CS 351 Exam 2 Mon. 11/13/2017

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.*
- When applicable, you may write your answers in the form *[mathematical expression][units]*. There is no need to actually do the arithmetic.
- You may detach the LEGv8 cheat sheet and processor diagram. No need to submit them with your exam.

Grade

	Your Score	**Max Score**
Problem 1: Datapath tracing		37
Problem 2: Pipelining		23
Problem 3: Data hazards and forwarding		20
Problem 4: Control hazards and branch prediction		20
Total		**100**

Problem 1: Datapath tracing (37 points)

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: `LDUR X2, [X19, #64]`

Assume the following:

- The bits of this instruction are stored in addresses 8000-8003 in memory.
- X19 contains the value 20000, which is the base address of a 100-element array of long long, where
$$a[i] = 2 * i$$
.
- Registers X0-X10 initially contain the value 5. Registers X11-X18 contain the value 6. Registers X20-X30 contain the value 7.

Part A: 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? Give this one in binary.

Part B: Register file (5 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 C: 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 D: Branch target adder (3 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 mux?

Part E: Data memory (9 points)

What two values are input to the data memory's *Address* and *Write Data* ports? How many bits are those values?

What is the output of the data memory? How many bits is it?

Part F: Modifying the diagram (10 points)

Consider a fake "load + increment" instruction that works like this:
`LDINC Rt, [Rn, #DTAddr]`

It works just like a regular load, with one addition: the stored value in Rn is incremented by 8 at the end of the instruction.

On the next page, clearly draw/explain 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 indicate where its inputs and outputs go.

I will assume that all the control signals should be set the same as LDUR; you should explain any differences or new signals.

Problem 1F continued

Problem 2: Pipelining (23 points)

Consider a non-pipelined processor with a cycle time of 1.5 ns (1500 ps).

Part A: Cycle time (6 points)

What is the lower bound on the cycle time for a 5-stage pipelined version of this processor? What would have to be true to get this cycle time?

Part B: Instruction latency (6 points)

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

Part C: Throughput (6 points)

What is the best-case CPI of a long sequence of instructions on both the ideal pipelined and non-pipelined versions? Which is faster?

Part D: Optimizations (5 points)

Explain how shaving 50 seconds off the data memory access time would change the pipelined and non-pipelined versions.

Problem 3: Data hazards and forwarding (20 points)

Attempt **one** of the following problems. **Provide diagrams to justify your answers.**

- For full credit; Give a sequence of LEGv8 instructions that takes 8 cycles with forwarding and 11 cycles without forwarding (i.e. with stalling).
- For up to 15 points: Give a sequence of LEGv8 instructions that has 4 cycles of stalls without forwarding. How long would it take with forwarding?
- For up to 10 points: Give a sequence of LEGv8 instructions that has 2 data dependencies between consecutive instructions, and explain how long it would take with and without forwarding.

Problem 4: Control hazards and branch prediction (20 points)

Part A: Architectural implications (5 points)

What if you didn't have the ability to squash mispredicted branches / mis-fetched instructions, but still had to design a correctly functioning processor? Briefly explain how this would affect the pipeline design and/or the performance.

Part B: Prediction accuracy (15 points)

Code for this problem:

```
L: ADD X2, XZR, #10
CBZ XZR, L
```

If branches are resolved when the branch reaches the EX stage, and the correct instruction is fetched in the following cycle, how many cycles will one iteration of this loop take in the steady state with...

- Predict-not-taken?
- A 1-bit predictor?
- A 2-bit predictor?

Partial credit for just giving the long-term accuracy of each predictor.

(intentionally blank)

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 (1B) from memory into least significant bits of register	D	0x1C2
STURB Rt, [Rn, DTAddr]	Stores 8b (1B) 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:

11b: opcode	9b: data trans. addr	00	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	Preserve for caller?
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.)