CSE 390B, Spring 2022

Building Academic Success Through Bottom-Up Computing

Cornell Notes & Memory

Cornell Note Taking Discussion, Storing Data: Bit, Representing and Building Memory, Program Counter Overview

Project 3 Check-in

- How has Project 3 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
- **Double-check** your submission on GitLab
  - Navigate to GitLab, open tags, and verify that the associated commit includes your expected changes
  - Guide for untagging on GitLab on the Ed discussion board

Lecture Outline

- **Cornell Note Taking Discussion**
  - **Compare and Contrast Cornell Notes**
- Storing Data: Bit
  - Bit Overview and Implementation
- Representing and Building Memory
  - Array Abstraction, Reading and Writing
  - Building From the Bit
- Program Counter (PC) Overview
  - Control Flow of Computer Programs

Cornell Note Taking Discussion

- In small groups, compare and contrast your Cornell Notes from Tuesday's lecture
  - What are some of the key points you wrote in your summary?
  - What were some of the questions you came up with?
  - What are you still left feeling confused/uncertain about after Tuesday's lecture?

Lecture Outline

- Cornell Note Taking Discussion
  - Compare and Contrast Cornell Notes
- **Storing Data: Bit**
  - **Bit Overview and Implementation**
- Representing and Building Memory
  - Array Abstraction, Reading and Writing
  - Building From the Bit
- Program Counter (PC) Overview
  - Control Flow of Computer Programs

Poll Everywhere

Vote at https://pollev.com/cse390b

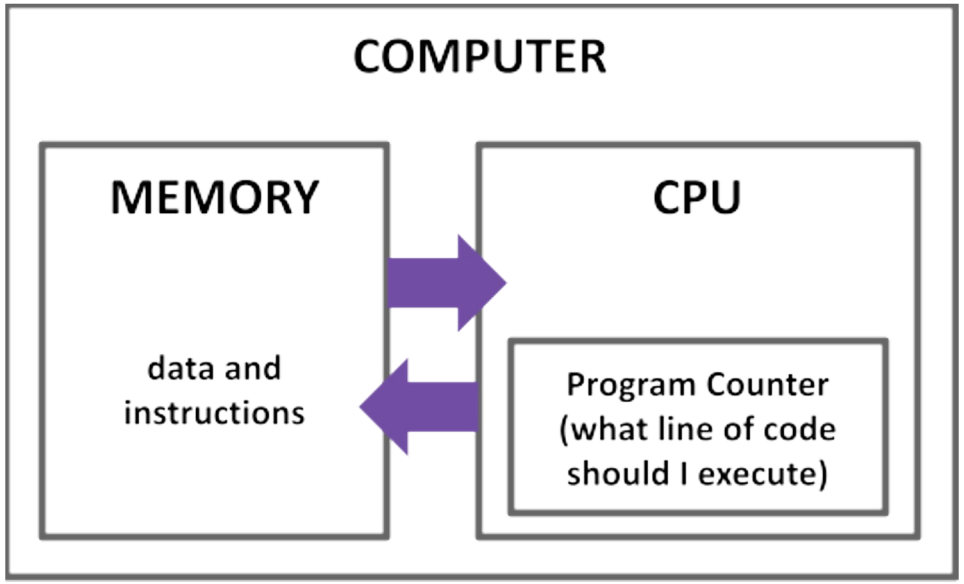
Memory can be though of as which data structure? How is memory similar to this data structure? Different?

A. Map
B. Array
C. Queue
D. Tree
E. We're lost…

Computer Overview

- CPU is the "brain" of our computer
  - Does necessary computations (add, subtract, multiply, etc.)

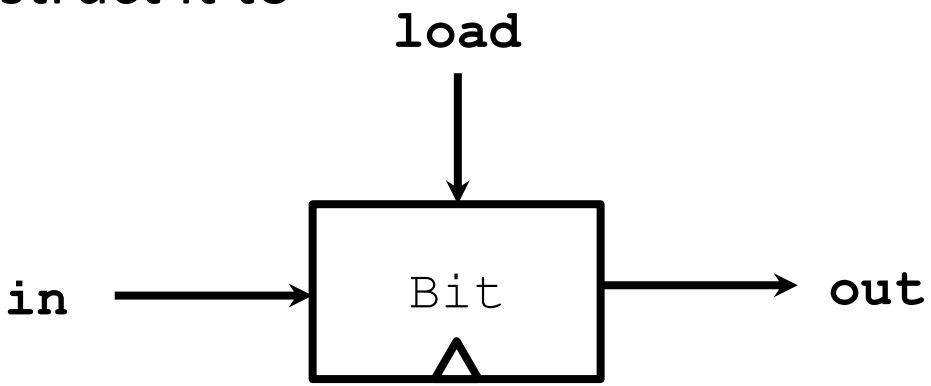
- Memory is used to store values for later use
  - Requires persistence across multiple computations
  - Needs to change values at our discretion



Storing Data: Bit

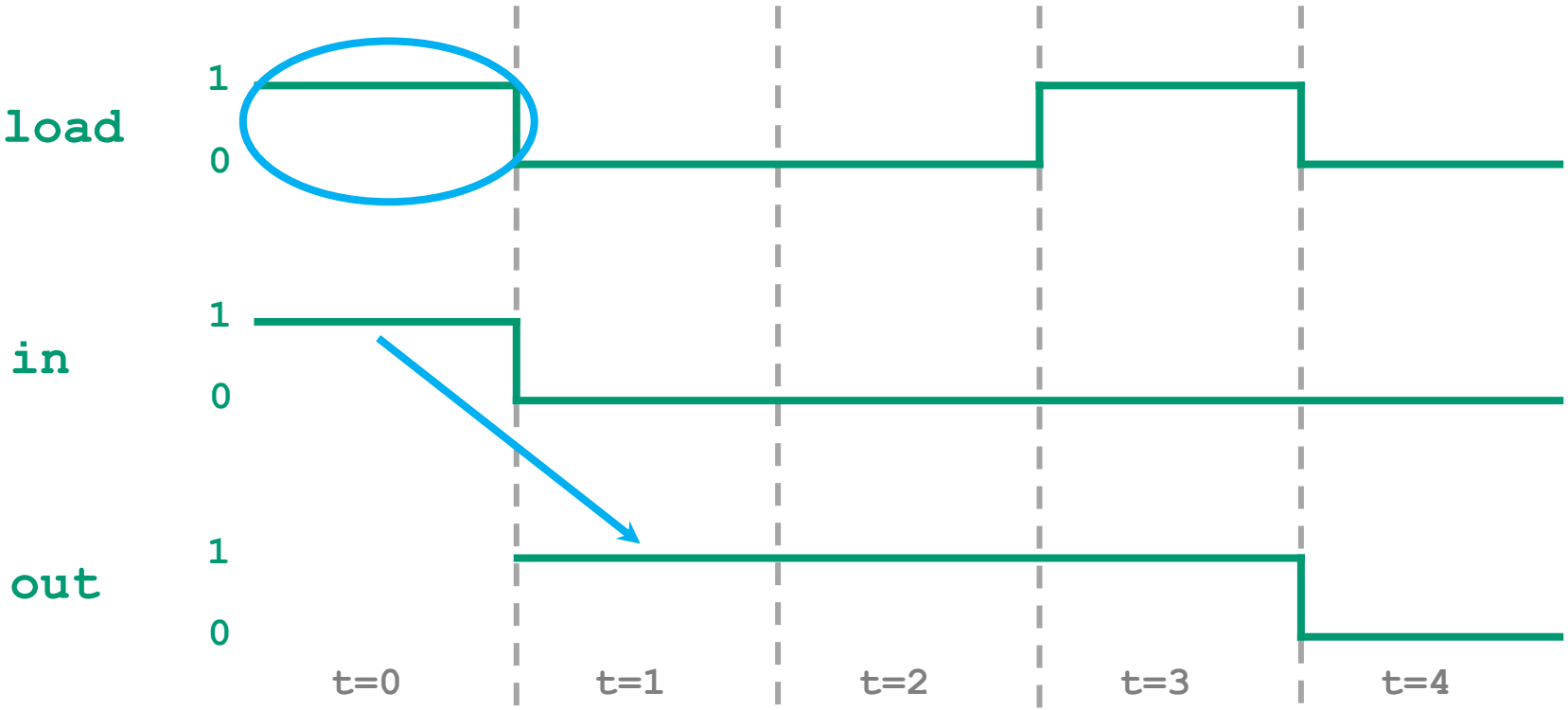
- A Flip-Flop changes state *every* clock cycle

- We will build the abstraction of a "Bit" that only changes when we instruct it to

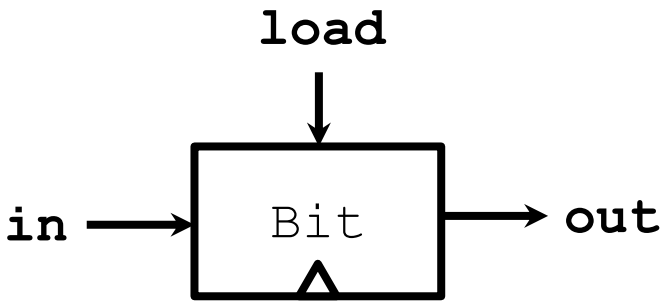



```
if load(t-1)     out(t) = in(t-1)
else             out(t) = out(t-1)
```

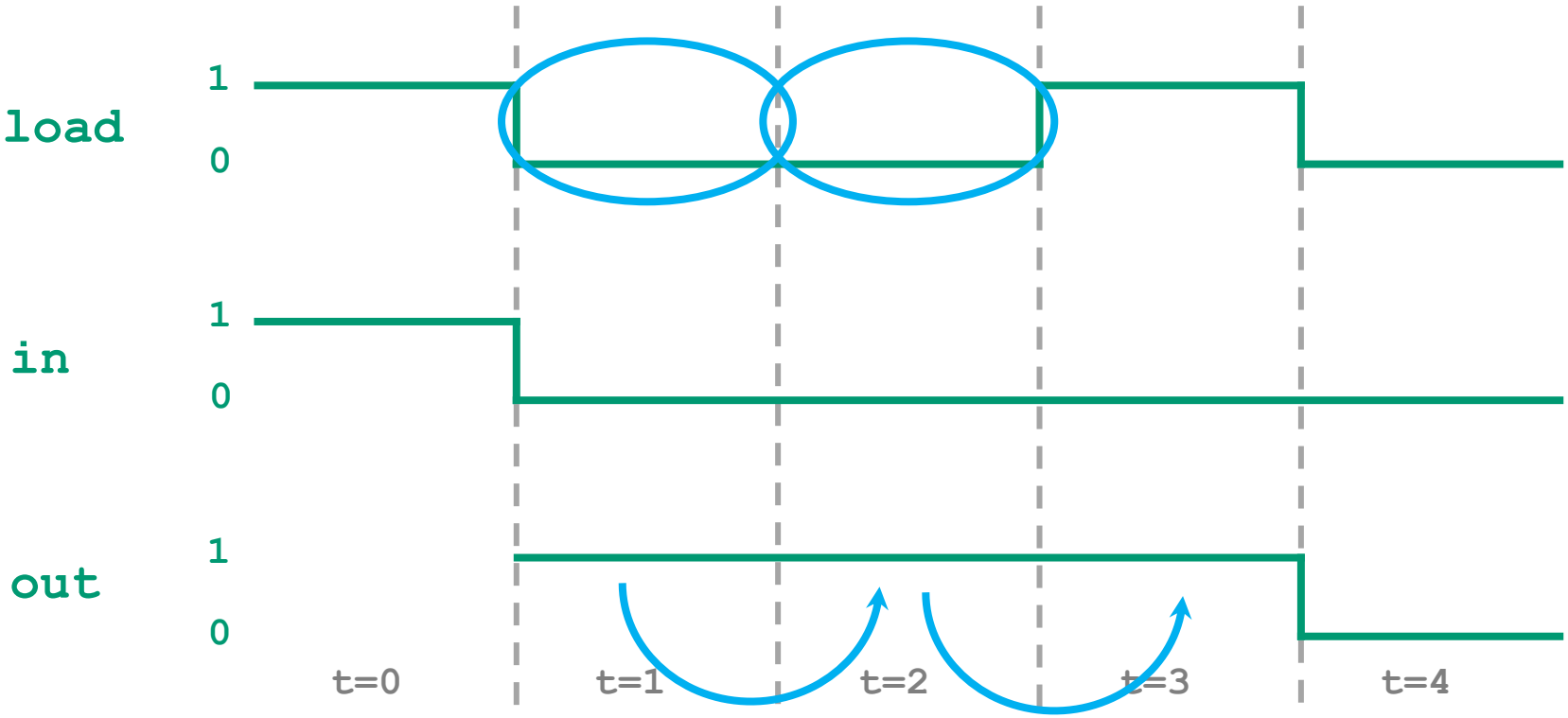
Bit Behavior



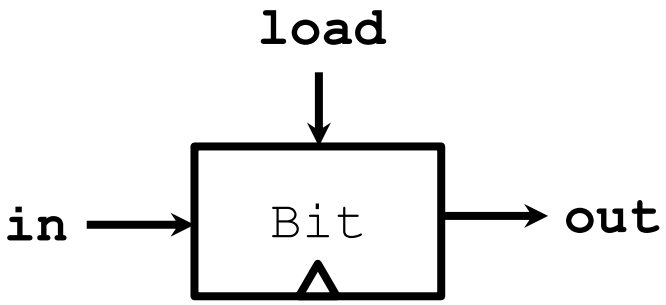

```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```



Bit Behavior




```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```



Bit Time Series

- Bit Specification:

```
if (load(t-1)): out(t) = in(t-1)
else:           out(t) = out(t-1)
```

load	1	0	0	1	1	1	0	...
in	1	0	0	0	1	0	1	...
out	0	1	1	1	0	1	0	...
time	**t=0**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=6**	...

Example 1: `load(t=0) == 1` so `out(t=1) = in(t=0)`

Bit Time Series

- Bit Specification:

```
if (load(t-1)): out(t) = in(t-1)
else:           out(t) = out(t-1)
```

load	1	0	0	1	1	1	0	...
in	1	0	0	0	1	0	1	...
out	0	1	1	1	0	1	0	...
time	**t=0**	**t=1**	**t=2**	**t=3**	**t=4**	**t=5**	**t=6**	...

Example 1: `load(t=0)` == 1 so `out(t=1)` = `in(t=0)`

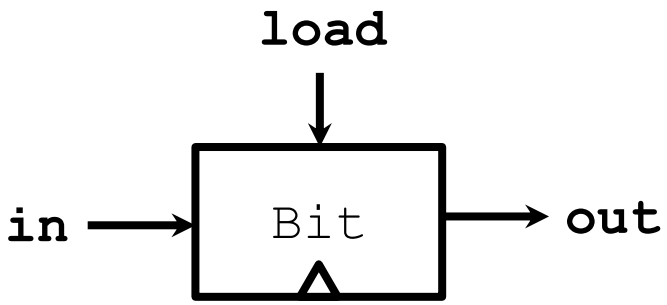
Example 2: `load(t=2)` == 0 so `out(t=3)` = `out(t=2)`

Poll Everywhere

Vote at https://pollev.com/cse390b

Which gates will we need to implement a Bit? Select all that apply.

A. Mux
B. Xor
C. And
D. DFF
E. We're lost…




```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```

Implementing a Bit

- Bit Specification:

```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```

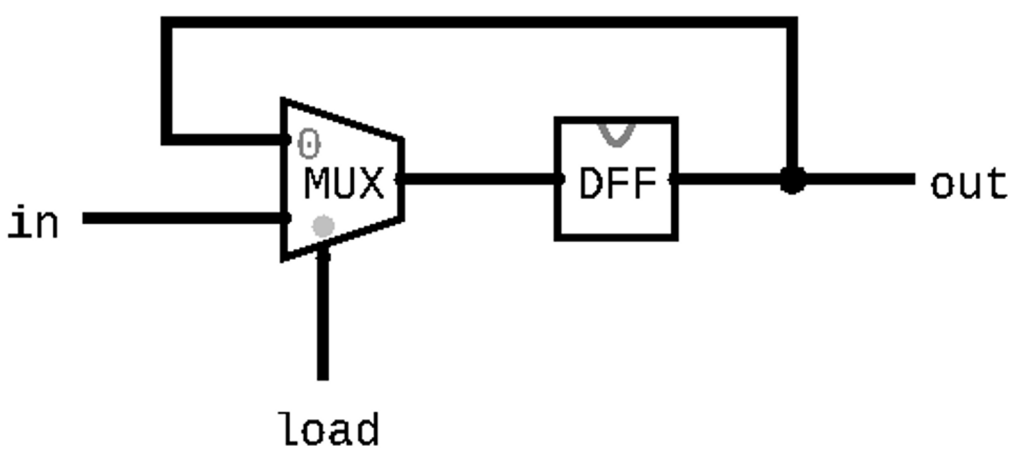
- Exercise: fill in the connections to the gates to create a circuit diagram of Bit

Implementing a Bit

- Bit Specification:

```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```

- Exercise: fill in the connections to the gates to create a circuit diagram of Bit



Lecture Outline

- Cornell Note Taking Discussion
  - Compare and Contrast Cornell Notes
- Storing Data: Bit
  - Bit Overview and Implementation
- **Representing and Building Memory**
  - **Array Abstraction, Reading and Writing**
  - Building From the Bit
- Program Counter (PC) Overview
  - Control Flow of Computer Programs

Memory Representation

- Memory can be abstracted as one huge array

- **Addresses** are indices into different memory slots
  - The width of an address is fixed for the system
  - The nand2tetris project will use 16-bit addresses

- Each **value** in memory takes up a fixed width
  - Not the same as address width
  - The nand2tetris project uses 16-bit slots (values) in memory

Memory Representation

- Read and write to memory by specifying an address
  - More details next week

- Example: `x = memory[01...00]`
  - Reads the value in memory at address `01...00` and stores it in `x`

- Example: `memory[01...00] = 7`
  - Writes the value 7 in the memory slot at address `01...00`

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.

Lecture Outline

- Cornell Note Taking Discussion
  - Compare and Contrast Cornell Notes
- Storing Data: Bit
  - Bit Overview and Implementation
- **Representing and Building Memory**
  - Array Abstraction, Reading and Writing
  - **Building From the Bit**
- Program Counter (PC) Overview
  - Control Flow of Computer Programs

Building Memory: Register

- Bits store a single value (0 or 1)
  - In memory, we need to store 16-bit values

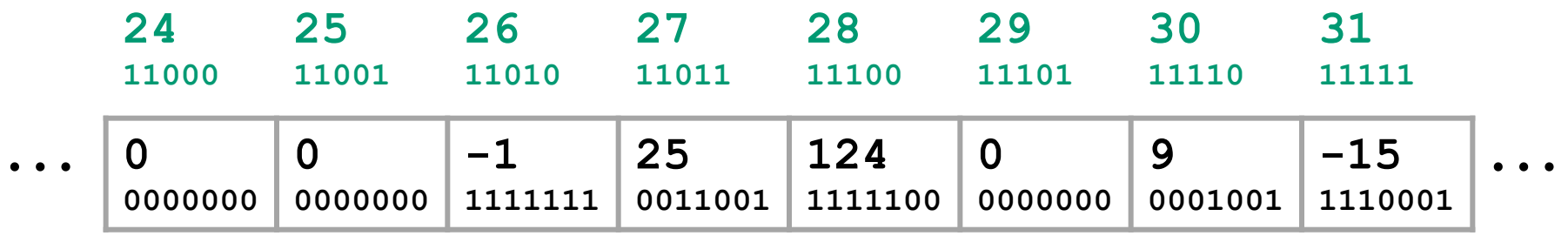
- Registers are conceptually the same as a Bit
  - Allows us to store and change 16-bit values
  - Groups together 16 individual bits that share a load signal

```
// if (load(t-1)): out(t) = in(t-1)
//              else: out(t) = out(t-1)
CHIP Register {
      IN in[16], load;
      OUT out[16];
      ...
}
```

RAM: Random Access Memory

- Abstraction of Computer Memory: just a giant array

- Goal: create hardware that can provide that abstraction

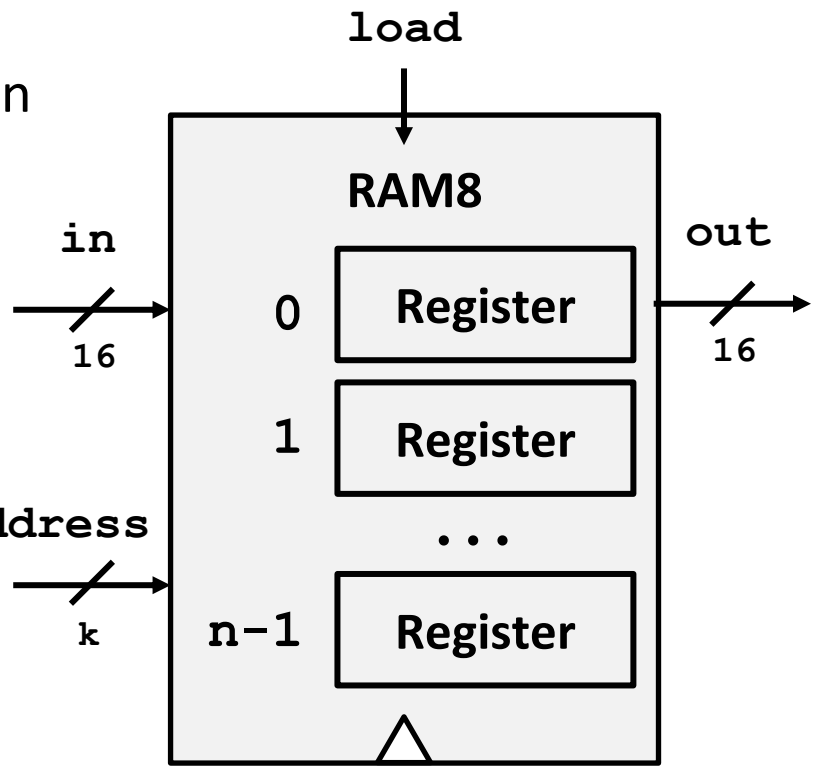


- Key attribute of arrays: “random access” lets us index into them at any point

```
memory[26] = -1;
```

Building Memory: RAM8 From Registers

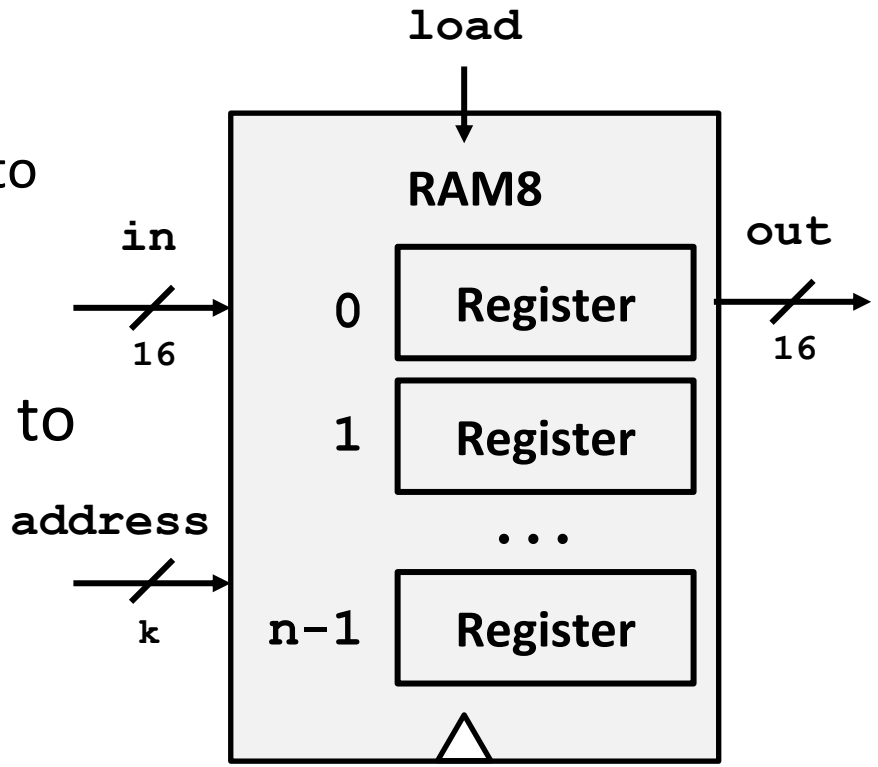
- RAM interface:
  - **`load`**: if 1, then in should be written to specified memory slot
  - **`in`**: 16-bit input used to update specified memory slot if load is 1
  - **`address`**: address used to specify memory slot
  - **`out`**: 16-bit output from the slot specified by address



- RAM8 can be built from 8 registers
  - address width is $\log_2(8) = 3$ bits

Building Memory: RAM8 From Registers

- Step 1: Route **`in`** to every register
  - We don't want to update every register, however
  - Solution: choose which register to enable with **`address`**
- Step 2: Choose which register to use for the output
- When we think about making *choices* in hardware, we want to think about Mux and DMux



Building Memory: The Rest of RAM

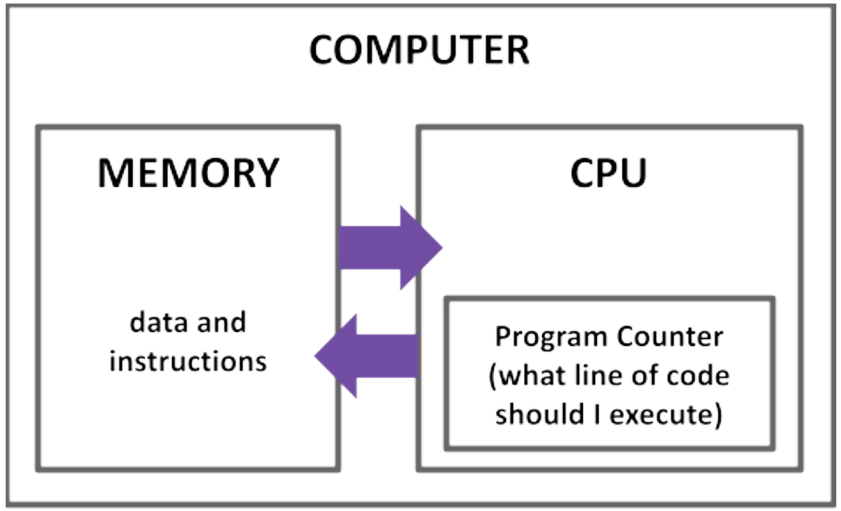
- After RAM8, can build larger RAM chips from a combination of smaller RAM chips
  - For example, RAM64 can be built using eight RAM8 chips
- Technique is similar to RAM8 but will have to use different portions of the address
- The blocks section of the reading will be helpful
  - For example, can think of each RAM8 as a block of RAM64

Lecture Outline

- Cornell Note Taking Discussion
  - Compare and Contrast Cornell Notes
- Storing Data: Bit
  - Bit Overview and Implementation
- Representing and Building Memory
  - Array Abstraction, Reading and Writing
  - Building From the Bit
- **Program Counter (PC) Overview**
  - **Control Flow of Computer Programs**

Program Counter (PC)

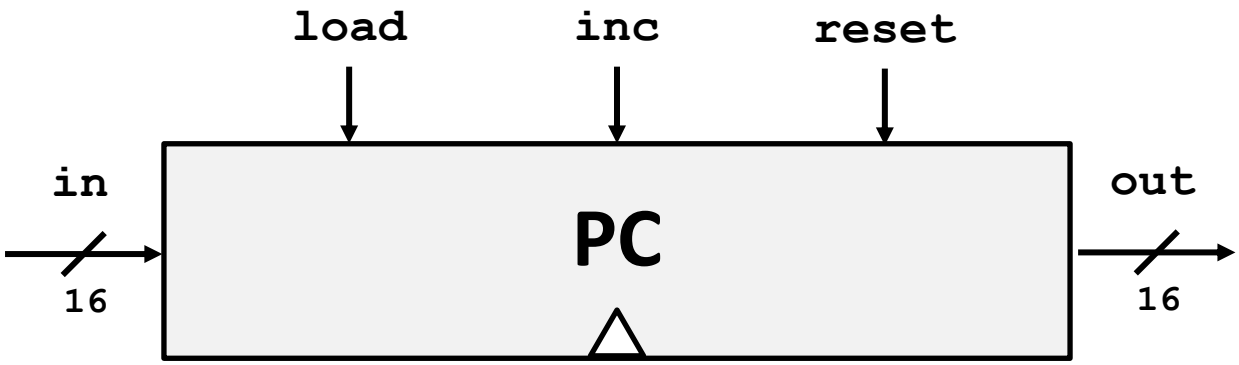
- Memory is used to store data as well as code
- Instructions and operations are stored at different addresses in memory
- Program Counter in the CPU keeps track of which address contains the instruction that should be executed next



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
- Program counter specification:

```
if      (reset[t] == 1) out[t+1] = 0
else if (load[t] == 1)  out[t+1] = in[t]
else if (inc[t] == 1)   out[t+1] = out[t] + 1
else                    out[t+1] = out[t]
```



Project 4 Overview

- Part I: Cornell Note Taking
  - Practice taking detailed notes in another class
  - Think critically about the technique
- Part II: Building Memory
  - Memory & Sequential Logic: Build our first sequential chips, from a 1-bit register to a 16K RAM module
  - Program Counter: Build counter that tracks where we are in a program, with support for several operations we'll need later
  - *Note: Folder split for performance reasons only*
- Part III: Project 4 Reflection

Lecture 6 Wrap-up

- Stoked for these subjects lined up for Week 4!
  - Metacognitive Subject: Annotation Strategies
  - Technical Subject: Machine Languages and Hack Assembly

- Project Reminders:
  - **Project 3 due tonight (Thursday, 4/14) at 11:59pm PDT**
  - Project 4 (Cornell Note Taking & Building Memory) released, due next Thursday (4/21) at 11:59pm PDT
  - Late days from Project 2 and before are updated on Canvas