```
a ─────┬──── [Not] ──── notb ──┐
       │                        ├── [And] ── aAndNotb ──┐
       │                                                  ├── [Or] ── out
       │                        ┌── [And] ── notaAndb ──┘
b ─────┴──── [Not] ──── nota ──┘
```

```
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    Not(in=a, out=nota);
    Not(in=b, out=notb);
    And(a=nota, b=b, out=w1);
    And(a=a, b=notb, out=w2);
    Or(a=w1, b=w2, out=out);
}
```

Memory

program

data

Memory output

Memory address input

instruction register

instruction

instruction

ALU

data

load on fetch

fetch / execute bit

mux

instruction address

data address

control bus

address bus

address bus   (data flows not shown, to minimize clutter)

```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```
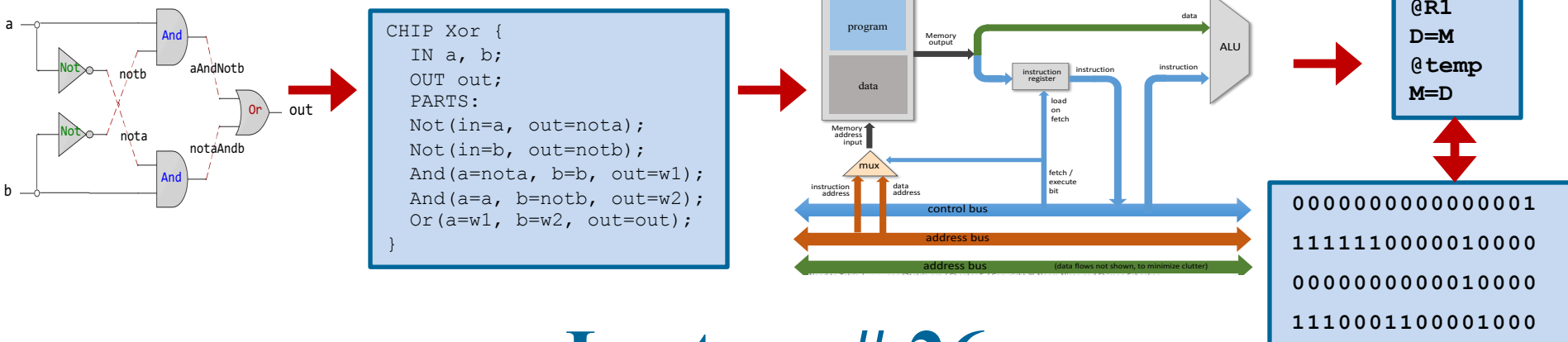
Lecture # 26
Data Path of Hack CPU

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
    msg: db "Learning is fun with Arif", 0Ah, 0h
    len_msg: equ $ - msg
SECTION .text
    main:
        mov rax,1
        mov rdi,1
        mov rsi,msg
        mov rdx,len_msg
        syscall
        mov rax,60
        mov rdi,0
        syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- Von Neumann Architecture

- Flow of Information inside Computers

- Buses

  - Data Bus

  - Address Bus

  - Control Bus

- Fetch Execute Cycle

- Fetch Execute Clash

- Harvard Architecture

- Hack CPU Interface

- Hack CPU Implementation

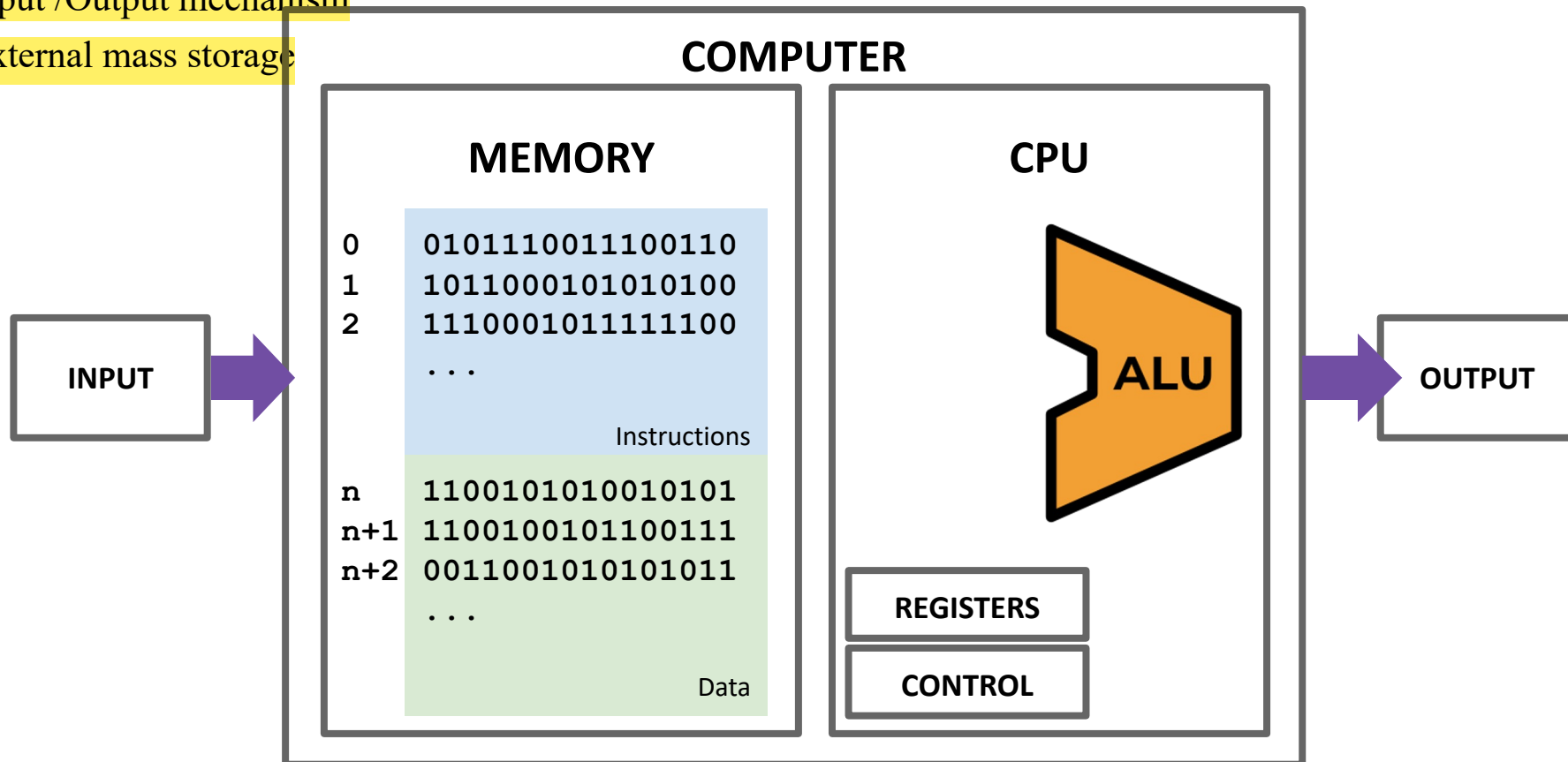- Input/output and Operations of Hack ALU

- Control Logic of Hack CPU

Von Neumann Architecture

Instructor: Muhammad Arif Butt, Ph.D.
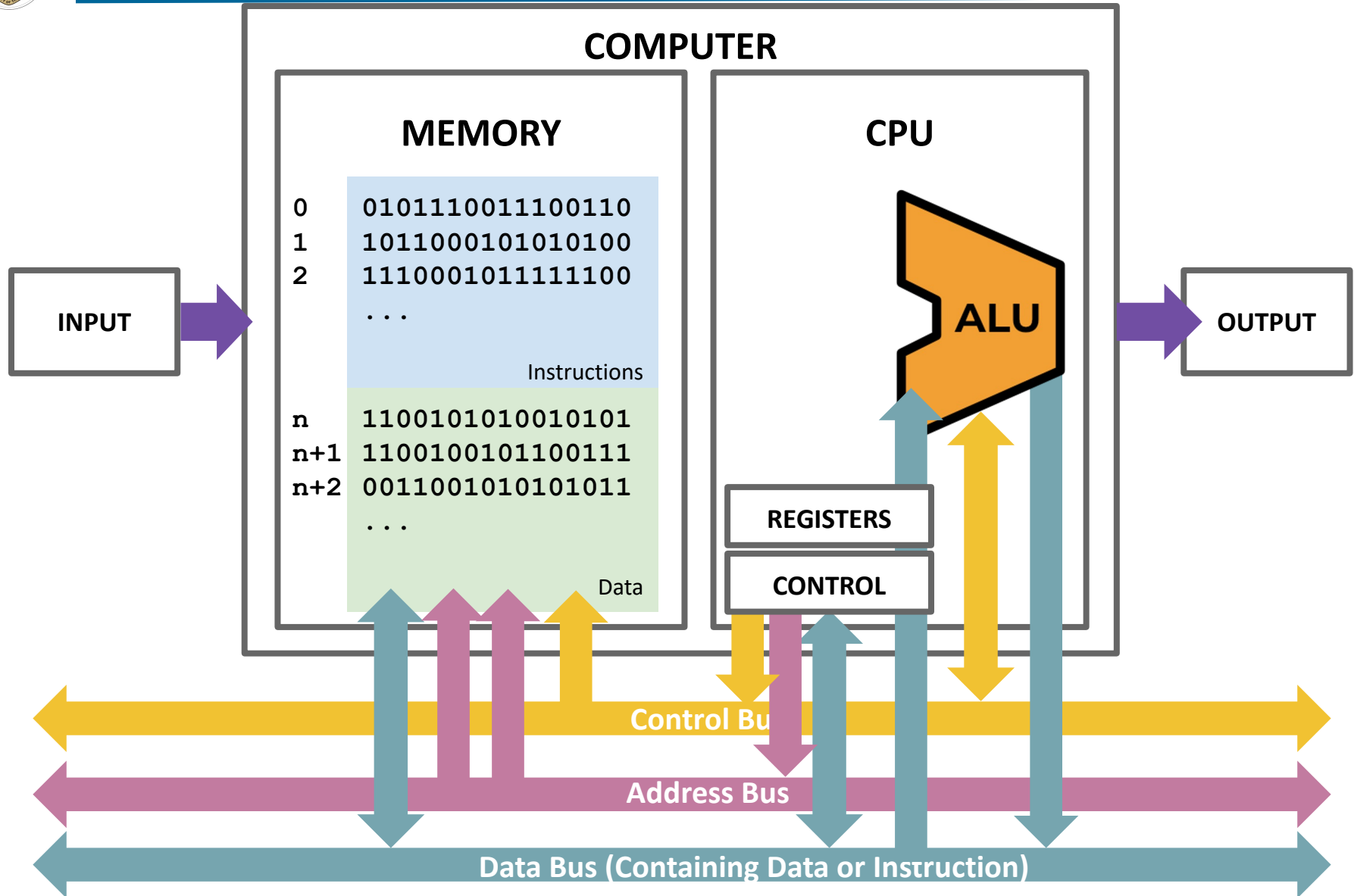
Von Neumann Architecture

The Von Neumann architecture is a computer architecture given by a mathematician and physicist John von Neumann describes the design architecture for an electronic digital computer with these components:

➤ A Processing Unit that contains an ALU and registers

➤ A Control Unit that contains an instruction register and program counter

➤ **A Memory unit that stores both data and instructions**

➤ Input /Output mechanism

➤ External mass storage

COMPUTER

| INPUT | | | OUTPUT |

MEMORY

```
0   0101110011100110
1   1011000101010100
2   1110001011111100
    . . .
```
Instructions

```
n    1100101010010101
n+1  1100100101100111
n+2  0011001010101011
     . . .
```
Data

CPU

ALU

REGISTERS

CONTROL

Information Flow / CPU Data Path

COMPUTER

MEMORY

0	0101110011100110
1	1011000101010100
2	1110001011111100
	. . .

Instructions

n	1100101010010101
n+1	1100100101100111
n+2	0011001010101011
	. . .

Data

CPU

ALU

REGISTERS

CONTROL

INPUT

OUTPUT

Control Bus

Address Bus

Data Bus (Containing Data or Instruction)

Instructor: Muhammad Arif Butt, Ph.D.

5

Overview of General Fetch-Execute Cycle

Instructor: Muhammad Arif Butt, Ph.D.

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

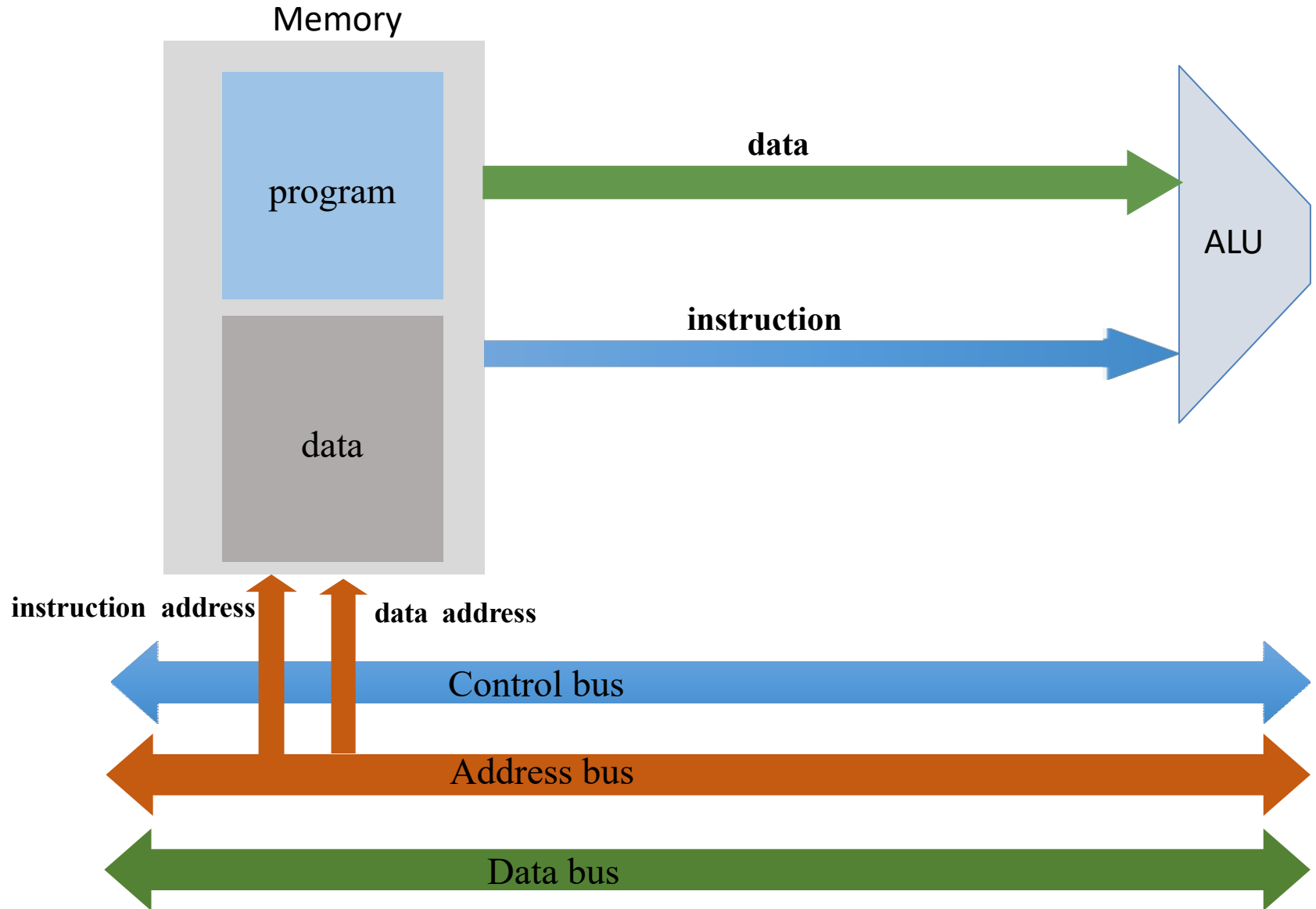- Read the contents of the memory from that location to get the instruction code

program

Memory output

Instruction

data

Memory address input

Program Counter

- Default: PC++
- Jump: PC = address
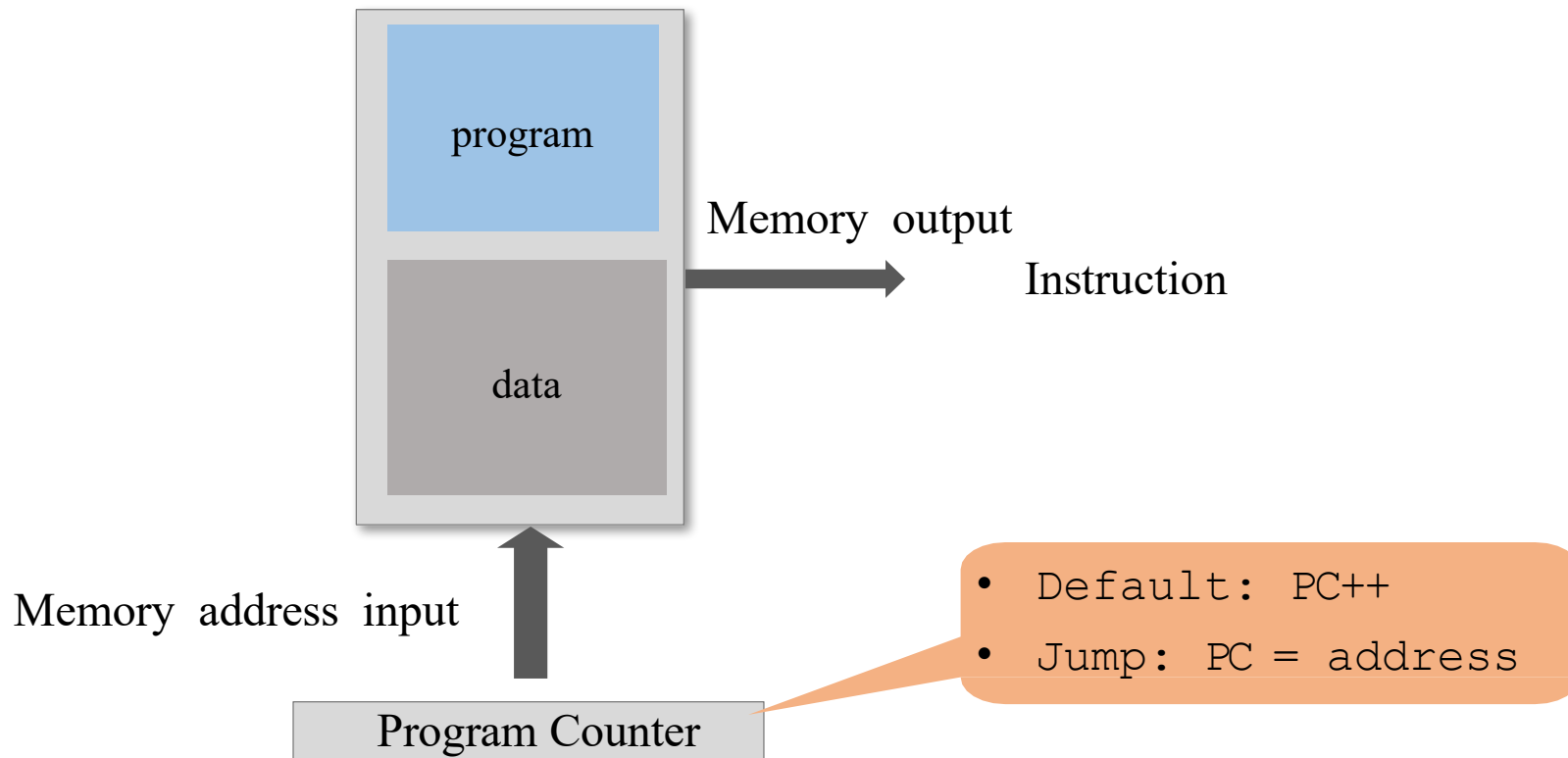
Executing

- 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
    - …

  Different subset of the instruction bits controls different aspects of the operation

- Executing the instruction involves:
    - accessing registers and / or
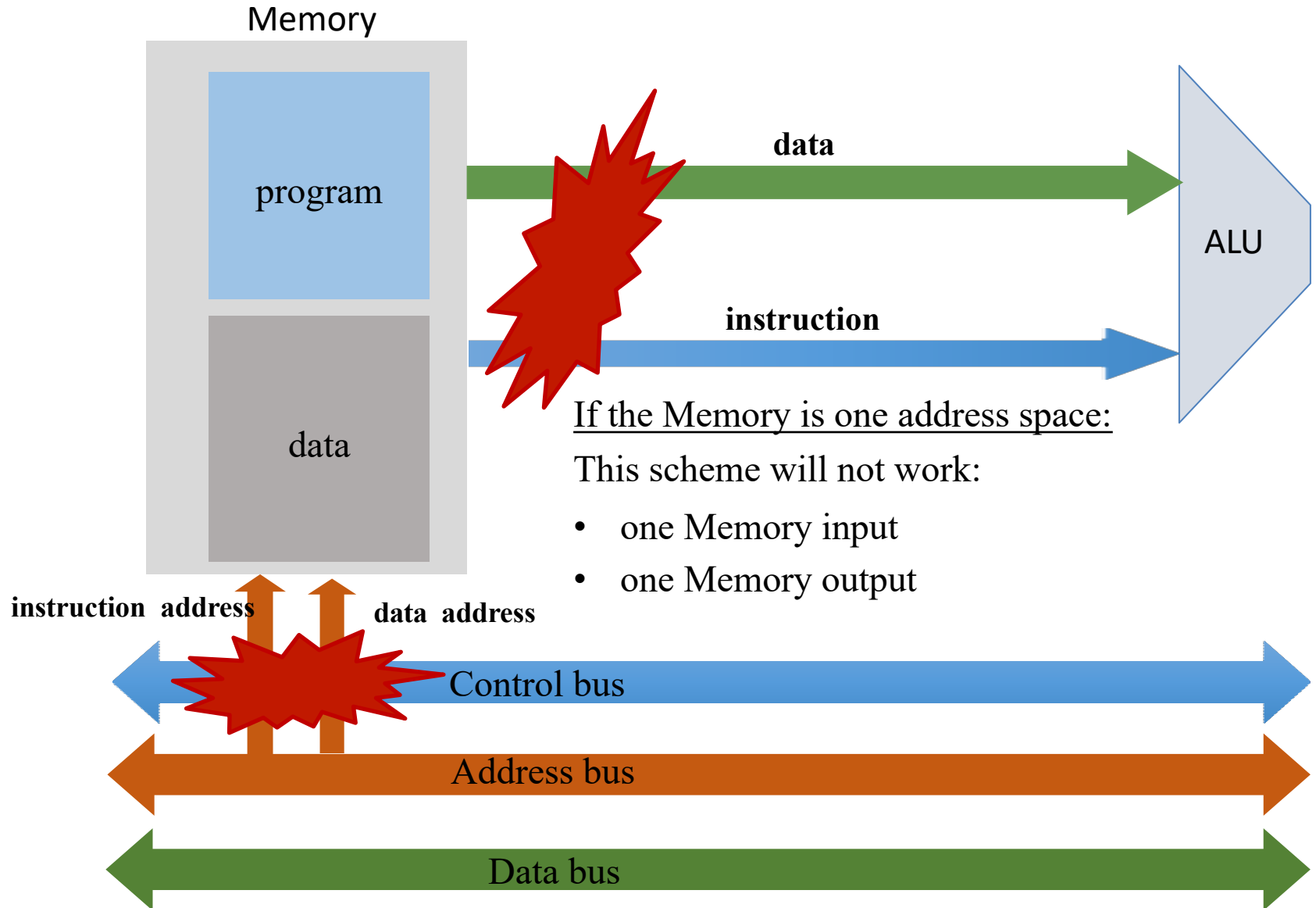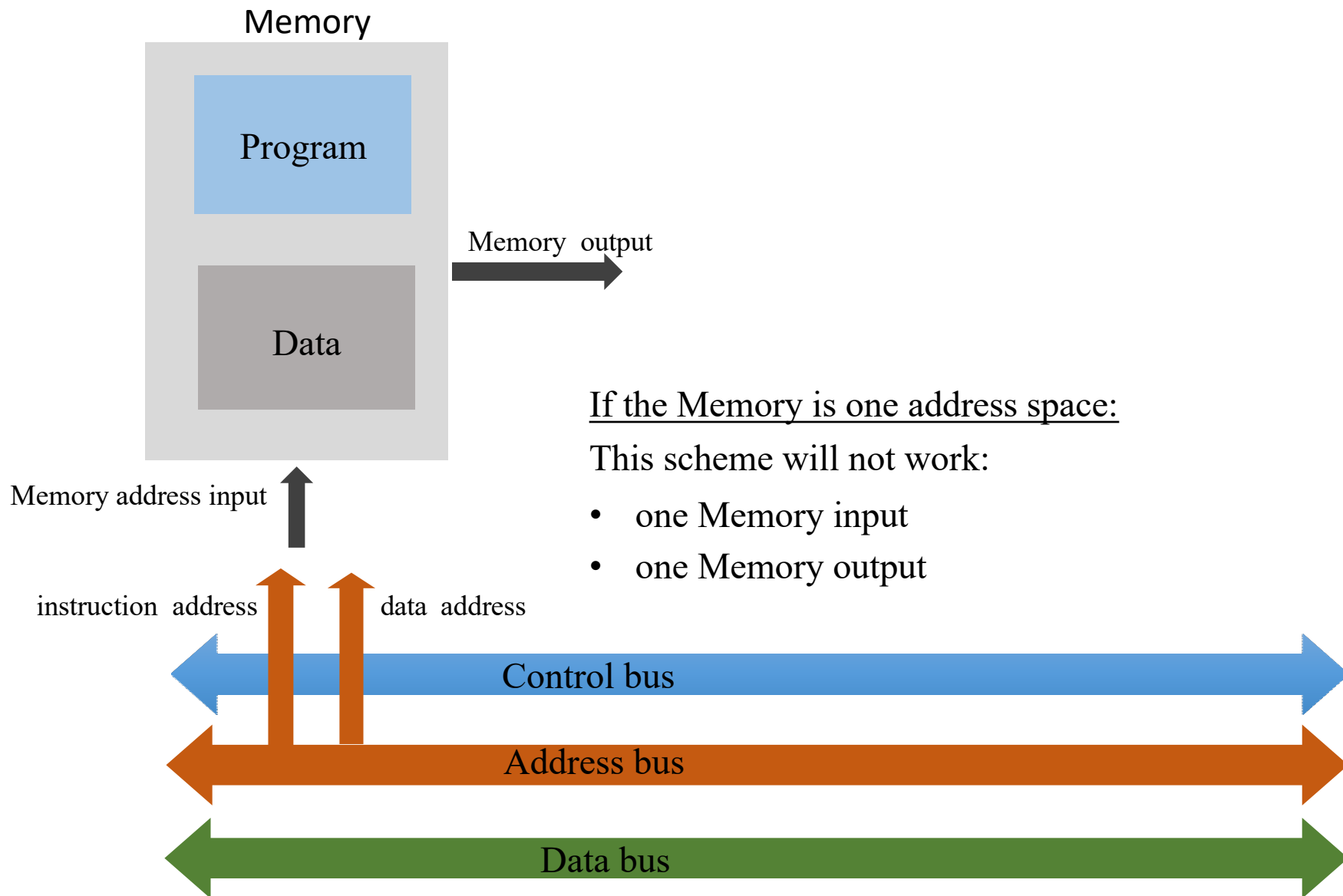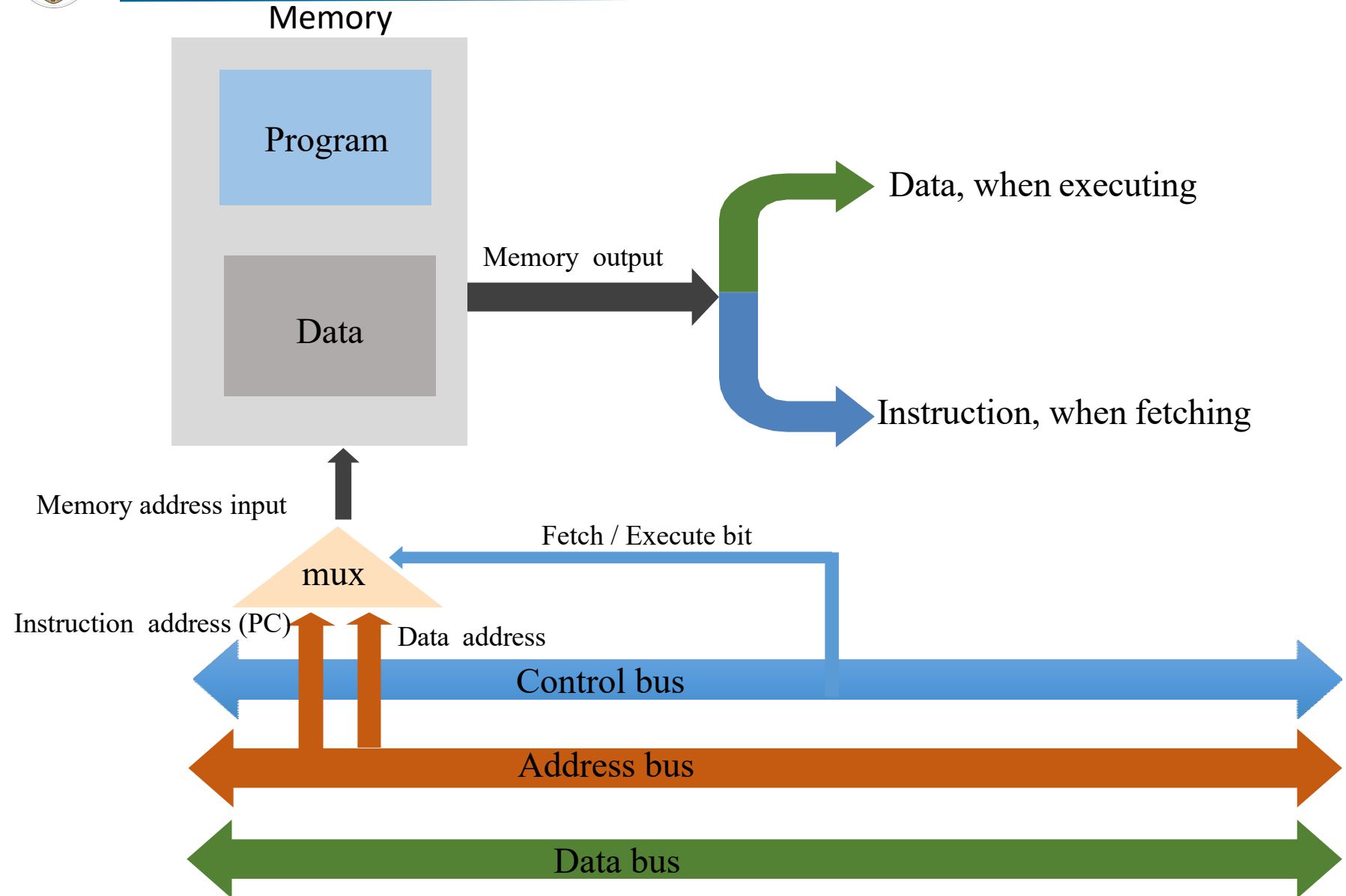   - accessing the data memory

Fetch Execute

Memory

program

data

ALU

data

instruction

instruction address

data address

Control bus

Address bus

Data bus

Fetch-Execute Clash

Memory

program

data

data

instruction

ALU

If the Memory is one address space:

This scheme will not work:

- one Memory input
- one Memory output

instruction address   data address

Control bus

Address bus

Data bus

Instructor: Muhammad Arif Butt, Ph.D.

Fetch-Execute Clash (cont…)

Memory

Program

Data

Memory output

Memory address input

instruction address   data address

If the Memory is one address space:

This scheme will not work:

- one Memory input
- one Memory output

Control bus

Address bus

Data bus

Instructor: Muhammad Arif Butt, Ph.D.

Solution: multiplex, using an instruction register

Solution: multiplex, using an instruction register

Memory

Program

Data

Memory output

data

ALU

Instruction register

instruction

load on fetch

instruction

Memory address input

mux

instruction address

data address

fetch / execute bit

Control bus

Address bus

Data bus
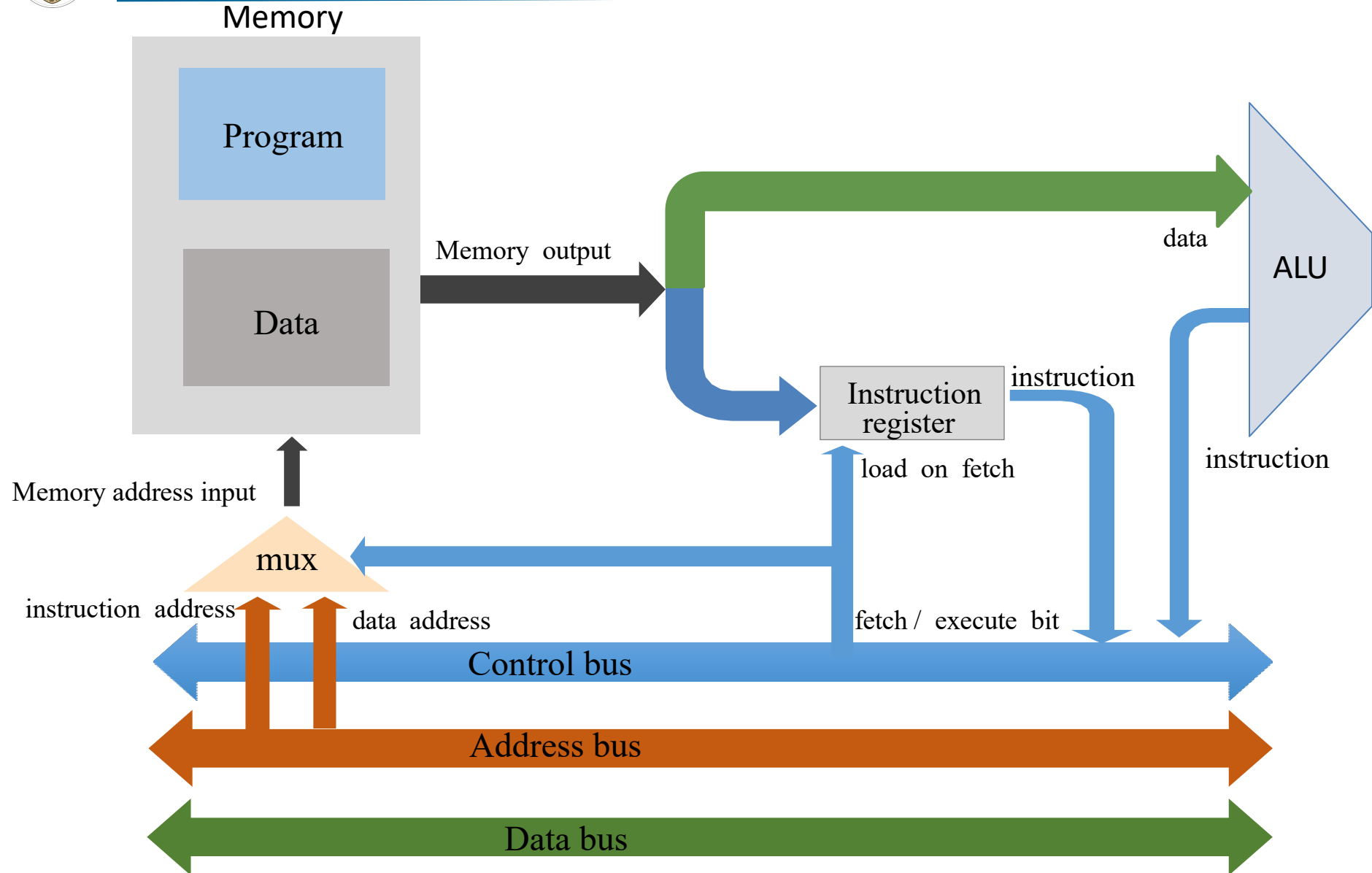
Simpler Solution: Harvard Architecture

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 seperately, and simultaneously
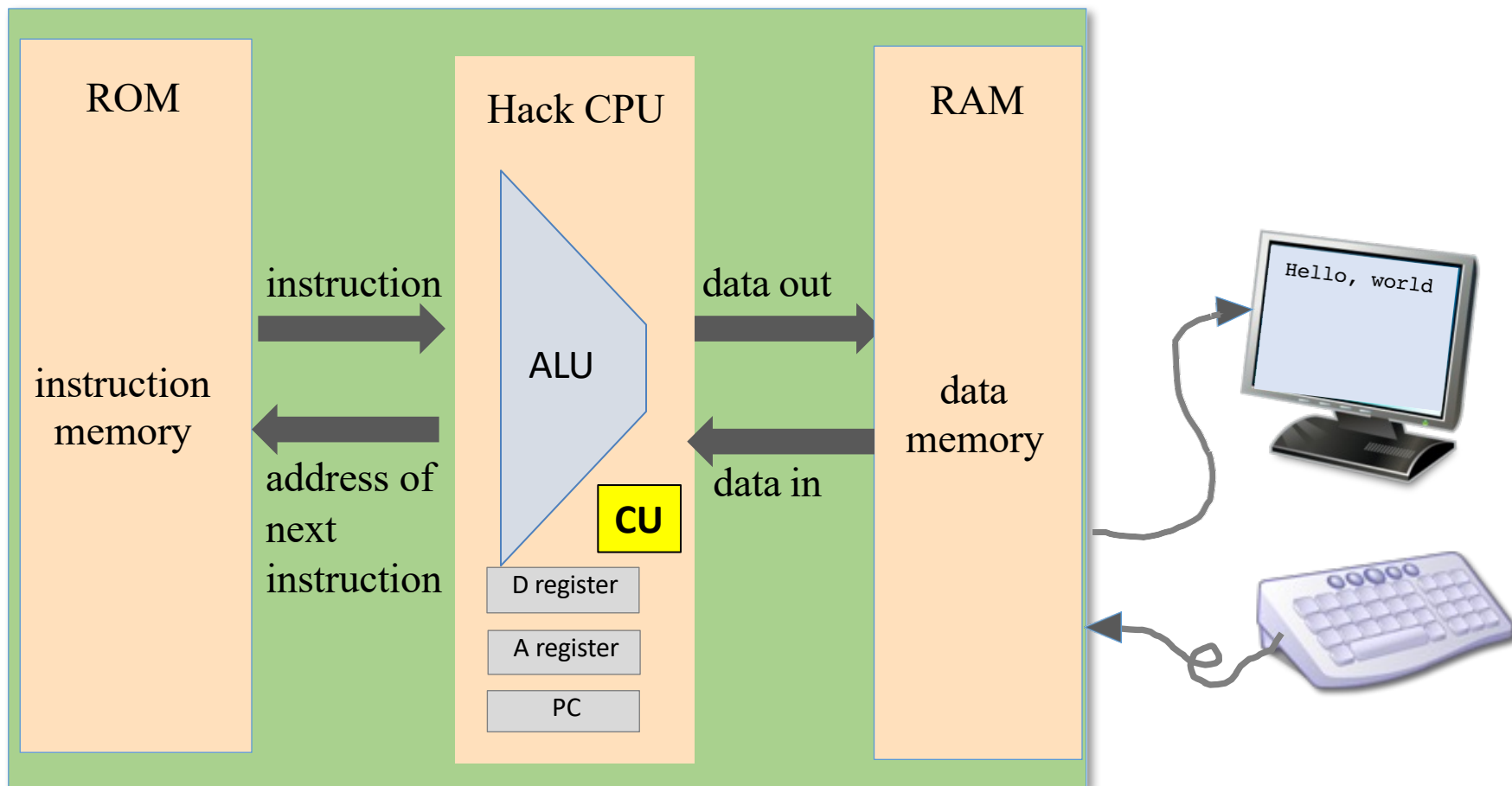
Advantage:

- Complication avoided

Disadvantage:

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

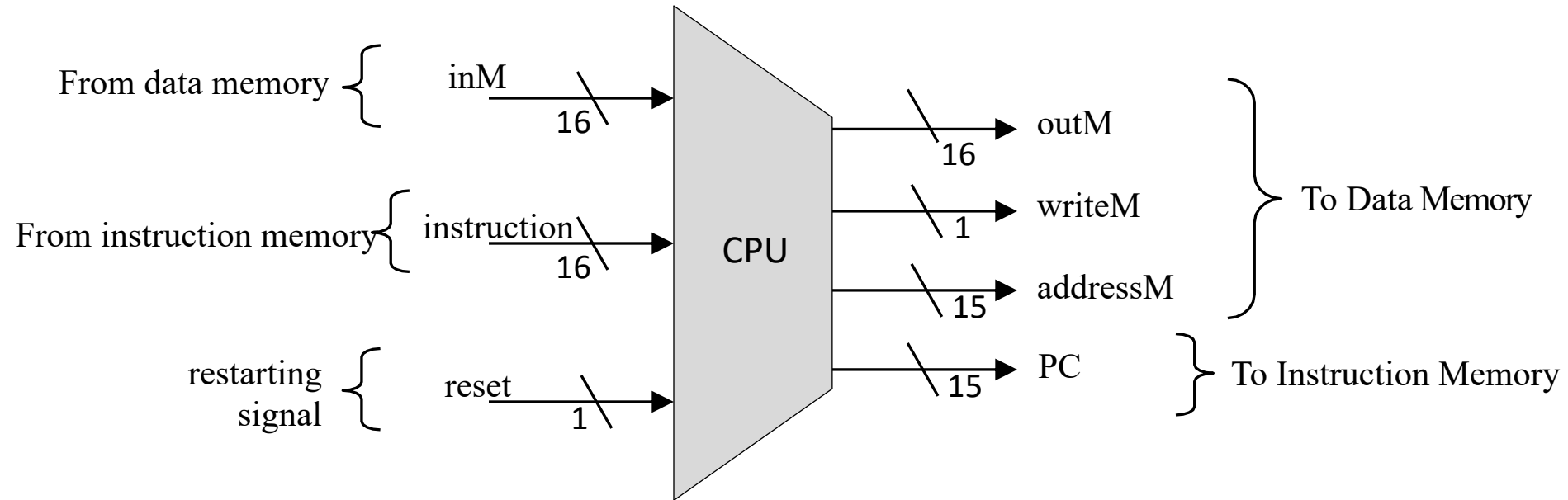# Designing the Data Path of Hack CPU

Instructor: Muhammad Arif Butt, Ph.D.

Hack Computer Architecture

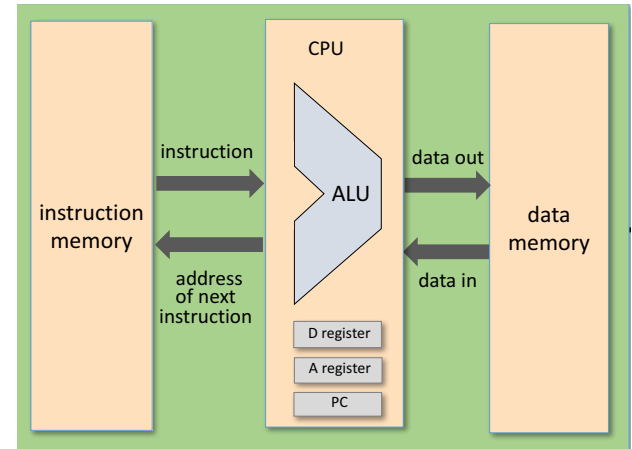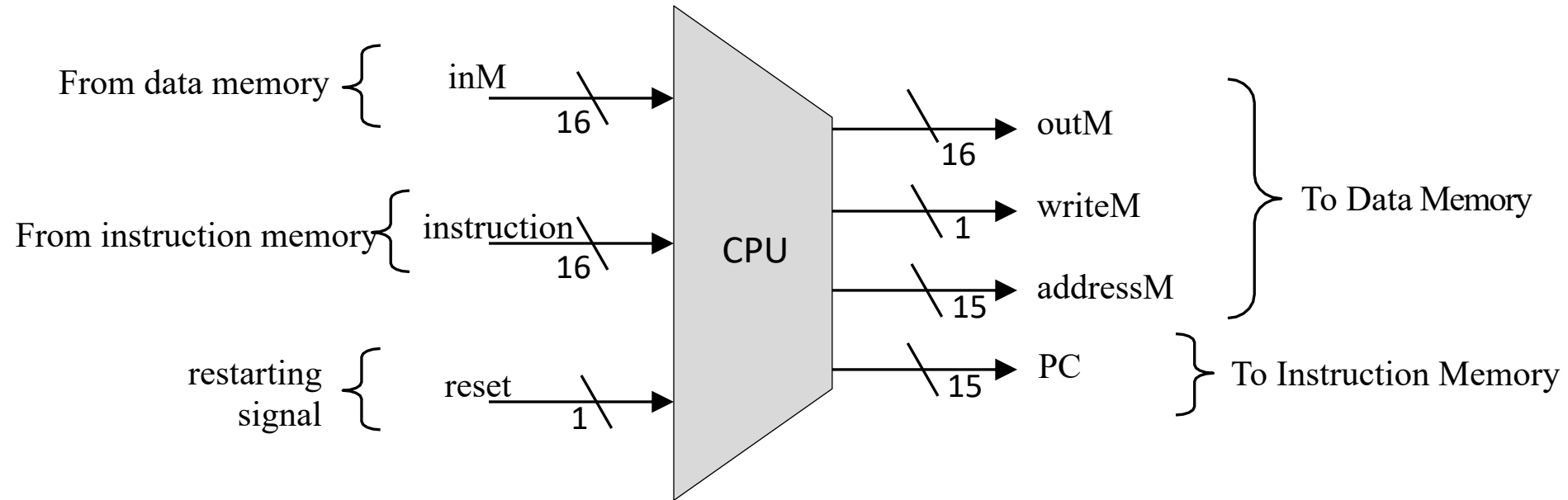Instructor: Muhammad Arif Butt, Ph.D.

Hack CPU Interface
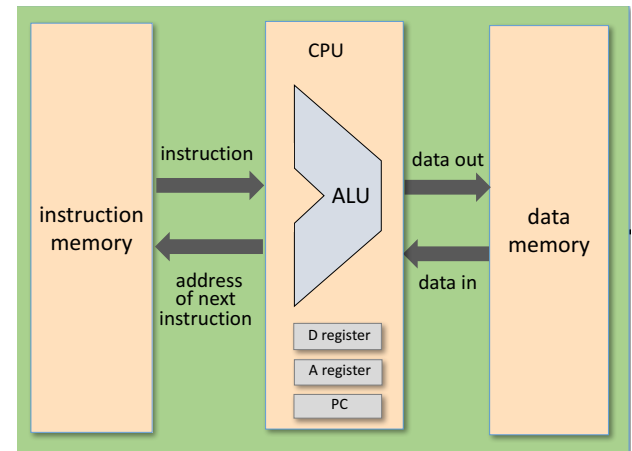

Inputs:
- Data Value
- Instruction
- Reset Bit

Hack CPU Interface

Outputs:
- Data Value
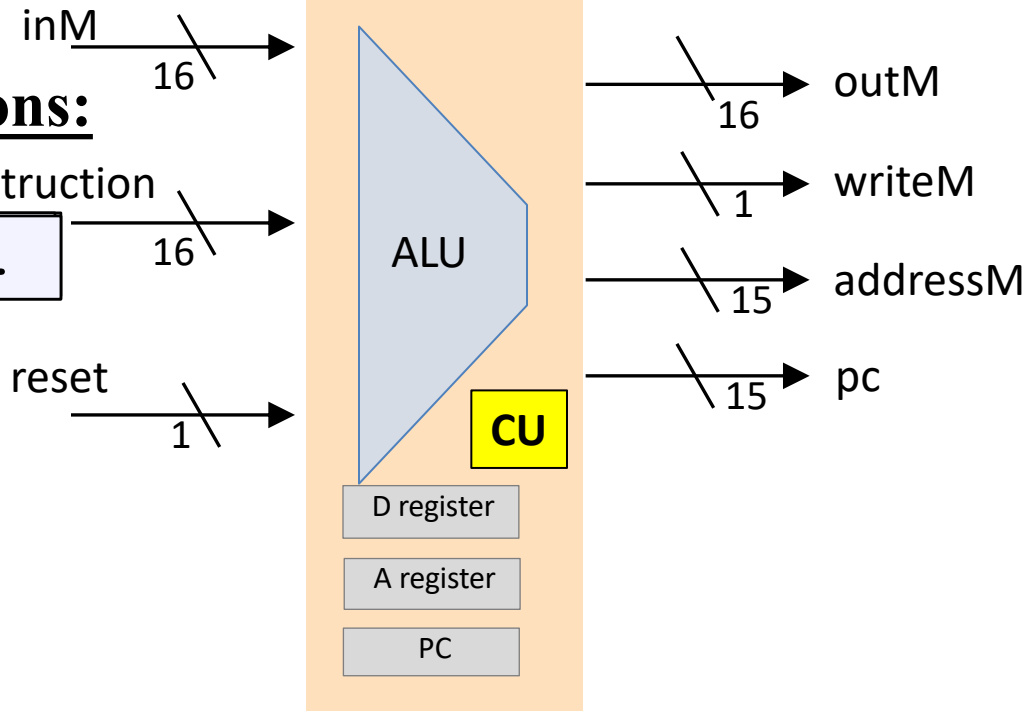- Write to Memory? (yes/no)
- Memory Address
- Address of next instruction

Hack Instructions:

D=M+1

Hack CPU

inM ——16——→

instruction ——16——→

reset ——1——→

ALU

CU

D register

A register

PC

——16——→ outM

——1——→ writeM

——15——→ addressM

——15——→ pc

Reset Bit and PC:

If (reset == 1)

The 15 bit output **pc** emits 0, causing the program to restart

If (reset==0)

The CPU logic uses the instruction's jump bits and the ALU's output to decide if there should be a jump. For example:

```
@54
D-1;JEQ
```

If (D-1==0)

PC is set to the value of the A-register

else

PC++

Hack CPU

inM →16→ ALU

instruction →16→

reset →1→

outM ←16

writeM ←1

addressM ←15

pc ←15

CU

D register

A register

PC
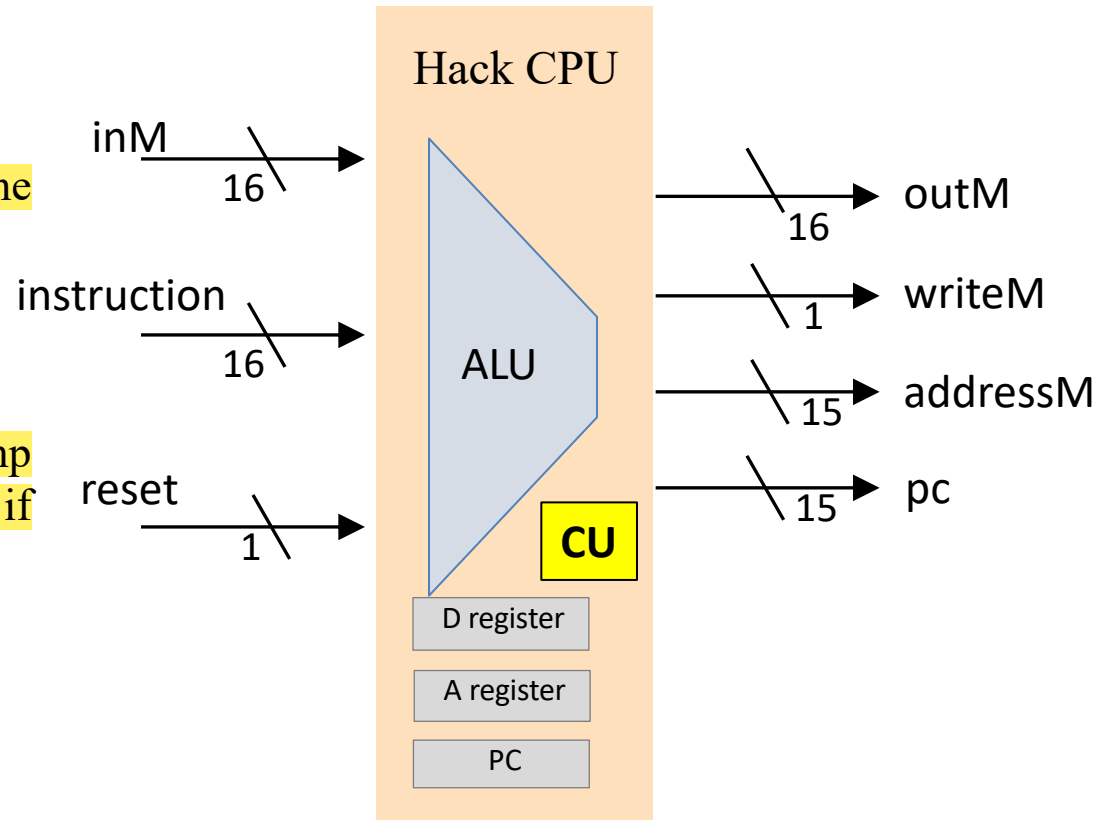
The updated PC value is emitted by 15 bit output named **PC**

Hack CPU Implementation
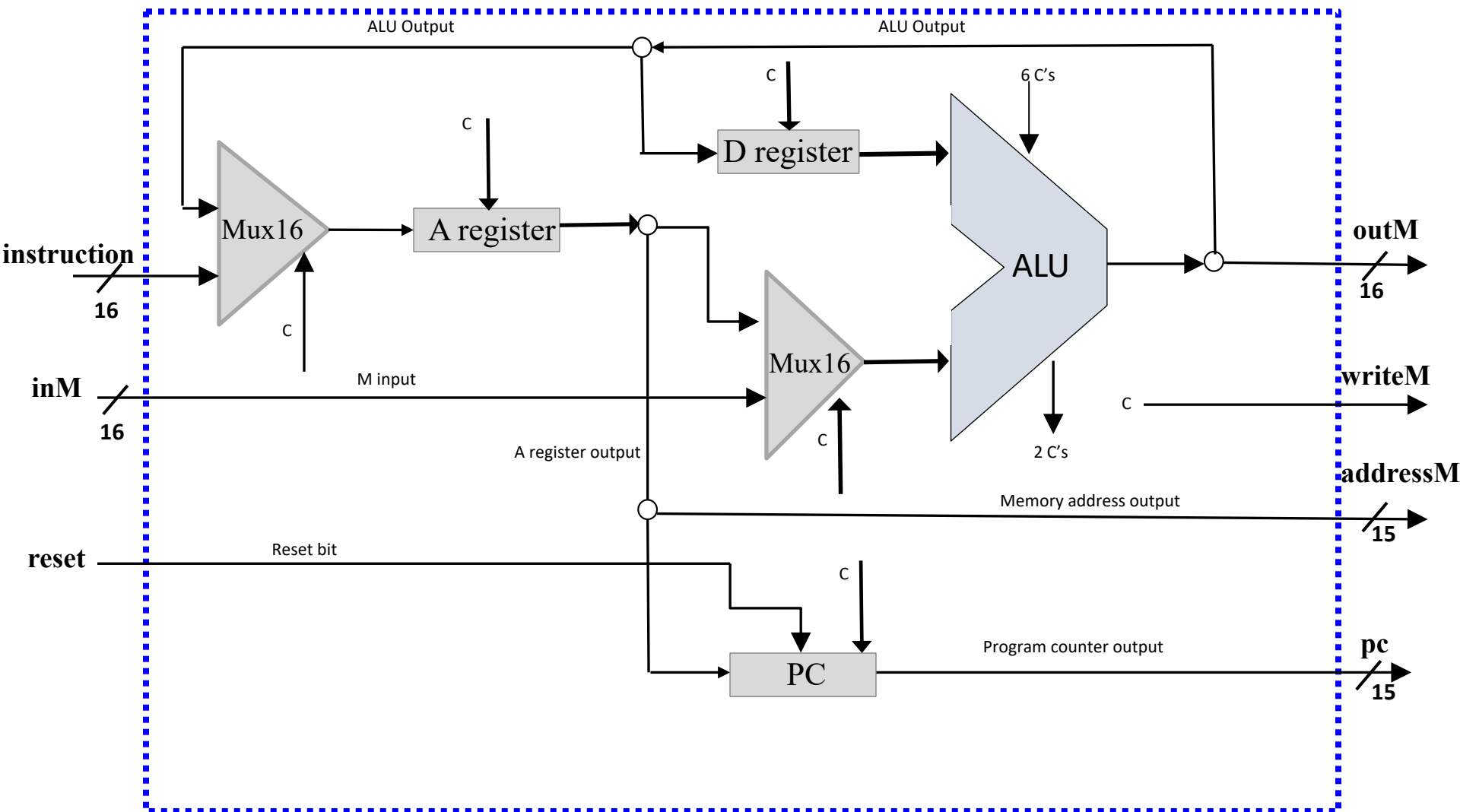
(each "C" symbol represents a control bit)

How A/C-Instructions Execute?

Instruction Handling

Handling A-Instruction

ALU Output

Instruction

Mux16

c

A register

c
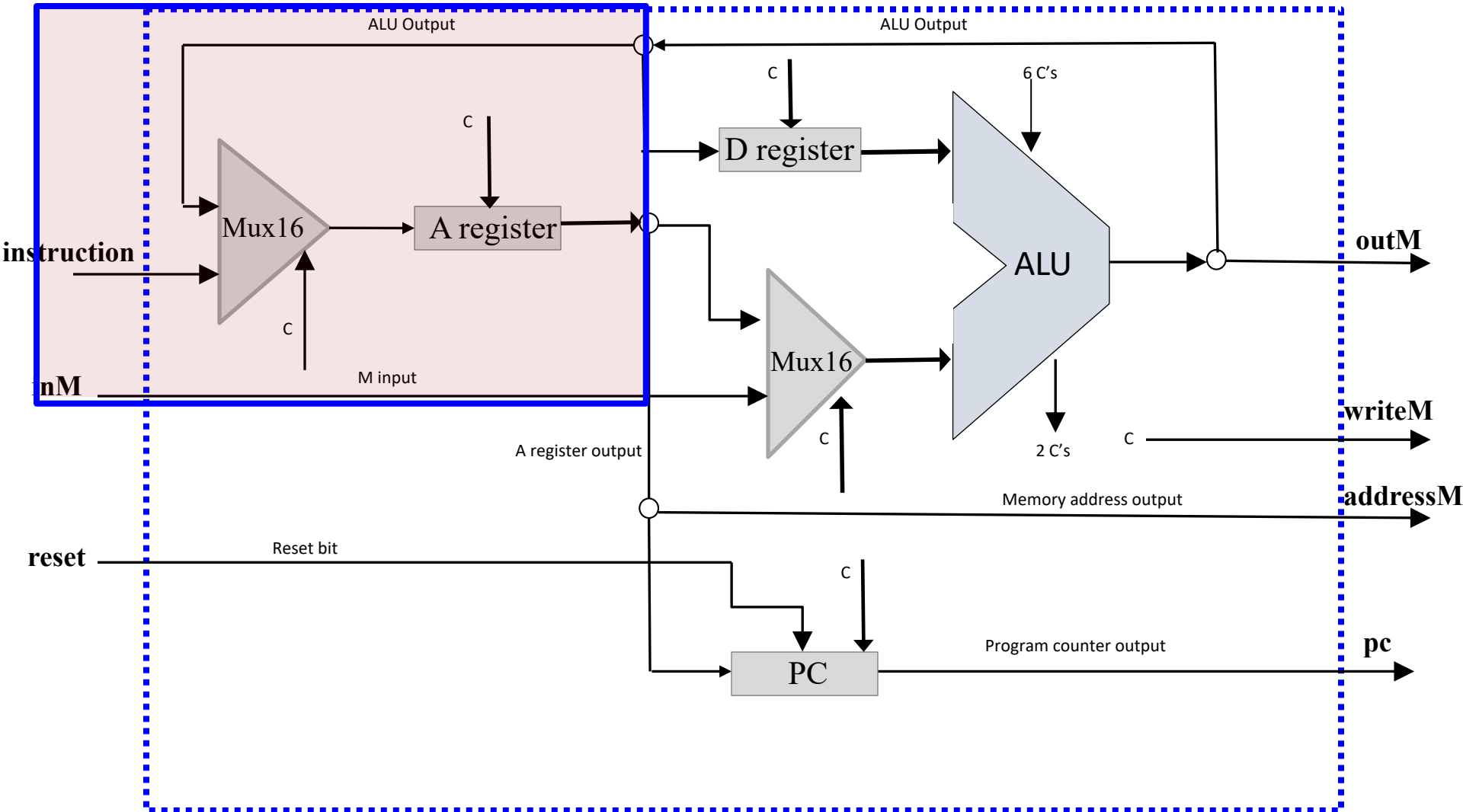
Bit#15 of Instruction
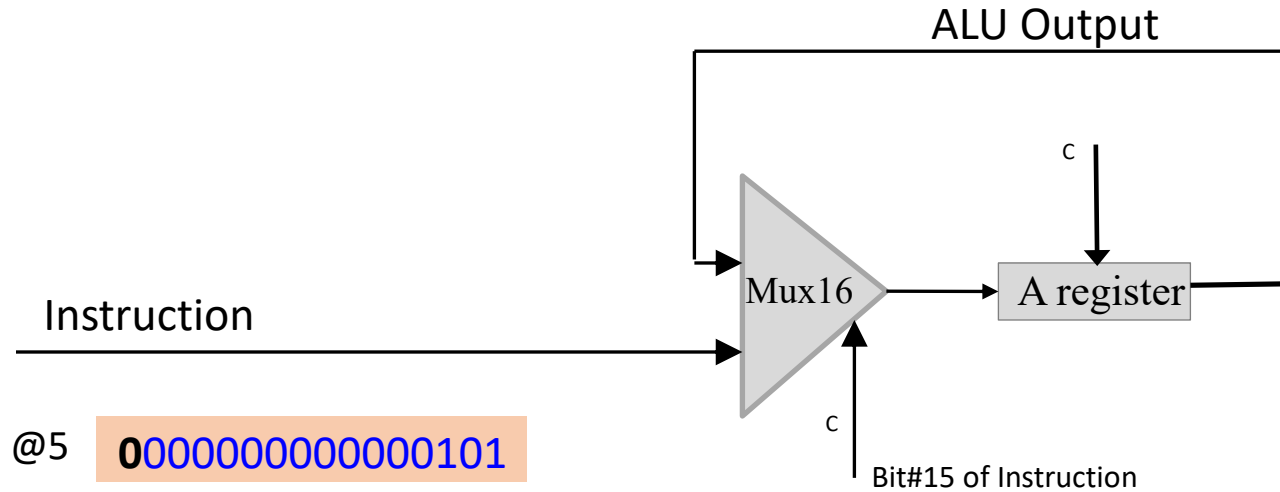
@5   **0**000000000000101

A-instruction

<u>CPU handling of an A-instruction:</u>
- Decodes the instruction into
  - op-code
  - 15-bit value
- Stores the 15 bit value in the A-register
- Outputs the value to ALU via Mux (not shown in this diagram)

Note:
- In case of A-instruction, the A-register get its input from the instruction part
- In case of C-instruction, the A-register get its input from the ALU output

Handling C-Instruction

ALU Output

Instruction

Mux16 → A register

c

c

Bit#15 of Instruction
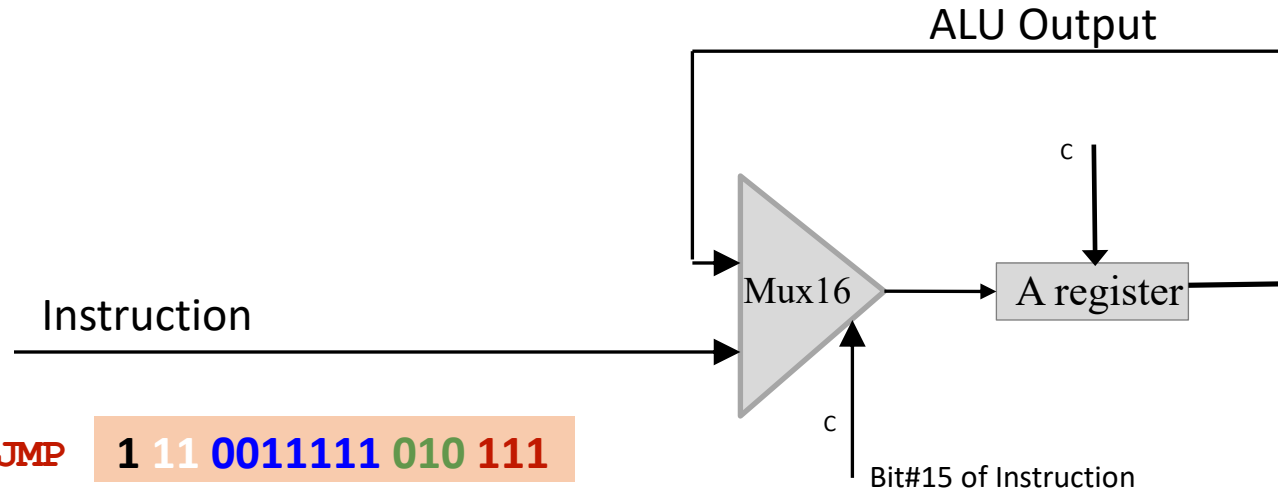
D= D+1 ;JMP   **1** 11 **0011111** 010 **111**

C-instruction

CPU handling of C-instruction:

`dest= comp ;jump`

- Decodes the instruction bits into:
  - Op-code
  - ALU control bits (D+1)
  - Destination load bits (D-Register)
  - Jump bits (Un-condional jump)
- Routes these bits to their chip-part destinations
- The chip-parts (most notably, the ALU) execute the instruction

Control Input of two Mux16 Chips

Instructor: Muhammad Arif Butt, Ph.D.

Select Input of First Mux16 Chip
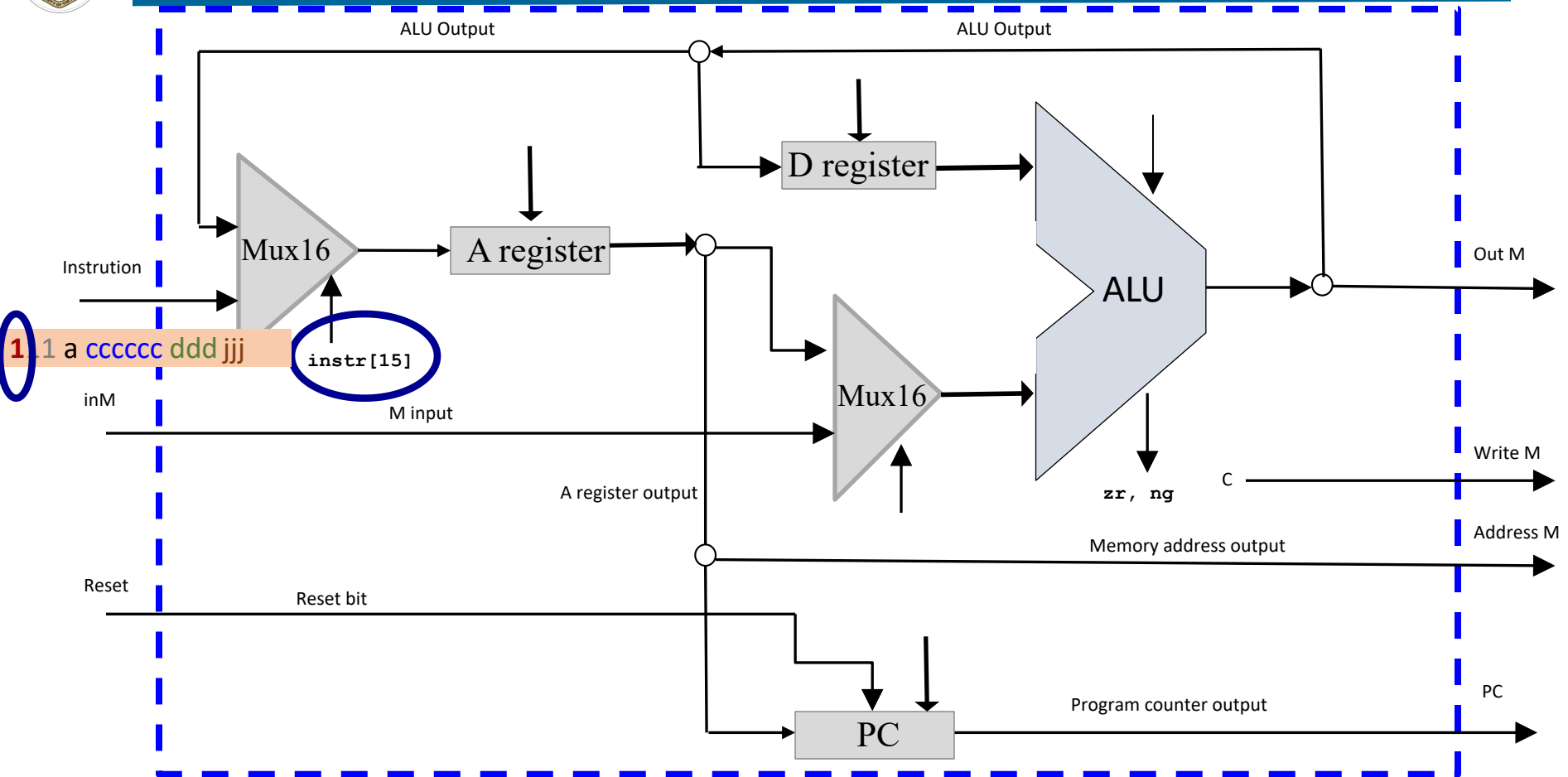
Select Input of Second Mux16 Chip

comp		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a==0	a==1						

Instructor: Muhammad Arif Butt, Ph.D.

ALU Operations

Instructor: Muhammad Arif Butt, Ph.D.
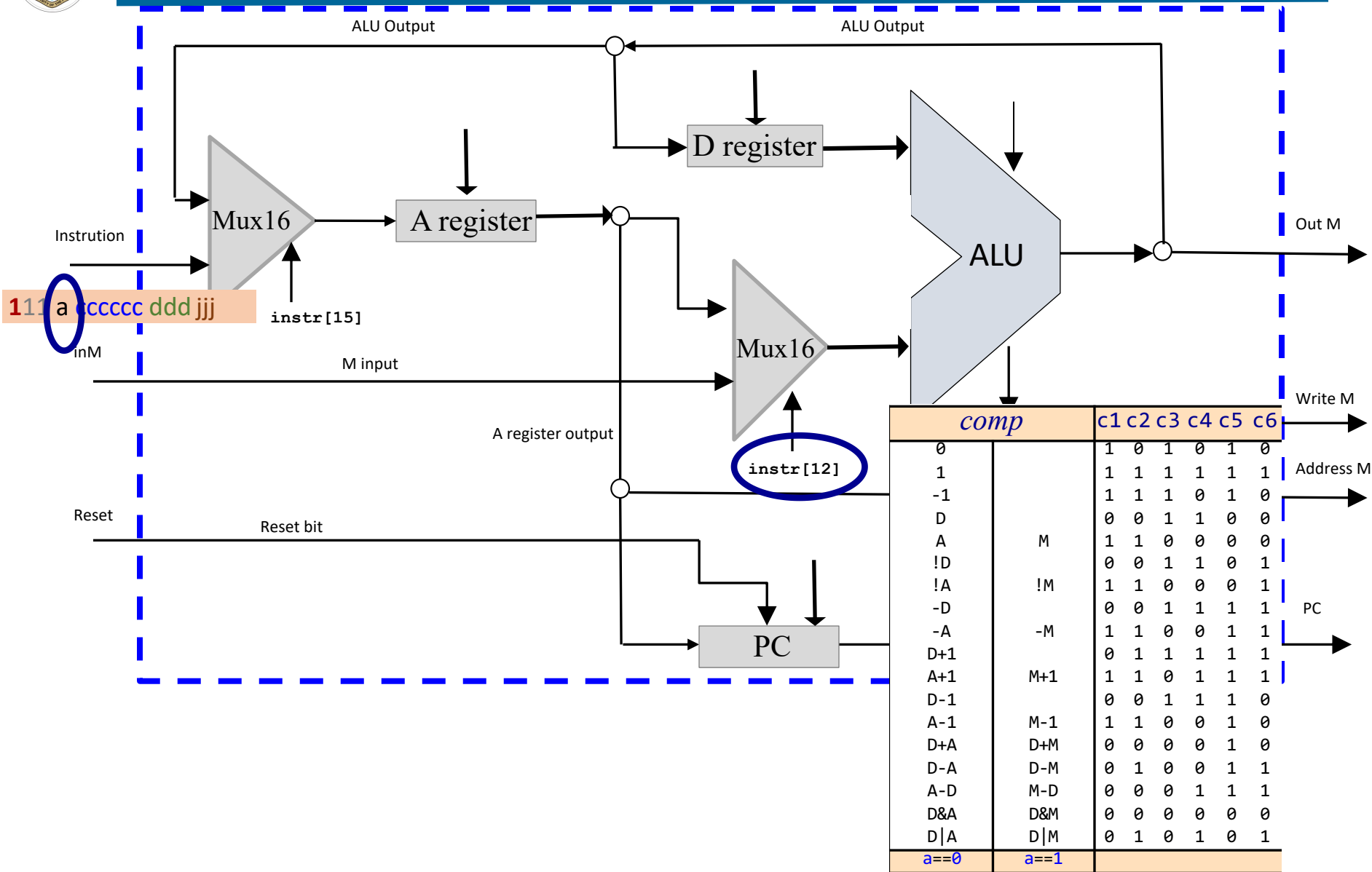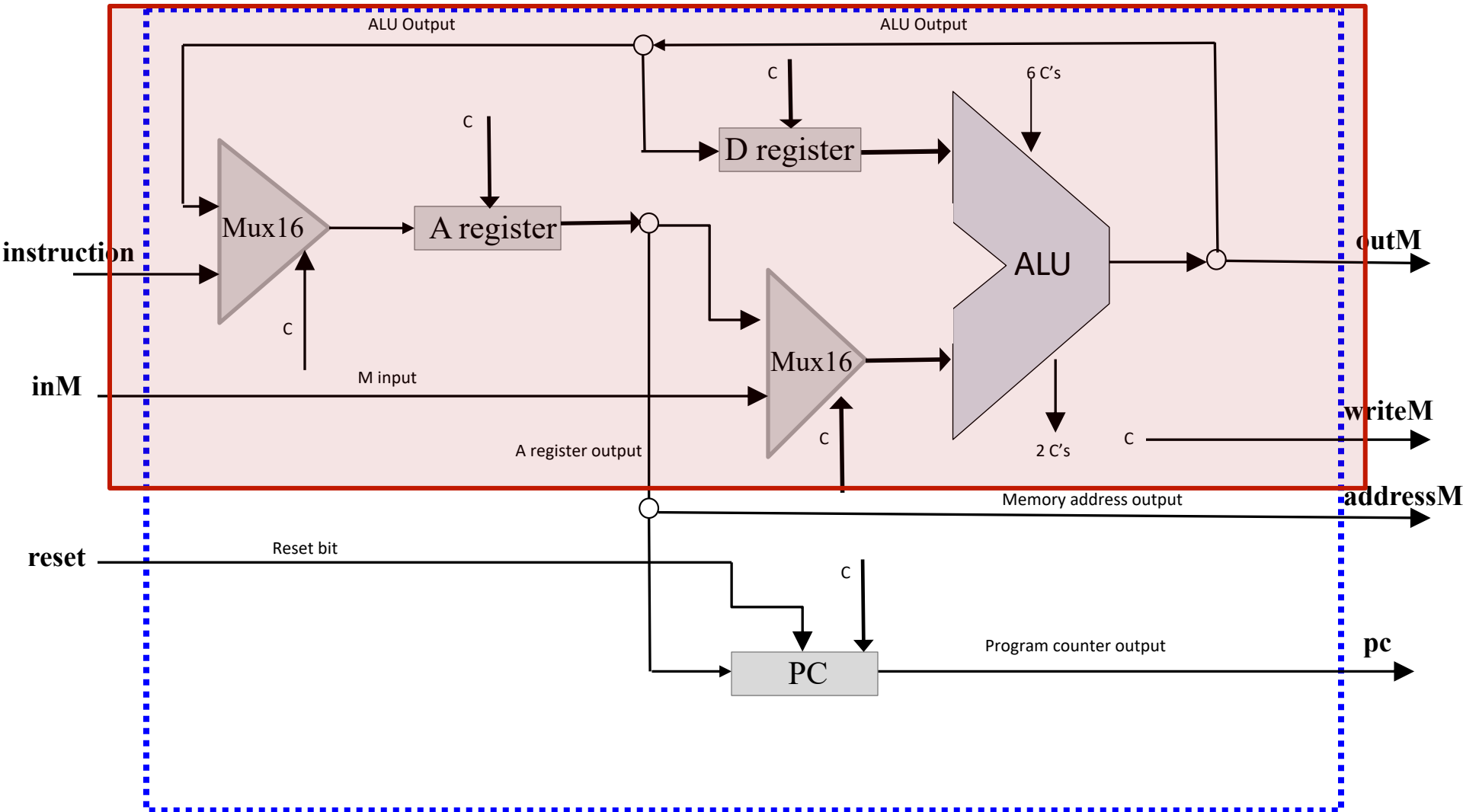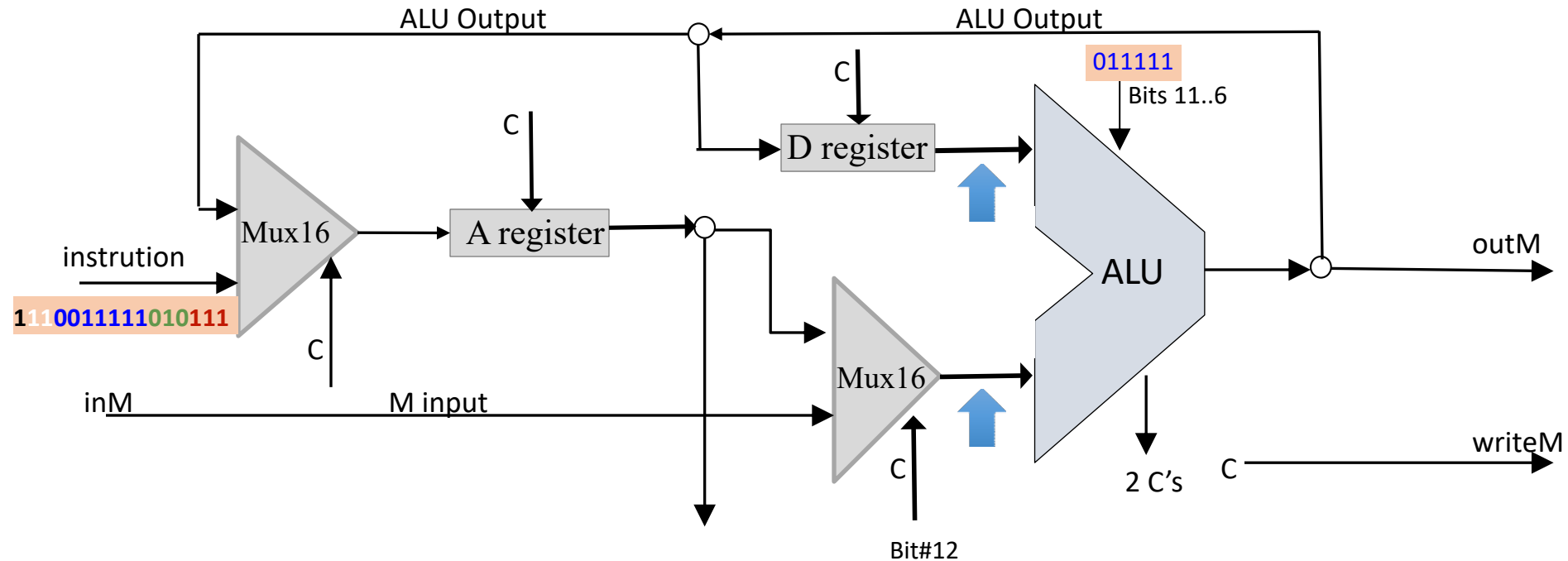
ALU Operation

ALU data inputs:

- Input 1: From D-register

- Input 2: From A-register or M-register (decided by bit#12 of Instruction: **a**)

ALU control inputs:

- 6 x Control bits (from bits 6-11 of the instruction: **ccccc**)

ALU Operation: Outputs
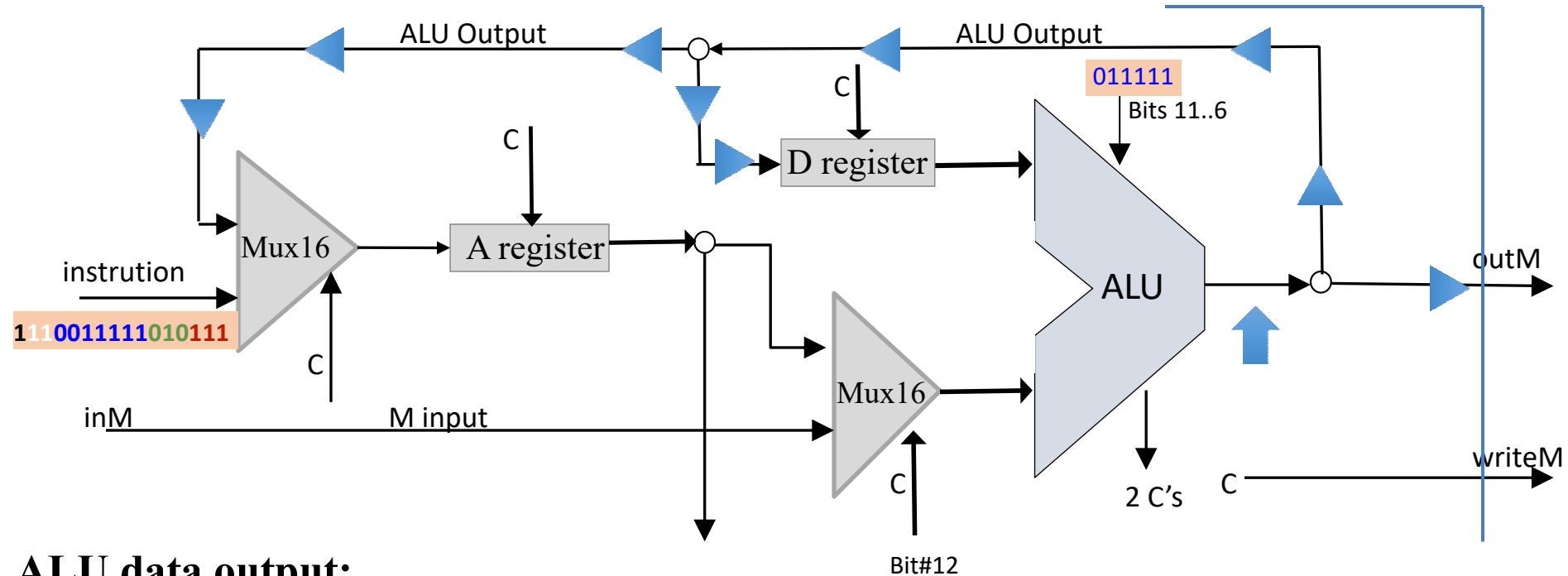
ALU data output:

- Result of ALU calculation, fed simultaneously to three locations:

  - D-register, A-register, M-register (data memory)

- Which out of these three destinations actually commits to the ALU output is determined by the instruction's destination bits

Note: **000** in the destination bits means don't save the ALU output, and **111** means save it simultaneously in D, A and M registers
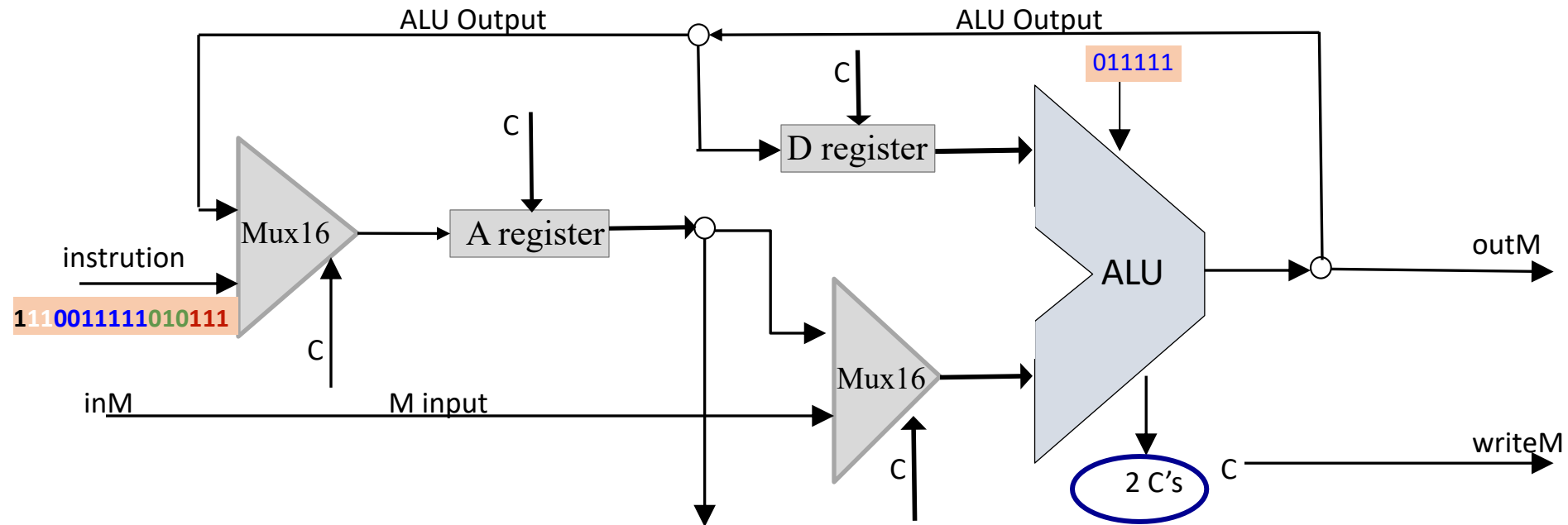
ALU control outputs:

- is the output negative? (ng)

- is the output zero? (zr)

Control Inputs of A-Register and D-Register Chips

Load Input of A-Register Chip

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

Instructor: Muhammad Arif Butt, Ph.D.

36

Load Input of D-Register Chip

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

Instructor: Muhammad Arif Butt, Ph.D.

Control Logic of PC Register

Control Logic of CPU
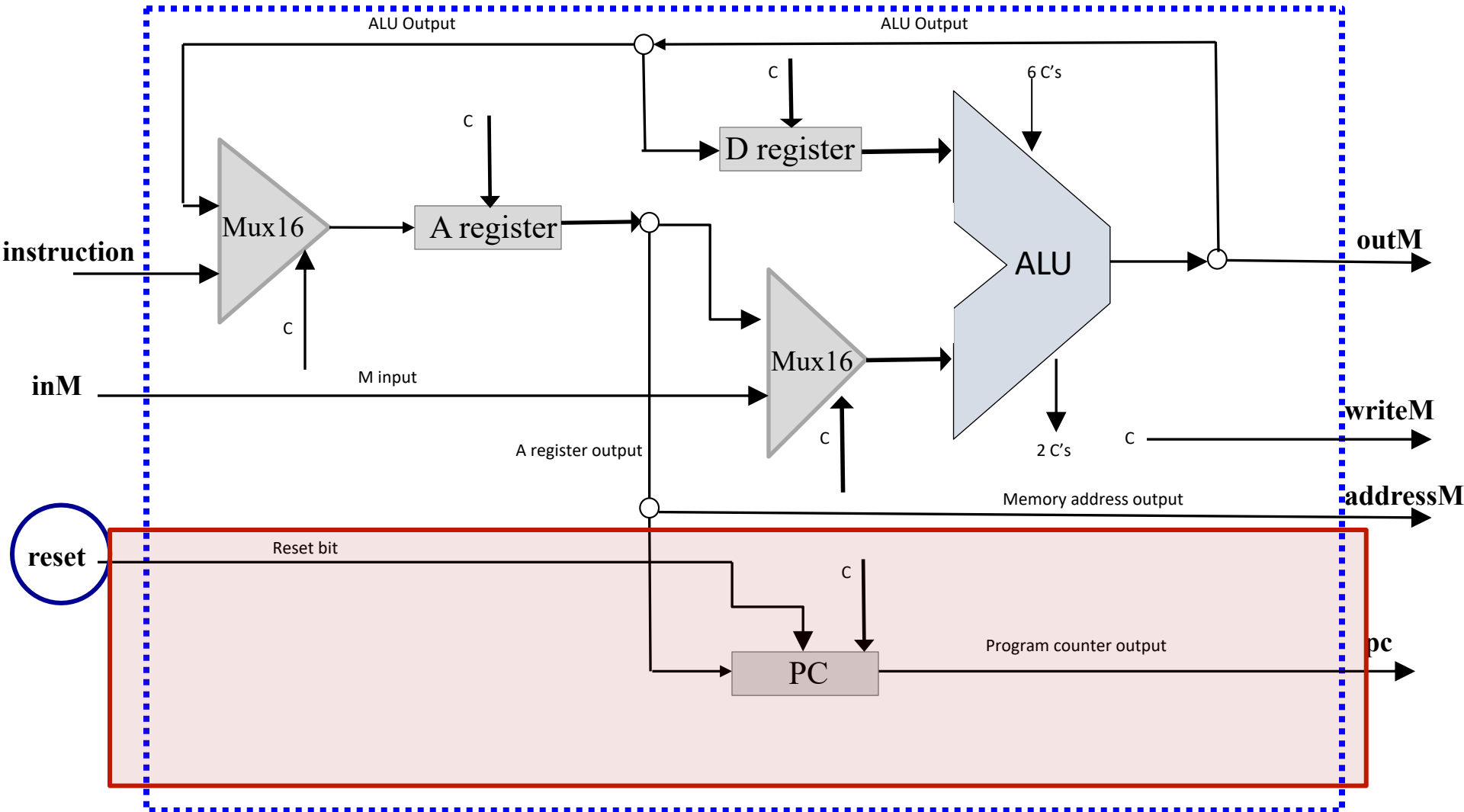
(each "C" symbol represents a control bit)

HACK

Reset Button

computer exterior
(well, kind of)
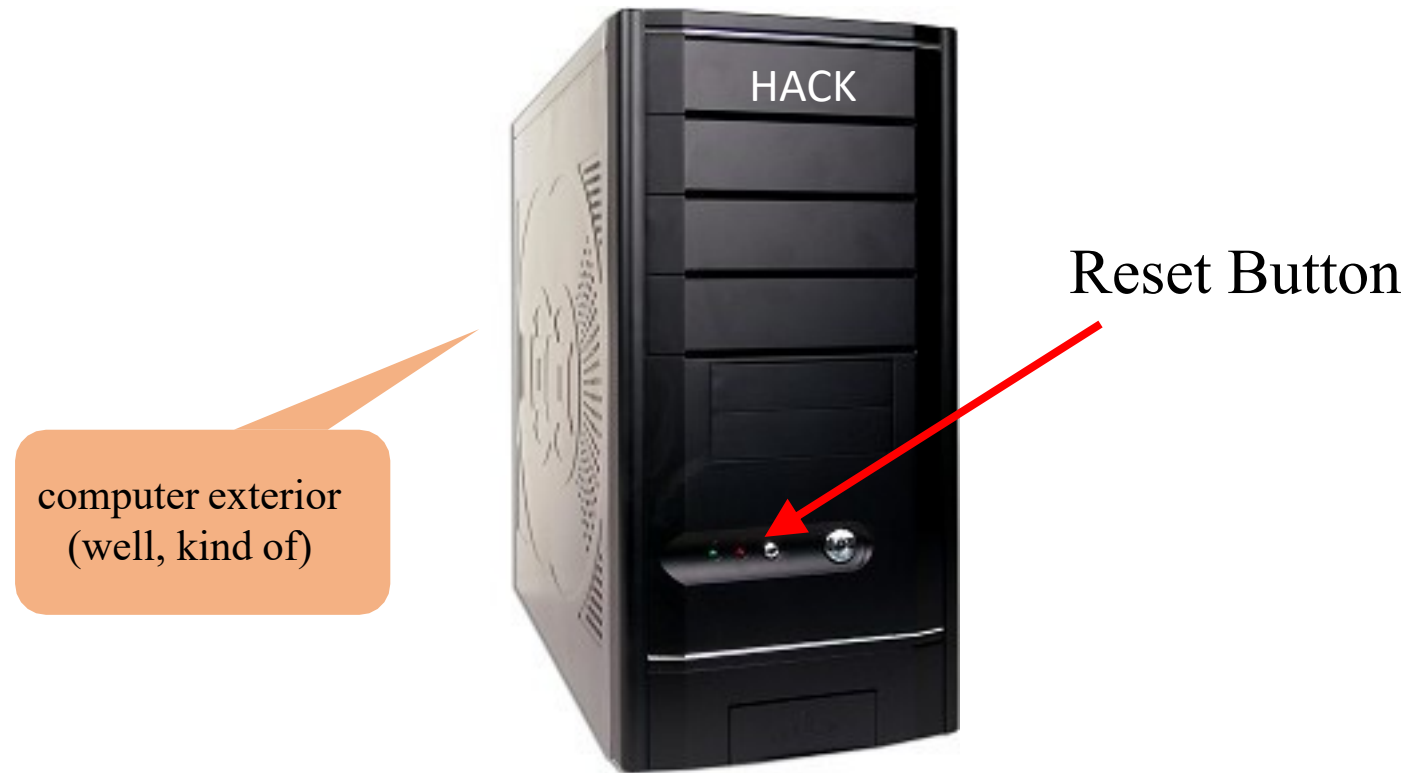
- The computer is loaded with some program
- Pushing reset button causes the program to start running from beginning

Control Abstraction

instruction

111 a c c c c c c d d d j j j

reset

C

A register

16

1

load

PC

16

PC

jump	j1 j2 j3	Effect
null	0 0 0	no jump
JGT	0 0 1	if out > 0 jump
JEQ	0 1 0	if out = 0 jump
JGE	0 1 1	if out ≥ 0 jump
JLT	1 0 0	if out < 0 jump
JNE	1 0 1	if out ≠ 0 jump
JLE	1 1 0	if out ≤ 0 jump
JMP	1 1 1	Unconditional jump
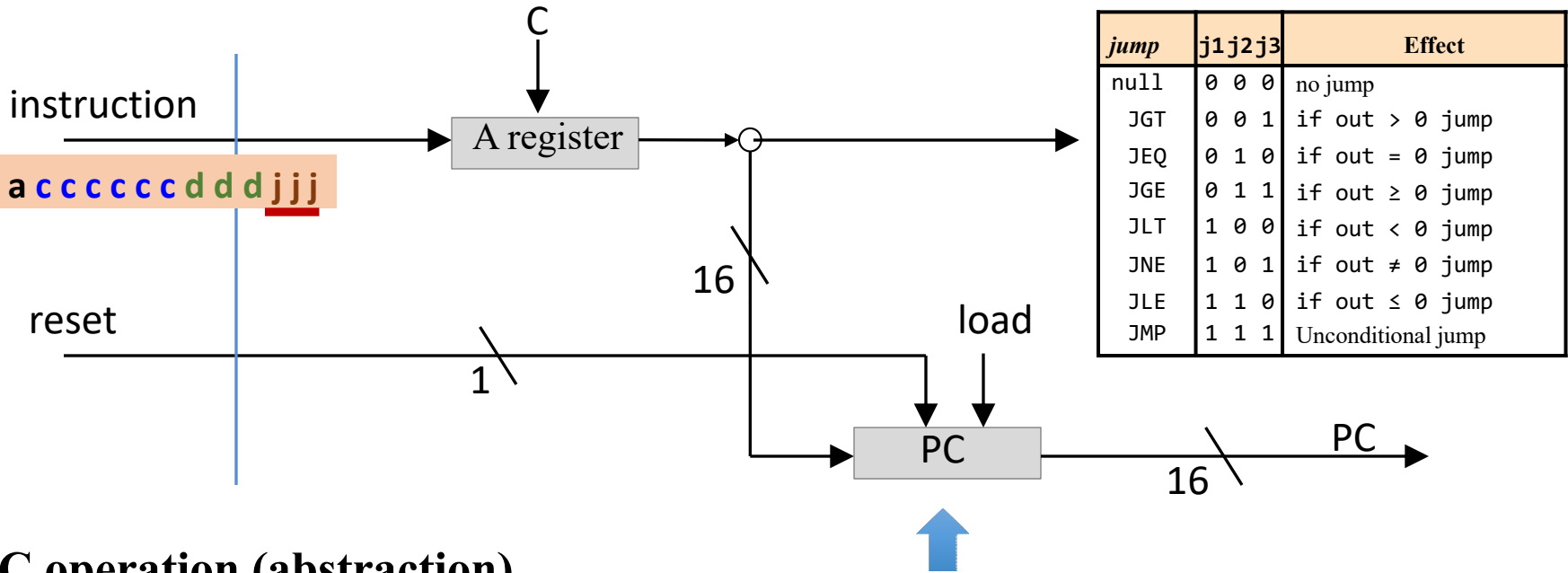
PC operation (abstraction)

- Emits the address of the next instruction

- <u>reset:</u>   PC=0

- <u>no jump:</u> **000**   PC++

- <u>goto:</u> **111**   PC=A   PC always goes to an address stored in "A" when it jumps

- <u>conditional goto:</u>   if (condition) PC= A else PC++

Control Implementation

instruction

111 a c c c c c c d d d j j j

reset

C

A register

16

1

load

16

PC

PC

16

jump	j1 j2 j3	Effect
null	0 0 0	no jump
JGT	0 0 1	if out > 0 jump
JEQ	0 1 0	if out = 0 jump
JGE	0 1 1	if out ≥ 0 jump
JLT	1 0 0	if out < 0 jump
JNE	1 0 1	if out ≠ 0 jump
JLE	1 1 0	if out ≤ 0 jump
JMP	1 1 1	Unconditional jump

address of next instruction

PC operation (implementation)

```
if (reset == 1)

    PC = 0

else

    load = f(jump bits, ALU control outputs (zr,ng))

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

    else         PC++           // next instruction
```

Instructor: Muhammad Arif Butt, Ph.D.

Load Input of PC-Register

ALU Output

ALU Output

instr[4]

loadAReg

D register

d1 d2 d3

Mux16 The value is

A register

Out M

Instrution

RAM[A]

ALU

111 a ccccccc ddd jjj

D
instr[15]

RAM[A]

inM

Mux16

A   M input

A   A register output

Write M

instr[12]

zr, ng   C

Memory address output

Address M

Reset

L E G

load

PC

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

PC

Program counter output

load = f(jump bits, zr/ng bits)

if (load == 1)

PC = A

Instructor: Muhammad Arif Butt, Ph.D.

Hack CPU Implementation

That's It!   **All that remains is to actually build it** ☺