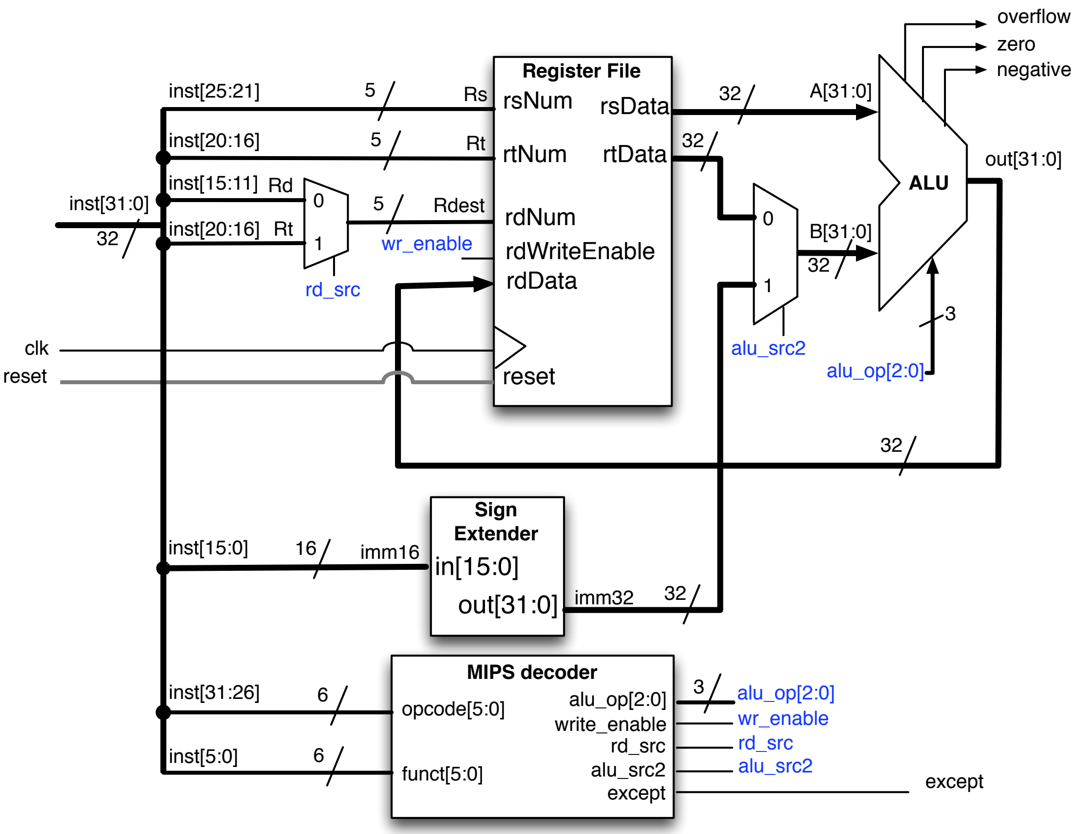
# Instruction Decoding

PICK UP 2 HANDOUTS

BRING "GREEN SHEET" TO LECTURE
& DISCUSSION SECTION.

# By the End of Today’s Lecture



# Today's lecture

- **Instruction Encoding** ←
  - R-type & I-type encodings
- **Instruction Decoding** ← HW
  - Operands
  - Sign-extending the immediate
  - Decoding the ALU operation

# How can we write MIPS code to compute the following expression?

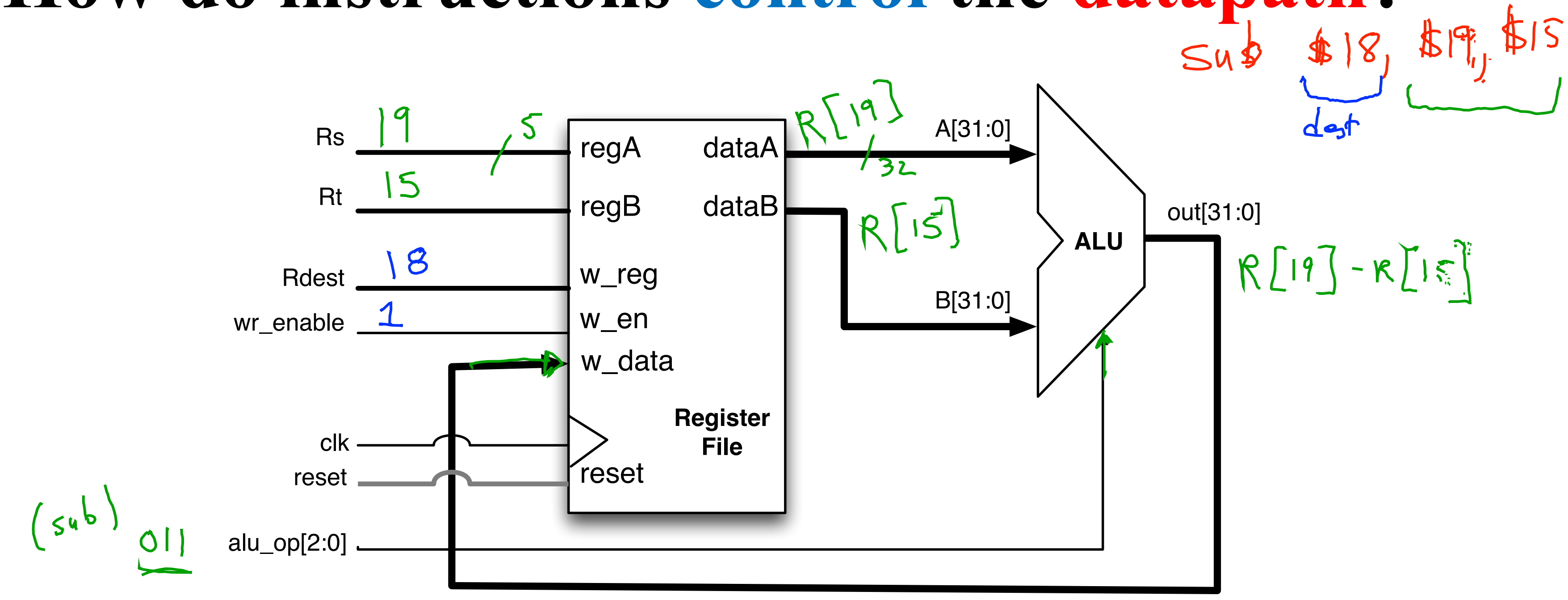
z = 4 + x*y - z;

mul dest, src1, src2

- Assume the following register allocation:
    - $13 = x, $20 = y, $15 = z

```
mul  $19, $13, $20
sub  $19, $19, $15   ←
add  $15, $19, 4
```

# How do instructions control the datapath?



sub $18, $19, $15

- First step is to learn how instructions are encoded

# Machine language is a binary format that can be stored in memory

- MIPS machine language is easy to decode
  - Each MIPS instruction is 32 bits wide
  - There are only three instruction formats
    - We'll see two of them today

R - arith

I - imm

# Register-to-register arithmetic instructions use the **R-type** format

and $17, $14, $7

op rd, rs, rt

dest src

32 reg

$\log_2(32) = 5b$

| | op | rs | rt | rd | shamt | func |
|---|---|---|---|---|---|---|
| 32b = | 6 bits | + 5 bits | + 5 bits | + 5 bits | + 5 bits | + 6 bits |

operation code or opcode

source registers

destination register

shift amount (rarely used)

Which arithmetic instruction?

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

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

rs rt

- Example: `add $5, $10, $4`

dest

0x0 0x20

| | | | | | |
|---|---|---|---|---|---|
| 00 0000 | 01010 | 00100 | 00101 | 00000 | 10 0000 |

~~X <<2~~ ~~slli~~ ~~00010~~

# Register-to-register arithmetic instructions use the **R-type** format

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

- Example: `or $22, $13, $8`

| | | | | | |
|---|---|---|---|---|---|
| | | | ????? | | |

a) xxxxx
b) 01000
c) 01101
d) 10110

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

|  | op | rs | rt | imm |
|---|---|---|---|---|
| 32b = | 6 bits + | 5 bits + | 5 bits + | 16 bits |

operation code
or opcode

source registers

ALWAYS

destination register

16-bit 2’s complement
immediate
(-32,768 to 32,767)

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

| op | rs | rt | imm |
|---|---|---|---|
| 6 bits | 5 bits | 5 bits | 16 bits |

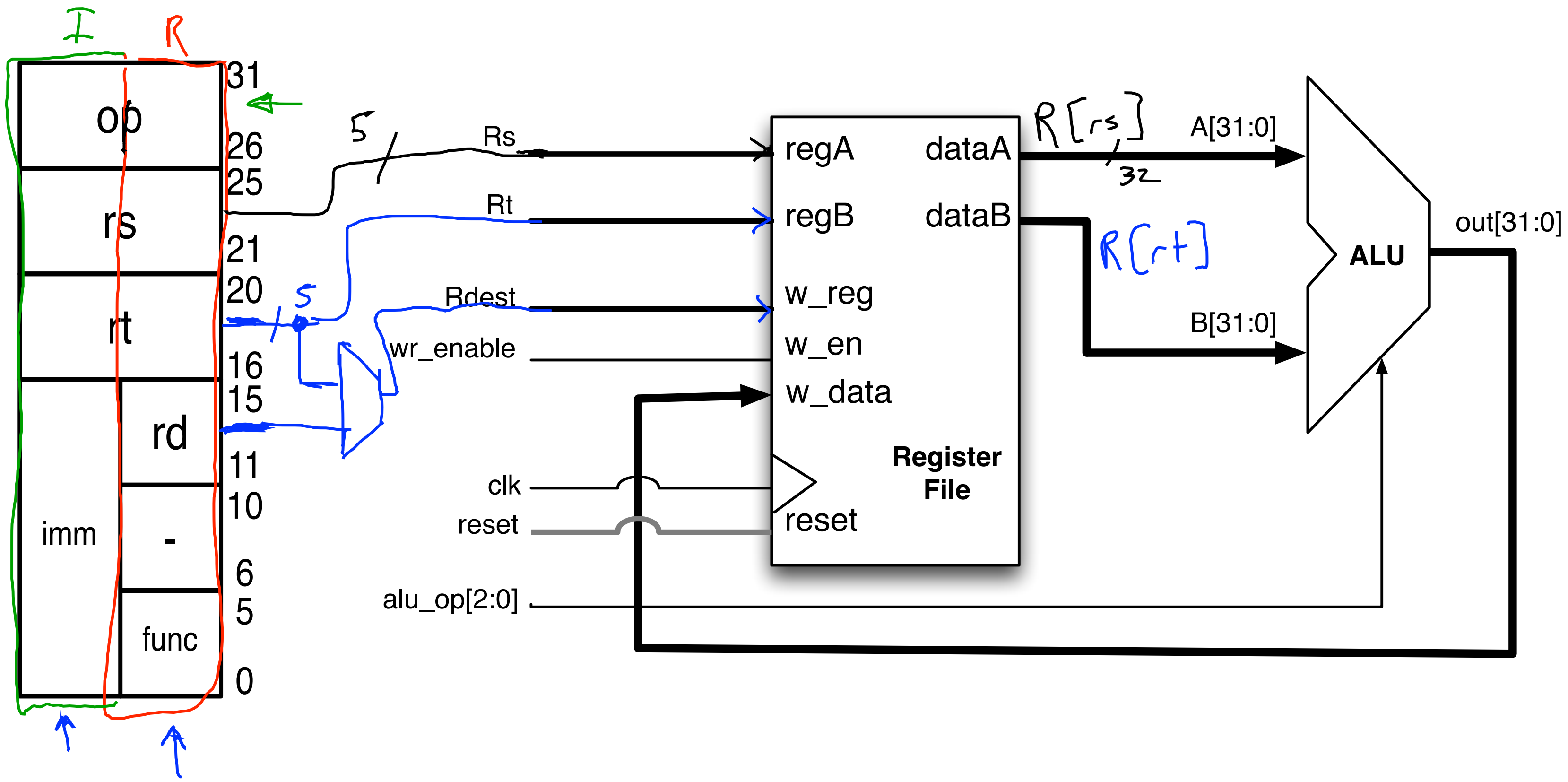
rt rs 4 −1

- Example 0xd ori $7, $2, 0xff 0x ffff

| | | | |
|---|---|---|---|
| 00 1101 | ??????<br>00010 | 00111 | 0000 0000 1111 1111 |

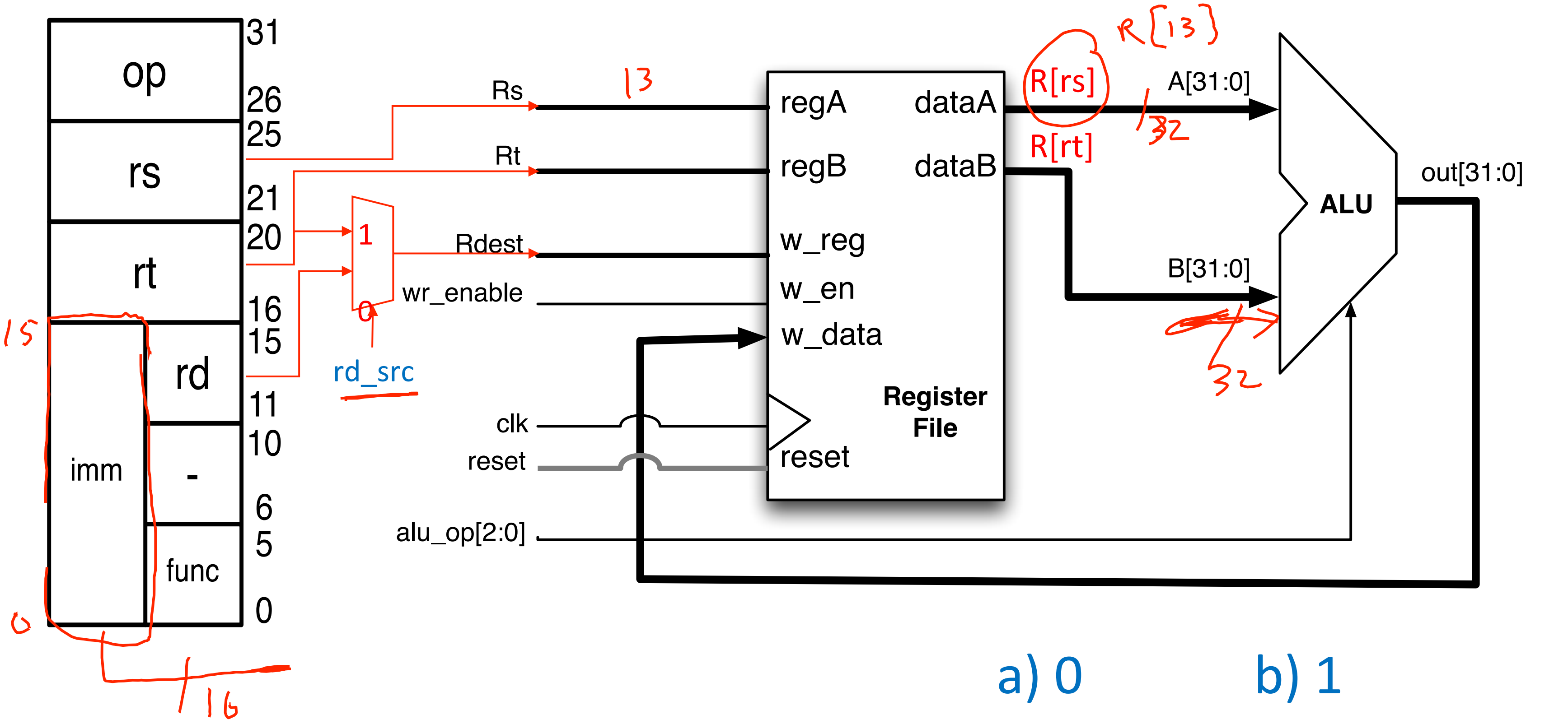
a) 01101
b) 00111
c) 00010
d) 11111

# Some control signals are encoded in the instruction



# If we have an I-type instruction, what should the control signal rd_src be?

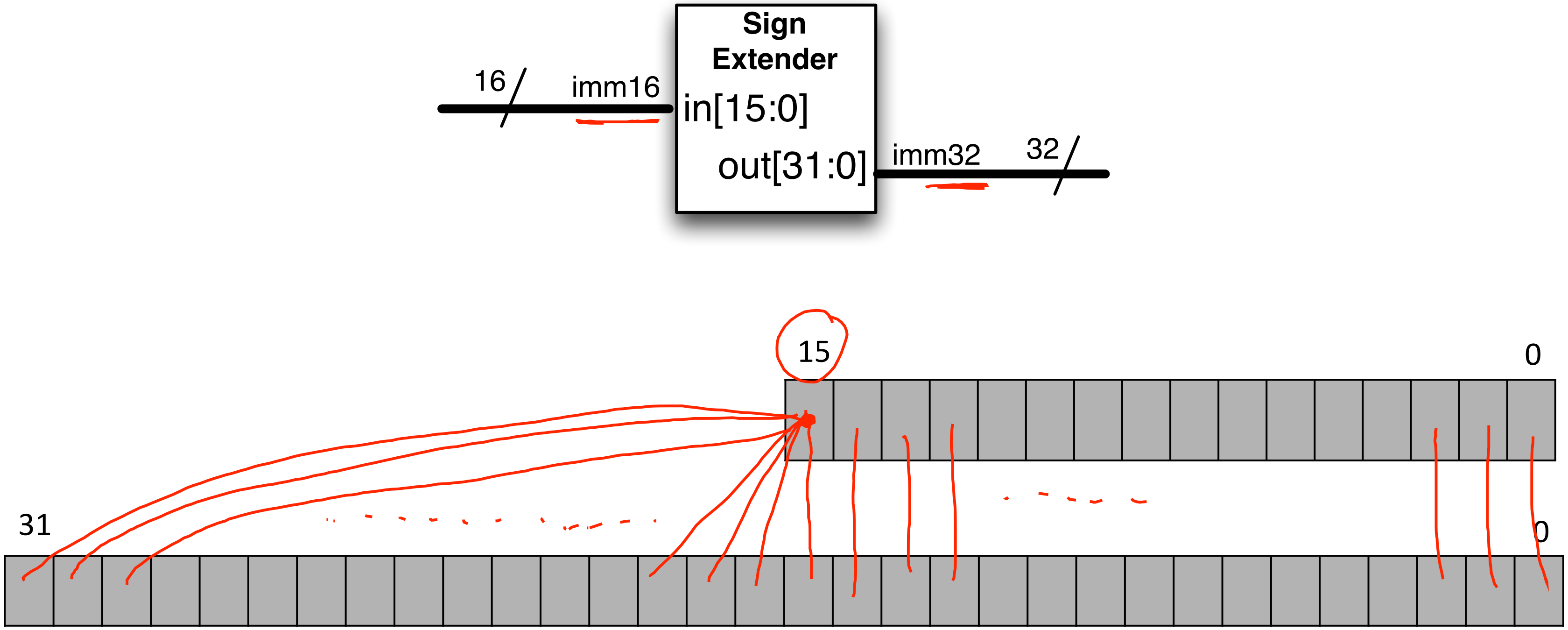
addi $7, $13, 1000



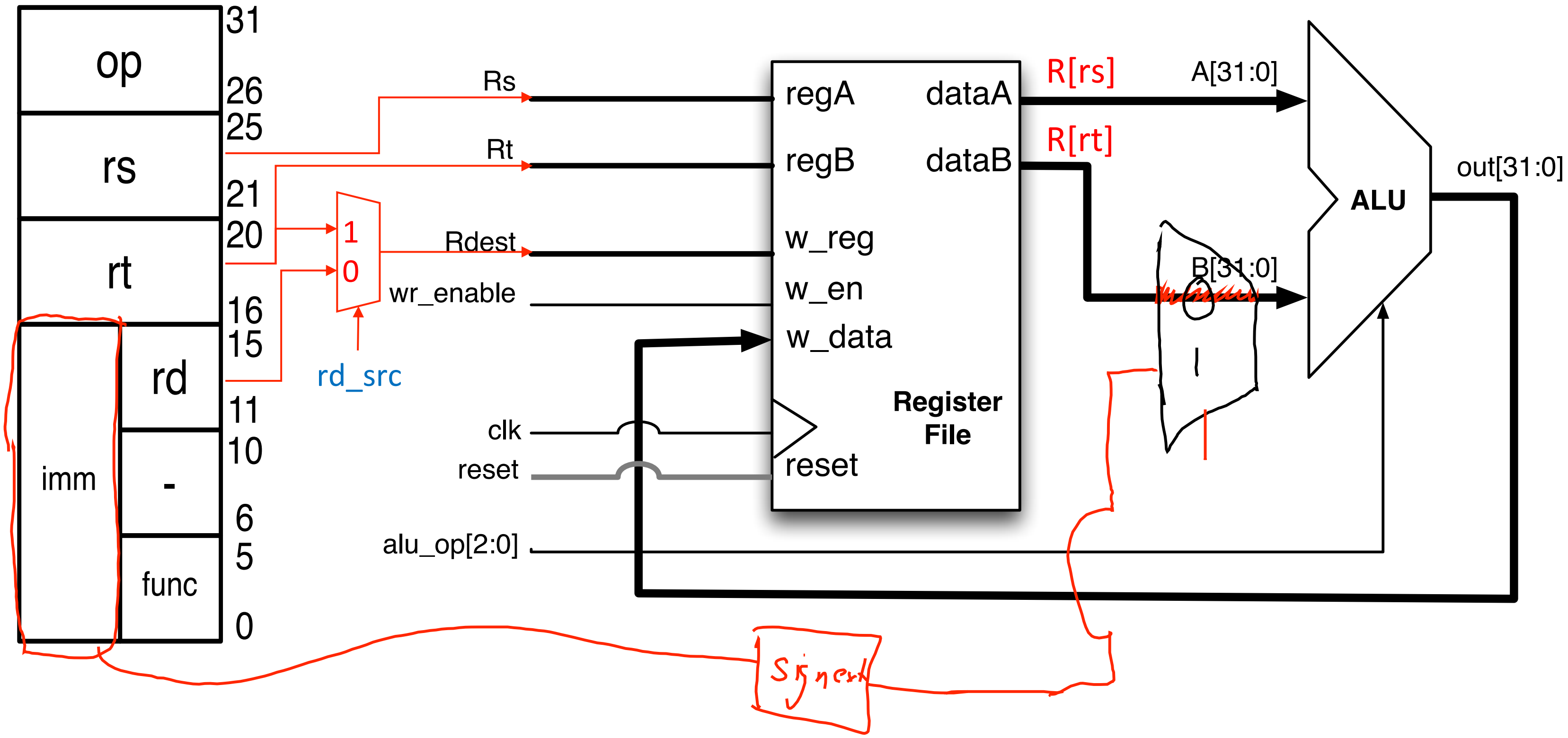
a) 0

b) 1

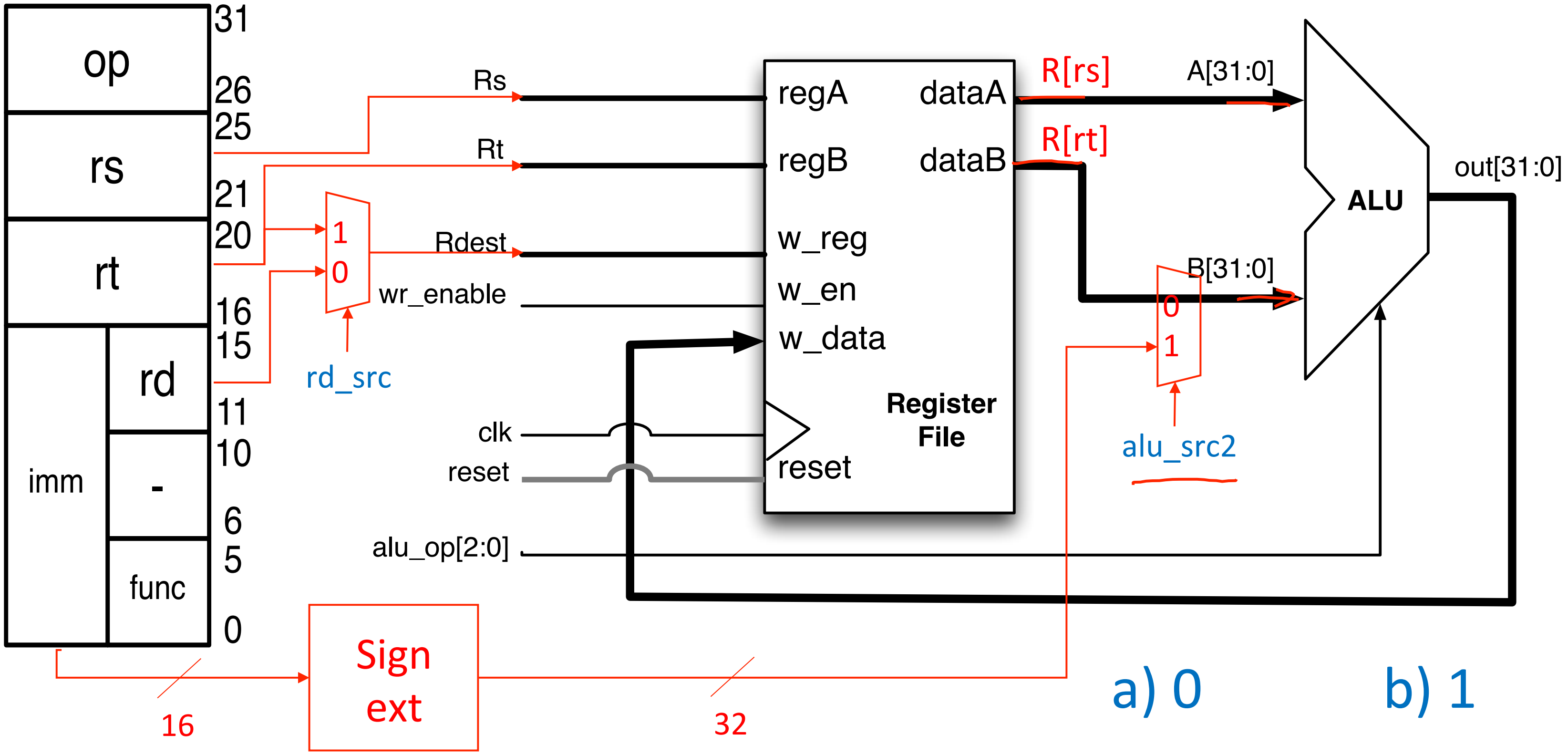
# Sign Extension replicates the MSb of imm16



# Select behavior of the ALU B input based on instruction type



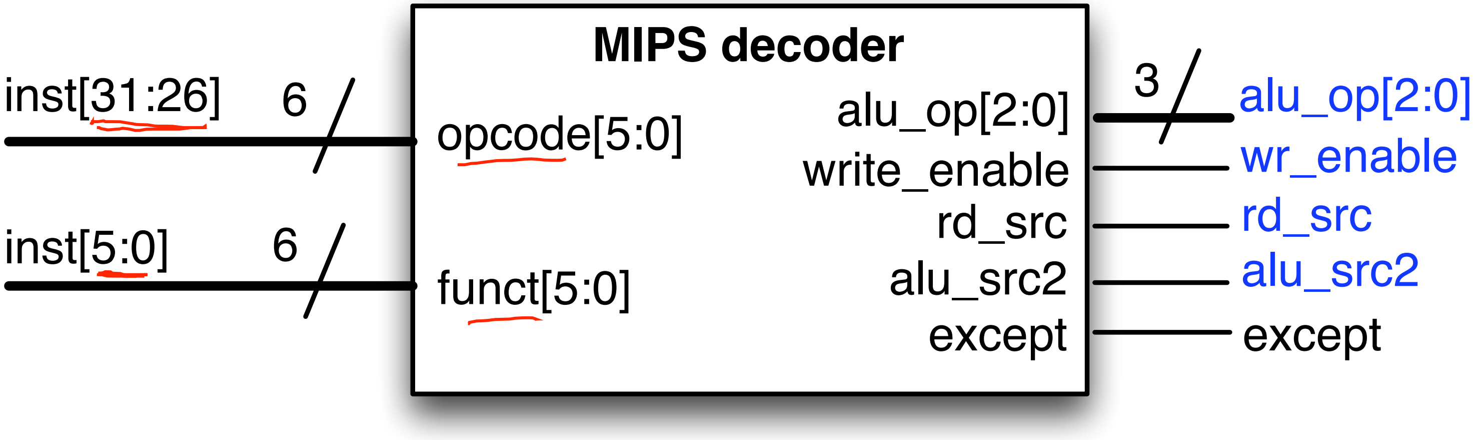
# If we have an R-type instruction, what should the control signal alu_src2 be?



a) 0

b) 1

# The instruction decoder translates bits from the instruction into control signals

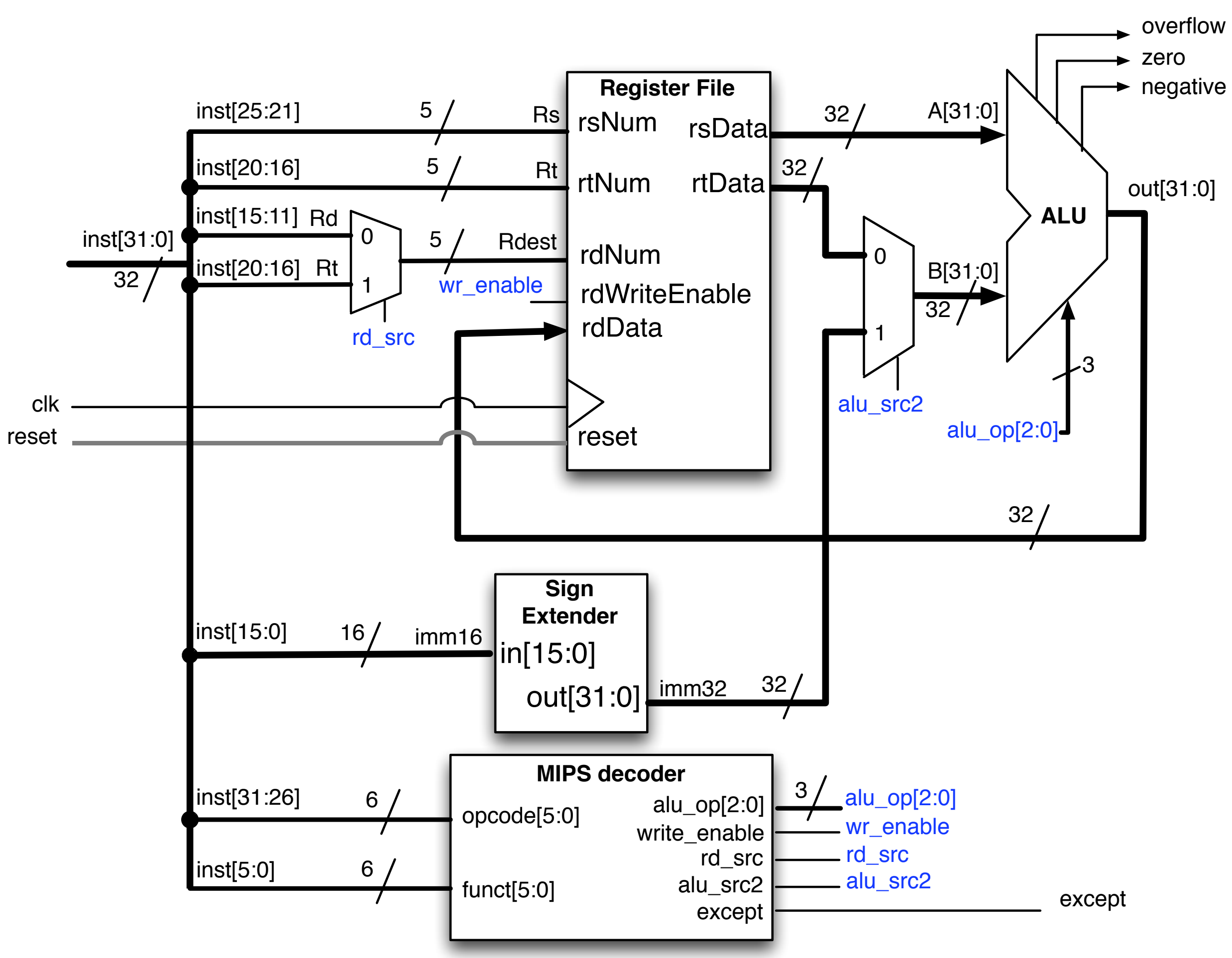


# Use a table to decode instructions into control signals

in

| Instruction | opcode | func | alu_op | rd_src | alu_src2 | wr_enable |
|---|---|---|---|---|---|---|
| add | 000000 | 100000 | 010 (Add) | 0 | 0 | 1 |
| sub | 000000 | 100010 | 011 (sub) | 0 | 0 | 1 |
| and | 000000 | . | ⋮ | 0 | 0 | 1 |
| or | 000000 | : | ⋮ | 0 | 0 | 1 |
| nor | 000000 | . | ⋮ | 0 | 0 | 1 |
| xor | 000000 | | | 0 | 0 | 1 |
| addi | 00 1000 | xxxxxx | 010 (add) | 1 | 1 | 1 |
| andi | 00 1100 | xxxxxy | (and) | 1 | 1 | 1 |
| ori | | | | 1 | 1 | 1 |
| xori | | | | 1 | 1 | 1 |

# Arithmetic Machine Datapath



overflow
zero
negative
inst[25:21]
5
Rs
inst[20:16]
5
Rt
inst[15:11]
Rd
inst[31:0]
32
inst[20:16]
Rt
0
1
5
Rdest
wr_enable
rd_src
Register File
rsNum
rsData
rtNum
rtData
rdNum
rdWriteEnable
rdData
reset
32
A[31:0]
32
0
1
B[31:0]
32
alu_src2
ALU
out[31:0]
3
alu_op[2:0]
clk
reset
32
Sign Extender
inst[15:0]
16
imm16
in[15:0]
out[31:0]
imm32
32
MIPS decoder
inst[31:26]
6
opcode[5:0]
inst[5:0]
6
funct[5:0]
alu_op[2:0]
write_enable
rd_src
alu_src2
except
3
alu_op[2:0]
wr_enable
rd_src
alu_src2
except