All Together: Instruction Memory + Arithmetic Machine

Lab 4 debrief – Timing and control will come back in a big way for exam 3 and when we discuss pipelining. Exam 2 (FSMs) is around the corner

Office hours @ 1pm today

Discussion section timeliness

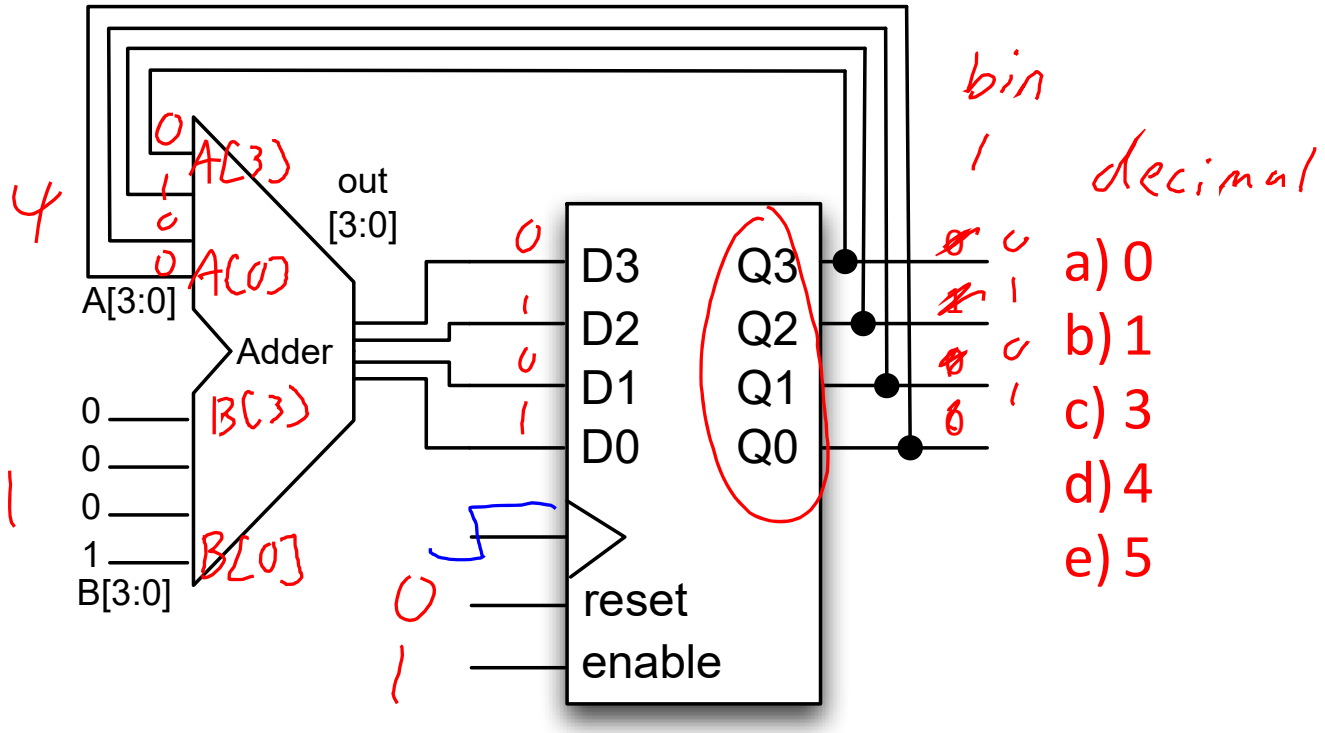
9/20 Clicker data not recorded…grrrr

Handout has 2 pages, grab both

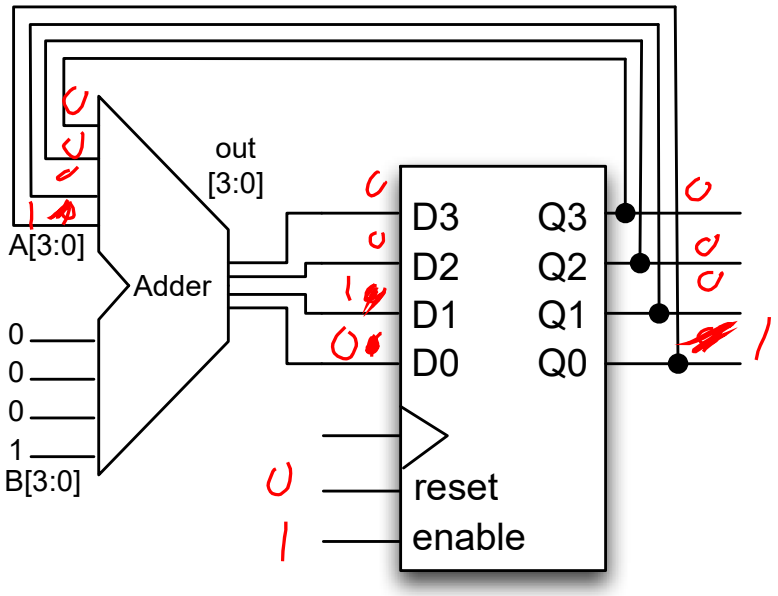
Today's lecture

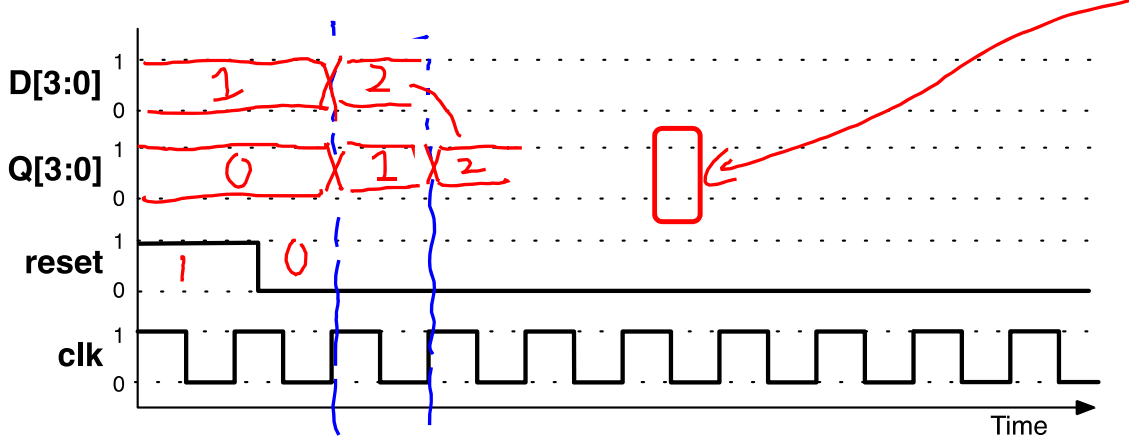
- Instructions control the datapath
  - Instruction Memory
  - Program Counter (PC) is the address unit for instruction memory
  - Adder
- Putting all together
  - Arithmetic unit to work

What will Q[3:0] be during the next clock cycle?



mod-16 Counter





a) 0x0
b) 0x2
c) 0x4
d) 0x6
e) 0x8

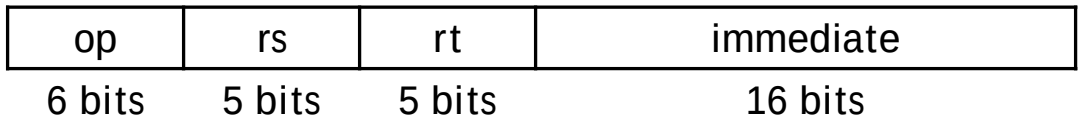
Previously…

- Register-to-register arithmetic instructions use the R-type format.

add $5, $10, $4

- Instructions with immediates all use the I-type format.

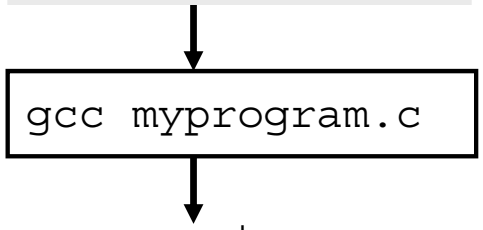
ori $7, $2, 0x00ff



Where are the instructions my program executes?

myprogram.c

```
void main() {
  int a = 0;
  int b = a+5;
}
```



To look at the assembly code of a.out:

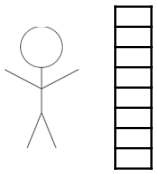
```
$ objdump -d a.out
```

The instructions executed by the program are in the .text section:

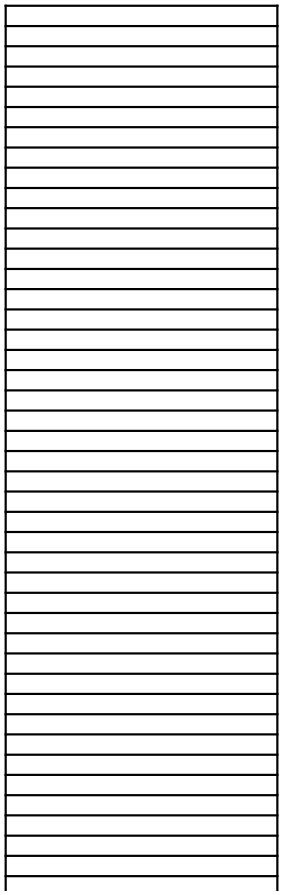
```
.text
main:
  addi $1, $0, 5
```

Programs require memory structures that are much larger than register files

Register file



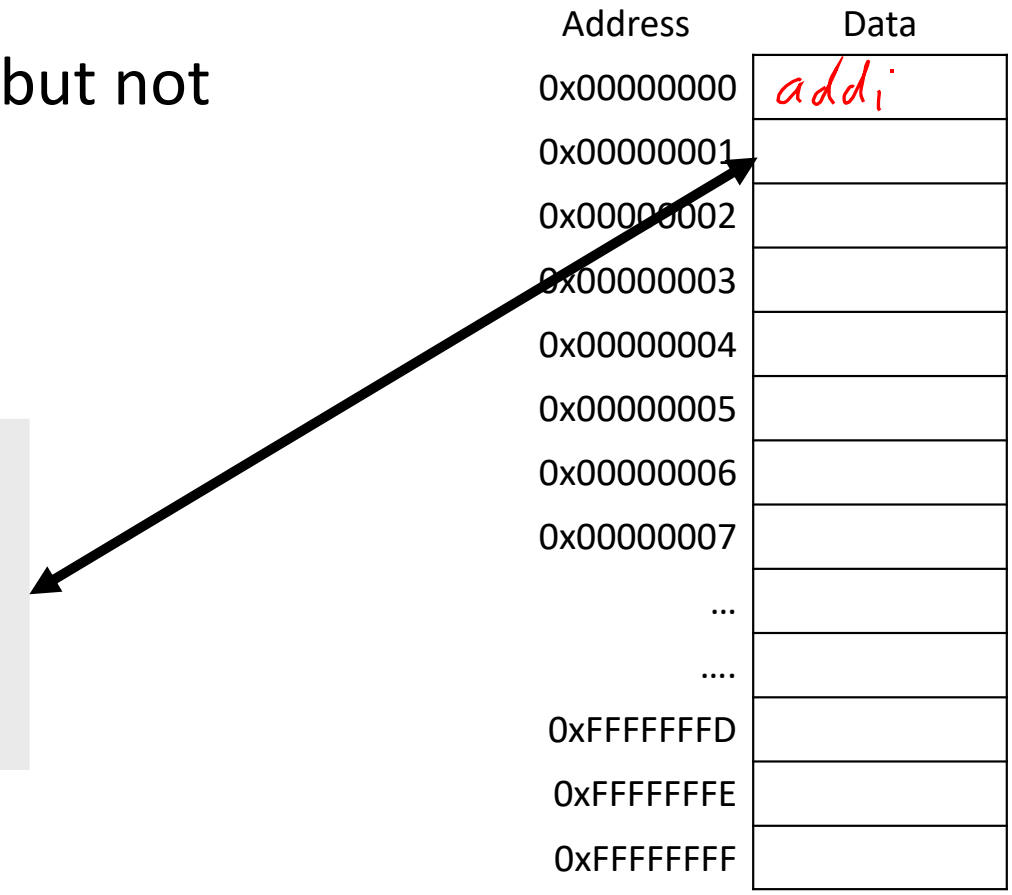
Memory



Programs are stored in an instruction memory

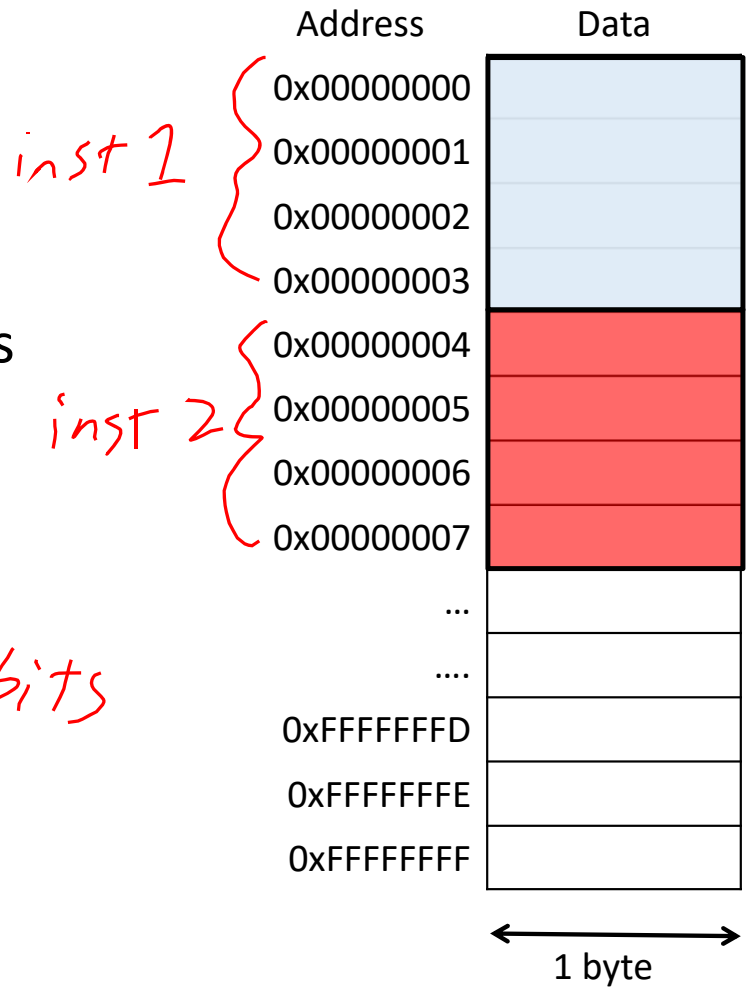
We will read the memory but not modify it

```
.text
main:
addi $1, $0, 5
sub  $2, $1, $3
```



The instruction memory is byte addressable

- Addresses are 32-bits
  - # addresses: 2^{32} = 4 G
- Each address contains 1 byte
  - Instructions occupy four contiguous locations
- Memory stores 4Gbytes



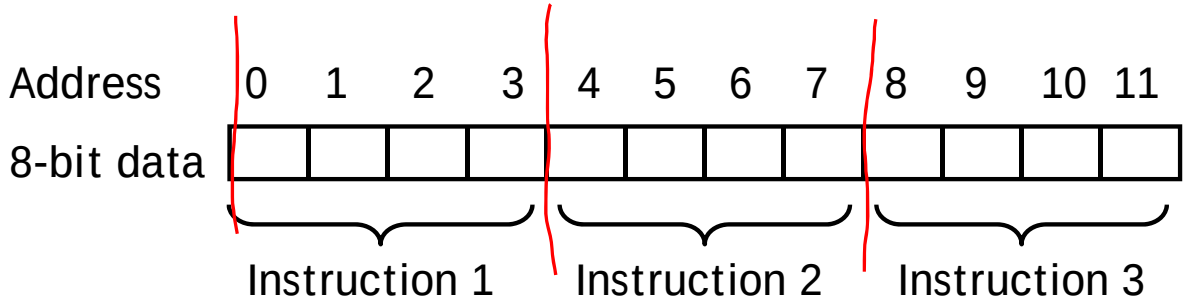
instructions – 32 bits

MIPS instructions start at an address that is divisible by 4

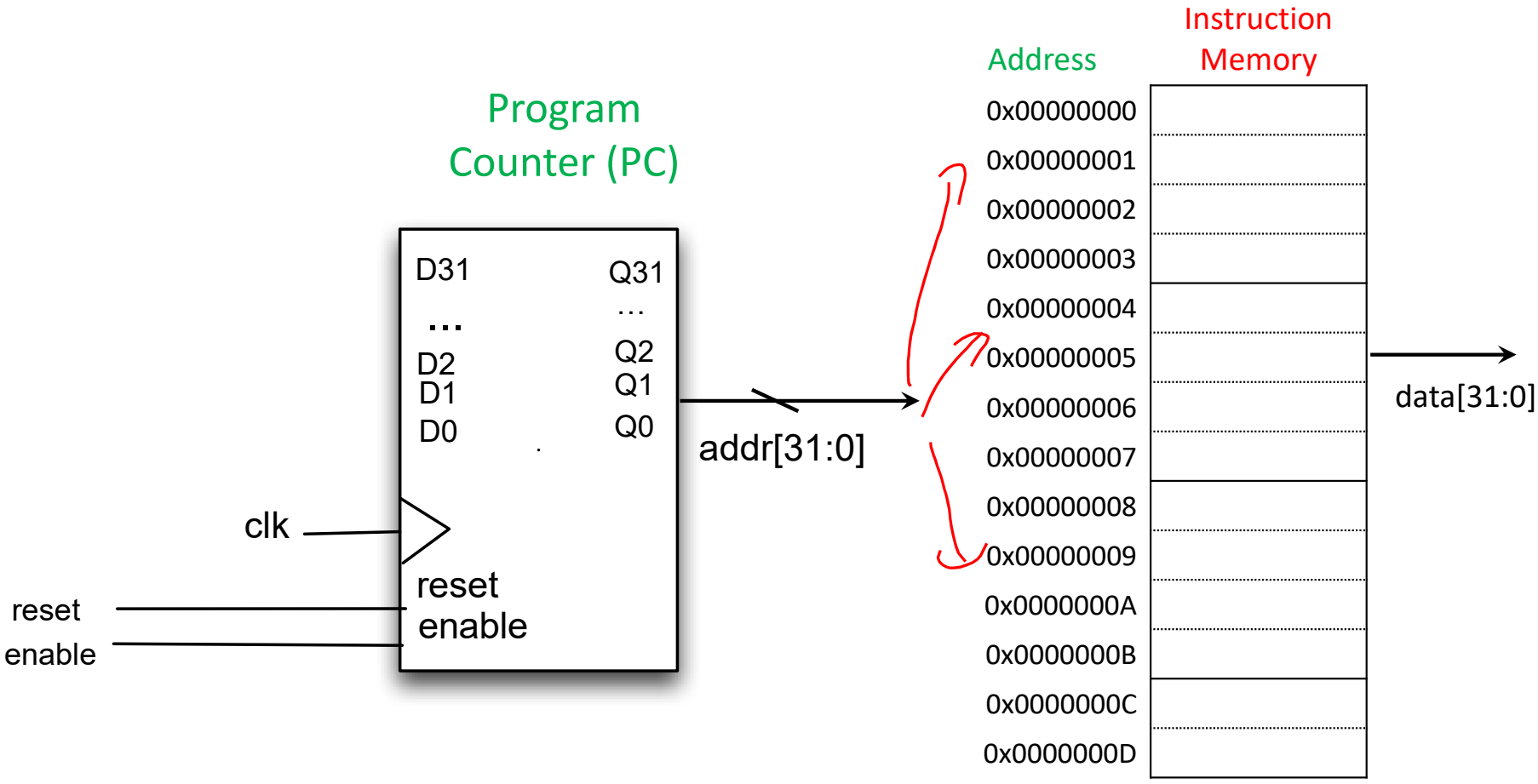
- 0, 4, 8 and 12 are valid instruction addresses.
- 1, 2, 3, 5, 6, 7, 9, 10 and 11 are *not* valid instruction addresses.

0 0000
4 0100
8 1000
12 1100

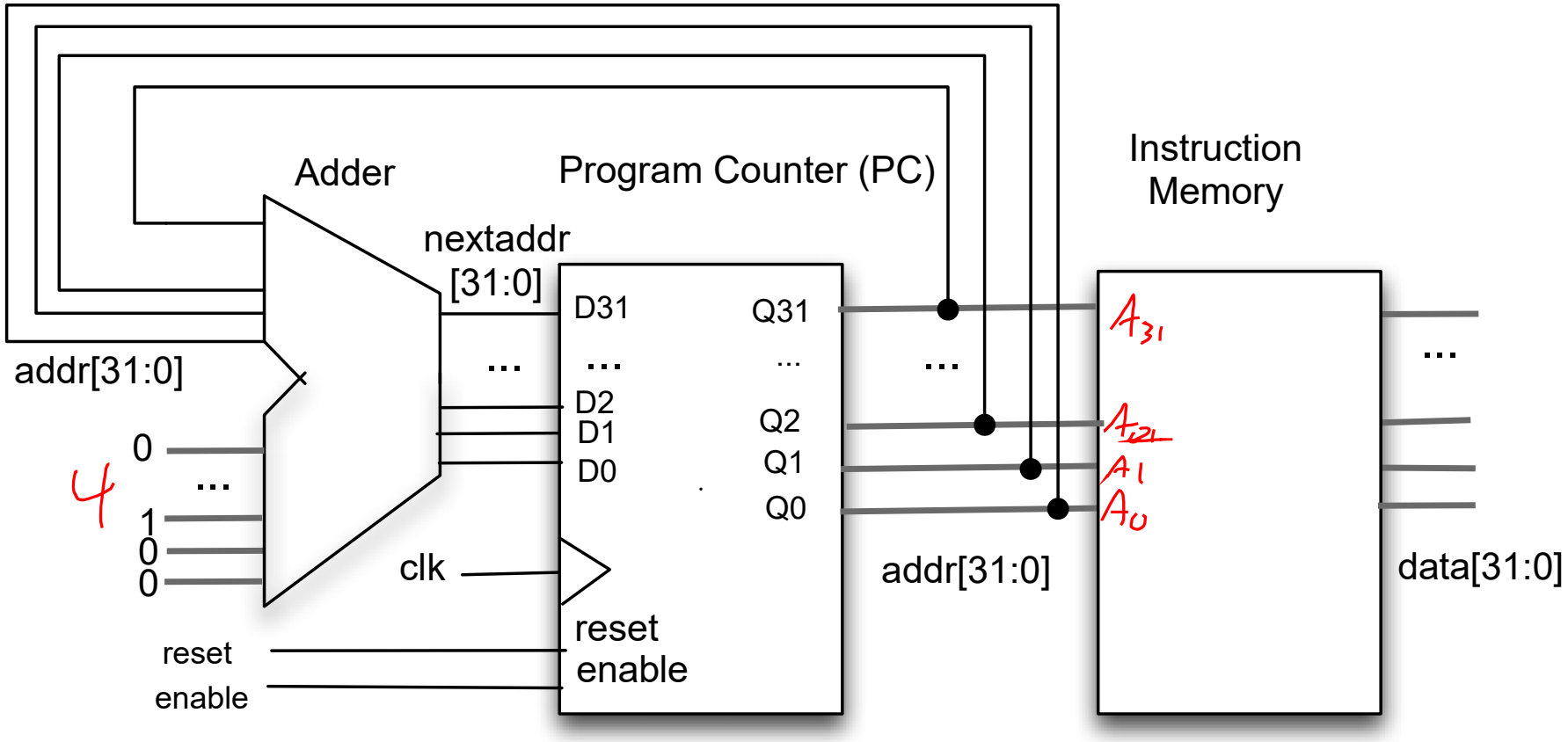
word alignment



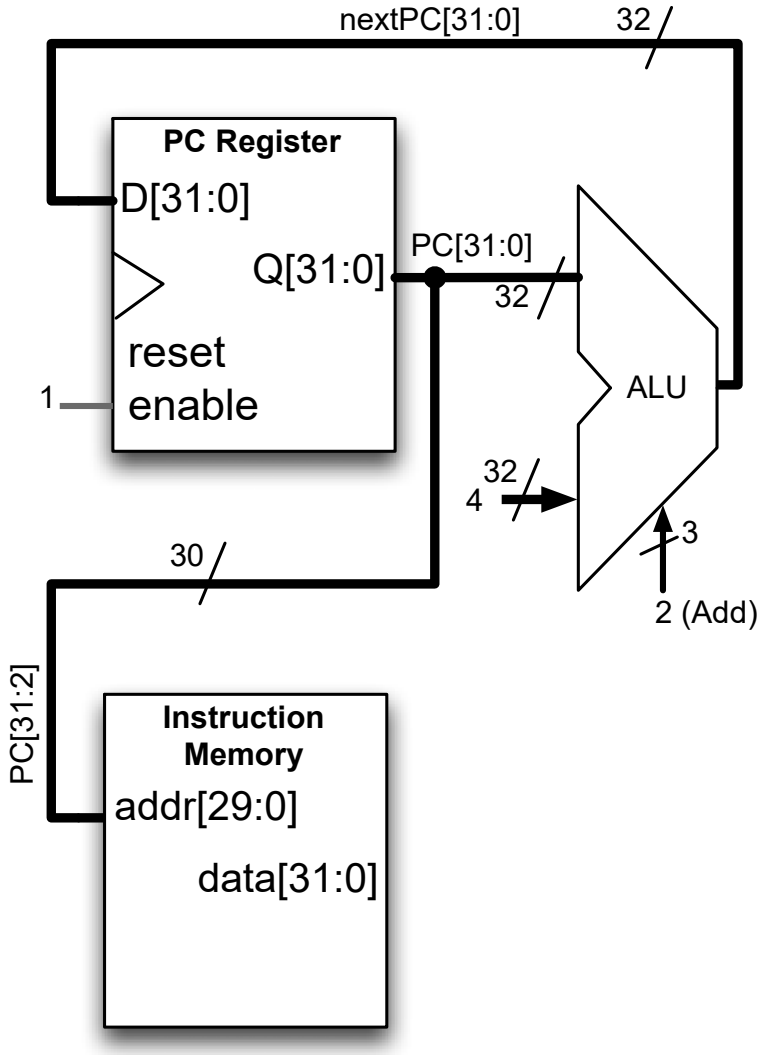
A special register called Program Counter (PC) contains the address of the next instruction to execute



Use an adder to increment PC to the next instruction



Redrawn to match the MIPS diagram



i>clicker

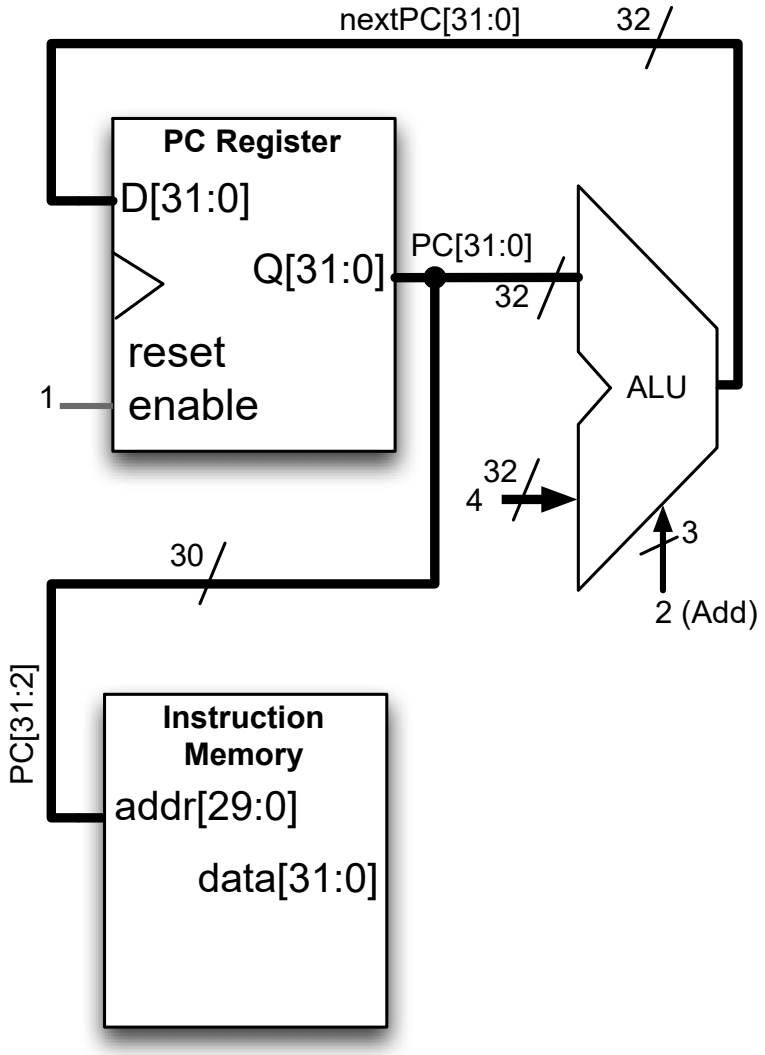
Why aren't 2 LSbs provided?

a) Bug in the slide

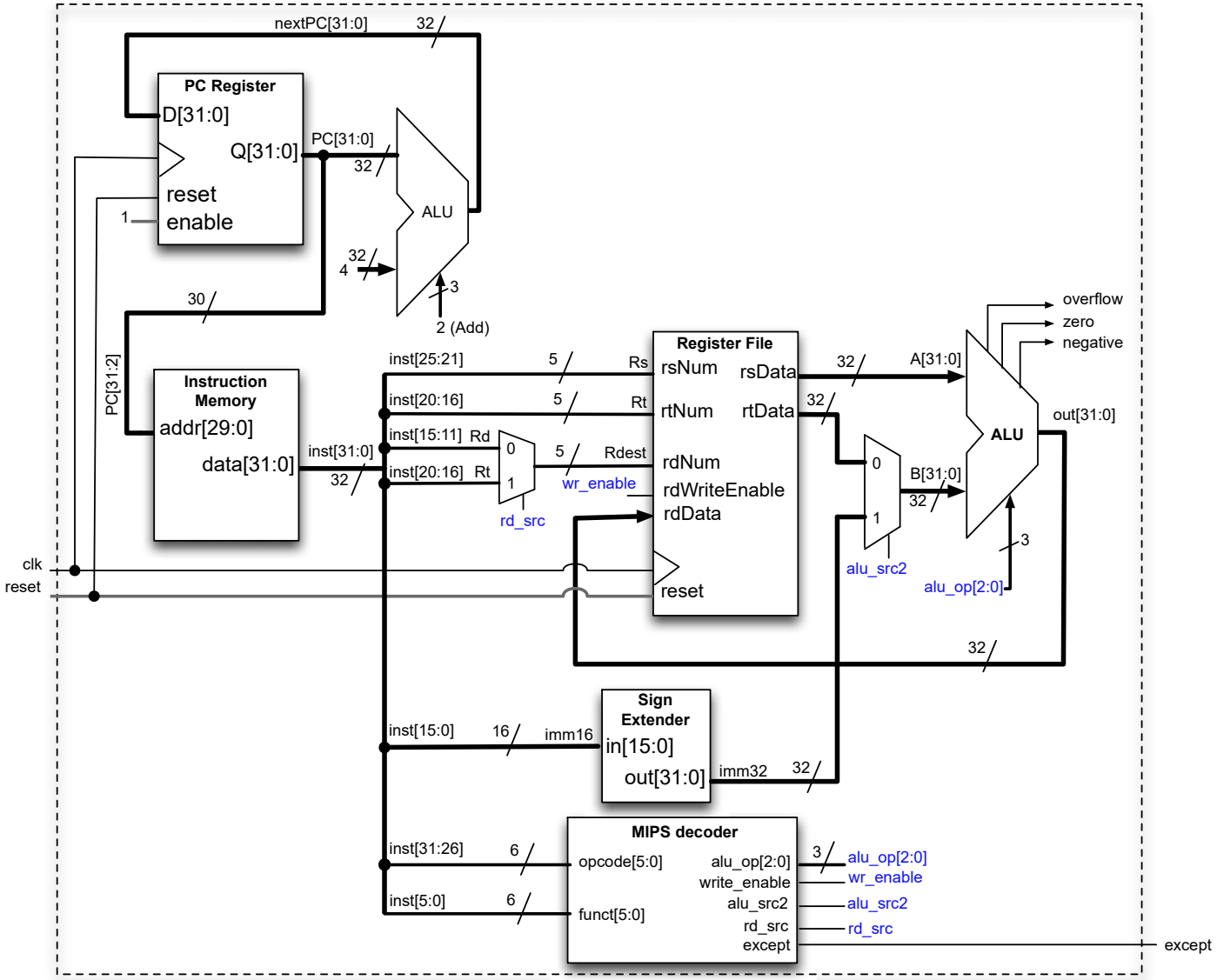
b) Memory is only 2^{30} big

c) Bits [1:0] are always 2'b00

d) Velociraptors ate them



MIPS datapath with a controlling instruction memory and program counter



Example

My program

$3 = 10
$5 = -7
$7 = $3 + $5

Assembly

What value will be stored in register 7 at the end of the program?

a) -7
b) 3
c) 5
d) 8
e) 10

Example

i-clicker

My program

$3 = 10
$5 = -7
$7 = $3 + $5

Assembly

Answer A
addi $3, $0, 0x000A
subi $5, $0, 0x0007
add $7, $3, $5

Answer B
addi $3, $0, 0x000A
addi $5, $0, 0xFFF9
add $7, $3, $5

Answer C
addi $3, $0, 0x000A
addi $5, $0, 0xFFF8
add $7, $3, $5

Answer D
add $3, $0, 0x000A
sub $5, $0, 0x0007
add $7, $3, $5

Example

My program

$3 = 10
$5 = -7
$7 = $3 + $5

Assembly

addi $3, $0, 0x000A
addi $5, $0, 0xFFF9
add $7, $3, $5

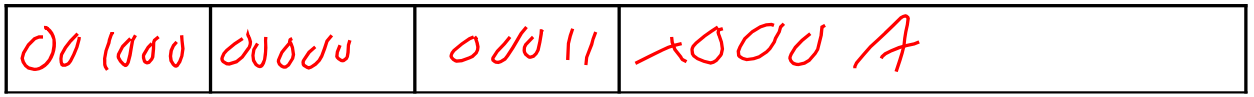
	opcode	funct
add	0x00	0x20
addi	0x08	

Machine code

a) 00000
b) 00011
c) 00101
d) 00111

rt rs ima

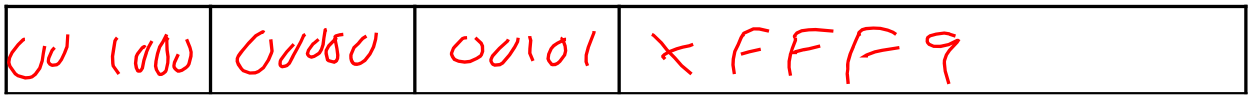
```
addi  $3, $0, 0x000A
```



op rs rt imm

rt rs

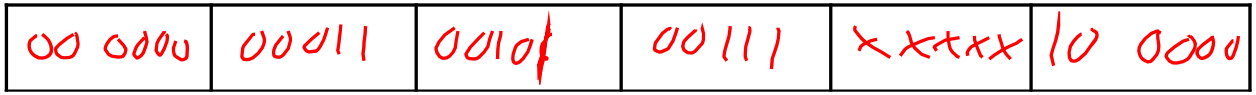
```
addi  $5, $0, 0xFFF9
```



op rs rt imm

rd rs rt

```
add   $7, $3, $5
```



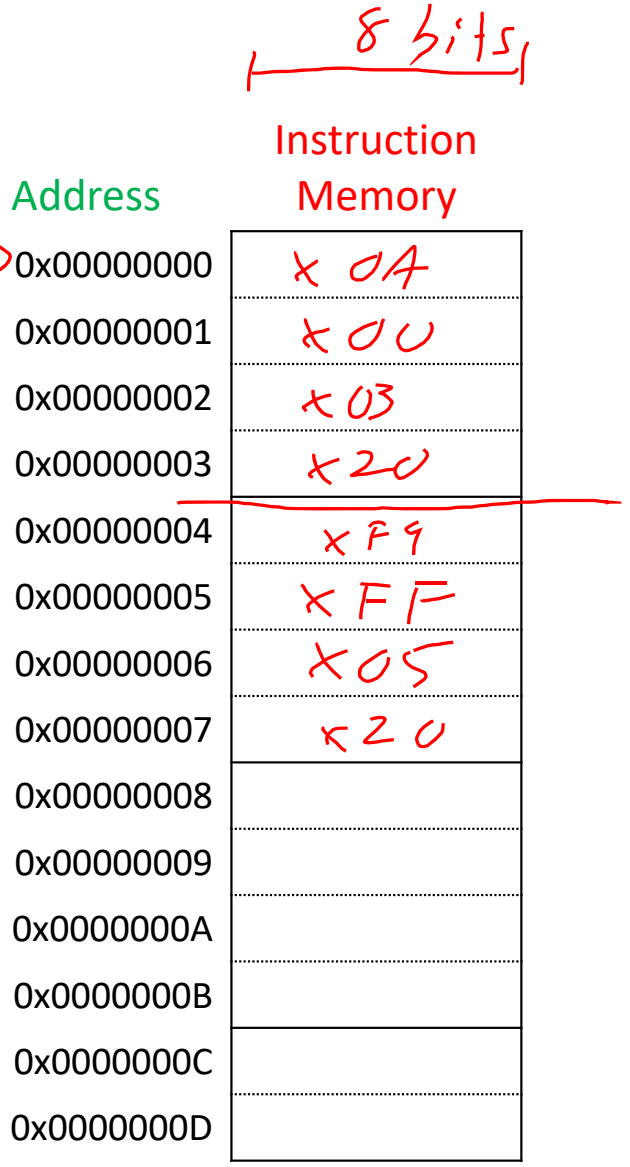
op rs rt rd shamt funct

Little Endian - Least significant bits (little end) go first

Assembly: addi $3, $0, 0x000A
Machine: 0x2003000A

Assembly: addi $5, $0, 0xFFF9
Machine: 0x2005FFF9

Assembly: add $7, $3, $5
Assembly: 0x00C53820

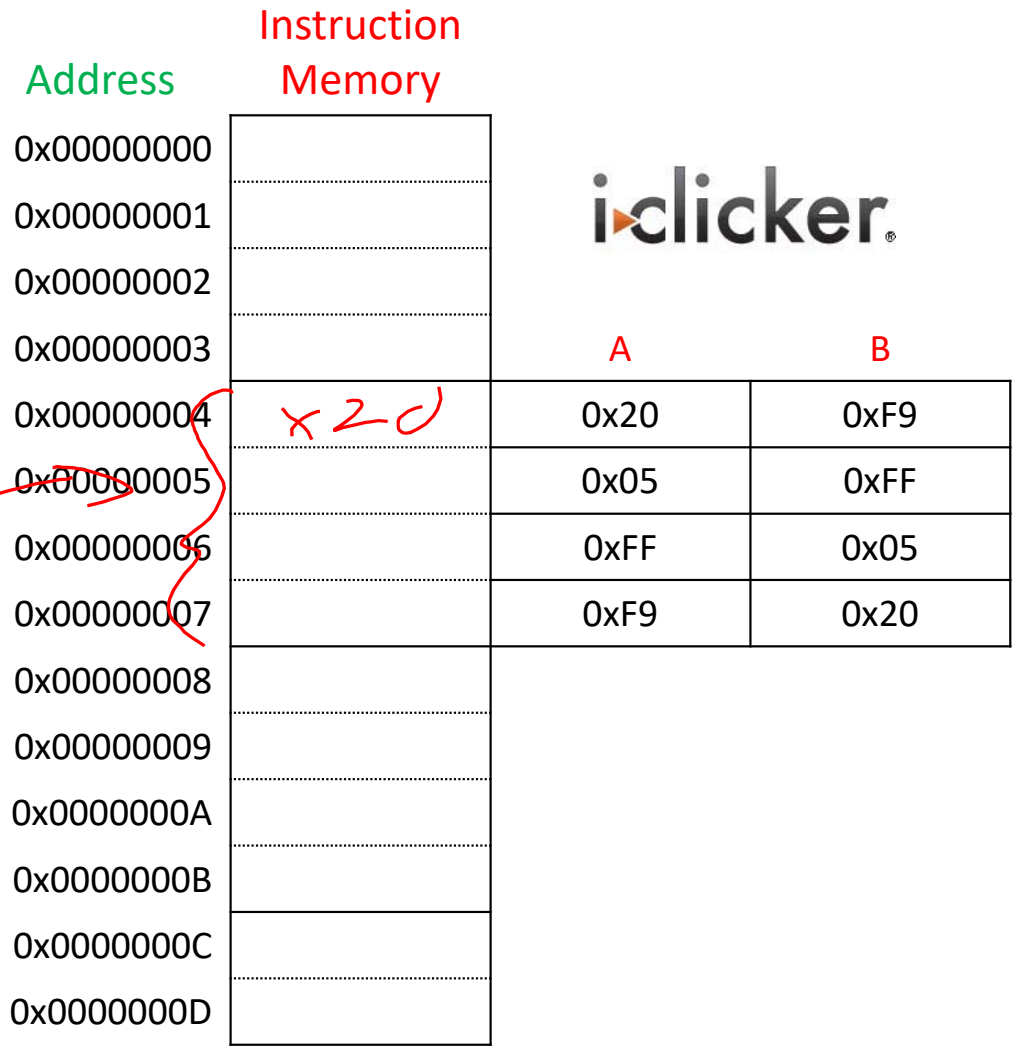


Big Endian – Most significant bits (big end) go first

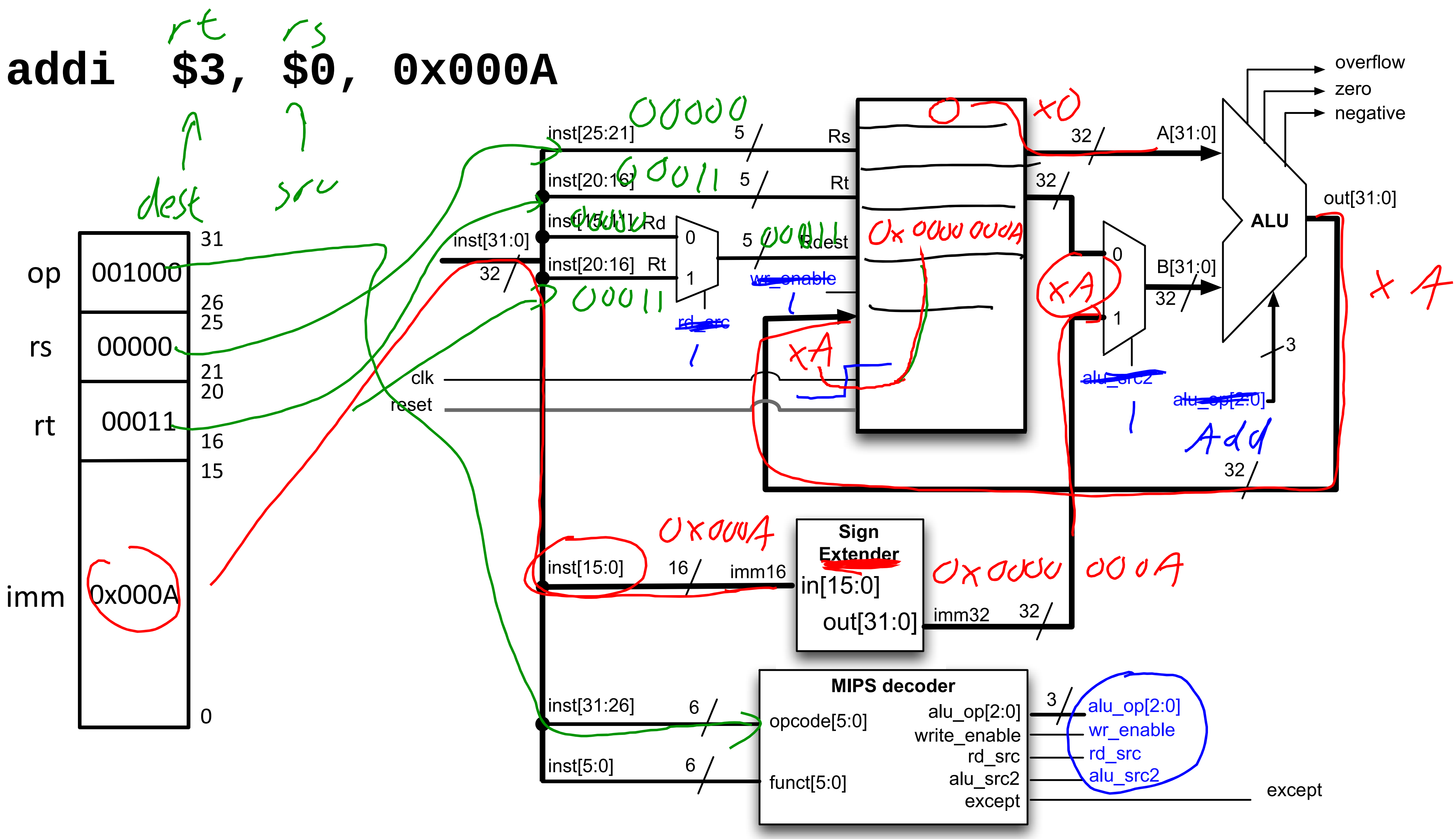
Assembly: addi $3, $0, 0x000A
Machine: 0x2003000A

Assembly: addi $5, $0, 0xFFF9
Machine: 0x20 05FFF9

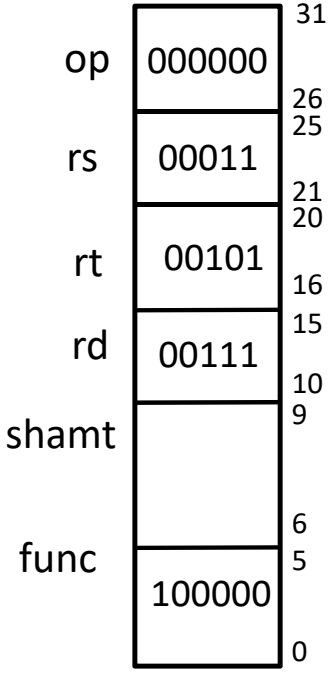
Assembly: add $7, $3, $5
Assembly: 0x00C53820

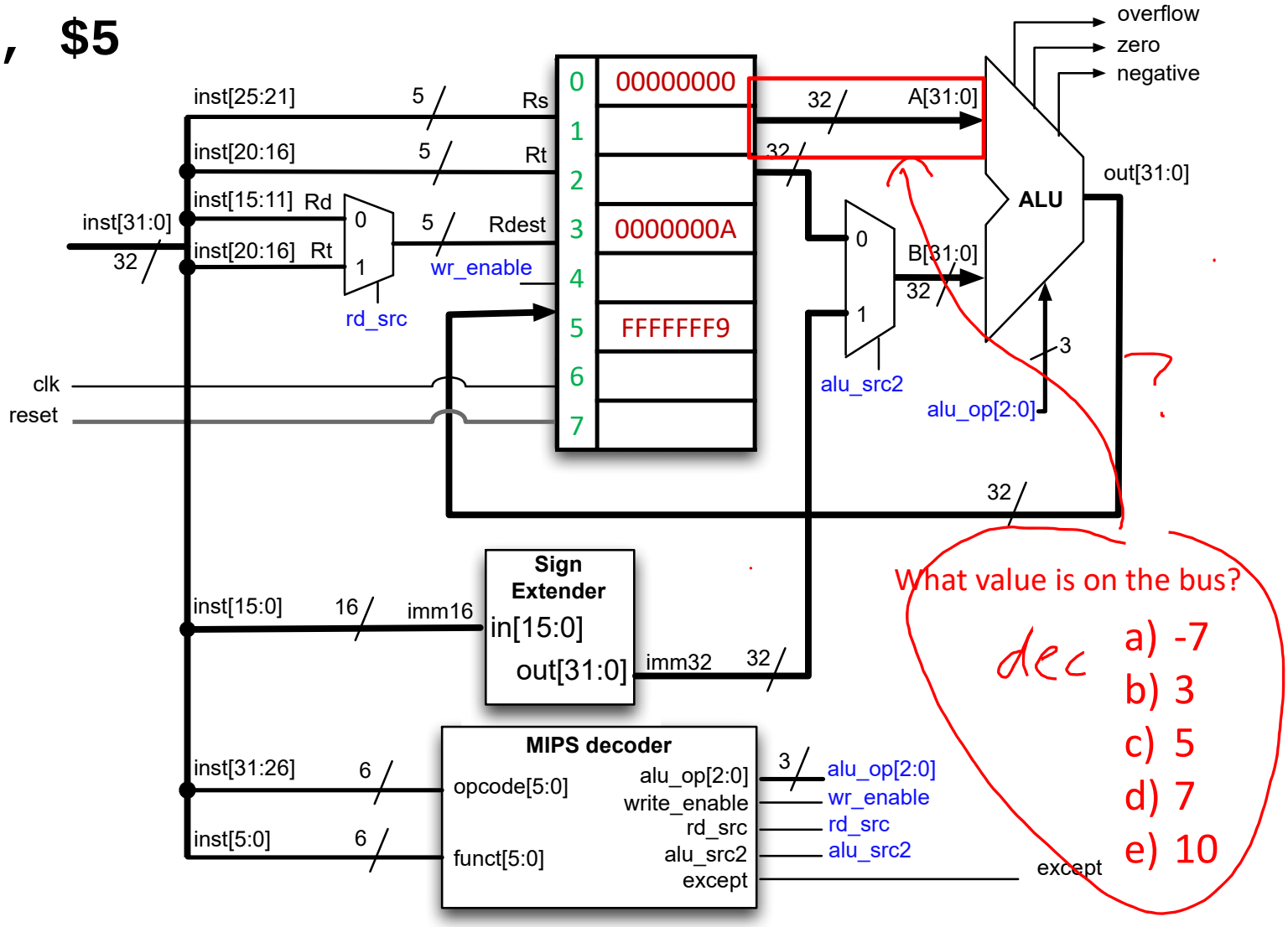


addi $3, $0, 0x000A



add $7, $3, $5





What value is on the bus?

dec

a) -7
b) 3
c) 5
d) 7
e) 10