Rebecca Lee
Danielle Yap

[Github Repository]

TASK 1
Repository used

- To run the file, use the commands
  - `iverilog *.v`
  - `./a.out; gtkwave dump.vcd test.gtkw`

FILE BREAKDOWN

- **`alu.v`**
  - Takes in two 32-bit integer operants (`a_in` and `b_in`) and a 3 bit opcode/operation (`f_in`) and outputs `zero` (whether the output is zero (`zero=1`) or not (`zero=0`)), any carryout from the operation (`c_out`), and result of the operation (`y_out`)
- **`aludec.v`**
  - Takes in 32-bit instruction (`instr`) and outputs 3-bits for `alucontrol`
    - Used in conjunction with `maindec.v` in `controller.v`; determines what to do with the 25th to 6th bits in machine code instruction (`instr`)
Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format
  - Sets `alucontrol` bits based on the value of middle 20 bits of `instr`
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR
- **`controller.v`**
  - Inputs 32-bit instruction (instr) and outputs `branch`, `jump`, `mem_to_reg`, `mem_write`, `reg_dst`, `reg_write`, 3-bit `alucontrol` and `alu_src`
Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format
  - Calls:
    - `maindec.v`
      - Determines first and last 6 bits

        - Input `instr`
        - Output `branch`, `jump`, `mem_to_reg`, `mem_write`, `reg_dst`, `reg_write`, and `alu_src`
    - `aludec.v`
        - Determines middle 20 bits
        - Input `instr`
        - Output `alucontrol`
- **`datapath.v`**
    - Takes in clock (`clk`), reset (`rst`), 3 bits for `alucontrol`, `alu_src`, `branch`, `jump`, `mem_to_reg`, `mem_write`, `reg_dst`, `reg_write`, 32 bits for the machine language instruction (`instr`), and 32 bits for data read from memory (`read_data`)
    - Outputs 32 bits for program counter (`pc`), 32 bits for operation calculations (`alu_result`), and 32 bits of data written in (`write_data`)
    - Makes all calculations for pc-related registers (jump, src, etc.)
    - Determines rs, rt fields from instruction
    -

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

    - Feeds inputs into `regfile.v`, `alu.v`, and `sign_extend.v`
- **`dmem.v`**
    - Inputs clock (`clk`), write enable (`we`), 32-bit address (`addr`), 32 bit data to be written (`wdata`), and outputs 32-bit data read out (`rdata`)
    - Data memory
    - Iterates through memory and reads/writes data (depending on input) stored at the specified memory address
- **`imem.v`**
    - Inputs 6-bit `address`, outputs 32 bits of `data`
    - Instruction memory; returns `data` value (containing instruction) depending on `address` entered
- **`maindec.v`**
    - Translates machine code to MIPS instruction
        - Inputs 32-bit instruction (instr)
        - Outputs `branch`, `jump`, `mem_to_reg`, `mem_write`, `reg_dst`, `reg_write`, and `alu_src`
        -

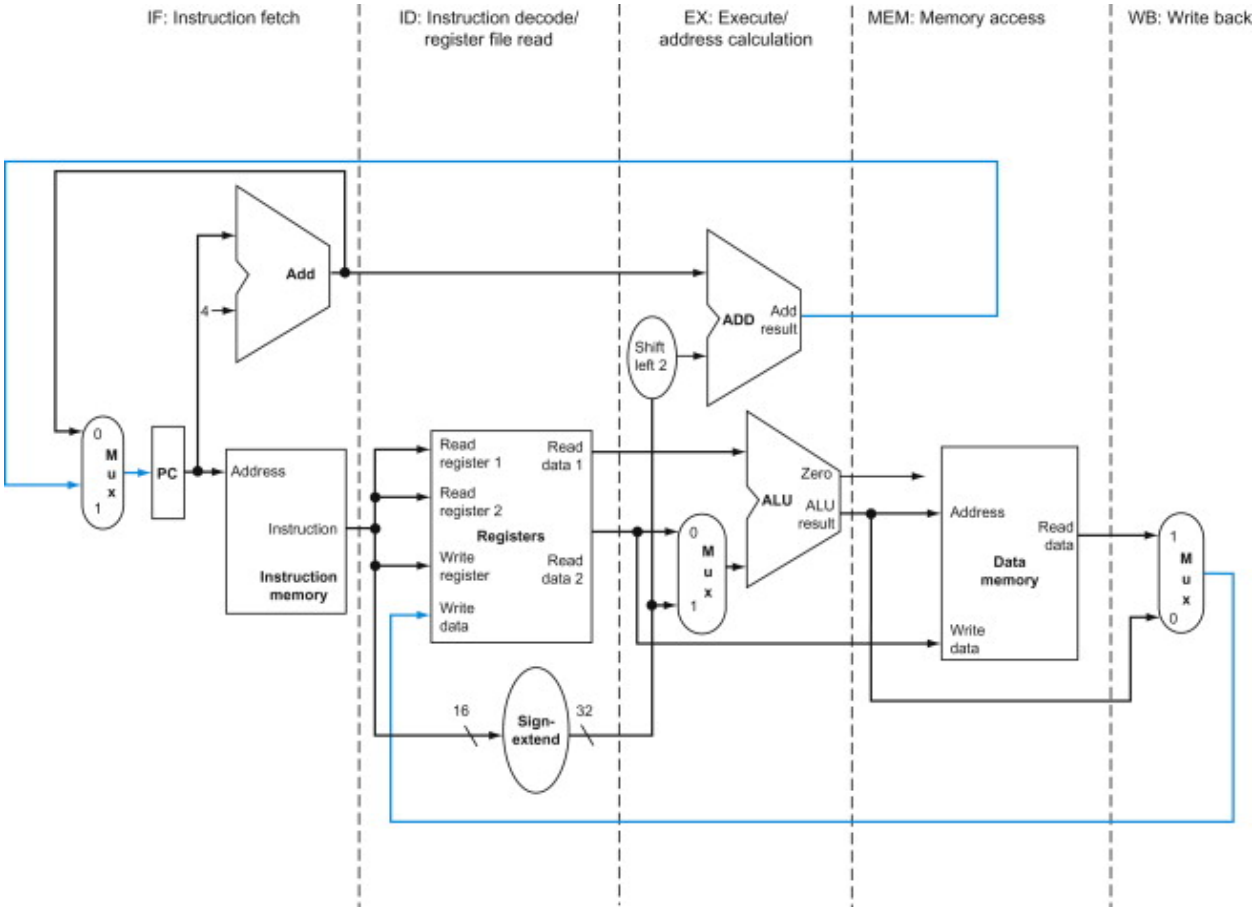
Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

            - From ZyBooks (fig 2.10.3)
    - First six bits of `instr` are denoted as the opcode (`opcode`)

    - Last six bits of `instr` are denoted as the function (`func`)
    - Determines MIPS instruction by comparing `opcode` (and `func` for ALU operations) to their corresponding values in hex, then assigns instructions to output values
- **`mips.v`**
    - Takes in clock (`clk`) and reset (`rst`), 32 bits for machine language instruction (`imem_data`), 32 bits of data read by data memory(`dmem_rdata`)
    - Outputs 32 bits for `imem_addr`, 32 bits of result of ALU computation(`dmem_addr`), 32 bits of data to be written (`dmem_wdata`), and the write enable bit (`dmem_we`)
    - Feeds inputs into `controller.v` and `datapath.v`
- **`regfile.v`**
    - Inputs clock (`clk`), three 5-bit addresses (`addr1`, `addr2`, `addr3`), 32-bit data to be write (`wdata`), and read/write register (`rw`)
    - Outputs two 32-bits data values (`data1`, `data2`)
    - Reads/writes (depending on input) data from register memory
- **`sign_extend.v`**
    - Takes in 16 bits (`idata`) and outputs 32 bits (`odata`)
    - Sign extends input number; 16 to 32 bits
- **`tb.v`**
    - Test bench for the program; sets clock and creates dump file
    - No input/ouput
- **`top.v`**
    - Takes in clock (`clk`) and reset (`reset`); no output
    - Feeds wire values and inputs to `imem.v`, `mips.v`, and `dmem.v`

TASK 2

The code already has the 5 stage pipeline implemented; each component in the system is its own .v file



1. IF - instruction fetch
    a. Operations relating to the PC occur in `datapath.v`
    b. Instructions are determined by their opcodes and func values in `maindec.v`
    c. Further details of instructions (i.e. middle 20 bits) are determined by `aludec.v`
    d. The instruction memory is located in `imem.v`
2. ID - instruction decode/register file read
    a. Registers are located in `dememv`
    b. Sign-extend is done by `sign_extend.v`
3. EX - execute/address calculation
    a. All operations are calculated in `alu.v`
4. MEM - memory access
    a. `regfile.v` stores/reads data values from register memory used in the processor's operation; when writing, waits for the clock to be on the positive edge to begin
5. WB - write back
    a. `dmem.v` stores/reads data values from *data memory*; when writing, waits for the clock to be on the positive edge to begin