TI-83+ Z80 ASM for the Absolute Beginner

LESSON EIGHT:

- *The F Register*
- *If-Then Statements in ASM*

THE F REGISTER

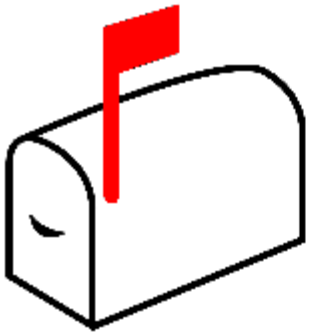
Unlike the registers you worked with so far, F is a one-byte register that you CANNOT modify. This register is used by the calculator's processor to see if certain events have occurred. It consists of a whole bunch of "flags," and you can think of these flags as mailbox flags.

Think about it. If you need to send a letter to someone, you need to let the mailman know that you have mail in your box that needs to be sent. To do this, to let the mailman know, you put the flag up on your mailbox. When you have no mail, the flag is down.

Sadly, when we refer to register F, we don't say that flags are "up" or "down." We say they are "set" or "reset." If an event has occurred, a flag is set. It's like if your mailbox has mail, the mailbox flag is "set" to the "up" position. If an event hasn't occurred, the flag is reset, almost like if your mailbox doesn't have mail, you "reset" the flag to its down position, its original position.

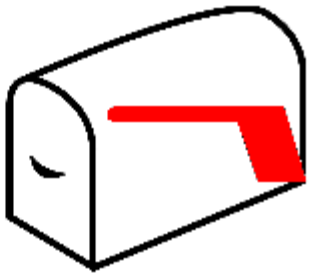
An event has occured, "The mailbox has mail"

The flag is "set" to its "up position"



An event has not occured, "There's no mail"

The flag is down, it is "reset" to its original position



There are several events that F will tell you have occurred or not. However, we are only concerned with two for these lessons. You can look in appendix D if you're curious about the other flags, which aren't used as often.

THE ZERO FLAG is set if any calculation causes a register to equal zero. If the zero flag is set, this is represented as the letter Z. If a calculation DOES NOT result in a register being equal to zero, the zero flag is reset, represented as NZ (meaning not-zero.)

Here are some examples of calculations / routines that will set the zero flag:

```
ld  a, 1
dec  a


ld  a, 250
ld  e, 6
add  a, e  ; Remember that a cannot be bigger than 255, so it resets to zero
```

For a reason I don't know about, ld register, 0 will not set the zero flag.

THE CARRY FLAG is set if you go over a register's maximum or under a register's minimum. In the case of one-byte registers, the carry flag is set if you go over 255 or under zero with a calculation. The carry flag is reset if this doesn't happen, that is, if you stay inside the range of the register. C means the carry flag is set, and NC means the carry flag is reset.

Here's some examples of some routines that will set the carry flag.

```
ld e, 0
dec e

ld  a, 250
ld  e, 6
add  a, e

ld  a, 255
inc  a
```

So how do we use the zero flag and the carry flag? We use them in If…Then statements. However, there's a different approach for If… Then in ASM programming.

IF-THEN STATEMENTS IN ASM

If you've programmed in C++, Basic, Ti-Basic (assumed) or Java, you'll find the if-then-else statements to be very similar to each other. You're going to find that you need a little bit more work to do this in ASM.

Let's start by introducing a new instruction: CP

CP **One-Byte Value**

CP means compare, meaning you compare a value with whatever is inside of A. CP subtracts the one-byte value from A. A is unaffected. For example, if A is equal to 86, CP 56 will mean 86 – 56 = 30. However, A will still equal 86.

Examples: CP 10

T-States: 7 Byte Storage: 2 Bytes

Although CP does not affect A, it DOES affect **flags**. Let's suppose that A is equal to 90. Let's say you use the instruction CP 90. Then A – 90 = 90 – 90 = 0. So the zero flag is set. The carry flag is reset, since A – 90 does not go below zero or above 255.

Let's suppose that A is still equal to 90, and you use the instruction CP 89. Then A – 89 = 90 – 89 = 1, which is not equal to zero. So the zero flag is reset, not set. ALSO, the carry flag is not set, since 90 – 89 does not go below zero or above 255.

What if you use CP 91? Then A – 91 = -1, but A cannot be below zero! So A becomes 255, and the carry flag is set, since 90 – 91 went below zero.

Did you notice something? In the first example, CP 90, 90 was equal to A. In the second example, 89 was less than A. In the third example, 91 was greater than A. What does this sound like? "If A = 90, if A < 91, if A > 89, etc."

Exercise: For each value of register A, and for each CP, figure other whether the zero and carry flags will be set or reset.

1. A = 40, CP 12
2. A = 36, CP 36
3. A = 49, CP 48
4. A = 60, CP 60
5. A = 55, CP 56
6. A = 80, CP 96

Answers:

1. Both the zero and carry flags are reset.
2. The zero flag is set, the carry flag is reset
3. Both the zero and carry flags are reset
4. The zero flag is set, the carry flag is reset
5. The zero flag is reset, the carry flag is set
6. The zero flag is reset, the carry flag is set

Now that you understand how CP is similar to If statements and sets flags, suppose that you had limitations in a Ti-Basic program. Let's pretend that you could do only three things depending on if a condition was true or false: Goto a label, run a subprogram, or stop your program/subprogram.

For example (and the conditions are just example conditions):

```
If A = 90
Goto AE  (Any Label)
If A < 91
Stop
If A > 89
pgrmSUBPGRM  (Any program)
```

What if that was all you could do? You can still use Lbl AE or pgrmSUBPGRM to run commands, but the fact remains that you could not just say "If this is true, do this" directly. You'd have to use your label/subprograms to tell the calculator what to do if a condition is true.

That's the way it is with ASM. You can only jump to a label, call a subroutine (explained in a moment), or exit a subroutine/end the program when using an if-then in ASM.

Now before we go return to if-then statements, what's this about subroutines? It's the same concept as running subprograms. You know how in Ti-Basic, by using the pgrm command, you can run a program from inside your program? For example, let's say you have a routine for drawing text that you have to use, say, 50 times. Do you paste the code 50 times in your Ti-Basic program? Well I certainly hope not! That would use a lot of wasted space, wouldn't it? No, you put the sprite routine in another program, and you run your subprogram every time you need it. Another term for this is "calling" your subprogram.

Just like you jump to a label in ASM, you "call" a label to run a subroutine in ASM.

CALL **Label**

CALL will run a subroutine in your ASM program, like running a Ti-Basic subprogram. Use RET to exit the subroutine and return to the line after your "CALL" line.

Examples: CALL Increase_Number_Of_Lives

T-States: 17 Byte Storage: 3 Bytes

Okay, back to If-Then. Remember that CP will set or reset flags. Then you check the flag of interest, telling the calculator to jump, call, or ret only if that flag is true/false to your liking. The next four paragraphs will be summarized in a table, but be sure to read them carefully nonetheless.

If you want to see if A is equal to a value, use CP value and check the zero flag. This is because A – value = 0 if A equals the value that you check. This means you check to see if the zero flag is set for A equal to a value. If you want to check and see if A is NOT equal to a value, check and see if the zero flag is reset.

If you want to see if A is less than a value, use CP value and check the carry flag. If the carry flag is set, that means A – value is less than zero, since the value is greater than whatever is in A. (For example, look at our previous example on page 6, where A = 90 and you use CP 91) Remember that the carry flag is only set if you go outside a register's maximum or minimum range. Since A – value goes below zero, the C flag is set.

If you want to see if A is greater than a value, you have to be careful. You might say at first, “Well, all I have to do is check to see if the carry flag is reset.” After all, for example, if A = 90 and you use CP 89, 90 – 89 does not go below zero, so the carry flag is reset. This means A=90 is greater than 89. HOWEVER, what if you use CP 90? Then A – 90 = 0, which still resets the carry flag. This means, technically, if the carry flag is reset after you use CP, A is greater than OR EQUAL to a value. If you want to see if A is strictly greater than a value, you need to increase the value you check in CP by 1, then check to see if the carry flag is reset. For example, if you want to see if A (in this case, 90) is greater than 89, you have to use CP 90.

Also, if you want to check and see if A is less than OR EQUAL to a value, you need to decrease the value you check in CP by 1, then check to see if the carry flag is set.

TI-BASIC STATEMENT	ASM EQUIVALENT	FLAG CONDITION TO CHECK
If A = 90	CP 90	Z (Zero)
If A != (Does not equal) 90	CP 90	NZ (Not Zero)
If A > 90	CP 91	NC (Not Carry)
If A >= 90	CP 90	NC (Not Carry)
If A < 90	CP 90	C (Carry)
If A <= 90	CP 89	C (Carry)

Exercise: For each condition, state the CP value function you should use and the flag condition you should check.

1. If A = 20
2. If A <= 80
3. If A != 69
4. If A > 60
5. If A = 127
6. If A >=77
7. If A < 21

ANSWERS:

1. CP 20, Z (zero flag set)
2. CP 79, C
3. CP 69, NZ
4. CP 61, NC
5. CP 127, Z
6. CP 77, NC
7. CP 21, C

JR **Condition**, **Label**

JP **Condition, Label**

Jumps to a label ONLY if the condition you specify—Z, NZ, C or NC—is true.

Examples: JR Z, Label ; Jumps only if the zero flag is set

JP NC, Label ; Jumps only if the carry flag is reset

T-States:	Byte Storage:
JR—12 if condition is true	2 Bytes
7 if condition is true	2 Bytes
JP – 10 no matter what	

CALL **Condition**, **Label**

Calls a subroutine ONLY if the condition you specify is true.

Examples: CALL NZ, Label ; Calls if the zero flag is reset

T-States: 17 if condition is true Byte Storage: 3 Bytes

10 if condition is false

RET **Condition**

Ends your program (or the subroutine you called) ONLY if the condition you specify is true

Examples: RET C ; Ends only if the carry flag is set

T-States: 11 if condition is true Byte Storage: 1 Byte

5 if condition is false

As you may have guessed, these are the only four instructions you can use after a CP "If" statement. If you're having trouble understanding the above 9 pages, check out these example programs before deciding if you need to read the chapter again.

A program that uses JR and RET in If...Then Statements

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

        B_CALL _ClrLCDFull

        ld  a, 1

        cp 1            ; If A = 1
        jr z, DisplayNumber1            ; A – 1 = 0.  So, if A = 1, CP 1 will
                                        ; give A – 1 = 1 – 1 = 0, therefore the
                                        ; zero (Z) flag will be set
        cp 2
        ret z                           ; If A = 2, end the program

DisplayNumber1:

        ld h, 0
        ld l, a         ; Remember that CP does not affect A.  A still equals 1.
        B_CALL _DispHL
        B_CALL _getKey
        B_CALL _ClrCLDFull

        ret
```

A program that uses the CALL routine. From this point, program text will be smaller. You should zoom in if you find it hard to read.

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

        B_CALL _ClrLCDFull

        ld  a, 27

        cp 27               ; If A = 1
        call c, Display_Register_A          ; If A < 27.  Subtracting 27 from A = 26 or less will
                                            ; cause A to go below zero, thus setting the carry
                                            ;flag
        ; If Display_Register_A is run, the program will run here when it is finished
        B_CALL _ClrLCDFull
        ret                                 ; End the program whether A < 27 or not



Display_Register_A:
        ld  h, 0
        ld  l, a
        B_CALL _DispHL
        B_CALL _getKey
        ret                         ; Don't forget to use ret to end the subroutine
```

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

          B_CALL _ClrLCDFull

          ld  a, 0

Increase_A_Until_A_Is_Strictly_Greater_Than_30:
          inc a
          cp 31
          jr c, Increase_A_Until_A_Is_Strictly_Greater_Than_30    ; We only want to go to Next_Step
                                                                  ; if A is greater than 30.
                                                                  ; Note that since C is the opposite of
                                                                  ; NC, this is the same as saying
                                                                  ; jr  nc, Next_Step.  In other words,
                                                                  ; if A is not greater than 30, A
                                                                  ; will be less than or equal to 30.

Next_Step:
          call Display_Register_A
          ret

Display_Register_A:
          ld  h, 0
          ld  l, a
          B_CALL _DispHL
          B_CALL _getKey
          ret                          ; Don't forget to use ret to end the subroutine
```

Be aware that you can only use CP for register A. However, you can use a register to compare with A.

CP **One-Byte Register**

Subtracts the value in the one-byte register from A. A is unaffected. For example, if A is equal to 86, and E = 56, CP E will mean 86 – 56 = 30. However, A will still equal 86.

Examples: CP H

T-States: 4 Byte Storage: 1 Byte

Just remember from the table, if you need to check to see if A is less than OR EQUAL to the value in the register, you need to decrease the value in the register of your choice by 1. Also, if you need to check to see if A is strictly greater than the value in your register, you need to increase the value in your register by 1.

Next week, you'll learn how to display text! Great job making it this far!