《计算机组成原理》第二次作业

网络空间安全学院 信息安全 陆皓喆 2211044

2.4

For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
sll $t0, $s0, 2          # $t0 = f * 4
add $t0, $s6, $t0        # $t0 = &A[f]
sll $t1, $s1, 2          # $t1 = g * 4
add $t1, $s7, $t1        # $t1 = &B[g]
lw $s0, 0($t0)           # f = A[f]
addi $t2, $t0, 4         #
lw $t0, 0($t2)           #
add $t0, $t0, $s0        #
sw $t0, 0($t1)           #
```

第一句，addi $t2, $t0, 4 ，意思是，将$t0的地址加上4，即往后移动一位，赋值给$t2

所以，对应的c代码应该是：

```
$t2=&A[f+1]
```

第二句，lw $t0, 0($t2) ，意思是，将原$t0(A[f])的值赋值成$t2的首地址，就是上一句中对应的A[f+1]

所以，对应的c代码应该是：

```
$t0=A[f+1]
```

第三句，add $t0, $t0, $s0 ，意思是，在$t0的基础上再加上$s0，注意到$s0其实是一个数组的A[f]

$t0的意思是，A[f+1]数组的地址，说明加上了A[f+1]

所以，对应的代码应该是：

```
$t0=A[f]+A[f+1]
```

第四句，sw $t0, 0($t1)，表示将$t0的值赋值给$t1的第一个元素，而$t1的第一个元素在前面提到过了，是B[g]

所以，对应的代码应该是：

```
B[g]=A[f]+A[f+1]
```

2.5

Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

由题意，我们需要给出字符串“0xabcdef12”的大端序与小端序的表示形式，使用从0开始存储的数据地址。

大端序：

在大端序中，越前面的字符就会被优先在低地址输出，所以结果应该是：

地址	0	4	8	12
字节	ab	cd	ef	12

小端序：

在小端序中，越前面的字符就会被优先在高地址输出，所以结果应该是：

地址	0	4	8	12
字节	12	ef	cd	ab

2.8

Translate the following MIPS code to C. Assume that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4,respectively. Assume that the base address of the arrays A and B are in registers $s6 and $s7, respectively.

```
addi $t0, $s6, 4
add $t1, $s6, $0
sw $t1, 0($t0)
lw $t0, 0($t0)
add $s0, $t1, $t0
```

第一句，addi $t0, $s6, 4的意思是，给$t0赋值，赋值为$s6(是数组A的首地址)加上4，即数组的后一位

所以，C语言代码应该是：

```
$t0=&A[1]
```

第二句，add $t1, $s6, $0的意思是，给$t1赋值为$s6与$0的地址和，就是数组A的A[0]再加上0

所以，C语言代码应该是：

```
$t1=A
```

即，$t1中存储的是A数组的首地址

第三句，sw $t1, 0($t0)的意思是，将$t1传给0($t0)的首元素，即A的地址传给A[1]

所以，C语言代码应该是：

```
A[1]=A
```

第四句，lw $t0, 0($t0)的意思是，将t0的首元素传给$t0，即将A传给寄存器$t0

所以，C语言代码应该是：

```
$t0=A
```

第五句，add $s0, $t1, $t0的意思是，将寄存器t1和t2的值相加起来，赋值给$s0

t1和t0的值都是一样的，是A

所以，C语言代码应该是：

```
$s0=2*A
```

综上所述，我们能够得出总体的C语言代码内容，就是将A的值赋给了A[1]，然后f的值变为了两倍的A地址

所以，总的C语言代码为：

```
A[1]=A;
f=2*A;
```

2.10

Assume that registers $s0 and $s1 hold the values 0x8000000000000000 and 0xD000000000000000, respectively.

2.10.1

What is the value of $t0 for the following assembly code?

add $t0, $s0, $s1

就是说，需要将两个寄存器中的值相加起来，进行运算

相加，得0x15000000000000000，由于数据位占到了17位，最多只能存储16位数据，所以我们将最前面的一位1丢去，最后得到的结果为**0x5000000000000000**

所以，$t0中存储的内容是**0x5000000000000000**

2.10.2

Is the result in $t0 the desired result, or has there been overflow?

由上一题可以得知，该数据是已经发生**溢出**后的结果，我们由于位数超出了存储的限制，导致我们必须将第一位进行丢弃。

2.10.3

For the contents of registers $s0 and $s1 as specified above,what is the value of $t0 for the following assembly code?
sub $t0, $s0, $s1

如果我们执行减法操作的话，第一个数的第一位是8，表示成二进制就是1000，第二个数的第一位是13，就是1101，两个相减，需要从前面进行借位，所以相减得到的前四位二进制的结果是1011，即为B

所以最后的结果为**0xB000000000000000**

2.10.4

Is the result in $t0 the desired result, or has there been overflow?

也是发生了溢出，因为我们在做减法的时候，往前面借了一个1，所以也没有得到我们想要的结果，所以也是发生了**溢出**。

2.10.5

For the contents of registers $s0 and $s1 as specified above,what is the value of $t0 for the following assembly code?
add $t0, $s0, $s1
add $t0, $t0, $s0

首先，第一步还是跟2.10.1一样，做一个加法，还是会发生一步溢出

第一步得到的结果是0x5000000000000000

第二步做的是在原结果上加上0x8000000000000000

所以我们得到结果，应该为**0xD000000000000000**

2.10.6

Is the result in $t0 the desired result, or has there been overflow?

在第一步计算时，其实就发生了溢出，第二步计算时没有发生溢出，总体来看，还是发生了**溢出**，因为得到的结果与预期的不同。

2.16

Assume that we would like to expand the MIPS register file to 128 registers and expand the instruction set to contain four times as many instructions.

2.16.1

How this would this affect the size of each of the bit fields in the R-type instructions?

原本我们需要6位来表示操作码（op），但是由于现在指令数扩展了4倍，所以我们需要8位来表示op

原本我们需要5位来表示源操作数寄存器，同理，现在变成7位

同理，第二个源操作数寄存器需要7个来进行表示

我们的目的寄存器需要7位来进行表示

现在，我们已经使用了8+7+7+7=29位

我们现在还有3位可以使用，所以我们对移位量和功能码随机分配，可以是1和2，也可以是2和1

所以最后的分配结果有两种可能

1. **8位op，7位rs，7位rt，7位rd，2位shamt，1位funct**
2. **8位op，7位rs，7位rt，7位rd，1位shamt，2位funct**

2.16.2

How this would this affect the size of each of the bit fields in the I-type instructions?

对于I型指令，源操作数寄存器变成了一个，所以我们需要进行缩减。

我们需要8位表示op，7位表示rs，7位表示rt，剩下的10位用于表示剩下的内容

所以最后的分配结果为

8位op，7位rs，7位rt，10位剩余部分

2.16.3

How could each of the two proposed changes decrease the size of an MIPS assembly program? On the other hand, how could the proposed change increase the size of an MIPS assembly program?

- 如何减少汇编程序的大小：

上述的改变，会使op，rs，rt部分的所占空间增大4倍，所以给我们提供了更多的语言，在一句指令中可以完成更多的内容，所以可以减少汇编程序的大小。

- 如何增大汇编程序的大小：

上述的改变虽然使op，rs，rt等指令数变大了，但是同样，缩小了后面的shamt和funct

这样的话，就会影响某些语句的执行，比如说一个语句需要执行右移8位的指令，但是我们的位移部分只有2位，不能提供三位的空间，所以会增大汇编程序的大小。