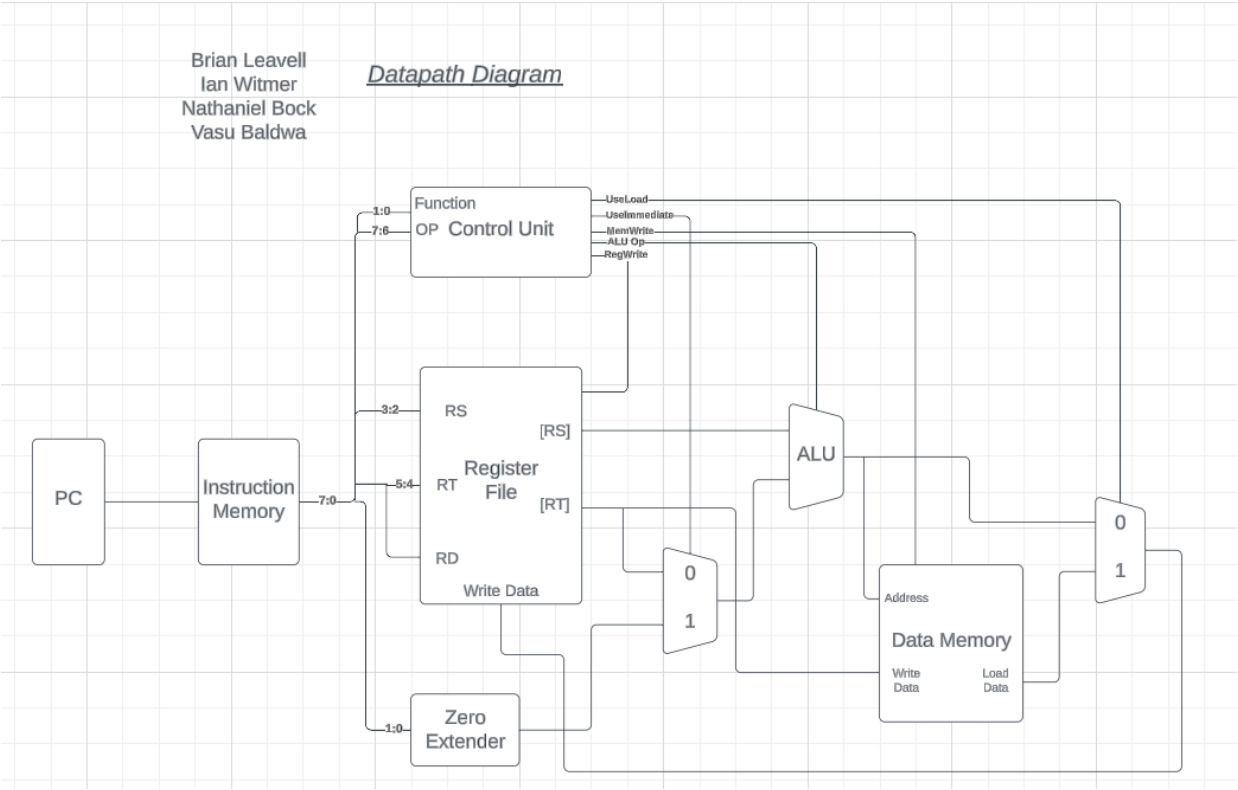
CSE 490 Project 1 Final Report

Group # 42

Team Members:

Ian Witmer
Nathaniel Bock
Vasu Baldwa
Brian Leavell



Work Done and Division of work:

- ALU design and testbench -Nate
- Decoder design -Brian
- Register File design and testbench -Ian
- PC counter design -Vasu
- Zero extender design and testbench -Vasu/Nate
- Multiplexor design -Nate
- Data memory design and testbench -Nate
- Instruction memory design -Nate
- Assembler (for funsies) -Vasu
- Datapath Design - Ian
- Datapath Implementation - Nate
- Decoder Testbench - Brian
- Control Unit design - Nate
- Datapath diagram - Brian
- Hardware testbench - Ian + Nate
- Xilinx bitstream generation - Nate
- Module Descriptions - Ian
- Simulation Waveforms - Nate

Module Descriptions:

ALU:

Performs addition, subtraction, logical left shift, or logical AND on two 8-bit inputs. The operation performed depends on the ALU Operation code provided as an input from the Control Unit.
Inputs:

- a: first value to be operated on
- b: second value to be operated on
- ALUop: determines which operation to do on a and b

Outputs:

- res: the result of the operation

Decoder:

Splits the instruction passed in into its four constituent parts; bits 1:0 to be used as function code or immediate based on the instruction type, bits 3:2 to be used as the source register, bits 5:4 to be used as the target and destination registers, and bits 7:6 to be used as the operation code.
Inputs:

- instr: an 8-bit instruction

Outputs:

- fun_imm: used as either the immediate in an I-type instruction or the function code in an r-type instruction
- rs: code to be used for the source register
- rtd: code to be used for the target and destination registers
- op: operation code to be used by the Control Module

Register File:

Stores the four general use registers, outputting two of their current values that are selected by the inputs “r_read1” and “r_read2” as outputs “r_read1_data” and

“r_read2_data” respectively. Updates a register designated by the “r_write input” with the value of the “r_write_data” input when the write input is true and a rising edge is detected on the “clk” input. Initial values for each register are loaded from the “regfile_data.dat” file.
Inputs:

- r_read1: code for the register to be used as the source register
- r_read2: code for the register to be used as the target/destination register
- r_write: code for the register to write “r_write_data” to if a write occurs (same as rt in this implementation)
- r_write_data: the data to write to the register file if “write” is true
- write: control signal to determine whether a write occurs when a rising edge is detected on “clk”
- clk: updated the register file on a rising edge if “write” is true

Outputs:

- r_read1_data: the data from the source register
- r_read2_data: the data from the target register
- reg0data: the data from the first register
- reg1data: the data from the second register
- reg2data: the data from the third register
- reg3data: the data from the fourth register

PC Counter:

Outputs an 8-bit value representing an instruction address. Starts at 0 and increases by 1 on every rising edge detected on its “clk” input.
Inputs:

- clk: causes the output to increase by 1 on a rising edge

Outputs:

- reg: the current pc value

Zero Extender:

Takes a 2-bit input and generates an 8-bit output composed of the input with 6 zeros prepended to it.
Inputs:

- in: the two-bit constant

Outputs:

- out: the “in” input with 6 zeros prepended to it

Multiplexor:

Takes two 8-bit inputs as well as a 1-bit selector input. Depending on the selector input, it chooses one of the other two inputs to become the output.
Inputs:

- a: first input to choose from
- b: second input to choose from
- sel: when 0, chooses “a” to be “res”, when 1 chooses “b” to be “res”

Outputs:

- res: the chosen input

Data Memory:

Stores a collection of 8-bit data words with unique addresses. Takes an 8-bit “addr” input and outputs the value at that address at the data output. If the “write_enable” input is true, the value at the “write_data” input will be stored at the address specified by the “addr” input when a rising edge is detected on the “clk” input. Initialized with values from the “data_mem.dat” file.
Inputs:

- clk: if “write_enable” is true, writes the value of “write_data” to the address in “addr”
- addr: the address to read/write from/to
- write_data: the data to write if a write occurs
- write_enable: determines whether a write occurs

Outputs:

- data: the data stored at the address in “addr”

Instruction Memory:

Stores a collection of 8-bit data words with unique addresses. Based on the “addr” input, it outputs the corresponding value in memory to the “data” output. Values cannot be changed, and are initialized by reading in values from “instr_mem.dat”.
Inputs:

- addr: the address that the “data” comes from

Outputs:

- data: the instruction loaded from “addr”

Control Unit:

Takes the Op Code and Function fields of the current instructions and generates control signals to properly perform the required operations.
Inputs:

- op: upper two bits of the instruction used to determine control bits
- funct: used to determine ALU Op with the “op” input

Outputs:

- regWrite: determines whether the register file is written to
- memWrite: determines whether the data memory is written to
- useImmediate: determines whether the value from the zero extender or the value from rt is used as the second input to the ALU
- useLoad: determines whether the value to be written to the register file comes from the data memory or the ALU
- aluop: determines the function of the ALU

Datapath:

The collection of everything put together:
This waveform shows a set of 10 instructions running on our processor (only 9 have their output shown). The registers only get updated from the alu or data memory on the rising edge after the instruction was executed. The registers are loaded with initial values of 0x55, 0xDE, 0xAD, and 0x69 respectively.
Inputs:

- clk: updates the PC and Data Memory when a rising edge is detected, moving the next instruction through. Updates the Register file when a falling edge is detected.

Outputs:

- pc: the current register address of the instruction running
- instruction: the 8-bit instruction running
- r0: the value of the first register in the register file
- r1: the value of the second register in the register file
- r2: the value of the third register in the register file
- r3: the value of the fourth register in the register file

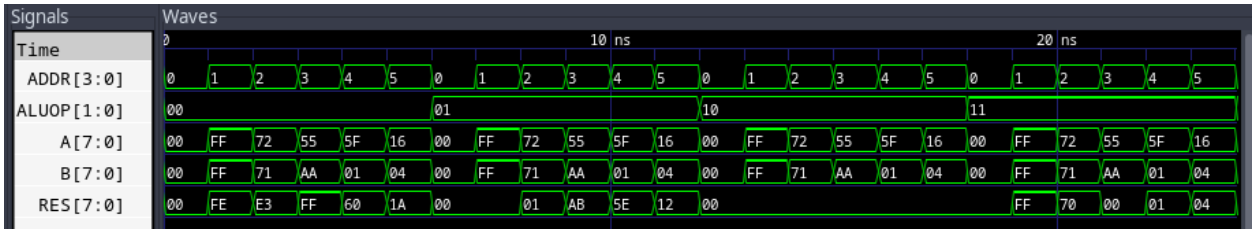
The assembly (form of `instr rt/rd, rs, #im`):

```
sub s0, s0
addi s1, s0, 3
sll s3, s1
and s2, s3
sw s3, s2, 3
addi s2, s2, 2
lw s0, s2, 1
add s0, s1
```

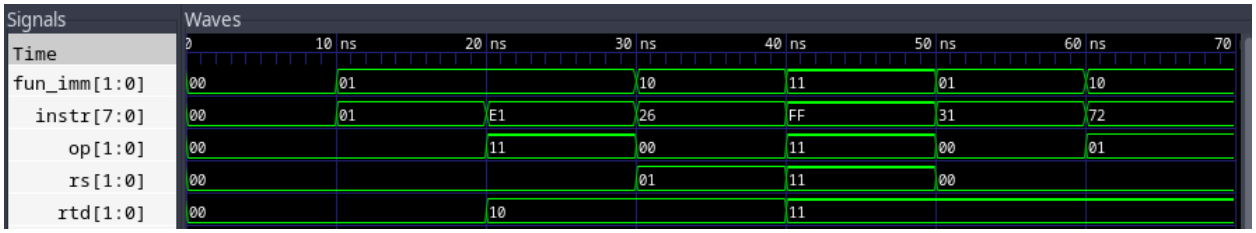
Module Simulations:

Note The simulations here are different from those used in the previous submission; the initial value for reg3 is changed and the register file updates on the falling edge of the current instruction, instead of the rising edge of the next instruction. This is to create proper functionality for the hardware demo.

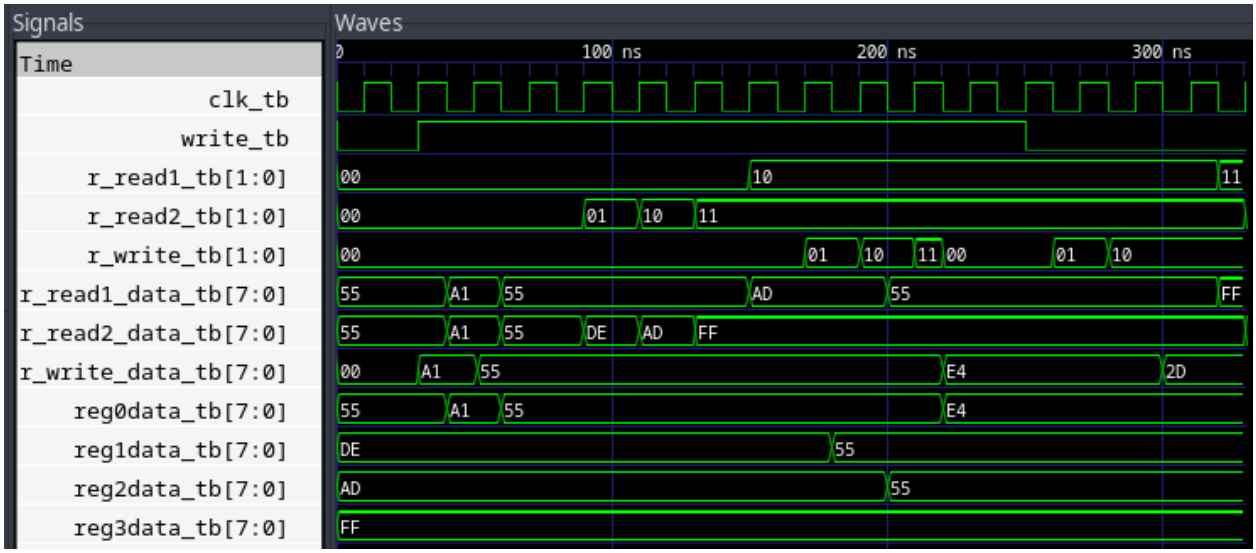
ALU



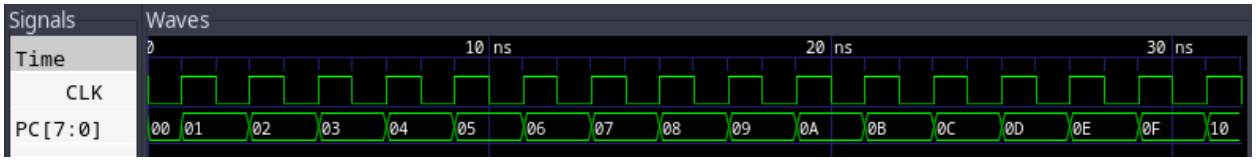
Decoder



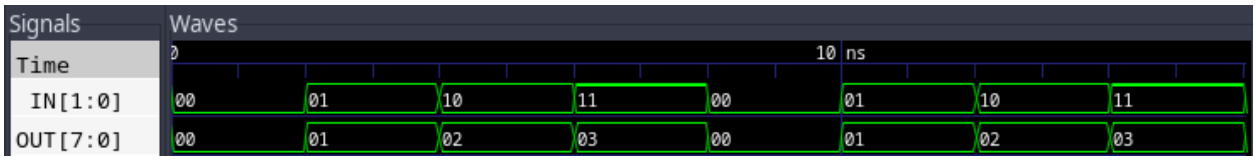
Register File



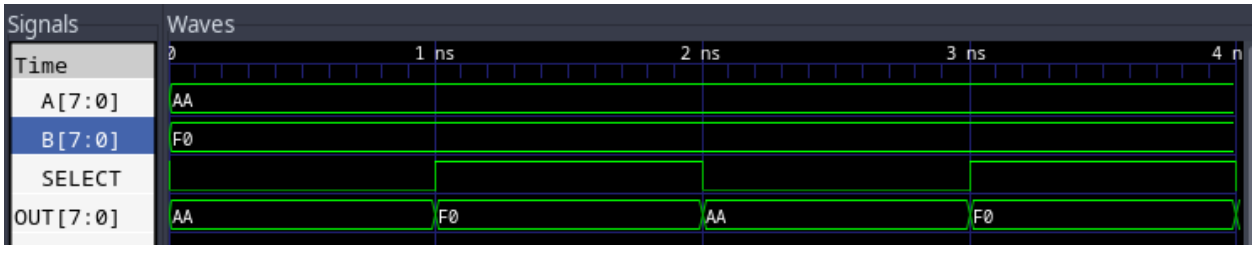
PC Counter



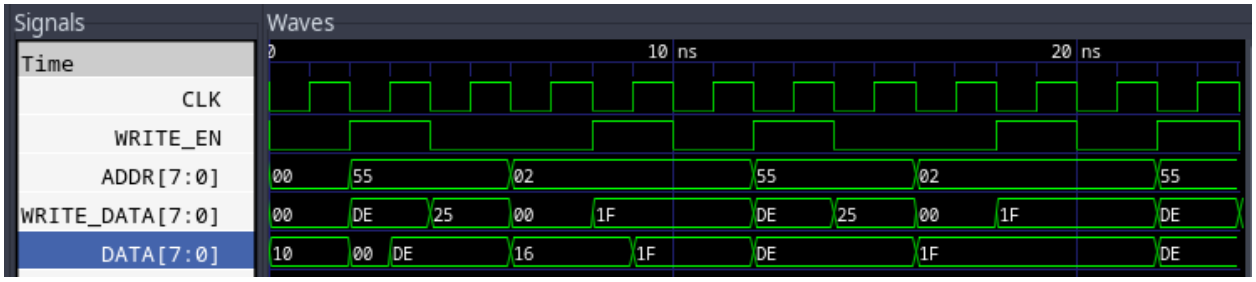
Zero Extender



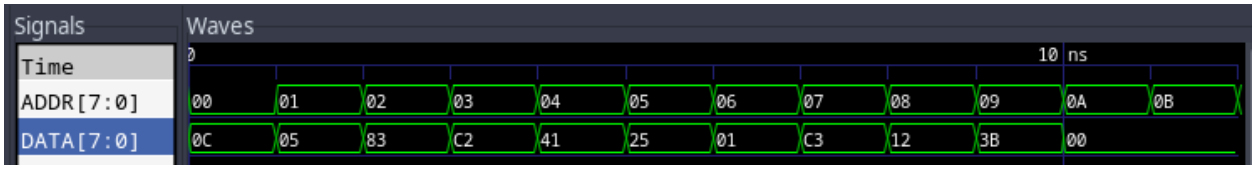
Multiplexor



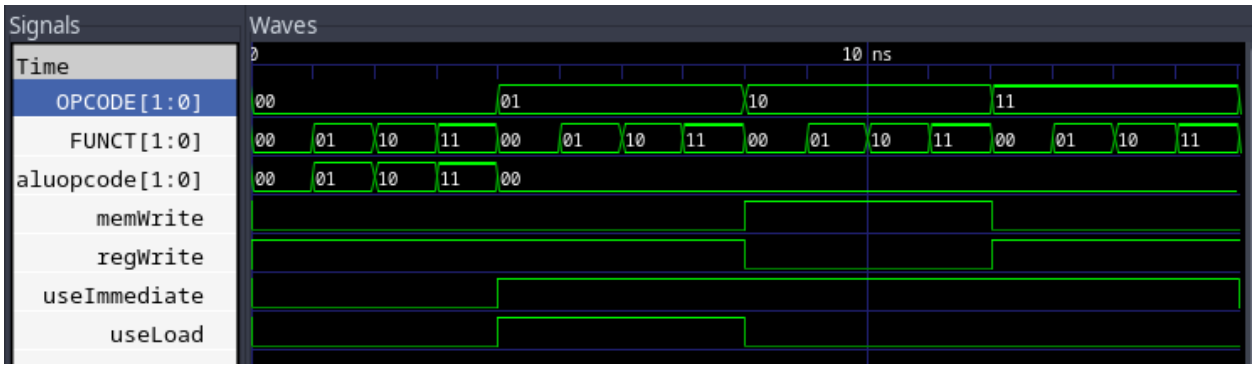
Data Memory



Instruction Memory



Control Unit



Datapath