# Project1 MIPS多周期处理器

一．实验要求

1.支持7条指令：addu，subu，ori，lw，sw，beq，jal。

2.处理器为多周期设计，运算指令可以暂不支持溢出。

二．详细设计

**1.设计分析：**

多周期处理器与单周期处理器一样都是每条指令依次执行。不同的是，多周期处理器将1条MIPS指令的执行过程分成5个阶段，每个阶段用一个时钟周期来完成，每条指令执行的时钟数不尽相同。

一般五个阶段为：IF -> ID -> EX -> MEM -> WB

注：IF：从指令存储器中读取指令

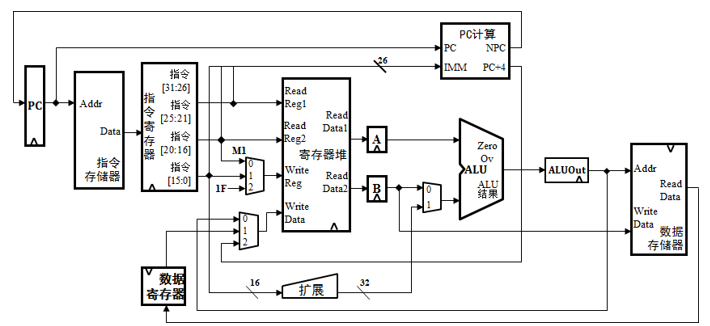
ID：指令译码的同时读取寄存器

EX：执行操作或计算地址

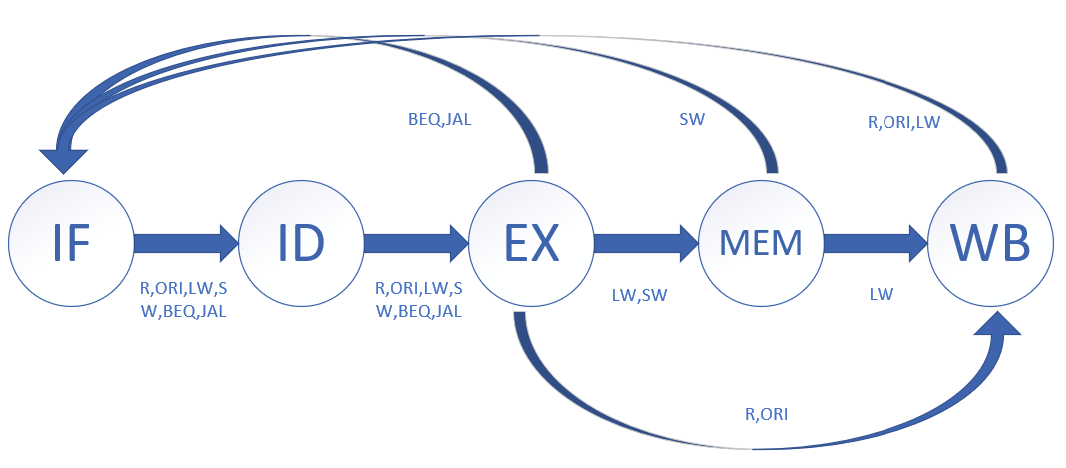
MEM：从数据存储器中读取操作数

WB：将结果写回寄存器

**2.数据通路图**：



**3.状态机设计**：



**4.模块说明**

***4.1 flop异步复位触发器***

可实例化为RFData1，RFData2，ALUout，Dataout等无写使能信号的寄存器。当实例化flop时，可使用#（x），实例化位宽是x的触发器。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Rst | I | 复位信号（高位有效） |
| Din | I | 输入数据 |
| Dout | O | 输出数据 |

***4.2 mux2to1二选一选择器***

当实例化mux2to1时，可使用#（x），实例化位宽是x的选择器。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A，B | I | 供选择数据 |
| Out | I | 片选后数据 |
| Ctrl | O | 片选信号 |

***4.3 mux3to1 三选一选择器***

当实例化mux3to1时，可使用#（x），实例化位宽是x的选择器。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A，B，C | I | 供选择数据 |
| Out | I | 片选后数据 |
| Ctrl | O | 片选信号 |

***4.4 pc 程序寄存器***

寄存当前需执行指令的地址。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Din[31:2] | I | 地址输入 |
| Clk | I | 时钟信号 |
| Rst | I | 复位信号 |
| Dout[31:0] | O | 地址输出 |
| PCWrite\_en | i | 写使能信号 |

***4.5 im 程序存储器***

容量为4KB（32bit\*1024），即1024字。存储指令。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[11:2] | I | 地址输入 |
| Dout[31:0] | O | 指令输出 |

***4.6 ir 指令寄存器***

寄存从im取到的指令。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Instr\_in[31:0] | I | 指令输入 |
| Clk | I | 时钟信号 |
| Rst | I | 复位信号 |
| Instr\_out[31:0] | O | 指令输出 |

***4.7 rf寄存器堆***

32个32位寄存器，支持读与写。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| R1\_addr,R2\_addr[4:0] | I | 读地址输入 |
| R1\_data,R2\_data[31:0] | O | 读数据输出 |
| W\_addr[4:0] | I | 写地址输入 |
| W\_data[31:0] | I | 写数据输入 |
| Clk | I | 时钟信号 |
| RegWrite\_en | I | 写使能信号 |

***4.8 alu 算术逻辑单元***

完成算术逻辑运算，包括加，减，与，或等操作。通过zero信号结合减法可判断两数是否相等。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Data1,Data2[31:0] | I | 数据输入 |
| Result[31:0] | O | 结果输出 |
| Zero | O | 零标志位 |
| Aluctrl[3:0] | I | 控制ALU执行的操作 |

***4.9 ext 扩展模块***

将16位的数据扩展至32位，可选择有符号扩展或无符号扩展。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Datain[15:0] | I | 数据输入 |
| Dataout[31:0] | O | 扩展后数据 |
| Zero\_sign | I | 选择扩展方式 |

***4.10 dm 数据存储器***

容量为4KB（32bit\*1024），即1024字。存储数据。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[11:2] | I | 地址输入 |
| Din[31:0] | I | 写数据输入 |
| DMWr | I | 写使能信号 |
| Clk | I | 时钟信号 |
| Dout[31:0] | O | 读数据输出 |

***4.11 npc 下一条指令地址计算模块***

计算下一条指令的地址，有三种情况：pc+4，beq指令，jal指令

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Pc[31:2] | I | 输入PC |
| Jaddr[25:0] | I | Jal指令的立即数 |
| Offset[31:0] | I | 扩展后的beq偏移 |
| Zero | I | 零标志位 |
| Pc\_4[31:2] | O | 输出PC+4 |
| Npcctrl[1:0] | I | 选择三种情况中的一种 |
| Nextpc[31:2] | O | 输出下一条指令的地址 |

***4.12 ctrl 指令译码模块***

通过使用状态机，根据输入指令的opcode段控制数据通路各模块的控制信号。状态机在上方已给出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Instr[31:26] | I | 指令opcode段输入 |
| Clk | I | 时钟信号 |
| Rst | I | 复位信号 |
| PCWrite\_en | O | PC写使能 |
| RegDst[1:0] | O | RF写地址选择  00表示rd  01表示rt  10表示31号寄存器 |
| RegWrite[1:0] | O | RF写数据选择  00表示Alu结果  01表示数据寄存器结果  10表示PC+4 |
| RegWrite\_en | O | RF写使能 |
| Zero\_sign | O | EXT扩展方式选择  0表示无符号扩展  1表示有符号扩展 |
| AluOp[1:0] | O | Alu二级译码的控制  00表示加  01表示减  10表示或  11表示由指令funct段决定 |
| AluSrc | O | Alu数据来源选择  0表示寄存器  1表示扩展后立即数 |
| MemWrite\_en | O | Mem写使能 |
| NPCSrc[1:0] | O | Npc计算方式选择  00表示PC+4  01表示Beq型指令  10表示Jal型指令 |

***4.13 aluctrl Alu二级译码模块***

对来自ctrl的AluOp进行二级译码。

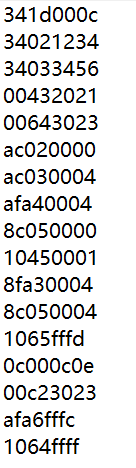
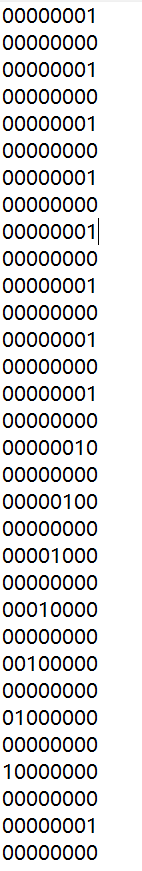
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Funct[5:0] | I | 指令的funct段 |
| AluOp[1:0] | I | 来自ctrl的控制信号 |
| Aluctrl | O | 输出Alu控制信号 |

**三．仿真验证**

**1.验证说明**

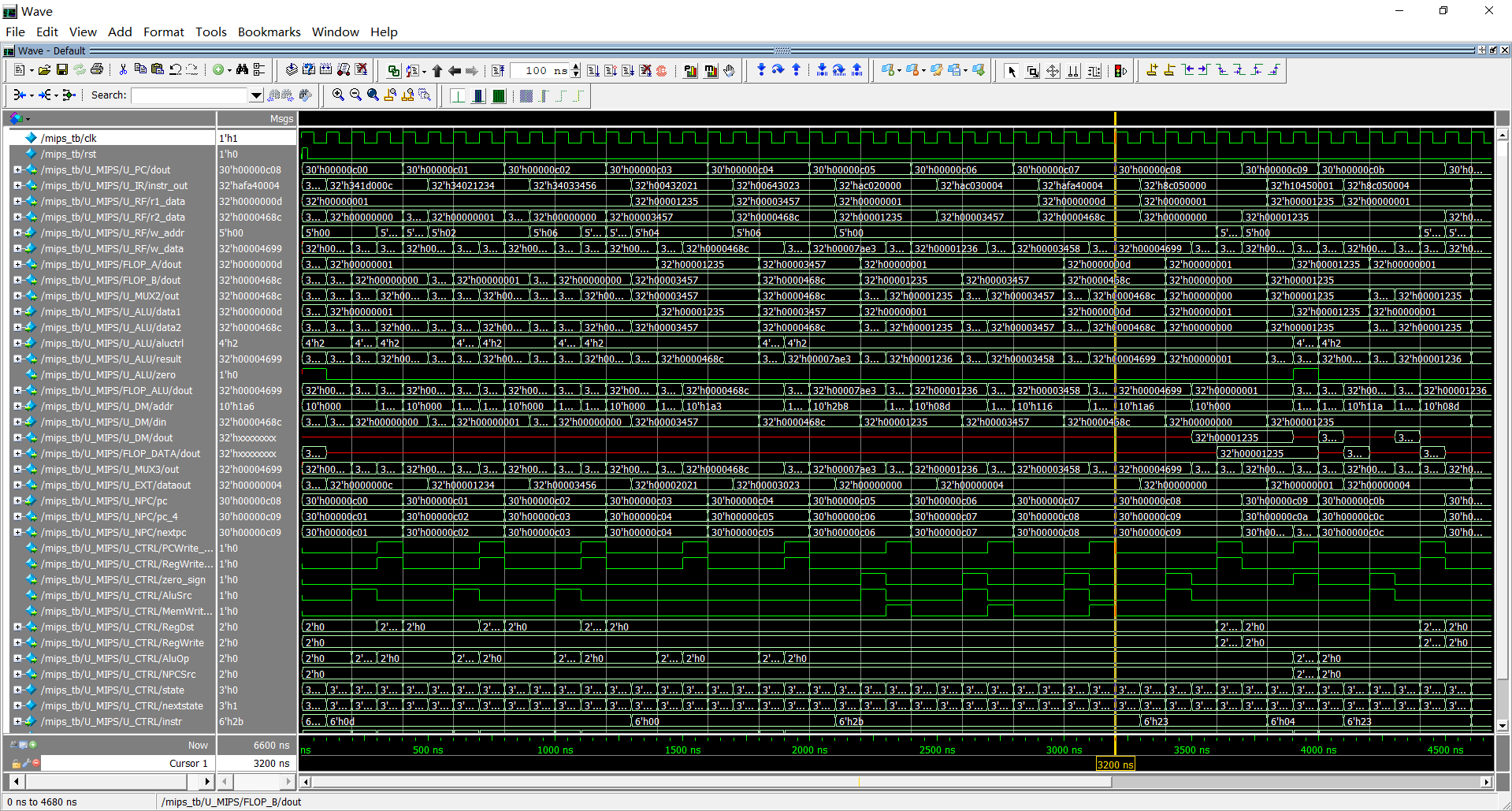
验证环境：ModelSim 10.4

复位后PC位于0x0000\_3000

上图为code 上图为32个寄存器初始化值

**2.验证波形图（部分）**



**3.逐指令分析验证**

* 1. ***0x341d000c***

即001101 00000 11101 0000000000001100

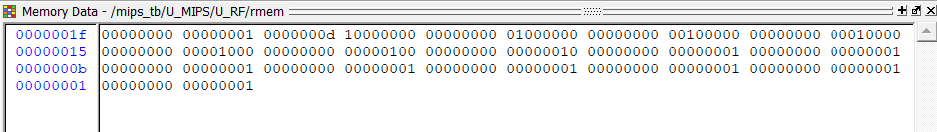
即**ori $0 $29 0000000000001100**

周期1 IF ：取指令，IR值变为0x341d000c

周期2 ID ：指令译码，读出$0的值为0x0000\_0001，保存至FLOP\_A

周期3 EX ：执行，通过FLOP\_A（0x0000\_0001）与无符号扩展后的立即数的或操作，得到0x0000\_000d，并保存至FLOP\_ALU

周期4 WB ：写回，将0x0000\_000d写入$29, PC变为0x0000\_3004

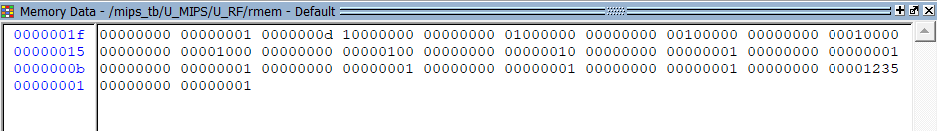


* 1. ***0x34021234***

即001101 00000 00010 0001001000110100

即**ori $0 $2 0001001000110100**

过程与1类似，将$0 (0000\_0001) 与无符号扩展后的立即数 (0000\_1234) 的或写入寄存器$2 (0000\_1235) ，PC变为0x0000\_3008

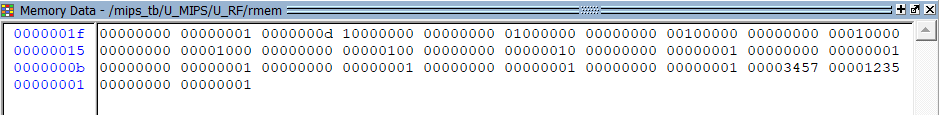


* 1. ***0x34033456***

即001101 00000 00011 0011010001010110

即**ori $0 $3 0011010001010110**

过程与1类似，将$0 (0000\_0001) 与无符号扩展后的立即数 (0000\_3456) 的或写入寄存器$3 (0000\_3457) ，PC变为0x0000\_300c



* 1. ***0x00432021***

即000000 00010 00011 00100 00000 100001

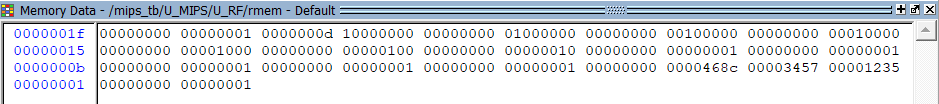
即**addu $2 $3 $4**

周期1 IF：取指令，IR值变为0x00432021

周期2 ID：指令译码，读出$2的值为0000\_1235，保存至FLOP\_A，读出$3的值0000\_3457，保存至FLOP\_B

周期3 EX：执行，将FLOP\_A与FLOP\_B的值相加得到0x0000\_468c，保存至FLOP\_ALU

周期4 WB：写回，将0000\_468c写入$4，PC变为0x0000\_3010

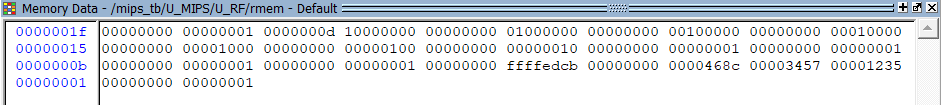


* 1. ***0x00643023***

即000000 00011 00100 00110 00000 100011

即**subu $3 $4 $6**

过程与4类似，将$3 (0000\_3457) 减去 $4 (0000\_468c) 得到ffff\_edcb，写入寄存器$6，PC变为0x0000\_3014



* 1. ***0xac020000***

即101011 00000 00010 0000000000000000

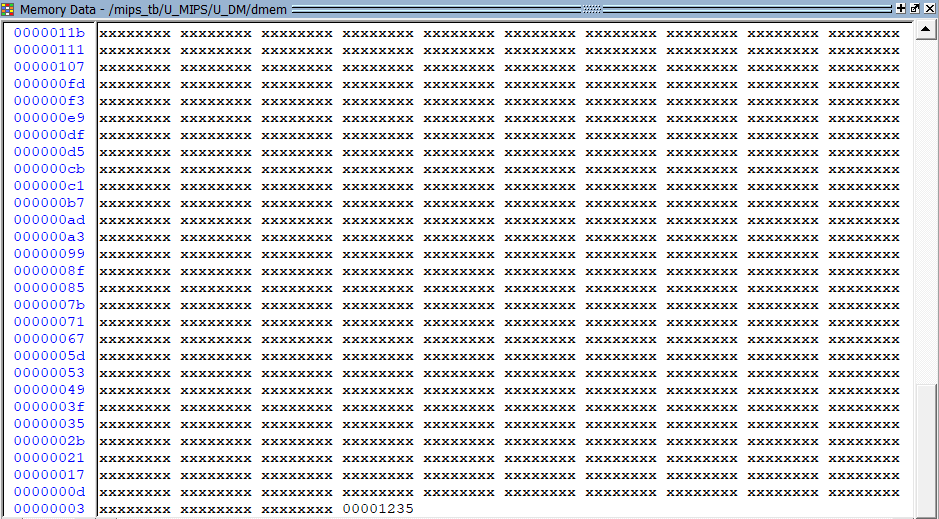
即**sw $0 $2 0000000000000000**

周期1 IF：取指令，IR值变为0xac020000

周期2 ID：指令译码，读出$0的值为0000\_0001，保存至FLOP\_A，读出$2的值为0000\_1235，保存至FLOP\_B

周期3 EX：执行，将FLOP\_A与符号扩展后的偏移量相加得到访存地址0000\_0001，存入FLOP\_ALU

周期4 MEM：访存，根据FLOP\_ALU，将FLOP\_B的值0000\_1235写入地址0000\_0001，PC变为0x0000\_3018



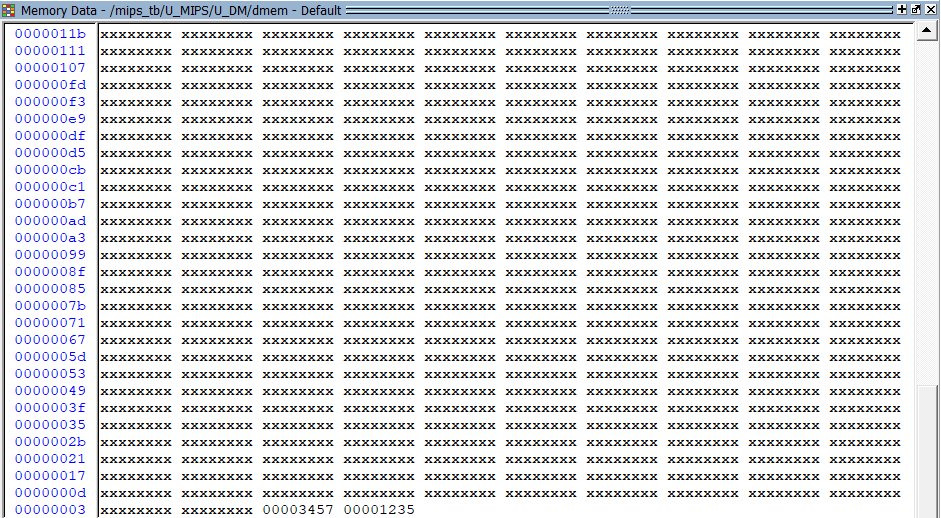
注：dmem没有在一开始的时候初始化，所以为x。由于地址是0000\_0001而指令sw是按字来访问的（地址没有凑好），所以发生截断地址变为0，这里可以认为是正确的。

* 1. ***0xac030004***

即101011 00000 00011 0000000000000100

即**sw $0 $3 0000000000000100**

过程与6类似，根据$0和偏移量计算出地址0000\_0005后，将$3的值0000\_3457写入，PC变为0x0000\_301c



* 1. ***0x8c050000***

即100011 00000 00101 0000000000000000

即**lw $0 $5 0000000000000000**

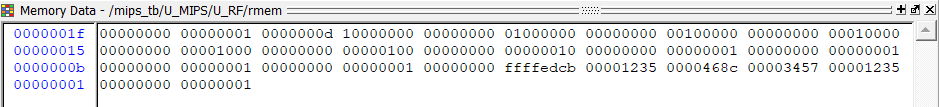
周期1 IF：取指令，IR值变为0x8c050000

周期2 ID：指令译码，读出$0的值为0000\_0001，保存至FLOP\_A

周期3 EX：执行，将FLOP\_A与符号扩展后的偏移量相加得到访存地址0000\_0001，并存入FLOP\_ALU

周期4 MEM：访存，根据FLOP\_ALU取出地址0000\_0001处的数据0000\_1235存入FLOP\_DATA

周期5 WB：写回，根据控制信号RegWrite将取出的数据0000\_1235写入寄存器$5，PC变为0x0000\_3024



* 1. ***0x10450001***

即000100 00010 00101 0000000000000001

即beq $2 $5 0000000000000001

周期1 IF：取指令，IR值变为0x10450001

周期2 ID：指令译码，读出$2的值为0000\_1235，存入FLOP\_A，读出$5的值为0000\_1235，存入FLOP\_B

周期3 EX：执行，将FLOP\_A的值减去FLOP\_B的值，得到0，ALU的zero标志位变为1。同时，EXT模块将偏移量进行符号扩展，得到0000\_0001。根据zero标志位，PC的值变为PC+4+0000\_0004，即0x0000\_302c（分支发生，跳过了地址0x0000\_3028）

* 1. ***0x0c000c0e***

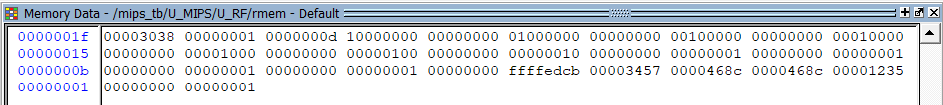
即000011 00000000000000110000001110

即jal 00000000000000110000001110

周期1 IF：取指令，IR值变为0x0c000c0e

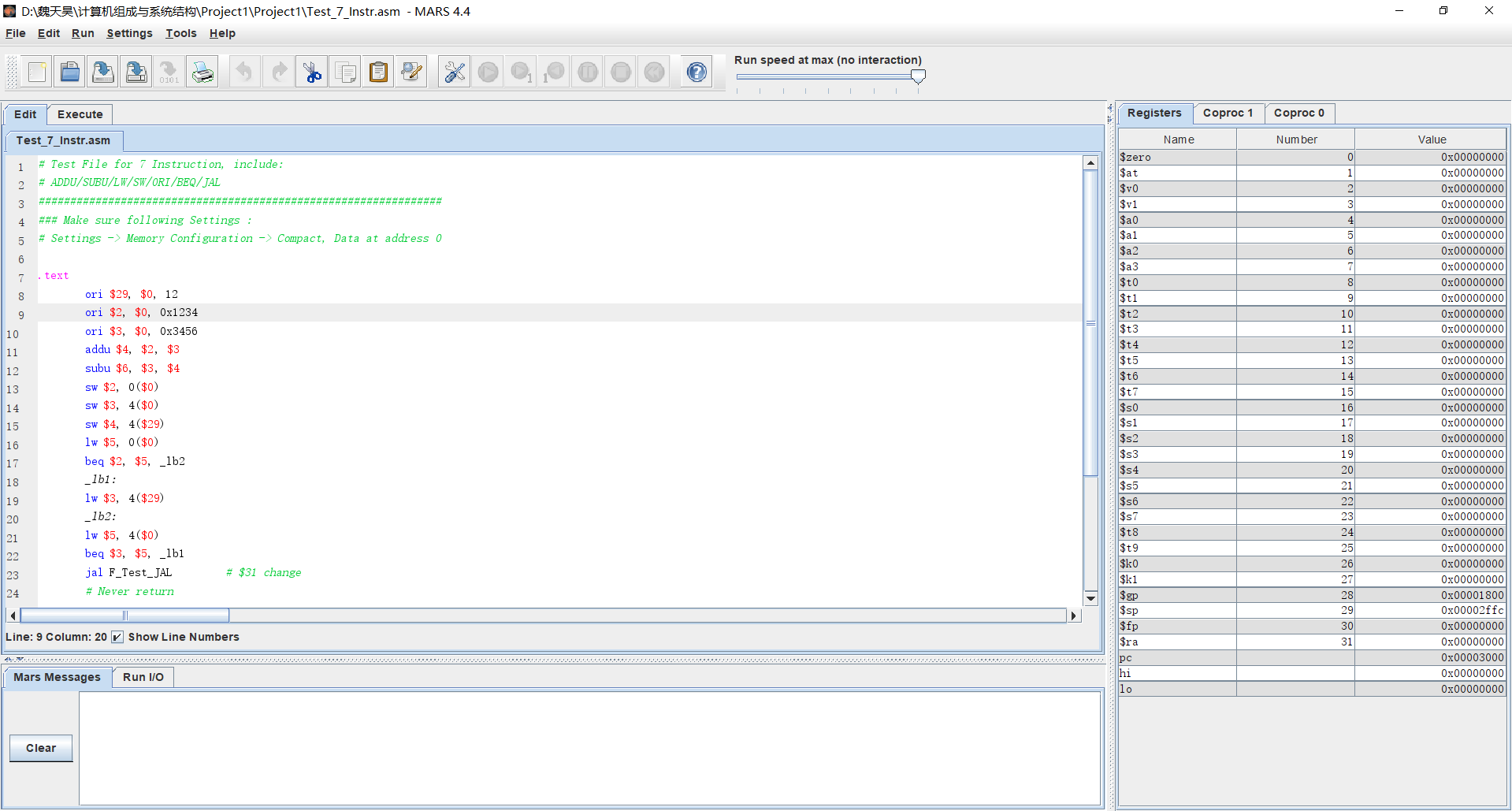
周期2 ID：指令译码

周期3 EX：执行，通过PC值0x0000\_3034和指令的26位立即数得到NPC，即0x0000\_3038，因此PC的值变为0x0000\_3038。同时，将原来PC+4的值( 0x0000\_3038 )存入寄存器$31，作为返回地址。

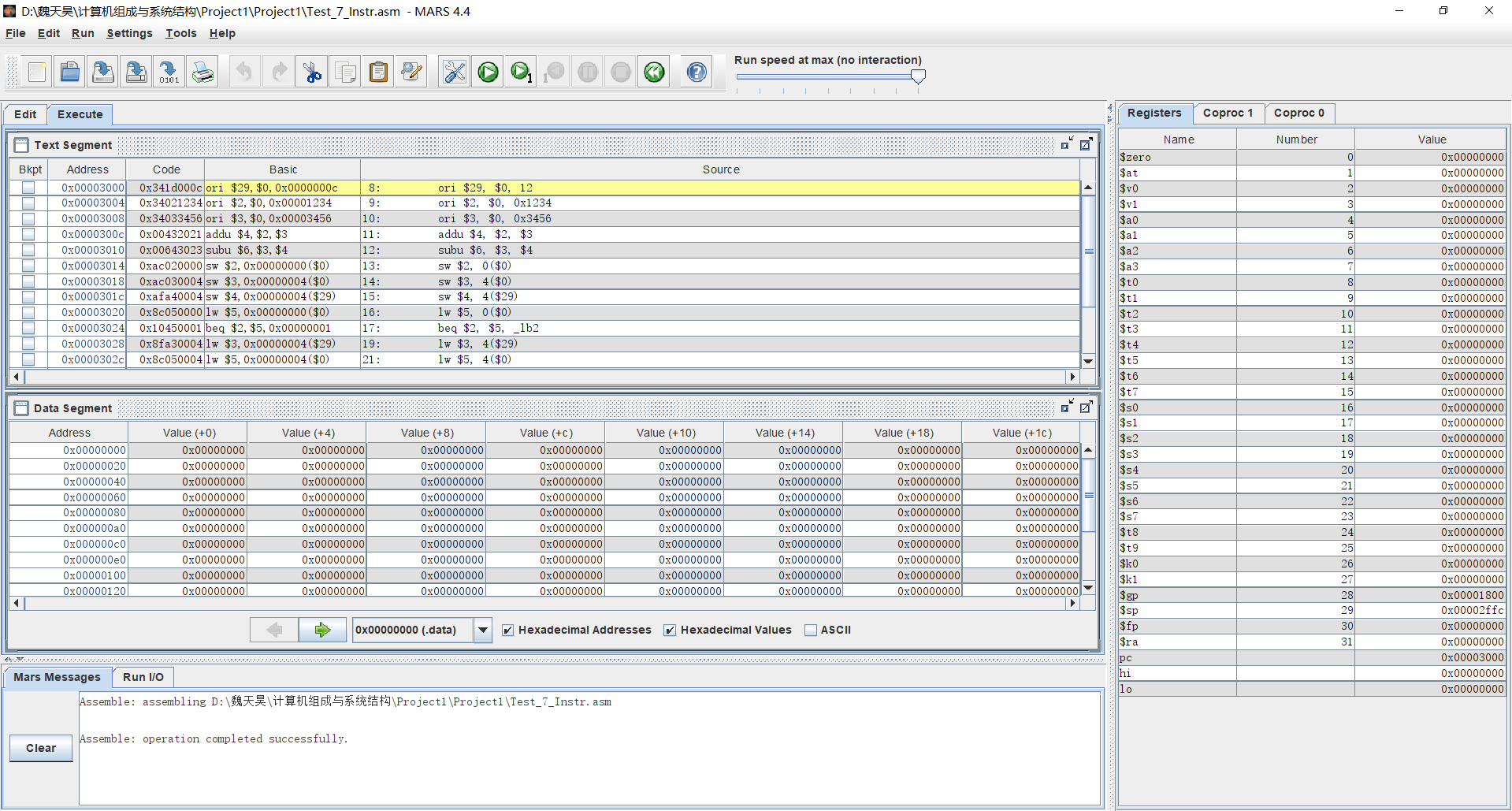


***总结：*** 上述分析10条指令包含了所需的7种指令，通过波形图与寄存器的验证，全部正确，严格按照周期执行，取得了很好的效果。

**4.Mars4\_4的使用**

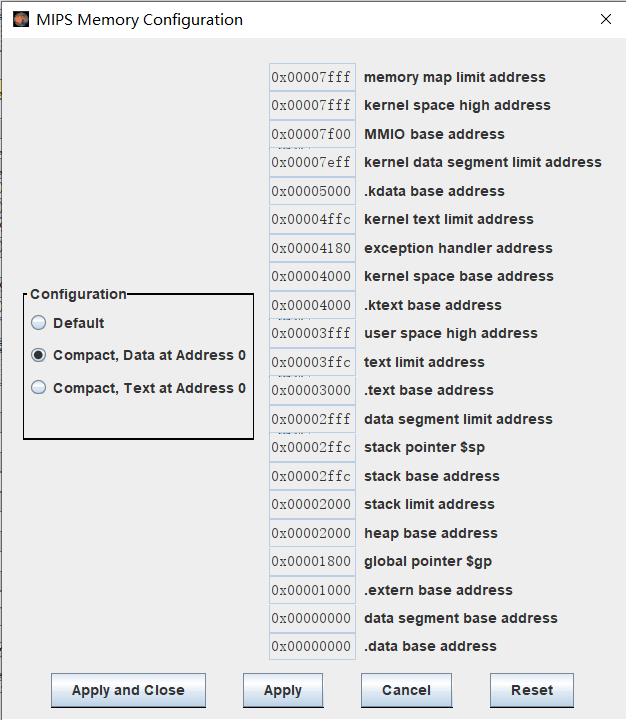
****

打开汇编文件后，点击工具栏的执行。



可以看到每条指令对应的地址，机器码，汇编和操作。在右侧可以看到寄存器的值。Mars提供了执行，单步执行，重新执行，复位等功能。

注意：在运行指令前，需要保证Settings->Memory Configuration与处理器保持一致。



**四．疑难分析**

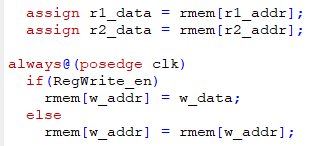
**1.地址问题：**

PC存放指令地址，根据MIPS指令均为32位的特点，指令地址为4字节对齐，因此传输时可以只传输[31：2]，后两位一定是0。IM与DM均为4K大小，即1024字，所以寻址只需[11：2]即可。

**2.寄存器堆的读写：**

寄存器堆既可以读也可以写，但是，两者并不相同。读并不需要时钟上升沿，写需要时钟上升沿。若也用时钟沿控制读，将会多综合出一个寄存器，也就多出了一个时钟周期。

代码写法：



**3.Jal指令的实现：**

Jal指令除了跳转外，还将PC+4存入寄存器31，常常与jr指令配合，实现函数的功能。所以在写入寄存器堆时，需要3选1多路选择器进行选择。注意这里的写回寄存器堆并不需要单独的时钟周期来完成。

**4.二级译码：**

在生成ALU的控制信号时，采用了二级译码的方式。由主控制单元生成ALUOp作为ALU控制单元的输入，再由ALU控制单元生成真正控制ALU的信号。这种方式有利于减小主控制单元的规模，提高控制单元的速度。

**5.周期正确：**

在判断处理器能否正确运行时，周期正确是很重要的。我们并不仅仅要求其能获得正确的输入输出，更要对其每一个周期进行分析验证。防止出现综合出多余的触发器等问题。