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

EE 382N.19, Spring 2020
Y. N. Patt, Instructor
Aniket Deshmukh, TA
Written midterm
April 1, 2020

There are 12 problems on the exam. You are being asked to solve problems 1 and 2, and to select 6 of the remaining 10 problems. That is, you are asked to solve a total of 8 of the 12 problems.

Problem (1): _____

Problem (2): _____

Problem ( ): _____

Problem ( ): _____

Problem ( ): _____

Problem ( ): _____

Problem ( ): _____

Problem ( ): _____

Total (100 points):_____

Feel free to use scratch paper while taking the exam. The documents you are allowed to reference are listed below:

1. The three sheets of paper that you prepared.
2. The x86 Instruction Set Reference: http://users.ece.utexas.edu/~patt/20s.382N/handouts/x86%20Instruction%20Set%20Reference.pdf
3. The List of Instructions for the project: http://users.ece.utexas.edu/~patt/20s.382N/problemset/project/inst.html

Problem 1 (14 points): A processor contains a single-ported data cache. When the processor wishes to read from the cache, it provides a 32-bit address, a 3-bit size which specifies the number of bytes to be read, and a 1-bit valid signal. The processor is not required to provide aligned addresses, meaning that read requests can span across cache lines.

The data cache, however, only supports accesses to a single cache line at a time. That is, if the processor wishes to read data that spans multiple cache lines, it must do so by generating two requests.

To solve this problem, the data cache is augmented with a state machine that can perform up to two cache line accesses to provide the correct data to the processor. The state machine module has two sets of inputs: each containing an address, a size which specifies the number of bytes to be read starting at that address, and a valid bit (whether or not an access should be made). Note that the two sets of inputs are provided to the state machine in the **same cycle**.

Your job: complete the Verilog module shown below which specifies the **combinational logic** that takes the processor output signals and breaks them down into one or two accesses depending on whether the access crosses the cache line boundary. The size of a cache line is **16B**.

```
module your_module (
  //Inputs from processor
  input[31:0]       addr,
  input[2:0]        size,
  input[0:0]        valid,

  //Outputs sent to data cache state machine
  output[31:0]      addr1,
  output[2:0]       size1,
  output[0:0]       valid1,

  output[31:0]      addr2,
  output[2:0]       size2,
  output[0:0]       valid2
);
```

You are only allowed to use the following modules in your design:

```
module add_32bit  (out, in0, in1);       //out = in1 + in0
module sub_32bit  (out, in0, in1);       //out = in1 - in0
module mux2_1_nbit (out, sel, in0, in1); //mux for n bit inputs, you can pick any n
Any basic logic gate (inverter, or, and, xor) you need
```

Problem 2 (14 points):

Part a: The contents of memory at starting location 0x3000 are shown below.

```
0x3000 : C1  E0  04  01
0x3004 : D0  05  78  56
0x3008 : 34  12  65  66
0x300C : 81  00  12  34
```

Note: Location 0x3000 contains C1, and location x300F contains 34.

Decode the x86 instructions specified by these bytes. 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. Shown below is some example assembly. Please use this as a reference for notation.

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

Part b: x86 provides complex addressing modes. As a result, it may be possible to reduce the number of instructions while retaining the functionality of the above code. If this is possible, write down the new instruction(s). If this is not possible, explain why.

Problem 3 (12 points): We said in class that three goals to highest performance processing are (1) fetching a full issue-width of instructions each cycle, (2) fetching data from an infinite capacity, single-cycle memory, and (3) processing instructions as soon as their dependencies are resolved. The Superblock addresses one of these goals. Which one, and how so?

Problem 4 (12 points): The Tomasulo Algorithm was invented for performing out-of-order execution in the floating point unit of IBM's 360/91 computer. It had a tragic flaw that is reflected in the following program segment:

```
MUL  R1, R2, R3
ADD  R4, R5, R6
DIV  R6, R1, R4
ADD  R4, R2, R3
```

Assume the Divide instruction causes an exception.

What is the problem. Explain within the context of the above program segment.

Problem 5 (12 points): Instead of the compiler always generating a branch instruction, the "wish branch" gave the compiler a choice to use either a branch instruction or predicated code, depending on which made more sense in each instance.

Part a: What determined whether the compiler generated a normal branch instruction or a wish branch?
Part b: If the compiler generated a wish branch, the decoder produced a branch instruction or predicated code at run time based on what? Please be specific.

Problem 6 (12 points): The predicate on which a branch is either taken or not taken is either found in the state of the condition codes or in the contents of a register as a result of a compare instruction.

Part a: What advantage do condition codes have over use of a register?
Part b: What advantage does the use of a register have over condition codes?
Part c: The IBM RS6000 is the only ISA I know of that used condition codes but did not suffer the disadvantage of part b. How did it do it?

Problem 7 (12 points): Suggested implementations of a trace cache suggest that a trace segment should consist of three basic blocks, call them A, B, C, the branch that terminates block A predicts block B, and the branch that terminates block B predicts block C. Some have suggested an alternate approach: the trace segment consists of blocks X, Y, Z where the branch terminating block X executes block Y or block Z, depending on whether that branch is taken or not taken.

Is the alternate approach a good idea or a bad idea. Why?

Problem 8 (12 points): Two characteristics of the Block-Structured ISA account for its improvement in performance, and both are enhanced if one introduces Enlarged Basic Blocks. Enlarged Basic Blocks also have two potential negatives.

Part a: Name one of the two good features, and explain.
Part b: Name one of the two negatives, and explain.

Problem 9 (12 points): Programs are written generally in high level languages, compiled into code expressed in a chip's ISA, and implemented by a microarchitecture that acts based on the control signals asserted each clock cycle. There are two schools of thought as to where to place the ISA: close to the High Level Language description, or close to the control signals that control the data path. The two approaches have tradeoffs associated with the implementation of each instruction in the ISA: (a) wasted microcycles, (b) opportunity to achieve parallel execution within the instruction, and (c) efficient use of the instruction cache. Compare the two approaches with respect to the three tradeoffs.

Problem 10 (12 points): The Cray 1 had no caches.

Part a: Why?
Part b: However, it did have two very useful structures (one for instructions and one for data) that served much the same purpose as I and D caches. What were they and how did they work?

Problem 11 (12 points): The second Alpha chip, the 21164 had very unusual characteristics of its Level 2 cache: 96KB, 3-way set associative. How come?

Problem 12 (12 points): The Pentium chip and the first Alpha chip both had pipeline control, both had branch predictors, both had two-wide issue. On a predicted taken branch the Alpha chip incurred a penalty, the Pentium chip did not.

Part a: What was the penalty?
Part b: How come the Pentium chip did not incur it?