第二章 汇编语言

华中科技大学
电子信息与通信学院
School of Electronic Information and Communications

学习目标

- MIPS寻址方式原理
- 程序的编译、链接、装载过程
- 程序的内存映像
- 编写简单MIPS汇编语言程序
- MIPS宏汇编程序设计
- MIPS仿真器使用
    - QTSPIM
    - MARS

2.9寻址原理

- 寻址指微处理器获取其操作对象，包括数据和指令存储位置的方式
  - 操作数寻址
  - 指令寻址

操作数寻址

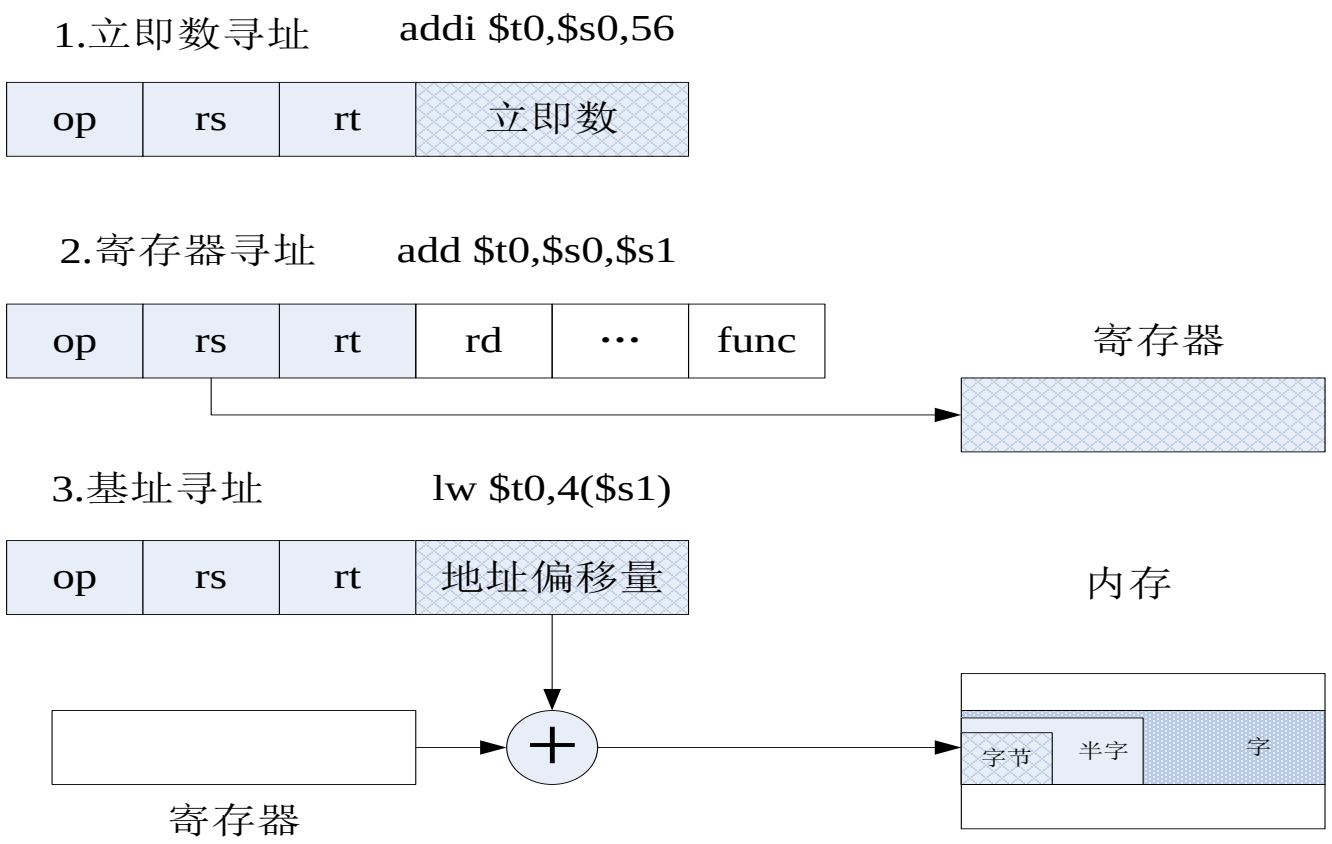
- 立即数寻址
  - 立即数直接编码在机器指令中，因此与指令存储在一起

lui $t0,32

立即数

- 寄存器寻址
  - 指令中的操作数为寄存器时，称为寄存器寻址
  - add $s0,$t0,$zero
- 基址寻址
  - 通过某个寄存器加上一个立即数来指示该内存单元的地址，这种方式就称为基址寻址
  - lw $t0,4($sp)

操作数寻址数据存放位置示例



指令寻址

- PC相对寻址
  - 要寻找的下一条指令的地址由PC寄存器的值和一个相对偏移量构成
  - beq $t2,$zero,L2;

6位操作码编码	5位第一寄存器编码	5位第二寄存器编码	16位偏移量编码

```
        beq $t2,$zero,L2#如果字符的值为0则转移到L2标号处
        addi $s0,$s0,1#否则修改索引指向到下一个字符
        j L1#返回到循环处
L2:     lw $s0,0($sp)#恢复$s0的值
        addi $sp,$sp,4#恢复$sp的值
        jr $ra #返回
```

跳转范围为16位有符号数所能表示的范围*4，即$(-2^{16} \sim 2^{16}-1)*4$

新PC的值=PC原始值+16位立即数*4

000100	01010	00000	0000 0000 0000 0010

偏移指令条数

- 伪直接寻址

指令编码

6位操作码编码	26位跳转地址

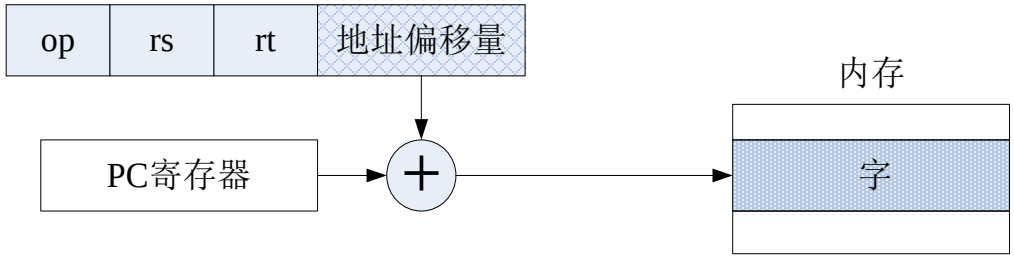
新的PC值

PC的高4位	26位跳转地址	00

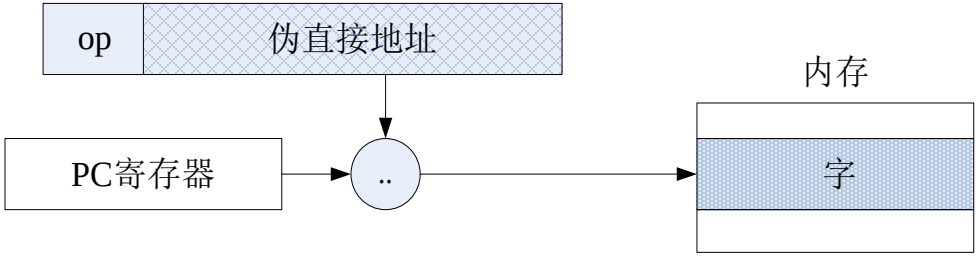
- 寄存器间接寻址
  - 利用寄存器保存指令内存地址，如指令jr $ra
  - 直接将指令中所表示的寄存器值赋给$pc

指令寻址方式原理

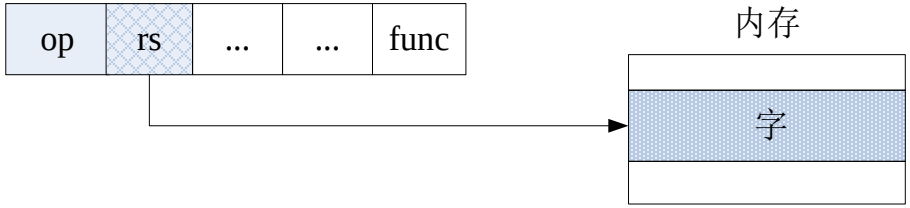
1.PC相对寻址　　beq $s0,$s1,L2



2.伪直接寻址　　J L2



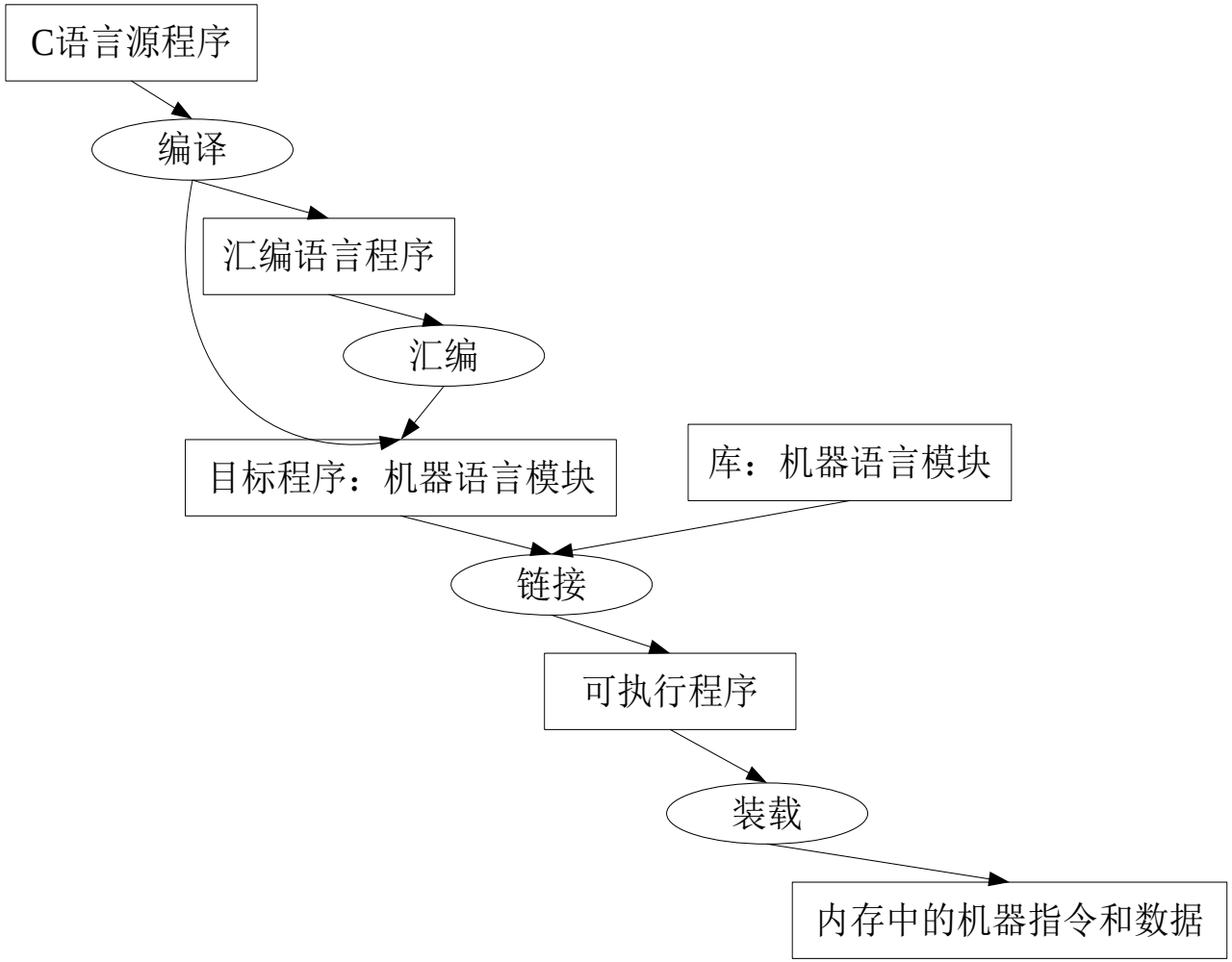
3.寄存器间接寻址　　jr $ra



2.10编译、汇编、链接、装载过程

- 目标程序：是指经汇编程序翻译而获得的机器指令序列的程序。此程序仍然不能被计算机执行。
- 链接：是指将各个相互关联的目标程序（包括库文件）连接起来组成可被计算机直接执行的程序的过程。
- 可执行程序：可以被系统软件调用并被计算机直接执行的程序。
- 程序装载：是指将可执行程序装载到内存，并实现相关内存、寄存器等初始化工作，且使得微处理器跳转到可执行程序入口的过程。

高级语言源程序到计算机执行所经历的过程



已知程序1引用了程序2的公共变量x，程序2引用了程序1的公共变量y,同时程序1调用了程序2的公共子程序B，程序2调用了程序1的公共子程序A

目标文件头			
	名称	过程A	
	代码段大小	0x100	
	数据段大小	0x20	
代码段	地址	机器指令	
	0	lw $a0,?($gp)	
	4	jal 0	
	。。。	。。。	
数据段	0	(y)	
	。。。	。。。	
重定位信息	地址	指令类型	依赖关系
	0	lw	x
	4	jal	B
符号表	符号	地址	
	x	--	
	B	--	

目标文件头			
	名称	过程B	
	代码段大小	0x200	
	数据段大小	0x30	
代码段	地址	机器指令	
	0	sw $a0,?($gp)	
	4	jal 0	
	…	…	
数据段	0	(x)	
	…	…	
重定位信息	地址	指令类型	依赖关系
	0	sw	y
	4	jal	A
符号表	符号	地址	
	y	--	
	A	--	

假定将可执行程序的代码段起始地址设定为0x40 0000,数据段的起始地址设定为0x1000 0000,$gp的值设定为0x0FFF 8000。

可执行文件头部		
	代码大小	0x300
	数据大小	0x50
代码段	地址	指令
	0x 0040 0000	lw $a0, 0x8020($gp)
	0x 0040 0004	jal 0x10 0040
	…	…
	0x 0040 0100	sw $a0, 0x8000($gp)
	0x 0040 0104	jal 0x10 0000
	…	…
数据段	0x 1000 0000	y
	…	…
	0x 1000 0020	x
	…	…

2.11 汇编程序设计

- 宏汇编语言有3类基本指令：符号指令、伪指令和宏指令。
  - 伪指令只为汇编程序将符号指令翻译成机器指令提供信息，没有与它们对应的机器指令
  - 把一个指令序列定义为一条宏指令

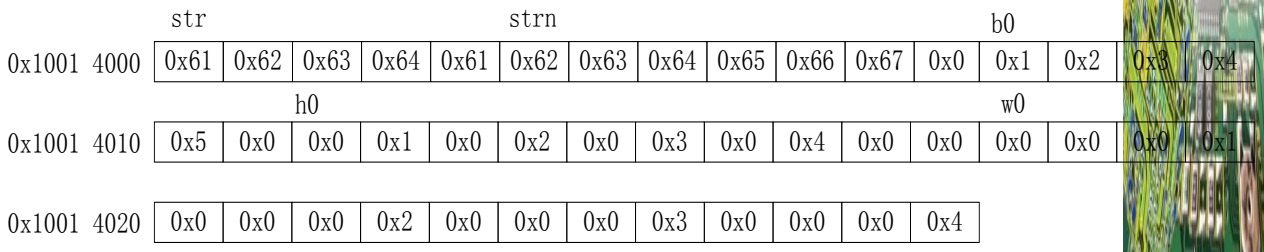
spim MIPS仿真器汇编伪指令

- .align n 紧接着的内存地址以2^n的地址实现字节边界对齐.
  - 如.align 2 表示下一个内存地址以字实现边界对齐;
  - 而.align 0 则关闭.half, .word, .float 以及.double等伪指令的自动边界对齐，直到碰到下一个.data 或 .kdata。
- .ascii str 在内存中存储str字符串，不包含字符串结束符null。
- .asciiz str在内存中存储str字符串，且包含字符串结束符null。
- .byte b1,..., bn 在n个连续的内存空间中依次存储字节b1,..., bn
- .word w1,..., wn在内存中连续存放n个字w1,..., wn
- .space n 分配n个连续的字节存储空间
- .globl sym 声明全局变量sym，这样sym就可以被外部文件使用
- .data <addr>定义用户数据段，addr为可选参数，addr用来定义用户数据段的起始地址。紧接着定义的内容将存放在用户数据段
- .text <addr>定义用户代码段，addr为可选参数，addr用来定义用户代码段的起始地址。紧接着定义的内容将存放在用户代码段。在spim中紧接着只能为指令。
- .kdata <addr> .ktext <addr>分别定义系统内核数据段和代码段。

- 数据
  - 字符串采用双引号括起来，
  - 特殊字符的定义与C语言的规范基本一致，如：
    - 换行符 \n
    - Tab　\t
    - 引号　\”
  - 数值的不同进制表示与C语言基本一致，
    - 无任何前缀的数为十进制数，
    - 十六进制数以0x开头。

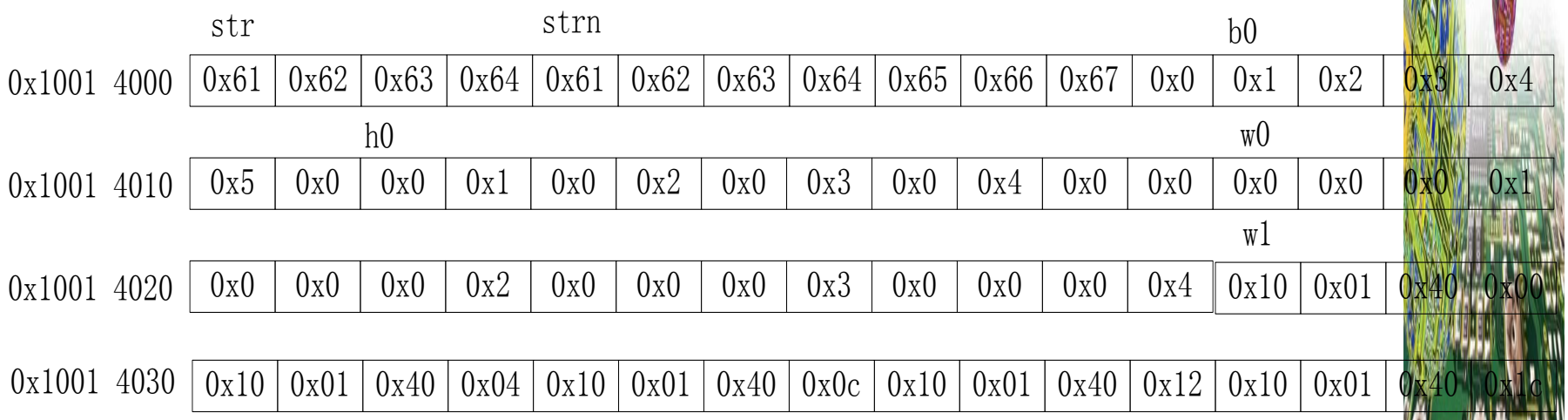
- 数据在内存中的地址通过变量来表示，变量在汇编语言中的定义方式与标号类似，即在定义数据的伪指令前，写上变量名，并且采用冒号隔开。
- 变量与标号的区别在于：
  - 变量表示数据在内存中的地址，
  - 标号表示指令在内存中的地址。

```
.data 0x10014000
.align 2
str: .ascii “abcd”
strn: .asciiz “abcdefg”
b0: .byte 1,2,3,4,5
h0: .half 1,2,3,4
w0: .word 1,2,3,4
```



地址表达式

```
.data 0x10014000
.align 2
str: .ascii “abcd”
strn: .asciiz “abcdefg”
b0: .byte 1,2,3,4,5
h0: .half 1,2,3,4
w0: .word 1,2,3,4
w1: .word str,strn,b0,h0,w0
```



宏指令

- *取变量或标号的地址:*
  - la Rd, Label
- *立即数初始化*
  - li Rd, value

系统功能调用

- 实现键盘输入和显示器输出等人机接口

功能描述	功能号（$v0）	入口参数	出口参数	备注
输出一个十进制整数	1	$a0=需要输出的数值	无	
输出字符串	4	$a0=字符串首地址	无	输出的字符串以字符串结束符标志
从键盘缓冲区读入一个十进制整数	5	无	$v0=输入的整数	回车表示输入结束
从键盘缓冲区读入字符串	8	$a0=保存字符串的首地址 $a1=预留的内存空间的大小	输入的字符的ASCII顺序存放在以$a0开始的连续的内存单元中	回车表示输入结束，实际能输入的字符个数为$a1-1，实际存放在内存中的字符串为用户输入的字符串+回车符0x0a
退出	10	无	无	

系统功能调用步骤为：
- 将功能号赋给$v0;
- 设置好入口参数；
- syscall;

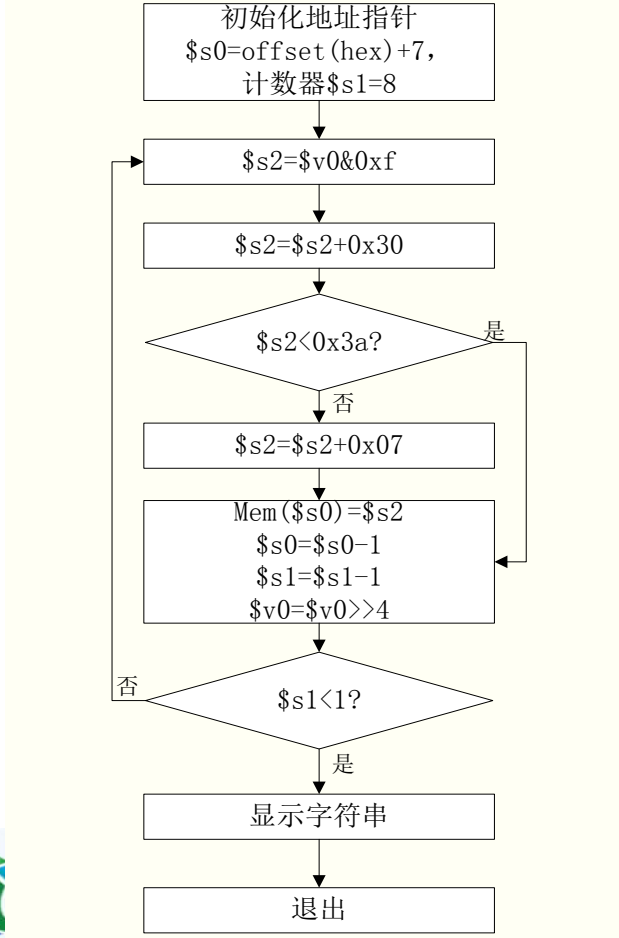
输出整数256,
- li $v0,1 #供能号1赋给$v0
- li $a0,256 #将要显示的数据256赋给入口参数$a0
- syscall

功能调用举例

```
.data 0x10014000
.align 2
str: .ascii "abcd"
strn: .asciiz "abcdefg"
b0: .byte 1,2,3,4,5
.text
main:li $v0,1
        li $a0,0x200
        syscall #输出十进制数据512①
        li $v0,4
        la $a0,str
        syscall #输出字符串str②
        la $a0,strn
        syscall #输出字符串strn③
        li $v0,5
        syscall #输入十进制整数④
        li $v0,8
        la $a0,b0
        li $a1,5
        syscall #输入字符串⑤
        li $v0,10
        syscall #程序结束退出
```

汇编程序设计举例

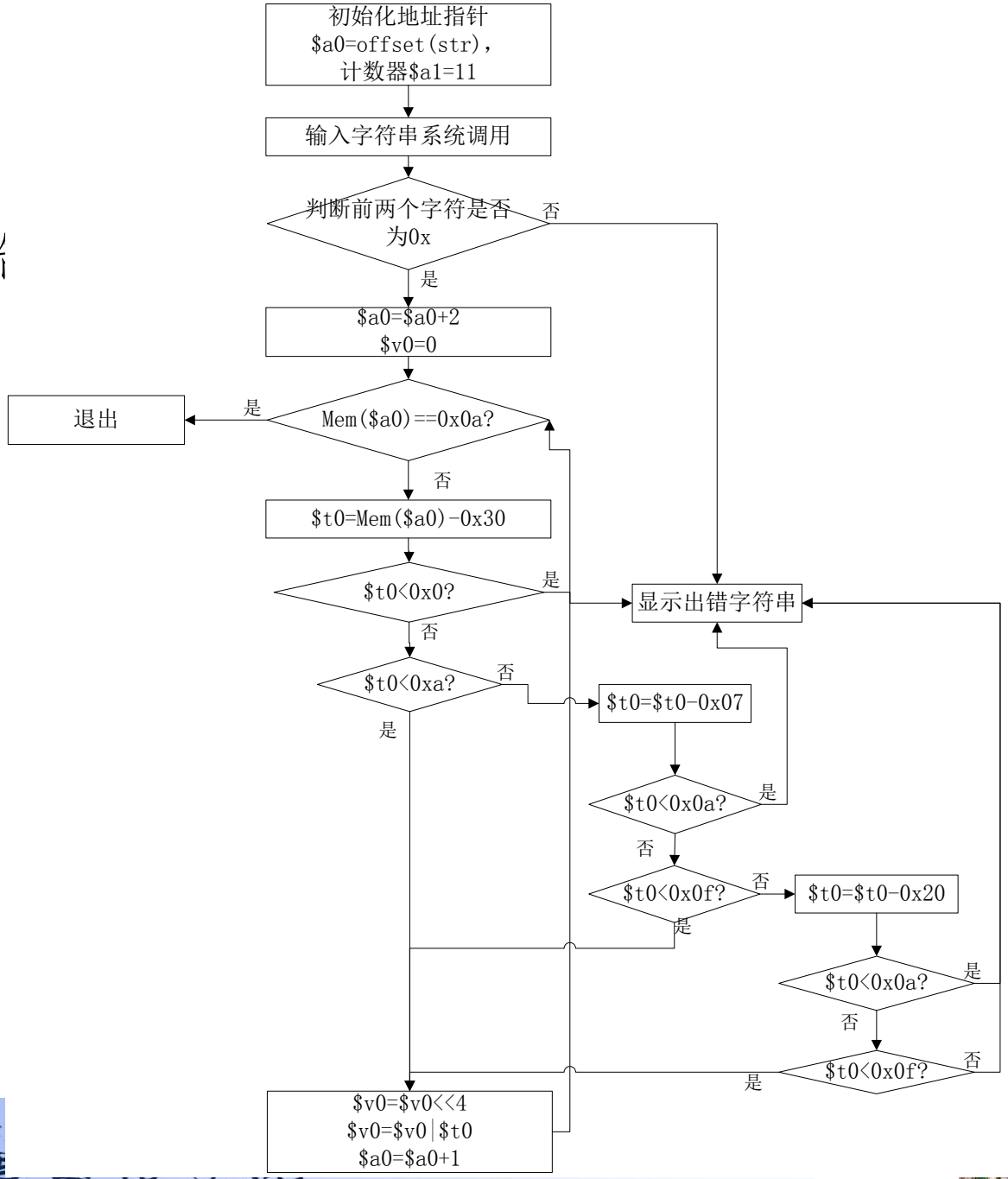
- 显示16进制数
  - 以16进制形式显示$v0寄存器中的32位二进制数




```
.data
str: .byte 0x20,0x30,0x78 #16进制显示前缀
hex: .space 8 #预留8个内存单元
     .byte 0x00 #字符串结束符
.text
main:
                li $v0,0x12AF5678
                la $s0,hex
                addi $s0,$s0,7
                li $s1,8 #初始化
rep:            andi $s2,$v0,0xf #获取低4位
                addi $s2,$s2,0x30 #转换为ASCII码
                slti $t0,$s2,0x3a
                bne $t0,$0,less
                addi $s2,$s2,0x07
less:           sb $s2,0($s0) #存储转换结果
                addi $s0,$s0,-1
                addi $s1,$s1,-1
                srl $v0,$v0,4
                slti $t0,$s1,1
                beq $t0,$0,rep
                li $v0,4 #显示整个转换后的字符
                la $a0,str
                syscall
                li $v0,10 #退出
                syscall
```

输入16进制数

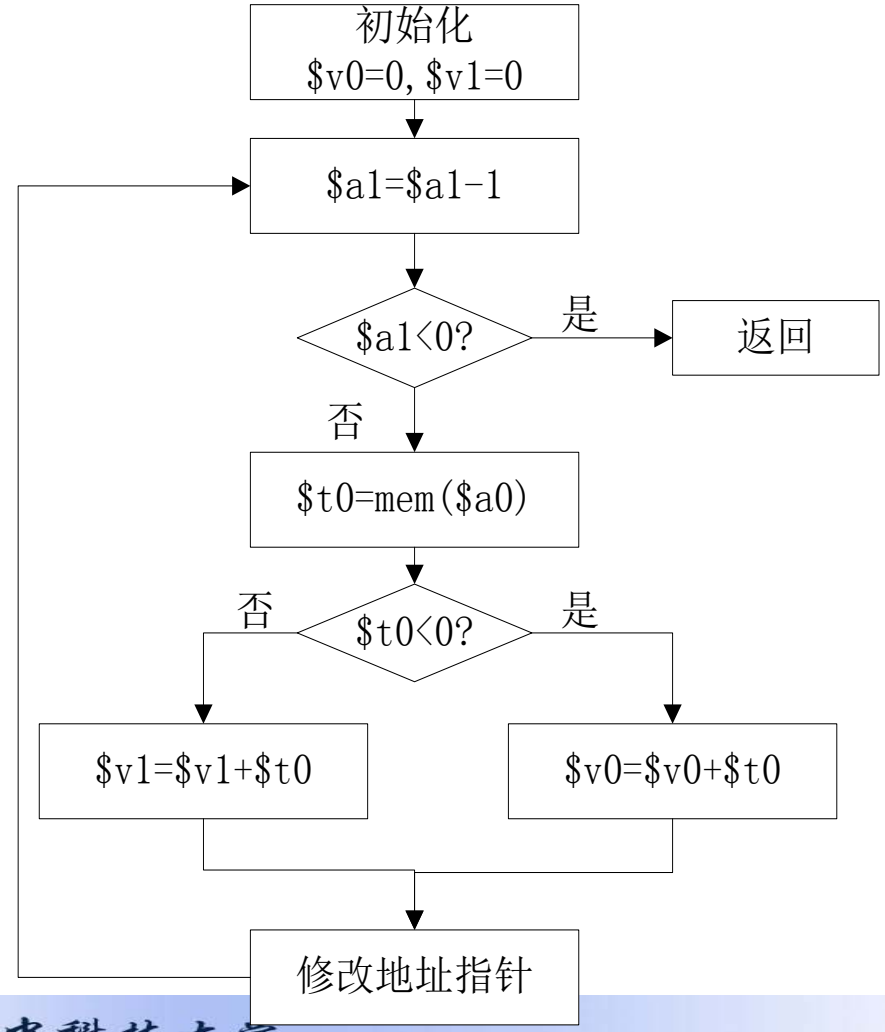
- 从键盘输入16进



子程序设计示例

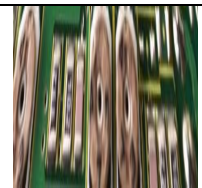
- 计算某数组中所有正数的和和所有负数的和，并分别显示结果。要求采用子程序计算数组中所有正数的和和所有负数的和。
  - 有两个输入参数：地址指针和计数器
  - 两个出口参数：正数的和和负数的和
  - $a0,$a1分别作为地址指针和计数器，$v0,$v1分别传递正数的和和负数的和

求和子程序流程

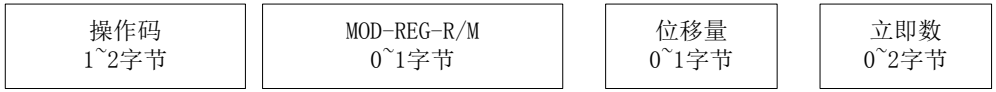

初始化
$v0=0, $v1=0
$a1=$a1-1
$a1<0?
是
返回
否
$t0=mem($a0)
否
$t0<0?
是
$v1=$v1+$t0
$v0=$v0+$t0
修改地址指针

假定该数组为字类型的数据

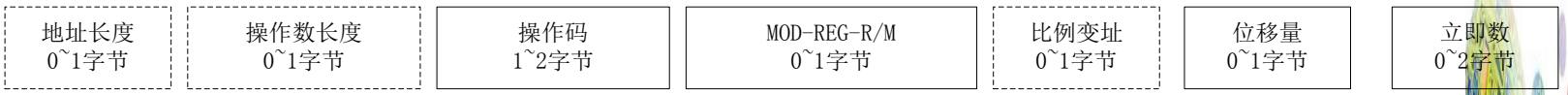
```
.data
array: .word -1,3,4,-5
posi: .asciiz "\nthe sun of positive numbers a
nega: .asciiz "\nthe sun of negative numbers
.text
main:
        li $v0,4
        la $a0,posi
        syscall  #显示字符串posi
        la $a0,array #准备好入口参数
        li $a1,4
        jal sum #调用求和子程序
        add $a0,$v0,$0
        li $v0,1
        syscall #显示正数的和
        li $v0,4
        la $a0,nega
        syscall  #显示字符串nega
        add $a0,$v1,$0
        li $v0,1
        syscall #显示负数的和
        li $v0,10
        syscall #退出
sum:    li $v0,0
        li $v1,0
loop:
        blez $a1, retzz # 如果(a1 <= 0) 则返回
        addi $a1, $a1, -1 # 计数器减1
        lw $t0, 0($a0) # 从数组中获取一个元素
        addi $a0, $a0, 4 # 指向下一个元素
        bltz $t0, negg # 如果是复数跳转到negg
        add $v0, $v0, $t0 # 加到正数的和$v0
        b loop # 重复处理下一个元素
negg:   add $v1, $v1, $t0 # 加到负数的和$v0
        b loop # 重复处理下一个元素
retzz:  jr $ra # 返回
```

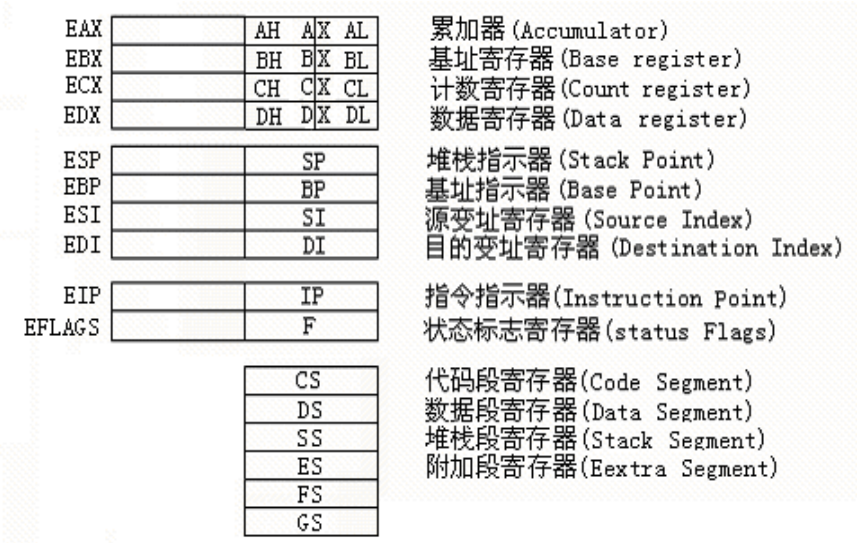
2.12 intel x86微处理器指令集简介



（a）16位指令格式



（b）32位指令格式



寻址方式

寻址类型	指令示例	源	目的
寄存器寻址	MOV AX,BX	BX	AX
立即寻址	MOV CH,3AH	3AH	CH
直接寻址	MOV [1234H]，AX	AX	DS:[1234H]
寄存器间接寻址	MQV [BX]，CL	CL	DS:[BX]
基扯加变扯寻址	MOV [BX+SI],BP	BP	DS:[BX+SI]
寄存器相对寻址	MOV CL，[BX+4]	DS:[BX+4]	CL
基址加变址寻址	MQV ARRAY[BX+51],DX	DX	DS:[ARRAY+BX+51]
比例变址寻址	MOV [EBX+2 * ESI],AX	AX	DS:[EBX+2*ESI]

常用指令

指令		含义
程序控制类指令	jz,jnz	条件跳转，判断z标志位为0否再决定跳转
	jmp	无条件跳转
	call	调用子程序
	ret	子程序返回
	loop	根据ECX的值循环执行一段指令
数据传输	mov	寄存器间，寄存器与存储器间数据传输
	push,pop	栈操作，数据在栈与寄存器间传输
	les	装在内存数据到通用寄存器和ES段寄存器
算术逻辑运算	add,sub	加减运算，不区分符号数和无符号数
	cmp	比较两操作数的大小，不回送结果，仅改变标志位
	shl,shr,rcr	逻辑左移，逻辑右移，带进位循环右移
	cbw	字节（8位）符号数扩展为字（16位）符号数
	test	位与运算，不回送结果，仅改变标志位
	inc,dec	加1，减1
	or,xor	位或，位异或
字符串操作	movs	字符串从一块内存区域传送到另一块内存区域
	lods	字符串中的字符从内存拷贝到EAX寄存器