Indirect Threaded Code

Article *in* Communications of the ACM · June 1975

DOI: 10.1145/360825.360849 · Source: DBLP

Programming Techniques
G. Manacher Editor

Indirect Threaded Code

Robert B.K. Dewar
Illinois Institute of Technology

An efficient arrangement for interpretive code is described. It is related to Bell's notion of threaded code but requires less space and is more amenable to machine independent implementations.

Key Words and Phrases: threaded code, SNOBOL4, interpretors, code generation
CR Categories: 4.12, 4.13

A machine independent version of SPITBOL [1], a fast SNOBOL4 system, is currently being implemented. The interpretative arrangement of code used in this compiler involves a concept which is of general applicability to code generation in a wide range of languages. Although developed independently, it has a relation to the notion of threaded code [2], and thus we have used the name Indirect Threaded Code (ITC). We will refer to the original scheme as Direct Threaded Code (DTC)

DTC involves the generation of code consisting of a linear list of addresses of routines to be executed. Some of these routines are standard library routines, e.g. the routine for integer addition. The routines for operand access are specific to a program and must be generated as part of the compiler output. Figure 1 shows an example.

Author's address: Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616.

Fig. 1. Example of Direct Threaded Code.

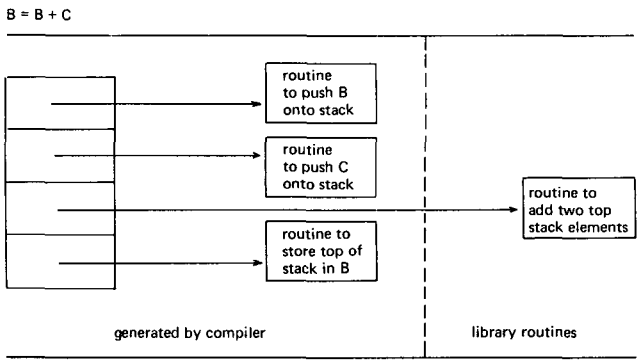

Fig. 2. Example of Indirect Threaded Code.

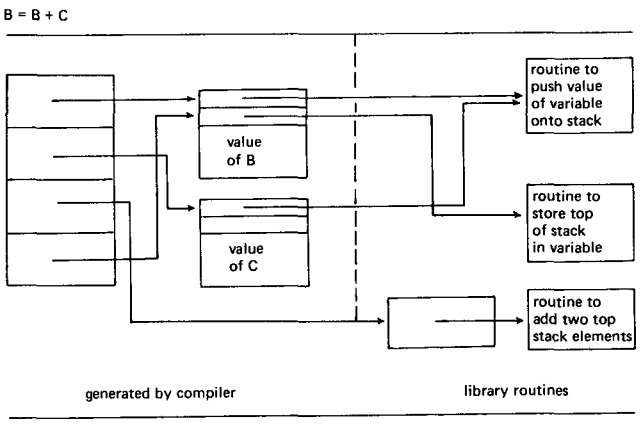

Fig. 3. PDP-11 code generation for B + C. Timing (excluding add)—DTC: 16 cycles; ITC: 13 cycles.

```
DTC
        $P0001  ; push B                    $ADR .
        $P0002  ; push C                        .
        $ADR    ; add                           .
                                                JMP @(R4)+
        $P0001: MOV #A+4,R0
                BR  $F0001
        $P0002: MOV #B+4,R0
                BR  $F0001

        $F0001: MOV -(R0),-(SP)
                MOV -(R0),-(SP)
                JMP @(R4)+

        B:      0   ;value of B
                0
        C:      0   ;value of C
                0

ITC
        B+4     ; push B            $Push   MOV -(R0),-(SP)
        C+4     ; push C                    MOV -(R0),-(SP)
        $ADR    ; add                       MOV (R4)+,(R0)
                                            JMP @(R0)
        B:      0   ;value of B
                0                   $ADR    .+2
                $PUSH
                0   ;value of C             MOV (R4)+,(R0)
        C:      0                           JMP @(R0)
                $PUSH
```

generated code — library routines

Space used for generate coded—DTC: 13 words (excluding $F0001); ITC: 9 words.
Note: The PDP-11 is a byte addressed machine, hence the offset of 4 past the two-word value. The value is placed at the start of the block (not the beginning as in Figure 2) because of peculiarities of addressing on the PDP-11.

ITC consists of a linear list of addresses of words which contain addresses of routines to be executed. This level of indirection is illustrated by the example shown in Figure 2.

Comparing Figures 1 and 2 shows an important qualitative difference. ITC does not involve the compilation of any code as such, only addresses. Since almost all machines implement the concept of words which can contain addresses, ITC may be generated in an essentially machine independent manner. On the other hand, DTC involves the generation of operand access routines, which makes machine independence much more difficult.

To examine space and time requirements, we compare the DTC generated by the PDP-11 FORTRAN compiler [3] to the ITC which might be generated by an equivalent compiler. Figure 3 shows that not only is the space decreased but also the time. The comparison is actually overly favorable to DTC. First, the routine $F0001, which must be generated as part of the compiler output to obtain the short BR jumps in the $P000*n* routines, has not been counted at all in the space since it is generated only once for several variables. Second, only the fetch (push) has been shown. Inclusion of the store (pop) adds an extra routine (DTC) or extra word (ITC) for each variable. Actually, the PDP-11 compiler generates a parametrized call for each store, which costs one extra word per assignment.

If threaded code were to be generated in register machine style as opposed to stack machine style, the saving would be even greater since in this case each register would require two addresses (ITC) or two routines (DTC) for each operand.

Several other advantages of ITC manifest themselves in specific connection with the SPITBOL system.

1. The address of the operand fetch routine which is associated with a dynamic value can serve as the type code identifying the value in dynamic memory.
2. The linking from one code block to another is easily affected by making the initial address in the new block point to the routine for switching code blocks. The SPITBOL system uses this fact to generate a separate code block for each statement leading to the interesting property that unreachable statements are garbage collected.
3. The operand fetch and store routine addresses, which are associated with a variable, may be modified in the case where the variable is input or output associated.

Received June 1974; revised September 1974

References

1. Dewar, R.B.K. SPITBOL. SNOBOL4 Doc. S4D23, Illinois Inst. of Tech., Chicago, Ill., Feb. 1971.
2. Bell, James R. Threaded code. *C.ACM 16*, 6 (June 1973), 370–372.
3. Digital Equipment Corporation, PDP-11 FORTRAN IV Programmer's Manual, DEC-11-KFDA-D, Maynard, Mass., 1971.

Programming Techniques
G. Manacher Editor

A Simplified Recombination Scheme for the Fibonacci Buddy System

Ben Cranston and Rick Thomas
The University of Maryland

A simplified recombination scheme for the Fibonacci buddy system which requires neither tables nor repetitive calculations and uses only two additional bits per buffer is presented.

Key Words and Phrases: Fibonacci buddy system, dynamic storage allocation, buddy system
CR Categories: 3.89, 4.32, 4.39

A severe problem in the Fibonacci dynamic storage allocation scheme is to locate the buddy buffer when recombining. Hirschberg [1] gives two algorithms. The first of these involves a table which grows in size rapidly with the maximum size buffer allowed. For a system allowing buffers of up to 17,717 words, this table requires 987 entries. The second algorithm involves a costly repetitive calculation. This paper presents a more efficient algorithm[1] which requires only two additional bits in some control field of each buffer.

Let the two additional bits be called the *B*-bit (for Buddy bit) and the *M*-bit (for Memory). These are in addition to the usual *A*-bit (buffer is currently in use and

This work was supported in part by Navy Contract ONR-N0014-67-A-0239-0032-01.

Authors' address: The University of Maryland, Computer Science Center, College Park, MD 20742.

[1] James A. Hinds [2] has described another scheme which has most of these advantages but requires more than two additional bits.