微机原理
(计算机原理)

第6讲 MIPS指令系统(3)

第6讲 MIPS指令系统(3)

- MIPS模拟器
- MIPS汇编语言语法
- MIPS汇编程序实例

SPIM

- SPIM是主要的MIPS模拟器，能够运行和调试MIPS汇编语言程序
- SPIM支持Uinx、Windows等多个操作系统平台
- http://spimsimulator.sourceforge.net/

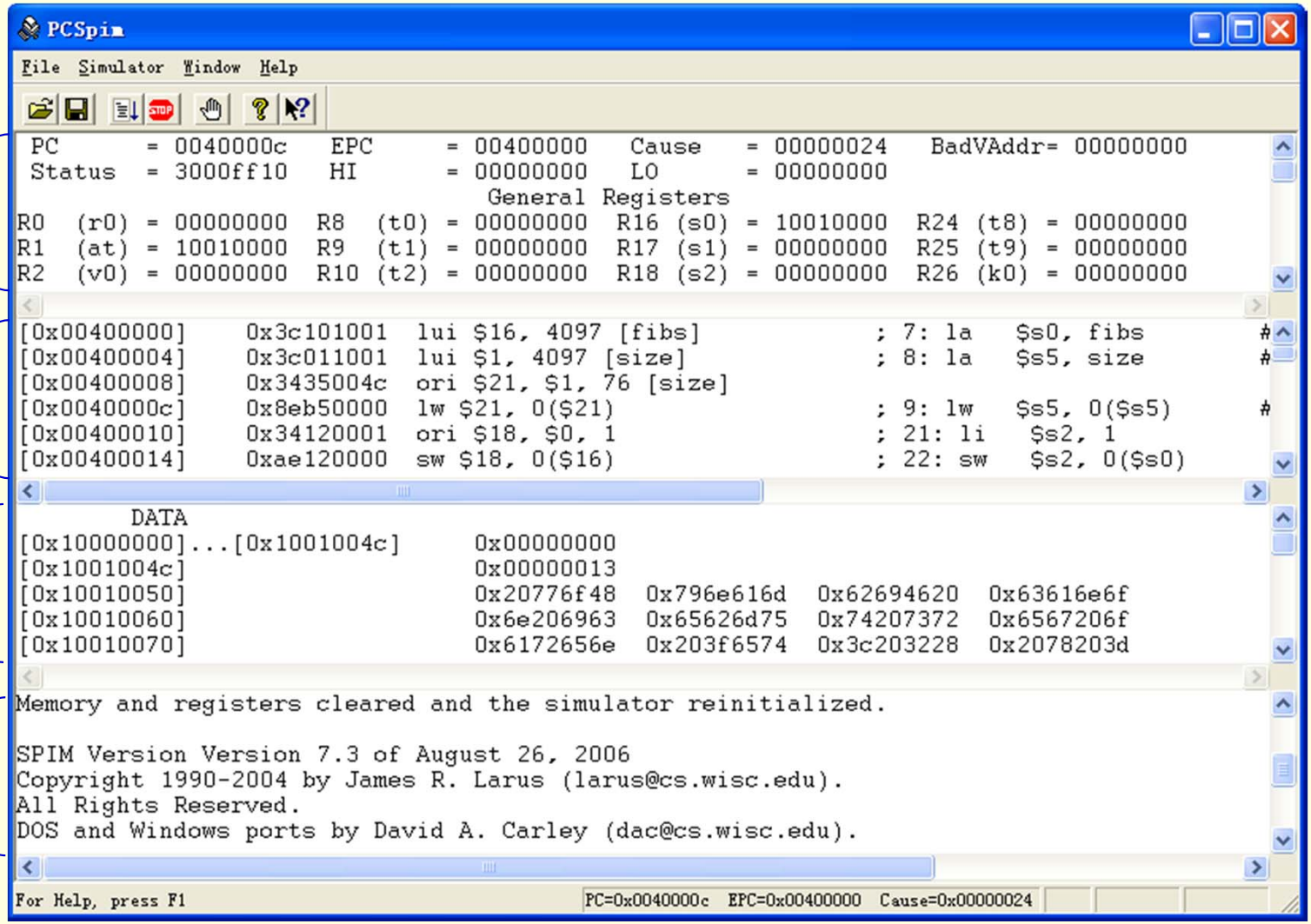
SPIM

寄存器窗口

正文段

数据与堆栈段

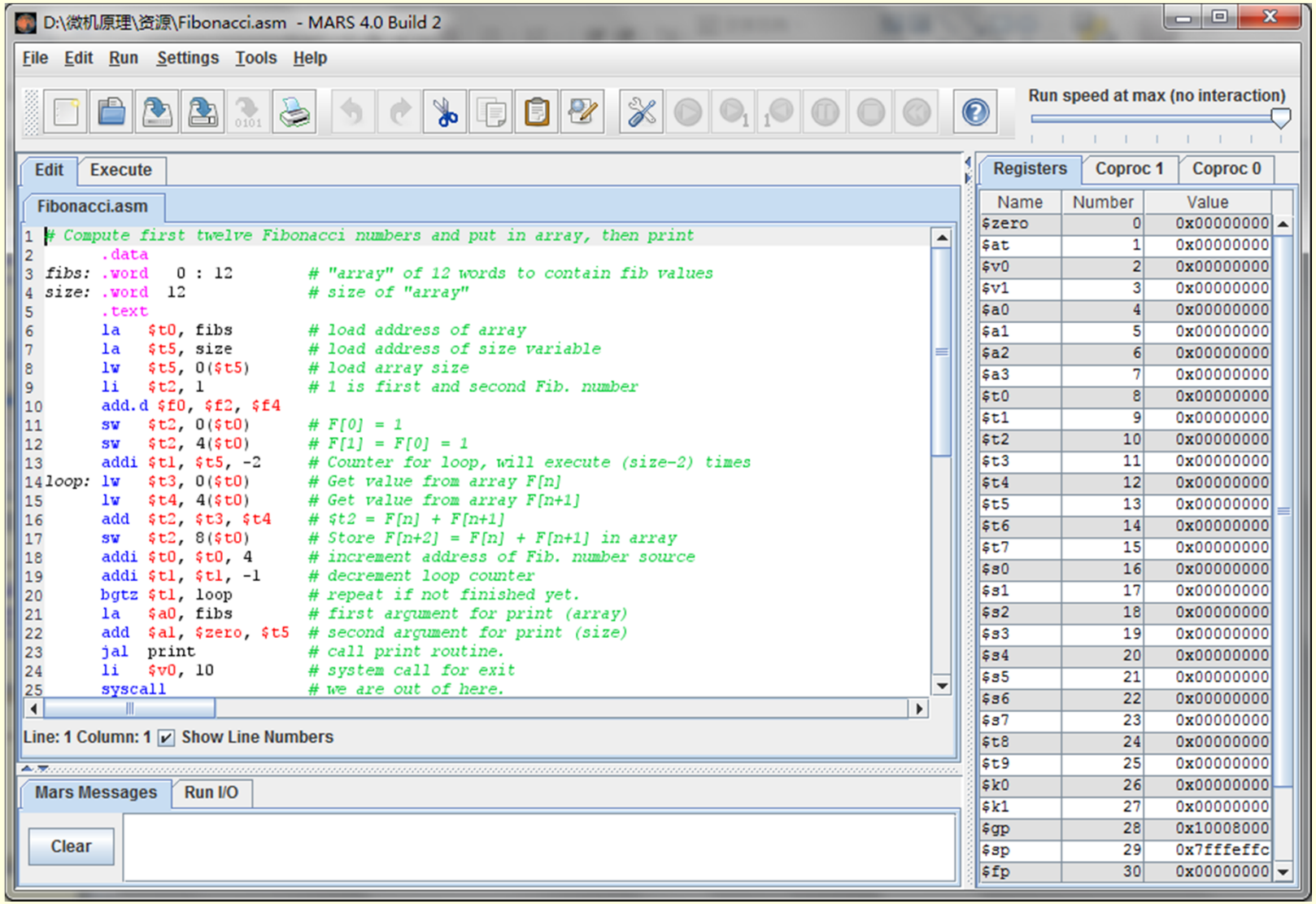
SPIM消息



MARS

- MARS 是MIPS Assembler and Runtime Simulator (MIPS汇编器和运行时模拟器)的缩写
- 能够运行和调试MIPS汇编语言程序
- MARS采用Java开发，跨平台
- http://courses.missouristate.edu/KenVollmar/MARS/

MIPS模拟器


D:\微机原理\资源\Fibonacci.asm - MARS 4.0 Build 2
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Fibonacci.asm
1 # Compute first twelve Fibonacci numbers and put in array, then print
2 .data
3 fibs: .word 0 : 12 # "array" of 12 words to contain fib values
4 size: .word 12 # size of "array"
5 .text
6 la $t0, fibs # load address of array
7 la $t5, size # load address of size variable
8 lw $t5, 0($t5) # load array size
9 li $t2, 1 # 1 is first and second Fib. number
10 add.d $f0, $f2, $f4
11 sw $t2, 0($t0) # F[0] = 1
12 sw $t2, 4($t0) # F[1] = F[0] = 1
13 addi $t1, $t5, -2 # Counter for loop, will execute (size-2) times
14 loop: lw $t3, 0($t0) # Get value from array F[n]
15 lw $t4, 4($t0) # Get value from array F[n+1]
16 add $t2, $t3, $t4 # $t2 = F[n] + F[n+1]
17 sw $t2, 8($t0) # Store F[n+2] = F[n] + F[n+1] in array
18 addi $t0, $t0, 4 # increment address of Fib. number source
19 addi $t1, $t1, -1 # decrement loop counter
20 bgtz $t1, loop # repeat if not finished yet.
21 la $a0, fibs # first argument for print (array)
22 add $a1, $zero, $t5 # second argument for print (size)
23 jal print # call print routine.
24 li $v0, 10 # system call for exit
25 syscall # we are out of here.
Line: 1 Column: 1 Show Line Numbers
Mars Messages Run I/O
Clear
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 0x7fffeffc
$fp 30 0x00000000

系统调用

- **MIPS模拟器通过系统调用指令(syscall) 提供了一组类似操作系统的服务**
- 调用方法：
  - 将系统调用代码装入**$v0(寄存器编号2)**寄存器
  - 将参数(如果有)装入**$a0~$a3(寄存器编号4~7)**或**$f12**寄存器
  - **Syscall**
  - 返回值保存在**$v0**或**$f0**寄存器中

系统调用

代码	系统调用	参数	结果
1	print integer	$a0	
2	print float	$f12	
3	print double	$f12	
4	print string	$a0	
5	read integer		integer in $v0
6	read float		float in $f0
7	read double		double in $f0
8	read string	$a0=buffer, $a1=length	
9	sbrk	$a0=amount	address in $v0
10	exit		

系统调用

代码	系统调用	参数	结果
11	print char	$a0	
12	read char		char in $v0
13	open	$a0=file name(string), $a1=flags, $a2=mode	file descriptor (fd) in $v0
14	read	$a0 =fd, $a1=buffer, $a2=length	num chars read in $v0
15	write	$a0 =fd, $a1=buffer, $a2=length	num chars write in $v0
16	close	$a0 =fd	
17	exit2	$a0=result	

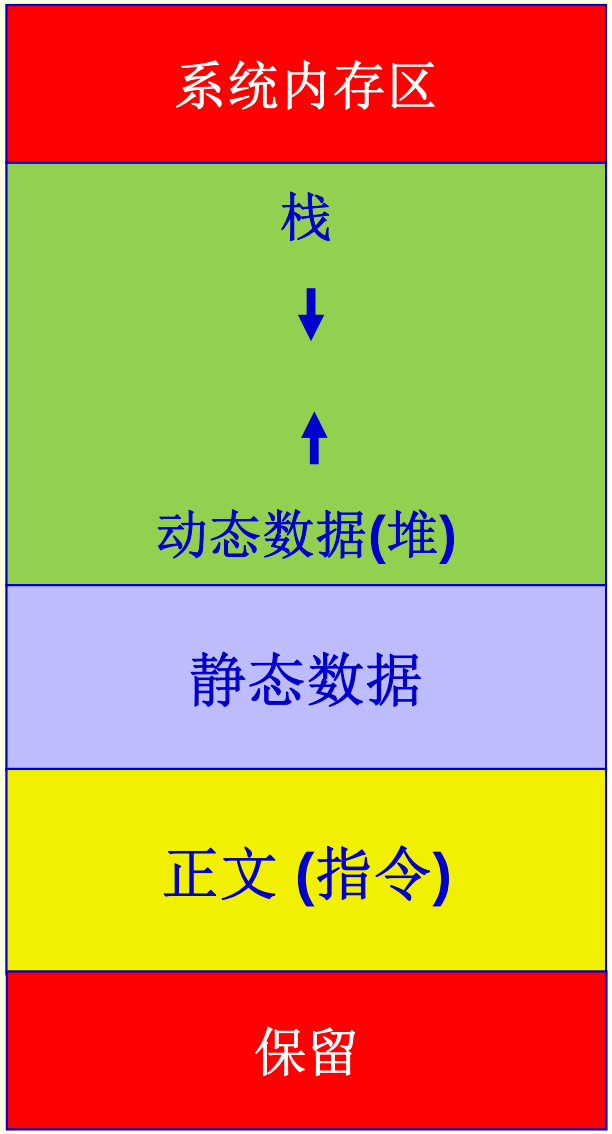
Hello world

```
        .text
main:
        la      $a0, str
        li      $v0, 4
        syscall                 # print string
        li      $v0, 10
        syscall                 # exit

        .data
        .align 2
str:
        .asciiz "Hello world."
```

内存布局

- 内存最低端保留，其上是正文段(代码段)
- 正文段之上是静态数据段，存放全局常量和其他静态数据
- 静态数据段之上为分配动态数据的堆
- 栈由用户内存区的高地址端开始，栈空间由高地址向低地址增长
- 各部分地址由软件规定， 并非MIPS体系结构的一部分

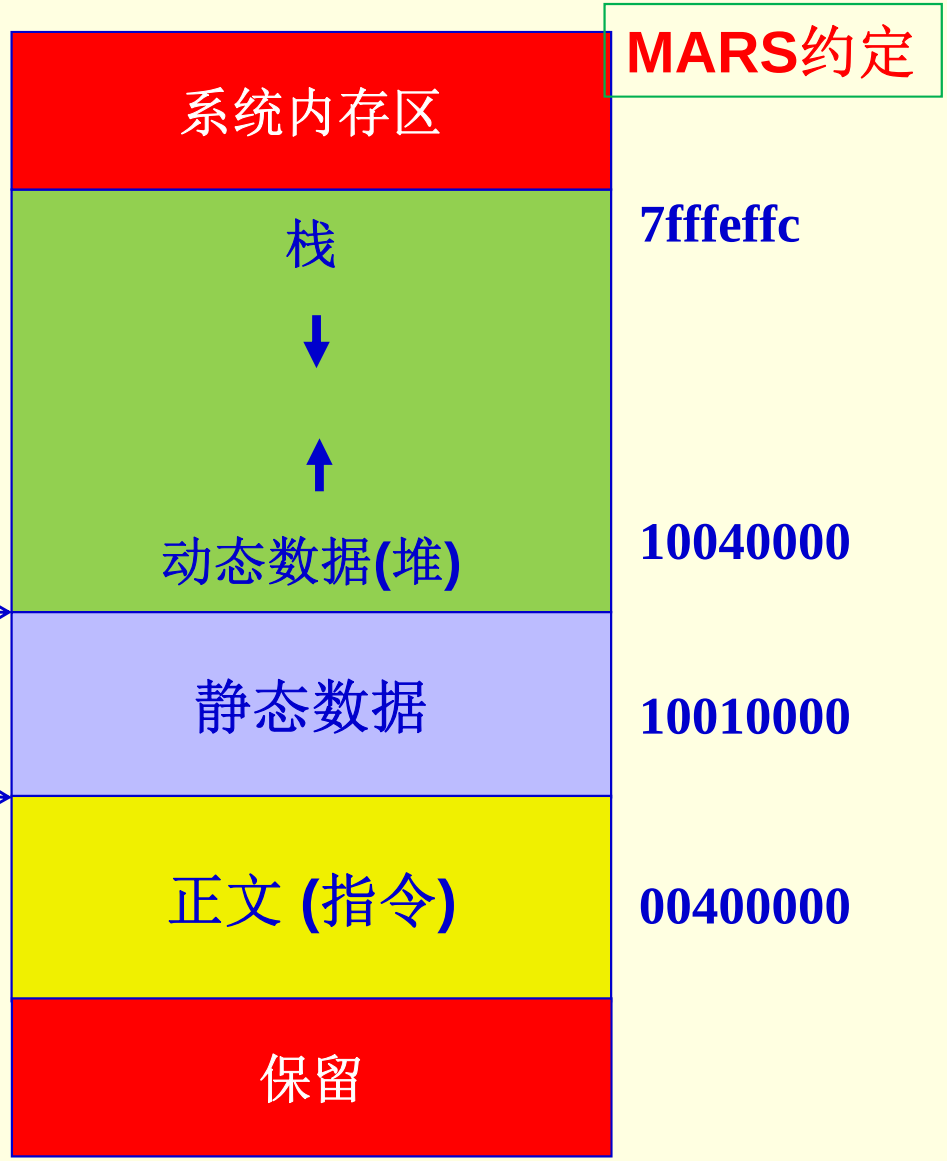


Hello world

```
            .text
main:
            la      $a0, str
            li      $v0, 4
            syscall

            li      $v0, 10
            syscall

            .data
            .align 2
str:
            .asciiz "Hello world."
```



第6讲 MIPS指令系统(3)

- MIPS模拟器
- MIPS汇编语言语法
- MIPS汇编程序实例

数据类型

- 整型数据：支持十进制、十六进制、八进制，表示法和C语言相同
  - 十进制数前不可以有无效的0
  - 八进制数加前导0
  - 十六进制加前导0x

数据类型

- 浮点型数据：表示方法

  <尾符>d1[.d2][e|E<阶符>d3]

- 其中，**d1、d2、d3**都是十进制数
- 例如：

  36.25e-2

数据类型

- 字符串：表示形式与C语言类似，以双引号为标志
- 遵守C语言的“\”使用规则：
  - 换行——\n
  - 制表符——\t
  - 引号——\”

标识符

- 由字母、数字、下划线(_)、点(.)构成，但不能以数字开头
- 指令助记符等保留字不能作标识符

MIPS汇编程序语句格式

- 指令与伪指令语句

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

- 汇编命令**(directive)**语句

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

- **Label**：为可选的标号，**#comment**为可选的注释

伪指令

- 伪指令**(Pseudo-instructions)**是为编程方便而对指令集进行的扩展，并非真正的指令。例如，**ble**、**move**等
- 编程时，伪指令可以和指令一样在程序中使用，在汇编时伪指令将被等效的指令取代

常用的伪指令

- **move——寄存器到寄存器的数据传送**

  move $t2, $t4

  → **add $t2, $zero, $t4**

- **li——装入立即数，不同立即数的可能汇编成不同的指令序列**

  li $t1, 40

  → **addi $t1, $zero, 40**

  li $t1, -4000000

  → **lui $at, 0xffc2**

  ori $t1, $at, 0xf700

常用的伪指令

- la——装入地址

  la $t4, 0x1000056c

  → **lui $at, 0x1000**
  ori $t4, $at, 0x056c

常用的伪指令

- 分支伪指令
  - **beqz rsrc, label——当寄存器rsrc=0时分支**
  - **bnez rsrc, label——当寄存器rsrc≠0时分支**

beqz $s1, label

→ **beq $s1, $zero, label**

常用的伪指令

- 分支伪指令
  - **bge(u) rsrc1, src2, label——rsrc1>=src2时分支**
  - **bgt(u) rsrc1, src2, label——rsrc1>src2时分支**
  - **ble(u) rsrc1, src2, label——rsrc1<=src2时分支**
  - **blt(u) rsrc1, src2, label——rsrc1<src2时分支**

  其中，**rsrc1**是寄存器，**src2**是寄存器或者立即数

```
bleu $s1, 100, label
addi $at, $zero, 100
sltu $at, $at, $s1
beq $at, $zero, label
```

汇编命令(directive)

- 指示汇编器定义数据段、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
```

汇编命令

- **.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

汇编命令

- **.double d1, d2, …, dn** # 在内存中存放n个双精度浮点数
- **.float f1, f2, …, fn** # 在内存中存放n个单精度浮点数
- **.global sym** # 指示sym为全局标记

宏

- 宏(macro)提供了一种对经常使用的指令序列进行命名的机制。使用宏时，程序员只需在源程序中定义一次，就可以多次调用
- 宏与子程序相似——对频繁使用的指令序列进行命名
- 宏与子程序的区别
  - 子程序调用由处理器完成，宏指令调用在汇编过程中由汇编器完成宏展开
  - 子程序调用可以减小目标程序的大小，宏指令调用不能

宏

- 宏可以有参数

```
.macro print_string($arg)
        la      $a0, $arg
        li      $v0, 4
        syscall                 # print string
.end_macro
```

第6讲 MIPS指令系统(3)

- MIPS模拟器
- MIPS汇编语言语法
- MIPS汇编程序实例

MIPS汇编程序实例

- 从键盘输入两个数，计算并输出这两个数的和

```
.data
        str1:  .asciiz  "Enter 2 numbers:"
        str2:  .asciiz  "The sum is "
.text
main:
        li   $v0, 4
        la  $a0, str1
        syscall
        li   $v0, 5
        syscall
        add  $t0, $v0, $zero
        li   $v0, 5
        syscall
        add  $t1, $v0, $zero
        li   $v0, 4
        la  $a0, str2
        syscall
        li    $v0, 1
        add  $a0, $t1, $t0
        syscall
```

MIPS汇编程序实例

- 计算$1^2+2^2+\ldots+100^2$

```
        .text
main:
        li      $t0, 1
        li      $t8, 0
loop:
        mul     $t7, $t0, $t0
        add     $t8, $t8, $t7
        addi    $t0, $t0, 1
        ble     $t0, 100, loop

        la      $a0, str
        li      $v0, 4
        syscall
        li      $v0, 1
        move    $a0, $t8
        syscall
        li      $v0, 10
        syscall

        .data
        .align 2
str:    .asciiz "The sum of square
from 1 to 100 is "
```

MIPS汇编程序实例

- 计算n! (n=6)

```
        .data
        .align 2
str:  .asciiz "The factorial is "

        .text
main:
        la      $a0, str
        li      $v0, 4
        syscall
        li      $a0, 6
        jal     fact
        move    $a0, $v0
        li      $v0, 1
        syscall
        li      $v0, 10
        syscall
```

```
fact:
        addi    $sp, $sp, -8
        sw      $ra, 4($sp)
        sw      $a0, 0($sp)
        slti    $t0, $a0, 1
        beq     $t0, $zero, L1
        addi    $v0, $zero, 1
        addi    $sp, $sp, 8
        jr      $ra
L1:
        addi    $a0, $a0, -1
        jal     fact
        lw      $a0, 0($sp)
        lw      $ra, 4($sp)
        addi    $sp, $sp, 8
        mul     $v0, $a0, $v0
        jr      $ra
```

小结

- MIPS模拟器
- MIPS汇编语言语法
- MIPS汇编程序实例

进一步学习的建议

- **2nd Edition：3.9、附录A**
- **3rd Edition：2.10、附录A**

课后作业

- **2.37、2.38**

实验1：MIPS汇编语言程序设计

- 题目：A.6、A.7、A.8、A.9、A.10
- 实验报告要求：
  - 设计思路说明
  - 算法说明(必要时可以包含流程图)
  - 程序清单(应当包含适当的注释)
  - 运行结果