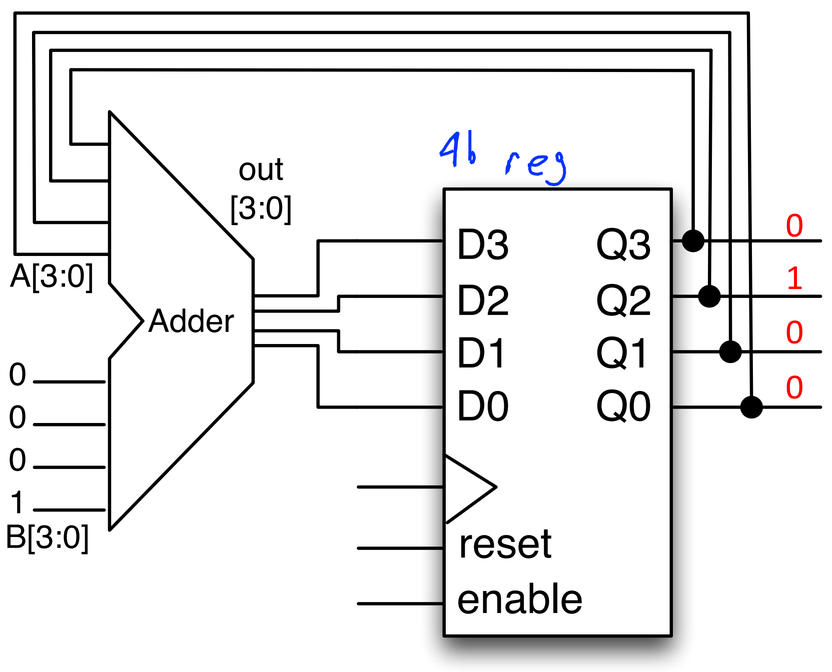
All Together: Instruction Memory + Arithmetic Machine

PICK UP HANDOUT !! ☺

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?



0100
+1   1

0101

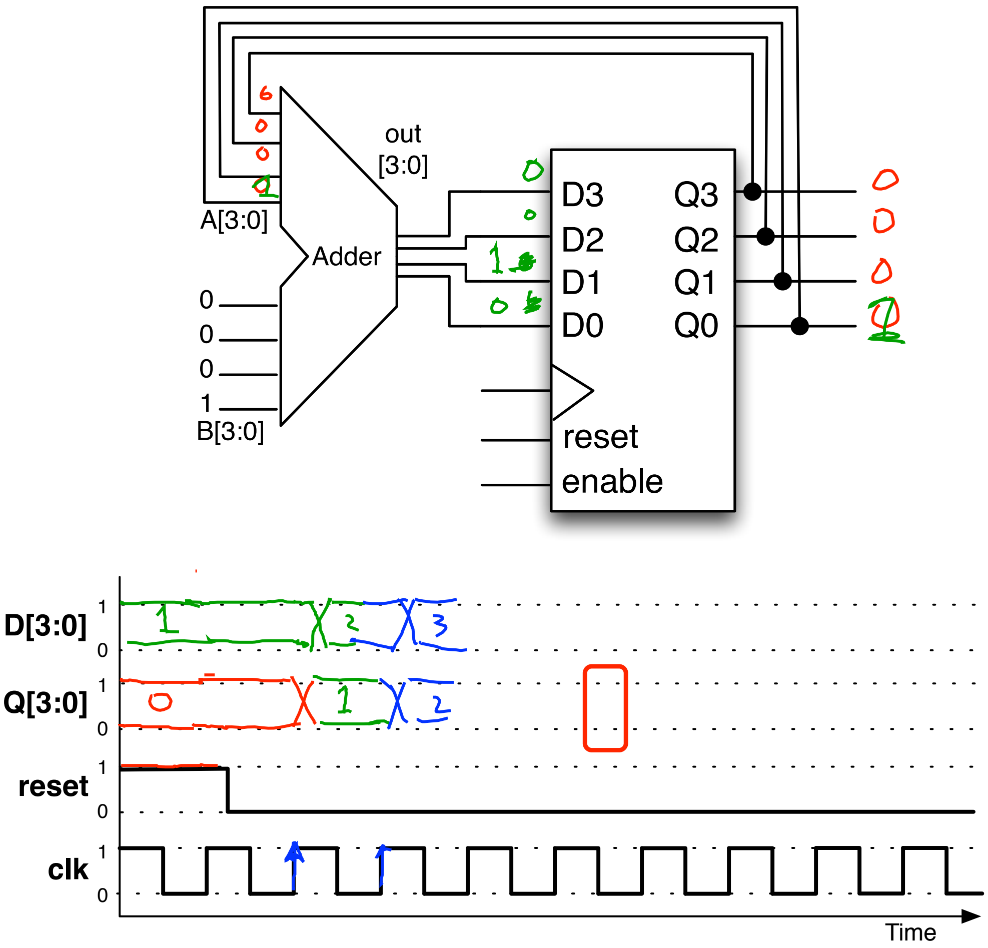
a) 0

b) 1

c) 3

d) 4

e) 5



a) 0x0

b) 0x2

c) 0x4

d) 0x6

e) 0x8

Previously…

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

3 register specifiers

add $5, $10, $4

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

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

immediate

ori $7, $2, 0x00ff

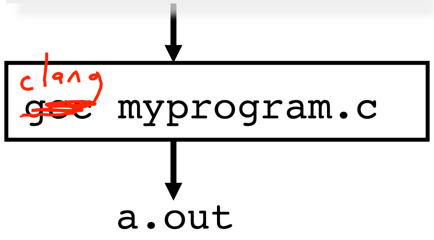
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

32b = 4B

Where are the instructions my program executes?

myprogram.c

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



~~gcc~~ clang myprogram.c

a.out

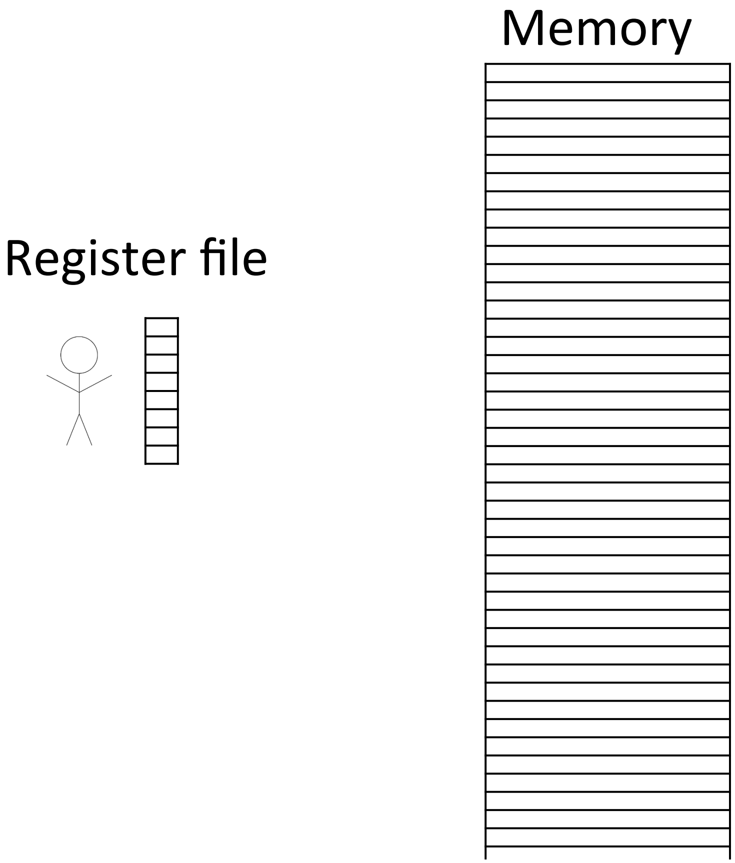
To look at the assembly code of a.out:

```
$ 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

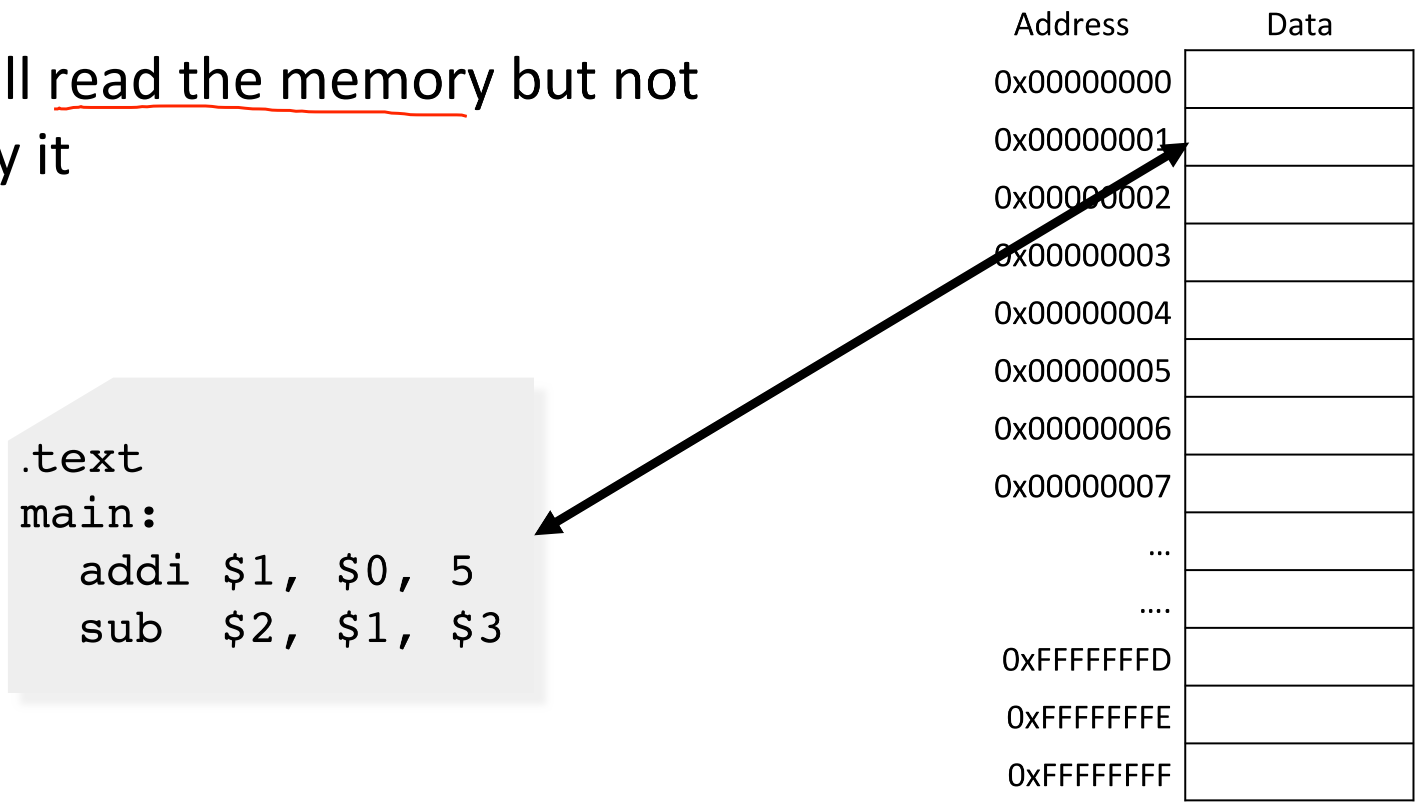


Memory

Register file

Programs are stored in an instruction memory

We will read the memory but not modify it



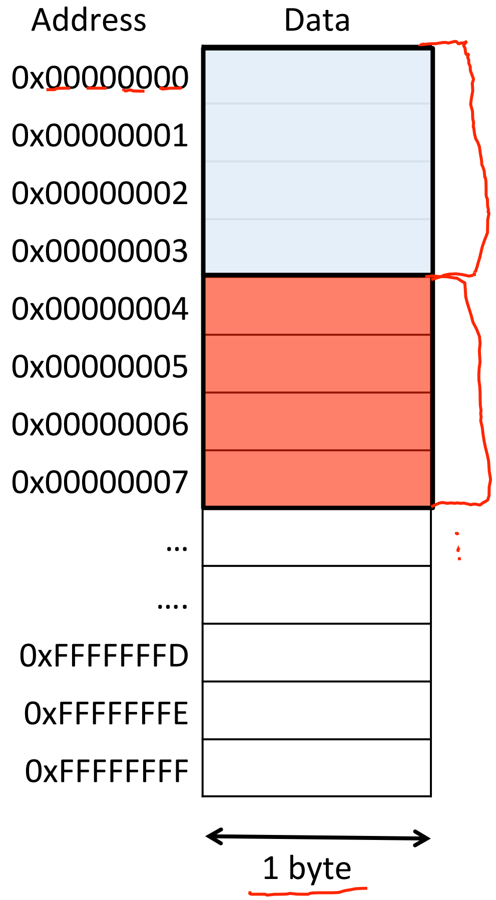
```
.text
main:
  addi $1, $0, 5
  sub  $2, $1, $3
```

Address	Data
0x00000000	
0x00000001	
0x00000002	
0x00000003	
0x00000004	
0x00000005	
0x00000006	
0x00000007	
…	
….	
0xFFFFFFFD	
0xFFFFFFFE	
0xFFFFFFFF	

The instruction memory is byte addressable

- Addresses are 32-bits
  - # addresses: 2^{32} = 4 Billion
- Each address contains 1 byte
  - Instructions are 4 bytes MIPS
    - occupy four contiguous locations
- Memory stores 4 GB (gigabytes)

1 Byte = 8 bits



Address Data

0x00000000
0x00000001
0x00000002
0x00000003
0x00000004
0x00000005
0x00000006
0x00000007
...
....
0xFFFFFFFD
0xFFFFFFFE
0xFFFFFFFF

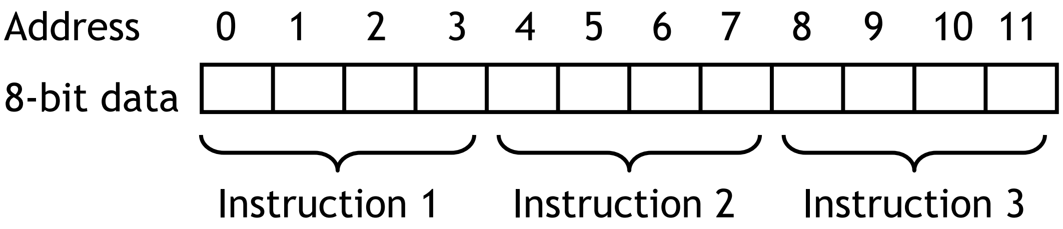
1 byte

MIPS instructions start at an address that is divisible by 4

word aligned

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

word = 4 B (32b machine)



Address 0 1 2 3 4 5 6 7 8 9 10 11

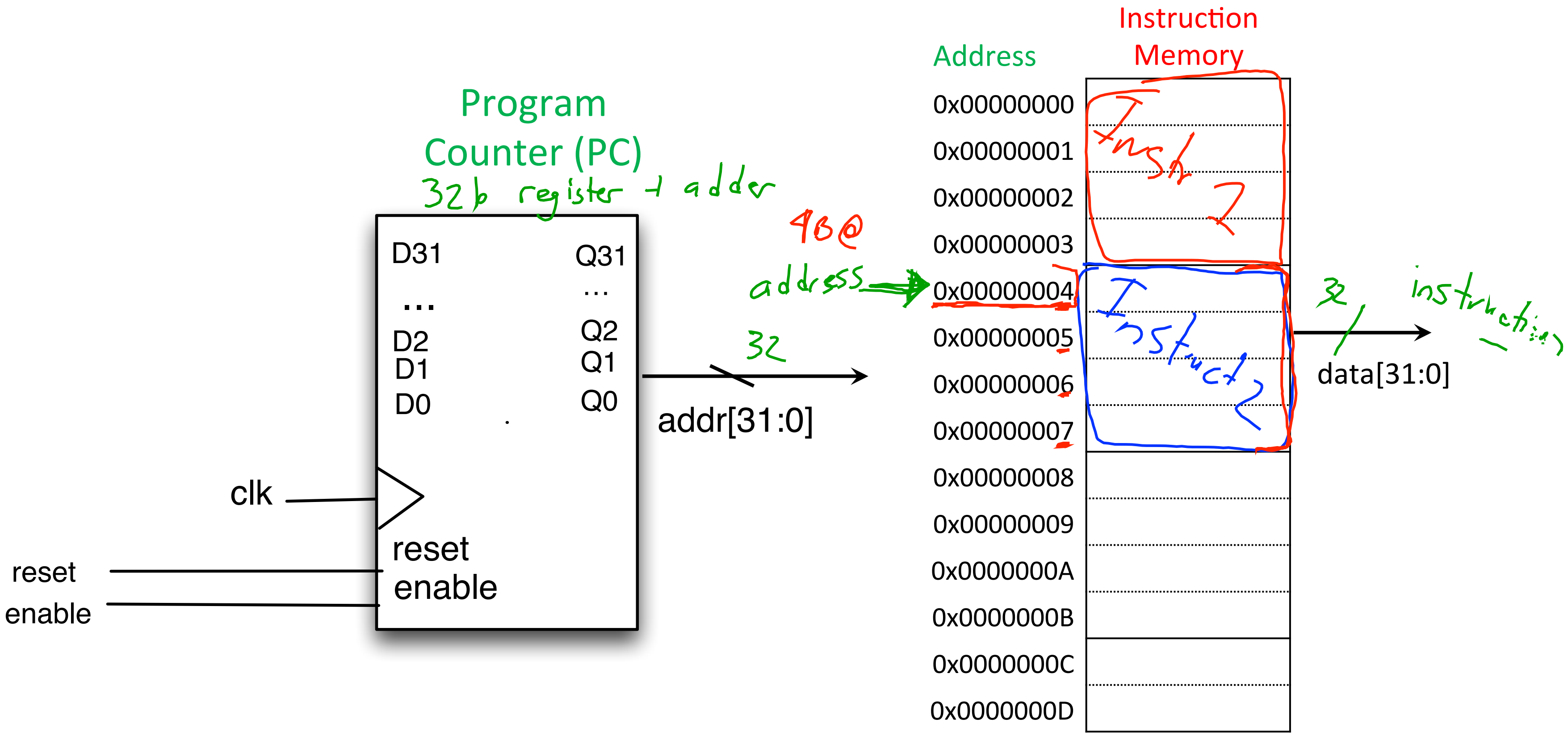
8-bit data

Instruction 1

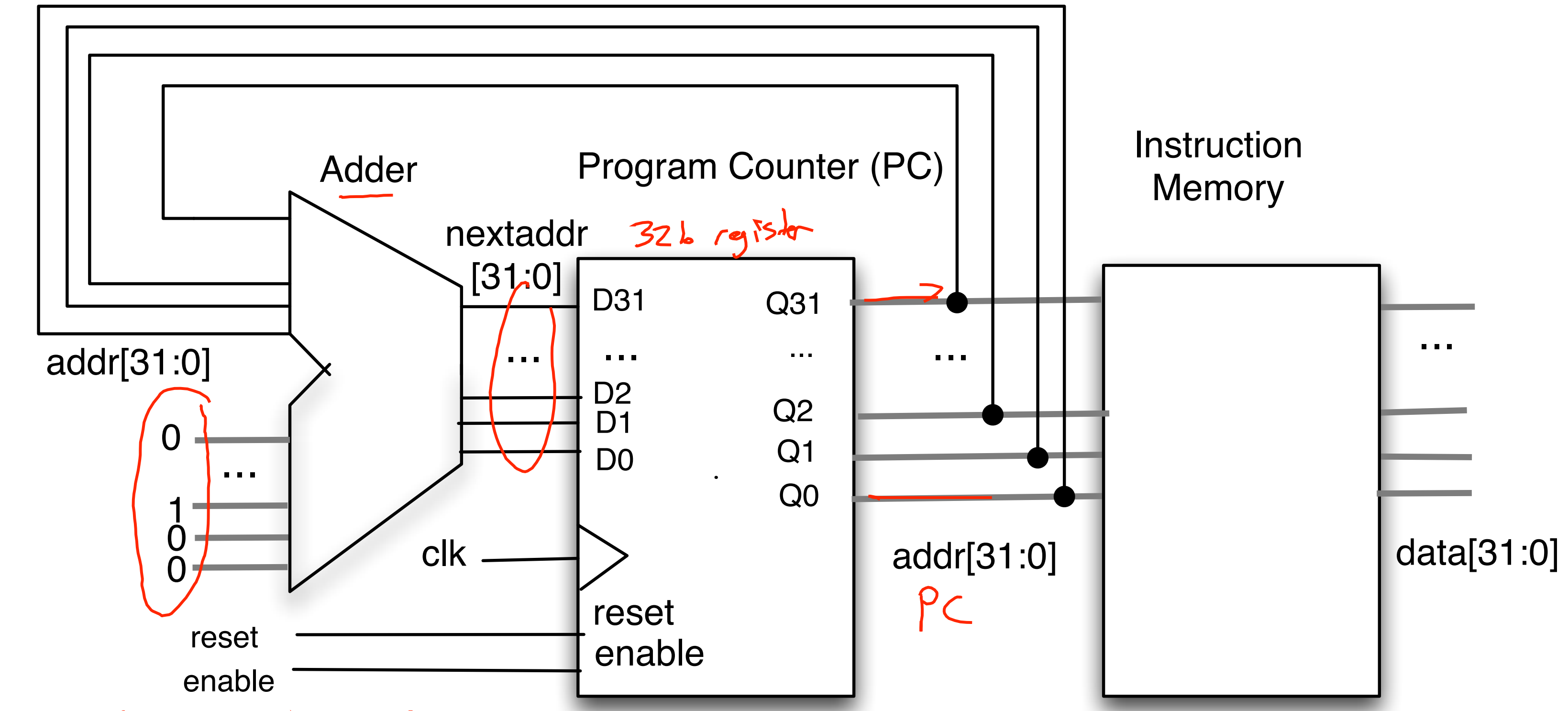
Instruction 2

Instruction 3

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



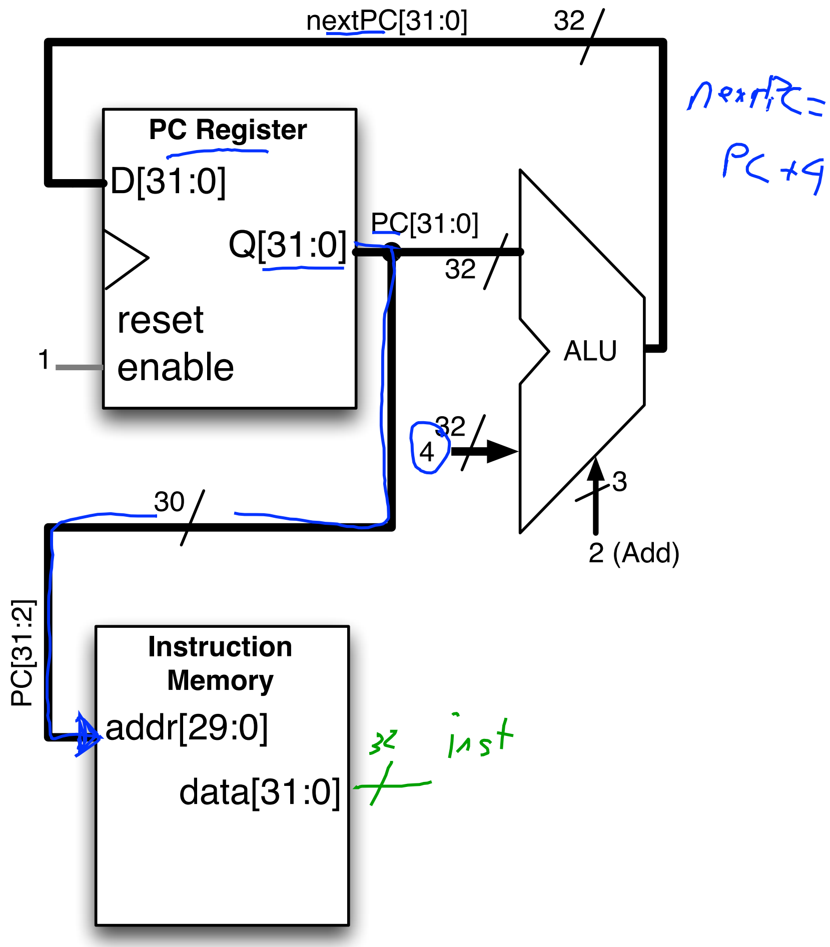
Use an adder to increment PC to the next instruction



nextaddr = addr + 4

= PC + 4

Redrawn to match the MIPS diagram



i>clicker

Why aren’t 2 LSbs provided?

a) Bug in the slide

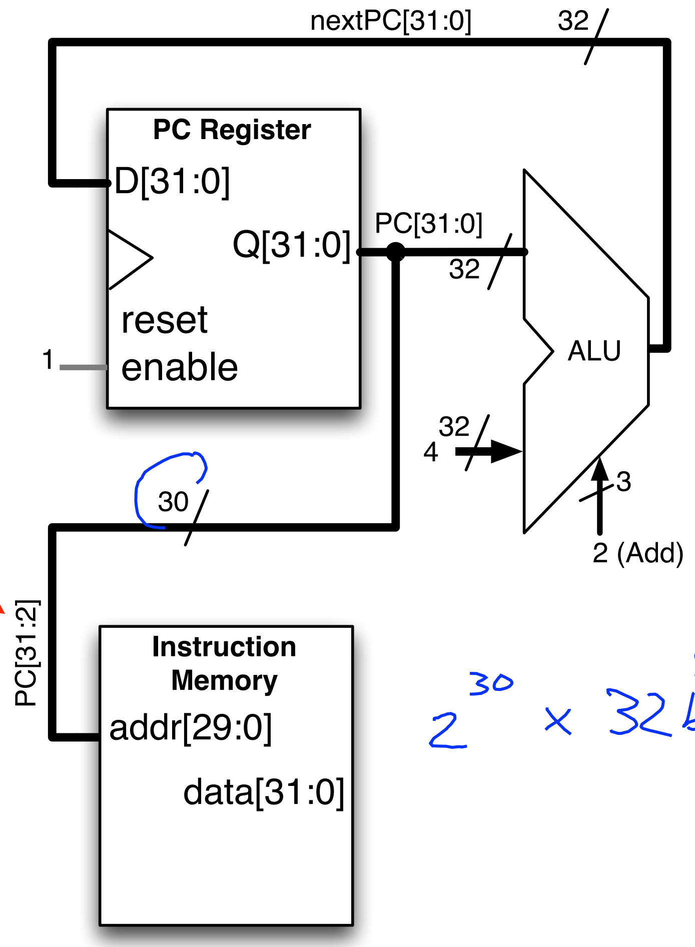
b) Memory is only 2^{30} big

c) Bits [1:0] are always 2’b00

d) Velociraptors ate them

Memory

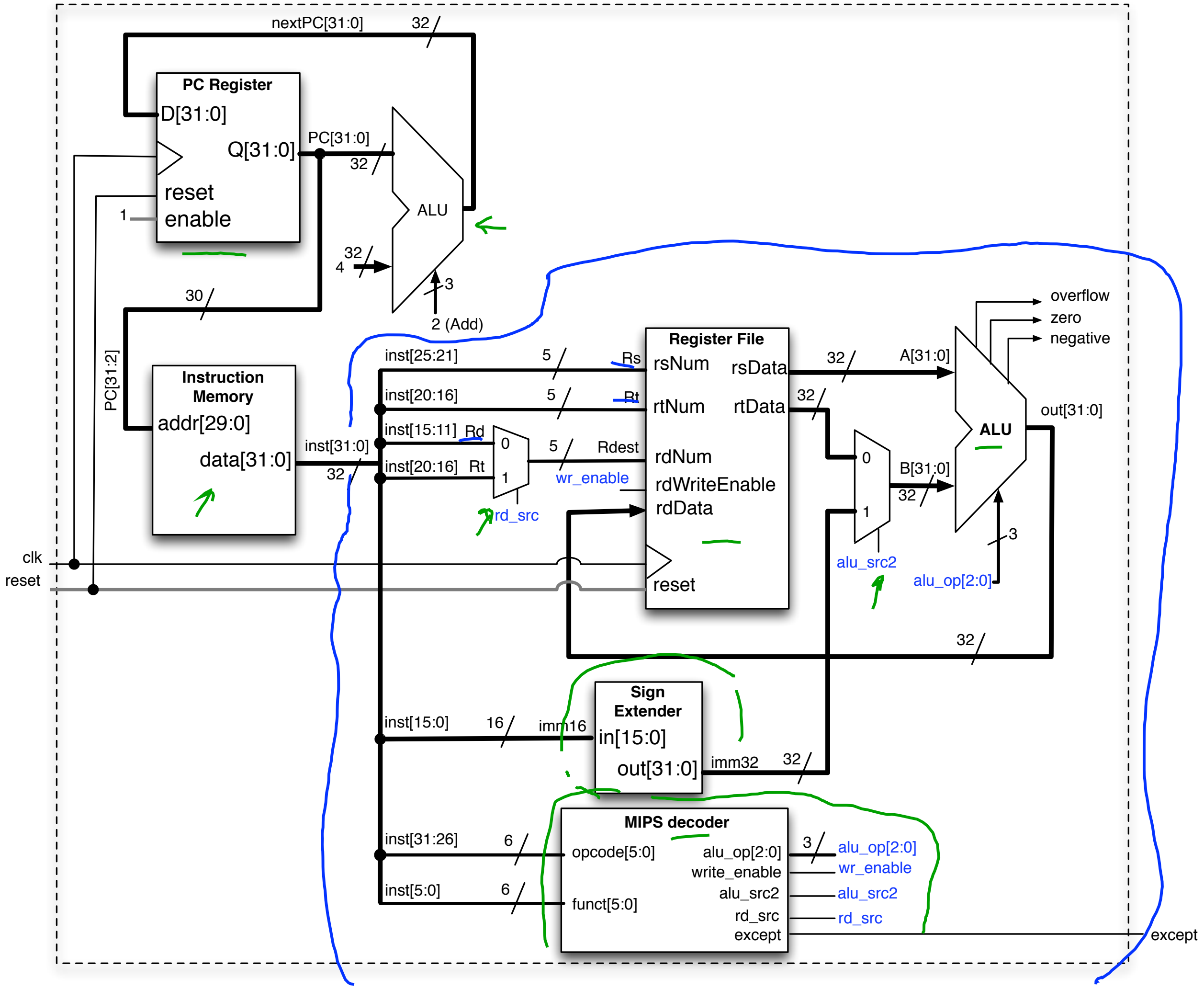
$2^{32} \times 8b$



$2^{30} \times 32b$

~~8~~ 4B

MIPS datapath with a controlling instruction memory and program counter



Example

My program

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

Assembly

```
addi   $3, $0, 10
addi   $5, $0, -7
add    $7, $3, $5
```

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

iclicker

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

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

My program

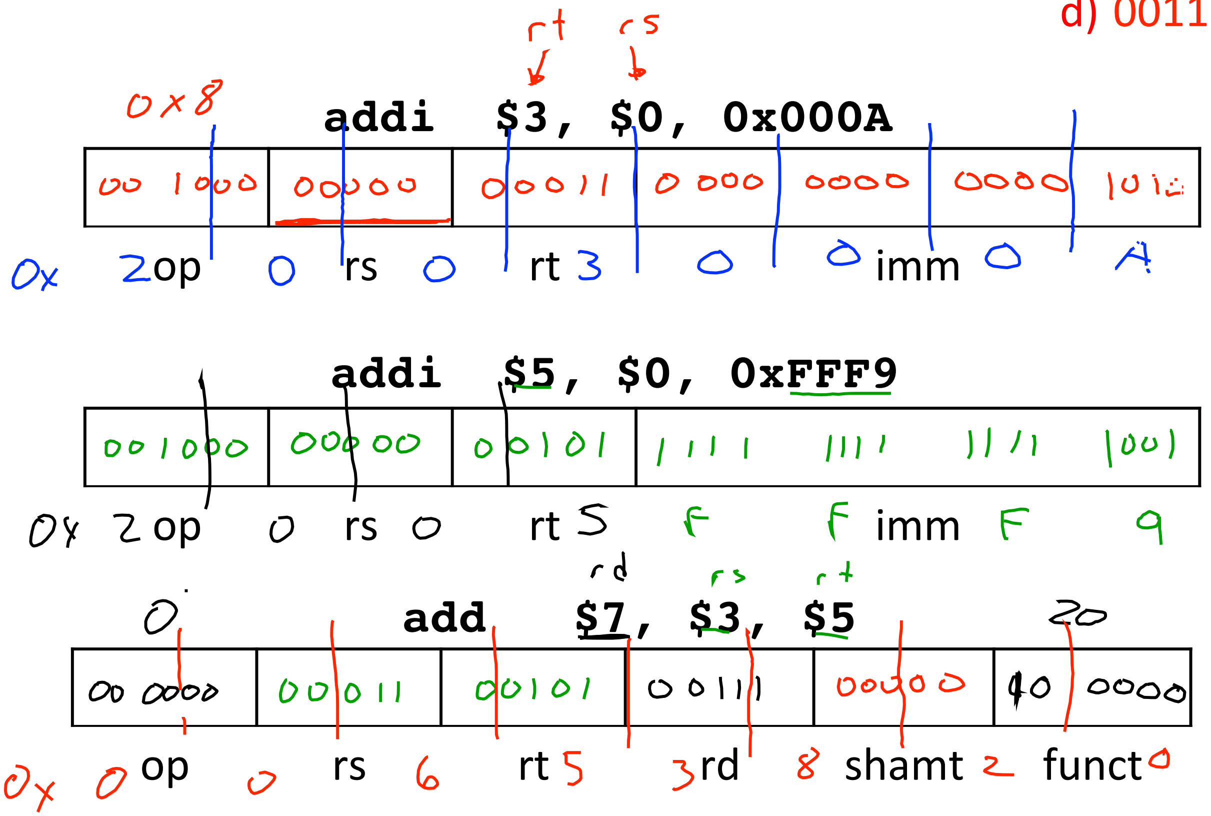
$3 = 10
$5 = -7
$7 = $3 + $5

Assembly

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

	opcode	funct
add	0x00	0x20
addi	0x08	

Machine code



rt rs
0x8
addi $3, $0, 0x000A
00 1000 | 00000 | 00011 | 0 000 | 0000 | 0000 | 1010
0x 2 op 0 rs 0 rt 3 0 0 imm 0 A

addi $5, $0, 0xFFF9
001000 | 00000 | 00101 | 1111 1111 1111 1001
0x 2 op 0 rs 0 rt 5 F F imm F 9

rd rs rt
0 20
add $7, $3, $5
00 0000 | 00011 | 00101 | 00111 | 00000 | 10 0000
0x 0 op 0 rs 6 rt 5 3 rd 8 shamt 2 funct 0

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

Address	Instruction Memory
0x00000000	0A
0x00000001	00
0x00000002	03
0x00000003	20
0x00000004	F9
0x00000005	FF
0x00000006	05
0x00000007	20
0x00000008	20
0x00000009	38
0x0000000A	C5
0x0000000B	00
0x0000000C	
0x0000000D	

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

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

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

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

int *p;
char c = *((char*)p);

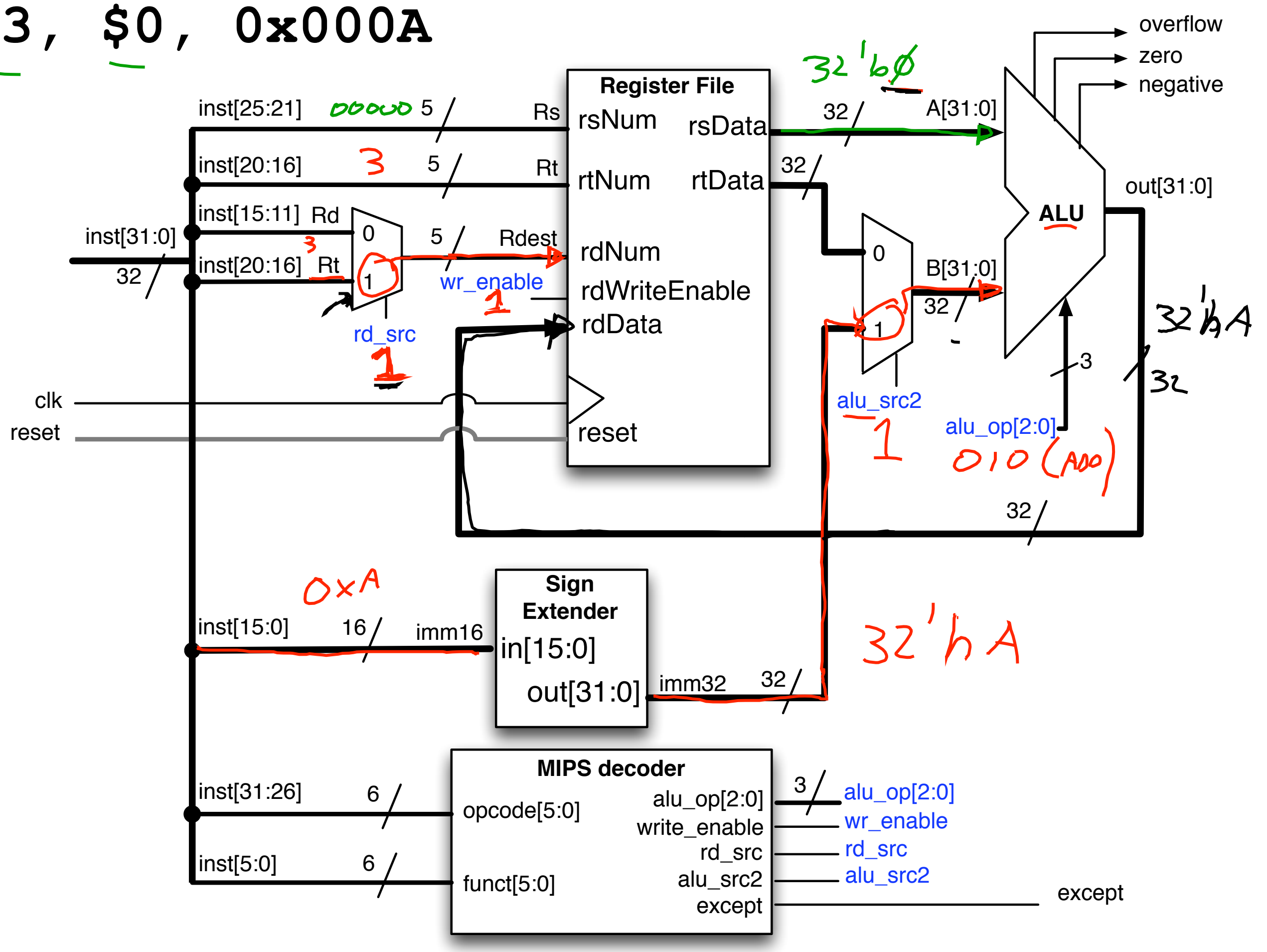
Address	Instruction Memory	A	B
0x00000000			
0x00000001			
0x00000002			
0x00000003			
0x00000004	20	0x20	0xF9
0x00000005	05	0x05	0xFF
0x00000006	FF	0xFF	0x05
0x00000007	F9	0xF9	0x20
0x00000008			
0x00000009			
0x0000000A			
0x0000000B			
0x0000000C			
0x0000000D			

rt rs

addi $3, $0, 0x000A

I-type

op	001000	31 26
rs	00000	25 21
rt	00011	20 16
imm	0x000A	15 0



Reg File

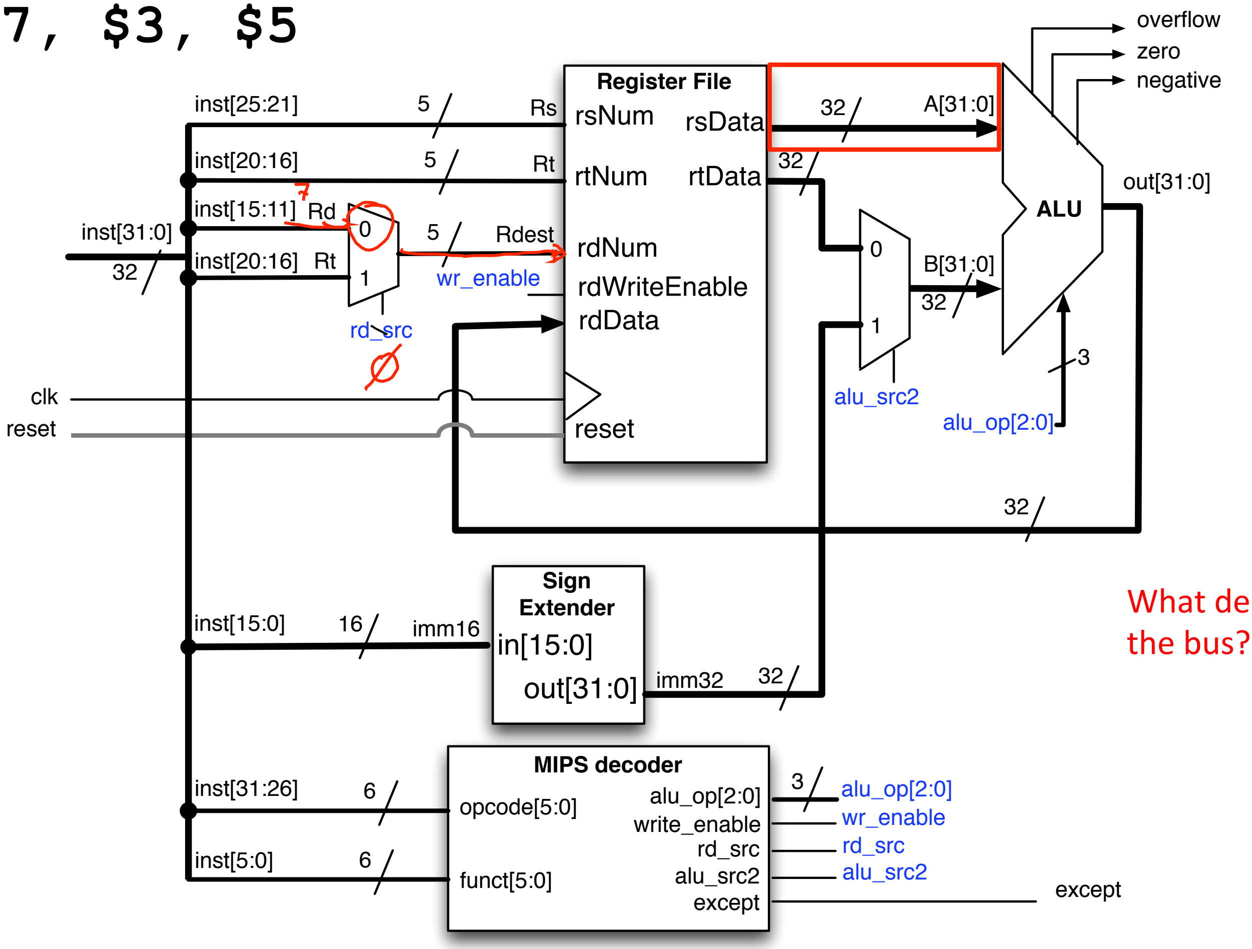
0	00000000
1	????
2	????
3	????
4	????
5	????
6	????
7	????

rd rs rt

add $7, $3, $5

r-type

		31
op	000000	
		26
		25
rs	00011	
		21
		20
rt	00101	
		16
		15
rd	00111	
		10
		9
shamt		
		6
		5
func	100000	
		0



Reg File

0	00000000
1	
2	
3	0000000A
4	
5	FFFFFFF9
6	
7	

What decimal value is on the bus?

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