TI-83+ Z80 ASM for the Absolute Beginner

APPENDIX D:

- *Interrupts*

INTERRUPTS

Take a moment to pretend that it's your turn to fix dinner for your family of nine people. You need to start dinner at 3, and have it ready by 5. So you decide to fix a tasty gourmet dinner that needs to be stirred every ten minutes.

Now, are you going to just stare at the food, stir it after 10 minutes, stare at it again, stir it, stare at it? Probably not. You may grab a book, type some emails, play some Xbox, or do something else while the 10 minute timer on your stove is running.

When the timer rings, are you going to finish what you were doing before stirring the dinner? Even if you had one hour before you finished playing an Xbox game? Chances are the food would burn to a crisp if you waited to stir it. Nope, you would **interrupt** whatever you were doing to stir the food, and then you would return to your relaxing until ten more minutes passed.

Sometimes, when you design a calculator program, you have something that needs to happen on a consistent, timely basis, no matter where in the program you are. For example, in a multiplayer game I'm writing, I want to constantly check the link port on the Ti-83+ to see if there's data to receive from the other player, and I want this to happen every $1/118^{th}$ of a second so that the multiplayer game stays in sync. So every $1/118^{th}$ of a second, my program will interrupt itself to run the code that will check the link port. Afterwards, the program will resume running from where it was interrupted.

Pretty cool that you can do this, huh? This is exactly what CALLS do: When you use CALL label, the program goes to the label, and after

reaching RET, the program returns to the line after the CALL statement, so that the rest of the program can run normally. However, CALLs do not happen on a timely and consistent basis. Why is that? Because the CALL will run ONLY when the calculator reaches the instruction in the program code. To state the obvious, the calculator is running your program instruction by instruction, and it won't run CALL if there is no CALL at that point in your code.

What we want is something that will be called at any point in the program whenever a certain amount of time has passed. This lesson is meant to teach you how to do that. You will need appbackupscreen to use interrupts in your program or your application, at least for this lesson. For the most part, we will have an interrupt routine that runs **approximately** every $1/118^{th}$ of a second, but I'll teach you how to change the speed later on in this lesson.

The calculator has 3 interrupt modes: Mode 0, Mode 1 and Mode 2. We want to use interrupt mode 2, since Mode 2 is the interrupt mode that lets us run our own interrupt routine that will run every $1/118^{th}$ of a second. (Mode 0 is not used by the Ti-83+, and we can't use Mode 1 since the calculator uses it.) Be careful: if you use interrupt mode 2, you cannot use _getKey and _getCSC.

To start off, you'll want to code your interrupt routine. (Fear not, there is an example program!) You should keep your interrupt small if you can. Begin it with a label, say "Begin_Interrupt_Routine," and end it with a label, say "End_Interrupt_Routine." Then you'll want to copy it to $9A9A, an address that points to an area in appbackupscreen.

```
        LD HL, Begin_Interrupt_Routine
        LD DE, $9A9A
        LD BC, End_Interrupt_Routine – Begin_Interrupt_Routine
; BC now equals how many bytes to copy
        LDIR
```

But how does the calculator know where your interrupt routine is? Well, that's a little bit tricky. Every 1/118th of a second, the calculator will check one of 256 different locations (of your choice) in RAM for a program location to jump to. When the program finds that two-byte value, it will interrupt itself and jump to that location that it found. So if we want to run Begin_Interrupt_Routine every 1/118th of a second, we need to store $9A9A at these 256 locations the calculator checks. That way, the calculator will always read $9A9A as the interrupt routine, and it will always jump to $9A9A about every 1/118th of a second.

We need to decide which 256 locations in RAM we want the calculator to check. A location is always a two-byte number, as you already know. We can give the calculator the first byte of this two-byte location by using the **I Register**, and this value will never change unless we want it to. The second byte of our two-byte number is always chosen at random…since this second byte can be from 0-255 (00 – FF in hexadecimal), and since register I will never change, that's 256 locations!

We will let Register I be equal to $99. This is because the numbers $9900 to $99FF (a total of 256 locations) all point to appbackupscreen. You can only access register I by using register A: Either LD I, A or LD A, I.

```
LD A, $99

LD I, A
```

Now that the calculator knows where to look for places to jump to every 1/118th of a second, we need to make sure that the calculator will always jump to $9A9A.

```
LD HL, $9900    ; The Beginning of the 256 locations that
                ; the calculator will search
LD B, 0         ; Believe it or not, if you use DJNZ on B = 0,
                ; your code will loop 256 times.  So we can
                ; store $9A9A to 256 locations!
LD DE, $9A9A    ; The location of our interrupt code
Store_Interrupt_Code_Location:


ld (hl), D
inc hl
ld (hl), E
DJNZ Store_Interrupt_Code_Location
```

As was aforementioned, the advantage to interrupt routines is that they can occur anywhere in the program, every 1/118th of a second. But what happens if you jump in the middle of a routine, and your registers have very important values that you can't afford to lose? If your interrupt routine uses any of those registers, you lose those values that the registers previously held.

You could solve this by pushing AF, BC, DE and HL during the interrupt routine, and then popping them before you exit your interrupt routine. Then your registers will hold whatever values they had before your code was interrupted. But there is a better way to save your valuable register data besides PUSH and POP. EX AF, AF' will save the value of AF temporarily, and EXX will save the values of BC, DE and HL. You must use these at the beginning of your interrupt routine, and again at the end of your interrupt routine.

There are some more simple instructions you need to know about, but I will give them to you in the form of the example program on the next few pages. This program will draw sprites on the screen, but it will calculate approximately how many seconds have passed, so that when you exit, you can see approximately how much time you spend goofing off on your calculator. Exit the program by pressing 2nd, then divide the result by 118 to see **approximately** how many seconds have passed. (If you are using a Ti-83+ Silver Edition or a Ti-84+, divide by 107.79 to get an approximation.)

```
#include "ti83plus.inc"

#define Use_RAM_Routine

.org $9D93
.db    t2ByteTok, tAsmCmp

            jp Routine

#include "fastcopy.asm"
#include "spriteroutines.asm"

routine:

;Even though appbackupscreen is needed for our interrupt code and data, it is safe to use at least the 700th byte of appbackupscreen.
Number_Of_Seconds .equ appbackupscreen + 700
End_Program .equ appbackupscreen + 702  ; This value of the variable will be equal to 1 if it's time to end our program.

            B_CALL _ClrLCDFull
            ld a, 0
            ld (End_Program), a    ; We don't want to end the program, so we let this value equal to 0.
            ld hl, 0
            ld (Number_Of_Seconds), hl        ; No time has passed yet

            di            ; DI means Disable Interrupts. We do not want interrupt routines running while we are preparing the interrupt code.
            ld hl, Interrupt_Routine
            ld de, $9A9A
            ld bc, End_Of_Interrupt_Routine - Interrupt_Routine
            ldir
            ld a, $99
            ld i, a
            ld hl, $9900
            ld b, 0
            ld de, $9A9A
Store_Interrupt_Code_Location:
            ld (hl), d
            inc hl
            ld (hl), e
            djnz Store_Interrupt_Code_Location
            im 2          ; We want interrupt mode 2.
            ld a, %00000110                    ; Tell the calculator we want an interrupt to happen every 1/118th of a second.
                                               ; You'll learn how to do this at the end of this lesson.
            out (4), a
            ei            ; Enable Interrupts.

            call fastcpy

            jr Draw_Picture_On_Screen

Draw_Picture_On_Screen:
;To draw our picture, we are going to use the sprite routine from appendix B.
;The picture scrolls from left to right.


            ld a, (X_Position)
            inc a
            ld (X_Position), a

            ld b, 16
```

```
		ld c, 2
		ld d, a ;Our X position changes constantly
		ld e, 0
		ld ix, Smiley_Face
		ld iy, 16 * 2 + Smiley_Face

		call LargeClippedMaskedSprite


Interrupt_Routine:
		ex af, af'			; Saves the values of our registers
		exx
		ld hl, (Number_Of_Seconds)
		inc hl
		ld (Number_Of_Seconds), hl

;Detects for a keypress without using _getKey, since interrupt mode 1 cannot be used.  If you did not read appendix C, don't try to understand this code.
		ld a, $FF
		out (1), a
;See if the second key is pressed
		ld a, $BF
		out (1), a
		nop
		nop
		in a, (1)
		BIT 5, a
		jr z, End_ASM_Program
		exx
		ex af, af'
		ret				;Exit our interrupt routine and return to the place where the program was interrupted


X_Position:

 .db 0

End_ASM_Program:
		im 1		;It is VERY important that you turn interrupt mode 1 back on before your program ends, because the calculator needs mode 1.
		ld IY, flags
		B_CALL _ClrLCDFull
		ld hl, (Number_Of_Seconds)
		ld a, 1
		ld (End_Program), a
		B_CALL _DispHL
		B_CALL _getKey

		ret
End_Of_Interrupt_Routine:

.option BM_SHD = 2
.option bm_min_w = 16
.option bm_msk = TRUE
.option BM_MSK_RGB = $00FF00

Smiley_Face:

#include "Smile.bmp"
```

Notice that I did not use CALL Use_IY_Safely and CALL Return_IY_To_Normal? When you use interrupt mode 2, IY is free to use. But when you return to interrupt mode 1, you must type in LD IY, flags. Otherwise, your calculator will crash.

Remember, try to keep your interrupt routines small, so that you don't run out of space in appbackupscreen. If you MUST have a large interrupt routine, place some code in another area, and CALL it from your interrupt routine.

Now, did I hear someone ask "How do I set an interrupt speed?"

CODE	SPEED
ld a, %00000110 out (4), a	Slowest. This is the default, and the speed used for the example program. About 118 times a second on a Ti-83+
ld a, %00000100 out (4), a	Medium-Slow. About 170 times a second on a Ti-83+
ld a, %00000010 out (4), a	Fast—About 248 times a second on a Ti-83+
ld a, %00000000 out (4), a	Fastest—About 560 times a second on a Ti-83+

One more thing: It is usually a good idea to disable interrupts while your interrupt routine is running. You don't want your interrupt routine to interrupt itself, or your program will never run. (In the

example program, I didn't disable interrupts, or else the program wouldn't count seconds correctly.) If you disable interrupts inside of your routine, however, you must reset the timer. Don't ask me why, I don't know why, but you must reset it.

To do so, use the following code at the end of your interrupt routine:

```
ld      a,%00001000
out     (3),a
ld      a,%00001010
out     (3),a
ei              ;Enable Interrupts Again
```