Computer Architecture

Project 2

Cycle Accurate Simulator

Jenna Uba

9 Mat 2018

Program Execution

**Overview**

The Cycle Accurate Simulator is a non-pipeline DLX hardware simulator which implements the assembler from project 1 and the data path for the DLX CPU. The simulator runs in a single step cycle function and contains two class, one for the state machine and another for the data path. The state machine class keeps track of which instruction step the simulator is running, sets the corresponding signals, and implements the following instructions: add, addi, and, beqz, bnez, j, lb, lh, lw, or, ori, sll, srl, sb, sh, sw, and sub. The data path class executes the state machine and uses the step number and signals to determine what items in the data path need to be updated and also what the final result of the instructions are. The final results of the simulator is a display of all the registers found in the CPU including the: register files, register buffers, hardware registers, memory registers, and what is on the data bus. After each cycle is complete the simulator waits until the user says it should continue to the next cycle.

**StateMachine**

**Steps (int instrType):** Function to keep track of which step the simulator is running.

**ReadTheInstruction():** Function that sets the signals for step one, when reading the instruction from memory.

**UpdatePC(int instrType):** Function that sets the signals for step two, when updating the PC.

**PerformOperationRType(int instrType):** Function that sets the signals for when an r-type ALU instruction must be executed.

**PerformOperationIType(int instrType):** Function that set the signals for when an i-type ALU instruction must be executed.

**CheckRS1(int reg, int instrType):** Function that sets the signals to check if rs1 is equal or not equal to zero during a branch instruction.

**UpdatePCOffset(int instrType):** Function that sets the signals to update the PC with the offset during a jump instruction.

**FindMAR(int instrType):** Function that sets the signals to find the MAR when executing a load or store instruction.

**UpdatePCIR(int instrType):** Function that sets the signals to update the PC with IR when executing a branch instruction.

**FindMDRStore(int instrType):** Function that sets the signals to find the MDR when executing a store instruction.

**FindMDRLoad(int instrType):** Function that sets the signals to find the MDR when executing a load instruction.

**WriteToMem(int instrType):**  Function that sets the signals to write to memory when executing a store instruction.

**FindC(int instrType):** Function that sets the signals to find value or the c register when necessary.

**SaveToReg(int instrType):** Function that sets the signals to save to the registers in the register file.

**getALUop(int instrType):** Function that determines the ALU opcode signal of the instruction being executed.

**getMemOp(int instrType):** Function that determines the memory opcode signal of the instruction being executed.

**getREGselect(int instrType):** Function that sets the Register Select signal to determines which register the data must be saved to.

**Data Path**

**LoadProgramCode(char\* fileName, unsigned char MainMemory[]):** Function that loads in the hexadecimal input file and places all the data into a character array.

**start(unsigned char MainMemory[]):** Function that determines which step the state machine is to allow the data path to know which signals it is analyzing.

**execute(unsigned char MainMemory[], int instrType):** Function calls the setAllData() function and the printResults() function.

**setAllData(unsigned char MainMemory[], int instrType):** Function that calls all the functions involving the CPU registers and busses to analyze the signals and properly execute the step.

**setS1Bus():** Function that determines if the Aoe or PCoeS1 signals are on in order to set the S1 bus.

**setS2Bus():** Function that determines if the Boe, IRoeS2, or MDRoeS2 signal is on in order to set the S2 bus.

**setDest():** Function that sends the data on the S1 bus and the data on the S2 bus as well as the ALU opcode signal through the ALU function to set the destination bus.

**setAddr():** Function that determines what the PCMARselect is equal to in order to set the addr bus.

**setData(unsigned char MainMemory[]):** Function that determines whether the data bus is carrying the IR or the MDR.

**setRegisterFile(int instrType):** Function which sign extends the IR to properly determine and set the register data is being loaded to and read from.

**setMemoryRegister(unsigned char MainMemory[], int instrType):** Function that determines what the memory opcode signal is to properly determine the MAR, MDR, and where in memory store functions are saving to.

**setHardwareRegister(unsigned char MainMemory[]):** Function that determines what the IR and PC are at all the different steps (ex. Jump/branch instructions updating PC).

**setRegisterBuffers(int instrType):** Function that sign extends the IR to properly set the A register, B register, and C register.

**getOpcode(unsigned char MainMemory[]):** Function that determines what instruction type is being read from memory.

**instrRType(unsigned char MainMemory[]):** Function which determines if the instruction being read is an r-type and sets the opcode equal to its function code.

**ALU(int S1, int S2, int opcode):** Function that correctly executes the ALU operation being performed.

**printResults():** Function that prints out all of the CPU register data as well as the bus data.

**checkS2op(int num):** Function that checks the S2 opcode signal value to properly sign extend the data on the S2 bus.

**Experience**

The goal of this program was to correctly use the signals set in the state machine to determine which CPU registers and data must be set in the data path. The concepts and theory of this program were fairly simple and not extremely difficult to implement. Most of the project consisted of debugging and logic pushing to ensure that all the registers and busses were being properly set during each step of all the required instructions.