 **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] ALU\_Sel,

input cin,

output [7:0] ALU\_Out,

output s,

output CarryOut // Carry Out Flag

);

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

CLA uut (

.a(A),

.b(B),

.ci(cin),

.co(CarryOut),

.s(s)

);

reg [7:0] ALU\_Result;

wire [8:0] tmp;

assign ALU\_Out = ALU\_Result;

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

assign CarryOut = tmp[8]; // Carryout flag

always @(\*)

begin

case(ALU\_Sel)

4'b0000: // Addition

ALU\_Result = A + B + cin ;

4'b0001: // Subtraction

ALU\_Result = A - B ;

4'b0010: // Multiplication

ALU\_Result = A \* B;

4'b0011: // Division

ALU\_Result = A/B;

4'b0100: // Logical shift left

ALU\_Result = A<<1;

4'b0101: // Logical shift right

ALU\_Result = A>>1;

4'b0110: // Rotate left

ALU\_Result = {A[6:0],A[7]};

4'b0111: // Rotate right

ALU\_Result = {A[0],A[7:1]};

4'b1000: // Logical and

ALU\_Result = A & B;

4'b1001: // Logical or

ALU\_Result = A | B;

4'b1010: // Logical xor

ALU\_Result = A ^ B;

4'b1011: // Logical nor

ALU\_Result = ~(A | B);

4'b1100: // Logical nand

ALU\_Result = ~(A & B);

4'b1101: // Logical xnor

ALU\_Result = ~(A ^ B);

4'b1110: // Greater comparison

ALU\_Result = (A>B)?8'd1:8'd0 ;

4'b1111: // Equal comparison

ALU\_Result = (A==B)?8'd1:8'd0 ;

default: ALU\_Result = A + B ;

endcase

end

endmodule

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

input [3:0]a,b;

output [4:0]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

**// BoothsAlgo**

`timescale 1ns / 1ps

module booth\_multiplier(

input signed[3:0] X,

input signed[3:0] Y,

input clk,

input load,

output reg signed[11:0] Z

);

reg [1:0] temp;

integer i;

reg E1;

reg [11:0] Y1;

always @ (posedge clk) begin

if(load==0)

begin

Z = 12'd0;

E1 = 1'd0;

Y1 = - Y;

Z[11:0]=X;

i <= 0;

end

else

begin

if(i<6)

begin

temp = {X[i], E1};

case (temp)

2'd2 : Z [10 : 0] = {Z [11 : 6] + Y1,Z[5:1]};

2'd1 : Z [10 : 0] = {Z [11 : 6] + Y,Z[5:1]};

2'd0 : Z [10 : 0] = {Z[11:1]};

2'd3 : Z [10 : 0] = {Z[11:1]};

default : begin end

endcase

Z[11] = Z[10];

E1 = X[i];

i <= i+1;

end

end

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

**// NonRestoration**

module Divide(

input [3:0] A, // Dividend

input [3:0] B, // Divisor

input clk, // clock

input reset, // Reset Flag

output reg done,

output wire [7:0] Res,

output reg [7:0] Q // Quotient

);

reg [15:0] divisor\_copy; // Copy

reg [15:0] rem;

integer i; // counter

assign Res = rem[7:0];

// always block to compute th result for positive edge clock

always @(posedge clk or posedge reset) begin

// Initialise the value

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; // increment the counter

if(i==9)

done = 1;

end

end

end

endmodule

**//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