Lab #8 ALU

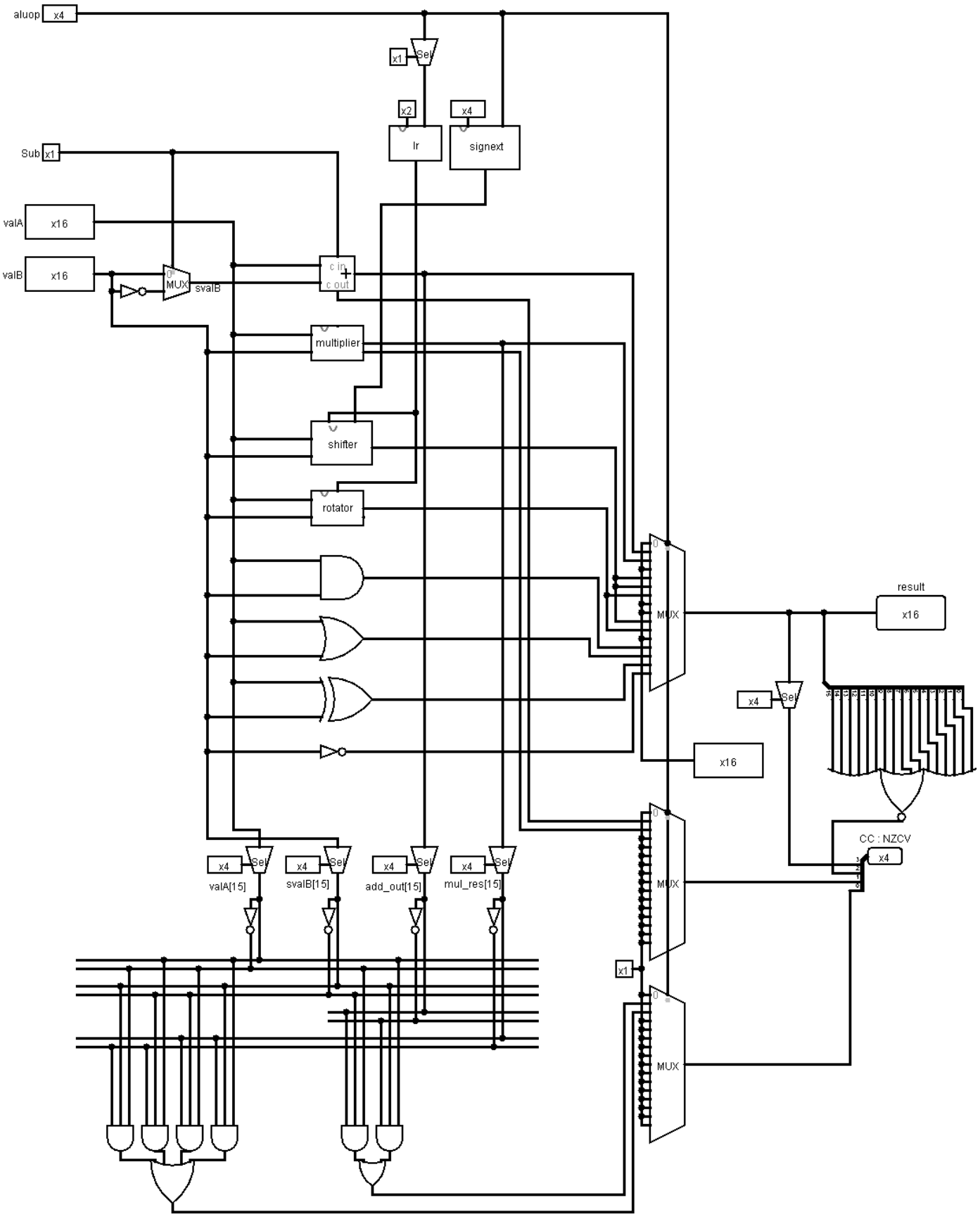
Class : 00

201602004 박태현

I. 실습 목적

ALU 구현

II. Design Procedure



III. Simulation

```
`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) ? and16b :
(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] & ~add_out[15]) |    // -a + -b = +c : 1 1 0
 (~valA[15] & ~svalB[15] &  add_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]) |     // +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);
```

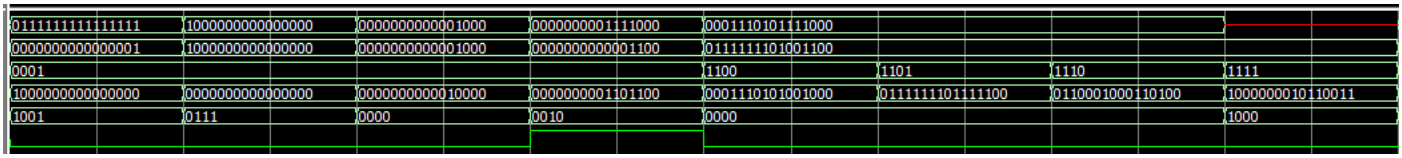
ALU를 만들고 연결해줍니다

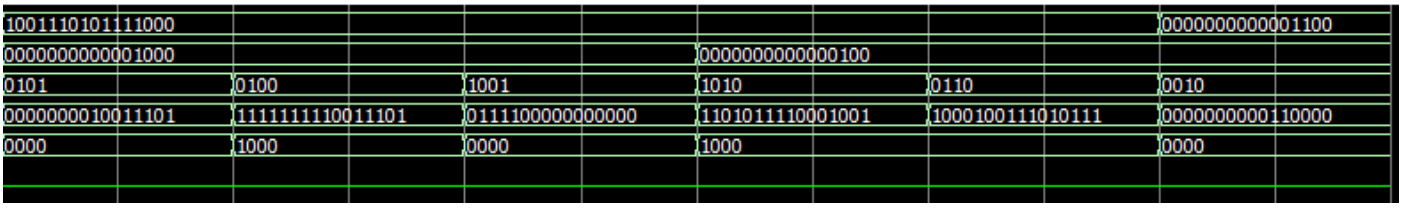
```
initial begin

    // 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;
    valB = 16'b0000_0000_0000_1000;
    aluop = `ADD;
    sub = 1'b0;
    #10;
    valA = 16'b0000_0000_0111_1000;
    valB = 16'b0000_0000_0000_1100;
    aluop = `ADD;
    sub = 1'b1;
    #10;
    valA = 16'b0001_1101_0111_1000;
    valB = 16'b0111_1111_0100_1100;
    aluop = `AND;
    #10;
    valA = 16'b0001_1101_0111_1000;
    valB = 16'b0111_1111_0100_1100;
    aluop = `OR;
    #10;
    valA = 16'b0001_1101_0111_1000;
    valB = 16'b0111_1111_0100_1100;
    aluop = `XOR;
    #10;
```

```
    valA = 16'b1001_1101_0111_1000;
    valB = 16'b0000_0000_0000_1000;
    aluop = `LSR;
    #10;
    valA = 16'b1001_1101_0111_1000;
    valB = 16'b0000_0000_0000_1000;
    aluop = `ASR;
    #10;
    valA = 16'b1001_1101_0111_1000;
    valB = 16'b0000_0000_0000_1000;
    aluop = `SHL;
    #10;
    valA = 16'b1001_1101_0111_1000;
    valB = 16'b0000_0000_0000_0100;
    aluop = `RL;
    #10;
    valA = 16'b1001_1101_0111_1000;
    valB = 16'b0000_0000_0000_0100;
    aluop = `RR;
    #10;
    valA = 16'b0000_0000_0000_1100;
    valB = 16'b0000_0000_0000_0100;
    aluop = `MUL;
    #10;
```

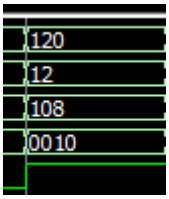
각 계산을 한번씩 해보았습니다

IV. Evaluation

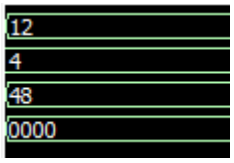




각 연산을 한번씩 해보았습니다

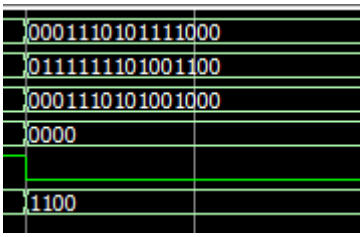


빼기

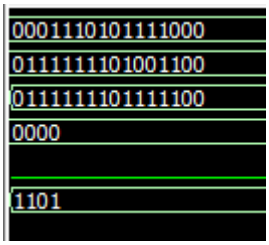


12 * 4 = 48

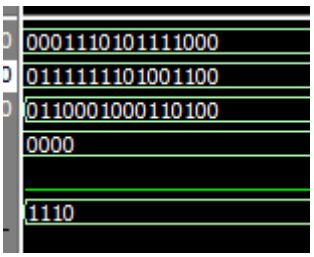
AND



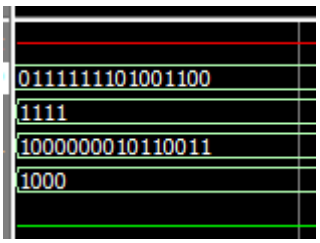
OR



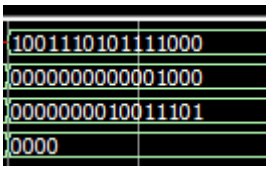
XOR



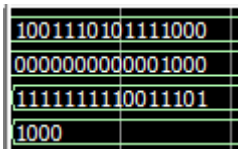
NOT



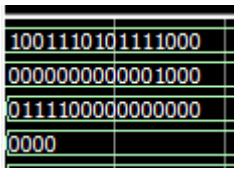
LSR



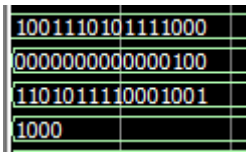
ASR



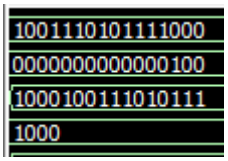
SHL



RL



RR



V. Discussion

여러 모듈을 이어서 하나를 만드는 것이 생각보다 헷갈렸습니다

만들고보니 MUX 입력이 남는데 어떻게 하면 좋을지 잘 모르겠습니다