MCS 80/85
RELOCATABLE OBJECT MODULE
FORMATS

An Intel Technical Specification

Order Number: 121747-001

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

PREFACE

This manual defines the internal format of 8080/8085 relocatable object files produced by Intel's resident or cross product language translators and read by other Intel software products. The information in this manual is normally not needed in order to use Intel software, but is provided for the person who needs to write programs to process these object files or to create files in the same format.

The character set of the American Standard Code for Information Interchange (ASCII) is defined in the following document:

American National Standard Institute, Code for Information Interchange, X3.4-1968.

CONTENTS

INTRODUCTION

Execution of computer programs requires that programs be loaded into memory. Therefore, a method of representing a desired memory state (memory content, memory image) is required.

This document defines a set of formats that permit specification of relocatable memory images that may be linked one to another. We assume that "relocation" and "linkage" are primitive concepts that need not be defined.

The reader may wish to refer to the ISIS-II USER'S GUIDE for additional information on relocation and linkage.

SEGMENTATION OF PROGRAMS

An object module contains one or more components called Segments. Except for the Absolute Segment, a Segment defines a contiguous area of memory. Semantics of Segments are pre-defined, and vary from Segment to Segment (see also the Appendix on Segment Combination and Segment Locating for additional information on the properties of the different segments). The Segments are identified by numbers:

0. ABSOLUTE SEGMENT.

The Absolute Segment is not actually a segment. However, in the object module format, several varieties of information must be preceded by a "Segment ID" that identifies the Segment to which the information belongs. The number zero is used to precede similar information that belongs to no Segment, but is absolute. References to the Absolute Segment should be understood to refer to the collection of all such information within a Module, without implication of contiguity or relocatability of that information.

1. CODE SEGMENT.

The translator puts information into the Code Segment which is destined for ROM. Such information may be the executable code produced by the PL/M compiler, or any information in Assembly Language following a CSEG directive.

2. DATA SEGMENT.

The semantics of this Segment are the same as those of the Code Segment, except that the information is destined for RAM.

3. STACK SEGMENT.

This Segment is, at run time, a contiguous area of RAM that is used as a run-time stack. Normally translators will not define symbols or structures in this area, but will assume the existence of a distinguished Name (e.g., STACK in 8080/8085 ASSEMBLY LANGUAGE) for the highest memory byte of this Segment.

4. MEMORY SEGMENT.

This Segment is, at run time, a contiguous area of RAM located in high address space (just below the address returned by the MDS MONITOR function MEMCK). Normally, translators will not define symbols or structures in this area, but will assume the existence of a distinguished Name (e.g., MEMORY in PL/M) for the lowest memory byte of this Segment.

5. RESERVED SEGMENT

The Segment identified by the number five is reserved for use by future Intel software.

254, 253, 252, ... 6. NAMED COMMON SEGMENTS

Each module may define a mapping between these segment numbers and FORTRAN Common Names. At link time, LINK will produce an output mapping that will maintain a unique number for each Common Name (thus there is a limitation that no more than 249 distinct Common areas may exist), and will resolve references to each Common area in the same fashion that references to the unnamed Common are resolved.

255. UNNAMED COMMON SEGMENT.

This Segment corresponds to FORTRAN's "blank" Common area. Several modules may define the size of this Segment; LINK will resolve these definitions in a single contiguous memory area.

RECORD FORMAT DIAGRAMS

The record format diagrams use the following conventions:

(A) (B) (C)

(A) represents a single-byte field; (B) represents a two-byte field that specifies a 16-bit value (8080 standard, i.e., low-order, or first byte contains the low-order half of the number); (C) represents a field of a variable number of bytes.

Some records contain a field or series of fields that may be repeated. Such portions are indicated by the "REPEATED" or "RPT" brackets in the diagrams.

Any field that indicates a "Name" has the following internal structure: the first byte contains a number between 1 and 255, inclusive, which indicates the number of remaining bytes in the field. These remaining bytes are interpreted as a byte string. Most translators will choose to constrain the values of these remaining bytes to ASCII codes of printable characters, but nothing in this specification requires such constraint.

A field marked "OFFSET" specifies a value in conjunction with another piece of information, a Segment ID. Whenever a SEG ID field immediately precedes an OFFSET field, it determines the SEGMENT for which the OFFSET field gives an offset. If a SEG ID field does not immediately precede the OFFSET field, (which occurs only in Fixup Records), then that offset is with respect to the SEG ID field in the previous Content Record.

Any field that indicates an "OFFSET" represents a value that is determined by adding the value in the offset field to the value of the base of the designated Segment.

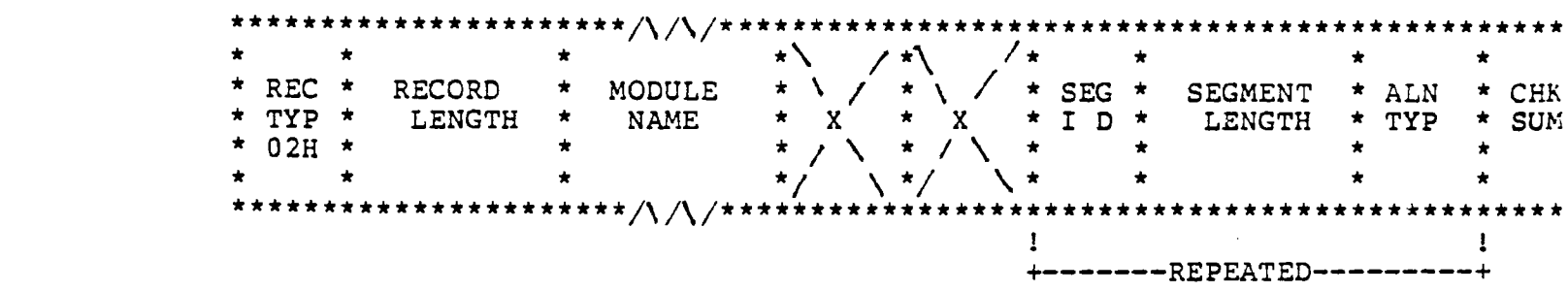
Any field with an "X" through it contains unspecified information and is to be ignored (but the field must be present where indicated). These fields are reserved for use by future Intel software. They must contain zeros, any value other than zero may not be compatible with current or future Intel products.

COMMON RECORD FIELDS

All record formats share this common organization: (1) The first byte is a field that contains a number that identifies which kind of record the record is. This is the RECORD TYPE (REC TYP) field. (2) The next two bytes are a field that contains a number called the RECORD LENGTH. It is the total number of bytes in the record, exclusive of the three bytes that comprise the RECORD TYPE and RECORD LENGTH fields. (3) The last byte in every record is the CHECK SUM (CHK SUM) field; it contains the two's complement of the sum modulo 256 of all other bytes in the record.

The maximum value contained in any RECORD LENGTH field is 1025 (decimal), with the exception of (1) Library Records, and (2) Content Records that specify the Absolute Segment in their SEG ID field and have no Fixup Records immediately following.

MODULE HEADER RECORD



This Record is the first record in an object module. It identifies the Module and defines the segments that are used by the module.

MODULE NAME

Every Module has a Name. The Module name will be listed in various summary listings produced by the LIB, LINK and LOCATE programs, and is a "handle" by which the user may designate certain Modules to programs such as LIB and LINK.

A valid module name contains between 1 and 31 characters, inclusive, each of which must be a capital letter (A, B, ..., Z), a digit (0, 1, 2, ..., 9), a question mark (?) or a commercial at sign (@). Furthermore, the first character of a module name may not be a digit.

"X" FIELDS

These fields are reserved for use by future Intel software, they must contain zeros.

SEG ID, SEGMENT LENGTH and ALIGNMENT TYPE (ALN TYP)

Every Module Header Record contains zero or more of these field trios. Each trio identifies a Segment by giving its identifying number, the length (size) in bytes, and the alignment type (also called the relocation type) of the Segment in the Module. No Segment may appear in more than one such trio; no Segment may appear in a Module unless its length (or an upper bound on its length) has been given in the Module Header Record. (Except the Memory Segment, which, if present, provides a lower bound (possibly zero) on its required length.)

The Absolute Segment is the exception to these rules. Since absolute information within a Module may be discontiguous, and since it cannot be "relocated", the "length" and "alignment type" of such information have no meaning. Thus, the Absolute Segment may not be specified in a SEG ID field of a Module Header Record.

The ALN TYP field specifies the type of alignment that must be applied to a Segment when it is relocated. This field must contain one of the following: 1, 2 or 3.

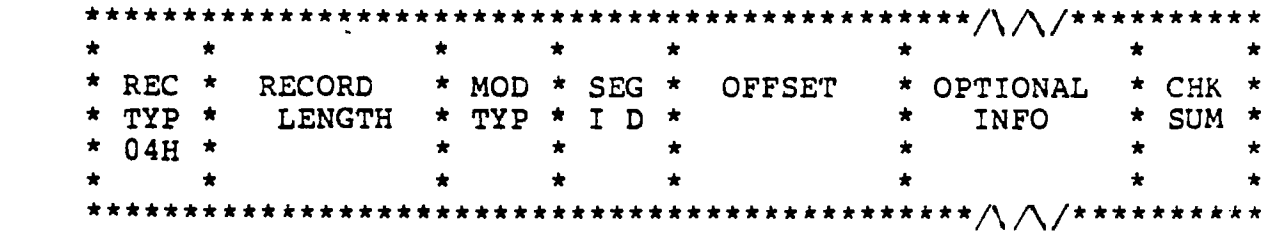
1 indicates that the Segment must fit within a single Page when relocated. A Page of memory is 256 bytes beginning on a Page boundary (see below). This is also called In-page relocatable.

2 indicates that the Segment must begin on a Page boundary when relocated. Page boundaries occur in multiples of 256 bytes beginning with zero (0, 256, 512, etc.) This is also called Page relocatable.

3 indicates that the Segment may begin on any byte boundary when relocated. This is also called Byte relocatable.

Additional information on these alignment types is found in the 8080/8085 ASSEMBLY LANGUAGE PROGRAMMING MANUAL and the ISIS-II USER'S GUIDE.

MODULE END RECORD



This Record is the last record in an object module. It identifies main programs and gives their starting address.

MODULE TYPE (MOD TYP)

This byte has value 0 or 1. 1 designates the Module as a main program; 0 designates the Module as not a main program.

SEG ID and OFFSET

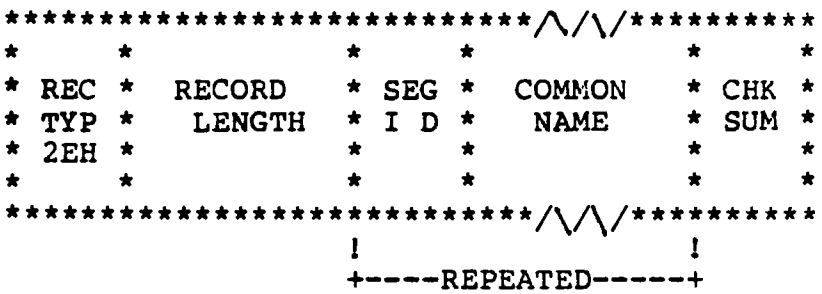
If the Module is a main program, then these two fields specify the Module's start address. Otherwise, these two fields have no significance (but must be present and should contain zeros).

OPTIONAL information

This field is not used by the LIB, LINK or LOCATE programs at this time. This field may be omitted from the Record, at the discretion of the translator.

This field is reserved for use by future Intel software. Any usage of this field by other than Intel software may not be compatible with current or future Intel products.

NAMED COMMON DEFINITIONS RECORD



Zero or more of these Records may follow the Module Header Record. This Record provides the vehicle by which a translator may declare (FORTRAN) Named Common Blocks.

A Named Common Block is a contiguous area of memory that can be referenced by one or more modules. This Record declares one or more Common Names. Each Name is defined by a pair of fields.

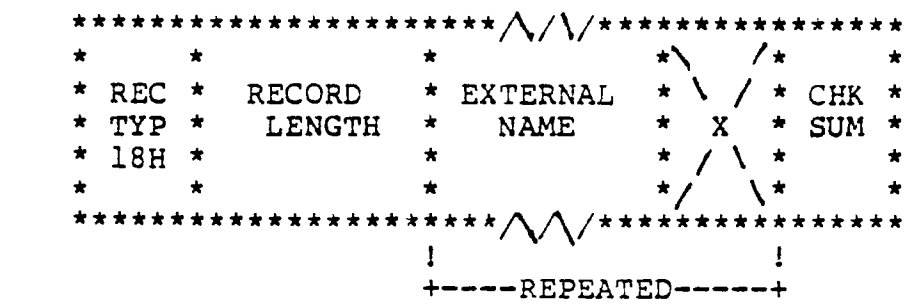
SEG ID

This field gives the number used to identify the Named Common Block in this module. For each distinct Name used, the translator must assign a unique SEG ID number. This number must be less than 255, and greater than the largest number assigned for non-Common segments (currently 5). Preferred practice is to start with 254 and work downwards as required.

COMMON NAME

This field gives the Name of the Named Common Block. The same Common Name may not appear more than once within the same Module.

EXTERNAL NAMES RECORD



This Record provides a (possibly partial) list of External Names.

An External Name is a symbolic reference to a memory location, where the location is defined in some other Module. This Record declares one or more External Names. Each Name is defined by a pair of fields:

EXTERNAL NAME

This field gives a Name, to which the Module (probably) makes one or more references (see External References Record). The same External Name may not appear more than once within the Module.

Inclusion of a Name in an External Names Record is an implicit request that the Module be linked to a module containing the same Name declared as a Public Name. This request occurs whether or not the Name is actually referenced within the Module, by an External References Record (see below).

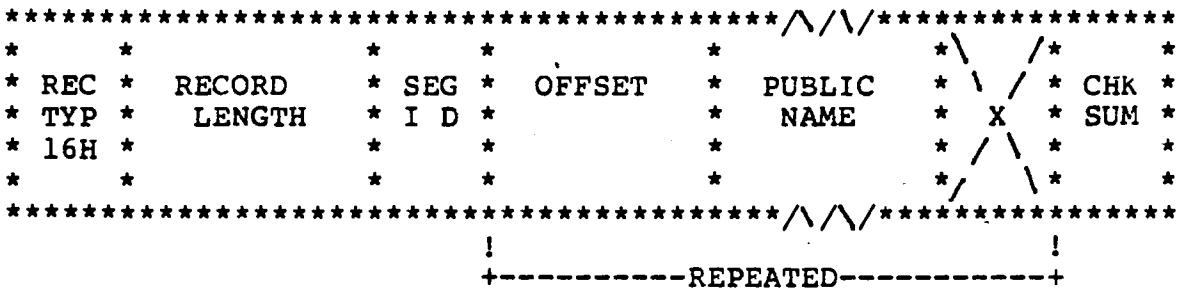
"X" FIELD

This field is reserved for use by future Intel software, it must contain a zero.

External Names Dictionary:

The ordering of External Names Records within a Module, together with the ordering of External Names within each External Names Record, induces an ordering on the set of all External Names requested by the Module. Thus, the External Names are considered to be numbered: 0, 1, 2, 3, ... These numbers are used in External Reference Records to indicate External Names.

PUBLIC DECLARATIONS RECORD



This Record declares that certain Names in this Module are "PUBLIC", i.e., other modules are allowed to make references to the (final) memory location assigned to these Names by making symbolic references to the Names. (These references to a Public Name are made by use of the External Names Records and the External References Records.)

This Record declares one or more Public Names. Each Name is defined by a trio of fields.

SEG ID

This field identifies a Segment by giving its identifying number. All Public Names declared in this record belong to this Segment.

OFFSET

This field defines the location of the Public Name, with respect to the base of the Segment identified by the SEG ID field.

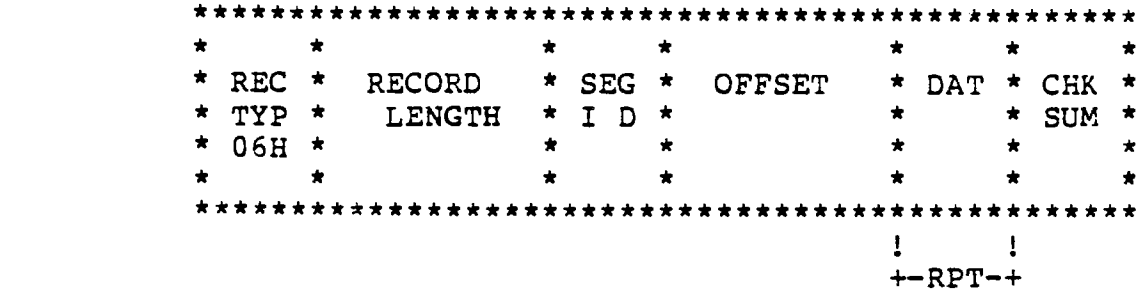
PUBLIC NAME

This field gives the Name by which the location may be referred to from another module. The same Public Name may not appear more than once within the same Module.

"X" FIELD

This field is reserved for use by future Intel software, it must contain a zero.

CONTENT RECORD



This Record provides one or more bytes of contiguous data, from which a portion of a memory image may be constructed. The data belongs to the specified Segment, and is subject to modification by "Fixup Records" which follow, if any. (Fixup Records are described below.)

SEG ID

This field identifies a Segment by giving its identifying number. The contiguous data defined by this Record belongs to this Segment.

Specification of the Stack Segment in a Content Record is forbidden; results of programs processing such a record are undefined.

OFFSET

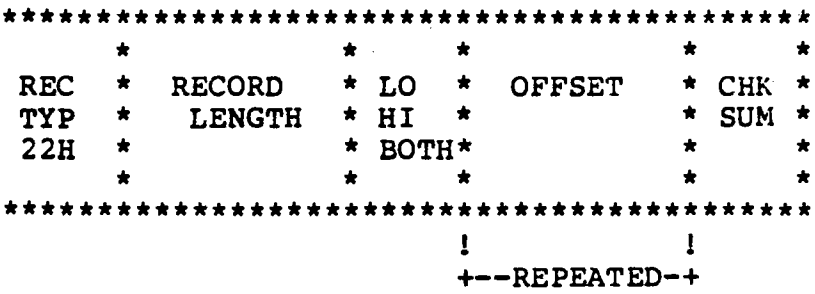
This field specifies the location of the first data byte, with respect to the Segment specified by the SEG ID field.

DAT

Following the OFFSET are one or more data bytes. If n is the number of data bytes, then OFFSET+n must be less than the LENGTH specified for the "owning" Segment in the Module Header Record. (This requirement does not hold for the Absolute Segment.)

Thus, this Record provides n consecutive bytes of a memory image, from BASE+OFFSET through BASE+OFFSET+n-1, inclusive, where "BASE" is the location of the base of the current Segment. It is illegal for BASE+OFFSET+n-1 < BASE+OFFSET (wrap around 64K), results of programs processing such a record are undefined.

RELOCATION RECORD



Zero or more of these Records may be used to indicate which bytes in the preceding Content Record contain references to locations relative to the base of the "owning" Segment, and which must therefore be modified dependent upon the Segment's final location in memory. This Record identifies one or more of these references.

LO HI BOTH

This field indicates what kind of references in the preceding Content Record are selected by this Record. This byte must have value 1, 2 or 3, corresponding to LO, HI or BOTH:

1 indicates that the references are to the low-order byte of a memory address.

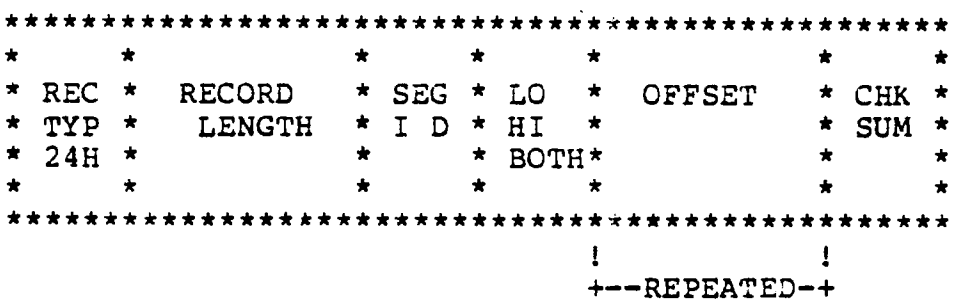
2 indicates that the references are to the high-order byte of a memory address.

3 indicates that the references are to both bytes of a memory address. In this case, the OFFSET (see below) refers to the low-order byte of the address, and the high-order byte follows the low-order byte. Note that both bytes being "fixed up" must be present in the preceding Content Record.

OFFSET

This field denotes the location of the reference to be "fixed up", by giving its address as an offset from the base of the "owning" Segment. This offset must specify a data byte within the previous Content Record.

INTER-SEGMENT REFERENCES RECORD



Zero or more of these Records may be used to indicate which bytes in the preceding Content Record contain references to locations in some other Segment. This Record identifies one or more of these references.

SEG ID

This field identifies the segment, which the preceding Content Record makes references to, by giving its identifying number. This field may not identify the Absolute Segment.

LO HI BOTH

This field indicates what kind of references in the preceding Content Record are selected by this Record. This byte must have value 1, 2 or 3, corresponding to LO, HI or BOTH:

1 indicates that the references are to the low-order byte of a memory address.

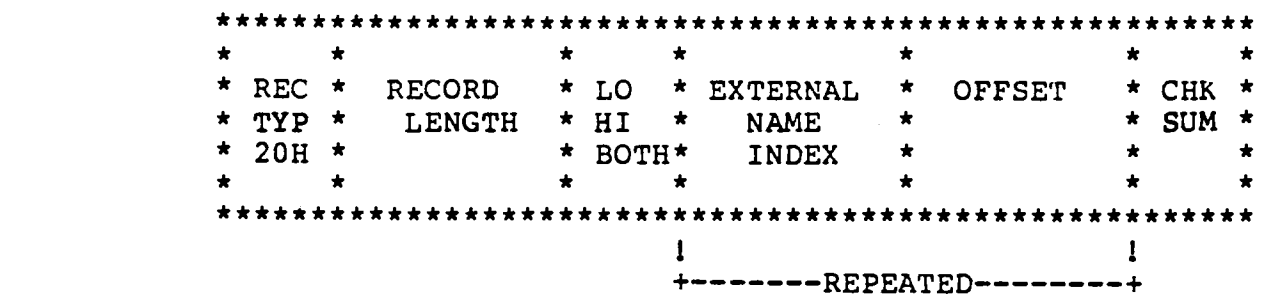
2 indicates that the references are to the high-order byte of a memory address.

3 indicates that the references are to both bytes of a memory address. In this case, the OFFSET (see below) refers to the low-order byte of the address, and the high-order byte follows the low-order byte. Note that both bytes being "fixed up" must be present in the preceding Content Record.

OFFSET

This field denotes the location of the reference to be "fixed up", by giving its address as an offset from the base of the "owning" Segment. This offset must specify a data byte within the previous Content Record.

EXTERNAL REFERENCES RECORD



Zero or more of these Records may be used to indicate which bytes in the preceding Content Record contain references to locations in other Modules. These locations are identified by sequences of characters called "Names"; these Names must be previously defined in an External Names Record. This Record identifies one or more of these references.

LO HI BOTH

This field indicates what kind of references in the preceding Content Record are selected by this Record. This byte must have value 1, 2 or 3, corresponding to LO, HI or BOTH:

1 indicates that the references are to the low-order byte of a memory address.

2 indicates that the references are to the high-order byte of a memory address.

3 indicates that the references are to both bytes of a memory address. In this case, the OFFSET (see below) refers to the low-order byte of the address, and the high-order byte follows the low-order byte. Note that both bytes being "fixed up" must be present in the preceding Content Record.

Each reference is defined by a pair of fields:

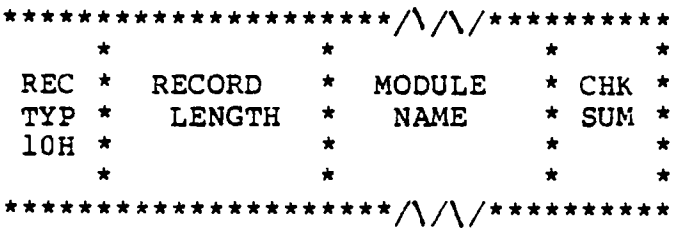
EXTERNAL NAME INDEX

This number identifies a Name by giving its index in the External Names Dictionary, which is implicitly defined by the External Names Records within the Module (see External Names Record, above). It is required that the definition of an External Name (in some External Names Record) precede the occurrence of an EXTERNAL NAME INDEX field that refers to it.

OFFSET

This field denotes the location of the reference to be "fixed up", by giving its address as an offset from the base of the "owning" Segment. This offset must specify a data byte within the previous Content Record.

MODULE ANCESTOR RECORD



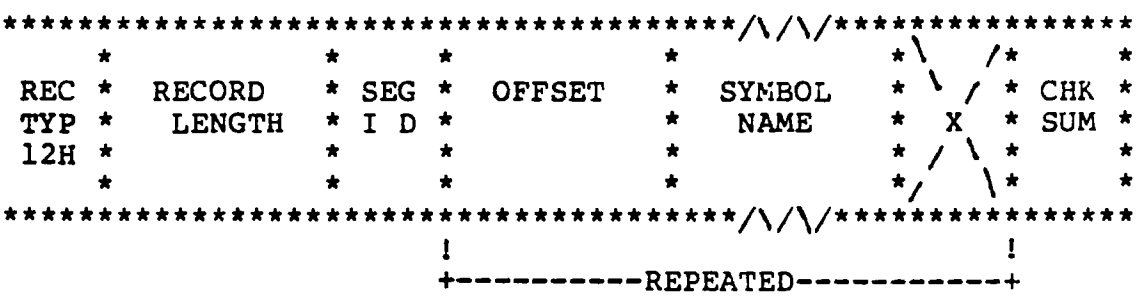
This Record, if present, provides the identity of the ultimate progenitor Module for all line numbers and/or local symbols encountered in the following Line Numbers Records and/or Local Symbols Records.

The Module Header Record, in addition to its normal functions, also serves as a Module Ancestor Record. This defines the method whereby a line number and/or local symbol may be associated with the name of its original containing Module.

MODULE NAME

This field gives the Name of the Module in which the following Line Numbers Records and/or Local Symbols Records have been defined originally.

LOCAL SYMBOLS RECORD



This Record provides the vehicle by which a translator may pass to a debugger program information about symbols (i.e., Names) that were used in the source language input to the translator that produced the Module. It is intended that this information may be useful to ICE or other debugger programs.

This information is processed, but not used, by LIB, LINK or LOCATE.

The symbols in the Record are deemed to have been originally defined in a source Module of Name given by the most recently preceding Module Ancestor Record (or, if none, by the Module named by the Module Header Record).

This Record declares one or more symbols. Each symbol is defined by a trio of fields.

SEG ID

This field identifies a Segment by giving its identifying number. All Symbols defined in this Record belong to this Segment.

OFFSET

This field defines the location of the symbol within the Segment identified in the SEG ID field.

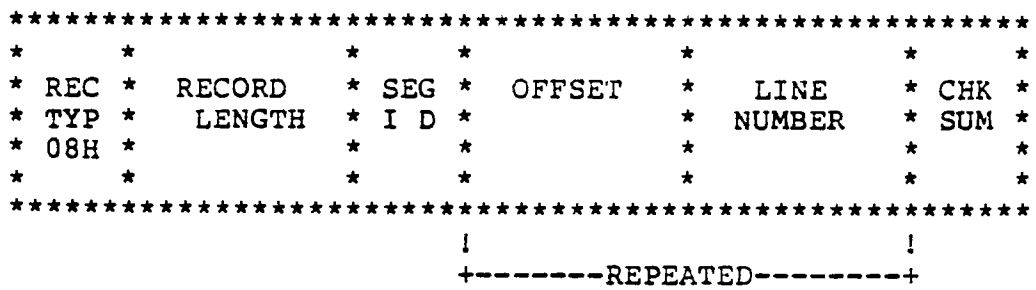
SYMBOL NAME

This is the symbol (Name, Identifier, label, etc.) used in the source language.

"X" FIELD

This field is reserved for use by future Intel software, it must contain a zero.

LINE NUMBERS RECORD



This Record provides the vehicle by which a translator may pass to a debugger program the correspondence between a line number in source code and the corresponding object code. It is intended that this information may be useful to ICE or other debugger programs. Normally only the source lines for which executable object code was generated need to be identified.

This information is processed, but not used, by LIB, LINK or LOCATE.

The lines identified in the Record are deemed to have been originally defined in a source Module of Name given by the most recently preceding Module Ancestor Record (or, if none, by the Module named by the Module Header Record).

This Record declares one or more line numbers. Each line number is defined by a trio of fields.

SEG ID

This field identifies a Segment by giving its identifying number. All line numbers defined in this Record belong to this Segment.

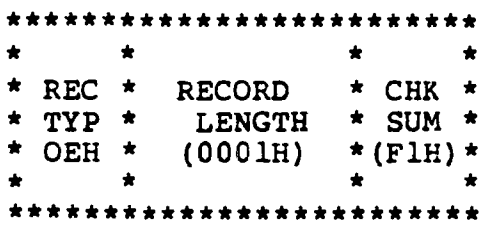
OFFSET

This offset defines the location within the Segment where the object code for the corresponding source line begins.

LINE NUMBER

The 2 bytes provide, in binary, a line number between 0 and 65535, inclusive.

END OF FILE RECORD



This Record indicates the end of file on a physical medium. To aid media transparency, it is required on all files.

LIBRARY HEADER RECORD

REC RYP 2CH	RECORD LENGTH (0007H)	MODULE COUNT	BLOCK NUMBER	BYTE NUMBER	CHK SUM

This Record is the first record in a library file. It immediately precedes the modules (if any) in the library. Following the modules (if any) are 3 more records (described on the following pages) in order: Library Module Names Record, Library Module Locations Record, and Library Dictionary Record.

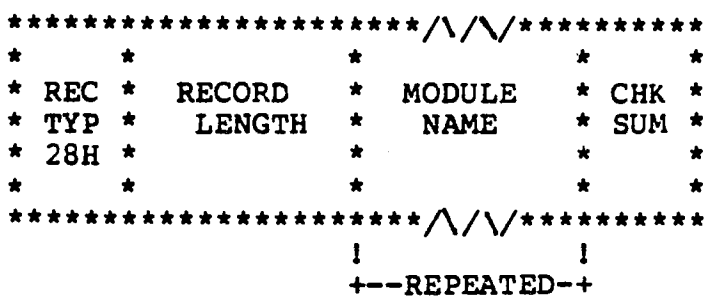
MODULE COUNT

This field indicates how many modules are in the library. It may have any value from 0 to 65535, inclusive.

BLOCK NUMBER, BYTE NUMBER

These fields indicate the relative location of the first byte of the Library Module Names Record in the library file.

LIBRARY MODULE NAMES RECORD

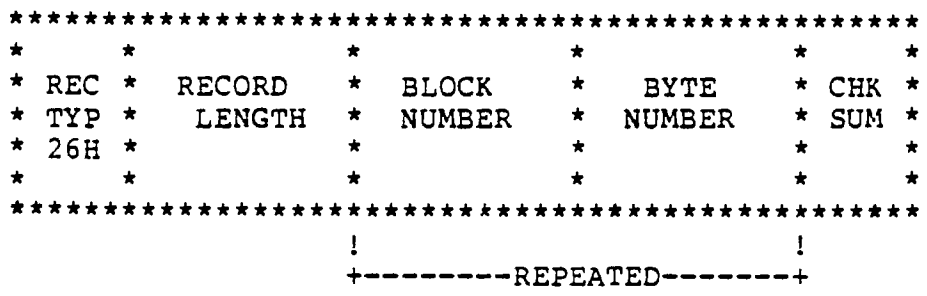


This Record gives the names of all the modules in the library. The order of the names correspond to the order of the modules within the library. Only one Library Module Names Record may appear in the library.

MODULE NAME

The i'th MODULE NAME field in the Record contains the module name of the i'th module in the library.

LIBRARY MODULE LOCATIONS RECORD



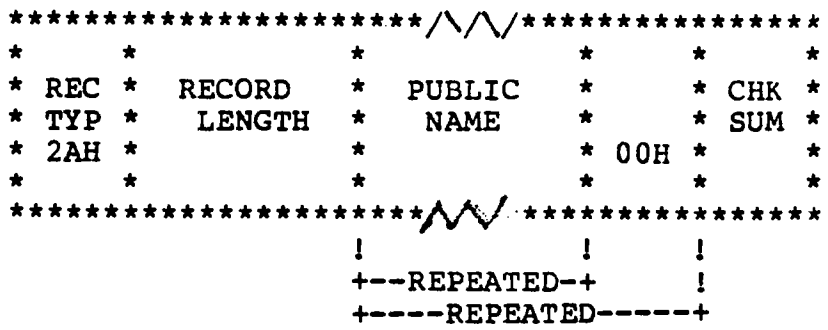
This Record provides the relative location, within the library file, of the first byte of (the Module Header Record of) each module. Only one Library Module Locations Record may appear in the library.

The order of the block-number/byte-number pairs corresponds to the order of the modules within the library.

BLOCK NUMBER, BYTE NUMBER

The i'th pair of fields provides the relative location within the library file of the first byte of the first record of the i'th module within the library.

LIBRARY DICTIONARY RECORD



This Record gives all the names of public symbols within the modules in the library. Only one Library Dictionary Record may appear in the library.

Since a name must have a non-zero length, the '00' bytes in the format are distinguishable from the PUBLIC NAME fields. Thus, the '00' bytes separate the public names into groups; all names in the i'th group are defined in the i'th module of the library.

PUBLIC NAME

This is the name of a Public symbol in the module. No Public symbol may appear more than once in this Record.

PROPER ORDER OF RECORDS

An Object Module or a Library is defined by a sequence of Records. The following syntax shows what orderings of Records are valid, and the following semantics give important information that is conveyed by the sequence, rather than mere content, of records. Definition of a valid string of Records is given by the following syntax: (Note: <ITEM>+ means the <ITEM> can occur one or more times, <ITEM>* means the <ITEM> can occur zero or more times.)

```
<VALID INPUT> ::= <MODULE> + <EOF RECORD>
                ! <LIB HEADER RECORD> <MODULE>* <LIB TAIL> <EOF RECORD>

<MODULE> ::= <MOD HDR> <COMPONENT>* <MODULE END RECORD>

<MOD HDR> ::= <MODULE HEADER RECORD> <NAMED COMMON DEFINITIONS RECORD

<COMPONENT> ::= <EXTERNAL NAMES RECORD>
              ! <PUBLIC DECLARATIONS RECORD>
              ! <CONTENT DEFINITION>
              ! <DEBUG RECORDS>

<CONTENT DEFINITION> ::= <CONTENT RECORD> <FIXUP RECORDS>*

<FIXUP RECORDS> ::= <RELOCATION RECORD>
                  ! <INTER-SEGMENT REFERENCES RECORD>
                  ! <EXTERNAL REFERENCES RECORD>

<DEBUG RECORDS> ::= <MODULE ANCESTOR RECORD>
                  ! <LOCAL SYMBOLS RECORD>
                  ! <LINE NUMBERS RECORD>

<LIB TAIL> ::= <LIB MODULE NAMES RECORD> <LIB MODULE LOCATIONS RECORD
             <LIB DICTIONARY RECORD>
```

The following semantics obtain:

1. All Named Common Definitions Records must follow immediately after the Module Header Record.

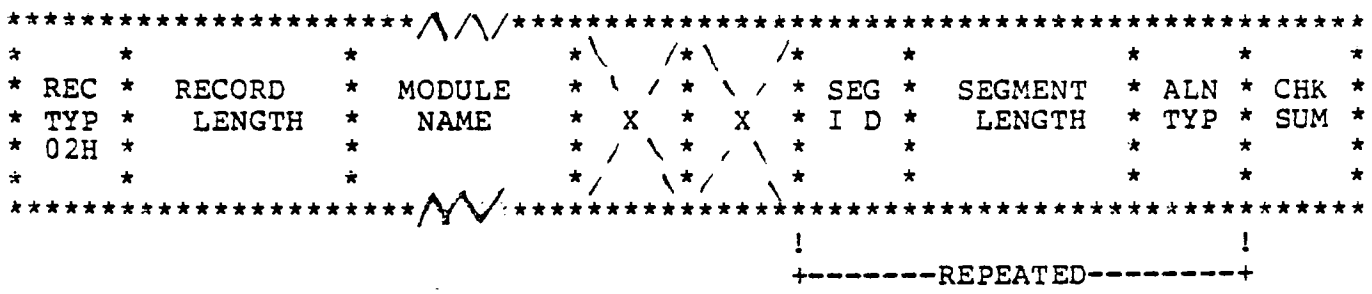
2. A Fixup Record always refers to the previous Content Record.

3. A Line Numbers Record, or a Local Symbols Record, is deemed to have originated in a Module named by the immediately preceding Module Ancestor Record, if any, otherwise in the Module itself (i.e., the Module named by the Module Header Record).

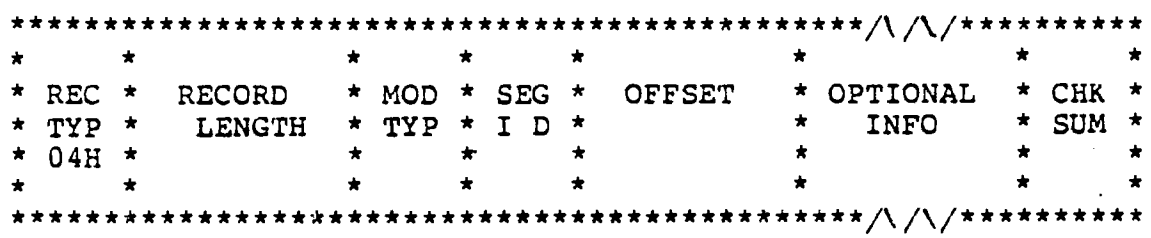
4. All External Names Records must precede all Fixup Records that refer to them.

APPENDIX: RECORD FORMATS for HANDY REFERENCE

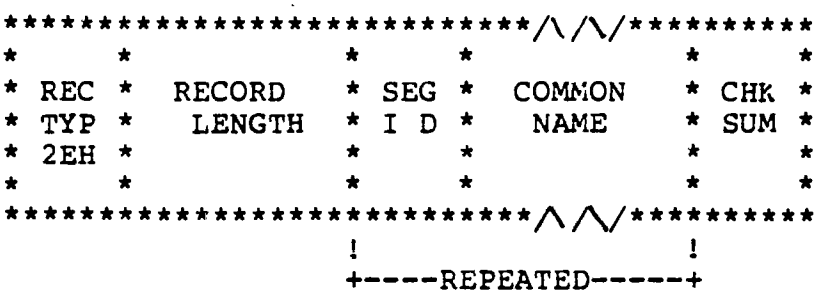
MODULE HEADER RECORD



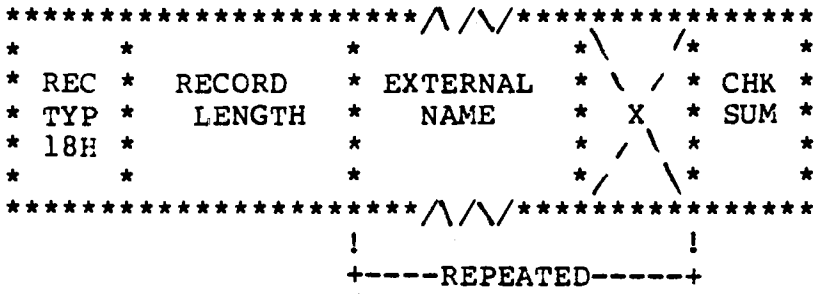
MODULE END RECORD



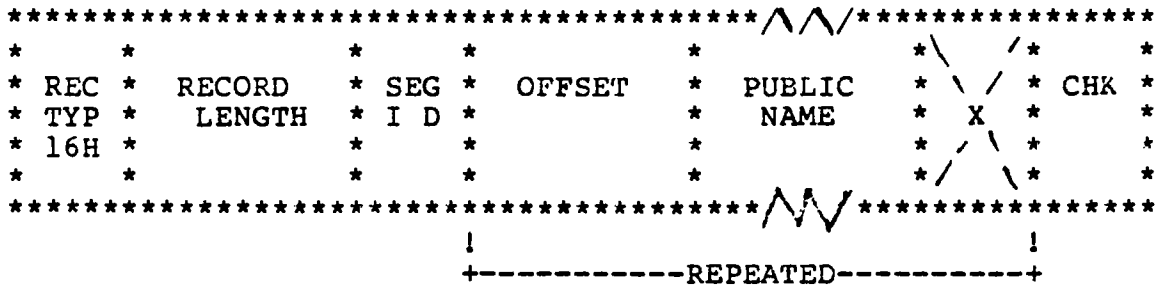
NAMED COMMON DEFINITIONS RECORD



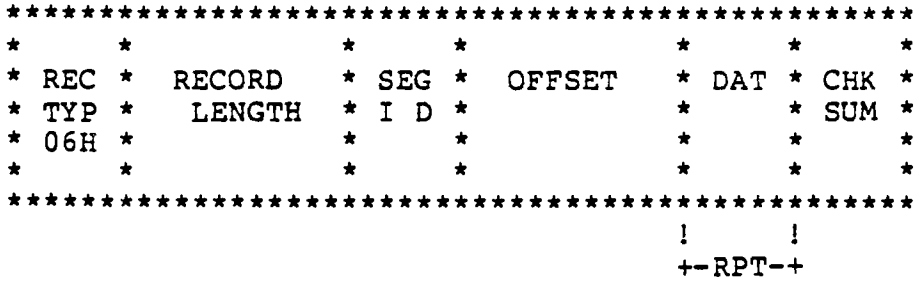
EXTERNAL NAMES RECORD



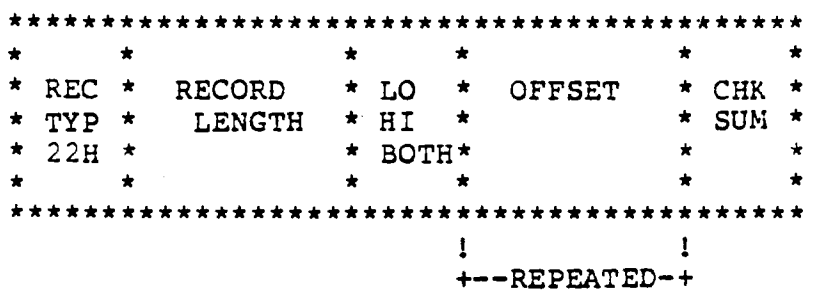
PUBLIC DECLARATIONS RECORD



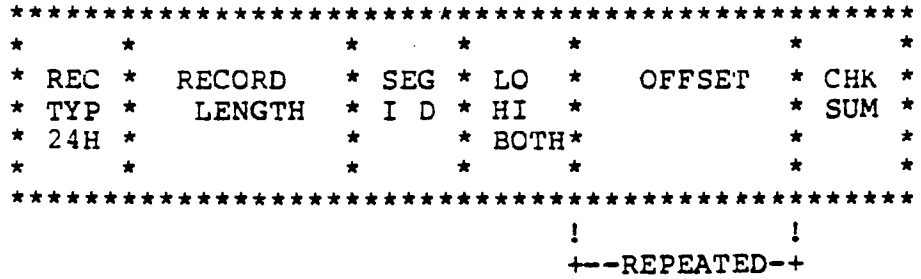
CONTENT RECORD



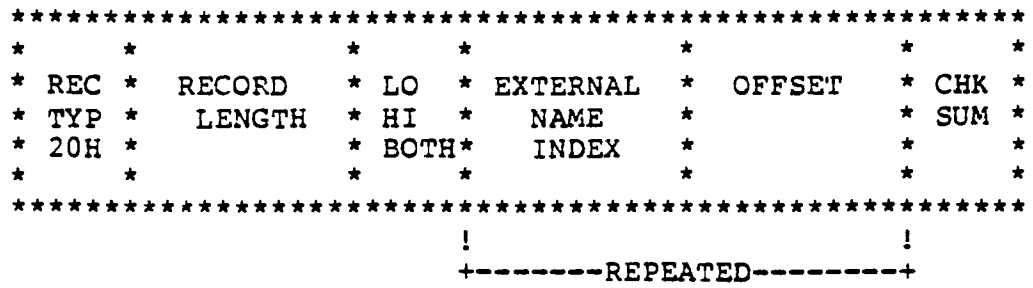
RELOCATION RECORD



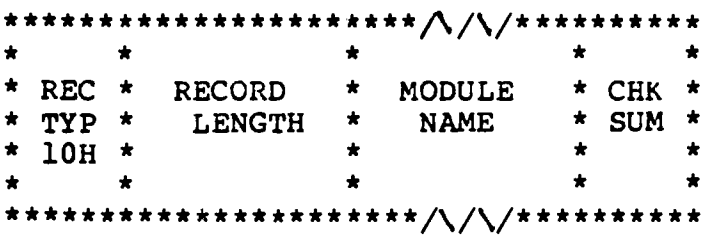
INTER-SEGMENT REFERENCES RECORD



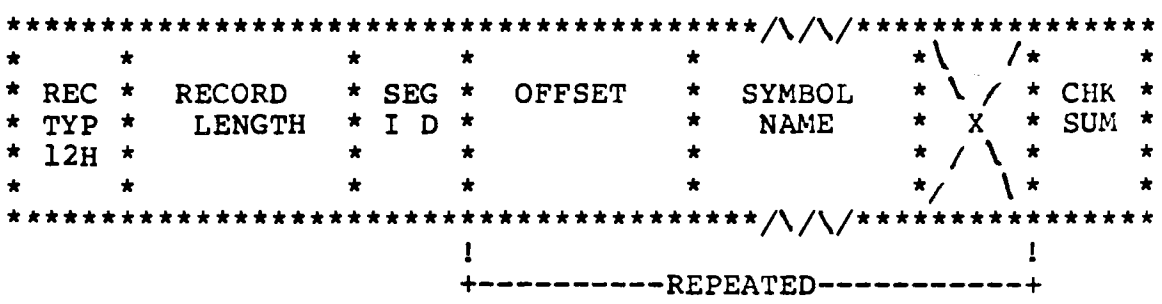
EXTERNAL REFERENCES RECORD



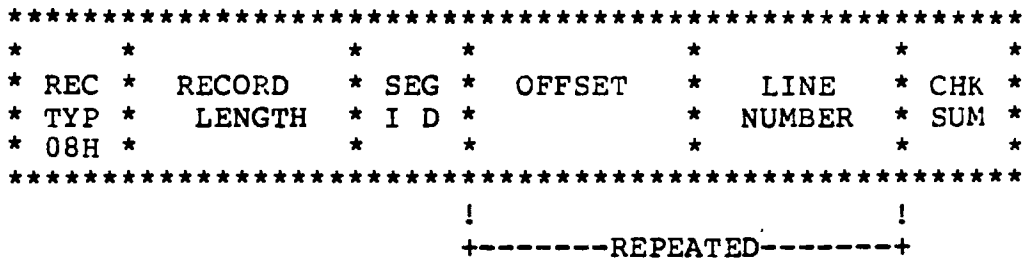
MODULE ANCESTOR RECORD



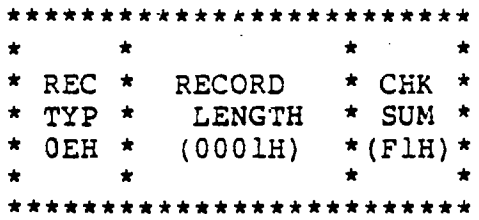
LOCAL SYMBOLS RECORD



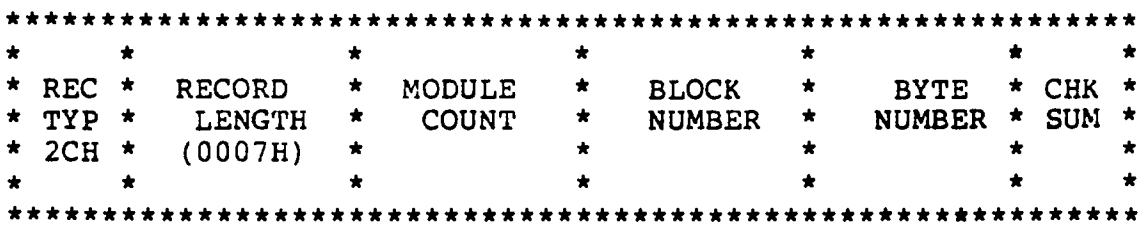
LINE NUMBERS RECORD



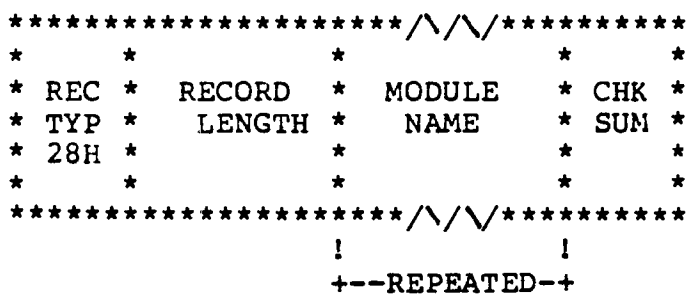
END OF FILE RECORD



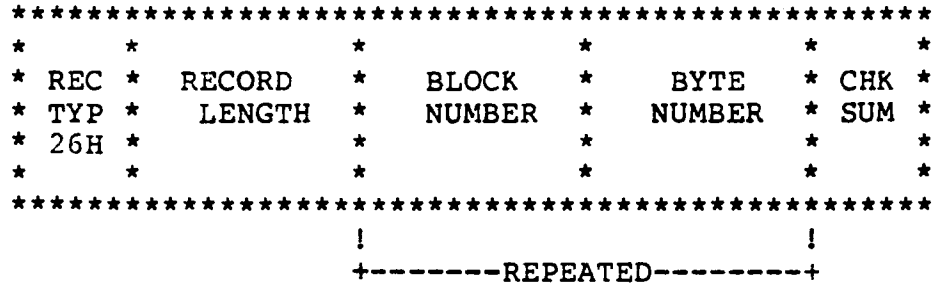
LIBRARY HEADER RECORD



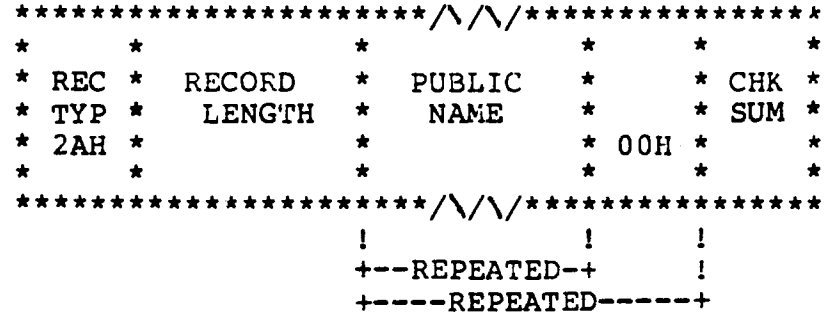
LIBRARY MODULE NAMES RECORD



LIBRARY MODULE LOCATIONS RECORD



LIBRARY DICTIONARY RECORD



APPENDIX: SEGMENT COMBINATION

When two or more modules are linked together by LINK, "like" segments are "combined" into a single segment for the output module; e.g., all the Code Segments are "combined" into a single Code Segment. The nature of "combining" depends on the segment, sometimes two segments being "combined" are concatenated, and sometimes only the lengths of the two segments are used to define a new "combined" segment.

A module may contain different segments (see section entitled Segmentation of Programs above). Each segment may have one of three alignment types: In-page relocatable (IP), Page relocatable (PR), and Byte relocatable (BR). Each segment's alignment type is specified in its Module Header Record (described above). These alignment types specify the way segments are placed in memory for execution.

The 8080 memory is often considered as being partitioned into "pages", where a page is 256 contiguous bytes (and the high order byte of the memory address of each byte is the same). This concept gains utility from the fact that two 8080 registers, e.g. H and L, are used to specify a memory address. If a value is put into H, then 256 different (and contiguous) bytes may be specified by the 256 possible different values placed in the L register. These bytes are said to form a "page". A "page boundary" is the first byte of a "page" (i.e., the low order byte of the memory address is zero).

A segment with alignment type IP must be of length 256 bytes or less, and it must be located within a single page, anywhere in memory; i.e., it must not cross a page boundary. Such segments will typically be referenced by first setting H, and thereafter modifying only L.

A segment with alignment type PR must be located such that the first byte of the segment is at a page boundary, anywhere in memory. Such segments are typically referenced by other segments in a method whereby the content of the HL register pair is made to point into the segment; and thereafter H and L are independently manipulated to reference other sections of the segment. Observe that it is frequently possible to change L, without changing H, because H selects the page, and L selects the byte within the page. Less frequently, H will be independently changed.

A segment with alignment type BR may be located anywhere in memory.

When two segments are "combined", the alignment types of each segment must be preserved in the resultant combined segment; furthermore the resultant alignment type must guarantee that any future linkage operations on the segment will not violate the original alignment types of its constituent sub-segments.

COMBINING TWO ABSOLUTE SEGMENTS

Combination of two Absolute Segments is analogous to the union of two sets, where each byte in an Absolute Segment is analogous to a set element. However, if the same byte is defined in each Absolute Segment being combined, it will be defined twice in the resultant Absolute Segment. This causes an error message to be generated by LINK and LOCATE.

COMBINING TWO CODE SEGMENTS

Let S1 and S2 be Code Segments that are to be combined to form a single Code Segment S3. Let L1 and L2 be the lengths of the segments, respectively, as specified in their Module Header Records. Define L1p as the smallest multiple of 256 that is greater than or equal to L1. The following table indicates the alignment type and length of S3, as a function of the alignment types and lengths of S1 and S2.

S1 \ S2	IP	PR	BR
IP	L1 + L2 (IP) or L1p + L2 (PR)	L1p + L2 (PR)	L1 + L2 (PR)
PR	L1 + L2 or L1p + L2 (PR)	L1p + L2 (PR)	L1 + L2 (PR)
BR	L1 + L2 or L1p + L2 (PR)	L1p + L2 (PR)	L1 + L2 (BR)

When two In-page relocatable (IP) Code Segments are combined, the resultant Code Segment is In-page relocatable (IP) if L1 + L2 $\leq$ 256; otherwise the resultant Code Segment is Page relocatable (PR).

When S2 is In-page relocatable (IP) and S1 is not, the length of S3 will be L1 + L2 if L1p - L1 $\geq$ L2; otherwise the length of S3 will be L1p + L2.

Combination of two Code Segments is in general not commutative with respect to the layout of memory or the length of the combined segment. For example, the length of combining S1 with S2 may be different from combining S2 with S1.

When the size of the resultant Code Segment is L1p + L2, and L1p > L1, then the spaces between S1 and S2 are permanently lost. These areas represent a "gap" in the Code Segment. These gaps, when formed, are reported in the map produced by LINK.

COMBINING TWO DATA SEGMENTS

Data Segments are combined in exactly the same manner as Code Segments, as described in the preceding section.

COMBINING TWO STACK SEGMENTS

The length of the Stack Segment given in the Module Header Record is the minimum stack size required for successful execution of the Module.

The length of the combined Stack Segment is the sum of the lengths of the two Stack Segments being combined. If both Stack segments are Byte relocatable (BR), then the resultant Stack Segment will be Byte relocatable (BR); otherwise the resultant Stack Segment is Page relocatable (PR).

COMBINING TWO MEMORY SEGMENTS

The length of the Memory Segment given in the Module Header Record is the minimum number of bytes in this segment required by the Module.

The length of the combined Memory Segment is the larger length of the two Memory Segments being combined. If both Memory Segments are Byte relocatable (BR), then the resultant Memory Segment will be Byte relocatable (BR); otherwise the resultant Memory Segment will be Page relocatable (PR).

COMBINING TWO NAMED COMMON SEGMENTS

Named Common Segments are different from other segments in that they do not have pre-defined segment numbers. In fact, two Named Common Segments will be selected for combination not by the equality of their segment number, but by the equality of their Names as defined in the Named Common Definitions Record.

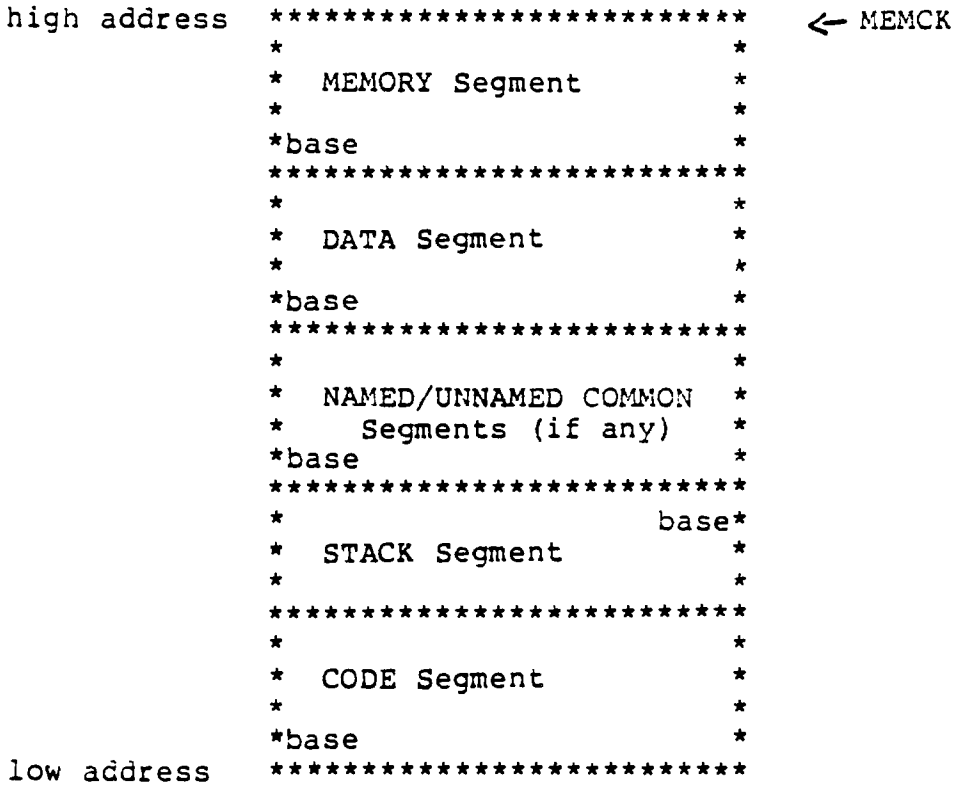
The lengths of the two Named Common Segments being combined must be the same; otherwise a warning message is generated. The length of the combined Named Common Segment is the larger length of the two Named Common Segments being combined (if the lengths are different). If both Named Common Segments are Byte relocatable (BR), then the resultant Named Common Segment will be Byte relocatable (BR); otherwise the resultant Named Common Segment will be Page relocatable (PR).

COMBINING TWO UNNAMED COMMON SEGMENTS

The length of the combined Unnamed Common Segment is the larger length of the two Unnamed Common Segments being combined. If both Unnamed Common Segments are Byte relocatable (BR), then the resultant Unnamed Common Segment will be Byte relocatable (BR); otherwise the resultant Unnamed Common Segment will be Page relocatable (PR).

APPENDIX: SEGMENT LOCATING

The LOCATE program will allow the user to assign absolute memory addresses to the different segments in an object module. This can be done using either user defined bases and orders or default bases and orders (see the ISIS-II USER'S GUIDE for more information on the LOCATE program). Default ordering will lead to the following memory organization:



This organization allows the Data Segment to grow dynamically into the unused memory space. By judicious choice of labels and the use of the MDS MONITOR function MEMCK, the user can programmatically determine the lengths of all Segments.

The length of the Memory Segment is always computed to be the amount of available memory on the system the module is LOCATEd. If the module is executed on the same system, the Memory Segment length calculated by LOCATE is correct. If the module is executed on a different system, the actual amount of memory depends on the configuration of that system. LOCATE has no knowledge of the configuration of the target system.

Note that all segments grow upward from their base except for the Stack Segment, which grows downward due to hardware design.

LOCATE will resolve references to the MEMORY and STACK bases. Translators will issue Inter-Segment References Records for the respective reserved words that reference these bases.

The base of the Absolute Segment is memory address zero.