CSE 390B, Winter 2023

Building Academic Success Through Bottom-Up Computing

Procrastination & The ALU

Combating Procrastination, Binary Number Representation, The Arithmetic Logic Unit (ALU), Project 3 Overview

Lecture Outline

- **Combating Procrastination**
  - **Procrastination Reflection and Avoidance Tips**
- Binary Number Representations
  - Unsigned, Signed, and Two's Complement
- The Arithmetic Logic Unit (ALU)
  - Specification and ALU Function Examples
- Project 3 Overview
  - ALU Implementation Strategy
  - HDL Tips and Tricks

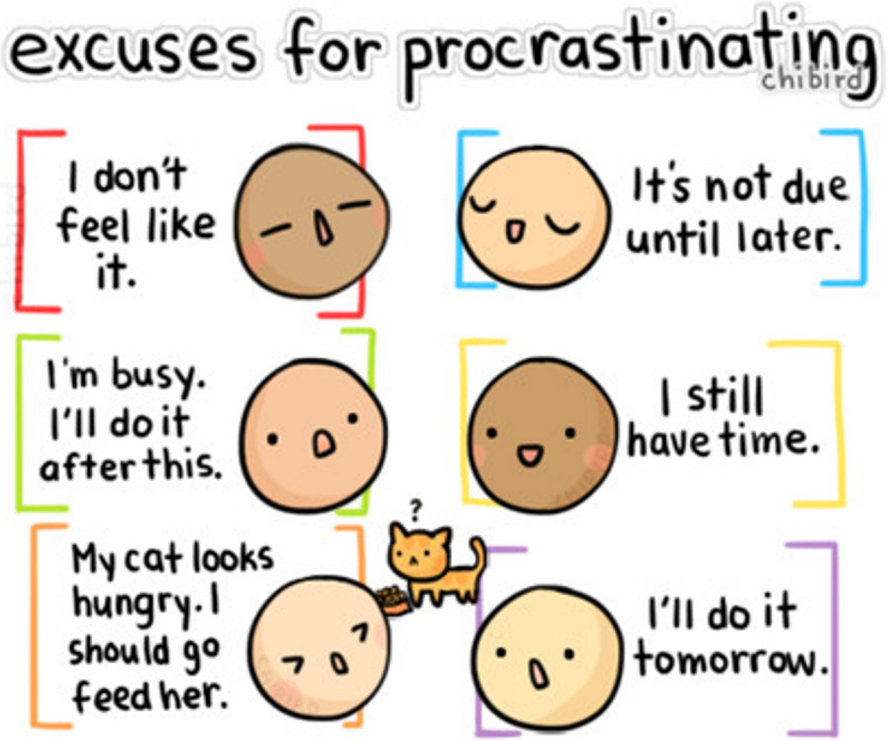
Let's Talk Procrastination

- What is procrastination?
  - Procrastination is the act of putting things off or choosing to do something you prefer to do (or might even need to do) instead of the actual project or chore or work you need to be doing now
  - Common challenge for college students, with about 80-95% of students reporting that they procrastinate (Steel, 2007)

Steel, Piers. "The nature of procrastination: a meta-analytic and theoretical review of quintessential self-regulatory failure." *Psychological Bulletin Journal* 133, no. 1 (2007): 65–94. https://www.researchgate.net/publication/6598646_The_nature_of_procrastination_a_meta-analytic_and_theoretical_review_of_quintessential_self-regulatory_failure_Psychol_Bull_133_65-94

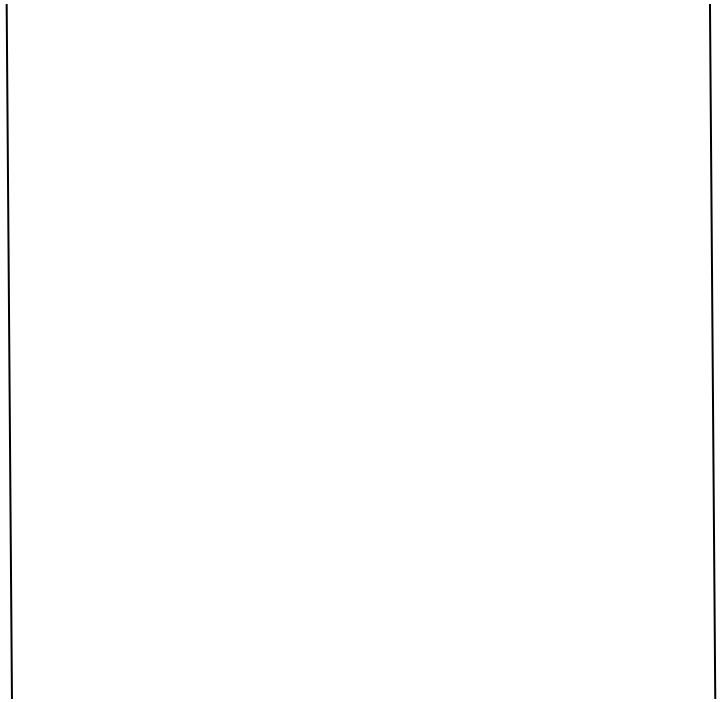
Combating Procrastination

- Identifying why we procrastinate and the internal dialogue we have with ourselves



- Create a proactive strategy to course-correct when you notice you're putting off what needs to be done

Grab a piece of paper!



Grab a piece of paper!

AVOIDANCE AREAS

When you procrastinate, what do you avoid doing?

Identify 3-5 areas

Where does procrastination impact you most?

PERSONAL

- Eating well
- Exercising / Wellness activities
- Getting enough sleep
- Bathing & hygiene
- Health care (i.e. doctor's visit)
- Balancing bank account
- Relaxation & hobbies

SCHOOL/COLLEGE

- Going to class
- Doing class readings
- Studying for tests/exams
- Doing homework/ assignments
- Writing papers
- Starting long-term projects
- Finding a study group
- Talking to an instructor or TA
- Making an advising appointment

SHOPPING/HOME/ MAINTENANCE

- Paying bills
- Getting financial aid taken care of (i.e. FAFSA, forms, etc)
- Doing laundry
- Cleaning
- Grocery shopping
- Doing dishes

SOCIAL/RELATIONSHIPS

- Talking with friends
- Writing email responses
- Socializing
- Calling relatives

WORK/CAREER

- Going to work
- Applying to internships/jobs
- Preparing a resume
- Studying for interviews

Grab a piece of paper!

AVOIDANCE AREAS

When you procrastinate, what do you avoid doing?

Identify 3-5 areas

PROCRASTINATION BEHAVIORS

How do you procrastinate? In other words, what do you do instead of the work that needs to be done?

Identify 3-5 behaviors

Grab a piece of paper!

AVOIDANCE AREAS	PROCRASTINATION BEHAVIORS	PLANNING FOR SUCCESS
When you procrastinate, what do you avoid doing?	*How do you procrastinate? In other words, what do you do instead of the work that needs to be done?*	*What can you do to avoid procrastination? What action can you take to **refocus** yourself on the task you need to complete?*
Identify 3-5 areas	*Identify 3-5 behaviors*	*Identify 3-5 actions*

Tips for Avoiding Procrastination

- Prioritize the tasks that you need to complete
- Plan for the tasks you need to complete
  - Review your to-do list and schedule of upcoming events
- Eliminate distractions that pull you away from focusing on the task at hand
  - Isolate yourself from your phone, close distracting websites, etc.
- Make productive behavior accessible and sources of procrastination harder to access

Lecture Outline

- Combating Procrastination
  - Procrastination Reflection and Avoidance Tips
- **Binary Number Representations**
  - **Unsigned, Signed, and Two's Complement**
- The Arithmetic Logic Unit (ALU)
  - Specification and ALU Function Examples
- Project 3 Overview
  - ALU Implementation Strategy
  - HDL Tips and Tricks

Unsigned Binary Representation

- To interpret, we multiply the value of each bit by the power of two that specific bit represents

- This system is unable to represent negative numbers

Exponent: 3 2 1 0

- Example: 0b1101 in **unsigned binary**
  - $(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= (1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$

$= 8 + 4 + 0 + 1$

$= 13$

Signed Binary Representation

- Also called the **sign and magnitude** number encoding
- Most significant bit (MSB) represents the **sign** of the number
  - The remaining bits represent the **weight** of the number

Exponent: 3 2 1 0

- Example: 0b1101 in **signed binary**
  - $-((1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))$

$= -((1 \times 4) + (0 \times 2) + (1 \times 1))$

$= -(4 + 0 + 1)$

$= -5$

Signed Binary Representation: Limitations

- Good first attempt at encoding negative numbers, but there are two main problems

- First, there exists two representations of zero (a positively and negatively signed zero)

- Second, adding numbers no longer works universally
  - Addition no longer works with negative numbers

```
Carry:  0 0 1 0 0

-----------------
        0 0 1 0     (2)
    +   1 0 1 0     (-2)

    -----------
        1 1 0 0     (-4) ← This should be 0
```

Two's Complement Binary Representation

- Standard for encoding numbers in computers

- Most significant bit (MSB) has a negative weight
  - Add the remaining bits as usual (with positive weights)

Exponent: 3 2 1 0

- Example: 0b1101 in Two's Complement
  - $-(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

$= -(1 \times 8) + (1 \times 4) + (0 \times 2) + (1 \times 1)$

$= -8 + 4 + 0 + 1$

$= -3$

Benefits of Two's Complement

- Only one representation of zero
- Represents more unique numbers compared to sign and magnitude given a fixed width binary number
- Simple negation procedure: Take the bitwise complement and add one ($\mathbf{-x = \sim x + 1}$)
  - Example: To negate $x = 4$:
  - $\sim 0b0100 + 1 = 0b1011 + 0b1 = 0b1100 = -8 + 4 = -4$

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: $-7 + 2 = -5$

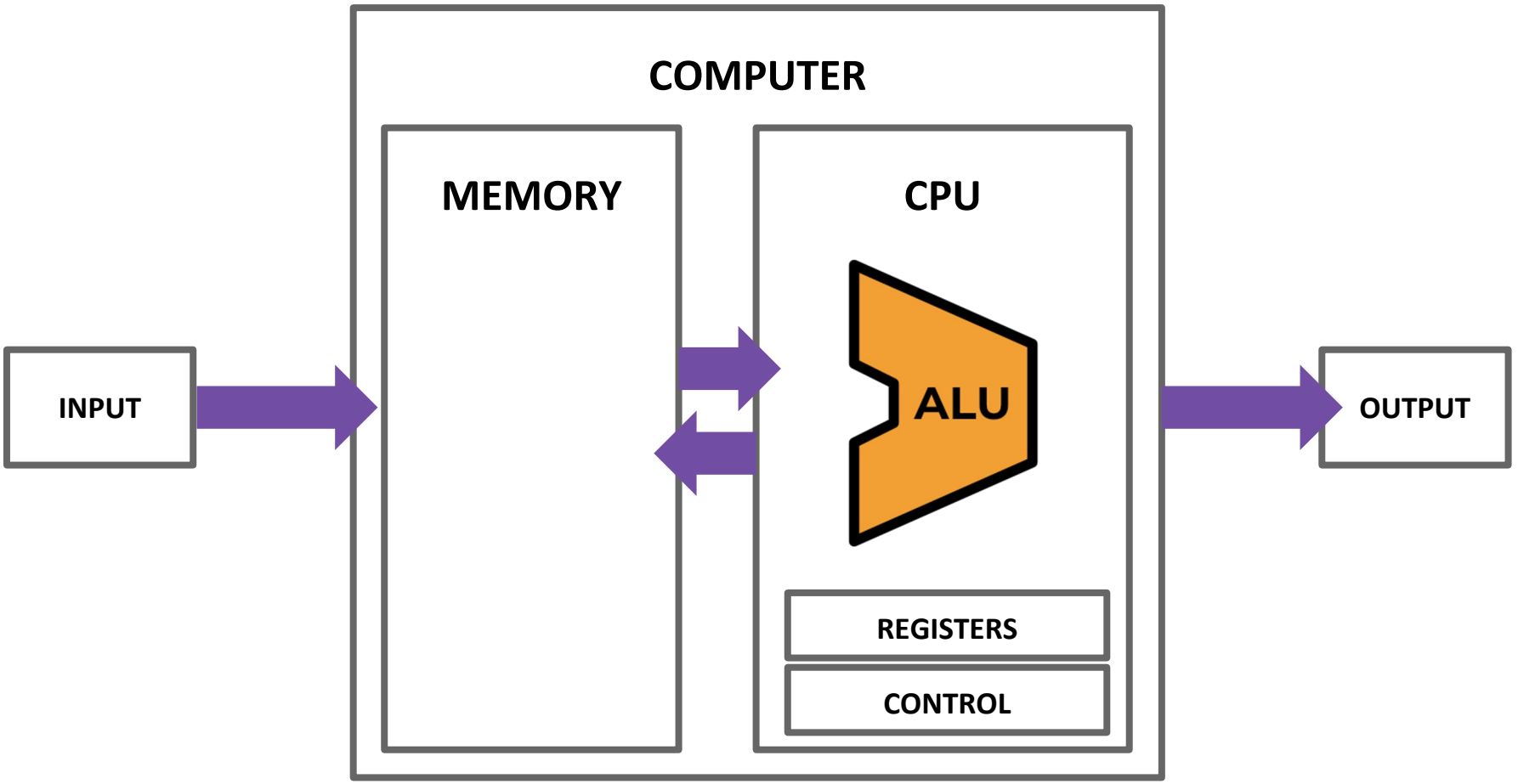
Lecture 4: Procrastination & The ALU

Loading...

Lecture Outline

- Combating Procrastination
  - Procrastination Reflection and Avoidance Tips
- Binary Number Representations
  - Unsigned, Signed, and Two's Complement
- **The Arithmetic Logic Unit (ALU)**
  - **Specification and ALU Function Examples**
- Project 3 Overview
  - ALU Implementation Strategy
  - HDL Tips and Tricks

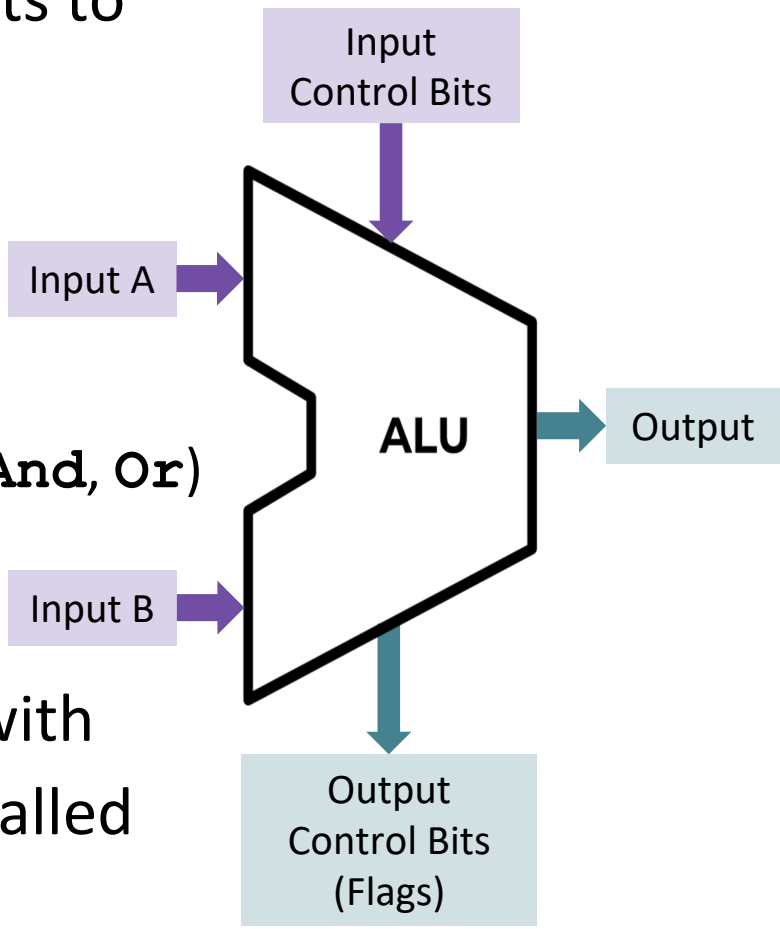
The Von Neumann Architecture



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

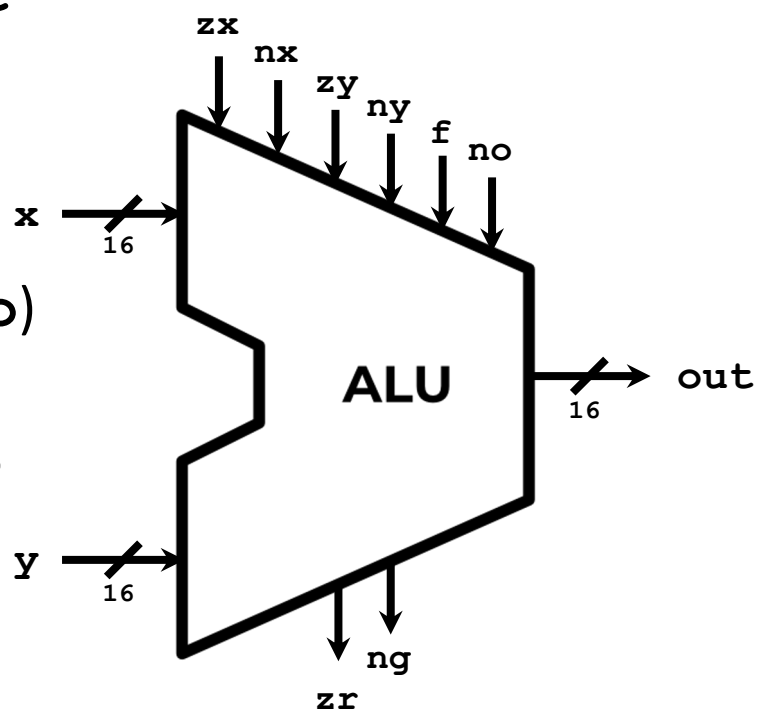
The Arithmetic Logic Unit

- 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**)



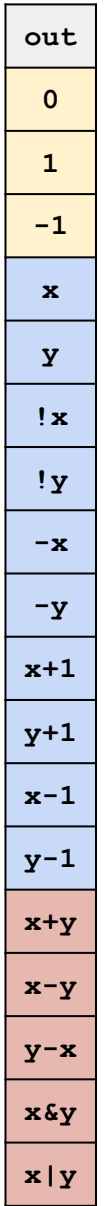
Our ALU Implementation

- 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 `nx==1`, negate `x`

if `zy==1`, set `y` to 0 → 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

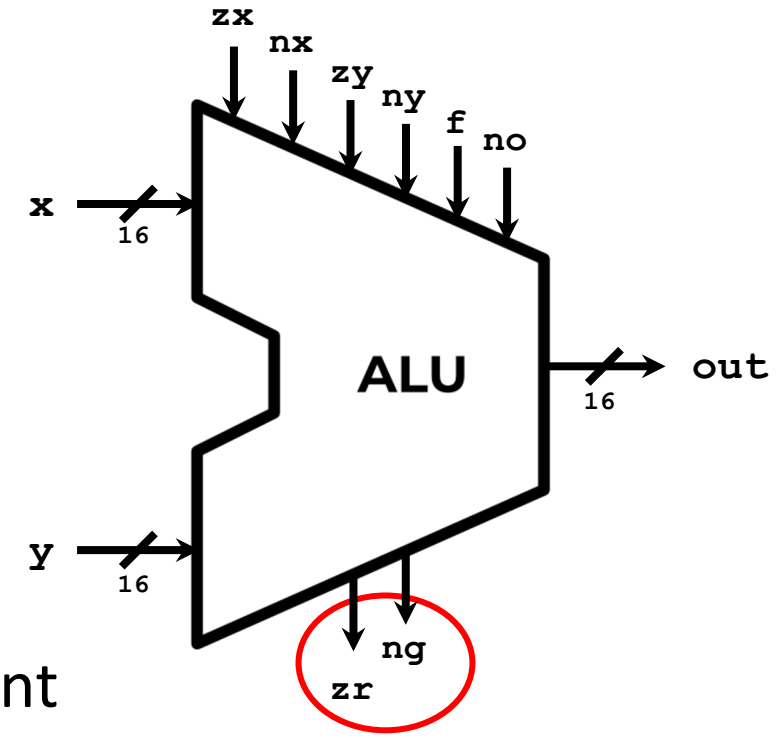
`out=0b0100` (4) *Unchanged*

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 difficult to implement
  - Applying time management strategies, please start early on Project 3!



Lecture 4: Procrastination & The ALU

Loading...

Lecture Outline

- Combating Procrastination
  - Procrastination Reflection and Avoidance Tips
- Binary Number Representations
  - Unsigned, Signed, and Two's Complement
- The Arithmetic Logic Unit (ALU)
  - Specification and ALU Function Examples
- **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

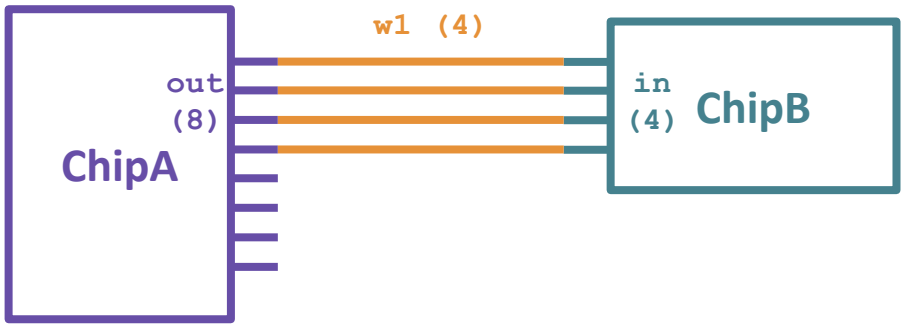
- Part III: Project 3 Reflection

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:

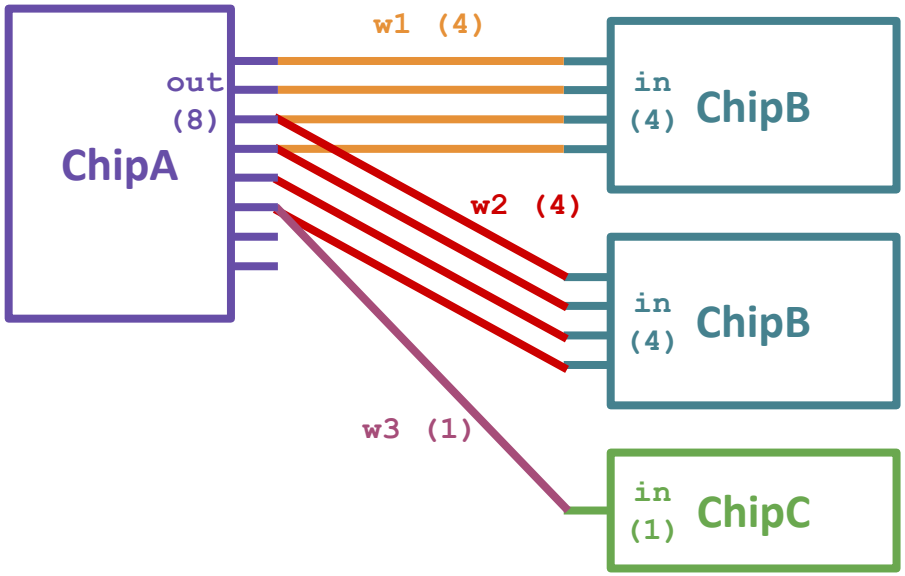



```
ChipA (out[0..3]=w1);
ChipB (in=w1);
```

- 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

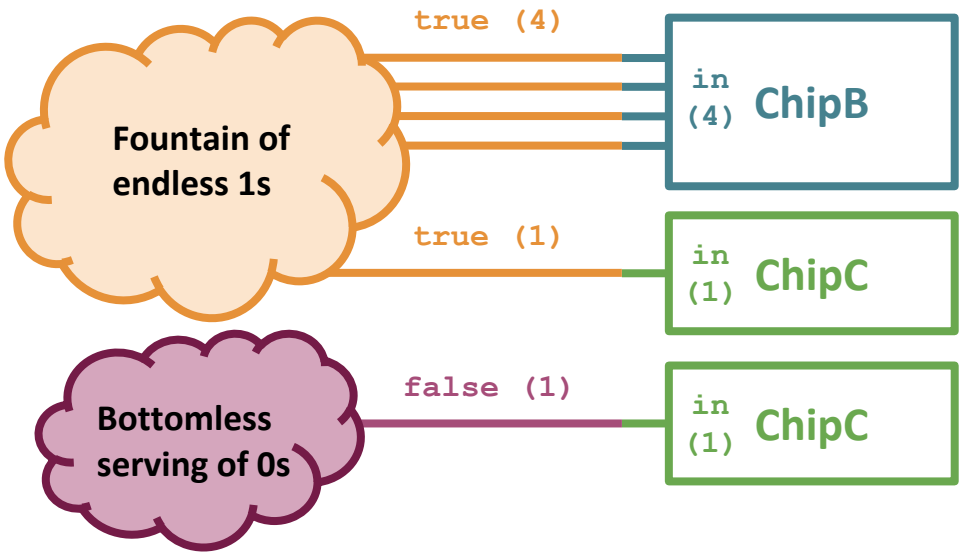
- Can connect a chip output multiple times, or not at all!
  - Hint: In `Add16.hdl`, do we need to use the last carry bit?




```
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 Reminders

- **Project 2 due tonight (1/12) at 11:59pm**

- Topics next week:
  - Metacognitive: Growth Mindset and Bloom's Taxonomy
  - Technical: Sequential Logic and Building Memory

- Course Staff Support
  - Eric has office hours in CSE2 153 today after lecture
  - Feel free to post your questions on the Ed discussion board too