

数字逻辑与处理器基础

MIPS汇编编程实验

实验指导书

2022年 春季学期

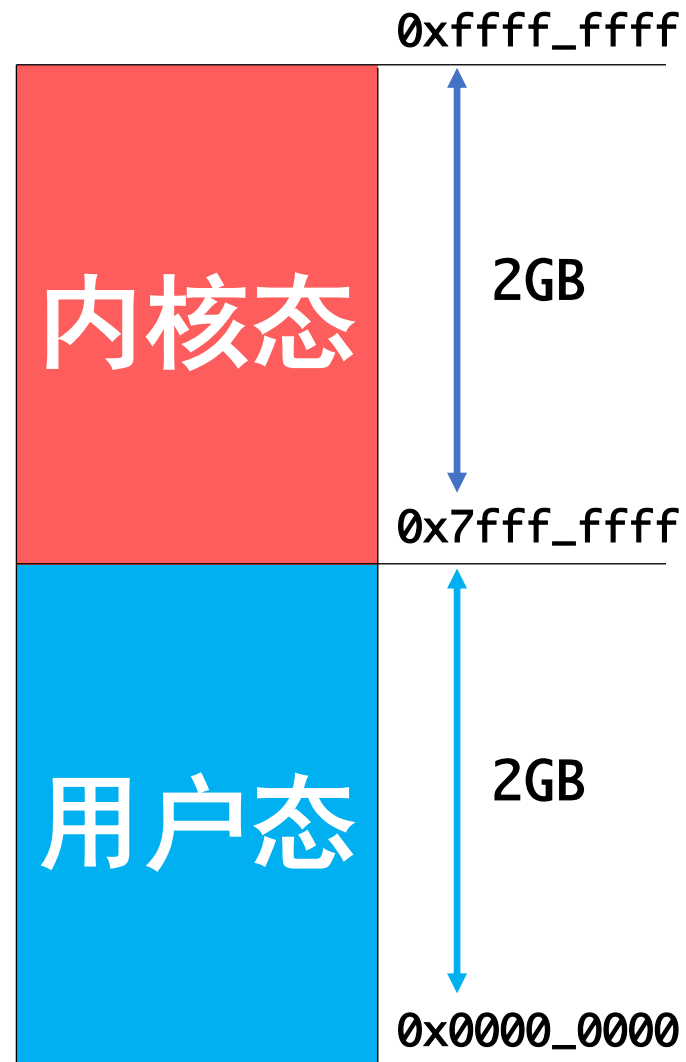
陈佳煜 黄成宇

目录

- 汇编程序设计基础
 - MIPS内存分配
 - 汇编的语法
 - 变量
 - 分支
 - 数组
- 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

32位MIPS的内存分配

- 内存空间分配
 - 32位的地址决定了能管理的内存最大为 $2^{32}=4\text{GB}$ 的大小。
 - 一般我们将低2GB规定为**用户态**空间，即一般的应用程序可以控制的空间。
 - 高2GB的空间属于**内核态**空间，这部分空间是由操作系统控制的，用户态程序不能控制。



32位MIPS的内存分配

- 应用程序中常见的数据

```
int global;  
  
int main()  
{  
    int local;  
    char array0[10];  
    int * array1;  
    array1=(int *) malloc(10*sizeof(int));  
    free(array1);  
}
```

程序运行时这些数据都被放在内存空间的什么地方呢？

MIPS 程序和数据的存储器空间使用约定

- 从顶端开始，对栈指针初始化为 7fffffc，并向向下向数据段增长；
- 在底端，程序代码（文本）开始于 00400000；
- 静态数据开始于 10000000；
- 紧接着是由C中malloc进行存储器分配的动态数据，朝堆栈段向上增长

全局指针被设定为易于访问数据的地址，以便使用相对于\$gp的±16位偏移量

$10000000_{\text{hex}} - 1000ffff_{\text{hex}}$

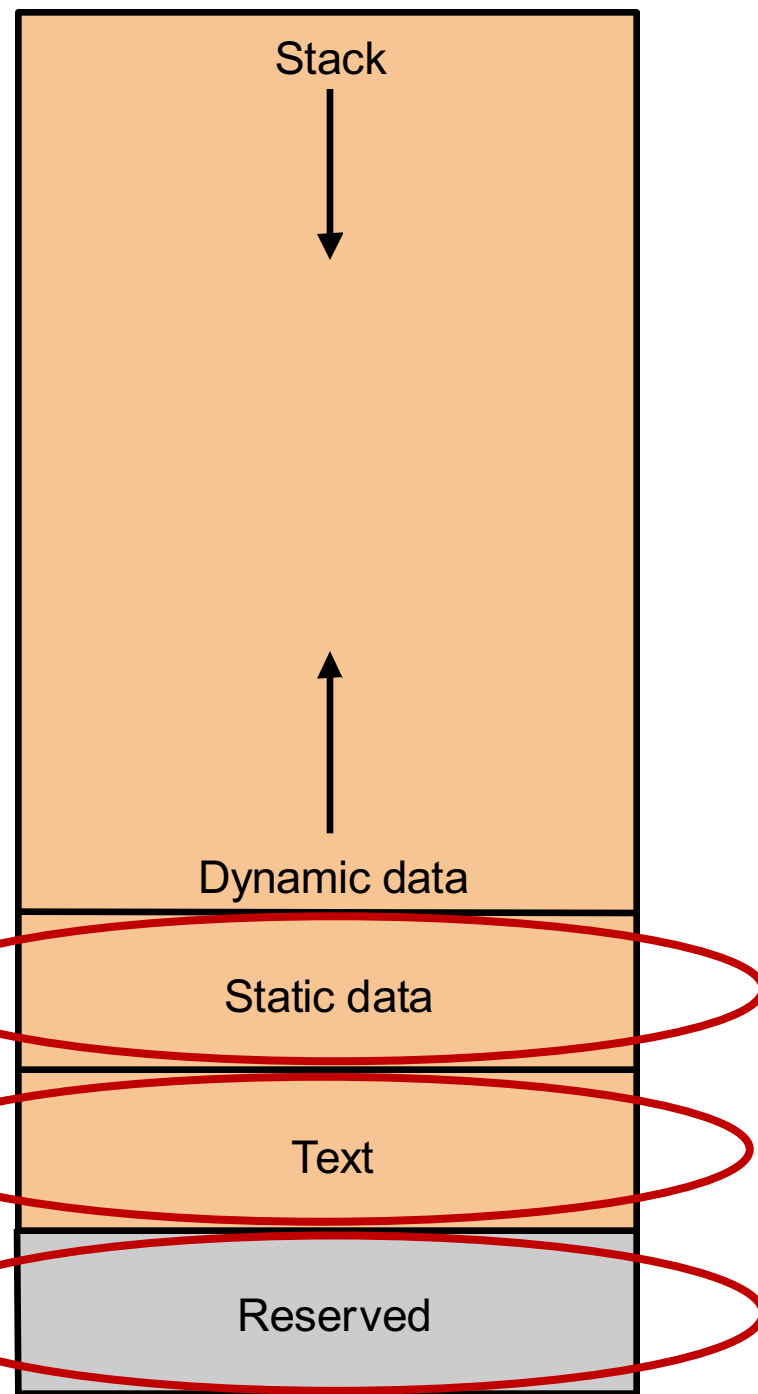
\$gp → 1000 8000 hex

1000 0000 hex

pc → 0040 0000 hex

0

0x7fff_ffff
\$sp → 0x7fff_effc



MIPS汇编器：语法

- 注释行以 “#” 开始；
- **标识符**由字母、下划线（_）、点（.）构成，但不能以数字开头，指令操作码是一些保留字，不能用作标识符；
- 标识符放在行首，后跟冒号（:），例如

```
.data          # 将子数据项，存放到数据段中
Item: .word 1,2  # 将2个32位数值送入地址连续的内存字中
.text          # 将子串，即指令或字送入用户文件段
.global main   # 必须为全局变量
Main: lw $t0, item
```

MIPS汇编器：语法

MIPS汇编语言语句格式

- 指令与伪指令语句

[Label:] <op> Arg1, [Arg2], [Arg3] [#comment]

例如 AddFunc: add \$a1 \$a2 \$a3 # a1=a2+a3

- 汇编命令(directive)语句

[Label:] .Directive [arg1], [arg2], ... [#comment]

例如 .word 0xa3

MIPS汇编器：语法

汇编命令

- 汇编器用来定义数据段、代码段以及为数据分配存储空间

`.data [address]` # 定义数据段，[address]为可选的地址

`.text [address]` # 定义正文段(即代码段)，[address]为可选的地址

`.align n` # 以 2^n 字节边界对齐数据，只能用于数据段

`.ascii <string>` # 在内存中存放字符串

`.asciiz <string>` # 在内存中存放NULL结束的字符串

`.word w1, w2, ..., wn` # 在内存中存放n个字

`.half h1, h2, ..., hn` # 在内存中存放n个半字

`.byte b1, b2, ..., bn` # 在内存中存放n个字节

MIPS汇编器：语法

- 汇编语言源文件：.s
 - “.” MIPS汇编命令标识符
 - “label:”
 - label被赋值为当前位置的地址
 - fact = 0x00400100
 - 编译时就确定了
 - 汇编程序在地址0x00400000开始

该程序功能是**求阶乘**，结果是 $f=n!$ ！通过跳转到fact子程序，通过一个循环将n到1都乘在\$*s0*上来求 $n!$ 。

注意！ *n*, *f*超出了16位的偏移，而sw和lw是I型指令，所以这样的指令是写不出来的，该程序实际无法运行。

| | |
|------------|--------------------------|
| 0x00400020 | move \$s5, \$31 |
| 0x00400024 | jal fact |
| 0x00400028 | sw \$s0,f(\$0) |
| ... | .text 0x00400100 |
| 0x00400100 | fact: addiu \$s0,\$0,1 |
| 0x00400104 | lw \$s1, n(\$0) |
| 0x00400108 | loop: mul \$s0,\$s1,\$s0 |
| 0x0040010C | addi \$s1,\$s1,-1 |
| 0x00400110 | bnez \$s1,loop |
| 0x00400114 | jr \$31 |
| ... | .data 0x10000200 |
| 0x10000200 | n: .word 4 |
| 0x10000204 | f: .word 0 |

MIPS汇编器：语法

- 指令与伪指令
 - 有一些MIPS指令是和机器码一一对应，可以直接翻译成机器码。
 - `add $a,$b,$c` `lw $t1 4($s1)`
 - 还有一些没有对应的机器码，不能直接翻译成机器码，需要先翻译成真的指令。
 - `li $s1 0x7f` `<===> addi $s1 $zero 0x7f`
 - 编写程序时使用伪指令有利于提高效率并增加可读性。

MIPS汇编器：变量

- **变量存储在主存储器内**（而不是寄存器内）
 - 因为我们通常有很多的变量要存，不止32个
- 为了实现功能, 用lw 语句将变量加载到寄存器中, 对寄存器进行操作, 然后再把结果sw回去
- **对于比较长的操作(e.g., loops):**
 - 让变量在寄存器中保留时间越长越好
 - lw and sw 只在一块代码开始和结束时使用
 - 节省指令
 - 而且事实上lw and sw 比寄存器操作要慢得多得多！
- 由于一条指令只能采用两个输入, 所以必须采用临时寄存器计算复杂的问题e.g., $(x+y)+(x-y)$

MIPS汇编器：变量

```
.data 0x10000000
.word 4,0
.text
main: addu $s3,$ra,$0
      ori  $s6,$0,0x1000
      sll  $s6,$s6,16
      addiu $s5,$s6,4
fact: addiu $s0,$0,1
      lw   $s1,0($s6)
loop: mul   $s0,$s1,$s0
      addi  $s1,$s1,-1
      bnez  $s1,loop
      sw    $s0,0($s5)
      jr    $ra
```

在程序起始处保存ra是一种习惯，目的是避免在程序中有jal指令修改了ra，我们跳不回去了，本程序中没有用可删除以节省寄存器和指令数。

`lui $s6, 0x1000`

`#s1 get 4`

`#s0 hold result`

`#return result in s0`

MIPS汇编器：分支

- 在符号汇编语句中,分支语句的目标位置是用绝对地址方式写的

- e.g., **beq \$0,\$0,fact**

means **PC \leftarrow 0x00400100**

- 不过在翻译器实现中,要用相对于PC的地址来定义

- e.g., **beq \$0,\$0,0x??**

means **PC \leftarrow 0x00400100 ?**

需要计算出偏移量

```
.text
main: addu $s3,$ra,$0
      ori  $s6,$0,0x1000
      sll  $s6,$s6,16
      addiu $s4,$s6,0x0200
      addiu $s5,$s6,0x0204
      beq  $0,$0, fact
result: sw  $s0,0($s5)
        addu $ra,$s3,$0
        jr   $ra
        .text 0x00400100
fact:   sw  $ra,0($s7)
        addiu $s0,$0,1
        lw   $s1,0($s4)    # $s0=n!
loop:   mul  $s0,$s1,$s0
        addi  $s1,$s1,-1
        bnez  $s1,loop      # f=n!
        j     result
        .data 0x10000200
n:      .word 4
f:      .word 0
```

MIPS汇编器：分支

- **偏移量**= 从下一条指令对应的PC开始到标号位置还有多少条指令
 - e.g., **beq \$0,\$0, fact**如果位于地址**0x00400000**的话,
word displacement=(**target** - (<PC>+4)) / 4
$$=(0x00400100-0x00400004)/4$$
$$=0xfc/4=0x3f$$
 - 偏移量为0则表示执行下一条指令不产生任何跳转
- **为什么在代码中用相对的偏移量?**
 - *relocatable* (可重新定位的), 分支语句可以在每次被加载到内存不同位置的情况下正常工作

MIPS汇编器：分支

- 分支

- 如果和 0 比较, 则直接使用blez, bgez, bltz, bgtz, bnez (指令)
- 更复杂的比较, 采用比较指令 (如slt), 然后再用与0比较

- Example: test2

```
if (x >= 0)
    y = x;
else
    y = -x;
```

MIPS汇编器：分支

功能：求绝对值

```
.data 0x10000000
.word -6,0
.text
```

#x : -6, y : 0

main:

```
ori    $s6,$0,0x1000
```

#计算内存中数据存放地址

```
sll     $s6,$s6,16
```

#\$s6=x的地址

```
addiu   $s5,$s6,4
```

#\$s5=y的地址

```
lw      $s0,0($s6)
```

#s0 = -6

```
slt     $s2,$0,$s0
```

#0<x, \$s2=1

```
beqz    $s2,else
```

#\$s2=0, 跳到else

```
move    $s1,$s0
```

#\$s2=1, 跳到done

```
j done
```

```
else:   sub    $s1,$0,$s0
```

```
done:   sw     $s1,0($s5)
```

```
jr      $ra
```


MIPS汇编器：数组

- 用 `.word` 来给数组开辟空间
 - 在编译时 **静态地** 开辟 $n \times 4$ bytes, (n 个 32-bit 字)

- 使用 `lw` 和 `sw` 访问数组

| | |
|--------------------------------|---------------------------|
| <code>lw \$temp, 0(\$A)</code> | <code>temp = A[0];</code> |
| <code>sw \$temp, 8(\$B)</code> | <code>B[2] = temp;</code> |

- 将常数 0, 8 作为地址偏移量
- 将寄存器 `$A` 和 `$B` 作为数组中的开始地址 (`A[]`, `B[]`)

注意这里 `$A`, `$B` 存的是地址, `$temp` 存的是具体的值

MIPS汇编器：数组

```
.data 0x10000000
.word 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17
.text
main: addu $s7,$ra,$0
      ori  $s5,$0,0x1000
      sll  $s5,$s5,16      # $s5=A[]=0x10000000,
      addiu $s6,$s5,0x400  # $s6=B[]=0x10000400
      addiu $s0,$0,0x11    # Size(A)=Size(B)=0x11
loop: subu  $s0,$s0,1      # 计数
      addiu $s1,$0,4
      mul   $s2,$s1,$s0    # 换算地址
      addu  $s3,$s2,$s5    # 计算A[]偏移量,送到$s3
      lw    $s4,0($s3)     # 读出A[]中的值
      addu  $s3,$s2,$s6    # 计算B[]偏移量,送到$s3
      sw    $s4,0($s3)     # 写到B[]中去
      bnez  $s0,loop
      addu  $ra,$0,$s7     # 返回调用程序
      jr   $ra
```

每个数4Bytes, 1 word

功能：将数组A的值依次拷贝到数组B中

MIPS汇编器：数组

使用移位操作代替mul和div：因为 mul 和 div 一般都比sll和srl慢

- sllv by k 等价于 mul by 2^k
 - srlv by k 等价于 div by 2^k
- 只对无符号数成立，
且没有超出数据表示范围
- 对于有符号数用 sra
 - 高位用符号位填充(在2的补码表示情况下)
 - e.g.,
R1 = -6 = 0b11...11010
SRL \$R1,\$R1,1 → 0b01...11101 ×
SRA \$R1,\$R1,1 → 0b11...11101 = -3 ✓

目录

- 汇编程序设计基础
 - MIPS内存分配
 - 汇编的语法
 - 变量
 - 分支
 - 数组
 - 过程调用

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?](#)

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。

负责助教联系方式

- 陈佳煜：jiayu-ch19@mails.tsinghua.edu.cn
- 黄成宇：huang-cy20@mails.tsinghua.edu.cn

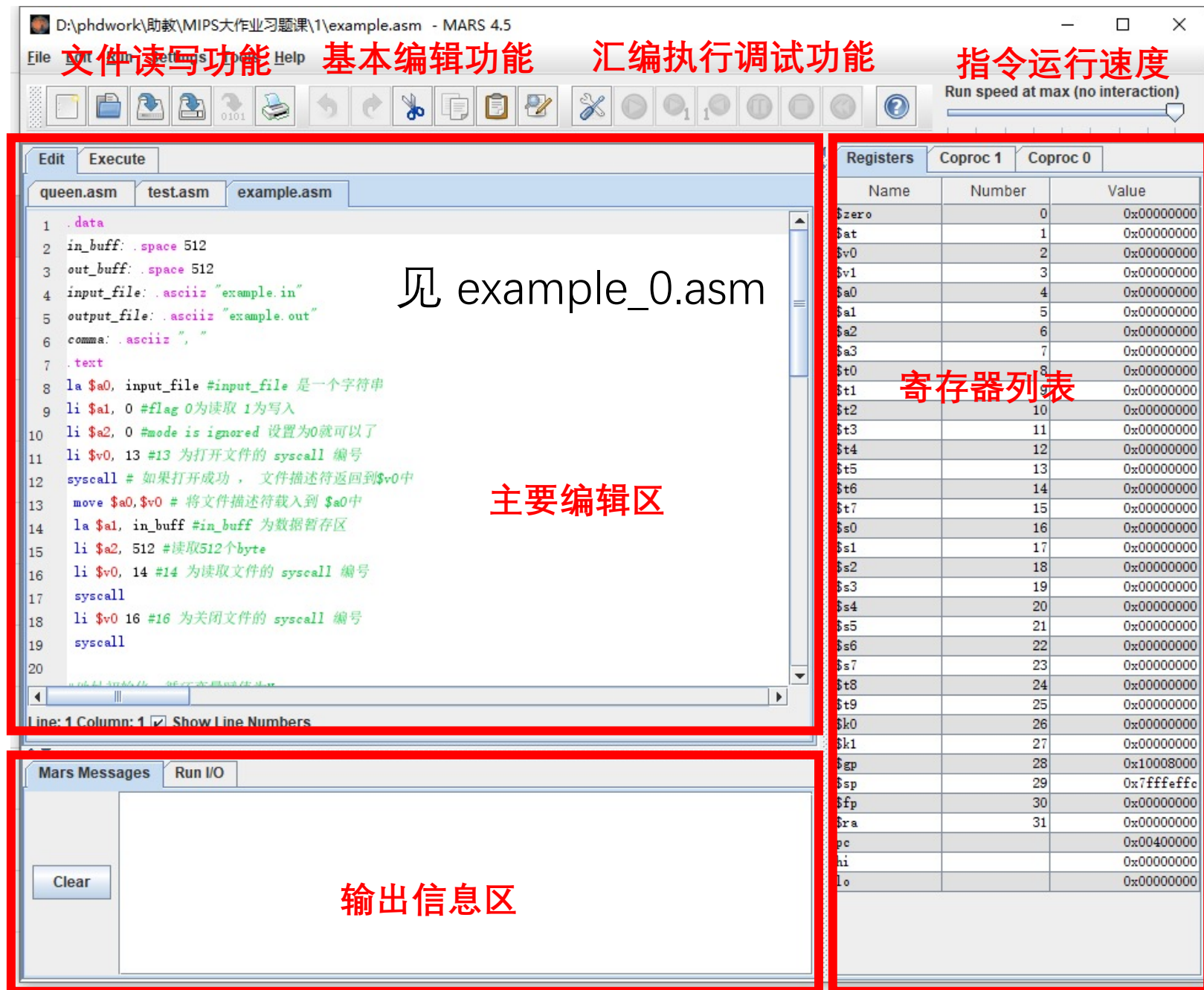
运行MARS

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。



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 | 源代码 |
|--------------------------|------------|------------|----------------------------|--|
| <input type="checkbox"/> | 0x00400000 | 0x3c011001 | lui \$1, 0x00001001 | 8: la \$a0, input_file #input_file 是一... |
| <input type="checkbox"/> | 0x00400004 | 0x34240400 | ori \$4, \$1, 0x00000400 | |
| <input type="checkbox"/> | 0x00400008 | 0x24050000 | addiu \$5, \$0, 0x00000000 | 9: li \$a1, 0 #flag 0为读取 1为写入 |
| <input type="checkbox"/> | 0x0040000c | 0x24060000 | addiu \$6, \$0, 0x00000000 | 10: li \$a2, 0 #mode is ignored 设置为0就... |
| <input type="checkbox"/> | 0x00400010 | 0x2402000d | addiu \$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 | addiu \$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

Mars Messages Run I/O

Clear Reset: reset completed.

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 | 0x7fffffc0 |
| \$fp | 30 | 0x00000000 |
| \$ra | 31 | 0x00000000 |
| pc | | 0x00400000 |
| hi | | 0x00000000 |
| lo | | 0x00000000 |

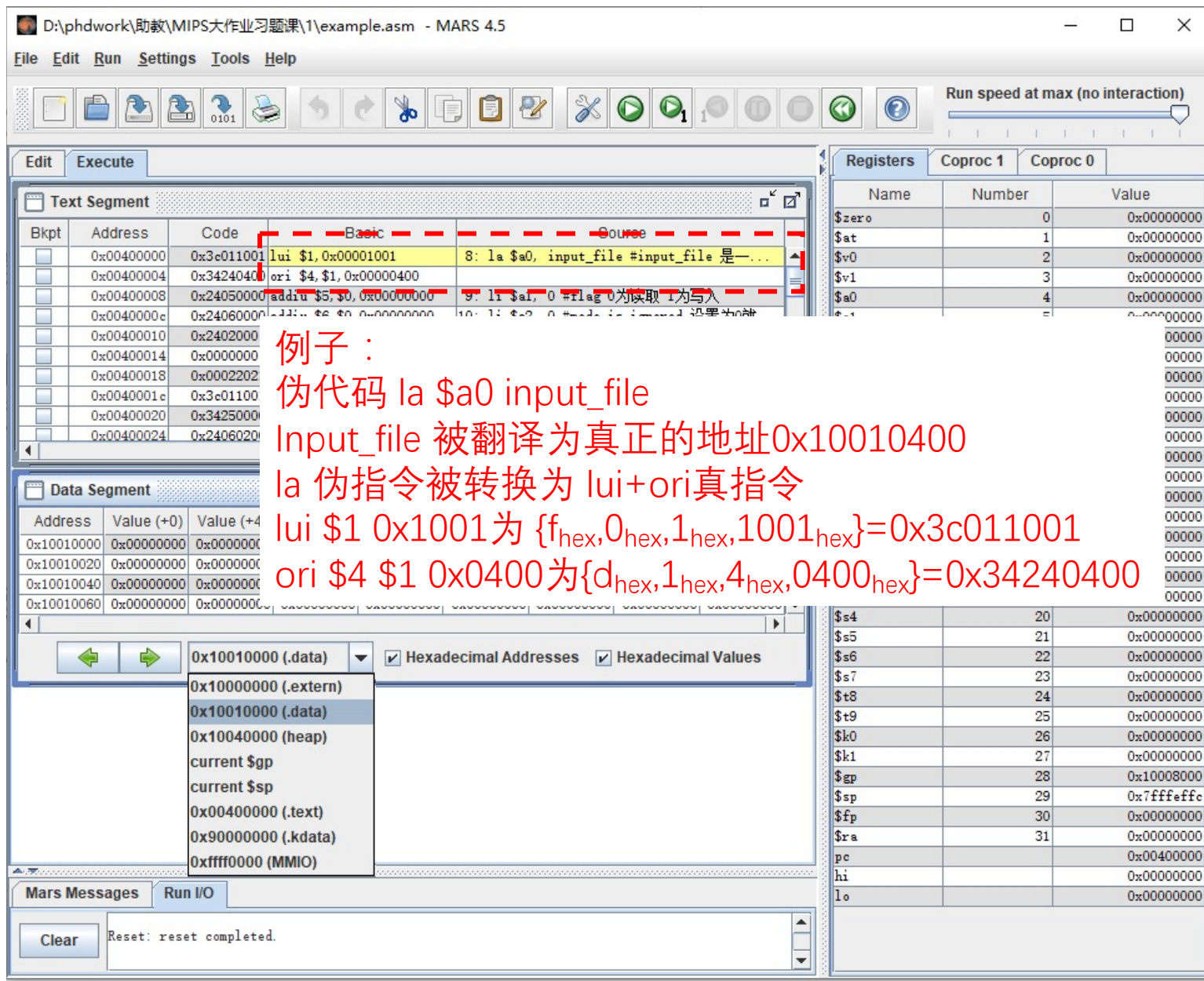
汇编运行

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)

Registers

| 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 |
| \$a4 | 8 | 0x00000000 |
| \$a5 | 9 | 0x00000000 |
| \$a6 | 10 | 0x00000000 |
| \$a7 | 11 | 0x00000000 |
| \$s0 | 12 | 0x00000000 |
| \$s1 | 13 | 0x00000000 |
| \$s2 | 14 | 0x00000000 |
| \$s3 | 15 | 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 | 0x7fffffc0 |
| \$fp | 30 | 0x00000000 |
| \$ra | 31 | 0x00000000 |
| pc | | 0x00400000 |
| hi | | 0x00000000 |
| lo | | 0x00000000 |

Text Segment

| Bkpt | Address | Code |
|------|------------|---------------------------------------|
| | 0x00400000 | 0x3c011001 lui \$1, 0x0001001 |
| | 0x00400004 | 0x34240400 ori \$4, \$1, 0x00000400 |
| | 0x00400008 | 0x24050000 addiu \$5, \$0, 0x00000000 |
| | 0x0040000c | 0x24060000 addiu \$6, \$0, 0x00000000 |
| | 0x00400010 | 0x24020000 addiu \$2, \$0, 0x00000000 |
| | 0x00400014 | 0x00000000 |
| | 0x00400018 | 0x0002202 |
| | 0x0040001c | 0x3c01100 |
| | 0x00400020 | 0x3425000 |
| | 0x00400024 | 0x2406020 |

Data Segment

| Address | Value (+0) | Value (+4) |
|------------|------------|------------|
| 0x10010000 | 0x00000000 | 0x00000000 |
| 0x10010020 | 0x00000000 | 0x00000000 |
| 0x10010040 | 0x00000000 | 0x00000000 |
| 0x10010060 | 0x00000000 | 0x00000000 |

0x10010000 (.data)

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

Hexadecimal Addresses Hexadecimal Values

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 | 0x7fffffc0 |
| \$fp | 30 | 0x00000000 |
| \$ra | 31 | 0x00000000 |
| pc | | 0x00400000 |
| hi | | 0x00000000 |
| lo | | 0x00000000 |

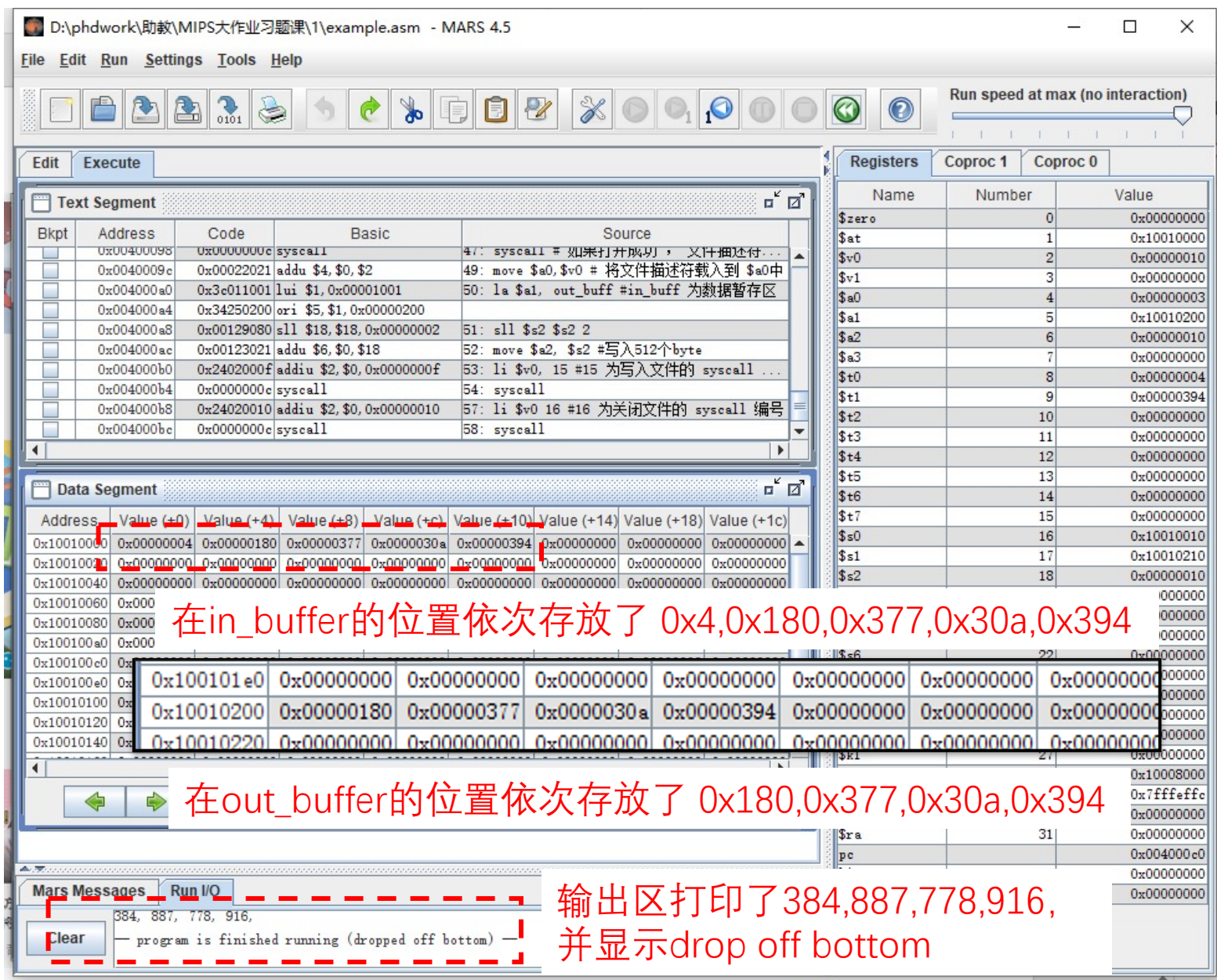
汇编运行

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）



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

输出区打印了384,887

PC取值为0x00400070

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

注意读取文件需要和Mars.jar在同一目录下这个例程才能正常读取

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。每执行完一条指令后会顺序执行下一条指令，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。

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

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。每执行完一条指令后会顺序执行下一条指令，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。

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

子过程，接上页

```
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
```

实验内容1

- 用MIPS32汇编指令完成下列任务， 调试代码并获得正确的结果。
- 练习1-1：系统调用。
- 练习1-2：循环， 分支。
- 练习1-3：数组， 指针。
- 练习1-4：函数调用。

练习1-1：系统调用

- 练习使用MARS模拟器中的系统调用syscall，使用syscall可以完成包括文件读写，命令行读写（标准输入输出），申请内存等辅助功能。
- 系统调用基本的使用方法是
 1. 向\$a*寄存中写入需要的参数（如果有）
 2. 向\$v0寄存器中写入需要调用的syscall的编号
 3. 使用” syscall” 指令进行调用
 4. 从\$v0中读取调用的返回值（如果有）
- 更多具体的使用方法可以参照MARS模拟器的Help中的相关内容。

练习1-1：系统调用

- 用MIPS汇编指令实现exp1_1_sys_call.cpp的功能并提交汇编代码，尽量在代码中添加注释。

- exp1_1_sys_call.cpp代码内容主要包括：
 - 申请一个8byte整数的内存空间。
 - 从“a.in”读取两个整数。
 - 向“a.out”写入这两个整数。
 - 从键盘输入一个整数i。
 - $i = i + 1$ 。
 - 向屏幕打印这个整数。

写汇编程序时注意文件的读取路径

exp1_1_sys_call.cpp文件内容

```
1. #include "stdio.h"
2. void main()
3. {
4.     FILE * infile, *outfile;
5.     int i, max_num=0, id;
6.     int* buffer;
7.     buffer = new int[2];
8.     infile = fopen("a.in", "rb");
9.     fread(buffer, 4, 2, infile);
10.    fclose(infile);
11.    outfile = fopen("a.out", "wb");
12.    fwrite(buffer, 4, 2, outfile);
13.    fclose(outfile);
14.    scanf("%d", &i);
15.    i = i + 1;
16.    printf("%d", i);
17. }
```

练习1-2：循环, 分支

2、用MIPS语言实现 exp1_2_loop.cpp中的功能并 提交汇编代码，尽量在代码中 添加注释。

- exp1_2_loop.cpp代码内容主要包括：
 - 将输入值取绝对值，存在变量i，j中
 - 从变量i开始，循环j轮，每轮i = i+1

```
1. #include "stdio.h"↵
2. void main()↵
3. {↵
4. int i,j,temp;↵
5. scanf("%d",&i);↵
6. scanf("%d",&j);↵
7. if (i<0){i=-i;}↵
8. if (j<0){j=-j;}↵
9. for(temp=0;temp<=j;++temp)↵
10. {↵
11. i += 1;↵
12. }↵
13. printf("%d",i);↵
14. }↵
```

exp1_2_loop.cpp文件内容

练习1-3：数组、指针

- 用MIPS汇编指令实现exp1_3_array.cpp 的功能并提交汇编代码，尽量在代码中添加注释。
- exp1_3_array.cpp代码内容主要包括：
 - 输入数组a的长度n
 - 任意输入n个整数
 - 将数组a逆序，并且仍然存储在a中
 - 打印数组a的值

exp1_3_array.cpp文件内容

```
1. #include "stdio.h"
2. void main()
3. {
4.     int *a, n, i, t;
5.     scanf("%d",&n);
6.     a = new int [n];
7.     for(i=0;i<n;i++)
8.     {
9.         scanf("%d",&a[i]);
10.    }
11.    for(i=0;i<n/2;i++){
12.        t = a[i];
13.        a[i] = a[n-i-1];
14.        a[n-i-1] = t;
15.    }
16.    for(i=0;i<n;i++) printf("%d ",a[i]);
17. }
```


练习1-4：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. 将返回地址、保存寄存器入栈。
2. 使用输入参数\$a0~\$a3执行函数内容并将结果存入返回寄存器\$v0~\$v1。
3. 将返回地址、保存寄存器出栈。
4. 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 |

逐步完成函数调用的编译

```
int Hanoi(int n)
{
    if (n == 1)
    {
        return 1;
    }
    else
    {
        return 2 * Hanoi(n - 1) + 1;
    }
}
```

```
Hanoi(3)
= 2*Hanoi(2) + 1
= 2*(2*Hanoi(1) + 1) + 1
= 2*(2*1 + 1) + 1
= 7
```

1. 先编译其他语句。
2. 拆解编译调用函数的语句
3. 在首尾分别补上入栈和出栈

```
int Hanoi(int n)
{
```

```
    if (n == 1)
```

```
    {
        return 1;
    }
```

```
else
```

```
{
    return 2 * Hanoi(n - 1) + 1;
}
```

```
}
```

Hanoi : #将参数放入\$a0

addi \$t0 \$0, 1

bne \$t0 \$a0 Next

addi \$v0 \$0 1 # 返回1

jr \$ra

Next:

(保护现场)

addi \$s1 \$0 1

(调用Hanoi(n-1), 返回值存在\$v0)

add \$s1 \$v0 \$s1

add \$s1 \$v0 \$s1

add \$v0 \$s1 \$0

(恢复现场)

jr \$ra

练习1-4：函数调用

- 继续完成Hanoi过程的编译，使得其可以完成计算Hanoi(n)的任务。（在实验报告中完成即可，不需要提交相应汇编程序）

作业提交要求

- 作业用一个压缩包提交，压缩包名称：“学号_姓名.7z”。推荐用7z格式，其他常见压缩格式也可以。
- 压缩包打开后需要包含：
 - 一个“实验报告.pdf”文件，完成实验内容
 - 一个“exp_1_1.asm”，一个“exp_1_2.asm”，一个“exp_1_3.asm”
- 注意所有的MIPS代码需要和C语言代码对应，不可使用其他C程序！！

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