Lab2: 32-bit ALU

TA-黃威淳

s094398.cs10@nycu.edu.tw

Tool

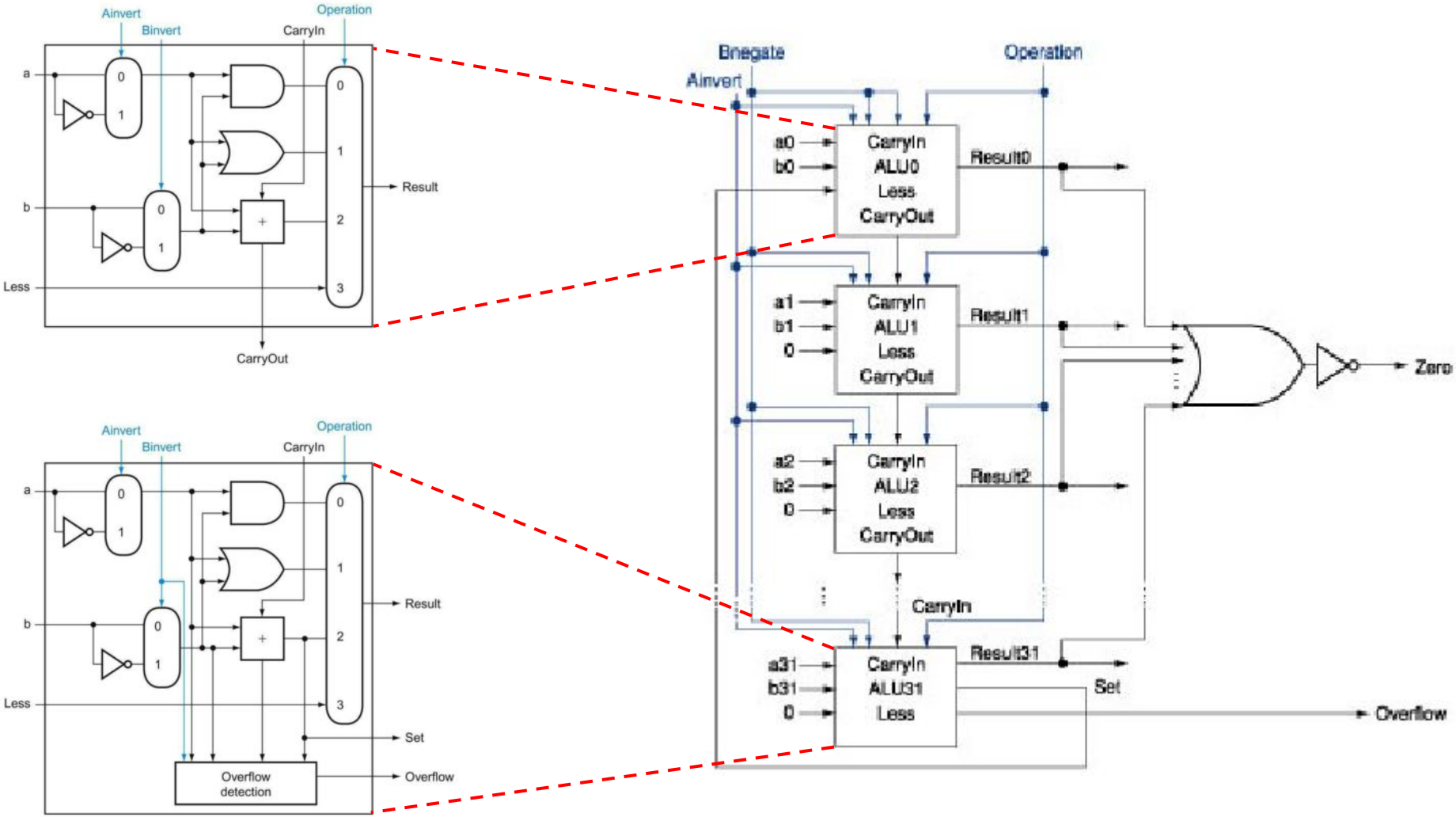
- **Icarus verilog**
  - Mac OSX
    - brew install icarus-verilog
  - Windows
    - https://bleyer.org/icarus/iverilog-v11-20210204-x64_setup.exe
  - Ubuntu
    - sudo apt install verilog
- **GTKWave (unnecessary)**
  - Easy to debug
- **Other HDL simulator (unnecessary)**
  - Easy for debug

Attached file

- **TODO**
  - alu.v (32-bit ALU)
  - alu_1bit.v
  - MUX2to1.v
  - MUX4to1.v
- **Validate the correction of your implementation (32 bit ALU)**
  - testbench.v
- **Testcase**
  - *.txt

Please do not modify these files

- **Simple testbench (1 bit ALU)**
  - alu_1bit_tb.v


Ainvert
Binvert
CarryIn
Operation
a
b
Less
Result
CarryOut
Bnegate
Ainvert
Operation
a0
b0
CarryIn
ALU0
Less
CarryOut
Result0
a1
b1
0
ALU1
Result1
a2
b2
ALU2
Result2
a31
b31
ALU31
Result31
Set
Overflow
Zero
Overflow detection

MUX 2to1 & 4to1

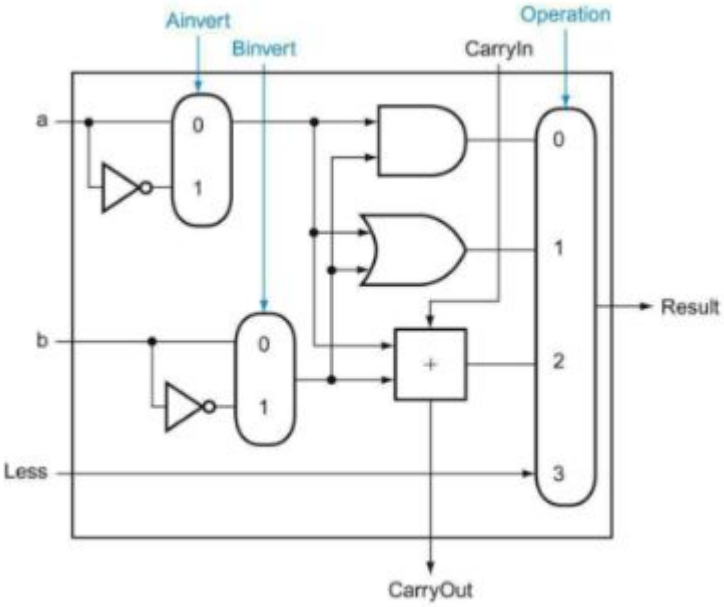
```
module MUX2to1(
    input       src1,
    input       src2,
    input       select,
    output reg result
    );
/* Write your code HERE */

endmodule
```

```
module MUX4to1(
    input            src1,
    input            src2,
    input            src3,
    input            src4,
    input   [2-1:0] select,
    output reg       result
    );
/* Write your code HERE */

endmodule
```

1 Bit ALU



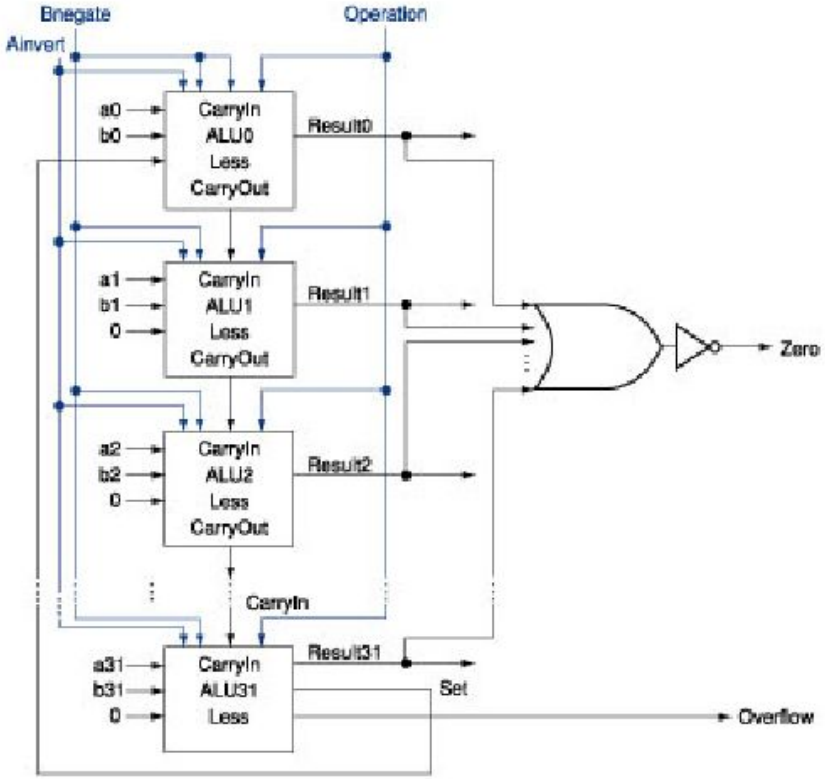

```
`timescale 1ns/1ps

module alu_1bit(
    input               src1,       //1 bit source 1  (input)
    input               src2,       //1 bit source 2  (input)
    input               less,       //1 bit less      (input)
    input               Ainvert,    //1 bit A_invert  (input)
    input               Binvert,    //1 bit B_invert  (input)
    input               cin,        //1 bit carry in  (input)
    input       [2-1:0] operation,  //2 bit operation (input)
    output reg          result,     //1 bit result    (output)
    output reg          cout        //1 bit carry out (output)
    );

/* Write your code HERE */

endmodule
```

32 bit ALU




```
`timescale 1ns/1ps

module alu(
    input                       rst_n,          // negative reset           (inp
    input       [32-1:0]        src1,           // 32 bits source 1         (inp
    input       [32-1:0]        src2,           // 32 bits source 2         (inp
    input       [ 4-1:0]        ALU_control,    // 4 bits ALU control input (inp
    output reg  [32-1:0]        result,         // 32 bits result           (out
    output reg                  zero,           // 1 bit when the output is 0, ze
    output reg                  cout,           // 1 bit carry out          (out
    output reg                  overflow        // 1 bit overflow           (out
    );

/* Write your code HERE */

endmodule
```

Testbench - 1 bit

Your code

- $ iverilog -o 1bit alu_1bit_tb.v alu_1bit.v MUX*

output file (default: a.out)

- $ ./1bit
- We provide simple 1-bit ALU testbench, feel fre
  to use it !
- Just for testing your 1-bit ALU correctness

```
sum x
carry x
===============
sum x
carry x
===============
sum x
carry x
```

alu_1bit_tb.v

```
initial // initial block executes only once
    begin

        // values for a and b
        a = 1;
        b = 1;
        less = 0;
        Ainvert = 0;
        Binvert = 0;
        cin = 0;
        operation = 2'b00;
        #1
        $display("sum %d", sum);
        $display("carry %d", carry);
        $display("===============");
        #period; // wait for period

        a = 1;
        b = 1;
        less = 0;
        Ainvert = 0;
        Binvert = 0;
        cin = 0;
        operation = 2'b01;
        #1
        $display("sum %d", sum);
        $display("carry %d", carry);
        $display("===============");
        #period;

        a = 1;
        b = 1;
        less = 0;
        Ainvert = 0;
        Binvert = 0;
        cin = 0;
        operation = 2'b11;
        #1
        $display("sum %d", sum);
        $display("carry %d", carry);
        $display("===============");
        #period;

    end
```

You can modify these value

Testbench

Your code

- $ iverilog -o lab2 alu.v alu_1bit.v testbench.v MUX*

output file (default: a.out)

- $ ./lab2
- Do not modify testbench.v

```
**************************************************
*             PATTERN RESULT TABLE               *
**************************************************
* PATTERN *              Result             * ZCV *
**************************************************
*      Congratulation! All data are correct!     *
**************************************************
Correct Count: 30
```

```
**************************************************
*             PATTERN RESULT TABLE               *
**************************************************
* PATTERN *              Result             * ZCV *
**************************************************
* No. 1 error!                                   *
* Correct result: eeeeeeee     Correct ZCV: 000  *
* Your result: xxxxxxxx        Your ZCV: xxx     *
**************************************************
* No. 2 error!                                   *
* Correct result: 00000000     Correct ZCV: 100  *
* Your result: xxxxxxxx        Your ZCV: xxx     *
**************************************************
* No. 3 error!                                   *
* Correct result: 9bf5fea6     Correct ZCV: 000  *
* Your result: xxxxxxxx        Your ZCV: xxx     *
**************************************************
* No. 4 error!                                   *
* Correct result: 00000000     Correct ZCV: 110  *
* Your result: xxxxxxxx        Your ZCV: xxx     *
**************************************************
```

Testcase

- testcase.txt
  - 30 cases

number	OP	function	src1	src2	result	zcv
1	d	NAND	11111111	11111111	eeeeeeee	000
2	0	AND	ffff0000	0000ffff	00000000	100
3	1	OR	9284fc02	8b71d6a6	9bf5fea6	000
4	2	ADD	ffffffff	00000001	00000000	110
5	2	ADD	7fffffff	7fffffff	fffffffe	001
6	6	SUB	c4fab894	54b51e1e	70459a76	011
7	6	SUB	76543210	01234567	7530eca9	010
8	7	SLT	00000000	0fffffff	00000001	000
9	C	NOR	00000000	00000000	ffffffff	000
10	2	ADD	00000000	00000000	00000000	100
11	2	ADD	00000000	ffffffff	ffffffff	000
12	2	ADD	00000001	ffffffff	00000000	110
13	6	SUB	00000000	00000000	00000000	110
14	6	SUB	00000000	ffffffff	00000001	000
15	6	SUB	00000001	00000001	00000000	110
16	6	SUB	00000100	00000001	000000ff	010
17	7	SLT	00000000	00000000	00000000	100
18	7	SLT	00000000	00000001	00000001	000
19	7	SLT	00000000	ffffffff	00000000	100
20	7	SLT	00000001	00000000	00000000	100
21	7	SLT	ffffffff	00000000	00000001	000
22	0	AND	ffffffff	ffffffff	ffffffff	000
23	0	AND	ffffffff	12345678	12345678	000
24	0	AND	12345678	87654321	02244220	000
25	0	AND	00000000	ffffffff	00000000	100
26	1	OR	ffffffff	ffffffff	ffffffff	000
27	1	OR	12345678	87654321	97755779	000
28	1	OR	00000000	ffffffff	ffffffff	000
29	1	OR	00000000	00000000	00000000	100
30	d	NAND	00000000	ffffffff	ffffffff	000

How to debug

- 1-bit ALU
  - $ iverilog -o 1bit alu_1bit_tb.v alu_1bit.v MUX*
  - $ ./1bit
  - You will get **alu_1bit.vcd** and open it with GTKWave
- 32-bit ALU
  - $ iverilog -o lab2 alu.v alu_1bit.v testbench.v MUX*
  - $ ./lab2
  - You will get **alu.vcd** and open it with GTKWave

Hint

Verilog generate & for may help you to finish this lab