

数字逻辑与处理器基础

MIPS汇编编程实验

实验指导书

2023年 春季学期

陈佳煜 杜禧瑞

目录

- MARS环境安装与基础使用方法
- 实验内容一：基础练习
 - 系统调用
 - 循环分支
 - 数组指针
 - 函数调用
- 参考资料

安装JRE

- 运行JAVA程序包需要运行环境：Java Runtime Environment (JRE)
- Windows系统运行 JavaSetup8u241.exe按提示进行安装。
- 如果安装中又遇到问题，或者其他操作系统可以访问JAVA官网：https://www.java.com/zh_CN/，下载完成后按提示进行安装。

报告问题

访问包含 Java 应用程序的
页时为什么始终重定向到此
页？

» [了解详细信息](#)

⚠ Oracle Java 许可重要更新

从 2019 年 4 月 16 起的发行版更改了 Oracle Java 许可。

新的适用于 Oracle Java SE 的 Oracle 技术网许可协议 与以前的 Oracle Java 许可有很大差异。新许可允许某些免费使用（例如个人使用和开发使用），而根据以前的 Oracle Java 许可获得授权的其他使用可能会不再支持。请在下载和使用此产品之前认真阅读条款。可在此处查看常见问题解答。

可以通过低成本的 [Java SE 订阅](#) 获得商业许可和技术支持。

Oracle 还在 [jdk.java.net](#) 的开源 [GPL 许可](#)下提供了最新的 OpenJDK 发行版。

免费 Java 下载

» [什么是 Java?](#) » [我有 Java 吗?](#) » [是否需要帮助?](#)

负责助教联系方式

- 陈佳煜: _jiayu-ch19@mails.tsinghua.edu.cn
- 杜禧瑞: dxr22@mails.tsinghua.edu.cn

运行MARS

- 如果JRE正确安装，双击Mars4_5.jar即可打开MARS仿真器。
- 如果打不开，先检查JRE是否安装正确，可以考虑重新安装。
- 如果是软件包的问题可以进入MARS官网下载。
<https://courses.missouristate.edu/KenVollmar/mars/download.htm>
- 接下来将用example_0.asm作为例子演示MARS的用法。

运行MARS

- 运行MARS后的主要界面如图所示。
- 主要编辑区用于编写汇编指令。
- 输出信息区可以查看程序运行过程中的输出和系统报错等。
- 寄存器列表实时显示当前运行状态下各个寄存器存储的值。

D:\phdwork\助教\MIPS大作业习题课\1\example.asm - MARS 4.5

File Edit Run Settings View Help 文件读写功能 基本编辑功能 汇编执行调试功能 指令运行速度

Run speed at max (no interaction)

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
\$pc		0x00400000
\$hi		0x00000000
\$lo		0x00000000

见 example_0.asm

寄存器列表

主要编辑区

Mars Messages Run I/O

Clear

输出信息区

```
1 .data
2 in_buff: .space 512
3 out_buff: .space 512
4 input_file: .asciiz "example.in"
5 output_file: .asciiz "example.out"
6 comma: .asciiz ","
7 .text
8 la $a0, input_file #input_file 是一个字符串
9 li $a1, 0 #flag 0为读取 1为写入
10 li $a2, 0 #mode is ignored 设置为0就可以了
11 li $v0, 13 #13 为打开文件的 syscall 编号
12 syscall # 如果打开成功, 文件描述符返回到$v0中
13 move $a0,$v0 # 将文件描述符载入到 $a0中
14 la $a1, in_buff #in_buff 为数据暂存区
15 li $a2, 512 #读取512个byte
16 li $v0, 14 #14 为读取文件的 syscall 编号
17 syscall
18 li $v0 16 #16 为关闭文件的 syscall 编号
19 syscall
20
```

Line: 1 Column: 1 Show Line Numbers

汇编运行

- 首先打开汇编文件 example_0.asm
- 点击**汇编**按钮即可切换到执行页面，源代码汇编成**基础指令**和**机器码**，PC 置为 0x00400000，并等待执行。
- 执行页面内可以看到汇编后的**基础指令**和对应的**机器码**，每条指令的**指令地址**。

D:\phdwork\助教\MIPS大作业习题课\1\example.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

指令断点 **地址** **机器码** **基础指令** **源代码**

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1, 0x00001001	8: la \$a0, input_file #input_file 是...
<input type="checkbox"/>	0x00400004	0x34240400	ori \$4, \$1, 0x00000400	
<input type="checkbox"/>	0x00400008	0x24050000	ddiu \$5, \$0, 0x00000000	9: li \$a1, 0 #flag 0为读取 1为写入
<input type="checkbox"/>	0x0040000c	0x24060000	ddiu \$6, \$0, 0x00000000	10: li \$a2, 0 #mode is ignored 设置为0就...
<input type="checkbox"/>	0x00400010	0x2402000d	ddiu \$2, \$0, 0x0000000d	11: li \$v0, 13 #13 为打开文件的 syscall ...
<input type="checkbox"/>	0x00400014	0x0000000c	syscall	12: syscall # 如果打开成功, 文件描述符...
<input type="checkbox"/>	0x00400018	0x00022021	addu \$4, \$0, \$2	13: move \$a0, \$v0 # 将文件描述符载入到 \$...
<input type="checkbox"/>	0x0040001c	0x3c011001	lui \$1, 0x00001001	14: la \$a1, in_buff #in_buff 为数据暂存区
<input type="checkbox"/>	0x00400020	0x34250000	ori \$5, \$1, 0x00000000	
<input type="checkbox"/>	0x00400024	0x24060200	ddiu \$6, \$0, 0x00000200	15: li \$a2, 512 #读取512个byte

Memory查看器

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values

0x10000000 (.extern)
0x10010000 (.data)
0x10040000 (heap)
current \$gp
current \$sp
0x00400000 (.text)
0x90000000 (.kdata)
0xffff0000 (MMIO)

可以选择查看哪一段地址

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Mars Messages Run I/O

Clear Reset: reset completed.

汇编运行

- 源代码：用户编写的汇编代码，包括标记，伪代码等。
- 基础指令：汇编后的指令，伪代码被转换，标记被翻译。
- 地址&机器码：与基础指令一一对应，32bit一条指令，地址依次加四。
- 断点：调试用，当执行到这一句时暂停。

D:\phdwork\助教\MIPS大作业习题课\1\example.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	
	0x00400000	0x3c011001	lui \$1, 0x00001001
	0x00400004	0x34240400	ori \$4, \$1, 0x00000400
	0x00400008	0x24050000	addiu \$5, \$0, 0x00000000
	0x0040000c	0x24050000	addiu \$5, \$0, 0x00000000
	0x00400010	0x24050000	addiu \$5, \$0, 0x00000000
	0x00400014	0x24050000	addiu \$5, \$0, 0x00000000
	0x00400018	0x24050000	addiu \$5, \$0, 0x00000000
	0x0040001c	0x24050000	addiu \$5, \$0, 0x00000000
	0x00400020	0x24050000	addiu \$5, \$0, 0x00000000
	0x00400024	0x24050000	addiu \$5, \$0, 0x00000000

Data Segment

Address	Value (+0)	Value (-0)
0x10010000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000

Registers

Name	Number	Value
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Mars Messages Run I/O

Clear Reset: reset completed.

例子：
伪代码 `la $a0, input_file`
`input_file`被翻译为真正的地址0x10010400
`la`伪指令被转换为`lui+ori`真指令
`lui $1, 0x1001`为 {f_{hex}, 0_{hex}, 1_{hex}, 1001_{hex}} = 0x3c011001
`ori $4, $1, 0x0400`为 {d_{hex}, 1_{hex}, 4_{hex}, 0400_{hex}} = 0x34240400

汇编运行

- 执行：从第一条指令开始连续执行直到结束。
- 单步执行：执行当前指令并跳转到下一条。
- 单步后退：后退到最后一条指令执行前的状态（包括寄存器和memory）
- 暂停&停止：在连续执行的时候可以停下来，一般配合较慢的指令运行速度，不用于调试。**调试最好使用断点功能。**
- 重置：重置所有寄存器和memory。

汇编 执行 单步执行 单步后退 暂停 停止 重置

指令运行速度

黄色指令代表当前指令
是即将执行但尚未执行的指令
也是pc寄存器对应的指令

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

注：MARS中是小端存储的方式

汇编运行

- 点击执行按钮后，所有指令执行完毕。
- 可以看到各个寄存器内的值发生了变化。
- Memory中in_buffer, out_buffer地址对应的数据发生变化。
- 输出区正确打印了对应的数据并提示，程序执行完地址最大的指令并且没有后续指令了（drop off bottom）

The screenshot displays the MARS MIPS assembler simulator interface. The main window shows the assembly code in the Text Segment panel, with addresses ranging from 0x00400093 to 0x004000bc. The Data Segment panel shows memory addresses and their values, with a red box highlighting the in_buffer and out_buffer addresses. The Registers panel shows the state of MIPS registers, with a red box highlighting the \$s0, \$s1, and \$s2 registers. The Mars Messages panel shows the output of the program, including the address 384, 887, 778, 916 and the message 'program is finished running (dropped off bottom)'.

在in_buffer的位置依次存放了 0x4,0x180,0x377,0x30a,0x394

在out_buffer的位置依次存放了 0x180,0x377,0x30a,0x394

输出区打印了384,887,778,916, 并显示drop off bottom

汇编运行

38 la \$a0, comma

- 在38行的指令处设置断点。并点击运行按钮两次，程序停在该位置。
- 可以看到程序向out_buffer中写入两个数，也向输出区打了两个数，各个寄存器也停留在对应状态。

在打印某个整数和打印逗号之间的指令设置断点

在in_buffer的位置依次存放了 0x4,0x180,0x377,0x30a,0x394

在out_buffer的位置依次存放了0x180,0x377

PC取值为 0x00400070

输出区打印了 384,887

example_0.asm 内包含一个从文件读取数据并写入另一个文件的例子

数据声明，此部分数据存在0x10010000

```
.data
in_buff: .space 512
out_buff: .space 512
input_file: .asciiz "example.in"
output_file: .asciiz "example.out"
comma: .asciiz ", "
```

```
.text
la $a0, input_file #input_file 是一个字符串
li $a1, 0 #flag 0为读取 1为写入
li $a2, 0 #mode is ignored 设置为0就可以了
li $v0, 13 #13 为打开文件的 syscall 编号
syscall # 如果打开成功，文件描述符返回到
move $a0, $v0 # 将文件描述符载入到 $a0中
la $a1, in_buff #in_buff 为数据暂存区
li $a2, 512 #读取512个byte
li $v0, 14 #14 为读取文件的 syscall 编号
syscall
li $v0, 16 #16 为关闭文件的 syscall 编号
syscall
```

打开读取文件，并将数据写入in_buff

初始化变量

```
la $s0, in_buff
la $s1, out_buff
lw $s2, 0($s0)
li $t0, 0
```

循环体

```
#地址加4 循环变量减1
for: addi $s0, $s0, 4
addi $t0, $t0, 1
lw $t1, 0($s0)
sw $t1, 0($s1)
addi $s1, $s1, 4
#打印整数
move $a0, $t1
li $v0, 1
syscall
#打印逗号
la $a0, comma
li $v0, 4
syscall
bne $t0, $s2, for
```

跳转条件

打开文件并将out_buff的数据写入

```
la $a0, output_file #output_file 是一个字符串
li $a1, 1 #flag 0为读取 1为写入
li $a2, 0 #mode is ignored 设置为0就可以了
li $v0, 13 #13 为打开文件的 syscall 编号
syscall # 如果打开成功，文件描述符返回到
move $a0, $v0 # 将文件描述符载入到 $a0中
la $a1, out_buff #in_buff 为数据暂存区
sll $s2, $s2, 2
move $a2, $s2 #写入512个byte
li $v0, 15 #15 为写入文件的 syscall 编号
syscall
#此时$a0 中的文件描述符没变
#直接调用 syscall 16 关闭它
li $v0, 16 #16 为关闭文件的 syscall 编号
syscall
```

汇编基本结构

见 example_1.asm

- 数据段
 - 以 “.data” 记号开头。
 - 包括常量数据和固定数组的声明。
- 代码段
 - 以 “.text” 记号开头。
 - 包括待执行的代码和行标记。
- 注释
 - “#”为注释标记。
 - 可以出现在任意位置。
 - “#”及其之后的所有内容均被忽略。

```
.data
    string: .asciiz "Hello World!\n"

.text
main:
    la $a0 string #载入字符串地址
    li $v0 4      #4代表打印字符串
    syscall      #执行系统调用

    li $v0 17     #17代表exit
    syscall      #执行系统调用
```


汇编基本结构

`.align x`

将下一个数据项对齐到特定的byte边界。如果要读取的数据是以2byte, 4byte, 8byte为单位的, 需要对齐到对应的边界上。x取值0表示1byte, 1表示2byte, 2表示4byte, 3表示8byte。

`.ascii, .asciiz`

表示字符串, 其中asciiz会自动在最后补上null字符。

`.byte, .half, .word`

表示数组常量按1byte, 2byte, 4byte存储

`.space`

表示一个以byte计长度的数组

见 example_2.asm

`.data`

```
stringz: .asciiz "Hello World!\n"
string: .ascii "Hello World!\n"
#让array对齐到4byte边界
.align 4 #没有这句话可能会出错
array: .space 512
#以下常数数组会自动对齐到对应边界
barray: .byte 1,2,3,4
harray: .half 1,2,3,4
warray: .word 1,2,3,4
```

代码段基本结构

- 代码段一般由若干段顺序排列的指令序列构成
 - 从第一条指令开始执行
 - 按顺序执行，除非发生跳转
- label_name : add \$0, \$1, \$2
 - 字符串+冒号代表标记，可以用在对应指令同一行开头或者对应指令上一行
 - 逗号可以省略
- 一个函数一般以函数名为第一条指令的label（函数入口）。程序内一般包括入栈，程序主体，设置返回值，出栈，返回上一级程序

见 example_3.asm

主过程

```
.text
main:
    li $v0 5      #5代表读入一个整数
    syscall      #执行系统调用
    move $a0 $v0  #将读入的整数作为第一个参数
    li $v0 5      #5代表读入一个整数
    syscall      #执行系统调用
    move $a1 $v0  #将读入的整数作为第二个参数

    jal product   #跳转到子过程product
    move $a0 $v0  #将返回值赋给$a0
    li $v0 1      #1代表打印一个整数
    syscall      #执行系统调用

    li $v0 17     #17代表exit
    syscall      #执行系统调用
```

代码段基本结构

见 example_3.asm

子过程，接上页

- 代码段一般由若干段顺序排列的指令序列构成
 - 从第一条指令开始执行
 - 按顺序执行，除非发生跳转
- label_name : add \$0, \$1, \$2
 - 字符串+冒号代表标记，可以用在对应指令同一行开头或者对应指令上一行
 - 逗号可以省略
- 一个函数一般以函数名为第一条指令的label（函数入口）。程序内一般包括入栈，程序主体，设置返回值，出栈，返回上一级程序

```
product:
```

```
move $t0 $a0  #将第一个参数赋给t0作为累加值
move $t1 $a1  #将第二个参数赋给t1作为计数器
move $t2 $zero#结果清零
loop: add $t2 $t2 $t0 #结果累加t0
      addi $t1 $t1 -1 #计数器减一
      bnez $t1 loop   #如果计数器不为零循环继续
move $v0 $t2  #将结果赋给返回值
jr $ra        #跳转回上一级程序
```


如何用MIPS实现函数调用

• 主调过程A的流程为：

1. （临时寄存器和参数寄存器入栈，保护后续仍需要使用的临时变量和参数）
2. 设置参数寄存器\$a0~\$a3。
3. 使用jal跳转到被调函数B。
4. （从栈中恢复临时寄存器与参数寄存器）
5. 使用被调函数的返回值\$v0~\$v1执行接下来的程序

寄存器编号	助记符	用法
0	zero	永远为0
1	at	用做汇编器的临时变量
2-3	v0, v1	用于过程调用时返回结果
4-7	a0-a3	用于过程调用时传递参数
8-15	t0-t7	临时寄存器。在过程调用中被调用者不需要保存与恢复
24-25	t8-t9	
16-23	s0-s7	保存寄存器。在过程调用中被调用者一旦使用这些寄存器时，必须负责保存和恢复这些寄存器的原值
26,27	k0,k1	通常被中断或异常处理程序使用，用来保存一些系统参数
28	gp	全局指针。一些运行系统维护这个指针来更方便的存取static和extern变量
29	sp	堆栈指针
30	fp	帧指针
31	ra	返回地址

练习1-4：函数调用

• 被调过程B的流程为：

1. 分配栈空间（ $\$sp - 4 * n$ ）。
2. 将返回地址、保存寄存器入栈。
3. 使用输入参数 $\$a0 \sim \$a3$ 执行函数内容并将结果存入返回寄存器 $\$v0 \sim \$v1$ 。
4. 将被调过程B入栈的数据恢复到寄存器。
5. `jr $ra` 返回主调过程A。

寄存器编号	助记符	用法
0	zero	永远为0
1	at	用做汇编器的临时变量
2-3	v0, v1	用于过程调用时返回结果
4-7	a0-a3	用于过程调用时传递参数
8-15	t0-t7	临时寄存器。在过程调用中被调用者不需要保存与恢复
24-25	t8-t9	
16-23	s0-s7	保存寄存器。在过程调用中被调用者一旦使用这些寄存器时，必须负责保存和恢复这些寄存器的原值
26,27	k0,k1	通常被中断或异常处理程序使用，用来保存一些系统参数
28	gp	全局指针。一些运行系统维护这个指针来更方便的存取static和extern变量
29	sp	堆栈指针
30	fp	帧指针
31	ra	返回地址

参考资料

- MIPS32 官方网站资料
<https://www.mips.com/products/architectures/mips32-2/>
- 指令集架构简介 *Introduction to the MIPS32 Architecture.pdf*
- 指令集手册 *MIPS32 Instruction Set Manual.pdf*
- 课程教材第V章
- 指令集课件附录

附件

- JAVA环境安装包 JavaSetup8u241.exe
- MARS模拟器 Mars4_5.jar
- 二进制文件查看器 pxBinaryViewerSetup.exe
- 随机数生成