CSE 390B, Winter 2022

Building Academic Success Through Bottom-Up Computing

The ALU, Growth vs. Fixed Mindset

Growth vs. Fixed Mindset, Goal-setting, Multiplexer, ALU, Project 2 Overview

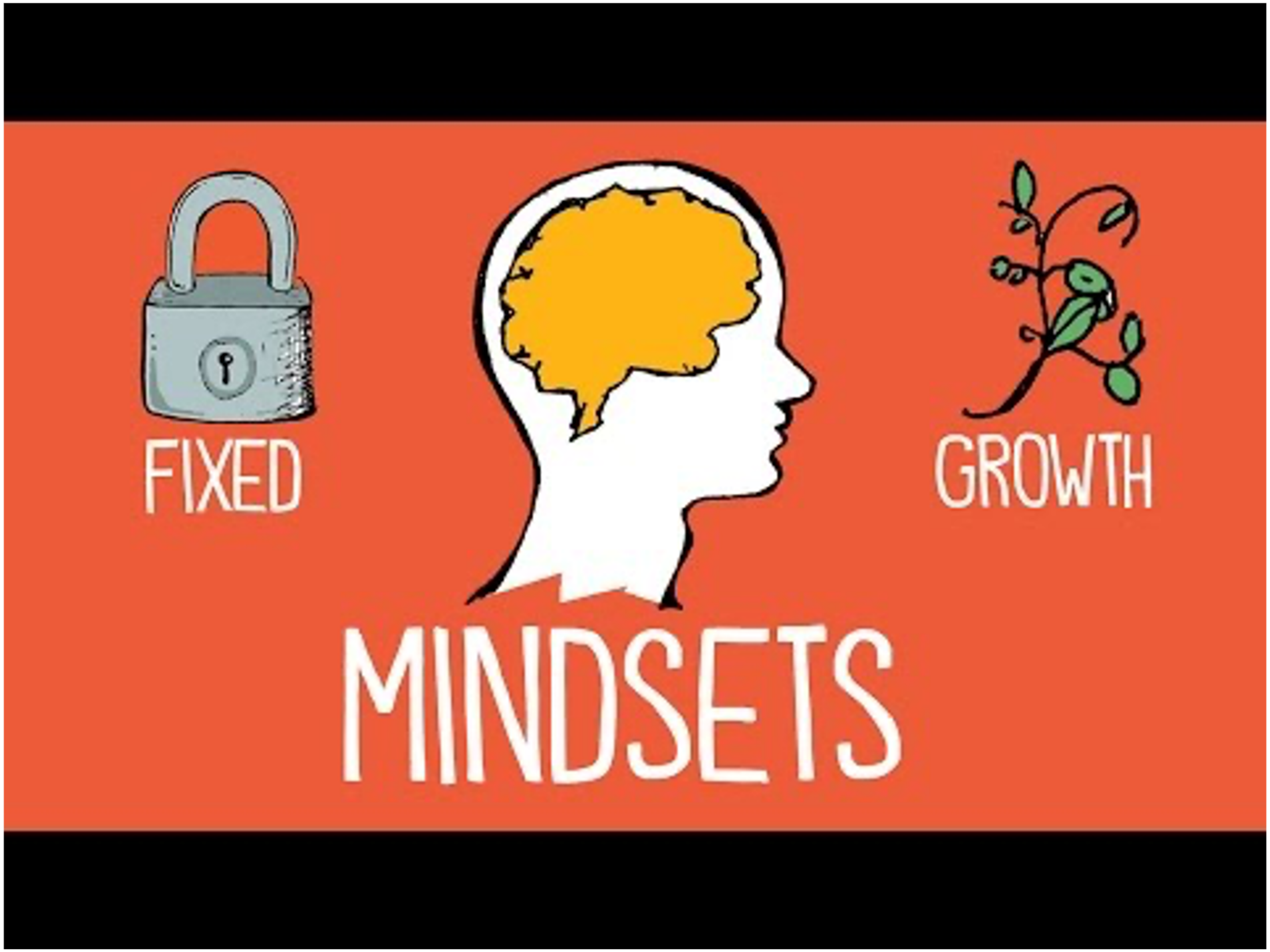
If you can, please have your camera turned on!

UNIVERSITY of WASHINGTON

Lecture Outline

- **Growth vs. Fixed Mindset**
  - **Setting SMART goals**
- Reading Review and Q&A
  - Negative Numbers in Binary
- If/Else Logic In Hardware
  - Multiplexer (Mux) logical gate
- Arithmetic Logic Unit (ALU) Introduction
  - ALU Functions and Implementation Strategy
- Project 2 Overview
  - HDL Tips

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 x goal?

Breakout Rooms

WINTER 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
- **Reading Review and Q&A**
  - **Negative Numbers in Binary**
- If/Else Logic In Hardware
  - Multiplexer (Mux) logical gate
- Arithmetic Logic Unit (ALU) Introduction
  - ALU Functions and Implementation Strategy
- Project 2 Overview
  - HDL Tips

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 $x = 4$
  - $-4 = \sim 0b0100 + 1 = 0b1011 + 0b1 = 0b1100 = -8 + 4 = -4$

4-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

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

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

Two's Complement Addition

- The same process applies for unsigned and Two's Complement binary addition

- Hardware performs the exact same calculation
  - 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

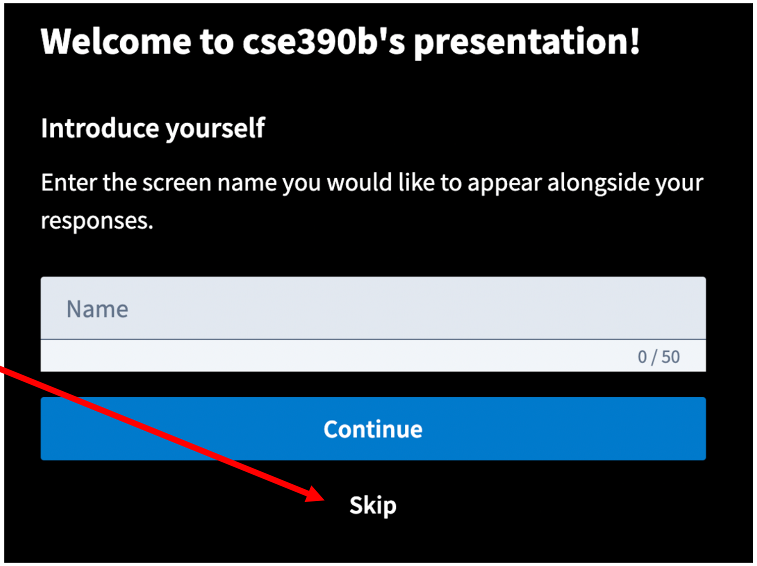
- Example: $0b1001 + 0b0010 = 0b1011$
  - Unsigned interpretation: $9 + 2 = 11$
  - Two's Complement interpretation: $-7 + 2 = -5$

Poll Everywhere

Vote at https://pollev.com/cse390b

- **What is something you learned, were surprised by, or had a question about from today's reading?**

- You can choose to respond anonymously by not adding your name (click "Skip")

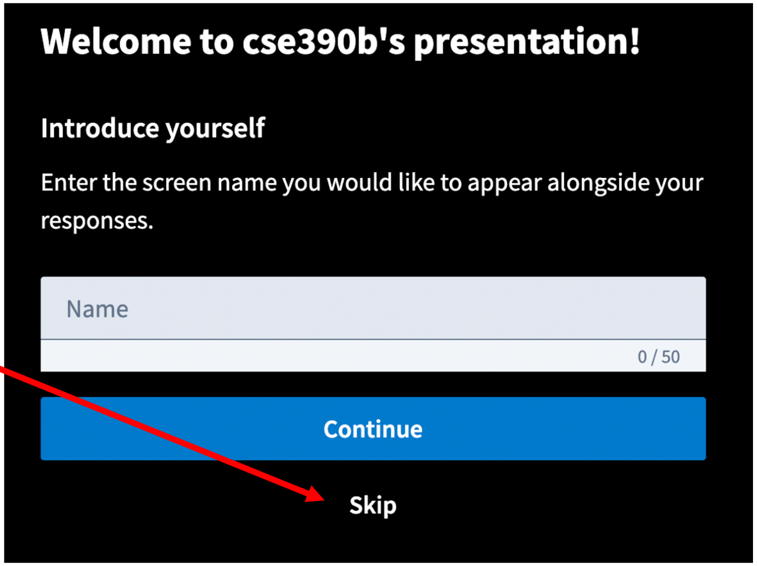


Poll Everywhere

Vote at https://pollev.com/cse390b

- **How are you feeling about Project 1? Questions, concerns, or other thoughts?**

- You can choose to respond anonymously by not adding your name (click "Skip")



Lecture Outline

- Growth vs. Fixed Mindset
  - Setting SMART goals
- Reading Review and Q&A
  - Negative Numbers in Binary
- **If/Else Logic In Hardware**
  - **Multiplexer (Mux) logical gate**
- Arithmetic Logic Unit (ALU) Introduction
  - ALU Functions and Implementation Strategy
- Project 2 Overview
  - HDL Tips

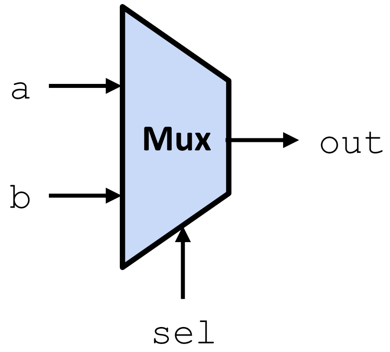
If/Else Decisions In Hardware

- We write if/else statements in Java with the understanding that only one of the branches will run
  - For example, in the following code, we expect to compute one of `a & b` or `a | b` (not both)

```
if (c == 0) {
    out = a & b;
} else {
    out = a | b;
}
```

If/Else Decisions In Hardware

- In hardware, all circuits are always executing
  - We can't "turn off" a circuit based on a condition
- We create circuits for different conditions and choose which output based on a condition instead
- We use Multiplexer (Mux) gates to choose which singular input to output



Mux Gate Implementation Example

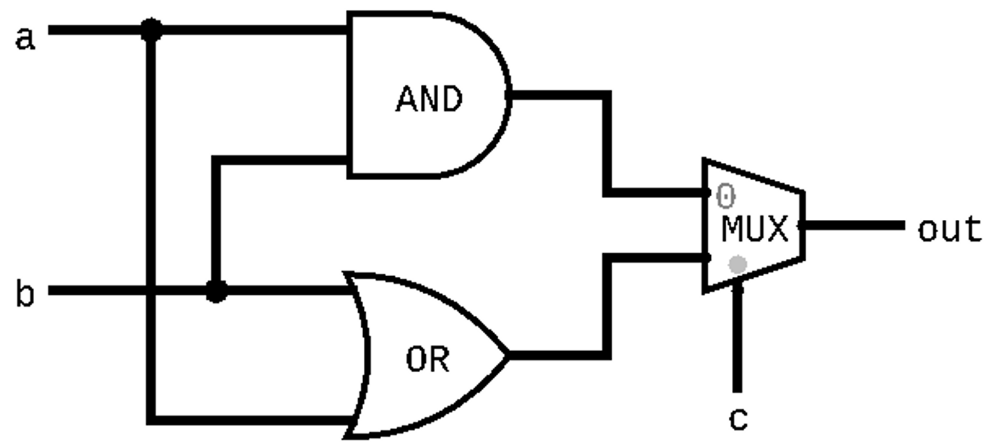
- Example of converting pseudocode into a hardware circuit diagram:

```
if (c == 0) {
    out = a & b;
} else {
    out = a | b;
}
```

Mux Gate Implementation Example

- Example of converting pseudocode into a hardware circuit diagram:

```
if (c == 0) {
    out = a & b;
} else {
    out = a | b;
}
```



Mux Gate Practice Problem #1

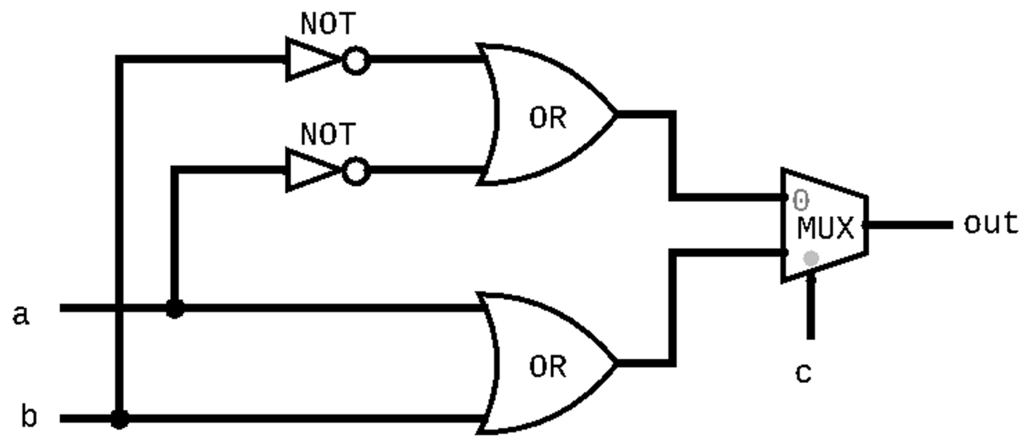
- Draw out the hardware circuit diagram that corresponds to the following pseudocode:

```
if (c == 0) {
    out = ~a | ~b;
} else {
    out = a | b;
}
```

Mux Gate Practice Problem #1

- Draw out the hardware circuit diagram that corresponds to the following pseudocode:

```
if (c == 0) {
    out = ~a | ~b;
} else {
    out = a | b;
}
```



Mux Gate Practice Problem #2

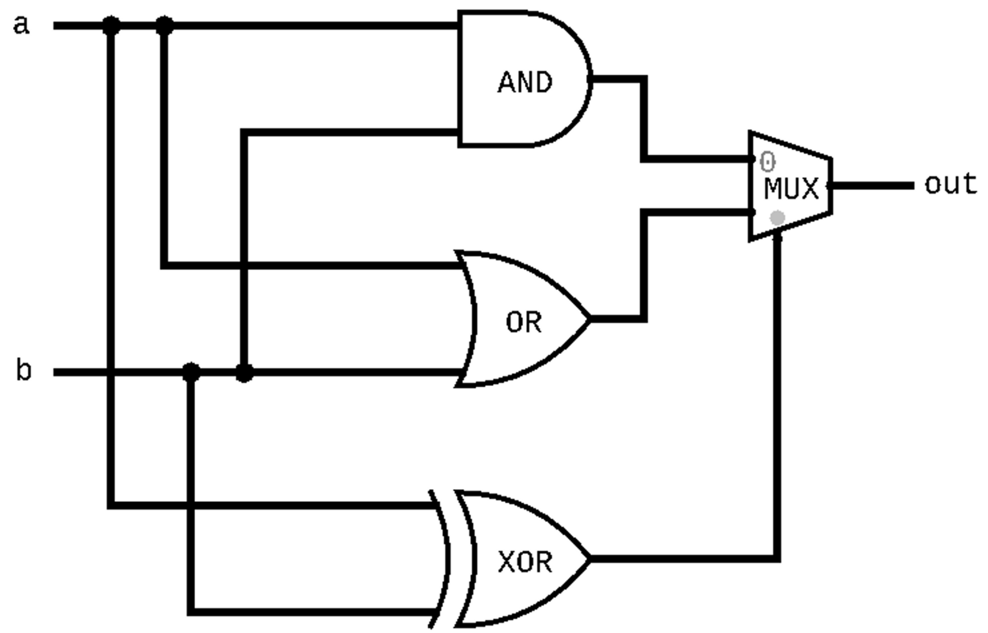
- Draw out the hardware circuit diagram that corresponds to the following pseudocode:

```
if (a == b) {
    out = a & b;
} else {
    out = a | b;
}
```

Mux Gate Practice Problem #2

- Draw out the hardware circuit diagram that corresponds to the following pseudocode:

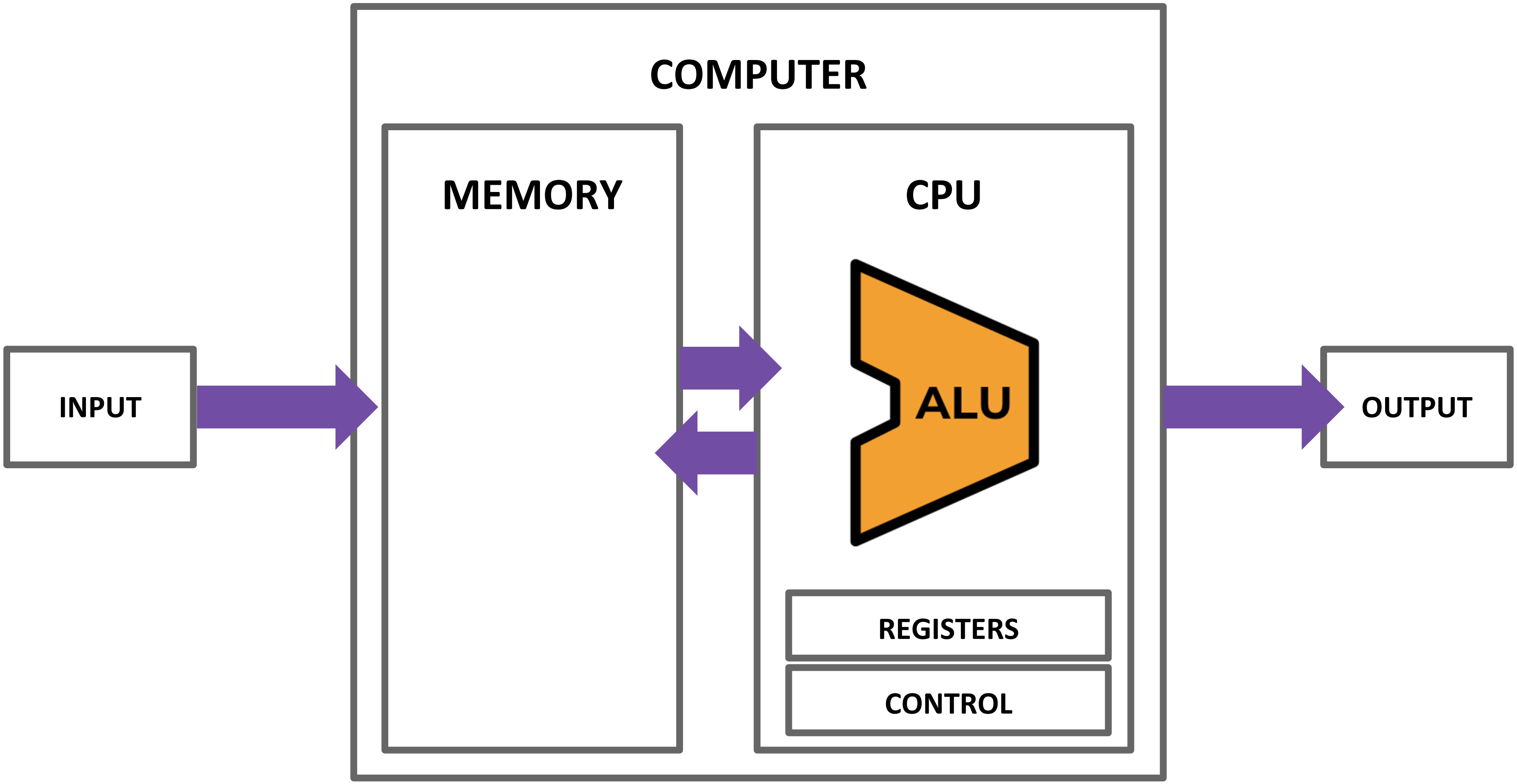
```
if (a == b) {
    out = a & b;
} else {
    out = a | b;
}
```



Lecture Outline

- Growth vs. Fixed Mindset
  - Setting SMART goals
- Reading Review and Q&A
  - Negative Numbers in Binary
- If/Else Logic In Hardware
  - Multiplexer (Mux) logical gate
- **Arithmetic Logic Unit (ALU) Introduction**
  - **ALU Functions and Implementation Strategy**
- Project 2 Overview
  - HDL Tips

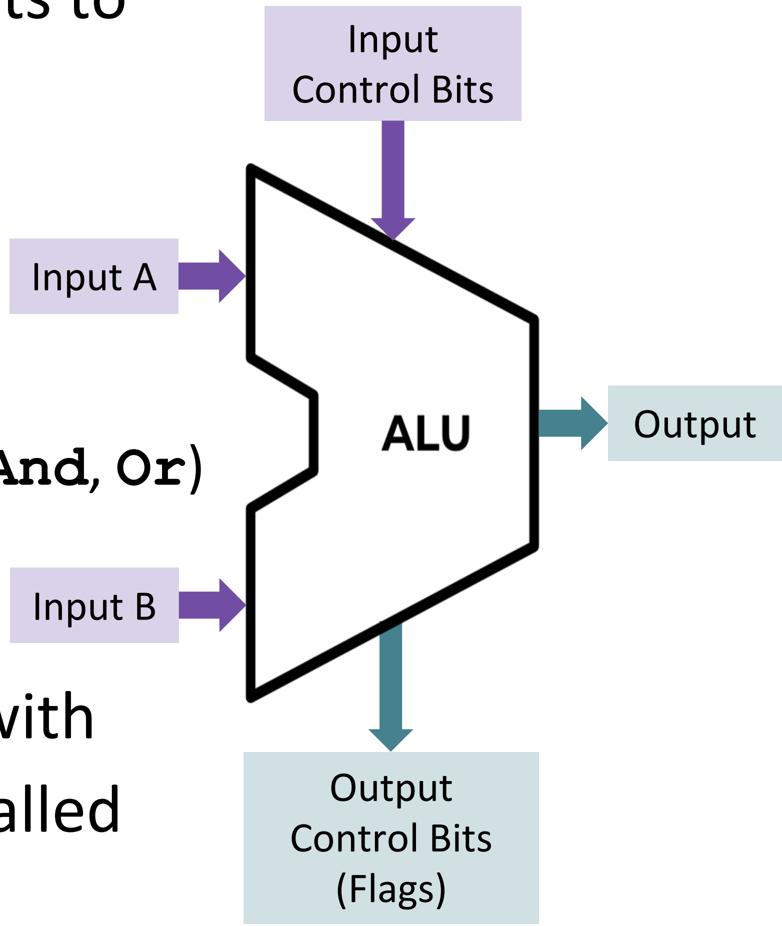
The Von Neumann Architecture



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

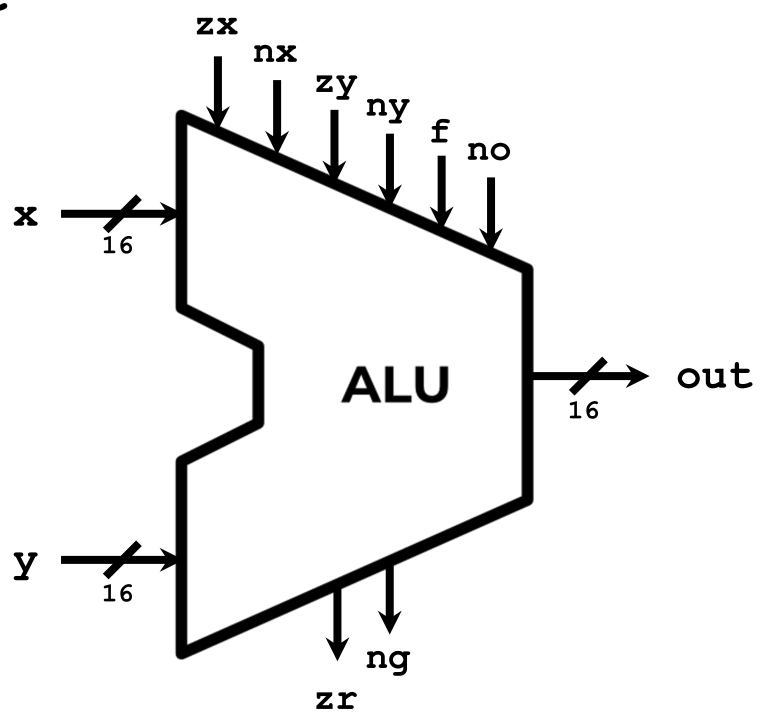
The Arithmetic Logic Unit

- 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 & Output
  - 16-bit inputs **`x`** and **`y`** and output **`out`**
  - Interpret in Two's Complement
- Input Control Bits
  - 6 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: "Black Box" 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

- Example: Compute **x - 1**
  - Given inputs **x=0101** (5), **y=0010** (2)

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

1 PREPROCESS INPUTS

if **zx**, set x to 0 → if **nx**, negate x

if **zy**, set y to 0 → if **ny**, negate y

x=0101 (5) *Unchanged* — **y=0000** (0) *Zeroed*

x=0101 (5) *Unchanged* — **y=1111** (-1) *Negated*

2 COMPUTE

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

out=0100 (4)
x (5) + y (-1)

3 POSTPROCESS OUTPUT

if **no**, negate result

out=0100 (4)
Unchanged

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=0b1010, zr=0, ng=1`

B. `out=0b0111, zr=0, ng=0`

C. `out=0b1001, zr=1, ng=0`

D. `out=0b1100, zr=1, ng=1`

E. We're lost…

zx	nx	zy	ny	f	no
…					
0	1	1	1	0	1
…					

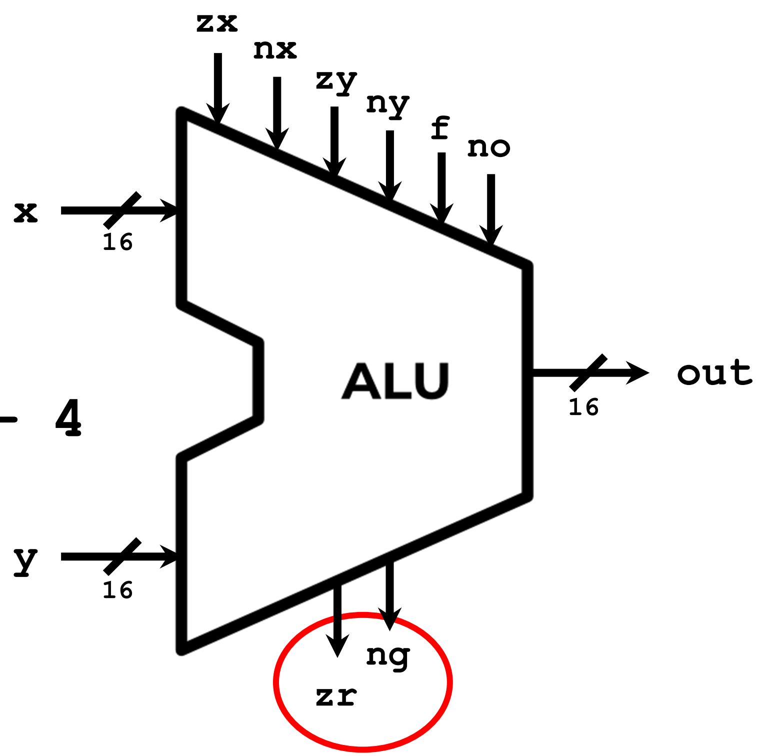
```
IN
  x[16], y[16], // 16-bit inputs
  zx, // zero the x input?
  nx, // negate the x input?
  zy, // zero the y input?
  ny, // negate the y input?
  f,  // compute out = x + y
         if 1 or x & y (if 0)
  no; // negate the out output?

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

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**
  - To evaluate if `x == 4`, compute `x – 4` and check `zr` flag



- These are deceptively difficult to implement
  - Start early on Project 2

ALU Implementation Strategy

- We suggest implement the ALU in three steps
- 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
  - 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`

Lecture Outline

- Growth vs. Fixed Mindset
  - Setting SMART goals
- Reading Review and Q&A
  - Negative Numbers in Binary
- If/Else Logic In Hardware
  - Multiplexer (Mux) logical gate
- Arithmetic Logic Unit (ALU) Introduction
  - ALU Functions and Implementation Strategy
- **Project 2 Overview**
  - **HDL Tips**

Project 2 Overview

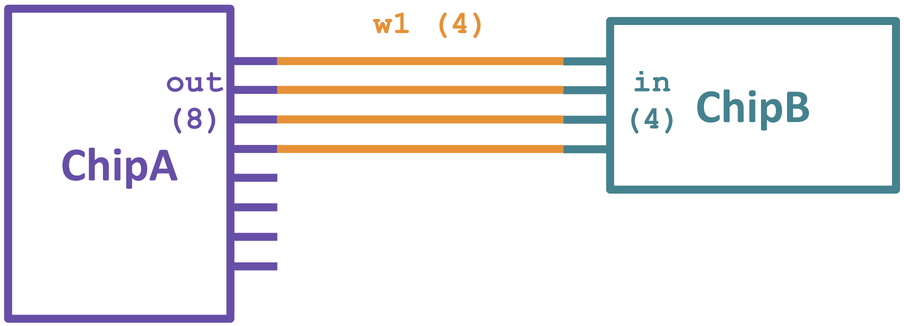
- Part I: 24-Hour Time Audit

- 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: Social Computing Reflection
  - Application of Boolean Arithmetic: Buffer Overflow

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:

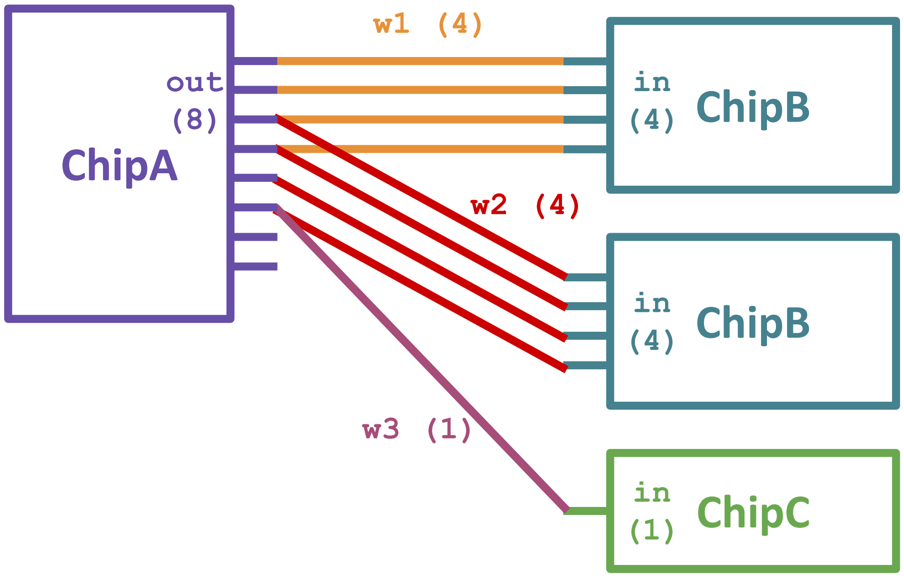



```
ChipA (out[0..3]=w1);
ChipB (in=w1);
```

- 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

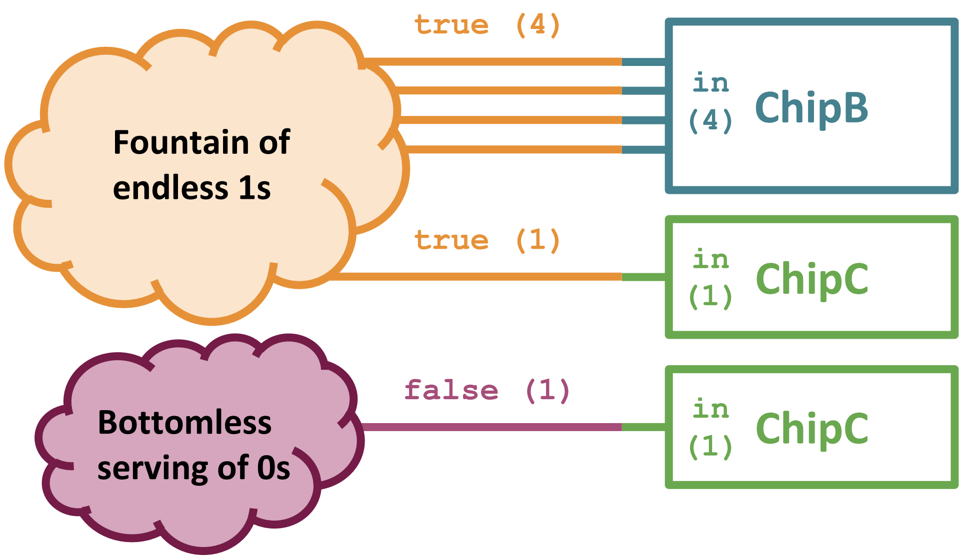
- Can connect a chip output multiple times, or not at all!
  - Hint: In `Add16.hdl`, do we need to use the last carry bit?




```
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 all 4 inputs the value of 1 (true)
  - `ChipC (in=true)` assigns the one input the value of 1 (true)
  - `ChipC (in=false)` assigns the one input the value of 0 (false)




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

Post-Lecture 4 Reminders

- What's in store for Week 3?
  - Technical Subject: Sequential Logic & Building Memory
  - Metacognitive Subject: Note-taking Practices
  - Project 3 released next Thursday (1/20)

- Reminders
  - Project 1 due **tonight (Thursday, 1/13) 11:59PM PST**
  - Project 2 (Boolean Arithmetic and 24-Hour Time Audit) is released
  - Join the Discord channel!