CS 351 Exam 2, Section 1 Wed. 11/2/2016

Name: _____

Rules and Hints

- You may use one handwritten 8.5 × 11″ cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- You may write your answers in the form *[mathematical expression][units]*. There is no need to actually do the arithmetic.
- Write your answers on scratch paper. Clearly label your answer to each question.

Grade

	Your Score	**Max Score**
Problem 1: Datapath tracing		30
Problem 2: Pipelining		20
Problem 3: Data hazards and forwarding		20
Problem 4: Control hazards and branch prediction		14
Problem 5: Caches		16
Total		**100**

Problem 1: Datapath tracing (30 points)

Part A (1 point)

+1 extra credit point if entire class gets it right. Offer valid for Section 1 only.
Before we begin: how many bits are in a byte?

Provide the exact numeric answer (in decimal, hex, or binary) to each question if possible. If not, describe the value without stating an exact number.

Consider the execution of this instruction: `STUR X9, [X3, #16]`

Assume the following:

- The bits of this instruction are stored in addresses 1200–1203 in memory.
- The initial value in X3 is 1500.
- The initial value in X9 is 600.

Part B: Instruction memory (4 points)

What value goes into the instruction memory's *Read address* port? How many bits is that value?

What is the value of the instruction memory's *Instruction[31–0]* output?

Part C: Register file (7 points)

What values go into the register file's *Read register 1* and *Read register 2* ports? How many bits are those values?

What values go into the register file's *Write register* and *Write data* ports? How many bits are those values?

What is the value of the *RegWrite* control signal?

Part D: ALU (6 points)

What are the values of the ALU's two data inputs? How many bits are those values?

What are the values of the ALU's *ALU result* and *Zero* outputs? How many bits are those values?

Part E: Branch target adder (5 points)

What two values are input to the top right adder? How many bits are those values?

What is the output of the top right adder?

What is the output of the top right mux? Conceptually, what is this mux choosing between?

Part F: Modifying the diagram (7 points)

How would you need to modify this diagram to support BR?

How would you set the following control signals to support BR?

- RegWrite
- MemRead
- MemWrite
- Reg2Loc
- ALUSrc
- ALUOp (just specify the operation in English)
- MemtoReg
- Branch

Problem 2: Pipelining (20 points)

Part A: Cycle time (5 points)

If you took a non-pipelined processor and pipelined it, would you expect the cycle time to increase, decrease, or stay *exactly* the same? Explain briefly.

Part B: Instruction latency (5 points)

If you took a non-pipelined processor and pipelined it, would you expect the latency of a single instruction (measured in seconds) to increase, decrease, or stay *exactly* the same? Explain briefly.

Part C: Throughput (5 points)

If you took a non-pipelined processor and pipelined it, would you expect the amount of time (measured in seconds) taken by a million instructions to increase, decrease, or stay *exactly* the same? Explain briefly.

Part D: Combining stages (5 points)

You may have noticed that, on our pipeline diagram, WB is much shorter than the other stages and consists only of a mux. What would be the argument against combining it with the MEM stage?

Problem 3: Data hazards and forwarding (20 points)

Part A: Hazard detection (5 points)

In what pipeline stage are data hazards detected? What information is used to detect them?

Part B: Pipelined execution (15 points)

Answer the following questions about the sequence of instructions:

```
LDUR X10, [X9, 0]
ADD X10, X10, X10
ANDI X10, X10, X9
```

Assuming all possible forwarding paths, how many cycles will this sequence take to execute? You may want to draw a pipeline diagram.

List all forwarding paths used. Each item in your list should contain the cycle number, the stage sending the data, and the stage receiving the data.

Problem 4: Control hazards and branch prediction (14 points)

Consider a branch with the following endlessly repeating pattern:
NT, T, T, NT, NT, T

Part A: Predict-not-taken (2 points)

What is the long-term accuracy of predict-not-taken for this branch?

Part B: 1-bit predictor (4 points)

What is the long-term accuracy of the 1-bit predictor for this branch?

Part C: 2-bit predictor (4 points)

What is the long-term accuracy of the 2-bit predictor for this branch? You can choose any starting state; just specify the one you picked.

Part D: Hybrid predictor (4 points)

A *hybrid predictor* combines 2 or more branch predictors to arrive at a final prediction. If you use a hybrid predictor that allows the 3 predictors in Parts A–C to "vote" and chooses the outcome predicted by the majority of the 3, what is the long-term accuracy? Use the same starting state for the 2-bit predictor as you did in Part C.

Problem 5: Caches (16 points)

Consider a system with an L1 cache whose access time is 1 cycle and whose hit rate is 95%, and a main memory whose access time is 200 cycles.

Part A: AMAT (7 points)

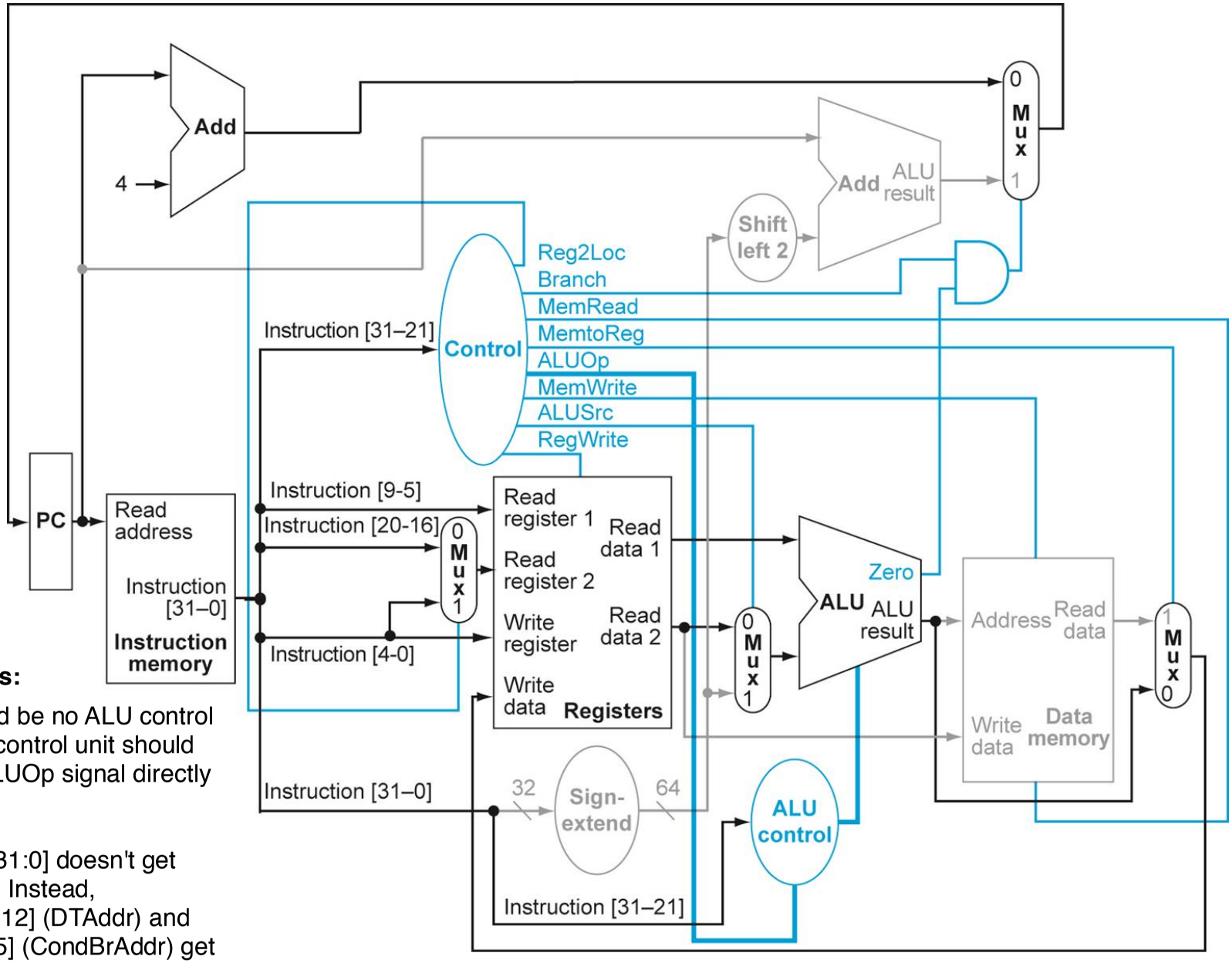
What is the average memory access time for this configuration?

Part B: L2 (5 points)

If you could add an L2 cache with a 10-cycle access time and a 50% hit rate, would it be worth it? You don't actually need to work out the arithmetic; just show the relevant expression(s) and explain how you would decide.

Part C: Access patterns (4 points)

You run some program on Tuesday. A certain short loop takes 100 iterations. On Wednesday, you rerun the program with slightly different input. The loop now takes 1 million iterations, but all of the other instructions executed are the same. Which day would you expect to have a higher cache hit rate, and why?



Modifications:

1. There should be no ALU control unit; the main control unit should send a 4-bit ALUOp signal directly to the ALU.

2. Instruction [31:0] doesn't get sign-extended. Instead, Instruction [20:12] (DTAddr) and Instruction[23:5] (CondBrAddr) get separately sign-extended. DTAddr goes to the ALUSrc mux, and CondBrAddr goes to the shift left.

LEGv8 Arithmetic Instructions

Instruction	Operation	Fmt	Opcode
ADD Rd, Rn, Rm	reg[Rd] = reg[Rn] + reg[Rm]	R	0x458
SUB Rd, Rn, Rm	reg[Rd] = reg[Rn] - reg[Rm]	R	0x658
ADDI Rd, Rn, imm	reg[Rd] = reg[Rn] + imm	I	0x488-0x489
SUBI Rd, Rn, imm	reg[Rd] = reg[Rn] - imm	I	0x688-0x689
ADDS Rd, Rn, Rm	reg[Rd] = reg[Rn] + reg[Rm]	R	0x558
SUBS Rd, Rn, Rm	reg[Rd] = reg[Rn] - reg[Rm]	R	0x758
ADDIS Rd, Rn, imm	reg[Rd] = reg[Rn] + imm	I	0x790-0x791
SUBIS Rd, Rn, imm	reg[Rd] = reg[Rn] - imm	I	0x788-0x789

The versions ending in S also set the Negative, Zero, Overflow, and Carry bits of the FLAGS register.

LEGv8 Logical Instructions

Instruction	Operation	Fmt	Opcode
AND Rd, Rn, Rm	reg[Rd] = reg[Rn] & reg[Rm]	R	0x450
ORR Rd, Rn, Rm	reg[Rd] = reg[Rn] | reg[Rm]	R	0x550
EOR Rd, Rn, Rm	reg[Rd] = reg[Rn] ^ reg[Rm]	R	0x650
ANDI Rd, Rn, imm	reg[Rd] = reg[Rn] & imm	I	0x490-0x491
ORRI Rd, Rn, imm	reg[Rd] = reg[Rn] | imm	I	0x590-0x591
EORI Rd, Rn, imm	reg[Rd] = reg[Rn] ^ imm	I	0x690-0x691
LSL Rd, Rn, shamt	reg[Rd] = reg[Rn] << shamt	R	0x69B
LSR Rd, Rn, shamt	reg[Rd] = reg[Rn] >> shamt	R	0x69A

LSL and LSR replace the shifted-out bits with 0s.

LEGv8 Data Transfer Instructions

Instruction	Operation	Fmt	Opcode
LDUR Rt, [Rn, DTAddr]	reg[Rt] = Mem[Rn + DTAddr]	D	0x7C2
STUR Rt, [Rn, DTAddr]	Mem[Rn + DTAddr] = reg[Rt]	D	0x7C0
LDURB Rt, [Rn, DTAddr]	Loads 8b from memory into least significant bits of register	D	0x1C2
STURB Rt, [Rn, DTAddr]	Stores 8b to memory	D	0x1C0
MOVZ Rd, MovImm, LSL Op	Rd[specific bits] = MovImm; Rd[all other bits] = 0	IM	0x694-0x697
MOVK Rd, MovImm, LSL Op	Rd[specific bits] = MovImm; Rd[all other bits] keep values	IM	0x794-0x797

Op is 0 to load MovImm in the rightmost 16 bits of the register, 16 (or 1 in machine code) to load it in bits 31-16, 32 (or 2) for the next 16 bits, and 48 (or 3) for the most significant bits.

LEGv8 Branch Instructions

Instruction	Operation	Fmt	Opcode
CBZ Rt, CondBrAddr	If (reg[Rt] == 0) PC = BrPC	CB	0x5A0-0x5A7
CBNZ Rt, CondBrAddr	If (reg[Rt] != 0) PC = BrPC	CB	0x5A8-0x5AF
B.cond CondBrAddr	If (FLAGS = cond) PC = BrPC	CB	0x2A0-0x2A7
B BrAddr	PC = BrPC	B	0x0A0-0x0BF
BR Rt	PC = reg[Rt]	R	0x6B0
BL BrAddr	reg[X30] = PC + 4; PC = BrPC	B	0x4A0-0X4BF

BrPC = PC + SignExt ([Cond]BrAddr << 2)

Flags: Negative (N), Zero (Z), Overflow (V), Carry (C)

Category	B.cond	Condition (if SUBS or SUBIS)	B.cond	Condition
Equality	B.EQ	Z = 1	B.NE	Z = 0
Signed < and <=	B.LT	N != V (signed)	B.LE	~(Z = 0 & N = V)
Signed > and >=	B.GT	Z = 0 & N = V	B.GE	N = V
Unsigned < and <=	B.LO	C = 0	B.LS	~(Z = 0 & C = 1)
Unsigned > and >=	B.HI	Z = 0 & C = 1	B.HS	C = 1

Instruction Formats

R-format:

11b: opcode	5b: Rm	6b: shamt	5b: Rn	5b: Rd

I-format:

10b: opcode	12b: immediate	5b: Rn	5b: Rd

D-format (note: op field is 2b):

11b: opcode	9b: data trans. addr	op	5b: Rn	5b: Rt

B-format:

6b: opcode	26b: branch address

CB-format:

8b: opcode	19b: conditional branch address	5b: Rt

IM-format:

11b: opcode	16b: MOV immediate	5b: Rd

Register List

Name	Use	Needs to be preserved across function call?
X0-X7	Function arguments / results	N
X8	Indirect result location	N
X9-X18	Temporary values	N
X19-X27	Saved values	Y
X28 (SP)	Stack pointer	Y
X29 (FP)	Frame pointer	Y
X30 (LR)	Return address	Y
XZR (31)	Constant value 0	n/a (const.)