The write-up of who-called has been altered so that when the user wants return information from a stack frame which also indicates a ring crossing, he is given that information. Previously who_called not only would not cross rings in search of the desired return information, but also would not divulge the return information at a ring crossing.

Changes are:

1) Appropriate explanations added in the first paragraph of implementation;

2) steps 3 and 4 are interchanged to allow the information to be returned.
Identification
who_called
R. J. Sobecki

Purpose
The procedure who_called provides a convenient way for
the user to obtain a pointer into the procedure which
was in control a number of Stack frames back from the
current one. The actual value of the pointer is the location
in that procedure to which control will eventually return.

Usage
\[\text{ptr1} = \text{who_called(n)};\]

Appropriate declarations for the above are:
\[
dcl (\text{ptr1, who_called ext entry (fixed bin (17))}) \text{ ptr,}
\text{n fixed bin (17)};\]

The procedure who_called is used to trace back \(n\) Stack
frames from its own to determine which procedure was the
caller at that Stack frame level. Who_called operates
in the ring from which it is called as do the error handling
procedures which call who_called (BY.11). It is possible
that who_called will encounter a ring-crossing Stack frame
in searching back through the Stack corresponding to who_called's
ring number. If this happens who_called returns a null
pointer value in \(\text{ptr1}\). If who_called exhausts the Stack
a null pointer value is also returned in \(\text{ptr1}\). Thus,
a null pointer value may be construed to mean that there
is no Stack frame \(n\) levels back which the caller is allowed
to know about. If who_called is called with \(n<0\), the
absolute value of \(n\) is used. Note that if procedure alpha
calls who_called with \(n = 0\), \(\text{ptr1}\) is a pointer into alpha.
If \(n = 1\), \(\text{ptr1}\) is a pointer into the caller of alpha.

Implementation
Who_called first checks \(n\); If \(n<0\), the absolute value
of \(n\) is used. Who_called then proceeds to step down \(n\)
Stack frames using the back pointer stored by the Multics
save mechanism (see BD.7.02). The return information
in the \(n\)th Stack frame back from the current one (who_called's)
is stored into \(\text{ptr1}\). \(\text{ptr1}\) then has an offset which is
the point of return to the calling procedure (see BD.7.02). As each successive back pointer is examined in each Stack frame cycled through, the back pointer’s op code field (bits 19-27 of the first word) is tested: If the op code field is equal to "000000001"b, this Stack frame is a dummy and indicates that a ring crossing took place at this point in the Stack. In this case if the nth stack frame has not yet been reached who_called returns with a null pointer as the return value of ptr1. Note that if the dummy frame is the nth, its return information will be returned. Who_called merely refuses to search through the next appropriate stack. Also, as each Stack frame is cycled through, its back pointer is examined for a null pointer. If the back pointer is null the base of the Stack has been reached and who_called returns with a null pointer as the value of ptr1. See figure 1 for the layout of a Stack frame.

Who_called is coded in EPLBSA because of its need to investigate the Stack. The following notes document the coding.

name who_called

segdef who_called

1)  * Place the absolute value of the argument n in x2
    * (index 2).
    * x2 controls the loop which steps back [n] Stack frames.

who_called:  save
            1x12  ap ↑ 2,*
            tpi  next
            erx2  -1,du
            adlx2  1,du

2)  * Obtain who_called’s Stack frame pointer in bp→bb.

next:  stpsp  base
       eapbp  base,*

3)  * Go to windup if [n] Stack frames have been stepped through.

       sb1x2  1,du
       tmi    windup
4) Transfer to nulrtn (return null pointer) if this
   Stack frame is a place holder for a ring crossing,
   (i.e., bits 19-27 of first word of previous Stack
   frame pointer = 0000000001).

   start: ldaq rngmsk
          cmk    bp ↑ 16
          tze    nulrtn

5) Temporary code to determine if previous Stack frame
   pointer = "0"b, which currently means that the base
   of the Stack has been reached.

   ldaq    bp ↑ 16
   tze     nulrtn

6) Nulptr is a Multics null pointer constant. If the
   previous Stack frame pointer (bp ↑ 16) is a null
   pointer the base of the Stack has been reached.

   ldaq    nulptr
   cmpaq   bp ↑ 16
   tze     nulrtn

7) Set bp→bb to point to the previous Stack frame.
   Transfer to the start of the loop which examines each
   Stack frame.

   eapbp   bp ↑ 16,*
   tra     start

8) The nulrtn identifier causes a null pointer to be
    returned as the second argument. Code at windup
    identifier causes the contents of bp→bb (usually a
    pointer to the Stack frame [n] levels previous to
    who_called's Stack frame) to be stored in the return
    argument position of who_called.

   nulrtn: eapbp nulptr,*
           tra    return
   windup: eapbp bp ↑ 20,*
           return: stpbp ap ↑ 4,*

* Constants follow:

    tempd base
    even

* Null pointer constant used in steps 6 and 8.

   nulptr: oct 777777000043
           oct 000001000000
The double word constant rngmsk is used as follows:

A register = first word (A register has bits 19-27 = 000000001).

Q register = second word which causes comparison only between bits 19-27 of A register and bp ↑16 in step 3.

rngmsk: oct  000000001000
         oct  777777000777
         end
Figure 1: `sp→sb` setting is used to get pointer to base of current Stack frame.