MIPS Simulator

Alexander Gonzalez
[redacted]

Introduction:

This is an instruction-by-instruction simulator for a MIPS Processor. It is faithful to the 5-stage MIPS Processor:

1. IF: Instruction fetch from memory
2. ID: Instruction decode & register read
3. EX: Execute operation or calculate address
4. MEM: Access memory operand
5. WB: Write result back to register

And includes the main structures of a MIPS Processor:

1. PC
2. Instruction Memory
3. Register File
4. Data Memory

With this simulator, you may input any assembly file and the simulator will provide the final state of the register file and values in data memory. It also gives you a debug option in which it will display the register file state after each instruction, the PC, and the Control Signals for each instruction.

Build/Run Instructions:

To build this simulator, you will need access to a Linux command line. Use the cd command to get into the MIPS_SIMULATOR folder. Use the line below to compile the simulator.

```
g++ main.cpp instruction_memory.cpp register_file.cpp alu.cpp
data_memory.cpp control_unit.cpp -o simulator
```

And use this line to run the simulator:

```
./simulator
```

Be sure to include the assembly file in the MIPS_SIMULATOR folder or the program will not find the file to read.

Limitations:

Pipelining is not a feature of this simulation, meaning that multiple instructions are not overlapping during execution. This means each instruction goes through each stage of the processor one at a time.

The simulator also only supports a select few instructions. These instructions include:

- `j`
- `beq`
- `add`
- `addi`
- `sub`
- `sw`
- `lw`
- `sll`
- `srl`
- `mul`
- `and`
- `or`
- `nop`

Because there is no `la` or `li` instructions, this simulator only shows the registers being used to complete the instruction, not the values stored in the registers.

Instruction `j` will only jump once to avoid infinite loops.

Instruction `beq` will assume the first comparison is true and branch. The next time it encounters the same branch, it will not branch to avoid infinite loop.

The simulator only allows for 1000 instructions to be loaded.

Example of How to Compile and Run:

1. Use the command cd to enter the MIPS_SIMULATOR folder

```
Alexs-MBP:~ alexgonzalez$ cd /Users/alexgonzalez/Documents/COMP_ARCH/MIPS_Simulator/MIPS_Simulator
Alexs-MBP:MIPS_Simulator alexgonzalez$
```

2. Compile and Run the simulator

```
Alexs-MBP:MIPS_Simulator alexgonzalez$  g++ main.cpp instruction_memory.cpp register_file.cpp alu.cpp data_memory.cpp control_unit.cpp -o simulator
Alexs-MBP:MIPS_Simulator alexgonzalez$ ./simulator
Please enter assembly file name includeing extension:
```

3. Enter file name you would like the Simulator to read. This will give you the final state of the register file and what is stored in memory. If you want to use the debug feature, type “Y” when prompted to get an in-depth look at your assembly code. If the file is not found, you will be given an error message saying that the assembly code is missing, so make sure the file you want read is included in the MIPS_SIMULATOR folder. (One test will be provided – assembly.asm).

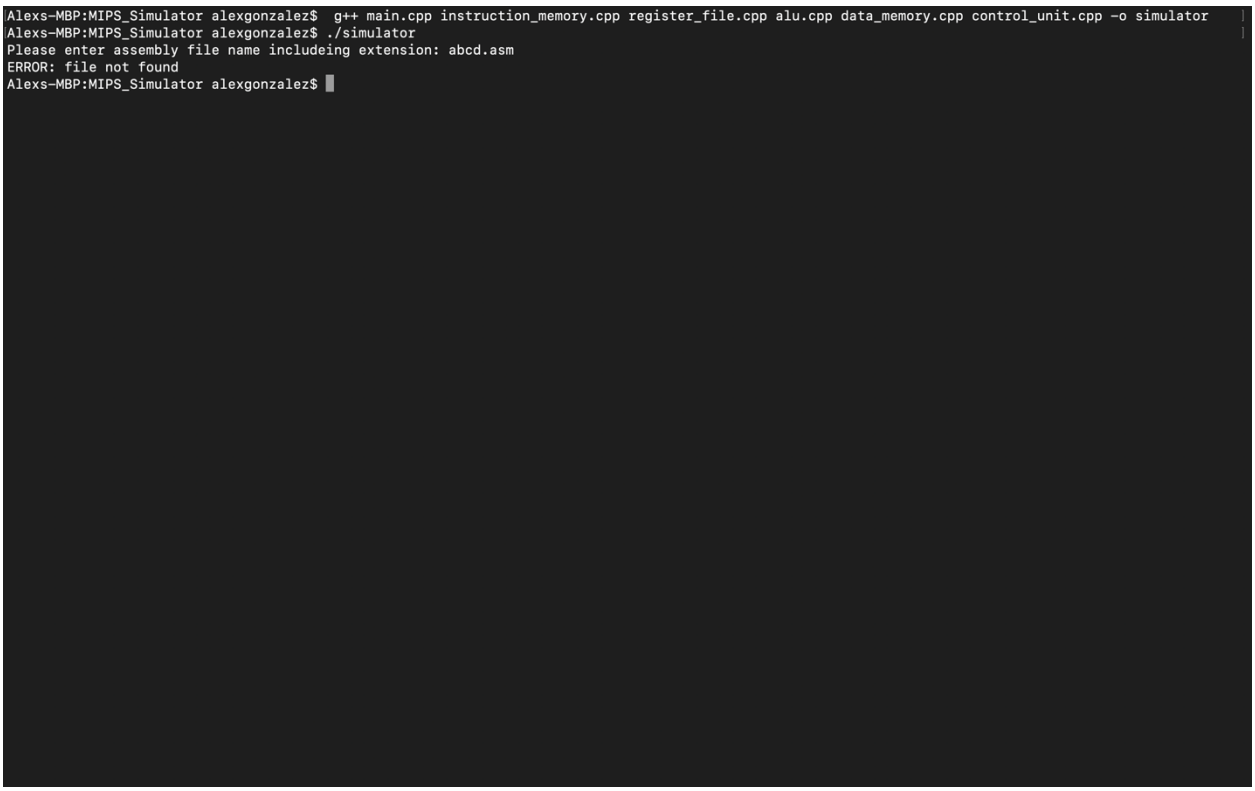
```
Alexs-MBP:MIPS_Simulator alexgonzalez$  g++ main.cpp instruction_memory.cpp register_file.cpp alu.cpp data_memory.cpp control_unit.cpp -o simulator
Alexs-MBP:MIPS_Simulator alexgonzalez$ ./simulator
Please enter assembly file name includeing extension: assembly.asm
-------------------------
|Register File:         |
-------------------------
|Read Reg 1: X
|Read Reg 2: X
|Write Reg: X
|Read Data 1: X
|Read Data 2: X
|Write Data: X
-------------------------
|Memory:                |
-------------------------
| 4       $3
-------------------------

Would you like to access the debug mode? (Y/N):
```

```
Would you like to access the debug mode? (Y/N): Y
******INSTRUCTION add ******
-------------------------
|Register File:         |
-------------------------
|Read Reg 1: $2
|Read Reg 2: $3
|Write Reg: $1
|Read Data 1: $2
|Read Data 2: $3
|Write Data: $2 + $3
-------------------------
|State Regesters:       |
-------------------------
|PC: 0
-------------------------
|Control Signals:       |
-------------------------
|RegDst: 1
|RegWrite: 1
|ALUSrc: 0
|PCSrc: 0
|MemWrite: 0
|MemRead: 0
|MemtoReg: 1
-------------------------

******INSTRUCTION addi ******
-------------------------
|Register File:         |
-------------------------
|Read Reg 1: $5
|Read Reg 2: X
|Write Reg: $4
|Read Data 1: $5
|Read Data 2: X
|Write Data: $5 + 100
-------------------------
|State Regesters:       |
-------------------------
|PC: 4
-------------------------
|Control Signals:       |
-------------------------
```

4. If the file is not found, the simulator will give you an error message, so you will have to run the simulator again using `./simulator`



Final Notes:

- "X" in the register file means that the instruction did not utilize that part of the register file.
- "X" in the Control Signal means that it did not matter if the signal was on or off.
- The simulator does not account for data or control hazards.
- Assembly instructions are not stored in binary format.