CS 351 Exam 1 Wed. 9/27/2017

Name: _____

Rules and Hints

- You may use one handwritten $8.5 \times 11''$ 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.

Grade

	Your Score	**Max Score**
Problem 1: Short answer		38
Problem 2: Processor performance and power		24
Problem 3: Assembly arrays		14
Problem 4: Assembly functions		24
Total		**100**

Problem 1: Short answer (38 points)

Part A (2 points each + 1 if entire class gets the first part right)

How many ***bits*** are in...

	Your answer
A byte?	
A `char`?	
The ID of a LEGv8 register?	
The value in a LEGv8 register?	
A memory address in the LEGv8 ISA?	

Part B (6 points)

Which of the following items are true?

The timing paths in a processor circuit constrain...

1. The minimum clock cycle time
2. The maximum clock cycle time
3. The minimum clock frequency
4. The maximum clock frequency

Part C (4 points)

State something clear and specific you learned from the Homework 1 article on chip fabrication. "Fabricating a chip is difficult and expensive." = 0 points.

Part D (6 points)

Considering your own productivity as a student and/or a software developer, define one metric of latency and one metric of throughput. Be specific and give units for each metric.

Part E (6 points)

Give two snippets of code (high-level or LEGv8), in which one snippet has a higher static instruction count but a lower dynamic instruction count than the other. Explain briefly.

Part F (6 points)

Decode the following binary LEGv8 instruction into assembly:

`0x913ffe81`

Problem 2: Processor performance and power (24 points)

You have a workload that takes 8 billion cycles on 1 core of a certain processor with a frequency of 2 GHz and a total power consumption of 70 W. Assume that the processor must operate all cores at the same frequency. ***Give units on all your answers to this problem.***

Part A: Execution time (4 points)

What is the execution time of your workload on this processor? You can refer to this value as T in your answers to subsequent subproblems.

Part B: Energy (4 points)

How much energy does this processor consume when running your program?

Part C: Parallelism, Part I (8 points)

There is one part of your program, which takes 60% of the sequential execution time, that can be parallelized across multiple cores with perfect speedup. Give the following values when your program runs on 2 cores, in the same frequency mode as before. Please circle or box each answer.

- Execution time
- Speedup over the 1-core version
- Power
- Energy

If you don't have enough information to calculate one or more of these metrics, please explain.

Part D: Parallelism, Part II (8 points)

There is another version of this processor that has 4 cores and runs at 1.8 GHz. If you use all 4 cores for the parallel part of the program, state all of the following quantities. Please circle or box each answer.

- Execution time
- Speedup over the 1-core, 2 GHz version
- Power
- Energy

If you don't have enough information to calculate one or more of these metrics, please explain.

Problem 3: Assembly arrays (14 points)

Write the following LEGv8 assembly. At the end of your snippets of code, all registers and memory locations should have their correct values according to the C semantics, except for temporary registers.
Assume that `i` is in X19 and `N` is in X20 and that both are long longs. You should also assume that `a` is a 2-element array of long longs that is stored at the top of the stack, and `b` is a long long array of size N that is stored on the stack immediately following `a`. It might be helpful to draw the stack before proceeding.

```
for ( ; i < N; i++) // don't initialize i; use its current value
     b[i] += a[i % 2];
```

Problem 4: Assembly functions (28 points)

Translate the following C functions to LEGv8, using only the instructions on your reference sheet. You should translate each function independently and not attempt to combine them. Obey all LEGv8 conventions about functions, registers, and stack usage.

Part A: Simple function (6 points)

```
long long f(long long x, long long y, long long z) {
     return x + y + z;
}
```

Part B: Calling a function (4 points)

Show assembly code to call your function from Part A, as follows. Assume that any necessary registers have already been preserved.

```
f(12, 5, -1);
```

Part C: A more complicated function (14 points)

Hint: recursion is no different from calling any other function. Don't get distracted by it.

```
long long f(long long n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return f(n - 1) + f(n - 2);
}
```

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.)