0.) **THE EASY ONE**  (1 point total)
Check cover sheet for name, recitation #, PennKey, and signature.

1.) **TREES RIPPLING IN THE WIND**  (7 points total)

1.1) (1 point) What is the delay of a 64-bit **ripple carry** adder? Assume you need to know if the addition overflows.
128

1.2) (1 point) How many levels are there in a 16-input, 64-bit ripple carry adder tree?
4

1.3) (1 point) What is the delay of a 16-input, 64-bit ripple carry adder tree?
4 × 128 = 2^2 + 2^7 = 2^9 = 512

1.4) (2 points) What is the delay of a 16-input, 64-bit **carry save** adder tree?
6 × 2 + 128 = 140

1.5) (2 points) What is the speedup of the carry save tree over the the ripple carry tree, rounded to the nearest tenth? If the speedup depends on the input values, write, “depends on input.”
512/140 = 128/35 ≈ 3.7
2.) TO BE WRITTEN OR NOT TO BE WRITTEN  

(8 points total)

A fundamental characteristic of a computer is the ability to take different actions based on some condition. In LC4, this is handled only by the branch instructions. Either the condition is true and the branch is taken, or it is false and the branch is ignored. It might be nice to allow other instructions to execute conditionally as well. For example, why not have an instruction `ADDn R1, R2, R3` that would add R2 to R3 and store the result in R1 only if the NZP bits were set to N? Otherwise the instruction would do nothing. This behavior is called *predication*, and it turns out the ARM and IA-64 (Itanium) microarchitectures support predication for nearly every instruction.

2.1) (5 points) Aside from changes to the decoder logic, and ignoring any hazards it might create, what changes would be necessary to our five-stage LC4 pipeline to support predication of any instruction. Your answer must be *simple and at most 30 words.*

And the branch taken bit with the regfile and memory WE bit in EX.

2.2) (2 points) What changes to the bypassing and stall logic would be necessary to resolve any potential hazards. Your answer must be *simple and at most 30 words.*

No changes are required.

2.3) (1 point) Aside from the added pipeline complexity, why doesn’t LC4 support predication? Your answer must be *simple and at most 30 words.*

There are not enough bits in the instructions to encode conditions.
3.) SHUT YOUR TRAP  (20 points total)

Fill in the cycle diagram for the code sequence below, starting at memory address x0000 and ending when the program reaches a memory address that is not included in the listing below. Mark all bypasses with arrows. Mark stalls with empty circles. You may stall an instruction at any reasonable stage in its execution. There may be empty rows and columns in the completed table. The first instruction is filled in for you (except any bypassing that may be required to the second instruction).

```
.CODE .OS
.ADDR x0000
.I1 CONST R0, x00
.I2 HICONST R0, x40
.I3 CONST R5, xAA
.I4 HICONST R5, xAA
.I5 HICONST R6, x55
.I6 XOR R5, R5, R6
.I7 TRAP SHUT

Insn Label | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20
.I1 | F | D | X | M | W |
```

4.) PERFORMANCE ANXIETY  (10 points total)

You are designing an embedded processor for a pacemaker. Based on an analysis of the monitoring software that it will run, you find the following mix of instructions, which have the specified latencies in your current design:

<table>
<thead>
<tr>
<th>Insn</th>
<th>Frequency</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>10%</td>
<td>7 cycles</td>
</tr>
<tr>
<td>store</td>
<td>15%</td>
<td>10 cycles</td>
</tr>
<tr>
<td>branch</td>
<td>15%</td>
<td>4 cycles</td>
</tr>
<tr>
<td>add</td>
<td>55%</td>
<td>4 cycles</td>
</tr>
<tr>
<td>divide</td>
<td>5%</td>
<td>5 cycles</td>
</tr>
</tbody>
</table>

4.1) (2 points) What is the CPI of your processor on this mix of instructions?

\[7 \times 0.1 + 10 \times 0.15 + 4 \times 0.15 + 4 \times 0.55 + 5 \times 0.05 = 5.25\]

4.2) (2 points) Based on your design analysis, you figure out that you can halve the cycle latency of any single category of instruction, although you will need to increase the cycle time by 20%. Should you make this change, and if so, what category of instruction should you speed up?

(a) load
(b) store
(c) branch
(d) **add**
(e) divide
(f) no change

4.3) (3 points) What is the CPI of your new design? (If you elected not to change the design, the CPI will not change.)

\[7 \times 0.1 + 10 \times 0.15 + 4 \times 0.15 + 2 \times 0.55 + 5 \times 0.05 = 4.15\]

4.4) (3 points) What is the speedup of your revised design over the original one, rounded to the nearest tenth?

\frac{5.25}{(4.15 \times 1.2)} \approx 1.1
5.) MAC VS. PC  (16 points total)

Your computer company’s customers are defecting in droves to Apple because they want a better experience listening to music and watching video. As an architect, you logically conclude your LC4 architecture must be at fault, and you look for ways to improve its performance at multimedia decoding. It turns out the main thing LC4 is missing is the ability to rapidly compute the product of two numbers plus a third one. This is called multiple-accumulate extremely common in multimedia applications, so you decide to introduce some MAC instructions to the LC4 ISA. In particular, you decide to create the following two instructions:

- **MAC Rd, Rs, Rt, Rw** computes \( Rd = (Rs \times Rt) + Rw \)
- **MACI Rd, Rs, Rt, Imm** computes \( Rd = (Rs \times Rt) + \text{sext}(Imm) \)

**5.1) (4 points)** Is it possible to encode both the MAC and MACI instructions in the LC4 ISA using the 0011 opcode (which is currently unused)? Provide an encoding for each instruction you can support, and an explanation of at most 30 words for each instruction that you cannot support.

**MAC:** 0011 ddd sss ttt www

**MACI:** No instruction bits left to distinguish from MAC or encode a reasonable immediate value.

**5.2) (5 points)** What changes do you need to make to your pipelined LC4 datapath to accommodate this instruction. Assume you have a carry-save multiplier. Your answer must be simple and at most 50 words.

Add a third read port to the regfile; add one mux to a spare multiplier CSA input to select between Rw and 0.

**5.3) (5 points)** How might the architectural changes affect your cycle time, assuming the execute stage was the slowest stage in your original pipeline? Your answer must be simple and at most 30 words.

No effect since the CSA doesn’t actually change.

**5.4) (2 points)** How might the architectural changes affect your stall and bypass logic? Your answer must be simple and at most 30 words.

Need to bypass/stall for Rw just like Rs and Rt.
6.) KEYSSTONE XL (13 points total)

Consider the following LC4 program, in which we have written the jump target using PC-relative addressing just like it is encoded in binary (recall that x82 is a hexadecimal value):

```
.OS
.CODE
.ADDR x8200
CONST R0, x00
HICONST R0, x82
LDR R1, R0, #6
STR R1, R0, #4
XOR R1, R1, R1
XOR R0, R0, R0
JMP (PC+1) + #1
ADDI R0, R0, #1
SLL R0, R0, #4
```

6.1) (2 points) If you run this program on your single-cycle datapath, what will the value of $R0$ be at the end (in hexadecimal)?

$x8200 \ll 4 = x2000$

6.2) (2 points) If you run this program on the fully bypassed, five-stage pipeline that we covered in lecture, what will the value of $R0$ be at the end (in hexadecimal)?

$x0000 \ll 4 = x0000$

6.3) (4 points) Why do you get a different result from the two datapaths? Your answer must be simple and at most 30 words.

The fetch stage doesn’t detect that the first XOR has been overwritten.

6.4) (5 points) What change could you make to one of the datapaths (and which datapath would you modify) to resolve the discrepancy? Your answer must be simple and at most 50 words.

In the pipeline, bypass stores from MEM to PC. If store affects contents of two following instructions, squash them.
7.) BONUS QUESTION (1 point total)

The image below is the mask of an actual processor. Draw an outline that **precisely** encloses the pipeline portion of the processor, including divisions to delineate each pipeline stage.

The Intel 4004 processor does not have a pipeline.