第 4 章　习　题 答 案

题目: 3,10,12,13,14

3. 假定某计算机中有一条转移指令，采用相对寻址方式，共占两个字节，第一字节是操作码，第二字节是相对位移量（用补码表示），CPU 每次从内存只能取一个字节。假设执行到某转移指令时 PC 的内容为 200，执行该转移指令后要求转移到 100 开始的一段程序执行，则该转移指令第二字节的内容应该是多少？

参考 P111

参考答案：

因为执行到该转移指令时 PC 为 200，所以说明该转移指令存放在 200 单元开始的两个字节中。因为 CPU 每次从内存只能取一个字节，所以每次取一个字节后 PC 应该加 1。

该转移指令的执行过程为：取 200 单元中的指令操作码并译码→PC+1→取 201 单元的相对位移量→PC+1→计算转移目标地址。假设该转移指令第二字节为 Offset，则 100=200+2+Offset，即 Offset = 100–202 = –102 = 10011010B

（注：没有说定长指令字，所以不一定是每条指令占 2 个字节。）

10. 下列指令序列用来对两个数组进行处理，并产生结果存放在$v0 中。假定每个数组有 2500 个字，其数组下标为 0 到 2499。两个数组的基地址分别存放在$a0 和$a1 中，数组长度分别存放在$a2 和$a3 中。要求为以下 MIPS 指令序列加注释，并简单说明该过程的功能。假定该指令序列运行在一个时钟频率为 2GHz 的处理器上，add、addi 和 sll 指令的 CPI 为 1；lw 和 bne 指令的 CPI 为 2，则最坏情况下运行所需时间是多少秒？

```
            sll   $a2,  $a2,  2
            sll   $a3,  $a3,  2
            add   $v0,  $zero,  $zero
            add   $t0,  $zero,  $zero
outer:      add   $t4,  $a0,  $t0
            lw    $t4,  0($t4)
            add       $t1,  $zero,    $zero
inner:      add   $t3,  $a1,  $t1
            lw    $t3,  0($t3)
            bne   $t3,  $t4,  skip
            addi  $v0,  $v0,  1
skip:       addi  $t1,  $t1,  4
            bne   $t1,  $a3,  inner
            addi  $t0,  $t0,  4
            bne   $t0,  $a2,  outer
```

参考 P213 表 5.4

参考答案：

1: 将 a2 的内容左移 2 位，即乘 4

2: 将 a3 的内容左移 2 位，即乘 4

3: 将 v0 置零

4: 将 t0 置零

5: 将第一个数组的首地址存放在 t4

6: 取第一个数组的第一个元素存放在 t4
7: 将 t1 置零
8: 将第二个数组的首地址存放在 t3
9: 取第二个数组的第一个元素存放在 t3
10: 如果 t3 和 t4 不相等，则跳转到 skip
11: 将 v0 的值加 1，结果存于 v0
12: 将 t1 的值加 4，结果存于 t1
13: 如果 t1 不等于 a3，即还未取完数组中所有元素，则转移到 inner
14: 将 t0 的值加 4
15: 如果 t0 不等于 a2，即还未取完数组中所有元素，则转移到 outer

该程序的功能是统计两个数组中相同元素的个数。

程序最坏的情况是：两个数组所有元素都相等，这样每次循环都不会执行 skip。因此，指令总条数为：4+2500×(3+2500×6+2)=37512504，

其中：add,addi 和 sll 的指令条数为：4+2500×(2+2500×3+1)=18757504

lw 和 bne 的指令条数为： 2500×(1+2500×3+1)=18755000

所以：程序执行的时间为：(2GHzclock 的 clock time=1/2G=0.5ns)

(18757504×1+18755000×2) × 0.5ns = 28133752ns≈0.028s

12. 以下程序段是某个过程对应的 MIPS 指令序列，其功能为复制一个存储块数据到另一个存储块中，存储块中每个数据的类型为 float，源数据块和目的数据块的首地址分别存放在$a0 和$a1 中，复制的数据个数存放在$v0 中，作为返回参数返回给调用过程。在复制过程中遇到 0 则停止，最后一个 0 也需要复制，但不被计数。已知程序段中有多个 Bug，请找出它们并修改。

```
        addi    $v0,    $zero,  0
loop:   lw      $v1,    0($a0)
        sw      $v1,    0($a1)
        addi    $a0,    $a0,    4
        addi    $a1,    $a1,    4
        beq     $v1,    $zero,  loop
```

参考答案：

修改后的代码如下：

```
        addi  $v0, $zero, 0
loop:   lw    $v1, 0($a0)
        sw    $v1, 0($a1)
        beq   $v1, $zero, exit
        addi  $a0, $a0, 4
        addi  $a1, $a1, 4
        addi  $v0, $v0, 1
        j   loop
exit:
```

13. 说明 beq 指令的含义，并解释为什么汇编程序在对下列汇编源程序中的 beq 指令进行汇编时会遇到问题，应该如何修改该程序段？

```
here:    beq   $s0,   $s2,   there
         ……
there:   addi   $s1,   $s0,   4
```

参考 P213 的表 5.4、表 5.5 ，P212 的 MIPS 指令格式、图 5.9，P197 相对寻址

参考答案：

beq 是一个 I-型指令，可以跳转到当前指令前，也可以跳转到当前指令后。其转移目的地址的计算公式为：PC+4+offset×4，offset 是 16 位带符号整数，用补码表示。因此，分支指令 beq 的相对转移范围如下。

其正跳范围为：0000 0000 0000 0000（4）～0111 1111 1111 1111（2^{17}=4+(2^{15}–1)×4）

负跳范围为：1000 0000 0000 0000（4–2^{17}=4+(–2^{15})×4）～1111 1111 1111 1111（0=4+(–1)×4）

超过以上范围的跳转就不能用上述指令序列实现。

因此，上述指令序列应该改成以下指令序列：

```
here:    bne $s0, $s2, skip
         j   there
skip:    ……
         ……
there:   addi $s1, $s0, 4
```

14. 以下 C 语言程序段中有两个函数 sum_array 和 compare，假定 sum_array 函数第一个被调用，全局变量 sum 分配在寄存器$s0 中。要求写出每个函数对应的 MIPS 汇编表示，并画出每个函数调用前、后栈中的状态、帧指针和栈指针的位置。

```
1    int sum=0;
2    int sum_array(int array[ ], int num)
3    {
4        int i;
5        for (i = 0; i < num; i ++)
6            if compare (num, i+1)    sum+=array[i] ;
7        return sum;
8    }
9    int compare (int a, int b)
10   {
11       if ( a >b)
12           return 1;
13       else
14           return 0;
15   }
```

参考 P220 例 5.10

参考答案：

程序由两个过程组成，全局静态变量 sum 分配给$s0。

为了尽量减少指令条数，并减少访问内存次数。在每个过程的过程体中总是先使用临时寄存器$t0~$t9，临时寄存器不够或者某个值在调用过程返回后还需要用，就使用保存寄存器$s0~$s7。

MIPS 指令系统中没有寄存器传送指令，为了提高汇编表示的可读性，引入一条伪指令 move 来表示寄存器传送，汇编器将其转换为具有相同功能的机器指令。伪指令“move $t0, $s0”对应的机器指令为“add $t0,$zero,$s0”。

（1）过程 sum_array：该过程和教材中例 4.10 中的不同，在例 4.10 中 array 数组是过程 set_array 的局部变量，应该在过程栈帧中给数组分配空间，但该题中的数组 array 是在其他过程中定义的，仅将其数组首地址作为参数传递给过程 sum_array（假定在在$a0 中），因此，无需在其栈帧中给数组分配空间。此外，还有一个入口参数为 num（假定在$a1 中），有一个返回参数 sum，被调用过程为 compare。因此，其栈帧中除了保留所用的保存寄存器外，还必须保留返回地址，以免在调用过程 compare 时被覆盖。是否保存$fp 要看具体情况，如果确保后面都不用到$fp，则可以不保存，但为了保证$fp 的值不被后面的过程覆盖，通常情况下，应该保存$fp 的值。

栈帧中要保存的信息只有返回地址$ra 和帧指针$fp，其栈帧空间为 4×2=8B。

汇编表示如下：

```
            move   $s0, $zero          # sum=0
set-array:  addi   $sp, $sp, –8        # generate stack frame
            sw     $ra, 4($sp)         # save $ra on stack
            sw     $fp, 0($sp)         # save $fp on stack
            addi   $fp, $sp, 4         # set $fp
            move   $t2, $a0            # base address of array
            move   $t0, $a1            # $t0=num
            move   $t3, $zero              # i=0
loop:       slt    $t1, $t3, $t0       # if i<num, $t1=1; if i>=num, $t1= 0
            beq    $t1, $zero, exit1       # if $t1= 0, jump to exit1
            move   $a0, $t0            # $a0=num
            move   $a1, $t3            # $a1=i
            addi   $a1, $a1, 1         # $a1=i+1
            jal    compare                 # call compare
            beq    $v0, $zero, else    # if $v0 = 0, jump to else
            sll    $t1, $t3, 2         # i×4
            add    $t1, $t2, $t1       # $t1=array[i]
            lw     $t4, 0($t1)         # load array[i]
```

```
            add     $s0, $s0, $t4       # sum+=array[i]
    else:   addi    $t3, $t3, 1         # i=i+1
            j       loop
    exit1:  move    $v0, $s0            # return sum
            lw      $ra, 4($sp)         # restore $ra
            lw      $fp, 0($sp)         # restore $fp
            addi    $sp, $sp, 8         # free stack frame
            jr      $ra                 # return to caller
```

（2）过程 compare：入口参数为 a 和 b，分别在$a0 和$a1 中。有一个返回参数，没有局部变量，是叶子过程，且过程体中没有用到任何保存寄存器，所以栈帧中不需要保留任何信息。

```
compare:    move    $v0, $zero          # return 0
            beq     $a0, $a1, exit2     # if $a0=$a1, jump to exit2
            slt     $t1, $a0, $a1       # if $a0<$a1, $t1=1; if $a0>=$a1, $t1= 0
            bne     $t1, $zero, exit2   # if $a0<$a1, jump to exit2
            ori     $v0, $zero,1        # return 1
exit2:      jr      $ra
```

```
int compare (int a, int b)
10  {
11      if ( a >b)
12          return 1;
13      else
14          return 0;
15  }
```

```
int compare (int a, int b)
10  {
11      if ( a >b)
12          return a;
13      else
14          return b;
15  }
```

```
compare:    ori     $v0, $a1, 1         # return b
            beq     $a0, $a1, exit1     # if $a0=$a1, jump to exit2
            slt     $t1, $a0, $a1       # if $a0<$a1, $t1=1; if $a0>=$a1, $t1= 0
            bne     $t1, $zero, exit2   # if $a0<$a1, jump to exit2
            ori     $v0, $a0, 1         # return 1
exit1:      jr      $ra
```