Basic DEBUG Programming and TASM
Due: January 27, 2020 on MyCourses by 23:55

In this lab we want to practice INTEL assembly programming. We will do this in two ways. The first way will be to use the DEBUG program from last class to write some basic text-based I/O INTEL code snips. The second way is to write our first true INTEL assembly program which we will compile and run. We wanted to compile and assemble a TASM program that inputs an integer and prints "Hello World" that many times on the screen.

PART 1: DEBUG CODE SNIPS

For part 1, you are supplied with two "debug programs" - one prints the character 'A' 3 times on the screen, the other prints 'Hello World!' on the screen.

The goal of the first code snip is to learn how to print a character and observe how the loop or jump instructions change the IP register, and the second one shows that all memory can be used to store code as well as data.

First program:

- Open Debug (open a DOS terminal and type 'debug')
- Enter assemble mode (type 'a')
- Enter this code to assemble:

```
mov cx, 3       ( set counter to 3 )
push cx         ( push counter to save it )
mov ah, 6       ( int 21h function 6 - Direct console input/output )
mov dl, 41      ( ASCII code for the 'A' character )
int 21          ( call the interrupt )
pop cx          ( restore counter )
dec cx          ( decrement the counter by 1 )
jnz 0103        ( jump to location 0103h if zero flag is not set, here I assume that the assembly started
                  at offset 0100h )
```

- Exit assemble mode (press enter on an empty line)
- Execute the program step by step (typing 'p' and pressing enter for each step)

NOTE: You will see the 'A' character printed right after you step over the 'int 21' instruction

Second program:

- Restart the Debug program and enter the following code:

```
jmp 010F              (jump to location 010Fh, skipping the string we are about to define in memory.
                          We assume the assembly starts at offset 0100h)
db 'Hello World!$'    ( define $ terminated string 'Hello World!', which is 13 characters total )
mov ah, 9             ( int 21h function 9 - Write a $ terminated string to standard output )
mov dx, 102           ( address of our string, note that we assume here that the DS data segment
                       register points to the same place our CS code segment points to, as the string
                       address is required to be in DS:DX. This is true in DEBUG but usually not true
                       for a normal program, in which case you would have to manually set the DS
                       segment to the value of the CS segment )
int 21                ( call the interrupt )
```

In this piece of code, after you step over the 'int 21' instruction the full string 'Hello World!' will be printed to the screen.

PART 2: TASM PROGRAMMING

You will need to do steps 1 and 3. Step 2 and 4 are informational. Step 2 you should read.

Step 1 - First we need to install TASM:

-The TASM ZIP file was included with this lab
-Extract the zip file somewhere in your home directory
-Open dosbox, mount your home directory, find your extracted TASM files and run "INSTALL.EXE".
-When prompted for the source drive, type "C" (or whatever you mounted as your home directory), it should then correctly find the extracted TASM folder
-When prompted for the install directory, type whatever path you want, keeping in mind that the "C" (or whatever you mounted as your home directory) drive is mapped to your home directory.
-When installing, don't worry if there is a warning about a windows directory or path, it won't affect the actual installed files. The INSTALL may take time.
-If you get an actual error at any stage and you are on the Trottier machines, it's likely that you have run out of disk space (quota exceeded). Free some space, re-download TASM and try again, or contact help@cs.mcgill.ca and tell them what you were trying to do.
-The installed files, tasm and tlink, will be in the BIN directory.

Step 2 - Using TASM to compile and link the program:

-You start with an assembly source file (*.ASM), you then need to compile it into an object (*.OBJ) file and then link it to get an .EXE file. In DOSBOX the .EXE or .COM files are like the Linux a.out file.
-Compiler is "TASM.EXE", linker is "TLINK.EXE".

Compiling and linking .ASM files with TASM:

Run these commands:
<TASM directory>\bin\tasm <your program name>.asm
<TASM directory>\bin\tlink <your program name>.obj

To make things simpler, we can create a .BAT file similar to this (DOSBOX Bash file) (directories may vary):
C:\DOS\TASM\bin\tasm %1.asm
C:\DOS\TASM\bin\tlink %1.obj

and then use the .BAT script with the program name as argument.

Step 3 – Run the following program:

- Using a text editor, input the given program and save with extension .asm in the BIN directory
- Then CD to BIN and use TASM to compile the program. Write: `tasm program.asm`
- If there were no errors, then link the program. Write: `tlink program.obj`
- If there were no errors, then there should be a program called program.exe in your directory
- Run that program by typing its name: `program` (with or without the file extension)
- Note: name your program any name you like.

Step 4 - Optional, but this is how you would debug the .EXE file:

Using "Debug":
-Open Debug by typing "debug" in DOS prompt
-To debug a specific program you would type "debug <program name>.exe"

You may also use the visual debugger that comes with TASM located in
<TASM directory>\bin\TD.EXE

The program to input:

```
.model small
.stack 100h
.data
        sPrompt         db "Enter a number: $"
        sHello  db 10,13,"Hello World!$"
.code
start:
        ; initialize data segment
        mov ax, @data
        mov ds, ax

        ; print prompt text
        mov ah, 9
        mov dx, OFFSET sPrompt
        int 21h

        ; input number (will be stored as an ASCII character in AL)
        mov ah, 1
        int 21h

        ; set counter register to input number
        ; (subtract '0' from input ASCII character to get a number)
        sub al, '0'
        xor cx, cx
        mov cl, al

        ; label that will be used for our loop

printMore:

        ; save our counter
        push cx

        ; print our string
        mov ah, 9
        mov dx, OFFSET sHello
        int 21h

        ; restore our counter
        pop cx

        ; decrement counter and jump back if it's not 0
        dec cx
        jnz short printMore

        ; terminate program
        mov ax, 4c00h
        int 21h
END start
```

Notes on the code:

- Pushing and popping the counter (CX) is not necessary in the above code because int 21h preserves the CX register, but often interrupts modify registers so it is good to have a habit of saving your registers before calling interrupts.
- The above code will print 65536 times the hello message if you input "0" because it decrements the counter before checking for zero (thus 0 will become -1, which also represents the unsigned value 65535 in a 16-bit register), and also because the first check is done after an initial hello message was already printed (we first print hello and then check for the counter status). It may be good for students to know that for many scenarios you need to set up your loops right in terms of handling these kind of boundary cases, and possibly ask them for some alternative code that handles the 0 input.

WHAT TO HAND IN

For PART 1 – take screen shots or a photo for each code snip. There should be two images.
For PART 2 – submit the program.asm file. The TA will compile and run the program.

HOW IT WILL BE GRADED

This lab is worth 20 points.

- Part 1 Snip 1	– 5 points
- Part 1 Snip 2	– 5 points
- Part 2	– 10 points