TI-83+ Z80 ASM for the Absolute Beginner

APPENDIX A:

- *Creating Flash Applications with SPASM*

CREATING FLASH APPLICATIONS WITH SPASM

I assume that as a Ti-83+ user, you know what an application is. And the nice thing about an application is, you can write very long ASM programs. As you probably know, a normal ASM program cannot be bigger than 8 kilobytes, but an application can be as large as your calculator archive will allow.

However, you need to be aware that applications do not run from RAM. They run from ROM. Furthermore, the calculator needs to know that your ASM program is actually an application. So we have some adjustments to make your program an application.

Please do not try to run any applications you create until read this entire lesson. There are some things you can do with an ASM program that you cannot do with an application, and I need to tell you what those are.

To create a Ti-83+ application, you need a special include file that I have provided. Do you remember how you always needed to type #include “ti83plus.inc” at the beginning of all your ASM programs? After that, you need to type #include “app.inc” if you are creating an application. And just like with ti83plus.inc, make sure you have app.inc in the same folder as your ASM application.

```
#include “ti83plus.inc”

#include “app.inc”
```

Now you need to decide whether you want a multi-page application or a single-page application. Most of the time, you'll want to write a single-page application, which provides you with a maximum size of 16KB. But if you want to write a very big application, you will need more than one page. The point I'm coming to is, you will need at least one .asm file for every page of the application that you write.

So let's say we are writing a single-page application, with one page. This is called **page 0**. (So the second page is page 1, the third page is page 2, etc.) After #include "app.inc," type in the following:

defpage(0, The Name of Your Application In Quotes)

SPASM now knows that this file is page zero of your application. In this case, page zero is our only page.

The name of your application can be no longer than 8 characters. For example,

defpage(0, "Hello")

defpage(0, "TestASM")

defpage(0, "PlayGame")

The very, very last line of your application should be the following: validate() Also, take out the .org $9D93 line and the .db t2ByteTok, tAsmCmp line. These are used only for ASM programs, not applications.

When you build your application with SPASM, DO NOT type .8xp at the end of your program name. Instead, type .8xk at the end.

That's all there is to it! You now have a one-page application. However, there's a bit more work involved when you want to create a multiple-page application.

First of all, recall that you should have one .asm file for every page your application requires. Then you need to include these files by typing #include before validate().

Let's say you have a very big application that requires 3 pages (48 KB). So you need three files. One is your main program, the first page, page 0. We'll call it pagezero.asm. Let's say your second file is called pageone.asm and your third file is called pagetwo.asm. On pagezero.asm, before validate(), you should have the following:

#include "pageone.asm"

#include "pagetwo.asm"

For pageone.asm, at the top, you need the line defpage(1). For pagetwo.asm, at the top, you need the line defpage(2).

When SPASM compiles your application, anything you typed on pageone.asm will go into page 1, the second page of your application. Anything you typed on pagetwo.asm will go into page 2, the third page of your application.

Now, the thing you need to be aware of is, **the Ti-83+ can only run one page at a time, starting with page zero.** If you

need to run some code on another page, you need to **switch pages.** If your Ti-83+ is running some code on page 0 and you need it to run code on page 2, you need to switch to page 2 of your application. Again, if the Ti-83+ is running code on page 1, it will not run code from page 0 until it switches to page 0.

The good news is that the calculator handles all the page switching. The bad news is that the calculator does not know which page to switch or WHEN to switch it. You have to tell it where to go. Your application will need what's called a **branch table**. The branch table is where you put labels that you need to access on a page DIFFERENT from the page you are on.

Let's say you have Label_Draw_Picture, on page 0 of your application. If you need to CALL it, JR it, or JP it from page 0, and ONLY from page 0, you **do not** need to place it in the branch table. However, if you need to access Label_Draw_Picture from another page of your application, you do need to put it in the branch table.

Likewise, suppose Label_Draw_Picture is on page 3 of your application. If you CALL it, JP it or JR it from page 0, 1, 2, 4, 5, etc., you need to place this in the branch table. If Label_Draw_Picture is ONLY used on page 3, you do not need to put it in the branch table.

So, what is this—branch table, and where do we put it? Well, this branch table needs to be before all of your code. At the place where your code actually starts, type in the label Start

```
Start:
```

Then, before all your code, type in the following:

```
jp  Start           ;Goes to the start of your code
.db 0
```

Now, this is where you tell your calculator where all the labels are, the labels that you need to CALL, JP or JR from different pages. All this information should go before the label Start. For every label you need to access on multiple pages, you type in the following:

```
.dw Label_Name
.db Page Number
```

Suppose you have a label on your second page called Access_Sprite_Data, and you need to call it from your first page. Remember, your first page is called page 0, and your second page is called page 1.

```
.dw Access_Sprite_Data      ;The label
.db 1          ;The page the label is on
```

Now, let's say you have a timer function on your third application page, page 2.

```
.dw Adjust_Timer
.db 2
```

So remember, your branch table should include ONLY labels that are accessed from pages that they are not a part of. If you have a label on page 0 that only page 0 uses, you don't need to put it in the branch table.

Now we have our branch table with all the labels that the application needs to access from separate pages. But, the calculator is stupid…it doesn't know that the data is there! So we need to tell the calculator where to find this data.

For every label in the branch table, come up with a nickname. I usually create nicknames by placing "_" before each label in the branch table. For example, _Adjust_Timer and _Access_Sprite_Data.

For each label in the branch table (going in the order you typed the labels in), type in your nickname for that label, then .equ, then the following formula:

(43 + (1 if the nickname pertains to the first label in the table, 2 if the label pertains to the second label in the branch table, 3 if the label pertains to the third label in the branch table, etc.)) * 3.

```
_Access_Sprite_Data .equ 44 * 3

_Adjust_Timer .equ 45 * 3
```

That's all there is to it. The calculator now knows on which pages all the labels are located. However, you cannot use CALL, JR or JP to access these labels that are on different pages, unless your application is running the page the label is on. You use B_CALL nickname to CALL a label on another page, and you use bjump(nickname) to JP to a label on another page.

Since Access_Sprite_Data is on page 1 of your application, you use B_CALL _Access_Sprite_Data when you are on page 0. If you are on page 1, you can use CALL Access_Sprite_Data, and **you don't need to use the nickname in this case.**

Wow, so now you can create single and multiple-page applications. But remember, applications can only run from ROM. That means anything that requires RAM in an ASM program CANNOT be used for an application.

So, you can't use variables! At least not normally. You can't, for instance, say:

```
Number_Of_Bananas:
    .db 99
```

Instead, variables must be stored directly in the calculator's RAM, for instance in appbackupscreen. (Refer to lesson _____ if you need to.)

And then a biggy…when you use routines to display text, your text needs to be located in RAM. So you can no longer say the following:

```
LD HL, Text

B_CALL _PutS
```

You need to copy the string to RAM, and THEN display it. We will use appbackupscreen to hold the area where we want our string copied.

```
TextBuffer .equ appbackupscreen + 100


LD HL, Text

LD DE, TextBuffer

B_CALL _StrCopy

LD HL, TextBuffer

B_CALL _PutS
```

So just remember that applications require ROM, and try not to include code that requires the application to mess around with itself as if it were in RAM.