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

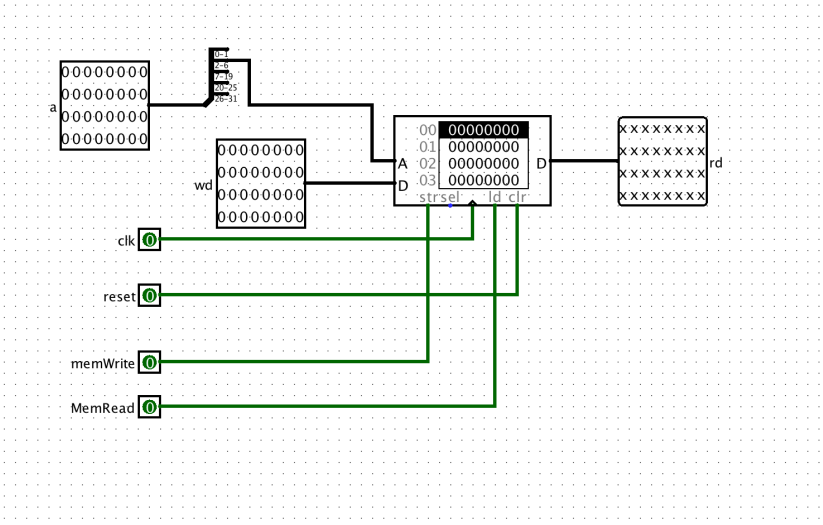
主要功能

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

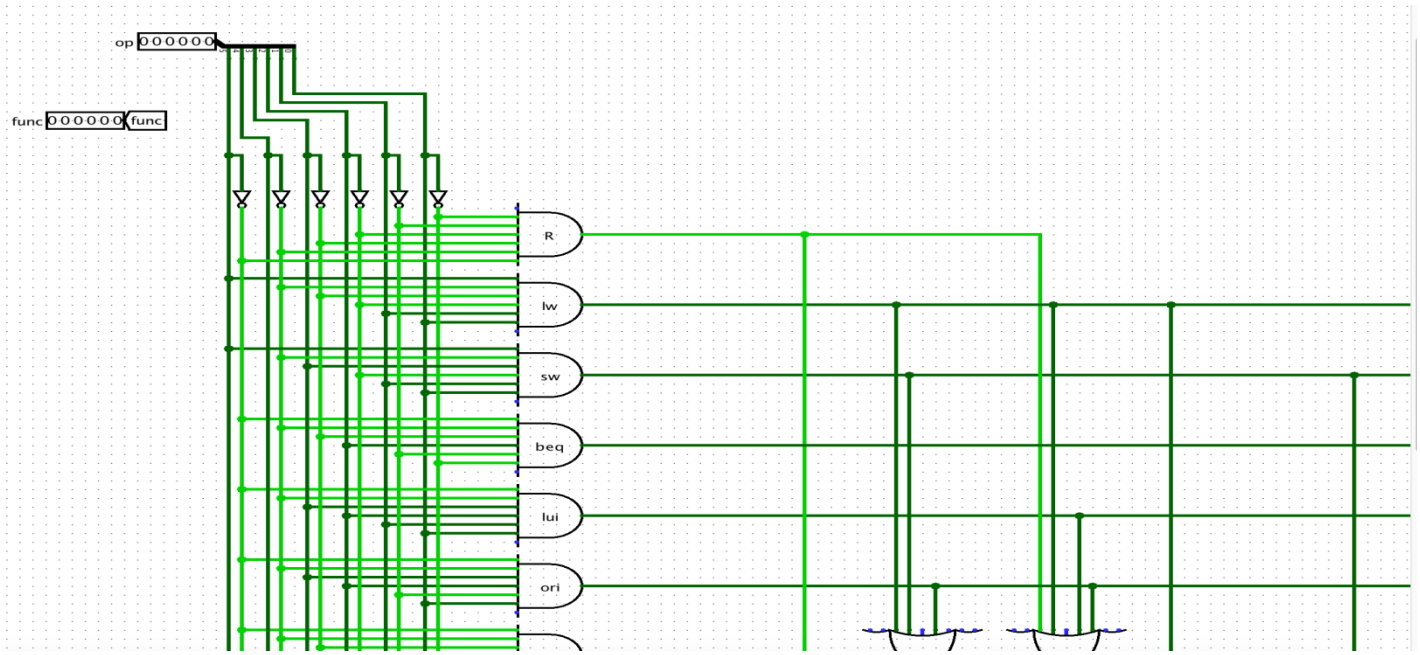
主要功能

根据地址值给出指令

CTRL

主要功能

产生控制信号

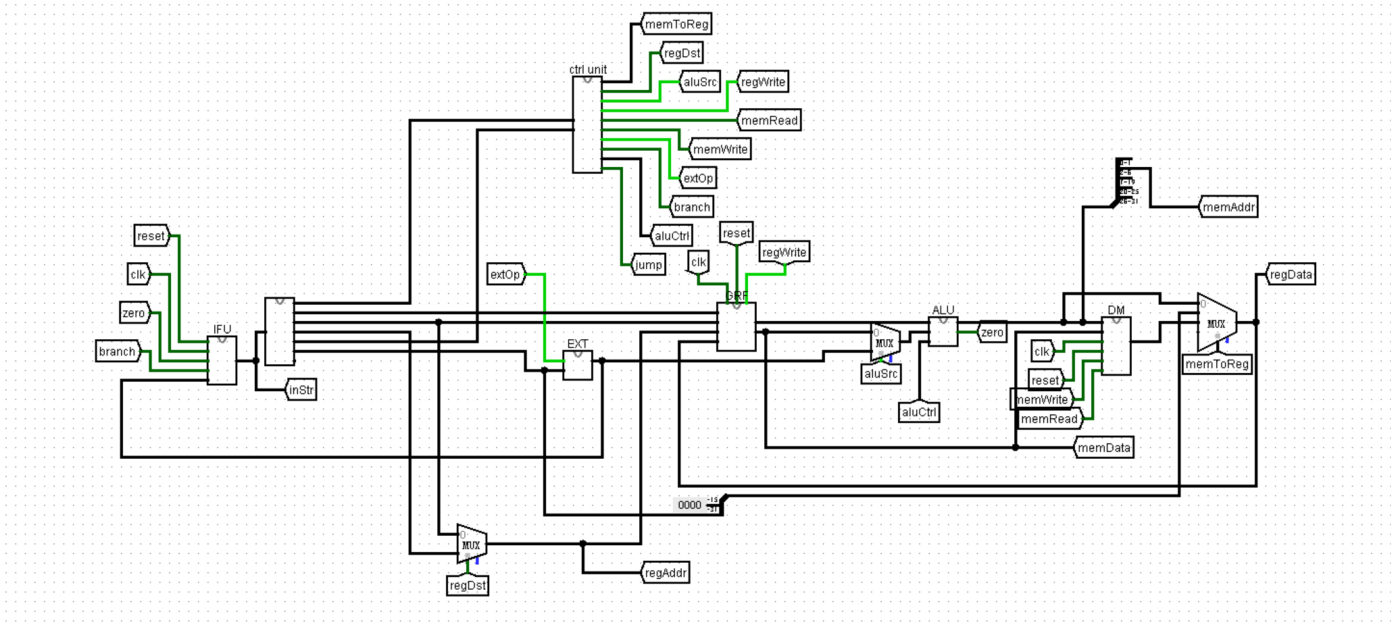


ALUCTRL

主要功能

产生ALU控制信号

顶层设计


memToReg
regDst
aluSrc
regWrite
memRead
memWrite
extOp
branch
aluCtrl
jump
ctrl unit
reset
clk
zero
branch
IFU
inStr
extOp
EXT
reset
regWrite
clk
GRF
MUX
aluSrc
ALU
zero
aluCtrl
DM
clk
reset
memWrite
memRead
memData
memAddr
MUX
memToReg
regData
0000
MUX
regDst
regAddr

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

```
ori $t1, $zero, 0
ori $t2, $t1, 2
ori $t3, $zero, 1
ori $a0, $zero, 5
ori $a1, $zero, 3
addu $t1, $t1, $t3
addu $a0, $a0, $t3
subu $a1, $a1, $t3
beq $t1, $t2, next
addu $t1, $t1, $t3
beq $t1, $t2, next
next:subu $t1, $t1, $t3
lui $t4, 0xfff
lui $t6, 0x123
ori $t2, $zero, 4
sw $t6, 8($t2)
sw $t4, 0x14($t2)
lw $t5, 0x14($t2)
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

```
34090000
352a0002
340b0001
34040005
34050003
012b4821
008b2021
00ab2823
112a0002
012b4821
112a0000
012b4823
3c0c0fff
3c0e0123
```

模拟器数据

DATA

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00000000	0x00000000	0x01230000	0x00000000	0x00000000	0x0fff0000	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

REG

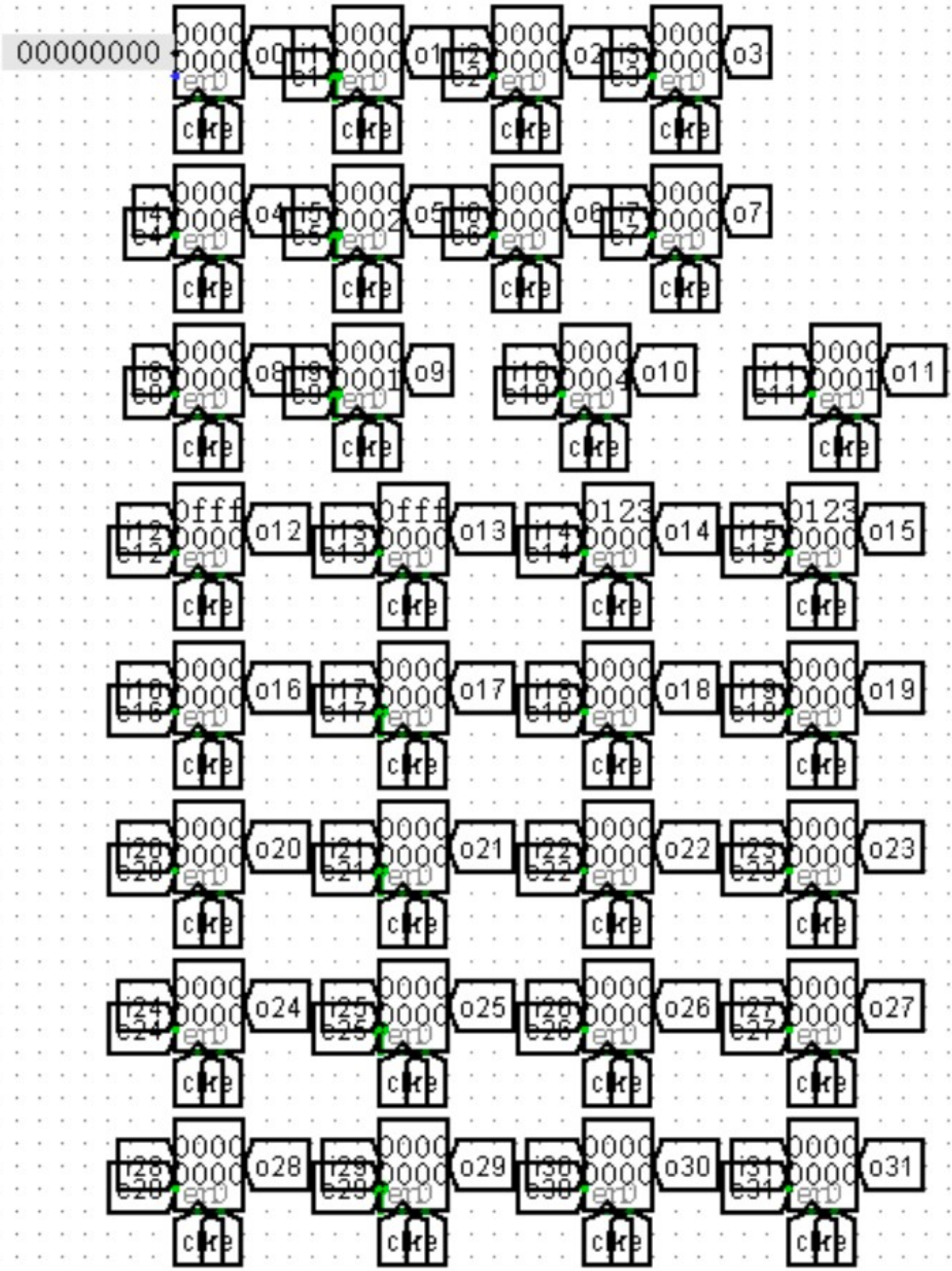
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000006
$a1	5	0x00000002
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000001
$t2	10	0x00000004
$t3	11	0x00000001
$t4	12	0x0fff0000
$t5	13	0x0fff0000
$t6	14	0x01230000
$t7	15	0x01230000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x0000304c
hi		0x00000000
lo		0x00000000

CPU运行结果

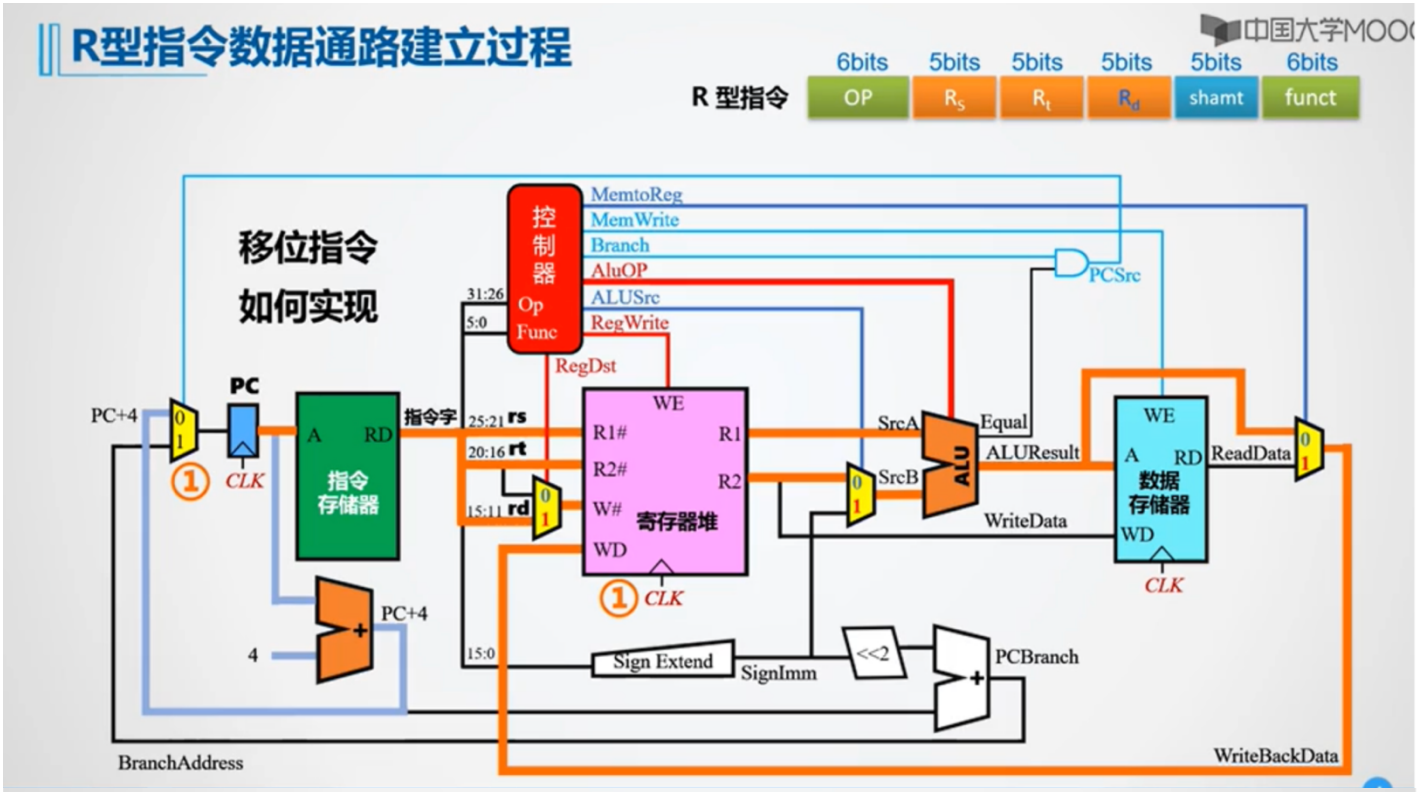
DATA

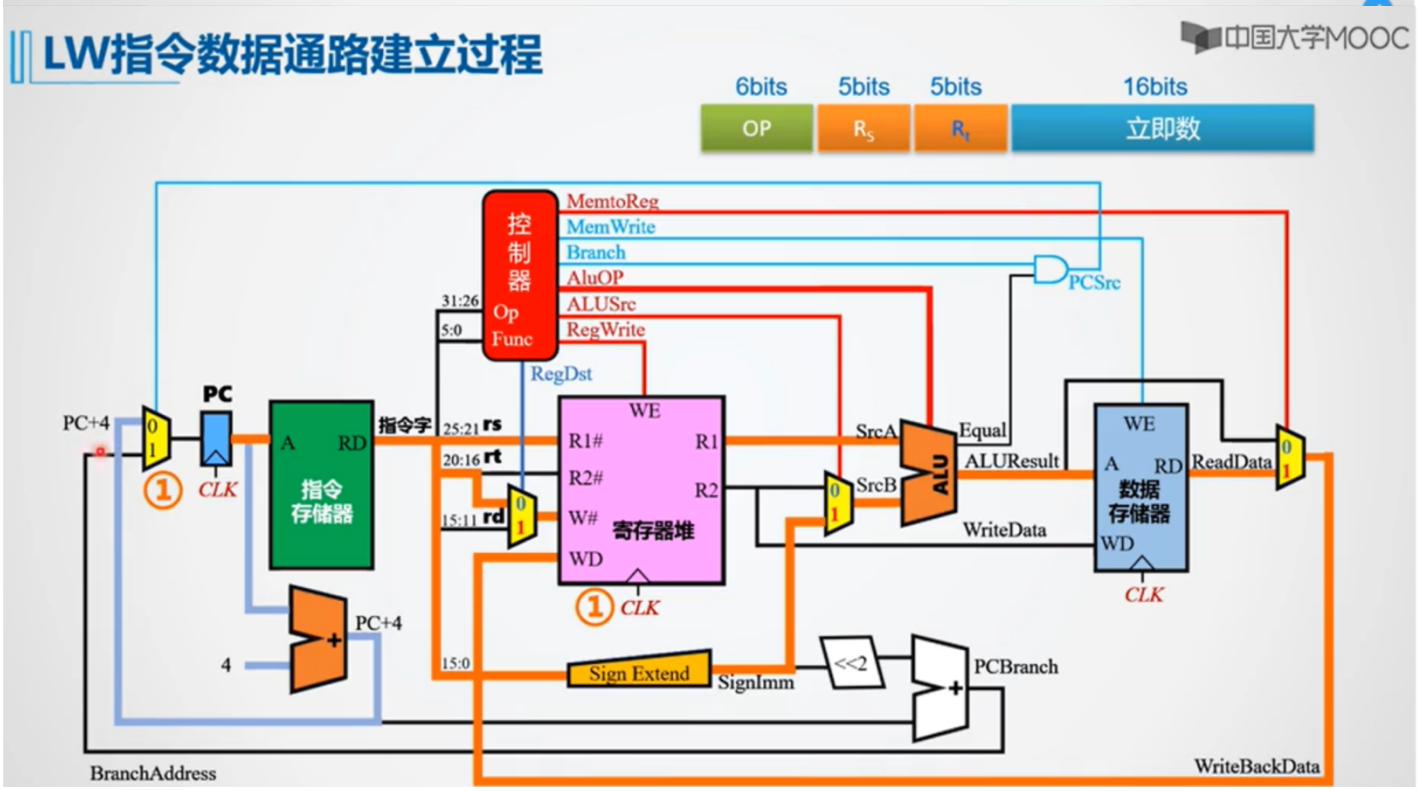
00	00000000	00000000	00000000	01230000	00000000	00000000	0fff0000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
10	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

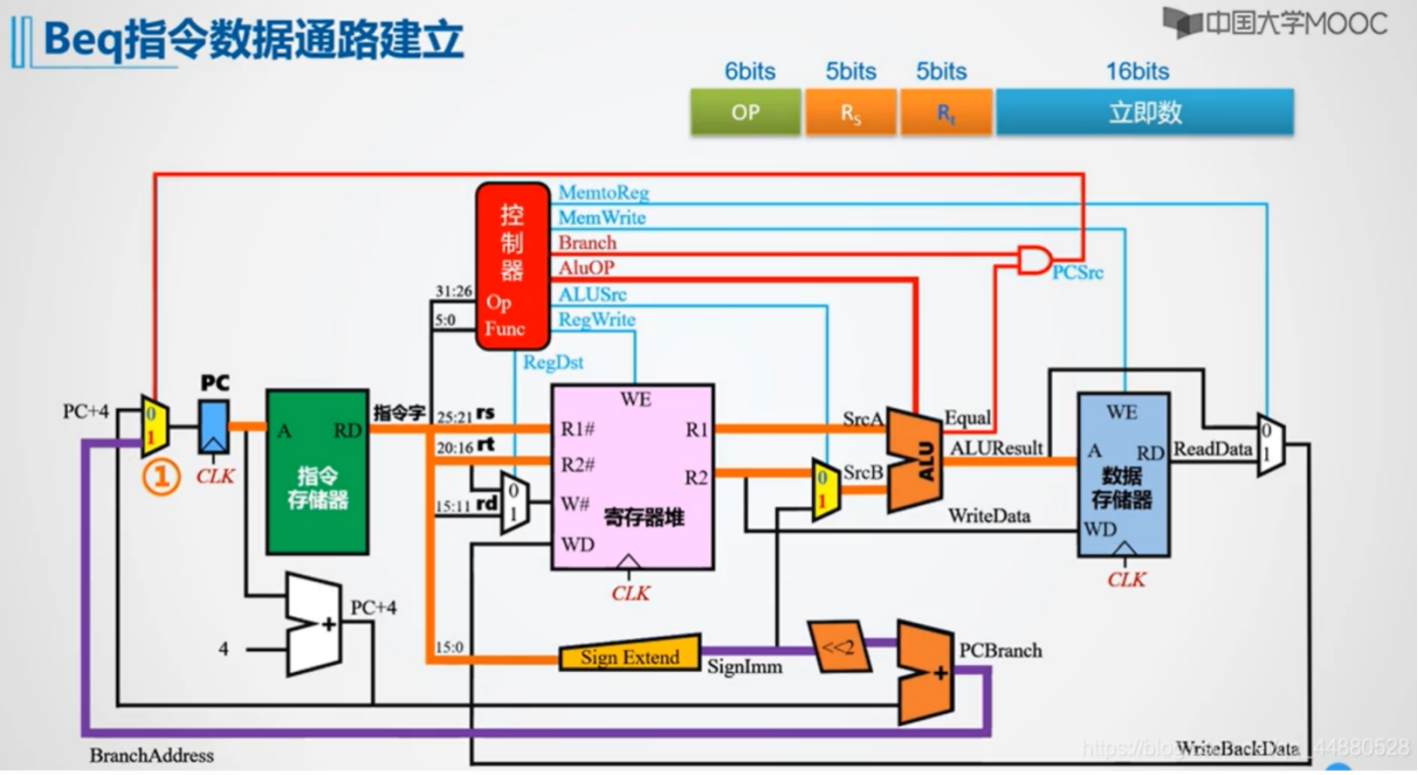
REG


00000000
o0
o1
o2
o3
o4
o5
o6
o7
o8
o9
o10
o11
o12
o13
o14
o15
o16
o17
o18
o19
o20
o21
o22
o23
o24
o25
o26
o27
o28
o29
o30
o31
clk
en

各种类型的数据通路


R型指令数据通路建立过程
中国大学MOOC
R 型指令
6bits 5bits 5bits 5bits 5bits 6bits
OP Rs Rt Rd shamt funct
移位指令
如何实现
控制器
Op
Func
MemtoReg
MemWrite
Branch
AluOP
ALUSrc
RegWrite
RegDst
PCSrc
31:26
5:0
PC
PC+4
指令存储器
A
RD
指令字
25:21 rs
20:16 rt
15:11 rd
寄存器堆
WE
R1#
R2#
W#
WD
R1
R2
CLK
SrcA
SrcB
ALU
Equal
ALUResult
WriteData
数据存储器
ReadData
4
15:0
Sign Extend
SignImm
<<2
PCBranch
BranchAddress
WriteBackData


LW指令数据通路建立过程
中国大学MOOC
6bits 5bits 5bits 16bits
OP Rs Rt 立即数
控制器
Op
Func
MemtoReg
MemWrite
Branch
AluOP
ALUSrc
RegWrite
RegDst
PCSrc
31:26
5:0
PC
PC+4
指令存储器
A
RD
指令字
25:21 rs
20:16 rt
15:11 rd
寄存器堆
WE
R1#
R2#
W#
WD
R1
R2
CLK
SrcA
SrcB
ALU
Equal
ALUResult
WriteData
数据存储器
ReadData
4
15:0
Sign Extend
SignImm
<<2
PCBranch
BranchAddress
WriteBackData



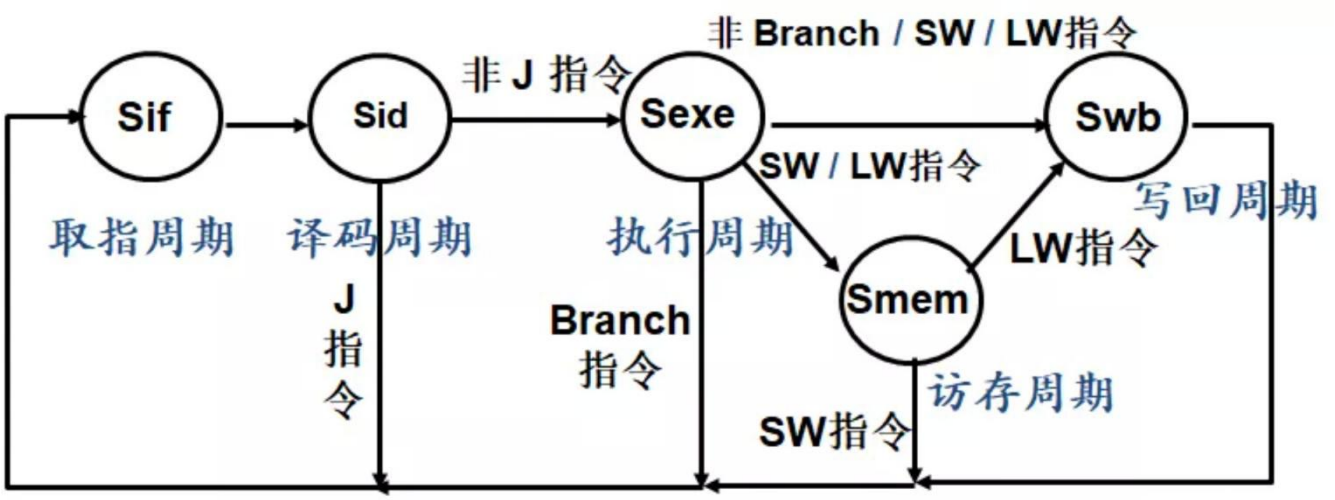
- 根据上面的三个图片小小注释：RegDst的作用在寄存器堆处，用于区分存取指令和R指令（0的时候 存取指令写入rt，1 的时候 R指令写入rd）
- RegWrite：寄存器堆可写入
- ALUScr：ALUSrc用于选择Read Data2或是Sign-extend后的offset立即数，0的时候第二个ALU操作数来自寄存器堆的第二个输出，1的时候第二个ALU操作数为指令低16位的符号扩展
- Memwrite：内存数据写入使能，用于sw指令
- MemtoReg：写入寄存器数据是否来自内存，用于lw指令，0的时候写入寄存器的数据来自ALU，1的时候写入寄存器的数据来自数据存储器
- PCScr：PCSrc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

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

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。