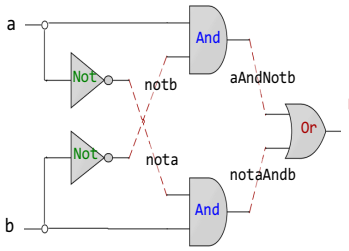
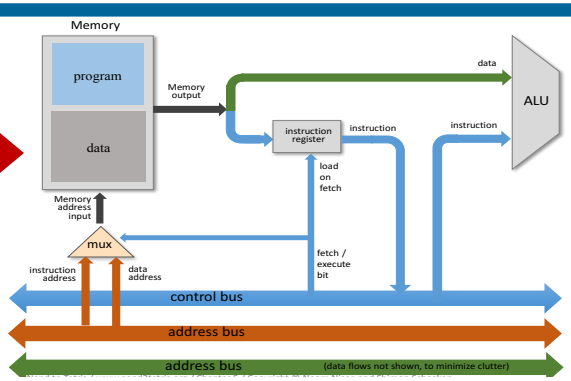
Computer Organization & Assembly Language Programming




```
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);
}
```




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

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 23

Data Path of Hack CPU - II

```
#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
```

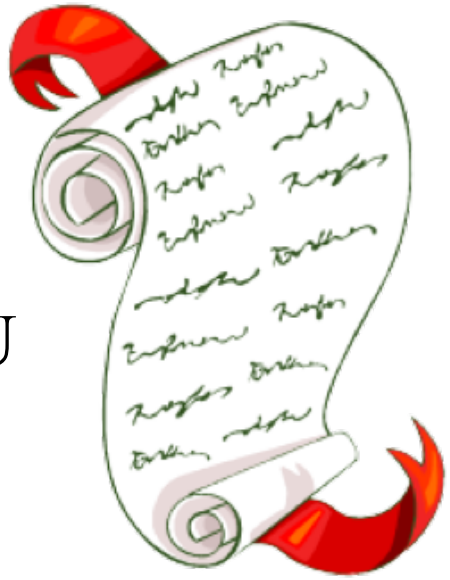
```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

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

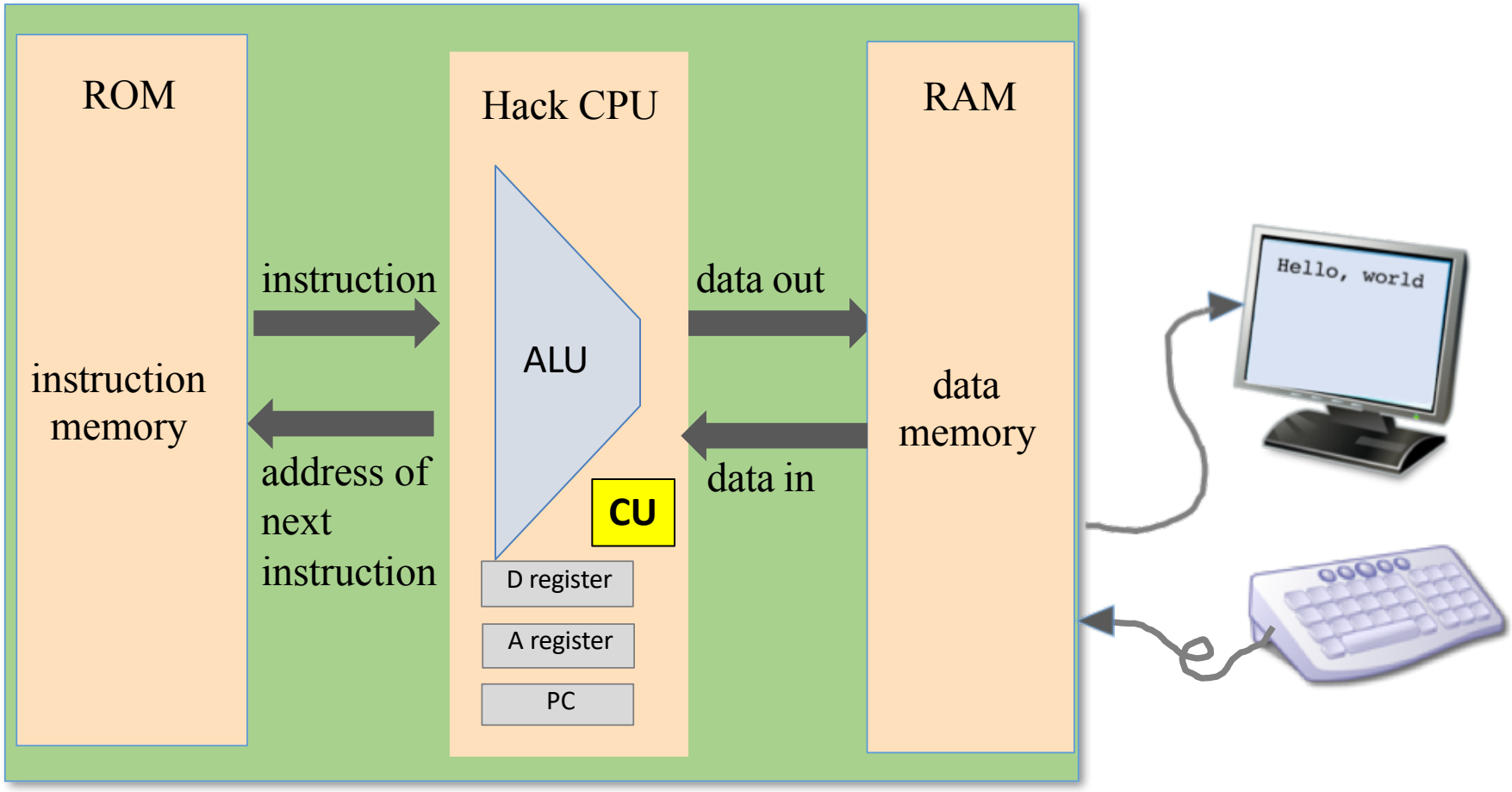
Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

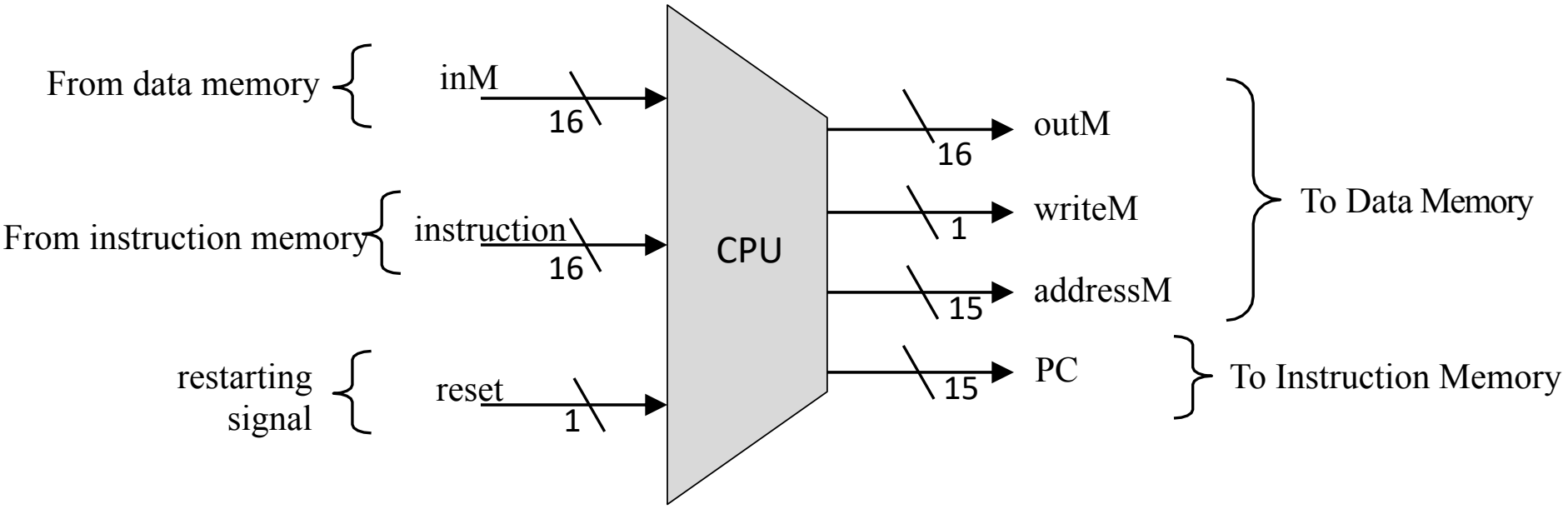
- Review of Hack Computer Architecture
- Hack CPU Interface
- Hack CPU Implementation
- Input/output and Operations of Hack ALU
- Control Logic of Hack CPU



Recap: Hack Computer Architecture

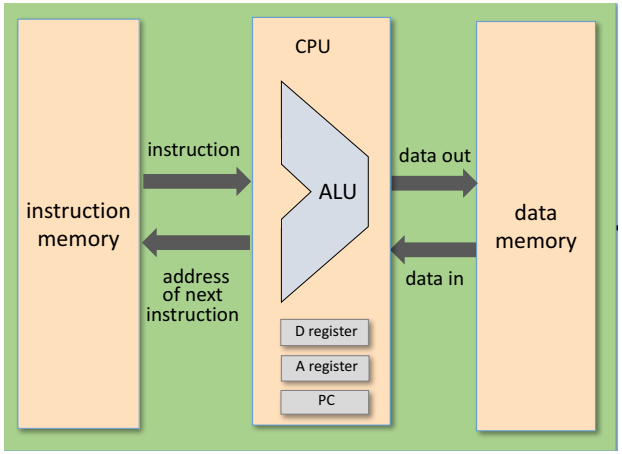


Hack CPU Interface

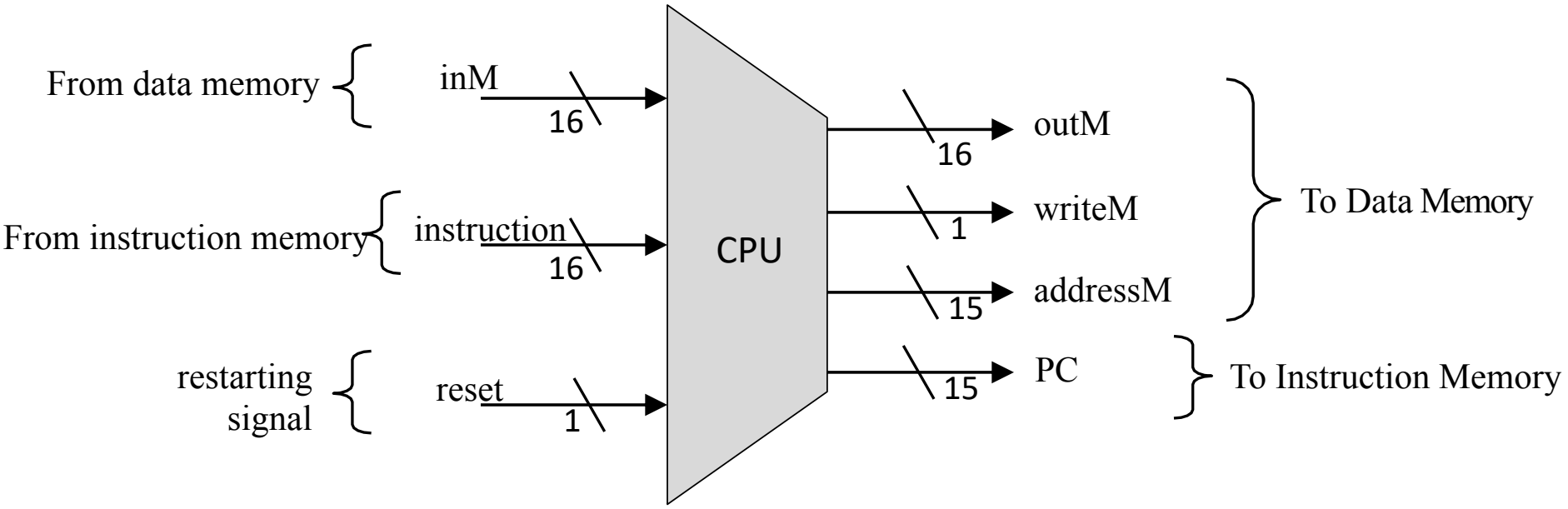


Inputs:

- Data Value
- Instruction
- Reset Bit

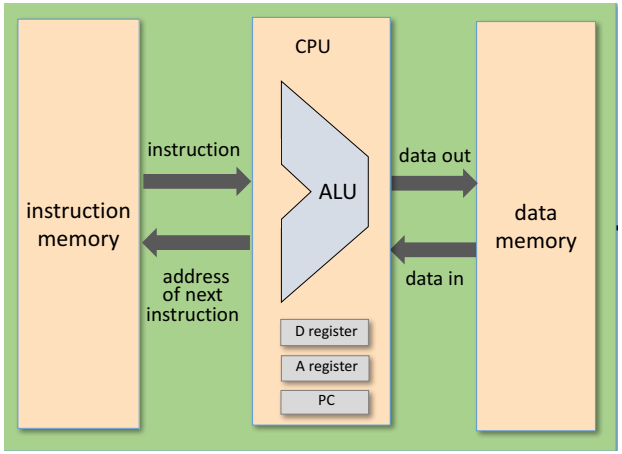


Hack CPU Interface



Outputs:

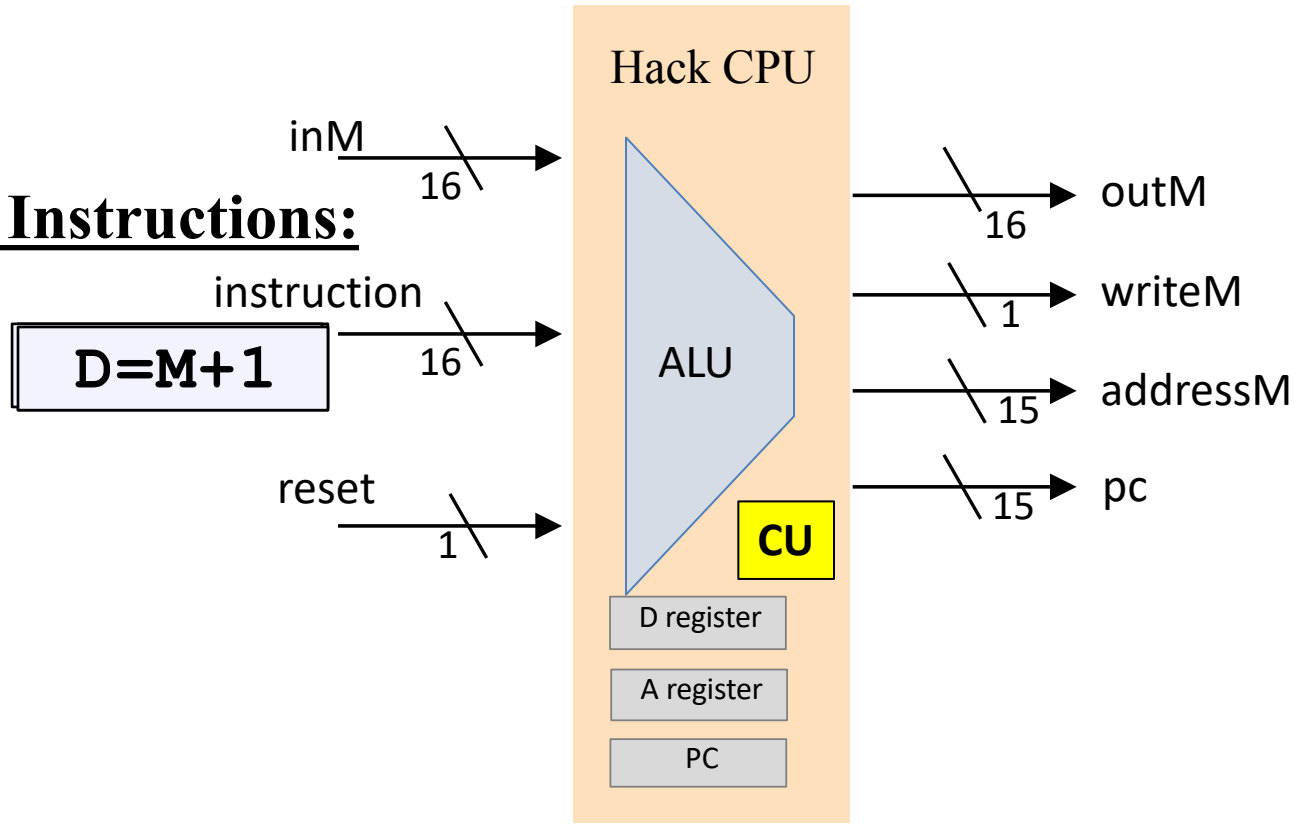
- Data Value
- Write to Memory? (yes/no)
- Memory Address
- Address of next instruction



Execution of Instructions by Hack CPU

Hack Instructions:

D=M+1



Execution of Instructions by Hack CPU

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
```

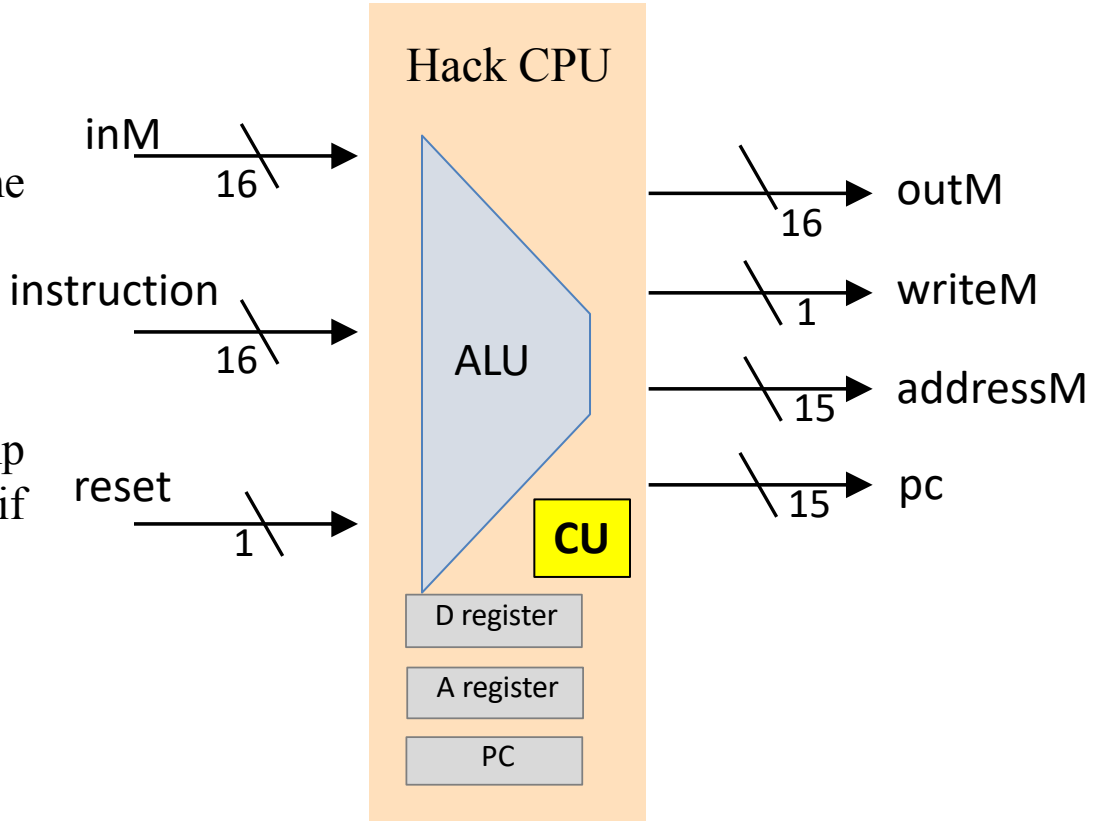
If (D-1==0)

PC is set to the value of the A-register

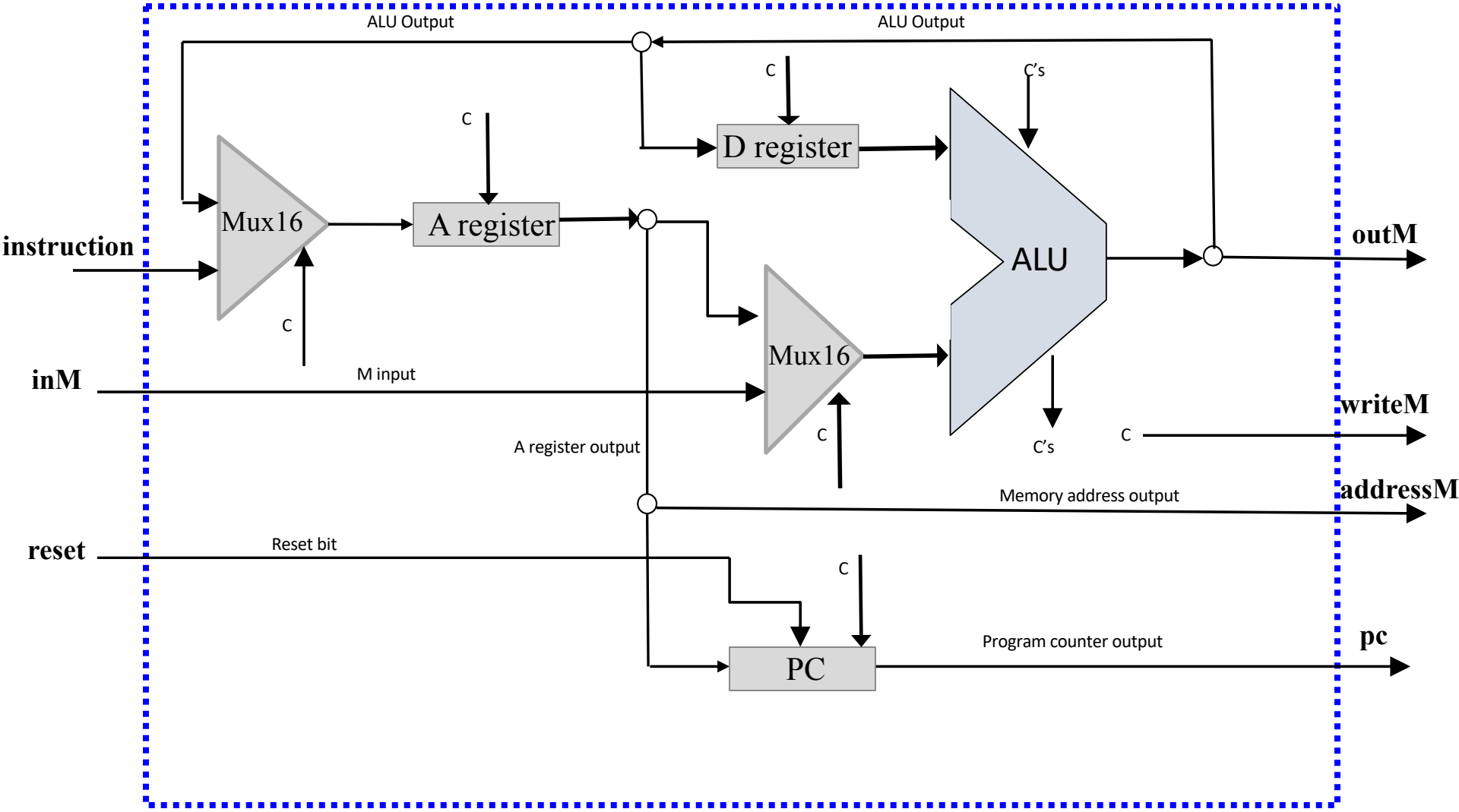
else

PC++

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

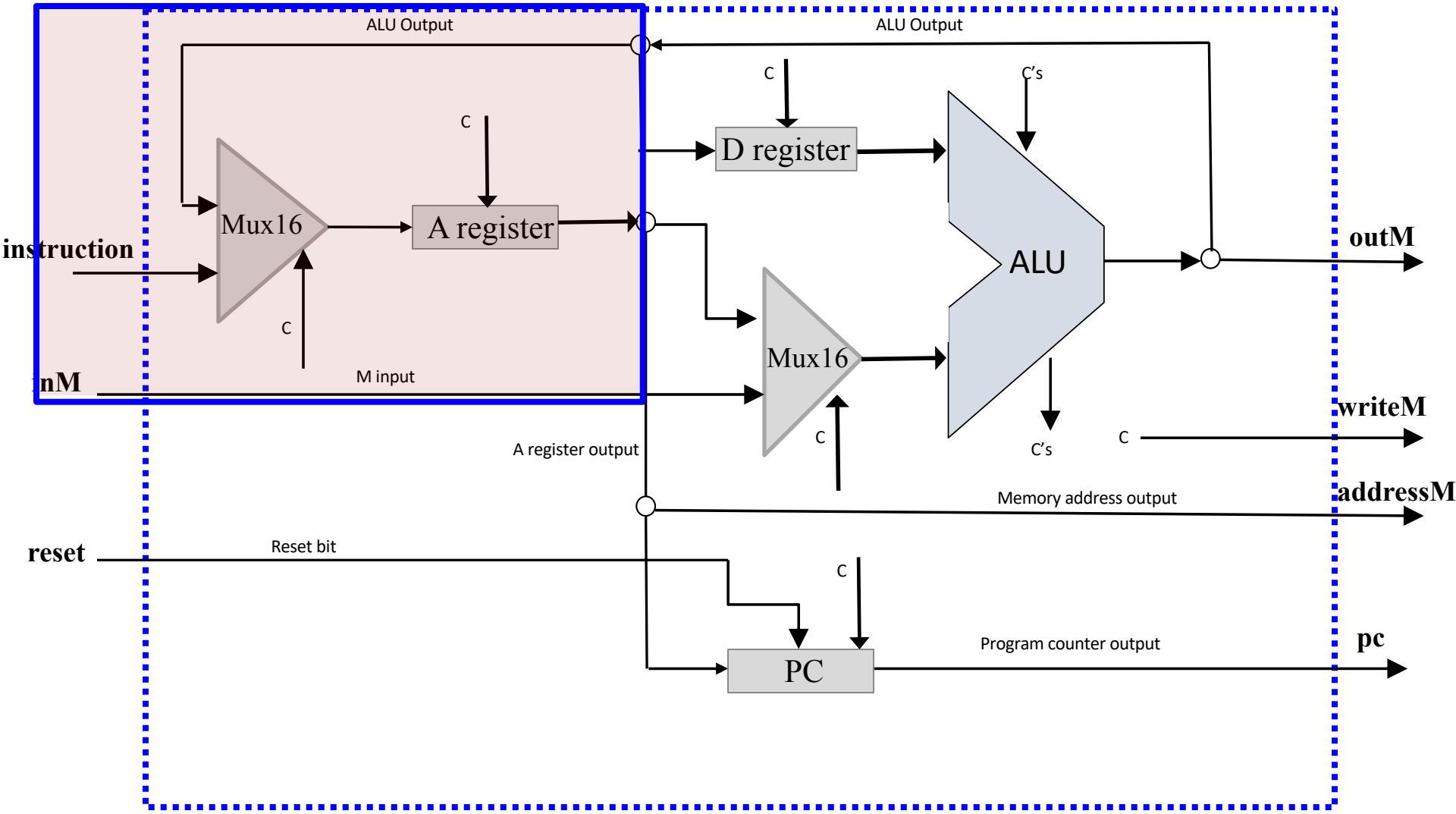


Hack CPU Implementation

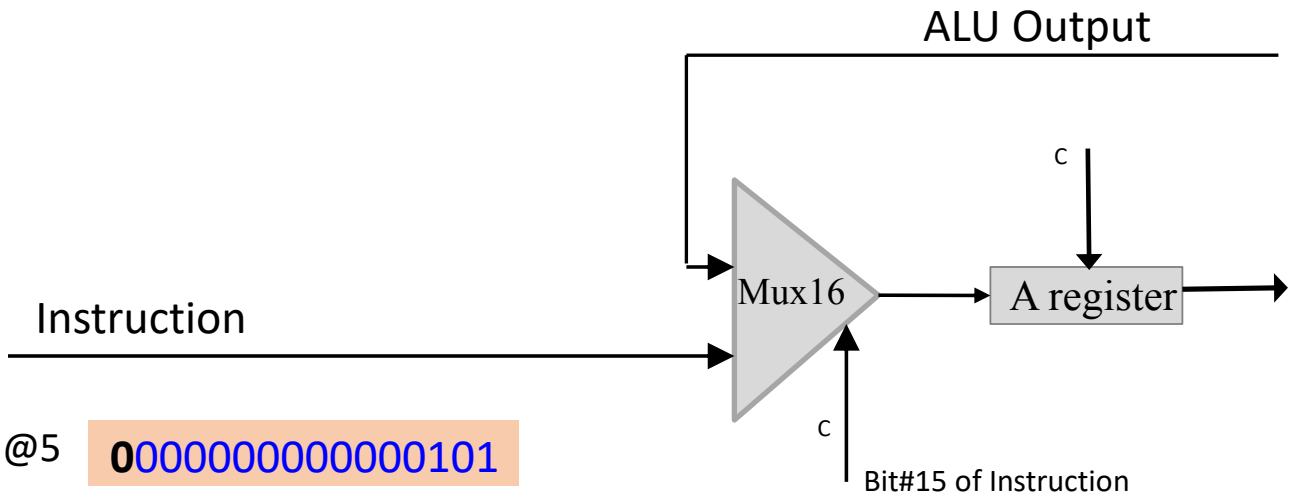


(each "C" symbol represents a control bit)

Instruction Handling



Handling A-Instruction



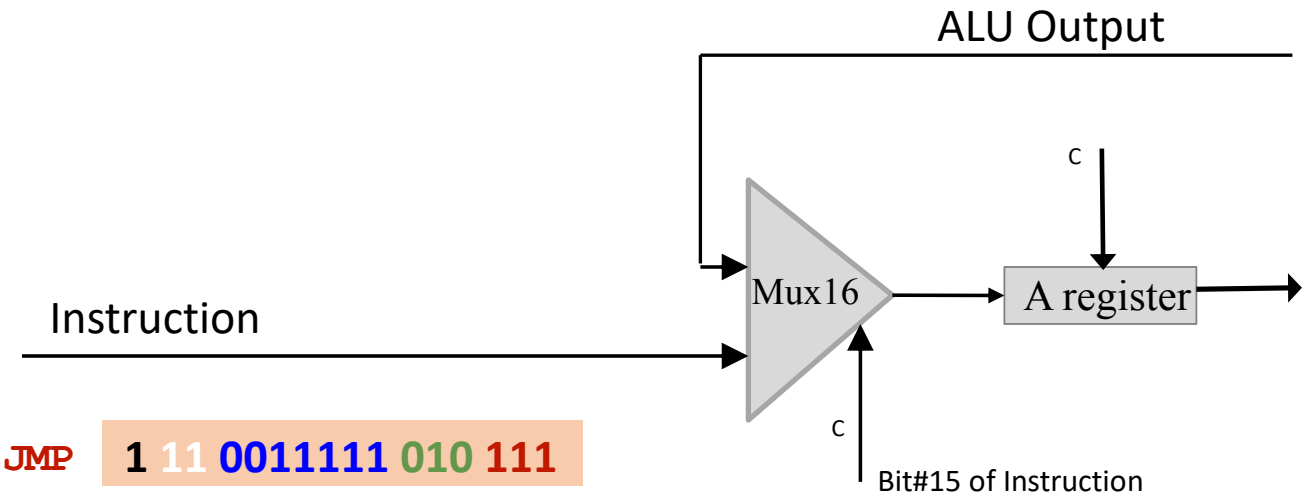
CPU handling of an A-instruction:

- 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



D= D+1 ;JMP **1 11 0011111 010 111**

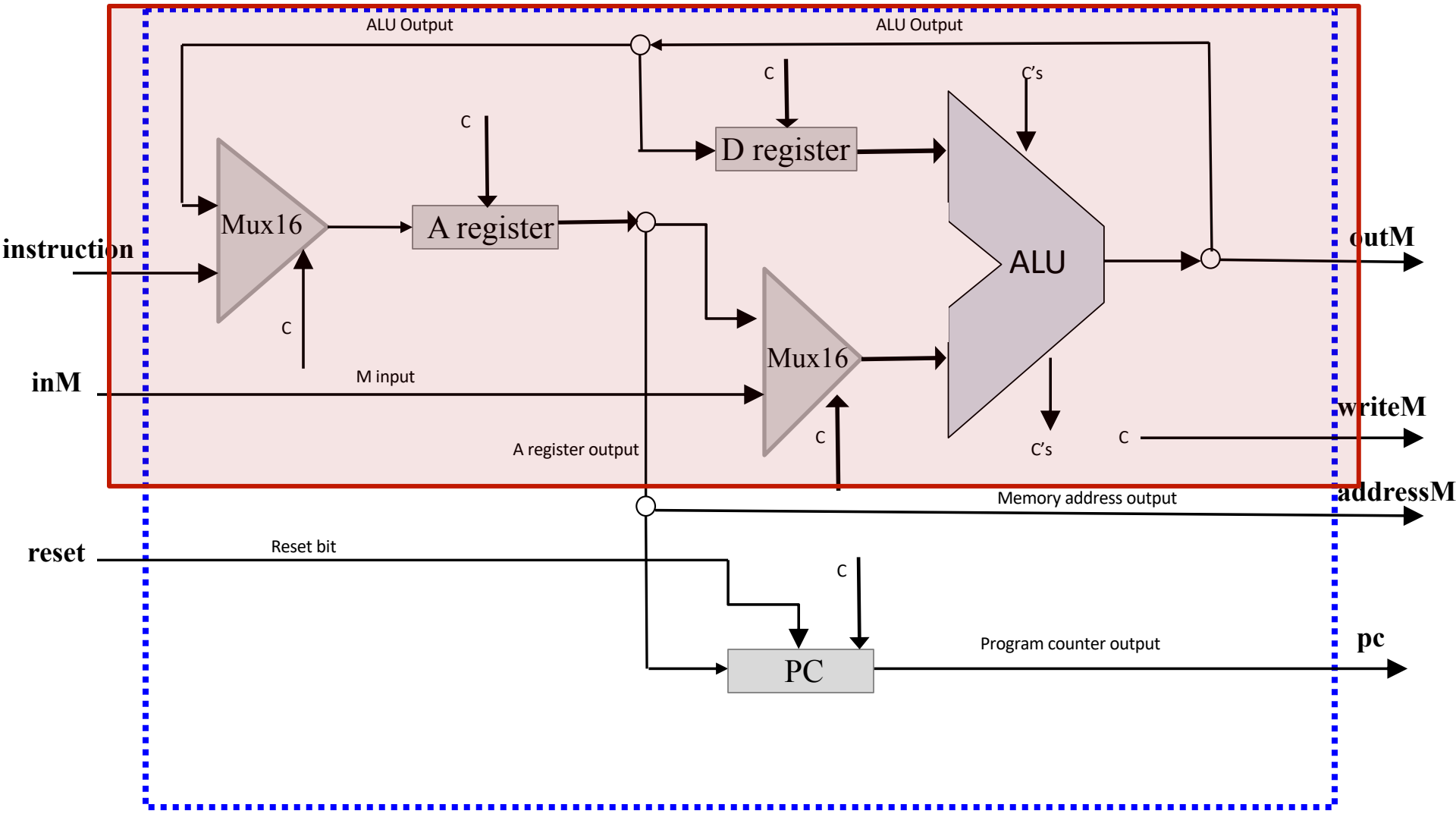
C-instruction

CPU handling of C-instruction:

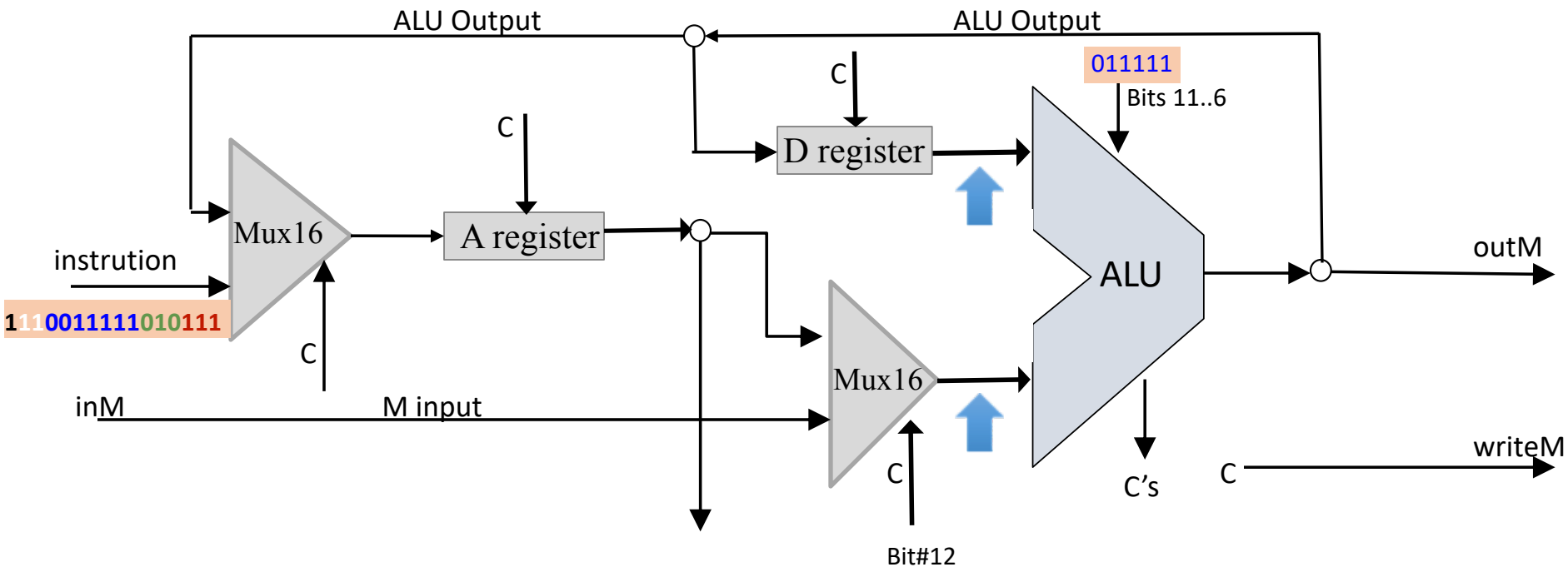
`dest= comp ;jump`

- 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

ALU Operation



ALU Operation: Inputs



ALU data inputs:

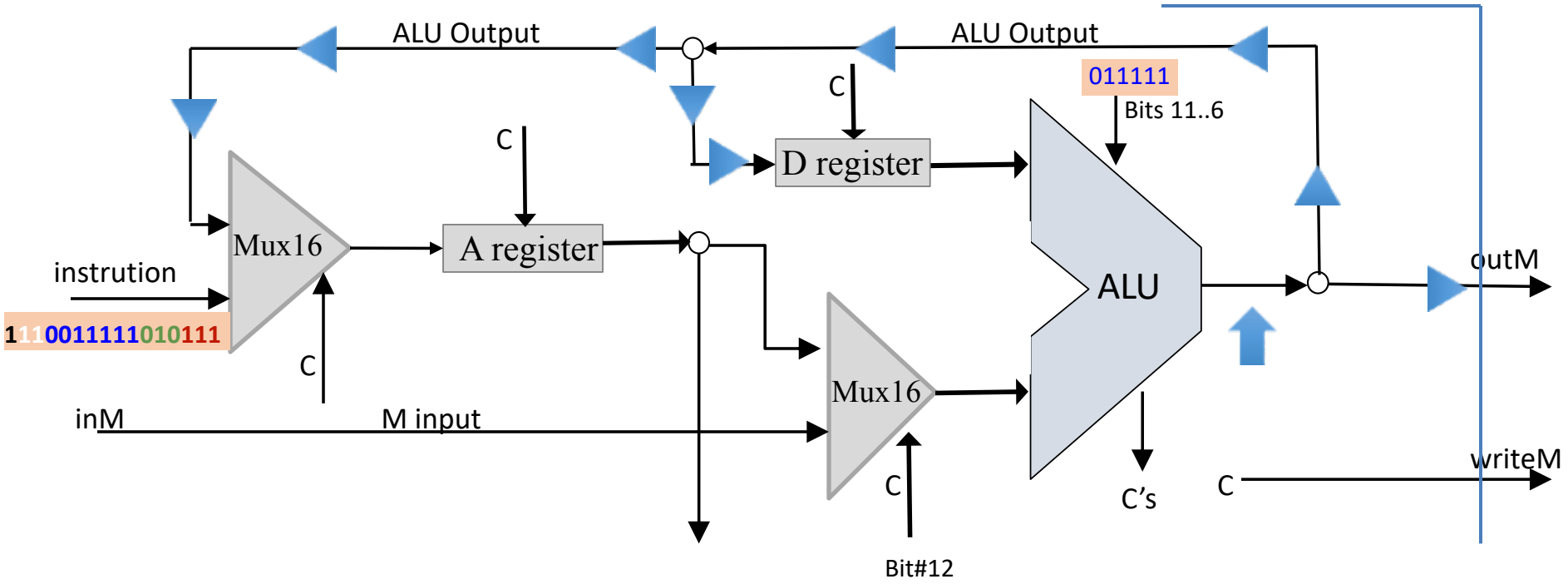
- Input 1: From D-register
- Input 2: From A-register or M-register (decided by bit#12 of Instruction)

1 11 a cccccc ddd jjj

ALU control inputs:

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

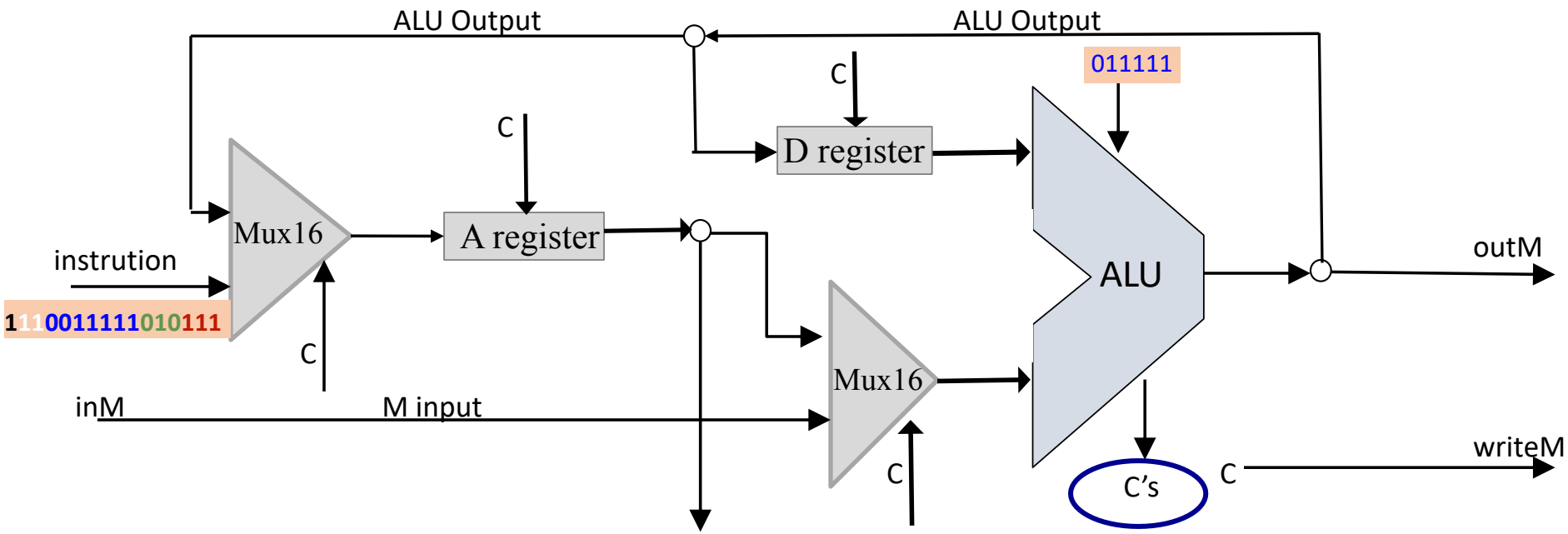
ALU Operation: Outputs



ALU data output:

- Result of ALU calculation, fed simultaneously to:
  - 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 (010 means D-register)

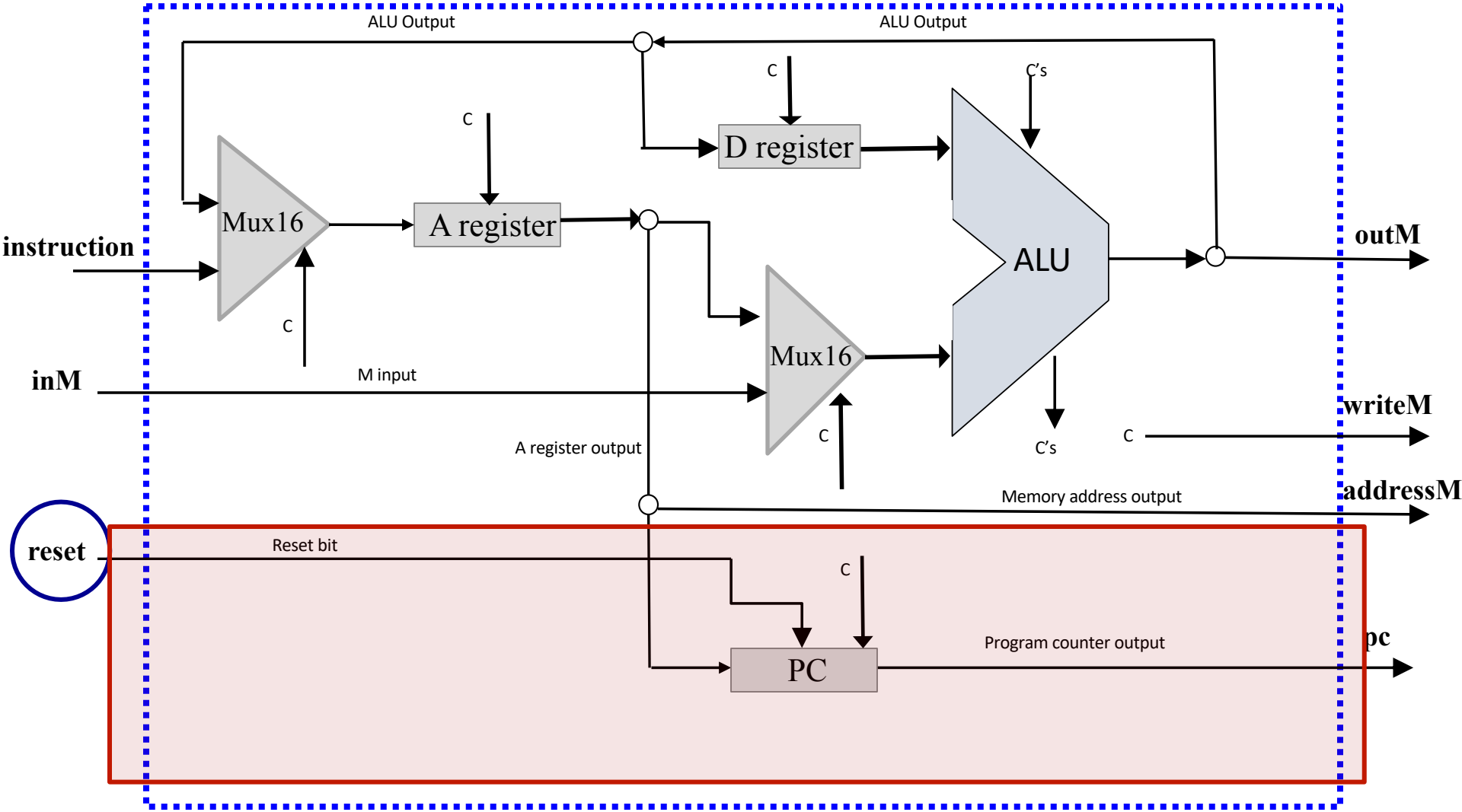
ALU Operation: Control Outputs



ALU control outputs:

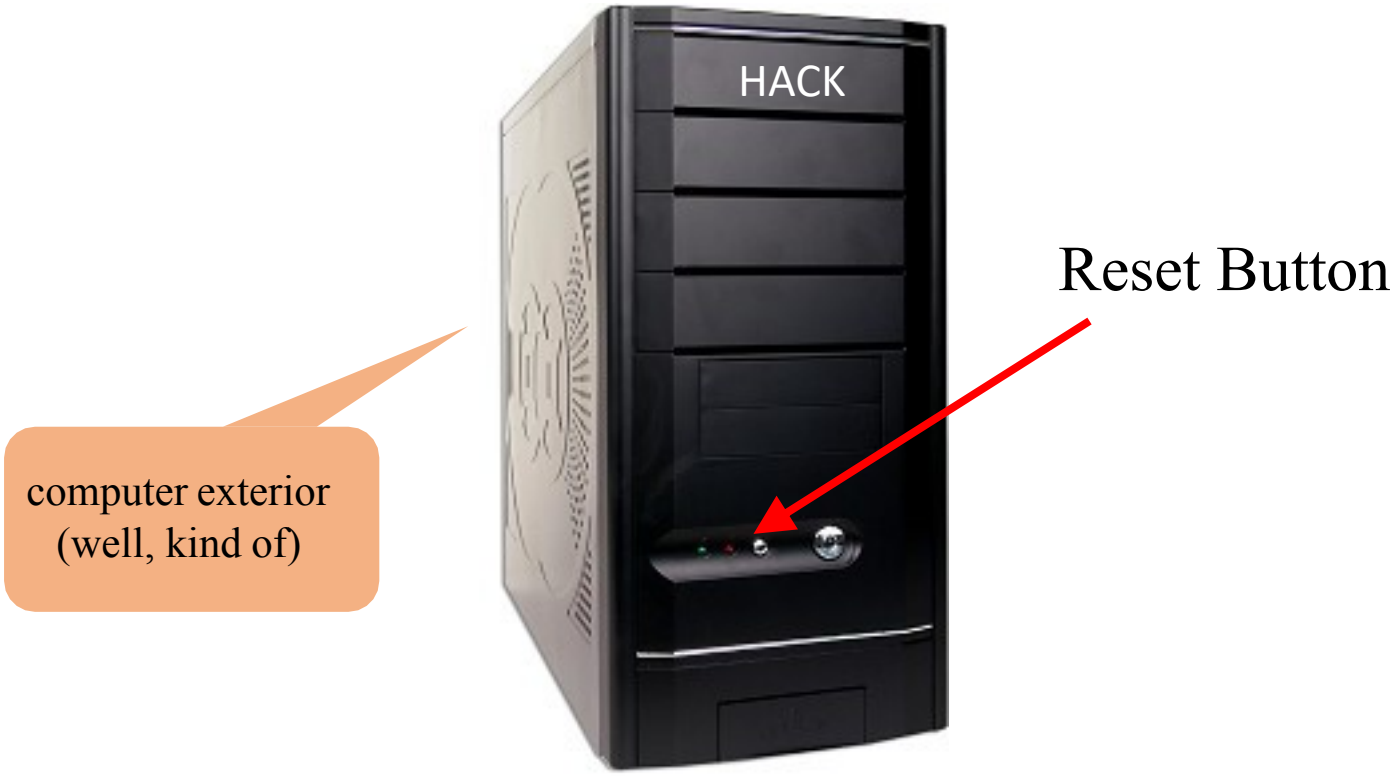
- is the output negative? (ng)
- is the output zero? (zr)

Control Logic of CPU



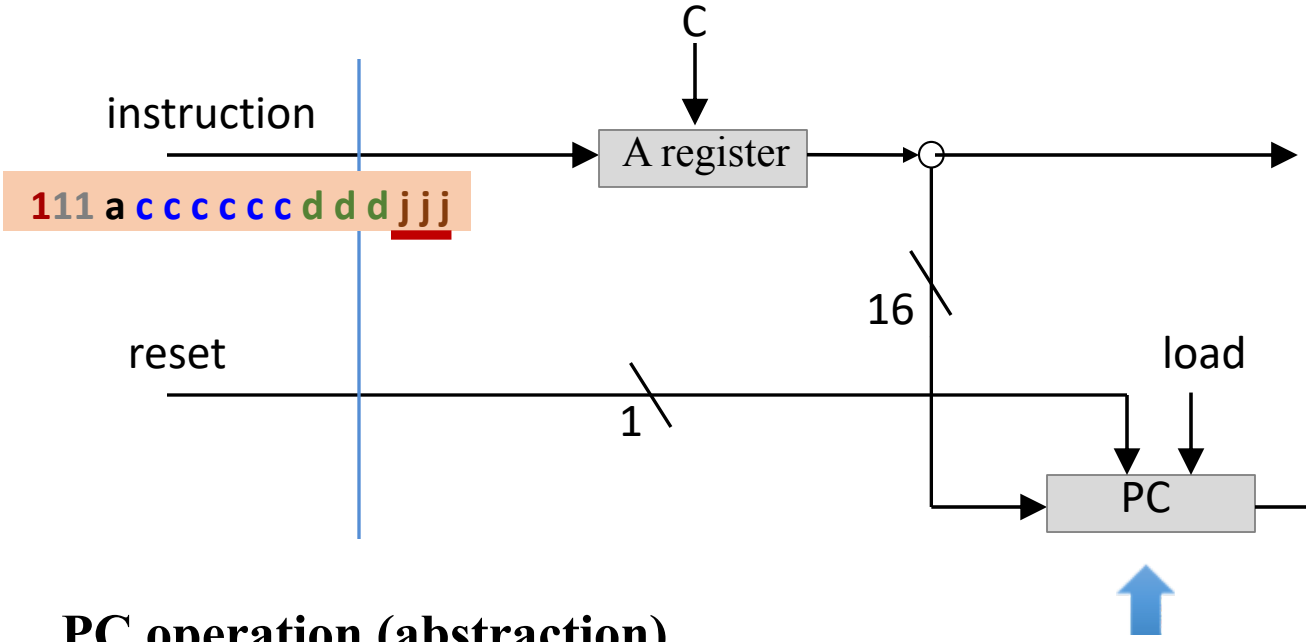
(each "C" symbol represents a control bit)

Possible Outside View of Hack Computer



- The computer is loaded with some program
- Pushing reset button causes the program to start running from beginning

Control Abstraction

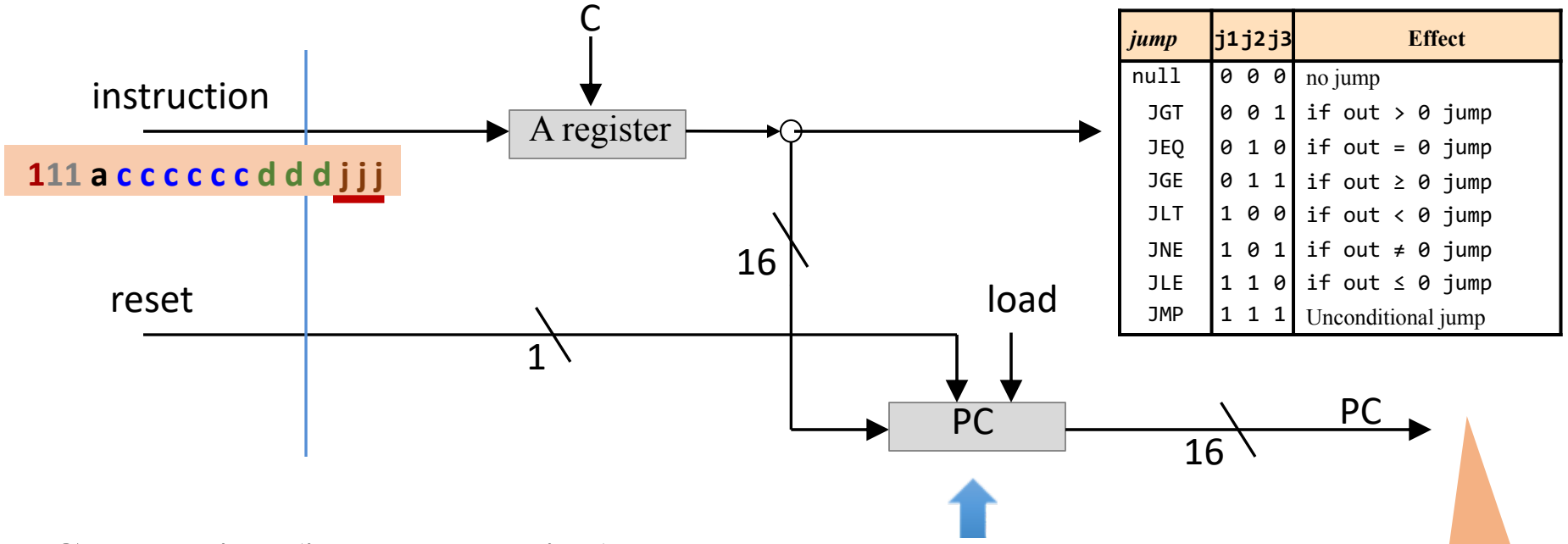


jump	j1 j2 j3	Effect
null	0 0 0	no jump
JGT	0 0 1	if out > 0 jump
JEQ	0 1 0	if out = 0 jump
JGE	0 1 1	if out ≥ 0 jump
JLT	1 0 0	if out < 0 jump
JNE	1 0 1	if out ≠ 0 jump
JLE	1 1 0	if out ≤ 0 jump
JMP	1 1 1	Unconditional jump

PC operation (abstraction)

- Emits the address of the next instruction
- restart PC=0
- no jump: PC++
- goto: PC=A
- conditional goto: if (condition) PC= A else PC++

Control Implementation



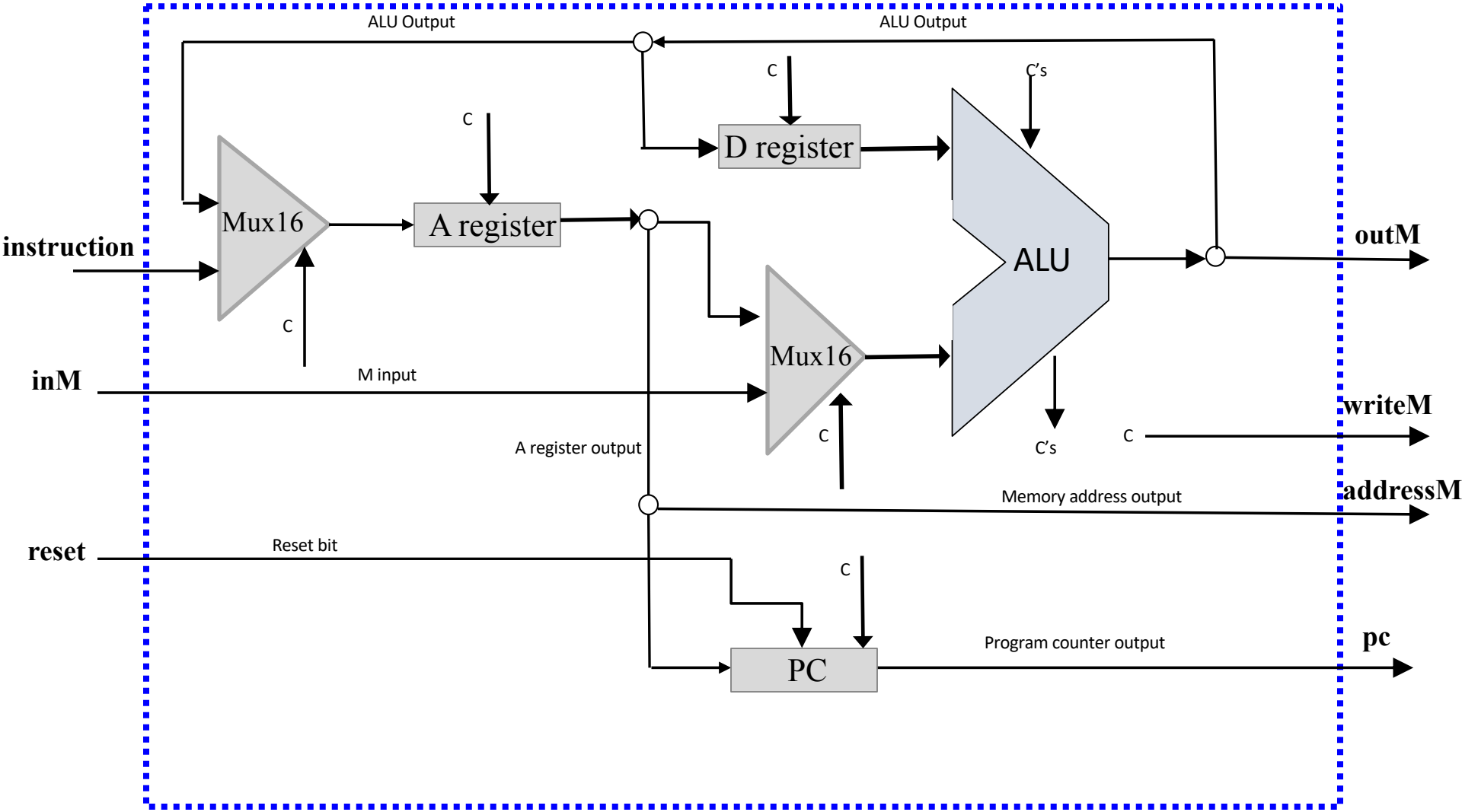
jump	j1 j2 j3	Effect
null	0 0 0	no jump
JGT	0 0 1	if out > 0 jump
JEQ	0 1 0	if out = 0 jump
JGE	0 1 1	if out ≥ 0 jump
JLT	1 0 0	if out < 0 jump
JNE	1 0 1	if out ≠ 0 jump
JLE	1 1 0	if out ≤ 0 jump
JMP	1 1 1	Unconditional jump

address of next instruction

PC operation (implementation)

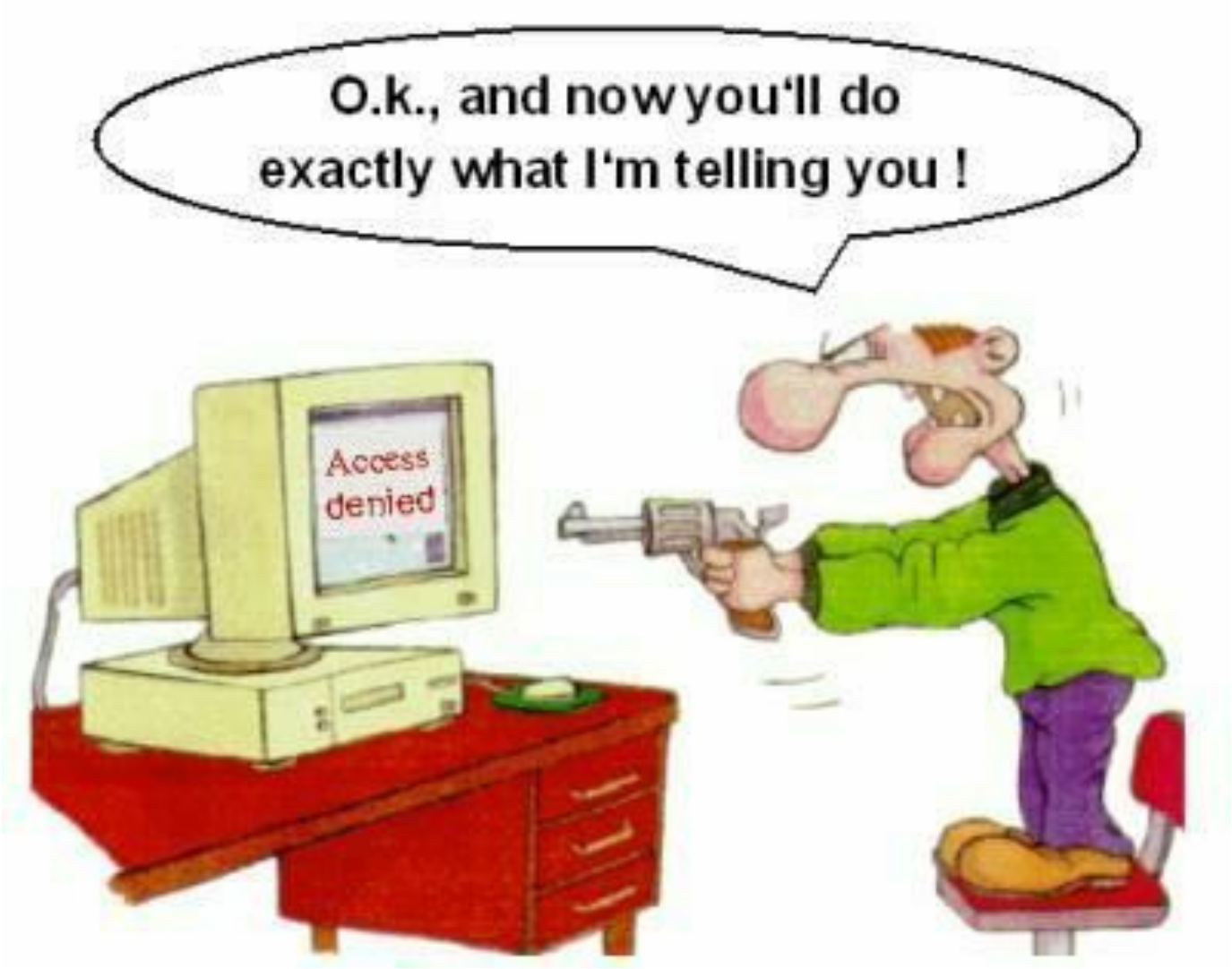
```
if (reset == 1)
    PC = 0
else
    load = f(jump bits, ALU control outputs (zr,ng))
    if (load == 1)  PC = A     // jump
    else            PC++       // next instruction
```

Hack CPU Implementation



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

Things To Do


O.k., and now you'll do exactly what I'm telling you !
Access denied