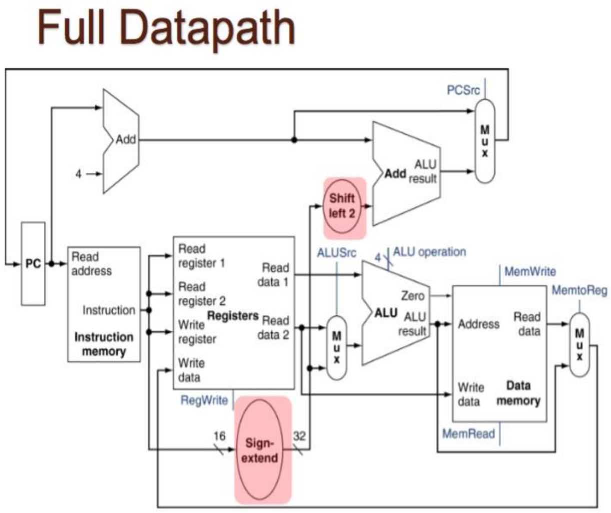
10520 CS410001 - Computer Architecture 2017

Project #1

1. Project Objectives
   a. Implement a single-cycle functional processor simulator.
   b. Design your own test case to test the functionality of your simulator and your classmates' simulator.

2. Project Description



   a. Architecture Design:
      - Refer to textbook *Chapter 2 "Instructions"* and *Chapter 4.1~4.4 "The Processor"*.

   b. Instruction Set:
      - Implement all the instructions specified in the reduced MIPS R3000 ISA in ***Appendix A*, "*Datasheet for the Reduced MIPS R3000 ISA*"**.

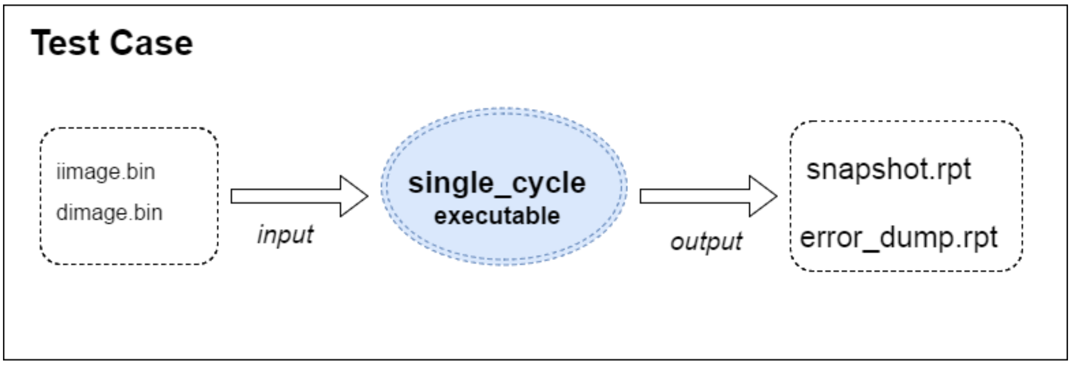
   The execution of the single-cycle processor simulator should **terminate** after executing the **"halt" instruction**.

   c. Processor Simulator Specification

   (i) Implementations

   - The simulator should be coded in C/C++ and be compiled with *makefile*.
   - The simulator source code will be compiled into a executable file should be named "single_cycle"
   - The simulator executes with no command-line argument.

   (ii) Data Flow Diagram



- The simulator reads "*iimage.bin*" and "*dimage.bin*" as input and write "*snapshot.rpt*" and "*error_dump.rpt*" as output report files. Both "*iimage.bin*" and "*dimage.bin*" constitute a test case.

(iii) Memory

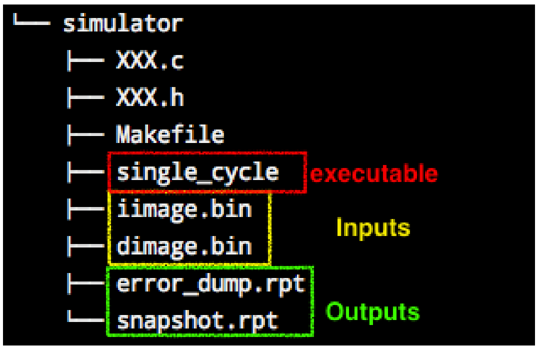
- The simulator have instruction memory (I memory) and data memory (D memory)
- Both of **D-memory** and **I-memory** are 1K bytes and are initialized to 0's by default.
- I memory stores MIPS instruction and is initialized through the content of *iimage.bin*.
- D memory stores data and is initialized through the content of *dimage.bin*.
- For format details, please refer to ***Appendix B*, "*Input Samples*"**.

(iv) Registers

- The simulator should include 32 general purpose register ($0~$31), LO, HI, PC and stack pointer ($sp).
- All 32 registers, LO and HI, except PC and $sp, are initialized to 0's.
- PC is initialized in *iimage.bin*, and $sp is initialized in *dimage.bin*.
- For format details, please refer to ***Appendix B*, "*Input Samples*"**.

(v) Simulator Read/Write Location

- The simulator writes out the output files (*snapshot.rpt* and *error_dump.rpt*) and read in the input files (*iimage.bin* and *dimage.bin*) at the same directory where your executable file resides.



```
└── simulator
    ├── XXX.c
    ├── XXX.h
    ├── Makefile
    ├── single_cycle
    ├── iimage.bin
    ├── dimage.bin
    ├── error_dump.rpt
    └── snapshot.rpt
```

(vi) Unrecognized Instructions

- If the PC points to an unrecognized instruction, not specified in our list, please print out "illegal instruction found at *0xaddress*" . Then terminate the execution.

d. Input Test Case Files

- The test case which includes *iimage.bin* and *dimage.bin* should be written in **binary format.**
- For format details, please refer to ***Appendix B*, "*Input Samples*"**.

e. Output Files

- For each test case, generate the following two output files:
  1. ***snapshot.rpt***: record all the register values at each cycle.
  2. ***error_dump.rpt***: record any error messages.
- For details, please refer to ***Appendix C-1, "Output Samples for Project 1"*** and ***Appendix D, "Error Detection Samples"***.

f. Design Your Test Case

(i) Valid Test Case Definition

- The test case you designed should pass both **your own simulator** and the **golden simulator** with the same output in order to be considered valid.
- The test case you designed should run no more than **500,000 cycles**.
- The test case allows no **address overflow** or **misaligned access** in **I memory**.

(ii) Test Case Competition

- TA will use all test cases collected from the class to evaluate your simulator. Your valid test case gets higher grade if more simulators fail running your test case.

g. Modularized implementation (Recommendation)

- Suggest that you should **modularize** your simulator implementation based on the given processor architecture. For example, this is a possible program structure:
  a. simulator.c // Define simulator behaviors and main function
  b. instruction.c // Define & decode instructions
  c. regfile.c // Register function
  d. memory.c // Memory function (for both instruction & data memories)
  e. etc.c…… // other miscellaneous functions
- Appropriate header file or object-oriented programming format are also highly recommended design pattern.

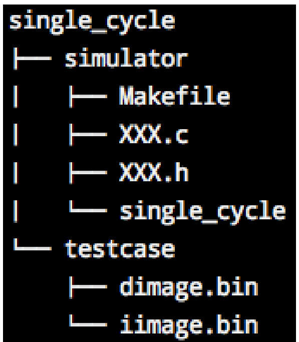
3. Project Submission Rules

a. There are two submissions for each project. The second due date is normally one week after the first one.

- Before 1st submission: we will release a **golden executable** and **open test cases** to help you verify your designs.
- 1st submission: submit your simulator and test case for evaluation.
- After 1st submission (before 2nd submission): we will release **all test cases (including hidden test cases and all test cases submitted by the class)** for you to polish your simulator.
- 2nd submission: submit your revised simulator and project **report** (format is specified in the Grading Policy section).

b. Prepare your project package for development and submission

- Before you start your project
  1. Use SSH to access workstation.
  2. Clone sample files from GitHub to your home directory.
     - Clone a repository NTHU Architecture 2017 from GitHub



```
single_cycle
├── simulator
|   ├── Makefile
|   ├── XXX.c
|   ├── XXX.h
|   └── single_cycle
└── testcase
    ├── dimage.bin
    └── iimage.bin
```

and name it as **single_cycle/** under **/home/archi/student*ID***. Inside the folder, it contains two sub-folders:

  i. **simulator/** : contains your *Makefile*, and source code.
    a. Your *Makefile* should support the following two functions:
      - make – to build your simulation environment
      - make clean – to erase from the build tree the files built by make all.
    b. Modulize your source code as recommended.
  ii. **testcase/** : contains your test case files for evaluation.

3. Do coding/debugging/testing using the Git version control tool.

- Before submission and after completing your project
  1. Check your output file format using test_script.py
  2. Compress the folder **single_cycle** as **single_cycle.tar.gz**, and upload **single_cycle.tar.gz** and **student*ID*_report.pdf** to the iLMS system.

- **Note:** TAs will check your Git log during the 1-on-1 Demo. Please follow version control rule when doing programming.
- **Note:** Verify your package format by executing *test_script.py* before each submission. **Wrong submission format earns no points.**

4. Grading Policy
  a. First submission
    - Correctness of simulator: 25%
      1. TA’s open test cases: 15%
      2. TA’s hidden test cases: 5%*Correct Ratio
      3. Students’ valid test cases: 5%*Correct Ratio
    - Test case strength: 20%*(1 – $[1.5]^{-n}$), n: number of other simulators defeated
      1. Submit your test case to participate in the pool test.
      - **Note:** You get zero points if your test case is invalid.

  b. Second submission
    - Correctness of simulator: 30%
      1. TA’s open test cases: 5%
      2. TA’s hidden test cases: 10%*Correct Ratio
      3. Students’ valid test cases: 15%*Correct Ratio
    - Performance: 5%
      1. TA will collect the execution time of your simulator running all the valid test cases (including all open, hidden and students’ test cases).

2. TA will rank all execution times in five levels and grade accordingly.
   - **Note:** If your simulator fails to execute the valid test cases, you will get the lowest performance grade.

- Report & Demo: 20%
  1. The report file should be named ***studentID_report.pdf***, where ***studentID*** is the NTHU student id you used for school registration.
  2. Each person should reserve a 15-min demo with TA. During the 1-on-1 demo, TA’s will ask you questions related to your project report, test case and your implemented code on workstation.
  3. Your project report is recommended to follow this outline:

     1) Project Description
         1-1) Program Flow Chart
         1-2) Detailed Description
     2) Test case Design
         2-1) Detailed Description of Test case

  Note: The project report is limited to 10 pages.
  Note: Your report can be either in Chinese or in English, or mixed.
  Note: For convenience, please synchronize your submission code with your code in workstation. This allows smoother demo on workstation.

Etiquette

a. Do not plagiarize others’ work, or you will fail this course.

b. No acceptance of late homework.

c. Please frequently check the class website announcements for possible updates.