Lab 1: Single Cycle MIPS

In this Lab assignment, you will implement an instruction-level simulator for a single cycle MIPS processor in C++ step by step. The simulator supports a subset of the MIPS instruction set and can model the execution of each instruction.

A MIPS program is provided for the simulator as a text file “imem.txt” file, which is used to initialize the Instruction Memory. Each line of the file corresponds to a Byte stored in the Instruction Memory in binary format, with the first line at address 0, the next line at address 1 and so on. Four contiguous lines correspond to a whole instruction. Note that the words stored in memory are in “Big-Endian” format, meaning that the most significant byte is stored first.

We have defined a “halt” instruction as 32’b1 (0xFFFFFFFF), which is the last instruction in every “imem.txt” file. As the name suggests, when this instruction is fetched, the simulation is terminated. We will provide a sample “imem.txt” file containing a MIPS program. You are encouraged to generate other “imem.txt” files to test your simulator.

The Data Memory is initialized using the “dmem.txt” file. The format of the stored words is the same as the Instruction Memory. As with the instruction memory, the data memory addresses also begin at 0 and increment by one in each line.

The instructions that the simulator supports and their encodings are shown in Table 1. Note that all instructions, except for “halt”, exist in the MIPS ISA. The *MIPS Green Sheet*at “NYUClasses->Resources->Slides” defines the semantics of each instruction.

Name	Format Type	Opcode(Hex)	Func(Hex)
addu	R-Type	00	21
subu	R-Type	00	23
addiu	I-Type	09	
and	R-Type	00	24
or	R-Type	00	25
nor	R-Type	00	27
beq	I-Type	04	
j	J-Type	02	
lw	I-Type	23	
sw	I-Type	2B	
halt	J-Type	3F	

Table 1. Instruction encodings for a reduced MIPS ISA

Skeleton Code

The file "MIPS.cpp" contains a skeleton code for the assignment. You need to *fill in* the missing code. In this section, we provide descriptions for each of the components in the skeleton code.

Classes

We have defined four C++ classes that each implement one of the four major blocks in a single cycle MIPS, namely RF (to implement the register file), ALU (to implement the ALU), INSMem (to implement instruction memory), and DataMem (to implement data memory).

1. **RF class:** contains 32 32-bit registers defined as a private member. Remember that register $0 is always 0. Your job is to implement the *ReadWrite()* member function that provides read and write access to the register file.
2. **ALU class:** implements the ALU. Your job is to implement *ALUOperation()* member function that performs the appropriate operation on two 32 bit operands based on ALUOP. See Table 1 for more details.
3. **INSMem class:**a Byte addressable memory that contains instructions. The constructor *InsMem()* initializes the contents of instruction memory from the file imem.txt (this has been done for you). Your job is to implement the member function *ReadMemory()* that provides read access to instruction memory. An access to the instruction memory class returns 4 bytes of data; i.e., the byte pointed to by the address and the three subsequent bytes.
4. **DataMem class:** is similar to the instruction memory, except that it provides both read and write access.

Main Function

The main function defines a 32 bit program counter (PC) that is initialized to zero. The MIPS simulation routine is carried out within a while loop. In each iteration of the while loop, you will fetch one instruction from the instruction memory, and based on the instruction, make calls to the register file, ALU and data memory classes (in fact, you might need to make two calls to the register file class, once to read and a second time to write back). Finally you will update the PC so as to fetch the next instruction. When the halt instruction is fetched, you are to break out of the while loop and terminate the simulation.

Make sure that the architectural state is updated correctly after execution of each instruction. The architectural state consists of the Program Counter (PC), the Register File (RF) and the Data Memory (DataMem). We will check the correctness of the architectural state after *each* instruction.

Specifically, the OutputRF() function is called at the end of each iteration of the while loop, and will add the new state of the Register File to "RFresult.txt". Therefore, at the end of the program execution "RFresult.txt" contains all the intermediate states of the Register File. Once the program terminates, the OutputDataMem() function will write the final state of the Data Memory to "dmem.txt". These functions have been implemented for you. Do not modify them.
(Note: **You should delete the "RFresult.txt" file before re-executing your program**, otherwise the new results will append to the previous results.)

What You Have to Submit

1. We have provided skeleton code in the file MIPS.cpp. Finish the code. **Submit the MIPS.cpp to "NYU Classes – Assignment".**
    - You can use any development environment to write and test your code but we use Visual studio 2019 which your can refer and we will test the code on it.
    - On Visual Studio 2019, you can compile your design by clicking"**Build--compile "** which would create the MIPS executable. Run the executable by clicking "**Debug–start Debugging**".
    - Code that does not compile will automatically be given a 0.
2. We have provided "imem.txt" and "dmem.txt" files containing a sample code (loading two variables and adding them) and initialized data. These files must be in the same directory as the source code.
3. We encourage you to write your own MIPS programs and check your design for different cases.
4. This lab can be finished by **a group of 1-2** students.

Some useful references for this lab:

1. A brief introduction to C++ (https://web.eecs.umich.edu/~sugih/pointers/c++.pdf)
2. A reference for the C++ bitset class (http://www.cplusplus.com/reference/bitset/bitset/)
3. A reference to the C++ string class (http://www.cplusplus.com/reference/string/string/)