计算机组成原理实验报告参考模板

一、CPU 设计方案综述

（一）总体设计概述

本 CPU 为 logisim 实现的单周期 MIPS-CPU，支持的指令集包括{addu，subu、beq、ori、lw、sw、lui、j、jal、jr}。为了实现这些功能，CPU 包含了 main、GRF 寄存器堆、ALU 算术逻辑、DM 数据存储器、EXT 位扩展、Control 控制器、PC 程序计数器以及 IM 指令存储器（IFU 取指令单元）。

（二）关键模块定义

1、IFU 取指令单元

（a）端口说明

名称	位数	Input/output	作用
PC	32	I	待更新 PC 值
Clr	1	I	复位端
Clk	1	I	时钟端
PCplus	32	O	PC+4
Instr	32	O	指令

（b）功能定义

序号	类型	作用
1	复位	当 clr 信号为 1 时，PC 值变成 0，回到指令的起始端
2	读取地址	当时钟上升沿到来时，读取 PC_的值
3	输出指令	将当前的 PC 值所对应指令取出

2、GRF 寄存器堆

（a）端口定义

名称	位数	输入/输出	作用
Rs	5	I	读取寄存器 rs
Rt	5	I	读取寄存器 rt
Rd	5	I	写入寄存器 rd
Data	32	I	写入数据
Clr	1	I	时钟端
Clk	1	I	复位端
Regw	1	I	写入使能
Data_in_rs	32	O	寄存器 rs 输出值
Data_in_rt	32	O	寄存器 rt 输出值

（b）功能定义

序号	功能	描述
1	复位	当 clr 信号为 1，GRF 所有寄存器清零。
2	写入	RegW=1，data 端和 rd 端生效，写入数据到 rd
3	读取	将 Rs 和 Rt 输出

3、ALU 算术逻辑

（a）端口说明

名称	位数	输入/输出	作用
Data_in_rs	32	I	读入 rs 的值
Data_in_rt	32	I	读入 rt 或者 imm 值
ALU[2:0]	3	I	计算操作
Beq_flag	1	O	Beq 判定端

ALUresult	32	O	输出计算结果

（b）功能定义

序号	功能名称	描述
1	逻辑运算	ALU=0，进行与运算 ALU=1，进行或运算 ALU=2，进行加运算 ALU=3，进行减运算 ALU=4，进行乘运算 ALU=5，进行除运算 ALU=6，进行左移运算 ALU=7，进行右移运算
2	相等判断	当进行减运算时，如果得到结果为 0，则 Zero=1，否则为 0

4、EXT 位扩展

（1）端口说明

名称	位数	输入/输出	作用
Im	16	I	指令后 16 位立即数
Zero_extern	1	I	指令零扩展到 32 位信号
Sign_extern	1	I	指令符号扩展到 32 位信号
Extern	32	O	扩展后指令

（2）功能定义

序号	功能	描述
1	符号扩展	当 Zeroop=0，signop=1，extern 符号扩展
2	无符号扩展	当 Zeroop=1，signop=0，extern 无符号扩展

5、DM 数据存储器

（1）端口说明

名称	位数	输入/输出	作用
ADD	5	I	作为读写的地址端
Wdata	32	I	写入的数据
MemW	1	I	写入控制信号
Clk	1	I	时钟端
MemR	1	I	读取控制信号
Clr	1	I	复位端
Rdata	32	O	读取的数据

（2）功能定义

序号	功能	描述
1	将 Wdata 写入 mem	当 MemW=1，根据 ADD 将 Wdata 写入对应的空间
2	读取 Rdata	当 MemR=1，根据 ADD 将 Rdata 取出

3	复位端	当 clr=1，将所有的内存数值清零

6、Control 控制器

（1）端口定义

名称	位数	输入/输出	作用
OP	6	I	功能定义
Fun	6	I	功能辅助定义
Regdst	1	O	选取 rt 还是 rs 进行写入
//J26	1	O	输出 imm26 扩展
ALUsrc	1	O	选取 rt 还是立即数
//Jal	1	O	Jal 信号
Memtoreg	1	O	是否从 Mem 取值到 grf
//jr	1	O	Jr 信号
RegW	1	O	是否写入寄存器
MemW	1	O	是否写入内存
MemR	1	O	是否读取内存
Beq	1	O	是否为 Beq 指令
Signop	1	O	是否为符号扩展
Zeroop	1	O	是否为零扩展
ALU0	1	O	操作数 0
ALU1	1	O	操作数 1

ALU2	1	0	操作数 2

（2）功能定义

详细见控制器设计说明。

二、控制器设计说明

端口名	描述
OP	Operation
Fun	Function
Regdst	位于 GRF 读入地址 rd 处： 0：rd/rt = rt 1：rd/rt = rd
ALUsrc	选择 rt 的值或者立即数的值： 0:data_rt = data_in_rt 1:data_rt = immediate
Memtoreg	将读取出的 Mem 值载入寄存器中 0：寄存器的值为 ALU 计算值 1：寄存器的值为 Mem 取出值
RegW	寄存器写入使能： 1：将 Wdata 里面的值写入 rd/rt 0：此时无法写入，屏蔽 rd/rt 和 Wdata
MemW	内存写入使能： 1：将 data_in_rt 值写入对应内存地址 ADD 0：此时忽略 data_in_rt

MemR	内存读入使能，与 Mem 对应： 1：将 ADD 处值输出到 Wdata2 0：忽略 Wdata2
Beq	确定 Beq 信号: Zero&Beq = 1：PCextern->PC Zero&Beq = 0：PCplus->PC
Signop zeroop	零扩展和符号扩展
ALU	计算指令，详细见上

三、指令真值表

指令	addu	subu	ori	lw	sw	beq	lui	j	Jal	Jr
Op	000000	000000	001101	100011	101011	000100	001111	000010	000011	000000
Fun	100001	100011	NA	NA	NA	NA	NA	NA	NA	001000
Regdst	1	1	0	0	X	X	0	X	X	X
ALUsrc	0	0	1	1	1	0	1	X	X	X
Memtoreg	0	0	0	1	X	x	0	X	0	0

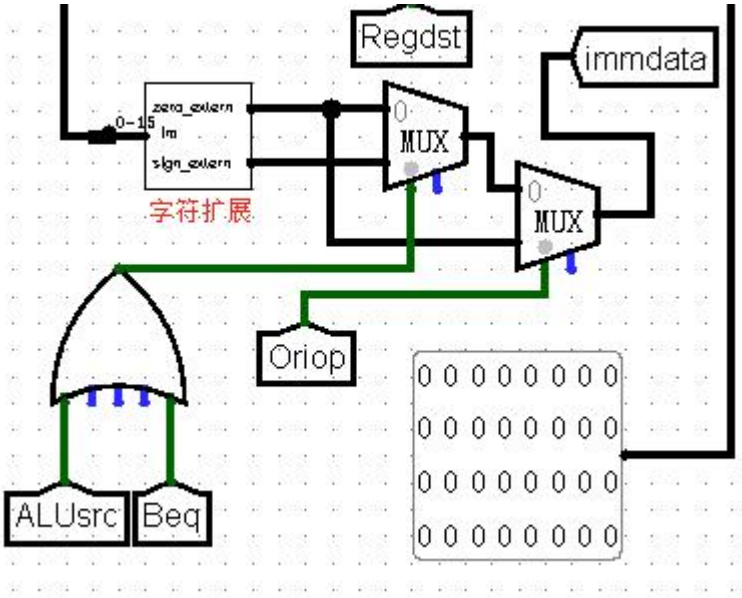
RegW	1	1	1	1	0	0	1	X	1	0
MemW	0	0	0	0	1	0	0	0	0	0
MemR	0	0	0	1	0	0	0	0	0	0
Beq	0	0	0	0	0	1	0	0	0	0
Oriop	0	0	1	0	0	0	0	0	0	0
Signop	0	0	0	1	1	1	0	1	1	0
ALU	010	011	001	010	010	011	110	X	X	X

（三）重要机制实现办法

1、Beq 实现方案

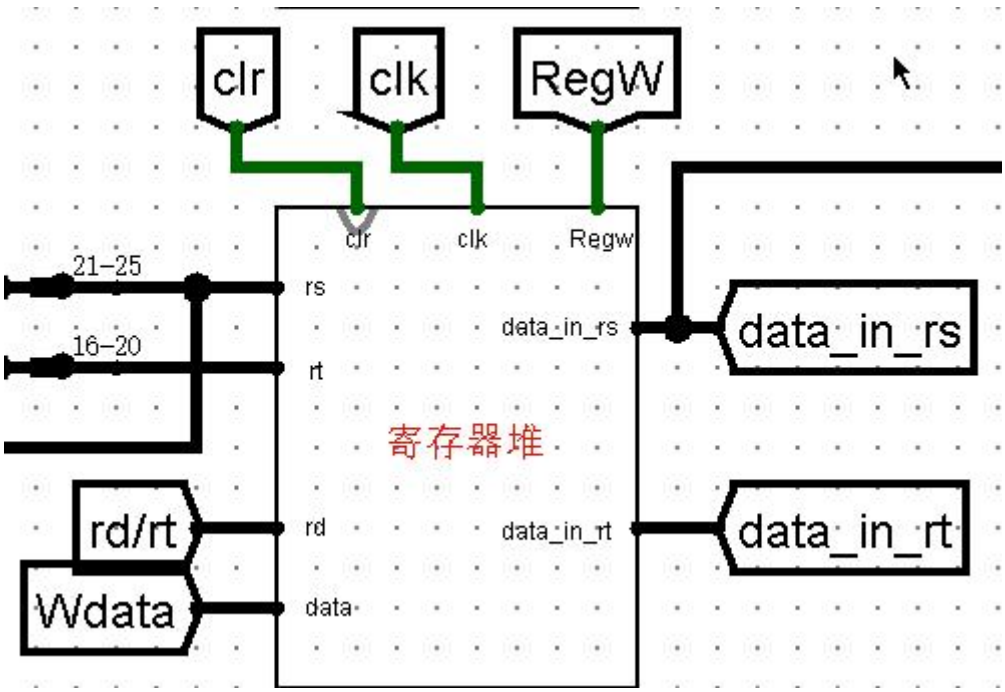
步骤一：字符扩展和寄存器堆

1）字符扩展：



此处选择 0~15 位进行符号扩展。ALUsrc 和 Oriop 此时为 0，而 Beq 此时为 1，因此输出为 sign_extern，故 immdata 里面存储了 32 位符号扩展立即数。

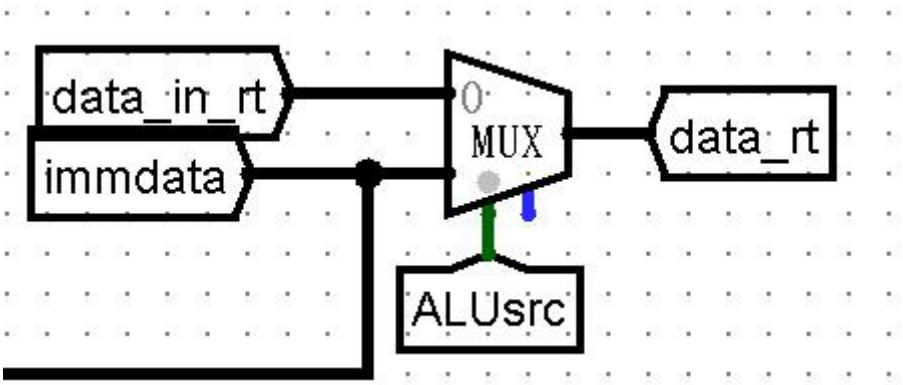
2）寄存器操作



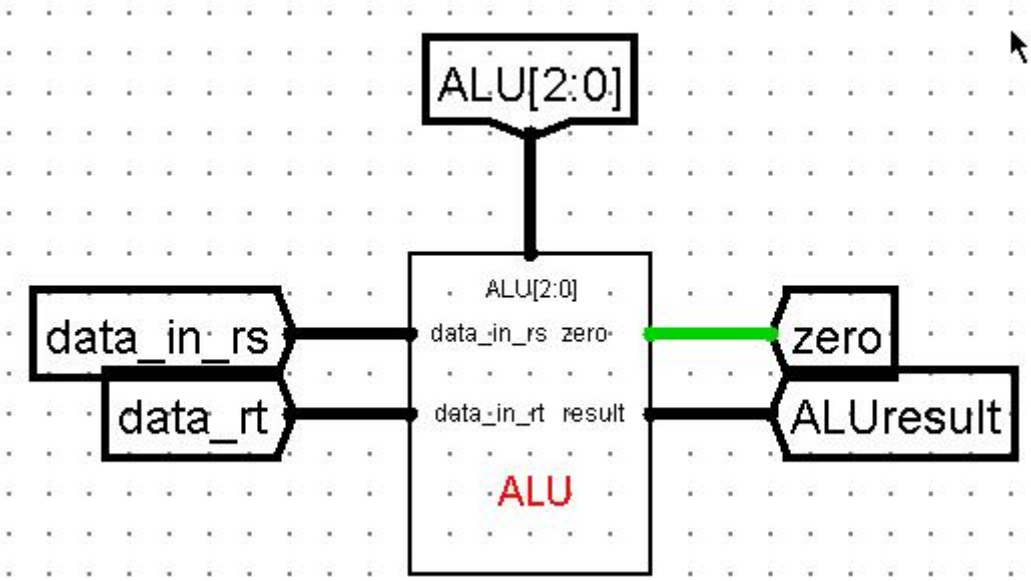
读入 rs 和 rt 的值，分别放在 data_in_rs，和 data_in_rt 中。此处 RegW 忽略。

步骤二：ALU 运算和 PC_extern 运算

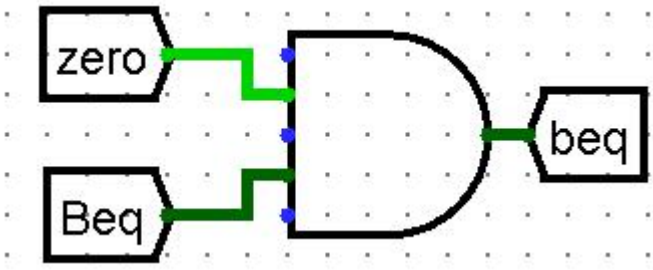
1）ALU 运算



首先由于 ALUsrc=0，从而选择 rt 里面的数据作为 ALU 的第二个运算数，即 data_rt。

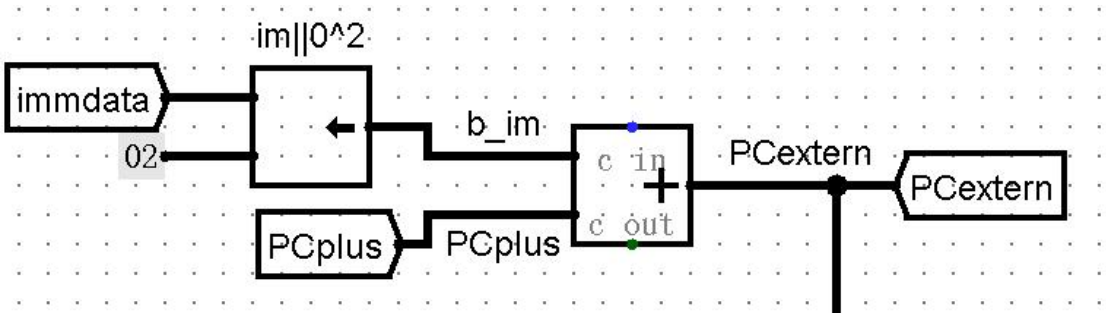


继续判断 rt 和 rs 是否相等，此时 ALU[2:0]为 011，即减法运算，如果 ALUresult=0，则 zero=0 输出。



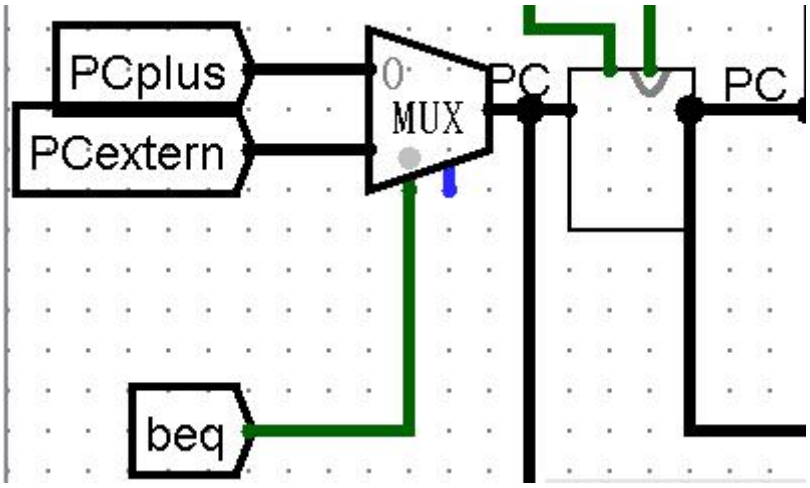
然后对 zero 和 Beq 进行与运算，只有保证 rt=rs 而且此时指令为 Beq 时我们才能选取 PC_extern 的值。

2）PC_extern 的运算



根据得到的符号扩展数 immdata,左移两位而后加上 PCplus,得到 PCextern。

步骤三：判断 PC 选取值



通过对 beq 的值选取 PC 值，原理如上所示。

至此，BEQ 指令完成。

二、测试方案

本测试方案依据不同的指令和对应输出进行测试。

1、addu、subu、ori 指令测试。

```
1.  ori             $t0,1
2.  ori             $t1,100
3.  ori             $t2,50
4.  ori             $t1,10
5.  addu             $t1,$t1,$t1
6.  subu             $t2,$t2,$t1
```

```
addu            $t0,$t1,$t2
subu            $t3,$t2,$t0
```

期望输出：

```
$t0 = 0x32
$t1 = 0xdc
$t2 = 0xffffff56
$t3 = 0xffffff24
```

2、lw、sw 指令测试

```
ori      $t0,20
ori      $t1,4
ori      $t2,8
sw    $t2,0($t1)
sw    $t1,8($0)
sw    $t0,-8($t0)
lw    $t2,8($t1)
lw    $t1,0($t1)
lw    $t0,-12($t0)
```

期望输出：

```
$t0 = 4
$t1 = 8
$t2 = 20
0x0 = 0
0x4 = 8
0x8 = 4
0xc = 20
```

3、beq 测试

```
ori      $t0,200
ori      $t1,1000
ori      $t2,200
```

```
ori       $t3,100
judge:
beq       $t0,$t1,end
nop
subu  $t1,$t1,$t3
beq       $t0,$t1,judge
nop
end:
ori       $t0,1
```

期待输出：

```
$t0 = 0xc9
$t1 = 0x384
$t2 = 0xc8
$t3 = 0x64
```

4、lui 测试

```
ori          $t0,10
lui          $t0,100
lui          $t0,2
```

期待输出：

```
$t0 = 0x20000
```

三、思考题

1、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。

答：

虽然进行了简化处理，但这种处理是合理的。

1）IM 使用 ROM，符合指令的只读而不写，同时避免指令被修改的风险。

2）DM 使用 RAM，符合 memory 的既能读又能写的特性，同时对于不同信号的把控从而控制读写，也实现了 DM 功能的正确性。

3）GRF 使用 Register，操作简单。

2、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？

答：

由于 nop 指令全为 0，所有信号都是 0，不进行任何操作。

3、上文提到，MARS 不能导出 PC 与 DM 起始地址均为 0 的机器码。实际上，可以通过为 DM 增添片选信号，来避免手工修改的麻烦，请查阅相关资料进行了解，并阐释为了解决这个问题，你最终采用的方法。

答：

由于 DM 只需要 2~6 位的地址码，从而高位上的编码可以作为片选信号，我们可以设定高位编码的某一特定值，在这一特定值下选择特定的 DM 端口，从而避免了手工修改机器码。

4、除了编写程序进行测试外，还有一种验证 CPU 设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)”了解相关内容后，简要阐述相比于测试，形式验证的优劣之处。

答：

优点：

1、形式验证正如题中描述，借用数学方法将电路和功能描述进行比较，从而测试者可以忽略测试向量。

2、形式验证的验证容量较大，可以对所有的情况进行验证，从而更加准确，相比于代码测试弥补了其只能纠正部分问题的不足。

缺点：

1、形式验证如果用于更加复杂的电路，可能会花上很长时间进行验证。

2、对于电路的一些特殊功能无法进行有效验证，存在自身的局限性。