**Lab 1**

## Discussion Questions

1. How many gates does your one-bit multiplexer use? The 32-bit multiplexer? Write down a formula for an N-bit multiplexer.

答：1位多选器：1个非门，2个与门，1个或门，总共4个门。

32位多选器：4\*32 = 128个

N位多选器：4\*N个门

1. We wrote a polymorphic function implementing an N-bit multiplexer. Explain how to write a polymorphic version of the left shifter.

答：module mkLeftShifter (LeftShifter);

method Bit#(32) shift(ShiftMode mode, Bit#(32) operand, Bit#(5) shamt);

在算数和逻辑右移的基础上，把右移判断是算数还是逻辑的flag变量去掉，因为左移都是右边添加符号0，然后改调用函数为下面：

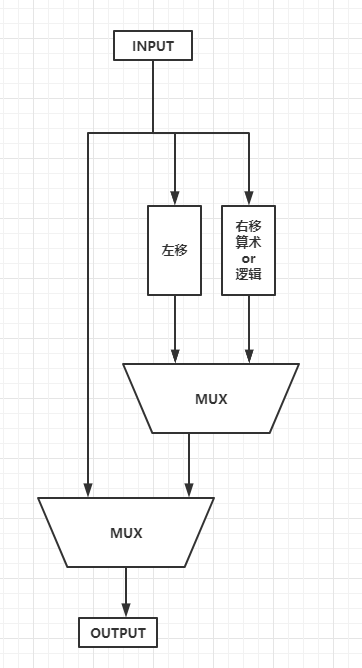
multiplexerN( shamt[1] , result, { result[31-k:0],copy\_frontnum(sign\_bit,k) }); k = 1，2,4,8,16

1. One purpose of this lab was to demonstrate a microarchitectural optimization. How many gates did we save by combining the logical and arithmetic right shifts?

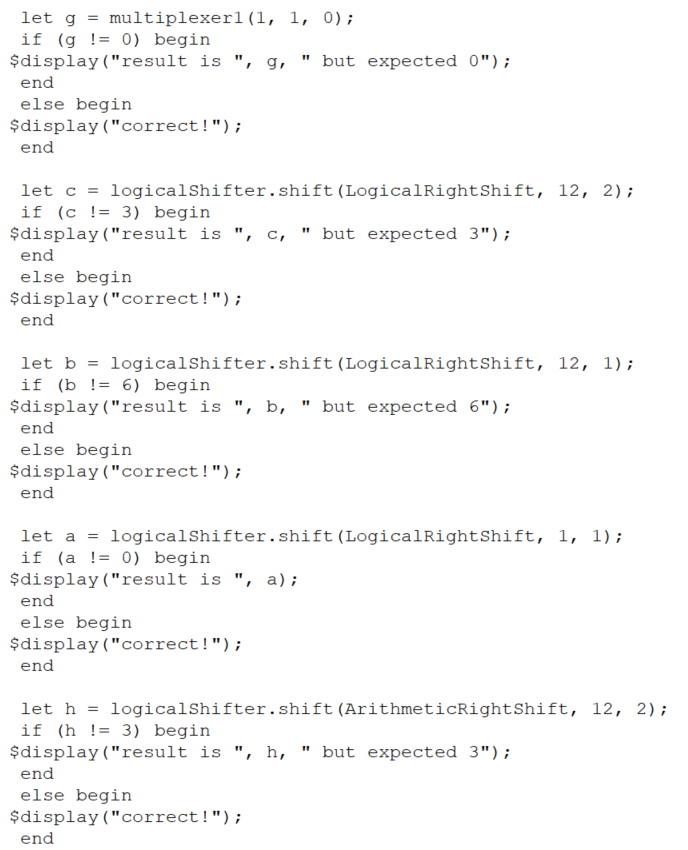
答：如果分开实现要10个MUX，合并实现则只要6个MUX。

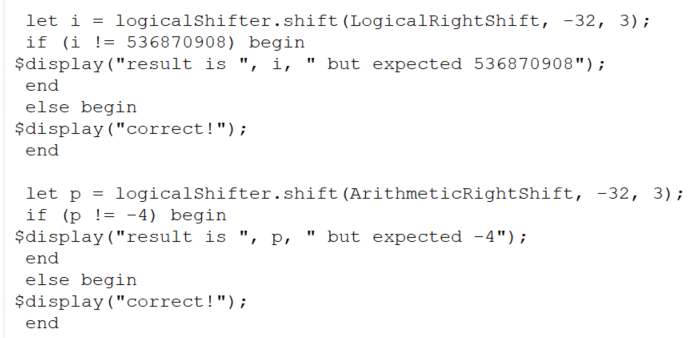
总共节省4个MUX使用，4\*32\*4 = 512 门。

1. Our right shifter handles right shifts only. However, with a small extension, it can handle left shifts as well. Draw a microarchitecture for this kind of combined shifter. How much hardware do we save?

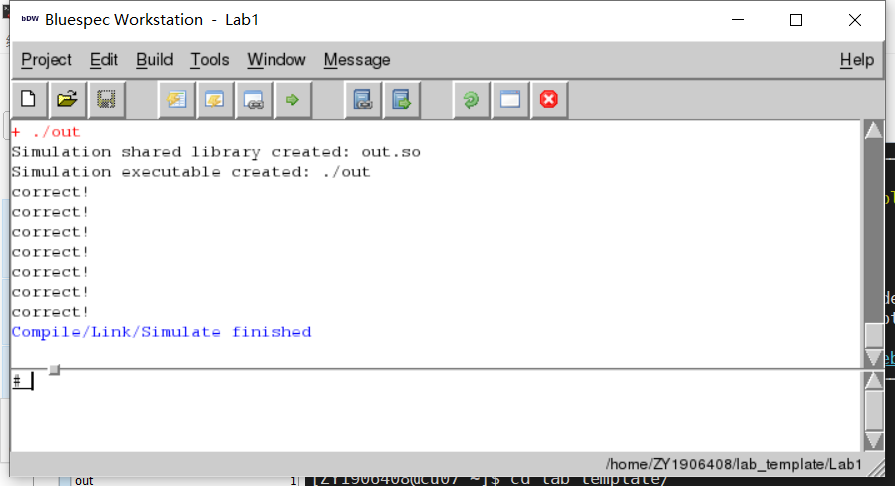
答：左移要5个MUX，减去一个判断符号位的MUX，右移合并逻辑和算数右移要6个MUX，同时判断输出是input、左移、右移也要2个MUX，总共13个MUX。

**测试代码：**





**测试结果**



**代码：**

import RightShifterTypes::\*;

import Gates::\*;

function Bit#(1) multiplexer1(Bit#(1) sel, Bit#(1) a, Bit#(1) b);

// Part 1: Re-implement this function using the gates found in the Gates.bsv file

// return (sel == 0)?a:b;

// 使用与或非门实现这条语句，对return a&~sel | b&sel 即可

let a\_out = andGate(a,notGate(sel));

let b\_out = andGate(b,sel);

let res = orGate(a\_out , b\_out);

return res;

endfunction

function Bit#(32) multiplexer32(Bit#(1) sel, Bit#(32) a, Bit#(32) b);

// Part 2: Re-implement this function using static elaboration (for-loop and multiplexer1)

// return (sel == 0)?a:b;

Bit#(32) aggregate = 0;

for(Integer i = 0; i < 32; i = i + 1)

begin

aggregate[i] = multiplexer1(sel, a[i], b[i]);

end

return aggregate;

endfunction

function Bit#(n) multiplexerN(Bit#(1) sel, Bit#(n) a, Bit#(n) b);

// Part 3: Re-implement this function as a polymorphic function using static elaboration

// return (sel == 0)?a:b;

Bit#(n) aggregate = 0;

for(Integer i = 0; i < valueof(n); i = i + 1)

begin

aggregate[i] = multiplexer1(sel, a[i], b[i]);

end

return aggregate;

endfunction

function Bit#(n) copy\_frontnum(Bit#(1) sign , Integer num);

Bit#(n) result = 0;

for(Integer i = 0;i < num;i = i+1)

begin

result[i] = sign;

end

return result;

endfunction

module mkRightShifter (RightShifter);

method Bit#(32) shift(ShiftMode mode, Bit#(32) operand, Bit#(5) shamt);

// Parts 4 and 5: Implement this function with the multiplexers you implemented

// pack函数用于将各种Bit数据类型（例如Bool，Int或UInt）转换（或打包）为（Bit＃（n））类型。

Bit#(32) result = 0;

Bit#(1) flag = 0;

Bit#(1) sign\_bit = operand[31];

// flag用来判断是逻辑还是算数右移，逻辑右移前面补0，算数右移前面补符号位。

if(mode == LogicalRightShift)

begin

flag = 1;

end

// 确定要右移前面要填充的0 OR 1

sign\_bit = multiplexerN(flag , operand[31] , 0);

// 按照位置右移

result = multiplexerN( shamt[0] , operand, {sign\_bit,operand[31:1]});

result = multiplexerN( shamt[1] , result, {copy\_frontnum(sign\_bit,2),result[31:2]});

result = multiplexerN( shamt[2] , result, {copy\_frontnum(sign\_bit,4),result[31:4]});

result = multiplexerN( shamt[3] , result, {copy\_frontnum(sign\_bit,8),result[31:8]});

result = multiplexerN( shamt[4] , result, {copy\_frontnum(sign\_bit,16),result[31:16]});

return result;

endmethod

endmodule