CSE 390 B Spring 2021

Building a Computer & Exam Preparation

Organization of a Computer, Fetch/Execute Cycle, Hack CPU Design, Exam prep fundamentals

Agenda

- **Cornell Note-Taking Debrief**
- Exam Preparation
- Building a Computer Overview
- Reading Review and Q&A
- Hack CPU Logic

Project 3 Cornell Note-Taking Debrief

Take a look at your Cornell notes from CSE 390B and from another course that you practiced Cornell note-taking with.

- What elements of the cornell note-taking method allowed you to better understand and work on Project 3?
  - How are these elements similar/different when comparing this to your other course?
- What were barriers that prevented you from fully engaging in the cornell note-taking method (either in this class or another class)?
  - What are ways that can help address this?

Agenda

- Cornell Note-Taking Debrief
- **Exam Preparation**
- Building a Computer Overview
- Reading Review and Q&A
- Hack CPU Logic

Gearing up for your exams…

- **Make a Study Plan**
  - What key topics/concepts with your exam cover?
  - How might your study guides look different for specific classes?
  - What resources, materials, or people might you need to engage with?

- **Create a Schedule**
  - DON'T CRAM
  - Office hours, review sessions, study groups
  - Reference your weekly time commitments & quarterly calendar

- **Test Yourself**
  - Utilize your cornell question notes
  - Replicate exam-like environments

Gearing up for your exams...

- **Make a Study Plan**
  - What key topics/concepts with your exam cover?
  - How might your study guides look different for specific classes?
  - What resources, materials, or people might you need to engage with?
- **Create a Schedule**
  - DON'T CRAM
  - Office hours, review sessions, study groups
  - Reference your weekly time commitments & quarterly calendar
- **Test Yourself**
  - Utilize your cornell question notes
  - **Replicate exam-like environments**

Project 5: Timed Mock Exam Problem

- Schedule a 30-minute session is based on your group members availability do **one** mock exam problem
- Determine how you will get in touch with each other if needed
- Determine who will be the zoom host for the session
- Email cse390b-staff@cs.washington.edu with your group's meeting day & time

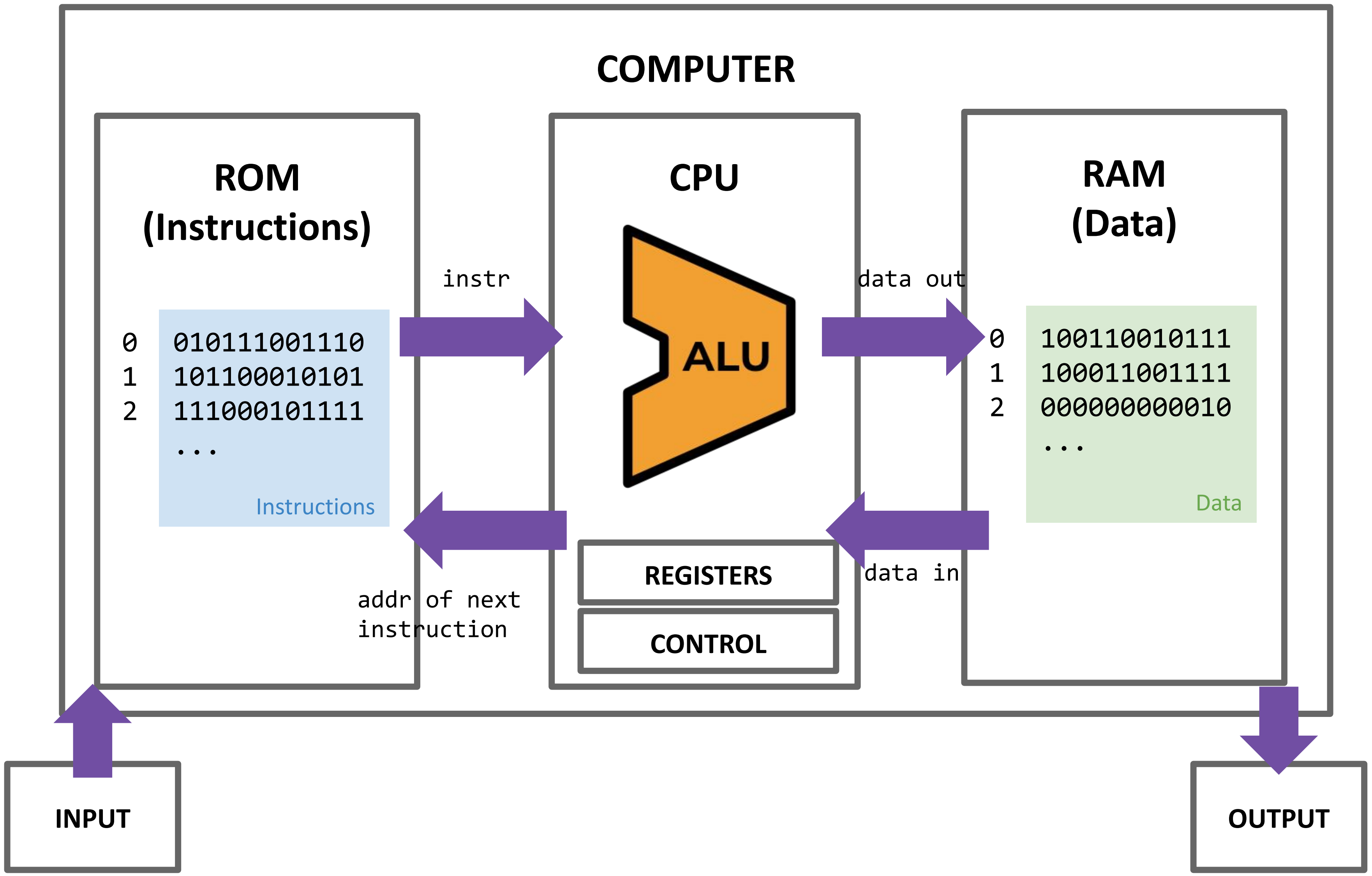
Agenda

- Cornell Note-Taking Debrief
- Exam Preparation
- **Building a Computer Overview**
- Reading Review and Q&A
- Hack CPU Logic

Building a Computer

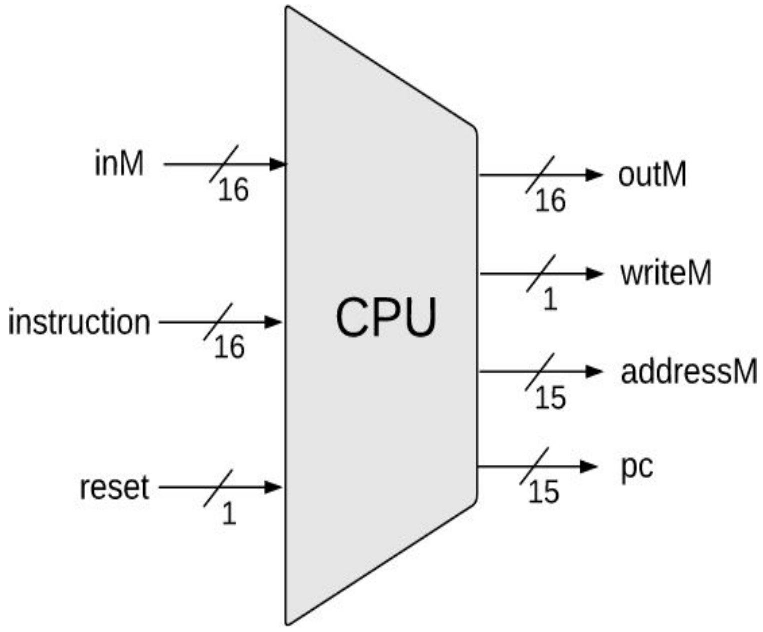
- All your hardware efforts are about to pay off! In project 5, you will build **Computer.hdl** -- the final, top-level chip in this course
  - For all intents and purposes, a real computer
  - Simplified, but organization very similar to your laptop
- Later projects we will start writing software to make it useful

Hack CPU


COMPUTER
ROM
(Instructions)
0 010111001110
1 101100010101
2 111000101111
...
Instructions
instr
addr of next instruction
CPU
ALU
REGISTERS
CONTROL
data out
data in
RAM
(Data)
0 100110010111
1 100011001111
2 000000000010
...
Data
INPUT
OUTPUT

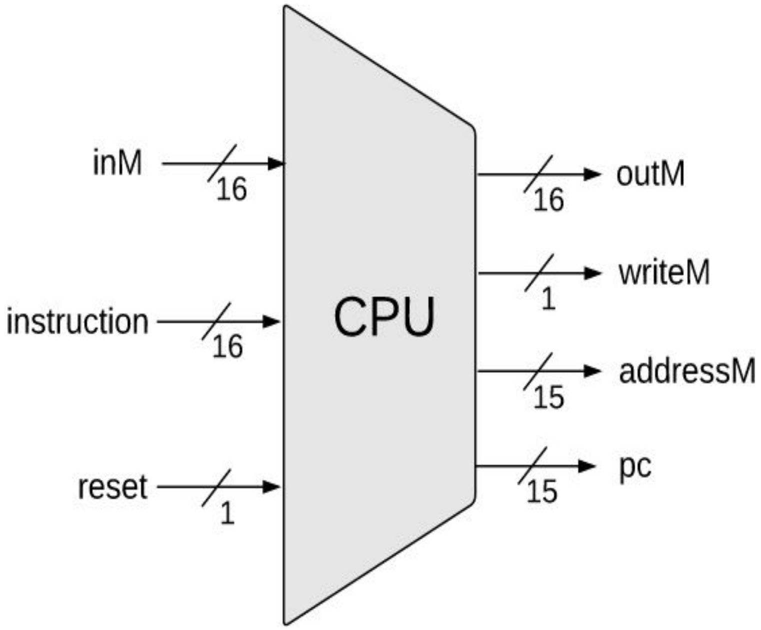
Hack CPU Interface Inputs

- Inputs:
  - **inM:** Value coming from memory
  - **instruction:** 16-bit instruction
  - **reset:** if 1, reset the program



Hack CPU Interface Outputs

- Outputs:
  - **outM:** value used to update memory if **writeM** is 1
  - **writeM:** if 1, update value in memory at **addressM** with **outM**
  - **addressM:** address to read from or write to in memory
  - **pc:** address of next instruction to be fetched from memory

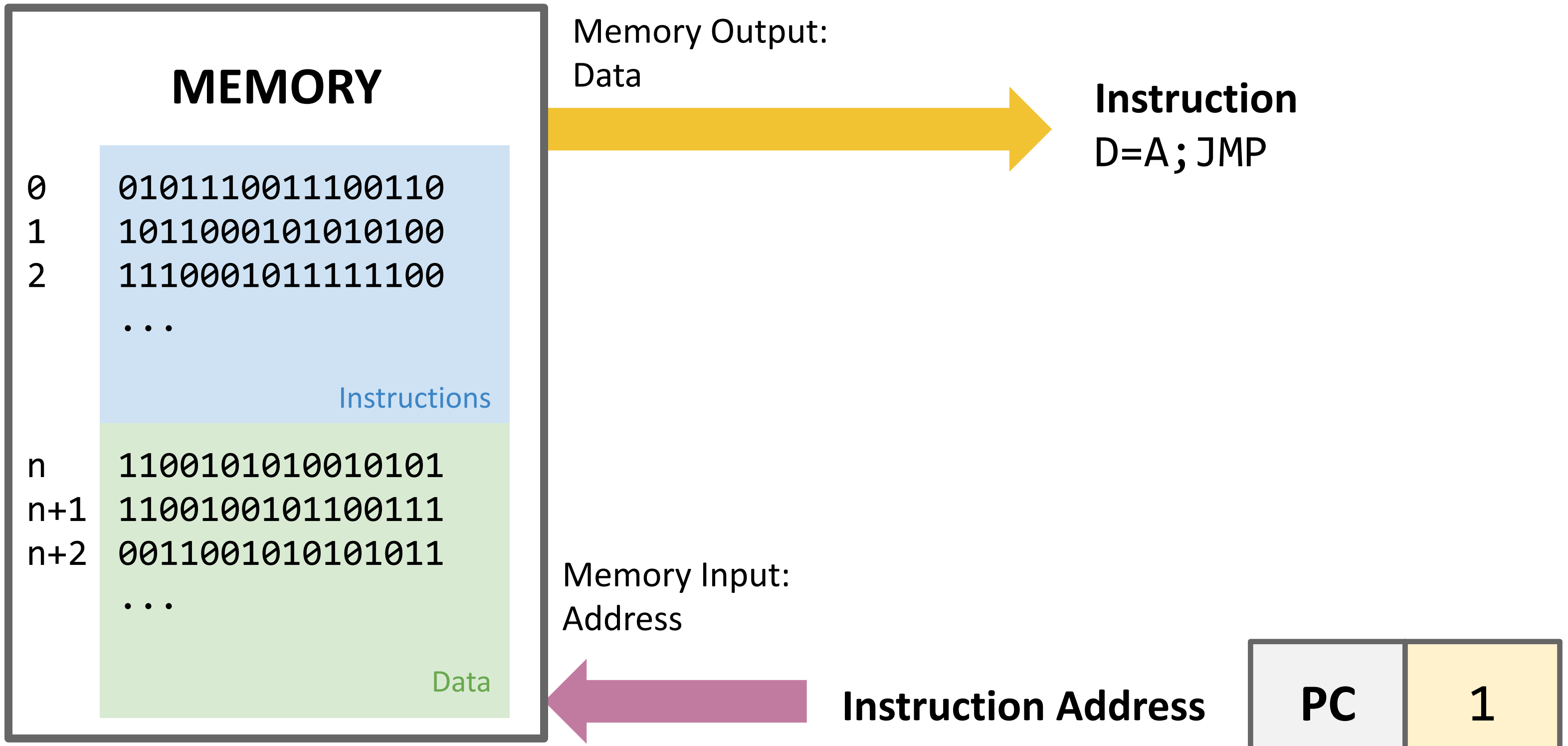


Basic CPU Loop

- Repeat forever:
  - ***Fetch*** an instruction from the program memory
  - ***Execute*** that instruction

Fetching

- Specify which instruction to read as the address input to our memory
- Data output: actual bits of the instruction



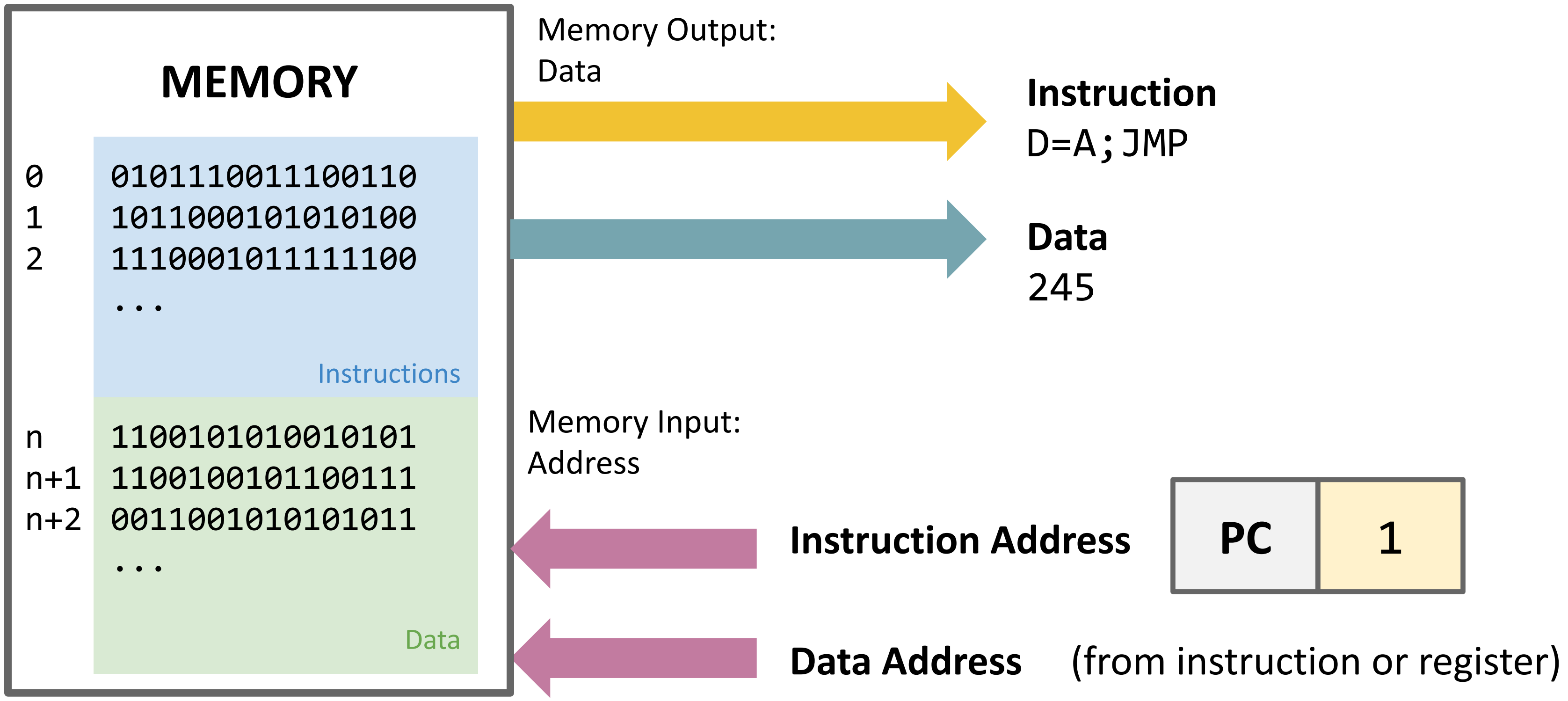
Executing

- The instruction bits describe exactly "what to do"
    - A-instruction or C-instruction? Which operation for the ALU? What memory address to read? To write? If / where to jump after this instruction?

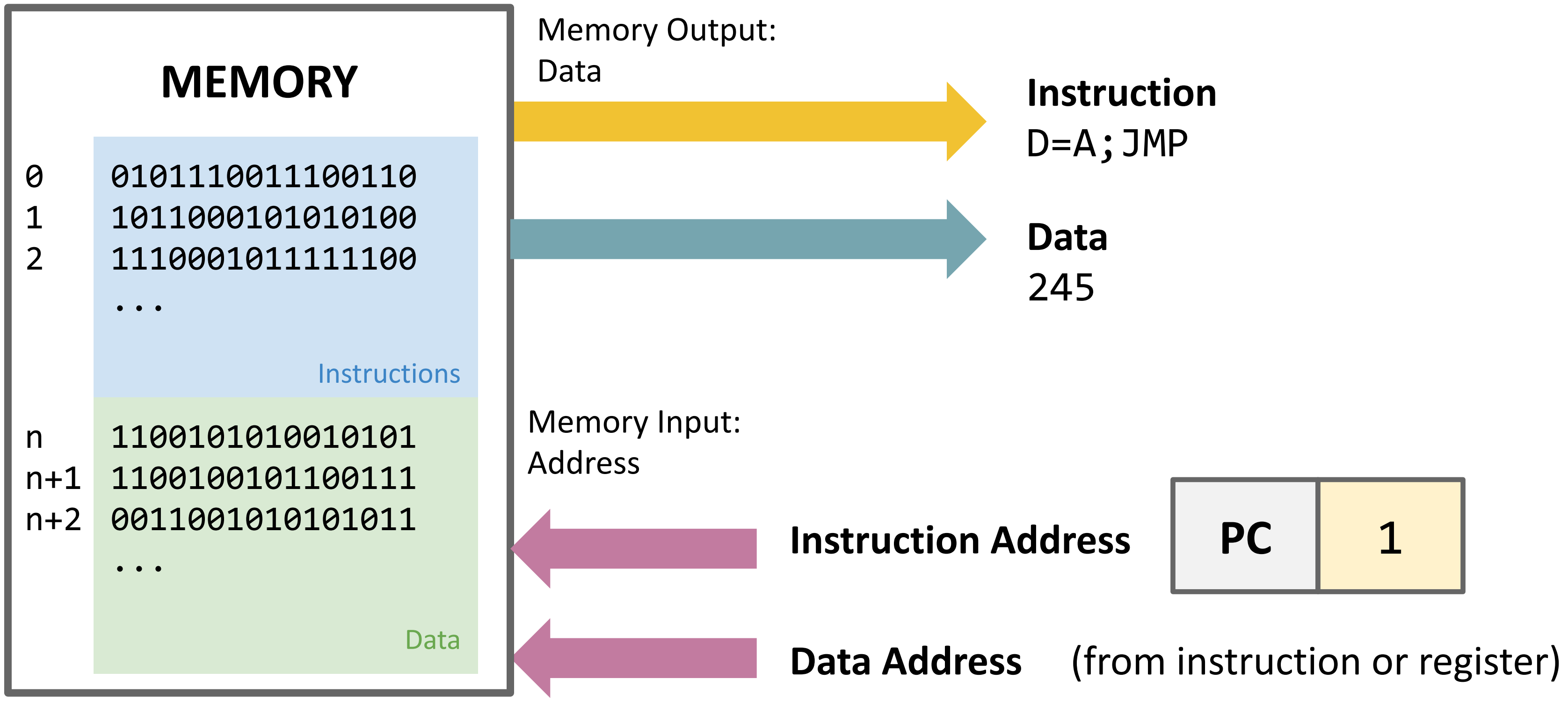
- Executing the instruction involves data of some kind.
    - Accessing registers

      and/or
    - Accessing memory

Combining Fetch & Execute

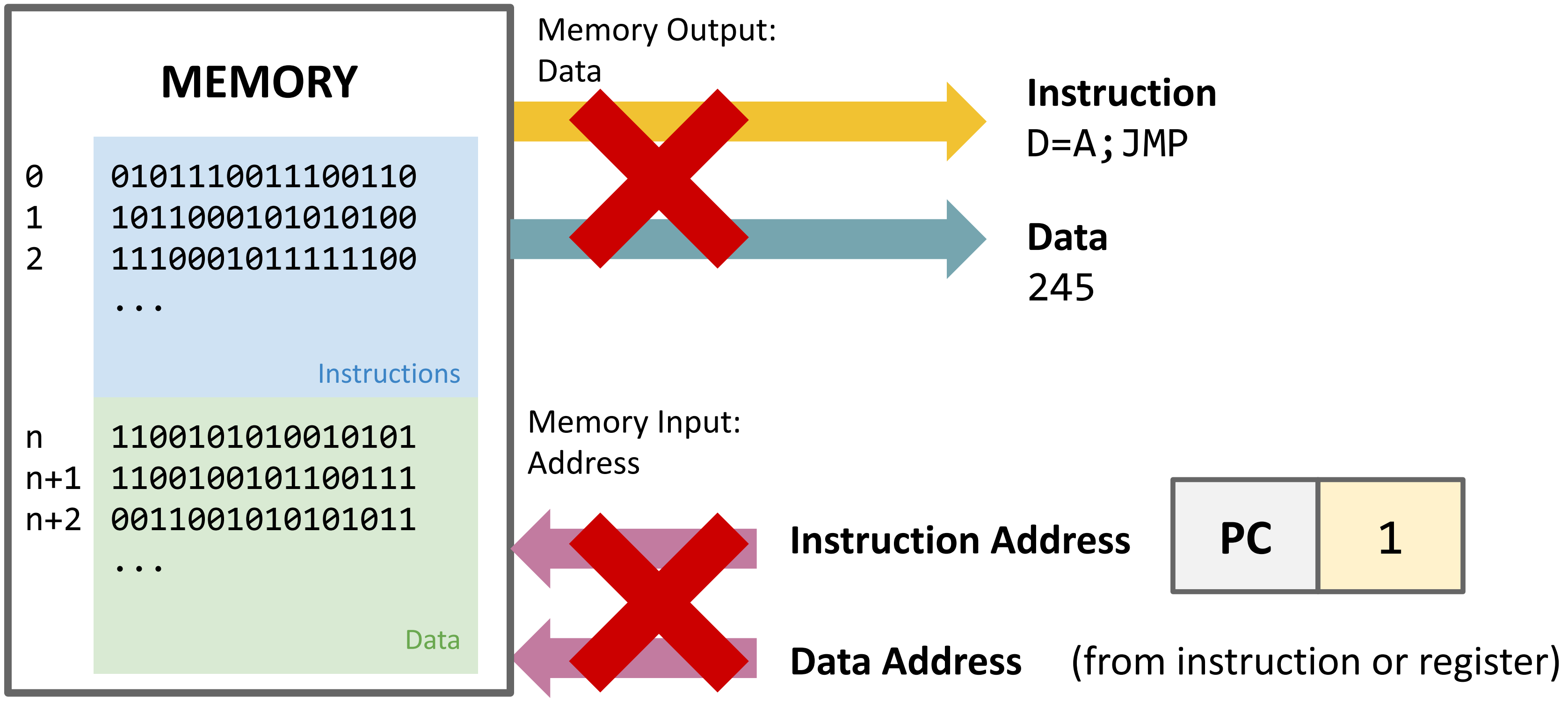


Combining Fetch & Execute



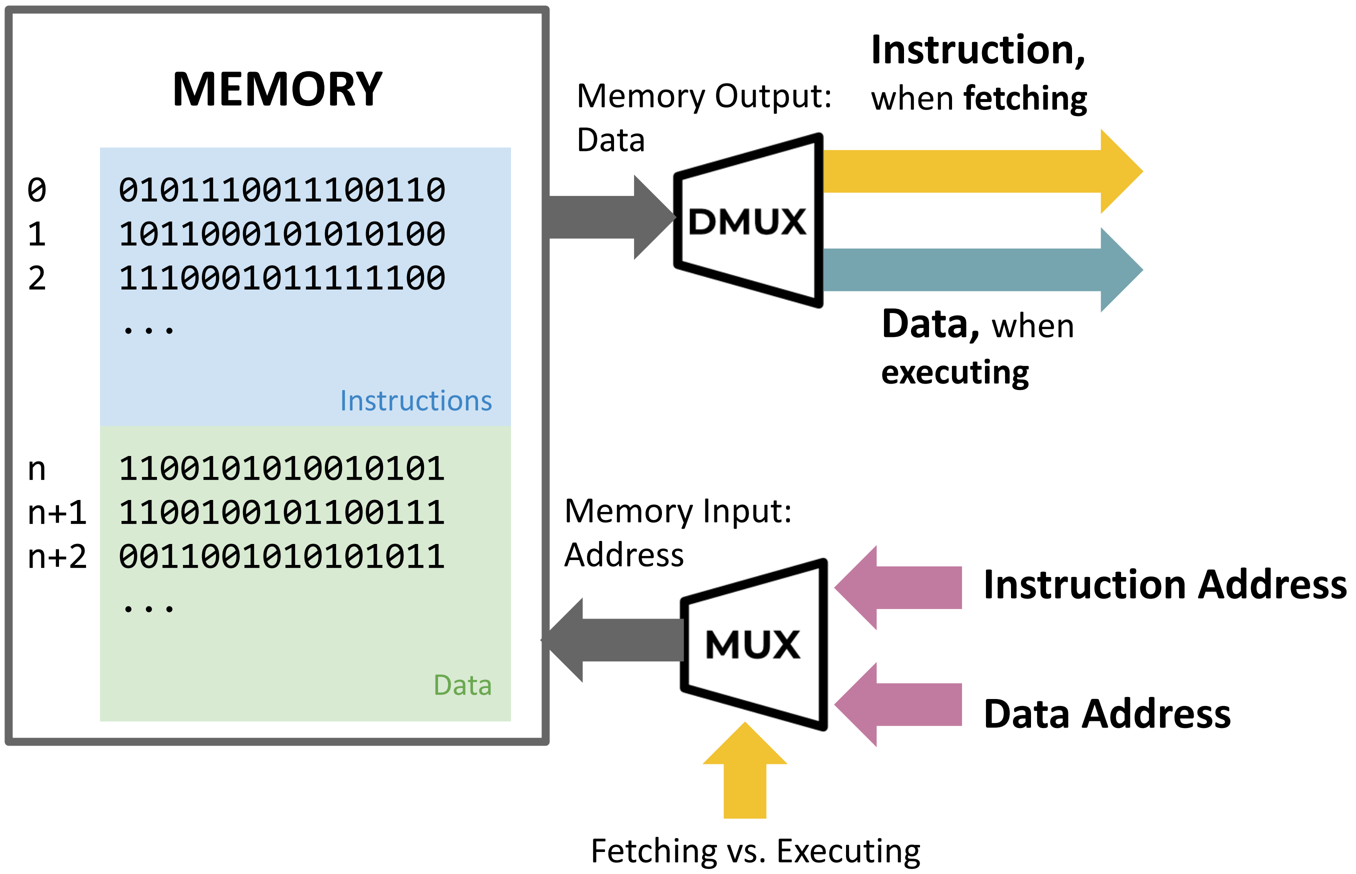
- Could we implement with `RAM16K.hdl`?

Combining Fetch & Execute



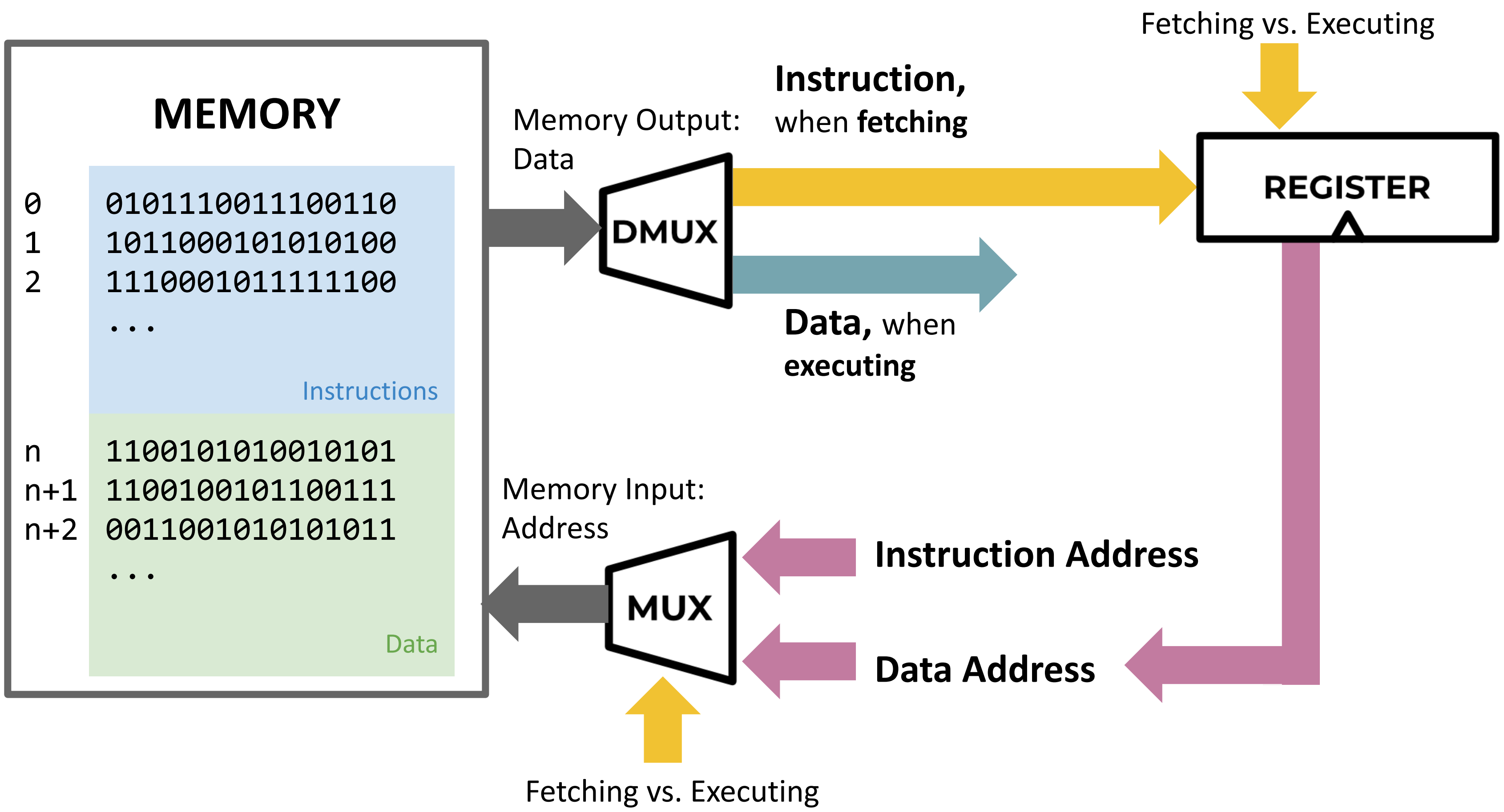
- Could we implement with `RAM16K.hdl`?
  - **No!** Our memory chips only have one input and one output!

Solution 1: Handling Single Input/Output



- Can use multiplexing to share single input/output!

Solution 1: Fetching/Executing Separately



- Need to store fetched instruction so it's available during execution phase.

Solution 2: Separate Memory Units

- Separate instruction memory and data memory into two different chips
    - Each can be independently addressed, read from, written to
- This is what we will do in Project 5!
    - See Chapter 5 for more detail on design
- Pros:
    - Simpler to implement
- Cons:
    - Fixed size of each partition, rather than flexible storage
    - Two chips ⟶ redundant circuitry

Agenda

- Cornell Note-Taking Debrief
- Exam Preparation
- Building a Computer Overview
- **Reading Review and Q&A**
- Hack CPU Logic

Hack CPU Implementation

- Need to be able to provide the functionality our assembly language specifies

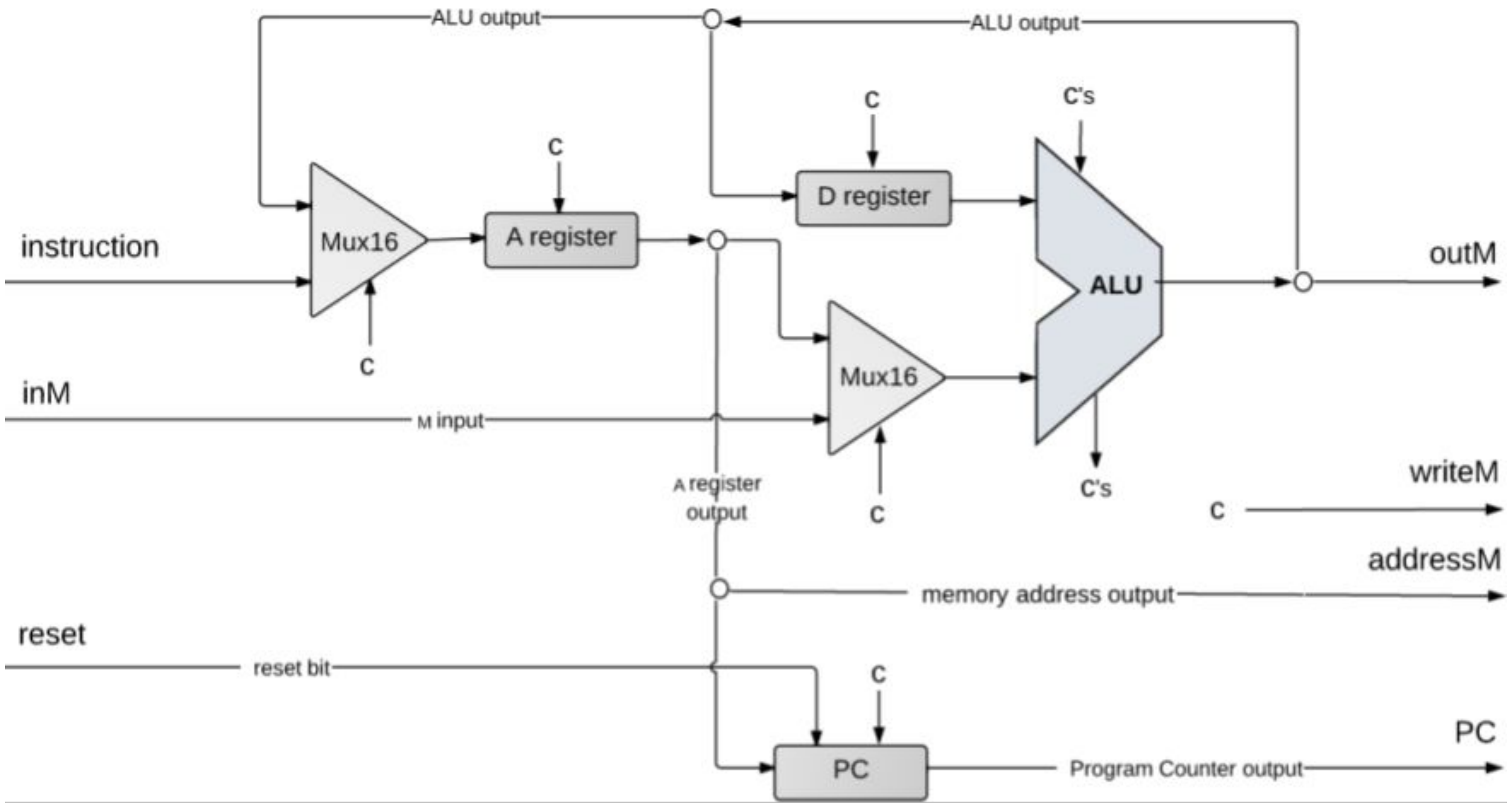
- A-instructions
  - Need to be able to load values into the A-Register
- C-instructions
  - Need to perform different computations w/varying inputs
  - Need to be able to store the results in different destinations
- Flow Control
  - Need to keep track of our current instruction address and know what address to execute next

Hack CPU Implementation

- Only 4 main components needed!
    - ALU
    - PC
    - Registers (x2) for A and D
        - For testing & debugging reasons, you'll use built-in ARegister.hdl and DRegister.hdl instead.

- Tricky Part: All the control logic
    - We'll recommend an overall flow of data
    - Your task: to design and implement proper control logic

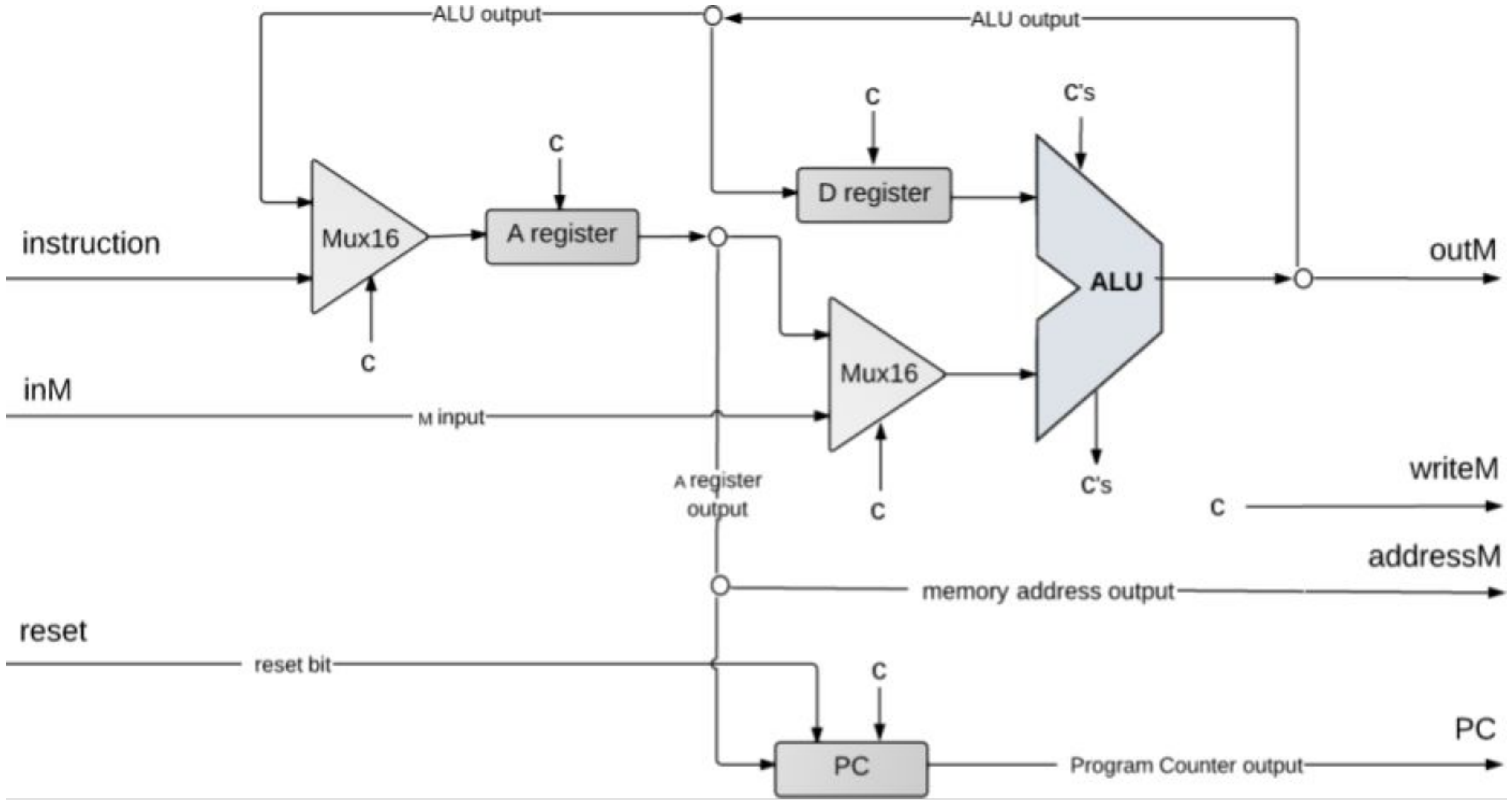
Hack CPU Design



A-Instruction Design

- Need to load value of instruction into the A-register
  - Corresponds to our @value syntax
- Possible solution: setup A-register w/instruction as input
  - Problem: sometimes need to store a computation result in the A-register (e.g. A = D + 1)
- Solution: use a mux to choose either the instruction value or the previous ALU output as the A-register input
  - Still need logic to determine if the A-regiseter should be loaded

Hack CPU Design: A-instructions



C-Instruction Design: Inputs

- Need inputs from the A, D, and M registers
    - Never need to use the A register and M register in a computation together!
- One ALU input will always be the D register
- The other will either be the A register or the M register
    - inM is the input w/the M register value
    - Can use a Mux to make this choice!
- Remember constants are generated by the ALU internally

C-Instruction Design: Computations

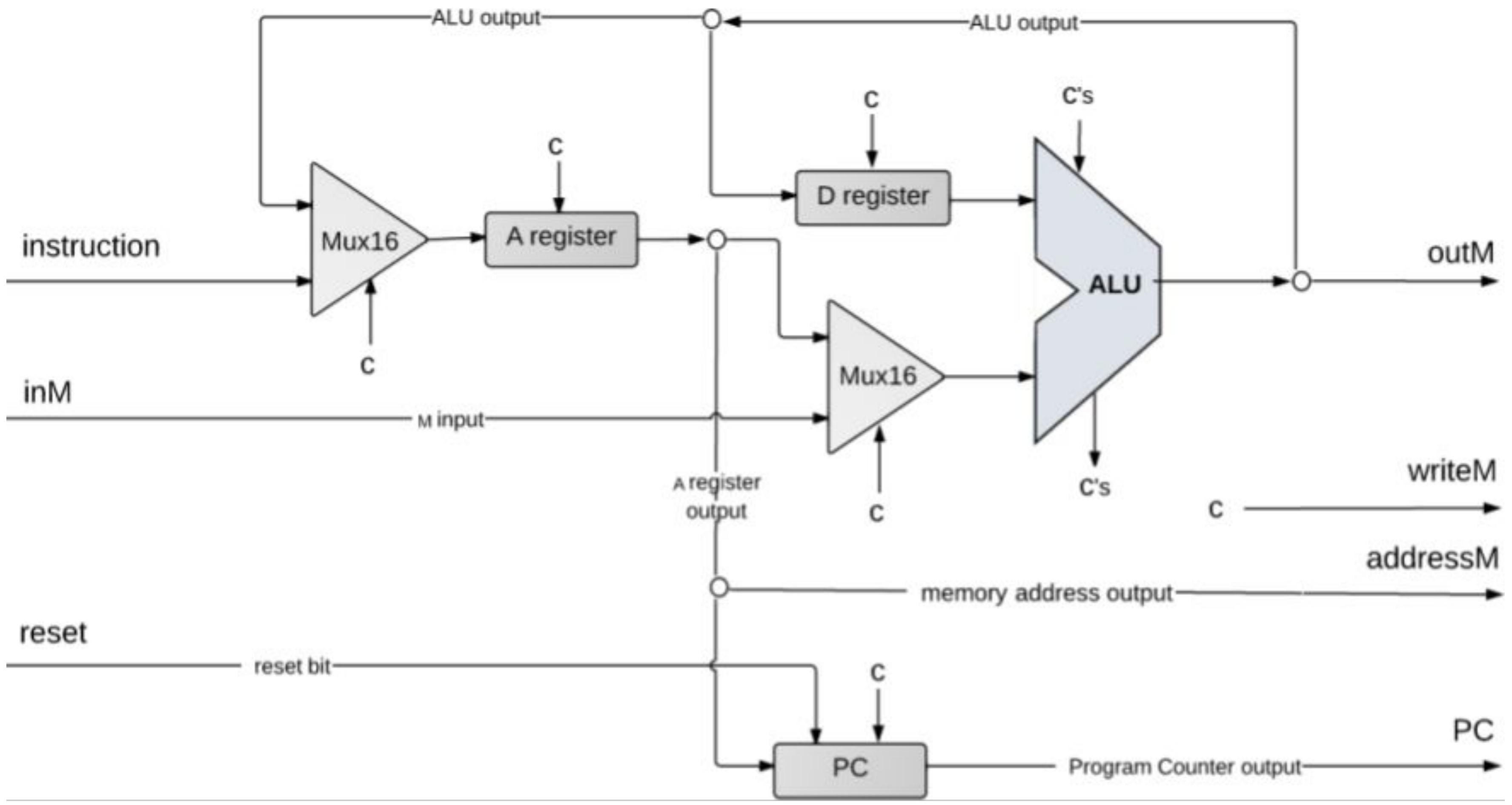
- ALU performs the computations
- Now that we have our inputs, just need to specify the correct computation for the ALU to execute
- You'll note the computation bits in the instruction binary are very similar to the control inputs to the ALU

C-Instruction Design: Destinations

- Can store computations in three destinations: the A, D, or M registers
  - Loop ALU output back to the A and D registers
  - outM, writeM, and addressM used to write to the M register

- Even though our ALU output is connected to these locations, we don't always want to update them
  - Control logic will specify when we want to write to these locations

Hack CPU Design: C-instructions



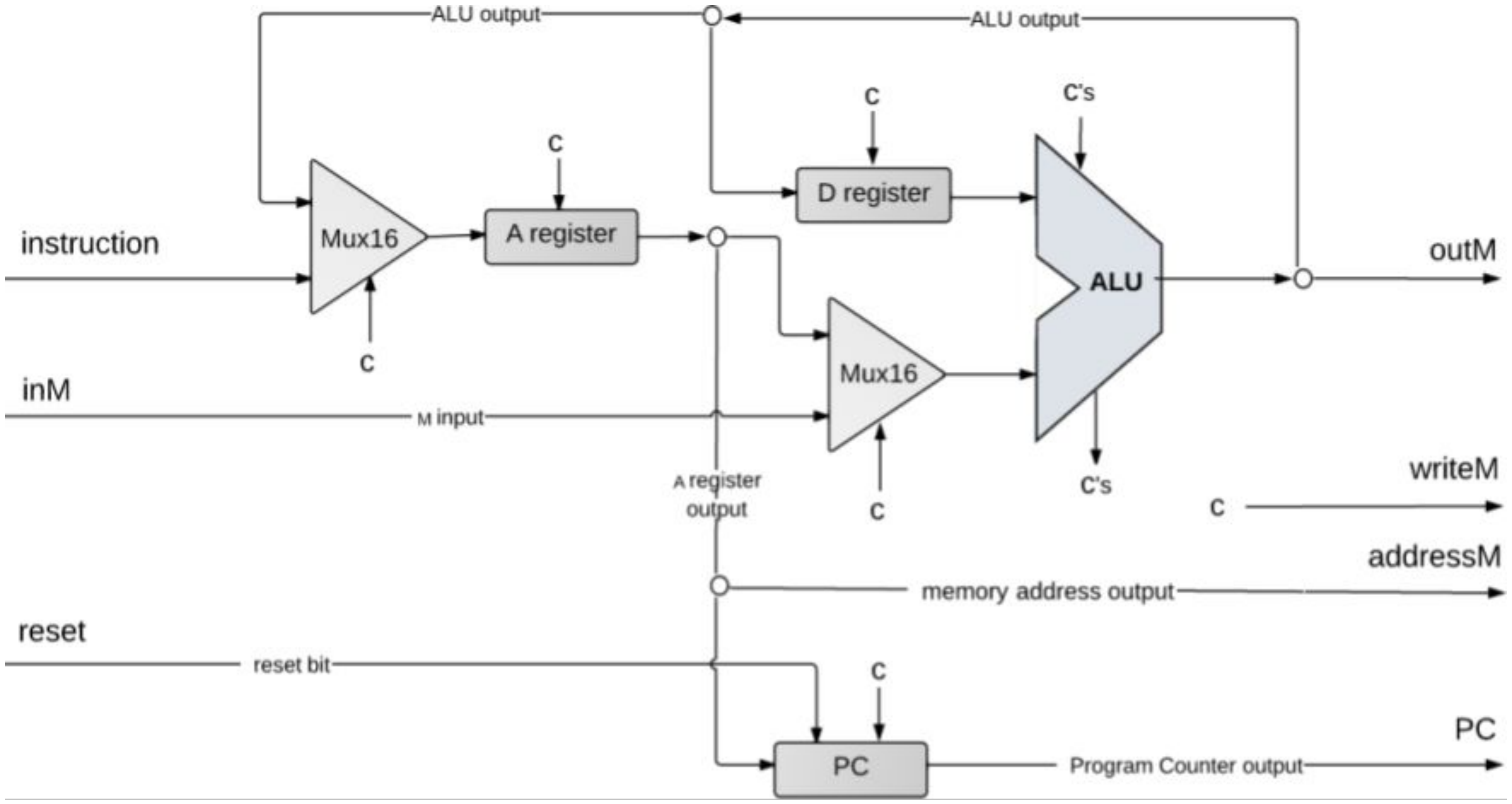
Flow Control Design: Flow Control

- Use the Program Counter chip to keep track of the current instruction address

- Input will be from the A-register
  - When we jump to an address in assembly, we jump to the address specified in the A-register

- Load (jump) or increment determined by the output from our ALU
  - More specifically can use the status flags to determine if the output is < 0, == 0, or > 0

Hack CPU Design: Flow Control



Agenda

- Cornell Note-Taking Debrief
- Exam Preparation
- Building a Computer Overview
- Reading Review and Q&A
- **Hack CPU Logic**

Hack CPU Logic

- How do we determine the unimplemented logic for the CPU (all of the c's in the diagrams)?

- Need to refer to the assembly specification!

- Project 5 will require a good bit of consulting of Chapter 4 to figure out how to use the instruction bits to implement the control logic

Hack CPU Logic Workflow

- Step 1: Figure out what to pay attention to
  - Usually will be some combination of instruction bits and/or intermediate outputs
  - These are the “inputs” to your sub-problem

- Step 2: determine logic for the part you are working on
  - Uses the “inputs” from step 1
  - Usually requires reading a relevant section of the textbook/assembly specification

Instruction Bits: A-instruction

```
16 bits: 0 v v v v v v v v v v v v v v v
```

- Most significant bit is a 0 (indicates an A-instruction)
- Rest of the bits are the value to be loaded
  - Since most significant bit is 0, entire A-instruction is also the value to be loaded

Instruction Bits: C-instruction

16 bits: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

- Most significant bit is a 1 (indicates a C-instruction)
- Next two most significant bits aren't used (always 1)
- a-bit and c-bits are related to computations
- d-bits are related to destination locations
- j-bits are related to jumping

Hack CPU Logic Example: writeM

- Example: determining when writeM should be set to 1

- Step 1: figure out what to pay attention to
    - writeM is related to where we store the output, or what **destination** we use
    - We need to look up the destination bits specification from Chapter 4!

Hack CPU Logic Example: writeM

- Example: determining when writeM should be set to 1

- Step 2: determine logic for specification
  - Read the "Destination Specification" section of Chapter 4
  - d3 determines if the output should be written to memory
  - Which bit of our instruction is that???
  - Instruction bits:

    `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

  - So writeM = instruction[3]?

Hack CPU Logic Example: writeM

- Example: determining when writeM should be set to 1

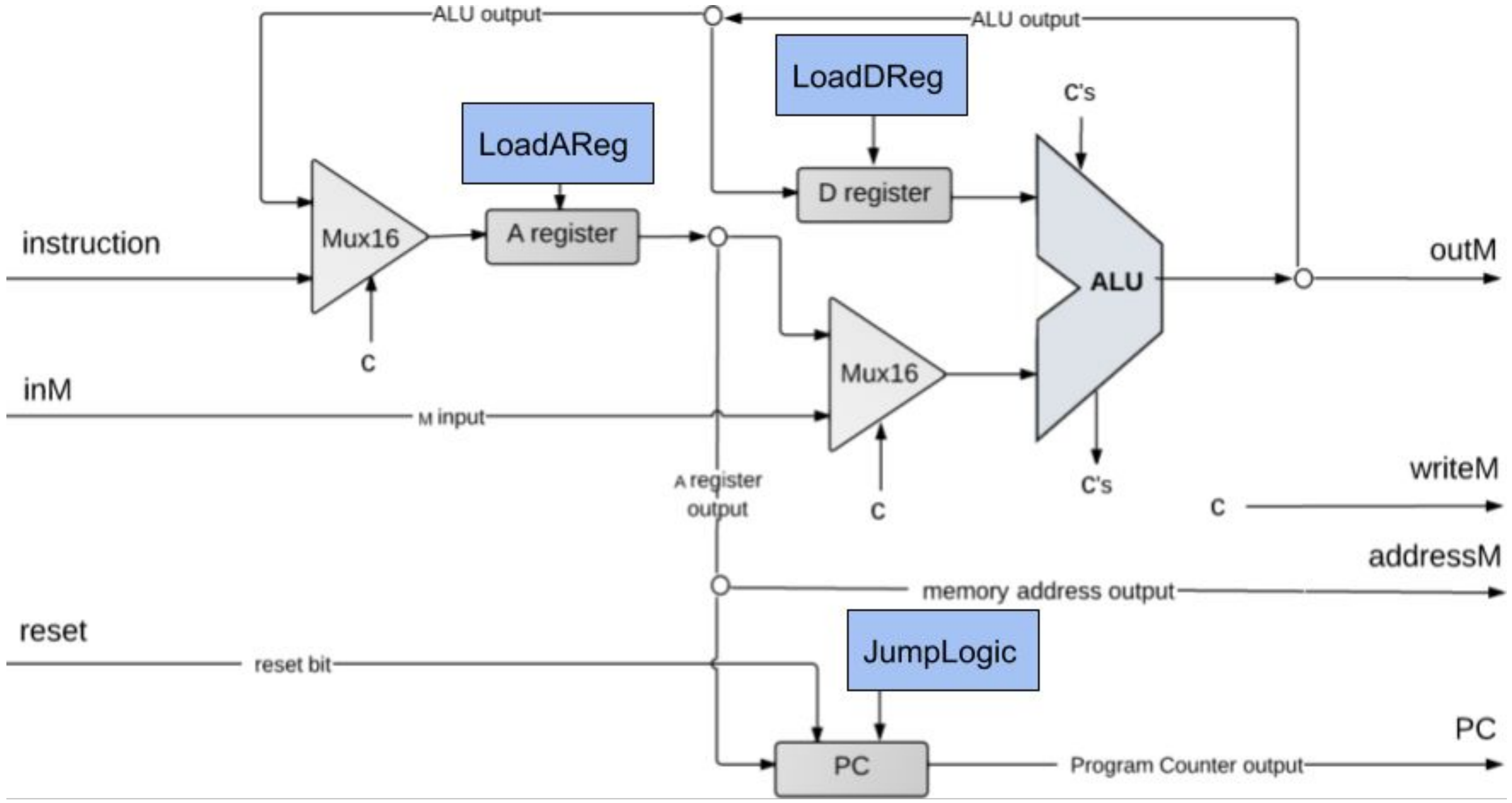
- Not so fast…
  - What happens if it's an A-instruction?
  - We only write to destinations in the case of a C-instruction
  - So writeM = C-instruction & instruction[3]
  - Remember that certain actions only occur on certain instruction types, you may have to include a check for instruction type in your logic depending on the action!

Hack CPU Implementation: Logic sub-chips

- We provide you with 3 sub-chips and tests that implement the control logic for the A Register, D Register, and PC
  - LoadAReg contains logic for loading the A Register
  - LoadDReg contains logic for loading the D Register
  - JumpLogic contains logic for determining if the PC should load/jump or increment

- Implement/test these first, then use them in your CPU implementation!
  - Intended to help you narrow the scope of any bugs you may have

Hack CPU Implementation: Logic sub-chips



Hack CPU Implementation: Logic sub-chips

- No in-class work today :(
- Thursday we will give you time to implement at least one of the Logic sub-chips

Reminders

- **Office Hours**
  - Eric & Margot’s office hours happening right after class!
- **Project 4:**
  - Due Thursday 11:59PM PDT
- **CSE 390B Midterm**
  - Thursday May 6th