CSE 390B, Autumn 2022

Building Academic Success Through Bottom-Up Computing

Test-taking Strategies & Midterm Practice Exam

Hack CPU Logic, Test-taking Strategies, Midterm Practice Exam, Practice Exam Walkthrough and Rubric

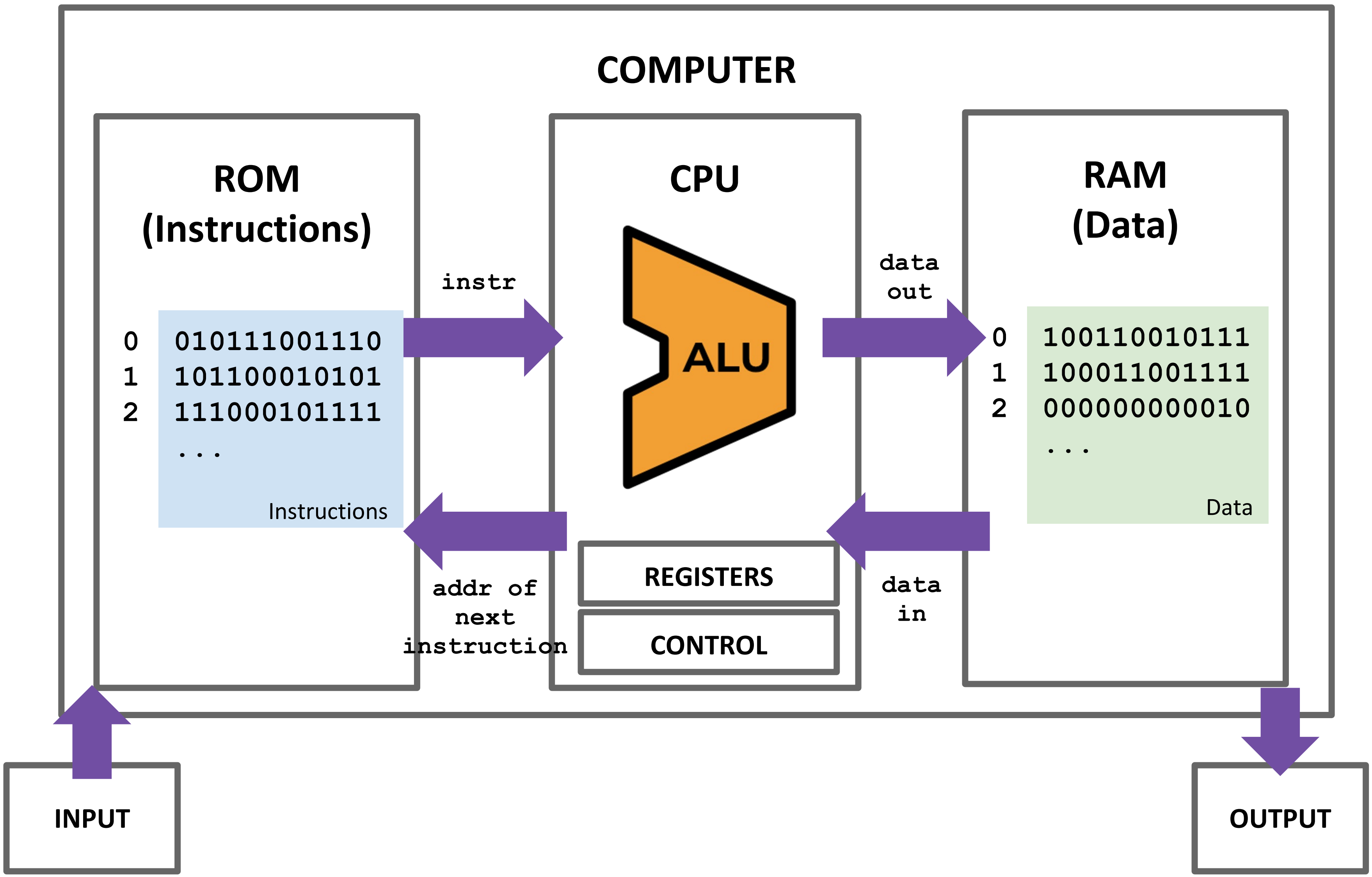
Week 7 Announcements

- CSE 390B final is Tuesday of finals (12/13), 4:30-6:20pm
  - Please let the course staff know if you have a conflicting final
- Project reminders
  - First check specification and textbook when you have a question
  - Please avoid submitting a link on the project documents
  - Please assign problems to the pages when submitting the project document on Gradescope
  - Please find a time for the Project 6 mock exam problem and update the spreadsheet (if your group has not already)
- Andres' office hours now on Mondays from 3:30-4:30pm
  - Location: CSE2 153

Lecture Outline

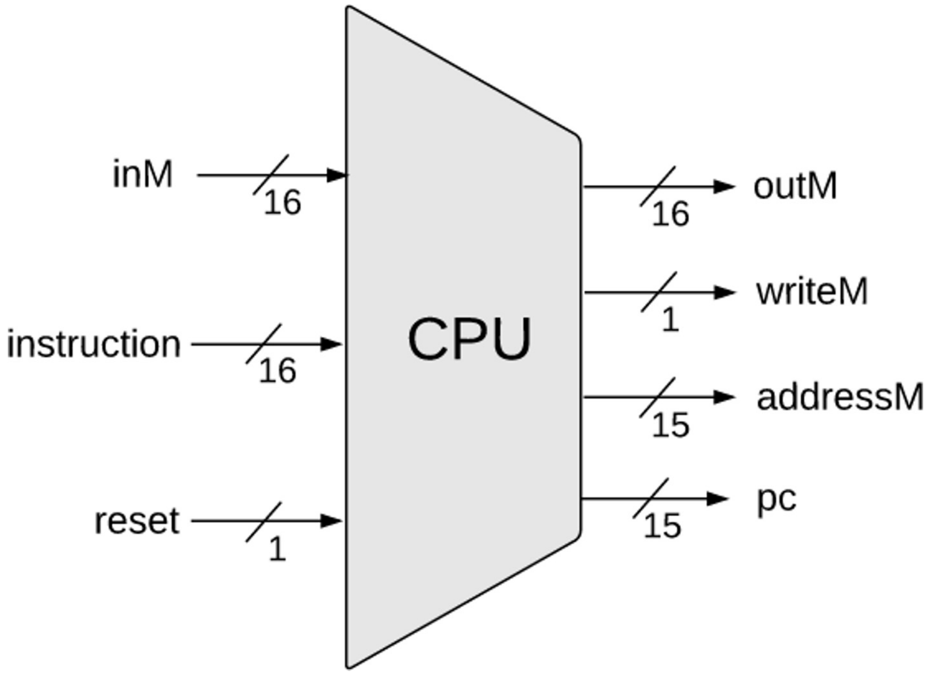
- **Hack CPU Logic**
  - **Implementation and Operations**
- Test-taking Strategies
  - Maximizing Success on Exam Day
- Midterm Practice Exam
  - Mock Exam, Debrief, and Reflection
- Practice Exam Solutions and Rubric
  - Walkthrough of Solutions and Exploring Sample Rubrics

Hack CPU


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

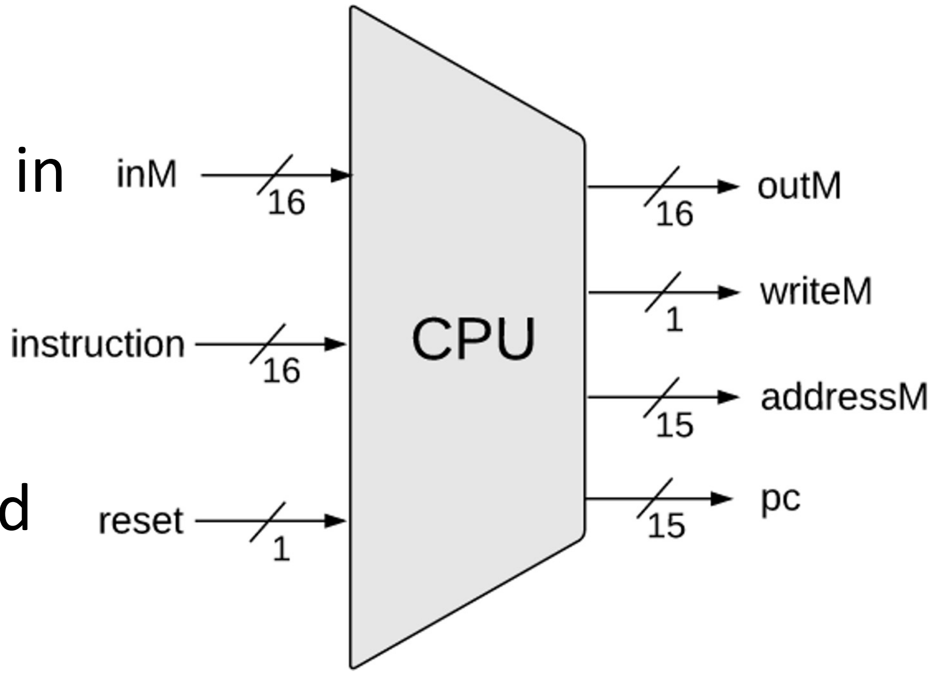
Hack CPU Interface Inputs

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

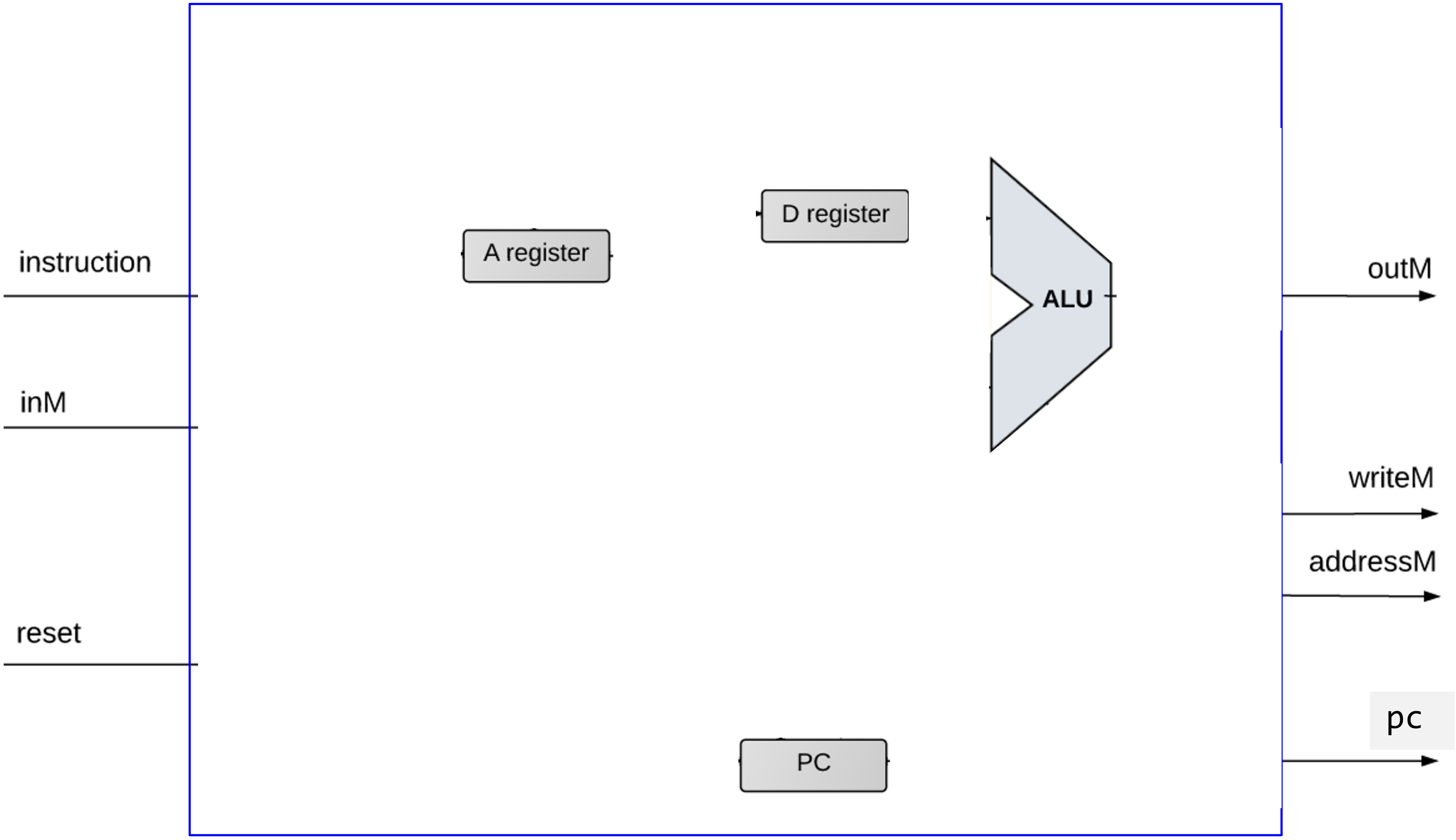


Hack CPU Interface Outputs

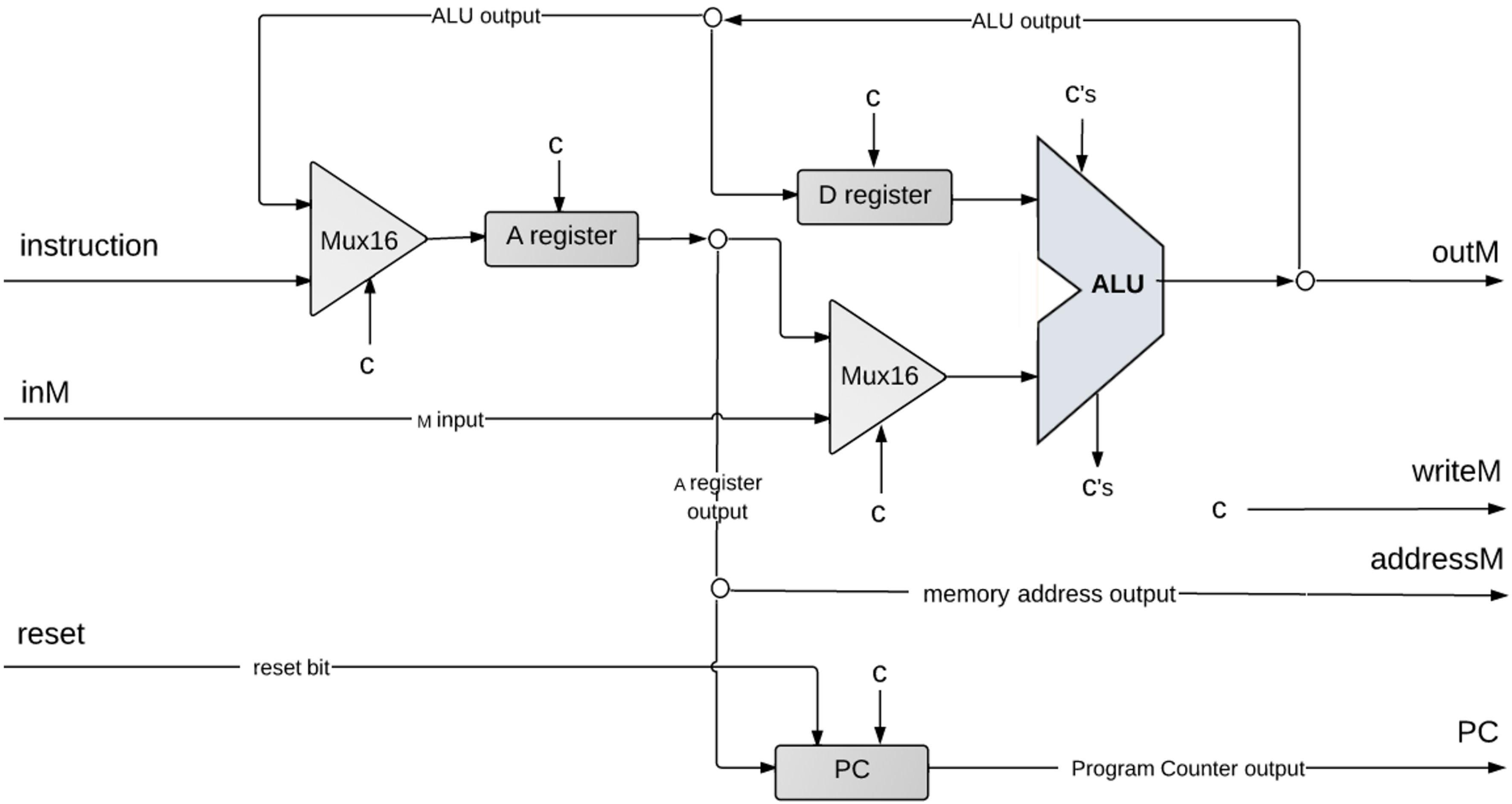
- **`outM`**: value used to update memory if writeM is 1
- **`writeM`**: if 1, update value in memory at addressM with outM
- **`addressM`**: address to read from or write to in memory
- **`pc`**: address of next instruction to be fetched from memory



Hack CPU Implementation

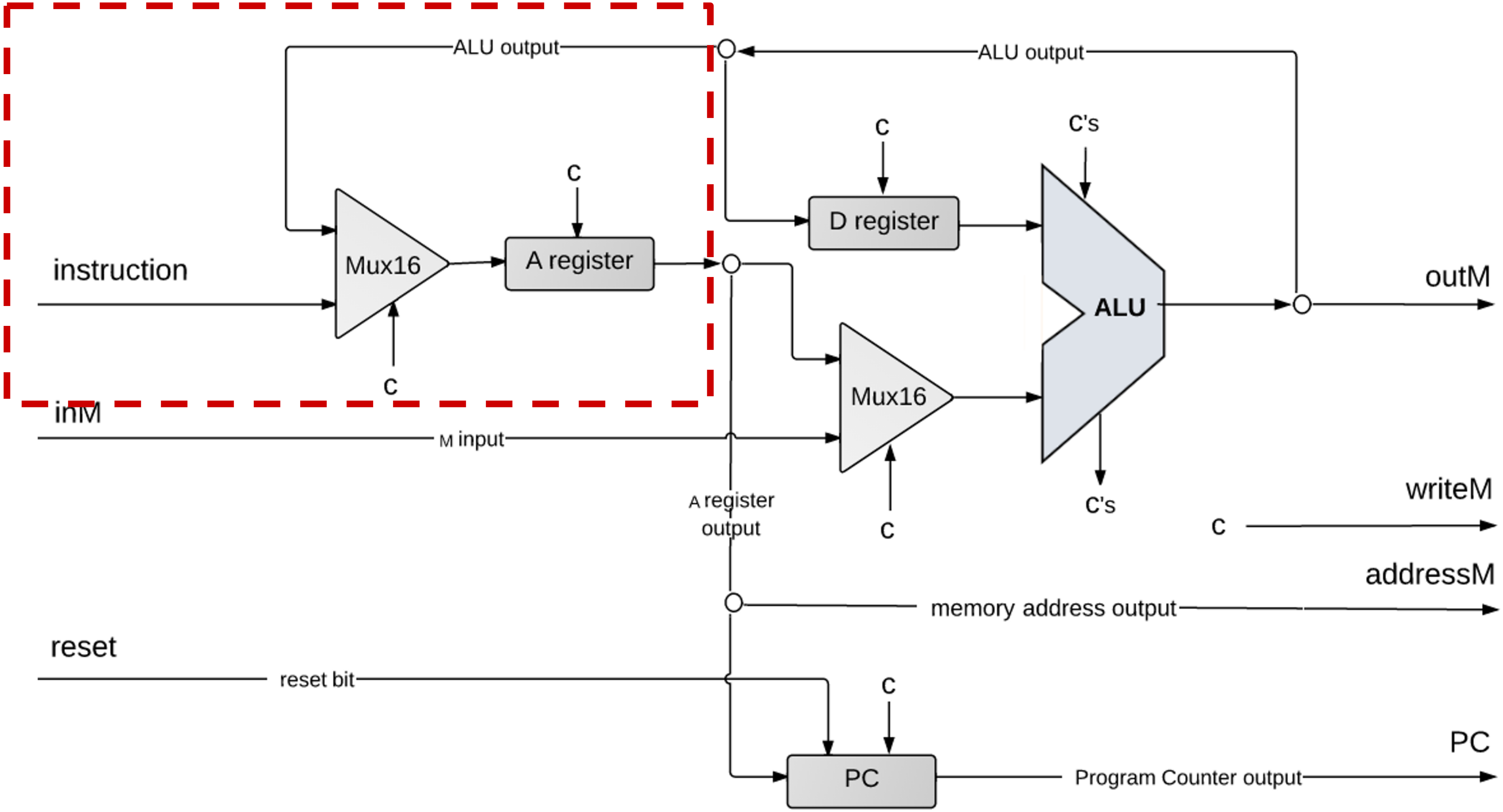


Hack CPU Implementation



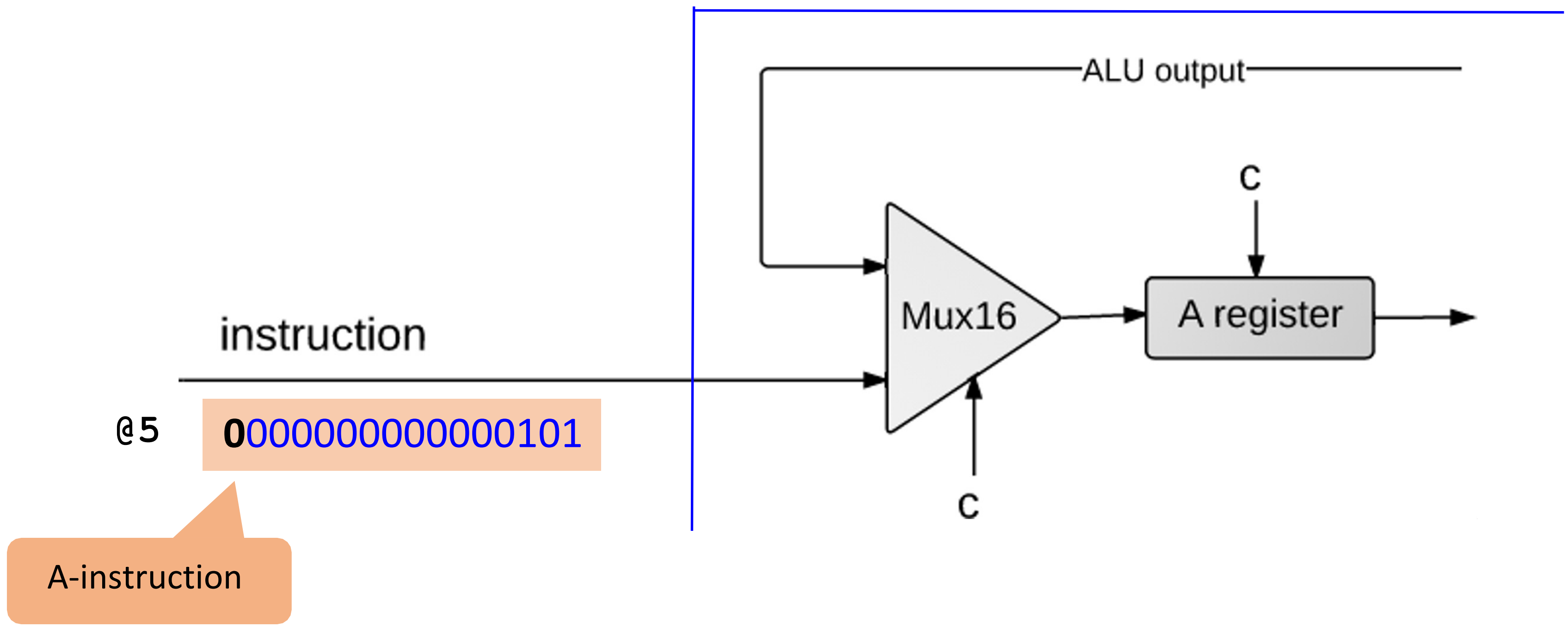
(each "C" symbol represents a control bit)

CPU Operation: Instruction Handling

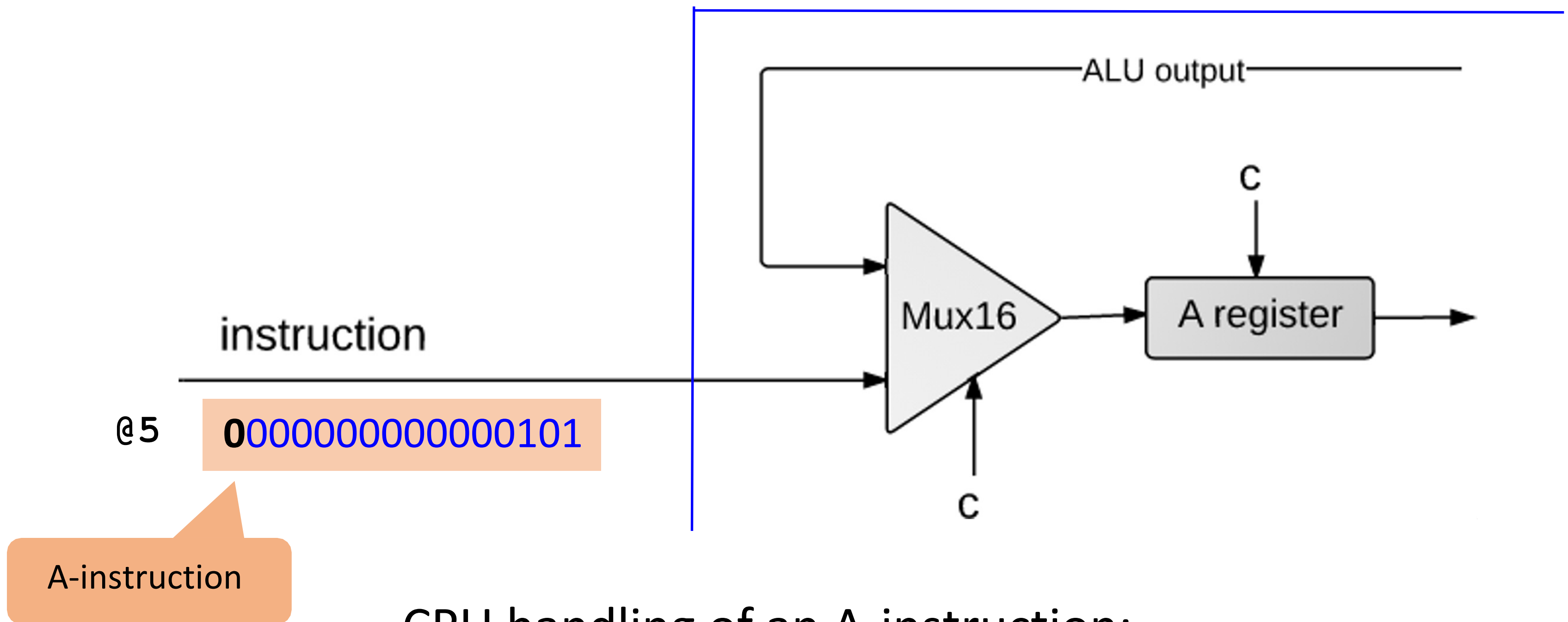


(each "C" symbol represents a control bit)

CPU Operation: Instruction Handling



CPU Operation: Instruction Handling



CPU handling of an A-instruction:

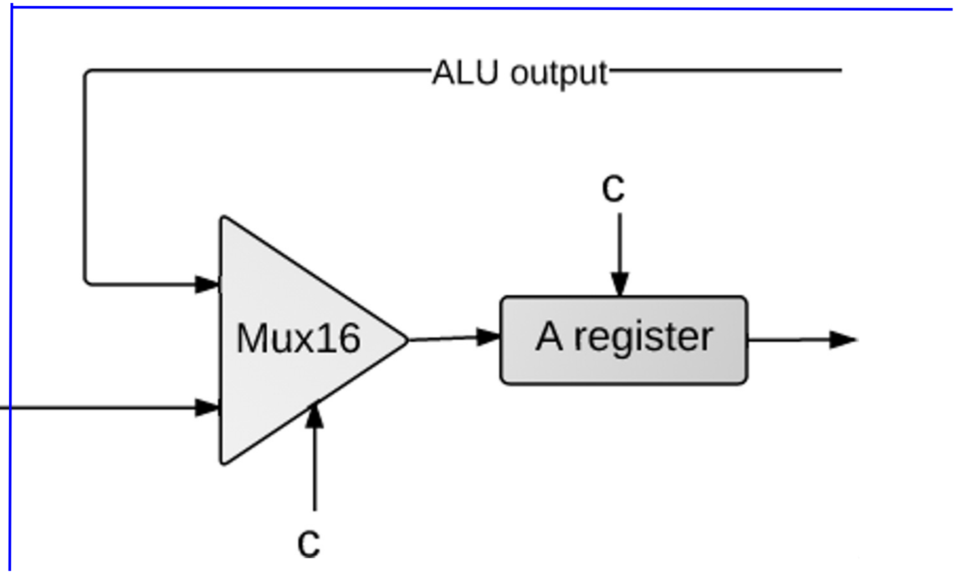
- Decodes the instruction into:
  - op-code
  - 15-bit value
- Stores the value in the A-register
- Outputs the value (not shown in this diagram)

CPU Operation: Instruction Handling

C-instruction

instruction

D=D+1;JMP 1110011111010111



Hack: C-Instructions

- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Family: C-Instruction

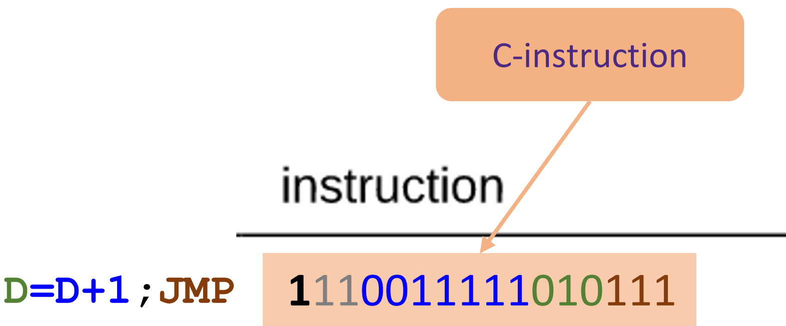
Unused

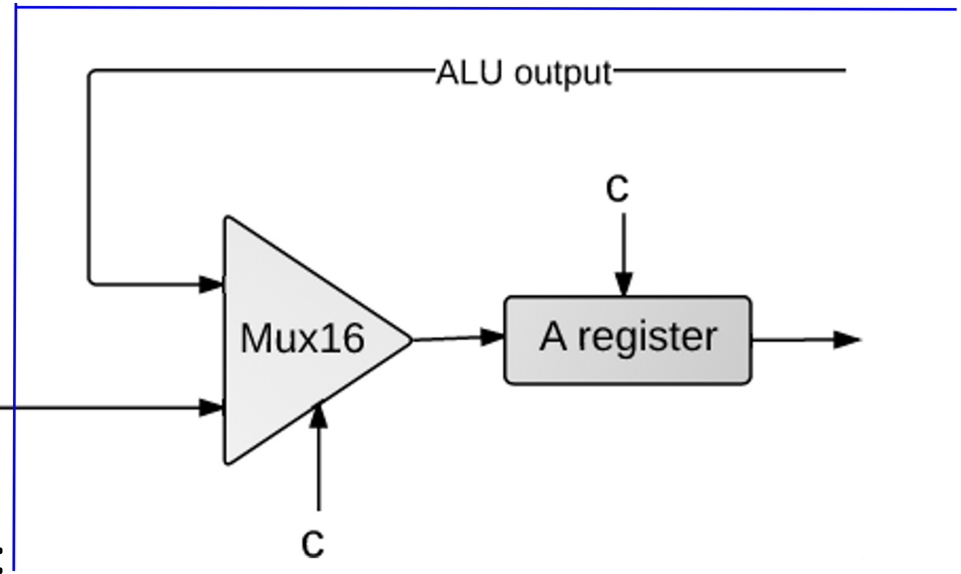
Comp: ALU Operation (a bit chooses between A and M)

Dest: Where to store result

Jump: Condition for jumping

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

Hack: C-Instructions

- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Comp:
ALU Operation (a bit chooses between A and M)

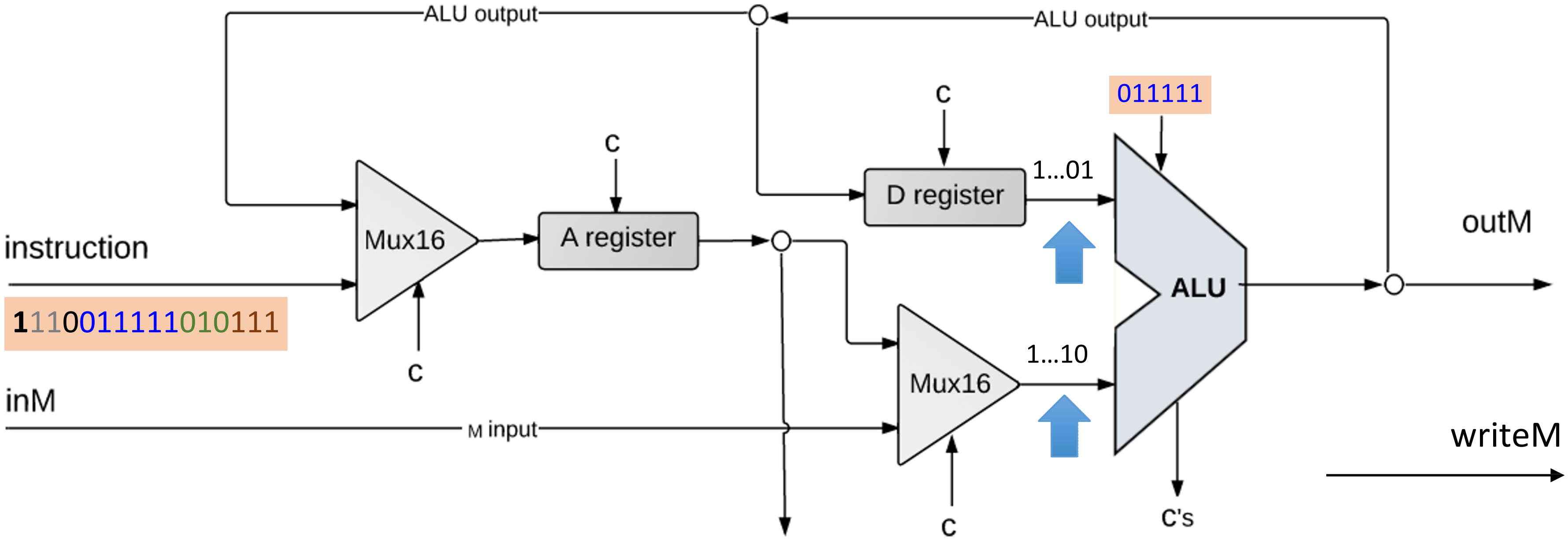
Chapter 4

(when a=0) *comp mnemonic*	c1	c2	c3	c4	c5	c6	(when a=1) *comp mnemonic*
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	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

Important: just pattern matching text!

Cannot have "**1+M**"

CPU Operation: Handling C-Instructions



<u>ALU data inputs:</u>

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

<u>ALU control inputs:</u>

- Control bits (from the instruction)

Hack: C-Instructions

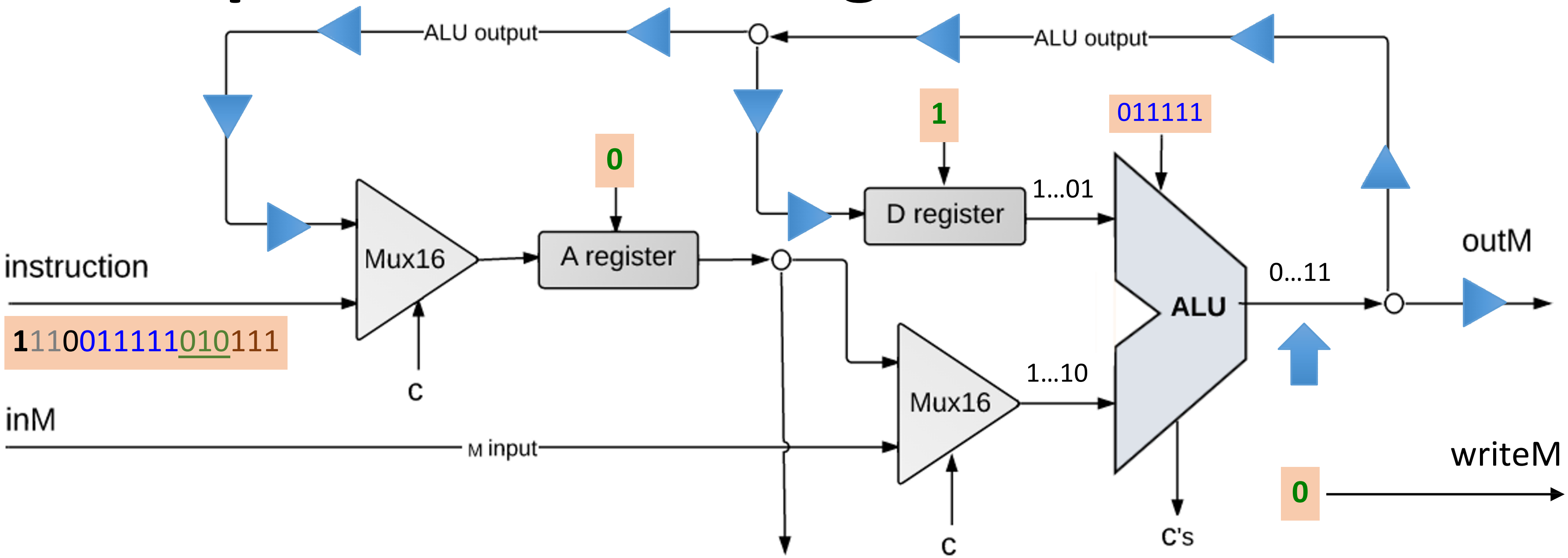
- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Dest:
Where to store result

Chapter 4

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

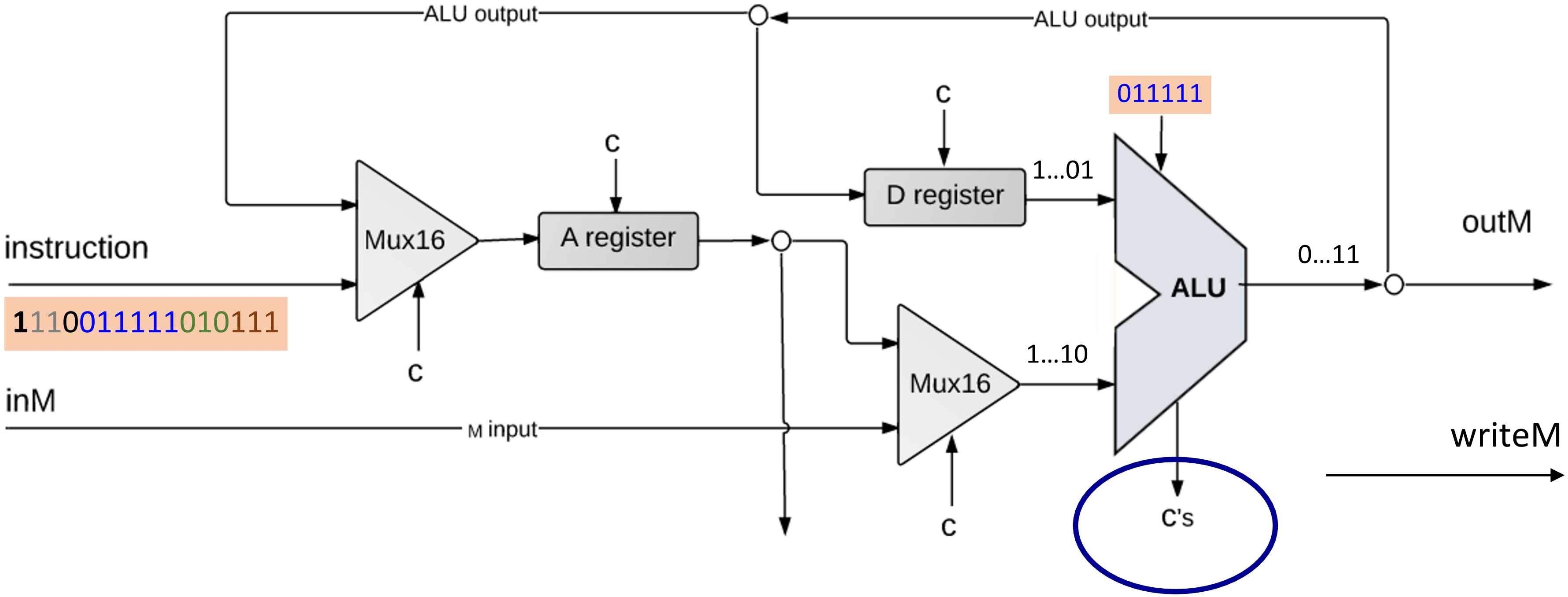
CPU Operation: Handling C-Instructions



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

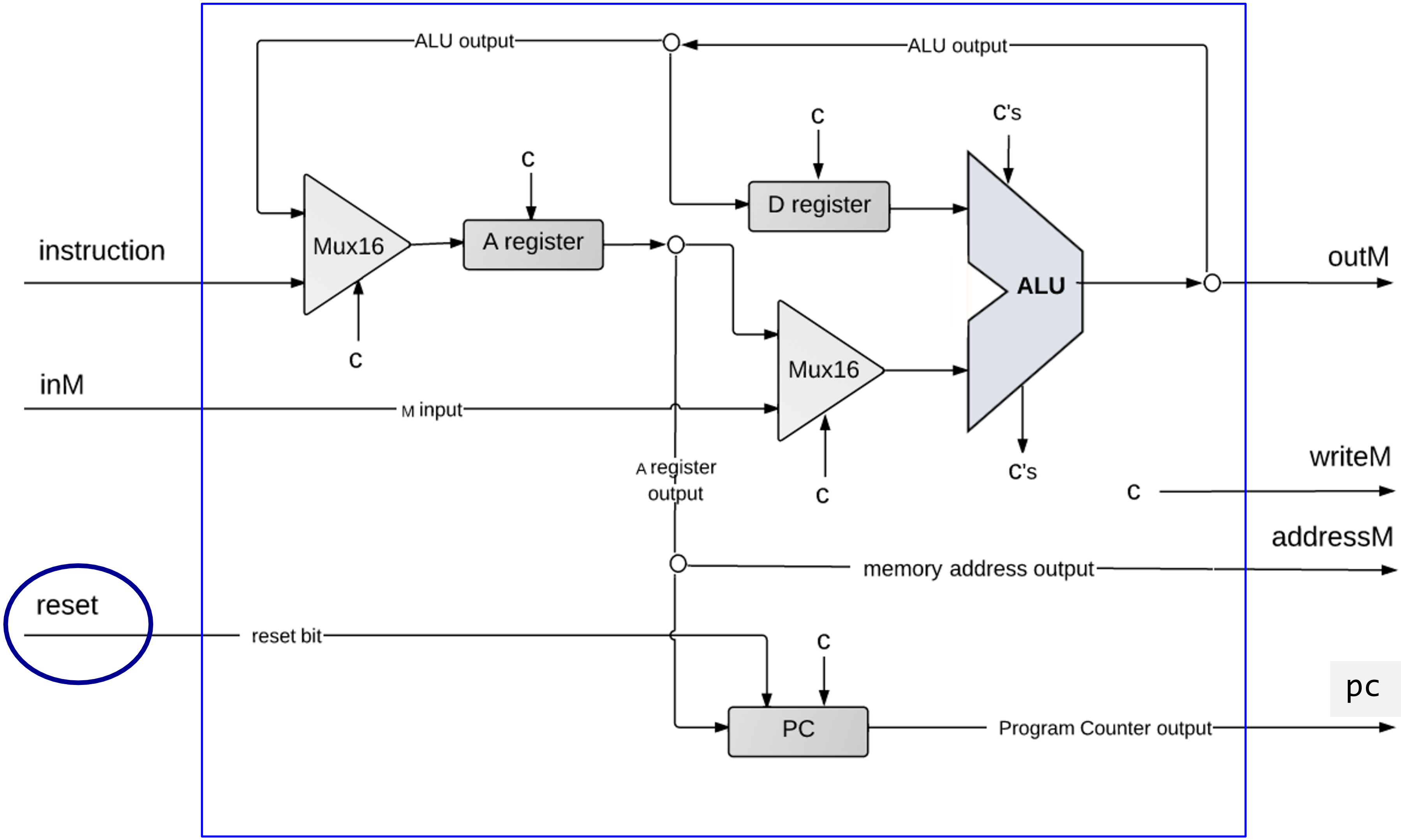
CPU Operation: Handling C-Instructions



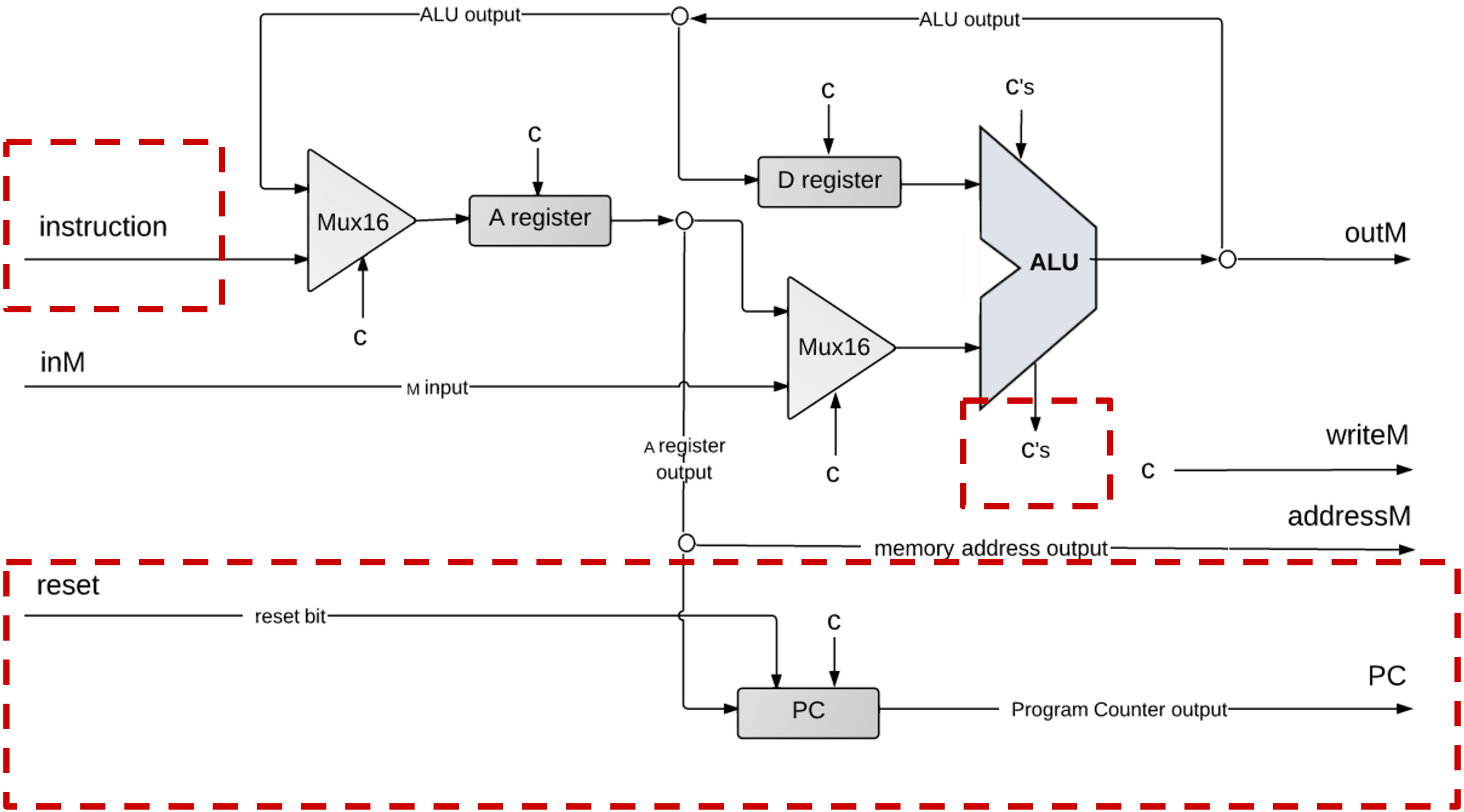
ALU control outputs:

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

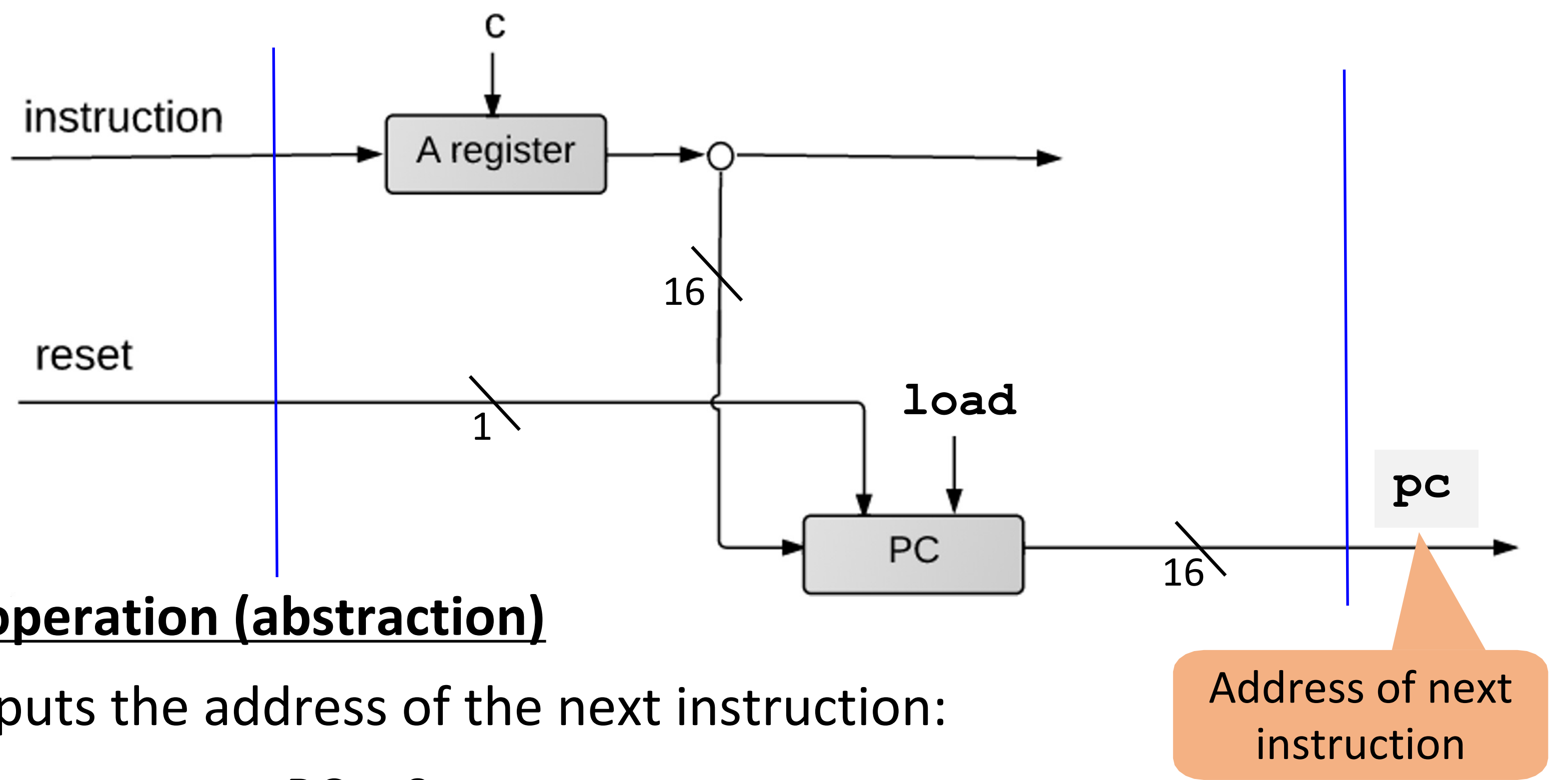
CPU Operation: Control



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 ++

Hack: C-Instructions

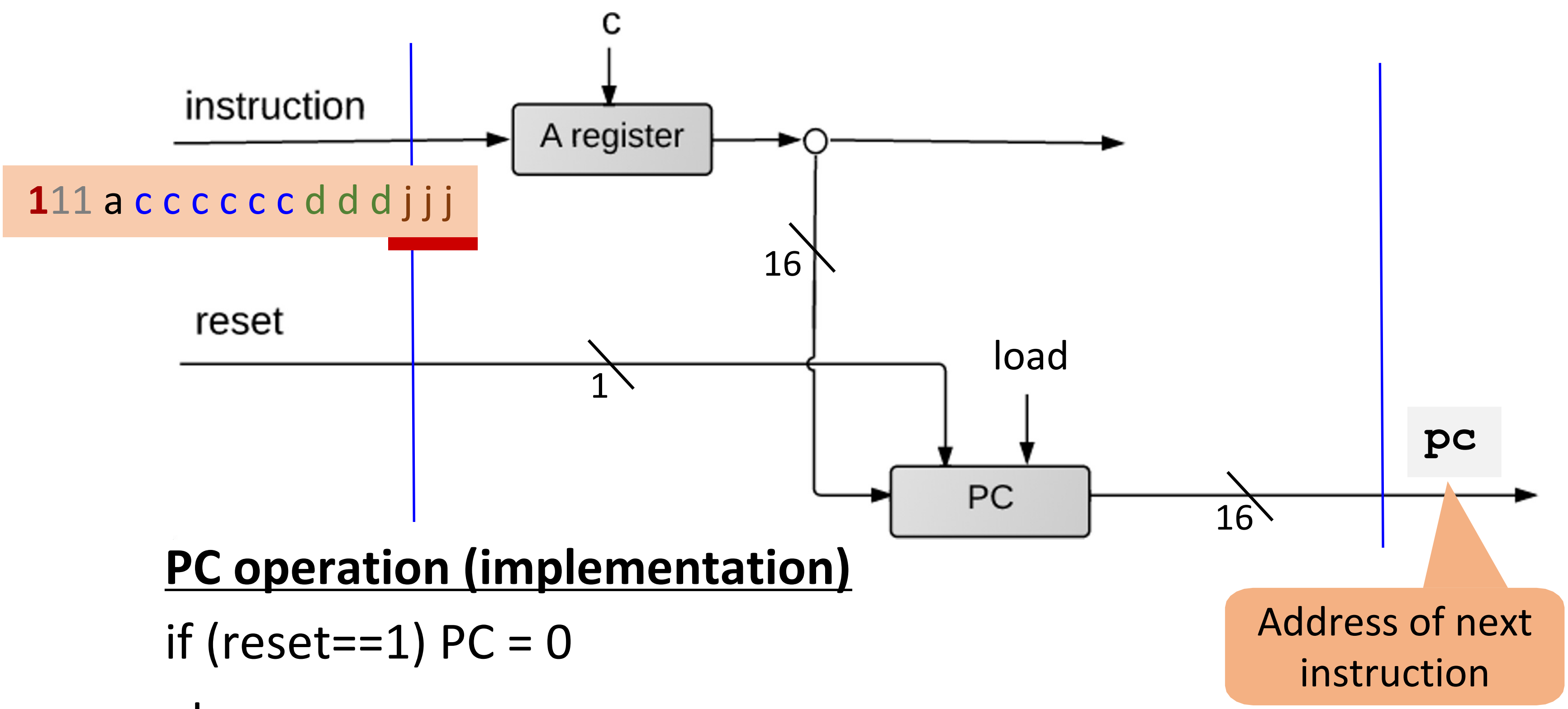
- Symbolic: `dest = comp ; jump`
- Binary: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

Jump: Condition for jumping

Chapter 4

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

CPU Operation: Control



PC operation (implementation)

if (reset==1) PC = 0

else

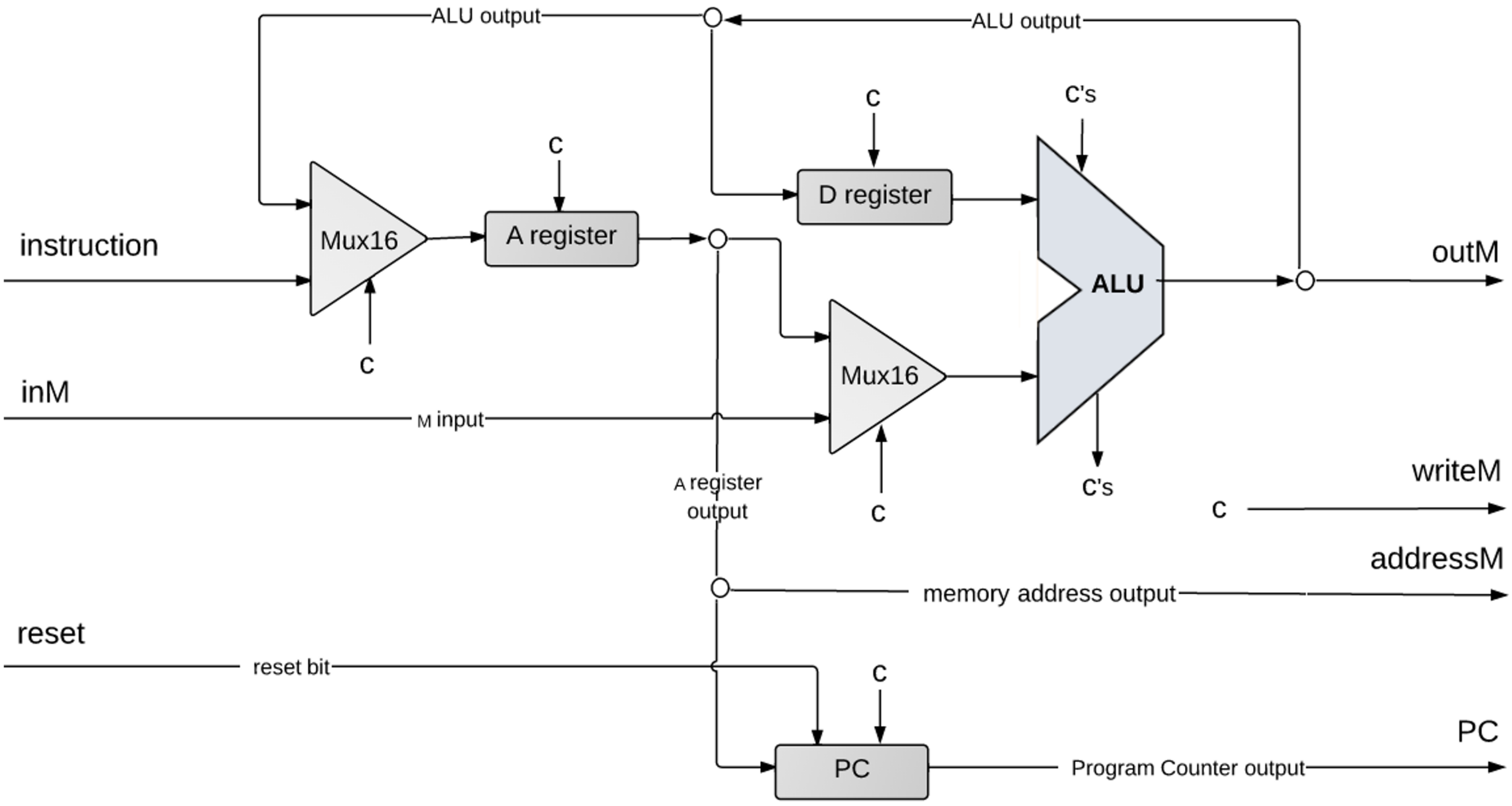
// In the course of handling the current instruction:

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

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

else PC++ // next instruction

Hack CPU Implementation: That's It!



Lecture Outline

- Hack CPU Logic
  - Implementation and Operations
- **Test-taking Strategies**
  - **Maximizing Success on Exam Day**
- Midterm Practice Exam
  - Mock Exam, Debrief, and Reflection
- Practice Exam Solutions and Rubric
  - Walkthrough of Solutions and Exploring Sample Rubrics

Test-taking Strategies Discussion

- What are some test-taking strategies you have previously utilized in taking your exams?

- Were those strategies you tried effective or not? Why?

- How might you try a new test-taking strategy on the CSE 390B midterm or any other upcoming exam?

Test-taking Strategies

- Survey the entire exam before beginning
  - Helps plan how much time to allocate for each problem
- Read exam directions and question statements carefully
  - Use highlights, underlines, circles on important details
- Answer the questions you feel the most confident in first
- If stuck on a problem, make a mark on the problem and revisit the question later

Test-taking Strategies

- Prioritize how you will answer questions
  - Do this based on confidence level for each type of question or how long you think each will take
- Rely on a methodological approach for each question
  - Helps make taking the test feel more systematic
- If stuck on a question, demonstrate what you know
  - Many exams reward partial credit
- If time allows, double check your answers
  - Catches any small mistakes that may have been made earlier

Lecture Outline

- Hack CPU Logic
  - Implementation and Operations
- Test-taking Strategies
  - Maximizing Success on Exam Day
- **Midterm Practice Exam**
  - **Mock Exam, Debrief, and Reflection**
- Practice Exam Solutions and Rubric
  - Walkthrough of Solutions and Exploring Sample Rubrics

Midterm Practice Exam

- The exam is closed-note, closed-book
  - You may only use the midterm reference sheet
- Questions are not necessarily in order of difficulty
- You will have 25 minutes to complete the mock exam
  - We will give you a 5-minute warning
- Remember to relax and take deep breaths

Mock Exam Debrief & Reflection

- What did you learn about yourself through this process? About your test-taking practices?

- What are two test-taking strategies that you would like to engage with in your next exam? Why?

- What is one thing that can help you relax and calm down before or during your exam?

Lecture Outline

- Hack CPU Logic
  - Implementation and Operations
- Test-taking Strategies
  - Maximizing Success on Exam Day
- Midterm Practice Exam
  - Mock Exam, Debrief, and Reflection
- **Practice Exam Solutions and Rubric**
  - **Walkthrough of Solutions and Exploring Sample Rubrics**

Question 1: Circuit Design

Part a: Truth Table

A_t	B_t	A_{t+1}	B_{t+1}
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

11 → 10 → 01 → 00 → 11

Question 1: Circuit Design

Part a: Truth Table

A_t	B_t	A_{t+1}	B_{t+1}
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

11 → 10 → 01 → 00 → 11

Question 1: Circuit Design

Part a: Truth Table

A_t	B_t	A_{t+1}	B_{t+1}
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

$A_{t+1} = (A_t \ \& \ B_t)$

$A_{t+1} = (\sim A_t \ \& \sim B_t)$

Part b: Boolean Expressions

Question 1: Circuit Design

Part a: Truth Table

A_t	B_t	A_{t+1}	B_{t+1}
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

$A_{t+1} = (A_t \,\&\, B_t)$

$A_{t+1} = (\sim A_t \,\&\, \sim B_t)$

Part b: Boolean Expressions

$A_{t+1} = (A_t \,\&\, B_t) \mid (\sim A_t \,\&\, \sim B_t)$

Question 1: Circuit Design

Part a: Truth Table

A_t	B_t	A_{t+1}	B_{t+1}
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

$A_{t+1} = (A_t \,\&\, B_t)$

$B_{t+1} = (A_t \,\&\, \sim B_t)$

$A_{t+1} = (\sim A_t \,\&\, \sim B_t)$ $B_{t+1} = (\sim A_t \,\&\, \sim B_t)$

Part b: Boolean Expressions

$A_{t+1} = (A_t \,\&\, B_t) \mid (\sim A_t \,\&\, \sim B_t)$

$B_{t+1} = (A_t \,\&\, \sim B_t) \mid (\sim A_t \,\&\, \sim B_t)$

$= \sim B_t \,\&\, (A_t \mid \sim A_t)$ [Factor out $\sim B_t$]

$= \sim B_t$ [$A_t \mid \sim A_t = 1$]

Question 1: Circuit Design

Part a: Truth Table

A_t	B_t	A_{t+1}	B_{t+1}
1	1	1	0
1	0	0	1
0	1	0	0
0	0	1	1

Part c: Drawing the Circuit

Part b: Boolean Expressions

$$A_{t+1} = (A_t \,\&\, B_t) \mid (\sim A_t \,\&\, \sim B_t)$$

$$B_{t+1} = (A_t \,\&\, \sim B_t) \mid (\sim A_t \,\&\, \sim B_t)$$

$$= \sim B_t \,\&\, (A_t \mid \sim A_t) \qquad [\text{Factor out } \sim B_t]$$

$$= \sim B_t \qquad [A_t \mid \sim A_t = 1]$$

Question 1: Circuit Design Sample Rubric

Category	**Points**	**Criteria**
Truth Table	4 points	1 point for each row in the truth table that is correct
Boolean Expressions	6 points	❖ 4 points for correct expression for A_{t+1} ▪ 2 points if truth table is wrong but expression matches truth table ❖ 2 points for correct expression for B_{t+1} ▪ 1 point if truth table is wrong but expression matches truth table
Circuit Diagram	5 points	❖ 3 points for having circuits that match the Boolean expressions in part b ❖ 2 points for fully correct diagram
Total	**15 points**	

Question 2: Math Puzzle

Dana needs 300 pickets for her colorful picket fence. She wants equal amounts of each of her 4 selected colors. She already has 32 red, 26 green, 9 yellow, and no blue. If the pickets cost 25 cents and you get 20% off if you purchase 50 or more of the same color, and 30% off if you purchase 60 or more of one color, how much does Dana need to spend? List your answer to two decimal places. You may use a calculator application on your computer to solve this problem.

Question 2: Math Puzzle

Dana needs 300 pickets for her colorful picket fence. She wants equal amounts of each of her 4 selected colors. She already has 32 red, 26 green, 9 yellow, and no blue. If the pickets cost 25 cents and you get 20% off if you purchase 50 or more of the same color, and 30% off if you purchase 60 or more of one color, how much does Dana need to spend? List your answer to two decimal places. You may use a calculator application on your computer to solve this problem.

Solution

$$75 - 32 = 43\ red$$
$$75 - 26 = 49\ green$$
$$75 - 9 = 66\ yellow$$
$$75 - 0 = 75\ blue$$

$$43 \times 0.25 + 49 \times 0.25 + 0.7 \times 66 \times 0.25 + 0.7 \times 75 \times 0.25$$
$$= \$47.675$$
$$= \$47.68$$ (Rounding down is also acceptable)

Question 3: Hack Assembly Programming

Write a Hack assembly program that stores -1, 0, or 1 in R1 based on the sign of R0. To be more specific, your program should store a -1 in R1 if R0 is negative, a 0 in R1 if R0 is 0, and a 1 in R1 if R0 is positive.

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

Question 3: Hack Assembly Programming

Write a Hack assembly program that stores -1, 0, or 1 in R1 based on the sign of R0. To be more specific, your program should store a -1 in R1 if R0 is negative, a 0 in R1 if R0 is 0, and a 1 in R1 if R0 is positive.

Equivalent pseudocode:

```
if (R0 < 0){
        R1 = -1;
} else if (R0 > 0){
        R1 = 1;
} else {  //R0 == 0
        R1 = 0;
}
```

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

Question 3: Hack Assembly Programming

Write a Hack assembly program that stores -1, 0, or 1 in R1 based on the sign of R0. To be more specific, your program should store a -1 in R1 if R0 is negative, a 0 in R1 if R0 is 0, and a 1 in R1 if R0 is positive.

Equivalent pseudocode:

```
if (R0 < 0){
        R1 = -1;
} else if (R0 > 0){
        R1 = 1;
} else {  //R0 == 0
        R1 = 0;
}
```

j1 ($out < 0$)	j2 ($out = 0$)	j3 ($out > 0$)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If $out > 0$ jump
0	1	0	JEQ	If $out = 0$ jump
0	1	1	JGE	If $out \geq 0$ jump
1	0	0	JLT	If $out < 0$ jump
1	0	1	JNE	If $out \neq 0$ jump
1	1	0	JLE	If $out \leq 0$ jump
1	1	1	JMP	Jump

One solution:

```
    @R0
    D = M
    @NEGATIVE
    D; JLT
    @POSITIVE
    D; JGT
    // R0 == 0 case
    @R1
    M = 0
    @END
    0; JMP
(NEGATIVE)
    // R0 < 0 case
    @R1
    M = -1
    @END
    0; JMP
(POSITIVE)
    // R0 > 0 case
    @R1
    M = 1
(END)
    @END
    0; JMP
```

Question 3: Hack Assembly Sample Rubric

Category	Points	Criteria
Has Infinite End Loop	1 point	❖ 1 point if program has an Infinite End Loop
Conditional Checks	4 points	❖ 2 points for having at least two checks for cases. Almost all solutions will need a check for 2 of the three cases (negative, zero, positive). ❖ 2 points for correctly matching jump condition to cases (e.g. jump to negative case when negative, etc.)
Assigns Correct R1 Value	3 points	❖ One point for each case: ▪ negative: R1 = -1 ▪ zero: R1 = 0 ▪ positive: R = 1
Fully Correct Implementation	2 points	❖ Covers any little mistakes that may result in an incorrect implementation (e.g., forgetting to jump to the end when a case is done)
Total	**10 points**	

Question 4: Metacognitive Skills

- Name two metacognitive skills that we have covered in CSE 390B thus far.

Post-Lecture 12 Reminders

- CSE 390B midterm this Thursday (11/10) during lecture at 2:30pm
- Project 6: Mock Exam Problem & Building a Computer due next Thursday (11/17) at 11:59pm
- Preston has office hours after class in CSE2 153
  - Feel free to post your questions on the Ed board as well