Data Structures in Digitek's FORTRAN IV Compiler for the SDS 900 Series

Robert W. Floyd

The FORTRAN IV compiler for the SDS 910-920-930-940 uses a set of data structure conventions together with interpretive operations for some of the data manipulations. According to the conventions, there are a fixed set of 25 lists numbered 0 - 24. They serve various purposes: one is a symbol table for fixed point scaler identifiers, others for floating, array, and dummy variable identifiers. One holds the stack of exits for recursive calls. One holds generated pieces of machine code awaiting rearrangement. One, the work list, plays a special role as a push-down accumulator.

Each list occupies a contiguous block of storage. (See Figure1)

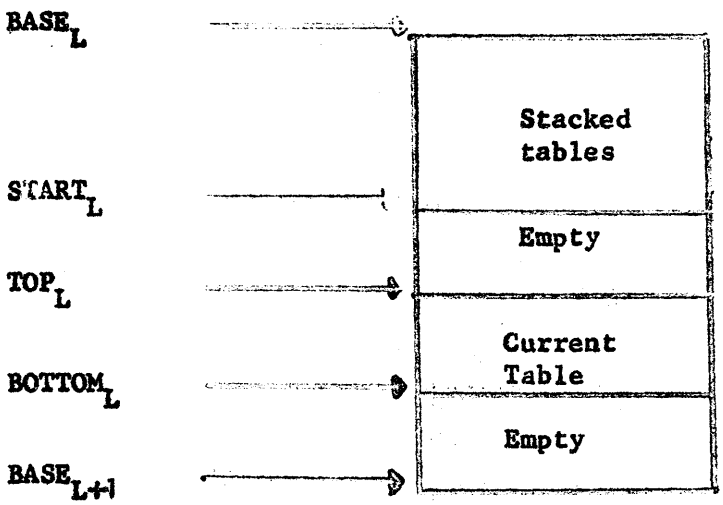

Figure 1

The list is a push-down list of tables, each of which can be used both as a last-in-first-out and as a first-in-first-out stack. For the list numbered L, there are four pointers to its block of storage.

(1) $BASE_L$ points to the word before the list. Thus the i^{th} word of the list is in location $BASE_L + i$.

(2) $START_L$ points to the last word of the pushed-down tables, the word before the region available to the current table.

(3) TOP_L points to the word before the current table.

(4) $BOTTOM_L$ points to the last word of the current table.

(5) $BASE_{L+1}$ serves as a limit on $List_L$. It is the word before the region allocated to the next list. We must therefore always have $BASE_L \leq START_L \leq TOP_L \leq BOTTOM_L \leq BASE_{L+1}$.

We now describe some of those interpretive operations (POPs; programmed operators) which use only the current table of a list. We use M as the effective memory address of an instruction, after indirect addressing and indexing, and m or [M] as the contents of that address. We shall designate the work list by W.

(1) FET M (Fetch m). $BOTTOM_W \leftarrow BOTTOM_W + 1$, $[BOTTOM_W] \leftarrow m$. This stacks m on the bottom of the work list.

(2) ADR M (Address M). Stack M on the bottom of the work list.

(3) SOB M (Save on bottom of M). Stack the contents of the hardware accumulator on the bottom of $List_M$.

(4) MON M (Move onto M). Unstack the bottom word of the work list and stack it as the bottom word of $List_M$.

(5) MOF M (Move off M). Unstack the bottom word of $List_M$, and stack it on the bottom of the work list.

(6) CLA* BOTTOM + L (A hardware instruction). Bring to the hardware accumulator the word from the bottom of $List_L$. The asterisk signifies indirect addressing.

(7) TOT M (Take off top of M). Unstack the top word of the current table of $List_M$, and save it on the bottom of the work list. $TOP_L \leftarrow TOP_L + 1$; $BOTTOM_W \leftarrow BOTTOM_W + 1$; $[BOTTOM_W] \leftarrow [TOP_L]$.

(8) SKR BOTTOM + L (A hardware instruction). Unstack the bottom word of $List_L$. $BOTTOM_L \leftarrow BOTTOM_L - 1$.

(9) MIN TOP + L (A hardware instruction). Unstack the top word of the current table of $List_L$. $TOP_L \leftarrow TOP_L + 1$.

(10) LCFM (Load Central from M). Load words CTL1 and CTL2 with the two words from the top of the current table of $List_M$.

(11) LCOM (Load Central off M). Same as LCF, except that the two words are unstacked from $List_M$.

(12) MCOM (Move Central onto M). Stack the two words CTL1 and CTL2 on the bottom of $List_M$.

Observe that words are added to a list at the bottom, but may be taken off at the bottom (last-in-first-out; a stack) or the top (first-in-first-out; a queue). All operations which add words to a list use the SOB operation, which checks whether the allocated space is full ($BOTTOM_L = BASE_{L+1}$), and if so calls on a storage allocator to move the lists and change the pointers.

It is possible to create a new current table T_1 on a list without harm to the previous current table T_0, which will again become the current table when T_1 is released. This is done by means of two operations:

(1) RSV M (Reserve M). Stack $START_M - BASE_M$ and $TOP_M - BASE_M$ on the current table of $List_M$, then $START_M \leftarrow BOTTOM_M$, $TOP_M \leftarrow BOTTOM_M$, creating a new(empty)current table on $List_M$.

Thus if the previous appearance of $List_M$ was as shown in Figure 2

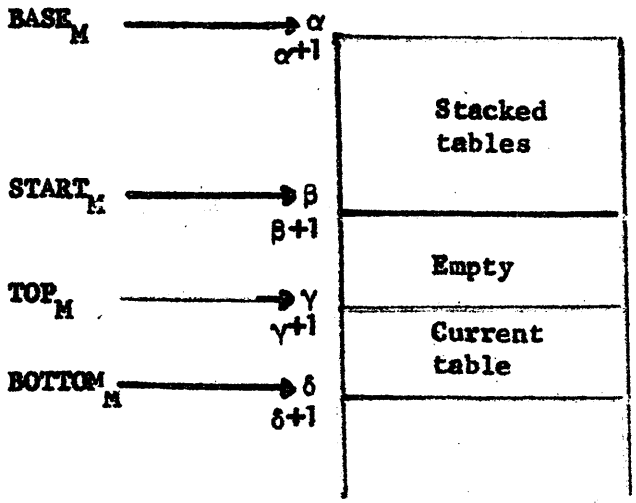

Figure 2

we would now have the situation shown in Figure 3.

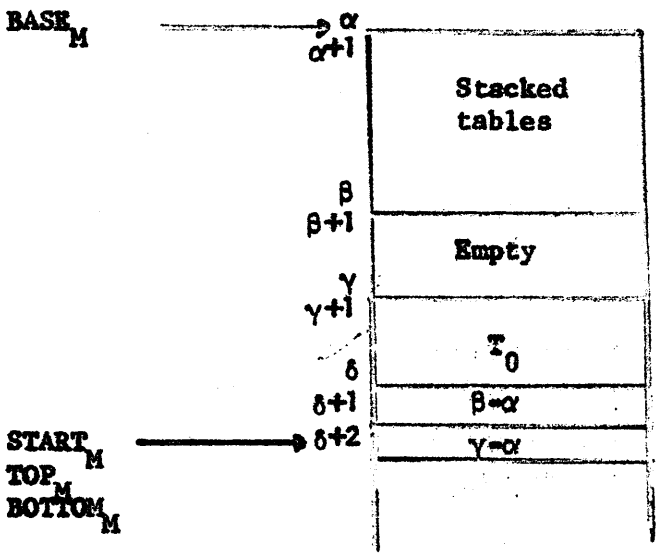

Figure 3

After several uses of SOB_M and TOT M, and possible memory reallocations which move the entire list without changing its contents, we might have, where r is a relocation constant, the situation of Figure 4.

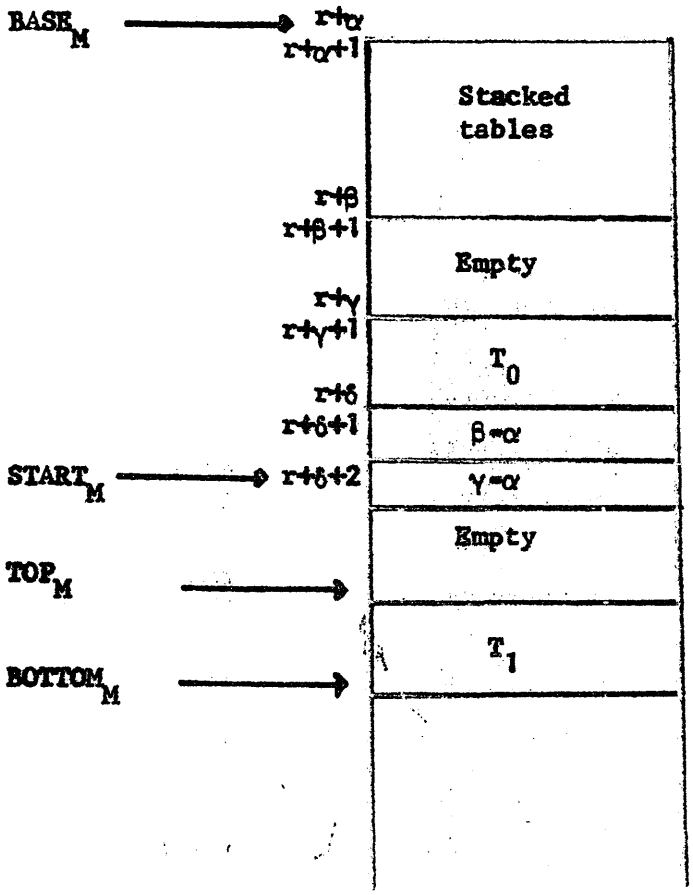

Figure 4

(2) RLS M (Release M) .

$BOTTOM_M \leftarrow START_M - 2$,

$TOP_M \leftarrow [START_M] + BASE_M$,

$START_M \leftarrow [START_M - 1] + BASE_M$.

We now have $BOTTOM_M = r + \delta$,

$TOP_M = r + \gamma$, $START_M = r + \beta$,

$BASE_M = r + \alpha$, so that the original status of the list with T_0 as the current table has been restored.

Because lists are constantly subject to relocation, it is not useful to save absolute addresses of words stored on lists. Instead, one stores a pointer; a word one of whose fields is the number of the list, the other being the location of the word relative to the base of the list.

BOP M (Bottom pointer of M) saves on the work list a pointer to the bottom word on list M.

CNT M (Count M) saves on the work list the size of the current table on $List_M$.

SAL M (Save a list M) saves on the save list $START_M - BASE_M$, $TOP_M - BASE_M$, and $BOTTOM_M - BASE_M$.

REC M (Recover M) is inverse to SAL, and restores $START_M$, TOP_M, and $BOTTOM_M$ to their earlier values.

Each list has a standard size (one to five words) of item stored on it. For example, a list of floating point constants might have two as its standard item length. In addition, lists such as symbol tables have items which begin with a one or two-word key, the symbol itself, followed by other information.

SER M (Search M). Search the list M for an item which has key equal to CTL2, or equal to the two-word pair (CTL1, CTL2), depending on what the key-length of M is. Save on the work list the pointer to the matching item.

Control Structures

The program is organized as a set of recursive subroutines. Exits are saved on one of the lists, the exit list. The programmed operators are implemented by recursive subroutines so that they can use themselves and each other. Apart from those subroutines accessed by the programmed operators, a recursive subroutine may be reached by

JRS M (Jump to recursive subroutine M).

At each level of subroutine nesting an answer bit is kept, and used to

record the results of tests.

JAT M (Jump, if answer is true, to M)

JAF M (Jump, if answer is false, to M)

Many of the operators described earlier set the answer true or false. For example, the programmed operators which remove items from lists set the answer false if the source list is empty. The search instruction sets the answer false if no match is found. We also have:

CSA M (Character scan or alternative).
If the next input character equals m, scan over it and set the answer true; otherwise do not scan, set the answer false.

SNE M (Set non-empty). Set the answer true if the current table of $List_M$ is non-empty, otherwise false.

SOC M (Set on character). Set the answer true if the next input character has, in its entry in a certain table, a flag bit set in the same position as the bit set in M. This instruction can be used, for example, to ask "Is the next character an alphanumeric?".

SOF M (Set on flag M). Same as SOC, but testing the bottom word on the work list, rather than the next input character.

SOL M (Set out of limit). Set the answer true if the absolute value of the double precision hardware accumulator is not greater than the double precision limit M.

In order to do backtracking, there are the following:

TRY M. Enter the recursive subroutine M. If it is left normally, by a transfer to EXIT, control returns to the instruction following the TRY with the answer set to TRUE. If an exit occurs by a transfer to FAIL, control returns with the answer set to FALSE, and with the list pointers reset to their values at the time TRY was executed.

FEX M (Fail exit M). After execution of this instruction, a transfer to FAIL will cause control to go to M, with lists restored to their state when FEX was executed.

CSF M (Character scan or fail). If the next input character is M, scan it; otherwise go to FAIL.

QSF M (Quote scan or fail). Scan on the input string the string stored at M, or go to FAIL.