CSE 390B, Spring 2022

Building Academic Success Through Bottom-Up Computing

Growth Mindset & Arithmetic Logic Unit

Growth vs. Fixed Mindset, Negative Numbers in Binary, Arithmetic Logic Unit (ALU), Project 3 Overview

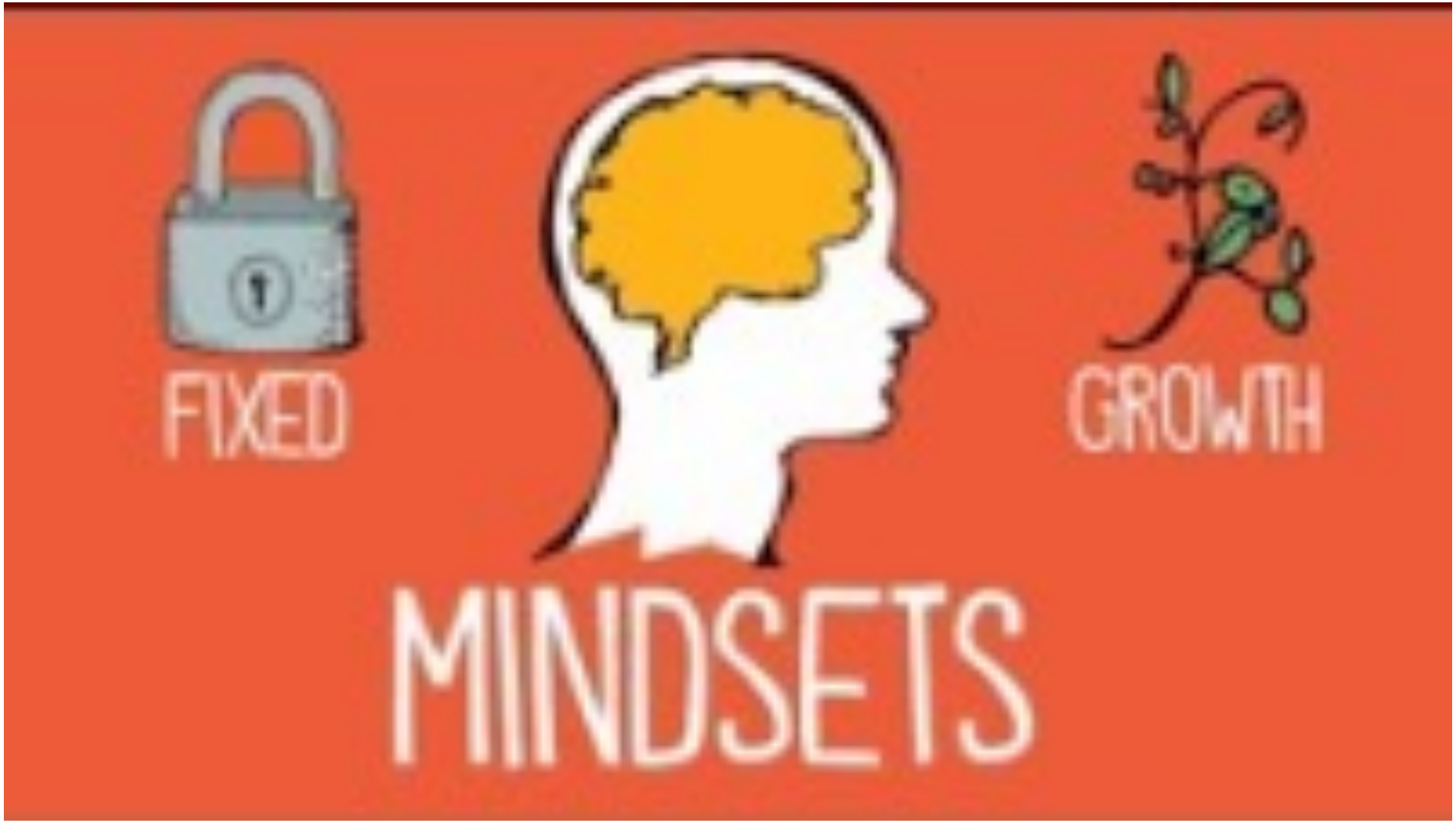
Project 2 Check-in

- How has Project 2 been going?
  - 👍 If you're feeling **great** about completing Project 2 by tonight
  - 👊 If you're feeling **decent** about where you are right now
  - 👎 If you're feeling **behind** or unlikely to submit on time
- A few common questions about Project 2:
  - Can't see message at bottom of Hardware Simulator? Try reducing window size (Settings → Displays → Change size to < 150%)
  - Can't use typical Boolean Function Synthesis on **`Or.hdl`**
  - **`DMux.hdl`**: Follow same implementation strategy as other gates
- **Double-check** your submission on GitLab
  - Navigate to GitLab, open tags, and verify that the associated commit includes your expected changes

Lecture Outline

- **Growth vs. Fixed Mindset**
  - **Setting SMART Goals**
- Negative Numbers in Binary
  - Unsigned, Signed, and Two's Complement Representations
- Arithmetic Logic Unit (ALU)
  - Overview and Examples of ALU Functions
- Project 3 Overview
  - ALU Implementation Strategy
  - HDL Tips and Tricks

Growth vs. Fixed Mindset



Setting SMART Goals

- **S** – Be specific, simple and significant.
- **M** – Make sure your goals are measurable. How many times within a week, month, the quarter do you want to do x goal?
- **A** – Make sure your goals are achievable. Is your goal within your scope of control?
- **R** – Be realistic and reasonable.
- **T** – Be time-bound. When will you accomplish your goal?

SMART Goals Group Discussion

SPRING QUARTER GOALS	SPHERE OF CONTROL	SMART GOAL FRAMEWORK
What are skills, practices or habits that are not strengths YET?	Getting a 4.0 in a course VS. Attending course office hours	**S** — Specific **M** — Measurable **A** — Achievable **R** — Realistic **T** — Timebound Attending CSE 390B office hours at least 5x this quarter (or once every other week)

Lecture Outline

- Growth vs. Fixed Mindset
  - Setting SMART Goals
- **Negative Numbers in Binary**
  - **Unsigned, Signed, and Two's Complement Representations**
- Arithmetic Logic Unit (ALU)
  - Overview and Examples of ALU Functions
- Project 3 Overview
  - ALU Implementation Strategy
  - HDL Tips and Tricks

Poll Everywhere

Vote at https://pollev.com/cse390b

What are the Sign and Magnitude (signed) and Two's Complement binary representations of the number -7?

A. Signed: 0b1001, Two's Complement: 0b1111
B. Signed: 0b0111, Two's Complement: 0b1001
C. Signed: 0b1111, Two's Complement: 0b0111
D. Signed: 0b1111, Two's Complement: 0b1001
E. We're lost…

Two's Complement

- One binary interpretation to represent negative values
- Most significant bit (MSB) has a negative weight
  - Add the remaining bits as usual (with positive weights)
- Example: $0b1101$ in Two's Complement
  - $-(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = -8 + 4 + 0 + 1$
    $= -3$
- Negation procedure: Take bitwise complement and add one
  - $\mathbf{-x = \sim x + 1}$
  - Example: Negate $x = 4$

Two's Complement

- One binary interpretation to represent negative values
- Most significant bit (MSB) has a negative weight
  - Add the remaining bits as usual (with positive weights)
- Example: $0b1101$ in Two's Complement
  - $-(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = -8 + 4 + 0 + 1$
    $= -3$
- Negation procedure: Take bitwise complement and add one
  - $\mathbf{-x = \sim x + 1}$
  - Example: Negate $x = 4$
  - $-4 = \sim 0b0100 + 1 = 0b1011 + 0b1 = 0b1100 = -8 + 4 = -4$

Four-bit Values in Various Representations

Binary Value	Unsigned Binary	Signed Binary	Two's Complement
0b0000	0	0	0
0b0001	1	1	1
0b0010	2	2	2
0b0011	3	3	3
0b0100	4	4	4
0b0101	5	5	5
0b0110	6	6	6
0b0111	7	7	7
0b1000	8	-0	-8
0b1001	9	-1	-7
0b1010	10	-2	-6
0b1011	11	-3	-5
0b1100	12	-4	-4
0b1101	13	-5	-3
0b1110	14	-6	-2
0b1111	15	-7	-1

Two's Complement Addition

- The process for adding binary in Two's Complement is the same as that of unsigned binary

- Hardware performs the exact same calculations
  - It doesn't need to know the sign of the values, it performs the same calculation
  - The only difference is representation of sum

carry				
a	1	0	0	1
b	0	0	1	0
sum				

- Example: $0b1001 + 0b0010$
  - Unsigned interpretation:
  - Signed interpretation:
  - Two's Complement interpretation:

Two's Complement Addition

- The process for adding binary in Two's Complement is the same as that of unsigned binary

- Hardware performs the exact same calculations
  - It doesn't need to know the sign of the values, it performs the same calculation
  - The only difference is representation of sum

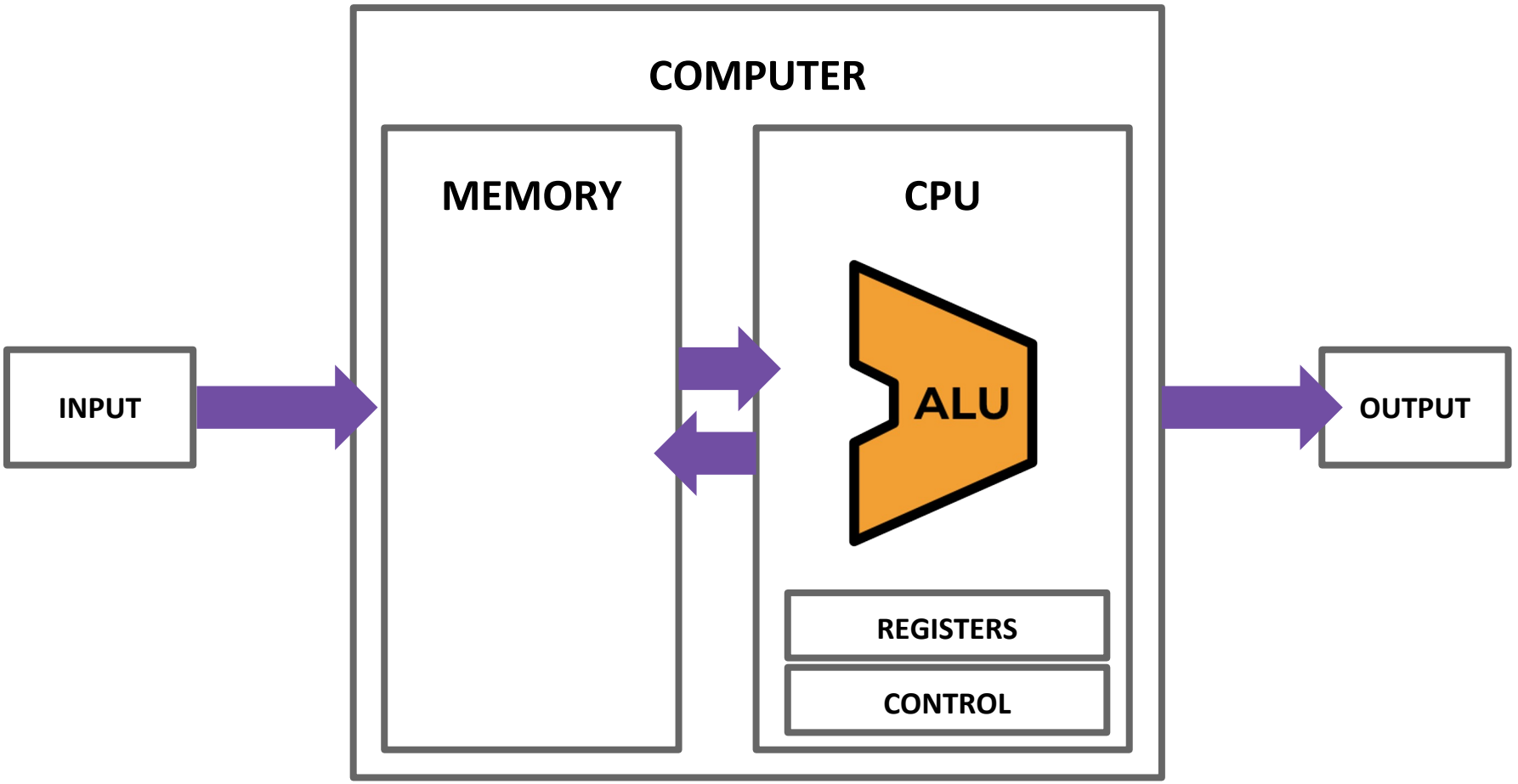
carry				
a	1	0	0	1
b	0	0	1	0
sum	1	0	1	1

- Example: $0b1001 + 0b0010 = 0b1011$
  - Unsigned interpretation: $9 + 2 = 11$
  - Signed interpretation: $-1 + 2 = -3$ (?)
  - Two's Complement interpretation: $-7 + 2 = -5$

Lecture Outline

- Growth vs. Fixed Mindset
  - Setting SMART Goals
- Negative Numbers in Binary
  - Unsigned, Signed, and Two's Complement Representations
- **Arithmetic Logic Unit (ALU)**
  - **Overview and Examples of ALU Functions**
- Project 3 Overview
  - ALU Implementation Strategy
  - HDL Tips and Tricks

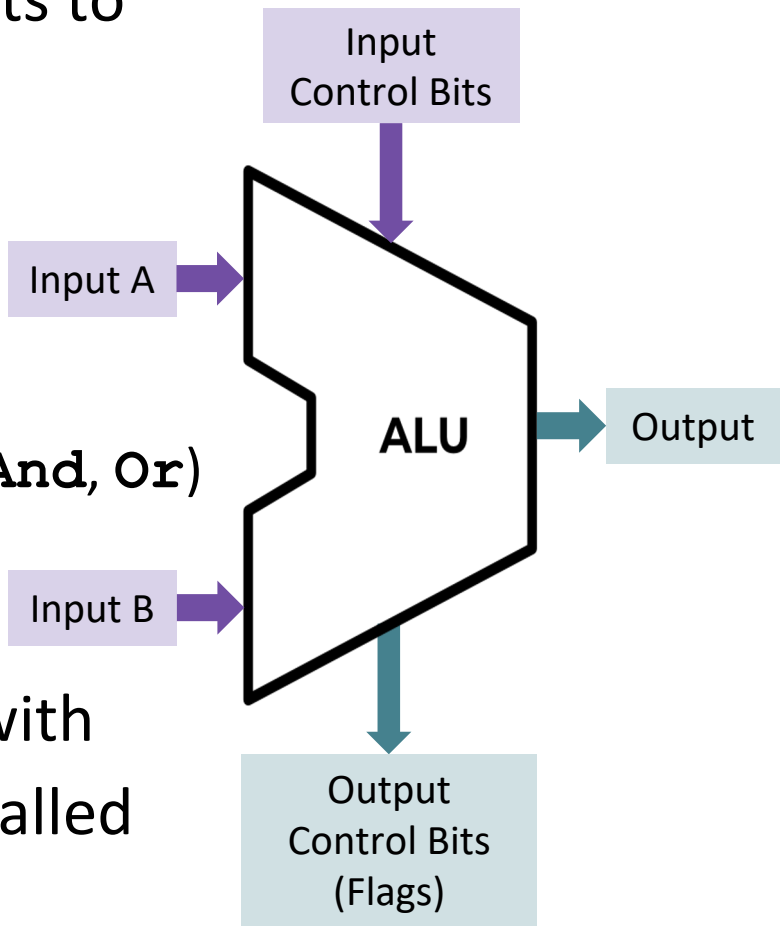
The Von Neumann Architecture



(This picture will get more detailed as we go!)

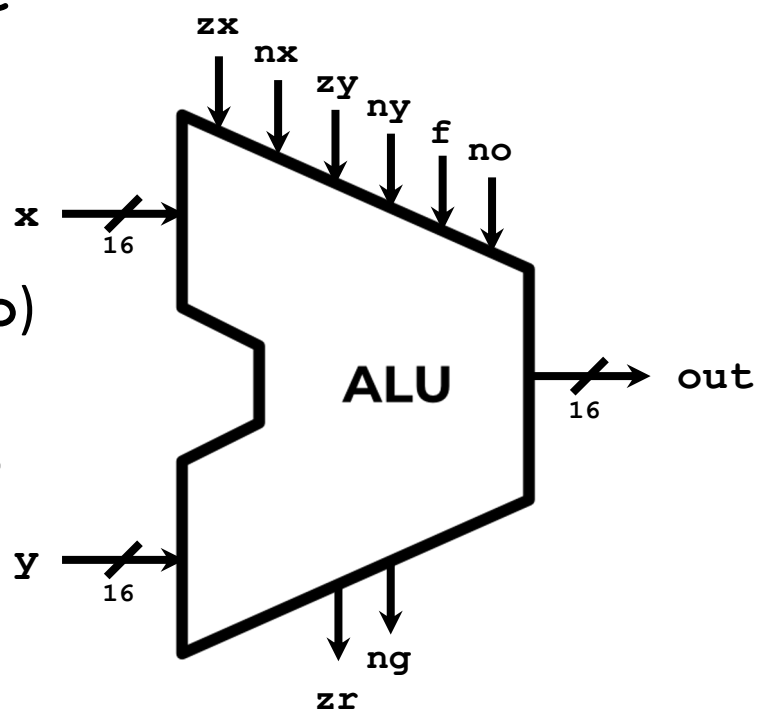
The Arithmetic Logic Unit

- Computes a function on two inputs to produce an output
- Input Control Bits specific which function should be computed
  - Supports a combination of logical (**`And`**, **`Or`**) and arithmetic operations (+, –)
- Indicate properties of the result with **Output Control Bits** (commonly called **Flags**)



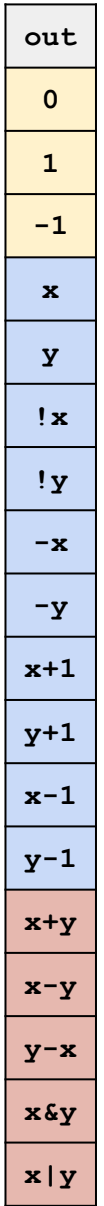
Our ALU Implementation

- Inputs & Outputs
  - 16-bit inputs **x** and **y** and output **out**
  - Interpret in Two's Complement
- Input Control Bits
  - Six control bits (**zx**, **nx**, **zy**, **ny**, **f**, **no**) specify which function to compute
  - $2^6 = 64$ different possible functions to choose from (only 18 of interest)
- Output Control Bits (Flags)
  - 2 bits (**zr** and **ng**) describing the properties of the output



ALU Functions: Client's View

- We support 18 different functions of interest
  - 3 that simply give constant values (ignoring operands)
  - 10 that change a single input, possibly with a constant
  - 5 that perform an operation using both inputs



ALU Functions: Client's View

- We support 18 different functions of interest
  - 3 that simply give constant values (ignoring operands)
  - 10 that change a single input, possibly with a constant
  - 5 that perform an operation using both inputs

- To select a function, set the control bits to the corresponding combination

zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

ALU Functions: Implementer's View

- These 18 functions are really a clever combination of 6 core operations:

1 PREPROCESS INPUTS

if **zx==1**, set **x** to 0

if **zy==1**, set **y** to 0

if **nx==1**, negate **x**

if **ny==1**, negate **y**

2 COMPUTE

if **f==1**, result is **x + y**
else, result is **x & y**

3 POSTPROCESS OUTPUT

if **no==1**, negate result

zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

ALU Functions: Implementer's View

- Example: Compute **`x - 1`**
  - Given inputs **`x=0b0101`** (5), **`y=0b0010`** (2)

zx	nx	zy	ny	f	no	out
...						...
0	0	1	1	1	0	x-1
...						...

ALU Functions: Implementer's View

- Example: Compute **`x - 1`**
  - Given inputs **`x=0b0101`** (5), **`y=0b0010`** (2)

zx	nx	zy	ny	f	no	out
...						...
0	0	1	1	1	0	x-1
...						...

1
PREPROCESS INPUTS

if **`zx==1`**, set **`x`** to 0

if **`zy==1`**, set **`y`** to 0

if **`nx==1`**, negate **`x`**

if **`ny==1`**, negate **`y`**

`x=0b0101` (5) *Unchanged*

`y=0b0000` (0) *Zeroed*

`x=0b0101` (5) *Unchanged*

`y=0b1111` (-1) *Negated*

ALU Functions: Implementer's View

- Example: Compute **`x - 1`**
  - Given inputs **`x=0b0101`** (5), **`y=0b0010`** (2)

zx	nx	zy	ny	f	no	out
...						...
0	0	1	1	1	0	x-1
...						...

1
PREPROCESS INPUTS

if **`zx==1`**, set **`x`** to 0

if **`zy==1`**, set **`y`** to 0

if **`nx==1`**, negate **`x`**

if **`ny==1`**, negate **`y`**

`x=0b0101` (5) *Unchanged*

`y=0b0000` (0) *Zeroed*

`x=0b0101` (5) *Unchanged*

`y=0b1111` (-1) *Negated*

2
COMPUTE

if **`f==1`**, result is **`x + y`**
else, result is **`x & y`**

`out=0b0100` (4)
x (5) + y (-1)

ALU Functions: Implementer's View

- Example: Compute `x - 1`
  - Given inputs `x=0b0101` (5), `y=0b0010` (2)

zx	nx	zy	ny	f	no	out
...						...
0	0	1	1	1	0	x-1
...						...

1 PREPROCESS INPUTS

if `zx==1`, set `x` to 0

if `zy==1`, set `y` to 0

`x=0b0101` (5) *Unchanged*

`y=0b0000` (0) *Zeroed*

if `nx==1`, negate `x`

if `ny==1`, negate `y`

`x=0b0101` (5) *Unchanged*

`y=0b1111` (-1) *Negated*

2 COMPUTE

if `f==1`, result is `x + y`
else, result is `x & y`

`out=0b0100` (4)
x (5) + y (-1)

3 POSTPROCESS OUTPUT

if `no==1`, negate result

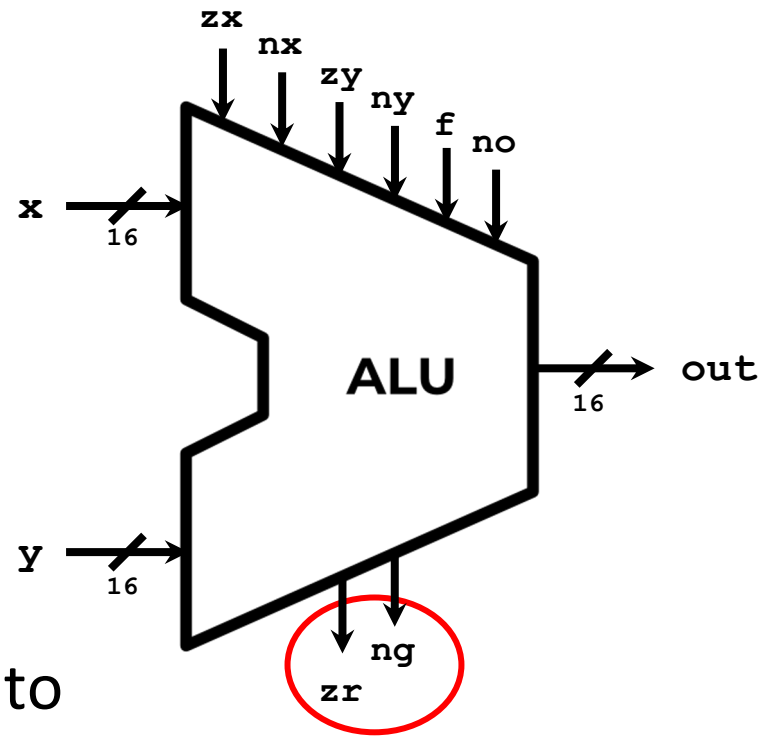
`out=0b0100` (4)
Unchanged

ALU Output Control Bits

- **`zr`** is **`1`** if **`out == 0`**
- **`ng`** is **`1`** if **`out < 0`**

- We'll use these in a later project
  - The basis of **comparison**
  - Example: To evaluate if **`x == 4`**, compute **`x - 4`** and check **`zr`** flag

- These can be deceptively difficult to implement
  - Start early on Project 3



Five-minute Break!

- Feel free to stand up, stretch, use the restroom, drink some water, review your notes, or ask questions
- We'll be back at:
- Any song recommendations? Respond on Poll Everywhere at https://pollev.com/cse390b
- Research shows mid-lecture breaks reduce the decline of attention in the middle of lecture (Olmsted, 1999)

Olmsted III, John. "The Mid-Lecture Break: When Less Is More." *Journal of Chemical Education* (1999). https://pubs.acs.org/doi/abs/10.1021/ed076p525.

Poll Everywhere

Vote at https://pollev.com/cse390b

Given inputs `x=0b1010` and `y=0b0110` and the following input control bits, what is the resulting output and output control bits?

A. out=0b1110, zr=0, ng=0

B. out=0b1011, zr=0, ng=1

C. out=0b1101, zr=1, ng=0

D. out=0b1001, zr=1, ng=1

E. We're lost…

zx	nx	zy	ny	f	no	out
...						...
0	1	1	1	1	1	x+1
...						...

```
IN
  x[16], y[16], // 16-bit inputs
  zx, // zero the x input?
  nx, // negate the x input?
  zy, // zero the y input?
  ny, // negate the y input?
  f,  // compute out = x + y
         if 1 or x & y (if 0)
  no; // negate the out output?

OUT
  out[16], // 16-bit output
  zr, // 1 if (out == 0),
         0 otherwise
  ng; // 1 if (out < 0),
         0 otherwise
```

Lecture Outline

- Growth vs. Fixed Mindset
  - Setting SMART goals
- Negative Numbers in Binary
  - Unsigned, Signed, and Two's Complement Representations
- Arithmetic Logic Unit (ALU)
  - Overview and Examples of ALU Functions
- **Project 3 Overview**
  - **ALU Implementation Strategy**
  - **HDL Tips and Tricks**

Project 3 Overview

- Part I: 24-Hour Time Audit

- Part II: Boolean Arithmetic
  - Goal: Implement the ALU, which performs the core computations we need (+ and &)
  - First, implement `HalfAdder.hdl`, `FullAdder.hdl`, and `Add16.hdl`
  - Then, implement the ALU in the order suggested by the specification
  - Chapter 2 of the textbook has more details on the adders and ALU

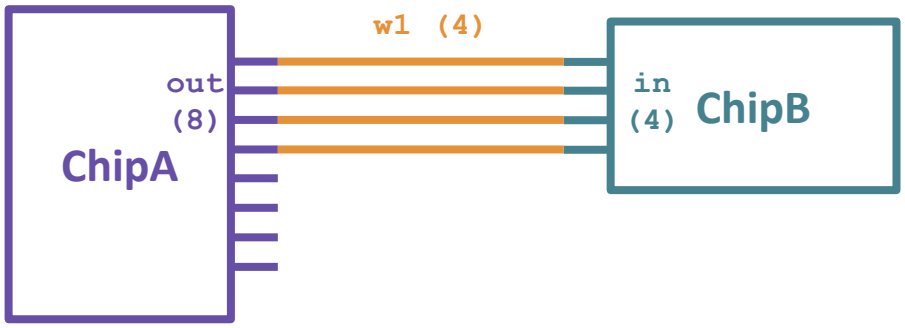
- Part III: Project 3 Reflection

ALU Implementation Strategy

- First, handle zeroing out and negating inputs **`x`** and **`y`** and negating the output
  - Ignore the **`f`** bit (only compute **`And`**) and ignore flag outputs
  - Test your implementation using `ALU-nostat-noadd.tst`
- Next, implement the **`And`** and **`Add`** operations using **`f`**
  - How do we make decisions in hardware?
  - Test your implementation using `ALU-nostat.tst`
- Lastly, implement the logic for the status flags (**`zr`** and **`ng`**)
  - Test your full ALU using `ALU.tst`

HDL Tips: Slicing

- Sometimes want to connect only part of a multi-bit bus
- HDL lets us with **slicing notation**
- Example: **ChipA** has eight output pins, and we want to connect the first four to **ChipB**'s four inputs:

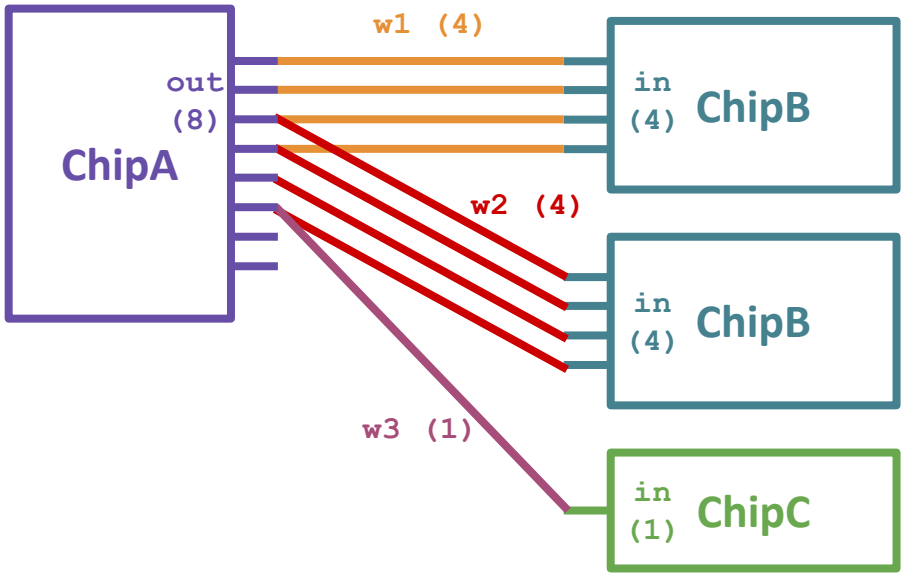



```
ChipA (out[0..3]=w1);
ChipB (in=w1);
```

- Note: We can only slice chip connections, not internal wires (e.g., `w1[0..3]` is not allowed)
  - If we need to use half an 8-bit wire, make two 4-bit wires and slice the output they're connected to

HDL Tips: Connections

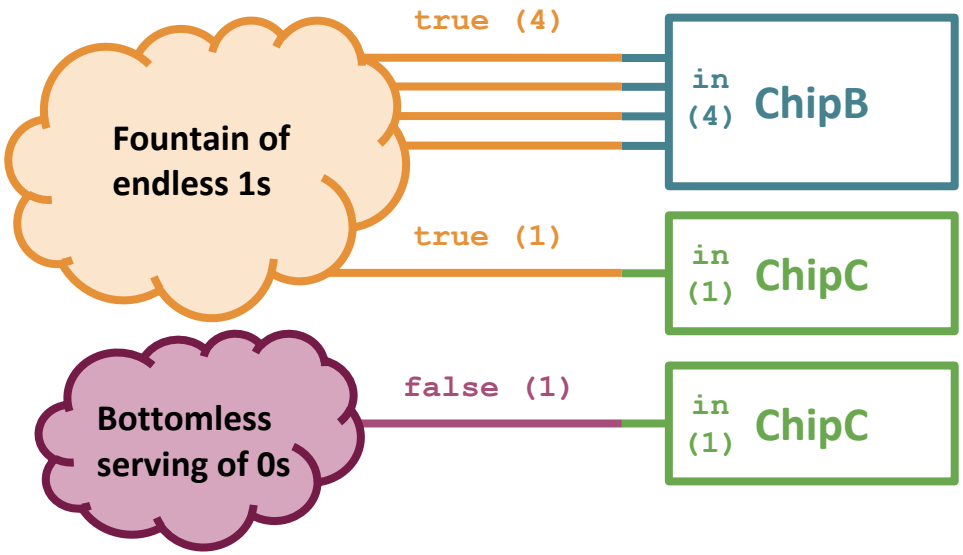
- Can connect a chip output multiple times, or not at all!
  - Hint: In `Add16.hdl`, do we need to use the last carry bit?




```
ChipA (out[0..3]=w1,
       out[2..5]=w2,
       out[5]=w3);
ChipB (in=w1);
ChipB (in=w2);
ChipC (in=w3);
```

HDL Tips: Constants

- A bus of `true` or `false` contain all 1s or all 0s, respectively, and implicitly act as whatever width is needed
- Example: `ChipB` has four inputs and `ChipC` has one input
  - `ChipB (in=true)` assigns four input bits a value of 1 (`true`)
  - `ChipC (in=true)` assigns one input bit a value of 1 (`true`)
  - `ChipC (in=false)` assigns one input bit a value of 0 (`false`)




```
ChipB (in=true);
ChipC (in=true);
ChipC (in=false);
```

Lecture 4 Wrap-up

- Exciting topics for Week 3!
  - Metacognitive Subject: Note-taking Practices 
  - Technical Subject: Sequential Logic & Building Memory

- Project Reminders
  - **Project 2 due tonight (Thursday, 4/7) at 11:59pm PDT**
  - Project 3 (24-Hour Time Audit & Boolean Arithmetic) released, due next Thursday (4/14) at 11:59pm PDT

- Join the students-only CSE 390B Discord channel!
  - https://discord.gg/Q6KBuBj8