DOS/65 NON-STANDARD CONTROLLER

VERSION 2.1

180 Ridge Road
Cimarron, CO 81220

SECTION 1 – INTRODUCTION

While the distribution version of DOS/65 contains input and output routines in BOOT and SIM which will work only with certain controllers, those with systems in some other configuration may still be able to modify the built in routines and use the system. This is, however, not a job for the inexperienced! You must understand your disk drive, controller, the rest of your system, and 6502 assembly language. More importantly you must have the ability to "manually" read, modify and then write individual sectors on your disk. If you cannot accomplish those steps or do not have the necessary knowledge to accomplish this difficult task - GET HELP!

SECTION 2 – WHAT TO CHANGE

Obviously, the first step is to figure out what must be changed. Both the SYSTEM INTERFACE GUIDE and the SIM, BOOT, and LOADER listings should be consulted at this point to figure out how to rework the software. The first routine to concentrate on is the LOADER. You really have two choices:

1. Create a new LOADER that reads in a new BOOT (or possibly more than just BOOT).

2. Create only a new BOOT.

The later option is viable since the initial reason for providing a separate LOADER was to allow a small (32 byte) loader to be located in a PROM on the controller itself so that the system would have an automatic power-on boot capability. That capability is really unnecessary if your monitor can be modified to include the LOADER or if you have ·some other mass storage device. If your system already has some mass storage I/O you could easily load the whole BOOT or possibly the entire system from those devices. I personally prefer to have a separate LOADER in EPROM for my systems since LOADER does not change as the Memory Size (MSIZE) or SIM length changes while BOOT does. The choice is yours. Remember that:

LOADER reads BOOT from Track 0, Sector 1 and then executes BOOT

BOOT reads CCM, PEM, and SIM into memory and then executes SIM

While a different method of loading the system can be implemented, e. g., the system could be stored as a binary file under some other operating system, such an approach probably will require modifications to SYSGEN itself to allow changes to be readily incorporated into your system.

SIM has not been mentioned yet since there will usually not be any way that you will be able to run the system without modifying SIM.

SECTION 3 – WRITE THE NEW ROUTINES

This step is certainly facilitated if you have access to an assembler. That is not absolutely necessary, however, since the routines of interest are not extremely large. LOADER and BOOT are both quite small and thus hand assembly is very practical. SIM is a good bit larger and while it certainly could be rewritten and hand assembled, the probability of error in that case is quite high. In any event, when you complete this step you should have, either on paper or some computer readable media the machine code for the new routines.

SECTION 4 – TEST LOADER

The modified LOADER should be tested to make sure it reads in BOOT properly . Make sure at this time that you do not attempt to execute BOOT since you have not yet modified BOOT contents. Once you have verified that the contents of Track 0, Sector 1 are being properly read, you can begin the next step.

SECTION 5 – TEST BOOT

Since BOOT can be tested without first actually changing what is stored on disk, it is strongly suggested that approach be used. Similarly to what was done for LOADER, make sure that the exits from BOOT are temporarily modified so that SIM will not be executed.

Once you are confident that BOOT is working properly, it is time for the first actual disk write. Fill a 128 byte block with the BOOT code and then write that block to the disk on Track 0, Sector 1. Then read it back with LOADER to make sure that the write was successful.

SECTION 6 – TEST SIM

This step is probably the most time consuming due to the amount of code involved. It is suggested that it be done in stages and that each stage be checked completely before going any further. Probably the first thing to do is to make sure that after BOOT loads everything, SIM will not actually attempt to set-up the system and execute CCM. Change the jump table at the beginning of SIM so·that it returns to your monitor when executed. Then, gradually subroutine-by-subroutine, modify and test the critical elements of SIM. It is suggested that the first thing you get working are the console I/O routines. Once that is done you can let SIM execution proceed a bit further to verify that those routines work. The last routines to implement should be the write routines. In fact, since CCM could be executed and the TYPE and DIR commands should work fine without any real write routine in SIM (substitute a LDA #0 and a RTS for the write routine) you can verify a good deal of the system.

SECTION 7 – CREATE BACKUP & MODIFY SOURCE CODE

Once everything is working, the next step is to create a backup diskette using COPY. As soon as that is done, the next step is to modify the BOOT and SIM source code routines provided, run SYSGEN, test the resulting new system and then make a new backup. At this point, you have completed your efforts. If your efforts are potentially of benefit to others, forward your notes, testings, and other documentation to me for inclusion in the DOS/65 documentation package.