CSE 390B, Spring 2022

Building Academic Success Through Bottom-Up Computing

Exam Preparation & Building a Computer

Exam Preparation, Overview of Building a Computer, Hack CPU Logic

Lecture Outline

- **Exam Preparation**
  - **Study Strategies, Mock Exam Problem**
- Overview of Building a Computer
  - Architecture, Fetch and Execute Cycle
- Hack CPU Logic
  - Implementation and Operations

Gearing Up For Exams

- Make a Study Plan
  - What key topics / concepts does your exam cover?
  - How might your study guides look different for specific classes?
  - What resources, materials, or people might you engage with?
- Create a Schedule
  - Do not cram
  - Office hours, review sessions, study groups
  - Reference your weekly time commitments & quarterly calendar
- Test Yourself
  - What are ways that can help address this?
  - Replicate exam-like environments

Exams Preparation Discussion

- How do you usually prepare for your exams?
- What is one thing that is effective and ineffective about the way you study? Why?
- What are some effective exam preparation strategies that you would find most helpful?
- How might you implement some of these effective strategies for change your exam preparation strategy for this quarter?

Project 6, Part I: Mock Exam Problem

- Schedule a 30-minute session based on your group members availability to do one mock exam problem
- Determine how you will connect with each other
- Determine where your session will be located
- Comment your group's meeting day and time on the Project 6 Mock Exam Groups spreadsheet
  - We will make an Ed post with this spreadsheet

Lecture Outline

- Exam Preparation
  - Study Strategies, Mock Exam Problem
- **Overview of Building a Computer**
  - **Architecture, Fetch and Execute Cycle**
- Hack CPU Logic
  - Implementation and Operations

Building a Computer

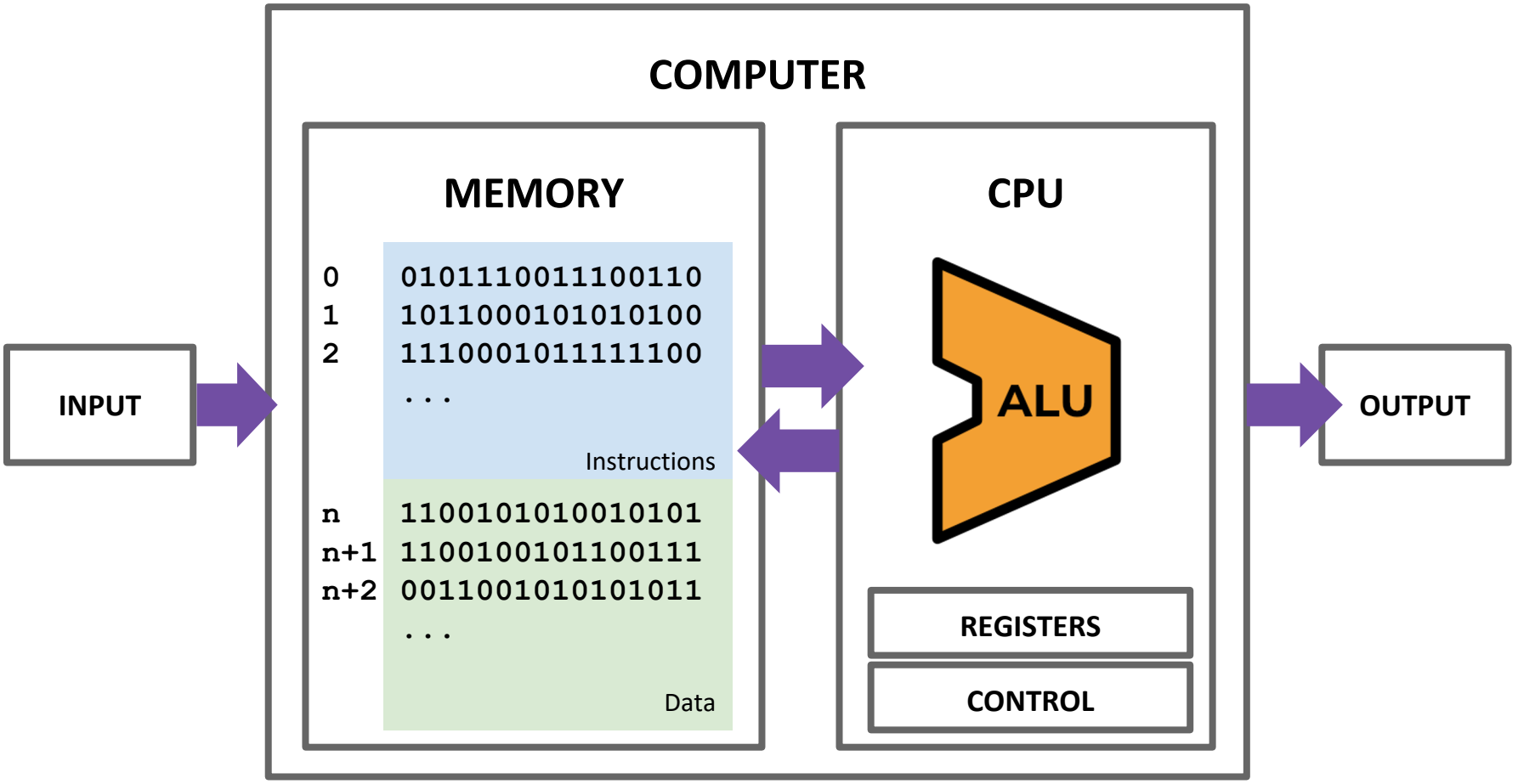
- All your hardware efforts are about to pay off!

- Perspective: **BUILDING A COMPUTER**

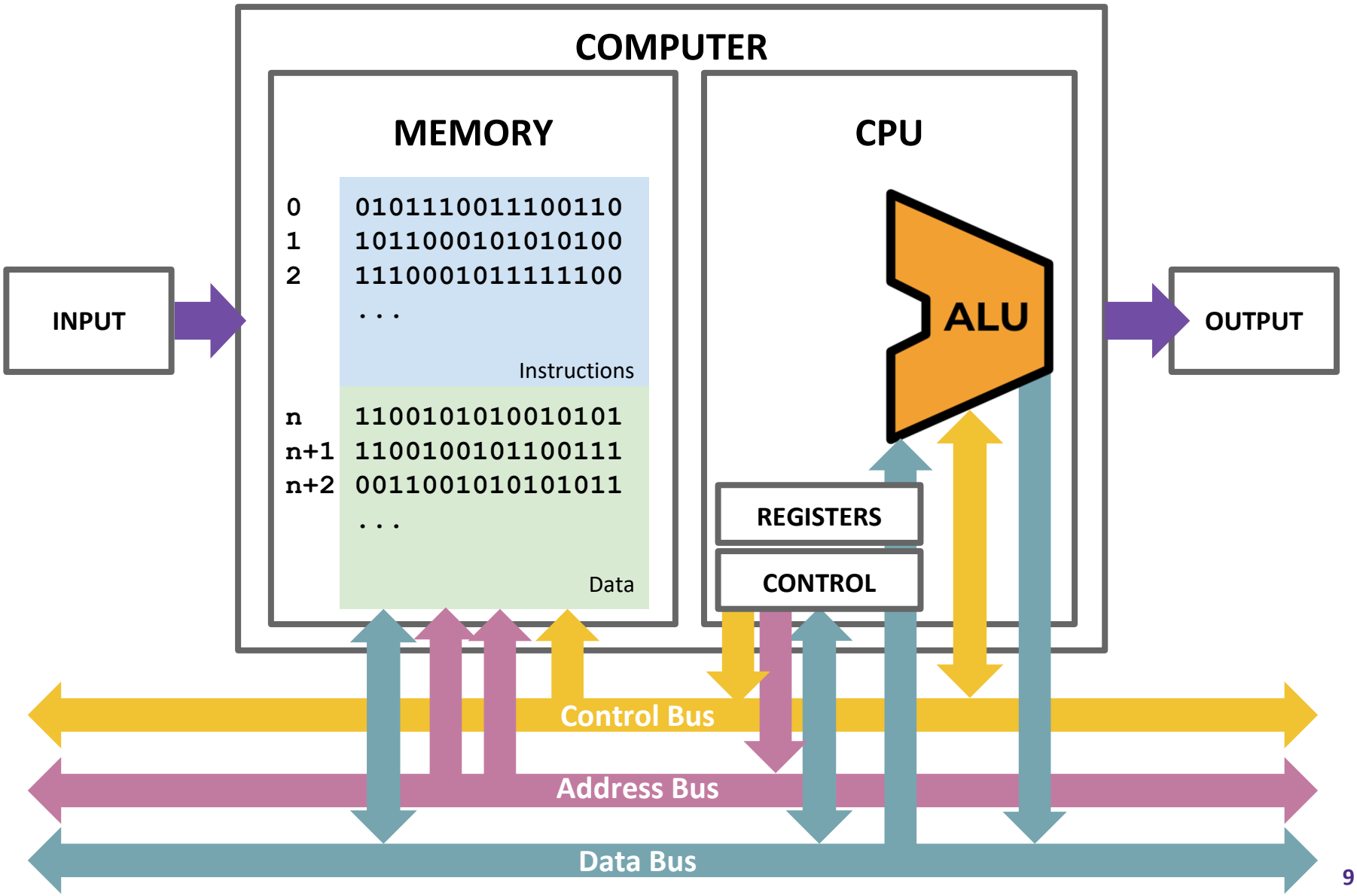
- In Project 6, you will build **`Computer.hdl`**, the final, top-level chip in this course
  - For all intents and purposes, a real computer
  - Simplified, but organization very similar to your laptop

- Project 7 onward, we will write software to make it useful

Von Neumann Architecture



Connecting the Computer: Buses

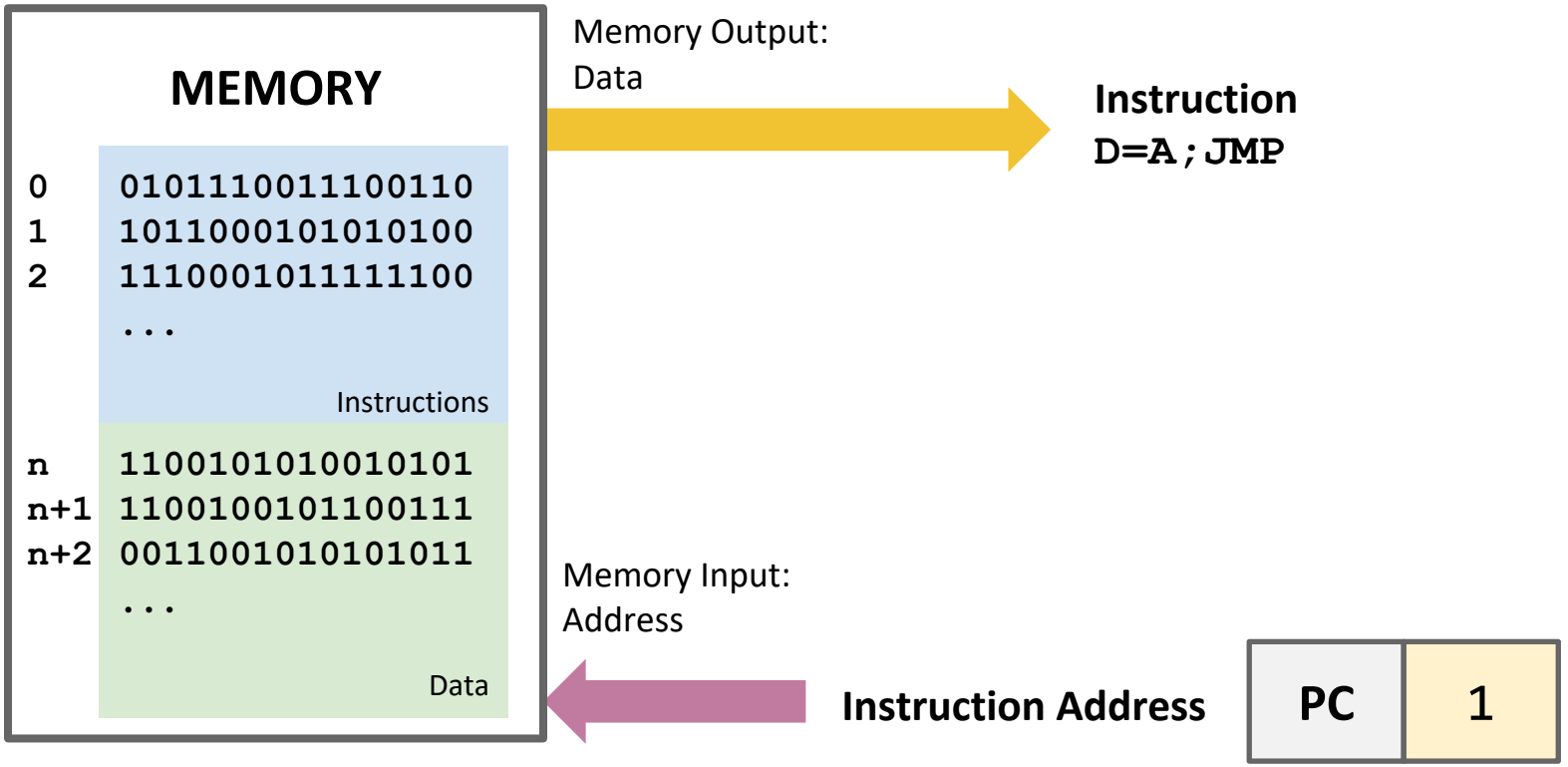


Basic CPU Loop

- 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

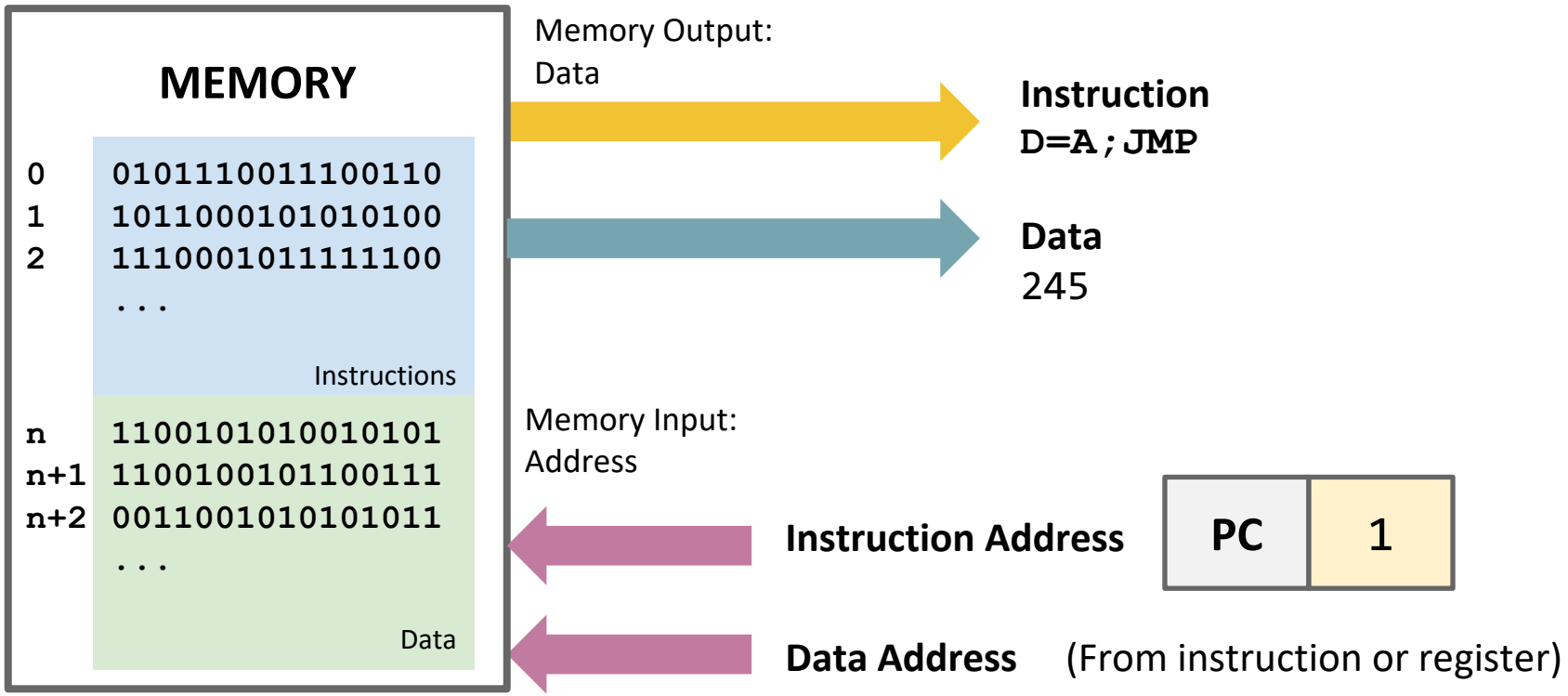


Executing

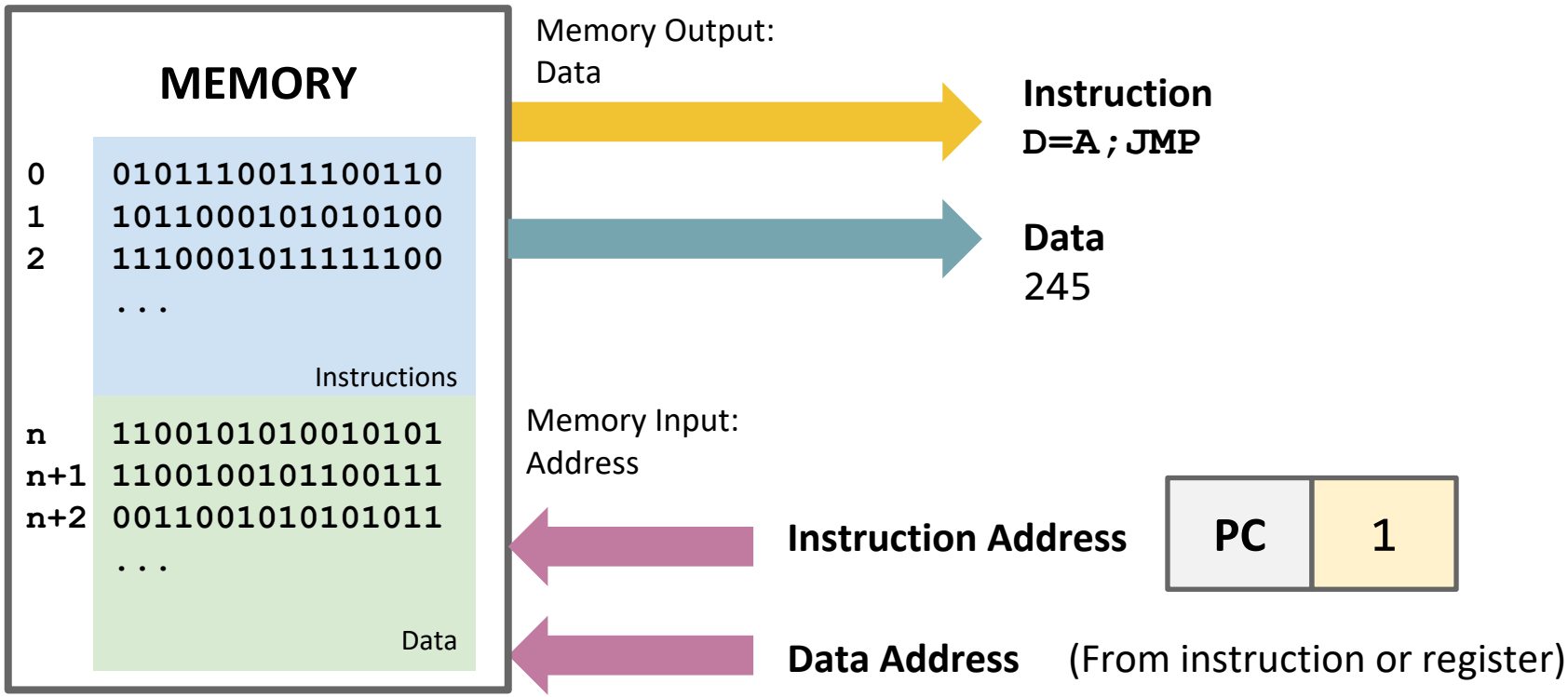
- The instruction bits describe exactly "what to do"
  - A-instruction or C-instruction?
  - Which operation for the ALU?
  - What memory address to read? To write?
  - Should I jump after this instruction, and if so, where?

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

Combining Fetch & Execute

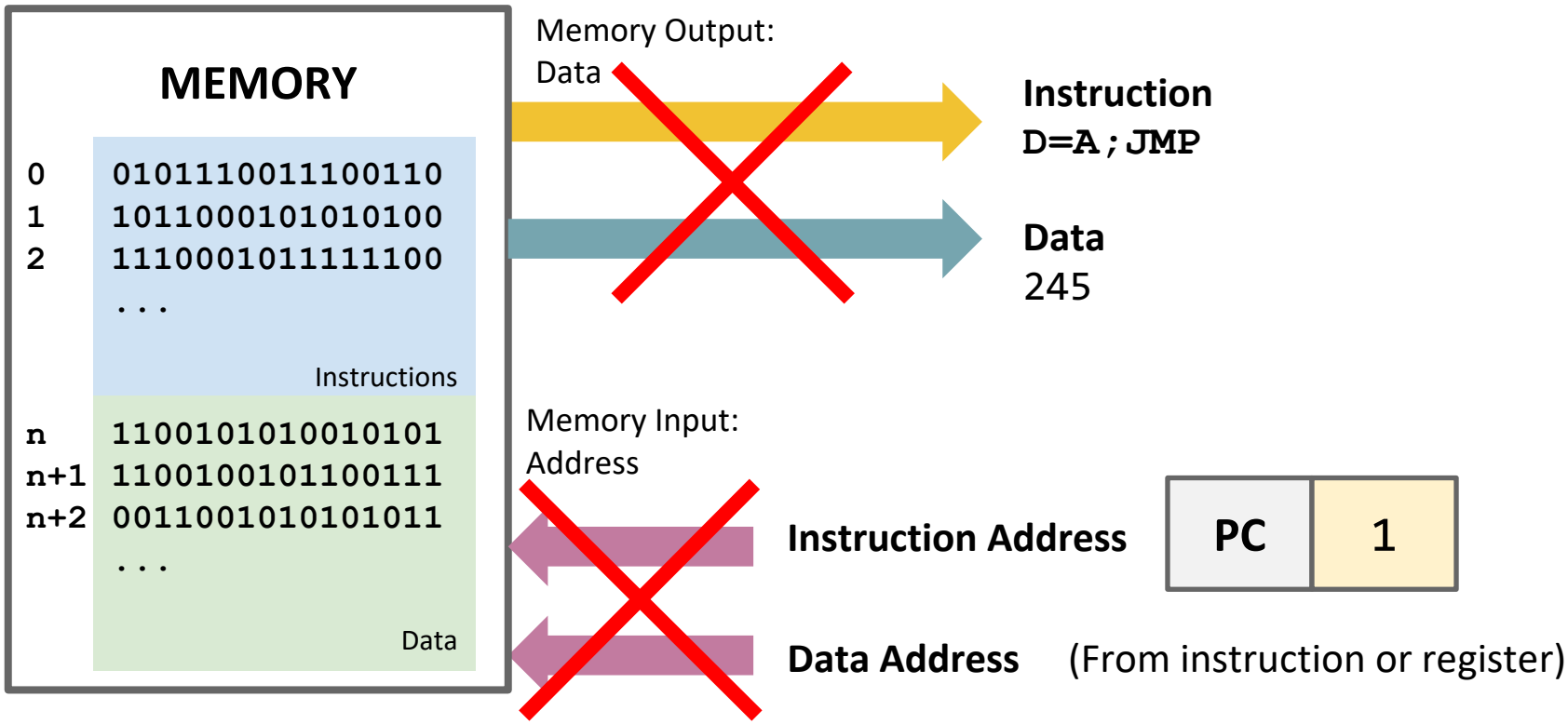


Combining Fetch & Execute



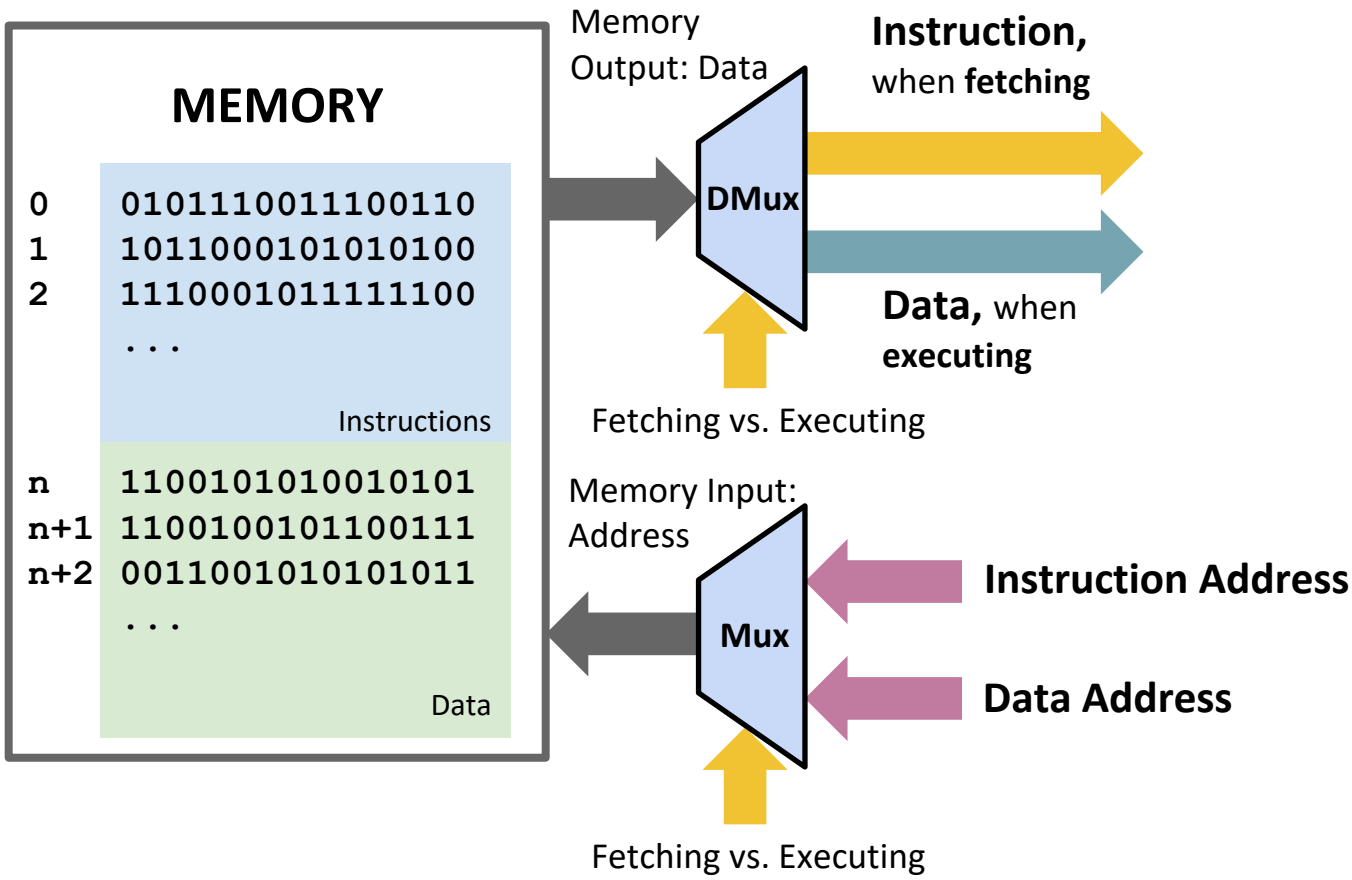
- Could we implement with **`RAM16K.hdl`**?

Combining Fetch & Execute



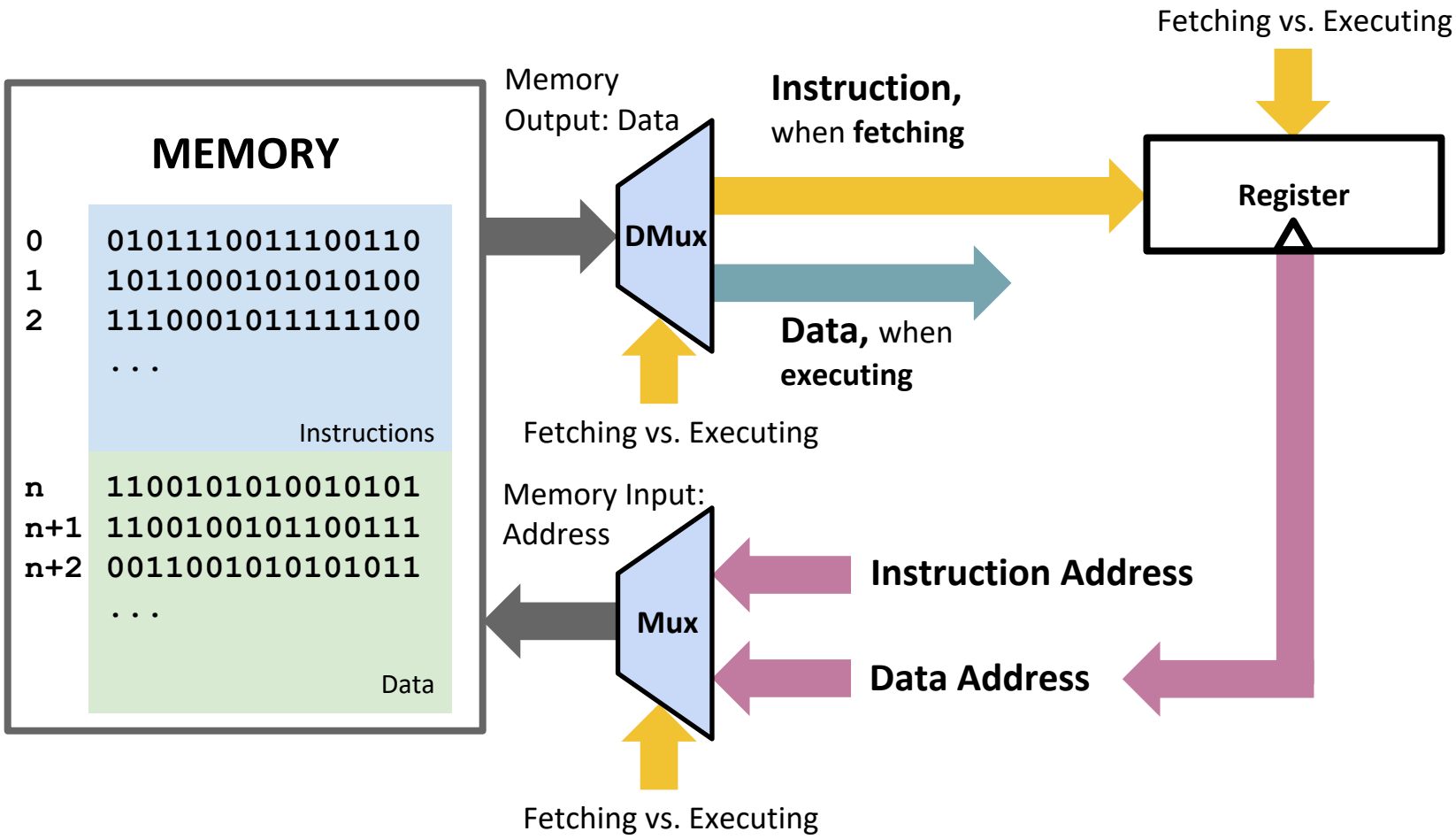
- Could we implement with **`RAM16K.hdl`**?
  - **No!** Our memory chips only have one address input and one output

Solution 1: Fetching / Executing Separately



- Can use multiplexing to share a single input or 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
- Pros:
  - Simpler to implement
- Cons:
  - Fixed size of each partition, rather than flexible storage
  - Two chips → redundant circuitry

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:
- 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

- Exam Preparation
  - Study Strategies, Mock Exam Problem
- Overview of Building a Computer
  - Architecture, Fetch and Execute Cycle
- **Hack CPU Logic**
  - **Implementation and Operations**

Poll Everywhere

Vote at https://pollev.com/cse390b

Which of the following statements about the Hack CPU is FALSE?

A. The Mux and ALU output is connected to the A register because it's a possible destination from a C instruction

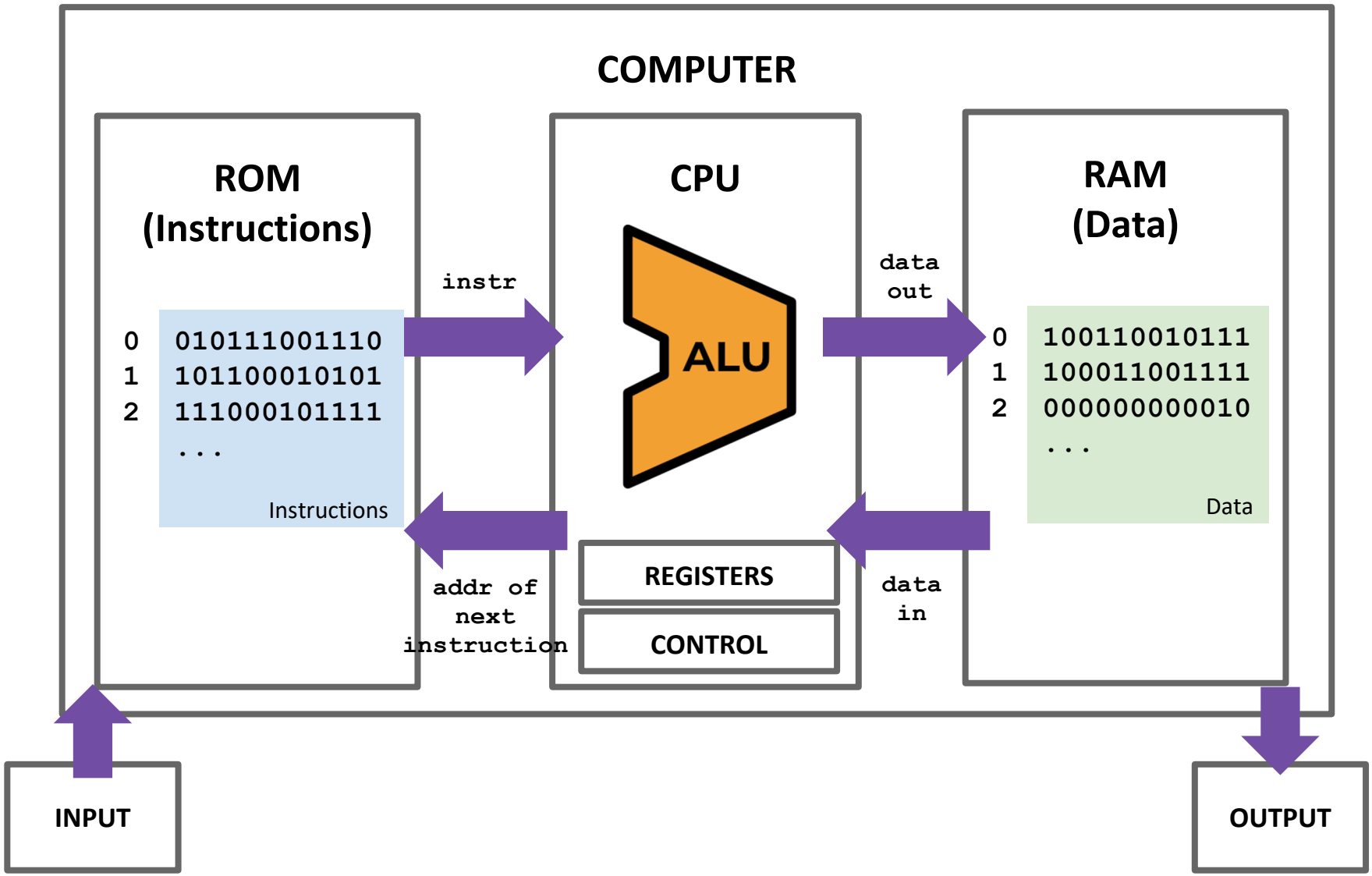
B. The A register and the M pseudo-register are never involved in a computation together

C. writeM determines whether the value should be updated in memory at addressM to outM

D. We feed the A register's output to the PC because this is the value that we increment

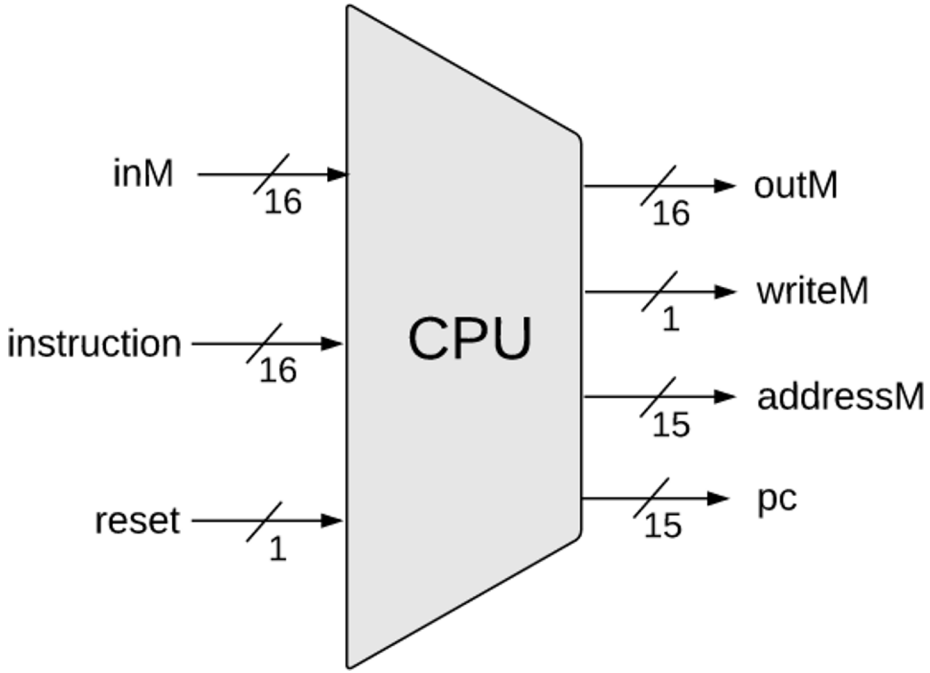
E. We're lost…

Hack CPU


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

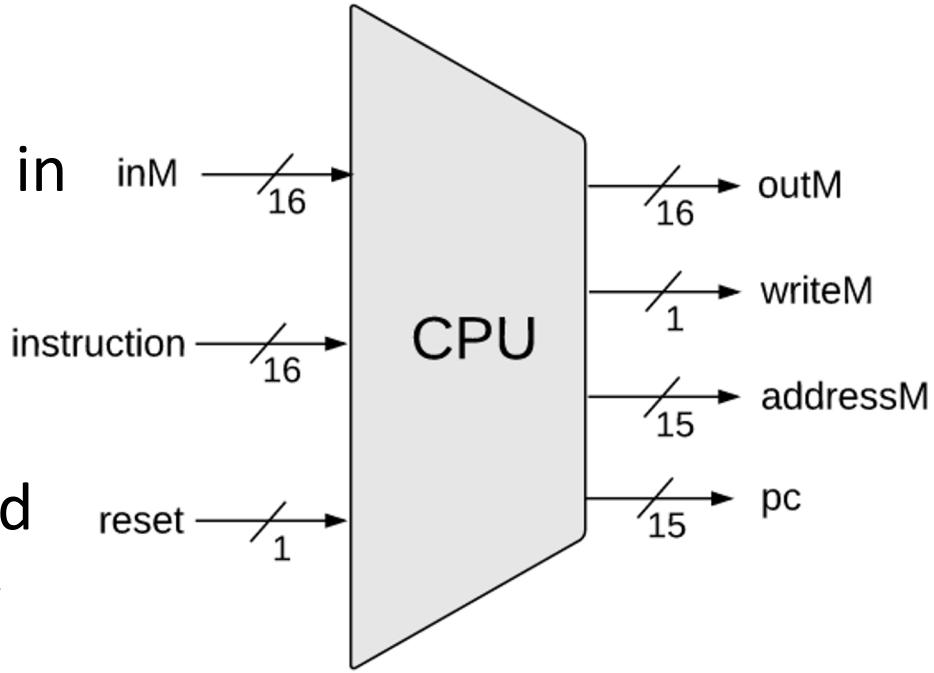
Hack CPU Interface Inputs

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

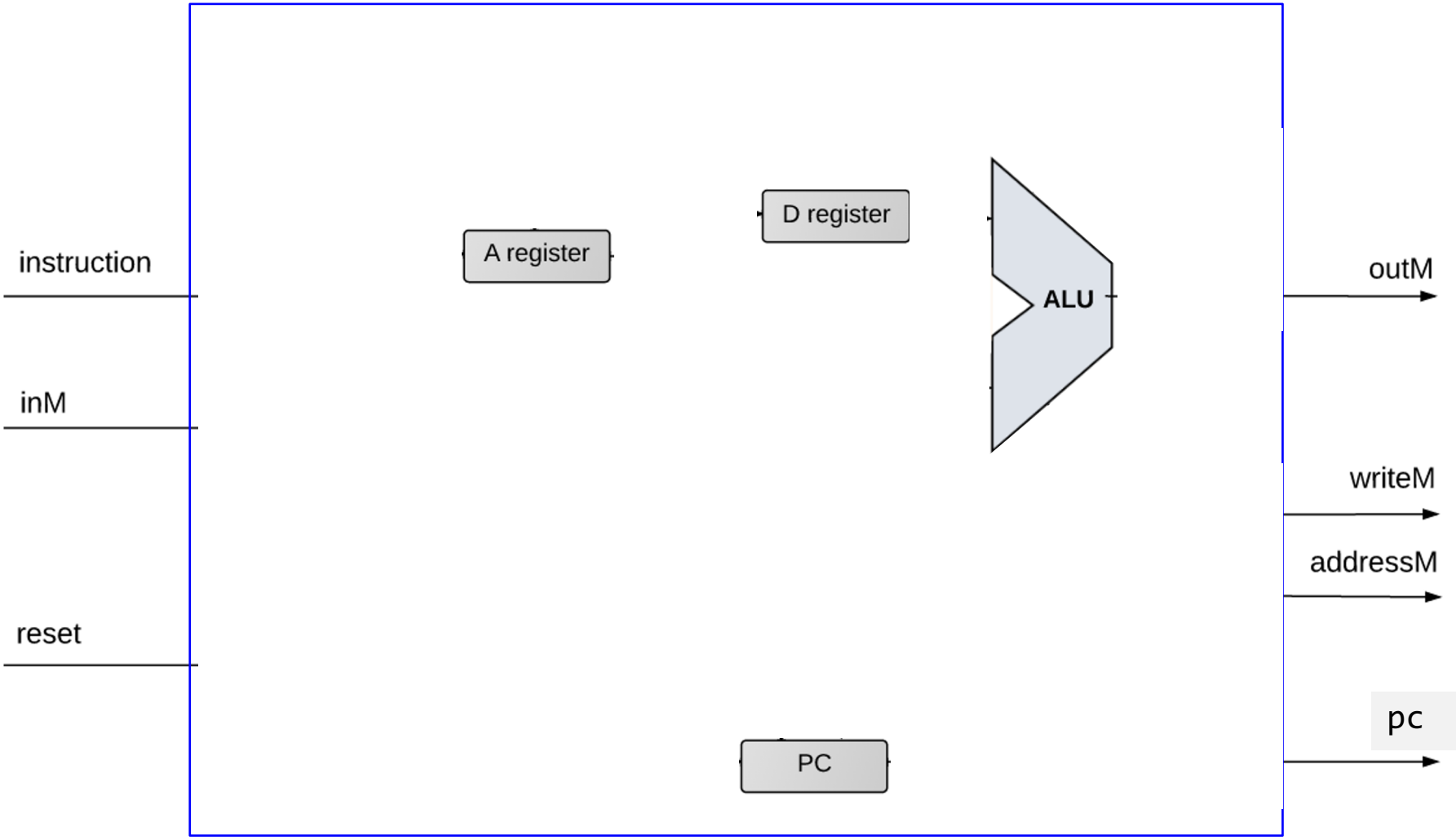


Hack CPU Interface Outputs

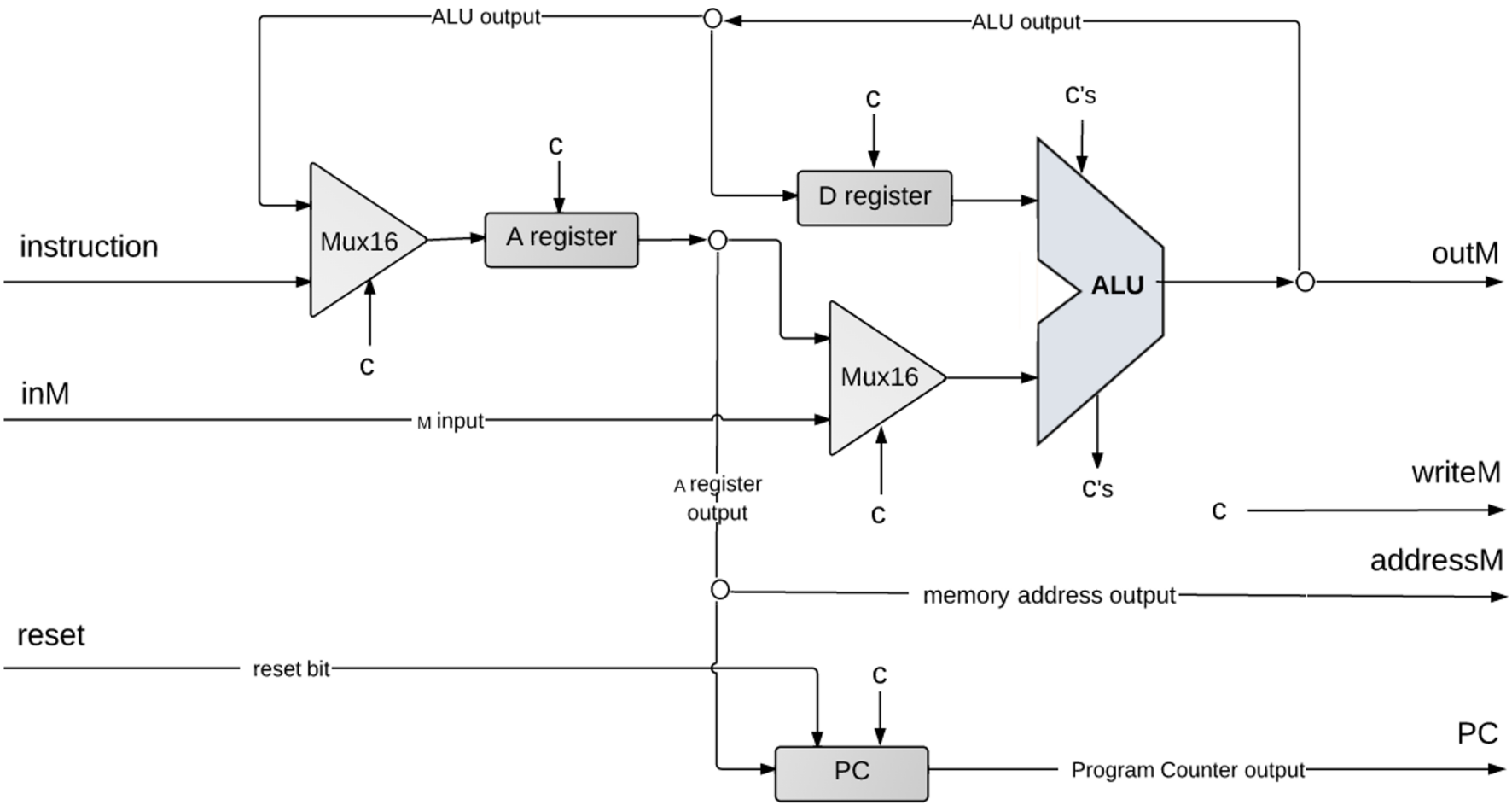
- **`outM`**: value used to update memory if writeM is 1
- **`writeM`**: if 1, update value in memory at addressM with outM
- **`addressM`**: address to read from or write to in memory
- **`pc`**: address of next instruction to be fetched from memory



Hack CPU Implementation

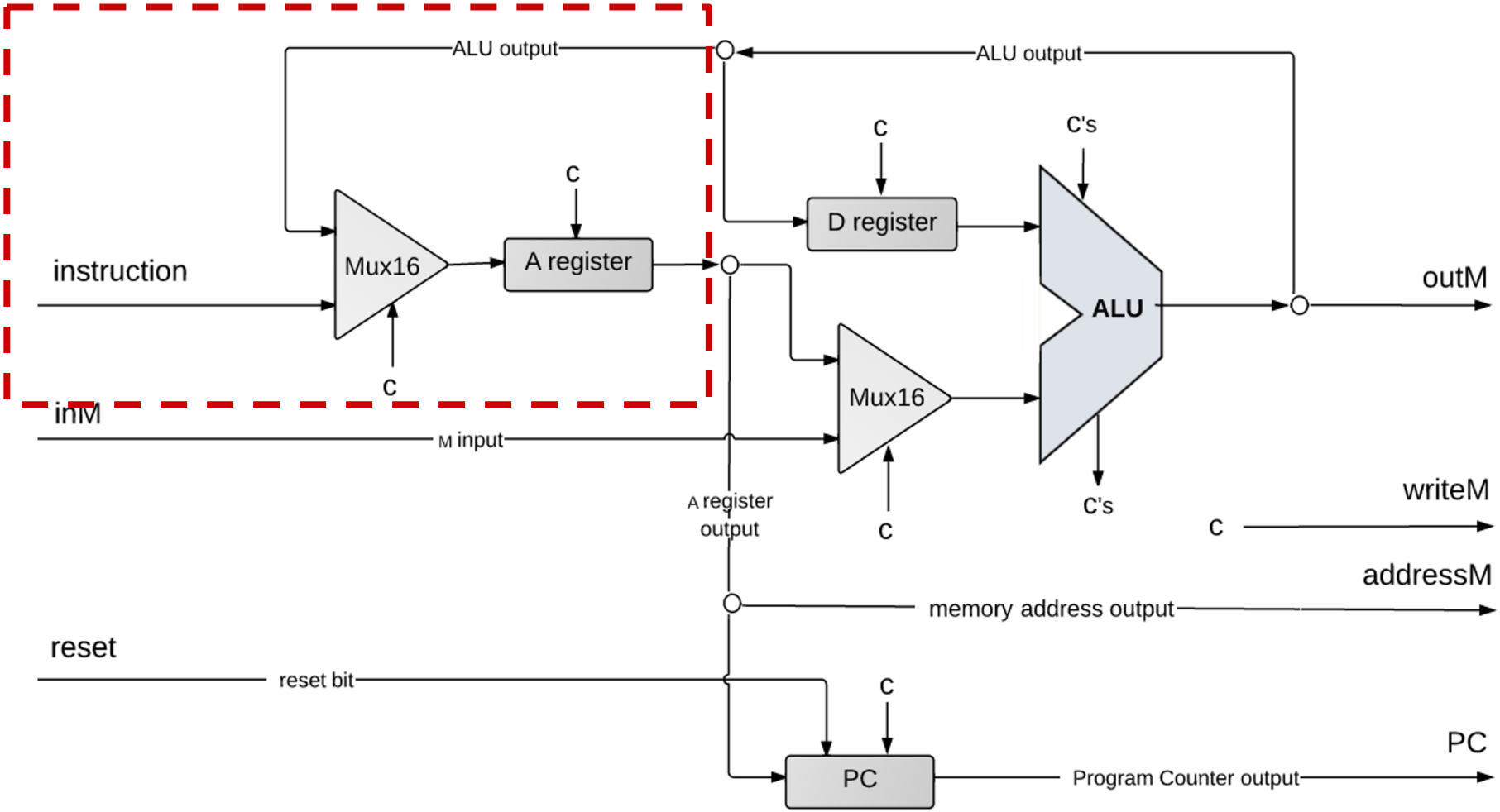


Hack CPU Implementation



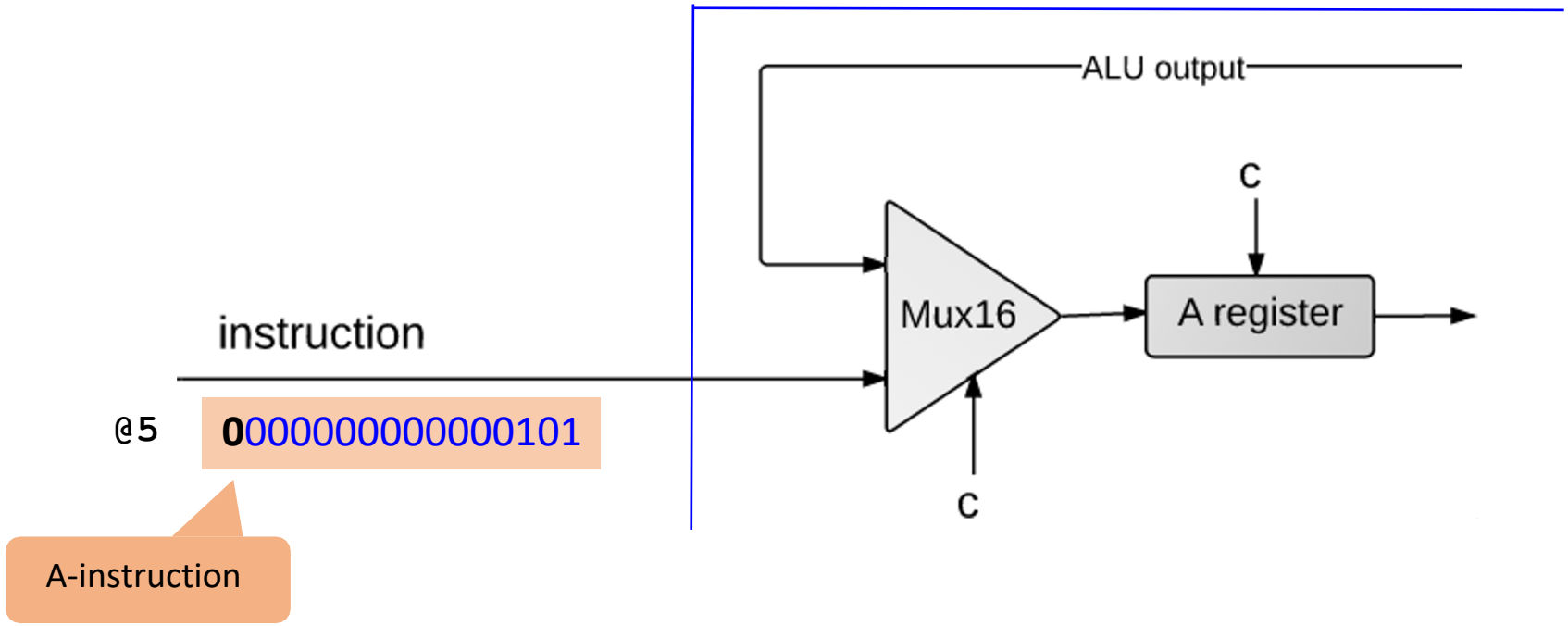
(each "C" symbol represents a control bit)

CPU Operation: Instruction Handling

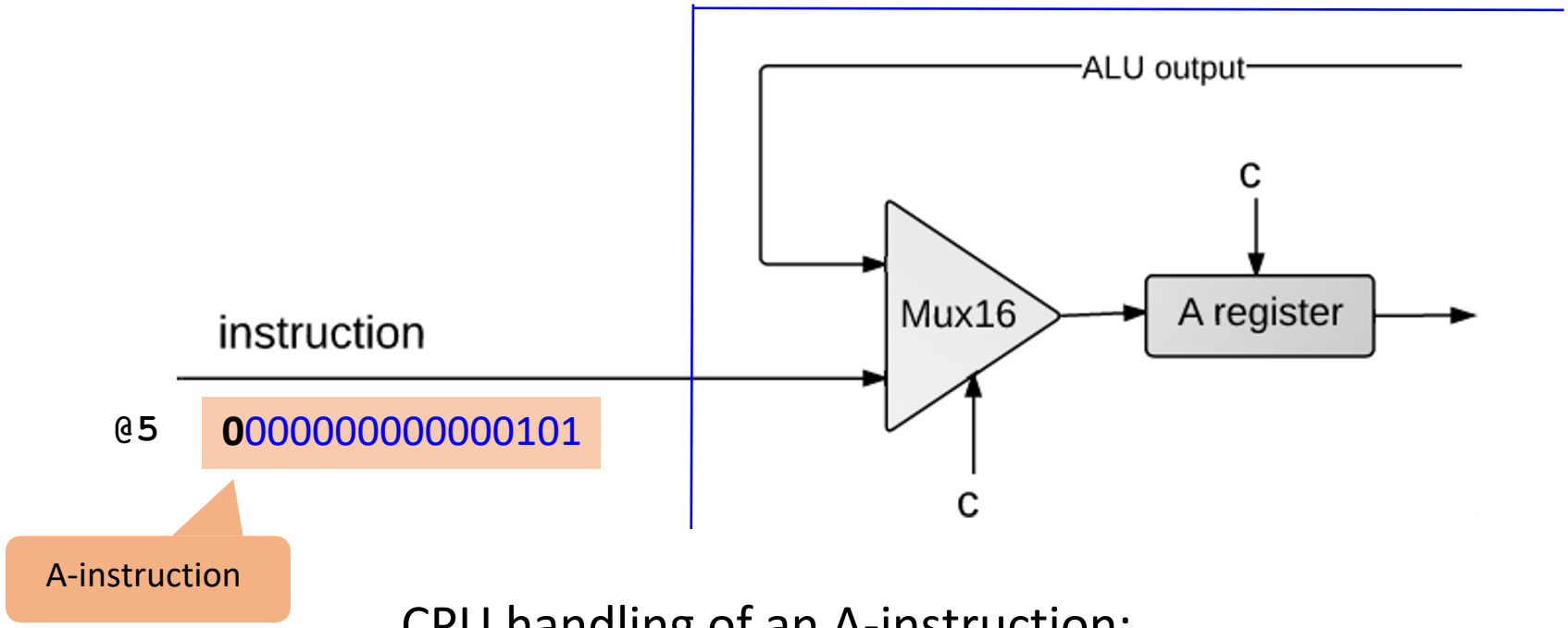


(each "C" symbol represents a control bit)

CPU Operation: Instruction Handling



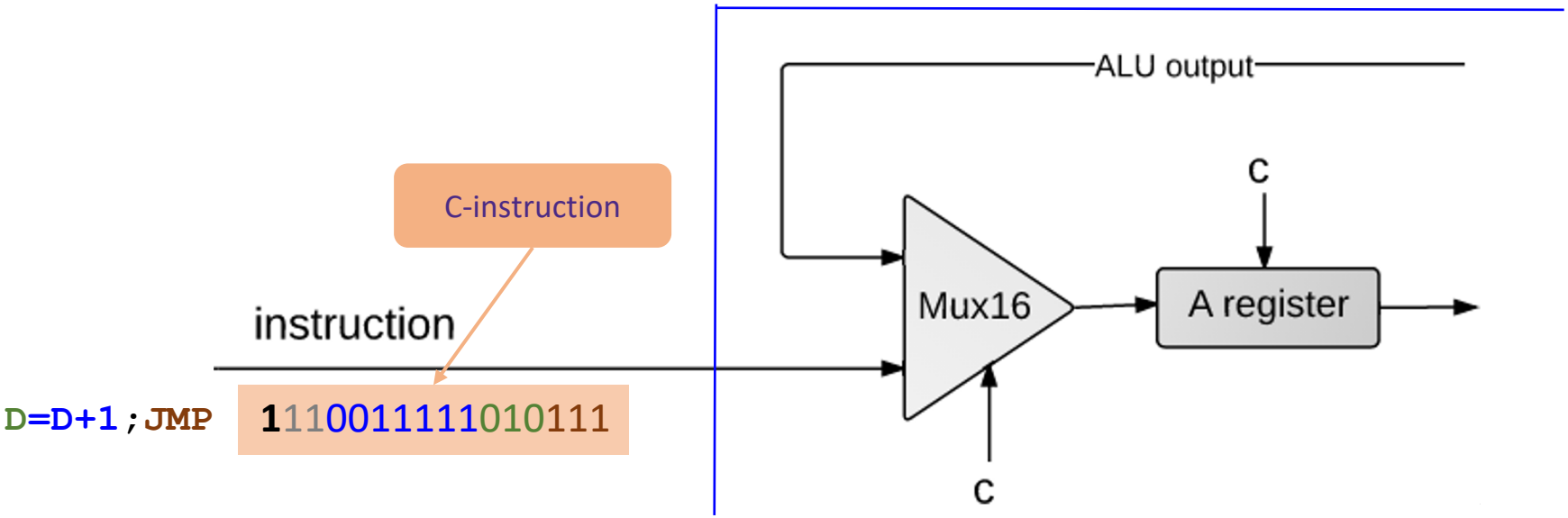
CPU Operation: Instruction Handling



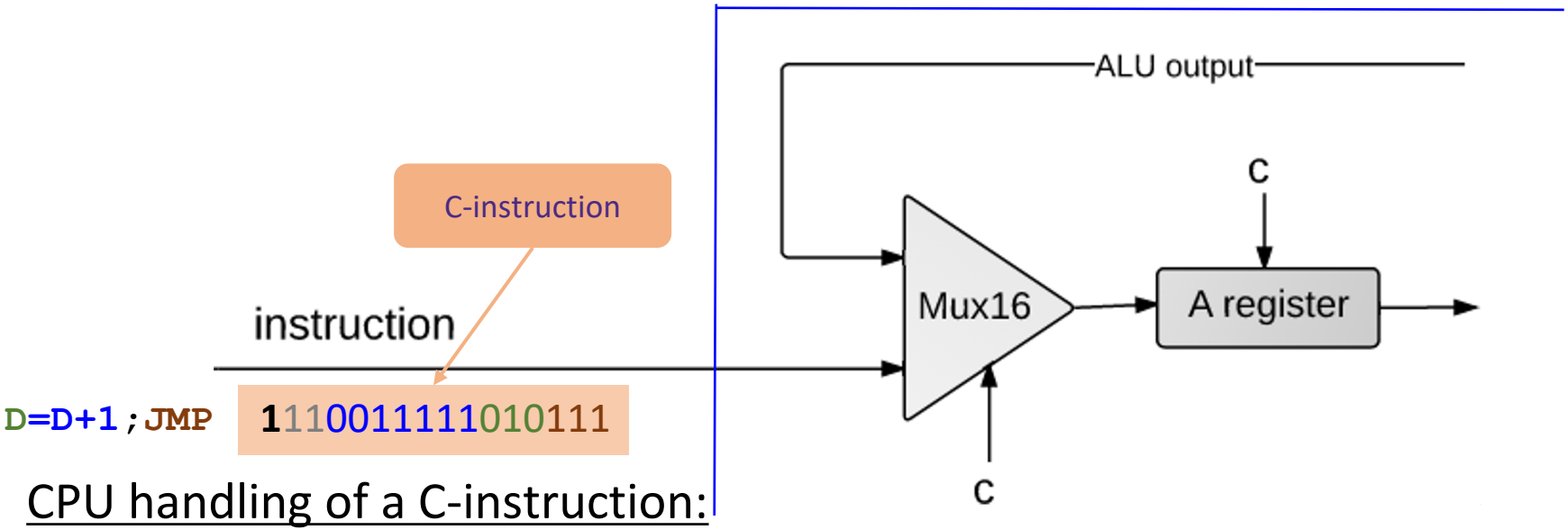
CPU handling of an A-instruction:

- Decodes the instruction into:
  - op-code
  - 15-bit value
- Stores the value in the A-register
- Outputs the value (not shown in this diagram)

CPU Operation: Instruction Handling



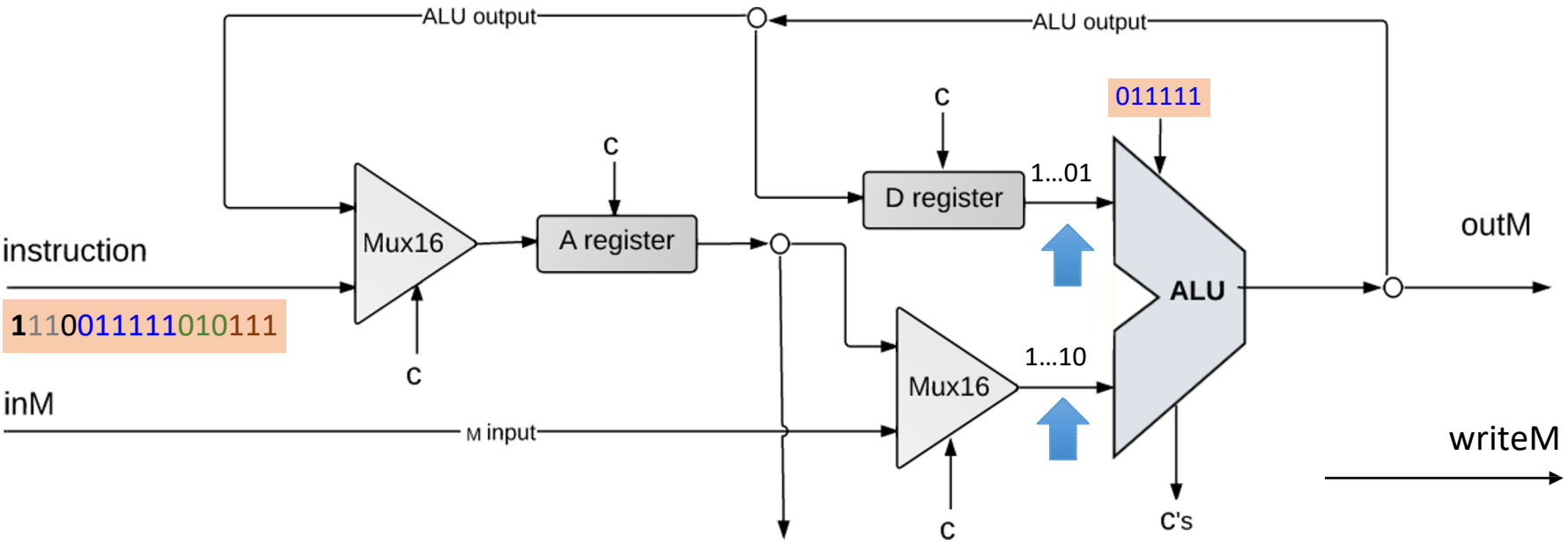
CPU Operation: Instruction Handling



CPU handling of a C-instruction:

- Decodes the instruction bits into:
  - Op-code
  - ALU control bits
  - Destination load bits
  - Jump bits
- Routes these bits to their chip-part destinations
- The chip-parts (most notably, the ALU) execute the instruction

CPU Operation: Handling C-Instructions



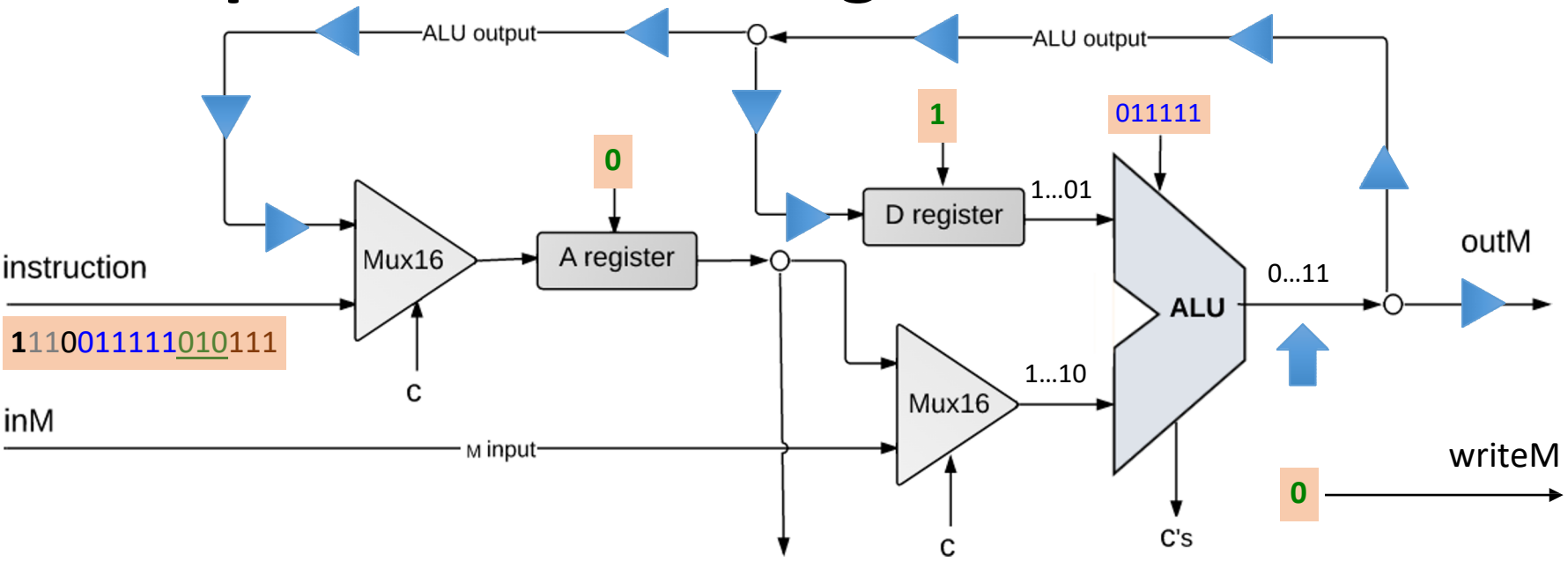
ALU data inputs:

- Input 1: from the D-register
- Input 2: from either:
  - A-register, or
  - data memory

ALU control inputs:

- Control bits (from the instruction)

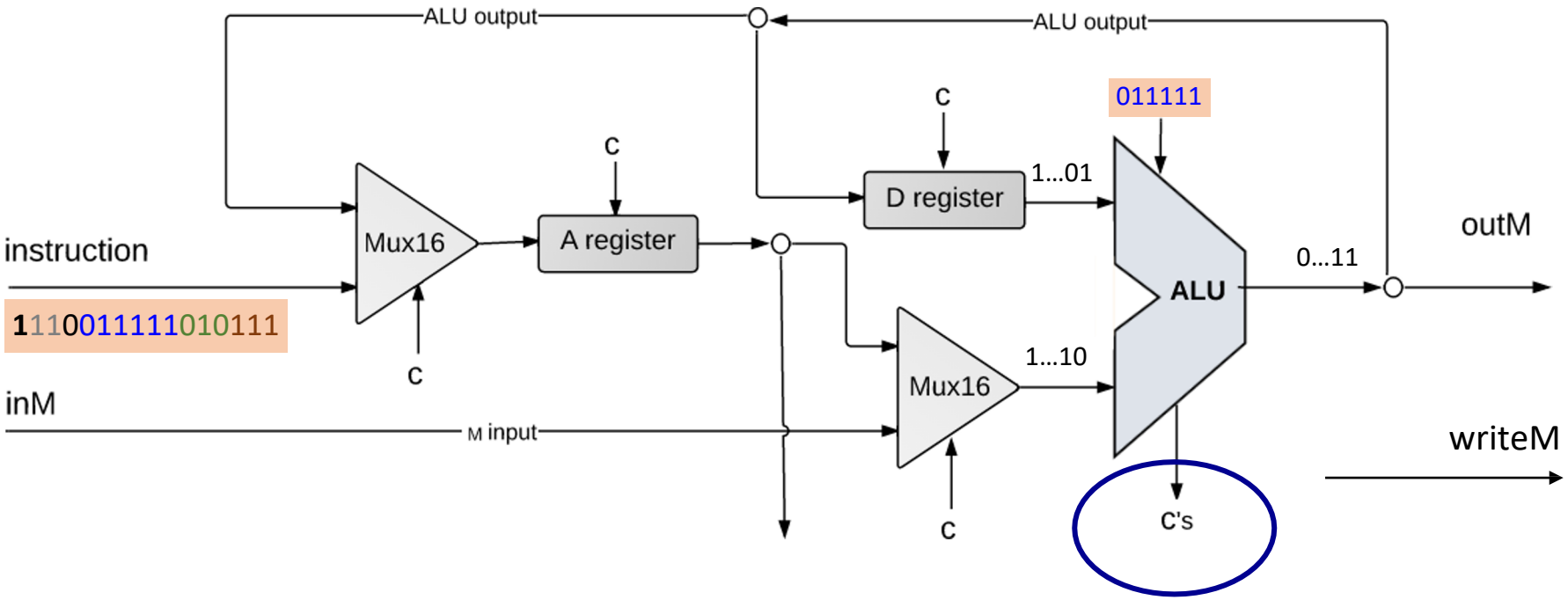
CPU Operation: Handling C-Instructions



ALU data output:

- Result of ALU calculation
- Fed simultaneously to: D-register, A-register, data memory
- Which destination *actually* commits to the ALU output is determined by the instruction's destination bits

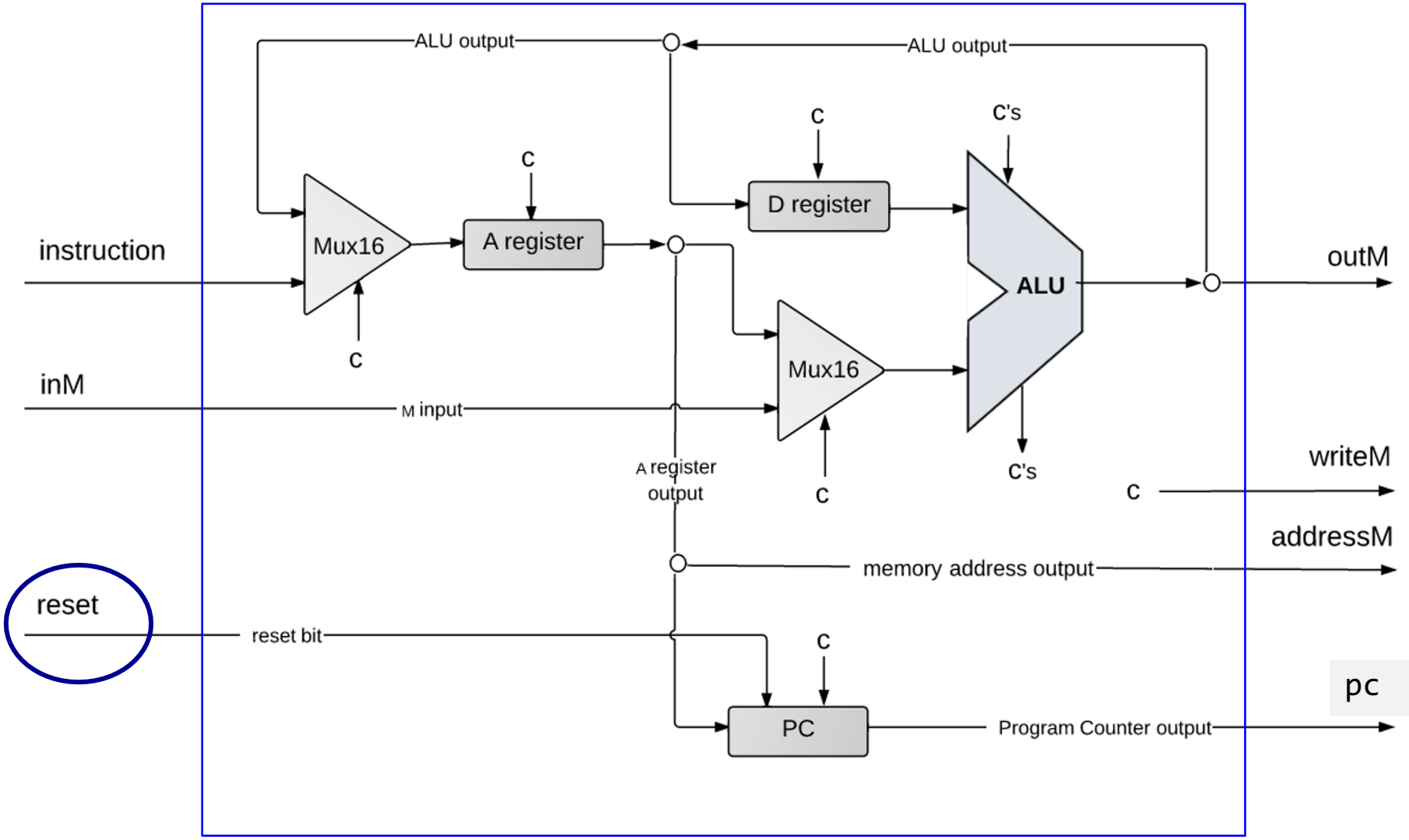
CPU Operation: Handling C-Instructions



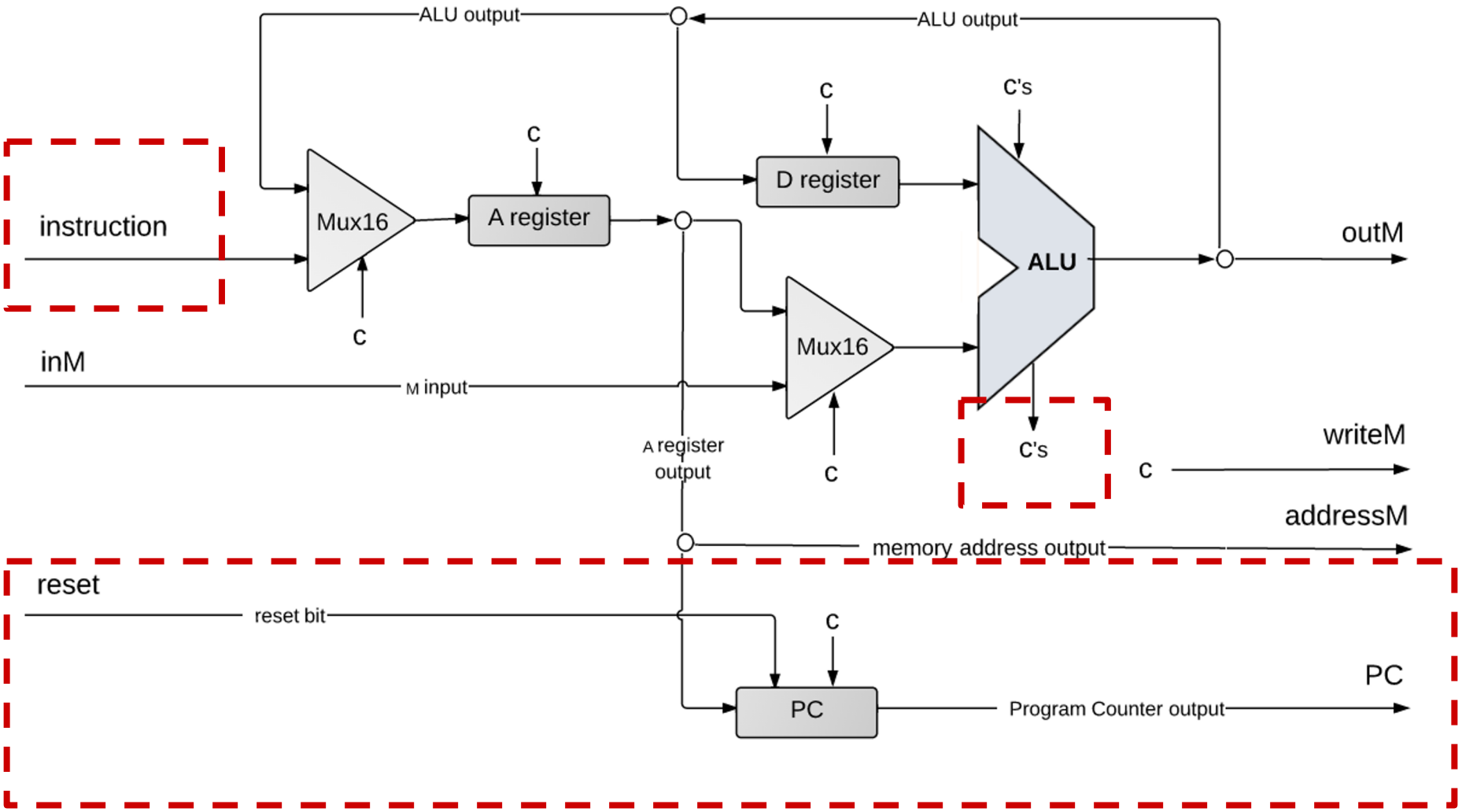
ALU control outputs:

- Is the output negative?
- Is the output zero?

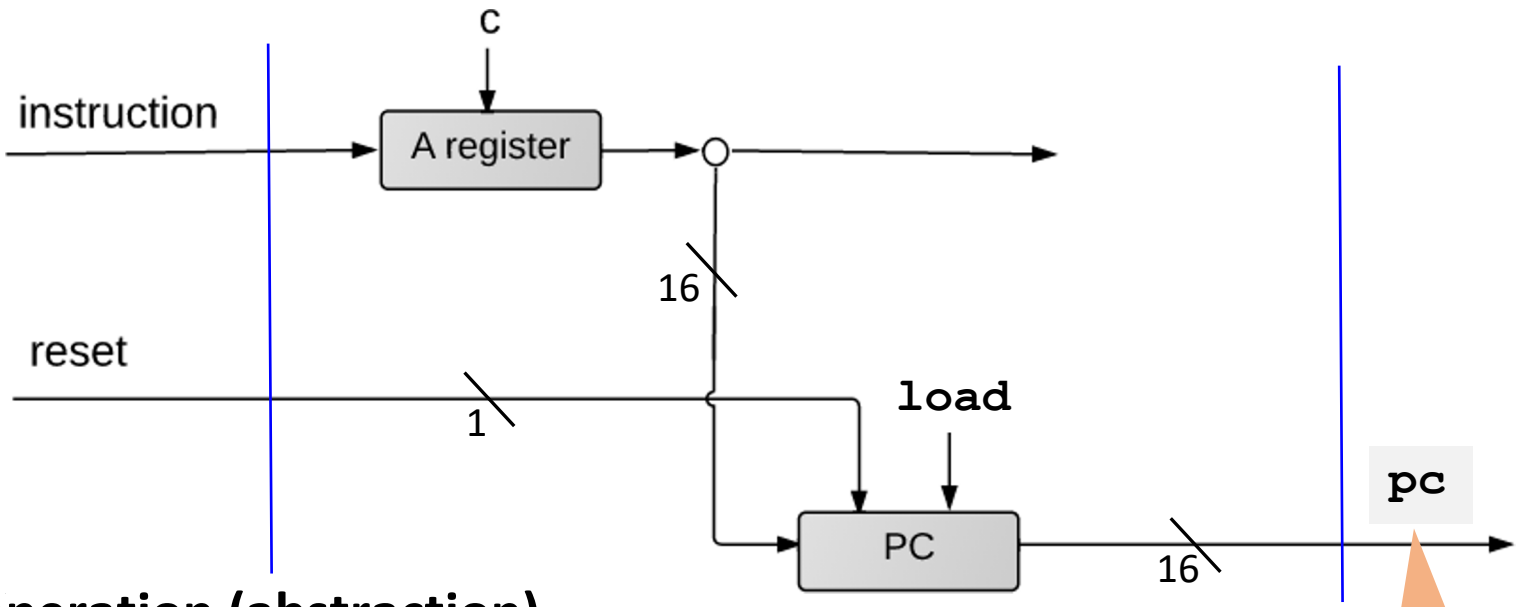
CPU Operation: Control



CPU Operation: Control



CPU Operation: Control

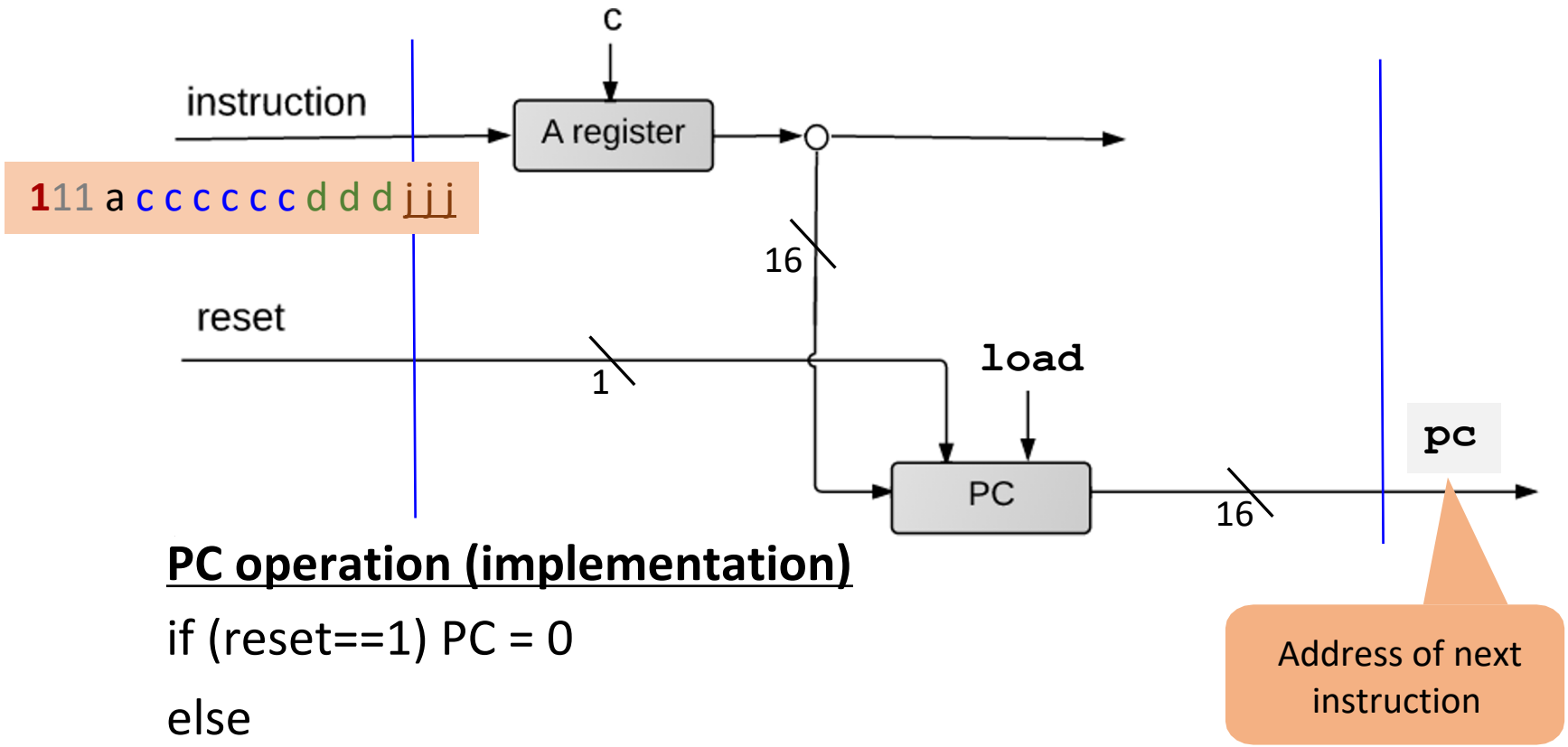


PC operation (abstraction)

Outputs the address of the next instruction:

- Restart: PC = 0
- No jump: PC++
- Go to: PC = A
- Conditional go to: if (condition) PC = A
  else PC ++

CPU Operation: Control



PC operation (implementation)

if (reset==1) PC = 0

else

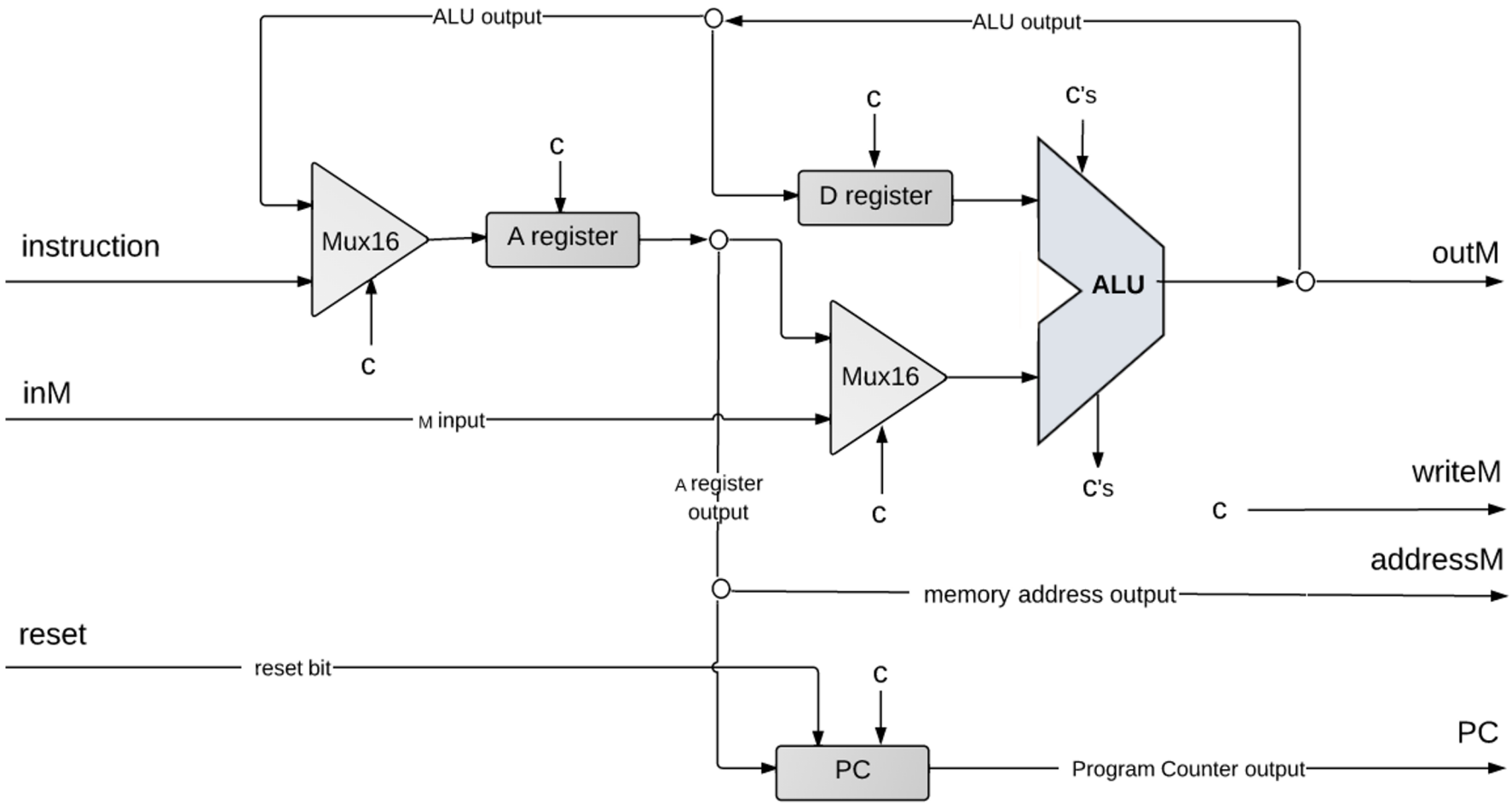
// In the course of handling the current instruction:

load = *f* (jump bits, ALU control outputs)

if (load == 1) PC = A // jump

else PC++ // next instruction

Hack CPU Implementation: That's It!



Lecture 9 Wrap-up

- Project Reminders
  - **Project 5 due Thursday (4/28) at 11:59pm PDT**
- CSE 390B Midterm Exam next Thursday (5/5) during lecture
- No reading for this Thursday's lecture