指令系统及MIPS（基于MARS）

汇编语言是直接面向内存以及寄存器的语言，他相当于是高级语言的转化（翻译）

寄存器

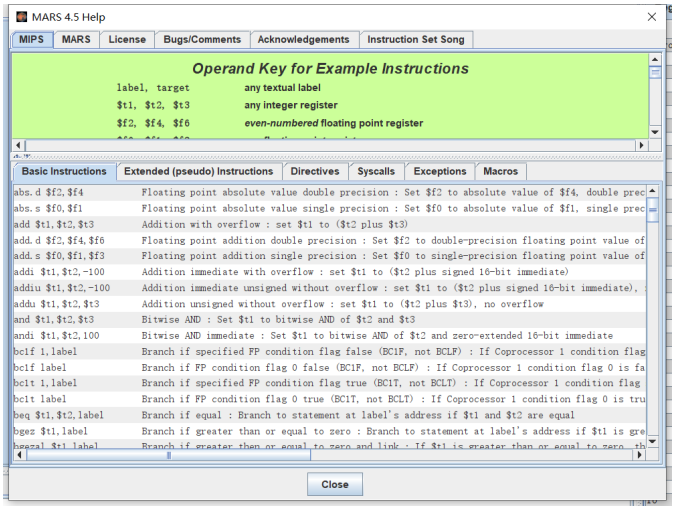
- MIPS中一共有32个通用寄存器
- 通用寄存器标志由$符开头，寄存器表示有两种方式
  - 直接使用该寄存器对应的编号，例如：从0到31
  - 使用对应的寄存器名称，例如：$t1$,sp
- 当然每个寄存器都有其特定的作用，有相应的规范

寄存器编号	助记符	用途
0	zero	值总是为 0
1	at	（汇编暂存寄存器）一般由汇编器作为临时寄存器使用。
2-3	v0-v1	用于存放表达式的值或函数的整形、指针类型返回值
4-7	a0-a3	用于函数传参。其值在函数调用的过程中不会被保存。若函数参数较多，多出来的参数会采用栈进行传递
8-15	t0-t7	用于存放表达式的值的临时寄存器; 其值在函数调用的过程中不会被保存。
16-23	s0-s7	保存寄存器; 这些寄存器中的值在经过函数调用后不会被改变。
24-25	t8-t9	用于存放表达式的值的临时寄存器; 其值在函数调用的过程中不会被保存。当调用位置无关函数 (position independent function) 时，25 号寄存器必须存放被调用函数的地址。
26-27	kt0-kt1	仅被操作系统使用。
28	gp	全局指针和内容指针。
29	sp	栈指针。
30	fp 或 s8	保存寄存器（同 s0-s7）。也可用作帧指针。
31	ra	函数返回地址。

如有疑问可浏览《计算机组成与系统结构》中5.4.1节

MIPS指令

- 指令的一般性语法格式：1个操作符，3个操作数（当然这只是一般形式，想要更多的了解，可以浏览Mars的**Help**中介绍或是浏览《计算机组成与系统结构》中的5.1节、5.2节、5.3.2节以及5.4.1节）



- mips中每一行一条指令，一条指令只有一个操作

  假设：x⟷$s0，a⟷$s1，b⟷$s2，c⟷$s3，d⟷$s4

  C语句：x = (a + b) - (c + d) ;

  MIPS汇编程序片段

  示例：执行序

```
1  add $t1, $s3, $s4     # t1 = c+d
2  add $t2, $s1, $s2     # t2 = a+b
3  sub $s0, $t2, $t1     # a = (a+b)-(c+d)
```

  MIPS汇编程序　　注释

  - $t1，$t2：临时变量寄存器

- 指令中出现的常量数值被称为**立即数**（有的操作符对应的操作数有的是常数：addi，lw，sw等，具体可以浏览**Help**）
- load指令和store指令（如果有疑问可以浏览《计算机组成与系统结构》中的5.2.3节）

  由于MIPS只能对寄存器与立即数进行运算，因此必须有特定的数 据传输指令实现主存单元与寄存器的数据交换

  语法格式

  ```
  op reg, off(base)
  ```

  - reg：写入或读出的寄存器
  - base：存储基地址的寄存器
    - 该寄存器的作用就是指针
    - 由于是地址，因此base的值被作为无符号数
  - off：以字节为单位的偏移
    - off是立即数，可正可负

  读写的存储单元的实际地址=base+off

  这种寻址方式被称为：基地址+偏移

  该寻址方式可以表示任意某个存储单元的地址

  - LOAD类指令：主存单元→寄存器
  - STORE类指令：寄存器→主存单元

  并且可以区分一下lw，lb，lh，sw，sb，sh这类的区别

- 跳转指令（用来实现判断以及循环）
  - j指令
    - j只能单独跳转到相应的标志位，且跳转范围不如jar，jalr等指令
      - 循环的例子（可浏览《计算机组成与系统结构》中5.4.3节）

    loop:

    xxxx

    xxxxx

    j loop

    - 这里的跳转需要注意

      ra寄存器，他会记录每一次跳转之后的$PC+4$值。(**这里的PC可以简单理解为当前程序执行的代码，$+4$就是下一条代码的位置**)

      ra的值（jr $ra是返回记录的相应位置）
  - b指令
    - b类指令就是通过相应的寄存器判断是否符合跳转条件，符合就跳转到相应的**标志**上去（后文有讲标志）
      - 选择结构可参考浏览《计算机组成与系统结构》中5.4.2节

    例：beq $s0$,t0,loop（更多b类指令请浏览**Help**）

常见指令示例表（如果有疑问可以浏览《计算机组成与系统结构》中的5.2.4节）

表 5.4 MIPS 汇编语言示例列表

类别	指 令	汇编举例	含 义	备 注
算术运算	add	add $s1,$s2,$s3	$s1=$s2+$s3	三个寄存器操作数
	subtract	sub $s1,$s2,$s3	$s1=$s2-$s3	三个寄存器操作数
存储访问	load word	lw $s1,100($s2)	$s1=Memory[$s2+100]	从内存取一个字到寄存器
	store word	sw $s1,100($s2)	Memory[$s2+100]=$s1	从寄存器存一个字到内存
逻辑运算	and	and $s1,$s2,$s3	$s1=$s2 & $s3	三个寄存器操作数,按位与
	or	or $s1,$s2,$s3	$s1=$s2 | $s3	三个寄存器操作数,按位或
	nor	nor $s1,$s2,$s3	$s1=~($s2 | $s3)	三个寄存器操作数,按位或非
	and immediate	andi $s1,$s2,100	$s1=$s2 & 100	寄存器和常数,按位与
	or immediate	ori $s1,$s2,100	$s1=$s2 | 100	寄存器和常数,按位或
	shift left logical	sll $s1,$s2,10	$s1=$s2≪10	按常数对寄存器逻辑左移
	shift right logical	srl $s1,$s2,10	$s1=$s2≫10	按常数对寄存器逻辑右移
条件分支	branch on equal	beq $s1,$s2,L	if($s1==$s2) go to L	相等则转移
	branch on not equal	bne $s1,$s2,L	if($s1!=$s2) go to L	不相等则转移
	set on less than	slt $s1,$s2,$s3	if($s2<$s3) $s1=1; else $s1=0	小于则置寄存器为 1,否则为 0,用于后续指令判 0
	set on less than immediate	slti $s1,$s2,100	if($s2<100) $s1=1; else $s1=0	小于常数则置寄存器为 1,否则为 0,用于后续指令判 0
无条件跳转	jump	j L	go to L	直接跳转至目标地址
	jump register	jr $ra	go to $ra	过程返回
	jump and link	jal L	$ra=PC+4; go to L	过程调用

伪指令

MIPS定义了一组伪指令，从而使得程序更可读更易编写，例：li $t0,1就是令t0寄存器为1的意思。但是伪指令不是真正的指令，当被编译的时候会转成相应的指令。

Address	Code	Basic	
0x00003000	0x24080001	addiu $8, $0, 0x00000001	1: li $t0, 1

常见的伪指令

move，la，li等，相应的伪指令也在**Help**中有解释

程序结构

程序结构包括数据段，代码段

- 数据段以**.data**为开始标志
- 代码段以**.text**为开始标志

基本模板

```
# Comment giving name of program and description of function
# 说明下程序的目的和作用（其实和高级语言都差不多了）
# Template.s
#Bare-bones outline of MIPS assembly language program

            .data       # variable declarations follow this line              # 数据变量声明
                        # ...

            .text       # instructions follow this line
                        # 代码段部分
main:                   # indicates start of code (first instruction to execute)                    # 主程序
                        # ...
```

可以注意到这段模板中的main这个标志。在mips的代码段中在循环中需要判断条件，需要跳转的时候，函数调用的时候都需要标志的指示，来判断相应跳转的位置。

示例：

```
.text
  beq $s0,$t0,is_not_prime
  li $t0,2
loop:
  beq $t0,$s0,is_prime
  beq $s1,$zero,is_not_prime
  addi $t0,$t0,1
  j loop
is_not_prime:
is_prime:
```

数据声明

格式：

```
name:                   storage_type    value(s)
变量名：（冒号别少了）      数据类型          变量值
```

示例：

```
.data
   pre:.space 400
   aft:.space 400
   str_enter: .asciiz"\n"
```

相关介绍：

1、.space就是在内存中开辟了一个相应大小的空间。就如示例中400就对应能存100个字（1个字四个字节，一个字节8位。内存是一个字一个字存储）

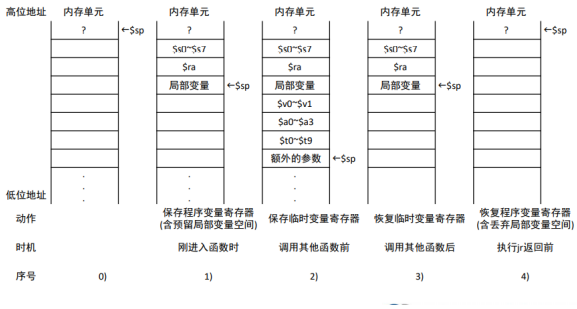
2、.word就是声明一个 word 类型的变量 name, 同时给其赋值为value

3、.byte并不是按字去申请空间，而是按照字节（与.space要区别）

4、.asciiz就是定义相应的字符串，便于后续的输出（可以思考一下ascii和asciiz的区别）

栈

- 由于寄存器只有32个，因此编译器绝大多数情况下不可能把函数需要的所有局部变量都分配在寄存器



栈指针使用的顺序是从上往下存储。

栈指针$sp一般在函数调用的时候会用到，在每次调用函数的时候，要将相应的ra寄存器的值以及相应保存当前状态的寄存器的值存入到栈中，在需要返回的时候，又将相应的值取出，进行返回。

如有疑问可浏览《计算机组成与系统结构》中5.4.4节

递归

> 首先需要理解递归本身的性质或运行流程，在此基础上进行mips编写

- 递归流程：判断是否终止递归、进入终止输出并回溯 或 进入包含递归模块的核心部分
  - 递归模块重点：递归终止条件、递归进入变量存储、递归回溯
  1. 递归终止条件
  - 即递归终止的情况
  1. 递归进入变量存储
  - 递归中参数的存储
    - 进入新一轮递归前的参数
  - 每个递归跳转指令的下一指令的地址的存储
    - $ra中存储着jal指令的下一指令,即本次递归结束时的下一条指令地址

```
sw $ra, 0($sp)      # store address
addi $sp, $sp, -4
sw index, 0($sp)
addi $sp, $sp, -4   # store index
sw i, 0($sp)
addi $sp, $sp, -4   #store i
```

  1. 递归回溯
     - 按照存储顺序的逆序将本递归的参数取回来继续完成本次递归函数

```
sw $ra, 0($sp)      # store address
addi $sp, $sp, -4
sw index, 0($sp)
addi $sp, $sp, -4   # store index
sw i, 0($sp)
addi $sp, $sp, -4   #store i
```

     - 递归回溯后的操作:

```
    bne $t4, 8, nxt
    li $s1, 0
nxt:
    add $s2, $s1, $s0
    sw $s2, fib_array($t4)
    move $s1, $s0       #前前一个
    move $s0, $s2       # $s0 前一个
    jr $ra
```

系统调用

- 系统调用可以简单的理解为发挥执行输入，输出，结束程序的作用（具体也浏览**Help**）
- 参数所使用的寄存器：$v0$，a0，$a1
- 返回值使用： $v0

详细用法：

Service	Code in $v0
print integer	1
print float	2
print double	3
print string	4
read integer	5
read float	6
read double	7
read string	8
sbrk (allocate heap memory)	9
exit (terminate execution)	10

```
e.g. Print out integer value contained in register $t2
        li  $v0, 1
        move    $a0, $t2
        syscall

e.g.   Read integer value, store in RAM location with label int_value (presumably declared in data section)
        li  $v0, 5
        syscall     #输入一个整数。并把整数值赋给$v0寄存器

e.g.   Print out string (useful for prompts)
        .data
               string1      .asciiz "Print this.\n"
        .text
               li   $v0, 4
               la   $a0, string1        #将要打印的字符串地址赋值  $a0 = address(string1)
               syscall

e.g. To indicate end of program, use exit system call
        li  $v0, 10             # system call code for exit = 10
        syscall
```

课下题目分析

最大公约数

- 考察程序基本结构是否理解（输入，输出，判断条件）
- 考察通过用寄存器保存相应的值并进行计算
- 考察对于相关指令的使用
- 循环的建立

字符串逆置

- 考察对于字符串的读写（参考syscall里的用法）
- 难度较为简单，不用想复杂了

汉诺塔

- 考察对于栈的了解
- 考察对于递归的理解
- 考察jal,jr，ra以及sp的理解与使用
- 难度较为复杂，不过在了解完递归后，会非常清晰

注：《计算机组成与系统结构》可在课程网站进行下载