CSE 390B, Winter 2022

Building Academic Success Through Bottom-Up Computing

# Building a Computer, Exam Preparation

Cornell Note-Taking Debrief, Exam Preparation, Building a Computer Overview, Hack CPU Logic

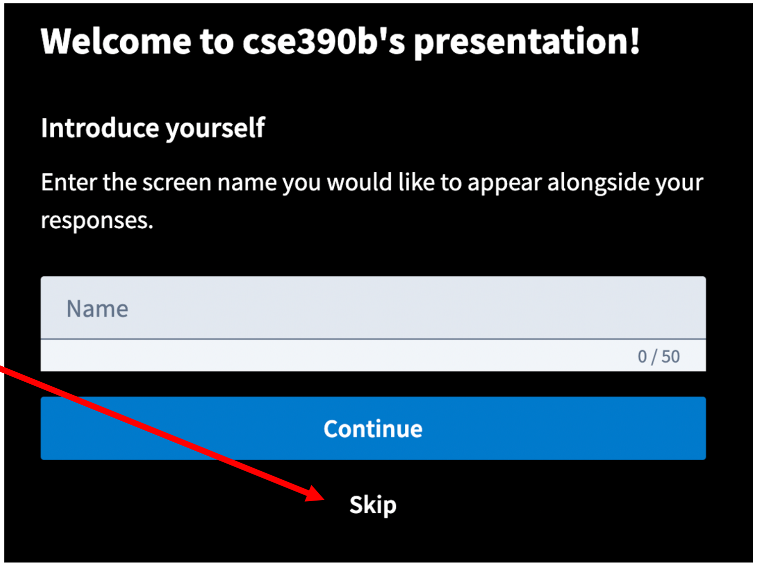
UNIVERSITY of WASHINGTON

# Poll Everywhere

Vote at https://pollev.com/cse390b

- **Would you like to keep CSE 390B's original finals time (Friday, March 18th from 2:30-4:30pm) or would you prefer to move it earlier in finals week?**

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



# Lecture Outline

- **Cornell Note-Taking Debrief**
  - **Reflection on notes from CSE 390B and other courses**

- Exam Preparation
  - Study strategies, mock exam problem

- Building a Computer Overview
  - Architecture, fetch and execute cycle

- Hack CPU Logic
  - Implementation and operations

# Project 3 Cornell Note-Taking Debrief

- *Look at your Cornell notes from CSE 390B <u>and</u> 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?

# Lecture Outline

- Cornell Note-Taking Debrief
  - Reflection on notes from CSE 390B and other courses
- **Exam Preparation**
  - **Study strategies, mock exam problem**
- Building a Computer Overview
  - Architecture, fetch and execute cycle
- Hack CPU Logic
  - Implementation and operations

# Gearing up for your 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 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 does 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 where your session will be located
- Post your group's meeting day & time to the Ed Board no later than Thursday

# 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 where your session will be located
- Post your group's meeting day & time to the Ed Board no later than Thursday

**PROJECT 5 GROUP ASSIGNMENTS**
**–**

**GROUP A:**
Eman, Hermona, & Kedist

**GROUP B:**
Sulaiman & Vasu

**GROUP C:**
Aynur & Preston

# Lecture Outline

- Cornell Note-Taking Debrief
  - Reflection on notes from CSE 390B and other courses
- Exam Preparation
  - Study strategies, mock exam problem
- **Building a Computer Overview**
  - **Architecture, fetch and execute cycle**
- Hack CPU Logic
  - Implementation and operations

# Building a Computer

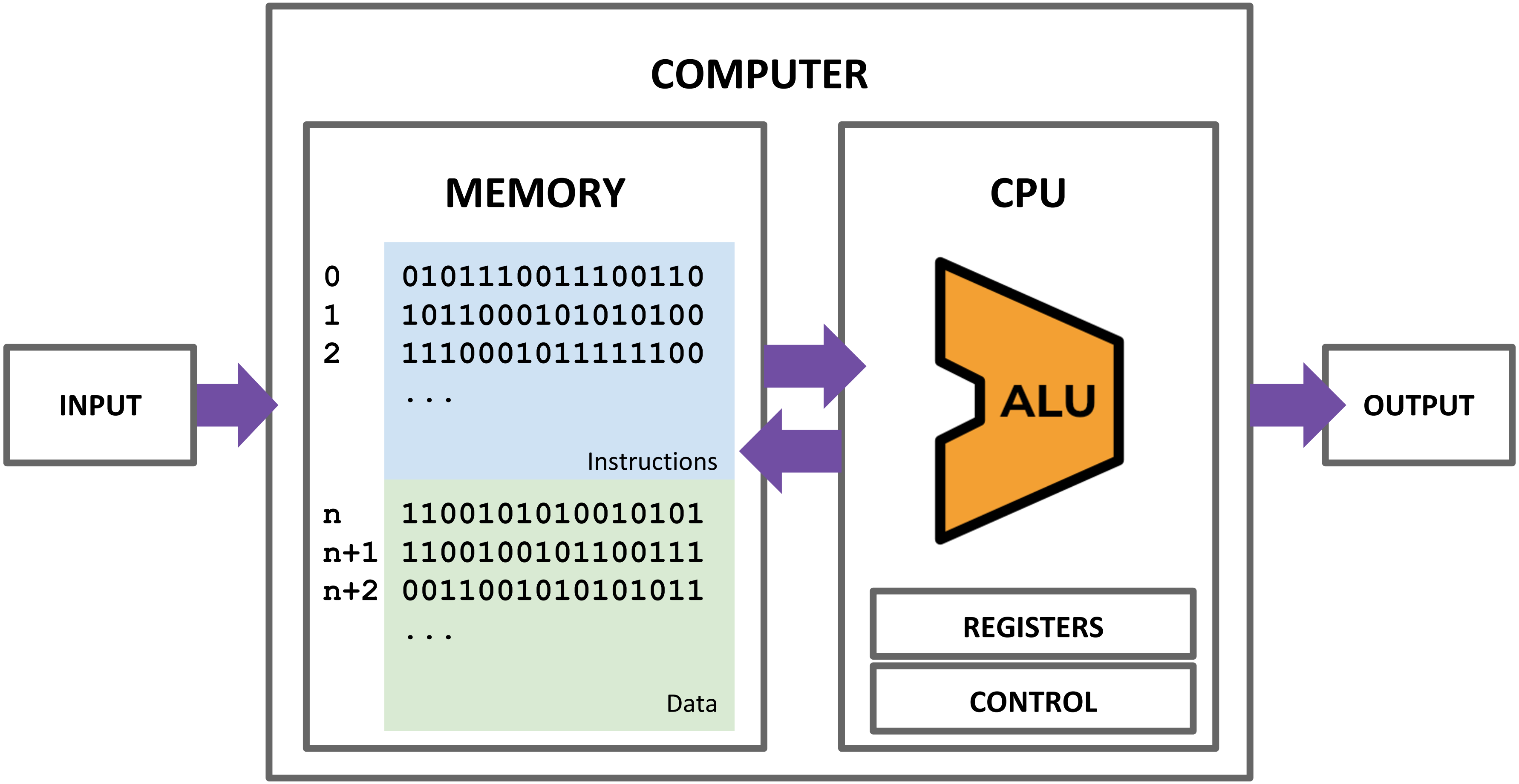
- Perspective: **BUILDING A COMPUTER**

- All your hardware efforts are about to pay off!

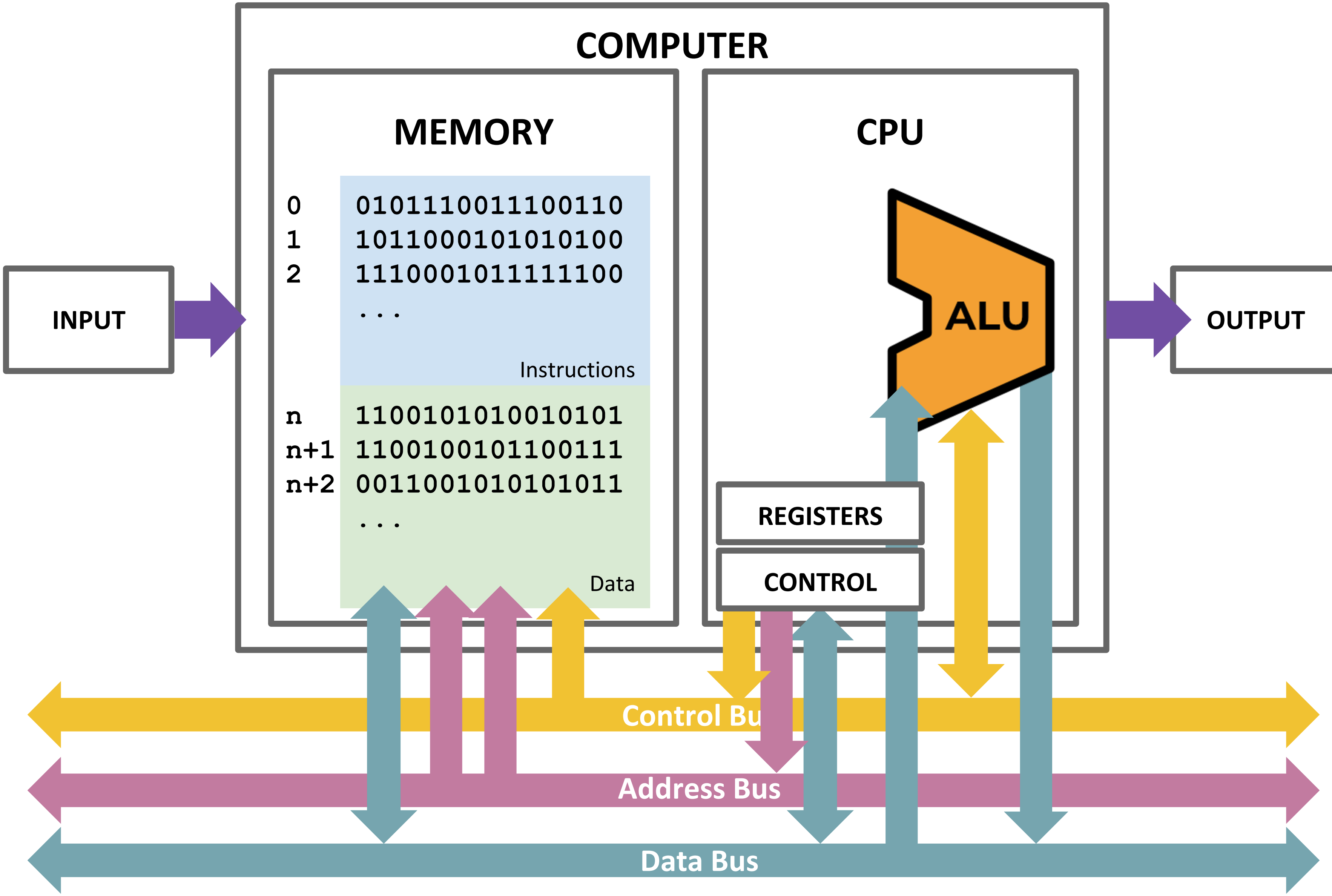
- In Project 5, you will build Computer.hdl—the final, top-level chip in this course
  - For all intents and purposes, a real computer
  - Simplified, but organization very similar to your laptop

- Project 6 onward, we will write software to make it useful

# Von Neumann Architecture



# Connecting the Computer: Buses

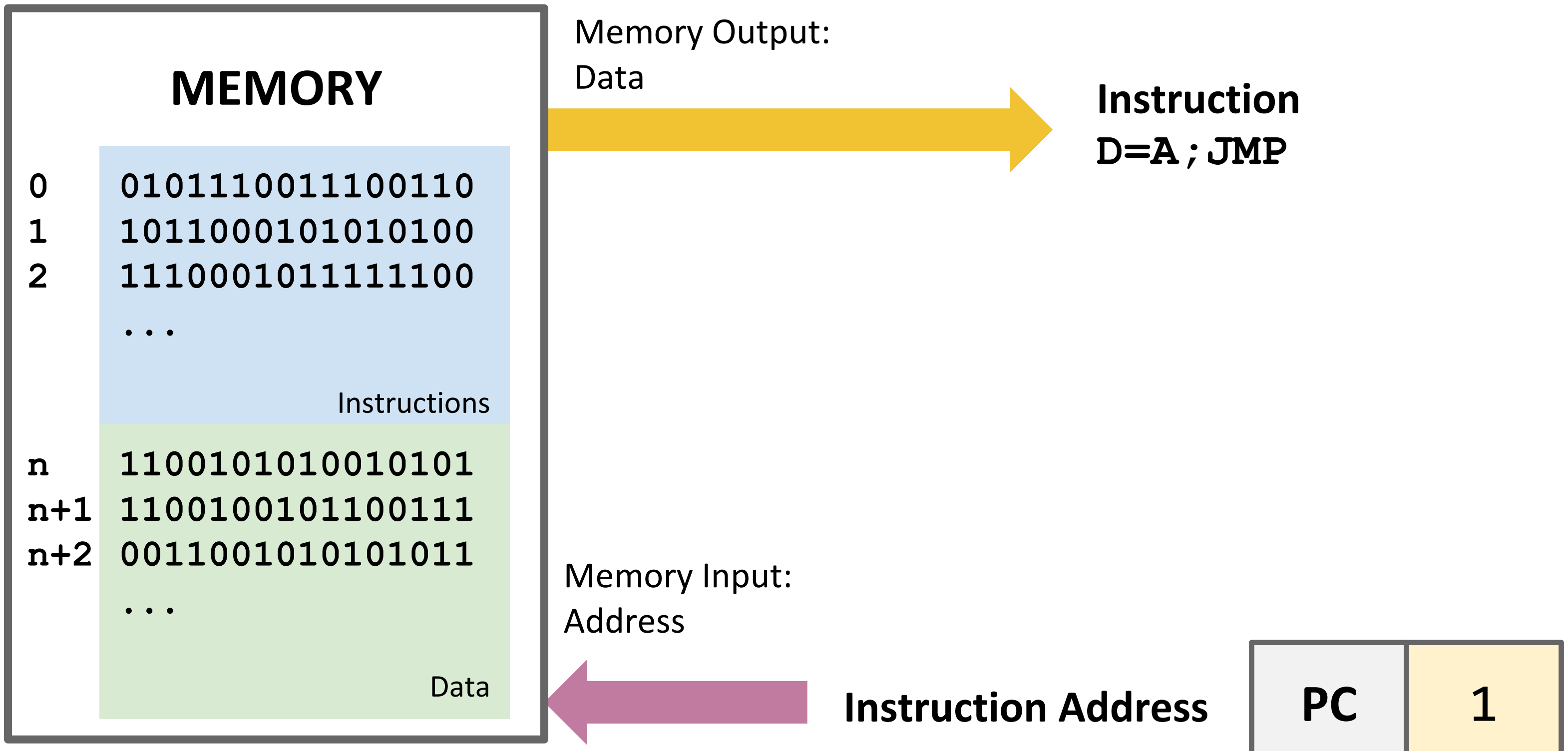


# Basic CPU Loop

- 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

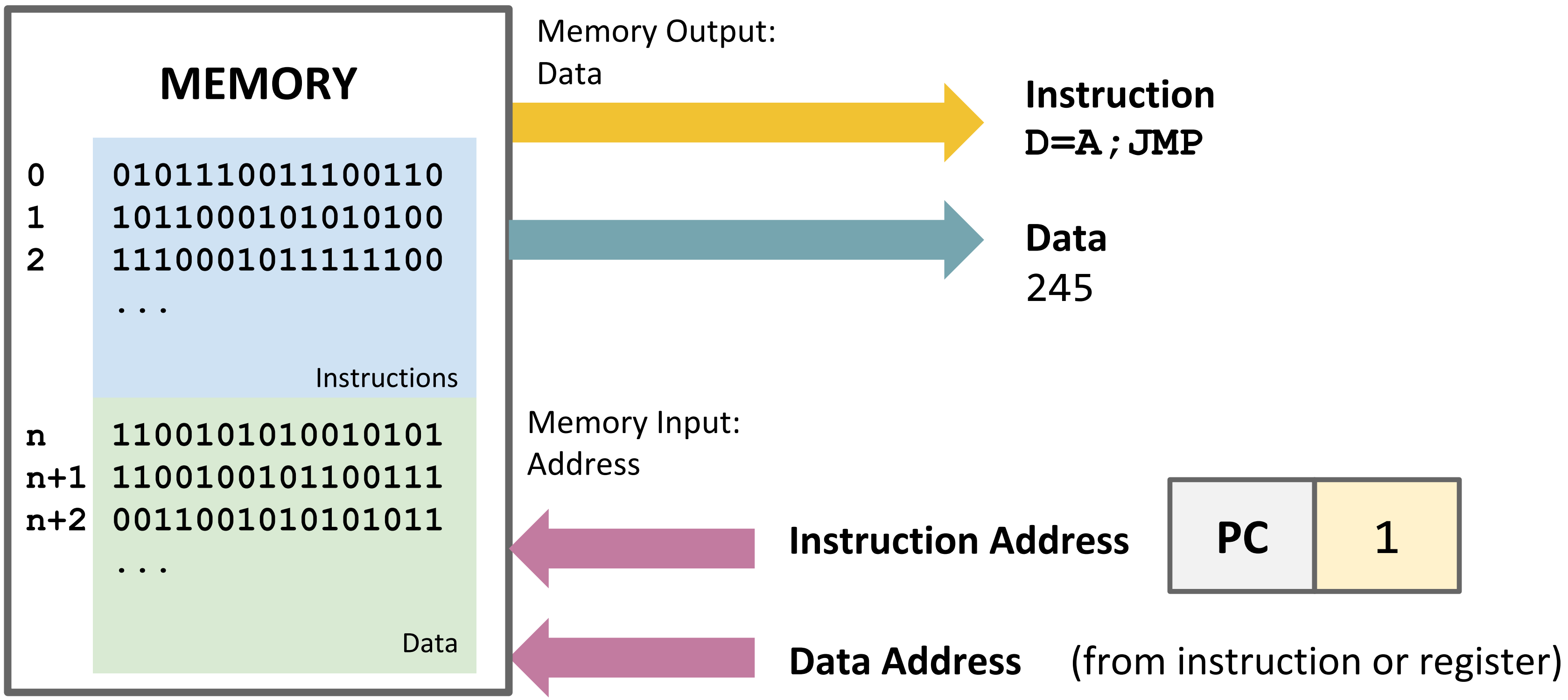


# Executing

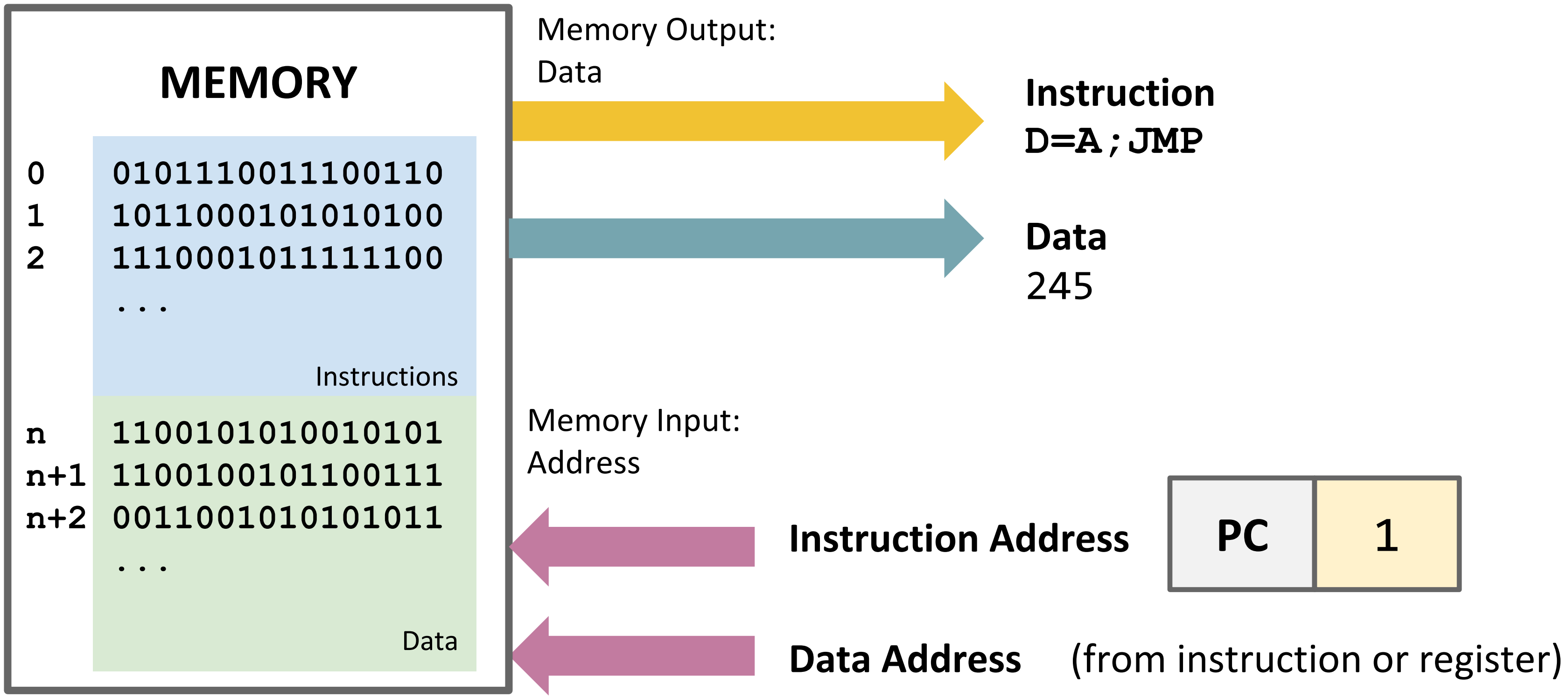
- The instruction bits describe exactly "what to do"
  - A-instruction or C-instruction?
  - Which operation for the ALU?
  - What memory address to read? To write?
  - If I should jump after this instruction, and where?

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

# Combining Fetch & Execute

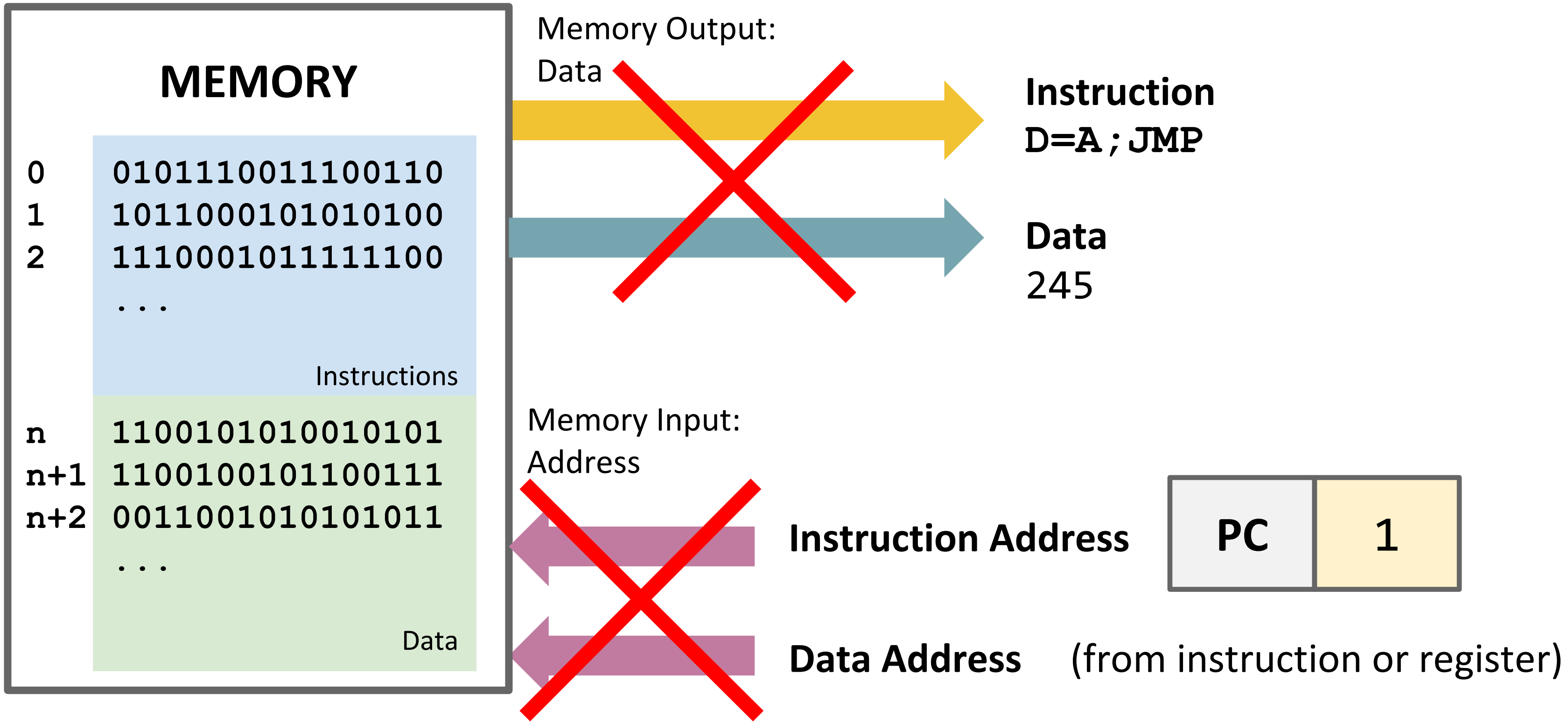


# Combining Fetch & Execute



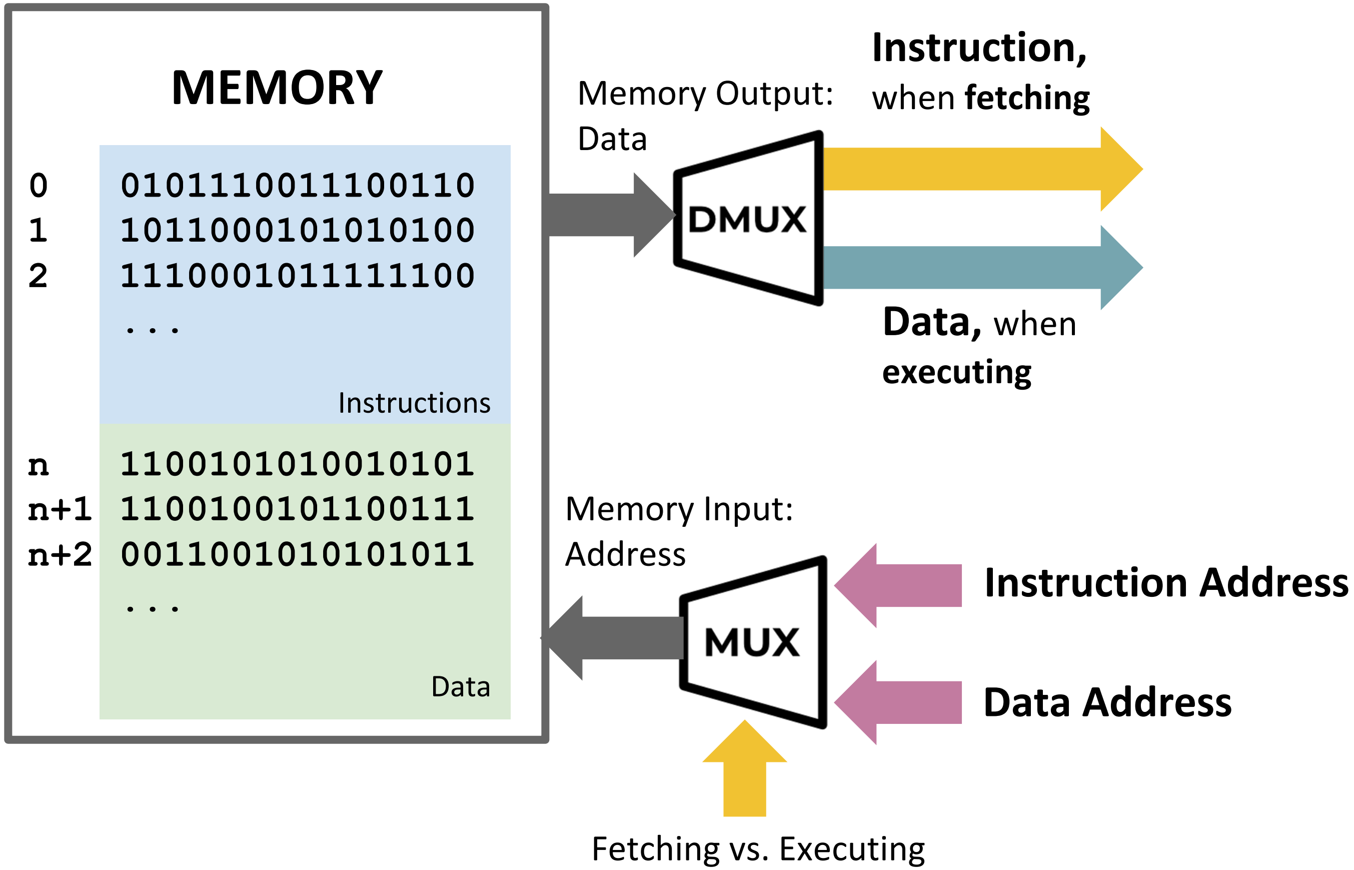

❖ Could we implement with **`RAM16K.hdl`**?

# Combining Fetch & Execute



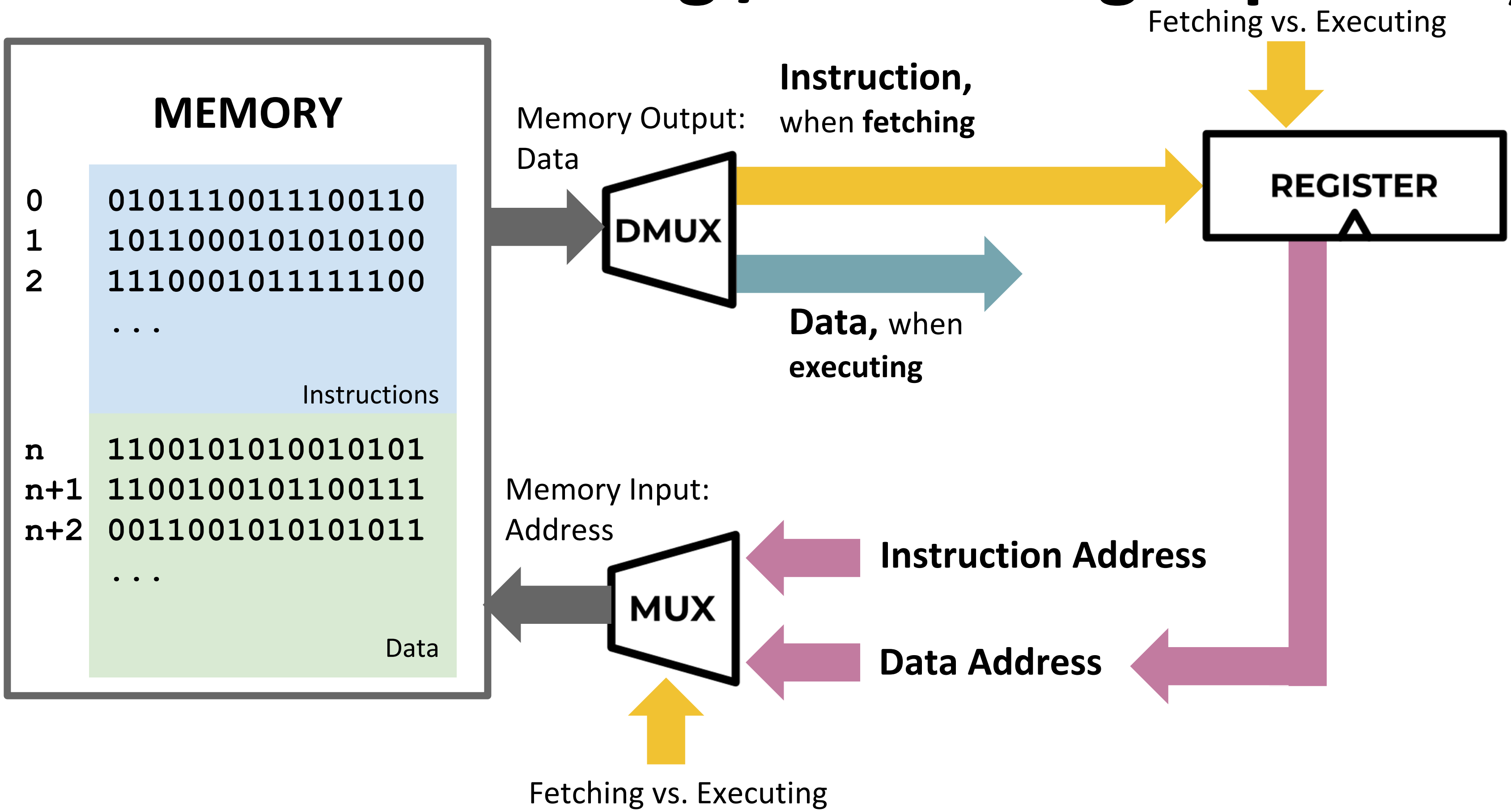

- Could we implement with **`RAM16K.hdl`**?
  - **No!** Our memory chips only have one input and one output!

# Solution 1: Handling Single Input / Output




- 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

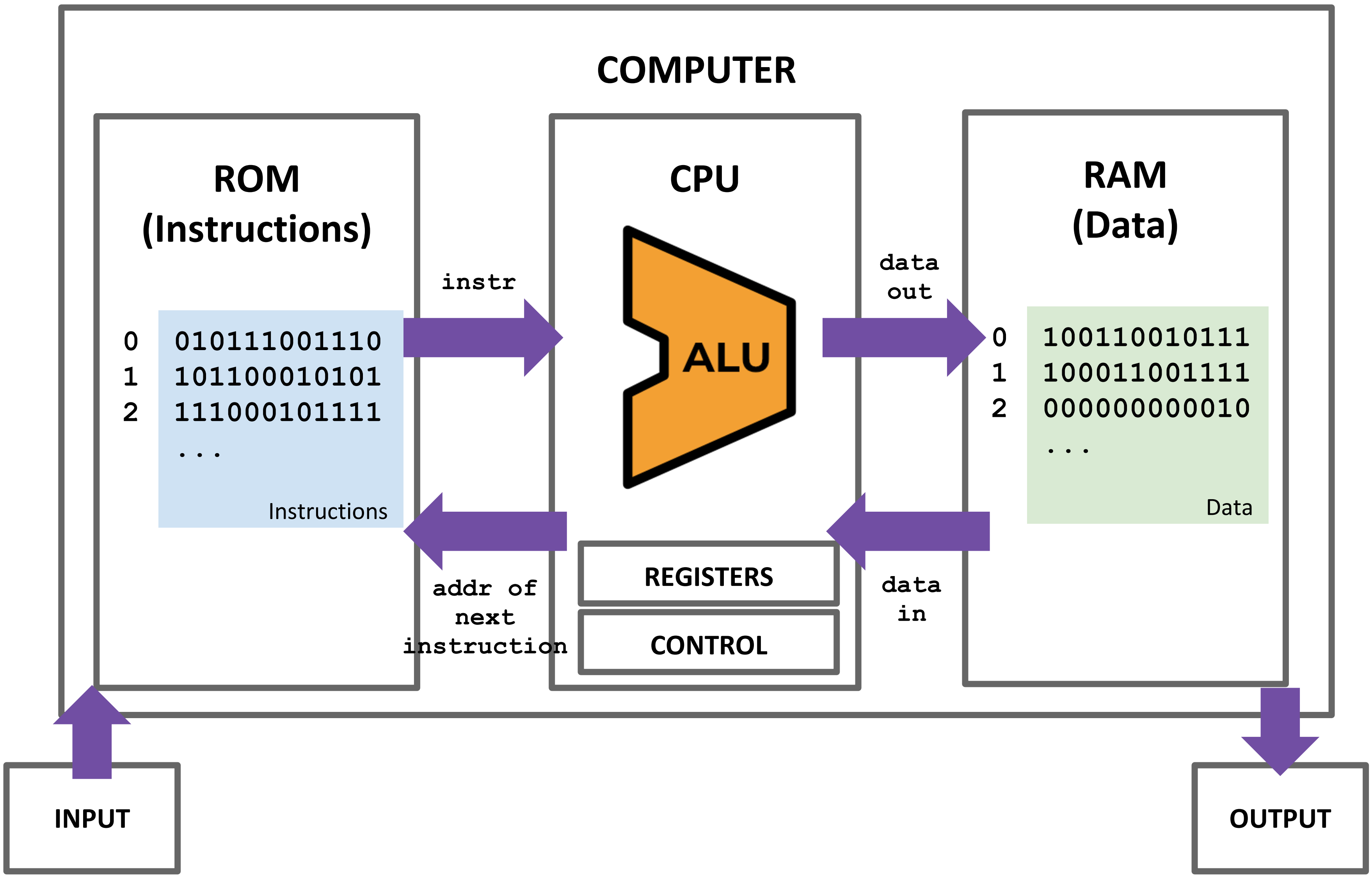
- Pros:
  - Simpler to implement

- Cons:
  - Fixed size of each partition, rather than flexible storage
  - Two chips → redundant circuitry

# Lecture Outline

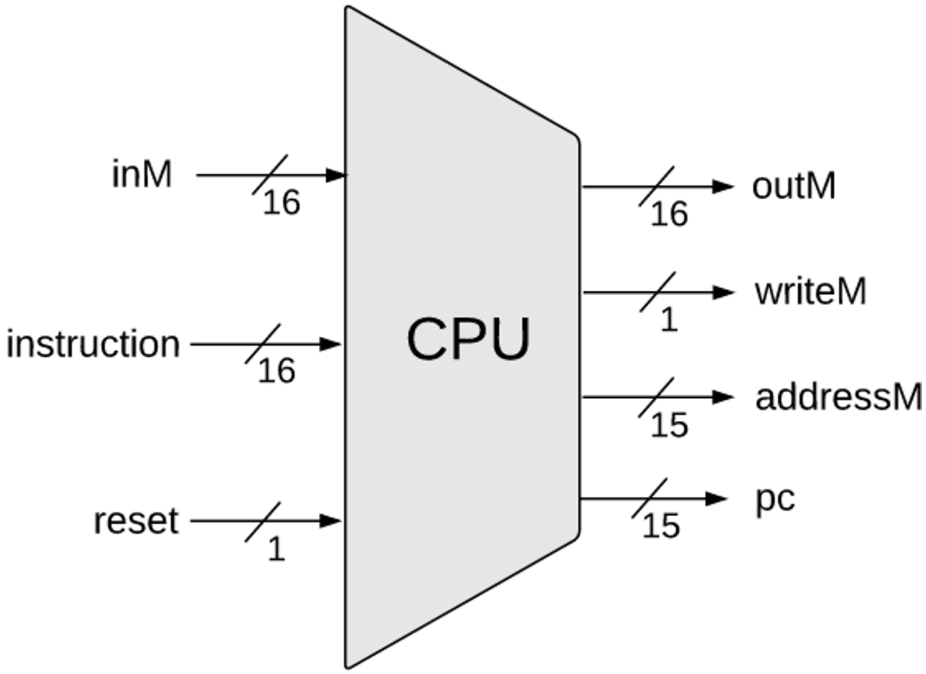
- Cornell Note-Taking Debrief
  - Reflection on notes from CSE 390B and other courses
- Exam Preparation
  - Study strategies, mock exam problem
- Building a Computer Overview
  - Architecture, fetch and execute cycle
- **Hack CPU Logic**
  - **Implementation and operations**

# 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

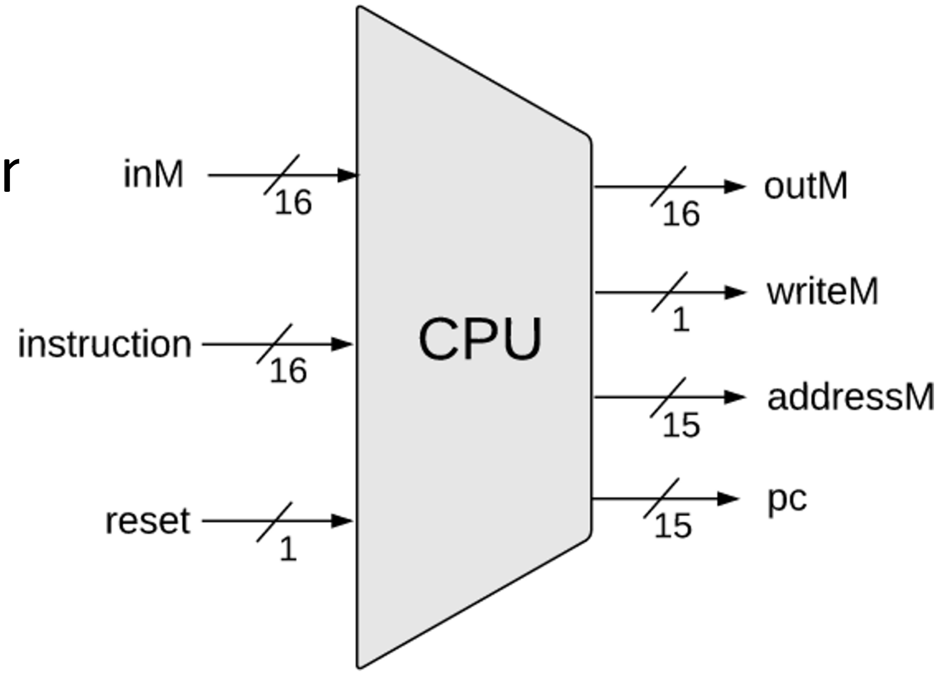
# Hack CPU Interface Inputs

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

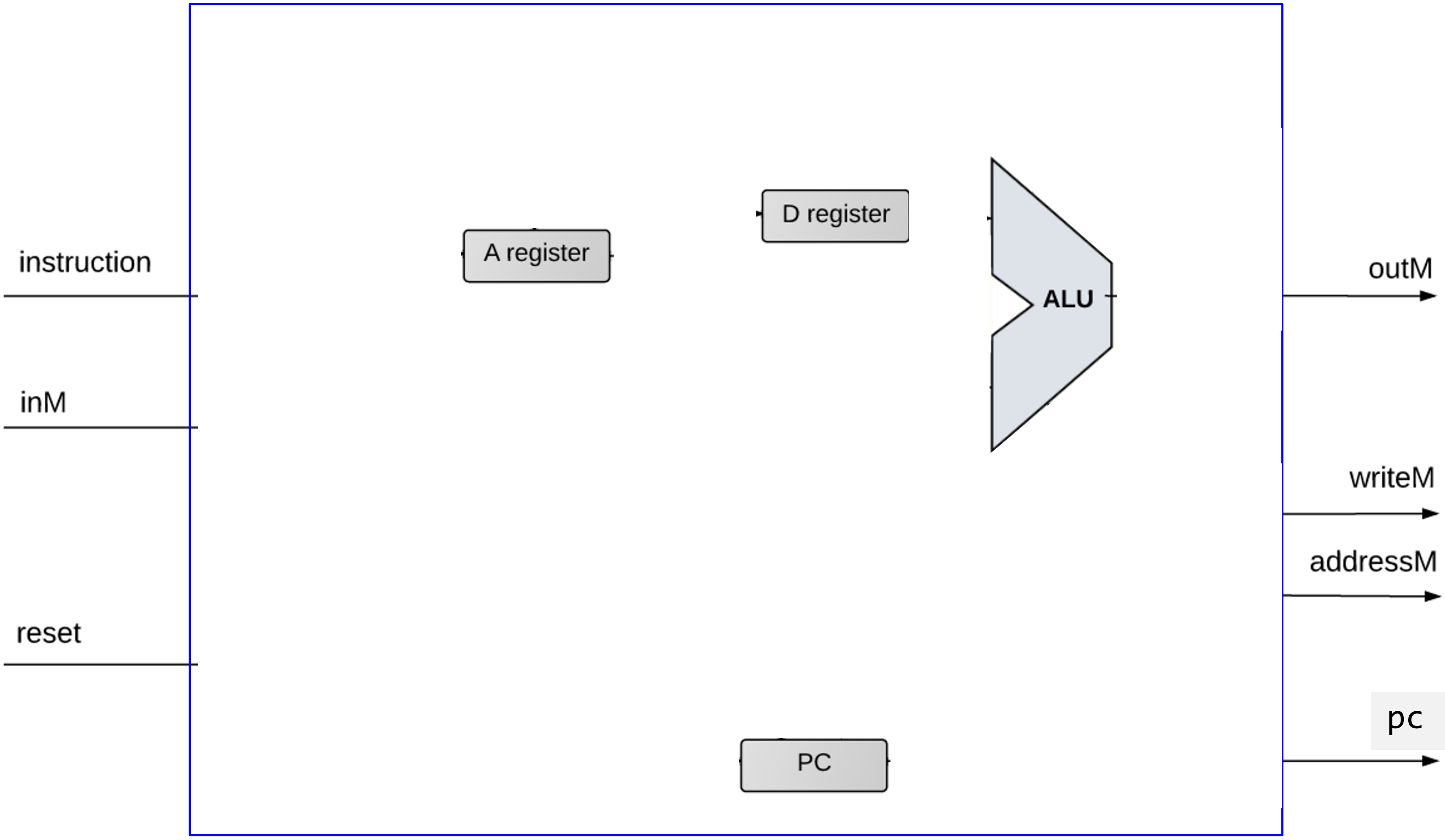


# Hack CPU Interface Outputs

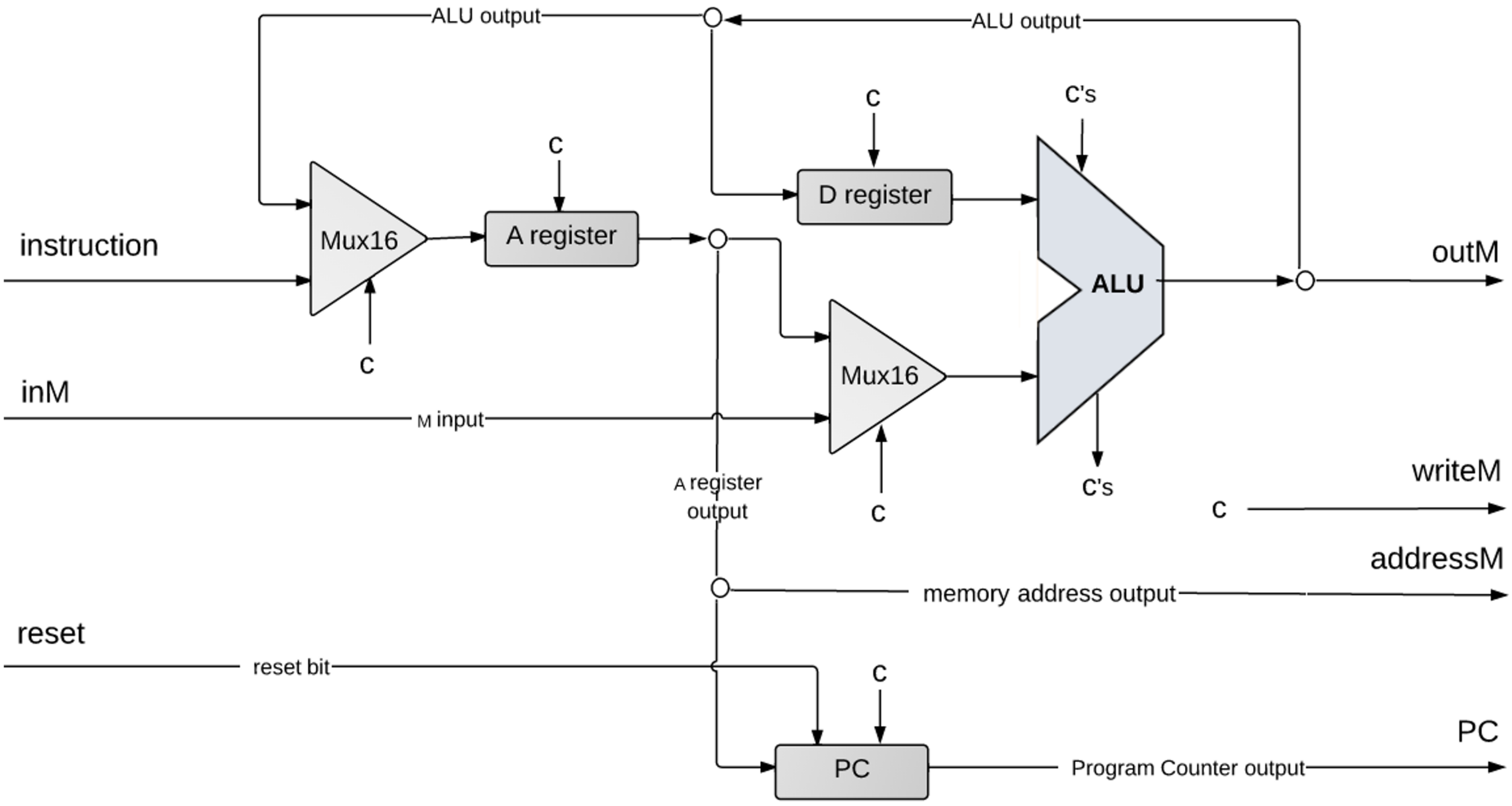
- **`outM`**: value used to update memory if writeM is 1
- **`writeM`**: if 1, update value ir memory at addressM with outM
- **`addressM`**: address to read from or write to in memory
- **`pc`**: address of next instruction to be fetched from memory



# Hack CPU Implementation

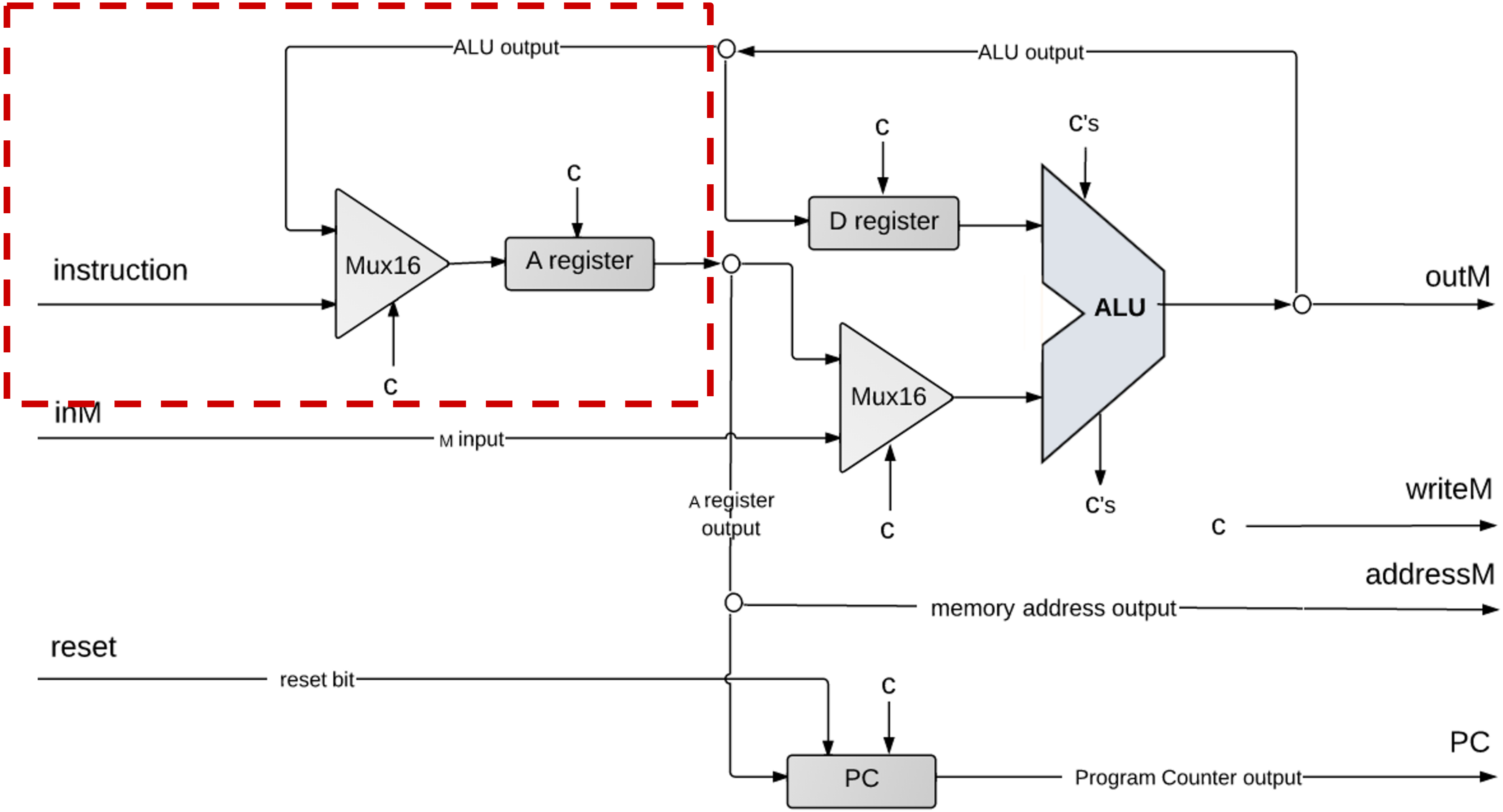


# Hack CPU Implementation



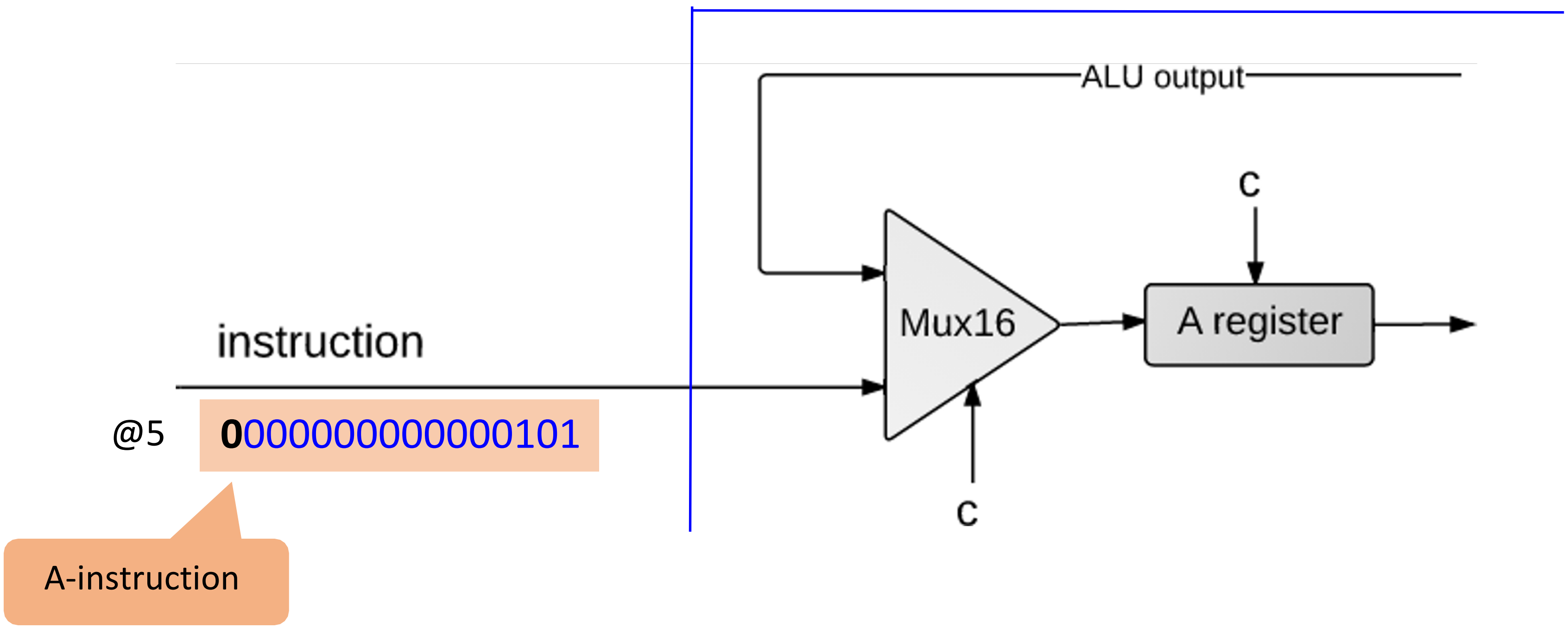

(each "c" symbol represents a control bit)

# CPU Operation: Instruction Handling

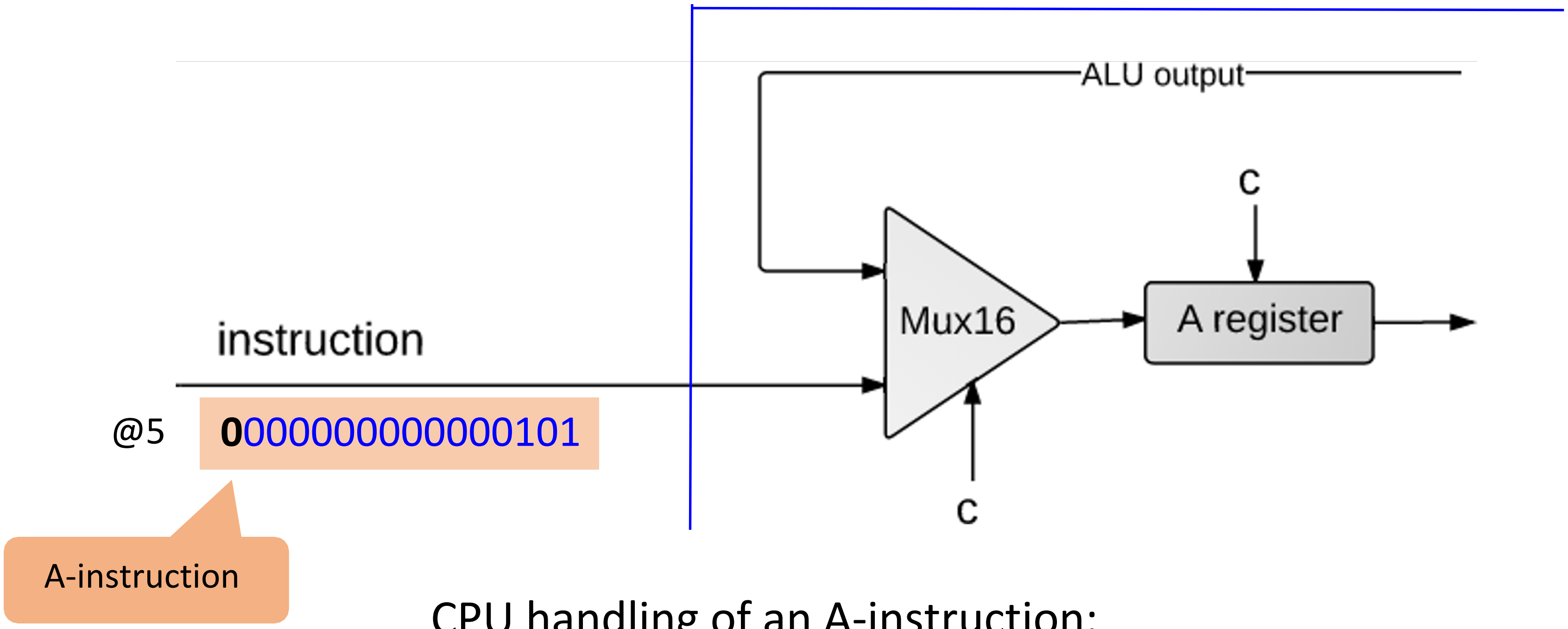



(each "C" symbol represents a control bit)

# CPU Operation: Instruction Handling



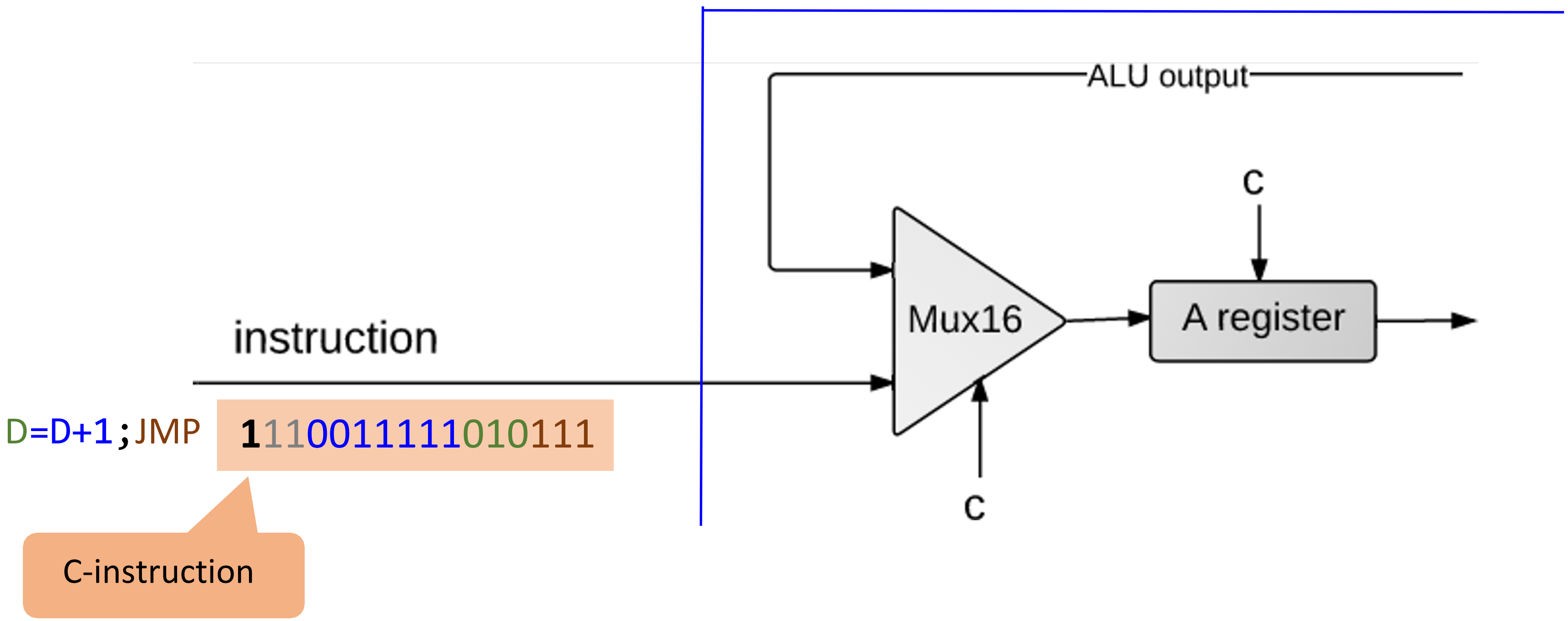
# CPU Operation: Instruction Handling



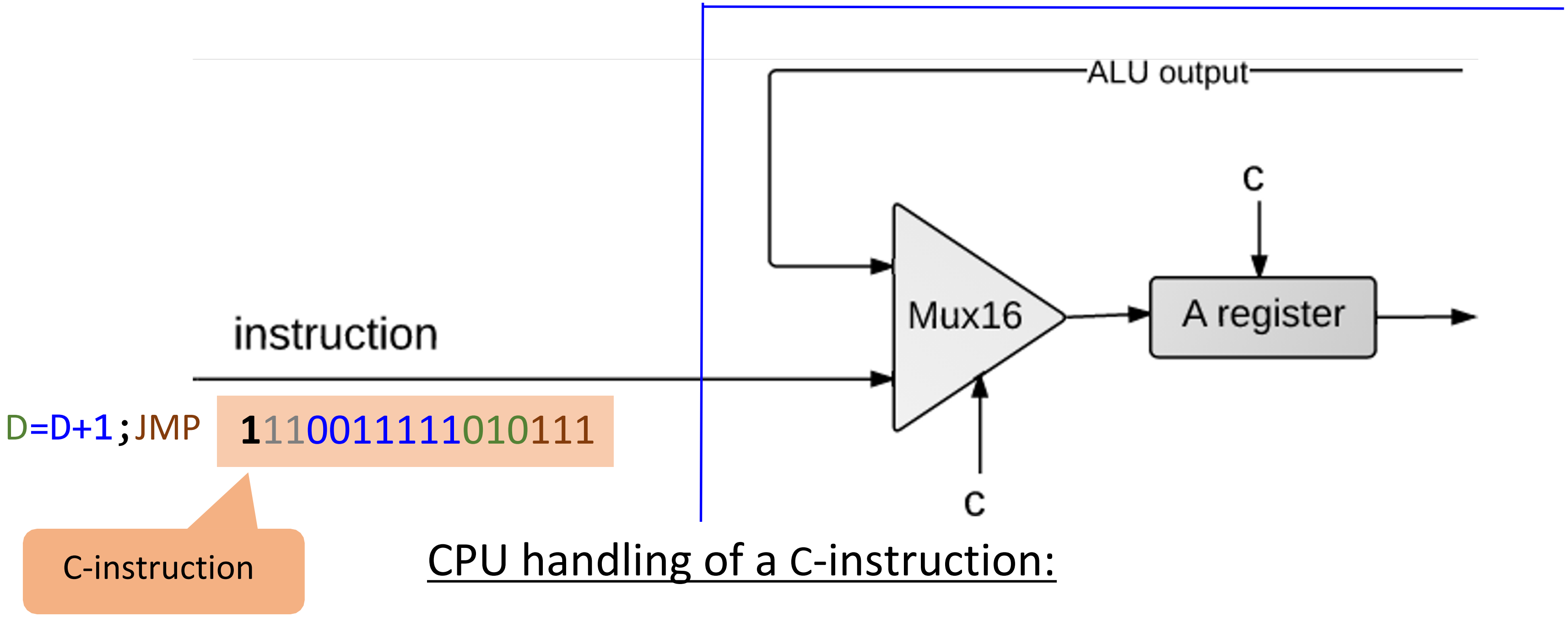

CPU handling of an A-instruction:

- 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



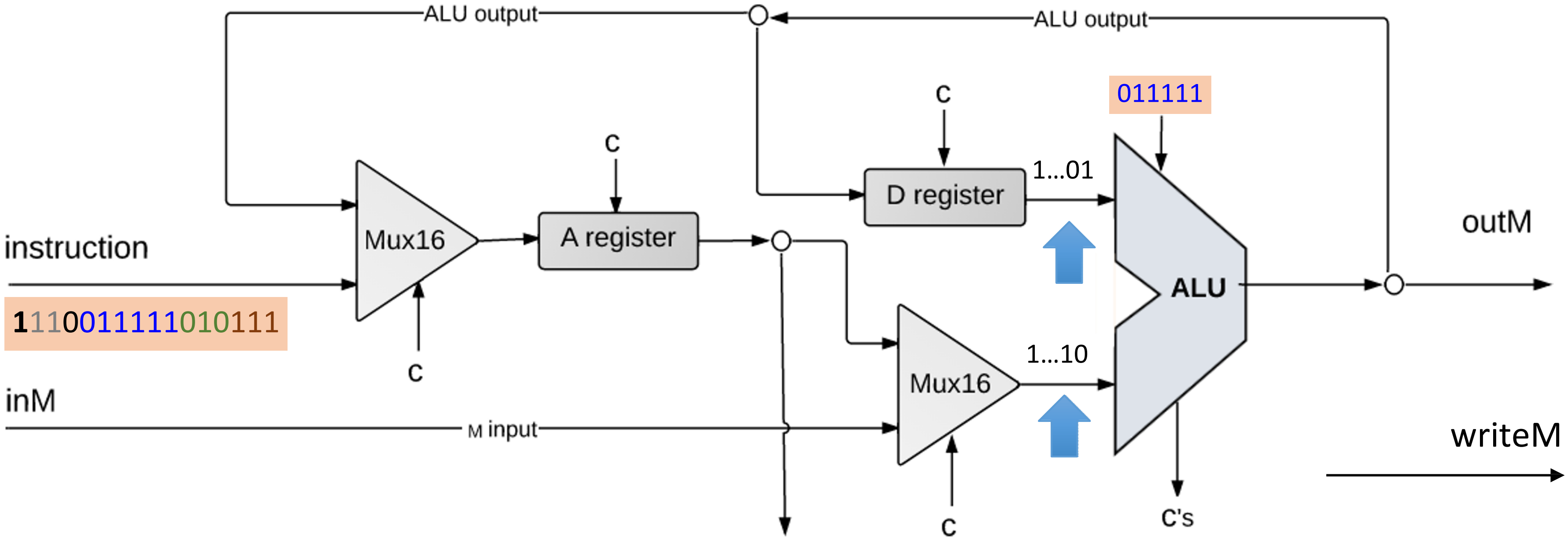
# CPU Operation: Instruction Handling




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



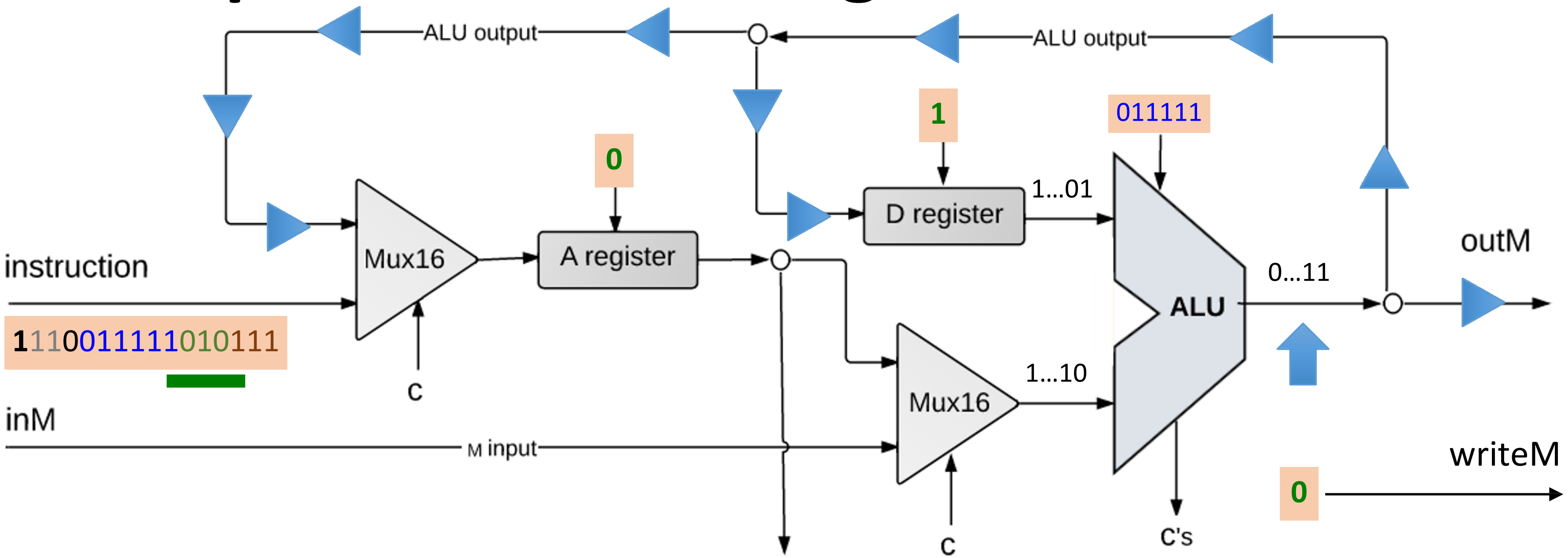

ALU data inputs:

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

ALU control inputs:

- control bits (from the instruction)

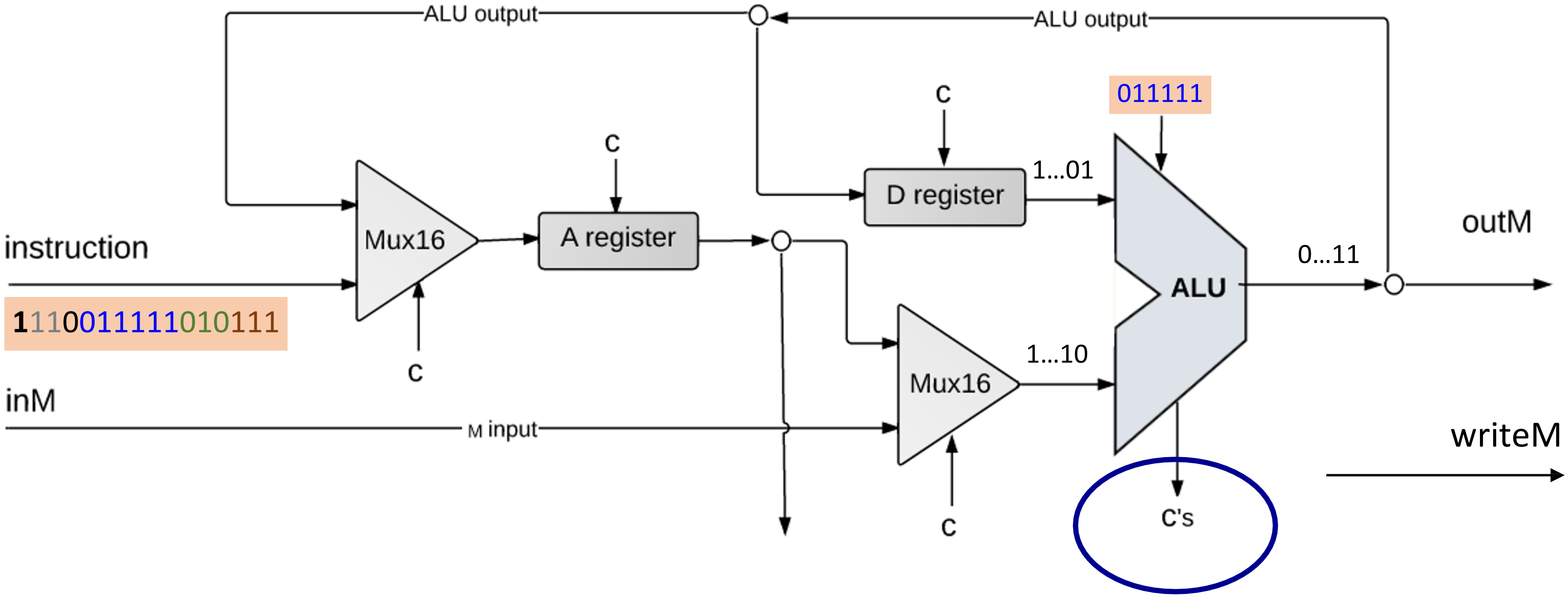
# CPU Operation: Handling C-Instructions




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.

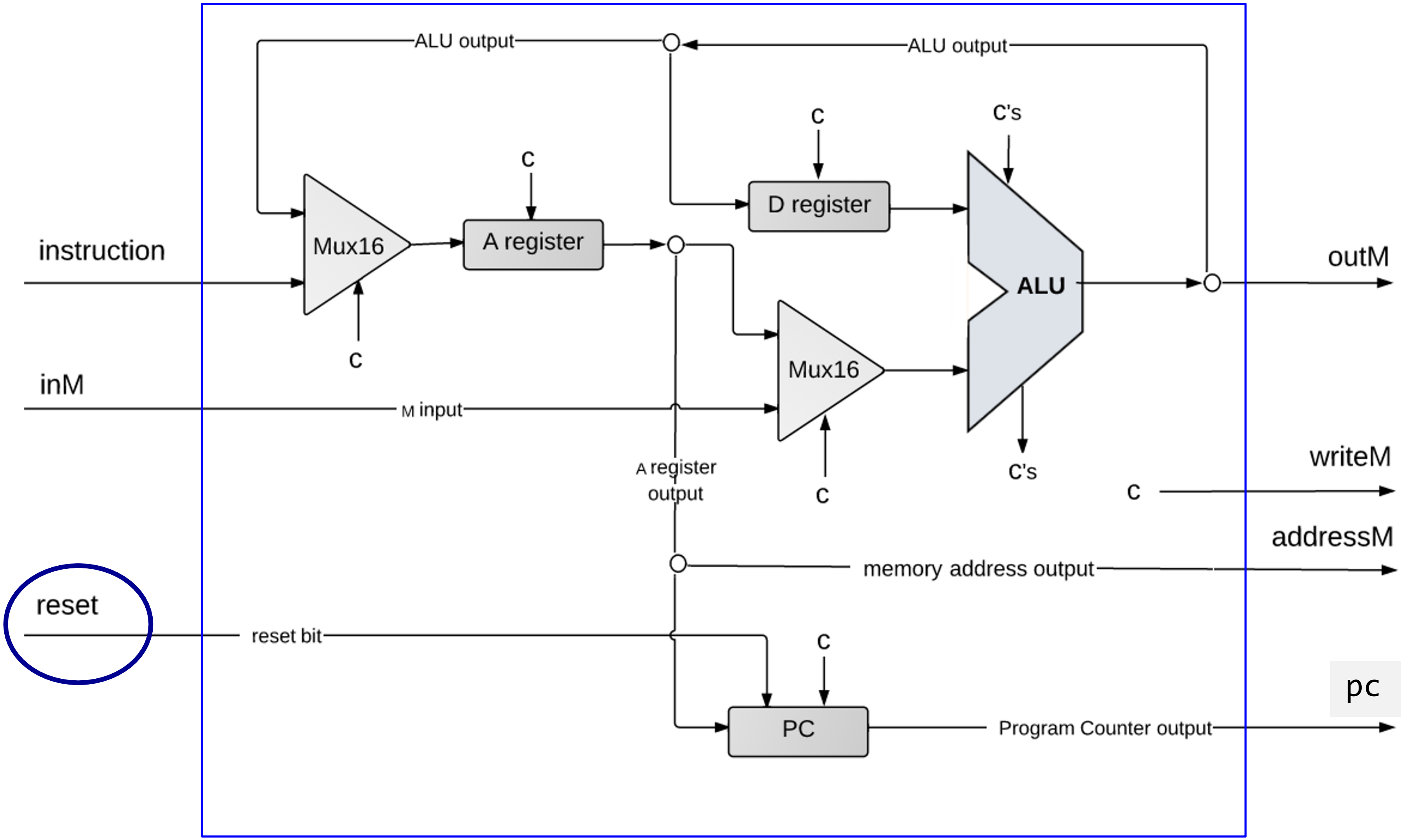
# CPU Operation: Handling C-Instructions



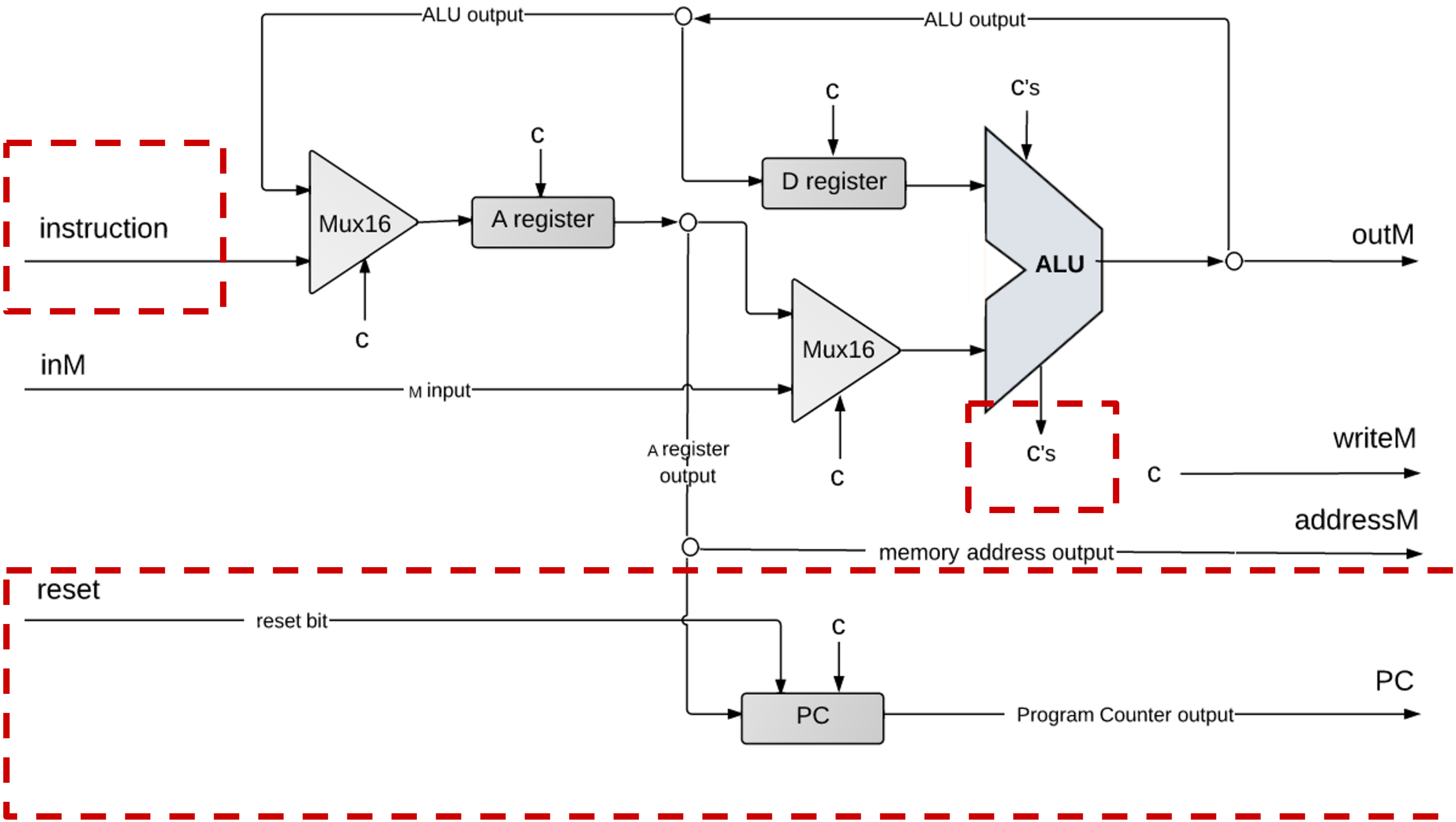

ALU control outputs:

- is the output negative?
- is the output zero?

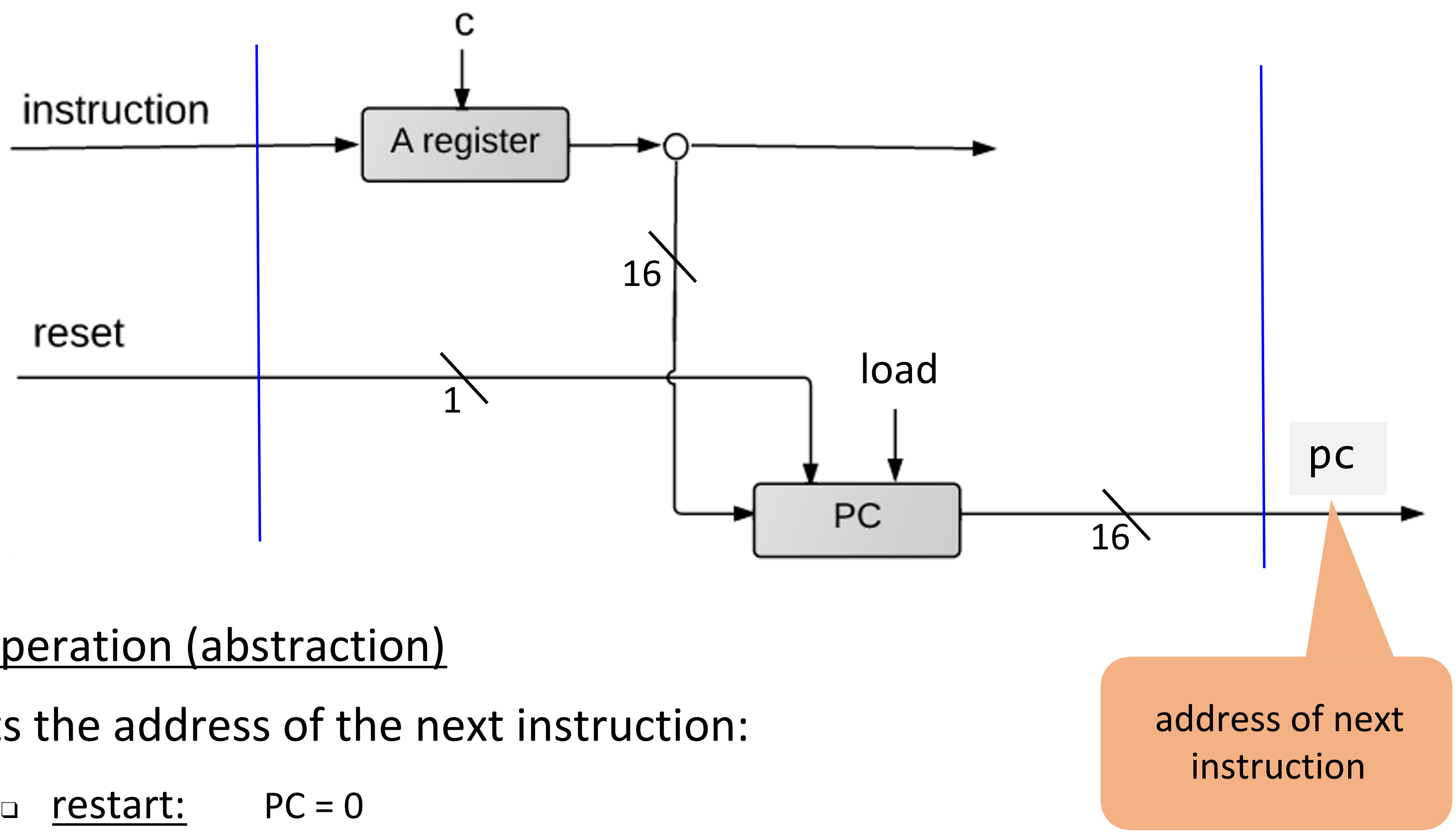
# CPU Operation: Control



# CPU Operation: Control



# CPU Operation: Control

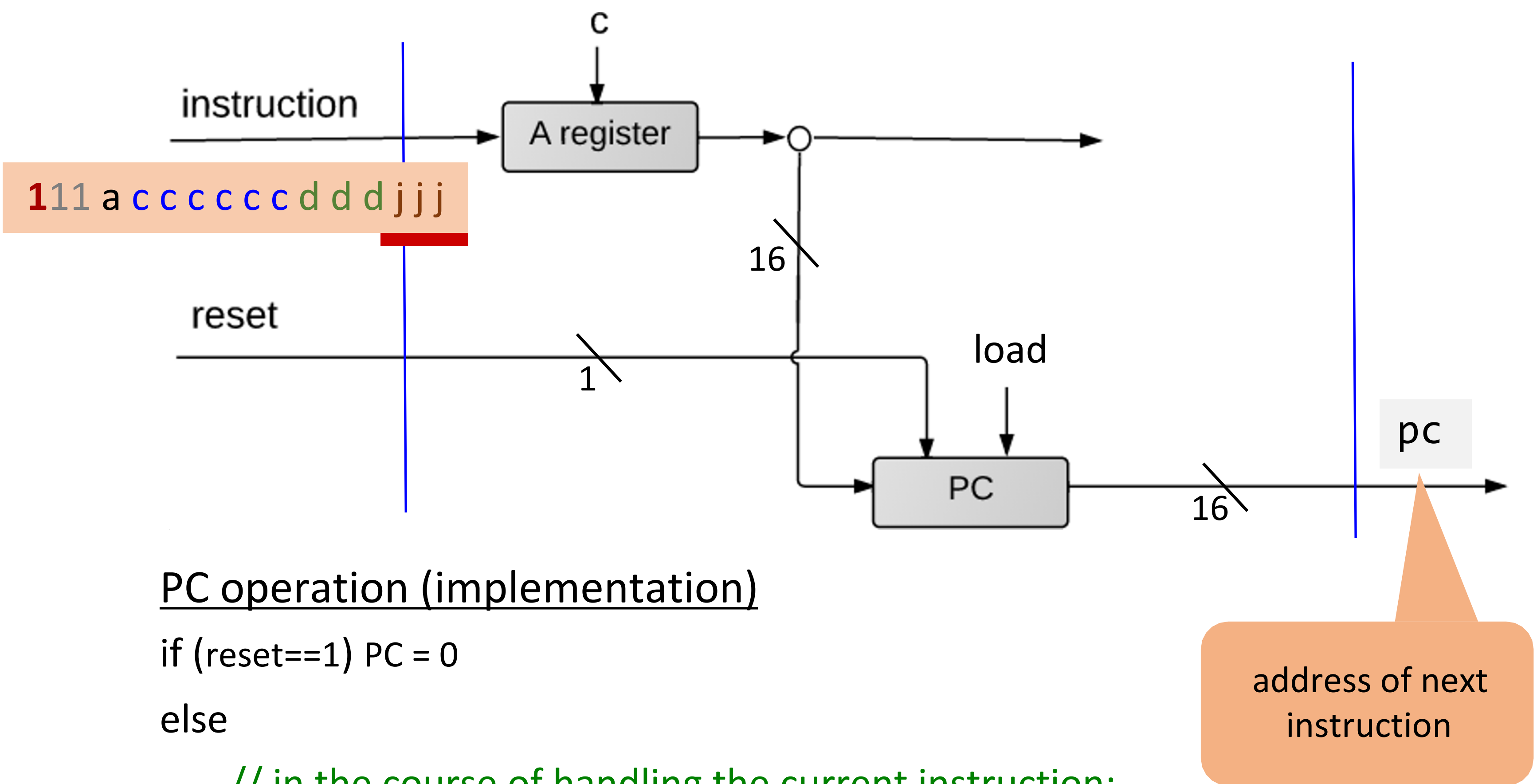



PC operation (abstraction)

Emits the address of the next instruction:

- restart: PC = 0
- no jump: PC++
- goto: PC=A
- conditional goto: if (*condition*) PC = A else PC++

# CPU Operation: Control




## PC operation (implementation)

if (reset==1) PC = 0

else

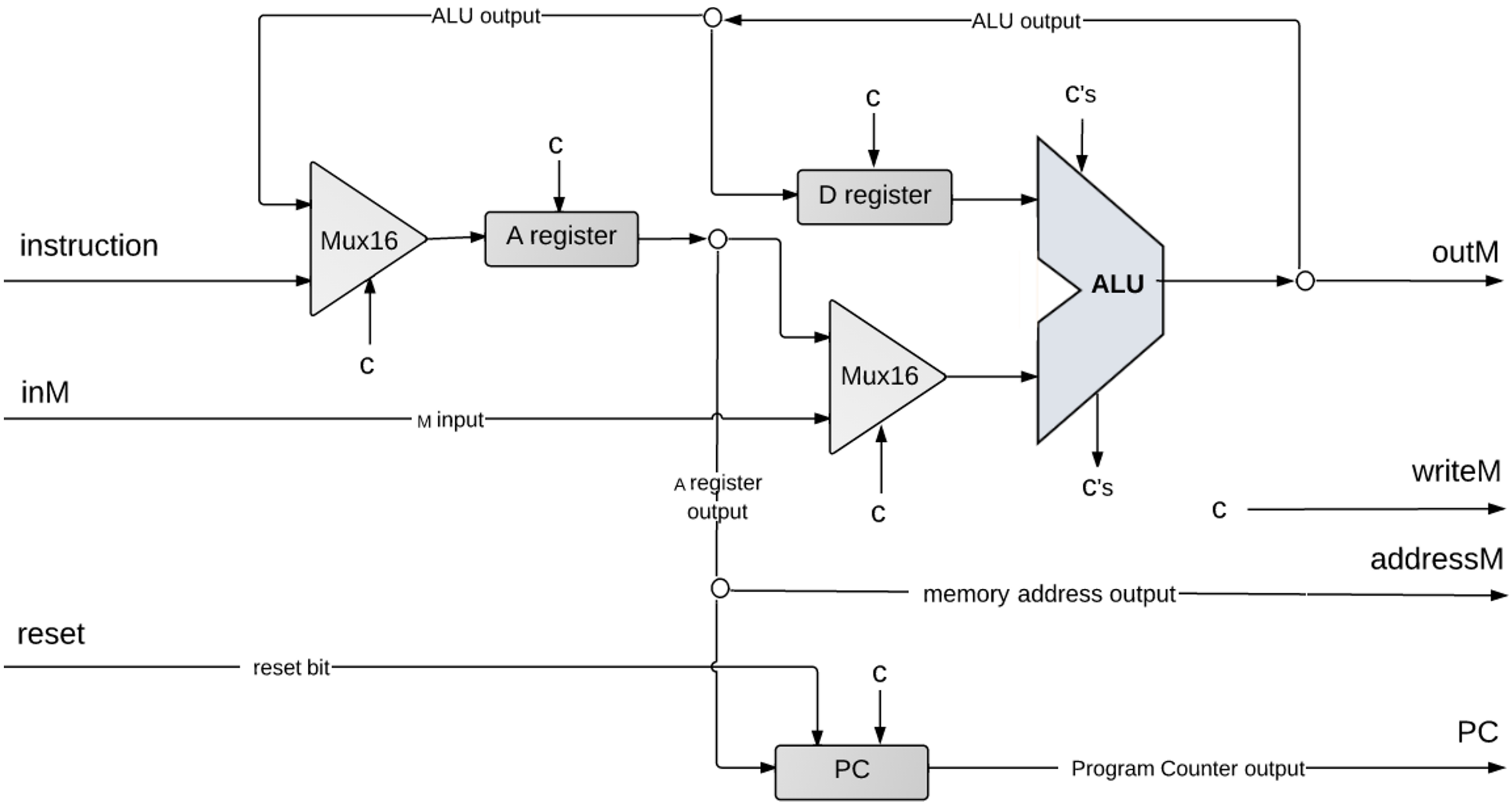
// in the course of handling the current instruction:

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

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

els
e PC++ // next instruction

# Hack CPU Implementation: That's It!



# Post-Lecture 9 Reminders

- Project 4: Machine Language, Building a Computer Part I, & Annotation
  - Due on Thursday (2/3) at 11:59PM PST

- CSE 390B Midterm
  - Thursday, February 10th during lecture