CSE 31

Midterm Sample

Time : 120 minutes

Name:

Problem	Points	Max Points
1		70
2		40
3		30
4		60
Total		200

1 : [70 pts] Number Representation

a) [20 pts] Fill in the following table :

Decimal (Base 10)	Binary (Base 2)	Hexadecimal (Base 16)
	1	
		0x3
101		
	0001 0010	
		0xC953
131071		

b) [20 pts] Fill in the following table :

Binary	Unsigned	Signed	1’s Complement	2’sComplement	Biased
0000 1111					
0101 0101					
1010 1010					
1111 1111					

c) [20 pts] Fill T/F in the following table :

Property	Unsigned	Signed	1’s Comp	2’s Comp	Biased
Can represent positive numbers					
Can represent negative numbers					
Has more than one representation for 0					
Use the same addition process as unsigned					

d) [5 pts] What is the value in decimal of the most negative 16-bit 2’s complement integer?

e) [5 pts] What is the value in decimal of the most positive 16-bit signed integer?

2 : [40 pts] C Code

The following program is compiled and run on a MIPS computer.

```
1       int main() {
2                 int i;
3                int four_ints[4];
4                char* c;
5
6                for(i=0; i<4; i++) four_ints[i] = 2;
7
8                c = (char*)four_ints;
9                for(i=0; i<4; i++) c[i] = 1;
10
11               printf("%x\n", four_ints[2]);
12      }
```

a) [5 pts] What does it print out? (The “%x” in printf is used print out a word in hexadecimal
format.)

b) [10 pts] If we change the 2 on line 11 to a 0, then recompile and run, what would be printed? (hint: Consider how many hex digits are in an int and in a character, ie not the same as bytes)

c) [20 pts] The following function should allocate space for a new string, copy the string from the passed argument into the new string, and convert every lower-case character in the **new** string into an upper-case character (do not modify the original string). Fill-in the blanks and the body of the for() loop:

```
        char* upcase(char* str) {
                char* p;
                char* result;

                result = (char*) malloc(____________________________);

                strcpy(_____________, ______________);

                for( p=result; *p!='\0'; p++ ) {
/* Fill-in 'A' = 65, 'a' = 97, 'Z' = 90 , 'z' = 122 */






                }
                return result;
        }
```

d) [5 pts] Consider the code below. The upcase name() function should convert the ith name to upper case by calling upcase by ref, which should in turn call upcase(). Complete the implementation of upcase_by_ref. You may not change any part of upcase_name.

```
                void upcase_by_ref( char** n ) {  /* Fill-in */




                }

                void upcase_name(char* names[], int i) { /* No not touch */
                        upcase_by_ref( &(names[i]) );
                }
```

3 : [30 pts] MIPS Translation

The program below is written using the MIPS instruction set. It is loaded into memory at address 0xF000000C (all instruction memory addresses are shown below).

```
F000000C loop: addi $1, $1, -1
F0000010       beq $1, $0, done
F0000014       j loop
F0000018 done:
```

Write out the number (in decimal) for each field (opcode, rs, rt etc) and the final bits representation in Hex. (Be sure to put down all your steps for partial credit in case you make some mistake at any steps)

addi :

beq :

j :

4 : [60 pts] MIPS Coding

a) [10 points] The original MIPS processor did not support multiplication; compilers were expected to break down multiplication and division into simpler operations. Even on newer MIPS processors (that have the MUL instruction), compilers sometimes still do this to improve performance.

Consider the following C function:

```
int foo(int x) {
     return x*257; }
```

Write the corresponding MIPS assembly code below. You may **not** use any form of MUL. Your answer should use as **few** a number of instructions as possible

```
foo:





     # return value should be in $v0
     jr $ra
```

(b) [10 pts] Multiplication is more difficult when neither argument is known at compile time. The general procedure for achieving multiplication of two unsigned numbers is to use a series of shift and add operations (think about how long-hand multiplication works). The following assembly code multiplies two unsigned numbers, $a0 and $a1, leaving the result in $v0. Assume that the result is sufficiently small that it fits in a single register.

Fill in the missing lines .

```
          addi $v0, $zero, $zero   # clear $v0
loop:     beq $a1, $zero, done     # if $a1==0, we are done
          andi $t0, $a1, 1         # check bottom bit of $a1...
          beq $t0, $zero, skip     # ...if it is 0, skip over
                                   # the next instruction

                                   # fill me in!

skip:     srl $a1, $a1, 1          # shift $a1 to the right

                                   # fill me in!
          j loop
```

c) [40 pts] Below is a recursive version of the function BitCount. This function counts the number of bits that are set to 1 in an integer. Your task is to translate this function into MIPS assembly code. The parameter x is passed to your function in register $a0. Your function should place the return value in register $v0.

```
int BitCount(unsigned x) {
     int bit;
     if (x == 0)
          return 0;
     bit = x & 0x1;
     return bit + BitCount(x >> 1);
}
```

Translate this procedure into MIPS assembly language, following our standard conventions for register use (arguments in registers, not stack, whenever possible).