# LINKHLL Homework

## Overview

- Specification on the WEB
- Write a C subroutine
- Passed 4 unsigned words on the stack

  *linkhll (v1, v2, v3, v4);*
- Find the two largest words
- Calculate their product
- Return the 32 bit product in dx:ax
- Get grading system from linkhll locker

  linkhll.m - model for your code

  linkdrvr.obj - test & grading driver
- testlink .... will test your code
- gradlink .... will grade your code

## Design Ideas

LINKHLL measures the number of instructions executed to solve the problem. Optimizing execution time requires a different way of thinking than is traditional in high-level languages.

1. Unrolling loops

   When coding in an HLL, the standard way to write code is to reduce the number of instructions written by putting repeating code in a loop. However, loops have execution overhead. You must execute instructions to manage the number of times the code goes through the loop and to update pointers.

   When coded in C, this loop executed 31 instructions. Most to manage loop control.

   ```
   FOR I=0 TO 3 DO  LIST2[I] = LIST1[I]
   ```

   When coded in C, the unrolled loop executed 8 instructions

   ```
   LIST2[0]=LIST1[0]
   LIST2[1]=LIST1[1]
   LIST2[2]=LIST1[2]
   LIST2[3]=LIST1[3]
   ```

2. Solving the right problem

   Are you fully sorting the data? That is not the problem. The problem is to find the two largest values. Just doing that may require less code than a full sort.

3. Exchanging two values

   How do you exchange the contents of two items?
   It can be done in one instruction … which is in the Class Notes.

4. Moving data around

   Are you loading the 4 values from the stack into 4 registers? Is that necessary?

5. Avoid spaghetti code

   Do not jump forward and then back. Change that to inline code.

```
        cmp   ax,bx
        jb    swap
nosw:


swap:   swap ax and bx
        jmp   nosw
```

```
        cmp   ax,bx
        jae    nosw
        swap ax and bx
nosw:
```

6. Use inline code instead of subroutines

   Subroutines reduce the amount of code ***written***. However, they have overhead in code ***executed*** for linkage and to pass parameters. For example, using a subroutine to swap two values will execute more instructions than duplicating the swap code when needed.

7. C only requires bp, si, di, ss, ds be restored. This means you can use ax, bx, cx, dx without saving and restoring them. This will tend to create more efficient and clearer code than creating memory variables.