Instructions:

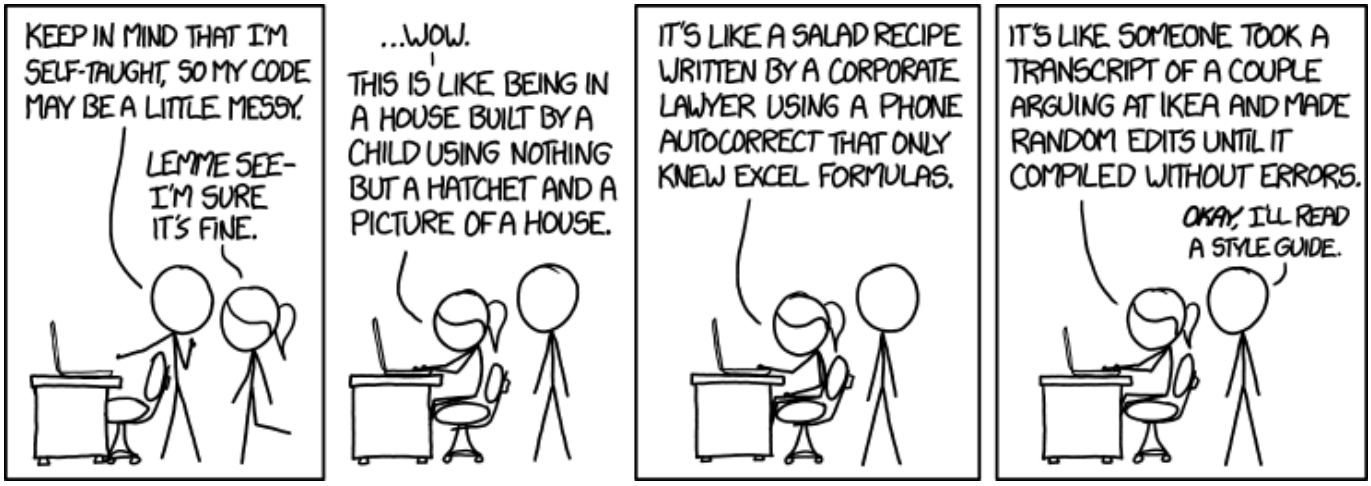
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted fact sheet, with a restriction: 1) one 8.5x11 sheet, both sides, handwritten or typed, and 2) no questions/answers taken from any old HW or tests. If you brought a fact sheet to the test, write your name on it and turn it in with the test.
- 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.
- 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. A key step in designing the MIPS pipeline was to determine how to restructure the components that were in the single-cycle datapath into a sequence of stages. The MIPS designers settled on a 5-stage design, but that was not logically inevitable.

a) [6 points] Identify and explain one significant performance advantage they might have gained if they had chosen a design that used more than 5 stages. Be precise.

If more stages were used, then at least some of the stages would have contained less hardware. That might have resulted in a reduction of the clock cycle length, and that would have improved instruction throughput, if the pipeline was performing optimally.

b) [6 points] Identify and explain one significant performance advantage they might have gained if they had chosen a design that used fewer than 5 stages. Be precise. Hint: consider the set of instructions to be supported.

With fewer stages, there might have been a reduction in the branch miss penalty, since there might have been fewer stages between IF and the point the branch decision was made.

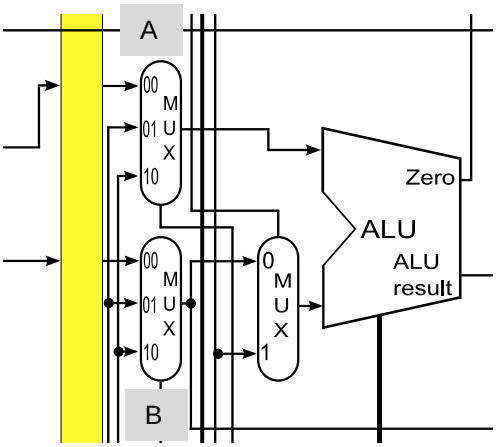
It's worth noting that an almost certain effect of reducing the number of stages would be to increase the cycle length, which would hurt throughput. That might, in theory, be compensated for if the change also reduced the need to insert stalls in the pipeline.

You should also note that this will not eliminate the need to stall on a mispredicted branch, unless the branch decision could be made in a combined IF/ID stage. Combining those two stages would be likely to cause a large increase in the cycle length, so it does not seem to be a likely choice.

Much the same can be said about the prospect of eliminating stalls for load-use hazards by combining EX and MEM into a single stage.

2. Consider executing the following sequence of instructions on the MIPS pipeline with support for forwarding and hazard detection. You may assume that any instructions that preceded these are irrelevant.

```
sub   $s0, $s3, $s2   # 1
add   $t0, $t1, $t2   # 2
add   $t1, $t0, $t2   # 3
add   $t1, $t0, $t1   # 4
add   $t4, $t1, $t0   # 5
```



The following questions concern the two MUXes, labelled A and B above, used to determine what operands are passed to the ALU. You might want to consult the full datapath diagram as well.

a) [4 points] How should the control signals be set for the each MUX when instruction `#3` enters the EX stage?

A: 10 (forward $t0 from the EX/MEM buffer for the 1st operand)
B: 00 (no forwarding is needed for the 2nd operand)

This follows from determining what forwarding operations are needed for #3 and examining the full datapath diagram to be sure of where the three inputs to the MUXes are coming from.

b) [4 points] How should the control signals be set for the each MUX when instruction `#4` enters the EX stage?

A: 01 (forward $t0 from the MEM/WB buffer for the 1st operand)
B: 10 (forward $t1 from the EX/MEM buffer for the 2nd operand)

c) [4 points] How should the control signals be set for the each MUX when instruction `#5` enters the EX stage?

A: 10 (forward $t1 from the EX/MEM buffer for the 1st operand)
B: 00 (no forwarding is needed for the 2nd operand)

3. [8 points] Suppose we decide it is too expensive to use 3-input MUXes, as shown in the diagram for question 2, and that we must use 2-input MUXes instead. As a result, we will choose between the following two options:

i) forward, when necessary, from the MEM/WB interstage buffer, and stall as needed to resolve all data hazards involving a value from the EX/MEM interstage buffer
ii) forward, when necessary, from the EX/MEM interstage buffer, and stall as needed to resolve all data hazards involving a value from the MEM/WB interstage buffer

Assume that data hazards are equally likely to be related the EX/MEM and MEM/WB interstage buffers. Evaluate option i) and option ii) and decide which you think is preferable. Justify your conclusion carefully.

With forwarding only from the EX/MEM interstage buffer, EX to 1st dependencies can be satisfied without stalls but any other dependencies (even when together with EX to 1st) incur a one-cycle stall.

With forwarding only from the MEM/WB interstage buffer, EX and MEM to 2nd dependencies incur no stalls. MEM to 1st dependencies still incur a one-cycle stall, and EX to 1st dependencies now incur one stall cycle because we must wait for the instruction to complete the MEM stage to be able to forward to the next instruction.

So, the potential for avoiding stalls seems to be higher when we have only forwarding from the MEM/WB interstage buffer.

---

4. Suppose that, instead of `beq`, we have `bez`, defined as follows:

```
bez   $rs, label     # take branch if rs == 0
                     # PC <-- (rs == 0 ? PC + 4 + offset <<1 2
                     #                 : PC + 4)
```

a) [8 points] Rewrite the following sequence of MIPS instructions so that it uses `bez` instead of `beq` to achieve exactly the same logical results; you may assume register `$t9` is available if you need temporary storage.

```
        j     test
loop:
        # code for loop body; not relevant to question
test:
        beq $t0, $t1, loop
```

```
        j    test
loop:
        # code for loop body; not relevant to question
test:
        sub  $t9, $t0, $t1
        bez  $t9, loop
```

b) [6 points] The datapath design given in the diagram does not necessarily handle the `beq` instruction correctly unless we insert a number of stalls (`nop` instructions) before every `beq`. Explain why this is so.

The decision whether to take the branch, and the branch target address, do not reach the IF stage until beq reaches the MEM stage of the pipeline.

The given design offers no way to stall (avoiding fetching any instructions behind the beq until the decision is made), or to roll back misfetched instructions (stalling instructions that were fetched but should not be executed).

c) [8 points] The obvious advantage of `bez` (over `beq`) is that `bez` does provide a conditional branch feature, but we don't have to perform an arithmetic operation with the ALU in order to determine whether the branch should be taken. Explain exactly how the given datapath could be modified to make the execution of the `bez` instruction more efficient than that of the `beq` instruction.

We could:

- **add a zero detector immediately after the operands are fetched from the register file in the ID stage**
- **move the AND gate from the MEM stage to the beginning of the EX stage**
- **send the output from the zero detector to the AND gate, along with the Branch signal**

However... if we need to forward an operand to bez then the decision hardware needs to be at the beginning of the EX stage, so the current forwarding logic will suffice, or else we must modify the forwarding logic to forward into the ID stage in this case.

---

5. [8 points] Suppose you need to choose a design for a L1 cache for a new processor, and you are told that you must fit the cache into a a fixed, specific amount of area on the processor die. Discuss the relative merits of using a fully-associative design vs using a direct-mapped design, in this situation.

A fully-associative cache would require a tag-comparator for every line in the cache.

A direct-mapped cache would require only one tag-comparator.

So, a direct-mapped design would need less area for comparators and hence could include more cache lines, and therefore store more bytes of user data.

A common mistake on this question was to ignore the fact that the space constraint for the cache hardware makes other considerations far less important.

6. A cache design is 16-way set associative, storing 512 KiB of user data, with a block size of 128 bytes. Here are some helpful values:

k	4	5	6	7	8	9	10	11	12
2^k	16	32	64	128	256	512	1024	2048	4096

a) [4 points] How many sets will this cache have? Justify your conclusion.

The capacity of a set equals the number of lines in the set (16 in this case) times the block size, so a line can store 2^4 * 2^7 = 2^11 bytes of user data (2 KiB). Since the capacity of the cache is 512 KiB = 2^9 * 2^10 = 2^19 bytes.

Therefore, the number of sets would be 2 ^19 / 2 ^11 = 2^8 = 256 sets.

b) [4 points] How many lines will there be, per set, in this cache? Justify your conclusion.

By definition, a 16-way set associative cache has 16 lines per set.

c) [4 points] How many bits of an address will be used to determine the set to which the data at that address will map? Justify your conclusion.

Since there are 256 sets, we need 8 bits to determine a set number.

d) [4 points] How many bits will there be in a tag field for this cache? Justify your conclusion.

The tag is composed of the high bits that are left over after the bits that determine the set number and the block offset are removed.

Since the block size is 128 bytes, we need 7 bits to determine the block offset; and we need 8 bits for the set number.

So, that leaves 32 – 15 = 17 bits for the tag (assuming 32-bit addresses).

7. [10 points] A system has an L1 cache and an L2 cache. The L1 cache has an average hit rate of 90%, and takes 1 clock cycle to access. The L2 cache has an average hit rate of 95%, and takes 20 clock cycles to access. The access time for DRAM is 150 clock cycles.

What is the average memory access time for this system? Round your answer to the nearest tenth of a clock cycle.

AMAT = 1 + .10 * 20 + .10 * .05 * 150 = 1 + 2 + .75 = 3.75 cycles

---

8. [12 points] Haskell Hoo IV makes the following assertion regarding k-way associative cache designs:

> If we use half as many lines per cache set, and twice as many cache sets, keeping the same block size and the same total capacity for user data, that will cut the time needed to search the cache for a specific tag in half.

Discuss whether Haskell Hoo's assertion is correct or incorrect, and explain why.

There are two keys steps in determining whether a given tag is in the cache:

- **determine which set is relevant**
- **determine if any line in that set contains a match for the given tag**

The first step is performed by "indexing" into the cache, using the relevant bits of the address to determine the correct set. This has essentially the same cost regardless of the number of sets.

The second step involves comparing the tag fields of all the lines in the relevant set to the given tag, all at the same time. So, this is also essentially the same cost regardless of the number of sets. The cost may change very slightly if the length of the tag field changes, but in this case the length of the tag will change by only 1 bit.

The ANDing of the comparator outputs (to determine if a hit occurred) would have to deal with half as many input bits, which might save a level or so of AND gates. Similarly, the selector that determines which block gets passed out would receive fewer candidates, and that might save a level or so of gates.

So, the cost of the search for a matching tag might change very slightly, but it will certainly not change by a factor of 2.

What effect this change might have on the cache's ability to support some form of locality was not relevant to the given question, so comments about cache collisions or locality were irrelevant.