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 more space to answer a question, you are probably thinking about it the wrong way.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 6 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, sign your fact sheet, and turn in the test and fact sheet.
- Note that failing to return this test, and discussing its content with a student who has not taken it are violations of the Honor Code.

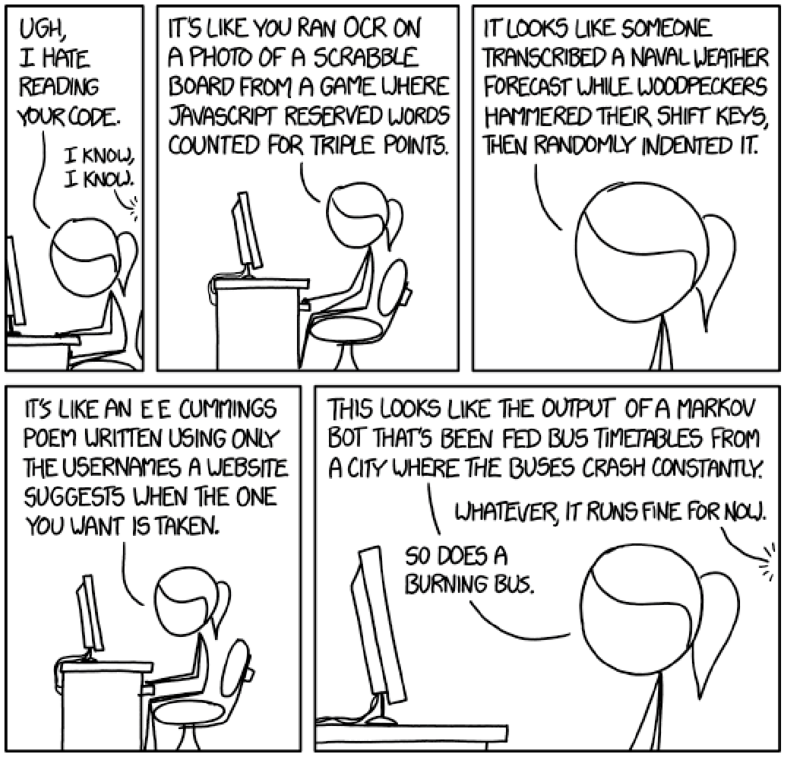
Do not start the test until instructed to do so!

Answers to questions are in blue; commentary explaining the answers is in green.

Name Solution
printed

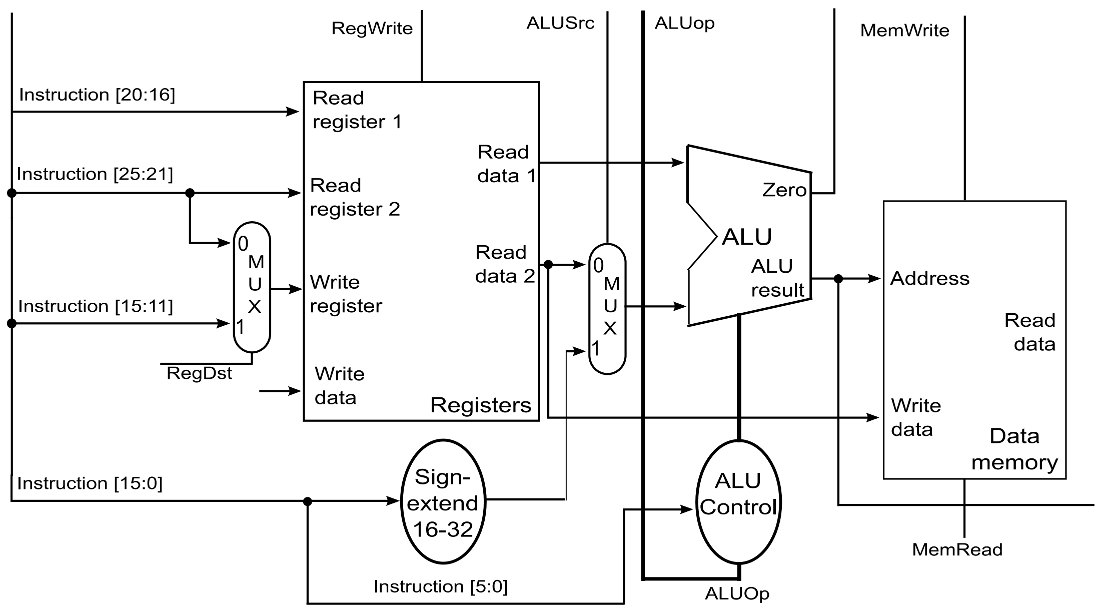
Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

____
signed


UGH, I HATE READING YOUR CODE.
I KNOW, I KNOW.
IT'S LIKE YOU RAN OCR ON A PHOTO OF A SCRABBLE BOARD FROM A GAME WHERE JAVASCRIPT RESERVED WORDS COUNTED FOR TRIPLE POINTS.
IT LOOKS LIKE SOMEONE TRANSCRIBED A NAVAL WEATHER FORECAST WHILE WOODPECKERS HAMMERED THEIR SHIFT KEYS, THEN RANDOMLY INDENTED IT.
IT'S LIKE AN E E CUMMINGS POEM WRITTEN USING ONLY THE USERNAMES A WEBSITE SUGGESTS WHEN THE ONE YOU WANT IS TAKEN.
THIS LOOKS LIKE THE OUTPUT OF A MARKOV BOT THAT'S BEEN FED BUS TIMETABLES FROM A CITY WHERE THE BUSES CRASH CONSTANTLY.
WHATEVER, IT RUNS FINE FOR NOW.
SO DOES A BURNING BUS.

xkcd.com

1. Suppose that a buggy implementation of the MIPS data path miswires the handling of the Read register 1 input and the Read Register 2 inputs to the Registers unit as shown below:



The datpath supports execution of any sequence of `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq`, and `j` instructions.

Everything else in the datapath is implemented as shown on the diagram in the Supplement. All the control signals would be set as discussed in class. For each question below, we will assume that registers and memory locations hold the values shown in the tables above.

The error has several effects:

- the values taken from the register file are swapped (i.e., Read data 1 is now what should have been the right operand, and Read data 2 is now what should have been the left operand); for an instruction that sends both values to the ALU, this means the ALU inputs have been reversed, and that will break any arithmetic operation that is not commutative; for sw, the value from the RS register will be written to memory, instead of the value from the RT field
- for lw and sw, the address will be computed using the RS field, not the RT field
- for R-type instructions and lw, the Write register is still specified correctly

a) **[6 points]** Which, if any, of the supported instructions would always be executed correctly on the buggy datapath shown above? No explanation is needed.

add, and, or	these operations are commutative R-type
beq, j	beq depends on whether the difference is 0, which is unaffected by the reordering; j does not depend on the values in registers; neither writes to a register

b) [10 points] Suppose that the Registers and Data memory initially hold the values shown in the tables below. How would the contents of the Registers and Data memory be changed, if the following instruction was executed? Justify your answer.

Register	value
$t0	0
$t1	1000
$t2	2000
$t3	3000

```
lw    $t2, 3000($t3)
```

The instruction becomes: lw $t3, 3000($t2)

So: $t3 ← Mem[$t2 + 3000]
$t3 ← Mem[5000]
$t3 ← 50

Address	value
0	segfault
1000	10
2000	20
3000	30
4000	40
5000	50
6000	60
7000	70

Data memory is unaffected since MemWrite is 0 for lw.

2. Suppose that when a **beq** instruction is executed the control signals RegDst, MemRead, MemtoReg, MemWrite, RegWrite, ALUOp, and Jump are set to values that would be correct if a **lw** instruction were being executed. And, suppose that the Branch and ALUSrc signals are set properly (for a **beq** instruction). Suppose the following instruction was executed:

```
beq   $t3, $t1, btarg    # btarg is the label for some instruction
```

a) [8 points] Under what conditions would the branch still be taken? Explain.

Branch is set to 1; therefore the branch will be taken iff the ALU sets Zero to 1.

The ALU will compute $t3 + $t1, since ALUOp is set for lw, which performs an add.

Therefore, the branch will be taken iff $t1 + $t3 == 0.

The question is specifically about the conditions under which the branch will be taken, given the described changes in the way the control signals are set.

b) [5 points] Could the execution of the **beq** instruction lead to any unintended changes be made to value(s) in Data memory? If so, describe what could happen. If not, explain why not.

MemWrite is set to 0 for lw, so Data memory is unaffected.

c) [5 points] Could the execution of the **beq** instruction lead to any unintended changes be made to value(s) in Registers? If so, describe what could happen. If not, explain why not.

RegWrite is set to 1 for lw, so a register WILL be written to; MemRead is set to 1 for lw, so data will be read from memory.

The value written is the value in Mem[$t1 + $t3], provided that address can be read from.

Since RegDst is set to 1 for lw, the register written to will be the one lw would write to, which is the RT register, $t1.

So, $t1 ← Mem[$t1 +$t3]

3. The following questions refer to the single-cycle datapath on the Supplement to the test. They refer only to instructions supported by the datapath.

a) **[9 points]** What value should the RegDst signal have for each of the instructions supported in this datapath? Explain your answers in detail.

RegDst	Instructions	Reason
0	lw	lw specifies its destination register in the RT field
1	R-type	R-type instructions specify their destination register in the RD field.
D/C	sw, beq, j	For each of these, RegWrite is set to 0, so it doesn't matter what value is sent to Write register on the Registers unit (and neither choice is "correct").

b) **[9 points]** For every instruction that is executed, the datapath appears to read the contents of Read register 1 and Read register 2, which are sent to the Read data 1 and Read data 2 lines, respectively.

For which instructions would the data read from Read data 2 <u>not be needed?</u> Explain why

lw, and j. j does not depend on register values; lw uses the immediate instead of Read data 2

Every R-type instruction, beq instructions, and sw instructions actually use the values read from BOTH registers.

Pick <u>one</u> of the instructions from your answer above, and explain why the fact that this unnecessary data is read, for that instruction, does not lead to any unintended results. Be complete and precise.

For a lw instruction:

- The value from Read data 2 is sent to the MUX to the right of the Register unit, and goes no further, since ALUSrc is set to 1 for lw

For a j instruction:

- RegWrite and MemWrite are set to 0, so j instructions do not store data anywhere.
- MemRead is set to 0, so no location in Data memory is read
- Jump is set to 1, so the branch target address is not used.

4. Assume an instruction set with five categories of instructions and a hardware implementation of this instruction set. In this implementation the instructions in each category take the number of cycles shown in the table below:

Category	A	B	C	D	E
CPI	2	3	1	6	4

a) **[10 points]** A certain program executes the following mix of instructions from the five categories:

Category	A	B	C	D	E
CPI	20%	10%	30%	?	?

Note that the percentages of instructions in the D and E categories are unknown. What is the minimum CPI and what is the maximum CPI that this hardware implementation can achieve for the program above? Explain your answers.

The percentages for A, B, and C instructions are set. All we can do is consider the effect of different percentages of D and E instructions; there's no reason to assume that the program MUST use any instructions of either type (so 0% is valid), but the percentages still MUST add up to 100.

To minimize the average CPI, we need to avoid type-D instructions, since they take 6 cycles; hence use 0% type-D and 40% type-E:

Min_CPI = 0.20*2 + 0.10*3 + 0.30*1 + 0.40*4 = .4 + .3 + .3 + 1.6 = 2.6

To maximize the average CPI, we need to avoid type-E instructions, since they take 4 cycles; hence use 40% type-D and 0% type-E:

Min_CPI = 0.20*2 + 0.10*3 + 0.30*1 + 0.40*6 = .4 + .3 + .3 + 2.4 = 3.4

b) **[6 points]** A marketing team from the company that sells the hardware described above claims that for a certain AI program their hardware can achieve a throughput of 1.2 instructions per cycle. Do you believe them? Answer with yes or no and explain why.

Suppose the throughput is 1.2 instructions/cycle.

Then the average CPI (cycles/instruction) must be 1/1.2 or 5/6 cycles per instruction.

That is a neat trick, since every instruction takes at least 1 cycle.

There's no suggestion this refers to the program described in part a).

5. Consider the following attributes of possible (hypothetical) designs for a pipeline (not the MIPS pipeline):

A: 300ps clock cycle, 6 stages
B: 400ps clock cycle, 4 stages

Both designs are intended for processors that employ the same machine language design.

There was a LOT of confusion here… as the instructions in part a) say, the latency of an instruction is the time from fetch to completion; in other words, the total time it spends going through the pipeline. This is simply the product of the cycle length and the number of stages.

a) **[4 points]** Calculate the ideal instruction latency (time from fetch to completion) for design **A**.

Latency = 300ps * 6 = 1800ps

b) **[4 points]** Calculate the ideal instruction latency for design **B**.

Latency = 400ps * 4 = 1600ps

Throughput is the rate at which instructions are completed; under ideal conditions (no stalls), a pipeline will complete one instruction per clock cycle. A number of answers were of the form

$$\lim_{n\to\infty}(1500+300n)$$

That, however, is infinite, and does not yield the throughput.

c) **[4 points]** Calculate the ideal throughput for design **A** (assuming an infinite sequence of instructions).

Once the pipeline is full, we complete 1 instruction per 300ps.

d) **[4 points]** Calculate the ideal throughput for design **B** (assuming an infinite sequence of instructions).

Once the pipeline is full, we complete 1 instruction per 400ps.

Performance is dominated by throughput, not latency.

e) **[6 points]** Which design offers better performance? Explain.

A yields lower execution time in the long run. Suppose we execute N instructions, where N > 5 (so both pipelines reach fullness):

Time_A = 1500 + 300N but Time_B = 1200 + 400N, so A is faster if N > 5.

6. [10 points] Suppose that a program executes in 100 seconds, spending 20 seconds on floating-point operations, 40 seconds on integer operations, 25 seconds on branch operations, and 15 seconds on other operations.

Suppose it's possible to use integer operations to simulate floating-point operations, and that integer operations, on average, take half as long to execute as floating-point operations. However, the number of integer operations would be 50% greater than the number of floating-point operations they were replacing.

Use Amdahl's Law to determine how long it would take to execute the program if we replaced all the floating-point operations with integer operations.

Time_after = Time_unaffected + Time_affected / Improvement_factor

= 80 + 20 / (2 * 2/3)

= 80 + 20 / (4/3) = 80 + 15 = 95 seconds

The question DOES specify that you should use Amdahl's Law, so full credit required some indication of that. There were a number of ways to approach the question.