

Project1 MIPS 多周期处理器

2019302733 魏天昊

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

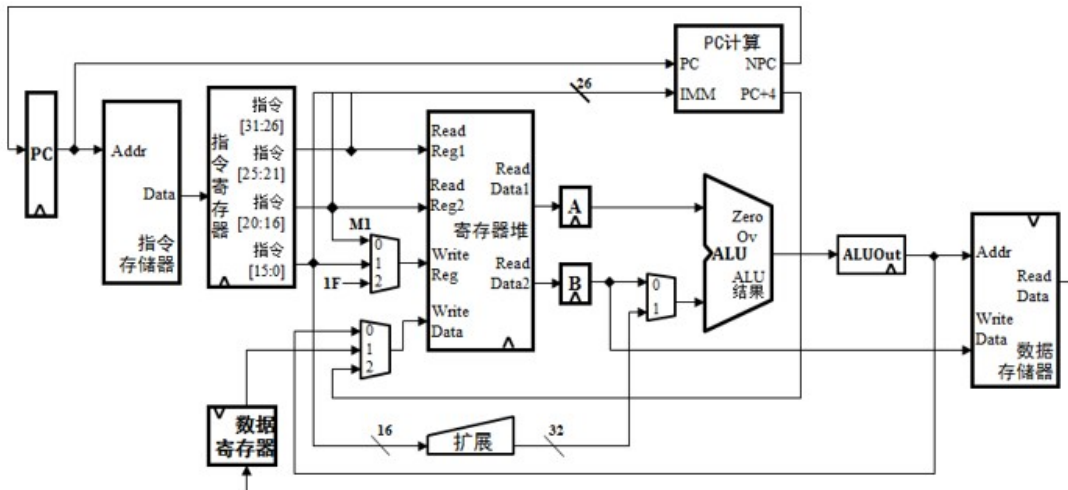
ID：指令译码的同时读取寄存器

EX：执行操作或计算地址

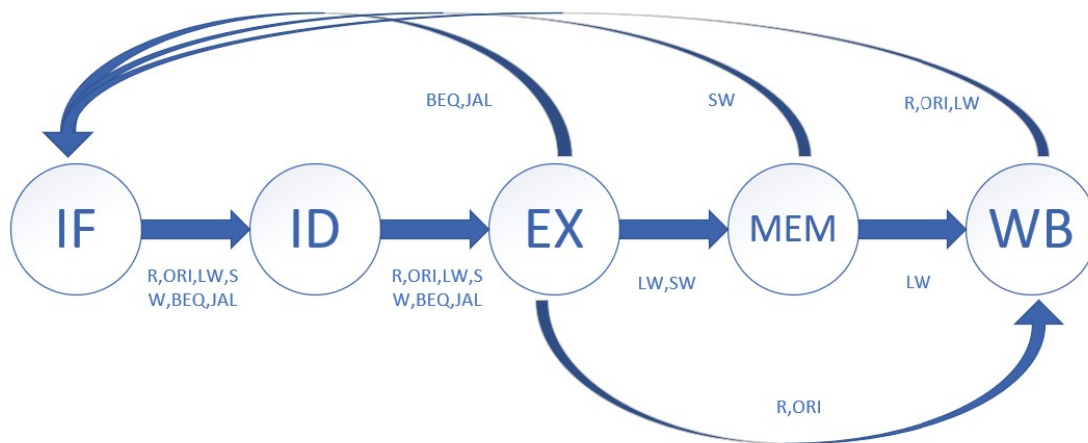
MEM：从数据存储器中读取操作数

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 RFDData1, RFDData2, 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	0	RF 写使能
Zero_sign	0	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]	0	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	0	Alu 数据来源选择 0 表示寄存器 1 表示扩展后立即数
MemWrite_en	0	Mem 写使能
NPCSrc[1:0]	0	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

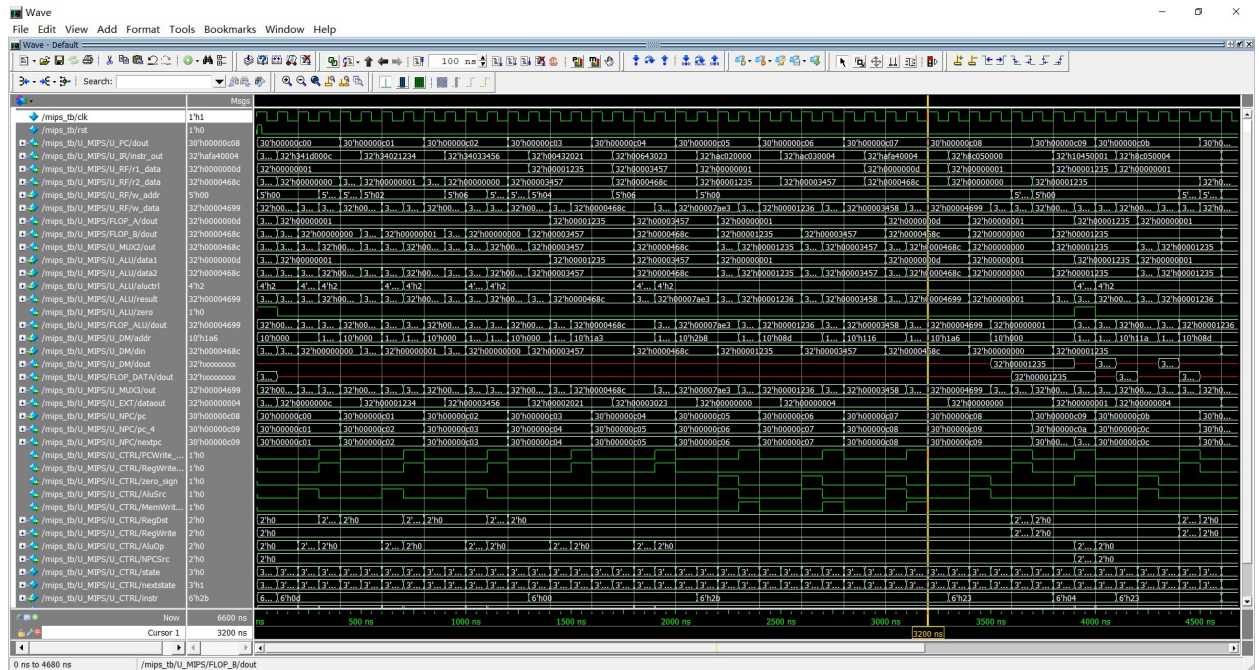
复位后 PC 位于 0x0000_3000

341d000c	00000001
34021234	00000000
34033456	00000001
00432021	00000000
00643023	00000001
ac020000	00000000
ac030004	00000001
afa40004	00000000
8c050000	00000001
10450001	00000010
8fa30004	00000000
8c050004	00001000
1065fffd	00010000
0c000c0e	00000000
00c23023	01000000
afa6fffc	00000000
1064ffff	10000000
	00000000
	00000001
	00000000

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

3.1 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

Memory Data - /mips_tb/U_MIPS/U_RF/rmem											
0000001f	00000000	00000001	0000000d	10000000	00000000	01000000	00000000	00100000	00000000	00010000	
00000015	00000000	00001000	00000000	00000100	00000000	00000010	00000000	00000001	00000000	00000001	
0000000b	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00000001	
00000001	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00000001	

3.2 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

Memory Data - /mips_tb/U_MIPS/U_RF/rmem - Default										
0000001f	00000000	00000001	0000000d	10000000	00000000	01000000	00000000	00100000	00000000	00010000
00000015	00000000	00001000	00000000	00000100	00000000	00000010	00000000	00000001	00000000	00000001
0000000b	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00001235
00000001	00000000	00000001								

3.3 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

Memory Data - /mips_tb/U_MIPS/U_RF/rmem										
0000001f	00000000	00000001	0000000d	10000000	00000000	01000000	00000000	00100000	00000000	00010000
00000015	00000000	00001000	00000000	00000100	00000000	00000010	00000000	00000001	00000000	00000001
0000000b	00000000	00000001	00000000	00000001	00000000	00000001	00000000	00000001	00003457	00001235
00000001	00000000	00000001								

3.4 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

Memory Data - /mips_tb/U_MIPS/U_RF/rmem - Default										
0000001f	00000000	00000001	0000000d	10000000	00000000	01000000	00000000	00100000	00000000	00010000
00000015	00000000	00001000	00000000	00000100	00000000	00000010	00000000	00000001	00000000	00000001
0000000b	00000000	00000001	00000000	00000001	00000000	00000001	00000000	0000468c	00003457	00001235
00000001	00000000	00000001								

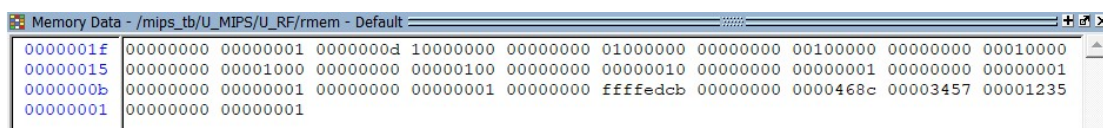
3.5 0x00643023

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



The image shows a 'Memory Data' window from a MIPS simulator. It displays the state of registers \$0 through \$31. The title bar indicates the path '/mips_tb/U_MIPS/U_RF/rmem - Default'. The registers are listed in a table with their hexadecimal values. Notably, register \$6 contains the value ffffedcb, which is the result of the subtraction performed in the previous step. The PC register is not explicitly shown but is mentioned in the text as 0x0000_3014.

Register	Value
\$0	00000000
\$1	00000001
\$2	0000000d
\$3	10000000
\$4	00000000
\$5	01000000
\$6	ffffedcb
\$7	00000000
\$8	00100000
\$9	00000000
\$10	00000000
\$11	00000001
\$12	00000000
\$13	00000000
\$14	00000000
\$15	00000000
\$16	00000000
\$17	00000000
\$18	00000000
\$19	00000000
\$20	00000000
\$21	00000000
\$22	00000000
\$23	00000000
\$24	00000000
\$25	00000000
\$26	00000000
\$27	00000000
\$28	00000000
\$29	00000000
\$30	00000000
\$31	00000000

3.6 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

Memory Data - /mips_tb/U_MIPS/U_DM/dmem										
0000011b	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000111	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000107	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000fd	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000f3	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000e9	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000df	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000d5	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000cb	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000c1	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000b7	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000ad	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000a3	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000099	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000008f	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000085	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000007b	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000071	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000067	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000005d	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000053	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000049	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000003f	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000035	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000002b	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000021	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000017	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000000d	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000003	xxxxxxxx	xxxxxxxx	xxxxxxxx	00001235						

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。

3.7 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

Memory Data - /mips_tb/U_MIPS/U_DM/dmem - Default										
0000011b	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000111	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000107	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000fd	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000f3	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000e9	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000df	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000d5	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000cb	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000c1	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000b7	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000ad	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
000000a3	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000099	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000008f	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000085	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000007b	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000071	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000067	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000005d	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000053	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000049	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000003f	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000035	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000002b	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000021	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000017	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0000000d	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
00000003	xxxxxxxx	xxxxxxxx	00003457	00001235						

3.8 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

Memory Data - /mips_tb/U_MIPS/U_RF/rmem										
0000001f	00000000	00000001	0000000d	10000000	00000000	01000000	00000000	00100000	00000000	00010000
00000015	00000000	00001000	00000000	00000100	00000000	00000010	00000000	00000001	00000000	00000001
0000000b	00000000	00000001	00000000	00000001	00000000	ffffedcb	00001235	0000468c	00003457	00001235
00000001	00000000	00000001								

3.9 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)

3.10 0x0c000c0e

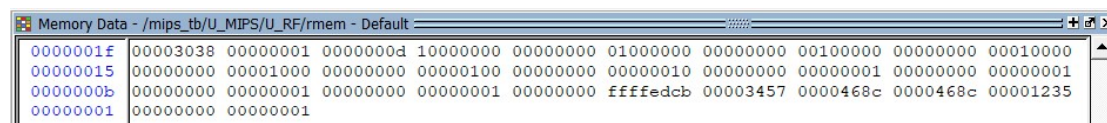
即 000011 0000000000000001100000011110

即 jal 0000000000000001100000011110

周期 1 IF: 取指令, IR 值变为 0x0c000c0e

周期 2 ID: 指令译码

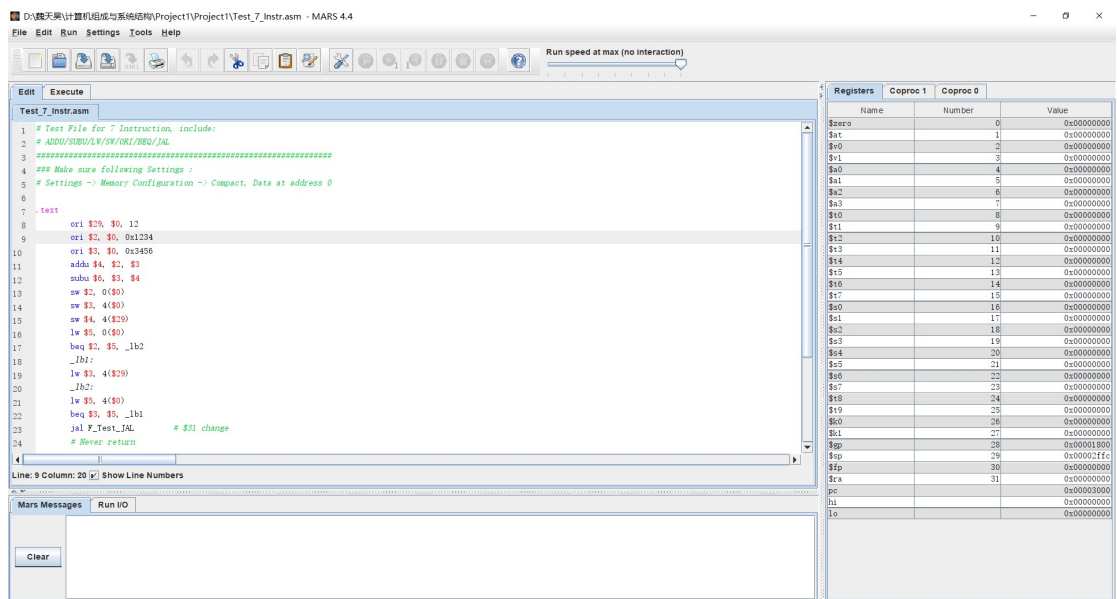
周期 3 EX: 执行, 通过 PC 值 0x0000_3034 和指令的 26 位立即数得到 NPC, 即 0x0000_3038, 因此 PC 的值变为 0x0000_3038。同时, 将原来 PC+4 的值 (0x0000_3038) 存入寄存器\$31, 作为返回地址。



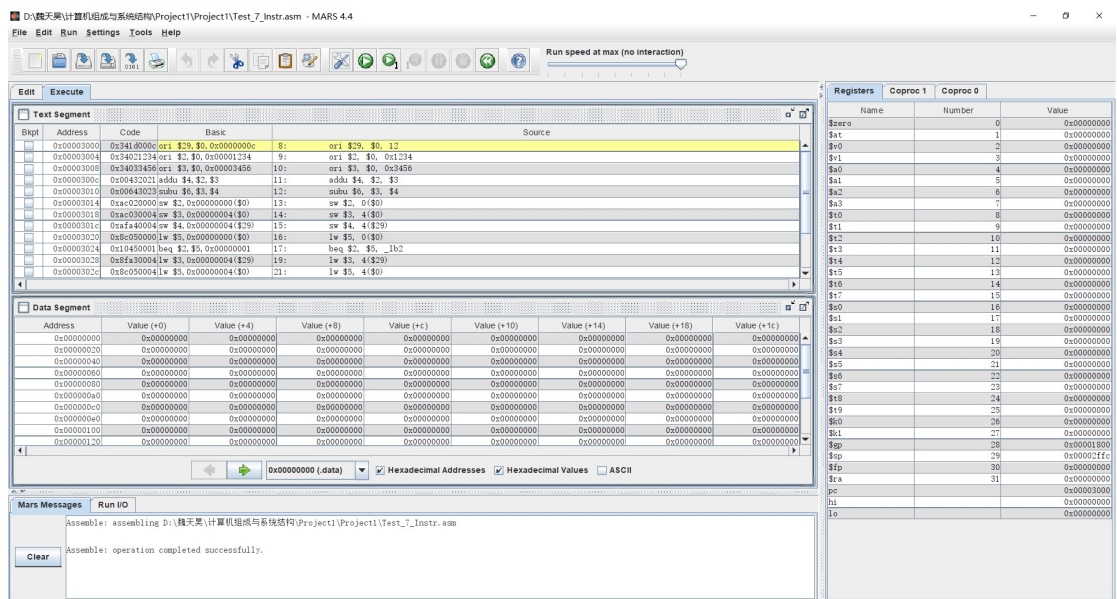
0000001f	00003038	00000001	0000000d	10000000	00000000	01000000	00000000	00100000	00000000	00010000
00000015	00000000	00001000	00000000	00000100	00000000	00000010	00000000	00000001	00000000	00000001
0000000b	00000000	00000001	00000000	00000001	00000000	ffffedcb	00003457	0000468c	0000468c	00001235
00000001	00000000	00000001								

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。

MIPS Memory Configuration

×

Configuration

☐ Default
 ☒ Compact, Data at Address 0
 ☐ Compact, Text at Address 0

0x00007fff	memory map limit address
0x00007fff	kernel space high address
0x00007f00	MMIO base address
0x00007eff	kernel data segment limit address
0x00005000	.kdata base address
0x00004ffc	kernel text limit address
0x00004180	exception handler address
0x00004000	kernel space base address
0x00004000	.ktext base address
0x00003fff	user space high address
0x00003ffc	text limit address
0x00003000	.text base address
0x00002fff	data segment limit address
0x00002ffc	stack pointer \$sp
0x00002ffc	stack base address
0x00002000	stack limit address
0x00002000	heap base address
0x00001800	global pointer \$gp
0x00001000	.extern base address
0x00000000	data segment base address
0x00000000	.data base address

Apply and Close

Apply

Cancel

Reset

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：

```
assign r1_data = rmem[r1_addr];
assign r2_data = rmem[r2_addr];

always@ (posedge clk)
    if (RegWrite_en)
        rmem[w_addr] = w_data;
    else
        rmem[w_addr] = rmem[w_addr];
```

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。