Instructions:

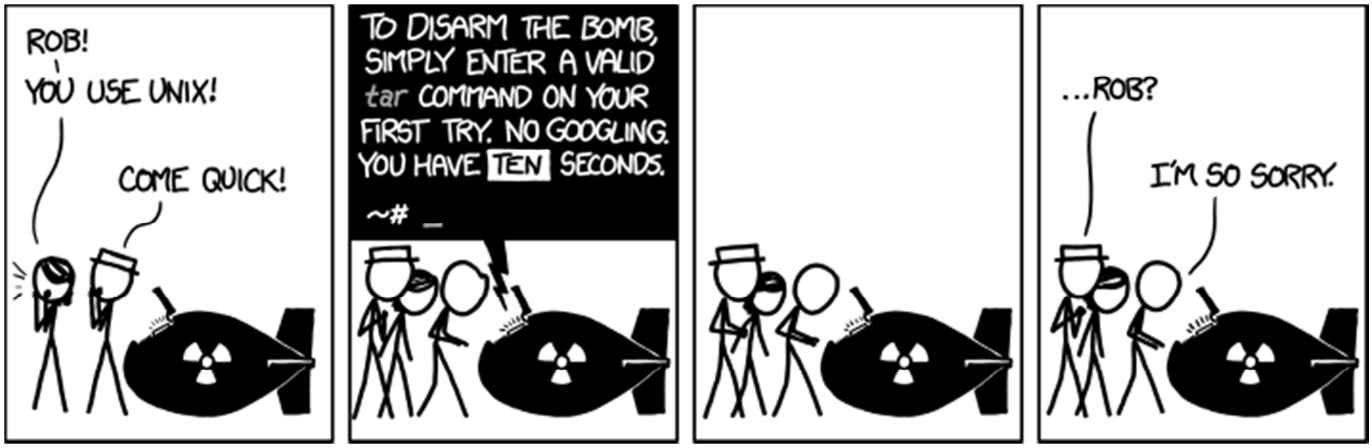
- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. No calculators or other computing devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test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


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

xkcd.com

1. [10 points] Consider the following statement about the MIPS32 datapath design:

> The adoption of a pipelined datapath design increases the utilization of the hardware (percentage of the time a unit is actually in use).

Is the statement true or false? Justify your conclusion.

Aside from the interstage buffers and the hardware related to forwarding and hazard detection, all the hardware in the pipeline is also in the single-cycle datapath, and plays the same roles. With the single-cycle datapath, hardware is only busy for a small part of the full clock cycle. For example, the ALU is only in use for a relatively short period of time, certainly no more than 1/5 of the cycle.

OTOH, each stage of the pipeline is busy during each clock cycle (barring execution of a stall). It's true that for certain instructions some hardware will be busy doing irrelevant things, but the same is true with the single-cycle datapath.

Yes, adopting a pipelined design will increase the percentage of the time that most, if not all, datapath components are actually in use.

---

2. Recall how forwarding is employed in the MIPS datapath. Suppose a sequence of instructions is currently in the pipeline; let's label the instructions `I1`, `I2`, `I3`, `I4` and `I5` and assume they are in WB, MEM, EX, ID and IF respectively.

a) [6 points] What are the precise conditions in which instruction `I2` must receive a forwarded value from the MEM/WB interstage buffer?

There is only one instructions shown to be ahead of I2 in the pipeline, and I2 would receive a forwarded value when it enters the EX stage. In order for the forwarded value to come from the MEM/WB buffer, it must be written by an instruction that's one stage ahead of I1. So:

I4 receives a value forwarded from MEM/WB iff an instruction preceding I1 writes to a register that I4 reads, and that I1 does not write to the same register.

b) [6 points] What are the precise conditions in which instruction `I3` must receive a forwarded value from the EX/MEM interstage buffer?

Similar to the previous part, except that now the writing instruction must be one stage ahead of the reading instruction, so:

I3 receives a forwarded value from the EX/MEM buffer iff I2 writes to a register that I3 reads.

3. Suppose the following sequence of instructions is executed on the MIPS pipeline:

```
lw   $t0, 0($s1)       # 1
add  $t2, $t1, $t0     # 2  reads $t0 written by #1 (load-use hazard)
lw   $t1, 4($t2)       # 3  reads $t2 written by #2
sub  $s1, $t2, $t1     # 4  reads $t1 written by #3 (load-use hazard),
                             $t2 written by #2
add  $s2, $t2, $s1     # 5  reads $s1 written by #4
sw   $s2, 8($s1)       # 6  reads $s1 written by #4, $s2 written by #5
```

a) [8 points] Would the Hazard Detection Unit take any action with respect to this sequence of instructions? If so, describe exactly what it would do, and why.

As noted above, #2 reads a value written by #1, but #1 is a load instruction, so forwarding alone will not handle the situation. And, #4 reads a value written by #3, but #3 is also a load instruction, so forwarding alone will not handle that situation either.

The Hazard Detection Unit will insert a stall between #1 and #2, and another stall between #3 and #4.

b) [12 points] Assuming any actions described in the previous question have been taken, would the Forwarding Unit ake any action with respect to this sequence of instructions? If so, identify the writing instruction, the reading instruction, the register involved, and which interstage buffer the value would be forwarded from.

With the stalls inserted, we must reassess the need for forwarding:

```
lw   $t0, 0($s1)       # 1
nop
add  $t2, $t1, $t0     # 2  forward $t0 from MEM/WB
lw   $t1, 4($t2)       # 3  forward $t2 from EX/MEM
nop
sub  $s1, $t2, $t1     # 4  forward $t1 from MEM/WB, $t2 no issue
add  $s2, $t2, $s1     # 5  forward $s1 from EX/MEM
sw   $s2, 8($s1)       # 6  forward $s1 from MEM/WB, $s2 from EX/MEM
```

To sum it all up:

writer	reader	register	source
#1	#2	$t0	MEM/WB
#2	#3	$t2	EX/MEM
#3	#4	$t1	MEM/WB
#4	#5	$s1	EX/MEM
#4	#6	$s1	MEM/WB
#5	#6	$s2	EX/MEM

Remember: operands from one stage ahead come from the EX/MEM buffer, operands from two stages ahead come from the MEM/WB buffer, operands from more than two stages ahead do not require forwarding.

4. [8 points] The Hazard Detection Unit essentially turns an instruction into a `nop` by setting many of the control signals that correspond to the instruction to zero. Could the same result be accomplished by setting just some of those signals to zero? If so, describe a minimal set of control signals that would suffice and explain why. If not, explain why.

What's obviously essential is to render an instruction harmless. No instruction produces output that's directly taken as input by another instruction. Input values are always read from a register or data memory.

So, the Hazard Detection Unit must prevent it from causing anything to be stored, whether in a register or data memory.

However… the instruction should not be allowed to read from an arbitrary memory location, since that could lead to a memory protection error.

The Branch signal might seem to be problematic as well, but the Control unit will always set Branch to 0 in any situation where the Hazard Detection Unit would come into play.

Therefore, it would be sufficient if the Hazard Detection Unit set the following control signals to zero:

RegWrite
MemWrite
MemRead

---

5. [10 points] Explain why the pipeline datapath shown in the test supplement does not correctly support execution of `beq` instructions. Be precise and complete.

The given pipeline design acts on the decision "to branch or not to branch" only when the beq instruction reaches the MEM stage and the AND gate there sets the control signal for the MUX in the IF stage that feeds an instruction address to the PC.

By the time beq has reached the MEM stage, succeeding instructions will have been fetched and will now occupy the EX and ID stages (assume the AND gate acts quickly enough to cause the correct fetch to occur while beq is in MEM).

If the branch IS NOT taken, all is well.

If the branch IS taken, there are two instructions in the pipeline that should have never been fetched, and the given design does not have any way to turn those in to NOP instructions.

6. [18 points] A system has 2^{36} bytes of DRAM. The system has a single level of cache memory, organized in 2^8 sets each holding 2^6 blocks, with a block size of 2^{10} bytes.

Recall that we number bits of an address from low to high, starting at 0: $A_n A_{n-1} \ldots A_1 A_0$

How many bits are needed for an address in this system?

2^{36} bytes of DRAM implies we need (at least) 36 bits for an address.

Which bits of a DRAM address **A** would be used to determine the set number to which that address would be mapped?

Since blocks are 2^{10} bytes, the low ten bits (A_0 to A_9) are used for byte offsets; since there are 2^8 sets, set numbers require the next 8 bits: A_{10} to A_{17}.

Which bits of a DRAM address **A** would be used for the tag field?

The tag consists of the bits not used for byte offsets or set numbers: A_{18} to A_{35}.

How many bytes of user (DRAM) data can the cache hold? Express your answer as a power of 2.

$2^8 * 2^6 * 2^{10} = 2^{24}$

For a given address, in how many places could the corresponding block of DRAM be stored in the cache?

For a given address, there is only one cache set that could store the corresponding DRAM block. Within that set, there are 2^6 possible locations for the block to be stored.

---

7. [10 points] Give an example of C code that exhibits spatial locality, and explain why it does.

There are many possible answers; the simplest will involve performing a stride-1 traversal of an array.

8. This question is about the effect of associativity on cache implementation.

a) [2 points] Which cache design approach implies that for each DRAM address, the corresponding block of data could be stored at a number of locations in the cache, but not just anywhere?

A k-way associative design fits this description.

b) [4 points] Describe the logical conditions that determine whether a particular cache line (block) contains the data matching a particular DRAM address.

There is a match if and only if:

the valid bit for the cache line is 1

AND

the tag field of the cache line matches the tag field of the DRAM address

c) [6 points] What property of a cache would you change in order to increase the chance of taking advantage of spatial locality? Describe the change precisely, and explain why it would be appropriate.

In order to take advantage of spatial locality, you want the first hit to a sequence of accesses to cause the loading (to the cache) of a significant number of subsequent memory locations.

That implies that you want to choose a larger size for the cache blocks.