TI-83+ Z80 ASM for the Absolute Beginner

LESSON FOUR:

- *Your First ASM Programs*

YOUR FIRST ASM PROGRAMS

Now that you have a basic understanding on the calculator processor, it's time to learn how to code! This lesson will cover several ASM programs, as well as provide exercises for you to practice with.

However, we're not going to code a “Hello World” program. To proceed in an orderly fashion, I feel that there are other things that should be covered first. After all, ASM instructions are vastly different from C++, Basic, and Java instructions.

So the question now is, what do you need to write a Ti-83+ ASM program? First, you need a text editor. I'm not going to list a whole bunch of different ones, so just use the one you're comfortable with. Then, you need a copy of ti83plus.inc, which I have included with these lessons.

Finally, you need an Assembler. This is what translates your code into the language that your calculator can understand. I can't list very many, because I haven't been programming in ASM for a long time, and I've had issues with many assemblers. Some that I've seen people use are Zilog Developer Studio 3.16 and TASM. The assembler I use, and HIGHLY recommend, is Spencer's Assembler, also called **spasm**. I have included this with the lessons. It is a Spencer's Assembler is the one I will be teaching you how to use, as it's very efficient and runs on Mac, Linux and Windows. I have provided files for Windows and Mac, and CLI instructions for Linux.

You will also need an emulator. An emulator imitates something on a computer, so a Ti-83+ emulator will allow you to run a Ti-83+ on your computer. This way, you can test your programs without putting them on your calculator. You can damage your calculator if you run an

ASM program that has errors on it, so make sure you always test your programs on an emulator before putting them on your calculator.

Which emulator should you use? WabbitEmu. Pretty much EVERYBODY who programs in Ti-83+ ASM uses WabbitEmu. I have provided applications for Windows and Mac, and CLI instructions for Linux.

You need a ROM to be able to use WabbitEmu. However, it is illegal for me to provide you with a ROM file, so instead, I will tell you how to easily create your own. You can download Ti-83+ Flash Debugger, and go to the file ti83plus.clc in the Exe folder. Change the filename to ti83plus.rom, and you have your ROM file!

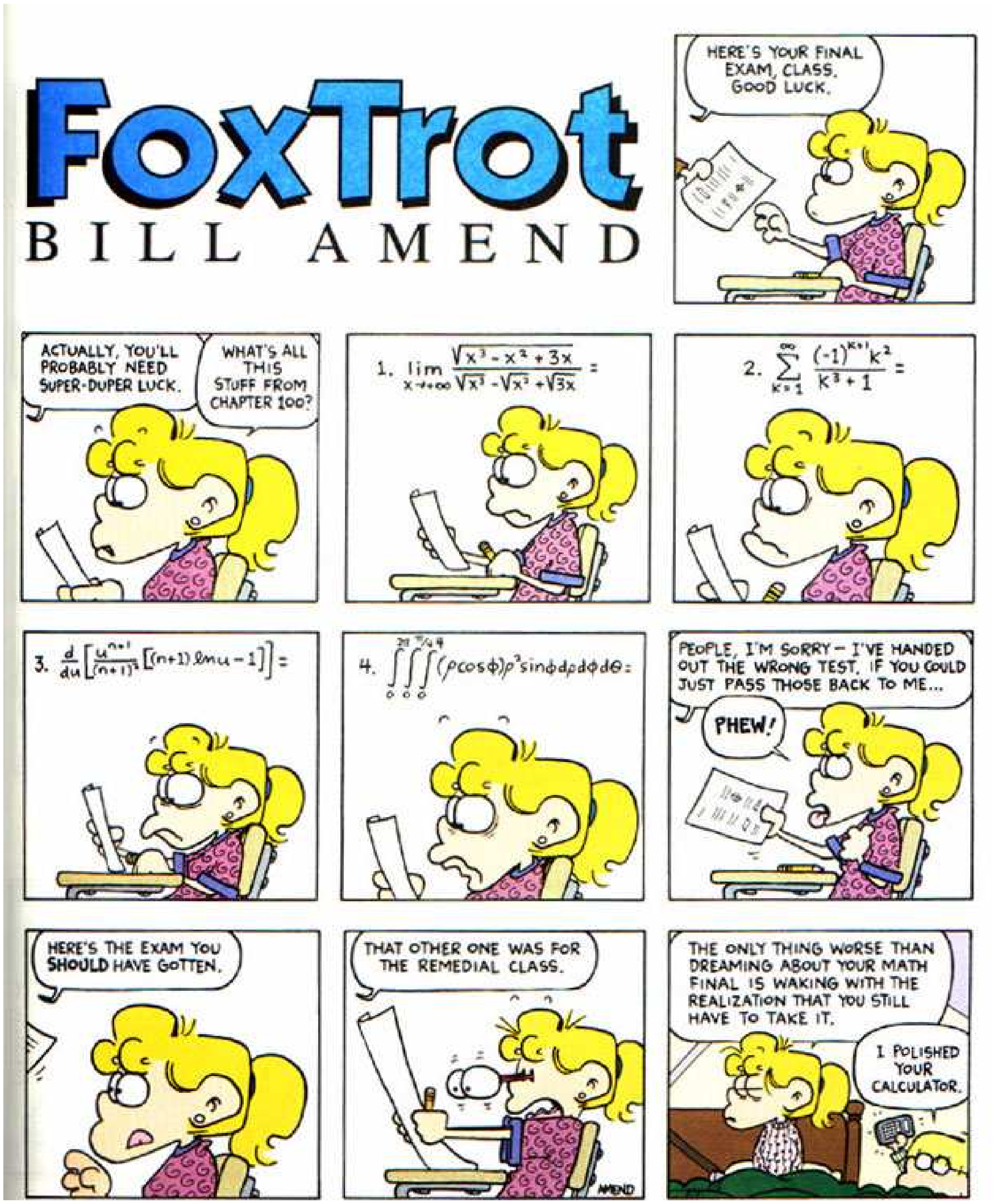
Open your text editor, and type in the following program making sure that you include the tabs. TYPE IT. Don't just copy and paste. I will then tell you how to compile and run it. Afterwards, I promise that I will explain the program to you one line at a time.

This program will solve the addition problem 1 + 5 and display the answer. Again, please type the program, don’t just copy and paste. Also, all indented text should be indented using the tab key.

```
#include “ti83plus.inc”
.org  40339
.db    t2ByteTok, tAsmCmp

        B_CALL  _ClrLCDFull
        ld a, 1
; Solve the problem 1 + 5
        add a, 5
        ld h,0
        ld l, a

        B_CALL _DispHL
        B_CALL _getKey
        B_CALL  _ClrLCDFull
        ret
```


FoxTrot
BILL AMEND
HERE'S YOUR FINAL EXAM, CLASS. GOOD LUCK.
ACTUALLY, YOU'LL PROBABLY NEED SUPER-DUPER LUCK.
WHAT'S ALL THIS STUFF FROM CHAPTER 100?
1. $\lim_{x \to +\infty} \frac{\sqrt{x^3 - x^2 + 3x}}{\sqrt{x^3} - \sqrt{x^2} + \sqrt{3x}} =$
2. $\sum_{k=1}^{\infty} \frac{(-1)^{k+1} k^2}{k^3 + 1} =$
3. $\frac{d}{du}\left[\frac{u^{n+1}}{(n+1)^2}\left[(n+1)\ln u - 1\right]\right] =$
4. $\int_0^{2\pi}\int_0^{\pi/4}\int_0^{4} (\rho\cos\phi)\rho^2 \sin\phi \, d\rho \, d\phi \, d\theta =$
PEOPLE, I'M SORRY – I'VE HANDED OUT THE WRONG TEST. IF YOU COULD JUST PASS THOSE BACK TO ME...
PHEW!
HERE'S THE EXAM YOU SHOULD HAVE GOTTEN.
THAT OTHER ONE WAS FOR THE REMEDIAL CLASS.
THE ONLY THING WORSE THAN DREAMING ABOUT YOUR MATH FINAL IS WAKING WITH THE REALIZATION THAT YOU STILL HAVE TO TAKE IT.
I POLISHED YOUR CALCULATOR.
AMEND

Now save this program as program1.asm, or whatever you want to call it. Give it a name you will remember. Also make sure that this program, spasm and ti83plus.inc are in the same folder.

To compile the ASM program using Windows, run command prompt and go to the folder containing spasm. Type in “spasm,” followed by a space and the name of your text file. Type in another space. Then type in the name of your program. For instance, “spasm program1.asm program.8xp”. Note that your program name will be cut down to 8 characters if it is too long.

The compile instructions are identical for Linux and Macintosh, although I have very little knowledge of the CLI and command terminals for these two computers.

Now, drag your newly created program onto wabbitemu.exe. If wabbitemu asks for a ROM, select the TI83Plus.rom you created. Also, if you are running your program on a Ti-84+ with Operating System 2.53 or greater, make sure your Ti-84+ is running in **Classic Mode.** Afterwards, run your program on the emulated Ti-83+ using Asm().

If you typed everything correctly, you should see the answer to your problem, 6. Two thumbs up! Now for the detailed explanation I promised you.

#include “ti83plus.inc” – To program for the Ti-83+ in the ASM language, most of the functions and data you need is in this file. However, these functions are not standard to ASM programming. They are only used when programming ASM for the Ti-83+! So #include tells spasm to include the functions in this file. For example, ti83plus.inc tells spasm exactly what to do with _getKey. But if you did not tell spasm to include ti83plus.inc, spasm would not know what to do with _getKey, and an error would occur.

.org 40339 – Remember what I mentioned in tutorial #3, about how RAM has addresses so the calculator knows where to find everything? On the Ti-83+, an ASM program must always, always run in RAM starting at address 40339. So .org 40399 tells spasm that this is the section in RAM that the program will be located at.

IMPORTANT: Almost every calculator programmer uses .org $9D93 instead of .org 40339. It is vital that you remember that they mean exactly the same thing. You will find out why later, but from this point on, except for portions of chapter 5, I will be using .org $9D93. This is so that when you ask other people for help they won't be puzzled.

.db t2ByteTok, tAsmCmp – This tells the calculator that the program is an ASM program, not a Ti-Basic program. We'll talk more about .db later.

B_CALL _ClrLCDFull – Stored on the calculator is a function that clears the screen. You cannot access this function using Ti-Basic. (Ti-Basic does something else for the function ClrHome.) You can only access this function using ASM. B_CALL will call this function and clear the screen.

IMPORTANT: You can also use bcall(_ClrLCDFull). Some people prefer this method. I will use B_CALL with no parenthesis throughout the lessons, but lowercase bcall with parenthesis will also work.

; **Solve the problem 1 + 5 –** This is a comment. You can put anything you want in a comment. It could be information about the program, a joke you want the reader to look at, or anything else you want to put in.

When you run spasm to translate your program, spasm will ignore any comments you make. A comment must start with a semicolon.

ld a, 1 – As a quick review, the calculator's processor can't solve a problem (such as 1 + 5) using regular RAM, so it needs to solve the problem using its "working memory," its registers. **A** is one such register. "A" stands for accumulator, and this register is where most of the calculator's math is done. Ld a, 1 is the same as saying, in Ti-Basic, "1 → A," and so we let a = 1. But remember, A **is not a variable.** It is a register, used by the calculator to perform math.

add a, 5 – This function adds 5 to whatever is inside of register A. In this case, since A = 1, "add a, 5" will cause A to equal 6.

ld h, 0 – H is another register. Later we'll talk about what H is mainly used for (because each of the calculator's registers has a special purpose), but for right now we let it equal 0.

ld l, a – L is yet another register. We let L = A. So now L = 6, since A = 1 + 5 = 6.

By the way, H and L can be used as a pair. Since H = 0 and L = 6, HL = 06, meaning HL = 6. For the time being, don't take this and run with it: if H = 1 and L = 7, HL DOES NOT equal 17. (It equals 263, and you will find out later why)

B_CALL _DispHL – Stored on the calculator is a function called DispHL, which will display whatever is inside of HL. Once again, only ASM programs can access this. Since HL = 6 after our addition problem, B_CALL _DispHL will display "6" on the screen.

B_CALL _getKey – Waits for a keypress.

ret – The ASM program will quit upon reaching ret. All ASM programs need to end with "ret", or else the calculator will crash.

Here's another program. This time, you should see the answer 11.

```
#include “ti83plus.inc”
.org  $9D93     ; Remember, this is so you can get help without
                ; confusing people
.db    t2ByteTok, tAsmCmp

        B_CALL  _ClrLCDFull
        ld a, 7
; Solve the problem 7 + 4
        add a, 4
        ld h,0
        ld l, a
        B_CALL _DispHL
        B_CALL _getKey
        B_CALL  _ClrLCDFull
        ret
```

Exercise:

Edit your program three times so that you can solve 55 + 67, 102 + 58, and 200 + 15.

Let’s go back to the program that solved 7 + 4. Change it to the following:

```
#include “ti83plus.inc”
.org  $9D93
.db    t2ByteTok, tAsmCmp

NumberSeven .equ 7
NumberFour .equ 4
          B_CALL  _ClrLCDFull
          ld a, NumberSeven


; Solve the problem 7 + 4


          add a, NumberFour
          ld h,0
          ld l, a


          B_CALL _DispHL
          B_CALL _getKey
          B_CALL  _ClrLCDFull
          ret
```

What happens now? You still get the answer 11. When spasm translates the program, it replaces “NumberSeven” with the number 7, and it replaces “NumberFour” with the number 4.

In the case of this program, NumberSeven and NumberFour are called **constants**. A constant is a number that never changes. Since NumberSeven is always equal to seven, you can use it instead of the number 7 anywhere in this program. Try it:

```
#include “ti83plus.inc”
.org  $9D93
.db    t2ByteTok, tAsmCmp

NumberSeven .equ 7
NumberFour .equ 4
        B_CALL  _ClrLCDFull
        ld a, NumberSeven

; Solve the problem 7 + 7

        add a, NumberSeven
        ld h,0
        ld l, a

        B_CALL _DispHL
        B_CALL _getKey
        B_CALL  _ClrLCDFull
        ret
```

You should get the answer 14.

This wraps up our programming for today. Here are a couple of things for you to look at and think about.

1. Be careful when working with register A. A is one byte, so it cannot be bigger than 255. In fact, try adding 200 + 100. Do you get 300? No, you get 44. Here’s what happens: 200 + 55 is 255. So you have 45 left to add to A. If you add one more to A, A

cannot get any bigger, so it resets to zero. Then you have 44 left to add to A, so A = 44.

2. To subtract a number from A rather than add a number to A, use the instruction **sub**. For instance,

   ```
   ld a, 5

   sub 2
   ```

   This will solve the problem 5 – 2, returning the answer 3.

   However, note that A cannot be less than zero. Just like A resets itself to 0 when you try to make it bigger than 255, A resets itself to 255 when you try to make it less than zero.

Next lesson, we'll look at variables and labels.