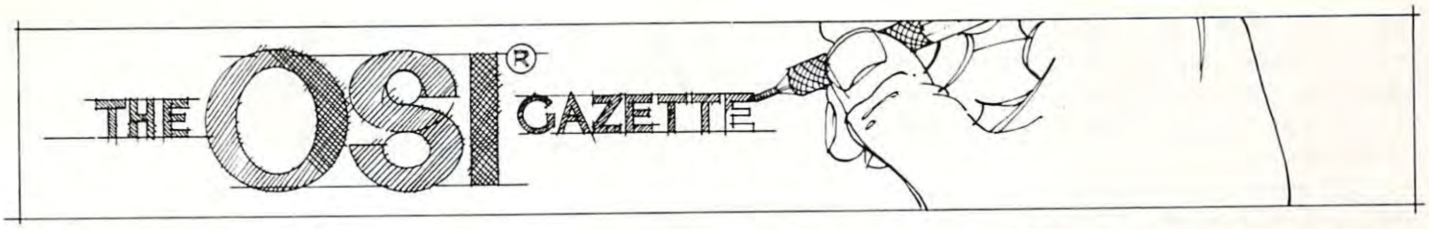


A Small Operating System: OS65D The Kernel

Part 3 of 3

Tom R. Berger
School of Math
University of Minnesota
Minneapolis, MN

Concluding Remarks

OS65D is a very small operating system. It is in no sense 'generalized' to run with a large variety of software or peripherals as, say, Digital Research's CP/M is for the Z80. If software and peripherals other than those supplied by OSI are to be used, then the operating system must be modified. There are advantages and disadvantages to such an operating system. Disadvantages result from its inherent inflexibility and lack of generalized commands. On the other hand, because the operating system is so very small and easy to understand, for those who choose to understand it, it is easy to modify to suit personal needs: a definite advantage.

Let's look now at some 'features' not available in OS65D. Essentially all the operating system is in memory at all times. This creates minor problems with peripherals and INPUT/OUTPUT. For example, the original conception by OSI of I/O leads to a sequence of routines exactly filling the I/O space. Time has shown that OSI did not make the perfect choice for all situations. In particular, the real time version of OS65D requires that certain of the I/O routines be partially overlaid or omitted to make room for expansions of other I/O routines. The missing routines are not easily returned except by special allocation. A more generalized system would have an area of memory for I/O routines (just as OS65D does), but this area would not have fixed routines in it. I/O routines would be written to run at any location and would be loaded into the special space from the disk when they were needed, and where a niche was available. After they had served their purpose, the space they occupy would become available for other routines. This 'generalized' approach eases I/O problems, but requires much additional coding to handle all the loading and space allocation.

The disk handling routines could not be made much more compact. In particular, many user functions are left out. Thus the user must do a large amount of housekeeping not required on larger systems. The most glaring deficiency is the file creation process. You cannot create a file until you know its size. Usually, you cannot know its size until it is in memory; but the file creation utility occupies the same space as the file. As a result, a scratch file must be created in order to temporarily save programs while a permanent file of the correct size is created. The process becomes even more involved if you wish to expand a current file beyond its current size.

If you use BASIC programs which process many files, then the error recovery process of OS65D is far too simple. If BASIC calls an operating system command (say DISK!"blah blah") and an error occurs, this error is often nonrecoverable. That is, the stack is reset and return to BASIC occurs through the WARM START. This often means your program will bomb if you try to CONTINUE. If you have a large amount of information stored in BASIC strings and in the process of saving it encounter a disk error, then without a great deal of knowledge about the internal working of BASIC, your information is lost.

Most file handling is done with BASIC utilities. If you are programming in assembly language, this leads to endless shuffling back and forth from BASIC to the Assembler and back.

The operating system lacks an adequate editor. Thus the Assembler and BASIC must contain their own editors. As a consequence, all input must be acceptable to one of these two editors if it is to be processed. In particular, line numbers are needed. A

BASIC program can be created to solve this numbering problem, but BASIC may be too slow. Solving this new problem leads to further complications which would not be necessary with a good operating system editor.

There are certain philosophical advantages to a small operating system. OS65D is small enough that its entire operation can be understood at once. This means hackers can modify and alter the system, not just by POKES and patches, but fundamentally, to suit their own needs. In my experience, most hobby OSI computer owners aspire to or already fall in this hacking category. The smallness of the system puts the user in direct contact with the most fundamental operating system commands and operations. Even though it is slightly more involved, this gives the user the very maximum of control over the system.

This article was written using disassemblies of OS65D V3.2 (NMHZ) Release November 1979. Future articles will cover: (1) the I/O routines; (2) the Disk routines; (3) the ROM, and (4) miscellaneous bits and pieces. The disassemblies I have made are fully annotated (by hand) and are available for those who would like to use them. Send a stamped, self-addressed postcard to me to determine availability.

Tom Berger
10670 Hollywood Blvd.
Coon Rapids, MN 55433 ©

SOFTWARE FOR OSI

VIDEO GAMES 1 .... $15.
Three Games. Head-On is like the popular arcade game. Tank Battle is a tank game for two to four. Trap! is an enhanced blockade style game.

VIDEO GAMES 2 .... $15.
Three games. Gremlin Hunt is an arcade-style game for one to three. Gunfight is a duel of mobile artillery. Indy is a race game for one or two.

ADVENTURE: MAROONED IN SPACE .... $12.
An adventure that runs in 8K! Save your ship and yourself from destruction.

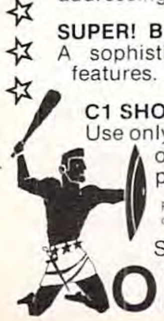
DUNGEON CHASE .... $10.
A real-time video game where you explore a twenty level dungeon.

BOARD GAMES 1 .... $15.
Two games. Mini-gomoku is a machine language version of five stones gomoku. Cubic is a 3-D tic-tac-toe game. Both with graphics.

DISASSEMBLER .... $12.
Use this to look at the ROMs in your machine to see what makes BASIC tick. Reconstruct the assembler source code of machine language programs to understand how they work. Our disassembler outputs unique suffixes which identify the addressing mode being used, no other program has this!

SUPER! BIORHYTHMS .... $15.
A sophisticated biorhythm program with many unique features.

C1 SHORTHAND .... $12.
Use only two keys to enter any one of the BASIC commands or keywords. Saves much typing when entering programs. Written in machine language.



For all BASIC-in-ROM systems. Selected programs available on disk. Color and sound on video games.

Send for **FREE** catalog

ORION SOFTWARE ASSO.
147 Main St. Ossining, NY 10562

A Six-Gun Shootout Game For The OSI C1P

Charles L. Stanford

The Six Gun Shootout game is a very pleasant and fun activity, particularly for the six to twelve or so age group. But this article concerns more than just the mechanics of writing another BASIC game for the C1P. When I originally wrote the program almost two years ago, we were reasonably satisfied with it. Sure, it was slow. Every time a player moved his gunfighter up or down the screen, the graphics POKEs took a lot longer than desired. And remembering that the "1" key was UP and the "2" key was down took a lot away. Those of you who have seen my articles on Fast Graphics (COMPUTE II Issue 3) and on interfacing the Atari Joystick to the C1P (COMPUTE Issue 7) can grasp what happened. Making that program work like it should has taught us more about the workings of the machine, over the past year, than any dozen manuals or articles.

This article, then, is a summing up of the methods we used to speed up both the software and the hardware to make BASIC games both more fun and much more saleable in the not inconsiderable Software marketplace.

BASIC Program Description

The game runs much as the early Arcade versions did. Each player has his gunfighter, who can shoot across the screen. Three Cacti obstruct some of the view, and move to a new location after each shot. Each player can move up or down, and shoot. Each gets 15 shots, and 5 hits wins.

The BASIC program shown in Listing 1 is fairly well annotated with REMs, but a few of the routines bear some discussion. The initialization starting at Line 5 sets the screen up as though no joysticks were available. This was deliberate, and makes the game more universally useful. It is a good idea to do this on all games, whether for paddles or for joysticks. The scoring from Line 200 is handled indirectly through the Fast Graphics Machine Language subroutine. Thus the POKEs of the ASCII characters are to that program rather than to the