Computer Science 418 Final

April 24, 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 6:00pm. **Good luck!**

(Leave this table blank.)

Name (1 pt)	
Buzzwords (40 pts)	
Short Answers (15 pts)	
Predication (15 pts)	
Newton's Method (10 pts)	
Pipeline Gating (20 pts)	
Total (out of 100 pts)	

1 Buzzwords (40 pts)

For each phrase, formula, or definition, fill in the best name or buzzword. I've done the first one to get you started. (2pts each)

- Parallelism between individual assembly language instructions.

  ____

- 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.

  ____

- Processor speed (or transistor density) doubles every 18 months.

  ____

- $x_{i+1} = x_i - f(x_i)/f'(x_i)$

  ____

- A small cache that holds entries evicted from another cache

  ____

- An architecture style in which each instruction tells what every one of the functional units in the processor are doing in parallel.

  ____

- A maximal contínguous sequence of instructions with no branches out of or into it.

  ____

- A branch prediction scheme in which the branch history is used to index a pattern table, which gives a prediction for that pattern.

  ____

- Intel and HP's proposed joint 64-bit instruction set architecture.

  ____

- The codename of the first processor to implement Intel and HP's proposed joint 64-bit instruction set architecture.

  ____

- A load instruction that does not generate any errors, even if the load fails.

- Global code scheduling based on optimizing one path of execution through the code.

- The old execution time divided by the new execution time.

- A cache in which a given line of memory has several locations where it may be cached.

- Guaranteeing that when an interrupt or exception occurs, that all instructions before the one with the exception have completed, all instructions after the one with the exception haven't started, and the PC points to the instruction with the exception.

- A small cache that gives the next instruction if a branch is predicted taken.

- A small cache of virtual-to-physical page translations.

- A cache that is able to service additional requests while it is waiting for a miss to be serviced.

- A cache that can handle multiple reads or writes at the same time.

- Moving an instruction from below a conditional branch to above it, so that it executes regardless of which way the branch goes.

- Making multiple copies of a loop body, such that one iteration through the new loop does the work of several iterations of the original loop.

---

2 Short Answers (15 pts)

Keep your answers short and to the point. Points will be deducted for excessively lengthy or extraneous answers. (For example, if I ask for two reasons, and you can think of five, pick out what you think are your best two and write them down. Don't list all five.)

- What number (give your answer in decimal) does the 8-bit binary pattern 11110000 represent if we interpret it as an unsigned number?

  ____

  As a 2s complement signed number?

  ____

  As a floating point number, using signed magnitude, a four-bit exponent field with a bias of 7, and a normalized fraction field with value between 1 and 2 and with the leading 1 hidden?

  ____

- Compute the arithmetic and geometric means of the numbers 1, 1, 8, 32.

  Arithmetic Mean:______________________________

  Geometric Mean:______________________________

- Give three reasons that make it hard to achieve high performance with x86 floating point.

  ____

  ____

  ____

- Give two advantages of asynchronous design over synchronous design.

  ____

  ____

  And an advantage of synchronous design over asynchronous design?

  ____

- Give two advantages of static code scheduling over dynamic code scheduling.

  ____

  ____

  Give an advantage of dynamic code scheduling over static code scheduling.

  ____

- Give a good reason for defining the instruction set architecture independently of the specific implementation.

  ____

3 Predication and Scheduling (15 pts)

Consider a superscalar processor with two integer ALUs, a memory load/store unit, and a branch unit. All instructions take one cycle to complete. However, if a conditional branch is mispredicted, there is a two cycle penalty until instructions along the correct path start to execute.

Consider the following sequence of code:

```
L1:     BGE     R1, 0, L5       -- if R1>=0 goto L5:
L2:     ADD     R1, R1, R2      -- R1 += R2
L3:     ADDI    R3, R3, #1      -- R3++
L4:     BR      L7
L5:     SUB     R1, R1, R2      -- R1 -= R2
L6:     SUBI    R3, R3, #1      -- R3--
L7:     ADD     R1, R1, R1      -- R1 += R1
L8:     ADD     R3, R3, R3      -- R3 += R3
```

Draw the control flow graph of this code. Indicate which statements are in each basic block.

For each combination of whether L1 was predicted taken or not-taken, and whether the prediction was correct or not, show the execution schedule of the instructions in the processor. You may assume an unlimited number of reorder buffer entries, rename registers, etc. (Therefore, you are limited by only the functional units.) I'll do the first one as an example.

Predict Taken, Correct				
Clock	ALU1	ALU2	Mem	Br
1				
2				
3				
4				
5				
6				

Predict Taken, Incorrect				
Clock	ALU1	ALU2	Mem	Br
1				
2				
3				
4				
5				
6				

Predict Not Taken, Correct				
Clock	ALU1	ALU2	Mem	Br
1				
2				
3				
4				
5				
6				

Predict Not Taken, Incorrect				
Clock	ALU1	ALU2	Mem	Br
1				
2				
3				
4				
5				
6				

Now, assume you have predicate registers P0..P31, and you have predicate assignment statements that look like the conditional branch instructions. For example, the instruction `PGE    R1, 0, P1` sets P1 to be true iff R1 is greater or equal to 0. Every instruction can be predicated by putting the corresponding predicate register, possibly negated, in parentheses after the instruction. For example:

```
ADD     R1, R2, R3 (P1)          -- if P1 then R1 = R2 + R3
ADD     R1, R2, R3 (!P1)         -- if (!P1) then R1 = R2 + R3
```

Convert the code sequence at the start of this problem to use predicated execution.

L1:____

L2:____

L3:____

L4:____

L5:____

L6:____

L7:____

L8:____

L9:____

Show the execution schedule of predicated code. You may assume that predicate assignments are executed in the branch unit, and that the ALU units are predicated.

With Predicated Execution				
Clock	ALU1	ALU2	Mem	Br
1				
2				
3				
4				
5				
6				
7				
8				

4 Newton's Method (10 pts)

For this problem, you will derive how to use Newton's method to compute square roots. Suppose we are given a number b. We wish to compute the square root of b.

- What is the formula for one iteration of Newton's method?

  $x_{i+1} =$ ____

- Give a function $f(x)$ which has a root when $x = \sqrt{b}$. Since we're trying to compute $\sqrt{b}$, your function must not depend on being able to compute square roots. For example, $f(x) = x - \sqrt{b}$ would be a bad answer to this question.

  ____

- Compute $f'(x)$ for the function you gave.

  ____

- Plug your answers into the formula for Newton's method, and simplify. What is your iteration for square root? Would this be easy to implement?

- Starting with $x_0 = 1$, use your iteration twice to approximate the square root of 2.

  $x_1 =$ ____

  $x_2 =$ ____

5 Pipeline Gating (20 pts)

I noticed none of you managed to attend Srilatha Manne's talk on pipeline gating for low power. In this problem, I will give a greatly simplified version of her work (which, of course, doesn't work as well).

Let's consider a simple 5-stage pipeline processor – similar to what you worked on in 318. This processor has no branch delay slots if a branch is predicted correctly. If the branch is mispredicted, there is a penalty of one cycle, so one instruction in the pipeline will need to be killed The processor will use a simple 2-bit saturating counter per branch for branch prediction.

For our crude version of pipeline gating, the idea is very simple. If a branch prediction is weakly taken or weakly not taken, we won't make any prediction at all. Instead, we will shut off power to the IF stage of the pipeline, saving a bit of power, and introducing a one cycle bubble in the pipeline (as if we had stalled). Essentially, we stall instead of making a prediction when the branch prediction strength is weak. (It's in the 10 or 01 weakly predict taken/not-taken state.)

Consider the following sequence of code:

```
        LDI     R1, #0          -- R1 = 0
Loop:   ANDI    R2, R1, #1      -- R2 = R1 & 0x1 -- get the low-order bit
        BEQ     R2, 0, Next     -- if R2 == 0 goto Next:
        ADDI    R3, R3, #1      -- R3 += 1
Next:   ADDI    R1, R1, #1      -- R1 += 1
        BNE     R1, 8, Loop:    -- if R1!=8 goto Loop:
```

Assume the prediction counters for forward branches are initially at 01, weakly predict not taken; and for backward branches, initially at 10, weakly predict taken.

What is the branch prediction rate for this code?

How many cycles does the code take to execute, if we don't use pipeline gating?

How many cycles does the code take to execute, if we use pipeline gating?

Suppose that the IF stage has 1/6 of the total capacitance of the chip, and suppose the pipeline gating logic consumes negligible power. How much energy do we save using pipeline gating while running this code?

If, instead of using pipeline gating, we simply reduced the supply voltage to get the same reduction in performance as we got from pipeline gating, how much energy would we save? (Speed drops linearly with voltage.)