南 京 理 工 大 学

多周期CPU课程设计

|  |  |  |  |
| --- | --- | --- | --- |
| **姓 名:** | 蒋旭钊 | **学 号:** | 918106840727 |
| **学院(系):** | 计算机科学与工程学院 | | |
| **专 业:** | 计算机科学与技术 | | |
| **课 程:** | 硬件课程设计（Ⅰ） | | |

2021 年 10 月

**1.实验目的**

1. 在单周期 CPU 实验完成的提前下，理解多周期的概念。

2. 熟悉并掌握多周期 CPU 的原理和设计。

3. 进一步提升运用 verilog 语言进行电路设计的能力。

4. 为后续实现流水线 cpu 的课程设计打下基础。

**2.实验原理**

本次多周期CPU硬件课程设计是基于单周期CPU设计的一次拓展和拔高，如果我们找到多周期CPU和单周期CPU的差别，然后基于单周期已有的知识，设计多周期CPU将事倍功半。

单周期CPU会在一个时钟周期内执行一条指令，时钟周期需要匹配消耗时间最长的指令，不能有效地发挥CPU的效率。多周期CPU即将一条指令拆分成若干个阶段，有利于之后的流水线提高指令的执行效率。多周期CPU在处理指令时，通常需要以下几个阶段：

（1）取指令(IF)：根据程序计数器PC中的指令地址，从存储器中取出一条指令,同时，PC根据指令字长度自动递增产生下一条指令所需要的指令地址。

（2）指令译码(ID)：对取指令操作中得到的指令进行分析并译码。

（3）指令执行(EXE)：根据指令译码得到的操作控制信号，执行指令动作。

（4）存储器访问(MEM)：进行存储器的访问，把数据写入存储器或者从存储器中读出数据。

（5）结果写回(WB)：指令执行的结果写回到寄存器中。

不同的指令有不同的执行阶段，因此对应着不同的CPU执行周期，我们可以依据《数字逻辑电路》中学到的“自动状态机”的知识，写出每条指令对应的状态转换，依据状态转换进行相应的CPU设计。

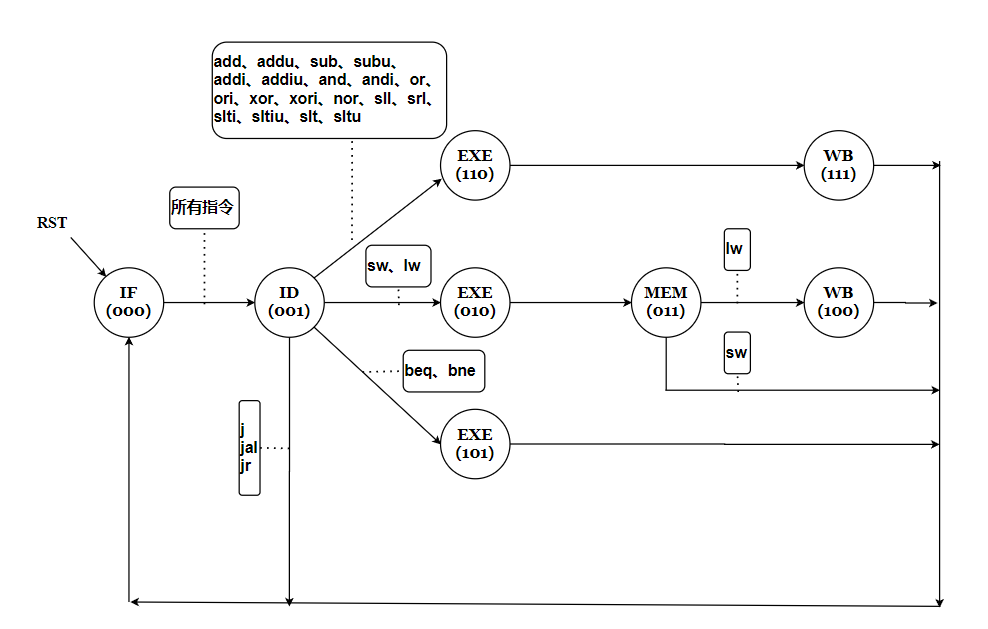
在状态转换的过程中，CPU需要对不同的指令执行不一样的操作，因此需要不同的命令发送给CPU，这就是对应着CPU的“控制信号”。

因此，我们只需要将实验分为主要的几个部分去攻克：

1. 在单周期CPU的基础上，设计多周期CPU特有的器件，如存储自动机状态的触发器。
2. 根据需要设计的指令，写出自动机状态转换图，设定相应的自动机状态。
3. 设计控制信号，能够保证CPU能够根据状态执行相应的动作。

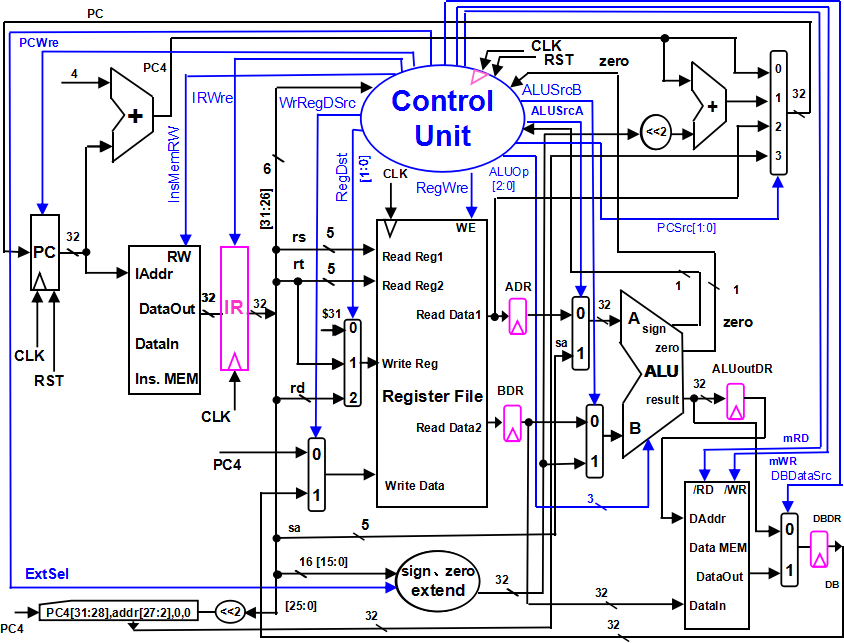
**3.实验设计**

我实现了近30余种指令的设计，并写出了它们的**状态转换图**：



**图1 多周期CPU状态转换图**

根据多周期CPU设计了相关的**数据通路和控制线路**：



**图2 多周期CPU数据通路和控制线路图**

**设计的CPU控制信号：**

|  |  |  |
| --- | --- | --- |
| **控制信号名** | **状态“0”** | **状态“1”** |
| **RST** | 对于PC，初始化 PC 为程序首地址 | 对于PC，PC 接收下一条指令地址 |
| **PCWre** | PC 不更改，相关指令：halt，另外， 除‘000’状态之外，其余状态慎  改 PC 的值。 | PC 更改，相关指令：除指令 halt 外， 另外，在‘000’状态时，修改PC 的值合适。 |
| **ALUSrcA** | 来自寄存器堆data1 输出，相关指令：add、sub、addiu、and、andi、ori、xori、slt、slti、sw、lw、beq、  bne | 来自移位数 sa，同时，进行(zero-extend)sa，即{{27{1'b0},sa}， 相关指令：sll |
| **ALUSrcB** | 来自寄存器堆data2 输出，相关指令：add、sub、and、slt、sll、  beq、bne | 来自 sign 或 zero 扩展的立即数，相关指令：addiu、andi、ori、xori、slti、lw、sw |
| **DBDataSrc** | 来自 ALU 运算结果的输出，相关指令：add、sub、addiu、and、andi、  ori、xori、sll、slt、slti | 来自数据存储器（Data MEM）的输出，相关指令：lw |
| **RegWre** | 无写寄存器组寄存器，相关指令：  beq、bne、j、sw、jr、halt | 寄存器组寄存器写使能，相关指令：  add、sub、addiu、and、andi、ori、xori、sll、slt、slti、lw、jal |
| **WrRegDSrc** | 写入寄存器组寄存器的数据来自  pc+4(pc4），相关指令：jal，写$31 | 写入寄存器组寄存器的数据来自 ALU 运算结果或存储器读出的数据，相关指令：add、addiu、sub、and、andi、ori、xori、sll、slt、slti、lw |
| **InsMemRW** | 写指令存储器 | 读指令存储器(Ins. Data) |
| **mRD** | 存储器输出高阻态 | **读数据存储器，相关指令：lw** |
| **mWR** | 无操作 | **写数据存储器，相关指令：sw** |
| **IRWre** | IR(指令寄存器)不更改 | IR 寄存器写使能。向指令存储器发出读指令代码后，这个信号也接着发出， 在时钟上升沿，IR 接收从指令存储器送来的指令代码。与每条指令都相关。 |
| **ExtSel** | (zero-extend)**immediate**，相关指令：andi、xori、ori； | (sign-extend)**immediate**，相关指令： addiu、slti、lw、sw、beq、bne； |
| **PCSrc[1..0]** | 00：pc<－pc+4，相关指令：add、addiu、sub、and、andi、ori、xori、slt、slti、sll、sw、lw、beq(zero=0)、bne(zero=1)；  01：pc<－pc+4+(sign-extend)**immediate ×4**，  相关指令：beq(zero=1)、bne(zero=0)；  10：pc<－rs，相关指令：jr；  11：pc<－{pc[31:28],addr[27:2],2'b00}，相关指令：j、jal； | |
| **RegDst[1..0]** | 写寄存器组寄存器的地址，来自：  00：0x1F($31)，相关指令：jal，用于保存返回地址（$31<-pc+4）；  01：rt 字段，相关指令：addiu、andi、ori、xori、slti、lw；  10：rd 字段，相关指令：add、sub、and、slt、sll；  11：未用； | |
| **ALUOp[3..0]** | ALU 12 种运算功能选择(0000-1011)，看功能表 | |

**表1 CPU控制信号表**

根据指令和控制信号相对应：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **opcode** | **func** | **ALUSrcA** | **ALUSrcB** | **DBDataSrc** | **WrRegDSrc** | **mRD** | **ExtSel** | **PCSrc[1:0]** | **RegDst[1:0]** | **ALUOp[3:0]** |
| add | 000000 | 100000 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0000 |
| addu | 000000 | 100001 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0000 |
| sub | 000000 | 100010 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0001 |
| subu | 000000 | 100011 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0001 |
| addi | 001000 | - | 0 | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 0000 |
| addiu | 001001 | - | 0 | 1 | 0 | 1 | 0 | 0 | 00 | 01 | 0000 |
| and | 000000 | 100100 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0101 |
| andi | 001100 | - | 0 | 1 | 0 | 1 | 0 | 0 | 00 | 01 | 0101 |
| or | 000000 | 100101 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0100 |
| ori | 001101 | - | 0 | 1 | 0 | 1 | 0 | 0 | 00 | 01 | 0100 |
| xor | 000000 | 100110 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 1010 |
| xori | 001110 | - | 0 | 1 | 0 | 1 | 0 | 0 | 00 | 01 | 1010 |
| nor | 000000 | 100111 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 1011 |
| sll | 000000 | 000000 | 1 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0010 |
| srl | 000000 | 000010 | 1 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0011 |
| slti | 001010 | - | 0 | 1 | 0 | 1 | 0 | 1 | 00 | 01 | 0111 |
| sltiu | 001011 | - | 0 | 1 | 0 | 1 | 0 | 0 | 00 | 01 | 0110 |
| slt | 000000 | 101010 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0111 |
| sltu | 000000 | 101011 | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 10 | 0110 |
| sw | 101011 | - | 0 | 1 | 0 | 1 | 0 | 1 | 00 | 00 | 0000 |
| lw | 100011 | - | 0 | 1 | 1 | 1 | 1 | 1 | 00 | 01 | 0000 |
| beq | 000100 | - | 0 | 0 | 0 | 1 | 0 | 1 | 01(zero)/00 | 00 | 0001 |
| bne | 000101 | - | 0 | 0 | 0 | 1 | 0 | 1 | 01(!zero)/00 | 00 | 0001 |
| j | 000010 | - | 0 | 0 | 0 | 1 | 0 | 0 | 11 | 00 | 0000 |
| jr | 000000 | 001000 | 0 | 0 | 0 | 1 | 0 | 0 | 10 | 00 | 0000 |
| jal | 000011 | - | 0 | 0 | 0 | 0 | 0 | 0 | 11 | 00 | 0000 |
| halt | 111111 | - | 0 | 0 | 0 | 1 | 0 | 0 | 00 | 00 | 0000 |

**表2 控制信号与指令对照表**

**其中的算术逻辑单元ALU：**

result，ALU 运算结果

zero，运算结果标志，结果为 0，则 zero=1；否则 zero=0

|  |  |  |
| --- | --- | --- |
| **ALUOp[3..0]** | **功能** | **描述** |
| **0000** | **Y = A+B** | **加** |
| **0001** | **Y = A-B** | **减** |
| **0010** | **Y = B<<A** | **B左移A 位** |
| **0011** | **Y = B>>A** | **B右移A 位（逻辑右移）** |
| **0100** | **Y = A|B** | **或** |
| **0101** | **Y = A&B** | **与** |
| **0110** | **Y=（A<B）?1: 0** | **比较 A<B**  **不带符号** |
| **0111** | **Y=(((A<B)&&(A[31] == B[31]))**  **||((A[31]==1&& B[31] == 0))) ? 1:0** | **比较 A<B**  **带符号** |
| **1000** | **Y=（A>B）?1: 0** | **比较 A>B**  **不带符号** |
| **1001** | **Y=(((A>B)&&(A[31] == B[31]))**  **||((A[31]==0&& B[31] == 1))) ? 1:0** | **比较 A>B**  **带符号** |
| **1010** | **Y = A^B** | **异或** |
| **1011** | **Y = ~（A|B）** | **或非** |

**表3 ALU功能表**

**该实验实现的MIPS指令如下：**

# ==>算术运算指令

1. add rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100000 |

功能：rd←rs + rt。

1. addu rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100001 |

功能：rd←rs + rt。

1. sub rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100010 |

功能：rd←rs - rt。

1. subu rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100011 |

功能：rd←rs - rt。

1. addi rt, rs, **immediate（符号拓展）**

|  |  |  |  |
| --- | --- | --- | --- |
| 001000 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

功能：rt←rs + (sign-extend)**immediate。**

1. addiu rt, rs, **immediate（0拓展）**

|  |  |  |  |
| --- | --- | --- | --- |
| 001001 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

功能：rt←rs + (zero-extend)**immediate。**

# ==>逻辑运算指令

1. and rd, rs , rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100100 |

功能：rd←rs & rt；逻辑与运算。

1. andi rt, rs ,**immediate（0拓展）**

|  |  |  |  |
| --- | --- | --- | --- |
| 001100 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

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

1. or rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100101 |

功能：rd←rs | rt；逻辑或运算。

1. ori rt, rs, **immediate（0拓展）**

|  |  |  |  |
| --- | --- | --- | --- |
| 001101 | 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。

1. xor rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100110 |

功能：rd←rs ^ rt；逻辑异或运算。

1. xori rt, rs, **immediate（0拓展）**

|  |  |  |  |
| --- | --- | --- | --- |
| 001110 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

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

1. nor rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 100111 |

功能：rd← ~（rs | rt）；逻辑或非运算。

# ==>移位指令

1. sll rd, rt,sa

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000000 | 00000 | rt(5 位) | rd(5 位) | sa（5位） | 000000 |

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

1. srl rd, rt,sa

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0000000 | 00000 | rt(5 位) | rd(5 位) | sa（5位） | 000010 |

功能：rd←rt>>(zero-extend)sa，右移 sa 位，(zero-extend)sa。

# ==>比较指令

1. slti rt, rs, **immediate**（符号拓展）

|  |  |  |  |
| --- | --- | --- | --- |
| 001010 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

功能：if (rs <(sign-extend)**immediate)** rt =1 else rt=0，具体请看ALU 运算功能表，带符号。

1. sltiu rt, rs, **immediate**（0拓展）

|  |  |  |  |
| --- | --- | --- | --- |
| 001011 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

功能：if (rs <(zero-extend)**immediate)** rt =1 else rt=0，具体请看ALU 运算功能表，不带符号。

1. slt rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 101010 |

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

1. sltu rd, rs, rt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | rt(5 位) | rd(5 位) | 00000 | 101011 |

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

# ==>存储器读写指令

1. sw rt, **immediate(**rs)

|  |  |  |  |
| --- | --- | --- | --- |
| 101011 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

功能：memory[rs+ (sign-extend)**immediate**]←rt。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。

1. lw rt, **immediate**(rs)

|  |  |  |  |
| --- | --- | --- | --- |
| 100011 | rs(5 位) | rt(5 位) | **immediate**(16 位) |

功能：rt ←memory[rs + (sign-extend)**immediate**]。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。

# ==>分支指令

1. beq rs,rt, **immediate** (说明：**immediate 从 pc+4 开始和转移到的指令之间间隔条数**）

|  |  |  |  |
| --- | --- | --- | --- |
| 000100 | 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。

1. bne rs,rt, **immediate** (说明：**immediate 从 pc+4 开始和转移到的指令之间间隔条数**）

|  |  |  |  |
| --- | --- | --- | --- |
| 000101 | 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。

# ==>跳转指令

（24）j addr

|  |  |
| --- | --- |
| 000010 | address(26位) |

功能：pc←{(pc+4)[31:28],address,2'b00}，跳转。

（25） jr rs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | rs(5 位) | 00000 | 00000 | 00000 | 001000 |

功能：pc←rs，跳转。

# ==>调用子程序指令

（26） jal addr

|  |  |
| --- | --- |
| 000011 | address(26位) |

功能：调用子程序，pc ← {(pc+4)[31:28], address, 2'b00}；$31←pc+4，返回地址设置；子程序返回，需用指令 jr $31

**4.实验过程与结果**

在多周期CPU的仿真过程中，我们按照执行顺序列出了以下的表格：

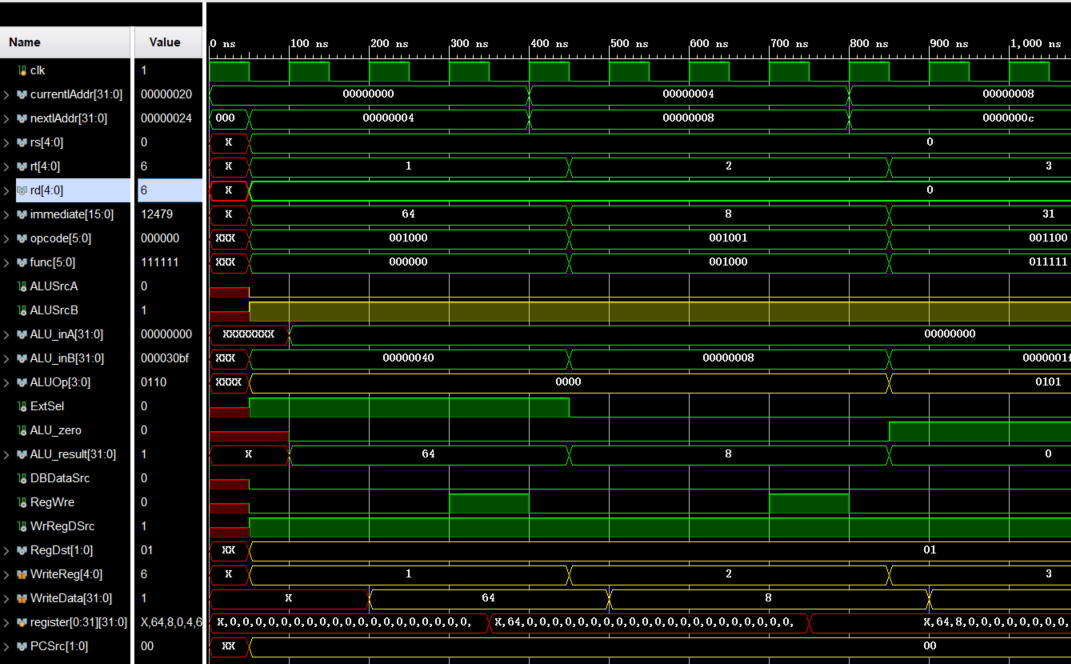
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 指令序号 | 指令地址 | 汇编程序 | 周期数 | 寄存器变化  （十进制） | 跳转情况 |
| 1 | 0x00000000 | addi $1,$0,64 | 4 | $1 = 64 |  |
| 2 | 0x00000004 | addiu $2,$0,8 | 4 | $2 = 8 |  |
| 3 | 0x00000008 | andi $3,$0,31 | 4 | $3 = 0 |  |
| 4 | 0x0000000C | ori $4,$0,4 | 4 | $4 = 4 |  |
| 5 | 0x00000010 | xori $5,$0,6 | 4 | $5 = 6 |  |
| 6 | 0x00000014 | slti $6,$5,7 | 4 | $6 = 1 |  |
| 7 | 0x00000018 | sltiu $7,$5,4 | 4 | $7 = 0 |  |
| 8 | 0x0000001C | sll $6,$6,2 | 4 | $6 = 4 |  |
| 9 | 0x00000020 | beq $6,$4,-2 | 3 |  | 相等,  转到0x0000001C |
| 10 | 0x0000001C | sll $6,$6,2 | 4 | $6 = 16 |  |
| 11 | 0x00000020 | beq $6,$4,-2 | 3 |  | 不等，  顺序执行 |
| 12 | 0x00000024 | add $7,$7,$5 | 4 | $7 = 6 |  |
| 13 | 0x00000028 | addu $7,$7,$5 | 4 | $7 = 12 |  |
| 14 | 0x0000002C | sub $7,$7,$5 | 4 | $7 = 6 |  |
| 15 | 0x00000030 | subu $7,$7,$5 | 4 | $7 = 0 |  |
| 16 | 0x00000034 | and $8,$4,$5 | 4 | $8 = 4 |  |
| 17 | 0x00000038 | or $9,$4,$5 | 4 | $9 = 6 |  |
| 18 | 0x0000003C | xor $10,$4,$5 | 4 | $10 = 2 |  |
| 19 | 0x00000040 | nor $11,$4,$5 | 4 | $11 = -7 |  |
| 20 | 0x00000044 | srl $1,$1,1 | 4 | $1 = 32 |  |
| 21 | 0x00000048 | bne $1,$6,-2 | 3 |  | 不等，  转到0x00000044 |
| 22 | 0x00000044 | srl $1,$1,1 | 4 | $1 = 16 |  |
| 23 | 0x00000048 | bne $1,$6,-2 | 3 |  | 相等，  顺序执行 |
| 24 | 0x0000004C | jal 0x0000060 | 2 | $31 = 0x00000050 | 转到0x00000060 |
| 25 | 0x00000060 | sw $1,4($s2) | 4 | $1 = 16 |  |
| 26 | 0x00000064 | lw $12,4($2) | 5 | $12 = 16 |  |
| 27 | 0x00000068 | jr $31 | 2 |  | 转到0x00000054 |
| 28 | 0x00000050 | slt $13,$4,$5 | 4 | $13 = 1 |  |
| 29 | 0x00000054 | sltu $14,$5,$4 | 4 | $14 = 0 |  |
| 30 | 0x00000058 | add $15,$13,$14 | 4 | $15 = 1 |  |
| 31 | 0x0000005C | j 0x0000006C | 2 |  | 转到0x0000006C |
| 32 | 0x0000006C | add $16,$13,$14 |  | $16 = 1 |  |

**表4 验证指令表**

具体的执行流程与验证如下，并针对其中具有代表性的几条指令，进行仿真波形图的分析和验证：

1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000000 | | 4 | | | $1=64 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| addi $1,$0,64 | | | | 00100000 00000001 00000000 01000000 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001000 | 00000 | | 00001 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000001000000 | | | | | | \ | | |



0-400ns，共花费4个时钟周期，当前PC地址为0x00000000。

IF状态，取出指令。

ID状态，进行指令的分析，由于是I指令，其中的rs,rt,immediate对应指令中的$0,$1,64；opcode和func也进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

EXE状态，进行指令的计算。其中ALUSrcA为0，代表ALU的输入A来自寄存器，对应rs；ALUSrcB为1，代表ALU的输入B来自立即数，对应immediate；ALUOp为0000，对应表3 ALU功能表中的“0000加”；ExtSel为1，代表立即数进行符号拓展；最终ALU\_result输出为64，ALU\_zero为0，说明result结果不为0。

WB状态，进行寄存器回写。DBDataSrc为0，代表DBDR中的数据来自ALU的运算结果输出（是为了和DBDataSrc为1，代表数据来自数据寄存器，对应lw指令区分）；RegWre为1，代表有写寄存器操作；WrRegDSrc为1，代表写入寄存器的操作来自ALU运算结果（是为了和WrRegDSrc为0的时候，代表写入寄存器数据来自PC+4，对应jal指令区分）；RegDst为01，代表写回的寄存器地址来自rt字段；WriteReg[4:0]和WriteData[31:0]与结果一致。Register中的寄存器1的结果成功变为64。

最终PCSrc为00，代表接下来的指令PC是PC+4。

2.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000004 | | 4 | | | $2 = 8 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| addiu $2,$0,8 | | | | 00100100 00000010 00000000 00001000 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001001 | 00000 | | 00010 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000001000 | | | | | | \ | | |

与指令1（addi $1,$0,64）类似，主要区别在于ExtSel为0，进行无符号拓展，具体对应表2 控制信号与指令对照表。

3.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000008 | | 4 | | | $3 = 0 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| andi $3,$0,31 | | | | 00110000 00000011 00000000 00011111 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001100 | 00000 | | 00011 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000011111 | | | | | | \ | | |

与指令1（addi $1,$0,64）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

4.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000000C | | 4 | | | $4 = 4 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| ori $4,$0,4 | | | | 00110100 00000100 00000000 00000100 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001101 | 00000 | | 00100 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000000100 | | | | | | \ | | |

与指令1（addi $1,$0,64）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

5.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000010 | | 4 | | | $5 = 6 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| xori $5,$0,6 | | | | 00111000 00000101 00000000 00000110 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001110 | 00000 | | 00101 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000000110 | | | | | | \ | | |

与指令1（addi $1,$0,64）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

6.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000014 | | 4 | | | $6 = 1 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| slti $6,$5,7 | | | | 00101001 00100110 00000000 00000111 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001010 | 01001 | | 00110 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000000111 | | | | | | \ | | |

与指令1（addi $1,$0,64）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

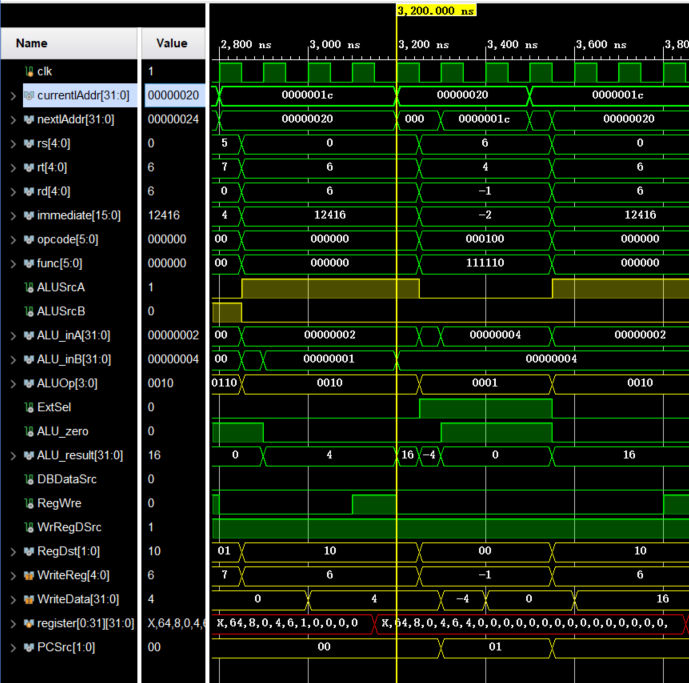
7.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000018 | | 4 | | | $7 = 0 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| sltiu $7,$5,4 | | | | 00101100 10100111 00000000 00000100 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 001011 | 00101 | | 00111 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000000100 | | | | | | \ | | |

与指令1（addi $1,$0,64）类似，主要区别在于ALU功能不同和ExtSel不一样，具体对应表2 控制信号与指令对照表。

8.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000001C | | 4 | | | $6 = 4 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| sll $6,$6,2 | | | | 00000000 00000110 00110000 10000000 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00000 | | 00110 | | | 00110 | 00010 | 000000 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

 2800-3200ns，共花费4个时钟周期，当前PC地址为0x0000001C。

IF状态，取出指令。

ID状态，进行指令的分析，由于是R指令，其中的rt,rd对应指令中的$6,$6；opcode和func也进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

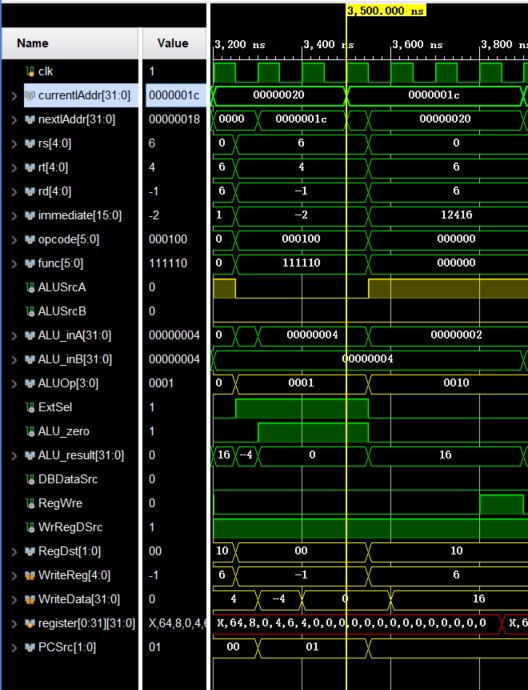
EXE状态，进行指令的计算。其中ALUSrcA为1，代表ALU的输入A来自移位数，对应shamt；ALUSrcB为0，代表ALU的输入B来自寄存器，对应rt；ALUOp为0010，对应表3 ALU功能表中的“0010左移”；ExtSel为0，代表立即数进行零拓展；最终ALU\_result输出为4，ALU\_zero为0，说明result结果不为0。

WB状态，进行寄存器回写。DBDataSrc为0，代表DBDR中的数据来自ALU的运算结果输出（是为了和DBDataSrc为1，代表数据来自数据寄存器，对应lw指令区分）；RegWre为1，代表有写寄存器操作；WrRegDSrc为1，代表写入寄存器的操作来自ALU运算结果（是为了和WrRegDSrc为0的时候，代表写入寄存器数据来自PC+4，对应jal指令区分）；RegDst为10，代表写回的寄存器地址来自r字段；WriteReg[4:0]和WriteData[31:0]与结果一致。Register中的寄存器6的结果成功变为4。

最终PCSrc为00，代表接下来的指令PC是PC+4。

9.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000020 | | 3 | | | 转到0x0000001C | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| beq $6,$4,-2 | | | | 00010000 11000100 11111111 11111110 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000100 | 00110 | | 00100 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 1111111111111110 | | | | | | \ | | |



3200-3500ns，共花费3个时钟周期，当前PC地址为0x00000020。

IF状态，取出指令。

ID状态，进行指令的分析，由于是I指令，其中的rs,rt,immediate对应指令中的$6,$4,-2；opcode和func也进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

EXE状态，进行指令的计算。其中ALUSrcA为0，代表ALU的输入A来自寄存器，对应rs；ALUSrcB为0，代表ALU的输入B来自寄存器，对应rt；ALUOp为0001，对应表3 ALU功能表中的“0001减”；最终ALU\_result输出为0，ALU\_zero为1，说明result结果为0，需要进行跳转。

最终PCSrc为01，ExtSel为1，代表立即数进行符号拓展并左移两位在于当前PC相加，作为下一PC地址。在时钟下降沿到来，PCWre变为1，下一条指令跳转到0x0000001C。

10.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000001C | | 4 | | | $6 = 16 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| sll $6,$6,2 | | | | 00000000 00000110 00110000 10000000 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00000 | | 00110 | | | 00110 | 00010 | 000000 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令8（sll $6,$6,2）相同。

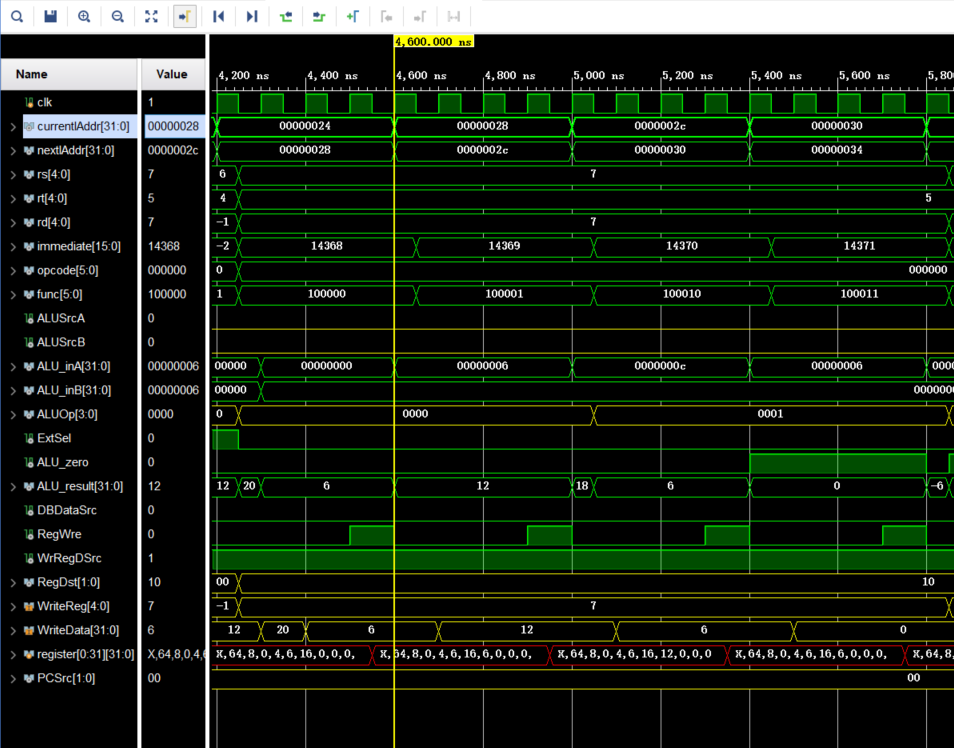
11.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000020 | | 3 | | | 顺序执行 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| beq $6,$4,-2 | | | | 00010000 11000100 11111111 11111110 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000100 | 00110 | | 00100 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 1111111111111110 | | | | | | \ | | |

与指令9（beq $6,$4,-2）类似，主要区别在于ALU运算结果不为0，不构成跳转条件，具体对应表2 控制信号与指令对照表。

12.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000024 | | 4 | | | $7 = 6 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| add $7,$7,$5 | | | | 00000000 11100101 00111000 00100000 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00111 | | 00101 | | | 00111 | 00000 | 100000 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

 4200-4600ns，共花费4个时钟周期，当前PC地址为0x00000024。

IF状态，取出指令。

ID状态，进行指令的分析，由于是R指令，其中的rs,rt,rd对应指令中的$7,$5,$7；opcode和func也进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

EXE状态，进行指令的计算。其中ALUSrcA为0，代表ALU的输入B来自寄存器，对应rs；ALUSrcB为0，代表ALU的输入B来自寄存器，对应rt；ALUOp为0000，对应表3 ALU功能表中的“0000加”；ExtSel为0，代表立即数进行零拓展；最终ALU\_result输出为6，ALU\_zero为0，说明result结果不为0。

WB状态，进行寄存器回写。DBDataSrc为0，代表DBDR中的数据来自ALU的运算结果输出（是为了和DBDataSrc为1，代表数据来自数据寄存器，对应lw指令区分）；RegWre为1，代表有写寄存器操作；WrRegDSrc为1，代表写入寄存器的操作来自ALU运算结果（是为了和WrRegDSrc为0的时候，代表写入寄存器数据来自PC+4，对应jal指令区分）；RegDst为10，代表写回的寄存器地址来自r字段；WriteReg[4:0]和WriteData[31:0]与结果一致。Register中的寄存器7的结果成功变为6。

最终PCSrc为00，代表接下来的指令PC是PC+4。

13.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000028 | | 4 | | | $7 = 12 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| addu $7,$7,$5 | | | | 00000000 11100101 00111000 00100001 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00111 | | 00101 | | | 00111 | 00000 | 100001 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

14.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000002C | | 4 | | | $7 = 6 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| sub $7,$7,$5 | | | | 00000000 11100101 00111000 00100010 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00111 | | 00101 | | | 00111 | 00000 | 100010 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

15.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000030 | | 4 | | | $7 = 0 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| subu $7,$7,$5 | | | | 00000000 11100101 00111000 00100011 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00111 | | 00101 | | | 00111 | 00000 | 100011 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同以及符号拓展方式不同，具体对应表2 控制信号与指令对照表。

16.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000034 | | 4 | | | $8=4 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| and $8,$4,$5 | | | | 00000000 10000101 01000000 00100100 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00100 | | 00101 | | | 01000 | 00000 | 100100 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

17.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000038 | | 4 | | | $9=6 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| or $9,$4,$5 | | | | 00000000 10000101 01001000 00100101 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00100 | | 00101 | | | 01001 | 00000 | 100101 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

18.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000003C | | 4 | | | $10=2 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| xor $10,$4,$5 | | | | 00000000 10000101 01010000 00100110 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00100 | | 00101 | | | 01010 | 00000 | 100110 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

19.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000040 | | 4 | | | $11=-7 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| nor $11,$4,$5 | | | | 00000000 10000101 01011000 00100111 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00100 | | 00101 | | | 01011 | 00000 | 100111 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

20.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000044 | | 4 | | | $1 = 32 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| srl $1,$1,1 | | | | 00000000 00000001 00001000 01000010 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00000 | | 00001 | | | 00001 | 00001 | 000010 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令8（sll $6,$6,2）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

21.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000048 | | 3 | | | 转到0x00000044 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| bne $1,$6,-2 | | | | 00010100 00100110 11111111 11111110 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000101 | 00001 | | 00110 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 1111111111111110 | | | | | | \ | | |

与指令9（beq $6,$4,-2）类似，主要区别在于判断条件不同，具体对应表2 控制信号与指令对照表。

22.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000044 | | 4 | | | $1 = 16 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| srl $1,$1,1 | | | | 00000000 00000001 00001000 01000010 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00000 | | 00001 | | | 00001 | 00001 | 00010 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令20（srl $1,$1,1）相同。

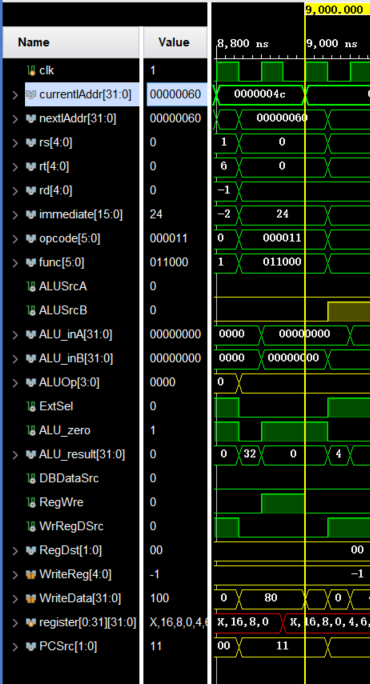
23.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000048 | | 3 | | | 顺序执行 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| bne $1,$6,-2 | | | | 00010100 00100110 11111111 11111110 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000101 | 00001 | | 00110 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 1111111111111110 | | | | | | \ | | |

与指令21（bne $1,$6,-2）类似，主要区别在于ALU运算结果为0，不构成跳转条件，具体对应表2 控制信号与指令对照表。

24.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000004C | | 2 | | | $31 = 0x00000050 转到0x00000060 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| jal 0x0000060 | | | | 00001100 00000000 00000000 00011000 | | | | J |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000011 | \ | | \ | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | 00000000000000000000011000 | | |



8800-9000ns，共花费2个时钟周期，当前PC地址为0x0000004C。

IF状态，取出指令。

ID状态，进行指令的分析。opcode和func进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。WrRegDSrc为0的时候，代表写入寄存器数据来自PC+4。RegDst为00，代表写回的寄存器地址为0x1f，写入31号寄存器。

最终PCSrc为11，代表接下来的指令PC是需要跳转到addr拼接地址。在时钟下降沿到来，PCWre变为1，下一条指令跳转到0x00000060。

25.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000060 | | 4 | | | $1 = 16 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| sw $1,4($s2) | | | | 10101100 01000001 00000000 00000100 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 101011 | 00010 | | 00001 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000000100 | | | | | | \ | | |



9000-9400ns，共花费4个时钟周期，当前PC地址为0x00000060。

IF状态，取出指令。

ID状态，进行指令的分析，由于是I指令，其中的rs,rt,immediate对应指令中的$2,$1,4；opcode和func也进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

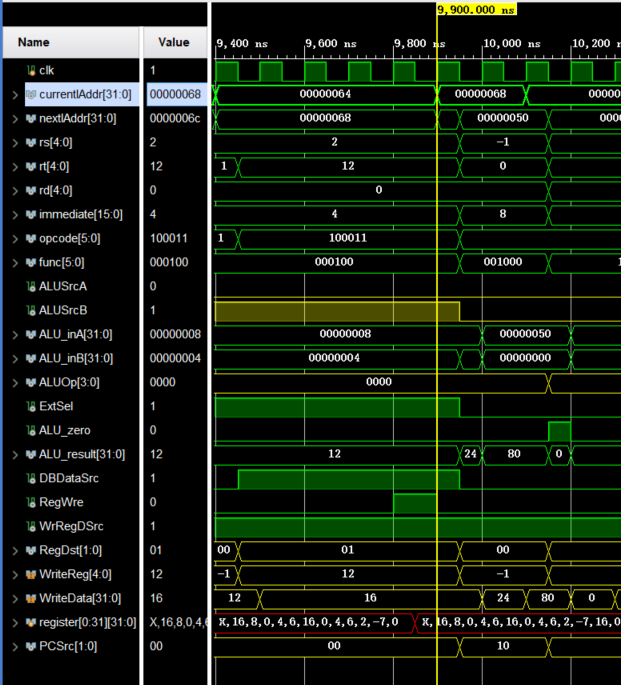
EXE状态，进行指令的计算。其中ALUSrcA为0，代表ALU的输入A来自寄存器，对应rs；ALUSrcB为1，代表ALU的输入B来自立即数，对应immediate；ALUOp为0000，对应表3 ALU功能表中的“0000加”；ExtSel为1，代表立即数进行符号拓展；最终ALU\_result输出为12，代表要存入的地址，ALU\_zero为0，说明result结果不为0。

MEM状态，进行存储器访问。DBDataSrc为0，代表DBDR中的数据来自ALU的运算结果输出（是为了和DBDataSrc为1，代表数据来自数据寄存器，对应lw指令区分）；RegWre为0，代表无写寄存器操作；WrRegDSrc为1，代表写入寄存器的操作来自ALU运算结果（是为了和WrRegDSrc为0的时候，代表写入寄存器数据来自PC+4，对应jal指令区分）。1号寄存器中的16被存入对应的地址。

最终PCSrc为00，代表接下来的指令PC是PC+4。

26.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000064 | | 5 | | | $12 = 16 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| lw $12,4($2) | | | | 10001100 01001100 00000000 00000100 | | | | I |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 100011 | 00010 | | 01100 | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| 0000000000000100 | | | | | | \ | | |

900 9400-9900ns，共花费5个时钟周期，当前PC地址为0x00000064。

IF状态，取出指令。

ID状态，进行指令的分析，由于是I指令，其中的rs,rt,immediate对应指令中的$2,$12,4；opcode和func也进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

EXE状态，进行指令的计算。其中ALUSrcA为0，代表ALU的输入A来自寄存器，对应rs；ALUSrcB为1，代表ALU的输入B来自立即数，对应immediate；ALUOp为0000，对应表3 ALU功能表中的“0000加”；ExtSel为1，代表立即数进行符号拓展；最终ALU\_result输出为12，代表要取出数据的地址，ALU\_zero为0，说明result结果不为0。

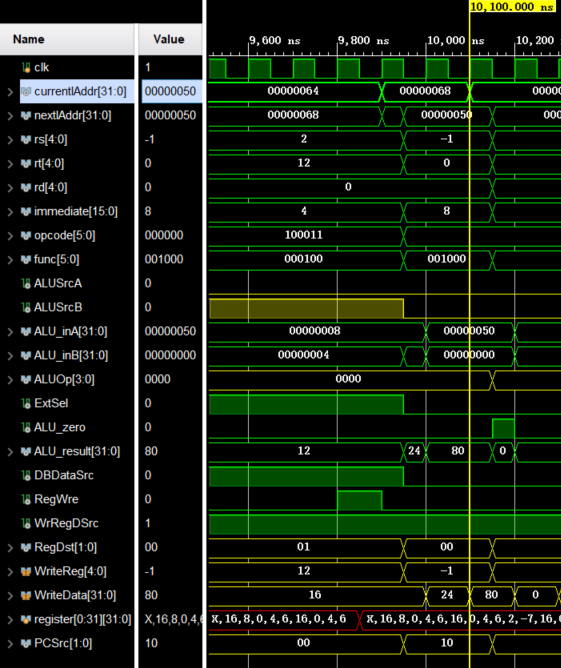
MEM状态，进行存储器访问。根据ALU\_result计算出来的数据地址，从中取出数据，DBDataSrc为1，代表数据来自数据寄存器。

WB状态，进行寄存器写回操作。RegWre为1，代表有写寄存器操作；WrRegDSrc为1，代表写入寄存器的操作来自ALU运算结果（是为了和WrRegDSrc为0的时候，代表写入寄存器数据来自PC+4，对应jal指令区分）;RegDst[1:0]为01，代表写入的寄存器地址来自rt字段，对应$12;WriteReg[4:0]和WriteData[31:0]与结果一致。Register中的寄存器12的结果成功变为16。与上一条lw指令存入的数据一致。

最终PCSrc为00，代表接下来的指令PC是PC+4。

27.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000068 | | 2 | | | 转到0x00000050 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| jr $31 | | | | 00000011 11100000 00000000 00001000 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 11111 | | 00000 | | | 00000 | 00000 | 001000 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |



9900-10100ns，共花费2个时钟周期，当前PC地址为0x00000068。

IF状态，取出指令。

ID状态，进行指令的分析。opcode和func进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。

最终PCSrc为10，代表接下来的指令PC是需要跳转到31号寄存器存入的地址。在时钟下降沿到来，PCWre变为1，下一条指令重新跳转到0x00000050。

28.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000050 | | 4 | | | $13 = 1 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| slt $13,$4,$5 | | | | 00000000 10000101 01101000 00101010 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00100 | | 00101 | | | 01101 | 00000 | 101010 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于ALU功能不同，具体对应表2 控制信号与指令对照表。

29.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000054 | | 4 | | | $14 = 0 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| sltu $14,$5,$4 | | | | 00000000 10100100 01110000 00101011 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 00101 | | 00100 | | | 01110 | 00000 | 101011 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令28（slt $13,$4,$5）类似，主要区别在于寄存器号不同。

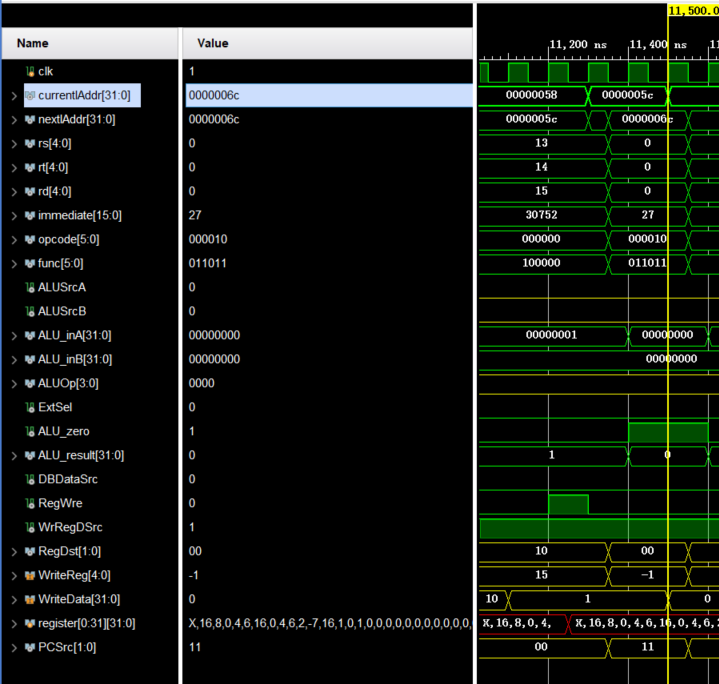
30.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x00000058 | | 4 | | | $15 = 1 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| add $15,$13,$14 | | | | 00000001 10101110 01111000 00100000 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 01101 | | 01110 | | | 01111 | 00000 | 100000 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于寄存器号不同。

31.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000005C | | 2 | | | 转到0x0000006C | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| j 0x000006C | | | | 00001000 00000000 00000000 00011011 | | | | J |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000010 | \ | | \ | | | \ | \ | \ |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | 00000000000000000000011011 | | |



11200-11400ns，共花费2个时钟周期，当前PC地址为0x0000005C。

IF状态，取出指令。

ID状态，进行指令的分析。opcode和func进行了正确的解析，与controller中的MIPS指令判断代码吻合一致。但是与jal指令不同的是，这里不需要将地址写入31号寄存器。

最终PCSrc为11，代表接下来的指令PC是需要跳转到addr拼接地址。在时钟下降沿到来，PCWre变为1，下一条指令跳转到0x0000006C。

32.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令地址 | | 周期数 | | | 变化 | | | |
| 0x0000006C | | 4 | | | $16 = 1 | | | |
| 指令 | | | | 二进制码 | | | | 指令类型 |
| add $16,$13,$14 | | | | 00000001 10101110 10000000 00100000 | | | | R |
| op(31-26) | rs(25-21) | | rt(20-16) | | | rd(15-11) | shamt(10-6) | func(5-0) |
| 000000 | 01101 | | 01110 | | | 10000 | 00000 | 100000 |
| Immediate(15-0) | | | | | | Address(25-0) | | |
| \ | | | | | | \ | | |

与指令12（add $7,$7,$5）类似，主要区别在于寄存器号不同。

**5.实验心得**

（1）本次多周期CPU实验是我基于单周期CPU实验进行的改进工作，我们只需要弄清楚这两个实验的相似和差异之处，就可以理清其中CPU部件的许多功能。InstructionMemory、RegisterFile、ALU等模块实现是想通的，其中ALU的功能我们可以根据自己的需要进行拓展（比如有符号比较和无符号比较，基本逻辑运算等）。

（2）Verilog语言的调试方法可以依据“从上往下差错，从下往上改错”的方式进行，当上层的仿真波形出现错误的时候，我们可以先将对应模块中的所有变量和信号展示出来，然后到对应的源文件中查看是否逻辑有错误，或者是变量未定义。在修改完以后，在从下往上进行相关的接口和变量名的改变，最终查看仿真波形输出是否正确。

（3）通过该实验，我更加加深了对“数字逻辑电路”和“计算机组成原理”的理解，理解了CPU中各种模块的功能及其实现、数据通路的功能、以及数据选择器在其中发挥的作用（包括PC地址的改变方式，ALU功能的选择等等）。