CPUville

Donn Stewart
13917 Deviar Dr
Centreville, VA 20120
dstew@cpuville.com

Designing, Building, and Selling Obsolete Computers -- for Educational Purposes -- since 2004

Home 8-bit Processor Kit Z80 Kits Projects Educational Code Videos
News and issues Site map Links Contact Privacy

Tiny BASIC

Early personal microcomputers, such as the Altair 8800, were programmed using assembly language and machine code. However, there was much interest in developing higher-level computer languages for these early machines. The BASIC programming language (Beginner's All-purpose Symbolic Instruction Code) was a natural target for these efforts. The original BASIC was developed in 1964 and ran on mainframe computers. Many computer science students wrote their first programs in BASIC. So, when personal computers appeared, there was a natural desire to develop a BASIC language interpreter that could run on these new machines.

Microcomputers where handicapped in the early years by the size of memory, which usually cost more than the microprocessors. Nonetheless, a number of BASIC interpreters were written that could run in small memory spaces of only a few kilobytes. One of the most influential was Palo Alto Tiny BASIC, written by Li-Chen Wang in 1976. This program is famous for the comment in its source file heading that read “@COPYLEFT ALL WRONGS RESERVED”. It was one of the early formal expressions of the free software philosophy.

Tiny BASIC was written for the Intel 8080 processor, and 8080 machine code will almost always run on the Z80[1]. This BASIC interpreter runs in 2K of ROM, and 2K of RAM is more than adequate for writing and running small programs. So, on first look it seems to be well suited for use in the CPUville Z80 computer with the serial interface. The original Tiny BASIC 8080 assembly language was written in a dialect for a mainframe assembler has since been lost. However, Roger Rauskolb in October of 1976 modified the assembly language so that it could be assembled by Intel's 8080 Macroassembler, also known as MAC-80. The source code for this assembler was written in Fortran 66. Most mainframe and minicomputers of the day had Fortran compilers, so this assembler was easy for most hobbyists to access. Eventually, the MAC-80 assembler was made available in 8080 code, and was one of the popular programs on early CP/M computers.

So, with the 8080 source code for Tiny BASIC, and an assembler source code available in Fortran, there was a chance I could get Tiny BASIC up and running on the CPUville Z80 computer. First, I had to compile the MAC-80 assembler. Fortunately, the open-source Fortran compiler gfortran has the ability to compile obsolete Fortran dialects such as Fortran 66, and I was able to compile MAC-80 and run it on a PC, in the Linux environment. Next, I modified the Tiny BASIC code to match the CPUville Z80 computer hardware. Specifically, I had to change the status and data port addresses for the

UART (called the ACIA in the Tiny BASIC comments), the test bits for UART status, and the UART initialization bytes. I changed the ORG statements at the end of the Tiny BASIC code to better match a system with 2K ROM and 2K RAM. And, that was it! Tiny BASIC assembled without errors. I loaded it onto a ROM, and it runs fine on the Original CPUville Z80 computer with the serial interface.

Here are the Tiny BASIC files for use with the Original CPUville Z80 computer:

Tiny BASIC assembly language file
Tiny BASIC list file (in pdf format to show my changes in **BOLD**)
Tiny BASIC Intel hex file
Tiny BASIC binary file

Tiny BASIC is quite limited in its capabilities. It is integer-only – no floating point numbers allowed. It can only initialize one array, and can use only 26 variables, named A to Z. It does not have higher arithmetic functions, such as exponent. But surprisingly it has a random number generator function, so games with probabilities can be programmed. There is a full summary of the Tiny BASIC language in Appendix A of the Tiny BASIC instruction manual.

[1]The Z80 was designed to be compatible with the 8080. The 8080 registers, flags, and machine code are a subset of Z80 registers, flags, and machine code. All 8080 machine instructions will work properly on the Z80, with the exception of instructions using the rarely used parity/overflow flag.