ARITHMETIC AND LOGIC UNIT

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

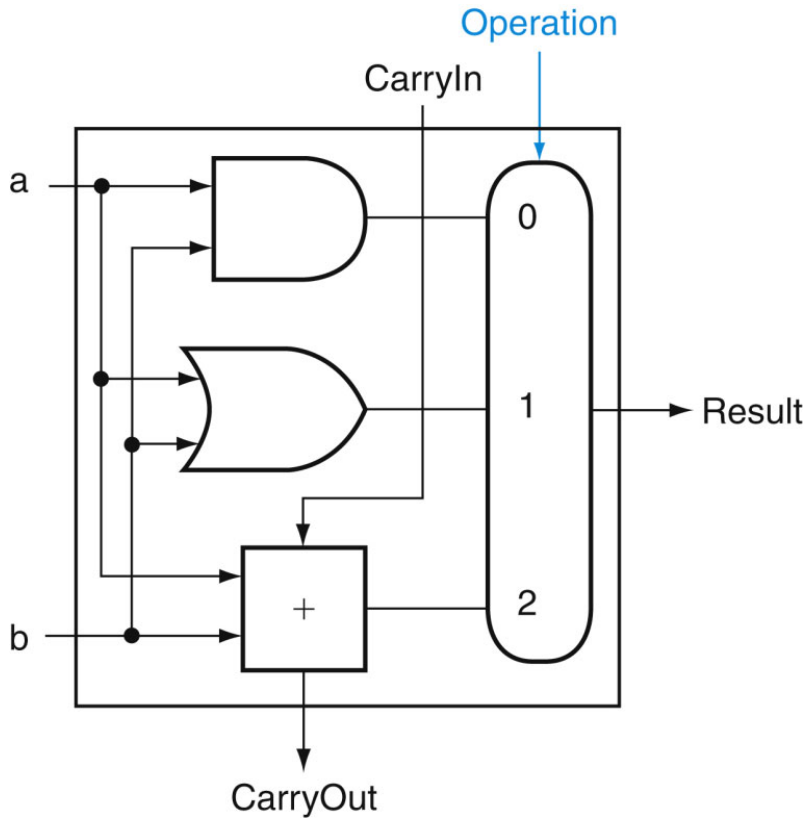
- Notes
  - Homework is due tonight
    - Verify your submitted file before midnight
  - No lecture on Thursday
    - Instead, a review on common mistakes and questions by the TAs
- This lecture
  - How to build an arithmetic and logic unit (ALU)

Recall: Common Logic Block

- Any Boolean function can be represented by a truth table and sum of products (two gate levels)
- An n-input decoder takes n inputs, based on which only one out of 2^n outputs is activated
- A multiplexer (or selector) reflects one of n inputs on the output depending on the value of the select bits
- A full adder generates the sum and carry for each bit position

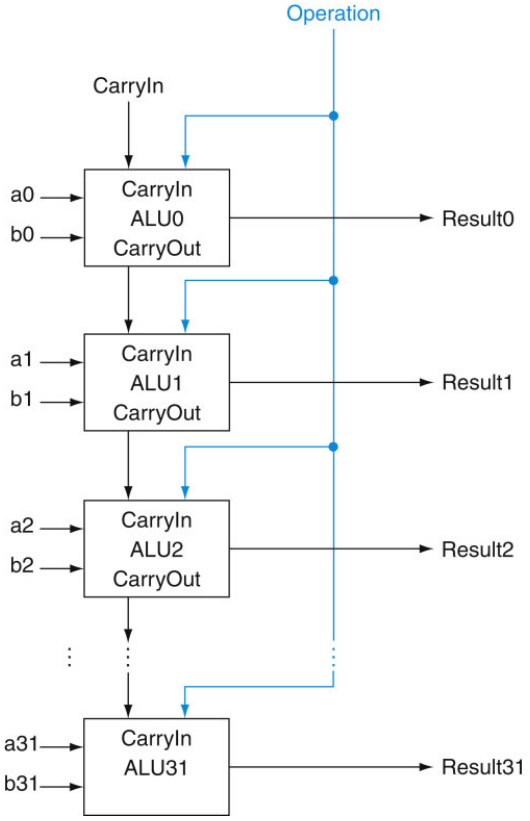
Multi-Function Blocks

- A 1-bit ALU with ADD, OR, and AND operations
  - A multiplexer selects between the operations



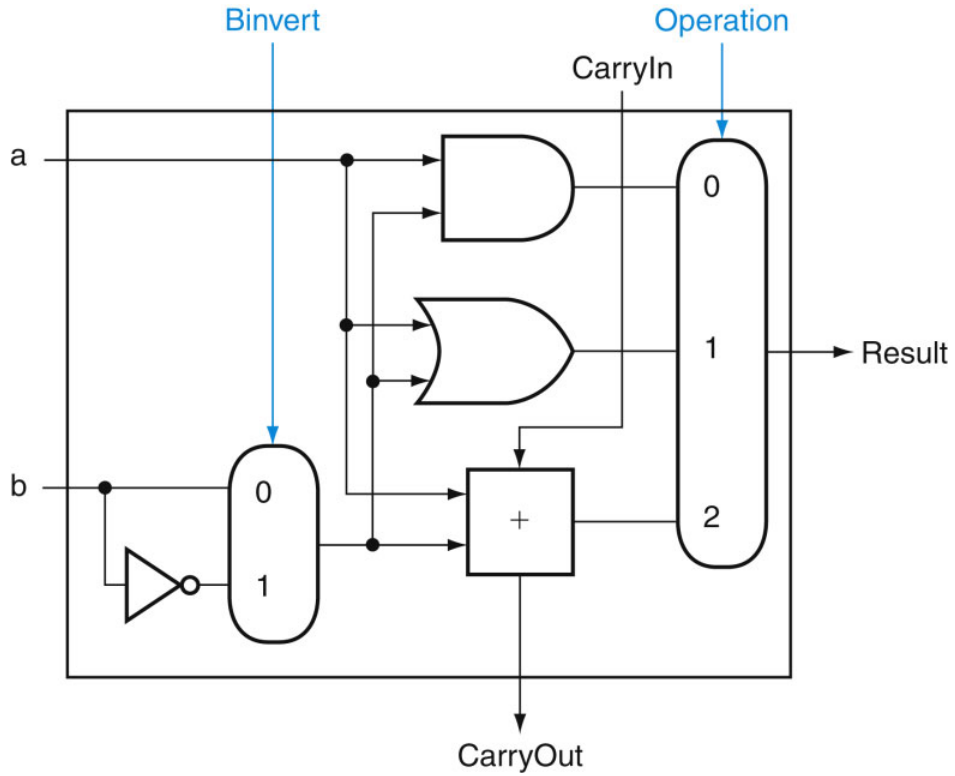
Multibit ALU

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



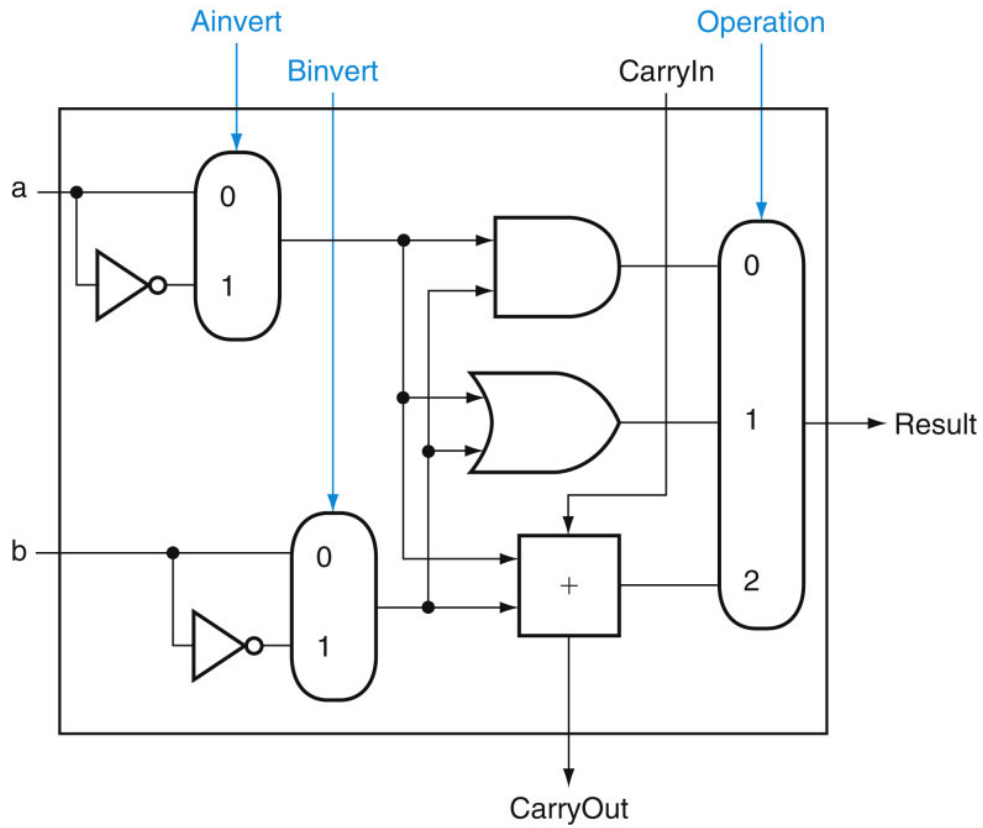
Incorporating Subtraction

- 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



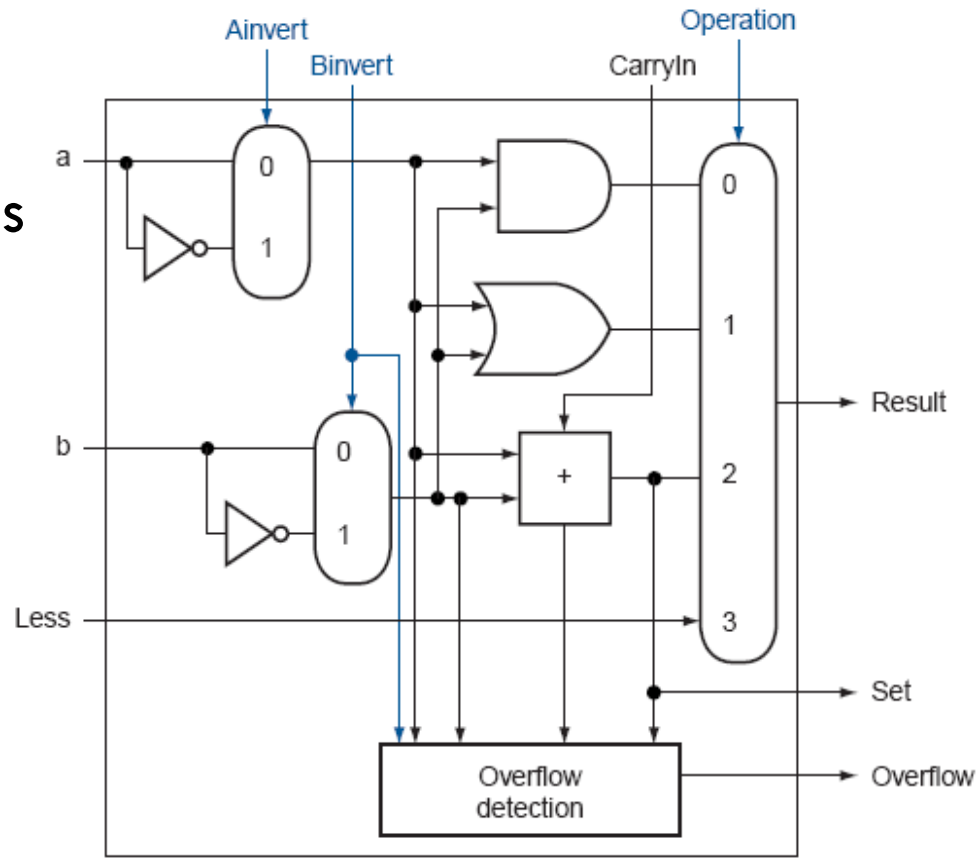
Incorporating NAND and NOR

- Must invert bits of A and B to realize NAND and NOR
  - Recall DeMorgan's Laws



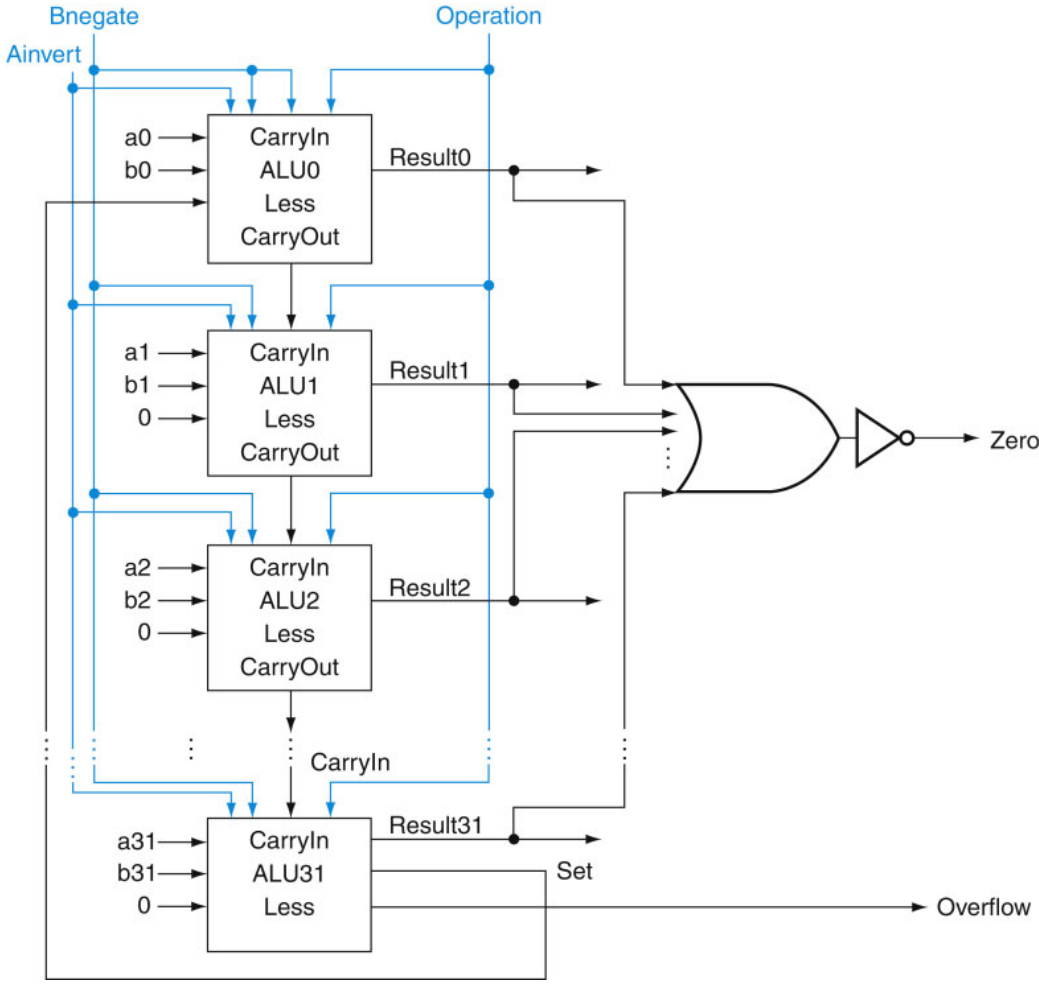
Incorporating Comparison

- For example: the slt instruction
  - 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 Comparison

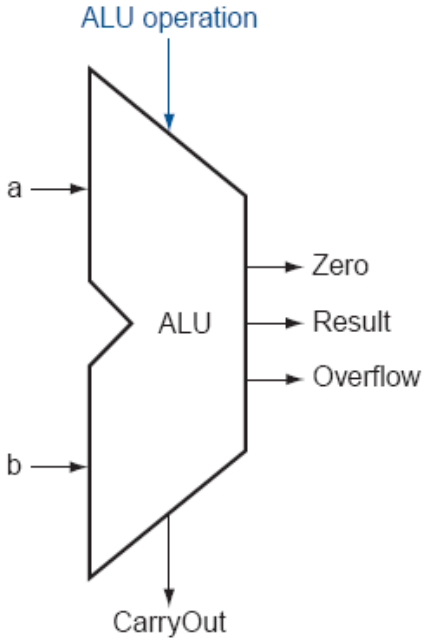
- For example: the beq instruction
  - Perform a – b and confirm that the result is all zero's
  - What about bne?
  - Control lines determine what operations to be done.



Incorporating Comparison

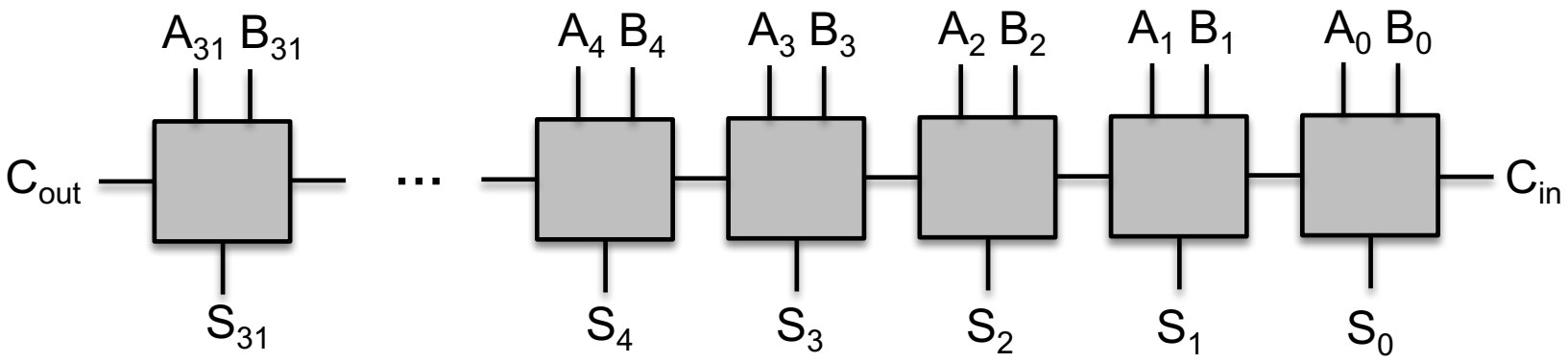
- For example: the beq instruction
  - Perform a – b and confirm that the result is all zero's
  - What about bne?
  - Control lines determine what operations to be done.

	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



Carry Ripple Adder

- Simplest design by cascading 1-bit boxes
  - Each 1-bit box sequentially implements AND and OR
  - Critical path: the total delay is the time to go through 64 gates
- How to make a 32-bit addition faster?



Carry Ripple Adder

- Simplest design by cascading 1-bit boxes
  - Each 1-bit box sequentially implements AND and OR
  - Critical path: the total delay is the time to go through 64 gates
- How to make a 32-bit addition faster?
- Recall: any logic equation can be expressed as the sum of products (only 2 gate levels!)
  - Challenges: many parallel gates with very large inputs
  - Solution: we'll find a compromise

Fast Adder

- Computing carry-outs
  - 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
- Each gate is enormous and slow!

Fast Adder

- Computing carry-outs: equation re-phrased
- $C_{i+1} = a_i.b_i + a_i.C_i + b_i.C_i$
- $= (a_i.b_i) + (a_i + b_i).C_i$

- Generate signal = $a_i.b_i$
  - The current pair of bits will generate a carry if they are both 1
- Propagate signal = $a_i + b_i$
  - The current pair of bits will propagate a carry if either is 1

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

Fast Adder

- Computing carry-outs: example

$c1 = g0 + p0.c0$

$c2 = g1 + p1.c1 \;\; = g1 + p1.g0 + p1.p0.c0$

$c3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0$

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

(1) (2) (3) (4) (4)

Either,

(1) a carry was just generated, or

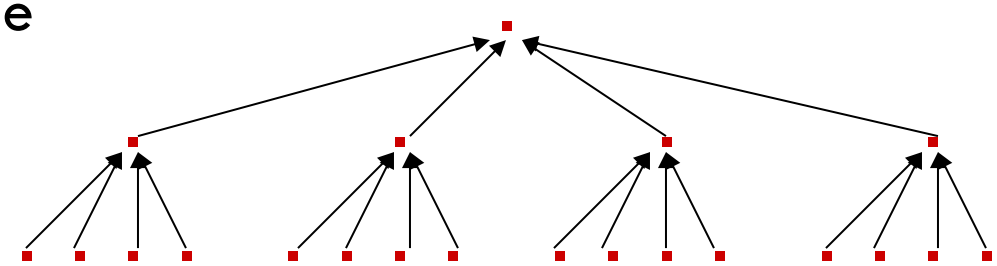
(2) a carry was generated in the last step and was propagated, or

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

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

Fast Adder

- Divide and Conquer
  - Challenge: for the 32nd bit, we must AND every single propagate bit to determine what becomes of c0 (among other things)
  - Solution: 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



Fast Adder

- 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
    - 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)
  - 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)

Fast Adder

- Example

Add	A	0001	1010	0011	0011
	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

Fast Adder: Carry Look-Ahead

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