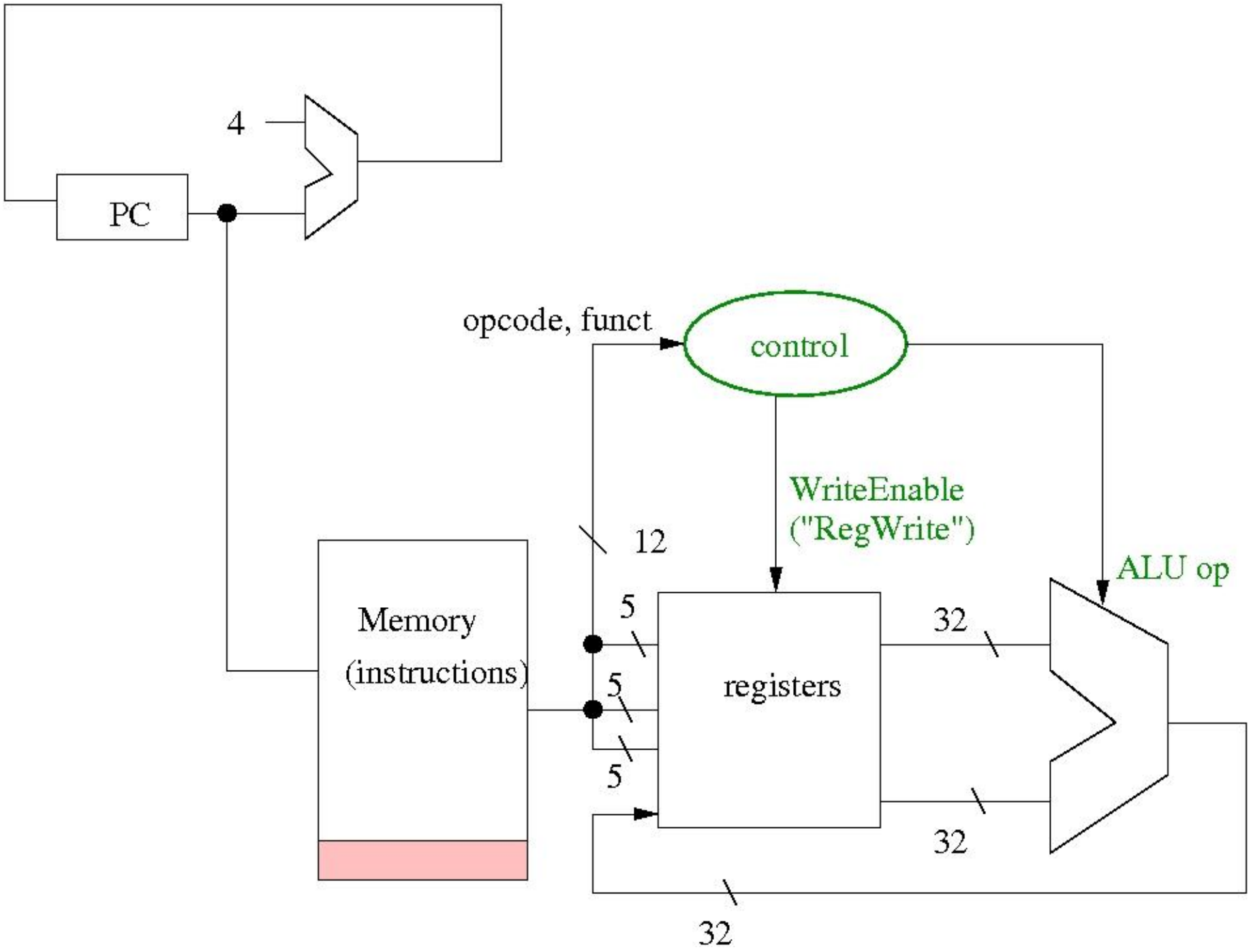
lecture 14

MIPS data path and control 2

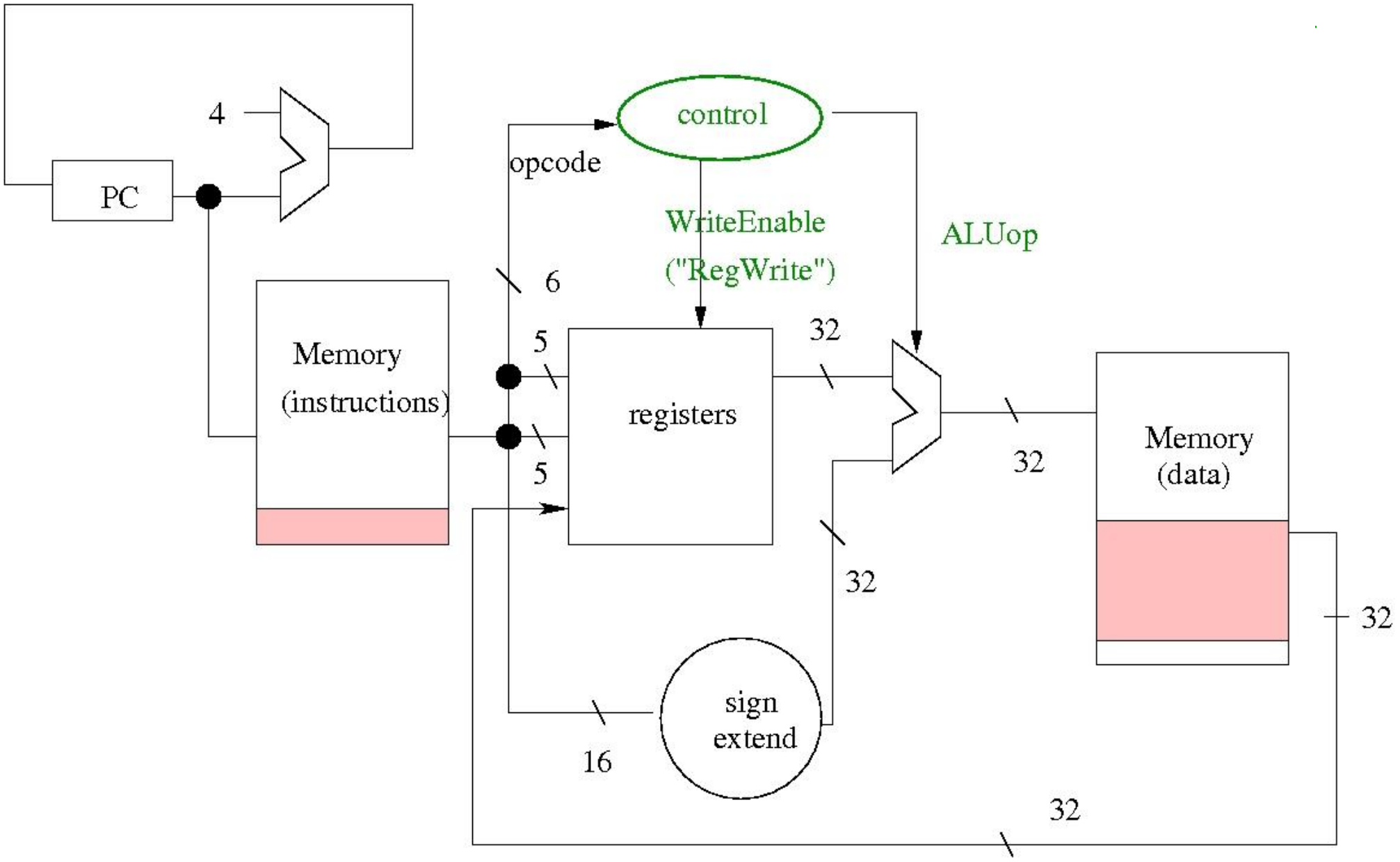
- merging data paths (for add, lw, sw, bne)
- controls
- more data paths

February 24, 2016

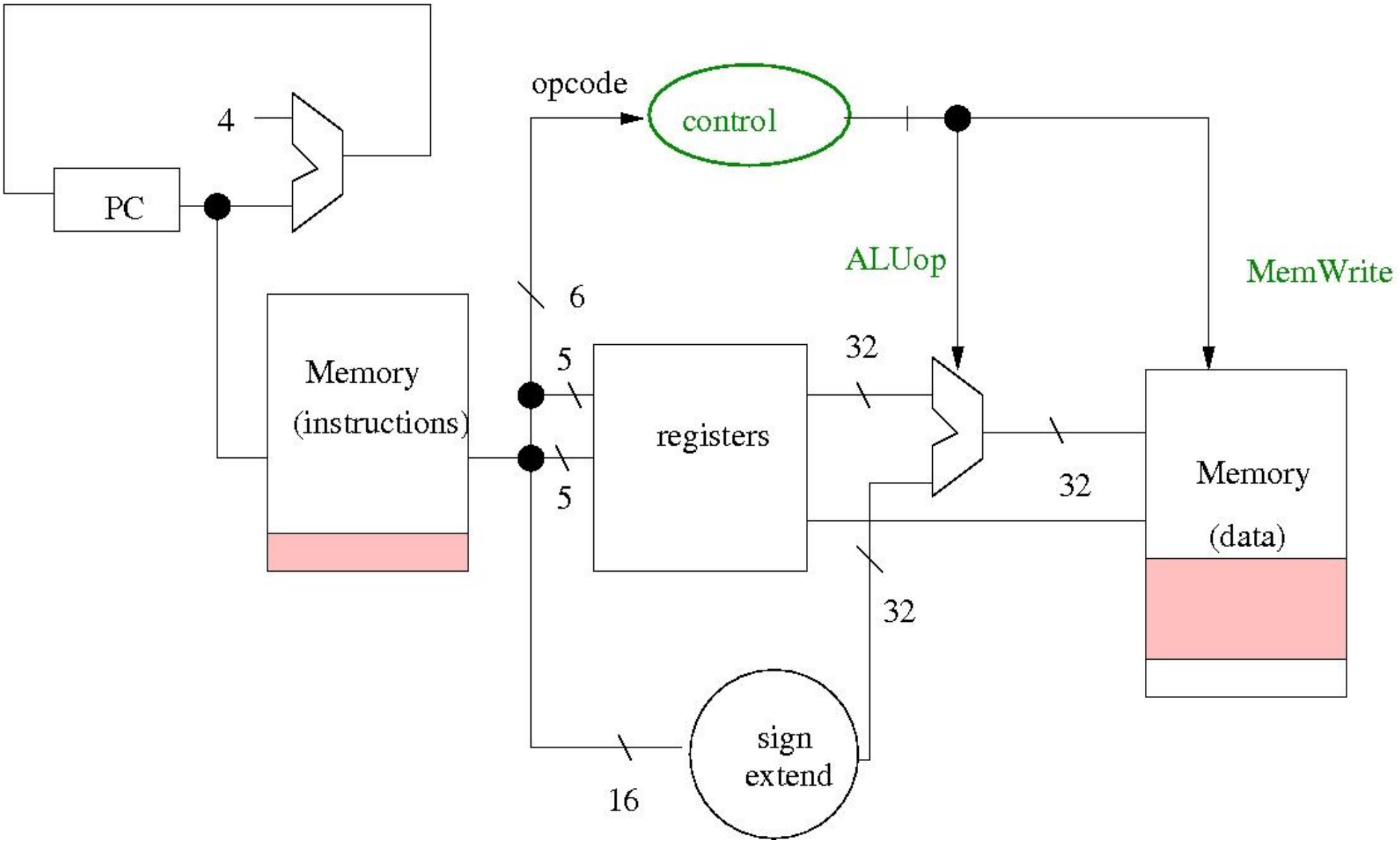
add $16, $17, $18


4
PC
opcode, funct
control
WriteEnable
("RegWrite")
ALU op
12
Memory
(instructions)
5
5
5
registers
32
32
32

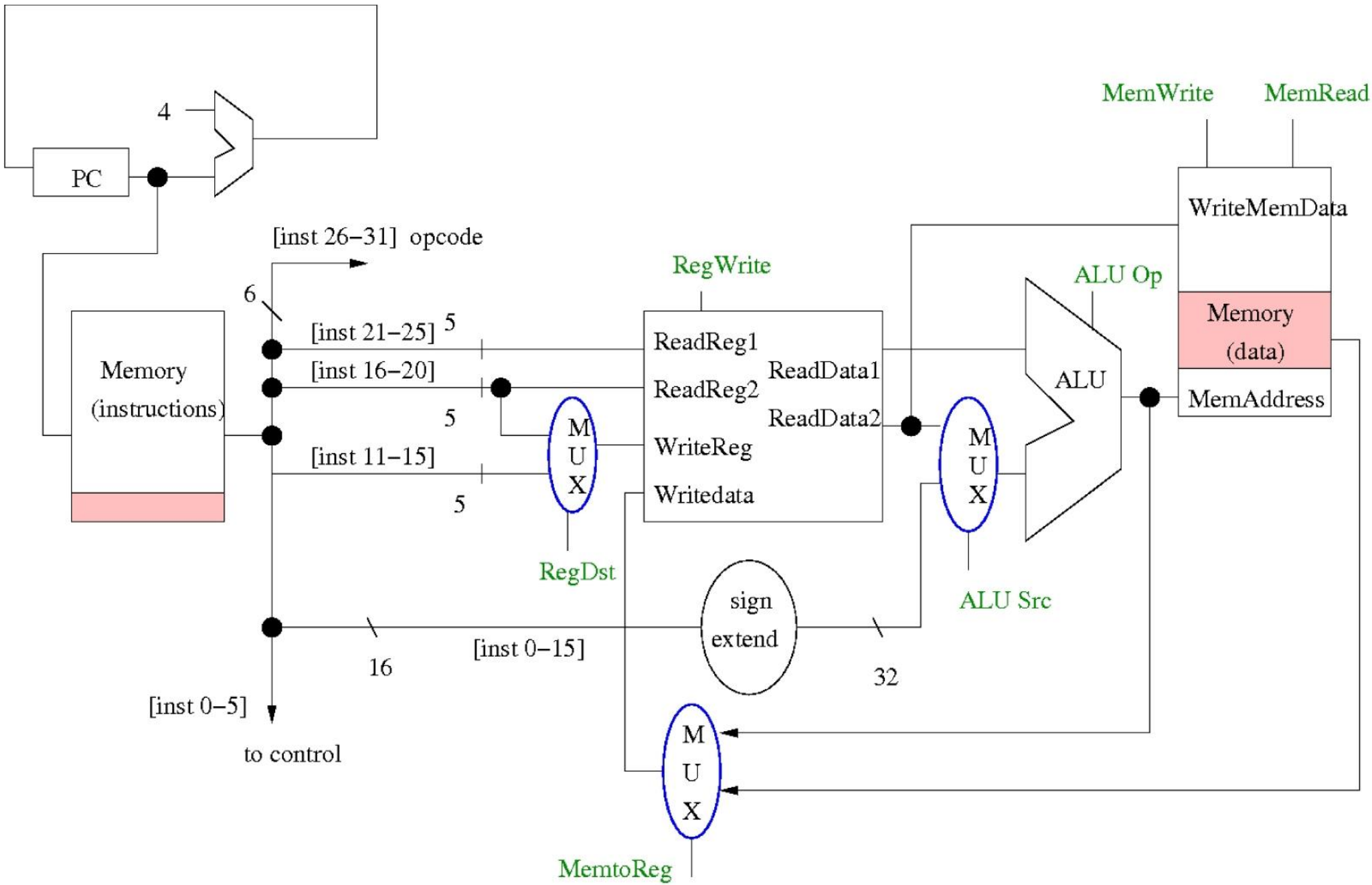
lw $16, 40($17)


4
PC
control
opcode
WriteEnable
("RegWrite")
ALUop
6
Memory
(instructions)
5
5
registers
32
32
Memory
(data)
32
32
32
sign
extend
16
32

sw $16, 40($17)

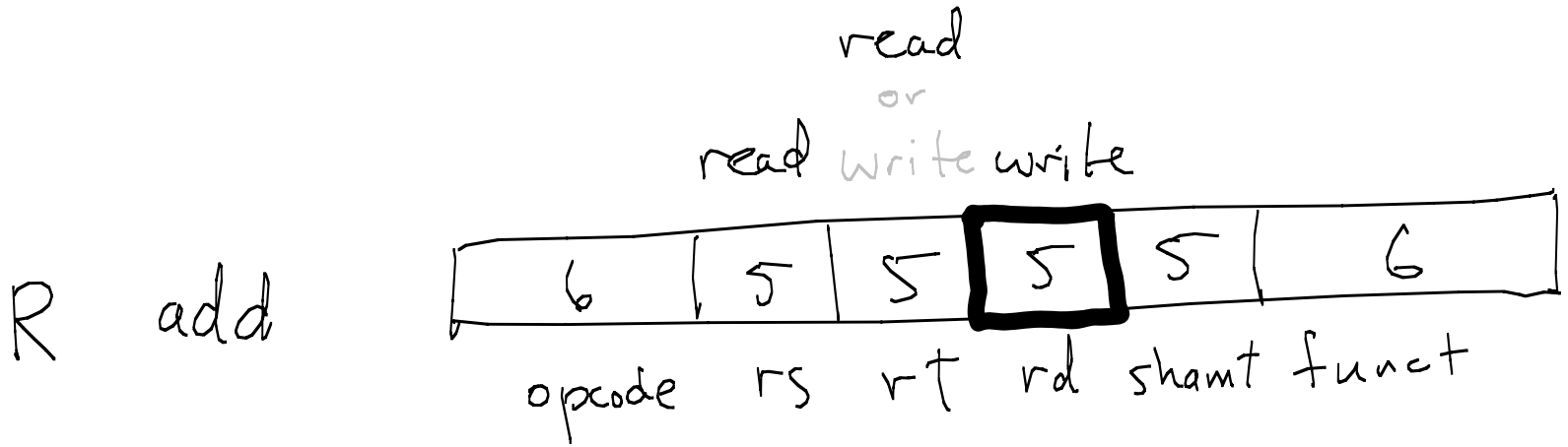

4
PC
opcode
control
ALUop
MemWrite
6
5
5
Memory
(instructions)
registers
32
32
32
Memory
(data)
sign
extend
16

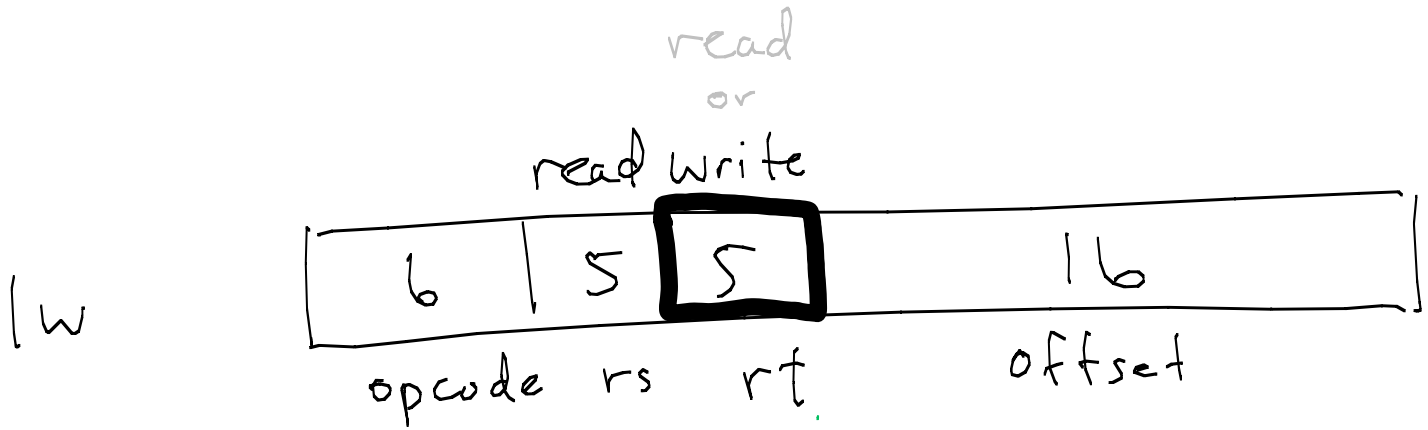
Merged data path (add, lw, sw)

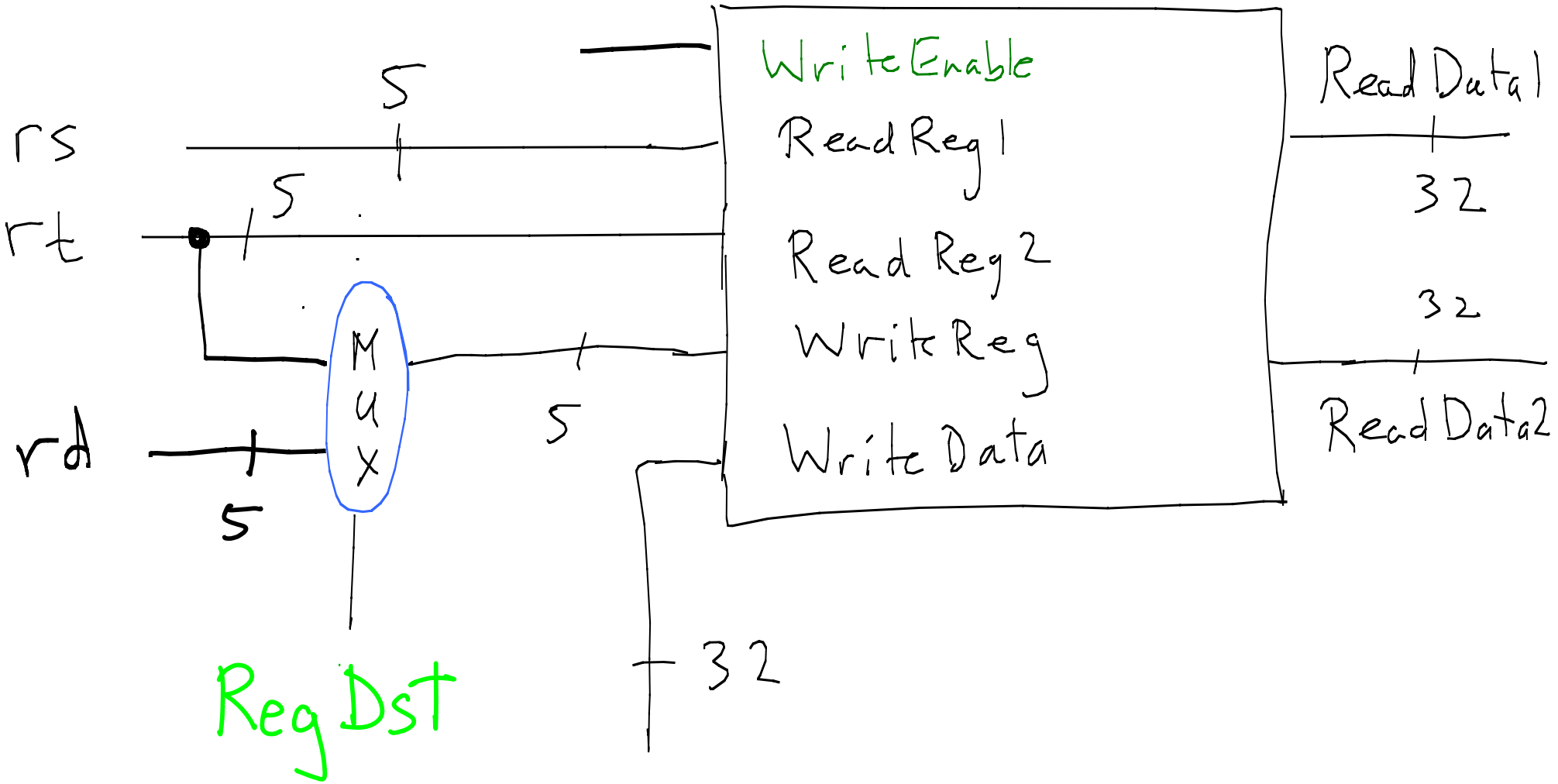


Let's look at the controls on the previous slide.

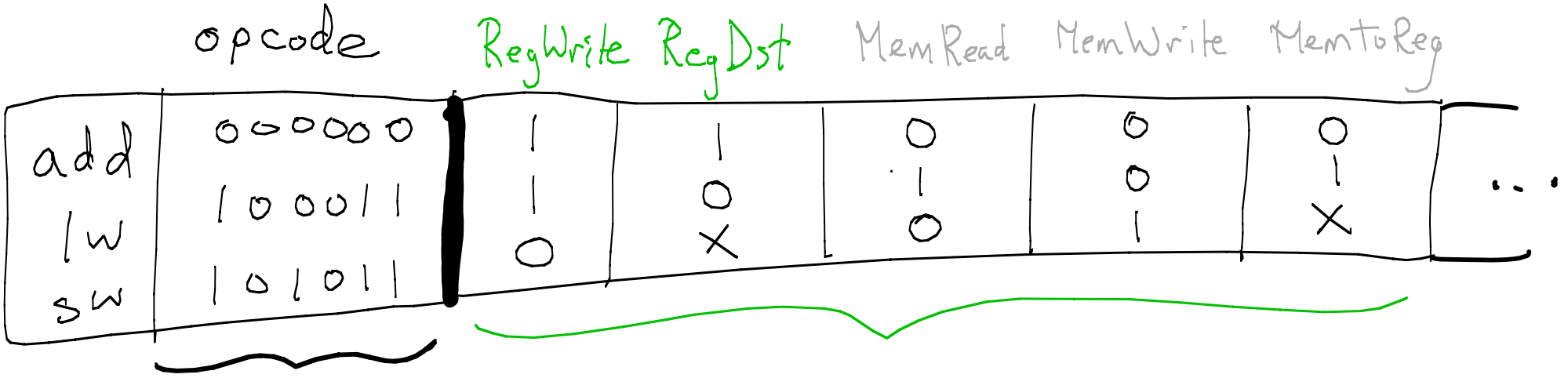
RegDst - if we are writing to a register, then to which **one**?






rs
rt
rd
5
5
5
5
M
u
x
RegDst
WriteEnable
ReadReg1
ReadReg2
WriteReg
WriteData
32
ReadData1
32
32
ReadData2

NEXT SLIDE

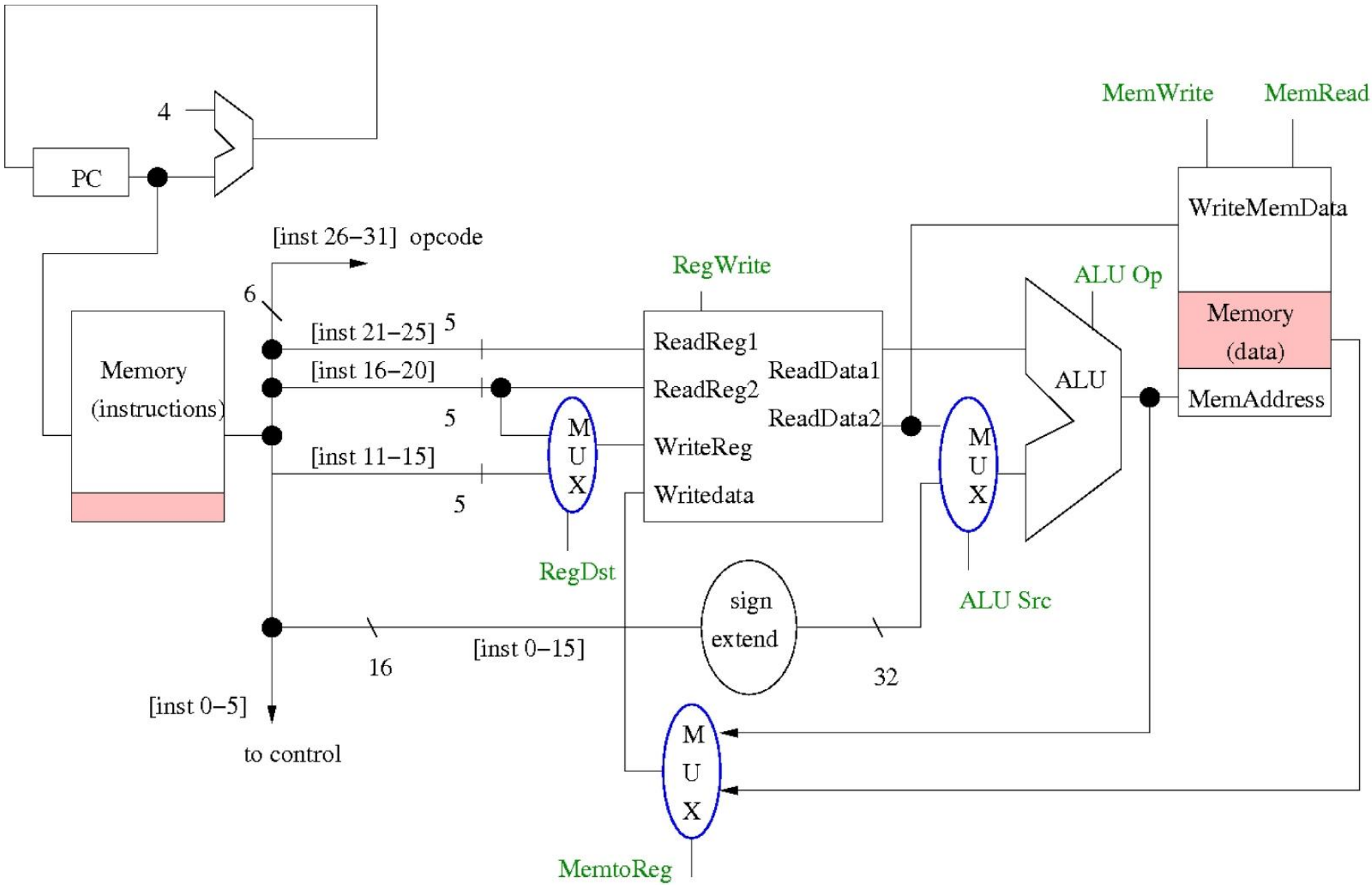


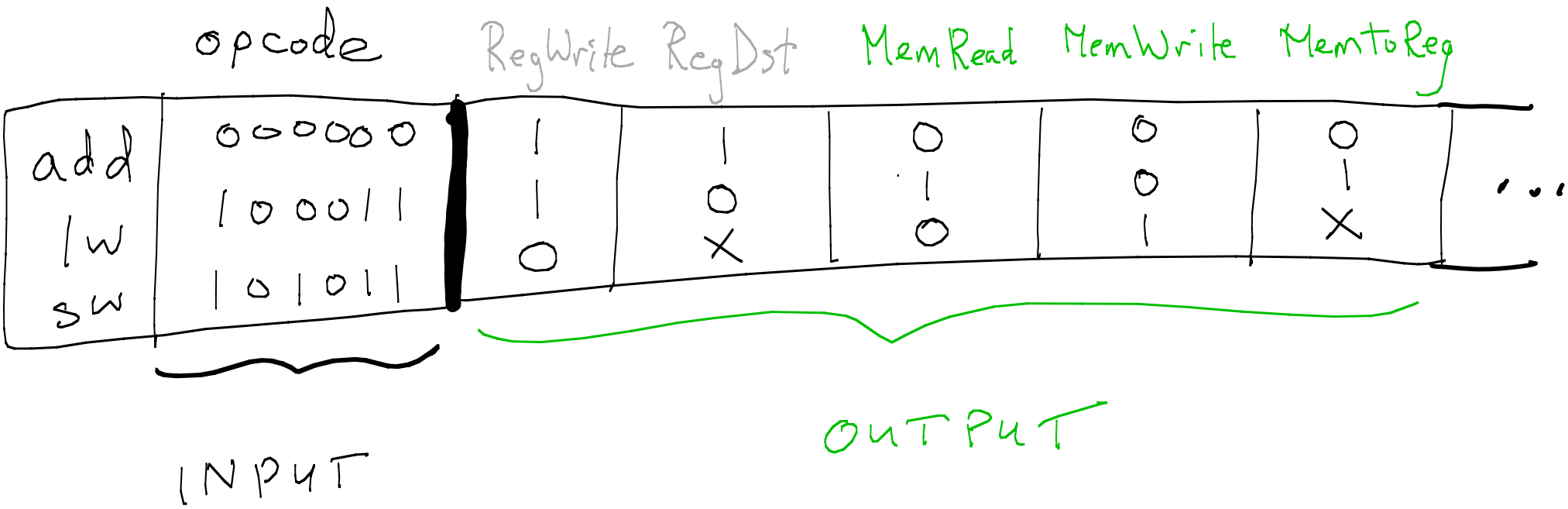
INPUT

OUTPUT

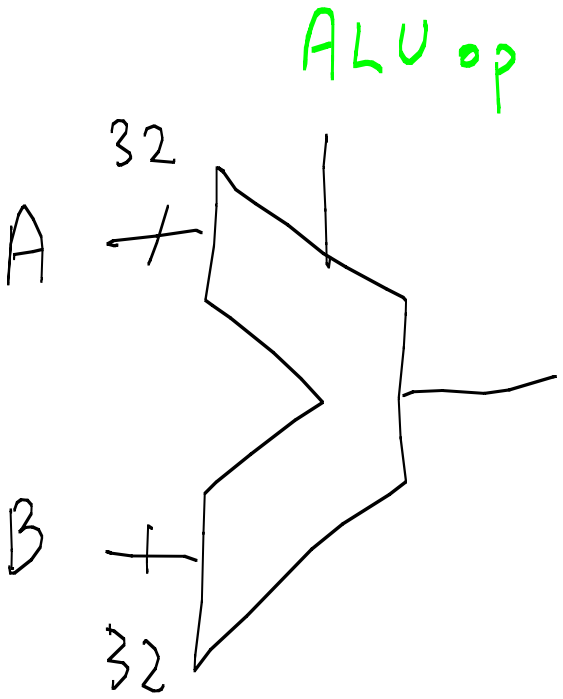
You can make a combinational circuit e.g. sum-of-products.

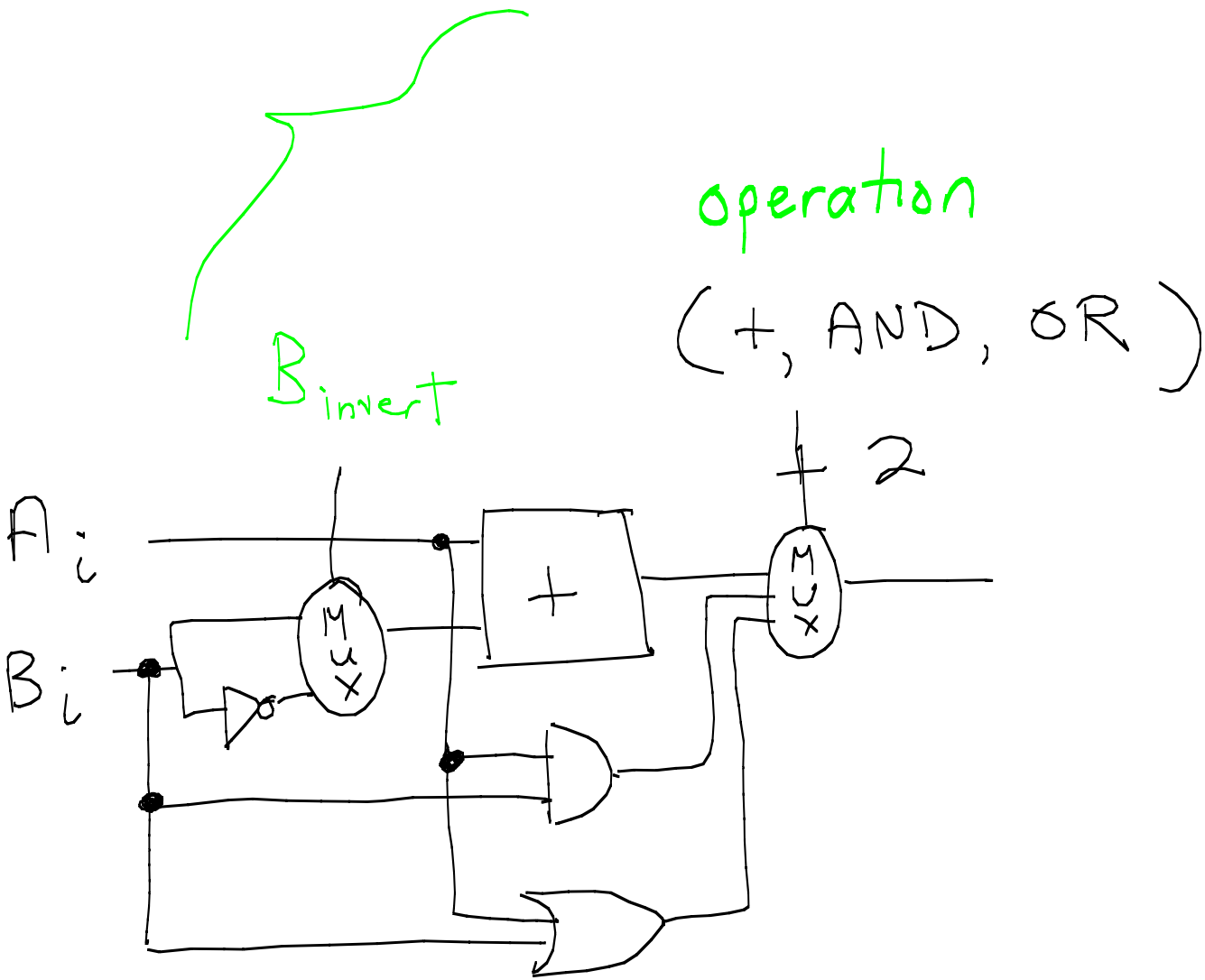
Merged data path (add, lw, sw)




opcode
RegWrite
RegDst
MemRead
MemWrite
MemtoReg
add
lw
sw
...
INPUT
OUTPUT

ALU (Recall lecture 4)

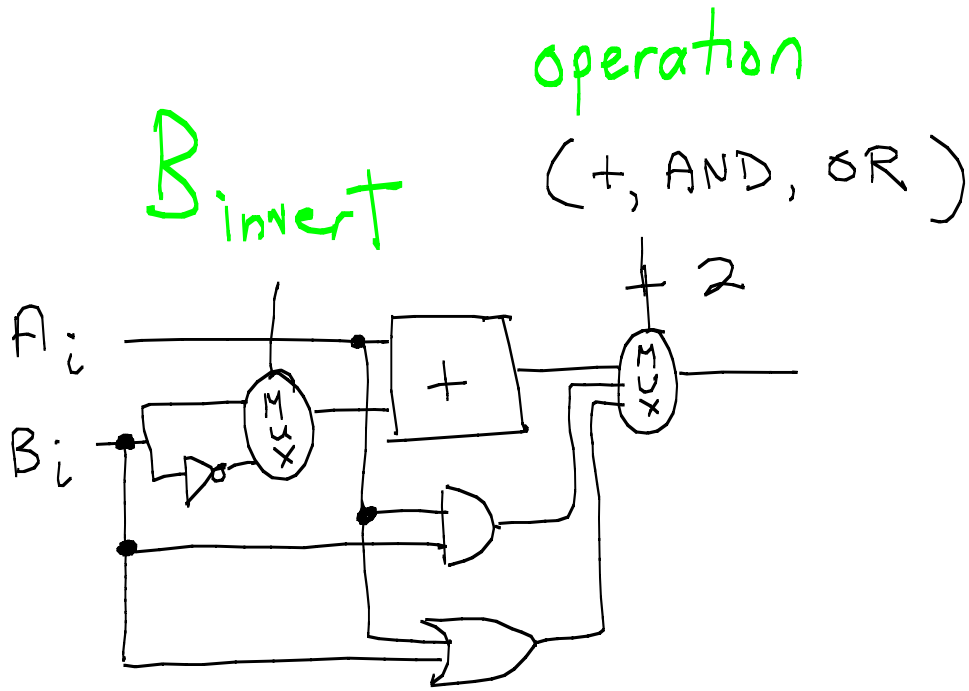




	INPUT		OUTPUT	
	opcode	funct	Binvert	Operation
add	000000	100000	0	10
sub	000000	100010	1	10
and	000000	100100	0	00
or	000000	100101	0	01
slt	000000	101010	1	11
lw	100011	XXXXXX	0	10
sw	101011	XXXXXX	0	10
bne	000101	XXXXXX	1	10
.	.	.	.	.

slt $t0, $s1, $s2

"Set if less than" performs subtraction and produces result 0 or 1.



Exercise 5 Q1

what more is needed in this circuit ?

For more details on the combinational circuit used to compute mapping on the previous slide, you would need to consult standard textbook by Patterson and Hennessey.

However, if you do, you will find that I'm defining ALUop in a more simplified way from what is done in the book.

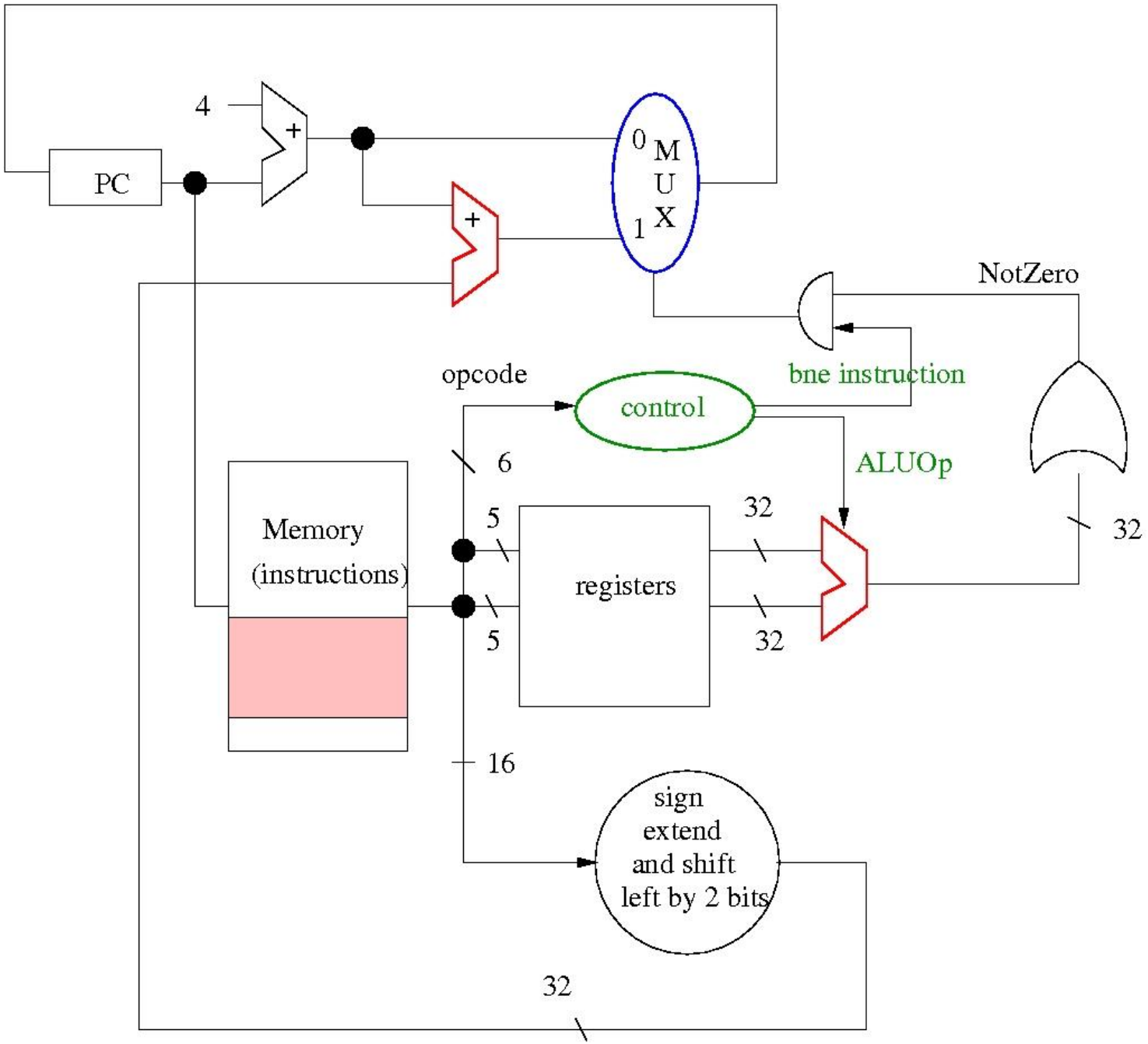
They use two combinational circuits: one for the instructions with opcode 000000 and funct field,
and one for instructions with other opcodes.

Branching

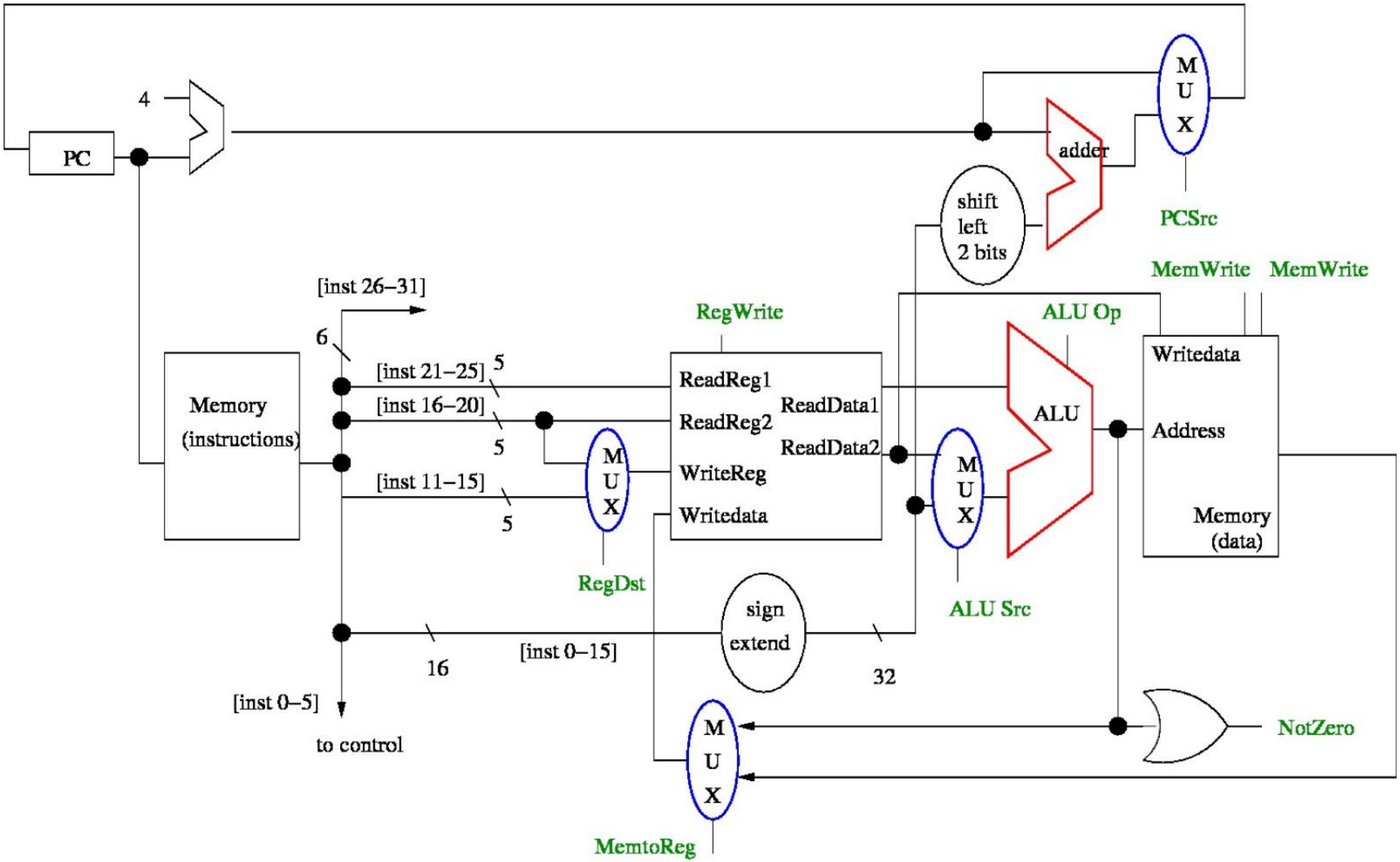
- conditional: bne, beq
- unconditional
  - jump ("j")
  - jump and link ("jal")
  - jump register ("jr")

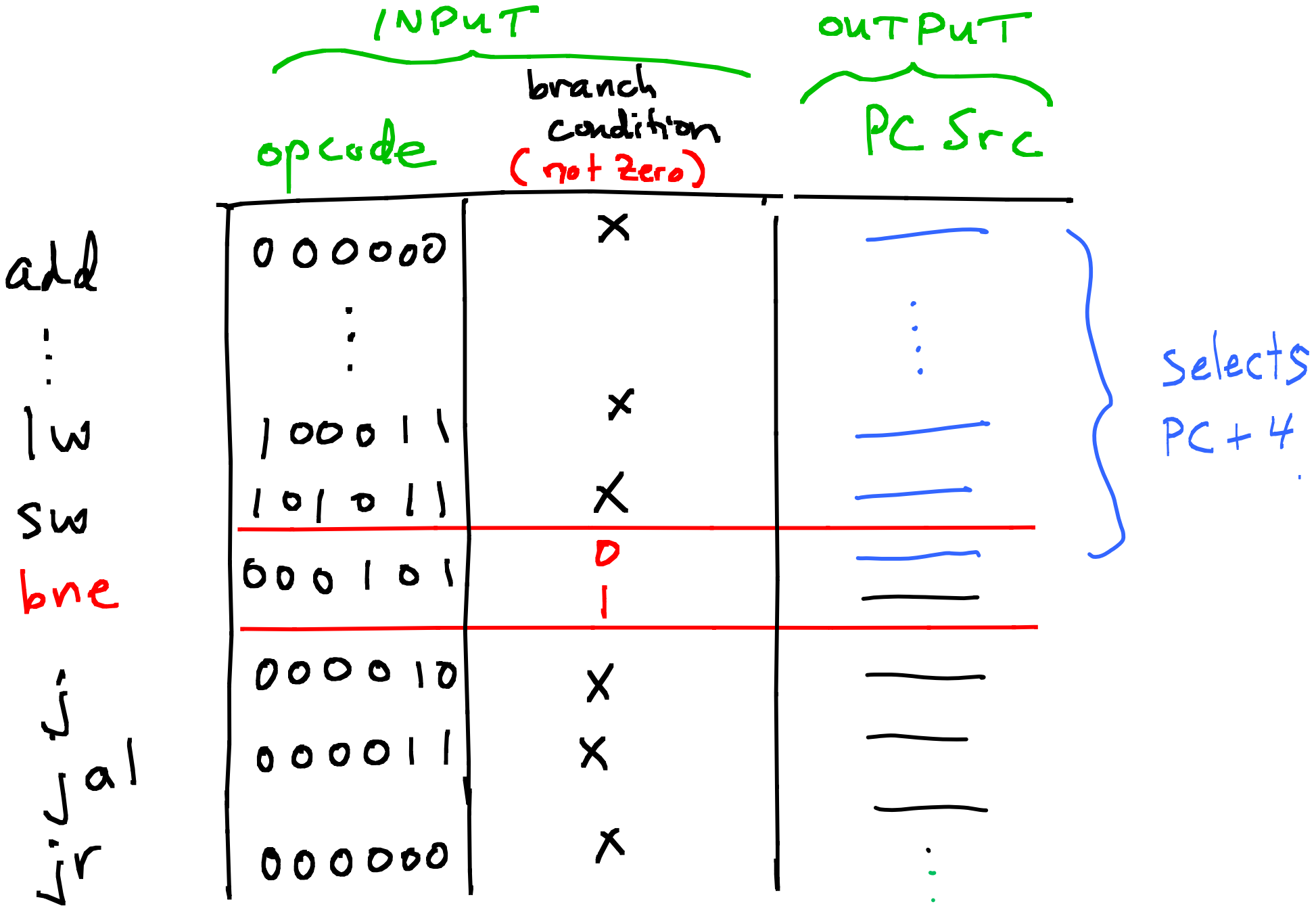
In each case, we need to determine the next value of the program counter (PC) and possibly do other stuff.

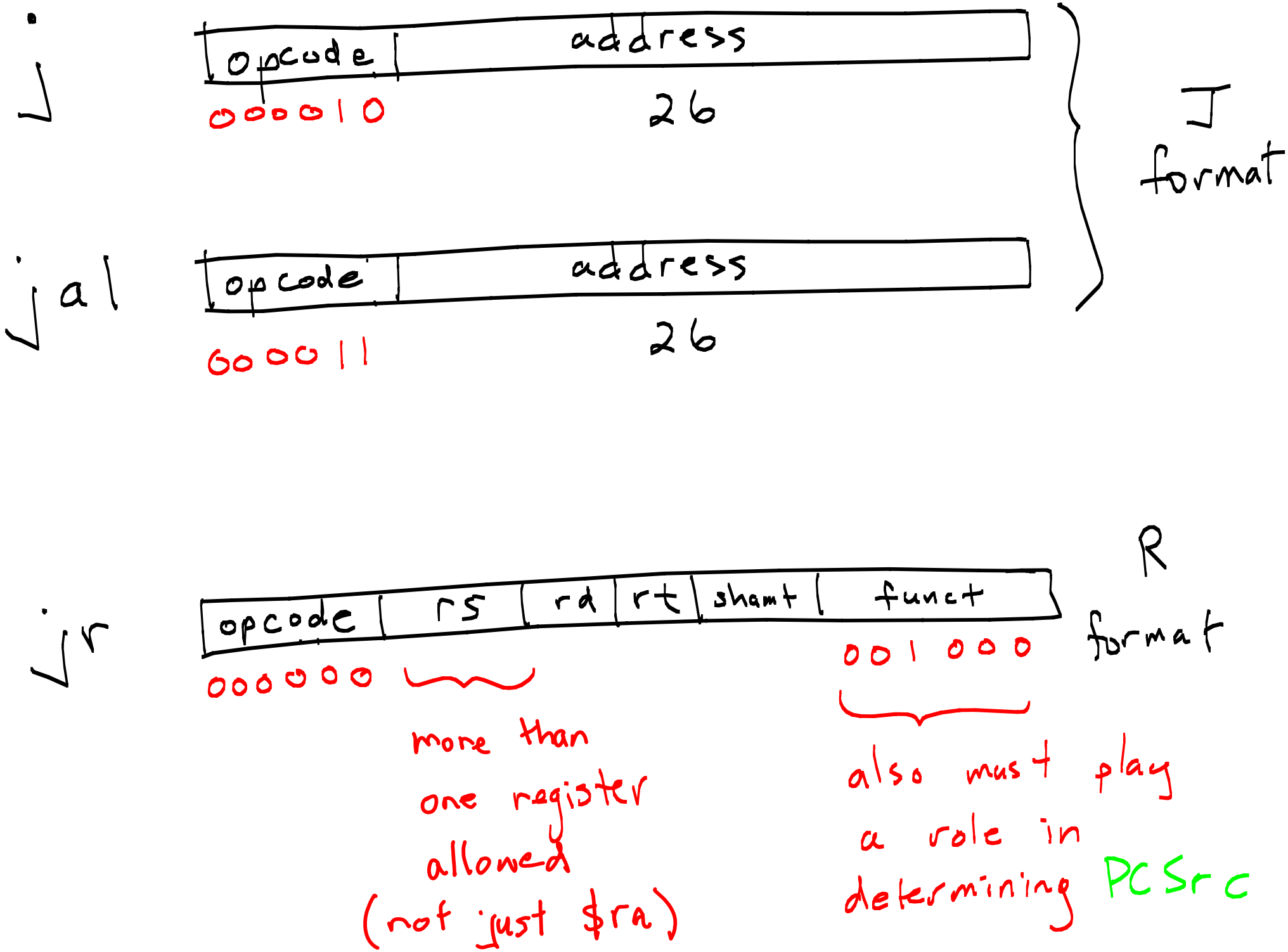
bne $18, $17, Exit



Here we merge the add/etc, lw, sw, **bne** datapaths.
We introduce a **PCSrc** control.

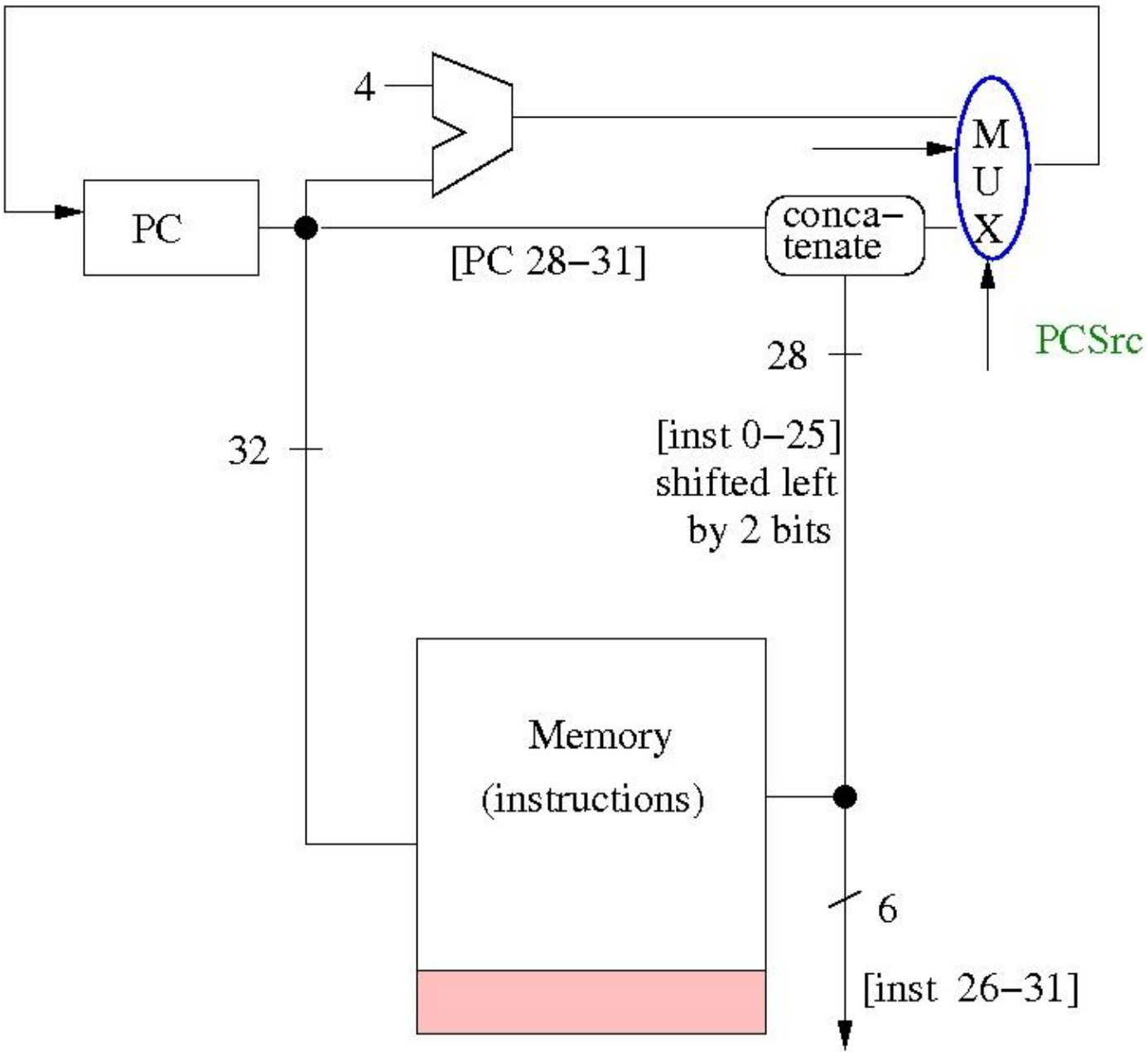


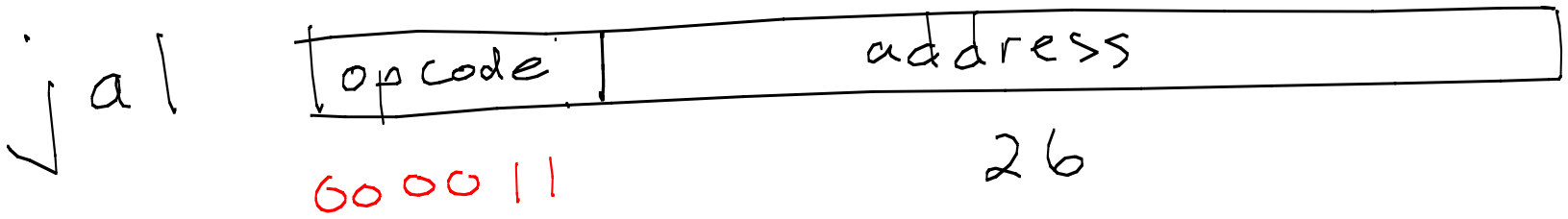

INPUT
OUTPUT
opcode
branch condition (not zero)
PCSrc
add
lw
sw
bne
jr
jal
000000
100011
101011
000100
000011
000000
X
X
X
0
1
X
X
X
Selects PC + 4


j
opcode
address
000010
26
jal
opcode
address
000011
26
J
format
jr
opcode
rs
rd
rt
shamt
funct
000000
001000
R
format
more than
one register
allowed
(not just $ra)
also must play
a role in
determining PCSrc

j

opcode	address
000010	26





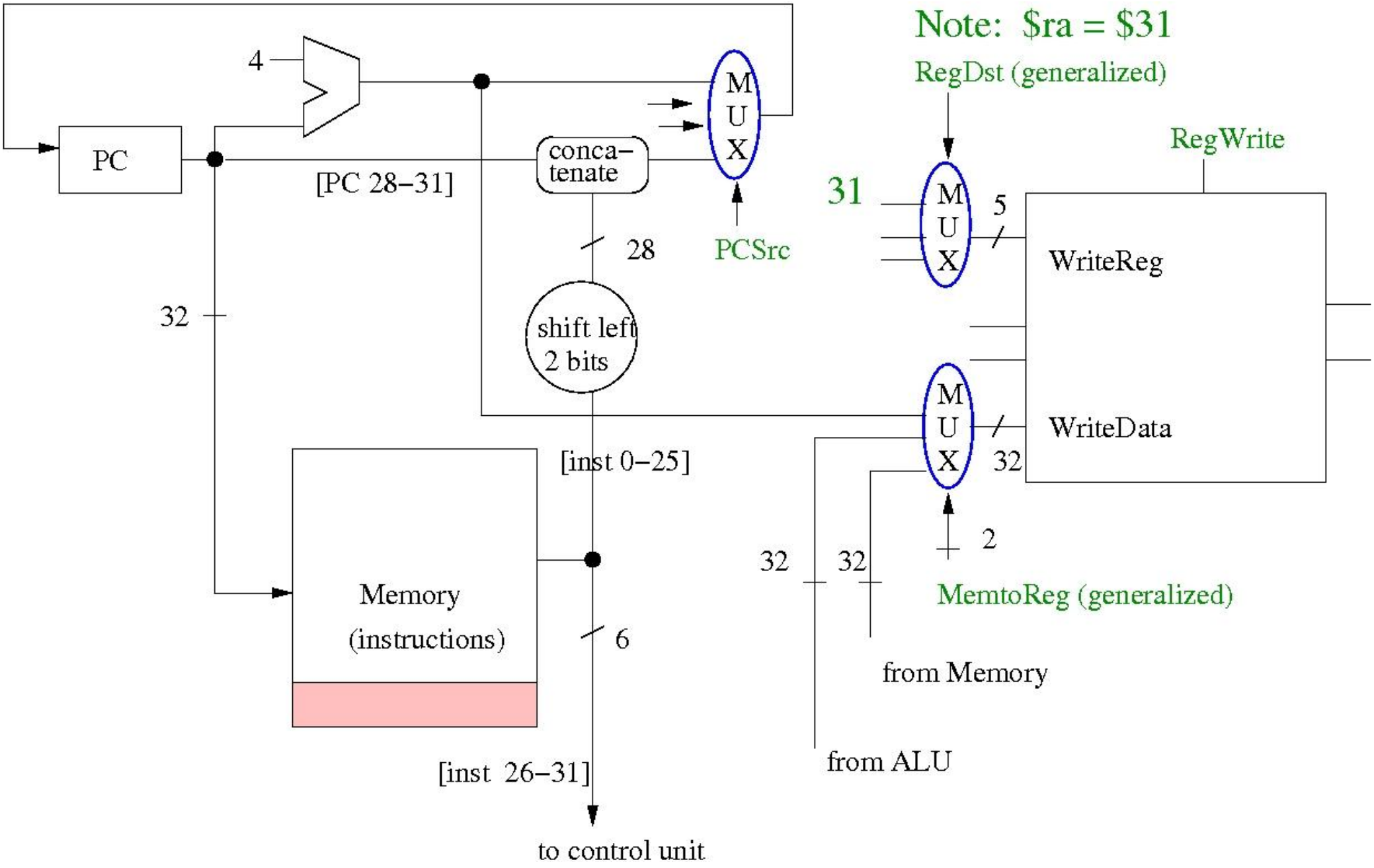
Jump and link ("jal") is used for function calls.

(Recall lecture 11. See Assignment 3.)

What is its datapath ?

jal

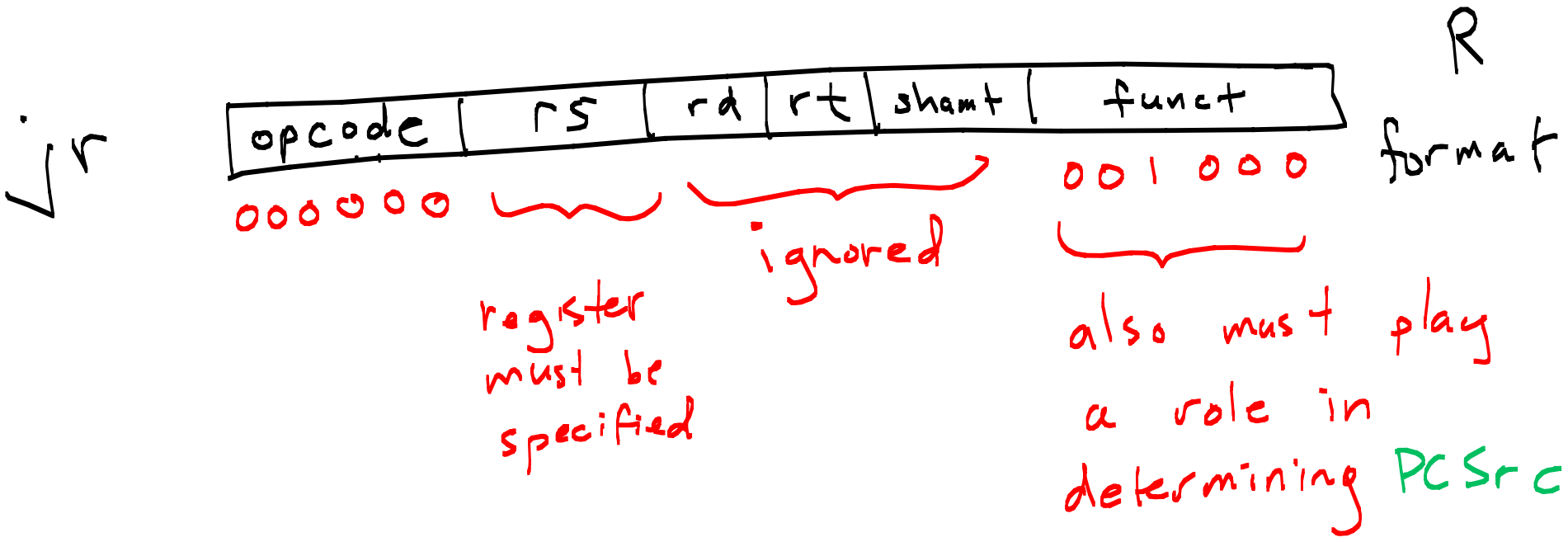

opcode
address
31, ... , 26, 25, 24, . 3 2 1 0


Note: $ra = $31
RegDst (generalized)
RegWrite
4
PC
[PC 28–31]
conca–
tenate
M
U
X
PCSrc
28
31
5
WriteReg
32
shift left
2 bits
WriteData
[inst 0–25]
2
32
32
MemtoReg (generalized)
Memory
(instructions)
6
from Memory
from ALU
[inst 26–31]
to control unit

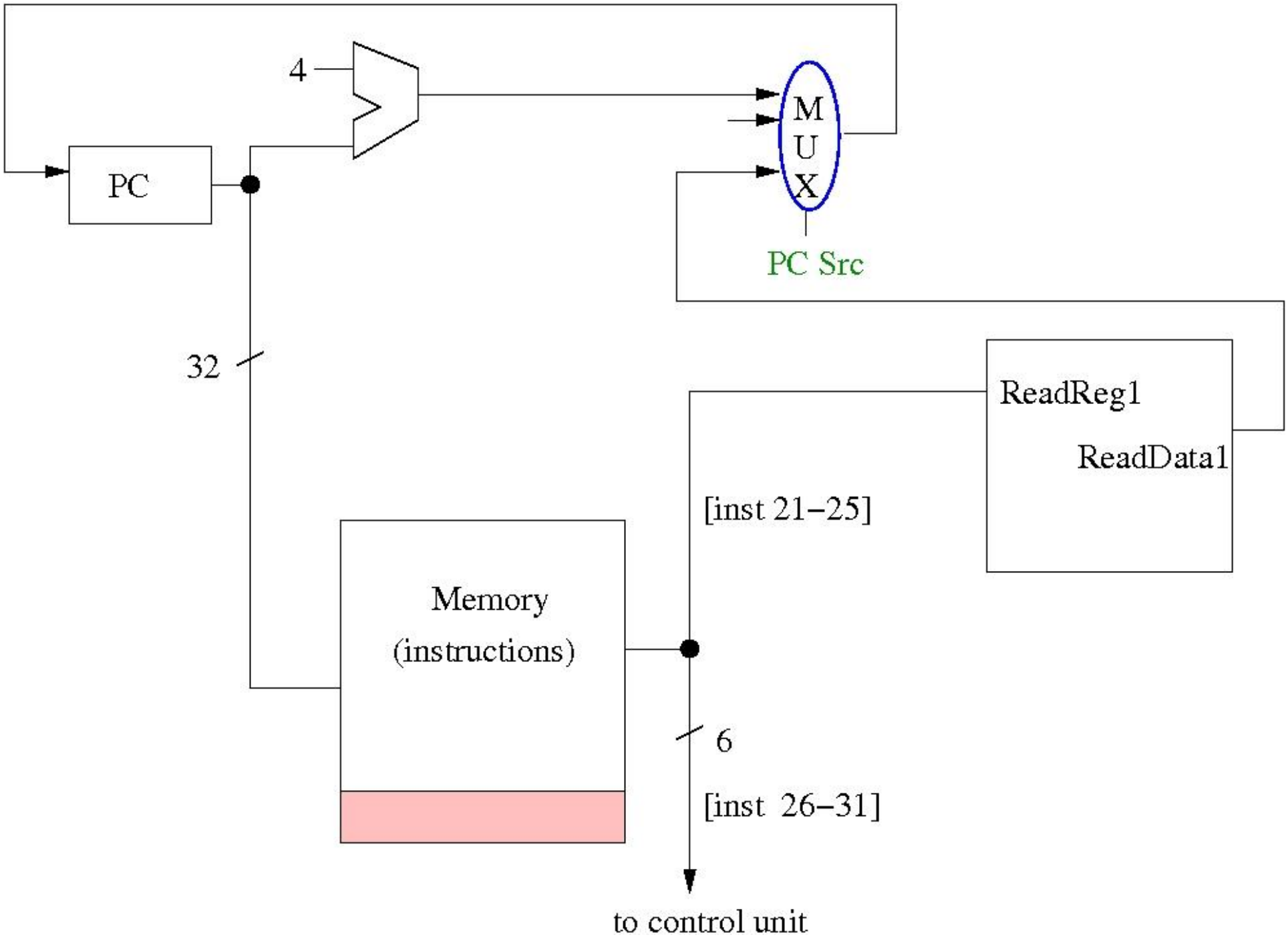
What does "jump register" (jr $ra) do ?

It is used to return from function calls:

(lecture 11 and Assignment 3)



jr $ra

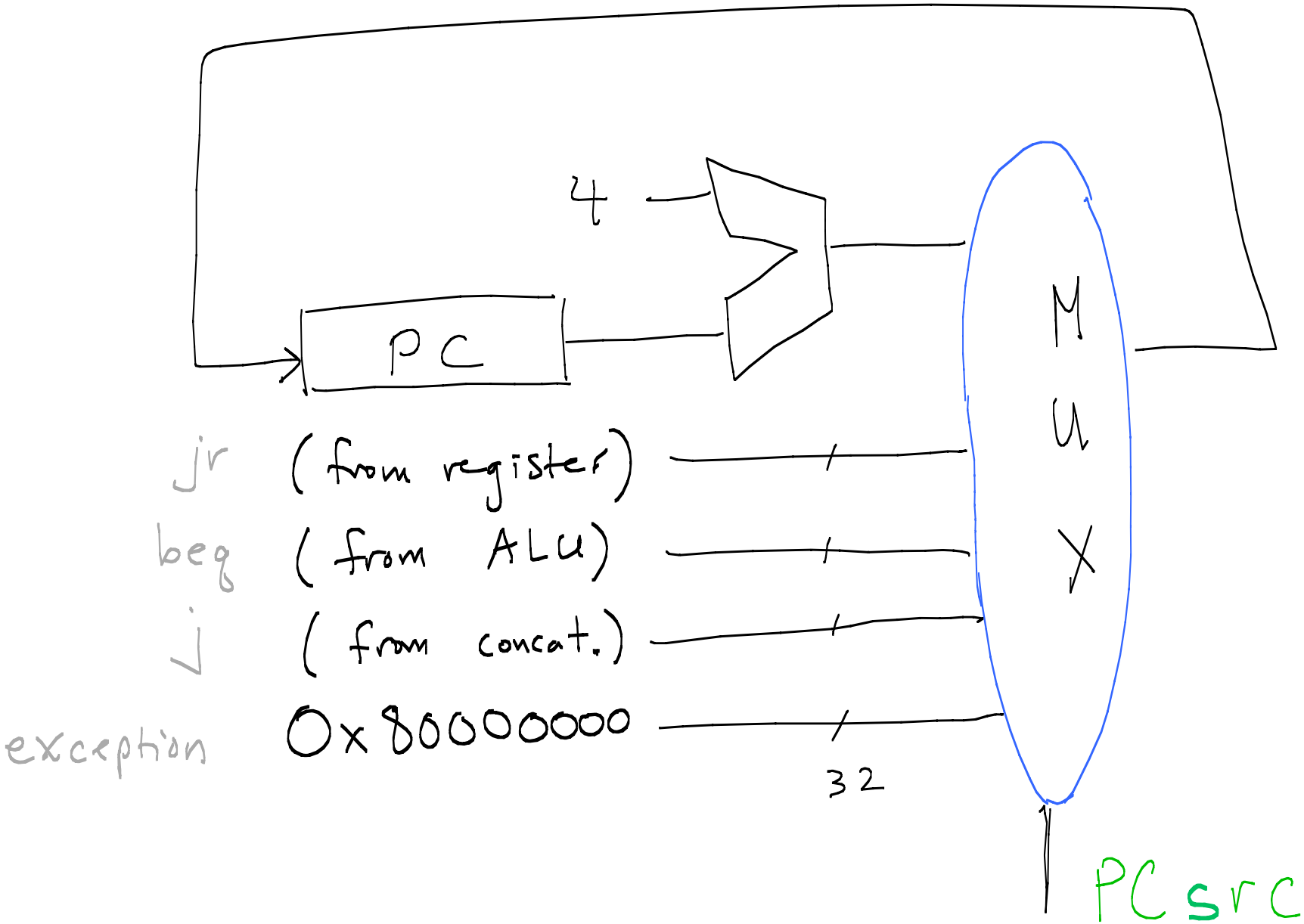

4
M
U
X
PC
PC Src
32
ReadReg1
ReadData1
[inst 21–25]
Memory
(instructions)
6
[inst 26–31]
to control unit

Exceptions

- integer division by 0
- bad address
- ....
- system call
- "interrupt" e.g. Ctl-Alt-Del

⟹ Program will branch from user to kernel (0x8000 0000)

PCsrc control is used to *select* the address of the next instruction to be executed.



WARNING

Single Cycle Model (last two lectures) is not used in real MIPS implementations. Why not?

- inefficient : each clock cycle must be long enough for worst case

  (Which instruction has longest data path on CPU ?)

- multiplication, division, floating point ops (co-processor 1) also require more than 1 clock cycle

After the Study Break, I will sketch out the "multicycle" model that MIPS CPU's do use.

MARS DEMO

Recursive function calls and stack

Use example of sumton from lecture 11.

I have added a link to the code:
http://www.cim.mcgill.ca/~langer/273/sumton.asm

FIRST REVIEW ALGORITHM

Example: recursion

```
int sumton(int n){ // n>0
    if (n == 0)
        return 0;
    else
        return n + sumton(n-1);
}
```

```
sumton:
      #  if n == 0 then branch to base case
      # else push the two items onto the stack:
      #       return address ($ra)
      #       argument n ($a0)
      #
      # compute  argument (n-1) for next call
      # jump and link to sumton

      # load the return address  (pop)
      # load argument from the stack (pop)
      # change the stack pointer

       # register $v0 contains result of sumton(n-1)
      # add argument n  to $v0
      # return to parent

basecase:
     #  assign 0 as the value in $v0
     # return to parent
```

Assignment 3

Given a list of N integers, return the one that is "of rank k", that is, the integer that would be at index k in a sorted version of the list.

Let L be the list. Choose an element from L and call it "pivot".

Partition the list into three lists: L1, L2, L3 such that:

- L1 contains all the elements less than pivot
- L2 contains all the element equal to pivot
- L3 contains all the elements greater than pivot.

What then? (recursion)

Which data structure to use for L1, L2, L3 ?

Announcements

- A2 grading scheme (small change)
- Quiz 3 and yellow stickies (mean ~70%)
- due date for A3 is Wed after study break