Project 2

CDA 3101: Fall 2014

Due Date: 11/25/2014 (Tuesday), 11:30 pm

You are not allowed to take or give help in completing this project. **No late submission will be accepted**. Please include the following sentence on top of your source file: "**On my honor, I have neither given nor received unauthorized aid on this assignment**".

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 and simulator need to support the three categories of MIPS instructions shown in **Figure 1.**

Category-1	Category-2	Category-3
* J, BEQ, BGTZ * BREAK, SW, LW	* ADD, SUB, MUL, AND * OR, XOR, NOR	* ADDI * ANDI, ORI, XORI

Figure 1: Three 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 "**00**" 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 for that specific instruction.

00	Opcode (4 bits)	Same as MIPS instruction

Figure 2: Format of Instructions in Category-1

Instruction	Opcode
J	0000
BEQ	0010
BGTZ	0100
BREAK	0101
SW	0110
LW	0111

Figure 3: Opcode for Category-1 instructions

[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.

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 address of the delay slot (next) instruction are used to form the four bits (MSB side) of the target address. Similarly, for **BEQ** 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 (next) instruction to form the target address. See the sample files to understand how it was done for specific cases.*

If the instruction belongs to **Category-2** which have the form "rd ← rs op rt", the first two bits (leftmost two bits) are always "01". Then the following 5 bits serve as rs. Then 5 bits for rt. Then 4 bits for opcode as indicated in **Figure 4**. The instruction format is shown in **Figure 5**.

Instruction	Opcode
ADD	0000
SUB	0001
MUL	0010
AND	0011
OR	0100
XOR	0101
NOR	0110

Figure 4: Opcode for Category-2 instructions

11 bits

01	rs (5 bits)	rt (5 bits)	opcode (4 bits)	rd (5 bits)	~~00000000000~~

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

If the instruction belongs to **Category-3** which has the form "rt ← rs op immediate_value", the first two bits (leftmost two bits) are always "10". Then the following 5 bits serve as rs. Then 5 bits for rt. Then 4 bits for opcode as indicated in **Figure 6**. The instruction format is shown in **Figure 7**.

Instruction	Opcode
ADDI	0000
ANDI	0001
ORI	0010
XORI	0011

Figure 6: Opcode for Category-3 instructions

10	rs (5 bits)	rt (5 bits)	opcode (4 bits)	immediate_value (16 bits)

Figure 7: Format of Category-3 instructions with source2 as immediate value

Sample Input/Output Fils

Your program will be given a text input file (see sample.txt). This file will contain a sequence of 32-bit instruction words which begin at address **"128"**. 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. DON'T JUST COPY THE CONTENT OF THE SAMPLE FILES AND PASTE IN YOUR OWN TEXT FILES. INSTEAD, DOWNLOAD THEM (RIGHT CLICK ON THE LINK AND CHOOSE "SAVE AS").

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. For example, it can be "sample.txt" or "test.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.

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. Mismatches will 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. Please submit only one source file. **Please add ".txt" at the end of your filename**. Your file name must be MIPSsim (e.g., MIPSsim.c.txt **or** MIPSsim.cpp.txt **or** MIPSsim.java.txt). On top of the source file, please include the sentence: "/* On my honor, I have neither given nor received unauthorized aid on this assignment */".
2. Please test your submission. These are the exact steps we will follow too.
    - Download your submission from eLearning (ensures your upload was successful).
    - Remove ".txt" extension (e.g., MIPSsim.c.txt should be renamed to MIPSsim.c)
    - Login to **thunder.cise.ufl.edu**. If you don't have a CISE account, go to http://cise.ufl.edu/help/account.shtml and apply for one CISE class account. Then you use putty and winscp or other tools to login.
    - Please compile to produce an executable named **MIPSsim**.
        - gcc MIPSsim.c –o MIPSsim **or** javac MIPSsim.java **or** g++ MIPSsim.cpp –o MIPSsim
    - Please do not print anything on screen.
    - Please do not hardcode input filename, accept it as command line option.
    - Execute to generate disassembly and simulation files and test with correct ones
        - ./MIPSsim inputfilename.txt
        - diff –w –B generated_disassembly.txt correct_disassembly.txt
        - diff –w –B generated_simulation.txt correct_simulation.txt
3. *In previous years, there were many cases where output format was different, filename was different, command line arguments were different, or e-Learning submission was missing, …. All of these led to un-necessary frustration and waste of time for TA, instructor and students*. **Please use the exactly same commands as outlined above to avoid any score penalty.**
4. **You are not allowed to take or give any help in completing this project**. *In previous years, some students violated academic honesty (giving help or taking help in completing this project). We were able to establish cheating in several cases - those students received "0" in the project and their names were reported to UF Ethics office.*