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

***The FUNC field.***

**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

***0x00001234 + 4 + 4 \* 4 = 0x00001248***

**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?

* 1. Memory[R[rs] + 0x1000] ← 0
  2. Memory[R[rs]] ← 0
  3. Memory[0x1000] ← 0
  4. Memory[R[rs] + R[rt]] ← 0
  5. R[rt] ← Memory[R[rs] + 0x1000]4

***(D) All of the storing instruction use R[rt] to represent the value to be stored, so R[st] + R[rt] must be performed by instructions like add.***

**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 = (0 + 0xFFFF) << 16 = 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

* 1. accessing the return address
  2. accessing one of its own local variables
  3. accessing a local variable belonging to its caller
  4. accessing its fifth argument
  5. none of the above

***(D)***

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

|  |  |
| --- | --- |
| foo: | $sp, $sp, -4 |
| addi |
| 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?

***48***

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

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x002cff00: | loop: | addu | $t0, | $t0, $t0 | | |  | | |  | | | | | | 0x21 | | |
| 0x002cff04: |  | jal | foo |  | | |  | | |  |  |  | | |
| 0x002cff08: |  | bne | $t0, | $zero, loop | | |  | | |  | | | | | | |
| ... |  |  |  |  |  |  |  |  |  |  |  |

0x00300004: foo: jr $ra $ra= 0x002cff08

# *0 8 8 8 0 32*

# *3 0x0030004*

# *5 8 0 0x002cff00*

# *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);

}

}

***sum\_numbers:***

***addiu $sp, $sp, -8***

***sw $ra, 4($sp)***

***sw $a0, 0($sp)***

***bne $a0, $zero, L***

***addi $v0,$zero,1***

***addi $sp,$sp,8***

***jr $ra***

***L:***

***add $a0,$a0,-1***

***jal sum\_numbers***

***lw $a0, 0($sp)***

***lw $ra, 4($sp)***

***mul $v0,$a0,$v0***

***addiu $sp, $sp, 8***

***jr $ra***