ECS 50 Midterm Review Guide

1. Know what all of the instructions on the Intel Cheat Sheet do and how to use them.
2. Understand how the location of a variable's declaration affects where it will be stored. Ie if a variable is local, global, or static where will space be made for it.
    1. Space for globals and static variables is made in the data section.
    2. Space for locals is made on the stack.
3. Understand how typing in C affects the assembly instructions that are generated by the compiler.
    1. Translate the following code to assembly. Assume that we want to store x in eax and c in ecx.

        ```
        int* x = 100;
        char* c = 70;
        x += 10;
        c += 10;
        ```

    2. movl $100, %eax
    3. movl $70, %ecx
    4. addl $10, (%eax)
    5. addb $10, (%ecx)
4. What are the gcc C calling conventions? What happens if a function breaks the calling conventions?
    1. EAX, ECX, and EDX will not have live values when a function is called. All other registers will contain live values
    2. Arguments are passed on the stack and are pushed from right to left
    3. The return value, if any, will be placed in EAX
5. Understand and be able to use the advanced indexing mode.
    1. Assume that eax = 100 and ecx = 5. Which addresses in memory are accessed by the following instructions?
        1. movl (%eax), %ebx
            1. bytes 100 - 103
        2. movb (%ecx,%eax), %bl
            1. byte 105
        3. movl %eax, %ebx
            1. No memory access
        4. movw (%ecx, %eax, 4), %bx
            1. bytes 405 - 406
6. What is considered the stack?
    1. Wherever esp points and all addresses higher than that
7. What is considered the current stack frame?
    1. Everything in between ebp and esp
8. Stack frames are “chained”. What does this mean? How is it achieved?
    1. This means that you can access previous stack frames from the current stack frame. This is achieved through the prologue with these two lines
        1. push %ebp
        2. movl %esp, %ebp
    2. How can you use this to say access the third argument of the function 4 calls prior to you.
        1. movl %ebp, %eax #the current stack pointer
        2. .rept 4
        3. movl (%eax), %eax #chase the stack frame pointers
        4. .endr

5. movl $4*wordsize(%eax), %ecx #access the third argument

9. Write assembly instructions that emulate the effect of a push instruction.
   1. subl $4, %esp
   2. movl src, (%esp)
10. Write assembly instructions that emulate the effect of a pop instruction.
   1. movl (%esp), dest
   2. addl $4, %esp
11. Write an assembly function that is callable from C that emulates the following C code.

```
short max(short* nums, int len){
  int index;
  short cur_max = nums[0];
  for( index = 1; index < len; index++)
    if( nums[i] > cur_max)
      cur_max = nums[i];
  return cur_max;
}
```

12. Write an assembly function that is callable from C that emulates the following C code.

```
short rec_max(short* nums, int len){
  short rest_max;
  if(len == 1)
    return nums[0];
  else{
    rest_max = rec_max(nums + 1, len -1);
    return nums[0] > rest_max ? nums[0] : rest_max;
  }
}
```