CSE 390B, Winter 2022

Building Academic Success Through Bottom-Up Computing

Memory, Cornell Note-Taking Method

Revisiting the Cornell Note-Taking Method, Building Memory

Lecture Outline

- **Cornell Note-Taking Review**
  - **Group discussion: Compare and contrast notes**
- Storing Data: Bit
  - Bit overview and implementation
- Reading Review: Memory Representation
  - Array abstraction, reading and writing memory
- Building Memory: Registers
  - Building up from Bit to Register, then from Register to RAM
- Program Counter (PC) Overview
  - How do we keep track of which instruction to execute?

Cornell Note-Taking Discussion

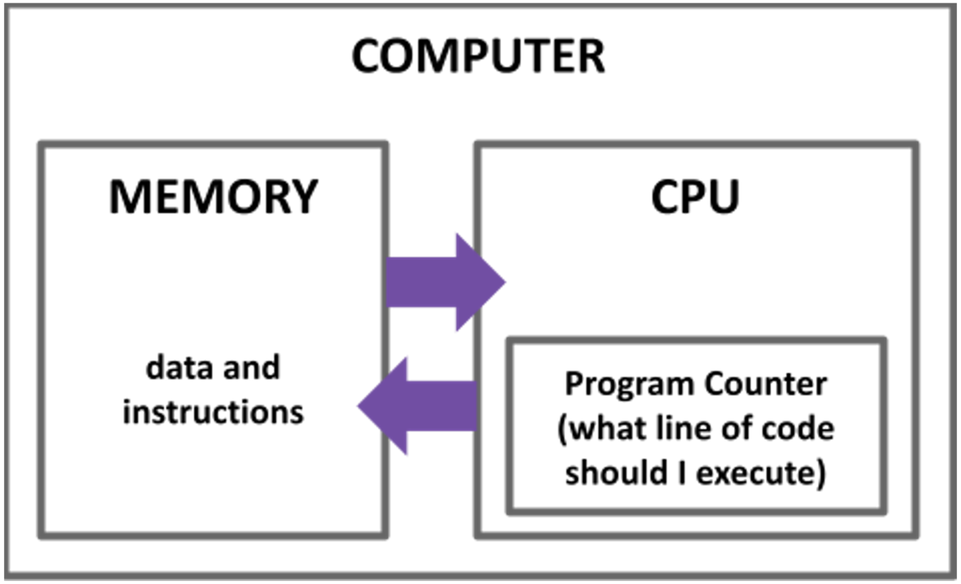
- In small groups, compare and contrast your Cornell Notes from Tuesday's lecture
  - What are some of the key points you wrote in your summary?
  - What were some of the questions you came up with?
  - What are you still left feeling confused/uncertain about after Tuesday's lecture?

Lecture Outline

- Cornell Note-Taking Review
  - Group discussion: Compare and contrast notes
- **Storing Data: Bit**
  - **Bit overview and implementation**
- Reading Review: Memory Representation
  - Array abstraction, reading and writing memory
- Building Memory: Registers
  - Building up from Bit to Register, then from Register to RAM
- Program Counter (PC) Overview
  - How do we keep track of which instruction to execute?

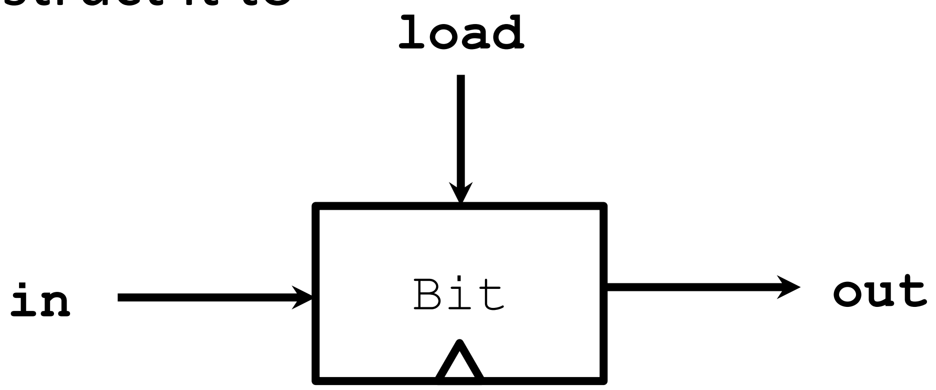
Computer Overview

- 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



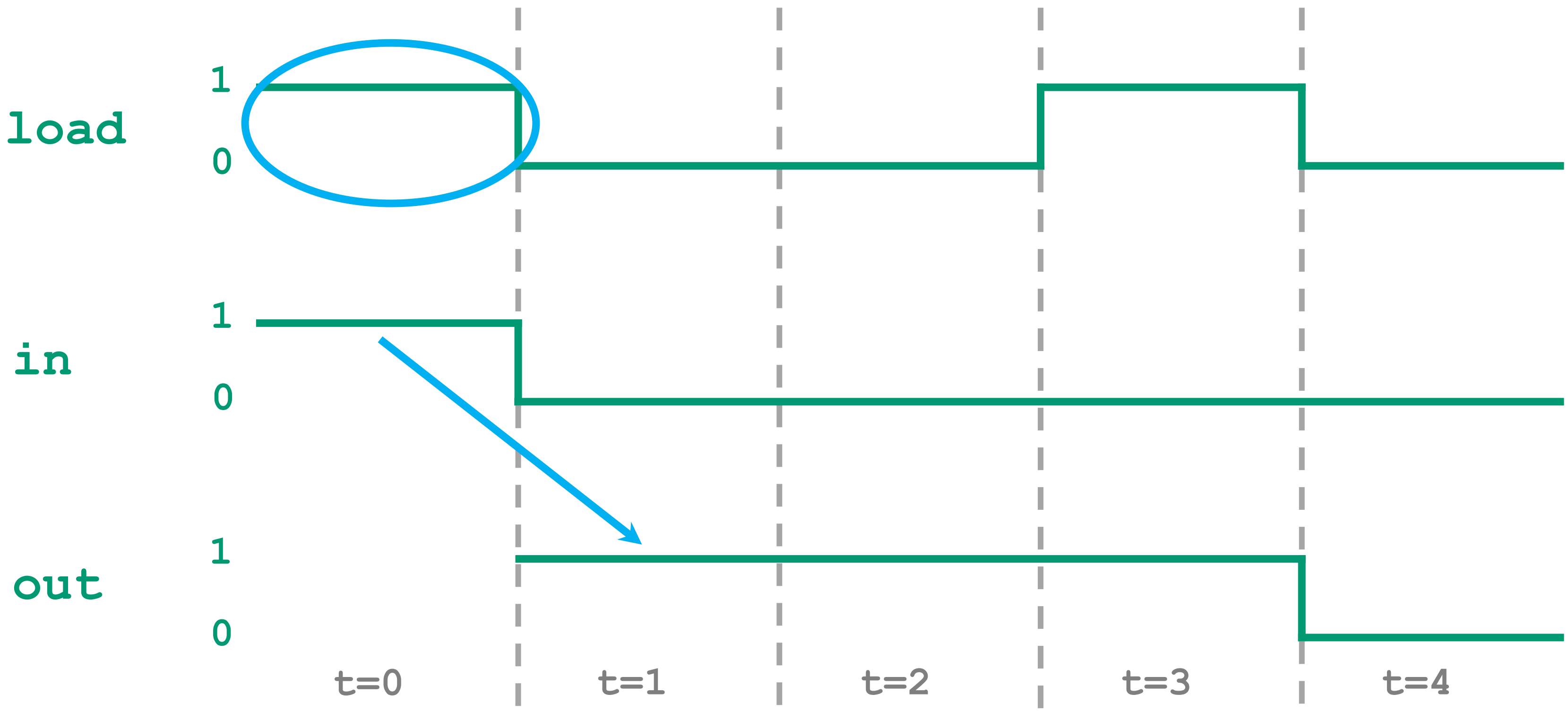
Storing Data: Bit

- A Flip-Flop changes state *every* clock cycle
- We will build the abstraction of a "Bit" that only changes when we instruct it to

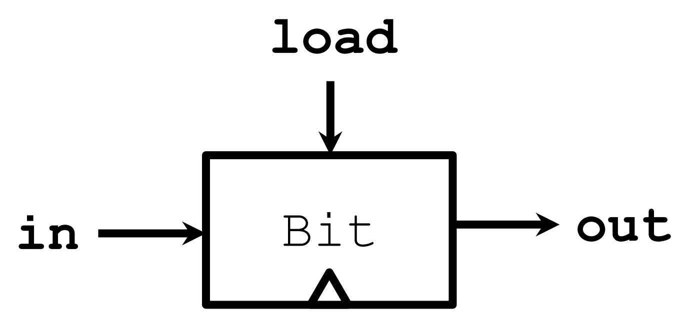



```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```

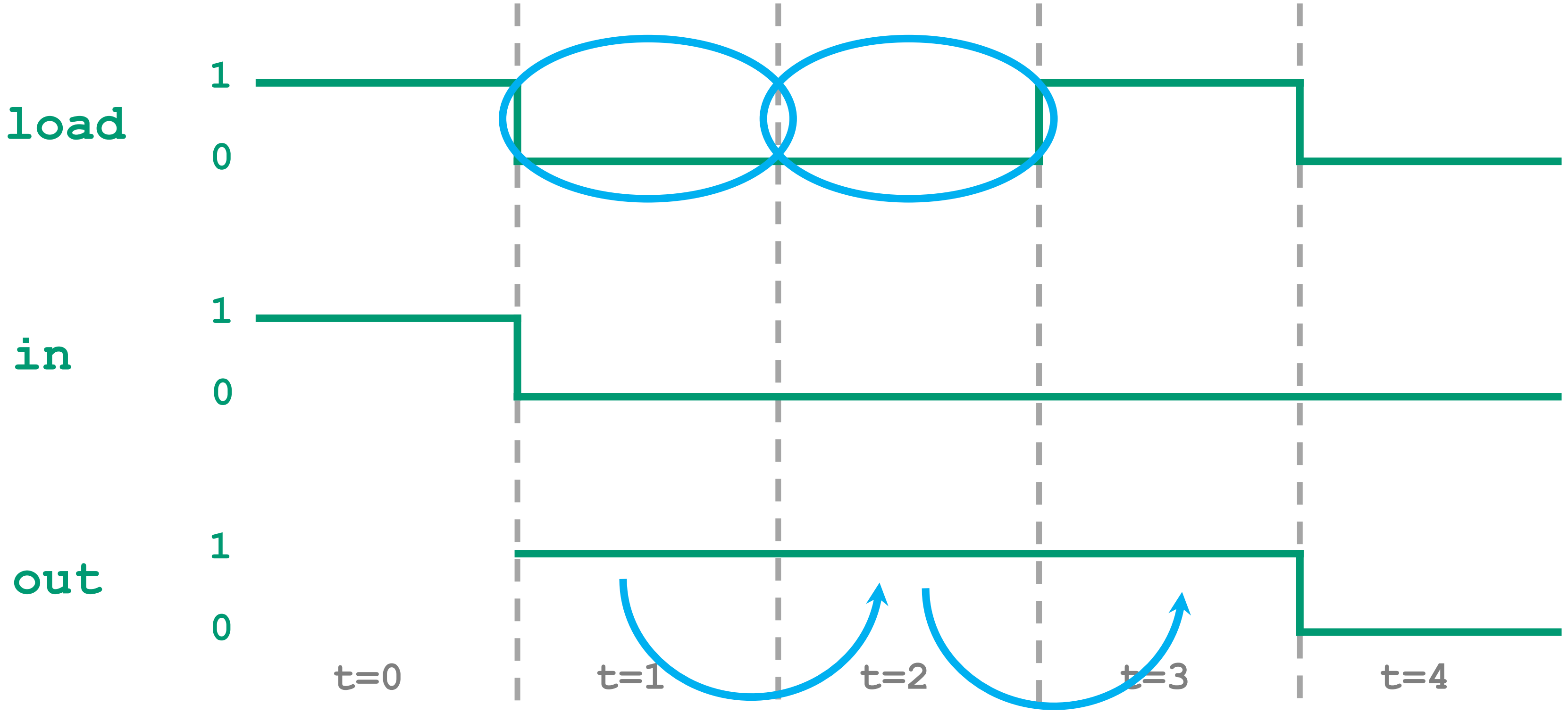
Bit Behavior



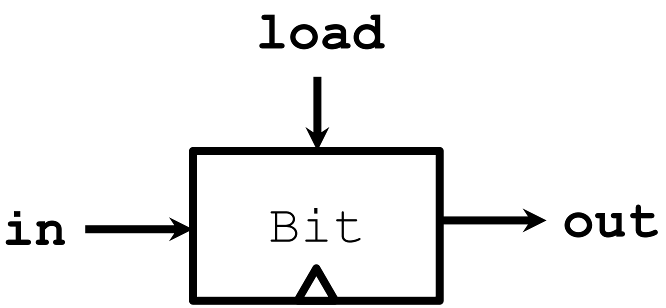

```
if load(t-1)      out(t) = in(t-1)
else              out(t) = out(t-1)
```



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)`

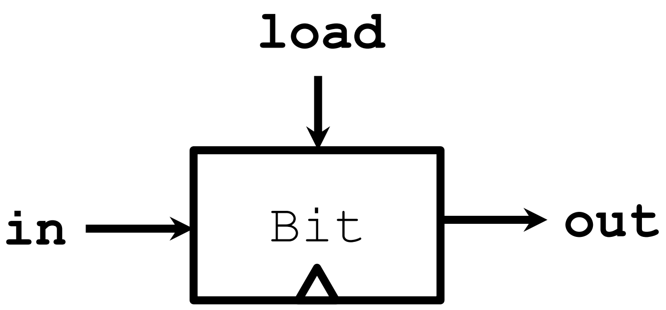
Example 2: `load(t=2)` `==` `0` so `out(t=3)` `=` `out(t=2)`

Poll Everywhere

Vote at https://pollev.com/cse390b

- **What 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)
```

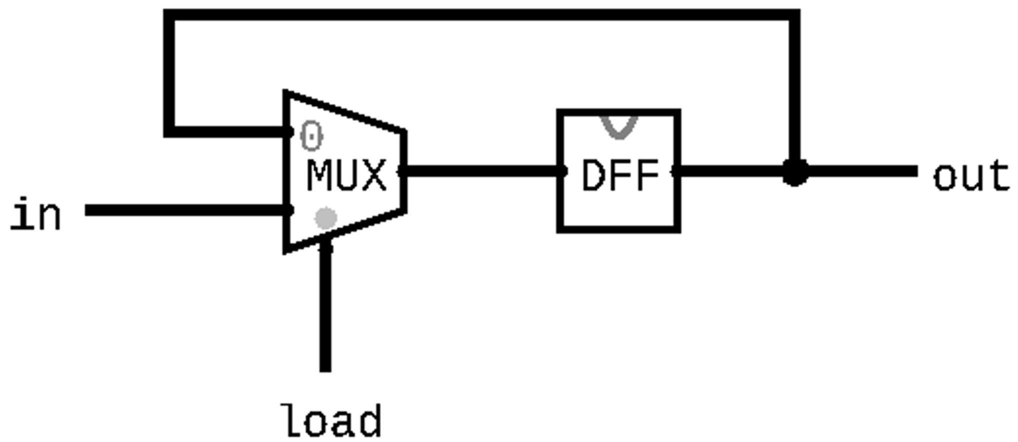
- Exercise: Fill in the connections to the gates to create a circuit diagram of Bit
  - May be helpful to review slides on Mux and sequential circuits

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
  - May be helpful to review slides on Mux and sequential circuits



Lecture Outline

- Cornell Note-Taking Review
  - Group discussion: Compare and contrast notes
- Storing Data: Bit
  - Bit overview and implementation
- **Reading Review: Memory Representation**
  - **Array abstraction, reading and writing memory**
- Building Memory: Registers
  - Building up from Bit to Register, then from Register to RAM
- Program Counter (PC) Overview
  - How do we keep track of which instruction to execute?

Reading Review: 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 slot in memory takes up a fixed width
  - Not the same as address width
  - The Nand2Tetris project uses 16-bit slots in memory

Reading Review: Memory Representation

- Can 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`

Lecture Outline

- Cornell Note-Taking Review
  - Group discussion: Compare and contrast notes
- Storing Data: Bit
  - Bit overview and implementation
- Reading Review: Memory Representation
  - Array abstraction, reading and writing memory
- **Building Memory: Registers**
  - **Building up from Bit to Register, then from Register to RAM**
- Program Counter (PC) Overview
  - How do we keep track of which instruction to execute?

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

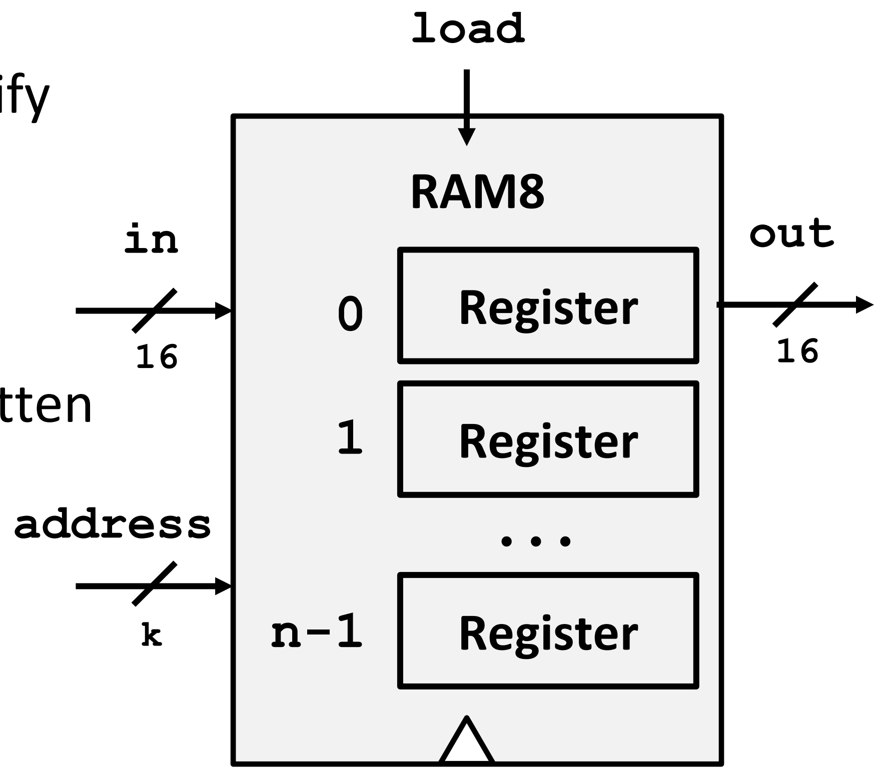
	24 11000	25 11001	26 11010	27 11011	28 11100	29 11101	30 11110	31 11111	
...	0 0000000	0 0000000	-1 1111111	25 0011001	124 1111100	0 0000000	9 0001001	-15 1110001	...

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

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

Building Memory: RAM8 From Registers

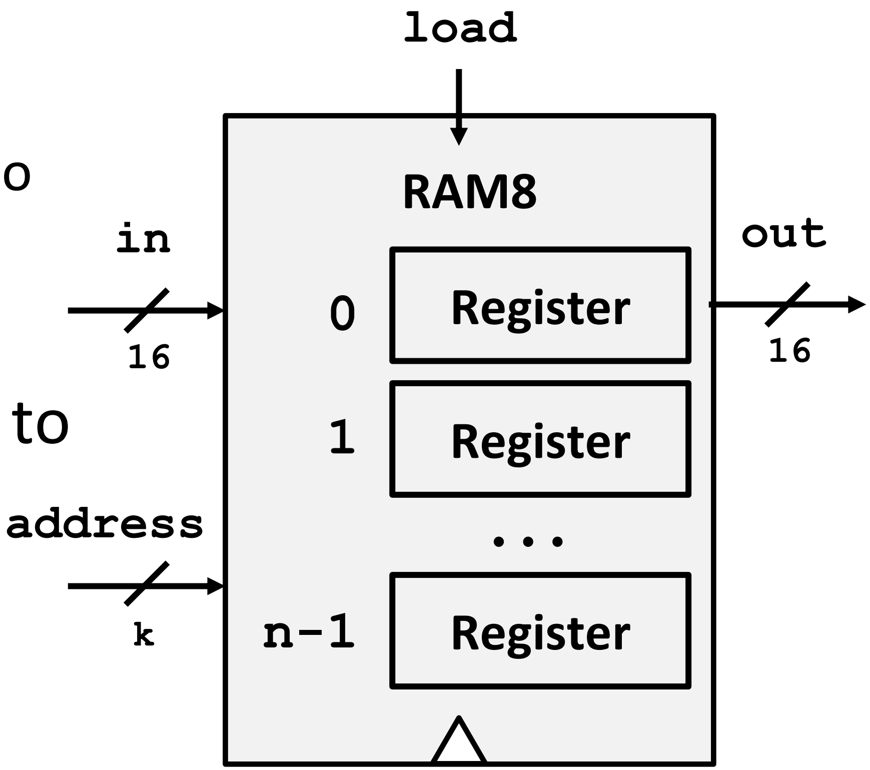
- RAM interface:
  - **`address`**: address used to specify memory slot
  - **`in`**: 16-bit input used to update specified memory slot if load is 1
  - **`load`**: if 1, then in should be written to specified memory slot
  - **`out`**: 16-bit output from the slot specified by address



- 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

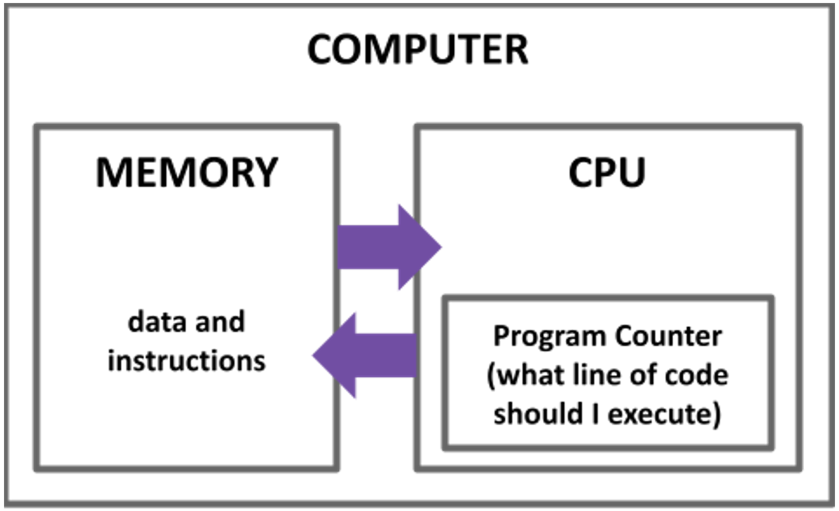
- After RAM8, can build larger RAM chips from a combination of smaller RAM chips
  - For example, RAM64 can be built using eight RAM8 chips
- Technique is similar to RAM8 but will have to use different portions of the address
- The blocks section of the reading will be helpful
  - For example, can think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

Lecture Outline

- Cornell Note-Taking Review
  - Group discussion: Compare and contrast notes
- Storing Data: Bit
  - Bit overview and implementation
- Reading Review: Memory Representation
  - Array abstraction, reading and writing memory
- Building Memory: Registers
  - Building up from Bit to Register, then from Register to RAM
- **Program Counter (PC) Overview**
  - **How do we keep track of which instruction to execute?**

Program Counter (PC)

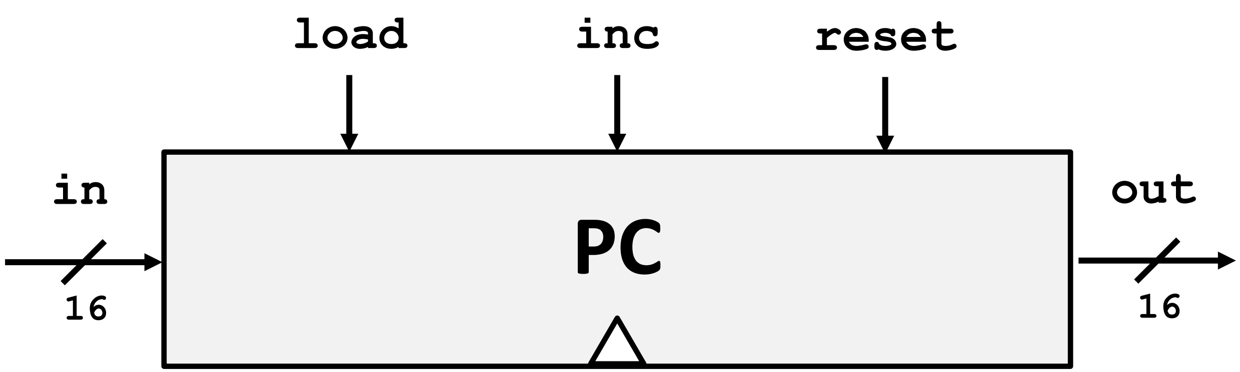
- Memory is used to store data as well as code
- Instructions and operations are stored at different addresses in memory
- Program Counter in the CPU keeps track of which address contains the instruction that should be executed next



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 3 Overview

- Part I: Cornell Note-Taking Method
  - Practice taking detailed notes in another class
  - Think critically about the technique
- Part II: 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: Social Computing Reflection
  - Applications of Memory and Sequential Logic

Post-Lecture 6 Reminders

- Reminders
  - Project 1 grades and feedback released on Gradescope
  - Project 2 due tonight (2/20) at 11:59PM PST
  - Eric has office hours after lecture today from 3-4pm

- Starting next week, Eric's office hours on Thursday from 3-4pm will be on Wednesdays from 4:30-5:30pm

- What's in store for Week 4?
  - Technical Subject: Machine and Assembly Languages
  - Metacognitive Subject: Annotation Strategies
  - Project 4 (Machine Language, Annotation) released next Thursday