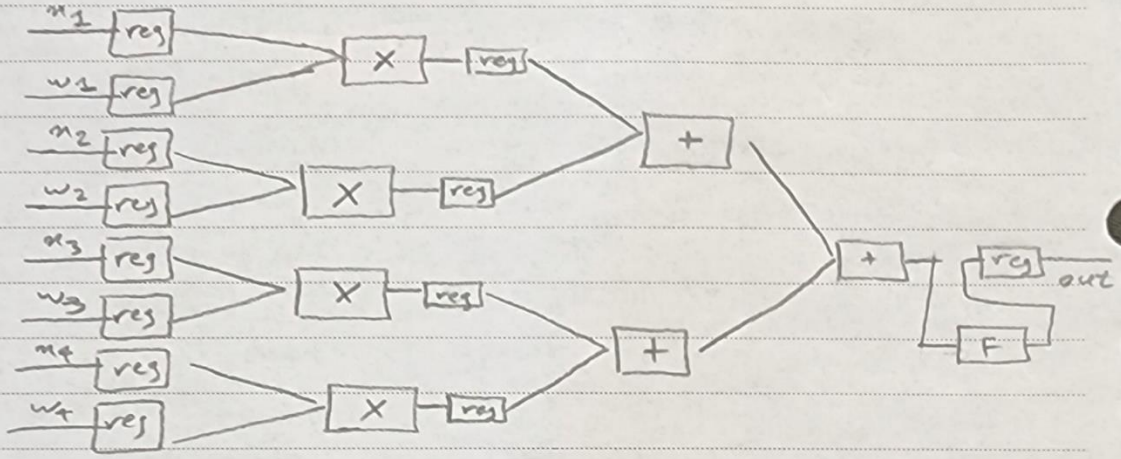
Data-Path and Components:

PU → we made changes to the processing unit such that the cycles match the discription.



this ensures that the PU itself is 3 cycles long and the first memory reading is 1 cycle.

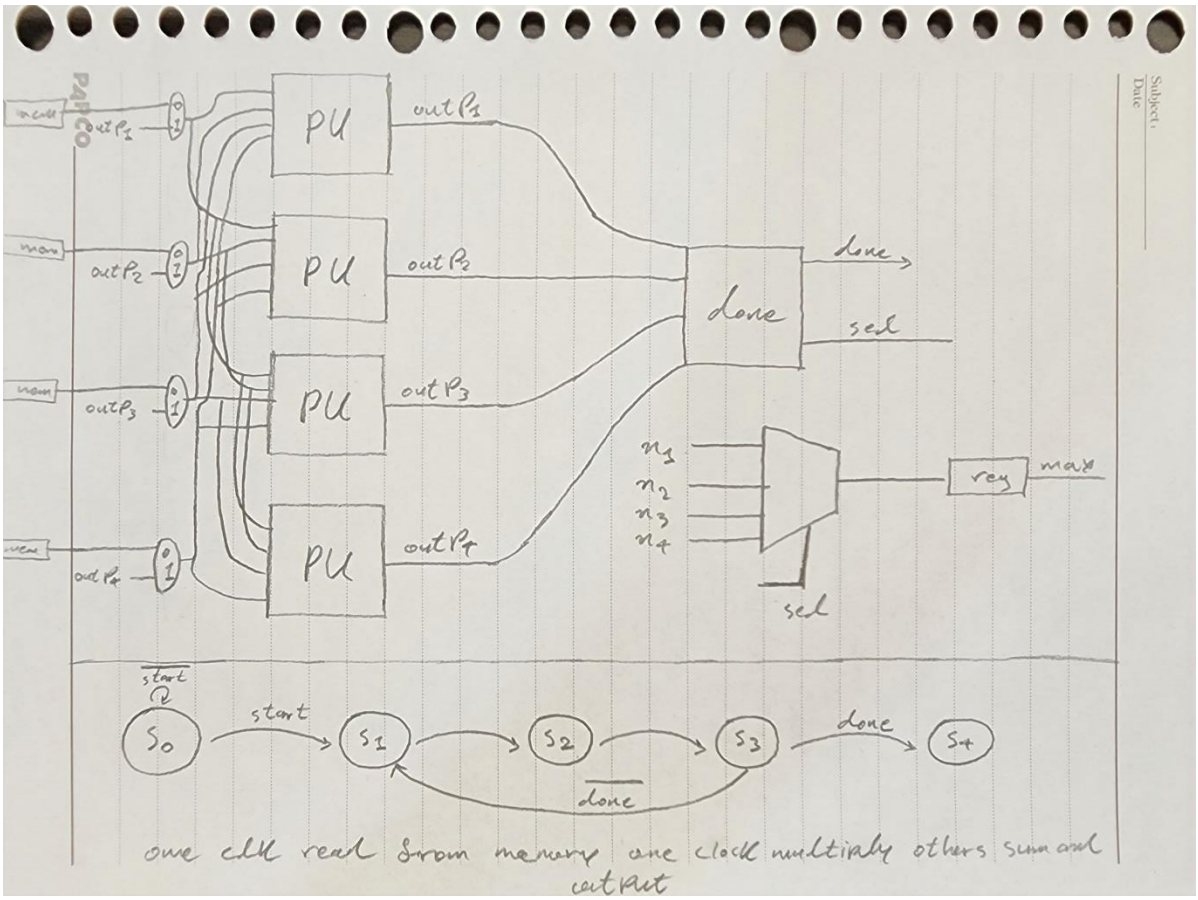
Done → this first or's all of the four inputs bits then checks the 3 zero condition.

F = (~A and ~B and ~C) or (~A and ~B and D) or ...

F → this is the f function if the input is less then 0 (negative) it makes it 0.

→ we put this at the end of out PU.

PAPCO



And:

```verilog
module And #(parameter INPUT_SIZE = 1)(input [INPUT_SIZE-1:0] A, B, output reg [INPUT_SIZE-1:0] F);

    wire [INPUT_SIZE-1:0] fTemp;

    genvar i;
    generate
        for (i = 0; i < INPUT_SIZE; i = i + 1) begin
            C2 u_C2 (
                .D00(1'b0),
                .D01(1'b0),
                .D10(1'b0),
                .D11(1'b1),
                .A1(1'b1),
                .B1(1'b1),
                .A0(A[i]),
                .B0(B[i]),
                .out(fTemp[i])
            );
        end
    endgenerate

    assign F = fTemp;
```

Or:

```
module Or #(parameter INPUT_SIZE = 1)(input [INPUT_SIZE-1:0] A, B, output reg [INPUT_SIZE-1:0] F);

    wire [INPUT_SIZE-1:0] fTemp;

    genvar i;
    generate
        for (i = 0; i < INPUT_SIZE; i = i + 1) begin
            C2 u_C22 (
                .D00(1'b0),
                .D01(1'b0),
                .D10(1'b1),
                .D11(1'b0),
                .A1(A[i]),
                .B1(B[i]),
                .A0(1'b0),
                .B0(1'b0),
                .out(fTemp[i])
            );
        end
    endgenerate

    assign F = fTemp;

endmodule
```

Xor:

```
module Xor #(parameter INPUT_WIDTH = 1) (input [INPUT_WIDTH-1:0] A, B,output reg [INPUT_WIDTH-1:0] F);

    wire [INPUT_WIDTH-1:0] fTemp;

    genvar i;
    generate
        for (i = 0; i < INPUT_WIDTH; i = i + 1) begin : mux_instances
            C2 xor_instance(.D00(1'b0), .D01(1'b1), .D10(1'b1), .D11(1'b0), .A1(A[i]), .B1(1'b0), .A0(B[i]), .B0(1'b1), .out(fTemp[i]));
        end
    endgenerate

    assign F = fTemp;

endmodule
```

Not:

```
module Not #(parameter INPUT_SIZE = 1) (input [INPUT_SIZE-1:0] A, output [INPUT_SIZE-1:0] F);

    wire [INPUT_SIZE-1:0] fTemp;

    genvar i;
    generate
        for (i = 0; i < INPUT_SIZE; i = i + 1) begin : gen_loop
            C1 u1 (.A0(1'b1), .A1(1'b0), .SA(A[i]), .B0(1'b0), .B1(1'b0), .SB(1'b0), .S0(1'b0), .S1(1'b0), .F(fTemp[i]));
        end
    endgenerate

    assign F = fTemp;

endmodule
```

Register:

```
module Reg #(parameter DATA_WIDTH = 1) (input wire [DATA_WIDTH-1:0] data_in, input wire clk, clr,en, output reg [DATA_WIDTH-1:0] data_out);

    wire[DATA_WIDTH - 1 : 0] dataOut;

    genvar i;
    generate
        for (i = 0; i < DATA_WIDTH; i = i + 1) begin : gen_block
            S2 inst_s2 (
                .D00(data_out[i]),
                .D01(data_in[i]),
                .D10(1'b0),
                .D11(1'b0),
                .A1(1'b0),
                .B1(1'b0),
                .A0(en),
                .B0(1'b1),
                .clr(clr),
                .clk(clk),
                .out(dataOut[i])
            );
        end
    endgenerate

    assign data_out = dataOut;

endmodule
```

We've built the other components based on the above gates:

for adder we used a standard full-adder from DLD Course and with a generative for made a n bit adder. for multiplier we used an array multiplier and handled the bits using Xor and two's complement. for our controller we used two Mux's one for choosing the ns and one for choosing the output signals and we used a register so that it's sync'd with our clock.

Controller:

```
`define S0 3'd0
`define S1 3'd1
`define S2 3'd2
`define S3 3'd3
`define S4 3'd4

module Controller(input done, output reg sel, ld, input clk, rst, start, output reg ctrDone);

    wire[2:0] nsSel0, nsSel1, nsSel2, nsSel3, nsSel4;
    wire[2:0] nsSelOut, ps, out;
    wire[3:0] pss;

    assign nsSel1 = `S2;
    assign nsSel2 = `S3;
    assign nsSel4 = `S4;
    assign pss = {1'b0, ps};

    Reg #(3) regg (.data_in(nsSelOut), .clk(clk), .clr(rst), .en(1'b1), .data_out(ps));

    Mux2to1 #(3) mx1 (.A(`S0), .B(`S1), .S(start), .F(nsSel0));
    Mux2to1 #(3) mx2 (.A(`S1), .B(`S4), .S(done), .F(nsSel3));

    Mux16to1 #(3) mxx1 (.A(nsSel0), .B(nsSel1), .C(nsSel2), .D(nsSel3), .E(nsSel4), .F(), .G(), .H(), .I(), .J(), .K(), .L(), .M(), .N(), .O(), .P(), .S(pss), .FF(nsSelOut));
    Mux16to1 #(3) mxx2 (.A(3'b010), .B(3'b100), .C(3'b100), .D(3'b100), .E(3'b101), .F(), .G(), .H(), .I(), .J(), .K(), .L(), .M(), .N(), .O(), .P(), .S(pss), .FF(out));

    assign sel = out[2];
    assign ld = out[1];
    assign ctrDone = out[0];

endmodule
```