1. (6 points) MIPS. Several load instructions have two forms:
   lh, lhu
   lb, lbu

Why are both forms necessary? When you would use each type?
ANS: lh and lb load 16 and 8 bits respectively and sign extend the value to 32 bits. They are used when
loading values that are to be treated as integers (because the sign extension converts the number into an
equivalent 32 bit number). lhu and lbu also load 16 and 8 bits respectively but they do not sign extend the
value and instead leave the top 16 (or 24) bits of the register as 0’s. They are used for data that is not to be
interpreted as an integer, e.g. characters and bit masks.

2. (5 points) MIPS. What is the machine language instruction for the following assembly language
   instruction:
   sub $4,$7,$8

Give your answer in hex.
ANS: 0000 0000 1110 1000 0010 0000 0010 0010 = 0x00e82022

3. (5 points) MIPS. Write a short sequence of MIPS code to load the i,j-th element (indexed from 0,0) of
   a 2D character array named CHAR that has 7 rows and 3 columns.
ANS:
   lw $t0,i
   lw $t1,j
   mul $t0,$t0,3
   add $t0,$t0,$t1
   lb $s0,CHAR($t0)

4. (5 points) MIPS. Simple procedures as we defined them do not need to perform any stack
   manipulations. Why not? Be specific. List the characteristics of simple procedures that make stack
   manipulations unnecessary.
ANS: Simple procedures have nothing to save on the stack because
1. they do not call other procedures (they do not over write the $ra register and don’t need to save it)
2. they use only the $t registers (so by convention do not have to save registers)
3. they have no more than 4 arguments (so can pass them all in the $a registers)
4. they have no more than 2 return values (so can pass them in the $v registers)

5. (5 points) Number Representation. What is the bit pattern that corresponds to -193.75 in IEEE
   Standard floating point (32-bit, single precision) representation?
ANS: 1 10000110 100000111000000000000000

6. (5 points) Single cycle machine. The RegDst control line in our Single Cycle design selects one of two
   registers as the “Write Register.” Which instructions use the “0” setting of this line? Which
   instructions use the “1” setting?
ANS: The r-type instructions set RegDst to 1 and the lw instruction sets it to 0.

7. (10 points) Single Cycle Design. There are two ALUs in our single cycle design, not counting the
   simple adder used in updating the PC. For each instruction type, describe what each of the two ALUs
   computes by filling in the following table.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Upper ALU</th>
<th>Lower ALU</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>nothing</td>
<td>Computes memory address (adds offset to register value)</td>
</tr>
<tr>
<td>sw</td>
<td>Nothing</td>
<td>Computes memory address (adds offset to register value)</td>
</tr>
<tr>
<td>r-type</td>
<td>nothing</td>
<td>Executes desired operation (add, sub, or, and)</td>
</tr>
<tr>
<td>beq</td>
<td>Computes target branch address (offset plus PC)</td>
<td>Compares values in specified registers (does a subtract &amp; uses zero line)</td>
</tr>
<tr>
<td>J</td>
<td>nothing</td>
<td>Nothing</td>
</tr>
</tbody>
</table>
8. (8 points) Multicycle Design. If the PCSource control line is set to 01, what value is sent to the PC? How is it computed?
ANS: The target of a branch

9. (7 points) In our pipelined design, the multiplexer that feeds control signals into the ID/EX buffer has a possible input of 0 that can be selected either by a signal from the the Hazard Detection Unit. What is the purpose of this alternative? That is, why is a 0 fed in as the control signals to the ID/EX buffer? What does this do? When is it needed. (Note: there are two cases.)
ANS: The 0 is selected in two cases: we need a stall because there is a lw followed immediately by a use of the loaded register or we need to “cancel” instructions that have already started because of a branch. In both cases, the 0 control lines essentially insert a “nop” or bubble into the pipeline.

10. (10 points) Draw a finite state machine that says “YES” to the set of all strings of 0’s and 1’s that start and end with a 1 and contain an odd number of 0’s. That is, your machine should say “YES” to 100111 and 101 but “NO” to 00011 or 101001. ANS:

11. (7 points) Performance. Compare our multicycle design Mm and our pipelined design Mp for a program P with the given instruction mix. Assume that 30% of the lw instructions are immediately followed by a use of that value and that 20% of branches are taken. What is the average CPI for each machine? Which machine is faster? How much faster?
ANS:

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency in P</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>15%</td>
</tr>
<tr>
<td>sw</td>
<td>15%</td>
</tr>
<tr>
<td>r-type</td>
<td>60%</td>
</tr>
<tr>
<td>beq/j</td>
<td>5%</td>
</tr>
<tr>
<td>j</td>
<td>5%</td>
</tr>
</tbody>
</table>

CPI_{Mm} = .15*5 + .15*4 + .60*4 + .05*3 + .05*3 = 4.05
CPI_{Mp} = .15*(.30*2 + .70*1) + .15*1 + .60*1 + .05(.20*2 + .80*1) + .05*2 = 1.105
The pipelined machine is 4.05/1.105 = 3.67 times faster.

12. (5 points) Performance. For a program P and machine M with a CPI of 1.9, if the clock rate is 850 MHz, at what MIPS is M running P?
ANS: 850x10^6 cycles/sec * 1/(1.9 cycles/instruction) = 850/1.9 * 10^6 instructions/sec = 447 MIPS

13. (6 points) Pipelining. Identify each of the data hazard(s) in the following code sequence. For each of them, explain how the hazard could be removed with minimum delay and forwarding. In each case where forwarding is used, name the buffer that the forwarded value comes from. Hint: there are 6 hazards.

    Instruction     | Frequency in P |
    ----------------|----------------|
    add             | $4,$5,$6       |
    sub             | $7,$4,$8       |
    or              | $7,$7,$4       |
    lw              | $6,13($7)      |
    and             | $5,$6,$6       |

Ans:
- Hazard between add & sub on $4 resolved thru forwarding from EX/MEM
- Hazard between add & or on $4 resolved thru forwarding from MEM/WB
- Hazard between sub & or on $7 resolved by forwarding from MEM/WB
- Hazard between sub & lw on $7, not resolved because hazard between or & lw overwrites
- Hazard between or & lw resolved by forwarding from MEM/WB
• Hazard between lw & and resolved by stall & forwarding from MEM/WB

14. (8 points) Cache. Assume 16-bit addresses, fill in the table and compute the miss rate of the following sequence of references (addresses) for a cache with 256 lines of 64 bytes each. What is the size of the tag, index, and byte offset fields of an address?

ANS: tag 2, index 8, offset 6 bits

<table>
<thead>
<tr>
<th>Address Trace (hex)</th>
<th>Tag</th>
<th>Line index</th>
<th>Offset</th>
<th>Hit or Miss?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ad43</td>
<td>10</td>
<td>10110101</td>
<td>000011</td>
<td>miss</td>
</tr>
<tr>
<td>ed43</td>
<td>11</td>
<td>10110101</td>
<td>000011</td>
<td>miss</td>
</tr>
<tr>
<td>ed5c</td>
<td>11</td>
<td>10110101</td>
<td>011100</td>
<td>hit</td>
</tr>
<tr>
<td>ad43</td>
<td>10</td>
<td>10110101</td>
<td>000011</td>
<td>miss</td>
</tr>
<tr>
<td>ab83</td>
<td>10</td>
<td>10101110</td>
<td>000011</td>
<td>miss</td>
</tr>
<tr>
<td>ad40</td>
<td>10</td>
<td>10110101</td>
<td>000000</td>
<td>hit</td>
</tr>
</tbody>
</table>

15. (8 points) Assume that a machine M has the following characteristics for Program P.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>CPI</th>
<th>Frequency in P</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>4</td>
<td>20%</td>
</tr>
<tr>
<td>sw</td>
<td>4</td>
<td>10%</td>
</tr>
<tr>
<td>r-type</td>
<td>3</td>
<td>65%</td>
</tr>
<tr>
<td>beq/j</td>
<td>2</td>
<td>5%</td>
</tr>
</tbody>
</table>

How does the performance of M with a perfect cache (i.e., no misses) compare to the Performance of M with a cache having a data miss rate of 2%, an instruction miss rate of 1%, and a miss penalty of 40 cycles? List the CPIs for program P in both cases.

Ans: 

CPI_{perfect} = (0.20 \times 4 + 0.10 \times 4 + 0.65 \times 3 + 0.05 \times 2) = 3.25

CPI_{realistic} = 3.25 + 0.01 \times 40 + 0.02 \times (0.30 \times 40) = 3.89

The machine with the perfect cache is 3.89/3.25 = 1.2 times faster.