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

EE 382N.19, Spring 2024
Y. N. Patt, Instructor
Ali Mansoorshahi, TA
Written Midterm
March 20, 2024

There are 16 problems on the exam. You are asked to solve **ALL problems.** Leaving a problem blank counts as 1 point.

Name: SOLUTION EID:

Problem 1: _____ Problem 9: _____

Problem 2: _____ Problem 10: _____

Problem 3: _____ Problem 11: _____

Problem 4: _____ Problem 12: _____

Problem 5: _____ Problem 13: _____

Problem 6: _____ Problem 14: _____

Problem 7: _____ Problem 15: _____

Problem 8: _____ Problem 16: _____

Total (100 points):_____

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: Be sure to sign your name in the space below to acknowledge that you've read the instructions and will not cheat on the exam. Please be sure your name is recorded on each sheet of the exam.

I WILL NOT CHEAT ON THIS EXAM

Signature:

GOOD LUCK!

Question 1.

We all know the importance of dependency checking. It is easy to check dependencies when the number of locations is small (e.g. the register file), but becomes more difficult when the number of locations is large, and each access can access multiple locations (e.g. memory). Still we have to check dependencies.

We say two memory instructions are dependent if they access the same bytes. In order to check this we need the starting address of the access, and the size of the access.

Your job: Finish the following verilog module on the next page that determines if two memory accesses overlap.

```
module your_module (
        input[31:0] addr0,
        input[31:0] addr1,
        input[2:0]  size0, //size provided in bytes
        input[2:0]  size1,
        output[0:0] is_dependent
)
```

You are allowed to use the following modules:

```
module comp_32b(leq, geq, in0, in1);  // leq = 1 if in0 <= in1, 0 otherwise
                                      // geq = 1 if in0 >= in1, 0 otherwise
module add_32b(out, in0, in1); // out = in0 + in2 (signed addition)
module mux2_1(out, in0, in1, sel); //1-bit 2 to 1 mux
Any basic logic gate (inverter, or, and, xor) you need
```

```
module your_module (
        input[31:0] addr0,
        input[31:0] addr1,
        input[2:0]  size0, //size provided in bytes
        input[2:0]  size1,
        output[0:0] is_dependent
)
Wire A,B;
Wire [31:0] addr0_end, addr1_end;

add_32b(addr0_end, addr0, {29'b0, size0});
add_32b(addr1_end, addr1, {29'b0, size1});

Comp_32b(A, 1'b0, addr0, addr1_end);
Comp_32b(1'b0, B, addr0_end, addr1);

and(is_dependent, A, B);


endmodule;
```

Question 2.

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

0x3000: 66 81 E2 81
0x3004: E2/81 E2 81
0x3008: E2 81 E2 0F
0x300C: 7F 04 CD 50
0x3010: 51 52 53

Note: Location 0x3000 contains x66, and location x3012 contains x53. The machine is running with user privileges in 32-bit mode.

Your job: 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

Note: you may not need to use all the rows

Instruction Bytes	Instruction
66 81 E2 81E2	AND DX, $E281
81 E2 81E2 81 E2	AND EDX, $E281E281
0F 7F 04 CD 50 51 52 53	MOVQ M_64[(DS<<16)+8*ECX+535251505], mm0

Question 3

One solution proposed for the branch prediction problem is to simply generate two pipelines, one if the branch is taken, the other if the branch is not taken. In fact, IBM developed a product that did that. Is it a good idea or a bad idea? EXPLAIN.

BAD IDEA
WITH A 4-WIDE ISSUE, EACH CLOCK CYCLE A BRANCH WILL BE FETCH, DOUBLING THE NUMBER OF PIPELINES. EVEN WITH A 5-STAGE PIPELINE, THAT MEANS 32 PIPELINES OF WHICH ONLY ONE IS GOOD.

Question 4

The x86 ISA has a mechanism that gets rid of the need for a conditional branch at the end of some "for loops." How does it work?

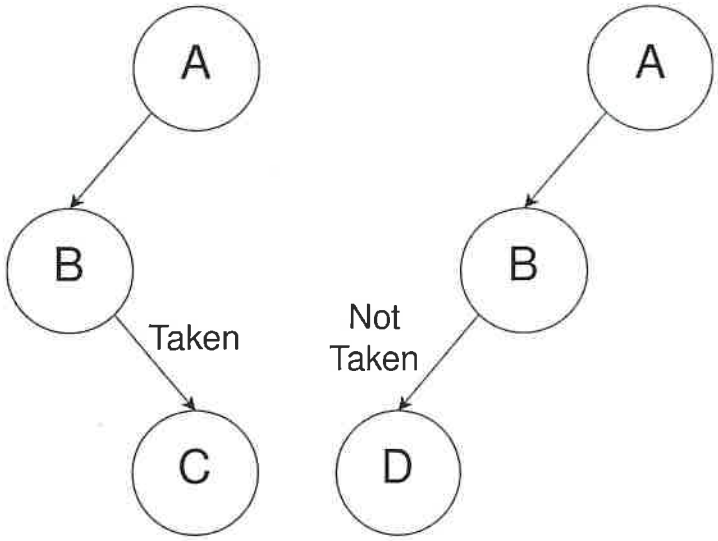
THE REPEAT PREFIX CAUSES THE ECX REGISTER TO SUBTRACT ONE AND IF THE RESULT IS NOT ZERO, TAKES THE BRANCH. WHEN ECX = 0, THE FALL THROUGH PATH IS TAKEN

Question 5

Scott McFarling's gshare predictor improves over my GAg predictor. How?

BY XORING THE BHR WITH BITS OF THE PC, TWO DIFFERENT BRANCHES WITH THE SAME HISTORY WILL INDEX INTO TWO DIFFERENT 2-BIT COUNTERS, REMOVING INTERFERENCE.

Question 6



A Trace Cache element contains three basic blocks A,B,C, as shown, and the predictions for the branches terminating each block. The next time (at t-one) that the Trace Cache access corresponds to the starting address of A, the branch terminating B is predicted to fetch the block D, rather than C, as shown. In my implementation of the Trace Cache, what does the microarchitecture do with each block (A,B,C,D) at time t-one?

A:	B:	C:	D:
SEND TO THE CORE	SEND TO CORE	PUT IT ASIDE. MARK IT INACTIVE ISSUE	GET IT FROM THE I CACHE

Question 7

A major plus of the Block-Structured ISA is that it eliminates most accesses to the register file. How is that possible?

IF THE RESULT OF ONE INSTRUCTION IS A SOURCE OF ANOTHER INSTRUCTION AND IS NOT A LIVE OUT, THE WRITE OF THE FIRST AND READ OF THE SECOND DO NOT ACCESS THE REGISTER FILE

Question 8

A wish branch can act like a normal branch or be turned into predication. This can be accomplished by the compiler and microarchitecture working together. The compiler's job is to determine whether predicating the branch can make sense. If the compiler determines that it can make sense, it turns the branch into a wish branch. How does the compiler decide if it makes sense? If the compiler turns the branch into a wish branch, the microarchitecture determines if the branch should be predicted or predicated. How does the microarchitecture determine that?

How the compiler decides:

IF THE MERGE POINT IS NOT CLOSE TO THE BRANCH, PREDICATION IS A BAD IDEA

How the microarchitecture determines:

IF BR. PREDICTION ACCURACY IS HIGH, PREDICT.
IF LOW, PREDICATE.

Question 9

What benefit does a two-address ISA have over a three-address ISA? The x86 is a two-address ISA. What problem results from this?

Benefits of two-address:

FEWER BITS OF THE INSTRUCTION NEEDED TO SPECIFY THE REGISTERS

Problems with two-address:

ONE OF THE SOURCES GETS CLOBBERED BY THE DESTINATION WRITE

Question 10

Consider an in-order pipeline. A common occurrence in programs is MUL R1,R2,R3 followed by ADD R4,R5,R1. This requires a hardware interlock to prevent the stale contents of R1 to be fetched as a source of the ADD instruction before R1 is written with the result of the MUL. The MIPS R2 initially had no hardware interlocks. How did it prevent the microarchitecture from using stale data from R1.

BY PUTTING NO-OP INSTRUCTIONS BETWEEN THE MUL AND THE ADD.

Question 11

Many would-be comparch experts promote performance as satisfying the equation: Performance = 1/(N times CPI times t), where N is the number of instructions executed, CPI is cycles required per instruction and t is cycle time. What tragic flaw is present in this equation? EXPLAIN!

CPI DEPENDS ON WHAT ELSE IS GOING ON IN THIS PROGRAM

Question 12

What is endianness? Part of the ISA or part of the microarchitecture?

ENDIANNESS SPECIFIES THE NUMBERING ORDER OF THE BITS OF A DATUM AND INSTRUCTION i.e., IS BIT 0 THE HIGH ORDER OR LOW ORDER BIT. IT IS A PART OF THE ISA.

Question 13

Intel has added 2MB and 1GB page sizes to the classic 4KB page size that has been around for more than 40 years. What benefit does that provide to the microarchitecture? What negative does it create?

FOR A GIVEN DATA STRUCTURE ONLY ONE PTE IS NEEDED. THEREFORE BETTER ACCESS OF THE TLB.

NEGATIVE IS THE AMOUNT OF PHYSICAL MEMORY NOT USED IN A PAGE.

Question 14

Most ISAs use condition codes to determine whether to take a conditional branch or use the fall through path. A few ISAs use general purpose registers. What is the advantage of using condition codes? What is the advantage of using general purpose registers?

ADVANTAGE: NOT NECESSARY TO USE THE BRANCH INSTRUCTION IMMEDIATELY AFTER CREATING THE CONDITION. ADVANTAGE OF CONDITION CODES: NO EXTR INSTRUCTION IS NEEDED TO CREATE CONDITION CODES.

Question 15

The industry has seen a few examples of machines that support two ISAs. IBM produced a Power PC chip that allowed Power PC code and x86 code to run on it. DEC produced a VAX machine that allowed VAX and PDP 11 code to run on it. What prompted these two companies to allow two ISAs to execute on its machine?

ALLOWING THE "OLD" ISA MEANS OLD CODE CAN STILL RUN.

Question 16

I have given the RISC V the nickname: buffet. Is the nickname appropriate? If yes, why? If not, why not?

A GOOD NICKNAME SINCE THE RISC V ALLOWS A DESIGNER TO PICK AND CHOOSE WICH SUB-ISAs HE/SHE NEEDS.