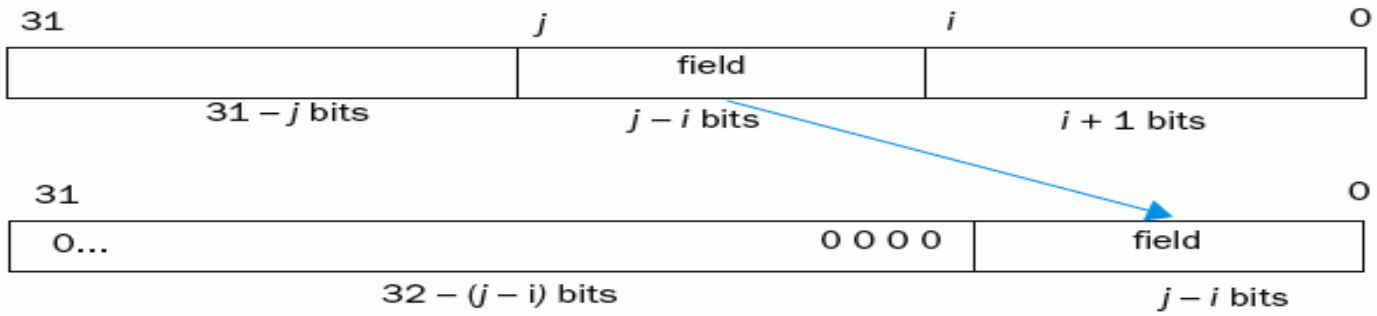
第二章 指令：机器语言

作业参考答案

第二章 指令-机器语言

2.6 [15] <§2.5> Some computers have explicit instructions to extract an arbitrary field from a 32-bit register and to place it in the least significant bits of a register. The figure below shows the desired operation:



Find the shortest sequence of MIPS instructions that extracts a field for the constant values $i = 5$ and $j = 22$ from register `$t3` and places it in register `$t0`. (Hint: It can be done in two instructions.)

最自然的考虑办法：先右移：000000 xxxxxxxxx xxxxxxxxxxxxxxxxx

再与某个掩码进行“与”操作：000000 000000000 11111111111111111

000000 000000000 xxxxxxxxxxxxxxxxx

```
srl   $t0, $t3, 6
andi  $t0, $t0, 131071
```

用上述两条指令行不行？ 不行，andi的立即数只有16位，而j-i=17位，131071有17个1！

要用其他办法（先左移9位，再右移15位）

```
sll $t0, $t3, 9
srl $t0, $t0, 15
```

若第一条指令中的$t0改成其他寄存器，则会带来什么问题？

若j=21，则可用右移和与实现：

```
srl   $t0, $t3, 6
andi  $t0, $t0, 65535
```

机器码

000000 00000 01011 01000 00110 000000

001100 01000 01000 1111 1111 1111 1111

第二章 指令-机器语言

2.15 [25] <§2.7> Implement the following C code in MIPS, assuming that `set_array` is the first function called:

```
int i;
void set_array(int num) {
     int array[10];
     for (i=0; i<10; i++) {
          array[i] = compare(num, i);
     }
}
int compare(int a, int b) {
     if (sub(a, b) >= 0)
          return 1;
     else
          return 0;
}
int sub (int a, int b) {
     return a-b;
}
```

Be sure to handle the stack and frame pointers appropriately. The variable code font is allocated on the stack, and `i` corresponds to `$s0`. Draw the status of the stack before calling `set_array` and during each function call. Indicate the names of registers and variables stored on the stack and mark the location of `$sp` and `$fp`.

复习：MIPS程序和数据的存储器分配

- 每个MIPS程序都按如下规定进行存储器分配
- 每个可执行文件都按如下规定给出代码和数据的地址

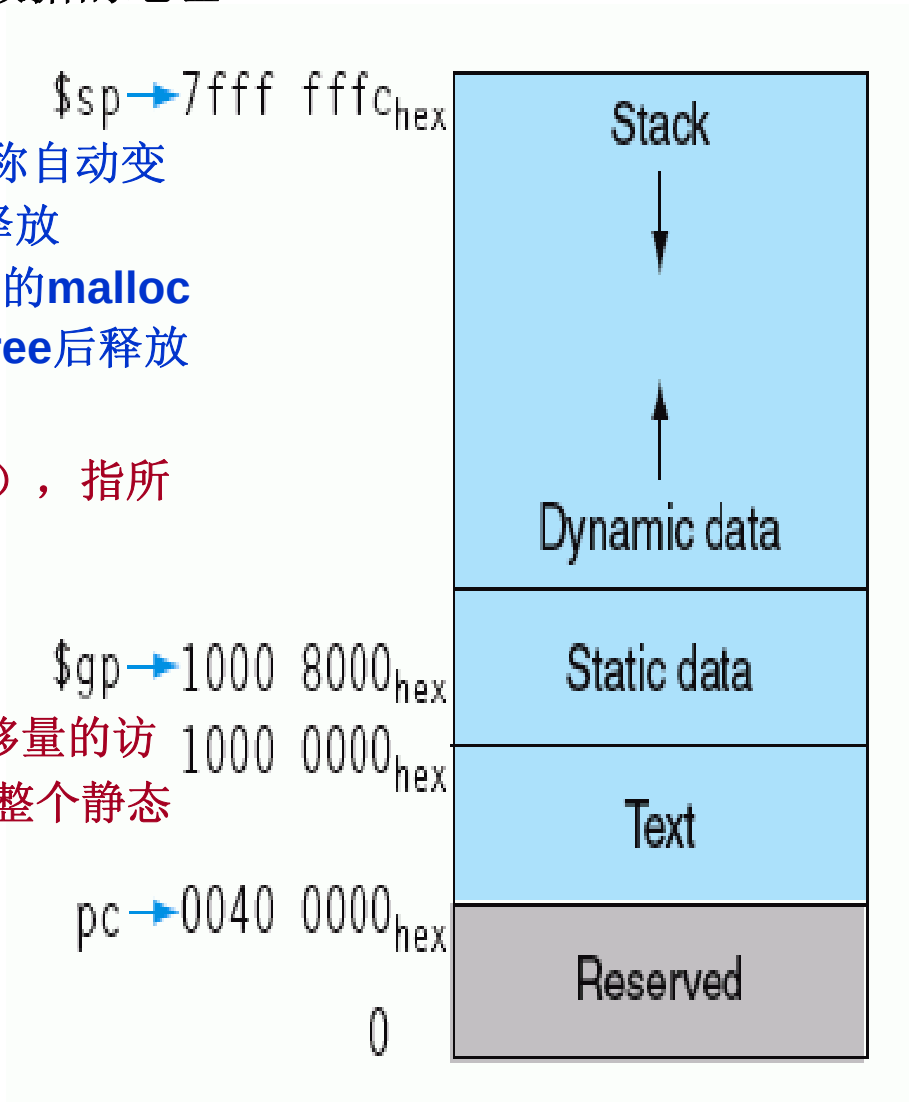
栈区位于堆栈高端，堆区位于堆栈低端

- 栈(Stack)区存放每个过程的局部数据（也称自动变量），从高往低长，从被调用过程返回后释放
- 堆(heap)区存放程序的动态数据（如：C中的malloc申请区域、链表等），从低往高长，执行free后释放

静态数据区存放的是全局变量（也称静态变量），指所有过程之外声明的变量和用Static声明的变量从固定的0x1000 0000处 开始存放

全局指针$gp固定为0x1000 8000，其16位偏移量的访问范围为0x1000 0000 到0x1000 ffff，可遍及整个静态数据区的访问

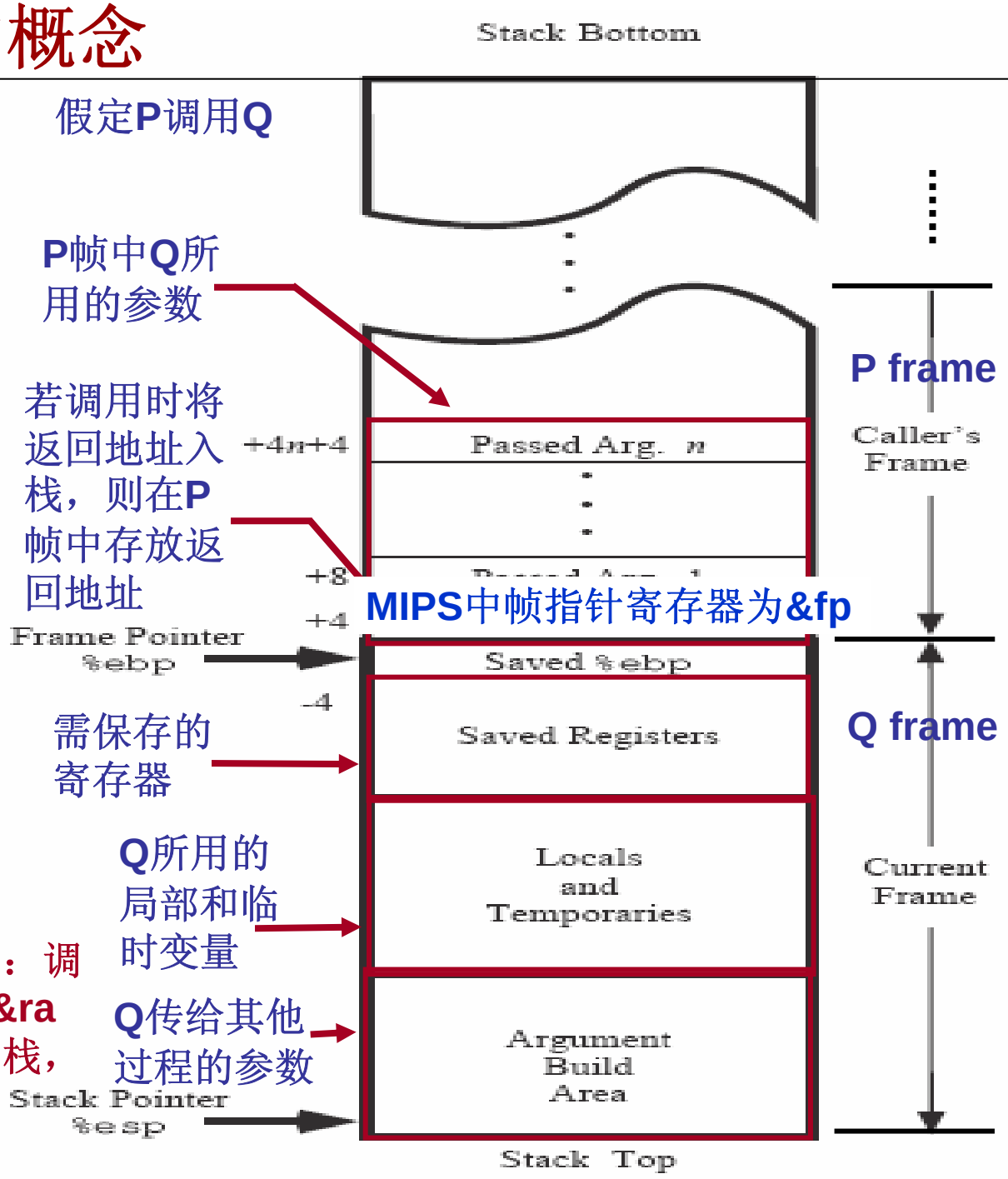
程序代码从固定的0x0040 0000处开始存放
故PC的初始值为0x0040 0000



BACK

复习：栈帧的概念

- 每个过程都有自己的栈区，称为栈帧（**Stack frame**）
- 堆栈由若干栈帧组成
- 用专门的帧指针寄存器指定起始位置
- 当前栈帧范围在帧指针和栈指针之间
- 程序执行时，栈指针可移动，帧指针不变
  所以，过程内对栈信息的访问大多通过帧指针进行，简便



MIPS返回地址处理有所不同：调用指令jal把返回地址保存在&ra中，Q把&ra入栈，返回前出栈，返回指令jr再根据&ra返回

复习： MIPS中的过程调用（假定P调用Q）

- 程序可访问的寄存器组是所有过程共享的资源，给定时刻只能被一个过程使用
- 过程调用时，一个过程中使用的寄存器的值不能被另一个过程覆盖！
- MIPS的寄存器使用约定：
  - 保存寄存器$s0 ~$s7 的值在从被调用过程返回后还要被用，被调用者需要保留
  - 临时寄存器$t0 ~$t9的值在从被调用过程返回后不需要被用（需要的话，由调用者保存），被调用者可以随意使用
  - 参数寄存器$a0~$a3在从被调用过程返回后不需要被用（需要的话，由调用者保存在栈帧或其他寄存器中），被调用者可以随意使用
  - 全局指针寄存器$gp的值不变
  - 在过程调用时帧指针寄存器$fp用栈指针寄存器$sp- 4来初始化
- 需在被调用过程Q中入栈保存的寄存器（称为被调用者保存）
  - 返回地址$ra (如果Q又调用R，则$ra内容会被破坏，故需保存)
  - 保存寄存器$s0 ~$s7 (从Q返回后P可能还会用到，Q中用的话就被破坏，故需保存)
- 除了上述寄存器以外，所有局部数组和结构也要入栈保存
- 如果局部变量发生寄存器溢出（寄存器不够分配），则也要入栈
- 每个处理器对栈帧规定的“调用者保存”和“被调用者保存”的寄存器可能不同。例：
  - x86处理器中返回地址保存在调用过程栈帧中；而MIPS则在被调用过程中保存
  - x86处理器中调用参数都保存在调用过程栈帧中；而MIPS则在被调用过程中保存额外参数
  - X86处理器中调用过程的帧指针保存在被调用过程的栈帧中；MIPS也一样。

第二章 指令-机器语言

题目分析如下：

程序由三个过程组成，全局静态变量有一个**i**，假定分配给**$s0**

（**1**）过程**set_array**：入口参数为**num**，没有返回参数，有一个局部数组，被调用过程为**compare(num, i)** 。所以其栈帧中除了保留所用的保存寄存器外，必须要保留返回地址（和旧的**$fp**），并给局部数组预留**4x10=40**个字节的空间；

（**2**）过程**compare**：入口参数为**a**和**b**，有一个返回参数，没有局部自动变量，被调用过程为**sub(a, b)**。所以其栈帧中除了保留所用的保存寄存器外，必须要保留返回地址**(**和旧**$fp)** ；

（**3**）过程**sub**：入口参数为**a**和**b**，有一个返回参数，没有局部自动变量，没有被调用过程（是一个叶过程）。所以栈帧中除了保留所要的保存寄存器外，不需要保留其他信息

（如果保留返回地址也不会错，但需额外的指令来执行保存和恢复，增加程序执行时间，一般不对叶过程的返回地址进行保存）

这里需要说明的是：

题目中给出的程序是示意性的，实际上该程序没有任何意义，为什么这么说？

过程**set_array**所做的工作就是把比较的结果写到数组**array**中，没有任何返回值，但数组**array**是局部的，当从**set_array**返回后，该过程的栈帧全部被释放，当然**array**中的值也全部无效。

相当于程序没有做任何工作。

第二章 指令-机器语言

Assuming variables i~ $s0, and base address of array is in $s1

Set-array (int num)函数：

```
Set-array:
            (保存$ra和$s0-$s4)
            ……
            addi $sp, $sp, -40
             move $s1, $sp
            move $s4, $a0          ;$s4=num
            move $s0, $zero        ;i=0
for-loop:   slti  $s2, $s0, 10     ;if i<10, $s2=1
            beq $s2, $zero, exit   ;if i>=10, exit
            sll   $s3, $s0, 2      ;i*4
            add $s3, $s3, $s1      ;$s3=array[i]
            move $a0, $s4          ;$a0=num
            move $a1, $s0          ;$a1=i
            jal compare            ;call compare
            sw $v0, 0($s3)
            addi $s0, $s0, 1
            j  for-loop
Exit:       addi $sp, $sp, 40
             ……  (恢复$ra和$s0-$s4)
            jr $ra
```

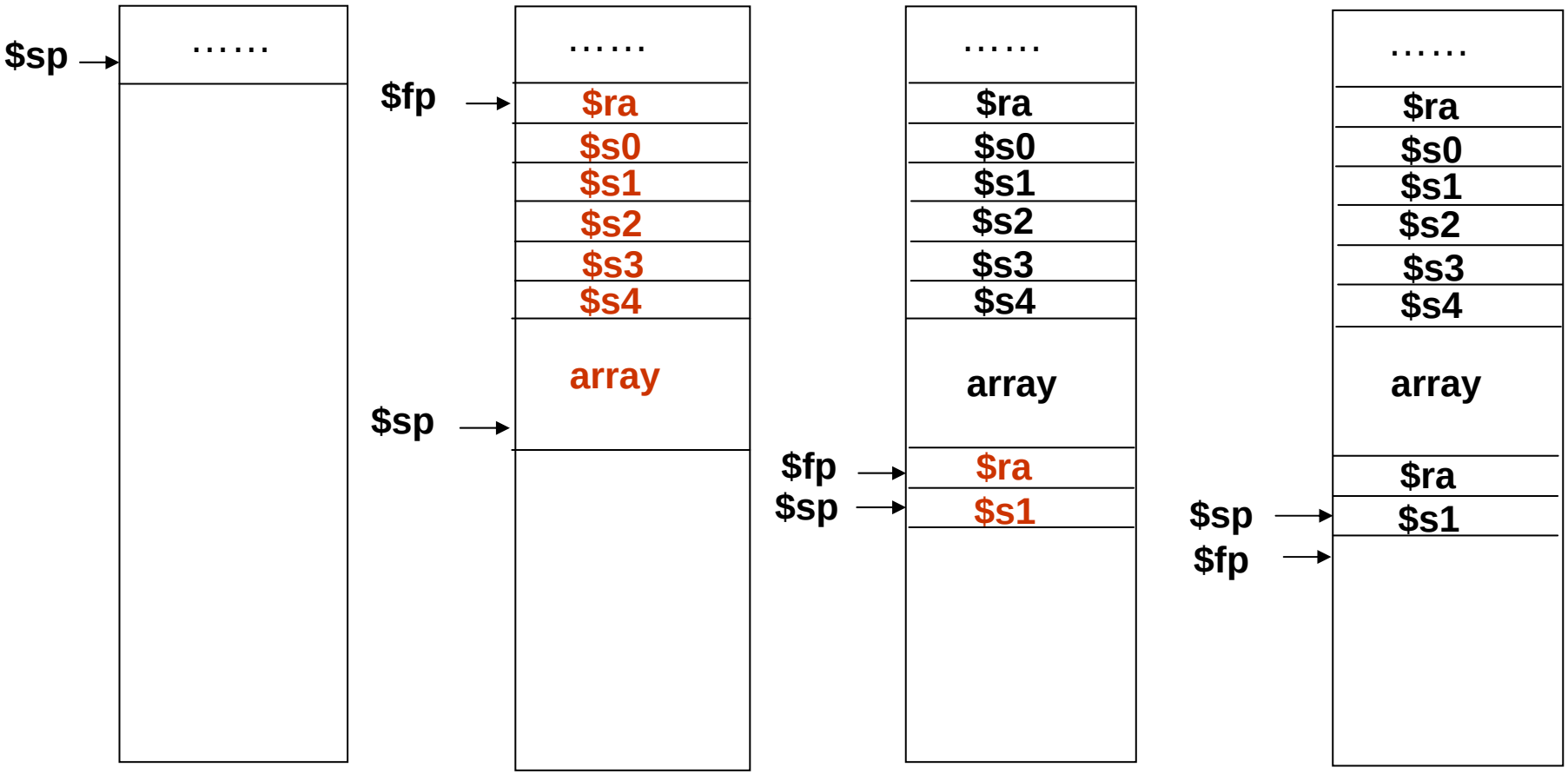
compare (int a, int b)函数：

```
Compare:addi $sp, $sp, -8
            sw $ra, 4($sp)
            sw $s1, 0($sp)
            jal  sub
            slt  $s1, $v0, $zero ; if<0, $s1=1
            beq $s1, $zero, else ;
            move $v0, $zero
            j   exit
      else: ori $v0, $zer0, 1
      exit: lw $s1, 0($sp)
            lw $ra, 4($sp)
            addi $sp, $sp, 8
            jr $ra
```

Sub (int a, int b)函数：

```
sub: addi $sp, $sp,-4
       sw $ra, 0($sp)
       sub $v0, $a0, $a1
       lw $ra, 0($sp)
       addi $sp, $sp,4
       jr $ra
```

第二章 指令-机器语言



调用set-array前
从set-array返回后

调用compare前
从compare返回后

调用sub前
从sub返回后

调用sub后

根据书中图2-16可以看出：每次过程调用开始，总是把当前$sp- 4送给$fp；

MIPS保存和恢复$fp的方式：若程序中用到$fp时就在被调用者栈帧中保存调用者的$fp，然后把$sp-4送$fp；若不用时，则不保存$fp的值。

请参看附录A-6！

第二章 指令-机器语言

2.29 [5] <§§2.3, 2.6, 2.9> Add comments to the following MIPS code and describe in one sentence what it computes. Assume that $a0 and $a1 are used for the input and both initially contain the integers *a* and *b*, respectively. Assume that $v0 is used for the output.

```
                add   $t0, $zero, $zero   ;$t0=0
loop:           beq   $a1, $zero, finish  ;if $a1=0 finish
                add   $t0, $t0, $a0       ;$t0=$t0+$a0
                sub   $a1, $a1, 1         ;$a1=$a1-1
                j     loop                ;
finish:         addi  $t0, $t0, 100       ;$t0=$t0+100
                add   $v0, $t0, $zero     ;$v0=$t0
```

其功能就是将**$a0**的值加**$a1**次（即**axb**）再加**100**

该指令序列完成以下功能：**$v0=ab+100**

第二章 指令-机器语言

2.30 [12] <§§2.3, 2.6, 2.9> The following code fragment processes two arrays and produces an important value in register $v0. Assume that each array consists of 2500 words indexed 0 through 2499, that the base addresses of the arrays are stored in $a0 and $a1 respectively, and their sizes (2500) are stored in $a2 and $a3, respectively. Add comments to the code and describe in one sentence what this code does. Specifically, what will be returned in $v0?

```
            sll     $a2, $a2, 2          ;$a2=$a2x4=10000
            sll     $a3, $a3, 2          ;$a3=$a3x4=10000
            add     $v0, $zero, $zero    ;$v0初始化为0
            add     $t0, $zero, $zero    ;$t0(偏移量:ix4)初始化为0
outer:      add     $t4, $a0, $t0        ;$t4=address of array[i]
            lw      $t4, 0($t4)          ;$t4=array[i]
            add     $t1, $zero, $zero    ;$t1(偏移量:jx4)初始化为0
inner:      add     $t3, $a1, $t1        ;$t3=address of array[j]
            lw      $t3, 0($t3)          ;$t3=array[j]
            bne     $t3, $t4, skip       ;if array[i] ≠ array[i] , skip
            addi    $v0, $v0, 1          ;$v0=$v0+1
skip:       addi$   t1, $t1, 4           ;j=j+4
            bne     $t1, $a3, inner      ;if j≠10000, 继续内循环
            addi    $t0, $t0, 4          ;i=i+4
            bne     $t0, $a2, outer      ;if i≠10000, 继续外循环
```

功能：**$v0**中存放的是两个数组中相同元素的个数。

第二章 指令-机器语言

2.31 [10] <§§2.3, 2.6, 2.9> Assume that the code from Exercise 2.30 is run on a machine with a 2 GHz clock that requires the following number of cycles for each instruction:

Instruction	Cycles
add,addi,sll	1
lw, bne	2

In the worst case, how many seconds will it take to execute this code?

2.30中程序最坏的情况就是：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：5+2500x(3+2500x6+2)=37512505

其中：add,addi和sll的指令条数为：4+2500x(2+2500x3+1)=18757504

lw和bne的指令条数为： 1+2500x(1+2500x3+1)=18755001

所以：程序执行的时间为：(2GHzclock的clock time=1/2G=0.5ns)
(18757504x1+18755001x2)x0.5ns=28133753ns≈0.028s

第二章 指令-机器语言

2.32 [5] <§2.9> Show the single MIPS instruction or minimal sequence of instructions for this C statement:

```
b = 25 | a;
```

Assume that a corresponds to register $t0 and b corresponds to register $t1.

是或运算，其中一个操作数为立即数**25**，故可用**ori**指令实现：

ori $t1, $t0, 25 机器码为：001101 01000 01001 0000 0000 0001 1001

13 8 9 25

如果把**25**换成**65536**，那指令是不是就换成： **ori $t1, $t0, 65536** ？

65536（**1 0000 0000 0000 0000**）不能用**16**位立即数表示。此时的指令序列应为：

```
lui $t1, 1
or  $t1, $t0, $t1
```

若换成**65537**呢？

```
lui $t1, 1
addi $t1, $t1, 1
or  $t1, $t0, $t1
```

由此可见，并不是所有的立即数都按相同的方式处理！

第二章 指令-机器语言

2.34 [10] <§§ 2.3, 2.6, 2.9> The following program tries to copy words from the address in register $a0 to the address in register $a1, counting the number of words copied in register $v0. The program stops copying when it finds a word equal to 0. You do not have to preserve the contents of registers $v1, $a0, and $a1. This terminating word should be copied but not counted.

```
      addi $v0, $zero, 0 # Initialize count
loop: lw   $v1, 0($a0)   # Read next word from source
      sw   $v1, 0($a1)   # Write to destination
      addi $a0, $a0, 4   # Advance pointer to next source
      addi $a1, $a1, 4   # Advance pointer to next destination
      beq  $v1, $zero, loop  # Loop if word copied != zero
```

There are multiple bugs in this MIPS program; fix them and turn in a bug-free version. Like many of the exercises in this chapter, the easiest way to write MIPS programs is to use the simulator described in Appendix A.

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:
```

第二章 指令-机器语言

2.38 [5] <§§2.9, 2.10> Given your understanding of PC-relative addressing, explain why an assembler might have problems directly implementing the branch instruction in the following code sequence:

```
here:                beq   $s0, $s2, there
…
there                add   $s0, $s0, $s0
```

Beq是一个I-Type指令，可以跳转到当前指令前，也可以跳转到当前指令后。
其计算公式为：PC+4+offset（**16**位立即数）
故偏移量offset是一个**16**位带符号整数（**4**的倍数，用补码表示）。
其正跳范围为：0000 0000 0000 0100（+4）~ 0111 1111 1111 1100（$+2^{15}-4$）
　负跳范围为：1000 0000 0000 0000（-2^{15}）~ 1111 1111 1111 1100（- 4）
超过以上范围的跳转就不能用上述指令序列实现。应该改成以下序列：

```
here:   bne $s0, $s2, skip
        j  there
Skip:
        ……
there:  add $s0, $s0, $s0
```