# Project 1

## CDA 5155: Fall 2015

## Due: Oct 07, 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 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 five categories of instructions shown in **Figure 1.**

| Category-1 | Category-2 | Category-3 | Category-4 | Category-5 |
|---|---|---|---|---|
| J, BEQ, BNE, BGTZ, SW, LW, BREAK | ADD, SUB, AND, OR, SRL, SRA | ADDI, ANDI, ORI | MULT, DIV | MFHI, MFLO |

**Figure 1: Five categories of instructions**

The format of **Category-1** instructions is described in **Figure 2**. If the instruction belongs to **Category-1**, the first three bits (leftmost bits) are always "**000**" followed by **3 bits** Opcode. Please note that instead of using 6 bits opcode in MIPS, we use 3 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.

| | | |
|---|---|---|
| 000 | Opcode (3 bits) | Same as MIPS instruction |

**Figure 2: Format of Instructions in Category-1**

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 next instruction are used to form the four bits (MSB side) of the target address. Similarly, for **BEQ, BNE** 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 next instruction to form the target address.* **Please note that we do not consider *delay slot* for this project.** In other words, an instruction following the branch instruction should be treated as a regular instruction (see *sample_simulation.txt*).

[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 | Opcode |
|---|---|
| J | 000 |
| BEQ | 001 |
| BNE | 010 |
| BGTZ | 011 |
| SW | 100 |
| LW | 101 |
| BREAK | 110 |

**Figure 3: Opcode for Category-1 instructions**

If the instruction belongs to **Category-2** which has the form "dest ← src1 op src2", the first three bits (leftmost three bits) are always "001" as shown in **Figure 4**. Then the following 5 bits serve as dest. The next 5 bits for src 1, followed by 5 bits for src2. The src1 is always register but src2 can be register (ADD, SUB, AND, OR) or immediate (SRL, SRA) depending on the opcode. The remaining bits are all 0's. The three bit opcodes are listed in **Figure 5**.

| 001 | opcode (3 bits) | dest (5 bits) | src1 (5 bits) | src2 (5 bits) | 00000000000 |
|---|---|---|---|---|---|

**Figure 4: Format of Category-2 instructions where both sources are registers**

| Instruction | Opcode |
|---|---|
| ADD | 000 |
| SUB | 001 |
| AND | 010 |
| OR | 011 |
| SRL | 100 |
| SRA | 101 |

**Figure 5: Opcode for Category-2 instructions**

If the instruction belongs to **Category-3** which has the form "dest ← src1 op immediate_value", the first three bits (leftmost three bits) are always "010". Then 3 bits for opcode as indicated in **Figure 6**. The subsequent 5 bits serve as dest followed by 5 bits for src1. The second source operand is immediate 16-bit value. The instruction format is shown in **Figure 7**.

| Instruction | Opcode |
|---|---|
| ADDI | 000 |
| ANDI | 001 |
| ORI | 010 |

**Figure 6: Opcode for Category-3 instructions**

| 010 | opcode (3 bits) | dest (5 bits) | src1 (5 bits) | immediate_value (16 bits) |
|---|---|---|---|---|

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

If the instruction belongs to **Category-4** which has the form (HI, LO) ← src1 op src2, the first three bits (leftmost three bits) are always "011". Then 3 bits for opcode as indicated in **Figure 8**. Then the following 5 bits serve as src1 followed by 5 bits for src2. The remaining bits are all 0's. The instruction format is shown in **Figure 9**. *Please see the MIPS manual to understand how HI and LO are used.*

| Instruction | Opcode |
|---|---|
| MULT | 000 |
| DIV | 001 |

**Figure 8: Opcode for Category-4 instructions**

| | | | | |
|---|---|---|---|---|
| 011 | opcode (3 bits) | src1 (5 bits) | src2 (5 bits) | 0000000000000000 |

**Figure 9: Format of Category-4 instructions with implicit destination of HI/LO registers**

If the instruction belongs to **Category-5** which has the form dest ← HI (or LO), the leftmost three bits are always "100". Then 3 bits for opcode as indicated in **Figure 10**. The next 5 bits serve as destination. The remaining bits are all 0's. The instruction format is shown in **Figure 11**.

| Instruction | Opcode |
|---|---|
| MFHI | 000 |
| MFLO | 001 |

**Figure 10: Opcode for Category-3 instructions**

| | | | |
|---|---|---|---|
| 100 | opcode (3 bits) | dest (5 bits) | 00000000000000000000 |

**Figure 11: Format of Category-5 instructions with only destination operand**

# Sample Input/output Files

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 **"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. *Please download the sample input/output files using "Save As" instead of using copy/paste of the content.*

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.

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

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

## Submission Policy:

Please follow the submission policy outlined below. There can be up to **10% 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 https://www.cise.ufl.edu/help/account#apply 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 a command line option.
    - Execute to generate disassembly and simulation files and test with correct/provided ones
        - ./MIPSsim inputfilename.txt **or** java MIPSsim inputfilename.txt
        - diff –w –B disassembly.txt sample_disassembly.txt
        - diff –w –B simulation.txt sample_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, etc. 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 10% score penalty.**
4. **You are not allowed to take or give any help in completing this project**. *In the previous years, some students violated academic honesty (giving help or taking help in completing this project). We were able to establish violation in several cases - those students received "0" in the project and their names were reported to UF Ethics office. This year we would also impose one additional letter grade penalty. Someone could potentially lose two letter grade points (e.g., "A-" grade to "B" grade) – one for getting 0 score in the project and then another grade point penalty on top of it. Moreover, the names of the students will also be reported to UF Dean of Students Office (DSO). If your name is already in DSO for violation in another course, the penalty for second offence is determined by DSO. In the past, two students from my class were suspended for a semester due to repeat academic honesty violation (implies deportation for international students).*