第二章 习题解答

1. 计算机指令由哪两部分构成？他们分别表示什么含义？
   解答：计算机的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，而操作数则指出该操作处理的数据或数据存放的地址。

2. 现代计算机系统存在哪两种类型的指令集？它们分别有什么特点？
   解答：
   复杂指令集计算机（CISC）和精简指令集计算机（RISC）
   CISC 结构指令集的特点是：
   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
   RISC 结构指令集的特点是：
   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；

3. 假定以下 C 语句中的所有变量为 32 位的寄存器，请分别采用 MIPS 汇编指令实现其功能：
   a)f=g+h+i+j;　　b)f=g+(h+5);
   c)f=f+f+i;　　d)f=g+(j+2);
   解答：灵活使用加法算术运算指令

```
a) add f, g,h
   add f,f,i
add f,f,j
b) addi f,h,5
   add f,f,g
c) add f,f,f
   add f,f,j
d) addi f,j,2
   add f,f,g
```

4. 假定以下 MIPS 汇编指令的所有寄存器都已经在 C 语言中定义，请分别采用相应的 C 语句实现其功能：
   a）add f,g,h　　b)add f,f,h
   c) add f,f,1　　d)sub f,$0,f

add f,g,h　　addi f,f,1

解答：a) f=g+h;　　b) f=f+h;　c)f=g+h; d) f=1-f;

5. 假定字变量 f,g,h,i,j 分别对应寄存器$s0,$s1,$s2,$s3,$s4,并且字数组 A 和 B 的起始地址分别存放在寄存器$s6,$s7 中，请分别采用 MIPS 汇编指令实现其功能：

a) f=g+h+B[4];　　b) f=g-A[B[4]];

c) f=g+h+B[1];　　d) f=A[B[g]+1];

解答：a)
```
lw $t0,16($s7)
add $s0,$s1,$t0
add $s0,$s2,$s0
```

b)
```
lw $t0, 16($s7)
sll $t0,$t0,2
add $t1,$s6,$t0
lw $t2,0($t1)
sub $s0,$s1,$t2
```

c)
```
lw $t1,4($s7)
add $s0,$s1,$t0
add $s0,$s0,$s2
```

d)
```
sll $s1,$s1,2
add $t0,$s7,$s1
lw $t0,0($t0)
addi $t0,$t0,1
sll $t0,$t0,2
add $t1,$t0,$s6
lw $s0,0($t1)
```

6. 假定变量 f,g,h,i,j 分别对应寄存器$s0,$s1,$s2,$s3,$s4,并且字数组 A 和 B 的起始地址分别存放在寄存器$s6,$s7 中，请分别采用相应的 C 语句实现以下汇编指令序列的功能：

a)
```
add $s0,$s0,$s1
add $s0,$s0,$s2
add $s0,$s0,$s3
add $s0,$s0,$s4
```

b)
```
lw $s0,4($s6)
```

c)
```
add $s0,$s0,$s1
add $s0,$s3,$s2
add $s0,$s0,$s3
```

d)
```
addi $s6,$s6,-20
add $s6,$s6,$s1
lw $s0,8($s6)
```

解答：a)f=f+g+h+i+j;　　b)f=A[1]

c) f=h+i+i;　　d) f=A[g/4-5+2]

7. 分别指出以下各指令序列执行后的结果，假定$t0=0x55555555,$t1=0x12345678

a)
```
sll $t2,$t0,4
and $t2,$t2,$t1
```

b)
```
sll $t2,$t0,4
andi $t2,$t2,-1
```

c)
```
srl $t2,$t0,3
andi $t2,$t2,0xffef
```

d)
```
sll $t2,$t0,1
or $t2,$t2,$t1
```

e)
```
sll $t2,$t0,1
andi $t2,$t2,0x00f0
```

解答：a) $t2=0x10145450　b)$t2=0x55555550 c) $t2=0xaaaa

d) $t2=0xbabefefa　e)$t2=0xa0

8. 采用汇编指令分别针对不同的 i，j 值组合完成将$t0 中的 field 域填充到$t1 中相对应的 field 域中，且$t1 的其余位都清 0.$t0 的构成如下：

31 i+1	i j+1	j 0
xx	field	xx

$t1 的构成分别如 a），b）所示：

a）

31 15+i-j	14+i-j 15	14 0
0000000…..	field	00 0000 0000 0000

b)

31 i-j	i-j-1 0
0000…..	field

1) i=7,j=2　　2)i=24,j=5

解答：1）a) andi $s0,$t0,0xf8　　b) andi $s0,$t0,0xf8

sll $t1,$s0,12　　slr $t1,$s0,3

2) a) andi $s0,$t0,0x1fe0　　b) andi $s0,$t0,0x1fe0

sll $t1,$s0,12　　slr $t1,$s0,6

9. 分别针对以下两个汇编指令段回答问题：

指令段a) loop : slt $t2,$0,$t1

bne $t2,$0,else

j done

else : addi $s2,$s2,2

addi, $t1,$t1,-1

j loop

done:

指令段 b) loop : addi $t2,$0,0xa

Loop2: addi $s2,$s2,2

addi $t2,$t2,-1

bne $t2,$0,loop2

addi $t1,$t1,-1

bne $t1,$0,loop

done:

问题：1)若指令执行前$t1=10,$s2=0,那么在分别执行以上指令序列后，$s2 的值为多少？

2）假设$s1,$s2,$t1,$t2 分别对应变量 A，B，i 和 temp，那么对应以上汇编指令序列的 C 语句分别是什么？

3）如果$t1 初始化为 N，针对以上指令序列分别需要执行多少条指令？

解答：1） a)$s2=0x14 2*$t1=20 b)$s2=0xc8 2*10*$t1

2) a) while (i>0) {　　b) do {

B=B+2;　　for(temp=0xa;temp!=0;temp--)

i--;　　B=B+2;

}　　i--;

}

while(i!=0)

3)当 N〉0 时　a)循环体内指令执行的次数为 N，循环体共 3 条指令，执行次数为 3N，循环条件判断指令执行的次数为 N+1，共 2 条指令，执行次数为 2（N+1），最后执行一次退出循环指令，因此共执行 5N+3 条指令

b）双重循环，内循环每次执行 3*10 条指令，外循环一次除了内循环之外还有另外 3 条指令，因此外循环一次共执行 33 条指令，总共执行 33N 条指令

当 N<=0 时，a) 仅执行判断及退出指令，共 3 条

b) N=0,将执行 $33*2^{32}$ 条指令；N<0,将执行 $33*(2^{32}+N)$条指令

10. 将以下C语句转换为MIPS汇编指令序列，假设变量a,b,i,j分别对应寄存器$s0,$s1,$t0,$t1,$s2保存着数组D的起始地址。

a）for(i=0;i<10;i++)
    a+=b;

b)while(a<10) {
    D[a]=b+a;
    a+=1;
}

解答：

1）

```
loop: slti $s3,$t0,10    #$s3=i<10?1:0
beq $s3,$0,exit    #$s3=0,exit
add $s0,$s0,$s1 #a+=b
addi $t0,$t0,1    #i++
j loop
exit:
```

2)

```
loop: slti $s3,$s0,10    #$s3=a<10?1:0
beq $s3,$0,exit
add $s4,$s0,$s1 #$s4=a+b
sll $t2,$s0,2 #数组偏移地址
add $t4,$s2,$t2
sw $s4,0($t4) #存储数组元素
addi $s0,$s0,1
j loop
exit:
```

11. 以下汇编语言程序段实现斐波那契(Fibonacci)数的计算，其输入的整数保存在$a0 中，输出结果保存在$v0 中。但是存在一些错误，请修正其中的错误。并画出当输入数字 4 时，该程序段每次被调用时栈的变化情况。

```
FIB: addi $sp,$sp,-12
     sw $ra,0($sp)
     sw $s1,4($sp)
     sw $a0,8($sp)
     slti $t0,$a0,1
     beq $t0,$0,L1
     addi $v0,$a0,$0
     j EXIT
L1:   addi $a0,$a0,-1
     jal FIB
     addi $s1,$v0,$0
     addi $a0,$a0,-1
     jal FIB
     add $v0,$v0,$s1
Exit:lw $ra,0($sp)
     lw $s1,4($sp)
```

```
    lw $a0,8($sp)
    addi $sp,$sp,12
    jr $ra
```

解答：斐波那契数列指的是这样一个数列：0，1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

斐波那契(Fibonacci)数的计算 C 语言函数原型 n=0 开始

```
int fib(int n,int f)
 { if(n==0)
     f=0;
     elseif(n==1)
         f=1;
         else
             f=fib(n-1,f)+fib(n-2,f);
     return f;
    }
```

正确的

```
FIB: addi $sp,$sp,-12
    sw $ra,0($sp)
    sw $s1,4($sp)
    sw $a0,8($sp)
    slti $t0,$a0,2
    beq $t0,$0,L1
    add $v0,$a0,$0
    j Exit
L1:  addi $a0,$a0,-1
    jal FIB
    add $s1,$v0,$0
    addi $a0,$a0,-1
    jal FIB
    add $v0,$v0,$s1
Exit:lw $ra,0($sp)
    lw $s1,4($sp)
    lw $a0,8($sp)
    addi $sp,$sp,12
    jr $ra
```

错误的

```
FIB: addi $sp,$sp,-12
    sw $ra,0($sp)
    sw $s1,4($sp)
    sw $a0,8($sp)
    slti $t0,$a0,1
    beq $t0,$0,L1
    addi $v0,$a0,$0      ;N=0，1 直接得到结果
    j EXIT
L1:  addi $a0,$a0,-1
    jal FIB
    addi $s1,$v0,$0      ；暂存 n-1 的结果
    addi $a0,$a0,-1
     jal FIB
    add $v0,$v0,$s1      ；将 n-1 与 n-2 的结果相加
Exit:lw $ra,0($sp)
    lw $s1,4($sp)
    lw $a0,8($sp)
    addi $sp,$sp,12
    jr $ra
```

```
[00400024] 34040004  ori $4, $0, 4; 1: li $a0,4
[00400028] 0c10000c  jal 0x00400030 [FIB]; 2: jal FIB
[0040002c] 0810001f  j 0x0040007c [exit]; 3: j exit
[00400030] 23bdfff4  addi $29, $29, -12; 4: addi $sp,$sp,-12
[00400034] afbf0000  sw $31, 0($29); 5: sw $ra,0($sp)
[00400038] afb10004  sw $17, 4($29); 6: sw $s1,4($sp)
[0040003c] afa40008  sw $4, 8($29); 7: sw $a0,8($sp)
[00400040] 28880003  slti $8, $4, 2; 8: slti $t0,$a0,2
[00400044] 11000003  beq $8, $0, 12 [L1-0x00400044]; 9: beq $t0,$0,L1
[00400048] 00801020  add $2, $4, $0; 10: add $v0,$a0,$0
```

```
[0040004c] 0810001a  j 0x00400068 [Exit]; 11: j Exit
[00400050] 2084ffff  addi $4, $4, -1; 12: addi $a0,$a0,-1
[00400054] 0c10000c  jal 0x00400030 [FIB]; 13: jal FIB
[00400058] 00408820  add $17, $2, $0; 14: add $s1,$v0,$0
[0040005c] 2084ffff  addi $4, $4, -1; 15: addi $a0,$a0,-1
[00400060] 0c10000c  jal 0x00400030 [FIB]; 16: jal FIB
[00400064] 00511020  add $2, $2, $17; 17: add $v0,$v0,$s1
[00400068] 8fbf0000  lw $31, 0($29); 18: lw $ra,0($sp)
[0040006c] 8fb10004  lw $17, 4($29); 19: lw $s1,4($sp)
[00400070] 8fa40008  lw $4, 8($29); 20: lw $a0,8($sp)
[00400074] 23bd000c  addi $29, $29, 12; 21: addi $sp,$sp,12
[00400078] 03e00008  jr $31; 22: jr $ra
```

Fib 函数嵌套调用栈变化过程

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
$sp	7ffff9c0	0040002c(&j 0x0040007c [exit])

首次调用 fib(4) $a0=4

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
$sp	7ffff9b4	00400058(& add $17, $2, $0)

嵌套调用 fib(3) $a0=3

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
	7ffff9b4	00400058(& add $17, $2, $0)
	7ffff9b0	00000002($a0)
	7ffff9ac	00000000($s1)
$sp	7ffff9a8	00400058(&add $17, $2, $0)

嵌套调用 fib(2) $a0=2

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
	7ffff9b4	00400058(& add $17, $2, $0)

	7ffff9b0	00000002($a0)
	7ffff9ac	00000000($s1)
	7ffff9a8	00400058(&add $17, $2, $0)
	7ffff9a4	00000001($a0)
	7ffff9a0	00000000($s1)
$sp	7ffff99c	00400058(&add $17, $2, $0)

嵌套调用 fib(1) $a0=1

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
	7ffff9b4	00400058(& add $17, $2, $0)
	7ffff9b0	00000002($a0)
	7ffff9ac	00000000($s1)
$sp	7ffff9a8	00400058(&add $17, $2, $0)

fib(1)返回 执行 *add $s1,$v0,$0 前*

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
	7ffff9b4	00400058(& add $17, $2, $0)
	7ffff9b0	00000002($a0)
	7ffff9ac	00000000($s1)
	7ffff9a8	00400058(&add $17, $2, $0)
	7ffff9b0	00000000($a0)
	7ffff9ac	00000001($s1)
$sp	7ffff9a8	00400064(&**add $2, $2, $17**)

嵌套调用 fib(0)

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
	7ffff9b4	00400058(& add $17, $2, $0)
	7ffff9b0	00000002($a0)
	7ffff9ac	00000000($s1)
$sp	7ffff9a8	00400058(&add $17, $2, $0)

fib(0)返回

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
$sp	7ffff9b4	00400058(& add $17, $2, $0)

fib(2)返回，执行 add $s1,$v0,$0 前

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
	7ffff9b4	00400058(& add $17, $2, $0)
	7ffff9b0	00000001($a0)
	7ffff9ac	00000001($s1)
$sp	7ffff9a8	00400064(&add $17, $2, $0)

fib(2)返回，再次执行 jal FIB 后,此时$a0=1 $s1=1

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000003($a0)
	7ffff9b8	00000000($s1)
$sp	7ffff9b4	00400058(& add $17, $2, $0)

fib(2)返回 $pc=00400064

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
$sp	7ffff9c0	0040002c(&j 0x0040007c [exit])

fib(3)返回，执行 add $s1,$v0,$0 前

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000002($a0)
	7ffff9b8	00000002($s1)
$sp	7ffff9b4	00400064(& add $17, $2, $0)

fib(3)返回，再次执行 jal FIB(2) 后,此时$a0=2 $s1=2

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)

	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000002($a0)
	7ffff9b8	00000002($s1)
	7ffff9b4	00400064(& add $17, $2, $0)
		00000001($a0)
		00000002($s1)
$sp		00400058(& add $17, $2, $0)

fib(3)返回，再次执行 jal FIB (1)后,此时$a0=1 $s1=2

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000002($a0)
	7ffff9b8	00000002($s1)
$sp	7ffff9b4	00400064(& add $17, $2, $0)

fib(3)返回 执行 add $s1,$v0,$0 前

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000002($a0)
	7ffff9b8	00000002($s1)
	7ffff9b4	00400064(& add $17, $2, $0)
		00000000($a0)
		00000001($s1)
$sp		00400064(& add $17, $2, $0)

fib(3)返回，再次执行 jal FIB (0)后,此时$a0=0 $s1=3

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])
	7ffff9bc	00000002($a0)
	7ffff9b8	00000002($s1)
	7ffff9b4	00400064(& add $17, $2, $0)

fib(3)返回中的 jal FIB (0)返回,

	7ffff9c8	00000004($a0)
	7ffff9c4	00000000($s1)
	7ffff9c0	0040002c(&j 0x0040007c [exit])

fib(3)返回中的 jal FIB (1)返回　　fib(4)返回，栈恢复

12．若PC=0x00000000，请分析是否可以仅采用条件跳转指令或无条件跳转指令跳转到下表所示地址?若能，分别需要采用多少条跳转指令？

跳转地址	条件跳转次数	无条件跳转次数

a. 0x00001000	1（正向跳转）	1
b. 0xFFFC0000	3(负向跳转)	否

解答：条件跳转指令中的立即数是 16 位符号数，其取值范围为-2^{15}~2^{15}-1，
新的 PC 的值=PC 的原始值+16 位立即数*4
因此当跳转指令的 PC=0 时，实际上 PC 的原始值已经变为 4，一次正向跳转可跳转到的地址范围为：0x00000004~0x00020000 ，一次负向跳转可跳转到的地址范围为 0xFFFE0004~0x0,当跳转到 0x FFFE0004，再次最远距离的跳转将跳转到 0xFFFC0008，此地址仍然大于 0xFFFC0000，因此还需要再进行一次跳转，共需跳转 3 次。
无条件跳转由于无法改变 PC 的最高 4 位，因此 a）可以实现，b）无法实现。

13．利用 MIPS 汇编语言编程实现以下算式功能，并将结果输出到屏幕中。
1）通过键盘输入两个字类型的 10 进制数据，并且将它们的和、差以 10 进制形式输出到屏幕上
2）通过键盘输入两个字类型的 16 进制数据，并且将它们的和、差以 16 进制形式输出到屏幕上
3）分别对两个半字类型的符号数数组进行从大到小的排序，要求采用子程序实现单一数组数据的排序，主程序调用子程序分别对两个数组排序，并且分别将结果输出到屏幕上。

解答：1）2）主要考察系统功能调用的熟悉程度，由于 QTspim 可以直接支持 10 进制整数的输入输出，因此 1）可以直接进行输入输出，但是 2）需要进行转换，字符串到 16 进制以及 16 进制到字符串的转换。

1）程序源码：

```
.data
prompt: .asciiz    "\nplease input a number:"
sumresult: .asciiz "\nthe sum of these two numbers is :" #定义提示信息
difresult: .asciiz "\nthe difference between these two numbers is :"
.text
main:
li $v0,4
la $a0,prompt
syscall              #输出提示信息
li $v0,5
syscall              #输入十进制数
add $s0,$v0,$0       #暂存输入结果
li $v0,4
la $a0,prompt
syscall              #输出提示信息
li $v0,5
syscall              #输入十进制数
add $s2,$s0,$v0      #暂存和的结果
sub $s1,$s0,$v0
li $v0,4
la $a0,sumresult
syscall              #输出提示信息
```

```
li $v0,1
add $a0,$s2,$0
syscall
li $v0,4
la $a0,difresult
syscall                 #输出提示信息
li $v0,1
add $a0,$s1,$0
syscall
li $v0,10
syscall
```

2）程序源码：

```
.data
number1: .space 11
number2: .space 11
prompt: .asciiz    "\nplease input a number:"
sumresult: .asciiz "\nthe sum of these two numbers is :" #定义提示信息
difresult: .asciiz "\nthe difference between these two numbers is :"
err: .asciiz "input error"
sum: .word 0
dif: .word 0
str: .byte 0x20,0x30,0x78 #16 进制显示前缀
hex: .space 8 #预留 8 个内存单元
        .byte 0x00 #字符串结束符
.text
main:
li $v0,4
la $a0,prompt
syscall                 #输出提示信息
la $a0,number1
li $a1,11
li $v0,8
syscall                 #输入 16 进制数字符串
la $a0,number1
jal asctohex
add $s0,$v0,$0       #暂存输入结果
li $v0,4
la $a0,prompt
syscall                 #输出提示信息
la $a0,number2
li $a1,11
li $v0,8
syscall                 #输入 16 进制数字符串
la $a0,number2
```

```
    jal asctohex
    add $s2,$s0,$v0     #暂存和的结果
    la $t0,sum
    sw $s2,0($t0)
    sub $s1,$s0,$v0
    la $t0,dif
    sw $s1,0($t0)       ##暂存差的结果
    li $v0,4
    la $a0,sumresult
    syscall             #输出提示信息
    la $t0,sum
    lw $v0,0($t0)
    jal hextoasc        #输出和的 16 进制数
    li $v0,4
    la $a0,difresult
    syscall             #输出提示信息
    la $t0,dif
    lw $v0,0($t0)
    jal hextoasc        #输出差的 16 进制数
    li $v0,10
    syscall
asctohex:               #字符串首地址为输入参数，保存在$a0 中
        addi $a0,$a0,2
        li $v0,0
    rep1:   lb $t0,0($a0)
        li $t2,0x0a
        beq $t0,$t2,exit #比较是否碰到输入结束字符
        addi $t0,$t0,-48 #0~9 字符转换
        slti $t1,$t0,10
        beq $t1,$0,upper
        j convert
    upper:  addi $t0,$t0,-7 #大写字符转换
            slti $t1,$t0,16
        beq $t1,$0,lower
        j convert
    lower:  addi $t0,$t0,-32 #小写字符转换
    convert:sll $v0,$v0,4 #合并数字结果
        or $v0,$v0,$t0
        addi $a0,$a0,1 #调整地址指针
        j rep1
    exit: jr $ra            #转换结果保存在$v0 中
hextoasc:               #输出 16 进制数保存在$v0 中
        la $s0,hex
        addi $s0,$s0,7
```

```
        li $s1,8 #初始化
rep2:       andi $s2,$v0,0xf #获取低 4 位
        addi $s2,$s2,0x30 #转换为 ASCII 码
        slti $t0,$s2,0x3a
        bne $t0,$0,less
        addi $s2,$s2,0x07
less:       sb $s2,0($s0) #存储转换结果
        addi $s0,$s0,-1
        addi $s1,$s1,-1
        srl $v0,$v0,4
        slti $t0,$s1,1
        beq $t0,$0,rep2
        li $v0,4 #显示整个转换后的字符
        la $a0,str
        syscall
        jr $ra
```

3）子程序入口参数$t0 为数组首地址，$t1 为数组长度，子程序采取嵌套循环的方式实现，内循环找到剩余数据的最小数，并保存在首地址，外循环比较 N-1 次。

```
.data
.align 2
array1: .half 34,78,12,4,7,9
array2: .half 0x12,0xfe,0x4,0xf,0xa
end:
prompt: .asciiz "\n"
prompt1: .asciiz ""
.text
main:
        la $t0,array1
        la $t1,array2
        sub $t1,$t1,$t0
        srl $t1,$t1,1 #计算数组 array1 的长度，保存到$t1 中
        jal disarray #调用显示子程序，显示原始数组序列
        la $t0,array1
        la $t1,array2
        sub $t1,$t1,$t0
        srl $t1,$t1,1 #计算数组 array1 的长度，保存到$t1 中
        jal inorder #调用排序子程序
        la $t0,array1
        la $t1,array2
        sub $t1,$t1,$t0
        srl $t1,$t1,1 #计算数组 array1 的长度，保存到$t1 中
        jal disarray ##调用显示子程序，显示排序后数组序列
        la $t0,array2
        la $t1,end
```

```
    sub $t1,$t1,$t0
    srl $t1,$t1,1 #计算数组 array2 的长度，保存到$t1 中
    jal disarray#调用显示子程序，显示原始数组序列
    la $t0,array2
    la $t1,end
    sub $t1,$t1,$t0
    srl $t1,$t1,1#计算数组 array2 的长度，保存到$t1 中
    jal inorder #调用排序子程序
    la $t0,array2
    la $t1,end
    sub $t1,$t1,$t0
    srl $t1,$t1,1 #计算数组 array2 的长度，保存到$t1 中
    jal disarray #调用显示子程序，显示排序后数组序列
    li $v0,10
    syscall
disarray:                   #显示数组子程序 $t0 数组首地址 $t1 元素个数
    lh $a0,0($t0) #$t0 为数组首地址，$t1 为数组长度
    li $v0,1
    syscall
    la $a0,prompt1      #显示空格
    li $v0,4
    syscall
    addi $t1,$t1,-1
    slt $t2,$0,$t1
    beq $t2,$0,finishdisplay
    addi $t0,$t0,2          #修改地址指针，半字类型数据地址间隔为 2
    j disarray
finishdisplay:
    la $a0,prompt           #显示换行
    li $v0,4
    syscall
    jr $ra
inorder:                    #排序子程序 $t0 数组首地址 $t1 元素个数
    slti $t2,$t1,2          #当数组元素个数为 1 时，退出子程序
    bne $t2,$0,exit
    add $t3,$t0,$0      #准备类循环，首地址$t3,比较次数$t4
    addi $t4,$t1,-1
    lh $a0,0($t3) #取第一个数据，且假定其为最小值
inloop: lh $s0,2($t3)#取下一个数据
    slt $t2,$s0,$a0   #$t2 临时标志，比较大小
    beq $t2,$0,next
sh $a0,2($t3)  #若下一个元素小于原最小值，则将原最小值存入这个元素的地址
    add $a0,$s0,$0 #较小的数交换到$a0
 next: addi $t4,$t4,-1 #比较次数减 1
```

```
        addi $t3,$t3,2 #地址指针修改到下一个元素
        bne $t4,$0,inloop
        sh $a0,0($t0)        #存储最小值到首地址
        addi $t0,$t0,2       #修改首地址为下一个元素的地址
        addi $t1,$t1,-1      #数组元素个数减 1
        j inorder
    exit: jr $ra
```

translate each of the following pseudocode expressions intoMIPS assembly language and test it in qtspim:

(a) t3 = t4 + t5 – t6;
(b) a0 = &array;
(c) sp = sp –16;
(d) t8 = Mem(a0);
(e) Mem(a0+ 16) = 32768;
(f) If (t0 < 0) then t7 = 0 – t0 else t7 = t0;
(g) while (t0 != 0) { s1 = s1 + t0; t2 = t2 + 4; t0 = Mem(t2) };
(h) for (t1 = 99; t1 > 0; t1=t1 -1) v0 = v0 + t1;
(i)int ipowr_for(int x, unsigned p) {

```
    int result;
    for (result = 1; p != 0; p = p>>1) {
        if (p & 0x1)
        result *= x;
        x = x*x;
        }
    return result;
    }
```

解答：

```
    .data
    abc: .asciiz "abc"
    array: .space 400
    .text
    main:
    add $t0,$t4,$t5 #    question (a)
    sub $t3,$t0,$t6

    la $a0,array # question (b)

    addi $sp,$sp,-16 # question (c)

    lw $t8,0($a0) # question (d)

    ori $t0,$zero,32768 # question (e)
    sw $t0,16($a0)
```

```
bgez $t0,else # question (f)
sub $t7,$zero,$t0
j exit
else: add $t7,$zero,$t0
exit:

j test # question (g)
loop: add $s1,$s1,$t0
addi $t2,$t2,4
lw $t0,0($t2)
test: bne $t0,$zero,loop
exit1:

li $t1,99 # question (h)
j test1
loop1: add $v0,$v0,$t1
    addi $t1,$t1,-1
test1: bgtz $t1,loop1
exit2:

    li    $t0,1 # int ipowr_for(int x, unsigned p)
    j test2
loop2:    andi $s1,$s2,1
    beq $s1,$zero,mullabel
    mult $t0,$t1
    mflo $t0
mullabel:mult $t1,$t1
    mflo $t1
    srl $s2,$s2,1
test2: bne $s2,$zero,loop2
exit3:
```

14. 已知一子程序如下，且编译器对数组地址空间分配采用从栈低地址往高地址方向分配的原则：即首先从栈的最低地址分配a[2]的内存空间，然后再分配d[1]的内存空间。

```
double fun(int i)
{
volatile long int a[2];
volatile double d[1] = {3.14};
a[i] = 1073741824; /* Possibly out of bounds */
return d[0];
}
```

试说明执行 fun(0), fun(1), fun(2), fun(3)的结果，并说明原因。

解答：

double 3.14 的 16 进制（64 位）表示为：0x40091eb851eb851f,

long 1073741824 的 16 进制（32 位）表示为：0x40000000

题目含义为栈的伸展方向为从低地址到高地址的方向伸展，因此每次进入子程序时栈的分配规则为

变量名称	内存地址	内存值
a[0]	低地址	
a[1]		
d[0]		
	高地址	

当调用 fun(0)时，a[0]初始化

变量名称	内存地址	内存值
a[0]	低地址	0x40000000
a[1]		
d[0]		0x40091eb851eb851f
	高地址	

当调用 fun(1)时，a[1]初始化

变量名称	内存地址	内存值
a[0]	低地址	

a[1]		0x40000000
d[0]		0x40091eb851eb851f
	高地址	

当调用 fun(2)时，a[2]初始化，此时 a[2]将覆盖 d[0]的前 4 个字节，假定采用小字节序

变量名称	内存地址	内存值
a[0]	低地址	
a[1]		
d[0]		0x40000000
		0x40091eb8
	高地址	

当调用 fun(2)时，a[2]初始化，此时 a[2]将覆盖 d[0]的前 4 个字节，假定采用小字节序

变量名称	内存地址	内存值
a[0]	低地址	
a[1]		
d[0]		0x51eb851f

		0x40000000
	高地址	

因此前两次调用 fun(0),fun(1)时，都是返回没有改变的 d[0]的值，即为 3.14 计算机表示之后的有误差的结果 3.1400000000000001，fun(2)返回的结果为修改了低 32 位的 d[0]的结果 3.1399998664855957（0x40091eb840000000）。fun(3) 返回的结果为修改了高 32 位的 d[0]的结果 2.0000006103515626（0x4000000051eb851f）

为方便大家作实验，将题目做如下修改：

14. 已知一子程序如下，且计算机系统栈的伸展方向为向低地址方向伸展。试说明在小字节序计算机系统中执行fun(0), fun(1), fun(2), fun(3)的结果，并说明原因。

```
double fun(int i)
{
volatile double d[1] = {3.14};
volatile long int a[2];
a[i] = 1073741824; /* Possibly out of bounds */
return d[0];
}
```

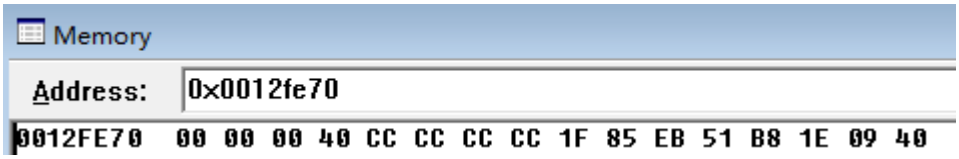
解答：修改题目之后，栈的内存映像仍然与上面相同：编译器从栈底向栈顶逐步为数组变量分配内存空间，而数组各元素的地址为低地址到高地址方向分配。

VC 实验代码：

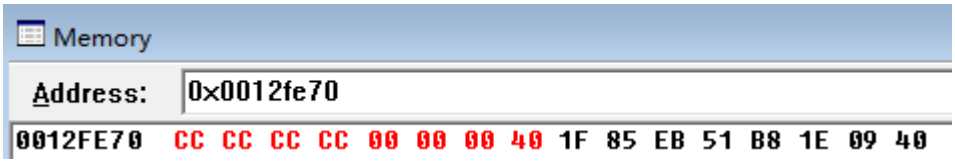
```
#include "stdafx.h"
double fun(int i)
{
volatile double d[1] = {3.14};
volatile long int a[2];

a[i] = 1073741824; /* Possibly out of bounds */
return d[0];
}


int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    // TODO: Place code here.
    double a,b,c,d;
    a=fun(0);
    b=fun(1);
    c=fun(2);
    d=fun(3);
    return 0;
}
```

Name	Value
a	0x0012fe70
[0x0]	0x40000000
[0x1]	0xcccccccc
d	0x0012fe78
[0x0]	3.1400000000000001

fun(0)



fun(1)



fun(2)

fun(3)