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.

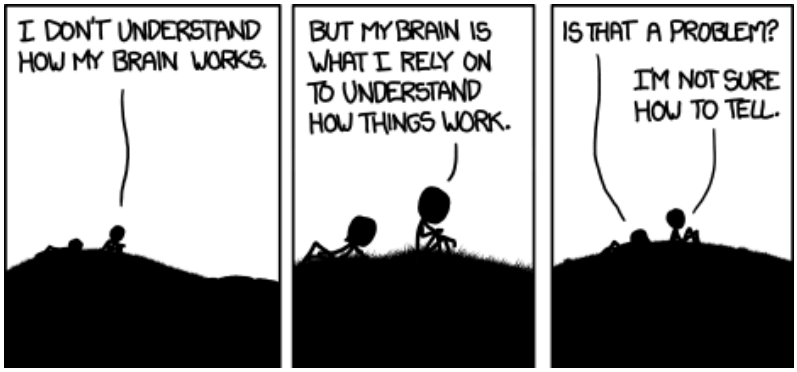
Do not start the test until instructed to do so!

Name Rubric

printed

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

signed


I DON'T UNDERSTAND HOW MY BRAIN WORKS.
BUT MY BRAIN IS WHAT I RELY ON TO UNDERSTAND HOW THINGS WORK.
IS THAT A PROBLEM?
I'M NOT SURE HOW TO TELL.

xkcd.com

1. [14 points] Consider the following statement about the MIPS32 datapath design.

> The adoption of a pipelined datapath design improves performance by increasing instruction throughput. Under ideal condition and with a large number of instructions, the speed-up from pipelining is approximately equal to the number of pipe stage; that is, a five-stage pipeline is nearly five times faster than a single-cycle datapath.

The above statement is true under ideal conditions. Explain two unideal cases in which speed-up could be less than the number of pipeline stages.

Solution:

- The pipeline stages are not balanced (e.g., one stage dominates the overall execution time)
- For (data/control) hazards, we have to stall the pipelining.
- Memory accesses often take more than a cycle (e.g., cache misses)

7pts: condition 1
7pts: condition 2

---

2. [10 points] Consider a standard MIPS 5-stage pipeline where:

- Branches are resolved in the MEM stage
- All branches are predicted to not be taken
- A Forwarding unit that can deal with data hazards by forwarding data to the EX stage, from either the EX/MEM or MEM/WB interstage buffers is implemented.
- A Hazard Detection unit that adds stalls in the pipeline, if necessary, is implemented.
- The CPI would be 1.0 if it were not for control or data hazards

The program running on the machine has the following characteristics

- 15% of all instructions are branches
- 40% of all branches are taken
- 20% of all instruction are loads
- 20% of all instructions are dependent on the instruction immediately before them
- 10% of all instruction are stores
- The program is quite long (millions of instructions)

What is the CPI that would be achieved by the machine on the program described? Round your answer to the nearest hundredth of a cycle.

Solution:

Ideal CPI = 1
Stalls due to Control Hazards = I * .15 (branch) * .4 (misprediction) * 3 (as resolved in MEM)
Stalls due to (load-use) Data Hazard = I * .2 (load) * .2 (dep.) * 1
➔ CPI = 1 +.15 * .4 * 3 + .2 * .2 * 1 = 1.22

5pts: control hazard (1pts – br, 2pts - misprediction, 2pts – 3 stalls)
5pts: load-use data hazard (1pts – load, 2pts – dep, 2pts – 1 stalls)
-1pt for unrelated part (if any)

3. The MIPS pipeline is designed specifically to execute its instruction set. A standard MIPS pipeline consists of five stages: IF, ID, EX, MEM, and WB. The EX stage is placed before the MEM stage to allow ALU in the EX stage to be used for address calculation. For load and store instructions, this pipeline design supports *register displacement addressing*, in which the target memory address is calculated as the sum of a register and a constant (immediate) in the instruction. For example, the following load instruction with register displacement addressing is supported.

```
lw $rt, imm ($rs)        # $rt = MEM[$rs + imm]
```

In this question, we will consider a variation in the MIPS pipeline structure and its impact on the instruction set. Suppose a new pipeline design, in which the MEM and EX stages are swapped. The new pipeline would look like this: IF, ID, MEM, EX, WB.

a) [8 points] Explain how this new pipeline design would affect the use of load/store instructions?

Solution:
Since there is no longer an EX in front of MEM, this change would prevent us from using register displacement addressing. This implies that many load/store instructions need to be converted into two instruction sequences: add instruction (to calculate target address) and memory instruction with register addressing (no offset).

E.g. lw $r1, 8($r2) ---> add $r3, $r2, 8
lw $r1, ($r3)

All or nothing.
Observation that register displacement addressing is no longer available is sufficient for full credit.

b) [8 points] Would this new pipeline design allow us to use a new form of instruction not supported before? If yes, give an example of such an instruction that could be added to the existing instruction set. If no, explain why this change would not enable support for any new instruction forms.

Solution:
We can use instructions with one memory input operand. For instance,
add_m $rd, $rs, ($rt) // $rd = $rs + MEM($rt)

All or nothing.

4. The Forwarding unit discussed in class can detect a MEM hazard that, if it exists, needs to forward data from the MEM/WB interstage buffer by comparing the source register numbers of the reading instruction in the EX stage, and the target register number of the writing instruction in the MEM stage (Condition $4 in the following checks). In addition to this check #4, the forwarding unit actually performs additional checks (#1, #2, #3) as follows.

Check for RegisterRs	if (MEM/WB.RegWrite
	and (MEM/WB.RegisterRd != 0)
	and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd !=0))
	and (EX/MEM.RegisterRd != ID/EX.RegisterRs)
	and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
		// Condition #1
		// Condition #2
		
		// Condition #3
		// Condition #4

Check for RegisterRt	if (MEM/WB.RegWrite
	and (MEM/WB.RegisterRd != 0)
	and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd !=0))
	and (EX/MEM.RegisterRd != ID/EX.RegisterRt)
	and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
		// Condition #1
		// Condition #2
		
		// Condition #3
		// Condition #4

a) [6 points] Explain why the Forwarding unit should check condition #1.

Solution:
Some instructions do not write registers. We only need to detect MEM hazard if the instruction is meant to write registers (R-type and load instructions.)

All or nothing.

b) [4 points] Explain why the Forwarding unit should check condition #2.

Solution:
In MIPS, register 0 ($r0, $zero) is hard-wired to a value of zero and can be used as the target register for any instruction whose result is to be discarded. (e.g., *add $r0, $r1, $r2* may be used to check if $r1+$r2 results in overflow, but did not want to store the result anywhere). For the case that an instruction in the pipeline has $r0 as its destination, we want to avoid forwarding its possible nonzero result value.

All or nothing.

c) [6 points] Explain why the Forwarding unit should check condition #3.

Solution:
When the source register of reading instruction depend on both the destination register of immediately preceding instruction (a cycle ahead) and that of the second preceding instruction (two cycles ahead), then we should forward the data from immediately preceding instruction to obey program order.

All or nothing.

5. This question relates to the execution of MIPS32 instructions on the pipelined data path studied in class, with a Forwarding unit and Hazard Detection unit for both load-use data hazards and control hazard. Assume that branches are resolved in the MEM stage and predicted not-taken.

Consider executing the following instructions:

```
Loop:  lw    $r1, 0($r2)        #1. $r1 = MEM[$r2 + 0]
       addi  $r1, $r1, 1        #2. $r1 = $r1 + 1
       sw    $r1, 0($r2)        #3. MEM[$r2 + 0] = $r1
       addi  $r2, $r2, 4        #4. $r2 = $r2 + 4
       sub   $r4, $r3, $r2      #5. $r4 = $r3 - $r2
       beq   $r3, $r4, Loop     #6. if ($r3 == $r4) goto Loop
```

a) [6 points] Suppose the loop iterates only once. In other words, `$r3` was not equal to `$r4` on the first check, and thus `#6 beq` branch was not taken. Calculate the total number of cycles to execute the given instructions. Give a brief explanation of your analysis.

Solution:
lw
load-use hazard (nop)
addi
sw
addi
sub
beq

6 + 1(stall) + 4 (cycles to complete all pipeline stages) = 11

3pts: load-use hazard (2pts for 1 stall, 1pt for the observation that load-use hazard exists)
2pts: 1 cycle/instr w/o hazards
1pt: the last 4 cycles
-1pt for unrelated (wrong) part

b) [8 points] Suppose the loop iterates twice. In other words, `#6 beq` branch was taken for the first time, and not taken for the second time. Calculate the total number of cycles to execute the given instructions. Give a brief explanation of your analysis.

Solution:
lw
load-use hazard (nop)
addi
sw
addi
sub
beq
mispredicted control-hazard (nop)
mispredicted control-hazard (nop)
mispredicted control-hazard (nop)
lw
load-use hazard (nop)
addi
sw
addi
sub
beq

6 + 1(load-use stall) + 3(branch stalls) + 6 + 1(load-use stall) + 4 (to complete) = 21

4pts: control hazard (2pts for 3 stalls, 2pt for the observation that control hazard exists)
1pts: load-use hazard
2pts: 1 cycle/instr w/o hazards
1pt: the last 4 cycles
-1pt for unrelated (wrong) part

6. A cache design is 16-way set associative, storing 256KB of user data, with a block size of 128 bytes.

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

Solution: There are 16 lines per set, so a set stores 16 * 128 or 2^11 bytes of user data. Since the total capacity of the cache is given as 2^18 bytes, there must be 2^7 or 128 sets in the cache.

2pts: calculating the #bytes of user data per set
2pts: dividing that into #bytes of user data in the cache

Note: 1 KB is 2^10 or 1024 bytes, not 1000 bytes.

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

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

All or nothing, based on consistency with part a).

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.

Solution: Since there are 2^7 sets, we need 7 bits to determine the set.

All or nothing, based on consistency with answer to part a).

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

Solution: We need 7 bits to index the bytes in a block, and 7 bits to determine the set, so there will be 18 bits in the tag fields, if we have 32-bit addresses.

2pts: explaining where the other 14 bits of an address go
2pts: stating consistent conclusion for size of tag field (we didn't actually say we have 32-bit addresses)

Note: Due to an oversight, the number of bits in an address was not stated. I didn't care what size you assumed, but you did need to be explicit about making the assumption unless you assumed a standard size (32 or 64 bits). Assuming 19 bits (since there are 2^19 bytes in the cache) does not make sense, since that would imply the cache was a large as memory.

7. [6 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 10 clock cycles to access. The access time for DRAM is 100 clock cycles.

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

Solution: 0.90 * 1 + 0.10 * 0.95 * (1 + 10) + 0.10 * 0.05 * (1 + 10 + 100)
= 0.9 + 0.095 * 11 + 0.005 * 111 = 2.5 cycles

(Alternatively: 1 + 0.10 * 10 + 0.10 * 0.05 * 100 = 2.5 cycles)

4pts: setting up a valid weighted sum expression
2pts: calculation of final value

---

8. [8 points] Tom Hoo III makes the following assertion regarding cache block sizes:

> If we increase the block size and keep the total capacity of a cache the same, the cache will be less likely to take advantage of temporal locality when the running process exhibits that behavior.

Jim Bob Neer neither disagrees nor agrees with that statement, but he says:

> If we decrease the block size and keep the total capacity the same, we will increase the likelihood that the running process will exhibit temporal locality.

Critique both assertions. Discuss whether either assertion is correct or incorrect, and explain why.

Solution:
Hoo's statement is likely to be correct. If the running process exhibits temporal locality, there is no reason to expect that the addresses that are being accessed repeatedly will necessarily be at nearby locations in memory. They might be, if they are local automatic variables for instance, but they could also be widely scattered elements in a dynamically-allocated data structure. If we have bigger blocks, we have fewer cache lines, and therefore fewer independently-chosen "chunks" of memory in the cache.

Neer's statement is simply silly. The behavior (as opposed to the performance) of a process depends on the code and the input, and not at all on the nature of the cache.

For Hoo:
3pts: discussing the (lack of) relation for addresses under temporal locality
1pts: reaching a sensible conclusion

For Neer:
3pts: discussing the fact that a process's behavior wrt locality is independent of the cache
1pts: reaching a sensible conclusion