intel®

ISIS-II ICE-80™ SUPPLEMENT
to ICE-80™ Operator's Manual (98-185C)

DISCLAIMER

Intel makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Intel reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Intel to notify any person of such revision or changes.

98-368A

PREFACE

This document is a supplement to the ICE-80 Operator's Manual, Intel* document number 98-185C. It describes the changes to the ICE-80** software driver (ICE80SD) that adapt it to the ISIS-II environment. Throughout this document, the term ICE80SD refers to the ISIS-II ICE-80 software driver unless otherwise noted.

It is assumed that you have read and understood the Operator's Manual mentioned above and the ISIS-II System User's Guide, Intel document number 98-306.

CONTENTS

1. INTRODUCTION

This document explains the differences between ISIS-II ICE80SD and older versions of ICE80SD. You will find that these differences are simple, and if you are already an ICE80SD user, you should be able to make the transition easily.

Most of the differences are new features that allow the ICE80SD user to deal efficiently with files containing multiple program modules. Such files are a major feature of the ISIS-II environment, as described in the ISIS-II System User's Guide and in the following section.

ISIS-II OBJECT FILES

Under ISIS-II, program files that are ready to load and run are absolute binary-coded OBJECT files. These files differ from the HEX files of older implementations. The following is a list of the differences that affect the operation of ISIS-II ICE80SD.

- The file may contain more than one module of code, if it was created by using LINK to combine two or more modules. These modules are loaded by ICE80SD in the same order in which they are found in the input file (which is the same order in which they were specified to LINK).
- For each module created from PL/M source code, the file may contain statement numbers. These numbers correspond to the original PL/M statements, and each number is associated with the first instruction compiled from the corresponding PL/M statement.
- The file may contain a module name for each module.
- The file may contain a separate symbol table for each module.

An absolute OBJECT file can be created in two ways:

- The ASM-80 assembler produces an absolute OBJECT file if the appropriate control is used to produce absolute code.
- LOCATE produces an absolute OBJECT file.

The module names, statement numbers, and symbol tables are collectively referred to as "debugging information." The module names

and symbol tables will be present in files assembled with the ASM-80 assembler, unless suppressed with the NODEBUG control. The debugging information will be present in files compiled with the PLM-80 compiler only if the DEBUG control is used.

The ISIS-II ICE80SD commands provide features for making use of this information in examining and debugging the code portion of the file.

SUMMARY OF NEW FEATURES IN ISIS-II ICE80SD

The following sections summarize the new features of ICE80SD. These features are described in more detail in Sections 2 through 10.

EFFECTS ON LOAD AND SAVE

The LOAD command of ISIS-II ICE80SD can only load an absolute OBJECT file. In loading a large file, it may be necessary to save memory space. Accordingly, it is possible to eliminate the code, the symbol tables, or the statement numbers (see below) from the load by using the new controls NOCODE, NOSYMBOLS, or NOLINES respectively. See Section 2 and Appendix A.

The SAVE command of ISIS-II ICE80SD creates an absolute OBJECT file. See Section 3.

USE OF MODULE NAMES

In earlier versions of ICE80SD, there was only one symbol table, and any reference to a symbol was looked up in this table. In ISIS-II ICE80SD, there is a separate symbol table for each module loaded.

In referring to a symbol, you may prefix it with a module name; the symbol will then be looked up in that module's symbol table.

Similarly, a module name may be prefixed to a statement number. Any module name can also be prefixed with another module name. See Section 5.

Also, module names are used by themselves in the new commands DEFAULT and REMOVE. See Sections 6 and 8.

THE DEFAULT INTERVAL

If a symbol reference is not prefixed with a module name, it is looked up in the symbol tables that lie within the default interval.

The default interval is initially the set of all modules in ICE80SD's

"module table" -- that is, all modules that have been loaded, except those that have been deleted with the REMOVE command (see Section 8). In this situation, a symbol reference without a module name causes all of the symbol tables to be searched in sequence until the symbol is found.

At any given time during an ICE80SD session, you will probably be working with one particular module, and it is cumbersome to have to prefix every symbol reference with the module name. To avoid this, simply change the default interval to include only the module that you are working on. This can be done by means of the DEFAULT command (see Section 6).

STATEMENT NUMBERS

A module that was originally written in PL/M may contain statement numbers, if the module was compiled using the DEBUG control of the PLM-80 compiler. The PL/M statements in the source module are numbered in sequence (starting with 1) by the compiler, and in the object-code module, each statement number is associated with the first instruction compiled from the corresponding PL/M statement.

A statement number is specified in ICE80SD commmands by prefixing a constant with a # character. For instance,

```
#25
```

is a reference to the location of the first instruction compiled from PL/M statement number 25 in the source module.

Like a symbol reference, a statement number can be prefixed with a module name. When this is done, the reference is to the specified statement number within the specified module.

If a statement number is not prefixed with a module name, it refers to the specified statement number in the first module in the default interval. See Section 4.

OTHER NEW FEATURES

The DISPLAY command has a new option, DISPLAY ALL MODULES. This lists the names of all modules in ICE80SD's module table. See Section 9.

Also, the DISPLAY ALL SYMBOLS command now lists only the symbols in the symbol tables in the default interval. The symbols are listed in groups under their module names. See Section 9.

A new command, REMOVE, is provided to delete symbols from symbol tables and to delete an entire module. See Section 8.

The action of the EQUATE command is affected by the new definition of a <symbol-name> (see below) and by the default interval. See Section 7.

The ICE80SD program itself now consists of four separate files:

ICE80
ICE80.OV0
ICE80.OV1
ICE80.OV2

All of these files must be on the same diskette; however, this diskette may be in any drive. The last three files must not be renamed.

NEW DEFINITIONS

The following new definitions apply to ISIS-II ICE80SD.

- An <identifier> is a string of ASCII characters beginning with a letter and not including the characters

  " , / = ; - @ [ ] < > + : # space
  carriage-return rubout control-X control-R escape

  This resembles the old definition of a <symbol-name> -- however, the characters : and # cannot be included in an <identifier>, although they were allowed in the old definition.

  An <identifier> does not appear explicitly in any ICE80SD command. However, we will use it in stating the other new definitions below.

- A <module-name> is an <identifier> followed by a colon. It is assumed that the <identifier> is the name of a module in ICE80SD's module table. The following are examples:

  MAIN:
  AUX:
  UTIL:
  MATH:

  A <module-name> may also be formed by stringing single

<module-name>s together, as in the following examples:

```
MAIN:MATH:
AUX:UTIL:
MAIN:MATH:AUX:
```

The meaning of this construction is explained in Section 5.

- A <symbol-name> may now be any of the following:
  - An <identifier> (as in older versions of ICE80SD).
  - Two or more <identifier>s separated by slashes (as in older versions of ICE80SD).
  - A <module-name> as defined above, followed by either a single <identifier> or two or more <identifier>s separated by slashes.

  The following are examples of valid <symbol-name>s:

```
HIGH
CORR/HIGH
CORR/HIGH/TMP
AUX:CORR
MAIN:AUX:HIGH
MAIN:AUX:CORR/HIGH
```

  The meanings of these constructions are explained in Section 5.

- The definition of a <value> is now somewhat expanded. A <value> may be any of the following:
  - A constant (as in older versions of ICE80SD).
  - A constant prefixed with a # character to denote a statement number (see description above; this is a new feature). A statement number may also be prefixed with a <module-name>. See Section 5.
  - A <symbol-name> as defined above. The value of this is the memory address of a source-language symbol, or the assigned value of a symbol created with the EQUATE command.
  - A register-pair name prefixed with an @ character, to denote the contents of the register pair (as in older versions).
  - The sum or difference of any two <value>s (as in older versions).

- A <value> enclosed in square brackets, to denote the contents of a double-memory location (as in older versions). The <value> inside the brackets is the address of the double-memory location.

The new definitions of <symbol-name> and <value> apply to all the commands described in the ICE-80 Operator's Manual where <symbol-name>s and <value>s can be used.

2. CHANGES IN THE LOAD COMMAND

The new format for the LOAD command is

LOAD <file-name> {NOCODE}{NOSYMBOLS}{NOLINES}

where

<file-name> is the name of an absolute OBJECT file to be loaded.

NOCODE causes all of the code in the input file to be omitted.

NOSYMBOLS causes all of the symbol tables in the input file to be omitted.

NOLINES causes all of the statement numbers in the input file to be omitted.

The controls NOCODE, NOSYMBOLS, and NOLINES do not have to be entered in the order shown.

If the input file is not a valid absolute OBJECT file, the LOAD command issues the error message

ERR=64

and aborts. Control returns to the command level of ICE80SD.

If any module in the input file contains an external symbol reference that is not satisfied by linkage to another module in the file, the LOAD command issues the error message

ERR=69

This is a warning message only. The LOAD completes normally, and ICE80SD operation can continue.

If the program you have loaded is incomplete, and you know that you can work with it without access to the "unsatisfied" symbol, go ahead.

However, if there is any attempt to access the "unsatisfied" symbol -- either by the program or by ICE80SD commands -- the results are undefined.

After any LOAD command, the default interval is reset to include all modules in ICE80SD's module table.

When a module containing no debugging information is loaded, its name is added to the module table but there is no other symbolic information. A subsequent LOAD may cause the module name to disappear from the module table, but the code of the module is not affected by this.

3. CHANGES IN THE SAVE COMMAND

The format of the SAVE command is unchanged. SAVE writes out the code in the specified <partition>, and all symbol tables and statement numbers (as modified by ICE80SD commands).

If an input file containing a start address has been loaded, the output file will also contain this start address. If more than one start address has been loaded by multiple LOAD commands, the output file will contain only the last start address loaded.

Information other than code, symbol tables, statement numbers, and one start address may be lost by SAVE. For example, if the input file contained a PUBLIC declaration, it is lost by SAVE.

If no LOAD has been performed, SAVE produces an output file containing only a single module. This module contains symbols created with the EQUATE command, if any.

4. STATEMENT NUMBERS

A statement number is specified by prefixing a constant with a # character. This construction is allowed wherever a <value> is allowed in ICE80SD commands, and is translated by ICE80SD into the memory address of the first instruction compiled from the corresponding PL/M statement.

Statement numbers are present only in modules compiled from PL/M source code using the DEBUG control.

Statement numbers may be prefixed with <module-name>s, as described in the next section.

5. MODULE NAMES

The module names in an OBJECT file are the names of the original source-language modules. The modules occur in the OBJECT file in the order in which they were linked together by LINK. ISIS-II ICE80SD preserves this order, and if two or more separate LOAD commands are given, the modules loaded by each LOAD follow the modules that are already in ICE80SD's module table.

The names of all modules in ICE80SD's module table can be listed in order by means of the new DISPLAY ALL MODULES command (see Section 9). This is useful for checking the order of the modules.

A <module-name> may be used to qualify any symbol reference or statement number reference in an ICE80SD command. This is done by prefixing the <symbol-name> or statement number with the <module-name> as shown in the following examples.

```
MAIN:INIT
```

means the first occurrence of the symbol INIT in the symbol table for the first module named MAIN.

```
MATH:#13
```

means the address of the first instruction compiled from PL/M statement 13 of the first module named MATH.

A <module-name> may also be used to qualify any other <module-name>, in cases where there are two or more modules with the same name. Thus

```
MATH:AUX:CORR
```

means the first occurrence of the symbol name CORR in the symbol table for the first module named AUX that follows the first module named MATH.

The use of <module-name>s may be combined with the qualification of one symbol name by another. The reference

```
MATH:INDEX/LIM
```

causes the following search for LIM:

- Starting at the beginning of the module table, find the first module named MATH.
- Within the symbol table for module MATH, find the first occurrence of the symbol INDEX.
- Starting from this point, find the first occurrence of the symbol LIM within this same symbol table.

The search will succeed only if both symbols are found in the symbol table of module MATH, in the specified order.

The reference

```
MAIN:AUX:HIGH/CORR
```

causes the following search for CORR:

- Find the first module named MAIN.
- Starting from this point, find the first module named AUX.
- Within the symbol table for module AUX, find the first occurrence of the symbol HIGH.
- Starting from this point, find the first occurrence of the symbol CORR within this same symbol table.

The search will succeed only if both modules are found in the specified order, and both symbols are found within the symbol table for module AUX, in the specified order.

6. THE DEFAULT COMMAND

The new DEFAULT command is used to change the default interval. The default interval is either the set of all modules in ICE80SD's module table, or one specific module. When an ICE80SD command contains a symbol reference or statement-number reference without a <module-name>, the symbol or statement number is searched for in the default interval. A more complete discussion of the default interval can be found in Section 1.

The DEFAULT command has the format

DEFAULT {<module-name>}

where

<module-name> is defined in Section 1.

If no DEFAULT command has been given, the default interval is the set of all modules in ICE80SD's module table. Any symbol reference that is not qualified with a <module-name> will be processed by searching all symbol tables in sequence until the first occurrence of the symbol name is found. Any statement number reference that is not qualified with a <module-name> will be processed by searching only the symbol table for the first module.

If a <module-name> is used in a DEFAULT command, the default interval consists of the specified module. Thus

*DEFAULT MAIN:

causes the default interval to be the first module named MAIN. Any symbol reference or statement number reference that is not qualified with a <module-name> will be looked up only in the symbol table for module MAIN.

The <module-name> may be qualified with another <module-name>, as follows:

*DEFAULT UTIL:AUX:

This causes the default interval to be the first module named AUX that follows the first module named UTIL. Any symbol reference or statement number reference that is not qualified with a <module-name> will be looked up only in the symbol table for module AUX.

If a DEFAULT command with no <module-name> is given, the default interval is (again) the set of all modules in ICE80SD's module table.

Also, the default interval is reset to include all modules in ICE80SD's module table after each LOAD command and after any REMOVE command (see Section 8) that deletes a module from the default interval.

7. CHANGES IN THE EQUATE COMMAND

The format of the EQUATE command is unchanged, but the new definition of a <symbol-name> (see Section 1) affects the operation of EQUATE.

EQUATE first searches for the symbol, using the rules given in Sections 5 and 6. If the search succeeds, meaning that the symbol already exists, EQUATE issues the error message

```
ERR=6
```

and makes no change in the symbol tables.

If the search fails, meaning that the symbol does not already exist, EQUATE creates it, assigns the specified <value> to it, and places it at the end of the interval that it has just searched. The name of the new symbol is the last <identifier> in the <symbol-name> in the command. Thus

```
*EQUATE AUX:CORR=2
```

causes CORR to become the last symbol in the symbol table for the first module named AUX -- if and only if AUX does not already contain a symbol named CORR.

If no <module-name> is used, the new symbol becomes the last entry in the symbol table for the last module in the default interval. Thus

```
*EQUATE LIM=100H
```

causes LIM to become the last entry in the symbol table for the last module in the default interval -- if and only if no symbol named LIM already exists in any module in the default interval.

The command

```
*EQUATE CORR/LIM=100H
```

first causes a search, within the default interval, for CORR. If CORR is not found, the search fails and LIM is created and placed at the

end of the last symbol table in the default interval. If CORR is found, we next search for LIM. If LIM is not found, the search fails and the new symbol LIM becomes the last symbol in the last symbol table in the default interval.

Note that the effect of this is different from the previous example only if there is already a symbol named LIM which comes before CORR. If this is the case, then the command

```
*EQUATE LIM=100H
```

will find the LIM that already exists, issue an error message, and do nothing else. However, the command

```
*EQUATE CORR/LIM=100H
```

does not look for LIM until it has already found CORR. When it finds that no symbol named LIM occurs after CORR, it creates LIM and places it at the end of the default interval.

8. THE REMOVE COMMAND

The new REMOVE command has three formats, and can be used to delete a single symbol, an entire module, or a set of symbols.

DELETING A SINGLE SYMBOL

The first format for the REMOVE comand is

REMOVE <symbol-name>

where

<symbol-name> is defined in Section 1.

The specified symbol is searched for, using the rules given in Sections 5 and 6, and if found it is deleted. Thus

*REMOVE MAIN:HIGH

causes the symbol HIGH to be deleted from the symbol table of the first module named MAIN.

*REMOVE LOW

causes the symbol table(s) in the default interval to be searched in sequence. The first occurrence of the symbol LOW is deleted from the symbol table where it is found.

DELETING A MODULE

The second format for the REMOVE command is

REMOVE <module-name>

where

<module-name> is defined in Section 1.

This causes the specified module to be deleted. Thus

*REMOVE AUX:

causes the first module named AUX to be deleted.

*REMOVE AUX:UTIL:

searches for the first module named UTIL that follows the first module named AUX, and deletes the module named UTIL.

Everything associated with the module is deleted -- the module name, the symbol table, and the statement numbers (if any). Note that the code is not deleted. Once loaded, the code of a module is not associated with the module.

If the default interval has been set to one specific module, and this module is deleted with a REMOVE command, then the default interval is automatically reset to include all modules in ICE80SD's module table.

DELETING A SET OF SYMBOLS

The third format for the REMOVE command is

REMOVE <symbol-name> TO <symbol-name>

where

<symbol-name> is defined in Section 1.

This searches for both symbols, using the rules given in Sections 5 and 6. If both are found, and the first symbol is found before the second one, all symbols from the first to the second (inclusive) are deleted.

If the second symbol is found before the first, REMOVE issues the error message

ERR=10

and does not delete any symbols.

The command

*REMOVE AUX:HIGH TO AUX:CORR

searches within the first module named AUX for the symbols HIGH and CORR. If they are found in the specified order, all symbols from HIGH to CORR (inclusive) are deleted from the symbol table for the module AUX.

The command

*REMOVE LOW TO LIM

searches all symbol tables in the default interval for the symbols LOW and LIM. If they are found in the specified order, all symbols from LOW to LIM (inclusive) are deleted. If LOW and LIM are found in different modules, this includes all symbols in intervening modules -- but not the module names or statement numbers.

9. CHANGES IN THE DISPLAY COMMAND

DISPLAY ALL SYMBOLS now lists only the symbols in the modules in the default interval. It groups the symbol names by modules, with the module name as a heading for each group.

DISPLAY ALL MODULES is a new option which lists the names of all modules in ICE80SD's module table.

10. CHANGES IN EFFECT OF BASE COMMAND

If a BASE SYMBOLIC command is given, values subsequently displayed will be in the form of a symbol name followed by a + sign and a decimal constant (as in older versions of ICE80SD). However, ISIS-II ICE80SD will only use symbols from the symbol tables of modules in the default interval. Therefore, the symbol displayed is not necessarily the symbol whose value is actually closest to but not greater than the value being displayed -- it is merely the closest one within the default interval.

Appendix A
REDUCING THE SIZE OF INPUT FILES

A file containing code and debugging information may be large, and in some cases -- especially when using a 32K system -- it may be too large to be loaded under ICE80SD. The following are some suggestions for overcoming this problem.

- At any time during an ICE80SD session, the register-pair UPPERLIMIT contains the highest available physical memory address. This is useful for determining how much memory space is available (see Chapter 1 of ICE-80 Operator's Manual).

- One approach to the problem is to write your program in small modules, and debug them one at a time (or a few at a time). If you do this, you only need the debugging information for the module(s) being debugged. When you compile or assemble the modules of the program, use the NODEBUG control of the compiler or assembler to eliminate debugging information from the other modules.

- You can eliminate all debugging information from your program file by using the PURGE control in LOCATE, and you can eliminate symbol tables and/or statement numbers by using the NOSYMBOLS and/or NOLINES controls in the LOAD command of ICE80SD. Then, by using an absolute symbol-table listing (obtained from LOCATE, or from ASM-80 if you used it to produce absolute code), you can use the EQUATE command to restore only the symbols that you need for debugging purposes. (Note that you cannot restore statement numbers.)

- You can load the file with the NOCODE control in the LOAD command, to load only the symbol tables (and statement numbers, if they exist). Then use the REMOVE command to delete all symbols that you do not need for debugging purposes. You can get rid of all symbols and statement numbers for a module by deleting the module with a REMOVE command. Finally, LOAD the file again using NOSYMBOLS and NOLINES, to load only the code. You now have the code and a selected set of debugging information loaded into ICE80SD.

- Note that after using either of the last two methods to obtain a cut-down version of the program within ICE80SD, you can save this cut-down version as a file by using the SAVE command. The resulting file can later be loaded into ICE80SD in a single operation.