Identification

Explanation of EPLBSA Code

Jean C. Scholtz

Introduction

Pass two of the EPL compiler (EPL2) uses as input the macro code produced by pass one and produces as output EPLSBA code. This code consists of blocks of GE-645 instructions identified by certain symbol sequences. It is perhaps helpful to think of EPL2 as analyzing the macro code and deciding which block of code (or routine) to output. A detailed description of the GE-645 instructions may be found in the GE-645 Manual. The discussion presented here will be of the general purpose of the various routines produced by EPL2. The following notation will be used. When one or more digits appear in a symbol these digits will be updated as the symbol reoccurs. For example, p1.0 (used to denote the prologue) will be updated as follows:

p1.1, p1.2, ..., p2.0, p2.1, ..., p2.6, etc.

The blocks so denoted will be chained together with tra instructions. It should be noted that these blocks will not usually appear in order in the pass 2 output. That is, p8.6 might appear before p1.10.

Sequence of Execution

Upon entering a procedure there is a certain order in which blocks of code are referenced. When a procedure is entered a transfer is made to the save routine where the bases and registers are stored and the next stack frame is set up. If the validate procedure option is in effect, external procedures next call .vℓ. Internal entries next update display, using .cp, which is supplemented by other code at block level 3 and by .cp1 at levels over 3. A transfer is then made to the prologue block of code. The sequence here is as follows:

1) If the block is an on-unit, a transfer is made to .on0 to establish the correct action for the on-condition.
2) The next three activities to be executed are intermixed in order of their appearance in the EPL text. These are setting up of handlers (.h0) to take care of on-conditions, the establishment of specifiers for automatic data, and the setting up of dope for automatic adjustable data.

3) If the block has an epilogue, the epilogue handler is now created.
4) If the block contains any automatic data to be initialized, the routine .ial.0 is now activated.
5) In order of declaration during the prologue, the auto varying strings are set to null by calling .vl .
6) At the conclusion of these activities, control is passed to the main sequence (sl.0). After the s sequence has been completed, control is passed to the epilogue sequence.

In the epilogue sequence the following order of activities prevails:

1) If this was an on-unit, .off0 is called to restore the condition.
2) In order of declaration during the epilogue block, on-statements are reverted, and automatic varying strings are freed by a transfer to .v2.
3) Next, the epilogue handler is restored.
4) Finally, a transfer is done to the return routine .rt .

Explanation of Sequences

There are three main symbol sequences appearing in EPLBSA code - pl.0, sl.0, and el.0. pl.0 is termed the prologue; sl.0 is the block or main sequence; el.0 is the epilogue. Very generally pl.0 is concerned with defining variables and constants and setting up proper storage. sl.0 contains the in-line code for executing each EPL statement. el.0 resets certain conditions as they were before execution. Note that pl.0, ..., pl.n (where n is some digit) refers to the first block level of the EPL procedure. As soon as a procedure block or a begin block occurs within the main procedure block, the prologue will be at level two - p2.0. sl.0 and el.0 are updated similarly. At execution time the entire prologue sequence for every block is executed before the block sequence. This, in turn, is executed before the epilogue sequence. In the output the p, s, and e blocks of code will be intermixed. el.0 could appear before pl.1, but the order of execution is as explained above.

Unique Symbols and Symbol Sequences

These three main symbol sequences contain some unique code but they also contain references to many other blocks of code, each block being identified by a certain symbol. These blocks of code will be classified as to whether they are referenced by the prologue, main or epilogue sequence. The bulk of the output of pass two falls under symbol sequences, of which pl.0, sl.0, and el.0 are examples. Other blocks of code are identified by unique symbols. These blocks of code appear only once in the pass two output. They are usually closed subroutines to be referenced many times by other blocks of code. The terms symbol sequence and unique symbol will be used to subdivide the discussion of EPLBSA code.

Code Referenced by Epilogue Sequence

The epilogue sequence references only one symbol sequence, .off0. The .off0 blocks of code deal with on-conditions. In particular, suppose the EPL procedure reads as follows:

```
          a:  procedure;
              on overflow x = 6;
              .
              .
              .
              end;
```

The activity "x = 6" is called the on-unit. The .off0 block is invoked in the epilogue of the on-unit to reestablish the on-unit.

The two unique symbols referenced from the epilogue are .rt and .v2. .v2 contains code to free automatic varying strings declared in the block. .rt is the return routine. MSPM BD.7.02 discusses this in detail.

Code Referenced by the Prologue Sequence

The prologue uses blocks of code that have to do with setting up storage for the variables used in the procedure. The following are the symbol sequences referenced:

.ia0 - These blocks contain the information to be stored in the dope vectors for variables with the automatic storage classification.

.h0 - These symbols name condition handlers (EPL design journal #3) which are used to preserve chains of stacked on-units. Condition handlers appear for each condition in the stack frame of each block in which the condition is mentioned. Handlers are chained by the prologue, pushed down by on-statements, consulted by signal statements and popped up by revert and epilogue.

.on0	This code is invoked in the prologue of an on-unit to roll the on-unit stack back one level. Its work is undone in the epilogue by a corresponding .off0.
.sv0	A special temporary used during the prologue of an on-unit.
.al	These symbols appear when a call is made within a procedure. The argument list to a procedure called from block 1 appears at sp|.al.
.ial.0	Initialization of automatic storage is done by this routine. It is entered upon completion of the remainder of prologue pl.0.
.asl	These symbols define the limit of automatic storage for each block.
.ul	These symbols denote stack space available for local scratch.
.wl	There is one of these constants per block. sp|.wl will hold the pointer to the dope during calculation of adjustable declarations for the block. Since this calculation is done in a constructed inner block, sp|.wl actually lies in the inner frame.
.bl	These blocks of code are used in the calculation of adjustable bounds and lengths. A routine (whose name is created by pass one) to evaluate each variable quantity is called by tsx 2 during evaluation of adjustable dope.

The following are unique symbols referenced by the prologue block:

.cp	This block of code copies display (MSPM BP.3.00) for internal blocks. It copies the innermost statically embracing stack pointer from the argument list into display, and just in case this block was invoked from another segment, restores $\ell b \leftarrow \ell p$.
.ds	The sp|.ds is the actual display. The display contains pointers to the statically embracing stack frames in case it is necessary to use (in this inner block) some variable which has been declared in an outer block.
.vl	This is a block of code used to set automatic varying strings to null.

.dpl - This block of code evaluates adjustable automatic dope and allocates storage for the associated data.

.u0 - The sp|.u0 is always equivalent to 32 decimal. This is local scratch available at block levels.

<u>Code Referenced by the Main Sequence</u>

The main or s1.0 blocks reference the following symbol sequences:

.yl - These indicate symbols created in pass two for various purposes.

.is0 - These blocks of code deal with initializing internal static storage.

.id0 - These blocks of code are executed before .is0. They set up specifiers for the internal static storage. The .id0 block is entered by linkage fault on first reference to internal static.

.ctl - One of these blocks of code appears for every controlled adjustable variable. Information contained in this block is used to evaluate the adjustable declarations.

.f35 and .f71 - These are routines used to convert a floating point number to a fixed point number. The routine used depends on the number of bits desired in the result - thirty-five or seventy-one. The fixed to floating conversion is simple enough that it calls no routine but is done in-line.

The following are the unique symbols which are referenced by the main or s blocks of code:

.sv - This is the save routine. Unlike the save expansion discussed in MSPM BD.7.02 which is output each time it is necessary to go down a level in the stack, EPL 2 uses .sv as a closed routine. This routine differs from the standard in making possible stack frames larger than 2^{14}-1 words. Upon exit from .sv the bp is pointing to the same location in the stack as the sp.

.sbl - This block of code is used for evaluating subscripts. .sblℓ and .sblt are symbols used within this block of code. .sb2 is also used to evaluate subscripts but a subscript range check is included. The .sb2 block would appear when "subscriptrange" has been enabled in the EPL procedure.

.of0 - This is a routine to compute bit offsets for packed data. .of1 computes this offset when the packed data is contained in a packed structure. Both routines arrange to keep the bit offset under 36. .of1a is a symbol used within this block.

.dv0 - This block contains the information to be stored in the dope vectors for static variables.

.va - This piece of code is used to set up specifiers for scalar varying strings contained in aggregates, which are passed as arguments.

.ei - This block of code performs the entry for routines used for external initial and internal static storage.

.sa - This routine is used to create dummy dope for scalar elements of packed aggregates passed as arguments.

.cta and .ctb - These blocks of code make up an argument list and call the dope vector calculator for adjustable based storage. .cta is used for locally declared variables, .ctb for nonlocal.

.dp0 - This routine completes the calculation of adjustable based dope vectors by invoking tdope - and then returning. Each .ct0 block ends with a <u>tra</u> .dp0.

<u>Miscellaneous Block of Code</u>

.vℓ - This block handles the validate procedure option.