Instructions:

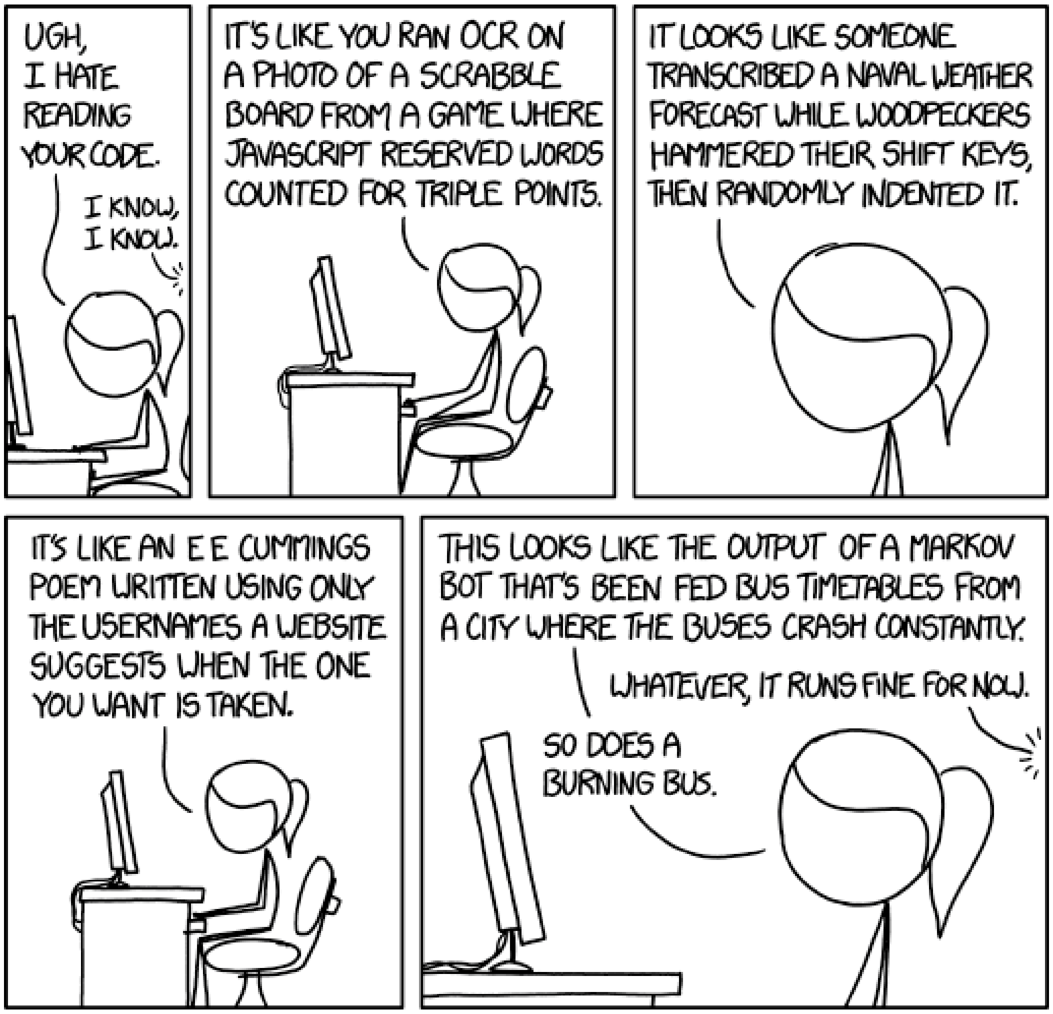
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet.
- No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test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

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. The single-cycle MIPS datapath design used one stage with a clock cycle of 800 ps. The MIPS pipeline had five stages, with a clock cycle of 200 ps. The pipelined version achieved better performance than the single-cycle version by improving instruction throughput. Let's consider some alternative outcomes.

Suppose that the MIPS designers managed (somehow) to produce a feasible pipeline design had resulted in six stages, with a clock cycle of 150 ps.

a) **[6 points]** How does the latency for a single instruction on this new design compare to the latency on the original, single-cycle design, and on the five-state pipeline discussed in class?

New design:	AvgCPI = 6 * 150 = 900 ps
SCD:	AvgCPI = 800 ps
Old Pipeline:	AvgCPI = 5 * 200 = 1000 ps

So, the new design leads to much greater latency than the SCD, but a smaller latency than the original pipeline design.

b) **[6 points]** Assuming an infinite sequence of instructions, what speedup will this design achieve when compared to the original, single-cycle design? Show calculations to support your conclusion.

$$\text{Speedup} = \lim_{n\to\infty} \frac{800n}{750+150n} = \frac{800}{150} = \frac{16}{3} \approx 5.33$$

c) **[6 points]** Assuming an infinite sequence of instructions, what speedup will this design achieve when compared to the five-stage pipeline design discussed in class? Show calculations to support your conclusion.

$$\text{Speedup} = \lim_{n\to\infty} \frac{800+200n}{750+150n} = \frac{200}{150} = \frac{4}{3} \approx 1.33$$

2. Two processors, which support the same ISA, support three classes of instructions, which have the following CPI:

Instruction class	**P1**	**P2**
arithmetic/logical	6	5
load/store	50	60
branch	7	8

a) **[10 points]** The dynamic instruction count for a particular machine language program includes 30% arithmetic/logical instructions, 40% load/store instructions, and 30% branch instructions. In other words, these percentages reflect the executable from a runtime point of view.

For each processor, what would be the average CPI for this executable? Justify your conclusions precisely.

$AvgCPI_1 = 0.30 * 6 + 0.40 * 50 + 0.30 * 7 = 1.8 + 20.0 + 2.1 = 23.9$

$AvgCPI_2 = 0.30 * 5 + 0.40 * 60 + 0.30 * 8 = 1.5 + 24.0 + 2.4 = 27.9$

b) **[6 points]** What do your answers to part a) tell you about the relative performance of P1 and P2, when executing this particular program? Explain.

This tells me absolutely nothing about the relative performance of the program on the two processors. Performance is defined by execution time, and to compute that we would need to know

- the number of instructions to be executed (the actual dynamic count)
- the cycle length for the clock on each processor

3. Pseudo-instructions give MIPS a richer set of assembly language instructions than those supported directly by the hardware. Write an implementation for each of the following pseudo-instructions, using only the MIPS32 instructions listed at the bottom of this page (those are sufficient). You must include comments explaining the logic of your solutions. If you need to use any "extra" registers, you may only use the register `$at`.

a) [8 points] `addfm  $rd, ($rs), ($rt)`

```
    # GPR[rd] = Mem[GPR[rs]] + Mem[GPR[rt]]
```

```
lw   $rd, ($rs)      # load Mem[GPR[rs]] into rd
lw   $at, ($rt)      # load Mem[GPR[rt]] into at
add  $rd, $rd, $at   # add Mem[GPR[rt]] to rd
```

b) [10 points] `settosmaller  $rd, $rs, $rt`

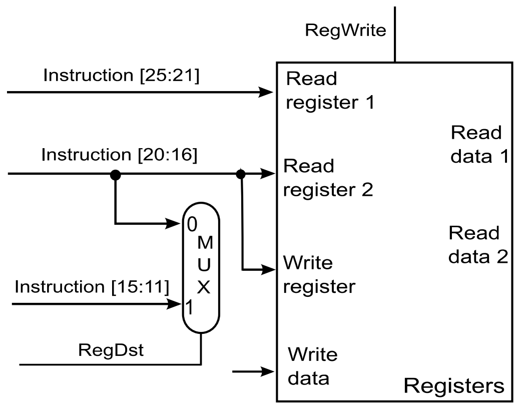
```
    # GPR[rd] = mimimum of GPR[rs] and GPR[rt]
```

```
        slt   $at, $rs, $rt        # at = (rs < rt)?
        beq   $at, $zero, else     # rt is smaller
        addi  $rd, $rs, $zero      # copy rs into rd
        beq   $zero, $zero, done   # finished
else:
        addi  $rd, $rt, $zero      # copy rt into rd
done:
```

```
add   rd, rs, rt       # GPR[rd] <-- GPR[rs] + GPR[rt]
addi  rd, rs, imm16    # GPR[rd] <-- GPR[rs] + imm16     (sign extended)
and   rd, rs, rt       # GPR[rd] <-- GPR[rs] AND GPR[rt] (bitwise)
beq   rs, rt, label    # if GPR[rs] == GPR[rt], jump to label
bne   rs, rt, label    # if GPR[rs] != GPR[rt], jump to label
lw    rt, imm16(rs)    # GPR[rt] = Mem[GPR[rt] + imm16]
or    rd, rs, rt       # GPR[rd] <-- GPR[rs] OR GPR[rt]  (bitwise)
sub   rd, rs, rt       # GPR[rd] <-- GPR[rs] - GPR[rt]
sll   rd, rt, sa       # GPR[rd] <-- GPR[rs] << sa       (logical shift)
slt   rd, rs, rt       # GPR[rd] <-- (GPR[rs] < GPR[rt] ? 1 : 0)
```

(**imm16** denotes a 16-bit immediate value)

4. Suppose that a buggy implementation of the MIPS datapath miswires the selection of the Write register number:



Everything else in the datapath operates normally.

a) **[8 points]** Describe, in detail, how this error would affect the execution of the following instruction:

```
sw   $t0, ($t2)
```

Since sw does not write a value to any register, the error has no effect whatsoever.

b) **[8 points]** Describe, in detail, how this error would affect the execution of the following instruction:

```
sub  $s0, $s1, $s2
```

The difference of $s1 and $s2 is computed correctly, but it is stored to $s1, not $s0.

5. **[16 points]** Suppose the following sequence of instructions was executed on the preliminary pipeline design derived in class, without any hardware to detect or handle hazards, as shown on the supplement:

```
sub    $t5, $t1, $t4    # 1

slt    $s7, $t1, $t4    # 2

sw     $t5, 0($t4)      # 3

add    $t4, $t5, $s7    # 4

or     $t1, $t4, $t5    # 5

lw     $t4, 0($t1)      # 6
```

Due to the limitations of the preliminary design, a number of different logical errors will occur. These may include possibly incorrect values being used as input to an instruction, possibly incorrect values being generated as output from an instruction, and other kinds of errors.

For each logical error that may occur, clearly identify the relevant instruction whose execution might be incorrect, and identify the register(s), if any, that are involved in the error, and describe the nature of the error.

```
Instruction #    Description of error
-----------------------------------------------------------------
```

Instruction #	Description of error
#1	no issues
#2	no issues
#3	will read a stale value from $t5 will write incorrect value to Mem[GPR[$t4]]
#4	will read a stale value from $s7 will compute incorrect value for $t4
#5	will read a stale value from $t4 will compute incorrect value for $t1
#6	will read a stale value from $t1 will read a value for $t4 from incorrect address

6. [16 points] Consider the following proposed instruction for the simplified single-cycle MIPS32 datapath discussed in class:

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

sas	rs	rt	rd	ignored	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 that address in register `$rd`.

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 (and that there is a currently-unused opcode that we can use for the new instruction).

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.)

Signal	Value
RegDst	1
Jump	0
Branch	0
MemRead	1
MemtoReg	1
MemWrite	0
ALUSrc	0
RegWrite	1