UNIVERSIDAD DE PUERTO RICO
RECINTO UNIVERSITARIO DE MAYAGUEZ
FACULTAD DE INGENIERIA
DEPARTAMENTO DE INGENIERIA

COMPUTER ARCHITECTURE

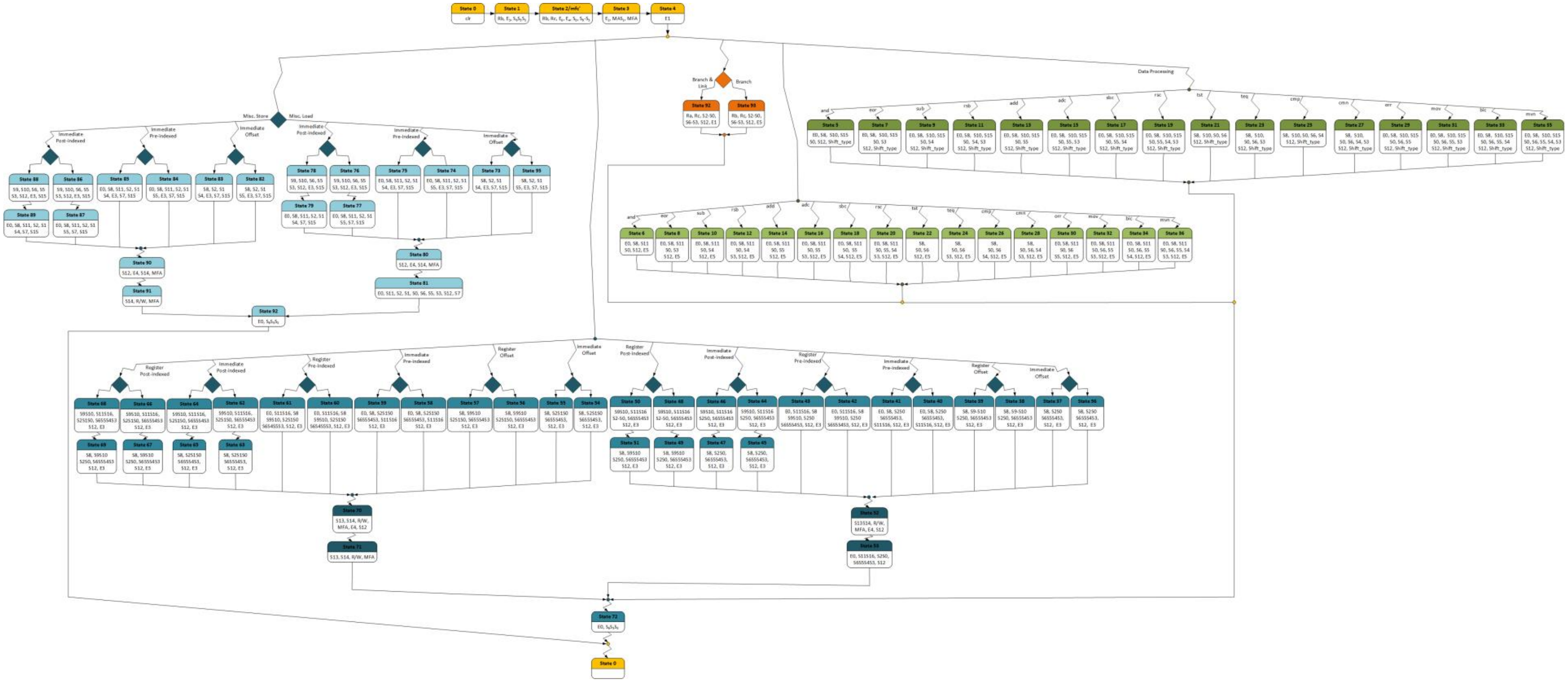
Report

Hector Hernandez
Pedro Colon
Marie Nazario

Prof. Rodriguez
ICOM4215-080

Contents

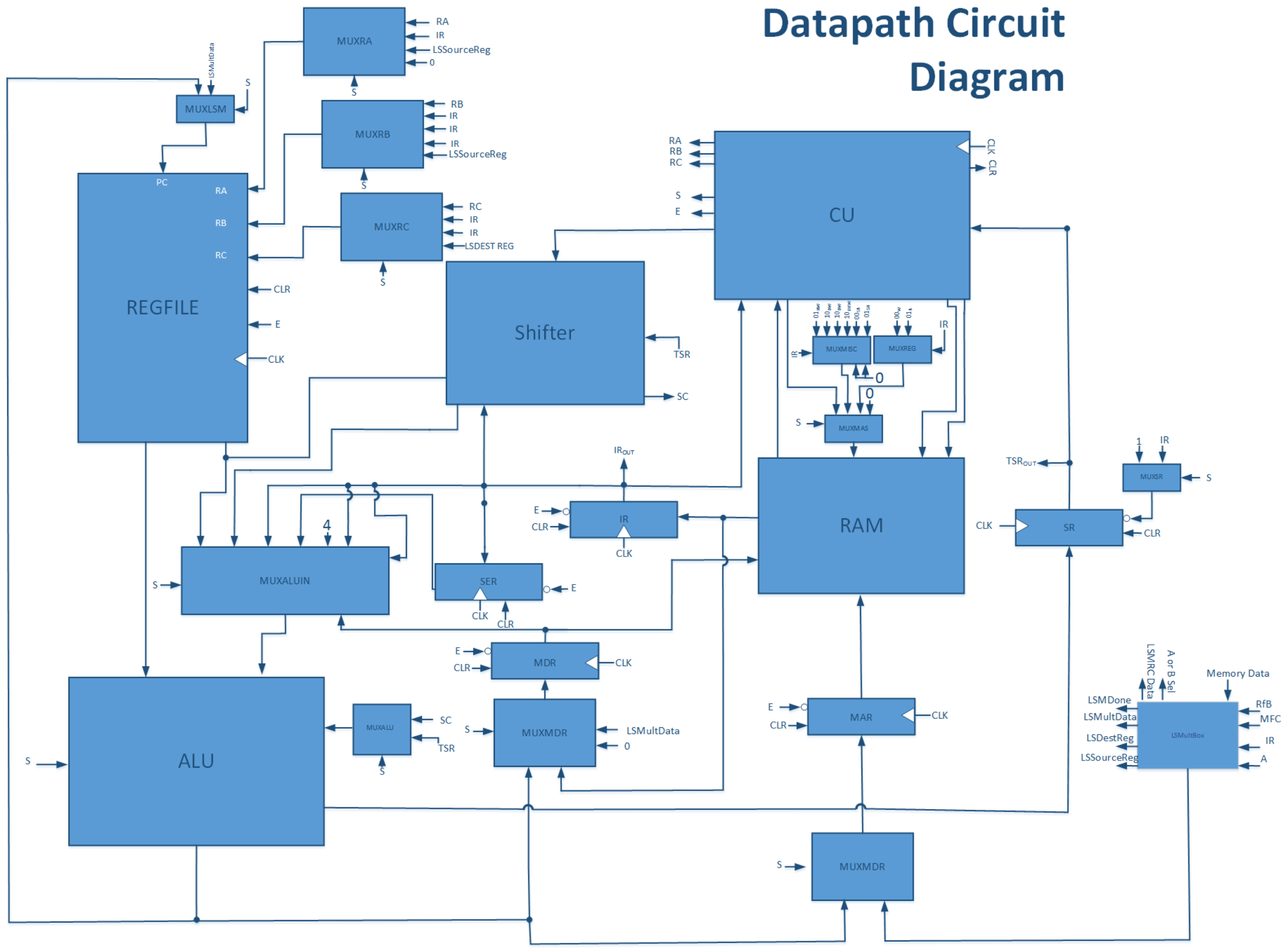
ASMD Diagram



State 0
clr
State 1
Rb, E_2, $S_0S_1S_2$
State 2/mfc'
Rb, Rc, E_0, E_4, S_0, S_5-S_3
State 3
E_1, MAS_1, MFA
State 4
E1
Branch & Link
Branch
State 92
Ra, Rc, S2-S0, S6-S3, S12, E1
State 93
Rb, Rc, S2-S0, S6-S3, S12, E5
Data Processing
and
eor
sub
rsb
add
adc
sbc
rsc
tst
teq
cmp
cmn
orr
mov
bic
mvn
State 5
E0, S8, S10, S15 S0, S12, Shift_type
State 7
E0, S8, S10, S15 S0, S3 S12, Shift_type
State 9
E0, S8, S10, S15 S0, S4 S12, Shift_type
State 11
E0, S8, S10, S15 S0, S4, S3 S12, Shift_type
State 13
E0, S8, S10, S15 S0, S5 S12, Shift_type
State 15
E0, S8, S10, S15 S0, S5, S3 S12, Shift_type
State 17
E0, S8, S10, S15 S0, S5, S4 S12, Shift_type
State 19
E0, S8, S10, S15 S0, S5, S4, S3 S12, Shift_type
State 21
S8, S10, S0, S6 S12, Shift_type
State 23
S8, S10, S0, S6, S3 S12, Shift_type
State 25
S8, S10, S0, S6, S4 S12, Shift_type
State 27
S8, S10, S0, S6, S4, S3 S12, Shift_type
State 29
E0, S8, S10, S15 S0, S6, S5 S12, Shift_type
State 31
E0, S8, S10, S15 S0, S6, S5, S3 S12, Shift_type
State 33
E0, S8, S10, S15 S0, S6, S5, S4 S12, Shift_type
State 35
E0, S8, S10, S15 S0, S6, S5, S4, S3 S12, Shift_type
State 6
E0, S8, S11 S0, S12, E5
State 8
E0, S8, S11 S0, S3 S12, E5
State 10
E0, S8, S11 S0, S4 S12, E5
State 12
E0, S8, S11 S0, S4 S3, S12, E5
State 14
E0, S8, S11 S0, S5 S12, E5
State 16
E0, S8, S11 S0, S5 S3, S12, E5
State 18
E0, S8, S11 S0, S5 S4, S12, E5
State 20
E0, S8, S11 S0, S5, S4 S3, S12, E5
State 22
S8, S0, S6 S12, E5
State 24
S8, S0, S6 S3, S12, E5
State 26
S8, S0, S6 S4, S12, E5
State 28
S8, S0, S6, S4 S3, S12, E5
State 30
E0, S8, S11 S0, S6 S5, S12, E5
State 32
E0, S8, S11 S0, S6, S5 S3, S12, E5
State 34
E0, S8, S11 S0, S6, S5 S4, S12, E5
State 36
E0, S8, S11 S0, S6, S5, S4 S3, S12, E5
Misc. Store
Misc. Load
Immediate Post-indexed
Immediate Pre-indexed
Immediate Offset
State 88
S9, S10, S6, S5 S3, S12, E3, S15
State 86
S9, S10, S6, S5 S3, S12, E3, S15
State 89
E0, S8, S11, S2, S1 S4, S7, S15
State 87
E0, S8, S11, S2, S1 S5, S7, S15
State 85
E0, S8, S11, S2, S1 S4, E3, S7, S15
State 84
E0, S8, S11, S2, S1 S5, E3, S7, S15
State 83
S8, S2, S1 S4, E3, S7, S15
State 82
S8, S2, S1 S5, E3, S7, S15
State 90
S12, E4, S14, MFA
State 91
S14, R/W, MFA
State 78
S9, S10, S6, S5 S3, S12, E3, S15
State 76
S9, S10, S6, S5 S3, S12, E3, S15
State 79
E0, S8, S11, S2, S1 S4, S7, S15
State 77
E0, S8, S11, S2, S1 S5, S7, S15
State 75
E0, S8, S11, S2, S1 S4, E3, S7, S15
State 74
E0, S8, S11, S2, S1 S5, E3, S7, S15
State 73
S8, S2, S1 S4, E3, S7, S15
State 95
S8, S2, S1 S5, E3, S7, S15
State 80
S12, E4, S14, MFA
State 81
E0, S11, S2, S1, S0, S6, S5, S3, S12, S7
State 92
E0, $S_0S_1S_2$
Register Post-indexed
Immediate Post-indexed
Register Pre-indexed
Immediate Pre-indexed
Register Offset
Immediate Offset
State 68
S9S10, S11S16, S25150, S6S55453 S12, E3
State 66
S9S10, S11S16, S25150, S6S55453 S12, E3
State 64
S9S10, S11S16, S25150, S6S55453 S12, E3
State 62
S9S10, S11S16, S25150, S6S55453 S12, E3
State 69
S8, S9S10 S2S0, S6S55453 S12, E3
State 67
S8, S9S10 S2S0, S6S55453 S12, E3
State 65
S8, S25150 S6S55453, S12, E3
State 63
S8, S25150 S6S55453, S12, E3
State 61
E0, S11S16, S8 S9S10, S25150 S6S45S53, S12, E3
State 60
E0, S11S16, S8 S9S10, S25150 S6S45S53, S12, E3
State 59
E0, S8, S25150 S6S55453, S11S16 S12, E3
State 58
E0, S8, S25150 S6S55453, S11S16 S12, E3
State 57
S8, S9S10 S25150, S6S55453 S12, E3
State 56
S8, S9S10 S25150, S6S55453 S12, E3
State 55
S8, S25150 S6S55453, S12, E3
State 54
S8, S25150 S6S55453, S12, E3
State 70
S13, S14, R/W, MFA, E4, S12
State 71
S13, S14, R/W, MFA
State 50
S9S10, S11S16 S2-S0, S6S55453 S12, E3
State 48
S9S10, S11S16 S2-S0, S6S55453 S12, E3
State 46
S9S10, S11S16 S2S0, S6S55453 S12, E3
State 44
S9S10, S11S16 S2S0, S6S55453 S12, E3
State 51
S8, S9S10 S2S0, S6S55453 S12, E3
State 49
S8, S9S10 S2S0, S6S55453 S12, E3
State 47
S8, S2S0, S6S55453, S12, E3
State 45
S8, S2S0, S6S55453, S12, E3
State 43
E0, S11S16, S8 S9S10, S2S0 S6S55453, S12, E3
State 42
E0, S11S16, S8 S9S10, S2S0 S6S55453, S12, E3
State 41
E0, S8, S2S0 S6S55453, S11S16, S12, E3
State 40
E0, S8, S2S0 S6S55453, S11S16, S12, E3
State 39
S8, S9-S10 S2S0, S6S55453 S12, E3
State 38
S8, S9-S10 S2S0, S6S55453 S12, E3
State 37
S8, S2S0 S6S55453, S12, E3
State 36
S8, S2S0 S6S55453, S12, E3
State 52
S13S14, R/W, MFA, E4, S12
State 53
E0, S11S16, S2S0, S6S55453, S12
State 72
E0, $S_0S_1S_2$
State 0

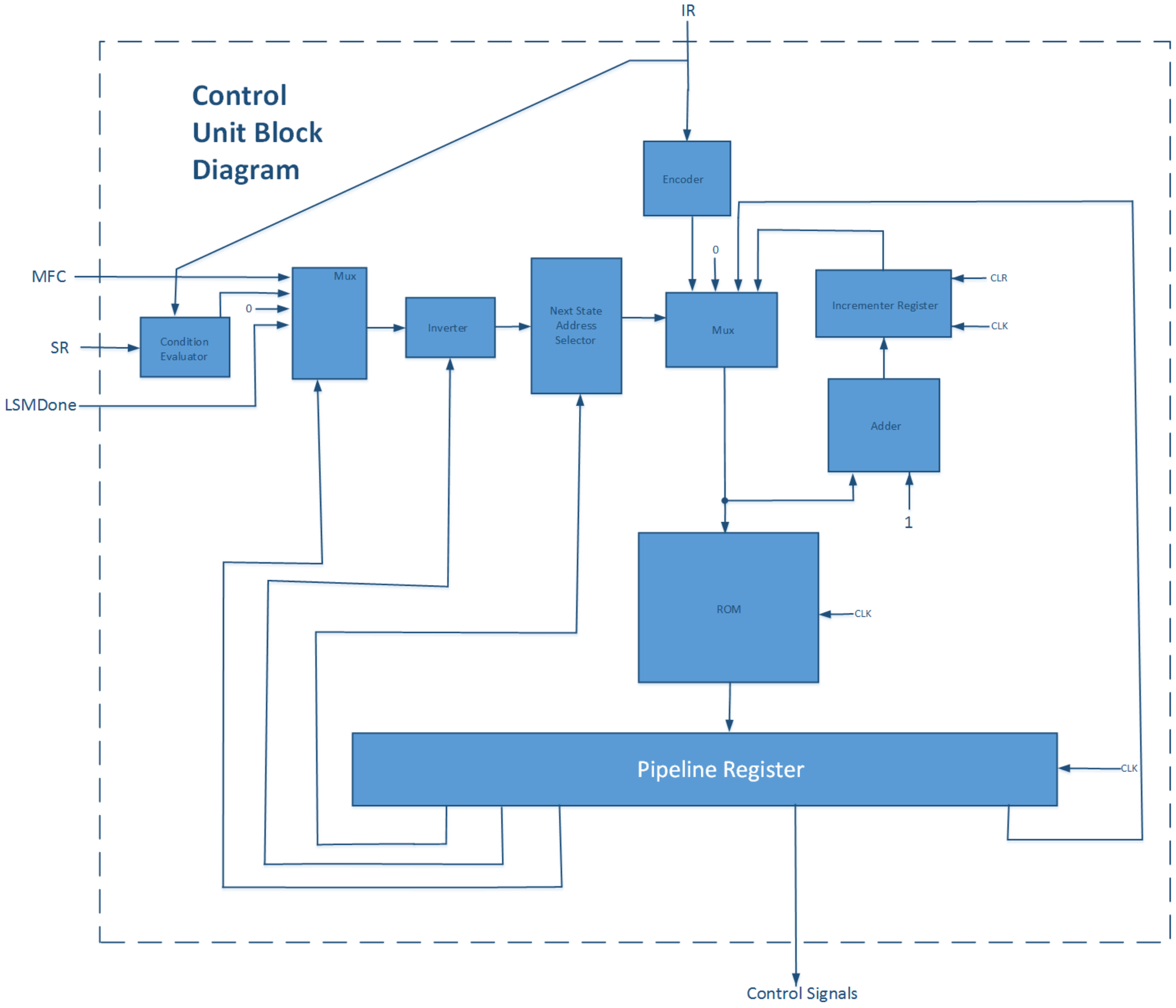
Block Diagrams

Data Path



Datapath Circuit Diagram

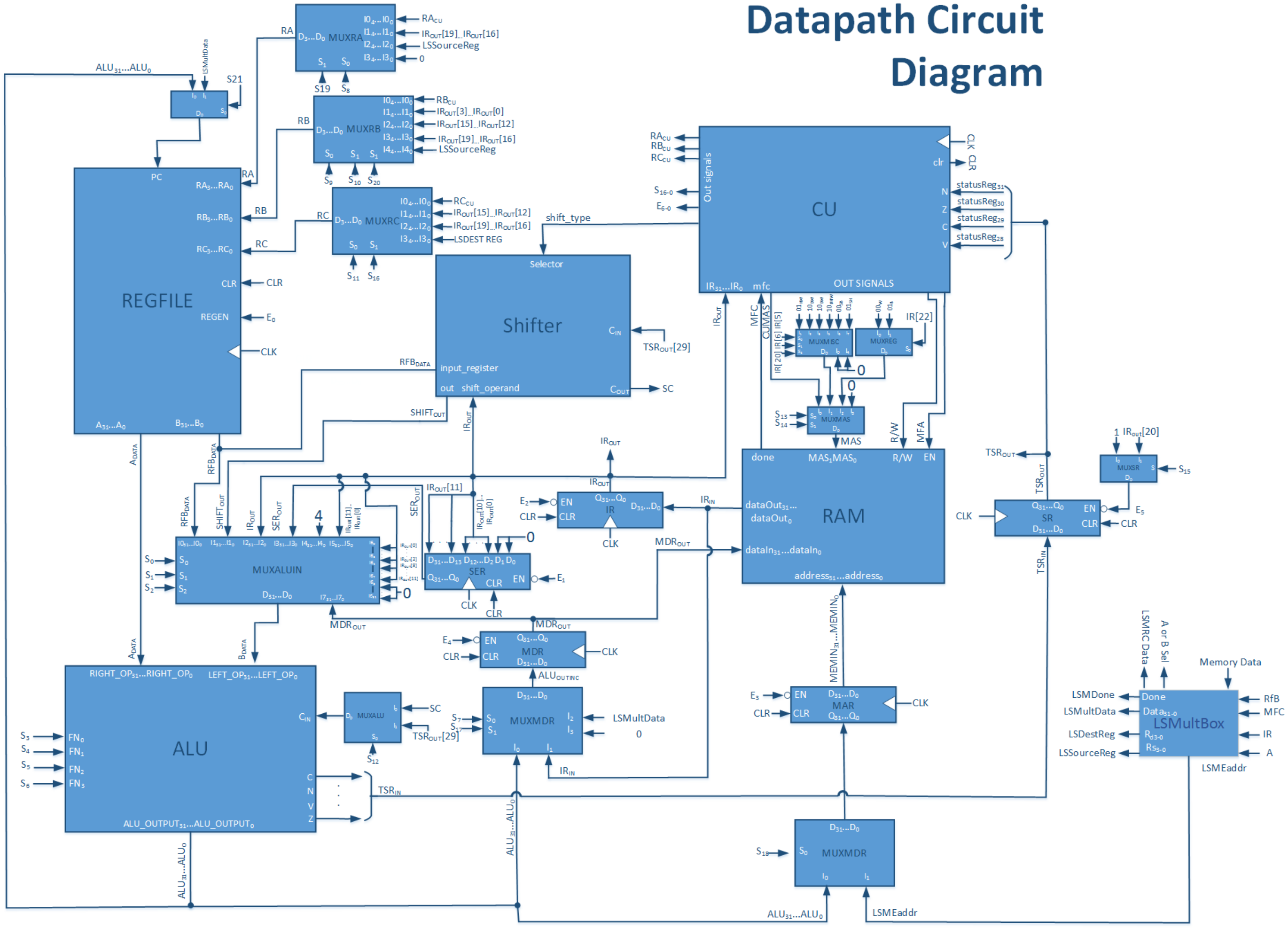
Control Unit



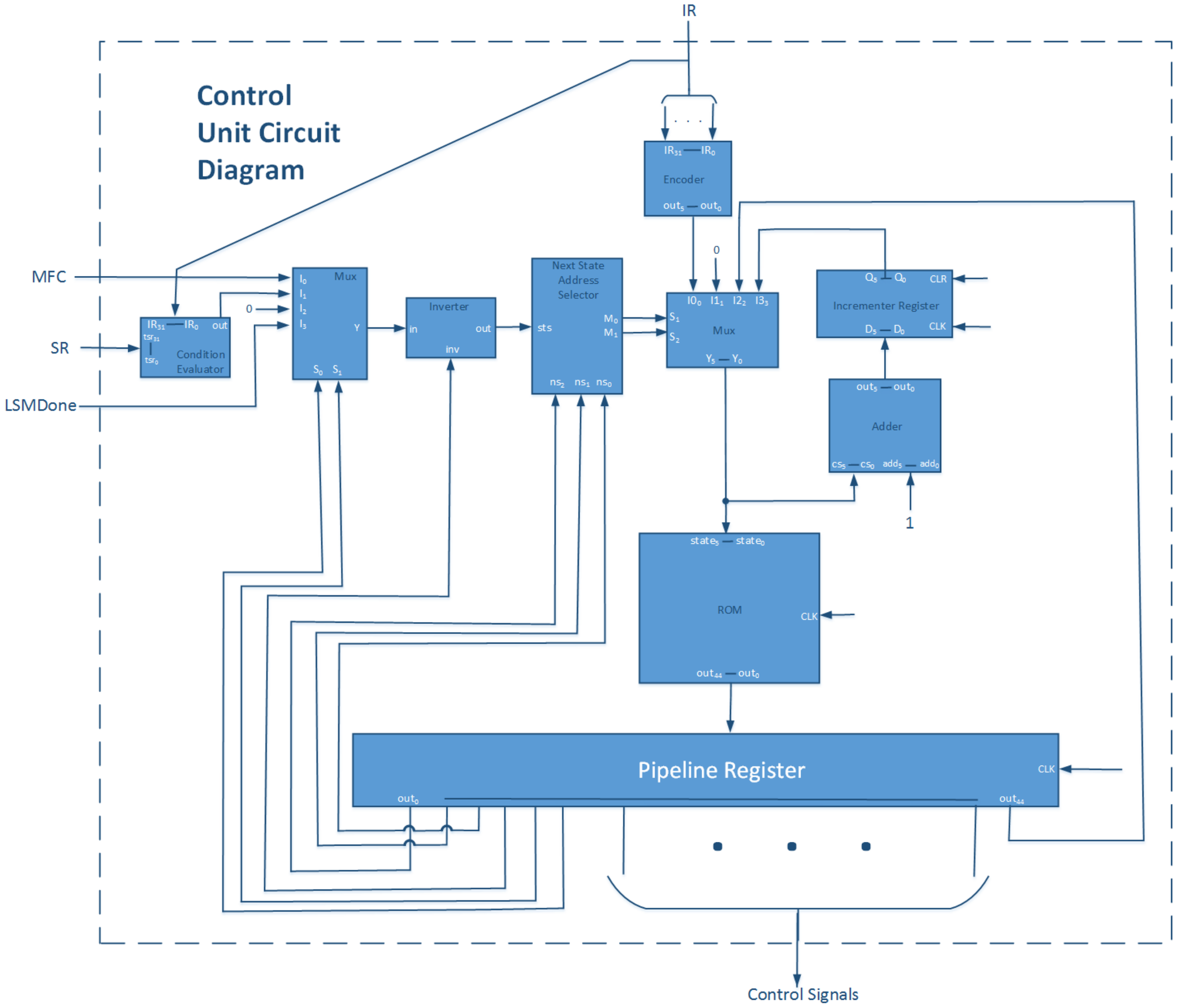
Circuit Diagrams

Data Path

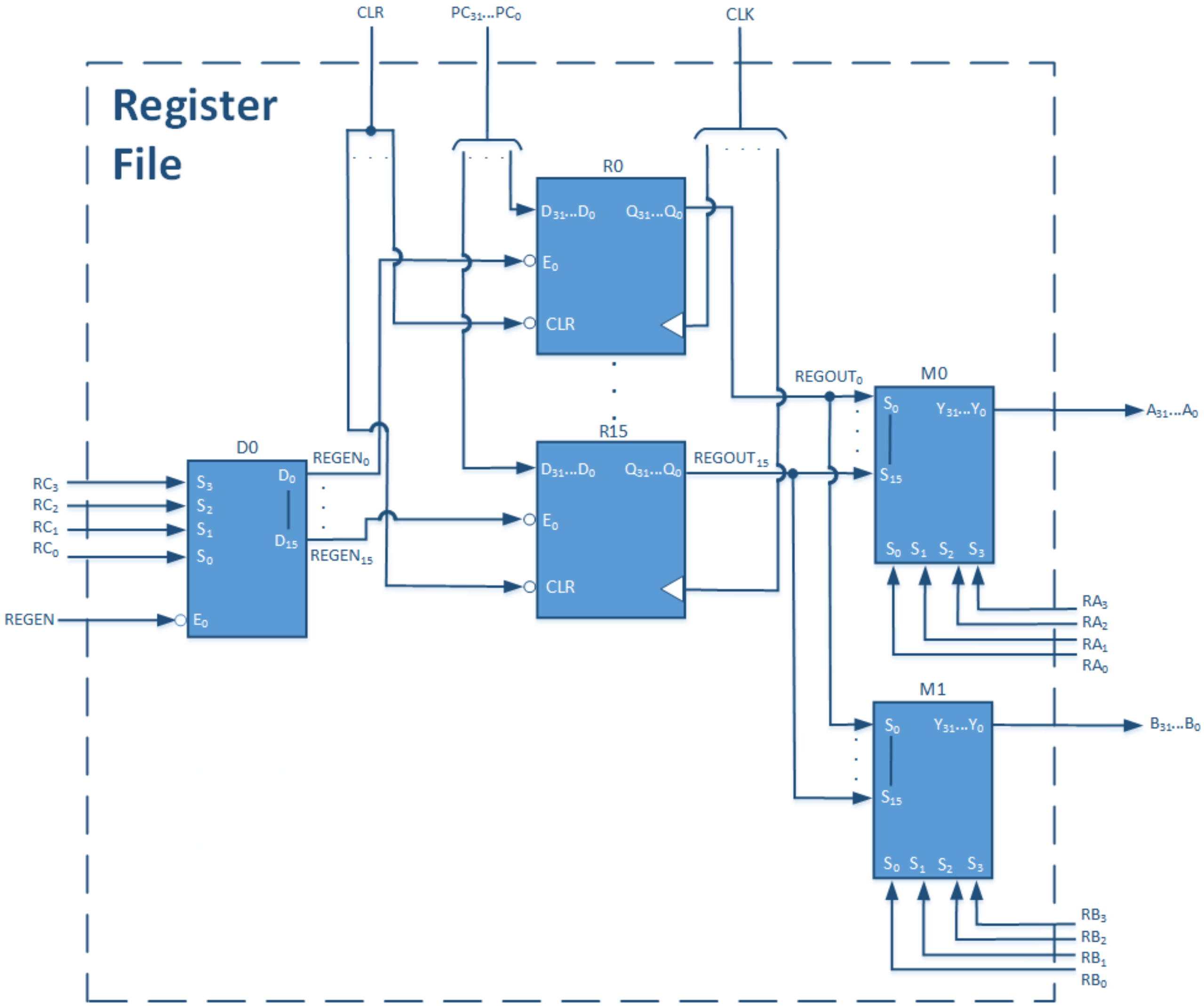
Datapath Circuit Diagram



Control Unit



Register File



Microprogram

```
mem[0][58:0]  = 59'b0000000001100000000101ZZZZ0ZZZZ00ZZZZ000000000000111100000010;
mem[1][58:0]  = 59'b0000000001100000000111ZZZZ0111100ZZZZ00000110100110100000010;
mem[2][58:0]  = 59'b000000000110000000011011110ZZZZ00111100100010000111100000010;
mem[3][58:0]  = 59'b000000001011000001110ZZZZ0ZZZZ00ZZZZ000000000000101100001011;
mem[4][58:0]  = 59'b00000001100100000001110000000000000ZZZZ000000000000011100000010;
mem[6][58:0]  = 59'b00000001010110111001000001000000ZZZZ010010000101111101000010;
mem[5][58:0]  = 59'b00000001010110111001000001000001ZZZZ010010000111111101000010;
mem[8][58:0]  = 59'b0000000101011011100100000100000000000010010001101111101000010;
mem[7][58:0]  = 59'b000000010101101110010000010000010000010010001111111101000010;
mem[10][58:0] = 59'b0000000101011011100100000100000000000010010010101111101000010;
mem[9][58:0]  = 59'b000000010101101110010000010000010000010010010111111101000010;
mem[12][58:0] = 59'b0000000101011011100100000100000000000010010011101111101000010;
mem[11][58:0] = 59'b000000010101101110010000010000010000010010011111111101000010;
mem[14][58:0] = 59'b0000000101011011100100000100000000000010010100101111101000010;
mem[13][58:0] = 59'b000000010101101110010000010000010000010010100111111101000010;
mem[16][58:0] = 59'b0000000101011011100100000100000000000010010101101111101000010;
mem[15][58:0] = 59'b000000010101101110010000010000010000010010101111111101000010;
mem[18][58:0] = 59'b0000000101011011100100000100000000000010010110101111101000010;
mem[17][58:0] = 59'b000000010101101110010000010000010000010010110111111101000010;
mem[20][58:0] = 59'b0000000101011011100100000100000000000010010111101111101000010;
mem[19][58:0] = 59'b000000010101101110010000010000010000010010111111111101000010;
mem[22][58:0] = 59'b0000000101011011100110000100000000000000011000101111101000010;
mem[21][58:0] = 59'b000000010101101110011000010000010000000011000111111101000010;
mem[24][58:0] = 59'b0000000101011011100110000100000000000000011001101111101000010;
mem[23][58:0] = 59'b000000010101101110011000010000010000000011001111111101000010;
mem[26][58:0] = 59'b0000000101011011100110000100000000000000011010101111101000010;
mem[25][58:0] = 59'b000000010101101110011000010000010000000011010111111101000010;
mem[28][58:0] = 59'b0000000101011011100110000100000000000000011011101111101000010;
mem[27][58:0] = 59'b000000010101101110011000010000010000000011011111111101000010;
mem[30][58:0] = 59'b0000000101011011100100000100000000000010011100101111101000010;
mem[29][58:0] = 59'b000000010101101110010000010000010000010011100111111101000010;
mem[32][58:0] = 59'b0000000101011011100100000100000000000010011101101111101000010;
mem[31][58:0] = 59'b000000010101101110010000010000010000010011101111111101000010;
mem[34][58:0] = 59'b0000000110011011100100000100000000000010011110101111101000010;
mem[33][58:0] = 59'b000000010101101110010000010000010000010011110111111101000010;
mem[36][58:0] = 59'b0000000101011011100100000100000000000010011111101111101000010;
mem[35][58:0] = 59'b000000010101101110010000010000010000010011111111111101000010;
```

```
mem[96][58:0] = 59'b00000001010101101001100001000000000000101010010110100000010;
mem[37][58:0] = 59'b00000001010101101001100001000000000000101001010110100000010;
mem[38][58:0] = 59'b00000001010101101001100001000001000000000100101101000000010;
mem[39][58:0] = 59'b00000001010101101001100001000001000000000010101101000000010;
mem[40][58:0] = 59'b00000001010111000011000001000000000010101010010110100000010;
mem[41][58:0] = 59'b00000001010111000011000001000000000010101001010110100000010;
mem[42][58:0] = 59'b00000001010111000011000001000001000010000010010110100000010;
mem[43][58:0] = 59'b00000001010111000011000001000001000010000001010110100000010;
mem[44][58:0] = 59'b00000001011101011011100000000011000000000110110110100000010;
mem[46][58:0] = 59'b00000001011101011111100000000011000000000110110110100000010;
mem[45][58:0] = 59'b00000001010111000011000001000000000010101010010111100000010;
mem[47][58:0] = 59'b00000001010111000011000001000000000010101001010111100000010;
mem[48][58:0] = 59'b00000001011101100011100000000011000000000110110110100000010;
mem[50][58:0] = 59'b00000001011101100111100000000011000000000110110110100000010;
mem[49][58:0] = 59'b00000001010111000011000001000001000010000010010111100000010;
mem[51][58:0] = 59'b00000001011111000011000001000001000010000001010111100000010;
mem[97][58:0]=59'b0000000101010110100 10ZZZZXZZZZXXZZZZXX00011011111111110000010;
mem[52][58:0] = 59'b00000000101101101001100000000000000000001111101101110101 00011;
mem[53][58:0] = 59'b00000001010110010001000000000000000000011111101111111110000010;
mem[54][58:0] = 59'b00000001010110001101100001000000000000101010010110100000010;
mem[55][58:0] = 59'b00000001010110001101100001000000000000101001010110100000010;
mem[56][58:0] = 59'b00000001010110001101100001000001000000000100101101000000010;
mem[57][58:0] = 59'b00000001010110001101100001000001000000000010101101000000010;
mem[58][58:0] = 59'b00000001010111000101000001000000000010101010010110100000010;
mem[59][58:0] = 59'b00000001010111000101000001000000000010101001010110100000010;
mem[60][58:0] = 59'b00000001010111000101000001000001000010000010010110100000010;
mem[61][58:0] = 59'b00000001010111000101000001000001000010000001010110100000010;
mem[62][58:0] = 59'b00000001011101111111100000000011000000000110110110100000010;
mem[64][58:0] = 59'b00000001011110000011100000000011000000000110110110100000010;
mem[63][58:0] = 59'b00000001010111000101000001000000000010101010010111100000010;
mem[65][58:0] = 59'b00000001010111000101000001000000000010101001010111100000010;
mem[66][58:0] = 59'b00000001011110000111100000000011000000000110110110100000010;
mem[68][58:0] = 59'b00000001011110001011100000000011000000000110110110100000010;
mem[67][58:0] = 59'b00000001010111000101000001000001000010000010010111100000010;
mem[69][58:0] = 59'b00000001011111000101000001000001000010000001010111100000010;
mem[98][58:0] = 59'b00000001010110001101 0ZZZZXZZZZXXZZZZXX00011011111111110000010;
mem[70][58:0] = 59'b00000001011110001111100000000010000000000110110111000100011;
mem[71][58:0] = 59'b00000000101110001111100000000000000000000110110111100100001;
mem[72][58:0] = 59'b00000001010100000001 10ZZZZXZZZZXXZZZZXX00011011111111110000010;
```

```
mem[95][58:0] = 59'b00000001010110100001100001000000000000110010010110111000010;
mem[73][58:0] = 59'b00000001010111000111100001000000000000110001010110111000010;
mem[74][58:0] = 59'b00000001010110100001000001000000000010110010010110111000010;
mem[75][58:0] = 59'b00000001010111000111000001000000000010110001010110111000010;
mem[76][58:0] = 59'b00000001011110011011100000000011000000000110110110101000010;
mem[78][58:0] = 59'b00000001011110011111100000000011000000000110110110101000010;
mem[77][58:0] = 59'b00000001010111000111000001000000000010110010000111111000010;
mem[79][58:0] = 59'b00000001011111000111000001000000000010110001000111111000010;
mem[99][58:0] = 59'b0000000101011010000010ZZZZXZZZZXXZZZZXX00011011111111110000010;
mem[80][58:0] = 59'b00000000101110100001100000000000000000000110110111010010011;
mem[81][58:0] = 59'b00000001010110111001000000000000000000111111011011101001001 0;
mem[82][58:0] = 59'b00000001010110110101100001000000000000110010000110111000010;
mem[83][58:0] = 59'b00000001010110110101100001000000000000110001000110111000010;
mem[84][58:0] = 59'b00000001010111001001000001000000000010110010000110111000010;
mem[85][58:0] = 59'b00000001010111001001000001000000000010110001000110111000010;
mem[86][58:0] = 59'b00000001011110101111100000000011000000000110110110101000010;
mem[88][58:0] = 59'b00000001011110110011100000000011000000000110110110101000010;
mem[87][58:0] = 59'b00000001010111001001000001000000000010110010000111111000010;
mem[89][58:0] = 59'b00000001011111001001000001000000000010110001000111111000010;
mem[100][58:0]=59'b00000001010110110101 0ZZZZXZZZZXXZZZZXX00011011111111110000010;
mem[90][58:0] = 59'b00000001011110110111100000000010000000000110100111000010001;
mem[91][58:0] = 59'b00000000101110110111100000000000000000000110110111100010011;
mem[92][58:0]=59'b0000000101010000000110ZZZZXZZZZXXZZZZXX00011011111111110000010;
mem[93][58:0] = 59'b00000000011110111101000000111100111000000110110000000000010;
mem[94][58:0] = 59'b00000000010110111001011110000000111100011010010100000000010;
mem[101][58:0]=59'b1111111110111100101 10ZZZZ0ZZZZ00ZZZZ11111110110110000001011;
mem[102][58:0]=59'b111111110101000000110ZZZZXZZZZXXZZZZXX11111011011111 0001010;
mem[105][58:0]=59'b111111110101110011110ZZZZ0ZZZZ00ZZZZ11000110110110000001000;
mem[103][58:0]=59'b111111111011110011110ZZZZ0ZZZZ00ZZZZ11000110110110000001001;
mem[104][58:0]=59’b111111110101000000110ZZZZXZZZZXXZZZZXX00011011011111 0001000;
```

Appendices

Appendix A: Code Implementation

```
/*
Shifter operand === RIGHTOP
0000 AND Logical AND Rd := Rn AND shifteroperand
0001 EOR Logical Exclusive OR Rd := Rn EOR shifteroperand
0010 SUB Subtract Rd := Rn - shifteroperand
0011 RSB Reverse Subtract Rd := shifteroperand - Rn
0100 ADD Add Rd := Rn + shifteroperand
0101 ADC Add with Carry Rd := Rn + shifteroperand + Carry Flag
0110 SBC Subtract with Carry Rd := Rn - shifteroperand - NOT(Carry Flag)
0111 RSC Reverse Subtract with Carry Rd := shifteroperand - Rn - NOT(Carry Flag)
1000 TST Test Update flags after Rn AND shifteroperand
1001 TEQ Test Equivalence Update flags after Rn EOR shifteroperand
1010 CMP Compare Update flags after Rn - shifteroperand
1011 CMN Compare Negated Update flags after Rn + shifteroperand
1100 ORR Logical (inclusive) OR Rd := Rn OR shifteroperand
1101 MOV Move Rd := shifteroperand (no first operand)
1110 BIC Bit Clear Rd := Rn AND NOT(shifteroperand)
1111 MVN Move Not Rd := NOT shifteroperand (no first operand)
*/
module ALU(output reg [31:0]ALUOUTPUT, output reg Z,N,C, V, input  [31:0] LEFTOP,RIGHTOP, input  [3:0]FN, input  CIN);
    reg [31:0] TEMP, TEMP1;
    reg CTEMP;
    initial begin
            N = 0;
            C = 0;
            V = 0;
            Z = 0;
    end
```

```
always @(LEFTOP, RIGHTOP, FN, CIN) begin

    case(FN)
        //AND
        4'b0000: begin
            //Set the output and C flag
            {ALUOUTPUT[31:0]} = LEFTOP[31:0] & RIGHTOP[31:0];
            C = CIN;
            //Set the N flag
            N = ALUOUTPUT[31];
            //Set the Z flag
            if(ALUOUTPUT==0)
                Z = 1;
            else
                Z = 0;
        end
        // //EOR
        4'b0001: begin
            {C,ALUOUTPUT[31:0]} = LEFTOP[31:0] ^ RIGHTOP[31:0];
            //Set the N flag
            N = ALUOUTPUT[31];
            //Set the Z flag
            if(ALUOUTPUT==0)
                Z = 1;
            else
                Z = 0;
        end
        // //SUB
        4'b0010: begin
            {CTEMP,ALUOUTPUT[31:0]} = LEFTOP[31:0] -  RIGHTOP[31:0];
            C = ~CTEMP;
            //Set the N flag
    N = ALUOUTPUT[31];
    //Set the Z flag
    if(ALUOUTPUT==0)
        Z = 1;
    else
        Z = 0;
    //Set the overflow flag
    //Check for 2's complement overflow
    TEMP1 = - RIGHTOP;
    if((LEFTOP[31]==TEMP1[31]))
    begin
        if(LEFTOP[31]!=TEMP[31])
            V=1;
        else
            V=0;
        end
    else
        begin
            if((LEFTOP[31]!=TEMP1[31])&&(TEMP[31]==TEMP1))
                V=1;
            else
                V=0;
        end
end
// //RSB
4'b0011: begin
    {CTEMP,ALUOUTPUT[31:0]} = RIGHTOP[31:0]-  LEFTOP[31:0];
    C = ~CTEMP;
    //Set the N flag
    N = ALUOUTPUT[31];
    //Set the Z flag
    if(ALUOUTPUT==0)
```

```
        Z = 1;
    else
        Z = 0;
    //Set the overflow flag
    //Check for 2's complement overflow
    TEMP1 = - LEFTOP;
    if((RIGHTOP[31]==TEMP1[31]))
    begin
        if(RIGHTOP[31]!=TEMP[31])
            V=1;
        else
            V=0;
        end
    else
        begin
            if((RIGHTOP[31]!=TEMP1[31])&&(TEMP[31]==TEMP1))
                V=1;
            else
                V=0;
        end
end
// //ADD
4'b0100: begin
    {C,ALUOUTPUT[31:0]} = LEFTOP[31:0] +  RIGHTOP[31:0];
    N = ALUOUTPUT[31];
    //Set the Z flag
    if(ALUOUTPUT==0)
        Z = 1;
    else
        Z = 0;
    //Set the overflow flag
    //Check for 2's complement overflow
    if((LEFTOP[31]==RIGHTOP[31]))
        if(LEFTOP[31]!=ALUOUTPUT[31])
            V=1;
        else
            V=0;
    else
        V=0;
end
// //ADC
4'b0101: begin
    {C,ALUOUTPUT[31:0]} = LEFTOP[31:0] + RIGHTOP[31:0] + CIN;
    //Set the Z flag
    if(ALUOUTPUT==0)
        Z = 1;
    else
        Z = 0;
    //Set the overflow flag
    //Check for 2's complement overflow
    if((LEFTOP[31]==RIGHTOP[31]))
        if(LEFTOP[31]!=ALUOUTPUT[31])
            V=1;
        else
            V=0;
    else
        V=0;
end
// //SBC
4'b0110: begin
    {CTEMP,ALUOUTPUT[31:0]} = LEFTOP[31:0] - RIGHTOP[31:0] - ~CIN;
    C = ~CTEMP;
    //Set the Z flag
    if(ALUOUTPUT==0)
```

```
                Z = 1;
        else
                Z = 0;
        //Set the overflow flag
        //Check for 2's complement overflow
        if((LEFTOP[31]==RIGHTOP[31]))
                if(LEFTOP[31]!=ALUOUTPUT[31])
                        V=1;
                else
                        V=0;
        else
                V=0;
end
// //RSC
4'b0111: begin
        {CTEMP,ALUOUTPUT[31:0]} =  RIGHTOP[31:0] - LEFTOP[31:0] - ~CIN;
        C = ~CTEMP;
        if(ALUOUTPUT==0)
                Z = 1;
        else
                Z = 0;
        //Set the overflow flag
        //Check for 2's complement overflow
        if((LEFTOP[31]==RIGHTOP[31]))
                if(LEFTOP[31]!=ALUOUTPUT[31])
                        V=1;
                else
                        V=0;
        else
                V=0;
end
// //TST
```

```
4'b1000: begin
    {TEMP[31:0]} = LEFTOP[31:0] & RIGHTOP[31:0];
    N = TEMP[31];

    if(TEMP==0)
        Z = 1;
    else
        Z = 0;
end
// //TEQ
4'b1001: begin
    {TEMP[31:0]} =  LEFTOP[31:0] ^ RIGHTOP[31:0];
    N =  TEMP[31];

    if(TEMP==0)
        Z = 1;
    else
        Z = 0;
end
// //CMP
4'b1010: begin
    {CTEMP,TEMP[31:0]} = LEFTOP[31:0] -  RIGHTOP[31:0];
    //Set the N flag
    C = ~CTEMP;
    N = TEMP[31];
    //Set the Z flag
    if(TEMP==0)
        Z = 1;
    else
        Z = 0;
    //Set the overflow flag
    //Check for 2's complement overflow
```

```
        TEMP1 = - RIGHTOP;
        if((LEFTOP[31]==TEMP1[31]))
        begin
                if(LEFTOP[31]!=TEMP[31])
                        V=1;
                else
                        V=0;
                end
        else
                begin
                        if((LEFTOP[31]!=TEMP1[31])&&(TEMP[31]==TEMP1))
                                V=1;
                        else
                                V=0;
                end
end
// //CMN
4'b1011: begin
        {C,TEMP[31:0]} = LEFTOP[31:0] +  RIGHTOP[31:0];
        N = TEMP[31];
        //Set the Z flag
        if(TEMP==0)
                Z = 1;
        else
                Z = 0;
        //Set the overflow flag
        //Check for 2's complement overflow
        if((LEFTOP[31]==RIGHTOP[31]))
                if(LEFTOP[31]!=TEMP[31])
                        V=1;
                else
                        V=0;
```

```
		else
			V=0;
	end
	// //ORR
	4'b1100: begin
		ALUOUTPUT[31:0] = LEFTOP[31:0] | RIGHTOP[31:0];
		// $display("Changing");
		//Set the N flag

		N = ALUOUTPUT[31];
		//Set the Z flag
		if(ALUOUTPUT==0)
			Z = 1;
		else
			Z = 0;
		C = 0;
		V = 0;

	end
	// //MOV
	4'b1101: begin
		ALUOUTPUT[31:0] = RIGHTOP[31:0];
		//Set the N flag
		N = ALUOUTPUT[31];
		//Set the Z flag
		if(ALUOUTPUT==0)
			Z = 1;
		else
			Z = 0;
	end
	// //BIC
	4'b1110: begin
```

```
                ALUOUTPUT[31:0] = LEFTOP[31:0] & ~RIGHTOP[31:0];
                //Set the N flag
                N = ALUOUTPUT[31];
                //Set the Z flag
                if(ALUOUTPUT==0)
                    Z = 1;
                else
                    Z = 0;
            end
            // //MVN
            4'b1111: begin
                ALUOUTPUT[31:0] = ~RIGHTOP[31:0];
                //Set the N flag
                N = ALUOUTPUT[31];
                //Set the Z flag
                if(ALUOUTPUT==0)
                    Z = 1;
                else
                    Z = 0;
            end
        endcase // FN
    end
endmodule

//-------------------------------------------------------------------------------------------------------------------------------------------------
module mux4x1(output reg[31:0] Y, input [1:0] S, input [31:0] I0, I1, I2, I3);
    always @ (S, I0, I1, I2, I3)
    case (S)
        2'b00: assign Y=I0[31:0];
        2'b01: assign Y=I1[31:0];
        2'b10: assign Y=I2[31:0];
        2'b11: assign Y=I3[31:0];
```

```
     endcase
endmodule

module mux4x12b(output reg[1:0] Y, input [1:0] S, input [1:0] I0, I1, I2, I3);
     always @ (S, I0, I1, I2, I3)
     case (S)
                  2'b00: assign Y=I0[1:0];
                  2'b01: assign Y=I1[1:0];
                  2'b10: assign Y=I2[1:0];
                  2'b11: assign Y=I3[1:0];
                  default: assign Y= 2'bXX;
     endcase
endmodule

module mux4x14b(output reg[3:0] Y, input [1:0] S, input [3:0] I0, I1, I2, I3);
     always @ (S, I0, I1, I2, I3)
     case (S)
                  2'b00: assign Y=I0[3:0];
                  2'b01: assign Y=I1[3:0];
                  2'b10: assign Y=I2[3:0];
                  2'b11: assign Y=I3[3:0];
                  default: assign Y= 4'bXXXX;
     endcase
endmodule
//------------------------------------------------------------------------------------------------------------------------------------------
module mux8x1(output reg[31:0] Y, input [2:0] S, input [31:0] I0, I1, I2, I3, I4,I5,I6,I7);
     always @ (S, I0, I1, I2, I3, I4,I5,I6,I7)
     case (S)
                  0: assign Y=I0[31:0];
                  1: assign Y=I1[31:0];
                  2: assign Y=I2[31:0];
                  3: assign Y=I3[31:0];
```

```
            4: assign Y=I4;
            5: assign Y=I5;
            6: assign Y=I6;
            7: assign Y=I7;
    endcase
endmodule

module mux8x12b(output reg[1:0] Y, input [2:0] S, input [1:0] I0, I1, I2, I3, I4,I5,I6,I7);
    always @ (S, I0, I1, I2, I3, I4,I5,I6,I7)
    case (S)
            0: assign Y=I0;
            1: assign Y=I1;
            2: assign Y=I2;
            3: assign Y=I3;
            4: assign Y=I4;
            5: assign Y=I5;
            6: assign Y=I6;
            7: assign Y=I7;
    endcase
endmodule

module mux8x11b(output reg Y, input [2:0] S, input I0, I1, I2, I3, I4,I5,I6,I7);
    always @ (S, I0, I1, I2, I3, I4,I5,I6,I7)
    case (S)
            0: assign Y=I0;
            1: assign Y=I1;
            2: assign Y=I2;
            3: assign Y=I3;
            4: assign Y=I4;
            5: assign Y=I5;
            6: assign Y=I6;
            7: assign Y=I7;
    endcase
endmodule

module mux8x14b(output reg[3:0] Y, input [2:0] S, input [3:0] I0, I1, I2, I3, I4,I5,I6,I7);
    always @ (S, I0, I1, I2, I3, I4,I5,I6,I7)
    case (S)
              0: assign Y=I0;
              1: assign Y=I1;
              2: assign Y=I2;
              3: assign Y=I3;
              4: assign Y=I4;
              5: assign Y=I5;
              6: assign Y=I6;
              7: assign Y=I7;
    endcase
endmodule
//-----------------------------------------------------------------------------------------------------------------------------------------
module mux2x1(output [31:0] Y, input S, input [31:0] I0, I1);
    assign Y=S? I1:I0;
endmodule

//-----------------------------------------------------------------------------------------------------------------------------------------
module mux2x11b(output Y, input S, input I0, I1);
    assign Y=S? I1:I0;
endmodule

module mux2x12b(output [1:0] Y, input S, input [1:0] I0, I1);
    assign Y=S? I1:I0;
endmodule

module mux2x14b(output [3:0] Y, input S, input [3:0] I0, I1);
    assign Y=S? I1:I0;
```

```
endmodule

//---------------------------------------------------------------------------------------------------------------------------------------
/*MAS: 00 B  //  01 H  //  10 w  // 11 undefined
1 = read  //  0 = write*/
module ramdummyreadfile (output reg [31:0]dataOut, output reg done, input enable, readWrite, input [7:0]address, input [31:0]dataIn,
input [1:0]MAS, input sign);
    reg [7:0]mem[0:511];
    initial begin
            $readmemb("data.bin", mem) ;
            done = 0;
    end
    always @ (enable, readWrite, MAS, dataIn, address, sign) begin
            done = 0;
            if (enable) begin
                    done = 0;
                    if (readWrite) begin
                            case(MAS)
                                    2'b00:  begin
                                            dataOut[7:0] = mem[address][7:0];
                                            if(sign)
                                                    dataOut[31:8] = {24{mem[address][7]}};
                                            else
                                                    dataOut[31:8] = 24'b000000000000000000000000;
                                    end
                                    2'b01:  begin
                                            dataOut[15:8] = mem[address][7:0];
                                            dataOut[7:0] = mem[address + 8'b0000001][7:0];
                                            if(sign)
                                                    dataOut[31:16] = {16{mem[address][7]}};
                                            else
                                                    dataOut[31:16] = 16'b0000000000000000;
```

```
                        end
                        2'b10:  begin // #30;
                                dataOut[31:0] = {mem[address][7:0],
                                                                mem[address + 8'b0000001][7:0],
                                                                mem[address + 8'b0000010][7:0],
                                                                mem[address + 8'b0000011][7:0]};
                        end
                        default: dataOut = dataOut;
                endcase
        end
        else begin
                if(dataIn||!dataIn)
                begin
                case(MAS)

                        2'b00:  mem[address][7:0] = dataIn[7:0];
                        2'b01:  begin
                                mem[address][7:0] = dataIn[15:8];
                                mem[address + 8'b0000001][7:0] = dataIn[7:0] ;
                        end
                        2'b10:  begin //#60;
                                mem[address + 8'b00000011][7:0] = dataIn[7:0];
                                mem[address + 8'b00000010][7:0] = dataIn[15:8];
                                mem[address + 8'b00000001][7:0] = dataIn[23:16];
                                mem[address][7:0] = dataIn[31:24];
                        end
                        default: dataOut = dataOut;
                endcase
        end
        end
        #4 done = 1;
end
```

```
        else
            dataOut = 32'bz;
    end
endmodule

//----------------------------------------------------------------------------------------------------------------------------------
module reg32(output reg [31:0] Q, input [31:0] D, input EN, CLR, CLK);
    initial    Q = 32'b00000000000000000000000000000000; // Start registers with 0
    always @ (negedge CLK, negedge CLR)
        if(!EN)
            Q <= D; // Enable Sync. Only occurs when Clk is high
        else if(!CLR) // clear
            Q <= 32'b00000000000000000000000000000000; // Clear Async
        else
            Q <= Q; // enable off. output what came out before
endmodule

//----------------------------------------------------------------------------------------------------------------------------------
module dec4x1632b(output reg [15:0] D, input[3:0] A, input EN);
    always @(A, EN)
    begin
        if (!EN)
            case(A)
                4'b0000: D = 16'b1111111111111110;
                4'b0001: D = 16'b1111111111111101;
                4'b0010: D = 16'b1111111111111011;
                4'b0011: D = 16'b1111111111110111;
                4'b0100: D = 16'b1111111111101111;
                4'b0101: D = 16'b1111111111011111;
                4'b0110: D = 16'b1111111110111111;
                4'b0111: D = 16'b1111111101111111;
                4'b1000: D = 16'b1111111011111111;
```

```
                            4'b1001: D = 16'b1111110111111111;
                            4'b1010: D = 16'b1111101111111111;
                            4'b1011: D = 16'b1111011111111111;
                            4'b1100: D = 16'b1110111111111111;
                            4'b1101: D = 16'b1101111111111111;
                            4'b1110: D = 16'b1011111111111111;
                            4'b1111: D = 16'b0111111111111111;
                            default: D = 16'b1111111111111111;
                    endcase
    end
endmodule

//----------------------------------------------------------------------------------------------------------------------------------
module reg32b(output reg [31:0] Q, input [31:0] D, input EN, CLR, CLK);
    initial    Q = 32'b00000000000000000000000000000000; // Start registers with 0
    always @ (negedge CLK, negedge CLR)
            if(!EN)
                    Q = D; // Enable Sync. Only occurs when Clk is high
            else if(!CLR) // clear
                    Q = 32'b00000000000000000000000000000000; // Clear Async
            else
                    Q <= Q; // enable off. output what came out before
endmodule

//----------------------------------------------------------------------------------------------------------------------------------
module mux8x132b(output reg [31:0] O, input [31:0] I0, I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12, I13, I14, I15, input [3:0] SEL);
    always @ (SEL, I0, I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12, I13, I14, I15) // if I change the input and enable is high then
            case(SEL)
                    4'b0000: O = I0;
                    4'b0001: O = I1;
                    4'b0010: O = I2;
                    4'b0011: O = I3;
```

```
                        4'b0100: O = I4;
                        4'b0101: O = I5;
                        4'b0110: O = I6;
                        4'b0111: O = I7;
                        4'b1000: O = I8;
                        4'b1001: O = I9;
                        4'b1010: O = I10;
                        4'b1011: O = I11;
                        4'b1100: O = I12;
                        4'b1101: O = I13;
                        4'b1110: O = I14;
                        4'b1111: O = I15;
                        default:O = O;
              endcase
endmodule

//----------------------------------------------------------------------------------------------------------------------------------
module registerFile (output [31:0] A, B, input[31:0] PC, input [3:0] REGEN, input REGCLR, input [3:0] M0SEL, input [3:0] M1SEL, input
REGCLK, RFE);
    wire [15:0] decoder2RegEnable; // 16 lines of one bit for Register Enables

    wire [31:0] reg0ToMux; // 1 line of 32 bits
    wire [31:0] reg1ToMux; // 1 line of 32 bits
    wire [31:0] reg2ToMux; // 1 line of 32 bits
    wire [31:0] reg3ToMux; // 1 line of 32 bits
    wire [31:0] reg4ToMux; // 1 line of 32 bits
    wire [31:0] reg5ToMux; // 1 line of 32 bits
    wire [31:0] reg6ToMux; // 1 line of 32 bits
    wire [31:0] reg7ToMux; // 1 line of 32 bits
    wire [31:0] reg8ToMux; // 1 line of 32 bits
    wire [31:0] reg9ToMux; // 1 line of 32 bits
    wire [31:0] reg10ToMux; // 1 line of 32 bits
    wire [31:0] reg11ToMux; // 1 line of 32 bits
    wire [31:0] reg12ToMux; // 1 line of 32 bits
    wire [31:0] reg13ToMux; // 1 line of 32 bits
    wire [31:0] reg14ToMux; // 1 line of 32 bits
    wire [31:0] reg15ToMux; // 1 line of 32 bits

    dec4x1632b      D0 (decoder2RegEnable, REGEN, RFE); // Enable selector

    reg32b R0  (reg0ToMux,  PC, decoder2RegEnable[0],  REGCLR, REGCLK);
    reg32b R1  (reg1ToMux,  PC, decoder2RegEnable[1],  REGCLR, REGCLK);
    reg32b R2  (reg2ToMux,  PC, decoder2RegEnable[2],  REGCLR, REGCLK);
    reg32b R3  (reg3ToMux,  PC, decoder2RegEnable[3],  REGCLR, REGCLK);
    reg32b R4  (reg4ToMux,  PC, decoder2RegEnable[4],  REGCLR, REGCLK);
    reg32b R5  (reg5ToMux,  PC, decoder2RegEnable[5],  REGCLR, REGCLK);
    reg32b R6  (reg6ToMux,  PC, decoder2RegEnable[6],  REGCLR, REGCLK);
    reg32b R7  (reg7ToMux,  PC, decoder2RegEnable[7],  REGCLR, REGCLK);
    reg32b R8  (reg8ToMux,  PC, decoder2RegEnable[8],  REGCLR, REGCLK);
    reg32b R9  (reg9ToMux,  PC, decoder2RegEnable[9],  REGCLR, REGCLK);
    reg32b R10 (reg10ToMux, PC, decoder2RegEnable[10], REGCLR, REGCLK);
    reg32b R11 (reg11ToMux, PC, decoder2RegEnable[11], REGCLR, REGCLK);
    reg32b R12 (reg12ToMux, PC, decoder2RegEnable[12], REGCLR, REGCLK);
    reg32b R13 (reg13ToMux, PC, decoder2RegEnable[13], REGCLR, REGCLK);
    reg32b R14 (reg14ToMux, PC, decoder2RegEnable[14], REGCLR, REGCLK);
    reg32b R15 (reg15ToMux, PC, decoder2RegEnable[15], REGCLR, REGCLK);

    mux8x132b M0 (A, reg0ToMux, reg1ToMux, reg2ToMux, reg3ToMux, reg4ToMux, reg5ToMux, reg6ToMux, reg7ToMux,
            reg8ToMux, reg9ToMux, reg10ToMux, reg11ToMux, reg12ToMux, reg13ToMux, reg14ToMux, reg15ToMux, M0SEL);

    mux8x132b M1 (B, reg0ToMux, reg1ToMux, reg2ToMux, reg3ToMux, reg4ToMux, reg5ToMux, reg6ToMux, reg7ToMux,
            reg8ToMux, reg9ToMux, reg10ToMux, reg11ToMux, reg12ToMux, reg13ToMux, reg14ToMux, reg15ToMux, M1SEL);
endmodule
```

```
//-----------------------------------------------------------------------------------------------------------------------------------
module internalshifter (input [31:0] amount, value, input [1:0] shifttype, output reg [31:0] shiftout);
    reg [63:0] temp;
    always @(amount, value, shifttype) begin
            case(shifttype)
                    0: shiftout[31:0] = value[31:0]<<amount;           //Logical Shift Left
                    1: shiftout[31:0] = value[31:0]>>amount;           //Logical Shift Right
                    2: shiftout[31:0] = $signed(value[31:0])>>>amount;        //right arithmetic
                    3: begin
                            temp = {value, value} >> amount;           //rotate right
                            shiftout[31:0] = temp[31:0];
                    end
            endcase //shifttype
    end
endmodule

//-----------------------------------------------------------------------------------------------------------------------------------
module shifter(input[31:0] inputregister, input[11:0] shifteroperand, input selector, output [31:0] out);
    wire[31:0] amounttointernal,valuetointernal;
    wire[1:0] shifttypetointernal;

    mux2x1 amountmux(amounttointernal,selector,{27'b000000000000000000000000000,shifteroperand[11:8],1'b0},
{27'b000000000000000000000000000,shifteroperand[11:7]});
    mux2x1 valuemux(valuetointernal,selector,{24'b000000000000000000000000,shifteroperand[7:0]},inputregister[31:0]);

    mux2x12b shifttypemux(shifttypetointernal,selector,2'b01,shifteroperand[6:5]);

    internalshifter intsh(amounttointernal,valuetointernal,shifttypetointernal,out);
endmodule

//-----------------------------------------------------------------------------------------------------------------------------------
module adder(input [31:0] pc, right, output reg [31:0] out);
```

```
    always @(pc, right)
            out = pc+right;
endmodule
//-------------------------------------------------------------------------------------------------------------------------------
// Control unit
// IR condition evaluator
module condEval(output reg out, input [31:0] IR, input [31:0] str);
    always @ (IR, str)
    case(IR[31:28])
            4'b0000: begin if (str[30]) out = 1;                        // Z=1
                            else out = 0; end
            4'b0001: begin if (str[30] == 0) out = 1;           // Z=0
                            else out = 0; end
            4'b0010: begin if (str[29]) out = 1;                        // C=1
                            else out = 0; end
            4'b0011: begin if (str[29]== 0) out = 1;            // C=0
                            else out = 0; end
            4'b0100: begin if (str[31]) out = 1;                        // N=1
                            else out = 0; end
            4'b0101: begin if (str[31] == 0) out = 1;           // N=0
                            else out = 0; end
            4'b0110: begin if (str[28]) out = 1;                        // V=1
                            else out = 0; end
            4'b0111: begin if (str[28]==0) out = 1;                     // V=0
                            else out = 0; end
            4'b1000: begin if (str[29] == 1 && str[30] == 0) out = 1;   // C=1 & Z=0
                            else out = 0; end
            4'b1001: begin if (str[29] == 0 || str[30] == 1) out = 1;   // C=0 or Z=1
                            else out = 0; end
            4'b1010: begin if (str[31] == str[28]) out = 1;     // N=Z
                            else out = 0; end
            4'b1011: begin if (str[31] != str[28]) out = 1;     // N!=V
```

```
                                else out = 0; end
            4'b1100: begin if (str[30] == 0 && str[31] == str[28]) out = 1;       // Z=0 & N=V
                                else out = 0; end
            4'b1101: begin if (str[30] == 1 || (str[31] != str[28])) out = 1;      //Z=1 or N!=V
                                else out = 0; end
            4'b1110: out = 1;
            4'b1111: out = 0;
            default: out = 1;
    endcase
endmodule
//-----------------------------------------------------------------------------
// mux to inverter with inputs if 1 bit
module mux4x11b(output reg Y, input [1:0] S, input I0, I1, I2, I3);
    always @ (S, I0, I1, I2, I3)
    case (S)
            2'b00: assign Y=I0;
            2'b01: assign Y=I1;
            2'b10: assign Y=I2;
            2'b11: assign Y=I3;
            default: Y=I1;
    endcase
endmodule
//-----------------------------------------------------------------------------
//inverter
module inverter(output reg out, input in, inv);
    always @ (in, inv)
    if(inv)
            out = ~in;
    else
            out = in;
endmodule
//-----------------------------------------------------------------------------
```

```
// 4bit mux selector upon conditions
module NSASel(output reg [1:0] M, input [2:0] ns, input sts);
    always @ (ns, sts)
    case(ns)
            3'b000: M = 2'b00;      //encoder
            3'b001: M = 2'b01;      //0
            3'b010: M = 2'b10;      //pipeline
            3'b011: M = 2'b11;      //incrementer
            3'b100:
                    case(sts)
                            0: M = 2'b00;   //encoder
                            1: M = 2'b10;   //pipeline
                    endcase
            3'b101:
                    case(sts)
                            0: M = 2'b11;   //incrementer
                            1: M = 2'b10;   //pipeline
                    endcase
            3'b110:
                    case(sts)
                            0: M = 2'b11;   //incrementer
                            1: M = 2'b00;   //encoder
                    endcase
            3'b111: M = 2'b01;      //0
    endcase
endmodule
//-----------------------------------------------------------------------------
// IR encoder for next state
module encoder(output reg [6:0] out, input [31:0] IR);
    always @(IR)
    case(IR[27:25])
            3'b000: begin
```

```
if (IR[4]) begin
        if (IR[7]) begin        //multiplies & extra load/stores
                if(IR[20]) begin                //load
                        if(IR[23]) begin                //add
                                if(IR[24]) begin                //offset or pre-indexed (P)
                                        if(IR[21]) out = 74;            //pre-indexed (W)
                                        else out = 95;                          //offset addresing
                                end
                                else begin                              //post-indexed
                                        if(IR[21]) out = 1;     //unpredictable
                                        else out = 76;                  //normal
                                end
                        end
                        else begin                              //sub
                                if(IR[24]) begin                //offset or pre-indexed (P)
                                        if(IR[21]) out = 75;            //pre-indexed (W)
                                        else out = 73;                          //offset addresing
                                end
                                else begin                              //post-indexed
                                        if(IR[21]) out = 1;     //unpredictable
                                        else out = 78;                  //normal
                                end
                        end
                end
                else begin                              //store
                        if(IR[23]) begin                //add
                                if(IR[24]) begin                //offset or pre-indexed (P)
                                        if(IR[21]) out = 84;            //pre-indexed (W)
                                        else out = 82;                          //offset addresing
                                end
                                else begin                              //post-indexed
                                        if(IR[21]) out = 1;     //unpredictable
                                        else out = 86;                          //normal
                                end
                        end
                        else begin                                      //sub
                                if(IR[24]) begin                        //offset or pre-indexed (P)
                                        if(IR[21]) out = 85;            //pre-indexed (W)
                                        else out = 83;                          //offset addresing
                                end
                                else begin                                      //post-indexed
                                        if(IR[21]) out = 1;     //unpredictable
                                        else out = 88;                  //normal
                                end
                        end
                end
        end
        else out = 1;
end
else begin
        if (IR[20])     begin   //data processing immediate shift
                case (IR[24:21])
                        0:  out = 5;    //and
                        1:  out = 7;    //eor
                        2:  out = 9;    //sub
                        3:  out = 11;   //rsb
                        4:  out = 13;   //add
                        5:  out = 15;   //adc
                        6:  out = 17;   //sbc
                        7:  out = 19;   //rsc
                        8:  out = 21;   //tst
                        9:  out = 23;   //teq
                        10: out = 25;   //cmp
                        11: out = 27;   //cmn
                12: out = 29;      //orr
                13: out = 31;      //mov
                14: out = 33;      //bic
                15: out = 35;      //mvn
            endcase
        end
        else begin
            if (IR[24:23] == 2'b10) out = 1 ;//;                //miscellaneous instructions
            else begin              //data processing immediate shift
                case (IR[24:21])
                    0:  out = 5;       //and
                    1:  out = 7;       //eor
                    2:  out = 9;       //sub
                    3:  out = 11;      //rsb
                    4:  out = 13;      //add
                    5:  out = 15;      //adc
                    6:  out = 17;      //sbc
                    7:  out = 19;      //rsc
                    8:  out = 21;      //tst
                    9:  out = 23;      //teq
                    10: out = 25;      //cmp
                    11: out = 27;      //cmn
                    12: out = 29;      //orr
                    13: out = 31;      //mov
                    14: out = 33;      //bic
                    15: out = 35;      //mvn
                endcase
            end
        end
    end
end  //done
3'b001: begin
```

```
if (IR[20])         begin   //data processing immediate (32-bit)
        case (IR[24:21])
                0:  out = 6;        //and
                1:  out = 8;        //eor
                2:  out = 10;       //sub
                3:  out = 12;       //rsb
                4:  out = 14;       //add
                5:  out = 16;       //adc
                6:  out = 18;       //sbc
                7:  out = 20;       //rsc
                8:  out = 22;       //tst
                9:  out = 24;       //teq
                10: out = 26;       //cmp
                11: out = 28;       //cmn
                12: out = 30;       //orr
                13: out = 32;       //mov
                14: out = 34;       //bic
                15: out = 36;       //mvn
        endcase
end
else begin
        if (IR[24:23] == 2'b10 && IR[21:20] == 2'b00) out = 1;              //undefined instruction
        else if (IR[24:23] == 2'b10 && IR[21:20] == 2'b10) out = 1;                 //move immediate to status register
        else begin                  //data processing immediate (32-bit)
                case (IR[24:21])
                        0:  out = 6;        //and
                        1:  out = 8;        //eor
                        2:  out = 10;       //sub
                        3:  out = 12;       //rsb
                        4:  out = 14;       //add
                        5:  out = 16;       //adc
                        6:  out = 18;       //sbc
```

```
                                7:  out = 20;   //rsc
                                8:  out = 22;   //tst
                                9:  out = 24;   //teq
                                10: out = 26;   //cmp
                                11: out = 28;   //cmn
                                12: out = 30;   //orr
                                13: out = 32;   //mov
                                14: out = 34;   //bic
                                15: out = 36;   //mvn
                        endcase
                end
        end
end  //done
3'b010: begin                                   //load/store immediate offset
        if(IR[20]) begin                                        //load
                if(IR[23]) begin                                        //add
                        if(IR[24]) begin                                        //offset or pre-indexed (P)
                                if(IR[21]) out = 40;                                    //pre-indexed (W)
                                else out = 96;                                                  //offset addresing
                        end
                        else begin                                                      //post-indexed
                                if(IR[21]) out = 1;                     //privilage loads/store
                                else out = 44;                                          //normal
                        end
                end
                else begin                                              //sub
                        if(IR[24]) begin                                        //offset or pre-indexed (P)
                                if(IR[21]) out = 41;                                    //pre-indexed (W)
                                else out = 37;                                                  //offset addresing
                        end
                        else begin                                                      //post-indexed
                                if(IR[21]) out = 1;                     //privilage loads/store
                                else out = 46;                  //normal
                        end
                end
        end
        else begin                              //store
                if(IR[23]) begin                //add
                        if(IR[24]) begin                //offset or pre-indexed (P)
                                if(IR[21]) out = 58;            //pre-indexed (W)
                                else out = 54;                          //offset addresing
                        end
                        else begin                              //post-indexed
                                if(IR[21]) out = 1;     //privilage loads/store
                                else out = 62;                  //normal
                        end
                end
                else begin                              //sub
                        if(IR[24]) begin                //offset or pre-indexed (P)
                                if(IR[21]) out = 59;            //pre-indexed (W)
                                else out = 55;                          //offset addresing
                        end
                        else begin                              //post-indexed
                                if(IR[21]) out = 1;     //privilage loads/store
                                else out = 64;                  //normal
                        end
                end
        end
end //done
3'b011: begin
        if(IR[4]) out = 1;      //arquitecturally undefined & media instructions
        else begin                                      //load/store register offset
                if(IR[20]) begin                //load
                        if(IR[23]) begin                //add
                if(IR[24]) begin                //offset or pre-indexed (P)
                        if(IR[21]) out = 42;            //pre-indexed (W)
                        else out = 38;                          //offset addresing
                end
                else begin                              //post-indexed
                        if(IR[21]) out = 1;     //privilage loads/store
                        else out = 48;                  //normal
                end
        end
        else begin                              //sub
                if(IR[24]) begin                //offset or pre-indexed (P)
                        if(IR[21]) out = 43;            //pre-indexed (W)
                        else out = 39;                          //offset addresing
                end
                else begin                              //post-indexed
                        if(IR[21]) out = 1;     //privilage loads/store
                        else out = 50;                  //normal
                end
        end
end
else begin                              //store
        if(IR[23]) begin                //add
                if(IR[24]) begin                //offset or pre-indexed (P)
                        if(IR[21]) out = 60;            //pre-indexed (W)
                        else out = 56;                          //offset addresing
                end
                else begin                              //post-indexed
                        if(IR[21]) out = 1;     //privilage loads/store
                        else out = 66;                  //normal
                end
        end
        else begin                              //sub
```

```
                                        if(IR[24]) begin                //offset or pre-indexed (P)
                                                if(IR[21]) out = 61;            //pre-indexed (W)
                                                else out = 57;                          //offset addresing
                                        end
                                        else begin                              //post-indexed
                                                if(IR[21]) out = 1;     //privilage loads/store
                                                else out = 68;                  //normal
                                        end
                                end
                        end
                end
        end //done
        3'b100: begin //load/store multiples
                if(IR[20]==1)//LOAD
                        out = 101;
                else //Store
                        out = 105;

        end
        3'b101: begin
                if (IR[24])     out = 93;               //branch with link
                else out = 94;                  //branch
        end //done
        3'b110: out = 1;        //does not apply
        3'b111: out = 1;        //does not apply
    endcase
endmodule
//-----------------------------------------------------------------------------
//6bit mux selector for state choosing (may increase depending on final states quantity)
module mux4x16b(output reg [6:0] Y, input [1:0] S, input [6:0] I0, I1, I2, I3);
    always @ (S, I0, I1, I2, I3)
    case (S)
```

```
            2'b00: assign Y=I0[6:0];
            2'b01: assign Y=I1[6:0];
            2'b10: assign Y=I2[6:0];
            2'b11: assign Y=I3[6:0];
            default: Y=I1;
    endcase
endmodule
//-----------------------------------------------------------------------------
//state adder
module cuAdder(output reg [6:0] out, input [6:0] cs, input [3:0] add);
    initial out = 7'b0000000;
    always @ (cs)
    out = cs + add;
endmodule
//-----------------------------------------------------------------------------
//state adder register
module IncReg(output reg [6:0] Q, input [6:0] D, input EN, CLR, CLK);
    initial Q = 6'b000000;     //Start registers with 0
    always @ (posedge CLK, negedge CLR)
            if(!EN)
                    Q = D;   //Enable Sync. Only occurs when Clk is high
            else if(!CLR)       //clear
                    Q = 6'b000000;  //Clear Async
            else
                    Q <= Q; //enable off. output what came out before
endmodule
//-----------------------------------------------------------------------------
//ROM (output may increce, depending on signals requiered, 1bit per signal)
module ROM (output reg [61:0] out, input [6:0] state, input clk);
    reg [61:0]mem[105:0];
    initial begin
```

```
        //                      61   59   58  57  56  55  54  53 |52   50  47  46    |39  38  37   33 32   28    26   22    20    17    13  12
11 10 9  8  7  6   5     3   1   0
        //                      S22  sign ENM S19 S20 S21 S18 S17|s0s1 NS  Inv pl     |clr E0  RA   S8 RB   S9S10  RC   S11S16 S2-S0
S6-S3 S12 Sel E1 E2 E3 E4 S7 S15 S13S14 MAS R/W MFA
        mem[0][61:0]  = 62'b00000000000011000000101ZZZZ0ZZZZ00ZZZZ00000000000111100000010;
        mem[1][61:0]  = 62'b00000000000011000000111ZZZZ0111100ZZZZ00000110100110100000010;
        mem[2][61:0]  = 62'b00000000000011000000110111110ZZZZ001111001000100001111000000010;
        mem[3][61:0]  = 62'b00000000000101100000111 0ZZZZ0ZZZZ00ZZZZ000000000000101100001011;
        mem[4][61:0]  = 62'b0000000000110010000001110000000000ZZZZ000000000000011100000010;
        mem[6][61:0]  = 62'b00000000001010110111001000001000000ZZZZ01001000010111101000010;
        mem[5][61:0]  = 62'b00000000001010110111001000001000001ZZZZ01001000011111101000010;
        mem[8][61:0]  = 62'b00000000001010110111001000001000000000000100100011011110100001 0;
        mem[7][61:0]  = 62'b00000000001010110111001000001000001000001001000111111101000010;
        mem[10][61:0] = 62'b00000000001010110111001000001000000000000100100101011110100001 0;
        mem[9][61:0]  = 62'b00000000001010110111001000001000001000001001001011111101000010;
        mem[12][61:0] = 62'b00000000001010110111001000001000000000000100100111011110100001 0;
        mem[11][61:0] = 62'b00000000001010110111001000001000001000001001001111111101000010;
        mem[14][61:0] = 62'b00000000001010110111001000001000000000000100101001011110100001 0;
        mem[13][61:0] = 62'b00000000001010110111001000001000001000001001010011111101000010;
        mem[16][61:0] = 62'b00000000001010110111001000001000000000000100101011011110100001 0;
        mem[15][61:0] = 62'b00000000001010110111001000001000001000001001010111111101000010;
        mem[18][61:0] = 62'b00000000001010110111001000001000000000000100101101011110100001 0;
        mem[17][61:0] = 62'b00000000001010110111001000001000001000001001011011111101000010;
        mem[20][61:0] = 62'b00000000001010110111001000001000000000000100101111011110100001 0;
        mem[19][61:0] = 62'b00000000001010110111001000001000001000001001011111111101000010;
        mem[22][61:0] = 62'b00000000001010110111001100001000000000000000110001011110100001 0;
        mem[21][61:0] = 62'b00000000001010110111001100001000001000000000110001111110100001 0;
        mem[24][61:0] = 62'b00000000001010110111001100001000000000000000110011011110100001 0;
        mem[23][61:0] = 62'b00000000001010110111001100001000001000000000110011111110100001 0;
        mem[26][61:0] = 62'b00000000001010110111001100001000000000000000110101011110100001 0;
        mem[25][61:0] = 62'b00000000001010110111001100001000001000000000110101111110100001 0;
        mem[28][61:0] = 62'b00000000001010110111001100001000000000000000110111011110100001 0;
```

```
mem[27][61:0] = 62'b00000000001010110111001100001000001000000001101111111101000010;
mem[30][61:0] = 62'b00000000001010110111001000001000000000000100111001011110100001 0;
```

```
mem[58][61:0] = 62'b00000000001010111000101000001000000000000101010010110100000010;
mem[59][61:0] = 62'b00000000001010111000101000001000000000000101001010110100000010;
mem[60][61:0] = 62'b00000000001010111000101000001000010000100000010010110100000010;
mem[61][61:0] = 62'b00000000001010111000101000001000010000100000001010110100000010;
mem[62][61:0] = 62'b00000000001011101111111110000000110000000000110110110100000010;
mem[64][61:0] = 62'b00000000001011111000001110000000110000000000110110110100000010;
mem[63][61:0] = 62'b00000000001010111000101000001000000000000101010010111100000010;
mem[65][61:0] = 62'b00000000001010111000101000001000000000000101001010111100000010;
mem[66][61:0] = 62'b00000000001011111000011110000000110000000000110110110100000010;
mem[68][61:0] = 62'b00000000001011111000101110000000110000000000110110110100000010;
mem[67][61:0] = 62'b00000000001010111000101000001000010000100000010010111100000010;
mem[69][61:0] = 62'b00000000001011111000101000001000010000100000001010111100000010;
mem[98][61:0] = 62'b000000000010101100011010ZZZZXZZZZXXZZZZXX000110111111110000010;
mem[70][61:0] = 62'b10000000001011110001111100000001000000000011011011100010001010;
mem[71][61:0] = 62'b10000000000101110001111100000001000000000011011011110010000101;
mem[72][61:0] = 62'b000000000010101000000110ZZZZXZZZZXXZZZZXX000110111111110000010;
mem[95][61:0] = 62'b00000000001010110100001100001000000000000110010010110111000010;
mem[73][61:0] = 62'b00000000001010111000111100001000000000000110001010110111000010;
mem[74][61:0] = 62'b00000000001010110100001000001000000000010110010010110111000010;
mem[75][61:0] = 62'b00000000001010111000111000001000000000010110001010110111000010;
mem[76][61:0] = 62'b00000000001011110011011100000000110000000000110110110101000010;
mem[78][61:0] = 62'b00000000001011110011111100000000110000000000110110110101000010;
mem[77][61:0] = 62'b00000000001010111000111000001000000000010110010000111111000010;
mem[79][61:0] = 62'b00000000001011111000111000001000000000010110001000111111000010;
mem[99][61:0] = 62'b000000000010101101000010ZZZZXZZZZXXZZZZXX000110111111110000010;
mem[80][61:0] = 62'b01000000000101110100001100000000000000000000110110111010010011;
mem[81][61:0] = 62'b01000000000101011011100100000000000000001111110110111010010010;
mem[82][61:0] = 62'b00000000001010110110101100001000000000000110010000110111000010;
mem[83][61:0] = 62'b00000000001010110110101100001000000000000110001000110111000010;
mem[84][61:0] = 62'b00000000001010111001001000001000000000010110010000110111000010;
mem[85][61:0] = 62'b00000000001010111001001000001000000000010110001000110111000010;
mem[86][61:0] = 62'b00000000001011110101111100000000110000000000110110110101000010;
```

```
			mem[88][61:0] = 62'b0000000000101111011001110000000001100000000110110110101000010;
			mem[87][61:0] = 62'b0000000000101011100100100000100000000000101100100001111111000010;
			mem[89][61:0] = 62'b0000000000101111100100100000100000000000101100010001111111000010;
			mem[100][61:0]= 62'b0000000000101011011010101ZZZZXZZZZXXZZZZXX000110111111110000010;
			mem[90][61:0] = 62'b0100000000101111011011110000000000100000000001101001110000100001;
			mem[91][61:0] = 62'b0100000000010111011011110000000000000000000011011011110001001 1;
			mem[92][61:0] = 62'b000000000010101000000110ZZZZXZZZZXXZZZZXX000110111111110000010;

			mem[93][61:0] = 62'b000000000000111101111010000001111001110000001101100000000000010;
			mem[94][61:0] = 62'b0000000000001011011100101111000000011110001101001010000000010;
			mem[101][61:0]= 62'b0001111111110111100101110ZZZZ0ZZZZ00ZZZZ111111101101100000001011;
			mem[102][61:0]= 62'b00011111111010100000011 1ZZZZXZZZZXXZZZZXX11111011011111 0001010;
			mem[105][61:0]= 62'b0001111111101011100111 10ZZZZ0ZZZZ00ZZZZ1100011011011 0000001000;
			mem[103][61:0]= 62'b000111111111011110011110ZZZZ0ZZZZ00ZZZZ110001101101100000 01001;
			mem[104][61:0]= 62'b000111111110101000000110ZZZZXZZZZXXZZZZXX000110110111110001000;
	end

	always @ (posedge clk)
			out = mem[state][58:0];
endmodule
//------------------------------------------------------------------------------
//control unit box (output depends on ROM output)
module ControlUnit (output reg [48:0] out, input clk, mfc, lsmDone, input [31:0] IR, statusReg);
	wire [6:0] state, stateSel0, stateSel3, addToR;
	wire [1:0] ms;
	wire invIn, invOut, condOut;
	wire [61:0] innerOut;

	condEval	condEv	(condOut,	IR,				statusReg);//Sirve
	mux4x11b	mux1b	(invIn,	innerOut[52:51],	mfc,		condOut, 1'b0, lsmDone);
	inverterinv		(invOut,	invIn,			innerOut[47]);
	NSASel		stateSel (ms,		innerOut[50:48], invOut);
```

```
    encoder                 iREnc     (stateSel0,  IR);//Sirve
    mux4x16b        mux6b   (state,    ms,                          stateSel0,         7'b0000000,  innerOut[46:40], stateSel3);
    cuAdder                 adderAlu (addToR,          state,                     4'b0001);
    IncReg          incR    (stateSel3,  addToR,                    1'b0,              innerOut[39],                clk);
    ROM                     rom                 (innerOut,   state,                    clk);

    // always @(posedge clk)
    //          out = innerOut[39:0];
    // always @ (state)
    //          $display("Next State %d",state);
    always @(innerOut)
                out = {innerOut[61:53], innerOut[39:0]};
endmodule
//------------------------------------------------------------------------------------------------------------------------------------

module LSMBlackBox(output reg [31:0] registerDataOut, memoryDataOut, effectiveAddress, output reg [3:0] sourceRegisterA,
sourceRegisterB, destinationRegister ,output reg done ,input [31:0] ir, memoryDataIn, a,b, input clk, mfc, enable);
    reg inc;
    reg [4:0] j;
    reg [4:0] cnt;
    reg [31:0] startaddress;
    reg [31:0] currAddress = 0;
    reg [12:0] i;
    always @(enable) begin
                if(enable) begin
                            done = 0;
                            cnt = 0;
                            sourceRegisterA = ir[19:16];
                            #8
                            begin
                            #4 ;//$display("SRA %d RA %d",sourceRegisterA, a);
                            for(j = 0; j<16;j=j+1) begin
    if(ir[j]==1)
        cnt = cnt+1;
end
//Calculate effective address
//01 increment after
// startaddress = Rn
// endaddress = Rn + (NumberOfSetBitsIn(registerlist) * 4) - 4
// if W == 1 then
// Rn = Rn + (NumberOfSetBitsIn(registerlist) * 4)
if(ir[24:23]==2'b01) begin
    inc = 1;
    #4 begin
        startaddress = a;
        if(ir[21]==1) begin
            destinationRegister = ir[19:16];
            registerDataOut = startaddress + (cnt*4);
            #3; //$display("Waiting to store");
        end
    end
end
//11 increment before
// startaddress = Rn + 4
// endaddress = Rn + (NumberOfSetBitsIn(registerlist) * 4)
// if W == 1 then
// Rn = Rn + (NumberOfSetBitsIn(registerlist) * 4)
else if(ir[24:23]==2'b11) begin
    // $display("IB");

    inc = 1;
    #4 begin
        startaddress = a+4;
        if(ir[21]==1) begin
```

```
            destinationRegister = ir[19:16];
            registerDataOut = startaddress + (cnt*4);
            #3;// $display("Waiting to store");
        end
    end
end
//00 decrement after
// startaddress = Rn - (NumberOfSetBitsIn(registerlist) * 4) + 4
// endaddress = Rn
// if W == 1 then
// Rn = Rn - (NumberOfSetBitsIn(registerlist) * 4)
else if(ir[24:23]==2'b00) begin
    inc = 0;
    // $display("DA");

    #4 begin
        startaddress = a-(cnt*4)+4;
        if(ir[21]==1) begin
            destinationRegister = ir[19:16];
            registerDataOut = startaddress - (cnt*4);
            #3; //$display("tato");
        end
    end
end
//10 decrement before
// startaddress = Rn - (NumberOfSetBitsIn(registerlist) * 4)
// endaddress = Rn - 4
// if W == 1 then
// Rn = Rn - (NumberOfSetBitsIn(registerlist) * 4)
else begin
    inc = 0;
                    // $display("DB");
        #4 begin
                startaddress = a-(cnt*4);
                if(ir[21]==1) begin
                        destinationRegister = ir[19:16];
                        registerDataOut = startaddress - (cnt*4);
                        #3 ;//$display("pot");
                end
        end
end
currAddress = startaddress;

// $display("Start addr %d", startaddress);
for(j=0;j<16;j=j+1) begin
        if(ir[j]==1) begin
                if(ir[20]==1) begin                                                         //load
                        // $display("IOADING");
                        effectiveAddress = currAddress;
                        // $display("Eff addr %d", effectiveAddress);
                        //Set RC
                        destinationRegister = j[4:0];
                        //Get the data to load.
                        while(!mfc) begin
                                #4 ;//$display("Waiting for memory");
                        end
                        #4 ;//$display("");
                        //Pipe the data out
                        registerDataOut = memoryDataIn;


                        //Wait a while
                        #4 ;//$display("Wait complete");
        destinationRegister = 4'bX;

    end
    else begin                                                                   //store
        //$display("Storing");
        //Set effective Address
        effectiveAddress = currAddress;
        //$display("Eaddr %d",effectiveAddress);
        //Get data
        sourceRegisterA = j[4:0];
        //Wait a bit
        #4;// $display("Done waiting");
        //Ouput data
        memoryDataOut = a;
        //$display("Memory data out %d",memoryDataOut);

        //Get the data to load.
        while(!mfc) begin
            #4;// $display("Waiting for memory");
        end
        //$display("Printing Memory:");
        // for (i = 0; i < 512; i = i +1) begin
//          $display ("Memory location %d content: %b", i, ram.mem[i]);
//      end
        #4;
        // $display("Done storing");
    end

    if(inc)
        currAddress = currAddress + 4;
    else
        currAddress = currAddress - 4;
```

```
              end

          end
          registerDataOut = 32'bX;
          memoryDataOut = 32'bX;
          effectiveAddress = 32'bX;
          sourceRegisterA = 4'bX;
          sourceRegisterB = 4'bX;
          destinationRegister = 4'bX;

          //mark done

          done = 1;

          end
      end
  end
endmodule

//----------------------------------------------------------------------------------------------------------------------------------------
module datapath;
  wire E5;
  wire [3:0] RA; // Selector of A Mux is 3 bits
  wire [3:0] RB; // Selector of B Mux is 3 bits
  wire [3:0] RC; // Register Enable Selectors (Input to Decoders 0 and 1)

  wire [1:0]MAS;
  wire MFC;

  //Flags
  wire N, COUT, V, ZERO;//ALU Flags
```

```
//Clock
reg CLK; // Register Clock Enable (All Clocks of Registers are Shared)

//General wires
wire CIN;

wire [31:0] PC, LEFTOP, B,TSROUT;

wire [31:0] aluinselmuxtoalu;
wire [31:0] martoram;
wire [1:0] muxregoutput, muxmiscout;

wire [31:0] memdata;
wire [31:0] irout, mdrout, mdrin;

wire [31:0] shifteroutput;
wire [31:0] serout;

wire SC;

wire [48:0] cuSignals;

wire [31:0]  LSMultData,LSMEaddr,LSMultRegData;
wire [3:0] LSMSourceRegA,LSMSourceRegB, LSMDestReg;
wire LSMDone, miscout, regout, SIGN;
wire [31:0] rfmuxtorf, marmuxtoram;

ControlUnit cu (cuSignals, CLK, MFC, LSMDone, irout, TSROUT);

//Register file muxes
mux4x14b ramux(RA, {cuSignals[44],cuSignals[33]}, cuSignals[37:34], irout[19:16],LSMSourceRegA,4'b0000);
```

```
    mux8x14b rbmux(RB, {cuSignals[43],cuSignals[28:27]}, cuSignals[32:29], irout[3:0], irout[15:12],
irout[19:16],LSMSourceRegB,4'b0000,4'b0000,4'b0000);
    mux4x14b rcmux(RC, cuSignals[22:21], cuSignals[26:23], irout[15:12], irout[19:16],LSMDestReg);

    //Register file
    mux2x1 rfmux(rfmuxtorf,cuSignals[42],PC,LSMultRegData);
    registerFile registerFile (LEFTOP, B, rfmuxtorf, RC, cuSignals[39], RA, RB, CLK, cuSignals[38]);

    //Input mux
    mux8x1 aluinputselectmux(aluinselmuxtoalu, cuSignals[20:18],
            B, shifteroutput, irout, serout,4,{{20{1'b0}},irout[11:0]},{{24{1'b0}},irout[11:8],irout[3:0]},mdrout);

    //Alu
    mux2x11b alucinmux(CIN, cuSignals[13], SC, TSROUT[29]);
    ALU alu1(PC, ZERO, N, COUT, V, LEFTOP, aluinselmuxtoalu, cuSignals[17:14], CIN);
    //Status register
    mux2x11b srmux(E5, cuSignals[6], 1'b1, ~irout[20]);
    reg32 statusregister(TSROUT, {N,ZERO,COUT,V,28'b0000000000000000000000000000}, E5, cuSignals[39], CLK);
    //Right side
    mux4x1 mdrmux(mdrin, {cuSignals[40],cuSignals[7]}, PC, memdata,LSMultData
            ,0);
    reg32 mdr(mdrout, mdrin, cuSignals[8], cuSignals[39], CLK);
    mux2x1 marmux(marmuxtoram, cuSignals[41], PC, LSMEaddr);
    reg32 mar(martoram, marmuxtoram, cuSignals[9], cuSignals[39], CLK);

    mux8x12b miscmux(muxmiscout, {irout[20],irout[6],irout[5]}, 2'b00 ,2'b01, 2'b10, 2'b10, 2'b10, 2'b00, 2'b00, 2'b01);
    mux2x12b regmux(muxregoutput, irout[22], 2'b10, 2'b00);
    mux4x12b masmux(MAS, cuSignals[5:4], cuSignals[3:2], muxmiscout, muxregoutput, 2'b00);

    mux8x11b miscsigmux(miscout, {irout[20],irout[6],irout[5]}, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b1, 1'b1);
    mux2x11b regsigmux(regout, irout[22], 1'b0, 1'b0);
    mux4x11b signedmux(SIGN, cuSignals[61:60], cuSignals[59], miscout, regout, 1'b0);
    //                                                                                       Enable    Read/Write    Input Address    Input Data Datasize
    ramdummyreadfile ram(memdata, MFC, cuSignals[0], cuSignals[1], martoram[7:0], mdrout, MAS, SIGN);

    reg32 ir(irout, memdata, cuSignals[10], cuSignals[39], CLK);

    shifter sh(B, irout[11:0], cuSignals[12], shifteroutput);

    reg32 ser(serout, {{18{irout[11]}},irout[11:0],2'b00}, cuSignals[11], cuSignals[39], CLK);

    LSMBlackBox lsm(LSMultRegData, LSMultData, LSMEaddr, LSMSourceRegA, LSMSourceRegB, LSMDestReg ,LSMDone ,irout, memdata,
LEFTOP,B, CLK, MFC, cuSignals[45]);
    //Vamos a probar
    parameter simtime = 5000;

    initial begin CLK = 1; end
    initial forever #2 CLK = ~CLK; // Change Clock Every Time Unit

    always @(martoram)
            if(martoram||!martoram)
                    $display("Memory Access: %b (%0d)",martoram,martoram);

    reg [12:0] i;

    initial #simtime begin
            $display("Printing Memory:");
            for (i = 0; i < 512; i = i +1) begin
                    $display ("Memory location %d content: %b", i, ram.mem[i]);
            end
    end

    initial #simtime $finish;
```

```
endmodule
```

Appendix B: Simulation Results

Test A

```
Memory Access: 00000000000000000000000000000000 (0)
Memory Access: 00000000000000000000000000000100 (4)
Memory Access: 00000000000000000000000000001000 (8)
Memory Access: 00000000000000000000000000101000 (40)
Memory Access: 00000000000000000000000000001100 (12)
Memory Access: 00000000000000000000000000101010 (42)
Memory Access: 00000000000000000000000000010000 (16)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000100000 (32)
Memory Access: 00000000000000000000000000101011 (43)
Memory Access: 00000000000000000000000000100100 (36)
Memory Access: 00000000000000000000000000101100 (44)
Printing Memory:
Memory location    0 content: 11100010
Memory location    1 content: 00000001
Memory location    2 content: 00000000
Memory location  3 content: 00000000
Memory location  4 content: 11100011
Memory location  5 content: 10000000
Memory location  6 content: 00010000
Memory location  7 content: 00101000
Memory location  8 content: 11100111
Memory location  9 content: 11010001
Memory location 10 content: 00100000
Memory location 11 content: 00000000
Memory location 12 content: 11100101
Memory location 13 content: 11010001
Memory location 14 content: 00110000
Memory location 15 content: 00000010
Memory location 16 content: 11100000
Memory location 17 content: 10000000
Memory location 18 content: 01010000
Memory location 19 content: 00000000
Memory location 20 content: 11100000
Memory location 21 content: 10000010
Memory location 22 content: 01010000
Memory location 23 content: 00000101
Memory location 24 content: 11100010
Memory location 25 content: 01010011
Memory location 26 content: 00110000
Memory location 27 content: 00000001
Memory location 28 content: 00011010
Memory location 29 content: 11111111
Memory location 30 content: 11111111
Memory location 31 content: 11111101
Memory location 32 content: 11100101
Memory location 33 content: 11000001
Memory location 34 content: 01010000
Memory location 35 content: 00000011
Memory location 36 content: 11101010
Memory location 37 content: 00000000
Memory location 38 content: 00000000
Memory location 39 content: 00000001
Memory location 40 content: 00001011
Memory location 41 content: 00000101
Memory location   42 content: 00000111
Memory location   43 content: 01001101
Memory location   44 content: 11101010
Memory location   45 content: 11111111
Memory location   46 content: 11111111
Memory location   47 content: 11111111
```

Test B

```
Memory Access: 00000000000000000000000000000000 (0)
Memory Access: 00000000000000000000000000000100 (4)
Memory Access: 00000000000000000000000000001000 (8)
Memory Access: 00000000000000000000000000001100 (12)
Memory Access: 00000000000000000000000000010000 (16)
Memory Access: 00000000000000000000000010011000 (152)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000010011100 (156)
Memory Access: 00000000000000000000000000100000 (32)
Memory Access: 00000000000000000000000000100100 (36)
Memory Access: 00000000000000000000000010100000 (160)
Memory Access: 00000000000000000000000000101000 (40)
Memory Access: 00000000000000000000000000101100 (44)
Memory Access: 00000000000000000000000010100100 (164)
Memory Access: 00000000000000000000000000110000 (48)
Memory Access: 00000000000000000000000000110100 (52)
Memory Access: 00000000000000000000000010101000 (168)
Memory Access: 00000000000000000000000000111000 (56)
Memory Access: 00000000000000000000000000111100 (60)
Memory Access: 00000000000000000000000010101100 (172)
Memory Access: 00000000000000000000000001000000 (64)
Memory Access: 00000000000000000000000001000100 (68)
Memory Access: 00000000000000000000000010110000 (176)
Memory Access: 00000000000000000000000001001000 (72)
Memory Access: 00000000000000000000000001001100 (76)
Memory Access: 00000000000000000000000001010000 (80)
Memory Access: 00000000000000000000000001010100 (84)
Memory Access: 00000000000000000000000010110100 (180)
Memory Access: 00000000000000000000000001011000 (88)
Memory Access: 00000000000000000000000010010100 (148)
Memory Access: 00000000000000000000000001011100 (92)
Memory Access: 00000000000000000000000010010000 (144)
Memory Access: 00000000000000000000000001100000 (96)
Memory Access: 00000000000000000000000001100100 (100)
Memory Access: 00000000000000000000000001101100 (108)
Memory Access: 00000000000000000000000010111000 (184)
Memory Access: 00000000000000000000000001110000 (112)
Memory Access: 00000000000000000000000001110100 (116)
Memory Access: 00000000000000000000000001111100 (124)
Memory Access: 00000000000000000000000010111100 (188)
Memory Access: 00000000000000000000000010000000 (128)
Memory Access: 00000000000000000000000010000100 (132)
Memory Access: 00000000000000000000000010111100 (188)
Memory Access: 00000000000000000000000010001000 (136)
Memory Access: 00000000000000000000000011000000 (192)
Memory Access: 00000000000000000000000010001100 (140)
Printing Memory:
Memory location   0 content: 11100010
Memory location   1 content: 00000001
Memory location   2 content: 00000000
Memory location   3 content: 00000000
Memory location   4 content: 11100011
Memory location   5 content: 10000000
Memory location   6 content: 00010000
Memory location   7 content: 00100010
Memory location   8 content: 11100011
Memory location   9 content: 10000000
Memory location  10 content: 01000000
Memory location  11 content: 10011000
Memory location  12 content: 11100001
Memory location  13 content: 10100000
Memory location  14 content: 10100000
Memory location  15 content: 00000100
Memory location  16 content: 11100100
Memory location  17 content: 10011010
Memory location  18 content: 00110000
Memory location  19 content: 00000100
Memory location  20 content: 11100000
Memory location   21 content: 10000001
Memory location   22 content: 00100000
Memory location   23 content: 00000011
Memory location   24 content: 11100001
Memory location   25 content: 10100000
Memory location   26 content: 01010000
Memory location   27 content: 00000010
Memory location   28 content: 11100100
Memory location   29 content: 10001010
Memory location   30 content: 01010000
Memory location   31 content: 00000100
Memory location   32 content: 11100000
Memory location   33 content: 01000001
Memory location   34 content: 01010000
Memory location   35 content: 00000010
Memory location   36 content: 11100100
Memory location   37 content: 10001010
Memory location   38 content: 01010000
Memory location   39 content: 00000100
Memory location   40 content: 11100000
Memory location   41 content: 01100001
Memory location   42 content: 01010000
Memory location   43 content: 00000010
Memory location   44 content: 11100100
Memory location   45 content: 10001010
Memory location   46 content: 01010000
Memory location   47 content: 00000100
Memory location   48 content: 11100000
Memory location   49 content: 10000000
Memory location   50 content: 01010001
Memory location   51 content: 11000011
Memory location   52 content: 11100100
Memory location   53 content: 10001010
Memory location   54 content: 01010000
Memory location   55 content: 00000100
Memory location   56 content: 11100000
Memory location   57 content: 10000000
Memory location   58 content: 01010001
Memory location   59 content: 10000011
Memory location   60 content: 11100100
Memory location   61 content: 10001010
Memory location   62 content: 01010000
Memory location   63 content: 00000100
Memory location   64 content: 11100000
Memory location   65 content: 10000000
Memory location   66 content: 01010101
Memory location   67 content: 01100011
Memory location   68 content: 11100100
Memory location   69 content: 10001010
Memory location   70 content: 01010000
Memory location   71 content: 00000100
Memory location   72 content: 11100001
Memory location   73 content: 01010010
Memory location   74 content: 00000000
Memory location   75 content: 00000001
Memory location   76 content: 11011011
Memory location   77 content: 00000000
Memory location   78 content: 00000000
Memory location   79 content: 00000001
Memory location   80 content: 11100010
Memory location   81 content: 00000001
Memory location   82 content: 11100000
Memory location   83 content: 00000000
Memory location   84 content: 11100100
Memory location   85 content: 10001010
Memory location   86 content: 11100000
Memory location   87 content: 00000100
Memory location   88 content: 11100101
Memory location   89 content: 00110100
Memory location   90 content: 11000000
Memory location   91 content: 00000100
Memory location   92 content: 11100101
Memory location   93 content: 00110100
Memory location   94 content: 10110000
Memory location   95 content: 00000100
Memory location   96 content: 11100001
Memory location   97 content: 01111100
Memory location   98 content: 00000000
Memory location   99 content: 00001011
Memory location  100 content: 01101010
Memory location  101 content: 00000000
Memory location  102 content: 00000000
Memory location  103 content: 00000001
Memory location  104 content: 11100010
Memory location  105 content: 00000001
Memory location  106 content: 01010000
Memory location  107 content: 00000000
Memory location  108 content: 11100100
Memory location  109 content: 10001010
Memory location  110 content: 01010000
Memory location  111 content: 00000100
Memory location  112 content: 11100000
Memory location  113 content: 10000001
Memory location  114 content: 01010000
Memory location  115 content: 00001011
Memory location  116 content: 01101010
Memory location  117 content: 00000000
Memory location  118 content: 00000000
Memory location  119 content: 00000001
Memory location  120 content: 11100010
Memory location  121 content: 00000001
Memory location  122 content: 01010000
Memory location  123 content: 00000000
Memory location  124 content: 11100100
Memory location  125 content: 10001010
Memory location  126 content: 01010000
Memory location  127 content: 00000100
Memory location  128 content: 11100011
Memory location  129 content: 10000000
Memory location  130 content: 00010000
Memory location  131 content: 00000100
Memory location  132 content: 11100111
Memory location  133 content: 00111010
Memory location  134 content: 11000000
Memory location  135 content: 00000001
Memory location  136 content: 11100111
Memory location  137 content: 10101010
Memory location  138 content: 11000000
Memory location  139 content: 00000001
Memory location  140 content: 11101010
Memory location  141 content: 11111111
Memory location  142 content: 11111111
Memory location  143 content: 11111111
Memory location  144 content: 10100000
Memory location  145 content: 00000000
Memory location  146 content: 00000000
Memory location  147 content: 00000000
Memory location  148 content: 11000000
Memory location  149 content: 00000000
Memory location  150 content: 00000000
Memory location  151 content: 00000000
Memory location  152 content: 11111111
Memory location  153 content: 11111111
Memory location  154 content: 11111111
Memory location  155 content: 11110101
Memory location  156 content: 00000000
Memory location  157 content: 00000000
Memory location  158 content: 00000000
Memory location  159 content: 00010111
Memory location  160 content: 00000000
Memory location  161 content: 00000000
Memory location  162 content: 00000000
Memory location  163 content: 00001011
Memory location  164 content: 11111111
Memory location  165 content: 11111111
Memory location  166 content: 11111111
Memory location  167 content: 11110101
Memory location  168 content: 11111111
Memory location  169 content: 11111111
Memory location  170 content: 11111111
Memory location  171 content: 11111110
Memory location  172 content: 11111111
Memory location  173 content: 11111111
Memory location  174 content: 11111111
Memory location  175 content: 10101000
Memory location  176 content: 11111101
Memory location  177 content: 01111111
Memory location  178 content: 11111111
Memory location  179 content: 11111111
Memory location  180 content: 00000000
Memory location  181 content: 00000000
Memory location  182 content: 00000000
Memory location  183 content: 01010000
Memory location  184 content: 11111101
Memory location  185 content: 01111111
Memory location  186 content: 11111111
Memory location  187 content: 11111111
Memory location  188 content: 10100000
Memory location  189 content: 00000000
Memory location  190 content: 00000000
Memory location  191 content: 00100010
Memory location  192 content: 10100000
Memory location  193 content: 00000000
Memory location  194 content: 00000000
Memory location  195 content: 00100010
```

Test C

```
Memory Access: 00000000000000000000000000000000 (0)
Memory Access: 00000000000000000000000000000100 (4)
Memory Access: 00000000000000000000000000001000 (8)
Memory Access: 00000000000000000000000000001100 (12)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000100000 (32)
Memory Access: 00000000000000000000000000100100 (36)
Memory Access: 00000000000000000000000000010000 (16)
Memory Access: 00000000000000000000000000101000 (40)
Memory Access: 00000000000000000000000000101100 (44)
Memory Access: 00000000000000000000000000110000 (48)
Memory Access: 00000000000000000000000000110100 (52)
Memory Access: 00000000000000000000000000010100 (20)
Memory Access: 00000000000000000000000000011000 (24)
Memory Access: 00000000000000000000000000011100 (28)
Memory Access: 00000000000000000000000000100010 (34)
Memory Access: 00000000000000000000000000100000 (32)
Memory Access: 00000000000000000000000000111000 (56)
Memory Access: 00000000000000000000000000100100 (36)
Printing Memory:
Memory location    0 content: 11100010
Memory location    1 content: 00000001
Memory location    2 content: 00000000
Memory location    3 content: 00000000
Memory location    4 content: 11100011
Memory location    5 content: 10000000
Memory location    6 content: 10100000
Memory location    7 content: 00011000
Memory location    8 content: 11100011
Memory location    9 content: 10010000
Memory location   10 content: 01000000
Memory location   11 content: 00101000
Memory location   12 content: 11101000
Memory location   13 content: 10111010
Memory location   14 content: 00100000
Memory location   15 content: 10100100
Memory location   16 content: 11101000
Memory location   17 content: 10101010
Memory location   18 content: 00100000
Memory location   19 content: 10100100
Memory location   20 content: 00000100
Memory location   21 content: 10001010
Memory location   22 content: 01000000
Memory location   23 content: 00000100
Memory location   24 content: 11100011
Memory location   25 content: 10000000
Memory location   26 content: 00010000
Memory location   27 content: 00000100
Memory location   28 content: 11100001
Memory location   29 content: 01010100
Memory location   30 content: 11000000
Memory location   31 content: 11110110
Memory location   32 content: 11100100
Memory location   33 content: 10001010
Memory location   34 content: 11000000
Memory location   35 content: 00000100
Memory location   36 content: 11101010
Memory location   37 content: 11111111
Memory location   38 content: 11111111
Memory location   39 content: 11111111
Memory location   40 content: 11100011
Memory location   41 content: 10000000
Memory location   42 content: 00010000
Memory location   43 content: 00000100
Memory location   44 content: 11100001
Memory location   45 content: 01010100
Memory location   46 content: 11000000
Memory location   47 content: 11110110
Memory location   48 content: 11100100
Memory location   49 content: 10001010
Memory location   50 content: 11000000
Memory location   51 content: 00000100
Memory location   52 content: 11101010
Memory location   53 content: 11111111
Memory location   54 content: 11111111
Memory location   55 content: 11111111
Memory location   56 content: 11111111
Memory location   57 content: 11111111
Memory location   58 content: 11000000
Memory location   59 content: 00000100
```