Lab #8 ALU

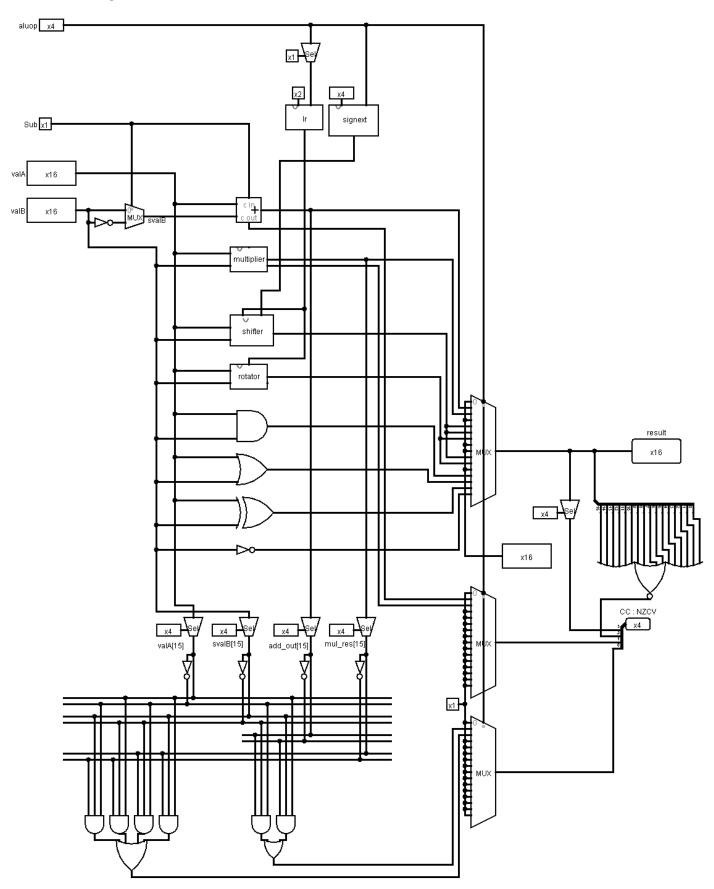
Class: 00

201602004 박태현

# l. 실습 목적

## ALU 구현

II. Design Procedure



```
`define ADD 4'b0001
`define MUL 4'b0010
//`define SUB 4'b0001
`define ASR 4'b0100
`define LSR 4'b0101
`define RR 4'b0110
`define SHL 4'b1001
`define RL 4'b1010
`define AND 4'b1100
`define OR 4'b1101
`define XOR 4'b1110
define NOT 4'b1111
```

ALU OP를 사전에 정의해주었습니다

```
module alu (valA, valB, aluop, sub, cc, result);
   input [15:0] valA;
   input [15:0] valB;
   input [ 3:0] aluop;
   input sub;
   output [ 3:0] cc;
   output [15:0] result;
   wire [15:0]
      and16b, or16b, xor16b, not16b, rotate out,
      shift out, add out, svalB, result;
   wire lr, signext, add co, mul ov, shift ov;
   wire [31:0] mul res;
   wire [3:0] cc;
   wire N,Z,C,V;
```

입력값과 중간 와이어 그리고 출력값을 정해주었습니다

```
assign and16b = valA & valB;
assign or16b = valA | valB;
assign xor16b = valA ^ valB;
assign not16b = ~valB;
논리 게이트를 계산합니다
 assign lr = (aluop[3:2] == 2'b10);
 assign signext = (aluop == `ASR);
 shifter shifter0 (valB, lr, signext, valA, shift_out);
 rotator rotator0 (valB, lr, valA, rotate_out);
시프팅을 계산합니다
 MUL16x16 mul0(valA, valB, mul res, mul ov);
 assign svalB = sub ? ~valB : valB;
 kogge stone myAdder (valA, svalB, sub, add co, add out);
곱하기와 더하기를 계산합니다
 assign result =
 (aluop == `ADD) ? add_out :
 (aluop == `AND) ? and 16b :
 (aluop == OR) ? or16b :
 (aluop = XOR) ? xor16b :
 (aluop == NOT) ? not16b :
 (aluop == ASR) ? shift_out :
 (aluop == `LSR) ? shift out :
 (aluop == `SHL) ? shift out :
 (aluop == `RL) ? rotate_out :
 (aluop == `RR) ? rotate out :
 (aluop = MUL)? mul res[15:0]: 16'bx;
계산 결과를 연산자에 맞게 MUX 해줍니다
 assign N = result[15];
 assign Z = ~\result;
음수와 0 플래그를 계산합니다
```

```
assign C =
// Add
(aluop==`ADD) ? add co :
// Multiply
(aluop==`MUL) ? mul res[16] :
// default
1'b0;
더하기와 곱하기의 carry out을 계산합니다
 assign V =
 // Add
 (aluop == ADD)?
 (valA[15] \& svalB[15] \& \sim add_out[15]) | // -a + -b = +c : 1 1 0
 (\text{-valA}[15] \& \text{-svalB}[15] \& \text{add}[\text{out}[15]) : // +a + +b = -c : 0 0 1
 (aluop == MUL)?
 ( valA[15] & valB[15] & mul res[15]) |
                                                     // -a * -b = -c : 1 1 1
 (~valA[15] & ~valB[15] & mul_res[15]) |
( valA[15] & avalB[15] a
                                                     // +a * +b = -c : 0 0 1
 ( valA[15] & ~valB[15] & ~mul_res[15]) | // -a * +b = +c : 1 0 0 (~valA[15] & valB[15] & ~mul_res[15]) : // +a * -b = +c : 0 1 0
 // default
 1'b0;
더하기 곱하기의 오버플로를 계산합니다
   - Testbench
```

```
module alu_tb;

reg [15:0] valA,valB;
reg [3:0] aluop;
wire [15:0] res;
wire [3:0] cc;
reg sub;

alu alu0(valA,valB,aluop,sub,cc,res);
```

ALU를 만들고 연결해줍니다

# initial begin

```
// ov
valA = 16'b0111 1111 1111 1111;
valB = 16'b0000 0000 0000 0001;
aluop = `ADD;
sub = 1'b0;
#10;
// zero over carry
valA = 16'b1000 0000 0000 0000;
valB = 16'b1000_0000_0000_0000;
aluop = `ADD;
sub = 1'b0;
#10;
valA = 16'b0000 0000 0000 1000;
                                          valA = 16'b1001_1101_0111_1000;
valB = 16'b0000 0000 0000 1000;
                                          valB = 16'b0000_0000_0000_1000;
aluop = `ADD;
                                          aluop = `LSR;
sub = 1'b0;
                                          #10;
#10;
                                          valA = 16'b1001_1101_0111_1000;
valA = 16'b0000 0000 0111 1000;
                                          valB = 16'b0000_0000_0000_1000;
valB = 16'b0000 0000 0000 1100;
                                          aluop = `ASR;
aluop = `ADD;
                                          #10;
                                          valA = 16'b1001 1101 0111 1000;
sub = 1'b1;
                                          valB = 16'b0000_0000_0000_1000;
#10;
                                          aluop = `SHL;
valA = 16'b0001_1101_0111_1000;
                                          #10;
valB = 16'b0111 1111 0100 1100;
                                          valA = 16'b1001_1101_0111_1000;
aluop = `AND;
                                          valB = 16'b0000 0000 0000 0100;
#10;
                                          aluop = `RL;
                                          #10;
valA = 16'b0001_1101_0111_1000;
                                          valA = 16'b1001 1101 0111 1000;
valB = 16'b0111 1111 0100 1100;
                                          valB = 16'b0000_0000_0000_0100;
aluop = ^{\circ}OR;
                                          aluop = `RR;
#10;
                                          #10;
valA = 16'b0001_1101_0111_1000;
                                          valA = 16'b0000 0000 0000 1100;
valB = 16'b0111 1111 0100 1100;
                                          valB = 16'b0000 0000 0000 0100;
aluop = XOR;
                                          aluop = `MUL;
#10;
                                          #10;
```

각 계산을 한번씩 해보았습니다

# IV. Evaluation

0111111111111111	1 1000000000	000000	00000000000	01000	00000000011	11000	00011101011	11000						
0000000000000000	1 1000000000	000000	00000000000	01000	000000000000	01100	01111111010	01100						
0001							1100		1101		1110		1111	
100000000000000000000000000000000000000	0000000000	000000	00000000000	10000	00000000011	01100	00011101010	01000	0111111101	11100	01100010001	10100	10000000101	10011
1001	0111		0000		0010		0000						1000	
		<u> </u>												

10011101011	11000									00000000000	01100
00000000000	01000					00000000000	00100				
0101		0100		1001		1010		0110		0010	
00000000100	11101	11111111100	11101	01111000000	00000	11010111100	01001	10001001110	10111	000000000001	10000
0000		1000		0000		1000				0000	

# 각 연산을 한번씩 해보았습니다

120
12
108
0010

# 빼기

12	
4	
48	
0000	

12 \* 4 = 48

# AND

00011101011110	00
01111111010011	00
00011101010010	00
0000	
1100	

# OR

00011101011111000	
01111111101001100	
0111111101111100	
0000	
1101	

## XOR

_	
5	0001110101111000
o	0111111101001100
D	0110001000110100
	0000
-	(1110

#### NOT

0111111101001100	
1111	==
11111	
1000000010110011	
(1000	
,1000	

## LSR

1001110101	11000
1001110101.	11000
000000000000	01000
000000000100	11101
0000	
,0000	

## ASR

10011101011111000
0000000000001000
1111111110011101
1000

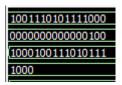
## SHL

100111010	1111000
000000000	0001000
011110000	0000000
0000	0000000
0000	

## RL

100111010	1111000
000000000	0000100
110101111	0001001
1000	
1000	

#### RR



## V. Discussion

여러 모듈을 이어서 하나를 만드는 것이 생각보다 헷갈렸습니다 만들고보니 MUX 입력이 남는데 어떻게 하면 좋을지 잘 모르겠습니다