COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface
5th Edition

# Chapter 3 & Appendix B (continued)

## Arithmetic for Computers

COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface
5th Edition

# Improving Our ALU

## Speed is important

- Using a *ripple carry adder*, the time required to perform addition is too long
  - each 1-bit ALU has two *levels* of gates
  - The input to the $i^{th}$ ALU includes an output from the $(i-1)^{th}$ ALU
  - For a 32-bit ALU, we face 64 gate delays before the addition is complete

 adding....



## Strategies for speeding things up

- We could derive the truth table for each of the 32 result bits as a function of the 64 inputs

- We could build SOP expressions for each bit and implement our ALU using two levels of gates…
  - …but that requires too much hardware

## A more efficient approach

- The problem is the *ripple*
  - The last (MSB) carry-in takes a rather long time to compute
- We can try to compute the carry-in bits faster by using a technique called *carry lookahead* to create a *carry-lookahead adder*
  - It turns out we can easily compute the carry-in bits much faster
    - (but still not in constant time…)



## Carry In Analysis

- $\text{CarryIn}_i$ is input to the $i^{th}$ 1-bit adder
- $\text{CarryOut}_{i-1}$ is connected to $\text{CarryIn}_i$ for $i>1$
- We know how to compute the CarryOuts from the truth table

| A | B | Carry In | Carry Out | Sum |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

## Computing the Carry Bits

- $CarryIn_0$ is an input to the adder
  - we don't compute this --- it's an input
- $CarryIn_1$ depends on $A_0$, $B_0$, and $CarryIn_0$:

$CarryIn_1$ =

$$(B_0 \cdot CarryIn_0) + (A_0 \cdot CarryIn_0) + (A_0 \cdot B_0)$$

SOP: Requires 2 levels of gates

Arithmetic for Computers — 7

## $CarryIn_2$

$CarryIn_2$ =

$$(B_1 \cdot CarryIn_1) + (A_1 \cdot CarryIn_1) + (A_1 \cdot B_1)$$

We can then substitute for $CarryIn_1$ and obtain:

$CarryIn_2$ =

$$(B_1 \cdot B_0 \cdot CarryIn_0) + (B_1 \cdot A_0 \cdot CarryIn_0) +$$
$$(B_1 \cdot A_0 \cdot B_0) + (A_1 \cdot B_0 \cdot CarryIn_0) +$$
$$(A_1 \cdot A_0 \cdot CarryIn_0) + (A_1 \cdot A_0 \cdot B_0) + (A_1 \cdot B_1)$$

The length of these expressions gets way too big!

Arithmetic for Computers — 8

## Another way to describe CarryIn?

$$C_{i+1} = (B_i \cdot C_i) + (A_i \cdot C_i) + (A_i \cdot B_i)$$
$$= ???$$



## Pop Quiz

How can we further rearrange expression $(B_i \cdot C_i) + (A_i \cdot C_i) + (A_i \cdot B_i)$ to isolate $C_i$?

## Another way to describe CarryIn

$$C_{i+1} = (B_i \cdot C_i) + (A_i \cdot C_i) + (A_i \cdot B_i)$$

Commutative (twice)

$$= (A_i \cdot B_i) + (B_i \cdot C_i) + (A_i \cdot C_i)$$
$$= (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$

Distributive (factoring out)

$$= (A_i \cdot B_i) + [\ (A_i + B_i) \cdot C_i\ ]$$

Arithmetic for Computers — 11

## Another way to describe CarryIn

$$C_{i+1} = (B_i \cdot C_i) + (A_i \cdot C_i) + (A_i \cdot B_i)$$
$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$

$A_i \cdot B_i$ : Call this *Generate* ($G_i$)

$A_i + B_i$ : Call this *Propagate* ($P_i$)

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

Arithmetic for Computers — 12

## Generate and Propagate

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

- Both $A_i$ and $B_i$ must be 1 for $G_i$ to become 1
  - i.e., to *generate* a CarryOut
- If $P_i$ is 1, then any CarryIn ($C_i$) is essentially *propagated* to CarryOut ($C_{i+1}$)

Arithmetic for Computers — 13

## Using $G_i$ and $P_i$

$C_1 = G_0+P_0 \cdot C_0$

$C_2 = G_1+P_1 \cdot C_1$
$= G_1+ P_1 \cdot (G_0+P_0 \cdot C_0)$
$= G_1+ P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$

$C_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$

$C_4$ = etc. *(try to write this out…)*

Arithmetic for Computers — 14

# Implementation

- Okay, so these expressions still get too big to handle (e.g., for 32 bits!)
- But we *can* minimize the time needed to compute all the CarryIn bits for say a 4-bit adder
- Then we can connect a bunch of 4-bit adders together and treat CarryIns to these adders in the same manner
  - i.e., use this 4-bit carry-lookahead adder as a single component to implement larger-width adders

Arithmetic for Computers — 15

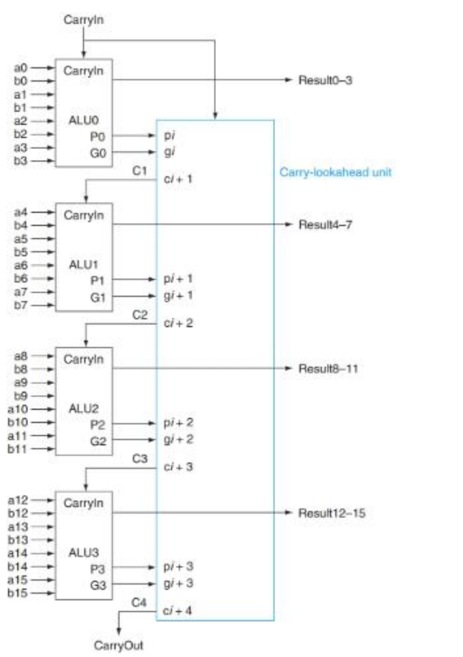


**FIGURE B.6.3 Four 4-bit ALUs using carry lookahead to form a 16-bit adder.** Note that the carries come from the carry-lookahead unit, not from the 4-bit ALUs.

Arithmetic for Computers — 16

## Concluding Remarks

§3.9 Concluding Remarks

- Bits have no inherent meaning
  - Interpretation depends on the instructions applied
- Computer representations of numbers
  - Finite range and precision
  - Need to account for this in programs

Chapter 3 — Arithmetic for Computers — 17

## Concluding Remarks

- ISAs support arithmetic
  - Signed and unsigned integers
  - Floating-point approximation to reals
- Bounded range and precision
  - Operations can overflow and underflow

Chapter 3 — Arithmetic for Computers — 18