Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet.
- No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 7 questions, some with multiple parts, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

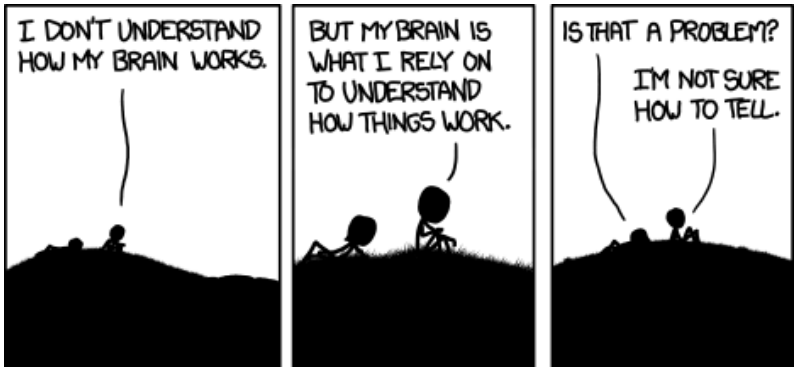
Do not start the test until instructed to do so!

Name Solution

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

signed


I DON'T UNDERSTAND HOW MY BRAIN WORKS.
BUT MY BRAIN IS WHAT I RELY ON TO UNDERSTAND HOW THINGS WORK.
IS THAT A PROBLEM?
I'M NOT SURE HOW TO TELL.

xkcd.com

1. A digital designer wants to implement a circuit, which will take three 1-bit inputs (x, y and z) and produce two outputs F and G. The functionality of the circuit is defined by the truth table below:

x	y	z	F	G
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

a) [12 points] Derive a Boolean expression for the output labeled F. Simplify the expression for F algebraically.

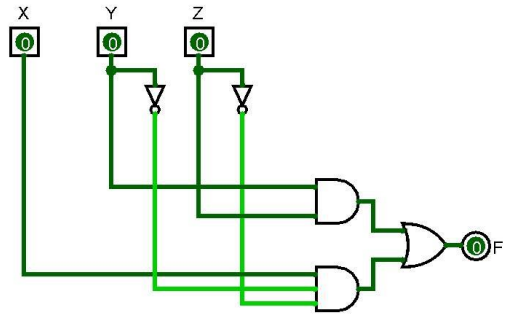
First, form product terms for the rows in which F == 1; a variable is complemented if it equals 0 in that row and uncomplemented otherwise, and sum the products:

$$F = \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot z$$

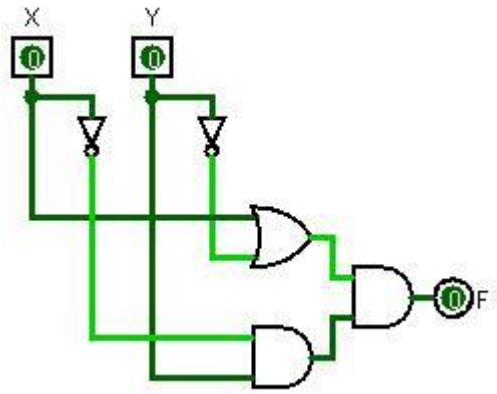
To simplify, grouping, distribution and a complement property yield:

$$\begin{aligned} F &= \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot \bar{z} + x \cdot y \cdot z \\ &= (\bar{x} + x) \cdot y \cdot z + x \cdot \bar{y} \cdot \bar{z} \\ &= y \cdot z + x \cdot \bar{y} \cdot \bar{z} \end{aligned}$$

b) [10 points] Sketch a diagram for the portion of the circuit that would produce the output F.



2. [10 points] Write a Boolean expression for the output F of the circuit below. Simplify the expression algebraically.



$$
\begin{aligned}
F &= \left(x + \overline{y}\right) \cdot \left(\overline{x} \cdot y\right) \\
&= x \cdot \overline{x} \cdot y + \overline{y} \cdot \overline{x} \cdot y \\
&= \left(x \cdot \overline{x}\right) \cdot y + \left(y \cdot \overline{y}\right) \cdot \overline{x} \\
&= 0 \cdot y + 0 \cdot \overline{x} \\
&= 0
\end{aligned}
$$

3. [10 points] Consider the Shift left 2 unit labeled 3 on the included datapath diagram. When it's applied to its input value, the two high bits of the input are lost (shifted out). Why does losing those two bits not matter? Be specific.

The input to the Shift unit is the sign-extended immediate from the instruction; therefore the 16 high bits of the input are all the same (copies of the high bit of the 16-bit immediate).

Therefore, the two high bits of the input are either both 0 or both 1, and either way just indicate the sign of the value that is represented.

Chopping two of those bits off will not affect the represented value.

4. Consider the AND gate labeled 4 on the included datapath diagram. Suppose the gate was replaced with an OR gate.

a) [8 points] What would be the effect if the current instruction is `beq`? Explain fully.

If the current instruction is beq, then Branch will be 1. If we had an OR gate there, the control signal to the MUX would be 1 whenever Zero was 1 or Branch was 1.

Therefore, every beq instruction would actually branch, whether the registers involved were equal or not.

b) [8 points] What would be the effect if the current instruction is `add`? Explain fully.

If we had an OR gate there, the signal to the MUX would also be 1 whenever Zero was 1.

So, if the result from performing the addition was 0, a branch would be attempted. In other words, whenever the operands to add were negatives of each other (additive inverses), a branch would be attempted.

The address of the instruction to be branched to would be computed from the second operand to add, and the low 16 bits of the add instruction (a register number, shift field and function bits). It's unlikely the resulting address will even be that of an instruction. In any case, havoc will ensue.

5. Consider the MUX labeled 5 on the included datapath diagram. Suppose the control signal for that MUX was stuck-at-0.

a) [8 points] Assuming no other errors occurred, which of the supported instructions would always still execute correctly?

The only instruction that needs for the MUX control to be set to 1 is jump (j). So, all the other instructions will still work correctly:

add, sub, and, or, slt, lw, sw, beq

b) [8 points] Would any of the other supported instructions would possibly execute correctly? If so, under what circumstances?

Nope… unless…

If the target of the jump instruction was the next sequential instruction (after the jump), then the address of that instruction would be PC + 4, and so the correct instruction would be executed next.

Since the address computed for jump cannot be passed through the MUX, and that address should ALWAYS be used for a jump instruction, jump can never execute correctly, except for the special circumstance described above.

6. Suppose the following instruction is being executed: `add $t1, $t2, $t3`

When that instruction is executed, the datapath hardware will perform some actions that are logically unnecessary for that instruction (although they would be necessary for some other instructions).

An action might be a computation or the delivery of a signal or data to a device.

Identify two such actions, and for each explain why the fact that the hardware performs that (unnecessary) action does not cause any difficulties.

[6 points] 1st action

[6 points] 2nd action

There are lots of candidates; anything that happens but isn't relevant to executing an R-type instruction will do, including:

- **computation of the jump target address**
- **computation of the branch target address**
- **sending Instruction[20:16] to the MUX before the Register file**
- **sign-extending Instruction[15:0]**
- **sending the sign-extender output to the MUX before the ALU**
- **sending the output from Read data 2 to the Write data input on the memory unit**
- **sending the Read data output from data memory to the MUX after the memory unit**

Here are sample answers for two of the options listed above:

The jump target address is calculated.

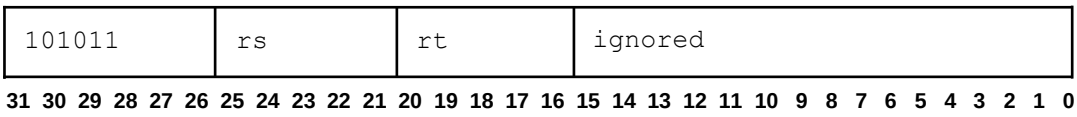
That is harmless because, if the current instruction is add, then the Jump control signal is set to 0, which means the jump target address does not pass beyond the final MUX at the top of the datapath.

Bits 15:0 of the instruction are sign-extended.

This is harmless because, if the current instruction is add, then ALUSrc is set to 0, so the sign-extended immediate does not go to the ALU, and Branch is set to 0, so the sign-extended immediate does not go past the MUX controlled by the AND gate.

7. [14 points] Consider the following proposed instruction for the simplified single-cycle MIPS32 datapath discussed in class:

```
sas    $rt, $rs              # Mem[rs + rt] = GPR[rt]
```



The instruction adds the values in registers `$rs` and `$rt` to obtain an address, and stores the contents of register `$rt` at that address.

Of course, this could already be accomplished by a sequence of supported instructions, but that would require more than one clock cycle. Supporting this new instruction would also require modifying the internals of the Control unit to recognize the `opcode` field for the new instruction, but assume that's easily accomplished.

Haskell Hoo IV, who proposed the new instruction, insists that it can be added to the current datapath design (as shown on the datapath diagram) with no changes other than to the internals of the Control unit (to recognize the new opcode). That is, there will be no need to add any new hardware to the datapath, nor will there be a need for any new control signals.

If Haskell Hoo IV is correct, explain how the eight existing control signals (excluding ALUop) would need to be set. Be sure to consider (and indicate) if any of the signals are don't-cares.

If Haskell Hoo IV is incorrect, describe at least one hardware modification that must be made to the existing datapath in order to support `sas`, and explain why that modification is necessary. (Do not show control signal settings in that case.)

RegDst	D/C	RegWrite == 0, so this doesn't matter
Jump	0	must be 0 to disable jump address selection
Branch	0	must be 0 to disable branch address selection
MemRead	0	must be 0 to prevent invalid memory access
MemtoReg	D/C	RegWrite == 0, so this doesn't matter
MemWrite	1	must be 1 so we can write to memory
ALUSrc	0	must be 0 so we add the register operands
RegWrite	0	must be 0 to prevent invalid register write

Haskell is correct. We can use the ALU to compute the address and, at the same time, the second value read from the register file ($rt) will be sent to the Write data input on the Data memory unit.

Given that, it's just a matter of setting the control signals properly.