Published: 12/15/67

Identification

Addendum to BX.17.01
E. W. Meyer, Jr.

Purpose

The implementation of the loader invoked by the gecos_seg command is discussed.

Implementation

The loader which operates as part of the gecos_seg command is a one pass loader which loads successive GMAP programs into a created Multics segment from the highest location downward and makes proper interprogram links. None of the GECOS standard options or debugging facilities are available, nor is a library search made. Only the preface and text cards are recognized; all others are ignored. On-line error messages are confined to errors within the loader program itself, such fatal errors as segment full or misplaced preface card, and a listing of all undefined SYMREFS following the loading of the program.

The control lines must be ordered as follows:

(a) one "segment_size n" control line

(b) one or more "object beta" control lines.

(c) any number or none of the following (in any order):

"reference_in gamma delta"

"call_in gamma delta"

"call_out gamma delta"

The loading process operates by sequentially reading the control lines and performing the action designated by each.

The control line

segment size n

creates and initiates text and linkage segments alpha and alpha.link in the process directory, deleting any old copies which may exist. It also initializes the loader by setting to zero the length of the currently allocated BLANK COMMON region (which runs downward from <alpha>|0) and the program region (which runs downward from <alpha>|n), and clearing the definitions and linkage tables (see below).

The control line

"object beta"

initiates the object deck segment beta.635object and starts the loader, reading preface and text card images from the object segment and processing them.

The processing of the preface and text cards involves the use of three internal tables declared within the procedure "tbl":

Definitions Table (def)

```
dcl 1 def (d_top) based (d_pntr),

      2 sym bit (36),              /* external symbol */

      2 val fixed bin (18),        /* value of sym */

      2 xlk fixed bin (18),        /* index to undefined
                                      symbol chain */

      2 sdef bit (1);              /* defined switch */
```

"def" is a linear array of substructures, each consisting of four elements:

sym - a six character bcd symbol designated as a SYMDEF, SYMREF, or LABELED COMMON type external symbol within a preface card.

val - the relocated value of this symbol. In the case of a SYMREF, the value is undefined prior to encountering the corresponding SYMDEF.

xlk - zero unless the symbol is undefined and fields involving it have been loaded into the created procedure segment. In that case xlk is an index to a list in the lnk table (see below) of the fields in the loaded program which require the value of this symbol.

sdef - a switch indicating whether or not the value has been defined. "0"b = undefined; "1"b = defined.

It is appended to whenever an external symbol entry which does not already exist in def is encountered on a preface card. The definitions table is maintained throughout the loading process; it is cleared only through the action of the "segment_size n" control line.

Reference Table (ref)

```
dcl ref (r_top) fixed bin (18) based (r_pntr);
```

The reference table is a linear array of indices to symbol substructures within "def". It is cleared upon encountering the first preface card of a new subprogram. Each SYMREF or LABELED COMMON symbol encountered thereafter within the preface card group causes an index to the symbol's position in "def" to be appended onto the reference table. Thus a text card entry using the jth external symbol reference declared within the preface card(s) need only refer to it by the number j. The symbol's position in "def" can be picked up from ref(j).

Linkage Table (lnk)

```
dcl 1 lnk_st(l_top) based (l_pntr),

      2 xlnk fixed bin (18),        /* index to next
                                       substructure in
                                       the list */

      2 xtseg fixed bin (18),       /* index to loaded
                                       program word */

      2 l_r bit (1),                /* left/right switch */

      2 p_m bit (1);                /* plus/minus switch */
```

It may happen that during the loading process a program with SYMREF external references is loaded before all those SYMREFs have been defined via SYMDEFs in the preface cards of other subprograms. Whenever a text card entry that uses such a SYMREF is encountered, the "def" entry for that symbol will be found to have an undefined value.

If this is to be a one pass loader, there must be a way to load these fields into the text segment as they are encountered and later relocate them when the SYMREF is defined with its SYMDEF in a subsequent program. A solution to this problem is to load the absolute value of the addend into the program field and to make an entry under a list for that symbol consisting of the following:

(1) the address of the field relative to the beginning of the text segment.

(2) a switch setting indicating whether the field is in the left or right half of the word.

(3) a switch setting indicating whether the sign of the addend is plus or minus.

When the external symbol is eventually defined by a SYMDEF entry in a subsequent program's preface card, the loader will go through the loaded field list for that symbol and properly relocate the listed fields using the newly defined value of the symbol. Subsequent fields using that SYMREF will be relocated and loaded directly.

The linkage table contains the loaded field lists of all undefined SYMREFs. It is essentially a free pool of substructures within an allocated structure. Each substructure is either unused (in which case it is chained to a free substructure list) or it is an entry in the loaded field list of some undefined SYMREF. Each substructure that is part of such a list contains the three elements "xtseg", "l_r", and "p_m" describing the position of the field requiring relocation and the sign of its addend, and also the index "xlnk" to the next entry in the list. The last entry in each list signals this fact with "xlnk" = 0. An index to the first entry in each list is contained in the substructure "xlk" of the corresponding SYMREF's entry in the definitions table.

Figure 1 illustrates the relationship between "ref", "def", "lnk", and the program segment.

The Preface Card

A preface card signals the beginning of a new subprogram. When it is encountered, space for the new program is allocated in an area of the text segment just below the previously allocated program region. The BLANK COMMON length becomes the maximum of the current length and the requirements of the new program. If the new program and BLANK COMMON regions intersect following this allocation, loading is discontinued and a "segment full" error is signaled.

Next the external symbols on the preface card are processed. The class code of each entry is checked to determine whether the symbol is a SYMDEF, SYMREF, or LABELED COMMON, and the indicated action is taken:

SYMDEF (class code = 0 or 1) -- def is searched for the symbol. If it does not exist a new structure is appended onto def to hold the symbol and the value supplied in its preface entry. If it exists and is already defined, no action is taken. If it exists in an undefined status (previously appended by a SYMREF) the SYMDEF's value is inserted and any existing loaded field list is operated upon.

SYMREF (class code = 5) -- def is searched for the symbol; if it isn't found, an entry for it with undefined status set is appended onto def. In either case, the index to the symbol's position in def is then appended onto the reference table.

LABELED COMMON (class code = 6 or 7) -- def is searched for the symbol. If it is not found in def, space is allocated for the labeled common at the bottom of the currently allocated program region (check for BLANK COMMON and program region intersection; if so, do a "segment full" error return), and a structure for the symbol is appended onto def with "val" set to the starting cell of the LABELED COMMON area. If the LABELED COMMON symbol is originally found in def, no program space is allocated as a previous allocation applies. In both cases an index to the symbol's position in def is appended onto ref.

All external symbol entries are processed in this manner. When the preface card is exhausted a check is made to determine whether or not more entries are expected on immediately following preface cards. If so the next card is read in (any type other than preface card generates an error). Otherwise it is assumed that text cards for the current program follow. The next preface card encountered terminates the loading of the current program and allocates space and initiates loading of the new program.

The Text Card

Whenever a text card is encountered in the object segment the program area allocation and external symbol declarations of the immediately preceeding preface card(s) apply. The header of the text card provides information as to whether the text card entries are to be relocated and loaded into the current program area or into one of the LABELED COMMON regions declared in the preface card(s). It also provides a loading address relative to the beginning of that region for the first text entry. Subsequent entries are loaded into sequentially higher locations.

Relocation and loading of the text card entries occurs as described in pp. 21-23 of the G.E. General Loader Manual CPB-1008D. Whenever a field involving an undefined SYMREF is encountered, the absolute value of the addend is inserted into the field and a structure pointing to the loaded field is appended onto the symbol's external linkage chain. These fields are relocated when the symbol is defined.

It may happen that another header word follows the text entries of the previous header on the card. In this case, reload the header information and process the following text entries accordingly.

The control line

reference_in gamma delta

causes a search of def for the 36 bit GMAP representation of delta and the creation in the external symbol table of the ascii symbol gamma and the value of delta found in def. If delta is not found in def or its value is undefined an error is generated.

The control line

call_in gamma delta

causes a search of def for delta. If it is not found or is undefined, an error is generated. Otherwise a Multics save sequence, a GMAP TSX1 <value of delta in def> call instruction, and a Multics return sequence are appended onto the bottom of the program region. If this addition to the program region causes the allocated BLANK COMMON and a program regions to intersect, a "segment full" error is generated. An entry for gamma and a linkage to the new call_in sequence is created in alpha's linkage section.

The control line

call_out gamma delta

causes a Multics calling sequence to the external entry gamma to be created in the text segment under construction. The first instruction of this sequence has the SYMDEF delta.

The following GMAP calling sequence is required:

```
loc     TSX1 delta    (delta is a SYMREF)

loc+1   <argument>

loc+2   <return location>
```

The instruction sequence created and loaded under SYMDEF delta does the following:

(a) saves the return address

(b) creates a one-argument list consisting of the argument count plus an its pair pointing to loc+1;

(c) performs a standard Multics call to the external entry point gamma

(d) (after returning from gamma) returns to loc+2.

After processing all control lines, if no undefined symbols remain in def, the segments alpha and alpha.link are transfered from the process directory to the working directory. Otherwise the segments remain in the process directory and a listing of all undefined external symbols is printed out on the user's console.

Options and Error Signaling

The only option relevant to the gecos_seg command is "brief". If it is set on the load map listing will be deleted and error comments kept to a bare minimum. Gecos_seg uses the standard error handling mechanism as described in MSPM sections BY.11.00 - BY.11.04. Specific error codes will be reported in a later addition to this section.

Procedure Segments of the Loader Program

gecos_seg

is the upper level procedure of the loader and is invoked by the shell upon receipt of a gecos_seg command.

It initiates the control line segment alpha.gecos_seg and reads the control lines, calling the proper procedure to handle each. Upon exhaustion of the control lines, it transfers alpha and alpha.link to the working directory or prints an undefined symbol listing.

tseg

processes the control line "segment_size n" by creating the segment alpha, calling link$init to create the segment alpha.link, and clearing the definitions table. It contains entries to retrieve and insert cells in the text segment alpha.

link

contains entries to create the linkage segment and to add linkage definitions, external symbols, and external entries to the segment.

ext

processes the control lines "reference_in", "call_in" and "call_out".

object

processes the control line "object beta". It calls "tob" (see below) to initiate the object deck segment, and reads the object deck card images, calling "pref" or "text" (see below) upon recognition of a preface or text card respectively.

pref

is called by "object" to process a preface card. If the preface card indicates that related preface cards follow, it reads these cards itself, and does not return to "object" until all related preface cards have be n processed.

text

is called by "object" to process a single text card. It returns to "object" after all entries on this card have been relocated and loaded.

next

is called by "text" to obtain the fields and relocation data of the next text card entry.

reloc

is called by "text" to relocate the fields of a text card entry.

tob

is the segment invoked to deal with card images of the object deck segment. It contains entries to initiate the object deck segment beta.635object, to advance to the next card image of the segment, and to retrieve data from the current card image.

tbl

contains all the entries involved in initializing and referencing the internal tables "def", "ref", and "lnk".

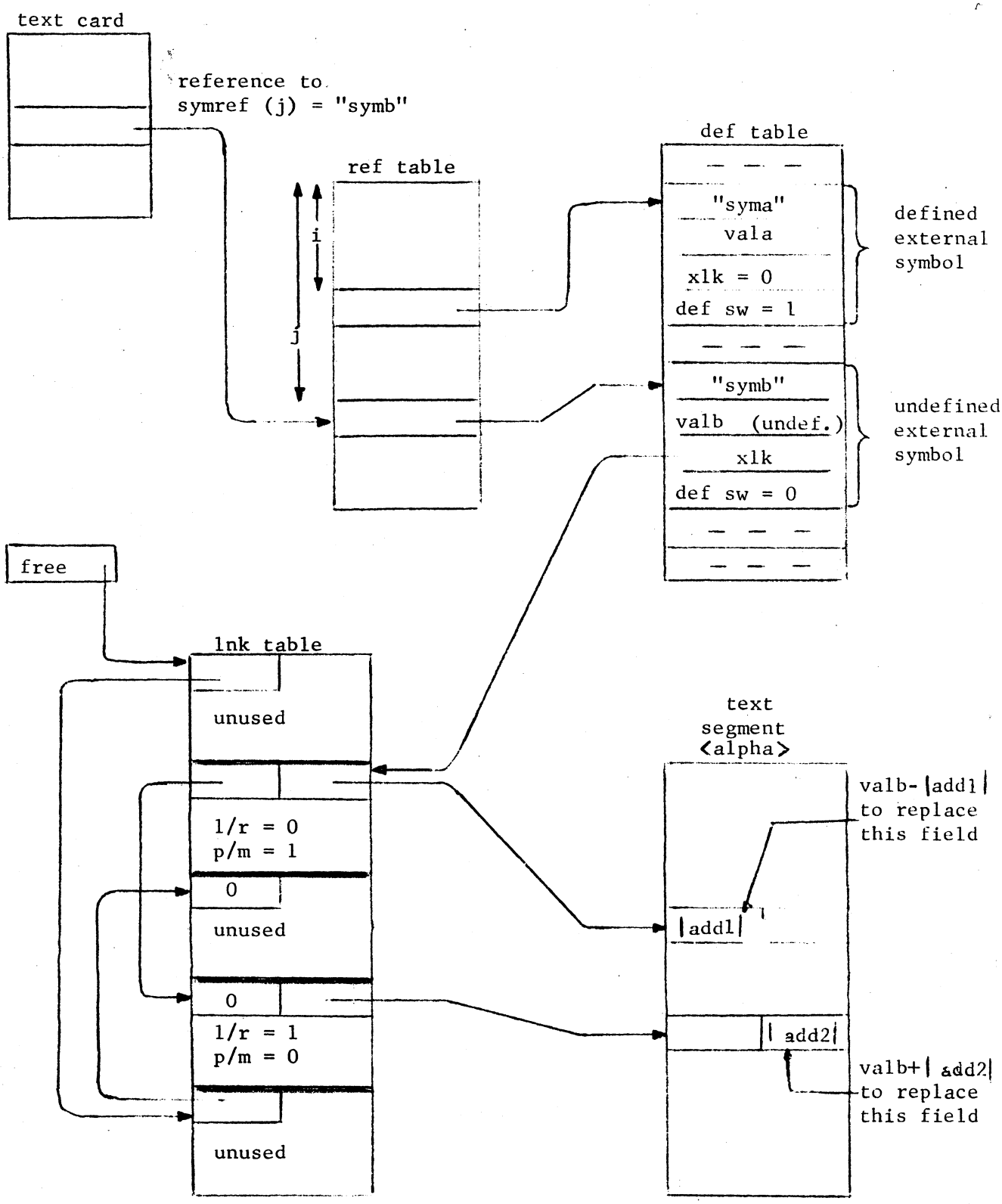

Figure 1. Relationships among the Internal Tables and the Text Segment.