EECE 417 Computer Systems Architecture

Department of Electrical and Computer Engineering

Howard University

Charles Kim

Spring 2007

Computer Organization and Design (3rd Ed)

-The Hardware/Software Interface

by

David A. Patterson
John L. Hennessy

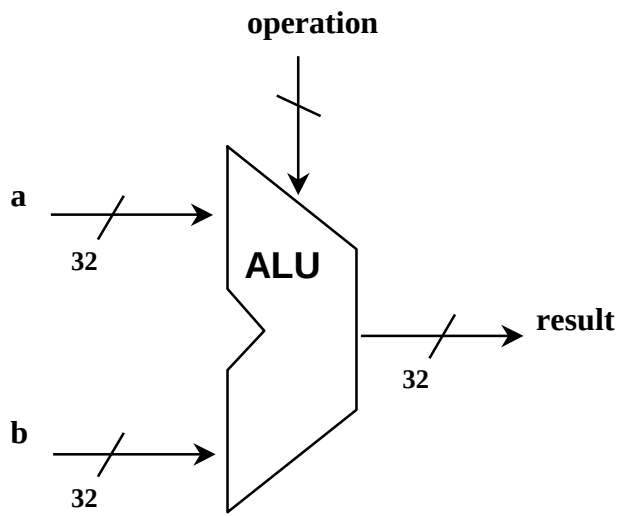
Chapter 4 - Part B

Transition to Chapter 5

Review on ALU

Lets Build a Processor

- Almost ready to move into chapter 5 and start building a processor
- let's review Boolean Logic and build the ALU we'll need



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

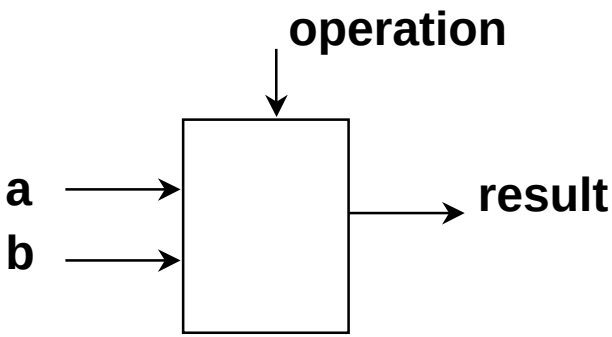
- **Show the truth table for these three functions.**

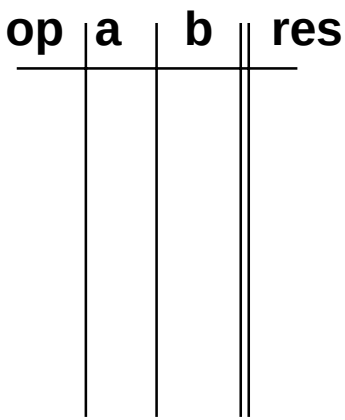
- **Show the Boolean equations for these three functions.**

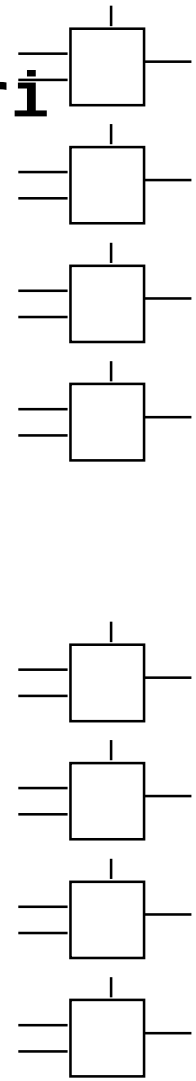
- **Show an implementation consisting of inverters, AND, and OR gates.**

An ALU (arithmetic logic unit)

- **Let's build an ALU to support the `andi` and `ori` instructions**
  - **we'll just build a 1 bit ALU, and use 32 of them**



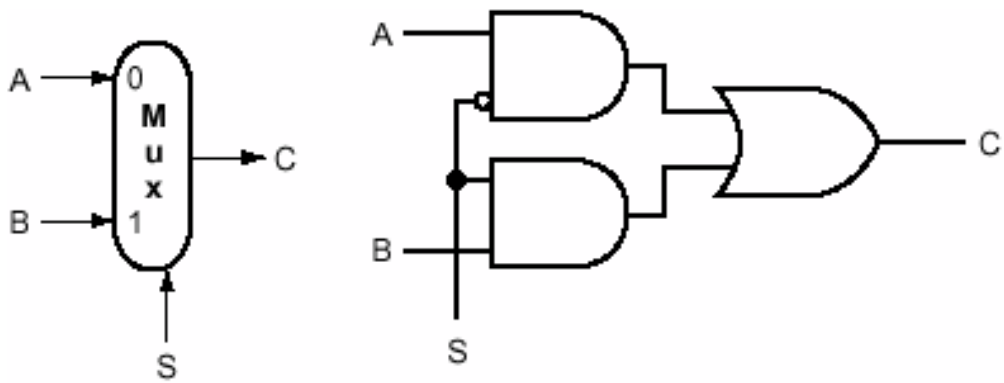




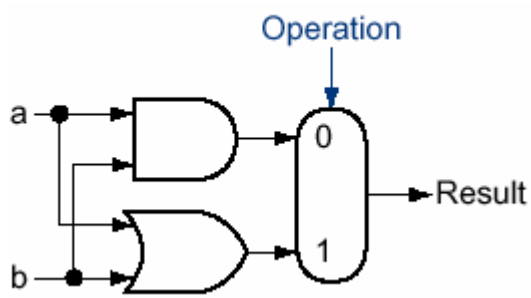
- **Possible Implementation (sum-of-products):**

Review: The Multiplexor

- **Selects one of the inputs to be the output, based on a control input**

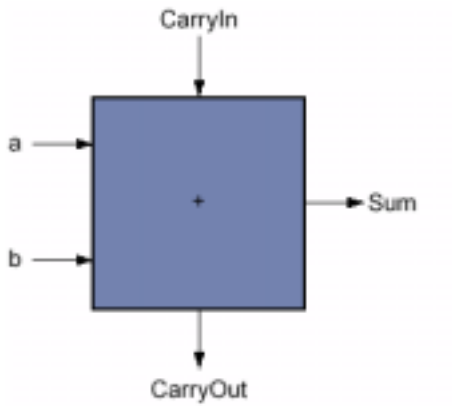


- **Lets build our 1-bit ALU using a MUX:**



Different Implementations

- 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 Adder (ALU for addition):

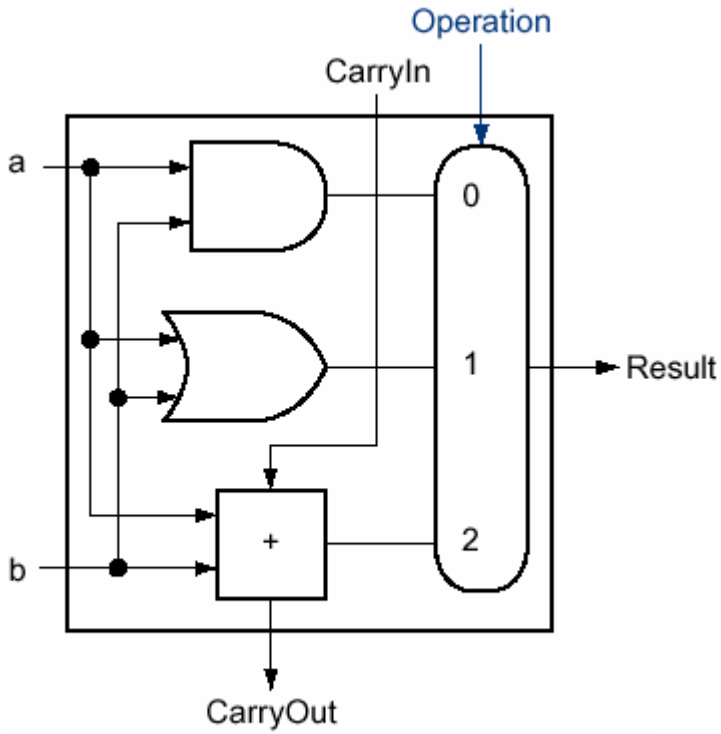


Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0+0+0=00_{two}$
0	0	1	0	1	$0+0+1=01_{two}$
0	1	0	0	1	$0+1+0=01_{two}$
0	1	1	1	0	$0+1+1=10_{two}$
1	0	0	0	1	$1+0+0=01_{two}$
1	0	1	1	0	$1+0+1=10_{two}$
1	1	0	1	0	$1+1+0=10_{two}$
1	1	1	1	1	$1+1+1=11_{two}$

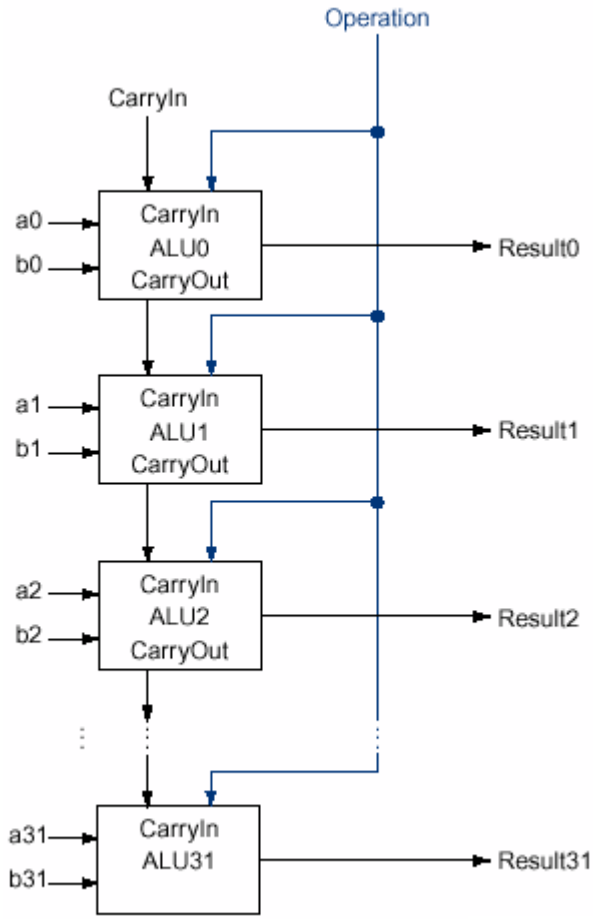
$$c_{out} = a\ b + a\ c_{in} + b\ c_{in}$$
$$sum = a\ \text{xor}\ b\ \text{xor}\ c_{in}$$

- How could we build a 1-bit ALU for add, and, and or?
- How could we build a 32-bit ALU?

Building a 32 bit ALU



A 1-bit ALU that performs AND, OR, and addition

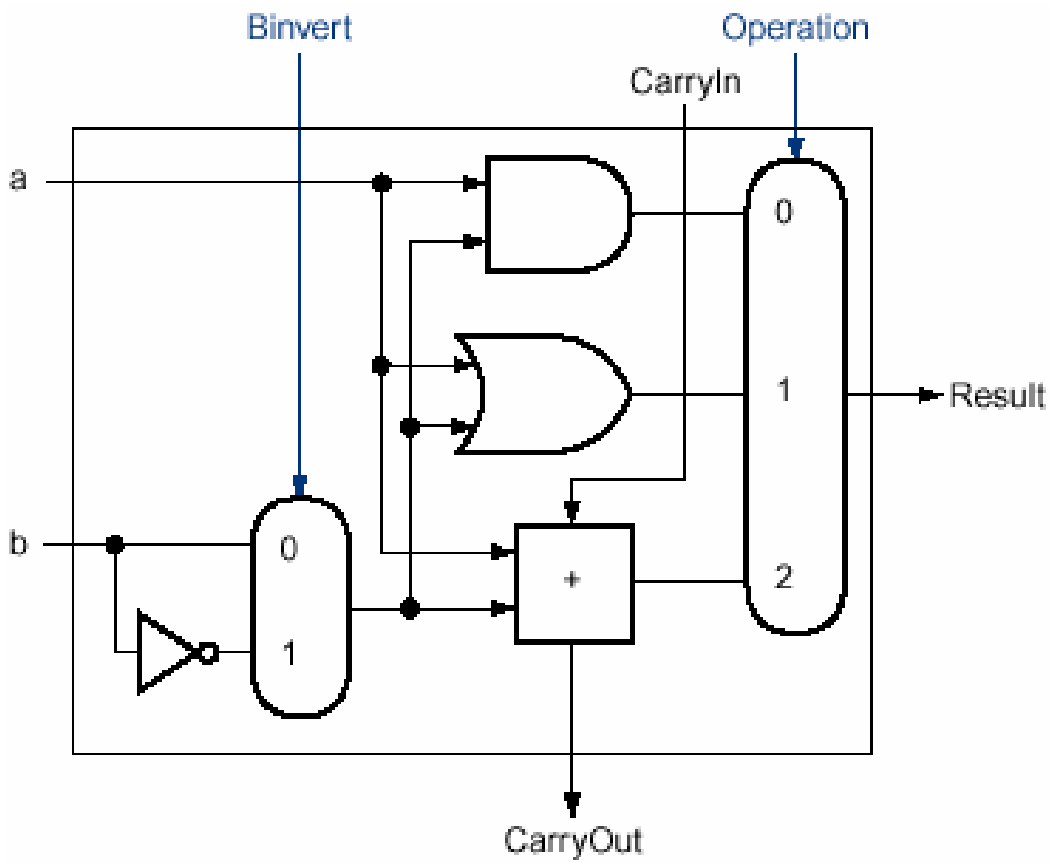


A 32-bit ALU constructed from 32 1-bit ALUs.

What about subtraction (a – b) ?

- **Two's complement approach: just negate b and add.**
- **How do we ne**

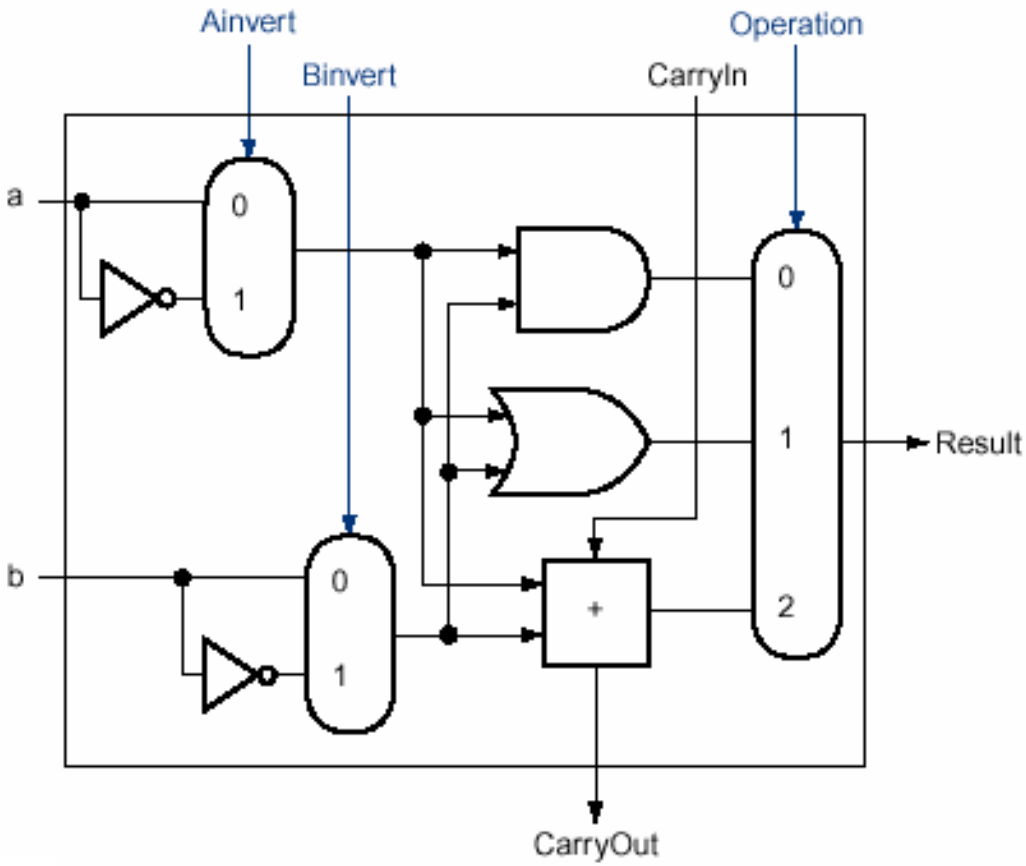
- **A very clever solution:**



A 1-bit ALU that performs AND, OR, and addition on a and b or a and $\bar{b}$.

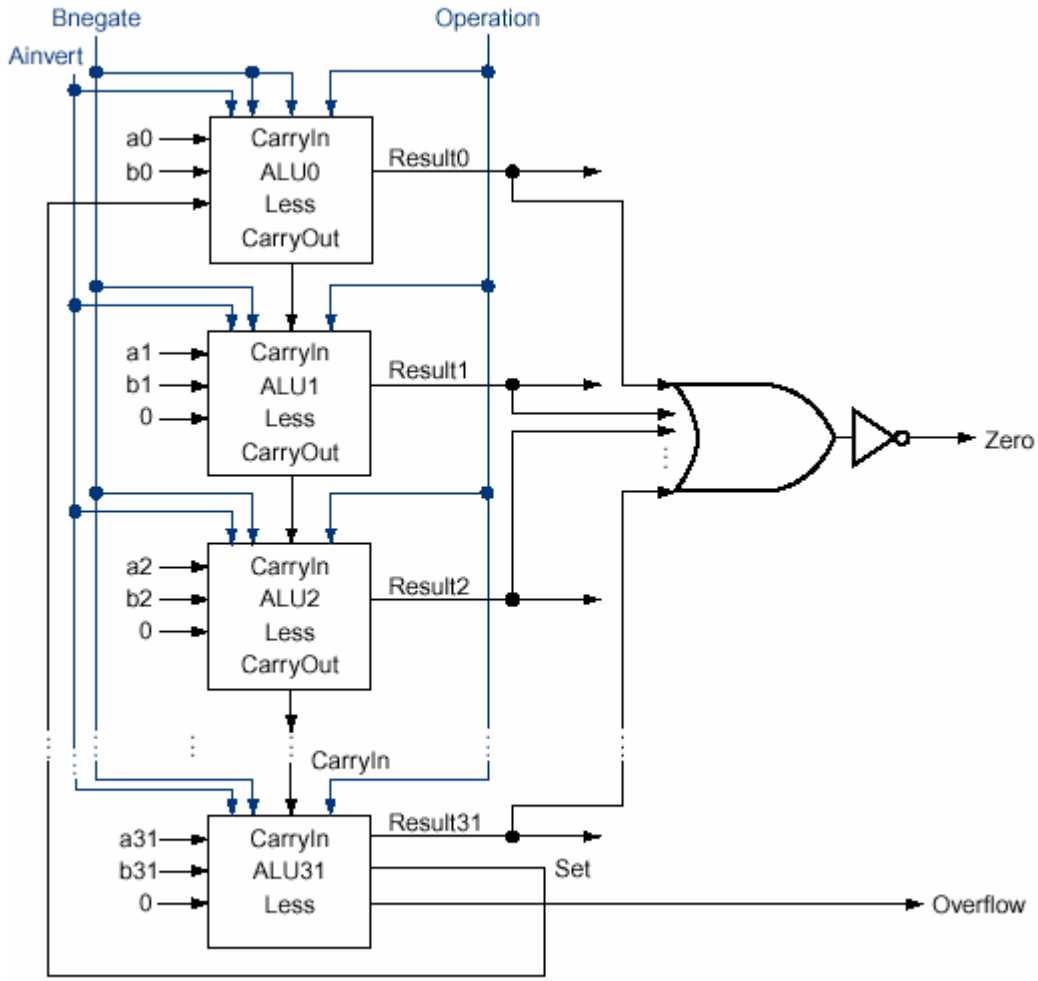
Adding a NOR function

- **Can also choose to invert a. How do we get "a NOR b" ?**

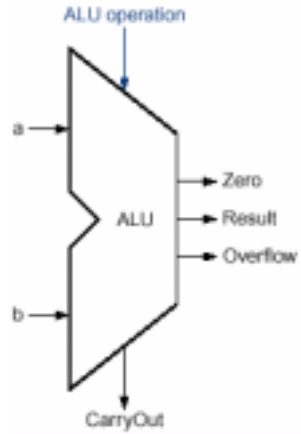


A 1-bit ALU that performs AND, OR, and addition on a and b or $\overline{a}$ and $\overline{b}$.

Final ALU



The final 32-bit ALU.



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")**