Building an ALU (Part 2):

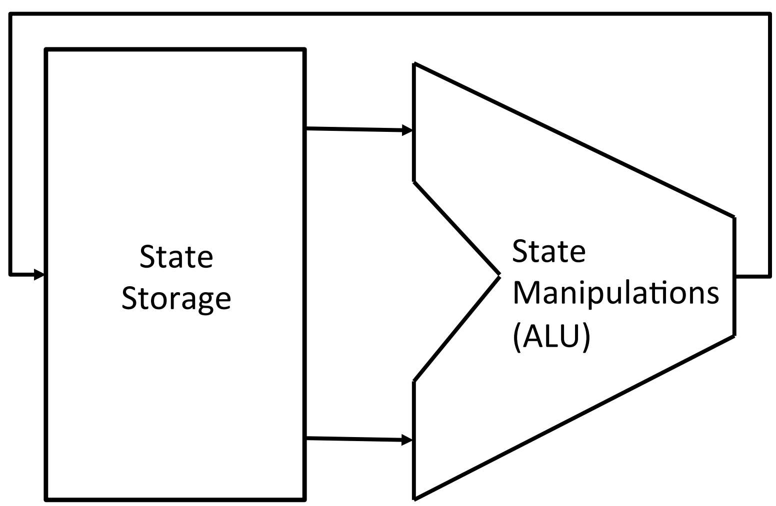
PICK UP HANDOUT.

SIGN UP FOR CBTF EXAM 1

State – the central concept of computing

Computer can do 2 things

1) Store state
2) Manipulate state (Combine arithmetic and logical operations into one unit)

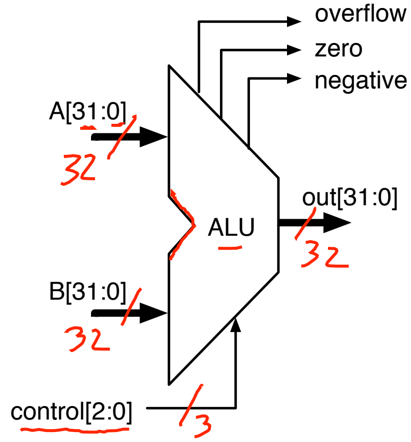


Today's lecture

- We'll finish the 32-bit ALU today!
  - 32-bit ALU specification

- Complete 1-bit ALU
- Assembling them to make 32-bit ALU
- Handling flags:
  - zero, negative, overflow

A specification for a 32-bit ALU



010
011
100
101
110
111

control	out=
0	undefined
1	undefined
2	A + B
3	A – B
4	A AND B
5	A OR B
6	A NOR B
7	A XOR B

```
module alu32(out, overflow, zero, negative, A, B, control);
   output[31:0] out;
   output       overflow, zero, negative;
   input [31:0] A, B;
   input  [2:0] control;
```

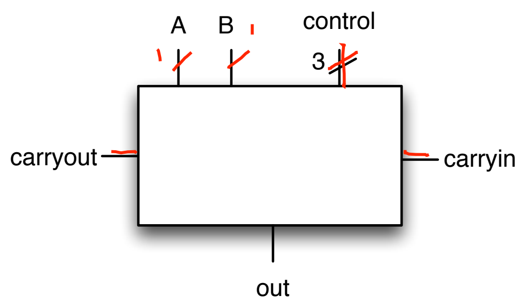
Did overflow occur?

Is the output equal to zero?

Is the output negative?

Use a modular 1-bit ALU to build 32-bit ALU

- Previously we showed 1-bit adder/subtractor, 1-bit logic unit
  - Time to put them together.

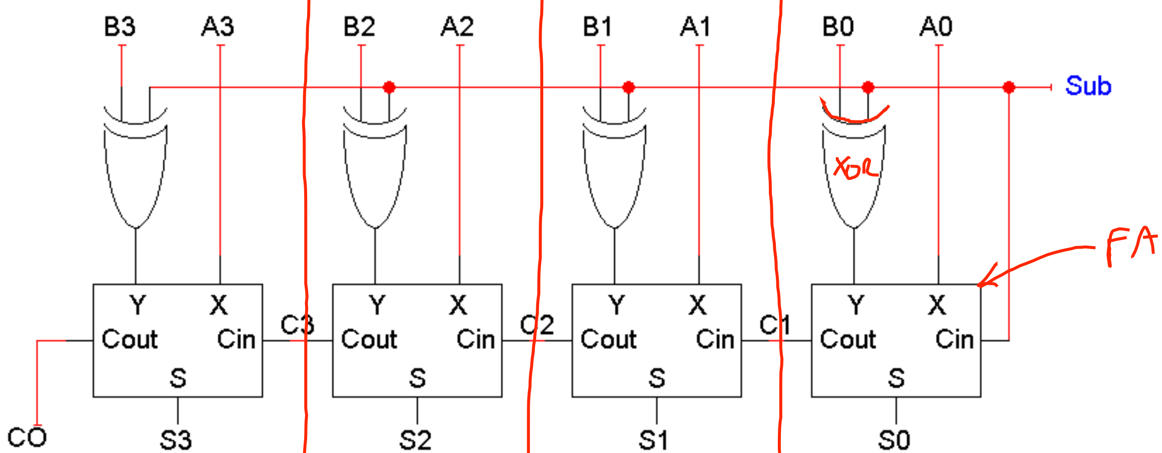


control	out_i=
0	undefined
1	undefined
2	$A_i + B_i$
3	$A_i - B_i$
4	A_i AND B_i
5	A_i OR B_i
6	A_i NOR B_i
7	A_i XOR B_i

```
module alu1(out, carryout, A, B, carryin, control);
   output       out, carryout;
   input        A, B, carryin;
   input [2:0] control;
```

Addition + Subtraction in one circuit (1-bit Arithmetic Unit)

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A – B.



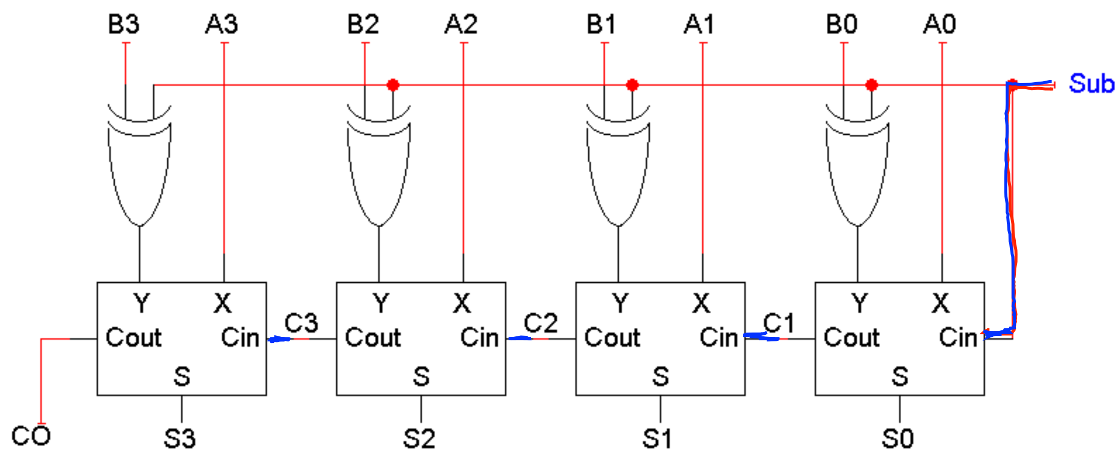
- Which parts belong in inside the 1-bit ALU?

A) the Full Adder, B) the XOR gate, C) Both, D) Neither

iclicker.

Addition + Subtraction in one circuit (1-bit Arithmetic Unit)

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A – B.

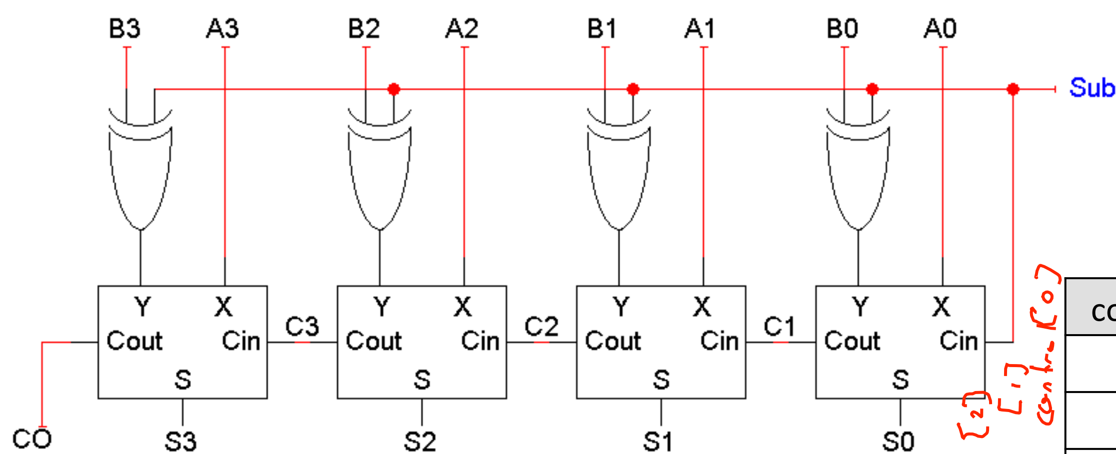


- What should we do with the full adder's `Cin` input?

A) Connect to Sub, B) Connect to 1-bit ALU's carryin

Addition + Subtraction in one circuit (1-bit Arithmetic Unit)

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A – B.

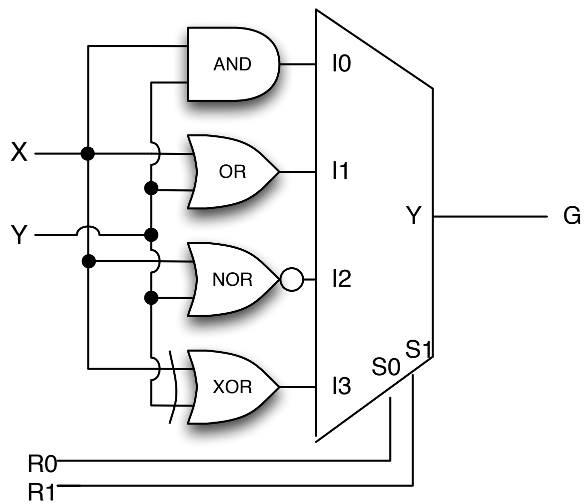


[2] [1] control[0]
0 1 0
0 1 1

control	out=
0	undefined
1	undefined
2	add A + B
3	sub A – B

- Where will the “Sub” signal come from?

Complete 1-bit Logic Unit

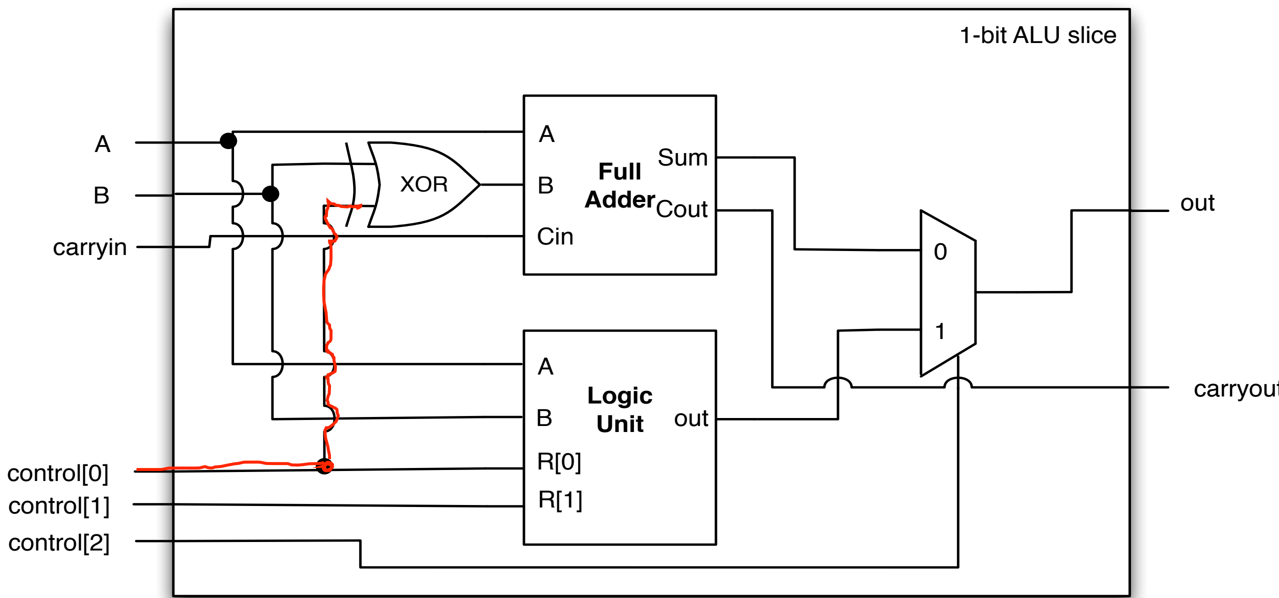


R_1	R_0	Output	
0	0	$G_i = X_iY_i$	AND
0	1	$G_i = X_i+Y_i$	OR
1	0	$G_i = (X_i+Y_i)'$	NOR
1	1	$G_i = X_iY_i$	XOR

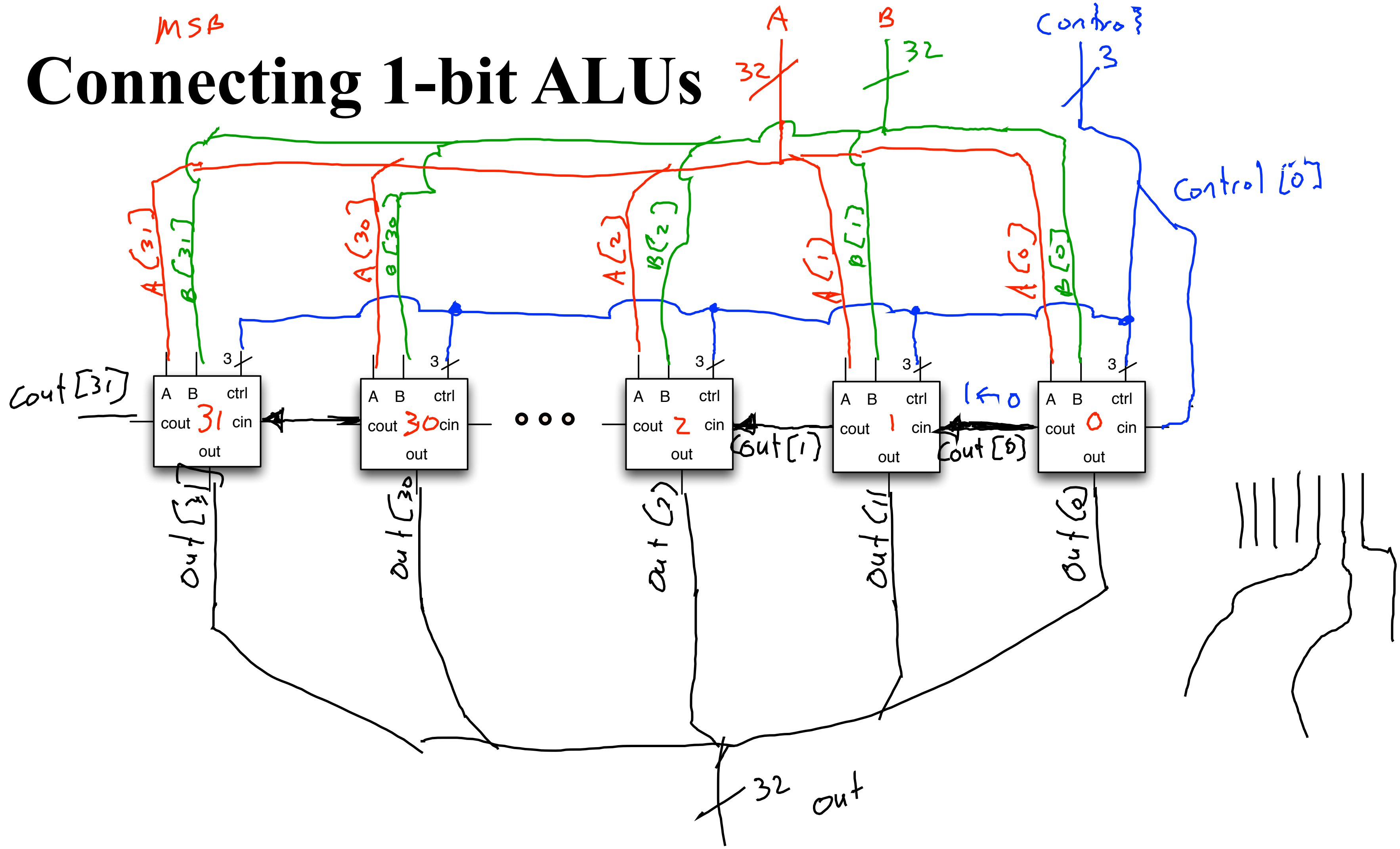
- What should the control inputs (R0, R1) connect to?

- How do we select between the adder and the logic unit?
- How do we control the selection?

Complete 1-bit ALU



Connecting 1-bit ALUs



Flags (overflow, zero, negative)

- Let’s do negative first; negative evaluates to:
  - 1 when the output is negative, and
  - 0 when the output is positive or zero

- Negative =

```
a)control[0]
b)carryout[32]
c)output[32]
d)carryout[31]
e)output[31]
```

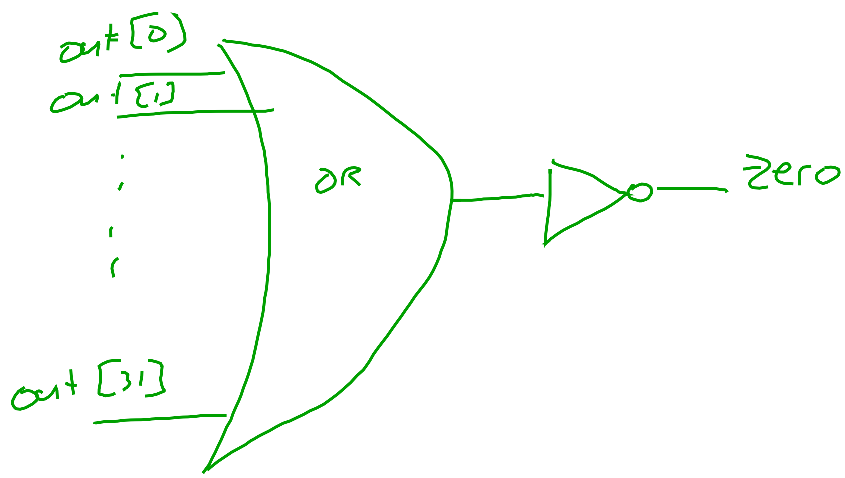
iclicker

Flags (overflow, zero, negative)

- zero evaluates to:
  - 1 when the output is equal to zero, else 0

nor (output, input1, input2, input3, input4, ...);

- Zero =



Flags (overflow, zero, negative)

- Overflow (for 2's complement) evaluates to:
  - 1 when the overflow occurred, else 0
    - adding two positive numbers yields a negative number
    - adding two negative numbers yields a positive number
- Consider the adder for the MSB:

X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

```
a)cin[31]  NOR cout[31]
b)cin[31]  AND cout[31]
c)cin[31]   OR cout[31]
d)cin[31]  XOR cout[31]
e)cin[31] NAND cout[31]
```

iclicker.

- Overflow =

Overflow examples

	1	1	0	1	(-3)
+	1	1	0	0	+ (-4)

	1	0	1	1	(-5)
+	1	1	0	0	+ (-4)

	0	1	0	0	4
+	0	1	0	0	4

	0	1	0	0	4
+	1	1	0	0	+ (-4)