Forth language

Complementing his description of a 6809-based microcomputer, Brian Woodroffe details the language used – Forth – and why he chose it, in this second series.

Forth is a language well suited to modern microprocessors and is widely used in such diverse applications as word processing, data-base management, instrument and process control, video games and data acquisition. In a kernel of less than 10Kbyte the following features are provided

– An interactive system.
– A high-level compiler with all standard control features.
– Fast execution, comparable with machine code because of the compiler.
– The language system is largely processor independent; only around 20% of the code written in assembly language need be changed to suit the computer.
– Virtual memory and application-oriented program modules.

Further, the system may be readily extended to suit new applications because the compiler can be modified by the user and new data structures introduced. These features are achieved by defining a virtual machine which is easily simulated by any target machine. Using 'threaded code', transferring control in the host from one virtual machine instruction to the next is quick and easy. Instructions of the virtual machine are used to build the monitor and compiler. Using the monitor the user may examine the effect of a series of Forth instructions and using the compiler this series may be added to the instruction set for future use.

Background

Forth is a computer language for fourth generation computers[1]. The language would have been called Fourth but six letters would not fit in the IBM1130 job-control language that its inventor, C. H. Moore, was then working with. Today Moore's company Forth Inc. is foremost in marketing FORTH for many different applications, besides the field of astronomy where it first found favour[2]. Other companies such as Miller Microcomputer Services and Laboratory Microsystems sell their own versions of Forth but the prime mover of Forth in the home-computer/hobby field is the Forth Interest Group* (FIG). They have made versions of Forth available for many computers including the PDP-11 and for 8080/Z80, 6800, 8086/8088 and 6502 processors. There are many versions of Forth and while all are similar no two are necessarily identical. For example, Poly Forth, FIG Forth and Forth 79 are all Forth but they are not the same. They differ primarily because of differences in the processor on which they run (16 or 8 bit memory, port or memory mapped i/o, etc.). FIG Forth will be used in all following examples.

*Forth Interest Group, PO Box 1105, San Carlos, CA94070, USA.

by B. Woodroffe

Forth is a collation of different sofware concepts forming a coherent whole. As an operating system, it is not as powerful as most but it takes care of all terminal and disc input and output. Small assembly-language routines must be supplied by the user to interface his hardware to the relevant system calls. It is also possible that memory-allocation changes may also have to be made. Most of Forth is written in Forth. It may seem strange that a language may be defined in terms of itself but one would use English words to explain the English language. Defining the language in this way means that programs may be transferred between different computers and implementations. There is a base instruction set which must be written in the machine code of the host computer. This is the only machine code required and the process is known as simulating a virtual Forth machine.

Most computer languages are programs which, recognizing statements in a source language, convert them into a target language. Usually the source language is text readable by humans in ASCII form and output is machine code of the computer. This is not always the case: cross compiling results in the target code being different from the host computer machine code. More exceptionally there are cases where the machine code can only be executed by a hypothetical computer, an example being O-code for the language BCPL[3] and P-Code for certain implementations of Pascal[4]. This is also the case for Forth and the virtual-machine execution mechanism will be explained first.

Threaded code

Explanation is simplified by visualizing a machine-code program for the processor concerned as a succession of subroutine calls. These calls transfer program control to each subroutine in turn. A stack, i.e., last-in-first-out list, would be the mechanism by which each subroutine returns control to the correct point in the main program. Knowing that the main program is solely a succession of calls it is now possible to reduce the main program to a list of subroutine addresses by removing the subroutine op-code, and to have a special program known as an address interpreter to transfer control down the main program address list. This is called threaded code, for the main program is the thread into and out of which the address interpreter threads control[5], List 1.

List 1. Comparisons of hard code and direct threaded code.

Normal code	Threaded code	
	Address interpreter	Thread
call A	1: ip+1 –> ip	A
call B	call [ip]	B
call C	jmp 1	C

In List 1, letters A, B and C denote machine-code subroutines, ip is the threaded-code instruction pointer and parentheses indicate one level of indirection. Threaded code trades the cost of the code for each call saved for address interpreter speed. In a long program the code cost of the address interpreter will be negligible. Further savings can be made by replacing the subroutine return statement by a jump to the address interpreter and changing the address interpreter as shown below. This releases the stack pointer used for subroutine calls and returns. It is important that the instruction pointer can be speedily accessed, for example by keeping it in a processor register, so as not to slow down the address interpreter by causing unnecessary memory activity.

If the lists are considered to be the actions of a virtual machine then a software routine NEXT represents the hardware execution fetch of the virtual machine. In a threaded-code computer the time of interpreting these lists is dominated by the time of the NEXT operation so it is best to run threaded code on a computer that handles NEXT efficiently or to use microcode.

```
Code routine including return
A: xxx
   jmp NEXT

New address interpreter
NEXT: ip+1 –> ip
      jmp [ip]
```

Indirect threaded code

The next improvement is to allow called routines to be not just pure machine code but also address lists. This is done by having a special routine that knows that the following data in the list are not code but addresses that must again be interpreted. Further, the routine must suspend interpretation of the main program while interpreting this new list of addresses. Return of control to the suspended list is done using a stack to save and restore the instruction pointer which is similar to the machine-code subroutine call/return operation. There must be an equivalent code routine to return control to the main list.

```
Normal code routine
A: machine code
   . . .
   jmp NEXT
```

```
Threaded routine
P: sp−1 −> sp
   ip    −>[sp]   (push current ip)
   #L−1 −> ip     (start interpreting new
                   list)
   jmp NEXT
L: A              (code routine)
   B
   C

Return routine
[sp] −> ip        (pop ip)
sp+1 −> sp
jmp NEXT
```

As most routines are likely to be lists and not machine code this stacking method, similar to subroutine calling, will take a lot of code area. Considerable space would be saved if there was just one copy of this routine. The address interpreter would normally jump to this routine but it would also have to execute code routines. This is done by making the first element of each list a pointer to code rather than the code itself. In the case of lists the pointer points to the stacking operator but with code routines it points to the next code address.

```
New address interpreter
NEXT: ip+1 −> ip
        [ip] −> w
        jmp [w]

Stacking operation
DOCOL: sp−1 −> sp
       ip    −> [sp]
       w+1 −> ip
       jmp NEXT

Destacking operation
SEMIS: [sp]  −>ip
       sp+1 −> sp
       jmp NEXT

Code routine
A: $+1   (point to next location)
   xxx
   . . .
   jmp NEXT

List routine
DOCOL
P
Q
SEMIS
```

This is the equivalent of machine-code subroutine call and return instructions. In Forth, the stacking and destacking operations are called DOCOL and SEMIS respectively. At the beginning of each address list, the extra address introduces a level of indirection – this is indirect threaded code[6]. In Forth the lists are divided into two parts, one being the code field which points to the address and the other known as the parameter field where the code is. These two parts and dictionary data, to be described, form a WORD. Code pointed to by the code field determines how the parameter field is interpreted. In the case of code words, the code field points to the parameter field. When the code field points to DOCOL, the parameter field is to be interpreted in a similar way to a subroutine. It is possible for the code field to point to some other routine which may make different use of the parameter field. Two examples of this in Forth are DOCON and DOVAR. The former treats the value in the parameter field as a constant and pushes it onto the data stack, to be described, whereas DOVAR pushes the address of the parameter field which is used as the storage location for that variable. To enable these routines to access the parameter field a third register, known as 'w', is required.

The address interpreter for indirect threaded code is more complicated than that for direct threaded code and so it is even more important to choose a processor with a suitable instruction set. Surprisingly for direct threaded code, NEXT can normally be coded using the processor subroutine-return op-code provided that the processor uses a stack that may be placed anywhere in memory. As the stack pointer is pointing to the thread, the processor must not receive interrupts for the status cannot be saved without destroying the thread. NEXT for indirect code is more complicated as it involves an extra level of indirection.

Choosing a processor, stacks and language-control structures are subjects of the next Forth language article.

An i.c. in the Forth computer switch-mode power supply on page 61 of the July issue was incorrectly designated the MC3045. The correct designation is MC3405.

References

1. C. Moore, *Forth dimensions*, vol. 1, no. 6, FIG
2. C. Moore, *Astronomical Astrophysics Supplement*, 1974, vol. 15, pp.497-511
3. M. Richards, The portability of the BCPL compiler, *Software Experience and Practice*, 1976, vol. 1, pp.135-146
4. D. Barron, Pascal, the language and its implications, Wiley, 1981
5. J. Bell, *Communications of the ACM*, vol. 16, no. 6, pp.370-372
6. B. Dewar, *Communications of the ACM*, vol. 18, no. 6, pp.330-331

Glossary

Machine code. The representation, usually in hexadecimal, of the instruction and data encoding that is understood by the computer.

Assembly code. A human readable form of machine code. There is a one-to-one correspondence between assembly code and machine code.

Instruction fetch. A computer works by successively fetching and executing instructions. The instruction fetch is made from the location pointed to by the program counter. The program counter is incremented one instruction at a time unless a jump (branch etc) occurs.

Virtual machine. At any level of analysis the computer will have a repertoire of instructions that it can execute. This is normally the machine/assembly level instructions. However by running a program on this machine it can be made to look as though it has a different instruction set. It is possible to time share the computer between two or more users so that they both think they have a separate computer. These techniques are known as creating a virtual machine.

Op-code. Each different instruction is encoded into a unique symbol (usually binary, known as an op-code.

Host computer. The computer on which the program is currently executing.

Target computer. The computer on which the program being developed will execute.

Cross compilation. A cross compiler runs on one machine and produces output for another. Host and target machines have either different op-code encodings or instruction sets, or both.

Compiler. A program recognizes that the input language agrees with a defined grammar. If it agrees it will usually produce an output in some other defined language, or error messages as to why the input is not in the source language. Normally, input is English-like (e.g. Forth/Pascal) and output is machine code.

Microcode. Microcode is a mechanism used to build computers to understand machine-code instructions. Within a microcoded computer there is another computer with its own microcode instruction set. By writing new microcode, the assembly-level machine can be made to have new instructions.

Kernel. A central program on whose resources all application program rely and interface to.

Operating system. A computer program which manages the computer's resources. (It will take care of all input-output etc, so that the application programmer need not worry about how to get characters to and from a terminal, etc)

Software driver. A small program specific to each input/output device that is included in the operating system.

Terminal. Visual-display unit and keyboard, teletype.

Indirection. An addressing mechanism. An instruction requires data to act upon – the instruction gives details of how to find that data. Normally it will give the address of the data, but in the cases of indirect addressing it will give the address at which the address of the data may be found. That is one level of indirection. Up to three levels, ie the address which contains the address which contains the address which contains the data, are common.

Call. A subroutine call is a mechanism whereby machine-code execution is temporarily suspended while the subroutine is executed. Execution will restart at the instruction after the call when the subroutine finishes. The restart address (return address) is often kept on a stack.

Code field. A part of a Forth Word definition. The contents of the code field always point to machine code of the target machine.

Machine. Computer, (state machine).

Monitor. A program that monitors user requests as typed in at the terminal. Usually gives message (<OK>) when the command has successfully executed. Monitor is also the name given to a technique used in real-time programming, developed by C.A.R. Hoare et al.

Virtual Forth machine. The assembly-language programmer creates a virtual machine that executes lowest-level Forth instructions.

Virtual-machine execution mechanism. The means by which the assembly-language programmer makes the virtual Forth machine transfer control from one Forth instruction to the next.