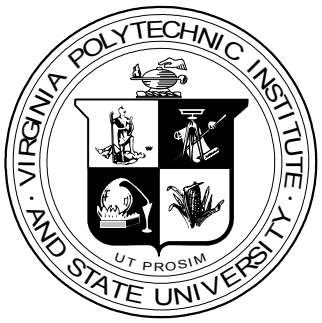


Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet and the MIPS reference card. No calculators or other computing devices may be used.
- 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 10 questions, 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

1. [6 points] The MIPS platform uses 2's complement representation for signed integers. What important advantage(s) does this provide, compared to using sign-magnitude representation?

The primary advantage is that the same algorithm can be used for both addition and subtraction; this simplifies the design of the ALU.

Lesser advantages include the fact that there is only one representation for zero, and so a (very slightly) increase in the range of representation.

There is no significant gain in efficiency for the individual operations, although there is a slight gain for in addition of signed numbers. For both representations, the high-order bit indicates the sign of the number and the low-order bit indicates whether the number is even or odd.

---

2. [12 points] Complete the truth table defining the output for a 1-bit full adder (with carry-in).

A	B	Cin	Sum	Cout
0	0	0	**0**	**0**
0	0	1	**1**	**0**
0	1	0	**1**	**0**
0	1	1	**0**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**1**
1	1	0	**0**	**1**
1	1	1	**1**	**1**

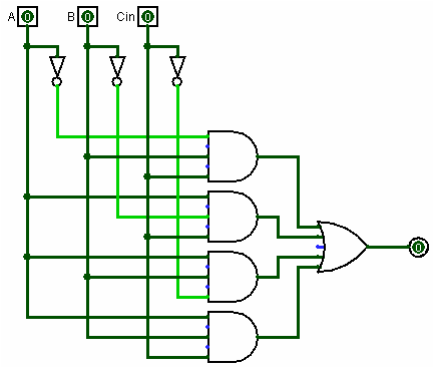
3. [10 points] Write a sum-of-products expression for the sum bit produced by the 1-bit full adder in the previous question.

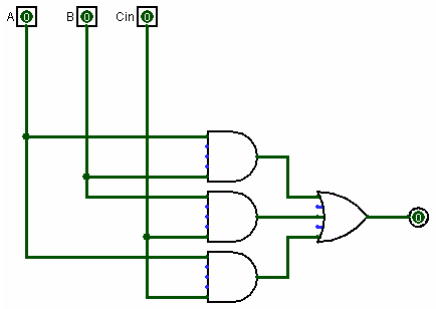
Sum = ~A*~B*Cin + ~A*B*~Cin + A*~B*~Cin + A*B*Cin

4. [10 points] Using only AND, OR and NOT gates, draw a circuit that computes the carry-out bit for the 1-bit full adder circuit from question 2.

From the truth table: `Sum = ~A*B*Cin + A*~B*~Cin + A*B*~Cin + A*B*Cin`

This yields the solution at left below. However, the expression above can be simplified because the last two terms reduce to A*B, and the negated variables in the first two terms can be viewed as don't cares. That yields the simpler solution at right below.



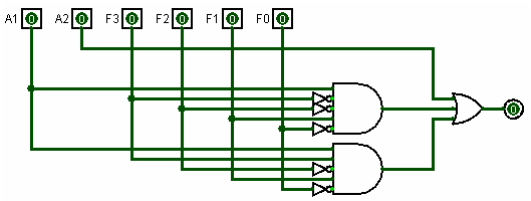


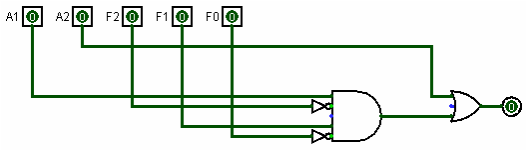
5. [12 points] Using only AND, OR and NOT gates, draw a circuit that computes the function `Out1` defined by the truth table given below. The "X" entries indicate that the corresponding input is not used to determine output values in that row.

Inputs						Outputs		
A1	A2	F3	F2	F1	F0	Out1	Out2	Out3
0	0	X	X	X	X	0	1	0
X	1	X	X	X	X	1	1	0
1	X	0	0	0	0	0	1	0
1	X	0	0	1	0	1	1	0
1	X	0	1	0	0	0	0	0
1	X	0	1	0	1	0	0	1
1	X	1	0	1	0	1	1	1

The corresponding expression is: `Out1 = A2 + A1*~F3*~F2*F1*~F0 + A1*F3*~F2*F1*~F0`

That yields the solution at left below. However, the expression can be simplified by factoring A1*~F2*F1*~F0 out of the last two terms and reducing the resulting term ~F2 + F2 to 1: `Out1 = A2 + A1*~F2*F1*~F0`. This yields the simpler solution at right below.





6. [6 points] What is the conceptual difference between a *combinational* logic circuit and a *sequential* logic circuit?

The output of a combinational circuit depends only on the current state of its inputs. The output of a sequential circuit depends both on the state of its inputs, and on the previous value(s) of its output.

For questions 7 – 10, feel free to refer to the copies of Fig 5.17 and 5.18 from P&H that were distributed along with the test.

7. [10 points] The single-cycle datapath shown in Fig 5.17 includes two different memory modules, one for data and one for instructions. Given that the design is for a single-cycle machine, is it logically necessary to have two different memories? Or is it convenient but not strictly necessary? Or is it not even justified by convenience? Justify your conclusion carefully. (You might want to base your argument on an example drawn from the MIPS instruction set.)

In a single-cycle design, it is not possible to perform two different memory access operations (on a single memory unit) in a single cycle. For example, in executing the `sw` instruction, we must fetch the instruction from memory, use the ALU to compute a target address, and then write a value into memory at that address.

Therefore, it is logically necessary to have separate memory units for instructions and for data.

---

8. [8 points] Refer to Fig 5.17 and 5.18.

   a) Explain briefly why, in order to execute the instruction `lw`, the ALUSrc signal must be set to 1 instead of 0.

The ALUSrc signal determines whether the second operand to the ALU is retrieved from the register file (ALUSrc = 0) or from the instruction (ALUSrc = 0) being executed (after sign-extension). For a load word instruction, a value is being retrieved from a memory location whose address is contained within the instruction.

   b) Explain briefly why, in order to execute the instruction `beq`, the setting of the MemtoReg signal is a don't care; that is, it doesn't matter whether that signal is a 0 or a 1.

The MemtoReg signal determines whether the value that is sent to be written to the register file is taken from the data memory or is computed by the ALU. However, the branch-on-equal instruction does not cause anything to be written to the register file (the RegWrite signal is set to low), so it doesn't matter what setting is used for MemtoReg.

9. [10 points] A friend proposes to simplify the single-cycle datapath shown in Fig 5.17 by eliminating the control signal MemtoReg. The multiplexor that currently takes MemtoReg as an input will instead use the ALUSrc control signal. Will the proposed modification work correctly? Why or why not?

According to Fig 5.18, the value of MemtoReg only matters when R-type instructions and `lw` are executed. In both those cases, the values for MemtoReg and ALUSrc are identical. For the other instructions (`sw` and `beq`), the value of MemtoReg is a don't care, so no matter what the value of ALUSrc might be, it will do nicely.

Therefore, yes, the design change will work correctly.

Of course, this may fail if additional instructions are to be supported, but that is not relevant to the question that was posed.

---

10. [16 points] A *stuck-at-x fault* occurs when a signal is always x regardless of what it should be. Consider the single-cycle datapath shown in Fig 5.17. Which instructions from the set {`add, lw, sw, beq`}, if any, would not work correctly under the following conditions? Each part is independent; i.e., there is never more than one signal suffering a stuck-at-x fault.

a) RegWrite is stuck at 0 **can't write anything into the register file** **add, lw**

b) MemRead is stuck at 0 **can't read from the data memory** **lw**

c) RegRead is stuck at 0 **no such signal**

d) ALUOp is stuck at 00 **from Fig 5.18, we couldn't set control for** **add, beq**