Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 382N, Spring 2006
Y. N. Patt, Instructor
Danny Lynch and Chang Joo Lee, TAs
Exam 1, March 29, 2006

Name :_____

Problem 1 (14 points):_____

Problem 2 (14 points):_____

Problem 3 (14 points):_____

Problem 4 (14 points):_____

Problem 5 (14 points):_____

Problem 6 (14 points):_____

Problem 7 (14 points):_____

Problem 8 (14 points):_____

Problem 9 (14 points):_____

Bonus for legibility on

all anwers (2 points):_____

Total (100 points):_____

Directions: The first two problems of this exam are required problems. You may answer any 5 of the last 7 problems. Place an "X" in the 2 lines above for the 2 problems that you choose not to answer.

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

GOOD LUCK!

Name:_____

Problem 1 - Required (14 points):

We have a RISC machine (with 32-bit fixed length instructions), and we would like to allow the fetch stage to prefetch future instructions if possible. To allow this we are going to design an 8-byte instruction buffer between the fetch and decode stages that will hold up to 2 instructions. The fetch stage will keep putting instructions into the buffer as long as there is room (even if the decode stage is stalled for instance). The fetch stage will then move on to fetching the next sequential instruction.

We have given the beginning of the verilog code for the instruction buffer below.

```
module inst_buf (out_valid,            //output - is the output sent to the decoder valid?
                 out_to_decode[31:0],  //output - the instruction that is sent from the
                                       //buffer to the decoder

                 decode_stall,         //input - is the decode stage stalled
                 icache_hit,           //input - did we have an icache hit, i.e. is the
                                       //icache_data valid?
                 icache_data[31:0],    //input - the next instruction from the icache
                 flush ){              //input - should we flush the buffer


} endmodule
```

Note: Anytime there is valid data in the instruction buffer then the **out_to_decode** data should always be valid. The icache line size is 4 bytes.

Please finish the definition of the instruction buffer using the following library.

```
module  dff$(clk, Din, Q);                      // 1-bit wide, D flip-flop
module  reg32e$(clk, Din, Q, write_enable);     // 32-bit wide, write-enabled register
module  nand2$(out, in0, in1);                  // two input nand gate
module  nor2$ (out, in0, in1);                  // two input nor gate
module  mux2$ (out, in0, in1, s0);              // 2 to 1 mux
module  inv1$ (out, in);                        // inverter
```

You may answer either in Verilog or with a block diagram, whichever you prefer. In either case, please use modular design principles in your solution. If you use a block diagram to answer, be sure to label all the signal names and the widths. Please make sure your solution is as simple and clear as possible.

Name:_____

Problem 1 (continued) - Required (14 points):

Additional space for your instruction buffer design.

Name:_____

Problem 2 - Required (14 points):

Decode the x86 instruction stream shown below into its component instructions.

```
2E  02  4D  00  66  89  AC  48  66  05  26  00  F3  A5  EA  66  0F  FE  6A  67  1F  F4
```

Show each instruction in the following form :

```
prefix opcode mode r/m SIB disp imm dest src1 src2
```

Where a field does not apply for a particular instruction, just ignore that field for that instruction. (We have provided room for 6 instructions. Hint: There are not more than six instructions in the above code.)

inst #	prefix	opcode	mode	r/m	SIB	disp	imm	dest	src1	src2
1										
2										
3										
4										
5										
6										

Name:_____

Problem 3 (14 points):

You designed a microprocessor. It came back from the fab with an error: one of the bits is stuck. We call the bit a stuck-at-0 fault if the bit is always 0 (i.e., you can not store a 1 in it). We call the bit a stuck-at-1 fault if the bit is always 1 (you can not store a 0 in it).

Consider each of the four structures below independently. Assume the structure contains a stuck at 0 or 1 fault. Does the fault affect the correctness of the chip? Does the fault affect performance? Explain. (Note: For each structure, consider separately stuck-at-0 and stuck-at-1 faults.)

A bit in the Register Alias Table:

The dirty bit in one of the tag store entries:

The LRU bit for one of the sets of a 2-way set associative cache:

A bit in the Branch History Register:

Name:_____

Problem 4 (14 points):

An engineer designs a piece of logic that, upon discovering that a branch has been mispredicted, cancels all memory requests that were initiated by load instructions along the mispredicted path.

We evaluate this new feature on application A, and notice a performance improvement. On application B, we notice a performance degradation. There is nothing wrong with the measurement methodology. Explain how both results are possible.

Name:_____

Problem 5 (14 points):

We are designing compilers for both a 3-wide VLIW processor and a Pentium Pro 3-wide issue machine. Assume the VLIW processor also uses the x86 instruction encodings for each of three instructions that make up its VLIW word. Both compilers' job is to produce code that optimizes the fetch and decode bandwidth of the corresponding front ends of the two machines. When will the VLIW code be better? When will the Pentium Pro code be better?

Name:_____

Problem 6 (14 points):

A perceptron branch predictor has been designed for use on the front end of a new microprocessor. The branch history register consists of two bits: x1, the direction of the most recent branch, and x2, the direction of the second most recent branch. At the time we examine the chip, w1=1, w2=4, and the Threshold (T) = 2. For each possible state of the branch history register, what will the perceptron predictor predict?

Name:_____

Problem 7 (14 points):

We have a 16 wide issue microprocessor and (magically) a perfect branch predictor (100% accuracy) and perfect caches (100% hit rate). Everything is working correctly on the chip (i.e. no bugs).

a. We measure the IPC on one application and it is less than 16. How is this possible? Explain.

b. Can the IPC be less than 1. Explain. If yes, give an example piece of code (in pseudo-assembly language) that will produce this. If no, why not?

c. The pipeline consists of 10 stages. If you could redesign the chip, would you change the number of stages? Explain.

Name:_____

Problem 8 (14 points):

A 382N graduate working in industry looks at the design of a microprocessor which has an IPC of 2.75. She notices that 20% of the transistors on the chip are unused. Using the extra transistors she designs a more accurate branch predictor which yields 2.90 IPC. The catch is that using the new branch predictor requires increasing the cycle time by 5%. Should the chip be redesigned with this new and improved branch predictor? Explain. Discuss all factors that could influence your decision. (Hint: there are at least three issues to consider.)

Name:_____

Problem 9 (14 points):

We have discussed the various base designs of the two-level branch predictor. Consider the classical GAs predictor, first reported in ASPLOS in 1992. Further study uncovered problems with GAs. They are listed below. Explain how you would solve each.

a. Interference. Interference is defined as the information introduced into a pattern history table entry by one branch, and later used as a basis of prediction by another branch having nothing to do with the first branch. How can we reduce negative interference? Explain how your suggestion works. (Hint: there are at least two ways. You get to choose which one you discuss.)

b. Suppose the branch history register contains three bits. The following code has PC=X, with the branch history register = 001. If the instruction flow came through a, the branch at X is always taken. If the instruction flow came through d, the branch at X is always not taken. About half the time the flow comes through a, about half the time the flow comes through d. How can we improve this branch predictor?

```
a: BrZ A
b: BrN B
c: BrG X
 ...
d: BrZ W
e: BrN Y
f: BrL X
 ...
X: BrN Z
```

c. One of the problems with the two-level predictor is its warmup time, the time required to store enough history information so that the predictor can make an intelligent prediction. How can this problem be fixed?