Instructions:

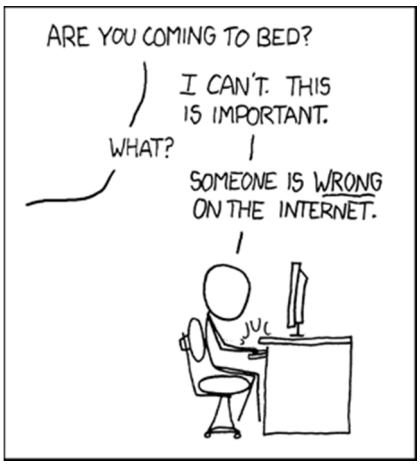
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test 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 8 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 and turn in the test.
- If you brought a fact sheet to the test, write your name on it and turn it in with 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


ARE YOU COMING TO BED?
I CAN'T. THIS IS IMPORTANT.
WHAT?
SOMEONE IS WRONG ON THE INTERNET.

xkcd.com

1. This question relates to the execution of MIPS32 instructions on a pipelined datapath, without instruction reordering.

a) [6 points] Give a specific example of a sequence of MIPS32 assembly instructions that include a data dependency (hazard) that cannot be resolved by forwarding alone.

```
lw   $t0, 0($t1)     // must be lw
add  $t2, $t0, $t1   // must read from register lw writes to
```

Every read-after-write hazard where the writer is R-type can be handled by forwarding.

b) [6 points] Explain precisely why forwarding alone cannot resolve the hazard in the example you stated above. Be precise and refer to the pipeline explicitly in your answer. Feel free to consult the relevant pipeline diagram included with the test.

The value `lw` writes to its destination register is not available for forwarding until the `lw` instruction reaches the end of the MEM stage.

By then, the `add` instruction has reached the end of the EX stage, and so forwarding is no longer possible.

2. [10 points] Suppose that an R-type MIPS32 instruction writes to a register that is read by a subsequent MIPS32 instruction. The instructions are executed on the MIPS32 pipeline. What is the minimum number of cycles the second instruction must be behind the first instruction, in order that the dependency between them can be resolved without using forwarding or instruction reordering? Justify your answer carefully.

No matter what the writing instruction is, it will not complete writing a value into its destination register until it reaches the first half of the WB stage.

That means the reading instruction (which reads the register during the second half of the ID stage), cannot enter the ID stage before writing instruction enters the WB stage.

So, the reading instruction must be at least three cycles behind the writing instruction.

3. This question relates to resolving MIPS32 data hazards by using instruction reordering.

a) [8 points] Identify all the data hazards in the following sequence of instructions. For each hazard, state the register involved, the writing instruction (by number) and the reading instruction (by number).

```
add  $t0, $s0, $s1
xor  $t1, $t0, $s2
lw   $s0, -12($a0)
sub  $s5, $s0, $s1
```

register	writer	reader
$t0	add	xor
$s0	lw	sub

b) [8 points] Is it possible to resolve any of the hazards you identified in the previous part by reordering the instructions so that forwarding would be unnecessary? If yes, show how. If not, explain why not.

There are two reorderings. This one eliminates the need to forward from `add` to `xor`:

```
add  $t0, $s0, $s1
lw   $s0, -12($a0)
sub  $s5, $s0, $s1
xor  $t1, $t0, $s2
```

This one eliminates the need to forward from `lw` to `sub`:

```
lw   $s0, -12($a0)
add  $t0, $s0, $s1
xor  $t1, $t0, $s2
sub  $s5, $s0, $s1
```

To eliminate forwarding, the reading instruction must be at least 3 cycles behind the writing instruction.

4. This question relates to resolving MIPS32 data hazards by using instruction reordering and stalls.

a) [8 points] Identify all the data hazards in the following sequence of instructions. For each hazard, state the register involved, the writing instruction (by number) and the reading instruction (by number).

```
lw    $t0, 0($s5)
addi  $s5, $s5, 4
add   $t0, $s3, $t0
sw    $t0, 0($s5)
xor   $t3, $t2, $s5
add   $t2, $t3, $t2
```

register	writer	reader
$t0	lw	add
$s5	addi	sw
$t0	add	sw
$t3	xor	add

b) [8 points] Is it possible to resolve (with no forwarding) any of the hazards you identified in the previous part by reordering the instructions and/or adding `nop` instructions? If yes, show an efficient way to do so. If not, explain why not.

Again, the reading instruction must be at least 3 cycles behind the writing instruction in order to avoid forwarding. You must also be careful to not reorder instructions in such a way that violates the dependencies between those instructions.

Here is an ordering that eliminates all forwarding, with the insertion of a two `nop` instructions:

```
lw    $t0, 0($s5)
addi  $s5, $s5, 4
nop
add   $t0, $s3, $t0
xor   $t3, $t2, $s5
nop
sw    $t0, 0($s5)
add   $t2, $t3, $t2
```

5. Refer to the diagram of the detailed MIPS32 pipeline without forwarding or hazard detection.

a) [6 points] Consider the control signal MemWrite, which is set in the decode stage of the pipeline. Why is that signal passed forward via the ID/EX and EX/MEM interstage buffers? Be precise.

The purpose of the interstage buffers is to maintain synchronization between instructions and the data values and control signals as instructions progress through the pipeline stages.

The MemWrite signal is set when an instruction reaches the ID stage of the pipeline, but that instruction will not need that signal until the instruction reaches the MEM stage two cycles later.

Passing the MemWrite signal via the interstage buffers guarantees the correct value for MemWrite reaches the MEM stage on the same clock cycle as the instruction it belongs with.

If we did not do this, the MemWrite signal could reach the data memory unit when a different instruction is in the MEM stage, and that would likely yield incorrect results by storing a value in memory, at who knows what address.

b) [6 points] A hardware engineer proposes moving the ALU Control unit from the execute stage to the decode stage (i.e., one stage earlier). Aside from rerouting the inputs to the ALU Control unit, no other physical changes would be made to the pipeline. If the proposed change were implemented, what effect(s) would you anticipate this to have on pipeline performance, and why? Explain carefully.

Every circuit component has a *latency*, the length of time it takes for the circuit to correctly update its outputs after its inputs have changed.

Moving the ALU Control unit from EX to ID could alter the latency for each stage.

Without more information, it is impossible to say whether this would increase or decrease the latency of either stage.

The stage with the greatest latency determines the minimum clock cycle, and that determines the latency of the pipeline itself.

If the change increased the latency of the stage that already had the greatest latency, that would require increasing the clock cycle, and that would decrease performance.

If the change decreased the latency of the stage that already had the greatest latency, that would allow us to decrease the clock cycle, and that would improve performance.

6. Refer to the simplified diagram for the MIPS32 pipeline, with forwarding and hazard detection.

a) [6 points] What purpose do the multiplexors to the left of the ALU serve? Why are they necessary for this datapath?

They support the ability to select what values are passed as operands to the ALU.

b) [4 points] Why are there multiplexors for both inputs to the ALU?

Forwarding could target either, or both, operands to the ALU.

c) [4 points] Why do those multiplexors take three data inputs?

There are three possible sources for each operand:

- **a value just read from the register file (no forwarding)**
- **a value forwarded from the EX/MEM interstage buffer, from the instruction 1 cycle ahead**
- **a value forwarded from the MEM/WB interstage buffer, from the instruction 2 cycles ahead**

7. [10 points] A direct-mapped cache unit for a machine that uses 32-bit addresses is designed to have 512 lines, each storing 32 words of data.

Explain carefully how the bits of an address would be used in resolving cache references. Be complete.

There are 2^9 cache lines, each holding 128 (2^7) bytes of data.

So, we need 7 bits to specify an offset within the line, 9 bits to specify the cache line, and the remaining 16 bits serve as the tag for the cache line.

The bits are parceled out as follows:

8. [10 points] A designer is planning a cache that will store a fixed number of bytes of data. The designer is more concerned about servicing spatial locality than temporal locality. Should she choose to have fewer lines each holding more data, or more lines each holding less data? Justify your conclusion carefully.

If a process exhibits spatial locality, we expect that if data at some address is currently accessed, then data at nearby addresses are likely to be accessed in the near future. The canonical example is the traversal of a row of a matrix (since matrices are stored row-by-row in C).

This argues in favor of having fewer cache lines, each storing more data:

- **Once data is loaded into a line, there is a good chance that more (perhaps most) of that data will be accessed soon.**
- **The primary cost of a cache miss is the time to fetch the required data from memory and place it into the cache; if most of the data in a line will be accessed, once that data is loaded, we will experience greater latency between cache misses.**