MIPS ALU

Review

- Steps to writing (stateless) circuits:
  - Create a truth table
    - Go through all different combinations of inputs
    - For each row, generate each output based on the problem description
  - Create a logic function (one per output)
    - Use only output rows that are ‘true’
    - Use inversion if the input is 0
    - And all of the inputs on a row
    - Or all of the rows

Review

- Steps to writing (stateless) circuits:
  - K-map (one per logic function)
    - Split half of your inputs and label them as columns and the other half as rows
    - Make your labels differ by 1 bit across columns/rows
    - Plot ‘true’ outputs in cells
    - Draw circles around cells in powers of two
    - Minimize number of circles
    - Maximize the size of the circle

Review

- Steps to writing (stateless) circuits:
  - Recreate logic function (one per K-map)
    - Again, use inversion if the input is a 0
    - And together the constant (not changing) inputs in a given circle
    - Or together the results from each circle

In-Class Exercise 4-1 Question

- Assuming that X consists of 3 bits – x2, x1, x0 – write a logic function that is true if and only if X when interpreted as an unsigned binary integer is less than 4

In-Class Exercise 4-1 Answer

X2	X1	X0		O
0	0	0		1
0	0	1		1
0	1	0		1
0	1	1		1
1	0	0		0
1	0	1		0
1	1	0		0
1	1	1		0

In-Class Exercise 4-1 Answer

X2	X1	X0		O
0	0	0		1
0	0	1		1
0	1	0		1
0	1	1		1
1	0	0		0
1	0	1		0
1	1	0		0
1	1	1		0

O = X2’X1’X0’ + X2’X1’X0 + X2’X1X0’ + X2’X1X0

In-Class Exercise 4-1 Answer

O = X2’X1’X0’ + X2’X1’X0 + X2’X1X0’ + X2’X1X0

X2X1

X0

	00	01	11	10
0	1	1		
1	1	1		

In-Class Exercise 4-1 Answer

X2X1

X0

	00	01	11	10
0	1	1		
1	1	1		

O = X2’

In-Class Exercise 4-2 Question

- Assuming that X consists of 3 bits – x2, x1, x0 – write a logic function that is true if and only if X when interpreted as an unsigned binary integer is odd

In-Class Exercise 4-2 Answer

X2	X1	X0		O
0	0	0		0
0	0	1		1
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1

In-Class Exercise 4-2 Answer

X2	X1	X0		O
0	0	0		0
0	0	1		1
0	1	0		0
0	1	1		1
1	0	0		0
1	0	1		1
1	1	0		0
1	1	1		1

O =X2’X1’X0 + X2’X1X0 + X2X1’X0 + X2X1X0

In-Class Exercise 4-2 Answer

O =X2’X1’X0 + X2’X1X0 + X2X1’X0 + X2X1X0

X2X1

X0

	00	01	11	10
0				
1	1	1	1	1

In-Class Exercise 4-2 Answer

X2X1

X0

	00	01	11	10
0				
1	1	1	1	1

O = X0

Exercise – Design a selector?

- I need a circuit that takes two input bits, a and b, and a selector bit s. The function output, f, is:
  - if s=0, f=a
  - if s=1, f=b.

Selector

s	a	b	f
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Selector

s	a	b	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

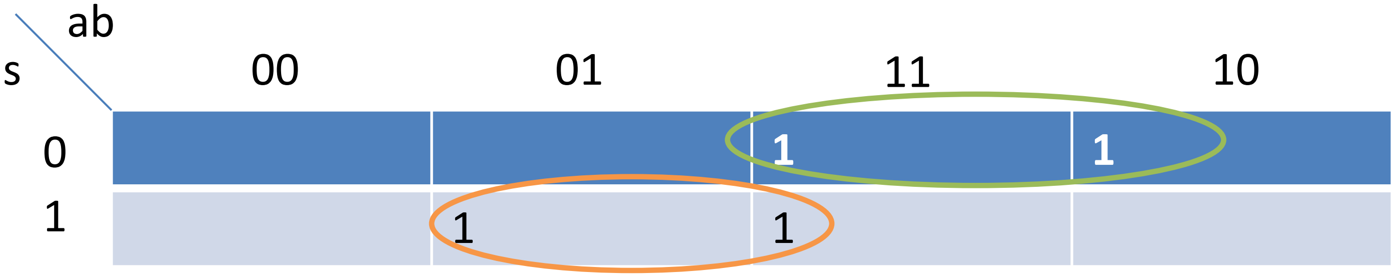
K-map

- F=s’ab’+s’ab+sa’b+sab

ab s	00	01	11	10
0			1	1
1		1	1	

K-map

- F=s’ab’+s’ab+sa’b+sab



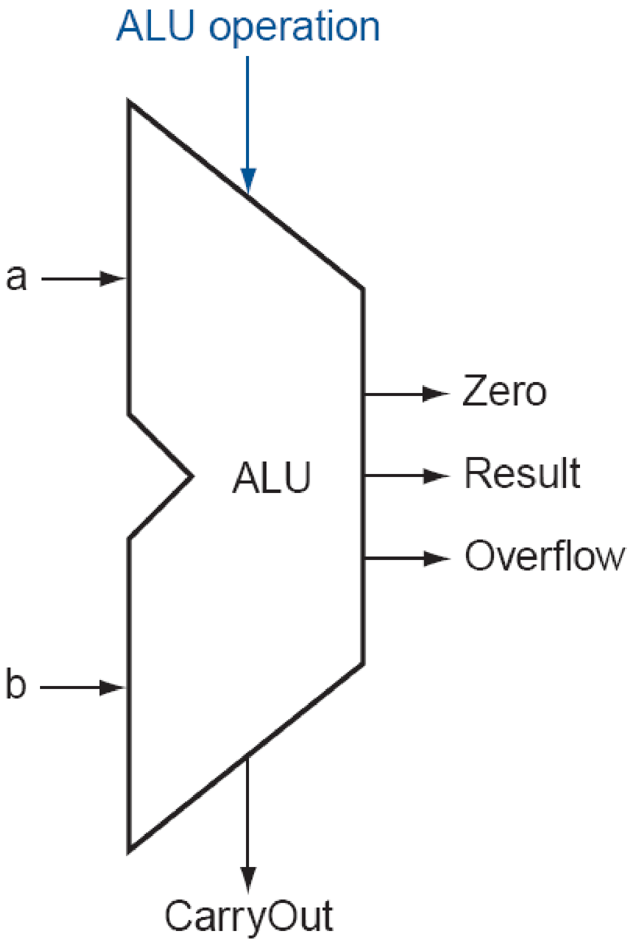
s \ ab	00	01	11	10
0			1	1
1		1	1	

- F=s’a+sb

Building from the adder to ALU

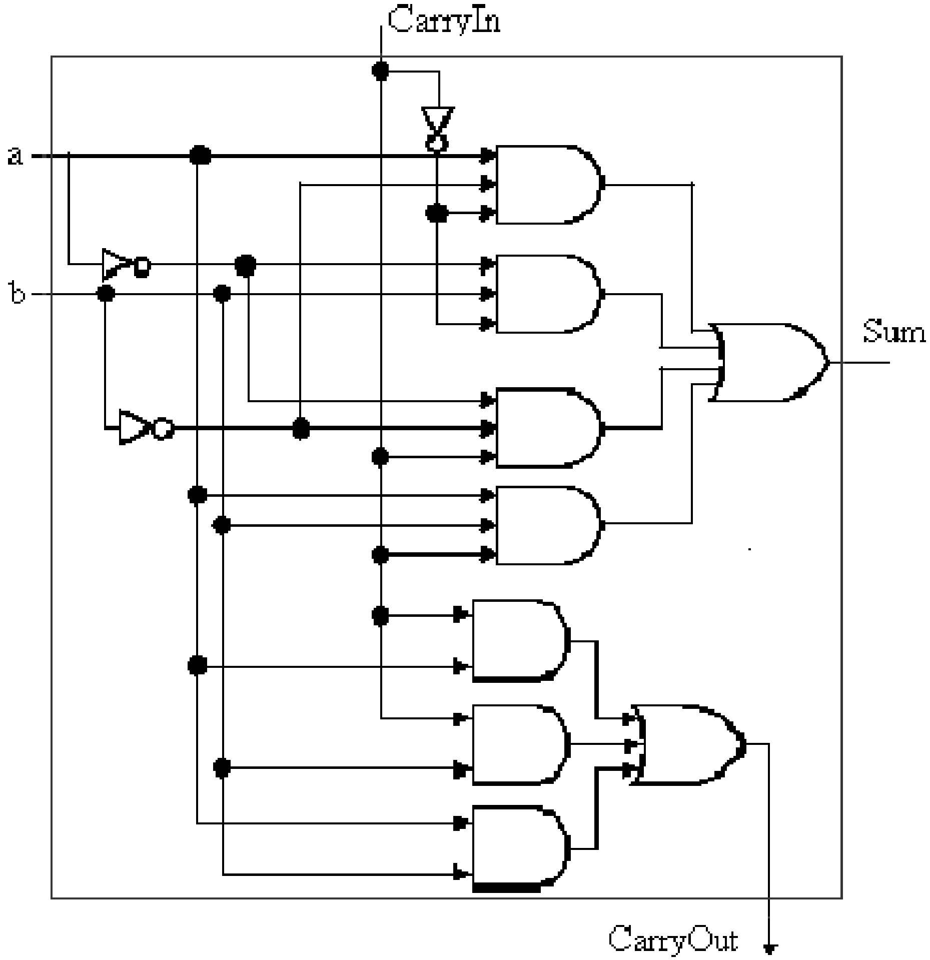
- ALU – Arithmetic Logic Unit, does the major calculations in the computer, including
  - Add
  - And
  - Or
  - Sub
  - …
- In MIPS, the ALU takes two 32-bit inputs and produces one 32-bit output, plus some additional signals
- Add is only one of the functions, and in this lecture, we are going to see how an full ALU is designed

ALU

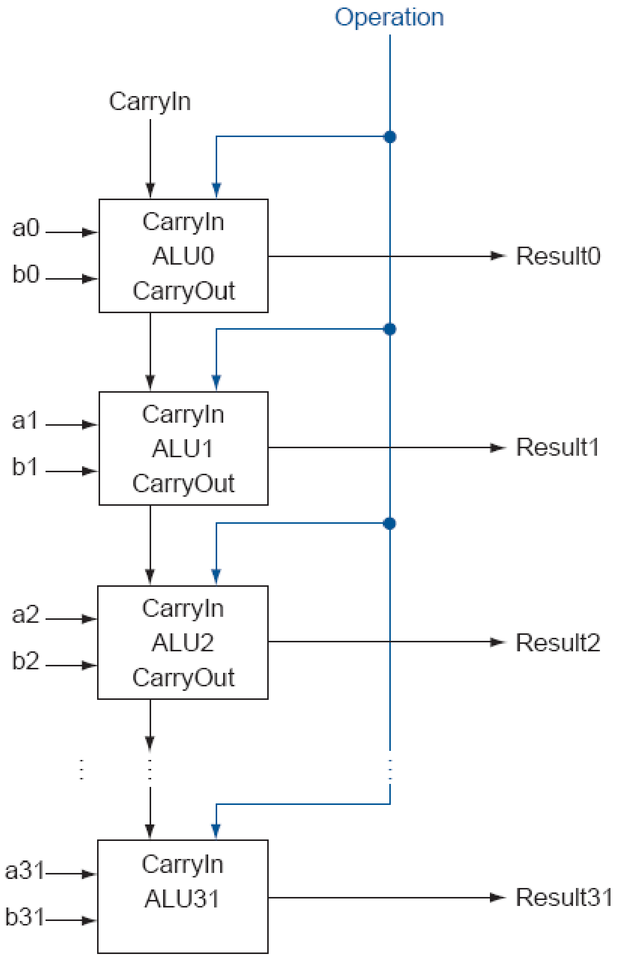


Review

- 1-bit full adder



32-bit adder

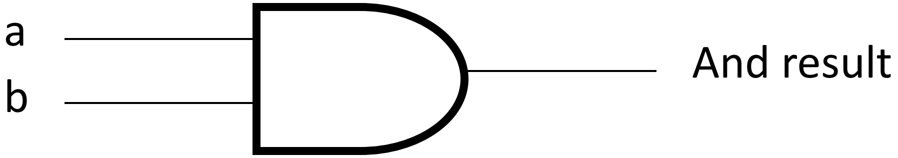

Operation
CarryIn
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a31
b31
CarryIn
ALU31
Result31

Building 32-bit ALU with 1-bit ALU

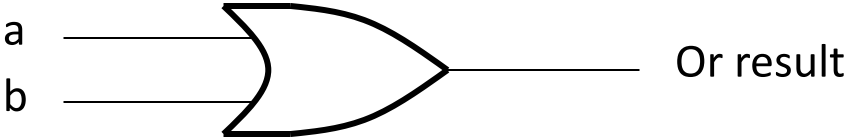
- Build 32-bit ALU with 1-bit ALU.
- Deal with the easy ones first – “and” and “or”

And and Or operations

- And



- Or

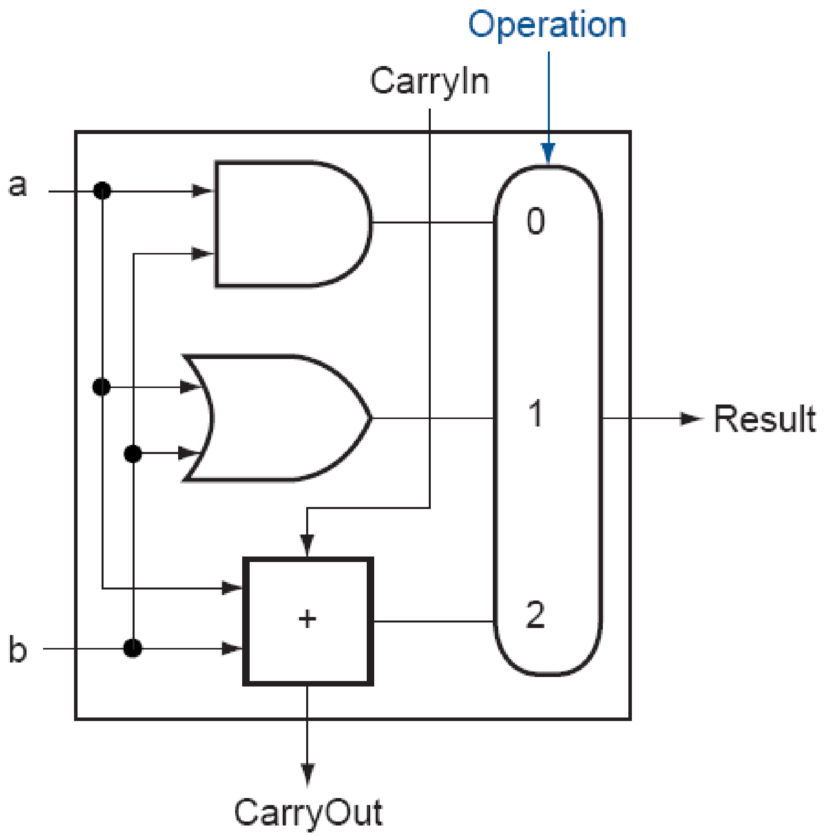


Putting them together

- Sometimes the instruction is add, sometimes it is or, sometimes is and, how to “put them together?”
- In MIPS instructions, there are many fields: op, funct, rs, rt, rd, shamt…

Putting them together

- Just do everything (add, and, or) and then select one **AS** the output with a selector.



Subtraction?

- How to implement subtraction?

Subtraction

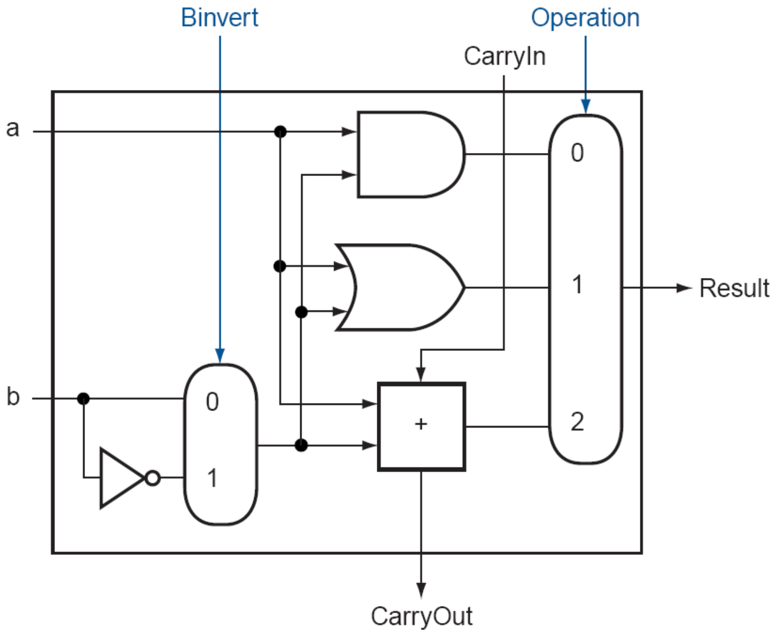
- Using two's complement representation, we can implement subtraction through addition

$$a + \bar{b} + 1 = a + (\bar{b} + 1) = a + (-b) = a - b$$

- The reason is that a negative number -b in 2's complement is actually 2^n-b. So if you do a+2^n-b and take only the lower n bits, it becomes a-b because 2^n is a one bit at bit n (bit indices are $0,1,2,\ldots,$ n-1, n).
- What do we need to add to the ALU we have in order to be able to perform subtraction?

1-Bit ALU That can Do Subtraction

- To do a-b, three things:
  1. Invert every bit of b.
  2. Add 1.
  3. Add with a.
- So, if it is a subtraction, invert the second operand, set the CarryIn of the last one-bit full adder to be 1, then select the adder output.



Subtraction

- Notice that every time we want the ALU to subtract, we set both CarryIn and Binvert to 1. For add or logical operations, we want both control lines to be 0. We can therefore simplify control of the ALU by combining the CarryIn and Binvert to a single control line called *Bnegate.*

Supporting Branch Instructions

- We need to be able to test if two numbers are the same

$$(a - b = 0) \Rightarrow a = b$$

$$\text{Zero} = (\overline{\text{Result31} + \text{Result30} + \ldots + \text{Result2} + \text{Result1} + \text{Result0}})$$

Supporting Set Less Than

- Set less than instruction produces 1 if rs < rt, and 0 otherwise
  - It needs to set all but the least significant bit to 0
  - The least significant bit is set according to the comparison
    - Which can be done using subtraction

$$\begin{aligned}(\mathrm{a}-\mathrm{b})<0 &\Rightarrow ((\mathrm{a}-\mathrm{b})+\mathrm{b})<(0+\mathrm{b}) \\ &\Rightarrow \mathrm{a}<\mathrm{b}\end{aligned}$$

    - That is, do a subtraction, check the sign bit (bit 31).

Complication

- If we only use the sign bit of the adder, sometimes we will be wrong
  - For the following example (using 4 bits only), we have

$$1001_{two} - 0110_{two}$$

$$= 1001_{two} + 1010_{two}$$

$$= 0011_{two}$$

  - Then we have $-7_{ten} \geq 6_{ten}$, which is clearly wrong

Overflow

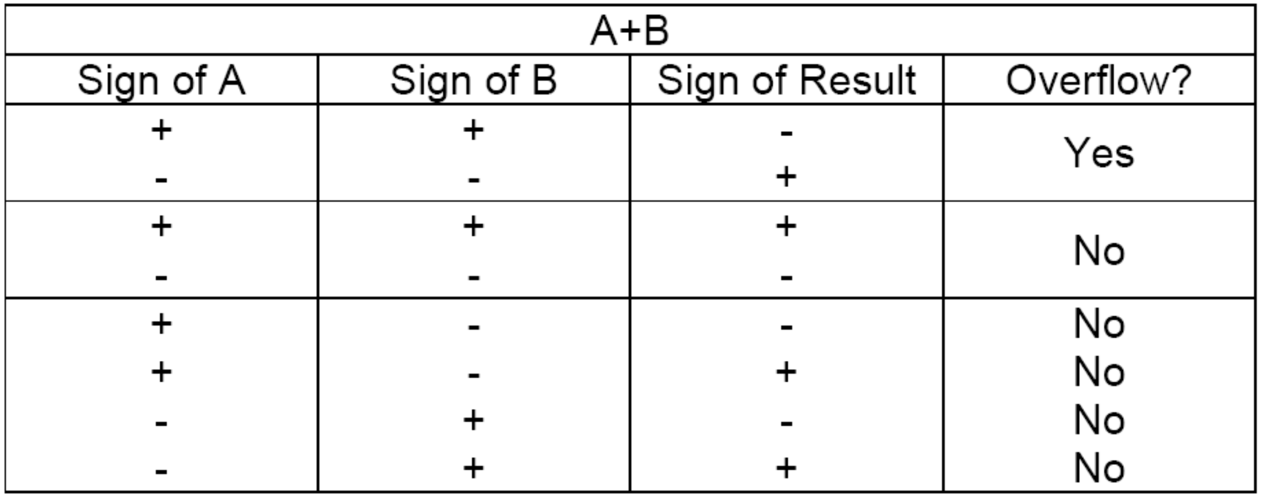
- The problem is that sometimes we have overflow.
  - If we have only 4 bits, a number greater than 7 or a number less than -8 will cause an overflow because it cannot be represented in 4 bits.
  - In the previous example, -7-6=-13, overflowed.

Dealing with overflow

- Overflow happens when the two numbers are of the same sign.
  - If they are of different signs, the addition result will be less than the larger one (the absolute value) and should be still within the range, assuming the two original numbers are within the range.

Overflow Detection

One way to detect overflow is to check whether the sign bit is consistent with the sign of the inputs when the two inputs are of the same sign – if you added two positive numbers and got a negative number, something is wrong, and vice versa.



A+B			
Sign of A	Sign of B	Sign of Result	Overflow?
+	+	-	Yes
-	-	+	
+	+	+	No
-	-	-	
+	-	-	No
+	-	+	No
-	+	-	No
-	+	+	No

Dealing with overflow

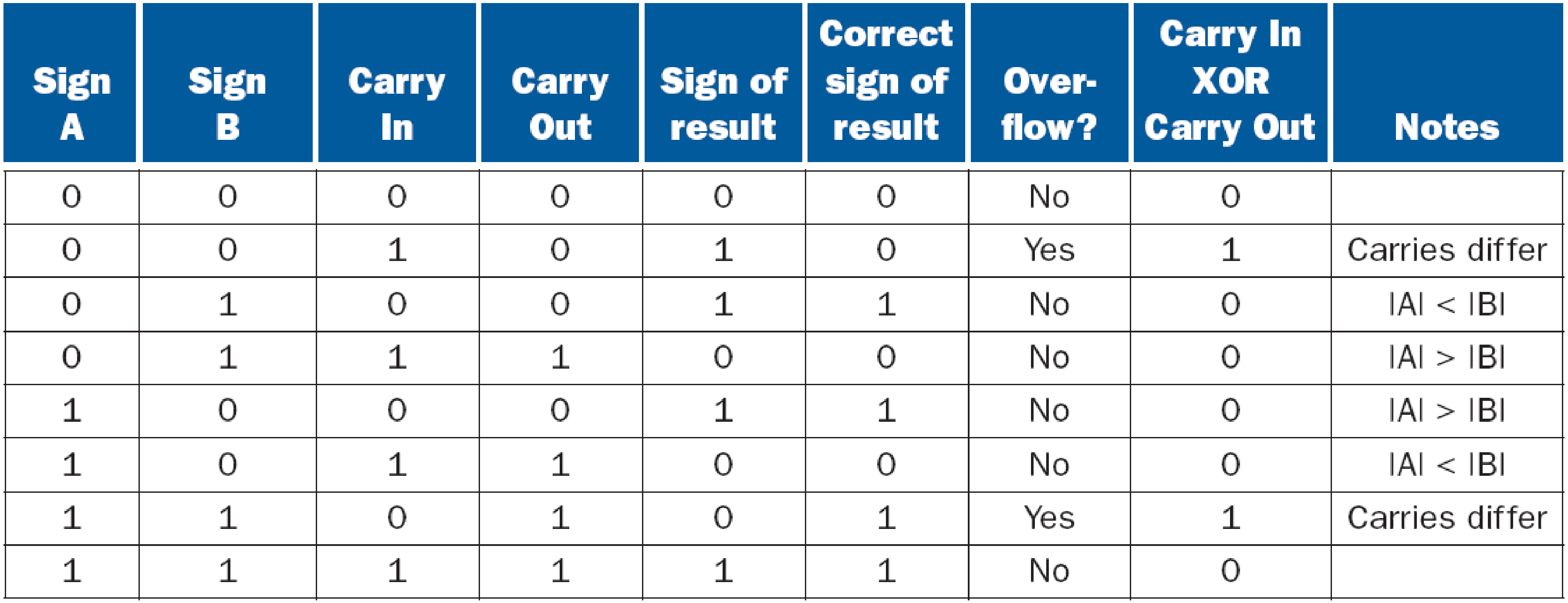
- For two positive numbers, after the addition,
  - The carryout of ALU31 must be 0, because in 2’s complement, positive numbers go from 000…1 to 011..1. The largest number is 011…1 and adding two 011…1 will lead to 111…10, the carry out is still 0.
  - if no overflow, the sign bit (bit 31) should be 0, because the result is a positive number.
  - If overflowed, the sign bit (bit 31) will be 1, caused by a carryin to ALU31.

Dealing with overflow

- For two negative numbers, after the addition,
  - The carryout of ALU31 must be 1, because in 2’s complement, negative numbers go from 100…0 to 111..1. Even if you are just adding two 100…0, you will have 1000…00, the carry out is 1.
  - if no overflow, the sign bit (bit 31) should be 1, because the result is a negative number.
  - If overflowed, the sign bit (bit 31) will be 0, caused by having no carryin to ALU31.

Overflow Detection

- So, we can detect the overflow by checking if the CarryIn and CarryOut of the most significant bit are different



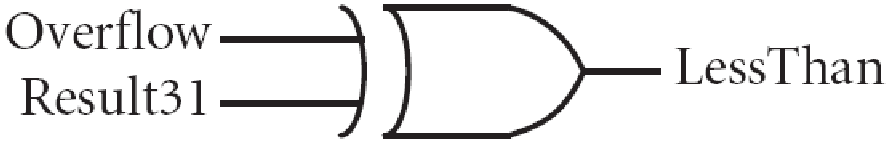
Sign A	Sign B	Carry In	Carry Out	Sign of result	Correct sign of result	Over-flow?	Carry In XOR Carry Out	Notes
0	0	0	0	0	0	No	0	
0	0	1	0	1	0	Yes	1	Carries differ
0	1	0	0	1	1	No	0	|A| < |B|
0	1	1	1	0	0	No	0	|A| > |B|
1	0	0	0	1	1	No	0	|A| > |B|
1	0	1	1	0	0	No	0	|A| < |B|
1	1	0	1	0	1	Yes	1	Carries differ
1	1	1	1	1	1	No	0	

Overflow

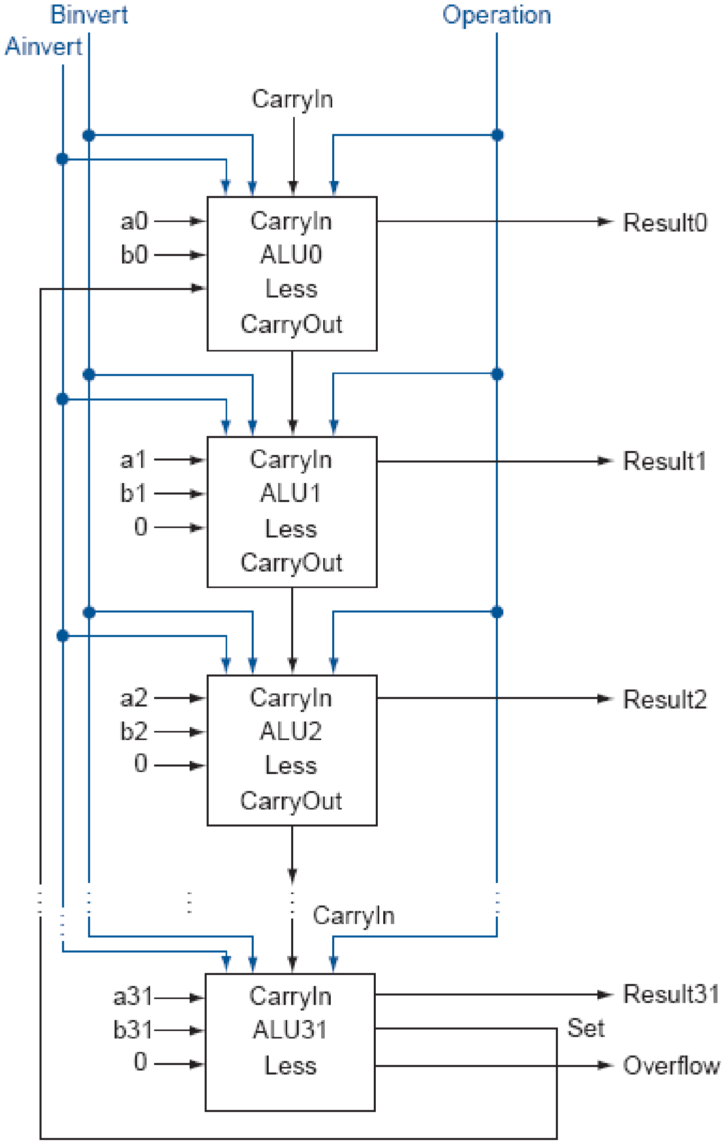
- The sign bit is correct if there is no overflow
- If there is overflow, the sign bit will be wrong and needs to be inverted

Overflow	Result31	LessThan
0	0	0
0	1	1
1	0	1
1	1	0

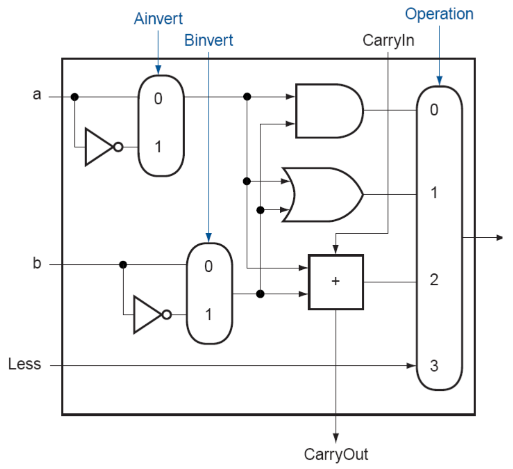
$$LessThan = Overflow \oplus Result31$$

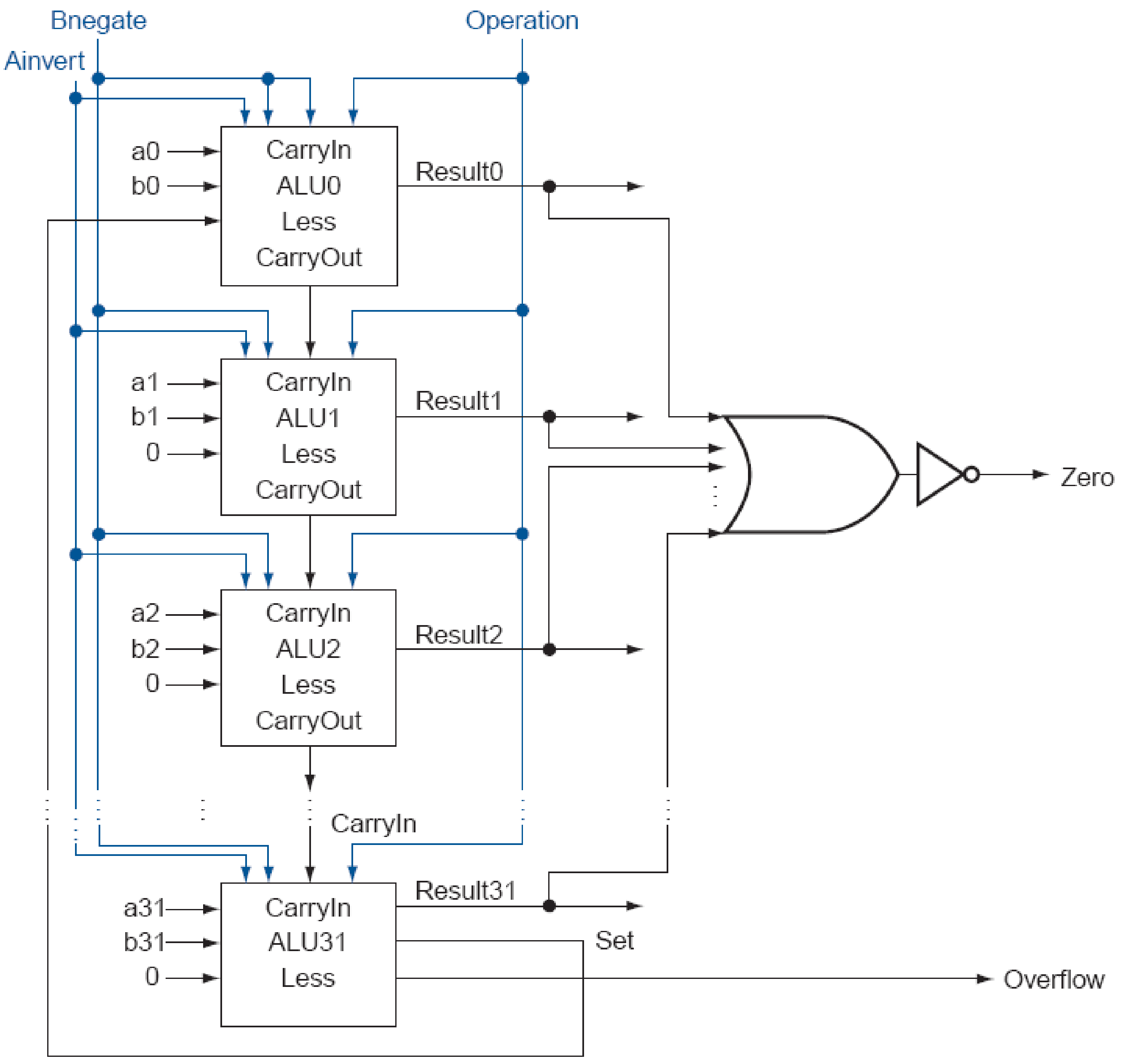


32-bit ALU that Supports Set Less Than



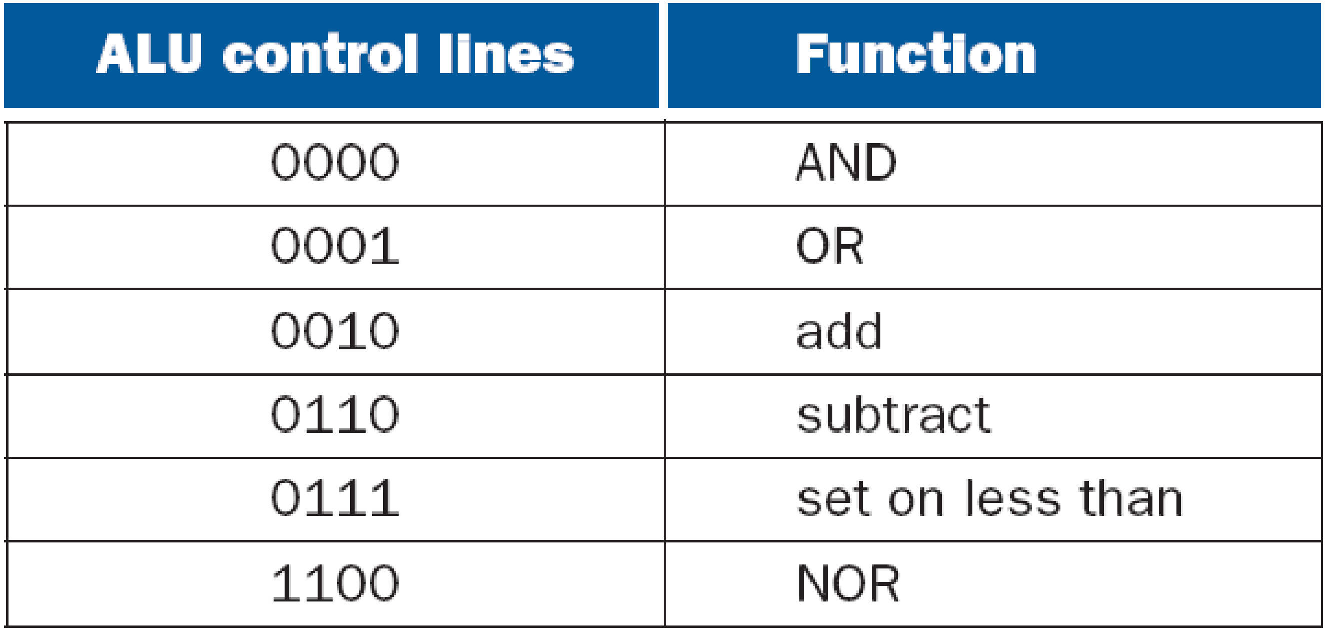
Final 32-Bit ALU





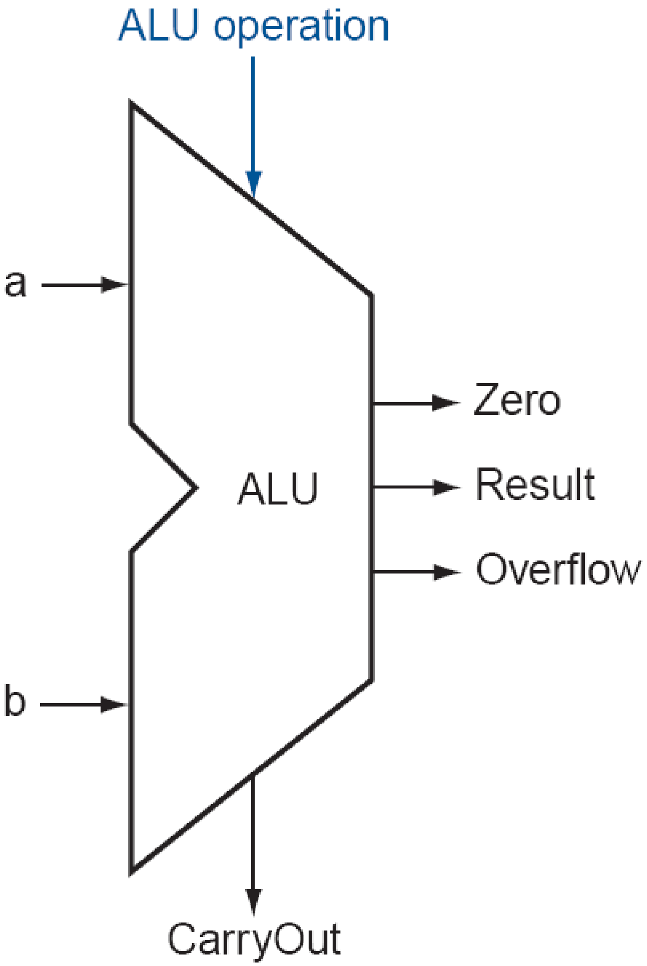
Final 32-Bit ALU

- ALU control lines are 1-bit Ainvert line, 1-bit Bnegate line, and 2-bit operation lines



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

ALU Symbol


ALU operation
a
b
ALU
Zero
Result
Overflow
CarryOut