Instructions:

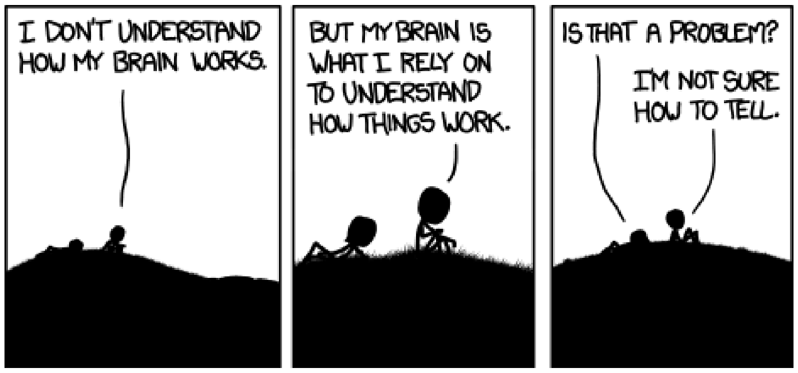
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet.
- No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test 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 G. Simplify the expression for G algebraically.

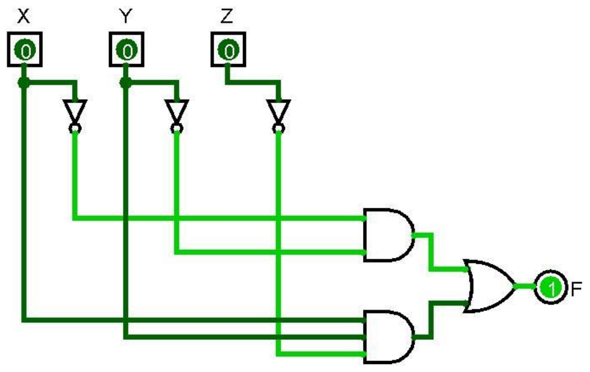
First, form product terms for the rows in which G == 1; a variable is complemented if it equals 0 in that row and uncomplemented otherwise, and sum the products:

$$G = \overline{x}\cdot\overline{y}\cdot\overline{z} + \overline{x}\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z}$$

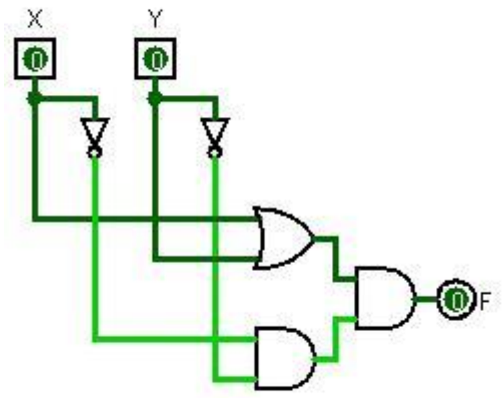
To simplify, grouping, distribution and a complement property yield:

$$\begin{aligned} G &= \overline{x}\cdot\overline{y}\cdot\overline{z} + \overline{x}\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} \\ &= \overline{x}\cdot\overline{y}\cdot\left(\overline{z} + z\right) + x\cdot y\cdot\overline{z} \\ &= \overline{x}\cdot\overline{y}\cdot 1 + x\cdot y\cdot\overline{z} \\ & \overline{x}\cdot\overline{y} + x\cdot y\cdot\overline{z} \end{aligned}$$

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



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



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

3. [10 points] Suppose the current instruction is `add`. Consider the Shift left 2 unit labeled 3 on the included datapath diagram. Even if the instruction that is being executed is `add`, the sign-extender operates on the low 16 bits of the instruction. Why, in that case doesn't this cause any problems? Be specific.

The output from the Shift unit does go to the hardware that computes the branch target address, and that address will be computed, but the branch target address will not be used, since Branch will be 0 for add.

So, nothing that's computed from the Shift unit output causes anything to be stored, or affects anything that's stored.

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?

This MUX determines which bits from the instruction are used to specify the register to be written to.

If the control signal is stuck-at-0, the MUX will always pass bits 20:16 to specify the write register.

That's correct for lw, but incorrect for all the R-type instructions. And, of course, it doesn't matter how we specify the write register for instructions that don't write to a register (since RegWrite will be 0 for those).

So the following instructions would execute correctly under these circumstances:

lw, sw, beq, and j

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

An R-type instruction would still execute correctly if bits 20:16 and 15:11 were the same. In other words, if the destination register was the same as the register for the second operand (right operand), this error won't cause any problems.

6. Suppose the following instruction is being executed: `beq  $t1, $t2, exit`

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 a beq instruction will do, including:

- computation of the jump target address
- sending any input to the MUX before the Register file
- sending a write register number to the Register file
- sending the sign-extender output to the MUX before the ALU
- sending the funct field bits (5:0) to the ALU control unit
- sending the output from Read data 2 to the Write data input on the memory unit
- sending the ALU result to the address input on the memory unit
- sending the Read data output from data memory to the MUX after the memory unit
- sending the ALU result to the MUX after the memory unit
- sending data to the Write data port on the Register file

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

Sending a write register number to the register file.

This is harmless because RegWrite is set to 0.

Sending bits 5:0 of the instruction to the ALU Control unit.

This is harmless because the main Control unit will made the decision that the ALU should perform an addition operation, and simply direct the ALU Control unit to pass that on.

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[rs]
```

101011	rs	rt	ignored

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

The instruction adds the values in registers `$rs` and `$rt` to obtain an address, and stores the contents of register `$rs` 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	
Jump	
Branch	
MemRead	
MemtoReg	
MemWrite	
ALUSrc	
RegWrite	

Haskell is incorrect.

We can use the ALU to compute the address but, 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.

The current design doesn't provide any way to send the first operand ($rs) to the Data memory unit. We'd have to add:

- another path to send the value from $rs to the Data memory unit
- a MUX so we could select whether the value from $rs or $rt goes to the Data memory
- (of course) a new control signal for that MUX