附录1、MIPS寄存器

编号	助记符	功能
0	$zero	常量0，永远返回值为0
1	$at	保留给汇编器作为暂时变量(Reserved for assembler)
2-3	$v0-$v1	保存过程调用返回的结果(values for results and expression evaluation)。
4-7	$a0-$a3	用作过程调用的参数(argument)。更多的参数采用栈传递。
8-15	$t0-$t7	暂时寄存器(temporary register)。过程使用它们时不需要保存与恢复
16-23	$s0-$s7	保存寄存器(saved register)。过程必须保存和恢复使用过的保存寄存器。
24-25	$t8-$t9	暂时寄存器(temporary register)。
26-27	$k0-$k1	通常被中断或异常处理程序使用作为保存一些系统参数。
28	$gp	全局指针(global pointer)，指向保存“static“和”extern”变量的静态数据区。
29	$sp	堆栈指针(stack pointer)，指向栈顶。
30	$fp/$s8	帧指针(frame pointer)，指向当前过程在栈中的帧的基址
31	$ra	保存调用过程的返回地址(return address)
	cp0	在MIPS体系结构中，最多支持4个协处理器(Co-Processor)。其中，协处理器CP0是体系结构中必须实现的。它起到控制CPU的作用。MMU、异常处理、乘除法等功能，都依赖于协处理器CP0来实现。它是MIPS的精髓之一，也是打开MIPS特权级模式的大门。

- **$0**: 即$zero,该寄存器总是返回零，为0这个有用常数提供了一个简洁的编码形式。
  move $t0,$t1
  实际为
  add $t0,$0,$t1
  使用伪指令可以简化任务，汇编程序提供了比硬件更丰富的指令集。

- **$1**:即$at，该寄存器为汇编保留，由于I型指令的立即数字段只有16位，在加载一个32位立即数需要 lui（装入高位立即数）和addi两条指令。汇编器可以用$at来保存大常数来完成。
  例子：指令li $t1,40是一条伪指令，在汇编器中会转换成addi $t1,$zero,40
  但是，指令 li $t1,-4000000 因为数字太大，需要拆开，则会被转换成
  lui $at,0xffc2
  ori $t1,$at,0xf700
  伪指令la $a0, error的功能与上述li类似。

- **$2..$3**:($v0-$v1)用于子程序的非浮点结果或返回值。MIPS约定堆栈中少数几个位置保留未做定义，可以用于存放更多的返回值。

- **$4..$7**:($a0-$a3)用来传递前四个参数给子程序，不够的用堆栈。MIPS编译器总是为参数在堆栈中留有空间以防有参数需要存储。

- $8..$15:($t0-$t7)临时寄存器，子程序可以使用它们而不用保留。

- **$16..$23**:($s0-$s7)保存寄存器，被调用者需要保存和恢复使用的保存寄存器，目的是减少了寄存器溢出(spilling)，即将不常用的变量放到存储器的过程。叶过程只有在临时寄存器不够用时使用保存寄存器。

- **$24..$25**:($t8-$t9)：同$t0-$t7。
- **$26..$27**:($k0,$k1)为操作系统/异常处理保留，至少要预留一个。
- **$28**:($gp) 全局指针gp(global pointer)指向静态数据区，在存取位于gp值上下32KB范围内的数据时，只需要一条以gp为基指针的指令即可。
- **$29**:($sp) MIPS硬件并不直接支持堆栈，堆栈是在内存中的一块区域。
- **$30**:($fp) GNU MIPS C编译器使用了帧指针(frame pointer), 而SGI的C编译器没有使用，而把这个寄存器当作保存寄存器使用($s8),这节省了调用和返回开销，但增加了代码生成的复杂性。
- **$31**:($ra)存放返回地址，MIPS的jal(jump-and-link)指令跳转到某个过程时，会把下一条指令的地址放到$ra中。过程中执行jr $ra返回。
- **cp0:**(control processor) 当执行指令发生异常时那条指令的地址会被保存在EPC (exception program counter)中。指令mfc0 (move from system control)可以将EPC中的地址复制到k0或k1中，再用jr跳转到异常指令处继续执行。

附录2、MIPS32汇编指令

CPU Arithmetic Instructions

助记符	指令说明	助记符	指令说明
ADD	Add Word	SLT	Set on Less Than
ADDI	Add Immediate Word		
ADDIU	Add Immediate Unsigned Word	SLTI	Set on Less Than Immediate
ADDU	Add Unsigned Word	SLTIU	Set on Less Than Immediate Unsigned
SUB	Subtract Word	SLTU	Set on Less Than Unsigned
SUBU	Subtract Unsigned Word	DIV	Divide Word
MUL	Multiply Word to GPR	DIVU	Divide Unsigned Word
MULT	Multiply Signed Word		
MULTU	Multiply Unsigned Word		

* GPR -- General Purpose Register

CPU Branch and Jump Instructions

助记符	指令说明	助记符	指令说明
BEQ	Branch on Equal	JAL	Jump and Link
BNE	Branch on Not Equal	JR	Jump Register
J	Jump		

CPU Load, Store, and Memory Control Instructions

助记符	指令说明	助记符	指令说明
LB	Load Byte	SB	Store Byte
LBU	Load Byte Unsigned	SH	Store Halfword
LH	Load Halfword	SW	Store Word
LHU	Load Halfword Unsigned	SC	Store Conditional Word
LW	Load Word		
LL	Load Linked Word		

* LBU和LB - 分别采用有符号和无符号扩展

CPU Logical Instructions

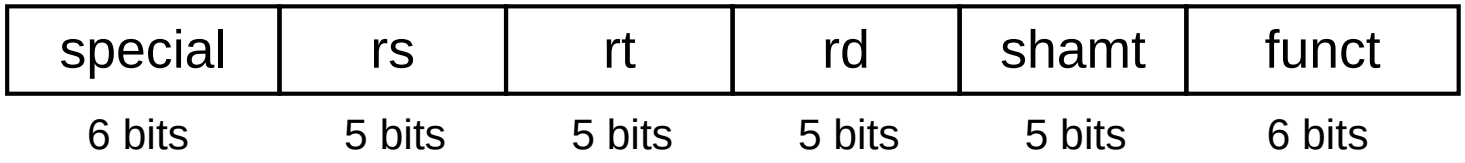
助记符	指令说明	助记符	指令说明
AND	And	XOR	Exclusive Or
ANDI	And Immediate	XORI	Exclusive Or Immediate
OR	Or	NOR	Not Or
ORI	Or Immediate	LUI	Load Upper Immediate

CPU Shift Instructions

助记符	指令说明	助记符	指令说明
SLL	Shift Word Left Logical		
SRL	Shift Word Right Logical		

附录3、MIPS32机器指令

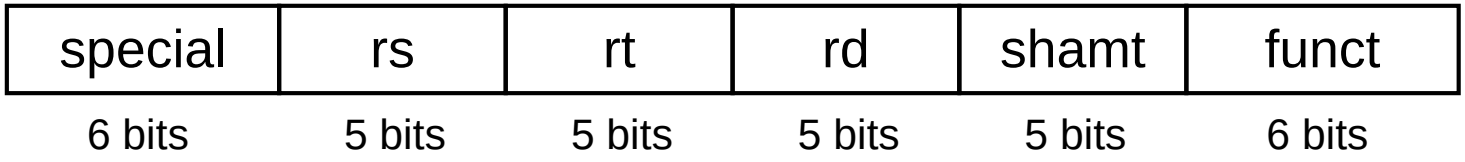
R-format Instructions（1）



format	description	special	shamt	funct
ADD rd, rs, rt	rd ← rs + rt	000000	00000	100000
ADDU rd, rs, rt	rd ← rs + rt	000000	00000	100001
SUB rd, rs, rt	rd ← rs - rt	000000	00000	100010
SUBU rd, rs, rt	rd ← rs - rt	000000	00000	100011
MUL rd, rs, rt	rd ← rs X rt	011100	00000	000010
MULT rs, rt	(HI, LO) ← rs × rt	000000	00000	011000
MULTU rs, rt	(HI, LO) ← rs × rt	000000	00000	011001
DIV rs, rt	(HI, LO) ← rs / rt	000000	00000	011010
DIVU rs, rt	(HI, LO) ← rs / rt	000000	00000	011011

* DIV执行结果的余数和商分别保存在32位内部寄存器HI和*LO*中，它们要用MFHI和MFLO才能取出内容。
* 指令中确实的rs、rt或rd的值位00000。
* ADDU不会产生溢出例外（exception），而ADD则会，溢出时ADD不会把结果保存在rd中。

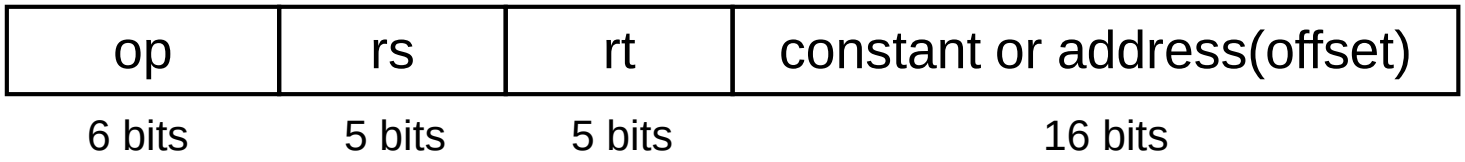
R-format Instructions （2）



format	description	special	shamt	funct
AND rd, rs, rt	rd ← rs AND rt	000000	00000	100100
OR rd, rs, rt	rd ← rs or rt	000000	00000	100101
NOR rd, rs, rt	rd ← rs NOR rt	000000	00000	100111
XOR rd, rs, rt	rd ← rs XOR rt	000000	00000	100110
SLL rd, rt, sa	rd ← rt << sa (logical)	000000	sa	000000
SRL rd, rt, sa	rd ← rt >> sa (logical)	000000	sa	000010
SLT rd, rs, rt	rd ← (rs < rt)	000000	00000	101010
SLTU rd, rs, rt	rd ← (rs < rt)	000000	00000	101011
JR rs	go to an address in a register	000000	00000	001000
MFHI rd	rd ← HI	000000	00000	010000
MFLO rd	rd ← LO	000000	00000	010010

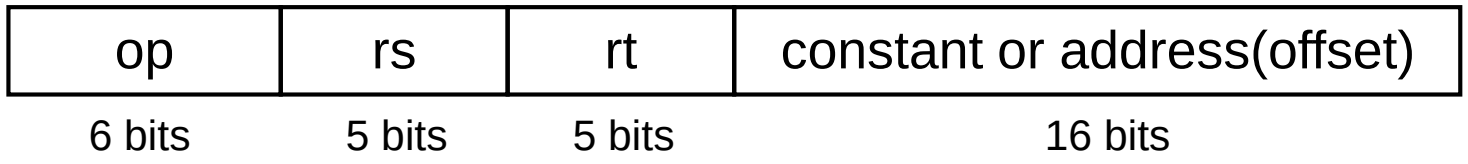
* mthi/mtlo: rd ← HI/LO

I-format Instructions（1）



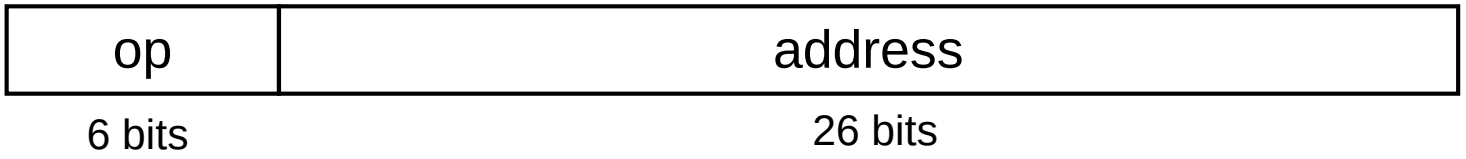
format	description	op
ADDI rt, rs, immediate	rt ← rs + immediate	001000
ADDIU rt, rs, immediate	rt ← rs + immediate	001001
ANDI rt, rs, immediate	rt ← rs AND immediate	001100
ORI rt, rs, immediate	rt ← rs or immediate	001101
XORI rt, rs, immediate	rt ← rs XOR immediate	001110
SLTI rt, rs, immediate	rt ← (rs < immediate)	001010
SLTIU rt, rs, immediate	rt ← (rs < immediate)	001011
BEQ rs, rt, offset	if rs = rt then branch （PC←PC+offsetX4）	000100
BNE rs, rt, offset	if rs ≠ rt then branch （PC←PC+offsetX4）	000101

I-format Instructions（2）



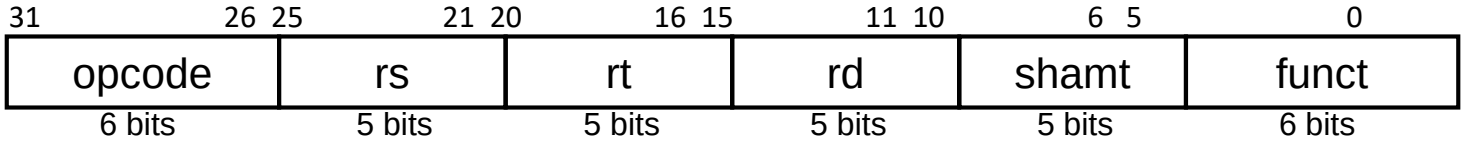
format	description	op
LW rt, offset(base)	rt ← memory[base(rs)e+offset]	100011
SW rt, offset(base)	memory[base(rs)+offset] ← rt	101011
LB rt, offset(base)	rt ← memory[base(rs)+offset] （8位，进行有符号扩展）	100000
LBU rt, offset(base)	rt ← memory[base(rs)+offset] （8位，进行无符号扩展）	100100
LH rt, offset(base)	rt ← memory[base(rs)+offset] （16位，进行有符号扩展）	100001
LHU rt, offset(base)	rt ← memory[base(rs)+offset] （16位，进行无符号扩展）	100101
SB rt, offset(base)	memory[base(rs)+offset] ← rt （保存最低有效字节）	101000
SH rt, offset(base)	memory[base+offset] ← rt (half word) （保存最低有效16位）	101001
LUI rt, immediate	rt ← immediate(16bits) + 16's 0 （把16位立即数送入高16位）	001111
LL rt, offset(base)	rt ← memory[base(rs)+offset] （Load and Link）	110000
SC rt, offset(base)	if atomic_update （Store Conditionally） then memory[base(rs)+offset] ← rt, rt ← 1 else rt ← 0	111000

J-format Instructions

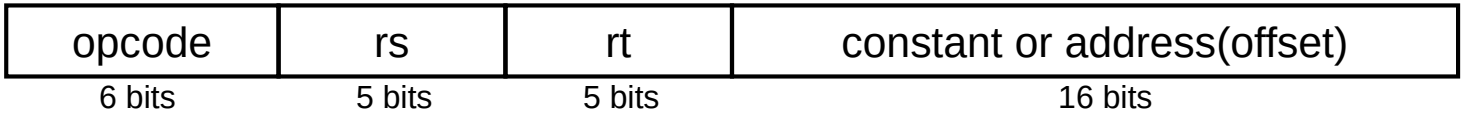


format	description	op
J target	To branch within the current 256 MB-aligned region	000010
JAL target	To execute a procedure call within the current 256 MB-aligned region	000011

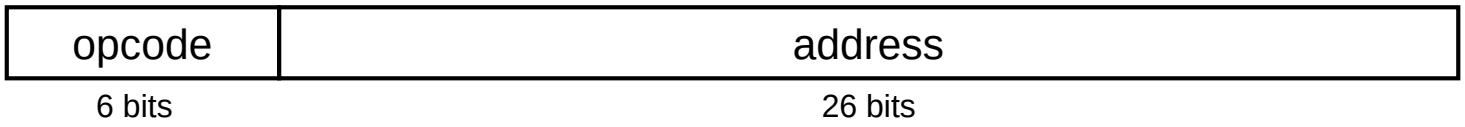
R-format Instructions



I-format Instructions



J-format Instructions



附录4、MIPS汇编语言程序

- Mars汇编源程序文件以.s 或.asm 为后缀。
  - 汇编源程序文件由数据段和代码段两部分组成。
  - 数据段部分以“.data”开始，用于声明变量名字。这些变量在主存中会创建对应的空间。
  - 代码段部分以“.text”开始，包含由指令构成的程序。这个程序以标号main的语句开始，在结束点应该调用exit system call系统调用的功能。
  - 程序中可以加入注释。注释行以“#”开始。
- 运行Mars
  - 先下载和安装Java（要求9.0以上版本，好像8.0也可以用）
  - 然后运行
    C:>java -jar Mars4_5.jar

- Mars汇编语言程序结构

```
.data        # 数据段
.ascii string        # 字符串 （不是0结尾）
.asciiz string       # 以空字符null结束  （末尾自动加0）
.word w1,w2,...      # 字类型（32 位，4 个字节）
.half h1,h2,...      # 半字类型（ 16 位，2 个字节）
.byte b1,b2,...      # 字节类型（ 8 位，1 个字节）
.float f1,f2,...     # 浮点数类型（ 32 位，4 个字节）
.double d1,d2,...    # 双精度浮点数（ 64 位，8 个字节）
.space n ;       # 空格符（n 个字节空间）
.align 2         # 对齐: 0-字节 1-半字 2-字 3-双字
.text        # 代码段
.globl main # 定义main 为全程量(入口标号)
main:
# MIPS 指令和伪指令的组合。                    16进制数: 0x12ef
# 程序结束
```

- 数据段定义示例

```
.data          # put things into the data segment...
  hello_msg: .asciiz  "Hello World\n"

.text          # put things into the text segment...
main:
  la $a0, hello_msg   # load the addr of hello_msg into $a0.
  li $v0,   4      # 4 is the print_string syscall.
  syscall      # do the syscall.
  li $v0, 10       # 10 is the exit syscall.
  syscall      # do the syscall.
```

hello_msg的以下两种方式的定义也一样：

```
hello_msg: .ascii "Hello"    # The word "Hello"
        .ascii " "     # the space.
           .ascii "World"    # The word "World"
           .ascii "\n"   # A newline.
        .byte 0        # a 0 byte.
```

```
hello_msg: .byte 0x48 # hex for ASCII "H"
        .byte 0x65 # hex for ASCII "e"
        .byte 0x6C # hex for ASCII "l"
        .byte 0x6C # hex for ASCII "l"
        .byte 0x6F # hex for ASCII "o"
          ... # and so on...
        .byte 0xA # hex for ASCII newline
        .byte 0x0 # hex for ASCII NUL
```

- Mars的系统调用

利用Mars的系统调用syscall可以实现命令行的输入和输出。

```
# 从键盘输入一整数并存放在$v0中
li      $v0,    5          # 功能号
syscall                 # 系统调用

# 将输出$a0中的整数
li      $v0,     1
addi    $a0,     $zero, 20
syscall

# 输出一个字符串
msg: .asciiz "Hello world!"
...
li      $v0,    4
la      $a0,    msg
syscall

# 退出系统
li      $v0,    10
syscall
```

- 伪指令

没有直接对应一条机器指令或只供汇编器使用的汇编指令称为伪指令。li、move、la和bgt为伪指令, 它们可以转成一条或多条机器指令。

```
.text
main:
      li     $s1,6       # addiu
      move     $t1,$s1    # addu
loop:
      li     $v0,  4
      la     $a0, msg   #lui, ori
      syscall
      addi $t1,$t1,-1
      bgt   $t1,2,loop  # addi,slt,bne

      li $v0, 10   # 返回系统
      syscall
.data
  .opnum word
  .space 16
msg:
  .asciiz "hello!\n"
```

运行结果:

```
hello!
hello!
hello!
hello!
```

例子:

```
##############################  counter ############################
# 计数程序：用户输入一个整数n，则程序一次输出从1到n的整数，
#             输出一个换行一次。
#################################################################
.text
.globl main
error:                                  # 错误处理
    li      $v0,    4                   # 打印字符串(功能号：4)
    la      $a0,    errormsg            # 取字符串errormsg首地址->$a0
    syscall                             # 系统调用
    j get                               # 转重新输入
main:
    addi    $s0,    $zero,   21         # 初始化 $s0=21,规定输出的数据小于这个数
get:
    li      $v0,    4                   # 打印字符串，输出
    la      $a0,    str1
    syscall
    li      $v0,    5                   # 从键盘输入一整数并存放在$v0中
    syscall
    slt     $s1,    $v0,    $s0         # 输入整数在范围内(1~20)，则$s1=1，否则$s1=0
    beq     $s1,    $zero, error        # 如果$s1=0(超出范围)，则转error出错处理
    blez    $v0,    error               # 如果输入的整数$v0<=0，则转error出错处理
    move    $t0,    $v0                 # 将输入的数（上界）保存在$t0
    add     $t1,    $0,   $0            # 初始化$t1=0,初始化计数器
```

要从main运行必须勾上Mars菜单settings/init ... 'main'

```
loop:
    addi    $t1,    $t1,    1           # $t1<-$t1+1
    li      $v0,    1                   # 打印整数，输出一整数
    move    $a0,    $t1                 # 将输出的数据保存到$a0，为输出做准备
    syscall                             # 系统调用

    li      $v0,    4                   # 换新一行输出
    la      $a0,    nline               # 换行符，"\n"
    syscall

    slt     $t2,    $t1,    $t0         # 如果$t1<$t0，$t2=1，否则，$t2=0
    bnez    $t2,    loop                # 如果$t2!=0，转loop，继续输出
    li      $v0,    10                  # 否则，退出，返回系统
    syscall
.data
   str1:
       .asciiz "please give an integer from 1 to 20: "
   errormsg:
       .asciiz "out of range(1 to 20)\n" # 字符串定义，以"00"字符作为终止符结束
   nline:
       .asciiz "\n"
```