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 and the MIPS reference card. No calculators or other computing devices may be used.
- 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 5 questions, 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

For question 1, refer to the copy of Fig 4.24 from P&H that was distributed along with the test.

1. a) [10 points] For which of the supported MIPS assembly instructions is the Shift-Left-2 unit labeled **A** necessary? Why?

`beq`:

The branch target address is computed by sign-extending the 16-bit immediate from the instruction, multiplying the resulting value by 4 and then adding that value to the value PC+4. The multiplication by 4 is accomplished by performing a 2-bit left shift, and this unit supplies the necessary circuitry to do that.

The `j` instruction doesn't use any of this… it has a separate shift unit and doesn't use an Add unit at all.

b) [10 points] For which of the supported MIPS assembly instructions must the select line, MemtoReg, for the multiplexor labeled **B** be set to 1? Explain why.

`lw`:

If the multiplexor's select bit is set to 1, then a value taken from the Read Data port on the data memory will be transferred to the Write Data port on the register file. The only instruction that transfers a value from data memory to the register file is `lw`.

c) [10 points] The MemWrite signal must be set to 1 for one particular supported instruction. Describe what could go wrong if it was set to 1 for any of the other supported instructions.

Setting the MemWrite signal to 1 causes the data memory to write the value on its Write Data port to the memory address that is present on its Address port.

If MemWrite were set to 1 for one of the R-type instructions or `beq`, then the value fetched for the second operand would be stored to the address corresponding to the value computed by the ALU, which is NOT an address logically.

If MemWrite were set to 1 for `lw`, the value of the base register would be stored into the address that is supposed to be read.

In each of those cases, a logically invalid value will have been written into a location in data memory.

For question 2, refer to the copy of Fig 4.33 from P&H that was distributed along with the test.

2. Consider the stages of the MIPS pipelined datapath and the timing assumptions.

a) [10 points] Consider the Add unit shown in the third stage of the pipeline. Is there a logical reason the Add unit could not be moved to the second stage? Explain.

Yes. The lower input to the Add unit must be sign-extended and shifted. According to the timing assumptions, it will take 50 ps to complete the sign-extension. Since the Add unit needs 200 ps to stabilize, we cannot move it into the same stage as the sign-extension logic if we are to complete the entire stage in 200 ps.

Instruction decoding is irrelevant. The branch target address is always computed, no matter what kind of instruction is received. All that is needed is the lower 16 bits from the instruction word, and that is available at the beginning of the second stage.

b) [10 points] Does any single supported instruction non-trivially use all of the hardware components shown in the third stage, EX? Explain.

Yes. The `beq` instruction uses the Shift and Add units to compute the branch target address, and the multiplexor and ALU to compute the difference of the two registers which are to be compared.

c) [10 points] Recall that register writes take place in the first half of a clock cycle and register writes take place in the second half. Write a sequence of MIPS assembly language instructions that would take advantage of this, and explain how they would do so.

Note that this has little to do with resolving a hazard by forwarding, unless you were very careful in your explanation. You need an example in which the result of one instruction (written to the register file) is used as input by a later instruction, and the writing of the result aligns in the same clock cycle with the reading of the input.

Now only R-type instructions and `lw` write a result into the register file, so you must start with one of those. And, the result won't be written to the register file until the R-type or `lw` reaches the WB stage, so you need a later instruction that will be reading its operands at the same time. Reading of operands takes place in stage 2, so we must overlap stage 5 and stage 2, which means we must have something like this:

```
add   $t0, $t1, $t2      # result in $t0
#
#  TWO instructions here to force the alignment described above
#
add   $t5, $t0, $t3      # uses result from first add
```

3. A program uses 40% integer multiplication instructions, 20% integer addition instructions, and 40% other instructions, and takes 100 seconds to run on a particular machine. Suppose that the design of the machine can be altered so that integer multiplication will be 50% faster and integer addition will be 30% faster, and everything else will take the same amount of time.

a) [10 points] How long will it take to run the program on the new machine? Justify your conclusion.

Applying Amdahl's Law:

$$Time_{improved} = 40 + \frac{40}{1.5} + \frac{20}{1.3} \approx 40 + 27 + 16 = 83 \text{ seconds}$$

Note: 50% faster doesn't mean that the time to complete is 50% smaller.

b) [5 points] For this particular program, what will be the speedup on the new machine? Justify your conclusion.

Speedup is the old execution time divided by the new execution time:

$$\frac{100}{83} \approx \frac{7}{6}$$

c) [5 points] State a tight upper bound on the amount of speedup that could be obtained by improving the speed of integer addition alone. Justify your conclusion.

At best, the time for integer addition could be reduced to nearly 0, which would still leave a total execution time of just over 80 seconds. So an upper bound on the speedup would be:

$$\frac{100}{80 + \alpha} \xrightarrow[\alpha \to 0]{} 1.25$$

4. [10 points] For a typical program, the average CPI on machine A is 1.5 and the average CPI on machine B is 2.0. From those facts, what can be established about the relative speeds of the two machines, with respect to this program? Explain fully.

Knowing the CPIs tells us nothing about the relative speeds of the two machines on this program. We would have to know the cycle time for each machine as well. As it stands, all we can say is that the program will require fewer cycles on machine A than on machine B.

5. [10 points] The MIPS machine language format for the R-type instructions places the bits for the destination register after those for the two source registers, in positions 15:11, even though the MIPS assembly language definition always places the destination register first. Considering the simple MIPS datapath referred to in the first question on this test, explain why this decision makes sense and what aspect of that datapath would be more complicated if the R-type format had been different.

In the current scheme, the two read registers are always specified by the same bit fields of a machine instruction, namely 25:21 and 20:16. If the destination register were not specified by bits 15:11, then it would displace one of the read register fields; for example, the read register might be specified by bits 20:16 and the second read register by bits 15:11.

But, that could only apply to instructions that take three register operands (R-type). For I-type instructions, we would still have to specify both read register numbers using fields 25:21 and 20:16.

If that were done, then we would have to know the instruction type before we would know whether the second read register was specified by 20:16 (`beq`, `sw`) or by 15:11 (for R-type). Therefore, we would lose the option of "optimistically" reading two registers during the decoding of the instruction's opcode, and that would slow the execution of instructions.

There are other options, of course. We could put the read register bits in place of the first read register; that has the same consequences discussed above. We could put the read register bits somewhere within 10:0, and that could complicate the processing of the immediate field, since we'd have to put 5 bits of that elsewhere, presumably in 15:11 but out of the usual order.