TO: MSPM Distribution

FROM: D. E. Joel

SUBJECT: 645 Simulator Escape (BE.7.10)

DATE: October 25, 1966

The change to the following document is the addition of a discussion on the escape mechanism used for dynamic loading of segments at execution time.

Published: 11/2/66
Supersedes: BE.7.10, 5/13/66;
BE.7.10, 7/22/66;
BE.7.10, 7/29/66.

Identification

645 Simulator Escape
D. E. Joel

Purpose

The concept of escape coding was introduced so that a 645 process under simulation may, in effect, call for the execution of a 635 subroutine in the GECOS environment. The primary purpose of escape coding is to provide an input/output facility. A second purpose is to provide a termination point for a 645 process.

Organization on the 645 side

A general escape routine - procedure segment escape - is provided whose function it is to perform the actual communication between the 645 process and the 635 GECOS environment. Segment escape has two entries:

1. finish - the user calls this entry in order to terminate his process.

   In EPL the call appears as

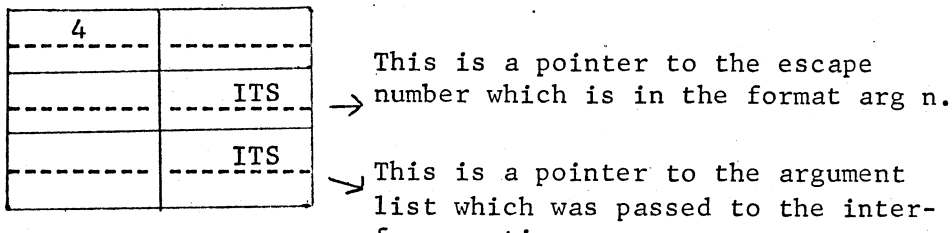
   call escape$finish

2. escape - This entry is called by any of a set of interface routines which are provided so that the user does not have to be aware of the requirements of segment escape. The call used by an interface routine is (in assembly language)

   call <escape>|[escape] (arglist)

   The arglist is as follows:

arglist:



To provide easy communication with the 635 GECOS subroutines (both under simulation and later on the 645 itself using GECOS) a fixed segment (number 5) has been used. This segment (named memory) describes the entire 635 slave memory as an unpaged segment consisting of 1024 word blocks.

Segment escape passes information to the 635 GECOS subroutines by placing the ITS pair described above at arglist + 4 into location 240 (decimal) in segment memory, and then passing control to an escape vector subroutine in 635 memory with the escape number in index register 7.

Organization on the 635 side

When the escape vector subroutine gets control, it checks to see that the escape number is valid. If valid, control is passed to the appropriate subroutine; otherwise, the escape is treated as Null, and control is returned to the 645 process with no action being taken. Index register 1 contains the location to which the 635 subroutine must return after performing its functions.

A 635 subroutine named TRNSLT is provided whose function is to transform a 645 address into a 635 address in segment memory. The call (in GMAP code) is:

```
CALL    TRNSLT (EXTERN, INTERN, RESULT, STATUS)
```

where:

EXTERN is an external base register stored im memory

INTERN is an internal base register stored in memory

RESULT is the location in which the 635 address is to be stored

STATUS is a location at which TRNSLT stores either zero (successful) or non-zero (unseccessful)

Example: determine the location of the argument list passed from the 645 process. If successful, pick up the first two words into the AQ registers. If not successful, go to location NØGØØD.

```
    REM  ITS POINTER TO ARGUMENT LIST

    REM  IS AT LOCATION 240.

    CALL  TRNSLT (240, 241, RESULT, STATUS)

    USE   REMOTE
```

```
STATUS   DEC    0 .          DEFINE STATUS WORD REMOTELY
         USE
         SZN    STATUS       CHECK TRNSLT STATUS
         TNZ    NOGOOD       EXIT IF UNSUCCESSFUL
RESULT   LDAQ   **           PICK UP TWO WORDS OF LIST
```

Note that the user should be aware of page boundaries in the 645 process memory, and plan translation of addresses accordingly.

Escape coding currently implemented

1. Pseudo - file system (See BE.10)

2. Pseudo - console (See BE.11)

3. Message writing to the global error file.

   The function provided is that of writing a message onto the global error file. The message is passed as a character string. A new line character is provided at the end of the message.

   The call in EPL is

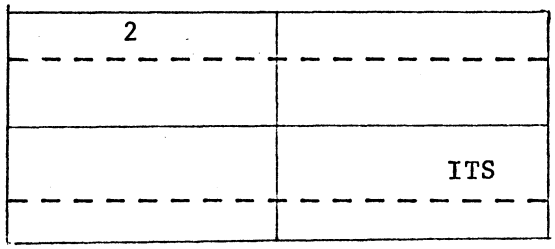
   call messag$messag ('character string')

   where messag is the interface routine with entry point messag.

   The call in assembly language is

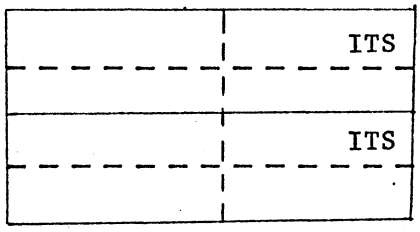
   call <messag>|[messag] (argument)

   argument:



Specifier pointer

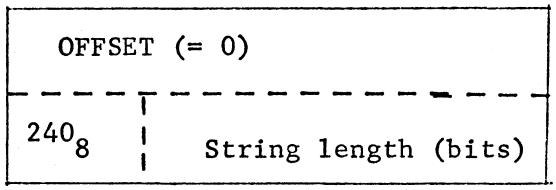
Specifier:



Pointer to character string

Pointer to dope.

Dope:



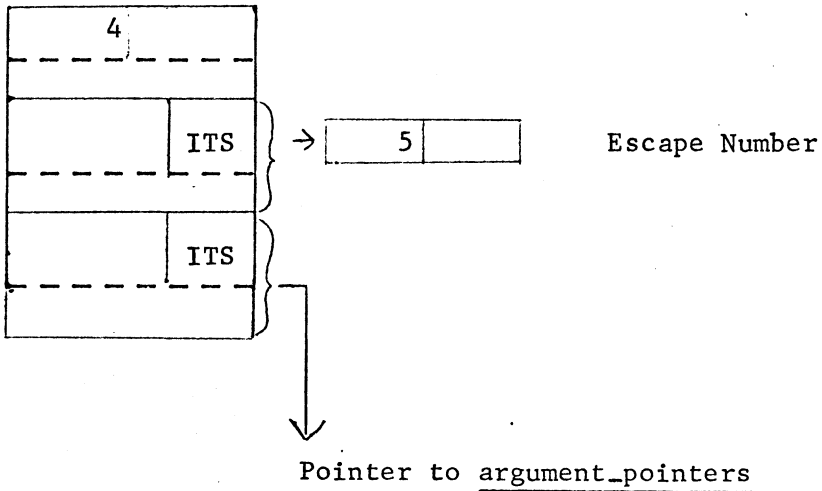
4. Dynamic loading

The function provided is that of loading a segment from the Segment Library into 645 core at defined locations.

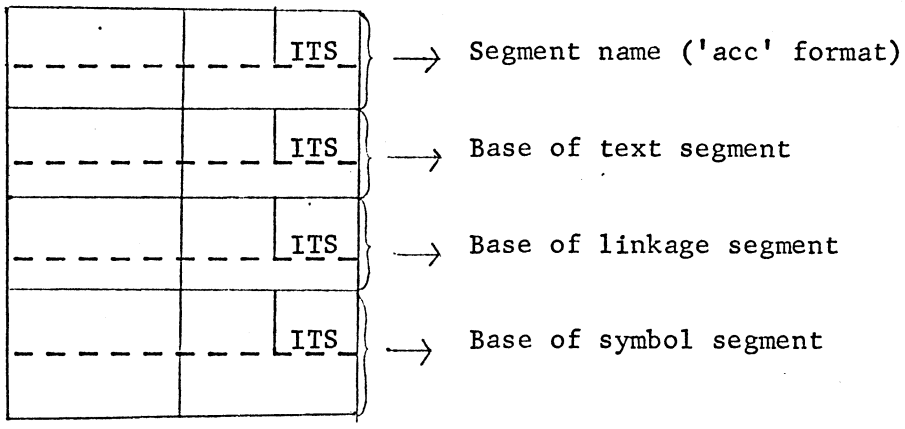
Dynamic loading is invoked by segment search which makes a direct call to segment escape:

call <escape>|[escape] (arglist)

arglist:



Argument_pointers:



Note: if the segment number for the text, link or symbol segment is zero, then that particular segment will not be loaded.