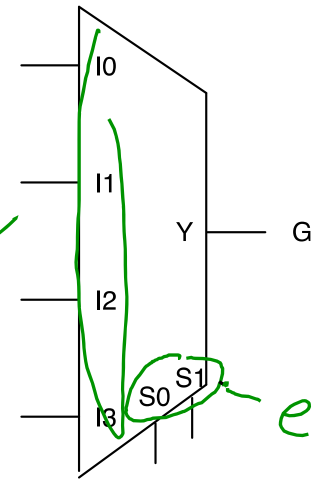
Mux Hierarchical Design (more inputs)

- How do we build a mux with 4 inputs?

2^1	2^0		
S1	S0		
0	0	→	I0
0	1	→	I1
1	0	→	I2
1	1	→	I3

decimal

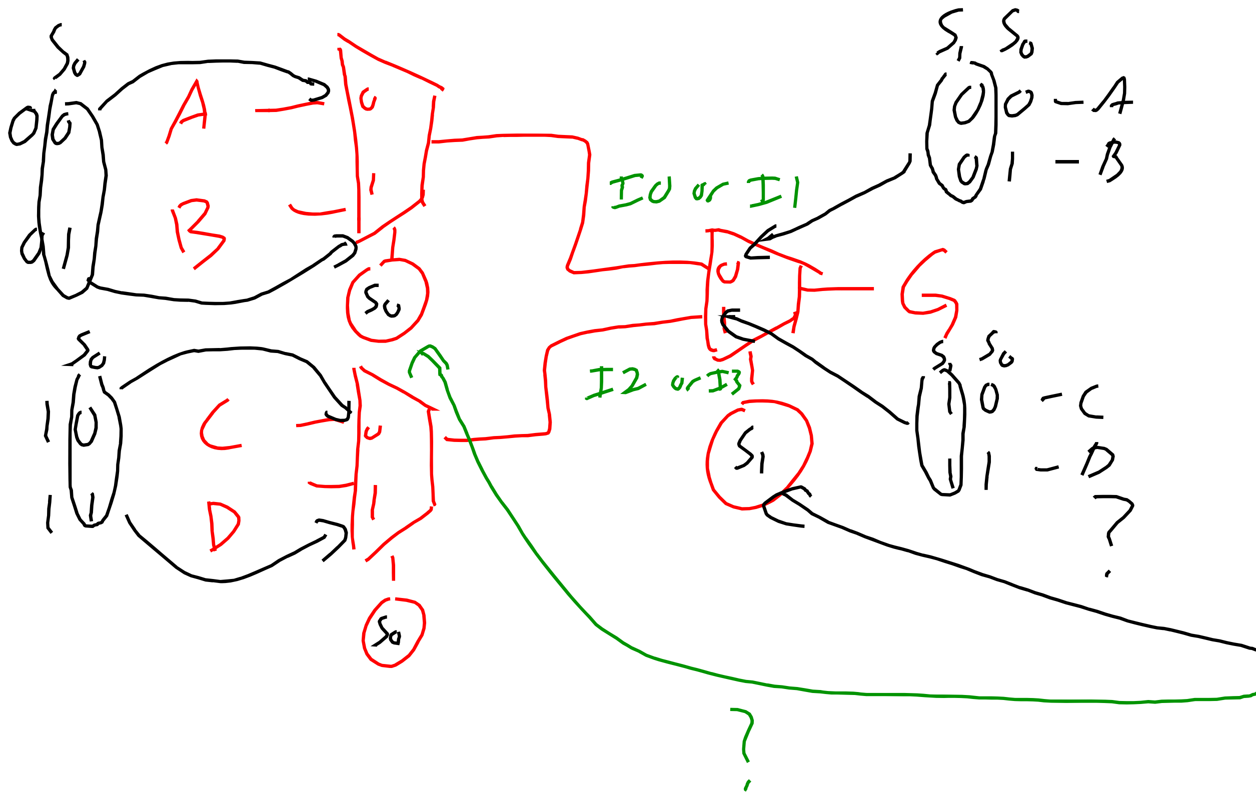


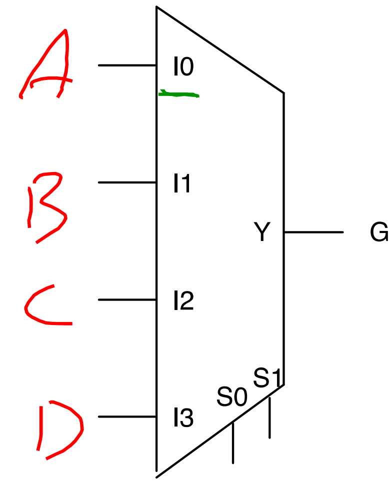
exponent

A:	S_0
B:	S_1
C:	Either

Mux Hierarchical Design (more inputs)

- How do we build a mux with 4 inputs?





A: S_0

B: S_1

C: Either

Building an ALU (Part 2):

Happy Monday!

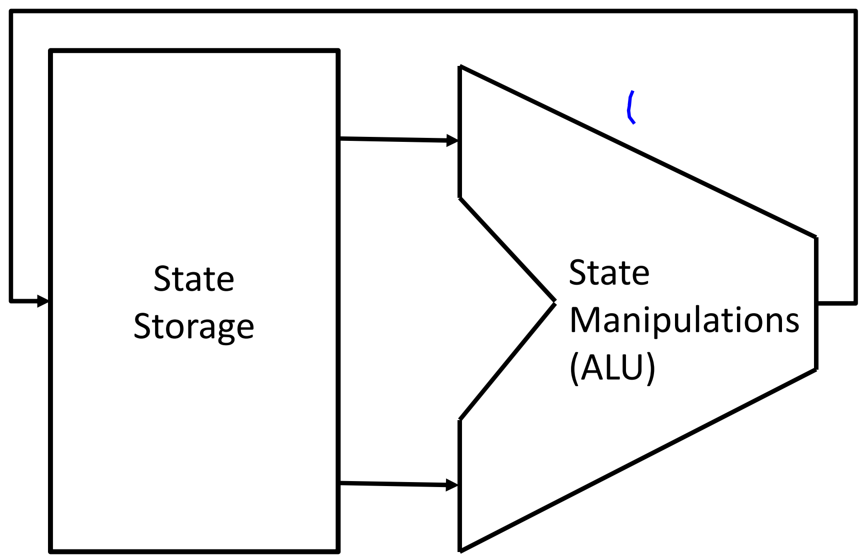
Exam 1 covers through today's lecture + delay

2 parts: short answer + verilog

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)

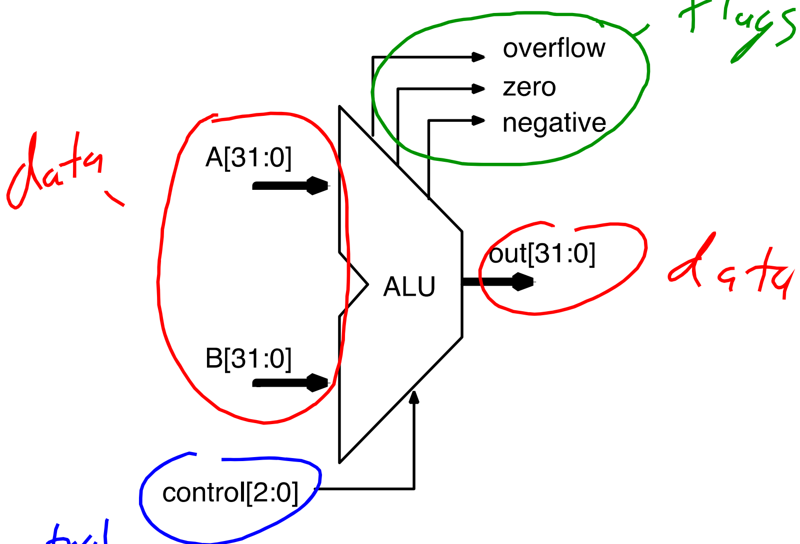


Today's lecture

- 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



control[2]	control	out=
000	0	undefined
001	1	undefined
010	2	A + B
011	3	A – B
100	4	A AND B
101	5	A OR B
110	6	A NOR B
111	7	A XOR B

arith ↑

↓ logical

Control

```
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;
```

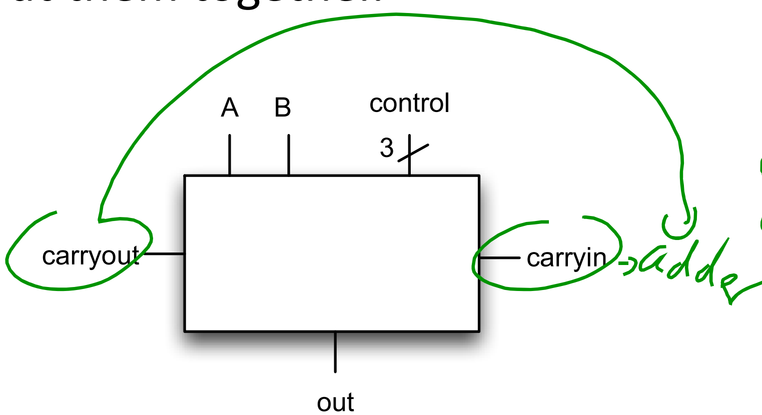
Did overflow occur?

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.



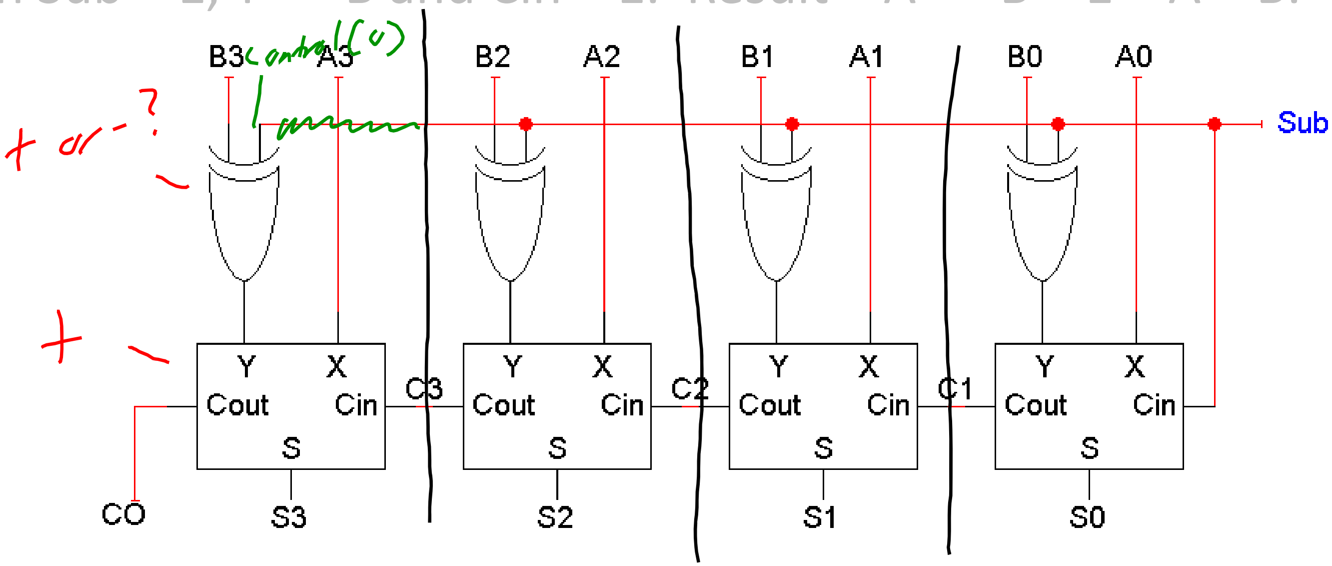
control	out_i=
0	undefined
1	undefined
010 2	$A_i + B_i$
011 3	$A_i - B_i$
4 100	A_i AND B_i
5 101	A_i OR B_i
6 110	A_i NOR B_i
7 111	A_i XOR B_i

control[0]
[1][0]

```
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.

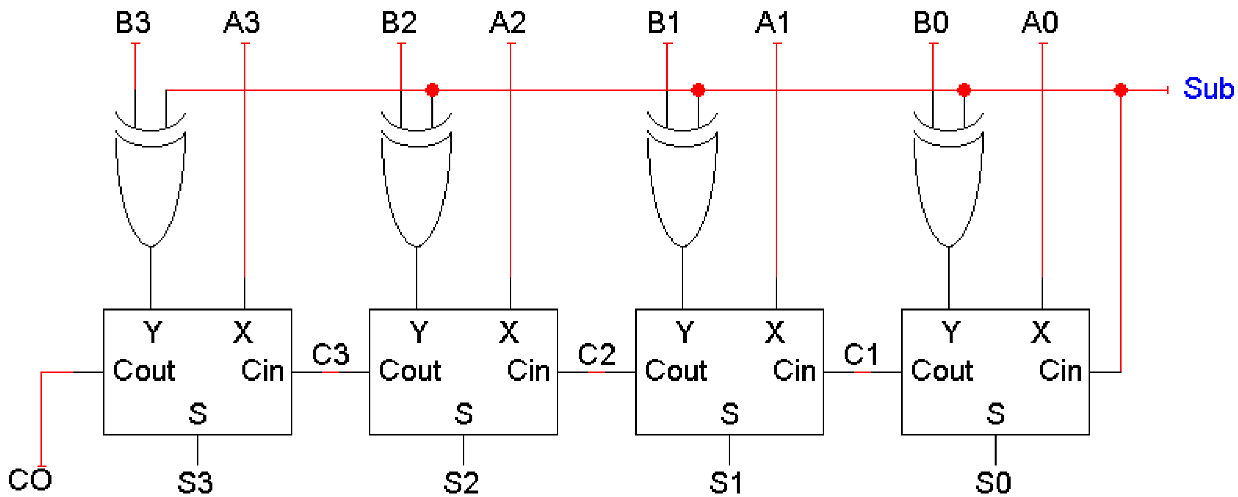


- Which parts belong in inside the 1-bit ALU?

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

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.



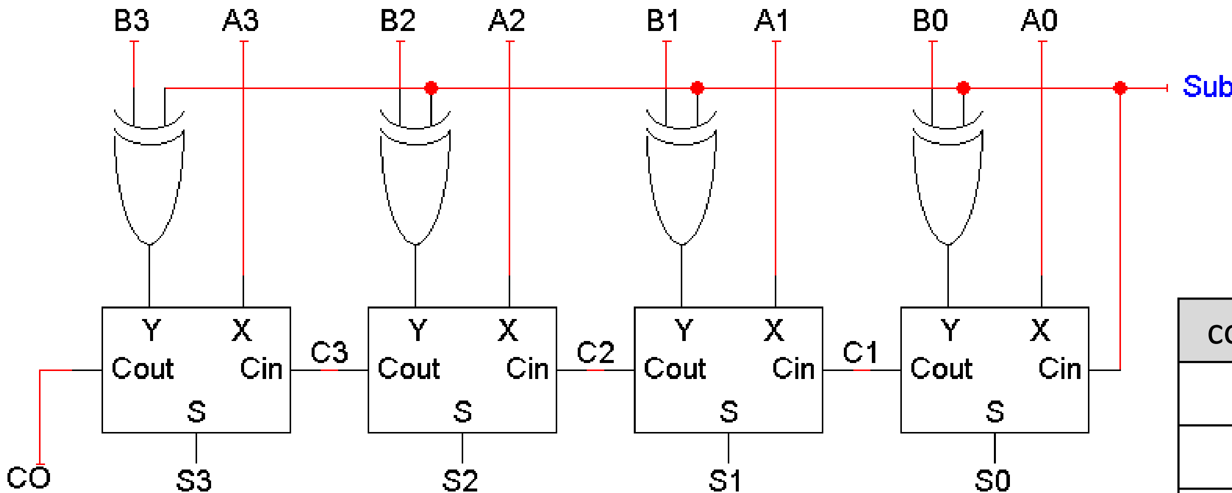
- What should we do with the full adder's `Cin` input?

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

iclicker.

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

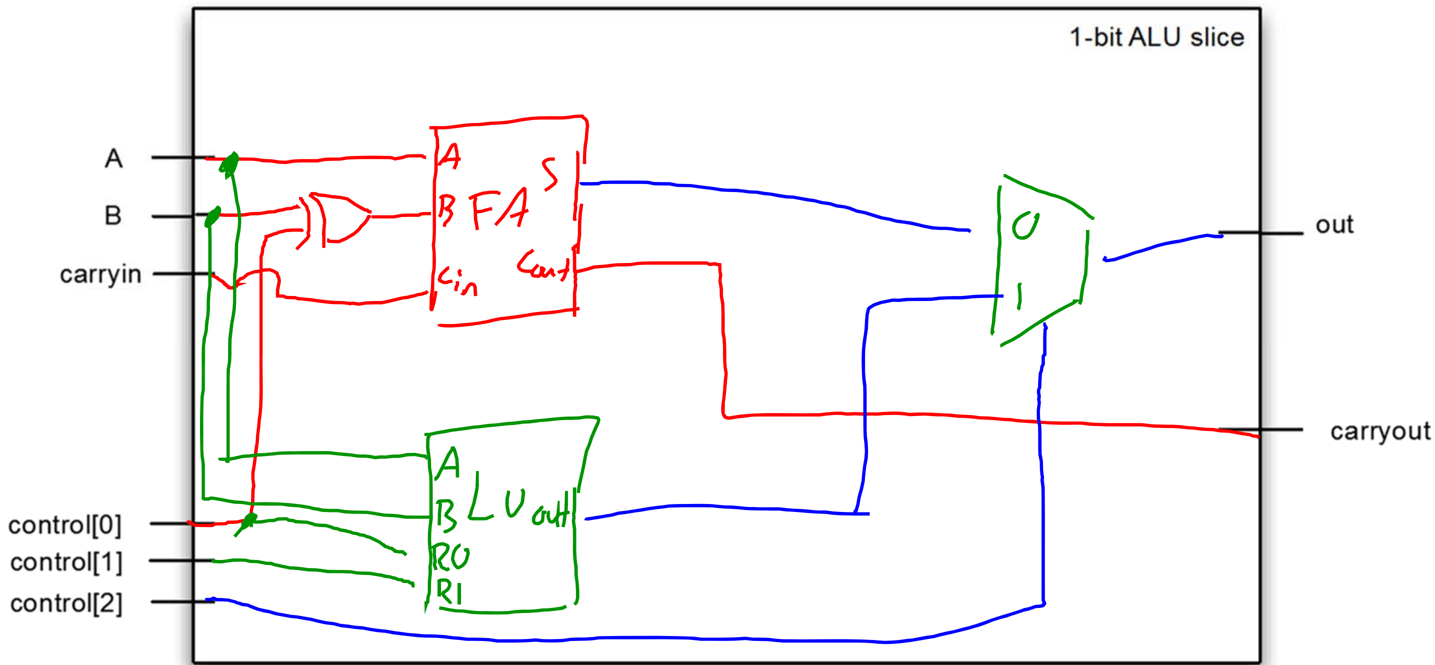
- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A – B.



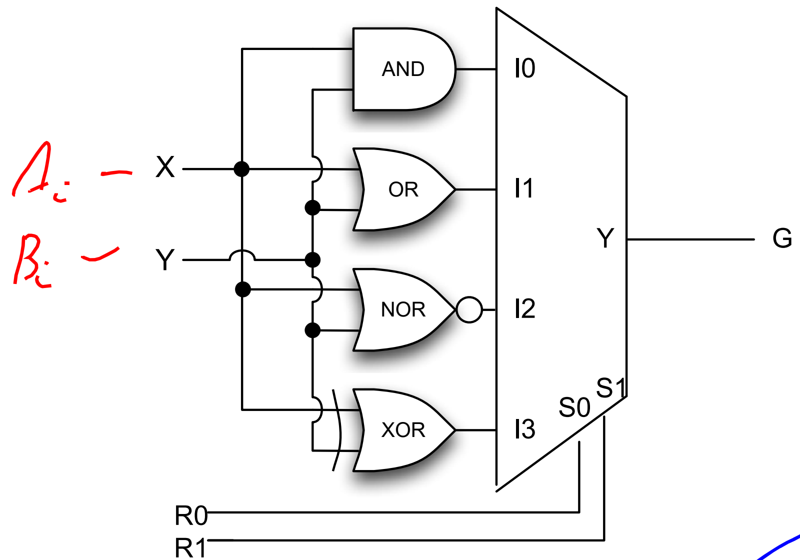
control	out=
0	undefined
1	undefined
2	A + B
3	A – B

- Where will the "Sub" signal come from?

Complete 1-bit ALU



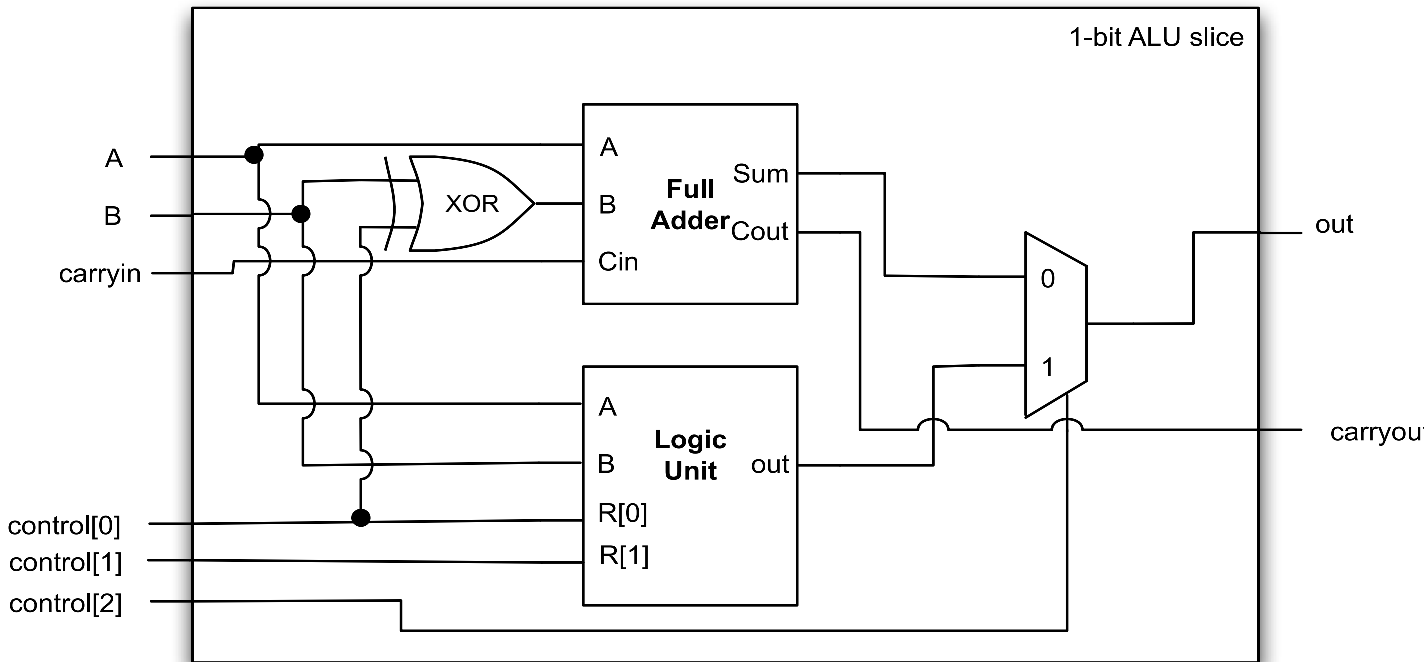
Complete 1-bit Logic Unit



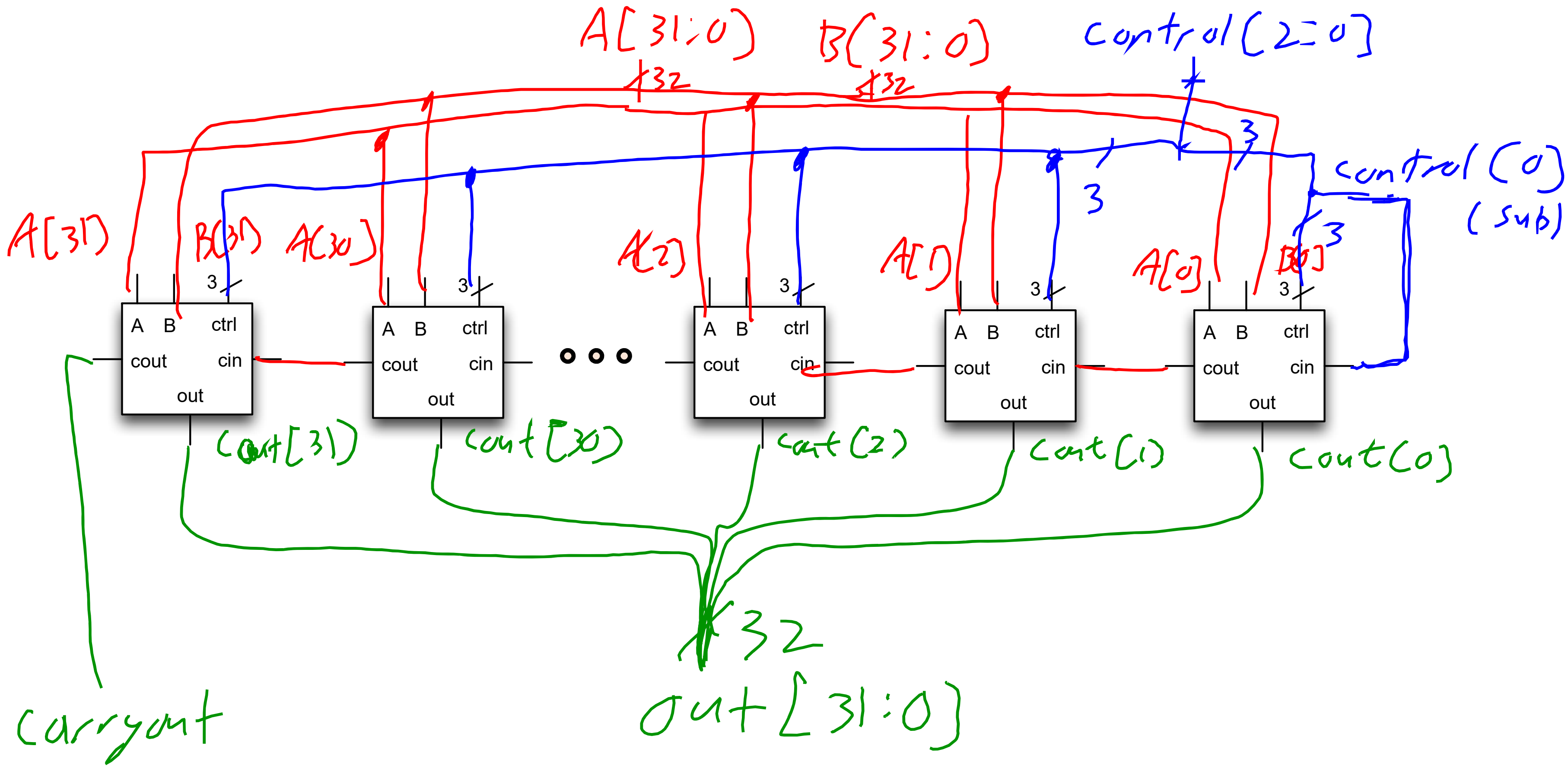
R_1	R_0	Output
0	0	$G_i = X_iY_i$
0	1	$G_i = X_i+Y_i$
1	0	$G_i = (X_i+Y_i)'$
1	1	$G_i = X_i \oplus Y_i$

- What should the control inputs (R0, R1) connect to? Control[1:0]
- How do we select between the adder and the logic unit? Control(2)
- How do we control the selection?

Complete 1-bit ALU



Connecting 1-bit ALUs → 32-bit ALU



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) `carryout[30]`
  b) `output[30]`
  c) `carryout[31]`
  d) `output[31]`
  e) `control[0]`

0 0 1 1

  0011 3
+ 1001 (-7)

?
  1100 (-4)

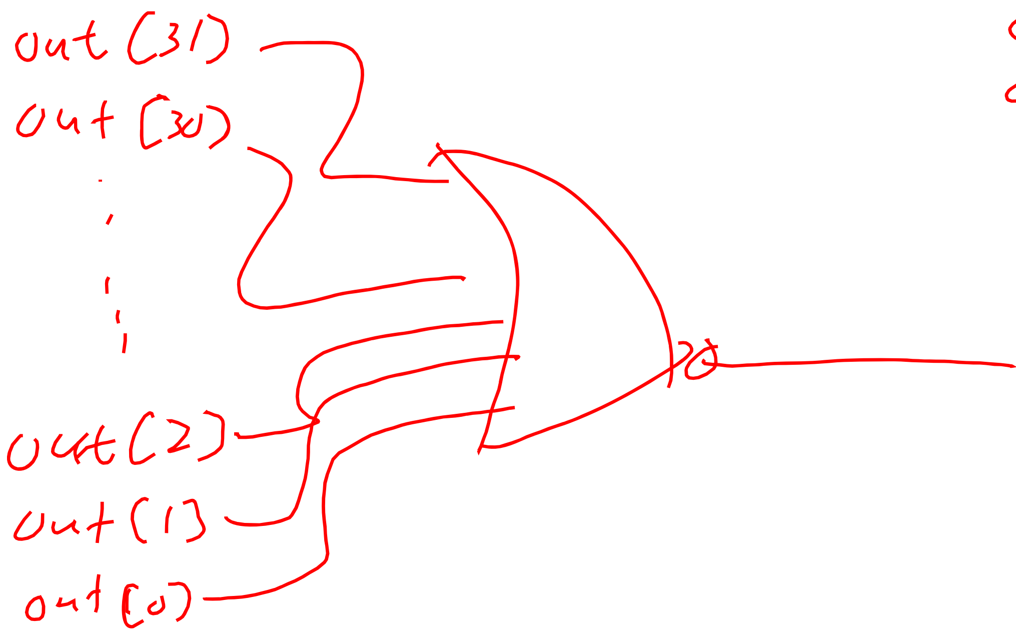
Flags (overflow, zero, negative)

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

x	y	zero?
0	0	1
0	1	0
1	0	0
1	1	0

NOR

- 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

C_{in}	C_{out}	overflow?
0	0	0
0	1	1
1	0	1
1	1	0

```
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 =