Question 1: (6 points – 2(a) + 2(b) + 2(c))

Consider a MIPS program with the following mix of instructions

- 45% R-type arithmetic instructions
- 25% lw instructions
- 15% sw instructions
- 15% beq instructions

a. Assume that the instruction mix for this program is typical of all the programs that execute on a 2.5 GHz machine in which the R-type instructions execute in 5 cycles, the lw and sw instructions in 6 cycles and the beq instructions in 3 cycles. What is the MIPS rate (million instructions per second) and the average CPI (clock per instruction) for that machine?

CPI: (0.45)(5 cycles) + (0.25)(6 cycles) + (0.15)(6 cycles) + (0.15)(3 cycles)
= 2.25 + 1.5 + 0.9 + 0.45 = 5.1

MIPS: \[ \frac{2.5 \text{ GHz}}{5.1 \text{ CPI}} \approx \frac{2.5 \times 10^9 \text{ cycles/s}}{5.1 \text{ cycles/inst.}} = 4.90 \times 10^8 \text{ inst./s} = 490 \text{ MIPS} \]

b. Assume that it is possible to increase the frequency of execution to 2.6 GHz by changing the architecture such that beq executes in 4 cycles rather than 3 cycles. Would you recommend that change? Justify your answer.

New CPI: (0.45)(5 cycles) + (0.25)(6 cycles) + (0.15)(6 cycles) + (0.15)(4 cycles)
= 2.25 + 1.5 + 0.9 + 0.6 = 5.25

MIPS: \[ \frac{2.6 \text{ GHz}}{5.25 \text{ CPI}} \approx \frac{2.6 \times 10^9 \text{ cycles/s}}{5.25 \text{ cycles/inst.}} = 4.95 \times 10^8 \text{ inst./s} = 495 \text{ MIPS} \]

The overall MIPS is higher with the higher frequency so yes, I would recommend this change.

c. Assume that it is possible to modify the compiler such that the total number of executed instructions is reduced by 5% but that the new instruction mix is changed to 50%, 20%, 15% and 15% for R-type, lw, sw and beq instructions, respectively. Would you recommend this modified compiler for any of the two machines (the 2.5 and 2.6 GHz machines).

For the 2.5GHz machine,
CPI: (0.5)(5 cycles) + (0.2)(6 cycles) + (0.15)(6 cycles) + (0.15)(3 cycles)
= 2.5 + 1.2 + 0.9 + 0.45 = 5.05
MIPS: \[
\frac{2.5 \text{ GHz}}{5.05 \text{ CPI}} = \frac{2.5 \times 10^9 \text{ cycles/s}}{5.05 \text{ cycles/inst.}} \approx 4.95 \times 10^8 \text{ inst./s} = 495 \text{ MIPS}
\]

For the 2.6GHz machine,
CPI: (0.5)(5 cycles)+ (0.2)(6 cycles)+ (0.15)(6 cycles)+ (0.15)(4 cycles)
= 2.5+1.2+0.9+0.6 =5.20

MIPS: \[
\frac{2.6 \text{ GHz}}{5.20 \text{ CPI}} = \frac{2.6 \times 10^9 \text{ cycles/s}}{5.20 \text{ cycles/inst.}} = 5 \times 10^8 \text{ inst./s} = 500 \text{ MIPS}
\]

With a 5% reduction in the number of instructions (represented as \(x\)), both would be proportionally affected:

For 2.5 GHz machine,
Old exec. time: \[
\frac{(x) \times 5.1 \text{ CPI}}{2.5 \text{ GHz}} \approx (x) \times 2.04 \times 10^{-9} \text{ s}
\]

New exec. time: \[
\frac{(0.95x) \times 5.05 \text{ CPI}}{2.5 \text{ GHz}} \approx (x) \times 1.92 \times 10^{-9} \text{ s}
\]

For 2.6 GHz machine,
Old exec. time: \[
\frac{(x) \times 5.25 \text{ CPI}}{2.6 \text{ GHz}} \approx (x) \times 2.01 \times 10^{-9} \text{ s}
\]

New exec. time: \[
\frac{(0.95x) \times 5.20 \text{ CPI}}{2.6 \text{ GHz}} \approx (x) \times 1.90 \times 10^{-9} \text{ s}
\]

In both cases, the execution time is smaller so yes, I would recommend the modified compiler for both machines

**Question 2: (6 points – 1 for each instruction)**

Determine the format for each instruction and the decimal values of each instruction field for the following program segment:

```
Loop: lw $t1, 48($s1)
lw $t2, 48($s1)
sub $t1, $t1, $t2
sw $t1, 48($s1)
add $s1, $s1, $s2
bne $s1, $zero, Loop
```

Give also the machine code instructions (32 bits per instruction) for that program segment (follow the format given in slide 15 of the first lecture for each instruction). See Appendix A.10 of the textbook for op-codes of operands.
Loop: lw $t1, 48($s1)  //$t1 = Mem[$s1 + 48], I-type format instruction

<table>
<thead>
<tr>
<th>op (6 bits)</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>immediate (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>35</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>M. code</td>
<td>100011</td>
<td>10001</td>
<td>01001</td>
</tr>
</tbody>
</table>

lw $t2, 48($s1)  //$t2 = Mem[$s1 + 48], I-type format instruction

<table>
<thead>
<tr>
<th>op (6 bits)</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>immediate (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>35</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>M. code</td>
<td>100011</td>
<td>10001</td>
<td>01010</td>
</tr>
</tbody>
</table>

sub $t1, $t1, $t2  //$t1 = $t1 - $t2, R-type format instruction

<table>
<thead>
<tr>
<th>op (6 bits)</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>rd (5 bits)</th>
<th>Shamt (5 bits)</th>
<th>Funct (6 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>0</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>M. code</td>
<td>000000</td>
<td>01001</td>
<td>01010</td>
<td>01001</td>
<td>00000</td>
</tr>
</tbody>
</table>

sw $t1, 48($s1)  //Mem[$s1 + 48] = $t1, I-type format instruction

<table>
<thead>
<tr>
<th>op (6 bits)</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>immediate (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw</td>
<td>43</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>M. code</td>
<td>101011</td>
<td>10001</td>
<td>01001</td>
</tr>
</tbody>
</table>

add $s1, $s1, $s2  //$s1 = $s1 - $s2, R-type format instruction

<table>
<thead>
<tr>
<th>op (6 bits)</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>rd (5 bits)</th>
<th>Shamt (5 bits)</th>
<th>Funct (6 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>0</td>
<td>17</td>
<td>18</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>M. code</td>
<td>000000</td>
<td>10001</td>
<td>10010</td>
<td>10001</td>
<td>00000</td>
</tr>
</tbody>
</table>

bne $s1, $zero, Loop  //if ($s1!=0) branch to ‘Loop’, I-type format instruction

<table>
<thead>
<tr>
<th>op (6 bits)</th>
<th>rs (5 bits)</th>
<th>rt (5 bits)</th>
<th>immediate (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bne</td>
<td>5</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>M. code</td>
<td>000101</td>
<td>10001</td>
<td>00000</td>
</tr>
</tbody>
</table>
Question 3: (6 points – 4(a) + 2(b))

In this question, we will augment the signal cycle processor architecture which implements R-type, lw, sw and beq instructions to add two new type of instructions, "R-type-rnr" and R-type-mrr" that are similar to R-type instructions except that for the former, one of the operands is fetched from memory and for the latter, the result is stored in memory:

R-type-mrr $r1, $r2, $r3
R-type-rnr $r1, $r2, $r3

As for the R-type instructions, the operands for the R-type-mrr instructions are in registers $r2 and $r3, while for the R-type-rnr instructions the first operand is in $r2 and the second operand is in memory at the address stored in $r3. Similarly, as for the R-type instructions, the result of the R-type-rnr instructions are stored in register $r1, while for the R-type-mrr instructions, the result is stored in memory at the address stored in $r1.

a. Using this figure, show the addition that you need to add to the data paths to implement the two additional instructions. Produce two new figures (one for each new instruction type), either by modifying the ppt file or just drawing the new data paths and muxes on top of print-outs of the figure. Your new figures should name any new control signals that you need to add.

b. Specify the values of the control signals (the ones already in the figure plus the new ones that you have to introduce) that are needed for the execution of each of the two new instruction types. You do not have to specify the ALUop control signal since the figure uses only one ALUop signal as opposed to the two ALUop1 and ALUOp2 signals specified in Table 4.18 of the textbook.

NOTE that the order of the registers in the assembly language instruction is different from the order in the machine (binary) instruction. Specifically, in a machine instruction, $r2 and $r3 are stored in bits 25-21 and 20-16, respectively, while $r1 is stored in bits 15-11.

NOTE also that I added a third read input/data to the register file in the figure since you will need to read three register values to implement the Rtype-mrr instruction.
For the Rtype-mrr instructions, the modification and the signal paths are given below:

<table>
<thead>
<tr>
<th>Control signal</th>
<th>RegDst</th>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>toMemA</th>
<th>toMemD</th>
<th>MemWrite</th>
<th>MemRead</th>
<th>MemToReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>
For the Rtype-rrm instructions, the modification and the signal paths are given below:

<table>
<thead>
<tr>
<th>Control signal</th>
<th>RegDst</th>
<th>RegWrite</th>
<th>ALUSrc</th>
<th>ALUSrc2</th>
<th>toMemA</th>
<th>MemWrite</th>
<th>MemRead</th>
<th>MemToReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Question 4: (6 points – 2(a) + 1(b) + 2(c) + 1(d))

In this question, we examine how latencies of individual components of the processor datapath affect the clock cycle time. For this question, assume the following latencies for logic blocks in the datapath and assume that blocks not listed have zero latencies:

<table>
<thead>
<tr>
<th>Component</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Mem</td>
<td>200 ps</td>
</tr>
<tr>
<td>Regs read</td>
<td>90 ps</td>
</tr>
<tr>
<td>PC read</td>
<td>5 ps</td>
</tr>
<tr>
<td>Add</td>
<td>100 ps</td>
</tr>
<tr>
<td>Regs write</td>
<td>90 ps</td>
</tr>
<tr>
<td>Mux</td>
<td>5 ps</td>
</tr>
<tr>
<td>D-Mem</td>
<td>200 ps</td>
</tr>
<tr>
<td>ALU</td>
<td>180 ps</td>
</tr>
<tr>
<td>Sign-extend</td>
<td>10 ps</td>
</tr>
</tbody>
</table>

a. Given that the delay for the lw instruction is the longest among the R-type, lw, sw and beq instructions, the delay for lw determines the cycle time. Compute the delay for the lw instruction (and hence the cycle time) for the architecture shown in this figure.

There are 2 parallel paths, that for the PC fetch, and the lw instruction – the latter path is highlighted in green

PC path: PC read + Add + Mux = 5 + 100 + 5 = 110 ps

lw instruction: PC read + I-mem + Regs read + ALU + D-mem + Mux(MemToReg) + Regs write

= 5 + 200 + 90 + 180 + 200 + 5 + 90 = 770 ps is the delay

Note that the Sign Extend (10ps) and Mux-ALUSrc (5ps) path runs in parallel to the Reg reads and hence does not need to be added to the total delay.
b. How will the delay for the lw instructions change after the modifications that you made in Question 3 to add the R-type-rrm and R-type-mrr instructions?

In both cases, only 1 Mux (5ps) has been added to the critical path before the MemRead, and so the delay for both is 775ps.

c. Compute the delays for the new instructions that you added in Question 3 and determine the new cycle time.

Addmrr: PC read + I-mem + Regs read + Mux(ALUSrc) + ALU + Mux(toMemD) + mem-write

= 5 + 200 + 90 + 5 + 180 + 5 + 200 = 685ps

Addrrm: PC read + I-mem + Regs read + Mux(toMemA) + mem-read + Mux(ALUSrc) + Mux(ALUSrc2) + ALU + Mux(MemToReg) + Regs write

= 5 + 200 + 90 + 5 + 200 + 5 + 5 + 180 + 5 + 90 = 785ps

d. Assume that the processor in this figure (without the modifications of Question 3 and with the above logic units) is pipelined by adding buffers between the five stages (as shown in Figure 4.35) and that the delay for reading the buffers is 5ps and for writing into the buffers is also 5ps. What is the pipeline cycle time?

Note that you can either read or write (but not read and write simultaneously) into the register file in 90ps. (Figure 4:35 shown below)
The pipeline consists of 5 stages, the times for which are broken down below:

IF stage: PC read + I-mem + Buffer write = 5 + 200 + 5 = 210ps  
(PC read + add + mux + PC write = 5 +100+5+5 = 115 runs in parallel)

ID stage: Buffer read + Regs read + Buffer write = 5 + 90 + 5 = 100ps  
(SignExtend-10ps- runs in parallel)

EXE stage: Buffer read + Mux + ALU + Buffer write = 5 + 5 + 180 + 5 = 195ps  
(shift + add = 0 + 100 = 100 runs in parallel)

MEM stage: Buffer read + D-mem + Buffer write = 5 + 200 + 5 = 210ps

WB stage: Buffer read + Mux + Regs write = 5 + 5 + 90 = 100ps

The pipeline cycle time is 210ps, which is determined by the longest stage. Note that during a 210ps cycle, the Regs write and the Regs read can be serialized so that the Regs write will occur during the first 100ps of the cycle, followed by the Regs read (from time 100 – 190 ps) and finally the Buffer write, which will complete at 195ns.

NOTE: Think about the cycle time if each of Regs read and Regs write would take 110ps, rather than 90ps.