Department of Electrical Engineering, Feng-Chia University

Chapter 3
Arithmetic for Computers
(Part 1)

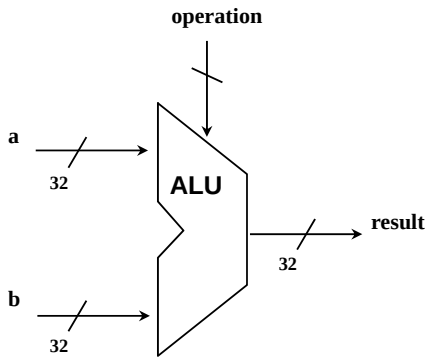
王振傑 (Chen-Chieh Wang)
ccwang@mail.ee.ncku.edu.tw

Department of Electrical Engineering, Feng-Chia University

Outline

Arithmetic

- Where we've been:
  - System Software
  - Instruction Set Architecture (ISA)
  - Instructions (arithmetic, logic, load/store, branch)
    - Assembly Language
    - Machine Language
- What's up ahead:
  - Implementing the Architecture



Department of Electrical Engineering, Feng-Chia University

Arithmetic for Computers

- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

Outline

Addition & Subtraction

- Just like in grade school (carry/borrow 1s)

```
  0111        0111          0110
+ 0110      - 0110        - 0101
```

- Two's complement operations easy
  - subtraction using addition of negative numbers

```
  0111
+ 1010
```

- Overflow (result too large for finite computer word):
  - e.g., adding two n-bit numbers does not yield an n-bit number

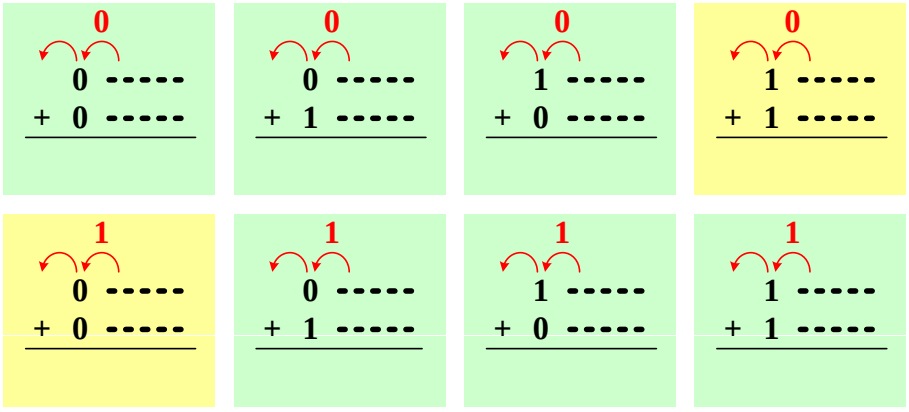
```
  0111
+ 0001
  1000
```

note that overflow term is somewhat misleading, it does not mean a carry "overflowed"

The maximum value in a 4-bit signed number is 7.

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive



Department of Electrical Engineering, Feng-Chia University

Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Details based on software system / language
  - example: flight control vs. homework assignment
- Don't always want to detect overflow
  - new MIPS instructions: `addu, addiu, subu`

note: `addiu` *still sign-extends!*
note: `sltu, sltiu` *for unsigned comparisons*

Outline

Text Book : C-4 ~ C-20

Logic Gate and MUX

- Functionally complete : AND, OR, NOT.

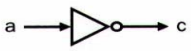
1. AND gate (c = a · b)

a	b	c = a · b
0	0	0
0	1	0
1	0	0
1	1	1

2. OR gate (c = a + b)

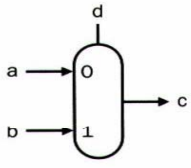
a	b	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter (c = ā)



a	c = ā
0	1
1	0

4. Multiplexor
(if d = = 0, c = a;
else c = b)



d	c
0	a
1	b

Text Book : C-4 ~ C-20

Review: Boolean Algebra & Gates

- Problem: Consider a logic function with three inputs: A, B, and C.

  Output D is true if at least one input is true
  Output E is true if exactly two inputs are true
  Output F is true only if all three inputs are true

- Show the truth table for these three functions.
- Show the Boolean equations for these three functions.
- Show an implementation consisting of inverters, AND, and OR gates.

Department of Electrical Engineering, Feng-Chia University

Text Book : C-26 ~ C-27

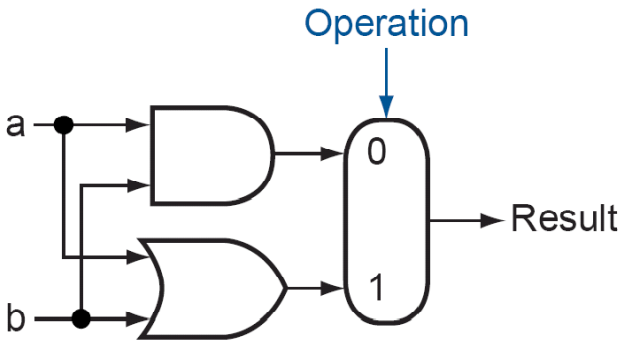
Arithmetic Logic Unit (ALU)

- Not easy to decide the “best” way to build something
  - Don’t want too many inputs to a single gate
  - Don’t want to have to go through too many gates
  - for our purposes, ease of comprehension is important
- Let's look at a 1-bit ALU :
  - How could we build a 1-bit ALU for add, and, and or ?
  - How could we build a 32-bit ALU?

Text Book : C-26 ~ C-27

1-bit logical unit for AND and OR

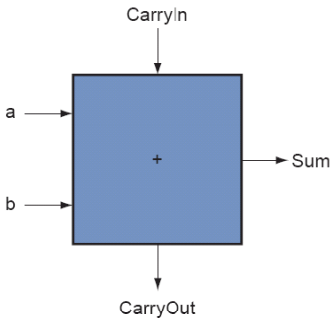
- Operation is from the control unit.



Text Book : C-27 ~ C-28

1-bit Adder

Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

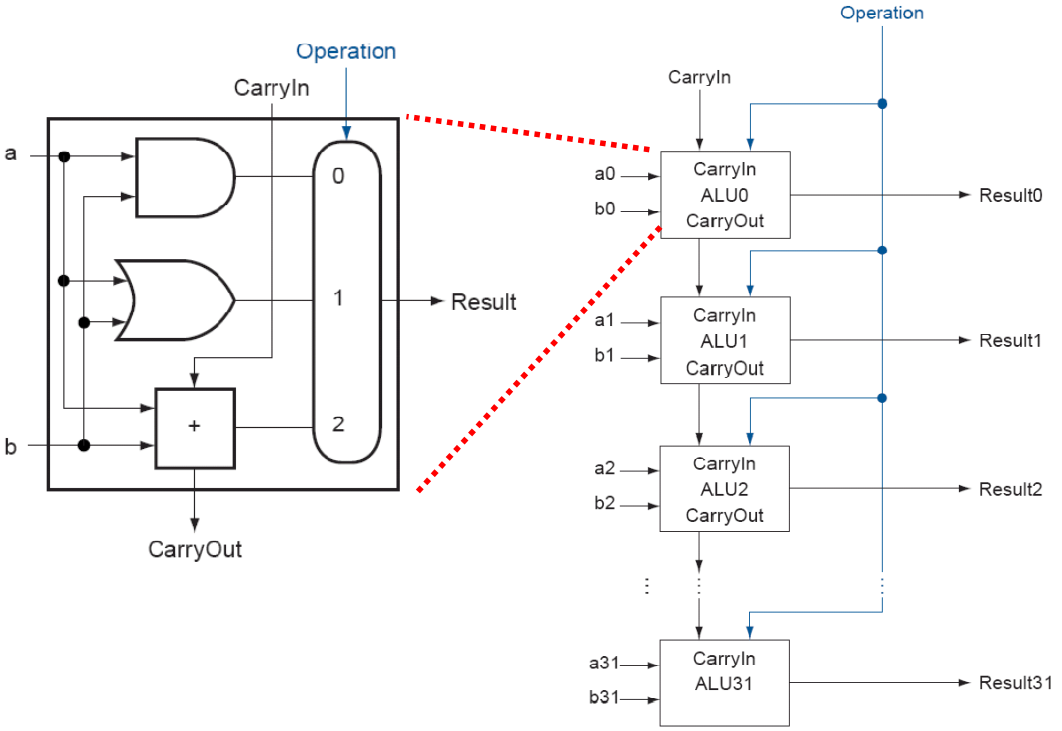


Sum = a ⊕ b ⊕ CarryIn

CarryOut = a · b + b · CarryIn + a · CarryIn

Text Book : C-29 ~ C-30

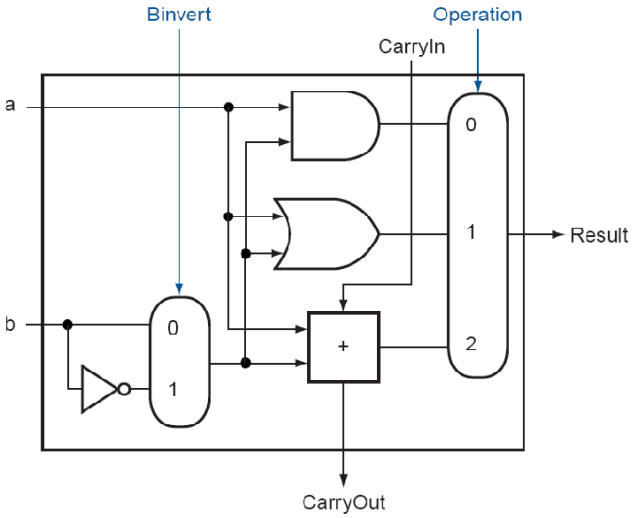
Building a 32-bit ALU



Text Book : C-30 ~ C-31

What about subtraction (a – b) ?

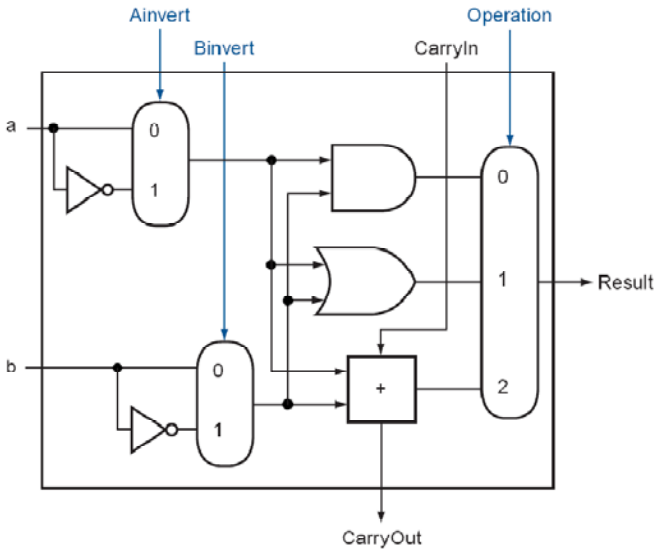
- Two's complement approach: just negate b and add.
- How do we negate?
- A very clever solution:



Text Book : C-31 ~ C-32

Adding a NOR function

- Can also choose to invert a. How do we get "a NOR b" ? (Hint: DeMorgan's theorem)



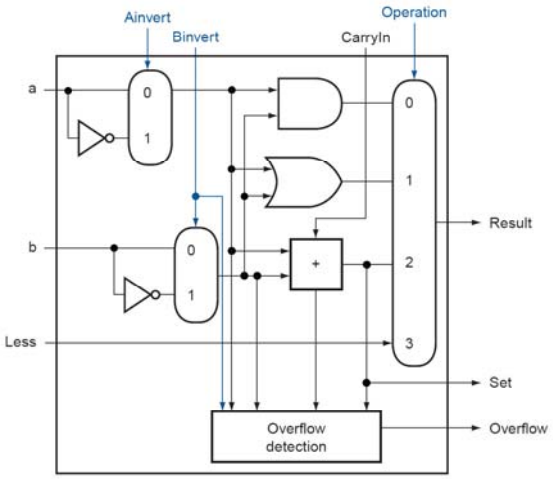
Text Book : C-32 ~ C-33

Tailoring the ALU to the MIPS

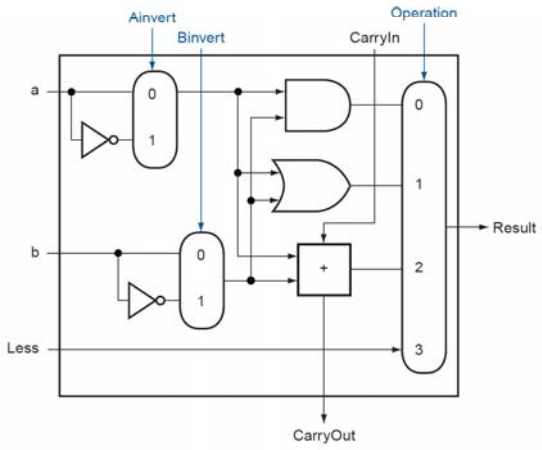
- Need to support the set-on-less-than instruction (slt)
  - remember: slt is an arithmetic instruction
  - produces a 1 if rs < rt and 0 otherwise
  - use subtraction: (a-b) < 0 implies a < b
- Need to support test for equality (beq $t5, $t6, $t7)
  - use subtraction: (a-b) = 0 implies a = b

Text Book : C-32 ~ C-33

Supporting slt



Use this ALU for most significant bit

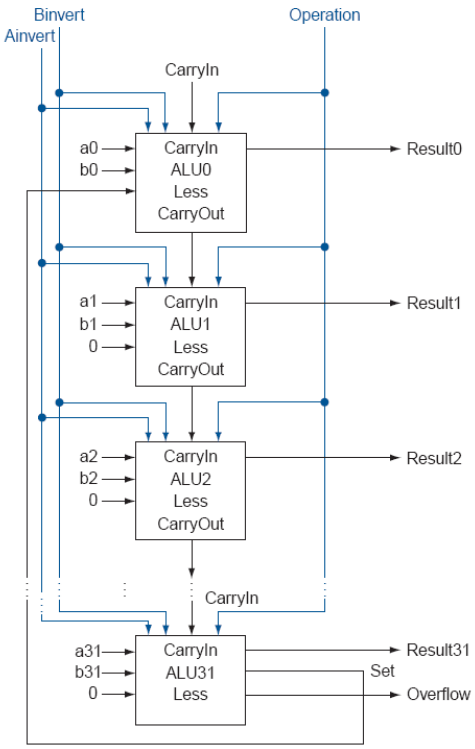


all other bits

Department of Electrical Engineering, Feng-Chia University

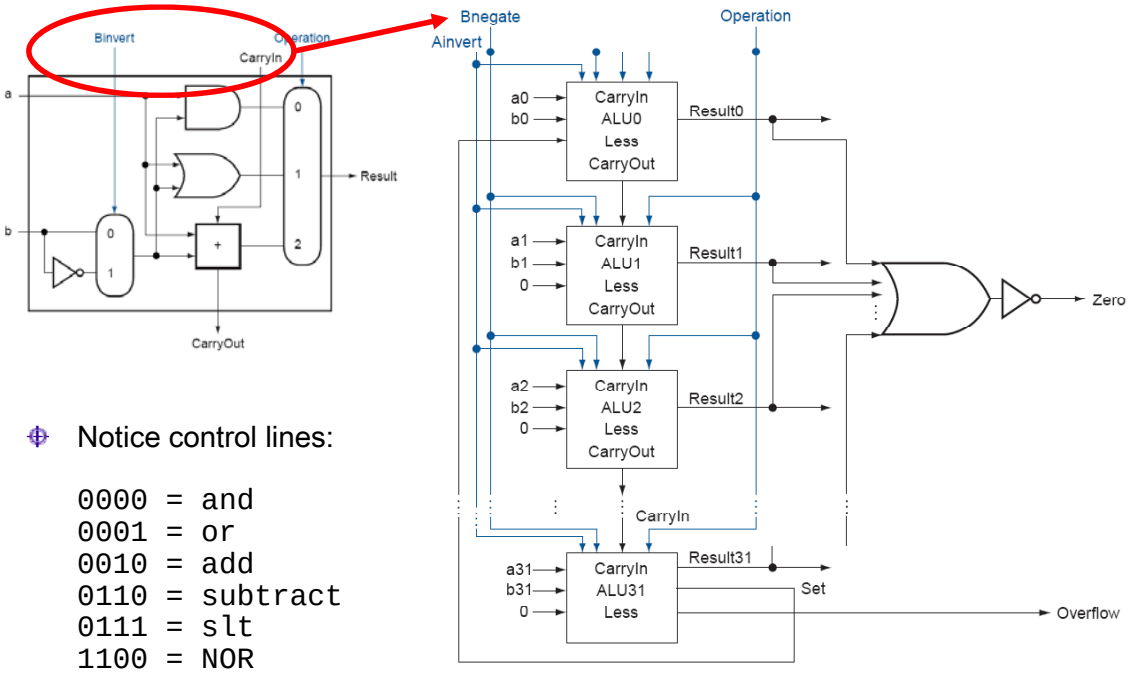
Text Book : C-34

Supporting slt



Text Book : C-35 ~ C-36

Test for equality



- Notice control lines:

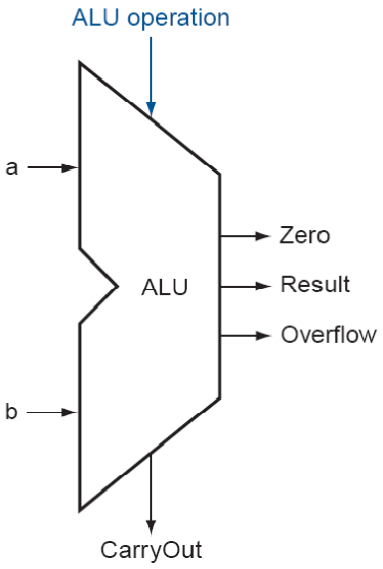
```
0000 = and
0001 = or
0010 = add
0110 = subtract
0111 = slt
1100 = NOR
```

- *Note: zero is a 1 when the result is zero!*

Text Book : C-37

ALU Symbol

- a, b, Result, are buses for data
- ALU operation are control signals
- Zero, Carryout, Overflow are status signals.



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Conclusion

- We can build an ALU to support the MIPS instruction set
  - key idea: use multiplexor to select the output we want
  - we can efficiently perform subtraction using two's complement
  - we can replicate a 1-bit ALU to produce a 32-bit ALU
- Important points about hardware
  - all of the gates are always working
  - the speed of a gate is affected by the number of inputs to the gate
  - the speed of a circuit is affected by the number of gates in series (on the "critical path" or the "deepest level of logic")
- Our primary focus: comprehension, however,
  - clever changes to organization can improve performance (similar to using better algorithms in software)
  - we'll look at two examples for addition and multiplication

Outline

Problem: ripple carry adder is slow

- Is a 32-bit ALU as fast as a 1-bit ALU?
- Is there more than one way to do addition?
  - two extremes: ripple carry and sum-of-products

Can you see the ripple? How could you get rid of it?

$$c_1 = b_0c_0 + a_0c_0 + a_0b_0$$
$$c_2 = b_1c_1 + a_1c_1 + a_1b_1 \qquad c_2 =$$
$$c_3 = b_2c_2 + a_2c_2 + a_2b_2 \qquad c_3 =$$
$$c_4 = b_3c_3 + a_3c_3 + a_3b_3 \qquad c_4 =$$

Not feasible! Why?

Department of Electrical Engineering, Feng-Chia University

Carry-Lookahead Adder (CLA)

- An approach in-between our two extremes
- Motivation:
  - If we didn't know the value of carry-in, what could we do?
  - When would we always generate a carry? $g_i = a_i b_i$
  - When would we propagate the carry? $p_i = a_i + b_i$
- Did we get rid of the ripple?

$$c_1 = g_0 + p_0c_0$$
$$c_2 = g_1 + p_1c_1 \qquad c_2 =$$
$$c_3 = g_2 + p_2c_2 \qquad c_3 =$$
$$c_4 = g_3 + p_3c_3 \qquad c_4 =$$

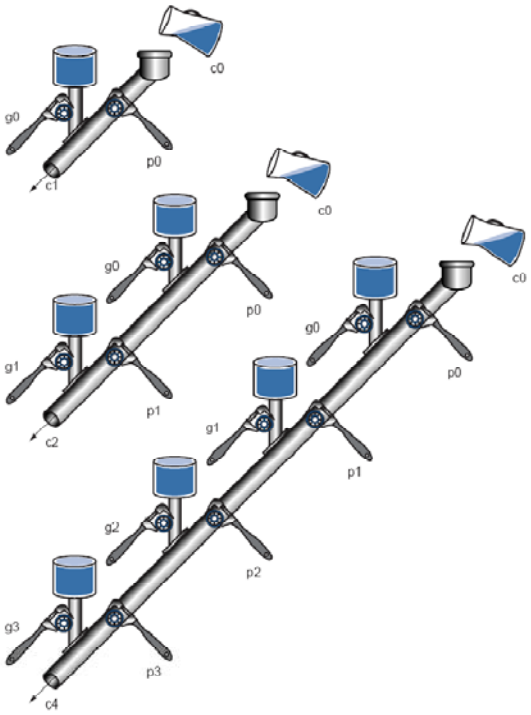
Computer Organization and Architecture, Fall 2010

Principle: Generate, Propagate

- Analogy

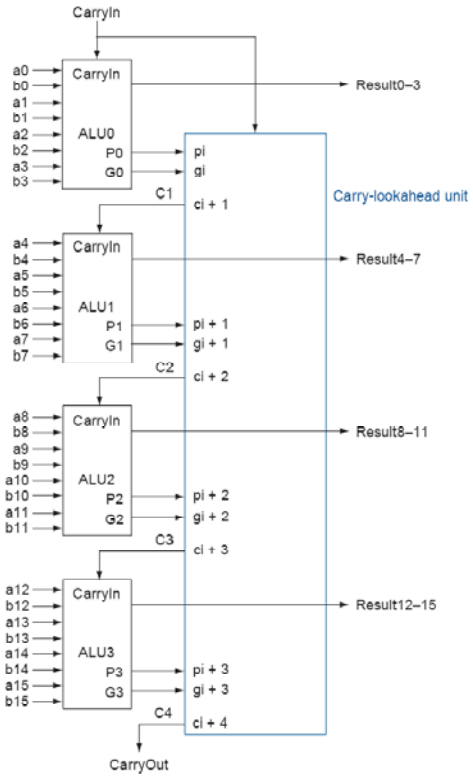
$$
\begin{aligned}
c1 &= g0 + (p0 \cdot c0) \\
c2 &= g1 + (p1 \cdot g0) + (p1 \cdot p0 \cdot c0) \\
c3 &= g2 + (p2 \cdot g1) + (p2 \cdot p1 \cdot g0) + (p2 \cdot p1 \cdot p0 \cdot c0) \\
c4 &= g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) \\
&\quad + (p3 \cdot p2 \cdot p1 \cdot p0 \cdot c0)
\end{aligned}
$$

- C4 is computed once inputs (a0~a3, b0~b3, and c0) are valid.



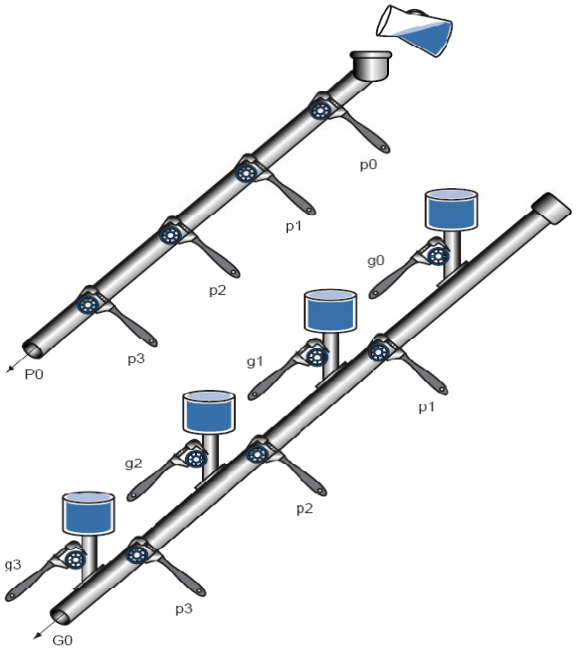
Use hierarchy to build a bigger adder

- Can't build a 16 bit adder this way … (too big)
- Could use ripple carry of 4-bit CLA adders
- Better: user the CLA principle again!



Second Level of Abstraction: CLA

- Reuse the same principle in second level



$$P0 = p3 \cdot p2 \cdot p1 \cdot p0$$
$$P1 = p7 \cdot p6 \cdot p5 \cdot p4$$
$$P2 = p11 \cdot p10 \cdot p9 \cdot p8$$
$$P3 = p15 \cdot p14 \cdot p13 \cdot p12$$

$$G0 = g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0)$$
$$G1 = g7 + (p7 \cdot g6) + (p7 \cdot p6 \cdot g5) + (p7 \cdot p6 \cdot p5 \cdot g4)$$
$$G2 = g11 + (p11 \cdot g10) + (p11 \cdot p10 \cdot g9) + (p11 \cdot p10 \cdot p9 \cdot g8)$$
$$G3 = g15 + (p15 \cdot g14) + (p15 \cdot p14 \cdot g13) + (p15 \cdot p14 \cdot p13 \cdot g12)$$

Text Book : C-41

Second Level Equation

- The same principle as in the first level is used.
- The result of the second level is computed as soon as its inputs are valid.

$$C1 = G0 + (P0 \cdot c0)$$
$$C2 = G1 + (P1 \cdot G0) + (P1 \cdot P0 \cdot c0)$$
$$C3 = G2 + (P2 \cdot G1) + (P2 \cdot P1 \cdot G0) + (P2 \cdot P1 \cdot P0 \cdot c0)$$
$$C4 = G3 + (P3 \cdot G2) + (P3 \cdot P2 \cdot G1) + (P3 \cdot P2 \cdot P1 \cdot G0) + (P3 \cdot P2 \cdot P1 \cdot P0 \cdot c0)$$

EXAMPLE

Both Levels of the Propagate and Generate

Determine the g*i*, p*i*, P*i*, and G*i* values of these two 16-bit numbers:

a: $0001\ 1010\ 0011\ 0011_{two}$
b: $1110\ 0101\ 1110\ 1011_{two}$

Also, what is CarryOut15 (C4)?

ANSWER

Aligning the bits makes it easy to see the values of generate g*i* ($a_i \cdot b_i$) and propagate p*i* ($a_i + b_i$):

a: 0001 1010 0011 0011
b: 1110 0101 1110 1011
g*i*: 0000 0000 0010 0011
p*i*: 1111 1111 1111 1011

where the bits are numbered 15 to 0 from left to right. Next, the "super" propagates (P3, P2, P1, P0) are simply the AND of the lower-level propagates:

$P3 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$

$P2 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$

$P1 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$

$P0 = 1 \cdot 0 \cdot 1 \cdot 1 = 0$

(Cont.)

Department of Electrical Engineering, Feng-Chia University

Text Book : C-44

ANSWER

The "super" generates are more complex, so use the following equations:

$$\begin{aligned}
G0 &= g3 + (p3 \cdot g2) + (p3 \cdot p2 \cdot g1) + (p3 \cdot p2 \cdot p1 \cdot g0) \\
&= 0 + (1 \cdot 0) + (1 \cdot 0 \cdot 1) + (1 \cdot 0 \cdot 1 \cdot 1) = 0 + 0 + 0 + 0 = 0 \\
G1 &= g7 + (p7 \cdot g6) + (p7 \cdot p6 \cdot g5) + (p7 \cdot p6 \cdot p5 \cdot g4) \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 1) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 1 + 0 = 1 \\
G2 &= g11 + (p11 \cdot g10) + (p11 \cdot p10 \cdot g9) + (p11 \cdot p10 \cdot p9 \cdot g8) \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 0 + 0 = 0 \\
G3 &= g15 + (p15 \cdot g14) + (p15 \cdot p14 \cdot g13) + (p15 \cdot p14 \cdot p13 \cdot g12) \\
&= 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0) = 0 + 0 + 0 + 0 = 0
\end{aligned}$$

Finally, CarryOut15 is

$$\begin{aligned}
C4 = {} & G3 + (P3 \cdot G2) + (P3 \cdot P2 \cdot G1) + (P3 \cdot P2 \cdot P1 \cdot G0) \\
& + (P3 \cdot P2 \cdot P1 \cdot P0 \cdot c0) \\
= {} & 0 + (1 \cdot 0) + (1 \cdot 1 \cdot 1) + (1 \cdot 1 \cdot 1 \cdot 0) + (1 \cdot 1 \cdot 1 \cdot 0 \cdot 0) \\
= {} & 0 + 0 + 1 + 0 + 0 = 1
\end{aligned}$$

Hence there *is* a carry out when adding these two 16-bit numbers.

Computer Organization and Architecture, Fall 2010