Please fill out Name and S# and attach to the first page of your homework.

NAME: First:__________________ Last:_____________________

S#:_____________________

TOTAL SCORE:           /280

Problem 1 (20pts).
Problem 4.1 in PH.

Problem 2 (20pts).
Problem 4.2 in PH.

Problem 3 (20pts).
Problem 4.3 in PH.

Problem 4 (20pts).
Problem 4.5 in PH. Use both loop unrolling and code scheduling to find a solution with no stall in the body of the loop --even though there may be stalls after the body.

Problem 5 (100pts).
This question is on the scoreboard processor covered in class. Answer the following questions.
1. Look at Fig. 4.7 page 250 in PH. There may be a subtle problem with WAR hazards. Consider an instruction I in the Read Operand stage waiting for operand Fj while its operand Fk is available (i.e. Rj=0 but Rk=1). Assume that Fj is produced by an instruction preceeding I and currently in FU1. Now assume that the instruction following I modifies Fj; assume that this instruction can execute and completes before the instruction in FU1. This instruction will see Rj =0 and thus will not wait in “Write results”. When the instruction in FU1 completes, instruction I gets its results, however, it overwrites the value of Fj written by the instruction following I. This the value in register Fj is now stale. Explain why this cannot happen.

2. We propose to add register renaming to the scoreboard processor. Instead of 32 physical registers we have now 64 physical registers. The issue logic maintains a table with 32 entries, one for each floating point register in the instructions. Each entry of that table contains 6 bits to point to one physical register. Every time an instruction modifying a register --say register Fj-- goes through the issue stage, the issue stage allocates one physical register --say register Pk-- to Fj and then modify the destination field of the instruction to point to Pk, so that Pk will now become the physical destination of the instruction. As following instructions reading Fj are issued the issue stage also changes the operand field of these instructions.
to Pk, so that the correct value is read.
Each pointer to a physical register also contains one counter. The counter for Fj is incremented everytime an instruction reading Fj goes through the issue logic and is decremented everytime an instruction reading Fj goes through the Write result stage. When the counter reaches zero, then the value in Pk has been read by all instructions that needed it and Pk can be reallocated to another register Fi. When the counters of all the physical registers are non-zero and an instruction needs to allocate a new one, the instruction remains in the issue logic until one physical register is free.

As it is this scheme will not work. What should be done to make it work??

3. With this addition, specify the functionality of a new scoreboard, similar to the table in Fig. 4.7 of PH for this new processor. Note that register renaming has eliminated WAR and WAW hazards and that the scoreboard must now manage the allocation/deallocation of physical registers. Thus the new table may be quite different from the one in Fig. 4.7.

**Problem 6 (100pts).**

Problem 4.14 parts (a) (c) and (d) in PH. Please assume the following modifications:

1. In all cases assume that functional units are not pipelined, but that there are enough functional units to avoid any stalls due to structural hazards on functional units.
2. Use the latencies shown in Figure 4.63 for the 5-stage DLX. In the case of scoreboard and Tomasulo these latencies mean that the execution time of a FPMPY is 7 clocks, that the execution time of FPADD is 5 clocks and that any integer instruction execute in one clock.
3. For the classic 5-stage DLX assume that all functional units are fully pipelined and bypassed (forwarded). If contention might exist for the WB stage for two consecutive then the second instruction is not issued until it is known in the issue logic that the contention will not happen.
4. For the scoreboard DLX, use the design specified in class (however, now the number of FUs is infinite, the number of entries in the window is also large enough and the latencies of Fig. 4.63 apply.
5. For the Tomasulo DLX use enough reservation stations so that structural stalls for these resources do not occur. Use the design as specified in class.
6. In all cases, show the schedule by filling the instruction status table as was used in class. Each entry in the table shows the clock cycle when a given instruction went through a specific phase of execution. Show the content of the table when the sgti instruction write its results.
7. In all cases give the execution time from T=0 to the cycle when sgti writes its result.
8. To show the schedule, you simply have to make 3 copies of the table in Figure 1 and fill the table for each of the 3 processors (do not show all the status tables as asked in the problem statement).
Table 1:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Exec. complete</th>
<th>Write Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F2,0(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTD F4,F2,F0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F6, 0(R2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F6,F4,F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD 0(R2),F6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI R1,R2,#8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI R2,R2,#8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGTI R3,R1,done</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>