Lab 1 Due Date: Friday October 27, Midnight

Instructor: Siddharth Garg

Lab 1: Pipelined MIPS

In this Lab assignment, you will implement an cycle-accurate simulator for a pipelined MIPS processor in C++. The simulator supports a subset of the MIPS instruction set and should model the execution of each instruction with cycle accuracy. The MIPS processor you will model has a three stage pipeline.

The MIPS program is provided to 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* from HW1 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 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.

MIPS Architecture

You will model a pipelined MIPS processor with a 3-stage pipeline. The first stage (Stage1) of the pipeline performs instruction fetch (IF). The second stage (Stage2) performs instruction decode/RF read (ID/RF) and execute (EX). The third stage (Stage3) performs data memory load/ store (MEM) and writes back to the RF (WB). Branches are resolved in Stage2 of the processor. The processor has 1 branch delay slot: i.e., the instruction immediately after a beq is *always* executed.

What You Need to Do

Using the skeleton code provided, write a cycle-accurate simulator for the 3-stage pipelined MIPS processor. Your simulator must correctly update the architectural state of the processor in each clock cycle. The architectural state consists of the Program Counter (PC), the Register File (RF) and the Data Memory (DataMem).

The main( ) function in the skeleton code provided has a while loop. Each iteration of this loop corresponds to one clock cycle. At the end of each clock cycle, your code must output the current PC and any writes to the RF and the Dmem to the file "results.txt" in the following format:

<32-bit PC> <5-bit RF Wrt. Address> <32-bit RF Wrt. Data> <1-bit RF Wrt. Enabke> <32-bit Dmem Wrt. Address> <32-bit Dmem Wrt. Data> <1-bit Dmem Wrt. Enable>

Here, the RF Wrt. Valid and the Dmem Wrt. Enable bits are set to 0 if there is no write to the RF and/ or Dmem in that clock cycle, respectively, and to 1 if there is a write. Note that if a Wrt Enable bit is zero, then the corresponding Address/Data bits can be set to any values.

The simulation stops as soon as the "halt" instruction is fetched in the IF stage. At this point, the OutputRF() function and OutputDataMem() functions are called and dump the contents of the RF and DMem into two files. These are implemented for you. Do not modfy them.

What You Have to Submit

1. We have provided skeleton code in the file MIPS.cpp. Finish the code.
    - A Makefile has been provided for you to compile the source code.
    - You will be provided with an account on Gauss (gauss.poly.edu). While you can use any development environment to write and test your code, we will test it on Gauss.
    - On Gauss, you can compile your design by typing "**make"** which would create the MIPS executable. Run the executable using "**./MIPS"** command.
    - You are welcome to write your own Makefile to compile the code (for example, if you add new header files etc) but in this case you must include a "howto.txt" file explaining how to compile your submitted source code.
    - 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.

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/)