TO: MSPM Distribution
FROM: R. H. Thomas
SUBJECT: BX.14.01, BX.14.02
DATE: August 26, 1968

The attached revisions of BX.14.01 and BX.14.02 reflect changes in both the coding scheme for relocation information (BD.2.01) and the implementation strategies for the binder and post_binder.

Published: 08/26/68
(Supersedes: BX.14.01, 12/28/66)

Identification

The Binder
bind
R. H. Thomas, D. L. Boyd

Purpose

The basic binder combines two segments into one segment. Binding is useful in reducing the number of system table entries necessary for a group of segments and thus the system overhead required to maintain the tables. In addition binding provides a means of dealing with the storage fragmentation problem. With binding the necessity of inter-segment links between bound segments is eliminated (see BX.14.02).

Introduction

A segment <input_name1> is actually a collection of three segments; <input_name1>, <input_name1.link>, and <input_name1.symbol> (BD.2.01). This collection will be referred to as a segment group. The binder combines the segment group <input_name1> with the segment group <input_name2> as follows: segment <input_name1> is combined with segment <input_name2>; segment <input_name1.link> and segment <input_name2.link> are combined; and segment <input_name1.symbol> and segment <input_name2.symbol> are combined. To the user, two segments that have been bound appear the same as one unbound segment. It is not possible to unbind or update a segment that has been bound.

Usage

The command to bind is given as follows:

```
bind input_name1 input_name2 output_name
```

Input_name2 is the name of a segment group which is to be bound to the end of the segment group named input_name1. The segment group which results from binding is called output_name.

It is possible to bind a newly bound segment group with still another segment group by giving the bind command again. The machine time used to bind two segments increases proportionally with the length of the segment input_name2. Therefore, it is advantageous to assign the shorter of the two segment groups to input_name2.

There is no console interaction with the bind command.

Implementation

Binding is done in two steps. The first step consists of performing the necessary initialization and the second step does the actual binding. All errors use the error handling mechanism described in BY.11.00.

Assume in all the following paragraphs that segment group y (input_name2) is being bound to segment group x (input_name1) and that the bound segments are placed in segment group z (output_name). With the initial version of the binder, for binding to occur, it is necessary that all six input segments mentioned in the introduction be available.

A) Initialization for Binding

The bind command attempts to initiate the six segments necessary for binding. If it fails an error comment is made using the error handling mechanism.

If the six segments can be initiated, pointers are created to each of them and to their binding information. The output segment group, output_name, output_name.link, output_name.symbol, is created and pointers to each output segment are saved. In addition the following parameters are calculated:

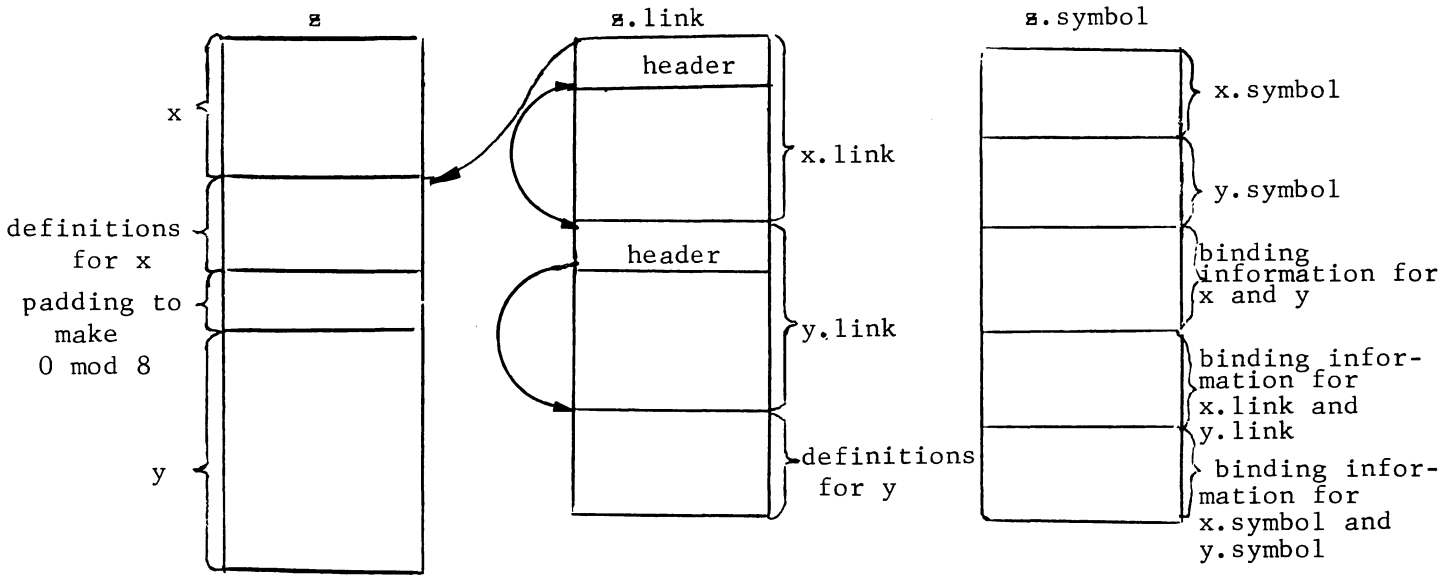
(1) len_text = length of segment x rounded up to 0 modulo 8.

(2) len_link = length of segment x.link rounded up to 0 modulo 8.

(3) len_symbol = length of segment x.symbol rounded up to 0 modulo 8.

B) The Binding Process

Given the case where the definition section for segments x and y are in segments x and y.link respectively, an example of how the bound segments appear follows:



The binder copies input_name1 and input_name1.link into output_name and output_name.link and appends words of zeroes as padding in order to make each component segment have length 0 modulo 8. Input_name1.symbol is copied into output_name.symbol up to the start of the binding information; it also is padded with zeroes to make it have length 0 modulo 8.

The binder then uses the binding information in input_name2.symbol to bind segment group input_name2 to segment group input_name1. Each segment is bound successively starting with input_name2, then input_name2.link and finally input_name2.symbol.

The binding information is arranged so that there is relocation information for every eighteen bits of each segment. The binding information is stored using the code described in MSPM BD.2.01. The chart below shows how this code is used to adjust each eighteen bit half-word. To read the chart assume K is the value of the half-word being relocated.

Code		Action
0	Absolute	no relocation
10000	Text	K = K + len_text
10001	Neg Text	K = K - len_text
10010	Link Pointer 18	K = K + len_link
10011	Neg Link Pointer 18	K = K - len_link
10100	Link Pointer 15	The left-most three bits are left unchanged and len_link is added to the remaining 15 bits of the word.
10101	Definition Pointer	no relocation
10110	Symbol	K = K + len_symbol
10111	Neg Symbol	K = K - len_symbol
11000	Link Block	no relocation
11001	Neg Link Block	no relocation
11010	Self relative	no relocation
11011	Unused	Should never occur
11100	Unused	Should never occur
11101	Unused	Should never occur
11110	Unused	Should never occur
11111	Escape	Should never occur

Binding is done for each segment in segment group output_name2 using the following loop. An eighteen bit half-word is read, the eighteen bits are adjusted according to the binding information for that half-word, and the new eighteen bit half-word is stored in the appropriate output segment.

Structure of the Bound Segment Group

The binder sets the binding indicator on in each symbol table header and threads the symbol table headers together. The symbol table header entries that contain the length of the component text and linkage sections are updated to account for the padding necessary to make them of length 0 modulo 8.

A segment group with a linkage section always contains "external symbol definitions" and "link definitions" threaded together (MSPM BD.7.01). When binding is completed, the definition sections of the two segment_groups are not threaded together; however, the linkage sections are threaded together as shown in figure 1.

The binder searches the definitions sections for any duplicate entry points or segment definitions. If any duplicates are found, it is considered an error.

Each definition section corresponding to a component segment will have a definition for the symbols rel_text, rel_link and rel_symbol. The binder updates these definitions so that each occurrence of rel_text points to the binding information for the bound text segment, each occurrence of rel_link to that for the bound linkage section, and each occurrence of rel_symbol to that for the bound symbol segment. In addition each definition section will contain a definition for the symbol: symbol_table (see BD.1.00). After binding each occurrence of symbol_table will point to the symbol table header corresponding to the first component segment. The duplicate definitions of these four symbols will be removed when the segment is post_bound. (See BX.14.02.)