CSE 390B, Winter 2022

Building Academic Success Through Bottom-Up Computing

Machine Languages, Annotation Strategies

Annotation Strategies, AMA, Machine Language Considerations, & Hack Assembly Language

If joining virtually, please have your camera turned on if you can!

Lecture Outline

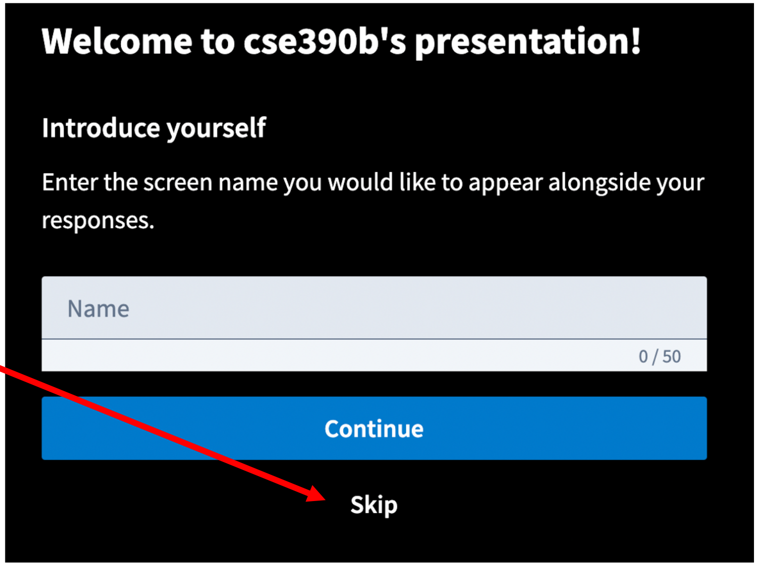
- **Pulse Check Survey**
- Annotation Strategies
- AMA: Ask Me Anything
- Reading Review and Q&A
  - Assembly Languages & Machine Code
- Project 4 Overview: Hack Assembly Language
  - Registers, A-Instructions, Symbols, C-Instructions

Poll Everywhere

Vote at https://pollev.com/cse390b

- What factors or barriers, if any, are preventing you from performing at your full potential in CSE 390B?
  - Think about CSE 390B projects, lecture attendance, lecture pace and quality, office hours, 1:1 sessions, etc.

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



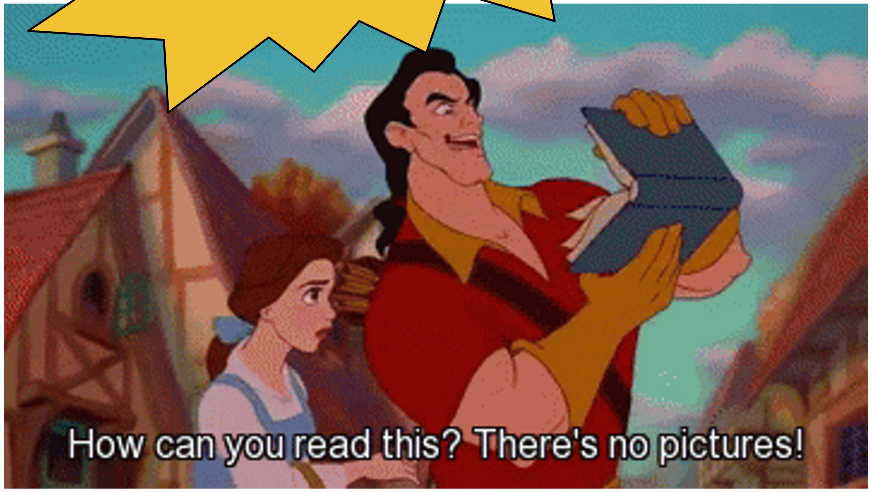
Lecture Outline

- Pulse Check Survey
- **Annotation Strategies**
- AMA: Ask Me Anything
- Reading Review and Q&A
  - Assembly Languages & Machine Code
- Project 4 Overview: Hack Assembly Language
  - Registers, A-Instructions, Symbols, C-Instructions

Annotating Your Texts

- **WHAT**
  Intentionality of interacting with a text to enhance the reader's understanding of, recall of, and reaction to the text

WHY DO WE CARE?!



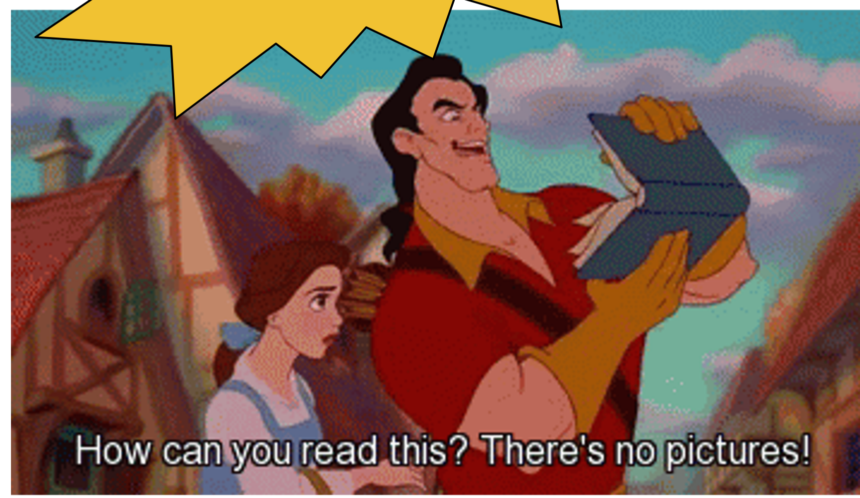
Annotating Your Texts

- **WHAT**

  Intentionality of interacting with a text to enhance the reader's understanding of, recall of, and reaction to the text

- **HOW**
  - Highlighting, underlining or using [brackets] to note key points or ideas

WHY DO WE CARE?!



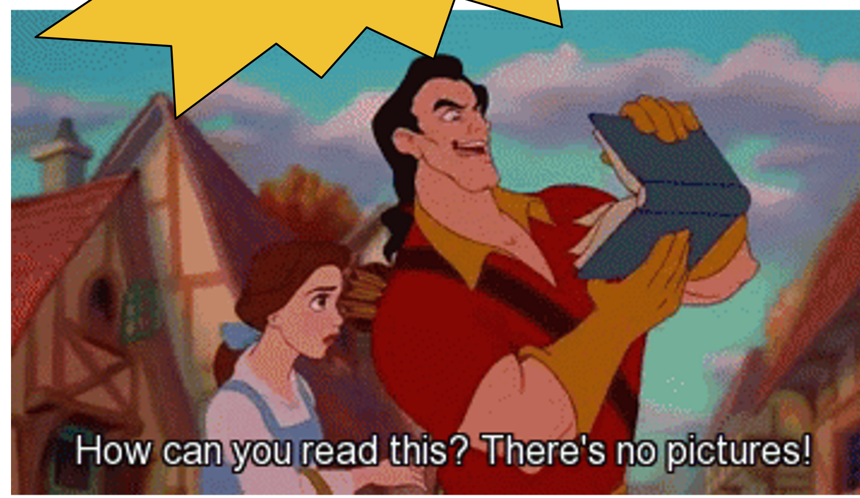
Annotating Your Texts

- **WHAT**

  Intentionality of interacting with a text to enhance the reader's understanding of, recall of, and reaction to the text

- **HOW**
  - Highlighting, underlining or using [brackets] to note key points or ideas
  - Circling unfamiliar words or confusing parts of the text

WHY DO WE CARE?!



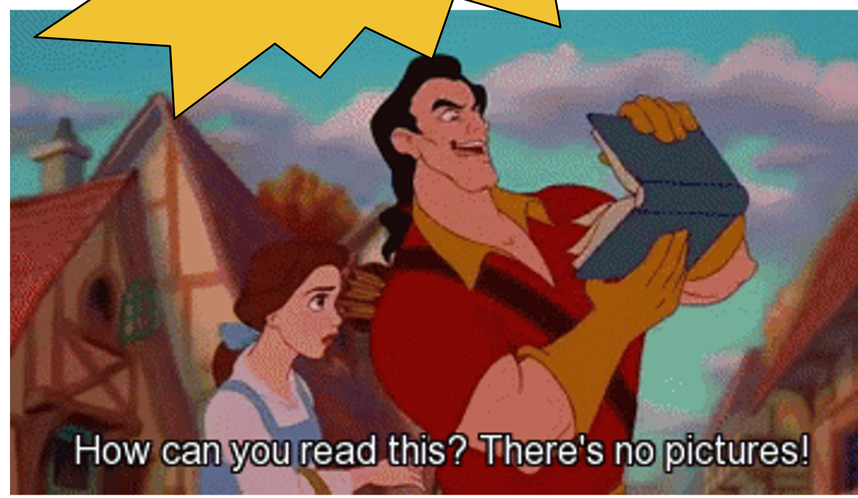
Annotating Your Texts

WHY DO WE CARE?!

- **WHAT**

  Intentionality of interacting with a text to enhance the reader's understanding of, recall of, and reaction to the text

- **HOW**
  - Highlighting, underlining or using [brackets] to note key points or ideas
  - Circling unfamiliar words or confusing parts of the text
  - *Paraphrasing or summarizing passages/chapters/sections*



Annotating Your Texts

- **WHAT**

  Intentionality of interacting with a text to enhance the reader's understanding of, recall of, and reaction to the text

- **HOW**
  - Highlighting, underlining or using [brackets] to note key points or ideas
  - Circling unfamiliar words or confusing parts of the text
  - *Paraphrasing or summarizing passages/chapters/sections*
  - Commenting or reacting to the text 🤯

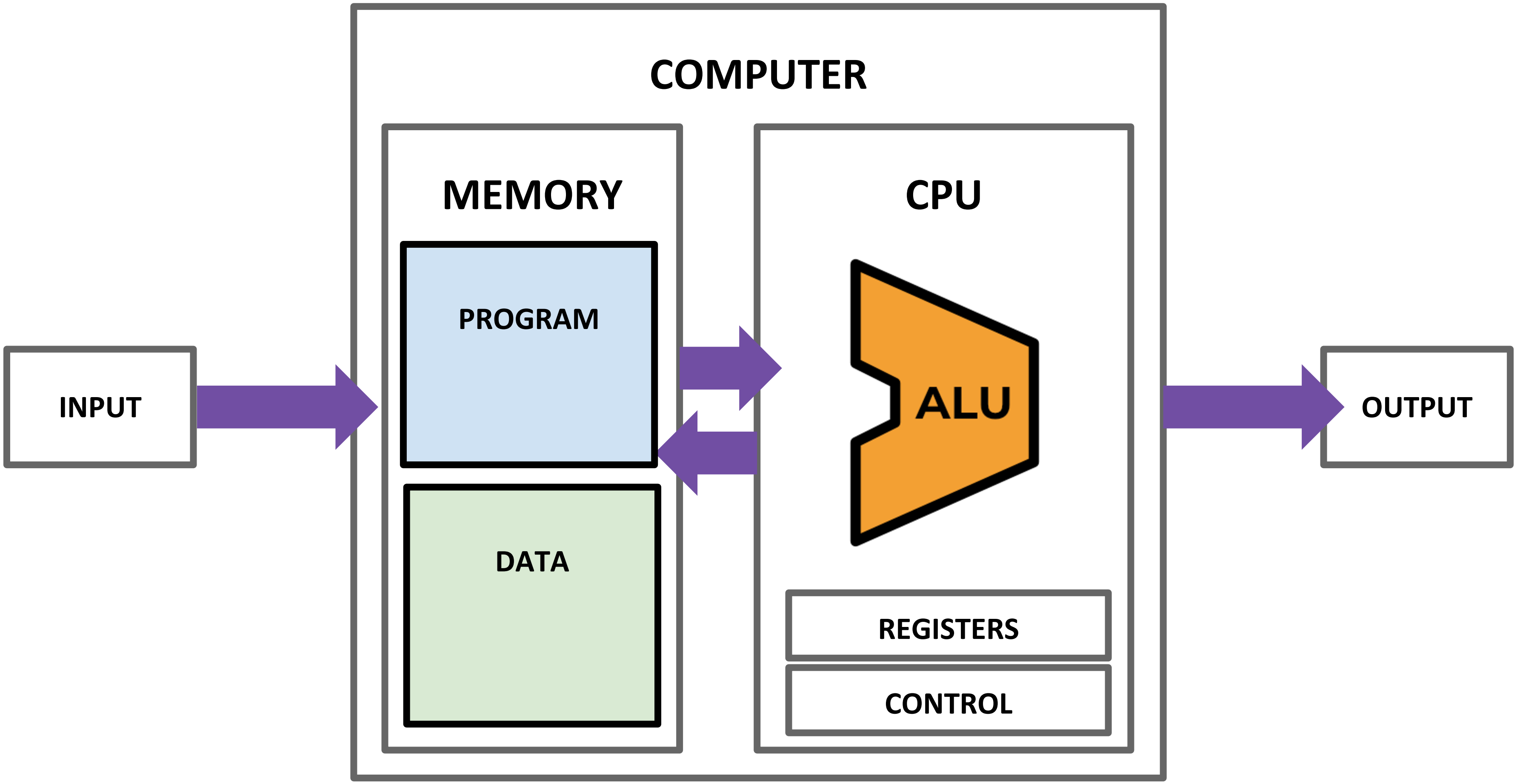
Lecture Outline

- Pulse Check Survey
- Annotation Strategies
- **AMA: Ask Me Anything**
- Reading Review and Q&A
  - Assembly Languages & Machine Code
- Project 4 Overview: Hack Assembly Language
  - Registers, A-Instructions, Symbols, C-Instructions

Lecture Outline

- Pulse Check Survey
- Annotation Strategies
- AMA: Ask Me Anything
- **Reading Review and Q&A**
  - **Assembly Languages & Machine Code**
- Project 4 Overview: Hack Assembly Language
  - Registers, A-Instructions, Symbols, C-Instructions

Revisiting The Von Neumann Architecture



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

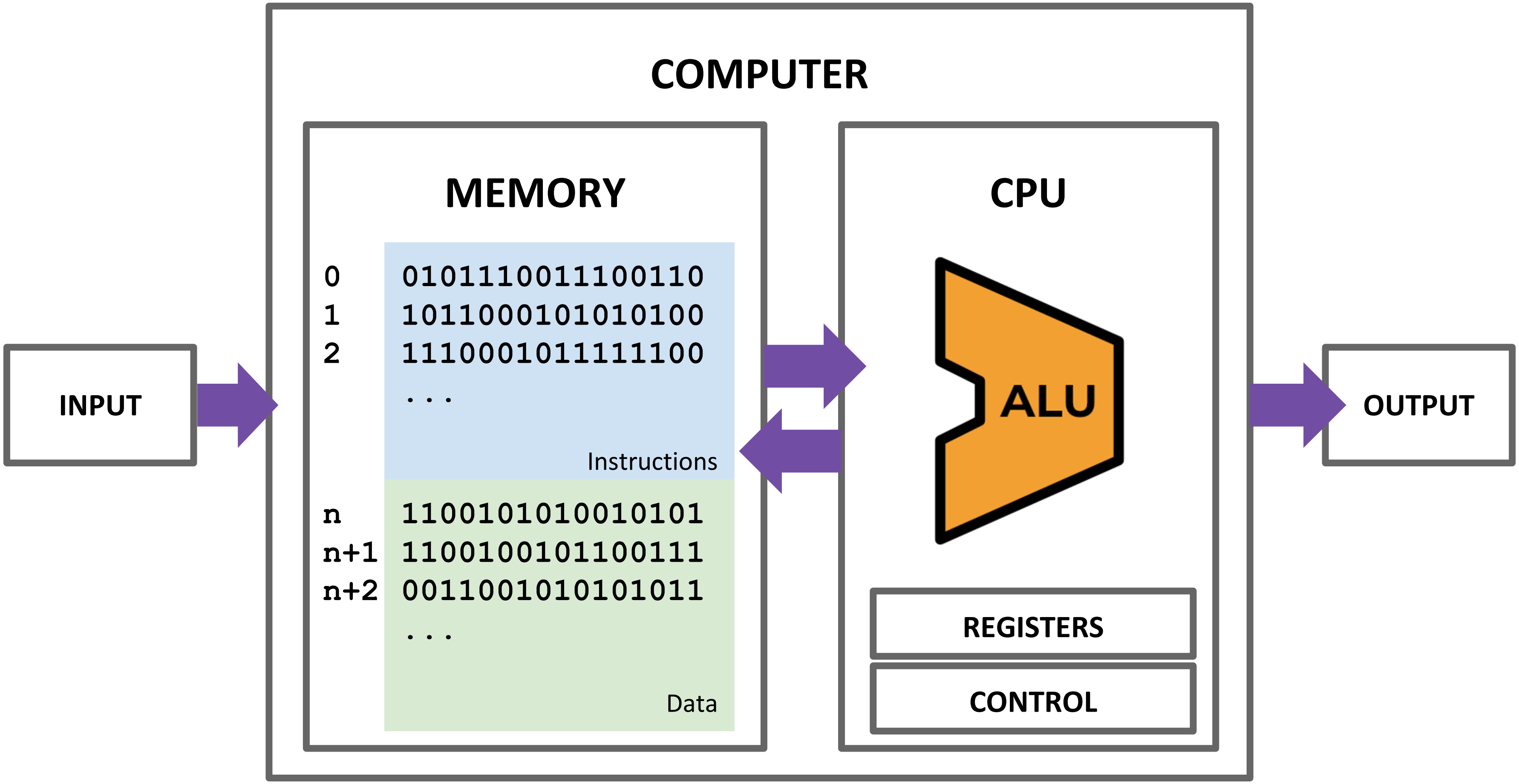
Machine Code

- Instructions are stored in memory, so they must be able to be encoded in binary

- When we refer to **machine code**, we are typically talking about this binary representation of code

- Each instruction is a sequence of 1s and 0s
  - Our computer / hardware specification is what gives meaning to each part of this sequence
  - E.g., Is this an add or subtract instruction? What are the inputs?

Storing the Program



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

Assembly Languages

- Writing code using 1s and 0s is tedious and error prone
- Assembly languages are a human-readable format of binary instructions that a CPU runs
- Each human-readable assembly instruction has a corresponding binary machine code instruction
  - Example: **`addq reg1, reg2 == 0b1011000101010100`**
- Assembly is often used as an intermediary between a high-level programming language and machine code

Producing Machine Code

```
while (i < 100)
{
  sum += arr[i];
  i++;
}
```

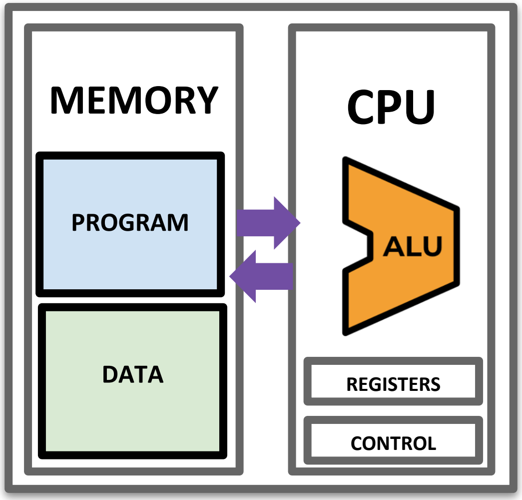
Java

Compile

```
0101110011100110
1011000101010100
1110001011111100
...
```

Machine Code Instructions

Load & Execute



Producing Machine Code

```
while (i < 100)
{
  sum += arr[i];
  i++;
}
```

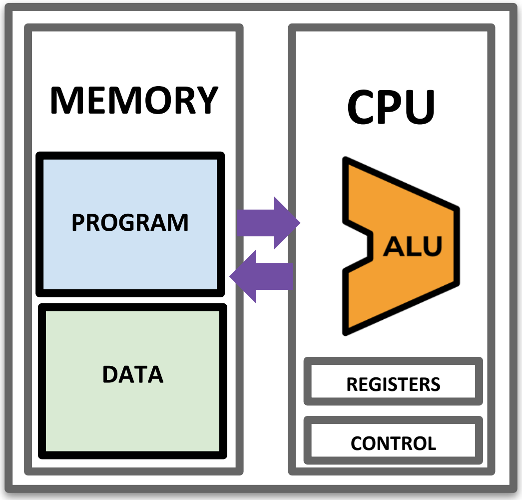
Java

Compile

```
0101110011100110
1011000101010100
1110001011111100
...
```

Machine Code Instructions

Load & Execute




```
movq $5, %rdx
addq %rsx, %rdx
movq %rdx, %rax
ret
```

Assembly Language

Assemble

Producing Machine Code

```
while (i < 100)
{
  sum += arr[i];
  i++;
}
```

Java

Compile

Compile

```
movq $5, %rdx
addq %rsx, %rdx
movq %rdx, %rax
ret
```

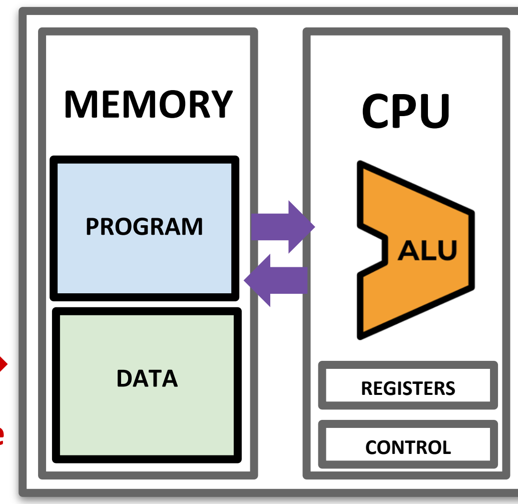
Assembly Language

Assemble

```
0101110011100110
1011000101010100
1110001011111100
...
```

Machine Code Instructions

Load & Execute



Machine Language

- Specification of the Hardware/Software interface
  - What operations are supported?
  - What do they operate on?
  - How is the program controlled?
- Usually in close correspondence with the hardware architecture
  - Different specification for different hardware platforms
- Cost and Performance Tradeoffs
  - Silicon area and complexity
  - Time to complete instruction
  - Power consumption

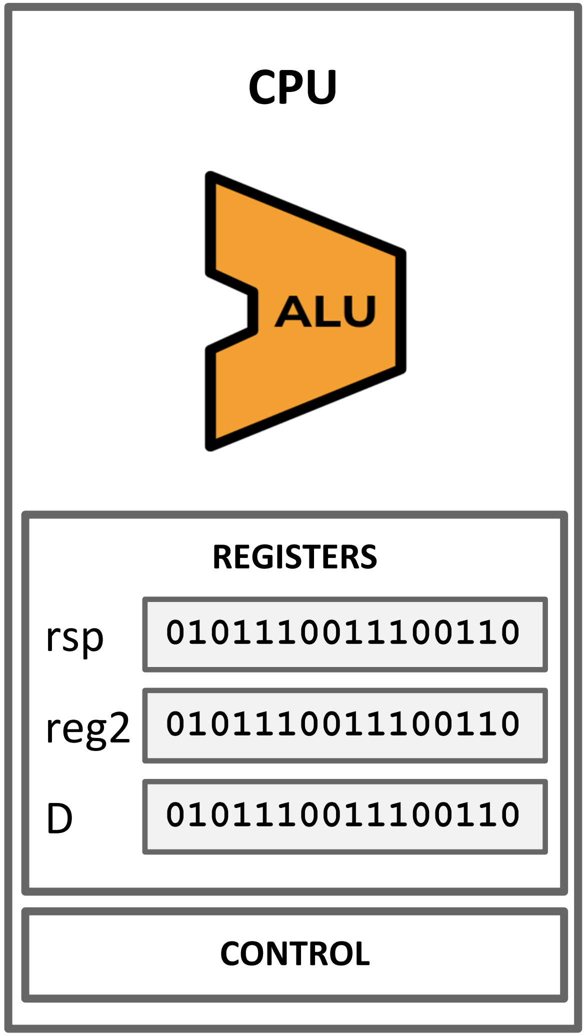
Machine Operations

- Correspond to the operations supported by hardware:
  - Arithmetic (+, –)
  - Logical (And, Or)
  - Flow Control ("go to instruction n", "if (condition) then go to instruction n")

- Differences between machine languages:
  - Instruction set richness (e.g., division? bulk copy?)
  - Data types (e.g., word size, floating point)

Registers

- CPU typically has a small number of **registers**
  - Very efficient to access
  - Used for intermediate, short-term "scratch work"
- Number and use of registers is a central part of any machine language

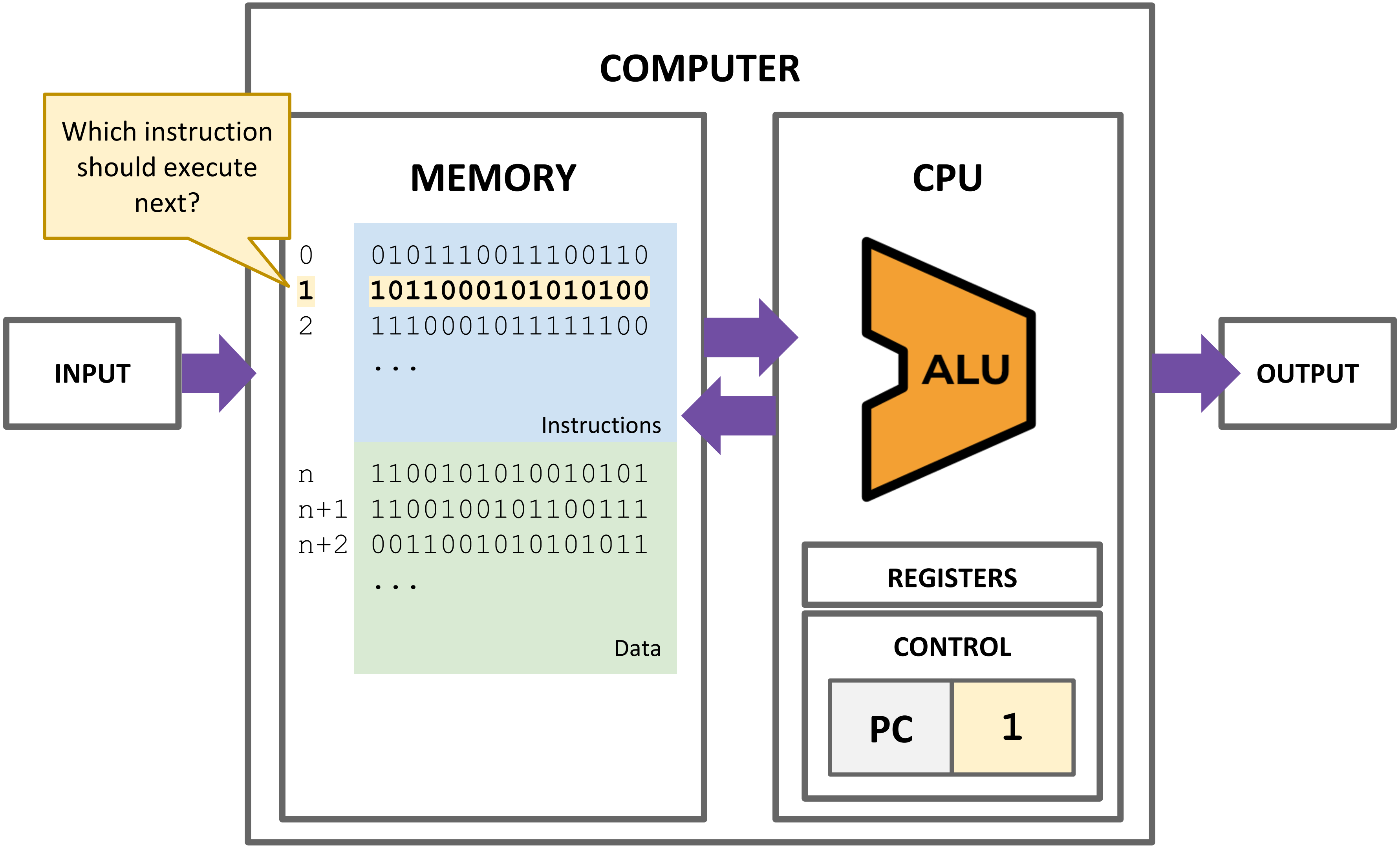


Addressing Modes

- "What locations can I specify in my assembly code?"

- Some useful options:
  - Register
    - `add reg1, reg2` ← Register names
  - Direct Memory Access
    - `add reg1, Memory[200]` ← Access the giant array (that is, memory) at index 200
  - Indirect Memory Access
    - `add reg1, Memory[reg2]`
  - Immediate
    - `add 100, reg2`

Flow Control



Flow Control

- Usually the CPU just executes machine instructions in a sequence
  - Typically moves to the instruction with the next highest address

- Sometimes we want to always “jump” to another location
  - Example: At the end of an infinite loop

High Level Code (similar to Java)	Assembly Code
while (true) { **reg1++;** *<more loop body>* **}** *<code after loop>*	**TOP:** **add 1, reg1** *<more loop body>* **jmp TOP** *<code after loop>*

Flow Control

- Usually the CPU just executes machine instructions in a sequence
  - Typically moves to the instruction with the next highest address
- Sometimes we want to "jump" only if a condition is met
  - Example: At the condition of an if statement

High Level Code (similar to Java)	Assembly Code

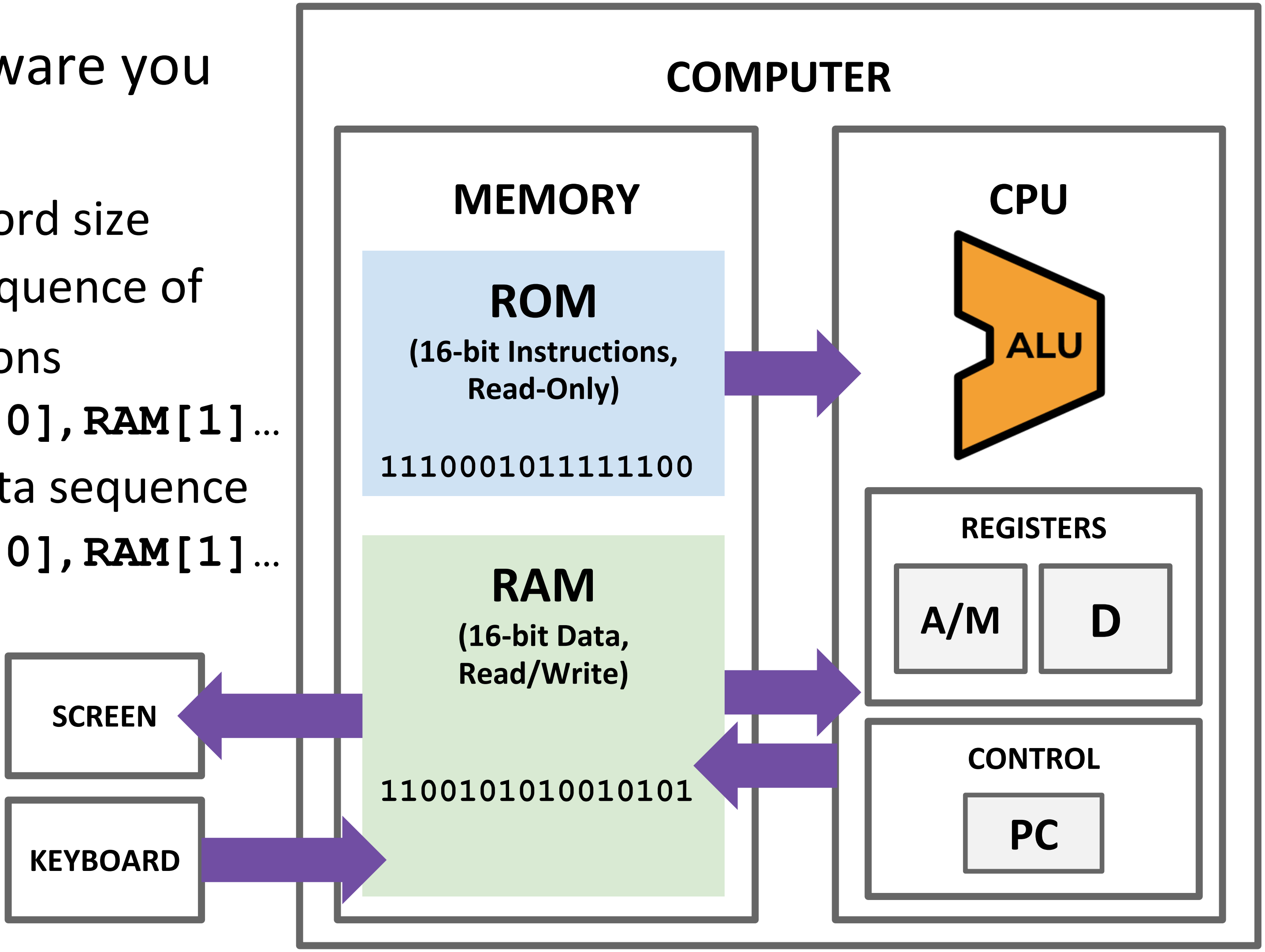
`if (reg1 < reg2) {`
`    reg1++;`
`}`
`reg2++;`
	`    cmp reg1, reg2`
	`    jge SKIP`
	`    add 1, reg1`
	`SKIP:`
	`    add 1, reg2`

Lecture Outline

- Pulse Check Survey
- Annotation Strategies
- AMA: Ask Me Anything
- Reading Review and Q&A
  - Assembly Languages & Machine Code
- **Project 4 Overview: Hack Assembly Language**
  - **Registers, A-Instructions, Symbols, C-Instructions**

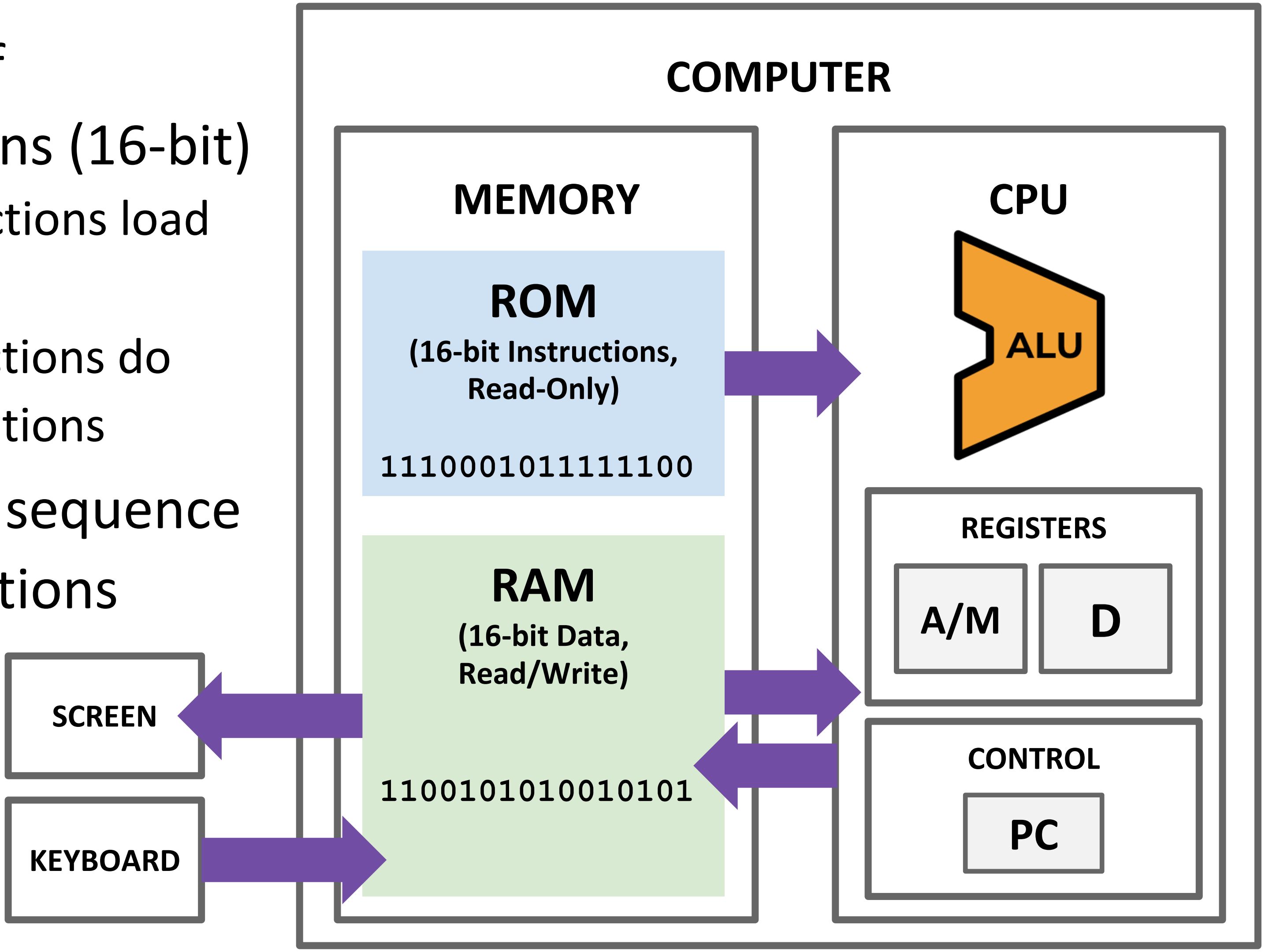
The Hack Computer

- The hardware you will build
  - 16-bit word size
  - ROM: sequence of instructions
    - **ROM[0],RAM[1]**…
  - RAM: data sequence
    - **RAM[0],RAM[1]**…



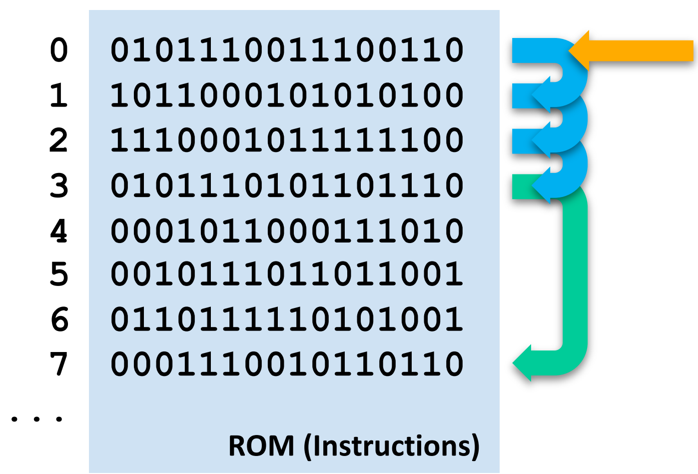
The Hack Machine Language

- 2 types of instructions (16-bit)
  - A-instructions load data
  - C-instructions do computations
- Program: sequence of instructions



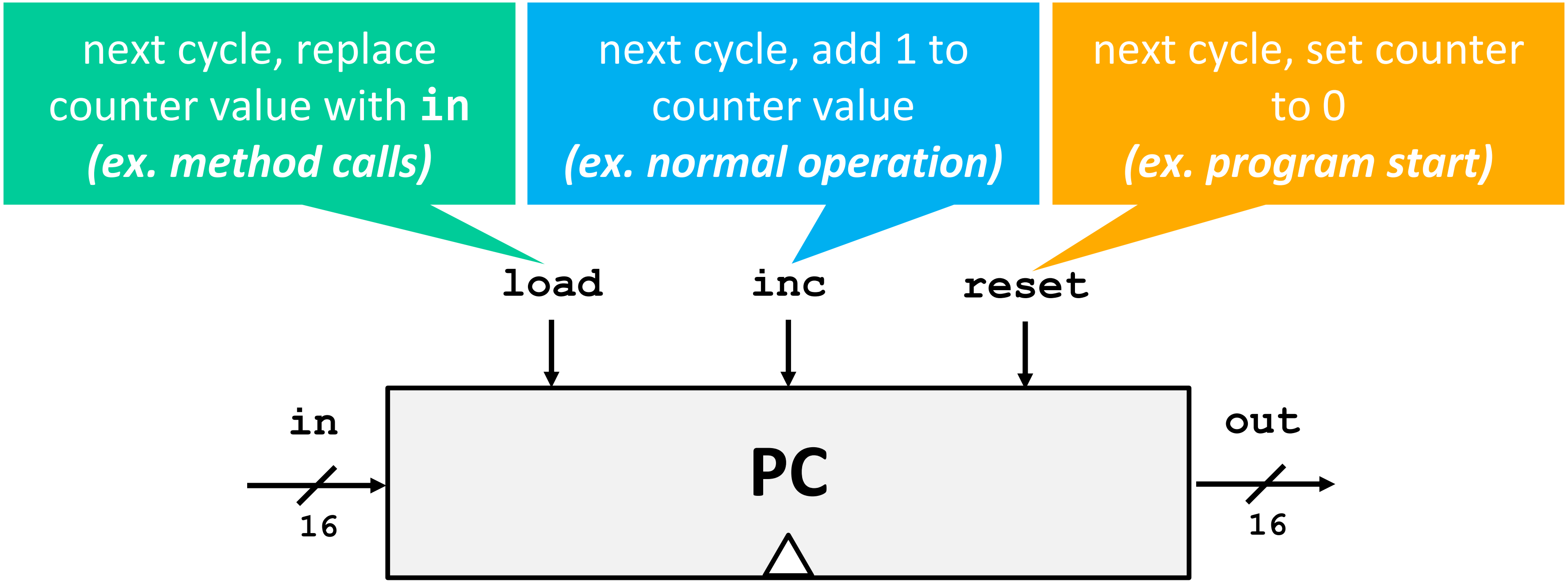
Hack: Control Flow

- Startup
  - Hack instructions loaded into ROM
  - Reset signal initializes computer state (instruction 0)
- Execution
  - Usually, advance to next instruction each cycle
  - On jump instruction, write a different address into the PC



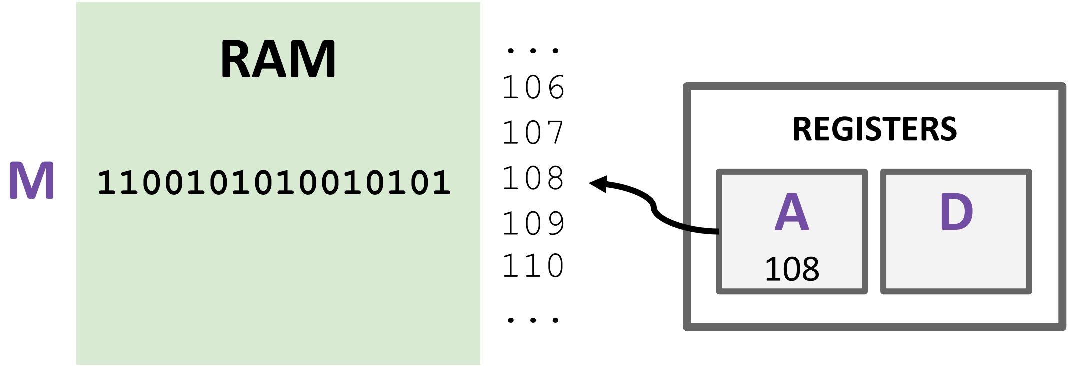
Program Counter (PC)

- Keeps track of what instruction we are executing
  - If the PC outputs 24, on the next clock cycle the computer runs the instruction at address 24 in the code segment



Hack: Registers

- D Register: For storing data
- A Register: For storing data *and* addressing memory
- M "Register": The 16-bit word of memory currently being referenced by the address in A



Hack: A-Instructions

- Syntax: `@value`

- `value` can either be:
  - A non-negative decimal constant
  - A symbol referring to a constant

- Semantics:
  - Stores `value` in the A register

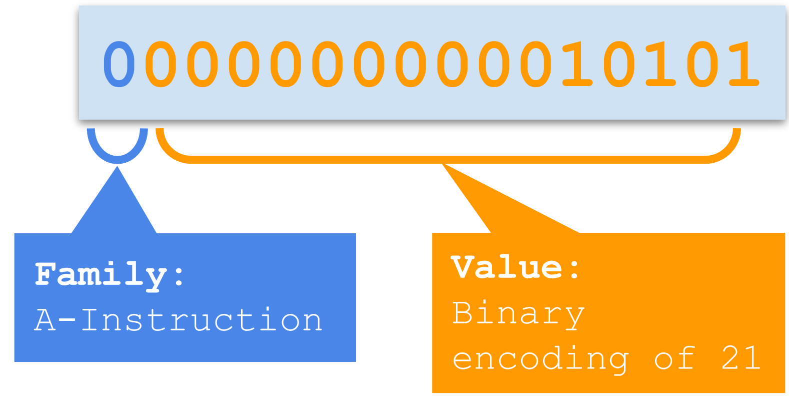
Hack: A-Instructions

- Symbolic Syntax

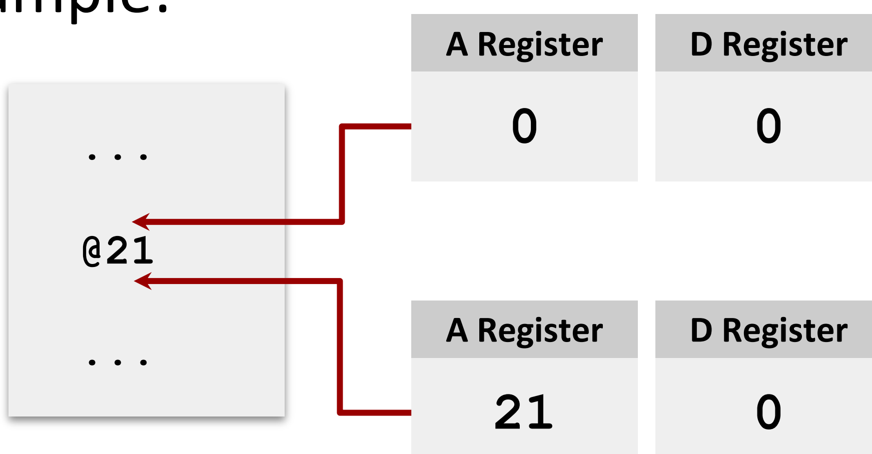
```
@value
```

  - Loads a value into the A register

- Binary Syntax



- Example:



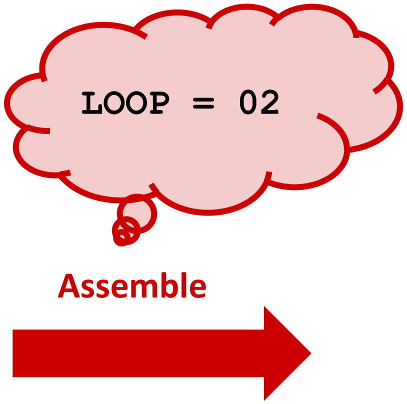
Hack: Symbols

- Symbols are simply an <u>alias</u> for some address
  - Only in the symbolic code—don't turn into a binary instruction
  - Assembler converts use of that symbol to its value instead

- Example:

```
00     @3
01     D=0
     (LOOP)
02     @21
03     D=1
04     @LOOP

       ...
```

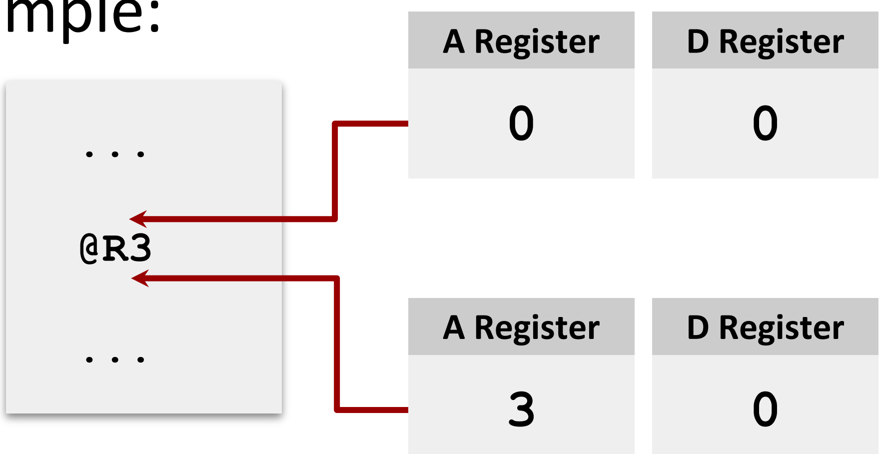



```
00   0000000000000011
01   1110101010010000
02   0000000000010101
03   1110111111010000
04   0000000000000010

     ...
```

Hack: Built-In Symbols

- Using `( )` defines a symbol in ROM / Instructions
- Assembler knows a few built-in symbols in RAM/Data
- `R0, R1, ..., R15`: Correspond to addresses at the very beginning of RAM (0, 1, …, 15)
  - "Virtual registers," Useful to store variables
- `SCREEN, KBD`: Base of I/O Memory Maps
- Example:



Hack: C-Instructions

- Syntax: `dest = comp ; jump` (dest a jump is optional)
  - **`dest`** is a combination of destination registers:

    `M, D, MD, A, AM, AD, AMD`
  - **`comp`** is a computation:

    `0, 1, -1, D, A, !D, !A, -D, -A, D+1, A+1, D-1, A-1, D+A, D-A, A-D, D&A, D|A, M, !M, -M, M+1, M-1, D+M, D-M, M-D, D&M, D|M`
  - **`jump`** is an unconditional or conditional jump:

    `JGT, JEQ, JGE, JLT, JNE, JLE, JMP`
- Semantics:
  - Computes value of **`comp`**
  - Stores results in **`dest`** (if specified)
  - If **`jump`** is specified and condition is true (by testing **`comp`** result), jump to instruction **`ROM[A]`**

Hack: C-Instructions

- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Family: C-Instruction

Unused

Comp: ALU Operation (a bit chooses between A and M)

Dest: Where to store result

Jump: Condition for jumping

Hack: C-Instructions

- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Jump: Condition for jumping

Chapter 4

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

Hack: C-Instructions

- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Dest: Where to store result

Chapter 4

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

Hack: C-Instructions

- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Comp: ALU Operation (a bit chooses between A and M)

Chapter 4

(when a=0) *comp mnemonic*	c1	c2	c3	c4	c5	c6	(when a=1) *comp mnemonic*
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Important: just pattern matching text!

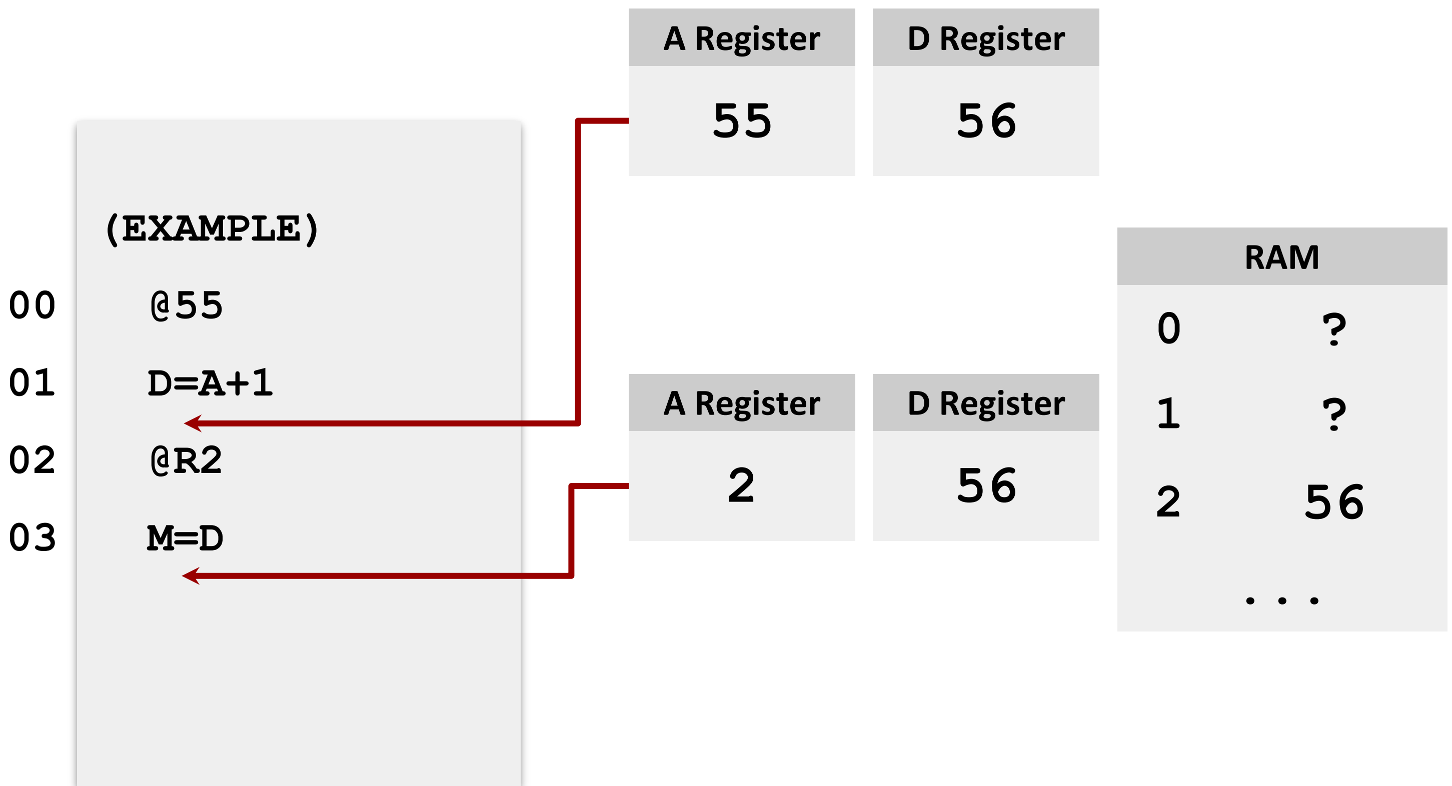
Can't do "**1+M**"

Hack: C-Instructions Example

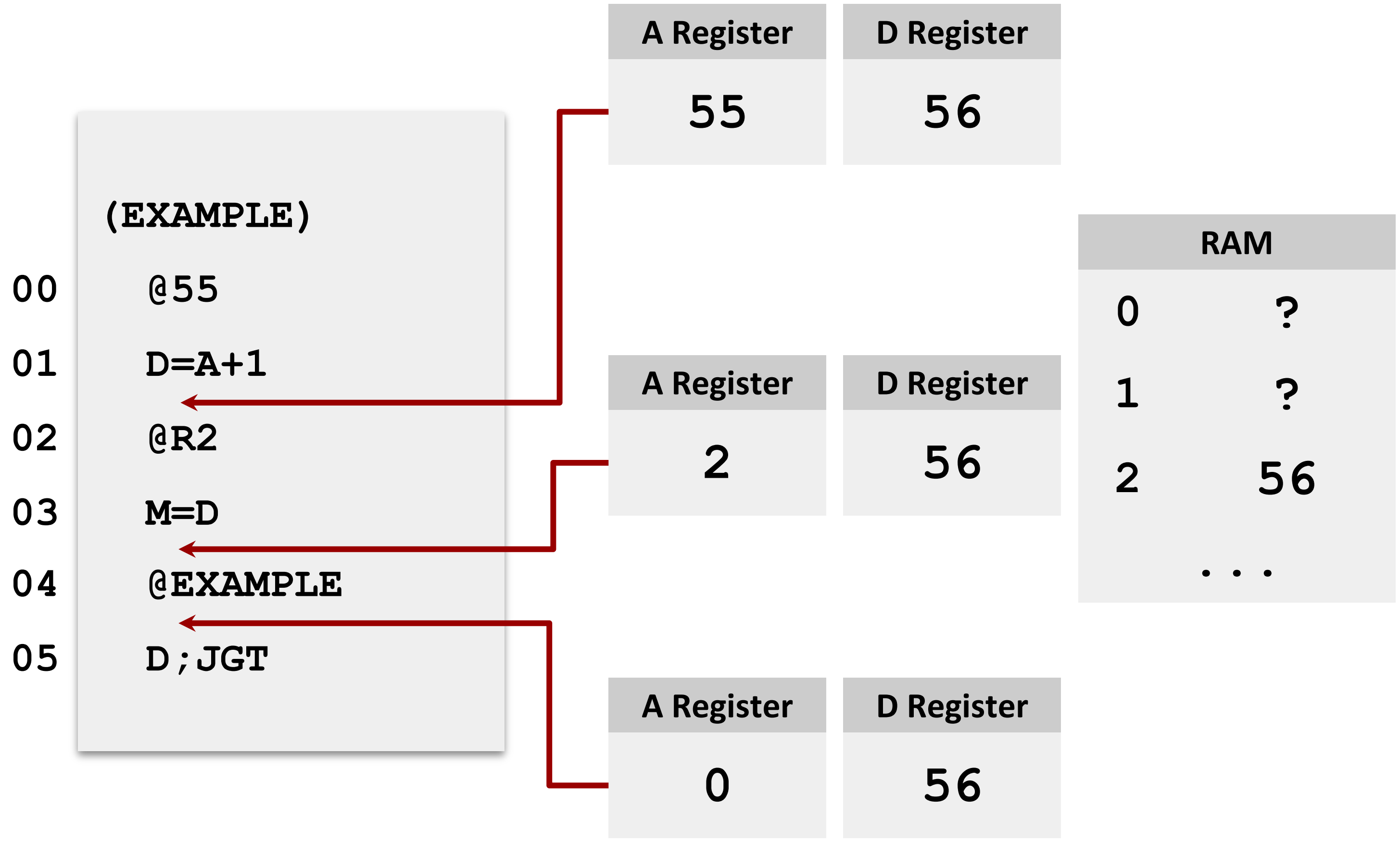
A Register	D Register
55	56

```
   (EXAMPLE)
00   @55
01   D=A+1
```

Hack: C-Instructions



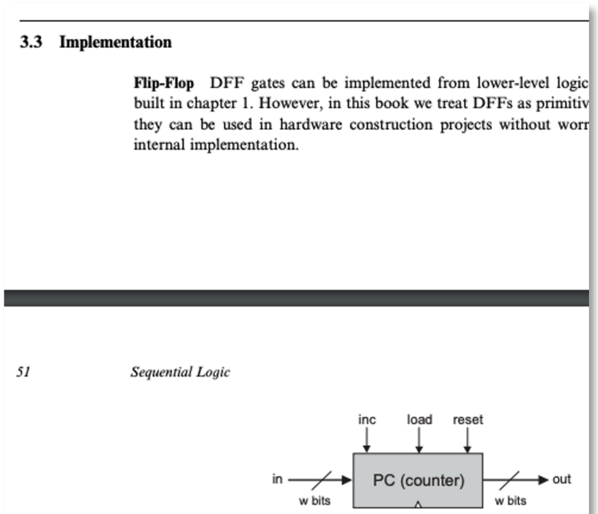
Hack: C-Instructions



(will jump to instruction 0, since D > 0)

Project 3 Resources

- Take advantage of hints in Project 3 readings

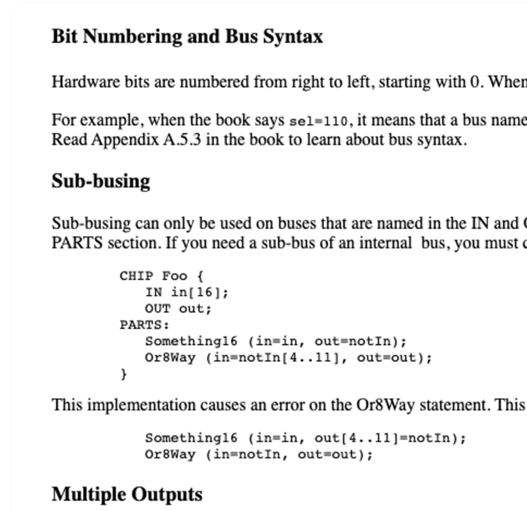


3.3 **Implementation**

Flip-Flop DFF gates can be implemented from lower-level logic built in chapter 1. However, in this book we treat DFFs as primitiv they can be used in hardware construction projects without worr internal implementation.

51 *Sequential Logic*

[Chapter 3]



Bit Numbering and Bus Syntax

Hardware bits are numbered from right to left, starting with 0. When a

For example, when the book says `sel=110`, it means that a bus named Read Appendix A.5.3 in the book to learn about bus syntax.

Sub-busing

Sub-busing can only be used on buses that are named in the IN and O PARTS section. If you need a sub-bus of an internal bus, you must cr

```
CHIP Foo {
   IN in[16];
   OUT out;
PARTS:
   Something16 (in=in, out=notIn);
   Or8Way (in=notIn[4..11], out=out);
}
```

This implementation causes an error on the Or8Way statement. This n

```
   Something16 (in=in, out[4..11]=notIn);
   Or8Way (in=notIn, out=out);
```

Multiple Outputs

[HDL Survival Guide]

- Reach out to your classmates and course staff
  - Post on discussion board: https://edstem.org/us/courses/16521/discussion/
  - Come to office hours: Today after lecture at 3pm!

Post-Lecture 7 Reminders

- Course Staff Office Hours
  - Eric: Tuesdays 3-4pm (hybrid) and Wednesdays 4:30-5:30pm (new, virtual)
  - Leslie: Wednesdays 4:30-5pm (hybrid)
  - Audrey and Sean: Wednesdays 1:30-2:30pm (hybrid)
- Project 3: Memory & Cornell Note-Taking
  - **Due this Thursday (1/27) at 11:59PM PST**
- Starting January 31st (next week), CSE 390B will be fully in-person
  - Please contact the CSE 390B course staff prior to class if you have a situation impacting your ability to attend class in-person