CPSC 313

06W Term 2

Problem Set #1 Solution

1. I edited the output slightly to make it shorter.

```
        .bss
_counter:
        .space 4                        # allocate space for global counter
        .text
_swap:
        pushl   %ebp                    # save ebp
        movl    %esp, %ebp              # set up ebp for this function
        subl    $8, %esp                # allocate space for saved regs
        movl    %ebx, (%esp)            # save ebx
        movl    8(%ebp), %edx           # load a into edx
        movl    16(%ebp), %ebx          # load j into ebx
        movl    %esi, 4(%esp)           # save esi
        movl    12(%ebp), %ecx          # load i into ecx
        movl    (%edx,%ebx,4), %eax     # load a[i] into t (eax)
        movl    (%edx,%ecx,4), %esi     # load a[j] into a temp (esi)
        movl    %eax, (%edx,%ecx,4)     # store t into a[j]
        movl    %esi, (%edx,%ebx,4)     # store temp into a[i]
        movl    _counter, %eax          # load counter to increment it
        movl    (%esp), %ebx            # restore saved ebx
        movl    4(%esp), %esi           # restore saved esi
        incl    %eax                    # increment counter
        movl    %eax, _counter          # write it back
        movl    %ebp, %esp              # set up for popping ebp
        popl    %ebp                    # restore saved ebp
        ret                             # return
```

2. I'm not going to include the output, because if you can run objdump you can do it yourself.

   The differences in the swap procedure are:

   (a) The instructions now have actual addresses.

   (b) The address of the global variable counter is now known (it is not shown as 0).

3. (a) `movl` copies data from one location to another and will read or write memory if either operand addresses memory. `leal` computes the address of a memory operand and places that address in a register, this never involves referencing memory.

   (b)
   - `leal -12(%ebp), %ebx`

     Adds -12 and the contents of the `%ebp` register, placing the result in the `%ebx` register.
   - `movl -12(%ebp), %ebx`

     Adds -12 and the contents of the `%ebp` register. Uses the result as an address, copying the contents of memory at that address into the `%ebx` register.

(c) The legal one is

```
movl    %ebp, (%ebx)
```

It is legal because only one operand references memory. It copies the contents of the `%ebp` register into memory at the address given by the contents of the `%ebx` register.

```
leal    %ebp, (%ebx)
```

is illegal for two reasons: because `%ebp` does not have an address and because `leal` must place its result in a register.

(d) The legal one is

```
movl    %ebp, %ebx
```

It is legal because no more than one operand references memory (0 in this case). It copies the contents of the `%ebp` register into the `%ebx` register.

```
leal    %ebp, %ebx
```

is illegal because `%ebp` does not have an address.

4.

```
int decode2(int x, int y, int z)
{
  int t1 = y - z;
  int t2 = x * t1;
  int t3 = (t1 << 31) >> 31;
  int t4 = t3 ^ t2;
  return t4;
}
```