

Computer Organization

Fall 2004

Test #1

Name _____ **Answer Key**

There are 6 pages - make sure you have all of them.
Answer all questions - pay attention to the # of points for each question.
Don't leave anything blank - partial credit is always possible!

Question 1 (15 pts): Complete the following tables by filling in each blank space with the appropriate value.

Decimal	8 bit 2's complement binary	Hex
1	00000001	01
-85	10101011	AB
-55	11001001	C9
70	01000110	46

Decimal	8 bit unsigned binary	Hex
187	10111011	B9
201	11001001	C9
29	00011101	1D

Decimal	IEEE Single Precision (32 bit)		
	Sign (1)	Exponent (8)	Significand (23)
134.25	0	10000110	00001100100000000000000

Question 2 (10 pts) : This question has three parts, all involve the following C statement:

width = pagewidth + (margin*2 + border*2);

Part a (5 pts): Show how this computation might look when using a processor based on a *stack-based instruction set architecture* (generic assembly is fine, with instructions like "push y", "add" and "pop tmp").

```
push margin
push border
add
push 2
mult
push pagewidth
add
pop width
```

Part b (5 pts): Show how the computation might look when using a processor based on an *accumulator based instruction set*. Generic assembly is fine here also, instructions like "add y", "load z" and "store x" are expected.

```
load margin
add border
mult 2
add pagewidth
store width
```

Question 3 (25 pts): Write a C function named `reversebits()` with the following prototype:

```
unsigned char reversebits(unsigned char x);
```

The function should return an unsigned char with binary representation corresponding to the bits of `x` reversed. For example, if the function is passed the value 55_{10} (in binary this is `00110111`) it should return the value 236_{10} (in binary this is `11101100`). The number 1_{10} when reversed becomes 128_{10} .

IMPORTANT! You cannot use arithmetic operators or functions like `*`, `/` or `pow()` – you must use logic operations such as `&` and `|` to manipulate the bits directly.

You should assume that `char` and `unsigned/char` are 8 bits.

```
unsigned char reversebits(unsigned char x) {  
  
    int i;  
    unsigned char acc;  
  
    for (i=0;i<8;i++) {  
        /* This could also be:  
           acc = (acc<<1) | (x & (0x01<<i)); */  
  
        acc = acc << 1;  
        if (x & (0x01<<i))  
            acc = acc | 0x01;  
    }  
    return(acc);  
}
```

Question 4 (20 pts): We are using a computer with the following characteristics:

- The C type `int` is represented using 32 bit two's complement representation.
- The C type `char` is signed, 8 bit two's complement.
- The C type `unsigned char` is 8 bit unsigned integer.
- All pointers are 32 bits.
- All signed integers are right shifted arithmetically.

Consider the following variable declarations found in a single C function:

```
unsigned char ux,uy;
signed char sx, sy
char *p;
int i,j;
```

Determine whether each of the expressions shown below is always true, always false, or you can't tell without knowing the actual value of the variables when the instructions are executed. I've done one for you (`ux == ux` is true for all possible values of `ux`).

Expression	always true?	always false?	can't tell?
<code>ux == ux</code>	true	false	can't tell
<code>(ux & uy) <= (ux uy)</code>	true	false	can't tell
<code>(sx & sy) <= (sx sy)</code>	true	false	can't tell
<code>(~i + 1) == -i</code>	true	false	can't tell
<code>(unsigned char)(ux+ux) >= ux</code>	true	false	can't tell
<code>((ux >> 1) << 1) == ux</code>	true	false	can't tell
<code>&ux == &sx</code>	true	false	can't tell
<code>(p+4) == p[4]</code>	true	false	can't tell
<code>sizeof(p)==18</code>	true	false	can't tell
<code>sizeof(ux)==sizeof(sx)</code>	true	false	can't tell
<code>(ux & (! ux)) == 0</code>	true	false	can't tell

Question 5 (15 pts): We want to create a (small) instruction set for a robot. The individual instructions that the robot needs are as follows:

- move forward one or two meters (2 different instructions!).
- turn to the right 45°
- turn to the left 45°
- fire phaser
- say "Drop Your Weapon"
- say "I'll be back"
- go back 1, 2, or 3 instructions (three different instructions!)
(back 1 would go to the previous instruction)

Your job is to develop a 4-bit binary *machine code* that encodes these instructions (each instruction must be 4 bits) and to then write a machine code program that instructs the robot to do the following (in the order shown):

- move 3 meters forward, turn 90° to the right and move 2 meters ahead.
- say "Drop Your Weapon" twice.
- repeat the following forever:
 - fire phaser.
 - say "I'll be back".
 - turn left 45° .

Show the machine language for each of the possible instructions, then show the machine code for the program. Use the back of this page if you need more room.

One possible machine code:

```
000x      move forward 0000 is 1 meter, 0001 is 2 meters
001x      turn 45 degrees, 0010 it to left, 0011 is right
0100      fire phaser
011x      say something 0110 is "Drop your weapon"
           0111 is "I'll be back"
10xy      go back xy instructions:
           1001 - go back one instructions
           1010 - go back two instructions
           1011 - go back three instructions
```

machine code program

```
0001 move 2 meters
0000 move 1 meters
0011 turn 45 deg. right
0011 turn some more
0001 move 2 meters
0110 say "Drop your weapon"
0011 say "Drop your weapon"

loop:
0100 fire phaser
0111 say "I'll be back"
0010 turn left
1011 jump back to loop
```

Question 6 (10 pts): Show the output generated by the following C program:

```
#include <stdio.h>

void showstring(char *s) {
    int size;

    size = sizeof(s);
    printf("size is %d\n",size);

    printf("The string is %s\n",s);
}

/* Some global variables */

char s1[] = "Hello There";
char s2[] = "Red and Blue";
char s3[] = "CompOrg is my favorite course";

char *stngs[] = { s1, s2, s3 };

int main() {
    char *p;

    printf("size of stngs[1] is %d\n",sizeof(stngs[1]));

    printf("size of stngs is %d\n",sizeof(stngs));

    printf("Difference is %d\n", *stngs - stngs[0]);

    p = "Hello Fred";
    showstring(p);

    p = 0;
    showstring(p);
    return(0);
}
```

```
size of stngs[1] is 4
size of stngs is 12
Difference is 0
size is 4
The string is Hello Fred
size is 4
The string is (null)
```