Instructions:

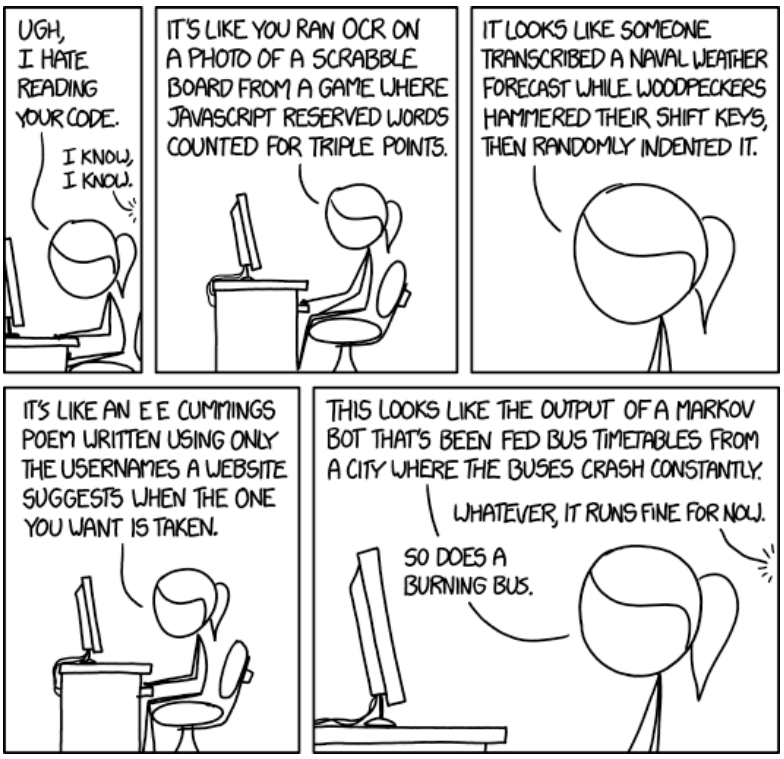
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet.
- No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test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

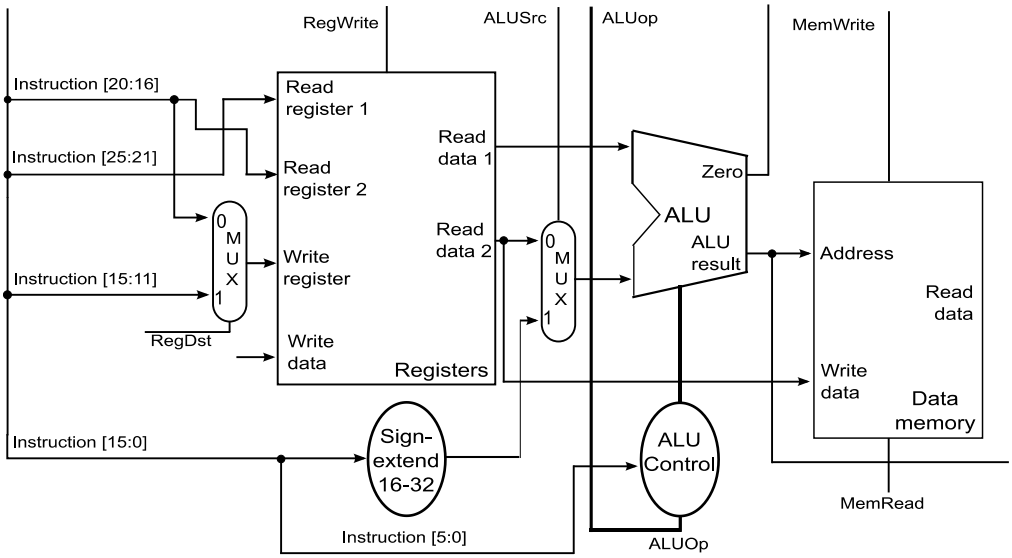
_____
signed


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

For questions 1 and 2:

Suppose that a buggy implementation of the single-cycle MIPS32 data path miswires the handling of bits [20:16] and bits [25:21] as shown below:



The datapath 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 single-cycle datapath 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 below.

1. **[10 points]** Which, if any, of the supported instructions would <u>always</u> be executed correctly on the buggy datapath shown above? No explanation is needed.

The first "bug" was that the read register number fields are reversed when pulled from the instruction bits; but the second "bug" is that the wiring then swaps them back. In addition, the correct bits [20:16] and [15:11] are sent to the MUX controlled by RegDst, so the result is that this will perform as the original design intended.

The unaffected instructions would be:

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

Note that, if the read registers were actually swapped:

- **Commutative R-type instructions would be unaffected (add, and, or)**
- **beq would be unaffected since it only cares if the registers store equal values**
- **j would still be unaffected, since it doesn't care about any registers at all**
- **lw and sw would be affected, since their address calculation would use the wrong register**

So, if the registers were actually swapped, the only unaffected instructions would be:

add, and, or, beq, j

2. **[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.

```
lw   $t0, 2000($t1)
```

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

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

The actual error means that:

$t1 is used for Read register 1, and goes to the ALU
$t0 is used for Read register 2 and also for the write-to register

The address is computed as: $t1 + 2000 == 3000, and so:

$t0 = Mem[3000] = 30

OTOH, if the read registers had been swapped (so this is incorrect but a common error):

$t0 would be used for Read register 1, and for the write-to register, and go to the ALU
$t1 would be used for Read register 2 and go to the ALU

The address would be computed as: $t0 + 2000 == 2000

$t0 = Mem[2000] = 20

There were many strange answers that seemed to confuse register values with memory addresses. The value in a register could be an address, but there's nothing in this question that suggests that.

For questions 3 and 4 we refer to the single-cycle datapath as shown on the Supplement:

Suppose that every control signal is set to the value that would be correct if a **beq** instruction were being executed, but that the following instruction was actually placed into the datapath:

```
add  $s7, $t1, $t5
```

3. **[10 points]** List every action that should occur when the **add** instruction is executed, but that will not be executed correctly in this case. Be specific in describing each action.

The following actions, are necessary for executing add correctly:

- **Reading the correct two registers**
- **Sending both those values to the ALU; i.e., ALUsrc == 0**
- **The ALU must perform an ADD; so ALUop must indicate an ADD**
- **The computed sum is sent to the Register unit; so MemtoReg == 0**
- **That value is stored in $s7; so RegWrite == 1 and RegDst == 0**
- **MemRead (arguably) and MemWrite must be set to 0**
- **Branch and Jump must be set to 0**

The following actions, necessary for executing ADD correctly, would not be done correctly:

- **The ALU will perform a subtraction operation, since BEQ requires that.**
- **No value will be written to $s7, since RegWrite is 0 for a BEQ instruction.**
- **Branch will be set to 1.**
- **An unknown value will be sent to the Register unit, since MemtoReg is a D/C for BEQ.**
- **An $s7 may or may not be chosen, since RegDst is a D/C for BEQ.**

4. **[10 points]** Could a branch be performed when this **add** instruction is executed? Explain precisely.

Yes, if the subtraction in the ALU yields 0, since Branch will be set to 1.

5. [10 points] Consider the pipeline design with interstage buffers shown on the Supplement. Suppose that the following instructions are currently in the indicated stages of the pipeline:

```
                                # stage
sw    $s4, 24($t1)              #     4
lw    $t2,  8($t5)              #     3
add   $t4, $t7, $t3             #     2
```

Explain what could go wrong when the **add** instruction is decoded (in stage 2) if our pipeline design **did not** use the interstage buffers to manage the control signals.

During the decoding of ADD, the following control signals will be set:

Signal	Value	Effect
RegDst	1	Write-to register will be [15:11]
Branch	0	No branch will occur
MemRead	0	No memory read will occur
MemWrite	0	A memory write will be attempted
MemtoReg	0	ALU result is sent to the Register unit
ALUop	signal an add	ALU will add
ALUSrc	0	ALU will use the second register instead of the immediate
RegWrite	1	A register write will occur

AND, those control signal values would immediately be sent to the following pipeline stages. So, the question is: what unfortunate effects would this cause, for instructions that are further along in the pipeline, during this clock cycle?

For LW in the EX stage:

- **ALU will use the wrong operand when computing the address to read from**

For SW in the MEM stage:

- **No write to data memory will be attempted.**

The fact that the register write will be affected isn't relevant, since neither LW nor SW will attempt a register write from their current pipeline stages. Of course, this could very well have an effect on the unknown instruction, if any, that's currently in stage 5.

6. **[10 points]** Suppose the following instructions are in the synchronized pipelined MIPS32 datapath, as shown on the Supplement:

```
                                # stage
add   $t3, $t1, $t2             #     5
and   $s4, $s1, $s2             #     4
or    $t2, $t1, $t2             #     3
sub   $t4, $s4, $t1             #     2
```

Now, we have a dependency between the **and** and the **sub** instructions, because **sub** needs the value that **and** writes into register `$s4`. Will the **sub** instruction receive the correct value from `$s4`? Explain.

No. The AND instruction will not write to $s4 until AND reaches stage 5. Therefore, the SUB instruction will read a "stale" value from $s4 while SUB is in stage 2.

For questions 7 and 8:

Suppose there is a pipeline design (not the MIPS32 pipeline) that uses 6 stages and has an instruction latency of 900ps. That is, every instruction takes 900ps to go through the pipeline. The design also avoids the need for any stalls.

7. **[10 points]** What must be the clock cycle length for this design? Explain.

Each stage will take exactly one clock cycle to complete its actions, so the cycle length must equal the instruction latency divided by the number of stages:

150ps

There were many strange answers here, like multiplying the latency times the number of stages. As the questions states, the instruction latency is how long an instruction takes to go through the pipeline, not how long an instruction takes to complete one stage.

8. **[10 points]** A new pipeline design is being proposed. That pipeline would have 4 stages and an instruction latency of 800ps. Could this new design offer better performance than the original design described above? Explain.

Performance is determined by instruction throughput; that is determined by the cycle length, but a correct answer must address throughput. The number of stages and the instruction latency are irrelevant..

For the new design, the cycle length would be 200ps.

For the original pipeline, throughput would (ideally) be 1 instruction every 150ps.
For the new design, the throughput would (ideally) be 1 instruction every 200ps.

So, the new design offers worse performance.

For questions 9 and 10:

We are interested in the performance of two computers, the Primus and the Secundus, both of which execute the same machine code.

9. **[10 points]** A particular benchmark requires executing $8x10^{11}$ machine instructions (on each computer). Analysis of the instructions that would be executed reveals that the average CPI is 5.6 for the Primus, and 4.9 for the Secundus. From the given information, what can you conclude about the relative performance of the benchmark on the two computers? Explain.

To make a comparison, we need to compute the program execution time for each machine.

To compute the program execution time, we would need to know the cycle length (or the clock frequency).

We do not, so we cannot reach any conclusion about the relative performance.

That's really all that is relevant… you are not given enough information to determine the execution times, and therefore cannot reach any conclusions about the relative performance.

10. [10 points] Another benchmark takes 100 seconds to execute on the Primus. A proposed hardware redesign will improve the execution time for memory access instructions on the Primus by 20% (e.g., by a factor of 1.2). The claim is made that this improvement will reduce the execution time of the benchmark on the Primus by 20%. If that is true, how much time must the benchmark time have spent on memory access instructions, running on the original hardware? Justify your conclusion.

Let X be the time in seconds spent on memory accesses with the original hardware. Then Amdahl's Law implies that, with a 20% improvement:

Taking the clumsy phrasing in the question literally:

80 = (100 – X) + X/1.2

X – X / 1.2 = 100 – 80 = 20

1.2X – X = 20 * 1.2

.2X = 24

X = 24 / .2 = 120s, which is impossible

Taking the clumsy phrasing in the question correctly:

80 = (100 – X) + X/1.25

X – X / 1.25 = 100 – 80 = 20

1.25X – X = 20 * 1.25

.25X = 25

X = 25 / .25 = 100s, which is unlikely but perhaps not impossible

There were a number of different ways to set this up, that led to slightly different answers, depending on how you interpreted the phrasing of the question.

It was also possible to realize that the only way that improving one aspect by 20% could also improve overall performance by the same amount would be if that one aspect (memory accesses in this case) is the only thing the program does…