Winter '18 CIS 314 Assignment 4 – 100/100 points – Due Monday, 2/12, 11:59 PM

Please submit individual source files for coding exercises (see naming conventions below) and a single solution document for non-coding exercises (.txt or .pdf only), when appropriate. Your code and answers need to be documented to the point that the graders can understand your thought process. Full credit will not be awarded if sufficient work is not shown.

1. [30] Consider the following IA32 code:

```
loop:
     pushl %ebp
     movl  %esp, %ebp
     pushl %esi
     pushl %ebx
     movl  8(%ebp), %esi
     movl  12(%ebp), %ecx
     movl  $2, %edx
     movl  $-1, %eax
.L2:
     movl  %esi, %ebx
     andl  %edx, %ebx
     xorl  %ebx, %eax
     sall  %cl, %edx
     cmpl  $1, %edx
     jg    .L2
     popl  %ebx
     popl  %esi
     popl  %ebp
     ret
```

The code above was generated by compiling C code that has the following overall form:

```
int loop(int x, int y) {
    int result = ? ;
    for (int mask = ? ; mask ? ? ; mask = ? ) {
        result ^= ? ;
    }
    return result;
}
```

Copy the above IA32 code into a C file as a comment. Annotate each line of the IA32 code in terms of *x*, *y*, *result*, and *mask*, and the register usage-conventions outlined in B&O'H section 3.7.3. Then implement the above C function by filling in the blanks so that it's functionally equivalent to the IA32 code. Hint: try compiling your C code to IA32 using gcc with the -S, -m32, and -O1 flags as you go (you'll need to additionally compile with -fno-omit-frame-pointer and -fno-asynchronous-unwind-tables flags on Ubuntu to match the IA32 code above).

Here are some test runs:

```
loop(1, 5): -1
loop(2, 4): -3
loop(3, 3): -3
loop(4, 2): -1
loop(5, 1): -5
```

Also write a main() function to test your loop function. Name your source file 4-1.c.

2. [40] The following C code transposes the elements of an N × N array:

```
#define N 10
typedef int array_t[N][N];

void transpose(array_t a) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < i; ++j) {
            int t = a[i][j];
            a[i][j] = a[j][i];
            a[j][i] = t;
        }
    }
}
```

When compiled with optimization level -O2, gcc generates the following code for the **inner loop** of the function:

```
.L3:
      movl  (%edx), %esi
      movl  (%eax), %ecx
      addl  $4, %eax
      addl  $40, %edx
      movl  %esi, -4(%eax)
      movl  %ecx, -40(%edx)
      cmpl  %ebx, %eax
      jne   .L3
```

Copy the above IA32 code into a C file as a comment. Annotate each line of the IA32 code in terms of *N*, *a*, *i*, *j*, and *t*. Then write a C version of transpose that makes use of the optimizations in the IA32 code. Hint: use pointer arithmetic to avoid the A[i][j] and A[j][i] lookups (similar to the example in Figure 3.28 in the textbook).

For example, the input (assuming *N* = 4):

```
{{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16}}
```

should be transposed as:

```
{{1, 5, 9, 13}, {2, 6, 10, 14}, {3, 7, 11, 15}, {4, 8, 12, 16}}
```

Also write a main() function to test your procedure. Name your source file 4-2.c.

3. [30] Consider the following C code:

```
int sum(int from, int to) {
    int result = 0;
    do {
        result += from;
        ++from;
    } while (from <= to);
    return result;
}
```

Implement the *do-while* loop above in IA32. Use the following as a framework:

```
int sum(int from, int to) {
      int result = 0;
      // Ensure that argument *from* is in %ecx,
      // argument *to* is in %edx, *result* is in %eax - do not modify.
      __asm__ ("movl %0, %%ecx # from in ecx;" :: "r" ( from ));
      __asm__ ("movl %0, %%edx # to in edx;" :: "r" ( to ));
      __asm__ ("movl %0, %%eax # result in eax;" :: "r" ( result ));

      // Your IA32 code goes below - comment each instruction...
      __asm__(
           "movl %edx, %eax # For example, this sets result = to;"
      );

      // Ensure that *result* is in %eax for return - do not modify.
      __asm__ ("movl %%eax, %0 #result in eax;" : "=r" ( result ));
      return result;
}
```

Add a comment describing the purpose of each of your IA32 instructions. Your IA32 code must follow the register usage conventions outlined in B&O'H section 3.7.3. Hint: use your answers from parts 1 and 2 above as a guide. FWIW, I was able to get this working with 4 instructions and 1 label.

Here are some test runs:

```
sum(1, 6): 21
sum(3, 5): 12
```

Also write a main() function to test your sum function. Name your source file 4-3.c.

Zip the source files and solution document (if applicable), name the .zip file <Your Full Name>Assignment4.zip (e.g., EricWillsAssignment4.zip), and upload the .zip file to Canvas (see Assignments section for submission link).