



A Professional Journal Exclusively for the Heath/Zenith Z-100 Computer

Z-100 LifeLine Web Site: <https://z100lifeline.swvagts.com> (new effective September 2019)

HOWGOZIT	1
Additional Comments on CP/M	1
Additional Comments on the 22DISK Utility	2
Disassembling CP/M-85's ASSIGN.COM	Insert
CP/M Assembly Language Programming Essentials	Insert

HOWGOZIT

Merry Christmas!! Can you believe it? As a child I often wished Christmas would come sooner. Now, as an adult, it seems to come every month!

Well, while the mid-west is again getting clobbered by snow, we here in North Carolina are seeing record temperatures. I have my plywood snowman out in the front yard with his "Let It Snow!" sign hoping for snow, but it looks like another futile jesture. My snow dance hasn't worked yet either.

I've been working hard on my two versions of CP/M trying to learn a rather simple assembly language for the 8085 CPU (compared to the DOS on 8088), but I'm not too proud to admit that the going has been slow. I've been getting some moral support from two of my friends and it has helped a lot just being able to vent my frustration on them. Thank you both, Thierry Klein and Charles Hett.

In this issue, we are going to continue the discussion on CP/M with a disassembly project that may interest a few of you.

As this is a learning experience, I'm going to go into great detail. Just keep in mind that I'm am not a very good programmer and this is almost as new to me as it may be to some of you. The key here is not to rush this along. We must take our time and be meticulous, otherwise we will end up chasing errors that should have been very obvious. However, I have been having a great, though sometimes frustrating, time and success has been a terrific reward to the challenge. I hope you will also enjoy it.

Additional Comments on CP/M

I've been somewhat disappointed with CP/M-Plus. If some of you feel differently, please feel free to straighten me out.

As I recall, CP/M-85 worked well for me, did the basic computing that I needed it to do very well, but lacked a few things that in Z-DOS were taken for granted.

One huge irritant was the ED utility, unimproved under Plus. How could that program have ever been called an editor? I'm glad that I found Magic Wand early in my CP/M days. Otherwise, my computing days would probably have ended quite early, with the computer left out in the cold.

Plus' ASSIGN program lacks the ability to assess the hard drive partition information that the earlier version had. Sorely missed, it is the reason for disassembling both to see if I can add the missing capability.

Plus does not include all the utilities that CP/M-85 had. Utilities, such as STAT, had to be taken from CP/M-85 and for some reason has problems calculating space remaining under Plus - another program I'll have to look at. But if you didn't have the earlier version, you were out of luck.

The biggest improvement, however, would be the addition of having date/time stamps on files - a necessity when trying to keep different draft versions straightened out. And this recent project brought it to a head.

Thankfully, I'm doing my code writing on a PC because I had three programs I was working

on at the same time - for EACH of the two versions of CP/M I was running - as I tried to reduce my main program, ASSIGN, into smaller elements that I could troubleshoot without wading through the main program. Date stamping was critical!

I'm finding CP/M-Plus to be entirely different from CP/M-85. Completely rewritten, not a single routine resembles its predecessor. Though more capable, accepting up to four disk drives of the three main types - 5", 8", the Hard Drive partitions, and a Memory Disk - its entire structure has been changed. And Disk I/O is a completely different animal, so I'm not learning one language, but two. And the lack of source code and explanations in both version's manuals make it much more difficult to follow, more of a guessing game for further experimentation.

I'm still working on Plus' ASSIGN program and will let you know how I make out. If I'm successful, I may be able to offer a much improved version to play with.

I also looked closely at CP/M-86. However, it is nothing but a 16-bit version of CP/M-85 and suffers the same problems - lack of date/time stamps, still limited to 2 drives of each type, etc. So, it appears that CP/M-Plus is the way to go. I did find a RDDOS file, however, that I'll try under CP/M-Plus.

Additional Comments on 22DISK CP/M <-> DOS Disk Interchange Utility

As mentioned last time, 22DISK is a greatly needed utility that provides the capability of reading and writing CP/M diskettes on a PC clone. It had installation difficulties on my PC, though I'm still not sure if the problems were not generated by me. In any case, the installation instructions could be vastly improved.

However, this utility, once installed, has become a necessity to my programming work, nearly as important as the multi-window display on the PC. Running WORDPAD on the PC, I can quickly generate new programs, copy sections of code at will between windows, or search and update code in a file. Then I copy the file to a CP/M 5" floppy, take it to the Z-100, and try assembling and running it.

New program not working? It is easy to run it under SID, the CP/M-Plus debugging utility (one of the few things that I like from Plus and it runs equally well under CP/M-85) and trace through the affected code. The finished .COM program can be copied back to the PC for storage with the rest of my CP/M library.

While there are CP/M simulators out there that will run on the PC, I'm not sure if they can be tailored to the Z-100 environment for the system work that I've been doing. They may be worth a try, however, if you have some generic programs that you already have or want to try under any generic CP/M program.

Closing

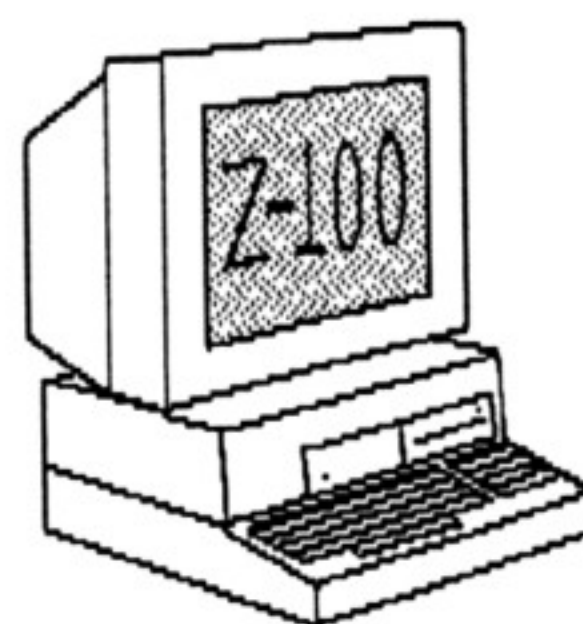
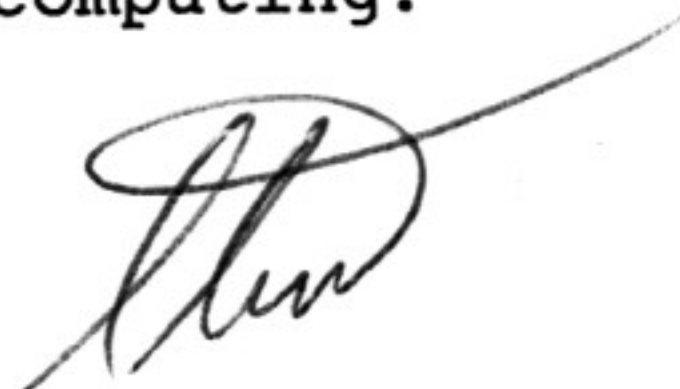
My New Year's resolution is to remodel our kitchen and family room over the next several months. So time spent on Z-100 projects will be very limited. However, I hope that this issue's disassembly project will help spark your interest in CP/M programming.

If you are so inclined, I hope you will give the attached procedures a try. In the next issue I hope to continue digging a bit deeper into CP/M. I'll let you know how I make out.

If you need software, I can e-mail it to you or send a CP/M disk to get you started. Also, please feel free to contact me with questions or suggestions on other changes for CP/M.

Cheers!!!

'Til next time,
happy computing!



Z-100 LIFELINE

Supporting the H/Z-100 Community
Since 1989

Steven W. Vagts
211 Sean Way
Hendersonville, NC 28792
(828) 685-8924
Email: z100lifeline@swvagts.com

Don't forget our Z-100 LifeLine Web Site:

<https://z100lifeline.swvagts.com>

Z-100 Parts & Service
DOS v3 Software & Documentation
MTR-ROM v4 & Z-DOS v4 Software
Z-207 w/High Density Drive mods
Z-205 Mods and RAMDisk Software
Past "Z-100 LifeLines" on CD-ROM
Z-100LL Software Library on CD-ROM



Disassembling CP/M-85's ASSIGN

In this issue I wanted to outline my procedures for disassembling CP/M-85's ASSIGN.COM utility. The procedures may be boring to most of you, and I apologize for that; however, if you were ever tempted to dig into assembly code, this is the time to do it.

We will be using assembly code for the Z-100's 8085 CP/M computer processing unit, which is considerably easier than the assembly code for its close cousin, the 8088 DOS computer processing unit with its large number of double registers.

So, if you are so inclined, please join me as we look into the workings of one of CP/M-85's most important utilities - ASSIGN.

I've collected much of the material that we will need from numerous other CP/M source files and included them in this issue for easy reference. However, for a more thorough understanding of these important concepts, I encourage you to actually read the manuals, if you have them available.

Please don't get intimidated by the quantity of material. I included it all, even though we'll only need a small portion of each and I'll try to explain the workings of each section as we go.

For space considerations, I'm not including the entire generated file, as it is very repetitive. I will only include the primary areas of interest to us. If you follow the procedures given, you can easily generate the entire file on your own.

So, let's get started.

I like the older CP/M-85 (CP/M v2) version of ASSIGN much better than the new CP/M-Plus (CP/M v3) version, but also wanted to add some enhancements. For example, in CP/M-85's version, ASSIGN ? was meant to list all the directories on the hard drive and there was no help file. So, as long as we are making changes anyway, I changed ASSIGN ? to provide a help screen and added a new function, ASSIGN * to list all the directories. The changes were simple, and were one of the reasons for going through all these arduous procedures in the first place.

Notes:

-- As CP/M-Plus (hereafter called CPM3) will be my primary CP/M version of interest, I copied CP/M-85's (hereafter called CPM2)

version of ASSIGN to CPM3 as ASSIGN2.COM. It will NOT work under CPM3; however, the following procedures will be using the CPM3 version of SID, which has capabilities that seemed more logical to me, and beyond those of DDT. I will also be using CPM3's PUT utility to our advantage later.

-- When it finally comes to reassembling the newly modified program, copy it to the CPM2 partition and Boot to CPM2. CPM2 uses the ASM utility to create a .HEX file, then the LOAD utility to create the .COM file from the .HEX file.

-- I am not an expert at any of this disassembly/assembly stuff. I'm muddling along on my own, experimenting, and reporting how I did it. Please, if you have any suggestions or improvements that would make this arduous process easier, send them to me and I'll be happy to report them in the next LifeLine.

-- If you are interested in any of the work being discussed here, but are lacking any of the programs, I have copies of all that I would be happy to send to you for a nominal fee of \$5.00 per disk to cover shipping and my time. However, do not be tempted to search the web and buy this software on-line. You must use CP/M versions designed for the H/Z-100!

OK, enough preliminaries. Let's get to work. The following procedures were used to disassemble CP/M-85's version of ASSIGN:

- 1) Boot to CPM3 and invoke **SID ASSIGN2.COM** and use the 'D'ump command to look at the entire program to find all the data strings that might be useful. Identify the start and stop addresses of each of these areas.
- 2) Using the {CTRL}-{P} command, create a printout of the entire program, from address 0000 to the end of memory used. Use the 'L'ist command to print out the coded statements, and when you reach an area containing strings, use the 'D'ump command.
- 3) On the printout, look for all **RET's** and **JMP's** and draw a long line across the page following their statement. These identify the ends of routines and the starts of new routines.
- 4) Now look at the first part of the program. The program actually begins at address 0100. But there is an area of memory below that is called 'Page Zero'.

----- PAGE ZERO -----

Addresses below 0100 (the beginning address of all CP/M programs) are in an area called Page Zero of memory. This area functions as an interface to the BDOS module from the Console Command Processor. See the section called "Page Zero of Low Memory..." in this issue for more information.

Note: Page Zero in CPM2 and CPM3 are different. For a copy of CPM2's Page Zero, invoke SID under CPM2 and use the 'D'ump command, e.g., #D0000.

However, please note that many opcodes in our program may reference these numbers as addresses or constants. Until you actually analyze the context, it is difficult to decide which they are. As they are eliminated as addresses (they were constants), they may be deleted as no use to us here. I left four as examples:

;ADDR?:	Reference:	Used at Addresses:
;0000	LXI Reg	06D7, 06FB
;0004	LXI Reg	033E, 0368
;0010	LXI Reg	0332, 0360, 0398
;0048	LXI Reg	0203

----- GENERAL EQUATES -----

They don't show up in our disassembled program yet (we have to add them ourselves later), but there are often EQUates listed at the beginning of the source code to identify some of these Page Zero areas:

```

BOOT    equ    0000h    ;System reboot
BDOS    equ    0005h    ;BDOS entry point
BIOSf    equ    004Eh    ;BIOS entry point
FCB1    equ    005Ch    ;First filename
SFCB    equ    FCB1     ;Source FCB
FCB2    equ    006Ch    ;Second filename
DBuff    equ    0080h    ;Default Disk Input Bufr
TPA      equ    0100h    ;Begin of Trans Prog Area

```

Make a note to add these later. Just for grins, using SID under CPM2, I 'D'umped the first line of code at address 0000 and found:

```

0000: C3 03 FB 00 00 C3 00 DA
      0F 00 10 00 20 00 FB 40

```

Unassembled, the two highlighted sets of numbers above become:

```

0000  JMP  FB03 ;CP/M-85 BDOS System Reboot
0005  JMP  DA00 ;BDOS Calls Entry Point

```

Later, you will find that 0000 is JMPed from addresses 026B, 037A, 0392, 043E, 046E, 047E, etc.

And BDOS, 0005, is CALLED from addresses 0382, 03F3, 042A, 0462, 04A8, 04AE, etc.

----- BDOS FUNCTION CALLS -----

BDOS in the EQUate above is the BDOS function call. These are listed separately in this issue in the section called "BDOS Function Calls..." for your reference.

5) Look for the CALL's to 0005 and note these on the printout. Generally, register C has the BDOS function number, and register pair DE has the Entry Value to be used in the CALL function. If not familiar with a particular function, it may help to create an info box with a description to remind you of its function, register use, etc.

----- BIOS ENTRY POINTS -----

BIOSf in the EQUate above is the BIOS function Base Address entry point. These are listed separately in this issue in the section called "BIOS Entry Points..." for your reference.

6) In our code, these are not as obvious as the CALL's to 0005. Instead, we need to look for a routine that has the opcode PCHL. You should find it at 05AE in our ASSIGN program. If this is correct, mark all CALL's to 05AE as BIOS calls.

Note: The sequence of opcodes become obvious after a bit:

```

LHLD 004E    ;BIOS Base Addr
LXI  D, (BIOSf# in hex; 001B, 001E, etc.)
DAD  D       ;HL=BIOS jmp vector
CALL 05AE    ;PCHL & Return

```

Note: The opcode PCHL is an indirect call to the address in HL (contains the BIOS jump vector address). HL has the BIOS Base Address and DE has the offset for the JMP Vector.

Again, if not familiar with a particular function, it may help to create an info box with a description to remind you of its function, register use, etc.

----- DEFAULT DMA BUFFER -----

One of the EQUates, DBuff EQU 0080h, takes additional explanation. The DMA address is used as the default drive buffer and for storing any Command Line parameters, which may be used and processed by the program.

For example, from within the CPM2 partition, if you wanted to assign a drive letter to the CPM3 partition using ASSIGN, you would enter the command:

```

ASSIGN B:=CPM3
or    ASSIGN B:=CPM3;CPM3

```

The B:CPM3;CPM3 is a command line parameter string. However, when we use the SID debugger to work with ASSIGN.COM, we have to enter the

command:

SID ASSIGN.COM

and can't enter these parameters in the normal manner. We must use SID's 'I'input Line command at SID's pound '#' sign command prompt:

```
#IB:=CPM3;CPM3
```

Now, if you were to dump the buffer beginning with 0080, using the command: #D0080, you would see:

```
0080 0C 42 3A 3D 43 50 4D 33 3B .. ..
```

where the first byte, 0C, is the total number of characters in the command line parameter string.

For other ASSIGN parameters, such as:

```
#I? for partition info, would give us:
0080 01 3F 00 .. ..
```

```
#I{space} for assignments, would give us:
0080 01 20 00 .. ..
```

In each case, the first byte is the total number of characters used in the buffer.

Further note of interest about the DMA buffer at 0080: For some unknown reason CPM2 and CPM3 take the last 128 bytes of the program, up to the next page boundary (xx00 or xx80), and places them at 0080 before starting the program. Why? I don't know. May be a crude attempt at clearing this buffer area. Should I leave my own 128 byte buffer of zeros at the end of the program to ensure we start with a clear buffer? More on this later.

----- SWAPPING CPU PROCESSORS -----

While early CP/M used only an 8-bit CPU processor, such as the Intel 8085, with the development of the H/Z-100, later versions of CP/M, such as CP/M-85 (CP/M v2), sought to make maximum use of the Z-100's 16-bit Intel 8088. To this end, most of the BIOS functions mentioned above were executed by swapping the Z-100's CPU processors to accomplish as much of the desired task as possible.

So once the function number was given, CP/M used the Z-100's OFEh swap port to change the processor doing the work. The 16-bit portion of the BIOS contained a separate list of functions for the 8088 that were then called to perform the function.

----- BIOS DISK DATA STRUCTURE -----

Finally, please read the section titled "BIOS Disk Data Structure". The BIOS Disk Data Structures describe the particular characteristics of the disk subsystems used in CP/M.

ASSIGN deals specifically with assigning drive letters to the Winchester hard drive partitions, so these tables will come in handy for that.

Some other equates might be handy to place at the start of the code, such as these non-graphic characters:

```
BEL      equ 07h ;Bell
CR       equ 0Dh ;Carriage Return
LF       equ 0Ah ;Line Feed
```

Following all these EQUates, we have the starting point of the code:

```
ORG TPA      ; Begin TPA at 0100
Start:
LXI SP,0200h ;Create Local Stack
JMP Begin    ;Jump over data and
              ; any stack area
DS 0FAh      ;Reserve 250 bytes for
              ; any data & Stack
```

Adjust 0FAh to make Begin start at 0200.

Generally, here you may find some data area that may contain one or more buffers, data strings, and perhaps the local stack. The JMP statement above jumps this area to go to the actual start of the code.

Use SID's 'D'ump command to print this area, and 'L'ist to print the actual source code statements. ASSIGN had no such text here; it lists these statements at the end.

It did, however, have data or code kept in this location, though I have not been able to determine where it came from. When I replaced this area with all zeros, however, there was NO influence on the running of the program. More on this later.

I usually add some sensible labels and place their initial memory locations towards the end of the line to search for these later and update the labels in the code. If desired, the buffer area can be further broken down to individual memory locations as we figure each of them out. But for now, we'll leave them lumped together as defined space, DS.

7) Studying the printout, look for each CALL and note each new routine. Note at the start of each, the Address Location it was called from, so later we can go back and relabel each CALL, if we so desired.

8) Look at each Jump instruction, and note at each destination, the Address Location it was jumping from, such as:

```
JMP from 0310, or JNC from 02FF
```

What we are doing is identifying each new chunk of code, breaking it down into manageable portions and giving it some structure.

9) Regarding the data areas before, and probably others in the program and after the last coded statement, these are often accessed by the program to store and retrieve data. While the original source code had labels for all these locations, at this point it is nearly impossible to relabel each of these to something meaningful. It is best to leave these memory locations as just memory addresses and as you figure out a function for any particular location, note it in the code or printout.

10) Using your favorite word-processing utility, create a file called DRAFT.ASM and type in whatever useful text from the above explanation of the disassembly process that you wish. Begin all lines with a semicolon so as not to generate errors during reassembly. Include the information on BDOS functions and BIOS entry points. The idea here is that we create a basic set of instructions that can be used in future disassembly projects and insert it at the start of every project. If you received disks from me, this DRAFT document has already been generated. Tailor it to your own tastes, however.

11) Boot to the CPM3 partition and copy CPM2's ASSIGN.COM to it as ASSIGN2.COM. Also copy DRAFT.ASM to it. Now, it's time to create our new source code. I found an interesting use for CP/M-Plus' PUT utility that can save us a considerable amount of typing.

The PUT utility is able to redirect printer output to a filename. So we can use PUT to send our listing from SID to a file, TEST.ASM, by using the following command sequence:

PUT PRINTER FILE B:TEST.ASM

In English, this is translated as:

PUT PRINTER OUTPUT TO FILE B:TEST.ASM

Now press {CTRL}-{P} to send everything to the printer. Then invoke SID using the command:

SID B:ASSIGN2.COM

The SID 'L'ist command will now list our unassembled code to the printer and the file, B:TEST.ASM at the same time. If you do not want a printout, turn the printer off.

At the SID # sign prompt, use the 'L'ist {First Addr},{Last Addr} command where First Addr is 0100, and the Last Addr is the 'NEXT' or 'MSZE' given by SID when it loaded our ASSIGN2.COM program.

The command becomes:

#L0100,0880

12) When SID has completed listing the program, exit with a {CTRL}-{C} to the A> system prompt. Now we can use the PIP utility to create an .ASM file, first loading our DRAFT.ASM file, then loading our B:TEST.ASM file, using the command:

PIP B:ASGN85.ASM=B:DRAFT.ASM,B:TEST.ASM

You can then edit your B:ASGN85.ASM using your favorite editor. I like Magic Wand on the Z-100. However, I found a PC works better. More on this later.

I chose the name ASGN85 because this will not be the actual source code for ASSIGN. CP/M v2 and v3 already had their own ASSIGN commands and I wanted to make some changes to each program, so I didn't want them to be confused with the official ASSIGN command of either version.

13) You will find that B:ASGN85.ASM consists of all the text from DRAFT.ASM followed by the opening title of SID:

```
CP/M 3 SID - Version 3.0
NEXT MSZE PC END
0880 0880 0100 D1FF
#L0100,0880
```

14) Place semicolons in front of these SID lines for now. They will remind you of these procedures to generate this .ASM file.

However, if you get tired of looking at them, this can all be deleted. Likewise, the data between 0100 and 0200 (our stack area) can be left for now, until we see if any of it is used, at least. If and when we ever decide to try reassembling this file, nearly all the preceding text can be deleted, if desired. If left in, make sure the informational text is at least commented out with semicolons.

Now we come to the real coded statements:

```
0100 LXI SP,0200h ;(0100)
0103 JMP Begin ;(0103) Jump to 0200
0106 INR C ;(0106)
0107 XRA D ;(0107)
0108 INR C ;(0108)
0109 ORA E ;(0109)
etc... to 0200
```

For now, leave these statements as they are until we find out if they are actually used or if they are just part of a data area. After defining everything else in the code, we will look back on this area from 0106 to 01FF and see if it was used for anything or needs to be deleted.

My testing has shown the above data lines between 0106 and 0200 have never been called nor jumped to and all CALL and JMP instructions go to no legitimate locations. Yet, the original ASSIGN had code placed here by CP/M

(from where and why?) and my program insists on leaving zeros here because of the 'JMP Begin' instruction.

I suspect that ASSIGN had a DS statement added as I've done above that caused Begin to actually start at 0200 exactly. The DS statement only saves the space, it sets no particular data in this space like a DB or DW statement does, so anything may be found here.

Additional testing has shown that replacing this strange data here with zeros has no affect on ASSIGN's operation. So I've subsequently eliminated this area from ASGN85.ASM.

Note: Our ASGN85 program, when completed, can be assembled under CPM2 or CPM3. CPM3 can use MAC (similar to CPM2's ASM utility) to generate a .HEX file which is then converted to a .COM file by the LOAD utility, or RMAC (which generates a .REL file which is then LINKed to become a .COM file).

However, if you use RMAC, comment out the ORG opcode. RMAC assumes the new .COM file is going to begin at 0100 and if the ORG statement is left in, RMAC will place a new JMP 0203 instruction at address 0100 and a gap of 256 zero bytes will be added to the program! Removal of the ORG 0100 statement fixes this.

We cannot leave the stack at 0200 either. We could either change it to 0100 and let it use the Page Zero buffer area or move it elsewhere. As I was already making changes to hopefully improve the program, I moved the stack to follow the data area at the end of the program, therefore changing the above opening lines to simply (I left the old statements at the start of the program listing so you can readily see the difference):

```
      ORG   TPA           ; Begin TPA at 0100
Start:
      LXI   SP,Stack      ;See text re Stack
```

15) In this disassembled code, transfer all our notes from the previous SID ASSIGN2.COM printout, documenting the Jumps and Call addresses, BDOS functions, and the like.

CAUTION: Often the code can get scrambled when going from a text area, buffer area, or from an area reserved for the stack, back into code. The disassembler gets confused. Not knowing where the code begins, it takes its best guess. For example, the following code was disassembled as:

```
01FF  CPI   2Ah
0201  MOV   C,M
0202  NOP
0203  LXI   B,0048h
0206  DAD   B
```

One good give away is the NOP opcode. If the disassembler can't find a legal opcode, it

inserts a NOP or DB to realign itself with meaningful opcodes. If you know the end of one of these data areas, begin disassembly where you know code begins again. For example, the actual disassembled code, if we begin at 0200 (top of stack and start of new code) is:

```
0200  LHLD  BIOSf           ;(004E)
0203  LXI   B,0048h
0206  DAD   B
```

16) Now is a good time to use SID to trace through and print out a good portion of the ASSIGN.COM program. As we want to see the ASSIGN ? routine, boot to CPM2 (remember this version of ASSIGN will NOT work under CPM3) and press {CTRL}-{P} at the CP/M prompt to enable printing, and use the command:

A>SID ASSIGN.COM

The computer will display something similar to:

```
CP/M 3 SID - Version 3.0
NEXT MSZE PC END
0880 0880 0100 CDFE
#
```

At the pound prompt, enter 'I?':

#I?

As we mentioned before, the 'I'nput Line command simulates the command line that the CCP normally prepares upon program load. SID does not recognize any characters following the program name, so the 'I'nput command permits us to finish the command by adding any additional required characters.

17) It is time to remove all the addresses listed before the opcodes. These addresses will confuse any attempt at reassembly. I like to list them as remarks for easy reference. Convert those addresses that we know to be used into labels:

For example, for all the addresses that were targets for Jumps, I created a label beginning with Jxxxx, where xxxx is the original address listed. Likewise, all addresses that were targets for Calls, now became Cxxxx. And, finally, all those addresses that were referenced somehow, but not actually used, became Rxxxx. Later, we can replace these labels with more descriptive labels, if you so desire.

18) Now is also a good time to transfer what we have to an IBM-PC clone. No offense meant to the Z-100 or CP/M, but working on the much more capable and faster PC, using a newer editor, even WORDPAD, and having multiple screen windows is so much easier.

One means to accomplish this is to download 22DISK from off the internet. This utility

works under DOS and permits the transfer of data to and from CP/M floppy disks in a format usable on the Z-100.

I work on the files using WORDPAD, make copies for archive purposes, and even started a CP/M Library, but I can address this further in a separate article.

Briefly, I take the CP/M 5/25" floppy disk to my PC, run 22DISK and copy the files to E:\CPMwork\ (filename). E: is a partition on the hard drive or can be a separate hard drive.

After making any changes, I copy the files back to the CP/M floppy for transfer back to the Z-100 for assembly under CP/M and run the programs.

Disassembly of the CP/M-85 ASSIGN.COM Utility

```

=====
ORG     TPA                ; Begin TPA at 0100
Start:
    LXI  SP,Stack          ;See text regarding Stack
;    LXI  SP,0200h          ; (0100) Stack here
;    JMP  Begin             ; (0103),0200
;    DS   0FAh              ;Reserve 250 bytes to
;                               ; make Begin=0200
;Begin:                     ;(0200) We jump here from Start,
;                               ; over any data area & Stack
=====

```

I've added some new code to check the correct version using the BDOS version function and provided some help in the form of an opening screen.

```

    MVI  C,VerNbr           ;Let's check the BIOS version is OK
    CALL BDOS               ;Returns ver# in A and HL
    CPI  20h                ;Version 2.0 or better?
    JC   ErrVers            ; (049F) Version bad
    CPI  30h                ;Version < 3.x?
    JNC  ErrVers            ; (049F) Version bad

    LXI  D,HdgMsg           ;Print an opening message
    CALL PString

;The original code checked for the correct version differently:
;    LHL  BIOSf              ; (0200) Load L,H w/contents of 004E, 004F
;    LXI  B,0048h            ; (0203) HL=1774, BC=0048
;    DAD  B                  ; (0206) HL=HL+BC
;    MOV  A,M                ; (0207) A=67h there
;    CPI  67h                ; (0208)
;    JNZ  ErrVers            ; (020A),049F Jump if not 67h

    LXI  H,DMAaddr          ; (020D),0080 Default DMA address
    MOV  A,M                ; (0210) A = # of chars of CmdLine
;                               ; See DMAaddr notes above for more info
    STA  CmdLnNbr           ; (0211),0828 Save CmdLine char count
    INX  H                  ; (0214)
    SHLD CmdLnPtr           ; (0215),0829 Save CmdLine char addr ptr
    CALL C05E4              ; (0218) Check chars on command line
    LDA  CmdLnNbr           ; (021B),0828 Get CmdLine char count
    ORA  A                  ; (021E)
    JZ   J037D              ; (021F) Jump if zero
    CALL C05B0              ; (0222)
    CPI  3Fh                ; (0225) Is char a '?'?
    JZ   Help               ; Display help message
    CPI  2Ah                ; Is char a '*'?
    JZ   J0434              ; (0227) Get partition info
    CPI  41h                ; (022A) Is char < 'A'?
    JC   ErrDrv             ; (022C),0481 Bad drive name msg
;                               ; Should we convert to upper case????
;    ANI  5Fh                ; (022F) Is char > '['?
;    CPI  5Bh                ; (022F) Is char > 'P', the last legal drive ltr
;    CPI  50h                ; (0231),0481 Bad drive name msg
;    JNC  ErrDrv             ; (0234) Subtract 41h
;    SUI  41h                ; (0236),0807 Save the drive number (unit 0-16)
;    STA  DrvNbr             ; (0239) Get second CmdLine char
;    CALL C05B0              ; (023C) Is char a colon?
    CPI  3Ah                ; ':'

```



```

        JNZ  ErrDrv      ;(023E),0481 No? Bad drive name msg
        LDA  DrvNbr      ;(0241),0807 Load the drive number (0-16)
L0244:
        CALL C05C0       ;(0244) Select DskDrv BIOSfunc
        JZ   ErrDrv      ;(0247),0481 Returns 0 if no drive
        SHLD DPHaddr     ;(024A),L082B Save HL = Disk Parm Hdr (DPH)
        LXI  B,DPHoset   ;(024D),0010h
        DAD  B           ;(0250) HL=HL+BC = DPH+10h
        MOV  A,M         ;(0251)
        ANI  0E0h        ;(0252) 0E0h = 1110 0000b Isolate drive bits
        CPI  40h         ;(0254) Z217 (0100 0000) bit set?
        JNZ  ErrDrv      ;(0256),0481 Bad drive name msg, if not 40h
        LHLD DPHaddr     ;(0259),082B Set HL = Disk Parm Hdr (DPH)
        LXI  B,DPHoset   ;(025C),0010h
        DAD  B           ;(025F)
        MOV  A,M         ;(0260)
        ANI  0F7h        ;(0261) 0F7h = 1111 0111b Remove ASGNmt bit
        MOV  M,A         ;(0263) and save remainder
        CALL C05E4       ;(0264)
        CALL C05B0       ;(0267)
        ANA  A           ;(026A)
        JZ   RetOS       ;(026B),047E ;Return to OS
        CPI  3Dh         ;(026E)
        JNZ  ErrPart     ;(0270),048D Bad partition name
        CALL C05E4       ;(0273)
        LDA  CmdLnNbr     ;(0276),0828 ;Get CmdLine char count
        ANA  A           ;(0279) A = 0?
        JZ   ErrPart     ;(027A),048D Bad partition name

```

;You get the idea. Let's hit the highlights of a few other sections:

```

J030B:      ;Jump from 0300
        LXI  D,001Bh     ;(030B) Add 001Bh offset
        DAD  D           ;(030E) HL=HL+DE
        PUSH H           ;(030F)
        CALL C05D0       ;(0310) Save M to A & HL
        SHLD L0822       ;(0313) Save SetDsk Result: DPHaddr or 0000h
        POP  H           ;(0316)
        LXI  D,001Eh     ;(0317) Add 001Eh offset
        DAD  D           ;(031A)
        CALL C05D0       ;(031B) Save M to A & HL
        DCX  H           ;(031E)
        SHLD L0824       ;(031F) Save addr
        XRA  A           ;(0322) Zero A & clear flags
        STA  L082F       ;(0323) Set HDrive Unit to zero

```

```

J0326:      ;Jump from 035A
        LDA  L082F       ;(0326) Get HDrive Unit
        CALL C05C0       ;(0329) Select DskDrv BIOSfunc
        JZ   J035D       ;(032C) Returns 0 if no drive
        SHLD L082D       ;(032F) Save DMA Buffer Pointer
        LXI  D,DPHoset   ;(0332),0010h
        DAD  D           ;(0335) HL=HL+DE
        MOV  A,M         ;(0336)
        ANI  0E8h        ;(0337) 0E8h = 1110 1000b
        CPI  48h         ;(0339) 0100 1000b Is Z217 and assigned?
        JNZ  J034F       ;(033B) No, jump
        LXI  D,0004h     ;(033E) Yes, Add 4 to HL
        DAD  D           ;(0341) HL=HL+DE
        CALL C05D0       ;(0342) Save M to A & HL
        XCHG             ;(0345) HL<->DE
        LHLD L0822       ;(0346) Load SetDsk Result: DPHaddr or 0000h
        CALL C05A8       ;(0349) Compare HL to DE
        JZ   ErrPartU    ;(034C),0487 Partition already in use

```

```

J034F:      ;Drive not Z217 or assigned
            ;Jump from 033B
        LHLD L082D       ;(034F) Get DMA Buffer Pointer
        LXI  D,0018h     ;(0352) Jump 24 bytes to next DPH +offset

```



```

DAD D ;(0355)
LXI H,L082F ;(0356) Get HDrive Unit addr
INR M ;(0359) Inc HDrive Unit
JMP J0326 ;(035A) Do again

J035D: ;Jmp from 032C ;No drive found

J037D: ;Jmp from 021F, No chars on CmdLine
; MVI C,09 ;(037D)
LXI D,CRLF1 ;(037F),0791 Print CR,LF
CALL Pstring ;(0382)
XRA A ;(0385) Zero A & clear flags
STA L0834 ;(0386) Save Partition unit#
STA L0835 ;(0389) Save Copy2 Partition unit#
J038C: ;Jmp from 0431
LDA L0834 ;(038C) Load Partition unit#
CALL C05C0 ;(038F) Select DskDrv BIOSfunc
JZ RetOS ;(0392),047E Returns 0 if no drive - reboot
SHLD DPHaddr ;(0395),082B Save HL = Disk Parm Hdr (DPH)
LXI D,DPHoset ;(0398),0010h
DAD D ;(039B) HL=HL+DE = DPH+10h = F8B5
MOV A,M ;(039C) Get data there
ANI 0E8h ;(039D) 0E8h = 1110 1000b
CPI 48h ;'H' ;(039F) 'H' = 0100 1000b?
JNZ J042D ;(03A1) No, jmp to increment data in 0834
LXI H,L0835 ;(03A4) Set Copy2 Partition unit# addr
MOV A,M ;(03A7) Get data there
ANA A ;(03A8)
JNZ J03B3 ;(03A9)
DCR M ;(03AC) Decrement data in HL
CALL C04ED ;(03AD)
JC ErrRd ;(03B0),0499 Disk Read error
J03B3: ;Jmp from 03A9
LHLD DPHaddr ;(03B3),082B Get HL = Disk Parm Hdr (DPH)
LXI D,0014h ;(03B6)
DAD D ;(03B9)
CALL C05D0 ;(03BA) Save M to A & HL
SHLD L0822 ;(03BD) Save SetDsk Result: DPHaddr or 0000h
LXI H,DMABfr ;(03C0),083C 128-byte Local DMA Buffer
J03C3: ;Jmp from 03E0

J03E3: ;Jmp from 03D9
SHLD L0836 ;(03E3) Save H & L from SetTrk BIOSf
LDA L0834 ;(03E6) Load Partition unit#
ADI 41h ;(03E9) Make A=drive letter
CALL C05D5 ;(03EB) Display drive letter ConOut BDOS func
; MVI C,09 ;(03EE)
LXI D,Colon ;(03F0),0794 ;Print ': = '$
CALL Pstring ;(03F3)
LHLD L0836 ;(03F6) Load H & L from SetTrk BIOSf
MVI C,10h ;(03F9)
J03FB: ;Jmp from 0406

J042D: ;Jmp from 03A1,03C6
LXI H,L0834 ;(042D) Set Partition unit# addr
INR M ;(0430) Inc Drive unit#
JMP J038C ;(0431) Do again for next partition

J0434: ;Jmp from 0227
XRA A ;(0434) Zero A & clear flags
STA L0834 ;(0435) Save Partition unit#

J0438: ;Jmp from 0454
LDA L0834 ;(0438) Load Partition unit#
CALL C05C0 ;(043B) Select DskDrv BIOSfunc
JZ RetOS ;(043E),047E Returns 0 if no drive - reboot
SHLD DPHaddr ;(0441),082B Save HL = Disk Parm Hdr (DPH)
LXI D,DPHoset ;(0444),0010h DPH Offset
DAD D ;(0447)

```



```

MOV    A,M                ;(0448) Get Partition info
ANI    0E1h               ;(0449) 0E1h = 1110 0001b Z217 & Pri DPE?
CPI    41h ; 'A'          ;(044B) 0100 0001b?
JZ     J0457              ;(044D) Yes, jump
LXI    H,L0834            ;(0450) Set Partition unit# addr
INR    M                  ;(0453) Increment Drive unit#
JMP    J0438              ;(0454) Try next partition

J0457:                    ;Jmp from 044D
CALL   C04ED              ;(0457) Read Disk
JC     ErrRd              ;(045A),0499 Disk Read error
; MVI    C,09              ;(045D)
LXI    D,PartHdg          ;(045F),0799 Print partition display hdg
CALL   Pstring            ;(0462)
LXI    H,DMABfr           ;(0465),083C 128-byte Local DMA Buffer
J0468:                    ;Jmp from 047B
SHLD   L082D              ;(0468) Save DMA Buffer Pointer
MOV    A,M                ;(046B)
CPI    20h ; ' '          ;(046C) Space?
JZ     RetOS              ;(046E),047E ;Return to OS
CALL   C0602              ;(0471)
LXI    D,001Eh            ;(0474) Add 001Eh offset
LHLD   L082D              ;(0477) Get DMA Buffer Pointer
DAD    D                  ;(047A)
JMP    J0468              ;(047B)

Help:                    ;Display help screen
LXI    D,HelpMsg          ;Print help message
CALL   PString
RetOS:                    ;Jmp from 026B,037A,0392,043E,046E,04B9
JMP    0000               ;(047E) Return to A>
;Error messages are all similar:

BCRLF:                    ;Jmp from 0484,048A,0490,0496,049C
PUSH   D                  ;(04A2)
LXI    D,Bell             ;(04A5),078C Bell & CRLF msg
CALL   Pstring            ;(04A8)
POP    D                  ;(04AD)
CALL   Pstring            ;(04AE) Display any of above msgs after bell
LXI    D,CRLF1            ;(04B3),0791 Display CRLF msg
CALL   Pstring            ;(04B6)
JMP    RetOS              ;(04B9),047E ;Return to OS

C04BC:                    ;Call from 050B,0571
PUSH   H                  ;(04BC) A=48h; HL=Addr of 128-byte DMA Buffer
PUSH   D                  ;(04BD) BC=DE=0000
; LHLD   BIOSf            ;(04BE),004E
LXI    D,SetTrk           ;(04C1),001E Set Track BIOS function
; DAD    D                ;(04C4)
CALL   BIOFunc            ;(04C5),05AE Execute BIOS function
POP    B                  ;(04C8) From PUSH D at 04BD
INX    B                  ;(04C9)
; LHLD   BIOSf            ;(04CA),004E
LXI    D,SetSec           ;(04CD),0021 Set Sector BIOS function
; DAD    D                ;(04D0) HL=HL+DE
CALL   BIOFunc            ;(04D1),05AE
POP    B                  ;(04D4) From PUSH H at 04BC
; LHLD   BIOSf            ;(04D5),004E
LXI    D,SetDMA           ;(04D8),0024 Set DMA addr BIOS function
; DAD    D                ;(04DB) HL=HL+DE
CALL   BIOFunc            ;(04DC),05AE
; LHLD   BIOSf            ;(04DF),004E
LXI    D,RdSec            ;(04E2),0027 Read Sector BIOS function
; DAD    D                ;(04E5) HL=HL+DE
CALL   BIOFunc            ;(04E6),05AE A=00, BC=09F8, DE=0027, HL=F724
ORA    A                  ;(04E9)
RZ                      ;(04EA) Rets to 050E
STC                      ;(04EB) Set Carry Flag
RET                      ;(04EC)

```



```

C04ED:                ;Call from 02C9,03AD,0457
;   LHL D BIOSf        ;(04ED),004E
;   LXI D,ClrBfr       ;(04F0),003F Clear Buffer BIOS func
;   DAD D              ;(04F3)
;   CALL BIOFunc       ;(04F4),05AE
;   LHL DPHaddr        ;(04F7),082B Get HL = Disk Parm Hdr (DPH)
;   LXI D,DPHoset      ;(04FA),0010h HL=F8A5
;   DAD D              ;(04FD) HL=HL+DE = F8B5

DoFunc:              ;Do BIOS function
;Call from 04C5,04D1,04DC,04E6,04F4,05A3,05CA
;   PCHL              ;(05AE) Put HL into PC
;   RET              ;(05AF)

C05B0:              ;Call from 0222,0239,0267,0282,0297,02B2,02C2
;   LXI H,CmdLnNbr     ;(05B0),0828 Load CmdLine char count addr
;   MOV A,M            ;(05B3) Get content from HL=0828
;   ANA A              ;(05B4) Is A = 0?
;   RZ                ;(05B5)
;   DCR M              ;(05B6) The content at HL=0828 is decremented
;   LHL CmdLnPtr       ;(05B7),0829 Get HL = CmdLine char addr ptr
;   MOV A,M            ;(05BA)
;   INX H              ;(05BB) HL is incremented and
;   SHLD CmdLnPtr      ;(05BC),0829 saved in CmdLine char addr ptr
;   RET              ;(05BF)

C05C0:              ;Select disk BIOS function
;Call from 0244,0329,038F,043B
;   MOV C,A            ;(05C0) C=Drive 0-15
;   LHL BIOSf          ;(05C1),004E
;   LXI D,SetDsk       ;(05C4),001B BIOS func#9 - Select Disk Drive
;   DAD D              ;(05C7) HL=HL+DE
;   MVI E,01           ;(05C8) E=Initial Select Flag???
;   CALL DoFunc        ;(05CA),05AE HL is moved into PC
;If drive exists, HL = Disk Parm Header(DPH) addr, (=B73E for A:)
;If drive does NOT exist, HL = 0000h
;   MOV A,H            ;(05CD)
;   ORA L              ;(05CE) A=valid# or 00h if failure
;   RET              ;(05CF)

C05D0:              ;Save M to A & HL
;Call from 0310,031B,0342,03BA,03CE,0518,
;   0522,052C,0609,0614

C05D5:              ;Display A=Char ConOut BDOS function
;Call from 03EB,0401,040B,041D,0645,0664

C05E4:              ;Check for Command Line chars
;Call from 0218,0264,0273,0294,02A3,02BF

L0685:              ;SpecCharString
;Ref: 062F,0639,0676
;   DB 22h,2Dh,08h,21h,2Fh,24h ; (0685)
;Represents: " , - , BS, EP, / , $
; where BS=Back Space and EP=Exclamation Point

C068B:              ;Check SpecCharString for 'zero' & make 'space'
;Call from 063C; Jmp from 0693
;Entry: HL = SpecCharString addr
;Exit: If any char of SpecCharString <> 30h, return
;      If any char of SpecCharString = 30h, make it a space,
;      and try next char.
;   MVI A,30h;'0'      ;(068B) A = zero
;   CMP M              ;(068D) SpecChar = zero?
;   RNZ                ;(068E) No, return
;   MVI A,20h;' '      ;(068F) Yes, make it a space and
;   MOV M,A            ;(0691) save it in the string
;   INX H              ;(0692) Try next char

```



```

        JMP      C068B                ;(0693) Loop

C06B8:                ;Complement DE, Divide BC by subtracting DE
                        ;Call from 0540,0556,0625,062A,069F
;Entry:    DE = Divisor, BC = HL = Number to divide
;Exit:     HL = Number result

C06FB:                ;Call from 055F
;Entry:    A  = Hdrive unit# (0-3)
;          BC = Bytes from DMA+26h
;          DE = Bytes from DMA+2Eh
;Exit:     BC = ???
;          HL = ???

;BIOFunc   -- This procedure executes a BIOS function
;Entry:    [DE] = BIOS function number
;          [Other] = Entry variable as required by function
BIOFunc:                ;(05AE) Do BIOS function
        LHL D    BIOSf        ;Set BIOS base
        DAD D
        CALL DoFunc        ;Execute BIOS function
        RET

;Note:  The following data areas are completely changed.
;-----
;DATA & ALTERNATE STACK AREAS
HdgMsg:
        DB 1Bh,45h,CR,LF,'
        DB 1Bh,70h,'  ASGN85 for CP/M Version 2.0  ',1Bh,71h,CR,LF
        DB '      Copyright 1982, Digital Research',CR,LF
        DB 'Modified ASSIGN.COM by Steven W. Vagts, Z-100 LifeLine, '
        DB '10/2012',CR,LF,'$'
HelpMsg:
        DB CR,LF,'Limitations:'
        DB CR,LF,'  - Only two partitions can be assigned at any time.'
        DB CR,LF,'  - Drive letters are restricted according to drive '
        DB 'configuration:'
        DB CR,LF,'      Boot Device:   Other Drives:   Assign Letters:'
        DB CR,LF,'      Hard Drive    - No matter -      A & B'
        DB CR,LF,'      5" floppy     Has 8" drives   E & F'
        DB CR,LF,'      5" floppy     w/o 8" drives  C & D'
        DB CR,LF,'      8" floppy     Has 5" drives   E & F'
        DB CR,LF,'      8" floppy     w/o 5" drives  C & D'
        DB CR,LF,LF,1Bh,70h,'ASGN85 ?',1Bh,71h,' will display this help '
        DB 'screen, then return to the CP/M prompt.'
        DB CR,LF,LF,1Bh,70h,'ASGN85 *',1Bh,71h,' will display a list of '
        DB 'the partition names on the hard drive.'
        DB CR,LF,LF,1Bh,70h,'ASGN85 x:={PartName}',1Bh,71h,' will assign '
        DB 'a drive letter (A - F) to the desired',CR,LF
        DB '      {Partition Name}.'
        DB CR,LF,LF,1Bh,70h,'ASGN85 (with no parameter)',1Bh,71h
        DB ' will list the assignments already provided.',CR,LF,'$'
BadDrv   DB 'Invalid Drive Letter$'                ;(071B),0481
BadPart  DB 'Bad Partition Name$'                  ;(072A),048D
BadOS    DB 'Bad OS Name$'                          ;(073D),0493
PartUse  DB 'Partition already in use$'             ;(0749),0487
BadVer   DB 'Sorry, you need CP/M v2.x$'            ;(0762),049F
RdErr    DB 'Disk Read Error$'                      ;(077C),0499
Bell     DB BEL,CR,LF,'$'                          ;(078C),04A5
Ltrk     DB 6Bh                                     ;(0790),067E
CRLF1    DB CR,LF,'$'                               ;(0791),037F,0427,04B3
Colon    DB ': = $'                                 ;(0794),03F0
PartHdg  DB CR,LF,'                                ;(0799),045F
        DB CR,LF,' PARTITION NAME      OS NAME      SIZE' ;(07B3)
        DB CR,LF,'-----' '-----' '-----',CR,LF,'$' ;(07D9)
Spc2     DB ' $'                                     ;(0804),064F,066C
DrvNbr   DB 0Ah                                     ;(0807),0236,0241
Spc22    DB ' '                                     ;(0808),027F,02DB
Spc4     DB ' ' ;(0818),02AF,02E9

```



```

L0822      DB  00h,00h      ;Store SetDsk Result: DPHAddr or 0000h
                        ;(0822),0313,0346,036D,03BD,03D2
L0824      DB  00h,00h      ;Store SetTrk addr
                        ;(0824),031F
L0826      DB  00h,00h      ;Store H & L bytes from DMA+26h
                        ;(0826),052F,054D
CmdLnNbr   DB  00h          ;Number of chars on the command line.
                        ;(0828),0211,021B,0276,02A6,05B0,05E4,05FA
CmdLnPtr   DB  00h,00h      ;Command Line Address Pointer
                        ;(0829),0215,05B7,05BC,05EA,05FB
DPHAddr    DB  00h,00h      ;Disk Parameter Header address
                        ;(082B),024A,0259,035D,0395,03B3,0441,04F7,0591
L082D      DB  00h,00h      ;Store DMA Buffer Pointer
                        ;(082D),032F,034F,0468,0477,0535,056E,057E
                        ;0585,0602,060D,063F,065A,0685
L082F      DB  00h          ;Hard Drive unit number
                        ;(082F),0323,0326,0356,054A,055C,0588
L0830      DB  00h,00h      ;Store (DMA+26h)/(DMA+2E) result
                        ;(0830),0559,0565
L0832      DB  00h,00h      ;Store H & L from C06FB call
                        ;(0832),0562,056A,0577,057B
L0834      DB  00h          ;Store Partition Letter as a unit#
                        ;(0834),0386,038C,03E6,042D,0435,0438,0450
L0835      DB  00h          ;Store Copy2 Partition unit#
                        ;(0835),0389,03A4
L0836      DB  00h,00h      ;Store H & L from SetTrk BIOS function
                        ;(0836),03E3,03F6,040E
L0838      DB  00h,00h      ;Store H & L bytes from DMA+2Ch
                        ;(0838),051B,0538,0621
L083A      DB  00h,00h      ;Store H & L bytes from DMA+2Eh
                        ;(083A),0525,0552

```

;This is an alternate location for our local stack. Due to the way that
;the different versions of CP/M treat the stack during assembly, I've
;moved the stack to the end of the program. However, it must be placed
;before the local DMA Buffer, which varies in size and may overwrite the
;stack if the stack is placed last. So the stack is created as follows:

```

          DS  28h          ;Reserve space for Stack
                        ;Size can be adjusted to simplify buffer addr
Stack     DS  0001h        ;Top of Stack is here

```

```

DMABfr:   DS  10h          ;Actually a 128-byte Local DMA Buffer
                        ;(083C),02CF,03C0,0465,0508,0511,0532

```

;This buffer takes some explanation. The last address used in ASSIGN
;was 083C and what appeared to be just 2 bytes long was a replacement
;128-byte local DMA buffer for use by the program. If my experimentation
;was correct, it was placed here such that the end went beyond the end of
;the actual ASSIGN program by about 60 bytes - a space-saving measure!

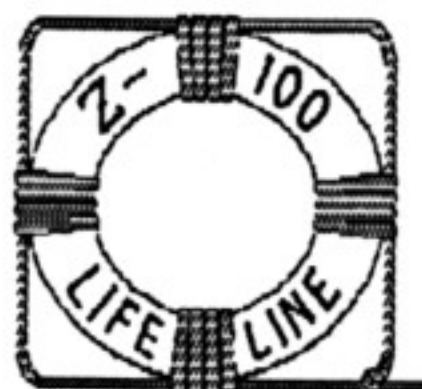
;Also note: CPM2 and CPM3 each load the default DMA buffer at 0080 with
;the last 128 bytes of the program from the end of a page boundary -
;(xx00 or xx80).

```

END      Start            ;End of program

```

;To assemble, rename ASGN85.DOC to ASGN85.ASM, eliminate any notes and
;documentation to reduce the size, if necessary, and use CP/M v2's ASM
;and LOAD utilities for all assembly work (see Notes below).





CP/M Assembly Language Programming Essentials

Assembly Language Programming requires access to a lot of material from various manuals and other sources that you may not have access to. So, I've collected much of this material and included the information here. I also provide the names of these source files as we use them and try to explain the workings of each section as we go.

----- Page Zero of Low Memory for CP/M-85 (CP/M v2) -----

Page Zero of low memory (addresses less than 0100) functions as an interface to the BDOS module from the Console Command Processor (CCP) and transient programs. It also contains critical system parameters. The contents of Page Zero are described briefly here.

For CP/M-Plus, also called CP/M v2, Page Zero has the following definitions:

- 0000 to 0002h contains a jump instruction to the BIOS warm start entry point. The address at 0001h can also be used to make direct BIOS calls to the BIOS console status, console input, console output, and list output primitive functions. CP/M2 has C3,03,F7.
- 0003h contains Intel's standard IOBYTE.
- 0004h contains the current default drive number (0=A,...,15=P).
- 0005h to 0007h contains a jump instruction to the BDOS and serves two purposes:
 - JMP 0005h provides the primary entry to the BDOS.
 - LHL 0006h places the address field of the jump instruction in HL.
 - CP/M2 has C3,00,CE
- 0008 to 0027h contains interrupt locations 1 through 5, not used.
- 0030 to 0037h contains interrupt location 6, not used.
- 0038 to 003Ah contains a jump instruction into the DDT or SID program when running in debug mode.
- 003B to 003Fh are not currently used - reserved.
- 0040 to 004Fh contain a 16-byte scratchpad area for CBIOS, but is not used.
- 0050 to 005Bh are not used.
- 005C to 007Ch contains the default File Control Block (FCB).
- 007D to 007Fh contains the optional default random record position.
- 0080 to 00FFh is the default 128 byte disk buffer; also filled with the command line when the CCP loads a transient program.

Source: For CP/M-85, see page 6.9 of the Digital Research CP/M Operating System Manual, Chapter 6, Alteration for more information.

Note: The default DMA address for transient programs is 0080h. The CCP also initializes this area to contain the command tail of the command line. The first position contains the number of characters in this buffer area.

----- BDOS Function Calls for CP/M-85 (CP/M v2) -----

All versions of CP/M use BDOS Function Calls to perform simple repetitive functions. The BIOS program will define this BDOS function as:

BDOS EQU 0005h ;BDOS function call

BDOS Calling Convention:

CP/M-85 (v2) uses a standard convention for BDOS function calls.

Entry: [C] contains the BDOS func#
 [DE] contains a byte or word value or an info address.
 Returns: [A] contains single-byte values
 [HL] contains double-byte value or 0FFFFh on error.

The BDOS Functions Calls for CP/M-85 (v2) are:

Function Name:	Reg C= Value/Func#:	Input:	Output:	Description:
Boot	equ 0000h /0	C=00h	None	Sys Reboot/Reset
ConIn	equ 0001h /1	C=01h	A=ASCII chr	Console Input
ConOut	equ 0002h /2	E=ASCII Chr	None	Console Output
RdrIn	equ 0003h /3		A=ASCII chr	Reader Input
PnchOut	equ 0004h /4	E=ASCII Chr	None	Punch Output
LstOut	equ 0005h /5	E=ASCII Chr	None	ListDevice Out
ConIO	equ 0006h /6	E=0FFh (Input) E=0FEh (Stat) E=Char (Output)	A=Chr or 0 A=Stat or 0 Char to Console	Direct Cons I/O
GetIobyte	equ 0007h /7	None	A=Iobyte Value	Get I/O Byte
SetIobyte	equ 0008h /8	E=Iobyte	None	Set I/O Byte
Pstrng	equ 0009h /9	DE=BufAddr	None	Print String
RdConBuf	equ 000Ah /10	DE=BufR	ConChr in BufR	Read Cons Buffer
ConStat	equ 000Bh /11	None	A=00h/NonZero	Get Cons Status
VerNbr	equ 000Ch /12	None	HL & A=Ver#	Return Vers#
DiskReset	equ 000Dh /13	None	None	Reset Disk Sys
SelDisk	equ 000Eh /14	E=Disk#	None	Select Disk
OpenFile	equ 000Fh /15	DE=FCB Addr	A=DirCode/0FFh	Open File
ClosFile	equ 0010h /16	DE=FCB Addr	A=DirCode/0FFh	Close File
Search1	equ 0011h /17	DE=FCB Addr	A=DirCode	Search for First
SearchNxt	equ 0012h /18	None	A=DirCode	Search for Next
DelFile	equ 0013h /19	DE=FCB Addr	A=None	Delete File
ReadSeq	equ 0014h /20	DE=FCB Addr	A=ErrCode	Read Sequential
WriteSeq	equ 0015h /21	DE=FCB Addr	A=ErrCode	Write Sequential
MakeFile	equ 0016h /22	DE=FCB Addr	A=0FFh onErr	Make File
RenamFile	equ 0017h /23	DE=FCB Addr	A=0FFh onErr	Rename File
RetLogVec	equ 0018h /24	None	HL=LoginVec*	Ret Login Vec*
CurentDsk	equ 0019h /25	None	A=CurDsk#	Ret Current Disk
SetDMAAddr	equ 001Ah /26	DE=DMA Addr	None	Set DMA Address
GetAddr	equ 001Bh /27	None	HL=AlloAddr*	Get Alloc Addr*
WrtProDsk	equ 001Ch /28	None	None	Wrt Protect Disk
ROvector	equ 001Dh /29	None	HL=ROvecVal*	Get Rd/Only Vec*
FileAttr	equ 001Eh /30	DE=FCB Addr	A=None	Set File Attribs
DPBAddr	equ 001Fh /31	None	HL=DPB Addr	Get DPB Address
UsrCode	equ 0020h /32	E=0FFh for Get E=00-0Fh Set	A=User#	Get/Set UserCode
RdRandom	equ 0021h /33	DE=FCB Addr	A=ErrCode	Read Random
WrtRandm	equ 0022h /34	DE=FCB Addr	A=ErrCode	Write Random
CmpFilsz	equ 0023h /35	DE=FCB Addr	R0, R1, R2	Compute FilSize
SRndmRec	equ 0024h /36	DE=FCB Addr	R0, R1, R2	Set Random Rec
ResetDrv	equ 0025h /37	DE=DriveVector	A=00h	Reset Drive
AccessDrv	equ 0026h /38	(MP/M function not supported)		Access Drive
FreeDrv	equ 0027h /39	(MP/M function not supported)		Free Drive
WRndmRec	equ 0028h /40	DE=FCB Addr	A=ErrCode	WrtRndm w/0Fill

* Note that: A = L and B = H upon return.

----- BIOS Entry Points for CP/M-85 (CP/M v2) -----

To achieve device independence, CP/M is separated into three distinct modules:

BIOS - Basic I/O system, which is environment dependent

BDOS - Basic disk operating system, which is NOT dependent upon the hardware configuration.

CCP - The Console Command Processor, which uses the BDOS.

Only the BIOS is dependent upon particular hardware configurations.

All versions of CP/M use BIOS Entry Points to send program control to the individual BIOS routines. Entry to the BIOS is through a 'jump vector', one of a series of jump instructions. The BIOS routines may be empty for certain functions (i.e., they may contain a single RET operation) if they are not implemented, but the entries must be present in the jump vector to avoid changing addresses.

The BIOS program will define the Base Address of this BIOS jump table as:

BIOS equ 004Eh; Set 004Eh for CP/M2; 000Dh for CP/M-Plus

The Offset Value is added to the Base Address to jump to the desired BIOS routine.

BIOS Calling Conventions:

CP/M-85 (v2) uses a standard convention for BDOS function calls.

Entry: Varies by function

Returns: [A] contains single-byte values.

[HL] contains double-byte values; 0FFFFh on error.

The BIOS Functions for CP/M-85 (v2) are defined in the BIOS85.A86 file as:

Function Name:		Offset Value	Dec Func#:	Input:	Output:	Description:
CBootE	equ	0000h	/0	None	None	Boot
WBootE	equ	0003h	/1	None	None	Warm Boot
ConStE	equ	0006h	/2	None	A=0FFh if ChrRdy A=0 if no Chr	ConsInput Stat
ConInE	equ	0009h	/3	None	A=ConChr	Get ConsChar
ConOutE	equ	000Ch	/4	C=ConChr	None	OutChr to Cons
LstOutE	equ	000Fh	/5	C=Char	None	OutChr to LstDev
PunOutE	equ	0012h	/6	C=Char	None	OutChr to PncDev
RdrInE	equ	0015h	/7	None	A=Char	InChar frm RdDev
HomeE	equ	0018h	/8	None	None	SelTrk0 of Drv
SetDsKE	equ	001Bh	/9	C=DskDrv# (0-15) E=InitSelFlg	HL=DPHaddr HL=0000h if no drive	Select DskDrive
SetTrKE	equ	001Eh	/10	BC=Track#	None	Set Trk# (0=1st)
SetSecE	equ	0021h	/11	BC=Sector#	None	Set Sec# (1=1st)
SetDMAE	equ	0024h	/12	BC=DMAaddr	None	Set DMA address
ReadE	equ	0027h	/13	None	A=0 if succ A=1 if nonrecover error A=0FFh if media changed	Read Sector
WriteE	equ	002Ah	/14	C=DeBlkCode 0=NormWrt 1=WrtDir 2=1stWrtSec	A=0 if succ A=1 if physical error A=2 if disk R/O A=0FFh if media changed	Write Sector
LstStE	equ	002Dh	/15	None	A=0 if NotRdy A=0FFh if DevRdy	Get LstDev Stat
SecTrnE	equ	0030h	/16	BC=LogSec# DE=TranTblAddr	HL=PhySec#	Translate Sec#
FormateE	equ	0033h	/17	C=VerifyFlg 0=No, 1=Yes	A=Status Byte A=0FFh if Rdy	Format Disk
RdTrKE	equ	0036h	/18	None	A=0 if NotRdy A=0FFh if Rdy	Read Track
WrtTrKE	equ	0039h	/19	None	A=0 if NotRdy A=0FFh if Rdy	Write Track
WpCE	equ	003Ch	/20	None	A=0 if R/W A=1 if R/O	WrtProtect Check
ClrBufD	equ	003Fh	/21	None	None	Clear Drv Bufrs
PEEK	equ	0042h	/22	DE=Offset, HL=Seg	A=Value	PEEK 8088 Memory
POKE	equ	0045h	/23	C=Value DE=Offset, HL=Seg	None	POKE 8088 Memory

----- BIOS Disk Data Structure for CP/M-85 (v2) -----

The BIOS Disk Data Structures describe the particular characteristics of the disk subsystems used in CP/M. In general, each disk drive has an associated Disk Parameter Header (DPH) that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. One of the elements of this DPH is a pointer to a Disk Parameter Block (DPB), which contains the actual disk description.

Each DPH in generic CP/M v2 consisted of 16-byte entries, one for each drive that may be attached to the system. The Heath/Zenith version, CP/M-85 for the H/Z-100, expanded this to 24-bytes each, adding 8 bytes at the end for certain tasks, including the assignment of drive letters to hard drive partitions.

As described in BIOS85.ASM, the first two Disk Parameter Entry Tables were for physical Z207 5" floppy drives 0 and 1, both currently set for double density, 6 ms step rate. The second two tables were for physical Z207 8" floppy drives 2 and 3, both set for double density, with head load solenoids. The last two tables were for physical Z217 hard drive partitions, DPE2A and DPE2B, labeled Partitions 1 and 2.

Note: The variables given in the Tables are the default conditions. The definitions from the file BIOS85.ASM are listed below the tables for reference.

I've included the first Disk Parameter Entry Table of each type:

Physical Drive 0 -- Z207 5 1/4":

```
DPE1A    DS      0                ;Label to mark the table beginning
          DW      0,0,0,0          ;Scratchpad area
          DW      DIRBUF          ;Directory Buffer Address
          DW      DPB1A           ;Disk Parameter Block Address
          DW      CSV1A           ;Change Disk Software Address
          DW      ALV1A           ;Storage Allocation Address
          DB      DPEZ207+DPEDD    ;Z207, Double Density
          DB      0
          DB      0
          DB      0
          DB      DPEUNK           ;Track Position unknown
          DB      DPEMO+FDFS6     ;Motor-Up-2-Speed & Step Rate 6
          DB      0
          DB      0
```

Physical Drive 2 -- Z207 8":

```
DPE1C    DS      0                ;Label to mark the table beginning
          DW      XLATE1          ;Translate Table 1
          DW      0,0,0           ;Scratchpad
          DW      DIRBUF          ;Directory Buffer Address
          DW      DPB1C           ;Disk Parameter Block Address
          DW      CSV1C           ;Change Disk Software Address
          DW      ALV1C           ;Storage Allocation Address
          DB      DPEZ207+DPEDD    ;Z207, Double Density
          DB      CONPC+CONDS8+0   ;Set Precomp for 8" drives
          DB      0
          DB      0
          DB      DPEUNK           ;Track Position unknown
          DB      FDFS6           ;Step Rate 6
          DB      DPEHLS          ;Drive has Head Load Solenoid
          DB      0
```

Z217 Winchester Hard Drive -- Partition 1:

```
DPE2A    DS      0                ;Label to mark the table beginning
          DW      0,0,0,0          ;Scratchpad
          DW      DIRBUF          ;Directory Buffer Address
          DW      DPB2A           ;Disk Parameter Block Address
          DW      0               ;Change Disk Software Address
          DW      ALV2A           ;Storage Allocation Address
          DB      DPEZ217+DPEPRIM ;Z217 and Primary Unit
          DB      0
          DB      WIRPS           ;Defines 4 CP/M records/sector
          DB      0
          DB      0
          DB      0
          DB      0
          DB      0
```


DPE2B is similar, except it has its own DPB2B, the Disk Parameter Block, and just DPEZ217, rather than DPEZ217+DPEPRIM.

Disk Parameter Entry Description:

DPEXLT	RW	1	;Sector Translate Table Address
	RW	3	
DPEDIRB	RW	1	;Directory Buffer Address
DPEDPB	RW	1	;Disk Parameter Block Address
DPECSV	RW	1	;Checksum Vector Address
DPEALV	RW	1	;Allocation Vector Address
DPEHTH	RB	8	;Heath Extensions
DPEL	equ	24	;Length of Disk Parameter Entry

Note: The RB and RW codes above are for use with the A86 Assembler and are similar to the DS opcode, where space is reserved for a variable; RB saves 1 byte, RW saves 2 bytes. RB 8 is the same as DS 8 and RW 3 is the same as DS 6.

Heath Extensions:

DPEFLAG	equ	DPEHTH+0	;Flags
DPETYPE	equ	11100000b	;Bit 7-5 = Device Type
DPENE	equ	00000000b	; Non-existent
DPEZ207	equ	00100000b	; Z207
DPEZ217	equ	01000000b	; Z217
DPE48RO	equ	00010000b	;Bit 4 -- For Z207
			; 48TPI Media in 96TPI Drive (R/O)
DPE96T	equ	00001000b	;Bit 3 -- 0=48TPI Drive 1=96TPI Drive
DPEASGN	equ	00001000b	;Bit 3 -- For Z217 Winchester Disk
			; 0=Unassigned a Partition
			; 1=Assigned a Partition
DPET0SD	equ	00000100b	;Bit 2 -- 1=Track 0 is Single Density
DPEDD	equ	00000010b	;Bit 1 -- 0=Single Density, 1=Double
DPELSIO	equ	00000010b	;Bit 1 -- Z217 Logical Sector I/O
DPE2S	equ	00000001b	;Bit 0 -- 0=Single Sided, 1=Double
DPEPRIM	equ	00000001b	;BIT 0 -- Z217 Primary DPE for unit
DPEUNIT	equ	DPEHTH+1	;Unit Select Value
DPERPS	equ	DPEHTH+2	;CP/M Records per Physical Sector
DPERPAB	equ	DPEHTH+3	;CP/M Records per Allocation Block
DPETRK	equ	DPEHTH+4	;Track Counter
DPEUNK	equ	10000000b	;Track Position Unknown
DPELPB	equ	DPEHTH+4	;Z217 Lower Partition Boundary
			; (Logical Sector #)
DPESEK	equ	DPEHTH+5	;Motor Speed and Seek Speed
			;Bit 3-0 = Seek Speed Value
DPEFS	equ	01000000b	;Bit 6 = Fast Step for Z207
DPEMO	equ	10000000b	;Bit 7 = Motor Up to Speed Flag
			; 0=1 Sec, 1=250 MSec
DPEUPB	equ	DPEHTH+6	;Z217 Upper Partition Boundary+1
DPEFLG2	equ	DPEHTH+6	;2nd Flag Byte
DPEHLS	equ	00000100b	;Bit 2 = Drive has Head Load Solenoid
DPEIMG	equ	00000010b	;Bit 1 = Imaginary Drive
DPE96TM	equ	00000001b	;Bit 0 = 0=48TPI Media, 1=96TPI Media
DPELUN	equ	DPEHTH+7	;Last Logical Unit mounted
DPELOG	equ	11110000b	;CP/M Logical Drive Name for this entry
DPEREAL	equ	00001111b	;For Imaginary Drive, Logical Drive
			; Name for corresponding Real Drive
DPEMNT	equ	00001111b	;For Real Drive, Logical Drive Name
			; for Currently Mounted Disk
DPEHL	equ	8	;Length of Heath Extension

Disk Parameter Block:

DPBSPT	RW	1	;Sectors per Track
DPBBSH	RB	1	;Block Shift Factor
DPBBLM	RB	1	;Block Mask
DPBEXM	RB	1	;Extent Mask
DPBDSM	RW	1	;Total # of Blocks-1
DPBDRM	RW	1	;# of Directory Entries-1
DPBAL0	RB	1	;Initial AL0 Value
DPBAL1	RB	1	;Initial AL1 Value
DPBCKS	RW	1	;Size of Directory Check Vector


```

DPBOFF    RW    1          ;Number of System Tracks
DPBL      equ   15        ;Length of Disk Parameter Block

```

CP/M Related Values:

```

WIRPS     equ   WICSZ/128      ;CP/M Records per Sector
WIRPT     equ   WIRPS*WINSPT   ;CP/M Records per Track
WINST     equ   1              ;# of System Tracks
WINSYS    equ   WINST*WINSPT   ;# Sectors in System Track(s)
WIMIN     equ   1024/WICSZ*64+WINSYS ;Min # useable Sectors
WIMAX     equ   1024/WICSZ*8*1024+WINSYS ;Max # useable Sectors

```

Z217 Equates:

```

WINSPT    equ   18            ;# Physical Sectors per Track
WICSZ     equ   512           ;Cell Size used

```

Type 1 Command Step Rate Flags:

```

FDFS RM   equ   00000011b     ;Step Rate Mask
FDFS 6    equ   00000000b     ;Step Rate 6(3) MS
FDFS 12   equ   00000001b     ; 12(6)
FDFS 20   equ   00000010b     ; 20(10)
FDFS 30   equ   00000011b     ; 30(15)

```

Control Register Flags:

```

CONDS     equ   00000011b     ;Drive Select Bits
CONDS 8   equ   00000100b     ; 0=5", 1=8"
CONDSEN   equ   00001000b     ;Drive Select Enable
CONPC     equ   00010000b     ;Write Pre-compensation
; 5" 0=Yes, 1=No
; 8" 0=All Tracks, 1=Tracks 44-76
CON5FS    equ   00100000b     ;5" Fast Step
CONWE     equ   01000000b     ;Enable Wait for DRQ or IRQ
CONSD     equ   10000000b     ;Enable Single Density

```

XLATE Table Addresses:

```

XLATES    DS    0              ;XLATE Table Addresses
          DW    0              ; No XLATE
          DW    XLATE1         ; 8" Single Density
          DW    XLATE2         ; 8" Double Density
XLATE1    DS    0              ;8" Single Density Sector Translate Table
          DB    1,7,13,19,25
          DB    5,11,17,23
          DB    3,9,15,21
          DB    2,8,14,20,26
          DB    6,12,18,24
          DB    4,10,16,22
XLATE2    DS    0              ;8" Double Density Sector Translate Table
          DB    1,2,19,20,37,38
          DB    3,4,21,22,39,40
          DB    5,6,23,24,41,42
          DB    7,8,25,26,43,44
          DB    9,10,27,28,45,46
          DB    11,12,29,30,47,48
          DB    13,14,31,32,49,50
          DB    15,16,33,34,51,52
          DB    17,18,35,36

```

Other Definitions Used Above (found at end of BIOS85.A86):

```

DIRBUF    DS    128
-- Directory Buffer, 128 bytes long
DPB1A,DPB1B,DPB1C,DPB1D,DPB2A,DPB2B DS DPBL ;DPBL = 15 bytes
-- DPB - Disk Parameter Blocks for every drive, 15 bytes long
CSV1A,CSV1B,CSV1C,CSV1D DS 64 ;Floppy Drives only
-- CSV - Address of software to check for changed floppy disks, 64 bytes
ALV1A,ALV1B,ALV1C,ALV1D DS 77 ;All Floppy Drives
ALV2A,ALV2B DS 256 ;HD Partitions
-- ALV - Address of area to keep track of a disk's storage allocation

```