MIPS CPU设计实验

MIPS CPU控制器设计

传统三级时序——定长指令周期：单周期实现

- 所有指令均在一个时钟周期内完成，CPI = 1
- 性能取决于最慢的指令，时钟周期过长

现代时序——变长指令周期：多周期实现

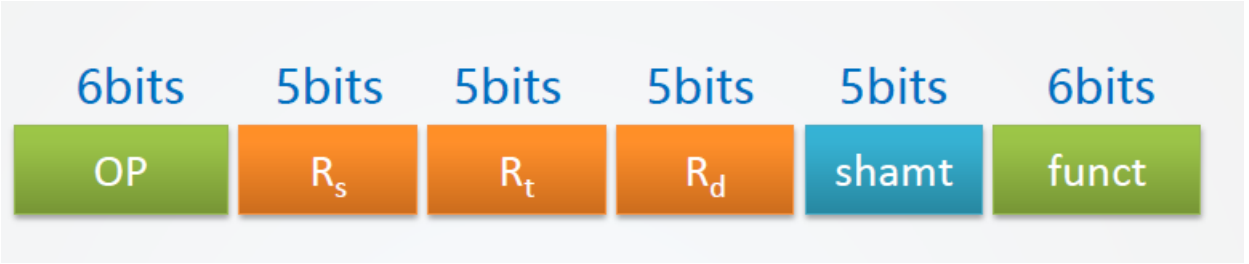
- 缩短时钟周期，复用器件或数据通路
- 可支持流水线操作，提升性能

MIPS 指令系统介绍

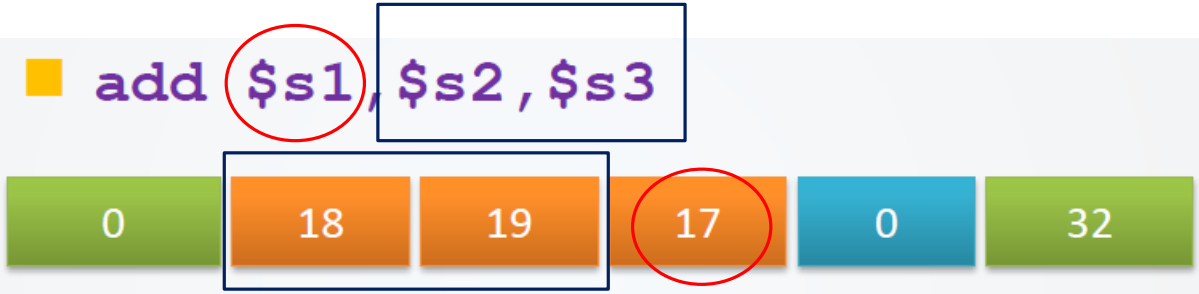
共有3类指令，可根据op判断指令类型

- Op = 0， R型指令，funct字段描述运算功能；
- OP不为0，根据op的值判断具体指令类型与功能

典型的R型指令



- `add $s1,$s2,$s3`



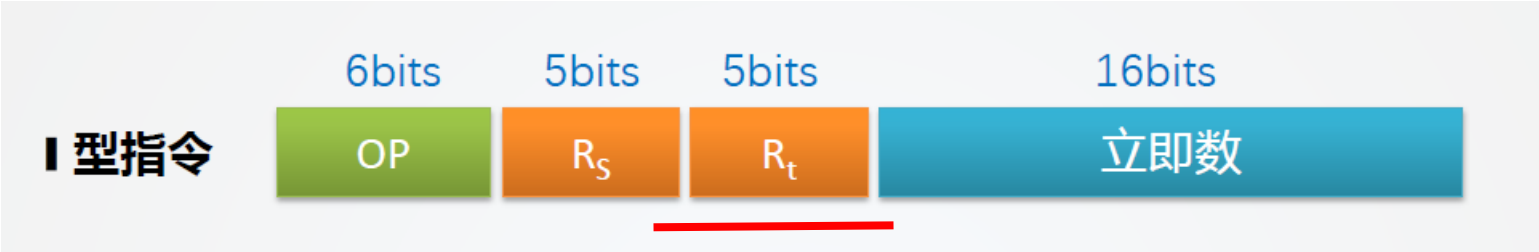
← Add的扩展操作码的值

- `sub $s0,$s1,$s2`

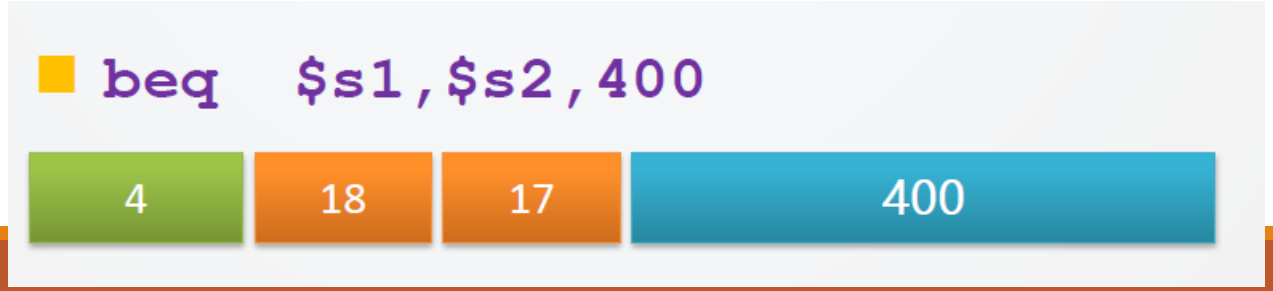
- `sll $s0,$s1,2`　　左移，S0 = S1<<2

不使用

典型的I型指令



与R型指令相比，指令中指示目的寄存器的位置变化



比较指令， 当 S1 == S2，则跳转到400

一、单周期的MIPS CPU设计实验（8条指令）

实验目标

- 掌握硬布线控制器设计的基本原理
- 能利用相关原理在Logisim中设计实现MIPS单周期CPU，支持内存中简单的冒泡排序程序

主要任务（设计思路）

- 设计绘制MIPS CPU的数据通路
- 实现单周期硬布线控制器
- 连接控制器与数据通路
- 软、硬件联调测试

复杂的数字系统

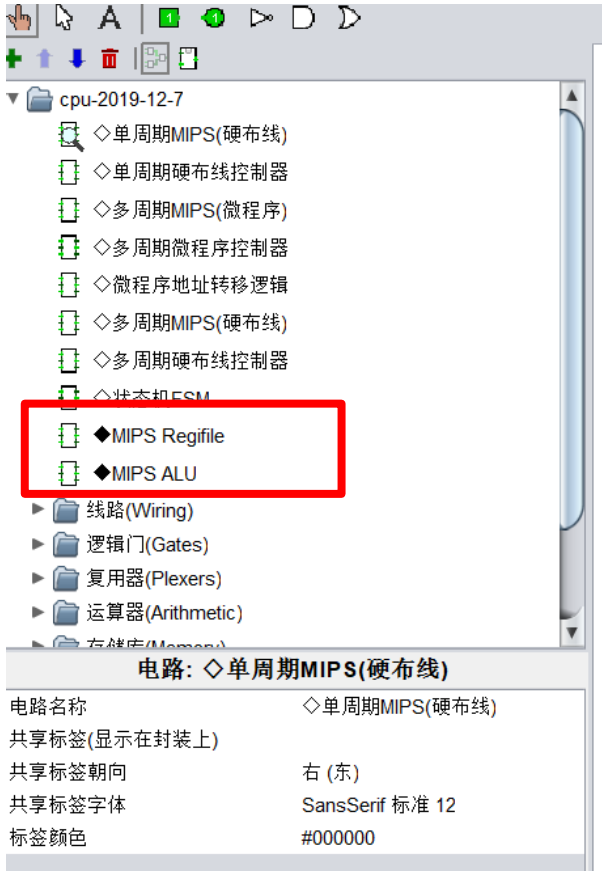
能支持下列8条指令组成的程序

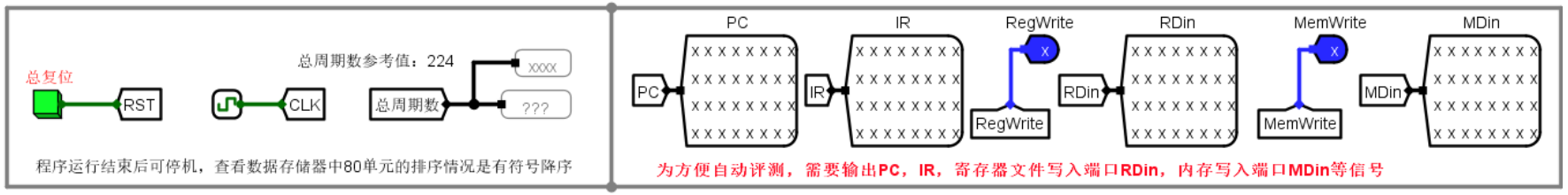
本次实验设计的原型系统，不处理溢出处理

#	MIPS指令	RTL功能描述
1	add $rd,$rs,$rt	R[$rd]←R[$rs]+R[$rt]　溢出时产生异常，且不修改R[$rd]
2	slt $rd,$rs,$rt	R[$rd]←R[$rs]<R[$rt]　小于置1，有符号比较
3	addi $rt,$rs,imm	R[$rt]←R[$rs]+$SignExt_{16b}$(imm)　溢出产生异常
4	lw $rt,imm($rs)	R[$rt]←$Mem_{4B}$(R[$rs]+$SignExt_{16b}$(imm))
5	sw $rt,imm($rs)	Mem_{4B}(R[$rs]+$SignExt_{16b}$(imm))←R[$rt]
6	beq $rs,$rt,imm	if(R[$rs] = R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})
7	bne $rs,$rt,imm	if(R[$rs] != R[$rt]) PC ← PC + $SignExt_{18b}$({imm, 00})
8	syscall	系统调用，这里用于停机

本次实验只用到其特殊的停机功能

（一）：单周期MIPS 的数据通路设计



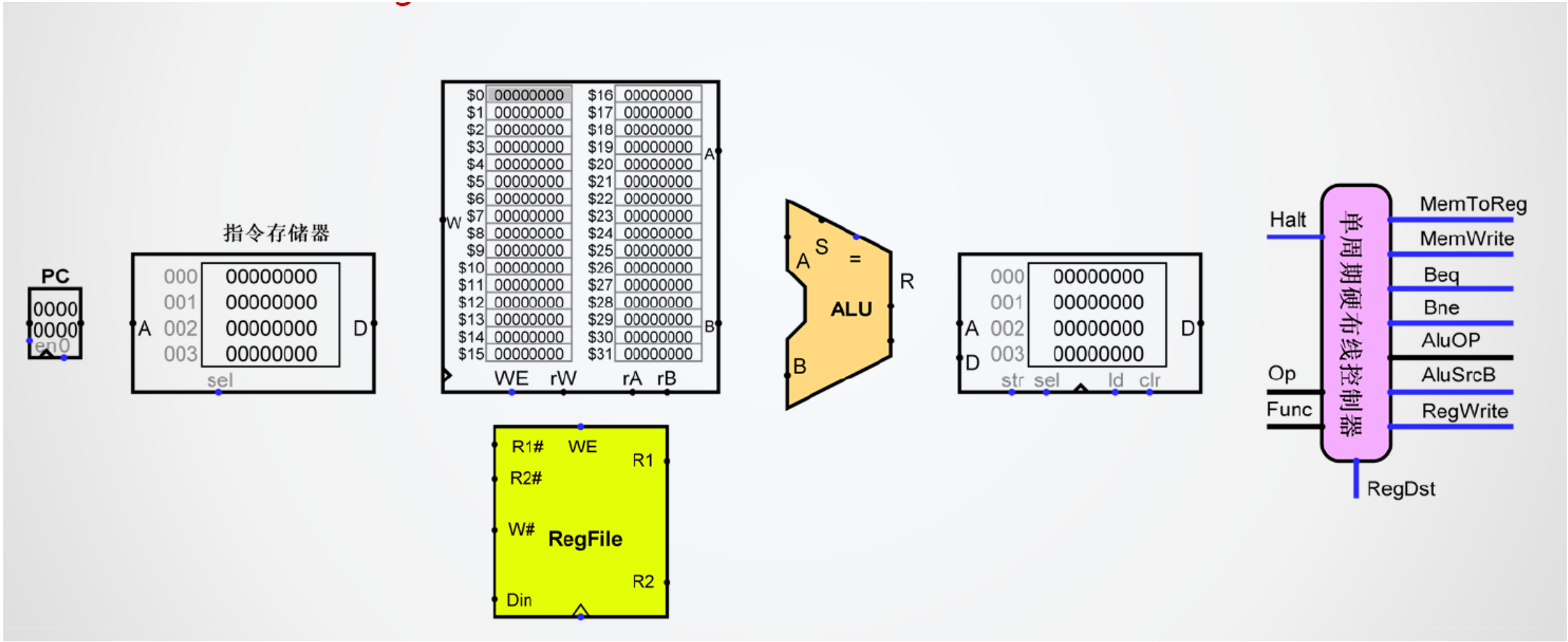


打开“单周期MIPS（硬布线）”，利用之前已经完成的运算器标准模块、寄存器标准模块以及其他组件构成数据通路

黑色星标，已经实现、封装好的MIPS寄存器文件和MIPS运算器的标准文件

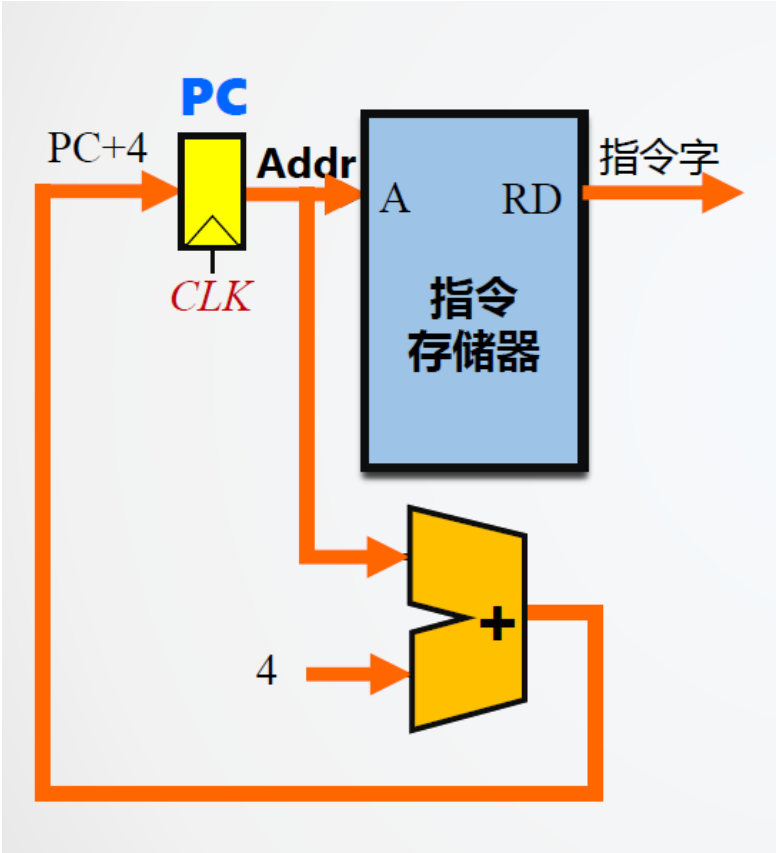
用到的组件包括：

PC、IMEM、RegFile、ALU、DMEM、Controller



1、单周期MIPS数据通路分析

（1）取指令数据通路



Mem[PC++] → IR

1、如何得到下一条指令的地址？

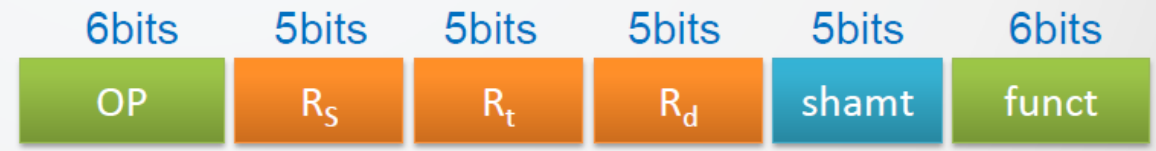
MIPS存储器按字节编址，因此为取下一条指令，PC+4->PC

2、如何满足只需要1个CPU周期的要求？

- 不单独设置AR、DR、IR寄存器
- 程序和数据分开存储：指令存储器、数据存储器
- 运算器和PC 累加器分离

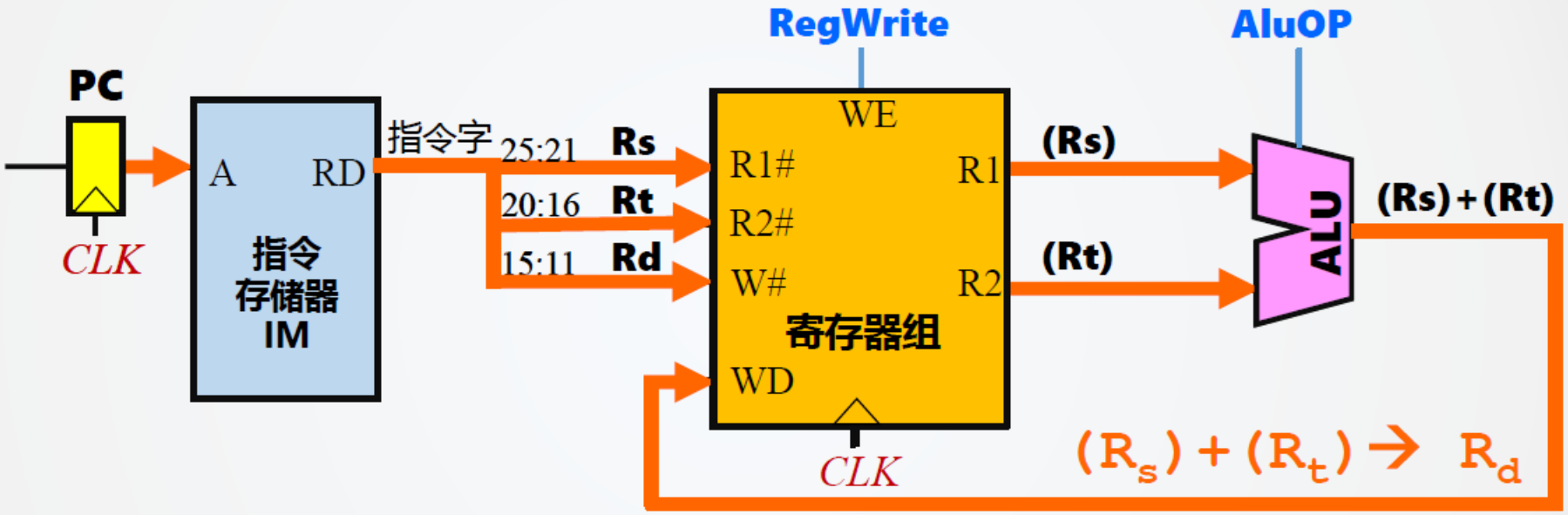
（2）加法指令（R型）的数据通路

```
add $s0,$s1,$s2
```



需要寄存器组，能支持同时2路读出，为ALU提供操作数，并能支持一路写入。

寄存器组需要的地址由指令字中对应字段提供



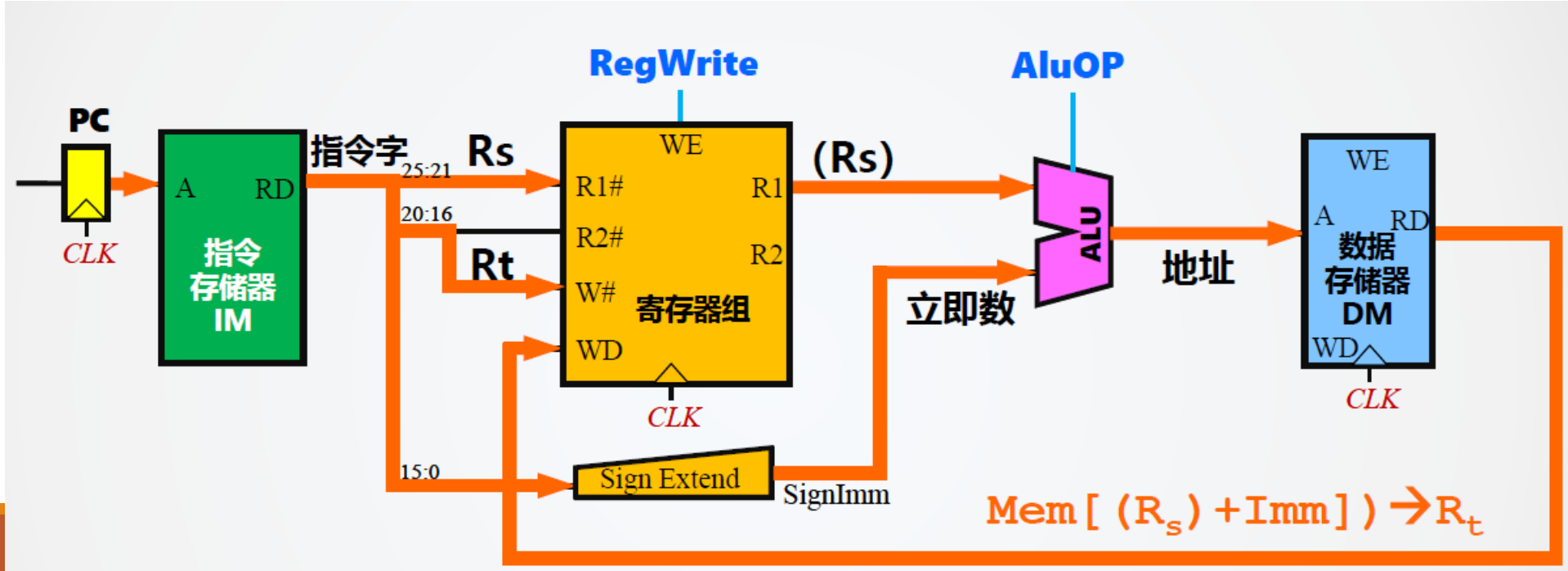
（3）LW指令（I型）的数据通路

```
lw $s0,32($s1)
```

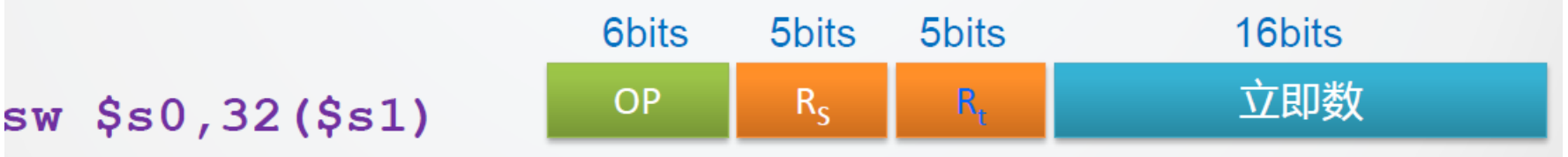
源、目寄存器地址由指令字对应字段提供；

寄存器的存储字长为32位，因此，16立即数要扩展到32位后方可参与后续工作；

计算后的地址直接送往数据存储器

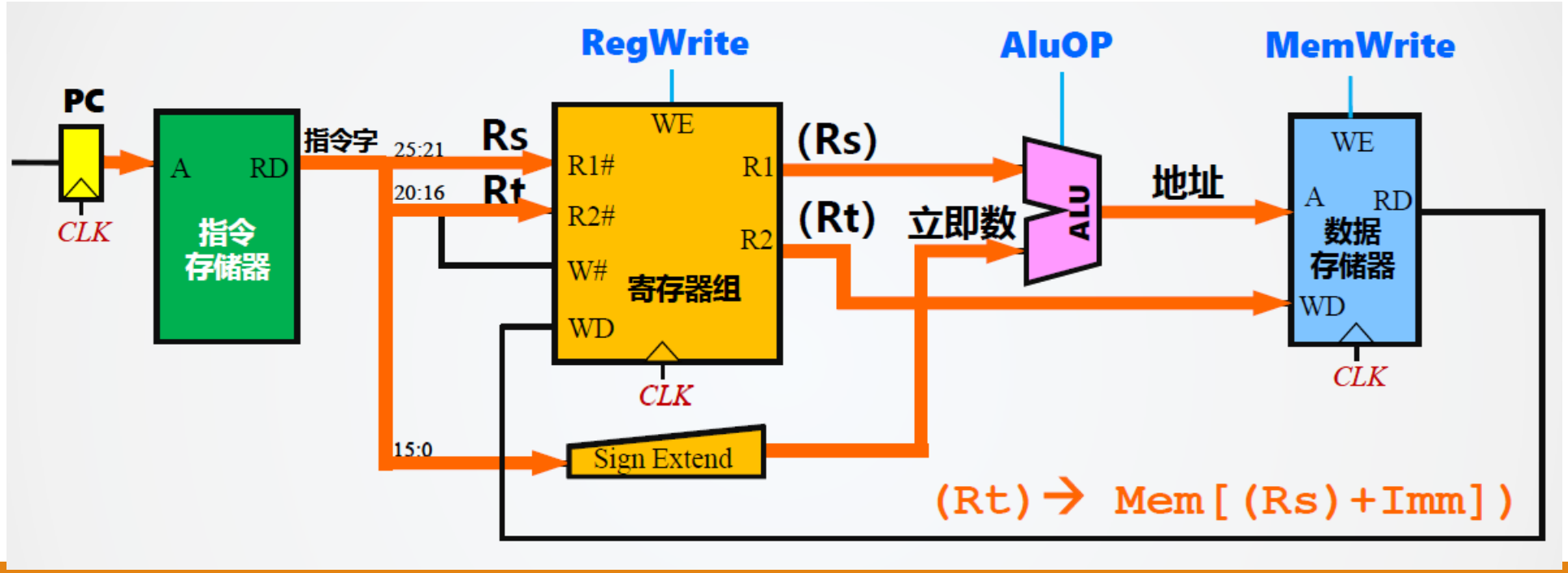


（3）SW指令（I型）的数据通路



源、目寄存器地址由指令字对应字段提供；

寄存器的存储字长为32位，因此，16立即数要扩展到32位后方可参与后续工作；

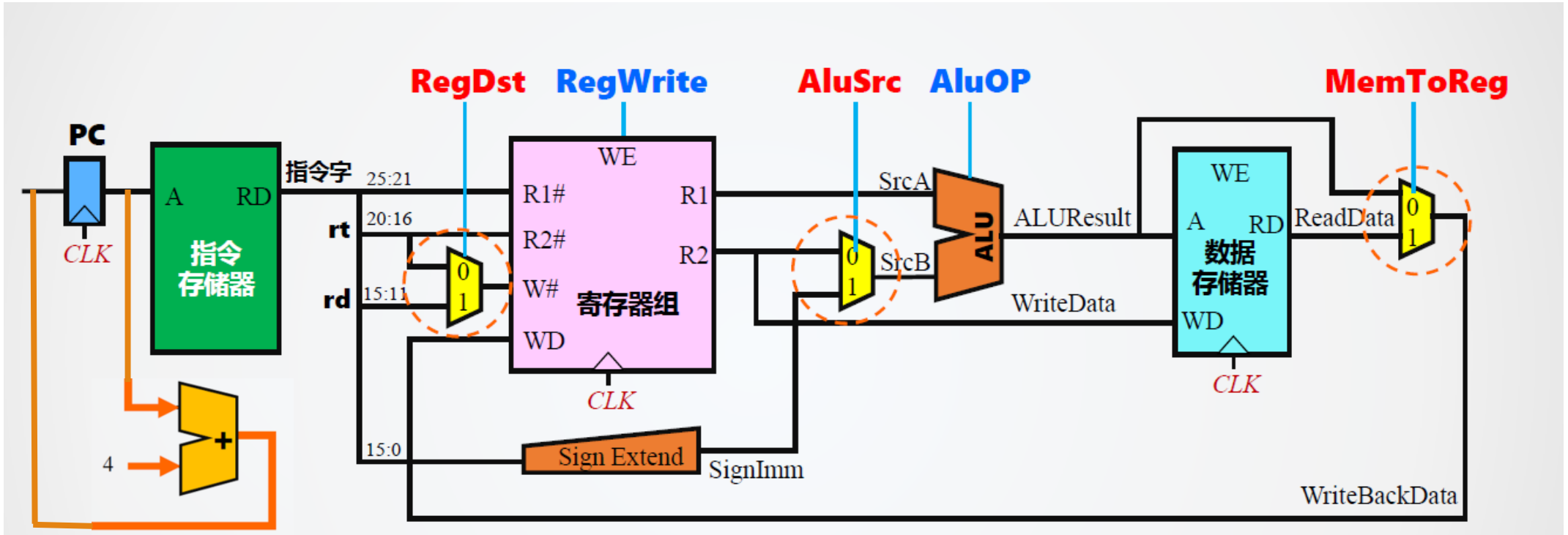


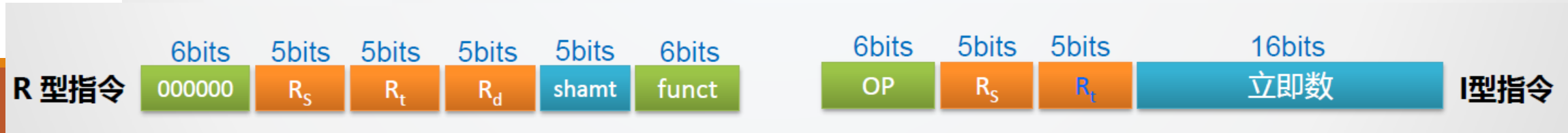
2、数据通路初步综合

多路选择器的选择信号、各执行部件的控制信号，可看做整个MIPS CPU的控点，需要MIPS CPU提供。

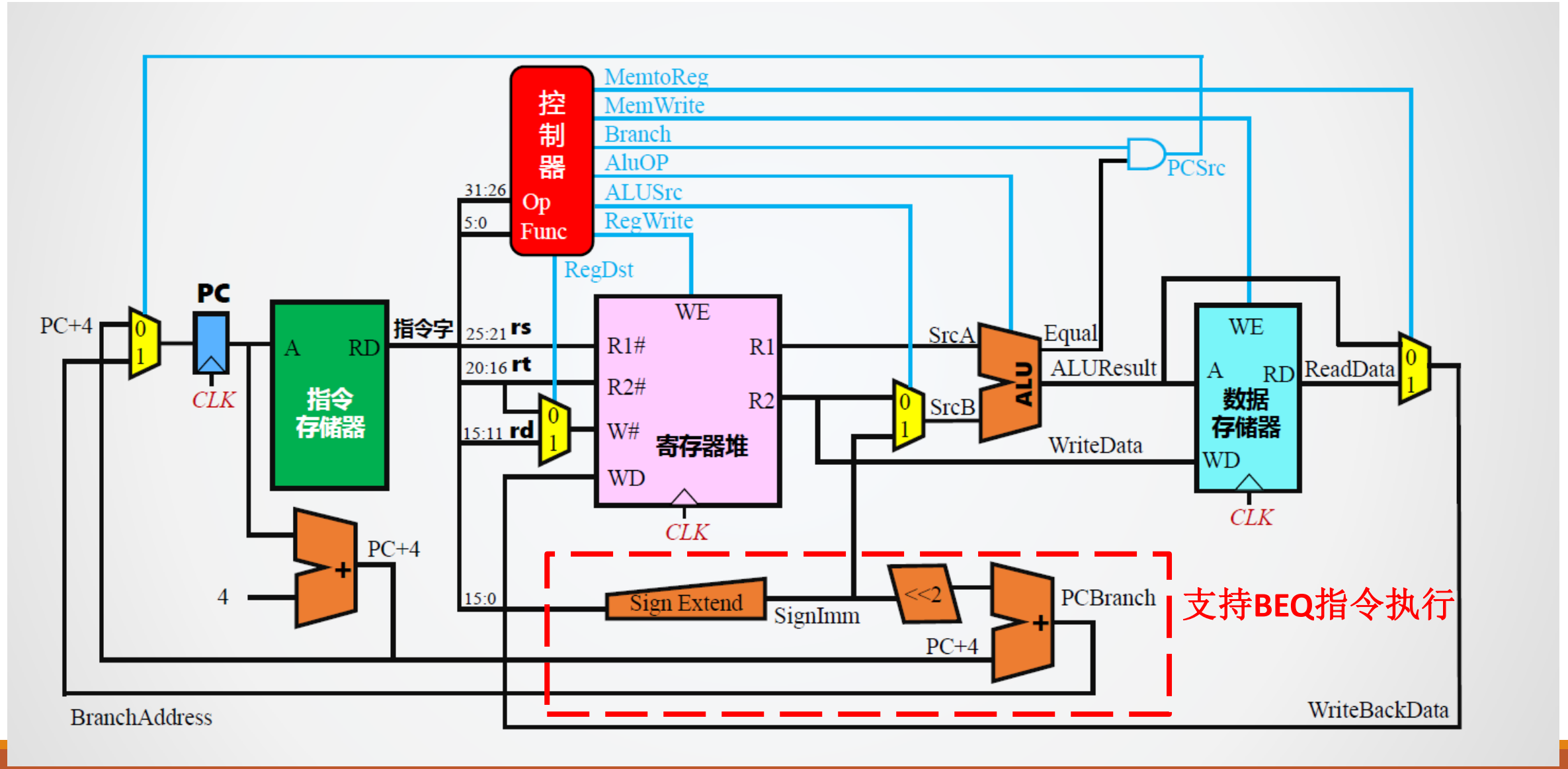
- 将不同类型指令下的数据通路综合起来
- 对具有多个输入源的引脚，添加多路选择器，进行信号的选择

（1）写入寄存器的编号 （2）运算器的第二个操作数 （3）写回寄存器的对应数据





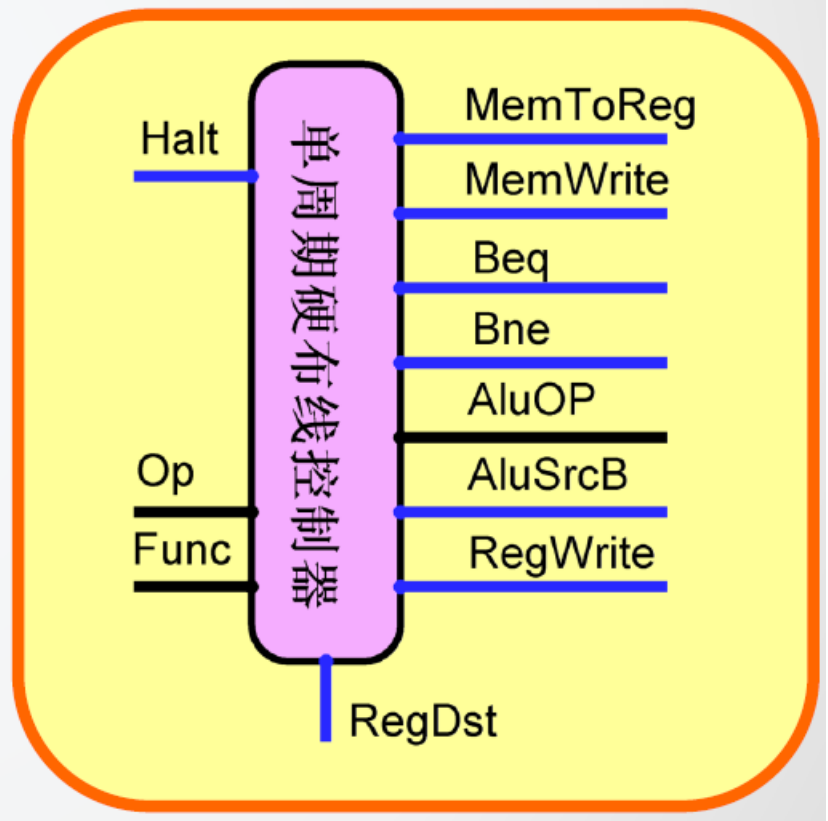
3、可参考的完整的单周期MIPS CPU数据通路



（二）：单周期MIPS控制器的设计

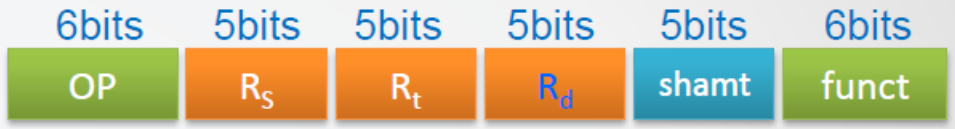
- 输入信号
  - 指令字Opcode，Func字段（12位）
- 输出信号
  - 多路选择器选择信号
  - 内存访问控制信号
  - 寄存器写使能信号
  - 运算器控制信号、指令译码信号
- 纯组合逻辑电路、无时序逻辑

分析不同指令的数据通路建立过程，
总结出控制信号的产生条件



R型指令数据通路建立过程

R 型指令

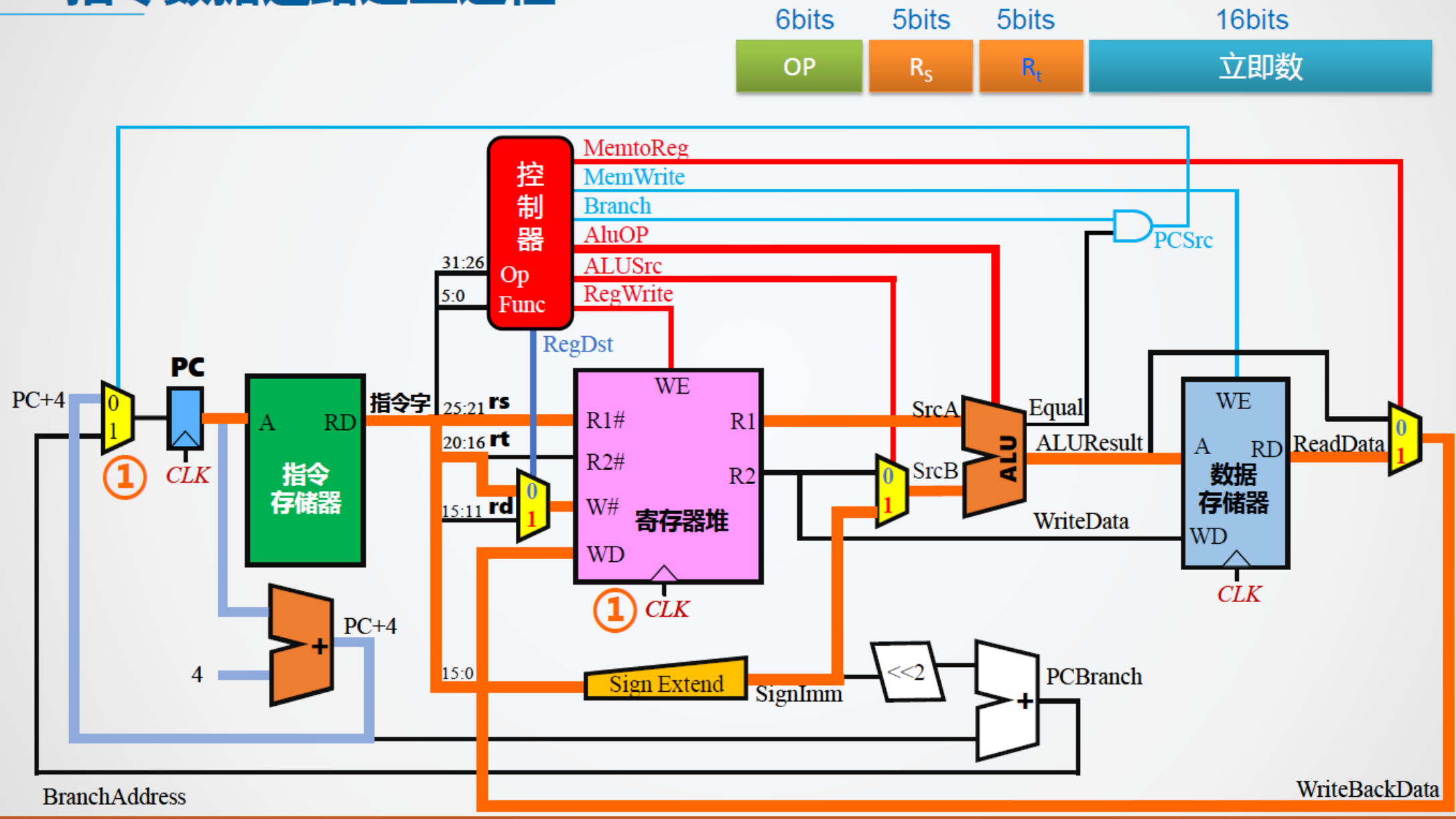


思考题：

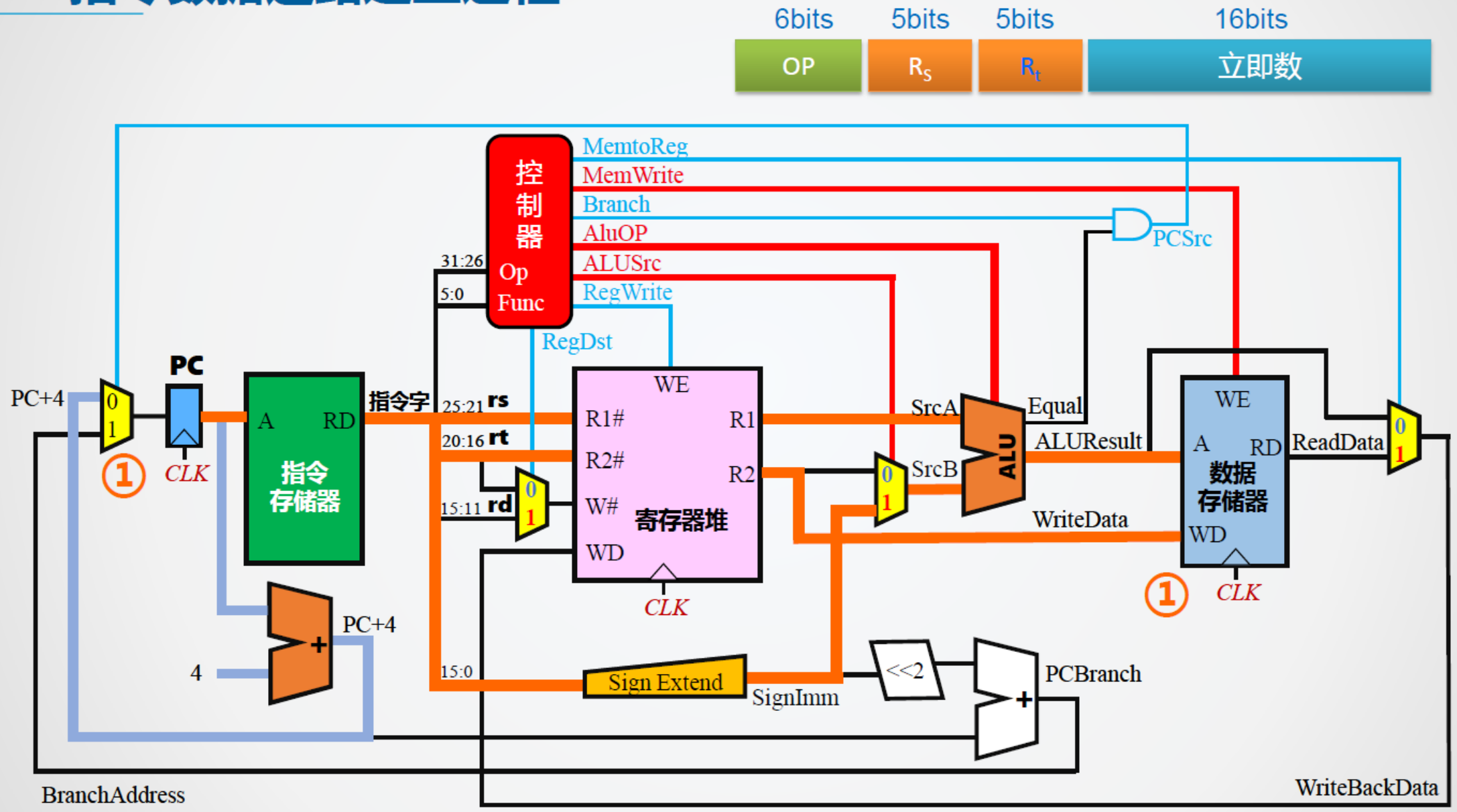
移位指令

如何实现

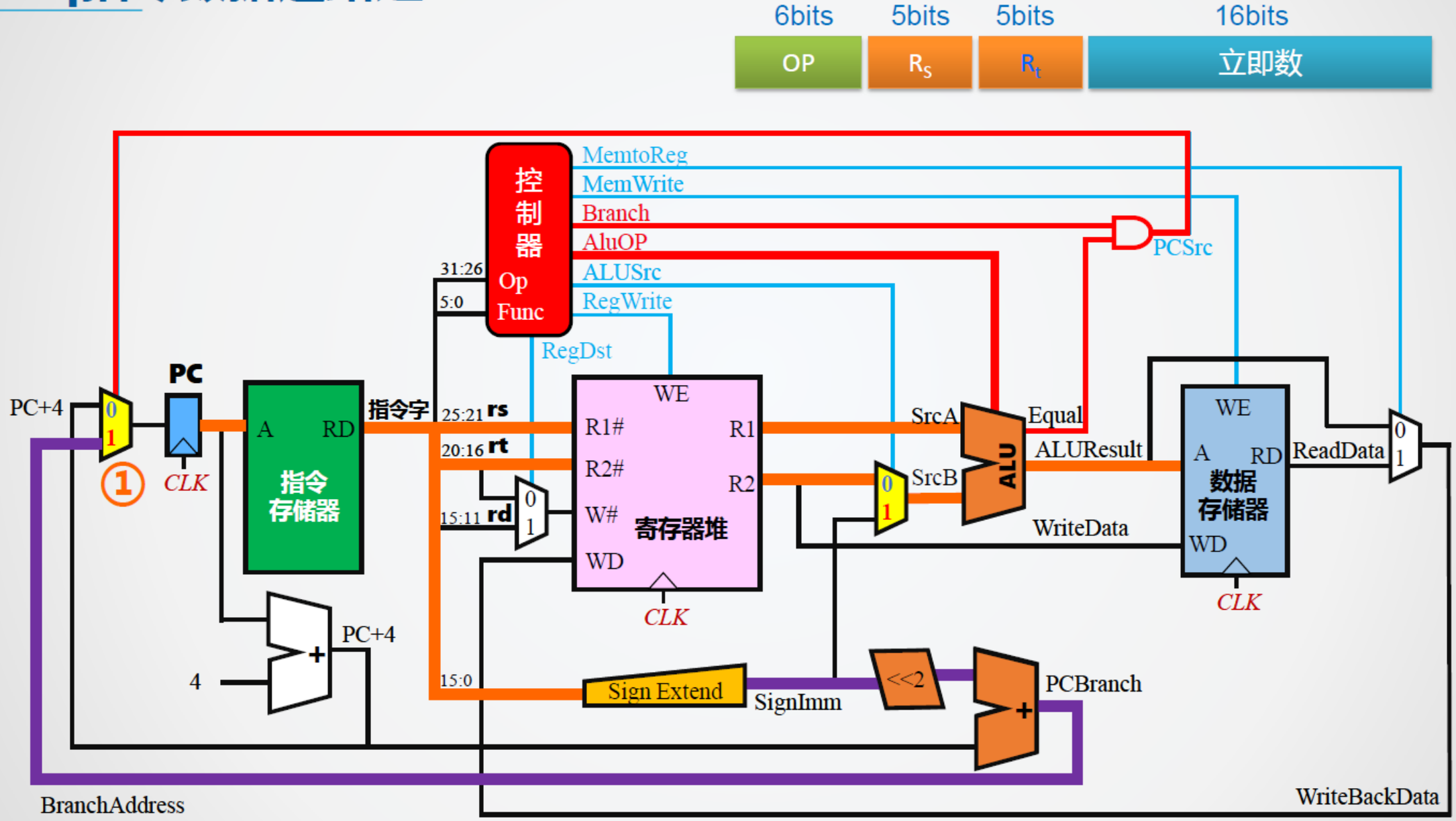
LW指令数据通路建立过程



SW指令数据通路建立过程



Beq指令数据通路建立



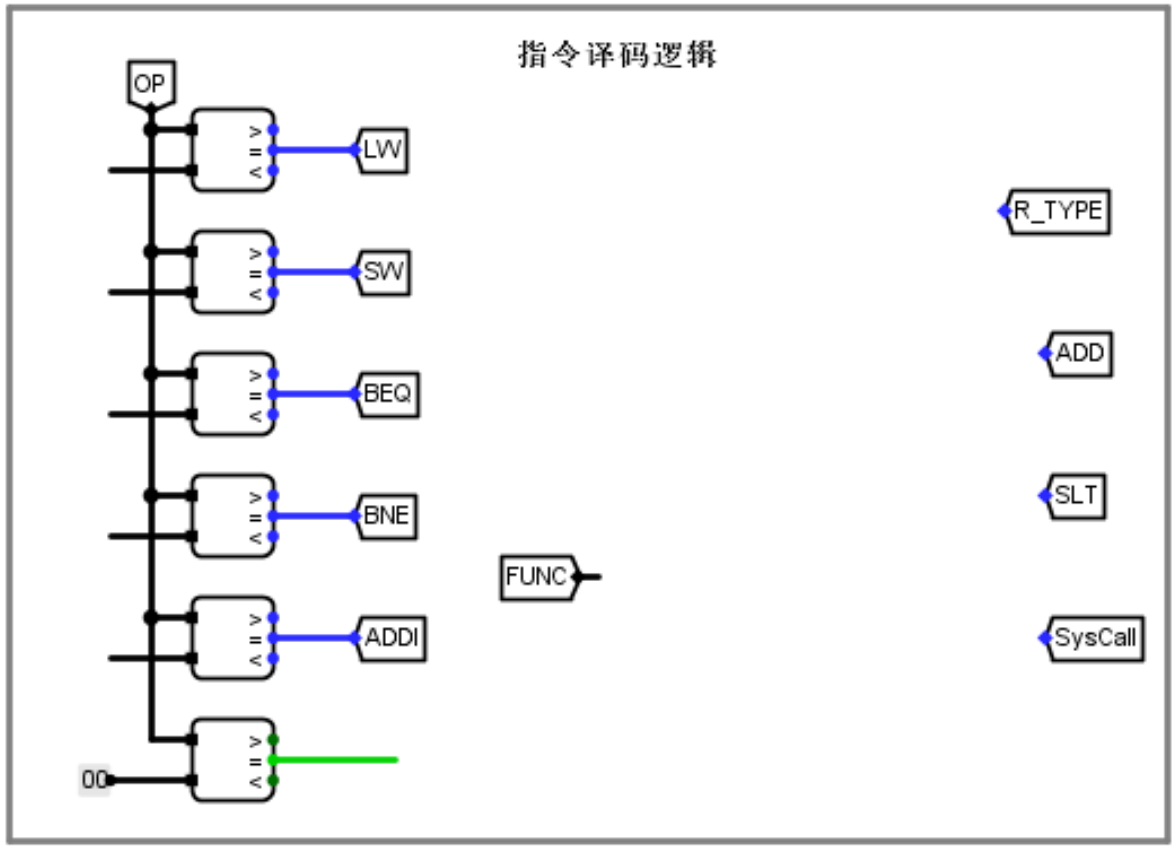
1、控制信号产生条件分析

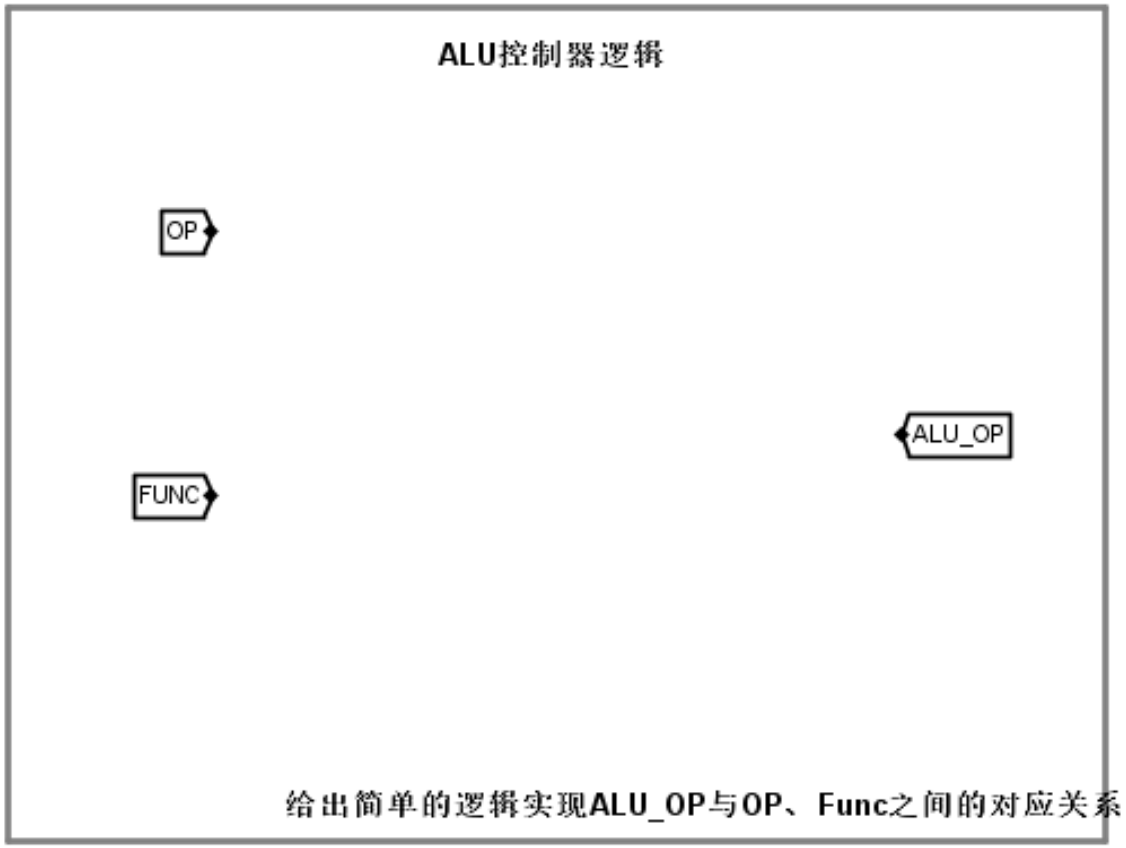
#	控制信号	信号说明	产生条件
1	MemToReg	写入寄存器的数据来自存储器	lw指令
2	MemWrite	写内存控制信号	sw指令　未单独设置MemRead信号
3	Beq	Beq指令译码信号	Beq指令
4	Bne	Bne指令译码信号	Bne指令
5	AluOP	运算器操作控制符	加法，比较两种运算
6	AluSrcB	运算器第二输入选择	Lw指令，sw指令，addi
7	RegWrite	寄存器写使能控制信号	寄存器写回信号
8	RegDst	写入寄存器选择控制信号	R型指令
9	Halt	停机信号，取反后控制PC使能端	syscall指令

2、完善硬布线控制器内部逻辑

打开“CPU.circ” 单周期硬布线控制器电路

- 实现指令译码、ALU控制逻辑
- 利用比较器判断操作码的值，从而快速完成译码（查询MIPS 32指令手册,找到8条指令）
- 根据OP和FUNC，ALU所需要的4位运算操作码（查询32位MIPS运算器功能表:有符号加0101、有符号比较1011）

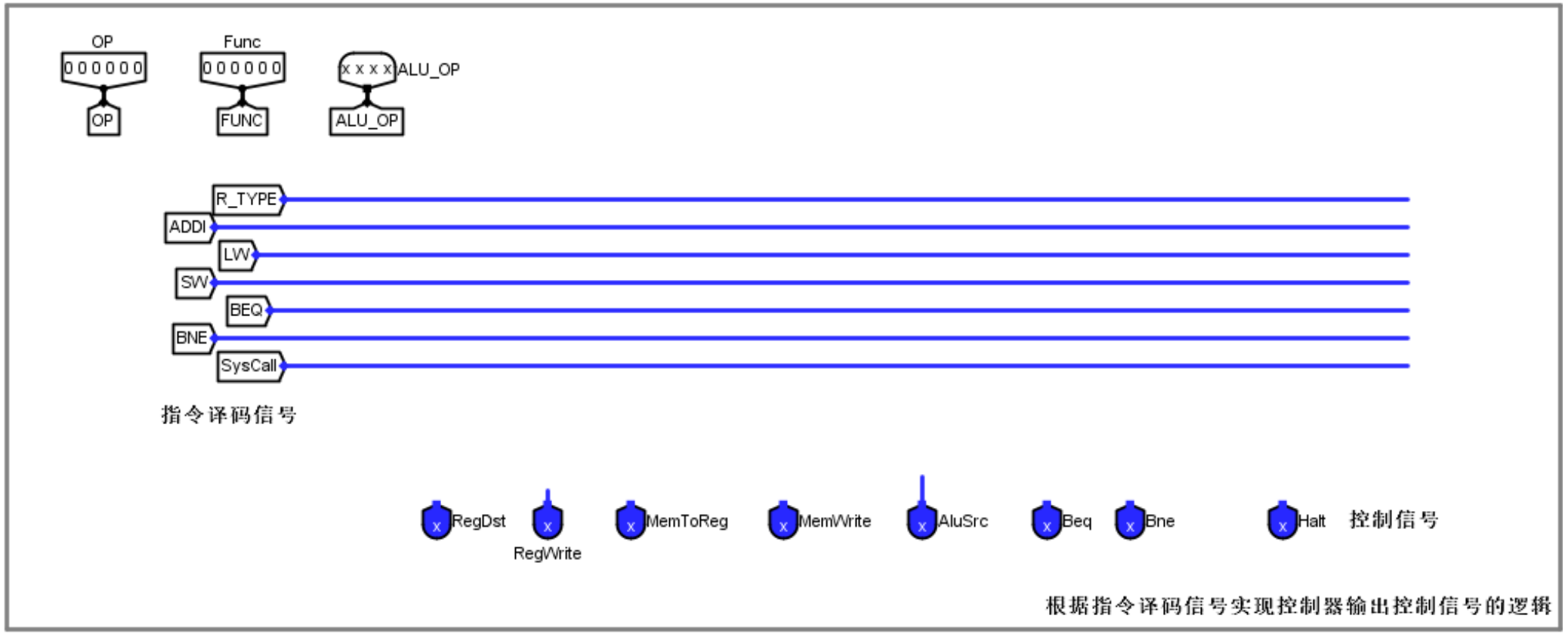




2、完善控制信号逻辑　　简单的组合逻辑

- 根据给出的指令译码信号，利用Logisim中的电路分析功能，实现所有控制信号的逻辑

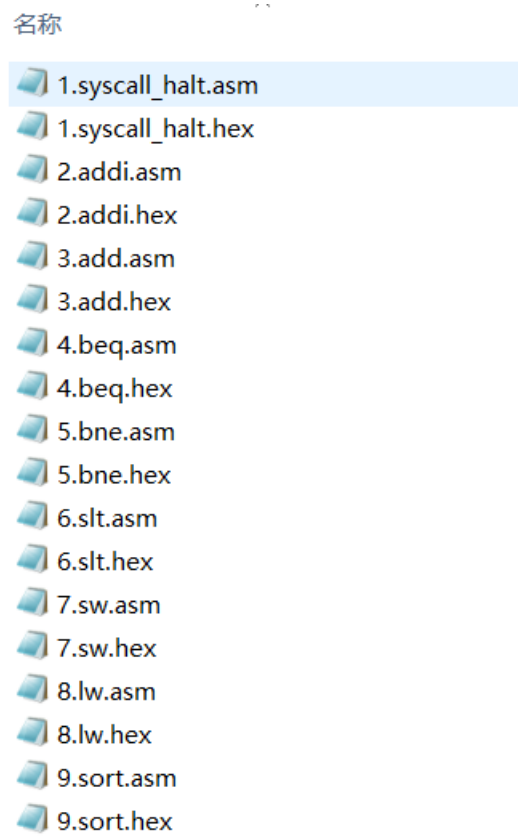
例如： MemWrite = SW； AluSrcB = LW +SW + addi



（三）：CPU测试

1、指令功能测试

- 在执行存储器中载入某一功能指令的测试程序（如add.hex; lw.hex）
- 时钟自动仿真，Ctrl +K 或Ctrl +t ，连续或单拍地运行程序
- 程序停机后，查看寄存器或者数据存储器中的情况是否正确，并对电路进行调试。
- 重复上述过程，直到所有测试样例均正确通过。



2、综合测试

- 在执行存储器中载入排序程序 sort.hex
- 时钟自动仿真，Ctrl +K，则开始运行程序
- 程序停机后，查看存储器中排序情况

注意：停机不能直接作用于时钟信号，而是作用于各寄存器或操作部件的使能端

000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
010	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
020	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
030	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
040	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
050	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
060	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
070	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
080	00000006	00000005	00000004	00000003	00000002	00000001	00000000	ffffffff

排序程序运行结果——有符号降序