William Lin

May 8, 2017

CSCI 2214-02

Project 2 Report

DLX Hardware Simulator

1. The project was successful. My simulator runs both test files correctly and other test files that I create. I use a Simulator class object to run the simulation and the Main Memory class object simulates the main memory.

**MainMemory:**

* loadProgramCode(string) – Takes an input filename as a string parameter. Attempts to open and read instructions from the given input file into the instruction section of main memory.
* readMemory(int, int) – Takes two ints as parameters. The first int is the address to read memory at. The second int is either 0, 1, or 2 which will specify whether to read a word, halfword, or a byte.
* writeMemory(int, int, int) – Takes the address of where to write into memory, the data to write into memory and the size of data that is to be written. The size of the data can either be 0 for a word, 1 for a halfword, or a 2 for a byte.

**Simulator:**

* run(bool) – Function to run the simulator after main memory has been loaded instructions. Takes a bool which will determine whether run the simulator step by step or all at once. If bool is true then it will run step by step. Using control signal functions, runs each line of instruction based on the opcode and by turning on and off control signals.
* void loadCode(string) – Takes an input file name and passes it to MainMemory loadProgramCode to load the instructions of the input file into main memory.
* void printStateHex – prints the state of the simulator all in Hex. The state includes all control signals and hardware component values like MDR and RF.
* void printStateDec – prints the state of the simulator all in decimal
* void printStatePretty – prints the state of the simulator with a mix of both hex and decimal. Hardware such as IR which is better viewed in hex is printed in hex and hardware that are better in decimal like register values are printed in decimal.
* int getInstrType(int) – takes an opcode as the parameter and returns an int that corresponds with the instruction type that the opcode is.
* void step1() – uses the correct control signal functions to use the program counter and get the corresponding instruction from memory and load it into the instruction register.
* void step2() – Uses the correct control signal functions to add 4 to PC then save to PC
* void runRtype() – Uses the correct control signal functions to run Rtype instructions.
* void runJtype() – Uses the correct control signal functions to run J type instructions. J and JAL
* void runJItype() – Uses the correct control signal functions to run I type J instructions
* void runLoadInstr() – Uses control signal functions to load from memory into RF
* void runStoreInstr() – Uses control signal functions to store into memory.
* void runItypeALU() – Performs I type ALU operations using control signal functions
* void runBranchInstr() – Runs branch instructions using the control signal functions

**Simulator Control Signal Functions (All return type void):**

* setPCMARselect(bool) – set PCMARselect control signal equal to bool. If false put PC onto the addr bus and if true then put MAR onto the addr bus
* void readMem(bool, bool, int) – The first bool determine whether to read Mem or not. If true then we will readMem. Else we just set memRead = 0 and do nothing. The second bool will be used to determine if it is reading an instruction or data from mem. If true then it is reading from data and the value of the addr bus will have 3000 added to it so it reads from the data sector of memory. The int is memop. If the first bool is true then it checks the second bool and if that is true then 3000 is added to addr. Else do nothing to addr. We set control signal memRead = 1 and call mainMemory’s readMem() function and pass it the value in addr bus and the memop. We store the returned value into the dest bus and call setMemRead(1).
* setIRload(bool) – if bool is false the function does nothing. If true the function read what’s on the data bus into IR and use bit manipulation to determine rd, rs1, rs2, imm, offset, opcode and opcodeALU despite what type of instruction it is.
* setPCoeS1(bool) – if bool is true then put pc onto the s1 bus. Else do nothing
* setS2op(int) – The int parameter is the s2op. Uses the s2op to manipulate the bits on what is in s2 op. Then sets the control signal s2op to the int parameter of the function.
* setALUop(int) – The int parameter is the ALUop. Uses the int parameter to determine the ALUop and carries out the correct ALU operation. Then set the control signal ALUop to the int parameter.
* setPCload(bool) – if bool is true then put what’s on the dest bus into pc. Else do nothing.
* setAload(bool) – if bool is true then set a = to the value of registers at index rs1. Else do nothing
* setBload(bool) – if bool is true then set b = to the value of registers at index rs2. Else do nothing
* setCload(bool) – if bool is true then set c = to what is on the dest bus. Else do nothing
* setREGload(bool) – if bool is true then set the register at index value of DMX = to c. So the value ofDMX determines which register to load into. Else do nothing
* setREGselect(int) – The int determines the value of the DMX mux which will be used to load into registers. If 0, DMX = rd. if 1, DMX = rs2, and if 2, DMX = 31.
* setIRoeS1(bool) – if bool is true then s1 = ir so if true then put ir onto s1. Else do nothing
* setIRoeS2(bool) – if bool is true then s2 = ir so if true then put ir onto s2. Else do nothing
* setMARload(bool) – if bool is true then mar = dest. Else do nothing
* setMDRload(bool) – if bool is true then mdr = mdrMux. Else do nothing
* setMDRoeS2(bool) – if bool is true then put mdr onto s2 (s2 = mdr). Else do nothing
* setMemRead(bool) – if bool is true then mdrMux = data. Else do nothing
* setMemReadDest(bool) – if bool is true then mdrMux = dest. Else do nothing
* setMemOp(int) – sets MemOp control signal equal to int parameter
* setMemWrite(bool) – if true put mdr onto data bus (data = mdr). then call main memory’s writeMemory function and pass it the value in addr bus, data bus, and value of MemOp control signal. Else do nothing.

1. In main.cpp a Simulator object is created and named sim. Then we call loadCode() on sim and pass it the desired input file name. In loadCode, simulator just passes the file name to the loadProgramCode function of MainMemory. In loadProgramCode() the input file is opened and read in 2 characters at a time and converted to an unsigned long before being stored into a char which is a byte. The function loops through 3000 empty instruction spaces of main memory and attempts to populate them. However, if eof is reached then the function breaks the loop early meaning not all the instruction sector of main memory will be used. After that those function calls, the instructions that were in the input file will be loaded into the instruction sector of main memory. We then call run() on sim and pass a 1 to the run function to allow step by step simulation. In the run function there is a forever for loop that won’t terminate unless the value of ir is equal to 0 which means that we have reached the end of the instructions. The forever for loop runs step1() and step2() continuously and after each step() it checks ir to make sure that the end of instruction has not been reached. If it has then it will break out of the loop. Then it runs step2(). Next based on the instruction type it runs the correct instruction type function. Then it prints the state after running each instruction.
2. [See output files in project file] test1’s output is in output1.txt and test2’s output is in output2.txt