The Philosophy of WHYP

Richard E. Haskell
Department of Computer Science and Engineering
Oakland University
Rochester, Michigan 48309

Abstract

WHYP (pronounced *whip*) is a subroutine-threaded version of Forth that is designed for use in embedded systems. WHYP stands for *W*ords to *H*elp *Y*ou *P*rogram. Its origin can be traced to eForth and the desire to make Forth more widely used in the design of embedded systems. WHYP is available and completely described in a new book on the 68HC12 microcontroller.

Introduction

The origin of WHYP goes back to 1990 when I implemented a 32-bit version of eForth for the Motorola 68000 microprocessor [1, 3]. This was a subroutine-threaded version of eForth that was also implemented for the Motorola 68HC11 [2, 4]. In this version of Forth, that I later called *ouForth* [5], all the headers were maintained on the PC and a host program written in 8086 assembly language ran on the PC and communicated with the target microcontroller through the serial (COM) port. In 1995 I rewrote the host program in C++ and changed the name to WHYP, which is pronounced *whip* and stands for *W*ords to *H*elp *Y*ou *P*rogram [6]. The language is specifically designed to be used for the design of embedded systems [7]. When the Motorola 68HC12 microcontroller came out I wrote a new version of WHYP for it that took advantage of the new instructions and addressing modes of the 68HC12 [8]. A new textbook on the design of embedded systems using 68HC12 (and 68HC11) microcontrollers was published by Prentice-Hall in the fall of 1999 [9]. This book, which is the first to describe the programming of the 68HC12 in detail, features WHYP as the central language used to program the 68HC12.

From Assembly Language to WHYP

Most 8-bit microcontrollers such as the 68HC12 are still programmed in assembly language. This means that each time a new and better microprocessor comes out the designer must first learn the new assembly language. The advantage of assembly language is that it is "closest to the hardware" and will allow the user to do exactly what he or she wants in the most efficient manner. While some feel that assembly language programs are more difficult to write and maintain than programs written in a high-level language, the major disadvantage of assembly language programs is related to the obsolescence of the microprocessor -- when upgrading to a new or different microprocessor, all of the software has to be rewritten! Even when upgrading from a 68HC11 to a 68HC12, which is upward compatible at the source-code level, to get the

best performance from the 68HC12 you will need to rewrite the code to use the newer, more powerful instructions and addressing modes.

This has led to a trend of using high-level languages such as C or C++ for microcomputer interfacing. While this helps to solve the obsolescence problem -- much of the same high-level code might be reusable with a new microprocessor -- high-level languages come with their own problems. The development environment is not always the most convenient. One has to edit the program, compile it, load it, and then run it to test it on the real hardware. This edit-compile-test cycle can be very time-consuming for large programs. Without sophisticated run-time debugging tools the debugging of the program on real hardware can be very frustrating. When designing microcomputer interfaces you would like to be as close to the hardware as possible.

What you would like is a computer language with the advantages of both a high-level language and assembly language, with none of the disadvantages. It would be nice if the language was also interactive so that you could sit at your computer terminal and literally "talk" to the various hardware interfaces. The language should also produce compact code so that you can easily embed the code in PROMS or flash memory for a stand-alone system. While you're at it why not embed the entire language in your target system so that you can develop your program "on-line" and even upgrade the program in the field once the product is delivered. This, of course, is what Forth has always been able to do.

However, these obvious advantages of Forth have somehow not made it into the consciousness of most embedded system designers. The writing of the book, *Design of Embedded Systems Using 68HC12/11 Microcontrollers* [9] (hereafter referred to as DESUM) is a modest attempt to correct this situation. The goal of the book is to introduce (seduce) the typical computer/electrical engineer, who wants to program the 68HC12 microcontroller, to programming in Forth (WHYP). WHYP is a subroutine threaded Forth which means that WHYP words are just the names of 68HC12(11) subroutines. In fact, it is introduced as a way of making assembly language programming easier by thinking of it as an interactive assembly language.

Chapter 1 of DESUM describes the architecture of the 68HC12 and shows how to write a simple assembly language program, assemble it, download it to the target board, and execute it. Chapter 2 shows how to write 68HC12 subroutines and describes how the system stack works. The question is how to pass parameters to the subroutines. This leads to the development of a separate data stack, using the 68HC12 index register, *X*, as a stack pointer. This data stack is then used to pass parameters to and from the 68HC12 subroutines (WHYP words). This makes it possible to access the 68HC12 subroutines interactively, by simply typing the name of the subroutine on the PC keyboard. A C++ program running on the PC is used to communicate with the target 68HC12 board using a serial line as shown in Figure 1.

The names of all WHYP words (subroutines) are stored in a dictionary that is maintained in the PC. We want to be able to "talk" to the target system by typing a WHYP word on the PC and having the corresponding WHYP subroutine execute on the target system. We do this by sending the address of the subroutine over the serial line to the target system. A small kernel program is running on the target system that waits for an address to be received on the serial line and then executes the subroutine at that address. The code for this kernel looks something like that shown in Figure 1, namely,

```
LOOP  BSR INWDY
      JSR 0,Y
      BRA LOOP
```

The subroutine *INWDY* waits for two bytes to be received in the serial port and stores this 16-bit address in index register *Y*. The statement *JSR 0,Y* then jumps to the subroutine whose address is in *Y*. The *BRA* (branch always) instruction always branches back to the label *LOOP*.

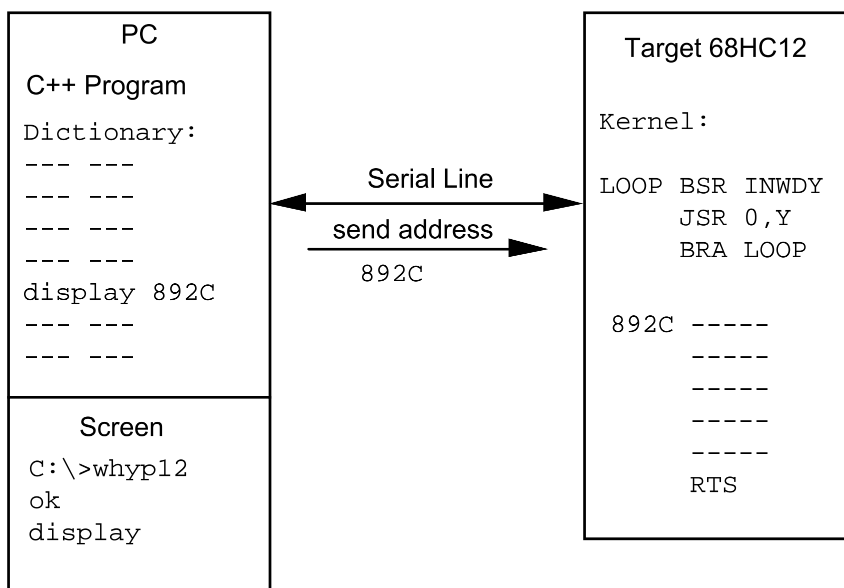

Figure 1 The structure of WHYP

Chapter 3 of DESUM discusses 68HC12 arithmetic with emphasis on the new 16-bit signed and unsigned multiplication and division instructions available on the 68HC12. These instructions are used to create WHYP words for all of the arithmetic operations. Chapter 4 shows how new WHYP words can be defined in terms of previously defined words using colon definitions. This chapter also introduces variables and constants and shows how to program the 68HC12 EEPROM. The 68HC12 branching instructions are described in Chapter 5 where it is shown how to use them to build the high-level WHYP branching and looping words IF...ELSE...THEN, FOR...NEXT, BEGIN...AGAIN, BEGIN...UNTIL, BEGIN...WHILE...REPEAT, and DO...LOOP.

The first five chapters give the reader a good understanding of the 68HC12 instructions and show how they are used to create the WHYP language. The next six chapters use WHYP as a tool to explore and understand the I/O capabilities of the 68HC12 (and 68HC11). The use of interrupts is introduced in Chapter 6 and specific examples of using interrupts in conjunction with various I/O functions are given in Chapters 7 - 11. These chapters cover parallel interfacing, the serial peripheral interface

(SPI), the analog-to-digital (A/D) converter, the programmable timer, and the serial communications interface (SCI) respectively.

More advanced topics related to programming in WHYP are described in Chapters 12 and 13. These include strings, string number conversion, and the use of defining words using CREATE...DOES>. The 68HC12 has special instructions that facilitate the implementation of fuzzy control. Chapter 14 discusses fuzzy control and shows how to design a fuzzy controller using WHYP on a 68HC12. A number of special topics related to the 68HC12 are covered in Chapter 15. Chapters 16 and 17 describe the C++ program for that part of WHYP that runs on the PC.

The essence of Forth (and WHYP) is simplicity -- always try to do things in the simplest possible way. Forth is a way of thinking about problems in a modular way. It is modular in the extreme. Everything in Forth is a word and every word is a module that does something useful. There is an action associated with Forth words. The words execute themselves. In this sense they are very object-oriented. We send words parameters on the data stack and ask the words to execute themselves and send us the answers back on the data stack. We really don't care how the word does it -- once we have written it and tested it so we know that it works.

WHYP is presented as a personal language that will allow the user to write programs for the 68HC12 incrementally and interactively. Because we develop WHYP from scratch in this book there is no mystery as to how it works. The entire source code -- both the assembly language and the C++ parts -- are included on the disk that comes with the book. In the true spirit of Forth this gives the user complete control over the programming environment. Forth is an extensible language -- and WHYP is a personal language that users will be able to extend and modify to suit their needs.

Running WHYP

WHYP is available for both the 68HC12 and the 68HC11. The kernel that resides on the target board contains the code shown in Figure 1 that waits for an address to be sent from the PC and then executes the subroutine at that address. The kernel contains over 135 subroutines written in 68HC12 assembly language, which represent basic WHYP words. The names and addresses of these WHYP words are contained in a .HED file that is loaded into a dictionary on the PC when the WHYP C++ program is executed. A configuration (.CFG) file is also read by the C++ program to configure WHYP for a particular 68HC12 system. This allows WHYP to be used on any 68HC12 system containing any member of the 68HC12 family of microcontrollers.

WHYP is run from a DOS prompt by typing WHYP12 (or WHYP11) at which point it looks like normal Forth. Each time a colon definition written, either by typing it on the keyboard or loading it from a file, the 68HC12 subroutine corresponding to the new word is downloaded to the target board at which point it is available to use.

Summary

WHYP is a subroutine-threaded Forth that is completely described in a new book on the 68HC12 microcontroller [9]. The goal of the book is to show the reader how the 68HC12 microcontroller can be understood and programmed by using WHYP. The book

develops WHYP from scratch as an interactive assembly language in such a way that by the end of the book the reader has not only learned about the 68HC12 but has also learned to program in Forth.

References

1. R. E. Haskell, "A 32-bit 68000 eForth Implementation for the Motorola Educational Computer Board," Proc. 1990 FORML Conference, Pacific Grove, CA, November 23-25, 1990.
2. R. E. Haskell, "Subroutine Threaded eForth," Proc. 1991 Rochester Forth Conference, Rochester, NY, June 18-22, 1991.
3. R. E. Haskell, A. M. Leshchyshyn and C. B Srinivas, "32-bit Subroutine Threaded eForth for the Motorola 68000," Proc. 1991 Rochester Forth Conference, Rochester, NY, June 18-22, 1991.
4. R. E. Haskell, "Design of a Subroutine Threaded Forth for Embedded Systems," Proc. Fourteenth FORML Conference, pp. 48-53, Asilomar Conf. Center, Pacific Grove, CA, November 27-29, 1992.
5. R. E. Haskell, "ouForth – a Subroutine Threaded Forth for Embedded Systems," Proc. 1993 Rochester Forth Conference, Rochester, NY, pp. 62-66, June 23-26, 1993.
6. R. E. Haskell, "WHYP – A C++ Based Version of ouForth for the Motorola 68HC11," Proc. 1995 Rochester Forth Conference, Rochester, NY, pp. 46-49, June 21-24, 1995.
7. R. E. Haskell, "Design of Embedded Systems Using WHYP," Proc. 1997 Rochester Forth Conference, Rochester, NY, June 25-27, 1997.
8. R. E. Haskell, "WHYP on a 68HC12," Proc. 1998 Rochester Forth Conference, Rochester, NY, June 24-26, 1998.
9. R. E. Haskell, *Design of Embedded Systems Using 68HC12/11 Microcontrollers*, Prentice-Hall, Upper Saddle River, NJ, 2000.