Problem 1: Assembly Programming Warmup

In this problem, you will be introduced to the 16 bit LC (Little Computer) 2200 assembly language. You will learn the syntax and semantics that underlie each of the supported operations. Although our instruction set is not as extensive as MIPS (Millions of Instructions Per Second) or x86, it is still able to solve a multitude of problems. In a LC-2200 computer, the word size is two bytes (16 bits) and there are 16 registers. We restrict memory to be addressable by words.

Register Conventions

Although the registers are for general-purpose use, we shall place restrictions on their use for the sake of convention and all that is good on this earth. Here is a table of their names and uses:

Table 1: Registers and their Uses

Register Number	Name	Use	Callee Save?
0	$zero	Always Zero	NA
1	$at	Reserved for the Assembler	NA
2	$v0	Return Value	No
3	$a0	Argument 1	No
4	$a1	Argument 2	No
5	$a2	Argument 3	No
6	$t0	Temporary Variable	No
7	$t1	Temporary Variable	No
8	$t2	Temporary Variable	No
9	$s0	Saved Register	Yes
10	$s1	Saved Register	Yes
11	$s2	Saved Register	Yes
12	$k0	Reserved for OS and Traps	NA
13	$sp	Stack Pointer	No
14	$fp	Frame Pointer	Yes
15	$ra	Return Address	No

Instruction Formats

There are four types of instructions: R-Type (Register Type), I-Type (Immediate value Type), J-Type (Jump Type), and O-Type (Other Type).

Here is the instruction format for R-Type instructions (ADD,NAND):

Bits	15 - 13	12 - 9	8 - 5	4 - 1	0
Purpose	opcode	RX	RY	RZ	Unused

Here is the instruction format for I-Type instructions (ADDI,LW,SW,BEQ):

Bits	15 - 13	12 - 9	8 - 5	4 - 0
Purpose	opcode	RX	RY	2's Complement Offset

Here is the instruction format for J-Type instructions (JALR):

Bits	15 - 13	12 - 9	8 - 5	4 - 0
Purpose	opcode	RX	RY	Unused (all 0s)

Symbolic instructions follow the same layout. That is, the order of the registers and offset fields align with the order given in the instruction format, ie. instructions in assembly are written as:
instruction RX, RY, RZ or
instruction RX, [optional offset](RY).

Table 2: Assembly Language Instruction Descriptions

Name	Type	Example	Opcode	Action
add	R	add $v0, $a0, $a2	000	Add contents of RY with the contents of RZ and store the result in RX.
nand	R	nand $v0, $a0, $a2	001	NAND contents of RY with the contents of RZ and store the result in RX.
addi	I	addi $v0, $a0, 7	010	Add contents of RY to the contents of the offset field and store the result in RX.
lw	I	lw $v0, 0x07($sp)	011	Load RX from memory. The memory address is formed by adding the offset to the contents of RY.
sw	I	sw $a0, 0x07($sp)	100	Store RX into memory. The memory address is formed by adding the offset to the contents of RY.
beq	I	beq $a0, $a1, done	101	Compare the contents of RX and RY. If they are the same, then branch to address PC + 1 + Offset, where PC is the address of the beq instruction. **Memory is word addressable**.
jalr	J	jalr $at, $ra	110	First store PC + 1 in RY, where PC is the address of the jalr instruction. Then branch to the address in RX. If RX = RY, then the processor will store PC + 1 into RY and end up branching to PC + 1.
halt	O	halt	111	Halts execution of the program.

LC 2200 provides a number of pseudo-instructions.

Table 3: Assembly Language Pseudo-Instructions

Name	Example	Action
la	la $a0 MyLabel	Loads the address of a label into a register.
noop	noop	No operation, does nothing. It actually does add $zero, $zero, $zero.
.word	.word 32, .word MyLabel	Fills the memory location it is located with a given value or the address of the label.

[**0 points**] Play around with the simulator. Try writing some simple programs to copy values from one register to another or to load/store values from memory. You should get familiar with the syntax for the assembler. When you extract the archive, you will have access to an assembler (assembler.py), a python file that defines the ISA for the assembler (lc2200hw1-isa.py), and a simulator (lc2200hw1-sim.py). The assembler and simulator run on any version of Python 2.6+. Here is the suggested work ow for writing and running your assembly programs on the simulator:

1. Edit and save your assembly file with your favorite text editor.
2. Assemble your code via the assembler. The command below will generate a file called myFile.bin by using a myFile.s assembly file and the ISA for this homework.
   `python assembler.py -i lc2200hw1-isa --sym myFile.s`
3. Run the simulator with the generated .bin file. The command below will simulate the execution of myFile.bin on a computer running the ISA for this homework.
   `python lc2200hw1-sim.py myFile.bin`

Problem 2: Factorial Test Program

In this problem, you have to use the LC 2200 assembly language to write a simple program.

1. [**30 points**] Answer the following questions based on the LC-2200 calling convention defined in the lecture slides and Chapter 2 of the textbook.
   (a) [**10 points**] Generally, what is the purpose of an ISA in terms of computer hardware and software?
   (b) [**10 points**] Use of the stack in the calling convention
      (i) What does the stack pointer point to?
      (ii) In which way does the stack grow in terms of memory addresses?
   (c) [**10 points**] Define the mechanics of a function call, including what the caller does to initiate a procedure call, what the callee does at the beginning of a procedure, what the callee does at the end of a procedure to return to the caller, and what the caller does to clean up after the procedure returned.
2. [**70 points**] Write a function in LC-2200 to compute factorial(num). Your function is required to follow the LC-2200 calling convention you just answered questions for. It should work for n$\geq$0, but you don't have to handle detecting/handling integer overflow. YOUR FUNCTION MUST BE RECURSIVE, PURELY ITERATIVE SOLUTIONS WILL NOT RECEIVE ANY CREDIT! YOU MUST USE THE STACK AND STACK POINTER TO IMPLEMENT RECURSION FOR FULL CREDIT! We recommend making a helper multiply function. Multiply can be done iteratively. Feel free to ask us questions in our oce hours, on Piazza, or in the weekly recitation. Make sure that fact.s is in a UNIX-readble format (no DOS/Windows nonsense). [Hint: If you are having trouble writing it out directly in assembly, write out the functions in C first and then proceed]

1 Deliverables

Turn in ALL of your files in T-Square in .tar.gz format.

- **answers.txt**, which has your answers for Problem 2, question 1.
- **fact.s**, which has your assembly code.
- **assembler.py**, the assembler.
- **lc2200hw1-isa.py**, this homework's ISA definition for the assembler.

- **lc2200hw1-sim.py**, this homework's simulator.

  The TAs should be able to simply type the following two commands to run your code.

  ```
  python assembler.py -i lc2200hw1-isa --sym fact.s
  python lc2200hw1-sim.py fact.bin
  ```

  If you cannot do this with your extracted submission, then you have done something wrong.