1.Which field determines the operation of an R-type instruction? function

2.Suppose the program counter, PC, has the value 0x00001234. What is the value of

PC after executing the following branch instruction?

beq $0, $0, 4

PC=PC+4+4*4=0x1248

3.Without making any assumptions about the contents of registers or memory, which of the following operations ***cannot*** be performed by a ***single*** MIPS instruction and why? D

没有指令可以将两个寄存器相加的结果作为内存地址。

(A) Memory[R[rs] + 0x1000] ← 0
(B) Memory[R[rs]] ← 0
(C) Memory[0x1000] ← 0
(D) Memory[R[rs] + R[rt]] ← 0
(E) R[rt] ← Memory[R[rs] + 0x1000]

4.Suppose you execute the following instruction sequence:

```
addi $t0, $0, -1

sll $t0, $t0, 16

srl $t1, $t0, 16

sra $t2, $t0, 16
```

What are the values of $t0, $t1 and $t2 after execution (in binary or hex)?

$t0 = 0xFFFF0000 $t1 = 0x0000FFFF $t2 = 0xFFFFFFFF

5.Assuming the standard MIPS procedure calling conventions, if we see an instruction of the form lw $t0, 4($fp), the program is most likely D

(A) accessing the return address
(B) accessing one of its own local variables
(C) accessing a local variable belonging to its caller
(D) accessing its fifth argument
(E) none of the above

6.For the next two questions, consider the following assembly language procedure:

```
foo:
      addi  $sp, $sp, -4
      sw    $ra, 0($sp)
      beq   $a0, $0, L1
      addi  $a0, $a0, -1
      add   $a1, $a1, $a1
      jal   foo
      add   $a1, $v0, $0
L1:   add   $v0, $a1, $0
      lw    $ra, 0($sp)
      addi  $sp, $sp, 4
      jr    $ra
```

Suppose there is a procedure called main which calls foo(4,3). [Assume that main places 4 in $a0, and 3 in $a1 before calling foo.]

a.What is the entire sequence of calls to foo,starting with foo(4,3)?

foo(4,3) 、foo(3,6)、 foo(2,12)、 foo(1,24) 、foo(0,48)

b.What is the final value returned to main? foo(0,48)=48

7.Given the following MIPS code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your green sheet!):

```
0x002cff00: loop: addu $t0, $t0, $t0    | 000000 | 01000 | 01000 | 01000 | 00000 | 100001 |
0x002cff04:      jal  foo               | 000101 |  00000011000000000000000001    |
0x002cff08:      bne  $t0, $zero, loop  | 000101 | 01000 |00000|  1111111111111101  |
...
0x00300004: foo:  jr $ra   $ra=0x002cff08
```

8.Writing MIPS Functions

Here is a general template for writing functions in MIPS:

```
FunctionFoo:    # PROLOGUE
# begin by reserving space on the stack addiu
$sp, $sp, -FrameSize

# now, store needed  registers sw
$ra, 0($sp)
sw $s0, 4($sp)
- - -
# BODY
- - -
```

```
# EPILOGUE
# restore  registers
lw $s0  4($sp)
lw  $ra  0($sp)

# release stack  spaces
addiu $sp, $sp,  FrameSize

# return to normal execution
 jr  $ra
```

8.Translate the following C code for a recursive function into a callable MIPS function.

```
// Finds the sum of numbers 0 to N int  sum_numbers(int  N) {
    int  sum  = 0

    if (N==0) {
        return 0;
    }  else  {
        return  N  +  sum_numbers(N  -  1);
    }
}
```

```
fact:
addiu $sp,$sp,-8   //修改栈顶指针，预留 8 个空间保存$a0,$ra 的值
sw $ra,4($sp)      //保存$ra 的值到栈中
sw $a0,0($sp)      //保存$a0 的值到栈中
slti $t0,$a0,1     //比较 n 是否小于 1，小于 1 则$t0 置 1，否则置 0
beq $t0,$zero,loop   //n>=1 时进入循环
addi $v0,$zero,0     //n=0 时 return 0
addiu $sp,$sp,8      //恢复栈顶指针
jr $ra               //返回
loop:
addiu $a0,$a0,-1     //n>=1 时，计算 n-1
jal fact             //递归，结果存在$v0 中
lw $a0,0($sp)        //从栈中获取当前子程序的入口
lw $ra,4($sp)        //从栈中获取当前子程序的返回地址
addiu $sp,$sp,8      //恢复栈顶指针
addu $v0,$v0,$a0  //return n+sum_numbers（n-1）
jr $ra               //返回
```