 **Northwestern Polytechnic University**

**EE488 - Computer Architecture**

**Homework Assignment #6**

**Due day: 12/5/2021**

**Instruction:**

1. **Push the answer sheet to GitHub in word file**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. **(For CS Students)** Verify "Carry Lookahead" algorithm for 16-bits adder in any of computer languages
5. **(For EE Students)** Write a Verilog module to design 8-bits ALU based on the following opcodes. In the summation operation, the submodule instantiated by "generate" block in the top module should implement "Carry Lookahead"

| Opcode | Operations |
| --- | --- |
| 0000 | Out = A + B |
| 0001 | Out = A - B |
| 0010 | Out = A \* B |
| 0011 | Out = A / B |
| 0100 | Out = A << 1 |
| 0101 | Out = A >> 1 |
| 0110 | Out = A rotated left by 1 |
| 0111 | Out = A rotated right by 1 |
| 1000 | Out = A and B |
| 1001 | Out = A or B |
| 1010 | Out = A xor B |
| 1011 | Out = A nor B |
| 1100 | Out = A nand B |
| 1101 | Out = A xnor B |
| 1110 | Out = 1 if A>B else 0 |
| 1111 | Out = 1 if A=B else 0 |

module alu(

input [7:0] A,B,

input [3:0] Sel,

input cin,

output [7:0] Out,

output s,

output CarryOut

);

wire [3:0]G,P,C;

CLA uut (.a(A),.b(B),.ci(cin),.co(CarryOut),.s(s));

reg [7:0] Result;

wire [8:0] tmp;

assign Out = Result;

assign tmp = {1'b0,A} + {1'b0,B}+{1'b0,cin};

assign CarryOut = tmp[8];

always @(\*)

begin

case(Sel)

4'b0000:

Result = A + B + cin ;

4'b0001:

Result = A - B ;

4'b0010:

Result = A \* B;

4'b0011:

Result = A/B;

4'b0100:

Result = A<<1;

4'b0101:

Result = A>>1;

4'b0110:

Result = {A[6:0],A[7]};

4'b0111:

Result = {A[0],A[7:1]};

4'b1000:

Result = A & B;

4'b1001:

Result = A | B;

4'b1010:

Result = A ^ B;

4'b1011:

Result = ~(A | B);

4'b1100:

Result = ~(A & B);

4'b1101:

Result = ~(A ^ B);

4'b1110:

Result = (A>B)?8'd1:8'd0 ;

4'b1111:

Result = (A==B)?8'd1:8'd0 ;

default: Result = A + B ;

endcase

end

endmodule

module CLA(a,b,ci,co,s);

input [7:0]a,b;

output s;

input ci;

output co;

wire [3:0]G,P,C;

assign G = a&b;

assign P = a^b;

assign co=G[3]+ (P[3]&G[2]) + (P[3]&P[2]&G[1]) + (P[3]&P[2]&P[1]&G[0]) +(P[3]&P[2]&P[1]&P[0]&ci);

assign C[3]=G[2] + (P[2]&G[1]) + (P[2]&P[1]&G[0]) + (P[2]&P[1]&P[0]&ci);

assign C[2]=G[1] + (P[1]&G[0]) + (P[1]&P[0]&ci);

assign C[1]=G[0] + (P[0]&ci);

assign C[0]=ci;

assign s = {co,P^C};

endmodule

1. **(For CS Students)** Verify 4-bits Booth’s algorithm and one of multiplication algorithms from 3 versions shown in the handout of *Lec06-alu.pdf* by any of computer languages
2. **(For EE Students)** Write two Verilog modules to design 4-bits multiplier which implements Booth’s algorithm and one of multiplication algorithms from 3 versions shown in the handout of *Lec06-alu.pdf,* respectively

**Booth Multiplier:**

module BoothMul(clk,rst,start,X,Y,valid,Z);

input clk;

input rst;

input start;

input signed [3:0]X,Y;

output signed [7:0]Z;

output valid;

reg signed [7:0] Z,next\_Z,Z\_temp;

reg next\_state, pres\_state;

reg [1:0] temp,next\_temp;

reg [1:0] count,next\_count;

reg valid, next\_valid;

parameter IDLE = 1'b0;

parameter START = 1'b1;

always @ (posedge clk or negedge rst)

begin

if(!rst)

begin

Z <= 8'd0;

valid <= 1'b0;

pres\_state <= 1'b0;

temp <= 2'd0;

count <= 2'd0;

end

else

begin

Z <= next\_Z;

valid <= next\_valid;

pres\_state <= next\_state;

temp <= next\_temp;

count <= next\_count;

end

end

always @ (\*)

begin

case(pres\_state)

IDLE:

begin

next\_count = 2'b0;

next\_valid = 1'b0;

if(start)

begin

next\_state = START;

next\_temp = {X[0],1'b0};

next\_Z = {4'd0,X};

end

else

begin

next\_state = pres\_state;

next\_temp = 2'd0;

next\_Z = 8'd0;

end

end

START:

begin

case(temp)

2'b10: Z\_temp = {Z[7:4]-Y,Z[3:0]};

2'b01: Z\_temp = {Z[7:4]+Y,Z[3:0]};

default: Z\_temp = {Z[7:4],Z[3:0]};

endcase

next\_temp = {X[count+1],X[count]};

next\_count = count + 1'b1;

next\_Z = Z\_temp >>> 1;

next\_valid = (&count) ? 1'b1 : 1'b0;

next\_state = (&count) ? IDLE : pres\_state;

end

endcase

end

endmodule

**Unsigned Multiplication:**

`timescale 1ns / 1ps

module array\_multiplier(

input [3:0] a,

input [3:0] b,

output reg [15:0] m

);

integer i;

integer j;

integer index;

reg temp, temp\_o, c\_o;

always @(\*) begin

m = 0;

for(i=0;i<=7;i=i+1)

begin

c\_o = 0;

for(j=0;j<=7;j=j+1)

begin

temp = a[i]&b[j];

index = (i+j);

{c\_o, temp\_o} = m[index]+temp+c\_o;

m[index] = temp\_o;

end

index = i+j;

m[index] = c\_o;

end

end

endmodule

1. **(For CS Students)** Verify any two of division algorithms in 4-bits from 3 versions shown in the handout of *Lec07-division.pdf* by any of computer languages
2. **(For EE Students)** Write two Verilog modules to design 4-bits divisor which implements any two of division algorithms from 3 versions shown in the handout of *Lec07-division.pdf,* respectively

**Restoration:**

`timescale 1ns / 1ps

module divres (Q,M,Quo,Rem);

input [7:0] Q;

input [7:0] M;

output [7:0] Quo;

output [7:0] Rem;

reg [7:0] Quo =0;

reg [7:0] Rem =0;

reg [7:0] a1,b1;

reg [7:0] p1;

integer i;

always@ (Q or M)

begin

a1 = Q;

b1 = M;

p1 = 0;

if(a1[7]==1)

a1 = 0-a1;

if(b1[7]==1)

b1 = 0-b1;

if((b1[7]==1)&& (a1[7]==1)) begin

b1 = 0-b1;

a1 = 0-a1;

end

for(i=0;i<8;i=i+1) begin

p1 = {p1[6:0],a1[7]};

a1[7:1] = a1[6:0];

p1=p1-b1;

if(p1[7]==1) begin

a1[0] = 0;

p1 = p1+b1; end

else

a1[0] = 1;

end

if((Q[7]==1)&&(M[7]==0))

begin

Quo = 0-a1;

Rem = 0-p1;

end

else if((Q[7]==0)&&(M[7]==1))

begin

Quo = 0-a1;

Rem = p1;

end

else if ((Q[7]==1)&&(M[7]==1))

begin

Quo = a1;

Rem = 0-p1;

end

else

begin

Quo = a1;

Rem = p1;

end

end

endmodule

**Non-Restoration:**

module Divide(

input [3:0] A, // Dividend

input [3:0] B, // Divisor

input clk,

input reset,

output reg done,

output wire [7:0] Res,

output reg [7:0] Q // Quotient

);

reg [15:0] divisor\_copy;

reg [15:0] rem;

integer i;

assign Res = rem[7:0];

always @(posedge clk or posedge reset) begin

if(reset) begin

divisor\_copy = {B[3:0], 4'b0};

rem = {4'b0, A[3:0]};

done = 0;

Q = 8'b0;

i = 0;

end

else begin

if(done==0) begin

rem = rem - divisor\_copy;

if(rem[15]==0) begin

Q = Q<<1;

Q[0] = 1;

end

else begin

Q = Q<<1;

Q[0] = 0;

rem = rem + divisor\_copy;

end

divisor\_copy = divisor\_copy>>1;

i = i+1;

if(i==9)

done = 1;

end

end

end

endmodule