# 实验3

# 简单计算机系统

# 系统设计A

# 主要内容

1、简单计算机系统实验任务简介

2、模块设计：

多路选择器，控制器

3、完善数据通路

4、动手练习：仿真验证功能

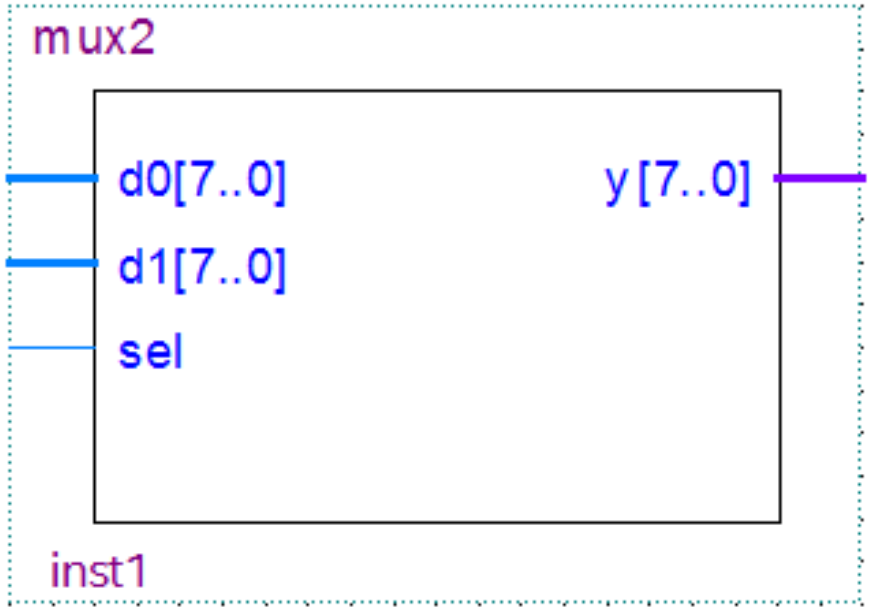
# 实验任务简介

- 实现一种简单计算机系统的设计.
  - ✓ 精简的MIPS指令集
  - ✓ EDA仿真
- 编写程序，仿真验证所设计系统的功能
  - ✓ 用汇编格式编写程序，并翻译成机器码.
  - ✓ 将机器码程序放入ROM，通过仿真验证简单计算机系统的功能.

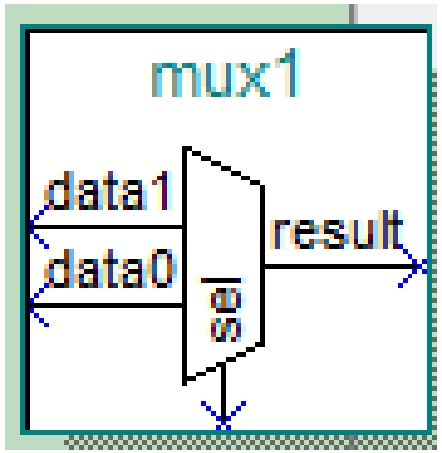
# 1、多路选择器模块设计

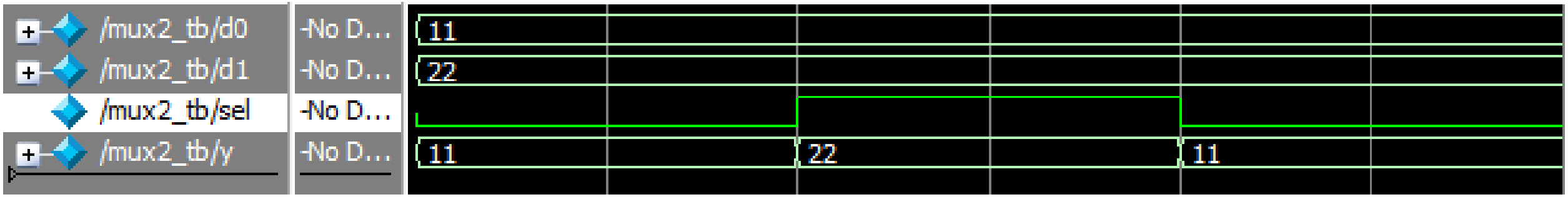
# mux2.v

```
module mux2(d0,d1,sel,y);
  input [7:0] d0;
  input [7:0] d1;
  input sel;
  output [7:0] y;
  ......
endmodule
```




# LPM_MUX



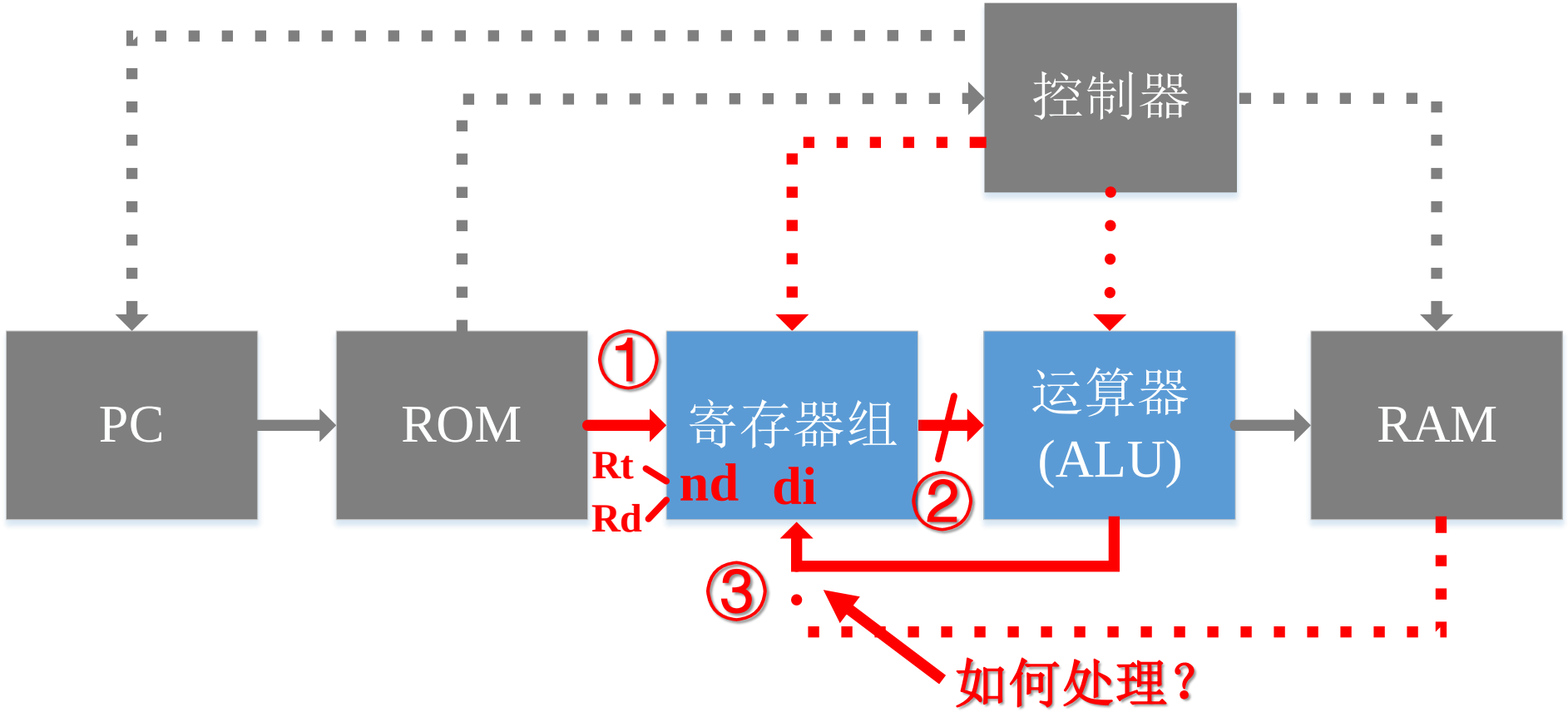



# 多路选择器模块用在哪里？

# 简单计算机系统指令集

| 操作名称 | 操作码 | 汇编语言格式指令 | 执行操作 | |
|---|---|---|---|---|
| 与 | 0000 | AND Rd, Rs, Rt | Rd←Rs and Rt; PC ← PC + 1 | R |
| 或 | 0001 | OR Rd, Rs, Rt | Rd←Rs or Rt; PC ← PC +1 | |
| 不带进位加 | 0010 | ADD Rd, Rs, Rt | Rd←Rs+ Rt; PC ← PC + 1 | |
| 不带借位减 | 0011 | SUB Rd, Rs, Rt | Rd←Rs– Rt; PC ← PC + 1 | |
| 无符号数比较 | 0100 | SLT Rd,Rs,Rt | If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1 | |
| 带借位减 | 0101 | SUBC Rd, Rs, Rt | Rd←Rs– Rt-(1-C); PC ← PC + 1 | |
| 带进位加 | 0110 | ADDC Rd, Rs, Rt | Rd←Rs+ Rt+C; PC ← PC + 1 | |
| | | | | |
| 立即数与 | 1000 | ANDI Rt, Rs, imm | Rt←Rs and imm; PC ← PC +1 | I |
| 立即数或 | 1001 | ORI Rt, Rs, imm | Rt←Rs or imm; PC ←PC +1 | |
| 立即数加 | 1010 | ADDI Rt, Rs, imm | Rt←Rs+ imm; PC ← PC +1 | |
| | | | | |
| 读存储器 | 1011 | LW Rt, Rs, imm | Rt←MEM[Rs+imm]; PC ←PC +1 | |
| 写存储器 | 1100 | SW Rt, Rs, imm | MEM[Rs+imm] ←Rt； PC ← PC +1 | |
| 相等时跳转 | 1101 | BEQ Rs, Rt, imm | If Rt=Rs, PC←PC+imm+1<br>else PC←PC+1 | |
| 不等时跳转 | 1110 | BNE Rs, Rt, imm | If Rt!=Rs, PC←PC+imm+1<br>else PC←PC+1 | |
| | | | | |
| 无条件跳转 | 0111 | JMP imm | PC ← imm | J |

# 何处需要多路选择器？




| | |
|---|---|
| ADD Rd, Rs, Rt | Rd←Rs+ Rt; PC ← PC + 1 |
| LW Rt, Rs, imm | Rt←MEM[Rs+imm]; PC ←PC +1 |
| ADDI Rt, Rs, imm | Rt←Rs+ imm; PC ← PC +1 |

# 何处需要多路选择器？

■ 指定目的寄存器号

✓ R指令时alu结果要写入Rd，I指令则要写入Rt；**需要一个多路选择器**

✓ 控制器需要输出一个选择信号**regdes**，为1时目的寄存器为Rd，为0时是Rt

■ alu第2个操作数之来源：R指令->q2，I指令->imm　　**需要一个多路选择器**

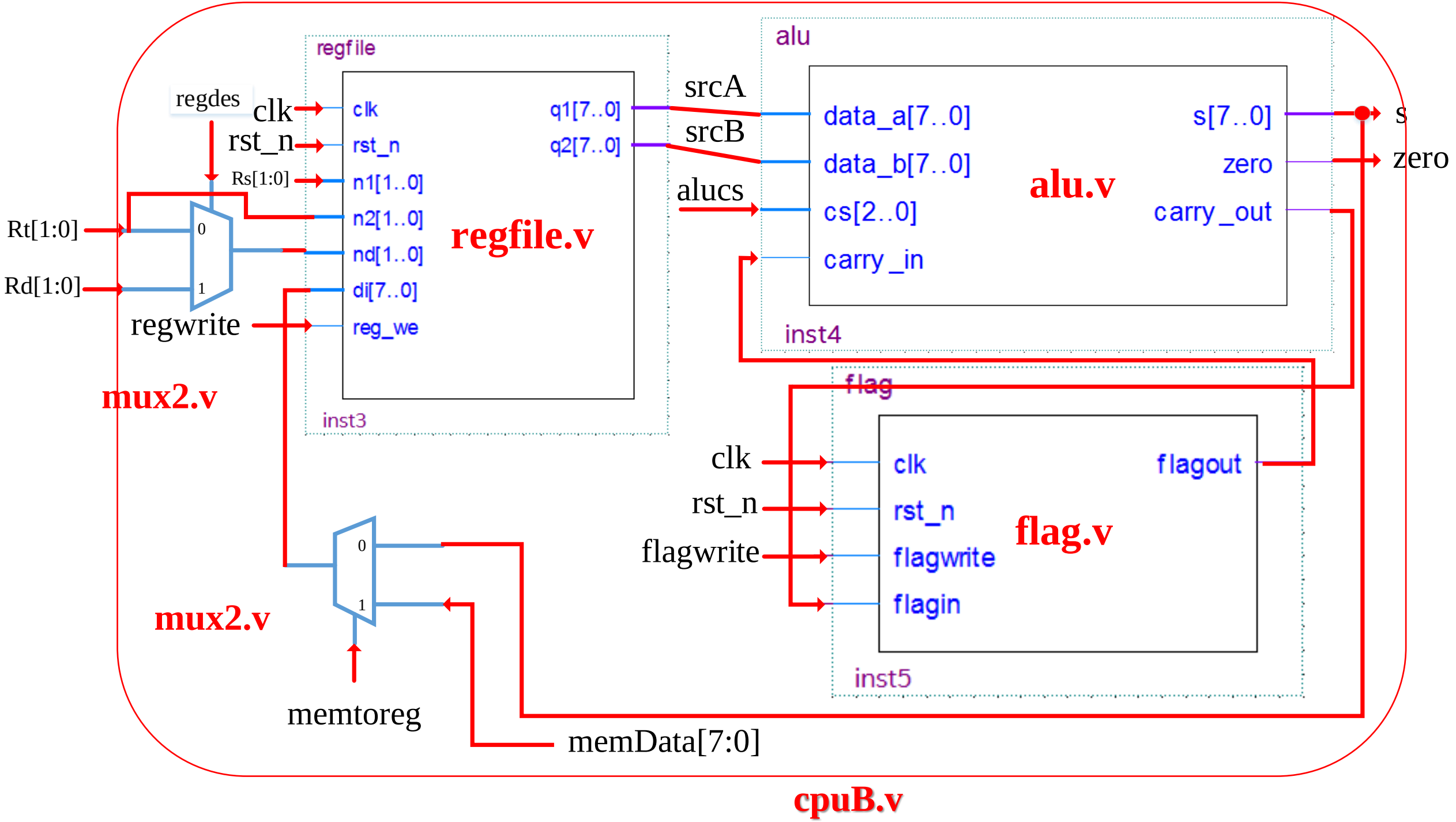
■ 写入寄存器数据之来源

✓ 来自alu：R指令等

✓ 来自RAM：LW指令

} **需要一个多路选择器**

✓ 控制器还要输出一个选择信号memtoreg，为1时来自RAM，0来自alu


regfile
regdes
clk
rst_n
Rs[1:0]
Rt[1:0]
Rd[1:0]
0
1
regwrite
mux2.v
clk
rst_n
n1[1..0]
n2[1..0]
nd[1..0]
di[7..0]
reg_we
q1[7..0]
q2[7..0]
regfile.v
inst3
srcA
srcB
alucs
alu
data_a[7..0]
data_b[7..0]
cs[2..0]
carry_in
s[7..0]
zero
carry_out
alu.v
inst4
s
zero
flag
clk
rst_n
flagwrite
flagin
flagout
flag.v
inst5
0
1
mux2.v
memtoreg
memData[7:0]
cpuB.v

# 实验任务3

任务3.1

（1）参照上面的步骤，设计、仿真mux2模块.

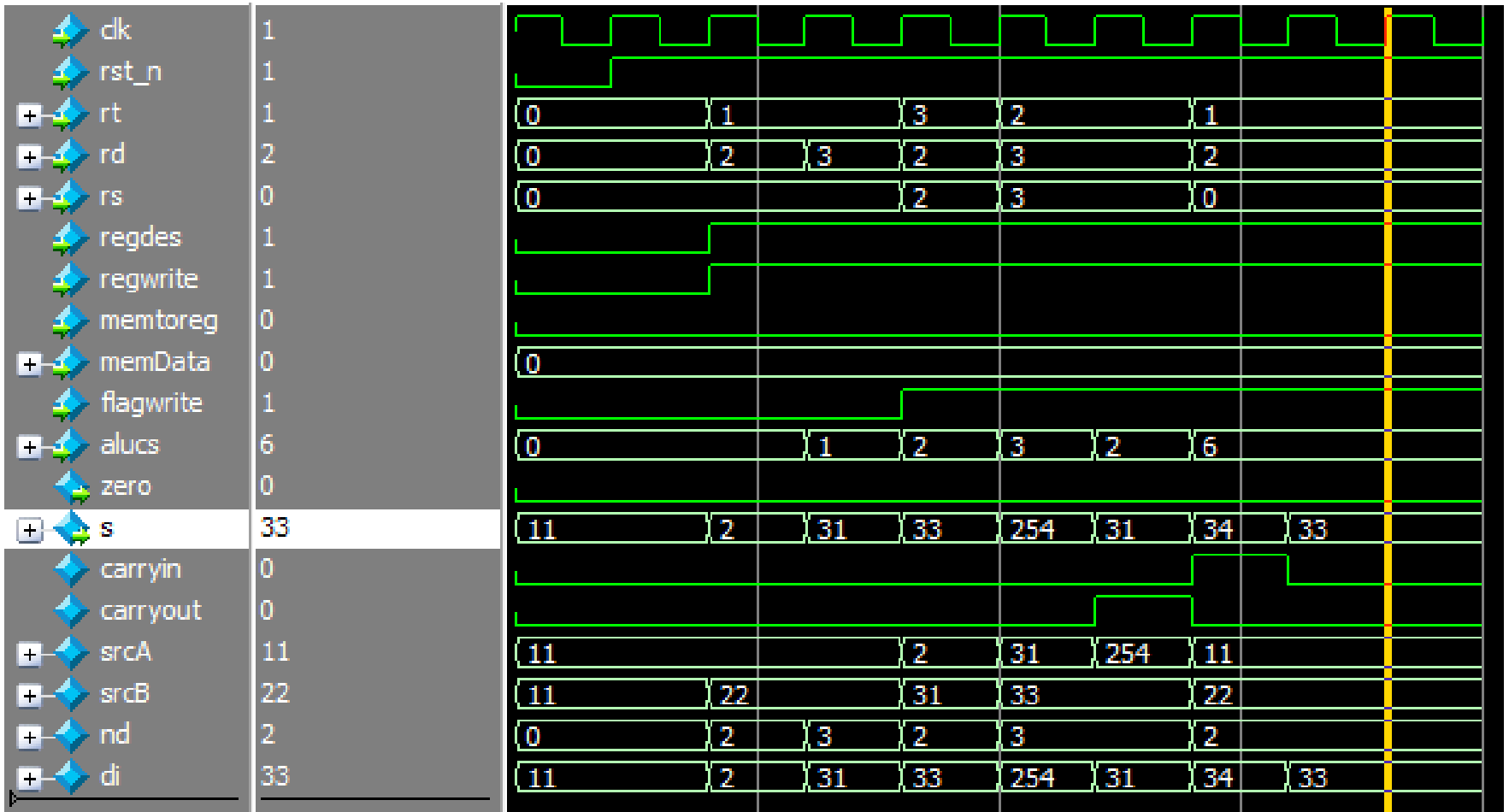
（2）编写顶层文件cpuB.v及其测试文件；通过修改regfile.v，给寄存器赋初值；在测试文件中，设置激励信号，对系统进行仿真验证；分析仿真结果.

给寄存器赋初值，如：
R0 = 11
R1 = 22

```
AND R2,R0,R1
OR R3,R0,R1
ADD R2, R2, R3
SUB R3,R3,R2
ADD R3,R3,R2
ADDC R2,R0,R1
……
```

R0 = 11
R1 = 22

AND R2,R0,R1 R2 = 2; R0=11;R1=22
OR R3,R0,R1 R3 = 31;R0=11;R1=22
ADD R2, R2, R3 R2=33;R2=2;R3=31
SUB R3,R3,R2 R3=254
ADD R3,R3,R2 R3=31
ADDC R2,R0,R1 R2=34
……

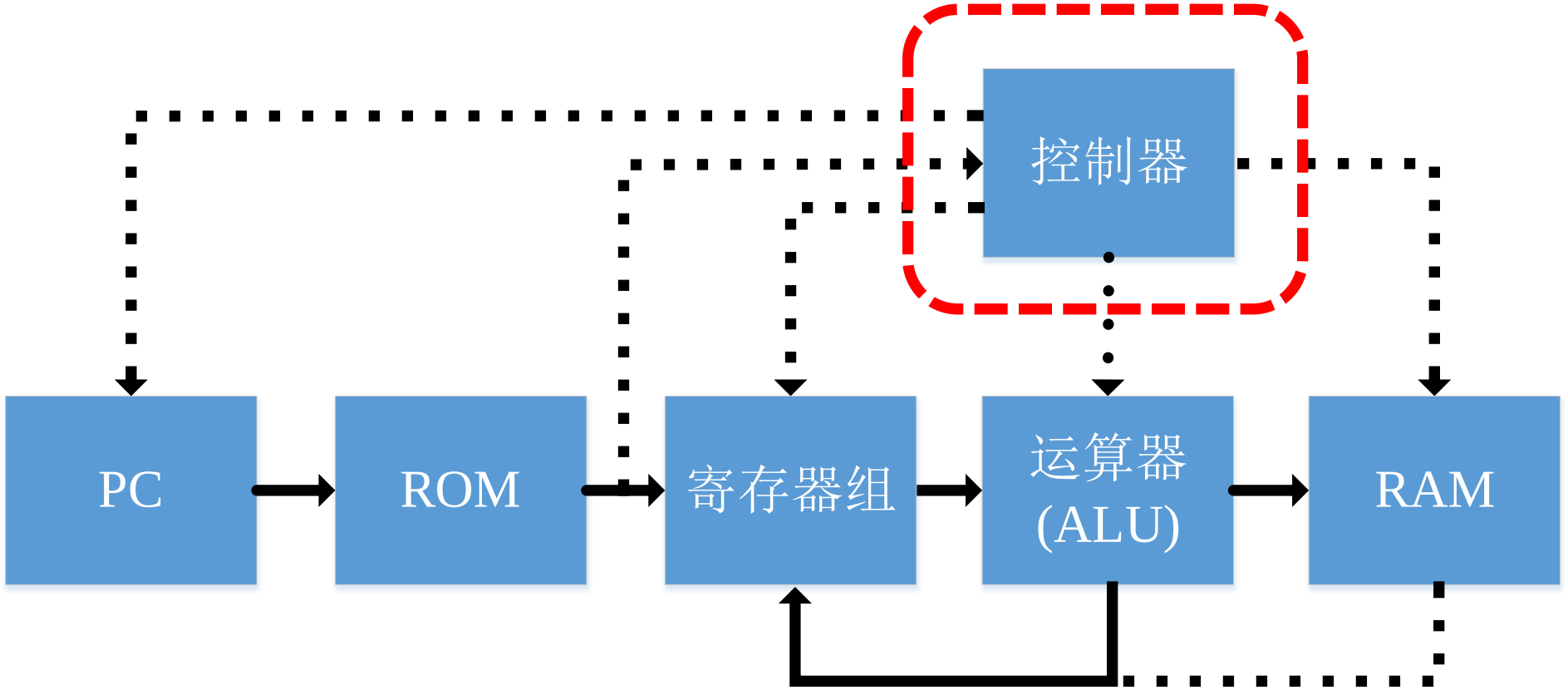



```
#20.1 rst_n = 1;
#20 alucs = 0;
    flagwrite = 0;
    rs = 0;
    rt = 1;
    rd = 2;
    memtoreg = 0;
    regdes = 1;
    regwrite = 1;
#20 alucs = 1;
    flagwrite = 0;
    rs = 0;
    rt = 1;
    rd = 3;
    memtoreg = 0;
    regdes = 1;
    regwrite = 1;
#20 alucs = 2;
    flagwrite = 1;
    rs = 2;
    rt = 3;
    rd = 2;
    regwrite = 1;
    memtoreg = 0;
    regdes = 1;

    ……
```

# 2、控制器模块设计

# 计算机模型




控制器要送出哪些信号？

# 简单计算机系统指令集

| 操作名称 | 操作码 | 汇编语言格式指令 | 执行操作 | |
|---|---|---|---|---|
| 与 | 0000 | AND Rd, Rs, Rt | Rd←Rs and Rt; PC ← PC + 1 | R |
| 或 | 0001 | OR Rd, Rs, Rt | Rd←Rs or Rt; PC ← PC +1 | |
| 不带进位加 | 0010 | ADD Rd, Rs, Rt | Rd←Rs+ Rt; PC ← PC + 1 | |
| 不带借位减 | 0011 | SUB Rd, Rs, Rt | Rd←Rs– Rt; PC ← PC + 1 | |
| 无符号数比较 | 0100 | SLT Rd,Rs,Rt | If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1 | |
| 带借位减 | 0101 | SUBC Rd, Rs, Rt | Rd←Rs– Rt-(1-C); PC ← PC + 1 | |
| 带进位加 | 0110 | ADDC Rd, Rs, Rt | Rd←Rs+ Rt+C; PC ← PC + 1 | |
| | | | | |
| 立即数与 | 1000 | ANDI Rt, Rs, imm | Rt←Rs and imm; PC ← PC +1 | I |
| 立即数或 | 1001 | ORI Rt, Rs, imm | Rt←Rs or imm; PC ←PC +1 | |
| 立即数加 | 1010 | ADDI Rt, Rs, imm | Rt←Rs+ imm; PC ← PC +1 | |
| | | | | |
| 读存储器 | 1011 | LW Rt, Rs, imm | Rt←MEM[Rs+imm]; PC ←PC +1 | |
| 写存储器 | 1100 | SW Rt, Rs, imm | MEM[Rs+imm] ←Rt； PC ← PC +1 | |
| 相等时跳转 | 1101 | BEQ Rs, Rt, imm | If Rt=Rs, PC←PC+imm+1<br>else PC←PC+1 | |
| 不等时跳转 | 1110 | BNE Rs, Rt, imm | If Rt!=Rs, PC←PC+imm+1<br>else PC←PC+1 | |
| | | | | |
| 无条件跳转 | 0111 | JMP imm | PC ← imm | J |

# R型指令编码

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | Rs | | Rt | | Rd | | --- | | | | | |

7条

| 操作名称 | 操作码(0p) | 汇编语言格式指令 | 执行操作 |
|---|---|---|---|
| 与 | 0000 | AND Rd, Rs, Rt | Rd←Rs and Rt; PC ← PC + 1 |
| 或 | 0001 | OR Rd, Rs, Rt | Rd←Rs or Rt; PC ← PC +1 |
| 加 | 0010 | ADD Rd, Rs, Rt | Rd←Rs+ Rt; PC ← PC + 1 |
| 减 | 0011 | SUB Rd, Rs, Rt | Rd←Rs– Rt; PC ← PC + 1 |
| 带进位加 | 0110 | ADDC Rd, Rs, Rt | Rd←Rs+ Rt+C; PC ← PC + 1 |
| 带借位减 | 0101 | SUBC Rd, Rs, Rt | Rd←Rs– Rt-(1-C); PC ← PC + 1 |
| 无符号数比较 | 0100 | SLT Rd,Rs, Rt | If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1 |

# R型指令编码

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | Rs | | Rt | | Rd | | --- | | | | | |

■ R型指令有什么特点？

✓ 3个操作数

✓ 操作数均为寄存器

✓ 均要用到alu

✓ alu计算结果均要写入寄存器

| 操作名称 | 操作码(Op) | 汇编语言格式指令 | 执行操作 |
|---|---|---|---|
| 与 | 0000 | AND Rd, Rs, Rt | Rd←Rs and Rt; PC ← PC + 1 |
| 或 | 0001 | OR Rd, Rs, Rt | Rd←Rs or Rt; PC ← PC +1 |
| 加 | 0010 | ADD Rd, Rs, Rt | Rd←Rs+ Rt; PC ← PC + 1 |
| 减 | 0011 | SUB Rd, Rs, Rt | Rd←Rs– Rt; PC ← PC + 1 |
| 带进位加 | 0110 | ADDC Rd, Rs, Rt | Rd←Rs+ Rt+C; PC ← PC + 1 |
| 带借位减 | 0101 | SUBC Rd, Rs, Rt | Rd←Rs– Rt-(1-C); PC ← PC + 1 |
| 无符号数比较 | 0100 | SLT Rd,Rs, Rt | If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1 |

■ 与控制器相关的信号

✓ 指令中的Op送给控制器模块

✓ 通知alu做相应运算，alu的cs[2:0]

✓ 送出写寄存器组信号

# R型指令编码

■ R型指令的特点

✓ 3个操作数

✓ 操作数均为寄存器

✓ 均要用到alu

✓ alu计算结果均要写入寄存器

**需要控制器送出何种信号？**

■ 与控制器相关的信号

✓ 指令中的Op送给控制器模块

✓ 通知alu做相应运算，alu的cs[2:0]

✓ 给出写寄存器组的信号

R指令时，控制器的信号

■ 输入信号：Op

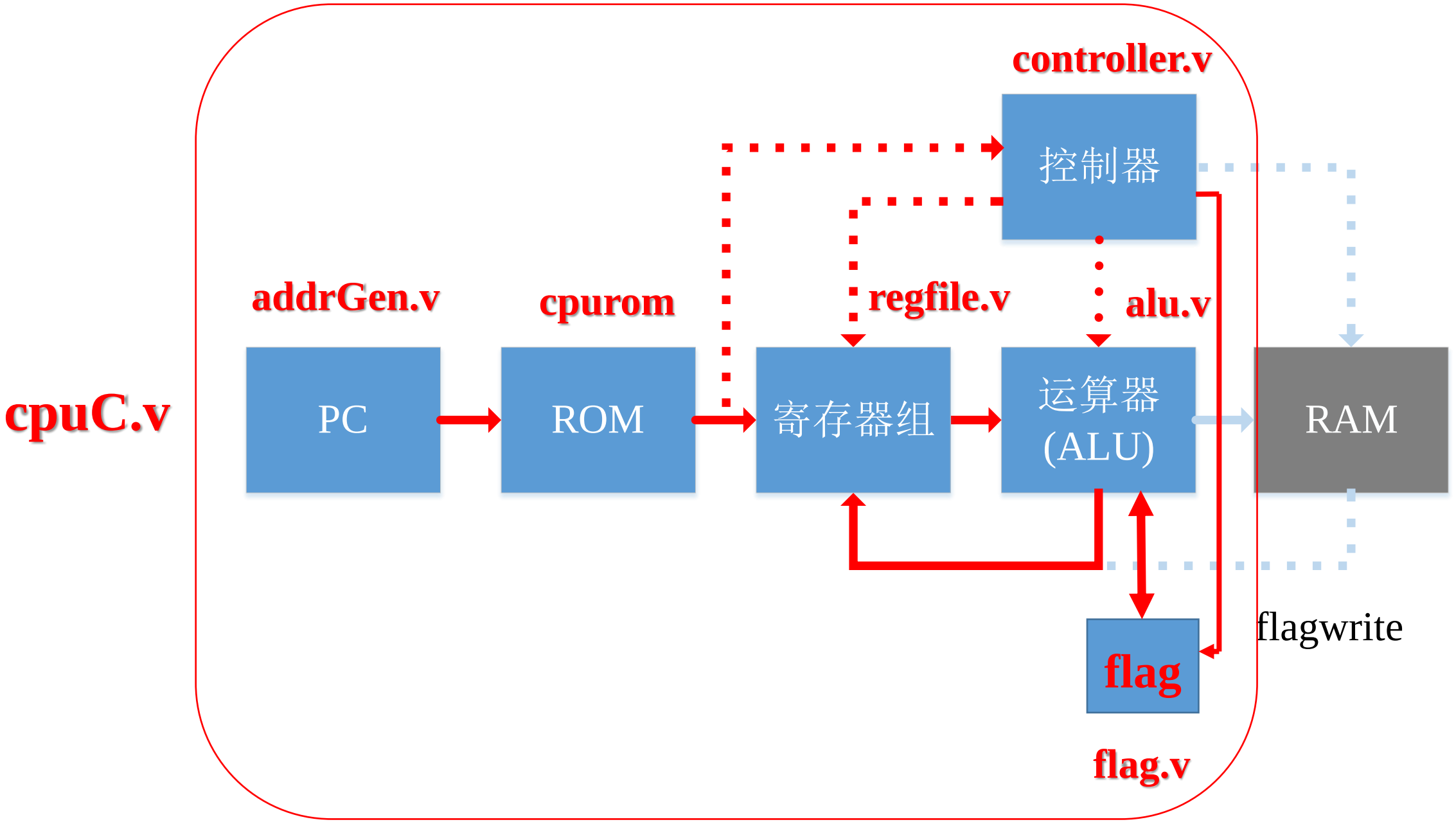
■ 输出信号：alucs[2:0]，regwrite……

```
module controller(op,alucs,regwrite);
  input [3:0] op;
  output reg [2:0] alucs;
  output reg regwrite;
  ……
  always @ (*)
    begin
      ……
    end
endmodule
```

**controller.v**

| 操作名称 | 操作码(Op) | 汇编语言格式指令 |
|---|---|---|
| 与 | 0000 | **AND Rd, Rs, Rt** |
| 或 | 0001 | **OR Rd, Rs, Rt** |
| 加 | 0010 | **ADD Rd, Rs, Rt** |
| 减 | 0011 | **SUB Rd, Rs, Rt** |
| 带进位加 | 0110 | **ADDC Rd, Rs, Rt** |
| 带借位减 | 0101 | **SUBC Rd, Rs, Rt** |
| 无符号数比较 | 0100 | SLT Rd,Rs, Rt |

| cs[2..0] | 运算名称 |
|---|---|
| **000** | 与 |
| **001** | 或 |
| **010** | 不带进位加 |
| **011** | 不带借位减 |
| **110** | 带进位加 |
| **101** | 带借位减 |
| **100** | 比较 |


cpuC.v
controller.v
控制器
addrGen.v
cpurom
regfile.v
alu.v
PC
ROM
寄存器组
运算器
(ALU)
RAM
flagwrite
flag
flag.v

# R型指令编码

■ R型指令的特点

✓ 3个操作数

✓ 操作数均为寄存器

✓ 均要用到alu

✓ alu计算结果均要写入寄存器

**需要控制器送出何种信号？**

■ 与控制器相关的信号

✓ 指令中的Op送给控制器模块

✓ 通知alu做相应运算，alu的cs[2:0]

✓ 给出写寄存器组的信号

R指令时，控制器的信号

■ 输入信号：Op

■ 输出信号：alucs[2:0]，regwrite，flagwrite

## 给寄存器赋初值：R0=11, R1=22

## regfile.v

```
……
if (!rst_n)
  begin
    rf[0] <= 11;
    rf[1] <= 12;
    rf[2] <= 0;
    rf[3] <= 0;
  end
else
……
```

| 行号 | 指令代码 | 机器码 |
|---|---|---|
| 0 | AND R2,R0,R1 | |
| 1 | OR R3,R0,R1 | |
| 2 | ADD R2,R2,R3 | |
| 3 | SUB R3,R3,R2 | |
| 4 | ADD R3,R3,R2 | |
| 5 | ADDC R2,R0,R1 | |
| 6 | SUB R1,R3,R2 | |
| 7 | SUBC R2,R3, R2 | |
| 8 | SLT R2,R1,R0 | |
| 9 | SLT R3,R0,R1 | |
| 10 | SUB R2,R0, R1 | |
| 11 | SUBC R3,R0, R1 | |

## R型指令二进制编码结构

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | Rs | | Rt | | Rd | | --- | | | | | |

## I型指令编码

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | Rs | | Rt | | Imm | | | | | | | |

## J指令编码

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | ---- | | | | Imm | | | | | | | |

# R型指令二进制编码结构

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | Rs | | Rt | | Rd | | --- | | | | | |

Op：指令操作码；

Rs：第1个源操作数的寄存器号；

Rt：第2个源操作数的寄存器号；

Rd：目的操作数的寄存器号；

---：表示任意值，编码时通常取0

OR　Rd, Rs, Rt　　**OR　R3, R1, R2**　　Rs=01，Rt=10，Rd=11

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op | | | | Rs | | Rt | | Rd | | ----- | | | | | |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

0001 0110 1100 0000B，以十六进制形式表示为0x16C0

| 行号 | 指令代码 | 机器码 |
| --- | --- | --- |
| 0 | AND R2,R0,R1 | |
| 1 | OR R3,R0,R1 | |
| 2 | ADD R2,R2,R3 | |
| 3 | SUB R3,R3,R2 | |
| 4 | ADD R3,R3,R2 | |
| 5 | ADDC R2,R0,R1 | |
| 6 | SUB R1,R3,R2 | |
| 7 | SUBC R2,R3, R2 | |
| 8 | SLT R2,R1,R0 | |
| 9 | SLT R3,R0,R1 | |
| 10 | SUB R2,R0, R1 | |
| 11 | SUBC R3,R0, R1 | |

# 实验任务3

任务3.2

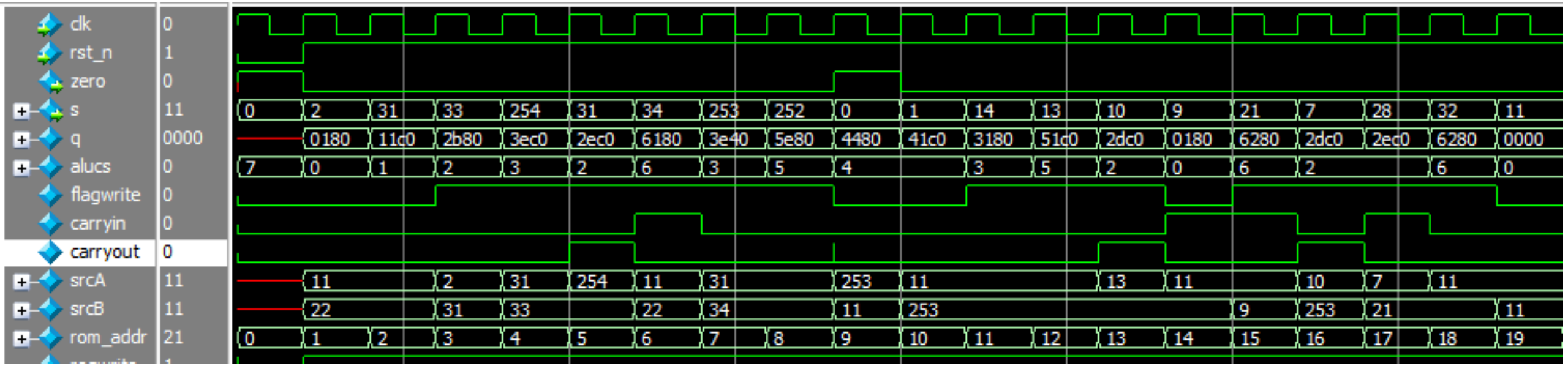
（1）参照上面的步骤，实现可以执行7条R指令的控制器模块，给出仿真结果，分析验证功能.

（2）用addrGen.v/cpurom IP核/regfile.v/alu.v/flag.v/controller.v，构建可以执行7条R指令的数据通路. **cpuC.v**

（3）分析上述12行指令构成的代码段；将它们翻译成机器码，写入ROM数据文件中；编译；仿真.

（4）分析仿真结果，**必要时修改相关模块**.

```
initial begin
  clk = 1;
  rst_n = 0;
  #20 rst_n = 1;
end
```



THE END