This exam is closed book and note. You may use one double-sided sheet of notes, but no magnifying glasses! Calculators, computers, and cell phones are not permitted. Slide rules, abaci, sextants, astrolabes, and sundials are permitted—in fact, they are encouraged.

This exam has 70 points. Basically, 1 point per minute plus 10 minutes of buffer. You should treat the point system as a timing mechanism. If you see a 3-point question, ask yourself “Can I answer this question in 3 minutes?”. If the answer is “no”, you may want to go on to another question.

Please ensure that your answers are legible. And please show your work.

Finally, please answer the question that is asked. If I give you 1 inch of space in which to write the answer, it’s because I am looking for a very short specific answer to a very specific question.

<table>
<thead>
<tr>
<th>Question</th>
<th>Topic</th>
<th>Max</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Performance</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Pipelining</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Instruction Sets</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Integer Arithmetic</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Verilog</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Feedback</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>68</strong></td>
<td></td>
</tr>
</tbody>
</table>
This page intentionally left blank
1. \[1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 10 \text{ Points}\] **Performance.**

(a) [1 point] Processor A has a CPI of 2 and a clock frequency of 2 GHz. What is its MIPS (millions of instructions per cycle)?

**Answer:** \[\frac{2,000 \text{ MHz}}{2} = 1,000 \text{ MIPS}.\]

(b) [1 point] Processor B has a CPI of 3 and a clock frequency of 2.5 GHz. Which processor is likely to have the deeper pipeline, A or B?

**Answer:** B.

(c) [1 point] Which processor has higher overall performance?

**Answer:** We can’t really tell because we don’t have information about their instruction sets and compilers.

(d) [1 point] What is one mechanism by which Moore’s Law, the continued miniaturization of MOS transistors, results in improved performance?

**Answer:** Miniaturization reduces transistor size and wire length resulting in faster switching. Moore’s Law allows existing designs to run at a higher clock frequency.

(e) [1 point] What is another mechanism by which Moore’s Law results in improved performance?

**Answer:** Architects can use the additional transistors to increase parallelism and reduce CPI. There is also the “Moore’s Curve” psychological effect in which companies (used to) design products that met the public’s performance expectations.

(f) [1 point] What is a microbenchmark?

**Answer:** A micro-benchmark is a small program (e.g., 8-queens, Towers-of-Hanoi) used to measure one or two aspects of processor performance.

(g) [1 point] Microbenchmarks should not be used to report machine performance. Why?

**Answer:** Microbenchmarks are not representative of real programs which have much more complex behaviors.

(h) [1 point] In a given design, different transistors will have different “widths.” What is an advantage of making a transistor wider?

**Answer:** Making a transistor wider increases its speed by lowering its resistance, i.e., increasing the cross section through which electrons can travel.

(i) [1 point] What is a disadvantage of making a transistor wider?

**Answer:** Making a transistor wide increases the capacitance of its “gate”, slowing down upstream “driver” transistors.

(j) [1 points] In a given design, all transistors have the same minimal “length” (which is a characteristic of the technology). Why?

**Answer:** Unlike width, there is no speed advantage (or any other advantage) to making a transistor longer than the absolute minimum it can be.
2. \[2 + 1 + 1 + 1 + 3 + 2 + 4 + 1 + 1 + 1 + 2 = 20 \text{ Points}] \textbf{Pipelining.}

(a) \{2 \text{ points}\} Consider a five stage pipeline. Doubling the number of stages in the pipeline to ten is likely to increase the clock frequency, but not quite to double it. What are two reasons for this?

\textbf{Answer:} One reason is that clock frequency is a function of the slowest stage. Doubling the number of pipeline stages will double the clock frequency only if the slowest stage in the new pipeline has half the delay of the slowest stage in the old pipeline. This is very difficult to do and is made more difficult by the second reason—adding stages requires adding latches which adds delay.

(b) \{1 \text{ point}\} Whereas doubling the number of stages is likely to increase clock frequency by less than a factor of two, it is likely to increase MIPS by even less than that. Why?

\textbf{Answer:} A deeper pipeline increases the frequency and penalty of stalls that are required to resolve control and data hazards, increasing CPI.

(c) \{1 \text{ point}\} What are \textit{software interlocks}?

\textbf{Answer:} Software interlocks refers to code scheduling and insertion of nops by the compiler (or assembly programmer) for the purpose of avoiding data hazards on a pipeline that does not avoid them in hardware.

(d) \{1 \text{ point}\} What is the disadvantage of software interlocks relative to hardware interlocks?

\textbf{Answer:} Software interlocks are not compatible across pipeline microarchitectures. Code compiled with software interlocks for one pipeline may break when run on another pipeline.

(e) \{1 \text{ point}\} Most instruction sets have multi-cycle arithmetic operations like multiply. In a typical pipelined implementation, multi-cycle instructions are typically split off the main pipeline at execute and reconverge with it at writeback. What is the advantage of this organization as opposed to putting those operations in the main line of the pipeline?

\textbf{Answer:} This was a little bit of a “thought” question. What would be the problem with putting a 4-cycle multiply in the main pipeline creating something like F,D,X1,X2,X3,X4,M,W? So sure, single cycle arithmetic operations would go through three additional stages doing nothing, but that’s not really a “problem” per se. The problem is that you would need much more bypassing, much more complicated bypass logic, and you would have either a much longer load-use penalty (if single cycle operations took place in X1) or a much longer branch-misprediction penalty (if they took place in X4).
(f) [3 points] Consider a FDXMW pipeline with only single cycle arithmetic operations. The pipeline has no MX bypassing, only WX and WM bypassing. Fill out the pipeline diagram for the code below.

**Answer:**

| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 0 | 1 |
| lw $2,0($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $3,4($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $3,$2,$3 |   | F | d* | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $4,8($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $5,12($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $5,$4,$5 |   | F | d* | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $5,16($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $1,$1,20 |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| beq $9,loop |   | F | d* | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

(g) [2 points] Write out the stall logic for this pipeline.

**Answer:**

\[
\text{Stall} = (F/D.RS1 == D/X.RD || (F/D.RS2 == D/X.RD && F/D.OP != STORE))
\]

The stall is not load specific because there is no MX bypassing.

(h) [4 points] Reschedule the code to reduce data hazards and fill in the pipeline diagram. Remember, there is no MX bypassing. Also, the branch has to remain at the end.

**Answer:**

| Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 0 | 1 |
| lw $2,0($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $3,4($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $4,8($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $3,$2,$3 |   | F | d* | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| lw $5,12($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $5,$4,$5 |   | F | d* | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $5,16($1) |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| add $1,$1,20 |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| slti $9,$1,2000 |   | F | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| beq $9,loop |   | F | d* | D | X | M | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Here is a schedule that completely eliminates the data hazard stalls. You did not have to have this schedule or to completely eliminate the stalls to receive full credit.
(i) [1 point] What is an advantage of a tagged BTB (branch target buffer) over an untagged BTB?

**Answer:** A tagged BTB yields higher accuracy primarily because it contains entries only for instructions that are actual control transfers. Tagging eliminates aliasing among different branches, but most importantly between branches and non-branches.

(j) [1 point] A return address stack (RAS) cannot be tagged. Why?

**Answer:** There is nothing to tag it with. In a BTB, a given entry is written and read by the same static instruction making it possible to use the instruction’s PC (or part of it) as a tag. A RAS entry is written and read by different instructions with different PCs, a call and a return.

(k) [1 point] Consider the following accuracy results for a tagged BTB with 8, 16, 32, 64, and 128 entries. The “Cond” column is the accuracy for conditional branches, “Return” is the accuracy for return instructions, “Other” is the accuracy for all control transfers other than conditional branches and returns, and “Overall” overall accuracy.

<table>
<thead>
<tr>
<th>BTB</th>
<th>Cond</th>
<th>Return</th>
<th>Other</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>50%</td>
<td>0%</td>
<td>40%</td>
<td>46%</td>
</tr>
<tr>
<td>16</td>
<td>54%</td>
<td>0%</td>
<td>46%</td>
<td>50%</td>
</tr>
<tr>
<td>32</td>
<td>68%</td>
<td>0%</td>
<td>78%</td>
<td>69%</td>
</tr>
<tr>
<td>64</td>
<td>73%</td>
<td>0%</td>
<td>96%</td>
<td>78%</td>
</tr>
<tr>
<td>128</td>
<td>73%</td>
<td>33%</td>
<td>97%</td>
<td>79%</td>
</tr>
</tbody>
</table>

What is a possible explanation for return accuracy jumping from 0% to 33% when BTB size is increased from 64 to 128 entries?

**Answer:** So a BTB can work for a return instruction as long as the function is always called from the same place and returns to the same place. This is what is happening at the 128-entry node. What’s happening at 64 entries and below? The return is aliasing with some other branch that occurs between any two consecutive instances of that return.

(l) [2 points] The fact that conditional branch prediction accuracy appears to saturate at 73% suggests that conditional branches are harder to predict than other forms of control instructions. Why is this?

**Answer:** There are two aspects to branch prediction: i) target prediction, and ii) taken/not-taken direction prediction. All control instructions require target prediction but target prediction is easy because most targets are “direct”, hard-coded into the instruction and don’t chance, and the most common control transfers with indirect targets are returns and these can be predicted by a RAS. Only conditional branches require direction prediction and this is indeed more difficult.
3. \[1 + 1 + 3 + 2 + 1 + 1 + 1 + 1 + 2 + 2 + 3 = 20 \text{ Points}\] **Instruction Sets**

(a) [1 point] What is one advantage of fixed-length instructions?

**Answer:** It's easy to compute the PC of the next instruction. And so it is easy to pipeline fetch, or fetch multiple instructions in parallel, or decode instructions, specifically to find out where they begin and end.

(b) [1 point] What is another advantage of fixed-length instructions?

**Answer:** Direct control transfers, which usually specify the target PC as an offset from the current PC, have a longer “reach” because the can specify that offset as a number of instructions rather than as a number of bytes.

(c) [3 points] One measure of the quality of an instruction set is the degree to which it facilitates or, at least does not limit, compiler optimizations. One of the important goals of a compiler is to reduce the number of loads in a program. What is one specific optimization that achieves this and what does it do? Give a code example.

**Answer:** There are two answers here, common sub-expression elimination (CSE) and register allocation. CSE eliminates a load if it can prove that the value already exists in some register. CSE is typically effective when there are redundant loads in separate expressions—here CSE eliminates the second load. Register allocation eliminates spill/fill (store/load) pairs of local variables if it can keep the local variable in a register for the duration of the spill/fill range.

(d) [2 points] Which ISA feature is a greater impediment to reducing the number of loads in a program: accumulator-style instructions that over-write one of their inputs or a small number of registers? Explain.

**Answer:** The bigger detriment is having too few registers. With accumulator-style instructions you may have to create copies of register values you want to keep around, but register copies are not loads.

(e) [1 point] What is one consideration that limits the number of registers an ISA can have?

**Answer:** There are several answers here. First, registers are fast in part because there are few of them. If there were too many, they would be slow. Too many registers would also mean large register specifiers and either larger instructions or fewer registers per instructions. Also, there is a limit to how many things can be put in registers. Finally, more registers means more saving and restoring across function calls.

(f) [1 point] What is another one?

**Answer:** See above.
(g) [1 point] Early instruction sets, like x86, were CISC and included many complex high-level instructions. What was the rationale for this?

**Answer:** In the days before compilers, most code was assembled by hand and humans prefer programming in high-level constructs.

(h) [1 point] What event led to the development of RISC instruction sets?

**Answer:** Moore’s Law crossed a device integration threshold that allowed the construction of a single-chip microprocessor ... but only for a minimal (i.e., RISC) instruction set.

(i) [2 points] Although RISC ISAs have technical advantages over CISC ones, the dominant ISA for high performance processors today is a CISC, x86. What technological force has helped x86 survive, and how?

**Answer:** Moore’s Law gave RISC a temporary advantage but then let x86 catch up. For a given pipeline, the number of additional transistors required for x86 is on the order of 100K. That number was very large in 1980 when individual chips contained only 60K transistors but was small by 1995 when chips contained 10M transistors.

(j) [2 points] What economic force helped x86 survive, and how?

**Answer:** The advantage of backward compatibility. x86 was established in the PC market before any RISC competitor came along. As long as Intel kept performance reasonably close, it maintained its market share because people didn’t want to throw away their existing software.

(k) [3 points] What is a uISA (micro ISA)? What problem does a uISA solve?

**Answer:** A uISA is an ISA that is internal to the processor. The pipeline translates the processor’s nominal external ISA to this internal uISA before execution. A uISA helps reconcile the concerns of a high performance implementation with compatibility.
4. \[1 + 2 + 1 + 1 + 1 + 1 + 3 = 10\] Points \textbf{Integer Arithmetic}

(a) [1 points] In a real world pipeline, clock period is typically determined by the latency of what?

\textbf{Answer:} 2C addition plus latch/bypass.

(b) [2 points] Consider a 32-bit adder that uses a three piece 10-10-12 carry select design. What is the latency of this adder in gate delays?

\textbf{Answer:} 26. \(DB(31) = 2 + \max(\text{DA}(31), DB(19)) = 2 + \max(24, 2 + \max(\text{DA}(19), DB(9))) = 2 + \max(24, 2 + \max(20, \text{DA}(9))) = 2 + \max(24, 2 + \max(20, 20)).\)

(c) [1 point] How many 1-bit full-adders does this design use?

\textbf{Answer:} 54. \(10 + (2 \times 10) + (2 \times 12).\)

(d) [1 point] True or false. A carry-lookahead (CLA) adder has a delay that is linear in the number of bits to be added.

\textbf{Answer:} False. CLA delay is logarithmic in the number of bits to be added. Ripple-carry delay is linear.

(e) [1 point] What does it mean for an N-cycle multiplier to be “pipelined”?

\textbf{Answer:} It means that although each multiply operation takes four cycles, you can start one each cycle.

(f) [1 point] One way to pipeline a multiplier is simply to implement it as a giant Wallace tree and then cut that tree up into stages. Consider an 8-bit by 8-bit multiplier with a 16-bit product. A combinational implementation of this multiplier uses a 7-gate delay conventional adder and carry-save adders (CSA). How many CSAs are needed?

\textbf{Answer:} Essentially we need to use CSAs to “reduce” 8 numbers down to 2. Each CSA reduces three numbers down to two (i.e., by one number) and so we need 6 of them.

(g) [3 points] What is the minimum number of layers in which these CSAs would need to be arranged? As a result, what is the gate delay of this multiplier? Remember the muxes and conventional adder.

\textbf{Answer:} We start with 8 numbers to add (this is combinational so there is no finite state machine and no product register). With one layer of 2 CSAs we are down to 6. With a second layer of 2 CSAs we are down to 4. With a third layer of 1 CSA we are down to 3. With a fourth later of 1 CSAs we are down to 2. The delay of the multiplier is \(2 + 4 \times 2 + 7 = 17\) gates.
5. \[1 + 1 + 2 + 1 + 1 + 1 = 7\] points] **Hardware Design and Verilog**

(a) [1 point] What is the difference between *structural* Verilog and *behavioral* Verilog?

**Answer:** Structural Verilog describes a module in terms of its gates and the connections between them. Behavioral Verilog describes a module in terms of the functions it performs.

(b) [1 point] What is the Verilog Programming Interface? What is it used for?

**Answer:** VPI is a C-style library used during Verilog simulation and debugging. Calls like `$display` and `$stop` are VPI calls.

(c) [2 points] When synthesizing a multiplier to an FPGA, it is preferred to specify the multiplier as $ \times $ rather than as a a collection of gates and wires. What are two reasons for this?

**Answer:** The first reason is just increased productivity that comes from working at a higher level of abstraction. This reason applies whether you are synthesizing to an FPGA or to custom circuits or even whether you are synthesizing or just simulating. The other reason is FPGA specific. An FPGA may contain multiplier blocks on it (the Xilinx Virtex-II Pro FPGAs we are using do) and a behavioral operator like $ \times $ is the only way to use that block. It is very unlikely that Xilinx will figure out that a given gate-level implementation of a multiplier is in fact a multiplier.

(d) [1 point] Consider the following Verilog declaration.

```verilog
wire [15:0] A, B, C, D;
assign B = {8'h00, A[7:0]};
```

In English, what does the following line of Verilog synthesize to?

**Answer:** Zero-extends the low order 8 bits of A and assigns them to B.

(e) [1 point] What does the following line of Verilog synthesize to?

```verilog
C = ˜A + 1;
```

**Answer:** This is actually an error, because wires must be assigned continuously. This is a procedural (i.e., register) assignment of the 2C negative of A to C.

(f) [1 point] What does the following line of Verilog synthesize to?

```verilog
```

**Answer:** Creates a mux wire-assigned to D that chooses C if A is negative and B otherwise.
6. [3 points] **Feedback and Evaluation.**

(a) [1 point] On a scale of 1-10, how difficult was this exam?
(b) [1 point] How many hours did you spend studying for this exam?
(c) [1 point] Which of these did you feel were most useful in terms of studying: attending lectures? reading the notes after lecture? reading the book? homeworks? office hours? old exams? the review session?