TI-83+ Z80 ASM for the Absolute Beginner

LESSON ELEVEN:

- *Inputting Binary Numbers*
- *Displaying Full-Sized Pictures*
- *If You Just Have to Have More RAM*

INPUTTING BINARY NUMBERS

So far in these lessons, you could write a program and easily get away with decimal numbers. Sure, hexadecimal numbers were introduced, but whenever you use hexadecimal numbers, it's for stupid tradition. (There is one exception, covered later)

But NOW, we're getting to the point where there will be some numbers better viewed and entered as binary numbers. Not because of tradition, but because of convenience. The next lesson is one such example.

To enter a binary number in ASM, just place a percent sign in front of your number, entered as 1s and 0s. This doesn't work on all assemblers, but it for sure works on SPASM.

Example: ld a, %00001000

is the same as saying ld a, 8

DISPLAYING FULL-SIZED PICTURES

From your Ti-Basic programming, you're probably aware of the size of the screen in terms of pixels. The full screen is 96 pixels wide by 64 pixels high, although you can only use 95 pixels horizontally and 63 pixels vertically for a Ti-Basic game. The full screen, 96 x 64, is available for use in an ASM program.

As you remember from lesson 3, the Ti-83+ has some special areas of RAM reserved to use for variables, functions, etc. There are **two** areas of RAM that the calculator uses for displaying pictures.

The first area is mostly for advanced users, and will not be discussed much in these lessons. You can read "Learn Ti-83+ ASM in 28 Days" to learn how to use this portion of RAM to the fullest extent. However, we do want to know about this area. To make a long story short, whatever you see on the screen of your calculator at any time is held in this RAM. By playing around with this area of RAM, you can change how your screen looks instantly.

The second area of RAM for pictures is what we're most interested in. Called a **buffer**, it is used for storing a picture that you don't want to display immediately.

For instance, if you want to make the screen black by drawing 100 lines, you can draw them to the buffer, and the user won't see them being drawn one at a time. In Ti-Basic, there was no way you could do this with the "Horizontal" command…if you ran the "Horizontal" instruction 63 times to blacken the screen, the user would see the screen get black from top to bottom, rather than all at once, because each line would be seen as soon as it was drawn. With ASM, you can draw these lines to the buffer, and then display the picture after all the lines are

drawn. Then all the user will see is an instant black screen, not several lines being drawn.

Without a buffer, you could not create a first person shooter that would scroll smoothly. The player would see walls being drawn one at a time, rather than all at once. By using the buffer, you can draw walls without the person seeing until you're finished; then by displaying the buffer, everything shows up at once.

So, what is this…buffer? It's an area of RAM called **plotsscreen**. Plotsscreen is a constant for the location/address of this RAM. So you can use HL on plotsscreen, as you probably guessed.

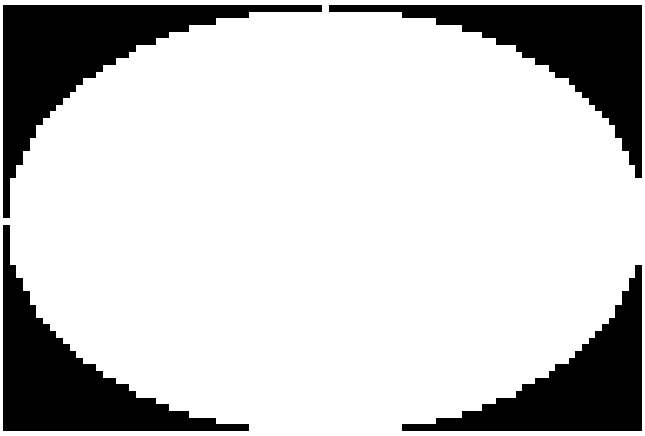
Anyways, plotsscreen holds 768 bytes of picture information. Here's the math: It takes 1 byte to hold eight pixels on the screen (read the next paragraph to find out why). So that means one row of 96 pixels uses 12 bytes. Multiply that by 64 rows, and you have 768 bytes.

But how does this 1 byte tell the calculator what the eight pixels should look like? Remember lesson one? Every BIT (not byte) tells the calculator whether a pixel is on or off. If a bit is equal to 1, the pixel for that bit is black, it's on. If a bit is equal to 0, the pixel is white, or off. Since a pixel is either on or off, only one bit is needed for each pixel. There's eight bits in a byte, so a byte can hold eight pixels. So if you have a value of %10001000 (a binary number), you'll get this kind of picture on your screen:

Plotsscreen will always start at the upper-right hand corner of your screen. So if you store information in the first byte of plotsscreen, you will affect only the first eight pixels of the first row. If you store

information in the twelth byte of plotsscreen (aka plotsscreen + 11), you will affect only the far right edge of the second row. Thus, plotsscreen + 767 will affect the very last eight pixels of the screen on your calculator.

So, if you want to display the following picture on your screen:



The next page shows what the data for the picture looks like. Notice that just like with any other raw data you enter, you need a label.

```
Full_Sized_Picture_Example:

.db %11111111,%11111111,%11111111,%11111111,%11111111,%11111111,%01111111,%11111111,%11111111,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%11111111,%11111000,%00000000,%00000000,%00001111,%11111111,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%11111111,%00000000,%00000000,%00000000,%00000000,%01111111,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%11110000,%00000000,%00000000,%00000000,%00000000,%00000111,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111110,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00111111,%11111111,%11111111
.db %11111111,%11111111,%11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111,%11111111,%11111111
.db %11111111,%11111111,%11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011,%11111111,%11111111
.db %11111111,%11111111,%10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%11111111,%11111111
.db %11111111,%11111110,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00111111,%11111111
.db %11111111,%11111100,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00011111,%11111111
.db %11111111,%11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111,%11111111
.db %11111111,%11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011,%11111111
.db %11111111,%11000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000001,%11111111
.db %11111111,%10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%11111111
.db %11111111,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%01111111
.db %11111110,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00111111
.db %11111100,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00011111
.db %11111000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00001111
.db %11111000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00001111
.db %11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111
.db %11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111
.db %11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011
.db %11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011
.db %11000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000001
.db %11000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000001
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000
.db %11000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000001
.db %11000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000001
.db %11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011
.db %11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011
.db %11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111
.db %11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111
.db %11111000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00001111
.db %11111000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00001111
.db %11111100,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00011111
.db %11111110,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00111111
.db %11111111,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%01111111
.db %11111111,%10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%11111111
.db %11111111,%11000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000001,%11111111
.db %11111111,%11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011,%11111111
.db %11111111,%11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111,%11111111
.db %11111111,%11111100,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00011111,%11111111
.db %11111111,%11111110,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00111111,%11111111
.db %11111111,%11111111,%10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%11111111,%11111111
.db %11111111,%11111111,%11100000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000011,%11111111,%11111111
.db %11111111,%11111111,%11110000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00000111,%11111111,%11111111
.db %11111111,%11111111,%11111110,%00000000,%00000000,%00000000,%00000000,%00000000,%00000000,%00111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%10000000,%00000000,%00000000,%00000000,%00000000,%00000000,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%11110000,%00000000,%00000000,%00000000,%00000000,%00000111,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%11111111,%00000000,%00000000,%00000000,%00000000,%01111111,%11111111,%11111111,%11111111
.db %11111111,%11111111,%11111111,%11111111,%11111000,%00000000,%00000000,%00001111,%11111111,%11111111,%11111111,%11111111
```

So this is the picture stored in your program. By now, you know enough to figure out how to copy this data to the buffer plotsscreen. However, the code you would have to write would be long and tedious. It's time to introduce a new instruction! We want to copy this picture data from Full_Sized_Picture_Example to plotsscreen, and there is an instruction that allows us to do just that, in a few steps.

LDIR

Copies data from one location to another. Use register BC to specify how many bytes you want to copy. Use HL to tell the calculator where you want to copy data from, and use DE to tell the calculator where you want the data copied to.

Examples:

```
LD HL, Full_Sized_Picture_Example
LD DE, plotsscreen
LD BC, 768
LDIR
```

T-States: Depends on how much is being copied.

The formula is

T-States = BC * 21 – 5

Byte Storage: 2 Bytes

Hopefully you remember that the buffer is only used to store a picture you don't want to display immediately. Even after everything has been copied to plotsscreen, we won't see the picture. We need to move the picture to the area of RAM that displays the picture immediately. B_CALL _GrBufCpy will do just that. It copies the buffer to the display RAM.

The next page contains a practice program. I recommend you create your own picture. To do this, save the picture you want as a 96x64 monochrome bitmap.

Type a label for your picture. For the example program, the label for the picture will be Full_Sized_Picture.

Immediately before this label, type in the following:

.option BM_SHD = 2

.option bm_min_w = 96

Finally, make sure your picture is in the same folder as spasm, and type in the following after your label:

#include "picture.bmp", where picture is whatever you decided to call your picture.

SPASM will compile your picture into the ones and zeros needed from the picture.

```
.option BM_SHD = 2

.option bm_min_w = 8

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



        B_CALL _RunIndicOff                ;Turns off the little bar you see running at the side of the screen
        LD HL, Full_Sized_Picture      ;Make sure you have a 96 x 64 picture.
        LD DE, plotsscreen
        LD BC, 768
        LDIR

        B_CALL _GrBufCpy
        B_CALL _getKey
        B_CALL _ClrLCDFull
        B_CALL _DispDone             ;Displays the word "Done" at the end of the program

        ret

Full_Sized_Picture:

#include “Bitmap.bmp”
```

By the way, for the purpose of these lessons, we will be using a lot of B_CALL functions, since they are excellent for beginners. However, don’t get used to them. To be quite frank, they are slow. If you use too many of them, your program will be un-optimized and not many people

will want to use it. Nonetheless, with what we are doing, B_CALLS will be fine.

IF YOU JUST HAVE TO HAVE MORE RAM

While you can store variables inside of a program, sometimes this is not going to be enough for your needs. An ASM program on a Ti-83+ can only be 8 KB in length, meaning that the more space you need reserved for variables, the less space you'll have for code and essential data.

Fortunately, the Ti-83+ has some RAM reserved for ASM programs. One of these areas, called appBackUpScreen (another constant to represent a location, a ram address), holds up to 768 bytes of RAM. You can use this area for anything you want, and it's big enough to hold a picture if need be. Just a quick review, you can store inside the first byte with appBackUpScreen, the second byte with appBackUpScreen + 1, etc.

Don't forget that you can use constants, too. If you are writing a Zelda game and you want to store the X and Y position of the character Link inside of appBackUpScreen, you can do the following:

```
Link_X .equ appBackUpScreen

Link_Y .equ appBackUpScreen + 1
```

Then, of course, you access the values with (Link_X) and (Link_Y).

The other area you can use is called saveSScreen, also 768 bytes. You have to be careful when using this area, making sure that the calculator does not shut down automatically after 5 minutes. To use this area, type RES apdAble,(IY+apdFlags) at the beginning of your program to turn off the automatic power down. Then you can use

saveSScreen the same way you use appBackUpScreen. Use SET apdAble,(IY+apdFlags) to turn on the automatic power down, before you leave your program.