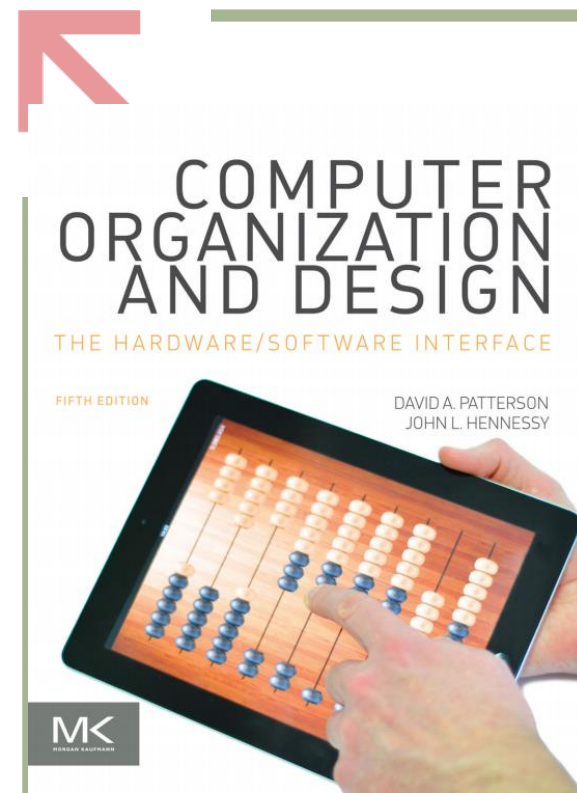
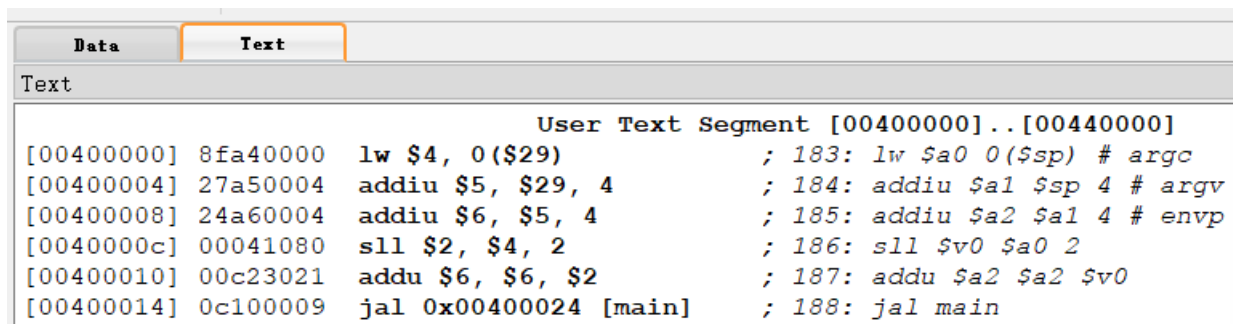


汇编语言的基础原理 与操作流程



机器语言与汇编语言

- ▶ 机器语言：计算机系统内部通信的二进制表示。
- ▶ 汇编语言：机器语言的符号表示。
- ▶ 汇编器，将汇编语言翻译成二进制指令。



The screenshot shows a debugger window with two tabs: 'Data' and 'Text'. The 'Text' tab is selected, displaying the 'User Text Segment [00400000]..[00440000]'. The window contains a list of assembly instructions with their corresponding memory addresses and hexadecimal values.

| Address | Hex Value | Assembly Instruction | Comment |
|------------|-----------|-----------------------|---------------------------------|
| [00400000] | 8fa40000 | lw \$4, 0(\$29) | ; 183: lw \$a0 0(\$sp) # argc |
| [00400004] | 27a50004 | addiu \$5, \$29, 4 | ; 184: addiu \$a1 \$sp 4 # argv |
| [00400008] | 24a60004 | addiu \$6, \$5, 4 | ; 185: addiu \$a2 \$a1 4 # envp |
| [0040000c] | 00041080 | sll \$2, \$4, 2 | ; 186: sll \$v0 \$a0 2 |
| [00400010] | 00c23021 | addu \$6, \$6, \$2 | ; 187: addu \$a2 \$a2 \$v0 |
| [00400014] | 0c100009 | jal 0x00400024 [main] | ; 188: jal main |

机器语言与汇编语言

```
00100111101111011111111111100000
101011111011111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
1010111110100000000000000011000
1010111110100000000000000011100
1000111110101110000000000011100
1000111110111000000000000011000
0000000111001110000000000011001
0010010111001000000000000000001
00101001000000010000000001100101
1010111110101000000000000011100
0000000000000000011110000010010
00000011000011111100100000100001
0001010000100000111111111110111
1010111110111001000000000011000
00111100000001000001000000000000
1000111110100101000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
1000111110111111000000000010100
00100111101111101000000000100000
000000111110000000000000001000
00000000000000000001000000100001
```

图 B-1-2 MIPS 用来计算和打印出 0 ~ 100 的整数的平方和的机器语言代码

```
addiu    $29, $29, -32
sw       $31, 20($29)
sw       $4, 32($29)
sw       $5, 36($29)
sw       $0, 24($29)
sw       $0, 28($29)
lw       $14, 28($29)
lw       $24, 24($29)
multu    $14, $14
addiu    $8, $14, 1
slti     $1, $8, 101
sw       $8, 28($29)
mflo     $15
addu     $25, $24, $15
bne      $1, $0, -9
sw       $25, 24($29)
lui      $4, 4096
lw       $5, 24($29)
jal      1048812
addiu    $4, $4, 1072
lw       $31, 20($29)
addiu    $29, $29, 32
jr       $31
move     $2, $0
```

图 B-1-3 同一个程序的汇编语言版
然而，这个程序的代码没有标记寄存器或者内存地址，也没有包含注释。

Java和汇编的区别

- ▶ 高级语言 vs 符号化的机器语言

- ▶ 处理对象

- ▶ java 可以随便定义变量：int l=1.此时l=1在你的计算机里面。
- ▶ 基于MIPS的汇编中处理的对象必须寄存器，如 Add \$s1,\$zero,1

- ▶ 程序结构

- ▶ Java 对循环，条件语句是直接执行逻辑
- ▶ 汇编里面循环条件的本质是读取地址

.....

流程

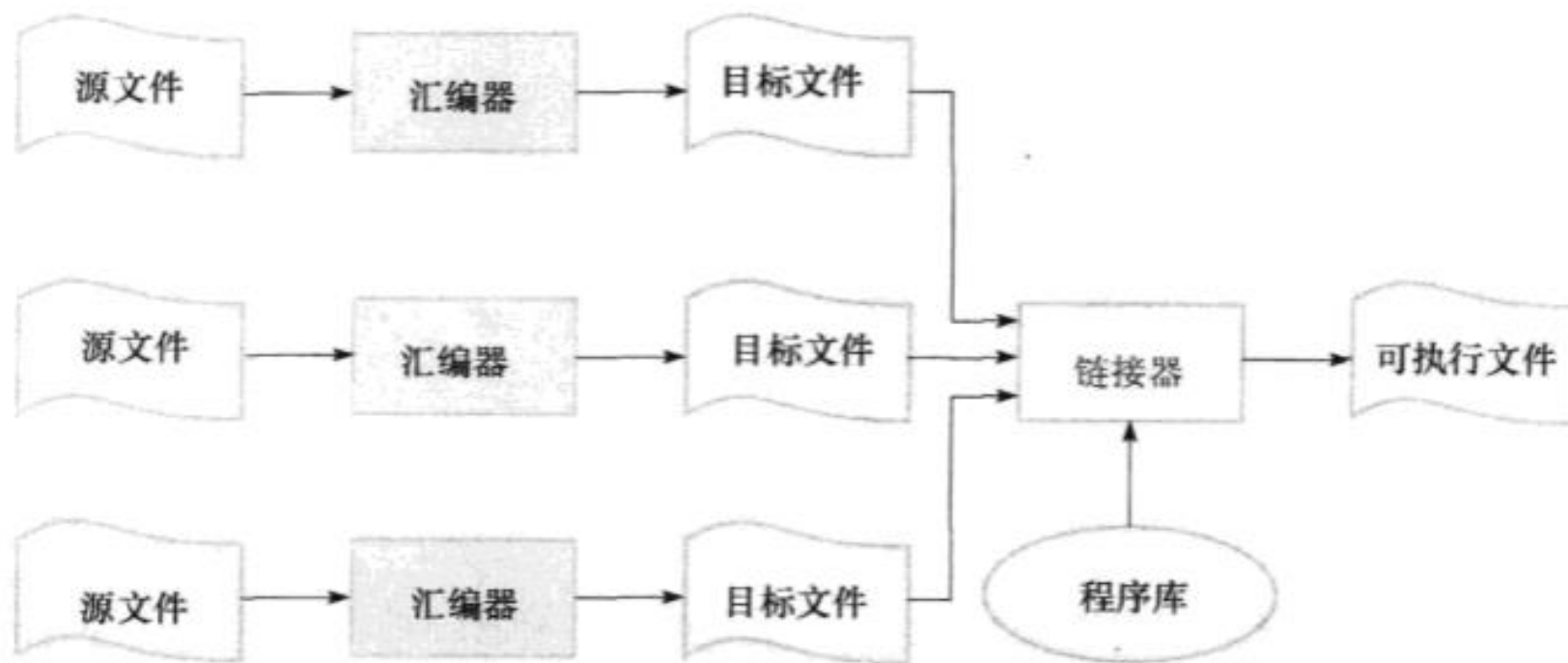


图 B-1-1 产生可执行文件的过程

一个汇编器将一个用汇编语言写的文件翻译成一个目标代码文件，这个目标代码文件又和其他的文件链接组成可执行文件。

汇编程序结构框架

- **数据声明部分**

- 在源代码中，数据声明部分以 `.data` 开始。声明了在代码中使用的变量的名字。同时，也在主存（RAM）中创建了对应的空间。

- **程序代码部分**

- 在源代码中，程序代码部分以 `.text` 开始。这部分包含了由指令构成的程序功能代码。
- 代码以 `main:` 函数开始。`main` 的结束点应该调用 `exit system call`。

- **程序的注释部分**

- 使用 `#` 符号进行注释。每行以 `#` 引导的部分都被视作注释。

数据声明

| name: | storage_type | value(s) |
|-------|--------------|----------|
|-------|--------------|----------|

example

```
var1:      .word    3          # create a single integer:
                                #variable with initial value 3

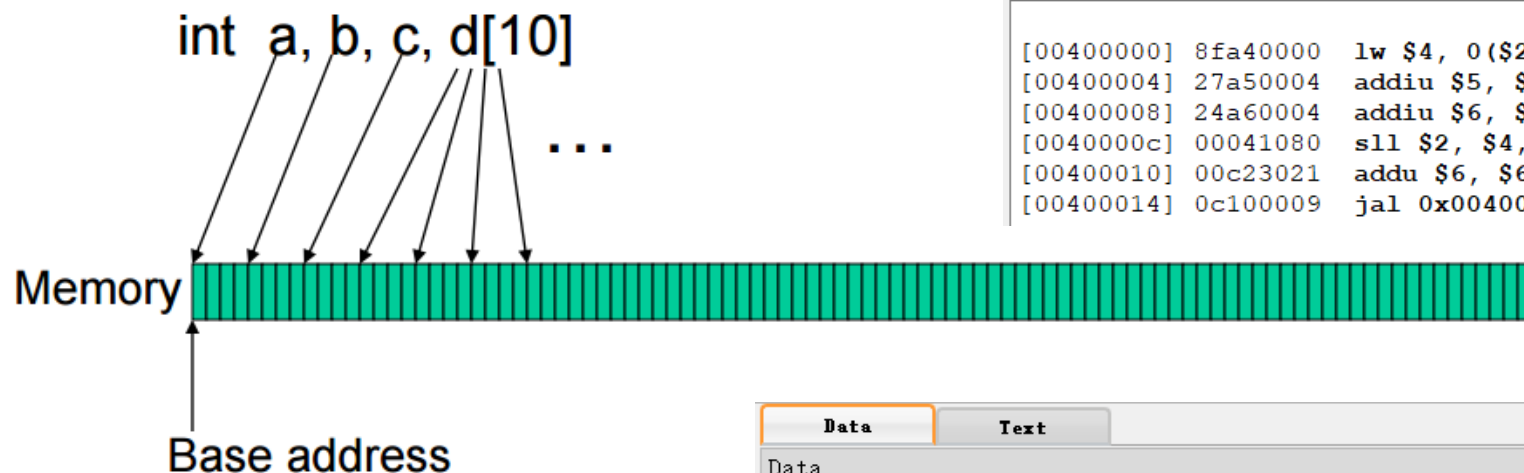
array1:    .byte    'a','b'   # create a 2-element character
                                # array with elements initialized:
                                # to a and b

array2:    .space    40        # allocate 40 consecutive bytes,
                                # with storage uninitialized
                                # could be used as a 40-element
                                # character array, or a
                                # 10-element integer array;
                                # a comment should indicate it.

string1    .ascii "Print this.\n"      #declare a string
```

内存 (memory)

- The compiler organizes data in memory... it knows the location of every variable (saved in a table)... it can fill in the appropriate mem-address for load-store instructions



| Data | Text |
|--|---|
| Text | |
| User Text Segment [00400000]..[00440000] | |
| [00400000] | 8fa40000 lw \$4, 0(\$29) ; 183: lw \$a0 0(\$sp) # argc |
| [00400004] | 27a50004 addiu \$5, \$29, 4 ; 184: addiu \$a1 \$sp 4 # argv |
| [00400008] | 24a60004 addiu \$6, \$5, 4 ; 185: addiu \$a2 \$a1 4 # envp |
| [0040000c] | 00041080 sll \$2, \$4, 2 ; 186: sll \$v0 \$a0 2 |
| [00400010] | 00c23021 addu \$6, \$6, \$2 ; 187: addu \$a2 \$a2 \$v0 |
| [00400014] | 0c100009 jal 0x00400024 [main] ; 188: jal main |

| Data | Text |
|--|---|
| Data | |
| User data segment [10000000]..[10040000] | |
| [10000000]..[1000ffff] | 00000000 |
| [10010000] | 6c6c6568 6e49206f 6e726574 000a7465 h e l l o I n t e r n e t . . |
| [10010010]..[1003ffff] | 00000000 |

MIPS的处理数据的过程

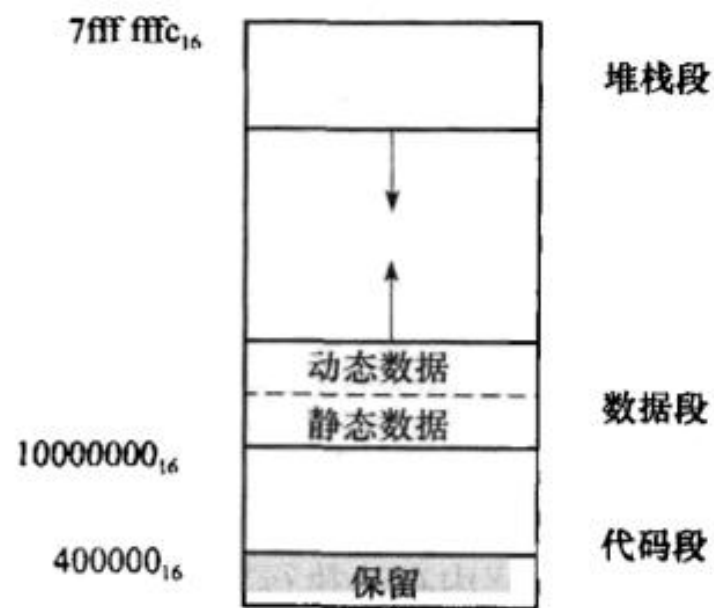
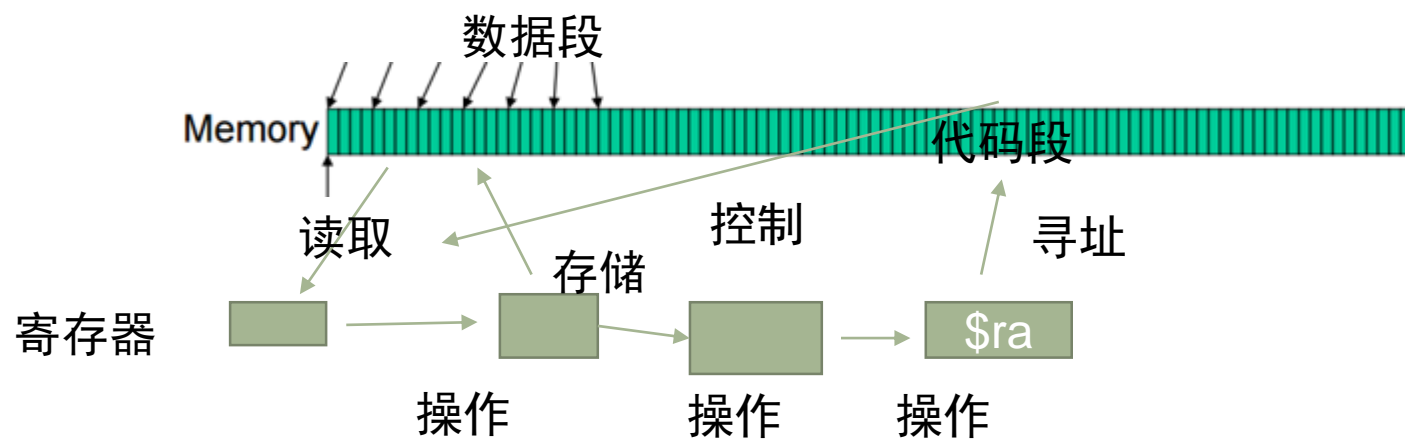
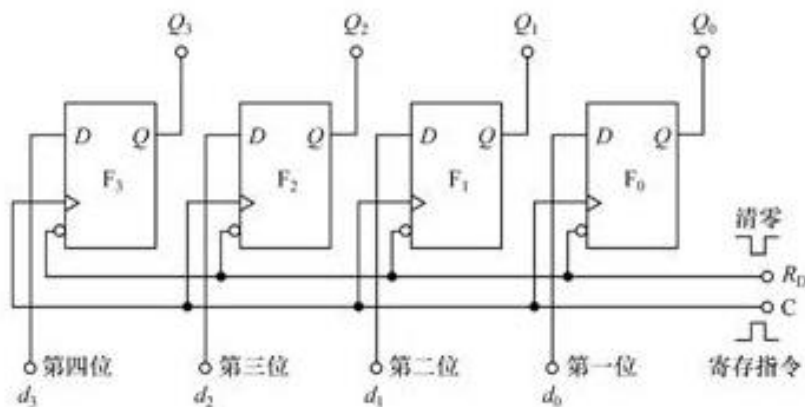


图 B-5-1 内存布局

寄存器（register）

- 寄存器是CPU内部用来存放数据的一些小型存储区域，用来暂时存放参与运算的数据和运算结果。是进行运算的基本单元。
- 所有的MIPS算术运算指令必须在寄存器上操作
- MIPS中寄存器大小32位，数量是32个



| Registers | Coproc 1 | Coproc 0 | |
|-----------|----------|----------|------------|
| Name | Number | | Value |
| \$zero | 0 | | 0x00000000 |
| \$at | 1 | | 0x10010000 |
| \$v0 | 2 | | 0x0000000a |
| \$v1 | 3 | | 0x00000000 |
| \$a0 | 4 | | 0x10010000 |
| \$a1 | 5 | | 0x00000000 |
| \$a2 | 6 | | 0x00000000 |
| \$a3 | 7 | | 0x00000000 |
| \$t0 | 8 | | 0x00000063 |
| \$t1 | 9 | | 0x00000000 |

寄存器名称

| 寄存器名称 | 编号 | 使用规则 | 寄存器名称 | 编号 | 使用规则 |
|--------|----|--------------|-------|----|---------------|
| \$zero | 0 | 恒为0 | \$s0 | 16 | 保存临时值（过程调用预留） |
| \$at | 1 | 为汇编器保留 | \$s1 | 17 | 保存临时值（过程调用预留） |
| \$v0 | 2 | 表达式求值以及函数的结果 | \$s2 | 18 | 保存临时值（过程调用预留） |
| \$v1 | 3 | 表达式求值以及函数的结果 | \$s3 | 19 | 保存临时值（过程调用预留） |
| \$a0 | 4 | 参数1 | \$s4 | 20 | 保存临时值（过程调用预留） |
| \$a1 | 5 | 参数2 | \$s5 | 21 | 保存临时值（过程调用预留） |
| \$a2 | 6 | 参数3 | \$s6 | 22 | 保存临时值（过程调用预留） |
| \$a3 | 7 | 参数4 | \$s7 | 23 | 保存临时值（过程调用预留） |
| \$t0 | 8 | 临时（不为过程调用预留） | \$t8 | 24 | 临时（不为过程调用预留） |
| \$t1 | 9 | 临时（不为过程调用预留） | \$t9 | 25 | 临时（不为过程调用预留） |
| \$t2 | 10 | 临时（不为过程调用预留） | \$k0 | 26 | 为OS内核保留 |
| \$t3 | 11 | 临时（不为过程调用预留） | \$k1 | 27 | 为OS内核保留 |
| \$t4 | 12 | 临时（不为过程调用预留） | \$gp | 28 | 全局区域的指针 |
| \$t5 | 13 | 临时（不为过程调用预留） | \$sp | 29 | 堆栈指针 |
| \$t6 | 14 | 临时（不为过程调用预留） | \$fp | 30 | 帧指针 |
| \$t7 | 15 | 临时（不为过程调用预留） | \$ra | 31 | 返回地址（函数调用使用） |

图 B-6-1 MIPS 寄存器和使用规则

数据的装载和保存（Load/Store 指令）

- 主存（RAM）的存取 access 只能用 load / store 指令来完成。
- 所有其他的指令都使用的是寄存器作为操作数。

load指令

```
lw          register_destination, RAM_source
            # copy word (4 bytes) at
            # source_RAM location
            # to destination register.
            # load word -> lw

lb          register_destination, RAM_source
            # copy byte at source RAM
            # location to low-order byte of
            # destination register,
            # and sign -e.g. tend to
            # higher-order bytes
            # load byte -> lb

li          register_destination, value
            #load immediate value into
            #destination register
            #load immediate --> li
```

store 指令

```
sw          register_source, RAM_destination
           #store word in source register
           # into RAM destination

sb          register_source, RAM_destination
           #store byte (low-order) in
           #source register into RAM
           #destination
```

内存寻址

- 装载地址：load address，相当于直接寻址，把数据地址直接载入寄存器。
- 间接寻址：indirect addressing，间接寻址，把寄存器内容作为地址。
- 基线寻址/索引寻址：based or indexed addressing，相对寻址，利用补偿值(offset)寻址。

直接寻址/装载地址：load address:

```
la      $t0, var1
```

- 把 var1 在主存（RAM）中的地址拷贝到寄存器 t0 中。var1 也可以是程序中定义的一个子程序标签的地址。

| | | |
|--------------|---|---|
| print string | 4 | \$a0 = address of null-terminated string to print |
|--------------|---|---|

```
.data
str:    .asciiz "the answer = "
.text
li $v0, 4           # system call code for print_str
la $a0, str          # address of string to print
syscall             # print the string
```


间接寻址：indirect addressing:

```
lw          $t2, ($t0)
```

- 主存中有一个字的地址存在 t0 中，按这个地址找到那个字，把字拷贝到寄存器 t2 中。

```
sw          $t2, ($t0)
```

- 把 t2 中的字存入 t0 中的地址指向的主存位置。

基线寻址/索引寻址: based or indexed addressing:

```
lw          $t2, 4($t0)
```

- 把 t0 中地址+4 所得的地址所对应的主存中的字载入寄存器 t2 中，4 为包含在 t0 中的地址的偏移量。

```
sw          $t2, -12($t0)          # offset can be negative
```

- 把 t2 中的内容存入 t0 中的地址-12 所得的地址所对应的主存中，存入一个字，占用 4 字节，消耗 4 个内存号，可见，地址偏移量可以是负值
- 适用于数组、堆栈等。

demo1

```

Edit Execute
lab2_ls.asm
1  .data
2      str: .asciiz "A"
3  .text
4  main:
5      la $a0, str
6      li $v0, 4
7      syscall
8      lb $t0, ($a0)
9      addi $t0, $t0, 32
10     sb $t0, str
11     syscall
12     li $v0, 10
13     syscall

```

```

.data
    arrayx: .word 11
.text
main:
    lw $a0, arrayx
    li $v0, 1
    syscall
    li, $v0, 10
    syscall

```

```

.data
    arrayx: .word 11
.text
main:
    la $t0, arrayx
    move $a0, $t0
    li $v0, 1
    syscall
    li, $v0, 10
    syscall

```

算术运算指令： Arithmetic Instructions

- 算术运算指令的所有操作数都是寄存器，不能直接使用 RAM 地址或间接寻址
- 操作数的大小都为 Word（4-Byte）

算术运算指令：Arithmetic Instructions

[illegible]

算术运算指令： Arithmetic Instructions

```
mult      $t3,$t4      # multiply 32-bit quantities in $t3
                        # and $t4, and store 64-bit
                        # result in special registers Lo
                        # and Hi:  (Hi,Lo) = $t3 * $t4

div        $t5,$t6      # Lo = $t5 / $t6    (integer quotient)
                        # Hi = $t5 mod $t6    (remainder)

mfhi       $t0          # move quantity in special register Hi
                        # to $t0:  $t0 = Hi

mflo       $t1          # move quantity in special register Lo
                        # to $t1:  $t1 = Lo,  used to get at
                        # result of product or quotient

move       $t2,$t3      # $t2 = $t3
```

demo2

```
.data
str1: .asciiz "13/4  quotient is: "
str2: .asciiz " , remainder is : "

.text
main:

    la $a0, str1
    li $v0, 4
    syscall

    li $t0, 13
    li $t1, 4
    divu $t0, $t1
    mflo $a0
    li $v0, 1
    syscall

    la $a0, str2
    li $v0, 4
    syscall

    mfhi $a0
    li $v0, 1
    syscall

    li $v0, 10
    syscall
```

作业1

- 1.在以下两种情况下计算并输出 $(a+b*c)/d$ 的商和余数,同时输出自己的学号和姓名。
 - 1. a,b,c,d的值在source file中给定 (a,b,c,d 为整型数)
 - 2. a,b,c,d的值由用户给定 (a,b,c,d 为整型数)
- 2. 将字符串 “abc” 改为大写样式, 在屏幕上打印输出。

预习（下次作业交）：

- A 为三个正整数的二次项的和 ($A=x^2+y^2+z^2+xy+yz+xz, x,y,z \in \mathbb{Z}$), 打印输出所有小于400的A的值, 使用寄存器 \$a0,\$a1,\$a2表示x,y,z。并输出作业完成当天的日期。

作业提交说明

- 统一提交到sakai站点；
- 提交内容：包括源码和实验报告
 - 源代码文件命名规则：lab编号_作业题号_子问题编号.s（如：lab2_1_1.s,lab2_2.s）。代码至少能在QtSpim 或 Mars 中的一个仿真器上仿真通过，运行结果符合预期。
 - 实验报告参考模板格式（生成pdf版本上交）。应包括对问题的分析和解决思路及实验结果截图。
- 截止日期：23:00 March 4, Monday。
- 所有班级截止日期相同。

Tips : 系统调用和I/O操作

Table of Available Services

| Service | Code in \$v0 | Arguments | Result |
|-----------------------------|--------------|---|---|
| print integer | 1 | \$a0 = integer to print | |
| print float | 2 | \$f12 = float to print | |
| print double | 3 | \$f12 = double to print | |
| print string | 4 | \$a0 = address of null-terminated string to print | |
| read integer | 5 | | \$v0 contains integer read |
| read float | 6 | | \$f0 contains float read |
| read double | 7 | | \$f0 contains double read |
| read string | 8 | \$a0 = address of input buffer \$a1 = maximum number of characters to read | <i>See note below table</i> |
| sbrk (allocate heap memory) | 9 | \$a0 = number of bytes to allocate | \$v0 contains address of allocated memory |
| exit (terminate execution) | 10 | | |
| print character | 11 | \$a0 = character to print | <i>See note below table</i> |
| read character | 12 | | \$v0 contains character read |