CS 351 Exam 2 Wed. 4/5/2017

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.
- A LEGv8 cheat sheet and processor diagram are included on the last pages of this exam. Please feel free to tear them out.
- Write your answers on scratch paper. CLEARLY label your answer to each question.

Grade

	Your Score	**Max Score**
Problem 1: Datapath tracing		35
Problem 2: Pipelining		25
Problem 3: Data hazards and forwarding		15
Problem 4: Control hazards and branch prediction		20
Total		**95**

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 silly instruction: `L: CBZ X9, L`

Assume the following:

- The bits of this instruction are stored in addresses 1600–1603 in memory.
- The initial value in X9 is 12.

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 (12 points)

Consider a fake "conditional add" instruction that works like this:
`CADD X1, X2, X3, X4`
This instruction will add the values in X2 and X3, placing the result in X1, *only if* the value in X4 is zero. This instruction should be encoded like a normal ADD, with the extra X4 operand going in the shamt field.
Clearly draw any hardware you would need to add to the processor diagram to support this instruction. You can draw the new hardware in isolation, but clearly show where it would go.

Problem 1F continued

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

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

Problem 2: Pipelining (25 points)

Consider a non-pipelined processor with a cycle time of 1.3 ns (1300 ps). It can be broken down into the following pipeline stages:

1. IF: 300 ps
2. ID: 250 ps
3. EX: 250 ps
4. MEM: 300 ps
5. WB: 200 ps

Part A: Cycle time (5 points)

What is the minimum cycle time for the pipelined version of this processor?

Part B: Instruction latency (5 points)

What is the latency of a single instruction on both the pipelined and non-pipelined versions of this processor?

Part C: Throughput (5 points)

What is the limit on the speedup of the pipelined version over the non-pipelined version, given an arbitrarily large series of independent instructions?

Part D: Splitting stages (10 points)

What would be the pros and cons of creating a 6-stage pipeline by splitting IF into two equal pieces? What about a 7-stage pipeline that splits both IF and MEM into 2 pieces?

Problem 3: Data hazards and forwarding (15 points)

Answer the following questions about the sequence of instructions:

```
ADD X2, X5, X7
AND X2, X5, X7
SUB X2, X2, X7
LDUR X7, [SP, #4]
ADD X9, X2, X7
```

Part A: Stalling (5 points)

If this processor has no data forwarding, how many cycles will this sequence take to execute?

Part B: Forwarding (10 points)

Assuming all possible forwarding paths, how many cycles will this sequence take to execute? 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 (20 points)

Part A: Predict-not-taken (5 points)

Give a sequence of branch outcomes for which predict-not-taken outperforms the 1-bit and 2-bit predictors, and give the long-term accuracy of each.

Part B: 1-bit predictor (5 points)

Give a sequence of branch outcomes for which the 1-bit predictor outperforms both predict-not-taken and the 2-bit predictor, and give the long-term accuracy of each.

Part C: 2-bit predictor (5 points)

Give a sequence of branch outcomes for which the 2-bit predictor outperforms both predict-not-taken and the 1-bit predictor, and give the long-term accuracy of each.

Part D: General (5 points)

What information about the current instruction is available to the branch predictor? What information about past instructions is available to it?

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 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 Rd	PC = reg[Rd]	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

LEGv8 Data Transfer Instructions

Instruction	Operation	Fmt	Opcode
LDUR Rt, [Rn, DTAddr]	reg[Rt] = Mem[Rn + SignExt(DTAddr)]	D	0x7C2
STUR Rt, [Rn, DTAddr]	Mem[Rn + SignExt(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

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

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.)