Simple Processor in System Verilog

Carson Holland

April 25th, 2023

Assistance from Professor Mark Welker

Modules:

- ➢ Top.sv
- ➢ testBench.sv
- ➢ MainMem.sv
- ➢ Instruct.sv
- ➢ ExecutionEngine.sv
- ➢ IntegerALU.sv
- ➢ Matrix_ALU.sv

Address Architecture

The address is a 16 bit long input to every module responsible for finding data. The upper nibble of the address indicates which module to talk to. The lower 12 bits act as the offset in which to locate the elements. For example, the address 1004'h refers to offset 4 in the instruction memory module. This allows for retrieving data values stored within memory, reading statuses and results from ALU units, and retrieving data within internal registers.

16 - Bit Address Architecture															
Module Number				Location											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Memory Location	Module
0h	Main Memory
1h	Instruciton Memory
2h	Matrix ALU
3h	Integer ALU
4h	Internal Registers
5h	Execution Engine

Instruct.sv

Inputs:
input logic nRead, nReset, Clk;
input logic [15:0] address;

Outputs:
output logic [31:0] Dataout;

Purpose: This is a read only memory for storing all instructions. Upon reset, the 10 project instructions are loaded into the ROM. The upper byte of the instruction is for storing the opcode which tells the processor what action is to be taken. The upper nibble of this byte is either a 0 or 1, indicating if the operation is related to integers or matrices. The other 3 bytes are for destination, source 1 and source 2 locations. The upper nibble of these fields are 0 for main memory, and 1 for internal registers. The other nibble is for offset to address these modules.

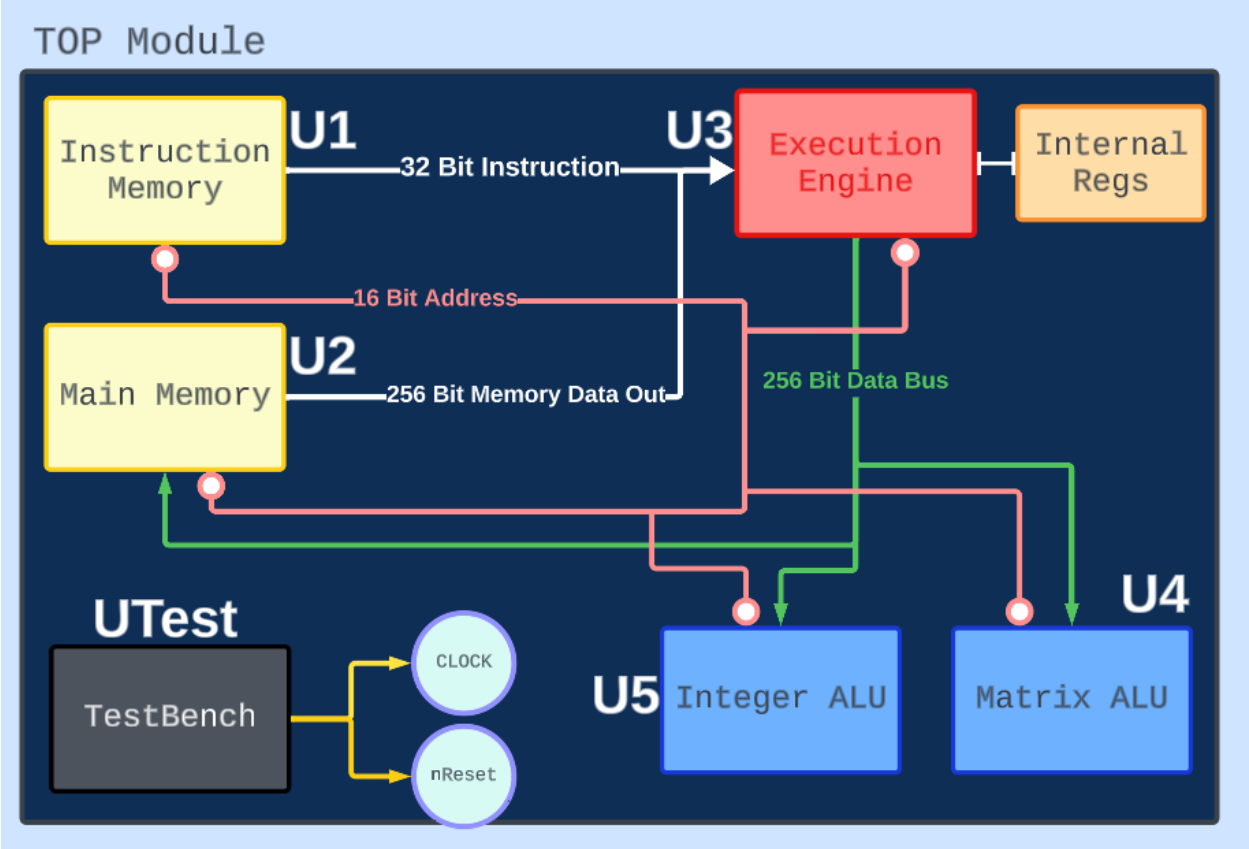
32 Bit Instruction Architecture																															
Op Code								Destination								Source 1								Source 2							
0 = Matrix 1 = Integer				Operation				0 = Main Mem 1 = Reg				0 = Main Mem 1 = Reg				0 = Main Mem 1 = Reg				Location				0 = Main Mem 1 = Reg				Location			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Instruction	Op Code	Destination	Source 1	Source 2
STOP	FFh	0	0	0
MMult1	00h	0n / 1n	0n / 1n	0n / 1n
MMult2	01h	0n / 1n	0n / 1n	0n / 1n
MMult3	02h	0n / 1n	0n / 1n	0n / 1n
Madd	03h	0n / 1n	0n / 1n	0n / 1n
Msub	04h	0n / 1n	0n / 1n	0n / 1n
Mtranspose	05h	0n / 1n	0n / 1n	0n / 1n
MScale	06h	0n / 1n	0n / 1n	0n / 1n
MScaleImm	07h	0n / 1n	0n / 1n	0n / 1n
IntAdd	10h	0n / 1n	0n / 1n	0n / 1n
IntSub	11h	0n / 1n	0n / 1n	0n / 1n
IntMult	12h	0n / 1n	0n / 1n	0n / 1n
IntDiv	13h	0n / 1n	0n / 1n	0n / 1n

Top.sv

Logic variables:
logic [31:0] InstructDataOut;
logic [255:0] MemDataOut;
logic [255:0] ExeDataOut;
logic [255:0] IntDataOut;
logic [255:0] MatrixDataOut;
logic nRead,nWrite,nReset,Clk;
logic [15:0] address;
logic Fail;

Purpose: Connects all the modules in the Simple Processor. Connects the processor modules and the testbench.



testBench.sv

Outputs:
output logic Clk,nReset;

Purpose: Simply drives the clock and reset control signals. It initially resets the system to load the instructions and set all data to 0, and then drives the clock for the modules to perform.

MainMem.sv
Inputs:
input logic [255:0] DataIn;
input logic nRead,nWrite, nReset, Clk;
input logic [15:0] address;

Outputs:
output logic [255:0] Dataout;

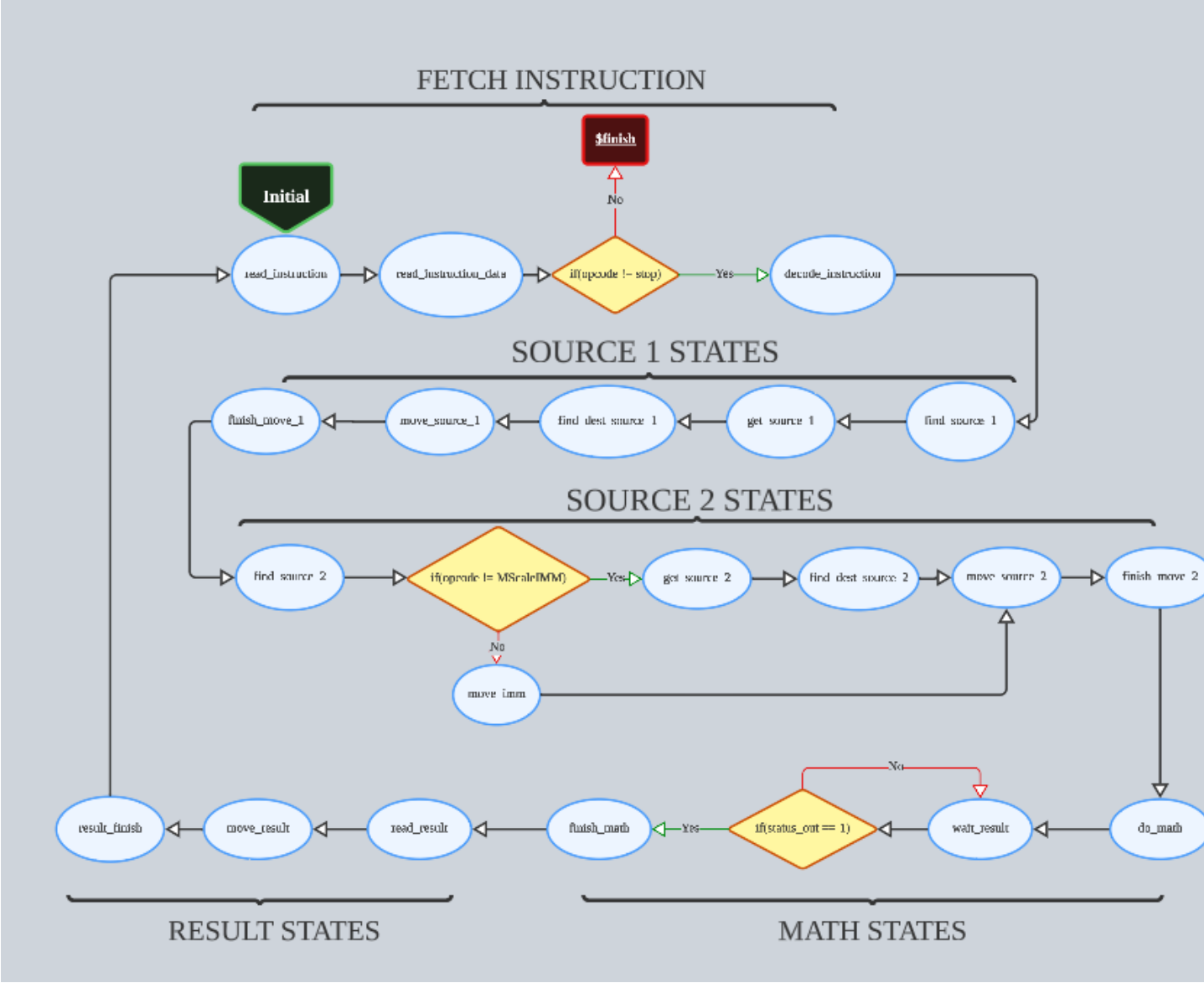
Purpose: This is the main memory for the system. It holds 256 bit data in each memory address. It can has initial data loaded into the memory upon reset to validate the processor's performance.

ExecutionEngine.sv

Inputs:
input logic Clk, nReset;
input logic [31:0] InstructDataOut;
input logic [255:0] MemDataOut, IntDataOut, MatrixDataOut;

Outputs:
output logic [255:0] ExeDataOut;
output logic [15:0] address;
output logic nRead, nWrite;

Purpose: The execution engine acts as the data flow control unit for the processor. It is the module that sets the address to the correct value for a given part of an instruction. It contains a PC, a program counter that initializes to 0. This means the first instruction it will grab will be the instruction stored at instruction ROM stored at address offset 0. The execution engine will then decode the instruction to decipher the operation that it will perform, locate and transfer source 1 operand, locate and transfer source 2 operand (if applicable), write the opcode into the integer or matrix ALU, and then grab the result and move it to its destination. It will increment the program counter by one to grab the next instruction.


FETCH INSTRUCTION
$finish
Initial
No
read_instruction
read_instruction_data
if(opcode != stop)
Yes
decode_instruction
SOURCE 1 STATES
finish_move_1
move_source_1
find dest source 1
get source 1
find source 1
SOURCE 2 STATES
find source 2
if(opcode != MScaleIMM)
Yes
get source 2
find dest source 2
move source 2
finish move 2
No
move imm
No
result_finish
move_result
read_result
finish_math
Yes
if(status_out == 1)
wait_result
do_math
RESULT STATES
MATH STATES

IntegerALU.sv

Inputs:
input logic Clk, nRead ,nWrite, nReset;
input logic [15:0] address;
input logic [255:0] ExeDataOut;

Outputs:
output logic [255:0] IntDataOut;

Purpose: This integer ALU is capable of adding, subtracting, multiplying, or dividing two 256 bit integers. At the moment, there is no capability for detecting overflow.

Matrix_ALU.sv

Inputs:
input logic Clk, nReset, nWrite,nRead;
input logic [15:0] address;
input logic [255:0] ExeDataOut;

Outputs:
output logic [255:0] MatrixDataOut;

Purpose: This module is capable of various matrix related operations. It can add and subtract two 4x4 matrices, it can transpose a 4x4 matrix, it can scale and scaleIMM a 4x4 matrix, and perform three different matrix multiplications: 4x4 * 4x4, 4x2 * 2x4, and 2x4 * 4x2. There is also a lack of overflow detection for this module.