CSE 390 B Spring 2021

Growth vs. Fixed Mindset & The ALU

Growth vs. Fixed Mindset, Goal-Setting and the ALU

Significant material adapted from www.nand2tetris.org. © Noam Nisan and Shimon Schocken.

W UNIVERSITY of WASHINGTON

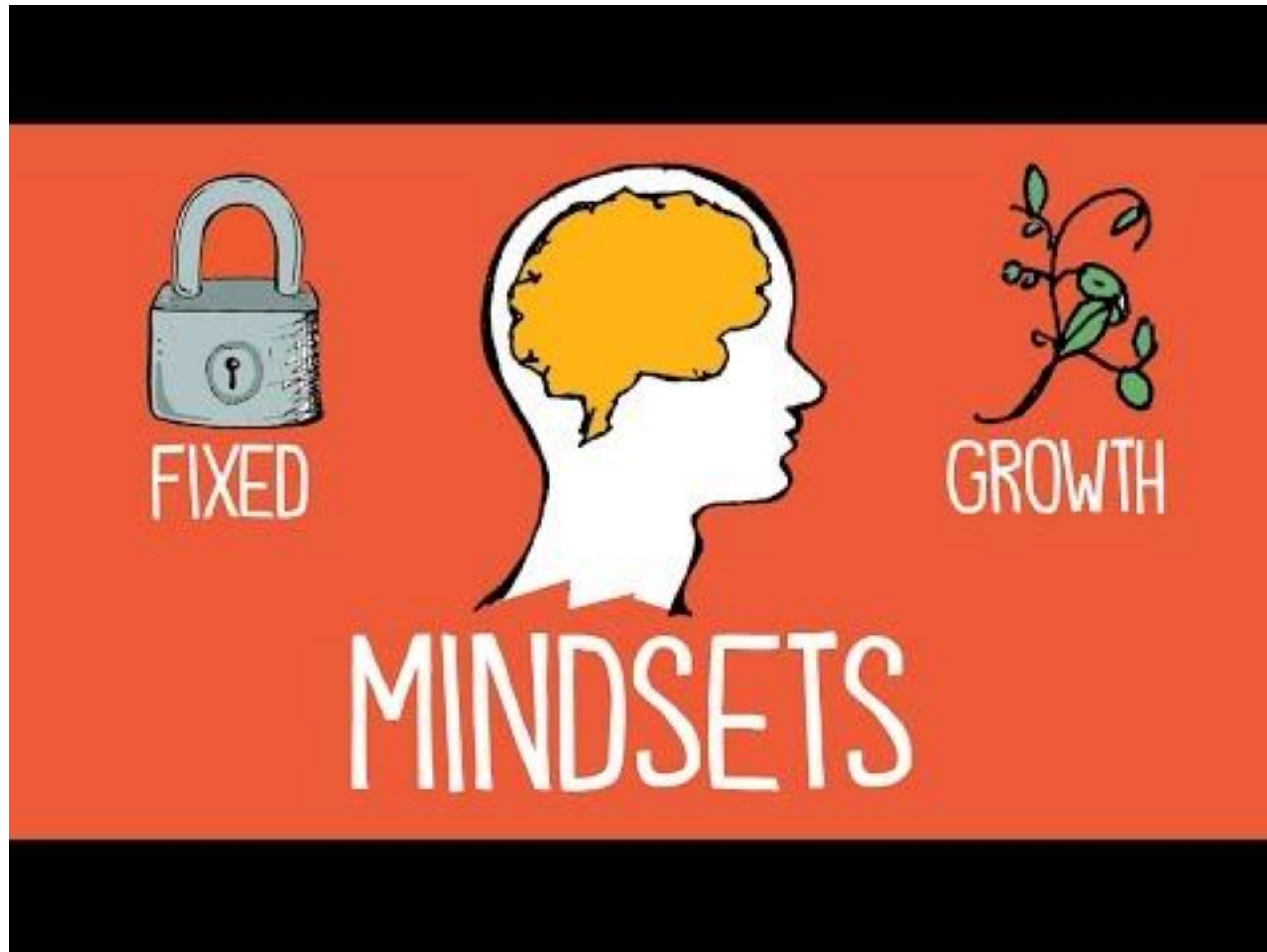
Agenda

- Growth vs. Fixed Mindset
- Reading Review and Q&A
- If/Else Logic In Hardware
- Introduction to the Arithmetic Logic Unit (ALU)
- Project 2 Overview

Agenda

- **Growth vs. Fixed Mindset**
- Reading Review and Q&A
- If/Else Logic In Hardware
- Introduction to the Arithmetic Logic Unit (ALU)
- Project 2 Overview

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

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)

Agenda

- Growth vs. Fixed Mindset
- **Reading Review and Q&A**
- If/Else Logic In Hardware
- Introduction to the Arithmetic Logic Unit (ALU)
- Project 2 Overview

Two's Complement

- Way of interpreting binary numbers to represent negative values
- Give a negative weight to the most significant digit, then compute the value like normal
- Example: 1101 in a 4-bit representation
  - -2^3 + 2^2 + 2^0 = -8 + 4 + 1 = -3
- For a value x, -x = ~x + 1
  - Example: x = 4 = 0100
  - -x = ~x + 1 = ~0100 + 1 = 1011 + 1= 1100 = -8 + 4 = -4

Two's Complement 4-bit Values

Binary	2's Complement Value
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Two's Complement Addition

- Can use exactly the same addition process for both unsigned and signed (two's complement) binary!
- Hardware doesn't need to know the sign of the values it is working on, just does the same calculation
- Example: 1001 + 0010 = 1011
  - Unsigned interpretation: 9 + 2 = 11
  - Signed interpretation: -7 + 2 = -5

carry	0	0	0	
x	1	0	0	1
y	0	0	1	0
result	1	0	1	1

Reading Q&A

Agenda

- Growth vs. Fixed Mindset
- Reading Review and Q&A
- **If/Else Logic In Hardware**
- Introduction to the Arithmetic Logic Unit (ALU)
- Project 2 Overview

Making If/Else Decisions In Hardware

- In programming languages like Java, we write if/else statements with the notion that only one of the branches will execute!
  - For example, in the following code we only expect to compute either add a & b **or** a | b, not both:

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

Making If/Else Decisions In Hardware

- In hardware, all of our circuits are executing at all times
  - Can't "turn off" a circuit based on a condition

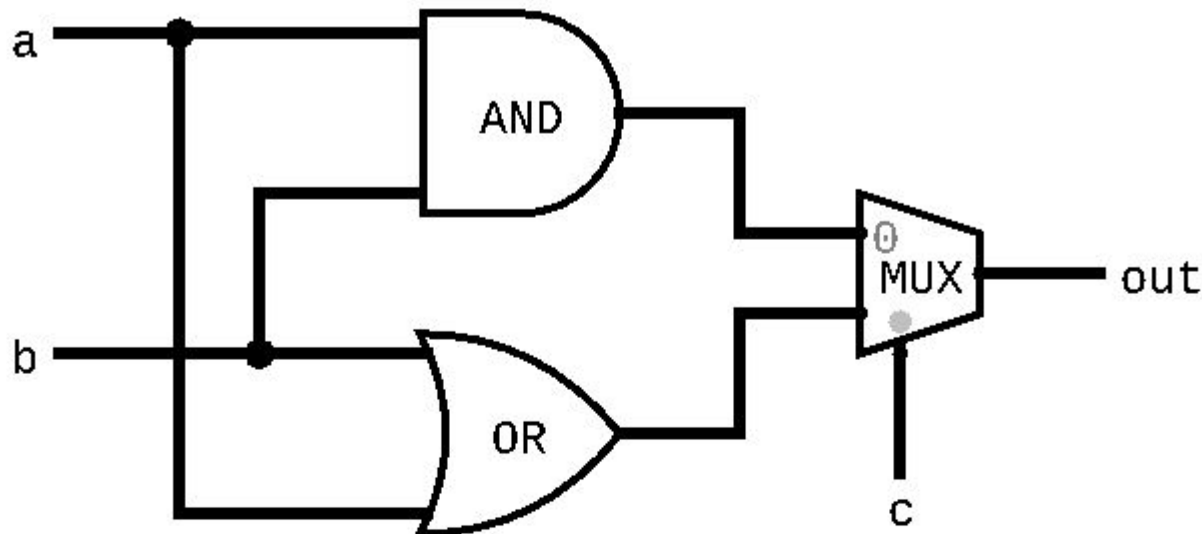
- Instead of expecting only one circuit to execute, we write circuits for different cases and choose which output we want to use based on a condition

- We can use Mux gates to choose which input!!!

Example:

High level view of implementing the following code:

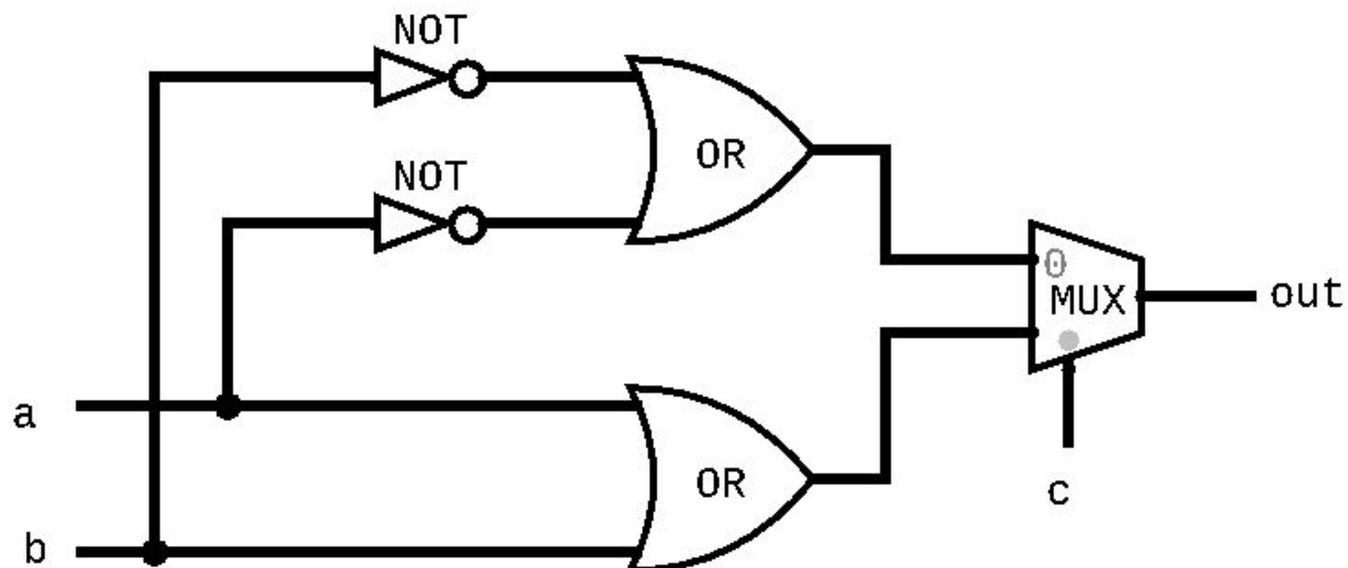
```
if (c == 0) {
    out = a & b;
} else {
    out = a | b;
}
```



Practice problem 1

Implement the following pseudo-code in hardware:

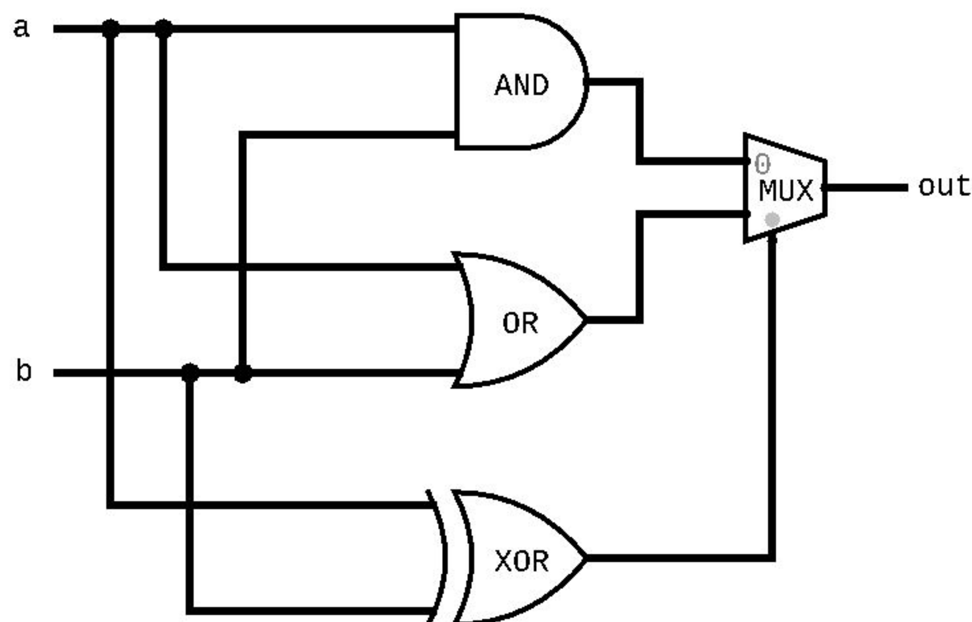
```
if (c == 0) {
    out = ~a | ~b;
} else {
    out = a | b;
}
```



Practice problem 2

Implement the following pseudo-code in hardware:

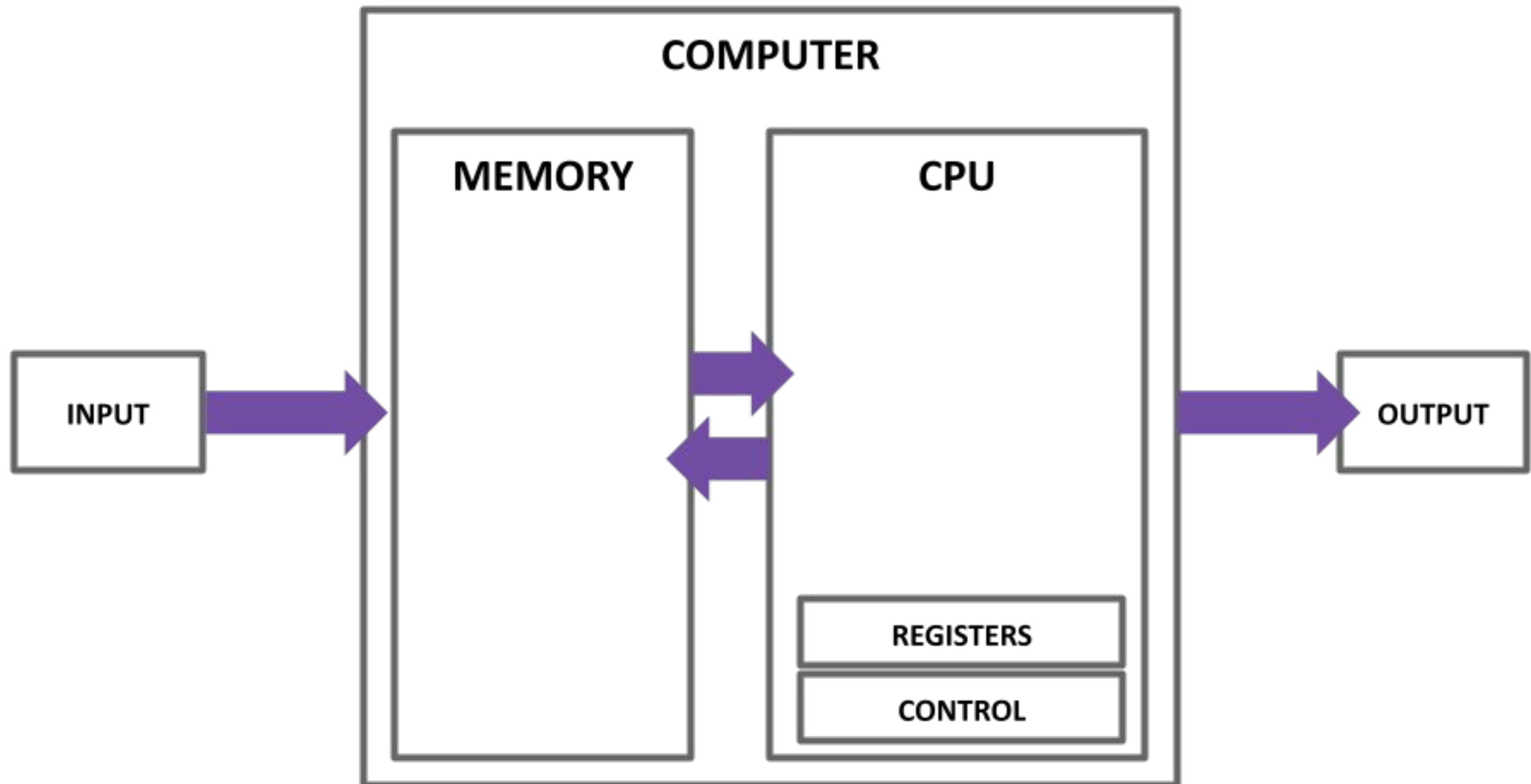
```
if (a == b) {
    out = a & b;
} else {
    out = a | b;
}
```



Agenda

- Growth vs. Fixed Mindset
- Reading Review and Q&A
- If/Else Logic In Hardware
- **Introduction to the Arithmetic Logic Unit (ALU)**
- Project 2 Overview

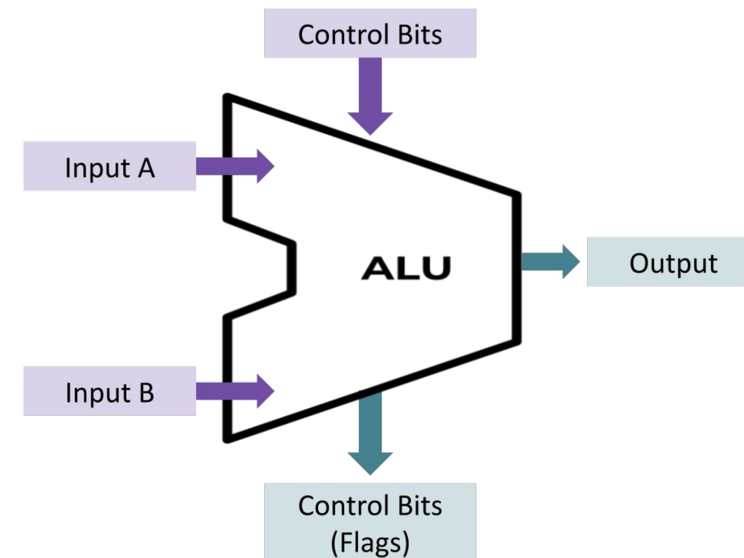
The Von Neumann Architecture



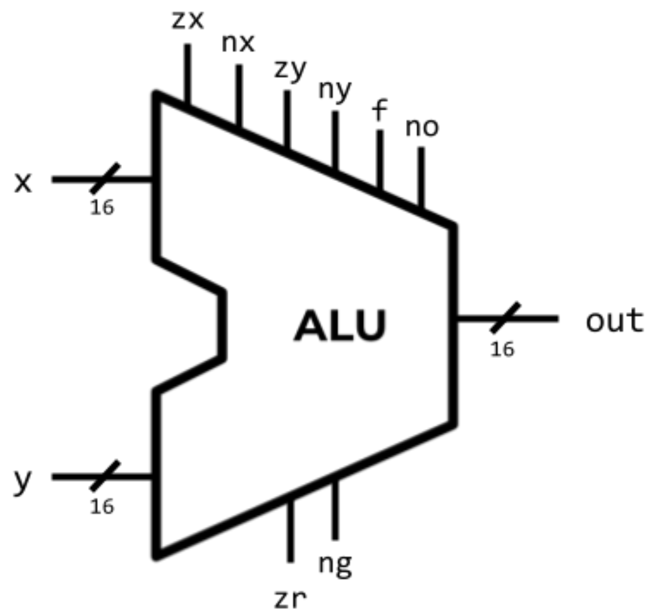
(This picture will get more detailed as we go!)

The Arithmetic Logic Unit

- Computes a function on two inputs to produce output
- Input Control Bits specify what should be computed
  - Supports some combo of logical (And, Or) and arithmetic (+, -)
- Indicate properties of the result with Output Control Bits (commonly called Flags)



Our ALU Implementation



- Inputs & Output
  - 16-bit
  - Interpret as two's complement
- Input Control Bits
  - 6 bits specify which function
  - 18 functions to choose from
- Output Control Bits (Flags)
  - 2 bits describe properties of the output

ALU Functions: "Black Box" View

- We support 18 different functions
  - 3 that simply give constant values (ignoring operands)
  - 10 that change a single operand, possibly with a constant
  - 5 that combine both operands using some operation
- 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

- "18 functions" is really clever combination of 6 core operations
- First, preprocess inputs
  - If zx is set, then zero out x
  - If nx is set, then negate x
  - If zy is set, then zero out y
  - If ny is set, then negate y
- Next compute the operation
  - If f is set, compute x + y, else compute x & y
- Finally postprocess the output
  - If no, negate the output

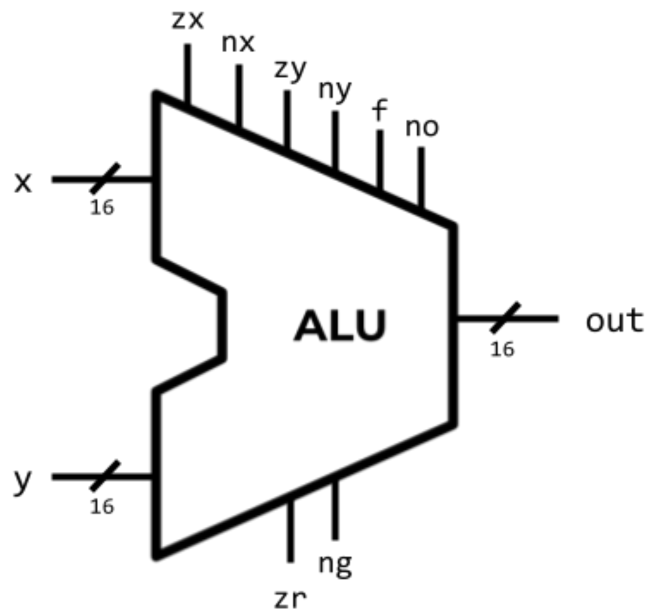
zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

ALU Functions: Implementer's View

- Example: Computing `x-1`
  - Given inputs `x=0101` (5), `y=0010` (2)
  - For x-1, zx=0, nx=0, zy=1, ny=1, f=1, no=0
  - zx=0 so **don't** zero out x
  - nx=0 so **don't** negate x
  - zy=1 so **do** zero out y
    - y becomes 0000
  - ny=1 so **do** negate y
    - y becomes 1111
  - f=1 so compute x **+** y
    - x hasn't changed, but y has become 1111 (or -1!), so this is equivalent to x + (-1)
  - no=0 so **don't** negate output
    - out = x + (-1) = 0101 + 1111 = 0100 (4)

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

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 hard to implement. Don't put them off too long!

Implementation Strategy

- We suggest doing the ALU in three parts:
  - First deal with zeroing out/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 support both And and Add operations by incorporating f
    - Test your implementation using **ALU-nostat.tst**
  - Finally implement the logic for the status flags (zr and ng)
    - Test your full ALU using **ALU.tst**

Agenda

- Growth vs. Fixed Mindset
- Reading Review and Q&A
- If/Else Logic In Hardware
- Introduction to the Arithmetic Logic Unit (ALU)
- **Project 2 Overview**

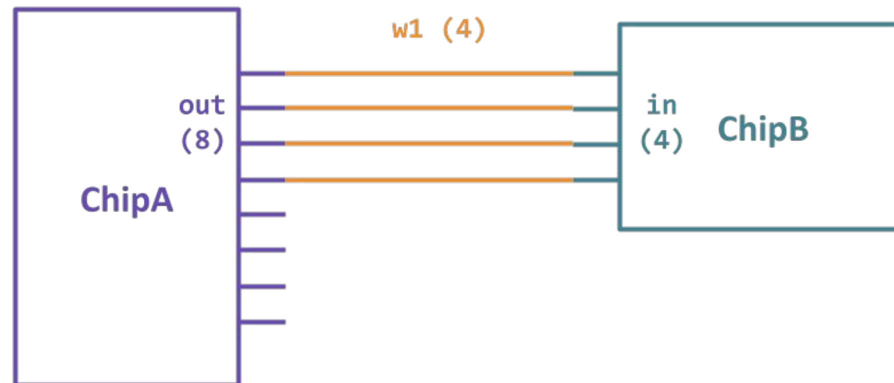
Project 2 Overview

- **Part I: 24-Hour Time Audit**
- **Part II: Boolean Arithmetic**
  - Goal: implement our ALU, which performs the core computations we need (+/&)
  - First implement HalfAdder, FullAdder, Add16
  - Then implement the ALU in the order suggested by the spec
  - Remember the textbook chapters can be helpful!!
- **Part III: Social Computer Reflection Prompt #1**

HDL Tips: Slicing

- Sometimes want to connect only part of a multi-bit bus
- HDL lets us with **slicing notation**
- Example: Say ChipA has 8 output pins, and we want to connect the first 4 to ChipB's 4 inputs:

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

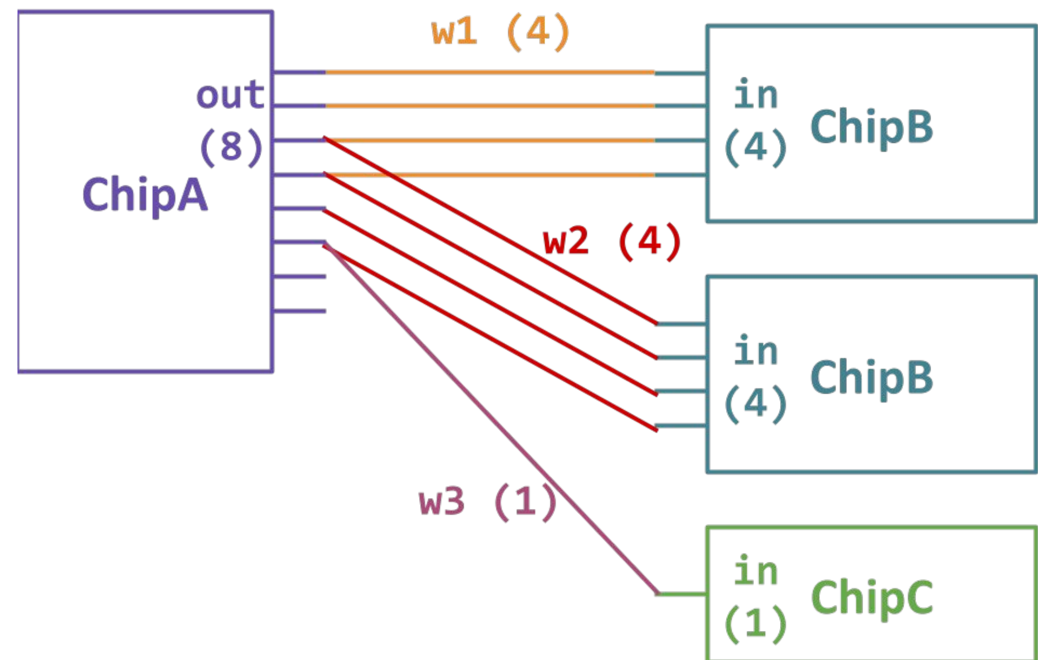


- Note: Can ONLY slice chip connections, can't slice internal wires (e.g. `w1[0..3]` not allowed!)
  - Need to use half an 8-bit wire? Instead, make two 4-bit wires and slice the output they're connected to

HDL Tips: Connections

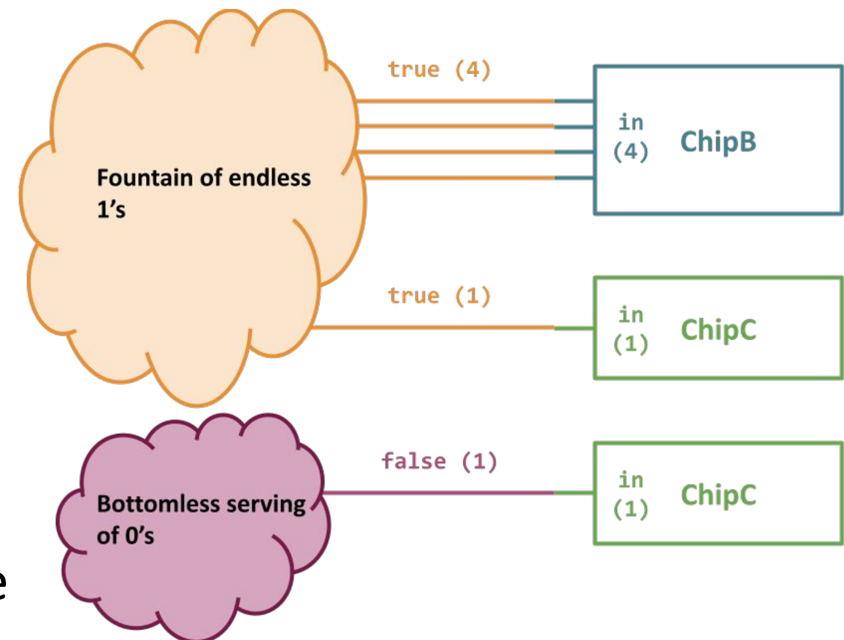
- Can connect a chip output multiple times, or not at all!
    - Hint: In Add16, do we need to use the last carry bit?

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



HDL Tips: Constants

- The special buses true and false contain all 1s and all 0s, and can implicitly act as whatever width is needed
- Example: Say ChipB has 4 inputs and ChipC has 1 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)



Wrapping Up

What's in store for Week 3?

- Sequential Logic & Building Memory
- Note-Taking Practices
- Project 3 Released

Reminders

- Project 1 Due Tonight 4/8 11:59PM PDT
- Join the discord channel