Instructions:

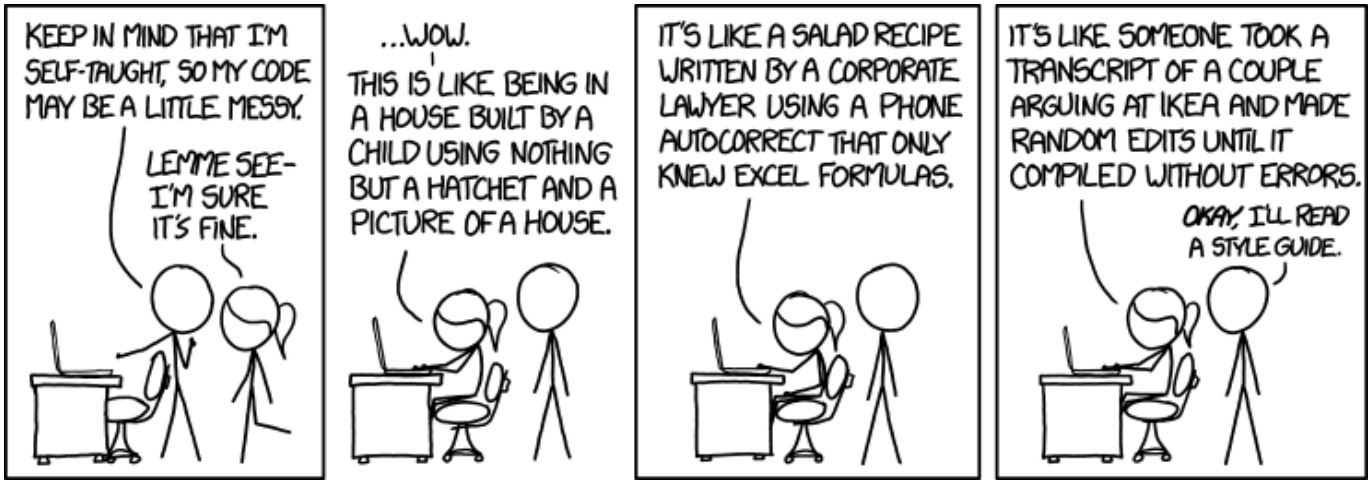
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet.
- No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test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

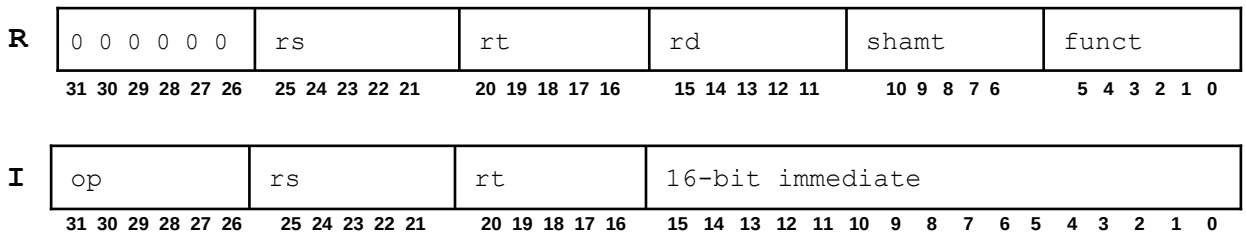
Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

signed


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE- I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

xkcd.com

1. Recall the formats for the R-type and I-type MIPS machine instructions:



a) [8 points] The design of these MIPS machine instructions makes the hardware for handling register reads simpler than the hardware for handling register writes. Explain why. Be specific.

Every instruction that reads from one or two registers stores the relevant register numbers in bits 25:21 and 20:16.

An instruction that writes to a register may store the relevant register number either in bits 20:16 or bits 15:11. That requires extra hardware in order to choose where to take the register number from.

There's no issue of the speed with which a register read/write will occur; the MUX needed to make the choice of the write register number is going to affect both reads and writes, and won't have significant additional latency when compared to the register file itself.

The relative proportion of instructions that only read registers vs those that both read and write registers is also irrelevant.

b) [8 points] In the single-cycle datapath (and in the pipelined version), the main Control unit sets the control signals RegDst, Jump, Branch, MemRead, MemWrite, MemtoReg, ALUop, ALUSrc and RegWrite, even if the instruction is R-type. How can that work correctly, since the main Control unit never knows which specific R-type instruction is being executed?

Each of those control signals needs to be set to exactly the same value for every R-type instruction.

Since the main Control unit receives the opcode bits, and those are 000000 for every R-type instruction, the main Control unit can safely set all of those signals without knowing precisely which R-type instruction is present.

The question focused on the fact that the main Control unit is setting all those signals w/o knowing exactly which R-type instruction is being executed. The relevant points are the ones that are made in my answer. Other things, like the operation of a separate ALU Control unit were not relevant.

2. For the following questions, refer to the single-cycle datapath diagram included at the end of the test. Both questions below refer only to the instructions that are supported by that datapath.

a) [8 points] The MUX labeled 2a is necessary due to the existence of one specific machine instruction. What is that instruction, and what attribute of that instruction makes this MUX necessary?

lw

The lw instruction writes to a register, and stores the number of that register in bits 20:16. Every other instruction that writes to a register (i.e., the R-types ones) stores the number of the register to be written in bits 15:11.

b) [8 points] No matter what instruction is being executed, a value will be read and sent to the Read data 2 port, labeled 2b, on the register file. For which instructions is that value not needed? Explain why, if one of those instructions is being executed, the presence of that value on Read data 2 does not lead to any logical errors when executing the instruction. Be complete.

The value is not needed for either lw or j.

When lw is executed, the value from Read data 2 has no effect because:

- **the MUX to which it goes passes its other input to the ALU**
- **the data memory will not write that value since MemWrite is 0 for lw**

When j is executed, the value from Read data 2 has no effect because:

- **the value from the ALU, which may be affected by Read data 2, is not used as an address for data memory (MemRead and MemWrite will be 0 for j)**
- **the data memory will not write that value since MemWrite is 0 for j**

All the other instructions, beq and R-type, do make use of the value from Read data 2.

3. The Shift left 2 unit, labeled 3, appends two zero bits to the immediate value taken from a `beq` instruction.

a) [7 points] The action of this Shift left 2 unit actually restores two bits that were removed from something when the machine instruction was formed. Why were the two bits that were removed guaranteed to be zeros? Be precise.

The immediate field of beq is based on the difference between the addresses of two machine instructions.

Since the address of a machine instruction is always a multiple of 4, the difference of two such addresses is a multiple of 4.

In 2's complement representation, multiples of 4 always have their two low bits equaling 0.

It helps to read the question. The bits that were removed were the low bits of the difference between the addresses of two MIPS machine instructions; that's how a beq machine instruction is formed.

There were a number of strange assertions, like "machine instructions are always multiples of 4". No. The address of a machine instruction is always a multiple of 4, and that's not the same thing at all.

b) [7 points] What was gained by removing those two bits when forming the machine instruction, and then restoring them in the hardware? Be precise.

Effectively, we have an 18-bit value for the branch distance, which increases the range of a branch instruction.

The point is that the "trick" that's used here lets us effectively have a larger immediate value, and therefore allows the beq to branch to an instruction that's farther away. There are no other relevant points.

4. Assume, as in the lectures, that the single-cycle design would require a clock cycle of 800 ps and the 5-stage pipelined design would require a clock cycle of 200 ps.

a) [7 points] Each instruction would spend 1000 ps going through the pipeline, which is longer than it would have spent going through the single-cycle datapath. Why does the pipeline increase the latency for an instruction? (Saying that "5 times 200 is greater than 800" does not address the question.)

Each stage in the pipeline will be allotted one 200 ps clock cycle to stabilize.

That is more time than is needed for the ID and WB stages, since register reads/writes take only 100 ps.

So, the pipeline effectively "loses" 100 ps in each of those stages.

There were a number of lesser and irrelevant points that were raised in answers to this:

- **latency due to interstage buffers; tiny in comparison to the point made above**
- **more complex hardware in the pipeline**
- **data hazards in the pipeline; these induce no delay in the pipeline, except for the load-use hazards, which cost us 200 ps (when they occur)**
- **stalls in the pipeline; a pipeline stall costs us 200 ps; but, in effect the single-cycle design will have an even longer stall between completions of successive instructions**

b) [7 points] Despite the increased latency, we expect the pipeline to execute a long sequence of instructions in less time than the single-cycle design. Explain why. Be thorough.

Both the single-cycle and pipeline designs will complete one instruction every clock cycle.

But the clock cycle is much shorter for the pipeline, so we have increased the throughput from one instruction per 800 ps to 1 instruction per 200 ps, for a speedup of 4 (if things go perfectly in the pipeline).

A lot of answers made vague statements about the fact that the pipeline would be executing 5 instructions at once vs 1 at a time for the SCD. That's true, but it doesn't explain why the pipeline actually achieves better performance. Mentioning "throughput" was good, but you needed to explain WHY the pipeline improves throughput (which is the real point).

5. [10 points] Suppose a program consists of 40% floating point multiply instructions, 20% floating point divide instructions, and the remaining 40% are other instructions. Floating point division, floating point multiplication, and each of the other instructions have the same CPI.

What speedup would be achieved, for this program, if floating point multiplication were made 2 times faster? Apply Amdahl's Law and justify your conclusion precisely.

Suppose the program requires executing K machine instructions, with a cycle time of C. Then:

$T_{before} = KC$

$T_{after} = T_{unaffected} + T_{affected}/improvement = 0.60KC + 0.40KC/2 = 0.80KC$

$Speedup = T_{before} / T_{after} = KC / 0.80KC = 1 / 0.80 = 5/4 = 1.25$

The definition of speedup is given in the notes.

6. A particular instruction set has five categories of instructions. For a hardware implementation, instructions in each category take the number of cycles shown below:

Category	A	B	C	D	E
CPI	1	3	2	5	4

A certain machine code program contains the following proportions of instructions from the five categories:

Category	A	B	C	D	E
Proportion	20%	30%	25%	20%	5%

a) [7 points] Calculate the average CPI for this program.

Avg CPI = 0.20 * 1 + 0.30 * 3 + 0.25 * 2 + 0.20 * 5 + 0.05 * 4
= 0.20 + 0.90 + 0.50 + 1.0 + 0.20
= 2.8

Examples of computing average CPI are in the notes.

b) [7 points] What is the minimum amount of additional information that would be necessary in order to determine the execution time for this program?

You must also know both of these:

- the total number of instructions that will be executed (or the number in each category, or the total number of cycles the program will take)
- the length of a clock cycle

7. Consider the following sequence of MIPS32 assembly instructions:

```
and   $t1, $t2, $t3    # 1
lw    $t4, 0($t1)      # 2
lw    $t5, 0($t2)      # 3
sub   $t1, $t2, $t4    # 4
sw    $t5, 0($t0)      # 5
```

A *data dependency* occurs when a later instruction requires an input value that is set by an earlier instruction. A *data hazard* occurs when one instruction writes a value into a register that will be used as input by a later instruction, but that value does not actually appear in the register by the cycle on which the later instruction attempts to read it. Note that a data hazard always implies a data dependency, but some data dependencies do not imply a data hazard.

You might want to refer to the preliminary version of the pipeline design shown at the end of this test.

a) [6 points] Identify the data hazards that would prevent the given sequence of instructions from executing correctly on the given hardware design above, unless we inserted one or more `nop` instructions. For each such dependency, list the register involved, the writing instruction and the reading instruction (identify the instructions by number).

register	writer	reader
$t1	#1	#2
$t4	#2	#4
$t5	#3	#5

b) [6 points] Rewrite the given sequence of instructions adding `nop` instructions so that the modified sequence would execute correctly on the given hardware design (which does not have any way to handle data dependencies built into it). Do not change the order of the given instructions. For full credit, accomplish this with the smallest possible number of `nop` instructions.

In order to avoid a read-after-write hazard, the reader must be at least 3 stages (cycles) behind the writer. We could insert nop instructions as follows:

```
and   $t1, $t2, $t3    # 1
nop
nop
lw    $t4, 0($t1)      # 2   This must be at least 3 cycles behind #1 for $t1.
lw    $t5, 0($t2)      # 3
nop
sub   $t1, $t2, $t4    # 4   This must be at least 3 cycles behind #2 for $t4.
sw    $t5, 0($t0)      # 5   This must be at least 3 cycles behind #2 for $t5.
```

c) [4 points] How many clock cycles would be required to execute the modified sequence of instructions you gave in the previous part of this question?

We now have 8 instructions; the last instruction will enter the pipeline on clock cycle 8, and require 4 more cycles to exit the pipeline. So, the entire sequence will take 12 cycles.

The answer here equals the original number of instructions plus the number of nops, plus 4.