Project 1

Due Date: April 15, 11:59 PM

You are not allowed to take or give help in completing this project. **No late submission will be accepted**.

In this project you will create a simple MIPS simulator which will perform the following two tasks:

- Load a specified MIPS text file[1] and generate the assembly code equivalent to the input file (**disassembler**). Please see the sample input file and disassembly output.
- Generate the instruction-by-instruction simulation of the MIPS code (**simulator**). It should also produce/print the contents of *registers* and *data memories* after execution of each instruction. Please see the sample simulation output file.

You do not have to implement any exception/interrupt handling during simulation for this project.

Instructions

For reference, please use the MIPS Instruction Set Architecture PDF (available from class project1 webpage) to see the format for each instruction **and pay attention to the following changes.**

Your disassembler/simulator need to support the two categories of MIPS instructions shown in **Figure 1.**

Category-1	Category-2
* J, JR, BEQ, BLTZ, BGTZ	* ADD, SUB, MUL, AND
* BREAK	* OR, XOR, NOR
* SW, LW	* SLT
* SLL, SRL, SRA	* ADDI
* NOP	* ANDI, ORI, XORI

Figure 1: Two categories of instructions

The format of Category-1 instructions is described in **Figure 2**. If the instruction belongs to **Category-1**, the first two bits (leftmost bits) are always "**01**" followed by **4 bits** Opcode. Please note that instead of using 6 bits Opcode in MIPS, we use 4 bits Opcode as described in **Figure 3**. The remaining part of the instruction binary is exactly the same as the original MIPS instruction set.

01	Opcode (4 bits)	Same as MIPS instruction

Figure 2: Format of Instructions in Category-1

1: This is a text file consisting of 0/1's (not a binary file). See the sample input file sample.txt in the Project1 webpage.

Instruction	Identification bits
J	0000
JR	0001
BEQ	0010
BLTZ	0011
BGTZ	0100
BREAK	0101
SW	0110
LW	0111
SLL	1000
SRL	1001
SRA	1010
NOP	1011

Figure 3: Opcode for Category-1 instructions

Please pay attention to the exact description of instruction formats and its interpretation in MIPS instruction set manual. *For example, in case of **J** instruction, the 26 bit instruction_index is shifted left by two bits (padded with 00 at LSB side) and then the leftmost (MSB side) four bits of the delay slot instruction are used to form the four bits (MSB side) of the target address. Similarly, for **BEQ**, **BLTZ** and **BGTZ** instructions, the 16 bit offset is shifted left by two bits to form 18 bit signed offset that is added with the address of the delay slot instruction to form the target address.*

If the instruction belongs to **Category-2**, the first two bits (leftmost two bits) are always "11". Then the following 4 bits serve as identification bits which are specified in **Figure 4**.

Instruction	Identification bits
ADD	0000
SUB	0001
MUL	0010
AND	0011
OR	0100
XOR	0101
NOR	0110
SLT	0111
ADDI	1000
ANDI	1001
ORI	1010
XORI	1011

Figure 4: Identification bits for Category-2 instructions

Instructions in Category-2 can be further divided into two sub-sets according to whether source2 is register or immediate. When the source2 is register, ("rd ← rs op rt"), the format is shown in **Figure 5**.

11	identification bits (4 bits)	rs (5 bits)	rt (5 bits)	rd (5 bits)	00000000000

Figure 5: Format of Category-2 instructions with source2 as register

When source 2 is an immediate ("rt ← rs op immediate_value"), the format is shown in **Figure 6**.

11	identification bits (4 bits)	rs (5 bits)	rt (5 bits)	immediate_value (16 bits)

Figure 6: Format of Category-2 instructions with source2 as register

Sample Input/Output Fils

Your program will be given a binary (text) input file. This file will contain a sequence of 32-bit instruction words which begin at address **"256"**. The final instruction in the sequence of instructions is always BREAK. Following the BREAK instruction (immediately after BREAK), there is a sequence of 32-bit 2's complement signed integers for the program data up to the end of the file. The NEWLINE character can be either "\n" (linux) or "\r\n" (windows). Your code should work for both cases.

Your MIPS simulator (with executable name as **MIPSsim**) should accept an input file (**inputfilename.txt**) in the following command format and produce two output files **in the same directory**: **disassembly.txt** (contains disassembled output) and **simulation.txt (**contains the simulation trace). You can hardcode the names of the output files. Please do not hardcode the input filename. It will be specified when running your program. It can be "sample.txt" or "t1.txt".

MIPSsim inputfilename.txt

Correct handling of the sample input file (with possible different data values) will be used to determine 60% of the credit. The remaining 40% will be determined from other valid test cases that you will not have access prior to grading. It is recommended that you construct your own sample input files with which to further test your disassembler/ simulator.

The disassembler output file should contain 3 columns of data with each column separated by one tab character ('\t' or char(9)). See the sample disassembly file in the class Project1 webpage.

1. The text (e.g., 0's and 1's) string representing the 32-bit data word at that location.

2. The address (in decimal) of that location

3. The disassembled instruction.

Note, if you are displaying an instruction, the third column should contain every part of the instruction, with each argument separated by a comma and then a space (", ").

The simulation output file should have the following format.
* 20 hyphens and a new line
* Cycle: < cycleNumber > < tab >< instr_Address >< tab >< instr_string >
* < blank_line >
* Registers
* R00:< tab >< int(R0) >< tab >< int(R1) >...< tab >< int(R7) >
* R08:< tab >< int(R8) >< tab >< int(R9) >...< tab >< int(R15) >
* R16:< tab >< int(R16) >< tab >< int(R17) >...< tab >< int(R23) >
* R24:< tab >< int(R24) >< tab >< int(R25) >...< tab >< int(R31) >
* < blank_line >
* Data
* < firstDataAddress >:< tab >< display 8 data words as integers with tabs in between >
* ..... < continue until the last data word >

The instructions and instruction arguments should be in capital letters. Display all integer values in decimal. Immediate values should be preceded by a "#" symbol. **Note that some instructions take signed immediate values while others take unsigned immediate values**. You will have to make sure you properly display a signed or unsigned value depending on the context.

Because we will be using "**diff –w -B**" to check your output versus ours, please follow the output formatting. TA may not be able to debug in case of any mismatch. In other words, mismatches can be treated as wrong output.

The course project webpage contains the following sample programs/files to test your disassembler/simulator.

- sample.txt : This is the input to your program.
- disassembly.txt : This is what your program should produce as disassembled output.
- simulation.txt : This is what your program should output as simulation trace.

Submission Policy:

Please follow the submission policy outlined below. There can be significant **score penalty** based on the nature of submission policy violations.
1. You are not allowed to take or give any help in completing this project.

2. Please submit only one source file (*see next comment if you have multiple source files*). Your file name must be MIPSsim (e.g., MIPSsim.c or MIPSsim.cpp or MIPSsim.java). Please do not submit any object files. On top of the source file, please include the sentence: "/* On my honor, I have neither given nor received unauthorized aid on this assignment */".

3. If you decide to submit more than one source files, Please use "**tar cvf project1.tar file1 file2 file3…"** to pack your source files. Please do not put your files in a separate

folder and pack. Please provide a **makefile** to compile your code.

4. Please test your submission before your submission.