CSE 31

Midterm 1

Time : 75 minutes

Name:

1 : Number Representation

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

Decimal (Base 10)	Binary (Base 2)	Hexadecimal (Base 16)
1	1	0x1
3	11	0x3
101	101 – 64 = 37 37 – 32 = 5 110 0101	0x65
18	0001 0010	0x12
12*16^3 + 9*16^2 + 5*16 +3 =51539	1100 1001 0101 0011	0xC953
131071	1 1111 1111 1111 1111	0x1FFFF

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

Binary	Unsigned	Signed	1's Complement	2'sComplement	Biased
0000 1111	15	15	15	15	15 - 2^(n-1)-1 15 - 127 = -112
0101 0101	64+16+4+1=85	85	85	85	85-127 = -42
1010 1010	128+32+8+2 = 170	-(32+8+2)= -42	- (0101 0101) = -85	- (0101 0110) = -86	170-127 = 43
1111 1111	256-1= 255	-(128-1) = -127	- (0000 0000) -0, 0	- (0000 0001) -1	255 - 127 = 128

Note: 1's/2's complement only works with NEGATIVE numbers

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

Property	Unsigned	Signed	1's Comp	2's Comp	Biased
Can represent positive numbers	T	T	T	T	T
Can represent negative numbers	F	T	T	T	T
Has more than one representation for 0	F	T	T	F	F
Use the same addition process as unsigned	T	F	T	T	F

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

- 2 ^ (n-1) = -2^15 = -32768

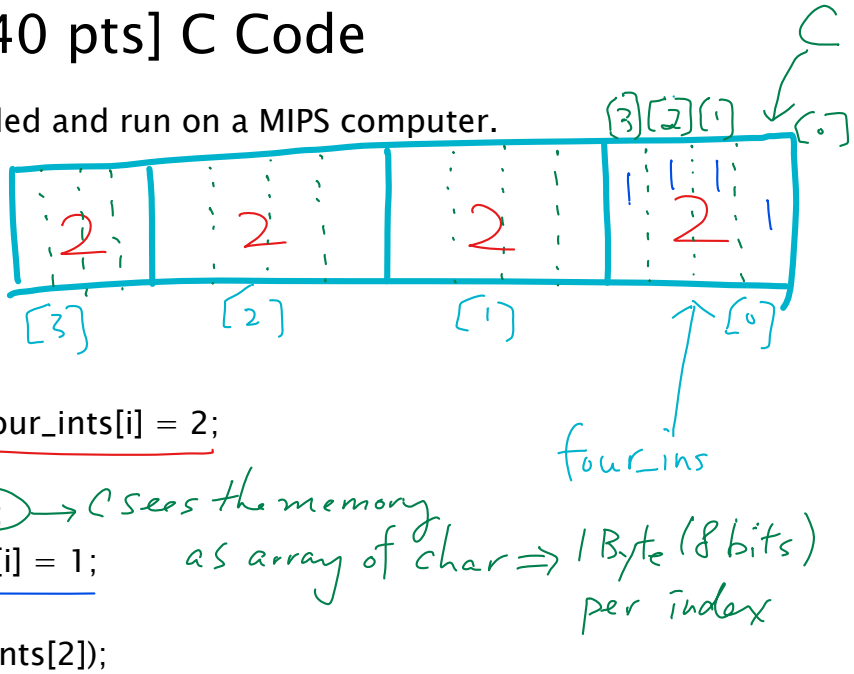
d) [5 pts] What is the value in decimal of the most positive 16-bit signed integer?

2^(n-1) - 1 = 32767

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.)

[2, 2, 2, 2]

[00 00 00 02, 00 00 00 02, 00 00 00 02, 00 00 00 02] ← content of array (before)

c is character pointer so the c[i] is in terms of bytes (not words)

c[0] = 1, c[1] = 1, c[2] = 1. c[3] = 1

[01 01 01 01, 00 00 00 02, 00 00 00 02, 00 00 00 02] ← content of array (after)

one int → 32 bits

→ 2

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)

printf("%x\n", four_ints[0]);

01010101

Note: the program is working on the same storage. four_int is seeing it as units of int (4 Bytes), while c is seeing it at units of char (1 Byte). As a result, the increment of index has different results.

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(   strlen(str)+1_____________________);

        strcpy(___result_____________,_____str_____________);

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

                if (*p >= 'a' && *p <= 'z')
                        *p += 'A' - 'a'; // -32

        }
        return result;
}
```

d) [5 pts] Consider the code below. The upcase name() function should convert the i^{th} 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 */

                *n = upcase(*n);

        }

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