

Chapter 5

Computer Architecture

These slides support chapter 5 of the book

The Elements of Computing Systems

By Noam Nisan and Shimon Schocken

MIT Press

→ Von Neumann Architecture

- Fetch-Execute Cycle
- The Hack CPU
- The Hack Computer
- Project 5 Overview

Universality

Same **hardware** can run many different **software** programs

Theory



Alan Turing:

Universal Turing Machine

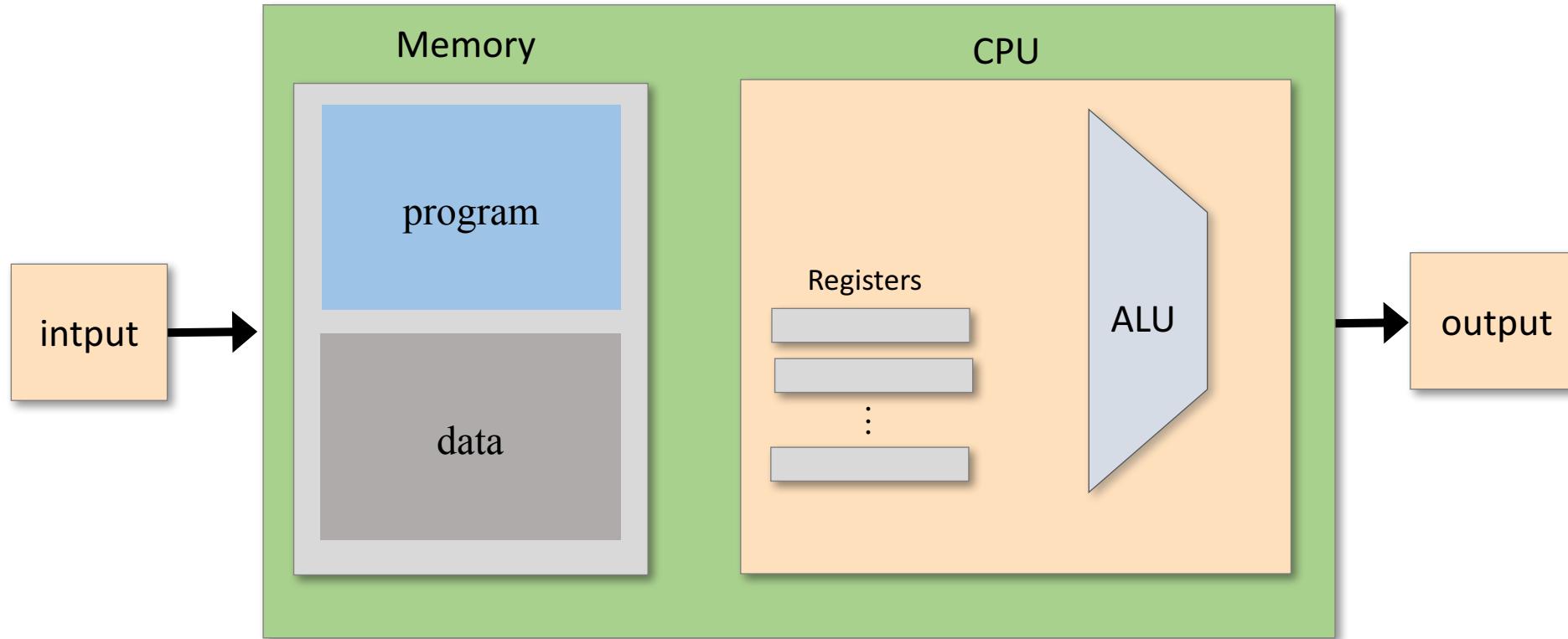
Practice



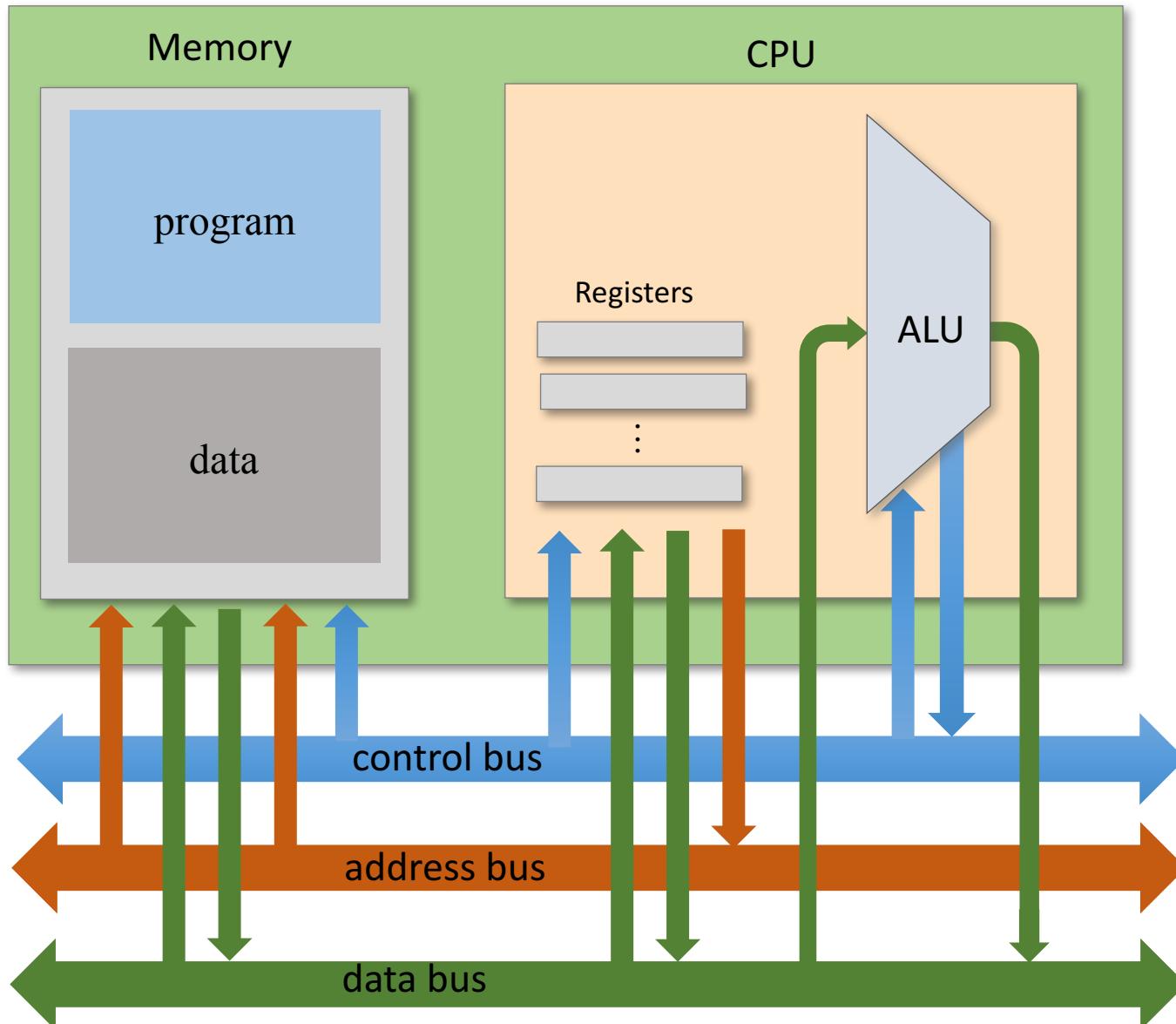
John Von Neumann:

Stored Program Computer

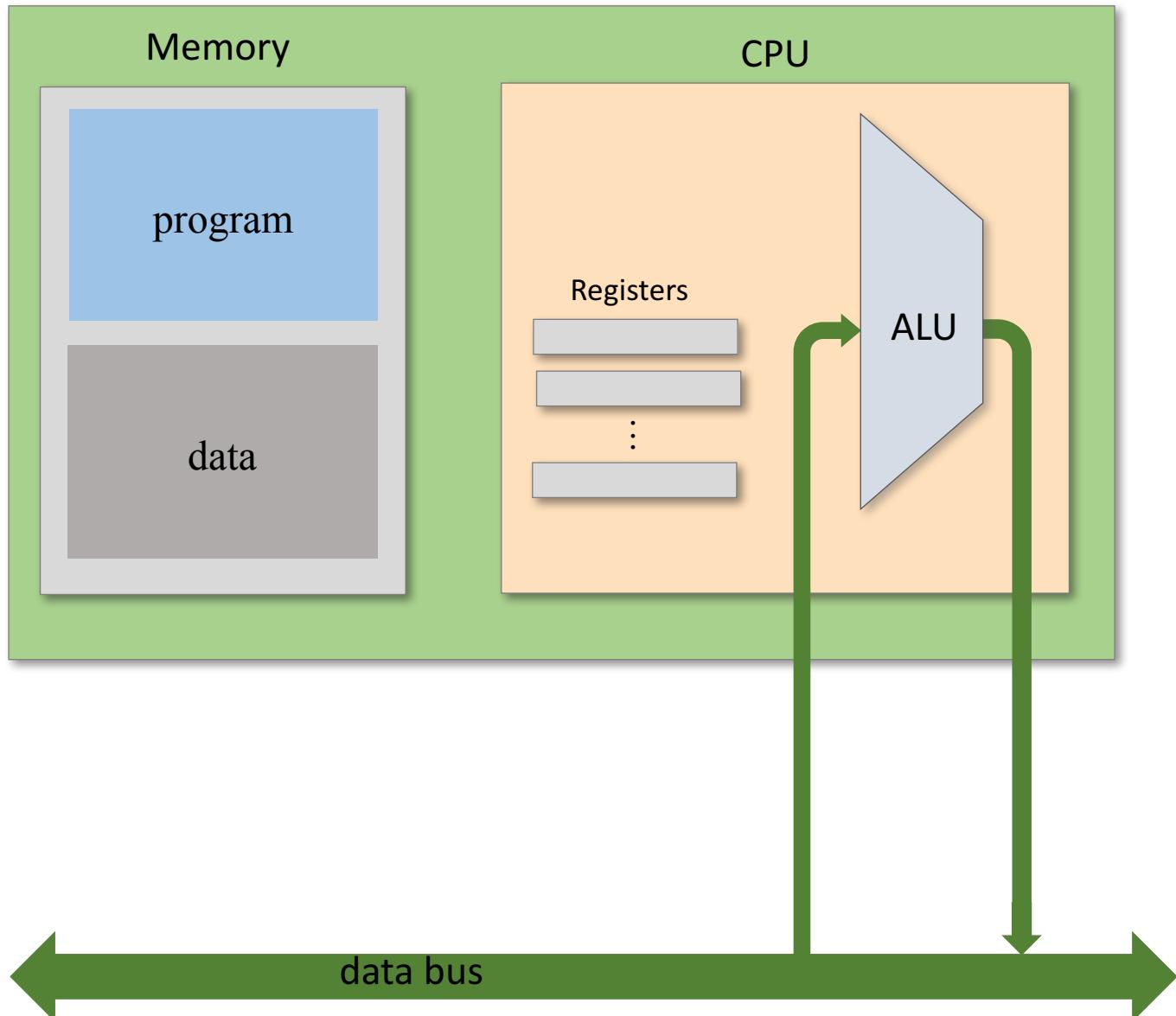
Computer architecture



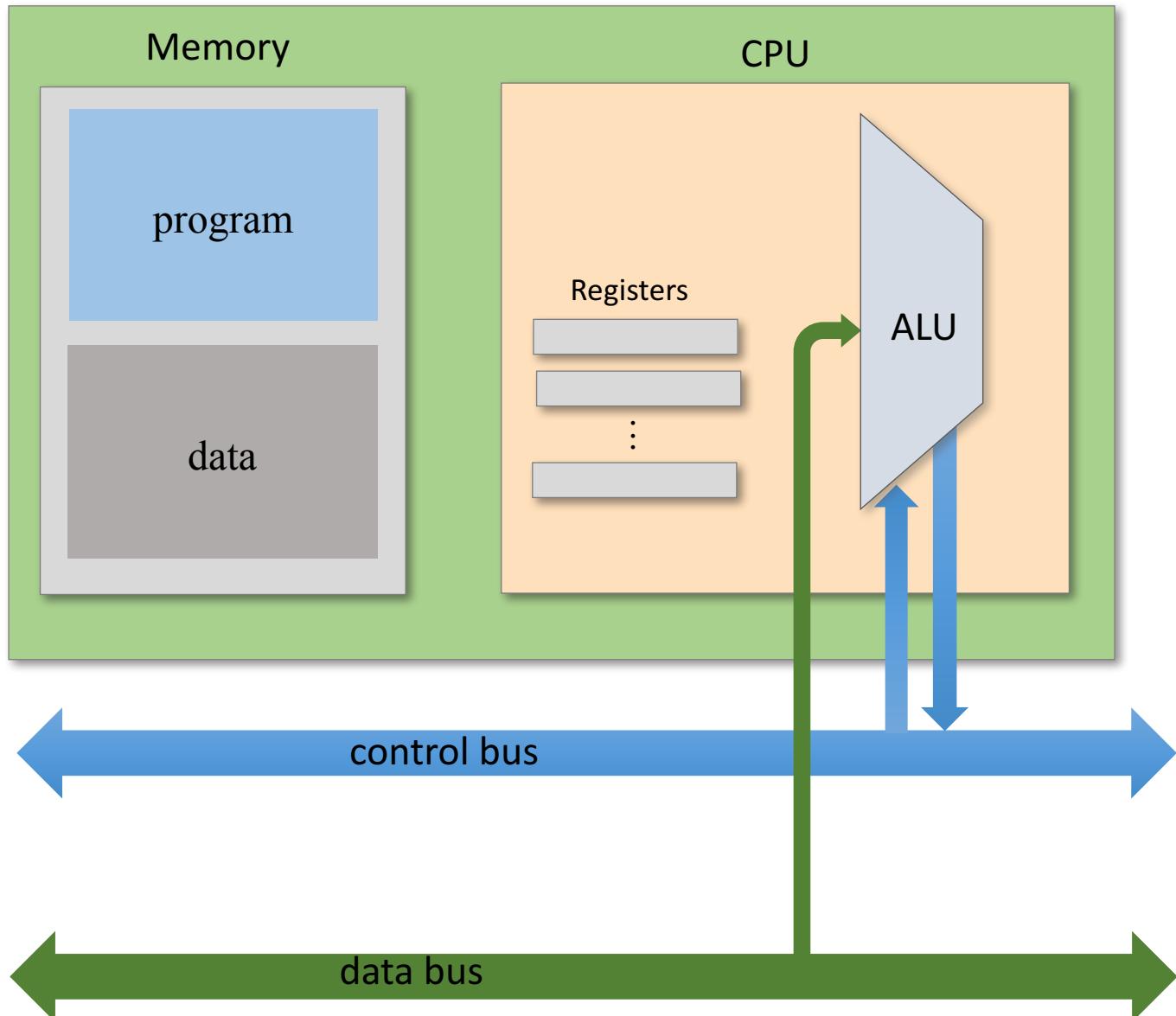
Computer architecture



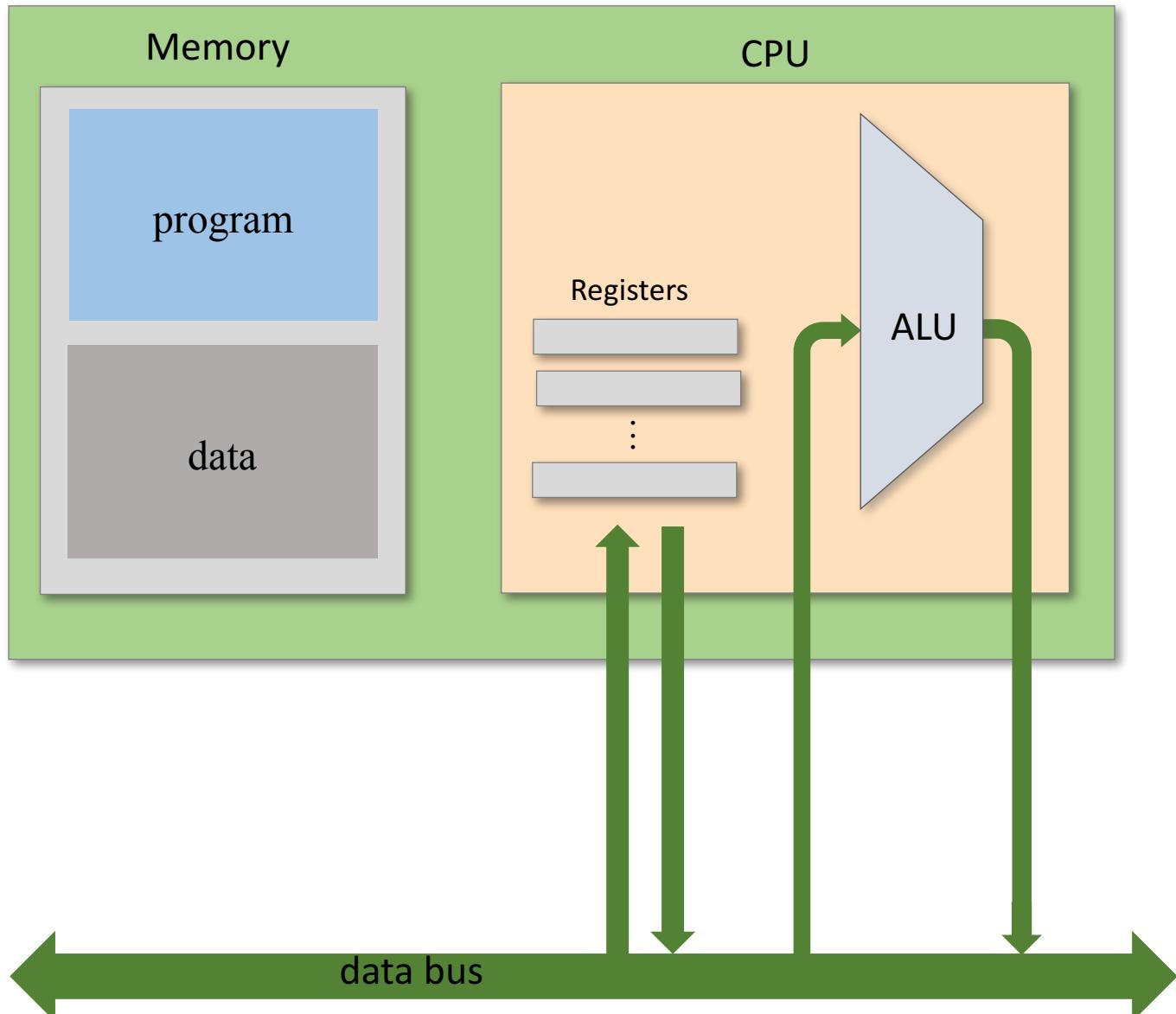
Computer architecture



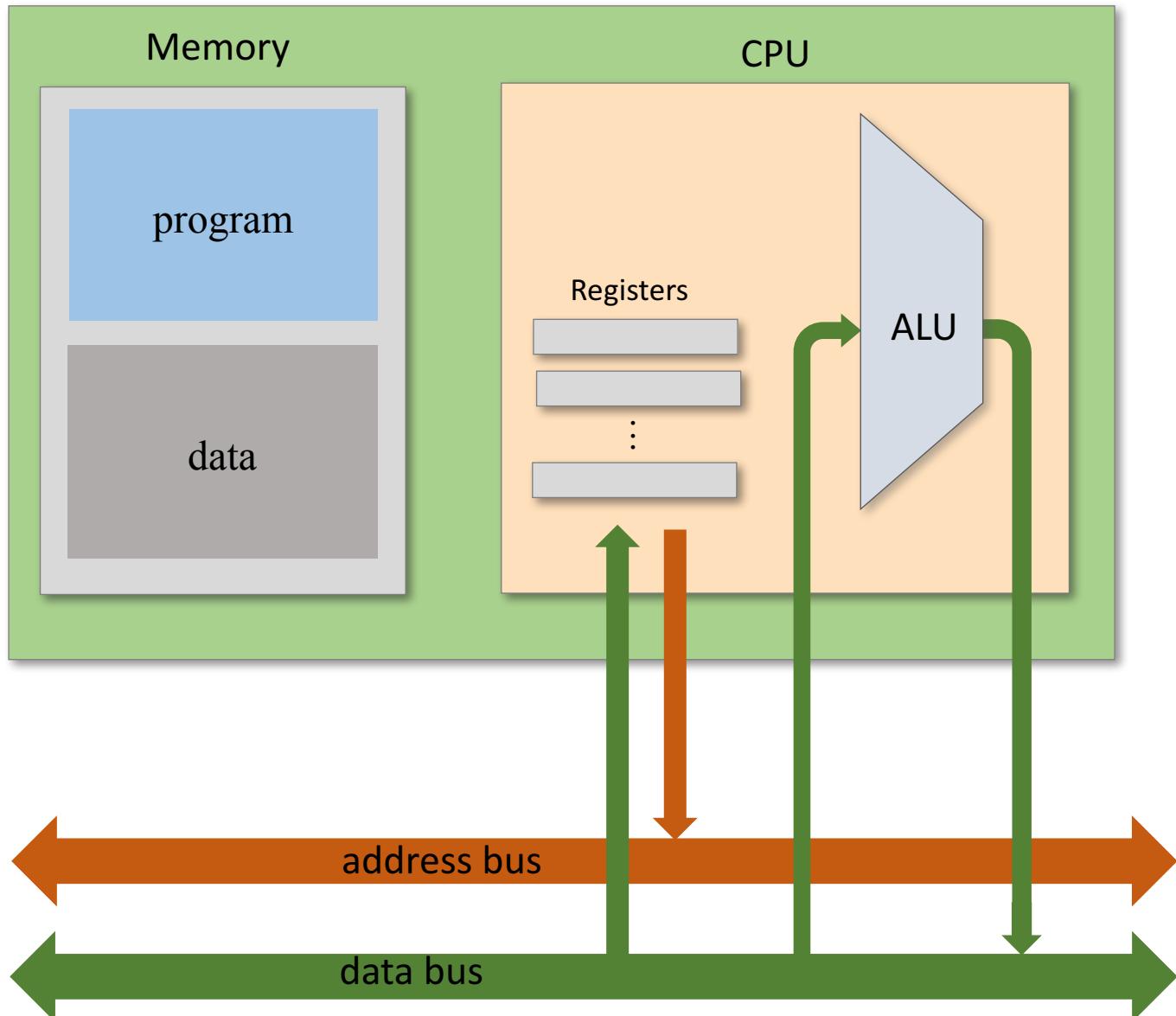
Computer architecture



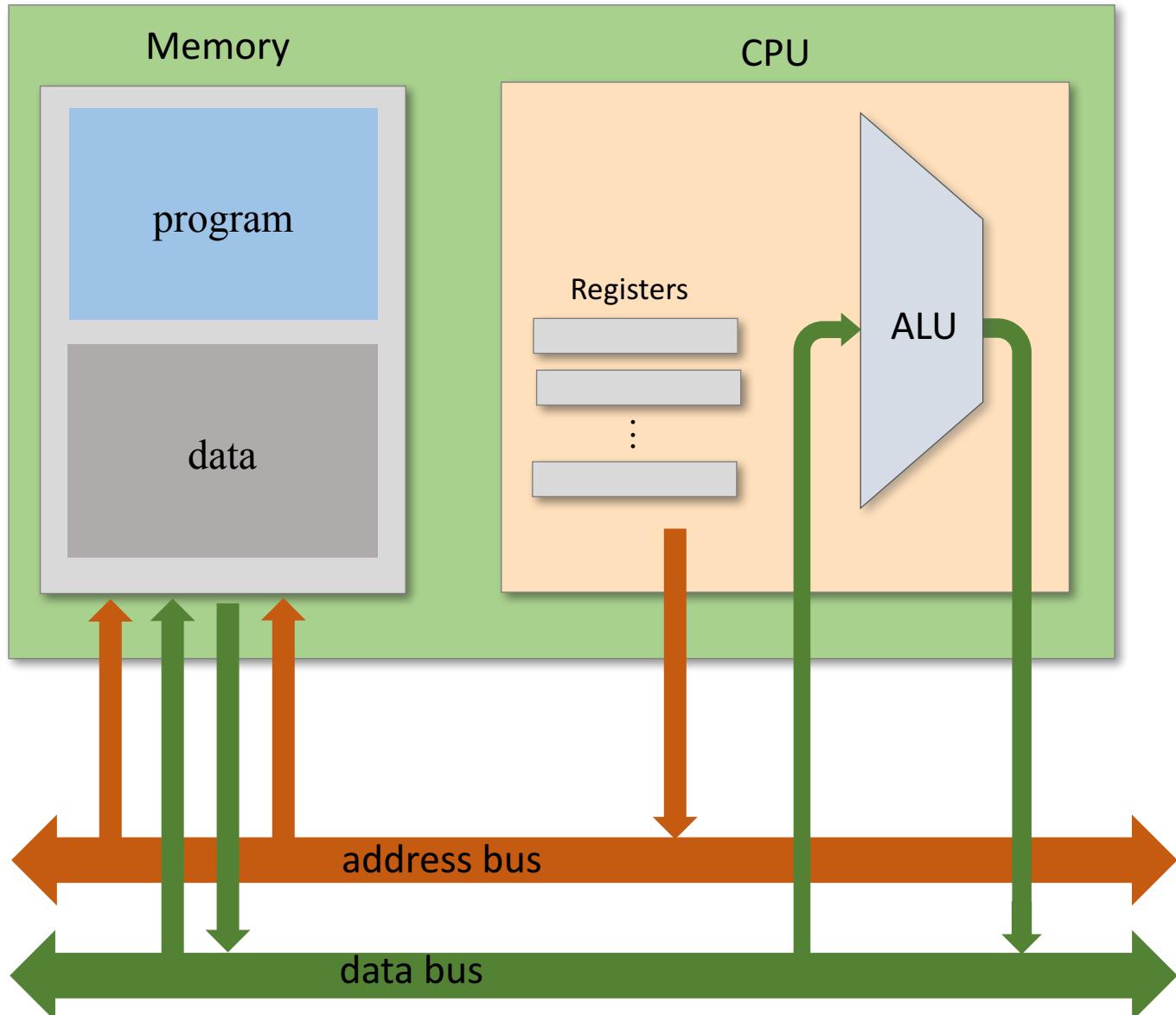
Computer architecture



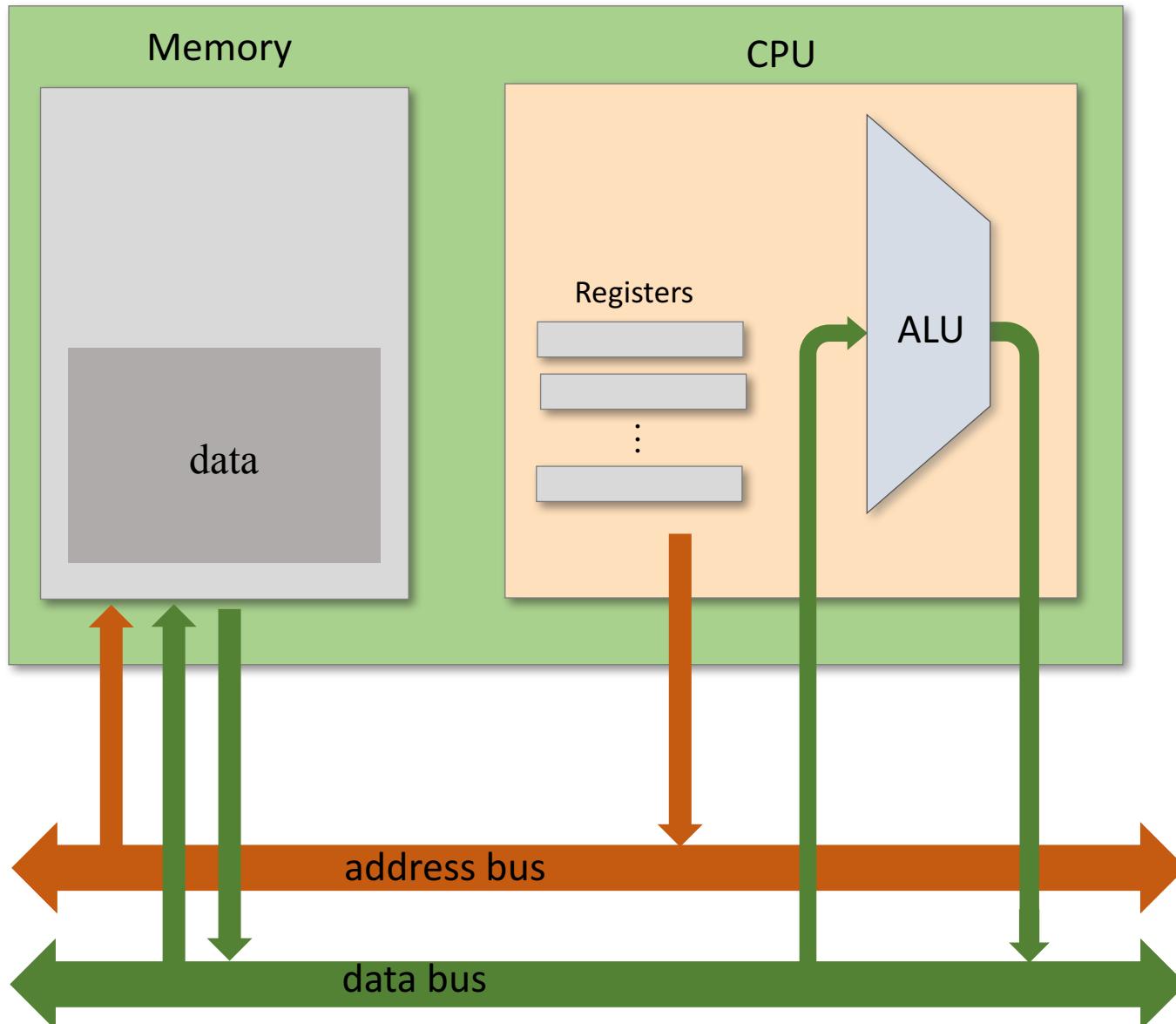
Computer architecture



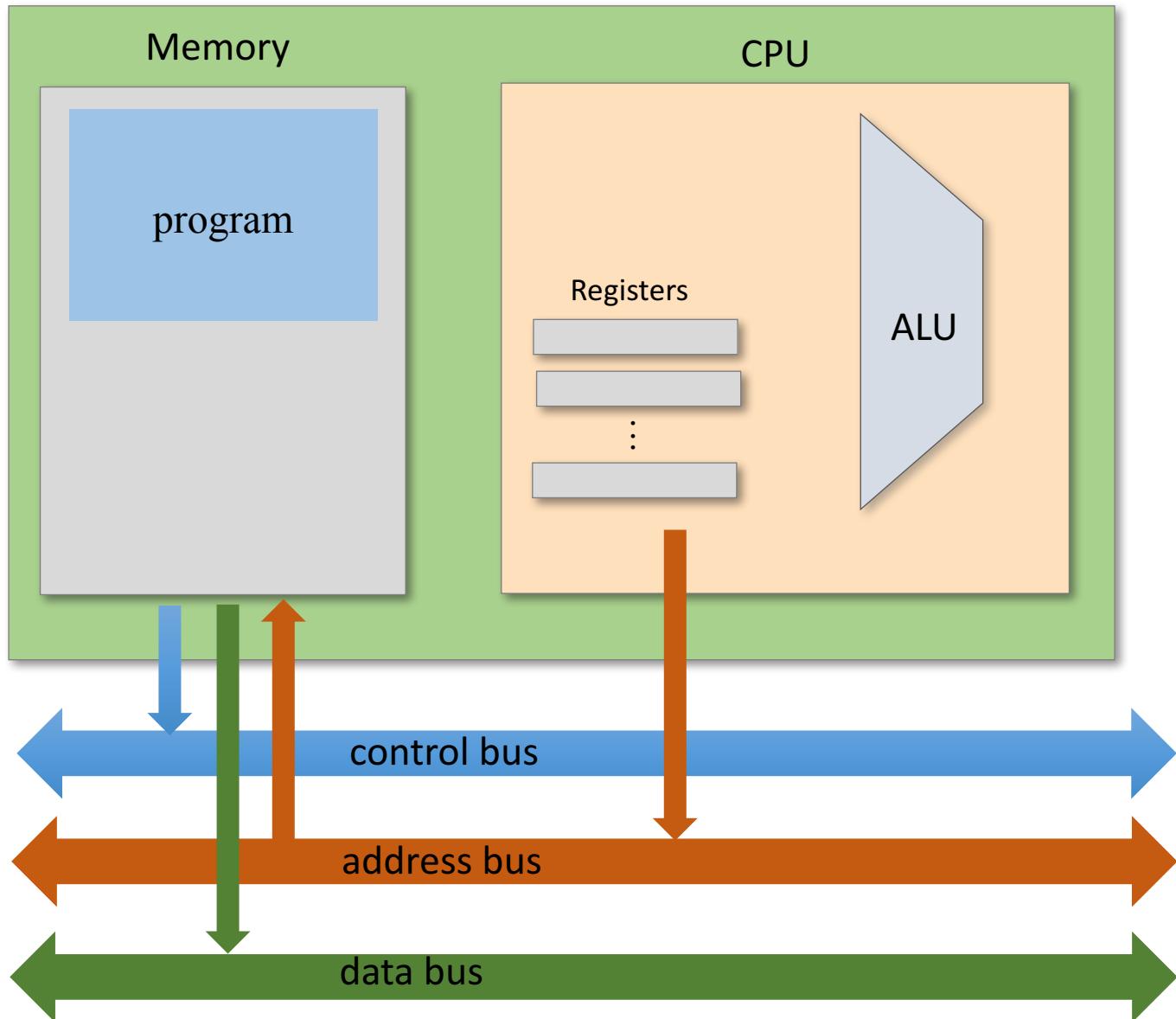
Computer architecture



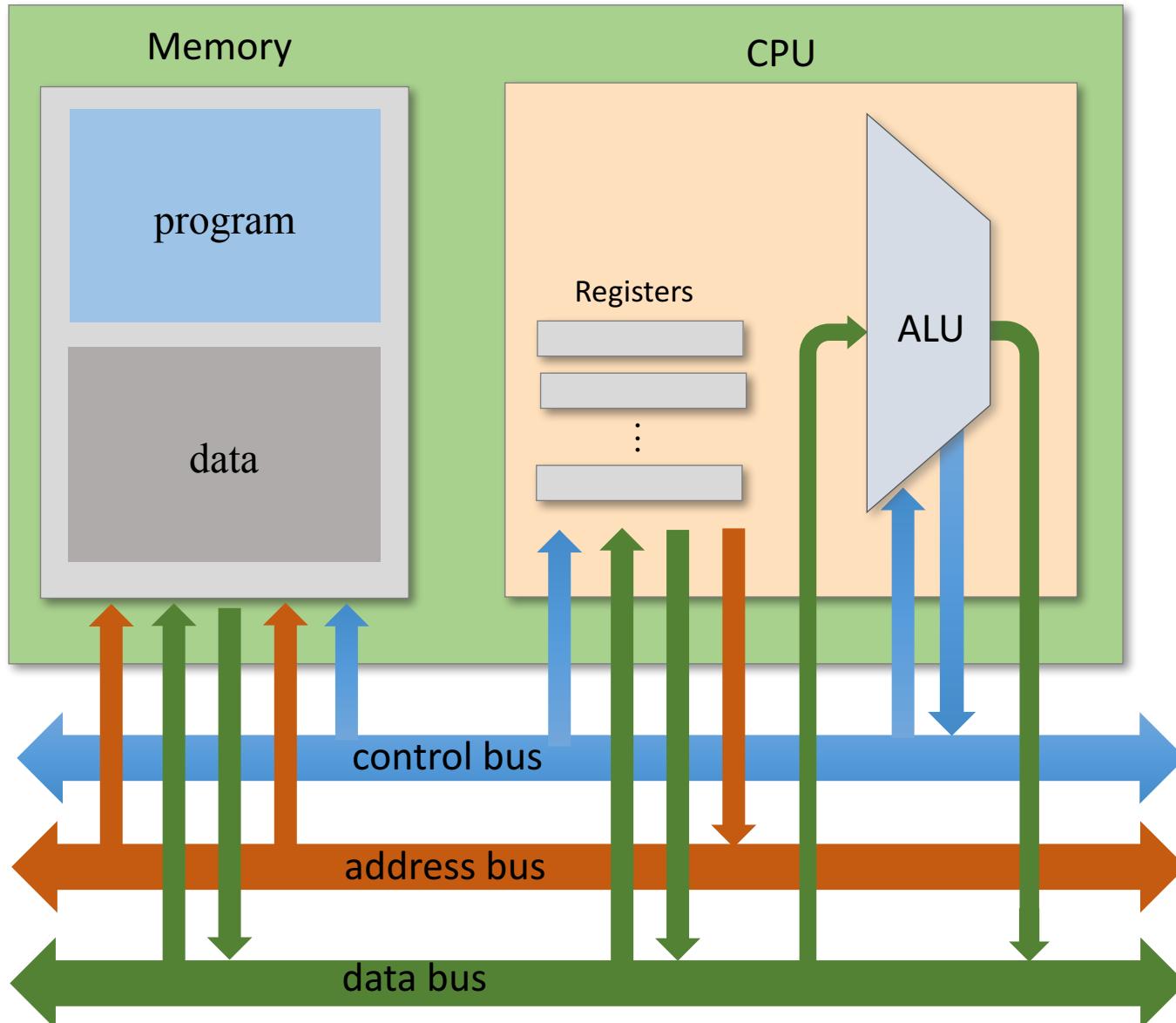
Computer architecture



Computer architecture



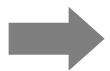
Computer architecture



Computer Architecture: lecture plan



Von Neumann Architecture



Fetch-Execute Cycle

- The Hack CPU
- The Hack Computer
- Project 5 Overview

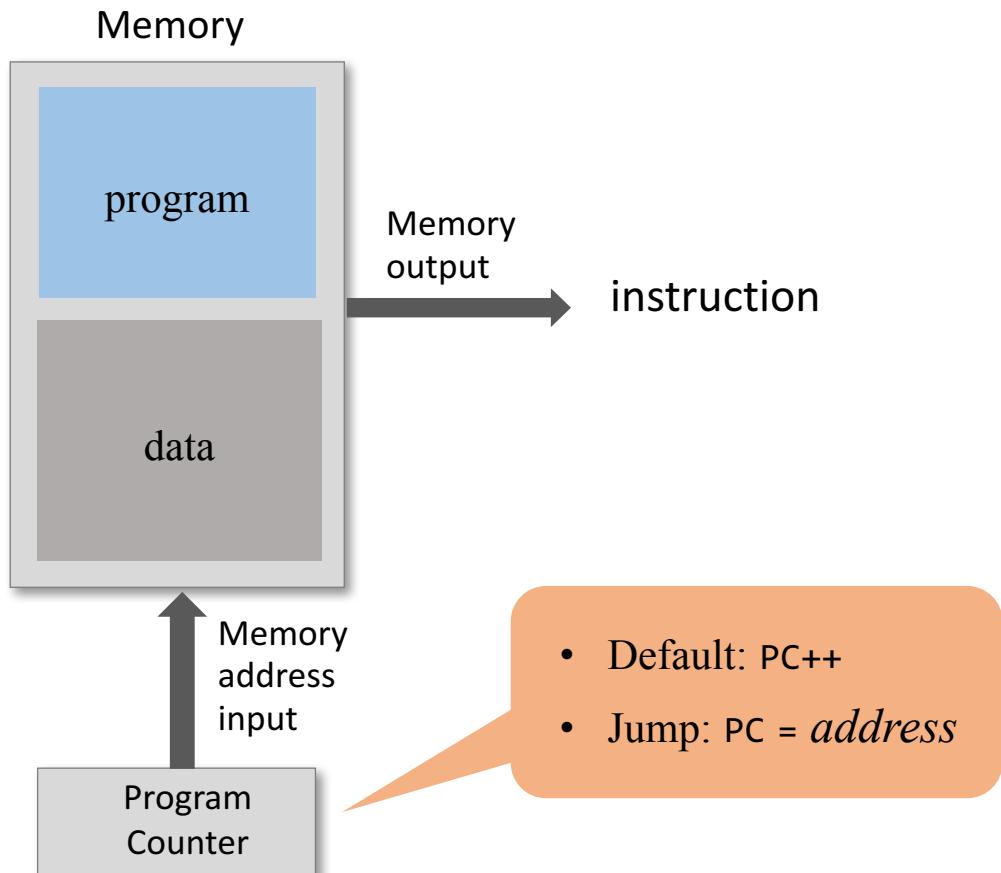
Basic CPU loop

Repeat:

- *Fetch* an instruction from the program memory
- *Execute* the instruction.

Fetching

- Put the location of the next instruction in the Memory *address* input
- Get the instruction code by reading the contents at that Memory location

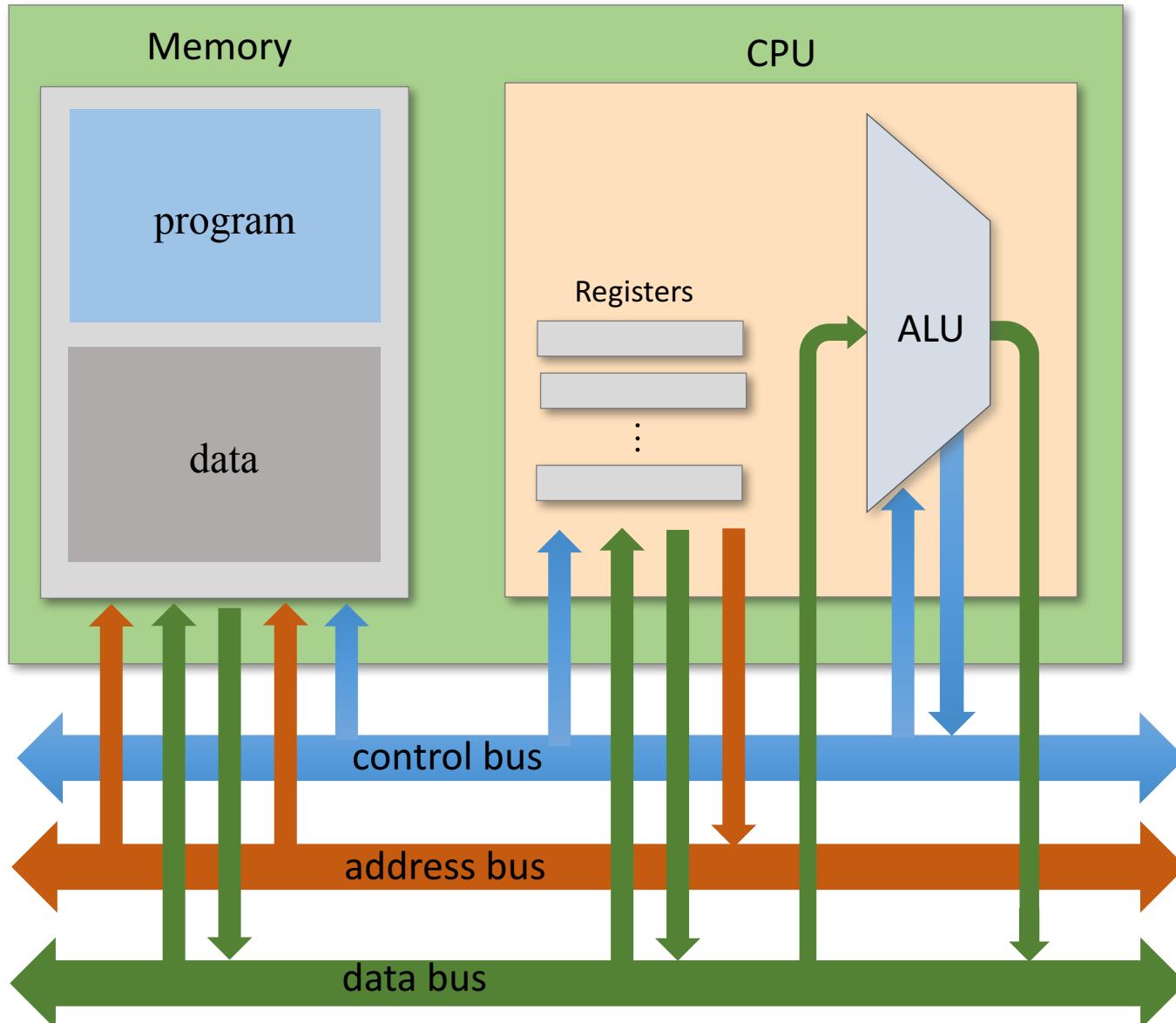


Executing

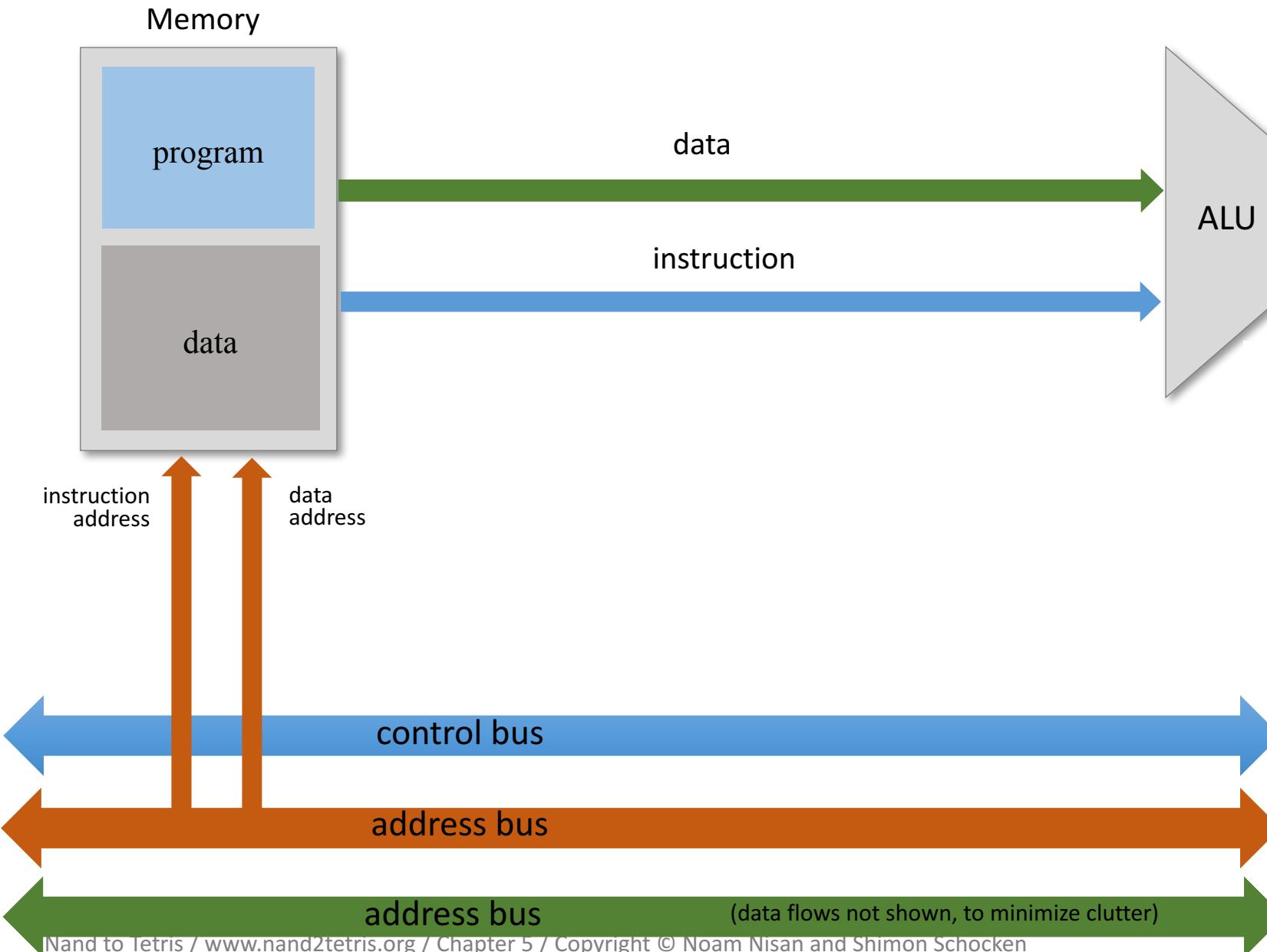
- The instruction code specifies “what to do”
 - Which arithmetic or logical instruction to execute
 - Which memory address to access (for read / write)
 - If / where to jump
 - ...
- Executing the instruction involves:
 - accessing registers
and / or:
 - accessing the data memory.

different subsets of the instruction bits control different aspects of the operation

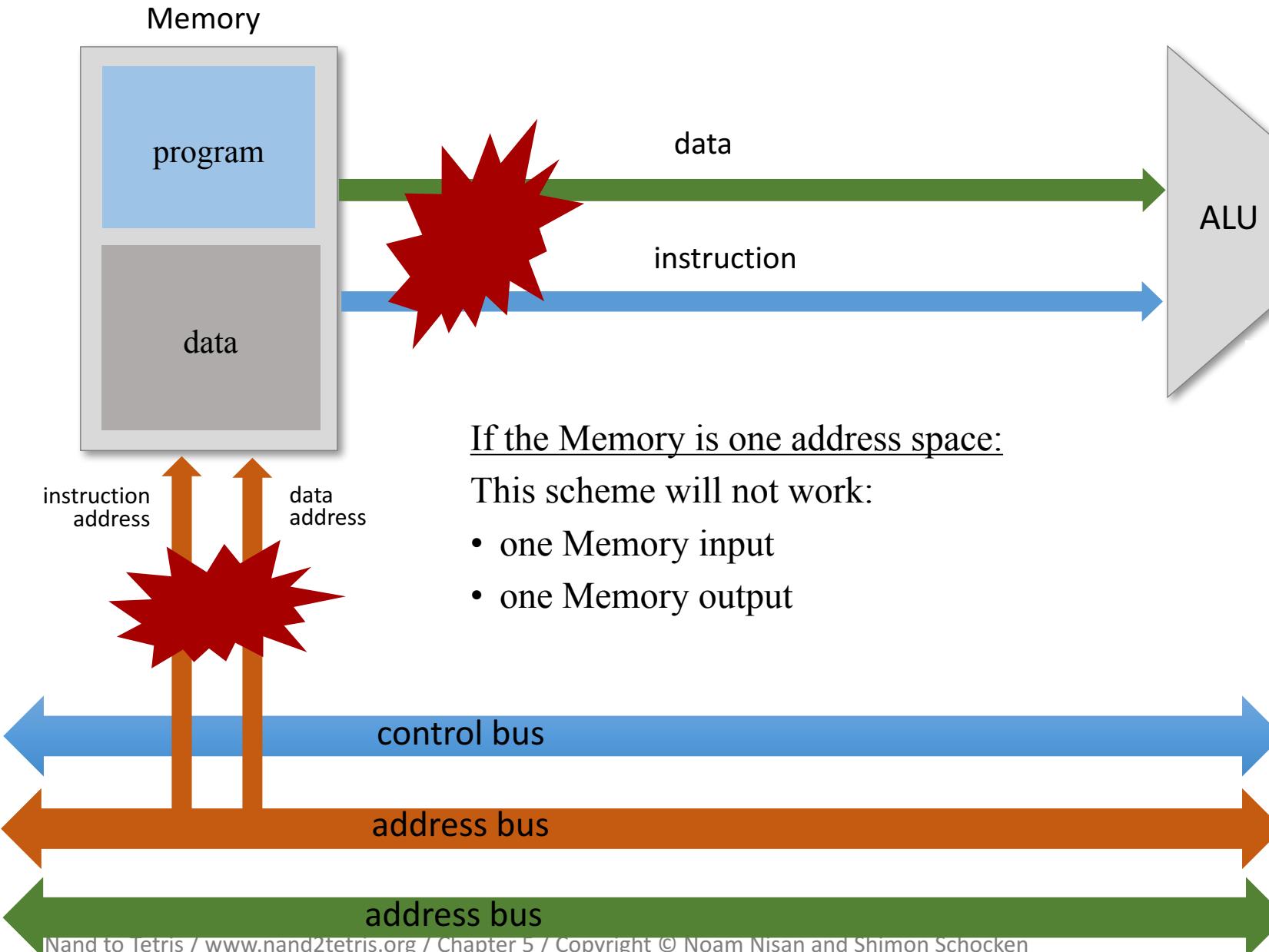
Computer architecture



Fetch – Execute



Fetch – Execute clash

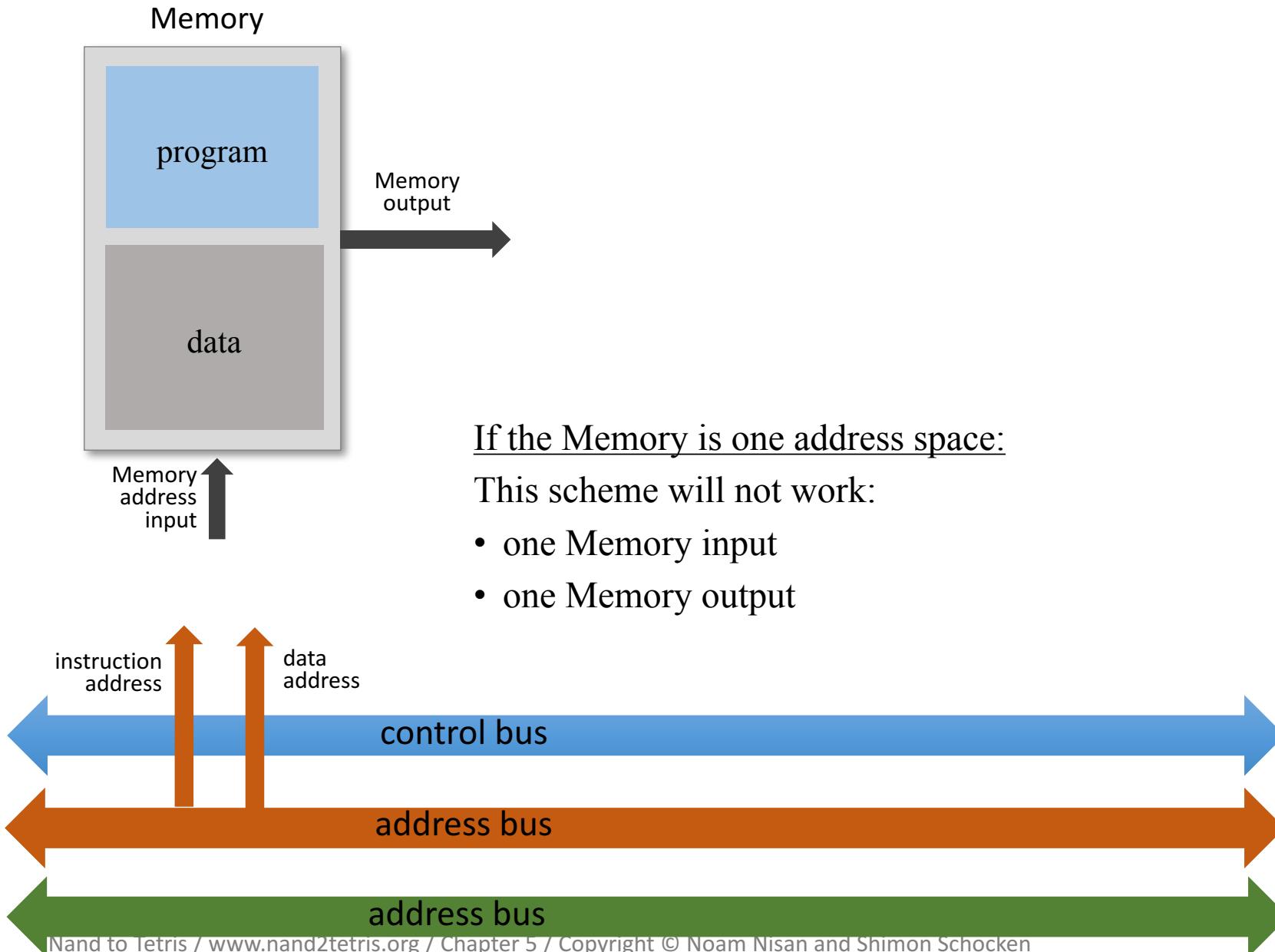


If the Memory is one address space:

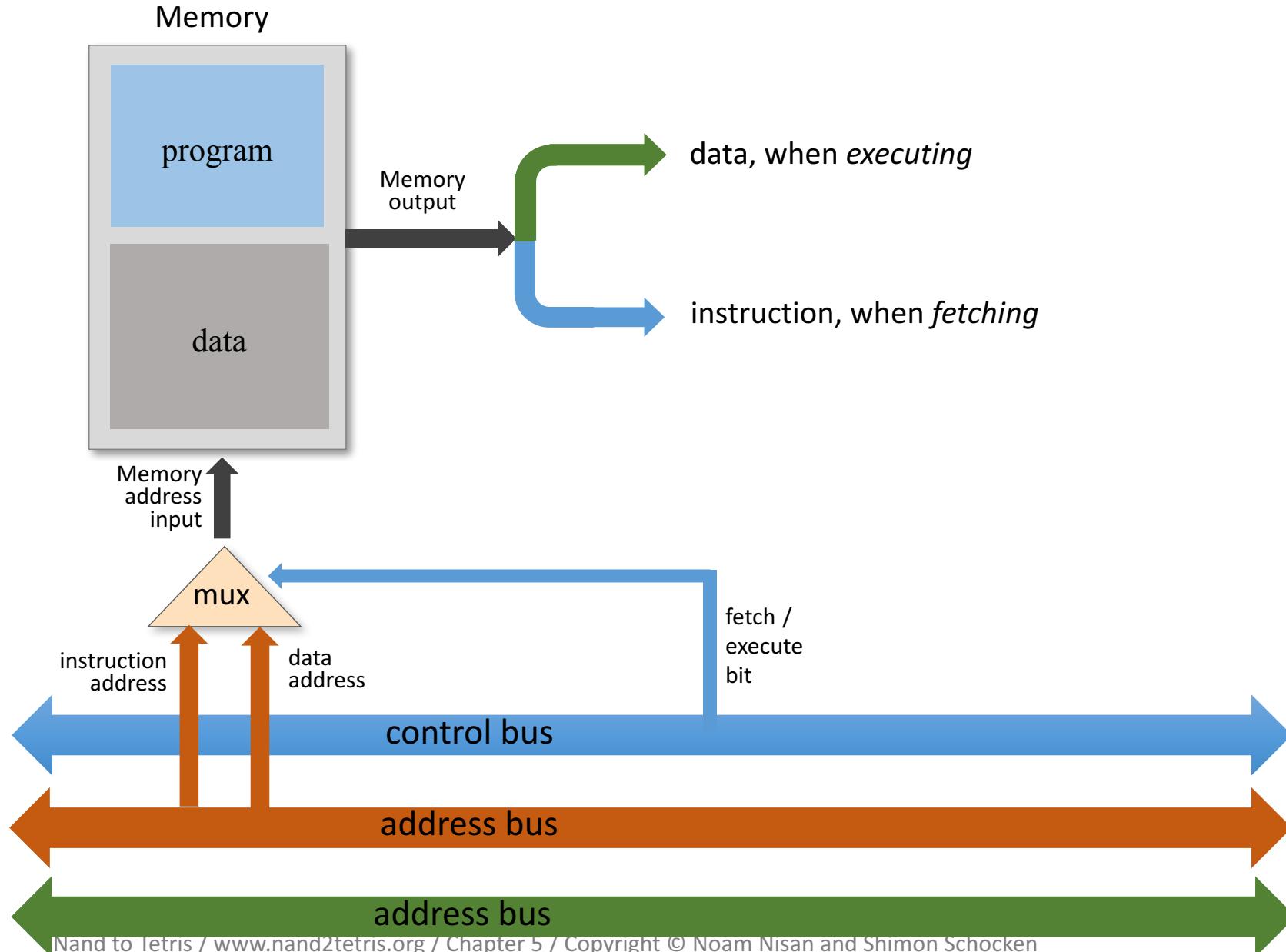
This scheme will not work:

- one Memory input
- one Memory output

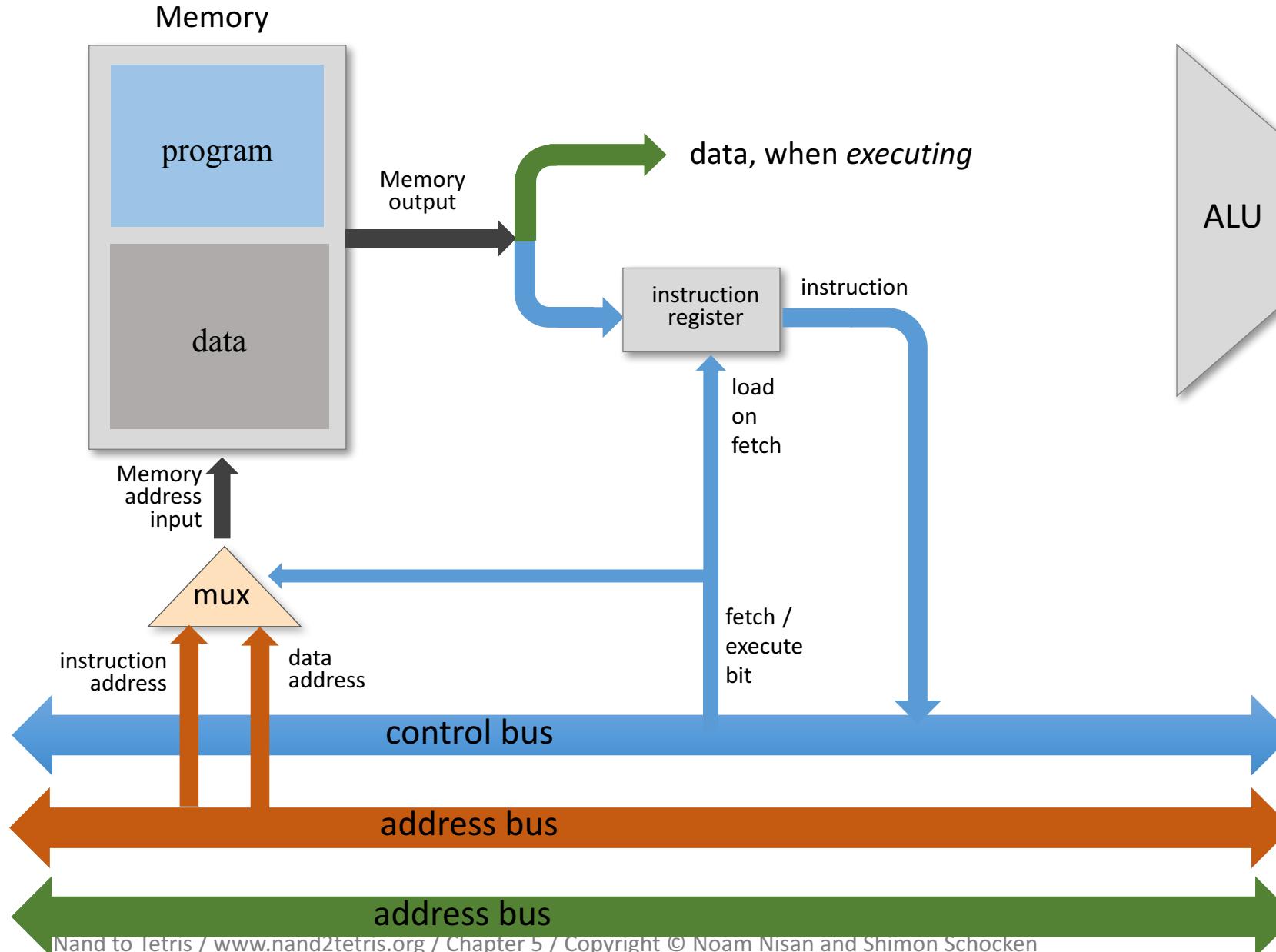
Fetch – Execute clash



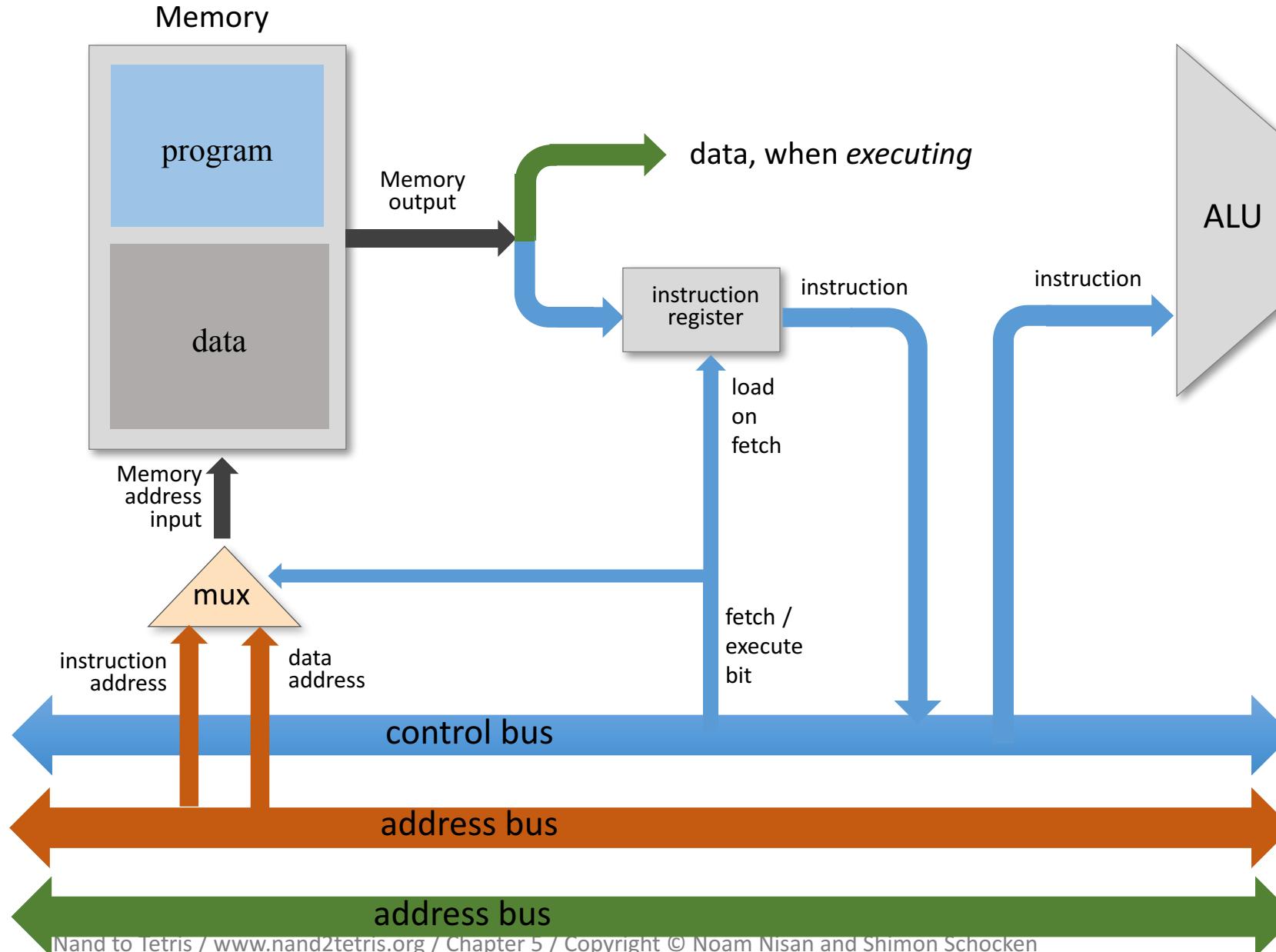
Solution: multiplex



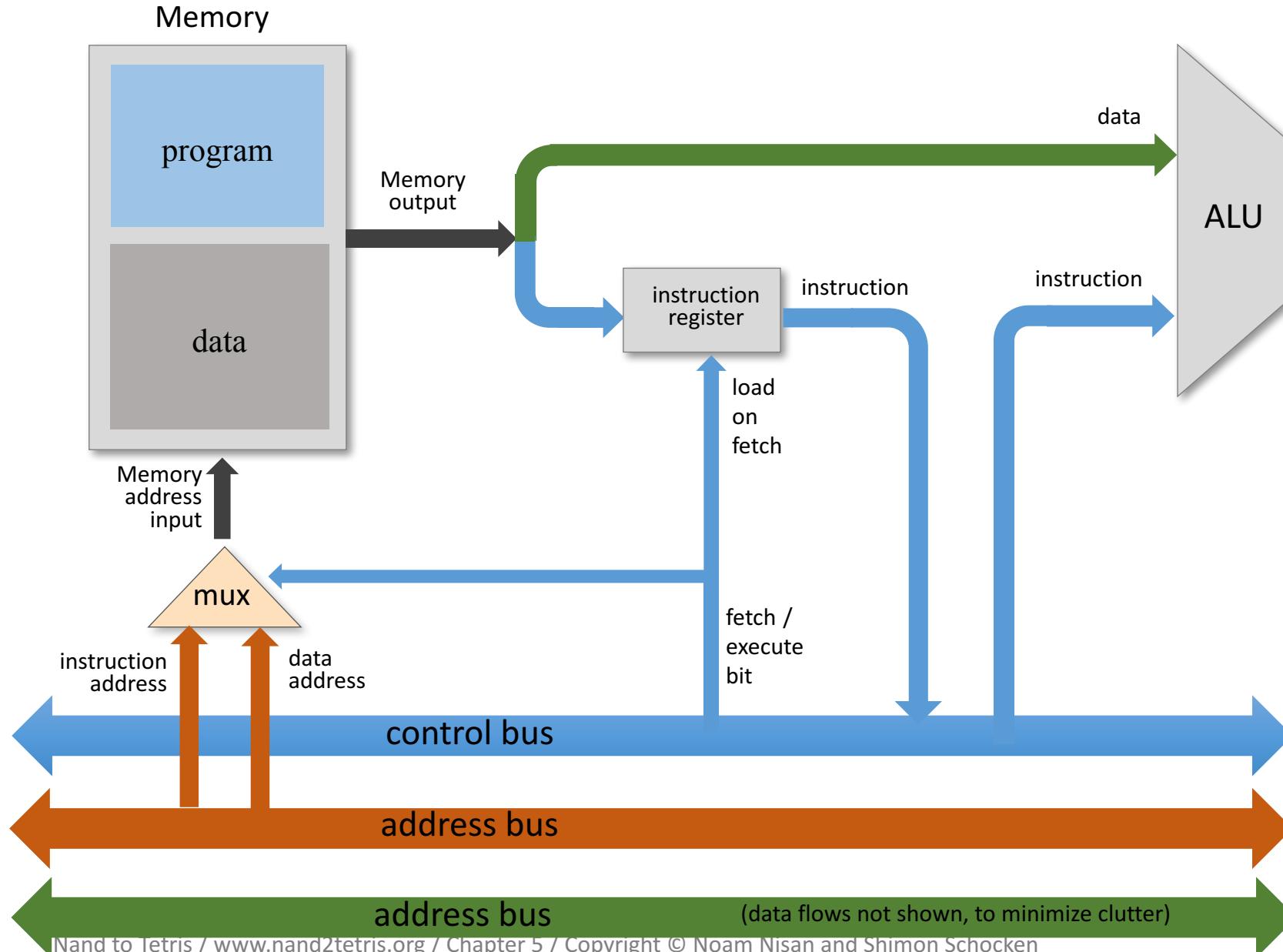
Solution: multiplex, using an instruction register



Solution: multiplex, using an instruction register



Solution: multiplex, using an instruction register



Simpler solution: separate memory units

Variant of von Neumann Architecture (used by the Hack computer):

Two physically separate memory units:

- Instruction memory
 - Data memory
- } Each can be addressed and manipulated separately, and simultaneously

- Advantage:

- Complication avoided

- Disadvantage:

- Two memory chips instead of one
 - The size of the two chips is fixed.

Sometimes called
“Harvard Architecture”

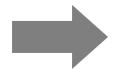
Computer Architecture: lecture plan



Von Neumann Architecture



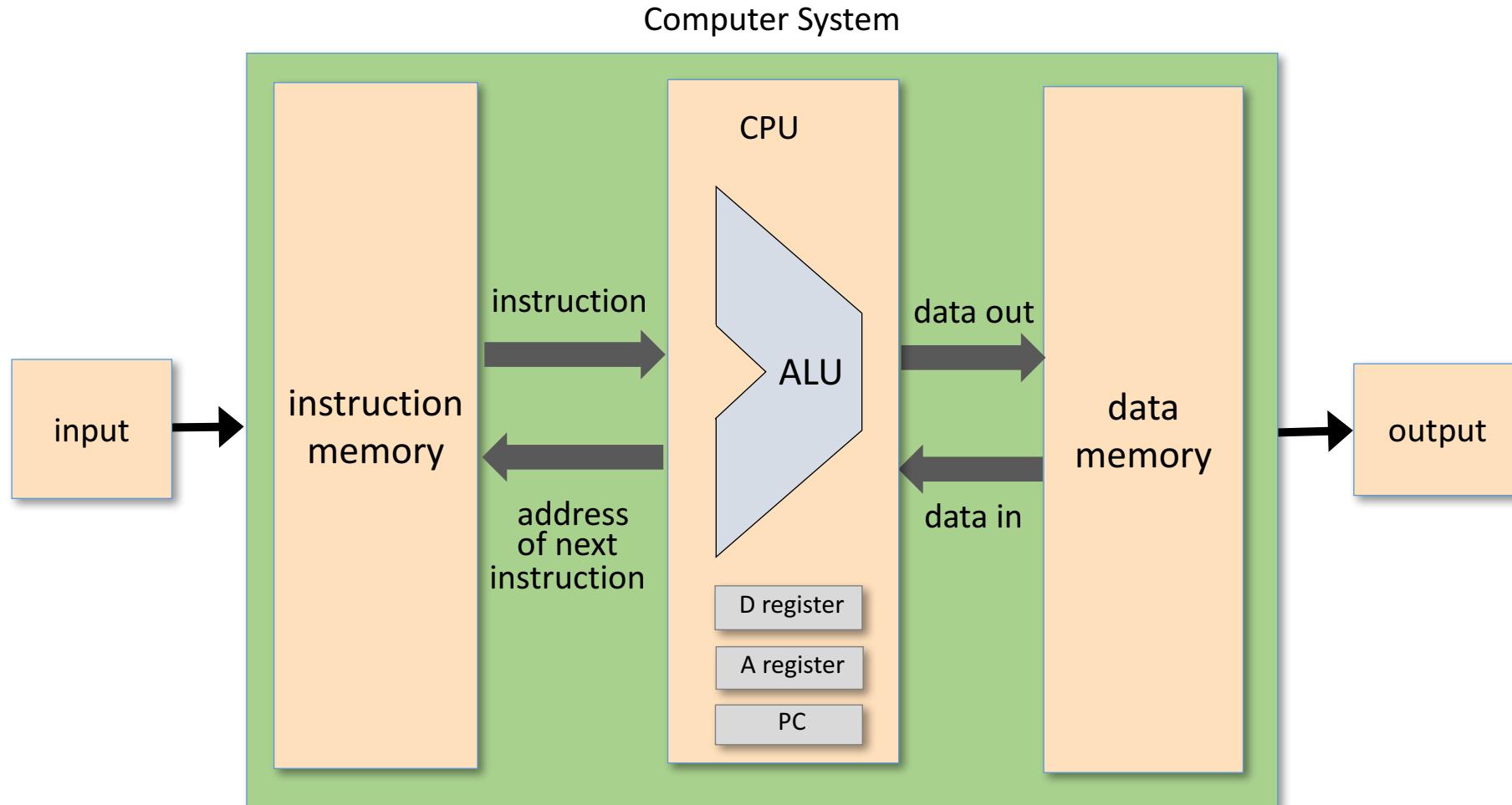
Fetch-Execute Cycle



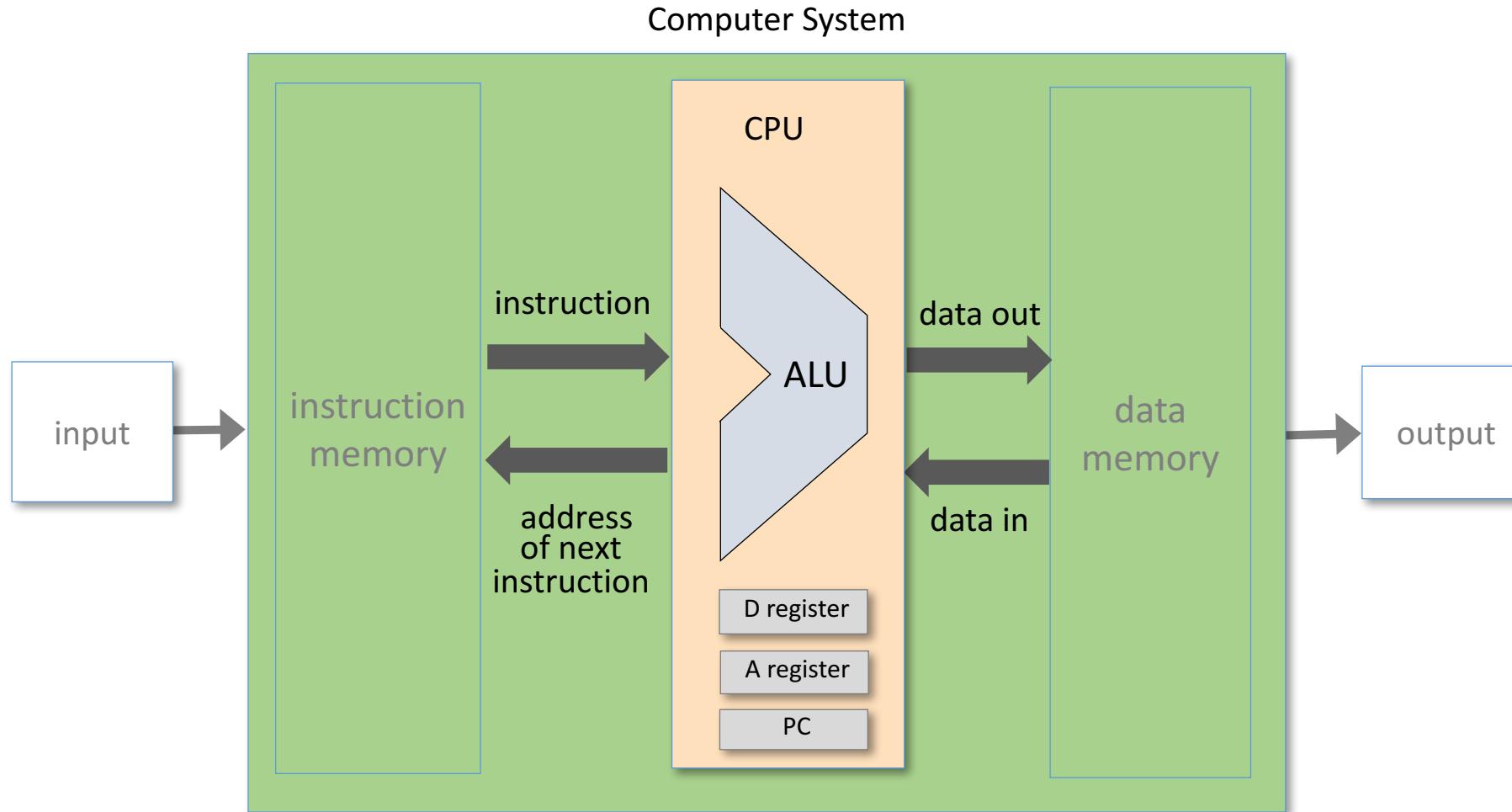
The Hack CPU

- The Hack Computer
- Project 5 Overview

Hack computer



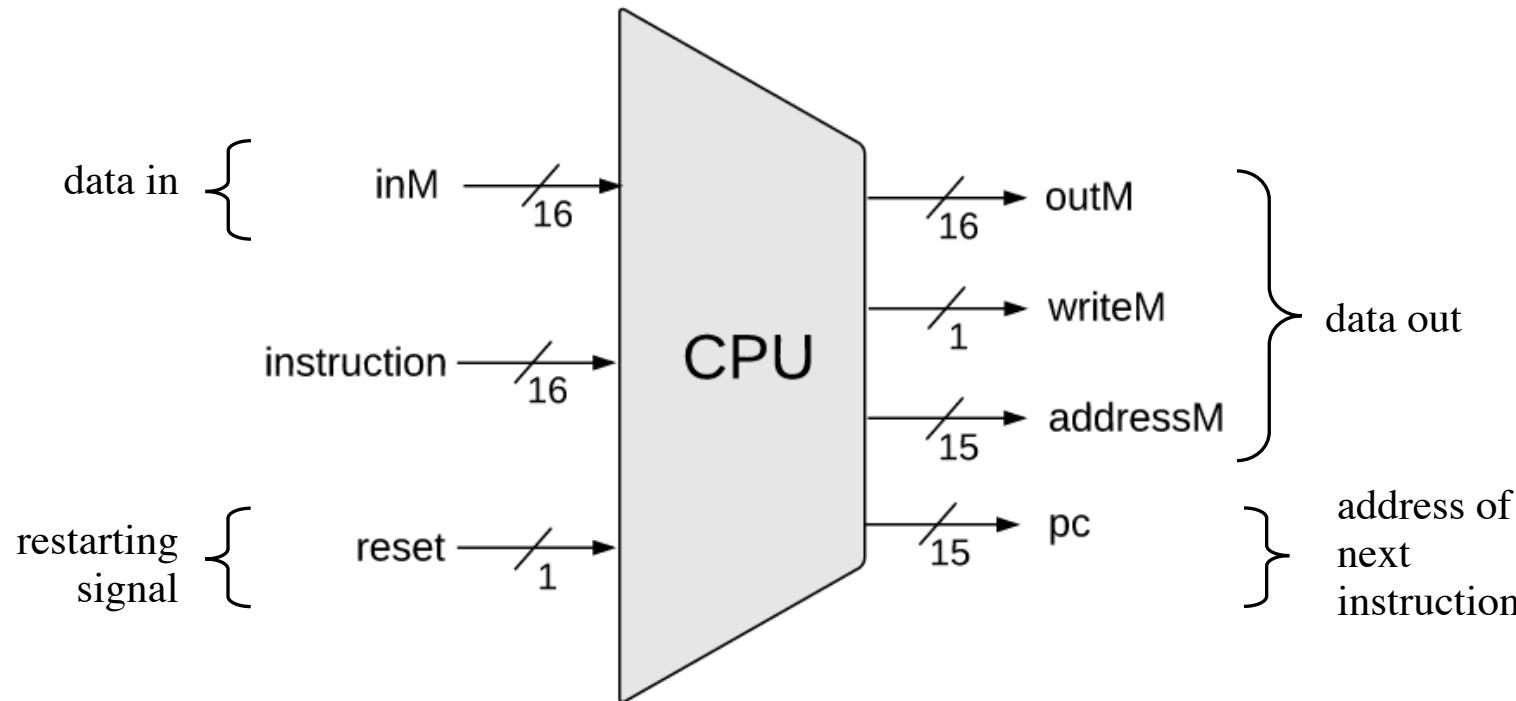
Hack CPU



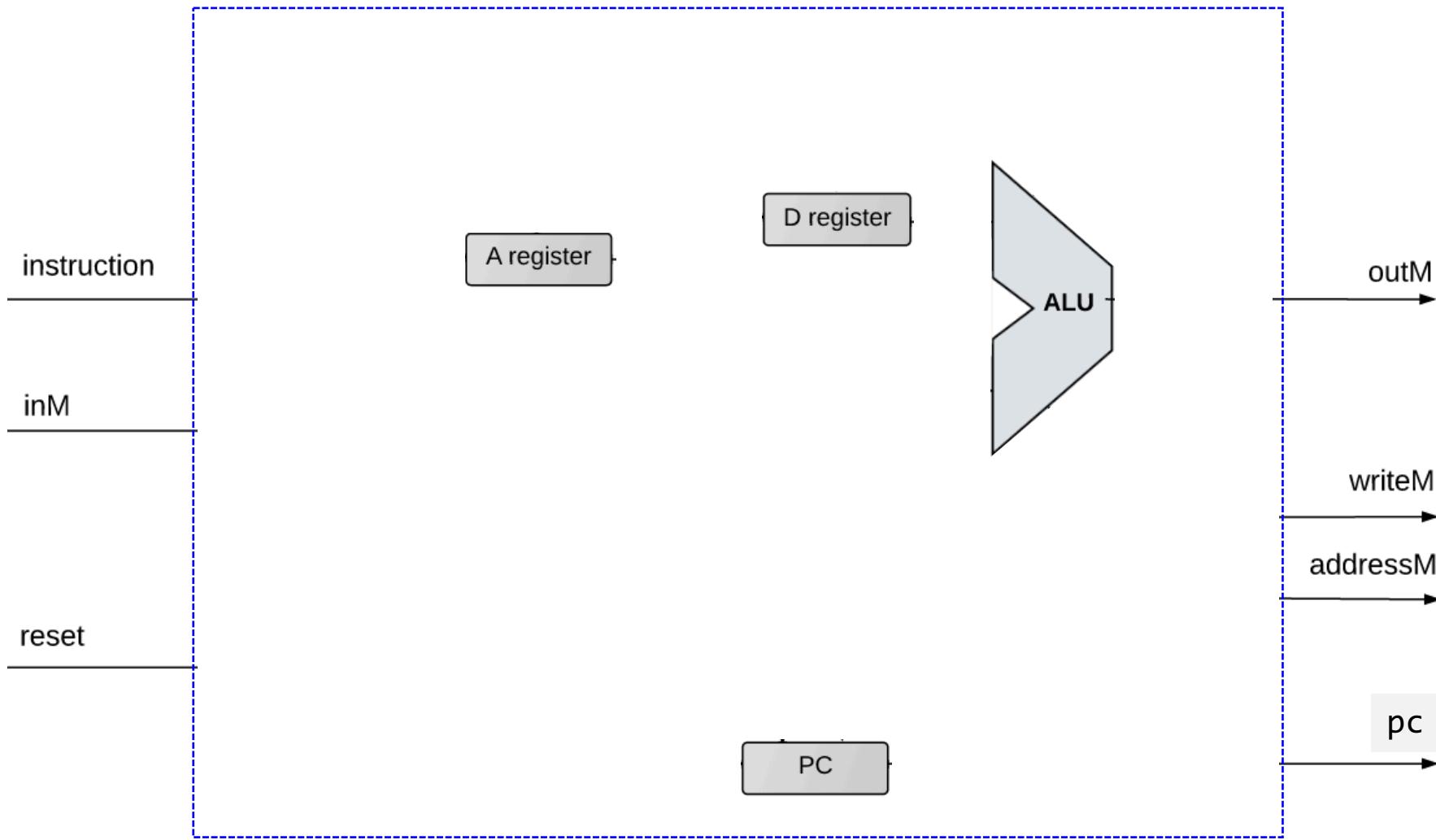
Hack CPU: A 16-bit processor, designed to:

- Execute the current instruction: $\text{dataOut} = \text{instruction}(\text{dataIn})$
- Figure out which instruction to execute next.

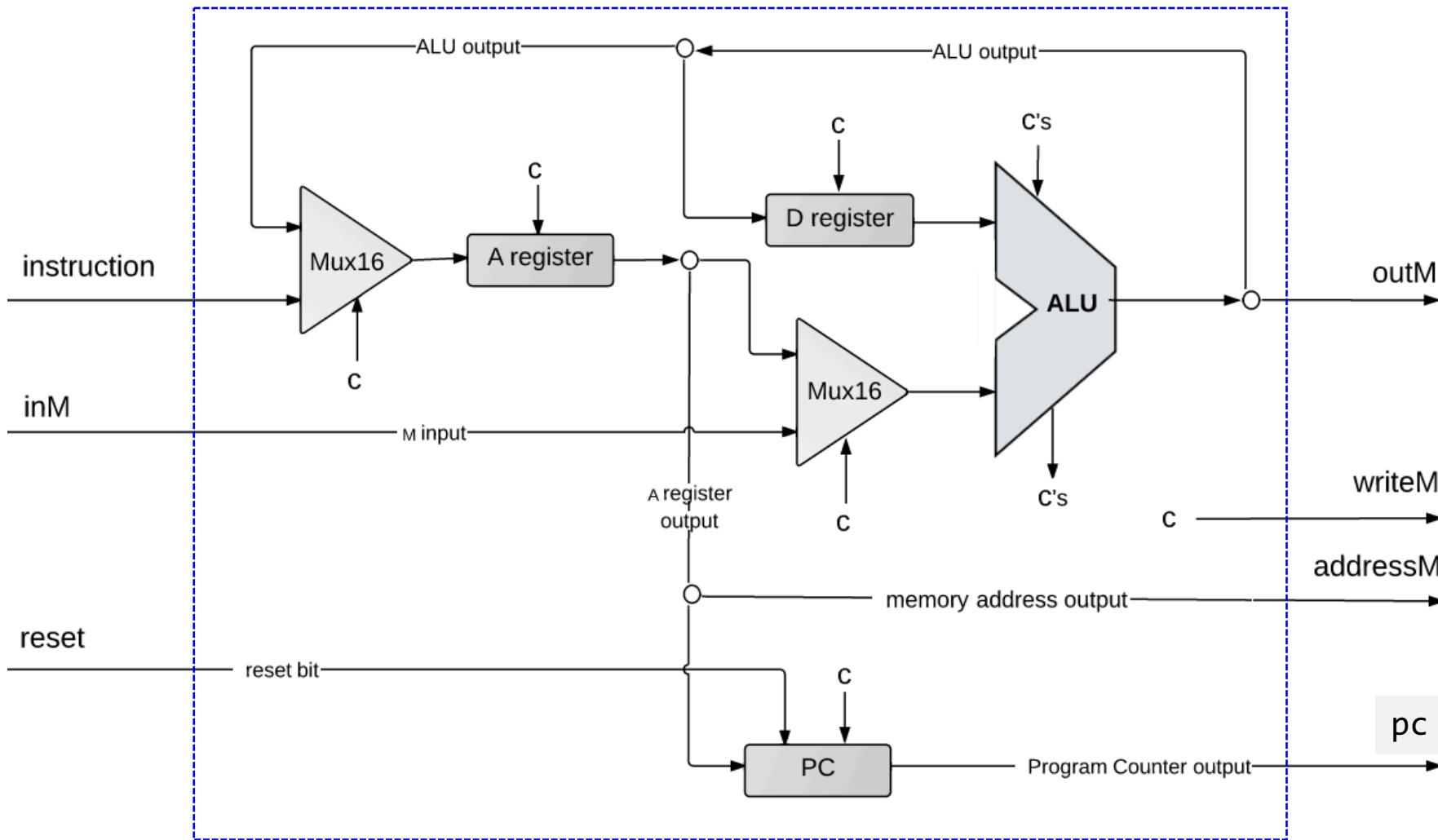
Hack CPU Interface



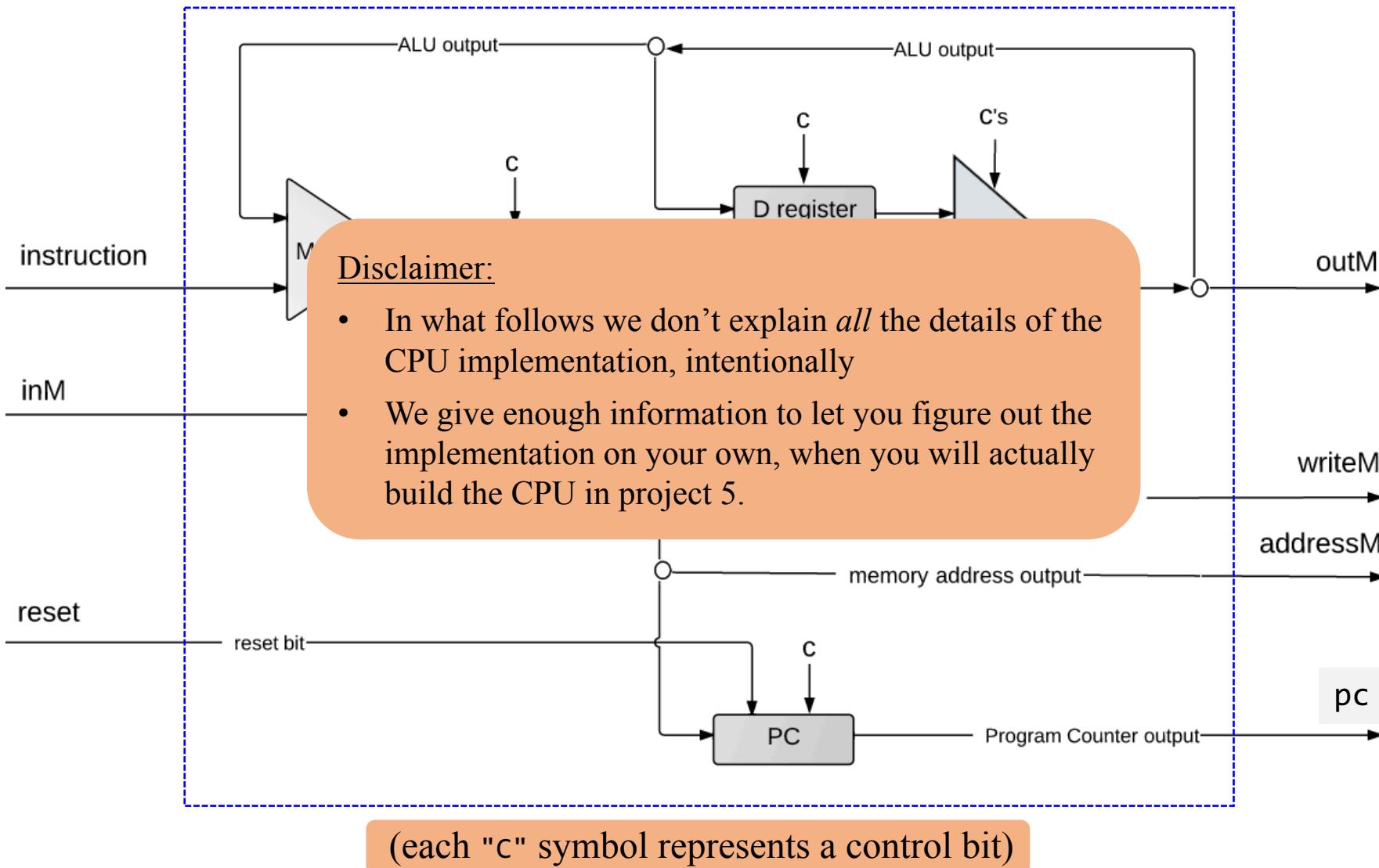
Hack CPU Implementation



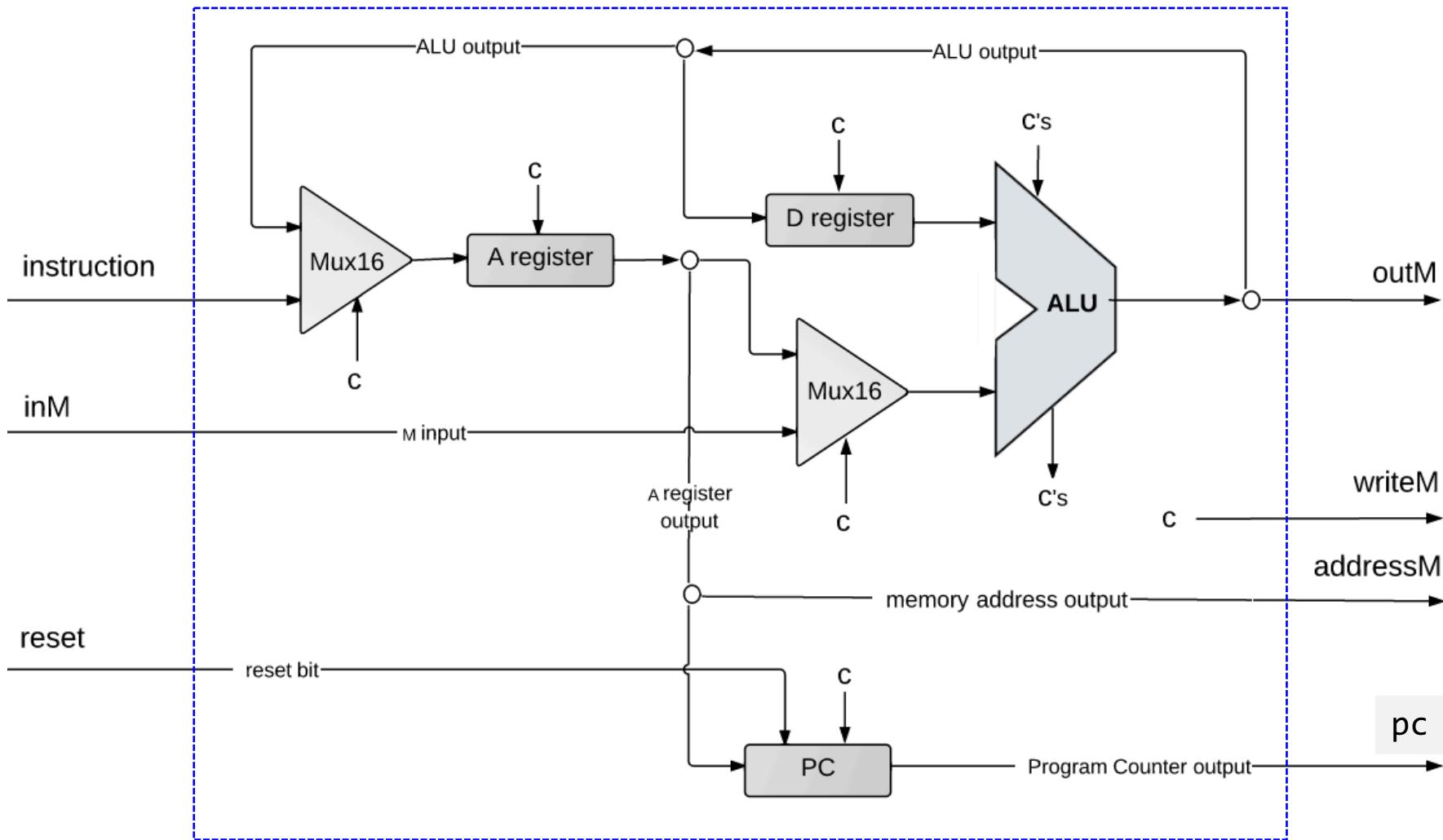
Hack CPU Implementation



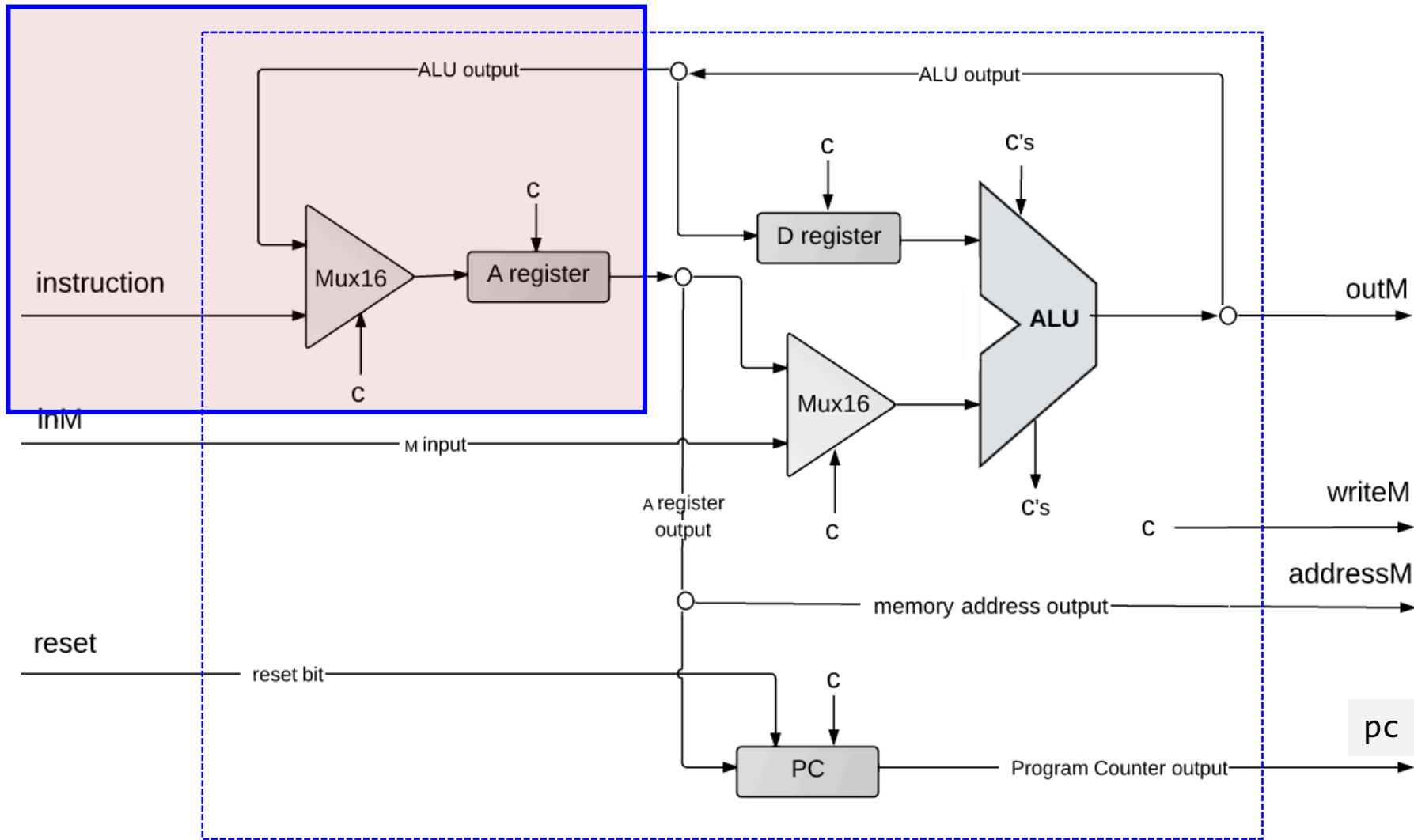
Hack CPU Implementation



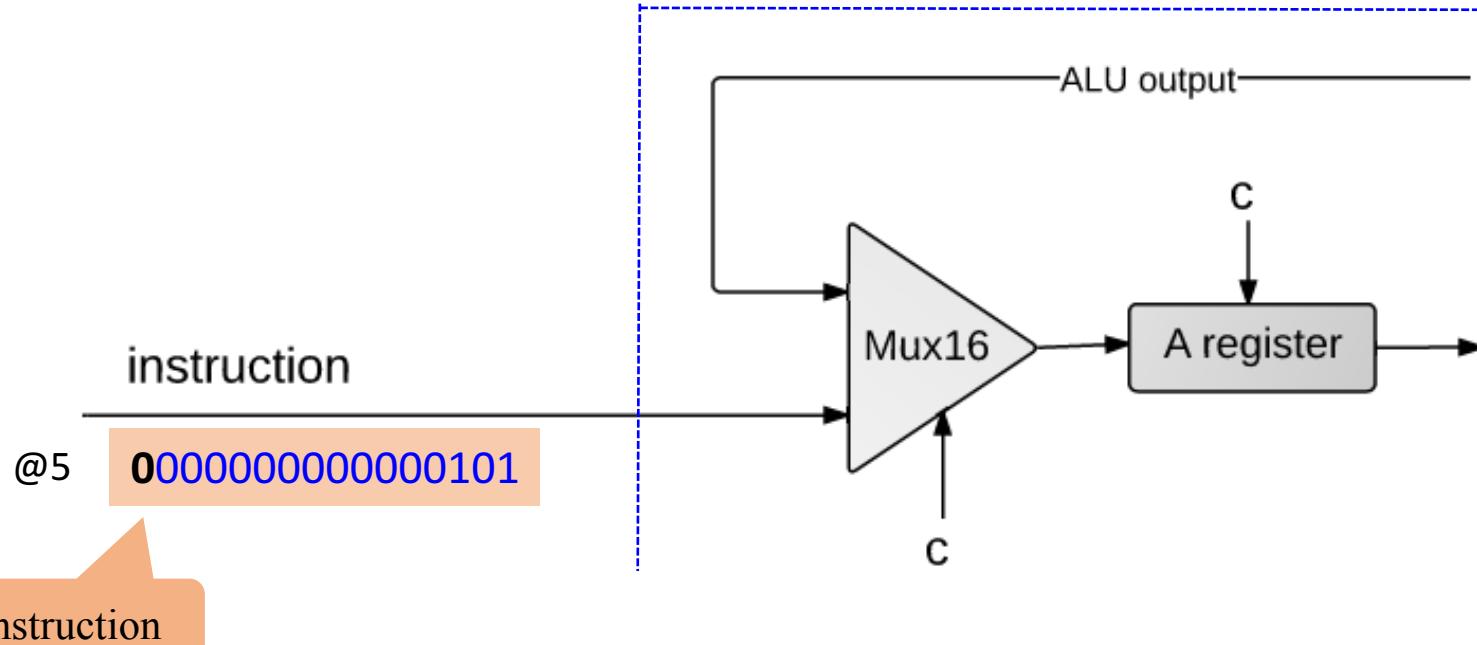
CPU operation



CPU operation: instruction handling

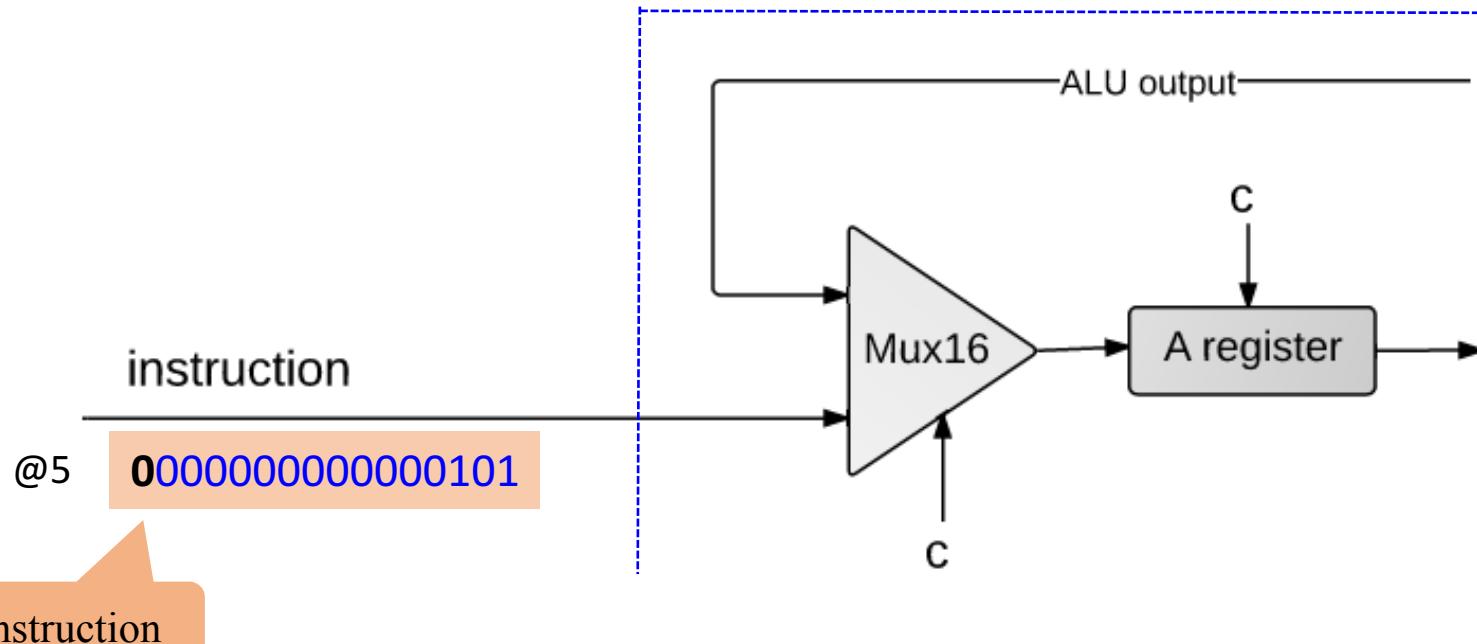


CPU operation: instruction handling



A-instruction

CPU operation: handling A-instructions

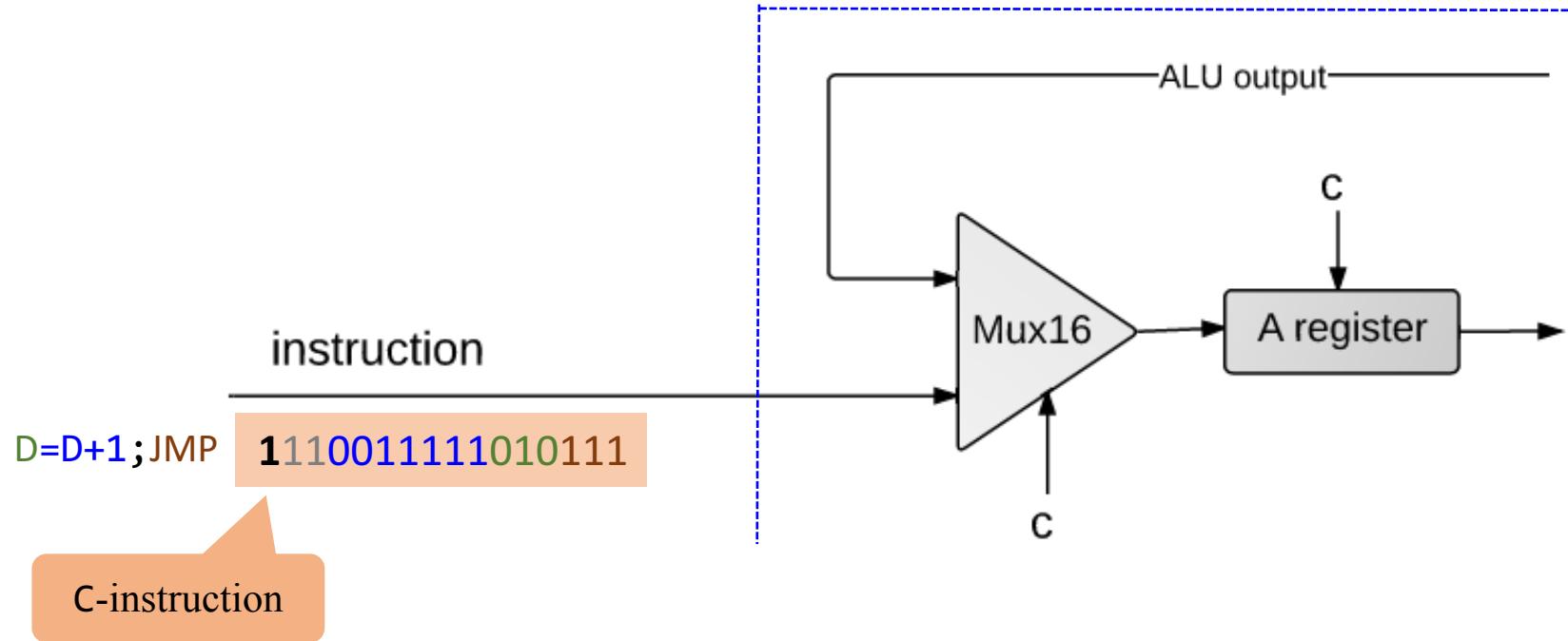


A-instruction

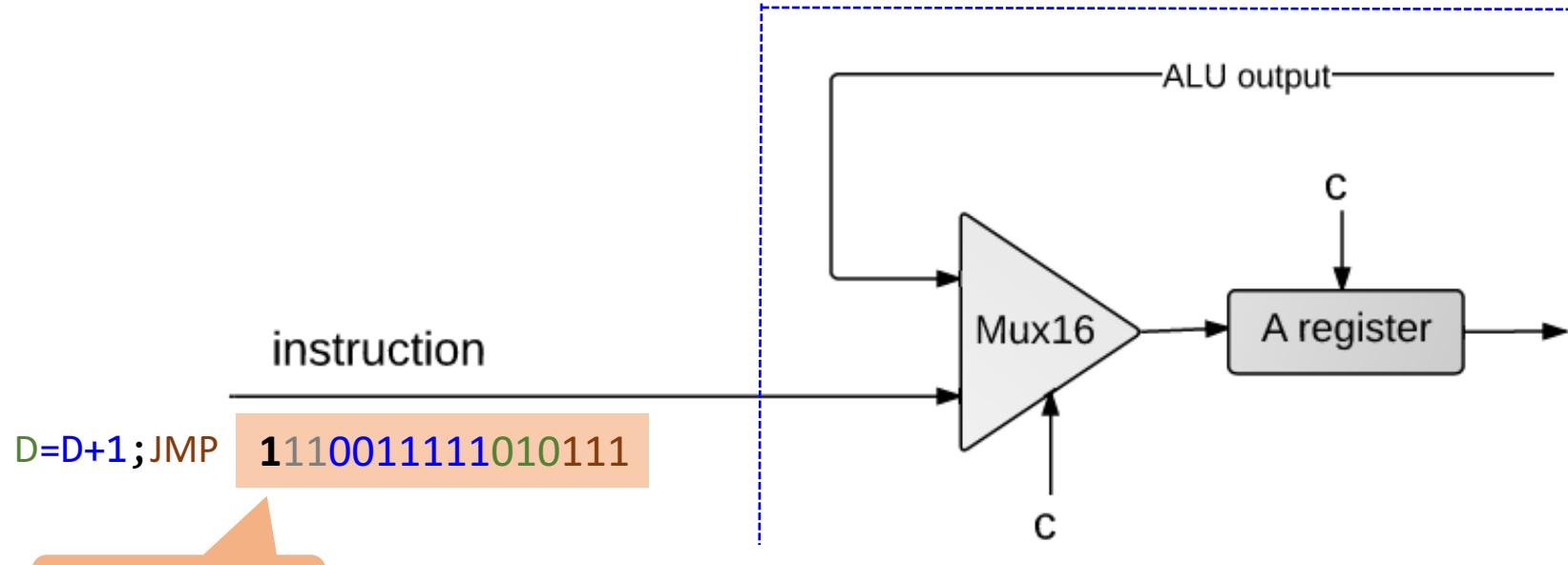
CPU handling of an A-instruction:

- 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: handling c-instructions

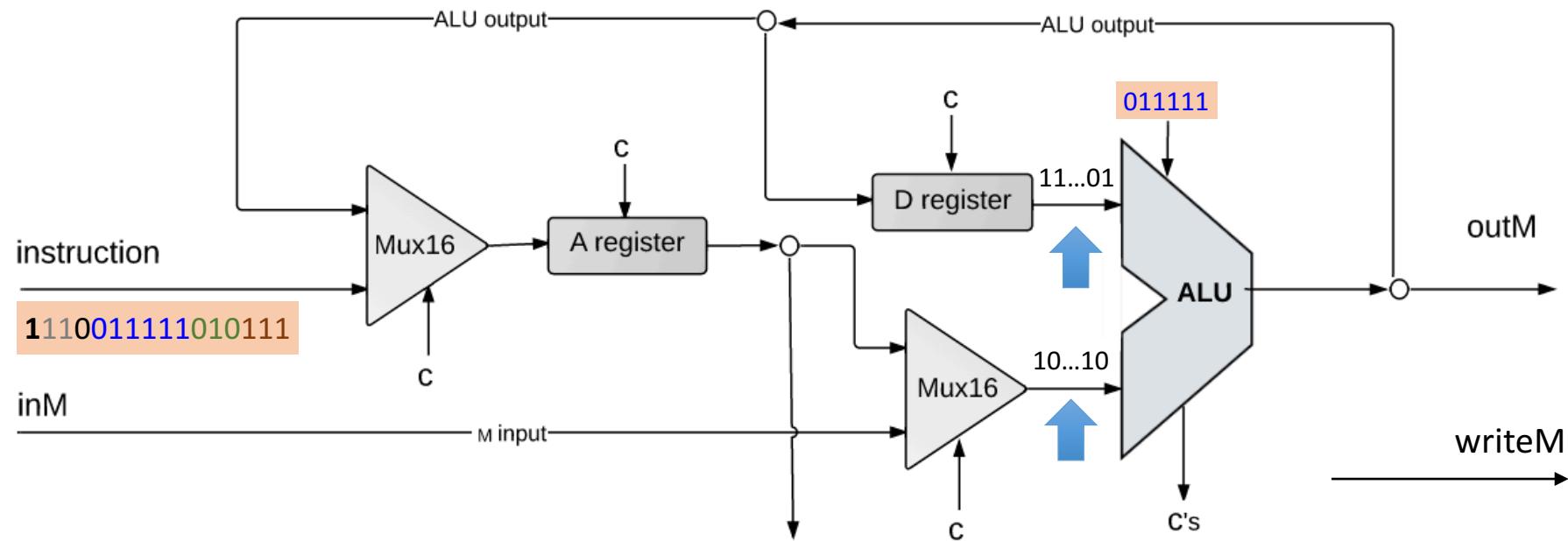


C-instruction

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



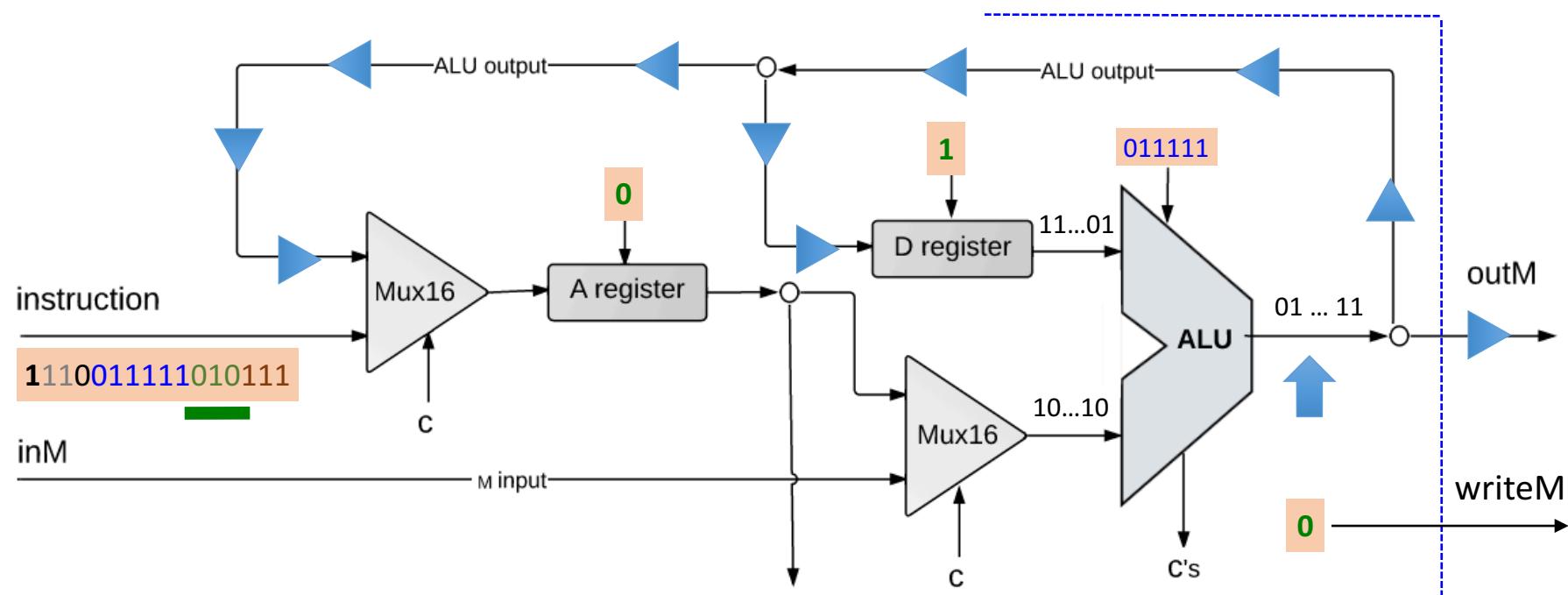
ALU data inputs:

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

ALU control inputs:

- control bits
(from the instruction)

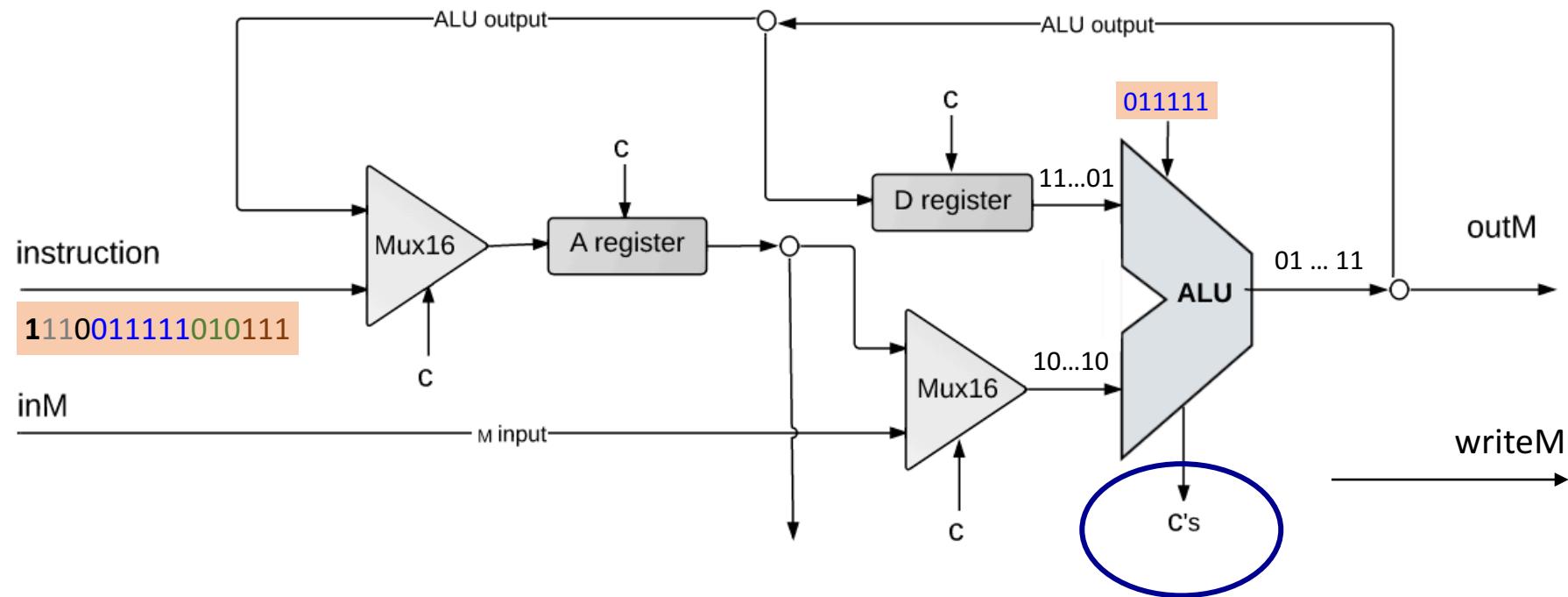
CPU operation: handling c-instructions



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.

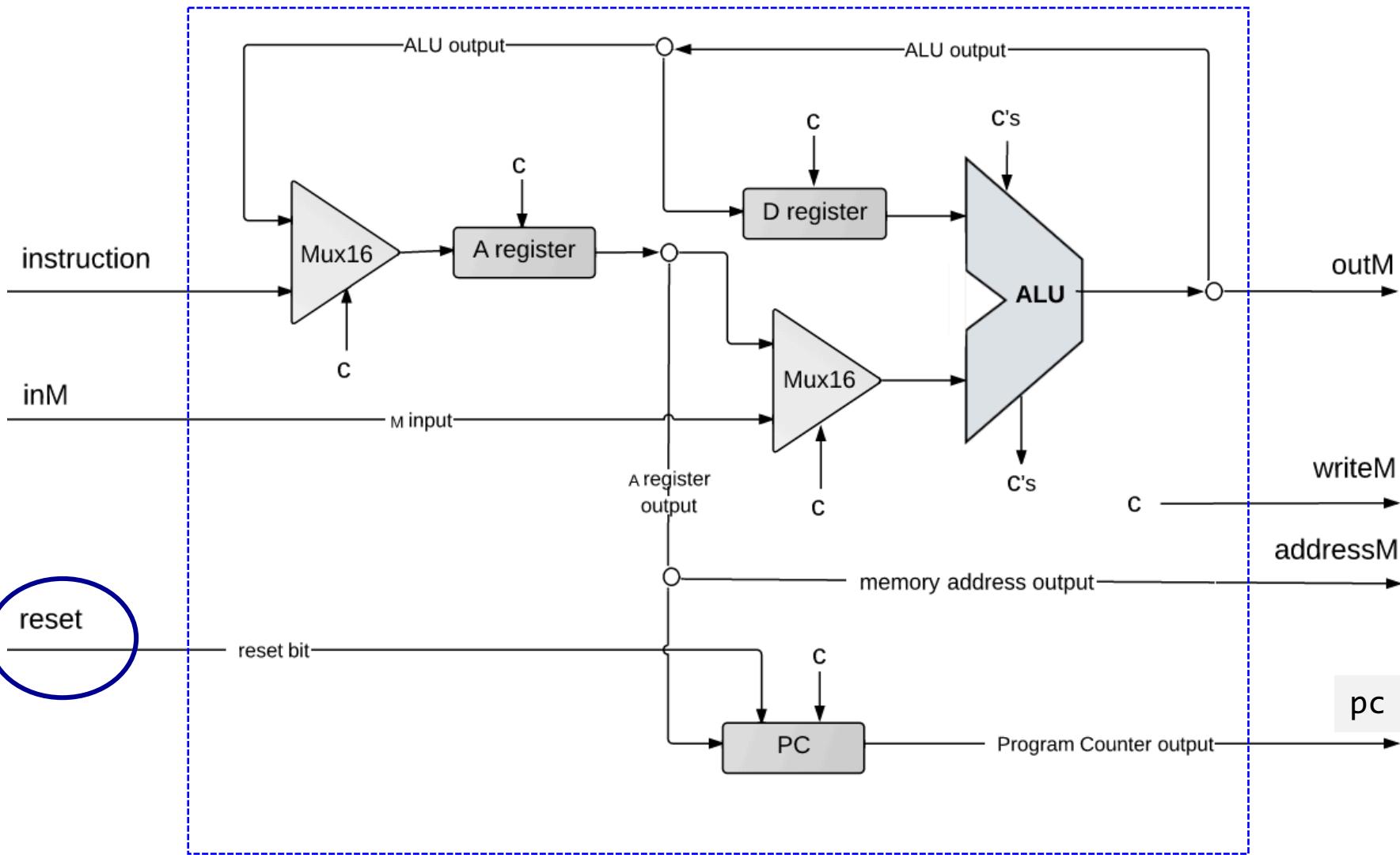
CPU operation: handling c-instructions



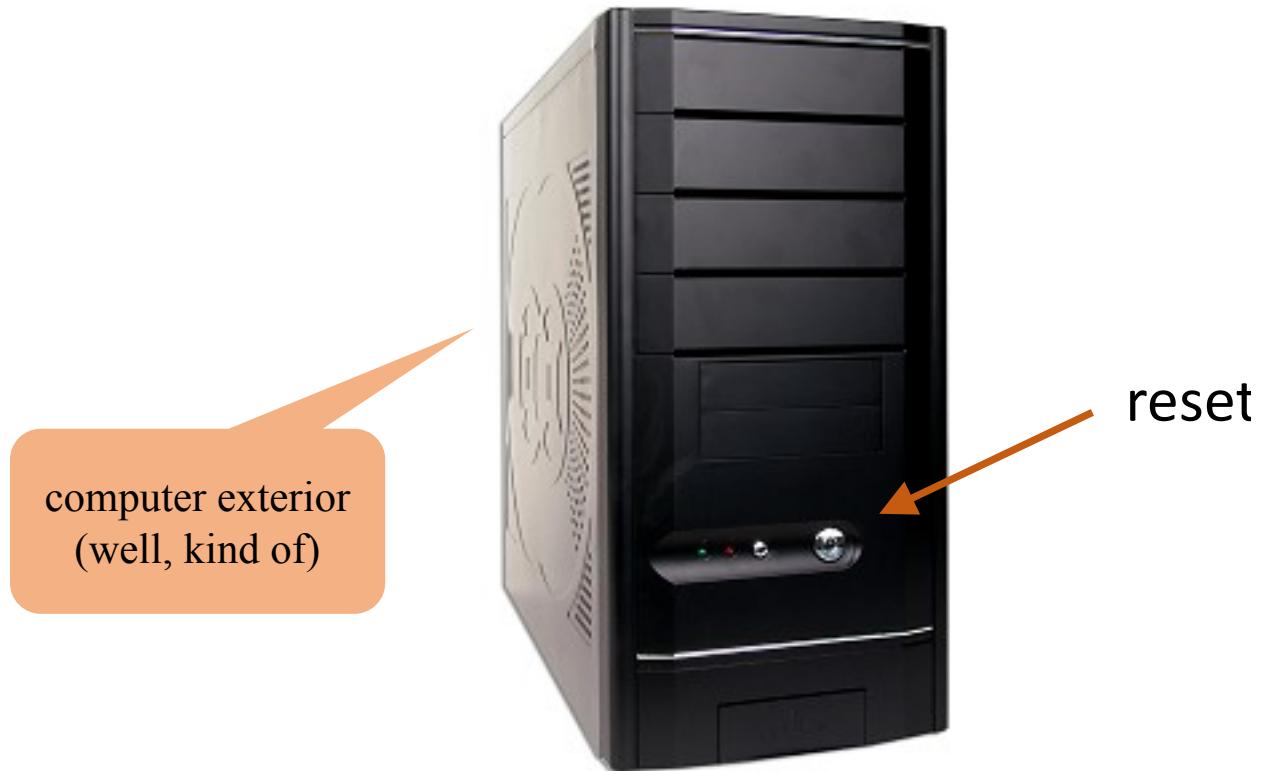
ALU control outputs:

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

CPU operation: control

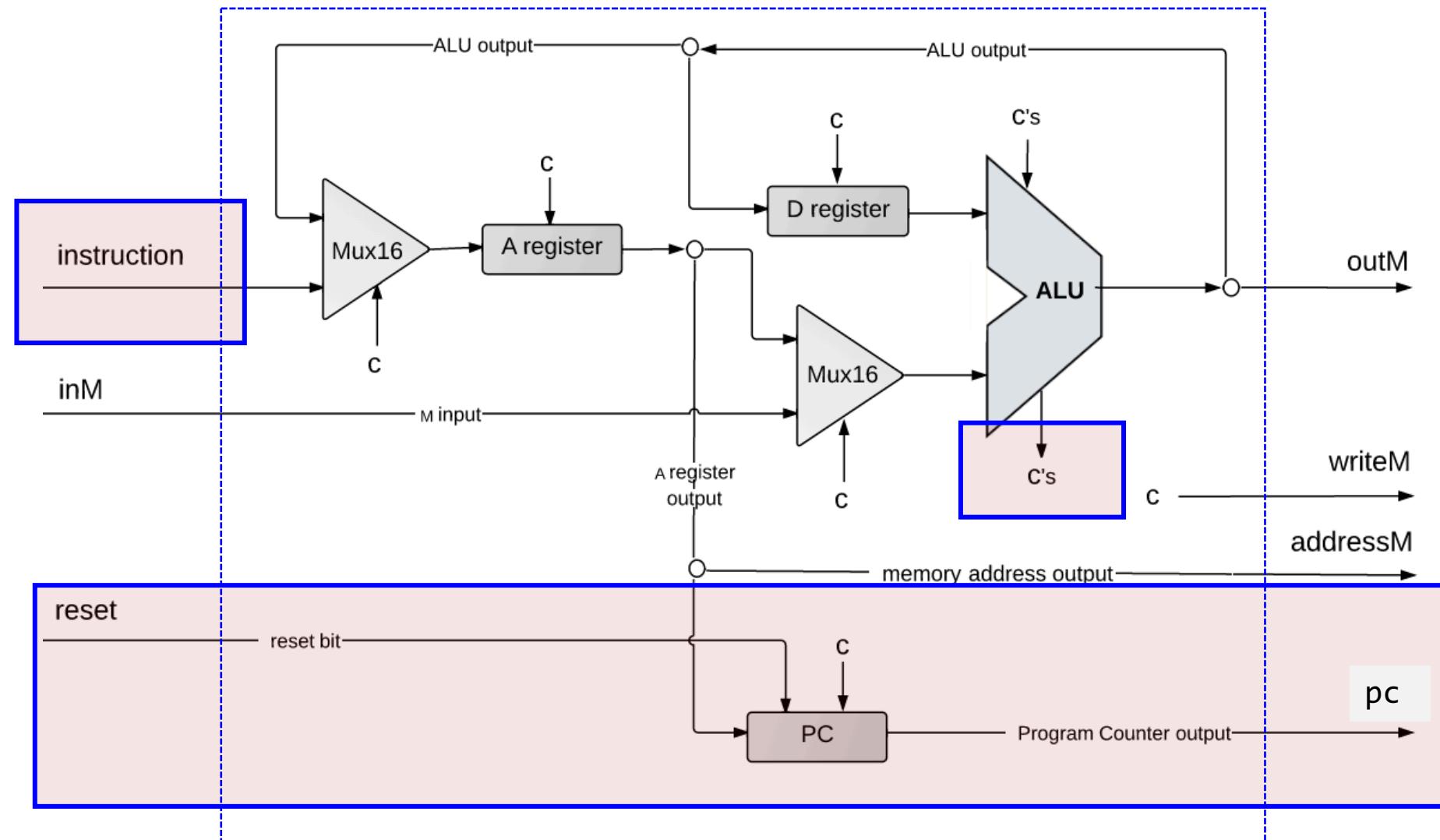


CPU operation: control

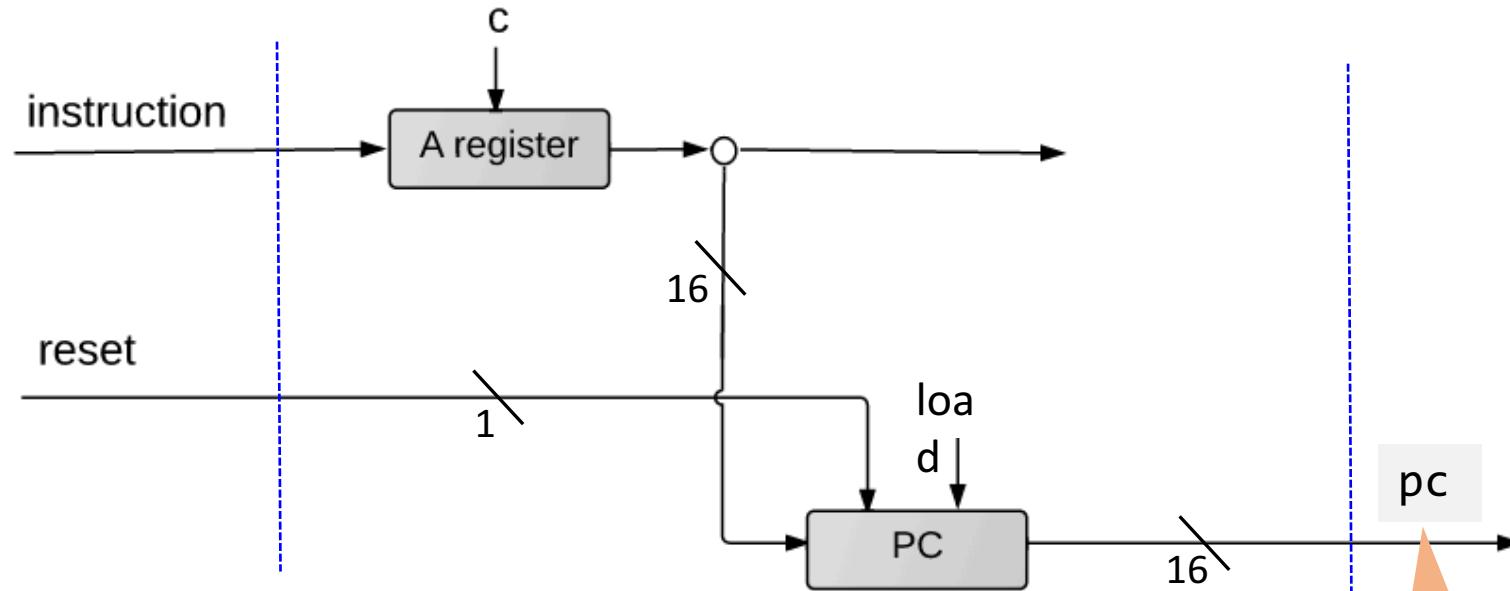


- The computer is loaded with some program;
- Pushing **reset** causes the program to start running.

CPU operation: control



CPU operation: control



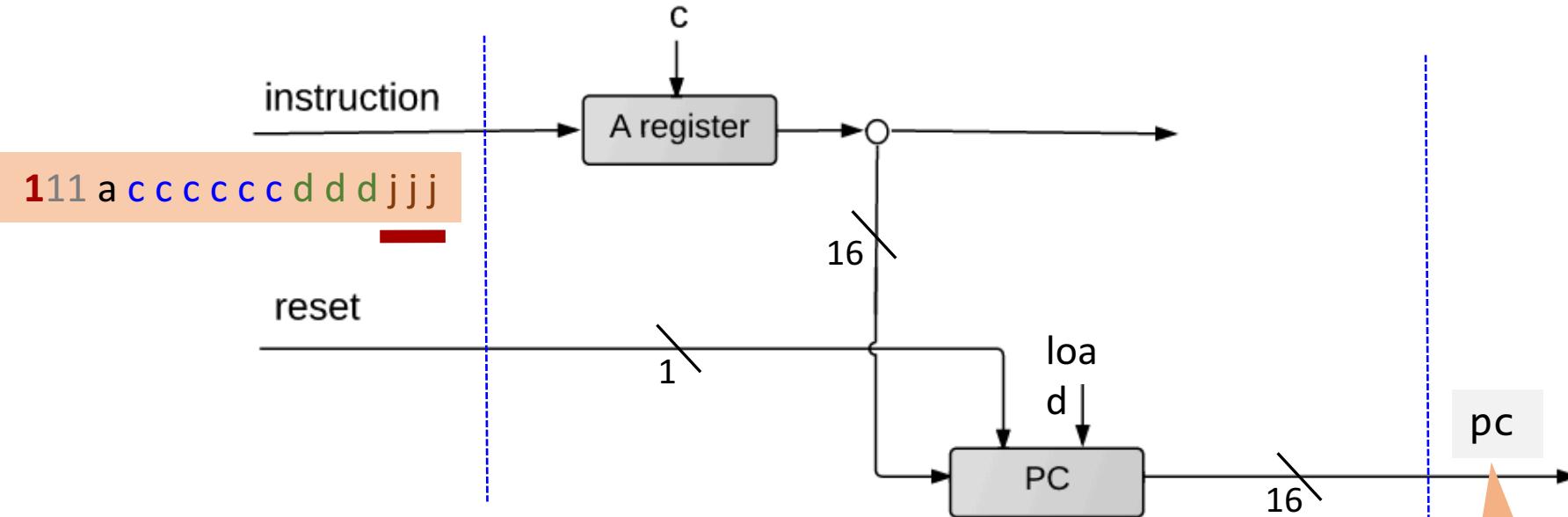
PC operation (abstraction)

Emits the address of the next instruction:

- restart: $PC = 0$
- no jump: $PC++$
- goto: $PC=A$
- conditional goto: $\text{if } (condition) \text{ } PC=A \text{ else } PC++$

address of next
instruction

CPU operation: control



PC operation (implementation)

```
if (reset==1) PC = 0  
else
```

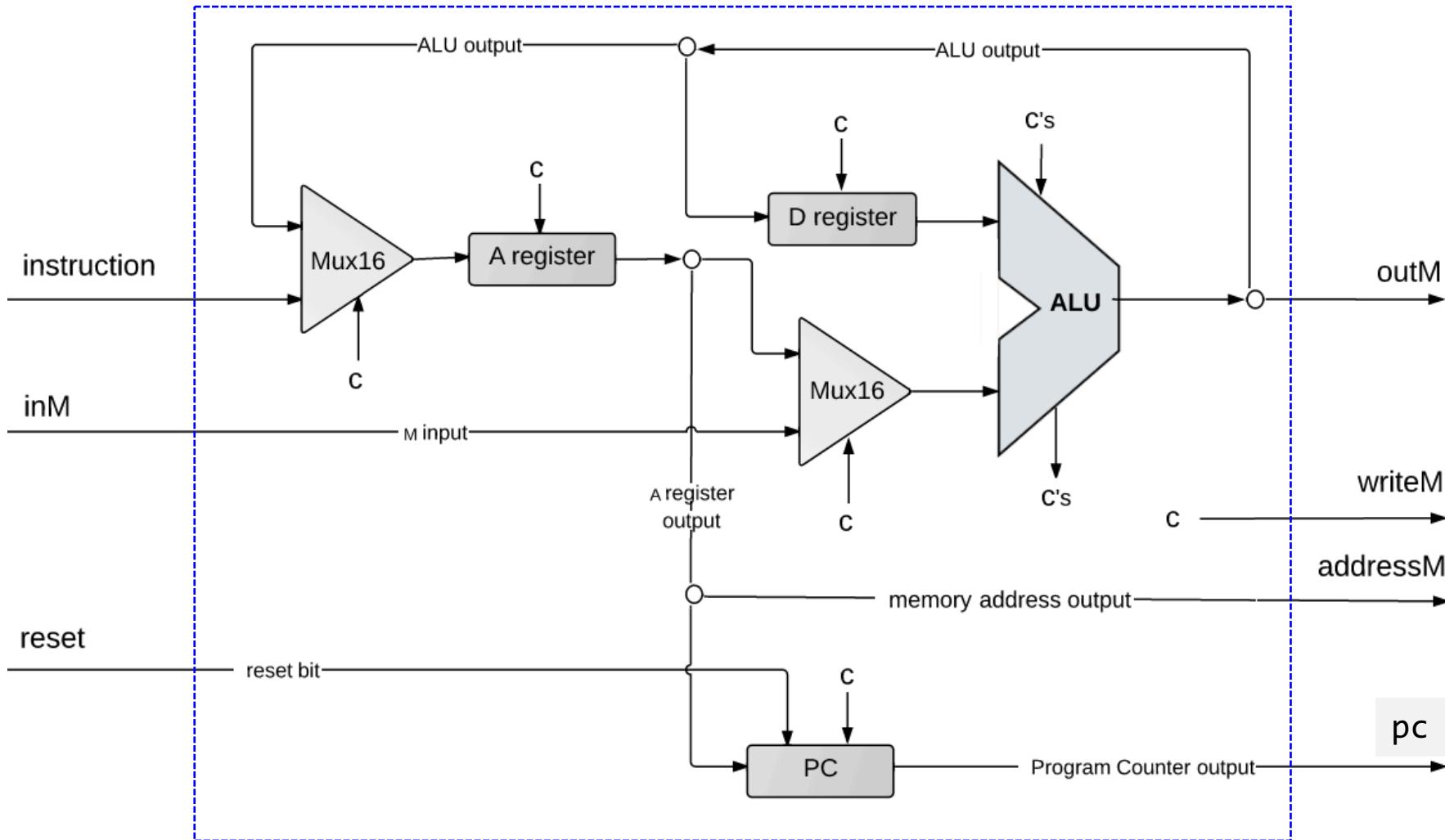
// in the course of handling the current instruction:

$load = f(\text{jump bits, ALU control outputs})$

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

else PC++ // next instruction

Hack CPU Implementation



That's It!

Computer Architecture: lecture plan

✓ Von Neumann Architecture

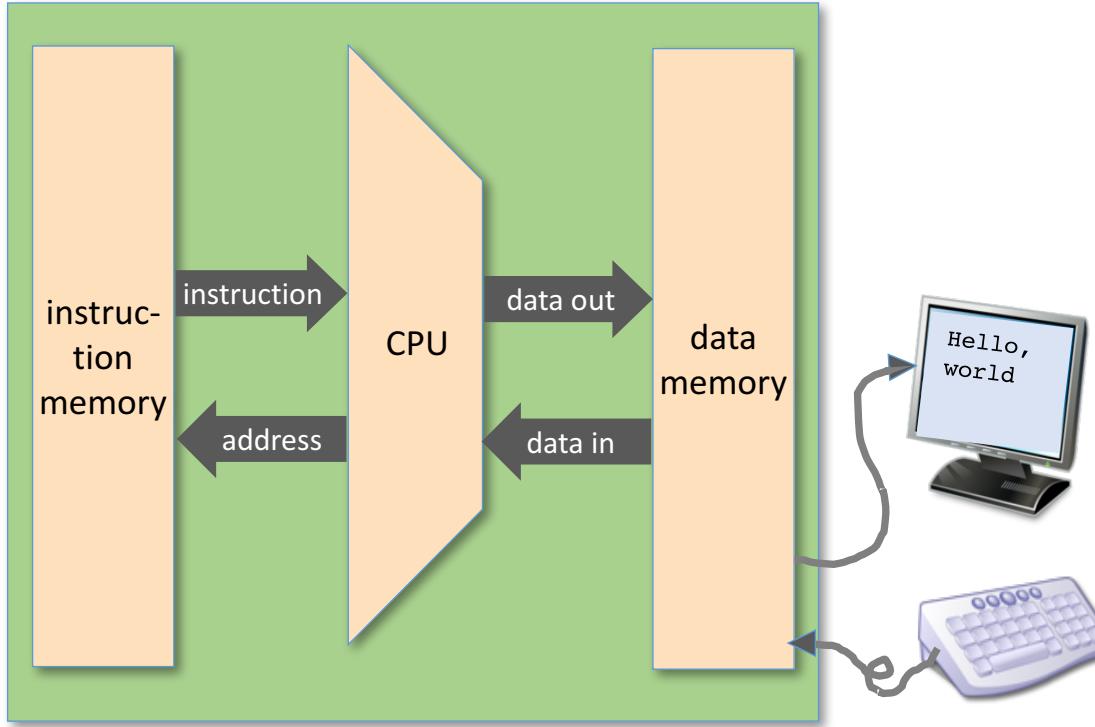
✓ Fetch-Execute Cycle

✓ The Hack CPU

→ The Hack Computer

- Project 5 Overview

Hack Computer



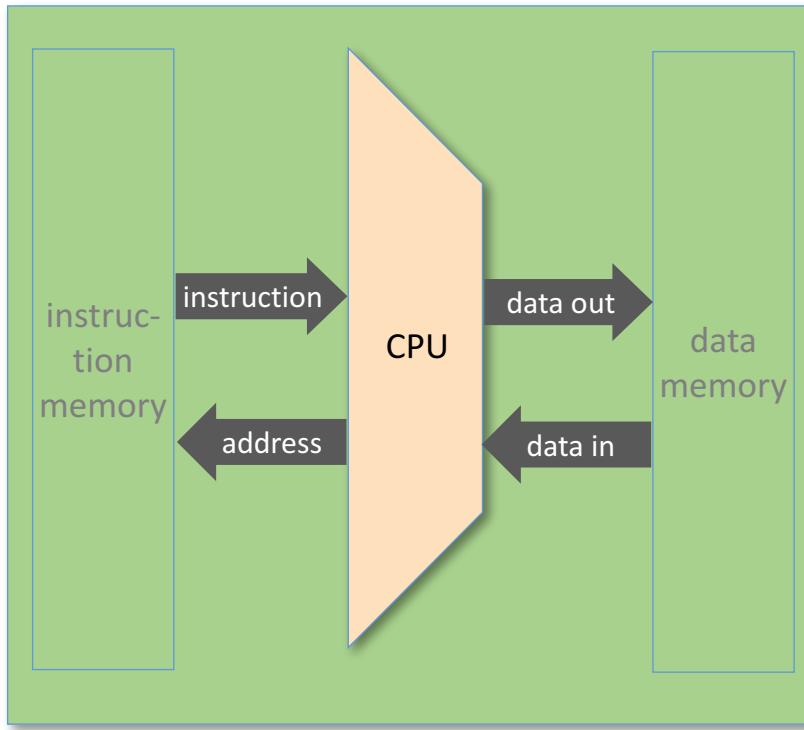
Abstraction:

A computer capable of running programs written in the Hack machine language

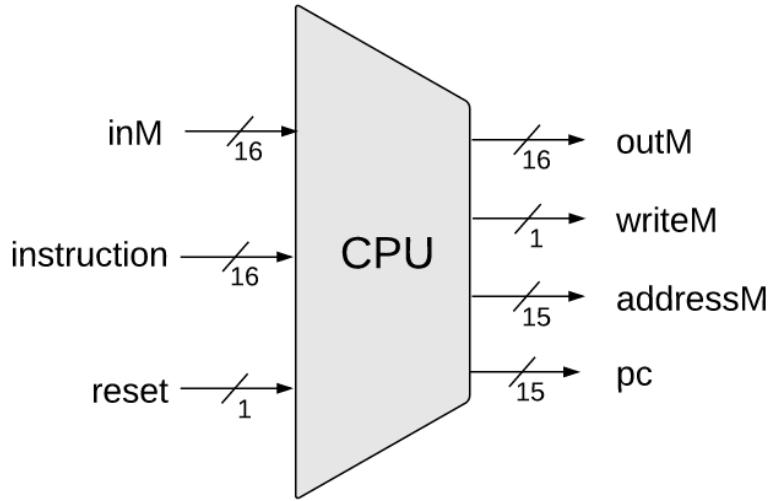
Implementation:

Built from the Hack chip-set.

Hack CPU



Hack CPU



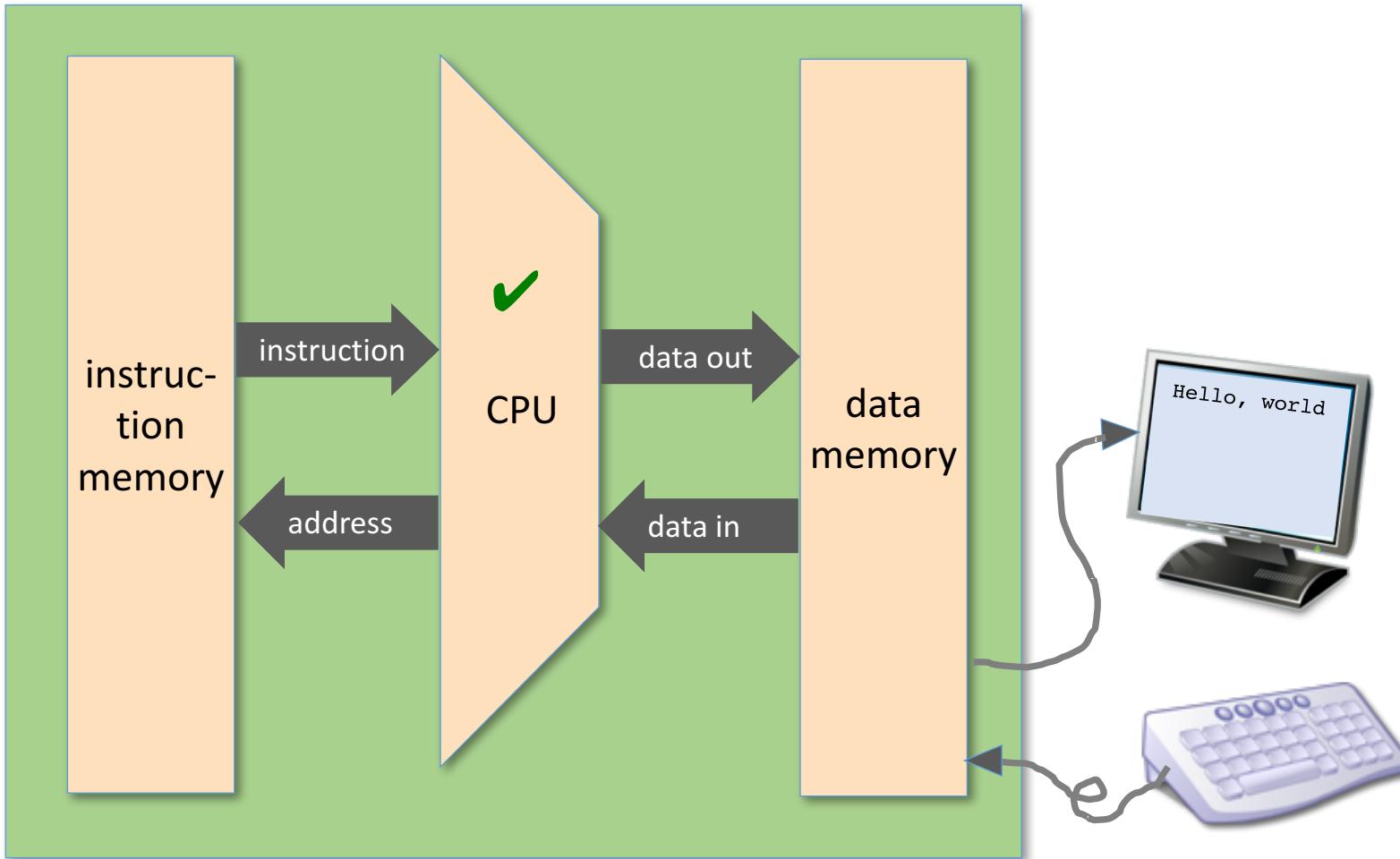
CPU abstraction:

Executes a Hack instruction and figures out which instruction to execute next

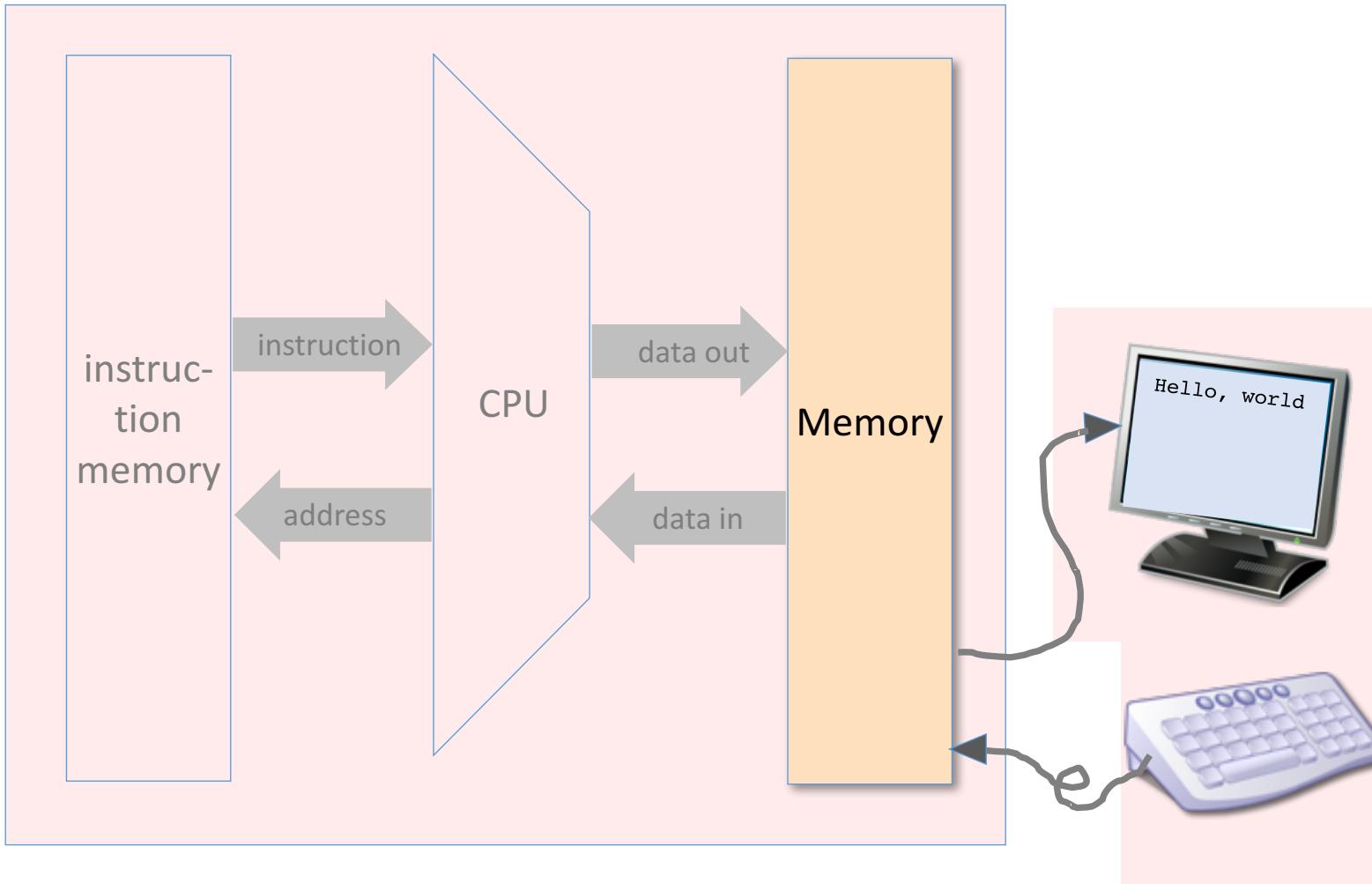
CPU Implementation:

Discussed before.

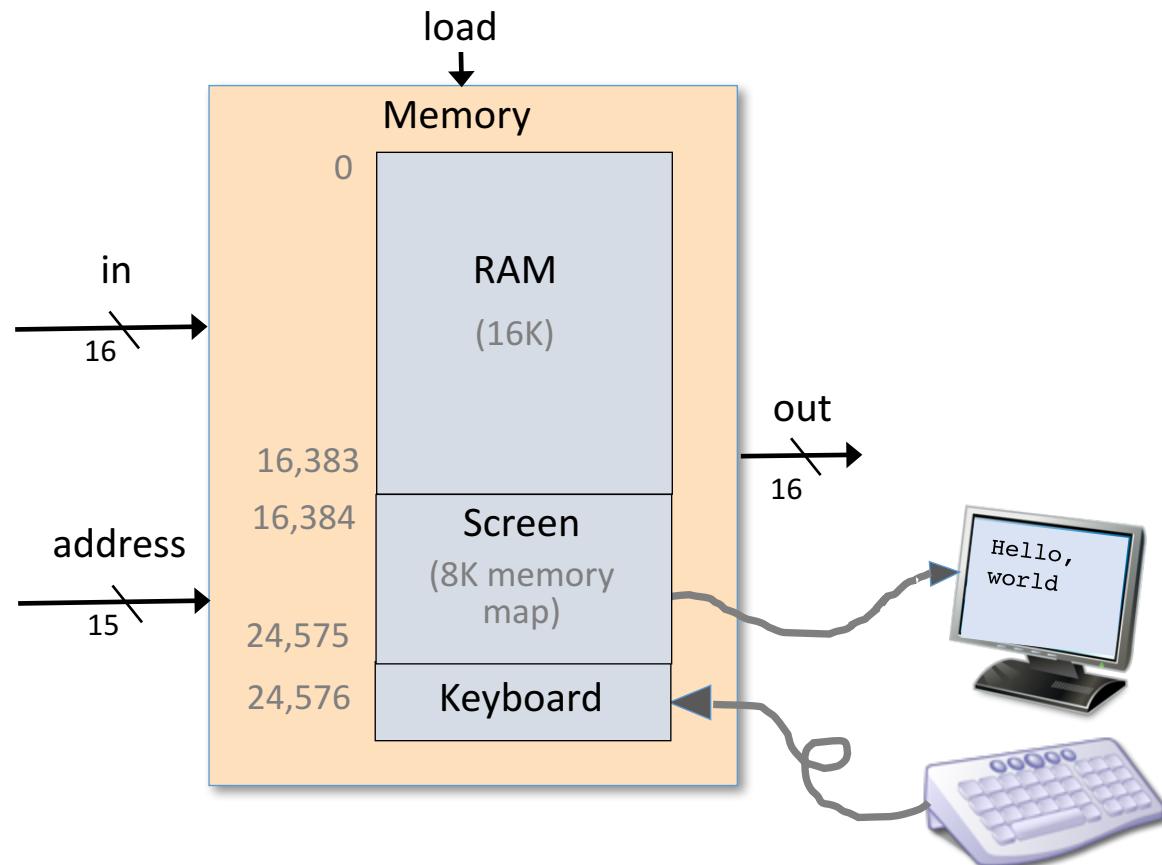
Hack Computer



Memory

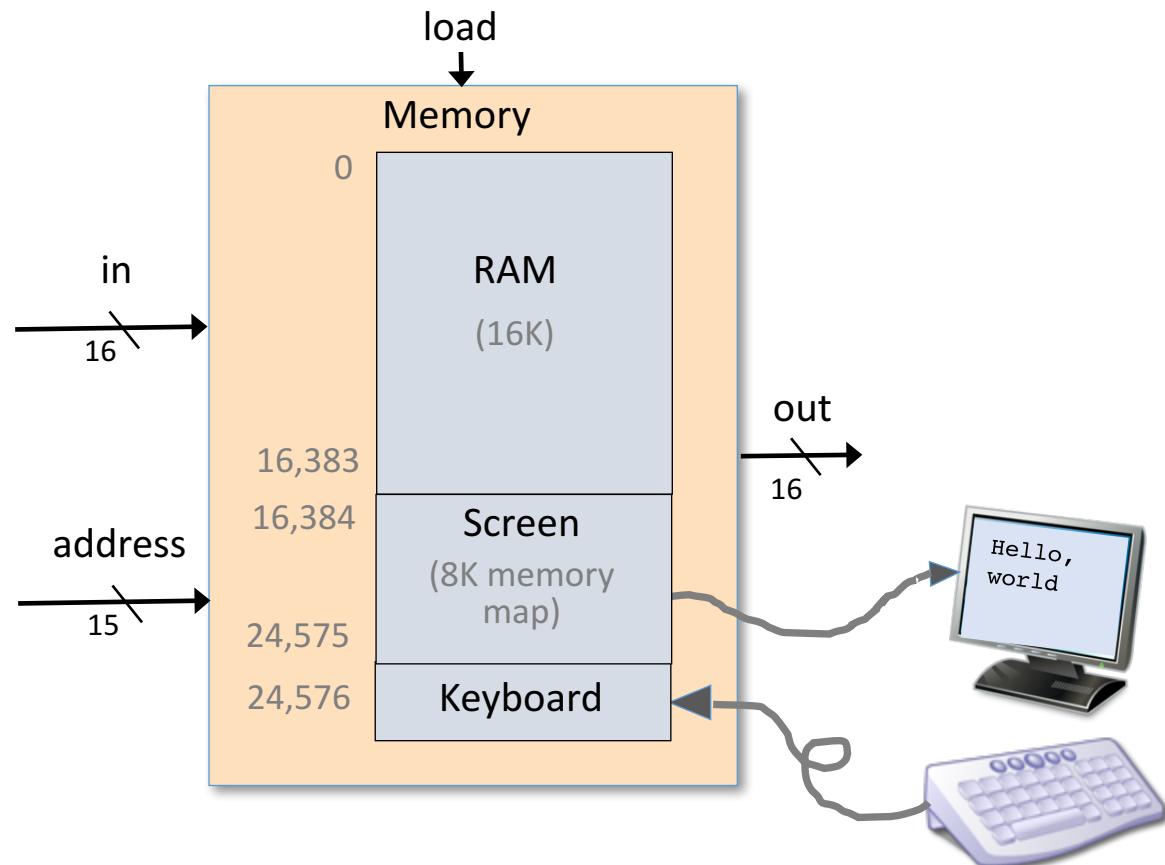


Memory: abstraction



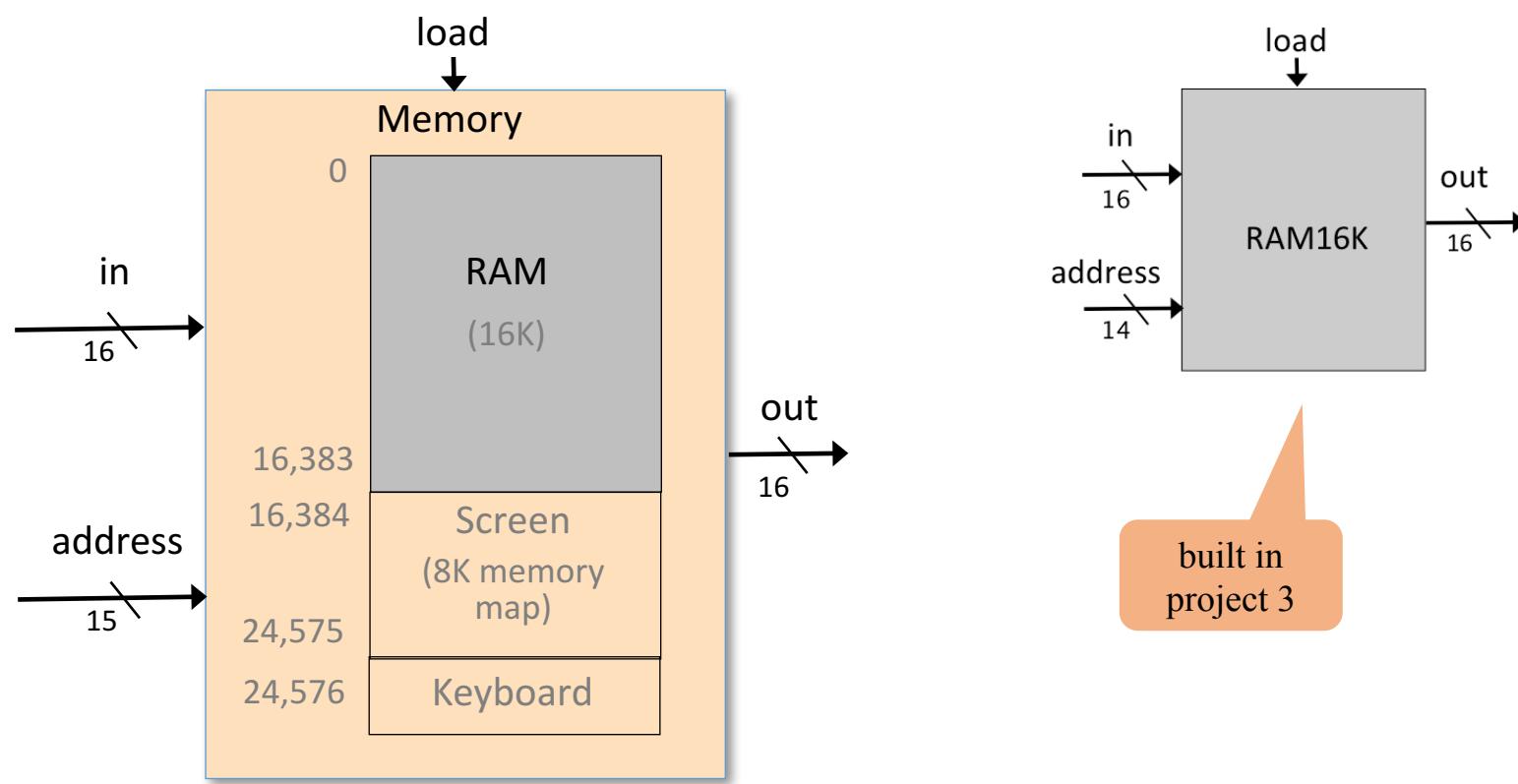
- Address 0 to 16383: data memory
- Address 16384 to 24575: screen memory map
- Address 24576: keyboard memory map

Memory: implementation



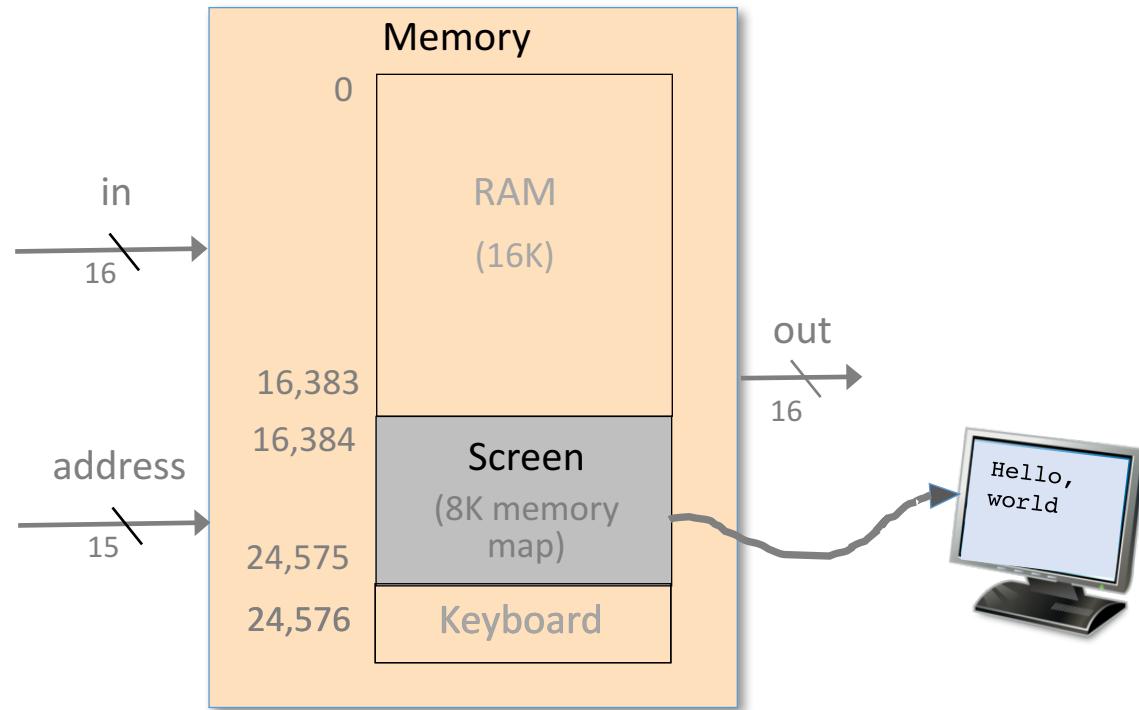
- Address 0 to 16383: data memory
- Address 16384 to 24575: screen memory map
- Address 24576: keyboard memory map

RAM

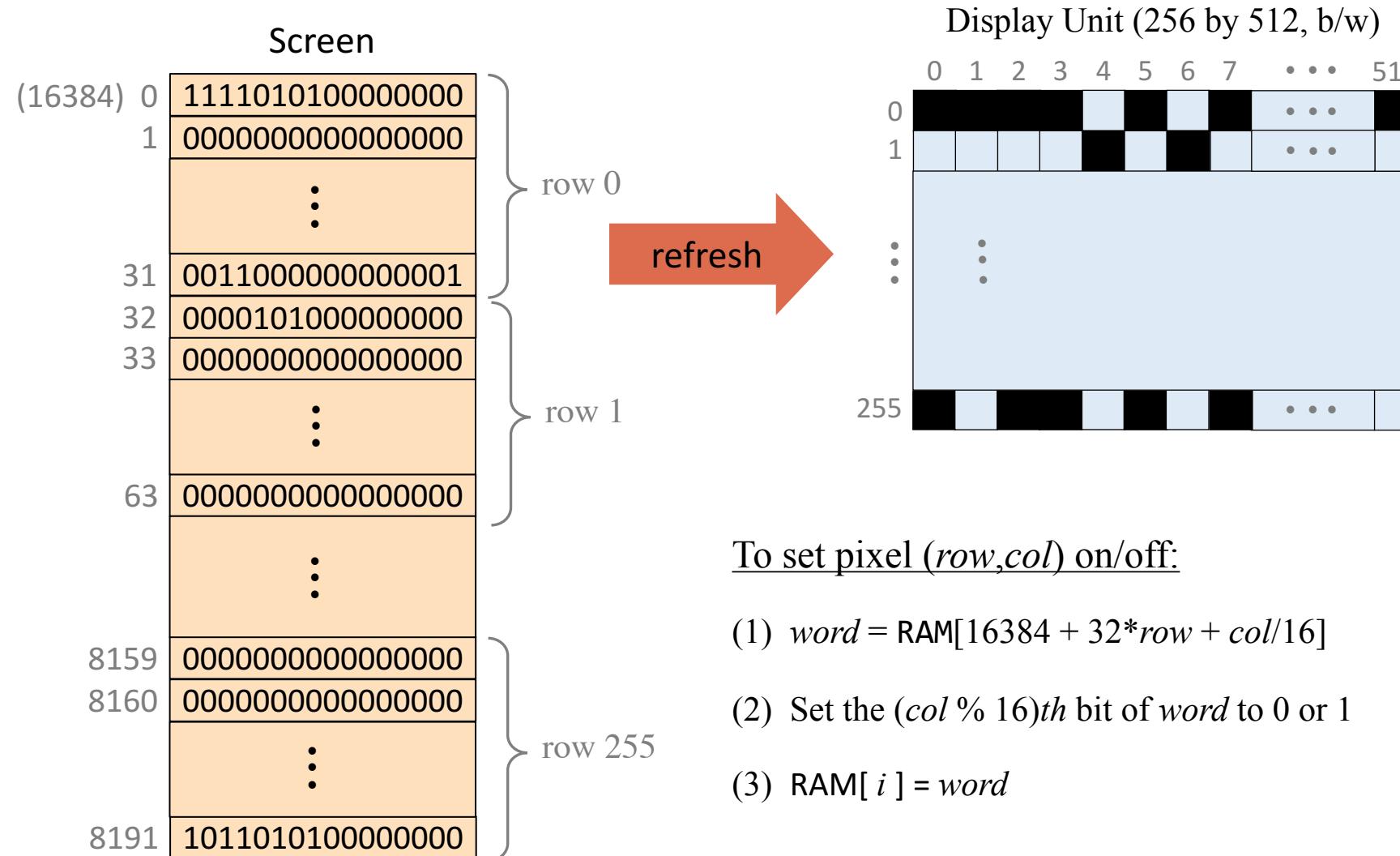


The Hack RAM is realized by the RAM16K chip implemented in project 3.

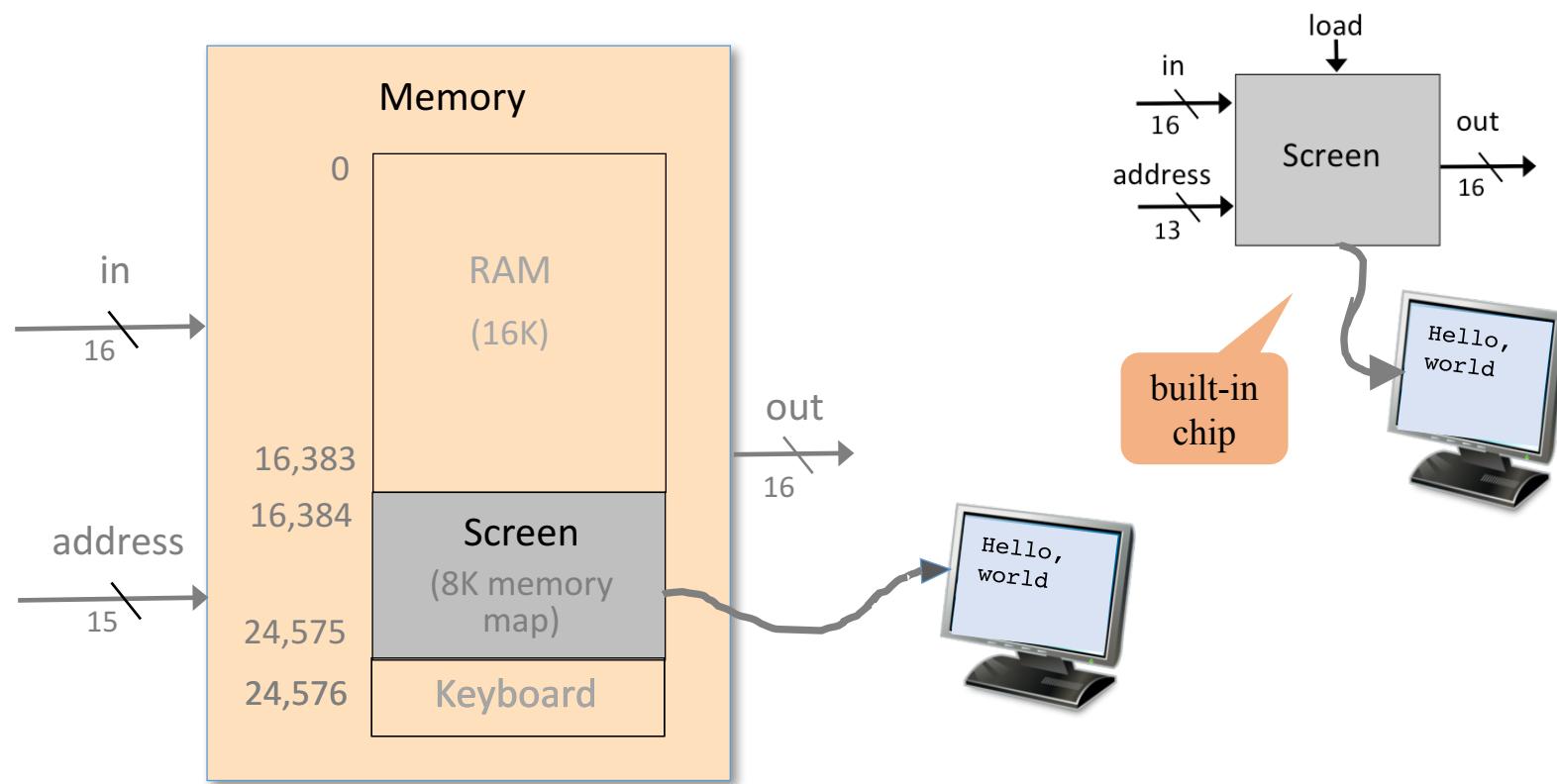
Screen



Screen memory map

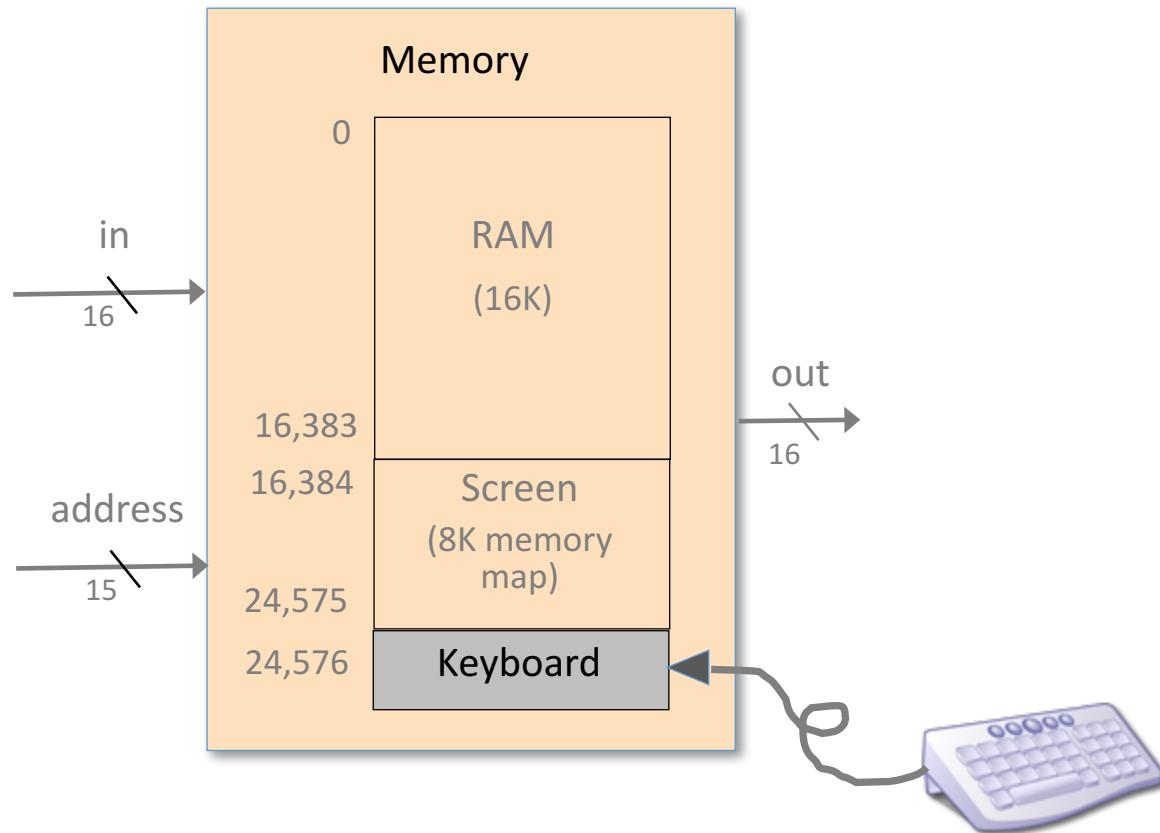


Screen

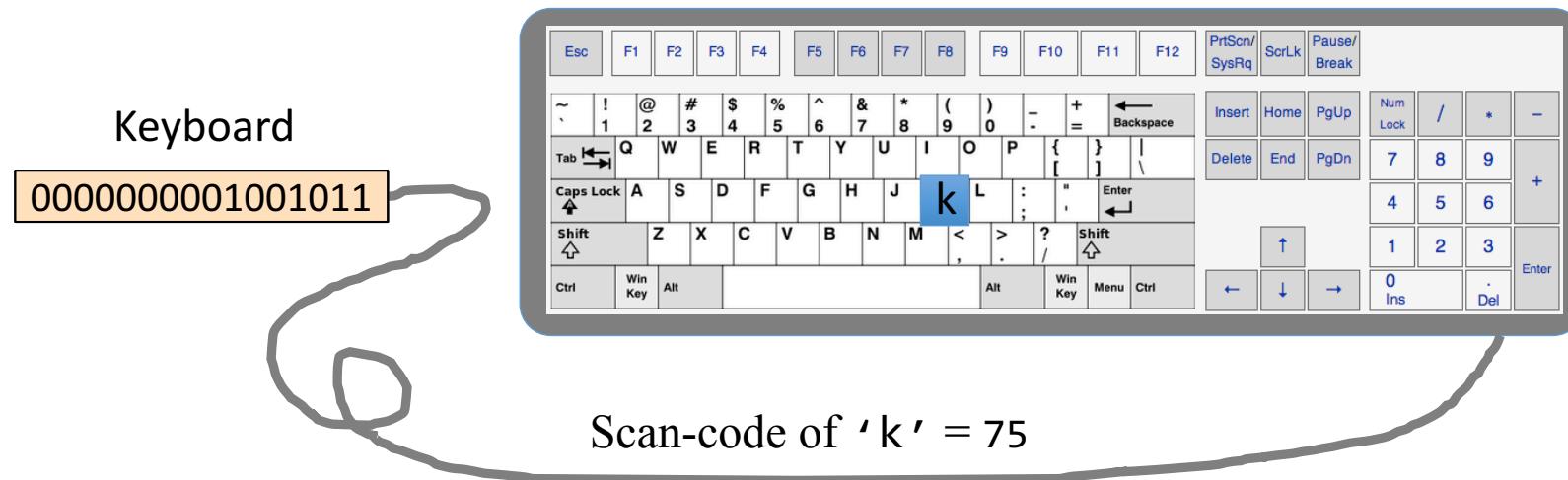


- The Hack screen is realized by a built-in chip named Screen
- Screen: a regular RAM + display output side-effect.

Keyboard



Keyboard memory map



The Keyboard chip emits the scan-code of the currently pressed key, or 0 if no key is pressed.

The Hack character set

| key | code |
|---------|------|
| (space) | 32 |
| ! | 33 |
| “ | 34 |
| # | 35 |
| \$ | 36 |
| % | 37 |
| & | 38 |
| ‘ | 39 |
| (| 40 |
|) | 41 |
| * | 42 |
| + | 43 |
| , | 44 |
| - | 45 |
| . | 46 |
| / | 47 |

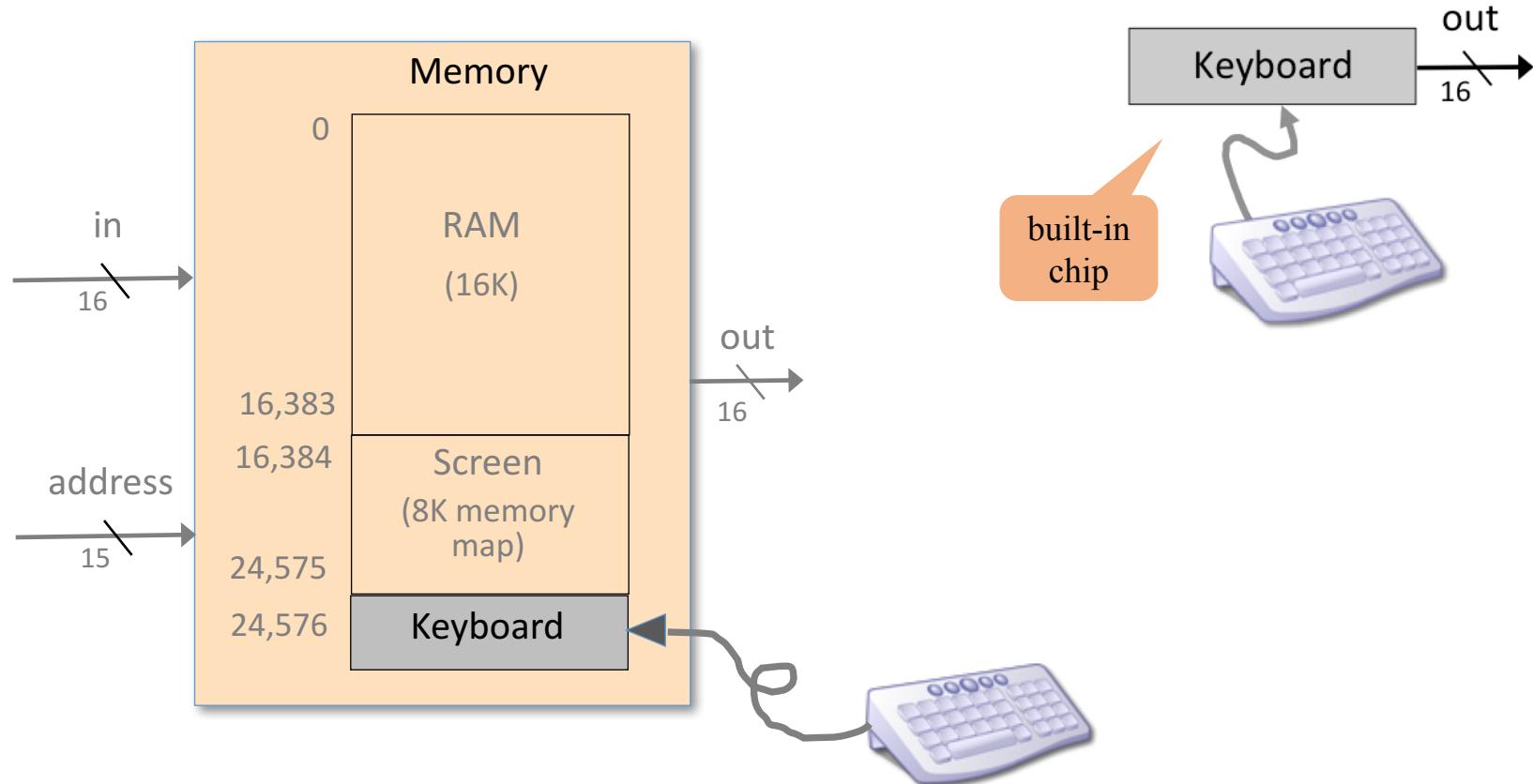
| key | code |
|-----|------|
| 0 | 48 |
| 1 | 49 |
| ... | ... |
| 9 | 57 |

| key | code |
|-----|------|
| A | 65 |
| B | 66 |
| C | ... |
| ... | ... |
| Z | 90 |

| key | code |
|-----|------|
| a | 97 |
| b | 98 |
| c | 99 |
| ... | ... |
| z | 122 |

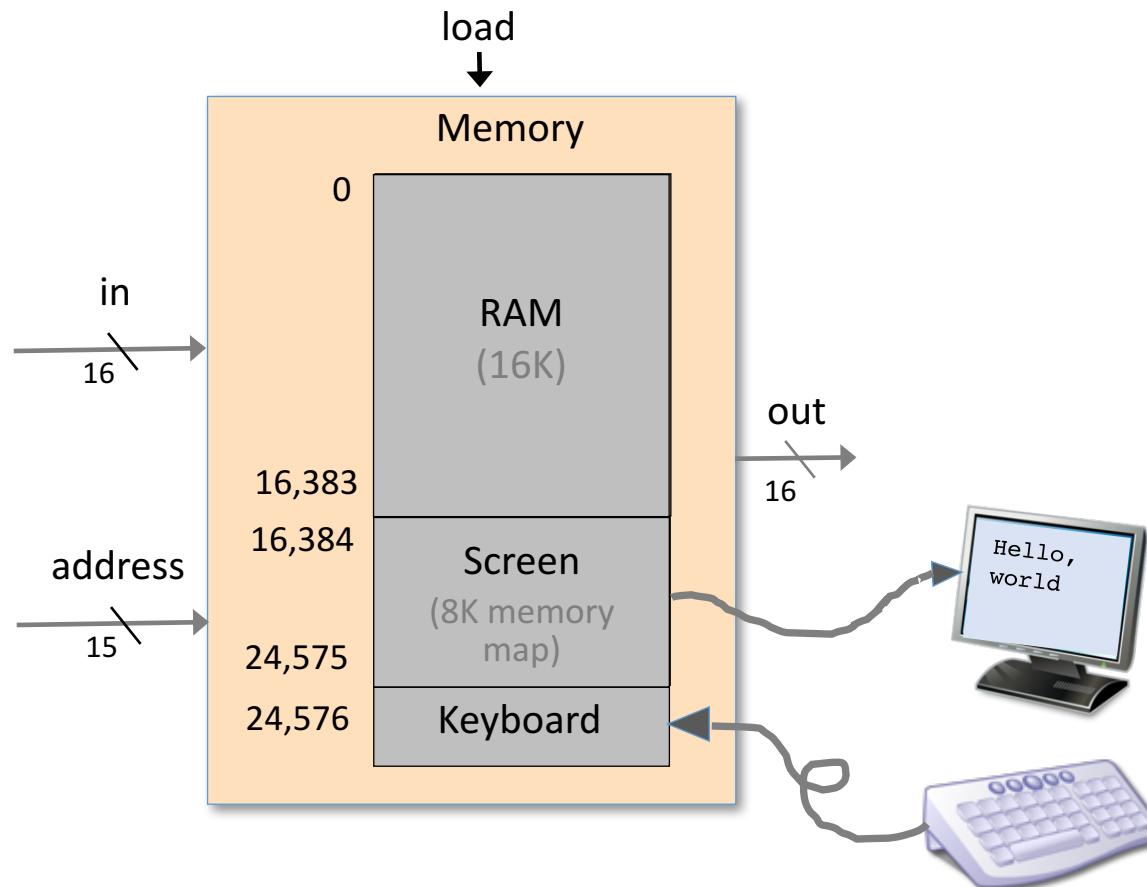
| key | code |
|-------------|------|
| newline | 128 |
| backspace | 129 |
| left arrow | 130 |
| up arrow | 131 |
| right arrow | 132 |
| down arrow | 133 |
| home | 134 |
| end | 135 |
| Page up | 136 |
| Page down | 137 |
| insert | 138 |
| delete | 139 |
| esc | 140 |
| f1 | 141 |
| ... | ... |
| f12 | 152 |

Keyboard



- Realized by a built-in chip named **Keyboard**
- **Keyboard:** A read-only 16-bit register + a keyboard input side-effect.

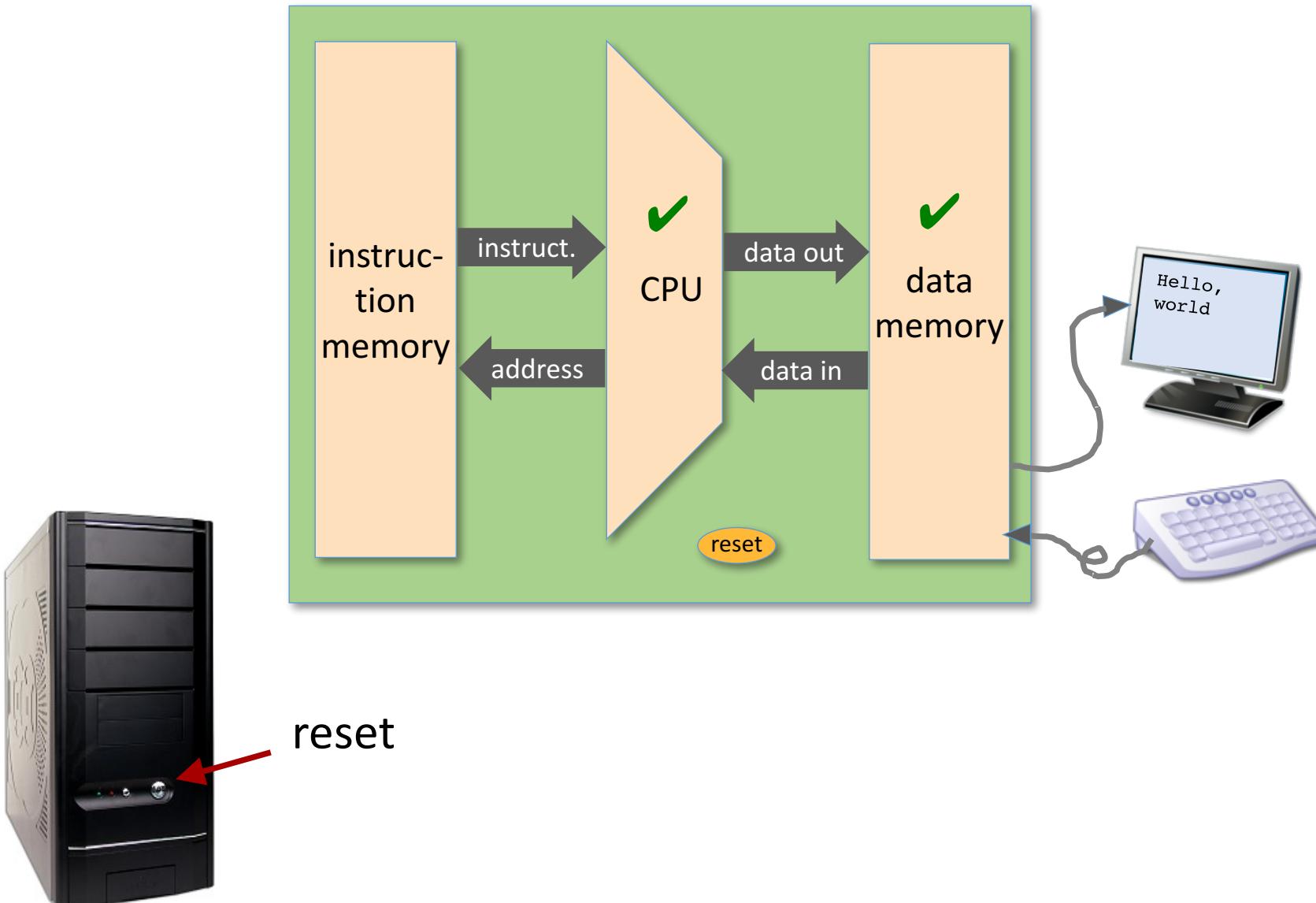
Memory implementation



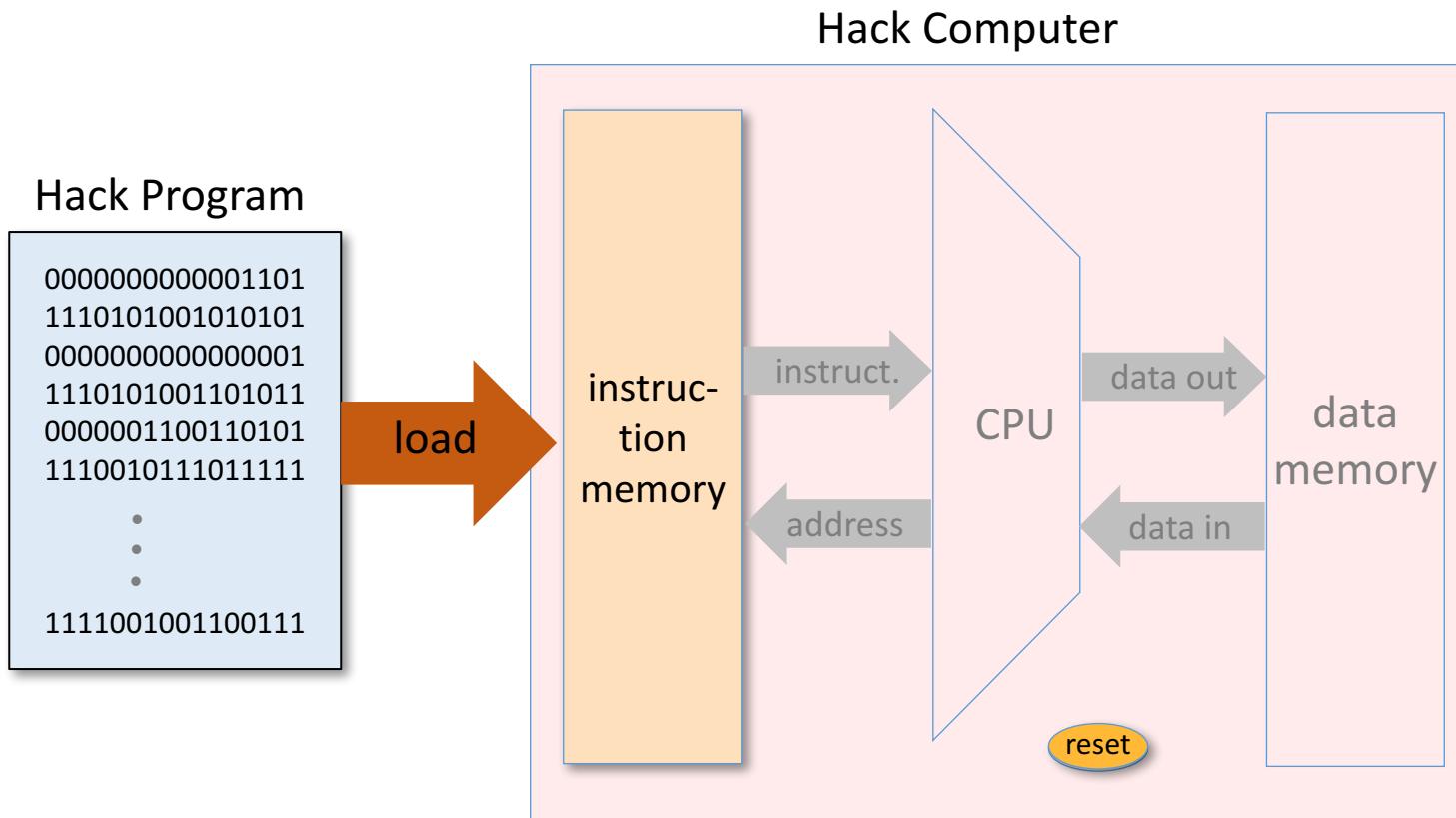
Implementation outline:

- Uses the three chip-parts RAM16K, Screen, and Keyboard (as just described)
- Routes the address input to the correct address input of the relevant chip-part.

Hack Computer



Instruction memory



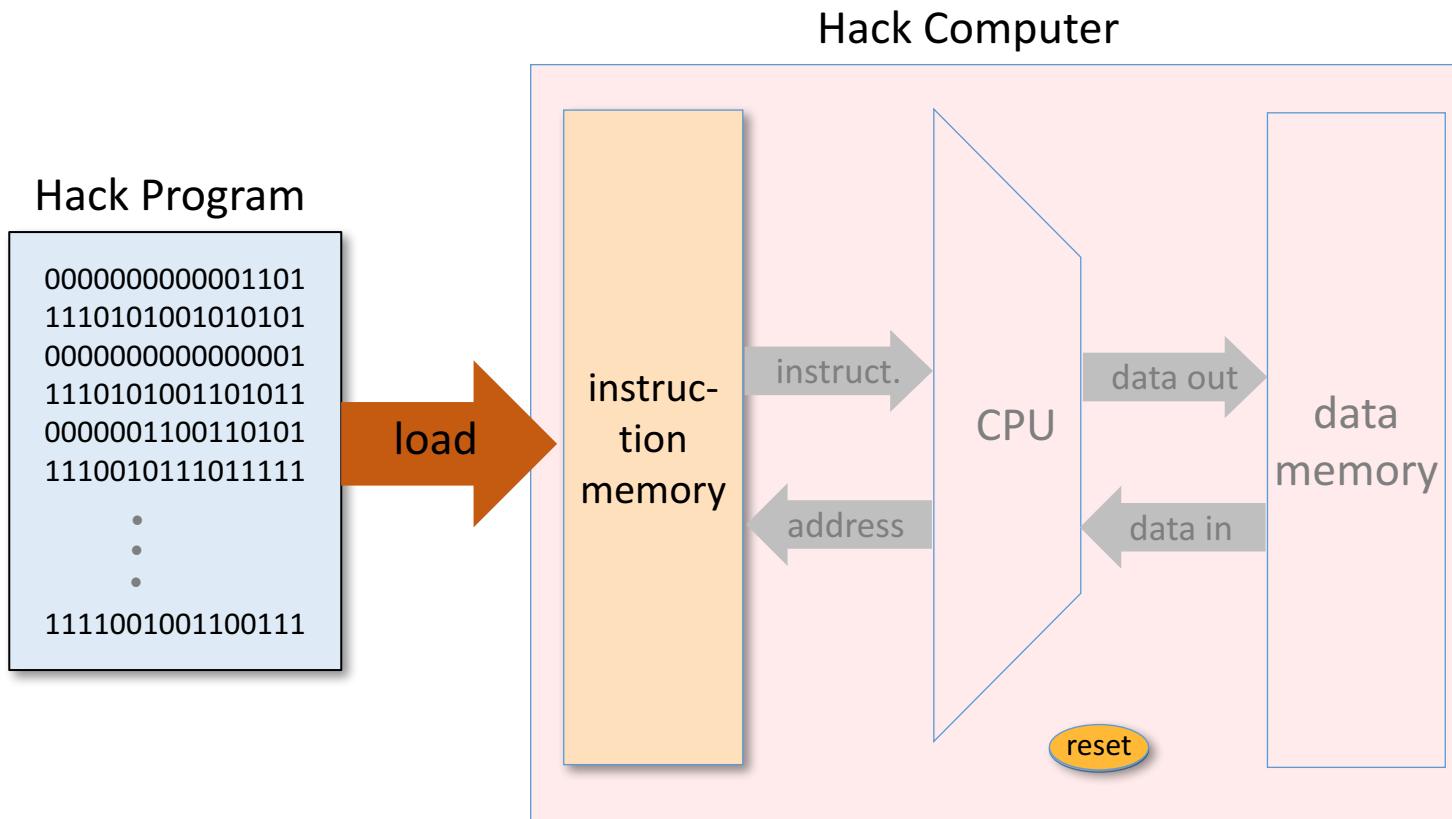
To run a program on the Hack computer:

- ❑ Load the program into the Instruction Memory
- ❑ Press “reset”
- ❑ The program starts running.



Load a program
into the Instruction
Memory? How?

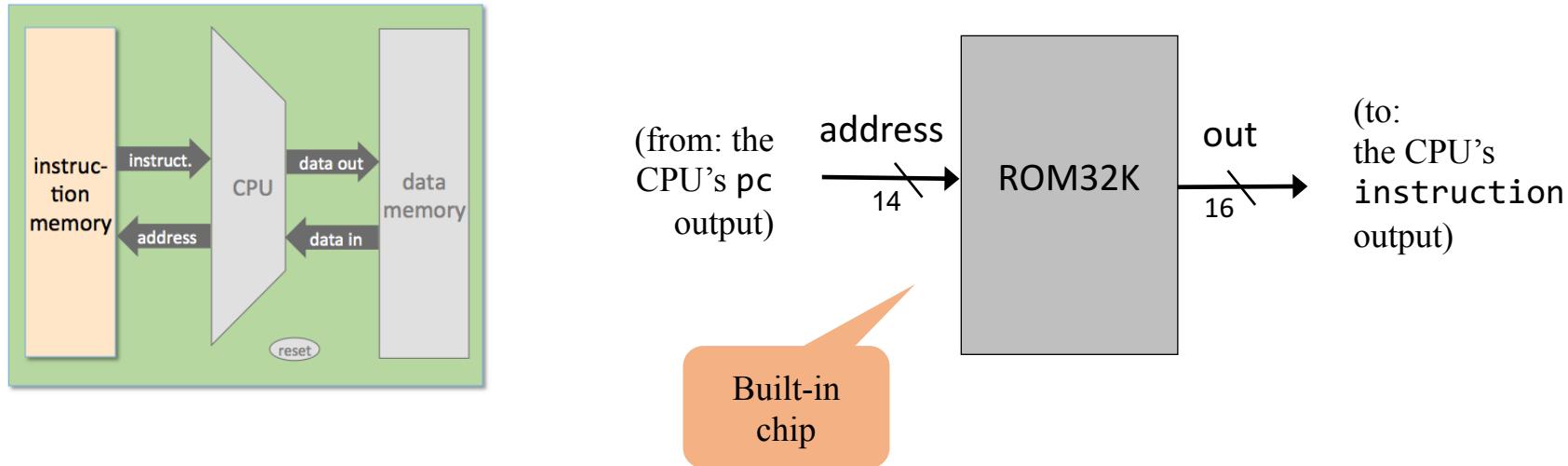
Instruction memory



Loading a program into the Instruction Memory:

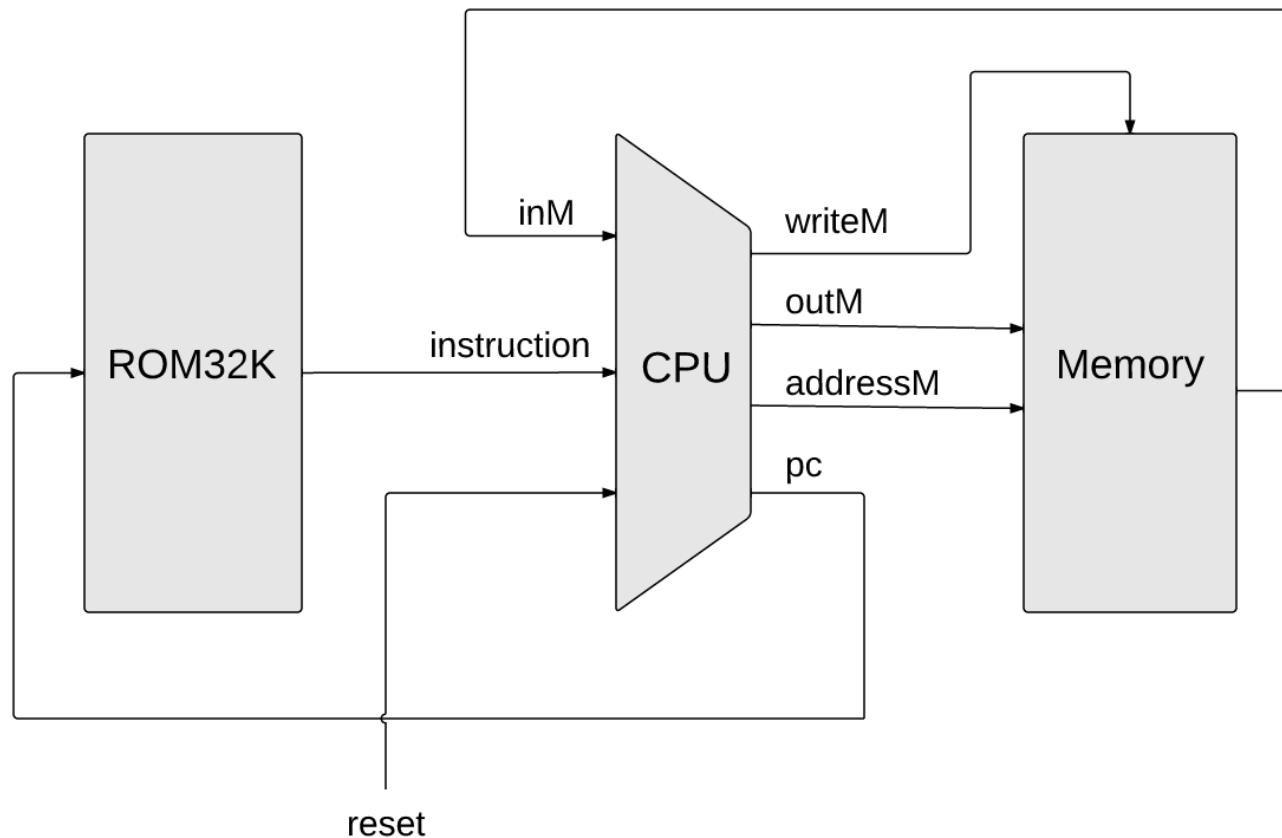
- Hardware implementation: plug-and-play ROM chips
(each comes pre-loaded with a program's code)
- Hardware simulation: programs are stored in text files;
The simulator's software features a load-program service.

Instruction memory

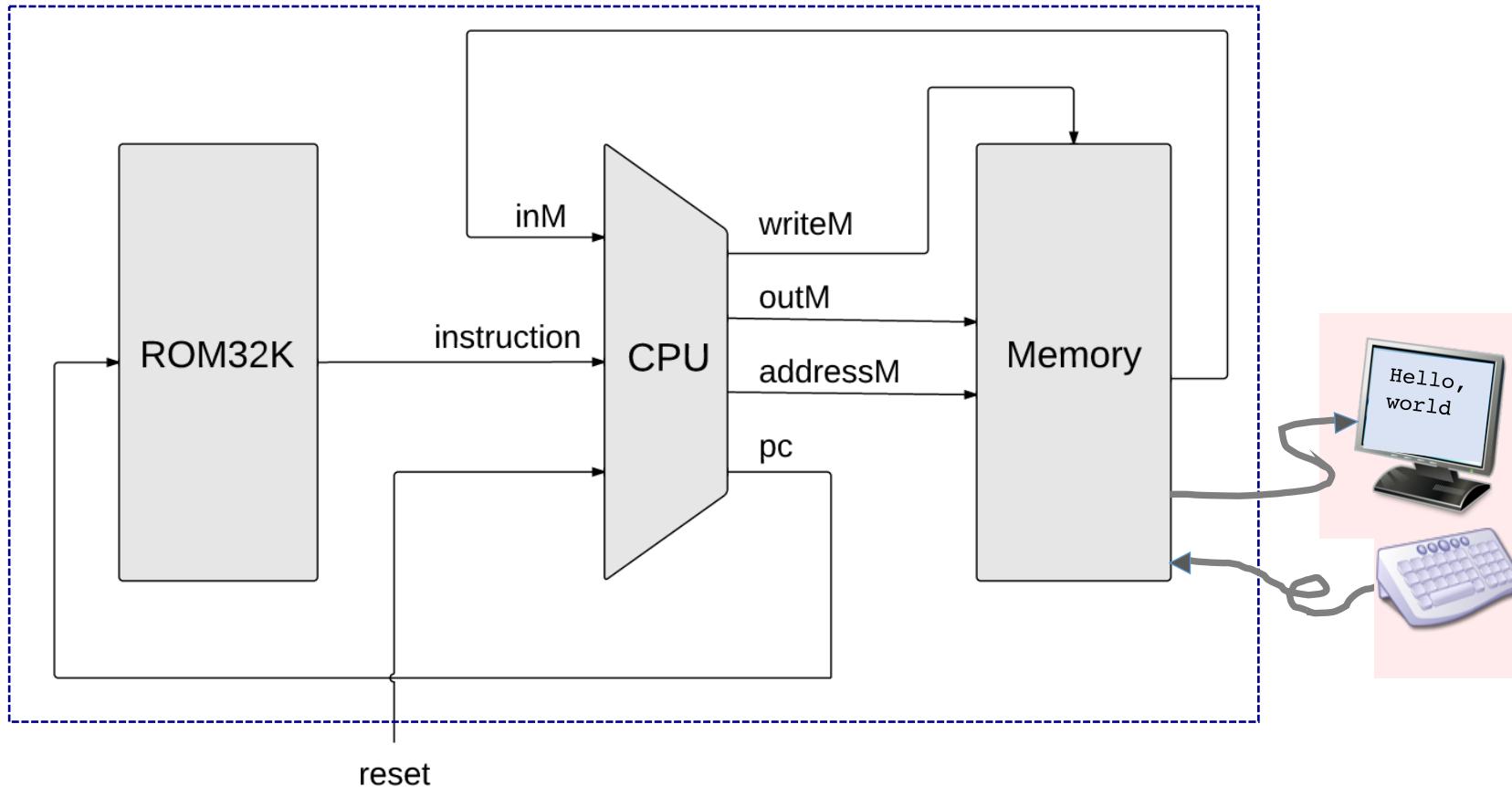


- The Hack Instruction Memory is realized by a built-in chip named ROM32K
- ROM32K: a read-only, 16-bit, 32K RAM chip + program loading side-effect.

Hack Computer implementation

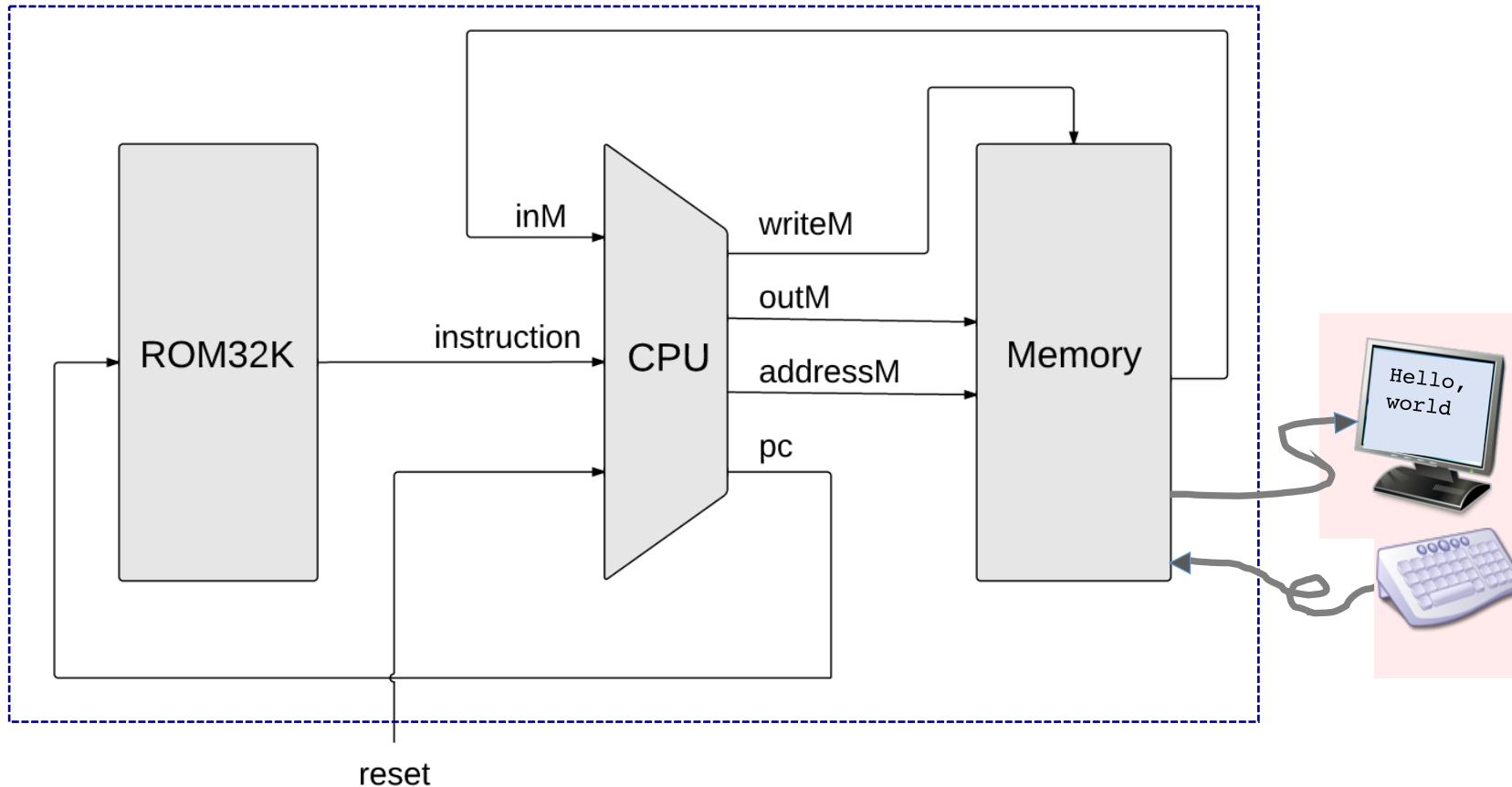


Hack Computer implementation



That's it!

Hack Computer implementation



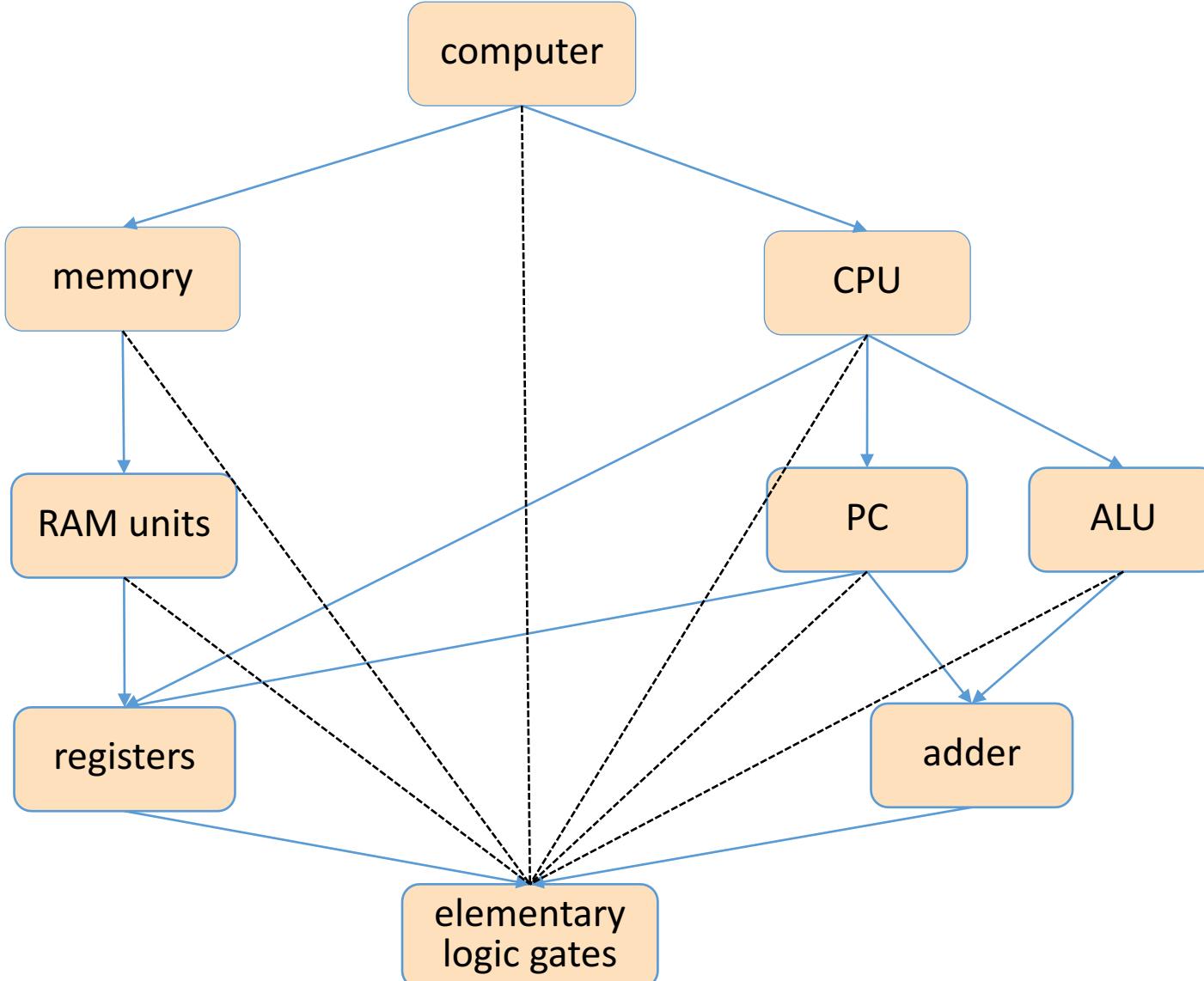
“We ascribe beauty to that which is simple; which has no superfluous parts; which exactly answers its end; which stands related to all things; which is the mean of many extremes.”

-- Ralph Waldo Emerson

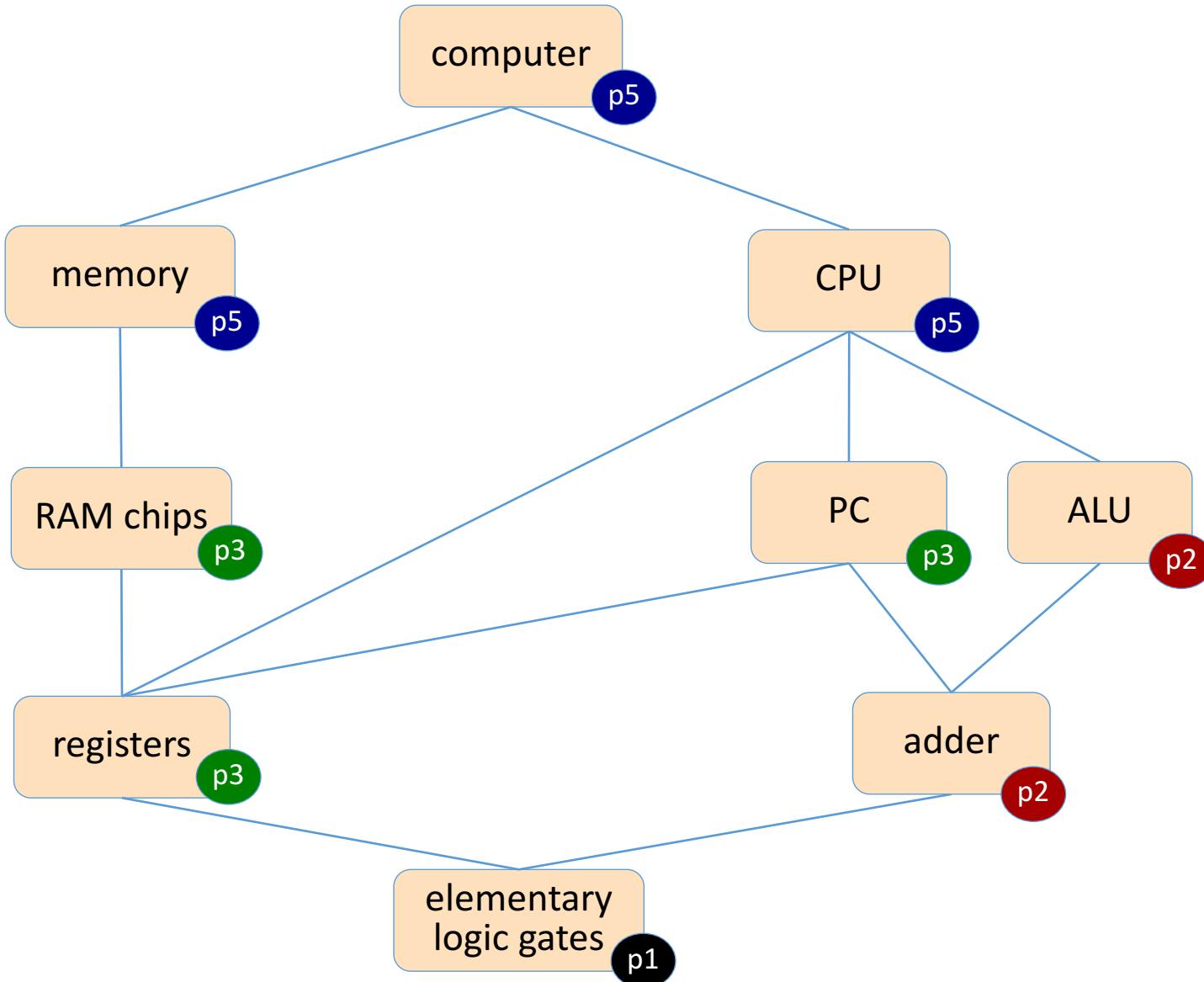
Computer Architecture: lecture plan

- ✓ Von Neumann Architecture
- ✓ Fetch-Execute Cycle
- ✓ The Hack CPU
- ✓ The Hack Computer
- Project 5 Overview

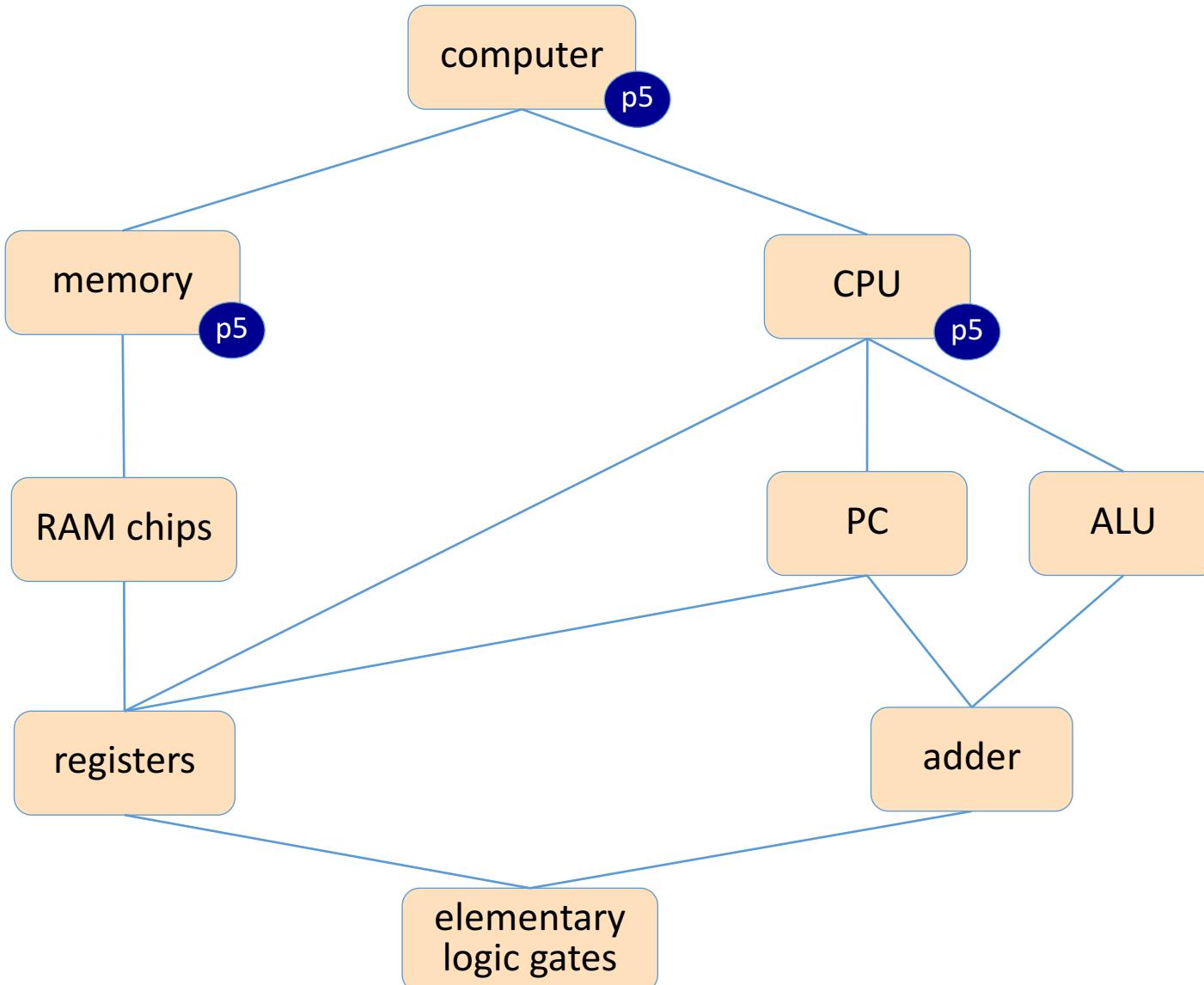
Hardware organization: a hierarchy of chip parts



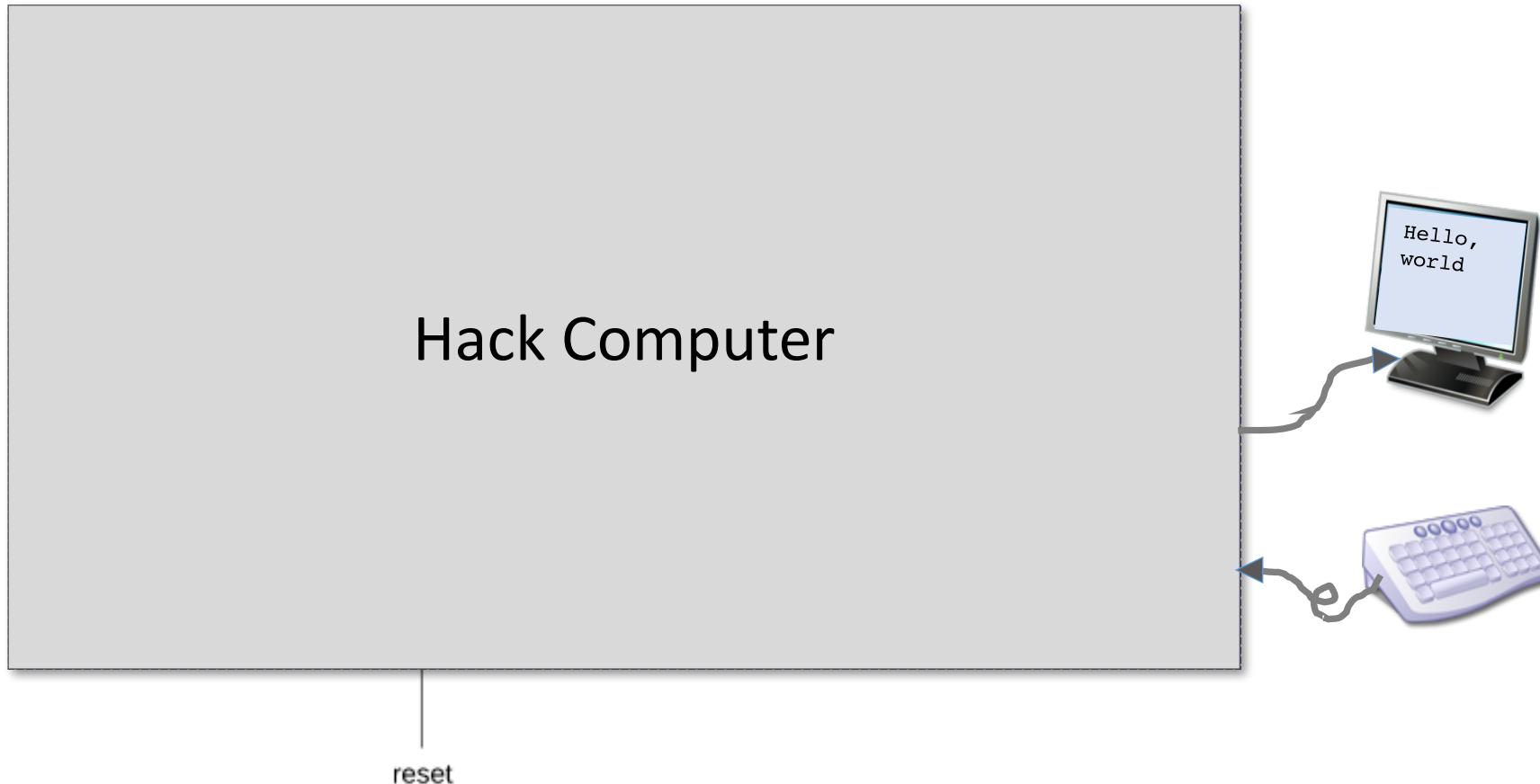
Hardware projects



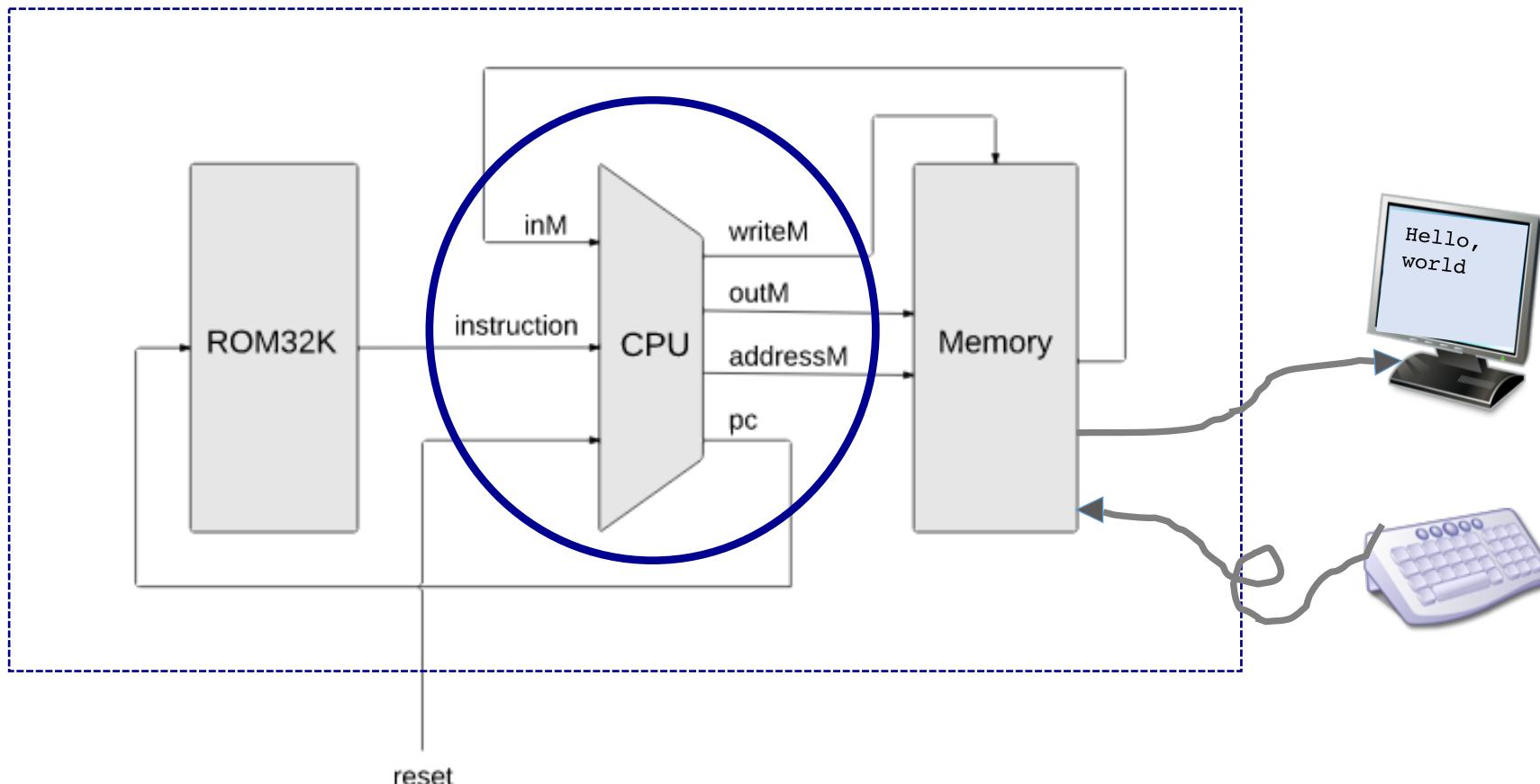
Project 5: building the Hack Computer



Abstraction



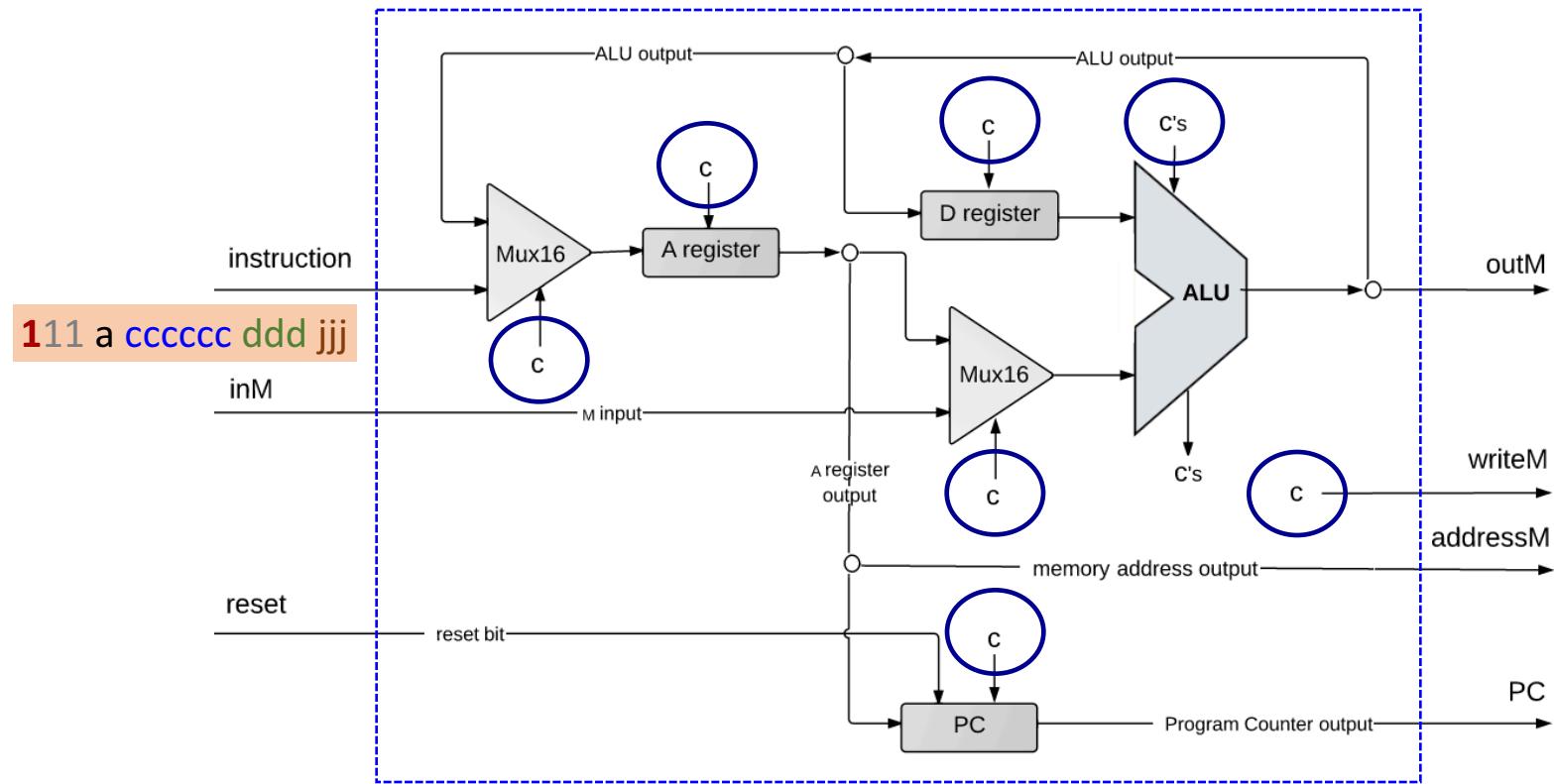
Implementation



CPU Abstraction



CPU Implementation



Implementation tips:

- Chip-parts: **Mux16**, **AResister**, **DRegister**, **PC**, **ALU**, ...
- Control: use HDL subscripting to parse and route the instruction bits to the control bits of the relevant chip-parts.

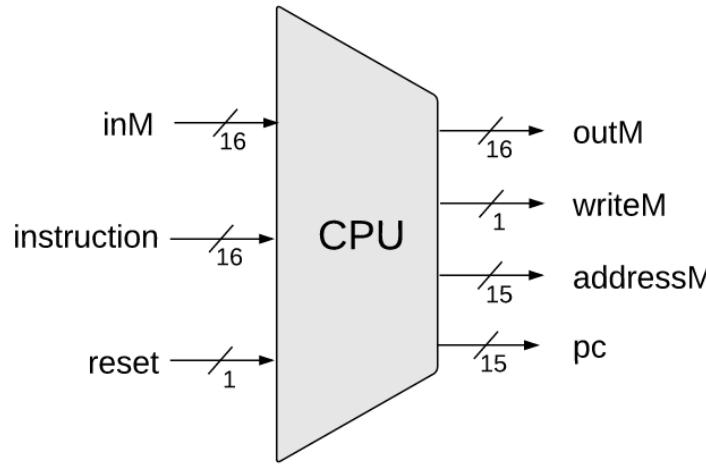
CPU Implementation

CPU.hdl

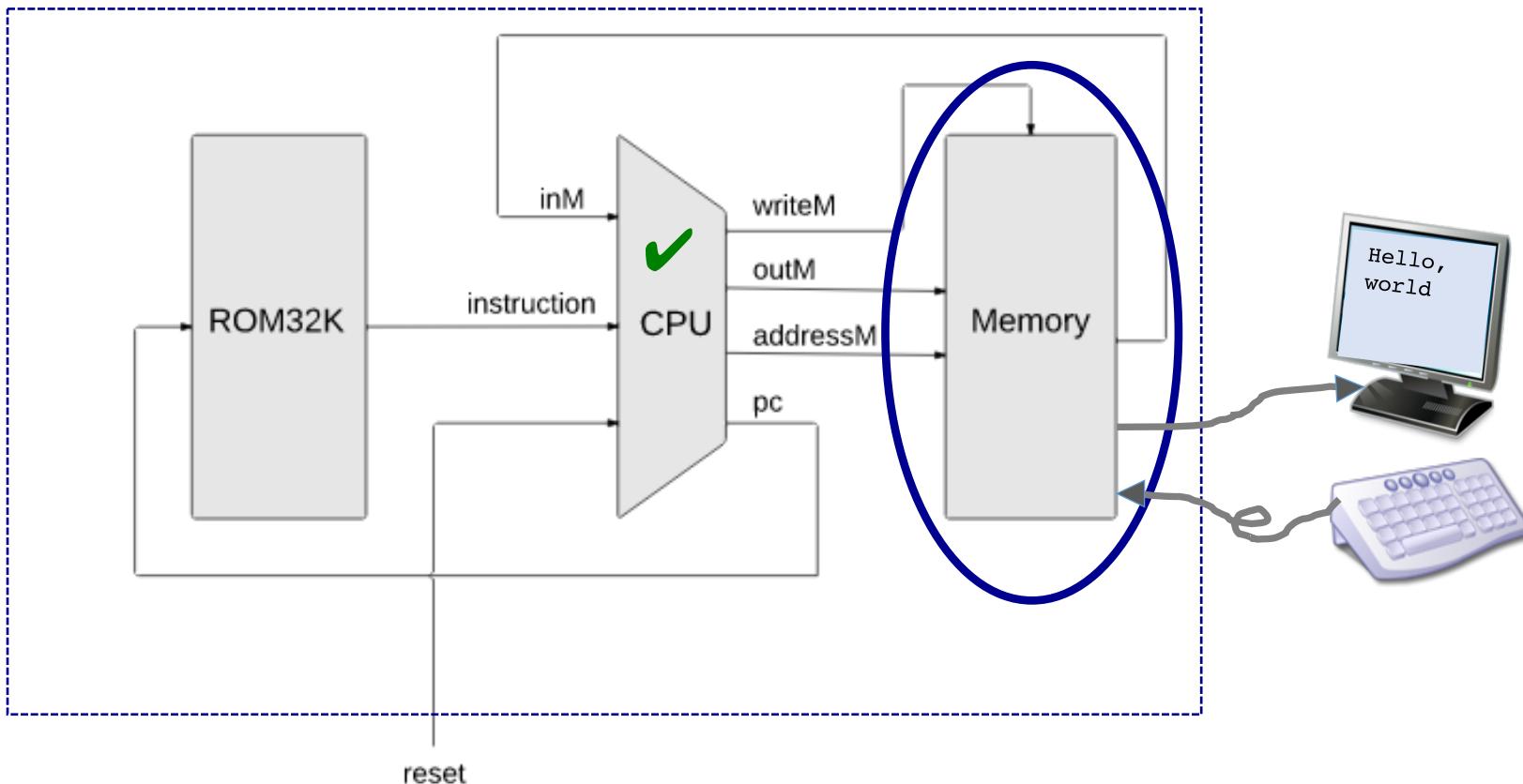
```
/*
 * The Central Processing unit (CPU).
 * Consists of an ALU and a set of registers,
 * designed to fetch and execute instructions
 * written in the Hack machine language.
 */
CHIP CPU {
    IN
        inM[16],          // value of M = RAM[A]
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current program
                        // (reset == 1) or continue executing the current
                        // program (reset == 0).

    OUT
        outM[16]          // value to write into M = RAM[A]
        writeM,           // Write into M?
        addressM[15],    // RAM address (of M)
        pc[15];          // ROM address (of next instruction)

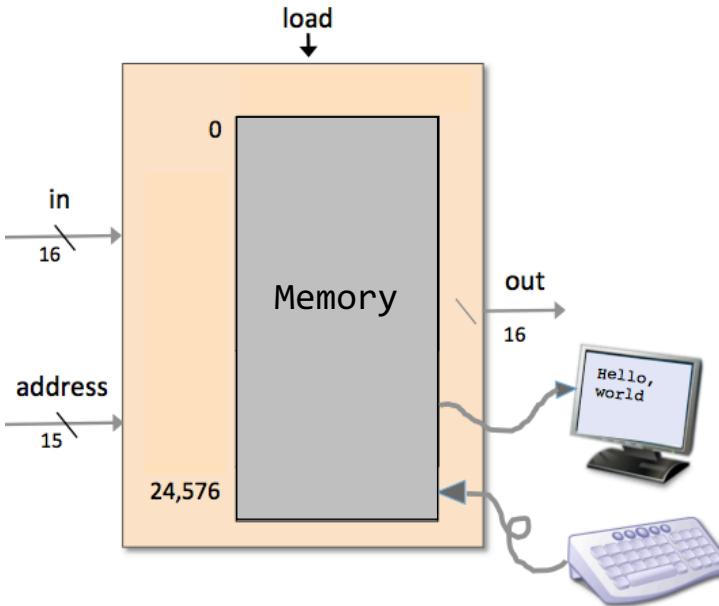
    PARTS:
        // Put your code here:
}
```



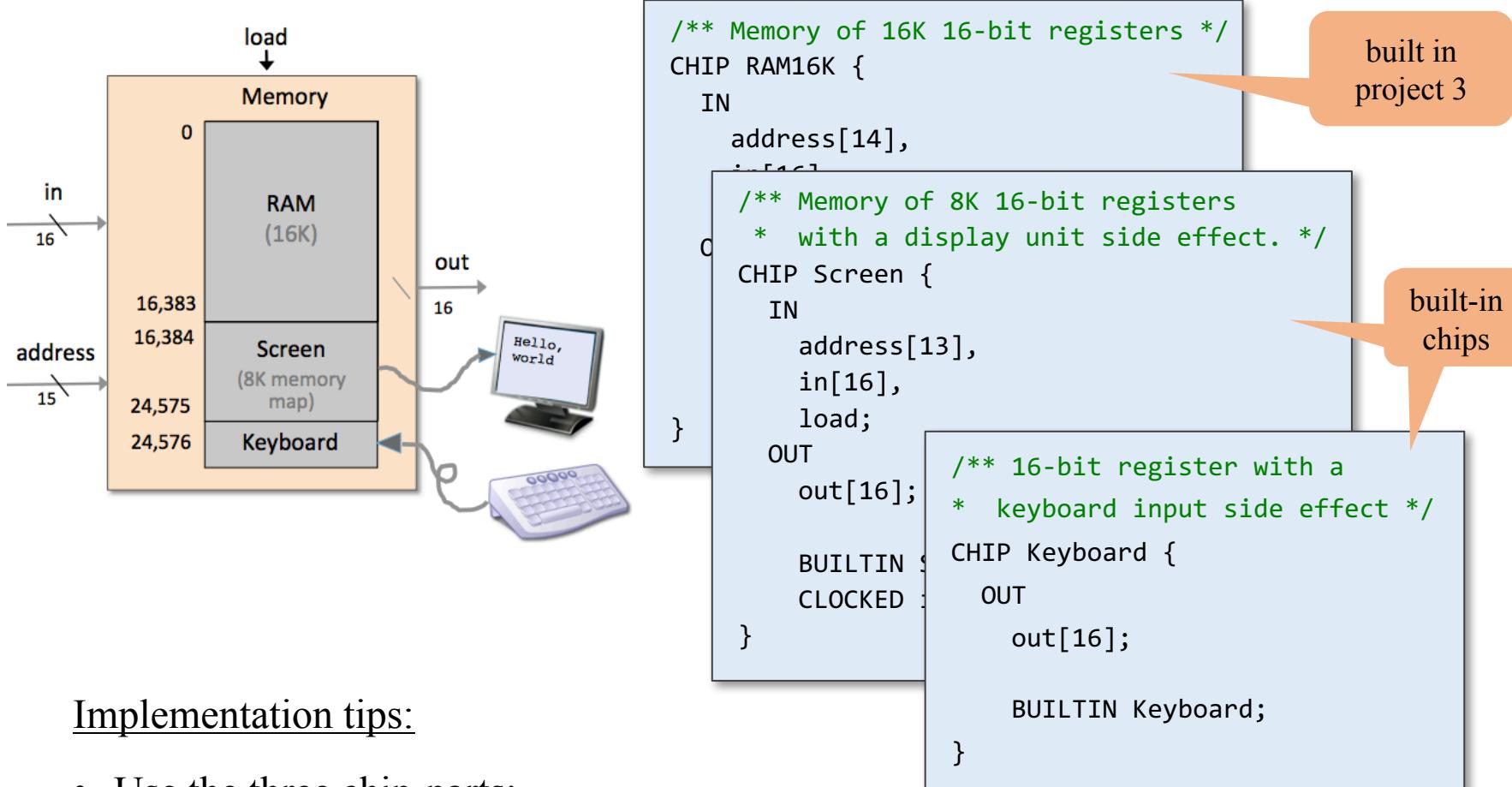
Hack Computer implementation



Memory implementation



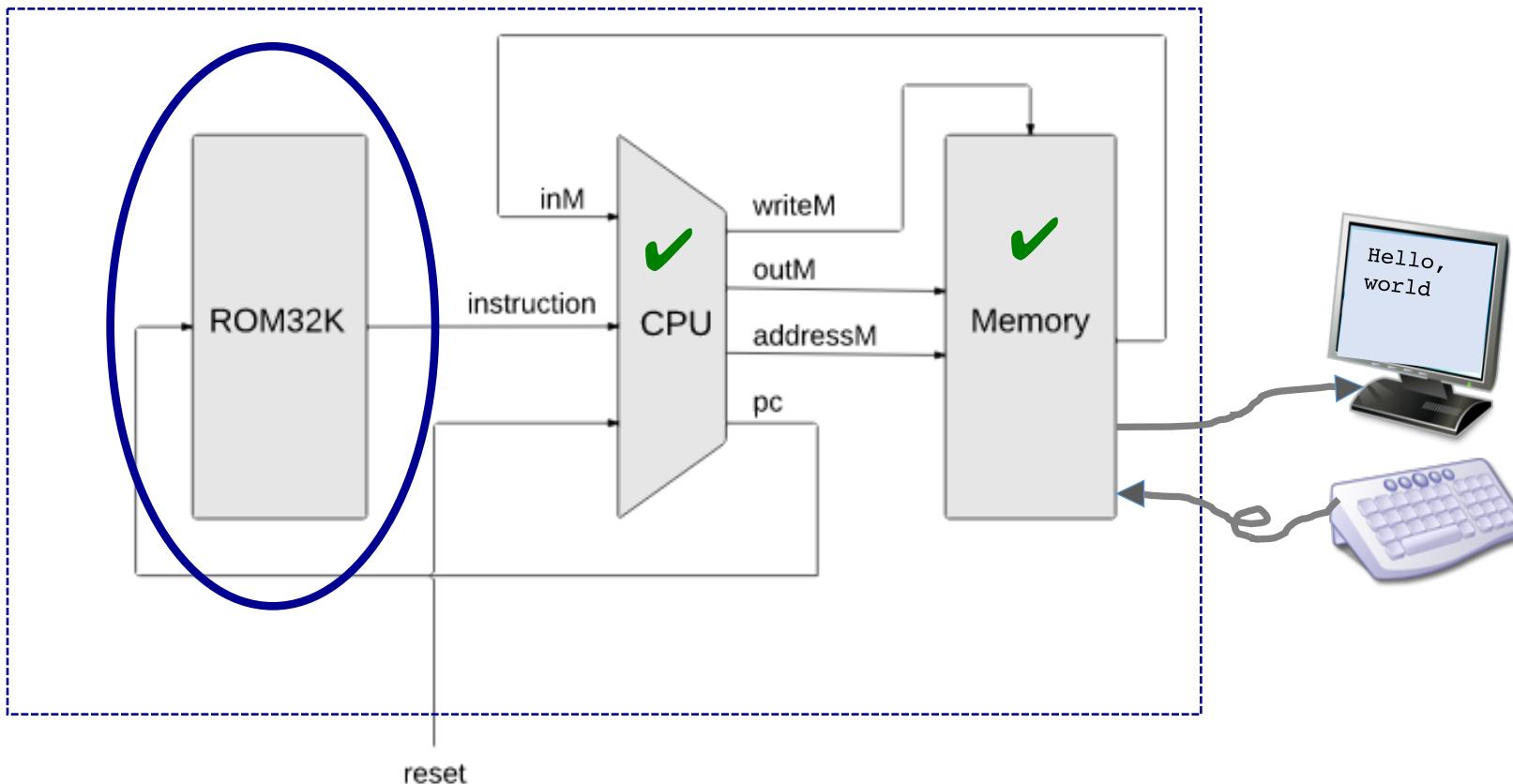
Memory implementation



Implementation tips:

- Use the three chip-parts:
RAM16K, Screen, and Keyboard
- Route the address input to the correct address input of the relevant chip-part.

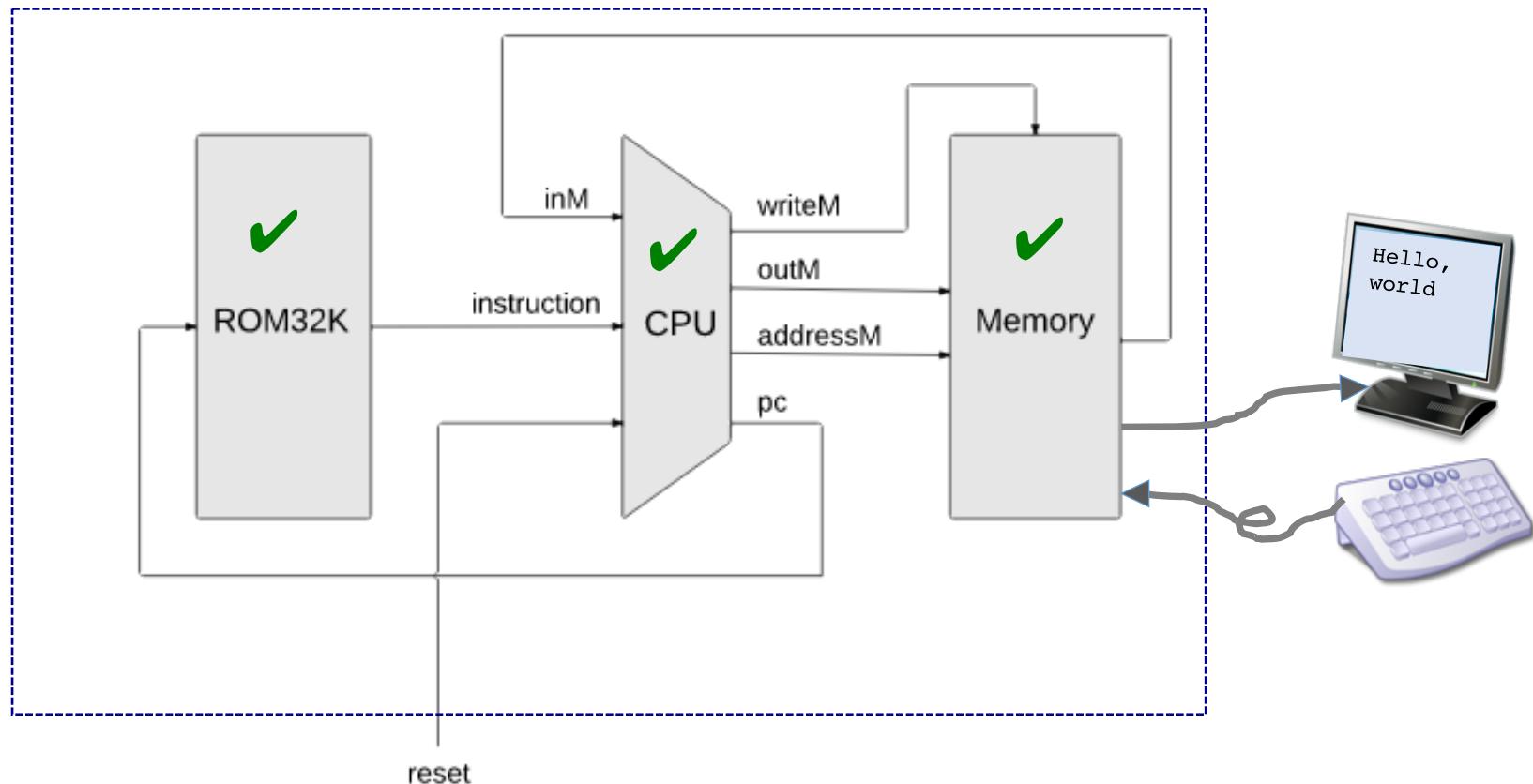
Hack Computer implementation



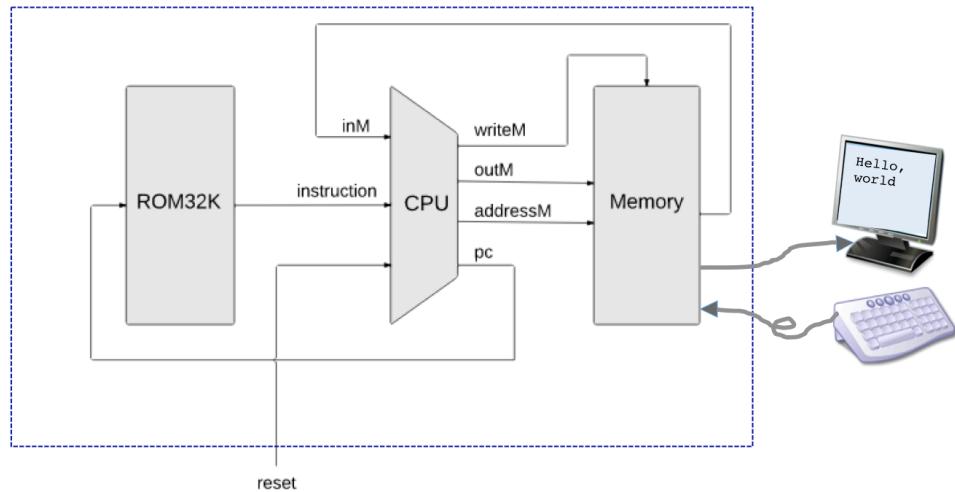
Implementation tip:

Use the built-in **ROM32K** chip.

Hack Computer implementation



Hack Computer implementation



Computer.hdl

```
/**  
 * The HACK computer, including CPU, ROM and RAM.  
 * When reset is 0, the program stored in the computer's ROM executes.  
 * When reset is 1, the execution of the program restarts.  
 */  
  
CHIP Computer {  
  
    IN reset;  
  
    PARTS:  
        // Put your code here.  
}
```

Project 5 resources

From NAND to Tetris
Building a Modern Computer From First Principles

www.nand2tetris.org



Home
Prerequisites
Syllabus

Course

Book
Software
Terms
Papers
Talks
Cool Stuff
About
Team
Q&A

Project 5: Computer Architecture

Background

In previous projects we've built the computer's basic *processing* and *storage* devices (*ALU* and *RAM*, respectively). In this project we will put everything together, yielding the complete *Hack Hardware Platform*. The result will be a general-purpose computer that can run any program that you fancy.

Objective

Complete the construction of the Hack CPU and computer platform, leading up to the top-most Computer chip.

Chips

| Chip (HDL) | Description | Testing |
|------------------------------|------------------------------|--|
| Memory.hdl | Entire RAM address space | Test this chip using Memory.tst and Memory.cmp |
| CPU.hdl | The Hack CPU | Recommended test files: CPU.tst and CPU.cmp . Alternative test files (less thorough but do not require using the built-in DRegister): CPU-external.tst and CPU-external.cmp . |
| Computer.hdl | The platform's top-most chip | Test by running some Hack programs on the constructed chip. See more instructions below. |

All the necessary project 5 files are available in:
[nand2tetris / projects / 05](#)

More resources

- HDL Survival Guide
- Hardware Simulator Tutorial
- nand2tetris Q&A forum



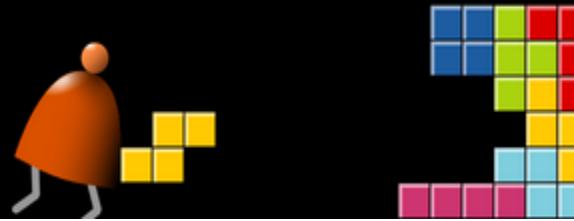
All available in: www.nand2tetris.org

Best practice advice

- Try to implement the chips in the given order
- Strive to use as few chip-parts as possible
- You will have to use chips that you've implemented in previous projects
- The best practice is to use their built-in versions.

Computer Architecture: lecture plan

- ✓ Von Neumann Architecture
- ✓ Fetch-Execute Cycle
- ✓ The Hack CPU
- ✓ The Hack Computer
- ✓ Project 5 Overview



Chapter 5

Computer Architecture

These slides support chapter 5 of the book

The Elements of Computing Systems

By Noam Nisan and Shimon Schocken

MIT Press