

# Computer Organization

Fall 2002

Test #1

Name \_\_\_\_\_ **Answer Key**

There are 5 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
-55	11001001	C9
102	01100110	66
-128	10000000	80

Decimal	8 bit unsigned binary	Hex
250	11111010	FA
33	00100001	21
128	10000000	80

Decimal	IEEE Single Precision (32 bit)		
	Sign (1)	Exponent (8)	Significand (23)
11.75	0	10000010	01111000000000000000000

**Question 2 (20 pts):** Write IA32 (x86) assembly code that will accomplish the computation described by the C statement:

**`x = a[0]+2*a[1]+a[2];`**

Your assembly code should assume the following:

- `x` is of type `int`, and `a` is an array of `int`
- The address of the array `a` is `123410` ( the address of array `a` is the same as the address of `a[0]` ).
- At the end of your assembly code, the value of `x` should be placed in register `%eax`

Your code must be based on the assembly syntax used by `gcc/gas` (the gnu assembly syntax). Comment your code!

```
# a is 1234
movl    $1234,%edx    # edx now holds a
movl    4(%edx),%eax  # eax is now a[1]
sall    $1,%eax      # eax is now a[1]*2
addl    0(%edx),%eax  # eax is a[0]+2*a[1]
addl    8(%edx),%eax  # eax is now a[0]+2*a[1]+a[2]
```

**There are lots of other possible solutions!**

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

```
int compare(char x, char y);
```

The function should return `-1`, `0` or `1` according to the following:

$$\text{compare}(x,y) = \begin{cases} -1 & \text{if } x < y \\ 0 & \text{if } x = y \\ 1 & \text{if } x > y \end{cases}$$

You must assume that the C data type `char` is signed (2's complement).

**IMPORTANT!** Your function cannot use comparison operators like `<` or `>` to simply compare `x` and `y`. You are not allowed to use arithmetic to determine the return value (you can't just do a subtraction to determine the return value). Your job is to **write a C function that looks at the individual bits of 2 signed 8 bit numbers to determine the return value**. All bitwise logic operators and shift operators are allowed. You may use comparison operators and arithmetic operators for loop control, etc.

```
// returns -1 if x<y, 0 if x==y, 1 if x>y
int compare(char x, char y) {
    int i=0;

    if ((x&0x80) == (y&0x80)) {
        // same sign now look at the remaining 7 bits
        x = x << 1;
        y = y << 1;

        while ( (i<7) && ((x&0x80) == (y&0x80)) ) {
            x = x << 1;
            y = b << 1;
            i++;
        }
        // if we looked at all the bits, they are identical
        if (i==7) return(0);

        // found a position with differing bits.
        // the one with a 1 must be larger
        if (x&0x80) {
            return(1);
        } else {
            return(-1);
        }
    } else {
        // different signs, if x is neg it must be smaller
        if (x&0x80) {
            return(-1);
        } else {
            return(1);
        }
    }
}
```

**Question 4 ( 20 pts):** The C function `compute()` is defined here.

The assembly language generated by `gcc -S` is shown below. The code is partially commented; finish commenting by indicating on each line what the assembly instruction does as related to the C program. Don't say things like "multiples `%eax` by `%edx`", say things like "`%eax = 3*y`".

**Note:** `%ebp+8` is the address of `x` and `%ebp+12` is the address of `y`.

```
int compute(x,y) {
    if (x>y) {
        return(x+y*3);
    } else {
        return(x-y*3);
    }
}
```

```
compute:
    pushl   %ebp                # setup
    movl   %esp, %ebp          # setup

    movl   8(%ebp), %eax        # eax = x
    cmpl   12(%ebp), %eax       # Set CC x-y
    jle    .L3                 # jump if x<y
    movl   12(%ebp), %eax       # eax = y
    movl   %eax, %edx           # edx = y
    sall   $1, %edx             # edx = 2*y
    addl   %eax, %edx           # edx = 3*y
    movl   %edx, %eax           # eax = 3*y
    addl   8(%ebp), %eax        # eax = x + 3*y
    movl   %eax, %eax           # gcc funny stuff
    jmp    .L2

.L3:
    movl   12(%ebp), %edx       # edx = y
    movl   %edx, %eax           # eax = y
    sall   $1, %eax             # eax = 2*y
    addl   %edx, %eax           # eax = 3*y
    movl   8(%ebp), %edx        # edx = x
    subl   %eax, %edx           # edx = x - 3*y
    movl   %edx, %eax           # eax = x - 3*y
    movl   %eax, %eax           # gcc funny stuff

.L2:
    popl   %ebp                # prepare for return
    ret                          # return
```

**Question 5 ( 20 pts):** The assembly language version of the function `compute()` shown in Question 4 can be made more efficient. Rewrite the code to eliminate as many instructions as possible. You must consider **both** the overall size of the function (make it as small as possible) and the number of instructions that can be executed in a single call to this function (minimize everything!). Comment your code!

```
compute:
    pushl    %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax    # eax is y
    movl    %eax, %edx       # edx is y
    sall    $1, %edx         # edx is 2*y
    addl    %eax, %edx       # edx is 3*y

    movl    8(%ebp), %eax     # ax is x
    cmpl    12(%ebp), %eax   # compare x to y
    jle     .L6              # jump if x < y
    addl    %edx, %eax       # eax = x + 3*y
    jmp     .L5
    .p2align 2

.L6:
    subl    %edx, %eax       # eax = x - 2*y

.L5:
    popl    %ebp
    ret
```

**There are lots of other possible solutions!**