**Project #1**

1. Project Objective
   1. Implement a single-cycle, functional processor simulator according to the reduced MIPS R3000 ISA specified in *Appendix A*, “*Datasheet for the Reduced MIPS R3000 ISA*.”
   2. Design a test case on your own to test the functionality of your simulator.
2. About the Simulation
3. The execution of the single-cycle processor simulator should **terminate after executing the “halt”** instruction.
4. Both the instruction memory and data memory are in **Big-endian** format.
5. **Register 0 is a hard-wired constant 0**; any attempt to write to register 0 takes no effect.
6. Assume that both the instruction memory and data memory are of 1K bytes size.
7. You should implement an **Error handler** to deal with erroneous instruction executions.

For more details please refer to *Appendix D*, “*Error Detection Samples*.”

1. The simulator should get input files and generate output files at the same directory as that of the executable. Note that the executable should be named **single\_cycle** and takes no command-line arguments.
2. Input Format

For each test case, **all registers, except PC and $sp, are initialized to 0’s**; other initial conditions are specified in the following two files.

* 1. **iimage.bin**: The instruction image (Big-endian format, encoded in binary). The first four bytes indicate the initial value of PC, **i. e. the starting address to load the instruction image**. The next four bytes specify the number of words to be loaded into instruction memory (I memory). The remaining are the program text to be loaded into I memory. All other addresses not covered by the image are assumed to have been initialized to 0’s.
  2. **dimage.bin**: The data image (Big-endian format, encoded in binary). The first four bytes show the initial value of $sp. The second four bytes indicate how many words to be loaded into data memory (D memory). The remaining words are to be loaded into D memory, starting from address 0. The contents of all other addresses not covered by the image are assumed to have been initialized to 0’s.

For more details regarding input format, please refer to *Appendix B*, “*Input Samples*.”

1. Output Requirement

For each test case, please output to **snapshot.rpt and Error\_dump.rpt**, which should contain all register values at each cycle, and error message respectively. For details please refer to *Appendix C*, “*Output Samples for Project 1”* and *Appendix D*, *“Error Detection Sample*”.

1. Test case design
   1. Your test case should be of the same format as described in list 3 and *Appendix B*, and it should be placed into a folder named *testcase*.
   2. For **iimage.bin**, note that initial value of PC is also the starting point of the execution of your program.
   3. There are 2 files should be in the folder:”iimage.bin”,”dimage.bin”
   4. Your testcase should run no more 30K cycles.

1. Evaluation
   1. Correctness of simulator: 50%

We will take the average of the two versions you submit, and for each submission

* + 1. From TAs: 20%
    2. From students: 30%
  1. Report: 20%
     1. Report is limited to 10 pages.
     2. Your report can be either in Chinese or English, or mixed. The report should be named archiXX\_report.doc, where XX is your group ID.
     3. Grading principle:
        1. Simulator design (5%): Discuss if any special data structures, design patterns, code structure, and general execution flow, etc.
        2. Simulator elaboration (5%): Clearly explain your design with tables, figures , etc.
        3. Testcase design (5%): How you implement your testcase in C code and assembly code? What corner cases are you targeting at? We will also look for your creativity of the algorithms used for your testcase.
        4. Testcase elaboration (5%): Comments to both C code and assembly code of your testcase; the mapping between variables and register/memory locations and that between codes and labels; etc.
  2. Testcase: 30% \* (1 – [1.5]-n), n: number of defeated groups

**You get zero points if your test case is invalid.** A test case is valid if it passes TA’s simulator and your own simulator.

You can improve your test case and resubmit in second due. The grading rule will be

1. If your test case is not updated
   * If it is valid in first submission: we use the test case score in this submission.
   * If it becomes valid after fixing your simulator in the second submission, the final score of your test case will be 80% of the test score of your test case against all simulators from the first submission.
   * If it is still invalid, the test case gets zero points.
2. If your test case is updated
   * The final score of your new test case will be 80% of the test score of your test case against all simulators from the *second* submission.
   1. Bonus for early submission

If you submit your project *x* days earlier to the first due date and suppose the project duration is *y* days long (to the first due date), then you will have a bonus of

20% \* *t* \* *x* / *y* points,

where *t* is the total points of your first submission.

Note that we use **nthucad workstation** as our evaluating environment.

Furthermore, we will evaluate your project using scripts. It is your responsibility to make sure that your project can be executed by the script provided by TAs. **If it cannot run through the script, you will get zero points even if your program or result is correct.**

1. Submission

Please put **all of your source codes, folder of test case, report** and ***Makefile*** into a folder named *archiXX*, compress the folder *archiXX* as *archiXX.tar.gz*, and upload *archiXX.tar.gz* to iLMS system. For example, if you are of group 0, then you should upload *archi00.tar.gz* to iLMS system.

1. **Etiquette**
   1. **Do not copy others’ works, or you will fail this course.**
   2. **No acceptance of late homework.**
   3. **Please constantly check the class website announcements for possible updates.**