SuperPET Deeper Down

by Brad Bjorndahl

Up to this point, my articles on the SuperPET have been intended to illustrate its character and potential. This may have resulted in an overly positive image of the machine. A little serious criticism is due, if only to placate the cynics.

Most of the topics have been general and at a high level: this is not the place to provide detailed descriptions of the ROM routines. At a high level the SuperPET is, in fact, well designed. The languages have only minor bugs, the ROM routines are useful and effective, and file handling is better than Commodore's.

On a nuts-and-bolts plane, difficulties arise. Waterloo used a high-level language called WSL to generate most of its code for the interpreters, editors and the operating system. Many sequences of instructions could have been reduced in size, if only WSL `knew' the functions of all the registers. Because of two projects that I will describe in a moment, it has become painfully clear just how difficult and inefficient the WSL code can be. For example, it has been estimated by more than one person (and I agree) that the operating system is about twice as large as it needs to be. Also, the editor has been rewritten so that it is several times faster, and requires two fewer disk blocks, than the WATCOM version. The rewritten version, by J. Toebes, is available from TPUG and ISPUG - see the December 1984 *SuperPET Software* article. To bring the point home, the WATCOM microBASIC interpreter requires 40K of memory despite using many ROM routines.

The first of the two projects involves a number of members of ISPUG (led by Toebes) who are attempting to prepare a BASIC compiler to run on the 6809 microprocessor. It will be modelled after the WATCOM microBASIC. There are currently no compilers for the 6809 side of the SuperPET and, since microBASIC is so well designed and BASIC is popular in general, it seems to me an obvious choice. After volunteering to help, I was sent two 4K banks of the WSL interpreter code. The first phase of the project, the disassembly, had been done before I entered. The second phase (at least as difficult) is to structure, comment and document the code as well as possible. After examining the output of WSL, I feel that the following comments are fair.

First, WSL is not optimised to the 6809 processor. There is a great deal of redundancy because registers are often loaded with values that they already contain. Secondly, many structures that are apparent in the code cannot be replaced with the structures provided by the microAssembler: I often find a 'Guess-Admit-Endguess' structure that contains a branch out of the structure, defeating its purpose. Perhaps WATCOM has a more sophisticated set of structure statements (as implied in their *Portable Software* article in the August/September 1984 issue). Thirdly, there are many `jump' statements, to the middle of other subroutines. What is worse, as far as disassembly is concerned, is that the destination subroutines are often in other banks of memory.

The same problems also made the second project more difficult. The OS-9 operating system, lately adapted for the SuperPET by TPUG (with special help from Avygdor Moise), requires routines called drivers, whose function is to interface the operating system with I/0 hardware. While working on the disk routines, Moise asked for assistance with the keyboard, screen and port drivers. I offered to write a driver for keyboard. In order to do this, I found that it was necessary to determine how the existing interrupt routines `interrogate' the keyboard PIA chip. That code was particularly messy, with numerous jumps and branches, but it was not very long. In short, I rewrote the keyboard driver in a structured way. Moise took the result, corrected a few bugs I had introduced, and improved the quality of the code still more. In addition, he added some new features (such as two-key rollover) and implemented a control key.

. . . It has become painfully clear just how difficult and inefficient the WSL code can be ...

As a result of being involved with these two projects, I can briefly describe the WSL output as low-quality code - the kind of work that, from a human programmer, might be termed `amateurish'. Does the quality of the code affect the user? This is the only important question for most SuperPET owners. There are two somewhat opposing considerations, with respect to the quality of software code - ease of maintenance, and efficiency. In this case, maintenance is not an issue. The average user will not be modifying the operating system or interpreters, as was done for the keyboard driver.

Code efficiency has two complementary aspects: size and speed. I have already described how much memory is demanded by the operating system and microBASIC. However, the memory used by the languages is bank-switched, and is not normally available to a user's program. All interpreters, large and small, provide about 30K of low memory to the user. No doubt PEEKS, POKES and system calls can make unused portions of bank-switched memory available, if necessary. Speed of code is a different problem. Any SuperPET user will tell you that the languages are relatively slow. Whenever I switch over to BASIC 4.0, I am surprised at how fast it seems to be. WATCOM microBASIC is much nicer to use than BASIC 4.0, but I will make a very rough guess that it is fifty per cent slower.

There was a time when I thought that the speed difference was somehow due to the extra features that had to be supported. I know now that it is mainly a result of inefficient code. Of course, no one uses interpreters for run time efficiency, but some users will find the long jobs tedious. For example, I have an APL program that searches numerical sequences for looping, and a search of five hundred sequences takes two to five hours, depending on the starting parameter. I have no doubt that reasonably efficient code would execute in minutes, not hours, but I can't complain too much. Without the APL interpreter I probably would not have the program at all.