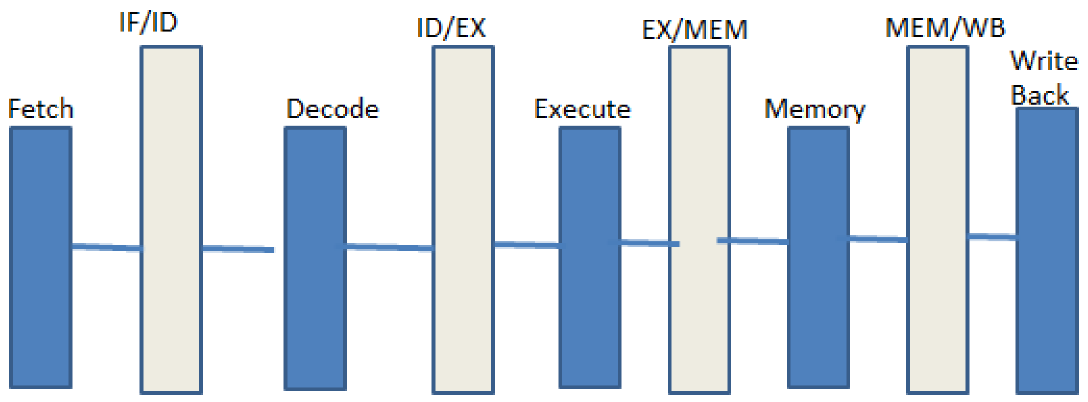
Team Members: Saroj Bardewa and Conor O'Connell

Week #5

THIS WEEK:

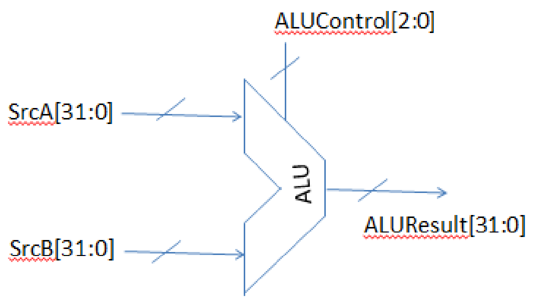
- Researched on Pipeline processor:

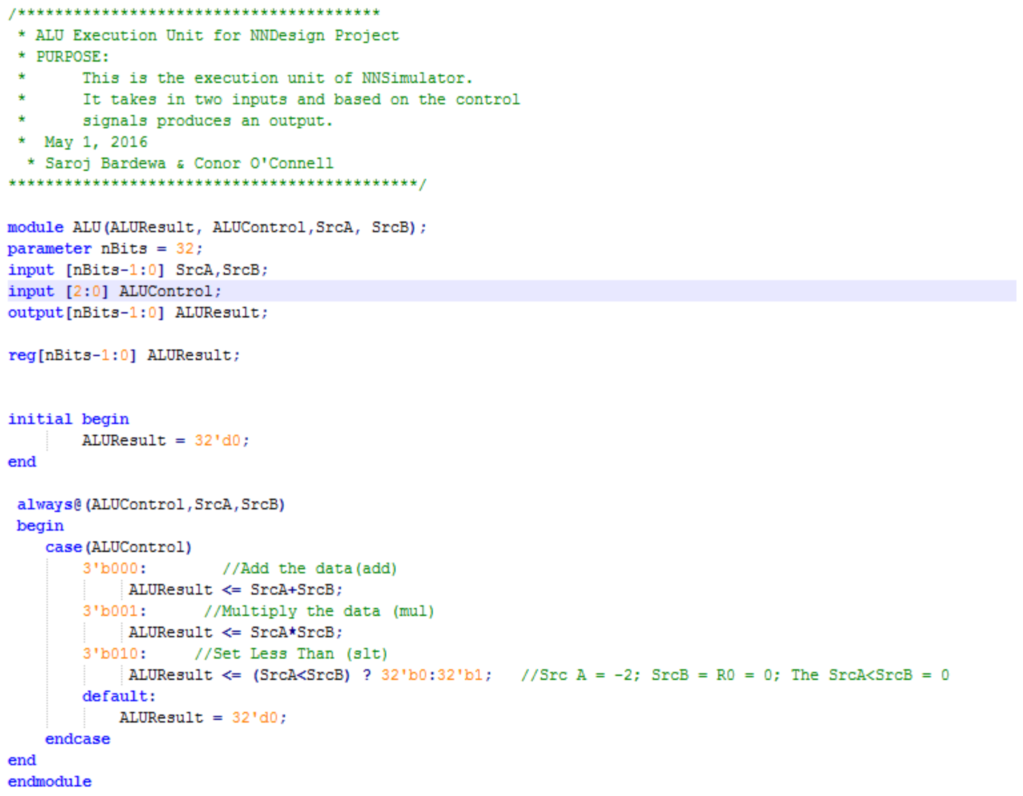
  Our pipeline processor will consists of 5 stages- Fetch, Decode, Execution, Memory and WriteBack- with Pipeline Registers – IF/ID, ID/EX, EX/MEM, and MEM/WB. These are the standard stages in a MIPS processor. As discussed in the class, these stages are necessary, to gain higher throughput than a single-cycle processor. Our design looks like this:



- Designed and simulated ALU Execution Unit:

  ALU is the execution unit of pipeline Neural Network system we are designing. To make the progress on pipeline design and also to review Verilog syntax, we wrote module ALU module as well as the testbench. This also allowed us to think about how we would program the execution unit to produce the desired result provided a given set up inputs.

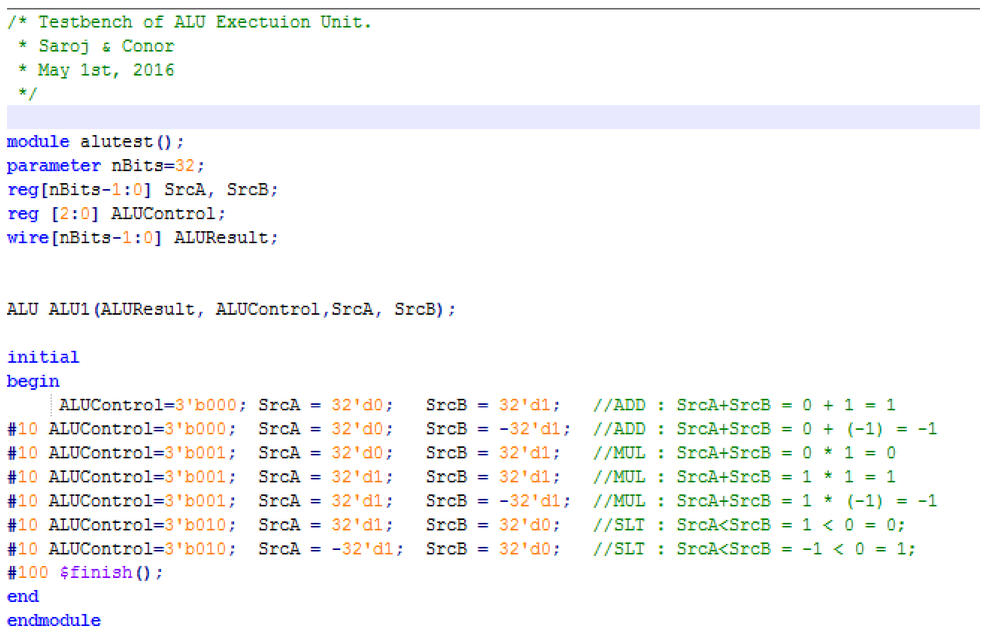




```
/*****************************************
 * ALU Execution Unit for NNDesign Project
 * PURPOSE:
 *       This is the execution unit of NNSimulator.
 *       It takes in two inputs and based on the control
 *       signals produces an output.
 *  May 1, 2016
  * Saroj Bardewa & Conor O'Connell
*********************************************/

module ALU(ALUResult, ALUControl,SrcA, SrcB);
parameter nBits = 32;
input [nBits-1:0] SrcA,SrcB;
input [2:0] ALUControl;
output[nBits-1:0] ALUResult;

reg[nBits-1:0] ALUResult;


initial begin
        ALUResult = 32'd0;
end

 always@(ALUControl,SrcA,SrcB)
 begin
    case(ALUControl)
        3'b000:        //Add the data(add)
             ALUResult <= SrcA+SrcB;
        3'b001:      //Multiply the data (mul)
             ALUResult <= SrcA*SrcB;
        3'b010:     //Set Less Than (slt)
             ALUResult <= (SrcA<SrcB) ? 32'b0:32'b1;   //Src A = -2; SrcB = R0 = 0; The SrcA<SrcB = 0
        default:
            ALUResult = 32'd0;
    endcase
end
endmodule
```

<u>Test bench for the ALU Simulation Unit:</u>

This is a basic test bench that tests all the functionality of the ALU.



```
/* Testbench of ALU Exectuion Unit.
 * Saroj & Conor
 * May 1st, 2016
 */

module alutest();
parameter nBits=32;
reg[nBits-1:0] SrcA, SrcB;
reg [2:0] ALUControl;
wire[nBits-1:0] ALUResult;


ALU ALU1(ALUResult, ALUControl,SrcA, SrcB);

initial
begin
    ALUControl=3'b000; SrcA = 32'd0;   SrcB = 32'd1;   //ADD : SrcA+SrcB = 0 + 1 = 1
#10 ALUControl=3'b000;  SrcA = 32'd0;   SrcB = -32'd1;  //ADD : SrcA+SrcB = 0 + (-1) = -1
#10 ALUControl=3'b001;  SrcA = 32'd0;   SrcB = 32'd1;   //MUL : SrcA+SrcB = 0 * 1 = 0
#10 ALUControl=3'b001;  SrcA = 32'd1;   SrcB = 32'd1;   //MUL : SrcA+SrcB = 1 * 1 = 1
#10 ALUControl=3'b001;  SrcA = 32'd1;   SrcB = -32'd1;  //MUL : SrcA+SrcB = 1 * (-1) = -1
#10 ALUControl=3'b010;  SrcA = 32'd1;   SrcB = 32'd0;   //SLT : SrcA<SrcB = 1 < 0 = 0;
#10 ALUControl=3'b010;  SrcA = -32'd1;  SrcB = 32'd0;   //SLT : SrcA<SrcB = -1 < 0 = 1;
#100 $finish();
end
endmodule
```

<u>Result of the simulation:</u>

In our simulation, we had both negative and positive numbers. The result of the simulation generated exactly the results we were expecting.

- **<u>Designed Decode unit</u>**

<u>testbed.v:</u>

This Verilog code first opens a file (including error handling for NULL file handles). It then reads the file line by line, interprets each line as a hex value, stores the value, and prints it (for debugging purposes). This will serve as a base for the decoder, the first stage of the pipeline.

It has been tested for a range of valid and invalid inputs. It accepts lines larger than 32 bits gracefully, by truncating the value. On the other hand, the line "-1" caused the code to enter an infinite loop. However, because the actual 32 bit hex representation of -1 is FFFFFFFF, this should not cause an issue.

<u>Assembler:</u>

This Python script has undergone significant revision and updating. It now supports up to 32 registers. It supports LD and ST opcodes, as well as opcodes for the the domain specific Multiply-and-Accumulate (MAC) and Set-If-Not-Negative (SINN). These specialized commands may or may not prove useful in our final submission, but we have reserved space for them in our in-coding, allowing us to make that decision at a later date.

<u>NEXT WEEK:</u>

- Work further on designing a basic pipeline NN processor.
- Extend capability of our simulation to load in a hex textfile and save the results to a hex file
- Complete the assembler and produce the instructions in hex format