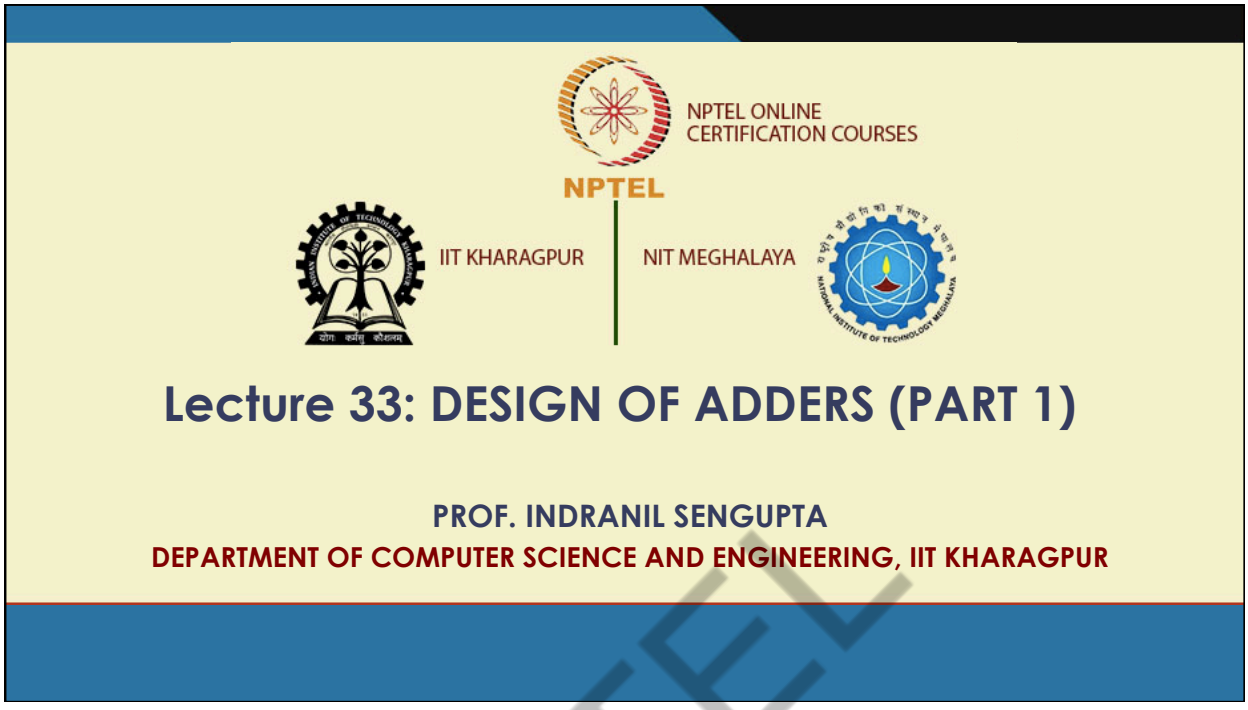
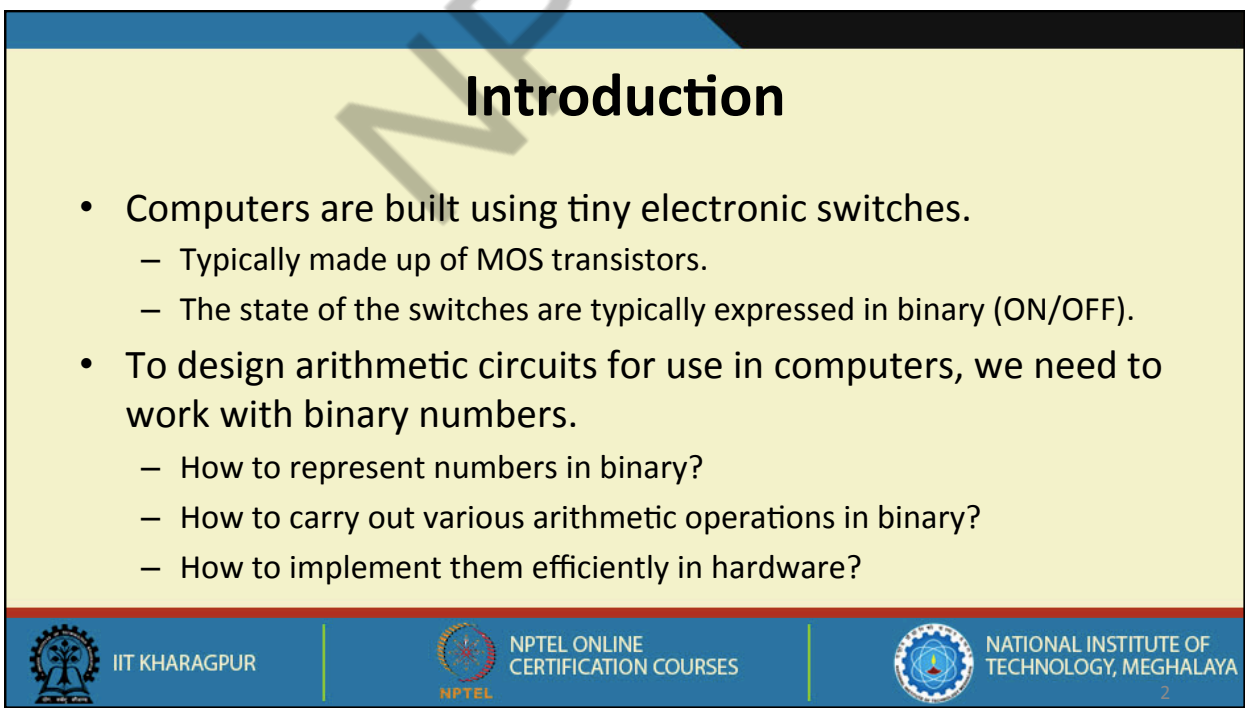

NPTEL ONLINE
CERTIFICATION COURSES
NPTEL
IIT KHARAGPUR
NIT MEGHALAYA
Lecture 33: DESIGN OF ADDERS (PART 1)
PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR


Introduction
• Computers are built using tiny electronic switches.
– Typically made up of MOS transistors.
– The state of the switches are typically expressed in binary (ON/OFF).
• To design arithmetic circuits for use in computers, we need to work with binary numbers.
– How to represent numbers in binary?
– How to carry out various arithmetic operations in binary?
– How to implement them efficiently in hardware?
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
2

NPTEL

Representation of Integers

- Unsigned integer number representation
  - For n-bit binary, range is 0 to (2^n-1).
- Signed integer number representation
  - For n-bit 1's complement, range is $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$.
  - For n-bit 2's complement, range is -2^{n-1} to $+(2^{n-1}-1)$.
  - For both the representations, subtraction can be done using addition.
  - 2's complement representation is most widely used.

NPTEL

Subtraction Using Addition :: 1's Complement

- How to compute A – B ?
  - Compute the 1's complement of B (say, B_1).
  - Compute R = A + B_1
  - If a carry is obtained after addition is '1':
    - Add the carry back to R (called *end-around carry*).
    - That is, R = R + 1.
    - The result is a positive number.

  Else
    - The result is negative, and is in 1's complement form in R.

Example 1 :: 6 – 2

1's complement of 2 = 1101

6 :: 0110
-2 :: 1101
1 0011
1
End-around carry
0100 = +4

Assume 4-bit representations.

Since there is a carry, it is added back to the result.

The result is positive.

Example 2 :: 3 – 5

1's complement of 5 = 1010

3 :: 0011
-5 :: 1010
1101 = -2

Assume 4-bit representations.

Since there is no carry, the result is negative.

1101 is the 1's complement of 0010, that is, it represents –2.

NPTEL

Subtraction Using Addition :: 2's Complement

- How to compute A – B ?
  - Compute the 2's complement of B (say, B_2).
  - Compute R = A + B_2
  - If a carry is obtained after addition is '1':
    - Ignore the carry.
    - The result is a positive number.

  Else
    - The result is negative, and is in 2's complement form in R.

Example 1 :: 6 – 2

2's complement of 2 = 1101 + 1 = 1110

6 :: 0110
-2 :: 1110
1 0100 = +4

Ignore carry

Assume 4-bit representations.

Presence of carry indicates that the result is positive.

No need to add the end-around carry like in 1's complement.

NPTEL

Example 2 :: 3 – 5

2's complement of 5 = 1010 + 1 = 1011

3 :: 0011
-5 :: 1011
1110 = -2

Assume 4-bit representations.

Since there is no carry, the result is negative.

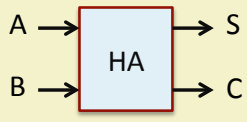
1110 is the 2's complement of 0010, that is, it represents –2.

Addition of Two Binary Digits (Bits)

- When two bits A and B are added, a sum (S) and carry (C) are generated as per the following truth table:

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

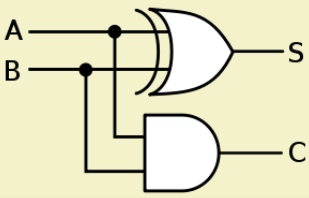
0 + 0 = 00
0 + 1 = 01
1 + 0 = 01
1 + 1 = 10



HALF ADDER

$S = A'.B + A.B' = A \oplus B$

$C = A.B$



NPTEL

Addition of Multi-bit Binary Numbers

0 0 1 0 1 1 0	←	Carry
0 1 0 1 0 1 1	←	Number A
+ 0 0 0 1 0 0 1	←	Number B
0 1 1 0 1 0 0	←	Sum S

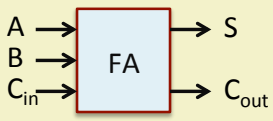
1 1 1 1 1 1 0	←	Carry
0 1 1 1 1 1 1	←	Number A
+ 0 0 0 0 0 0 1	←	Number B
1 0 0 0 0 0 0	←	Sum S

- At every bit position (stage), we require to add 3 bits:
  - 1 bit for number A
  - 1 bit for number B
  - 1 carry bit coming from the previous stage

WE NEED A FULL ADDER

Full Adder

Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

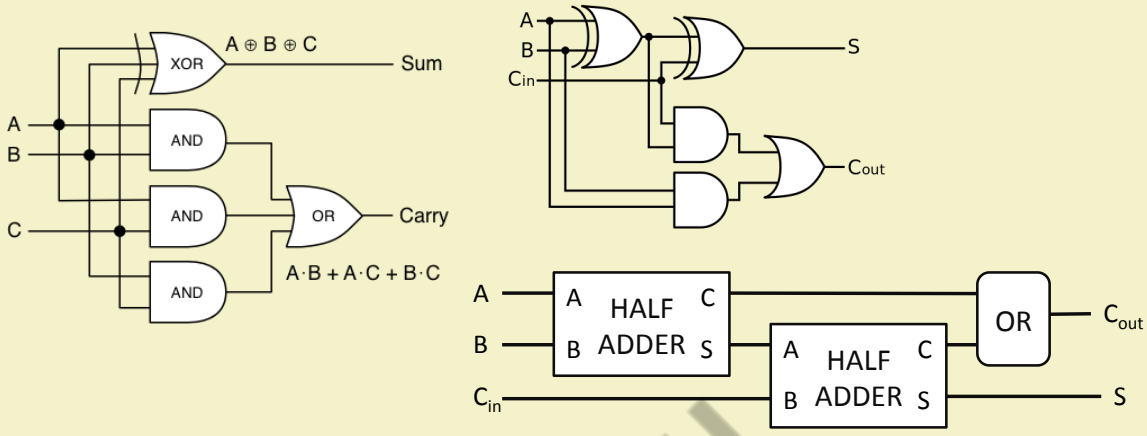


$S = A'.B'.C_{in} + A'.B.C_{in}' + A.B'C_{in}' + A.B.C$
$= A \oplus B \oplus C_{in}$

$C_{out} = B.C_{in} + A.C_{in} + A.B + A.B.C_{in}$
$= A.B + B.C_{in} + A.C_{in}$

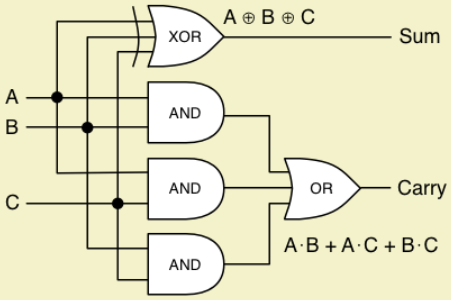
NPTEL

Various Implementations of Full Adder



- Delay of a full adder:
  - Assume that the delay of all basic gates (AND, OR, NAND, NOR, NOT) is δ.
  - Delay for Carry = 2δ
  - Delay for Sum = 3δ

    (AND-OR delay plus one inverter delay)



NPTEL

Parallel Adder Design

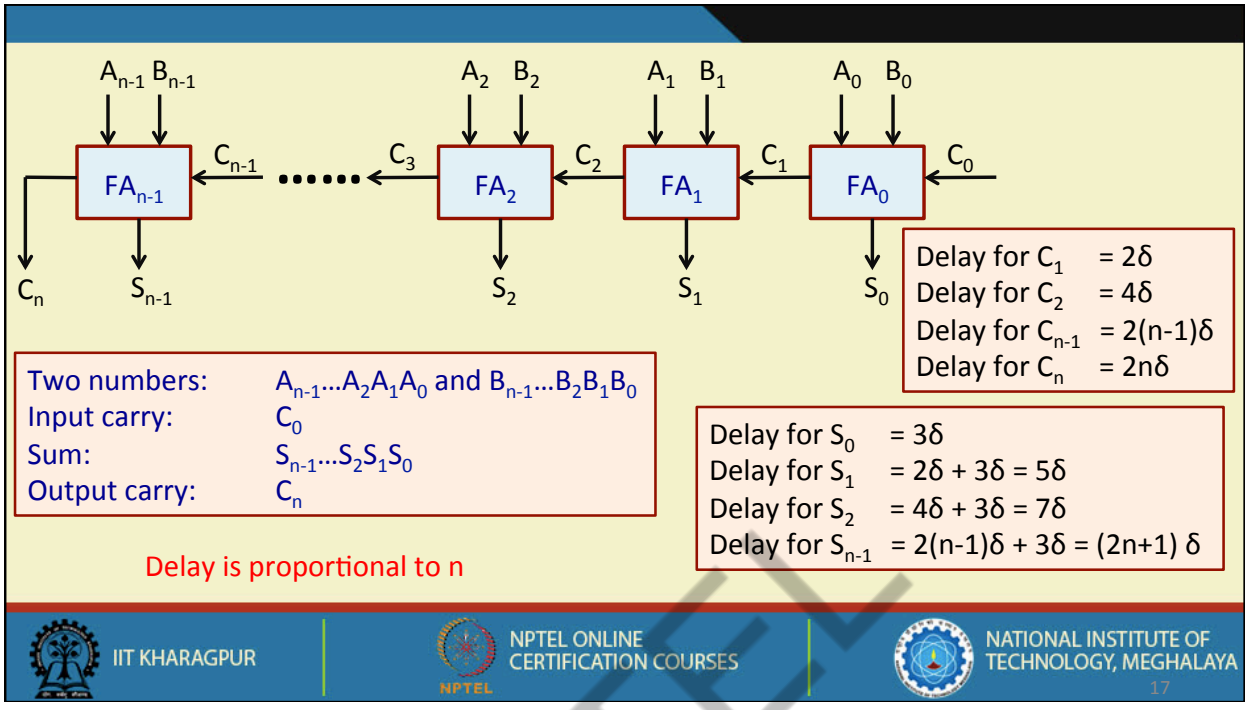
- We shall look at the various designs of n-bit parallel adder.
  a) Ripple carry adder
  b) Carry look-ahead adder
  c) Carry save adder
  d) Carry select adder

NPTEL

Ripple Carry Adder

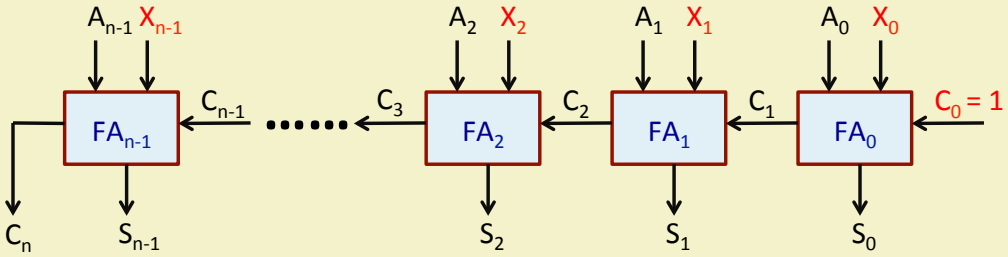
- Cascade n full adders to create a n- bit parallel adder.
- Carry output from stage-i propagates as the carry input to stage-(i+1).
- In the worst-case, carry ripples through all the stages.

```
  1 1 1 1 1 1 0  <-- Carry
  0 1 1 1 1 1 1  <-- Number A
+ 0 0 0 0 0 0 1  <-- Number B
-----------------
  1 0 0 0 0 0 0  <-- Sum S
```



How to Design a Parallel Subtractor?

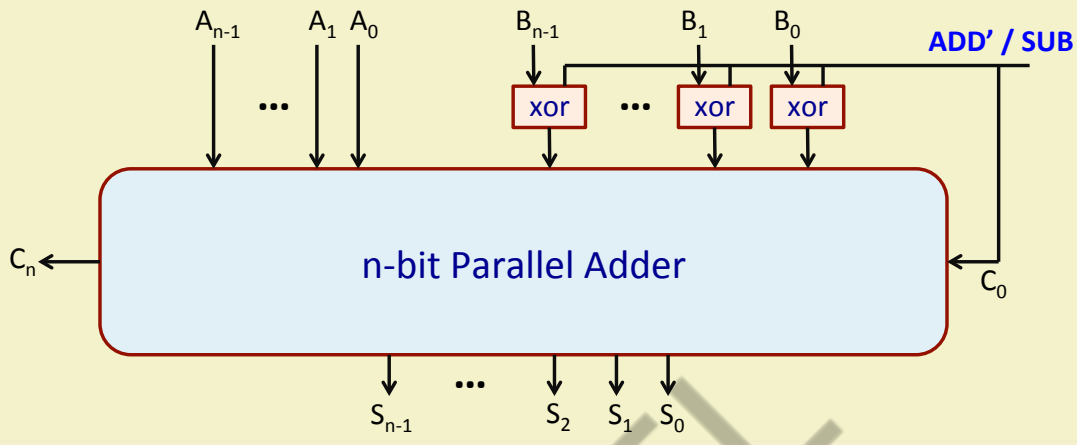
- Observation:
  - Computing A-B is the same as adding the 2's complement of B to A.
  - 2's complement is equal to 1's complement plus 1.
  - Let $X_i = B_i'$.



IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
18

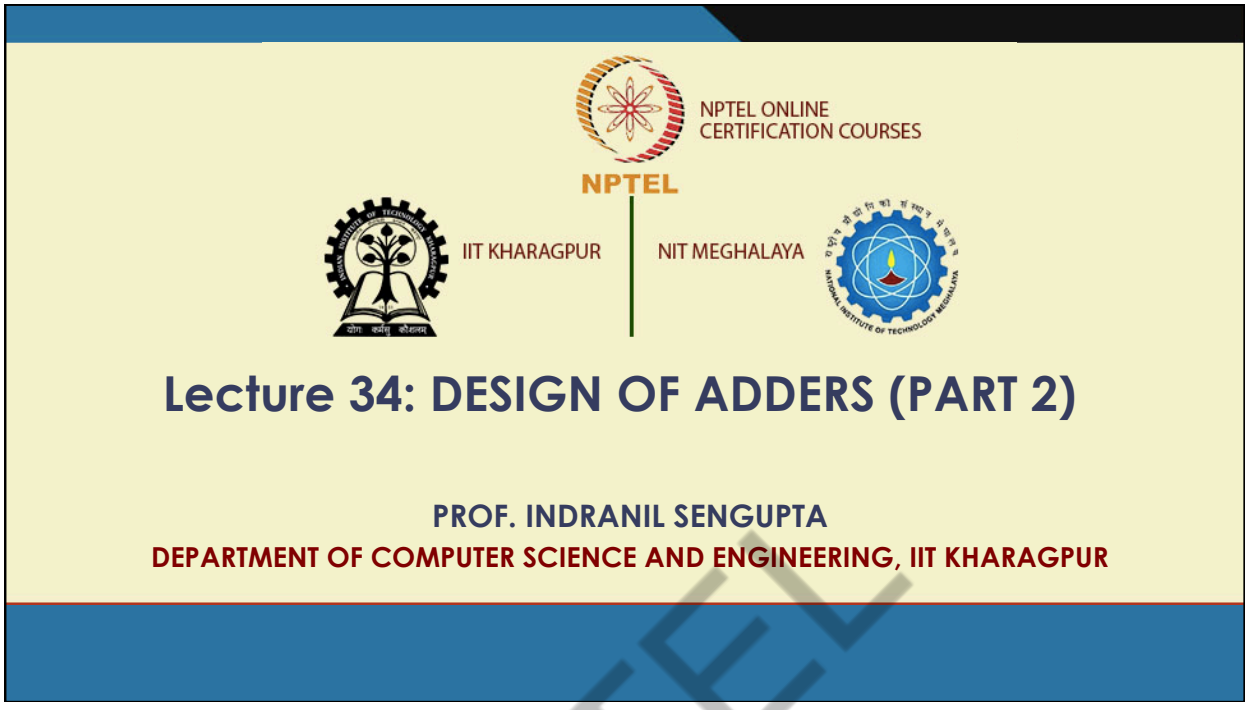
NPTEL

A Parallel Adder/Subtractor



NPTEL

END OF LECTURE 33



NPTEL

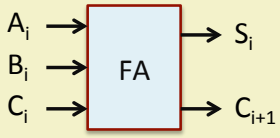
Carry Look-ahead Adder

- The propagation delay of an n-bit ripple carry order has been seen to be proportional to n.
  - Due to the rippling effect of carry sequentially from one stage to the next.
- One possible way to speedup the addition.
  - Generate the carry signals for the various stages in parallel.
  - Time complexity reduces from O(n) to O(1).
  - Hardware complexity increases rapidly with n.

- Consider the i-th stage in the addition process.
- We define the *carry generate* and *carry propagate* functions as:

  $G_i = A_i.B_i$

  $P_i = A_i \oplus B_i$
- $G_i = 1$ represents the condition when a carry is generated in stage-i independent of the other stages.
- $P_i = 1$ represents the condition when an input carry C_i will be propagated to the output carry C_{i+1}.



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

Unrolling the Recurrence

$$C_{i+1} = G_i + P_iC_i = G_i + P_i(G_{i-1} + P_{i-1}C_{i-1}) = G_i + P_iG_{i-1} + P_iP_{i-1}C_{i-1}$$
$$= G_i + P_iG_{i-1} + P_iP_{i-1}(G_{i-2} + P_{i-2}C_{i-2})$$
$$= G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + P_iP_{i-1}P_{i-2}C_{i-2} = \ldots\ldots$$

$$C_{i+1} = G_i + \sum_{k=0}^{i-1} G_k \prod_{j=k+1}^{i} P_j + C_0 \prod_{j=0}^{i} P_j$$

NPTEL

Design of 4-bit CLA Adder

$C_4 = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$

$C_3 = G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$

$C_2 = G_1 + G_0P_1 + C_0P_0P_1$

$C_1 = G_0 + C_0P_0$

$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$

$S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

$S_3 = P_3 \oplus C_3$

4 AND2 gates
3 AND3 gates
2 AND4 gates
1 AND5 gate
1 OR2, 1 OR3, 1 OR4 and 1 OR5 gate

4 XOR2 gates

NPTEL

Design of 4-bit CLA Adder

$C_4 = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$

$C_3 = G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$

$C_2 = G_1 + G_0P_1 + C_0P_0P_1$

$C_1 = G_0 + C_0P_0$

$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$

$S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

$S_3 = P_3 \oplus C_3$

4 AND2 gates
3 AND3 gates
2 AND4 gates
1 AND5 gate
1 OR2, 1 OR3, 1 OR4 and 1 OR5 gate

4 XOR2 gates

Design of 4-bit CLA Adder

$C_4 = G_3 + C_3P_3$

$C_3 = G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$

$C_2 = G_1 + G_0P_1 + C_0P_0P_1$

$C_1 = G_0 + C_0P_0$

$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$

$S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

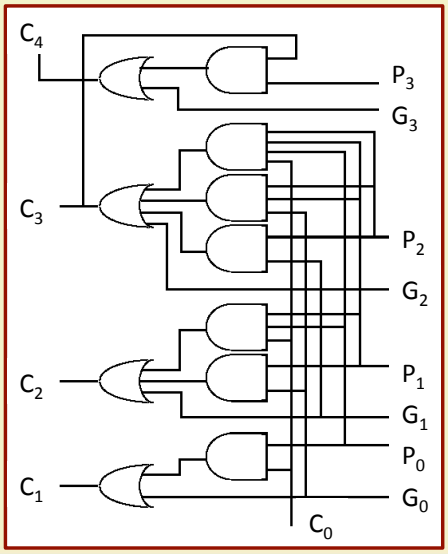
$S_3 = P_3 \oplus C_3$

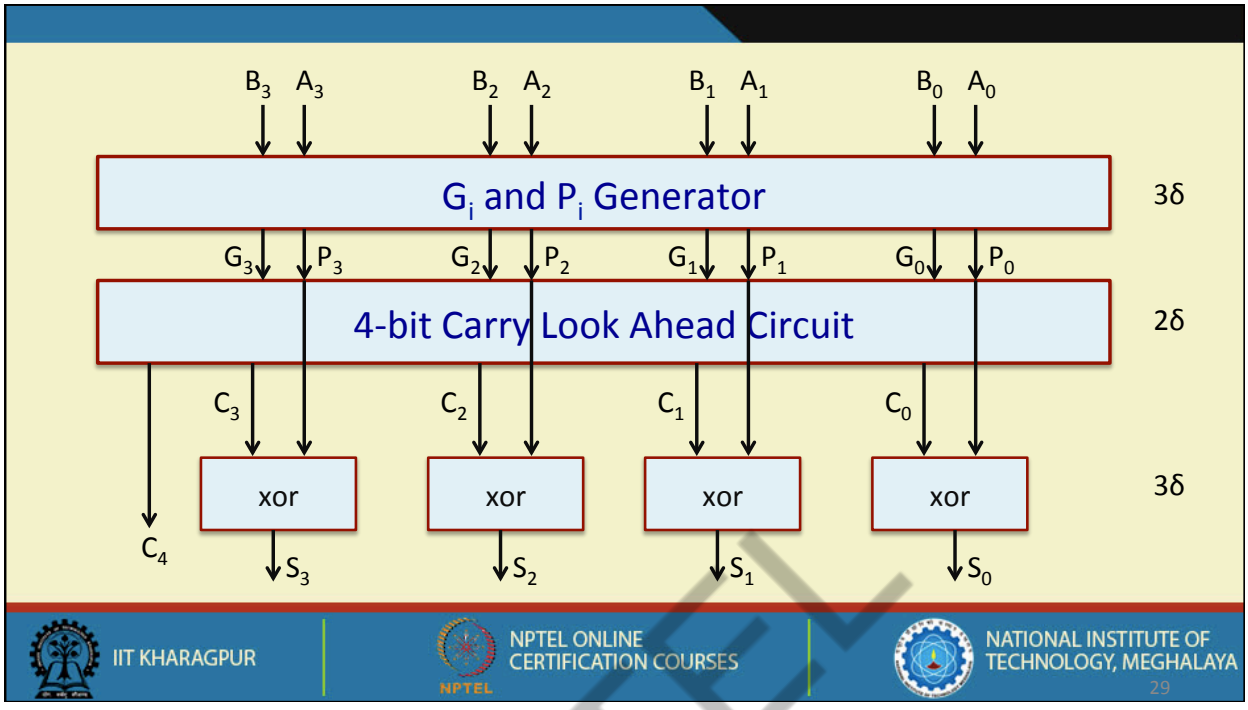
4 AND2 gates
2 AND3 gates
1 AND4 gates
~~1 AND5 gate~~
1 OR2, 1 OR3, 1 OR4 and 1 **OR2** gate

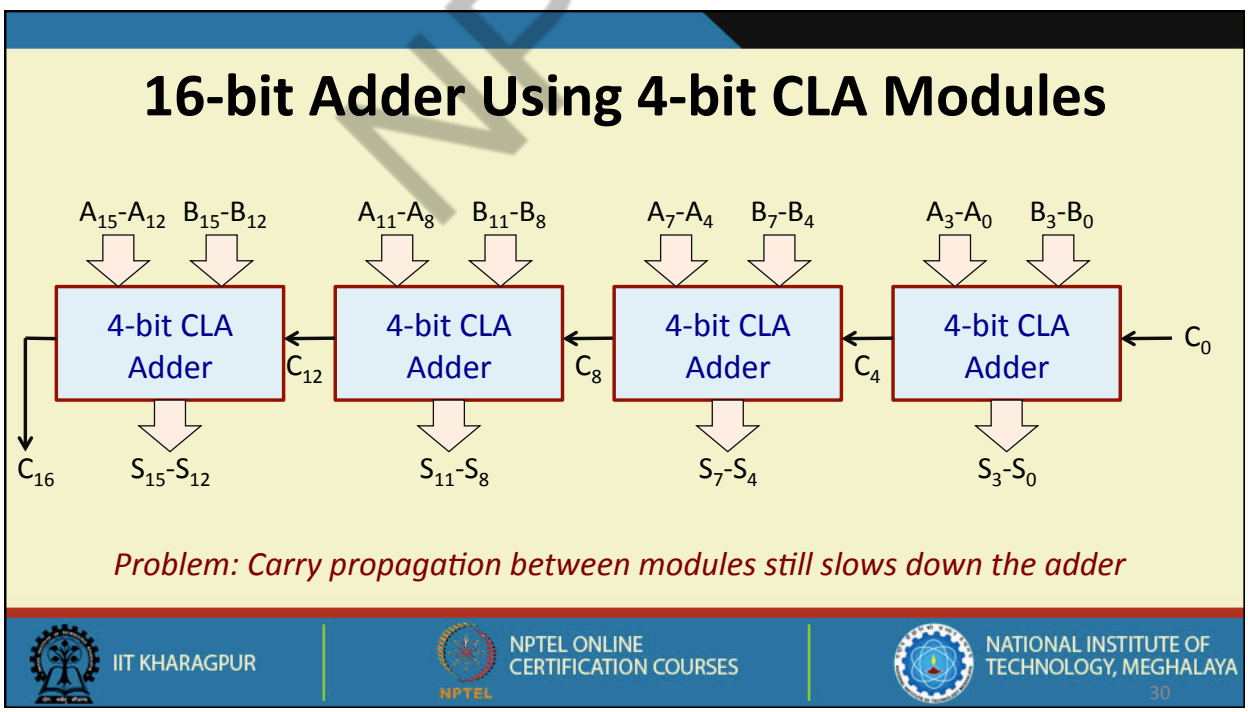
4 XOR2 gates

NPTEL

The 4-bit CLA Circuit




B3 A3
B2 A2
B1 A1
B0 A0
Gi and Pi Generator
3δ
G3 P3
G2 P2
G1 P1
G0 P0
4-bit Carry Look Ahead Circuit
2δ
C3
C2
C1
C0
xor
xor
xor
xor
3δ
C4
S3
S2
S1
S0
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
29


16-bit Adder Using 4-bit CLA Modules
A15-A12 B15-B12
A11-A8 B11-B8
A7-A4 B7-B4
A3-A0 B3-B0
4-bit CLA Adder
4-bit CLA Adder
4-bit CLA Adder
4-bit CLA Adder
C0
C12
C8
C4
C16
S15-S12
S11-S8
S7-S4
S3-S0
Problem: Carry propagation between modules still slows down the adder
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
30

NPTEL

- Solution:
  - Use a second level of carry look-ahead mechanism to generate the input carries to the CLA blocks in parallel.
  - The second level of CLA generates C4, C8, C12 and C16 in parallel with two gate delays (2δ).
  - For larger values of n, more CLA levels can be added.
- Delay calculation of a 16-bit adder:
  a) For original single-level CLA: 14δ
  b) For modified two-level CLA: 10δ

Delay of a k-bit Adder

n	T_{CLA}	T_{RCA}
4	8δ	9δ
16	10δ	33δ
32	12δ	65δ
64	12δ	129δ
128	14δ	257δ
256	14δ	513δ

$$T_{CLA} = (6 + 2\lceil \log_4 n \rceil)\ \delta$$

$$T_{RCA} = (2n + 1)\ \delta$$

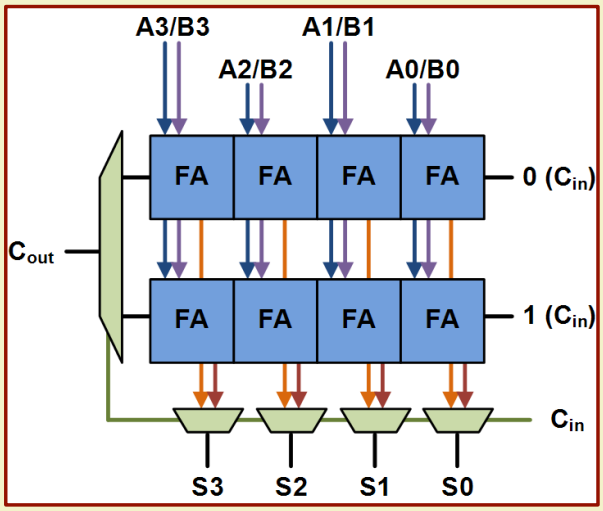
Carry Select Adder

- Basically consists of two parallel adders (say, ripple-carry adder) and a multiplexer.
- For two given numbers A and B, we carry out addition twice:
  - With carry-in as 0
  - With carry-in as 1
- Once the correct carry-in is known, the correct sum is selected by a multiplexer.

NPTEL

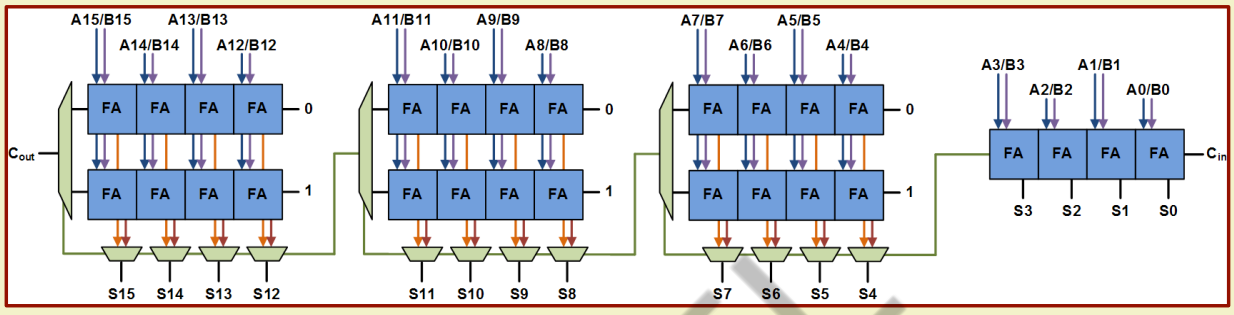
Basic building block of a carry-select adder, with block size of 4.

- For a multi-bit adder, the number of bits in each carry select block can be either uniform or variable.



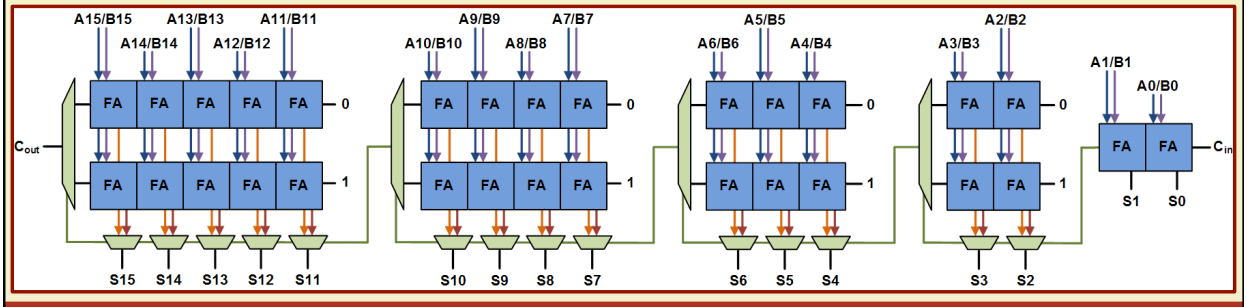
Uniform sized adder

- A 16-bit carry select adder with a uniform block size of 4 is shown.
- The least significant block needs a single adder (since the carry-in is known).
- Total delay is 4 full adder delays, plus 3 MUX delays.



Variable-sized adder

- A 16-bit carry select adder with variable block sizes of 2-2-3-4-5 is shown.
- Total delay is 2 full adder delays, plus 4 MUX delays.



NPTEL

Carry Save Adder

- Here we add three operands (say, X, Y and Z) together.
- For adding multiple numbers, we have to construct a tree of carry save adders.
  - Used in combinational multiplier design.
- Each carry save adder is simply an independent full adder without carry propagation.
- A parallel adder is required only at the last stage.

NPTEL

- An illustrative example:

X:	1 0 0 1 1
Y:	+ 1 1 0 0 1
Z:	+ 0 1 0 1 1
C:	1 1 0 1 1

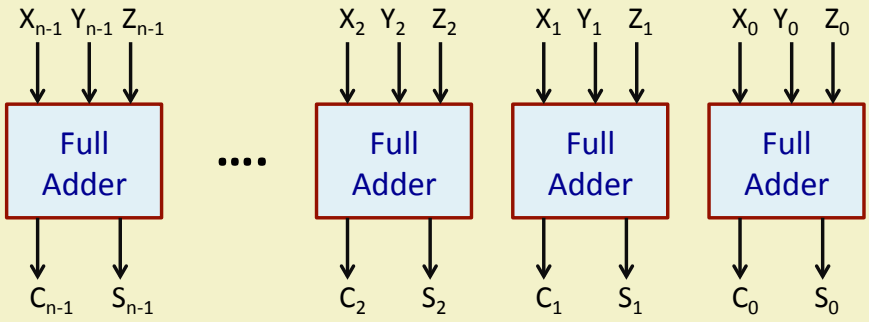
X:	1 0 0 1 1
Y:	+ 1 1 0 0 1
Z:	+ 0 1 0 1 1
S:	0 0 0 0 1

X:	1 0 0 1 1
Y:	+ 1 1 0 0 1
Z:	+ 0 1 0 1 1
S:	0 0 0 0 1
C:	1 1 0 1 1
Sum:	1 1 0 1 1 1

A set of full adders generate carry and sum bits in parallel

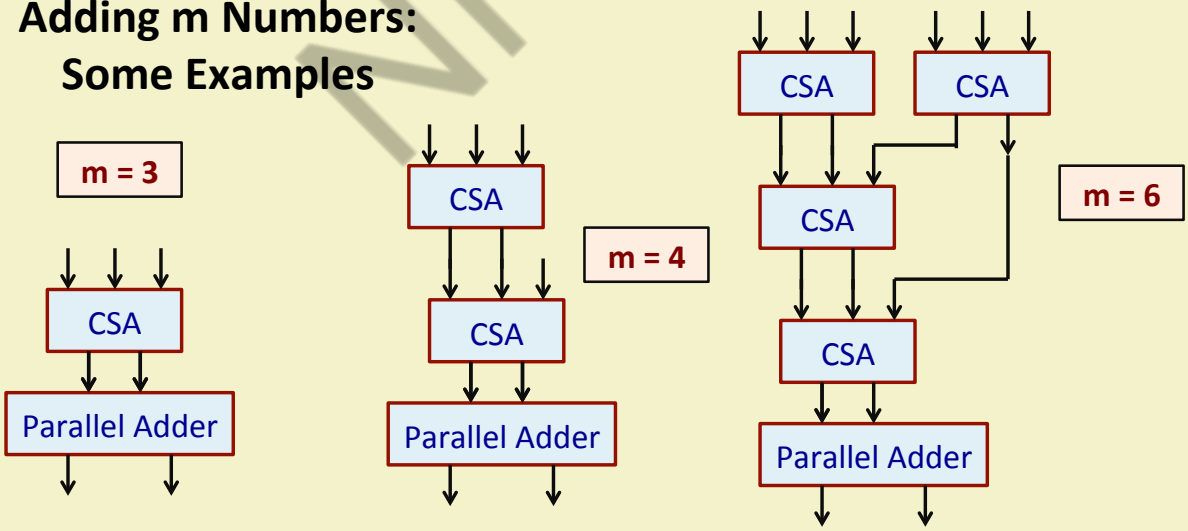
The sum and carry vectors are added later (with proper shifting)

An n-bit Carry Save Adder



The carry input of the full adder is used as the third input

Adding m Numbers: Some Examples



NPTEL

END OF LECTURE 34

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

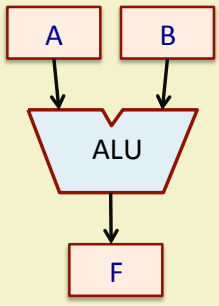
IIT KHARAGPUR | NIT MEGHALAYA

Lecture 35: DESIGN OF MULTIPLIERS (PART 1)

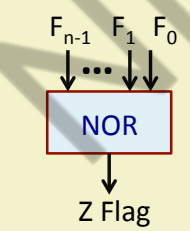
PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR

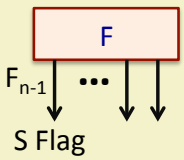
Generating the Status Flags

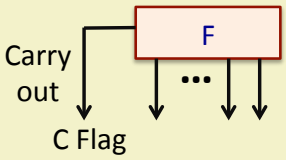
- Many contemporary processors have a flag register that contains the status of the last arithmetic / logic operation.
  - **Zero (Z)**: tells whether the result is zero.
    - Can be used for both arithmetic and logic operations.
  - **Sign (S)**: tells whether the result is positive (=0) or negative (=1).
    - Can be used for both arithmetic and logic operations.
  - **Carry (C)**: tells whether there has been a carry out of the most significant stage.
    - Used only for arithmetic operations.
  - **Overflow (V)**: tells whether the result is too large to fit in the target register.
    - Used only for arithmetic operations (addition and subtraction).



Assume A, B and F are n-bit registers







- Overflow can occur during addition when the sign of the two operands are the same.
  - ✧ Sign of the result becomes different from the sign of the operand(s).

$V = A_{n-1}.B_{n-1}.F_{n-1}' + A_{n-1}'.B_{n-1}'.F_{n-1}$

$V = F_{n-1} \oplus Carry\_out$

NPTEL

- The MIPS architecture does not have any status flags.
- Why?
  - MIPS ISA is designed for efficient pipeline implementation.
  - Several instructions can be in various stages of execution in the pipeline.
  - Flag registers result in ***side effects*** among instructions.
- MIPS stores information about the flags temporarily in a GPR.

```
slt     $t0, $s1, $s2
beq     $t0, $zero, Label
```


Multiplication of Unsigned Numbers

- Multiplication requires substantially more hardware than addition.
- Multiplication of two n-bit number generates a 2n-bit product.
- We can use shift-and-add method.
  - Repeated additions of shifted versions of the multiplicand.

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

Multiplicand M (10)
Multiplier Q (13)

Product P (130)

NPTEL

A General Case

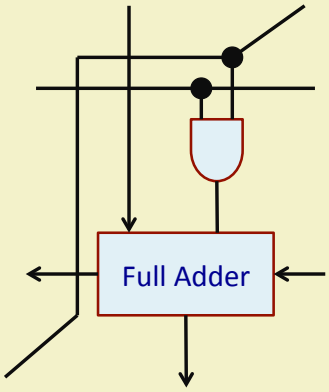
$$
\begin{array}{cccccc}
 & & A_3 & A_2 & A_1 & A_0 \\
 & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & \\
A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & \\
\hline
\end{array}
$$

- Each A_iB_j is called a partial product.
- Generating the partial products is easy.
  - Requires just an AND gate for each partial product.
- Adding all the n-bit partial products in hardware is more difficult.

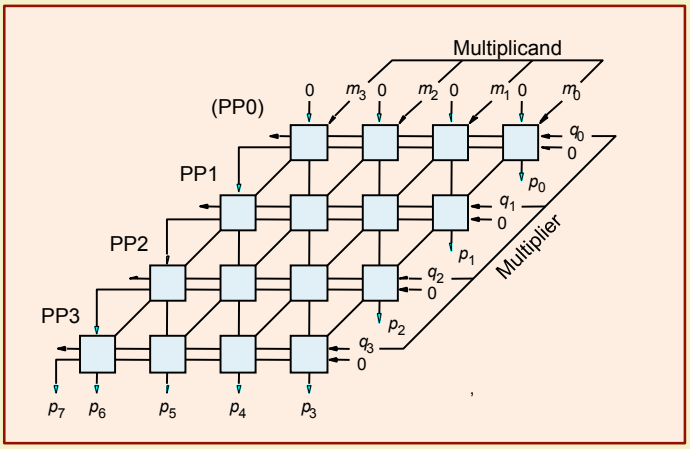
NPTEL

Design of a Combinational Array Multiplier

- We can directly map the multiplication process as discussed to hardware.
  - We use an array of cells to generate the partial products.
  - Instead of adding the partial products at the end, we add the partial products at every stage of the multiplication.
- The required multiplication cell is as shown.
  - Combines capabilities of partial product generation and also addition of partial products.



- Extremely inefficient, and requires very large amount of hardware.
- n^2 multiplication cells for an n x n multiplier.
- Advantage is that it is very fast.



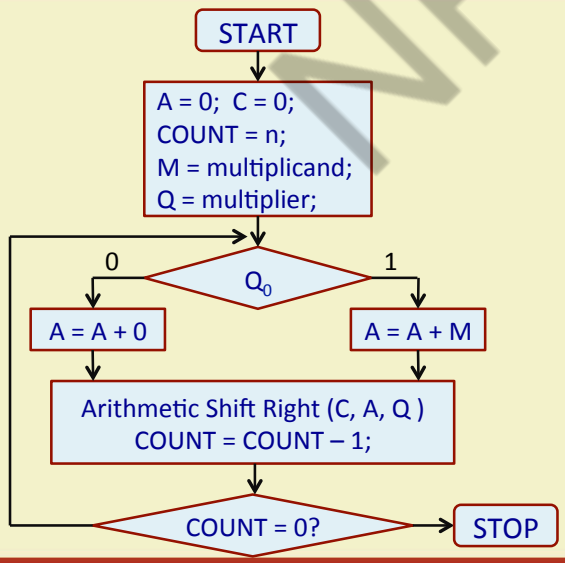
Product: $p_7\ p_6\ p_5\ p_4\ p_3\ p_2\ p_1\ p_0$

Unsigned Sequential Multiplication

- Requires much less hardware, but requires several clock cycles to perform multiplication of two n-bit numbers.
  - Typical hardware complexity: O(n).
  - Typical time complexity: O(n).
- In the "*hand multiplication*" that we have seen:
  - If the i-th bit of the multiplier is 1, the multiplicand is shifted left by i bit positions, and added to the partial product.
  - The relative position of the partial products do not change; it is the multiplicand that gets shifted left.

NPTEL

- In the "*shift-and-add*" multiplication that we discuss now, we make the following modifications.
  - We do not shift the multiplicand (i.e., keep its position fixed).
  - We right shift an 2n-bit partial product at every step.



M: n-bit multiplicand

Q: n-bit multiplier

A: n-bit temporary register

C: 1-bit carry out from adder

Example 1: (10) x (13)

Assume 5-bit numbers.

M: $(0\ 1\ 0\ 1\ 0)_2$
Q: $(0\ 1\ 1\ 0\ 1)_2$

Product = 130
= $(0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0)_2$

	C	A	Q		
	0	0 0 0 0 0	0 1 1 0 (1)	Initialization	
Step 1	0	0 1 0 1 0	0 1 1 0 1	A = A + M	
	0	0 0 1 0 1	0 0 1 1 (0)	Shift	
Step 2	0	0 0 1 0 1	0 0 1 1 0	A = A + 0	
	0	0 0 0 1 0	1 0 0 1 (1)	Shift	
Step 3	0	0 1 1 0 0	1 0 0 1 1	A = A + M	
	0	0 0 1 1 0	0 1 0 0 (1)	Shift	
Step 4	0	1 0 0 0 0	0 1 0 0 1	A = A + M	
	0	0 1 0 0 0	0 0 1 0 (0)	Shift	
Step 5	0	0 1 0 0 0	0 0 1 0 0	A = A + 0	
	0	0 0 1 0 0	0 0 0 1 0	Shift	

Example 2: (29) x (21)

Assume 5-bit numbers.

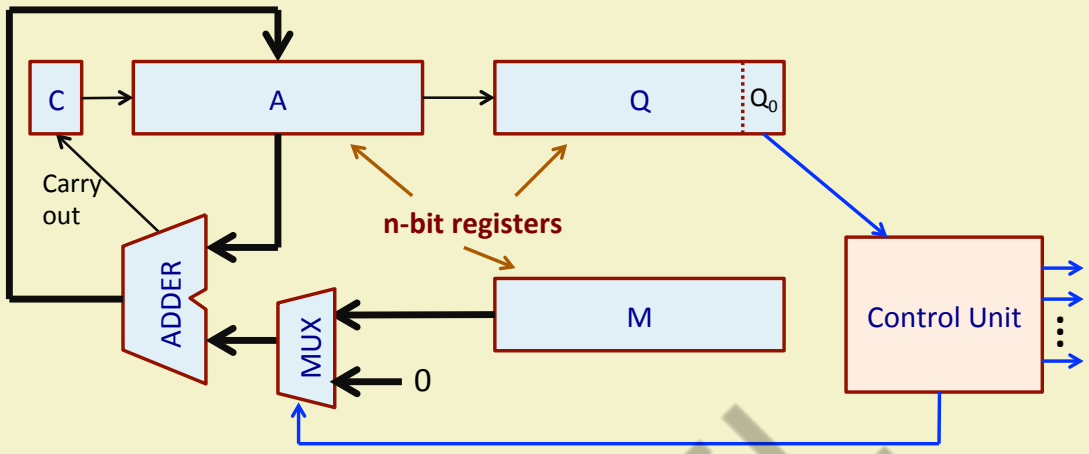
M: $(1\ 1\ 1\ 0\ 1)_2$
Q: $(1\ 0\ 1\ 0\ 1)_2$

Product = 609
= $(1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1)_2$

	C	A	Q		
	0	0 0 0 0 0	1 0 1 0 (1)	Initialization	
Step 1	0	1 1 1 0 1	1 0 1 0 1	A = A + M	
	0	0 1 1 1 0	1 1 0 1 (0)	Shift	
Step 2	0	0 1 1 1 0	1 1 0 1 0	A = A + 0	
	0	0 0 1 1 1	0 1 1 0 (1)	Shift	
Step 3	1	0 0 1 0 0	0 1 1 0 1	A = A + M	
	0	1 0 0 1 0	0 0 1 1 (0)	Shift	
Step 4	0	1 0 0 1 0	0 0 1 1 0	A = A + 0	
	0	0 1 0 0 1	0 0 0 1 (1)	Shift	
Step 5	1	0 0 1 1 0	0 0 0 1 1	A = A + M	
	0	1 0 0 1 1	0 0 0 0 1	Shift	

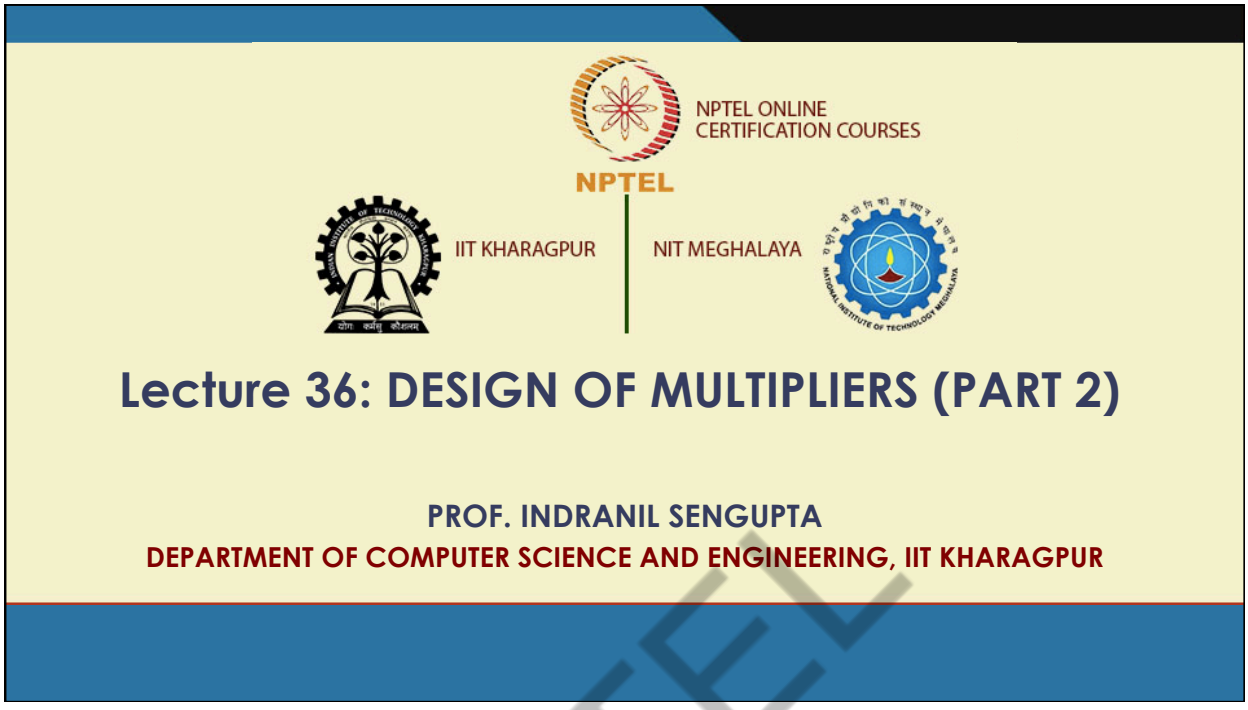
NPTEL

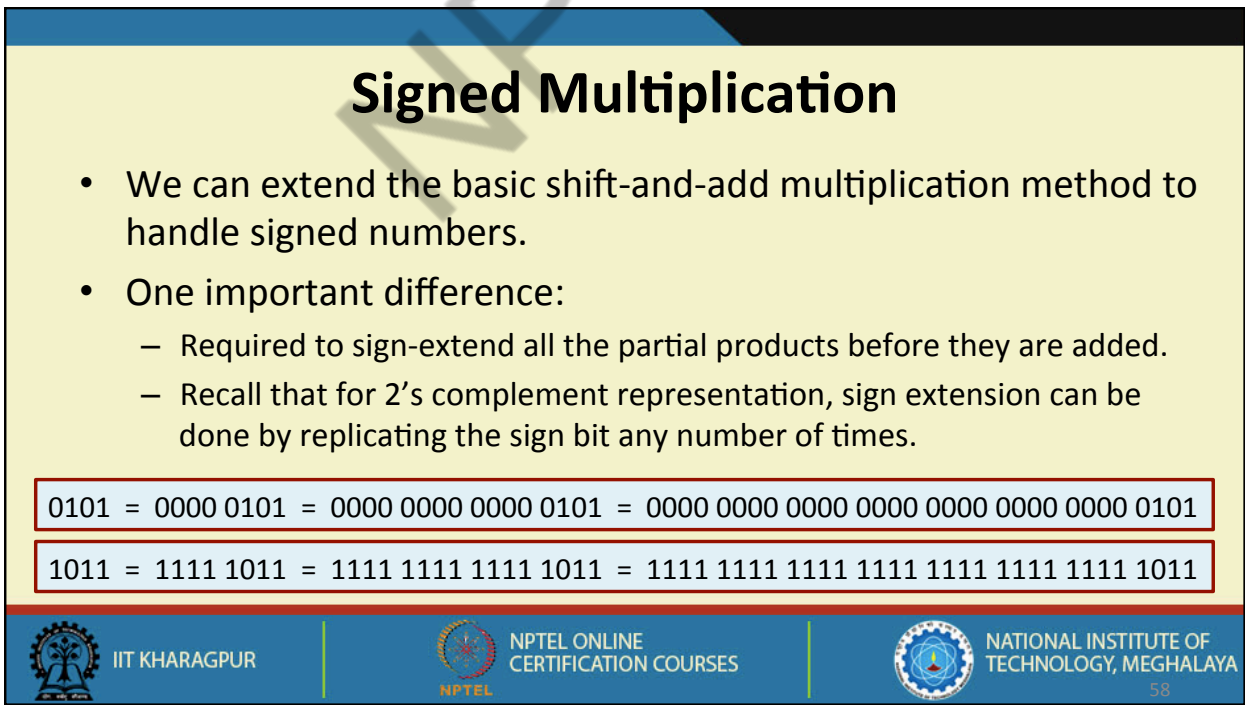
Data Path for Shift-and-Add Multiplier



END OF LECTURE 35

NPTEL


NPTEL ONLINE
CERTIFICATION COURSES
NPTEL
IIT KHARAGPUR
NIT MEGHALAYA
Lecture 36: DESIGN OF MULTIPLIERS (PART 2)
PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR


Signed Multiplication
• We can extend the basic shift-and-add multiplication method to handle signed numbers.
• One important difference:
– Required to sign-extend all the partial products before they are added.
– Recall that for 2's complement representation, sign extension can be done by replicating the sign bit any number of times.
0101 = 0000 0101 = 0000 0000 0000 0101 = 0000 0000 0000 0000 0000 0000 0000 0101
1011 = 1111 1011 = 1111 1111 1111 1011 = 1111 1111 1111 1111 1111 1111 1111 1011
IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES
NATIONAL INSTITUTE OF
TECHNOLOGY, MEGHALAYA
58

An Example: 6-bit 2's complement multiplication

Note: For n-bit multiplication, since we are generating a 2n-bit product, overflow can never occur.

```
                1 1 0 1 0 1     (-11)
              x 0 1 1 0 1 0     (+26)
-------------------------
0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0 1 0 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 1 0 1
1 1 1 1 0 1 0 1
0 0 0 0 0 0 0
-------------------------
1 1 1 0 1 1 1 0 0 0 1 0   (-286)
```


Booth's Algorithm for Signed Multiplication

- In the conventional shift-and-add multiplication as discussed, for n-bit multiplication, we iterate n times.
  - Add either 0 or the multiplicand to the 2n-bit partial product (depending on the next bit of the multiplier).
  - Shift the 2n-bit partial product to the right.
- Essentially we need *n additions and n shift operations*.
- Booth's algorithm is an improvement whereby we can avoid the additions whenever consecutive 0's or 1's are detected in the multiplier.
  - Makes the process faster.

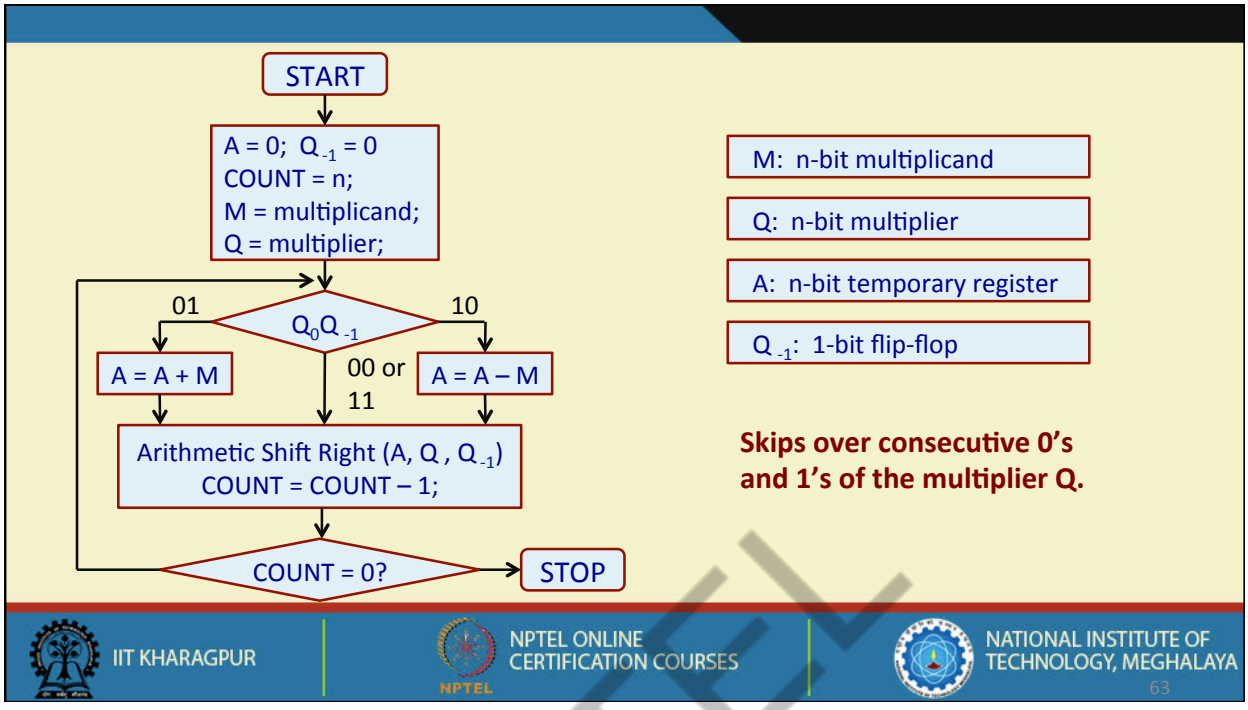
NPTEL

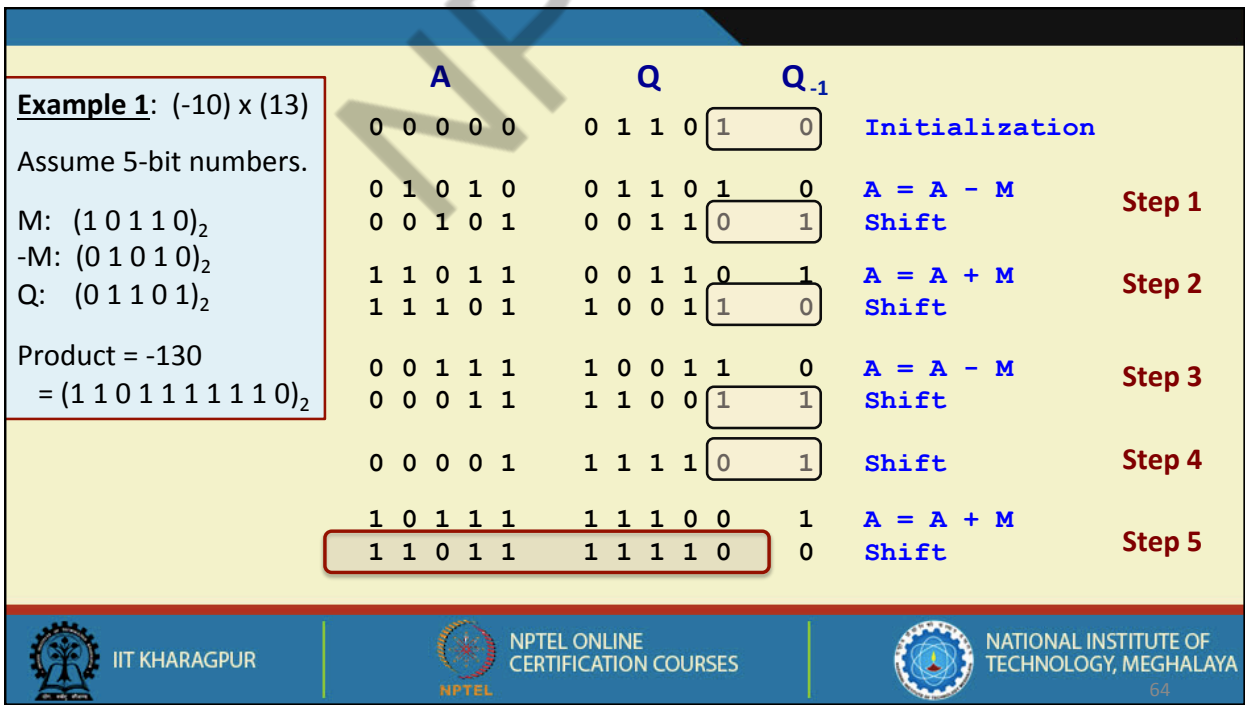
Basic Idea Behind Booth's Algorithm

- We inspect two bits of the multiplier (Q_i, Q_{i-1}) at a time.
  - If the bits are same (00 or 11), we only shift the partial product.
  - If the bits are 01, we do an addition and then shift.
  - If the bits are 10, we do a subtraction and then shift.
- Significantly reduces the number of additions / subtractions.
- Inspecting bit pairs as mentioned can also be expressed in terms of *Booth's Encoding*.
  - Use the symbols +1, -1 and 0 to indicate changes w.r.t. Q_i and Q_{i-1}.
  - 01 → +1, 10 → -1, 00 or 11 → 0.
  - For encoding the least significant bit Q_0, we assume $Q_{-1} = 0$.

NPTEL

- Examples of Booth encoding:
  a) 0 1 1 1 0 0 0 0 :: +1 0 0 -1 0 0 0 0
  b) 0 1 1 1 0 1 1 0 :: +1 0 0 -1 +1 0 -1 0
  c) 0 0 0 0 0 1 1 1 :: 0 0 0 0 +1 0 0 -1
  d) 0 1 0 1 0 1 0 1 :: +1 -1 +1 -1 +1 -1 +1 -1
- The last example illustrates the worst case for Booth's multiplication (alternating 0's and 1's in multiplier).
  - In the illustrations, we shall show the two multiplier bits explicitly instead of showing the encoded digits.


START
A = 0; Q -1 = 0
COUNT = n;
M = multiplicand;
Q = multiplier;
Q0Q -1
01
10
00 or 11
A = A + M
A = A – M
Arithmetic Shift Right (A, Q , Q -1)
COUNT = COUNT – 1;
COUNT = 0?
STOP
M: n-bit multiplicand
Q: n-bit multiplier
A: n-bit temporary register
Q -1: 1-bit flip-flop
Skips over consecutive 0's and 1's of the multiplier Q.
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
63


Example 1: (-10) x (13)
Assume 5-bit numbers.
M: (1 0 1 1 0)2
-M: (0 1 0 1 0)2
Q: (0 1 1 0 1)2
Product = -130
= (1 1 0 1 1 1 1 1 1 0)2
A Q Q -1
0 0 0 0 0 0 1 1 0 1 0 Initialization
0 1 0 1 0 0 1 1 0 1 0 A = A - M
0 0 1 0 1 0 0 1 1 0 1 Shift
Step 1
1 1 0 1 1 0 0 1 1 0 1 A = A + M
1 1 1 0 1 1 1 0 0 1 0 Shift
Step 2
0 0 1 1 1 1 0 0 1 1 0 A = A - M
0 0 0 1 1 1 1 0 0 1 1 Shift
Step 3
0 0 0 0 1 1 1 1 1 0 1 Shift
Step 4
1 0 1 1 1 1 1 1 0 0 1 A = A + M
1 1 0 1 1 1 1 1 1 0 0 Shift
Step 5
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
64

Example 2:

(-31) x (28)

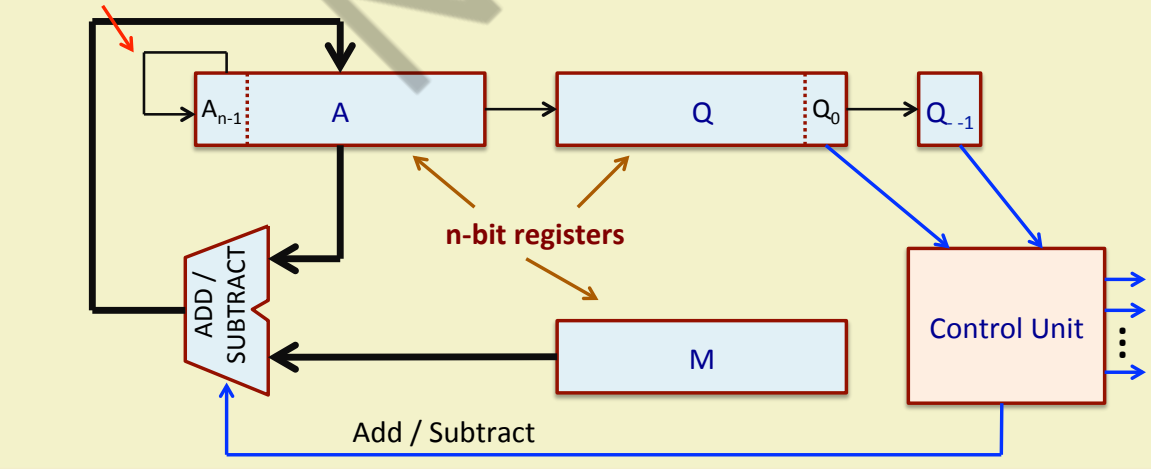
Assume 6-bit numbers.

M: $(1\ 0\ 0\ 0\ 0\ 1)_2$
-M: $(0\ 1\ 1\ 1\ 1\ 1)_2$
Q: $(0\ 1\ 1\ 1\ 0\ 0)_2$

Product = -868
= $(1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0)_2$

A	Q	Q_{-1}		
0 0 0 0 0 0	0 1 1 1 0	0	Initialization	
0 0 0 0 0 0	0 0 1 1 1	0	Shift	Step 1
0 0 0 0 0 0	0 0 0 1 1	1	Shift	Step 2
0 1 1 1 1 1	0 0 0 1 1	1	A = A - M	Step 3
0 0 1 1 1 1	1 0 0 0 1	1	Shift	
0 0 0 1 1 1	1 1 0 0 0	1	Shift	Step 4
0 0 0 0 1 1	1 1 1 0 0	0	Shift	Step 5
1 0 0 1 0 0	1 1 1 0 0 0	1	A = A + M	Step 6
1 1 0 0 1 0	0 1 1 1 0 0	0	Shift	

Data Path for Booth's Algorithm



NPTEL

Design of Fast Multiplier

a) Bit-Pair Recoding of Booth's Multiplication

- A technique that halves the maximum number of summands; derived directly from the Booth's algorithm.
- If we group the Booth-coded multiplier digits in pairs, we observe:
  - (+1, -1): (+1, -1) * M = 2 * M – M = M
  - (0, +1): (0, +1) * M = M
- We need a single addition instead of a pair of addition & subtraction.
  - Other similar rules can be framed.
  - Shown on next slide.

Original Booth-coded Pair	Equivalent Recoded Pair
(+1, 0)	(0, +2)
(-1, +1)	(0, -1)
(0, 0)	(0, 0)
(0, 1)	(0, 1)
(+1, 1)	--
(+1, -1)	(0, +1)
(-1, 0)	(0, -2)

- Every equivalent recoded pair has at least one 0.
- Worst-case number of additions or subtractions is 50% of the number of multiplier bits.
- Reduces the worst-case time required for multiplication.

NPTEL

Example: (+13) X (-22) in 6-bits.

```
Original:  Multiplier --   1   0   1   0   1   0
Booth:     Multiplier --  -1  +1  -1  +1  -1   0
Recoded:   Multiplier --   0  -1   0  -1   0  -2

          0  0  1  1  0  1
          . -1  . -1  . -2
-------------------------
  1 1 1 1 1 1 1 0 0 1 1 0
  1 1 1 1 1 1 0 0 1 1
  1 1 1 1 0 0 1 1
-------------------------
  1 1 0 1 1 1 1 0 0 0 1 0
```

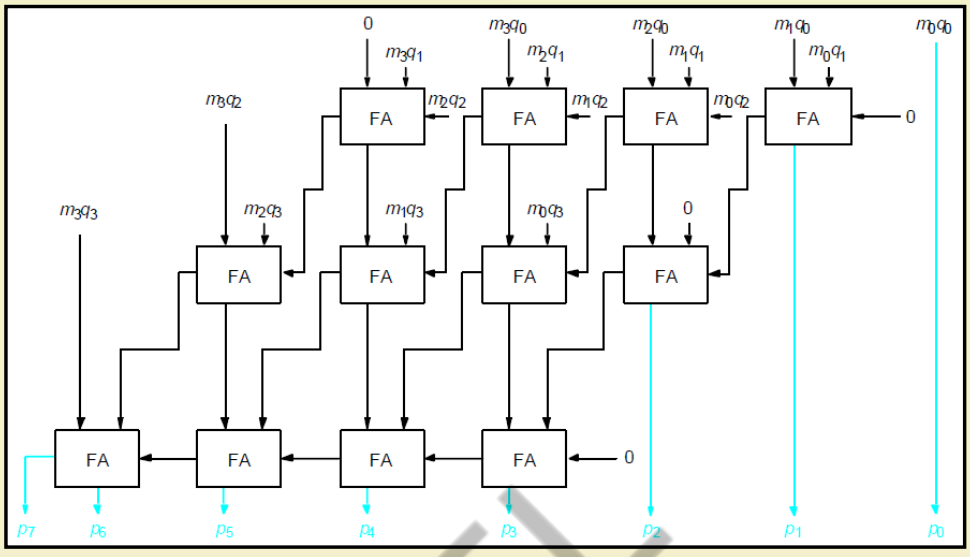
- M = 0 0 1 1 0 1 (+13)
- -1 * M = 1 1 0 0 1 1
- -2 * M = 1 0 0 1 1 0

NPTEL

b) Carry Save Multiplier

- We have seen earlier how carry save adders (CSA) can be used to add several numbers with carry propagation only in the last stage.
- The partial products can be generated in parallel using n^2 AND gates.
- The n partial products can then be added using a CSA tree.
- Instead of letting the carries ripple through during addition, we *save* them and feed it to the next row, at the correct weight positions.

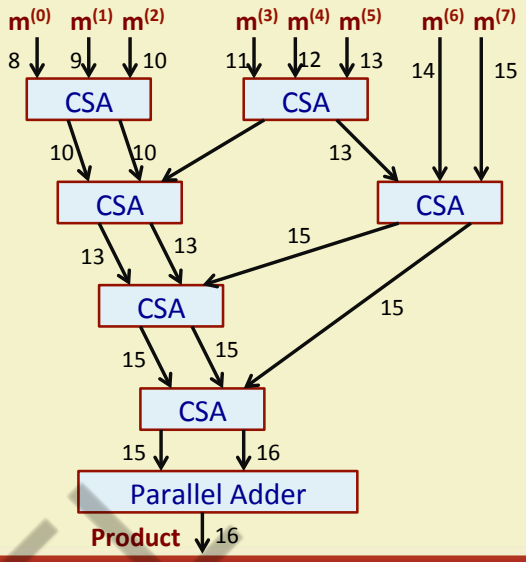
4 x 4 Carry Save Multiplier



- Wallace Tree Multiplier
  - A Wallace tree is a circuit that reduces the problem of summing n n-bit numbers to the problem of summing two Θ(n)-bit numbers.
  - It uses n/3 (floor of) carry-save adders in parallel to convert the sum of n numbers to the sum of 2n/3 (ceiling of) numbers.
  - It then recursively constructs a Wallace tree on the 2n/3 (ceiling of) resulting numbers.
  - The set of numbers is progressively reduced until there are only two numbers left.
  - By performing many carry-save additions in parallel, Wallace trees allow two n-bit numbers to be multiplied in $\Theta(\log_2 n)$ time using a circuit of size $\Theta(n^2)$.

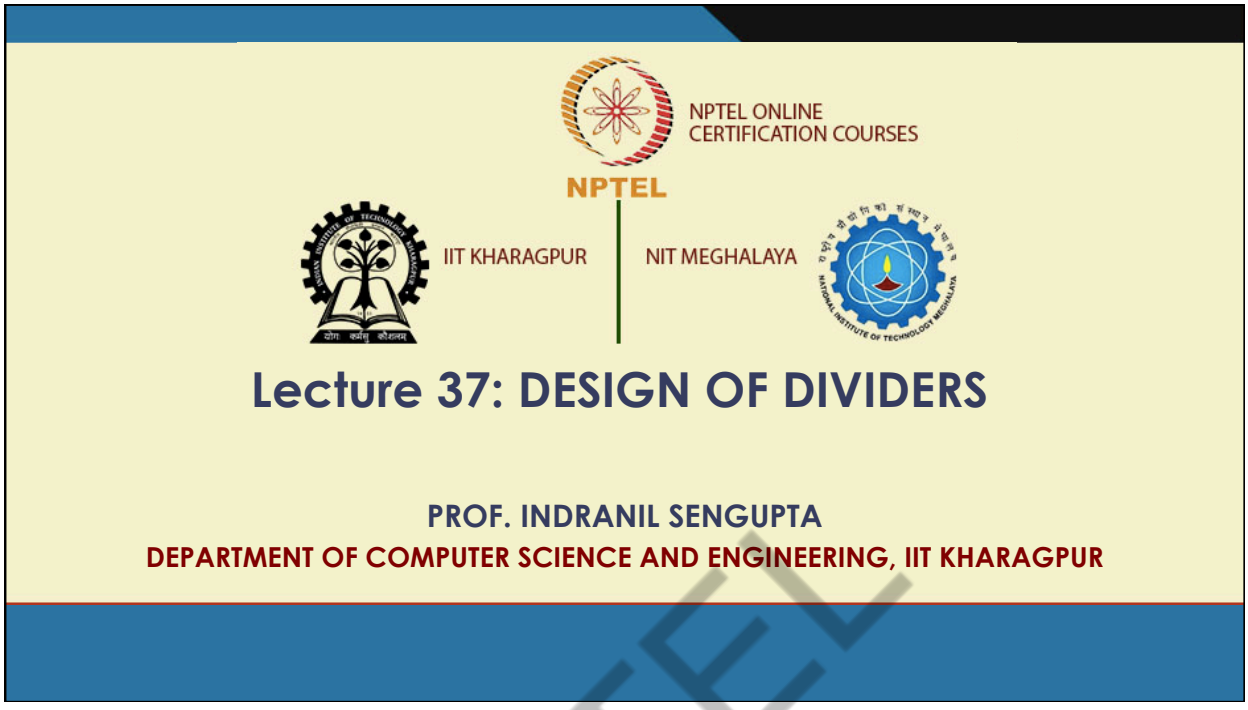
NPTEL

- The figure shows a Wallace tree that adds 8 partial products $m^{(0)}$, $m^{(1)}$, …, $m^{(7)}$.
- The partial product $m^{(i)}$ consists of (n+i) bits.
- Each line represents an entire number – the label of an edge indicates the number of bits.
- The carry-lookahead adder at the bottom adds a (2n-1)-bit number to a 2n-bit number to give the 2n-bit product.



END OF LECTURE 36

NPTEL



NPTEL

Introduction

- Division is more complex than multiplication.
- *Example*: Typical values in Pentium-3 processor.
  - Not easy to construct high-speed dividers.
- The ratios have not changed much in later processors.

Instruction	Latency	Cycles / Issue
Load / Store	3	1
Integer Multiply	4	1
Integer Divide	36	36
Floating-point Add	3	1
Floating-point Multiply	5	2
Floating-point Divide	38	38

- Latency:
  - Minimum delay after which the first result is obtained, starting from the time when the first set of inputs is applied.
- Cycles/Issue:
  - Whenever a new set of inputs is applied to a functional unit (e.g. adder), it is called an *issue*.
  - Pipelined implementation of arithmetic unit reduces the number of clock cycles between successive issues.
  - For non-pipelined arithmetic units (e.g. divider), the number of clock cycles between successive issues is much higher.
    - Next input can be applied only after the previous operation is complete.

The Process of Integer Division

- In integer division, a *divisor* M and a *dividend* D are given.
- The objective is to find a third number Q, called the *quotient*, such that

  $D = Q \times M + R$

  where R is the *remainder* such that $0 \leq R < M$.
- The relationship D = Q x M suggests that there is a close correspondence between division and multiplication.
  - Dividend, quotient and divisor correspond to product, multiplicand and multiplier, respectively.
  - Similar algorithms and circuits can be used for multiplication and division.

NPTEL

- One of the simplest division methods is the sequential digit-by-digit algorithm similar to that used in pencil-and-paper methods.

```
                    0 1 1 0          Quotient Q = Q0Q1Q2Q3
Divisor M   1 1 0 | 1 0 0 1 0 1      Dividend D = R0
                    1 1 0            Q0.M        (Does not go; Q0 = 0)
                  -------------
                    1 0 0 1 0 1      R1
                -   1 1 0            Q1.2^-1.M   (Does go; Q1 = 1)
                  -------------
                      0 1 1 0 1      R2
                -     1 1 0          Q2.2^-2.M   (Does go; Q2 = 1)
                  -------------
                        0 0 0 1      R3
                          1 1 0      Q3.2^-3.M   (Does not go; Q3 = 0)
                  -------------
                          0 0 1      R4 = Remainder R
```

$D = 37 = (1\,0\,0\,1\,0\,1)_2$
$M = 6 = (1\,1\,0)_2$
Quotient Q = 6
Remainder R = 1

NPTEL

- In the example, the quotient $Q = Q_0Q_1Q_2\ldots$ is computed one bit at a time.
  - At each step i, the divisor shifted i bits to the right (i.e. $2^{-i}.M$) is compared with the current partial remainder R_i.
  - The quotient bit Q_i is set to 0 (1) if $2^{-i}.M$ is greater than (less than) R_i,
  - The new partial remainder R_{i+1} is computed as:

$$R_{i+1} = R_i - Q_i.2^{-i}.M$$

- Machine implementation:
  - For hardware implementation, it is more convenient to shift the partial remainder to the left relative to a fixed divisor; thus

    $R_{i+1} = 2R_i - Q_i.M$ *(instead of* $R_{i+1} = R_i - Q_i.2^{-i}.M$*)*
  - The final partial remainder is the required remainder shifted to the left, so that $R = 2^{-3}.R_4$ (see next slide).

Divisor M			Quotient Q
1 1 0	1 0 0 1 0 1	Dividend = $2R_0$	
Do not subtract	1 1 0	$Q_0.M$	0
	1 0 0 1 0 1	R_1	
	1 0 0 1 0 1 0	$2R_1$	
	1 1 0	$Q_1.M$	0 1
	0 1 1 0 1 0	R_2	
	0 1 1 0 1 0 0	$2R_2$	
	1 1 0	$Q_2.M$	0 1 1
	0 0 0 1 0 0	R_3	
	0 0 0 1 0 0 0	$2R_3$	
	1 1 0	$Q_3.M$	0 1 1 0
	0 0 1 0 0 0	$R_4 = 2^3.R$	

$D = 37 = (1\ 0\ 0\ 1\ 0\ 1)_2$

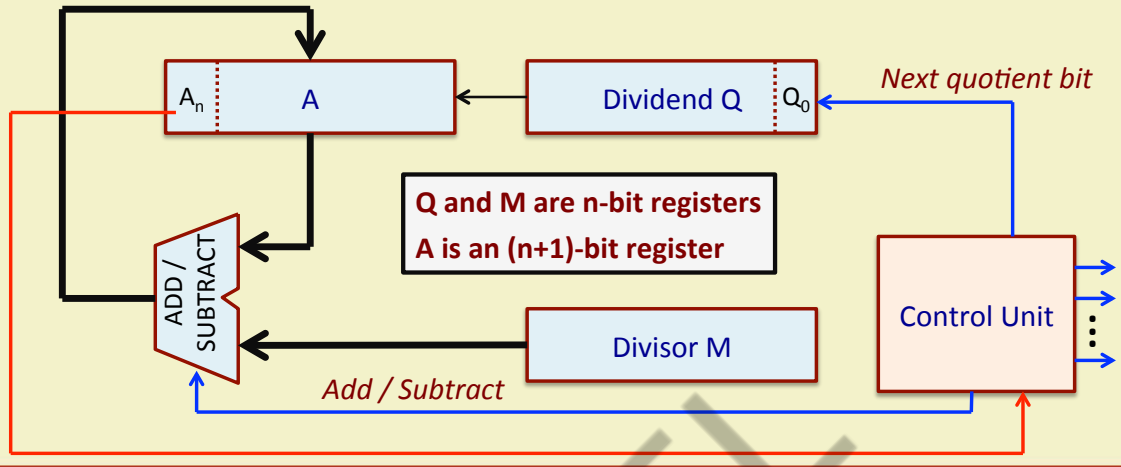
$M = 6 = (1\ 1\ 0)_2$

Quotient $Q = 6$

Remainder $R = 1$

NPTEL

Restoring Division: The Data Path



Basic Steps

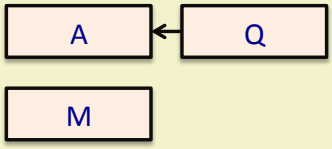
Repeat the following steps n times:

a) Shift the dividend one bit at a time starting into register A.

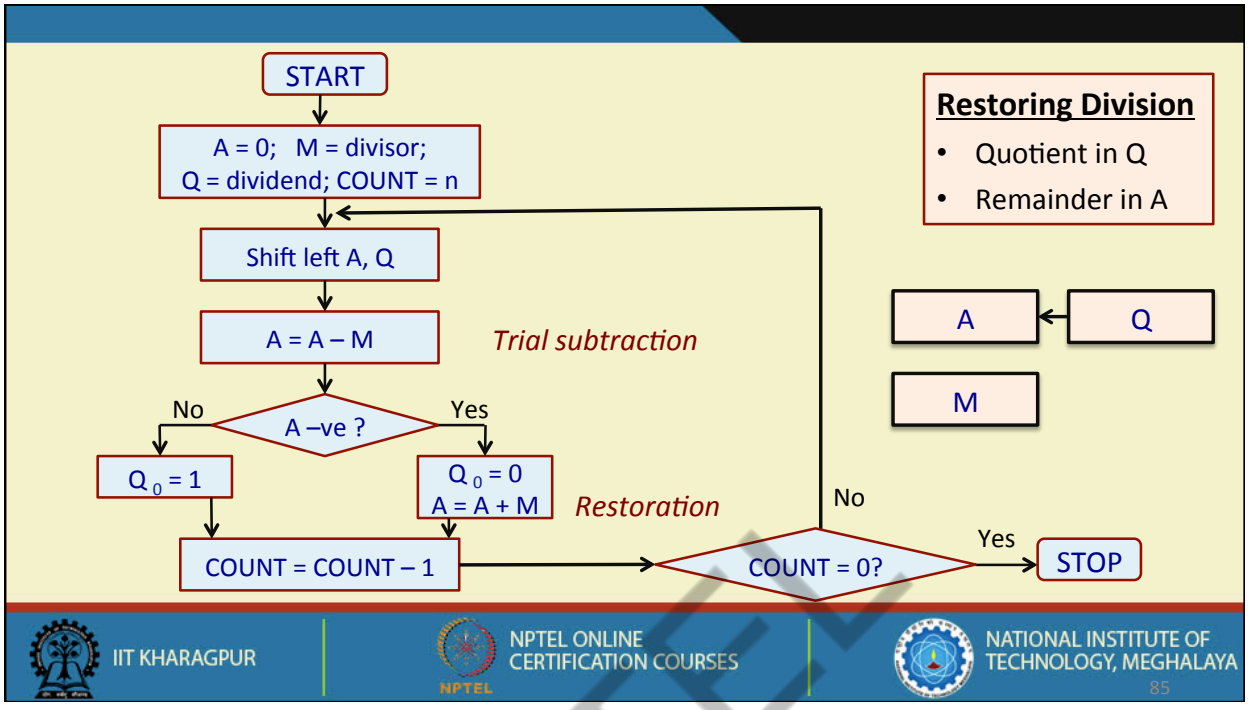
b) Subtract the divisor M from this register A (*trial subtraction*).

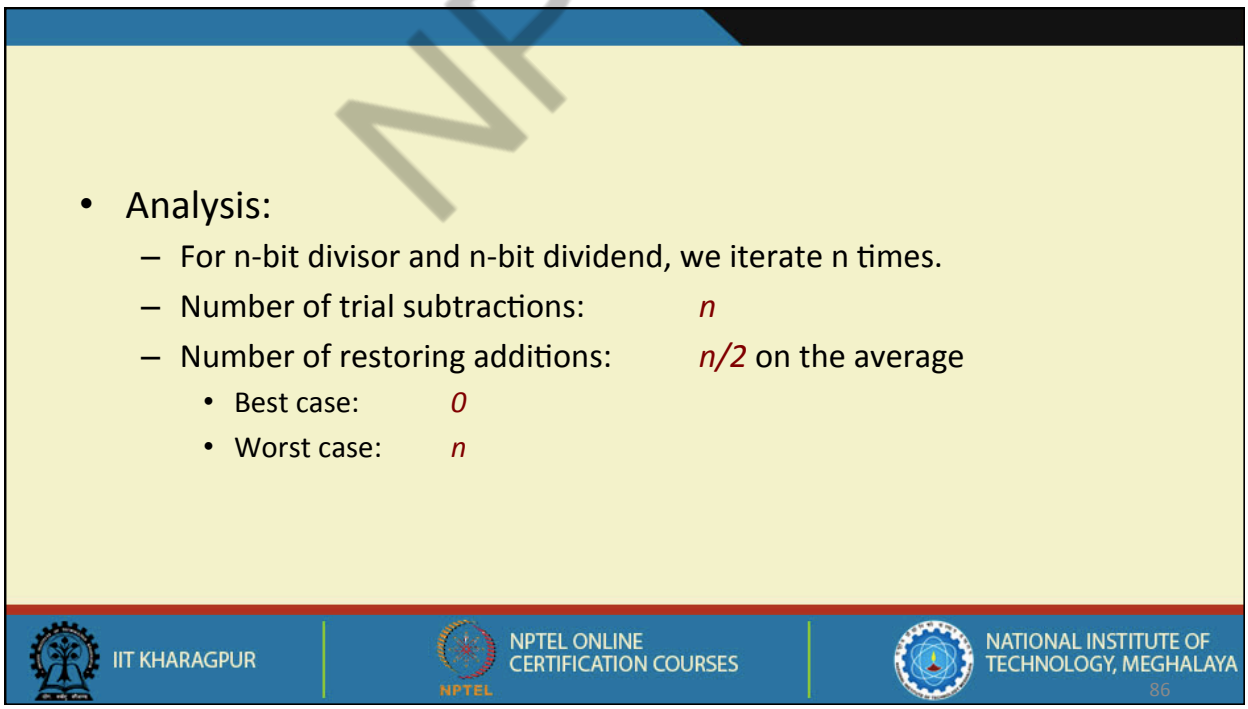
c) If the result is negative (*i.e. not going*):
- Add the divisor M back into the register A (*i.e. restoring back*).
- Record 0 as the next quotient bit.

d) If the result is positive:
- Do not restore the intermediate result.
- Record 1 as the next quotient bit.



NPTEL


START
A = 0; M = divisor;
Q = dividend; COUNT = n
Shift left A, Q
A = A – M
Trial subtraction
No
A –ve ?
Yes
Q 0 = 1
Q 0 = 0
A = A + M
Restoration
COUNT = COUNT – 1
COUNT = 0?
No
Yes
STOP
Restoring Division
• Quotient in Q
• Remainder in A
A
Q
M
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
85


• Analysis:
– For n-bit divisor and n-bit dividend, we iterate n times.
– Number of trial subtractions: n
– Number of restoring additions: n/2 on the average
• Best case: 0
• Worst case: n
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA
86

NPTEL

A Simple Example: 8/3 for 4-bit representation (n=4)

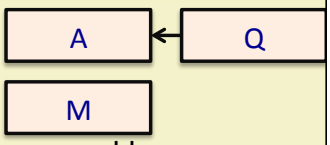
Initially:	0 0 0 0 0	1 0 0 0
	0 0 0 1 1	
Shift:	0 0 0 0 1	0 0 0 –
Subtract:		
Set Q_0:	(1) 1 1 1 0	
Restore:	0 0 0 1 1	
	0 0 0 0 1	0 0 0 (0)
Shift:	0 0 0 1 0	0 0 0 –
Subtract:		
Set Q_0:	(1) 1 1 1 1	
Restore:	0 0 0 1 1	
	0 0 0 1 0	0 0 0 (0)

Shift:	0 0 1 0 0	0 0 0 –
Subtract:		
Set Q_0:	(0) 0 0 0 1	
	0 0 0 0 0	0 0 0 (1)
Shift:	0 0 0 1 0	0 0 1 –
Subtract:		
Set Q_0:	(1) 1 1 1 1	
Restore:	0 0 0 1 1	
	0 0 0 1 0	0 0 1 (0)

Remainder 00010 = 2

Quotient 0010 = 2

Non-Restoring Division

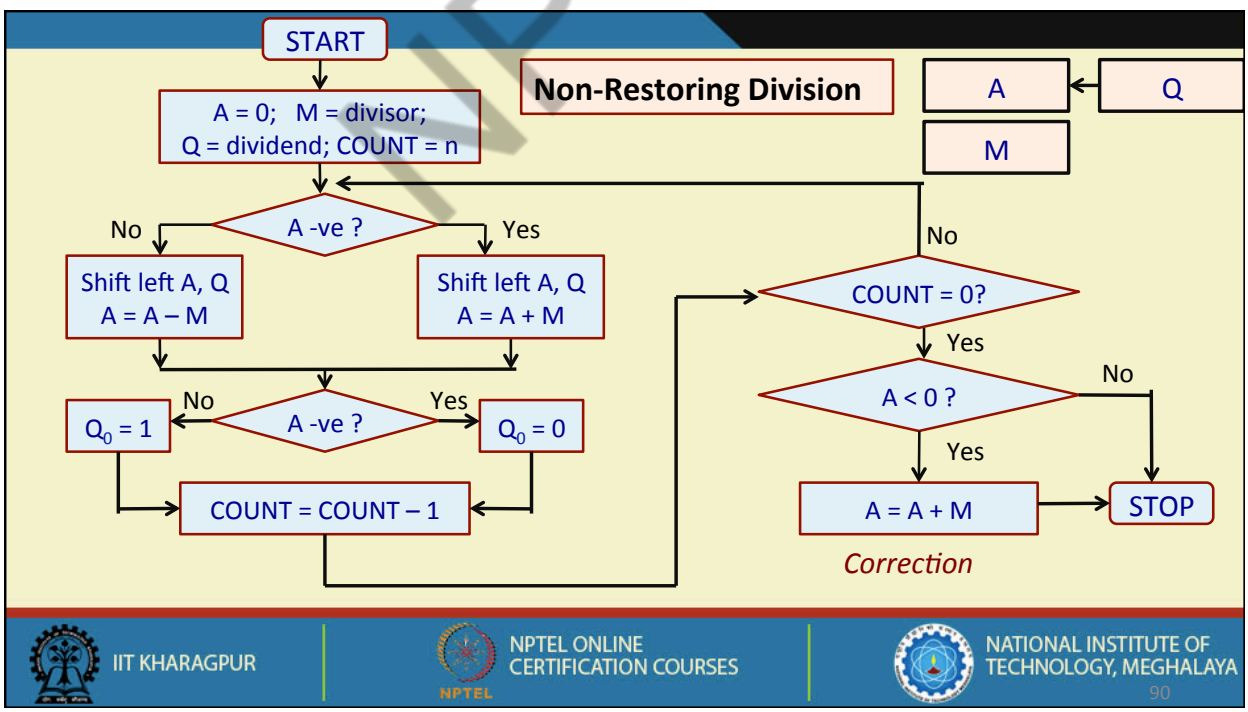


- The performance of restoring division algorithm can be improved by exploiting the following observation.
- In restoring division, what we do actually is:
  - If A is positive, we shift it left and subtract M.
    - That is, we compute 2A – M.
  - If A is negative, we restore is by doing A+M, shift it left, and then subtract M.
    - That is, we compute 2(A + M) – M = 2A + M.
- We can accordingly modify the basic division algorithm by eliminating the restoring step → *NON-RESTORING DIVISION*.

Shift left means multiplying by 2.

NPTEL

- Basic steps in non-restoring division:
  a) Start by initializing register A to 0, and repeat steps (b)-(d) *n* times.
  b) If the value in register A is positive,
     - Shift A and Q left by one bit position.
     - Subtract M from A.
  c) If the value in register A is negative,
     - Shift A and Q left by one bit position.
     - Add M to A.
  d) If A is positive, set $Q_0 = 1$; else, set $Q_0 = 0$.
  e) If A is negative, add M to A as a final corrective step.



A Simple Example: 8/3 for n=4

Initially:	0	0	0	0	0		1	0	0	0
Shift:	0	0	0	0	1		0	0	0	–
Subtract: –			0	0	1	1				
Set Q_0:	(1)	1	1	1	0		0	0	0	(0)
Shift:	1	1	1	0	0		0	0	0	–
Add:			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	–
Add:			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	–
Subtract: –			0	0	1	1				
Set Q_0:	(1)	1	1	1	1		0	0	1	(0)

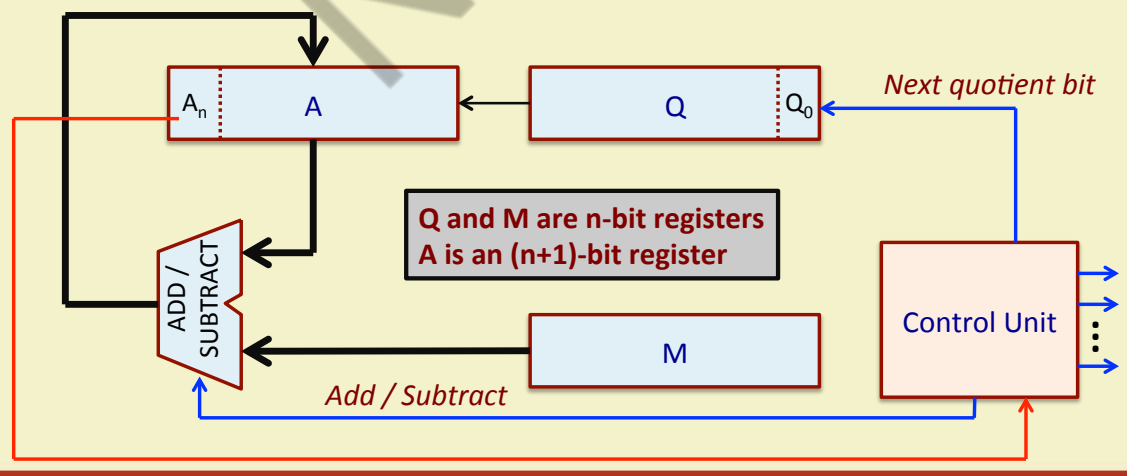
Correction Add:

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

Quotient 0010 = 2

Remainder 00010 = 2

Data Path for Non-Restoring Division



NPTEL

High Speed Dividers

- Some of the methods used to increase the speed of multiplication can also be modified to speed up division.
  - High-speed addition and subtraction.
  - High-speed shifting.
  - Combinational array divider (implementing restoring division).
- The main difficulty is that it is very difficult to implement division in a pipeline to improve the performance.
  - Unlike multiplication, where carry-save Wallace tree multipliers can be used for pipeline implementation.

NPTEL

END OF LECTURE 37