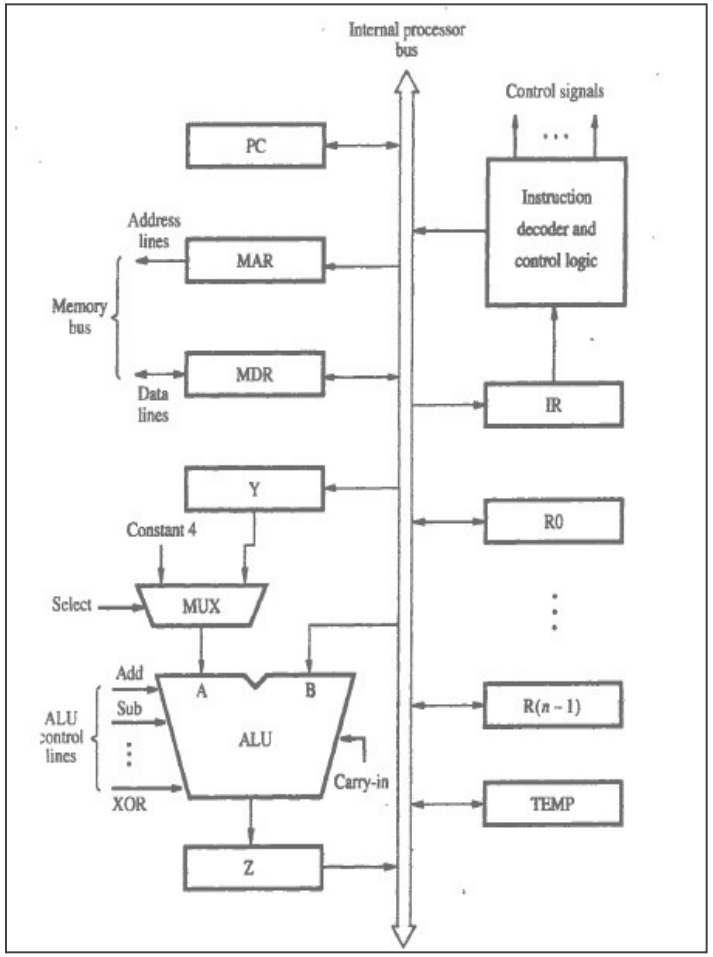
Execution of a Complete Instruction

Step 6:

- When Read operation is completed, memory-operand is available in MDR, and the addition is performed.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



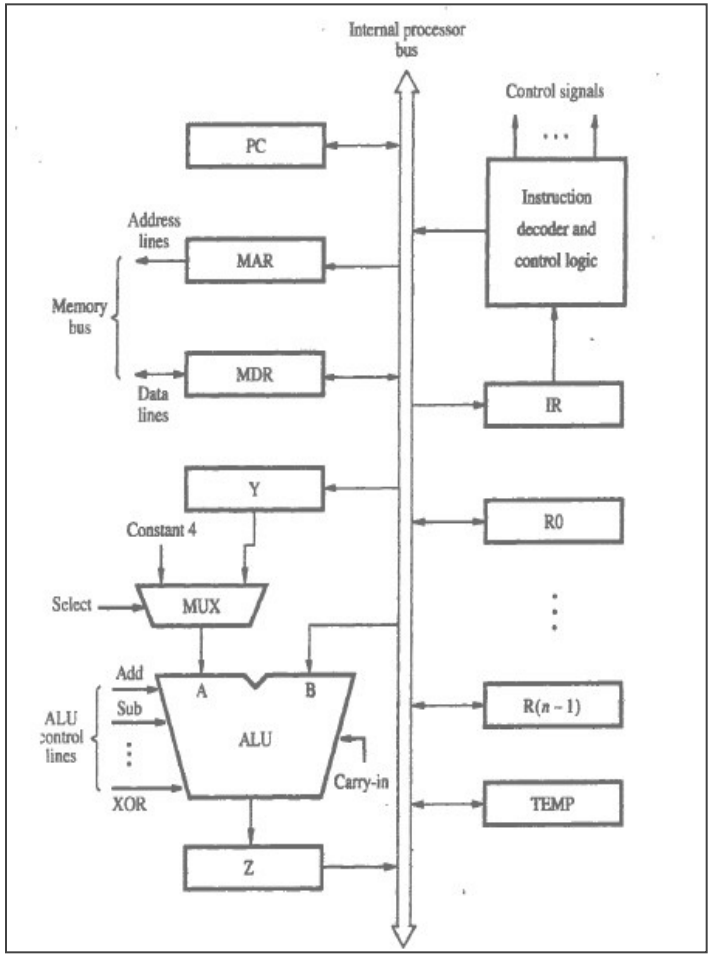
Execution of a Complete Instruction

Step 7:

- Sum is stored in Z, then transferred to R1.
- The End signal causes a new instruction fetch cycle to begin by returning to step1.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



RAMAIAH Institute of Technology

Execution of a Complete Instruction

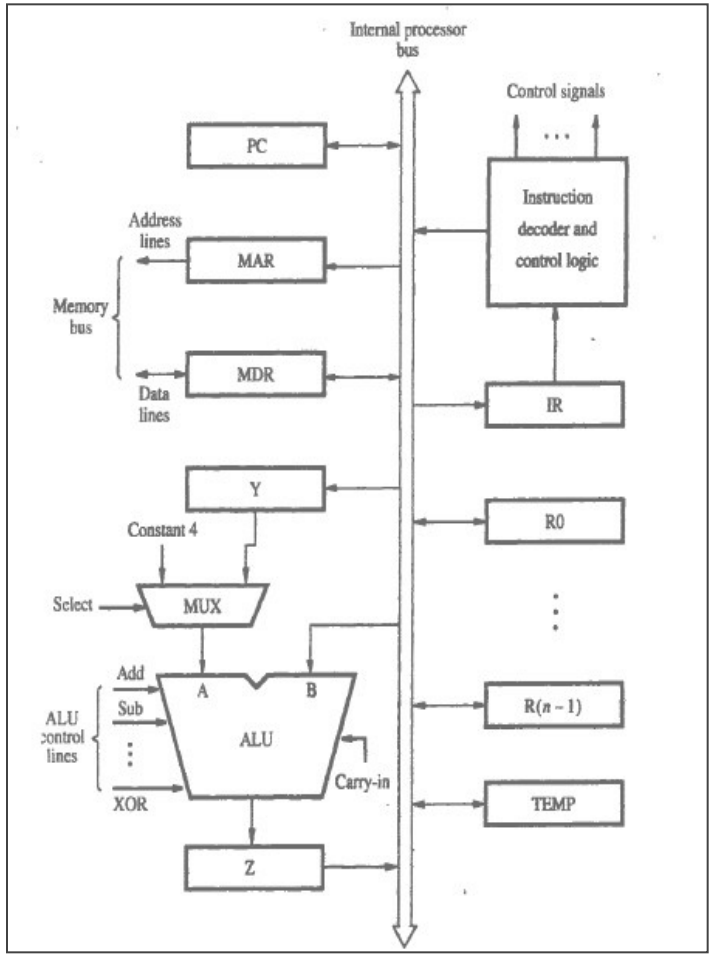
Execution of Branch Instructions

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Control sequence for an unconditional Branch instruction.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, 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



RAMAIAH Institute of Technology

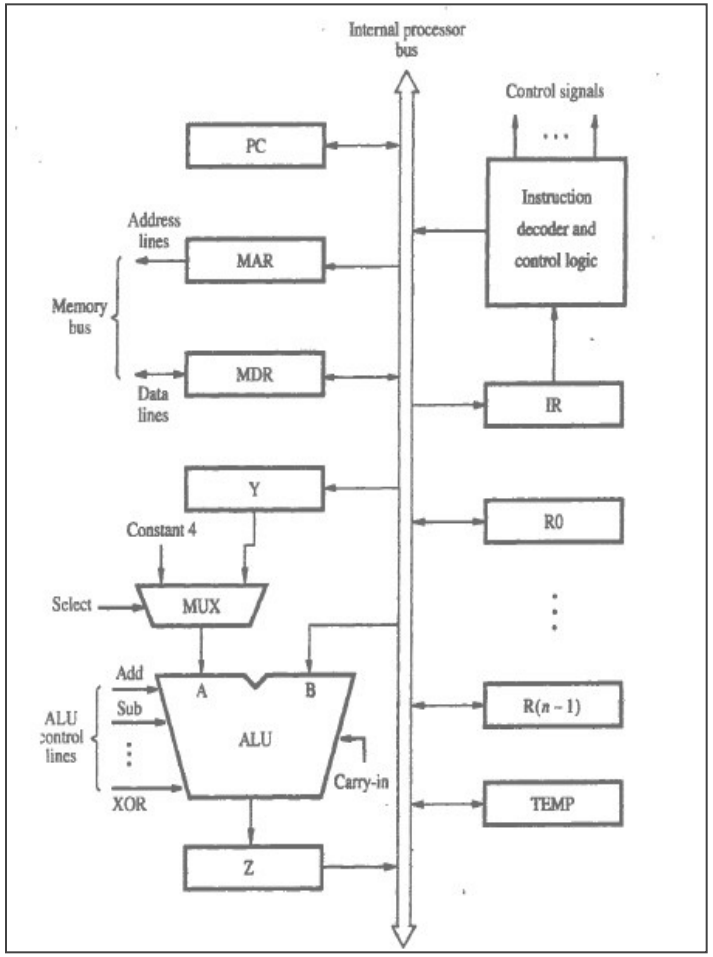
Execution of a Complete Instruction

Execution of Branch Instructions

- Step 1-3 : The processing starts & the fetch phase ends in step3.
- Step 4 :The offset-value is extracted from IR by instruction-decoding circuit.
- Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.
- Step 5: Result, which is the branch-address, is loaded into the PC.

Control sequence for an unconditional Branch instruction.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, 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



RAMAIAH
Institute of Technology

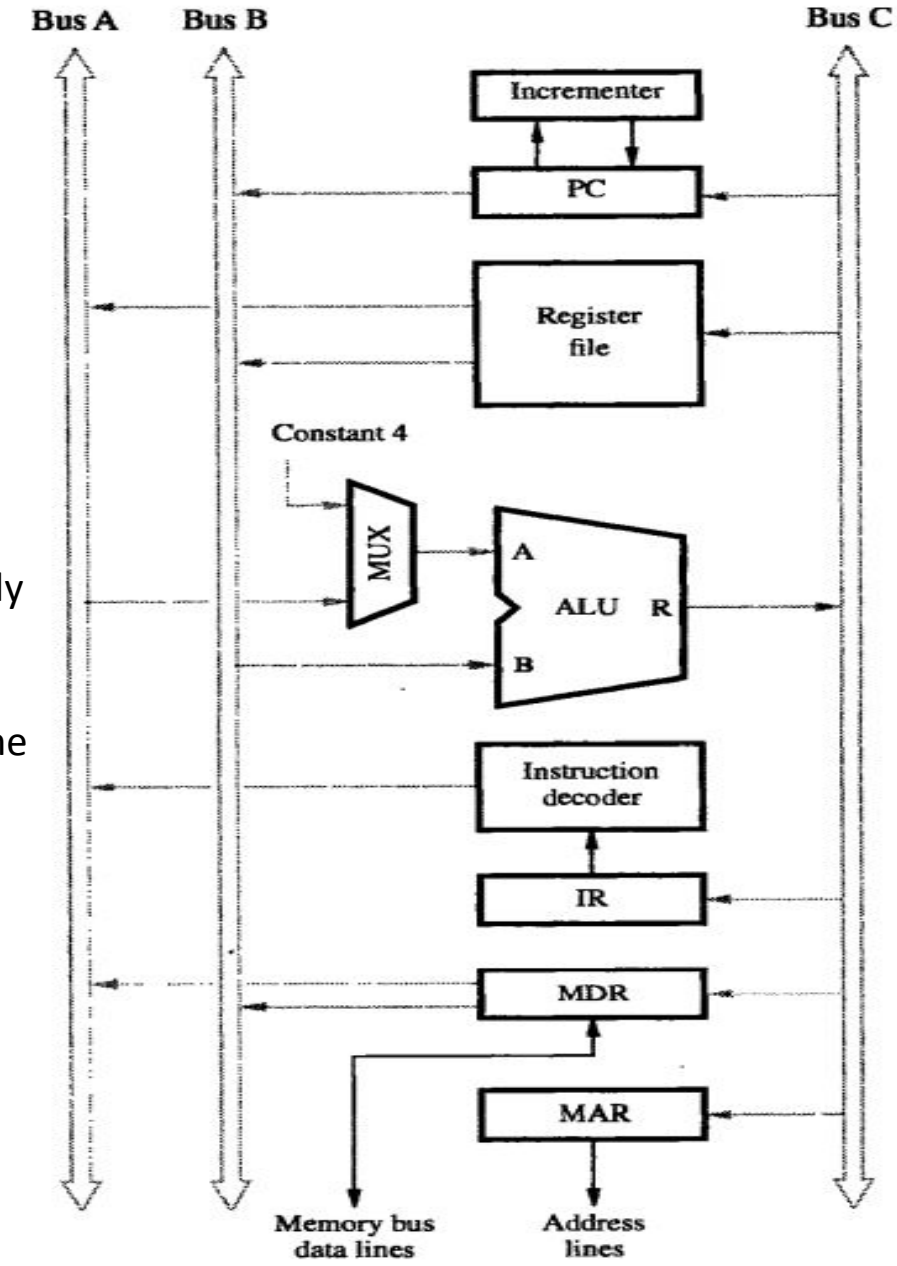
UNIT 2 – Arithmetic unit & Processing unit

- Fundamental concepts
- Execution of complete instruction
- Multiple bus organization
- Hardwired control
- Microprogrammed control

RAMAIAH
Institute of Technology

Multiple-Bus Organization

- Three buses are used to connect registers and the ALU of the processor.
- All general-purpose registers are grouped into a single block called the Register File.
- Register-file has 3 ports:
- Two output-ports allow the contents of 2 different registers to be simultaneously placed on buses A & B.
- Third input-port allows data on bus C to be loaded into a third register during the same clock-cycle.
- Buses A and B are used to transfer source-operands to A & B inputs of ALU.
- The result is transferred to destination over bus C.
- Incrementer Unit is used to increment PC by 4.



Multiple-Bus Organization

Add R4, R5 ,R6

Step 1: Contents of PC are

- passed through ALU using R=B control-signal
- loaded into MAR to start memory Read operation.
- At the same time, PC is incremented by 4.

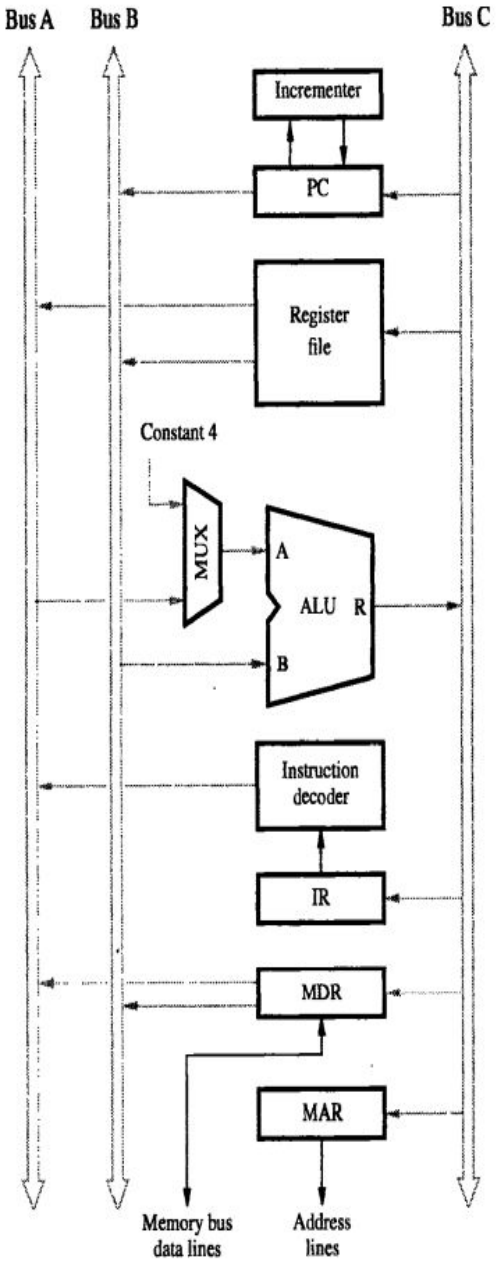
Step2 : Processor waits for MFC signal from memory.

Step3: Processor loads requested data into MDR, and then transfers them to IR.

Step4: The instruction is decoded and add operation takes place in a single step.

Control sequence for the instruction Add R4,R5,R6 for the three-bus organization

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End



RAMAIAH
Institute of Technology

UNIT 2 – Arithmetic unit & Processing unit

- Fundamental concepts
- Execution of complete instruction
- Multiple bus organization
- Hardwired control
- Microprogrammed control

Design of Control Unit

- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.

The Control Unit is classified into two major categories:

- Hardwired Control
- Microprogrammed Control

Hardwired control

- The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.
- The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs.
- The outputs of the state machine are the control signals.

Hardwired control

Add (R3),R1

This instructions adds the content of a memory location pointed by R3 to register R1

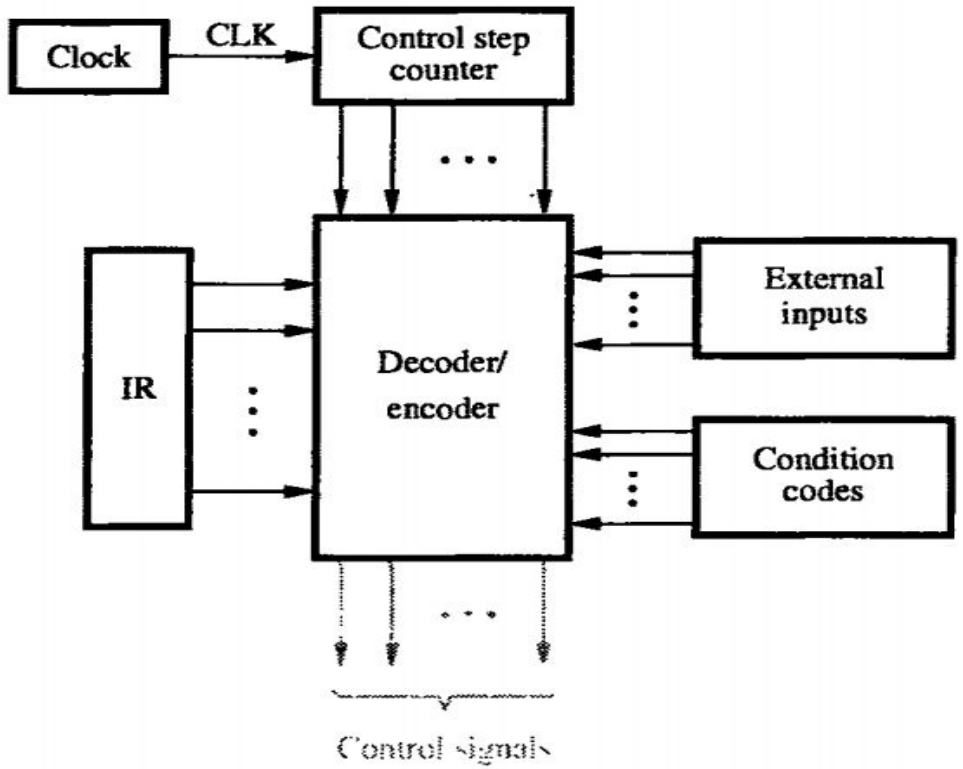
Each step in this sequence is completed in one clock period.

Counter tracks the control steps

Each state or count of this counter correspond to one control step

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End



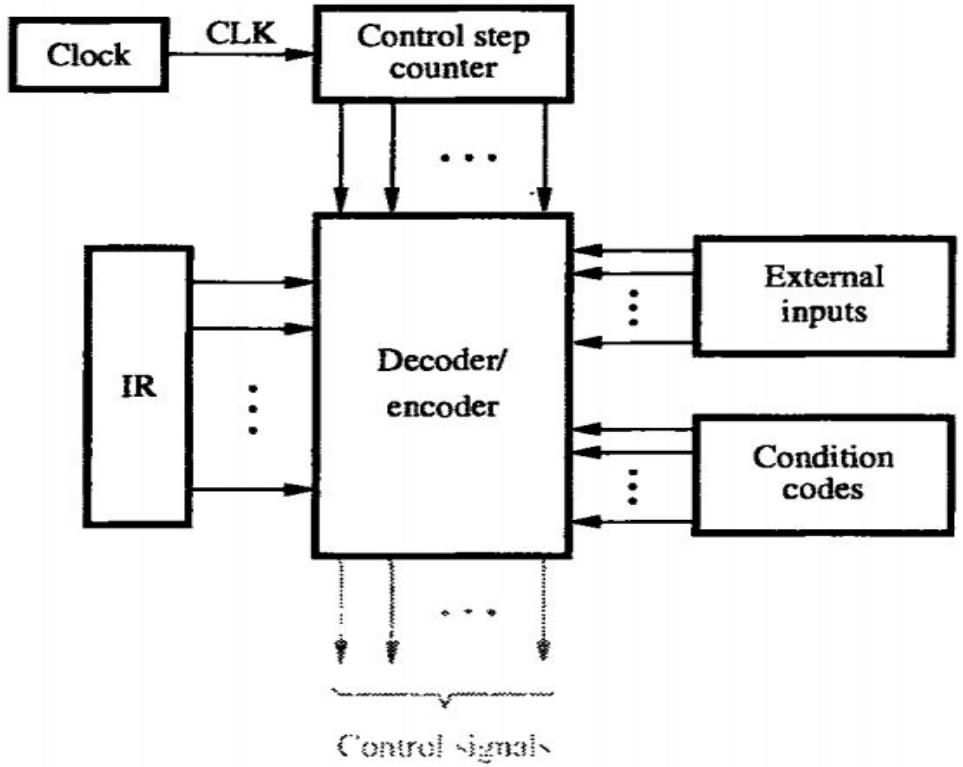
Hardwired control

Control unit depends on

1. Contents of control step counter
2. Contents of IR
3. Contents of Condition codes
4. External input such as MFC/interrupt request

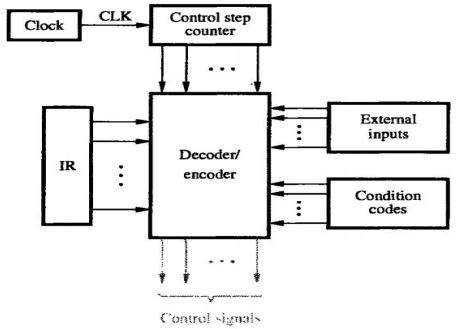
Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

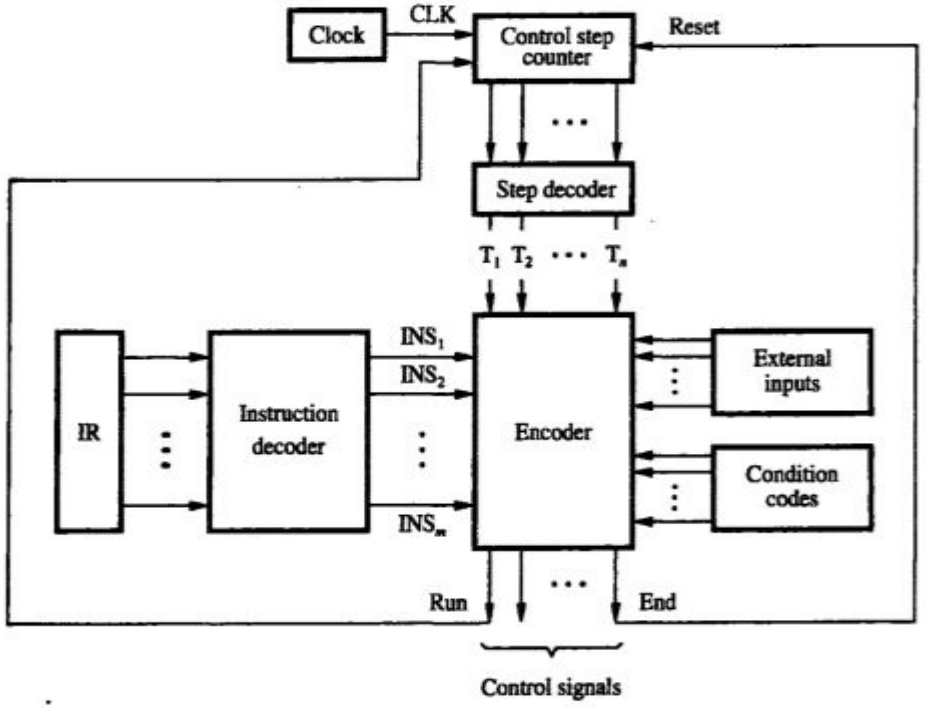


RAMAIAH Institute of Technology

Hardwired control

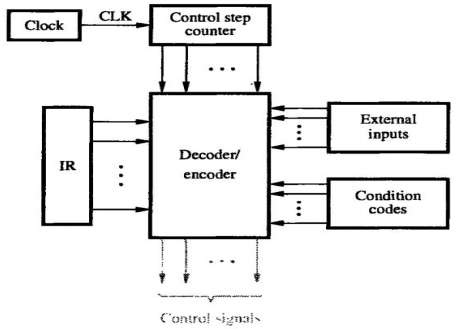


- Decoder/Encoder Block is a combinational-circuit that generates required control-outputs depending on state of all its inputs.
  Instruction Decoder(IR) decodes the instruction loaded in the IR, then generates separate output-lines for each machine instruction.
  Step-Decoder provides a separate signal line for each step in the control sequence.
  Encoder gets the input from instruction decoder, step decoder, external inputs and condition codes and generate individual control-signals



Separation of the decoding and encoding functions.

Hardwired control



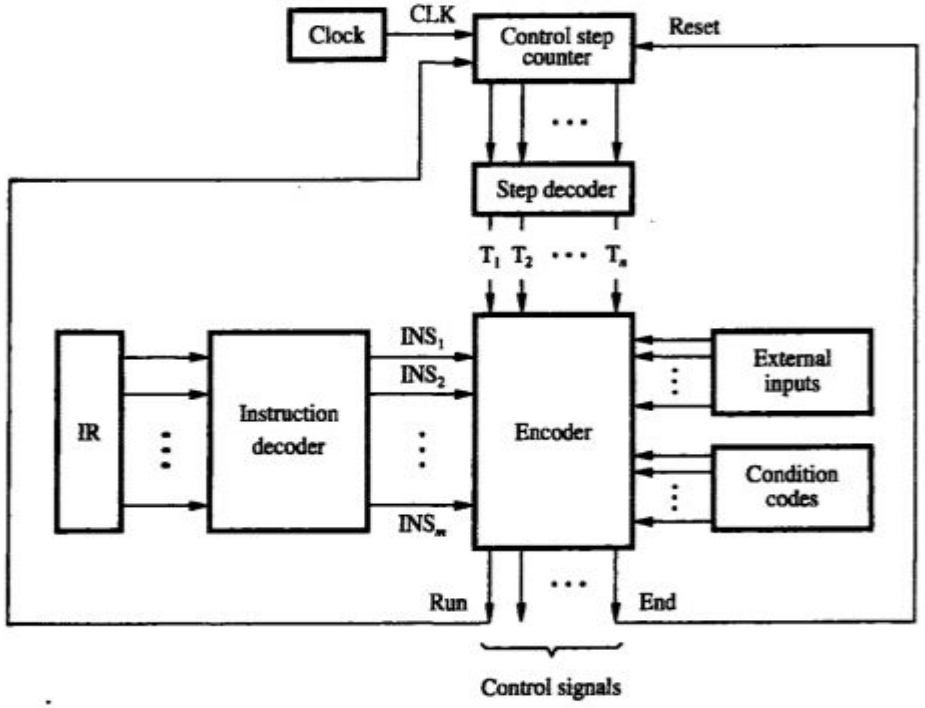
Advantage: Can operate at high speed.

Disadvantages:

Since no. of instructions/control-lines is often in hundreds, the complexity of control unit is very high.

It is costly and difficult to design.

The control unit is inflexible because it is difficult to change the design.

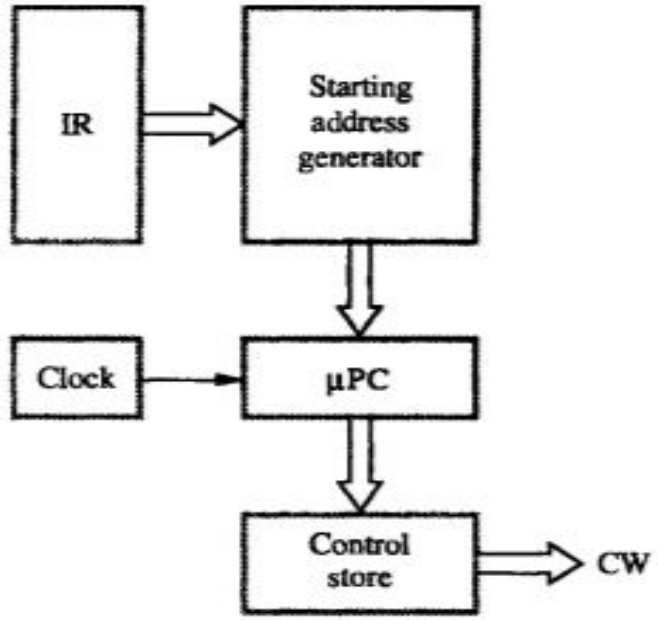


Separation of the decoding and encoding functions.

Micro programmed Control

- Control signals are implemented with the help of a program
- Microprogrammed control unit is a software/program
- A control unit with its binary control values stored as words in memory is called as microprogrammed control.

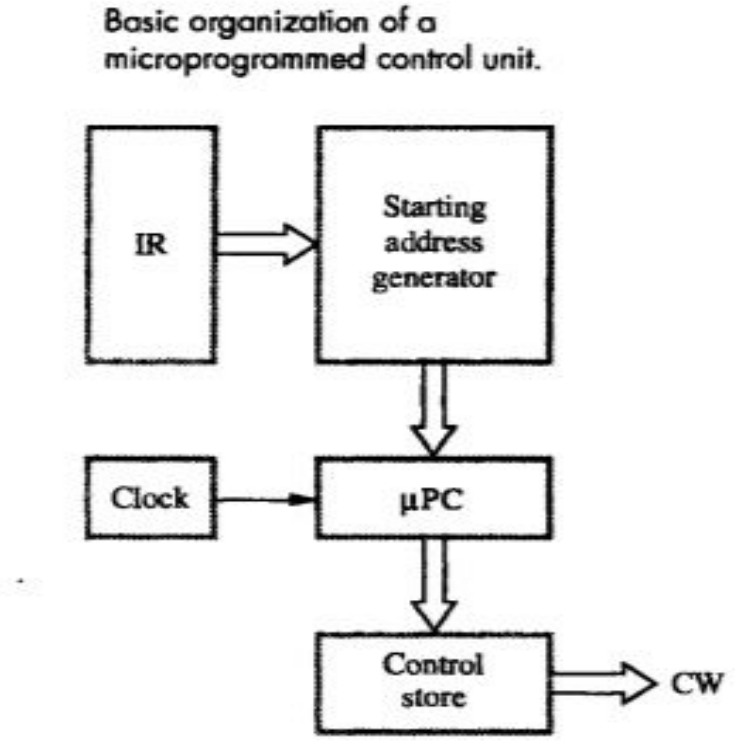
Basic organization of a microprogrammed control unit.



Micro programmed Control

Every time instruction is loaded in IR , the output of Starting address generator is loaded into the µPC

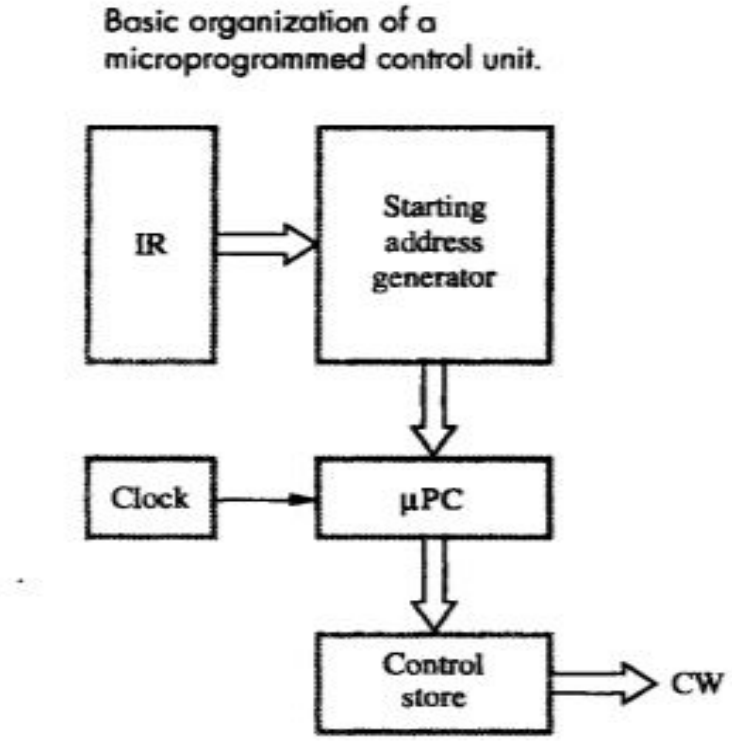
The µPC is automatically incremented by the clock, causing successive microinstruction to be read from control store



Micro programmed Control

Control Memory stores the microprogram, Microprogram is a collections of microinstructions and each misconstructions is contains one or more microoperations.

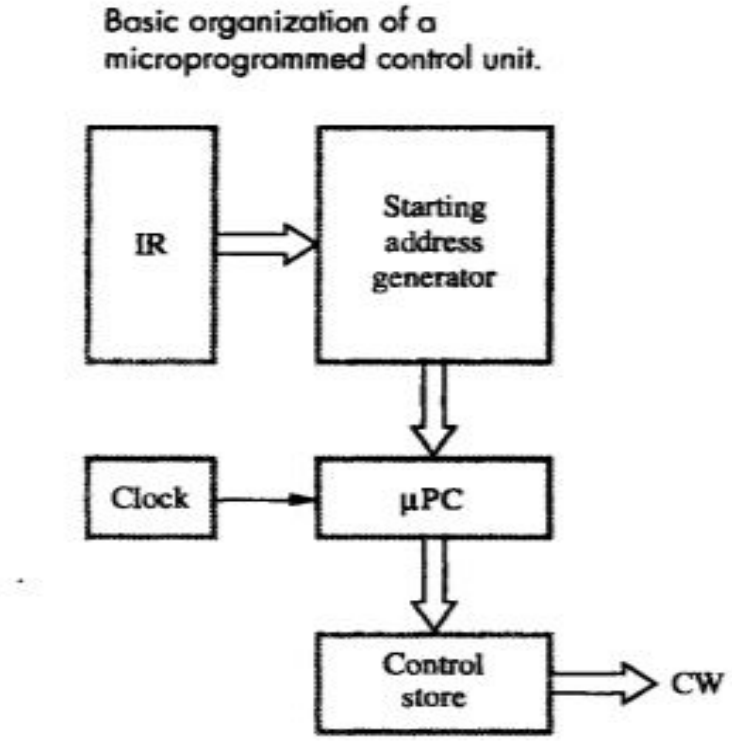
Control Store contains the microinstruction that is to be executed



Micro programmed Control

Control unit generates the control signal for any instruction by sequentially reading the CW of the corresponding micro routine from the control store

Control Word : A control word is a word whose individual bits represent various control signals.



RAMAIAH Institute of Technology

Micro programmed Control - Example

Control signals are generated by a program similar to machine language programs.

Control sequence for execution of the instruction Add (R3),R1.

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	..
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Questions

1. Explain the working of a hardwired control unit with a diagram

1. Write the sequence of control steps required for the single bus structure for the following instruction Add (R2),R1.

1. Draw and explain the Single bus organization of a Processor.

4. Draw and explain the multi-bus organization of a Processor.


Thank You!