Lecture 12: Hardware for Arithmetic

- Today's topics:
  - Designing an ALU
  - Carry-lookahead adder
- Reminder: Assignment 5 will be posted in a couple of days (due Thursday 10/12), no class on Thursday 10/19, mid-term exam Tuesday 10/24

DeMorgan's Laws

- $\overline{A + B} = \overline{A} \cdot \overline{B}$

- $\overline{A \cdot B} = \overline{A} + \overline{B}$

- Confirm that these are indeed true

Sum of Products

- Can represent any logic block with the AND, OR, NOT operators
  - Draw the truth table
  - For each true output, represent the corresponding inputs as a product
  - The final equation is a sum of these products

A	B	C	E
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$(A \cdot B \cdot \overline{C}) + (A \cdot C \cdot \overline{B}) + (C \cdot B \cdot \overline{A})$

- Can also use “product of sums”
- Any equation can be implemented with an array of ANDs, followed by an array of ORs

Adder Algorithm

	1	0	0	1
	0	1	0	1
Sum	1	1	1	0
Carry	0	0	0	1

Truth Table for the above operations:

A	B	Cin	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Adder Algorithm

	1	0	0	1
	0	1	0	1
Sum	1	1	1	0
Carry	0	0	0	1

Truth Table for the above operations:

A	B	Cin	Sum	Cout
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

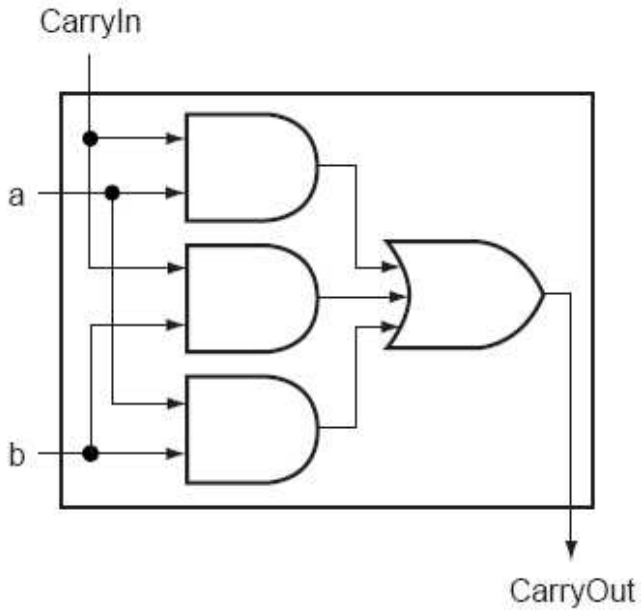
Equations:

$$\text{Sum} = Cin \cdot \overline{A} \cdot \overline{B} + B \cdot \overline{Cin} \cdot \overline{A} + A \cdot \overline{Cin} \cdot \overline{B} + A \cdot B \cdot Cin$$

$$\text{Cout} = A \cdot B \cdot Cin + A \cdot B \cdot \overline{Cin} + A \cdot Cin \cdot \overline{B} + B \cdot Cin \cdot \overline{A}$$

$$= A \cdot B + A \cdot Cin + B \cdot Cin$$

Carry Out Logic



Equations:

$$\text{Sum} = Cin \cdot \overline{A} \cdot \overline{B} + B \cdot \overline{Cin} \cdot \overline{A} + A \cdot \overline{Cin} \cdot \overline{B} + A \cdot B \cdot Cin$$

$$\begin{aligned} \text{Cout} &= A \cdot B \cdot Cin + A \cdot B \cdot \overline{Cin} + A \cdot Cin \cdot \overline{B} + B \cdot Cin \cdot \overline{A} \\ &= A \cdot B + A \cdot Cin + B \cdot Cin \end{aligned}$$

FIGURE B.5.5 Adder hardware for the carry out signal. The rest of the adder hardware is the logic for the Sum output given in the equation on page B-28.

1-Bit ALU with Add, Or, And

- Multiplexor selects between Add, Or, And operations

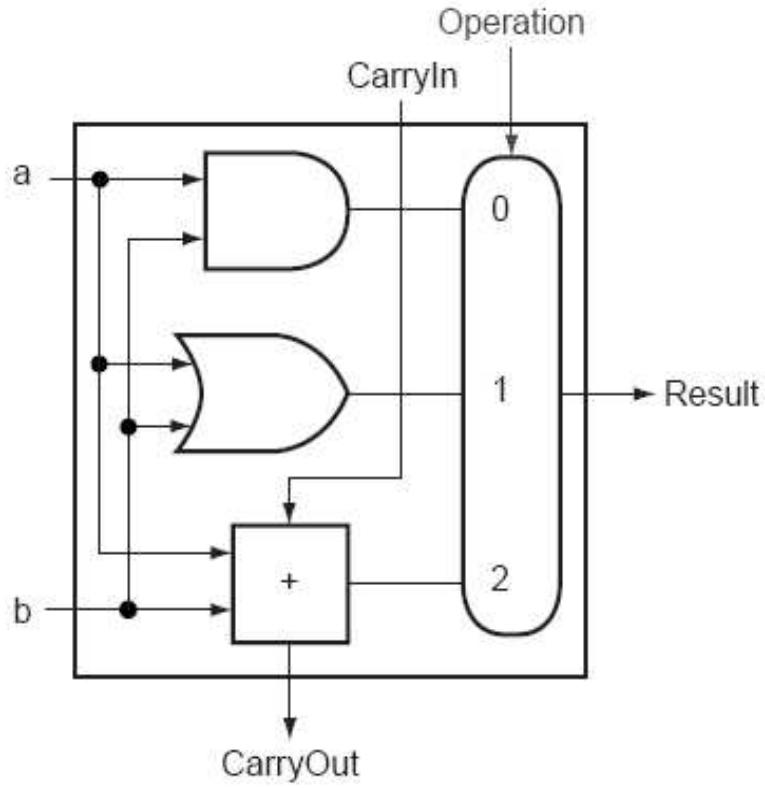


FIGURE B.5.6 A 1-bit ALU that performs AND, OR, and addition (see Figure B.5.5).

32-bit Ripple Carry Adder

1-bit ALUs are connected "in series" with the carry-out of 1 box going into the carry-in of the next box

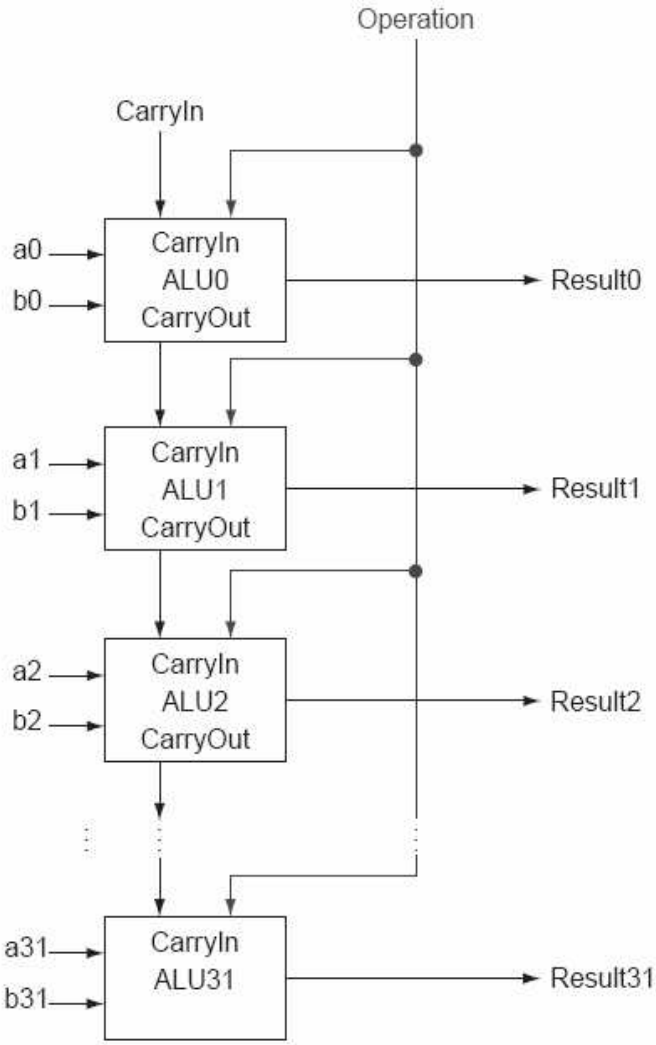


FIGURE B.5.7 A 32-bit ALU constructed from 32 1-bit ALUs. CarryOut of the less significant bit is connected to the CarryIn of the more significant bit. This organization is called ripple carry.

Incorporating Subtraction

Incorporating Subtraction

Must invert bits of B and add a 1

- Include an inverter
- CarryIn for the first bit is 1
- The CarryIn signal (for the first bit) can be the same as the Binvert signal

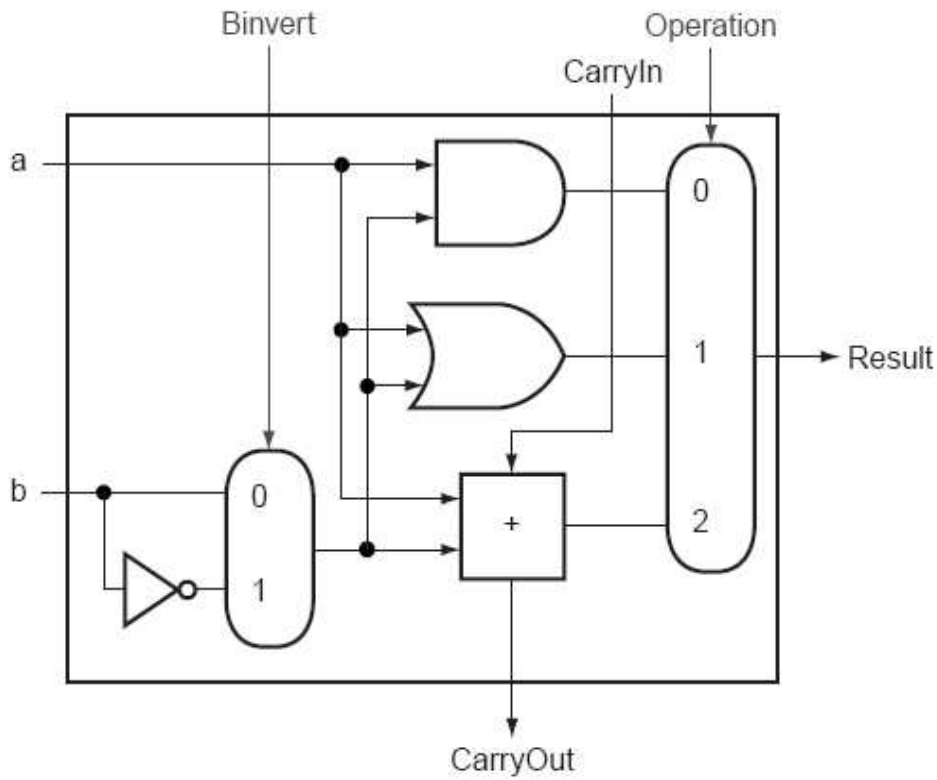


FIGURE B.5.8 A 1-bit ALU that performs AND, OR, and addition on a and b or a and $\overline{b}$. By selecting $\overline{b}$ (Binvert = 1) and setting CarryIn to 1 in the least significant bit of the ALU, we get two's complement subtraction of b from a instead of addition of b to a.

Incorporating NOR

Incorporating NOR

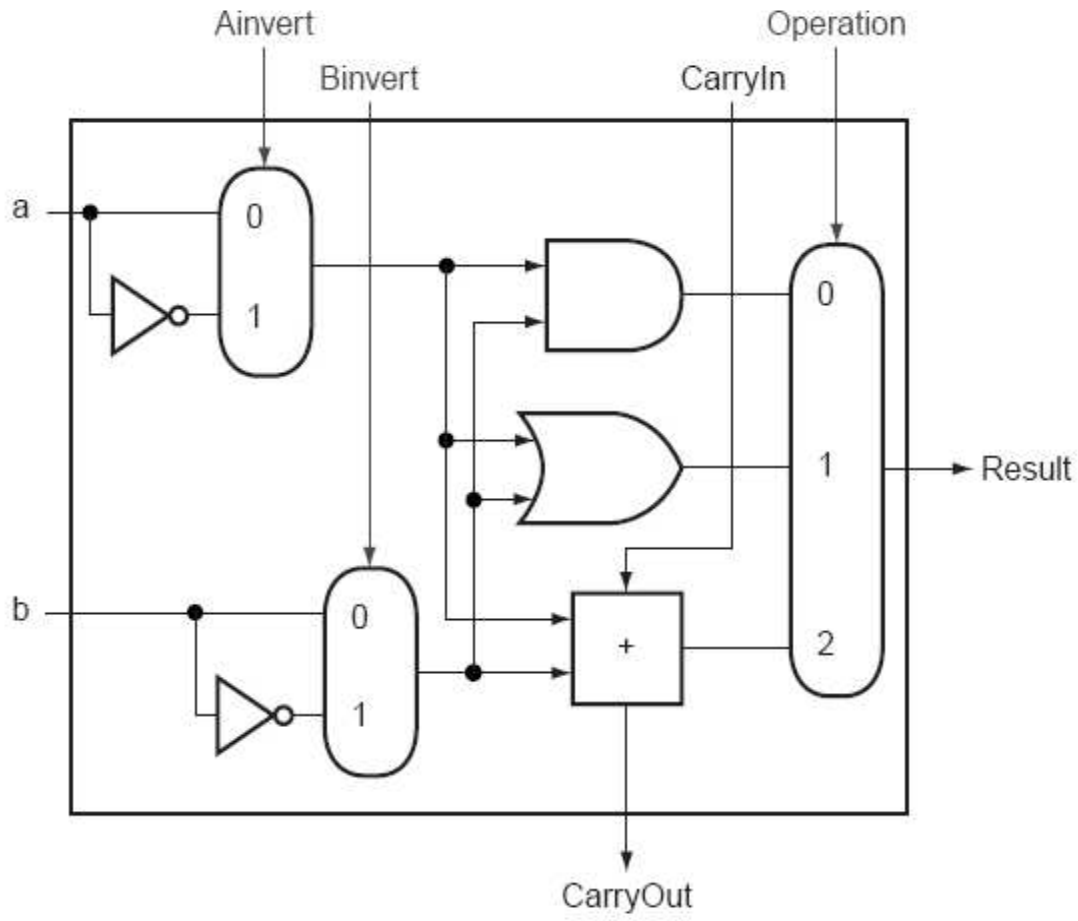
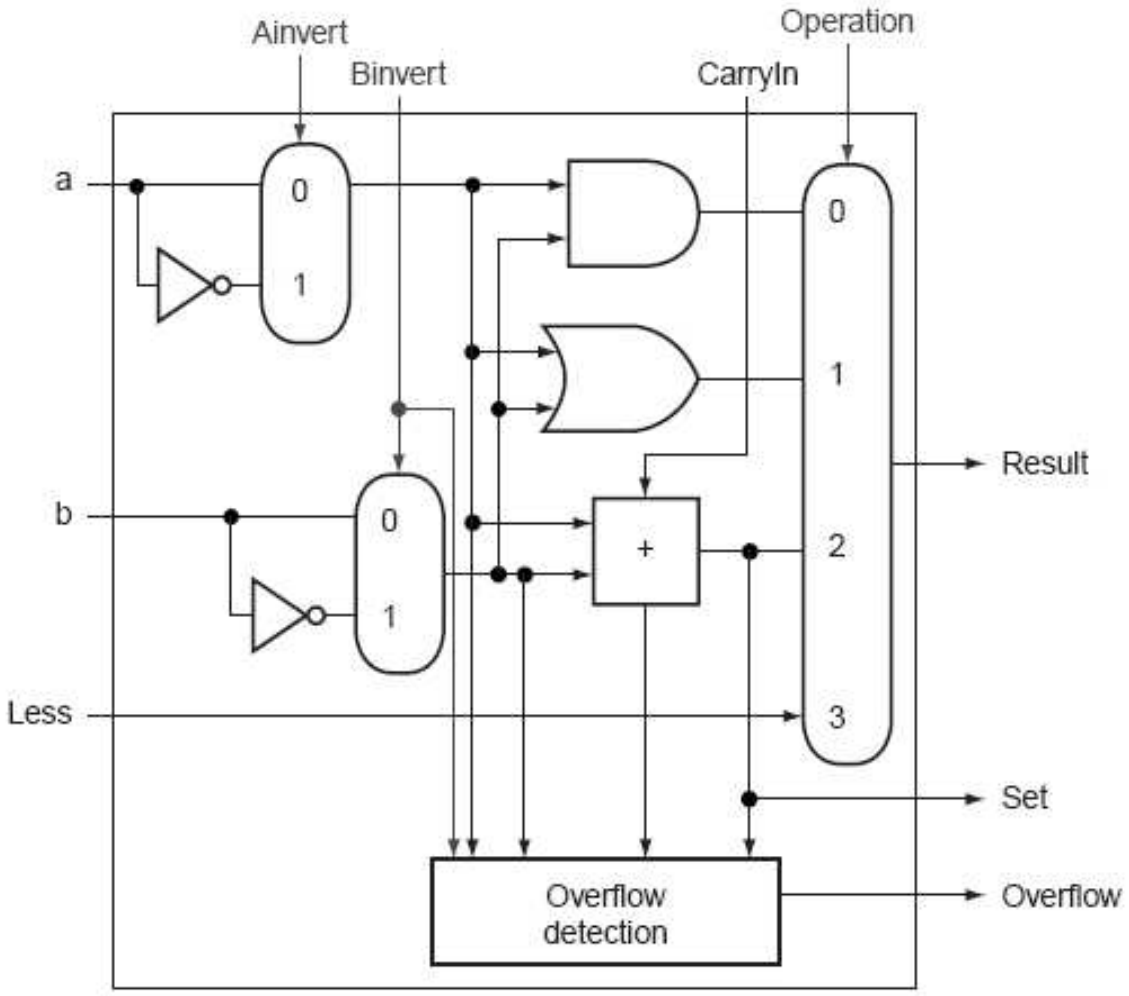


FIGURE B.5.9 A 1-bit ALU that performs AND, OR, and addition on a and b or $\overline{a}$ and $\overline{b}$. By selecting $\overline{a}$ (Ainvert = 1) and $\overline{b}$ (Binvert = 1), we get a NOR b instead of a AND b.

Incorporating slt

Incorporating slt

- Perform a – b and check the sign
- New signal (Less) that is zero for ALU boxes 1-31
- The 31st box has a unit to detect overflow and sign – the sign bit serves as the Less signal for the 0th box



Incorporating beq

- Perform a – b and confirm that the result is all zero's

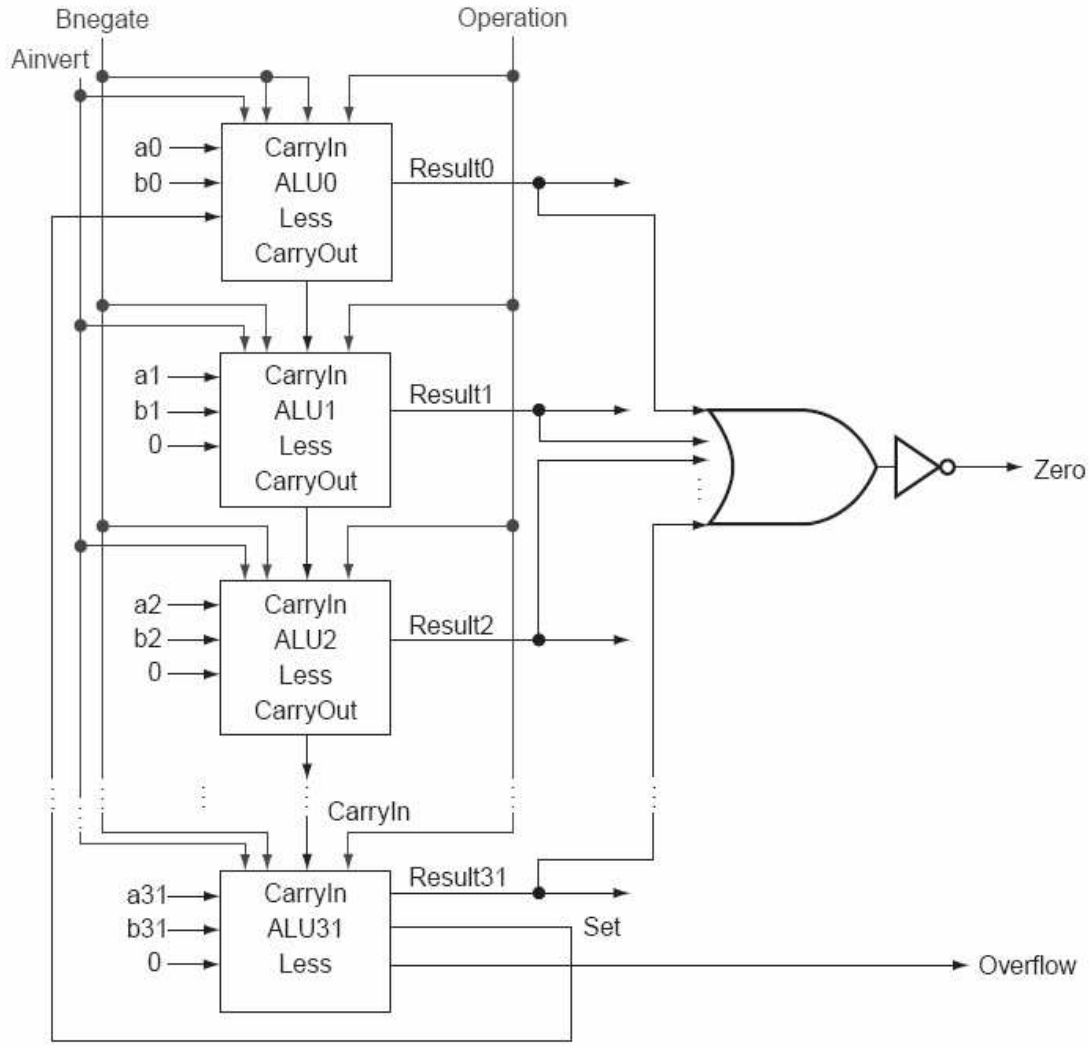


FIGURE B.5.12 The final 32-bit ALU. This adds a Zero detector to Figure B.5.11.

Control Lines

What are the values of the control lines and what operations do they correspond to?

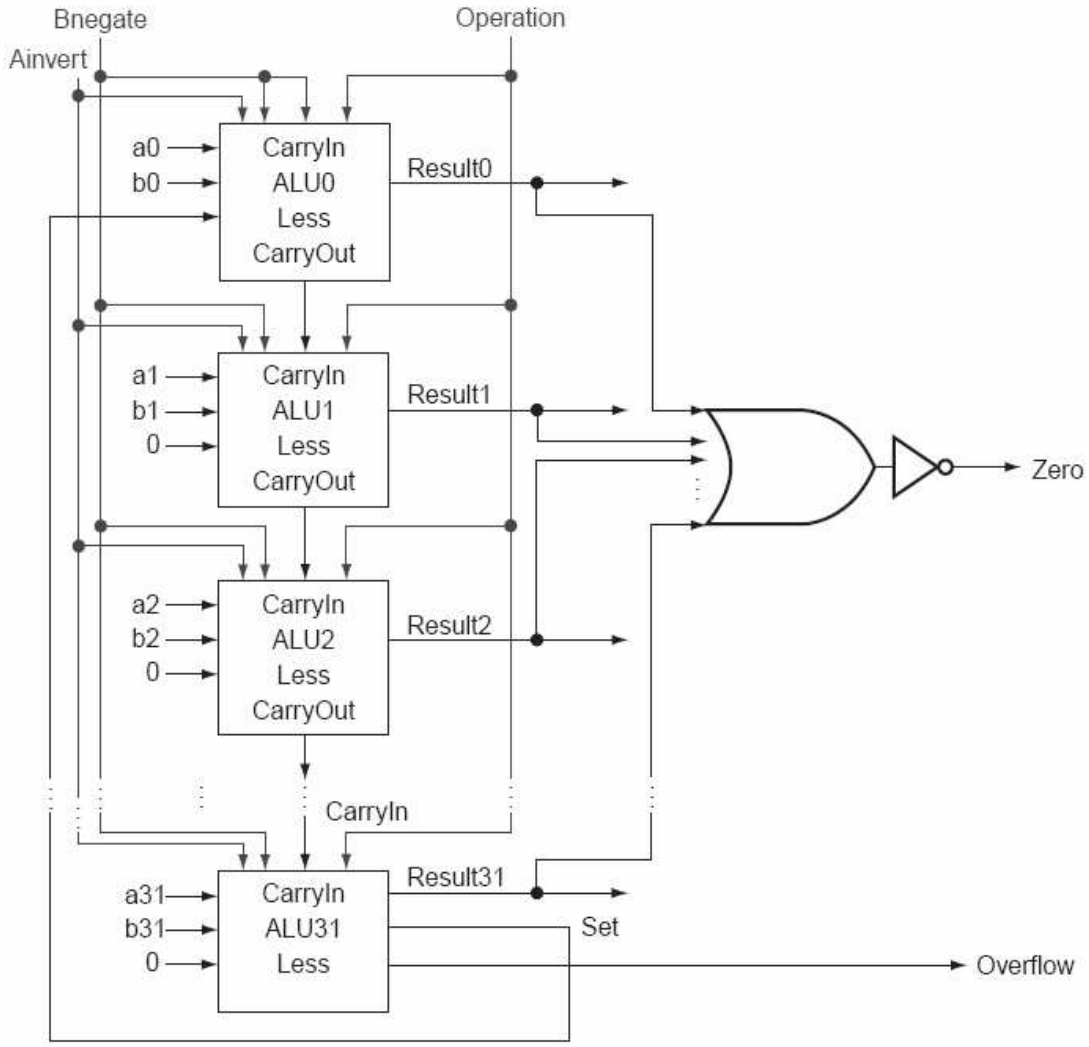
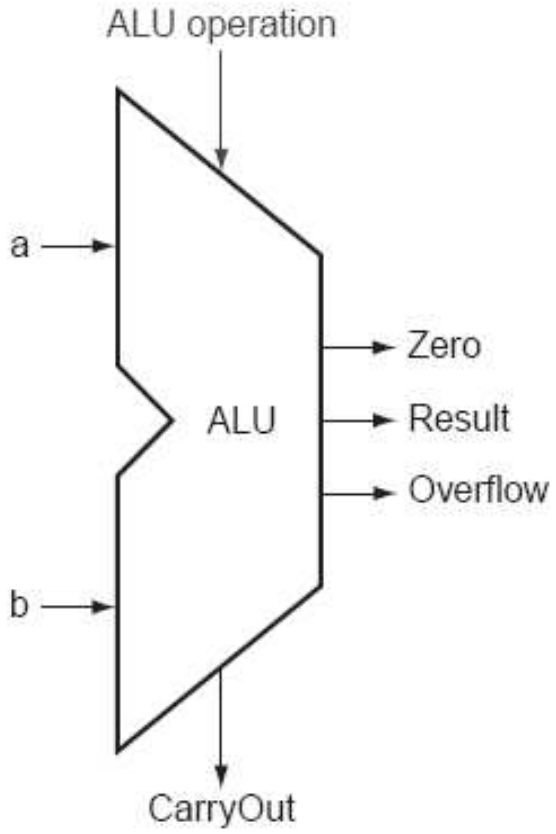


FIGURE B.5.12 The final 32-bit ALU. This adds a Zero detector to Figure B.5.11.

Control Lines

What are the values of the control lines and what operations do they correspond to?

	Ai	Bn	Op
AND	0	0	00
OR	0	0	01
Add	0	0	10
Sub	0	1	10
SLT	0	1	11
NOR	1	1	00



Speed of Ripple Carry

- The carry propagates thru every 1-bit box: each 1-bit box sequentially implements AND and OR – total delay is the time to go through 64 gates!

- We've already seen that any logic equation can be expressed as the sum of products – so it should be possible to compute the result by going through only 2 gates!

- Caveat: need many parallel gates and each gate may have a very large number of inputs – it is difficult to efficiently build such large gates, so we'll find a compromise:
  - moderate number of gates
  - moderate number of inputs to each gate
  - moderate number of sequential gates traversed

Computing CarryOut

CarryIn1 = b0.CarryIn0 + a0.CarryIn0 + a0.b0
CarryIn2 = b1.CarryIn1 + a1.CarryIn1 + a1.b1
= b1.b0.c0 + b1.a0.c0 + b1.a0.b0 +
a1.b0.c0 + a1.a0.c0 + a1.a0.b0 + a1.b1

…

CarryIn32 = a really large sum of really large products

- Potentially fast implementation as the result is computed by going thru just 2 levels of logic – unfortunately, each gate is enormous and slow

Generate and Propagate

Equation re-phrased:

$$C_{i+1} = a_i.b_i + a_i.C_i + b_i.C_i$$
$$= (a_i.b_i) + (a_i + b_i).C_i$$

Stated verbally, the current pair of bits will *generate* a carry if they are both 1 and the current pair of bits will *propagate* a carry if either is 1

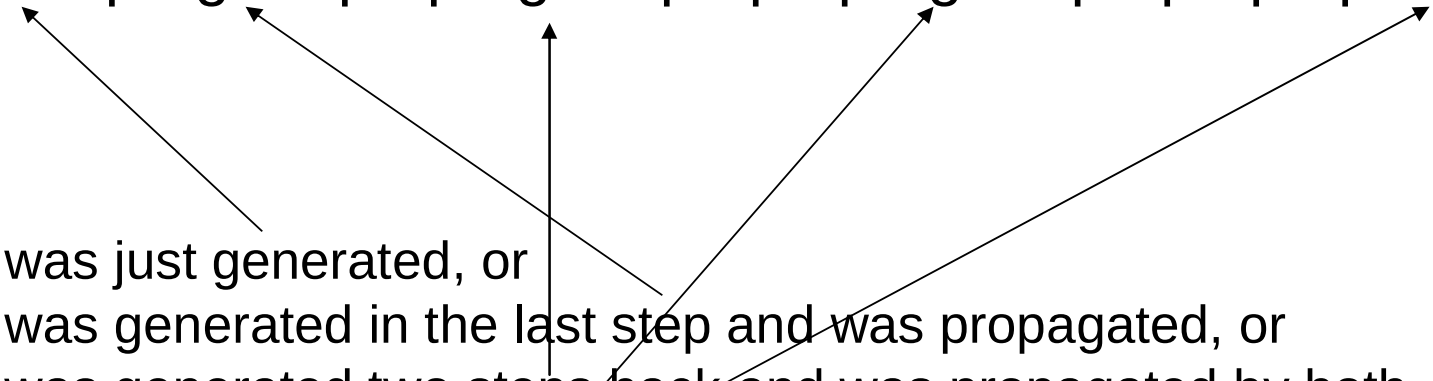
Generate signal = $a_i.b_i$

Propagate signal = $a_i + b_i$

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

Generate and Propagate

$$c1 = g0 + p0.c0$$
$$c2 = g1 + p1.c1$$
$$= g1 + p1.g0 + p1.p0.c0$$
$$c3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0$$
$$c4 = g3 + p3.g2 + p3.p2.g1 + p3.p2.p1.g0 + p3.p2.p1.p0.c0$$



Either,

a carry was just generated, or

a carry was generated in the last step and was propagated, or

a carry was generated two steps back and was propagated by both the next two stages, or

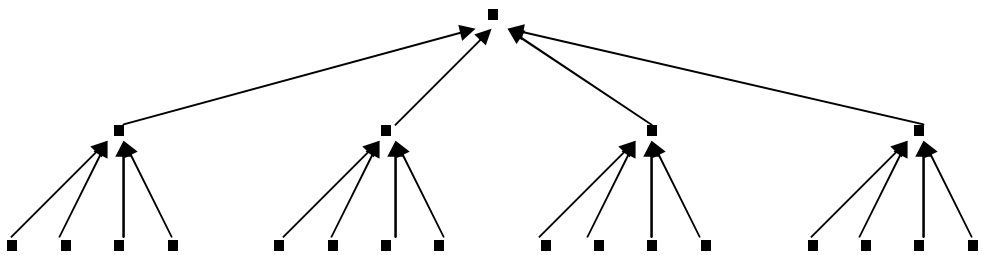
a carry was generated N steps back and was propagated by every single one of the N next stages

Divide and Conquer

- The equations on the previous slide are still difficult to implement as logic functions – for the 32nd bit, we must AND every single propagate bit to determine what becomes of c0 (among other things)

- Hence, the bits are broken into groups (of 4) and each group computes its group-generate and group-propagate

- For example, to add 32 numbers, you can partition the task as a tree



P and G for 4-bit Blocks

- Compute P0 and G0 (super-propagate and super-generate) for the first group of 4 bits (and similarly for other groups of 4 bits)

  $P0 = p0.p1.p2.p3$

  $G0 = g3 + g2.p3 + g1.p2.p3 + g0.p1.p2.p3$

- Carry out of the first group of 4 bits is

  $C1 = G0 + P0.c0$

  $C2 = G1 + P1.G0 + P1.P0.c0$

  …

- By having a tree of sub-computations, each AND, OR gate has few inputs and logic signals have to travel through a modest set of gates (equal to the height of the tree)

Example

Add	A	0001	1010	0011	0011
and	B	1110	0101	1110	1011
	g	0000	0000	0010	0011
	p	1111	1111	1111	1011
	P	1	1	1	0
	G	0	0	1	0

C4 = 1

Carry Look-Ahead Adder

- 16-bit Ripple-carry takes 32 steps
- This design takes how many steps?

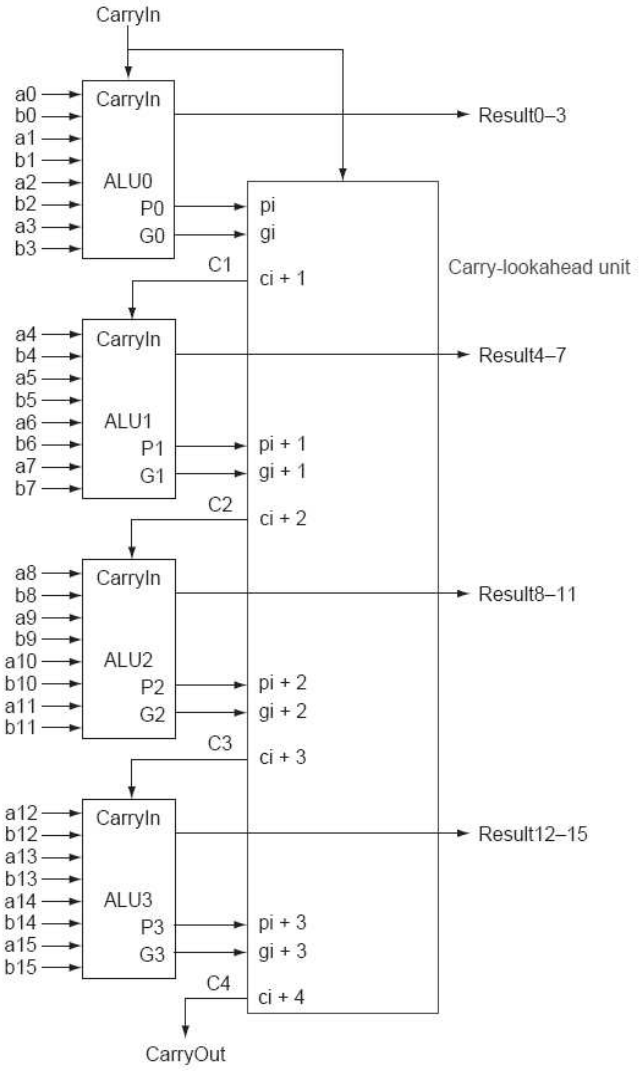


FIGURE B.6.3 Four 4-bit ALUs using carry lookahead to form a 16-bit adder. Note that the carries come from the carry-lookahead unit, not from the 4-bit ALUs.

Title

- Bullet