Varieties of Threaded Code for Language Implementation

Terry Ritter
Gregory Walker
Motorola Inc, Mail Drop M2880
3501 Ed Bluestein Blvd
Austin TX 78721

Between a high-level language (HLL) and its underlying machine architecture lurk many language implementation techniques. These include the older techniques of interpretation and compilation, as well as newer ones like intermediate languages and threaded code. In this article, we will present four types of threaded code techniques for implementing intermediate languages. We will examine how these four logically equivalent techniques offer various trade-offs of execution speed, program storage, and use of processor resources.

Implemention of a Language

The implementation of a high-level language on various logical or physical machine architectures involves such characteristic trade-offs as size of the language implementation, size of generated code, and speed of program execution. We will bypass other issues of high-level language use (eg: interaction, debugging, testing, etc) and concentrate on language implementation considerations.

Language implementation techniques can be logically divided into two categories: translation and interpretation.

Translation: Translation techniques replace elements of higher-level syntax with lower-level instructions that perform an equivalent operation. The resulting translation is then executed in order to run the program. A compiler is a computer program that translates high-level language programs into instructions of another language. Traditionally, assemblers and compilers translate their input into machine-level code.

Interpretation: Interpretation techniques directly execute the high-level language program. The interpreter is a program that sees the high-level language source program as a series of operation (op) codes used to guide its execution. The interpretive system appears to the user as a "virtual machine" that has the architecture of the high-level language.

Any form of interpretation offers significant opportunities for implementing debugging tools. Tests performed as each command is interpreted can result in a programmer-controlled display of debugging information. This is the basis for trace or breakpoint facilities that can be included in the interpreter.

Combinations: Combination techniques may translate the sequence of characters representing a high-level-language keyword into a form that is easier to interpret. Most BASIC interpreters translate the BASIC keywords into one-byte tokens that are easier to identify. This technique avoids the continual string searches of a traditional interpreter, but executes a language that is syntactically unchanged from the high-level-language source program. (For our purposes here, the term *syntax* will specifically refer to the structural relationship between language elements.)

Intermediate language: Intermediate-language (IL) techniques translate the high-level-language programs into a language that is simultaneously easier to deal with and syntactically different from the original. Many compilers translate a high-level-language program into an intermediate language, which is then translated into

About the Authors

Terry Ritter and Gregory Walker are software engineers at the Motorola Microprocessor Design Group, where their exploration into the structure of computer languages led them to examine FORTH and other threaded languages for use as a possible software tool. Terry Ritter is one of the co-architects of the MC6809 microprocessor and has been involved with personal computing since 1974. Gregory Walker is on the IEEE floating-point standards committee and has been involved with microcomputers since 1975.

Diagnostics I

for CP/M* & TRSDOS#

Someday your computer is going to break; even the most reliable computer systems "go down". Often, finding exactly what is wrong can account for the most time consuming part of repairing the system, and the longer the system is down, the more money you lose.

DIAGNOSTICS I is a complete program package designed to check every major area of your computer, detect errors, and find the cause of most common computer malfunctions, often before they become serious. For years, large installations have run daily or weekly diagnostic routines as a part of normal system maintenance and check-out procedures.

DIAGNOSTICS I is designed to provide that kind of performance testing for 8080/Z80 micro computers.

DIAGNOSTICS I will really put your system through its paces. Each test is exhaustive and thorough. The tests include:

- Memory Test
- CPU Test (8080/8085/Z80)
- Printer Test
- Disk Test
- CRT Test

To our knowledge, this is the first CPU test available for 8080/Z80 CPU's. Many times transient problems, usually blamed on bad memory, are really CPU errors.

A good set of diagnostics is an indispensable addition to your program library even if your system is working fine. Hours have been wasted trying to track down a "program bug" when actually hardware was to blame!

DIAGNOSTICS I also allows you to be confident of your system. This can be critical when file merges or sorts and backups are involved. You want to be as sure of your computer as possible during these critical times. Running DIAGNOSTICS I prior to these and other important functions helps to insure that your system is operating at peak performance.

DIAGNOSTICS I is supplied on discette with a complete users manual.

DIAGNOSTICS I: $60.00 Manual only: $15.00

Requires: 24K CP/M; 16K disc for TRS-80

formats: CP/M 8" SOFT SECTORED, NORTHSTAR CP/M AND TRS-80 DOS

master charge VISA

All Orders and General Information:
SUPERSOFT ASSOCIATES
P.O. BOX 1628
CHAMPAIGN, IL 61820
(217) 359-2112
Technical Hot Line: (217) 359-2691
(answered only when technician is available)

*CP/M REGISTERED TRADEMARK DIGITAL RESEARCH
#TRSDOS TRS-80 TRADEMARKS TANDY CORP

Intermediate-language techniques offer the advantage of machine independence of the source language.

machine code. When used in this manner, the intermediate language can allow global code-optimization techniques to be more easily applied.

Since the translation into the intermediate language is independent of the target machine, different compilers for the same target machine need only produce the simpler code of the intermediate language. Similarly, different code generators (which translate the intermediate language into machine language) can allow the same compiler to produce code for different computers. Intermediate-language techniques offer the advantage of machine independence of the source language and allow *program portability*, the ability to execute the same source program on widely different computers.

The intermediate-language representation of a program might also be interpreted instead of translated to machine code. To minimize interpretation overhead, we need complex and powerful machine-language routines. But machine independence is best accomplished by having simple, easy-to-write machine-language routines. This same trade-off of machine independence versus execution speed must be made in the design of any intermediate language. An example of this use of intermediate language is the pseudocode (p-code) used to implement most versions of Pascal.

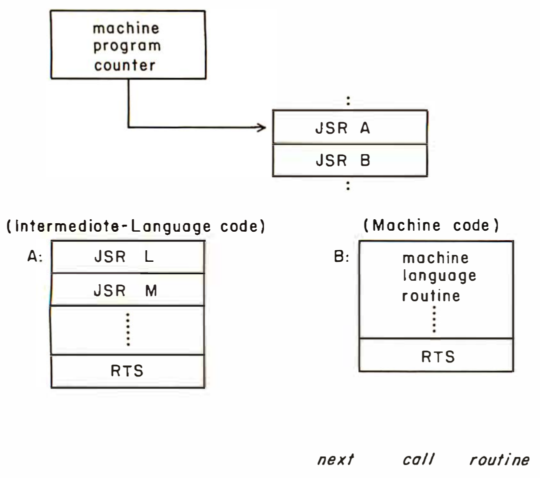
This article is principally concerned with a class of intermediate-language representations particularly suited to interpretation; these are known as *threaded codes*. Naturally, the intermediate-language code will be generated by a compiler or by some other translation program. We will not discuss the translation process, which is a function of the syntax of the high-level language and other programming considerations; rather, we will discuss the resulting intermediate language and its interpreter.

Aspects of Intermediate-Language Architecture

An intermediate language is composed of a set of primitive operations (which, in combination, can express any algorithm) and storage capabilities for both internal and program data. In particular, it must be possible to pass data values between routines that make up the intermediate language. The intermediate-language program can use a fixed number of memory locations to simulate general-purpose registers, but then routines are needed that load (and store) each register from memory, as well as routines that simply move values between registers. If the intermediate language approaches the complexity of the original machine language, its use is of dubious value.

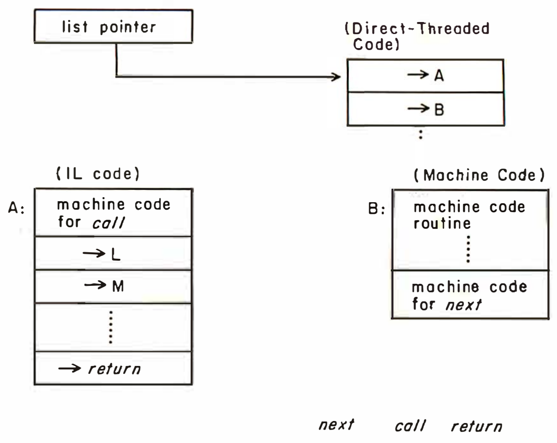
One approach that simplifies an instruction set is a "zero-address" or *stack* architecture. In this architecture, all operations will obtain values by pulling them from the stack and results will be returned by pushing them onto the stack. Only two operations with memory are now required: the "pull (from stack) and store (to memory)" operation and the "load (from memory) and push (on the stack)" operation. By designing a zero-address architec-

Circle 127 on inquiry card.



next	JSR/RTS pair
call	JSR instruction
return	RTS instruction

Figure 1: *Diagram of subroutine-threaded code (STC). In this and figures 2 thru 4, the pointer points to the main program being executed. Both A and B are subprograms called by the main program; A is an intermediate-language subprogram of the same type as the main program, and B is an in-line machine-language program that directly executes the machine language of the host computer. The words* next, call, *and* return *refer to operations that must be performed for any threaded-code language. The information to the right of these words tells how each operation is performed in the current type of threaded code.*



next	1. copy current list item to temporary storage 2. point list pointer to next list item 3. jump to machine code at address in temporary storage
call	1. push current list pointer onto stack 2. load list pointer with address of the intermediate-language subroutine list 3. do "next"
return	1. load list pointer with top of stack 2. do "next"

Figure 2: *Diagram of direct-threaded code (DTC). Here, "temporary storage" refers to a memory location that is used to hold the address of the machine-code routine associated with the current unit of code.*

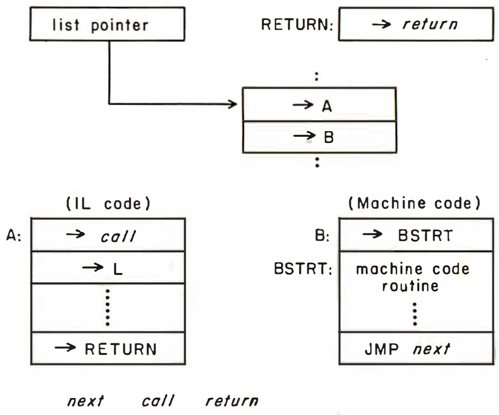
TERMINALS FROM TRANSNET

PURCHASE PLAN	12-24 MONTH FULL OWNERSHIP PLAN			36 MONTH LEASE PLAN
DESCRIPTION	PURCHASE PRICE	PER MONTH 12 MOS.	24 MOS.	36 MOS.
LA36 DECwriter II	$1,695	$162	$ 90	$ 61
LA34 DECwriter IV	1,095	105	59	40
LA34 DECwriter IV Forms Ctrl.	1,295	124	69	47
LA120 DECwriter III KSR ...	2,495	239	140	90
LA180 DECprinter I	2,095	200	117	75
VT100 CRT DECscope	1,895	182	101	68
VT132 CRT DECscope	2,295	220	122	83
DT80/1 DATAMEDIA CRT ...	1,995	191	106	72
TI745 Portable Terminal	1,595	153	85	57
TI765 Bubble Memory Terminal	2,595	249	146	94
TI810 RO Printer	1,895	182	101	68
TI820 KSR Printer	2,195	210	117	79
TI825 KSR Printer	1,595	153	85	57
ADM3A CRT Terminal	875	84	47	32
ADM31 CRT Terminal	1,450	139	78	53
ADM42 CRT Terminal	2,195	210	117	79
QUME Letter Quality KSR ...	3,295	316	176	119
QUME Letter Quality RO	2,895	278	155	105
HAZELTINE 1420 CRT	945	91	51	34
HAZELTINE 1500 CRT	1,195	115	64	43
HAZELTINE 1552 CRT	1,295	124	69	47
Hewlett-Packard 2621A CRT .	1,495	144	80	54
Hewlett-Packard 2621P CRT .	2,650	254	142	96

FULL OWNERSHIP AFTER 12 OR 24 MONTHS
10% PURCHASE OPTION AFTER 36 MONTHS

ACCESSORIES AND PERIPHERAL EQUIPMENT
ACOUSTIC COUPLERS • MODEMS • THERMAL PAPER
RIBBONS • INTERFACE MODULES • FLOPPY DISK UNITS
PROMPT DELIVERY • EFFICIENT SERVICE

TRANSNET CORPORATION
1945 ROUTE 22
UNION, N.J. 07083
201-688-7800
TWX 710-985-5485



next	1. copy current list item to indirect temporary storage 2. point list pointer to next list item 3. load code temporary storage with item at address in indirect temporary storage 4. jump to machine code at address in code temporary storage
call	1. push current list pointer onto stack 2. point indirect temporary storage to next list item 3. load current list pointer from indirect temporary storage 4. do "next"
return	1. load current list pointer from top of stack 2. do "next"

Figure 3: *Diagram of indirect-threaded code (ITC). Here, "indirect temporary storage" and "code temporary storage" store the indirect and direct pointers to the machine code routine associated with the current unit of code.*

Circle 129 on inquiry card.

ture into the intermediate language, the parameter transfer location is implied and need not be part of the intermediate language representation. (A stack architecture is certainly *simpler* than other architectures, but that does not mean it is *better*; many complex trade-offs that are beyond the scope of this article are involved.)

Threaded Code

Threaded code is an intermediate-language implementation technique that organizes the control of program flow into a sequence of subroutine invocations. *No other aspects of the language are represented in threaded code.* Threaded code is especially applicable to interpretation; the interpretation process consists of transferring control to the routines selected by the threaded-code op codes. The functions available in the intermediate language are provided by the subroutines that are invoked and are not an inherent part of the threaded code itself.

[*The characteristics of the language FORTH are independent of its current implementation via threaded code. FORTH enthusiasts often blur the distinction, attributing the language's speed and compactness to the language instead of to its threaded-code implementation. I think this is an important point to remember when talking about the advantages of FORTH....*GW]

Threaded-code intermediate languages are especially applicable to the implementation of virtual machines embodying zero-address architectures. As such, the technique of using threaded code to implement a language can be applied to, for example, Pascal (using the p-code intermediate language), LISP interpreters, or, of course, FORTH. We classify four varieties of threaded code: subroutine, direct, indirect, and token.

All varieties of threaded code consist of a data structure that is a sequence of unique subroutine identifiers. Traditionally, threaded code has been kept close to the machine level and has included actual pointers to the subroutines (which themselves may be either intermediate language or machine code). Also traditionally, a portion of the processor resources—in particular, processor registers—has been dedicated to the use of the threaded-code interpreter. As we shall see, neither absolute pointers nor register resources need be used to implement threaded code.

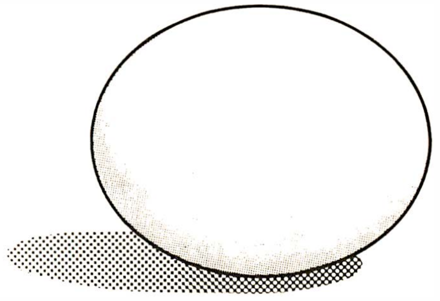
Implementing Threaded Structures

We will now describe the structures associated with the various types of threaded code. Figures 1 through 4 present diagrams of subroutine-, direct-, indirect-, and token-threaded code structures, respectively, along with a description of the three operations, *next*, *call*, and *return*, which make up the complete threaded-code interpreter. In the diagrams, the notation "→A" means a pointer to the memory location labeled "A".

Subroutine-threaded code: A sequence of subroutine calls *with no other embedded instructions* implements an intermediate language. Each subroutine call may be considered a single intermediate-language operation, which need not be related to the underlying machine architecture. Subroutine-threaded code (STC) is a control mechanism that is widely supported at the machine-hardware level.

The peculiar program organization consisting only of

The 2nd Generation is shaping up...



MEASUREMENT
systems & controls
incorporated

MicroByte Software
2415-C Gateway Plaza
Crabtree Blvd.
Raleigh, North Carolina 27604 (919) 833-4094

AT LAST! A fully implemented computer based file management system. . . Only a few minutes of instruction and you are creating and using your own client lists, mailing lists, inventories, bibliographies, vendor lists, and more.

D B M S 8 0

Files, lists, or records, with user defined formats, can be created, sorted, edited, and printed with ease. Sub-files can be created out of parts of existing files, selecting parts of a record or individual records by a search criterion.

ALSO available with DBMS80. . .

R E P O R T 8 0

Build your own custom defined and formatted reports and data summaries. Print lables with user specified formats that will fit your own forms.

DBMS80 and REPORT80 will run under either CP/M or TRSDOS

DBMS80	$250.00
REPORT80	$100.00
Manuals each	$25.00

OTHER PRODUCTS OF MICROBYTE SOFTWARE:

EDIT80 .... **$100.00**
Text editor and print formatter which runs under CP/M or TRSDOS
DISK80 .... **$50.00**
Utility which allows you to examine and patch a disk.
UTILS **$50.00**
Apple PASCAL utilities: extensions to Apple Pascal, together with file control utilities, cross-reference, etc.
PAYROLL **$100.00**
Apple PASCAL payroll for 150 employees, full deduction options, etc.

Write or call today for further details on our products.
Source ID#TCE373

APPLE is a trademark of Apple Computer Corp.
TRSDOS is a trademark of Tandy Corp.
CP/M is a trademark of Digital Research

Owing to a printer error a wrong telephone number was run in August. Our apologies for any inconvenience this caused. Our correct telephone number is **919-833-4094.**

subroutine calls is rarely used by programmers (who have no reason to resist obvious opportunities for optimization), but it is sometimes used by compilers. It is the most general intermediate language possible, and it retains the advantages of machine independence by not generating in-line machine language. (The difference in the form of subroutine call and return instructions on various computers is usually trivial.)

Subroutine-threaded code will incur less execution overhead than most intermediate languages because its interpretation is handled by hardware rather than by a sequence of instructions. Furthermore, subroutine-threaded code can be optimized by using in-line machine code for operations where subroutine overhead is excessive, an advantage unobtainable with other types of threaded code. Of course, the resulting optimized code is no longer machine-independent; the additional translation step converts the intermediate language into object code for a particular machine.

Direct-threaded code: Direct-threaded code (DTC) may be considered a sequence of machine-language subroutine calls with the "call" op code removed. This results in a list of addresses, each of which points to a machine-language subroutine. Since the direct-threaded program includes no op codes, a short machine-language program must be written to read the next address in the list and transfer control to that address. Traditional direct-threaded code implementations do not allow the use of true subroutines at the machine level but instead require that each routine terminate by executing the *next* operation.

In order to call direct-threaded routines (see the instructions for "call" in figure 2), machine-language code (executing the instructions for "call") must be included at the beginning of each direct-threaded routine to put the current value of the list pointer on an address stack, load the list-pointer register with the start address of the list of routine addresses for this just-begun, direct-threaded routine, and execute the *next* operation.

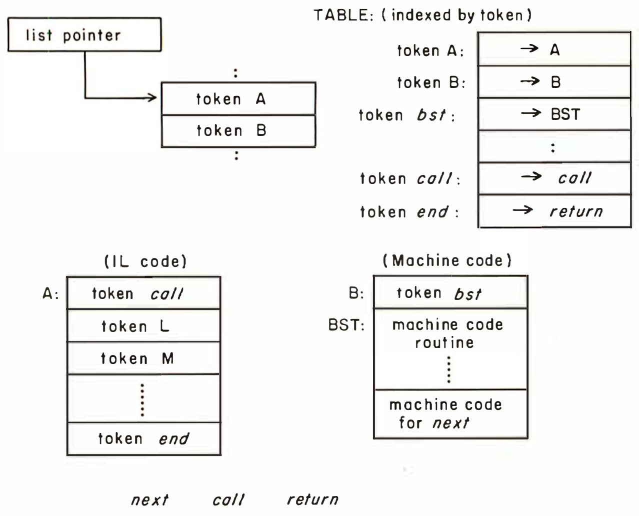
The *next* operation (coded here as in-line machine code) causes the computer to execute the routine pointed to by the list pointer, regardless of whether the routine pointed to is another intermediate-language routine or a machine-language routine.

In order to return to a higher level of nesting, the last list item in an intermediate-language routine points to the code for the *return* operation. When executed by the *next* operation, this operation recovers the previous value of the list pointer from the stack, then executes the *next* operation, which in turn executes the first routine past the routine the computer just returned from.

Thus direct-threaded code is implemented in three operations: *next*, *call*, and *return*.

Indirect-threaded code: Indirect-threaded code (ITC) consists of a list of addresses, but each address points to another address which then points to the machine-code routine. (See figure 3.) As compared to direct-threaded code, in indirect-threaded code, the interpreter must go through an extra level of indirection. Indirect-threaded intermediate-language subroutines do not contain machine-language code for the *call* operation, and one advantage of indirect-threaded code is that a compiler using it need only produce pointers. By manipulating only pointers, the compiler generates intermediate-language code that does not include machine-language code itself; thus it is independent of the target machine. However, a disadvantage of indirect-threaded code is that the interpreter has the overhead of an extra level of indirect addressing.

Token-threaded code: The varieties of threaded code previously mentioned contained pointers that were actual addresses of the subroutines in memory. Using memory addresses to select routines wastes storage because the number of subroutines in the system is far smaller than the number of memory locations. A savings in intermediate-language program size can be obtained by using short tokens to identify the subroutines to be invoked. Typically, token-threaded code (TTC) can be implemented by using the current token to index into a table of subroutine addresses. (See figure 4.)



next
1. get current token from list
2. point list pointer to next list item
3. look up address corresponding to token
4. get indirect token at address
5. look up address corresponding to indirect token
6. jump to machine code at that address

call
1. push current list pointer onto stack
2. load list pointer with start of new list
3. do "next"

return
1. load current list pointer from top of stack
2. do "next"

Figure 4: *Diagram of token-threaded code (TTC). Since tokens can be made shorter than addresses, this makes the threaded code more compact, but the table lookup makes the resulting code slower. Here, the "indirect token" is the contents of the table entry that matches the current token of code.*

High-Level Descriptions of Threaded-Code Interpreters

Listings 1 thru 3 illustrate the logical implementation of direct-, indirect-, and token-threaded code, respectively. The program descriptions are written in a high-level language that is similar in appearance to Pascal. It differs from Pascal in that the variables are not declared as standard Pascal data types. Also, the *next*, *call*, and *return* operations are not written as Pascal procedures; this was done to remain faithful to actual implementations where these three code segments are reached by jump instructions rather than by subroutine calls.

Several other notational conventions used in these listings may also need explanation. The data type *pointer* means an actual machine address. If *ip* is a pointer variable, then → *ip* means the value at the location which is pointed to by the address in variable *ip*. Therefore, the statement

goto → ip;

means jump to a new location using the contents of variable *ip* as the address at which to proceed with execution.

Implementation Concerns

The traditional implementations of threaded-code interpreters have had one or more machine registers dedicated to the exclusive use of the interpreter; implementations on microcomputers have tended to use *all* microprocessor resources. One problem with these implementations is that all machine-language routines (where all real computation is done) must save processor registers before modifying them and must restore them before returning to the interpreter.

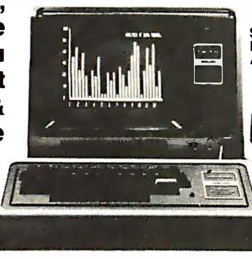
Additionally, this use of machine resources, simply for the transfer of control, obstructs the use of standard machine-language subroutines that pass parameters through the registers. In the context of microcomputer

TRS-80 LEVEL II $685 COMPLETE SYSTEM

Limited quantity in stock

The world's most popular microcomputer, with 16K of memory and Level I I basic for only $685, complete with full 90 day Radio Shack warranty. We accept check, money order or phone orders with Visa or MasterCharge. (Shipping costs added to charge orders).

Disk drives, printers, peripherals, software and games . . . you name it, we've got it (Both Radio Shack & other brands). Write or call for our complete price list.



Shown is Level I. Level II includes Alphanumeric keypad.

C&S ELECTRONICS MART Ltd.

AUTHORIZED DEALERSHIP Radio Shack

32E. Main Street ● Milan Michigan 48160 ● (313)439-1400

Listing 1: *Description of a direct-threaded code interpreter in a Pascal-like language. See figure 2.*

```
const pointer_length = (length of an address pointer);
      call_code_length = (length of "call" code segment);
var   list_pointer: pointer; { interpreted program counter  }
      list_item:  pointer; {  contains threaded-code item   }
label next, call, return;

next: list_item := ^list_pointer;
      list_pointer := list_pointer + pointer_length;
      goto ^list_item;

call: push_on_stack(list_pointer);
    {  The value of list_item was set by the preceding        }
    {       "next" operation.                                 }
      list_pointer := list_item + call_code_length;
    {  The following code duplicates the "next" operation.    }
      list_item := ^list_pointer;
      list_pointer := list_pointer + pointer_length;
      goto ^list_item;

return:  list_pointer := pop_from_stack();
     {     The following code duplicates the "next" operation. }
         list_item := ^list_pointer;
         list_pointer := list-pointer + pointer_length;
         goto ^list_item;
```

Listing 2: *Description of an indirect-threaded code interpreter in a Pascal-like language. See figure 3.*

```
const pointer_length = (length of an address pointer) ;
var   list_pointer: pointer;  { interpreted program counter  }
      list_item: pointer;     { contains threaded-code item   }
      code_pointer: pointer;  { points to actual machine code  }
label next, call, return;

next: list_item := ^list_pointer;
      list_pointer := list_pointer + pointer_length;
      code_pointer := ^list_item;  { here is the extra       }
                                   {   level of indirection  }
      goto ^code_pointer;

call: push_on_stack(list_pointer);
   {   The value of list_item was set by the                 }
   {       preceding "next operation.                        }
      list_pointer := list_item + pointer_length;
   {   The following code duplicates the "next" operation.   }
      list_item := ^list_pointer;
      list_pointer := list_pointer + pointer_length;
      code_pointer := ^list_item;
      goto ^code_pointer;

return:  list_pointer := pop_from_stack();
    {    The following code duplicates the "next" operation.    }
         list_item := ^list_pointer;
         list_pointer := list_pointer + pointer_length;
         code_pointer := ^list_item;
         goto ^code_pointer;
```

Listing 3: *Description of a token-threaded code interpreter in a Pascal-like language. See figure 4.*

```
const token_length = (length of token) ;
      call_code_length  = (length of "call" code segment);
      toknumber = (number of tokens possible); { is 256 for an }
                                               { 8-bit token   }
var  list_pointer: pointer;  { interpreted program counter  }
     code_pointer: pointer;  { pointer to machine code      }
     table: array[1..toknumber] of pointer; {subroutine table }
     token_item: short token;
label next, call, return;

next: token_item := ^list_pointer;
      list_pointer := list_pointer + token_length;
      code_pointer := table[token_item];
      token_item := ^code_pointer;
      code_pointer := table[token_item];
      goto ^code_pointer;

call: push_on_stack(list_pointer);
   {    The value of the code_pointer was set by the preceding }
   {       "next" operation.                                   }
      list_pointer := code_pointer + call_code_length;
   {   The following code duplicates the "next" operation.     }
      token_item := ^list_pointer;
      list_pointer := list_pointer + token_length;
      code_pointer := table[token_item];
      goto ^code_pointer;

return:  list_pointer := pop_from_stack();
    {    The following code duplicates the "next" operation.    }
         token_item := ^list_pointer;
         list_pointer := list_pointer + token_length;
         code_pointer := table[token_item];
         goto ^code_pointer;
```

Circle 132 on inquiry card.

Listing 4: *A simple direct-threaded code interpreter for the MC6809 microprocessor.*

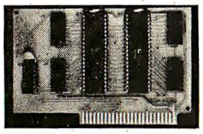
```
        RETURN:  PULS   Y           GET NEW THREAD PTR
                 JMP   [,Y++]       DO "NEXT"


Machl Routine          IL Routine

                CALL:  PSHS  Y
 ______                LEAY  *+7,PCR       STACK OLD THREAD POINTER
 ______                JMP   [,Y++]        ADDR OF FOLLOWING IL CODE
 ______
JMP   [,Y++]           ______
                       ______
                       FDB   RETURN        ADDR OF "RETURN"
```

systems (which may want to use read-only memory modules), this limitation requires that special "header" and "trailer" code be written to move data values used by the intermediate language to and from the registers used by previously written machine-language code.

It is also possible to eliminate the use of processor resources in an intermediate language by storing the interpreter's "registers" in memory; this leaves the processor free for use by machine-language code at the expense of additional overhead during interpretation. [*This overhead consists of having to move these registers between memory and the hardware registers of the host processor when you want to manipulate the contents of the interpreter registers....*GW] The use of absolute locations in memory would itself be a problem, because these locations can then conflict with locations used by other software packages. By saving the intermediate-language registers on the *stack*, the language may be made inde-

MICRO MISCELLANY

APPLE II PARALLEL INTERFACE

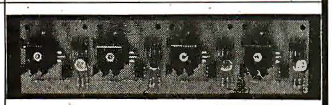


$79.95

Interfaces printers, synthesizers keyboards, and JBE A-D D-A Converter & Switches. This Interface has 4 I/O ports with handshaking logic, 2-6522 VIA's and a 74LS74 for timing. Inputs and outputs are TTL compatible.

79-295K	Complete Kit	$69.95
79-295A	Assembled	$79.95

SOLID STATE SWITCH



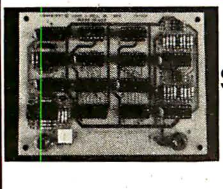
$44.95

$12.50

Your computer can control power (120VAC) to your printer, lights, and other 120VAC appliances up to 720 watts (6AMPS at 120VAC). Input 3 to 15 VDC, 2-13 MA TTL compatible, isolation 1500V.

79-282	1 Channel Kit	$ 9.95
	Assm.	$12.50
79-282	4 Channel Kit	$34.95
	Assm.	$44.95

AtoD DtoA CONVERTER



$69.95

Analog to Digital, Digital to Analog Converter, AtoD conversion time 20us. DtoA conversion 5us. Uses include speech and music synthesizing and slow scan TV. Single power supply (5V), 8 Bits wide, latched I/O, strobe lines.

79-287K	Complete Kit	$49.95
79-287A	Assembled	$69.95

BARE BOARDS

SINGLE BOARD COMPUTERS

8088 5-CHIP SYSTEM	$29.95
8085 3-CHIP SYSTEM	$24.95

MEMORY BOARD

8208 64K DYNAMIC	$39.95

ALL PRODUCTS AVAILABLE FROM:

JOHN BELL ENGINEERING
P.O. Box 338
Dept. 4
Redwood City, CA 94064
(415) 367-1137

Add 6% sales tax in California and $1.00 shipping and handling for orders less than $20. Add 4% for VISA or M.C.

JOHN BELL ENGINEERING

Listing 5: *A simple indirect-threaded code interpreter for the MC6809 microprocessor. In this and listings 6 thru 8, each block of information in lowercase is a "stack picture"—ie: a diagram of what is on the stack at that particular place in the code.*

```
           s ->thread ptr 1
               thread ptr 2

 NEXT:     LEAS   -2,S                  MAKE SPACE
           PSHS   X                     SAVE X

           s ->x
               space
               thread ptr 1
               thread ptr 2

           LDX   [,Y++]                 GET ADDRESS OF ROUTINE
           STX    2,S                   SAVE AS UPCOMING PC

           s ->x
               routine addr
               thread ptr 1
               thread ptr 2


           PULS   X,PC                  RECOVER X AND GO!

           s ->thread ptr 1
               thread ptr 2


 CALL:     PSHS   Y                     SAVE CURRENT THREAD PTR
           LDY    ,--Y                  GET PREVIOUS INDIRECT PTR
           LEAY   2,Y                   NEW THREAD PTR
           BRA    NEXT

RETURN:    PULS   Y                     RECOVER OLD THREAD PTR
           BRA    NEXT
```

Listing 6: *A more complex direct-threaded code interpreter for the MC6809 microprocessor. Execution of the intermediate-language subroutine starts at the label ENTRY.*

```
             s ->next
                 thread ptr 1
                 thread ptr 2

RETURN:      LEAS   2,S           DISCARD "NEXT"
             PULS   Y             GET SAVED THREAD PTR

N1:          BSR    N2            PUSH ADDR OF NEXT
               s ->thread ptr 2
NEXT:        BRA    N1            SET UP RETURN TO NEXT
N2:          JMP   [,Y++]         GO TO ROUTINE

             s -> next
                  thread ptr 2


      I-Code Routine  (start at ENTRY)

            PSHS   X              SAVE X
             s -> x
                 thread ptr 0
                 space
                 next
                 thread ptr 1
                 thread ptr 2
            LDX   6,S             GET ADDR OF "NEXT"
            STX   4,S             MOVE IT
            STY   6,S             SAVE OLD THREAD PTR
             s ->x
                thread ptr 0
                next
                y (old thread ptr)
                thread ptr 1
                thread ptr 2
            PULS   X,Y            RECOVER X, NEW THREAD PTR
            JMP   [,Y++]          DO SIMPLE "NEXT"
ENTRY:      LEAS  -2,S            MAKE SPACE
            BSR   *-14         PUSH NEW THREAD PTR, GOTO PSHS X
    0:      ______                START OF THE IL CODE
            ______
            ______
            FDB   RETURN          ADDR OF "RETURN"
```

Circle 134 on inquiry card.

Software can be written to function properly on widely varying computers that use the same microprocessor.

pendent of particular programmable memory locations.

Another way to eliminate the use of processor resources, as well as maximize throughput, is to use subroutine-threaded code (STC). Subroutine-threaded code makes use of only the program counter and the subroutine return stack, resources already dedicated to the control of program flow. Thus, the processor resources traditionally available to the programmer remain free for use by machine-language code.

Distribution of Software

It is possible to conceive of a mass market for software; such a market would allow high-quality programs to be distributed at low cost. We will assume that such code will be distributed in the form of read-only memory modules, so that a purchaser actually receives a physical product for his money. Furthermore, the memory needed to store the program is included in the purchase price, a characteristic not obtained with distribution on magnetic media. Software piracy will be possible for advanced hobbyists, but these represent only a small portion of the consumer market.

To maximize sales, it is necessary that everyone who has a computer and who wants to use the program be able to do so. Given machine-language distribution, the market is already limited to those users with a particular processor; it should not also be limited to those users with a particular computer system.

Software can be written such that it functions properly on systems that use different locations for programmable memory, read-only memory, and input/output (I/O) devices, as well as systems that use completely different I/O devices. The system-independent read-only memory must be written in code that is position independent, and it must also include features for linking to other similar modules. These criteria can be satisfied with machine-language code (on certain processors) or with a correctly designed intermediate language. Widest distribution requires such properly written code.

Machine-Language Examples of Threaded-Code Interpreters

Here we present assembly-language code for the Motorola MC6809 microprocessor which implements complete interpreters for direct-threaded code, indirect-threaded code, and token-threaded code. Most of these listings are punctuated by "stack pictures" (typed in lowercase) that represent the current state of the stack at various points in the listing; visualization of the stack is often crucial to understanding the interpretive process.

An illustration of subroutine-threaded code (using subroutine jump and return instructions) would be trivial, and thus is not included. However, it should be noted that a position-independent form of subroutine-threaded code is available on computers with long rela-

---

New Produced and widely used in England and U.S.A.

COMPLETE BUSINESS PACKAGE

INCLUDES EVERYTHING FROM INVENTORY TO SALES SUMMARY
PROMPTS USER, VALIDATES EACH ENTRY, MENU DRIVEN

Approximately 60-100 entries/Inputs require only 2-4 hours weekly and your entire business is under control.

PROGRAMS ARE INTEGRATED·	SELECT FUNCTION BY NUMBER·
01 = ENTER NAMES/ADDRESS, ETC	13 = PRINT CUSTOMER STATEMENTS
02 = ENTER/PRINT INVOICES	14 = PRINT SUPPLIER STATEMENTS
03 = ENTER PURCHASES	15 = PRINT AGENT STATEMENTS
04 = ENTER A/C RECEIVABLES	16 = PRINT TAX STATEMENTS
05 = ENTER A/C PAYABLES	17 = PRINT WEEK/MONTH SALES
06 = ENTER/UPDATE INVENTORY	18 = PRINT WEEK/MONTH PURCHASES
07 = ENTER/UPDATE ORDERS	19 = PRINT YEAR AUDIT
08 = ENTER/UPDATE BANKS	20 = PRINT PROFIT/LOSS ACCOUNT
09 = EXAMINE/MONITOR SALES LEDGER	21 = UPDATE END MONTH FILES MAINTENANCE
10 = EXAMINE/MONITOR PURCHASE LEDGER	22 = PRINT CASH FLOW FORECAST
11 = EXAMINE/MONITOR (INCOMPLETE RECORDS)	23 = ENTER/UPDATE PAYROLL (NOT YET AVAILABLE)
12 = EXAMINE PRODUCT SALES	24 = RETURN TO BASIC

WHICH ONE? (ENTER 1-24)
01 SUB. MENU EXAMPLE: 01 = EXAMINE: 02 = INSERT: 03 = AMEND: 04 = DELETE
05 = PRINT (1,2,3): 06 = NUMERIC COMBINATIONS: 07 = SORT
VERY FLEXIBLE. ADD YOUR OWN FUNCTIONS. EASY TO INTEGRATE.
All programs in BASIC for CP/M. PET. 6800

G. W. COMPUTERS LTD, the producers of this beautiful package in U.K.

WE EXPORT TO ALL COUNTRIES:
BARCLAYCARD ACCEPTED
CBM APPROVED

CP/M Ver. 9.00 is one 16 K core program using random access releasing both drives for data storage, and 250 word vocabulary is translatable in any foreign language.

CALLERS BY APPOINTMENT ONLY
89 Bedford Court Mansions
Bedford Avenue
London WC1, U.K.

CONTACT TONY WINTER 01-636-8210
BARCLAYCARD ACCEPTED
CBM APPROVED

CP/M Ver. 9.00 is one 16 K core program using random access releasing both drives for data storage, and 250 word vocabulary is translatable in any foreign language.

PRICES: Programs 1-23 EXC (19,20,22,23) £475 £575 Stock Integrated Option + £100 Bank Integrated Option + £100

tive branch instructions (eg: the LBSR, long branch-to-subroutine, and RTS, return-from-subroutine, instructions on the MC6809).

Listing 4 illustrates a very simple implementation of a direct-threaded code interpreter. This particular implementation is very fast, but it has the following undesirable properties:

- it requires a special machine-language return instruction (ie: JMP [,Y++]);
- it reserves the Y register for use by the interpreter;
- it requires that the interpreter location (the address of RETURN) be known to the compiler, making the resulting intermediate-language code definitely position-dependent.

In operation, the Y register points to the next address in a direct-threaded code list; that address, of course, points directly to machine code. Executing the operation JMP [,Y++] (indirect, autoincrement by 2) causes the machine to start execution at the address contained in the list element; simultaneously, the Y register is updated to point at the next item in the list of addresses.

The single instruction JMP [,Y++] ends each machine-language subroutine. By reserving a processor register for use as the current thread pointer, a speed advantage is obtained; transfer of control using JMP [,Y++] requires nine machine cycles (on the MC6809), while a JSR-RTS pair requires thirteen.

The situation becomes more complex when control is transferred to a subroutine composed of intermediate-language statements. Machine-language instructions are included at the beginning of the intermediate-language subroutine to perform the *call* operation. The Y register may be thought of as the topmost location of the stack of intermediate-language return addresses; its contents are pushed onto the stack, and Y is loaded with the address of the start of the intermediate-language subroutine list.

The last item in an intermediate language list is the address of the *return* routine. This recovers an old intermediate-language pointer from the stack and continues interpretation where it left off when it did a subroutine call.

In listing 5, we show a very simple indirect-threaded code interpreter. As in the previous example, the interpretation process is fast, but again it has the following limitations:

- it must use a position-dependent, machine-language return instruction (eg: JMP NEXT);
- it uses the Y register to hold the list pointer;
- it still requires that the compiler generate position-dependent pointers to the CALL and RETURN routines.

Listing 6 is an example of a moderately complex direct-threaded code interpreter. It is somewhat slower than the simple interpreter in listing 4, but it uses a standard RTS instruction to return from machine-language routines. Thus, the machine-language routines need not contain pointers to the *next* operation. Still, this advantage is bought at the expense of additional machine-language code in each intermediate-language subroutine. The intermediate-language subroutines themselves do have

Listing 7: *An improved direct-threaded code interpreter for the MC6809 microprocessor. This interpreter does not use any of the microprocessor registers.*

```
         s ->ptr to new thread
             addr of "next"
             old thread ptr

 CALL:   PSHS D            SAVE D
         LDD 2,S           GET NEW PTR
         STD 4,S           THREAD PTR

         s ->d
             space
             new thread ptr
             old thread ptr

         PULS D            RECOVER D
         LEAS 2,S          DELETE SPACE
 NEXT:   LEAS -4,S         MORE SPACE

         s ->space
             space
             thread ptr

RETURN:  PSHS X,D          SAVE X, D

         s ->d
             x
             space
             space
             thread ptr

         LDX 8,S           GET THREAD PTR
         LDD ,X++          GET NEXT MACHL ADDR
         STX 8,S           STACK THREAD PTR
         STD 4,S           STACK ROUTINE ADDR
         LEAX NEXT,PCR     GET ADDR OF "NEXT"
         STX 6,S           SAVE AS MACHL RETURN

         s ->d
             x
             machl routine
             addr of "next"
             thread ptr

         PULS D,X,PC       GO TO MACHL ROUTINE

         s ->addr of "next"
             thread ptr


 I-CODE:  JSR CALL <instl> ... <RETURN>
```

Listing 8: *Token-indirect token-threaded interpreter for the MC6809 microprocessor. Because of the use of two levels of lookup, this interpreter is completely position independent.*

```
        s -> table addr
             old indirect
             thread ptr

NEXT:   LEAS  -4,S        MAKE FREE STACK SPACE
        PSHS  U,X,D       SAVE REGISTERS

        s -> d
             x
             u
             space
             space
             table addr
             indirect
             thread ptr
```

Listing 8 continued on page 222

BUSINESS - PROFESSIONAL - GAME
SOFTWARE FOR APPLE AND TRS-80

☐ **HOME FINANCE PAK I: Complete package $49.95 Apple, TRS-80**

☐ BUDGET: The heart of a comprehensive home finance system. Allows user to define up to 20 budget items. Actual expense input can be by keyboard or by automatic reading of CHECKBOOK II files. Costs are automatically sorted and compared with budget. BUDGET produces both monthly actual/budget/variance report and a year-to-date by month summary of actual costs. Color graphics display of expenses. . . $24.95

☐ CHECKBOOK II: This extensive program keeps complete records of each check/deposit. Unique check entry system allows user to set up common check purpose and recipient categories. Upon entry you select from this pre-defined menue to minimize keying in a lot of data. Unique names can also be stored for completeness. Rapid access to check files. Check register display scrolls for ease of review. 40 column printout. Up to 100 checks per month storage. Files accessible by BUDGET program $19.95

☐ SAVINGS: Allows user to keep track of deposits/withdrawals for up to 10 savings accounts. Cpmplete records shown via screen or 40 column printer. $14.95

☐ CREDIT CARD: Keep control of your cards with this program. Organizes, stores and displays purchases, payments and service charges. Screen or 40 column printer display. Up to 10 separate cards $14.95

☐ **THE UNIVERSAL COMPUTING MACHINE: $39.95 Apple, TRS-80**
A user programmable computing system structured around a 20 row x 20 column table. User defines row and column names and equations forming a unique computing machine. Table elements can be multiplied, divided, subtracted or added to any other element. User can define repeated functions common to a row or column greatly simplifying table setup. Hundreds of unique computing machines can be defined, used, stored and recalled, with or without old data, for later use. Excellent for sales forecasts, engineering design analysis, budgets, inventory lists, income statements, production planning, project cost estimates-in short for any planning, analysis or reporting problem that can be solved with a table. Unique curser commands allow you to move to any element, change its value and immediately see the effect on other table values. Entire table can be printed by machine pages (user-defined 3-5 columns) on a 40 column printer. Transform your computer into a UNIVERSAL COMPUTING MACHINE.

☐ **COLOR CALENDAR:** HI-RES color graphics display of your personal calendar. Automatic multiple entry of repetitive events. Review at a glance important dates, appointments, anniversaries, birthdays, action dates, etc. over a 5 year period. Graphic calendar marks dates. Printer and screen display a summary report by month of your full text describing each day's action item or event. Ideal for anyone with a busy calendar . . (Apple Only) $19.95

☐ **BUSINESS SOFTWARE SERIES: Entire package $199.95 Apple, TRS-80**

☐ MICROACCOUNTANT: The ideal system for the small cash business. Based on classic T-accounts and double-entry bookkeeping, this efficient program records and produces reports on account balances, general ledger journals, revenue and expenses. Screen or 40 column printer reports. Handles up to 500 journal entries per period, up to 100 accounts. Instructions include a short primer in Financial Accounting. $49.95

☐ UNIVERSAL BUSINESS MACHINE: This program is designed to SIMPLIFY and SAVE TIME for the serious businessman who must periodically Analyze, Plan and Estimate. The program was created using our Universal Computing Machine and it is programmed to provide the following planning and forecasting tools.

CASH FLOW ANALYSIS	PROFORMA BALANCE SHEET	SOURCE AND USE OF FUNOS
PROFORMA PROFIT & LOSS	SALES FORECASTER	JOB COST ESTIMATOR

Price, including documentation and a copy of the base program. Universal Computing Machine $89.95

☐ INVOICE: Throw away your pens. Use the ELECTRONIC INVOICE facsimile displayed on your CRT. The program prompts and you fill in the data. Includes 3 address fields (yours, Bill to and Ship to), Invoice No., Account No., Order No., Salesman, Terms, Ship Code, FOB Pt. and Date. Up to 10 items per sheet with these descriptions: Item No., No. of units, Unit Price, Product Code, Product Description, Total Dollar amount per item and invoice total dollar amount. Generates, at your option, hard copy invoices, shipping memos, mailing labels, audit copies and disc updates to master A/R files. (48K) $49.95

☐ BUSINESS CHECK REGISTER: Expanded version of the Checkbook II program. Handles up to 500 checks per month with complete record keeping. (48K). $29.95

☐ BUSINESS BUDGET: As described above and companion program to Business Check Reuister. Handles 500 transactions per month, up to 20 cost categories. Accesses BCR files for actual costs. (48K) $29.95

☐ **ELECTRICAL ENGINEERING SERIES: Both programs $159.95 Apple**

☐ LOGIC SIMULATOR: SAVE TIME AND MONEY. Simulate your digital logic circuits before you build them. CMOS, TTL, or whatever, if it's digital logic, this program can handle it. The program is an interactive, menu driven, full-fledged logic simulator capable of simulating the bit-time by bit-time response of a logic network to user-specified input patterns. It will handle up to 1000 gates, including NANOS, NORS, INverters, FLIP-FLOPS, SHIFT REGISTERS, COUNTERS and user-defined MACROS. Up to 40 user-defined, random, or binary input patterns. Simulation results displayed on CRT or printer. Accepts network descriptions from keyboard or from LOGIC DESIGNER for simulation. Specify 1000 gate version (48K required) or 500 gate version (32K required) $89.95

☐ LOGIC DESIGNER: Interactive HI-RES Graphics program for designing digital Ingic systems. A menu driven series of keyboard commands allows you to draw directly on the screen up to 15 different gate types, including 10 gate shape patterns supplied with the program and 5 reserved for user specification. Standard patterns supplied are NAND, NOR, INVERTER, EX-OR, T-FLOP, JK-FLOP, D-FLOP, RS-FLOP, 4 Bit COUNTER and N-BIT SHIFT REGISTER. User interconnects gates just as you would normally draw using line graphics commands. Network descriptions for LOGIC SIMULATOR generated simultaneously with the CRT diagram being drawn. Drawing is done in pages of up to 20 gates. Up to 50 pages (10 per disc) can be drawn, saved and recalled. Specify 1000 gate (48K) or 500 gate (32K) system $89.95

☐ **MATHEMATICS SERIES: Complete Package $49.95 Apple only**

☐ NUMERICAL ANALYSIS: HI-RES 2-Dimensional plot of any function. Automatic scaling. At your option, the program will plot the function, plot the INTEGRAL, plot the DERIVATIVE, determine the ROOTS, find the MAXIMA and MINIMA and list the INTEGRAL VALUE. For 16K $19.95

☐ MATRIX: A general purpose, menu driven program for determining the INVERSE and DETERMINANT of any matrix, as well as the SOLUTION to any set of SIMULTANEOUS LINEAR EQUATIONS. Disk I/O for data save. Specify 55 eqn. set (48K) or 35 eqn. (32K) $19.95

☐ 3-D SURFACE PLOTTER: Explore the ELEGANCE and BEAUTY of MATHEMATICS by creating HI-RES PLOTS of 3-dimensional surfaces from any 3-variable equation. Disc save and recall routines for plots. Menu driven to vary surface parameters. Demos include BLACK HOLE gravitational curvature equations. . $19.95

☐ **ACTION ADVENTURE GAMES SERIES: Entire series $29.95 Apple only**

☐ REO BARON: Can you outfly the REO BARON? This fast action game simulates a machine-gun DOG-FIGHT between your WORLD WAR I BI-PLANE and the baron's. You can LOOP, DIVE, BANK or CLIMB in any one of 8 directions - and so can the BARON. in HI-RES graphics $14.95

☐ BATTLE OF MIDWAY: You are in command of the U.S.S. HORNETS' DIVE-BOMBER squadron. Your targets are the Aircraft carriers, Akagi, Soryu and Kaga. You must fly your way through ZEROS and AA FIRE to make your DIVE-BOMB run. In HI-RES graphics $14.95

☐ SUB ATTACK: It's April, 1943. The enemy convoy is headed for the CORAL SEA. Your sub, the MORAY, has just sighted the CARRIERS and BATTLESHIPS. Easy pickings. But watch out for the DESTROYERS - they're fast and deadly. In HI-RES graphics $14.95

☐ FREE CATALOG—All programs are supplied in disc and run on Apple II w/Disc & Applesoft ROM Card & TRS-80 Level II and require 32K RAM unless otherwise noted. Detailed instructions included. Orders shipped within 3 days. Card users include card number. Add $1.50 postage and handling with each order. California residents add 6½% sales tax. Make checks payable to:

VISA

SPECTRUM SOFTWARE

master charge

DEALER INQUIRIES INVITED

P.O. BOX 2084 - 142 CARLOW, SUNNYVALE, CA 94087
FOR PHONE ORDERS – 408-738-4387

Listing 8 continued:

```
LDU   10,S          GET TABLE ADDR
LDX   14,S          GET THREAD PTR

LDB   ,X+           GET INDIRECT TOKEN
STX   14,S          SAVE THREAD PTR
CLRA                :
ASLB                : TWO BYTES PER TOKEN
ROLA                :
LDX   D,U           TABLE-RELATIVE INDIRECT PTR
ADDD  4,S           NOW ABSOLUTE
TFR   D,X

LDB   ,X+           GET TOKEN
STX   12,S          SAVE INDIRECT PTR
CLRA
ASLB
ROLA
LDD   D,U           TABLE-RELATIVE MACHL ADDR
ADD   4,S           NOW ABSOLUTE
TFR   D,X

STX   6,S           SAVE AS UPCOMING PC
LEAX  NEXT, PCR     ADDR OF NEXT
STX   8,S           SAVE FOR MACHL RTS
PULS  D,X,U, PC     RECOVER REGS + GO!


s -> addr of "next"
     table addr
     indirect
     thread ptr

  CALL:    PSHS   D          SAVE D

           s -> d
                addr of "next"
                table addr
                indirect
                thread ptr

           LDD   4,S         GET TABLE ADDR
           STD   2,S         MOVE IT
           PULS  D           RECOVER D
           BRA   NEXT

RETURN:    PSHS   D          SAVE D

           s -> d
                addr of "next"
                table addr
                old indirect
                thread ptr 1
                thread ptr 2

           LDD   4,S         GET TABLE ADDR
           STD   6,S         MOVE IT
           LDD   0,S         RECOVER D
           LEAS  6,S         DISCARD JUNK
           BRA   NEXT
```

pointers to the *return* operation, of course (making the code position-dependent), and the interpreter reserves the Y register for its own use.

Listing 7 illustrates a direct-threaded code interpreter that does not reserve any processor registers; this interpreter also allows the return from machine-language routines by means of a standard RTS instruction. The absolute locations of the interpreter *call* and *return* routines must be included in each direct-threaded code subroutine; this usually precludes the distribution of such subroutines in read-only memory.

Circle 138 on inquiry card.

Type of Threaded Code	MC6809 Machine Cycles Used	Ratio of Cycles Used	Relative Size of Resulting Intermediate-Language Code	Can this Code Be Marketed to All Users of a Given Microprocessor?
Subroutine-threaded code	91	1.0	3	no
Relative subroutine-threaded code	98	1.1	3	yes
Simple direct-threaded code (listing 4)	93	1.1	2	no
Simple indirect-threaded code (as in listing 5)	371	4.1	2	no
Moderately complex direct-threaded code (as in listing 6)	228	2.5	2	no
Improved direct-threaded code (as in listing 7)	552	6.1	2	no
Token-threaded code (as in listing 8)	1083	11.9	1	yes

Table 1: *Comparison of threaded-code techniques. Notice that only two forms of threaded code, the relative subroutine-threaded code and the token-indirect token-threaded code are sufficiently system-independent to be used for mass distribution to (potentially) all users of a given microprocessor.*

A possible alternative would be to modify the direct-threaded code interpreter in listing 7 to use strictly self-relative pointers. Then by including code for *call* and *return* in each read-only memory device, a form of distributable direct-threaded code might be obtained. However, because the read-only memory still contains machine-dependent code, the use of direct-threaded code in a read-only memory environment offers little advantage.

The improved direct-threaded code interpreter allows the use of most previously coded machine-language modules and allows these routines to pass parameters through the processor registers. Routines cannot pass parameters on the hardware stack (which is used to maintain the state of the interpreter), but could easily use the user stack of the MC6809 microprocessor for parameter transfer.

A similarly improved interpreter could be built for indirect-threaded code, but the position-independence problem is inherent in this intermediate language as well. Each indirect-threaded subroutine must include a pointer to the *call* routine, thus making the resulting intermediate-language code unsuitable for distribution in read-only memory.

However, it *is* possible to build a token-thread interpreter that has a completely position-independent intermediate-language representation. Listing 8 shows one implementation that achieves these goals. Notice the increased complexity and overhead when compared to our original direct-threaded code interpreter.

This token-thread interpreter produces intermediate-language code that is more compact than that produced by previously mentioned interpreters. The advantage of a compact representation need not affect execution speed severely; remember that the overall efficiency of any interpretation scheme (including the hardware interpretation of op codes) depends more upon the work actually accomplished than the time spent in the interpretation process itself.

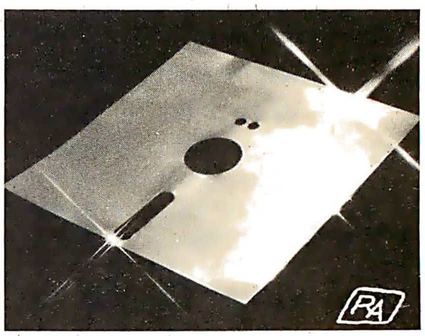
This particular implementation is essentially a token-indirect token-thread interpreter. Two levels of token lookup are involved so that neither machine-language nor absolute addresses need be included as part of the intermediate-language subroutine. Of course, perhaps

$ GOLD DISK $

CP/M® Compatible Z-80 Disassembler

- RECREATES Z-80 ASSEMBLY LANGUAGE SOURCE FILES FROM ABSOLUTE CODE (.COM FILES) FOR ALTERATION.
- FEATURES MNEMONIC LABELS FOR EASY PROGRAM TRACING.
- INCLUDES COMPLETE DOCUMENTATION AND FREE UTILITY FOR SPECIFYING AND DECODING ASCII SECTIONS OF CODE.
- OPERATES UNDER MINIMUM CP/M* CONFIGURATION (16K RAM).
- DOCUMENTATION ONLY: $12 (MAY BE APPLIED TO DISK ORDER).

$62.00 POSTPAID



"WORTH ITS WEIGHT IN GOLD"

ONE DAY SERVICE FOR CREDIT CARD CUSTOMERS: ORDER DISK BY PHONE FREE! (WE WILL PAY YOU BACK FOR THE PHONE CALL)

CALIF. RESIDENTS ADD 6 % SALES TAX.

VISA

BOWER-STEWART & ASSOCIATES
P.O. BOX 1389
HAWTHORNE, CA. 90250
(213) 676-5055

master charge

SPECIFY DRIVE AND SYSTEM
AVAILABLE ON 5¼" OR 8" IBM SS/SD DISK
*CP/M IS A TRADEMARK OF DIGITAL RESEARCH

Circle 140 on inquiry card.

other, more advantageous forms of token-threaded code interpreters are possible. However, we have shown that there is no longer a question whether position-independent threaded code is possible; now the question is: "at what cost?"

The Cost of Implementation

The claims made for threaded-code techniques in an intermediate-language implementation include reduced program storage and high speed of execution. Unfortunately, these claims are justified only in certain limited contexts. The original implementations of threaded code, which occurred on the Digital Equipment Corporation PDP-11, made use of the instruction JMP @(Rn)+ ; this instruction jumps through a memory pointer while retaining the location of *next* in a register. This is equivalent to the MC6809 instruction JMP [,r++] .

The instruction JMP @(Rn)+ does not save a return address on a memory stack and thus is faster than a JSR instruction. In the environment of a single intermediate-language program that calls only machine-language subroutines, stacking and unstacking of the return address need not occur. Of course, when intermediate-language programs call intermediate-language subroutines, such stacking must occur in a process that will take *longer* than a normal JSR. Thus, for maximum speed, the threaded-code intermediate-language program should not call intermediate-language subroutines.

On the other hand, the instruction JMP @(Rn)+ does eliminate the in-line 16-bit JSR op code for a 50% code reduction (on the PDP-11). But the 50% code reduction achieved on the PDP-11 (which uses a 16-bit JSR op code) is only a 33% code reduction on most microcomputers, which have 8-bit JSR op codes. (The LBSR instruction can be used in the case of the MC6809.) And if the motivation for threaded code is reduction of the intermediate-language code size, token-threaded code implementations can improve the storage efficiency by *another* 50%.

The two traditional forms of threaded code (direct and indirect) are optimized for the environment of a particular computer architecture that is represented by the PDP-11 (and also reflected in the MC6809). Consequently, many microcomputer threaded-code implementations have provided neither maximum code efficiency nor maximum speed and have devoured virtually all of the machine-level microprocessor resources. Comparisons of the four types of threaded code demonstrate that it is unlikely that the speed and code-efficiency maxima will ever coincide.

The main factor affecting code compaction is the use of subroutines instead of in-line code; but the use of subroutines inherently increases interpretation overhead. Since all methods of threaded-code implementation allow the use of subroutines, effects due to the use of subroutines can be disregarded and the efficiency of the implementation methods can be compared directly. Table 1 shows this comparison with values from the machine-language routines developed earlier (based on six *next* operations for each *call* and *return* operation).

Conclusions

Languages that have been historically associated with threaded code will probably continue to use these techniques when implemented on microcomputers. New implementations should take advantage of the interpretive nature of threaded code to provide extensive debugging facilities. However, there is no excuse for the threaded-code implementor to prohibit the use of previously coded machine-language modules by eliminating parameter passage through microprocessor registers. Either the interpreter can be designed to keep these registers free, or special routines must be written by the implementor to save and restore these registers when using library routines stored in read-only memory.

Similarly, the motivation for distributing software in an open market (to many different users with many different systems) leads directly to the requirement for position independence. While the MC6809 directly supports position-independent code at the machine-language level, it is also possible to devise threaded-code intermediate languages that are position independent. But any intermediate language or interpreter that requires particular absolute storage locations is so obnoxious as to be unworthy of discussion in polite programming society. Absolute-address storage requirements are simply unacceptable in code written for mass distribution.

Within these constraints, the various forms of threaded code offer different trade-offs of speed and code efficiency. Because these forms are logically equivalent, a single compiler could be used to generate any of them at the user's choice. Thus, without changing the source program, a threaded-code technique could be selected that would give the desired trade-off between speed and code efficiency for a particular situation.

In the end, threaded-code implementation techniques

Computers & Gambling Magazine*

PRESENTS:

PROBABILITY HANDICAPPING DEVICE I

A 16K BASIC PROGRAM FOR:

HORSE RACE HANDICAPPING!

This amazing program was written by a professional software consultant to TRW Space Systems and is being introduced by the publishers of Computers and Gambling Products Magazine. "PHD-1" is a large complex basic program requiring a full 16K. It is carefully human factored for easy use. PHD-1 is a comprehensive horse racing systam for spotting overlays in thoroughbred sprint races (less than 1 mile). You simply sit down with your computer and the Racing Form the night before the race and answer 5 or 6 questions about each horse's past performance. Your computer then accurately predicts the win probability and odds-line for each horse allowing you to spot overlaid horses while at the track. The users manual contains a complete explanation of overlay betting.

Statistics for thousands of horses were used to develop this handicapping system. The appendix of the manual contains a detailed tab run of a 100 consecutive race system workout showing an **amazing 45% positive return** (45¢ for each $1.00 wagered). A graph is also included showing PHD-1's close fit to the ideal predicted probability vs. actual win percentage curve.

This program features: ☐ Win probability and odds for each horse ☐ Verification display of each horse's parameters prior to entry for easy error correction ☐ Bubble-sort routine for final display ☐ Facility for line printer output ☐ Cassette ARCHIVE routine to store PHD-1's output for later analysis ☐ Complete users manual.

The user's manual may be ordered separately for your perusal for $7.95 and will be credited if you purchase PHD-1.

Order now to get on our list and receive back issues FREE!

PHD-1 User's Manual and 16K Cassette for: Apple II Applesoft, Challenger (Specify Type), TRS-80† Level II $29.95

Ca. res. add 6%

Make checks payable to: **JOE COMPUTER**
22713 Ventura Blvd., Suite F, Woodland Hills, CA 91364

*BE A WINNER: Get on the Computers and Gambling Products Magazine mailing list for $3.00 and receive available back issues.

† *TRS-80 is a registered trademark of Tandy Corporation.*

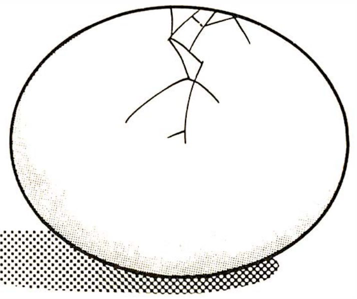
Circle 143 on inquiry card.

are neither particularly compact nor are they particularly fast. Continued development of direct-threaded code structures could result in a language representation that would look more like Pascal p-code than threaded code. Threaded code does offer a conceptually simple and general control-transfer technique that displays a clear boundary between interpretation and language. However, threaded code is probably not an optimal representation for any particular language, including FORTH. ■

Bibliography

1) Bartholdi, P, *Stepwise Development and Debuging (sic) Using a Small Well Structured Interactive Language for Data Acquisition and Instrument Control.* Copy received from the author. (Author's address: Observatoire De Geneve, CH-1290-Sauverny, Switzerland.)
2) Bell, James R, "Threaded Code," *Communications of the ACM,* volume 16, number 6, June 1973, pages 370 thru 372.
3) Dewar, Robert K, "Indirect Threaded Code," *Communications of the ACM,* volume 18, number 6, June 1975, pages 330 thru 331.
4) Dewar, Robert K and A P McCann, "MACRO SPITBOL—A SNOBOL4 Compiler," *Software Practice and Experience,* volume 7, number 1, 1977, pages 95 thru 113.
5) *fig-FORTH Installation Manual,* FORTH Interest Group, San Carlos CA, May 1979.
6) *FORTH Dimensions,* volume 1, numbers 1 to 4, FORTH Interest Group, San Carlos CA.
7) Gaebler, Robert F, "Make it Natural," *Electronics,* volume 52, number 14, July 5, 1979, page 6.
8) Grappel, Robert D, "STRUBAL vs FORTH," *Dr Dobb's Journal,* volume 3, number 8, September 1978, page 28.
9) Brinch Hansen, Per and C Heyden, "Microcomputer Comparison," *Software Practice and Experience,* volume 9, number 3, 1979, pages 211 thru 217.
10) James, John S, "FORTH Dump Programs," *Dr Dobb's Journal,* volume 3, number 28, September 1978, pages 26 thru 27.
11) James, John, "FORTH for Microcomputers," *Dr Dobb's Journal,* volume 3, number 25, May 1978, pages 26 thru 27.
12) Main, Richard B, "FORTH vs Assembly," *Dr Dobb's Journal,* volume 4, number 31, January 1979, pages 45 thru 47.
13) Meinzer, Karl, "IPS, An Unorthodox High Level Language," January 1979 BYTE, volume 4, number 1, pages 146 thru 159.
14) *MicroFORTH Primer.* FORTH Inc, Hermosa Beach CA, December 1976.
15) Moore, Charles H, "FORTH: A New Way to Program a Mini-Computer," *Astronomical Astrophysics Supplement,* volume 15, 1974, pages 497 thru 511.
16) Moore, Charles H, and Elizabeth D Rather, "The Use of FORTH in Process Control," *Proceedings of the International Mini-Micro Computer Conference,* Geneva, March 26, 1977.
17) Oliver, John P, "Astronomy Application for PET FORTH," *Dr Dobb's Journal,* volume 3, number 30, November and December 1978, page 46.
18) Phillips, J B, M F Burke, and G S Wilson, "Threaded Code for Laboratory Computers," *Software—Practice and Experience,* volume 8, 1978, pages 257 thru 263.
19) Rather, Elizabeth D, and Charles H Moore, "The FORTH Approach to Operating Systems," *ACM '76 Proceedings,* October 1976, pages 233 thru 240.
20) Rawson, Edward B, "Let it Be," *Electronics,* February 14 1980, volume 52, number 4, page 8.
21) Ritter, Terry F, and Joel Boney, "A Microprocessor for the Revolution: The 6809—Part 1: Design Philosophy," January 1979 BYTE, volume 4, number 1, pages 14 thru 42; "Part 2: Instruction Set Dead Ends, Old Trails and Apologies," February 1979 BYTE, volume 4, number 2, pages 32 thru 42; "Part 3: Final Thoughts," March 1979 BYTE, volume 4, number 3, pages 46 thru 52.
22) Roichel, Ancelme, "SAM76-FORTH-STRUBAL," *Dr Dobb's Journal,* volume 3, number 30, November and December 1978, pages 44 thru 45.
23) Sachs, John, *STOIC (Stack Oriented Interactive Compiler),* MIT and Harvard Biomedical Engineering Center, Cambridge MA, 1977.
24) Sirag, David J, "DTC Versus ITC for FORTH on the PDP-11," *FORTH Dimensions,* volume 1, number 3, June and July 1978, pages 25 thru 29.

The 2nd Generation...
It's all that it's Cracked up to be.



MEASUREMENT
systems & controls
incorporated

CP/M® USERS!
The ED-80 TEXT EDITOR

- $50,000 in Development Costs — Yours for Only $99!
- For all CP/M, Cromemco, TRS-80 Mod II, and North Star Systems.
- Full Screen Text Editor w/Scrolling.
- For all CRT and Video Monitors.
- Features Found only on IBM, CDC, UNIVAC and DEC Systems.
- Forward or Backward Locate and Change Commands.
- Field Proven — More than 2 Years.

A Terrific Value — $99.00 **Write for FREE Color Brochure**

Software Development & Training, Inc.
Post Office Box 4511, Dept. B
Huntsville, Alabama 35802
VISA or MC