All Together:
Instruction Memory + Arithmetic Unit

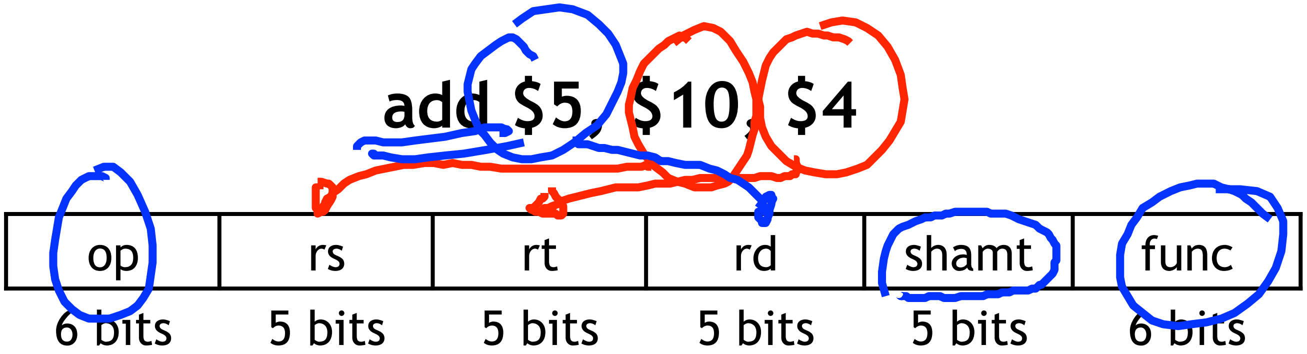
Pick up a handout

Today's lecture

- **Instructions**
  - Instruction Memory
  - Program Counter (PC)
  - Adder
- **Putting all together**
  - Arithmetic unit to work

Friday's lecture

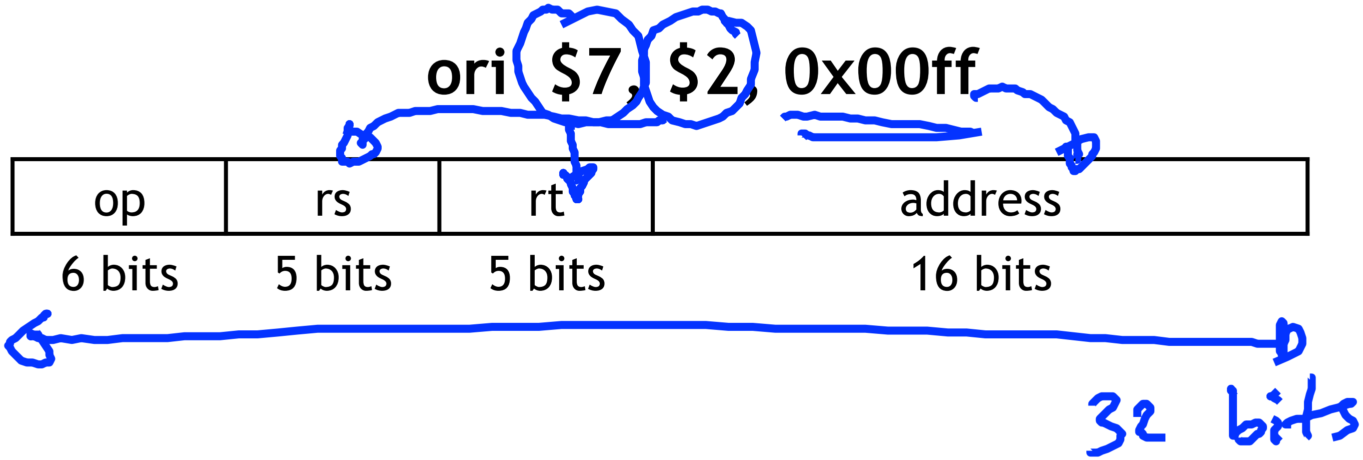
- Register-to-register arithmetic instructions use the R-type format.



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.

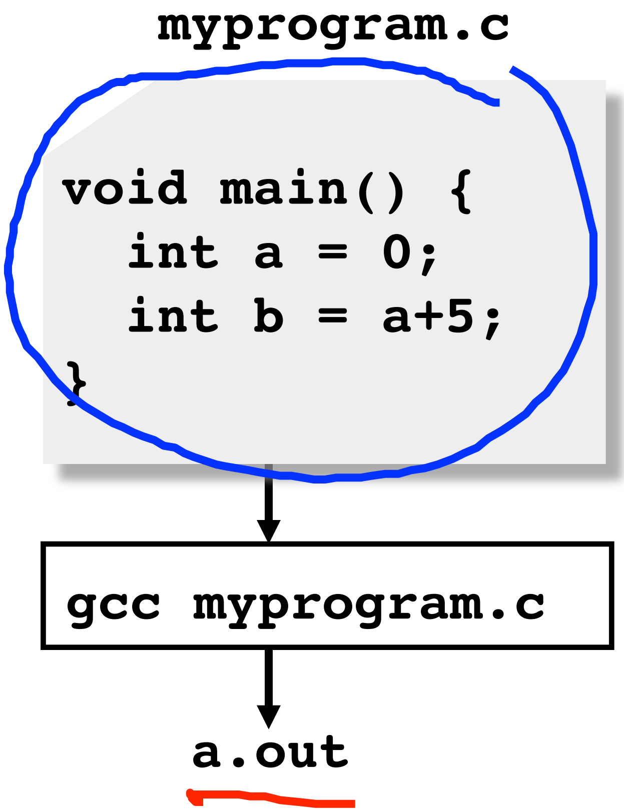


ori $7, $2, 0x00ff

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

32 bits

Where are the instructions my program executes?



myprogram.c

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

```
gcc myprogram.c
```

a.out

- To look at the assembly code of a.out:

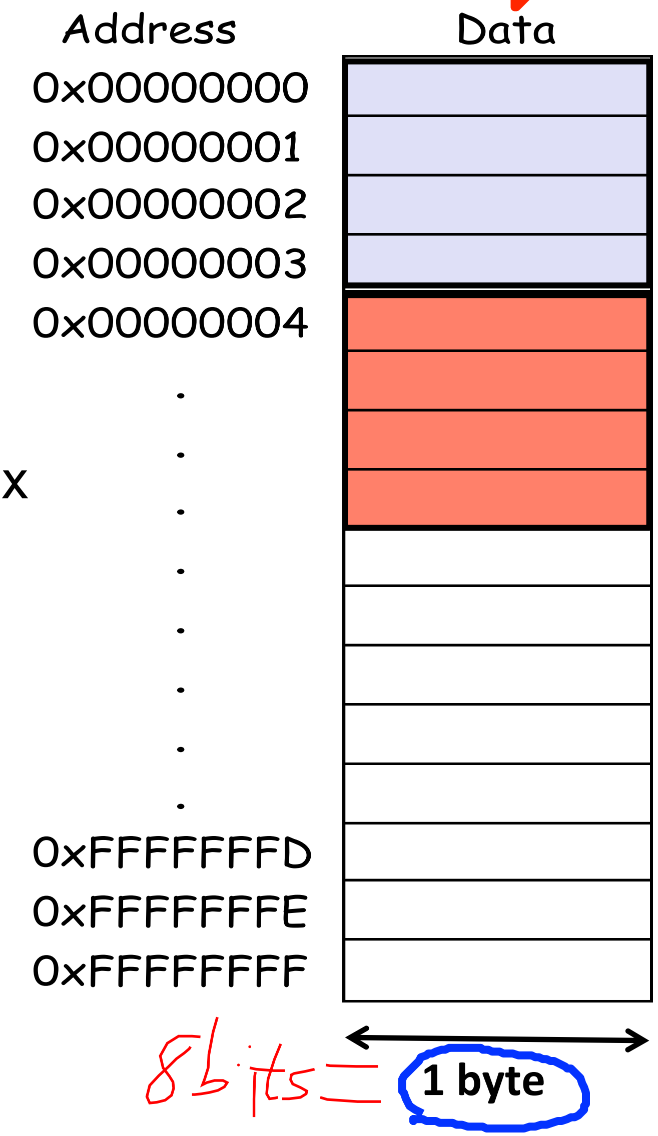
```
$ objdump –d a.out
```

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

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

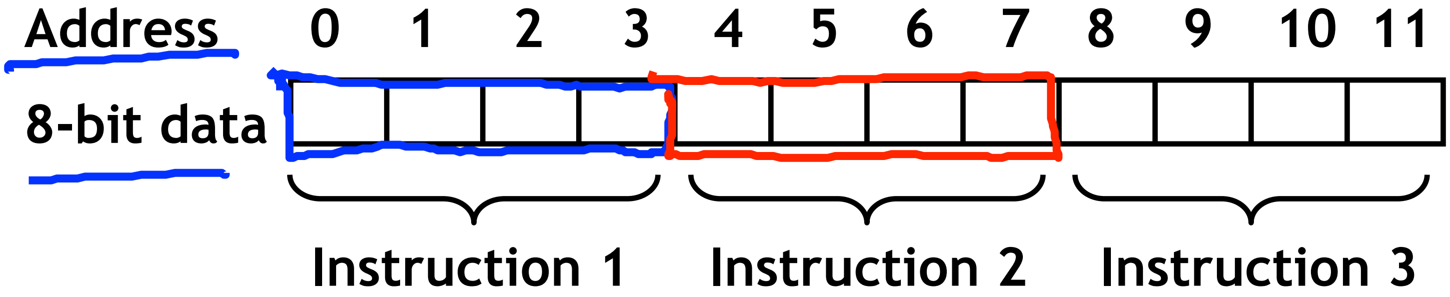
Where are the instructions my program executes?

- The program is stored in Memory
- Assume our program is stored in a Read Only Memory (ROM)
  - We can read its contents, but cannot modify them
  - A 32-bit address serves as an array index
    - # addresses: 2^{32} = 4 G
    - Each address contains 1 byte: 4Gbytes
  - MIPS memory is byte-addressable, so a 32-bit instruction actually occupies four contiguous locations (bytes) of memory



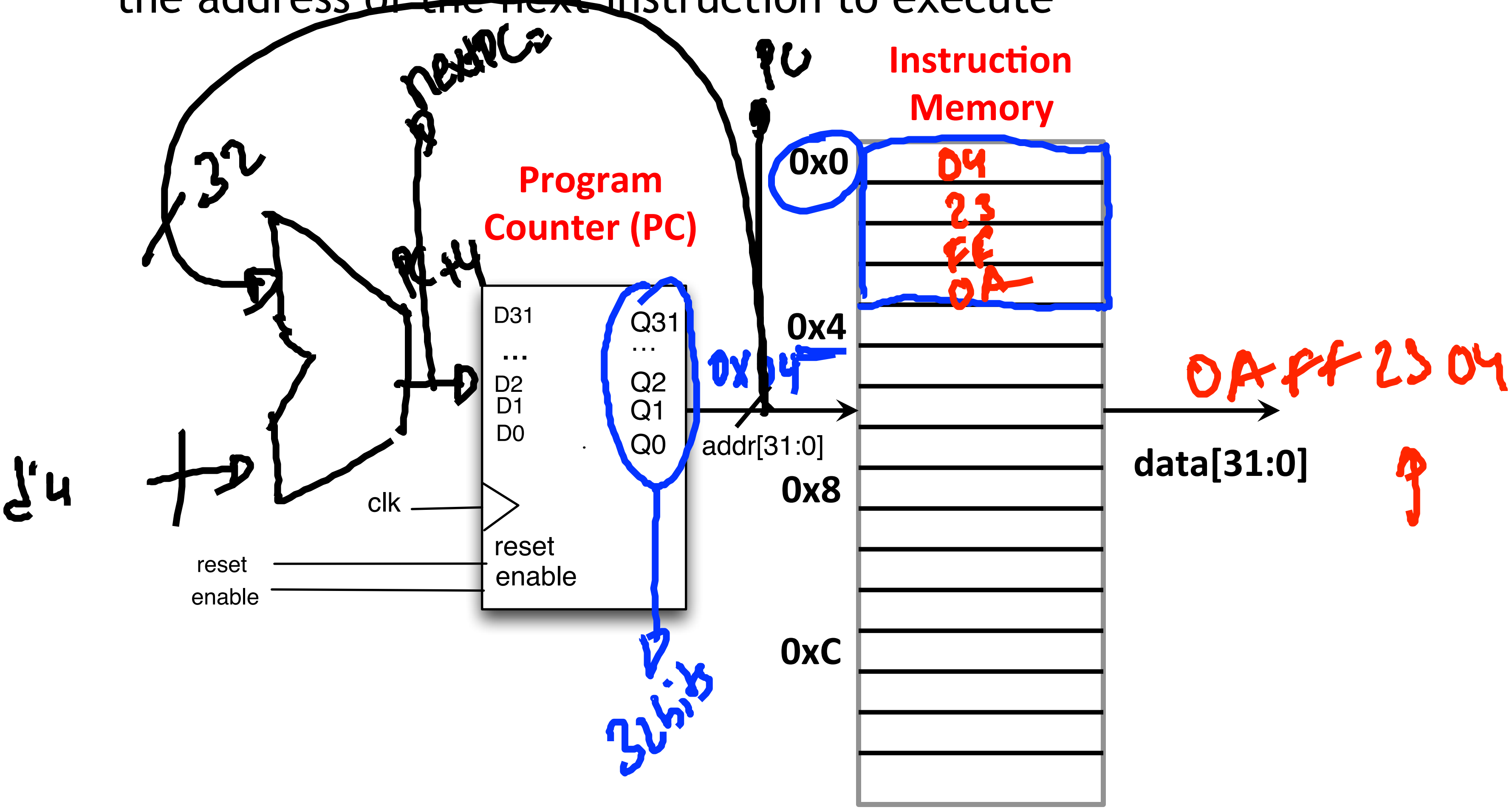
Memory alignment

- 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.

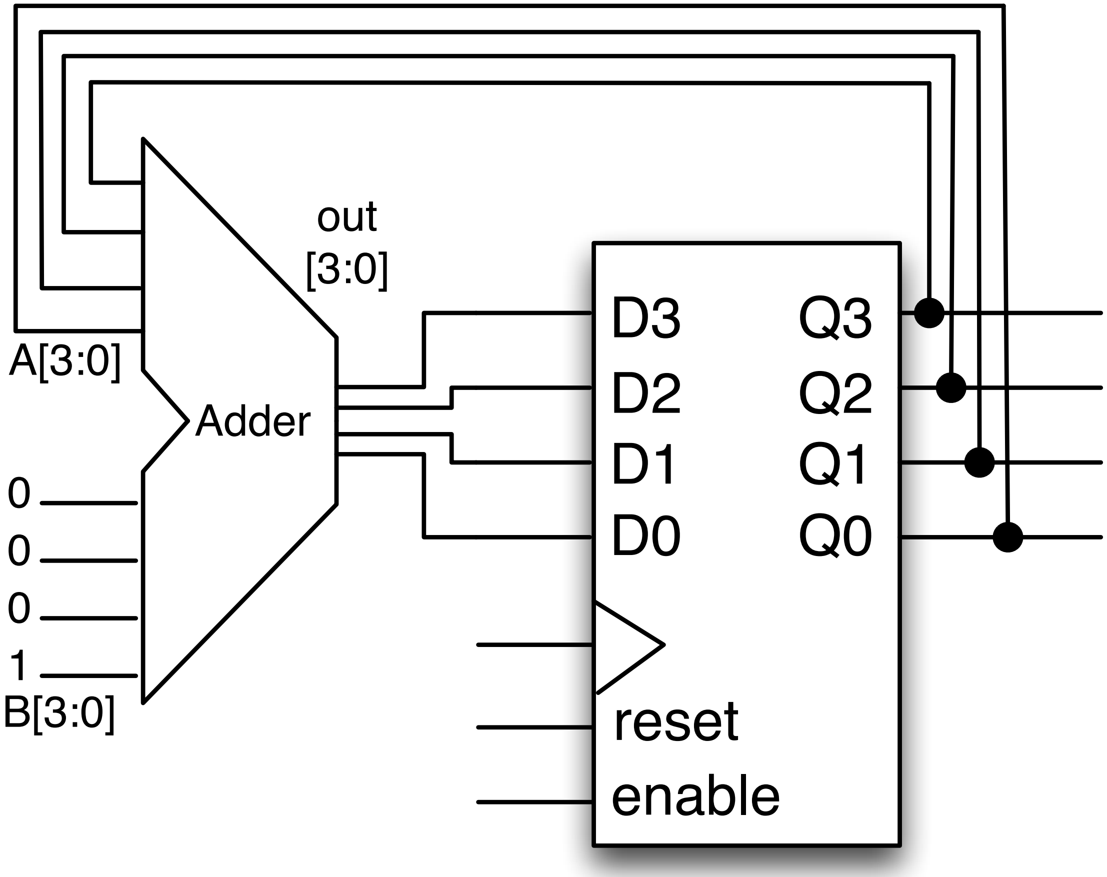


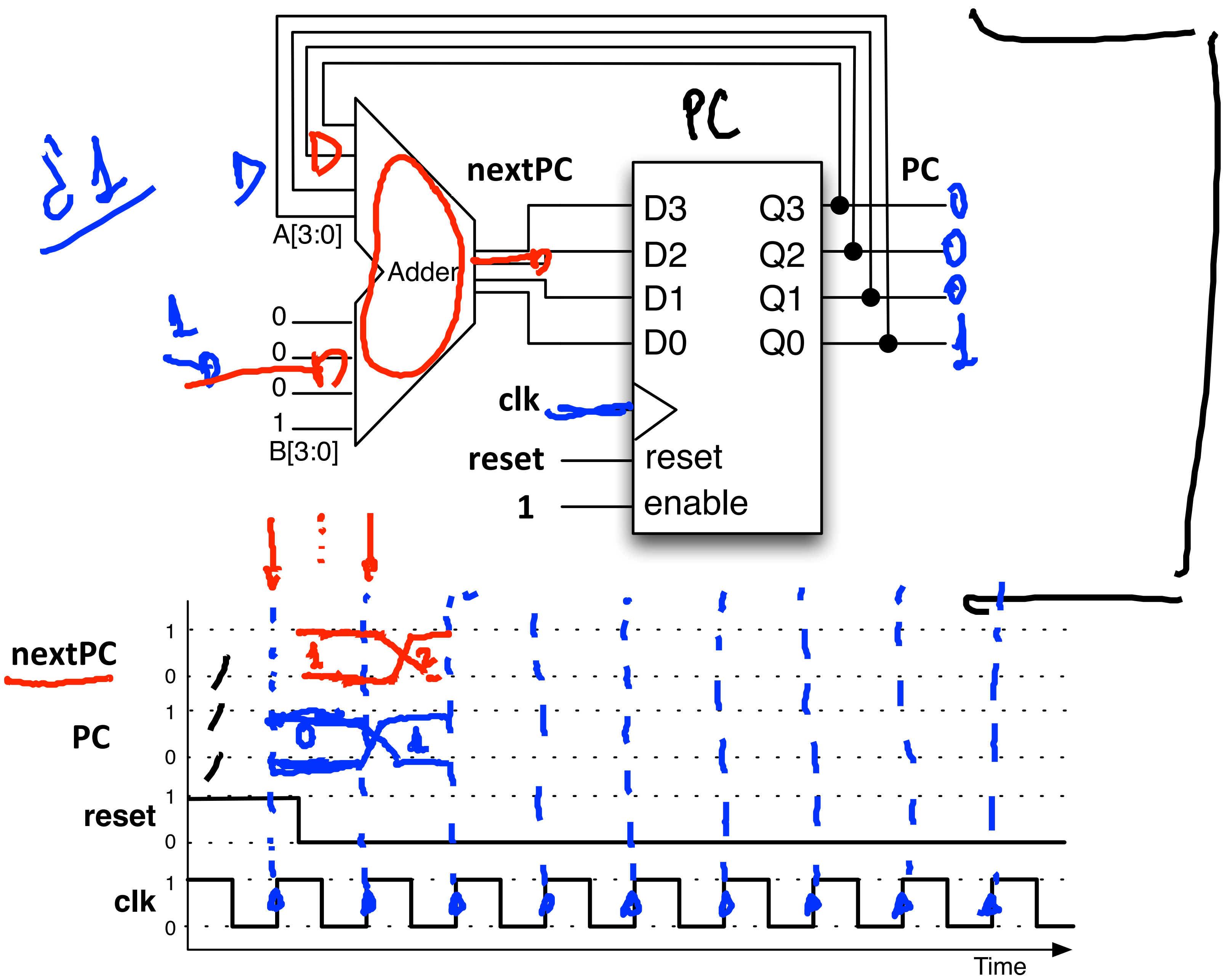
How do we know which instruction to execute?

- We have a register called Program Counter (PC) that contains the address of the next instruction to execute

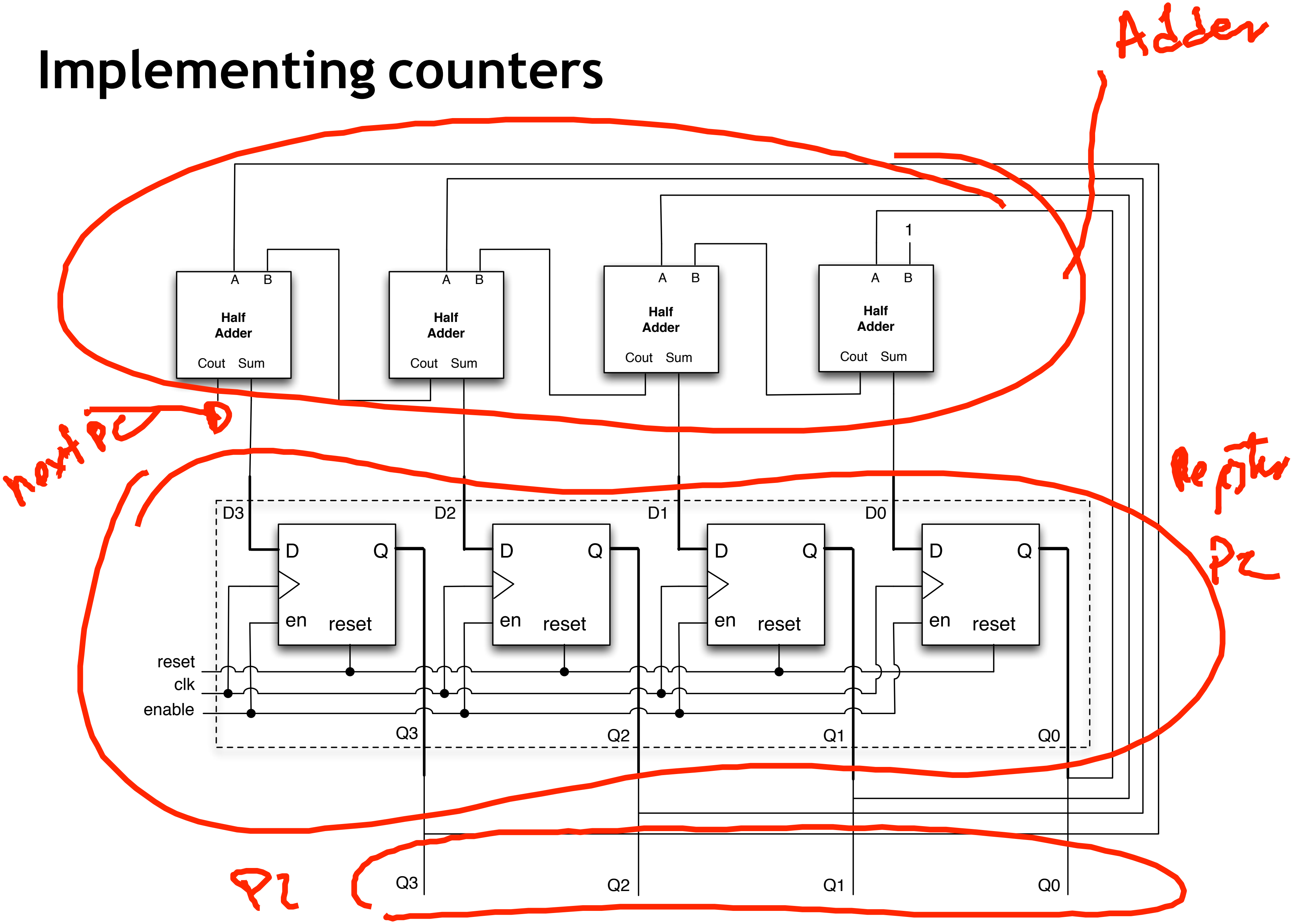


What do you get if you connect a register to an adder?

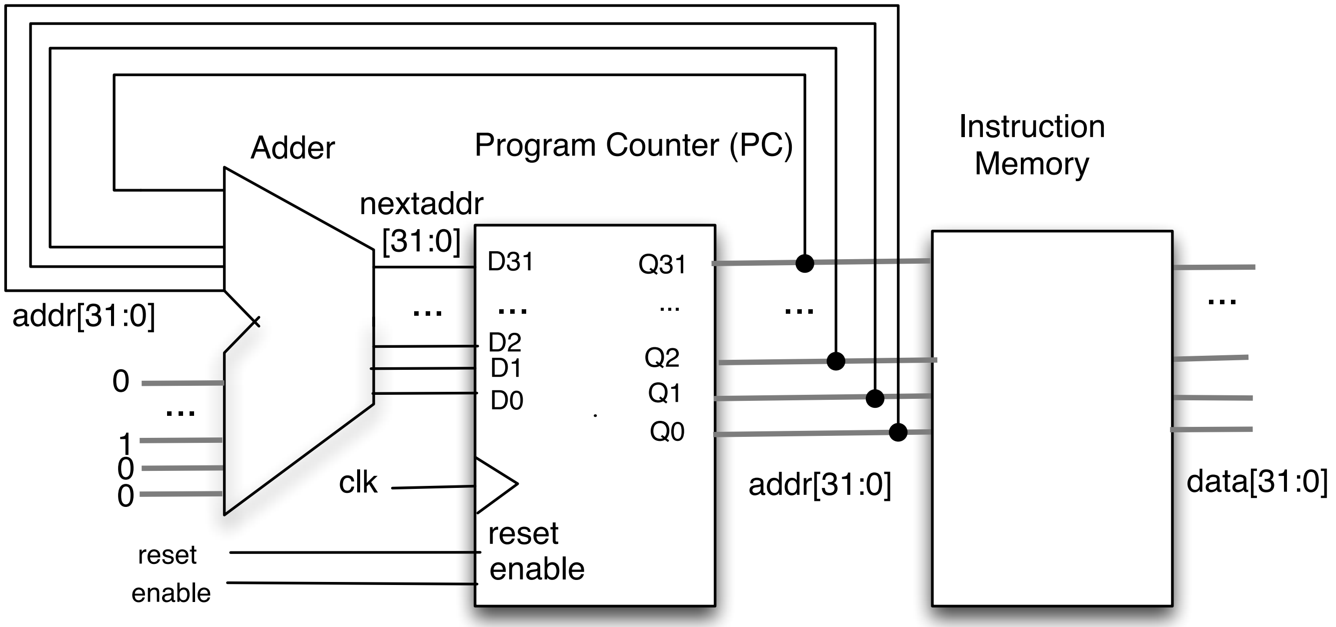



PC
A[3:0]
Adder
B[3:0]
0
0
0
1
nextPC
D3
D2
D1
D0
Q3
Q2
Q1
Q0
PC
0
0
0
1
clk
reset
reset
1
enable
nextPC
PC
reset
clk
1
0
1
2
0
1
Time

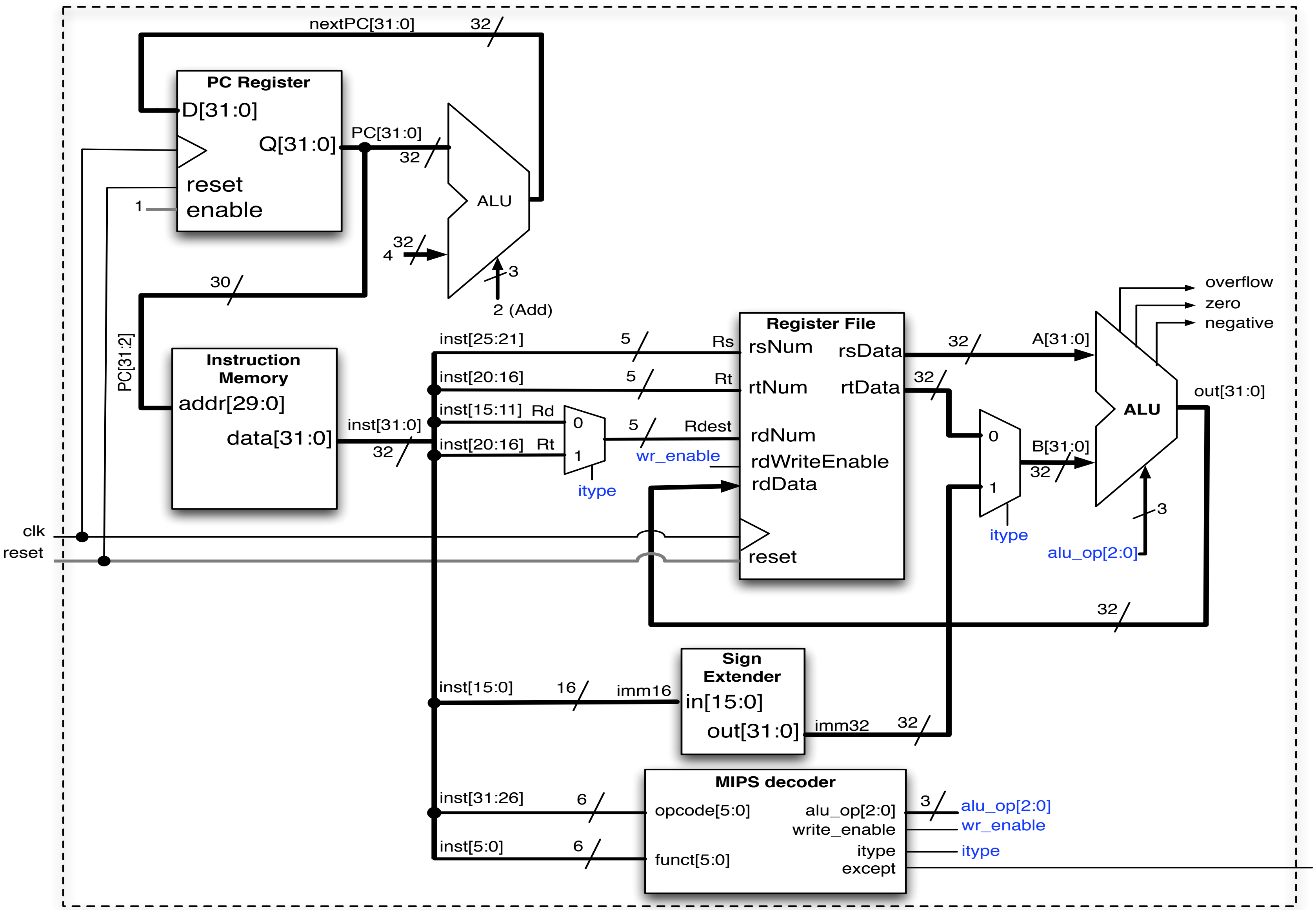
Implementing counters



Instruction Memory + PC + Adder



Putting all together



Example

My program

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

Assembly

Example

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
```

-7 2's C

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

Machine code

2 0 0 3 0 0 A

Ø

`addi  $3, $0, 0x000A`

op	rs	rt	imm
001000	00000	00011	000A

4

`addi  $5, $0, 0xFFF9`

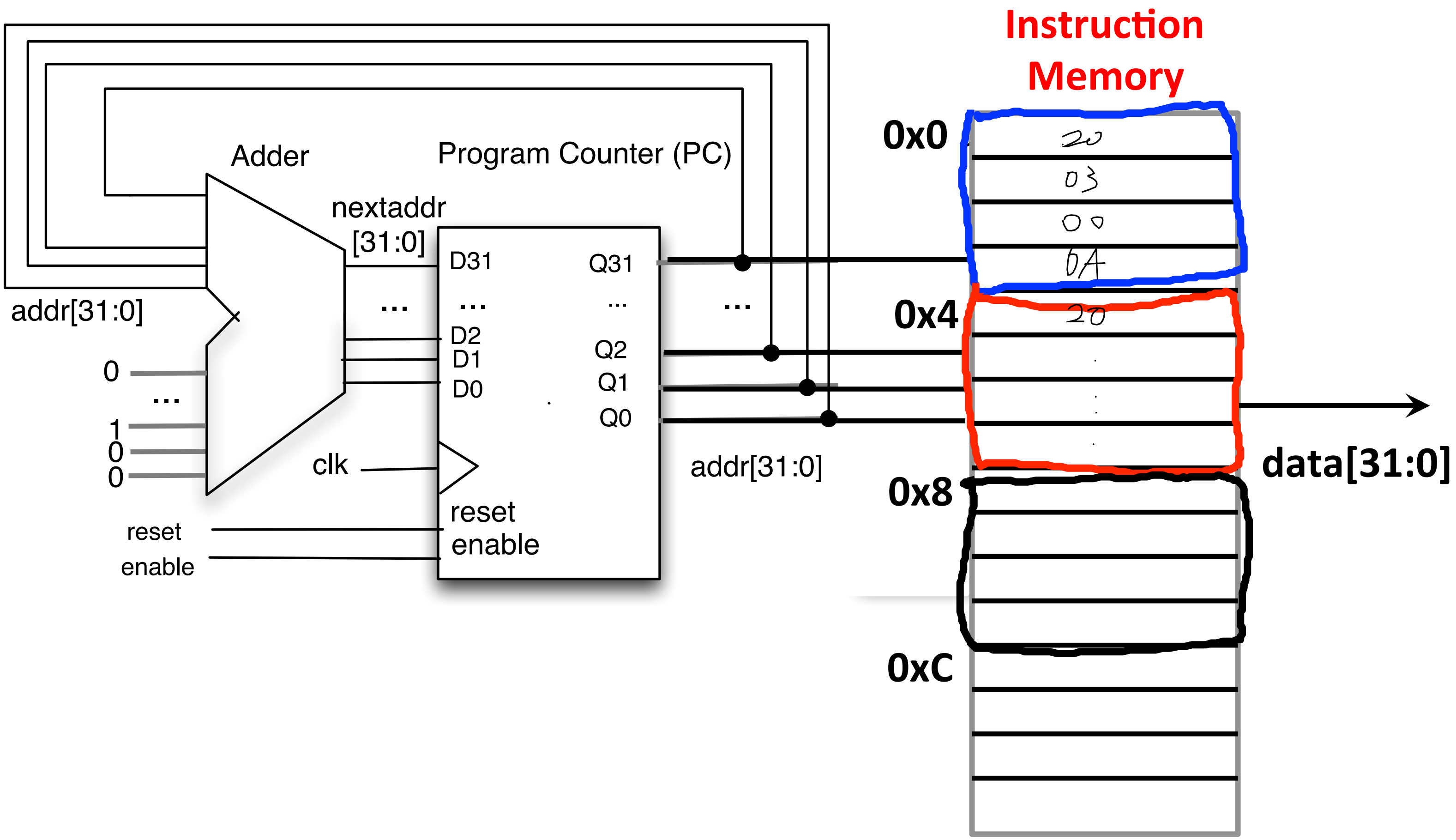
op	rs	rt	imm
001000	00000	00101	FFF 9

8

`add  $7, $3, $5`

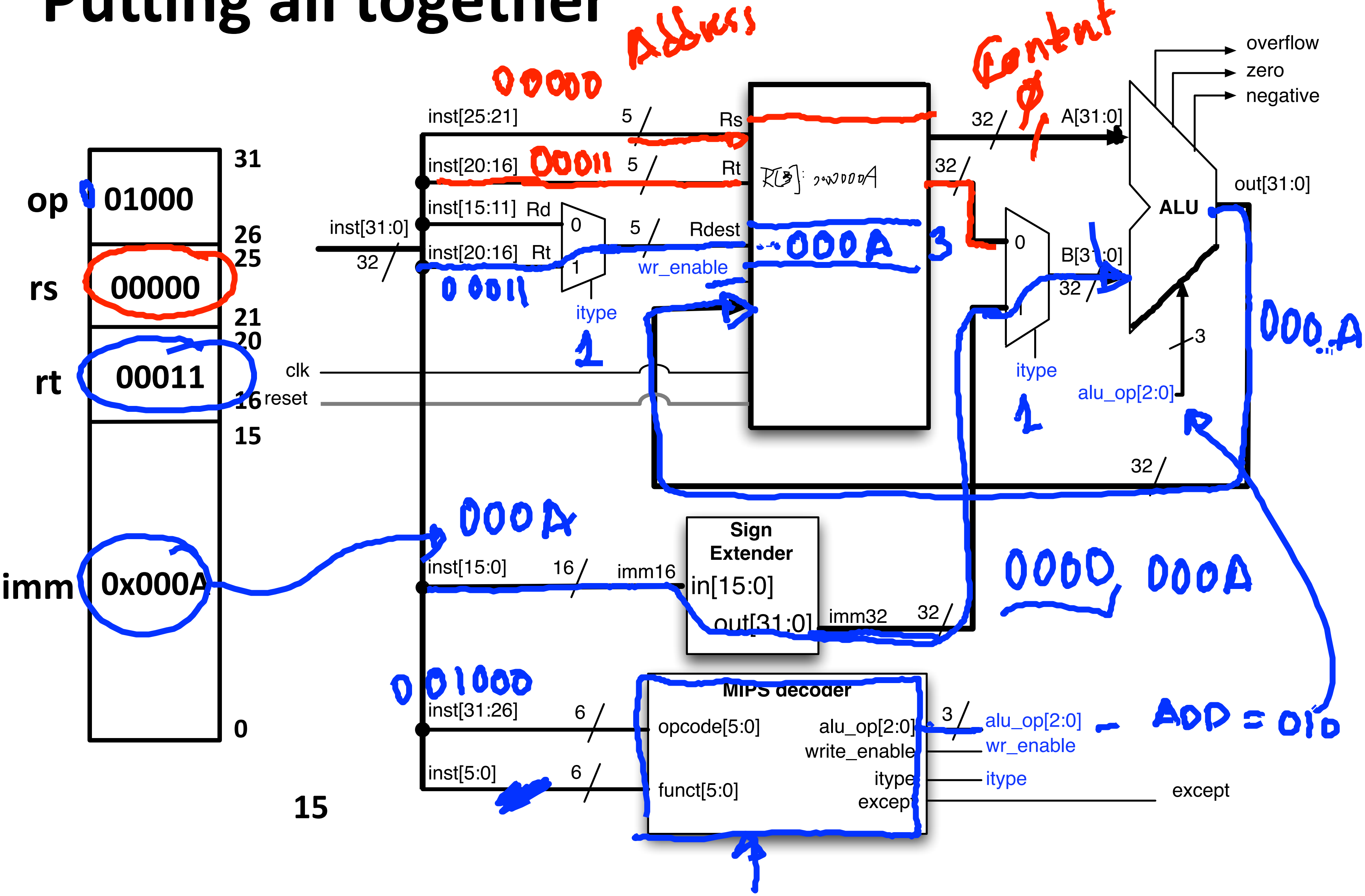
op	rs	rt	rd	shamt	func
000000	00011	00101	00111		100000

	opcode	funct
add	0x00	0x20
addi	0x08	


Instruction Memory
Adder
Program Counter (PC)
nextaddr
[31:0]
addr[31:0]
D31
...
D2
D1
D0
Q31
...
Q2
Q1
Q0
0
...
1
0
0
clk
reset
enable
reset
enable
...
...
addr[31:0]
0x0
0x4
0x8
0xC
20
03
00
0A
20
data[31:0]

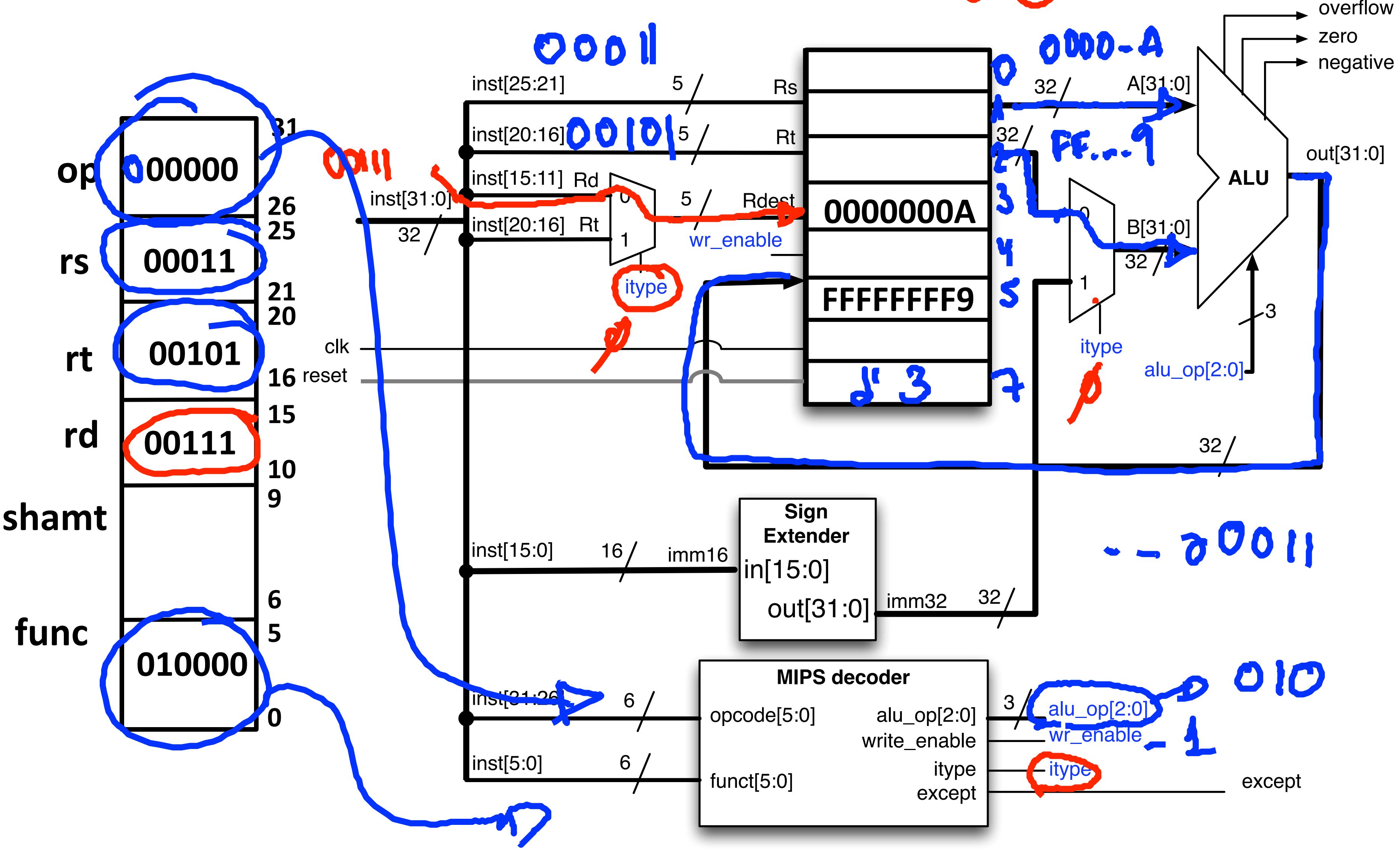
Putting all together

`addi  $3, $0, 0x000A`

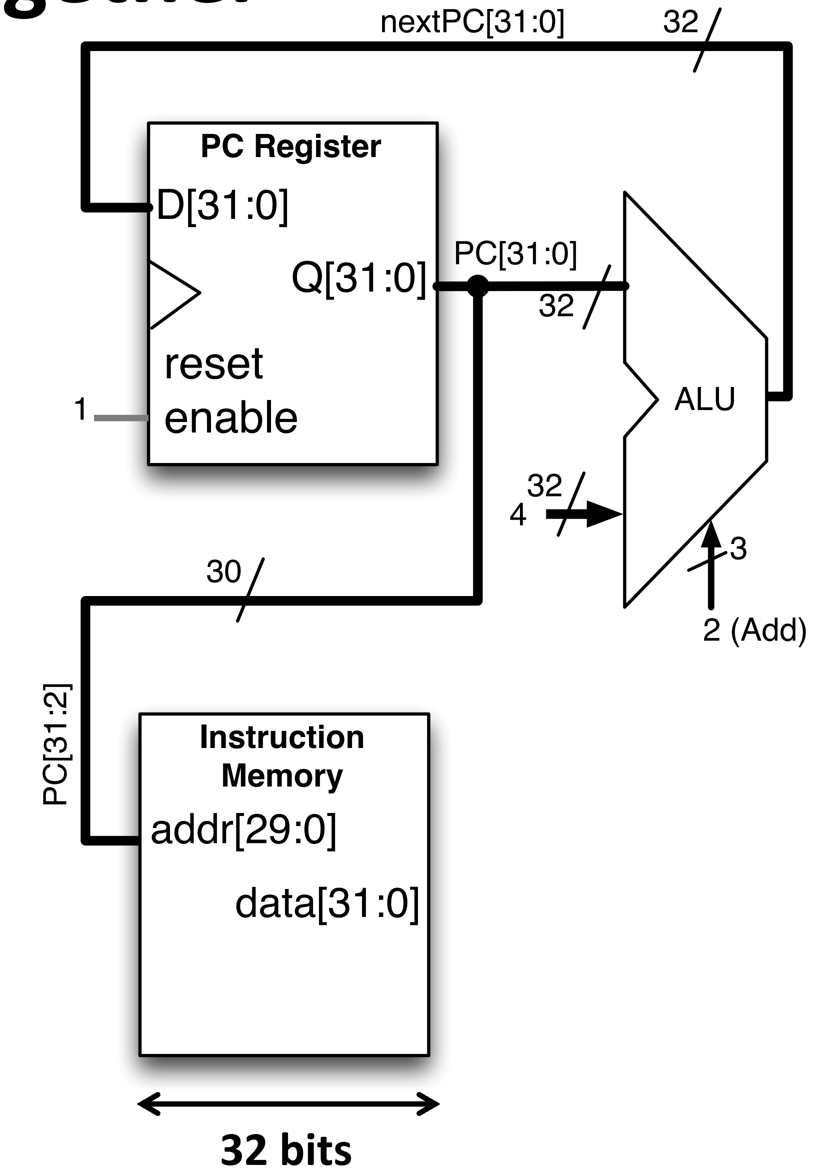


Putting all together

`add $7, $3, $5`



Putting all together



nextPC[31:0]
32
PC Register
D[31:0]
Q[31:0]
reset
1
enable
PC[31:0]
32
ALU
32
4
3
2 (Add)
30
PC[31:2]
Instruction Memory
addr[29:0]
data[31:0]
32 bits