Published: 07/12/68

Identification

BCPL-MULTICS compiler - code generation
R. H. Canaday

Purpose

This document discusses the code produced by the BCPL-MULTICS compiler. Most of this document consists of descriptions of various detailed aspects of the code. The reader is presumed to be familiar with Ref(1) and with MSPM sections BZ.6.00 through BZ.6.03.

This document is not intended for the casual reader. It will be helpful to refer frequently to an EPLBSA listing produced by the BCPL compiler (such as is provided in the appendix) while reading the following discussions.

Summary

This document is subdivided as follows:

1) Initialization

Unlike EPL, BCPL programs cannot always be initialized on first reference. For example, a BCPL procedure cannot communicate through the global vector with procedures in other segments unless those segments have been initialized.

2) Stack management

Since BCPL does not use a standard MULTICS call-save-return for intercommunication, a separate BCPL stack is set up and maintained. The programmer has a great deal of (optional) control over the stack and global vector.

3) Call-Save-Return

Intercommunication between BCPL procedures is described.

4) Communication with EPL Procedures

Run-time routines have been provided which make possible recursive calls from EPL to BCPL procedures and vice-versa.

5) <u>How to read EPLBSA produced by the BCPL compiler</u>.

The listing from a compilation is not easy to correlate with the source code which produced it. However, the details given in this section should make the task somewhat easier.

<u>Initialization</u>

A BCPL procedure segment name is the same as the name of the CTSS or MULTICS source file which produced it. Unlike EPL, the segment name has nothing to do with the name of any procedure within the segment.

A BCPL procedure segment should be initialized before it is executed. The initialization of a segment named `XXX' can be done either by calling (from EPL) the initialization entry point `XXX$XXX' [e.g., `call XXX'] or by calling (from EPL) a procedure within the segment [e.g., `call XXX$PRO2 (arg1,arg2,...)].

Example:

A program consists of three BCPL procedure segments named `ASEG', `BSEG', and `CSEG'. The entry point is the procedure `PRO3' in `BSEG'. There are no arguments. The EPL driver would be as follows:

```
START:    PROC (  );
          CALL ASEG;
          CALL CSEG;
          CALL BSEG$PRO3;
          END;
```

In BCPL there are two forms of static storage which must be initialized: `global' and `local'. Local static storage is kept in the linkage segment and is truly static - like EPL static it is initialized only once in a given process. Repeated calls to initialize it will have no effect. A switch word, "INITSW", in each linkage segment is set to zero to indicate that initialization has been done. Global static, on the other hand, is kept in the `global vector' (see the next section, <u>Global and Stack Management</u>, for details on the allocation of the global vector). Global static can be reinitialized repeatedly. Calling a BCPL procedure (`call BSEG$PRO3') will initialize global static only if it has never been initialized for the segment BSEG (i.e., if INITSW $\neq$ 0), but calling the initialization entry point (`call BSEG') will always reinitialize global static for BSEG. This is used to prevent interference between logically separate BCPL programs running in the same process.

Segment `BCPLGL' contains the routines which initialize static storage and which accept EPL calls to BCPL procedures and do argument conversion. The entry points are:

`BCPLGL$GINIT' called automatically by BCPL procedures to initialize global static and, if necessary, local static.
`BCPLGL$GSETU' called automatically by BCPL procedures to do stack setup and argument conversion.

There are also other entry points which are described in later sections of this document.

Global and Stack Management

Since BCPL does not conform to MULTICS standards in its stack management, it cannot use `SP'to point to its current stack frame. `SP' always points to a valid MULTICS stack frame. While BCPL procedures are in execution, `SP' points to a stack frame immediately inferior to (called by) that of the caller of BCPL, which contains only the header information and two words of diagnostic information: SP|32 contains an ITS pair which identifies the BCPL procedure segment which was called from EPL, and SP|34 contains an ITS pair which points to the global vector and the stack frame of the first BCPL procedure that was called.

The BCPL global vector and stack are contained in a segment which may be supplied by the caller or routines superior to it, or which will be created during BCPL initialization if not supplied. BCPL uses "unique_chars" and "smm$set_name_status" to create a segment of 250K words. The global vector, which has a standard length of 2048 words, always starts at word 0 of the segment. The stack area begins immediately after the last word of the global vector.

During execution of a BCPL procedure the address register pairs are used as follows:

AP - points to current BCPL stack frame
(AB points to global vector)

BP - general usage

SP - MULTICS stack frame

LP - linkage segment

The registers are always paired. AB can be used for the global vector because the global vector always begins at word 0 of the segment containing the BCPL stack frames.

BCPL stack segment management centers around a pointer called the "Current Start of Stack" pointer, abbreviated "CSS". This pointer is kept in static storage and is accessed and changed through four routines. Routine BCPLGL$SETAP can be called from EPL to set CSS to any address, including null. Routine BCPLGL$LOOKAP returns in its argument the current value of CSS. Both entries have one argument, of type `PTR'.

The third routine, BCPLGL$GSETU, is not callable from EPL. It is invoked automatically whenever an EPL procedure calls a BCPL procedure. Its function is to do EPL-BCPL argument conversion and to load the address pair AB-AP with the value of CSS.

The fourth routine is the BCPL-callable procedure `Call', accessed through global vector location 14. Its effect on CSS is to save the old value of CSS and reload CSS with a pointer to the first free word after the current BCPL stack frame. `Call' then calls the desired EPL procedure. On return the current value of CSS is discarded and the saved value is restored.

The commonest use of these routines is to make sure that two independent BCPL programs (sets of procedures) running in the same process do not conflict in their use of the global vector. The simplest way to guarantee that a BCPL program will be executed with a clean global-and-stack segment is to issue the call

```
CALL BCPLGL$SETAP  (NULL)
```

before initializing any BCPL procedure segment in the program. Note that *any* `CALL BCPLGL$SETAP' issued after the initialization of a BCPL procedure segment will invalidate that initialization. The only exception to this is that if a BCPL procedure calls an EPL procedure, nothing done by that procedure or any inferior procedure can affect that value of CSS which will be restored when the EPL procedure returns to the BCPL procedure. In fact, a good way of insuring that the global vector will be unchanged by (and invisible to) inferior procedures is to issue the call

```
CALL BCPLGL$SETAP  (NULL);
```

in any EPL procedure called from a BCPL procedure.

In the case of a BCPL-to-EPL-to-BCPL calling chain in which the inferior BCPL procedure should reference the same global vector as the superior one, the superior routine can protect itself from changes made by the inferior routine by issuing the BCPL calls:

Save ( )

before calling the EPL routine, and

Restore ( )

on return. The effect of `Save ( )' and `Restore ( )' is to save and restore the global vector in a pushdown stack (which is kept in the MULTICS stack). The call `Save ( )' does not change the contents of the global vector. `Save ( )' and `Restore ( )' can be used not only to bracket a call to an EPL procedure, but in fact at any point in a BCPL procedure.

The routines `BCPLGL$SETSP(PTR)' `BCPLGL$LOOKAP(PTR)', `Save ( )', and `Restore ( )' give the programmer complete control over global static storage. However, he has very little control over local static storage. Thus in the MULTICS environment it may be useful to use portions of the global vector for static storage. However, be warned that these routines for controlling global static may be dependent on the MULTICS environment. Thus coding dependent on them may not be machine independent.*

Call-Save-Return

The BCPL stack frame header consists of 8 words, containing the following information:

WORD	CONTENTS	STORED BY
0-1	RETURN ITS	SAVE
2-3	LP FOR CURRENT PROCEDURE	SAVE
4	UNUSED	
5 UPPER	LEFT-HAND-SIDE FLAG	CALL
5 LOWER	ARGUMENT COUNT	CALL
6-7	AP FOR SUPERIOR PROCEDURE	SAVE

The code produced by the compiler for call-save-return in procedure "PROCED" is as follows. In this case the new stack frame will start at the nth word of the previous frame.

CALL

```
LDA       m,DL           # arguments
STA       AP|n+5
EAX2      n
LDA       CALLADDRESS    A BCPL ADDRESS
EABBB     0,AU
TSBBP     BB|0,AP
```

---

* The CTSS (and GECOS) environments act as if a `BCPLGL$SETAP(NULL)' call were issued at every return to command level (between activities).

```
SAVE                                          in the linkage segment

CALLADDRESS:  EAPLP       NULL
              EAX7        STAT-*,IC                  Set LP
              TRA         <PROCED>|SV                The procedure
                                                     Save Sequence

_____________________in the procedure segment ('PROCED')

SV:         NULL
            STPAP      AP|6,2             old AP
            STPBP      AP|0               return point
            EAPAP      AP|0,2             new AP
            STPLP      AP|2               new LP
            SZN        LP|INITSW          zero if static has
                                          been initialized

            TZE        0,7                enter procedure body

INIT:       NULL                          initialization procedure

____________

RETURN
            EAPAP      AP|6,*             old AP
            EAPLP      AP|2,*             old LP
            TRA        AP|0,*             return
```

Communication with EPL Procedures

As has previously been stated, communication between BCPL procedures is by means of a special calling sequence rather than the standard MULTICS call. Some of the significant differences between a BCPL call and a standard MULTICS call are:

1) BCPL arguments do not have dope
2) BCPL arguments are by value.
3) The BCPL stack frame header is 8 words long rather than 32.
4) Registers are not saved and restored across a call, except for AB-AP, LB-LP, and SB-SP.
5) SB-SP is not used. The stack pointer is AB-AP.
6) The address of the callee is stored in a variable.

The link-by-name features of MULTICS are not used. Obviously, despite these differences, a mechanism must exist to permit calls from EPL procedures to BCPL procedures and vice-versa.

In order to enable EPL procedures to call BCPL procedures the BCPL compiler generates two entry points for each BCPL procedure. One of these is used in BCPL-to-BCPL calls to the procedure. The other entry point, labeled with the name of the procedure, is callable from EPL. This entry point invokes the routine `BCPLGL$GSETU' which transforms the call into BCPL format and generates a BCPL argument list from the EPL argument list. Dope and specifiers are not interpreted. The addresses in the EPL arglist are translated into BCPL addresses and placed in the BCPL arglist. Dope and specifiers can be interpreted by the callee as desired. Two functions exist to aid in interpreting arguments. These are `MtoBaddr(X)' which accepts the (BCPL) address of an ITS pair and returns the value of the ITS pair as a BCPL address, and `MtoBstring(X,V)' which accepts the (BCPL) address `X' of a MULTICS string specifier and places a corresponding BCPL string in vector `V'.

Calling from a BCPL procedure to an EPL procedure is done using the two library routines `Getadr' and `Call' (cf MSPM BZ.6.02). All of the arguments of `Call' are addresses. The first argument is the address to be called. The remaining addresses will be converted to ITS pairs and stored in an EPL-format argument list. If any arguments require dope and specifiers, they must be generated by the caller before calling `Call'. Two library functions, `ITS' and `BtoMstring', are provided for this purpose. Function `ITS' creates an ITS pair from a BCPL address. Function `BtoMstring' creates a fixed length MULTICS character string from a BCPL string. MSPM BZ.6.02 contains more detail on `ITS' and `BtoMstring'.

The only complicated part of the method for calling between EPL and BCPL procedures is the stack management, which was described in detail above. In general it is possible to call freely between EPL procedures and BCPL procedures if one remembers that in the absence of calls to `BCPLGL$SETAP', there will not be any conflicts in stack usage. It is also worth noting that the EPL call

```
CALL BCPLGL$SETAP (NULL);
```

is always a safe way of guaranteeing that BCPL programs which precede and follow the call will be independent.

<u>How to read EPLBSA produced by the BCPL compiler</u>

The BCPL compiler produces some comments to help correlate the EPLBSA code with the source program: the source-program name of each variable is given at the point it is declared; the source name of each function and variable labels the corresponding entry point; and each call to a function or routine is labelled, at its occurrence in the code, with the source name of the callee.

Register usage by the code generators is as follows:

A Q	used for addresses or computations
AQ	used for floating point, multiply, and divide
x3-x4	always used as a pair - x4 = uppper half word, x3 = lower
x5-x6	a pair like x3-x4
x2	contains the stack increment (frame size) during a call
x0	used with BB for indirection
x1,x7	unused

Multiple location counters are used, as follows:

MAINC	procedure segment	main counter
LASTC	procedure segment	builtin subroutines
STATC	linkage segment	static storage to be initialized
LSTATC	linkage segment	static storage not needing initialization
IGLØBC	procedure segment	data to be put in the global vector

The next part of this discussion consists of examples of BCPL statements and the code generated by them, together with a brief description where necessary.

<u>Example A</u> declarations

```
let a,b,c = 0,0,5    STZ AP|6+10          "Declare a
                     STZ AP|6+11          "Declare b
                     LDA 5,DL
                     STA AP|6+12          "Declare c

let v = vec 100      EAPBP AP|6+14
                     EAX3  AP|6+14
                     SBRBB AP|6+13
                     SXL3  AP|6+13        "Declare v
```

This is an example of a very common operation, namely address generation. The variable `v' is at location 19 of the stack frame. The vector begins at location 20. `19' is written `6+13' because of a historical carry-over from GMAP.

<u>Example B</u> assignments

```
a :=v[20]                  LDA    AP|6+13       v
                           EABBB  0,AU
                           LDA    BB|20,AL
                           STA    AP|6+10       a
```

This is an example of using an address. Addresses may be used from registers A or Q, or from register pairs x3-x4 or x5-x6.

```
v[20] := b[a]              LDQ AP|6+13    v
                           LDA AP|6+11    b
                           ADA AP|6+10    a
                           EABBB 0,AU
                           LDA BB|0,AL
                           EABBB 0,QU
                           STA BB|0,QL
```

<u>Example C</u> address arithmetic

```
v[0] := v[20] * a[b+c]
                           LDA    AP|6+10      a
                           ADA    AP|6+11      b
                           ADA    AP|6+12      c
                           EAX4 0,AU
                           EAX3 0,AL
                           LDQ    AP|6+13      v
                           EABBB 0,QU
                           LDQ    BB|20,QL     v[20]
                           EABBB 0,4
                           MPY    BB|0,3       a[b+c]
                           EABBB AP|6+13       v
                           STA    BB|0,8
```

<u>Example D</u>

<u>a string</u> (does not need initialization)

```
     USE                   LSTATC
L5:  NULL
     ØCT                   005141142143
     ØCT                   144145000000
```

an address (this will be transformed into a BCPL address which will be stored in place of the EAPBP)

```
     USE                      STATC
L6:  NULL
     EAPBP                    LP|L8   label in linkage seg.
L7:  NULL
     EAPBP                    AP|L9   label in procedure seg.
```

an address to go in global vector location 20

```
     USE                      IGLØBC
     EAPBP                    AB|L10  label in procedure seg.
     ZERO                     0,20
```

```
BCPL invoked through Mrgedt
Control Card =  old bcpl  test99


        //  This is a demonstration of BCPL code generation,
        //  The program does not do anything and cannot be executed,

        global $( Demonstration:100; Function:101  $)
        let Demonstration (A1,A2)  = valof  $(
                let a,b,c = 0
                let v = vec 100
                a := v[20]
                v[20] := b[a]
                v[0] := v[20] * a[b+c]
                c := "This is a string"
                a := Function(a, A1)
           Label:
                Label := Label + A2
                resultis A1 / Label $)
```

```
12564 03  07-09-68              OUTPUT FROM EPLBSA ASSEMBLY

.EPLBSA. PACKAGE 10 VERSION, 15 MAY 68.

.EPLBSA. BEGIN COMPILATION.

.EPLBSA.   ASSEMBLY OF FILE  $TEST99$,     SEGMENT NAME IS   TEST99

        000000                        1
        000000                        2              NAME      TEST99
        000000                        3              FILE      TEST99
 S      000000           000032       4              LINK      SAVER,<TEST99>^SV
                                      5              USE       LASTC
        000114                        6   LAST:      NULL
                                      7              USE       LSTATC
     0A 000016   000121 0000 00       8   INITSW:    ARG       INIT
                                      9              USE       STATC
                                     10              EIGHT
     AA 000013    000000 000000      11              BSS       STAT,4    "TEMP STORAGE
                                     12              USE       IGLOBC
        000110                       13   IGLOB:     NULL
                                     14              USE       MAINC
        000000                       15   START:     NULL
                                     16              JOIN      /LINK/STATC,LSTATC/TEXT/IGLOBC,LASTC
        000000           000000      17              ENTRY     TEST99
        000000                       18   TEST99:    NULL
     AA 000000  6 00022 3521 20      19              SAVE
     AA 000001  2 00020 6521 00
     AA 000002  2 00040 3521 00
     AA 000003  2 77762 2521 00
     AA 000004  2 77740 3321 00
     AA 000005  6 00032 2501 00
     0A 000006   000121 7070 00      20              TSX7      INIT
     AA 000007  6 00020 1731 20      21              RETURN
     AA 000010  6 00010 0731 00
     AA 000011  6 00024 6101 00
     0A 000012   000110 7100 00      22              TRA       L2
```

```
0A 000013   000110 7100 00                 TRA       L3
   000014                        "
   000014                        "
   000014                        "  MAX STACK USED =  0
   000014                                  SEGREF    TEST99,DEMONSTRATION
   000014               000014             ENTRY     DEMONSTRATION
   000014                        DEMONSTRATION:      NULL
0A 000014   000127 7070 00                 TSX7      BXSETU
                                           USE       LSTATC
   000017                        L4:       NULL
LA 000017   777761 3700 04                 EAPLP     STAT-*-8,IC
0A 000020   000015 6270 00                 EAX7      L4A
4A 000021  4 00032 7101 20                 TRA       LP*SAVER,*
                                           USE       MAINC
   000015                        L4A:      NULL
AA 000015  0 00012 4501 00                 STZ       AP*6+4    "DECLARE A
AA 000016  0 00013 4501 00                 STZ       AP*6+5    "DECLARE B
AA 000017  0 00014 4501 00                 STZ       AP*6+6    "DECLARE C
AA 000020  0 00016 3531 00                 EAPBB     AP*6+8
AA 000021  0 00016 6231 00                 EAX3      AP*6+8
AA 000022  0 00015 5431 00                 SBRBB     AP*6+7
AA 000023  0 00015 4431 00                 SXL3      AP*6+7    "DECLARE V
AA 000024  0 00015 2351 00                 LDA       AP*6+7
AA 000025   000000 3130 01                 EABBB     0,AU
AA 000026  3 00024 2351 05                 LDA       BB*20,AL
AA 000027  0 00012 7551 00                 STA       AP*6+4
AA 000030  0 00015 2361 00                 LDQ       AP*6+7
AA 000031  0 00013 2351 00                 LDA       AP*6+5
AA 000032  0 00012 0751 00                 ADA       AP*6+4
AA 000033   000000 3130 01                 EABBB     0,AU
AA 000034  3 00000 2351 05                 LDA       BB*0,AL
AA 000035   000000 3130 02                 EABBB     0,QU
AA 000036  3 00024 7551 06                 STA       BB*20,QL
AA 000037  0 00015 2351 00                 LDA       AP*6+7
AA 000040   000000 6240 01                 EAX4      0,AU
AA 000041   000000 6230 05                 EAX3      0,AL
AA 000042  0 00012 2361 00                 LDQ       AP*6+4
AA 000043  0 00013 0761 00                 ADQ       AP*6+5
AA 000044  0 00014 0761 00                 ADQ       AP*6+6
```

```
AA 000045   000000 3130 02     62              EABBB     0,QU
AA 000046  3 00000 2361 06     63              LDQ       BB*0,QL
AA 000047   000000 3130 14     64              EABBB     0,4
AA 000050  3 00024 4021 13     65              MPY       BB*20,3
AA 000051  0 00015 2201 00     66              LDX0      AP*6+7
AA 000052   000000 3130 10     67              EABBB     0,0
AA 000053  0 00015 7201 00     68              LXL0      AP*6+7
AA 000054  3 00000 7561 10     69              STQ       BB*0,0
4A 000055  4 00022 3531 00     70              EAPBB     LP*L6
4A 000056  4 00022 6231 00     71              EAX3      LP*L6       FORM ADDRESS
AA 000057  0 00014 5431 00     72              SBRBB     AP*6+6
AA 000060  0 00014 4431 00     73              SXL3      AP*6+6
AA 000061  0 00012 2351 00     74              LDA       AP*6+4
AA 000062  0 00174 7551 00     75              STA       AP*6+118
AA 000063  0 00010 2351 00     76              LDA       AP*6+2
AA 000064  0 00175 7551 00     77              STA       AP*6+119
AA 000065   000002 2350 07     78              LDA       2,DL
AA 000066  0 00171 7551 00     79              STA       AP*6+115
AA 000067   000164 6220 00     80              EAX2      6+110
AA 000070  1 00145 2351 00     81              LDA       AB*101
AA 000071   000000 3130 01     82              EABBB     0,AU
AA 000072  0 00000 3571 00     83              STCD      AP*0
AA 000073  3 00000 2721 05     84              TSBBP     BB*0,AL    "CALL FUNCTION *
AA 000074  1 00000 2351 00     85              LDA       AB*0
AA 000075  0 00012 7551 00     86              STA       AP*6+4
   000076                      87    L7:       NULL
4A 000076  4 00014 2351 00     88              LDA       LP*L5
AA 000077  0 00011 0751 00     89              ADA       AP*6+3
4A 000100  4 00014 7551 00     90              STA       LP*L5
```

* NOTE: CALL, SAVE, AND RETURN AS SHOWN HERE USE "STCD" BECAUSE OF A HARDWARE BUG IN THE "TSBBP" INSTRUCTION.

```
AA 000101  0 00010 2361 00     91          LDQ      AP*6+2
4A 000102  4 00014 5061 00     92          DIV      LP*L5
AA 000103  1 00000 7561 00     93          STQ      AB*0
0A 000104    000105 7100 00    94          TRA      L8
                               95          USE      LSTATC
   000022                      96   L6:    NULL
AA 000022    020124 150151     97          OCT      020124150151
AA 000023    163040 151163     98          OCT      163040151163
AA 000024    040141 040163     99          OCT      040141040163
AA 000025    164162 151156    100          OCT      164162151156
AA 000026    147000 000000    101          OCT      147000000000
                              102          USE      STATC
   000014                     103   L5:    NULL
AA 000014  1 00076 3521 00    104          EAPBP    AB*L7
                              105          USE      MAINC
   000105                     106   L8:    NULL
AA 000105  0 00006 3501 20    107          EAPAP    AP*6,*
AA 000106  0 00002 3701 20    108          EAPLP    AP*2,*
AA 000107  0 00000 6101 00    109          RTCD     AP*0
   000110                     110   L3:    NULL
   000110                     111   "
   000110                     112   "
   000110                     113   "  MAX STACK USED =  128
                              114          USE      IGLOBC
   000110                     115   L2:    NULL
4A 000110  4 00017 3521 00    116          EAPBP    LP*L4
AA 000111    000000 000144    117          ZERO     0,100
AA 000112    000000 000000    118          ZERO
AA 000113    000000 000000    119          ZERO
                              120          USE      STATC
AA 000015    000000 3520 00   121          EAPBP 0
                              122          USE      LASTC
   000114                     123   SV:    NULL
AA 000114  0 00006 2501 12    124          STPAP    AP*6,2
AA 000115  0 00000 3501 12    125          EAPAP    AP*0,2
AA 000116  0 00002 6501 00    126          STPLP    AP*2
4A 000117  4 00016 2341 00    127          SZN      LP*INITSW
AA 000120    000000 6000 17   128          TZE      0,7
   000121                     129   INIT:  NULL
```

```
4A 000121  4 00016 2351 00      130              LDA      LP*INITSW
4A 000122  4 00016 4501 00      131              STZ      LP*INITSW
AA 000123  6 00000 6501 00      132              STPLP    SP*0
0A 000124    000110 6260 00     133              EAX6     IGLOB
2A 000125    000014 6250 00     134              EAX5     STAT+4
4A 000126  4 00050 2721 20      135              TSBBP    <BCPLGL>*[GINIT]     RETURN BY  TRA   BB*0,7
   000127                       136    BXSETU:   NULL
AA 000127  6 00022 3521 20      137              SAVE
AA 000130  2 00020 6521 00
AA 000131  2 00040 3521 00
AA 000132  2 77762 2521 00
AA 000133  2 77740 3321 00
AA 000134  6 00032 2501 00
0A 000135    000114 3520 00     138              EAPBP    SV
AA 000136  6 00000 6501 00      139              STPLP    SP*0
4A 000137  4 00052 7101 20      140              TRA      <BCPLGL>*[GSETU]
                                141              END

 NO LITERALS

 ENTRY POINTS AND SEGDEF NAMES
5A 000140     000006 000000
2A 000141     000044 000001
AA 000142  015 104 145 155      142    DEMONSTRATION
AA 000143  157 156 163 164
AA 000144  162 141 164 151
AA 000145  157 156 000 000
5A 000146     000012 000000
2A 000147     000036 000001
AA 000150  006 164 145 163      143    TEST99
AA 000151  164 071 071 000
5A 000152     000020 000000
6A 000153     000000 000002
AA 000154  014 163 171 155      144    SYMBOL\TABLE
AA 000155  142 157 154 137
AA 000156  164 141 142 154
AA 000157  145 000 000 000
5A 000160     000025 000000
6A 000161     000030 000002
AA 000162  010 162 145 154      145    REL\TEXT
AA 000163  137 164 145 170
AA 000164  164 000 000 000
5A 000165     000032 000000
```

```
AA 000167  010 162 145 154    146   REL\LINK
AA 000170  137 154 151 156
AA 000171  153 000 000 000
5A 000172     000037 000000
AA 000174  012 162 145 154    147   REL\SYMBOL
AA 000175  137 163 171 155
AA 000176  142 157 154 000
AA 000177     000000 000000

 EXTERNAL NAMES
AA 000200  005 147 163 145    148   GSETU
AA 000201  164 165 000 000
AA 000202  005 147 151 156    149   GINIT
AA 000203  151 164 000 000
AA 000204  006 142 143 160    150   BCPLGL
AA 000205  154 147 154 000

 NO TRAP POINTER WORDS

 TYPE-PAIR BLOCKS
AA 000206     000004 000000
55 000207     000044 000040
```