

# 北京邮电大学

## 实验报告



题目： MIPS 指令系统和 MIPS 体系结构

班 级： 2020211310

学 号： 2020211616

姓 名： 付容天

学 院： 计算机学院（国家示范性软件学院）

2023 年 4 月 18 日

# 一、实验目的

- (1) 了解和熟悉指令级模拟器。
- (2) 熟练掌握 MIPSsim 模拟器的操作和使用方法。
- (3) 熟悉 MIPS 指令系统及其特点，加深对 MIPS 指令操作语义的理解。
- (4) 熟悉 MIPS 体系结构。

# 二、实验平台

实验平台采用指令级和流水线操作级模拟器 MIPSsim。

# 三、实验内容

首先要阅读附录中的 MIPSsim 模拟器的使用方法，然后了解 MIPSsim 的指令系统和汇编语言。之后在 MIPSsim 模拟器中载入样例程序 alltest.s，在模拟器中进行非流水执行，查看并分析每条指令执行后相关寄存器的值。

# 四、实验步骤及实验分析

在实验中，我首先启动了 MIPSsim，然后配置为非流水方式，并载入了样例程序 alltest.s，然后分步进行一条指令、多条指令、连续执行、设置断点等方式的运行。载入样例程序之后，可以看到：

```
PC = 0x00000000
LO = 0x0000000000000000
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 1：初始 PC 寄存器的值

即 PC 寄存器的值为 [PC]=0x00000000，这是因为程序中第一条指令的地址为 0x00000000，即为 PC 寄存器的值。

单步执行一条指令后，PC 中显示下一条指令的地址，即为：[PC]=0x00000004：

```
PC = 0x00000004
LO = 0x0000000000000000
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 2：特殊寄存器 PC 的值

地址	断点标记	机器码	流水段	符号指令
main		0x2408007C		ADDIU \$r8,\$r0,124
0x00000004		0x81010000		LB \$r1,0(\$r8)

图 3：PC 指向地址的指令

该地址所指向的指令为一条有符号载入字节指令。

继续单步执行，得到：

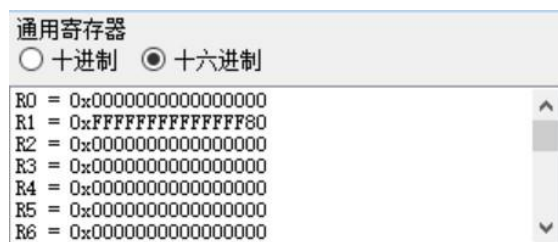


图 4：通用寄存器的值

地址	断点标记	机器码	流水段	符号指令
main		0x2408007C		ADDIU \$r8,\$r0,124
0x00000004		0x81010000		LB \$r1,0(\$r8)
0x00000008		0x8D010000		LW \$r1,0(\$r8)

图 5：下一条指令内容

其中寄存器 R1 的值为 [R1]=0xFFFFFFFFFFFF80，这是-128 的十六进制表示（补码形式）。下一条指令的地址为 [PC]=0x00000008，是一条有符号载入字指令。

单步执行一条指令，得到：

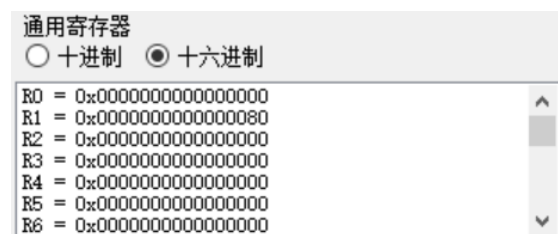


图 6：通用寄存器的值

地址	断点标记	机器码	流水段	符号指令
main		0x2408007C		ADDIU \$r8,\$r0,124
0x00000004		0x81010000		LB \$r1,0(\$r8)
0x00000008		0x8D010000		LW \$r1,0(\$r8)
0x0000000C		0x91010000		LBU \$r1,0(\$r8)

图 7：下一条指令内容

其中寄存器 R1 的值为 [R1]=0x0000000000000080。下一条指令的地址为 [PC]=0x0000000C，是一条无符号载入字节指令。

单步执行一条指令，得到寄存器 R1 的值为 [R1]=0x0000000000000080（图略）。

单步执行一条指令，此时 PC 值为 [PC]=0x00000014，位于该地址的指令是一条保存字指令：

0x0000000C	0x91010000	LBU \$r1,0(\$r8)
0x00000010	0x24080080	ADDIU \$r8,\$r0,128
0x00000014	0xAD010000	SW \$r1,0(\$r8)
0x00000018	0x10000001	BEQ \$r0,\$r0,BROG2

图 8：位于 0x00000014 处的指令

单步执行一条指令，查看内存中 BUFFER 处的字的值，发现值为 [BUFFER] = 0x00000080：

符号表

main	=	0x00000000
PROG2	=	0x00000020
PROG3	=	0x00000030
PROG4	=	0x00000040
LABEL1	=	0x0000004C
LABEL2	=	0x00000058
LABEL3	=	0x00000064
LABEL4	=	0x00000074
DATA	=	0x0000007C
BUFFER	=	0x00000080

图 9: 内存中 BUFFER 的值

接下来执行算术运算类指令，首先进行：

- (1) 双击“寄存器”窗口中的 R1，将其值修改为 2；
- (2) 双击“寄存器”窗口中的 R2，将其值修改为 3；
- (3) 单步执行指令；

得到如下结果：

```
PC = 0x00000020
LO = 0x0000000000000000
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 10: 特殊寄存器 PC 的值

0x00000018	0x10000001	BEQ \$r0,\$r0,PROG2
0x0000001C	0x00000000	SLL \$r0,\$r0,0
PROG2	0x0022182C	DADD \$r3,\$r1,\$r2

图 11: 下一条指令的内容

下一条指令的地址为 [PC]=0x00000020，是一条加法指令。

单步执行一条指令，得到：

☐ 十进制 ☒ 十六进制

```
R0 = 0x0000000000000000
R1 = 0x0000000000000002
R2 = 0x0000000000000003
R3 = 0x0000000000000005
R4 = 0x0000000000000000
R5 = 0x0000000000000000
R6 = 0x0000000000000000
```

图 12: 通用寄存器的值

可以发现寄存器 R3 的值为 [R3]=0x0000000000000005（即寄存器 R1 和 R2 的加法结果）。下一条指令地址为 [PC]=0x00000024，是一条乘法指令。

单步执行一条指令，得到：

```
PC = 0x00000028
LO = 0x0000000000000006
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 13: 查看 LO 和 HI 寄存器的值

可知 [LO]=0x0000000000000006，[HI]=0x0000000000000000。

接下来执行逻辑运算类指令，首先进行：

- (1) 双击“寄存器”窗口中的 R1，将其值修改为 0xFFFF0000；

- (2) 双击“寄存器”窗口中的 R2，将其值修改为 0xFF00FF00；
- (3) 单步执行一条指令。

得到如下结果：

地址	断点标记	机器码	流水线	符号指令
main		0x2408007C		ADDIU \$r8,\$r0,124
0x00000004		0x81010000		LB \$r1,0(\$r8)
0x00000008		0x8D010000		LW \$r1,0(\$r8)
0x0000000C		0x91010000		LBU \$r1,0(\$r8)
0x00000010		0x24080080		ADDIU \$r8,\$r0,128
0x00000014		0xAD010000		SW \$r1,0(\$r8)
0x00000018		0x10000001		BEQ \$r0,\$r0,PROG2
0x0000001C		0x00000000		SLL \$r0,\$r0,0
PROG2		0x0022182C		DADD \$r3,\$r1,\$r2
0x00000024		0x0022001C		DMULT \$r1,\$r2
0x00000028		0x10000001		BEQ \$r0,\$r0,PROG3
0x0000002C		0x00000000		SLL \$r0,\$r0,0
PROG3		0x00221824		AND \$r3,\$r1,\$r2
0x00000034		0x30230000		ANDI \$r3,\$r1,0
0x00000038		0x10000001		BEQ \$r0,\$r0,PROG4

图 14：当前执行状态

可以得到下一条指令地址为 [PC]=0x00000030，是一条逻辑与运算指令，第二个操作数寻址方式是 寄存器直接寻址。

单步执行一条指令，得到：

地址	断点标记	机器码	流水线	符号指令
main		0x2408007C		ADDIU \$r8,\$r0,124
0x00000004		0x81010000		LB \$r1,0(\$r8)
0x00000008		0x8D010000		LW \$r1,0(\$r8)
0x0000000C		0x91010000		LBU \$r1,0(\$r8)
0x00000010		0x24080080		ADDIU \$r8,\$r0,128
0x00000014		0xAD010000		SW \$r1,0(\$r8)
0x00000018		0x10000001		BEQ \$r0,\$r0,PROG2
0x0000001C		0x00000000		SLL \$r0,\$r0,0
PROG2		0x0022182C		DADD \$r3,\$r1,\$r2
0x00000024		0x0022001C		DMULT \$r1,\$r2
0x00000028		0x10000001		BEQ \$r0,\$r0,PROG3
0x0000002C		0x00000000		SLL \$r0,\$r0,0
PROG3		0x00221824		AND \$r3,\$r1,\$r2
0x00000034		0x30230000		ANDI \$r3,\$r1,0
0x00000038		0x10000001		BEQ \$r0,\$r0,PROG4

图 15：当前执行状态

可以得到寄存器 R3 的值为 [R3]=0x00000000FF000000，下一条指令地址为 [PC]=0x00000034，是一条逻辑与运算指令，第二个操作数寻址方式为 寄存器直接寻址。

单步执行一条指令，得到：

通用寄存器	
<input type="radio"/> 十进制	<input checked="" type="radio"/> 十六进制
R0	= 0x0000000000000000
R1	= 0x00000000FFFFFF00
R2	= 0x00000000FF00FF00
R3	= 0x0000000000000000
R4	= 0x0000000000000000
R5	= 0x0000000000000000
R6	= 0x0000000000000000

图 16：通用寄存器的值

其中寄存器 R3 的值为 [R3]=0x0000000000000000，这是因为执行的是逻辑与运算指令，R1 的值与立即数 0 做与运算后，得到的结果是 0，存入 R3 寄存器中。

接下来执行控制转移类指令，首先进行：

- (1) 双击“寄存器”窗口中的 R1，将其值修改为 2；
- (2) 双击“寄存器”窗口中的 R2，将其值修改为 2；
- (3) 单步执行一条指令。

得到如下结果：

```
PC = 0x00000040
LO = 0x0000000000000006
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 17: 特殊寄存器 PC 的值

0x0000003C	0x00000000	SLL \$r0,\$r0,0
PROG4	0x10220002	BEQ \$r1,\$r2,2
0x00000044	0x00000000	SLL \$r0,\$r0,0
0x00000048	0x00000000	SLL \$r0,\$r0,0
LABEL1	0x04210002	BGEZ \$r1,2

图 18: 下一条指令的内容

可以看到下一条指令的地址为 [PC]=0x00000040，是一条 BEQ 指令，其测试条件为 r1=r2，目标地址为 0x0000004C。

单步执行一条指令，得到：

```
PC = 0x0000004C
LO = 0x0000000000000006
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 19: 特殊寄存器 PC 的值

可以发现此时的 PC 值为 [PC]=0x0000004C，表明分支成功，这是因为我们之前将寄存器 R1 和 R2 的值都设为了 2，即相等，所以会执行跳转指令。并且，下一条指令是一条 BGEZ 指令，其测试条件为 r1>=0，目标地址为 0x00000058。

单步执行一条指令，得到：

```
PC = 0x00000058
LO = 0x0000000000000006
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 20: 特殊寄存器 PC 的值

LABEL1	0x04210002	BGEZ \$r1,2
0x00000050	0x00000000	SLL \$r0,\$r0,0
0x00000054	0x00000000	SLL \$r0,\$r0,0
LABEL2	0x04310002	BGEZAL \$r1,2

图 21: 下一条指令的内容

此时 PC 的值 [PC]=0x00000058，表明分支成功，这是因为寄存器 R1 的值为 2 大于 0，所以执行跳转。并且，下一条指令为 BGEZAL 指令，其测试条件为 r1>=0，目标地址为 0x00000064。

单步执行一条指令，得到：

```
PC = 0x00000064
LO = 0x0000000000000006
HI = 0x0000000000000000
FCSR = 0x00000000
```

图 22: 特殊寄存器 PC 的值

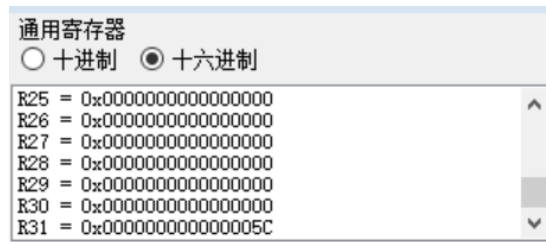


图 23: 通用寄存器的值

此时 PC 的值 [PC]=0x00000064，表明分支成功，然后查看通用寄存器 R31 的值为 [R31]=0x000000000000005C，这是因为 BGEZAL 指令将转移指令后面指令的地址作为返回地址保存到 R31 中。

单步执行一条指令，得到：

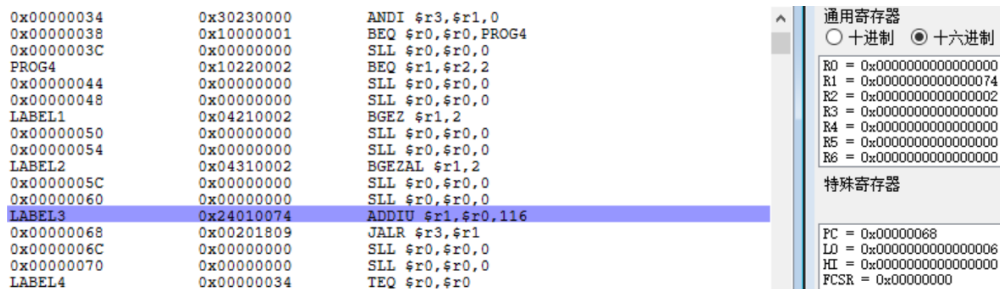


图 24: 此时执行状态

此时寄存器 R1 的值为 [R1]=0x0000000000000074，下一条指令的地址为 [PC]=0x00000068，是一条 JALR 指令，保存目标地址的寄存器为 R1，保存返回地址的寄存器为 R3。

单步执行一条指令，得到：



图 25: 通用寄存器和特殊寄存器的值

可以得到 [PC]=0x00000074，[R3]=0x000000000000006C。

## 五、实验结果分析与总结

在本次实验中出现的部分指令及其作用可以用下表概括：

指令及其内容	作用
ADDIU \$R8, \$R0, 124	R8=R0+124
LB \$R1, 0(\$R8)	从[R8+0]位置处读取字节到 R1
LW \$R1, 0(\$R8)	从[R8+0]位置处读取字到 R1
LBU \$R1, 0(\$R8)	从[R8+0]处读取无符号字节到 R1
ADDIU \$R8, \$R0, 128	R8=R0+128
SW \$R1, 0(\$R8)	将 R0 数据存到[R8+0]位置处
BEQ \$R0, \$R0, PROG2	条件转移指令（到 PROG2 处）
DADD \$R3, \$R1, \$R2	R3=R1+R2
DMULT \$R1, \$R2	两定点寄存器相乘
BEQ \$R0, \$R0, PROG3	条件转移指令（到 PROG3 处）
AND \$R3, \$R1, \$R2	R3=R1 与 R2
ANDI \$R3, \$R1, 0	R3=R1 与 0
BEQ \$R0, \$R0, PROG4	条件转移指令（到 PROG4）处
BGEZ \$R1, 2	条件转移指令（偏移量 2）
JALR \$R3, \$R1	使用寄存器的跳转指令

在本次实验中，我学习了 MIPSsim 模拟器的基本使用方法，圆满完成了实验一既定的实验任务。我熟悉了 MIPS 指令系统，并加深了对执行中 MIPS 程序行为的理解。我对实验一中程序的执行过程分析得很细致，弄清楚了每一处细节，收获颇丰！