Assignment 2 Documentation

Hyeonwook Kim
118010488

MIPS Simulator

This programming project converts MIPS assembly language into binary code and simulates it. Based on the command, the Register and Memory of the MIPS simulator is updated every step. A checklist file is also used to document certain steps of the simulator, and when certain values are typed in, the states of the memory and register at that time is output.

Functions & Classes

adder( )
Takes 2 values as inputs and adds them in binary. Output is a 32 bit binary string. Multiple adders are used throughout the simulator.

shiftLeft( )
Takes values **a** and **n** as inputs. a is shifted to the left n times. For this code, all instructions are only shifted to the left by 2 bits.

andGate( )
Simulates an AND Gate. if both inputs are 1, the output is 1. Else, output is 0.

mux( )
Takes 2 values and a switch as inputs. The output is determined based on the value of the switch (1 or 0).

mainControl
The main control of the simulator. Takes OPCODE as the input and determines the following regDst, jump, branch, memRead, memToReg, aluOp, memWrite, aluSrc, regWrite.

vectorBin
Take a vector a, reverse it, return its n bit binary form in string.

binDecVector
Takes a 32 bit binary string and turns it into a vector of 4 elements. Used for writing to memory / register.

signDec
Converts Sign Binary to Decimal.

decSign
Converts Decimal a to Signed Binary with n bits.
regWrite
Takes Read Registers 1 and 2, Write Registers and reads its information. After the data has been extracted from the designated registers, it is sent to the ALU and ALU Control for the information to be processed. If control.regWrite is 1, the processed information is fed back into regWrite and the information is written to the designated register.

aluControl
Acts as the ALU Control. Takes the function code / opcode of the instruction and determines its instruction type.

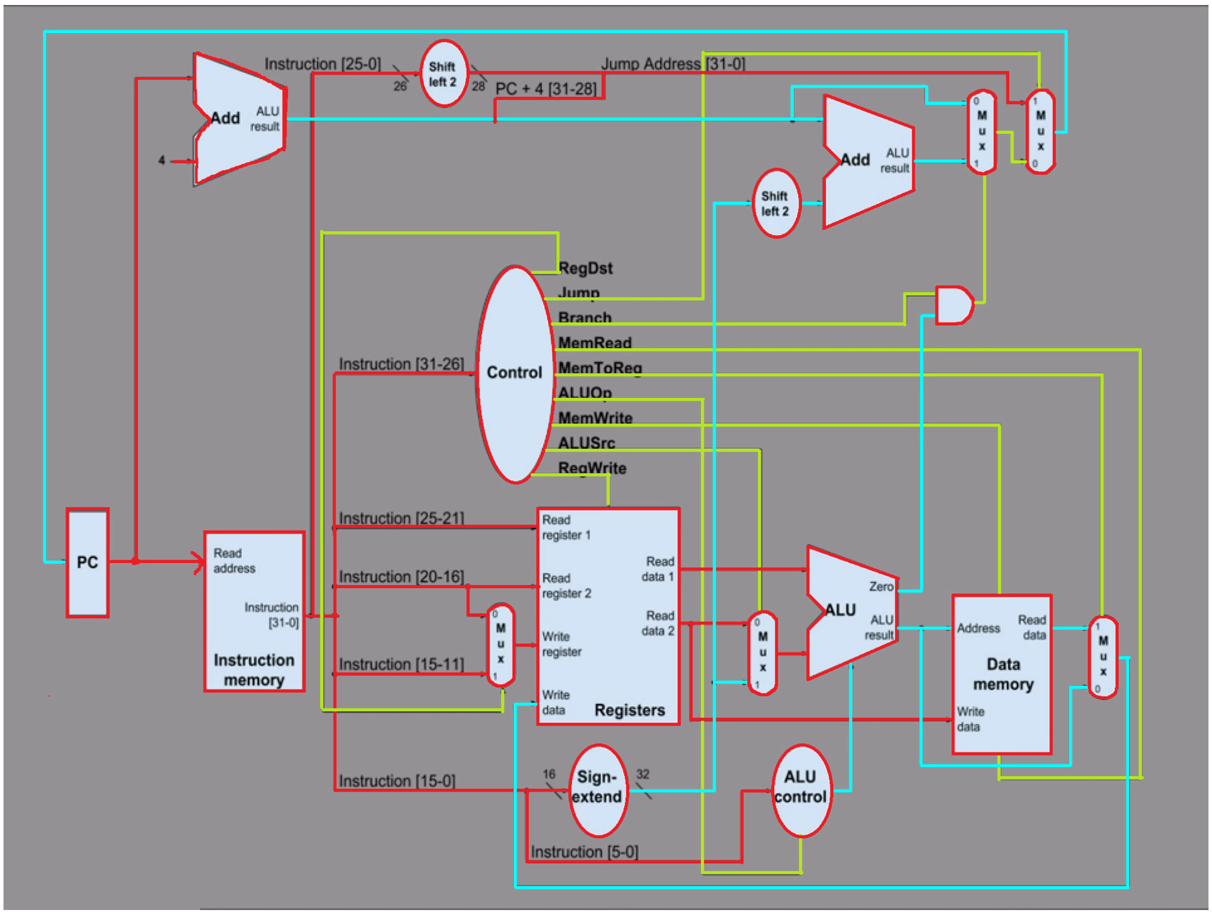
ALU
Takes the information of the 2 registers / sign extended immediate and performs an operation determined by the ALU Control. The output is released to the Data Memory and multiplexer. If the ALU Result is zero, a zero output turns to 1 and sends the signal to an AND gate with control.branch.

dataMem
Takes the data from the ALU as the address and the read data 2 as the writable data. The control.memWrite and control.memRead determines if the given information will be written or read. The output is fed into a multiplexer with the ALU data again, and control.memToReg determines which data will be fed back into regWrite.

PC
Keeps track of the PC. At the end of the single cycle the PC is written to the class. At the start, the PC is loaded onto the instruction memory.

instructionMemory
Takes the PC as input, and outputs the appropriate binary code determined by the PC and address. The binary code is also split into chunks so that other parts of the simulator can read its data (opcode, registers etc.)

Process



The register memory, data memory, pc, input code, binary code and checkpoints are first initialized. The code then extracts the input code from **test.asm** and puts it into **test.txt**. Afterwards, the binary code is inserted into the **.text** portion of the data memory, and the **.data** section is put into the static data. The inserted data can receive **.word, .half, .byte, .ascii** and **.asciiz** as possible data types.

A single cycle is executed for every line of code. The PC is fed into instruction memory, which loads the appropriate binary code based on the address. The main control then determines which parts of the simulator to activate depending on the instruction's opcode. Afterwards, the Registers are inserted into regWrite, and it feeds the information into ALU and data memory.

Meanwhile, the 16 bit immediate is extended to a 32 bit code, shifted left by 2 and fed into an adder. The function code and opcode are also put into ALU Control, which determines its instruction type. Then the register information and the ALU Control output is fed into the ALU, and the results are output to data memory, a multiplexer and an adder depending on the result of the operation.

Data Memory takes the data from the ALU as the address and the read data 2 as the writable data. The control.memWrite and control.memRead determines if the given information will be written or read. The output is fed into a multiplexer with the ALU data again, and control.memToReg determines which data will be fed back into regWrite.

The PC is then added by 4 by default. A separate value takes the PC and adds it with the extended immediate mentioned before for the branch instruction. A third value combines the first 6 digits of the PC and the 26 bit address from instruction memory. Depending on the outputs of regWrite, one of three PC addresses are chosen and fed back into the PC and the data is written. This concludes the single cycle, and the process continues until all binary code is read.