Computer Science 418 Midterm

February 26, 1998

Name (1 point):____

Student ID:____

This exam is **closed-book, closed-notes**. You may have one 8.5 by 11 inch sheet of notes, and you may use a calculator. The exam ends at 9:55am. **Good luck!**

(Leave this table blank.)

Name (1 pt)	
Buzzwords (40 pts)	
Superscalar Execution (20 pts)	
Precise Exceptions (10 pts)	
Branch Prediction (10 pts)	
Caching (10 pts)	
Putting It All Together (10 pts)	
Total (out of 100 pts)	

1 Buzzwords (40 pts)

For each word or phrase, choose the best definition (on next page). You get 2 points for each correct answer, 1 point for each **blank**, and 0 points for each wrong answer. I've done the first one to get you started.

Cost Related to Die Area	
TLB	
CPI	
BTFN	
Control Hazard	
Data Hazard	
Structural Hazard	
Moore's Law	
Amdahl's Law	
Wall Clock Time	
Speedup	
Static Scheduling	
Dynamic Scheduling	
Local Scheduling	
Basic Block	
Non-Blocking Cache	
Multiported Cache	
Exclusive Cache	
Victim Cache	
Bandwidth	
Fill Frequency	

1. Speedup $= 1/(1 - f_e + f_e/s_e)$ where f_e is the fraction of the program that can use an enhancement, and s_e is the speedup from using the enhancement.
2. A branch prediction strategy that predicts that backward branches are always taken, and forward branches are never taken.
3. A cache that can handle multiple read or write operations at a time.
4. A cache that can handle other operations while waiting for a miss.
5. A lower-level cache (e.g. L2) that doesn't always contain all items in the next higher level cache (e.g. L1).
6. A set of contiguous instructions with no branches into or out of them.
7. A small cache of recent virtual-to-physical page translations.
8. A small cache that holds cache lines evicted because of conflict misses.
9. Bandwidth divided by granularity.
10. Cost is $O(\text{Die Area}^4)$
11. Having the compiler schedule instructions to maximize ILP.
12. Information divided by time.
13. Scheduling instructions at runtime.
14. Scheduling instructions within a single basic block.
15. The new execution time divided by the old execution time.
16. The number of clock cycles require to execute an instruction.
17. The number of transistors on a chip doubles every 18 months.
18. The total amount of time elapsed to execute a program.
19. When ILP is reduced because one instruction depends on data from another.
20. When ILP is reduced because we cannot determine which instruction to execute next.
21. When ILP is reduced because we don't have sufficient hardware to execute two instructions in parallel.

2 Superscalar Execution (20 pts)

Consider a superscalar processor with the following specifications:

- The instruction fetcher can fetch two instructions per clock cycle. The instruction buffer can hold at most two instructions.
- The instruction decoder/dispatcher can process two instructions per clock cycle, handling register renaming using a mapping table and reorder buffer (ROB) and placing appropriate entries in the reservation station.
- The reservation station can issue up to four operations per clock to the functional units. Reservation station slots are freed as soon as the operation issues to a functional unit.
- There are six functional units in total: two identical integer units, two identical floating point units, a memory unit, and a branch unit. The integer units can compute addition, subtraction, multiplication and division in a single cycle. The floating point units compute addition and multiplication in 3 cycles. Division requires 6 cycles. The memory and branch units require 1 cycle each. The memory unit has a memory address register (MAR) which contains the address to be loaded/stored, and a memory data register (MDR) which contains the value to loaded/stored. None of the functional units are pipelined. However, results are bypassed directly from the functional units to the reservation station, so a dependent operation can issue on the cycle immediately after the result it is waiting for is computed (as was done in class and homework).
- The writeback unit can commit up to 3 ROB entries per clock.
- There are 4 ROB slots and 4 reservation station slots in total.

For the following code sequence, show the execution over the next 5 clock cycles. You may assume that the branch is predicted correctly.

```
Loop:   LD      F2, 1000(R0)    -- F2 = memory[1000+R0]
        FADD    F0, F0, F2      -- F0 += F2
        FMUL    F2, F2, F2      -- F2 *= F2
        FADD    F1, F1, F2      -- F2 += F2
        ADDI    R0, R0, #8      -- R0 += 8
        ADDI    R1, R1, #1      -- R1++
        BNE     R0, #8000, Loop -- if R0==8000 goto Loop:
```

Initially, the processor data structures are empty, and all registers are equal to zero. Memory location 1000 contains the value 5.

Use the next five pages to work this problem!

Clock 1: I've done this step as an example:

Register Values				
R0:	R1:	F0:	F1:	F2:

Instruction Buffer

Reg. Map Table	
Reg.	Maps to
R0	
R1	
F0	
F1	
F2	

Reorder Buffer			
	Dest	Value	Done?
rob1			
rob2			
rob3			
rob4			

Reservation Station							
Operation	Src 1	Value	Valid?	Src 2	Value	Valid?	Dest

Functional Units:
For each functional unit that isn't idle, indicate the operation it is performing, what the operands are, and the destination for the result. For the memory unit, give the contents of MAR and MDR:

Clock 2:

Register Values				
R0:	R1:	F0:	F1:	F2:

Instruction Buffer

Reg. Map Table	
Reg.	Maps to
R0	
R1	
F0	
F1	
F2	

Reorder Buffer			
	Dest	Value	Done?
rob1			
rob2			
rob3			
rob4			

Reservation Station							
Operation	Src 1	Value	Valid?	Src 2	Value	Valid?	Dest

Functional Units:
For each functional unit that isn't idle, indicate the operation it is performing, what the operands are, and the destination for the result. For the memory unit, give the contents of MAR and MDR:

Clock 3:

Register Values				
R0:	R1:	F0:	F1:	F2:

Reg. Map Table	
Reg.	Maps to
R0	
R1	
F0	
F1	
F2	

Reorder Buffer			
	Dest	Value	Done?
rob1			
rob2			
rob3			
rob4			

Instruction Buffer

Reservation Station							
Operation	Src 1	Value	Valid?	Src 2	Value	Valid?	Dest

Functional Units:
For each functional unit that isn't idle, indicate the operation it is performing, what the operands are, and the destination for the result. For the memory unit, give the contents of MAR and MDR:

Clock 4:

Register Values				
R0:	R1:	F0:	F1:	F2:

Instruction Buffer

Reg. Map Table	
Reg.	Maps to
R0	
R1	
F0	
F1	
F2	

Reorder Buffer			
	Dest	Value	Done?
rob1			
rob2			
rob3			
rob4			

Reservation Station							
Operation	Src 1	Value	Valid?	Src 2	Value	Valid?	Dest

Functional Units:
For each functional unit that isn't idle, indicate the operation it is performing, what the operands are, and the destination for the result. For the memory unit, give the contents of MAR and MDR:

Clock 5:

Register Values				
R0:	R1:	F0:	F1:	F2:

Instruction Buffer

Reg. Map Table	
Reg.	Maps to
R0	
R1	
F0	
F1	
F2	

Reorder Buffer			
	Dest	Value	Done?
rob1			
rob2			
rob3			
rob4			

Reservation Station							
Operation	Src 1	Value	Valid?	Src 2	Value	Valid?	Dest

Functional Units:
For each functional unit that isn't idle, indicate the operation it is performing, what the operands are, and the destination for the result. For the memory unit, give the contents of MAR and MDR:

3 Precise Interrupts (10 pts)

In the preceding problem, suppose that the multiplication throws an exception (e.g. overflow). Briefly describe what happens in the processor in order to give a precise interrupt. What happens to the values in the reorder buffer, reservation station, etc.? What values do the registers (R0, R1, F0, F1, F2) have?

4 Branch Prediction (10 pts)

Assume register R0 equals 4, and register R1 equals 6. Consider this code fragment:

```
L1:     beq     R0, R1, exit:    -- if R0==R1 goto exit
L2:     sub     R1, R1, R0       -- R1 -= R0
L3:     bgt     R1, R0, L2       -- if R1>R0 goto L2
L4:     addi    R2, R1, #0       -- R2 = R1
L5:     addi    R1, R0, #0       -- R1 = R0
L6:     addi    R0, R2, #0       -- R0 = R2
L7:     br      L1               -- goto L1
exit:
```

Suppose we have a 2-level branch predictor, with a single 3-bit history register for all branches (not including unconditional ones), and a single global pattern table for all branches (not including unconditional ones). The history register is initially to 000 (last three branches not taken), and the pattern table is initially 10 (weakly predict taken) for all patterns. Assume the unconditional branch L7 is always predicted correctly and doesn't interact with the branch prediction at all.

For each branch encountered until this code exits, indicate what the prediction is, whether the prediction is correct, and the updated values of the history register and the pattern table. I've done the first branch for you as an example.

Branch	Prediction	Correct?	History Register	Pattern Table 000	001	010	011	100	101	110	111
Initial			000	10	10	10	10	10	10	10	10

What is the branch prediction rate (including the unconditional branches)?

5 Caching (10 pts)

Consider a machine with 4-bit addresses. Each address specifies an 8-bit byte. The L1 cache is direct-mapped, with each line holding 1 byte, and total capacity of 2 bytes. The L2 cache uses **exclusive** caching with the L1 cache. It is also direct-mapped, with each line holding 1 byte, and total capacity of 4 bytes.

In main memory, the byte at address i contains the value $8i$. The caches are initialized as follows (Data values are in decimal.):

L1 Cache		
Line	Tag	Data
0	000	0
1	000	8

L2 Cache		
Line	Tag	Data
00	01	32
01	01	40
10	01	48
11	01	56

Show the caches after each of the following requests is executed (one after another).

Read location 0100:

L1 Cache		
Line	Tag	Data
0		
1		

L2 Cache		
Line	Tag	Data
00		
01		
10		
11		

Read location 1001:

L1 Cache		
Line	Tag	Data
0		
1		

L2 Cache		
Line	Tag	Data
00		
01		
10		
11		

Read location 0111:

L1 Cache		
Line	Tag	Data
0		
1		

L2 Cache		
Line	Tag	Data
00		
01		
10		
11		

6 Putting It All Together... (10 pts)

Your are in charge of designing the follow-on to the highly successful Foobar 1000 microprocessor. The Foobar 1000 was a superscalar processor with functional units as described in Problem 2. In the Foobar 2000, you have already decided to allocate enough reservation station and ROB slots that these resources will never constrain your execution. Your remaining design decisions are:

1. Whether to add a two-level branch predictor, which would increase die size by 5% (over the Foobar 1000). The Foobar 1000 uses BTFN.
2. Whether to add two more integer units, which would increase die size by 10%.
3. Whether to add two more floating point units, which would increase die size by 20%.
4. Whether to add two more floating point adders/multipliers (which can't divide), which would increase die size by 15%.
5. Whether to add two pipelined floating point adders/multipliers (which can't divide), which would increase die size by 20%. These units would still have a 3 cycle latency, but they can start a new operation every cycle.

The area penalties are additive, i.e., you could have branch prediction, two additional integer units, and two additional floating point units for a die size that is 35% larger than the Foobar 1000. However, you can add at most two additional floating point units (so the above choices for floating point are exclusive).

Fortunately, your customers care about performance only on code like the following:

```
for (i=0; i<LENGTH; i++) {
        result[i] = (A[i]+B[i]*wt)/scale;
}
```

where i is an integer, LENGTH is very, very large, and all other variables are floating point.

What design choice gives you the best price/performance? Justify your answer. What is the approximate speedup over the Foobar 1000? Partial credit will be given for partial solutions.