Data Path Design

UNIT 4

Dr. Sumita Nainan, MPSTME,NMIMS University

- **Data path Design**
- IEEE 754 data format, IEEE 754 data format numerical
- Design of serial and parallel adder and subtractor
- Booth's algorithm
- ALU -Combinational and sequential ALU.
- Block diagrams of high speed adders multipliers
- Block diagrams of high speed multipliers
- Overview of math coprocessor.

Adders and Subtractor

1. Half-adder A logic circuit for the addition of two one-bit numbers is referred to as an *half-adder*. The addition process is illustrated in Section 2.5 and is reproduced in truth table form in Table 5.11. Here, A and B are the two inputs and S (SUM) and C (CARRY) are the two outputs.

Table 5.11 ***Truth Table of an Half-adder***

Inputs		Outputs	
A	***B***	***S***	***C***
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the truth table, we obtain the logical expressions for S and C outputs as

$$= \bar{A}B + A\bar{B} = A \oplus B \qquad (5.34a)$$

$$C = AB$$

Adders and Subtractor

The realisation of an half-adder using gates is shown in Fig. 5.19.

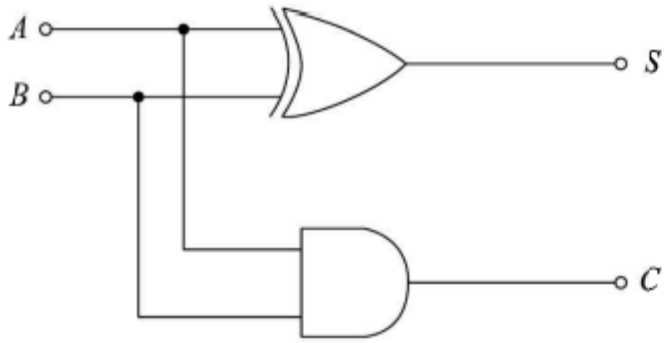


Fig. 5.19 ***Realisation of an Half-adder***

2. Full-adder An half-adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multibit addition is performed.

For this purpose, a third input terminal is added and this circuit is used to add A_n, B_n, and C_{n-1}, where A_n and B_n are the nth order bits of the numbers A and B respectively and C_{n-1} is the carry generated from the addition of $(n-1)$th order bits. This circuit is referred to as *full-adder* and its truth table is given in Table 5.12.

The K-maps for the outputs S_n and C_n are given in Fig. 5.20 and the minimised expressions are given by Eq. (5.35).

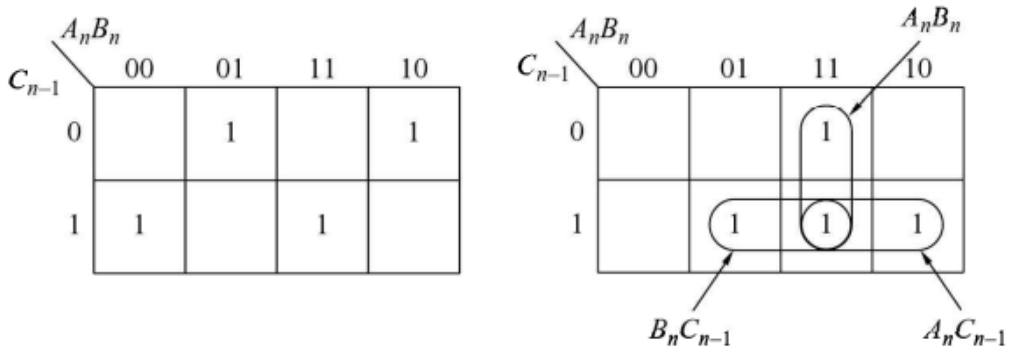
$$S_n = \bar{A}_n\, B_n\, \bar{C}_{n-1} + \bar{A}_n\, \bar{B}_n\, C_{n-1} + A_n\, \bar{B}_n\, \bar{C}_{n-1} + A_n\, B_n\, C_{n-1} \qquad (5.35a)$$

$$C_n = A_n\, B_n + B_n\, C_{n-1} + A_n\, C_{n-1} \qquad (5.35b)$$

Full Adder

Table 5.12 ***Truth Table of a Full-adder***

Inputs			Outputs	
A_n	B_n	C_{n-1}	S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



(a) (b)

Fig. 5.20 ***K-maps for (a) S_n (b) C_n***

Full Adder

The NAND–NAND realisations are given in Fig. 5.21.

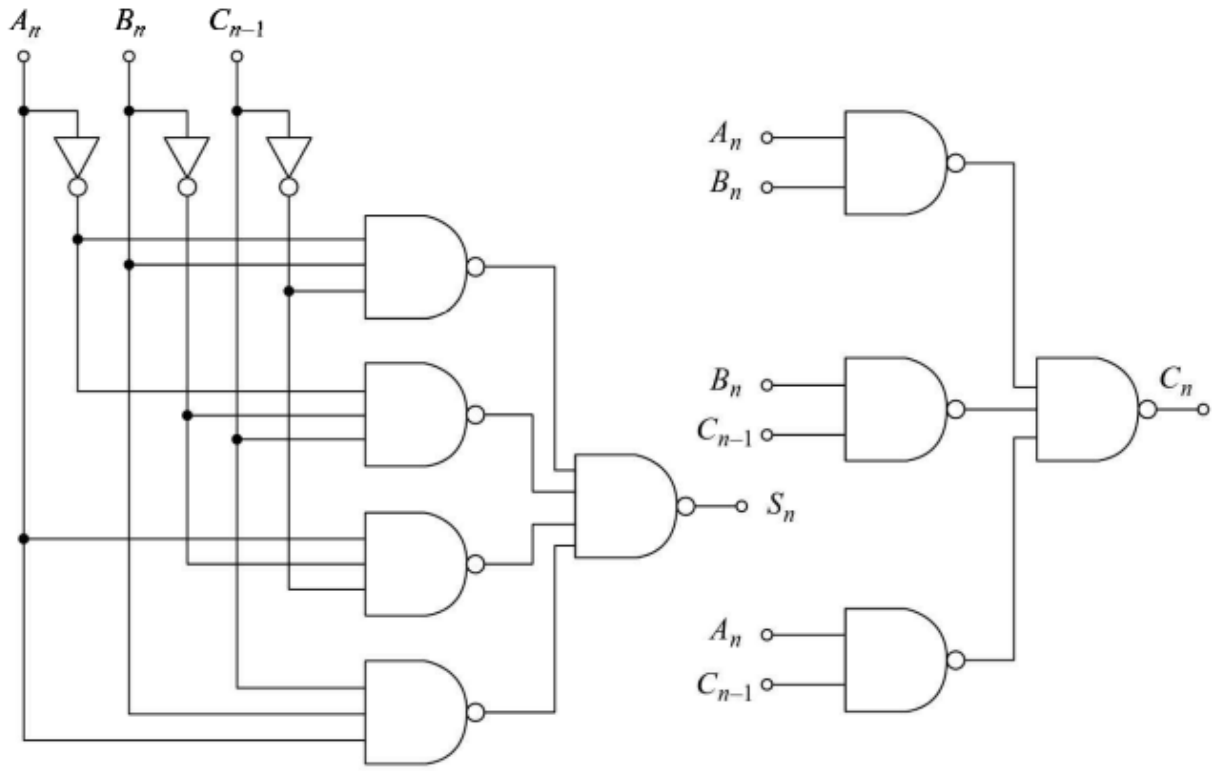


Fig. 5.21 **NAND–NAND *Realisation of (a) S_n (b) C_n***

Half-Subtractor

3. Half-subtractor A logic circuit for the subtraction of B (subtrahend) from A (minuend) where A and B are 1-bit numbers is referred to as a *half-subtractor*. The subtraction process is illustrated in Section 2.5 and is reproduced in truth table form in Table 5.13. Here, A and B are the two inputs and D (difference) and C (borrow) are the two outputs.

Table 5.13 ***Truth Table of a Half-subtractor***

Inputs		Outputs	
A	B	D	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

From the truth table, the logical expressions for D and C are obtained as

$$D = \bar{A}B + A\bar{B} = A \oplus B \quad (5.36a)$$

$$C = \bar{A}B \quad (5.36b)$$

Full-Subtractor

4. Full-subtractor Just like a full-adder, we require a *full-subtractor* circuit for performing multibit subtraction wherein a borrow from the previous bit position may also be there. A full-subtractor will have three inputs, A_n (minuend), B_n (subtrahend) and C_{n-1} (borrow from the previous stage) and two outputs, D_n (difference) and C_n (borrow). Its truth table is given in Table 5.14. The K-map for the output D_n is exactly same as the K-map for S_n of the adder circuit and therefore, its realisation is same as given in Fig. 5.21*a*.

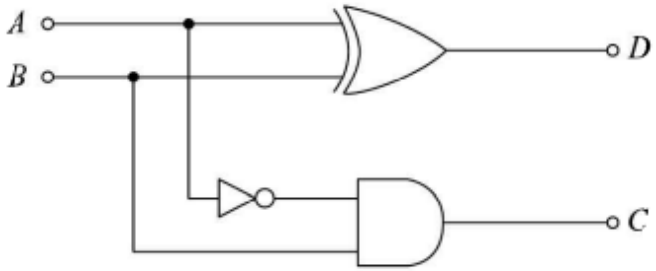


Fig. 5.22 ***Realisation of a Half-subtractor***

Table 5.14 ***Truth Table of a Full-subtractor***

Inputs			Outputs	
A_n	B_n	C_{n-1}	D_n	C_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full-Subtractor

The K-map for C_n is given in Fig. 5.23*a* and its realisation is given in Fig. 5.23*b*. The simplified expression for C_n is

$$C_n = \bar{A}_n B_n + \bar{A}_n C_{n-1} + B_n C_{n-1} \tag{5.37}$$

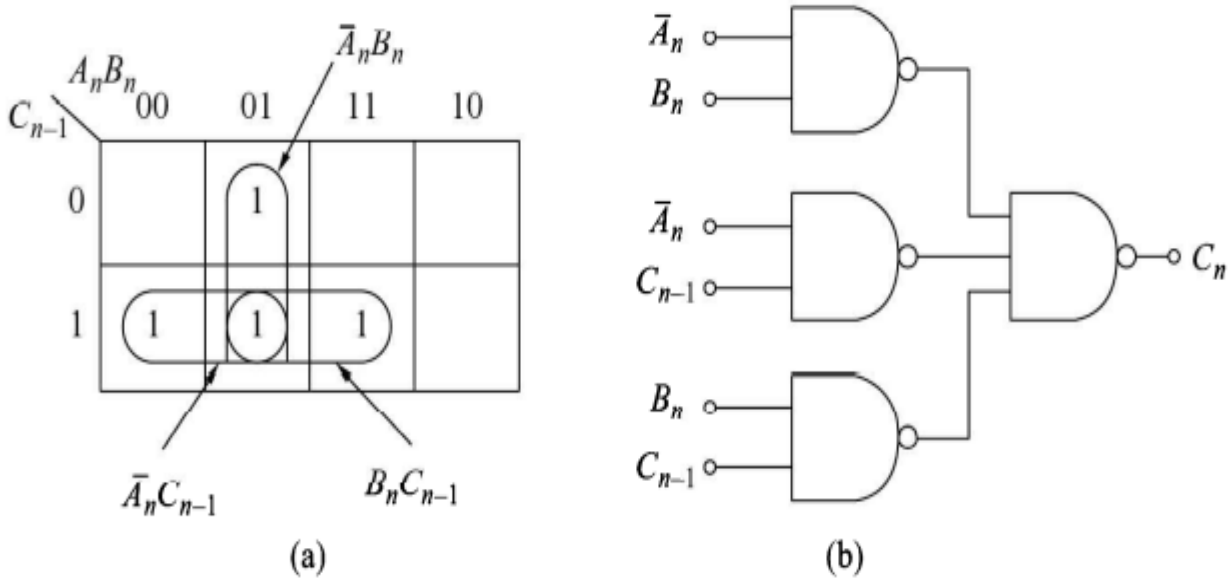


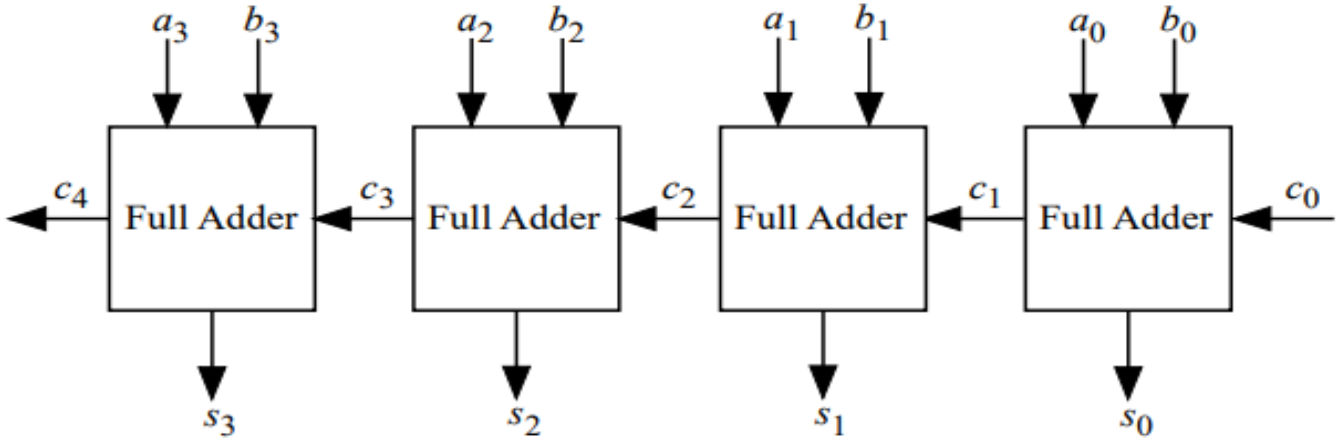
Fig. 5.23 ***(a) K-map for C_n (b) Realisation of C_n***

4-bit binary parallel adder

- a,b are the addend and augend inputs
- Co, C1…… are carry inputs
- S0…. sum

Most significant stage

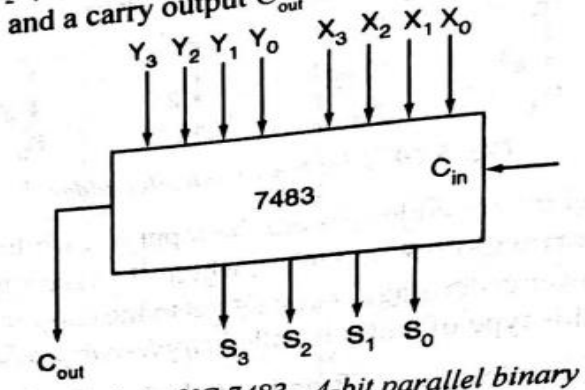
Least significant stage



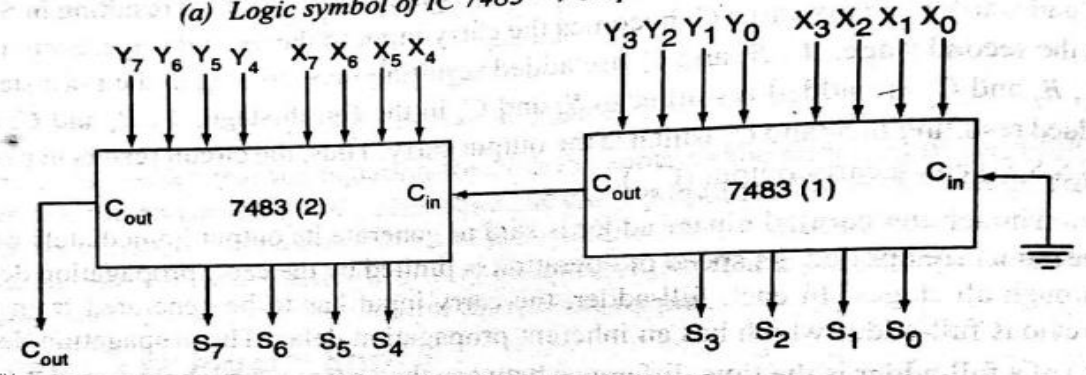
Parallel Adder

5.8.1 IC 7483—4-bit Parallel Binary Adder

The IC 7483 is a commonly available TTL 4-bit parallel binary adder chip. It contains four interconnected full-adders and a look-ahead carry circuitry for its operation. The logic symbol of IC 7483 is shown in Fig. 5.11(a). It has two 4-bit inputs $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$, and, a carry input C_{in} in the LSB stage. The outputs are a 4-bit sum $S_3S_2S_1S_0$ and a carry output C_{out} from the most significant bit stage.



(a) Logic symbol of IC 7483—4-bit parallel binary adder



(b) Cascading of two 7483 ICs

Parallel Subtractor

5.8.2 4-bit Parallel Binary Subtractor

Just as a parallel binary adder can be implemented by cascading several full-adders, a parallel binary subtractor can also be implemented by cascading several full-subtractors. A 4-bit parallel binary subtractor that subtracts a 4-bit number $Y_3Y_2Y_1Y_0$ from another 4-bit number $X_3X_2X_1X_0$ is shown in Fig. 5.12. It has 4 difference outputs ($D_3D_2D_1D_0$) and a borrow output (B_{out}). Note that the B_{in} of the LSB full-subtractor is connected to 0 and B_{out} of ith full-subtractor is connected to B_{in} of $(i+1)$th full-subtractor.

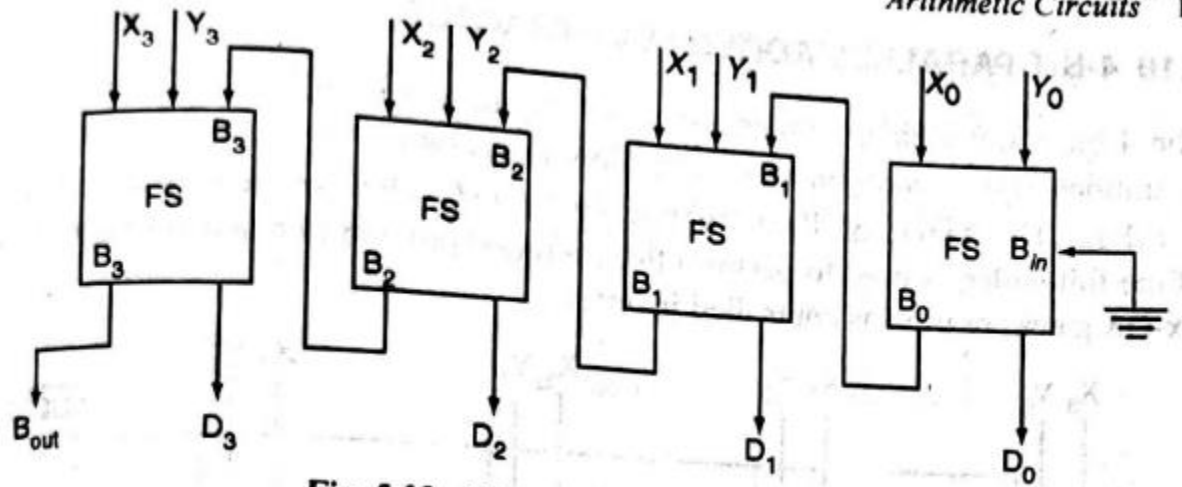


Fig. 5.12 4-bit parallel binary subtractor

Parallel Adder and Subtractor

5.10 4-BIT PARALLEL ADDER / SUBTRACTOR

The 4-bit parallel binary adder/subtractor circuit shown in Fig. 5.15 performs the operations of both addition and subtraction. It has two 4-bit inputs $X_3X_2X_1X_0$ and $Y_3Y_2Y_1Y_0$. The $\overline{\text{ADD}}$ / SUB control line, connected with C_{in} of the least significant bit of the full-adder, is used to perform the operations of addition and subtraction. The Ex-OR gates are used as controlled inverters.

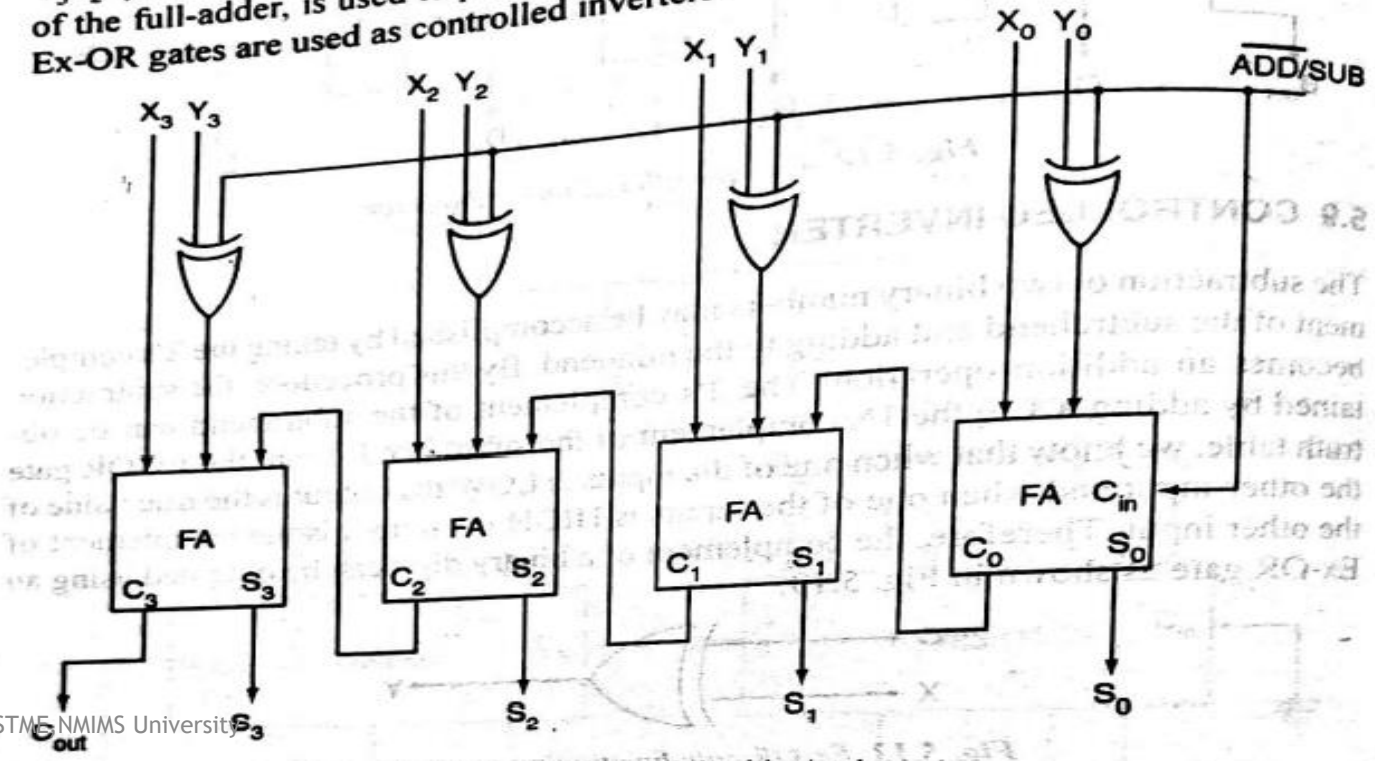
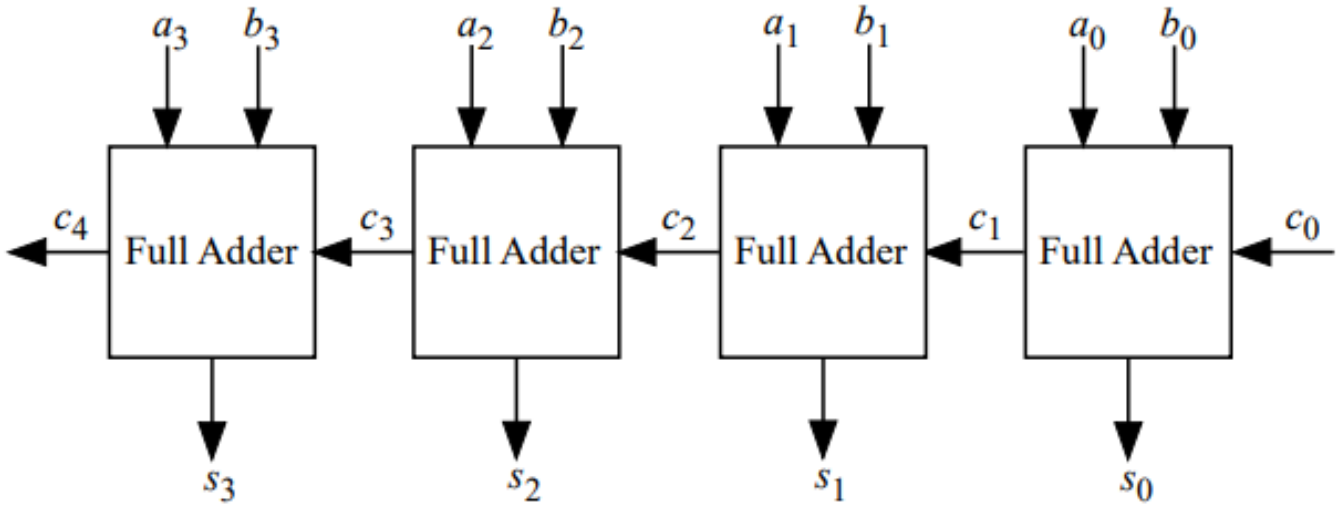


Fig. 5.15 4-bit parallel adder/ subtractor

Ripple Carry Adder/Parallel Adders

- A ripple carry adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascaded
- the carry output from each full adder connected to the carry input of the next full adder in the chain.
- The interconnection of four full adder (FA) circuits to provide a 4-bit ripple carry adder.
- Notice from Figure that the input is from the right side because the first cell traditionally represents the least significant bit (LSB). Bits a0 and b0 in the figure represent the least significant bits of the numbers to be added. The sum output is represented by the bits s0 - s3.



RIPPLE CARRY ADDER

- In the ripple carry adder, the output is known after the carry generated by the previous stage is produced.
- Thus, the sum of the most significant bit is only available after the carry signal has rippled through the adder from the least significant stage to the most significant stage.
- As a result, the final sum and carry bits will be valid after a considerable delay.
- For a, n-bit ripple carry adder the sum and carry bits of the most significant bit (MSB) are obtained after normalization.

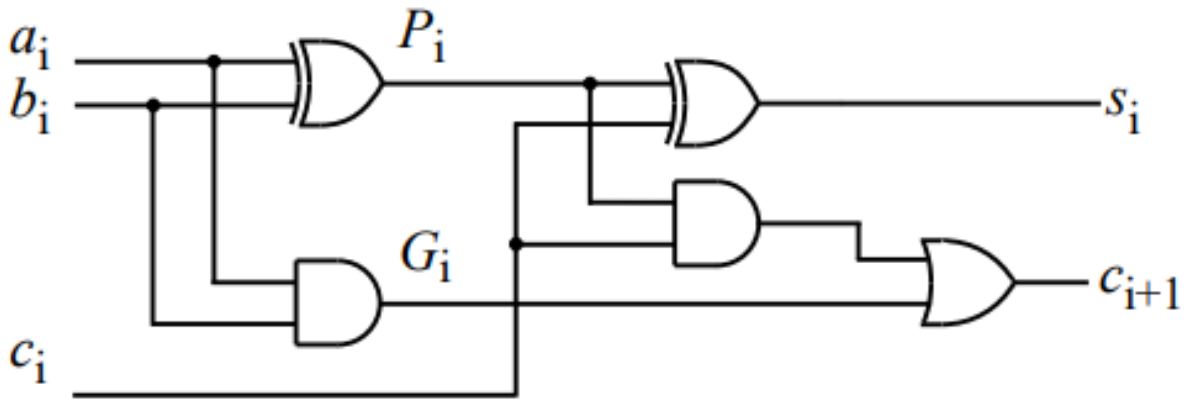
CARRY LOOK AHEAD ADDER (CLA)

- The ***carry look ahead adder (CLA)*** solves the carry delay problem by calculating the carry signals in advance, based on the input signals.
- It is based on the fact that a carry signal will be generated in two cases: (1) when both bits ai and bi are 1, or (2) when one of the two bits is 1 and the carry-in is 1. Thus, one can write,

$$\begin{aligned} c_{i+1} &= a_i.b_i + (a_i \oplus b_i).c_i \\ s_i &= (a_i \oplus b_i) \oplus c_i \end{aligned}$$

- The above two equations can be written in terms of two new signals and Pi and Gi, which are shown in Figure:

CARRY LOOK AHEAD ADDER (CLA)



$$c_{i+1} = a_i.b_i + (a_i \oplus b_i).c_i$$
$$s_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = G_i + P_i.c_i$$
$$s_i = P_i \oplus c_i$$

$$G_i = a_i.b_i$$
$$P_i = a_i \oplus b_i$$

Gi and Pi are called the carry generate and carry propagate terms, respectively.

CARRY LOOK AHEAD ADDER (CLA)

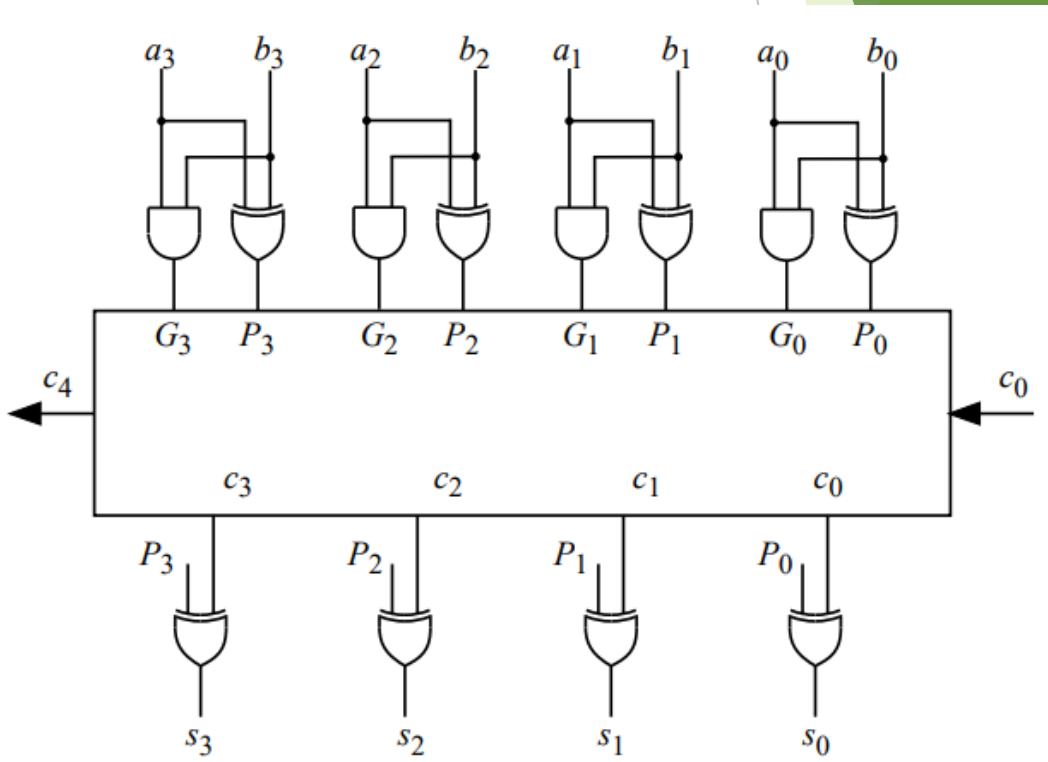
- The generate and propagate terms only depend on the input bits and thus will be valid after one and two gate delay, respectively.
- If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value.
- Applying this to a 4-bit adder the carry equation will be

Putting $i = 0, 1, 2, 3$ in Equation 5, we get

$$
\begin{aligned}
c_1 &= G_0 + P_0.c_0 \\
c_2 &= G_1 + P_1.G_0 + P_1.P_0.c_0 \\
c_3 &= G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.c_0 \\
c_4 &= G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.
\end{aligned}
$$

$$
\begin{aligned}
c_{i+1} &= G_i + P_i.c_i \\
s_i &= P_i \oplus c_i
\end{aligned}
$$

$$
\begin{aligned}
G_i &= a_i.b_i \\
P_i &= a_i \oplus b_i
\end{aligned}
$$



the carry-out bit, C(i+1) , of the last stage will be available after four delays

Serial Adder

The diagram of a 4-bit serial adder is shown in Fig. 5.16. The two shift registers A and B are used to store the numbers to be added serially. A single full-adder is used to add one pair of bits at a time along with the carry. The D flip-flop, i.e. carry flip-flop, is used to store the carry output of the full-adder so that it can be added to the next significant position of the numbers in the registers. The contents of the shift registers shift from left to right and their outputs starting from A_0 and B_0 are fed into a single full-adder along with the output of the carry flip-flop upon application of each clock pulse.

The sum output of the full-adder is fed to the Most Significant Bit (S_3) of the sum register. For each succeeding clock pulse, the contents of both the shift registers are shifted once to the right, and new sum bit and new carry bit are transferred to sum register and carry flip-flop respectively. This process continues until all the pairs of -bits are added.

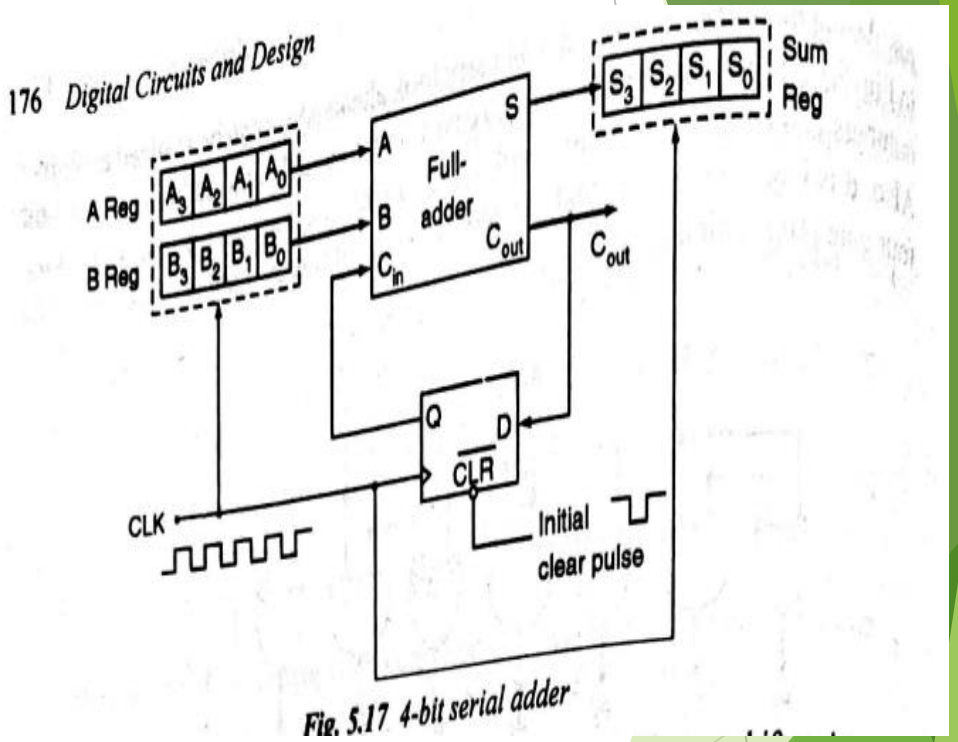


Fig. 5.17 4-bit serial adder

Serial Subtractor

5.13 SERIAL SUBTRACTION USING 2'S COMPLEMENT

A serial subtractor can be obtained by converting the serial adder of Fig. 5.17 using the 2's complement system. For subtraction, the subtrahend is stored in the B register and must be 2's-complemented before it is added to the minuend stored in the A register. One simple way to accomplish this is to complement the B register and to make the initial $C_{in} = 1$ instead of 0 prior to the first clock pulse. This can be easily accomplished by feeding the inverted output $\overline{B_0}$ into the full-adder instead of B_0 and initially setting the carry flip-flop to 1 instead of clearing it. The remaining circuitry is the same as serial adder.

Serial Subtractor

The circuit for 4-bit serial subtractor using full-adder is shown in Fig. 5.18.

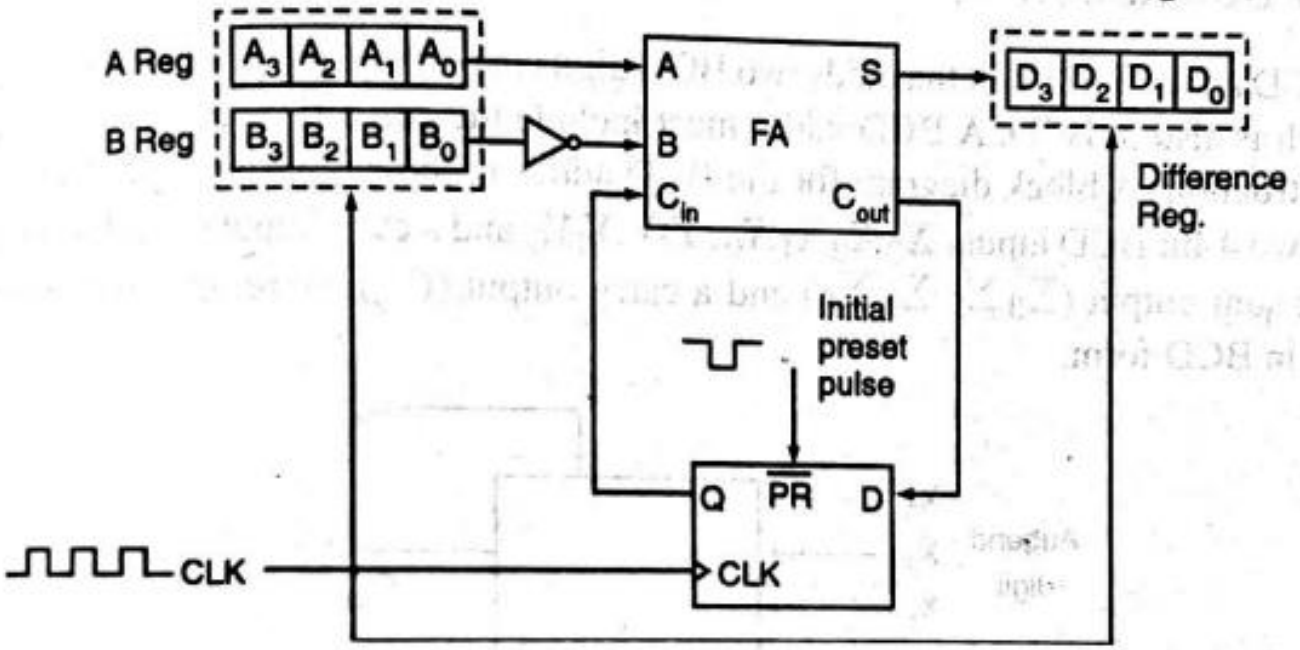


***Fig. 5.18** 4-bit serial subtractor using full-adder*

4-Bit Serial Subtractor

5.14 4-BIT SERIAL ADDER / SUBTRACTOR

A 4-bit serial adder/subtractor can be constructed in a manner similar to its parallel counterpart as shown in Fig. 5.19.

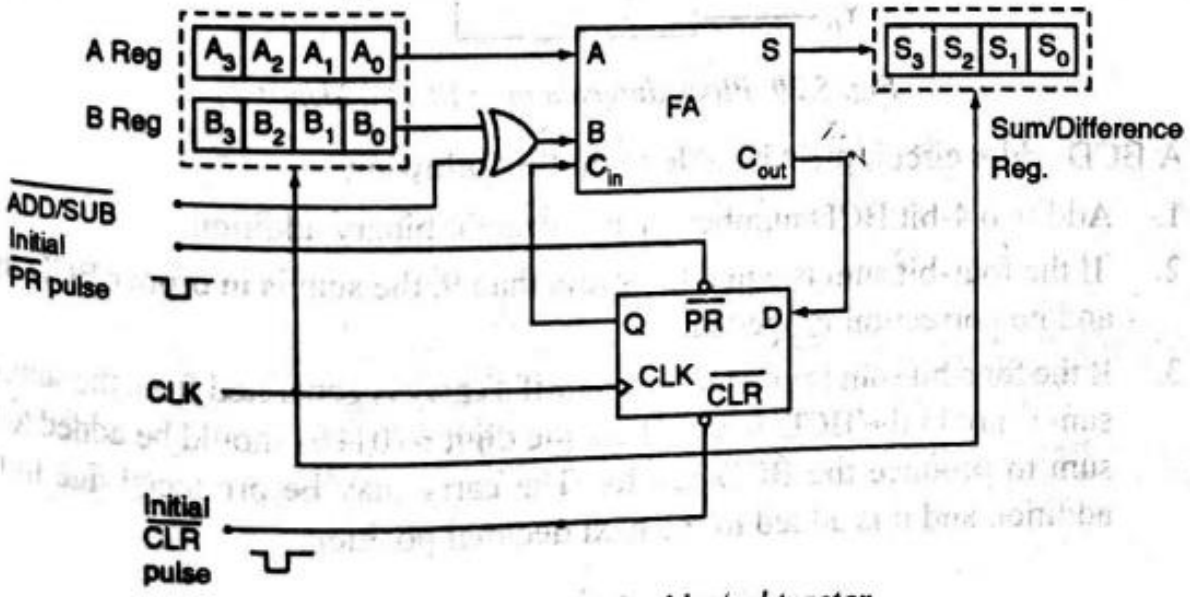


Fig. 5.19 4-bit serial adder/subtractor

Booth's Multiplication algorithm

- Booth's algorithm is a multiplication algorithm that multiplies two signed binary numbers in 2's compliment notation.

PROCEDURE:

1. Let M is the multiplicand.
2. Let Q is the multiplier.
3. Consider a 1-bit register Q_{-1} and initialize it to 0.
4. Consider a register A and initialize it to 0.

CONDITIONS:

1. If $Q_0\,Q_{-1}$ are same i.e. 00 or 11 then, perform arithmetic right shift by 1 bit.
2. If $Q_0\,Q_{-1}$ = 10 then perform

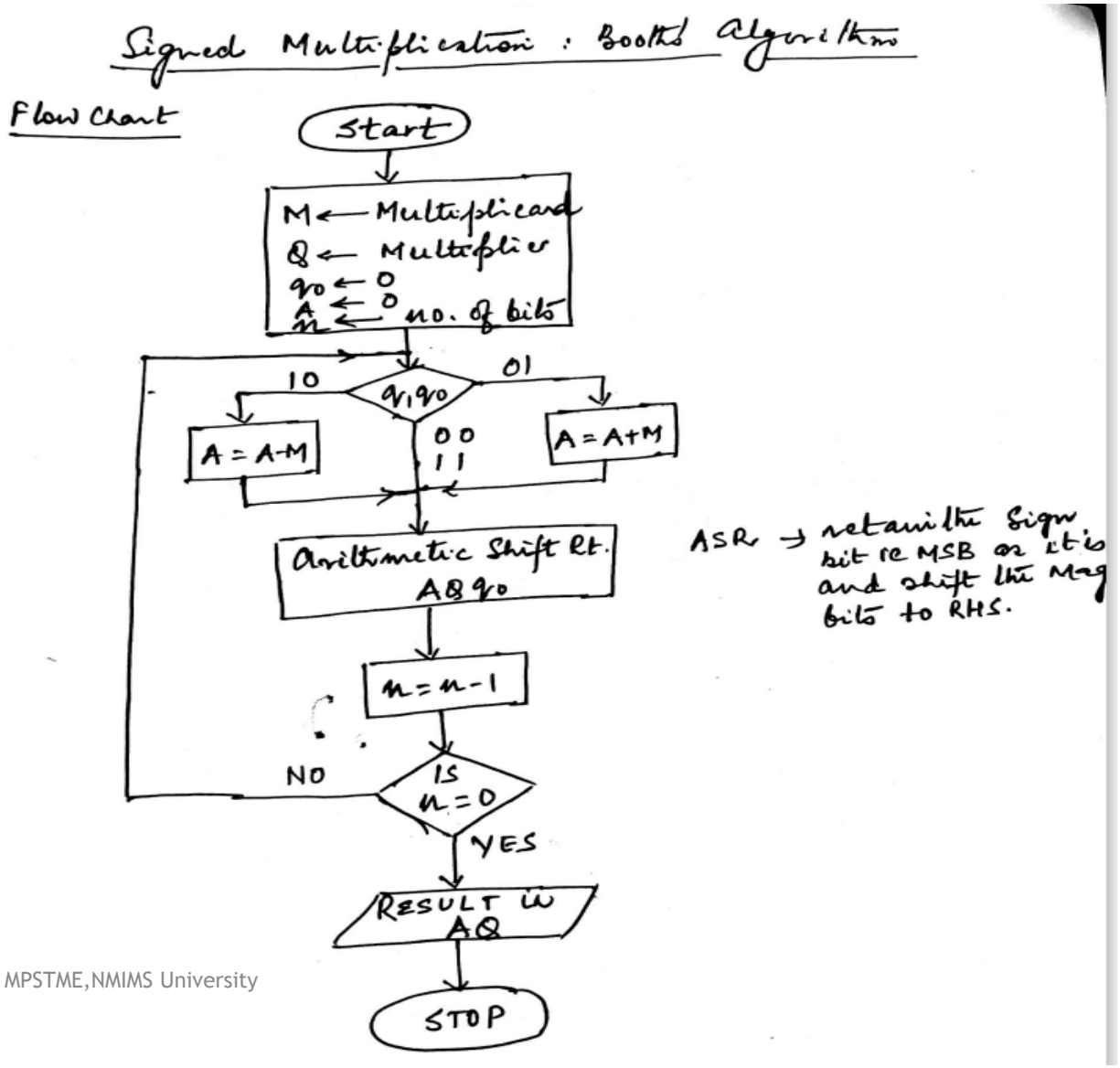
   A←A-M

   And then perform arithmetic right shift.
3. If $Q_0\,Q_{-1}$ = 01 then perform

   A←A+M

   And then perform arithmetic right shift.

Booth's Algorithm



Booth's Algorithm

Example

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

Multiplicand × Multiplier = Product

Multiplicand = $(-7)_{10}$
= 2's Complement of $(0111)_2$ $(1000 + 1 \rightarrow 1001)$
= 1001_2
ie $-M = 0111_2$

Q = 0011 (Multiplier)
* n = 4 bit no.
A initialized to 0 (Accumulator)
q_0 = initialized to 0
$q_1 q_0$ (00, 01, 10, 11)
$\leftarrow q_1 q_0 = 10$
* A − M = A + (−M)

Tracing Table

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

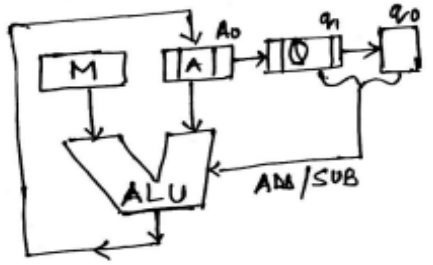
As n = 0, the op. is over
now we see the MSB = 1 ∴ the sign bit is 1 ∴ the number is (−ve).
So we write the 2's Complement of the no. 1110 1011

1's Comp = 0001 0100
2's Comp = 0001 0101 (+1)

⇓

(−) 21 ie $(-21)_{10}$

HARDWARE STRUCTURE IMPLEMENTING BOOTH'S ALGORITHM



Booth's Multiplication algorithm

For example:

Consider two numbers 6 and 2 and we have to perform their multiplication by using Booth's algorithm.

Here 6 is multiplicand (M) and 2 is multiplier (Q).

Now write 6 and 2 in binary form.

M = 6 = 0 1 1 0

Q = 2 = 0 0 1 0 (Q_3, Q_2, Q_1, Q_0)

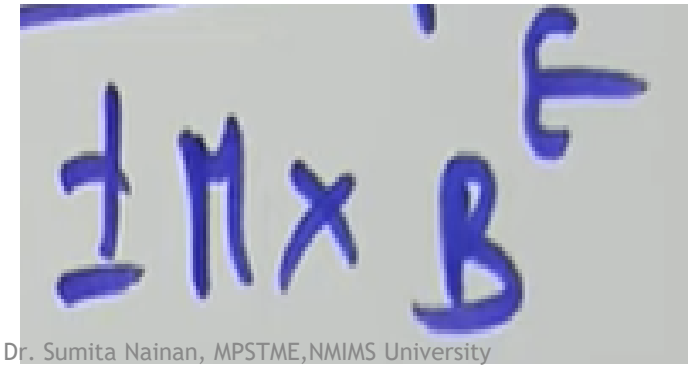
Booth's algorithm calculates the product in n steps where n is the number of bits used to represent the numbers.

Booth's Multiplication algorithm

INITIALISE	A	B	Q_{-1}	OPERATIONS
	0 0 0 0 ↓↘↘↘↘	0 0 1 0 ↘↘↘↘	0	
Step 1.	0 0 0 0	0 0 0 1 ↓	0 ↓	Arithmetic right shift
Step 2.	1 0 1 0 ↓↘↘↘↘ 1 1 0 1	0 0 0 1 ↘↘↘↘ 0 0 0 0	0 1	A←A-M Then shift
Step 3.	0 0 1 1 ↓↘↘↘↘ 0 0 0 1 ↓↘↘↘↘	0 0 0 0 ↘↘↘↘ 1 0 0 0 ↘↘↘↘	1 0	A←A+M Then shift
Step 4.	0 0 0 0	1 1 0 0 In binary, 12 = 1100 Hence 3*2 = 12	0	Arithmetic right shift

COMPUTER REPRESENTATION OF FLOATING POINT NUMBERS (IEEE 754 FORMAT)

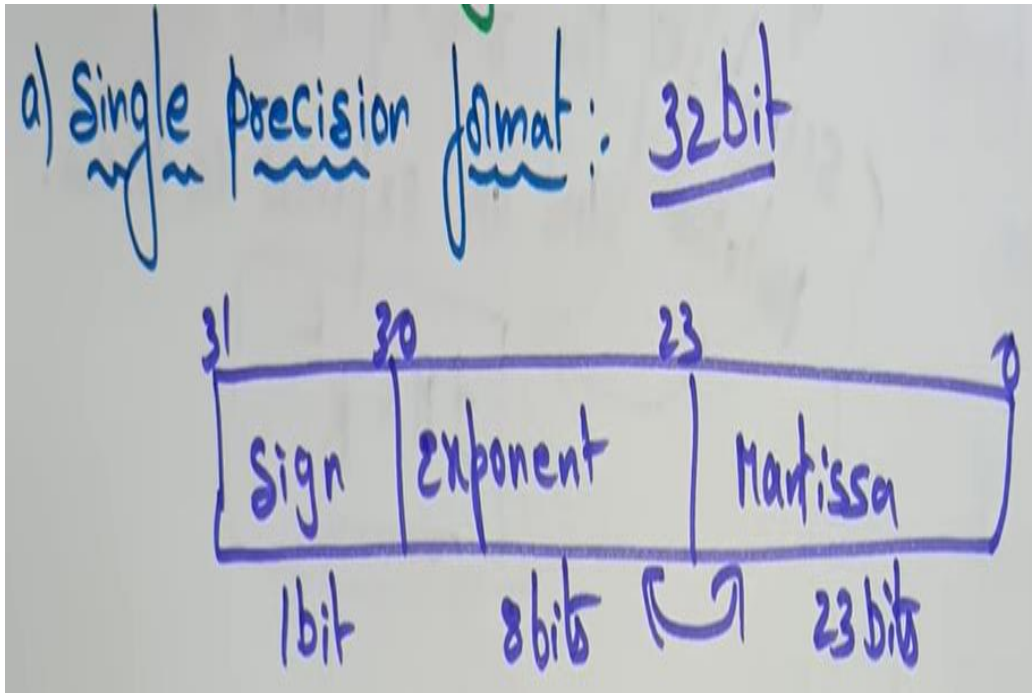
- In the CPU, a 32-bit floating point number is represented using IEEE standard format as follows:
- S | MANTISSA | BASE | EXPONENT
- where S is one bit, the EXPONENT is 8 bits, and the MANTISSA is 23 bits.
- The mantissa represents the leading significant bits in the number.
- The exponent is used to adjust the position of the binary point (as opposed to a "decimal" point)

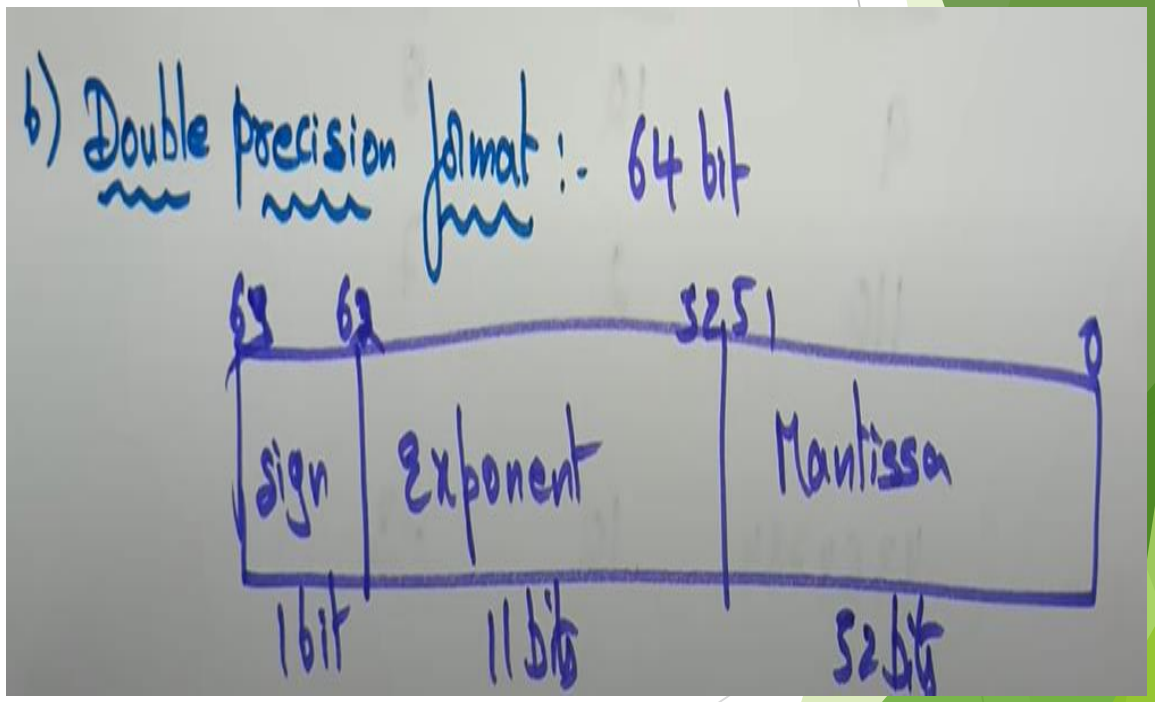


Number	Mantissa	Base	Exponent
9×10^8	9	10	8
110×2^7	110	2	7
4364.784	4364784	10	-3

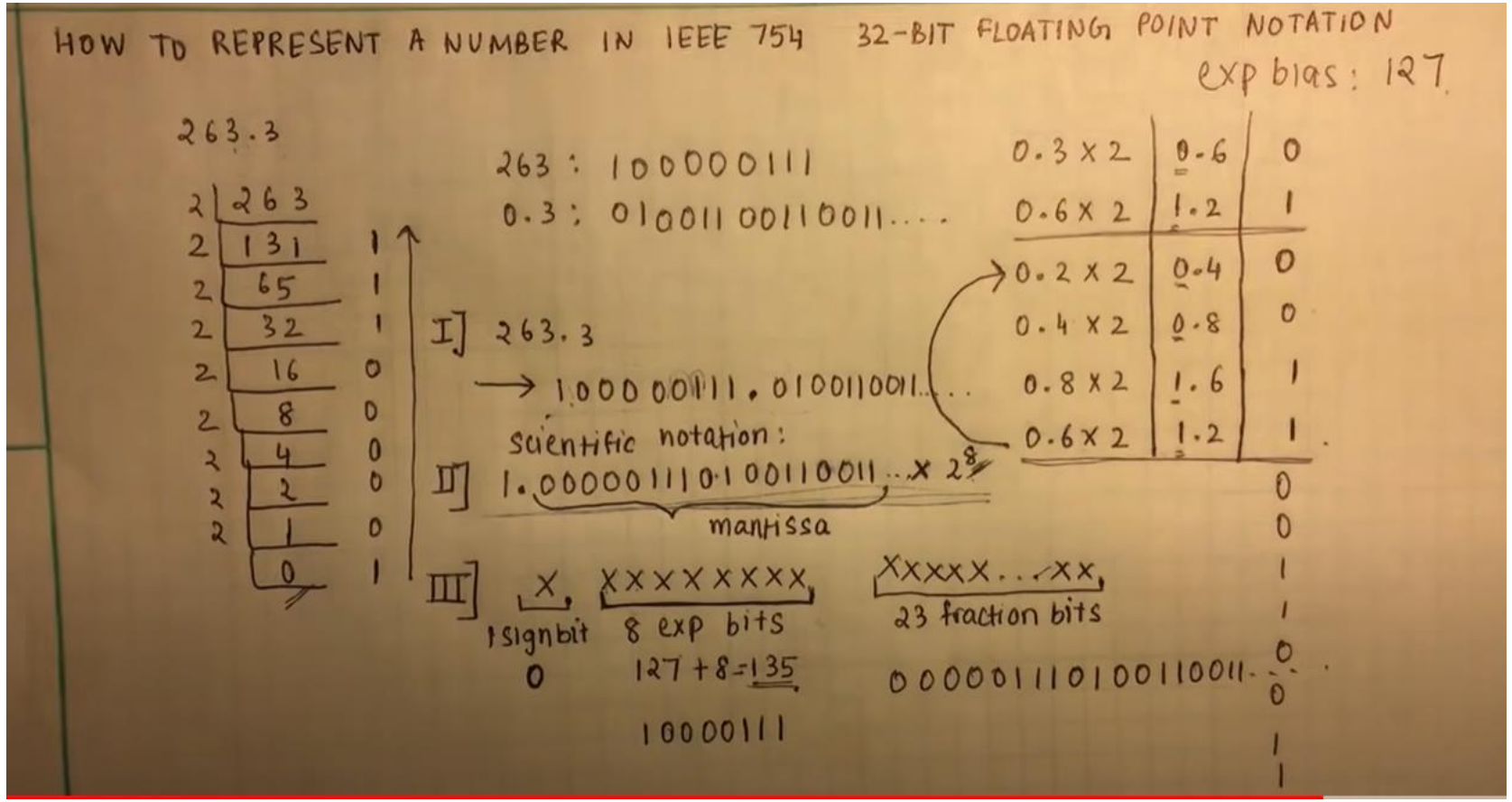
IEEE 754 format

- Single precision format:32 bit format
- Double precision format:64 bit format





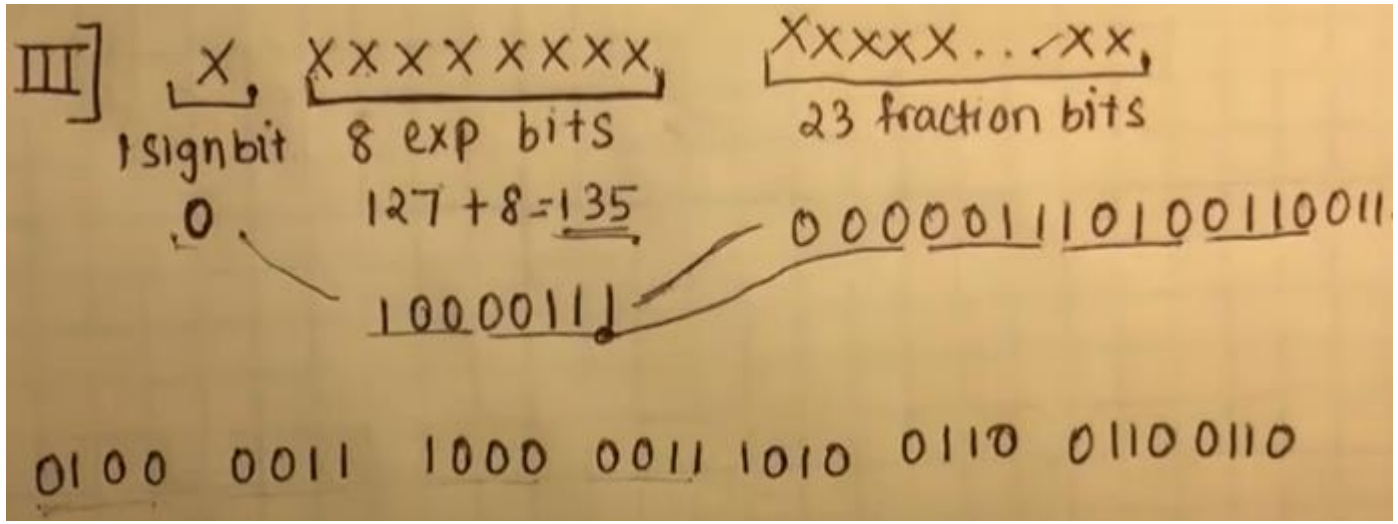
IEEE 754 32 bit Floating Point representation



Exponent has to cover –ve and +ve bias. Exp bias for signed representation is 127 so if exponent is –ve no. you sub. from the bias and if exp is +ve you add to the bias.

IEEE 754 Representation

Final IEEE 754 representation of the number 263.3 is as given below



Combinational and Sequential ALU

- Introduction to ALU
- Introduction to Combinational Circuits
- Design Procedure of Combinational Circuits
- Analysis Procedure of Combinational Circuits
- Introduction to Sequential Circuits
- Types of Sequential Circuits

Arithmetic Logic Unit

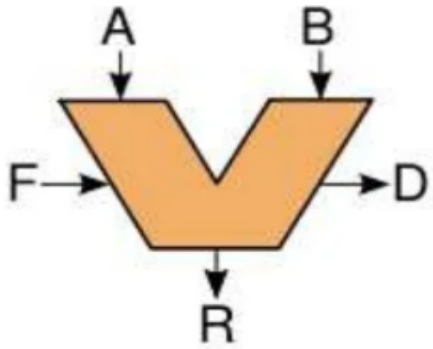
- ALU stands for: Arithmetic Logic Unit
- ALU is a digital circuit that performs Arithmetic (Add, Sub, . . .) and Logical (AND, OR, NOT) operations.
- John Von Neumann proposed the ALU in 1945 when he was working on EDVAC.

Schematic of the ALU

Typical Schematic Symbol of an ALU

A and B: the inputs to the ALU (aka operands)
R: Output or Result
F: Code or Instruction from the Control Unit (aka as op-code)
D: Output status; it indicates cases such as:

- carry-in
- carry-out,
- overflow,
- division-by-zero
- And . . .

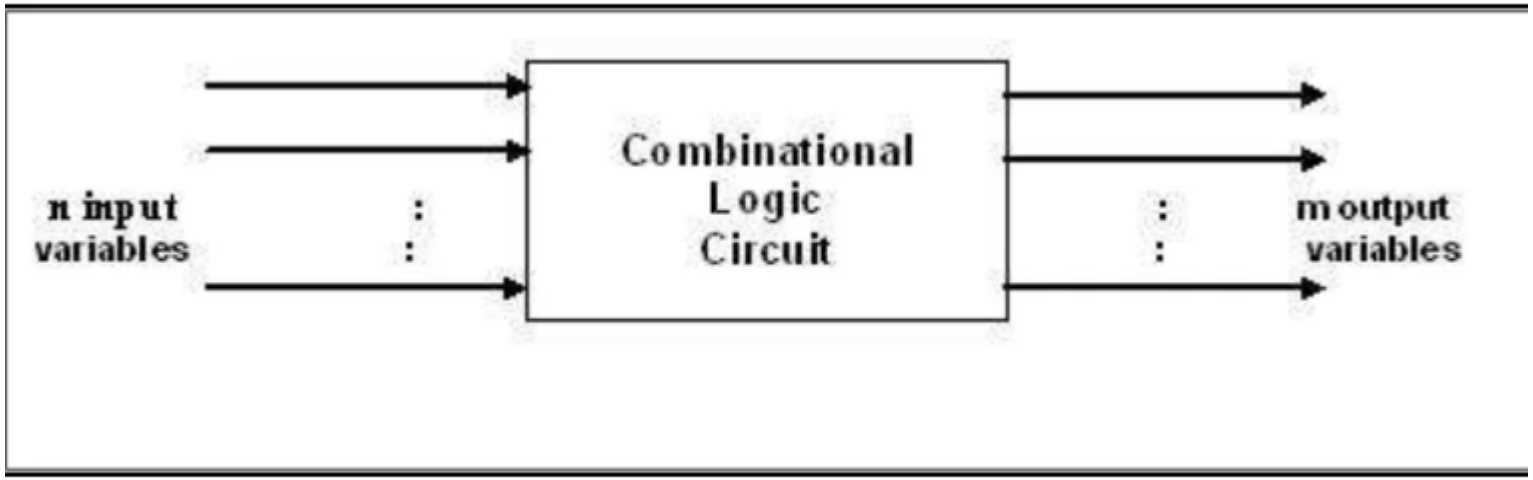


Types of Digital Logic circuits in ALU

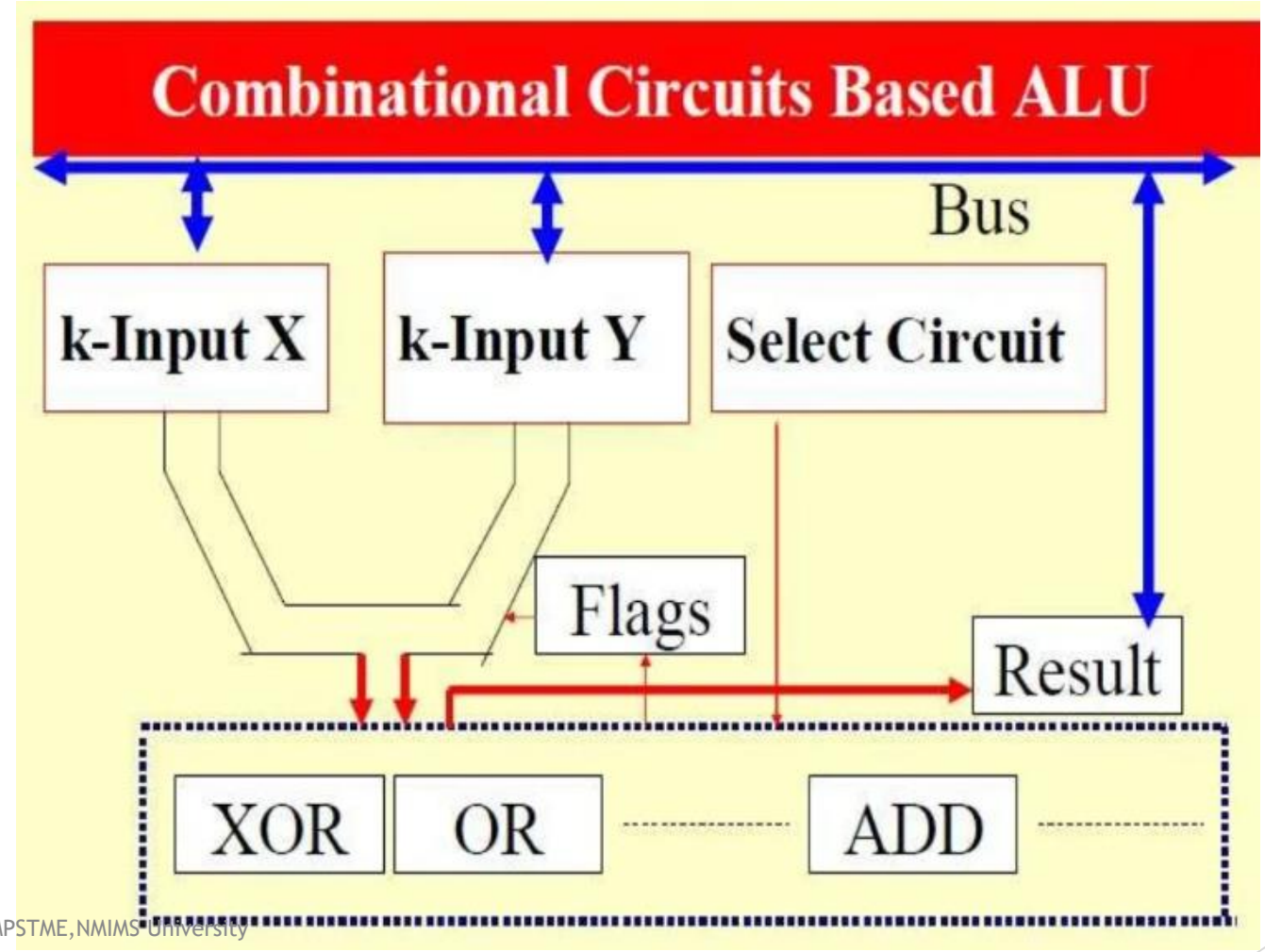
- COMBINATIONAL CIRCUITS
- SEQUENTIAL CIRCUITS

- Combinational Circuits are made of logic gates.
- Doesn't contain memory element , that's why they cant store any information.
- Value of present output is determined by present input.
- Examples of combinational circuits are half adders, full adders, sub tractors etc.

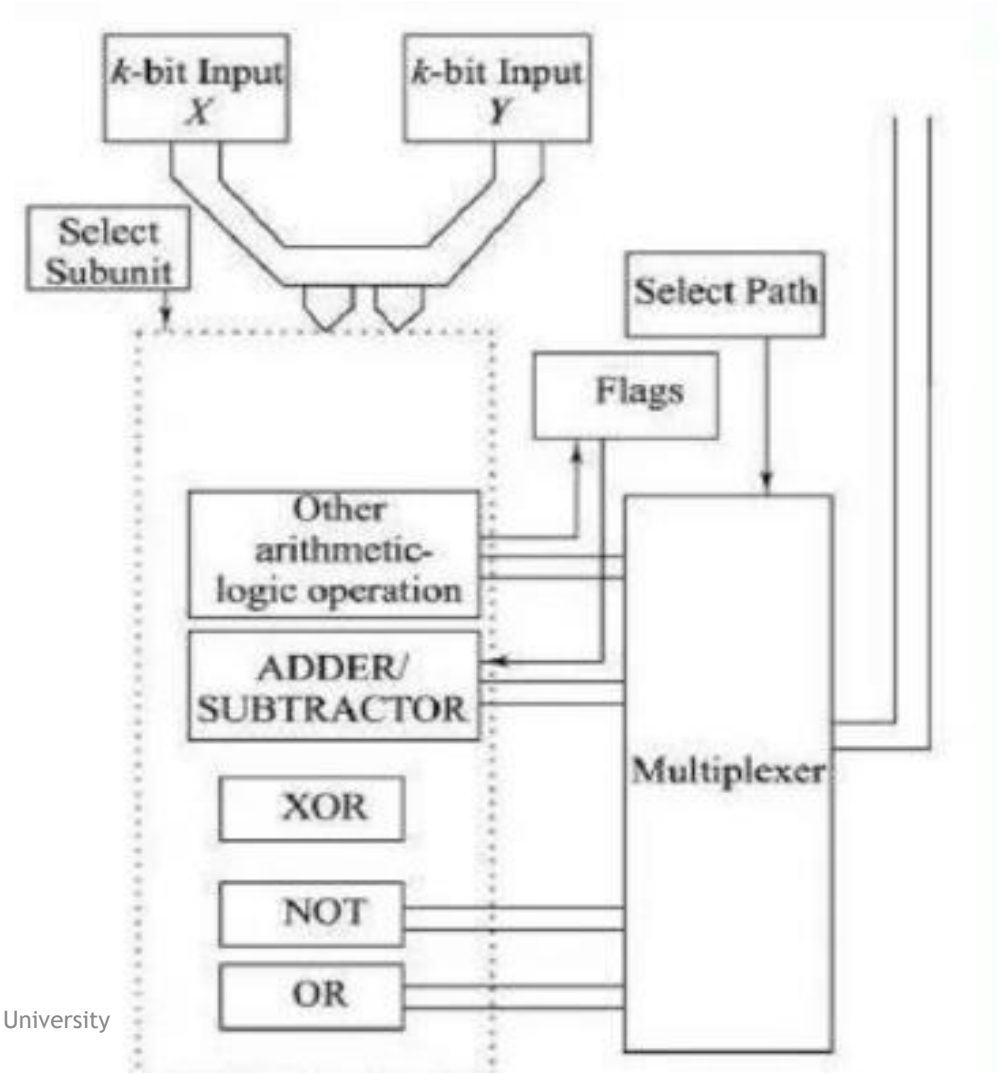
Block Diagram of Combinational Circuit



Combinational Circuit based ALU



Combinational Circuit based ALU



Examples of Combinational Circuits

- Multiplexer
- Demultiplexer
- Encoder
- Decoder
- Half Adder
- Full Adder

Multiplexer and Demultiplexer

- Multiplexer-
  - A multiplexer is a combinational circuit where binary information from one of many input lines is selected and directs it to a single output line.
- Demultiplexer-
  - Demultiplexing is the reverse process of multiplexing; i.e., a demultiplexer is a combinational circuit that receives information on a single line and transmits this information on one of 2n possible output lines.
- Encoder-
  - An encoder is a combinational circuit that produces the reverse function from that of a decoder.
- Decoder-
  - A decoder is a combinational logic circuit that receives coded information on n input lines and feeds them to maximum of 2n unique output lines after conversion.

Combinational Circuits

- Half-Adder :
  - A half-adder is a combinational circuit that performs the addition of two bits.

- Full Adder :
  - This type of adder is a little more difficult to implement than a half-adder.
  - The main difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs.

Design Procedure for Combinational Circuits

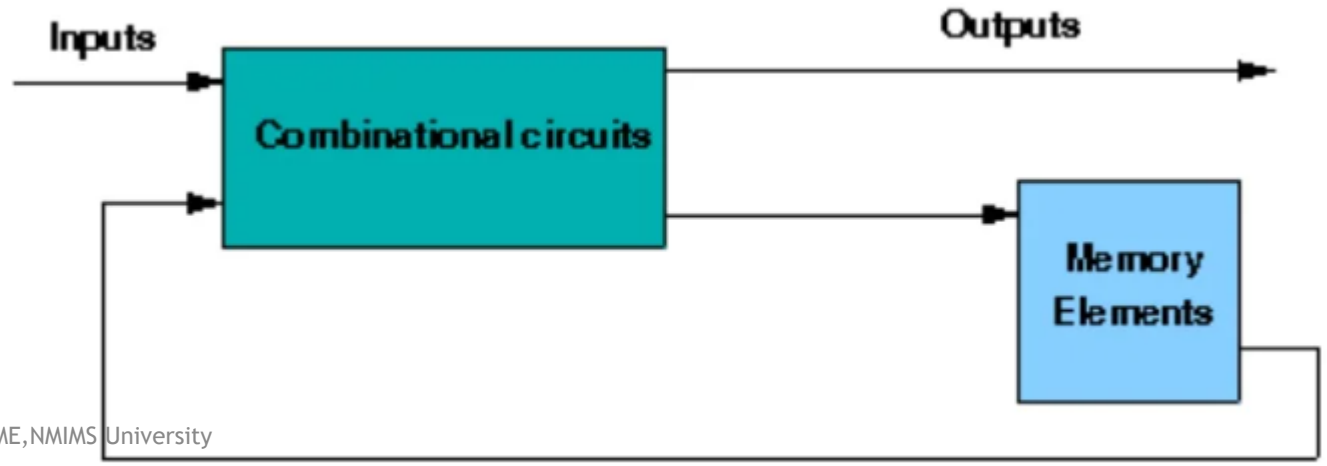
This procedure involves the following steps:

- The problem is stated.
- The number of available input variables and output variables is determined.
- The input and output variables are assigned letter symbols.
- Truth table is drawn
- Boolean function for output is obtained.
- The logic diagram is drawn.

- TO DETERMINE THE OUTPUT FUNCTIONS AS ALGEBRAIC EXPRESSIONS.
- It is the reverse process of design procedure.
- Logic diagram of the circuit is given.
- Obtain the truth table from the diagram.
- Obtain Boolean function from the Truth Table for output.

Sequential Logic Circuits

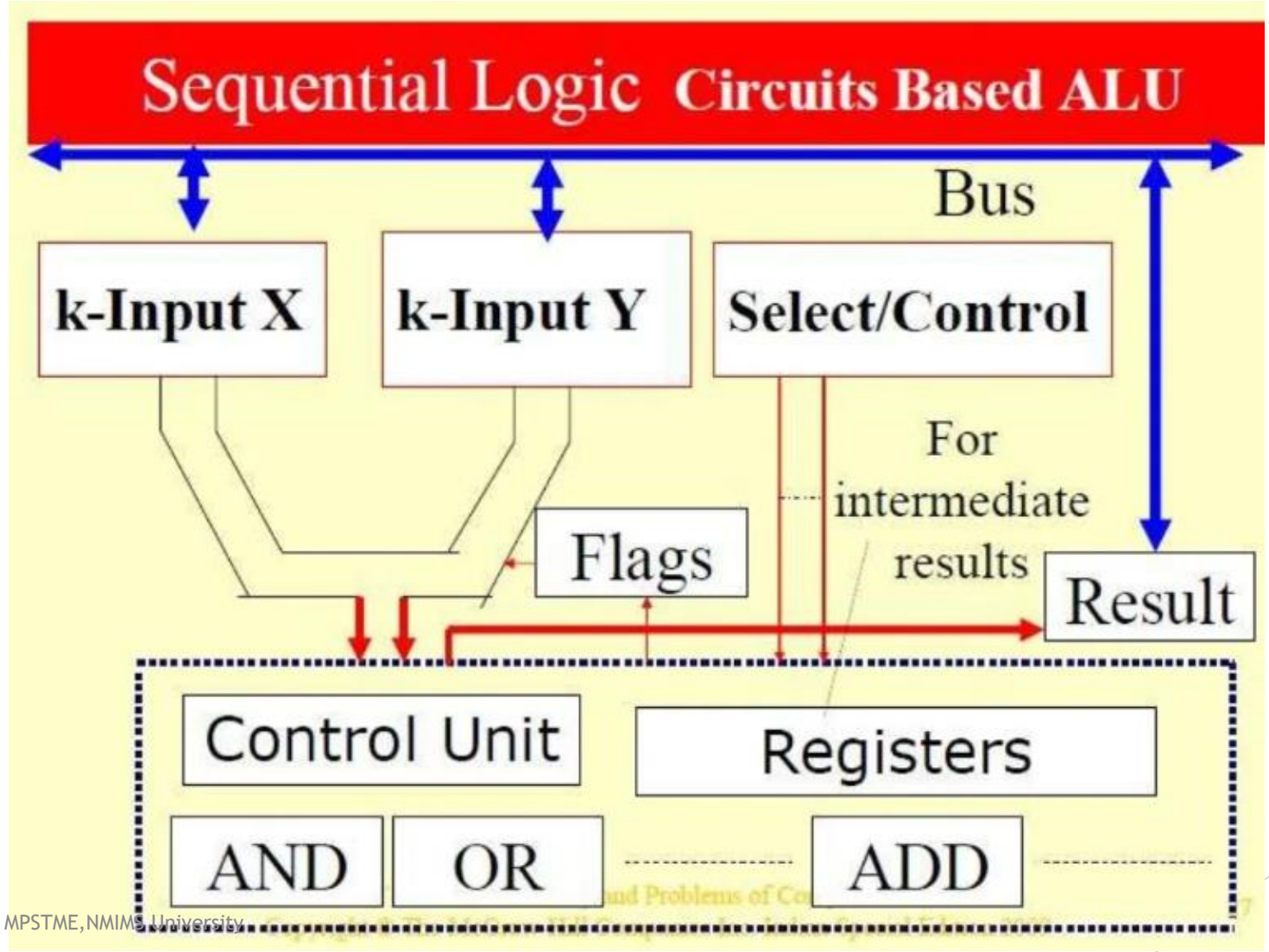
- Made up of combinational circuits and memory elements.
- These memory elements are devices capable of storing ONE-BIT information.
- Output depends on input and previous state.
- Examples of sequential circuits are flip flops, counters, shift registers



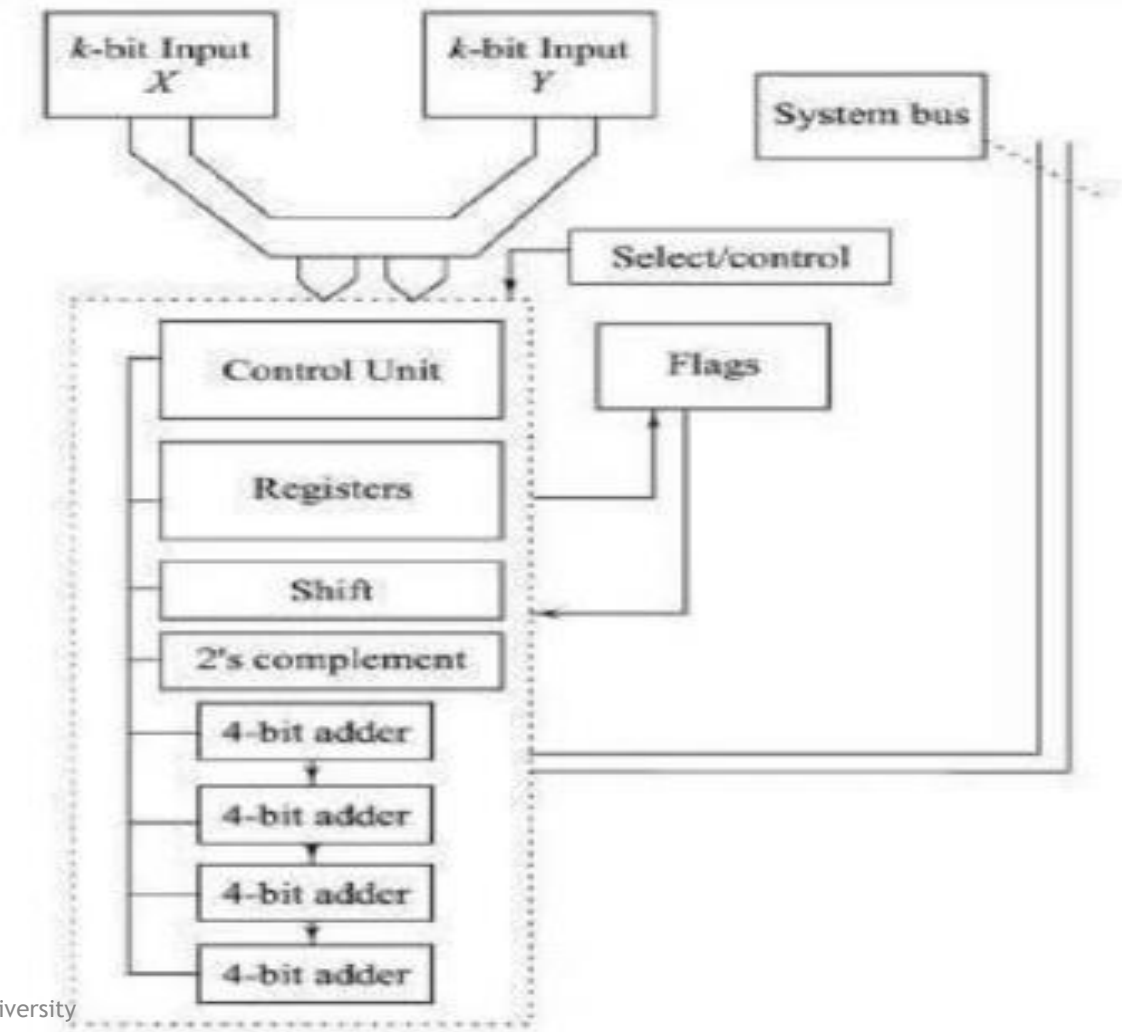
Sequential Logic Circuits

- An ALU is the fundamental unit of any computing system.
- Understanding how an ALU is designed and how it works is essential to building any advanced logic circuits.
- Using this knowledge and experience, we can move on to designing more complex integrated circuits.
- The ALU is the "heart" of a processor—you could say that everything else in the CPU is there to support the ALU.

Sequential Logic Circuits



Sequential Logic Circuits



Examples of Sequential Circuits

- Flip-Flops
  - JK Flip-Flop
  - RS Flip-Flop
  - PR Flip-Flop
  - D Flip-Flop
- Registers
- Counters

- Flip-Flops are the basic building blocks of sequential circuits.
- A flip-flop is a binary cell which can store a bit of information.
- A basic function of flip-flop is storage, which means memory. A flip-flop (FF) is capable of storing 1 (one) bit of binary data.
- It has two stable states either '1' or '0'. A flip-flop maintains any one of the two stable states which can be treated as zero or one depending on presence and absence of output signals.

Sequential circuits are of two types:

- *SYNCHRONOUS SEQUENTIAL CIRCUITS*
- *ASYNCHRONOUS SEQUENTIAL CIRCUITS*

Sequential Logic Circuits

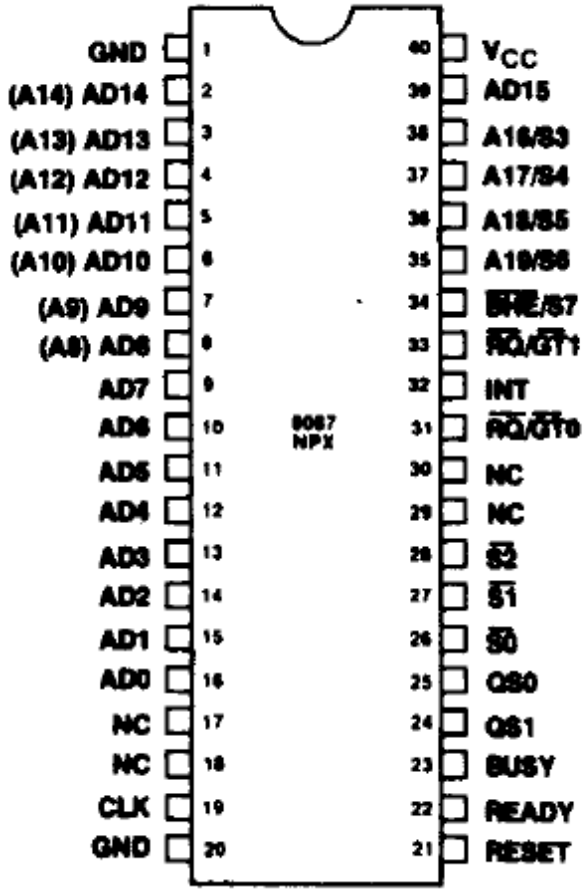
- A circuit with flip-flops is considered a sequential circuit even in the absence of combinational logic.
- Circuits that include flip-flops are usually classified by the function they perform.
- Two such circuits are registers and counters:
- **Registers-**
  - It is a group of flip-flops.
  - Its basic function is to hold information within a digital system so as to make it available to the logic units during the computing process.
- **Counters-**
  - It is essentially a register that goes through a predetermined sequence of states.

- **Link for high speed multipliers**
- https://www.youtube.com/watch?v=YiuUINlgX2g&t=886s
- https://www.youtube.com/watch?v=lK4-TNdOpYo
- https://www.youtube.com/watch?v=T12nc1Hdmq4

Overview of 8087 Maths co-Processor

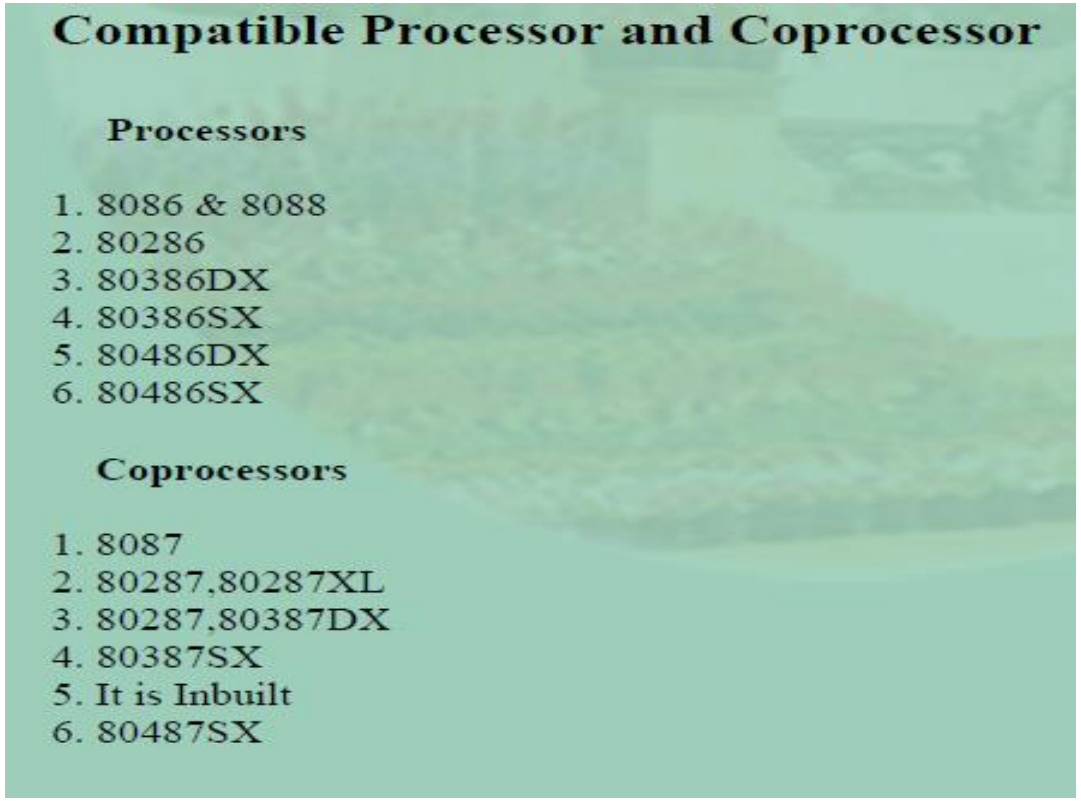
- **Maths Co-Processor is also known as**
- **NPX** (Numeric Processor Extension)
- **NDP** (Numeric Data Processor)
- **FUP** (Floating point Unit)

- **Adds Arithmetic, Trigonometric, Exponential, and Logarithmic Instructions to the Standard 8086/8088 and 80186/80188 Instruction Set for All Data Types**
- **CPU/8087 Supports 7 Data Types: 16-, 32-, 64-Bit Integers, 32-, 64-, 80-Bit Floating Point, and 18-Digit BCD Operands**
- **Compatible with IEEE Floating Point Standard 754**
- **Available in 5 MHz (8087), 8 MHz (8087-2) and 10 MHz (8087-1): 8 MHz 80186/80188 System Operation Supported with the 8087-1**
- **Adds 8 x 80-Bit Individually Addressable Register Stack to the 8086/8088 and 80186/80188 Architecture**
- **7 Built-In Exception Handling Functions**
- **MULTIBUS System Compatible Interface**

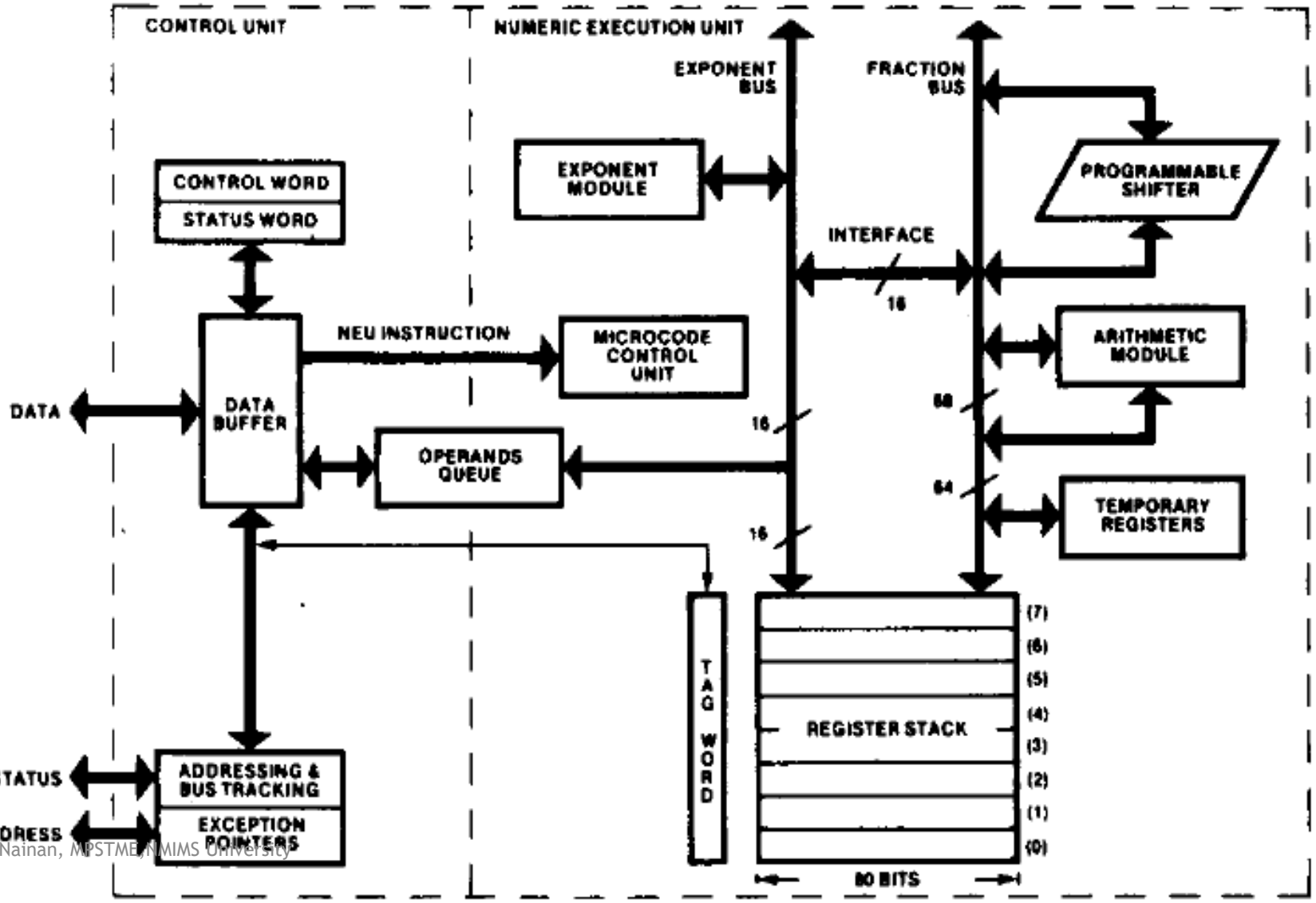


Adds 64 mnemonics to the instruction set of 8086. It has 40 pins.

Overview of 8087 Maths co-Processor



Block Diagram of 8087- Maths Coprocessor



Overview of 8087 Maths co-Processor

Control Unit (CU)

- data buffer
- shared operand queue
- control, and status word register
- addressing and bus tracking unit
- and exception pointer.

Numeric Execution Unit (NEU)

- register stack
- microcode control unit
- programmable shifter
- arithmetic module
- temporary registers
- shared operand queue
- exponent module

Units of 8087

- **The control unit** of the coprocessor controls the instruction execution for which NEU is responsible.
- Basically, the control unit of the NEU gets the numeric instructions from the control unit of the coprocessor.
- The 8087 has a total of eight registers of 80 bits each and these are used in the LIFO stack.
- The operands which the coprocessor instructions will take, reside in the register stack.
- The current top of the stack is pointed by the 3-bit stack pointer which holds the binary values from **000** to **111** so as to show the eight registers of the stack.
- It operates in a circular stack manner in LIFO mode. However, when reset action takes place then the pointer is initialized with 000 of the binary value.
- The three classifications of numeric data over which the coprocessor operates are binary integers, packed decimal numbers, and real numbers.
- The binary integers can be of three types namely word integer (16-bit), short integer (32-bit), and long integer (64-bit).
- BCD format of 80 bit represents the packed decimal numbers whereas real numbers are of three types namely short real (32-bit), long real (64-bit), temporary real (80-bit).

Units of 8087

- **In order to transfer the numeric data within the coprocessor either a 64-bit mantissa bus or 16-bit exponent bus is used.**
- The math processor has a 16-bit control word and a 16-bit status word.
- The control word is to be written into the control register and this takes place in a way that the processor first writes the control word in the memory location and then the coprocessor reads the control word from the memory location and store it into the control register.
- In a similar way, the status word is read in a way that the coprocessor sends the data within the status register to a memory location, and further the processor reads that status register from that particular memory location.
- **This means the microprocessor and coprocessor communicate with each other through the main memory.**

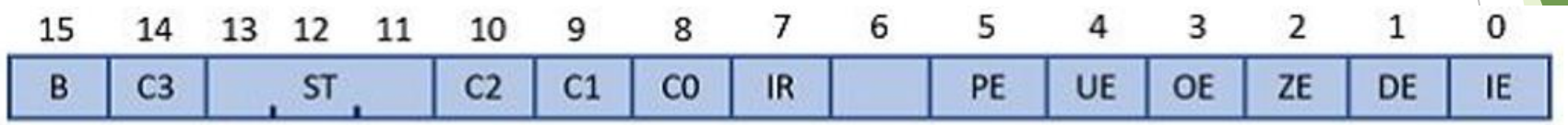
Control word of 8087

The figure below represents the format of the control word:

Format of Control Word of 8087

- IM/IE: Invalid Operation
- DM/DE: Denormalized Operand
- ZM/ZE: Zero Divide
- OM/OE: Overflow
- UM/UE: Underflow
- PM/PE: Precision
- Blank: Reserved
- IEM: Interrupt Enable Mask
- PC: Precision Control
- RC: Rounding Control
- IC: Infinity Control

Status word of 8087



Format of Status Word of 8087

- IR: Interrupt Request
- C0, C1, C2, C3: Condition Code
- ST: Stack Top Pointer
- B: Busy

Operation of 8087

How Coprocessor 8087 works with 8086?

- There is a separate set of instructions of the 8087 coprocessor.
- When the microprocessor works in conjunction with the coprocessor then at the time of writing the program the instructions of both microprocessor and 8087 are included in it.
- In the assembly language program, the instructions that have F, in the beginning, represent the coprocessor instructions, and those without a prefix ‘F’ shows the microprocessor instructions.

Operation of 8087

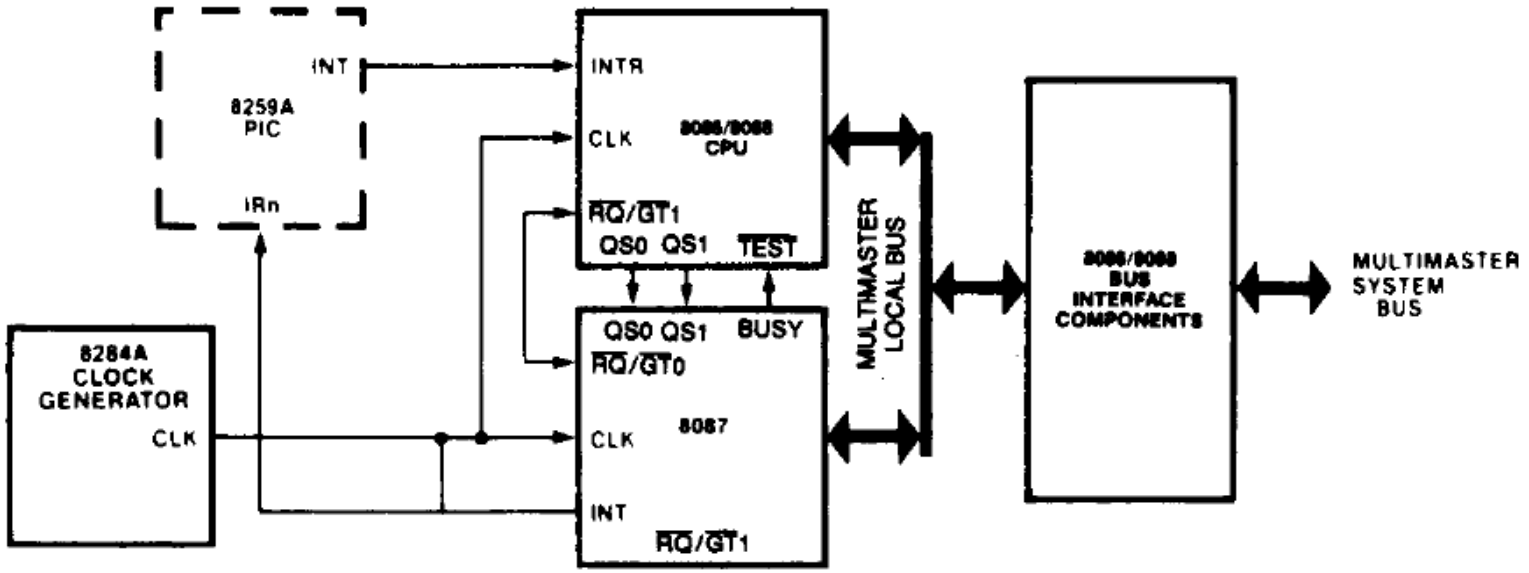
How Coprocessor 8087 works with 8086?

- First, the microprocessor fetches the instructions from the memory location and loads them sequentially in the queue,
- Simultaneously, 8087 also reads and stores the instructions in an internal queue.
- This means every single instruction is read by both processor and coprocessor but at the time of execution both microprocessor and coprocessor perform the execution of their respective instructions.
- This means that instruction is read and decoded and after decoding if the microprocessor checks that it is a coprocessor instruction then that instruction is treated as NOP i.e., No-operation.
- Likewise, if the coprocessor comes across any microprocessor instruction then it will be handled as no-operation.

Overview of 8087 System

- 8087 provides Arithmetic and logical support to a variety of numeric data types.
- It also executes tangent and log functions.
- Executes instructions as a coprocessor when CPU is operating as MX mode processor.
- It extends the Registers, instruction set , brings along the various data types & controls at the hardware level.
- At the programmers level CPU & 8087 are viewed as a single unified processor.

Connection of 8087 to the CPU



8087 is wired in parallel to the CPU.CPU's S0-S2 lines & QS0-QS1(queue status)lines enable 8087 to monitor and decode instructions in Synchronization with the CPU.

- Once started 8087 can process in parallel with & independent of the host CPU.
- BUSY signal used for resynchronization.
- Wait inst. used by CPU to test BUSY signal to insure subsequent 8087 instructions.
- 8087 can interrupt the CPU when error/exception is detected.
- Interrupt request is routed thru the 8259 PIC.