EE 109 HW 7
Processor Organization

Name: _____
Due: See Website **Score: _____**

Blackboard ONLY Submission.

In this homework we will design a simple computer system (processor hardware and instruction set) to evaluate Boolean expressions that use AND, OR, and NOT operations. For example, we want to be able to evaluate an expression like:

(~X0+X1)(X2+~X3)(~X1+X3)

Here are some important choices we have made about the processor:

- The equations will use at most 8 unique Boolean variables: X0 through X7.
- Each variable (Xi) will have a single value: 1 = True and 0 = False and be stored in a small 8 row x 1 column memory.
- We will use a register-to-register architecture (inputs and outputs of an instruction will be taken from and stored to a register) with **three 1-bit registers: R0, R1, R2**.
- The computer system will have a single LED to display the result of the expression (ON = 1 = True, OFF = 0 = False).

As we define the instruction set, we will obviously need to have AND, OR, and NOT instructions. In addition, we will need an instruction to LOAD a Boolean variable (one of X0-X7) into a register (either R0, R1, or R2) which performs Rx = Xi. Finally, we will need an OUTput instruction to take the value from one of the registers and save it to the output register which connects to the LED.

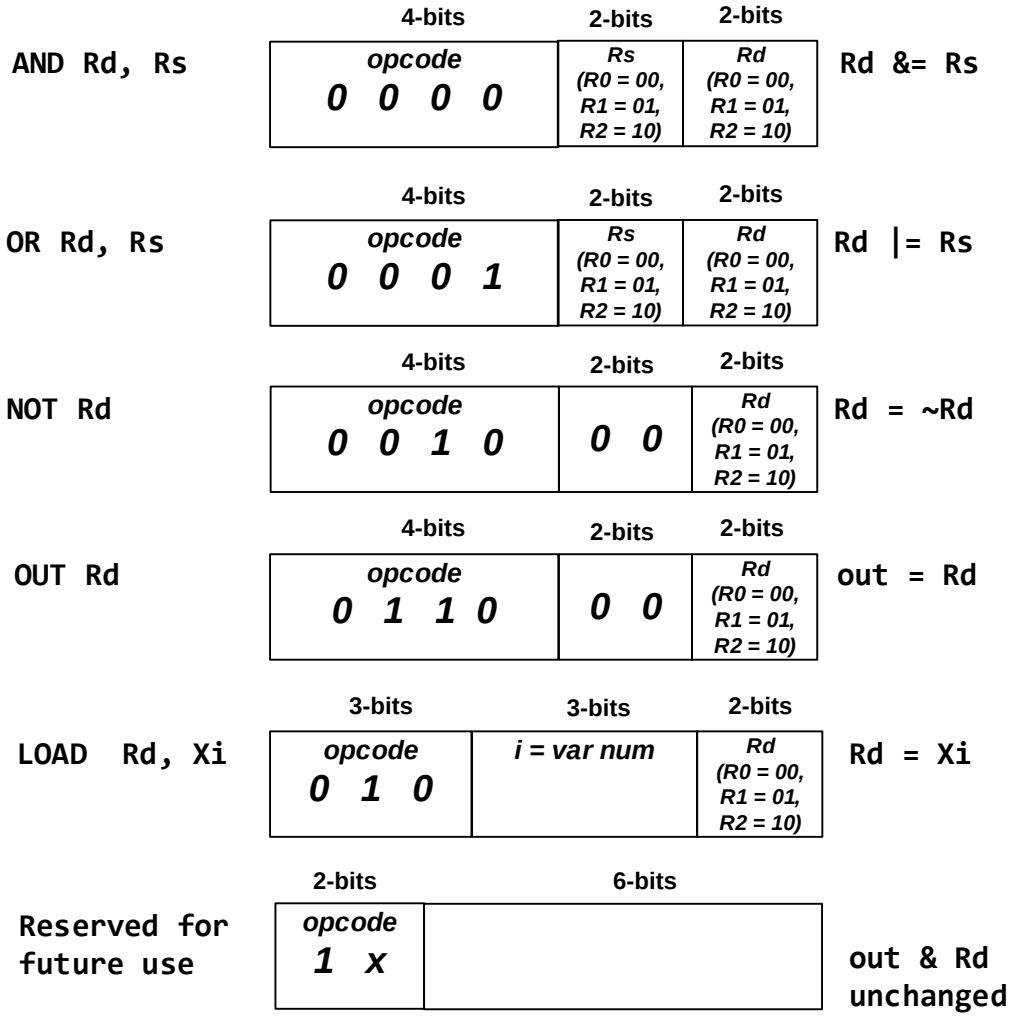
Example Assembly Code

As an example, suppose we need to compute: **(~X0+X1)•~(X2+~X3)** we could use the assembly code:

```
LOAD R0, X0  (R0 = X0)
NOT  R0      (R0 = ~R0)
LOAD R1, X1  (R1 = X1)
OR   R0, R1  (R0 |= R1)
LOAD R1, X2  (R1 = X2)
LOAD R2, X3  (R2 = X3)
NOT  R2      (R2 = ~R2)
OR   R1, R2  (R1 |= R2)
NOT  R1      (R1 = ~R1)
AND  R0, R1  (R0 &= R1)
OUT  R0
```

The instruction set and its machine code format is defined below:

[Note: In the assembly code Rd (the src1/destination) is listed first but in the machine code format Rd comes after Rs.]



HW 7a – Instruction Set Design and Assembly coding

1. Billy Bruin want to add a new instruction **`SET Rd`** which would put the constant **`1`** into Rd without getting any variables from the variable memory or assuming any specific initial register values. Tommy Trojan said this could already be achieved using 3 instructions. Complete the blanks below to show Billy how this can be done:

```
OR   R0, R1
NOT  _R1_
_OR_   R0, R1
```

2. Fill in the blanks below in the assembly code to correctly evaluate the expression below:

(~X0+X1)(X2+~X3+~X4)

Instruction Address	Instruction
0	LOAD R0, _X0__
1	NOT R0
2	LOAD _R1__, X1
3	OR R0, R1
4	LOAD _R1__, X2
5	LOAD R2, _X3__
6	NOT R2
7	OR _R1__, R2
8	LOAD _R2__, X4
9	NOT _R2__
10	OR _R1__, R2
11	AND _R0__, R1
12	OUT R0

3. Given the code you found above, correctly convert the LAST 5 instructions to 8-bit machine code. Enter each instruction's machine code as 2 hex digits preceded by '0x'.

Assembly	Machine Code
8. LOAD ___, X4 =	8.. 0x 5 2
9. NOT ___	9. 0x 2 2
10. OR ___, R2	10. 0x 1 9
11. AND ___, R1	11. 0x 0 4
12. OUT R0	12. 0x 6 0

4. Suppose the processor could have more than 3 registers if needed. How many registers (minimum) would be required to be able to correctly evaluate the expression below (in any order you like as long as it does not alter the intended result).

(X1~X2 + ~X3X4)(X5~X6 + ~X7X8)

4 (1 for result of **(X1~X2 + ~X3X4),** 1 for result of **(X5~X6)**, 1 for result ~X7 and 1 for X8)

5. Given the expression from the previous problem (above), suppose you could transform the equation by applying Boolean algebra theorems. How many registers (minimum) would be required to be able to correctly evaluate the expression.

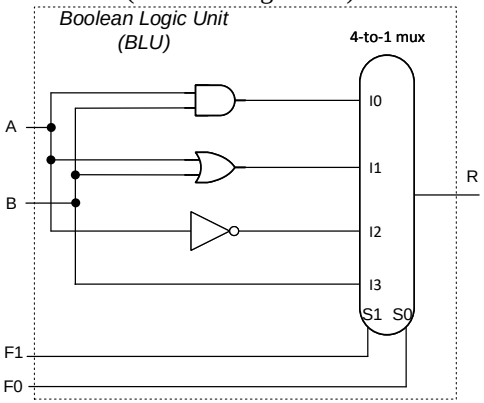
(X1~X2X5~X6 + X1~X2~X7X8 + ~X3X4X5~X6 + ~X3X4~X7X8) only needs 3 registers

6. **True / False**: Putting aside the impact on the hardware, we could add a 4th register (R3) without modifying the **bit layout** of the machine code format of the instruction set.
   True: we can just code 11 in the Rs and Rd fields of the machine code format

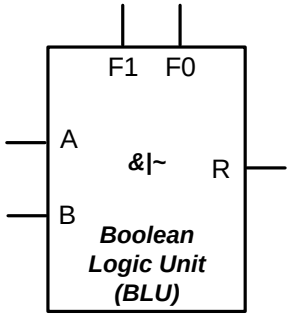
Processor Datapath:

To compute the Boolean equations we will design a unit to perform AND, OR, NOT, and pass B.

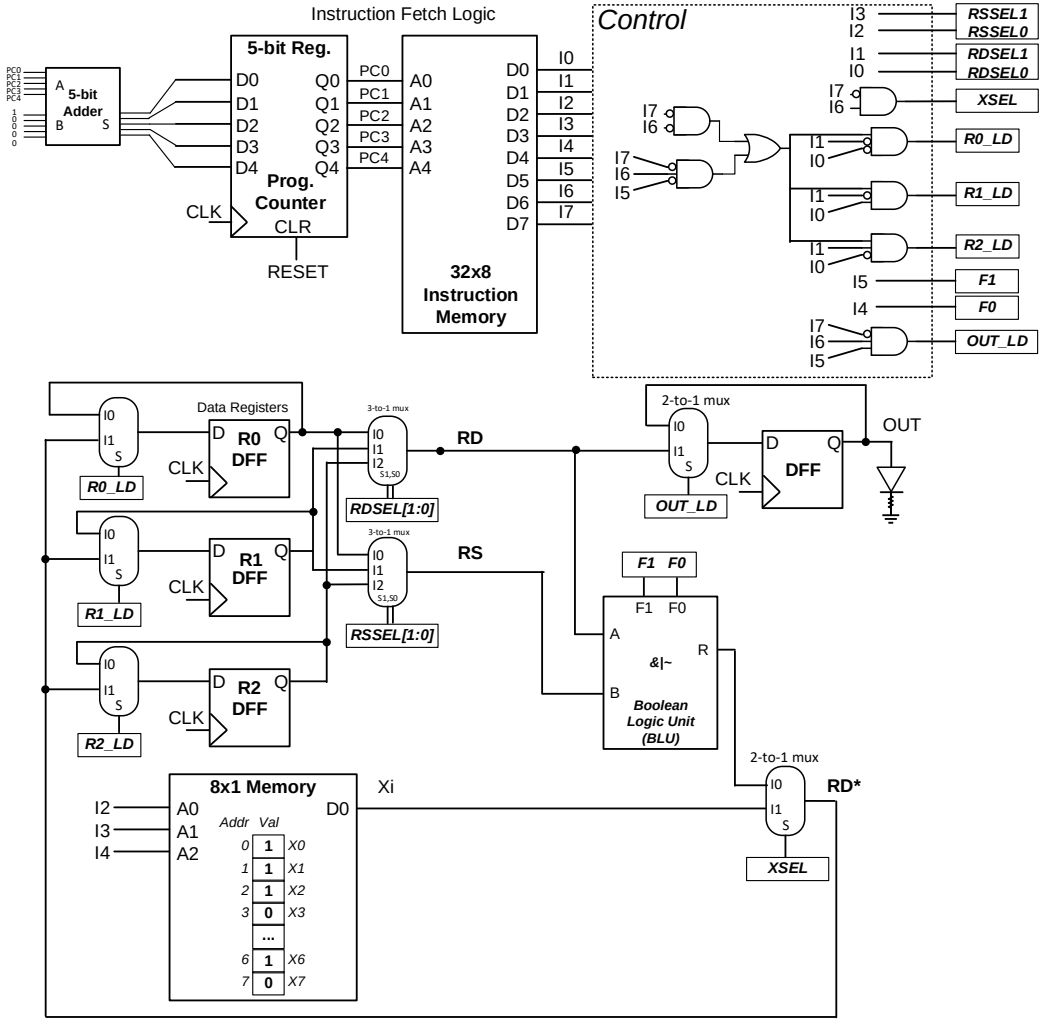
Rather than calling it an ALU (Arithmetic and Logic Unit) we will call it a BLU (Boolean Logic Unit)



Internal schematic of the Boolean Logic Unit



Block Symbol for the Boolean Logic Unit



HW 7b – Processor Hardware Organization

Study the schematic on the previous page and try to understand why the various connections exist and then answer the questions.

Look at both the machine code format and the processor datapath to answer the following questions.

1. What processor signals should be connected to RDSEL[1:0]? **I1, I0**
2. What processor signals should be connected to RSSEL[1:0]? **I3, I2**
3. What processor signals should be connected to F1,F0 of the BLU? **I5, I4**
4. What signals should be connected to the address inputs of the 8x1 variable memory (i.e. A2, A1, A0 should be connected to which signals)? **I4, I3,I2**
5. For what instruction(s) should XSEL be a 1? **LOAD**
6. What value should XSEL be during an OUT instruction? **Don't care**
7. True / False: If desired, the F1 and F0 signals to the BLU can be considered "Don't Care" during a LOAD instruction? **True**
8. Select which of the following is a correct implementation of OUT_LD.
   a) I6 and I1' and I0'
   b) I7' and I6 and I5
   c) I7' and I6
   d) I7 and I6' and I5' and I4
   e) None of the above

This answer results if you write a truth table for the first 4-bits of an instruction (i.e. the opcode bits). OUT_LD need only be true for the OUT instruction (opcode = 0110, but no other instruction starts with 0111 so that would output a 'd' = don't care and allow us to simplify our equation).

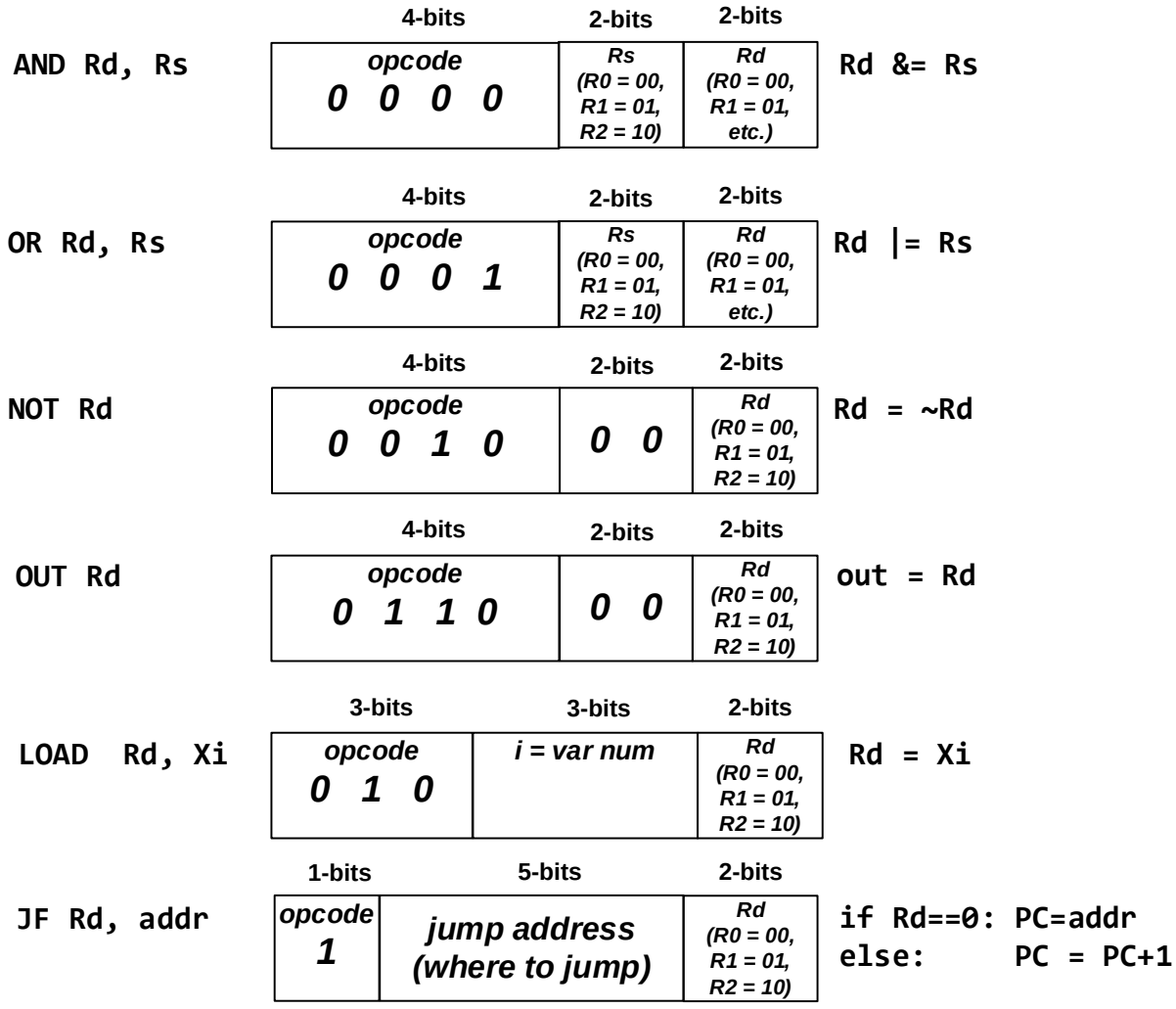
9. Select which of the following is a correct implementation of R2_LD.
   a) ((I7' and I6') or (I7' and I6 and I5')) and (I1' and I0)
   b) I1 and I0'
   c) ((I7' and I5') or (I7' and I6' and I5)) and (I1 and I0')
   d) I7' and I1 and I0'

To load (change) R2 the bits I1,I0 must be 1,0 respectively = 2 decimal. This is where the I1' and I0 comes from. But not all instructions should change R2. So again we can write a truth table for the first 4 bits (opcode bits) and indicate which instructions will cause the Rd register to update. If we construct a K-map and group 1's, it should yield the first 2 terms in the selected answer.

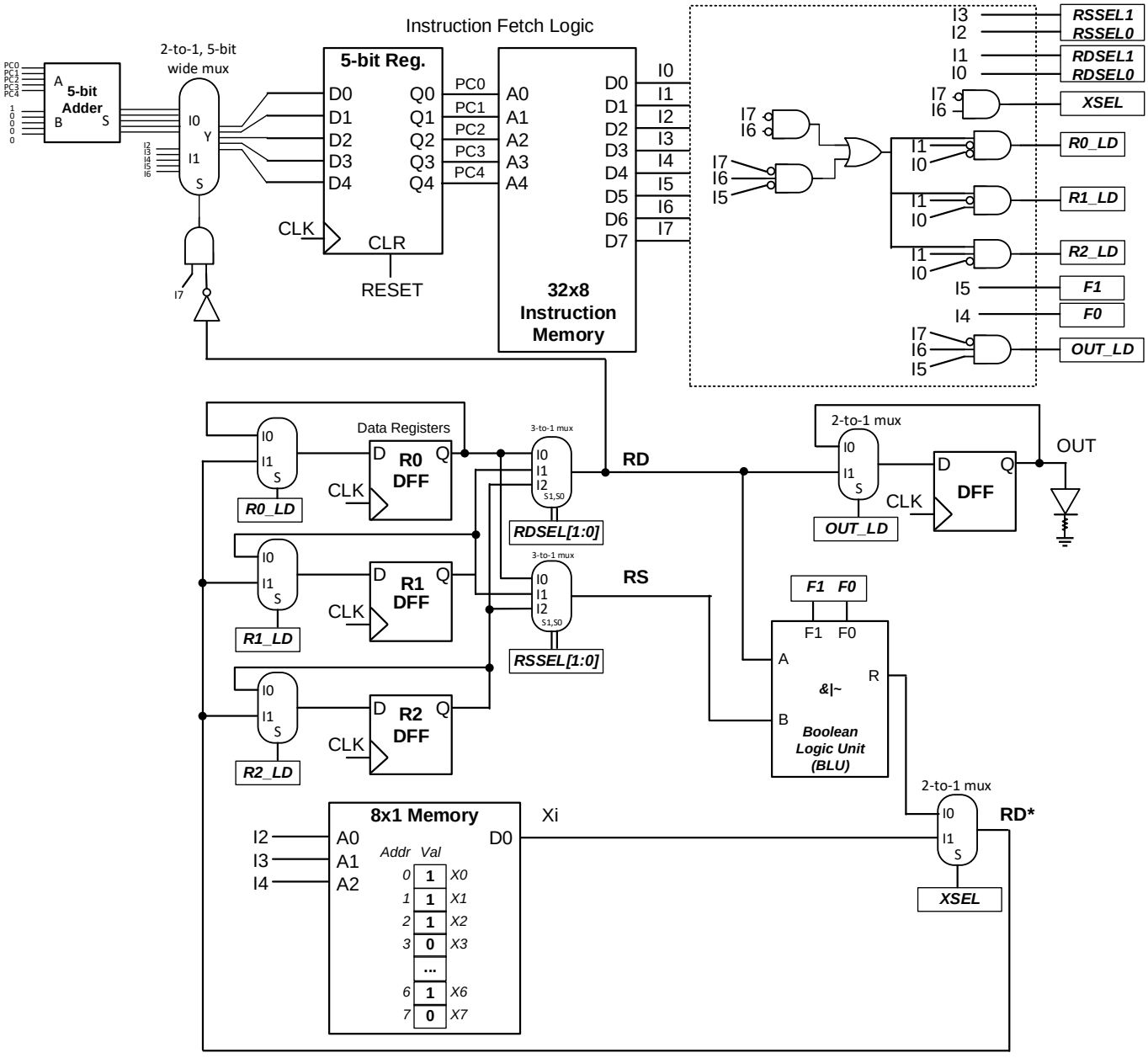
Teresa Trojan noticed that for certain Boolean expressions we may not have to evaluate the entire expression to find the answer. For example, in the expression below if the first term (~X0+X1) is false, the whole expression will be false. Rather than wasting time evaluating other terms, we could immediately "jump" to the last OUT instruction. Teresa proposed adding a new instruction call JF (Jump if False). It would have format: **`JF Rd,addr`** and cause the Program Counter to jump to **`addr`** if Rd is False. Otherwise, execution will continue sequentially.

We have updated the instruction set and the datapath on the following pages. Please answer the related questions that follow.

Updated Machine Code Format with JF instruction



Updated (but incomplete) Processor Datapath for JF instructions



10. Complete just the JF instruction at instruction address 4 (we assume the rest is the same from the earlier question regarding this expression).

(~X0+X1)(X2+~X3+~X4)

Instruction Address	Instruction
0	LOAD R0, _X0__
1	NOT R0
2	LOAD _R1__, X1
3	OR R0, R1
4	JF _R0__, _13_
5	LOAD ___, X2
6	LOAD R2, ___
7	NOT R2
8	OR ___, R2
9	LOAD ___, X4
10	NOT ___
11	OR ___, R2
12	AND ___, R1
13	OUT R0

11. Billy Bruin noticed that a similar tactic to improve performance could be used for certain other expressions that involved ORing term such as: **(X0•X1)+(X2•X3•X4)** and he proposed adding a **JT** (Jump if True) instruction. Teresa Trojan said that wouldn't be necessary and suggest instead of an instruction like **JT R0, addr** one could simply use a 2 instruction sequence:

```
_NOT___ __R0__
_JF___ _R0__, addr
```

12. Looking at the datapath for the new JF instruction, what processor signals should be connected to input 1 of the 2-to-1, 5-bit mux that feeds the PC?

    I6,I5,I4,I3,I2 (from the description of the machine code these are the bits that correspond to the address to jump to)

13. What is the minimal logic for the select bit of that mux (i.e. the 2-to-1, 5-bit mux that feeds the PC)?

    I7 and (not Rd) - I7 will be 1 only for a JF instruction but we only want to jump if the bit in the selected register is 0 (i.e. false), so we 'AND' I7 with ~RD.