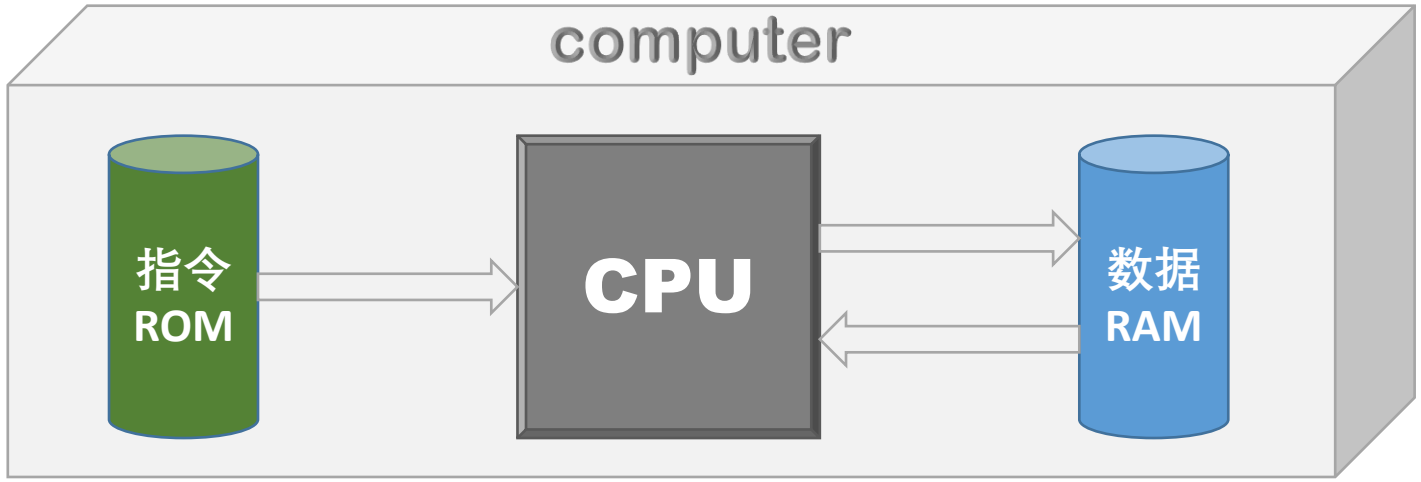
13. 数字系统设计



xgsun@fudan.edu.cn
孙晓光

2022-12-20

数字系统

- 由许多基本的逻辑功能部件有机连接起来完成某种任务的数字电子系统。

  可大可小、可简可繁

- 交互式、以离散形式表示的具有**存储**、**传输**、**处理**信息能力的逻辑子系统集合物。

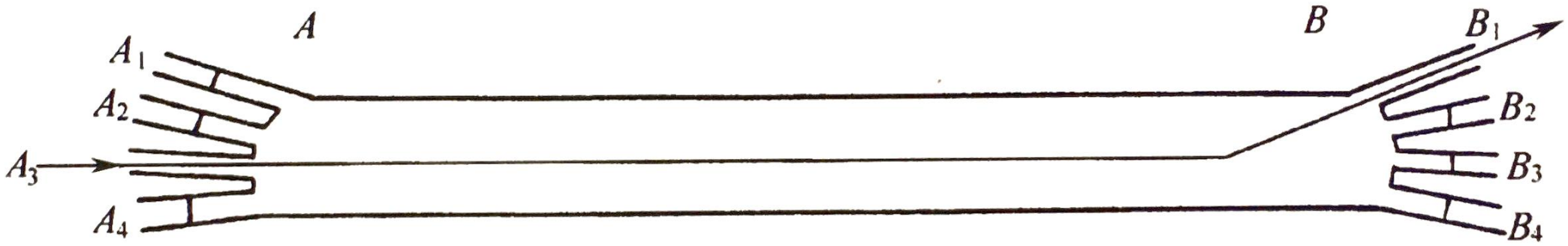
  一台数字计算机是一个最完整的数字系统。

有本质区别

数字系统	**数字逻辑功能部件**
有**控制部件**	无**控制部件**
自上而下设计过程	自下而上设计过程

总线 Bus

多个信息源**分时**传送数据流到多个目的地的传输通路。

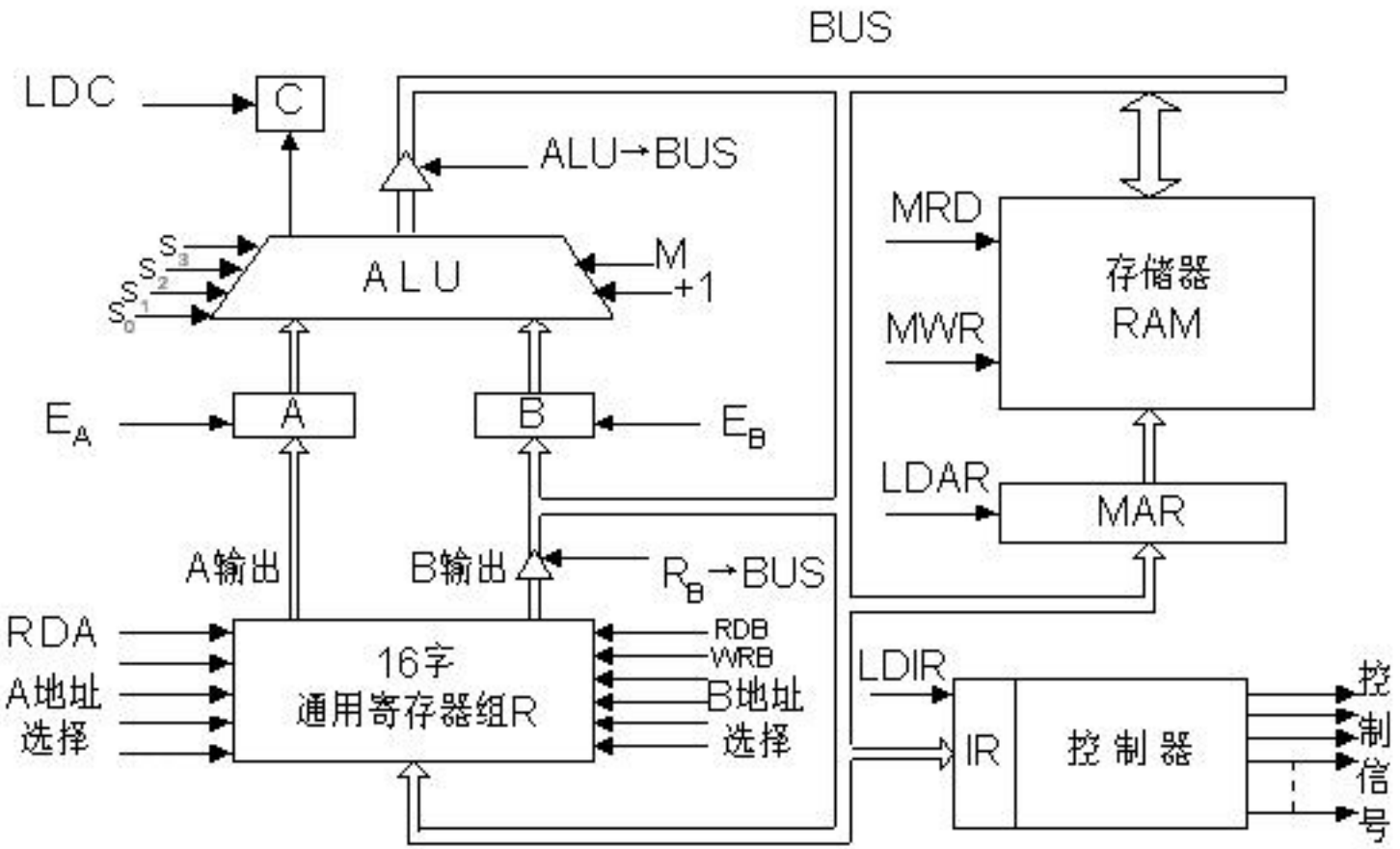


如果一组导线只连接一个信息源和一个负载，就不能称为总线。

- **总线**：计算机各种功能部件之间传送信息的公共通信干线。
- 单向总线、双向总线
- **数据总线**、**地址总线**和**控制总线**。

数据通路 datapath

各个子系统通过数据总线联结形成的**数据传送路径。**



控制器 Controller

- 整个数字系统的中心环节。
- 其控制程序：可直接由硬件实现(与一般时序电路并无区别)，也可固化的控制软件。

- 不必过分追求状态最简：控制器成本只占总成本很小一部分，
  而其性能对整个系统的工作有举足轻重的影响。
- 有时，增加一些多余状态，会使系统工作更加直观，便于监视和故障检查。
- 为了使控制状态单纯而明确，可采用“一对一”法设置触发器，
  即一个状态设一个触发器，避免状态分配的麻烦。

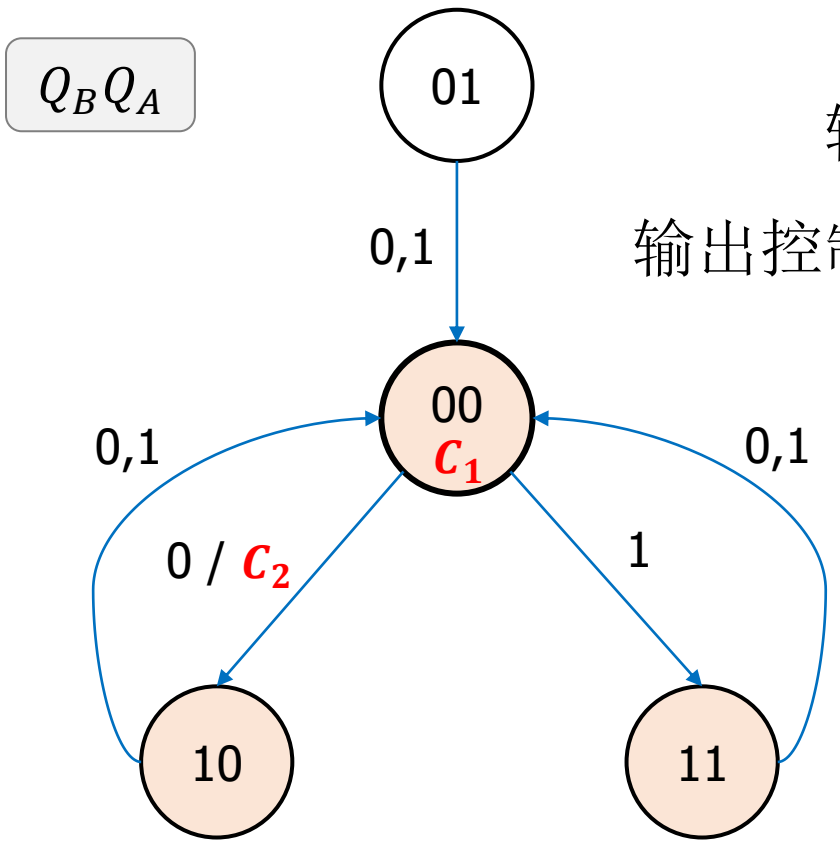
解放战争三大战役:

西柏坡发出**408**封电报

“我们这个指挥部可能是世界上最小的指挥部，一不发人，二不发枪，三不发粮，只是天天发电报，就把国民党打败了。”

周恩来

控制器的设计

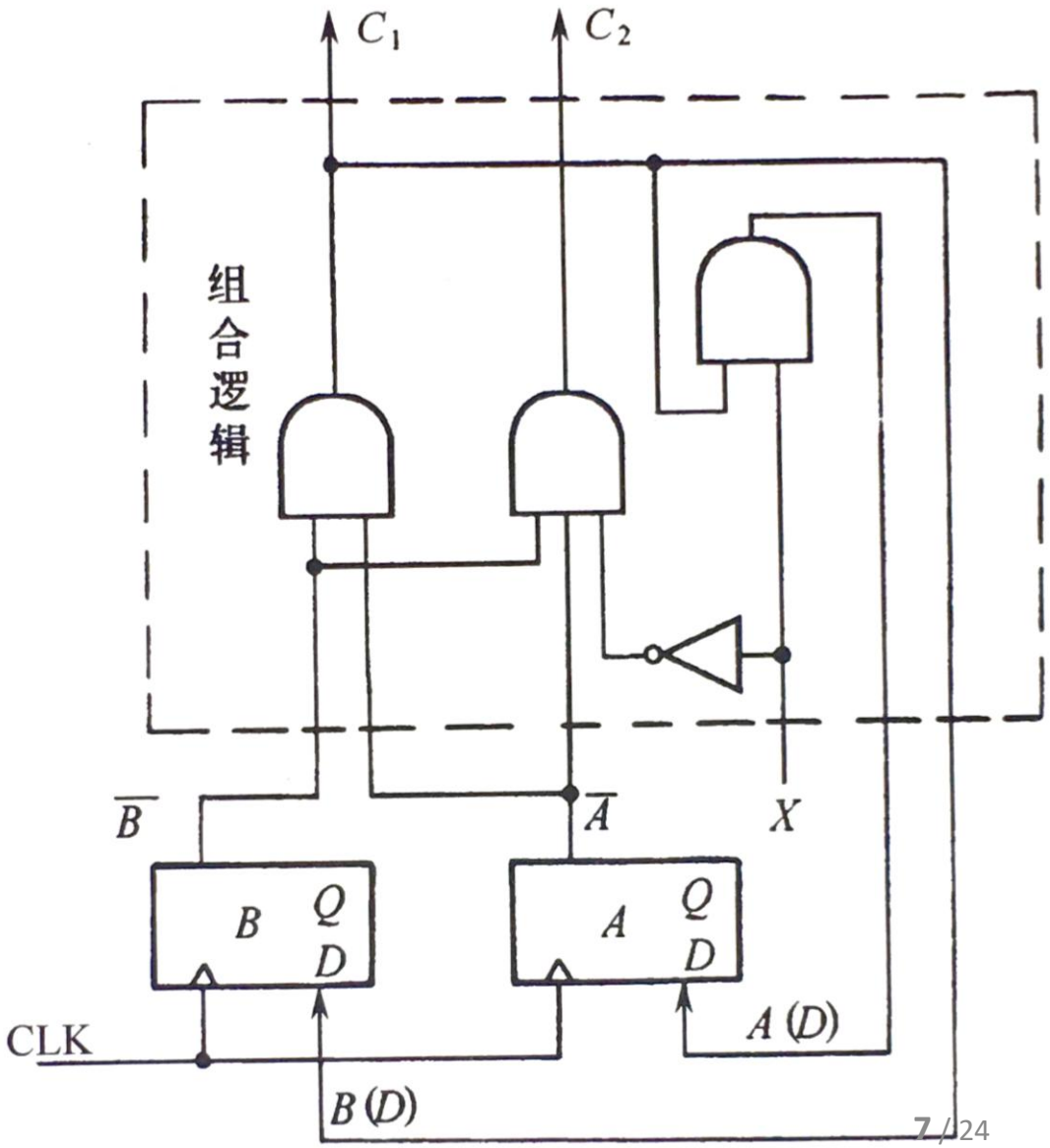


输入：X

输出控制信号：C_1, C_2

$$D_A = \bar{A}\bar{B}X \qquad C_1 = \bar{A}\bar{B}$$

$$D_B = \bar{A}\bar{B} \qquad C_2 = \bar{A}\bar{B}\bar{X}$$



micro computer

CPU2

三种微体系结构：单周期、多周期、流水线

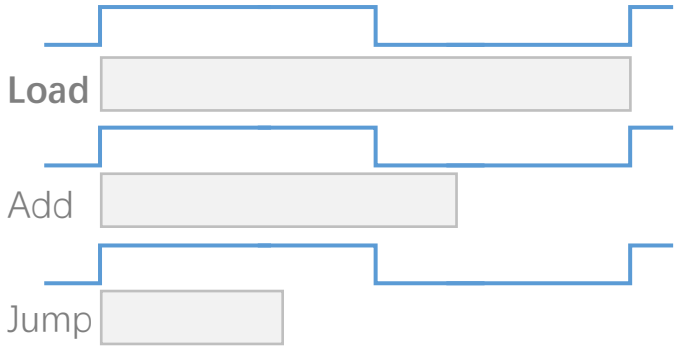
在性能、成本、复杂度之间折中

① 单周期

在一个时钟周期内执行完一条完整指令(CPI=1)。

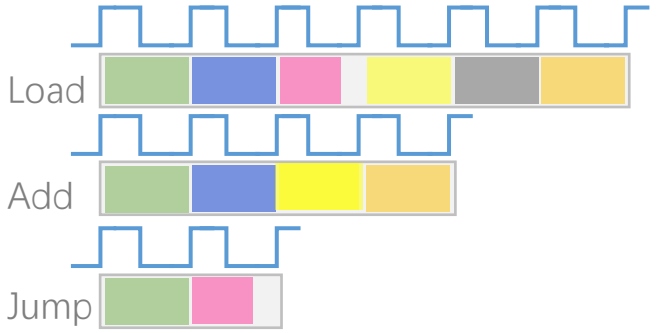
时钟周期以最长的指令(如Load指令)所花的时间为准。

控制简单，但速度慢、成本高。　　已不用



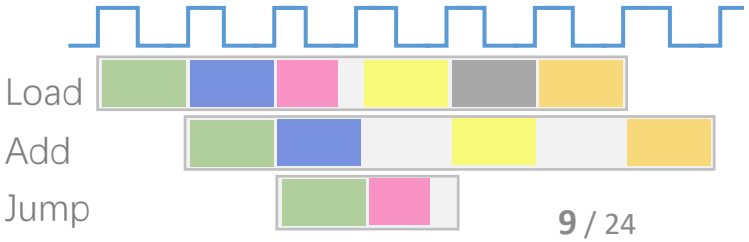
② 多周期

用多个时钟周期执行一条指令。



③ 流水线

多条指令重叠执行。显著提高吞吐量。

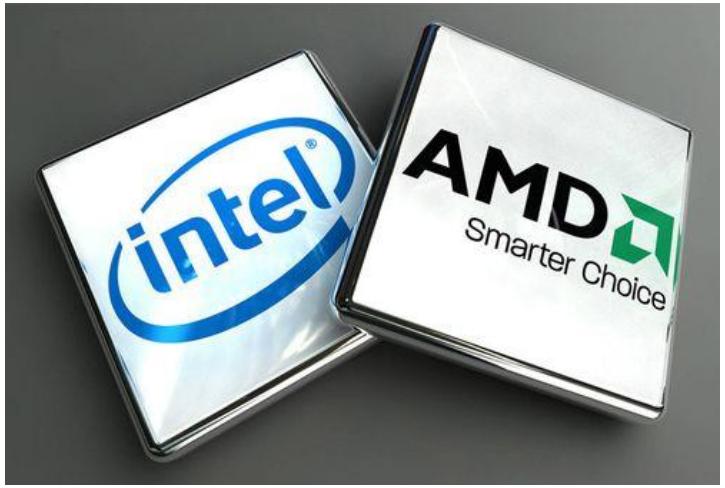


复杂指令集计算机 CISC vs RISC 精简指令集计算机

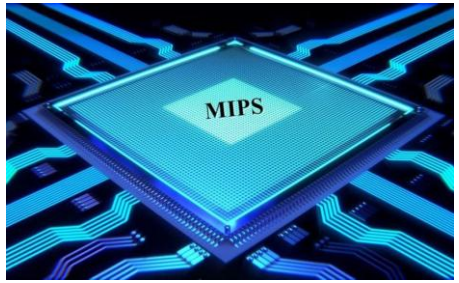
Complex **I**nstruction **S**et **C**omputer

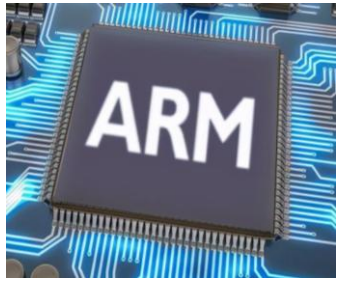
Reduced **I**nstruction **S**et **C**omputer

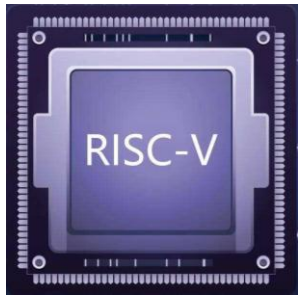
- 采用**复杂的指令系统**
- 指令数量多，功能复杂
- 指令长度可变，指令格式多样
- 寻址方式多



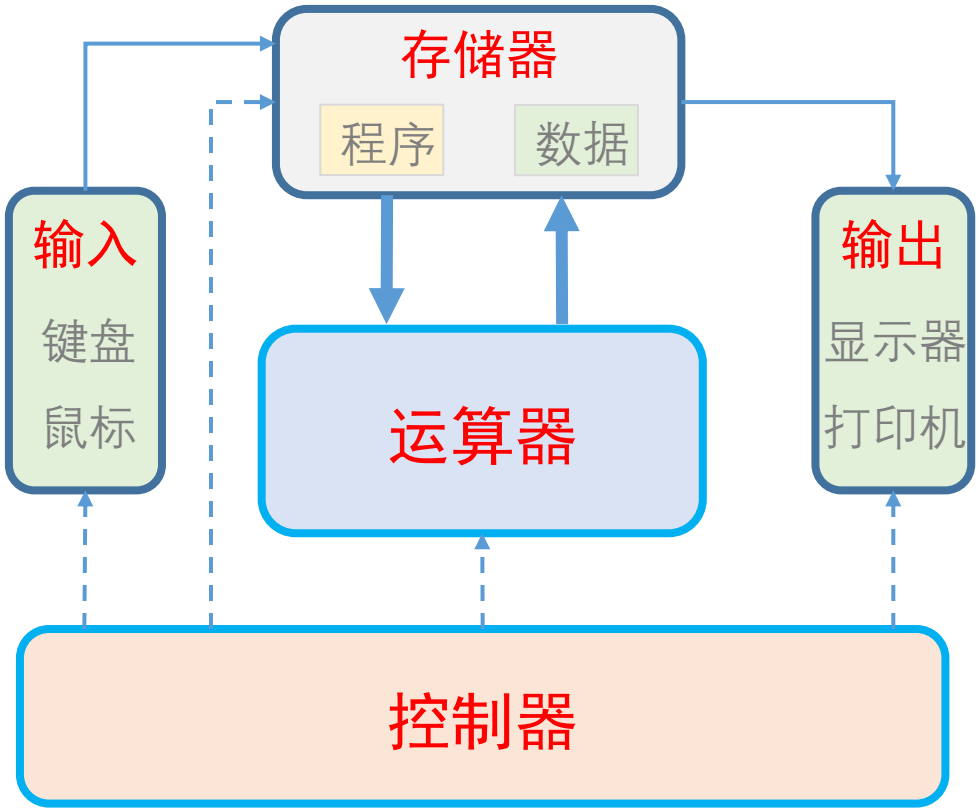
- 采用**简化的指令系统**
- 指令集只包含常用的指令
- 提供大量通用寄存器， 少访问内存
- 只有**Load**和**Store**指令才能访问内存



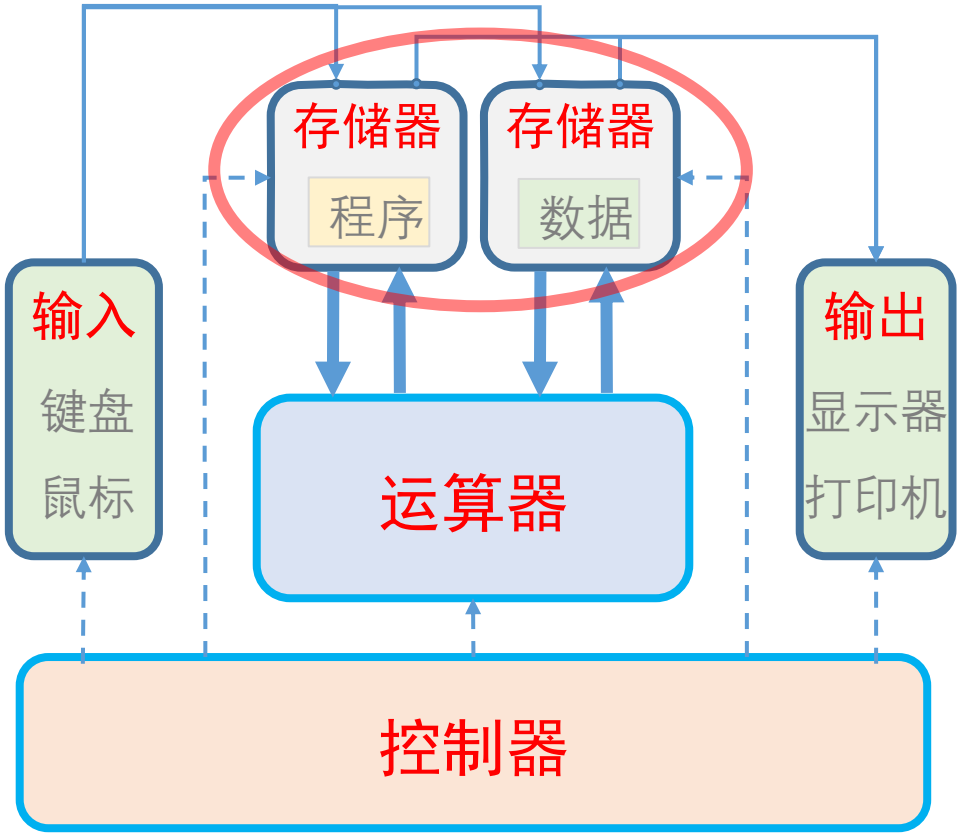




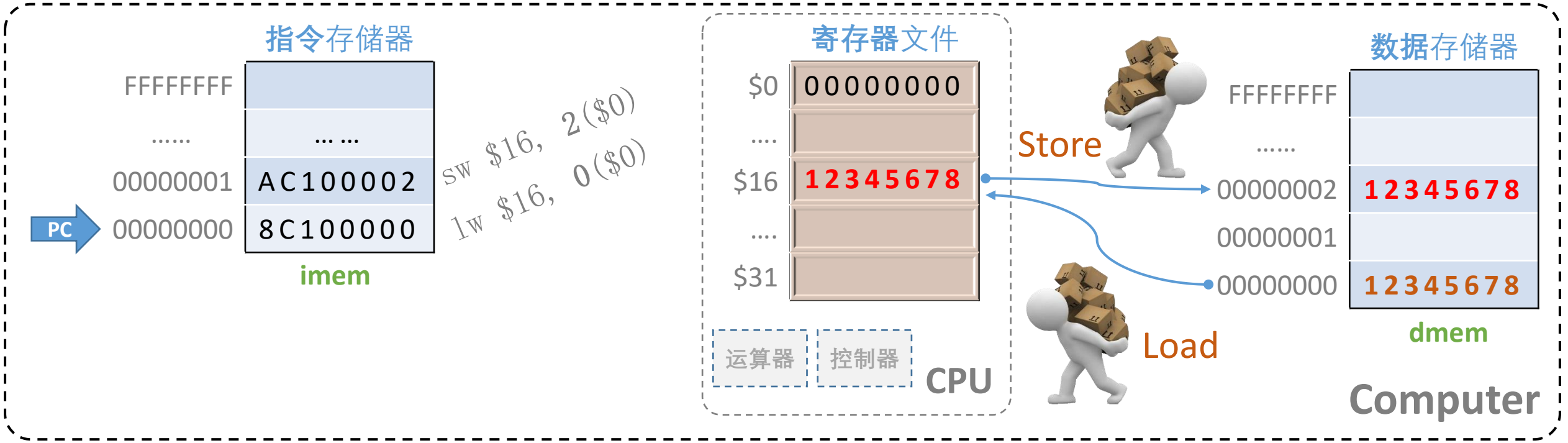
冯·诺伊曼模型


存储器
程序
数据
输入
键盘
鼠标
运算器
输出
显示器
打印机
控制器

哈佛模型


存储器
程序
存储器
数据
输入
键盘
鼠标
运算器
输出
显示器
打印机
控制器

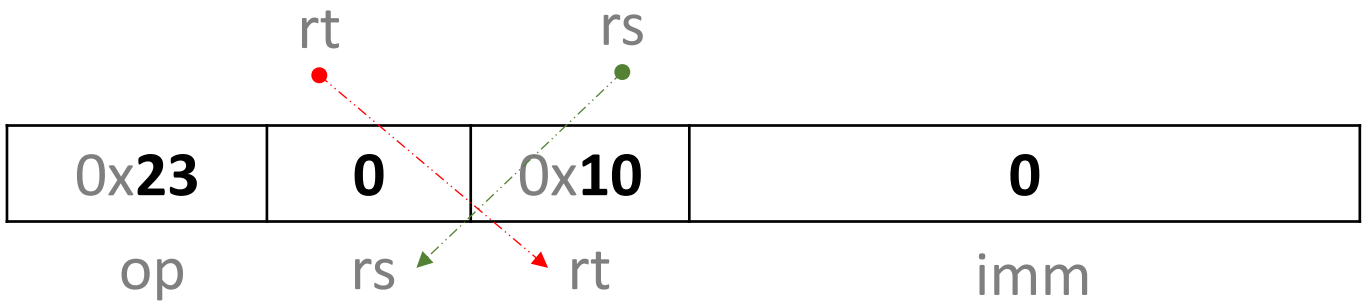
“三大件”




```
1  // 只能运行2条指令的计算机
2  module Computer(input logic  clk, reset);
3      logic memWrite;
4      logic [31:0] pc, instr, readData;
5      logic [31:0] ALUout,      writeData;
6
7      CPU C2(clk, reset,    instr, readData,
8             pc, memWrite, ALUout, writeData);
9      iROM imem(pc, instr);
10     dRAM dmem(clk, memWrite, ALUout, writeData, readData);
11 endmodule
```

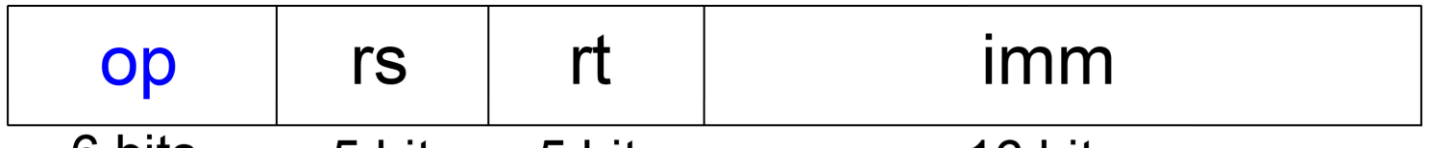
Load word　lw 指令

汇编代码: `lw $16, 0($0)`



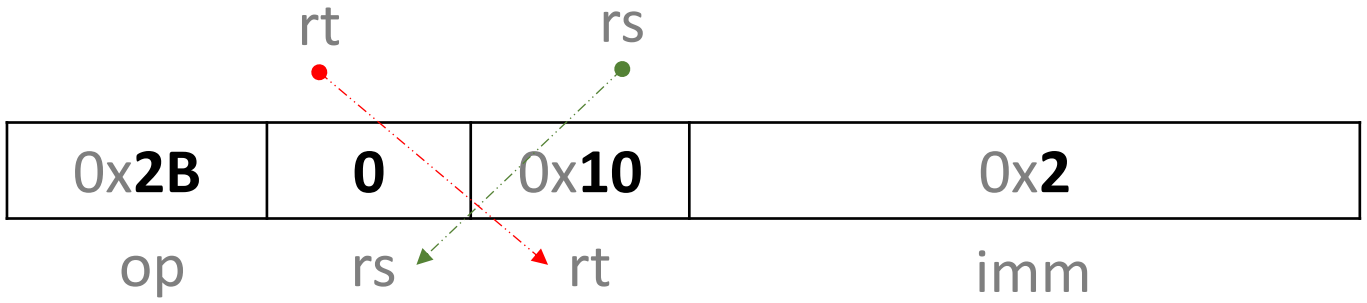
机器指令: **0x8C10_0000**

Store word　sw 指令



操作码　寄存器　立即数

汇编代码: `sw $16, 2($0)`



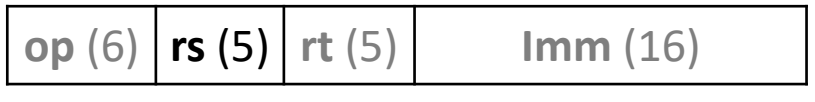
机器代码:

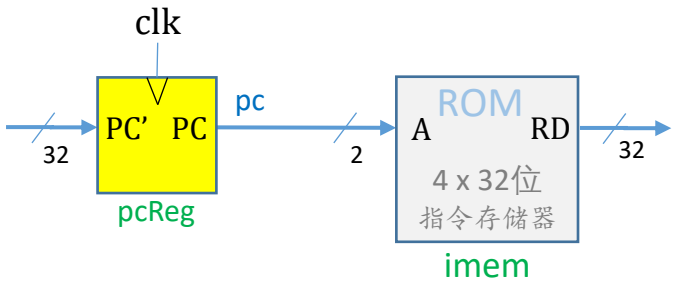
机器指令: 0xAC10_0002

lw rt, imm(rs)

STEP 1: 从指令存储器中取出指令

100011



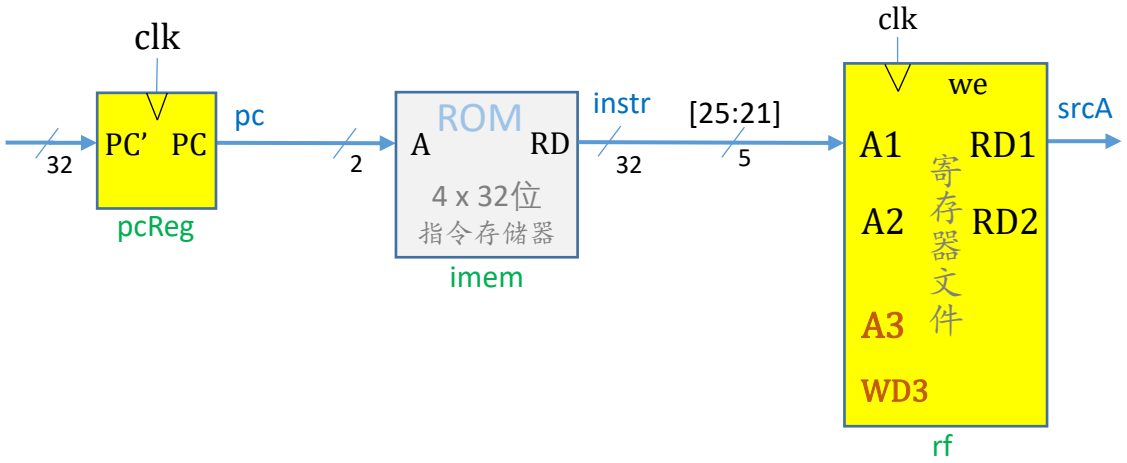



```
1  // PC 寄存器
2  module PC_register #(parameter WIDTH = 32)
3          (input  logic             clk, reset,
4           input  logic [WIDTH-1:0] d,
5           output logic [WIDTH-1:0] q);
6
7      always_ff @(posedge clk, posedge reset)
8          if (reset) q <= 0;
9          else       q <= d;
10 endmodule
```

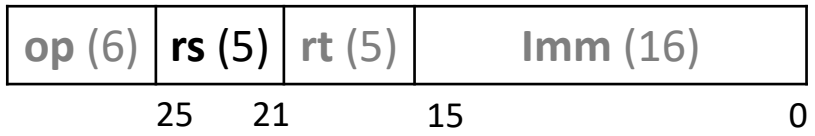
```
1  // 指令只读寄存器
2  module iROM(
3      input  logic [ 1:0] addr,
4      output logic [31:0] readData );
5
6      //     位宽      字宽
7      logic [31:0] ROM [3:0]; // 4x32bit
8
9      // initialize memory
10     initial
11     begin
12         ROM[0] = 32'h8C10_0000; // lw  $16, 0($0)
13         ROM[1] = 32'hAC10_0002; // sw  $16, 2($0)
14     end
15
16     assign readData = ROM[addr];
17 endmodule
```

lw rt, imm(rs)

STEP 2: 从寄存器文件中读出源操作数 rs



100011




```
1  // 寄存器文件
2  module registerFile(
3      input  logic        clk, we,
4      input  logic [4:0]  writeAddr3,
5      input  logic [31:0] writeData3,
6      input  logic [4:0]  readAddr1,
7      input  logic [4:0]  readAddr2,
8      output logic [31:0] readData1,
9      output logic [31:0] readData2);
10
11     //    位宽      字宽
12     logic [31:0] rf [31:0]; //32个32位寄存器
13
14     // register 0 hardwired to 0.
15     assign readData1 = (readAddr1==0) ? 0 : rf[readAddr1];
16     assign readData2 = (readAddr2==0) ? 0 : rf[readAddr2];
17
18     always_ff @(posedge clk)
19         if (we) rf[writeAddr3] <= writeData3;
20 endmodule
```

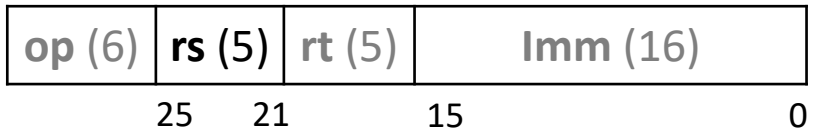
一写

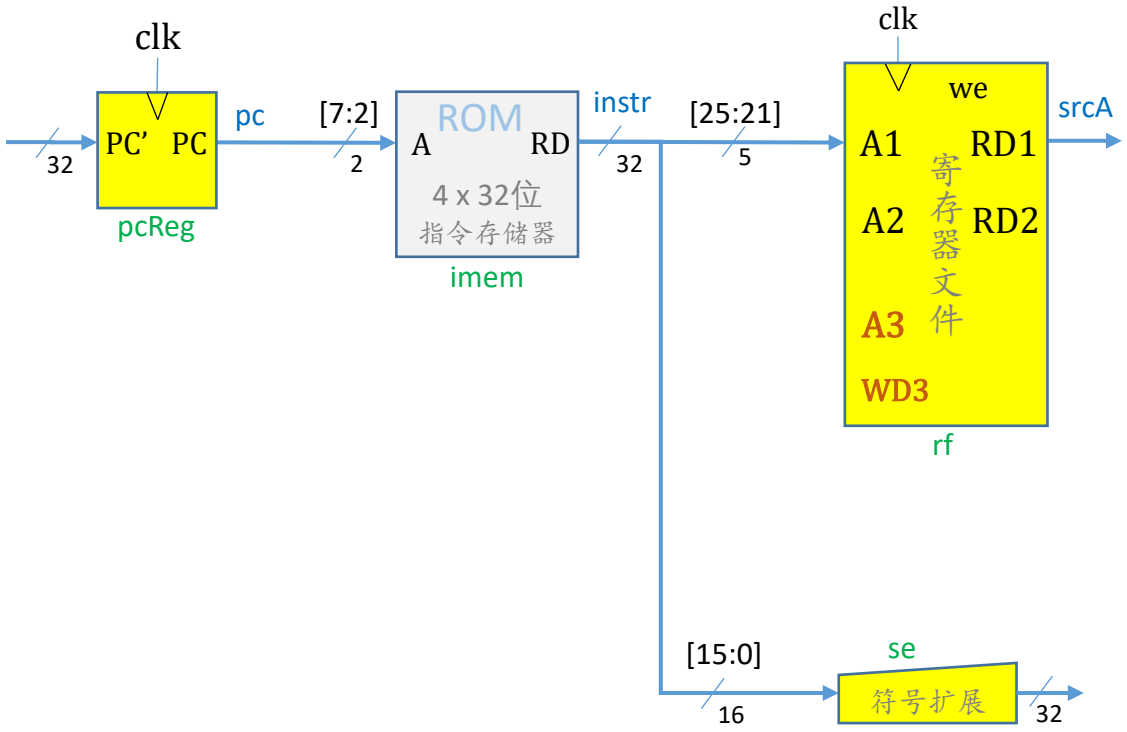
两读

lw rt, imm(rs)

STEP 3: 符号扩展立即数

100011

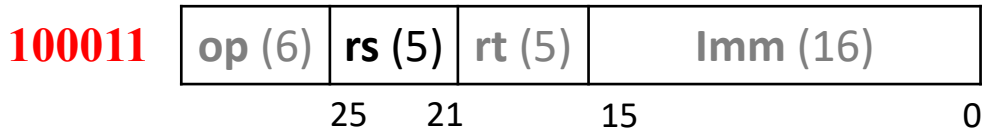


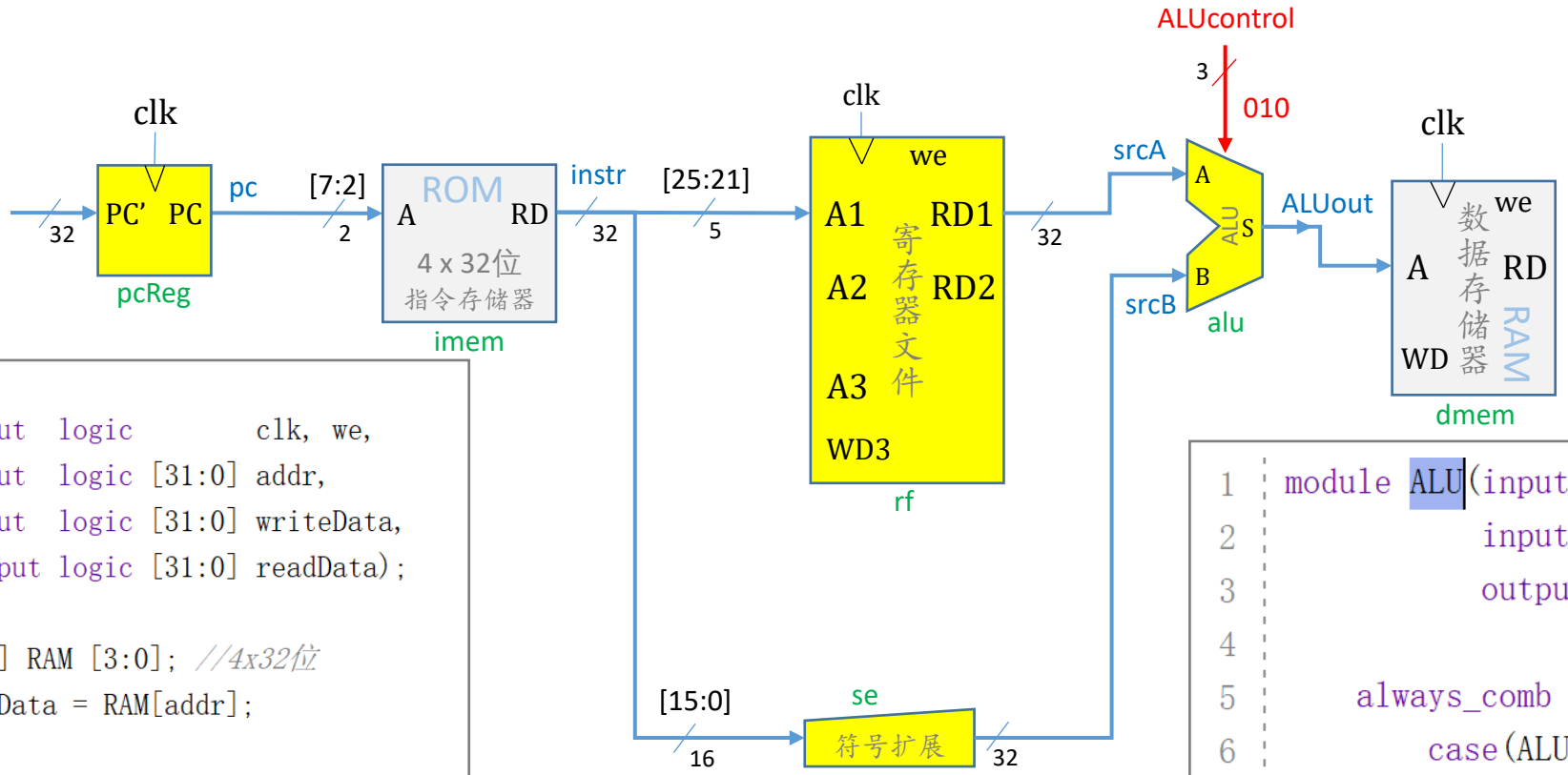



```
1  // 符号扩展 16位-> 32位
2  module SignExt(input  logic [15:0] a,
3                 output logic [31:0] y);
4
5      assign y = {{16{a[15]}}, a};
6  endmodule
```

lw rt, imm(rs)

STEP 4: 计算存储器地址






```
1  // 数据寄存器
2  module dRAM(input  logic        clk, we,
3              input  logic [31:0] addr,
4              input  logic [31:0] writeData,
5              output logic [31:0] readData);
6
7      logic [31:0] RAM [3:0]; //4x32位
8      assign readData = RAM[addr];
9
10     always_ff @(posedge clk)
11         if(we) RAM[addr] <= writeData;
12
13     initial RAM[0] = 32'h1234_5678;//赋初值
14 endmodule
```

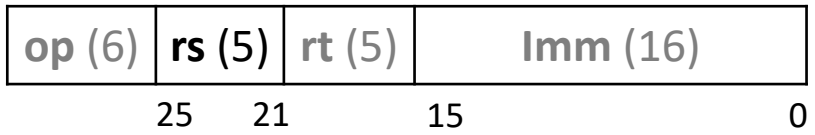
```
1  module ALU(input  logic [31:0] a, b,
2             input  logic [2:0]  ALUcontrol,
3             output logic [31:0] result );
4
5      always_comb
6          case(ALUcontrol)
7              3'b010:  result = a + b;
8              default: result = 0;
9          endcase
10 endmodule
```

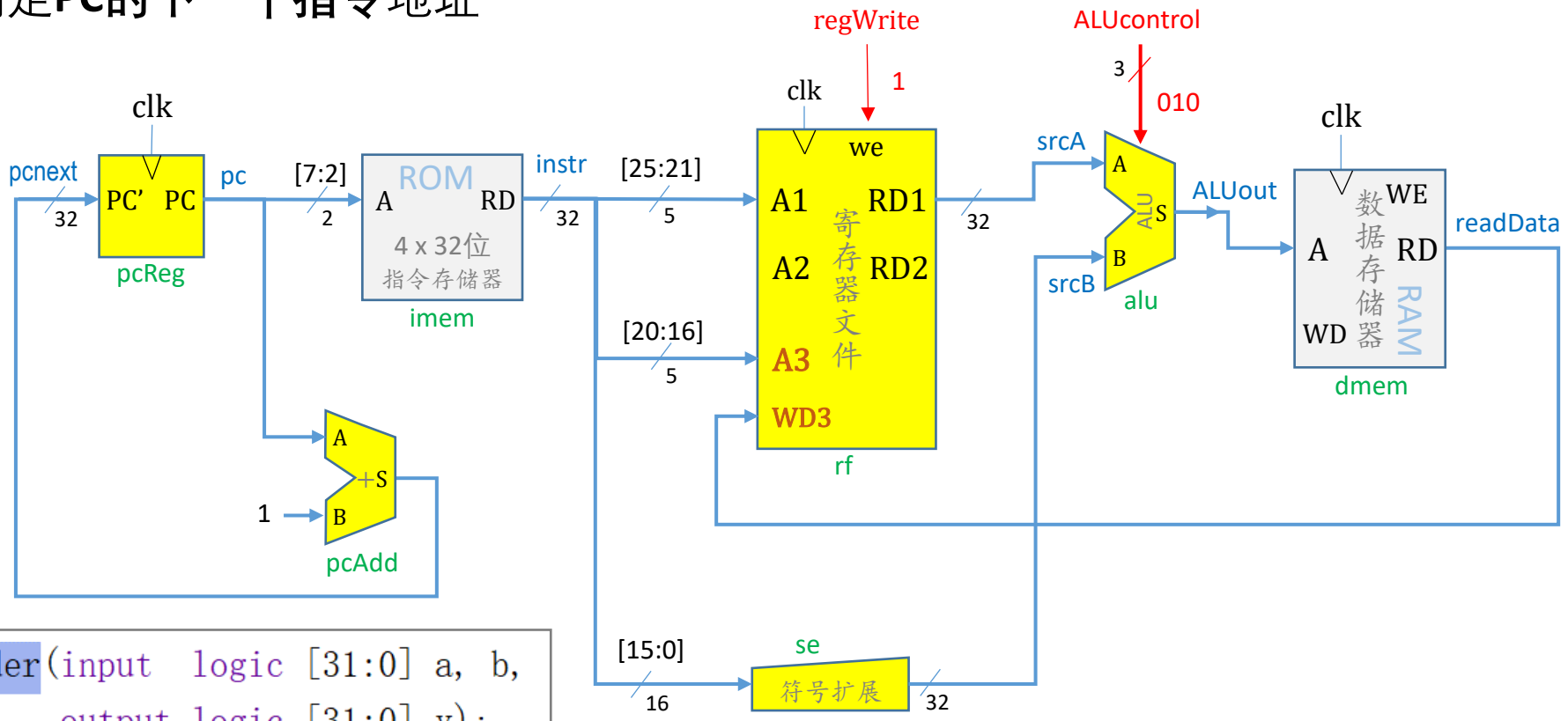
lw rt, imm(rs)

STEP 5: 向寄存器文件写入数据

STEP 6: 确定PC的下一个指令地址

100011





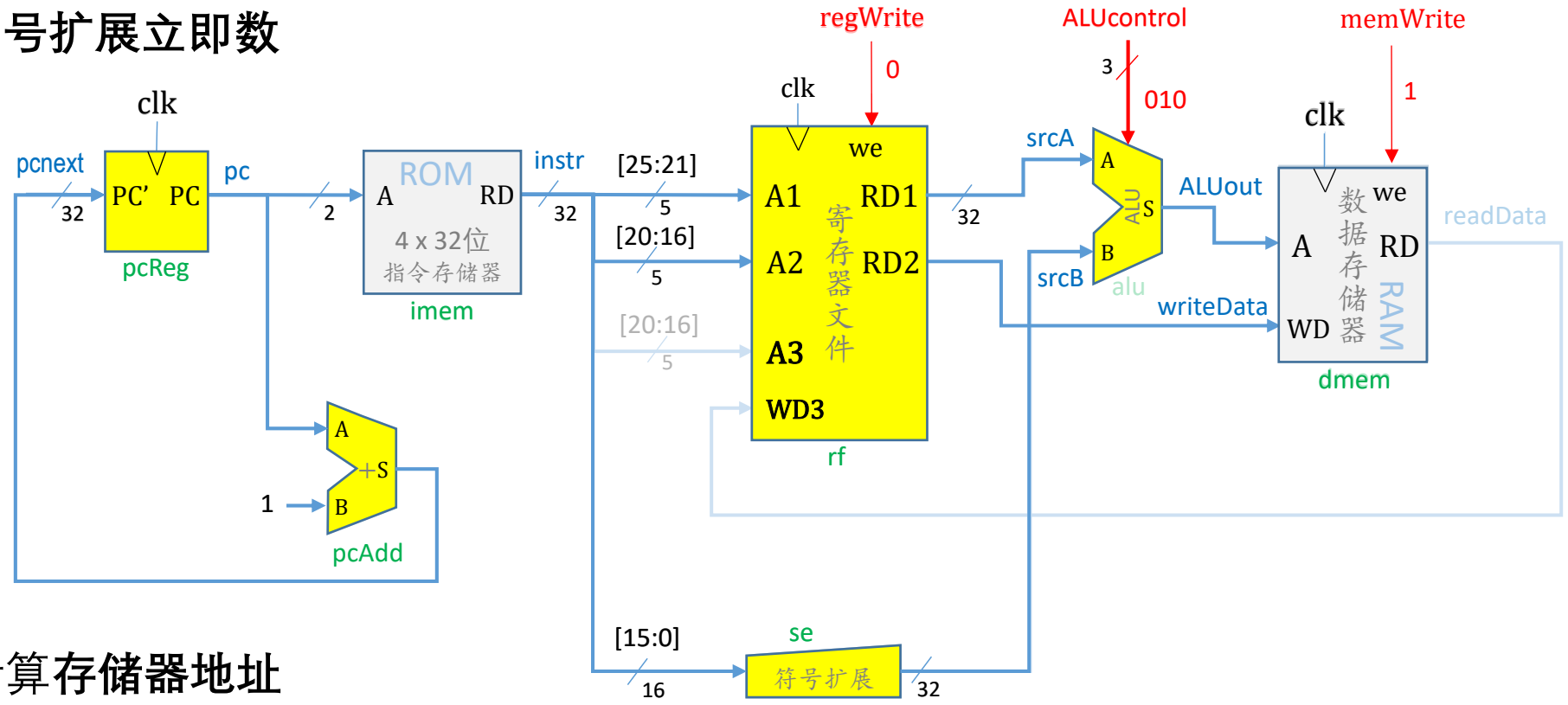

```
1  module adder(input  logic [31:0] a, b,
2               output logic [31:0] y);
3
4    assign y = a + b;
5  endmodule
```

sw rt, imm(rs)

STEP 1: 从**指令存储器**中取出指令

STEP 2: 从**寄存器文件**中读出**源操作数**

STEP 3: **符号扩展立即数**



STEP 4: 计算**存储器地址**

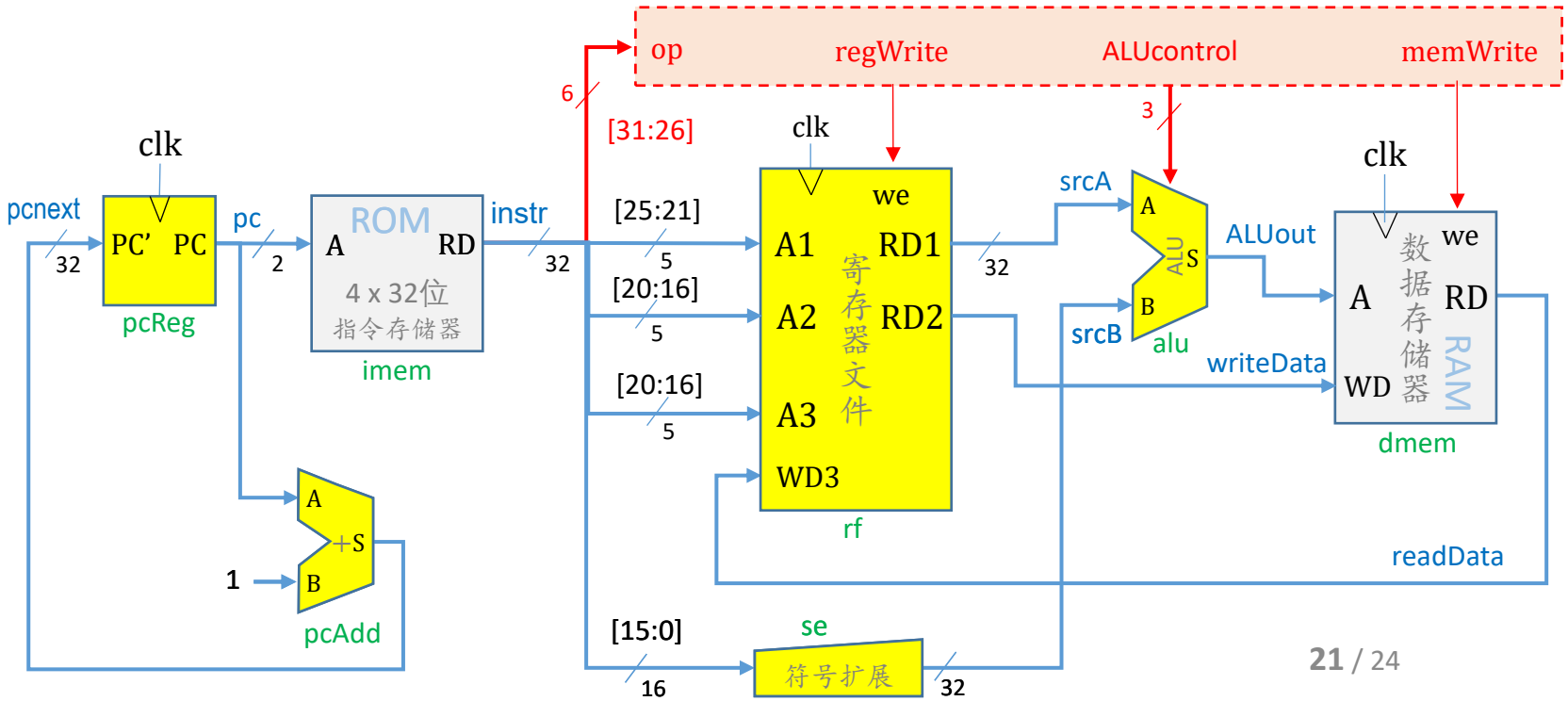
STEP 5: 向**数据存储器**写入**数据**

STEP 6: 确定**PC的下一个指令**地址

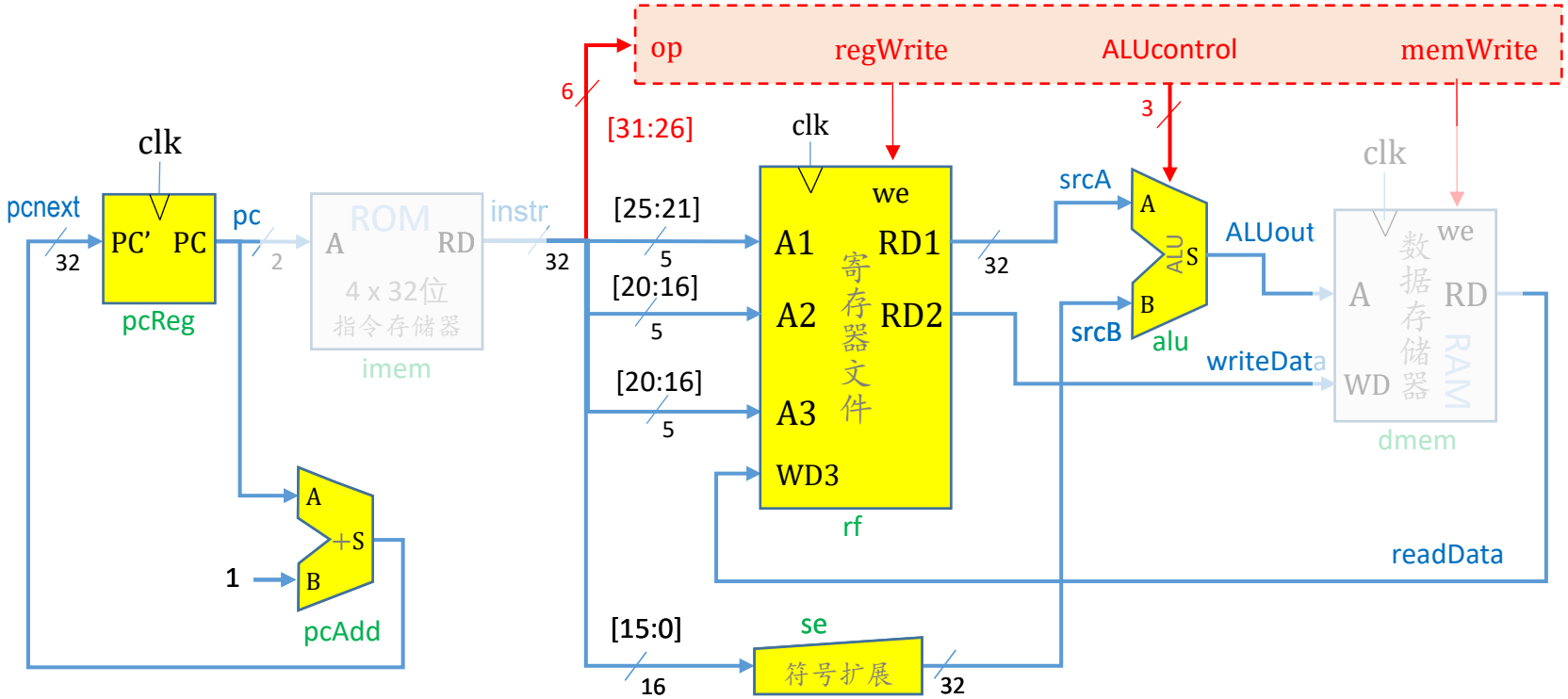
数据路径 + 控制单元

```
module datapath(
    input  logic        clk, reset,
    // from iROM
    input  logic [31:0] instr,
    // from dRAM
    input  logic [31:0] readData,
    // from controller 控制信号
    input  logic        regWrite,
    input  logic [2:0]  aluControl,
    // to iROM
    output logic [31:0] pc,
    // to dRAM
    output logic [31:0] aluOut,
    output logic [31:0] writeData );

    logic [31:0] pcnext, srca, srcb;

    PC_register pcReg(clk, reset, pcnext, pc);
    adder       pcAdd(pc, 32'b1, pcnext);
    registerFile rf(clk,          regWrite,
                    instr[20:16], readData,
                    instr[25:21], instr[20:16],
                    srca,         writeData );
    SignExt     se(instr[15:0], srcb);
    ALU         alu(.a(srca), .b(srcb),
                    .ALUcontrol(aluControl),
                    .result(aluOut));
endmodule
```

```
module controller(input  logic [5:0] op,    // 操作码
                  output logic       regWrite, // to registerFile
                  output logic       memWrite, // to dRAM
                  output logic [2:0] aluControl);// to ALU
    always_comb
        case(op)
            6'b100011: begin regWrite = 1;     memWrite = 0;     end //LW
            6'b101011: begin regWrite = 0;     memWrite = 1;     end //SW
            default:   begin regWrite = 1'bx; memWrite = 1'bx; end
        endcase
    assign aluControl = 3'b010; // ADD
endmodule
```

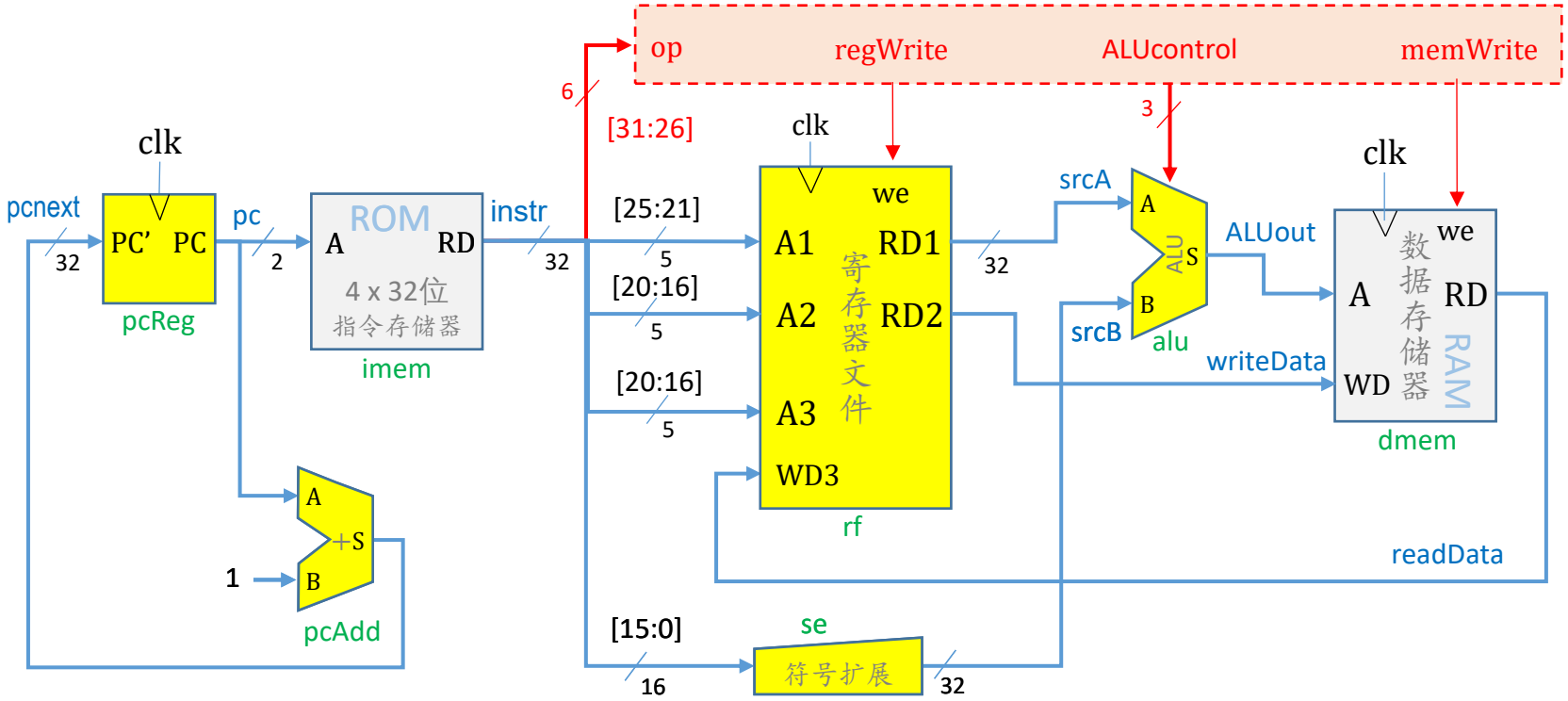


CPU



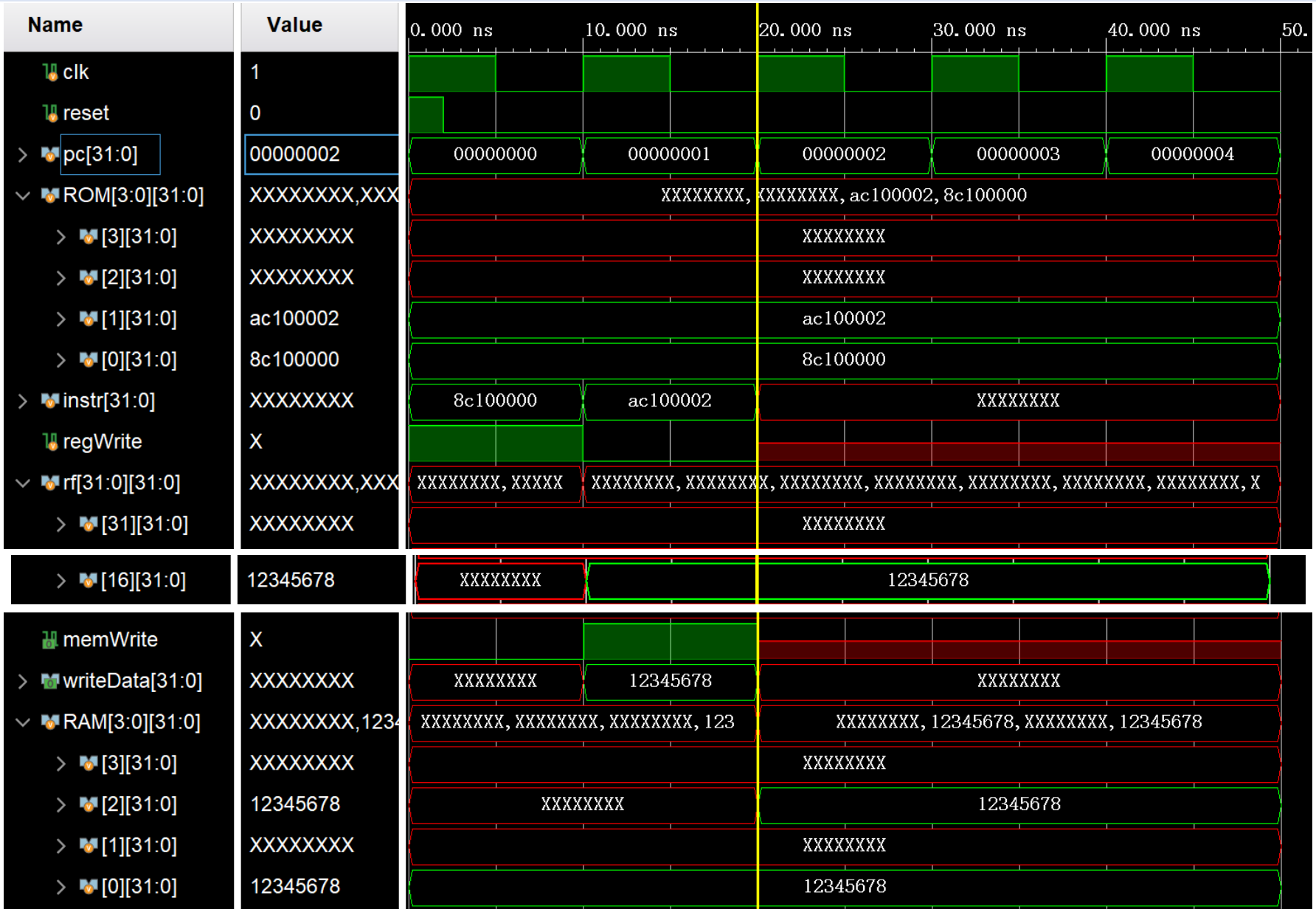

```
2  module CPU(input  logic        clk, reset,
3             input  logic [31:0] instr,
4             input  logic [31:0] readData,
5             output logic [31:0] pc,
6             output logic        memWrite,
7             output logic [31:0] aluOut,
8             output logic [31:0] writeData  );
9
10    logic       regWrite;
11    logic [2:0] aluControl;
13    controller c(instr[31:26], // input 操作码
14                 regWrite, memWrite, aluControl);
15    datapath dp(clk, reset,
16                instr,     // from iROM
17                readData,  // from dRAM
18                regWrite, aluControl, //from 控制单元
19                pc,        // to iROM
20                aluOut, writeData);   //to dRAM
21 endmodule
```

micro Computer




```
1  // 只能运行2条指令的计算机
2  module Computer(input logic  clk, reset);
3      logic memWrite;
4      logic [31:0] pc, instr, readData;
5      logic [31:0] ALUout,      writeData;
7      CPU C2(clk, reset,   instr, readData,
8             pc, memWrite, ALUout, writeData);
9      iROM imem(pc, instr);
10     dRAM dmem(clk, memWrite, ALUout,
11               writeData, readData);
12 endmodule
```

仿真结果



- 0ns时， ROM中有指令
- 0ns时， RAM中有数据
- PC指针不断+1
- instr不断读出指令
- 第1时钟后, lw到rf中
- 第2时钟后,sw到RAM中