# 实验报告

13061076赵乐

一、模块定义

### 1、PC

#### 1.1基本描述

PC主要功能是完成输出当前指令地址并保存下一条指令地址。复位后，PC指向0x0000\_3000，此处为第一条指令的地址。

#### 1.2模块接口

表1

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位  2：无效 |
| npc | I | 下条指令的地址 |
| pc | O | 32位指令存储器地址 |

#### 1.3功能定义

表2

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时， PC 被设置为 0x0000\_3000。 |
| 2 | 保存 npc 并输出 | 在每个时钟的上升沿保存 npc，并输出。 |

### 2、npc

#### 2.1基本描述

npc主要功能是根据当前指令地址与其他跳转信号判断下一条指令地址，同时输出pc+4。

#### 2.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| zero | I | ALU计算结果为0的标志。  1：计算结果为0  0：计算结果非0 |
| ifbeq | I | 是否为beq信号 |
| j | I | 是否为j |
| jal | I | 是否为j al |
| jr | I | 是否为jr指令的信号 |
| pc | I | 当前指令的地址 |
| imm26 | I | 指令中的后25位 |
| rd1 | I | 从寄存器堆中读出的数据 |
| spc | O | 当前pc加4,用于jal指令保存地址 |
| npc | O | 下一条指令的地址 |

#### 2.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 计算下一条指令地址 | 如果当前指令是beq指令，并且zero为0，则pc←pc+4  如果当前指令是beq指令，并且zero为1，则pc←pc+4+{14'b0,imm26[15:0],2'b00};  如果当前指令是j指令或jal指令，则pc←{pc[31:28],imm26,2’b00}  如果当前指令是jr指令，则pc←rd1  其他情况，则pc←pc+4 |
| 2 | 输出当前指令地址加一 | 当前指令地址+4，并输出 |

### 3、regfile

#### 3.1基本描述

regfile以32个32位具有写使能的寄存器为基础，辅以多路选择器。其主要功能是对寄存器堆进行存值取值的操作。

#### 3.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号。 |
| a1 | I | 5位的地址输入，用于指定32个寄存器中的一个。在单周期CPU中为指令的[21:25]即rs字段 |
| a2 | I | 5位的地址输入，用于指定32个寄存器中的一个。在单周期CPU中为指令的[16:20]即rt字段。 |
| a3 | I | 5位的地址输入，用于指定32个寄存器中的一个。在单周期CPU中为指令的[11:15]即rd字段或[16:20]rt字段。 |
| enable | I | 写使能信号。  1：可向GPR写入数据。  0：只可从GPR读出数据。 |
| wd | I | 32位的数据输入，当写使能时，可以向寄存器中写入。 |
| rd1 | O | 输出32位的a1指定的寄存器的数据。 |
| rd2 | O | 输出32位的a2指定的寄存器的数据。 |

#### 3.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 复位 | 当复位信号有效时，32个寄存器中的值被置为0。 |
| 2 | 读数据 | 根据5位地址输入确定寄存器，并将寄存器中的数据输出。 |
| 3 | 写数据 | 当写使能时，根据5位地址输入确定寄存器，并将数据写入寄存器。 |

### 4、alu

#### 4.1基本描述

alu的主要功能是对输入到alu的两个数进行加法、减法、按位或、对其中的一个数移位等操作。alu除了以上操作外，还需要判断输入的两个值是否相等。

#### 4.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a | I | 参与alu计算的第一个值 |
| b | I | 参与alu计算的第二个值 |
| aluop[1:0] | I | alu功能的选择信号。  00：alu进行加法运算  01：alu进行减法运算  10：alu进行或运算  11：alu将第二个数左移16位 |
| aluout | O | alu计算的结果 |
| zero | O | 1：result为0  0：result非0 |

#### 4.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 加法运算 | 将a与b进行加法运算。aluout←a+b |
| 2 | 减法运算 | 将a与b进行减法运算。aluout←a-b |
| 3 | 或运算 | 将a与b进行按位或运算。aluout←a|b |
| 4 | 左移16位运算 | 将b左移16位。aluout←b<<16 |

### 5、ext

#### 5.1基本描述

将输入的数据改变位宽后输出，其中变化的为可以用0,1扩充，也可以把符号扩展，还可以额外输入一个数填充。

#### 5.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| extop | I | 选择扩展的类型。  0：按零扩展成32位  1：按符号位扩展成32位 |
| imm16 | I | 需要扩展的16位二进制。 |
| imm32 | O | 扩展后的数据 |

#### 5.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 按零扩展 | 在input的前面扩展16位0 |
| 2 | 按符号位扩展 | 根据input的符号位在input右移两位后得到数据的前面扩展16位1（符号位为1时）或16位0（符号位为0时） |

### 6、dm\_4k

#### 6.1基本描述

dm容量为32bit\*1024字。其主要功能是对数据存储器进行存值取值的操作。

#### 6.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr | I | 10位的地址输入，用于指定1024个字中的一个。 |
| din | I | 32位的数据输入，当写使能时，可以写入数据。 |
| we | I | 写使能信号。  1：可以向dm写数据  0：不可以向dm写数据 |
| clk | I | 时钟信号。 |
| dout | O | 输出32位的addr指定地址的数据。 |

#### 6.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 读数据 | 根据5位addr输入确定地址输出数据 |
| 2 | 写数据 | 当写使能时，根据5位addr输入确定地址写入数据。 |

### 7、im\_4k

#### 7.1基本描述

IM容量为32bit\*1024字。其主要功能是对指令存储器进行取值的操作。

#### 7.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr | I | 10位的地址输入，用于指定1024个字中的一个。 |
| dout | O | 输出32位的addr指定地址的指令。 |

#### 7.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 读指令 | 根据5位addr输入确定地址输出指令 |

### 8、control

#### 8.1基本描述

controller根据指令操作码字段op（指令31:26位）及func（指令5:0位）决定regdst、alusrc、memtoreg、regwrite、aluop(2位)、j、jal、jr、ifbeq等控制端口的状态。

#### 8.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| op | I | 6位的输入，在单周期CPU中为指令的[31:26]即Op字段。 |
| func | I | 6位的输入，在单周期CPU中为指令的[5:0]即Func字段。 |
| regdst | O | 寄存器堆写入端地址的选择控制信号。  00：选择Rt字段  01：选择Rd字段  10：选择31 |
| alusrc | O | ALU输入端B的选择控制信号。  0：选择寄存器堆输出端RD2  1：选择EXT输出端output |
| wd | O | 寄存器堆写入端数据的选择控制信号。  00：选择ALU的输出  01：选择DM的输出  10：选择PC+1 |
| regwrite | O | 寄存器堆写使能端的控制信号。  0：无法将数据写入寄存器堆  1：可以将数据写入寄存器堆中对应的寄存器 |
| we | O | 数据存储器DM写数据控制信号。  0：无法将数据写入数据存储器  1：可以将数据写入数据存储器 |
| ifbeq | O | npc的ifbeq信号。 |
| j | O | npc的j信号。 |
| jr | O | npc的jr信号。 |
| jal | O | npc的jal信号。 |
| extop | O | ext的控制信号。  0：ext选择按零位扩展  1：ext选择按符号位扩展 |
| aluop | O | alu的alu信号  00：alu进行加法运算  01：alu进行减法运算  10：alu进行或运算  11：alu将第二个数左移16位 |

#### 8.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 产生控制信号 | 根据op及func决定各个控制信号。 |

### 9、select5

#### 9.1基本描述

根据控制信号进行5位数据的选择

#### 9.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| en | I | 2位控制信号 |
| A | I | 第一个输入的数据 |
| B | I | 第二个输入的数据 |
| C | I | 第三个输入的数据 |
| out | O | 输出选择后的数据。 |

#### 9.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 选择 | en的信号  00：选择A  01：选择B  10：选择C  11：选择A(用不到) |

### 10、select32\_3

#### 10.1基本描述

根据控制信号进行32位数据的选择,输入为3个数

#### 10.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| en | I | 2位控制信号 |
| A | I | 第一个输入的数据 |
| B | I | 第二个输入的数据 |
| C | I | 第三个输入的数据 |
| out | O | 输出选择后的数据。 |

#### 10.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 选择 | en的信号  00：选择A  01：选择B  10：选择C  11：选择A(用不到) |

### 11、select32\_2

#### 11.1基本描述

根据控制信号进行32位数据的选择,输入为2个数

#### 11.2模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| en | I | 1位控制信号 |
| A | I | 第一个输入的数据 |
| B | I | 第二个输入的数据 |
| out | O | 输出选择后的数据。 |

#### 11.3功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 选择 | en的信号  0：选择A  1：选择B |

二、程序测试

1.程序代码及每行程序的解释

ori $0,$0,0 #立即数或，$寄存器0的值与0或，结果送给$0寄存器

ori $1,$0,1 #$1=1

ori $2,$0,2 #$2=2

ori $3,$0,3 #$3=3

ori $4,$0,4 #$4=4

ori $5,$0,5 #$5=5

sw $0,0x00000000 #$0存入第一个内存

sw $1,0x00000020 #$1存入第一个内存

sw $2,0x00000040 #$2存入第一个内存

sw $3,0x00000080 #$3存入第一个内存

sw $4,0x000000a0 #$4存入第一个内存

sw $5,0x000000c0 #$5存入第一个内存

lw $6,0x00000000 #取出内存中的数：$6=0

lw $7,0x00000020 #取出内存中的数：$7=1

lw $8,0x00000040 #取出内存中的数：$8=2

lw $9,0x00000080 #取出内存中的数：$9=3

lw $10,0x000000a0 #取出内存中的数：$10=4

lw $11,0x000000c0 #取出内存中的数：$11=5

lui $6,1 #把1加载到$6的高位

lui $7,2 #把1加载到$7的高位

beq $5,$11,go1 #$5=$0则转移到go1

addu $12,$1,$2 #$12=1+2=3,beq指令正确则不执行

go1:addu $12,$1,$3 #$12=1+3=4

beq $5,$1,go2 #$5!=$1,所以不应跳转

subu $13,$10,$3 #$13=4-3=1

go2:addu $14,$3,$9 #$14=3+3=6

ori $20,$0,6 #$20=6

ori $22,$0,3 #$22=3

jal go3 #转移到go3

addu $15,$3,$4 #$15=3+4=7

j go4 #转移到go4

ori $17,$0,2 #$17=2

go3:addu $16,$3,$5 #$16=3+5=8

jr $ra #跳回到jal后的第一条指令

go4:addu $17,$5,$11 #$17=5+5=10

go5:subu $20,$20,$1 #$20=$20-1=5

beq $20,$3,go6 #$20=3,go6

beq $22,$3,go7 #$22=3,then go7

go7:beq $11,$20,go5 #$11==$20时则转移到go5,then,$20=4

j go5 #跳转到go5，then $20=3

go8:addu$23,$1,$1 #$23=2

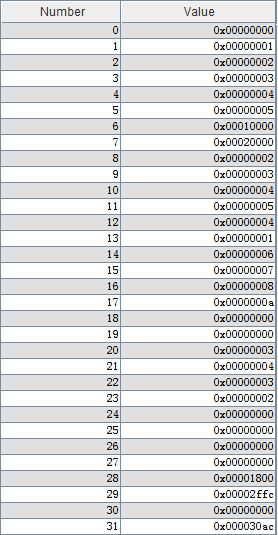
jr $ra #跳回到jal后的第一条指令

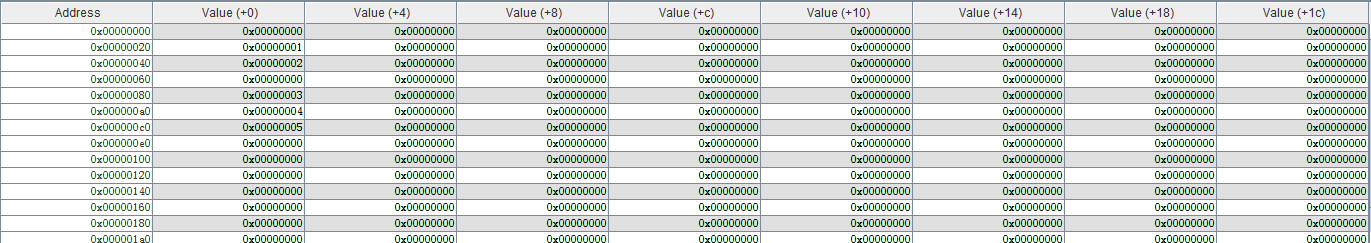
go6:jal go8 #跳转到go5，then $20=2

addu $21,$20,$1 #$21=2+1=3

2.程序预期结果

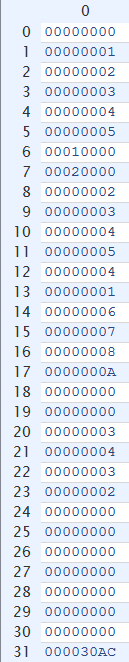
2.1寄存器：



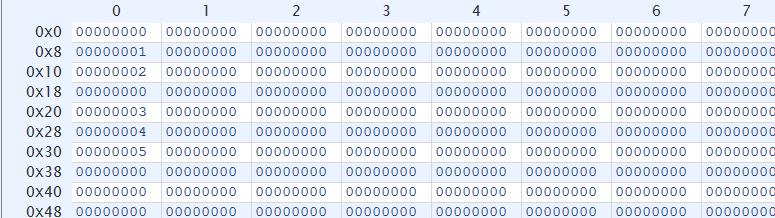
2.2内存：

3.实际运行结果

3.1寄存器：



3.2内存：



三、问答

在不考虑溢出的情况下，add与addu执行结果是一样的，即将指令中指定的两个寄存器中的值相加并存入指定的寄存器中。两者运行结果等价。若考虑溢出，addu仍执行上述工作，而add则报告溢出，不执行写入操作，二者运行结果不等价。

同理，在不考虑溢出的情况下，addi与addiu运行结果等价。