**Projects**

For the course project, you will create a simple MIPS simulator. Your MIPS simulator will be capable of loading a specified MIPS binary file and outputting the assembly code equivalent or the cycle-by-cycle simulation of the MIPS binary code. The project is **individual project*.***That is, there must be one project turned in for each student in the class.

The project is split into two parts.

* Part I will be to create the disassembler. Your program will be capable of loading a provided binary file and displaying the MIPS assembly code equivalent along with the binary code.  (Details is through PartI link below.)
* Part II will be to develop a cycle-by-cycle MIPS simulator (Details is through PartII link below.)

**Submission**

You may write your simulator using any tools available on the CISE SunOS environment, including *gcc, perl, php,*and *java*. You must provide a*Makefile* which will allow us to correctly build your project irrespective of the tools you selected. You must also provide a *README* containing your full name, gatorlink account, e-mail address, and any special instructions to the grader or notes about any assumptions you made about the project that the grader should take into consideration. You must submit these files, along with all the files necessary to build and run your project, in a single *jar* file to Sakai. You should make a separate directory to work on your project and then use *jar -cvf project.jar \**to create your jar file for Sakai. Please only submit the .jar file to Sakai. Submission of multiple files makes grading time-consuming and will not be accepted.

You need to include the honor code in your *README*:  ``*On my honor, I have neither given nor received unauthorized aid on this assignment.''*

**Instructions**

For reference, please use the MIPS Instruction Set Architecture file ([mips.pdf](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/mips.pdf)) to see the format for each instruction.

Your disassembler/simulator will need to support the following MIPS instructions:

* SW, LW
* J, BEQ, BNE, BGEZ, BGTZ, BLEZ, BLTZ
* ADDI, ADDIU
* BREAK
* SLT, SLTI , SLTU
* SLL, SRL, SRA
* SUB, SUBU, ADD, ADDU
* AND, OR, XOR, NOR
* NOP

**Input**

Your program will be given a binary input file. This file will contain a sequence of 32-bit instruction words which begin at address "584". The final instruction in the sequence of instructions is always BREAK. The data section is followed the BREAK instruction and begins at address "700". Following that is a sequence of 32-bit 2's complement signed integers for the program data up to the end of file. Note that the instruction words always start at address "584". If the instructions occupy beyond address "700", the data region will be followed the BREAK instruction immediately.

Your MIPS simulator (*MIPSsim)* should provide the following options to users:

MIPSsim *inputfilename outputfilename operation*[-T*m:n*]

* *Inputfilename* - The file name of the binary input file.
* *Outputfilename* - The file name to which to print the output.
* *Operation* - Either"dis" or"sim" to specify disassembly or simulation.
* -T*m:n* - Optional argument to specify the start (*m)* and end (*n)* cycles of simulation tracing output. Tracing should be done in a single-step fashion with the contents of registers and memory shown after every processor cycle. -T0:0 indicates that no tracing is to be performed; eliminating the argument specifies that every cycle is to be traced.

Your program should output a simple help text describing proper usage if the user fails to specify valid arguments.

Correct handling of the sample code (with possible different data values) will be used to determine 60% of credit awarded. The remaining 40% will be determined from other test cases that you will not have access to prior to grading.

[**Part I - Disassembler**](file:///E:\ta\cda5155sp07\public_html\project1\index.html)

**Due Date for on-campus students:**Sep 22nd, 2012 11:55pm

**Due Date for EDGE students:**Sep 23rd, 2012 11:55pm

For this first part of the project, your *MIPSsim* program will accept a binary input file *inputfilename*and given *operation* **dis** will output to*outputfilename* the MIPS disassembly of the input file.

The disassembler output file will contain 4 columns of data with each column separated by one space character (' '):

* The binary (e.g., 0's and 1's) string representing the 32-bit data word at that location. For instructions you should split this into six groups of digits to represent different parts of the MIPS instruction word: a group of 6 bits, 4 groups of 5 bits, and a final group of 6 bits.
* The address (in decimal) of that location.
* The disassembled instruction opcode+other fields, or a signed decimal integer value, depending on whether the current location is after the BREAK instruction.
* If you are displaying an instruction, the fourth column should contain the remaining part of the instruction, with each argument separated by a comma and then a space. (", ")

The instructions and instruction arguments should be in capital letters. Display all integer values in decimal. Immediate values should be proceeded by a "#" symbol. Be careful 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' to check your output versus ours, try to follow the exact output format.

**Sample Data**

Here is a sample program to test your disassembler with.

* [fibonacci\_c](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/fibonacci_c.txt) : This contains the C source code for the test program. This is for your reference only.
* [fibonacci\_mips](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/fibonacci_mips.txt) : This is the compiled version of the C code in MIPS assembly. This is for your reference only.
* [fibonacci\_bin](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/fibonacci_bin.bin) : **This is the assembled version of the above assembly code. It is the input to your program**.
* [fibonacci\_out](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/fibonacci_out.txt) : **This is what your program should output given the above binary input file**.
* [fibonacci\_bin\_txt](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/fibonacci_bin_txt.txt) : This is the txt version of the input binary file. You can use the following program to convert it to binary or use the binary file provided above.

Remember that we will also test your program with other data that you will not know of in advance. It is recommended that you construct your own sample input files with which to further test your disassembler. For your convenience we have provided a program to convert a text file of binary words into a binary file with which to test your disassembler.

* [txt2bin.cpp](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/txt2bin.cpp) (click right to save) This program will convert a text file of binary words into a binary file. This program has been compiled and tested on CISE thunder machince. Because of Endian differences on different platforms, Please compile and run this program ONLY ON CISE thunder machine.
* [fibonacci\_bin\_txt](http://www.cise.ufl.edu/class/cda5155fa12/projects/project1/fibonacci_bin_txt.txt):  Running txt2bin on this file should produce the Assembler binary file for the sample data above.