CS61C MIDTERM EXAM
FALL 1999

1. Facts, Figures and Bit [12 points]

a. Describe the set of numbers that are represented by a 64-bit two's complement integer.
b. Add the following 16-bit integers by hand, show binary result and translate it to hexadecimal.

0001011011100101
+0011101001101100

c. Multiply the following 16-bit unsigned integers by hand, show binary result and translate to hexadecimal

0000000001101111
+0000000000001010

d. Encode the value 17.25_{10} according to the single precision IEEE Floating-Point standard and show its representation in hexadecimal.
e. For each of the following utilities, specify what it takes as input and what it produces as output. Describe one key function it performs in this translation.

- Compiler
- Assembler
- Linker
- Loader

f. The C character string "UCB" appears in memory. The first character is word aligned on a little endian machine. Given that the ASCII code for 'A' is 65, what binary bit pattern appears in the word.

2. Logical Operations [5]

Write a sequence of no more than six MIPS instruction that extracts bits 17:11 of register $s0 and inserts them into bits 8:2 of register $s1, leaving all the remaining bits of $s1 unchanged. You may use t-registers as temporaries.

3. Assembler/Instruction Format [5]

Using the opcode and register map tables on page A-54 , assemble the following MIPS instructions into binary. Show the position of each field by drawing a box around the corresponding bit positions.

Address	Instruction	Instruction
0x400000	addi $a0, $a0, -4	
0x400004 L0:	bne $s1, $t2, L1	
0x400008	lw $s2, 128($sp)	
0x40000c	j L0	
0x400010 L1:	subu $v0, $a0, $s0	

4. Correct Assembly Language [8]

Consider the folloiwng MIPS assemble language routine. (The numbers oin the left are just line numbers to help in your answer.) `foo` takes two integer arguments. The caller of `foo` and its callee `bar` follow the MIPS procedure call conventions. Assume VAR1 has been declared in the `.data` section with the `.word` directive.

```
1 foo: addi $sp, $sp, -20
2      sw   $s0, 16($sp)
3      sw   $s1, 12($sp)
4      la   $t0, VAR1
5      lw   $t0, 0($t0)
6      add  $t1, $a1, $a0
7      addi $s0, $t1, 10
8      add  $s2, $s0, $t1
9      add  $a0, $0, $s2
10     jal  bar
11     add  $t2, $t1, $v0
12     add  $s1, $t2, $a1
13     add  $v0, $0, $s1
14     lw   $s1, 12($sp)
15     lw   $s0, 16($sp)
16     addi $sp, $sp, 20
17     jr   $ra
```

a. List below four bugs that are present in the code
b. For each of these bugs, explain in one sentence either (i) why it will definitely cause the program to not work or (ii) under what condition will the program work correctly, in spite of the bug.

5. Pointers, Arrays, and strings [5 points]

Given the following (correct) C declarations.

```
char foo = 'A', garply[] = "MIPS", bar = 'C';
char *phi = &foo, *beta = phi, *gamma = garply;
```

In the following, some of the statements are incorrect or illegal; cross out any such bad statements. Show in the spaces provided what the remaining print statements will print when the program is executed.

```
printf("%c", *beta);
printf("%c", phi);
printf("%c", *gamma);
printf("%s", bar);
beta = garply;
printf("%c", *(garply + 2));
printf("%s", &garply[1]);
garply = phi;
printf("%c", *beta);
```

6. Compile the following C code into MIPS [10 points]

```
struct Node {
        int data;
        struct Node *next;
};

int SumList (struct Node *nptr) {
        if (nptr == NULL) return 0;
        else return (nptr->data + SumList (nptr->next));
}
```

The idea here is that Nodes form a singly linked list, much like what you dealt with in cs61a. The Node structure is represented as two consecutive words, one for each field. The `data` field contains an integer data value and the `next` field contains a pointer to another. The -> notation is used to reference fields of the structure pointed to by a pointer variable. `nptr->data` is the same as `(*nptr).data` - it accesses the data field of the structure pointed to by `nptr`.

Your code must contain meaningful comments and adhere to the MIPS calling convention and register usage convention. You are allowed to use pseudoinstructions to make it more readable. It should be clean and well structured. It needs to be right, not optimal, but your answer cannot be longer than 20 instructions.
