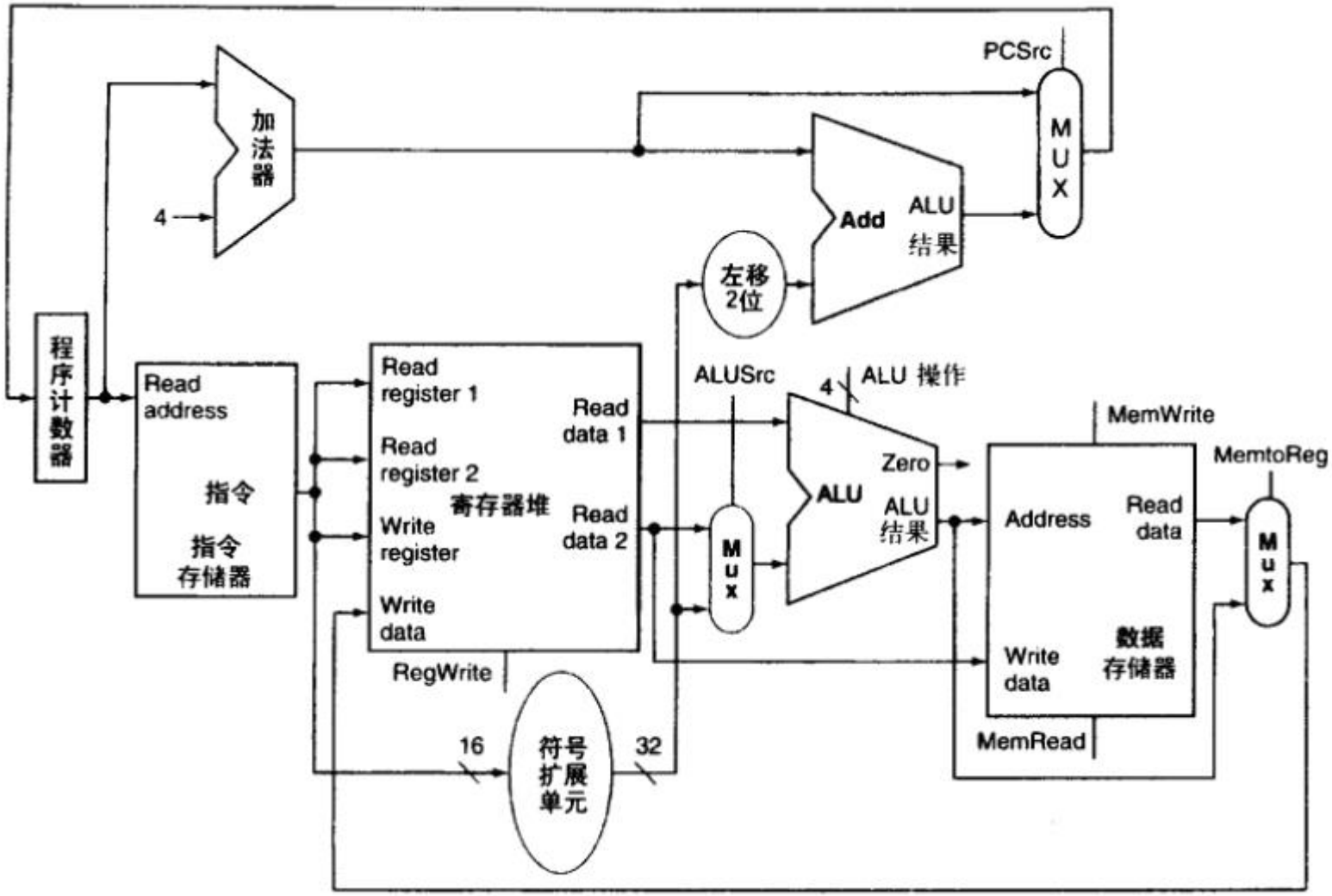
Verilog & Modelsim

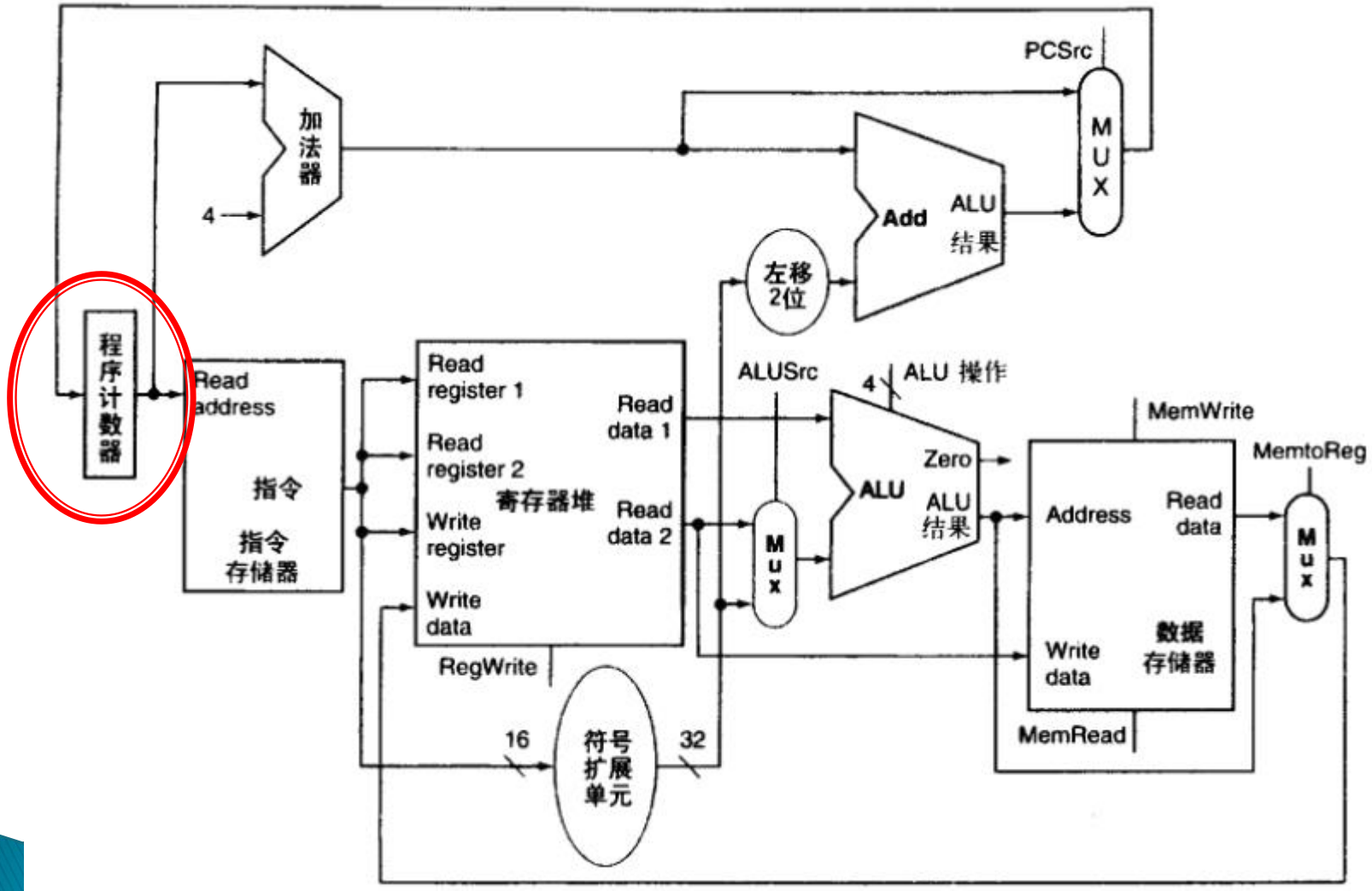
什么是VerilogHDL ？

- 定义：Verilog HDL是一种用于数字逻辑电路设计的硬件描述语言（Hardware Description Language）。
- 使用 Verilog HDL可以像设计软件一样设计硬件系统

简单的不完整数据通路

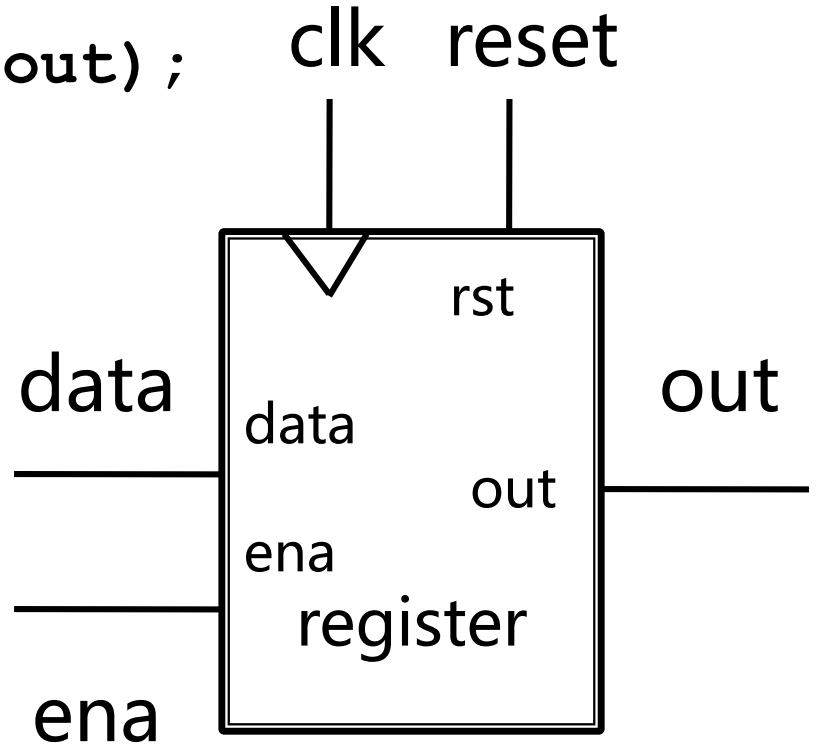

PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

程序计数器（register）

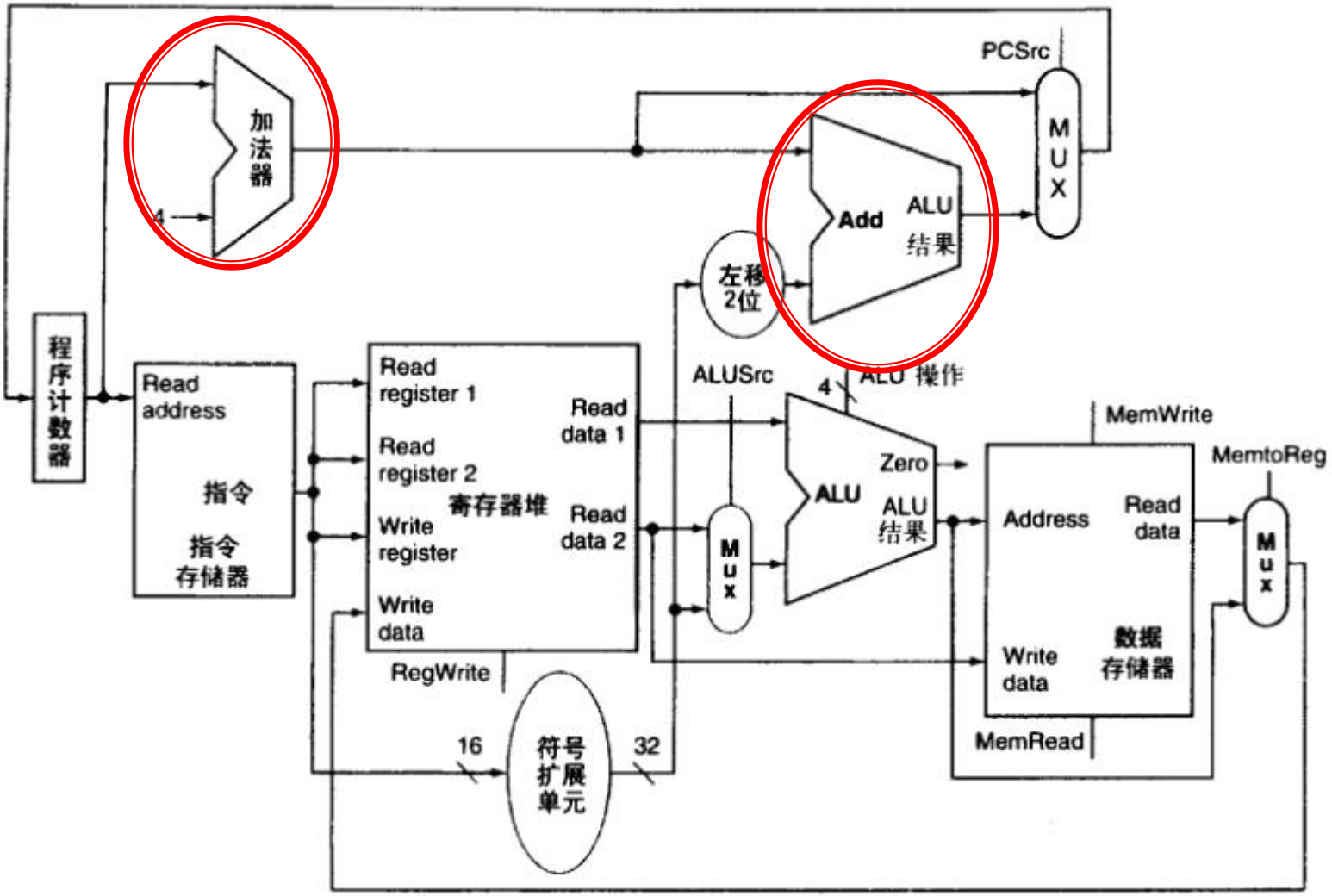


寄存器

```
module register #(parameter
  WIDTH = 32)
  (input ena, clk, rst,
  input [WIDTH-1:0] data,
  output reg [WIDTH-1:0] out);

always @(posedge clk)
  if(rst)
     out <= 0;
  else if (ena)
     out <= data;

endmodule
```

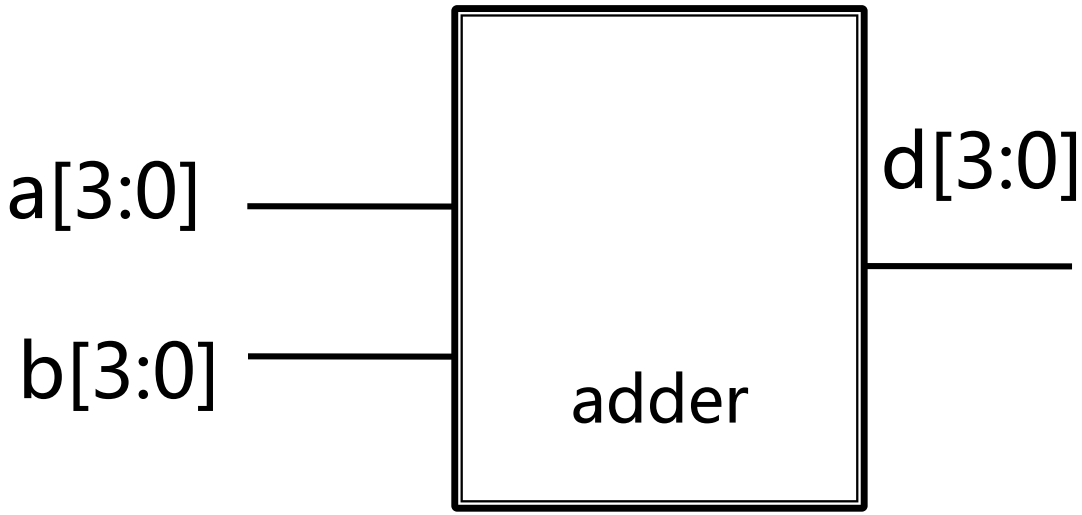


加法器

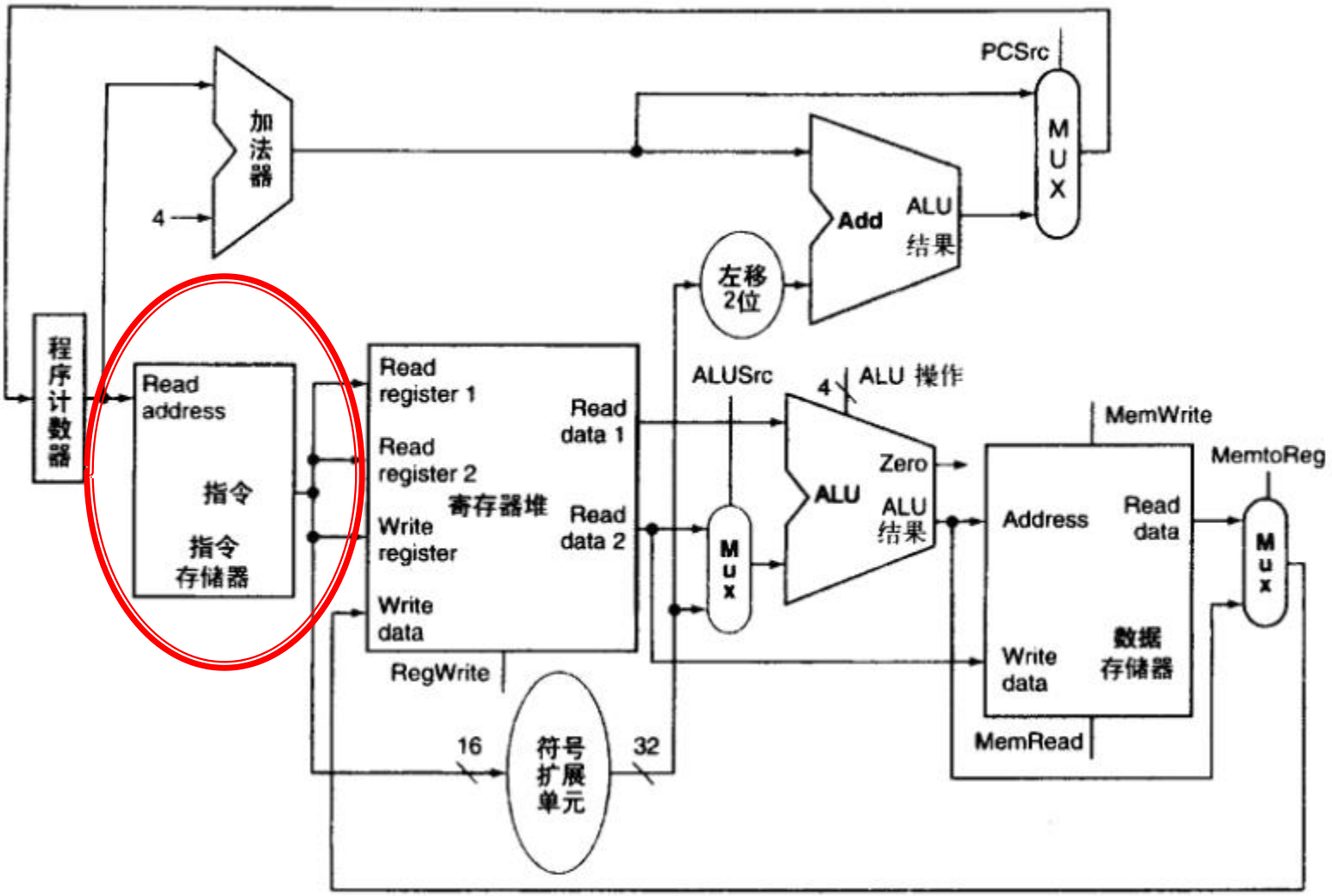

加法器
4
PCSrc
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
寄存器堆
RegWrite
ALUSrc
4
ALU 操作
M
u
x
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

加法器

```
module adder
  (input [3:0]a,
   input [3:0]b,
   output [3:0]y);

  assign y = a + b;

endmodule;
```

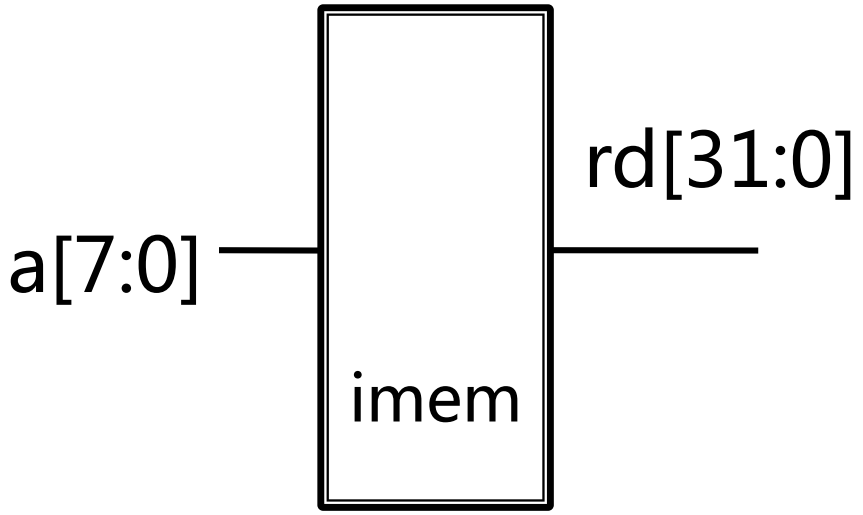


指令存储器

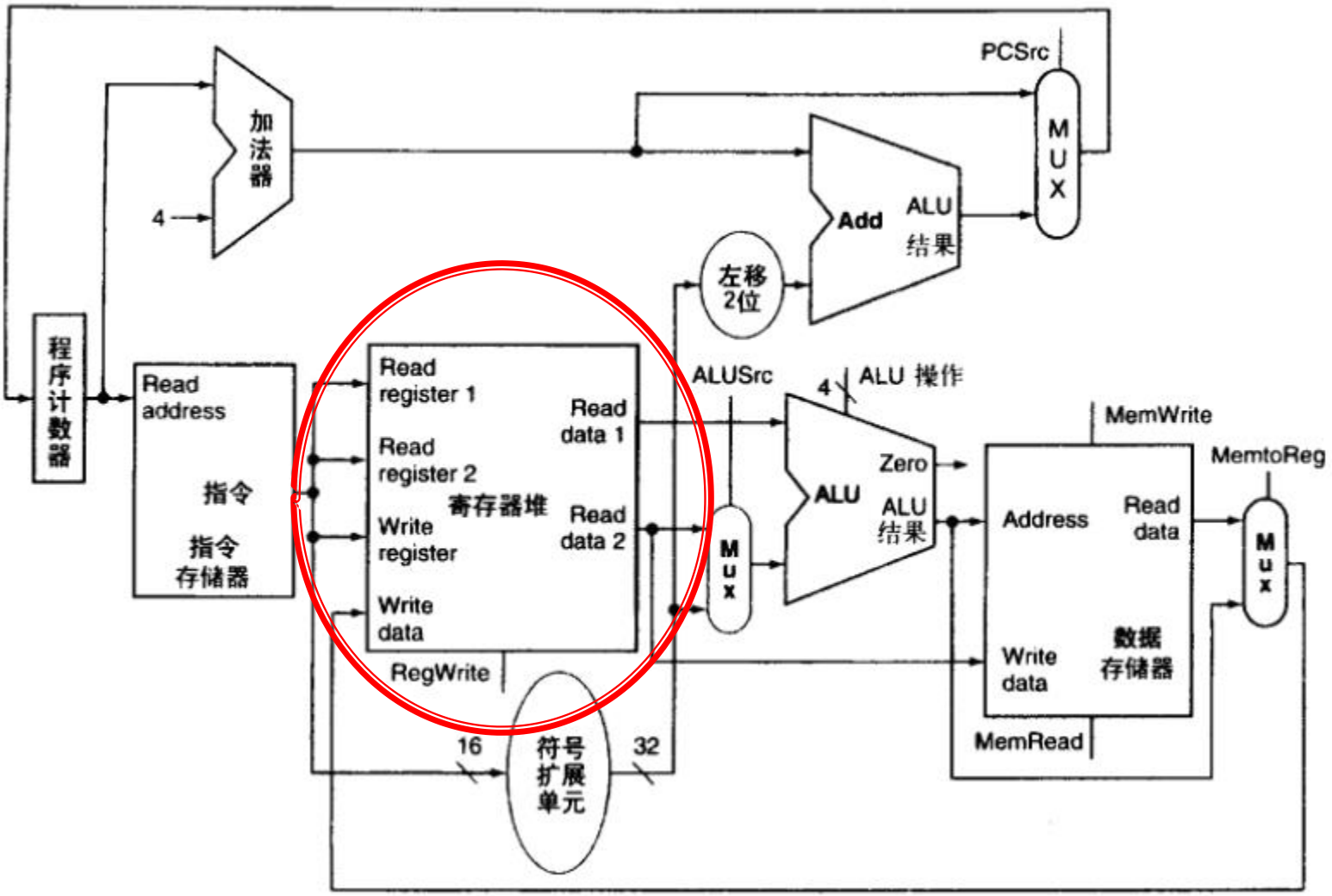

PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
Zero
ALU
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

指令存储器

```
module imem(
  input [7:0] a,
  output [31:0] rd);

  reg[31:0] RAM[63:0];

  initial
  begin
  $readmemh("testadd.dat",RAM);
  end

  assign rd = RAM[a[7:2]];
endmodule
```



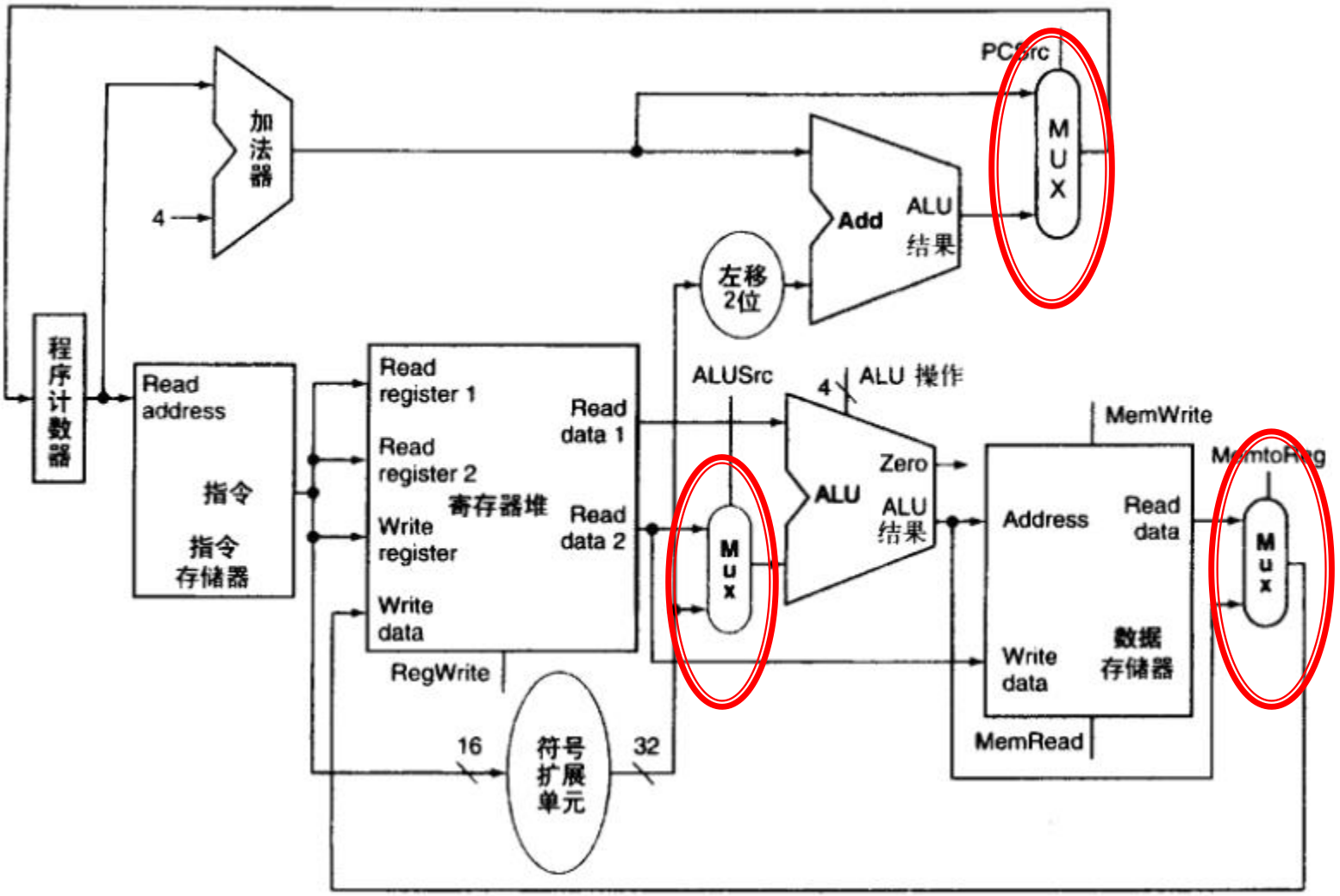
寄存器组


PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
M
u
x
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
M
u
x
16
符号
扩展
单元
32

寄存器组

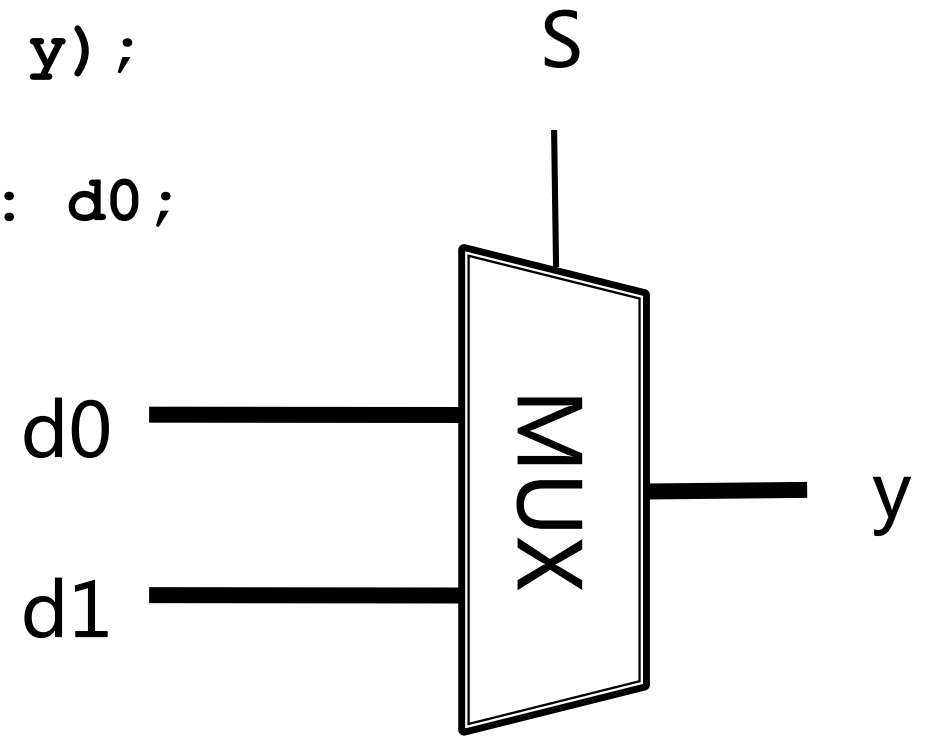
```
module regfile(input clk, input we3,
     input [4:0] ra1, ra2, wa3,
     input [31:0] wd3,
     output [31:0] rd1, rd2);

  reg[31:0]rf[31:0];
  always @(posedge clk)
     if(we3) rf[wa3] <= wd3;

  assign rd1 = (ra1 != 0)? rf[ra1]:0;
  assign rd2 = (ra2 != 0)? rf[ra2]:0;

endmodule
```

二路选择器

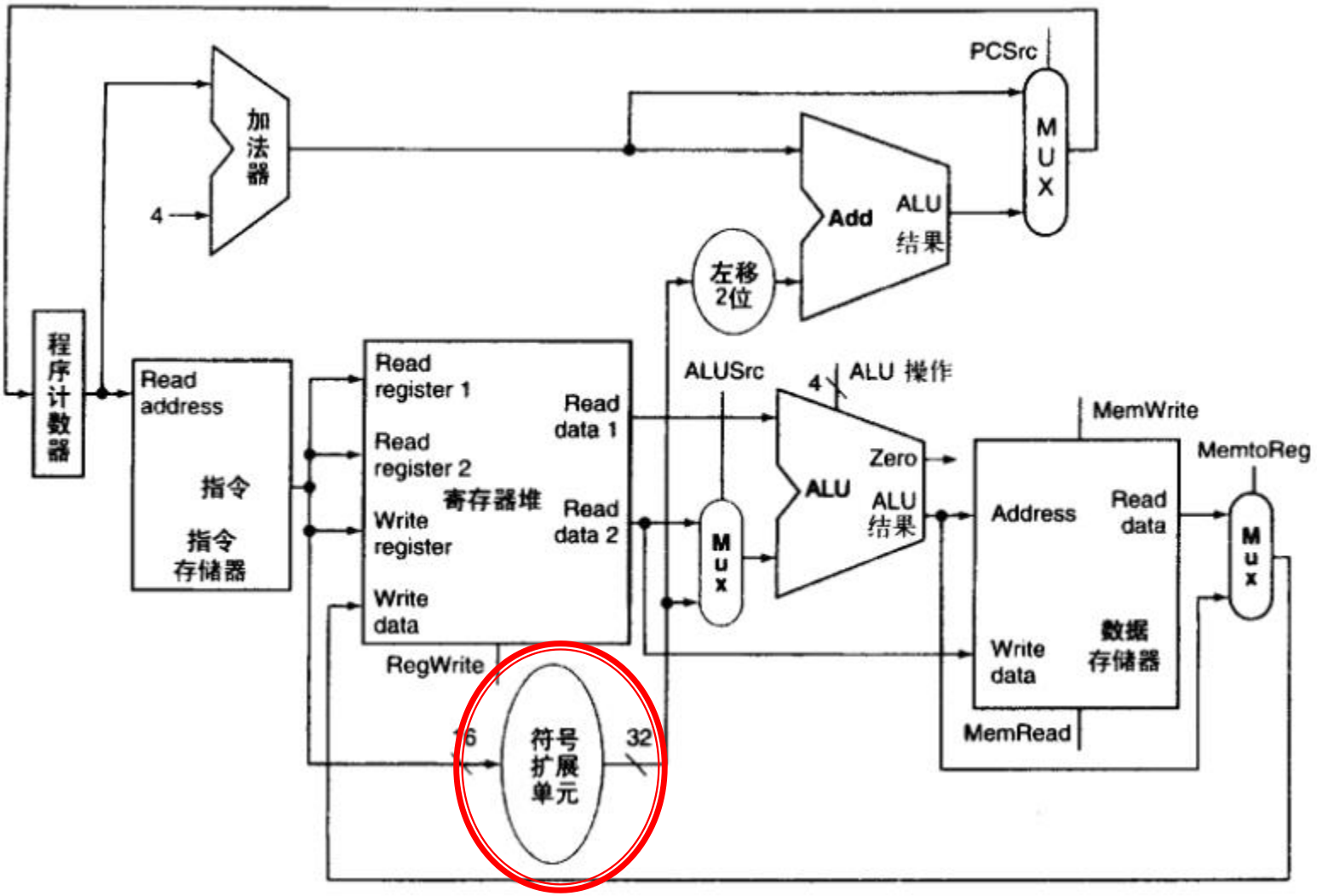

PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

二路选择器

```
module mux2 #(parameter WIDTH = 32)
(input [WIDTH-1:0] d0, d1,
  input s,
  output [WIDTH-1:0] y);

  assign y = s ? d1 : d0;

endmodule
```



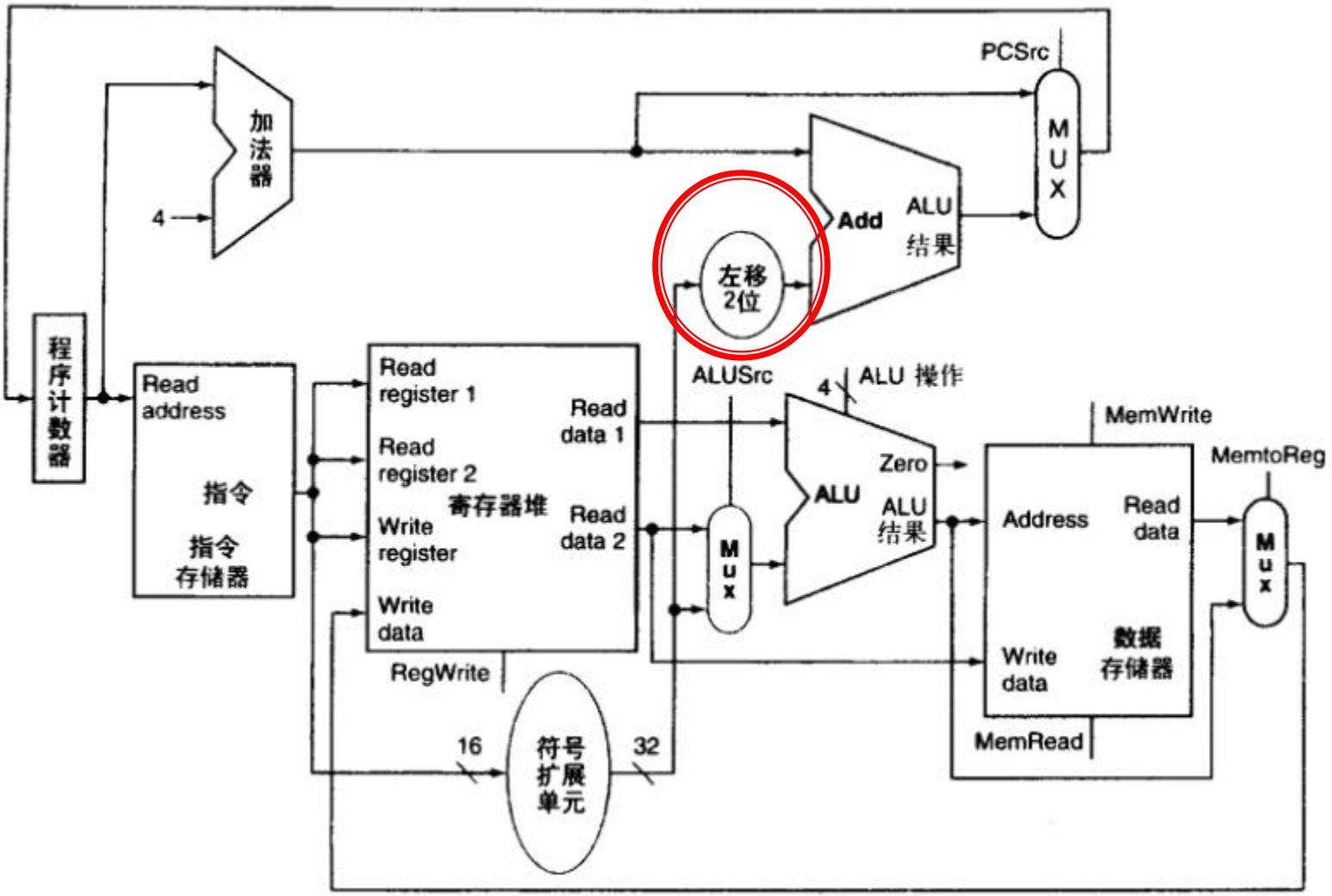
符号扩展单元


PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

符号扩展单元

```
module signext
  #(parameter WIDTH= 16)
  (input [WIDTH-1:0] a,
  output [31:0] y);

  assign y = {{32-WIDTH{a[WIDTH-1]}}, a};

endmodule
```

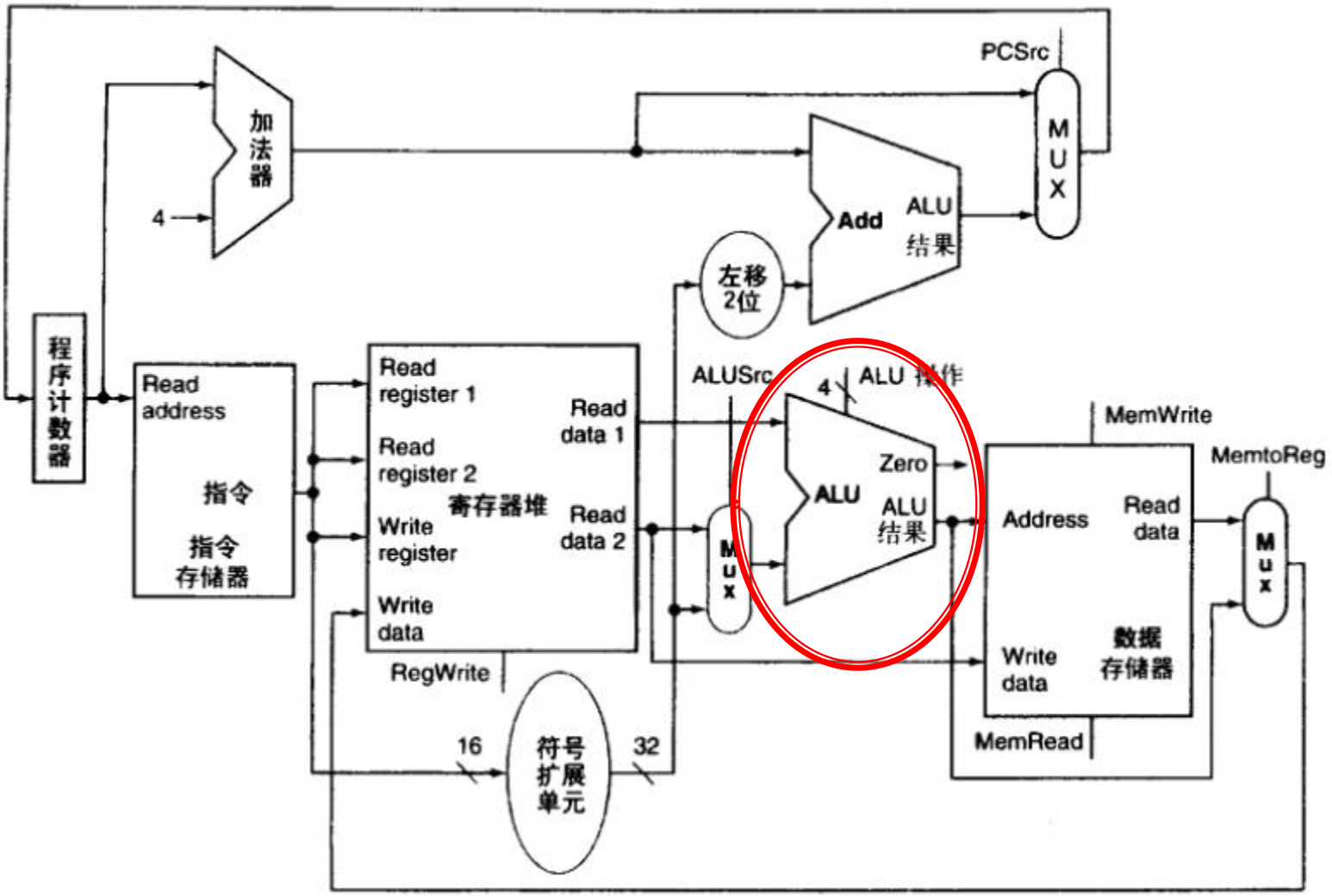
移位单元


PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

移位单元

```
module sl2 #(parameter WIDTH = 32) (
  input [WIDTH-1:0] a,
  output [WIDTH-1:0] y);
  //shift left by 2

  assign y = {a[WIDTH-3:0], 2'b00};

endmodule
```

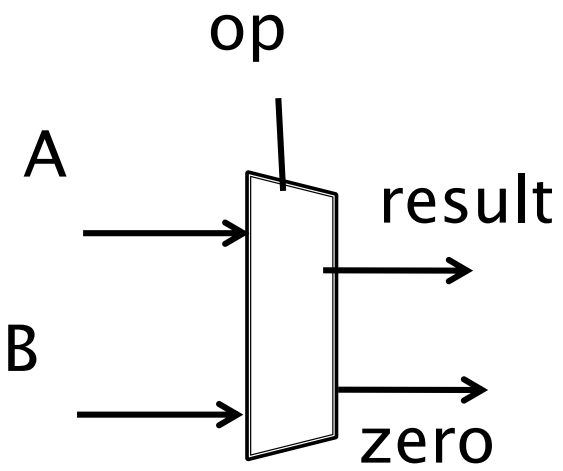
ALU 算术逻辑单元


PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

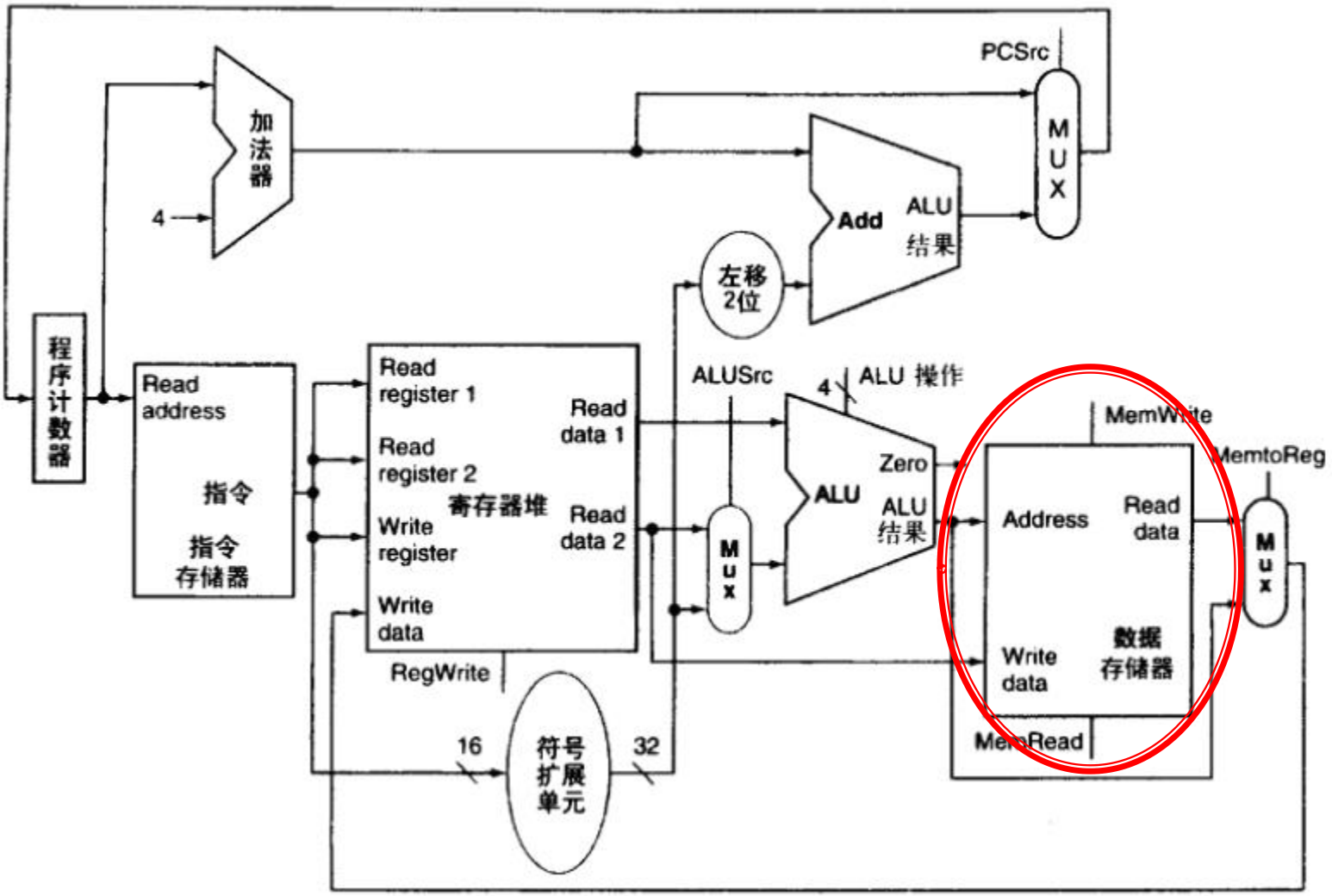
ALU

```
module ALU(
input [1:0] op,
input [7:0] A, B,
Output zero,
output reg [7:0] result,
);
assign zero = (result == 0) ? 1 : 0;
always @ (op or A or B)
  begin
      case(op)
            2'd00: result = A + B;
            2'd01: result = A - B;
            2'd10: result = A | B;
            2'd11: result = A & B;
      endcase
  end
endmodule
```

op	result
00	A + B
01	A – B
10	A | B
11	A & B



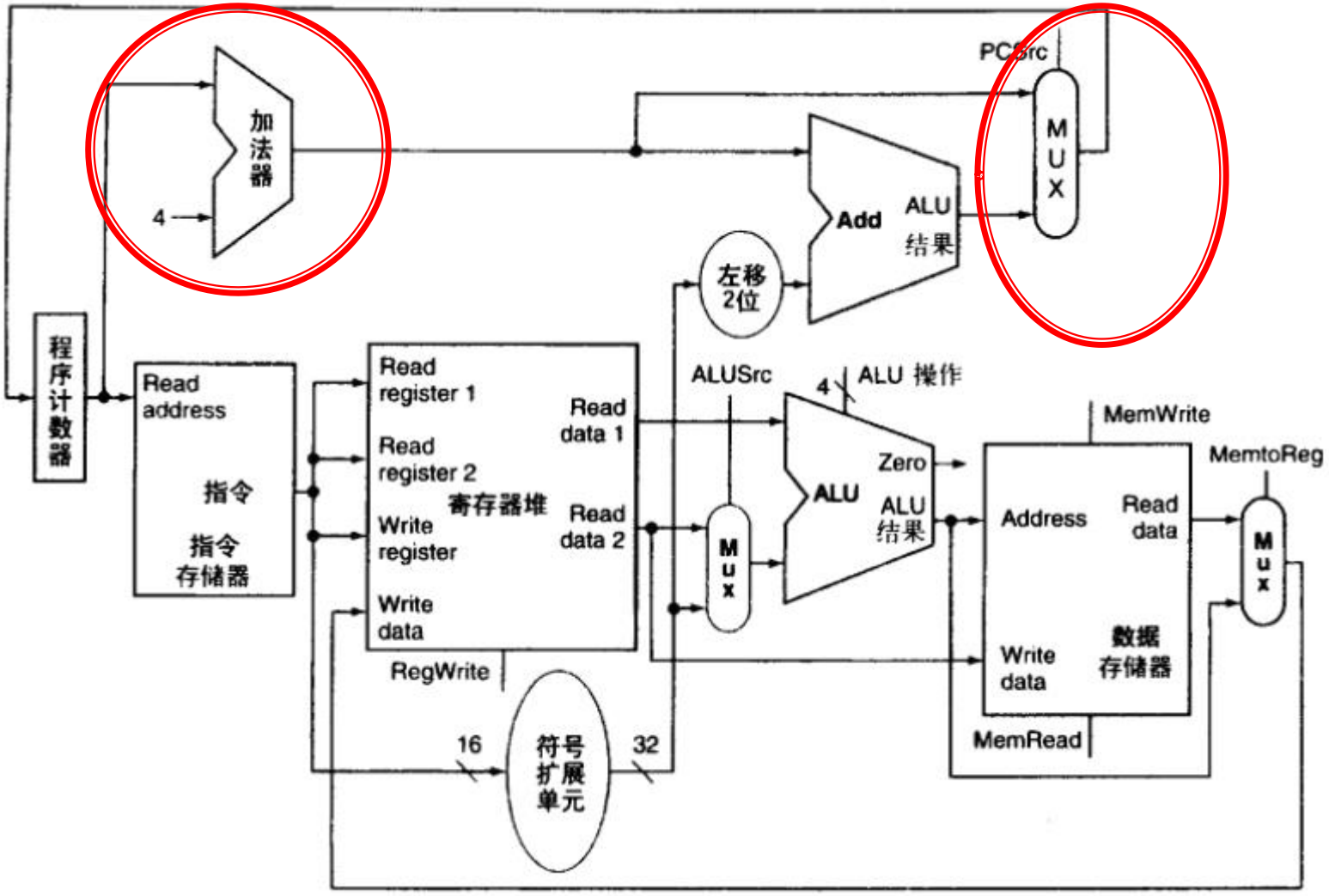
数据存储器


PCSrc
加法器
4
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
寄存器堆
RegWrite
16
符号
扩展
单元
32
ALUSrc
4
ALU 操作
M
u
x
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
M
u
x

数据存储器

```
module dmem(
  input clk, we,
  input [31:0] a, wd,
  output [31:0] rd);

  reg[31:0] dRAM[63:0];

  assign rd = dRAM[a[7:2]];

  always @(posedge clk)
     if(we)
          dRAM[a[7:2]] <= wd;

endmodule
```

NPC


加法器
4
PCSrc
M
U
X
Add
ALU
结果
左移
2位
程序计数器
Read
address
指令
指令
存储器
Read
register 1
Read
register 2
寄存器堆
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU 操作
M
u
x
ALU
Zero
ALU
结果
MemWrite
MemtoReg
Address
Read
data
Write
data
数据
存储器
MemRead
16
符号
扩展
单元
32

NPC

```
module NPC(
  input immi[15:0],
  input br, zero,
  input [31:0] pc,
  output [31:0] npc);

  wire [31:0] pc_plus_4, pc_br;
  assign pc_plus_4 = pc + 4;
  assign pc_br = pc_plus_4 +
                 {{14{immi[15]}}, immi, 2'b00};

  mux2 #(32) MUX (.d0(pc_plus_4), .d1(pc_br),
      .s(zero & br), .y(npc) );

endmodule
```

Modelsim工具简介

- ModelSim是业界最优秀的HDL语言仿真软件
- 我们主要用它进行功能仿真

功能仿真

- 什么是功能仿真？
- 确定一个设计是否实现了预定的功能
- 根据需要观察电路输入输出端口和电路内部任一信号和寄存器的波形
- 功能仿真是比较理想的仿真，不会因为器件的物理信息出现因为延迟等现象。

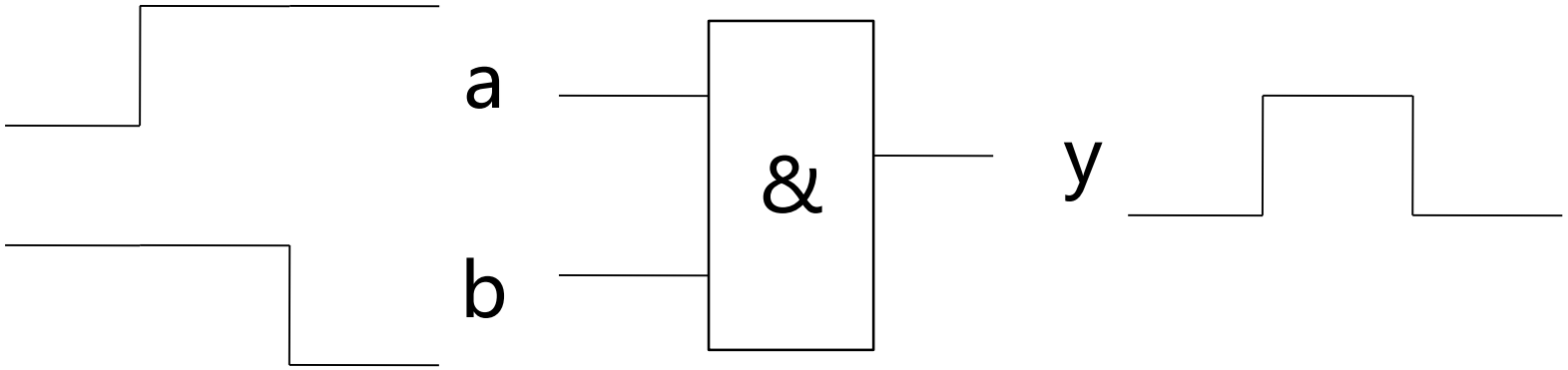
功能仿真示意图

- 下面的verilog语言描述了一个怎样的模块？

```
module s( a, b, y);

input a, b;
output y;

assign y = a & b;

endmodule
```

功能仿真示意图

- 怎样对这个模块进行功能仿真？



设计输入波形

验证输出波形

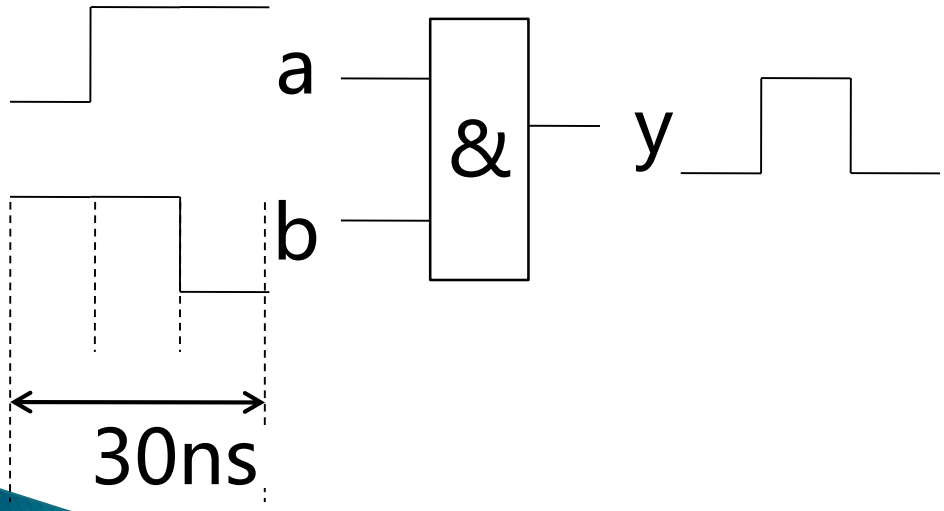
输出波形是否与预期相符？

设计输入波形的方法

- 怎样设置输入波形？
- 使用波形编辑器（有兴趣可自学）
- 使用verilog语言编写激励文件
- 什么是激励文件？
- verilog语言描述的给出输入信号具体值的文件

编写激励文件

- 怎样设置输入波形？




```
module testbench();

reg a, b;

s an(.a(a),.b(b), .y(out));

initial
begin
a = 1'b0;
b = 1'b1;
#10 a = 1'b1;
#10 b = 1'b0;
end

endmodule
```

testbench语法回顾

 # + number ： 时序控制，延时长度

initial块： 对存储器变量赋初值

```
  initial
     begin
          input = 1'b1;
      #10 input = 1'b0;
      #10 input = 1'b1;
      #10 input = 1'b0;
     end
```

always #number <语句>: 每隔**number**个时间后执行语句

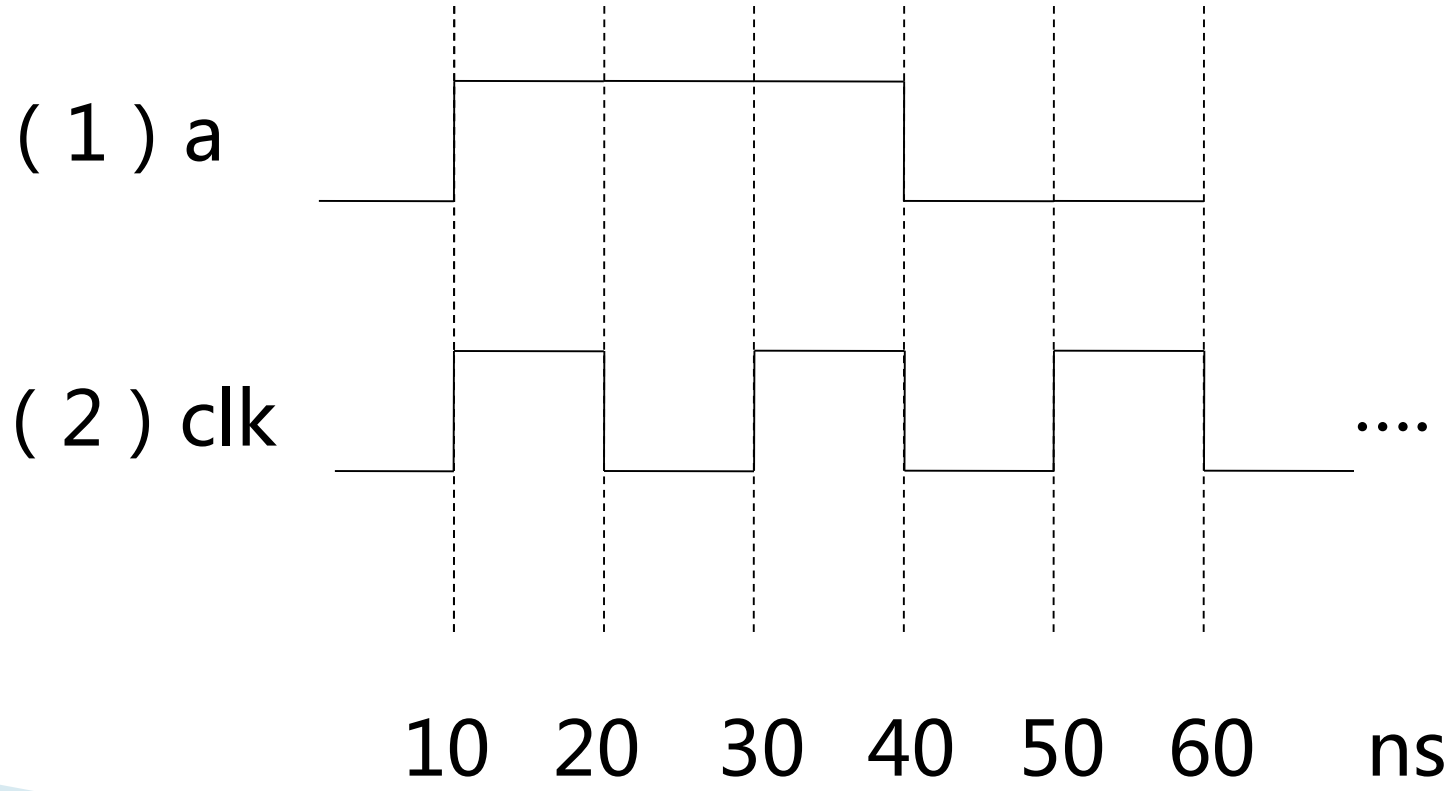
编写激励文件

- 思考：下列d的波形是怎样的？

```
initial
begin
  b = 1'b1; c = 1'b0;
  #10 d = 1'b0;
end

initial
begin
  d = #25 (b|c);
end
```

编写激励文件

- 思考：如下波形的激励文件是怎样的？



modelsim的使用范例

- 与门的时序仿真
- 分频计数器

modelsim的使用范例

- 什么是分频计数器？
- 分频计数器代码解析。

```
module div_clk(reset, clk, dclk);
    input reset, clk;
    output dclk;

reg [2:0] cnt;

assign dclk = cnt[2];

always @ (posedge clk or posedge reset)
   if(reset) cnt <= 3'd0;
   else cnt <= cnt+1'b1;

endmodule
```

modelsim的使用范例

- 分频计数器激励文件

```
module testdiv();

    reg clk, reset;
    wire dclk;

    div_clk divclk (.reset(reset),
                    .clk(clk), .dclk(dclk));

    initial
    begin
    clk = 1'b0;  reset = 1'b1;
    #22 reset = 1'b0;
    end

    always #10 clk = ~clk;

endmodule
```

作业

1、用verilog描述语言描述一个或门，并设计激励文件验证它的正确性

2、一个比较器的输入是X(8位), Y(8位)，当X>Y的时候输出Z(1位)为1， 否则输出Z(1位)为0，设计并验证这个比较器。

思考题

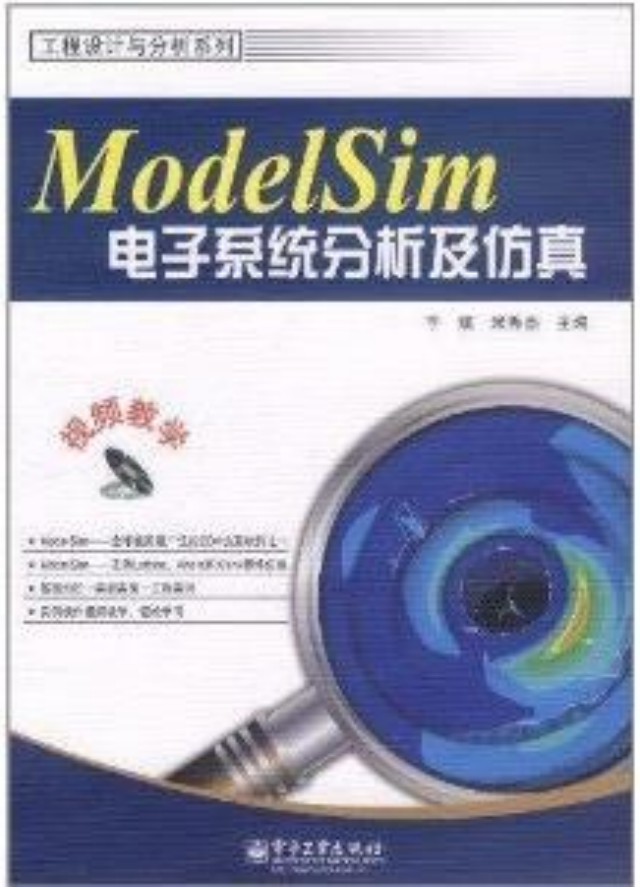
设计一个算术逻辑单元，alucontrol是一个3位的输入信号，它的值和运算类型的对应关系如下(注：r代表result，110和111分别是逻辑左移和算术右移)：

Alucontrol	运算类型	Alucontrol	运算类型
000	加法：r = x + y	100	非：r = ~x
001	减法：r = x – y	101	r = x<y ? 1:0
010	与：r = x & y	110	r = x << y[4:0]（逻辑）
011	或： r = x | y	111	r = x>> y[4:0]（算术）

输入和输出位宽均为32位。输入引脚为：x(32位), y(32位), alucontrol(3位), 输出引脚为：result(32位)。用verilog描述这个ALU，并设计testbench验证其正确性。

参考资料

- ModelSim电子系统分析及仿真
- 于斌、 米秀杰
- 电子工业出版社

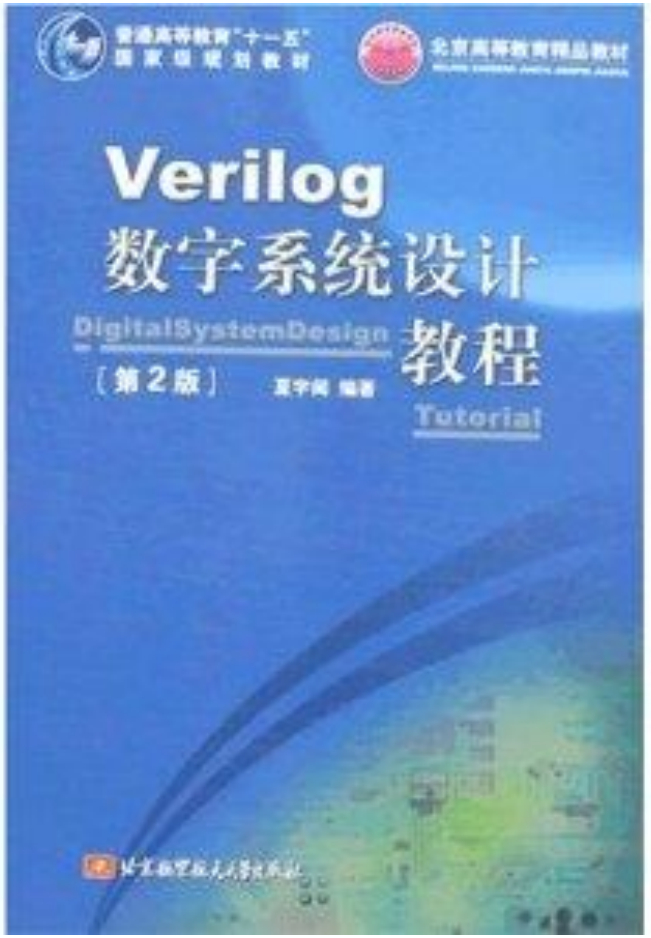


参考资料

- 数字设计和计算机体系结构
- David money harris,
- Sarah L.harris
- 陈虎 译
- 机械工业出版社

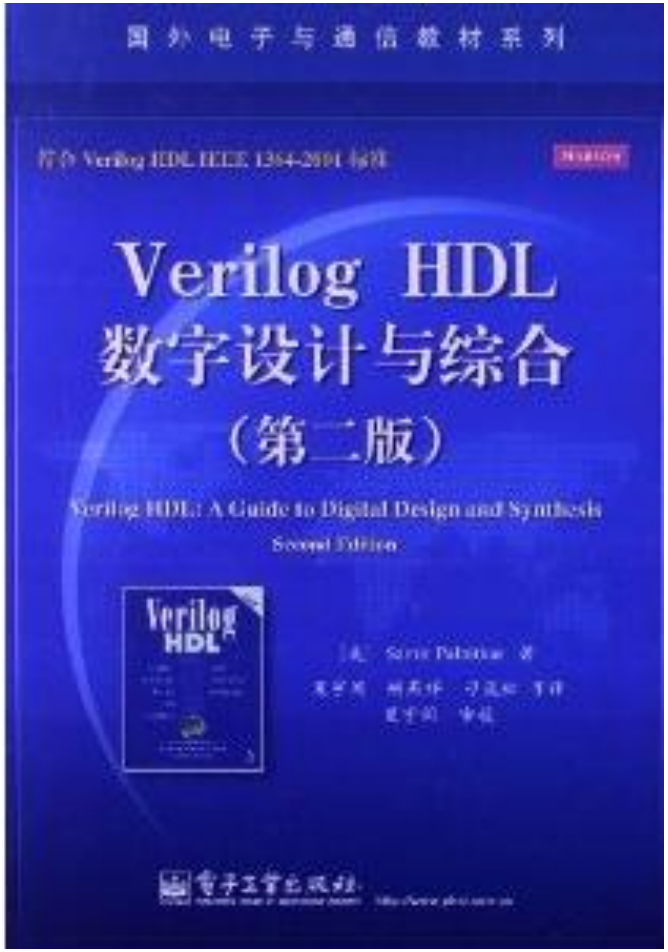
参考资料

- Verilog数字系统设计教程(第2版)
- 夏宇闻
- 北京航空航天大学出版社



参考资料

- Verilog HDL数字设计与综合(第2版)
- Samir Palnitkar
- 夏宇闻 译
- 电子工业出版社



END