Forth for IBM Mainframe Computers

S.N. Baranoff
Leningrad Institute for Informatics and Automation
USSR Academy of Sciences
14 liniya 39, Leningrad, 199178, U.S.S.R.

Abstract

The FORTH-ES system is described. It is an implementation of the Forth language (Forth-83 standard) for the ES Ryad 1 and Ryad 2 mainframes (IBM 360 and IBM 370 compatibles). The system is substantially modular, with the same Forth nucleus running in different operating environments.

The System Design

The FORTH-ES system runs under the OS/360 and CMS operating systems. It consists of a Forth nucleus (binary threaded code placed into the dictionary), its extensions (Forth screens with source text), and auxiliary support modules, which are written in IBM assembler and which once compiled reside in a STEPLIB library as load modules of OS/360 (CMS uses object modules). The module interface works under standard system conventions [1]. Every auxiliary module performs several related functions, its first parameter usually being the function number. The Forth nucleus is stored as a library member and consists of binary data. Implementation of Forth screens is determined by the corresponding support module, which performs exchange with mass storage.

A run of the FORTH-ES system is started by an initial support module that receives control from the operating system. It issues a system GETMAIN macro for the Forth address space, initiates storage by placing the desired Forth nucleus into its lower addresses, and then passes control to this loaded Forth system. On getting control back, the initial support module analyzes the return code and either reiterates this cycle or stops the run by returning control to the operating system.

A minimal Forth nucleus consists of a small dictionary (about 200 words with a size of 8K) with a text interpreter. Its entry point is its very beginning. On being entered as an ordinary assembler program the nucleus dynamically loads and initializes all the necessary support modules (their names are stored within the nucleus as text constants, the loading being done through the load SVC). A dialog support module is loaded first, and after that a dialog with the user is started. By entering certain words the user may load new modules, change parameters of the starting procedure, compile new word definitions, and so on. As a result, a new Forth nucleus is created. The user may save the current dictionary contents as a new Forth module and the next time start running with the new version.

Dialog

The standard Forth words `KEY`, `CR`, and `EMIT`, which perform I/O with the terminal are implemented through calls to a module that supports dialog with fixed function numbers. The called module should reside in memory and be initialized at that moment. There are several versions of the dialog module, each one supporting a specific dialog system, OS, TSO, and BTAM being among them. Batch processing is considered as a particular case; the input text is read from a sequential file and the output is written into some other file.

As an example, terminal I/O under the TSO is carried out by the TGET and TPUT macros, which read and write whole terminal screens. Logically, the screen is divided into input and output fields, one for the user to place his input string into, and the other for the Forth system to display its output as the last portion of the dialog protocol. The user may save the protocol in a file and print it out later.

Because of the screen-mode I/O with the terminal, the dialog support modules include code for the words **EXPECT** and **TYPE**. But those two words may also be defined using **KEY** and **EMIT**.

Mass Storage

Three different types of Forth disk storage are used: usual text libraries, special libraries, and direct access files. With libraries, each Forth screen is represented by a library member, the number of the screen being contained in the 8 byte long member name. With a direct access file, the screen number is regarded as a corresponding block number within the given file.

The disk support allows further extensions of the Forth nucleus by simply **LOAD**ing the desired Forth source screens.

Threaded Code

The system works with 16-bit direct addressing within the standard Forth address space of 64K. That means that the main unit of information is an IBM halfword. The Forth dictionary starts at address zero, and the high end of the address space is occupied by the disk buffers, the user area, the text input buffer, and stacks that grow toward low memory.

After reading the Forth module into the address space, control is passed to its first byte according to the standard call interface. The size of the address space and the size of the loaded nucleus are passed as parameters. The module starts its work by setting pointers within the address space and then jumps to the infinite loop of the text interpreter.

The implementation uses indirect threaded code [2]. An entry consists of the name field, link field, code field, and a possible parameter field. Because of alignment requirements, each entry starts at an IBM halfword boundary and the name field always contains an even number of bytes, being padded with a zero byte if necessary. This does not influence standard programs since the word **COUNT** returns a "true" length that may differ from the value of the count. The parameter field in a colon definition is a series of code field addresses of other definitions, and its code field itself points to the CALL procedure of the address interpreter. The series of code field addresses within the parameter field ends with a reference to the **EXIT** entry, which performs the **RETURN** action. The code field of a code definition points to its own parameter field where the appropriate machine code resides. At the end of execution the code performs a jump to **NEXT**.

The address interpreter of this implementation is as follows:

```
        USING   NEXT,RBASE
NEXT    LH      14,0(RI,RBASE)     Code field address
        AR      RI,RTWO            Increment I
        NR      14,RMASK           Clear high-order halfword
        LH      15,0(14,RBASE)     Forth address of machine code
        NR      15,RMASK           Clear high-order halfword
        AR      15,RBASE           Absolute code address
        AR      14,RTWO            Forth parameter field address
        BR      15                 Execute the code
```

```
CALL      SR     RRET,RTWO    Move return stack pointer
          STH    RI,0(,RRET)  Save I
          LR     RI,14        New series of pointers
          BR     RBASE        Jump to NEXT
RETURN    LH     LI,0(,RRET)  Restore I
          NR     RI,RMASK     Clear high-order halfword
          AR     RRET,RTWO    Release return stack
          BR     RBASE        Jump to NEXT
```

In the above source code the general register RBASE is a base register for the FORTH address space, **NEXT** being at the zero address for Forth. RI is a pointer to the address currently being interpreted. RTWO contains the important constant 2 (the increment). RMASK contains another important constant, 65535, for clearing the high-order halfword of a register. And RRET contains the absolute address of the top of the return stack.

The Assembler

Implementation of the built-in assembler follows the general pattern for Forth assemblers [3]. The only difficulty is with the base-displacement scheme of addressing data in the IBM mainframe architecture.

During compilation, address operands of machine instructions are represented by two values — the address (which may be not yet defined) and its possible modifiers. Upon compiling an undefined address, this reference is stored in a special table to be resolved later. The word **END-CODE** checks all unresolved references. Following the system conventions, every piece of machine code is addressed through register 15 when it is entered from the address interpreter. The words **USING**, and **DROP**, control allocation of the base register.

Labels may be used within code definitions, as in the MIX assembler [4], by using the similar words **=F** (a forward reference), **=B** (a backward reference), **=H** (define a label) and by denoting labels by integers. These constructs allow conditional branches and loops on count according to the appropriate machine instructions.

The Target Compiler

In order to bring about full modularity, a package for target compilation was developed as two sets of Forth screens — one with the word definitions for the target compiler and the other with the model of the Forth nucleus. The target compiler allows "build-up" of a new Forth system at the very beginning. The user may substantially diminish the system by excluding from the nucleus all unnecessary definitions and by compiling for some (or all) entries only their bodies (code and parameter fields without the name and link fields). [This is what would be termed headerless code in English, Ed.] The code produced by the target compiler is stored in an outer library as a new, specialized, Forth module.

References

[1] *IBM System/360 Operating System*. Supervisor and Data Management Services. IBM Systems Reference Library. 1970.

[2] Ritter, T. and G. Walker. "Varieties of Threaded Code for Language Implementation." *Byte*, v. 5 # 9, pp. 206-227. 1980

[3] Ragsdale, W. F. "A Forth Assembler for the 6502." *Dr. Dobb's Journal*, # 59, pp. 12-24. Sept. 1981.

[4] Knuth, D. *The Art of Computer Programming*. v. 1. 1973.