ECE391: Computer Systems Engineering **FALL 2018**
Problem Set 1 **Due: Tuesday 9/11 at 2:00 pm (ECEB 3rd floor Dropbox #54)**

Work in groups of at least 3 and hand in ONE copy of your group's work. You must put each and every group member's **name** and **netID** on the problem set. **Do not put your name on more than one problem set or you will receive a 0. Or please type up your solutions (black text on white background) or you will receive a 0.**

Problem 1 GDB Warm Up

Please write the command(s) you should use to achieve the following tasks in GDB.

1. Show the value of variable "test" in hex format.
2. Show the top four bytes on your stack word-by-word, e.g. it should look something like this "0x0102 0x0304", NOT "0x01020304".
3. Check all register values.
4. Set a breakpoint at "ece.c" line 391.
5. Connect to the test_(no)debug vm in the lab setup.

Problem 2 Mapping C to Assembly Convert the following C function to Assembly. Assume that the *merge* function is implemented for you and performs as expected. You only need to convert the *mergeSort* function to assembly. You may also assume that the input array will never be null when $l < r$.

```
/* func: merge
 * merges the sub array arr[l, ..., m] and arr [m+1, ... ,r]
 * returns: none */
merge(int* arr, int l, int m, int r) {
    // Merges Sub Arrays
}

/* func: mergeSort
 * Applies Merge Sort algorithm to unsorted integer array
 * returns: none */
void mergeSort(int* arr, int l, int r) {
    if(l < r) {
        /* Find the middle of the array or sub-array */
        int m = l+(r-l)/2;

        /* Recurse on left and right sub-arrays */
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        /* Merge sub arrays into sorted array */
        merge(arr, l, m, r);
    }
}
```

Problem 3 Mapping Assembly to C

Write a C function equivalent to the following x86 assembly code. Please remember to use black text (e.g. Courier New is a code-friendly font) on a white background.

```
.GLOBAL calculate
calculate:
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx
    pushl %esi
    pushl %edi
    movl 8(%ebp), %ecx
    movl 12(%ebp), %ebx
    movl 16(%ebp), %esi

    cmpl $2, %ecx
    ja default
    jmp *jumptable(,%ecx,4)
op1:
    movl %ebx, %eax
    imull %esi, %eax
    jmp done
op2:
    cmpl $0, %esi
    je default
    movl $0, %edx
    movl %ebx, %eax
    idivl %esi
    jmp done
op3:
    movl $0, %eax
lp:
    cmpl %esi, %ebx
    jle  done
    addl %ebx, %eax
    subl $-1, %ebx
    jmp  lp
default:
    movl $-1, %eax
done:
    popl %edi
    popl %esi
    popl %ebx
    leave
    ret
jumptable:
    .long op1, op2, op3
```