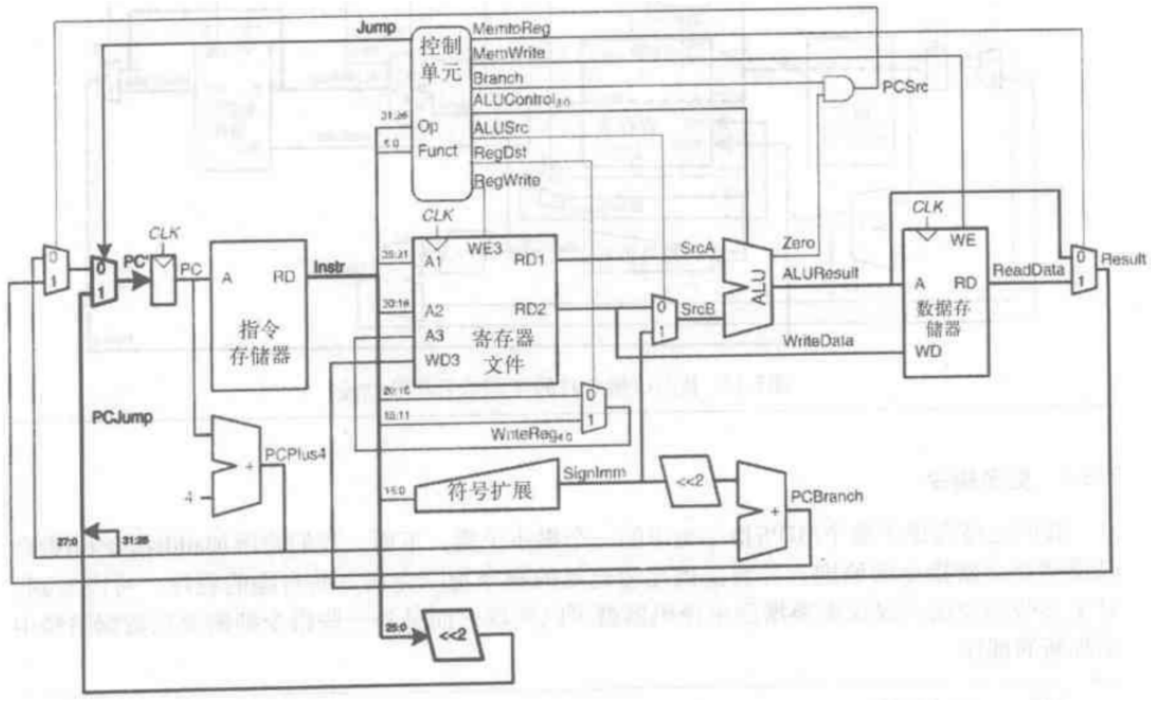
P3 单周期CPU设计文档

设计概述

- 设计的处理器为32位单周期处理器
- 处理器支持的指令集为`add, sub, ori, lw, sw, beq, lui, nop, j, jal, jr, lb, sb, lh, sh`等
- `nop`为空指令，其机器码为`0x00000000`，不进行任何有效行为，如修改寄存器等
- `add, sub`按无符号加减法处理，不考虑溢出

顶层设计

参考了《数字设计与计算机体系结构》图7-14，在实际设计上略有改动。



模块定义

IFU(取指令单元)

内部包括 PC（程序计数器）、IM（指令存储器）及相关逻辑。

PC 用寄存器实现，应具有**异步复位**功能，复位值为起始地址。

起始地址：0x00003000。

地址范围：0x00003000 ~ 0x00006FFF。

IM用ROM实现，容量为4096 × 32bit。

IM实际地址宽度仅为12位，需要使用恰当的方法将PC中储存的地址同IM联系起来。

> PC的处理方法：

- PC的变化范围为0x00003000 ~ 0x00006FFF，考虑使用PC' = PC - 0x00003000，则PC'的范围为0x00000000-0x00003FFF，不仅保证PC和PC'在数值上一一对应，而且在设计处理时更加方便。
- 注意，输出是需要输出PC的值，而不是PC'。
- IM的实际地址宽度为12位，而PC的有效位数（可能发生变化的位数）为低14位。因为ROM是按字寻址，在从IM读取指令时只需要用PC[13:2]作为地址，就可以正确读取数据。

端口定义

表2-1-1 IFU模块端口定义

信号名	方向	描述
clk	I	时钟信号
reset	I	**异步**复位信号，将PC置0
offset[15:0]	I	beq等branch指令的偏移量，即Instr[15:0]
imm[25:0]	I	j指令和jal指令中的立即数，即Instr[25:0]
PCsel[2:0]	I	指定更新PC的方式
Instr[31:0]	O	输出IM中PC地址上的指令
PC	O	输出当前PC的值
PC+4	O	输出PC+4的值

功能定义

表2-1-2 IFU模块功能定义

序号	功能	描述
1	异步复位	reset置1时，将PC置为0x00003000
2	更新下一个PC的值	时钟上升沿来临时，更新PC的值 PCsel为2'b00时，PC ← PC + 4 PCsel为2'b01时，PC ← PC + 4 + sign_extend(offset||0^2) PCsel为2'b10时，PC ← PC[31:28]||imm||0^2 PCsel为2'b11时，PC ← PC[31:28]||imm||0^2; GPR[31] ← PC + 4

GRF（寄存器文件）

使用**具有写使能功能**的寄存器实现，寄存器总数为**32个**，具有**异步复位**功能。

其中，**0号寄存器**($zero)的值始终保持为0。其他的寄存器**初始值（复位后）均为0**，无需专门设置。

端口定义

表2-2-1 GRF模块端口定义

信号名	方向	描述
clk	I	时钟信号
reset	I	**异步**复位信号，将32个寄存器中的值全部清零 1:复位 0:无效
WE	I	写使能信号 1:可向GRF中写入数据 0:不能向GRF中写入数据
A1[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1
A2[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2
A3[4:0]	I	5位地址输入信号，指定32个寄存器中的一个将WD中的数据写入
WD[31:0]	I	32位数据输入信号
RD1[31:0]	O	输出A1指定的寄存器中的32位数据
RD2[31:0]	O	输出A2指定的寄存器中的32位数据

功能定义

表2-2-2 GRF模块功能定义

序号	功能	描述
1	异步复位	reset信号置1时，所有寄存器存储的数值清零，其行为与logisim自带部件register的reset接口完全相同
2	读数据	读出A1，A2地址对应寄存器中所存储的数据到对应的RD1，RD2
3	写数据	当**WE有效且时钟上升沿来临**时，将WD写入A3所对应的寄存器中

ALU(算术逻辑单元)

提供 32 位加、减、或运算及大小比较功能。

加减法按无符号处理（不考虑溢出）。

端口定义

表2-3-1 ALU模块端口定义

信号名	方向	描述
A[31:0]	I	第一个32位计算数
B[31:0]	I	第二个32位计算数
ALUop[2:0]	I	指定ALU进行的计算
res[31:0]	O	运算结果
comp[2:0]	O	输出A与B的大小关系

功能定义

表2-3-2 ALU模块功能定义

序号	功能	描述
1	比较大小	A > B时，comp=2'b00 A = B时，comp=2'b01 A < B时，comp=2'b10
2	加运算	ALUop = 3'b000时，res = A + B，不考虑溢出
3	减运算	ALUop = 3'b001时，res = A - B，不考虑溢出
4	或运算	ALUop = 3'b010时，res = A | B
5	B置高16位	ALUop = 3'b011时，res = B || 10^{16}

> 多余的ALUop为扩展指令预留。

DM（数据存储器）

使用RAM实现，容量为3072 × 32bit，应具有**异步复位**功能，复位值为0x00000000。

起始地址：0x00000000。

地址范围：0x00000000 ~ 0x00002FFF。

RAM 应使用**双端口模式**，即设置 RAM 的 **Data Interface** 属性为 **Separate load and store ports**。

端口定义

表2-4-1 DM模块端口定义

信号名	方向	描述
clk	I	时钟信号
reset	I	**异步**复位信号，将DM内的RAM重置为0
WE	I	写使能信号，WE为1时，允许写入数据；WE为0时，禁止写入
DMop[1:0]	I	指定DM进行的读/写操作类型
A[31:0]	I	需要进行读/写操作的地址
WD[31:0]	I	写入RAM的32位输入数据
RD[31:0]	O	从RAM读出的32位输出数据

功能定义

表2-4-2 DM模块功能定义

序号	功能	描述
1	异步复位	reset置1时，异步重置RAM内存为0
2	写数据	当**WE有效且时钟上升沿到来**时，将WD中的数据写入A对应的RAM地址中 DMop为2'b00时，执行lw指令 DMop为2'b01时，执行lh指令 DMop为2'b10时，执行lb指令
3	读数据	读取A对应的RAM地址中存储的数据到RD DMop为2'b00时，执行sw指令 DMop为2'b01时，执行sh指令 DMop为2'b10时，执行sb指令

> 与处理IFU中地址的方法相同，使用A[13:2]即可从DM的RAM中正确读取数据。

EXT(扩展单元)

使用Logisim内置的Bit Extender。

端口定义

表2-5-1 EXT模块端口定义

信号名	方向	描述
num[15:0]	I	需要扩展的16位立即数
sel	I	指定进行扩展的方式
result[31:0]	O	扩展完成的32位数

功能定义

表2-5-2 EXT模块功能定义

sel	功能	描述
1'b0	零扩展	result = zero_extend(num)
1'b1	符号扩展	result = sign_extend(num)

Controller(控制器)

使用与或门阵列构造控制信号。

和逻辑的功能是**识别**，将输入的机器码识别为相应的指令；或逻辑的功能是**生成**，根据输入的指令的不同，产生不同的控制信号。

端口定义

表2-6-1 Controller模块端口定义

信号名	方向	描述
op[5:0]	I	32位指令Instr[31:26]
comp[1:0]	I	ALU中两运算数的大小，决定是否执行branch指令
funct[5:0]	I	32位指令Instr[5:0]
RegDst[1:0]	O	指定数据写入GRF的寄存器序号 RegDst为2'b00时，序号为Instr[20:16]，对应I型指令的rt RegDst为2'b01时，序号为Instr[15:11]，对应R型指令的rd RegDst为2'b10时，序号为31，即**$ra**的序号，用于jal指令
ALUSrc	O	指定ALU第二个运算数是否是立即数 ALUSrc为0时，运算数来自GRF ALUSrc为1时，运算数为立即数
MemToReg[1:0]	O	指定写入GRF的数据的来源 MemToReg为2'b00时，数据为ALU的输出res MemToReg为2'b01时，数据为DM的输出RD MemToReg为2'b10时，数据为PC + 4，用于jal指令将PC + 4写入$ra的操作
RegWrite	O	是否可向GRF中写入数据
MemWrite	O	是否可向DM中写入数据
PCsel[1:0]	O	指定更新PC的方式
Extop	O	指定EXT进行立即数扩展的方式 ExtOp为0时，EXT进行零扩展 ExtOp为1时，EXT进行符号扩展
DMop[1:0]	O	指定操作DM的方式
ALUop[2:0]	O	指定ALU进行的计算

功能定义

表2-6-2 Controller模块功能定义

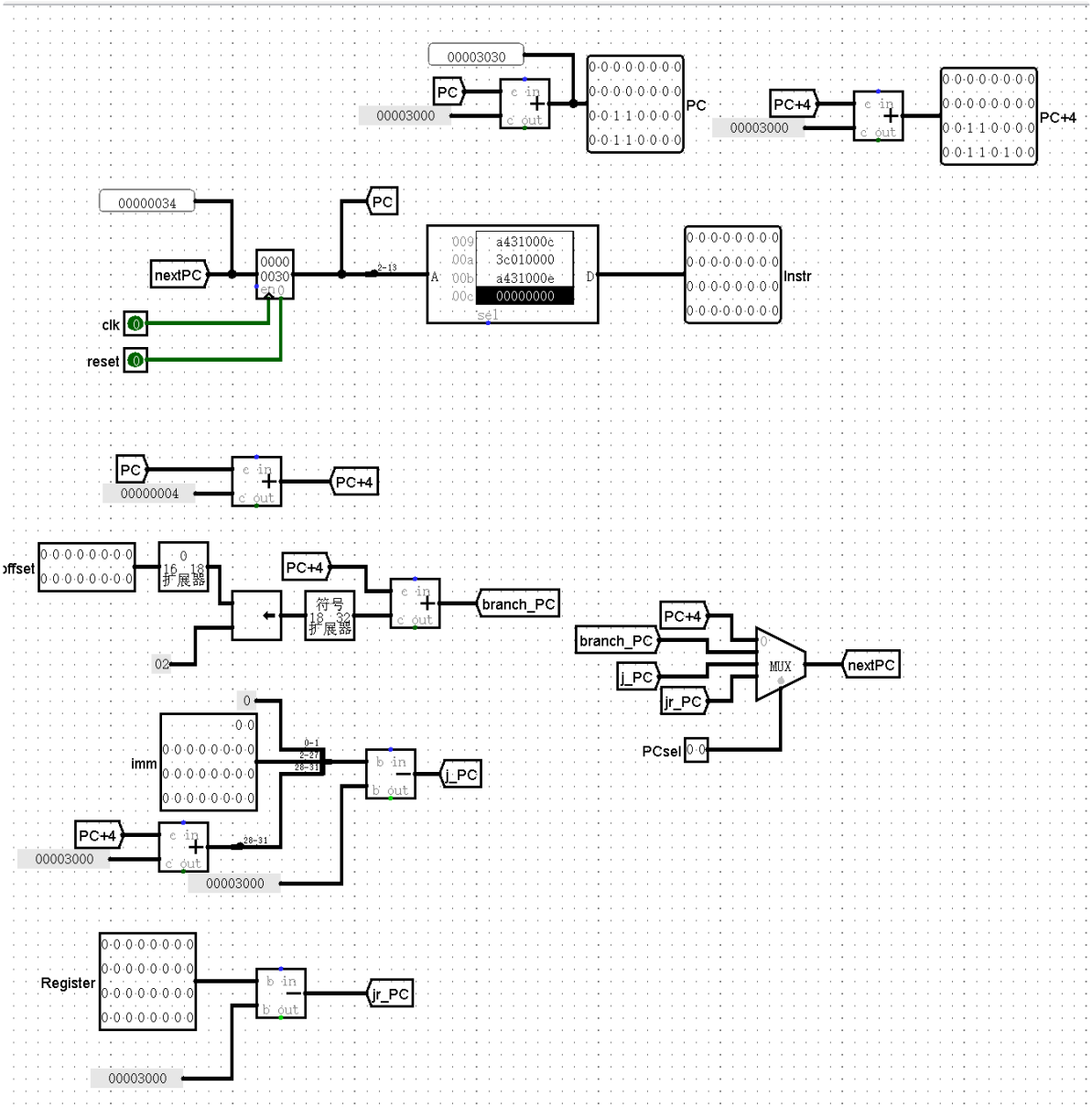
序号	功能	描述
1	生成控制信号	生成控制信号

重要机制实现方法

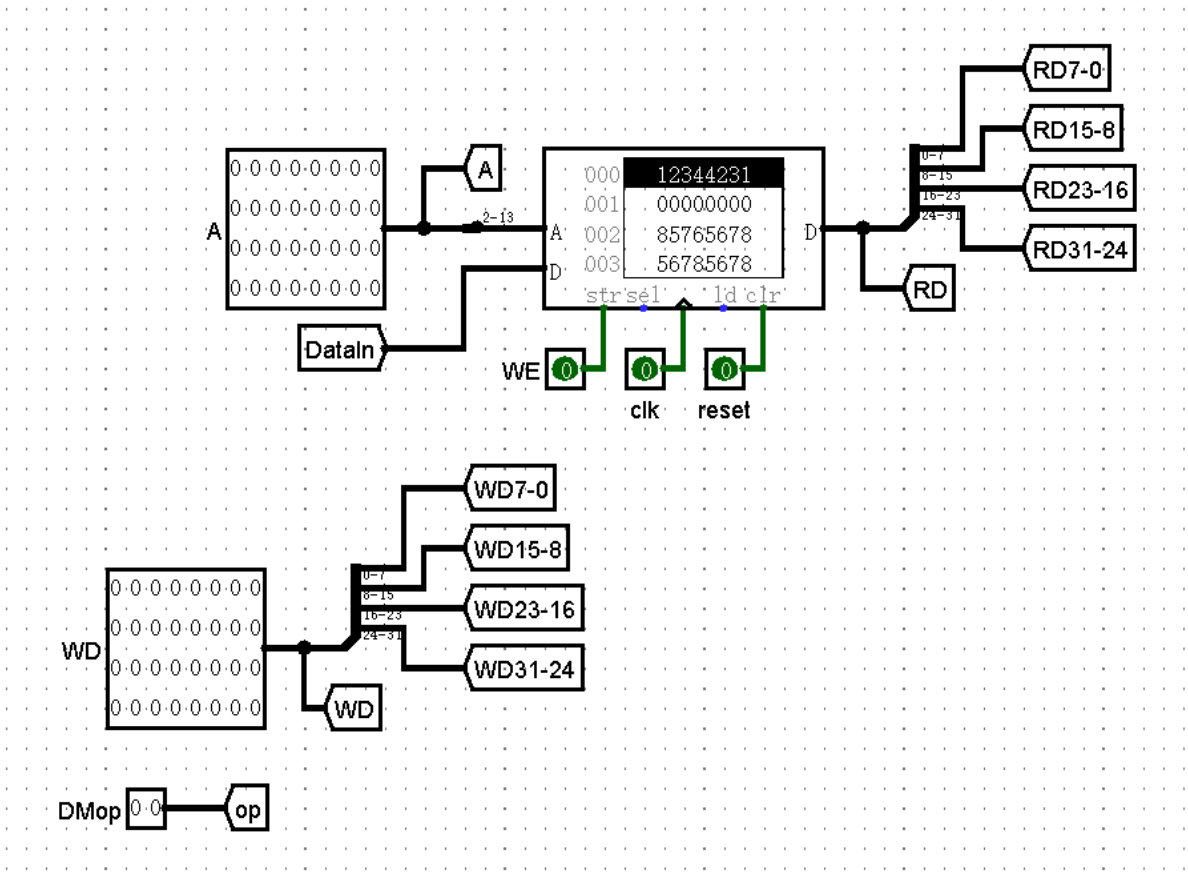
生成控制信号

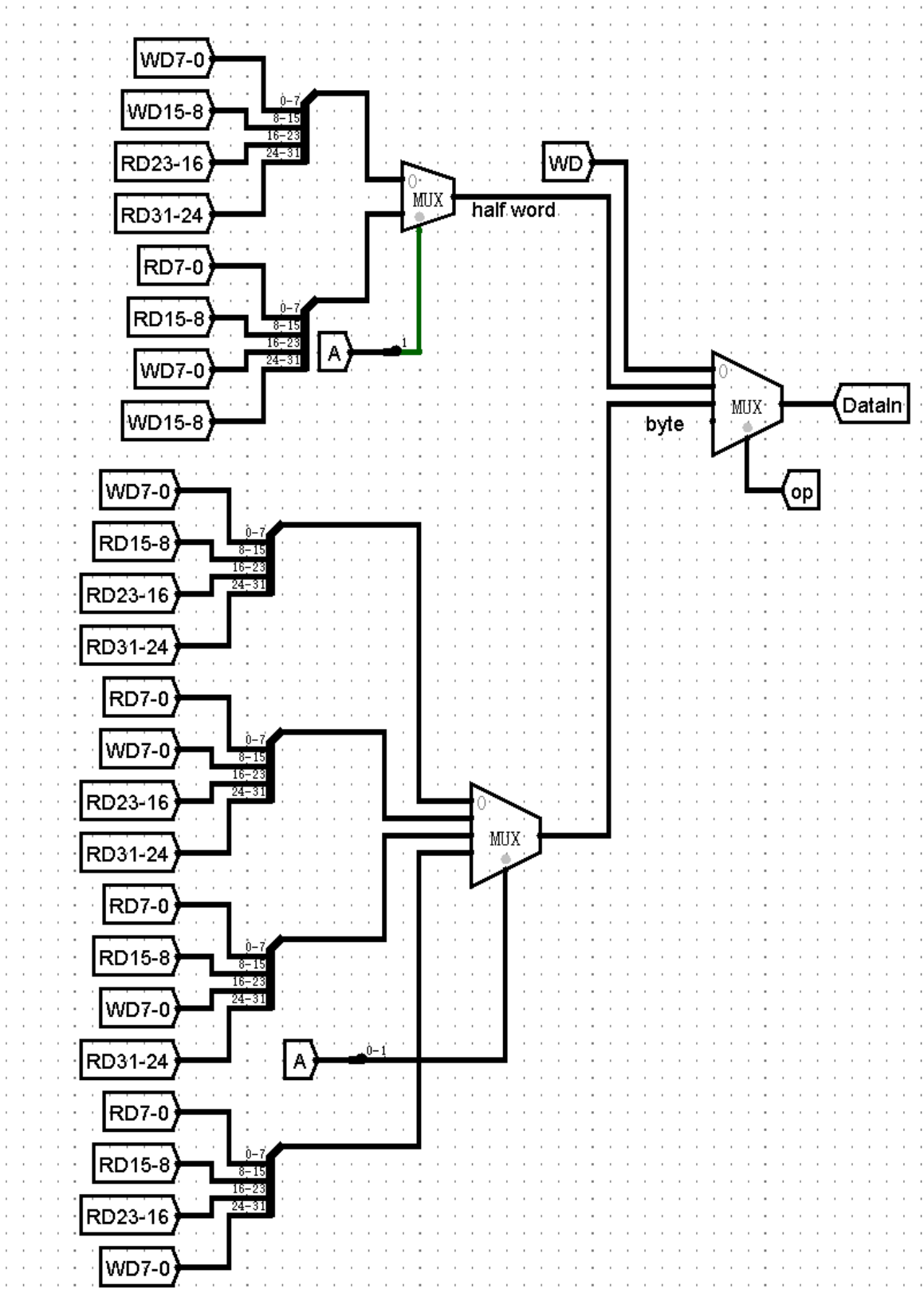
	add	sub	ori	lw	sw	beq	lui	j	jal	jr	lb	sb	lh	sh
funct	**100000**	**100010**	**undefined**							**001000**	**undefined**			
op	000000	000000	001101	100011	101011	000100	001111	000010	000011	000000	100000	101000	100000	101000
RegDst	1	1	0	0	x	x	0	x	2	x	0	x	0	x
ALUSrc	0	0	1	1	1	0	1	x	x	x	1	1	1	1
MemToReg	0	0	0	1	x	x	0	x	2	x	1	x	1	x
RegWrite	1	1	1	1	0	0	1	0	1	0	1	0	1	0
MemWrite	0	0	0	0	1	0	0	0	0	0	0	1	0	1
PCsel	0	0	0	0	0	1	0	2	2	3	0	0	0	0
Extop	x	x	0	1	1	x	x	x	x	x	1	1	1	1
DMop	x	x	x	0	0	x	x	x	x	x	2	2	2	2
ALUop	0	1	2	0	0	1	3	x	x	x	0	0	1	1

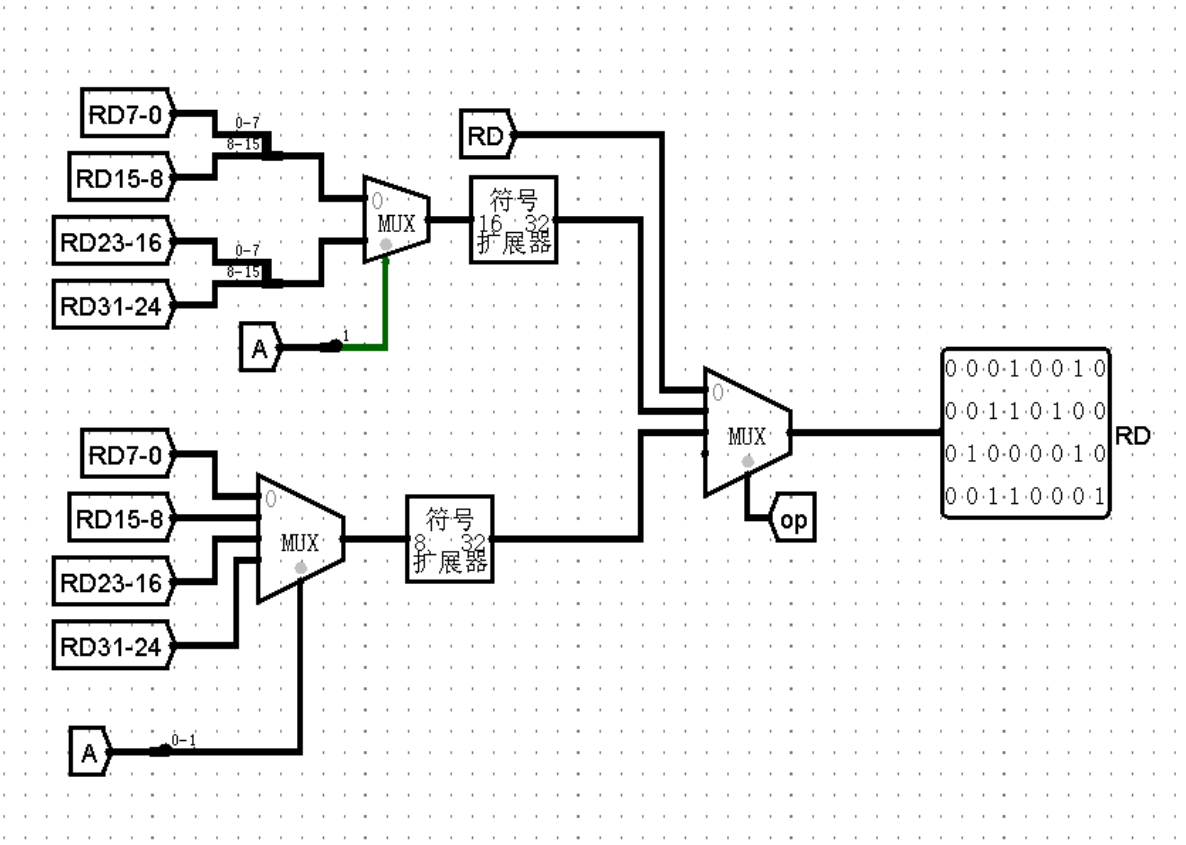
程序计数器



DM中对读写半字/字节的处理


A
2-13
A
D
DataIn
000 12344231
001 00000000
002 85765678
003 56785678
str sel ld clr
WE
clk
reset
D
RD7-0
RD15-8
RD23-16
RD31-24
RD
0-7
8-15
16-23
24-31
WD
WD7-0
WD15-8
WD23-16
WD31-24
WD
DMop
op


WD7-0
WD15-8
RD23-16
RD31-24
RD7-0
RD15-8
WD7-0
WD15-8
0-7
8-15
16-23
24-31
MUX
A
1
half word
WD
byte
MUX
DataIn
op
WD7-0
RD15-8
RD23-16
RD31-24
RD7-0
WD7-0
RD23-16
RD31-24
RD7-0
RD15-8
WD7-0
RD31-24
RD7-0
RD15-8
RD23-16
WD7-0
MUX
A
0-1



测试方案

测试代码1

```
ori $t0,$0,156
ori $t2,$0,135
ori $a3,$a3,1035
lui $a1,101
ori $a1,$0,2211
nop
loop:
beq $t3,$t2,end
ori $t4,8
lui $s6,170
add $t3,$t3,$t4
add $t3,$t2,$0
lw $s0,4($t1)
out:
add $t2,$t2,$t1
sub $t3,$t2,$0
beq $t3,$t2,loop
end:
lui $v0,11111
```

导出为

```
v2.0 raw
3408009c
340a0087
34e7040b
```

```
3c050065
340508a3
00000000
116a0008
358c0008
3c1600aa
016c5820
01405820
8d300004
01495020
01405822
116afff7
3c022b67
```

测试代码2

```
ori $t0, $0, 1
ori $s0, $0, 4
ori $s1, $0, 1
ori $s2, $0, 8
lui $s3, 123

l1:
    beq $t0, $s0, l1e
    add $t0, $t0, $t0
    j   l1

l1e:
    jal sum
    ori $t0, $0, 1
    j   l2

sum:
    add $a1, $0, $s0
    add $a2, $0, $s2
    sub $v0, $a1, $a2
    jr  $ra

l2:
    ori $v0, $0, 10
```

测试代码3

```
lui $s0, 0xf123
ori $s0, $s0, 0x3f21
lui $s1, 0x4567
ori $s1, $s1, 0x7465
lui $s2, 0x89ab
ori $s2, $s2, 0xb89a
lui $s3, 0xcdef
ori $s3, $s3, 0xcfed

ori $t0, $0, 4
```

```
sw $s0, 0($0)
sw $s1, 0($t0)
sw $s2, 4($t0)

add $t0, $t0, $t0
lw $t9, 0($t0)
lw $t8, -4($t0)

ori $t0, $0, 20
sw $s3, -8($t0)

ori $t0, $0, 19
sh $s1, -1($t0)
sh $s2, 1($t0)

ori $t0, $0, 25
sb $s3, -1($t0)
sb $s1, 0($t0)
sb $s2, 1($t0)
sb $s0, 2($t0)

ori $t0, $0, 2
lb $t1, -2($t0)
lb $t3, -1($t0)
lb $t1, 0($t0)
lb $t2, 1($t0)

ori $t0, $0, 19
lh $t2, -1($t0)
lh $t5, 1($t0)
```

思考题

1. 上面我们介绍了通过 FSM 理解单周期 CPU 的基本方法。请大家指出单周期 CPU 所用到的模块中，哪些发挥状态存储功能，哪些发挥状态转移功能。

状态存储：GRF、DM

状态转移：IFU、ALU、EXT、Controller

2. 现在我们的模块中 IM 使用 ROM， DM 使用 RAM， GRF 使用 Register，这种做法合理吗？请给出分析，若有改进意见也请一并给出。

我认为是合理的。

IM只需要被读取，而ROM是只读的，下次打开文件时内存依然存在，且运行过程中不会被篡改；

DM需要支持读、写功能，一个时钟周期内只会进行读、写的其中一种操作。RAM即可支持读写操作，又在占用空间上优于寄存器文件。

GRF需要支持读、写功能，且与ALU直接相连，对读、写速度要求较高，故使用寄存器文件。

3. 在上述提示的模块之外，你是否在实际实现时设计了其他的模块？如果是的话，请给出介绍和设计的思路。

并未设计新的模块。

4.事实上，实现 nop 空指令，我们并不需要将它加入控制信号真值表，为什么?

Controller采用与或门阵列实现，读入nop指令时所有的控制信号均保持在低电平，只进行了PC ← PC + 4，而不会产生其他任何操作。

5. 阅读 Pre 的“MIPS 指令集及汇编语言”一节中给出的测试样例，评价其强度（可从各个指令的覆盖情况，单一指令各种行为的覆盖情况等方面分析），并指出具体的不足之处。

我认为该样例覆盖了该CPU中支持的所有指令，且先由最基本的可独立判断正误的指令进行验证，之后再对更高层的指令的结果正误进行验证，能对CPU的设计起到较为准确的反馈。

可以考虑加入一些32位数、16位无符号数的边界情况，多增加一些目标寄存器为 $0 的指令，达到更好的测试效果。