汇编与接口课程设计实验报告

实验名称: MIPS 指令系统与汇编程序设计

班　　级: 07111602

学　　号: 1120163589

姓　　名: 唐 容 川

指导老师: 王 娟

实验时间: 2019.08.26-2019.09.14

目录

1 实验目的

1. 学习计算机硬件设计的理论
2. 学习软硬件工作的原理
3. 学习计算机硬件接口的控制
4. 学习 RISC 指令集的 CPU 设计流程
5. 学习 Vivado 等设计软件的使用
6. 培养计算机系统能力

2 实验内容

完成计算机 CPU 的模块设计，要求该设计可以支持 MIPS 指令集的一部分，包括 j,Beq,Sw,Lw,Add,Addiu,Lui 以及以及抽取的 BNE 指令的设计，并进行 Vivado 仿真实验，给出实验结果。

3 实验环境

3.1 硬件

HuaWei matebook 14

3.2 操作系统

Windows 10

3.3 使用工具

Vivado

4 实验原理

4.1 指令描述

4.1.1 算术运算指令

格式：Add rd,rs,rt

真值：000000 rs(5 位) rt(5 位) rd(5 位) reserved

功能：rd←rs + rt。reserved 为预留部分，即未用，一般填“0”。

（1） 格式：addi rt , rs ,immediate

真值：000001 rs(5 位) rt(5 位) immediate(16 位)

功能：rt←rs + (sign-extend)immediate；immediate 符号扩展再参加“加”运算。

（2） 格式：sub rd , rs , rt

真值：000010 rs(5 位) rt(5 位) rd(5 位) reserved

功能：rd←rs - rt

4.1.2 逻辑运算指令

（1） 格式：ori rt , rs ,immediate

真值：010000 rs(5 位) rt(5 位) immediate(16 位)

功能：rt←rs | (zero-extend)immediate；immediate 做“0”扩展再参加“或”运算。

（2） 格式：and rd , rs , rt

真值：010001 rs(5 位) rt(5 位) rd(5 位) reserved

功能：rd←rs & rt；逻辑与运算。

（3） 格式：or rd , rs , rt

真值：010010 rs(5 位) rt(5 位) rd(5 位) reserved

功能：rd←rs | rt；逻辑或运算。

4.1.3 移位指令

（1） 格式：sll rd, rt,sa

真值：011000 未用 rt(5 位) rd(5 位) sa reserved

功能：rd<－rt<<(zero-extend)sa，左移 sa 位 ，(zero-extend)sa

4.1.4 比较指令

（2） 格式：slt rd, rs, rt 带符号数

真值：011100 rs(5 位) rt(5 位) rd(5 位) reserved

功能：if (rs<rt) rd =1 else rd=0, 具体请看表 2 ALU 运算功能表，带符号

4.1.5 存储器读/写指令

（1） 格式：sw rt ,immediate(rs) 写存储器

真值：100110 rs(5 位) rt(5 位) immediate(16 位)

功能：memory[rs+ (sign-extend)immediate]←rt；immediate 符号扩展再相加。即将 rt 寄存器的内容保存到 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元 中。

（2） 格式：lw rt , immediate(rs) 读存储器

真值：100111 rs(5 位) rt(5 位) immediate(16 位)

功能：rt ← memory[rs + (sign-extend)immediate]；immediate 符号扩展再相加。即读取 rs 寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中的数，然后 保存到 rt 寄存器中。

4.1.6 分支指令

（1） 格式：beq rs,rt,immediate

真值：110000 rs(5 位) rt(5 位) immediate(16 位)

功能：if(rs=rt) pc←pc + 4 + (sign-extend)immediate <<2 else pc ←pc + 4 特别说明：immediate 是从 PC+4 地址开始和转移到的指令之间指令条数。immediate 符号扩展之后左移 2 位再相加。为什么要左移 2 位？由于跳转到的指令地址肯定是 4 的倍数 （每条指令占 4 个字节），最低两位是“00”，因此将 immediate 放进指令码中的时候，是 右移了 2 位的，也就是以上说的“指令之间指令条数”。

（2） 格式：bne rs,rt,immediate

真值：110001 rs(5 位) rt(5 位) immediate

功能：if(rs!=rt) pc←pc + 4 + (sign-extend)immediate <<2 else pc ←pc + 4 特别说明：与 beq 不同点是，不等时转移，相等时顺序执行。

（3） 格式：bgtz rs,immediate

真值：110010 rs(5 位) 00000 immediate

功能：if(rs>0) pc←pc + 4 + (sign-extend)immediate <<2 else pc ←pc + 4

4.1.7 跳转指令

（1） 格式：j addr

真值：111000 addr[27..2]

功能：pc <－{(pc+4)[31..28],addr[27..2],0,0}，无条件跳转。

4.1.8 停机指令

（1） 格式：halt

真值：111111 00000000000000000000000000(26 位)

功能：停机；不改变 PC 的值，PC 保持不变。

4.2 CPU 执行原理

具有一个周期的周期意味着一个指令在一个时钟周期内完成，并且下一个指令开始执行，即一个指令在一个时钟周期内完成。电平从低变为高的时刻称为时钟的上升沿，两个相邻小时的上升沿之间的时间间隔称为一个时钟周期。时钟周期也称为振荡周期。处理 CPU 命令时，通常必须执行以下步骤：

（1）检索指令（IF）：根据程序计数器的 PC 命令的地址从 PC 存储器中检索指令，同时检索 PC 命令的字长。自动递增以生成下一条指令所需的指令地址。但是，如果检测到“地址传送”命令，则控制器将“传送地址”发送到 PC。当然，收到的“地址”必须在发送前更改 PC。

（2）解码指令（ID）：分析和解码通过指令获取操作获得的指令，以确定该指令应该执行的操作以及用于控制执行状态中的各种类型的相应操作。生成控制信号。手术室。

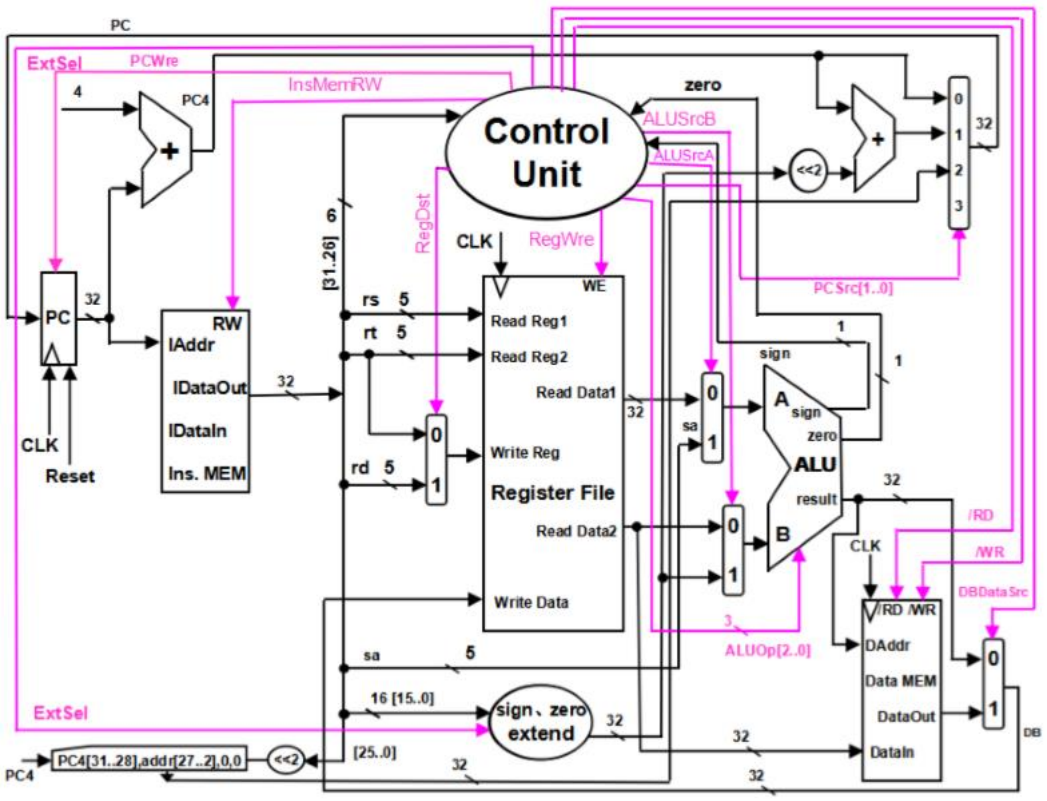
（3）命令执行（EXE）：通过解码命令获得的操作控制信号被特别执行并进入结果的写回状态。

（4）存储器访问（MEM）。在该步骤，执行需要存储器访问的所有操作，指示存储器中的数据的地址以及将数据写入存储器中的指定存储器位置或从存储器中的指定存储器位置写入数据。你。记忆。数据以数据地址为单位收集。

（5）记录结果记录（WB）：在访问存储器中接收的命令或数据的结果被写回适当的目的地寄存器。

5 模块设计

5.1 CPU 整体设计图



5.2 部件及引脚说明

相关部件和引脚说明：

指令存储器：指令存储器，

Iaddr，指令存储器地址输入端口

IDataIn，指令存储器数据输入端口（指令代码输入端口）

IDataOut，指令存储器数据输出端口（指令代码输出端口）

指令存储器读和写控制信号写为 RW，0,1 读

数据存储器：数据存储器

Daddr，数据存储器地址输入端口

DataIn，数据存储器数据输入端口

DataOut，数据存储器数据输出端口/ RD，数据存储器读控制信号，0 读/ WR，数据存储器写控制信号，写 0

注册文件：注册组

读取 Reg1，rs 寄存器地址输入端口

Reg2 读取，rt 寄存器地址输入端口

写入寄存器，用于写入数据的寄存器端口，地址源的 rt 或 rd 字段

写入数据，写入寄存器数据输入端口

Data1 读取，rs 寄存器数据输出端口

读取 Data2，rt 寄存器数据输出端口

写入使能信号，WE 触发 1 时的时钟沿

ALU：算术逻辑单元结果，ALU 运算结果为零，运算结果标志，结果为 0，零= 1，否则为零= 0，运算结果标志，最高有效位为 0，符号= 0，正数; ，符号= 1，否定

5.3 关键模块设计思路

5.3.1 PC 模块

PC 模块主要完成下一个指令地址的计算。此 CPU 设计中的下一条指令的地址有三种计算方法。对于非跳转指令，address = address 4，对于跳转指令，有一个不同的指令 pc <。 -pc 4（符号扩展）立即和 pc < - {（pc 4）[31:28]，addr [27：2]，0,0}两种计算新地址的方法，以及中断指令时不要改变电脑地址。因此，PCWre 信号用于控制是否更改 pc 地址，PCSrc 信号用于控制 PC 地址的更改方式（3 种类型）。

5.3.2 RAM 模块

RAM 模块:输入:读/写地址地址的地址，写入的 32 位寄存器值。writeData，读写信号 nRD，nWR，读取结果 Dataout

5.3.3 寄存器组 RegFile 模块

RegWre 用作读/写输入控制终端，因为 RegFile 模块用作寄存器组，因此必须实现寄存器组的基本读/写功能。来自控制中心的信号和寄存器 rs 和 rt 分别

作为输入 InputReg1 和 ReadReg2，输出值来自指令 rs，rt 片段，输出 ReadData1 和

ReadData2，因为寄存器需要同时写入，要写入的寄存器号用作输入 WriteReg，从 rt 输入的值用作输入 WriteData，ALU 或 RAM 读取值的计算结果是数据选择器由输入信号控制。

5.3.4 指令存储器模块

指令存储器模块

输入：PC 地址 pc，读/写命令控制信号 InsMemRW

输出：指令操作码 op，指令寄存器号 rs，rt，rt，rd，sa，立即立即值，跳转指令地址 addr

5.3.5 字拓展模块

单词扩展模块分为两部分，ExtendUnit 通过零扩展或符号位扩展将 16 位立即值扩展为 32 位立即值，saExtendUnit 扩展指令 sll 5 位移位数（立即）sa 到 32 的立即值有点。

扩展模块 EntendUnit：

输入：16 位立即值，字扩展/符号位扩展控制信号 ExtSel

输出：扩展结果 32 位立即 extensionResult

5.3.6 信号控制模块

控制模块输入指令代码，每个信号输出参考附件“控制模块输出信号表.xls”设计。有两种设计方法，一种是直接使用 case 语句在每条指令上写入相应的输出信号。一种设计方法是在设计中使用组合逻辑。在这里，选择第二种方法。

5.3.7 ALU 模块

ALU 模块的一个输入是指令操作码 ALUop，其确定 ALU 执行的操作。上面给出了 ALU 计算控制表。另一个输入是两个操作数，rega 和 regb。有多个源，数据选择器会过滤正确的信号。输出包含零标志零。如果运算结果为 0，则输出为 1，符号位符号，如果运算结果符号，则结果为 1. ALU 的结果是输出信号的结果。

5.3.8 数据选择器模块

使用总共多个数据选择器，每个数据选择器由控制信号DBDataSrc等控制，以选择要输入到相应模块的信号，并且可以看到以下单周期CPU设计图。

6 实验结果

6.1 测试内容

以下开始一个接一个地检查和解释。该部分连续分析和检查测试设置指令的正确性。

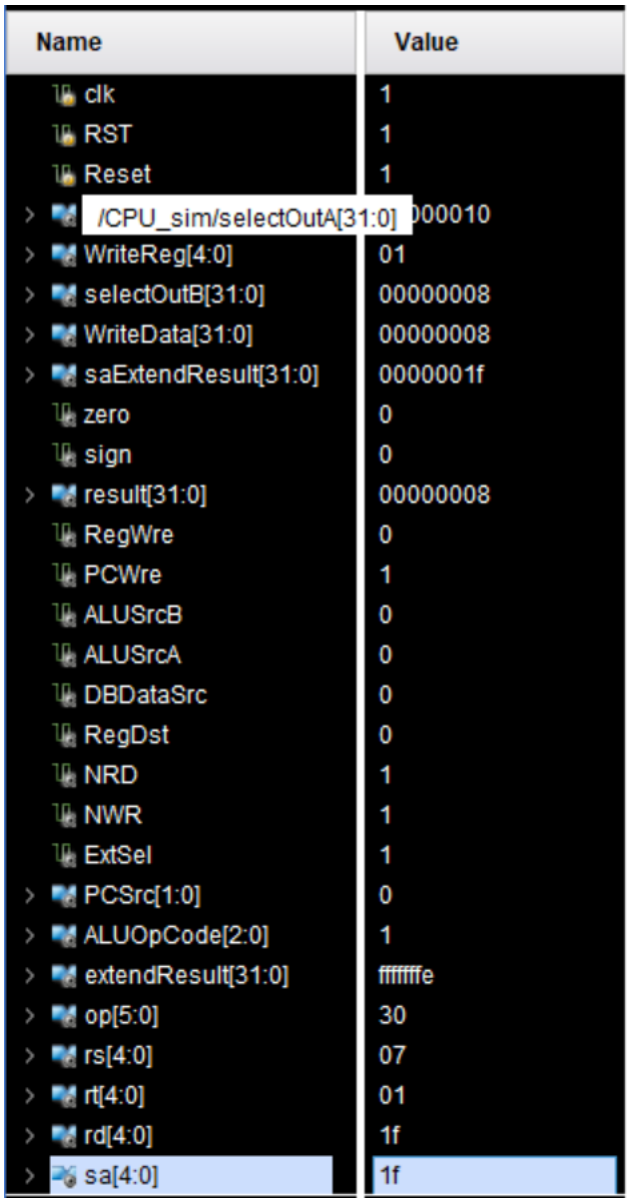
检查所有指令将分为两步，第一步验证结果，特别是操作指令的结果是否正确，第二步检查主中间信号，特别是检查 Command 的执行是否完全正确，做出劣质结果检查，以确保CPU的正常运行。

测试的具体内容如下：

```
addi  $1,$0,8
ori  $2,$0,2
add  $3,$2,$1
sub  $5,$3,$2
and  $4,$5,$2
or  $8,$4,$2
bne $8,$1,-2
beq $7,$1,-2  (≠,转 28)
sw  $2,4($1)
lw  $9,4($1)
j  0x00000038
```

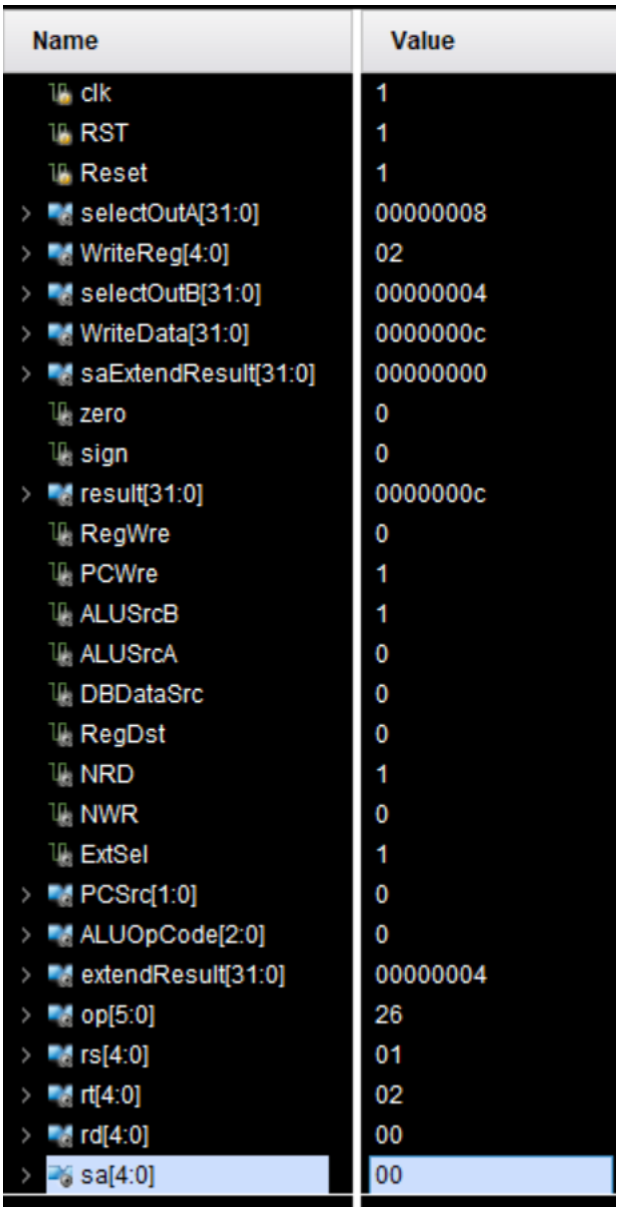
6.2 仿真结果

6.2.1 addi $1,$0,8



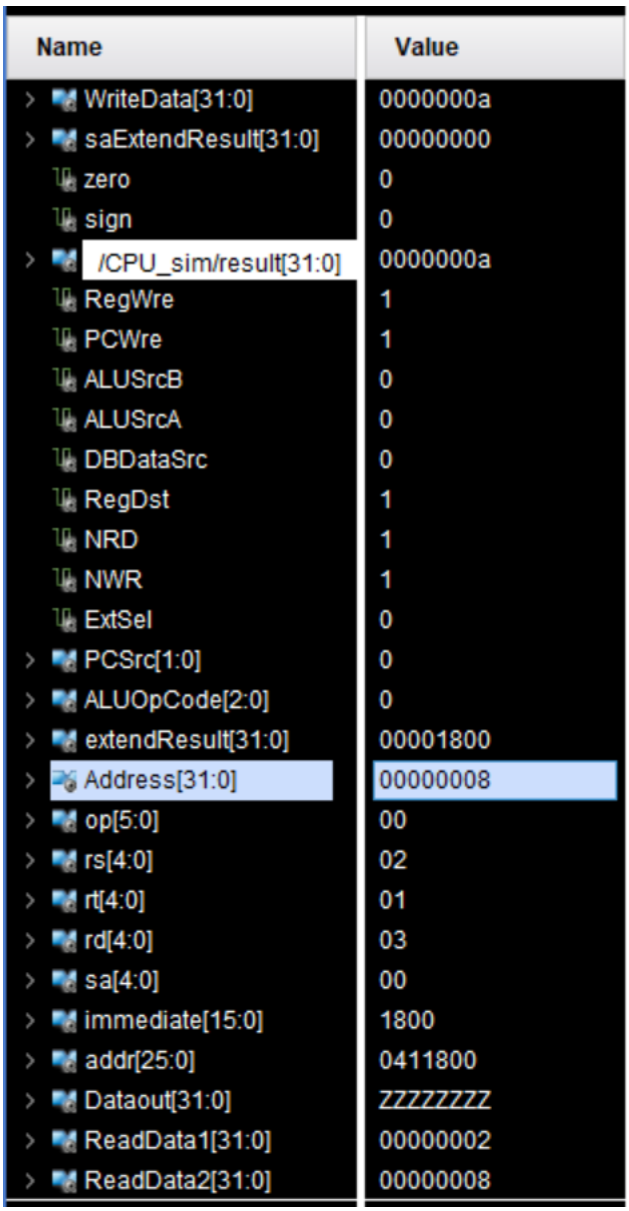
在仿真图中，可以看到 PC 地址，并且相应的指令代码与上述两个表匹配，这是正确的。输出 ReadData1 寄存器设置正确的输出寄存器 RS($ 0)值为 0,是 ALU 计算结果 8,写入寄存器号信号 WriteReg 的权利写入结果为 1（即$ 1）写入到 value .WriteData 信号是操作 8 的结果，正确。

6.2.2 ori $2,$0,2



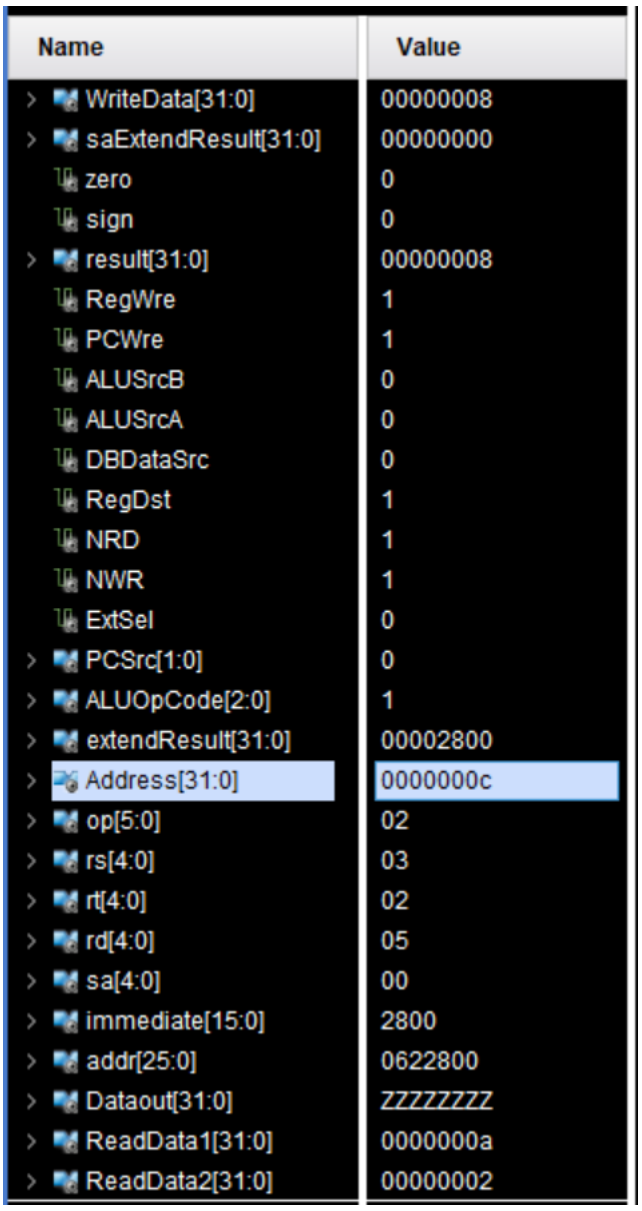
在该图中，可以看出寄存器的操作码 op，寄存器代码 rs，rt 等对应于上表并且是正确的。输出 ReadData1 寄存器设置是正确的。输出寄存器 RS（$ 0）值 0，ALU 结果计算等于 2，写入结果寄存器编号信号 WriteReg 的权限为 2（对应$ 2），写入值信号 WriteData 是操作 2 的结果，正确。

6.2.3 add $3,$2,$1



从图中可以看出，指令的地址，指令操作码 op，寄存器号 rs，rt 等都是正确的。寄存器 RS 值输出信号 ReadData1 和 ReadData2 成功输出正确的寄存器 RS（$ 2）和寄存器 RT（$ 1）值 2 和 8. ALU 操作的结果是 10，这是正确的。要写入的寄存器号 WriteReg 是 3（即 $ 3），这是正确的。必须写入寄存器$ 3 的结果值 WriteData 是 10，正确。

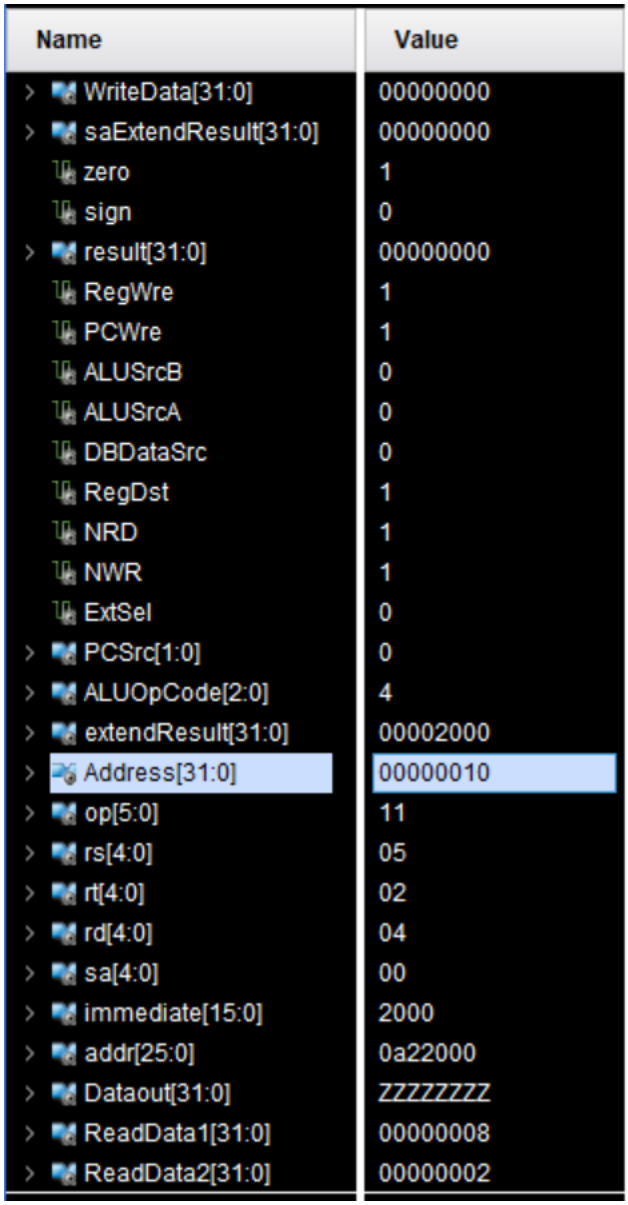
6.2.4 sub $5,$3,$2



Name	Value
WriteData[31:0]	00000008
saExtendResult[31:0]	00000000
zero	0
sign	0
result[31:0]	00000008
RegWre	1
PCWre	1
ALUSrcB	0
ALUSrcA	0
DBDataSrc	0
RegDst	1
NRD	1
NWR	1
ExtSel	0
PCSrc[1:0]	0
ALUOpCode[2:0]	1
extendResult[31:0]	00002800
Address[31:0]	0000000c
op[5:0]	02
rs[4:0]	03
rt[4:0]	02
rd[4:0]	05
sa[4:0]	00
immediate[15:0]	2800
addr[25:0]	0622800
Dataout[31:0]	ZZZZZZZZ
ReadData1[31:0]	0000000a
ReadData2[31:0]	00000002

从图中可以看出，指令的地址，指令操作码 op，寄存器号 rs，rt 等都是正确的。寄存器 rs 值输出信号 ReadData1 和 ReadData2 成功输出寄存器 rs（$ 3）和 rt（$ 2）10 和 8 的值是正确的。 ALU 操作的结果是 8，这是正确的。要写入的寄存器号 WriteReg 是 5（即 $ 5），这是正确的。必须写入寄存器$ 3 的结果值 WriteData 为 8，正确。

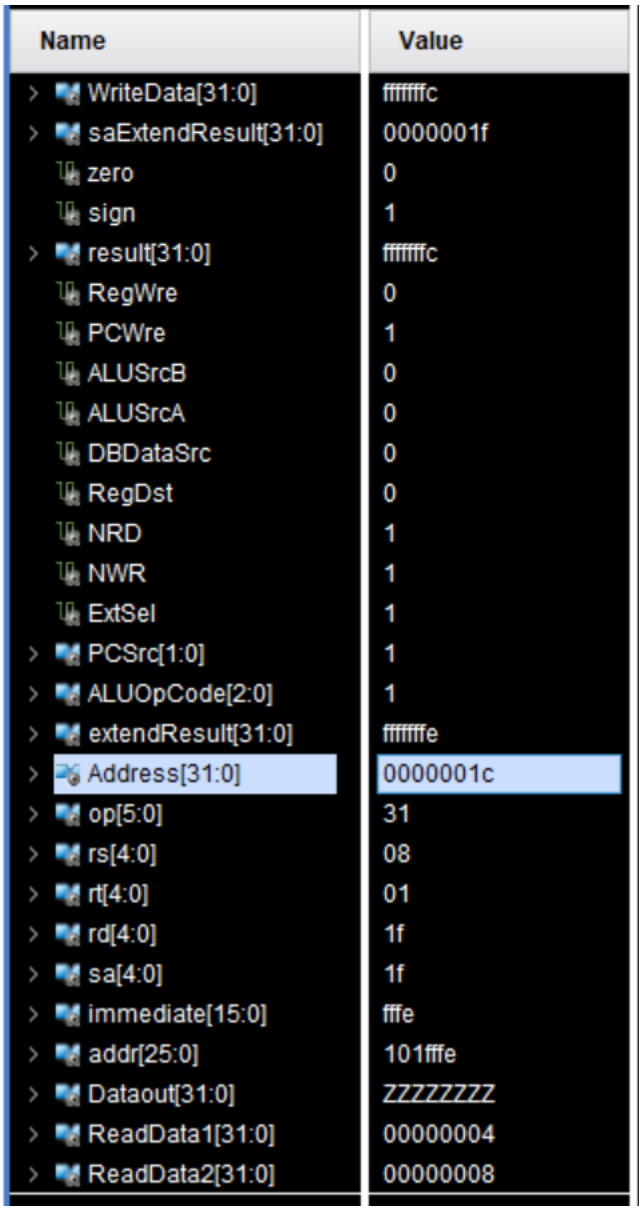
6.2.5 and $4,$5,$2



Name	Value
WriteData[31:0]	00000000
saExtendResult[31:0]	00000000
zero	1
sign	0
result[31:0]	00000000
RegWre	1
PCWre	1
ALUSrcB	0
ALUSrcA	0
DBDataSrc	0
RegDst	1
NRD	1
NWR	1
ExtSel	0
PCSrc[1:0]	0
ALUOpCode[2:0]	4
extendResult[31:0]	00002000
Address[31:0]	00000010
op[5:0]	11
rs[4:0]	05
rt[4:0]	02
rd[4:0]	04
sa[4:0]	00
immediate[15:0]	2000
addr[25:0]	0a22000
Dataout[31:0]	ZZZZZZZZ
ReadData1[31:0]	00000008
ReadData2[31:0]	00000002

地址命令的地址，指令操作码 op，寄存器号 rs，rt 等都是正确的。寄存器 rs 值输出信号 ReadData1 和 ReadData2 使用正确的值 8 和 2 成功输出寄存器 rs（$ 5）和寄存器 rt（$ 2）.ALU 操作的结果为 0，这是正确的。要写入的寄存器号 WriteReg 是 4（即$ 4），这是正确的。必须写入寄存器$ 3 的结果值 WriteData 为 0，正确。

6.2.6 bne $8,$1,-2



从图中可以看出，指令的地址，操作码，寄存器代码等是正确的。寄存器 rs 的输出信号 ReadData1 和 ReadData2，rt 值分别为 4 和 8，并且寄存器 rs（$ 8），rt（$ 1）的值被正确读取。 ALU 正确计算两个寄存器的值之差，由于差值为负，则零标志信号等于 0，此命令将跳转到指令地址 0x00000018，当控制信号 PC 地址如下：

此时，PC 控制信号为 PCSrc 01，并生成新的 PC 地址，如下所

示。

结果检查：通过执行指令的下一条指令，观察指令的下一条指令的地址，判断跳转是否成功。下面的模拟中可以看到该指令的下一条指令。执行结果是正确的。

6.2.7 beq $7,$1,-2 (≠,转 28)

Name	Value
WriteData[31:0]	00000000
saExtendResult[31:0]	0000001f
zero	1
sign	0
result[31:0]	00000000
RegWre	0
PCWre	1
ALUSrcB	0
ALUSrcA	0
DBDataSrc	0
RegDst	0
NRD	1
NWR	1
ExtSel	1
PCSrc[1:0]	1
ALUOpCode[2:0]	1
extendResult[31:0]	fffffffe
Address[31:0]	0000002c
op[5:0]	30
rs[4:0]	07
rt[4:0]	01
rd[4:0]	1f
sa[4:0]	1f
immediate[15:0]	fffe
addr[25:0]	0e1fffe
Dataout[31:0]	ZZZZZZZZ
ReadData1[31:0]	00000008
ReadData2[31:0]	00000008

如果从指令地址可见并且指令代码正确，则 ALU 操作的结果将为 0，表示两个寄存器相等，则信号空标志设置为 1，并且寄存器 RS

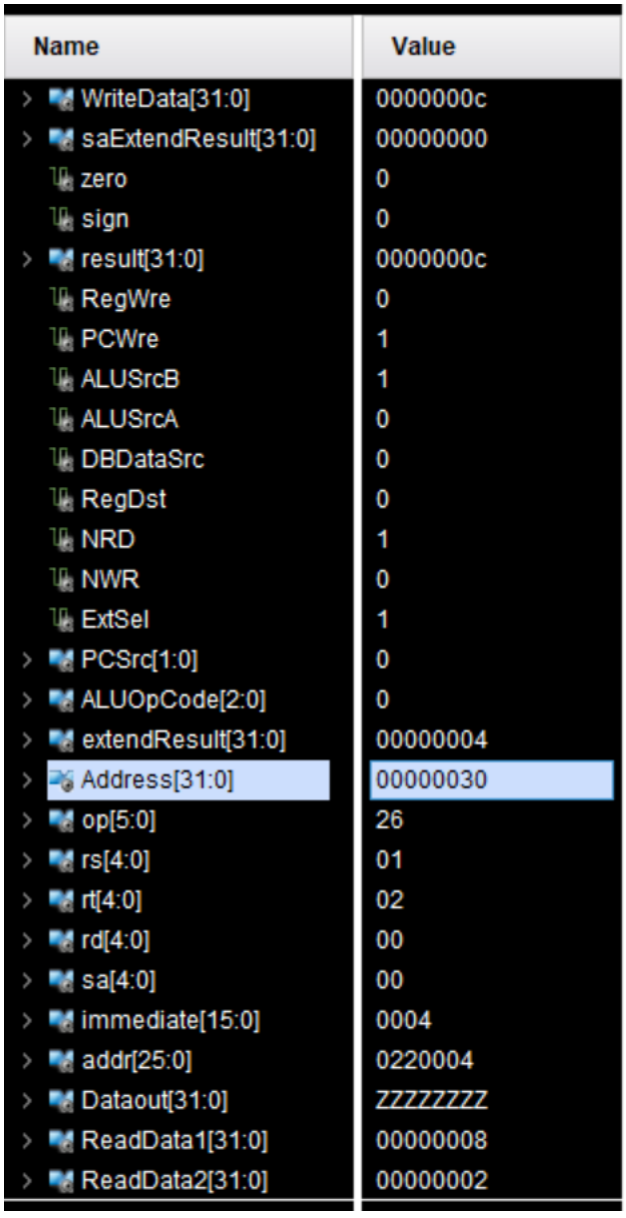
（图$ 7），rt 输出信号 ReadData1 和 ReadData2 的值（$ 1）都是 8，命令跳转到地址 0x00000028。

结果测试：执行命令后直接检查下一个命令，可以检查命令是否正确。

在模拟中，语句的下一个陈述是：

模拟如下所示：

6.2.8 sw $2,4($1)

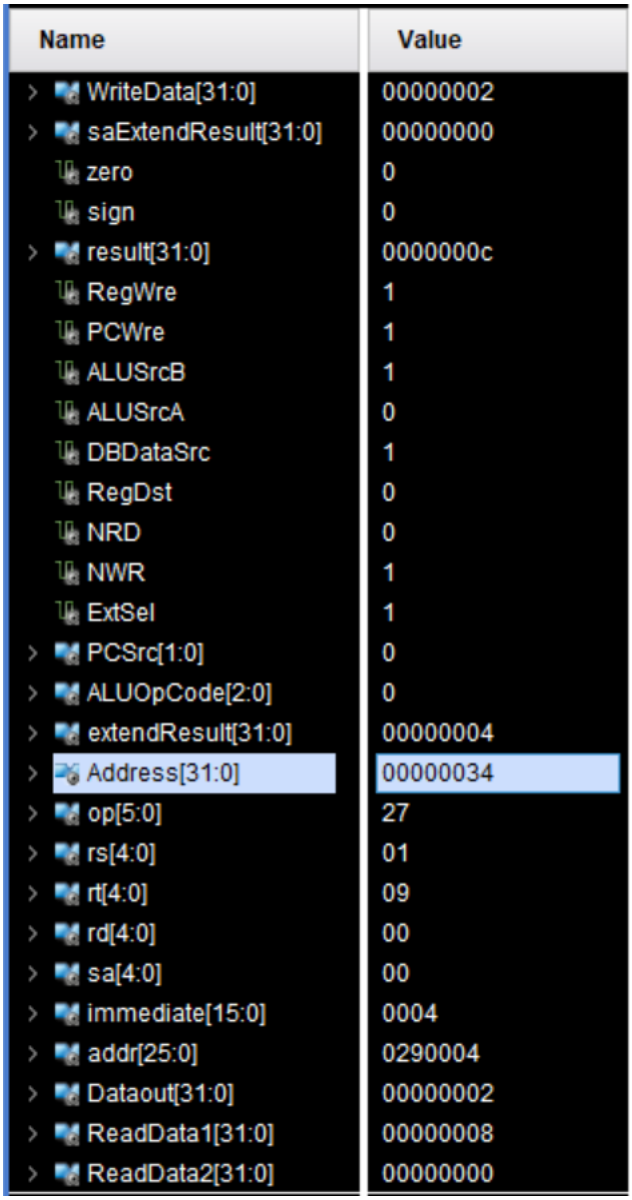


从图中可见 rt($2)寄存器的值的输出信号 ReadData2 为 2，即能够正确读取寄存器 rt 的值，该值将会被写入 RAM 中，这时写入 RAM 的地址由 ALU 计算得出为 result=12，因此地址正确。

结果检验：直接执行下一条指令： 0x00000034 lw $9,4($1) 来检验该条指令的正确性，这时将从地址 4($1)读取数据，若读取的值为 2，则该条指令能够正常执行。

仿真图如下：

6.2.9 lw $9,4($1)



如果从指令地址可见并且指令代码正确，则 ALU 操作的结果将为 0，表示两个寄存器相等，则信号空标志设置为 1，并且寄存器 RS（图$ 7），rt 输出信号 ReadData1 和 ReadData2 的值（$ 1）都是 8，

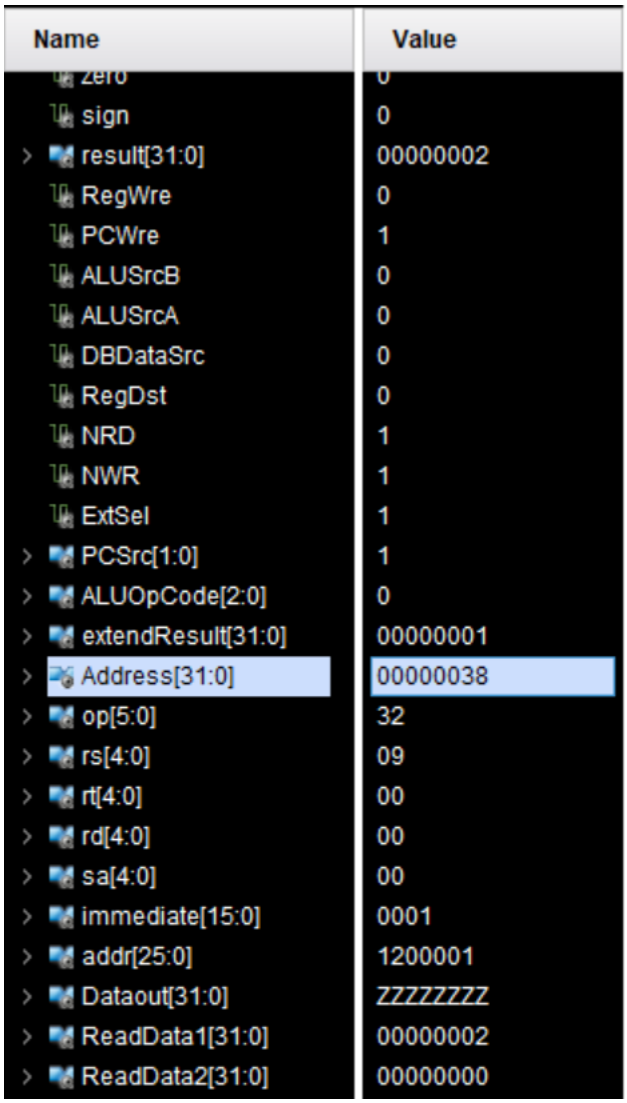
命令跳转到地址 0x00000028。

结果测试：执行命令后直接检查下一个命令，可以检查命令是否正确。

在模拟中，语句的下一个陈述是：

模拟如下所示：

6.2.10 j 0x00000038



结果检验：直接通过检查下面一条指令的地址来判断该指令是否正确执行，下一条指令如下图：

0x00000038 bgtz $9,1 （不跳转）

7 实验心得

7.1 Verilog 语言：

在之前的数字实验中，Verilog 语言没有得到正式研究，这导致了 CPU 早期阶段的许多语法问题。这充分揭示了基础知识不足的问题，并为系统学习提供了一些信息，这些信息将作为后续学习的基础。

7.2 模块化设计：

在初始阶段编写 CPU 代码时，通常会组合和写入许多模块，从而产生许多错误。之后，模块化的想法将逐步采用。例如，一些数据选择器与模块分开，这可能是有效的。模块可重复使用，减少代码冗余。另一方面，降低了代码的复杂性，有效地减少了语法错误并节省了时间。

7.3 从上而下的思想：

在许多模块的设计，有必要的前期规划，包括输入和各个模块的输出等，实现这个想法，事前没有直接写入总体规划每个模块，它是在最上面的文件很容易最终发生冲突。

7.4 代码书写规范：

本实验总结的良好写入规范包括：

重要的输入和输出信号要进行释。

避免寄存器和信号命名类似导致错误。

学会了复杂布线和其他内容是优先于编程的，需要其他的设计工具进行辅助。