1. (20 pts) Consider the pipelined SPIM datapath with forwarding and branches are predicted as not taken. Assume branch instructions occur 10% of the time and are taken 45% of the time with a penalty of 3 cycles. With forwarding, the load delay slot is one cycle and can be filled 60% of the time with useful instructions. 25% of the instructions are loads and 30% of these introduce load delay hazards.

   a. (10 pts) What is the increase in CPI due to load delay slots and branch hazards?

b. (10 pts) With the goal of reducing execution time, we wish to pipeline all memory accesses to 3 cycles. This will reduce the clock cycle time by 15%. Is this a good idea? Provide a quantitative justification for your answer.
2. (40 pts) Consider the pipelined SPIM datapath shown at the end of this question.
   
a. (10 pts) Show the additions to the datapath that would be necessary to correctly implement forwarding. Specifically, i) identify the locations of the forwarding muxes on the figure (clearly!), ii) list the specific inputs to each forwarding mux below (i.e., signal names), and iii) identify the location on the datapath that is the source of each signal. Full credit requires that I can read and understand your modifications.

b. (10 pts) Write the equation for the forwarding logic for the Rs input to the ALU. Make sure that your notation makes clear the signals being used and the stage they are from.
c. (10 pts) Now consider this datapath after augmenting it with forwarding and appropriate hazard detection logic, executing the following instruction sequence. Register $12 and $13 are initialized to 0x1028 and 0x4004 respectively. The first instruction is at 0x40040000. Assume all other registers are properly initialized. Further consider the state of the data path during cycle 7 (first cycle is 0). What are the values of the datapath elements below?

```
loop:
lw $7, 4($12)
lw $8, 4($13)
add $9, $8, $7
sw $9, 4($13)
addi $12, $12, 4
addi $13, $13, 4
addi $5, $5, -1
bne $5, $0, loop
```

<table>
<thead>
<tr>
<th>IF/ID.PC</th>
<th>MEM/WB.RegWrite</th>
<th>EX/MEM.ALUResult</th>
<th>EX/MEM.BranchAddress</th>
<th>WB/ID.MemToReg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

d. (5 pts) On what cycle does bne exit the pipeline in part c. the first time through the loop.

e. (5 pts) If 10% of all instructions were branches, what would be the maximum impact on CPI of having a perfect branch predictor, i.e., one that was correct 100% of the time?
3. **(25 pts)** Consider the following code segment executing on the pipelined datapath (from Problem 2, i.e., no forwarding or hazard detection). In ID, writes occur in the first half of the cycle and reads in the second half.

```
1. loop:   lw $t3, 0($t0)    # load first element
2.       add $t2, $t2, $t3  # update sum
3.       sw $t2, 1028($t0) #store partial sum
4.       addi $t0, $t0, 4  # point to next word
5.       addi $t1, $t1, -1 # decrement count
6.       bne $t1, $zero, loop # check if done
```

a. **(10 pts)** Insert `nops` to ensure this code executes correctly on this datapath. Show the code with `nops`.

b. **(10 pts)** Shorten the execution time by re-scheduling any instructions to remove any `nops`. Show the rescheduled code.
c. (5 pts) What is the speedup achieved of executing a single unscheduled loop iteration on the pipelined datapath relative to executing a single loop iteration on a multi-cycle datapath? Assume the same clock frequency and ignore pipeline setup time.
4. **(15 pts)** Consider a pipelined SPIM core with full support for forwarding and load-to-use hazard detection between instructions within a thread. Now consider execution of two threads on this core where each thread executes the following function but over distinct data sets.

```assembly
func:    move $v0, $a0     #get array base address
         move $v1, $a1      #get array count
         add $v1, $v1, $v0  #array ending address
         move $t1, $0      #initialize loop sum

loop:   lw $s6, 0($v0)      #load first value
        add $t1, $t1, $s6    #update sum
        addi $v0, $v0, 4     #update pointer
        slt $t0, $v0, $v1     #check for termination
        bne $t0, $0, loop   #onto next element
         move $v0, $t1        #pass sum back to main
         jr $ra             #return to main
```

(a. **(5 pts)** With fine-grained, instruction level hardware multithreading and two threads (T1 and T2) what instruction from each thread, is in each stage of the pipeline on cycle 11 (the first cycle is cycle 0). Assume each instruction is a native instruction.

```
IF   ______________
ID   ______________
EX   ______________
MEM  ______________
WB   ______________
```

(b. **(5 pts)** Assuming branches are tested in EX and the branch condition is available in MEM (as in our base pipeline), how many hardware threads do you need with fine grained multithreading before branch penalties can be completely hidden? Explain.


c. **(5 pts)** What is the difference between thread level parallelism (TLP) and instruction level parallelism (ILP)?