

# **Computer Organization and Architecture**

## **Module 6** **Computer Arithmetic**

**Prof. Indranil Sengupta**

**Dr. Sarani Bhattacharya**

**Department of Computer Science and Engineering**

**IIT Kharagpur**

1

## **Computer Arithmetic**

2

## 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 carry out various arithmetic operations in binary?
  - How to implement them efficiently in hardware?

3

3

## Addition / Subtraction

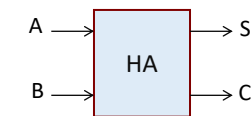
4

## 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

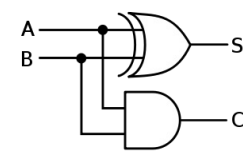
0 + 0 = 00  
 0 + 1 = 01  
 1 + 0 = 01  
 1 + 1 = 10



**HALF ADDER**

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

$$C = A.B$$



5

5

## 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

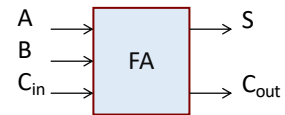
**WE NEED A FULL ADDER**

6

6

## Full Adder

Inputs			Outputs	
A	B	C <sub>in</sub>	S	C <sub>out</sub>
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



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

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

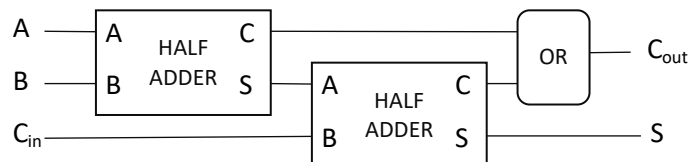
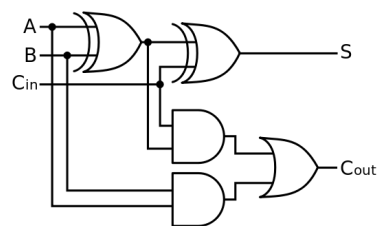
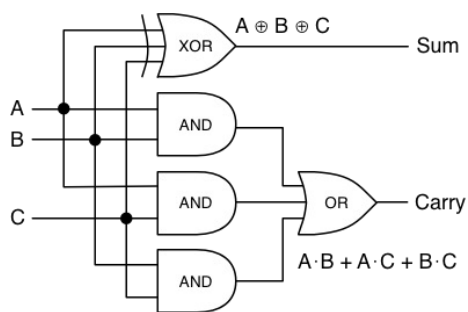
$$C_{out} = B.C_{in} + A.C_{in} + A.B + A.B.C_{in}$$

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

7

7

## Various Implementations of Full Adder

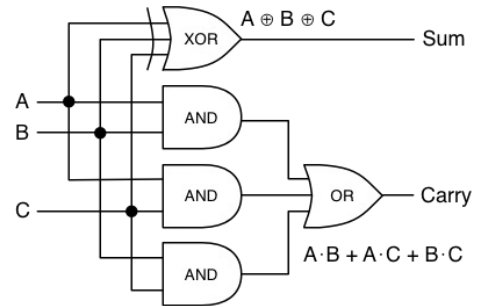


8

8

### • Delay of a full adder:

- Assume that the delay of all basic gates (AND, OR, NAND, NOR, NOT) is  $\delta$
- Delay for Carry =  $2\delta$
- Delay for Sum =  $3\delta$   
(AND-OR delay plus one inverter delay)



9

9

## 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

10

10

## Ripple Carry Adder

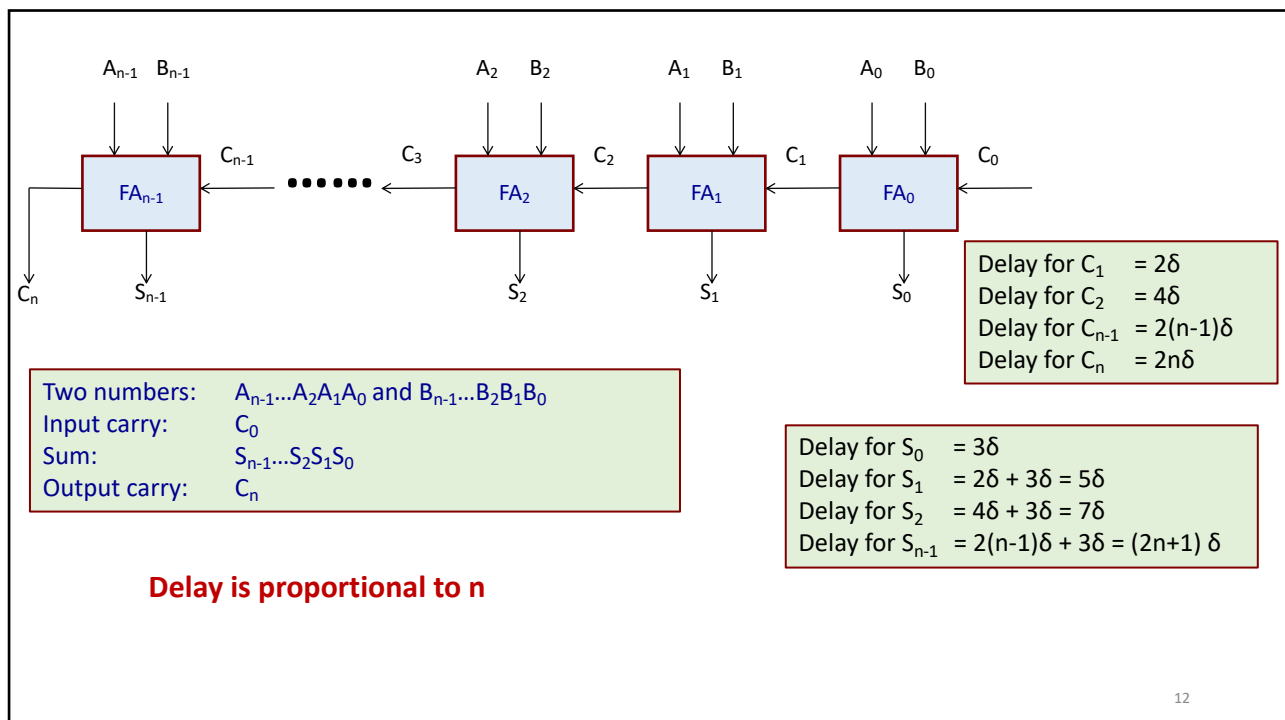
- Cascade  $n$  full adders to create a  $n$ -bit parallel adder.
- Carry output from stage- $i$  propagates as the carry input to stage- $(i+1)$ .
- In the worst-case, carry ripples through all the stages.

$$\begin{array}{r}
 1\ 1\ 1\ 1\ 1\ 1\ 0 \\
 0\ 1\ 1\ 1\ 1\ 1\ 1 \\
 +\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 0\ 0\ 0
 \end{array}$$

Carry  
 Number A  
 Number B  
 Sum S

11

11



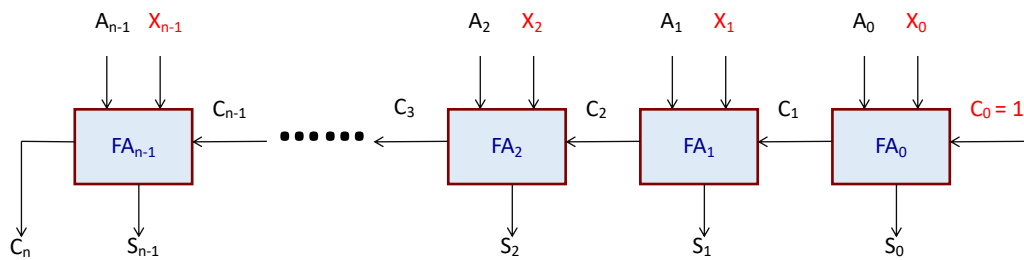
12

12

## How to Design a Parallel Subtractor?

- **Observation:**

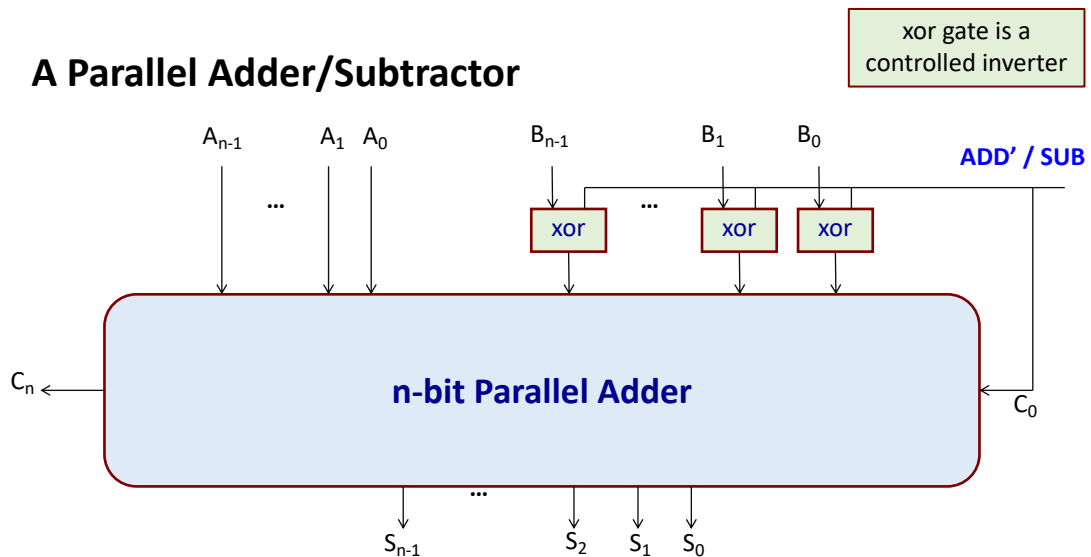
- Computing  $A - B$  is the same as adding the 2's complement of  $B$  to  $A$ .
- 2's complement is equal to 1's complement plus 1.
- Let  $X_i = B_i'$ .



13

13

## A Parallel Adder/Subtractor



14

14

## 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  $O(1)$ .
  - However, hardware complexity increases rapidly with  $n$ .

15

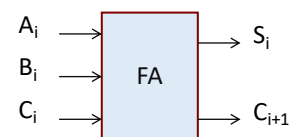
15

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

$$G_i = A_i \cdot B_i$$

$$P_i = A_i \oplus B_i$$

- $G_i = 1$  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  $C_{i+1}$ .



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

16

16



## Unrolling the Recurrence

$$\begin{aligned}
 C_{i+1} &= G_i + P_i C_i = G_i + P_i (G_{i-1} + P_{i-1} C_{i-1}) = G_i + P_i G_{i-1} + P_i P_{i-1} C_{i-1} \\
 &= G_i + P_i G_{i-1} + P_i P_{i-1} (G_{i-2} + P_{i-2} C_{i-2}) \\
 &= G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + P_i P_{i-1} P_{i-2} C_{i-2} = \dots
 \end{aligned}$$

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

17

17

## Design of 4-bit CLA Adder

$$\begin{aligned}
 C_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3 \\
 C_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2 \\
 C_2 &= G_1 + G_0 P_1 + C_0 P_0 P_1 \\
 C_1 &= G_0 + C_0 P_0
 \end{aligned}$$

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

$$\begin{aligned}
 S_0 &= A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0 \\
 S_1 &= P_1 \oplus C_1 \\
 S_2 &= P_2 \oplus C_2 \\
 S_3 &= P_3 \oplus C_3
 \end{aligned}$$

4 XOR2 gates

18

18

## 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

19

19

## Design of 4-bit CLA Adder

$$C_4 = G_3 + C_3P_3$$

$$C_3 = G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$$

$$C_2 = G_1 + G_0P_1 + C_0P_0P_1$$

$$C_1 = G_0 + C_0P_0$$

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

$$S_1 = P_1 \oplus C_1$$

$$S_2 = P_2 \oplus C_2$$

$$S_3 = P_3 \oplus C_3$$

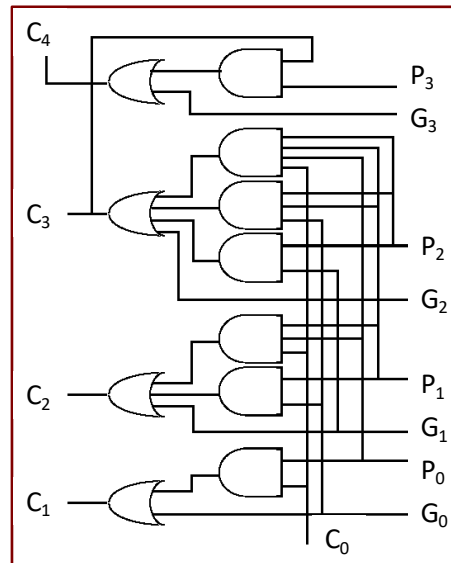
4 AND2 gates  
2 AND3 gates  
1 AND4 gates  
~~1 AND5 gate~~  
1 OR2, 1 OR3, 1 OR4 and  
1 **OR2** gate

4 XOR2 gates

20

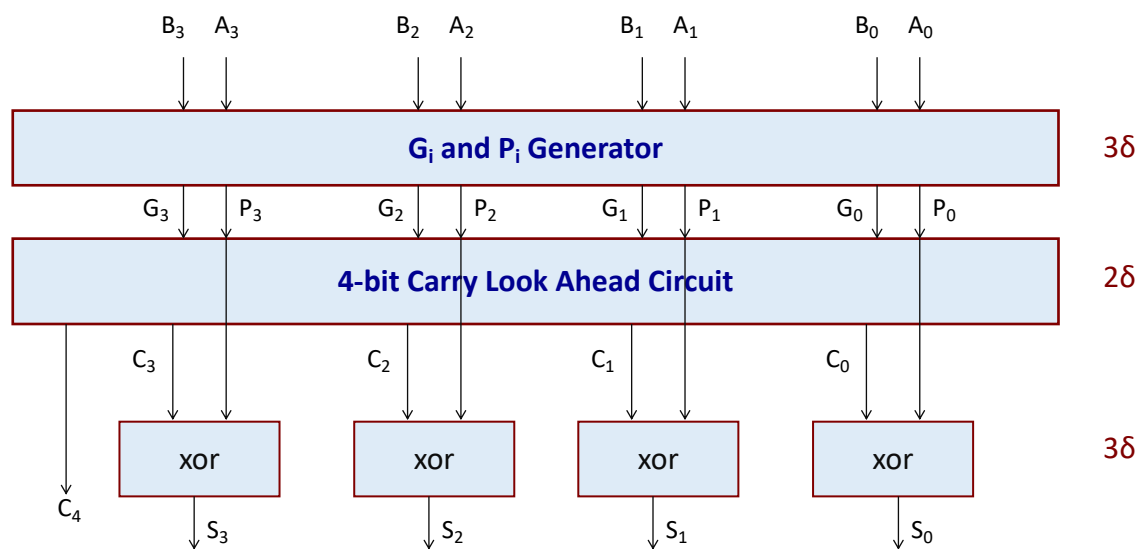
20

## The 4-bit CLA Circuit



21

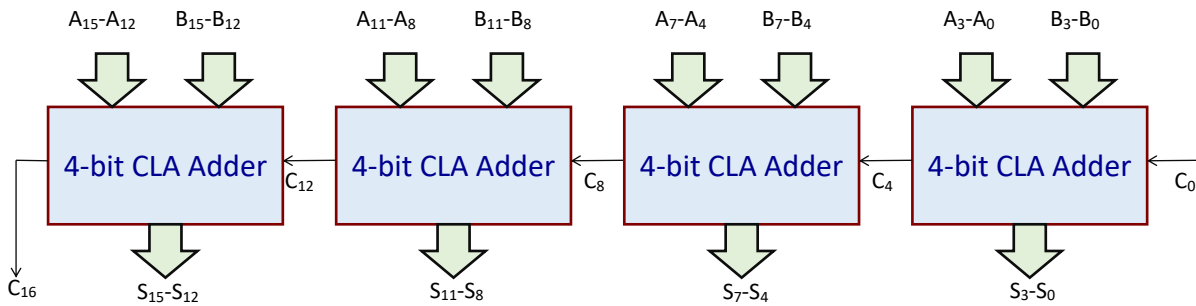
21



22

22

## 16-bit Adder Using 4-bit CLA Modules



*Problem: Carry propagation between modules still slows down the adder*

23

23

### • 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 C<sub>4</sub>, C<sub>8</sub>, C<sub>12</sub> and C<sub>16</sub> in parallel with two gate delays (2 $\delta$ ).
  - For larger values of  $n$ , more CLA levels can be added.
- Delay calculation of a 16-bit adder:
- a) For original single-level CLA: 14 $\delta$
  - b) For modified two-level CLA: 10 $\delta$

24

24

## Delay of a k-bit Adder

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

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

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

25

25

## Carry Select Adder

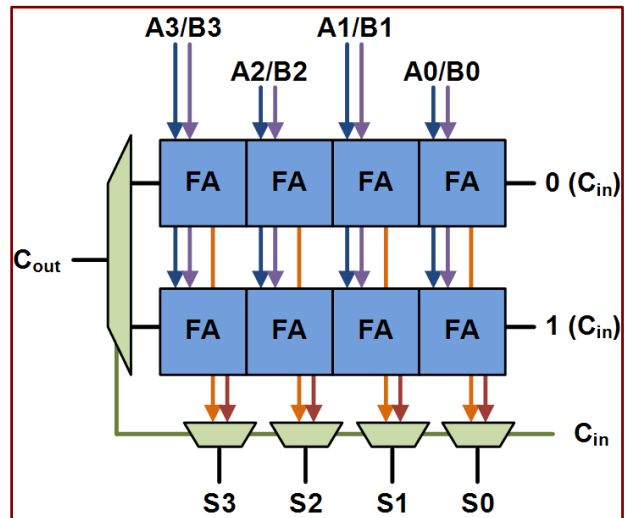
- Basically consists of two parallel adders (say, ripple-carry adder) and a multiplexer.
- For two given numbers  $A$  and  $B$ , we carry out addition twice:
  - With carry-in as 0
  - With carry-in as 1
- Once the correct carry-in is known, the correct sum is selected by a multiplexer.

26

26

### 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.

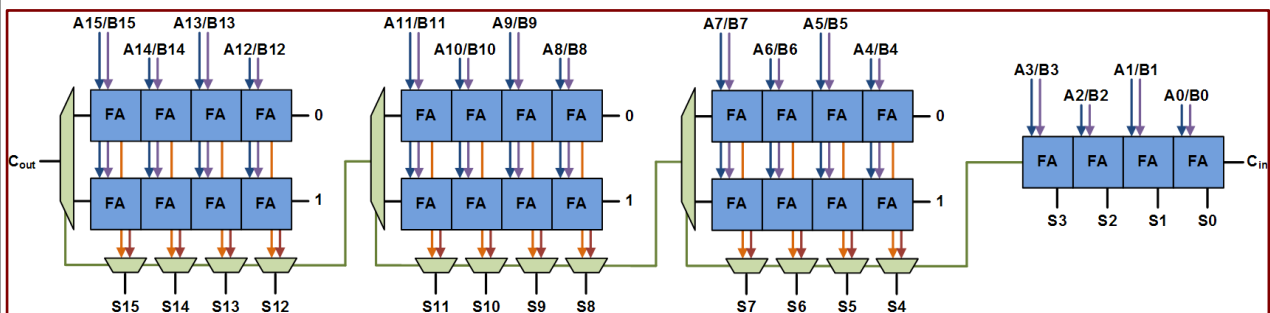


27

27

### 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.

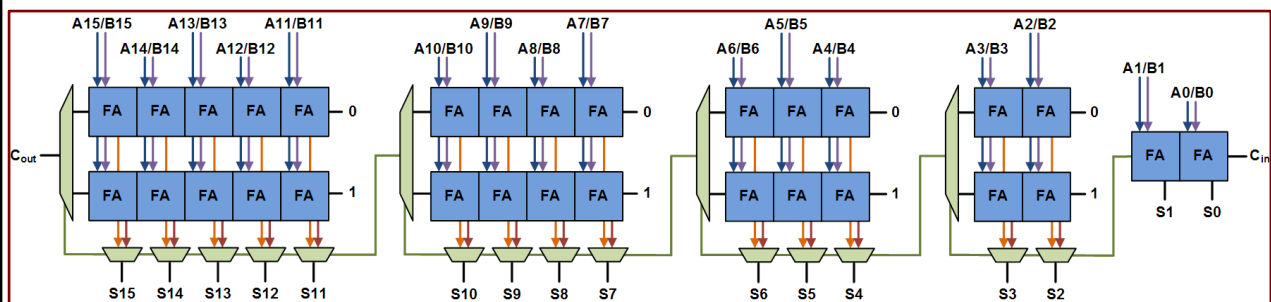


28

28

### 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.



29

29

### 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.

30

30

• An illustrative example:

$$\begin{array}{r} X: \quad 1\ 0\ 0\ 1\ 1 \\ Y: + 1\ 1\ 0\ 0\ 1 \\ Z: + 0\ 1\ 0\ 1\ 1 \\ \hline C: \quad 1\ 1\ 0\ 1\ 1 \end{array}$$

$$\begin{array}{r} X: \quad 1\ 0\ 0\ 1\ 1 \\ Y: + 1\ 1\ 0\ 0\ 1 \\ Z: + 0\ 1\ 0\ 1\ 1 \\ \hline S: \quad 0\ 0\ 0\ 0\ 1 \end{array}$$

$$\begin{array}{r} X: \quad 1\ 0\ 0\ 1\ 1 \\ Y: + 1\ 1\ 0\ 0\ 1 \\ Z: + 0\ 1\ 0\ 1\ 1 \\ \hline S: \quad 0\ 0\ 0\ 0\ 1 \\ C: \quad 1\ 1\ 0\ 1\ 1 \\ \hline \text{Sum: } 1\ 1\ 0\ 1\ 1\ 1 \end{array}$$

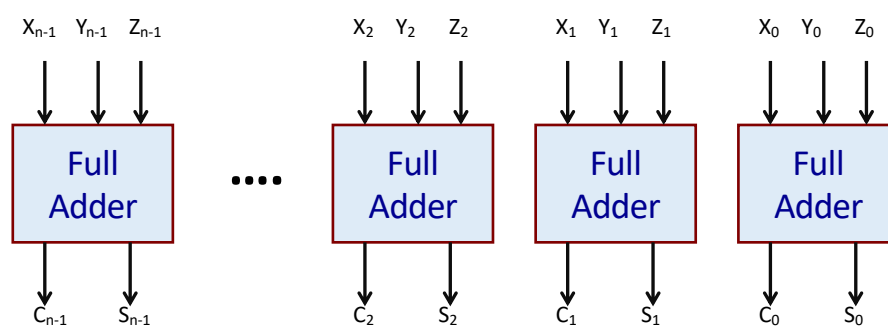
A set of full adders generate carry and sum bits in parallel

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

31

31

### An n-bit Carry Save Adder



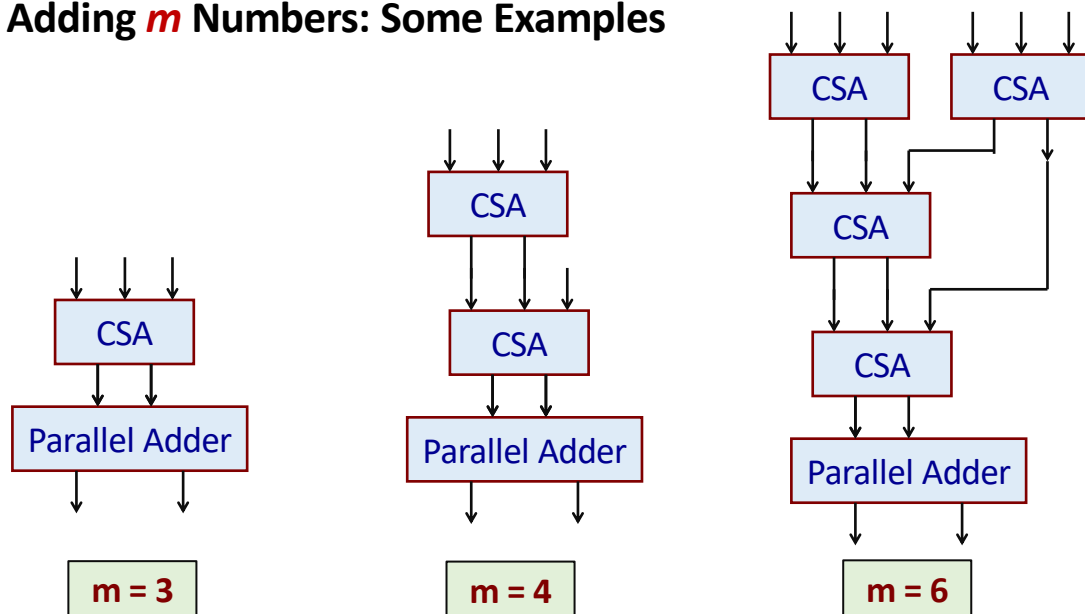
*The carry input of the full adder is used as the third input*

32

32



## Adding $m$ Numbers: Some Examples



33

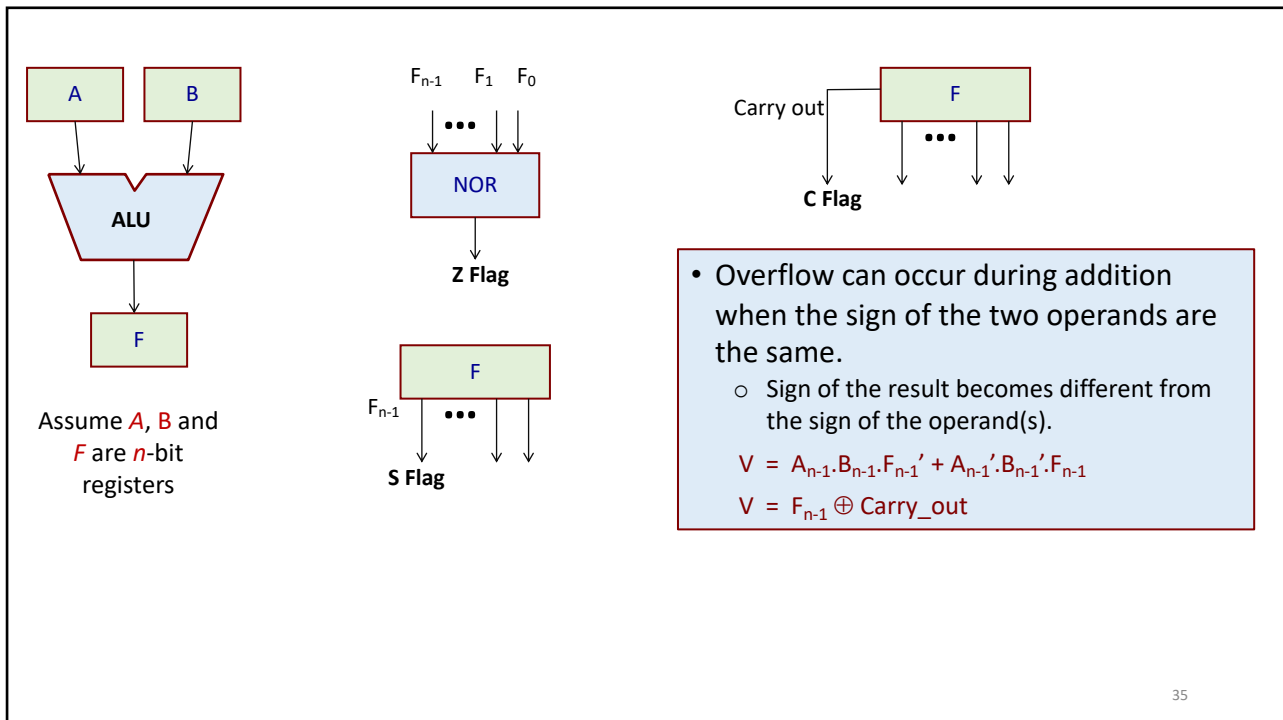
33

## 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).

34

34



35

- The MIPS32 processor 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
```

36

36