MCS 80/85

ABSOLUTE OBJECT FILE FORMATS

An Intel Technical Specification

Order Number: 9800183B

intel®

PREFACE

This manual defines the internal format of absolute 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 a program to read 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.

ICE-30	INTELLEC	MULTIBUS
ICE-80	LIBRARY MANAGER	PROMPT
INSITE	MCS	UPI
INTEL	MEGACHASSIS	μSCOPE
	MICROAMP	

CONTENTS

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

INTRODUCTION TO OBJECT FILE FORMATS

The formats described in this technical specification are for absolute object files produced by Intel's language translators. An absolute object file contains a form of machine-language instructions and/or data that permits loading into memory for execution or interpretation. In addition, it contains control information that may govern the loading process, such as load addresses and a starting address (the address at which execution is to begin).

An absolute object file can be used to program a ROM or PROM. In this case it is loaded into RAM not for execution, but simply as data for the program that transfers it to the ROM or PROM. The RAM address at which it is loaded is usually not the one at which the program will execute properly.

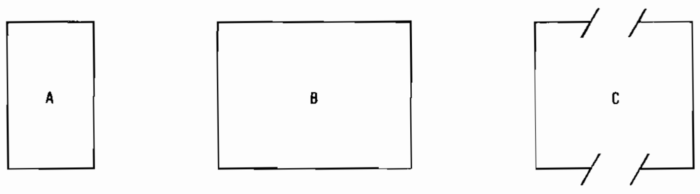
This document describes three absolute object file formats:

- 8080 absolute object file format for object code files produced by 8080 and 8085 language translators. This format is for absolute object files on diskette.
- Hexadecimal absolute object file format for paper tape.
- BNPF absolute object file format for paper tape.

Fields in absolute object files not described in this document are nonetheless reserved by Intel for certain uses. These uses may change from time to time; any use of or modification to the content of these fields may interfere with the proper functioning of the affected programs.

RECORD FORMAT DIAGRAMS

The record format diagrams use the following conventions:



(A) represents a single-byte field; (B) represents a 2-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 printing characters, but nothing in this specification requires such constraint.

Any field with an "X" through it contains unspecified information and is to be ignored.

8080 ABSOLUTE OBJECT FILE FORMAT

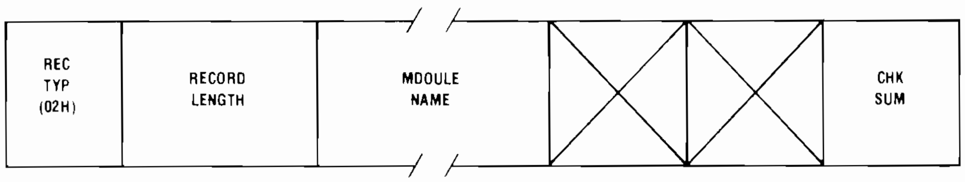
The 8080 Absolute Object File Format is a proper subset of the 8080 Object File Format. The absolute subset consists of three record formats:

1. Module Header record
2. Content record
3. Module End record

COMMON RECORD FIELDS

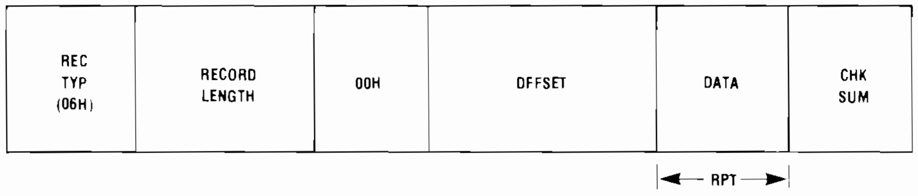
All record formats share this common organization: (1) The first byte identifies the record. This is the RECORD TYPE (REC TYPE) field. (2) The next two bytes are a field that contains a number called the RECORD LENGTH. It is the total number of bytes in the record, exclusive of the three bytes that comprise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

MODULE HEADER RECORD



Module Name. Every Module has a name. 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.

CONTENT RECORD



This Record provides contiguous data, from which a portion of a memory image may be constructed.

OFFSET. This field specifies the absolute location of the first data byte.

DATA. Following the OFFSET are one or more data bytes. Thus, this record provides *n* consecutive bytes of a memory image, from OFFSET through OFFSET+*N*-1, inclusive.

MODULE END RECORD

REC TYP (04H)	RECORD LENGTH	MOD TYP	00H	OFFSET	OPTIONAL INFO	CHK SUM

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.

OFFSET. If the Module is a main program, then this field specifies the Module's execution start address. Otherwise, this field has no significance (but must be present).

Optional Information. This field may be omitted from the Record, at the discretion of the translator.

SUMMARY

A proper Absolute Object File will contain at least the above record types in the following order:

- One Module Header record
- One or more Content records
- One Module End record

It may also contain other record types which, if present, will follow the Module Header record. These other record types fall into two categories:

- Extraneous — these usually contain symbolic debug information and should be ignored by an absolute loader.
- Erroneous — these usually contain relocation information (indicating that the information is not yet in absolute form) and should cause the rejection of the entire module by an absolute loader.

The category these other records fall into are:

- Extraneous — record type numbers 08H, 0EH, 10H, 12H, 16H, 18H, 20H
- Erroneous — all other record type numbers

HEXADECIMAL OBJECT FILE FORMAT FOR PAPER TAPE

Hexadecimal object file format is a way of representing an object file in ASCII. The ASCII character set is defined by the reference listed in the Preface.

The hexadecimal representation of binary is coded in ASCII. For example, the 8-bit binary value 0011 1111 is 3F in hexadecimal. To code this in ASCII, one 8-bit byte containing the ASCII code for 3 (0011 0011, or 33H) and one 8-bit byte containing the ASCII code for F (0100 0110, or 46H) are required. This representation (ASCII hexadecimal) requires twice as many bytes as the binary.

4-BIT AND 8-BIT DATA

A hexadecimal object file can contain either 8-bit or 4-bit data, but not both. Two ASCII hexadecimal digits are used to represent both 8-bit and 4-bit data. In the case of 4-bit data, only one of the digits is meaningful. Whether it is the high-order or the low-order digit must be specified to the program reading the file, and must be consistent throughout the file.

PARITY BITS

Because ASCII characters need only seven bits for their representation, the highest-order bit of each 8-bit byte can be used as a parity bit that generates the hexadecimal object file. However, when such a file is loaded by an Intel product, the highest-order bit is masked when the ASCII is converted to binary. Intel software does not generate parity bits when creating object files.

PAPER TAPE FORMAT

The format described here is for paper tape and does not define the format for other media, which may use record separators such as the ASCII code for carriage return.

On paper tape, one ASCII character requires one frame. The record format is described here according to the fields in the records.

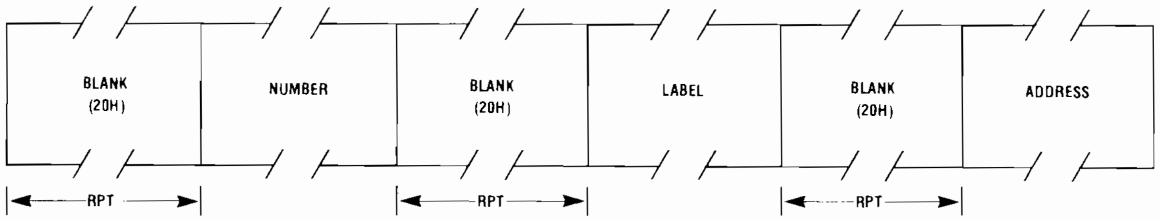
PARTS OF THE OBJECT FILE

The object code file contains two parts: the symbol table, which describes the symbols used in the program and data to be loaded, and the object code itself.

The symbol table is a series of records terminated by a dollar sign ($). Each record contains three fields separated by one or more ASCII spaces: a NUMBER field, a LABEL field, and an ADDRESS field. All fields must be present, although they may contain dummy values (such as zero) if they are not required.

The object code may be preceded and followed by a row of asterisks (*) so the object code can be visually located on paper tape. Each record in the object code may be preceded by one or more ASCII spaces.

SYMBOL TABLE RECORD



NUMBER. The NUMBER field contains the line number of the source statement that contains the label. This field is filled by the cross-product language translator. If not applicable, the field contains ASCII zeros (30H).

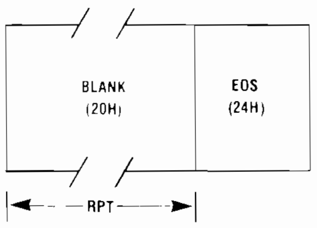
LABEL. The LABEL field contains the label of a symbol from the source program or a symbol generated by the language translator.

ADDRESS. The ADDRESS field contains the address of the symbol as an absolute value. The address must begin with a decimal digital (0-9) and end with a character that identifies the base:

H	Hexadecimal
O or Q	Octal
B	Binary
D	Decimal

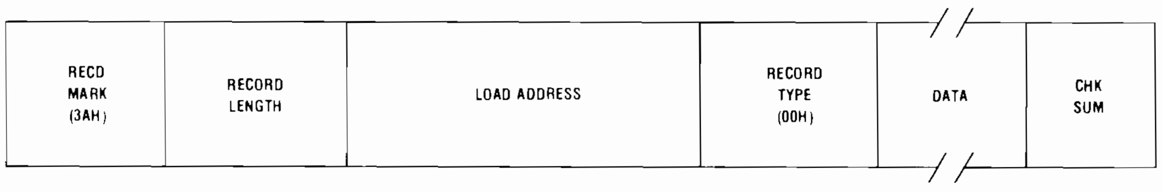
If the base identifier is omitted, decimal is assumed.

END-OF-SYMBOL-TABLE RECORD



EOS. The END-OF-SYMBOL-TABLE record contains 24H, the ASCII code for a dollar sign ($), as the first non-blank character.

DATA RECORD



RECD MARK. The RECORD MARK field contains 3AH, the ASCII code for a colon (:).

RECORD LENGTH. The RECORD LENGTH field contains the number of data bytes in the record. The high-order digit comes first. Maximum value is 255 (FFH).

LOAD ADDRESS. The LOAD ADDRESS field contains the address at which the data is to be loaded. The high-order digit comes first, proceeding to the low-order digit. The first data byte is loaded at this storage location; successive bytes are stored in successive storage locations.

RECORD TYPE. The RECORD TYPE field in a DATA record contains 00H.

DATA. The DATA field contains a pair of hexadecimal digits that represent the ASCII code for each data byte. The high-order digit is the first digit of each pair.

CHK SUM. The CHECKSUM field contains the ASCII hexadecimal representation of the twos complement of the 8-bit sum of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from the RECORD LENGTH field to and including the last byte of the DATA field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the RECORD LENGTH field to and including the CHECKSUM field, is zero.

END-OF-FILE RECORD

RECD MARK (3AH)	RECORD LENGTH (3030H)	LOAD ADDRESS	RECORD TYPE (01H)	CHK SUM

RECD MARK. The RECORD MARK field contains 3AH, the ASCII code for a colon (:).

RECORD LENGTH. The RECORD LENGTH field contains two ASCII zeros (3030H).

LOAD ADDRESS. The LOAD ADDRESS field contains either ASCII zeros (30H) or the starting address of the program.

RECORD TYPE. The RECORD TYPE field contains 01H.

CHECKSUM. The CHECKSUM field contains the ASCII hexadecimal representation of the twos complement of the 8-bit sum of the 8-bit bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from the RECORD LENGTH field to and including the last byte of the DATA field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the RECORD LENGTH field to and including the CHECKSUM field, is zero.

EXAMPLE

Figure 1 shows an example of hexadecimal absolute object file format. Each record is on a separate line. Note the dollar sign ($) separating the symbol table records from the object code records, and the colon at the beginning of each object code record.

```
0 BLOCK01 0
0 ACTUA 0318CH
0 AFT   0317EH
0 BEGIN 03100H
0 BUFFE 03196H
0 CAFT  0317AH
0 CBLK  0317AH
0 CLOSE 00001H
0 DONE  03150H
0 EBLK  03192H
0 ERR   03160H
0 ERROR 0000CH
0 EXIT  00009H
0 ISIS  00040H
0 LOOP  03127H
0 OBLK  03170H
0 OPEN  00000H
0 RBLK  0317EH
0 READ  00003H
0 STACK 03216H
0 STATU 03192H
0 WBLK  03188H
0 WRITE 00004H
0 XBLK  03190H
 $
:10310000311A320E03117E31CD40003A9231B7C2EE
:1031100060310E00117031CD40003A9231B7C2607B
:10312000312A7E31227A310E03117E31CD40003AB0
:103130009231B7C260312A8C317CB5CA50310E044D
:10314000118831CD40003A9231B7C26031C3273186
:103150000E01117A31CD40000E09119031CD4000A1
:103160000E0C119231CD40000E09119031CD40006E
:0A3170007E3196310100000092311B
:10317C0092310100963180008C3192310000963 1F1
:04318E009231923 1B7
:02319400923176
:00310001CE
```

Figure 1. Hexadecimal Object File

BNPF OBJECT FILE FORMAT FOR PAPER TAPE

The BNPF object file format contains two parts: the object code preceded by a symbol table. Only the object code is in BNPF. The format of the symbol table is identical to that for hexadecimal object files described in Chapter 2.

The BNPF object code uses the ASCII characters N and P to represent the actual binary digits 0 and 1 respectively. An ASCII "B" indicates the beginning of a byte, an ASCII "F" indicates the end of the byte. All characters following the F are ignored until another B is encountered. This allows comments that do not contain the character B to appear between bytes of data in BNPF format.

Ten bytes (frames on paper tape) are required to represent one byte of data. For example, the byte containing the value 0011 1111 (3F in hexadecimal) would be represented as follows in BNPF:

```
BNNPPPPPPF
```

Exactly 8 characters must occur between the B and its following F. No character other than N or P can be used between the B and the F.

A BNPF object file may contain either 8-bit or 4-bit data but not both. In the case of 4-bit data, only the high-order or low-order four bits are meaningful. Which bits are meaningful must be known to the program reading the file and must be consistent throughout the file.

INDEX

intel®

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A./B-84/1277/5K BL