Lab1 实验报告

PART1 MULTIPLEXER1

```
function Bit#(1) multiplexer1(Bit#(1) sel, Bit#(1) a, Bit#(1) b);
    // ret = a & ~sel | b & sel
    return orGate(
        andGate(a, notGate(sel)),
        andGate(b, sel)
    );
endfunction
```

PART2 MULTIPLEXER32

```
function Bit#(32) multiplexer32(Bit#(1) sel, Bit#(32) a, Bit#(32) b);
    Bit#(32) mux_out;
    for (Integer i = 0; i < 32; i = i + 1) begin
        mux_out[i] = multiplexer1(sel, a[i], b[i]);
    end
    return mux_out;
endfunction
```

PART3 MULTIPLEXERN

```
function Bit#(n) multiplexerN(Bit#(1) sel, Bit#(n) a, Bit#(n) b);
    Bit#(n) mux_out;
    for (Integer i = 0; i < valueOf(n); i = i + 1) begin
        mux_out[i] = multiplexer1(sel, a[i], b[i]);
    end
    return mux_out;
endfunction
```

PART4 & PART5 RIGHT SHIFTER

```
module mkRightShifter(RightShifter);
    method Bit#(32) shift(ShiftMode mode, Bit#(32) operand, Bit#(5) shamt);
        Bit#(32) result = operand;
        Bit#(1) sign_flag = case(mode)
            LogicalRightShift: 1'b0;
            ArithmeticRightShift: operand[31];
        endcase;

        for (Integer i = 0; i < 5; i = i + 1) begin
            Bit#(32) shifted = signExtend(sign_flag);
            for (Integer j = 0; j < 32 - 2 ** i; j = j + 1) begin
                shifted[j] = result[2 ** i + j];
            end
            result = multiplexer32(shamt[i], result, shifted);
        end

        return result;
    endmethod
endmodule
```

PART6 UNIT TESTING

```
$display("my test going ...");
// Test multiplexer1
Bit#(1) arr_mux_a[8] = {0, 1, 0, 1, 0, 1, 0, 1};
Bit#(1) arr_mux_b[8] = {0, 0, 1, 1, 0, 0, 1, 1};
Bit#(1) arr_mux_s[8] = {0, 0, 0, 0, 1, 1, 1, 1};
Bit#(1) arr_mux_o[8] = {0, 1, 0, 1, 0, 0, 1, 1};
for (Integer i = 0; i < 8; i = i + 1) begin
    let out = multiplexer1(arr_mux_s[i], arr_mux_a[i], arr_mux_b[i]);
    if (out != arr_mux_o[i]) begin
        $display("result is ", out, " but expected ", arr_mux_o[i]);
    end
    else begin
        $display("correct!");
    end
end
$display("multiplexer1 test finished!");
```

```
// Test LogicalRightShift
Bit#(32) arr_lrs_oprand[4] = {
    32'hFFFFFFFF, 32'hFFFFFFFF, 32'd255, 32'd255};
Bit#(5) arr_lrs_shamt[4] = {5'd15, 5'd31, 5'd3, 5'd0};
Bit#(32) arr_lrs_out[4] = {32'h1FFFF, 32'h1, 32'd31, 32'd255};
for (Integer i = 0; i < 4; i = i + 1) begin
    let out = logicalShifter.shift(
        LogicalRightShift, arr_lrs_oprand[i], arr_lrs_shamt[i]);
    if (out != arr_lrs_out[i]) begin
        $display("result is ", out, " but expected ", arr_lrs_out[i]);
    end
    else begin
        $display("correct!");
    end
end
$display("LogicalRightShift test finished!");

// Test ArithmeticRightShift
Bit#(32) arr_ars_oprand[5] = {
    32'hFFFFFFFF, 32'hFFFFFFFF, 32'hFFFFFFF1, 32'd255, 32'd255};
Bit#(5) arr_ars_shamt[5] = {5'd15, 5'd31, 5'd3, 5'd3, 5'd0};
Bit#(32) arr_ars_out[5] = {
    32'hFFFFFFFF, 32'hFFFFFFFF, 32'hFFFFFFFE, 32'd31, 32'd255};
for (Integer i = 0; i < 5; i = i + 1) begin
    let out = logicalShifter.shift(
        ArithmeticRightShift, arr_ars_oprand[i], arr_ars_shamt[i]);
    if (out != arr_ars_out[i]) begin
        $display("result is ", out, " but expected ", arr_ars_out[i]);
    end
    else begin
        $display("correct!");
    end
end
$display("ArithmeticRightShift test finished!");
```

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.

   1-bit MUX 使用了 1 个或门，2 个与门，1 个非门，一共 4 个门电路。

32-bit MUX 实际上是 32 个 1-bit MUX 组合而成，每个比特之间相互独立，一共用了 128 个门电路。

N-bit MUX 使用的门电路数目是$4N$。

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

```
module mkLeftShifter(LeftShifter);
    method Bit#(32) shift(Bit#(32) operand, Bit#(5) shamt);
        Bit#(32) result = operand;
        for (Integer i = 0; i < 5; i = i + 1) begin
            Bit#(32) shifted = 32'h0;
            for (Integer j = 0; j < 32 - 2 ** i; j = j + 1) begin
                shifted[2 ** i + j] = result[j];
            end
            result = multiplexer32(shamt[i], result, shifted);
        end
        return result;
    endmethod
endmodule
```

3. 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?

将逻辑右移和算数右移的逻辑聚合在一起节省了新增的算术右移逻辑中的 5 个 32-bit MUX，和最终选择运算结果的 1 个 32-bit MUX，增加的代价是最初判断扩展比特的 1-bit MUX，因此节省的门电路数目为

$$128 \times 6 - 4 = 764$$

4. 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?

如果将操作数的所有比特倒置，那么对原操作数的左移运算便能够转化为倒置后的操作数的逻辑右移运算，再将移位运算的结果倒置便得到期望的运算结果。示意如下：

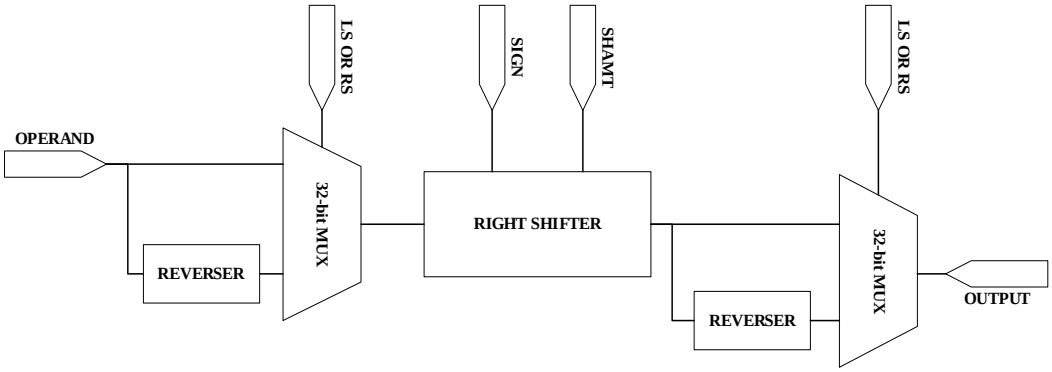

图 1 复用 RIGHT SHIFTER 电路的 LEFT SHIFTER

由上图可得，复用之后省去了左移器的 5 个 32-bit MUX 开销，增加了 1 个数据入口处 32-bit MUX 的开销。因此省去了 4 个 32-bit MUX 开销，共计 512 个基础门电路。