

CP/M Primer

a most sophisticated operating system

Dr. John F. Stewart
University of Miami
Box 248237
Coral Gables FL 33156



8041 NEWMAN AVE. — SUITE 208
HUNTINGTON BEACH, CA 92647
(714) 848-1922

Reprinted from April 1978 Kilobaud Magazine.

Dr. John F. Stewart
University of Miami
Box 248237
Coral Gables FL 33156

CP/M Primer

a most sophisticated operating system



The author works on an Imsai 8080 in the University of Miami's Hertz Computer Lab.

Many fine articles have appeared in *Kilobaud* describing the principles of operating systems, but, as yet, no one has taken it on himself to present a detailed description of one of the several commercially available operating systems. In this article, we will take a look at the disk-based CP/M system written by Digital Research. In particular, we will look at the version of CP/M that is currently available on the Imsai 8080 microcomputer system. There are only minor differences between the Imsai version and the original, so most of the following will apply to both.

The CP/M operating system is currently available from Digital Research for \$70, including documentation and system diskette. I estimate it would take three to six man-months for a sophisticated programmer to produce an operating system with CP/M capabilities. Thus, one of the original problems inherent in the microcomputer field — a lack of inexpensive software — has apparently been alleviated.

Environment of CP/M

The essential structure of the CP/M operating system is shown in Fig. 1. The CCP is the Console Command Pro-

cessor — the part of the operating system with which a user converses. A wide variety of commands is available, and these commands will be discussed below.

Basic Input/Output System (BIOS) is the section of CP/M that deals with input/output commands to all peripheral devices except the floppy disks. This includes I/O to Teletype, CRT, printer, etc. A nice feature of BIOS is that the system I/O routines are available to the user through appropriate subroutine calls in assembly-language programs. This capability constitutes a powerful addition to the assembly-language arsenal.

Basic Disk Operating System (BDOS) interfaces the system with the floppy-disk peripherals. Again, these routines are available to the user, eliminating what is typically one of the trickiest aspects of assembly-language programming — that of I/O programming.

The first 100₁₆ (256₁₀)

bytes of memory are used primarily as a scratchpad by the system. Various system parameters, such as where to jump on a restart, are contained in this area. In addition, a fair amount of it is available to the user. In particular, the default location of the top of the stack is location FF₁₆ (255₁₀).

Finally, the Transient Program Area (TPA) is the area of memory available for user programs. It comprises the bulk of memory, even in the 16K system where it is 26FF₁₆ (9983₁₀ or 9.75K) bytes in length. In addition to user programs, all CCP transient commands are executed in the TPA. Thus, all user and most service programs originate at location 100₁₆.

The CP/M system is a disk-based system, so that an important part of the environment confronting the user has to do with the way the diskettes are structured. The diskettes are composed logically of 77 concentric tracks numbered from outside to inside as track 0 through track 76. The first two tracks (0 and 1) are used to hold the CP/M system, which is bootstrapped into memory as indicated in Fig. 1, when a cold-start procedure is initiated. Tracks 2 through 76 are available for the directory (usually on track 2) and user or system disk files (programs or data files). Each track contains 26 sectors, each of which is capable of holding 128 bytes of information. Total disk capacity, then, is a little in excess of 250K bytes, of which just over 240K is available for user files.

There are several important points to make about the disk environment. First, it

File Type	Meaning
BAS	BASIC program.
ASM	Assembly program.
SUB	Submit file.
INT	Intermediate BASIC.
PRN	Assembly results.
HEX	Assembly output.
COM	Command file.

Fig. 2. File types.

is not necessary for the user to specify where on a diskette a particular file will go. The (BDOS) system automatically finds the necessary space and keeps a record of the name and location of each file in the diskette directory. This saves the user the trouble of remembering where a particular file is located. Names of disk files are made up of three parts:

1. The first letter is used to indicate which drive (A or B) the diskette is on. This letter is optional if the operating system is told to assume that all files are on a particular drive.
2. The second part of the name is called the file name. It consists of from one to eight letters and/or numbers.
3. The last part is the file type. File type is used to indicate whether a file is a BASIC program (BAS), an assembly-language program (ASM), etc. A list of file types is given in Fig. 2.

Valid disk file names are shown in Example 1.

A file name as defined above constitutes an unambiguous file reference. In many cases, it is desirable to refer to a whole set of files with similar characteristics. This is done through the use of an *ambiguous file reference*. File references can be

```
A:MYPROG.BAS
B:F12.ASM
PIP.COM (uses default drive)

Example 1.
```

BIOS/BDOS	3100 ₁₆ - 31FF ₁₆
CCP	2800 ₁₆ - 30FF ₁₆
TPA	100 ₁₆ - 27FF ₁₆
System Area	0 - FF ₁₆

Fig. 1. Structure of CP/M 16K system.

ambiguous in one of two ways:

1. An asterisk can be used in place of either file name or file type to indicate any file name or file type. Thus, *.BAS refers to all BASIC language source files while MYPROG.* refers to all files named MYPROG, no matter what type they are.
2. One or more question marks can be used in place of characters in either file name or file type to indicate that any character in that position is acceptable. Thus, files TEST1.BAS, TEST2.BAS and TEST3.BAS could be referred to as TEST?.BAS.

Console Command Processor

As stated above, the CCP is the part of CP/M with which a user communicates. CCP prompts the user with a letter that indicates from which disk drive the system has been taken (also the default disk drive for file references) followed by a greater-than character (i.e., A > or B >).

Two types of commands are possible in CCP. There are built-in commands such as DIR (list directory of default disk), ERA (erase a file), REN (rename a file), TYPE (list a file) and SAVE. These commands are referred to as built in since the code for them is in the CCP area. The DIR and ERA commands allow the use of the full range of file references. For example:

```
DIR *.BAS
```

would list the names of all directory entries on the default drive that have file type BAS.

Transient commands execute in the TPA just as user programs do. A nice feature of CP/M is that, in order to execute any program (system or user), the user merely types its name in response to a CCP prompt. Thus, the runnable version of a program has a file type COM (for command).

There are five important areas addressed by CCP tran-

Command	Action
B	Moves pointer to beginning of file.
-B	Moves pointer to end of file.
±nC	Moves pointer ±n characters.
nFxxx	Places pointer after nth recurrence of string xxx.
±nL	Moves pointer up or down n lines. 0L places the pointer at the beginning of a line.

Fig. 3a. Pointer positioning commands.

Command	Action
±nD	Delete ±n characters.
I	Insert text.
±nK	Kill (delete) ±n lines.
±nT	Type ±n lines on console. 0T types line up to pointer. 1T (or T) types line from pointer to end.

Fig. 3b. Basic edits.

sient commands:

1. Program entry and editing.
2. Utilities such as copying a file from one disk to another.
3. Generating and saving various versions of the operating system.
4. Debugging aids.
5. Language processing.

We will discuss these areas one at a time.

Entry and Editing (ED)

A powerful editor (ED) is included in the CP/M operating system. This is a character editor as opposed to a line editor, meaning that a file is considered to consist of one long string of characters with CR (carriage return) and LF (line feed) characters separating each logical line. This string is held in a buffer area in memory. A pointer must be properly positioned in the text to indicate the location of each edit. Some of the basic pointer manipulation commands are shown in Fig. 3a. As an example of the use of these editing commands, the command

```
*B2FXYZ↑Z-3C
```

accomplishes the following:

- Moves pointer to beginning of buffer.
- Positions the pointer immediately after the second occurrence of the string XYZ.

Note, ↑Z (control-Z) is used to delimit the string.

- Moves the pointer back three characters, i.e., it is now positioned before the X in the second occurrence (XYZ).

Once the pointer is positioned, a number of edits can be performed. These are listed in Fig. 3b. As an example of the use of these commands, consider the following two edit lines that are equivalent:

```
*B2FXYZ↑Z-3DOTT
*B2FXYZ↑Z-3C3DOTT
```

The first line positions the pointer immediately after the second occurrence of the string XYZ; deletes the three characters preceding the pointer, i.e., the XYZ; and finally prints out the resulting line. The second example first positions the pointer following the second occurrence of XYZ, then moves the pointer back three spaces, finally deletes the XYZ and prints out the resulting line.

Since programs are line oriented, it is useful to be able to perform the basic functions of a line-oriented editor: inserting a line between two existing lines, deleting a line and replacing a line. While these functions are not entirely obvious, they can be accomplished. Assume for

<pre> A> ED BIG.BAS (CR) 100A (CR) (bring first 100 lines into buffer) * (edit first 100 lines) 100W (CR) (write edited lines to temporary area) 100A (CR) (bring second 100 lines in) * (edit second 100 lines) *E (CR) (end edit) </pre> <p style="text-align: center;"><i>Example 2.</i></p>
<pre> A> PIP X.ASM = MAIN.ASM, SUB1.ASM, SUB2.ASM, SUB3.ASM </pre> <p style="text-align: center;"><i>Example 3.</i></p>
<pre> A> PIP TEST.BAS = CON: , X.BAS, Y.BAS </pre> <p style="text-align: center;"><i>Example 4a.</i></p>
<pre> A> PIP LST: = ONE.ASM, TWO.ASM, THR.ASM. </pre> <p style="text-align: center;"><i>Example 4b.</i></p>

demonstration purposes that the BASIC program shown in Fig. 4 is to be edited. We wish to insert the following line:

```
30 INPUT X
```

and also to replace line 50 by the line

```
50 NEXT I
```

These edit lines accomplish these functions:

```
*BF40↑Z-2C↑30 INPUT X
*BFX2YP↑ZOLK↑150 NEXT I
```

The first edit line positions the pointer after the 40 and then moves it back two characters so that it is before 40. Then the INPUT statement is inserted. The second example positions the pointer after the erroneous string X2YP, then moves it to the beginning of the line, kills the line and inserts 50 NEXT I. Thus, line replacement is done by first deleting, then inserting the new line. Actually, using this editor does grow on you after some practice, even though it seems complicated at first.

As the icing on the cake,

```
10 S = 0
20 FOR I = 1 TO 10
40 S = S + X
50 X2YP
60 PRINT S/10
70 END
```

Fig. 4. Sample text.

ED has several additional editing commands. For instance, the edit line

```
*BMSFIRST↑ZSECOND↑ZOTT
```

does a search for all occurrences of the string FIRST, replaces each occurrence with the string SECOND and prints out each altered line. The only new editing characters in this line are the S, which is the search command, and the M, which indicates that the next commands are to be repeated as many times as possible, i.e., until the end of the file. If a user is careful, he can perform most desired edits with the Search (S) command.

The above discussion assumes that the file to be edited is located in a memory buffer. The designers of ED were aware, however, that a particular user might not have enough memory to hold an entire program at once. Thus, the editor contains commands that have to do with bringing parts of the file into memory and writing already edited sections to a temporary disk file to make room for another segment of the unedited source. Fig. 5 gives a list of some of these commands. A user with enough memory to hold only 100 lines of BASIC might perform the sequence in Example 2 to

edit a 200-line program.

Note that the end edit (E) command does several things. First, it appends any remaining lines in the memory buffer to the temporary file, then it appends any remaining source file lines to the temporary file. Next, it renames the original file, giving it file type BAK (BIG.BAK) for backup purposes. Finally, it creates a file under the original name (BIG.BAS) from the edited temporary file. So part of every editing run is a backup of the original file.

Peripheral Interface Program (PIP)

A second major transient command is the PIP program. PIP consists of a number of parts that perform utility functions for the user. One of the basic utility functions available allows the user to make a copy of an existing file. The PIP statement

```
A> PIP NEW.BAS = OLD.BAS
```

will copy the file OLD.BAS on the default disk to a new file called NEW.BAS. A rather interesting extension of this basic idea is to copy several files back to back to a newly created file. The statement in Example 3 could be used to create a program file from a main program (MAIN.ASM) and append three subroutines (SUB1.ASM, SUB2.ASM, SUB3.ASM) called by the main program. This makes a modular approach to programming easy to implement. Just save commonly used subroutines as separate files and, when needed, append them to the main program with PIP.

In order to understand the final application of PIP, we must recall the difference between logical and physical devices. A physical device is just what you would think — a TTY, a CRT, a printer, etc. Logical devices are devices defined in the BIOS, such as CON (console) and LST (list). Logical devices must be assigned to specific physical de-

vices before communication between them is possible. Consequently, the cold-start procedure would be to assign CON to your TTY or CRT and to assign LST to your TTY (normally you would want hard-copy output). This assignment is accomplished by the use of a set of eight front-panel switches called the IOBYTE switches. For example, the switch settings



accomplish this assignment. The switches not used in this example are for assigning a tape reader and punch; so unless you have such devices, these would always be left in the zero position.

PIP allows the user to refer to these logical devices, and therefore to the corresponding physical devices. If we decided to write a program called TEST.BAS, consisting of a main program to be typed in at the console that calls two subroutines X.BAS and Y.BAS already located on the default disk, we could use Example 4a.

If we simply wanted a listing of ONE.ASM, TWO.ASM and THR.ASM, we could use Example 4b. The colon is necessary to distinguish the logical device name from a disk file.

System Creation and Maintenance

CP/M contains the software necessary for creating itself in various forms. A system can be created to accommodate any amount of memory from 16K to 64K bytes in increments of 8K. The transient commands CPM and SYSGEN are necessary to accomplish a change of system size, while SYSGEN alone will make a copy of an existing system. Typically, a user who has just installed a third 8K memory board would use the command CPM 24 * to generate a 24K system. The SYSGEN command is then used to write the newly generated

Command	Action
nA	Append next n lines of source to memory buffer area. n = # implies whole program.
E	End edit run. Create edited file.
Q	Quit edit run. Make no changes to file.
nW	Write n lines from memory buffer to temporary work file.

Fig. 5. Text movement editor commands.

Option	Meaning
A	Enter assembly-language mnemonics.
D	Display memory.
G	Execute with breakpoints.
L	Disassemble
M	Move a segment of memory.
S	Change memory values.
T	Trace program execution.
X	Examine CPU state.

Fig. 6. DDT command types.

system onto the first two tracks of the disk in drive B. This system can be given control by placing it in drive A and doing a restart. Thus, it is relatively easy to change system size. Even if the user has only one disk drive, this can be accomplished by modifying the SYSGEN command to write its output to drive A instead of drive B. Exactly how this is done is part of the next subject.

Dynamic Debugging Tool (DDT)

One of the more surprising transient commands to be found in the CP/M system is DDT. This command has a variety of options that enable the user to interactively execute an assembly-language program. Included in the package is the ability to set breakpoints, single or multiple step through the program, alter the command (runnable) version of a program, disassemble the command version of a program, insert assembly-language statements, examine status flags and more. These capabilities make DDT a useful and powerful part of CP/M. Fig. 6 gives a partial listing of DDT command types.

The customary process for using DDT is first to write and assemble a program so that the command version (file type COM) is available to

DDT. The debugging package is then invoked as follows:

```
A> DDT TEST.COM (CR)
```

This command loads DDT into memory instead of CCP, and DDT in turn loads TEST.COM at location 10016. Now any of the command types can be executed. For example, suppose we desire to test the code shown in Fig. 7a, which writes a 2 out to the front-panel programmed output lights. The assembled version is shown in Fig. 7b. Given that we have invoked DDT, we can illustrate its capabilities with a few ex-

```
JFS ORG 100H
MVI A, 2
OUT OFFH
JMP 0
END JFS
```

Fig. 7a. Sample assembly-language program.

Location	Machine Language
0100	3E02
0102	D3FF
0104	C30000

Fig. 7b. Machine-language version of TEST program.

amples. First, let's check to see if the program is in memory beginning at location 10016. This is done in Example 5a and this agrees with the machine-language version in Fig. 7b.

Now let's single step through the program to see if it performs its intended function (See Example 5b). The Trace command gives the state of the CPU, as indicated by the carry (C), zero (Z), minus (M), even parity (E) and auxiliary carry flags (I), the contents of the registers (A, B-C, D-E, H-L); the contents of the stack pointer (S), the program counter (P); the mnemonics of the instruction at the location pointed to by P (i.e., the instruction to be executed next); and, finally, the location from which the following instruction will be taken (010216). Let's take another step in Example 5c.

Here, the MVI A,2 instruction has been executed so the A register is changed accordingly. None of the status

flags have changed. The instruction about to be executed is the OUT, OFFH, and the next instruction will come from 010416. To finish the program, one more step (Example 5d) is required. Here, the program returns control to the operating system via JMP 0. The program seems to work properly.

Two examples of the other command types are as follows:

```
- L100,106 (CR)
0100 MVI A,02
0102 OUT FF
0104 JMP 0000
```

The disassemble command recreates the assembly-language mnemonics.

```
- A100 (CR)
0100 MVI A,01 (CR)
0102 (CR)
```

This sequence replaces the following instruction by MVI A,1. Assembly-language statements can thus be entered at any location in the program. DDT takes care of assembling such statements. Finally, the

```
- D100,10 6
0100 3E 02 D3 FF C3 00 00
```

Example 5a.

```
- T (CR) (trace one step)
COZOMOEIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI A,02 *0102
```

Example 5b.

```
- T (CR)
COZOMOEIO A=02 B=0000 D=0000 H=0000 S=0100 P=0102 OUT FF *0104
```

Example 5c.

```
- T (CR)
COZOMOEIO A=00 B=0000 D=0000 H=0000 S=0100 P=0108 JMP 0000*0000
```

Example 5d.

following sequence changes back to the original MVI instruction by changing the 0116 in location 10116 to an 0216.

```
_S100  
0101 01 02 (CR)  
0102 D3 (CR)
```

Note that the period ends the substitute mode.

As is readily apparent, DDT offers an invaluable tool for debugging assembly-language programs.

The Language Processors

The two main languages supported by CP/M are 8080 assembly language and BASIC. The assembler (ASM) interacts with CP/M as follows. Utilizing the editor, the user creates an assembly source file, say TEST.ASM, on disk. This file is assembled via the transient command ASM TEST. There are two important outputs of the assembly process. The results of the assembly, including error messages, are placed into a file named TEST.PRN. These results can be viewed via the TYPE TEST.PRN command. The other output is a disk file named TEST.HEX, which contains the machine-language output of the assembly. The LOAD TEST command now is invoked to create a new file named TEST.COM, which contains the binary

```
ED $1.ASM  
ERA $1.BAK  
ASM $1  
TYPE $1.PRN  
ERA $1.PRN  
LOAD $1  
$1
```

Fig. 8. SUBMIT file for editing, assembly and tests.

(runnable) version of the program. This version of the program can be tested simply by typing its name as a CCP command or via DDT as described above.

This whole process of editing, assembling, loading and running is such a common sequence that it would be helpful to be able to teach CP/M to do the whole sequence by itself. In fact, this can be accomplished using the concept of SUBMIT files. A SUBMIT file is a disk file of CCP commands, except that the specific names (or name) of the parameters are left unspecified. Instead, they are represented by \$1, \$2, etc. Fig. 8 shows a listing of a SUBMIT file named AS.SUB that is useful for the above editing, assembly and test process. To instruct CP/M to execute this SUBMIT file, the user simply types the transient command

```
A SUBMIT AS TEST
```

All occurrences of \$1 are

replaced by the first parameter in the parameter list (TEST), and CPM executes the list of commands as though they had been typed individually. SUBMIT files give CP/M a capability similar in nature to the job-stream concept in larger machines.

The CP/M BASIC is a full version of BASIC with floating-point arithmetic and the full complement of built-in numeric and character-handling functions. It takes 20K to run the BASIC language processors.

The procedure for running a BASIC program is to first create a disk file of BASIC source statements, say TEST.BAS. The BASIC-E TEST transient command does a partial compilation of the source file, producing an intermediate file called TEST.INT. The RUN-E TEST command is used to load and run the program.

Conclusion

The intention of this article has been to present enough details of the CP/M operating system to give the reader a flavor for the degree of sophistication of currently available software.

It is an interesting intellectual exercise to think about writing one's own operating system, but it seems clear that with such sophisti-

cated software available at a reasonable price, the time and cost of writing an operating system is prohibitive.

No comparison has been attempted between CP/M and similar software products on the market. It's difficult enough to keep track of the names of all the companies dealing in various aspects of the microcomputer market. It would take a great deal of time to evaluate all the software competitive with CP/M. I hope this article will offer a friendly challenge to others knowledgeable in particular micro operating systems. Let's see an article or two on these other systems. Let's bring micro-systems software out in the open. The personal effort is worthwhile and would be instructive to us all. ■

References:

- Imsai CP/M Floppy Disk Operation System Version 1.31, Rev. 0, 1976*, Imsai Manufacturing Corporation, San Leandro CA 94577.
- An Introduction To CP/M Features and Facilities, 1976* (this and all following refs. by Digital Research, Pacific Grove CA 93950).
- ED - a Context Editor for the CP/M Disk System, User's Manual, 1976*.
- CP/M Assembler (ASM) User's Guide, 1976*.
- CP/M Interface Guide, 1976*.
- CP/M System Alteration Guide, 1976*.

"Turn-Key" CP/M Systems

James J. Frantz

System power-up and loading procedures are almost too simple after implementing this technique. Perfect for small business (and home) applications.

The marriage of BASIC and Digital Research's CP/M has provided a powerful software team to the developers of business software. The hobbyist and the home computer game lover can also enjoy the result of this popular team. But, as the development of business systems reaches out to more and more users; ease of operation becomes more important.

As the development of business systems reaches out to more and more users, ease of operation becomes more important.

To bring a system up under CP/M after power-on is a multi-phased process and is not always easily performed by someone not accustomed to computer systems. As currently implemented, CP/M must first be loaded into memory from the diskette. This is normally very simple and involves inserting the diskette into the drive and depressing the "RESET" button. Once CP/M gets control, it responds with the "A>" prompt. The user must then type in the proper command. For most versions of BASIC the user must type a multi-word command in order to cause CP/M to load the BASIC which in turn will load and execute the desired program. In contrast to this elaborate procedure, most "big" systems immediately display a menu from which the user can select the desired program by entering the menu number. This minimizes, if not eliminates,

James Frantz, 94-285 Hiwahiwa Pl., Waipahā, HI 96797.

the chance of typing errors. Why can't this be done with CP/M?

Well, it can. This article will describe how to make version 1.4 of CP/M start executing a program immediately after "RESET" is pressed. A menu program will also be provided that shows how this feature can be implemented with a game diskette. This technique was developed to allow my young daughter and her friends to select and run their choice of game from my collection of BASIC and machine language games without adult intervention. This same technique is even more suitable to business applications, especially dedicated systems.

CP/M Fundamentals

To understand how this works, the organization and operation of CP/M must be considered. CP/M is loaded into the top of existing memory from the diskette. There are various schemes used by vendors of disk systems which offer CP/M to accomplish this, but in every case CP/M begins execution after being loaded

by entering at it's base. The base of CP/M is $2900H + b$, where b is the bias determined by the amount of memory in the system. In a 16K system this bias is zero and a 32K system has a bias of $4000H$. Starting at the base, CP/M is arranged as shown in Figure 1. The location of the input, or command buffer, and the storage location for the pointer to the command string are important in implementing automatic startup of a program.

Notice the zero byte at location seven. This zero tells CP/M that the command buffer is empty, i.e., there are zero characters in the buffer. The copyright notice which appears after the zero byte is over-written by whatever the user types after the "A>" appears on the screen. If this location contained something other than zero, CP/M would think that a command had already been typed, skip printing the prompt and begin processing the contents of the command buffer. By modifying the contents of the command buffer to contain a program name and changing the buffer length

Location $2900H+b+x$ x	Contents (Hex)	Description	Remarks
0 to 2	$C3\ 55\ zz^*$	JMP Instruction	Normal Entry Point
3 to 5	$C3\ 51\ zz^*$	JMP Instruction	Alternate Entry
6	7F	Max. Length of command buffer	Max. number of input characters allowed
7	00	Length of command string in buffer	Normally zero
8 to 23	20	ASCII blanks	
24 to 61	various	Copyright Notice	
62 to 135	00	Remainder of the command buffer	Initially Empty
136 & 137	08 (base)	Scan Pointer Storage	Points location 8

*zz = $2900H + b + 0300H$

Figure 1

Turn-Key, con't...

byte to non-zero, CP/M can be made to execute that program every time CP/M is loaded just as if the user had typed it.

Also notice the storage location of the scan pointer. Initially this location contains the address of the beginning of the command buffer (base + 8).

This technique was developed to allow my young daughter and her friends to select and run their choice of game from my collection of BASIC and machine language games without adult intervention.

This pointer is updated as CP/M scans the command buffer during processing. After processing the command, the pointer is again stored at this location for possible continuation. This will become important when the need to start processing from the beginning of the command buffer is desired. Use of this storage location will be demonstrated in the MENU program described later.

Modifying CP/M

The procedure to modify CP/M is straightforward and can be easily accomplished with either SID or DDT. DDT is normally provided with CP/M when purchased, so this program will be used to make the modifications. First, be sure that a spare diskette is available to receive the modified CP/M. Next, a copy of CP/M must be made so DDT can bring it into memory. This is accomplished using SYSGEN, a program that is also distributed with CP/M. Run the SYSGEN program as follows:

SYSGEN start the SYSGEN program
SYSGEN VERSION 1.4 SYSGEN
 sign on message
SOURCE DRIVE NAME (OR RETURN TO SKIP) A Type "A"
SOURCE ON A, THEN TYPE RETURN

At this point, be sure the diskette in drive A contains the CP/M to be used, then type a "RETURN." The program should respond with:

FUNCTION COMPLETE CP/M is now in memory
 DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

The system will reboot at this point leaving a complete copy of CP/M in memory starting at 0980H and ending at 207FH. This copy of CP/M must be saved on the diskette. The save operation can be accomplished by typing:

SAVE 32CPM.COM Save 32 pages of memory

command of DDT. Type "S0987[cr]" and DDT will respond with:
 0987 00 - DDT waits for user entry "MENU" is four characters in length, so enter "04" at location 0987H. DDT will respond with the next address. Enter the hexadecimal value for the each letter of the command name followed by a "RETURN." Repeat the process until the complete name is entered. Finally, and this is very important, enter a "00" immediately after the program name. After completing these entries, exit the "S" command of DDT by typing "[cr]" and then perform another "D0980[cr]." The data displayed should now appear as shown in Figure 3.

```

D980
0980 C3 55 6C C3 51 6C 7F 04 40 45 4E 55 00 20 20 20 .UL.01.MENU
0990 20 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48 .COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 38 2C 20 44 49 47 49 T <C> 1978. DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 .TAL RESEARCH
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0A00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0A10 05 00 C5 CD 8C 69 C1 C9 3E 0D CD 8C 69 3E 0A C3 .I.>.I>.I>.I>
0A20 8C 69 C5 CD 98 69 E1 7E B7 C8 23 E5 CD 8C 69 E1 .I.>.I.>.I.>.I.>
0A30 C3 A7 69 0E 0D C3 05 00 5F 0E 0E C3 05 00 0E 0F .I.>.I.>.I.>.I.>
  
```

Figure 3

Now that a copy of CP/M exists as a disk file, DDT can load this file for modification by typing:

DDT CPM.COM Load DDT which loads CPM.COM

DDT should respond with:

NEXT PC
2100 0100

CP/M is now back in memory starting at 0980H. Using the "D" command, type D0980[cr]. The display should look like Figure 2. Notice the two JMP instructions, followed by a 7F, then the zero byte. The zero byte will be replaced by the length of the program name, although any non-zero byte will work. After this length byte, enter the name of the program that is to automatically get control. This must be a disk file of type ".COM" in order to be executable. In the example, the program name is "MENU.COM," so "MENU" is entered beginning at location 0988H. The easiest way to enter these modifications is with the "S"

When the copy of CP/M in memory has been correctly modified as described, exit DDT by typing: "G0[cr]." Without doing any intervening operation which might destroy the memory image of the modified CP/M, save this copy by typing:

SAVE 32 AUTOCPM.COM save 32 pages of memory

Then, using SYSGEN again, construct a diskette with the new system as follows:

SYSGEN start the SYSGEN program
SYSGEN VERSION 1.4 SYSGEN
 sign on message
SOURCE DRIVE NAME (OR RETURN TO SKIP) type "RETURN"
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) A
DESTINATION ON A, THEN TYPE RETURN

Remove the diskette from drive A and insert the spare diskette which was previously prepared to receive the copy of the newly modified CP/M. When ready, type "RETURN." SYS-GEN responds:

FUNCTION COMPLETE CP/M is now on diskette
 DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Remove the new diskette and replace the original diskette. Type "RETURN" to reboot the system. The new diskette should be labeled and safely set aside until the MENU program has been prepared.

```

D0980
0980 C3 55 6C C3 51 6C 7F 00 20 20 20 20 20 20 20 .UL.01.
0990 20 20 20 20 20 20 20 20 43 4F 50 59 52 49 47 48 .COPYRIGH
09A0 54 20 28 43 29 20 31 39 37 38 2C 20 44 49 47 49 T <C> 1978. DIGI
09B0 54 41 4C 20 52 45 53 45 41 52 43 48 20 20 00 00 .TAL RESEARCH
09C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
09F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0A00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0A10 05 00 C5 CD 8C 69 C1 C9 3E 0D CD 8C 69 3E 0A C3 .I.>.I>.I>.I>
0A20 8C 69 C5 CD 98 69 E1 7E B7 C8 23 E5 CD 8C 69 E1 .I.>.I.>.I.>.I.>
0A30 C3 A7 69 0E 0D C3 05 00 5F 0E 0E C3 05 00 0E 0F .I.>.I.>.I.>.I.>
  
```

Figure 2

Turn-Key, con't...

Menu Program In BASIC

At this point it should be pointed out that some versions of BASIC which operate under CP/M allow entering a command in the form:

```
A BASIC MENU.BAA
```

where BASIC is a disk file of type ".COM." Entering a command in this form causes CP/M to load and execute BASIC.COM, which in turn will load and execute MENU.BAA. (Note

Conditional assembly was used to allow both machine language programs (.COM files) and BASIC programs to be "menu-ized."

that the file type ".BAA" is optional in some versions - check your User's Manual.) CBASIC and a version of MICROSOFT BASIC distributed by TEL, Inc. are known to perform in this way. If the menu scheme is to be implemented in a BASIC of this type, or if it is desired to have a specific BASIC program begin execution immediately after power-up, this can be done by entering the entire command string, exactly as it would be typed, in the command buffer. Be sure to enter a non-zero character count in location 0987H. Most importantly, the command string must be followed by a 00. If the command string is so long as to overwrite the copyright notice, move the copyright notice to after the 00 byte following the command string. All the space up to and including 0A07H can be used, but be absolutely sure that no modifications occur above this location.

For those readers who use BASIC exclusively, the only work remaining is to transfer BASIC.COM and the desired startup program to the diskette with the CP/M which has been modified for automatic execution. If, however, the use of machine language programs is desired, or the version of BASIC doesn't have the facilities to load programs under the control of another BASIC program, the MENU program to be described might be the solution. In my case, new games were frequently being added to the game diskette, and many of the games were written in machine language. This program was developed to allow menu-ization of either BASIC games or machine language programs.

Assembly-Language Menu

The menu program in Listing A is written in 8080 machine language and is fully compatible with the Z-80. A fully commented listing is provided. The program has six major parts: 1) search and sort; 2) assign menu numbers; 3) compute column offsets; 4) display the menu in four columns; 5) input the user's menu selection; and 6) load and execute the selected program.

The search and sort loop uses the built-in capability of CP/M to search the diskette directory for files which match a specified pattern name. The pattern is selected so only the desired "type" of files are found. This is done by using the "?" which forces any character in the corresponding position to be a match. For example, if all files of type ".BAS" are wanted, a pattern of "???????BAS" would be specified. The desired pattern is set up in what is called a "File Control Block" or FCB for short. The standard convention for "calling" CP/M routines is to put the memory address of the FCB in the [DE] register pair, and the command number in register [C]. A call is made to the standard CP/M entry point which is memory location 0005. CP/M returns the directory "address" in the accumulator. Since 64 file names are allowed, this address is simply the sequential position in the diskette directory (0-63). If no file name matches the specified pattern, a FFH is returned. The "Search First" command is used to find the first occurrence of a match, and the "Search Next" command is used to find all subsequent file names which match the pattern.

As each file name is found, the directory address is converted to a memory address. Since all disk reads are performed in 128 byte blocks starting at the default address of 0080H, four directory entries are loaded into memory. Thus, the file names will be 32 bytes apart and only the two least significant bits of the directory address are needed. The file name is then alphabetically compared to the previous file names and inserted into its proper place in the "Directory Table" (labeled DIR\$TABLE in the listing).

The second major part of the program assigns the menu numbers to each file name in the Directory Table. Notice that space was allocated in the Table for the menu number. This arrangement greatly facilitates the display of the menu in column form. The CP/M "print buffer" command requires a "\$" as a termination character, so this is also inserted in the directory table to make printing

the table entries easier.

The third major part of the program figures out the column arrangement based on the number of files to be displayed. The files are listed alphabetically from top to bottom in four columns. The algorithm seems complicated, but was devised so that each column would have the same number of file names with the extras being added from left to right.

The fourth part of the program displays the Directory Table on the user's console using the offsets computed in part three. Extra line feeds are added to completely fill a 16 line video display terminal. This is easily changed to accommodate 24 line displays, or can be deleted as desired.

The fifth part of the program displays instructions to the user and waits for the menu number to be entered. Incorrect entries force a re-display of the menu. This portion of the program makes use of still another CP/M capability - buffered input. The [DE] register pairs are setup to point to a section of memory to be used to receive the input text. The first byte of this memory must contain the maximum length of the buffer area, and the second byte will be set by CP/M to the actual count of characters entered.

The last part of the program converts the menu number entered by the user into the program name. The ASCII representation which was entered from the console keyboard is converted to binary. This is then con-

The instructions to the operator would be reduced to explaining how to turn on the computer, how to properly insert the diskette, and the procedure for pressing the RESET button.

verted to the memory address of the corresponding file name within the Directory Table. Once the correct file name is pointed, the proper command string is positioned in the CP/M command buffer. Notice how conditional assembly was used to allow both machine language programs (.COM files) and BASIC programs to be "menu-ized." In the first case, only the file name followed by a zero byte is placed in the command buffer. In the case of BASIC, the name of the BASIC as a .COM file is followed by the selected program name. Some

the
mn
of
are
to
or-
ras
uld
es
eft

am
he
ets
ine
16
is
24
as

am
nd
be
re-
of
till
red
are
ory
xt.
ust
he
vill
of

on-
he
he
en-
is
on-

ne
ed
rn
to
k-
or
it-

he
he
ile
nd
/M
di-
ow
ms
to
ily
/te
In
he
by
ne

NG

Turn-Key, con't...

versions of BASIC, such as CBASIC, assume a file type, while other versions require that the file type be specified in the command line. The program allows either case to be handled.

After the menu number is converted to the correct address within the Directory Table, the location of the CP/M command buffer must be found. The buffer is known to start at the base + 7 (refer to Figure 1). The base is a known distance below the address stored at location 0006H. The base is needed since this is the entry point which will cause CP/M to process the command line which the program has constructed. So, by doing some simple arithmetic, the base of CP/M and the location of the command buffer can be found. The program does these calculations and the desired command string is properly positioned. There is one remaining problem which the program must solve. When first executed after initial load, CP/M detected the prepositioned command which was installed by the modification to CP/M. During the processing of this command, the scan pointer was moved to the end of this command and is no longer in the correct position to scan the new command line built by the program. Fortunately, the scan pointer is stored in a known location immediately after the end of the command buffer. Since the command buffer location is known, so is the storage location of the scan pointer. The program resets the scan pointer to the beginning of the command buffer and CP/M is ready to process the command.

Summary

This article has explained how to make CP/M execute a program upon power up by prepositioning the command in the command buffer. The requirement to reset the scan pointer to force CP/M to again process the command buffer was also explained. The producers of dedicated system software can take advantage of this capability and offer a system which is simpler to operate. The instructions to the operator would be reduced to explaining how to turn on the computer, how to properly insert the diskette, and the procedure for pressing the RESET button. Any additional instructions could be displayed by the program automatically executed. A true "turn-key" system? Perhaps not, but how much closer can you get without Read Only Memory? Happy Computing! □

DECEMBER 1979

DUNJONQUEST™ Presents

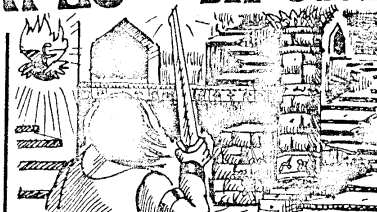
THE TEMPLE OF APSHAI

**VISIT THE
VAULT
OF THE
DEAD!**

The Vault of the Dead is but one of the many dark and fearsome mysteries within the ruined Temple of Apschai. The Temple of Apschai is your first adventure in the DUNJONQUEST™ series of fantasy role playing games.

DUNJONQUEST™ is a complete game system and The Temple of Apschai is a complete fantasy adventure game for you and your microcomputer.

**OVER 200 ROOMS!
OVER 30 MONSTERS!
OVER 70 TREASURES!**



- Take your favorite character - or let the computer create one for you!
- Let the Book of Lore guide you through DUNJONQUEST™ within the Temple.
- Decide to fight the monsters or grab the treasure and run - but don't think too long they'll come after you!

The Temple of Apschai for the TRS-80 (Level II 16K) and PET (32K) microcomputers.

Ask your local dealer or send us your check for \$24.95 to:

Automated Simulations-Department R, P.O. Box 4232, Mountain View, CA 94040

California residents please add 6% sales tax.

CIRCLE 115 ON READER SERVICE CARD

SUPERPASBALL

5-10 times faster
and more

- Circle 780 on Reader Service Card
- Optimized for speed
- Fully transparent
- Includes 128

- Circle 780 on Reader Service Card
- Optimized for speed
- Fully transparent
- Includes 128

CIRCLE 156 ON READER SERVICE CARD

