实验报告

# 设计

## 概述

使⽤logisim实现单周期MIPS处理器，⽀持add, sub, ori, lw, beq, lui, nop指令，主要包括IFU、GRF、ALU、

DM、EXT、Ctrl模块

## 模块

### GRF

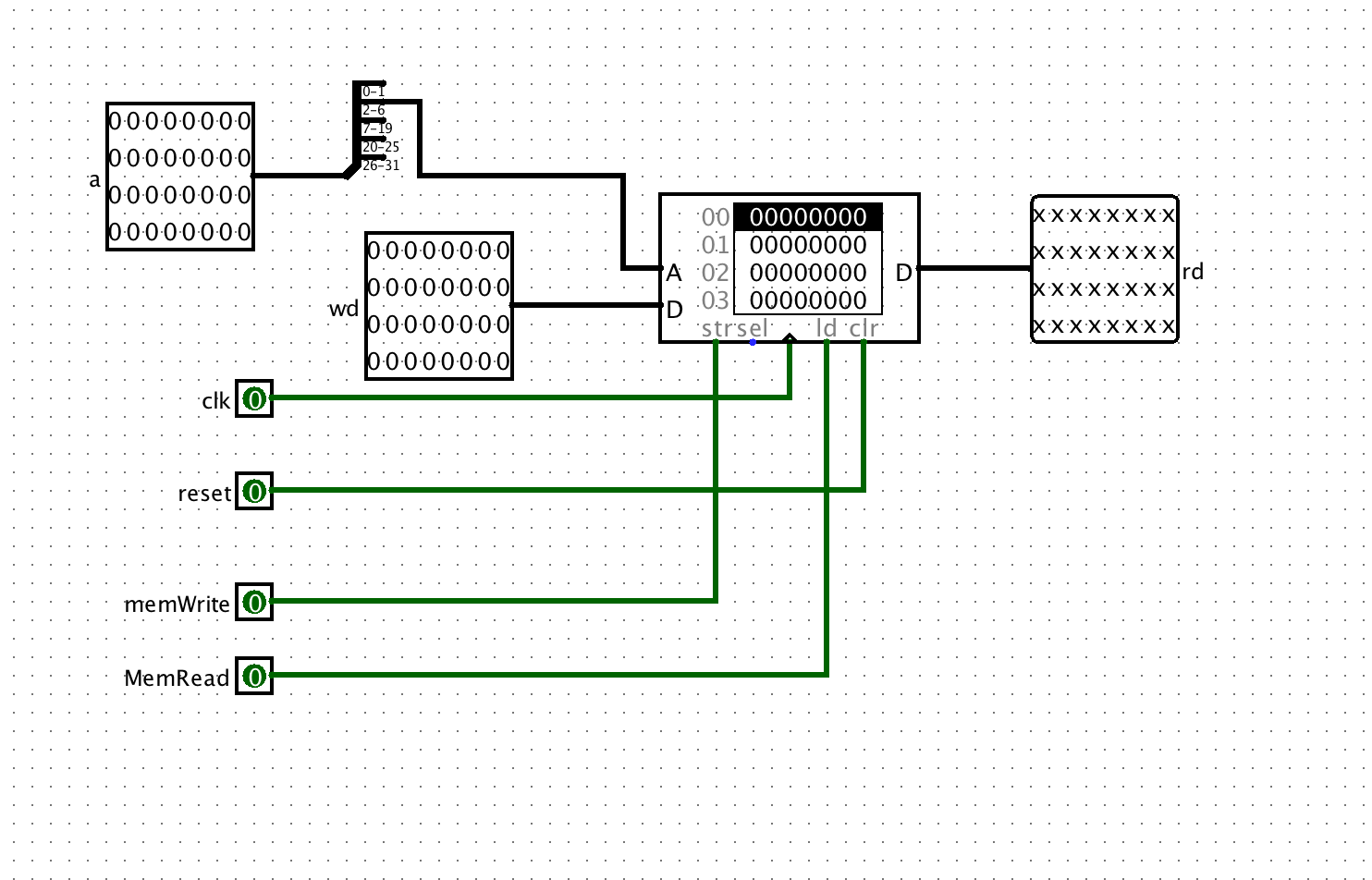
#### 主要功能

1. 复位，接受复位信号时重置所有寄存器为0x0。
2. 读取，根据输⼊地址读出32位数据
3. 写⼊，根据输⼊地址将数据写⼊寄存器

### DM

#### 主要功能

1. 复位，接受复位信号时重置所有数据为0x0。
2. 读取，根据输⼊地址读取数据
3. 写⼊，根据输⼊地址写⼊数据



### ALU

#### 主要功能

实现与，或，加，减，是否相等以及⽐较⼤⼩功能，有个信号输出输出计算结果是否为 0，用于计算 PCSrc

### EXT

#### 主要功能

拓展16位数据为32位

### IFU

#### 主要功能

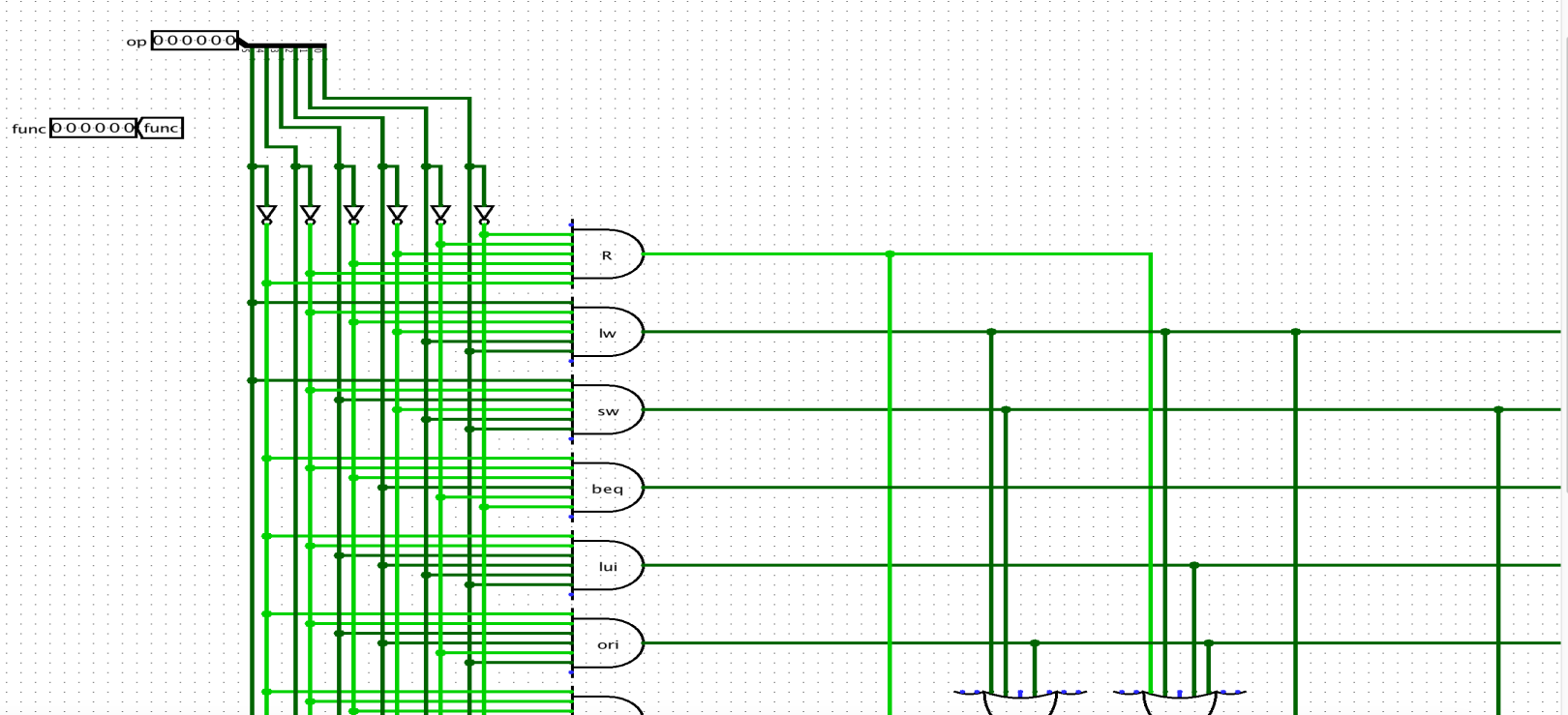
1. 复位，接受复位信号时PC设置为0x0。
2. 取指令，根据PC值从IM中取指令。
3. 计算下⼀条指令地址。

#### IM

##### 主要功能

根据地址值给出指令

### CTRL

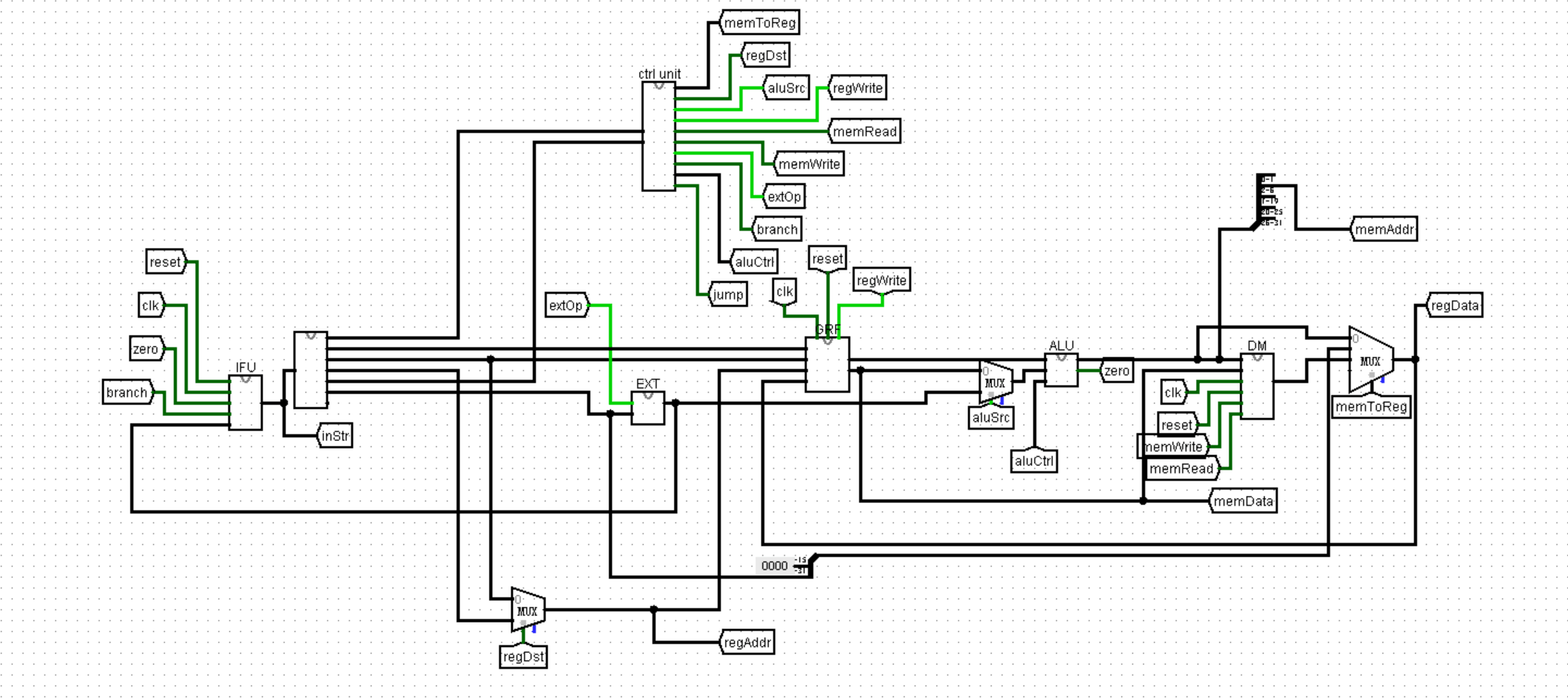
**主要功能** 产⽣控制信号

**ALUCTRL**

##### 主要功能

产⽣ALU控制信号

顶层设计



# 测试⽅案

将⼀段MIPS源码转为MARS代码，并在CPU上运⾏，然后对⽐模拟结果与我的数据

1 ori $t1, $zero, 0

2 ori $t2, $t1, 2

1. ori $t3, $zero, 1
2. ori $a0, $zero, 5
3. ori $a1, $zero, 3

6 addu $t1, $t1, $t3

7 addu $a0, $a0, $t3

8 subu $a1, $a1, $t3

9 beq $t1, $t2, next

10 addu $t1, $t1, $t3

1. beq $t1, $t2, next
2. next:subu $t1, $t1, $t3
3. lui $t4, 0xfff

14 lui $t6, 0x123

15 ori $t2, $zero, 4

16 sw $t6, 8($t2)

17 sw $t4, 0x14($t2)

18 lw $t5, 0x14($t2)

19 lw $t7, 8($t2)

注释：#测试ori

将 $t1 赋值为 0，$t2 赋值为 $t1 加上 2，$t3 赋值为 1。

将 $a0 赋值为 5，$a1 赋值为 3。

第一条我们先测最基本的，与 0 进行 or 运算，然后再测试两个非 0 数直接的 or 运算。从指令集的解释中，我们知道，ori 指令的第三个立即数参数是无符号扩展，所以就不存在负数的情况。

#测试addu

将 $t1 的值加上 $t3 的值，将 $a0 的值加上 $t3 的值

#测试subu

将 $a1 的值减去 $t3 的值。

#测试beq

如果 $t1 的值等于 $t2 的值，跳转到标签 "next"。

将 $t1 的值加上 $t3 的值，如果 $t1 的值等于 $t2 的值，跳转到标签 "next"。

标签 "next"：将 $t1 的值减去 $t3 的值。

#测试lui

将 $t4 赋值为 0xfff，将 $t6 赋值为 0x123。

将 $t2 赋值为 4。

#测试sw 和lw

将 $t6 的值存储到 $t2 加 8 的内存地址，将 $t4 的值存储到 $t2 加 0x14 的内存地址。

将 $t2 加 0x14 的内存地址的值加载到 $t5 中

机器码

1 34090000

2 352a0002

3 340b0001

4 34040005

5 34050003

6 012b4821

7 008b2021

8 00ab2823

9 112a0002

10 012b4821

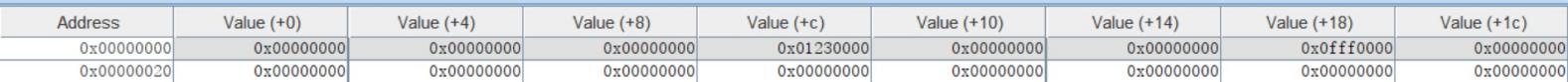
11 112a0000

12 012b4823

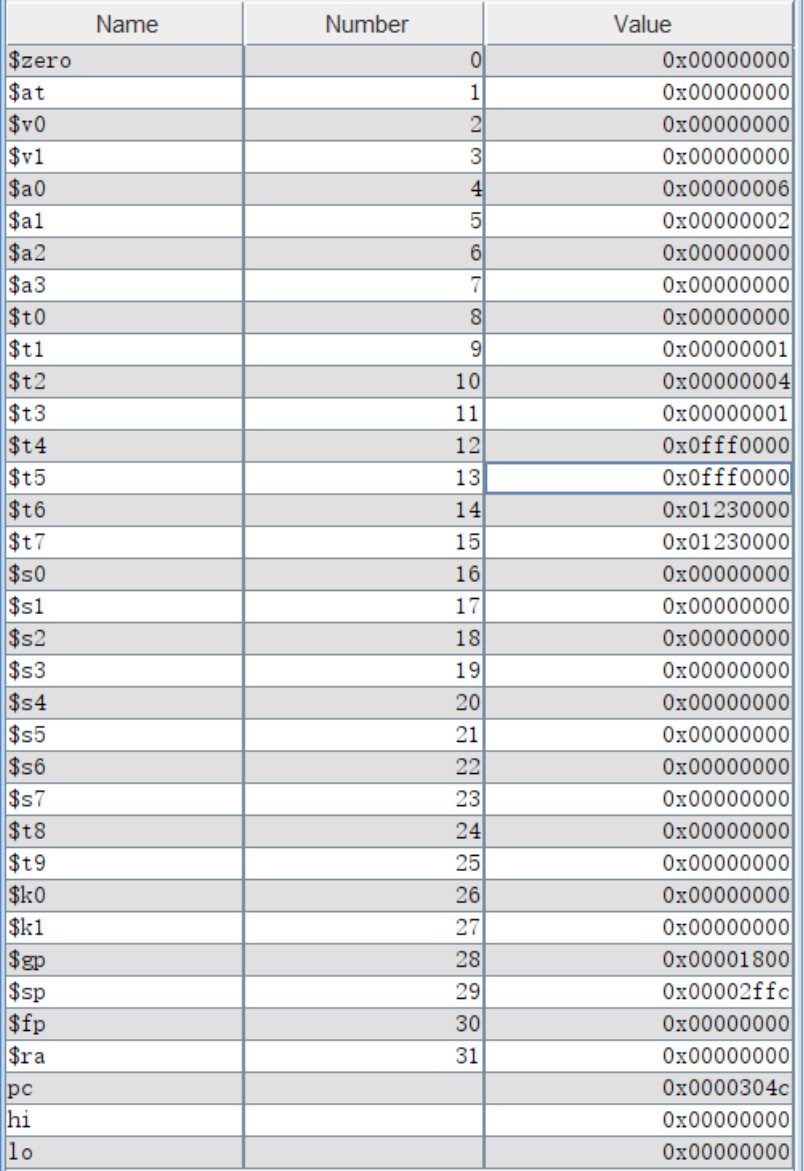
13 3c0c0fff 14 3c0e0123 15 340a0004 16 ad4e0008 17 ad4c0014 18 8d4d0014 19 8d4f0008

模拟器数据

DATA

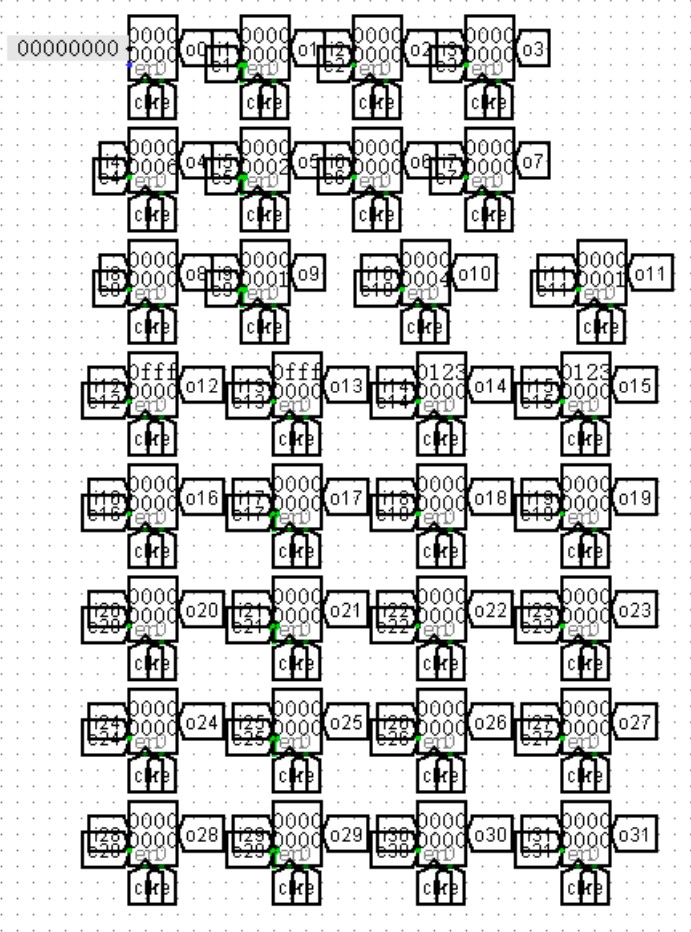


REG

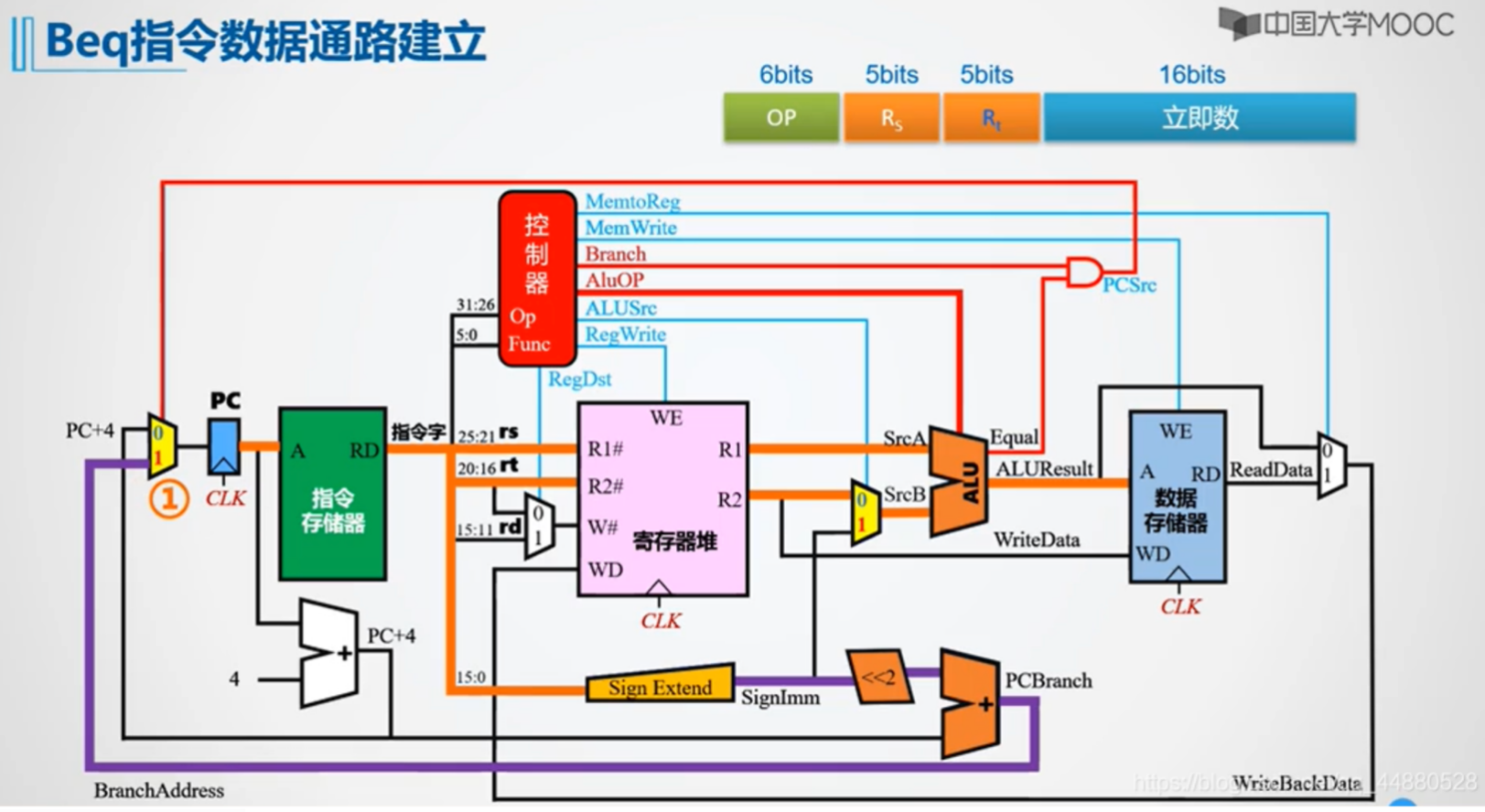
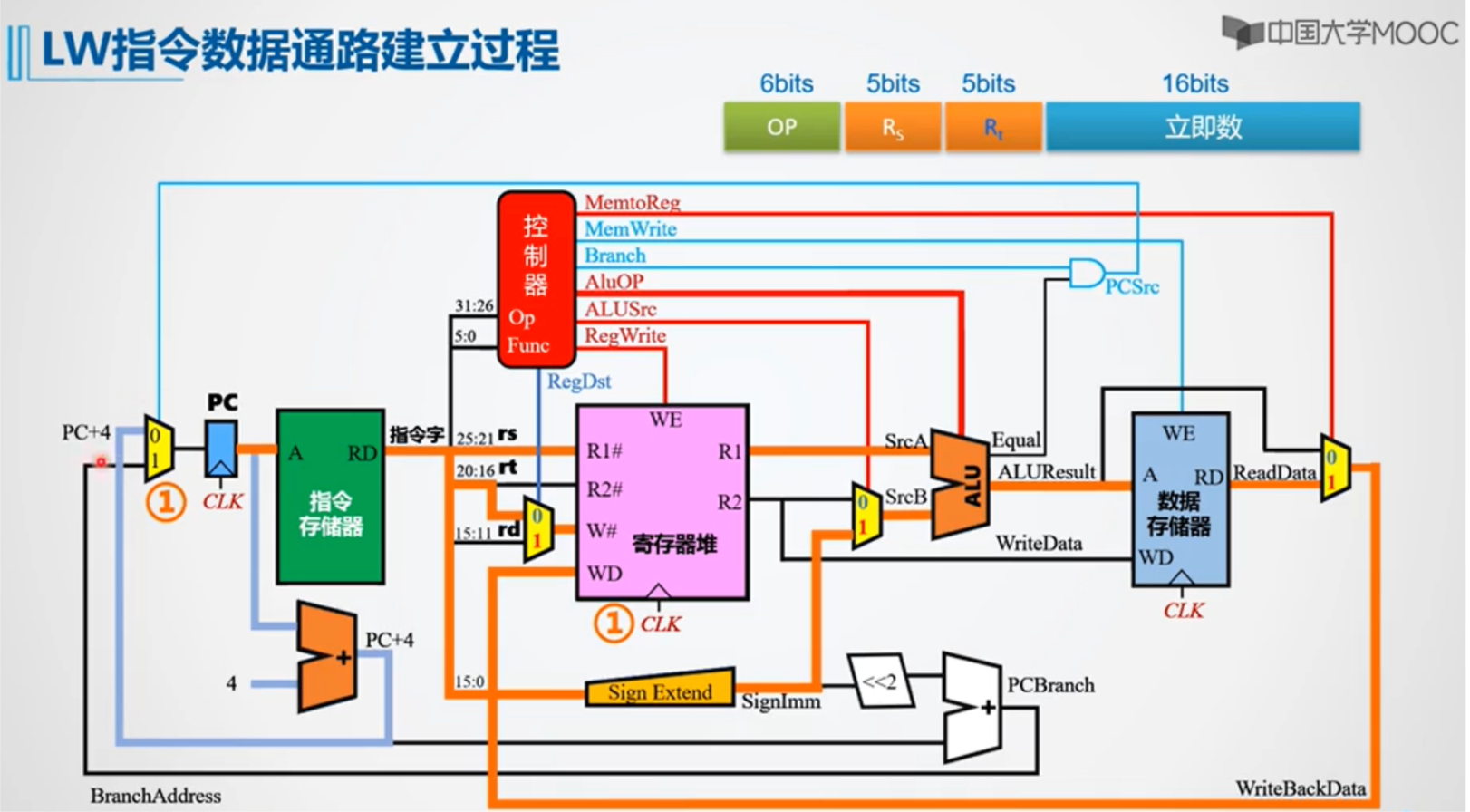
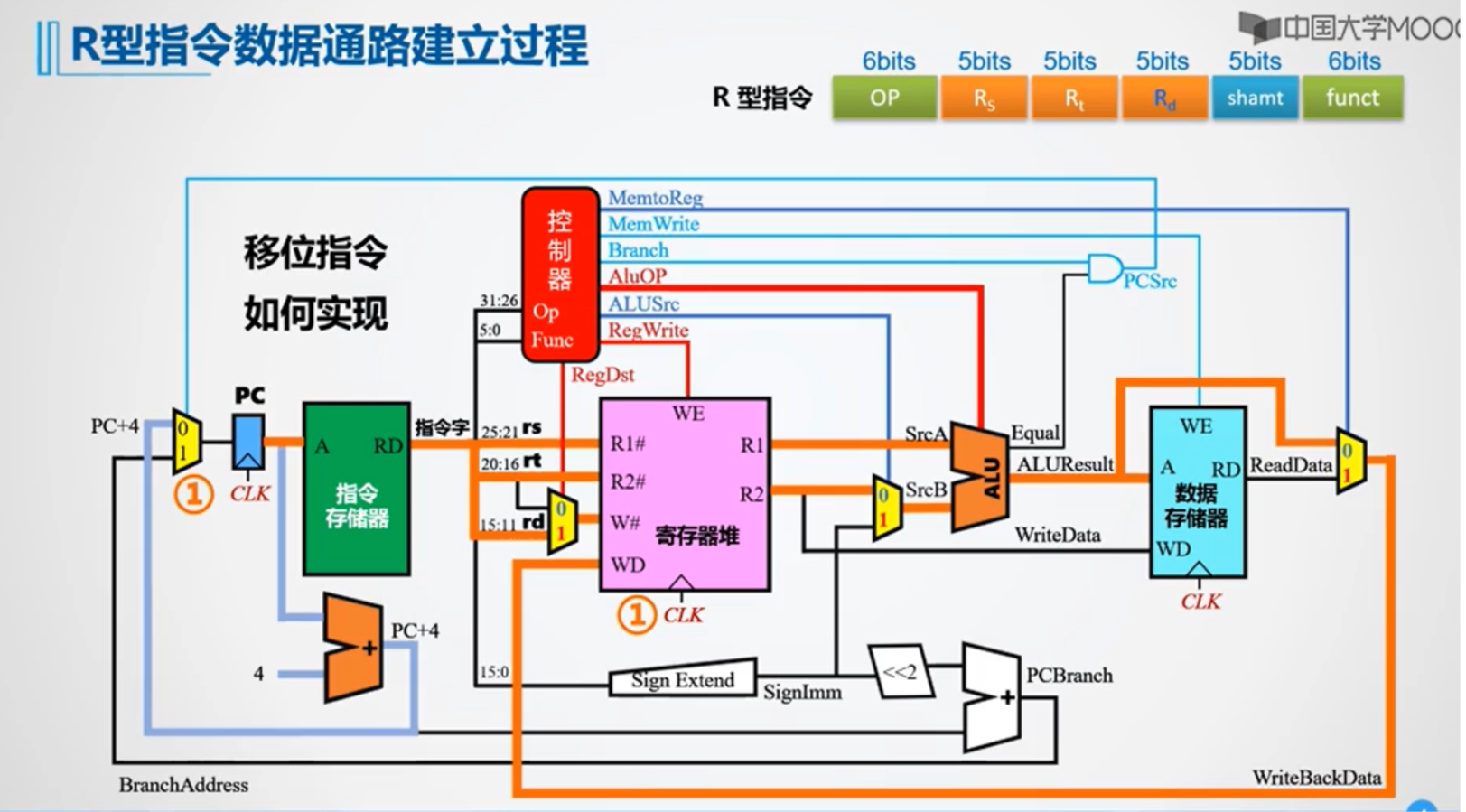


CPU运⾏结果DATA

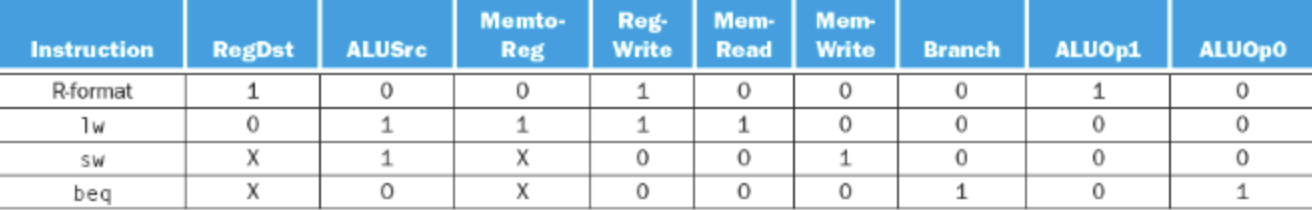
REG



各种类型的数据通路



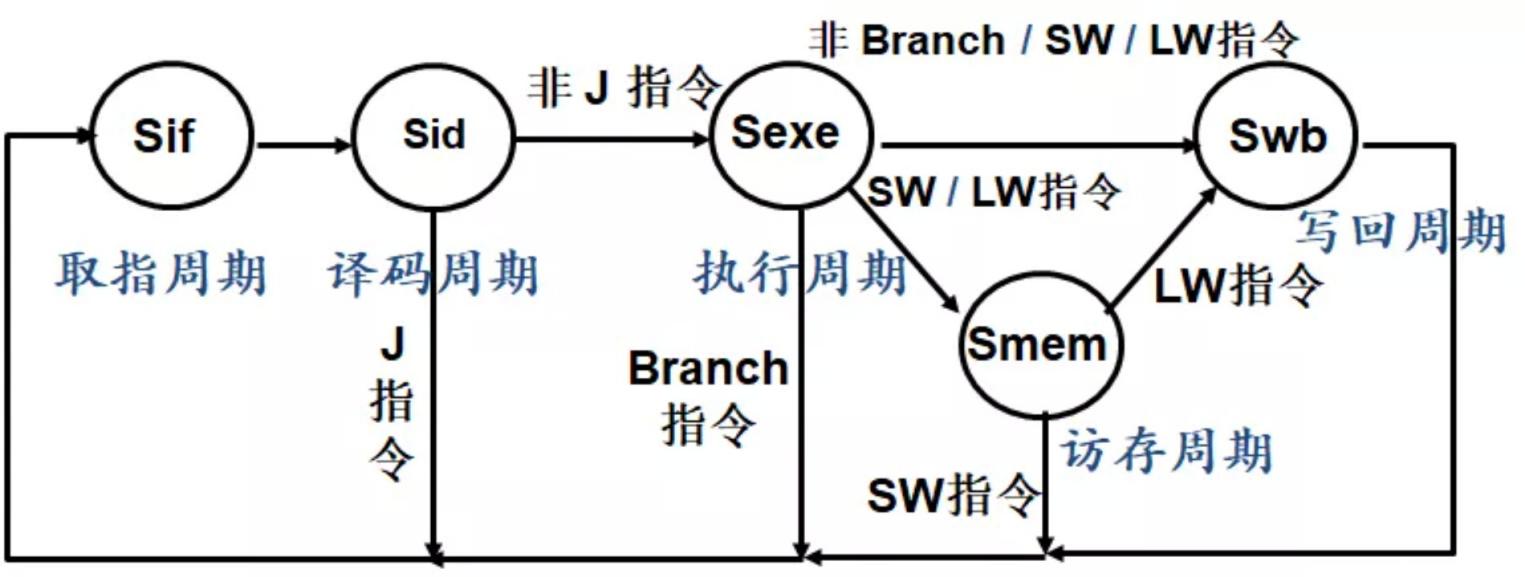
* 根据上面的三个图片小小注释：RegDst的作用在寄存器堆处，用于区分存取指令和R指令（0的时候 存取指令写入rt，1 的时候 R指令写入rd）
* RegWrite：寄存器堆可写入
* ALUScr：ALUSrc用于选择Read Data2或是Sign-extend后的offset立即数，0的时候第二个ALU操作数来自寄存器堆的第二个输出，1的时候第二个ALU操作数为指令低16位的符号扩展
* Memwrite：内存数据写入使能，用于sw指令
* MemtoReg：写入寄存器数据是否来自内存，用于lw指令，0的时候写入寄存器的数据来自ALU，1的时候写入寄存器的数据来自数据存储器
* PCScr：PCSrc即用于Branch Instructions
* MemRead：内存数据读取使能，用于lw指令



# 思考

### T1

状态存储功能主要由寄存器和锁存器来实现。寄存器用于存储指令的操作数，以及中间运算的结果，而锁存器则用于在指令执行过程中保存相关的控制信息。状态转移功能则主要由控制器来实现。控制器通过读取当前指令，并根据指令的类型和操作数来决定下一步的操作。



**T2**

ROM是只读存储器，适合存储固定不变的数据，⽽指令数据在单周期cpu运⾏过程有限个周期中不 变的，因此选⽤ROM是合理的。RAM是读写存储器，DM要求存储器可以读和写，且速度要求不⾼，因此选取速度不⾼但是功能齐全的RAM⽐较合理。GRF需要经常对数据进⾏读写且速度要求⽐较⾼，因此 选取Register⽐较合理。

### T3

额外增加了⼀个split模块，将IFU模块的输出内容在模块内部分割并输出，这样保证了顶层设计的美观性，并在拓 展IFU功能的时候可以简单的更改split模块⽽不是⼤幅度修改顶层电路

### T4

nop指令数据是0x0，除了PC=PC+4之外，没有进⾏任何其他操作，因此没有对电路中的逻辑真值运算产⽣任何影响，存在与否对电路⽆影响。也就是说它并没有任何有效的操作内容，由于 nop 指令只是一条空指令，它并不会对 CPU 的状态产生任何影响。因此，我们也无需在控制信号真值表中对它进行记录

### T5

假如DM存储⼤⼩是256MB，数据地址从0x10000000-1fffffff，则将要存储的地址的最⾼四位和 0x1进⾏⽐较，得到

⼀个⽚选信号，相同则将这个数据存储到DM中，否则将数据存储到其他相应的位置。

### T6

强度⼀般，在add指令中没有测试溢出情况，MIPS的⽂档中add指令应当具备对溢出的处理能⼒，对sub同理。 在sw和lw指令中我们并没有对错误的偏移量做额外的检测。