Computer Science 418 Midterm

February 25, 1999

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 9:50am. **Good luck!**

(Leave this table blank.)

Name (1 pt)	
Buzzwords (40 pts)	
Superscalar Execution (15 pts)	
Precise Exceptions (10 pts)	
Branch Prediction (15 pts)	
Caching (10 pts)	
Cost Model (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.

User Time	24
Static Scheduling	
Dynamic Scheduling	
Local Scheduling	
Basic Block	
Non-Blocking Cache	
Amdahl's Law	
Speedup	
Multiported Cache	
Victim Cache	
Bandwidth	
Fill Frequency	
FPM	
TLB	
BTFN	
BTAC	
Moore's Law	
CPU Time Equation	
Control Hazard	
Data Hazard	
Structural Hazard	

1. Scheduling instructions at runtime.
2. Scheduling instructions within a single basic block.
3. Memory interface scheme in which multiple column addresses can be requested for each row address given.
4. A cache of the target addresses that branches have taken recently.
5. CPU Time = IC $\times$ CPI $\times$ Clock Period.
6. 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.
7. Cost is $O(\text{Die Area}^4)$
8. A branch prediction strategy that predicts that backward branches are always taken, and forward branches are never taken.
9. A cache that can handle multiple read or write operations at a time.
10. A cache that can handle other operations while waiting for a miss.
11. A small cache of recent virtual-to-physical page translations.
12. A small cache that holds cache lines evicted because of conflict misses.
13. A set of contiguous instructions with no branches into or out of them.
14. Bandwidth divided by granularity.
15. Information divided by time.
16. Having the compiler schedule instructions to maximize ILP.
17. The old execution time divided by the new execution time.
18. The number of clock cycles required to execute an instruction.
19. The number of transistors on a chip doubles every 18 months.
20. When ILP is reduced because one instruction depends on data from another.
21. When ILP is reduced because we cannot determine which instruction to execute next.
22. When ILP is reduced because we don't have sufficient hardware to execute two instructions in parallel.
23. The total amount of time elapsed to execute a program.
24. The time that a program spends runnings the user's program, in user mode.

2 Superscalar Execution (15 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 four 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.
- Reorder buffer entries are allocated and retired in increasing numerical order with wrap-around, e.g., rob2 gets allocated just before rob3 does, rob7 gets allocated just before rob0 does.
- 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 four arithmetic functional units: (We will ignore branch processing and memory access for this problem.) two identical integer units, a floating point add/multiplier, and a floating point divide/square-root. The integer units can compute addition, subtraction, multiplication and division in a single cycle. The floating point add/multiplier computes addition, multiplication and assorted other operations (such as truncation, incrementing, etc.) in 3 cycles. It cannot do division or square root. The divide/square-root unit requires 10 cycles to do either a division or square root. The functional units are not pipelined.
- The writeback unit can commit up to 3 ROB entries per clock.
- There are 8 ROB slots and 4 reservation station slots in total.

On the following page, I have given you the current state of the processor. You may assume that the next instructions to be fetched by the processor are as follows:

```
...
FADD    F0, F0, F2      -- F0 += F2
FMUL    F2, F2, F2      -- F2 *= F2
ADDI    R0, R0, #8      -- R0 += 8
ADDI    R1, R1, #8      -- R1 += 8
ADDI    R2, R0, R1      -- R2 = R0+R1
...
```

Your task is to show the state of the processor at the next clock cycle.

Present State: This is the current state of the processor:

Register Values									
R0:	17	R1:	23	F0:	0.1	F1:	2.3	F2:	4.5

Instruction Buffer (Higher is older.)
`ADD R0,R0,R1 # R0 += R1`

Reg. Map Table	
Reg.	Maps to
R0	rob1
R1	rob2
F0	rob4
F1	F1
F2	rob0

Reorder Buffer			
	Dest	Value	Done?
rob0	F2	?	No
rob1	R0	?	No
rob2	R1	?	No
rob3	F0	?	No
rob4	F0	?	No
rob5			
rob6			
rob7	F2	8.0	Yes

Reservation Station							
Operation	Src 1	Value	Valid?	Src 2	Value	Valid?	Dest
ADD	rob1	?	No	rob2	?	No	rob3
FADD	imm	1.0	Yes	rob7	8.0	Yes	rob4
FDIV	imm	1.0	Yes	rob7	8.0	Yes	rob0

Functional Units:

Integer ALU1: adding 3 and 2, result to rob1

Integer ALU2: (idle)

FADD/FMUL: Float-to-int conversion of 6.02×10^{23}, result to rob2, cycle 1 of 3.

FDIV/FSQRT: (idle)

Next State: Fill in this page to show what happens after one clock tick:

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

Instruction Buffer (Higher is older.)

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

Reorder Buffer			
	Dest	Value	Done?
rob0			
rob1			
rob2			
rob3			
rob4			
rob5			
rob6			
rob7			

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

Functional Units:

Integer ALU1:

Integer ALU2:

FADD/FMUL:

FDIV/FSQRT:

3 Precise Interrupts (10 pts)

In the preceding problem, suppose that the float-to-int conversion notices an overflow in the present state (the state I gave you, not the one you worked out) and throws an exception (e.g. overflow). If the processor has precise interrupts, what are the values of the registers that the interrupt handling routine will see? (In other words, what is the precise state that gets restored when the interrupt happens?)

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

4 Branch Prediction (15 pts)

Assume register R0 equals 12, register R1 equals 8, register R2 equals 0, and register R3 equals 3. Consider this code fragment:

```
L1:     sub     R0, R0, R1      -- R0 -= R1
L2:     bge     R0, L5          -- if R0>=0 goto L5
L3:     add     R0, R0, R1      -- R0 += R1
L4:     subi    R2, R2, #1      -- R2--
L5:     addi    R2, R2, #1      -- R2++
L6:     add     R2, R2, R2      -- R2 += R2
L7:     add     R0, R0, R0      -- R0 += R0
L8:     subi    R3, R3, #1      -- R3--
L9:     bge     R3, L1          -- if R3>=0 goto L1
```

Suppose we have a 2-level branch predictor, with a 3-bit history register for each branch, and a single global pattern table for all branches. The history registers and pattern table have the initial values shown.

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 registers and pattern table. I've done the first branch for you as an example.

			History Reg.		Pattern Table							
Line	Predict	Correct?	L2	L9	000	001	010	011	100	101	110	111
Initial	—	—	000	101	00	10	11	01	00	10	11	00
L2	NT	No	100	101	01	10	11	01	00	10	11	00

What is the branch prediction rate?

5 Multiported Banked Caching (10 pts)

Consider a machine with 8-bit addresses. Each address specifies an 8-bit byte. The L1 cache is divided into four direct-mapped banks. Each cache line holds 1 byte, and the total capacity of the cache is 32 bytes.

Show how an 8-bit address is divided (and how many bits in each field) into the bank select, the byte offset, the cache line index, and the tag.

Given the following stream of memory references, group them into the largest sets that the above cache could handle simultaneously. Assume that memory operations are not allowed to be performed out-of-order, and you must respect data dependencies. You may ignore the issue of whether the cache hits or misses on the requests. (These are 8-bit addresses in decimal.)

`read 64, write 65, read 66, write 67, read 68, read 71, write 67, read 66, read 130, read 34.`

For example, if you think the above should be performed two at a time, you could group them as: (read 64, write 65), (read 66, write 67), (read 68, read 71), (write 67, read 66), (read 130, read 34).

Suppose the machine implemented a 4-ported cache by making four identical copies of the data instead of four banks. Again, suppose you only have room for 32 bytes of total data storage, and that the cache is direct-mapped. What is the effective cache size (as visible to the user, i.e. how many bytes of memory can be cached)? In this case, how would an 8-bit address be divided (and how many bits in each field) into the bank select, the byte offset, the cache line index, and the tag? What would be the best grouping of the above stream of memory references?

6 Cost Model (10 pts)

You are in charge of manufacturing the high volume Foobium III processor. The die measures 1cm by 1cm square. Your current process uses 20cm wafers, which cost \$3000 each, and your process engineers are reporting a defect density of 0.8 defects per cm^2. You can assume a wafer yield of 100%, and that $\alpha = 3$.

Using the cost model presented in class, what is your cost per die? (This ignores testing and packaging costs.)

Prof. Alan Schmoo at UBC has invented a revolutionary process that creates perfectly square wafers that measure 20cm by 20cm. If you moved production of the Foobium III to the new square wafers, the cost of each wafer goes up to \$3820. Assuming identical defect density, wafer yield, and α as in the old process, derive the cost per die if using the Schmoo Square Wafers.