TO: MSPM Distribution
FROM: R. M. Graham
SUBJECT: BD.7.01
DATE: 02/12/68

The attached issue of BD.7.01 supersedes the issue of 05/12/67 and the Appendix of 05/29/67, and contains the following revisions:

1. The location of the static block has been changed.

2. The linkage block threads now terminate with null pointers.

3. The calling sequence to trap routines has been changed.

4. Entries in the linkage section now contain a usage counter for performance monitoring.

5. Gates and doors (used by the protection mechanism) have been added.

Published: 02/12/68
(Supersedes: BD.7.01, 05/18/67;
BD.7.01, 03/07/67;
BD.7.01, 06/30/66;
BD.7.01, 04/08/66)

Identification

Linkage Section
R. M. Graham

Purpose

To permit symbolic inter-segment references and dynamic linking during execution, it is necessary that the symbolic information needed to assemble a final machine address be available at execution time. The linkage section contains the needed information. This section describes the structure of the linkage section.

A segment <d> is actually a collection of up to three segments; <d>, <d.link>, and <d.symbol>. <d.link> contains part or all of the linkage section and <d.symbol> contains a symbol table and parameter type information (BD.1.00), binding information (BD.2.01), unlinking information, and other supporting information.

The principle users of the linkage section are the linker and its supporting linkage maintenance routines. In particular, the linker has the option of combining a number of linkage sections into a single segment. This combining takes place without any relocation of any items in the linkage section. This means all "addresses" in the linkage section must be self relative (relative to the IC) or relative to the header (relative to the lp base).

The Linkage Section

Language processors and binders produce a linkage section consisting of one or more linkage blocks threaded together with two pointer threads. Each linkage block begins with an eight word header. This is followed by a (possibly empty) block of static storage. The block terminates with the links. The format of the header is,

0	definitions pointer (def_ptr)	
2	next block pointer(nxt_blk_ptr)	
4	preceding block pointer(pre_blk_ptr)	
6	link_begin/0	block_length
7	segment_number/0	segment_length/0

Segment_length is zero in all blocks but the first.

link_begin	The location, relative to the first word of the header, of the links. If link_begin equals 8 there is no static storage. The block of words between the header and the links is for use by <d> (if it is a procedure) as internal storage of the type called "internal static" in PL/I and "own" in ALGOL.
block_length	is the length of this linkage block (including the header and the static block).
segment_number	is the number of the segment <d>, i.e., d#. This will be set by the linker for its own use.
segment_length	is the assembled length of the segment <d.link>.

The next and preceding block pointers form the two threads connecting the linkage blocks. The nxt_blk_ptrs form one thread (the nxt_blk_ptr = null in the last block) and the pre_blk_ptrs form the other thread (the pre_blk_ptr = null in the first block). The static block is followed by the links (including the entries). The def_ptr points to the beginning of the external symbol definitions and the link definitions. If def_ptr is zero, there are no definitions. The definitions need not be in the same segment as the links and entries. Further, it is possible for any block to be in a different segment. Figure 1 shows an example of a linkage section with several linkage blocks. Figure 2 shows the layout for one linkage block. In general, initially a segment has only one or possibly two linkage blocks, any others getting added during execution. The initial block is the one normally pointed to by the base pair (lb◄-lp).

Links and Link Definitions

For every distinct external reference from within a segment at least one link occurs in the linkage section. A link initially contains an ft2 modifier (see Figure 2) which will cause a fault to the linker when the link is first referenced. External references are made indirectly through the link. For example, suppose that the pair of words at <d.link>|dk2 in Figure 2 is a link for the external reference <s>|[x] and that the value of (lb◄-lp) is <d.link>|d.lp, then the instruction,

```
lda  lp|dk2-d.lp,*
```

will load the A-register with the contents of <s>|[x]. The linker replaces the original contents of the link by an its pair which points directly to the referenced location (this replacement is called linking). Figure 3 shows the different types of external and self references and the contents of the link after linking. In all types either exp or m or both may be omitted, in which case they are zero in the link. Before linking, the link contains two pointers; the head_ptr and the exp_ptr. The head_ptr points back to the beginning of the header for this block. The exp_ptr is relative to the def_ptr; i.e., def_ptr + exp_ptr is the location of the beginning of the link definition.

The link definition always contains an expression word and a type pair. In addition, there may be a segment name and/or external symbol and/or a trap word. The right half of the expression word contains the 18-bit value of exp to be used in calculating the address part of the link (the left half of the second word of the its pair). The left half of the expression word is a pointer to the type pair. All pointers in the definitions are relative to the def_ptr.

The left half of the first word of the type pair is the type # of the reference, as defined in Figure 3. The left half of the second word is either the base (in the left most 3 bits when type# = 2), the seg_ptr which points to a segment name (when type# = 3 or 4), or the self-reference type (when type# = 1 or 5). The right half of the second word is either zero (when type# = 3 or 1) or the ext_ptr which points to an external symbol. Segment names and external symbols are stored as variable length character strings with the character count stored in the first 9-bit field and the last word filled out with the null character, (000)8; e.g., "matrix" is stored:

(006)8	m	a	t
r	i	x	(000)8

The use of a type 1 or type 5 link in <d.link> permits self-reference without explicitly mentioning the segment name. A type 1 link is similar to the type 3 except that the seg_ptr indicates which of the segments <d>, <d.link>, or <d.symbol> is being referenced. The following code is used:

seg ptr value	segment being referenced
0	<d>
1	<d.link>
2	<d.symbol>

A type 5 link is similar to the type 4, however seg_ptr is ignored since the definition for the external symbol indicates the appropriate segment.

The right half of the first word of the type pair is either zero or contains the trap_ptr, which points to the trap word. This word contains two pointers; the call_ptr and the arg_ptr. Both of these pointers are relative to the origin of the header and point to links. The trap_ptr is non-zero when the user wishes to gain control from the linker before the linker attempts to generate the link. The call_ptr points (indirectly through a link) to the procedure entry that is to be called before the link is generated. The arg_ptr points (indirectly through a link) to an argument list which the user wishes to make available to the called procedure. Suppose the user wishes that before the reference <s>|2 is linked, the linker should call <A>|[x] with argument list <D>|[y]. Before generating the link to <s>|2 the linker will execute the call,

```
call  A$x (Dp, mc, fp, trap_flag)
```

where,

```
dcl(Dp,fp)ptr, mc (0:22)bit(36), trap_flag fixed bin (1)
```

Dp a pointer to the beginning of the users argument list.

mc an array into which the linker has stored a copy of the machine conditions as they were at the time of the fault which invoked the linker.

fp a pointer to the faulting pair

trap_flag (not used for trap before link)

Since the call to <A>|[x] goes through a standard link, the call may cause an ft2 fault to the linker. This causes no problem since the linker is a normal multics pure procedure (i.e., recursive and re-entrant). The trap feature is used by various subsystems (e.g., PL/I, FORTRAN) for their own peculiar storage management. Their storage management sub-routines are able to gain control before links are generated, and thus may actually create the segment about to be referenced, combine existing segments, or add new external symbol definitions.

External Symbol Definitions

The def_ptr points to the first external symbol definition. A definition consists of two words preceding the symbol. The left half of the first word contains the next_ext_ptr which points to the next external symbol definition. This pointer is relative to the def_ptr. All of the external symbol definitions are threaded together by the next_ext_ptrs. The thread for this block terminates with a zero word (there may be additional definitions in succeeding linkage blocks). If there are no external symbol definitions, def_ptr points to a zero word. The second word preceding the symbol contains the value of the symbol and its class. A symbolic reference <d>|[x] may refer to any of the three segments <d>, <d.link>, or <d.symbol> depending on the class of the external symbol [x]. If the class is zero or not defined, the value of the symbol is taken to be relative to the origin of <d>. For example, if [x] is defined with class = 0 and value = 25, then <d>|[x] will generate the link:

d#		its
25		

Four class codes have been defined.

1. When class = 1, [x] is a procedure entry point. In this case, value is relative to the origin of the header of the linkage block in which the definition occurs. In Figure 2, if [x] has class = 1 and value = inpoint-d.lp, then <d>|[x] will generate the link:

d.link#		its
inpoint		

Thus, a link is made to the group of instructions at <d.link>|inpoint. The eaplp instruction sets the base pair (lb←lp), to their correct values for the procedure in segment <d>. The tra instruction then transfers (indirectly through a link at <d.link>|in) to the procedure in <d>. The aos instruction increments a counter which is used by the system for performance monitoring.

2. When class = 2, the value of [x] is taken to be relative to the origin of <d.symbol>. For example, if [x] has class= 2 and value = 8, then <d>|[x] will generate the link:

d.symbol#		its
8		

3. When class= 777777(8), this definition is ignored by the linker.

4. When class = 4,[x] is an external label, i.e. a label to which an abnormal return may be made. A class = 4 symbol is identical to a class = 1 symbol in every other respect (including the inclusion of an instruction to load lp, which is its purpose).

When the right half of the first word is non-zero it is a trap_ptr which points to a trap word just as in a link definition. This allows the specified procedure to be executed before the definition is used by the linker. The interpretation of this trap word is identical to that of the trap word for link definitions. Before using the definition the linker will set trap_ptr to zero and execute the call,

```
call A$x (Dp, mc, fp, trap_flag)
```

where,

```
dcl (fp,Dp)ptr, mc(0:22)bit(36), trap_flag fixed bin(1)
```

Upon return from <A>|[x] if trap_flag = 1 trap_ptr is left set to zero, thus preventing the trap from occurring again, or if flag = 0 trap_ptr is reset to its original value. Any definition in which the trap_ptr may be modified during execution, due to the use of the trap feature, must be in the linkage segment. This may require that such a linkage section originally contain at least two linkage blocks. Dp, mc, and fp are the same as for a trap before link. Trap_flag is set by the called trap procedure to indicate if further traps on this definition are to be suppressed.

Gates and Doors

Gates and doors are special kinds of entries required by the protection mechanism. They are, in fact, entries which may be called from outside the ring in which they appear. When called from within their own ring they are indistinguishable from other entries. When called from outside their ring the gatekeeper, in response to a wall crossing fault, requires the information specified in this section. See BD.9 and BD.9.01 for further information about the protection mechanism and an explanation of the information attached to a gate or door. Only the format is specified in this section.

An entry which is a gate or a door is distinguished from a normal entry by the presence of an additional instruction (a nop) in the entry sequence in the linkage section (see figure 7). The address field of this instruction contains a relative pointer (relative to the header) which locates the descriptive information for the gate or door. This information begins with a single word containing the following information,

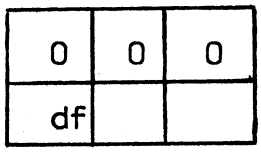
bits	information
0 - 5	The highest ring number permitted to call this entry, i.e., the call bracket.
6 - 7	code indicating if this entry is a gate or a door: 1 - this entry is a gate 2 - this entry is a door
8	code indicating if the gatekeeper is to validate the arguments when this entry is called: 0 - gatekeeper should not validate arguments 1 - gatekeeper is to validate arguments
9 - 17	not used: should be zero
18 - 35	N, the number of arguments for this entry

If the N arguments are to be validated by the gatekeeper (bit 8 = 1) then the descriptive information continues with N/2 words which describe the type of each argument. Each half-word (18 bits) describes one argument,

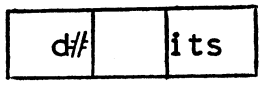
bits	information
0	flag indicating if the gatekeeper is to consider this argument a return value 0 - not a return value 1 - argument is a return value, gatekeeper will copy the final value of this argument back into its original location in the callers ring.
1 - 17	code indicating the data type of this argument. The legal codes are described in BB.2.02.

Linkage Section Before Loading

The links and entries of the initial linkage block for segment <d> are part of the file (segment) "d.link". Segment <d> itself is the file "d". If <d> is pure, as is the case when <d> is a procedure segment, the definitions are normally part of <d>. If <d> is an impure data segment all of the initial linkage block will normally be in <d.link>. If there is more than one linkage block, as there may be if the trap before definition feature is used, the additional blocks will always be in <d.link>. The headers of the initial linkage blocks are included in the file "d.link". If the definitions are in <d>, the first two words have the form:



where <d>|df is the beginning of the definitions. When the segment <d> becomes active, the linker will fill in the first word with



If the definitions are in <d.link>, the first two words have the form:

0		*
k	0	0

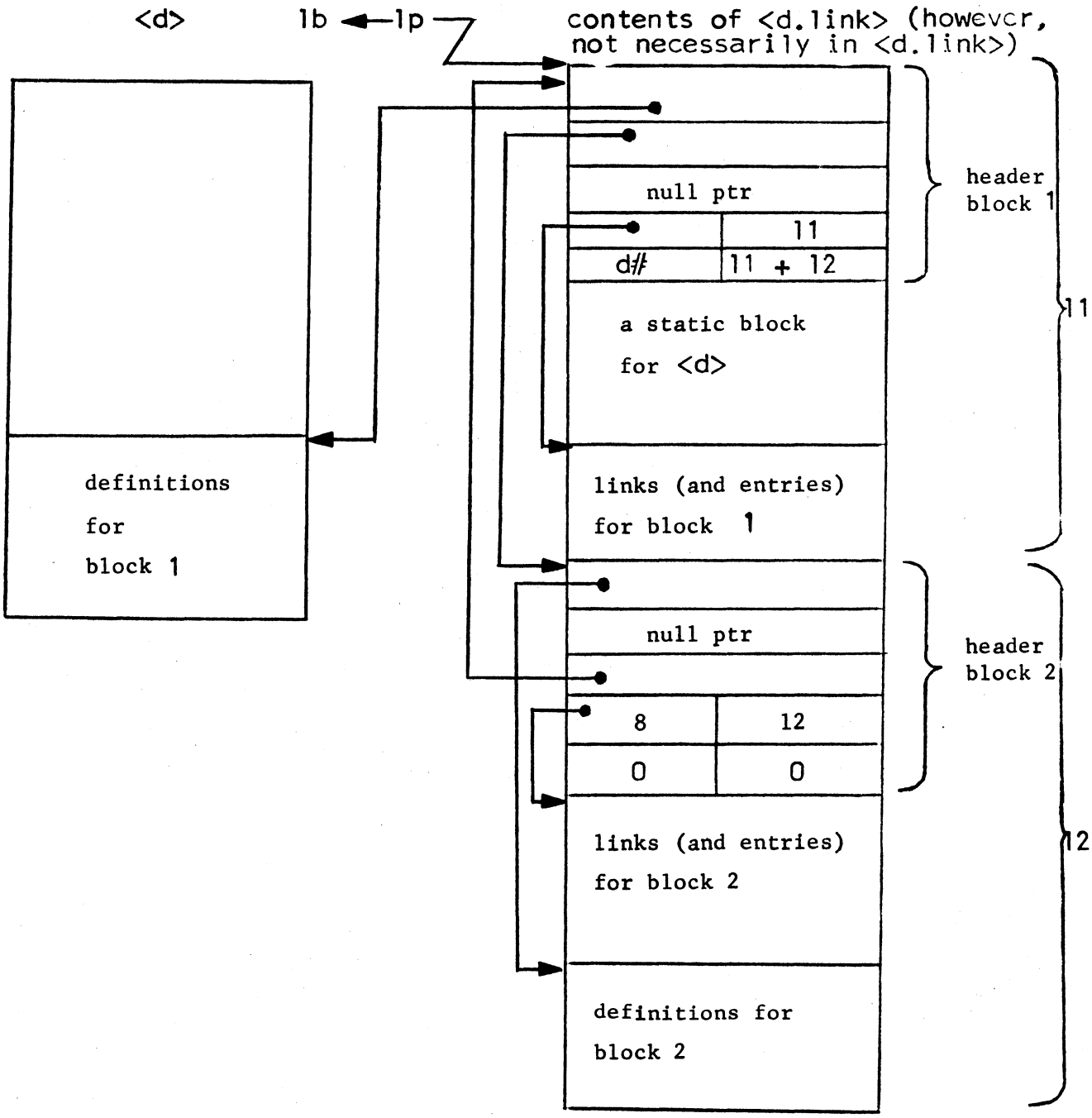
and the definitions begin at <d.link>|k + d.lp. If there is only one linkage block nxt_blk_ptr and pre_blkptr contain null pointers. If there is more than one linkage block nxt_blk_ptr contains,

0		*
nx	0	0

and the second block starts at <d.link>|nx + d.lp.

Example Linkage Section

As an example of the linkage section, consider the EPLBSA program shown in Figure 4. Suppose this is a procedure segment <d>. Figure 5 shows the contents of <d> during execution. Figure 6 shows the contents of the linkage section for <d> immediately after loading and before any links have been established.



Linkage Section for <d>

Figure 1

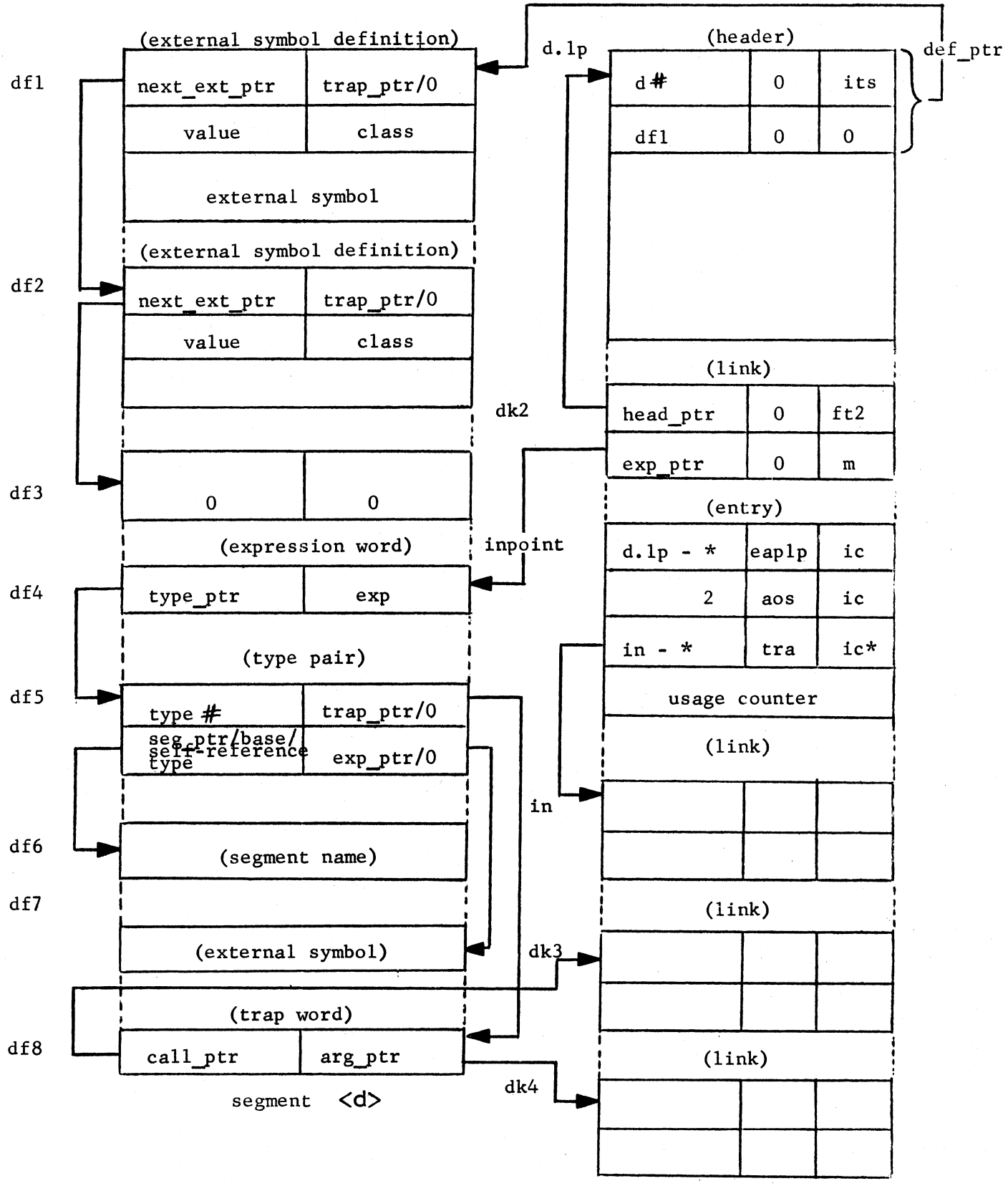

(external symbol definition)
df1
next_ext_ptr
trap_ptr/0
value
class
external symbol
(external symbol definition)
df2
next_ext_ptr
trap_ptr/0
value
class
df3
0
0
(expression word)
df4
type_ptr
exp
(type pair)
df5
type #
trap_ptr/0
seg_ptr/base/
self-reference
type
exp_ptr/0
df6
(segment name)
df7
(external symbol)
(trap word)
df8
call_ptr
arg_ptr
segment <d>
d.lp
(header)
def_ptr
d#
0
its
df1
0
0
(link)
dk2
head_ptr
0
ft2
exp_ptr
0
m
(entry)
inpoint
d.lp - *
eaplp
ic
2
aos
ic
in - *
tra
ic*
usage counter
(link)
in
(link)
dk3
(link)
dk4
Linkage Block for <d>
<d.link>

Figure 2

type #	reference type	generated link	
1	* | exp,m	*/#	its
		exp	m
2	base|[ext]+exp,m	base*2**15	itb
		ext+exp	m
3	<seg>|exp,m	seg #	its
		exp	m
4	<seg>|[ext]+exp,m	seg #	its
		ext+exp	m
5	*|[ext]+exp,m	*/#	its
		ext+exp	m

External Reference Types for links in <d.link>.

*refers to one of the segments <d> <d.link>, or <d.symbol> and seg_ptr identifies which one

Figure 3

```
        segdef      mass
        entry       clc
        segref      dta,x(<all>|[str](<dpe>|0))
clc:    save
        lda         <rat>|[gas]+5,*
        sta         x + 40
        eapbp       <rat>|0
        lda         bp|[heat]
        sta         <rat>|[gas]+5
        lda         <rat>|[gas]
        sta         <rat>|0
        return
mass:   arg         30
        end
```

EPLBSA Program for Procedure Segment <d>

Figure 4

Figure 5: Contents of Segment <d> During Execution

```
<d>     0       eapbp   sp|18,*
                stpsp   bp|16
        2       eapbp   bp|32
                stpbp   bp|-14
        4       eabsp   bp|-32
                stpap   sp|26
        6       lda     lp|18,*
                sta     lp|20,*
        8       eapbp   lp|22,*
                lda     lp|24,*
       10       sta     lp|26,*
                lda     lp|28,*
       12       sta     lp|22,*
                ldb     sp|16,*
       14       lreg    sp|8
                rtcd    sp|20
                arg     30
```

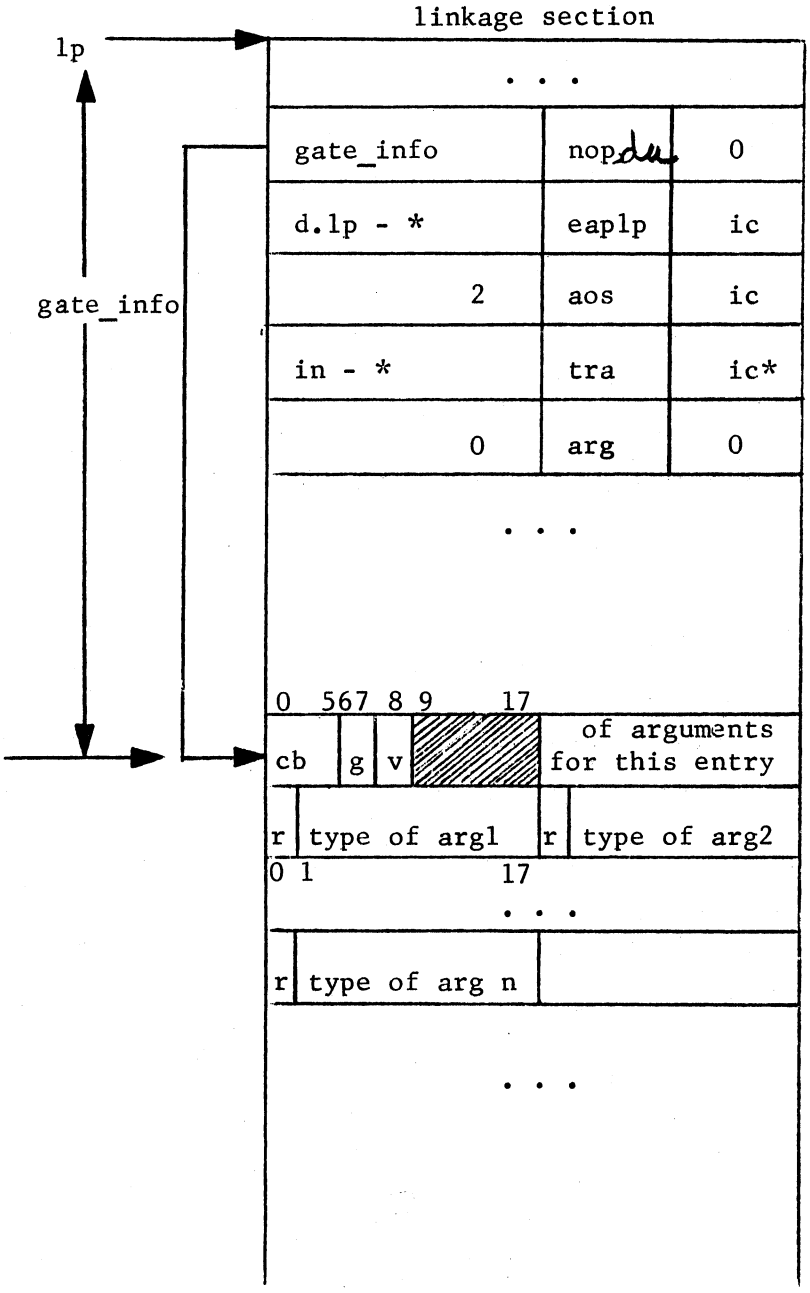
17	20-17		0		
	8		1		
	(3)8	c	1	c	
20	24-17		0		
	16		0		
	(4)8	m	a	s	
	s				
24	0		0		
25	26-17		0		<d>|0
26	1		0		
	0		0		
28	29-17		0		<dpe>|0
29	3		0		
	31-17		0		
31	(3)8	d	p	e	

32	33-17		0		<all>|[str]
33	4		0		
	35-17		36-17		
35	(3)8	a	1	1	
36	(3)8	s	t	r	
37	38-17		5		<rat>|[gas]+5,*
38	4		0		
	40-17		41-17		
40	(3)8	r	a	t	
41	(3)8	g	a	s	
42	43-17		40		<dta>|[x]+40
43	4		46-17		
	45-17		56-17		
45	(3)8	d	t	a	
46	12		14		
47	48-17		0		<rat>|0
48	3		0		
	40-17		0		
50	51-17		0		bp|[heat]
51	2		0		
	bp		53-17		
53	(4)8	h	e	a	
	t				
55	38-17		0		<rat>|[gas]
56	(1)8	x			

k + 0	d#	0	its	
	17		0	
2	-1	0	its	
	1	0	0	
4	-1	0	its	
	1	0	0	
6	8		30	
	d#		30	
8	-8	eaplp	ic	
	2	aos	ic	
10	12-10	tra	ic*	
			0	
12	-12		ft2	<d>|0
	25-17		0	
14	-14		ft2	<all>|[str]
	32-17		0	
16	-16		ft2	<dpe>|0
	28-17		0	
18	-18		ft2	<rat>|[gas]+5,*
	37-17		*	
20	-20		ft2	<dta>|[x]+40
	42-17		0	
22	-22		ft2	<rat>|0
	47-17		0	
24	-24		ft2	bp|[heat]
	50-17		0	
26	-26		ft2	<rat>|[gas]+5
	37-17		0	
28	-28		ft2	<rat>|[gas]
	55-17		0	

Contents of <d.link>, Links and Entries Immediately After Loading

Figure 6



gate or door entry

cb = highest ring number in call bracket

g = 1 if this entry is a gate

2 if this entry is a door

v = 0 if arguments are not to be validated

1 if gatekeeper is to validate arguments

r = 1 if this argument is a return value

0 if no value is allowed to be returned for this argument

Figure 7