CSC 236 Programming Assignment

LINKHLL

Summary

The LINKHLL assignment is worth 4% of your grade. For this assignment, you will be submitting a file named **linkhll.ans**. This file is created by the grading system when you grade your program. The grade is based on correct answers, documentation and the number of instructions executed during testing.

If you have questions about the assignment, feel free to use our EdStem board, but be sure not to post any code from your solution in a public post (even code that's not working). You can also stop by during office hours to get help from anyone on the teaching staff. Remember, if you need help, you need to make sure your code is documented, including block comments above sections of your code and line comments saying what you are trying to do with each instruction. We'll also expect you to have a design for the program that you're trying to follow.

Program description

The goal of LINKHLL is to give you experience writing assembly language that might be used as part of a larger program in a high-level language (C in this case). Your code will be called as a subroutine from a C program, so you must use the correct C/C++ calling conventions.

Specification for the LINKHLL subroutine

The **linkhll** subroutine will be called from a C program. It will be passed four unsigned words as parameters. As with other C functions, these parameters will be passed on the stack (by value rather than by address). The job of the subroutine is to find the two largest unsigned values and multiply them, creating a 32-bit unsigned product. The resulting, 32-bit product will be returned in the dx:ax register pair.

The source file must be named: **`linkhll.asm`**

In your assembler code, the routine must be named: **`_linkhll`**

Your subroutine will be called like the following, where v1 .. v4 are unsigned, 16-bit values:

```
linkhll (v1, v2, v3, v4);
```

The return value from the **linkhll** subroutine will be the dx:ax pair which will contain the 32 bit unsigned product of the two largest values of v1, v2, v3, v4. The dx:ax pair must be identical to the values created by executing the mul instruction. All registers required to be saved by C calling conventions must be restored to their original value when **linkhll** returns.

Step 1: Planning your Solution

Keep the following in mind as you plan for implementing your function:

- Follow the C linkage conventions.
- Access the four parameter values passed on the stack.
- Select the two largest unsigned values.
- Multiply them together creating a double word result in the dx:ax pair Return to the calling program with the result in the dx:ax pair.

Step 2: Creating your Solution

Your source code must be named **linkhll.asm**, and all your source should be in this one assembly language file.

To get started, retrieve the testing and grading files. As usual, these are packed together in one self-extracting file named **unpack.exe**. You can download it from the LINKHLL block in the course page in Moodle.

Put a copy of the unpack.exe file in the LINKHLL subdirectory of your shared DOSBox folder. Then, start DOSBox, go through your usual steps to start a DOSBox session (see previous assignments if you need a reminder) and then change directory to LINKHLL and execute **unpack.exe**. This will create the testing and grading files you need.

The archive will create a starter file for your subroutine implementation. Copy the file named **linkhll.m** to the name **linkhll.asm**. Then add your code to the **linkhll.asm** file.

Unpacking the archive will also create a C driver program for your subroutine. It is named **linkdrvr.obj**. This driver program will be used for testing and grading your subroutine. Assemble and link your linkhll subroutine with the driver program **linkdrvr.obj** using these commands:

```
ml /c /Zi /Fl linkhll.asm

link /CO linkdrvr.obj linkhll.obj
```

Step 3: Testing and Debugging

You will use the tests built into the driver program to test your **linkhll** subroutine. To run a test, type the following command. The output from this test will go to a file named testout:

```
testlink
```

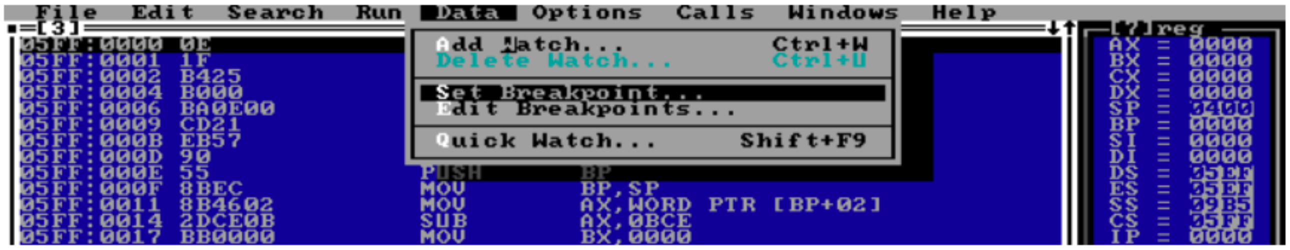
You can watch the execution of your subroutine using the CodeView Debugger with the driver program. To run under CodeView under DOSBox, first enter the following command. This copies the CodeView configuration file to your current directory:

```
cvset
```

Then, enter the following to run the driver program and send output to the same test output file:

```
cv linkdrvr > testout
```

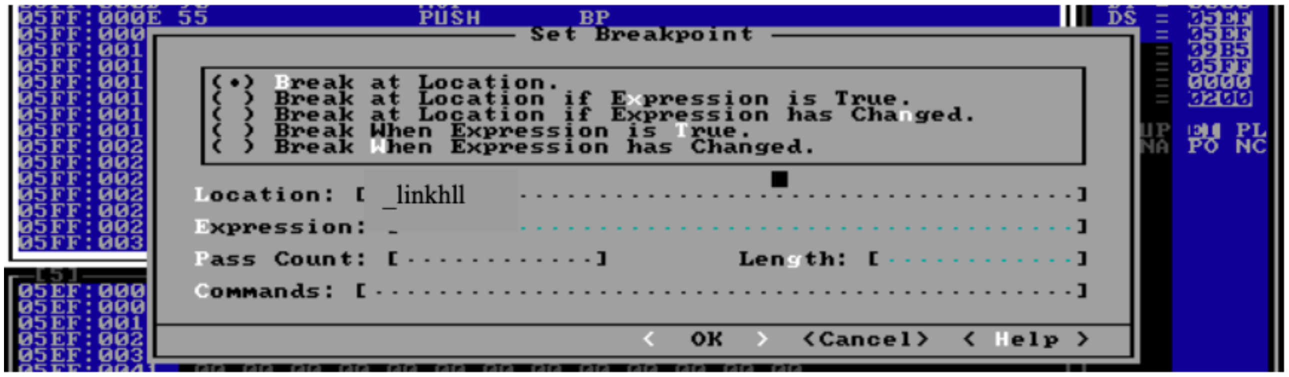
After CV loads, you’ll want to set a breakpoint to see what your code is doing. On the top row of the CV interface, select "Data", then select "Set Breakpoint"



```
05FF:0000 0E
05FF:0001 1F
05FF:0002 B425
05FF:0004 B000
05FF:0006 BA0E00
05FF:0009 CD21
05FF:000B EB57
05FF:000D 90
05FF:000E 55          PUSH   BP
05FF:000F 8BEC        MOV    BP,SP
05FF:0011 8B4602      MOV    AX,WORD PTR [BP+02]
05FF:0014 2DCE0B      SUB    AX,0BCE
05FF:0017 BB0000      MOV    BX,0000

AX = 0000
BX = 0000
CX = 0000
DX = 0000
SP = 0400
BP = 0000
SI = 0000
DI = 0000
DS = 05EF
ES = 05EF
SS = 09B5
CS = 05FF
IP = 0000
```

Under "Location:[...]" type _linkhll so it looks like "Location:[_linkhll]". Then click <OK> down at the bottom of the window.



```
(•) Break at Location.
( ) Break at Location if Expression is True.
( ) Break at Location if Expression has Changed.
( ) Break When Expression is True.
( ) Break When Expression has Changed.

Location: [ _linkhll                                    ]
Expression: [                                           ]
Pass Count: [            ]          Length: [           ]
Commands: [                                             ]
```

Press F5 to start execution. CV should run until it reaches the start of your **_linkhll** routine.

In the Memory Window you will see data at xxxx:yyyy Make sure that xxxx matches the value in the DS register. You should be able to just put the cursor on the xxxx field and set it to match the DS register. This will let you see what’s in your data segment via this window.

Use F8 to step through your code and see what’s going on.

Step 4: Grading

Run the grading system to determine how many points your program has earned. In DOSBox, go to the directory where you unpacked the archive for this project and enter the following command:

```
gradlink
```

You may re-run the grading system. However, grading is not a replacement for your testing, so the grading system will count the number of times it was executed. The results file will contain the chronology of both your testing and grading.

Once your program is working, you can make additional grading runs to improve efficiency or documentation. To mark these as grading runs that were only made to improve efficiency or documentation, execute this command after your program has earned the 60 points for getting the correct answers:

```
testlink mark
```

All subsequent grading runs will be marked as only being done to improve efficiency or documentation.

Your grade for the assignment will be based on:

- 60 points for getting the correct answers to the grading program test cases.
- 20 points for the number of instructions executed in the grading run that had the correct answers.
- 20 points for documentation of a program that functions correctly.

Efficiency and documentation are a concern only after the code works correctly. So, efficiency and documentation grading is only done for code that passes all the functional tests.

Step 5: Submitting your Solution

Electronically submit the **linkhll.ans** file created by the grading system.

Submit this to the LINKHLL assignment on Moodle.

Before you submit, you will want to make sure:

- Your **linkhll.ans** file contains two concatenated files. First the results file and then your source file **linkhll.asm**.
- Your **linkhll.ans** file contains the line: ++ Grade ++ nnn = Total grade generated by the Grading System.

Incorrect electronic submissions will result in your program not being graded.
Be sure not to edit or modify any of the grading system files. This is especially true for the files named results and **linkhll.ans**, which contain the grading results. Any modification of grading system files will look like cheating.