TO: MSPM Distribution
FROM: R. M. Graham
SUBJ: BD.7.03
DATE: 07/14/67

The attached revision of BD.7.03 contains the following changes.

1. The use of passwords has been dropped. They provide little security beyond minimizing the chances of accidental returns out of phase with calls. A simple scheme using a linear index has been substituted.

2. A master mode, slave access procedure must have a slave procedure linkage section in order to be called by a slave procedure. On the other hand a master mode, master access procedure cannot be called by a transfer instruction in such a linkage section. Thus the transfer instruction must actually be in the master mode procedure.

Published: 07/14/67
(Supersedes: BD.7.03, 01/20/66;
BD.7.03, 09/20/66)

Identification

CALL, SAVE, and RETURN Sequences for Execute-Only and Master Mode Procedures.
R. Montrose Graham

Purpose

Arbitrary entries to a procedure may cause it to become an unwitting accomplice to a disastrous act. In addition it is desirable that the renter of a private procedure be prevented from reading the code - else he could make a copy to use free of rent. To prevent this the GE 645 hardware recognizes two special types of procedure segments: execute-only, (EØ), and Master Mode (MM). These segments can be referenced externally only by a transfer of control to the first word of the segment. Once inside the segment, valid internal references are determined by the other access controls (e.g., read/write, slave/master). It is desirable that the caller need not know the type of the called procedure and inversely that the called procedure need not know the type of the caller. To achieve these goals, standard call and save sequences have been adopted. The return sequence is identical to that for ordinary slave (ØS) procedures; see section BD.7.02. The standard sequences defined herein permit an EØ or MM procedure to have multiple entry points.

Stack Usage and Argument Lists

Stack usage is identical to that of ØS procedures except that sp|22 is used to store an index which verifies the return when calling out of an EØ or MM procedure. The format of argument lists is the same as it is for ØS procedures. However, since an EØ or MM procedure may not be referred to from outside as data, the writer must arrange that neither the argument list nor any of the arguments of a call out are located in the EØ or MM procedure.

The Save Sequence (Entry Points)

Just as in ØS procedures, the save sequence is distributed between the procedure and its linkage section. However, since entry to the EØ or MM procedure must be at its physically first location, multiple entry points are achieved by setting an index in that portion of the save sequence which is in the linkage section and transferring to the beginning of the procedure segment. Here is found a prologue which checks the validity of the index and then fans out to the true entry point via a transfer vector. Figure 1 shows the linkage section and prologue for an EØ or MM procedure with m+1 entry points and pseudo-entry points (which are used for returns and will be explained below). Just as in the case of ØS procedures, the original contents of the link at in is a pointer to the link definition with an fi modifier and the link is generated by the linker in the normal way. The code at epi loads index register 0 with the numerical index of the corresponding entry point and establishes the linkage pointer. At <a>|0 the entry index is verified. If the index is out of range an error procedure is invoked. If the index is valid control is transferred to the true entry point. The code at pti is the remainder of the save sequence as in ØS procedures preceded by two instructions which restore index register 0 and the indicators which were used in verifying the entry.

The Call Sequence (Calling Out)

When calling an external subroutine a return point must be set up in the linkage section similar to an entry point since the called procedure may not transfer into the middle of an EØ or MM procedure (even with an rtcd instruction). In addition, to prevent accidental entry at a return point by a procedure which was not called by the EØ or MM procedure, an index is used to verify the return. Figure 2 shows the code in an EØ or MM segment <a> for a call out and return and the corresponding contents of its linkage section. The code from callout to pti-1 is the call to an external procedure. The bases and registers are saved and the argument pointer established as in a normal call. A return to a pseudo-entry point in the linkage section is fabricated. The indicators are then saved, the index of this return point is moved into the stack at sp|22, and the q-register and indicators are restored. Control is then transferred to the subroutine being called. Note that the transfer is actually executed in the procedure but that the return is to the linkage section. This insures that if this procedure is MM and the called procedure is Master Access only the transfer will be valid. When the called procedure executes a normal return sequence, control returns to out and index register 0 is loaded with an index, just as if this were a real entry point, and control is transferred to the prologue of the EØ or MM procedure. Finally, control reaches pti via the transfer vector in the prologue. At pti, which follows the original call out, the return index is retrieved from the stack and tested against the original. If they do not match, an error procedure is invoked. If the index checks, the return is valid. The index position in the stack is cleared and the registers which were used are restored.

One other problem remains, that of abnormal return points. This is solved in a fashion similar to the normal return using a pseudo-entry point. An abnormal return point is passed in the form of a label as an argument to the called procedure. The called procedure returns to this point by invoking the unwinder, (BD.9.05). In figure 2, if ptn is an alternate return for the call at callout, then the argument passed to the called procedure must be a pointer to the pseudo-entry point abnrtn in the linkage section. When control actually reaches ptn, the index is verified and registers restored as in the normal return index.

Notes and Comments

The notes about pointers, use of bases, cautions, etc. in Section BD.7.02 also hold for EØ and MM procedures. It is expected that the compilers and assemblers will have a modal switch activated by some declaration such as:

```
        executeonly
        mastermode
```

Using assembly as an example, the assembler would generate the code callout through callend in the segment and out through out + 1 with the link at sub in the linkage section when the macro:

```
        callout: call <lib>|[entry](arglist)
```

was encountered rather than the sequences defined for the call macro in section BD.7.02. The save macro would be modified to produce the code at pti in <a> and epi with the link at in in the linkage section. The assembler would also generate the prologue. There would be a pseudo-op:

```
        abnreturn ptn(callout)
```

which would declare _ptn_ an abnormal return point for the call labeled _callout_. This pseudo-op would produce a class 4 definition for _ptn_ (see BD.7.01) in either mode of assembly. In EØ or MM mode this pseudo-op would also produce a pseudo-entry for _ptn_ and produce the appropriate code. In setting up _ptn_ as an alternate return, the symbol _callout_ declares that the expected password upon return is the same as the one set for the call at _callout_, i.e., _ptn_ is an abnormal return for the call of _callout_. With these features in the assembler, it will be possible, in most cases, to change a procedure from ØS to EØ (or MM) (or vice-versa) by merely adding the declaration _executeonly_ and reassembling.

The use of the return index reduces the probability of accidentally causing either, 1) a return which is out of sequence or 2) a return when no unsatisfied return remains. The use of return indices, or even passwords, cannot prevent a user from deliberately causing either of the above events. The writer of an EØ procedure should keep this in mind. The execute only feature does prevent entry except at those entry and return points specified by the writer. It also prevents any user from making a copy of the procedure. On the other hand there is no way to prevent a user from copying the linkage section. This means that knowledge of what procedures and data segments an EØ procedure references can not be kept secret. In addition, there is no way to prevent the user from writing in the linkage section. This means that the user can substitute his own version of any segments referred to by doctoring the EØ procedure's linkage section.

Linkage Section for <a>

```
      ...
      ...
epi:  ldx0     i,du
      eaplp    -*,ic
      tra      in-*,ic*

      ...
      ...
```

in:			
	a#		its
	0		0

EØ/MM Segment <a>

```
<a>|0      cmpx0    top-bottom,du
           tnc      bottom,0
           call     <mmerror>|[entry]
           ...
           ...
bottom:    tra      pt0
           ...
           tra      pti
           ...
           tra      ptm
top:       syn      *
           ...
           ...
pti:       ldx0     sp|8
           ldi      sp|21
           eapbp    sp|18,*
           stpsp    bp|16
           eapbp    bp|t
           stpbp    bp|18-t
           eabsp    bp|-t
           stpap    sp|26
           ...
           ...
```

Figure 1

Linkage Section for <a>

```
            ...
            ...

out:        ldx0      j,du
            tra       in-*,ic*

            ...
            ...

 in:       | a#        |    | its |
           | 0         |    | 0   |

            ...
            ...

sub:       | lib#      |    | its |
           | entry     |    | 0   |

            ...
            ...
abnrtn:     ldx0      n,du
            eaplp     -*,ic
            tra       in-*,ic*
            ...
            ...
```

EØ/MM Segment <a>

```
            ...
            ...
callout:    stb       sp|0
            ~~sreg      sp|8~~
            eapap     arglist
            eapbp     lp|out
            stpbp     sp|20
            sti       sp|21
            ldq       =j
            stq       sp|22
            ldq       sp|13
            ldi       sp|21
            tra       lp|sub,*
ptj:        ldq       sp|22
            cmpq      =j
            tnz       rtnerror
            stz       sp|22
            ldq       sp|13
            ldx0      sp|8
callend:    ldi       sp|21
            ...
            ...
ptn:        ldq       sp|22
            cmpq      =j
            tnz       rtnerror
            stz       sp|22
            ldq       sp|13
            ldx0      sp|8
            ldi       sp|21
            ...
            ...
rtnerror:   call <mmerror>|[return]
            ...
            ...
```

Figure 2