第二章 汇编语言

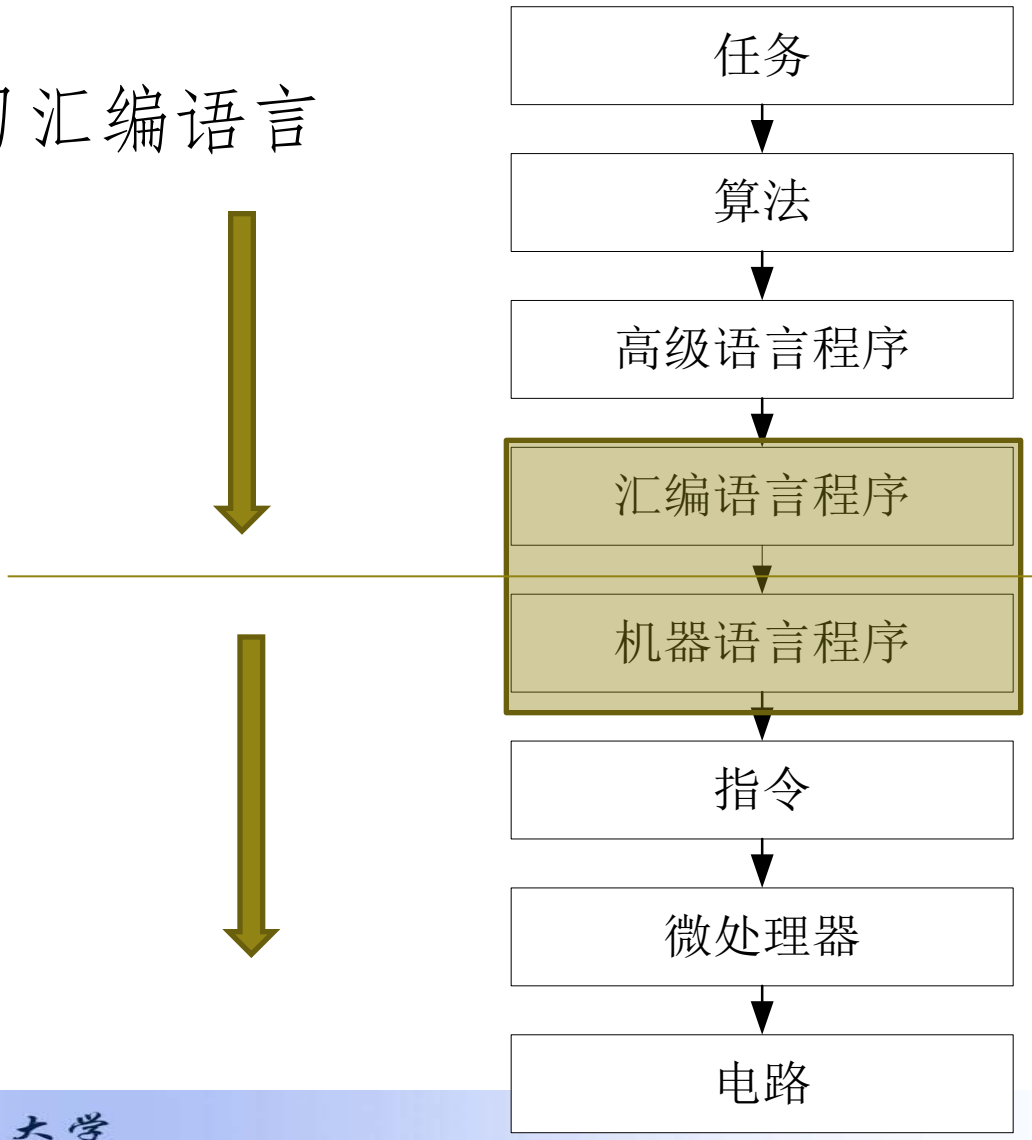
华中科技大学
电子信息与通信学院
School of Electronic Information and Communications

学习目标

- 汇编指令的构成
- 指令集结构的特点
- MIPS指令集指令的编码
- MIPS汇编指令的功能和应用
  - 算术运算指令、传输类指令

2.1 计算机语言

- 为什么要学习汇编语言



几种计算机语言的特点

- 高级语言：
  - 是一类面向问题的程序设计语言，且独立于计算机硬件，对具体算法进行描述，所以又成为“算法语言”。它的特点是独立性、通用性和可移植性好。例如：BASIC，FORTRAN，PASCAL，C，C++等语言都是高级语言。
- 汇编语言：
  - 是指使用助记符号和地址符号来表示指令的计算机语言，也称之为“符号语言”。每条指令有明显的标识，易于理解和记忆。
- 机器语言：
  - 是最初级的计算机语言，它依赖于硬件，是由1、0组成的二进制编码形式的指令集合。不易被人识别，但可以被计算机直接执行。

几个基本概念

- 汇编：
  - 把汇编语言翻译为机器语言的过程
- 汇编程序：
  - 实现汇编过程的软件程序
- 汇编语言源程序：
  - 用户采用汇编语言编写的程序
- 指令
  - 计算机能执行的代码的最小单位
- 程序
  - 指令的有序组合
- 指令集
  - 计算机能执行的所有指令的集合

2.2计算机指令

- 计算机指令通常由两个部分构成：操作码和操作数。
  - 操作码指明计算机应该执行什么样的操作，
  - 操作数:指出该操作处理的数据或数据存放的地址。
  - 操作码和操作数:二进制编码。

指令集结构——微处理器架构

- 指令的构成，编码方式，支持的指令集
  - 复杂指令集计算机（CISC）
    - 强化指令功能，实现软件功能向硬件功能转移
  - 精简指令集计算机（RISC）
    - 尽可能地降低指令集结构的复杂性，以达到简化硬件实现、提高性能的目的
  - VLIW(超长指令字指令集),
  - EPIC(显式并行指令集)

CISC结构指令集特点

①指令系统复杂庞大，指令数目一般多达2、3百条。
②寻址方式多
③指令格式多
④指令字长不固定
⑤可访存指令不加限制
⑥各种指令使用频率相差很大
⑦各种指令执行时间相差很大

- 适合于通用机
- Intel公司的X86系列CPU是典型的CISC体系的结构

RISC结构指令集的特点

①精简了指令系统，流水线以及常用指令均可用硬件执行；

②采用大量的寄存器，使大部分指令操作都在寄存器之间进行，提高了处理速度；

③每条指令的功能尽可能简单，并在一个机器周期内完成；

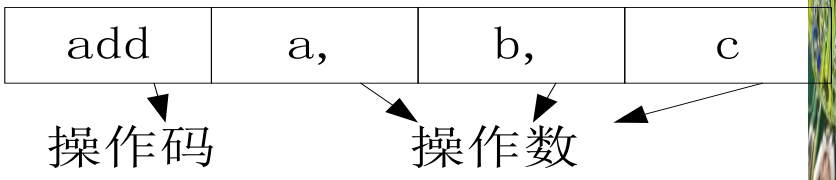
④所有指令长度均相同；

⑤只有Load和Store操作指令才访问存储器，其它指令操作均在寄存器之间进行；

- 适合于专用机
- MIPS R3000、HP-PA8000系列，Motorola M88000

2.3汇编指令——MIPS RISC

- 汇编指令是机器指令的符号表示，包括操作数和操作码
- MIPS汇编指令的一般格式为：
  - [标号：]操作码 操作数1，操作数2，操作数3 [#注释]
    - 标号代表指令在内存中的存储地址，
    - 操作码表示执行什么操作，
    - 操作数表示操作的对象（数据或地址），
    - 注释解释指令的功能，为方便读者读懂程序的功能而添加的，汇编程序将忽略这部分内容。
  - 加法运算汇编指令：
    - add a,b,c



- 不同的微处理器对add，sub等操作码的符号表示不同，
- 不同的指令结构对操作数a,b,c,d,e,t0,t1等的存取方式也不同

32位MIPS处理器常用的汇编指令

类型	指令	指令举例	含义	备注
算术运算	加法	add $s1,$s2,$s3	$s1=$s2+$s3	三个寄存器操作数
	减法	sub $s1,$s2,$s3	$s1=$s2-$s3	三个寄存器操作数
	加立即数	addi $s1,$s2,20	$s1=$s2+20	用来加立即数
数据传送	读取字	lw $s1,20($s2)	$s1=mem[$s2+20]	从内存读字到寄存器
	存储字	sw $s1,20($s2)	mem[$2+20] =$s1	从寄存器写字到内存
	读取半字	lh $s1,20($s2)	$s1=mem[$s2+20]	从内存读半字到寄存器
	读取无符号半字	lhu $s1,20($s2)	$s1=mem[$s2+20]	从内存读半字到寄存器
	存储半字	sh $s1,20($s2)	mem[$s2+20] =$1	从寄存器写半字到内存
	读取字节	lb $s1,20($s2)	$s1=mem[$s2+20]	从内存读字节到寄存器
	读取无符号字节	lbu $s1,20($s2)	$s1=mem[$s2+20]	从内存读字节到寄存器
	存储字节	sb $s1,20($s2)	mem[$s2+20] =$s1	从寄存器写字节到内存
	读取链接字	ll $s1,20($s2)	$s1= mem[$s2+20]	读字作为原子交换的第一半
	条件存储字	sc $s1,20($s2)	mem[$s2+20] =s$1;$s1=0 或 1	写字作为原子交换的第二半
	读取立即数到高半字	lui $s1,20	$s1=20*$2^{16}$	读取一个常数到高16位
逻辑操作	与	and $s1,$s2,$s3	$s1=$s2&$s3	三个寄存器，位与
	或	or $s1,$s2,$s3	$s1=$s2|$s3	三个寄存器，位或
	或非	nor $s1,$s2,$s3	$s1=~($s2|$s3)	三个寄存器，位或非
	与立即数	andi $s1,$s2,20	$s1=$s2&20	寄存器与立即数位与
	或立即数	ori $s1,$s2,20	$s1=$s2|20	寄存器与立即数位或
	逻辑左移	sll $s1,$s2,10	$s1=$s2<<10	左移常数次
	逻辑右移	srl $s1,$s2,10	$s1=$s2>>10	右移常数次
条件跳转	相等转移	beq $s1,$s2,25	If ($s1=$s2) goto PC+4+25*4	相等测试，转移
	不相等转移	bne $s1,$s2,25	If ($s1!=$s2) goto PC+4+25*4	不相等测试，转移
	小于设置	slt $s1,$s2,$s3	If($s2<$s3) $s1=1 else $s1=0	比较小于设置$s1=1
	低于设置	sltu $s1,$s2,$s3	If($s2<$s3) $s1=1 else $s1=0	比较低于设置$s1=1
	小于常数设置	slti $s1,$s2,20	If($s2<20) $s1=1 else $s1=0	和常数比较小于设置$s1=1
	低于常数设置	sltiu $s1,$s2,20	If($s2<20) $s1=1 else $s1=0	和常数比较低于设置$s1=1
无条件跳转	直接跳转	j 2500	goto 2500*4	跳转到目标地址
	间接跳转	jr $ra	goto $ra	用在分支和子程序返回
	跳转并链接	jal 2500	$ra=PC+4; goto 2500*4	用在子程序调用

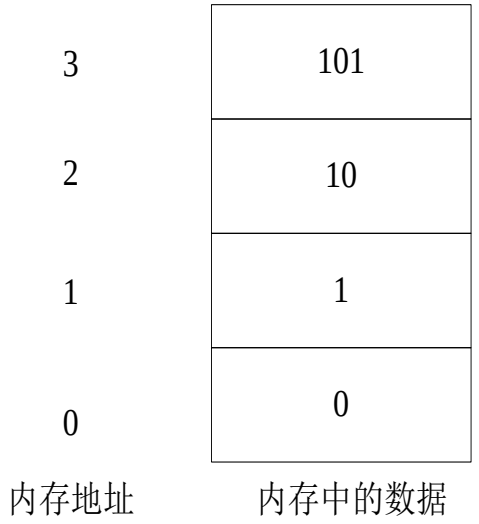
2.4操作数类型

- 寄存器操作数

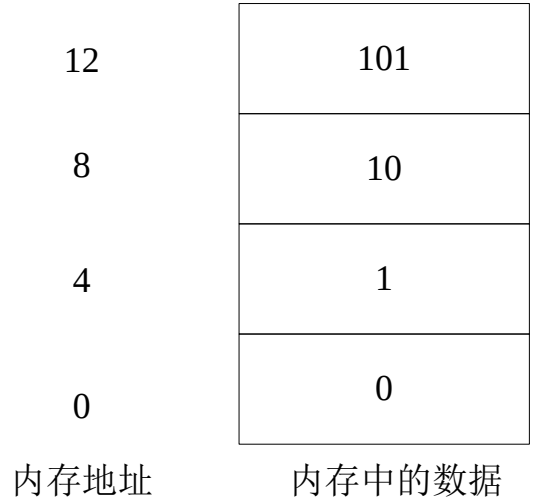
编号	名称	用途(编译器约定，指令约定）
$0	$zero	常量0(constant value 0)
$2-$3	$v0-$v1	函数调用返回值(values for results and expression evaluation)
$4-$7	$a0-$a3	函数调用参数(arguments)
$8-$15	$t0-$t7	临时寄存器(temporary)
$16-$23	$s0-$s7	存储寄存器，C语言中定义的变量可以保存在这些寄存器中。同时这些寄存器也可以保存内存单元的起始地址（基地址）
$24-$25	$t8-$t9	临时寄存器(temporary)
$28	$gp	全局指针(Global Pointer)
$29	$sp	栈指针(Stack Pointer)
$30	$fp	帧指针(Frame Pointer)
$31	$ra	子程序调用返回地址(return address)

存储器操作数

- 字节类型数组在内存中的存储



- 字类型数组在内存中的存储



- 采取字节对齐的策略去存放数据：即半字类型的数据从偶地址开始存放，而字类型的数据从4的整数倍地址开始存放

例2.1假设数组A是一个具有100个元素的字类型数组，其在内存中的起始地址（基地址）保存在寄存器$s2中，g和h分别是保存在寄存器$s0和$s1中的数据，要完成C语言中的以下语句：

g=h+A[8];

汇编指令：

lw $t0,32($s2)

#该指令从地址为$s2+32的连续的4个内存单元取值保存到寄存器$t0中

add $s0,$s1,$t0

- 由于每个字类型的数据占据4个内存单元，所以A中的第8个元素的地址相对于该数组的基地址的偏移量为8*4，而不是8。

立即数

- 立即数或常数。
- 立即数在处理器设计中，通常将它们和指令绑定在一起，成为指令的一部分，这样可以加快立即数操作指令的执行效率。
  - 如指令lw $t0,32($s2)中的32即为一个立即数，
  - 指令addi $s1,$s2,40中的40也是一个立即数，
  - 前者32是操作数地址中的立即数，后者40是立即数操作数

2.5 MIPS指令编码

- R型指令
  - 仅具有寄存器操作数的指令

op	rs	rt	rd	shamt	funct
6位（第一段）	5位(第二段）	5位（第三段）	5位（第四段）	5位（第五段）	6位（第六段）

    - op：操作码的编码，表明该指令的基本功能
    - rs：第一个源操作数寄存器的编码
    - rt：第二个源操作数寄存器的编码
    - rd：目的操作数寄存器的编码
    - shamt：移位指令的移位次数编码
    - funct：功能码，确定op域范围内的具体的指令功能

- I型指令
  - 含有立即数操作数的指令

op	rs	rt	常数地址（constant address）
6位（第一段）	5位(第二段）	5位（第三段）	16位（第四段）

- op：操作码的编码，表明该指令的基本功能；
- rs：第二个操作数寄存器的编码；
- rt：第一个操作数寄存器的编码；
- 常数地址（constant address）：常数或内存地址偏移量立即数的二进制编码。

- J型指令
  - 为实现远距离的跳转，设计了一类伪直接跳转指令，这类指令为无条件跳转指令

6位操作码编码	26位跳转地址

  - 实际内存地址为

PC的高4位	26位跳转地址	00

2.6常用MIPS汇编指令

- C语言函数

```
int sum_pow2(int b, int c)
{
int pow2[8] = {1, 2, 4, 8, 16, 32, 64, 128};
int a, ret;
a = b + c;
if (a < 8)
ret = pow2[a];
else
ret = 0;
return(ret);
}
```

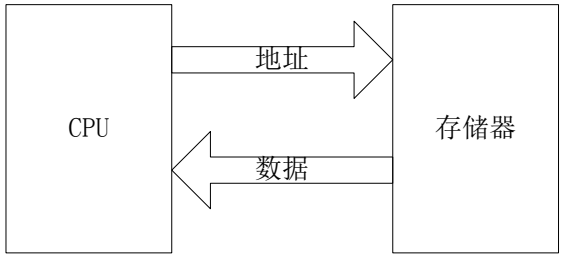
已知变量pow2,a,ret保存在内存中，而参数b，c以及返回结果保存在寄存器中。

- 该函数执行哪些操作：
  - 语句a=b+c：首先执行b+c的算术运算，然后再将该算术运算的结果保存到内存中。
  - 语句if(a<8)：比较判断内存值是否小于8，并且根据比较结果跳转到不同语句执行
  - 函数的调用和返回
- 该C语言函数需要实现算术运算、数据存取以及程序控制等操作，这些操作都需要微处理器提供相应的指令来实现

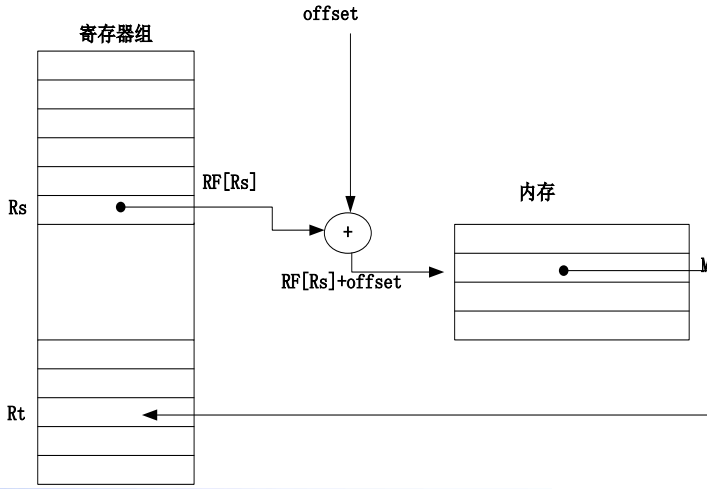
数据传送指令

- 数据从存储器传送到寄存器称为装载（load）

lb Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]
lbu Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]
lh Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]
lhu Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]
lw Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]
lwl Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]
lwr Rt, offset(Rs) # RF[Rt] = Mem[RF[Rs] + Offset]



- b表示字节传送，
- h表示半字传送，
- w表示字传送，
- u表示无符号扩展，不带u表示符号扩展



- 例3.1假设寄存器$s0=0x00000000，地址为0x00000000开始处内存单元存储的数据如图所示，执行以下指令之后，试问各Rt寄存器的值是多少？

lb $s1,0($s0)　　$s1=0xffffff80

lbu $s1,0($s0)　　$s1=0x00000080

lh $s1,0($s0)　　$s1=0xffff8081.

lhu $s1,0($s0)　　$s1=0x00008081

lw $s1,0($s0)　　$s1=0x80818283

0x00000000	0x80
0x00000001	0x81
0x00000002	0x82
0x00000003	0x83
0x00000004	0x84
0x00000005	0x85
0x00000006	0x86
0x00000007	0x87

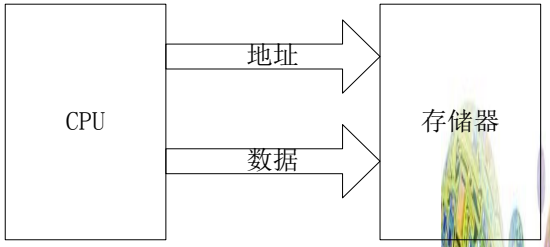
lwl $s1,6($s0)　　$s1=0x86858400

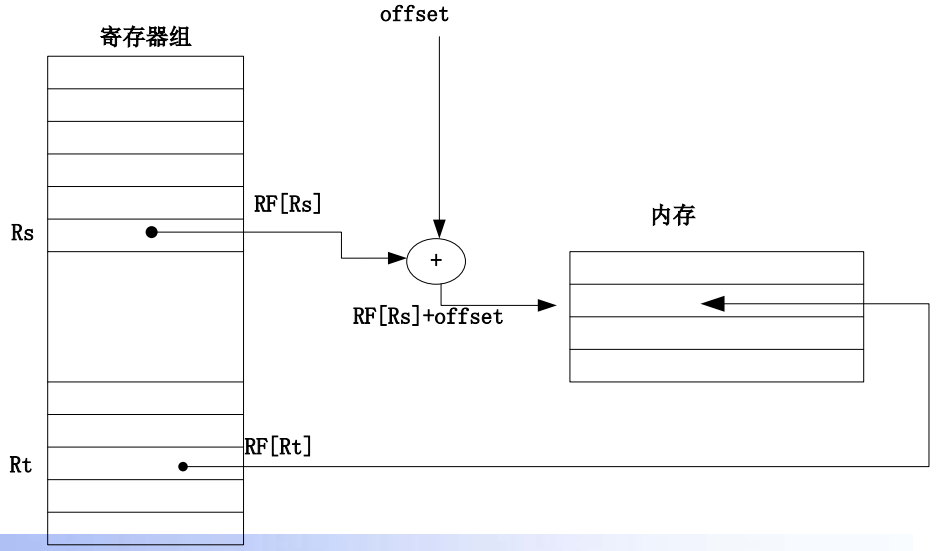
$s1的初始值为0x00000000

lwr $s1,3($s0)　　$s1=0x00000083

- 数据从寄存器传送到存储器称为存储（store）

sb Rt, offset(Rs) # Mem[RF[Rs] + Offset] = RF[Rt]
sh Rt, offset(Rs) # Mem[RF[Rs] + Offset] = RF[Rt]
sw Rt, offset(Rs) # Mem[RF[Rs] + Offset] = RF[Rt]
swl Rt, offset(Rs) # Mem[RF[Rs] + Offset] = RF[Rt]
swr Rt, offset(Rs) # Mem[RF[Rs] + Offset] = RF[Rt]





- 假设寄存器$s0=0x00000000,$s1=0x81828384,地址为0x00000000开始处的内存单元存储的数据如图所示，执行以下指令之后，试问各内存单元的值是多少?

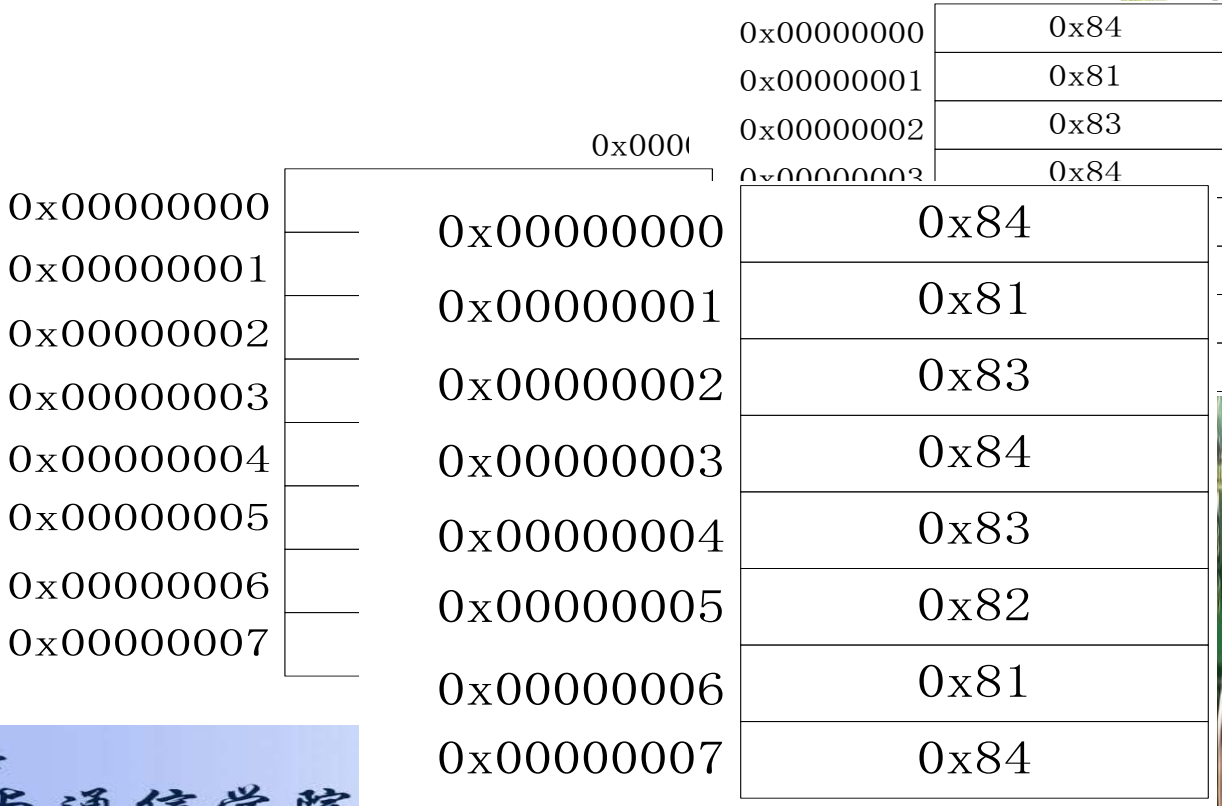
sb $s1,0($s0)

sh $s1,2($s0)

sw $s1,4($s0)

swl $s1,6($s0)

swr $s1,3($s0)



- 特殊寄存器（low，high）到通用寄存器的数据传送指令
  - 将数据移出low或high寄存器

```
mfhi Rd # RF[Rd] = HIGH
mflo Rd # RF[Rd] = LOW
```

  - 将数据移入low或high寄存器

```
mthi Rs # HIGH = RF[Rs]
mtlo Rs # LOW = RF[Rs]
```

- 立即数到寄存器

lui Rt, Imm # RF[Rt] = Imm<<16 | 0x0000 Imm表示立即数

算术运算指令

- 加法运算

```
add   Rd, Rs, Rt    # RF[Rd] = RF[Rs] + RF[Rt]
addu  Rd, Rs, Rt    # RF[Rd] = RF[Rs] + RF[Rt]
addi  Rt, Rs, Imm # RF[Rt] = RF[Rs] + Imm
addiu Rt, Rs, Imm # RF[Rt] = RF[Rs] + Imm
```

  - 符号数加法结果产生溢出，微处理器将产生异常；
  - 而无符号加法不会产生溢出异常。
  - 立即数加法中
    - 如果是符号数加法，则16位立即数进行符号数扩充为32位之后再加；
    - 若为无符号加法则进行无符号扩充，即在高16位补0，然后再进行加法运算。

- 减法运算

```
sub   Rd, Rs, Rt # RF[Rd] = RF[Rs] - RF[Rt]
subu Rd, Rs, Rt # RF[Rd] = RF[Rs] - RF[Rt]
```

- 乘法运算

```
mult   Rs, Rt # High |Low = RF[Rs] * RF[Rt]
multu Rs, Rt # High |Low = RF[Rs] * RF[Rt]
```

- 除法运算

```
div   Rs, Rt # Low = 商 ( RF[Rs] / RF[Rt] );# High = 余数 ( RF[Rs] / RF[Rt] )
divu Rs, Rt# Low = 商 ( RF[Rs] / RF[Rt] );# High = 余数 ( RF[Rs] / RF[Rt] )
```

作业

- 3. (1) , (2)
- 4. (3) , (4)
- 5. (2)