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

EE 382N, Spring 2016
Y. N. Patt, Instructor
Ben Lin and Stephen Pruett, TAs
Exam 1, March 9, 2016

Name :_____

Problem 1 (12 points):_____

Problem 2 (12 points):_____

Problem 3 (12 points):_____

Problem 4 (12 points):_____

Problem 5 (12 points):_____

Problem 6 (12 points):_____

Problem 7 (12 points):_____

Problem 8 (12 points):_____

Problem 9 (12 points):_____

Problem 10 (12 points):_____

Problem 11 (12 points):_____

Bonus points for legibility of all answers (4 points):_____

Total (100 points):_____

Directions: The exam contains 11 problems. You are required to do #1, #2, and 6 of the remaining 9. For the the 3 problems you do not want us to grade, please cross them out with an 'X' in the space provided for the grade of that problem. Be sure to sign your name in the space below to acknowlege that you've read the instructions and will not cheat on the exam.

Please make your handwriting clear, legible, and use a dark enough pen or pencil that it is easily readable. If I can not easily read your handwriting, the answer will be marked wrong. I apologize if you take this as offense, but I think it is unreasonable to scribble on the exam and expect me to squint and struggle to decipher your handwriting.

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.

I WILL NOT CHEAT ON THIS EXAM

Signature:_____

GOOD LUCK!

Name:_____

Problem 1 - Required (12 points): An instruction buffer exists between the fetch and decode stages of the pipeline. The Fetch stage fetches bytes from the I-cache and inserts them into the instruction buffer, while the Decode stage removes bytes from the instruction buffer and decodes them.

At the start of the cycle there are 'n' valid bytes in the buffer. The decoder takes the 'n' valid bytes from the buffer and attempts to decodes them. Since x86 is a variable-length instruction set, the number of bytes that the decoder can decode is also variable. Let's say the decoder was able to actually decode 'd' bytes from the instruction buffer ($0 \leq d \leq n$) in the current cycle. The top 'd' bytes of the instruction buffer can then be removed from the instruction buffer.

Meanwhile, the fetch stage is working to keep the instruction buffer full. A Fetch Pointer is maintained in the Fetch stage which contains the address of the next byte that needs to be inserted into the instruction buffer. The Fetch stage fetches bytes from the I-cache and inserts them into the instruction buffer as space becomes available in the instruction buffer.

Complete the module "fetch_across_line_boundary" on the next page which determines whether or not we need to fetch across an I-cache cacheline boundary in order to completely fill the instruction buffer for the next cycle.

The only modules you're allowed to use are 32-bit adders and inverters, as specified below:

```
module add32$ (sum, a, b, cin);
       input[31:0] a, b;   // the two numbers to be added
       input cin;          // the carry-in into bit 0
       output[31:0] sum;   // the resulting sum
endmodule

module  inv32$ (out, in);
        input[31:0] in;    // inverter input
        output[31:0] out;  // inverter output
endmodule
```

Assume we have an instruction buffer that can hold 16 bytes and assume that the I-cache line size is 16 bytes.

Name:_____

```
module fetch_across_line_boundary (cross_boundary, n, d, fetch_pointer);
       input[31:0] n;             // number of valid bytes in the buffer at the start of the cycle
       input[31:0] d;             // number of bytes the decoder was able to decode this cycle
       input[31:0] fetch_pointer; // address of the next byte that needs to be inserted into the
                                  // instruction buffer
       output cross_boundary;     // 1 if we need to fetch across an I-cache cacheline boundary to
                                  // completely fill the instruction buffer; 0 otherwise

       // your code below









endmodule
```

Name:_____

Problem 2 - Required (12 points):

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

```
81 44 48 08 ce fa ed fe 66 ea 66 ea 66 ea
```

'81' is byte 0, and 'ea' is byte 13. Show the exact address calculation for any memory operand. Use M_32[ ] to indicate a 32 bit memory location. Immediates, displacements, and literals should begin with the $ symbol. We have provided room for 7 instructions. There are 7 or fewer instructions in the above code.

1	
2	
3	
4	
5	
6	
7	

Show below is some example assembly. Please use this as a reference for notation.

```
OR M_32[(ES<<16) + EDX + (ECX*4) + $0x12345678], $0x87654321
MOV AX, $0xAB87
DAA
```

Name:_____

Problem 3 (12 points): I have noted in class that sometimes a new feature appears on a chip because of good luck. Explain what I meant by that, and give an example.

Problem 4 (12 points): We have pointed out several papers worth reading. Listed below are four papers. Pick one, and identify an important contribution that paper makes to our field. Explain.

(1) Subharao Palacharla, Norman Jouppi, J.E. Smith, "Complexity-Effective Superscalar Processors," ISCA 1997.

(2) Hirata, H. et. al.,"An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads" ISCA 1992.

(3) Robert S. Chappell et. al., "Simultaneous subordinate microthreading" ISCA 1999.

(4) Eric Sprangle et.al., "Facilitating superscalar processing via a combined static/dynamic register renaming scheme" Micro 1994.

Name:_____

Problem 5 (12 points): One of the decisions we had to make in designing a Trace Cache was whether to allow path associativity, that is whether to allow both the trace A,B,C and the trace A,D,E to be present in the Trace Cache at the same time. Did we decide to allow it or not? What was our rationale.

Problem 6 (12 points): The Pentium 4 ALU was able to run at twice the frequency of the chip, that is, 7.8 GHz, rather than 3.9 GHz, and still do a sequence of arithmetic operations back-to-back, with no bubbles in the pipeline. What do we mean by back-to-back operations? How was this possible. Illustrate with an example.

Name:_____

Problem 7 (12 points): The Block-structured ISA improves performance if enlarged blocks are allowed. Give two key reasons for this, and explain how they improve performance.

Problem 8 (12 points): The Pentium Pro had a single permanent register file for storing results on retirement. The Pentium 4 did not move the data on retirement. In order to not move the data, Pentium 4 had to introduce a second register alias table. Why was this necessary? What function did each register alias table provide.

Name:_____

Problem 9 (12 points): In an out-of-order execution machine, one often has the situation where an older store's memory address has not been computed at the time a subsequent load is ready to access memory. If there is overlap between the two addresses, we should wait for the store address to be computed. If there is no overlap, we can improve performance by allowing the load to access memory as soon as possible. The problem: Do we aggressively allow the load to access memory, or do we wait until the store address is known. I called the problem"the unknown address problem." Josh Fisher called it "memory disambiguation" What motivated our very different names for this problem, such that both names make sense. Please be specific.

Problem 10 (12 points): The Decimal Adjust (DAA) instruction allows integers encoded in BCD to be added with a standard 2's complement adder. The mechanism is to first do the addition, and then execute the DAA instruction to correct for any error created by adding BCD digits with a 2's complement adder. What extra hardware is included in the x86 data path to make this possible, and show how it is used with an example.

Name:_____

Problem 11 (12 points): The term Semantic Gap has been given to the "distance" between the high level language humans write programs in, and the ISA. A small semantic gap means instructions in the ISA that are very much like high level constructs. AOBLEQ (Add one and branch if less than or equal) conjures up thoughts of the iteration control of a "for loop" A large semantic gap means instructions that are very simple and look more like micro-ops than high level language constructs. What are the advantages of each.