(1) (10) What are Benchmarks, and why are they important? List the 4 classes of benchmarks, give an example for each class, and tell me what the perfect benchmark would be.

(2) (4) Why do most high-performance machines used a fixed instruction size? (What is the difficulty that variable-length instructions present?)

(3) (4) The book lists many Compiler Optimizations. List 4 of them.

(4) (6) What are the advantages and disadvantages of stack machines?

(5) (8) There are 3 kinds of Hazards, Structural and two others. List the other two, give brief descriptions of all three, and some ways to alleviate the impact of each on processor performance.

(6) (8) Why is branch prediction important? What performance enhancing techniques have made it so? List 3 examples of existing Branch Prediction strategies in order of their (average) effectiveness.

(7) (17) Suppose we are considering two alternatives for our conditional branch instructions, as follows:

M1 - A compare is included in the branch.

M2 - A condition code is set by a compare instruction and followed by a branch that tests the condition

On both machines, the conditional branch instruction takes 2 cycles, and all other instructions take 1 cycle. On M2, 20% of all instructions executed are conditional branches. Because M2 does not have the compare included in the branch, the clock cycle is 1.25 times faster than M1.

a. Which design is faster?

b. What if M2 was only 1.2 times faster than M1?

(8) (18) Suppose the branch frequencies (as percentages of all instructions) are as follows:

Conditional branches, 20%
Jumps and Calls, 5%
Conditional branches, 60% are taken

Assume the machine in question has a 4-deep pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Given that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?

(9) (25) Time to hand-execute Tomasulo's algorithm! On the table provided, fill in the appropriate boxes. I have provided a few answers - for example, I have indicated where memory returns values, and I have show (time 0) the format I would like you to use in the table. Basically, if a register is waiting for a result, you should mark it busy ("B") and write down which Functional Unit it is waiting for. FLR0 and FLR8, for example, show that instructions previous to time 1 wrote to these two registers. In the case of FLR0, it is waiting for the output of functional unit 12. FLR8 is waiting for the output of FU 10. Note that at time 0 the "12" tag appears on the CDB, and at time 1 the busy flag in FLR0 is cleared. You should adopt this same convention in your answer. Also, the entries for the functional units should indicate which operand is Valid ("V") and if it is not valid, which FU it is waiting for. Note the operation of the floating point adder during times 0, 1 and 2 for clarification on how you should mark your multipliers. (At time 0 the adder has 1 valid input and is waiting for the output of FU12, at time 1 it has grabbed 12 off the CDB and is executing, and at time 2 it finishes and writes to the CDB.)

You are to execute the following loop:

```
UP:  LD    F2,0(R1)
     MULT  F6,F0,F2
     SD    0(R1),F6
     JMP   UP
```

The multiplier takes 8 cycles - as an extra bonus, I have indicated on the chart when the multiply completes. You are to give precedence to the multiply if there happens to be a CDB structural hazard ...

```
UP:     LD      F2,0(R1)
        MULT    F6,F0,F2
        SD      0(R1),F6
        JMP     UP
```

Time →	0	1	2	3	4	5	6	7	8	9	10	11	12
Decoder	*NOP*	*LD*	*MULT*	*SD*	*JMP*	*LD*	*MULT*	*SD*	*JMP*	*LD*	*MULT*	*SD*	*JMP*
LBuff (1)				**Valid**				**Valid**				**Valid**	
LBuff (2)													
LBuff (3)													
i+ L,R (7)													
i+ L,R (8)													
f+ L,R (10)	**V,12**	**V,V**	**Valid**										
f+ L,R (11)													
f* L,R (12)	**Valid**											**Valid**	
f* L,R (13)													
f* L,R (14)													
CDB Tag	**12**		**10**	**1**				**1**					
FLR0	**B,12**												
FLR2													
FLR4													
FLR6													
FLR8	**B,10**	**B,10**	**B,10**										
FLR10													
SDB1													
SDB2													
SDB3													