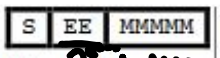
Consider two competing 8-bit floating point formats. Each contains the same fields (sign, exponent, significand) and follows the same general rules as the 32-bit IEEE standard (denorms, biased exponent, non-numeric values, etc.), but allocates its bits differently. To save you time, *you only need to complete and circle the (LEFT or RIGHT) blank whose value is closest to zero*, that's the only one we'll grade! (If they're the same value, write the answer in both, & circle both). E.g,. The number represented by `0x00` was 0 for both, so we circled both. But for "exponent bias", just from the # of `EE…E` bits in each, we know |LEFT's bias| < |RIGHT's bias|, so there's no need to calculate or write the answer on the RIGHT.

sign · $2^{exp-bias+1}$ · 1.mant

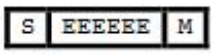
sign · 2^0 · 1.mant

unsig -b

"LEFT" format:



"RIGHT" format:



scratch space (show all work here)

1.MM…

scratch space (show all work here)

$-1 \le exp \le 1$ $b = 2^{n-1}-1$

$1 \cdot 2^5 = 32$

val of exp · mant choices

$1 \cdot 2^5 = 32$

exp mant

$2^{exp} \cdot 2^{-5} = stepsize$

$2^0 \cdot 2^{-5} = 2^{-5}$

$2^1 \cdot 2^{-5} = 2^{-4}$

$-31 \le exp \le 1$

$31 \cdot 2^1 = 62$

vals of exp · mant choices

1 choice for exp · $2^1 = 2$ mant

$2^{exp} \cdot 2^{-1} = stepsize$

$2^{-30} \cdot 2^{-1} = 2^{-31}$

$2^{31} \cdot 2^{-1} = 2^{30}$

	LEFT	RIGHT
Number represented by `0x00`:	0	0
Exponent Bias:	1	31 (not graded, so no need to write)
a) # Numbers ($0 \le n < 1$):	32	62
b) # Numbers ($1 \le n < 2$):	32	2
c) Difference between two smallest positive values:	2^{-5}	2^{-31}
d) Difference between two biggest non-∞ values:	2^{-4}	2^{30}
e) Positive Integer closest to 0 it cannot represent:	4	5

f) Which implementation is better for *approximating* π: LEFT or RIGHT ? (circle one) $= 3.14$

stepsize $= 2^1 = 2^{exp} \cdot 2^{-5}$ step $= 2^1 = 2^{exp} \cdot 2^{-1}$

$exp = 2$

$\Rightarrow$ exp: 1, mant: 0b1.111 ← exp 2 mant: 0

← exp: 2 mant: 1

X exp: 2, mant 0b000000 $= 4$ $2^2 \cdot 0b1.1$

0b1.10

$= 6$

1) For a 12-bit integer represented with two's complement, what is the:

a) Most positive value (in decimal): $2^{11}-1 = 2047$

b) Binary representation of that number: 0b0111 1111 1111

c) Most negative value (in decimal): $-2^{11} = -2048$

d) Hex representation of that number: 0x800

e) In general, for an n-bit, two's complement integer:

i) What is the most positive you can represent, in decimal? $2^{n-1} - 1$

ii) What is the most negative you can represent, in decimal? -2^{n-1}

2) Fill in the blank below so that the function `mod16` will return the remainder of `x` when divided by 16. The first blank should be a bitwise operator, and the second blank should be a single decimal number:

```
unsigned int mod16(unsigned int x) {
    return x & 15;
}
```

set to 0: & w/ 0

does nothing: & w/ 1

set to 1: | w/ 1

does nothing: | w/ 0

flip the bit: ^ w/ 1

does nothing ^ w/ 0

<<, >>, ~

set to x % 16

1 0 0 1 1 0

2^4 2^1 2^0

mask: 0000 1111

= 15

Connect the definition with the name of the process that describes it.

a) Compiler
b) Assembler
c) Linker
d) Loader

1) Outputs code that may still contain pseudoinstructions. a
2) Takes binaries stored on disk and places them in memory to run. d
3) Makes two passes over the code to solve the "forward reference" problem. b
4) Creates a symbol table. b
5) Combines multiple text and data segments. c
6) Generates assembly language code. a
7) Generates machine language code. b
8) Only allows generation of TAL. b
9) Only allows generation of binary machine code. c

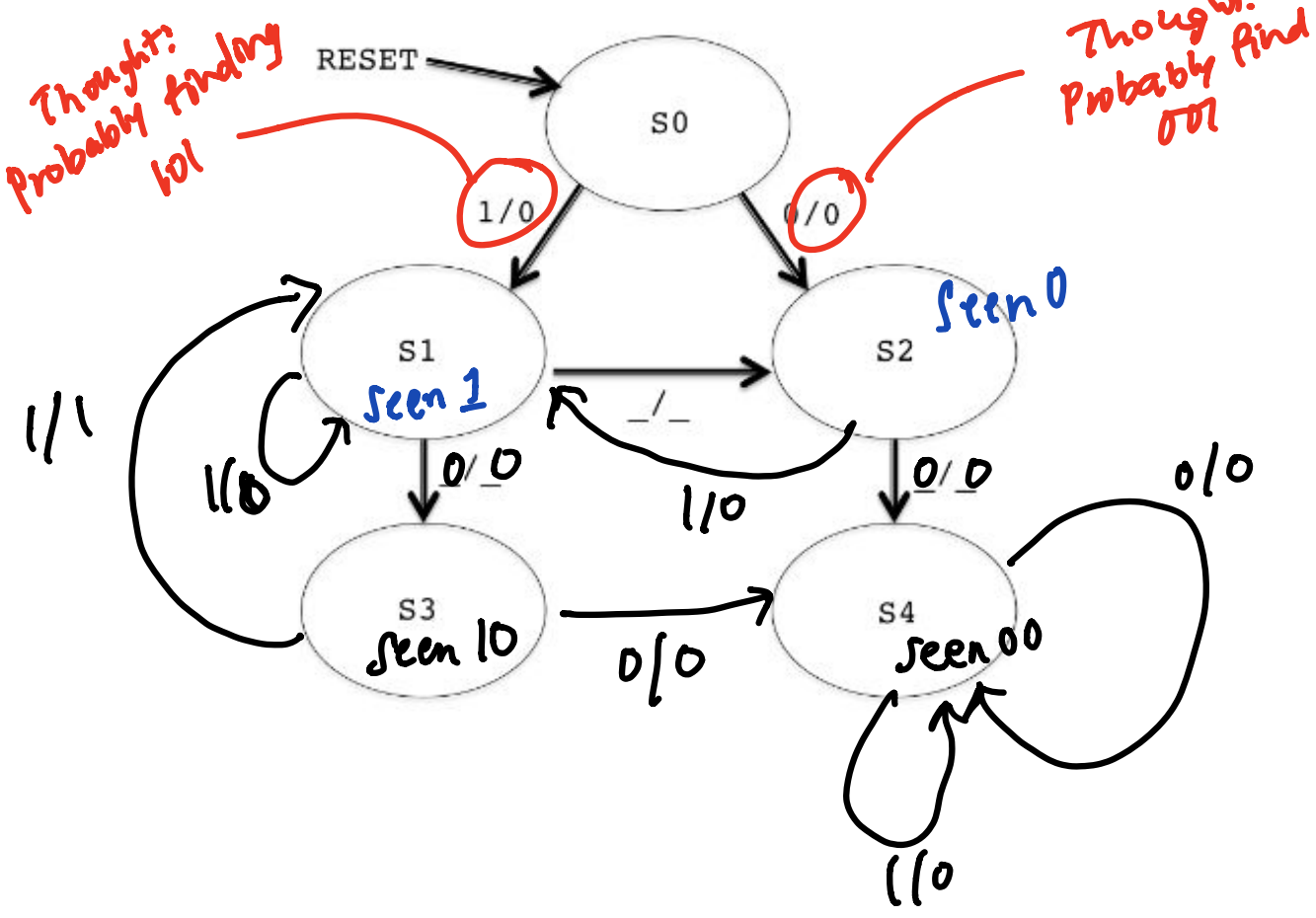
(b) 2-input NOR gates are said to be complete because any Boolean function can be computed with them. Prove this fact. Hint: implement a subset of the standard gates (AND, NOT, OR, NOR, NAND, XOR, XNOR) using just NOR gates, then apply a standard boolean algebra technique using these gates.

see next page for detailed steps (b) (c)

(c) We want to implement a very simple finite state machine that determines its next state by the result of and AND operation on the current state and the input. The output is always the current state. Assume registers have a CLK to Q delay of 5ns, a setup time of 2ns, and a hold time of 3ns. To achieve a clock rate of 25MHz, what is the maximum propogation delay that a NOR gate could have, assuming we are implementing AND as a combination of one or more of the gates built in part (b)?

(d) Complete the state diagram for a finite state machine that outputs 1 if and only if it has just seen the input sequence 101 and it has never seen the input sequence 001. You may add more arrows or more states as you see fit. Provide a brief description of each state.

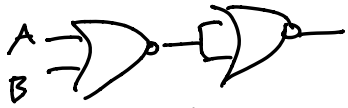
```
Example
Input : 1101010100101
Output: 0001010100000
```

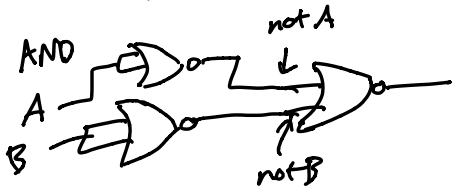


(b)

NAND: $\bar{A}\bar{B} + A\bar{B} + \bar{A}B$

① NOT 

② OR 

③ AND 

AND Truth Table

A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

NOR truth table

A	B	out
0	0	1
0	1	0
1	0	0
1	1	0

NOT

A	out
1	0
0	1

$\overline{AB} \rightarrow \bar{A} + \bar{B}$

$\bar{A}B + A\bar{B} + B\bar{A}$

$\bar{A}B + \bar{A}\bar{B} + \bar{B}A$

(C)

$t_{critical} = t_{clk\text{-}to\text{-}Q} + NOR + NOR + t_{setup}$

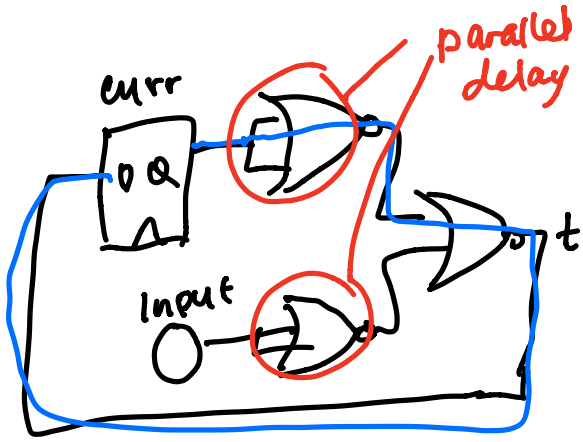
40ns $= 5ns + 2 * NOR + 2ns$

$\frac{40 - 5 - 2}{2} = 16.5\ ns$

$25MHz = 25Ms^{-1}$

$\frac{1}{25 \times 10^6 s^{-1}} = \frac{4 \times 10^{-8}}{40ns}\ s$

$\frac{25 \times 10^6 s^{-1}}{1} \rightarrow \frac{1s}{25 \times 10^6} = 4 \times 10^{-8}$ (via $\frac{1 \times 10^{-6}}{25}$)



$t_{crit\,path} = \underbrace{t_{clk\text{-}to\text{-}Q}}_{\text{Input arrival time}} + \underbrace{t_{CL}}_{\text{longest CL}} + \underbrace{t_{setup}}_{\text{for output}}$

(a) → See next page for detailed steps

Considering the standard 32-bit RISC-V instruction formats, convert `lw t5, 17(t6)` to machine code:

(a) 0x 01 1F AF 03

Prof. Wawrzynek decides to design a new ISA for his ternary neural network accelerator. He only needs to perform 7 different operations with his ISA: XOR, ADD, LD, SW, LUI, ADDI, and BLT. He decides that each instruction should be 17 bits wide, as he likes the number 17. There are no funct7 or funct3 fields in this new ISA.

(b) What is the minimum number of bits required for the opcode field?

3

$2^3 = 8$

(c) Suppose Prof. Wawrzynek decides to make the opcode field 6 bits. If we would like to support instructions with 3 register fields, what is the maximum number of registers we could address?

8 registers

$17 - 6 = 11/3 \approx 3.666$

3 bits

$2^3 = 8$

(a)

lw	I	Load Word		R[rd] = {32'bM[](31),M[R[rs1]+imm](31:0)}

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIMAL
lw	I	0000011	010		03/2

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

lw t5 17(t6)

16-rs1 = x3

imm = 17

rd = t5 = x30

imm[11:0]	rs1	func3	rd	op
000000010001	11111	010	11110	00

0 1 1 F A F 0 3

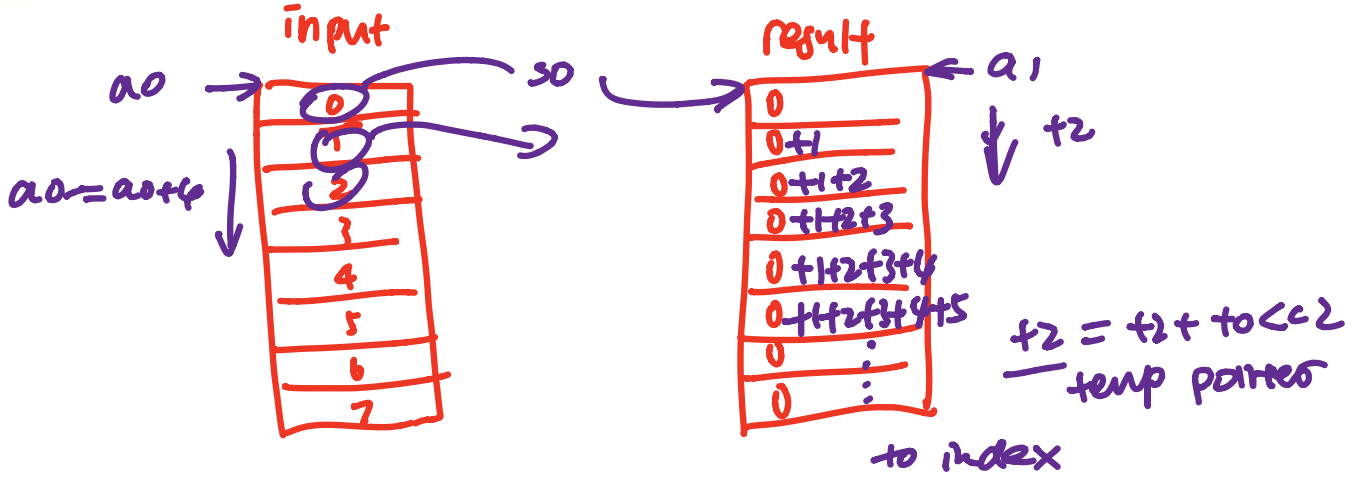
00
11

Assume we have two arrays input and result. They are initialized as follows:

```
int *input = malloc(8*sizeof(int));
int *result = calloc(8, sizeof(int));
for (int i = 0; i < 8; i++) {
    input[i] = i;
}
```

You are given the following RISC-V code. Assume register a0 holds the address of input and register a2 holds the address of result when MAGIC is called by main.

```
main:
    ...
    # Start Calling MAGIC
    addi a1, x0, 8
    jal ra, MAGIC      # a0 holds input, a2 holds result
    # Checkpoint: finished calling MAGIC
    ...
exit:
    addi a0, x0, 10
    add a1, x0, x0
    ecall      # Terminate ecall
MAGIC:
    # TODO: prologue.  What registers need to be stored onto the stack?
    mv s0, x0
    mv t0, x0
loop:
    beq t0, a1, done
    lw t1, 0(a0)
    add s0, s0, t1
    slli t2, t0, 2
    add t2, t2, a2
    sw s0, 0(t2)
    addi t0, t0, 1
    addi a0, a0, 4
    jal x0, loop
done:
    mv a0, s0
    # TODO: epilogue.  What registers need to be restored?
    jr ra
```



(a) Consider the function `MAGIC`. The prologue and epilogue for this function are missing. Which registers should be saved/restored in `MAGIC`'s prologue/epilogue? Select all that apply.

- [ ] t0
- [ ] t1
- [ ] t2
- [x] s0
- [ ] a0
- [ ] a1
- [ ] a2
- [ ] ra
- [ ] x0

callee

don't need to be not calling other functions

(b) Assume you have the prologue and epilogue correctly coded. You set a breakpoint at "Checkpoint: finish calling `MAGIC`" and call `main`. What does `result` contain when your program pauses at the breakpoint? Please write the 8 numbers starting at `result` in the blanks below.

0 1 3 6 10 15 21 28

t5 t6 t7

(c) Translate `MAGIC` into C code. You may or may not need all of the lines provided below.

```
// sizeof(int) == 4
int MAGIC(int* a, int b, int* c) {
    int sum = 0;
    for (int i=0; i<b; i++){
        sum = sum + a[i];
        c[i] = sum;
    }
}
```

a0 a1 a2

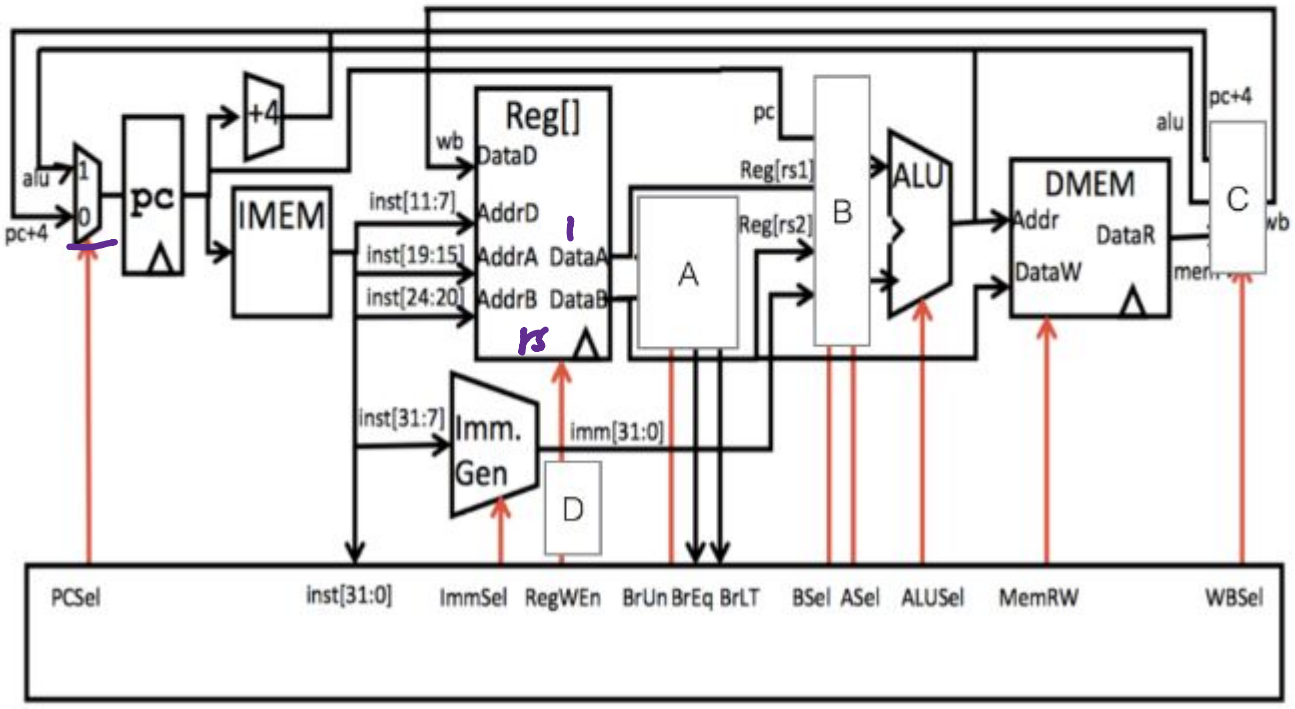
length

We wish to introduce a new instruction into our single-cycle datapath. The instruction **SIZ** (set if zero) works as follows:

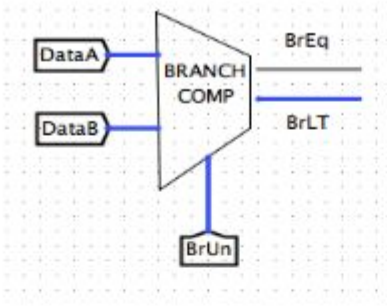
```
if (R[rs2] == 0):
    R[rd] = R[rs1]
```

Given the single cycle datapath below, select the correct modifications in parts (a) - (d) such that the datapath executes correctly for this new instruction (and all core instructions!). You can make the following assumptions:

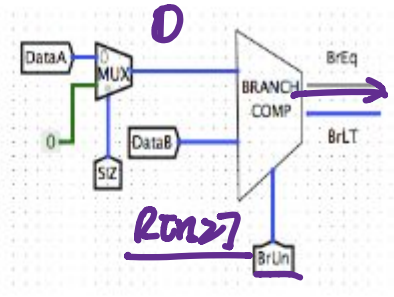
- the `SIZ` signal is 1 if and only if the instruction is `SIZ`
- `ALUSel` is **ADD** when when we a have `SIZ` instruction.
- **the immediate generator outputs ZERO** when we have a `SIZ` instruction.



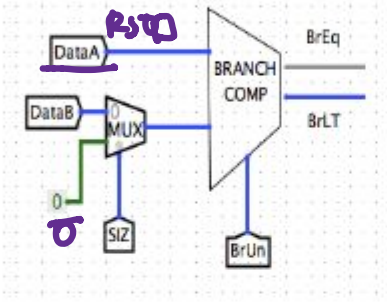
(a) Consider the following modifications to the branch comparator inputs. Which configuration will allow this instruction to execute correctly without breaking the execution of other instructions in our instruction set?



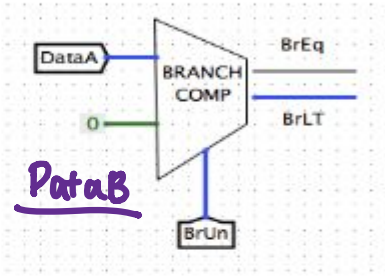
○



●



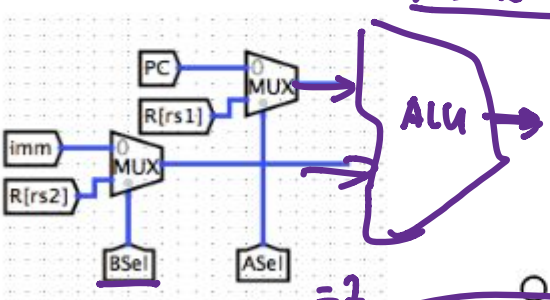
○



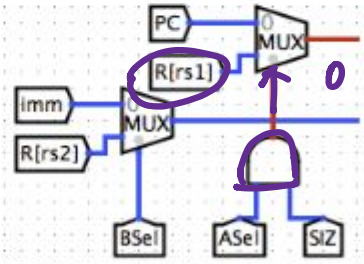
○

(b) Consider the following modifications to the ALU inputs. Which configuration will allow this instruction to execute correctly without breaking the execution of other instructions in our instruction set? Select the configuration that requires **minimum** modifications to the original datapath. Notice in the bottom left choice `BSel` is unused.

BSel = 0
imm = 0
BSel = 1
R[rs2]
= 0

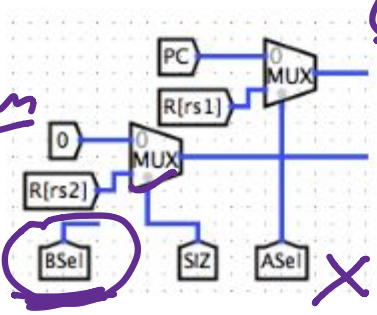


●



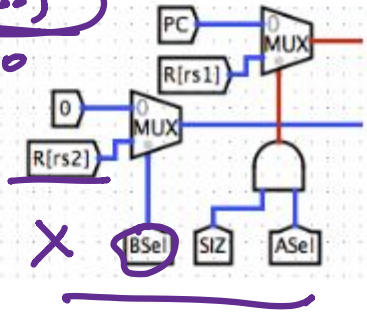
→ work but not minimum

○



X

○

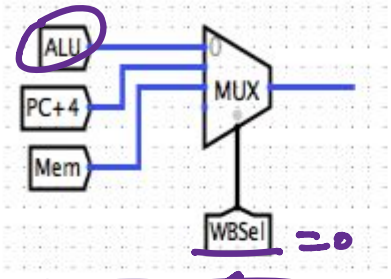


X

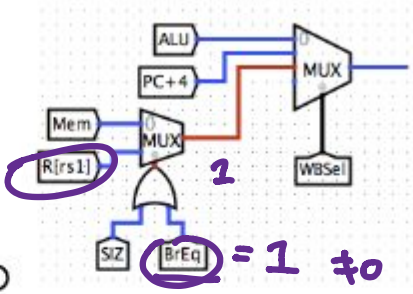
→ about immediate field.

addi BSel = 1 (imm)

(c) Consider the following modifications to the WB mux inputs. Which configuration will allow this instruction to execute correctly without breaking the execution of other instructions in our instruction set? Select the configuration that requires **minimum** modifications to the original datapath.

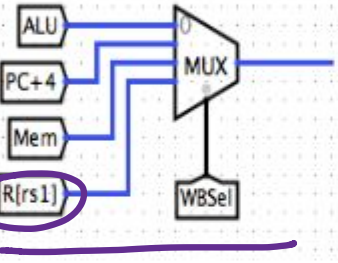


WBSel = 0

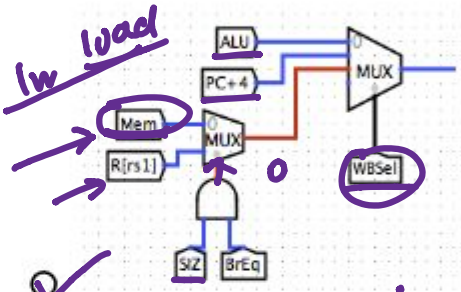


= 1 to

All 4 choices work but not minimum

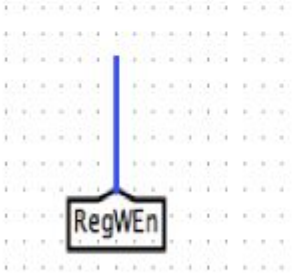


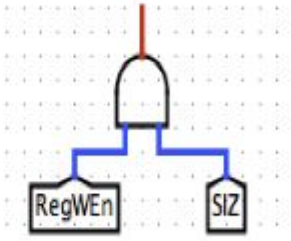
lw load

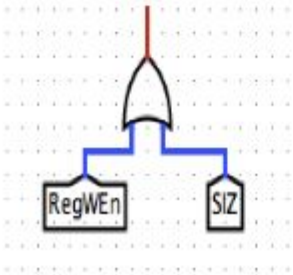


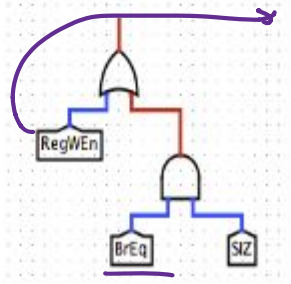
SIZ = 0

(d) Consider the following modifications to the RegWEn inputs. Which configuration will allow this instruction to execute correctly without breaking the execution of other instructions in our instruction set?









needs to default to 0 so when output from ∩ is 0, ∪ will output 0, we don't write back

only when

BrEq = 1 and R[rs2] == 0

SIZ = 1 is SIZ instruction

we want to write to return register

(e) Given your selections above, decide the rest of the control signals for this instruction based on the diagram given at the beginning of the problem. Select `X` when a signal's value doesn't matter. You can assume:

- the `SIZ` signal is 1 if and only if the instruction is `SIZ`
- `ALUSel` is **ADD** when when we a have `SIZ` instruction.
- the immediate generator outputs **ZERO** when we have a `SIZ` instruction.

1. `PCSel`:

   ○ 1 ● 0 ○ X

2. `RegWEn`:

   ○ 1 (Enable) ● 0 (Disable) ○ X

3. `BrUn`:

   ○ 1 (Signed) ○ 0 (Unsigned) ● X

   R[rs2] == 0

4. `BSel`:

   ○ 1 ○ 0 ● X

5. `ASel`:

   ● 1 ○ 0 ○ X

6. `MemRW`:

   ○ 1 (Enable) ● 0 (Disable) ○ X

7. `WBSel`

   ● ALUOut R[rs1] ○ MemOut

   ○ PC + 4 ○ Other: Please specify:_____