iRMX™ 86 BOOTSTRAP LOADER REFERENCE MANUAL

CONTENTS

CONTENTS (continued)

TABLE

FIGURES

***

CHAPTER 1
INTRODUCTION

The Bootstrap Loader is a means of loading part or all of an application from secondary storage into RAM, either upon system reset or under program control. With the Bootstrap Loader you have flexibility that can simplify system maintenance and increase the versatility of your hardware.

The Bootstrap Loader operates in two stages. The first stage determines the location of the second stage and the name of the load file, then loads part of the second stage and passes control to the second stage. The second stage finishes loading itself, transfers the load file into memory, and passes control to the load file. The load file usually consists of an iRMX 86 application system. Getting the load file into memory and passing control to it is the objective of the bootstrap loading process.

A device driver is a small program that interfaces between your software and a hardware device (or a controller for the device). When you perform Bootstrap Loader configuration, which is independent of application system configuration with the ICU, you specify the device drivers that the Bootstrap Loader requires. As you complete the Bootstrap Loader configuration process, the device drivers needed for bootstrap loading -- which are distinct from (although possibly identical to) the drivers needed by the application itself -- are linked in automatically.

The following sections contain miscellaneous facts that will enable you to understand the later discussion about incorporating the Bootstrap Loader into your system.

THE FIRST STAGE OF THE BOOTSTRAP LOADER

The first stage consists of two parts. One part is the code for the first stage. It varies from 100 to 1000 bytes (and averages about 500 bytes) in length, depending upon the options you request during configuration. The other part is a set of minimal device drivers the first and second stages need to accomplish their objectives.

When the bootstrap loading process begins, the first stage can be in either of two places. If you are still developing your application, you can keep your first stage in secondary storage on your development system, then load it and start it running by means of the iSBC 957B package or the iSDM 86 or iSDM 286 System Debug Monitor. You can also burn the first stage into ROM along with the iSDM 86 or iSDM 286 monitor. When your application is finished and ready to use, you will probably burn the first stage into ROM, so it can be invoked when you turn on or reset your system.

THE SECOND STAGE OF THE BOOTSTRAP LOADER

Unlike the first stage, the second stage is not configurable. That is, it is always the same -- its size is less than 8K bytes -- and does not depend on the application in any way. Because of this, the software that formats iRMX 86-based random access volumes places the random-access version of the second stage on every named volume it formats, so it is always available for loading applications residing on random-access devices.

When the application system begins to run, RAM that had been used or occupied by the first and/or second stage becomes part of the memory pool for the application system.

NOTE

You must ensure that the memory locations occupied by the first stage, the second stage, and the load file (application system) are mutually non-overlapping.

THE LOAD FILE

The load file must be placed on an iRMX 86-based named volume of secondary storage. Recall that this volume also contains the second stage of the Bootstrap Loader.

DEVICE DRIVERS

For every bootstrap device in your system, you must include a device driver. As part of the iRMX 86 product, Intel has provided you with many device drivers that are specifically for bootstrap loading. These drivers are for the following controllers:

- iSBC 204 Flexible Diskette Controller
- iSBC 206 Disk Controller
- iSBC 208 Flexible Disk Drive Controller
- iSBC 215 Winchester Disk Controller
- iSBX 218A Flexible Disk Controller when used with the iSBC 215 controller
- iSBX 218A Flexible Disk Controller when used on a CPU board
- iSBC 220 SMD Disk Controller

- iSBX 251 Bubble Memory Controller
- iSBC 254 Bubble Memory Controller
- SASI (Shugart Associates Systems Interface) Peripheral Bus Controller
- SCSI (Small Computer Systems Interface) Peripheral Bus Controller

These drivers are small, ranging from 300 to 1000 bytes in length, and averaging about 500 bytes.

If you need additional device drivers, see Chapter 4.

***

CHAPTER 2
CONFIGURATION

The key to using the Bootstrap Loader is to ensure that the first stage is properly configured into your application. How to do that is the subject of this chapter. (Recall that the second stage is constant and therefore does not have to be configured.)

Configuring the first stage of the Bootstrap Loader consists of editing three or more configuration files. If you have devices for which Intel does not supply a device driver, you must prepare a device driver for each of them. Chapter 4 describes how to do this.

The configuration files are the following:

BS1.A86	This assembly language source file consists primarily of macros that describe the device units that can be used for bootstrap loading and the manner in which the bootstrap device and load file are to be selected.
BSERR.A86	This assembly language source file consists primarily of macros that tell the Bootstrap Loader what to do when bootstrap loading is not successful.
B204.A86 B206.A86 B208.A86 B215.A86 B218.A86 B251.A86 B254.A86 BSASI.A86 BSCSI.A86	These assembly language source files contain configuration information about device drivers that your bootstrap system can use.
BS1.CSD	This SUBMIT file contains the commands needed to assemble the preceding source files, to link together the resulting modules (and any others that you supply), and to locate the resulting object module.

The files requiring editing are BS1.A86, BSERR.A86, and BS1.CSD.

BS1.A86 CONFIGURATION FILE

The BS1.A86 file, shown in Figure 2-1, consists of two INCLUDE statements and several macros. The BS1.INC file contains the definitions of the macros in the BS1.A86 file.

```
 name bsl

$include(:f1:bcico.inc)
$include(:f1:bsl.inc)

%cpu(8086)
;iAPX_186_INIT(y,ofc38h,none,80bbh,none,003bh)  ; 188/48 board
;iAPX_186_INIT(y,none,none,80bbh,none,0038h)    ; 186/03 and 186/51
                                                ;      boards
%console
%manual
%auto
%loadfile
%defaultfile('/system/rmx86')
%retries(5)

;cico

;    iSBC 86/05/12A/14/30
;serial_channel(8251a,0d8h,2,8253,0d0h,2,2,8)
;
;    iSBC 351 (on iSBX #0)
;serial_channel(8251a,0A0h,2,8253,0B0h,2,2,8)
;
;    8MHz iSBC 186/03/51
;serial_channel(8274,0d8h,2,80186,0ff00h,2,0,0dh)
;serial_channel(8274,0dah,2,80186,0ff00h,2,1,0dh)
;serial_channel(8274,0dah,2,80130,0e0h,2,2,034h)
;
;    6MHz iSBC 186/03/51
;serial_channel(8274,0d8h,2,80186,0ff00h,2,0,0ah)
;serial_channel(8274,0dah,2,80186,0ff00h,2,1,0ah)
;serial_channel(8274,0dah,2,80130,0e0h,2,2,027h)
;
;    iSBC 188/48 SCC #1
;serial_channel(82530,0d0h,1,82530,0d0h,1,0,0eh,a)
;serial_channel(82530,0d2h,1,82530,0d2h,1,0,0eh,b)
;
;
;
;
;
%device(f0, 0, deviceinit204, deviceread204)
%device(f1, 1, deviceinit204, deviceread204)
%device(f2, 2, deviceinit204, deviceread204)
%device(f3, 3, deviceinit204, deviceread204)
%device(af0, 0, deviceinit208gen, deviceread208gen)
%device(af1, 1, deviceinit208gen, deviceread208gen)
%device(af2, 2, deviceinit208gen, deviceread208gen)
%device(af3, 3, deviceinit208gen, deviceread208gen)
```

Figure 2-1. First Stage Configuration File BS1.A86

```
%device(d0, 0, deviceinit206, deviceread206)
%device(w0, 0, deviceinit215gen, deviceread215gen)
%device(wf0, 8, deviceinit215gen, deviceread215gen)
%device(wf1, 9, deviceinit215gen, deviceread215gen)
%device(wf2, 10, deviceinit215gen, deviceread215gen)
%device(wf3, 11, deviceinit215gen, deviceread215gen)
%device(pmf0, 0, deviceinit218Agen, deviceread218Agen)
%device(bx0, 0, deviceinit251, deviceread251)
%device(b0, 0, deviceinit254, deviceread254)
%device(sa0, 0, deviceinitsasi, devicereadsasi)
%device(sc0, 0, deviceinitscsi, devicereadscsi)
%end
```

Figure 2-1. First Stage Configuration File BS1.A86 (continued)

The following sections describe the functions of the macros in the BS1.A86 file. For each macro, if a percent sign (%) precedes the name, then the macro is included (invoked). If a semicolon (;) precedes the name, then the macro is treated as a comment and is not included.

The BS1.A86 file does not specifically mention iSBC 220 SMD devices because they are covered by the entries containing "215".

In each %DEVICE macro shown in Figure 2-1 as having "gen" as a suffix on its last two parameters, those parameters can also be present without the suffix. That is, for each of those macros, Intel has supplied two versions of the device$init and device$read procedures, one with the "gen" suffix and one without the suffix. The "gen" (for general) versions, which provide automatic device recognition (see Appendix A), require more (about 500 bytes) code.

%CPU MACRO

You must include the %CPU macro, to identify the type of CPU that performs the bootstrap loading operation.

The form of the %CPU macro is:

```
%CPU(cpu_type)
```

where:

cpu_type	8086, 8088, 80186, 80188, or 80286. These are informal names for the Intel processors whose formal names are iAPX 86, iAPX 88, iAPX 186, iAPX 188, and iAPX 286, respectively.

iAPX_186_INIT MACRO

The iAPX_186_INIT macro specifies the initial chip select and mode values for 80186 and 80188 CPUs. Include this macro if and only if the CPU type is either 80186 or 80188.

The form of the %iAPX_186_INIT macro is:

```
%iAPX_186_INIT(RMX, UMCS, LMCS, MMCS, MPCS, PACS)
```

where RMX must contain "y" as it does in the file. The remaining parameters define initial values for the chip-select control registers. They stand, respectively, for upper-memory chip-select, lower-memory chip-select, midrange-memory chip-select, memory-peripheral chip-select, and peripheral-address chip-select block address. These registers are described in the data sheets for the iAPX 186 and iAPX 188 processors.

%CONSOLE, %MANUAL, AND %AUTO MACROS

The %CONSOLE, %MANUAL, and %AUTO macros let you specify how the first stage is to identify the load file and the device where the file will be found.

You can include any combination of the %CONSOLE, %MANUAL, and %AUTO macros. Because including %MANUAL causes the automatic inclusion of both %CONSOLE and %AUTO, there are five functionally-distinct combinations of these macros. The following indicates the significance of each of the five combinations.

None

(Requires that the device list, defined by means of the %DEVICE macro, have only one entry.)

- It (the Bootstrap Loader) tries once to load from the device in the device list.
- It tries once to load the file with the default pathname (either the system default or one you define by means of the optional %DEFAULTFILE macro).

%CONSOLE only

(Requires that the device list have only one entry.)

- It tries once to load from the device in the device list.
- It issues an asterisk (*) prompt for a pathname at the application system terminal and then tries once to load the file the operator specifies.
  - If a pathname is entered, it loads the file with that pathname.
  - If only <cr> is entered, loads the file with the default pathname.

%MANUAL only

(Requires a device list with at least one entry.)

- It issues an asterisk (*) prompt for a pathname at the application system terminal.
- It chooses a device depending upon the operator's response.
  - If a device name is entered, it loads from the device with that device name. It tries to load until the device becomes ready or until no more tries are allowed (as limited by the optional %RETRIES macro).
  - If no device name is entered before the carriage return, it looks for a ready device by searching through the list of devices (in the order in which they appear in the BS1.A86 file). The search continues until a ready device is found or until no more tries are allowed (as limited by the optional RETRIES macro). If it finds a ready device, it loads from that device.
- It chooses a file depending upon the operator's response to the prompt.
  - If a pathname is entered, it tries once to load the file with that pathname.
  - If no file name is entered, it tries once to load the file with the default pathname.

%AUTO only

(Requires a device list with at least one entry.)

- It looks for a ready device by searching through the list of devices (in the order in which they appear in the BS1.A86 file). The search continues until a ready device is found or until no more tries are allowed (as limited by the optional RETRIES macro).
- If it finds a ready device, it tries once to load the file with the default file name.

%AUTO and %CONSOLE

(Requires a device list with at least one entry.)

- It issues an asterisk (*) prompt for a pathname at the application system terminal.
- If the operator responds with a pathname that contains no device name, it looks for a ready device by searching through the list of devices (in the order in which they appear in the BS1.A86 file). The search continues until a ready device is found or until no more tries are allowed (as limited by the optional %RETRIES macro).

- If it finds a ready device or the operator responds with a pathname containing a device name, it tries once to load the file indicated by the operator's response.
  - If a pathname is entered, it tries to load the file with that pathname.
  - If only <cr> is entered, it tries to load the file with the default pathname.

In the foregoing macro descriptions, there are several references to an asterisk (*) prompt. If you have a monitor in PROM, with a pointer to its location in position 3 of the interrupt vector table, then when responding to this prompt you can use the Bootstrap Loader's Debug switch, which is described in Chapter 3.

The forms of the %CONSOLE, %MANUAL, and %AUTO macros are:

```
%CONSOLE

%MANUAL

%AUTO
```

%LOADFILE MACRO

The %LOADFILE macro causes the Bootstrap Loader to display at the console the pathname of the file it loads. It displays the pathname after loading the second stage and before loading the load file. The form of the %LOADFILE macro is:

```
%LOADFILE
```

%DEFAULTFILE MACRO

The %DEFAULTFILE macro specifies the hierarchical path of the default file. Its form is:

```
%DEFAULTFILE(pathname)
```

where pathname is the hierarchical path of the file enclosed in single quotes, as, for example, '/SYSTEM/TEST/RMX86'. If this macro is omitted, the pathname '/SYSTEM/RMX86' is assumed.

Do not omit this macro if you include the %LOADFILE macro.

%RETRIES MACRO

The %RETRIES macro, when included along with the %AUTO or %MANUAL macro, limits the number of times that the first stage goes through the device list in search of a ready device. If this macro is not included along with %AUTO or %MANUAL, and no device in the list is ready, then the search continues indefinitely. The form of the %RETRIES macro is:

```
%RETRIES(number)
```

where number, which must lie in the range 1 through 0FFFEH, is the maximum number of times the first stage checks each device for a ready condition.

%CICO MACRO

The %CICO macro specifies that console input and output are to be performed by standalone CI and CO routines, that is, routines that are not part of an iSDM 86, iSDM 286, or iSBC 957B monitor. If you include the %CICO macro, you must do some other things as well, depending upon whether the CI and CO routines you want to use are your own or those supplied by Intel.

If you use the Intel-supplied standalone CI and CO routines, you must do the following:

- Change the line in the BS1.CSD file that reads

  ```
  &    :f1:bcico.obj, &
  ```

  to

  ```
       :f1:bcico.obj, &
  ```

- Include exactly one instance of the %SERIAL_CHANNEL macro (described next) in the BS1.A86 file.

If you supply your own standalone CI and CO routines, you must do the following:

- Change the line in the BS1.CSD file that reads

  ```
  &    :f1:bcico.obj, &
  ```

  to

  ```
       :f1:mycico.obj, &
  ```

  where mycico.obj is an object file containing the CI and CO routines and a file called CINIT, which performs initialization functions required to prepare the console for input and output operations.

- Include no instances of the %SERIAL_CHANNEL macro.

The form of the %CICO macro is:

```
%CICO
```

%SERIAL_CHANNEL MACRO

Your CPU board can communicate over a serial channel by means of either an 8251A USART, an 8274 Multi-Protocol Serial Controller, or an 82530 Serial Communications Controller. The %SERIAL_CHANNEL macro, which requires you to include the %CICO macro, allows you to specify which serial controller device your CPU board uses as well as information that defines the use characteristics of the device.

You can omit this macro if your system does not use a terminal during bootstrap loading, if your supply your own CI and CO routines, or if you system use the iSDM 86, iSDM 286, or iSBC 957B monitor. Otherwise, include one instance of it in your BS1.A86 file for the serial controller device that supports the terminal your system uses for bootstrap loading. Including multiple %SERIAL_CHANNEL macros causes an assembly error when the BS1.CSD file runs.

The format of the %SERIAL_CHANNEL macro is as follows:

```
%SERIAL_CHANNEL (serial_type, serial_base_port, serial_port_delta,
                 counter_type, counter_base_port, counter_port_delta,
                 baud_counter, count, flags)
```

where:

serial_type — The serial controller device you are using. The valid values are 8251A, 8274, and 82530.

serial_base_port — The 16-bit address of the base port used by the device. This port varies according to the type of the device and, if applicable, the channel used on the device, as follows:

8251A	Data Register Port
8274 Channel A	Channel A Data Register Port
8274 Channel B	Channel B Data Register Port
82530 Channel A	Channel A Command Register Port
82530 Channel B	Channel B Command Register Port

serial_port_delta — The number of bytes between consecutive ports used by the serial device.

counter_type — The type of device containing the timer your CPU board uses to generate a baud rate for the serial device defined by this macro. The valid values are 8253, 8254, 80130, 80186, 82530, and NONE. Specifying NONE implies that the baud rate timer is automatically initialized and the Bootstrap Loader does not have to perform this function.

```
;
; *-*-*   BS1.CSD   *-*-*
;
;   Generate the iAPX 86, 88 Bootstrap Loader V5.0 first stage.
;
;   Invocation:  submit bs1(first stage location, second stage location)
;
run
;
asm86 :f1:bs1.a86   macro(50) object(:f1:bs1.obj)   print(:f1:bs1.lst)
asm86 :f1:bserr.a86 macro(50) object(:f1:bserr.obj) print(:f1:bserr.lst)
asm86 :f1:b204.a86  macro(50) object(:f1:b204.obj)  print(:f1:b204.lst)
asm86 :f1:b206.a86  macro(50) object(:f1:b206.obj)  print(:f1:b206.lst)
asm86 :f1:b208.a86  macro(50) object(:f1:b208.obj)  print(:f1:b208.lst)
asm86 :f1:b215.a86  macro(50) object(:f1:b215.obj)  print(:f1:b215.lst)
asm86 :f1:b218a.a86 macro(50) object(:f1:b218.obj)  print(:f1:b218.lst)
asm86 :f1:b251.a86  macro(50) object(:f1:b251.obj)  print(:f1:b251.lst)
asm86 :f1:b254.a86  macro(50) object(:f1:b254.obj)  print(:f1:b254.lst)
asm86 :f1:bsasi.a86 macro(50) object(:f1:bsasi.obj) print(:f1:bsasi.lst)
asm86 :f1:bscsi.a86 macro(50) object(:f1:bscsi.obj) print(:f1:bscsi.lst)
;

link86
    :f1:bs1.obj,                        &
    :f1:bserr.obj,                      &
&   :f1:bcico.obj,                      & ;for standalone serial channel
                                        & ;support
    :f1:b204.obj,                       &
    :f1:b206.obj,                       &
    :f1:b208.obj,                       &
    :f1:b215.obj,                       &
    :f1:b218.obj,                       &
    :f1:b251.obj,                       &
    :f1:b254.obj,                       &
    :f1:bsasi.obj,                      &
    :f1:bscsi.obj,                      &
    :f1:bs1.lib                         &
    to :f1:bs1.lnk print(:f1:bs1.mp1) &
    nopublics except(firststage,boot_186,bootstrap_entry)
;
loc86 :f1:bs1.lnk                                       &
    addresses(classes(code(%0),stack(%1)))              &
    order(classes(code,code_error,stack,data,boot))     &
    noinitcode                                          &
    start(firststage)                                   &
 &  ; change above line to start(boot_186) if iAPX_186_INIT is invoked  &
    segsize(boot(1800H))                    &
    map print(:f1:bs1.mp2)                  &
    ; Add "bootstrap" to loc86 when locating the first stage in ROM
```

Figure 2-2. First Stage Configuration File BS1.CSD

```
;
exit
;
;   Bootstrap Loader first stage generation complete.
;
```

Figure 2-2. First Stage Configuration File BS1.CSD (continued)

counter_base_port — The 16-bit address of the base port used by the baud rate timer. This port varies according to the type of the device and, if applicable, the channel used on the device, as follows:

8253	Counter 0 Count Register Port
8254	Counter 0 Count Register Port
80130	ICW1 Register Port
80186	Use 0FF00H on all Intel boards
82530 Channel A	Channel A Command Register Port
82530 Channel B	Channel B Command Register Port

counter_port_delta — The number of bytes between consecutive ports used by the timer.

baud_counter — The baud rate-generating counter on the timer. The devices and the counters you can specify for them are as follows:

8253	0, 1, and 2
8254	0, 1, and 2
80130	2
80186	0, 1
82530	0

count — A value that, when loaded into the timer register, generates the desired baud rate. The method of calculating this value is described in the paragraphs following these parameter definitions.

flags — A value that, when present, specifies which channel of an 82530 Serial Communications Controller will serve as your serial controller. If you give any value except 82530 for the serial_type parameter, omit this parameter; that is, write the macro as if the count parameter is the last parameter. If you give 82530 as the value of the serial_type parameter, specify A (for Channel A) or B (for Channel B) for this parameter.

To derive the correct value for the count parameter, you must perform a short series of computations. The starting values for these computations are the desired baud rate and the clock input frequency to the timer.

The first computation yields a temporary value and depends upon the timer used, as follows:

```
temporary_value = (clock frequency in Hertz)/(baud rate x 16)
```

if the timer is an 8253, 8254, 80130, or 80186, but

```
temporary_value = ((clock frequency in Hertz)/(baud rate x 2)) - 2
```

if the timer is an 82530.

The second computation yields the fractional part of the temporary value, as follows:

```
fraction = temporary_value - INT (temporary_value)
```

where the INT function gives the integer portion of temporary_value.

The third and fourth computations yield the desired count value and another value, called error_fraction. The error_fraction value is then used to determine whether the calculated count value is feasible, given the clock frequency specified in the first computation. These computations, which are performed according to the size of the value of "fraction" from the second computation, are as follows:

```
count = INT (result) + 1
error_fraction = 1 - fraction
```

if the value of "fraction" is greater than or equal to .5, but

```
count = INT (result)
error_fraction = fraction
```

if the value of "fraction" is less than .5.

The fifth and final computation, which yields the percentage of error that occurs when the given clock frequency is used to generate the given baud rate, is as follows:

```
% error = (error_fraction / count) x 100
```

If the % error value is less than 3, then the calculated count value is appropriate and will lead to the desired baud rate being generated by the specified clock frequency. However, if the % error value is 3 or greater, you must do either or both of the following two things:

- Provide a higher clock frequency
- Select a lower baud rate

After choosing one or both of these options, go through the series of computations again so as to get a new value of "count" and to see whether the revised value of "% error" is less than 3.

Continue this process -- raise the clock frequency and/or lower the baud rate, then do the computations -- until you finally get a "% error" value lower than 3.

The % SERIAL_CHANNEL macro can generate the following error messages:

```
ERROR - invalid port delta for the Serial Device
ERROR - <ser_type> is an invalid Serial Port type
ERROR - Invalid port delta for the Baud Rate Timer
ERROR - 8253/4 Baud Rate Counter is not 0, 1, or 2
ERROR - 2 is the only valid 80130 Baud Rate Timer
ERROR - 80186 counter counter_type is not a valid baud rate counter
ERROR - <counter_type> is an invalid Baud Rate Timer type
ERROR - Counter 0 is the only valid 82530 baud rate counter
ERROR - 82530 channel must be specified as A or B only
ERROR - Baud Rate Count must be greater than 1
```

%DEVICE MACRO

The %DEVICE macro defines a device unit from which your application system can be bootstrap loaded. If the BS1.A86 file contains multiple %DEVICE macros, their order in the file is the order in which the first stage searches for a ready device unit. Recall that multiple %DEVICE macros may be included only if the %AUTO or %MANUAL macro is included. (Otherwise, there is an assembly error when the BS1.CSD file runs.) The form of the %DEVICE macro is:

```
%DEVICE(name, unit, device$init, device$read)
```

where:

name	The physical name of the device, not enclosed in quotes or between colons. The first stage passes the physical name to the second stage, which, in turn, passes it to the load file. If the Automatic Boot Device Recognition (see Appendix A) capability is configured into the load file, then the physical names in the %DEVICE macro invocations must match the device unit names in the load file. Otherwise, the load file will not initialize properly and could "hang."
unit	The number of this unit on this device.

device$init	The name of the device$init procedure of the device driver the first stage will call for this device unit. If you are using an Intel-supplied driver, specify the procedure name as shown in Figure 2-1. (You may omit the "gen" suffix; see the discussion of this topic earlier in this chapter.) If you are supplying your own driver, which you have written in accordance with the instructions in Chapter 4, use the name of the initialization procedure.
device$read	The name of the device$read procedure of the device driver the first stage will call for this device unit. If you are using an Intel-supplied driver, specify the procedure name as shown in Figure 2-1. (You may omit the "gen" suffix; see the discussion of this topic earlier in this chapter.) If you are supplying your own driver, which you have written in accordance with the instructions in Chapter 4, use the name of the read procedure.

%END MACRO

The %END macro is required at the end of the BS1.A86 and BSERR.A86 assembly language source files. Its form is:

```
%END
```

BSERR.A86 CONFIGURATION FILE

The BSERR.A86 file, shown in Figure 2-3, defines what the first stage of the Bootstrap Loader does if it cannot load the load file.

```
 name bserr

$include(:f1:bserr.inc)

;console
%text
%list

%again
;int3
;halt

%end
```

Figure 2-3. First Stage Configuration File BSERR.A86

The BSERR.A86 file consists of an INCLUDE statement and several macros. The BSERR.INC file in the INCLUDE statement contains the definitions of the macros in the BSERR.A86 file.

The following sections describe the functions of the macros in the BSERR.A86 file. For each macro, if a percent sign (%) precedes the name, then the macro is included (invoked). If a semicolon (;) precedes the name, then the macro is treated as a comment and is not included.

The first three macros, %CONSOLE, %TEXT, and %LIST, determine what the Bootstrap Loader displays at the console whenever a bootstrap loading error occurs. The other three macros, %AGAIN, %INT3, and %HALT, determine what recovery steps, if any, the Bootstrap Loader takes whenever a bootstrap loading error occurs. Only one of the latter three macros can be included in the BSERR.A86 file.

%CONSOLE MACRO

The %CONSOLE macro causes the Bootstrap Loader to display a brief message at the console whenever a bootstrap loading error occurs. This message indicates the nature of the error. The messages are given in Chapter 3. The form of the %CONSOLE macro is:

```
%CONSOLE
```

This %CONSOLE macro is completely unrelated to the %CONSOLE macro in the BS1.A86 file. Be careful not to confuse them with each other.

%TEXT MACRO

The %TEXT macro resembles the %CONSOLE macro in that it causes the Bootstrap Loader to display a message at the console whenever a bootstrap loading error occurs. The advantage of the %TEXT macro is that its messages are longer and more descriptive. The disadvantage of the %TEXT macro is that it generates more code and therefore makes the assembled BSERR.OBJ file larger. The %TEXT macro has the form:

```
%TEXT
```

If you include the %TEXT macro, the %CONSOLE macro is automatically included, as well.

%LIST MACRO

The %LIST macro causes the Bootstrap Loader to display a list of the ready device units whenever the operator enters an invalid device unit name. You may include this macro only if you include the %MANUAL macro in the BS1.A86 file, described earlier in this chapter. The %LIST macro has the form:

```
%LIST
```

If you include the %LIST macro, the %CONSOLE and %TEXT macros are automatically included, as well.

%AGAIN MACRO

The %AGAIN macro causes the bootstrap loading sequence to return to the beginning of the first stage whenever a bootstrap loading error occurs. It is a good idea to include this macro if you include the %CONSOLE macro in the BSERR.A86 file, either directly or by including the %TEXT or %LIST macro. The form of the %AGAIN macro is:

```
%AGAIN
```

Exactly one of the %AGAIN, %INT3, and %HALT macros must be included, or there will be an assembly error when the BS1.CSD file runs.

%INT3 MACRO

The %INT3 macro causes the Bootstrap Loader to execute an INT 3 (software interrupt) instruction whenever a bootstrap loading error occurs. If you are using the iSDM 86 or iSDM 286 System Debug Monitor or the iSBC 957B package, then the INT 3 instruction passes control to the monitor. Otherwise, the INT 3 instruction will not produce the desired results unless you have placed the appropriate address in position 3 of the interrupt vector table. The form of the %INT3 macro is:

```
%INT3
```

Exactly one of the %AGAIN, %INT3, and %HALT macros must be included, or there will be an assembly error when the BS1.CSD file runs.

The %INT3 macro and the %HALT macro (described next) are reasonable choices if none of the %CONSOLE, %TEXT, and %LIST macros are included in the BSERR.A86 file.

%HALT MACRO

The %HALT macro causes the Bootstrap Loader to execute a halt instruction whenever a bootstrap loading error occurs. The form of the %HALT macro is:

```
%HALT
```

Exactly one of the %AGAIN, %INT3, and %HALT macros must be included, or there will be an assembly error when the BS1.CSD file runs.

The %HALT macro and the %INT3 macro are reasonable choices if none of the %CONSOLE, %TEXT, and %LIST macros are included in the BSERR.A86 file.

INTEL-SUPPLIED DEVICE DRIVER CONFIGURATION FILES

There is a separate configuration file for each device driver provided with the Bootstrap Loader. These files are named B204.A86, B206.A86, B208.A86, B215.A86, B218.A86, B251.A86, B254.A86, BSASI.A86, and BSCSI.A86. Each consists of an include statement and a macro call. The include statement always has the form:

```
$include(:f1:bxxx.inc)
```

where:

xxx	Either 204, 206, 208, 215, 218, 251, 254, SASI, or SCSI, depending upon the device driver.

The macro call has a form that depends upon the device driver. This form is discussed in the following sections. The default parameter values for the macros in these sections are compatible with the default parameter values of the iRMX 86 Interactive Configuration Utility.

%B204 MACRO

The %B204 macro has the form:

```
%B204(io_base, sector_size, track_size)
```

where:

io_base	I/O port address selected (jumpered) on the iSBC 204 controller board.
sector_size	Sector size for the device, in bytes.
track_size	Track size for the device, in bytes.

The default form of this macro in the B204.A86 file is:

```
%B204(0A0H, 128, 26)
```

%B206 MACRO

The %B206 macro has the form:

```
%B206(io_base)
```

where:

io_base	I/O port address selected (jumpered) on the iSBC 206 controller board.

The default form of this macro in the B206.A86 file is:

```
%B206(068H)
```

%B208 MACRO

The %B208 macro has the form:

```
%B208(io_base)
```

where:

io_base	I/O port address selected (jumpered) on the iSBC 208 controller board.

The default form of this macro in the B208.A86 file is:

```
%B208(180H)
```

%B215 AND %B220 MACROS

The B215.A86 file contains two macros, of which you can use only one. They are the %B215 and the %B220 macros. Both of them have the form:

```
%Bxxx(wakeup, cylinders, fixed_heads, removable_heads, sectors,
     dev_gran, alternates)
```

where:

xxx	Either 215 or 220.
wakeup	Base address of the wakeup port

cylinders	Number of cylinders on the disk drive or drives. (Note that, if your %DEVICE macro for 215 or 220 devices in the BS1.A86 file has deviceunit215 (rather than deviceunit215gen) as its third parameter, then all iSBC 215 or iSBC 220 drives used by the Bootstrap Loader must have the same characteristics. That is, they must have the same number of cylinders per platter, fixed heads, removable heads, sectors per track, bytes per sector, and alternate cylinders. However, if the %DEVICE macro specifies deviceunit215gen, these restrictions do not apply and these values are not used.)
fixed_heads	Number of heads on fixed platters.
removable_heads	Number of heads on removable platters.
sectors	Number of sectors per track.
dev_gran	Number of bytes per sector.
alternates	Number of cylinders set aside as backups for cylinders having imperfections.

In the B215.A86 file, the default form of the %B215 macro is:

```
%B215(100H, 256, 2, 0, 9, 1024, 5)
```

and the default form of the %B220 macro is:

```
%B220(100H, 256, 2, 0, 9, 1024, 5)
```

%B218 MACRO

The %B218 macro has the form:

```
%B218(base_port_address, motor_flag)
```

where:

base_port_address	The base port address of this device unit, as selected on the iSBX 218A controller board.
motor_flag	A value indicating whether the motor of a 5 1/4" flexible diskette drive should be turned off after bootstrap loading. Specify Yes, which slows bootstrap loading, only if this device is not the system device. For Yes, specify 0FFH, and for No, specify 0.

The default form of this macro in the B218.A86 file is:

```
%B218(80H, 00H)
```

%B251 MACRO

The %B251 macro has the form:

```
%B251(io_base, dev_gran)
```

where:

io_base	I/O port address selected (jumpered) on the iSBX 251 controller board.
dev_gran	Page size, in bytes.

The default form of this macro in the B251.A86 file is:

```
%B251(80H, 64)
```

%B254 MACRO

The %B254 macro has the form:

```
%B254(io_base, dev_gran, num_boards, board_size)
```

where:

io_base	I/O port address selected (jumpered) on the iSBC 254 controller board.
dev_gran	Page size, in bytes.
num_boards	Number of boards grouped in a single device unit.
board_size	Number of pages in one iSBC 254 board.

The default form of this macro in the B254.A86 file is:

```
%B254(0880H, 256, 8, 2048)
```

%BSASI MACRO

The %BSASI macro has the form:

```
%BSASI(a_port, b_port, c_port, control-port, init_command,
       init_byte_count, init_bytes)
```

where:

a_port	The address of Port A of the 8255 Programmable Peripheral Interface (PPI) used by this SASI driver.

b_port	The address of Port B of the 8255 PPI used by this SASI driver.
c_port	The address of Port C of the 8255 PPI used by this SASI driver.
control-port	The address of the control word register of the 8255 PPI used by this SASI driver.
init_command	The command that initializes the controller. (If the controller does not require initialization, use the SCSI driver instead of the SASI driver.) For the initialization command, look in the owner's manual for the controller. It might be labelled there either as a "command" or as an "opcode."
init_byte_ count	The number of initialization bytes that follow this parameter.
init_bytes	A list of initialization bytes, separated by commas, that define the characteristics of the drive. These values depend upon both the type of the controller and the type of the drive. The values can be found in the owner's manual for the controller.

NOTE

The BSASI macro is different than the other macros in that, if there are multiple occurrences of it in the BSASI.A86 file, then the corresponding devices must be either identical or completely compatible. That is, the devices must have identical specifications and can differ only in their unit number.

The default form of this macro in the BSASI.A86 file is:

```
%BSASI(C8H, CAH, CCH, CEH, 0CH, 8, 01H, 32H, 6, 0, 0FFH, 0,
       0FFH, 0BH)
```

This default macro definition is for a Xebec S1410 5 1/4-inch Winchester disk controller and a Computer Memories, Inc. CMI-5419 19-megabyte Winchester disk drive.

%BSCSI MACRO

The %BSCSI macro has the form:

```
%BSCSI(a_port, b_port, c_port, control_port, host_id, arbitrate)
```

where:

a_port	The address of Port A of the 8255 Programmable Peripheral Interface (PPI) used by this SCSI driver.
b_port	The address of Port B of the 8255 PPI used by this SCSI driver.
c_port	The address of Port C of the 8255 PPI used by this SCSI driver.
control_port	The address of the control word register of the 8255 PPI used by this SCSI driver.
host-id	The ID of the host computer on the SCSI bus.
arbitrate	A flag indicating whether bus arbitration is supported. Set to 0, which signifies that bus arbitration is not supported.

The SCSI driver can be used to bootstrap load from any Winchester device on the SCSI bus.

The default form of this macro in the BSCSI.A86 file is:

```
%BSCSI(C8H, CAH, CCH, CEH, 80H, 00H)
```

The default macro definition is for an iSBC 186/03 board.

USER-SUPPLIED DRIVERS

If you want to bootstrap load your system from a device other than one controlled by an iSBC 204, 206, 208, 215, 218A, 220, 251, or 254 board, or one that interfaces with the SASI or SCSI driver, you must write your own init and read device driver procedures. In addition, you must specify their procedure names in the %DEVICE macro in the BS1.A86 file, and you must assemble them and link them to the rest of the Bootstrap Loader object files and libraries. Chapter 4 describes how to write the device driver procedures.

GENERATING THE BOOTSTRAP LOADER SYSTEM

To generate the bootstrap loading system, enter the command

```
SUBMIT BS1.CSD(first_stage_address, second_stage_address)
```

where the parameters specify the low-memory addresses of the stages.

The size of the first stage area depends upon the device drivers in the first stage. If you use Intel-supplied drivers, the size is always less than 8K bytes, even with all of the drivers configured in at once.

Recall that the first stage can be either in PROM or in RAM. The second stage area, which includes the code of the second stage and the data areas for both stages, consists of slightly less than 8K contiguous bytes. The second stage always resides in RAM.

***

CHAPTER 3
USING THE BOOTSTRAP LOADER

This chapter describes how to set up and invoke the Bootstrap Loader and what to do if it fails to perform as expected.

PREPARING TO USE THE BOOTSTRAP LOADER

There are four ways to bootstrap load your application. The key to each of these methods is the first stage of the Bootstrap Loader: where you put it and how you invoke it. The four methods are as follows:

- Place the first stage, configured for standalone operation, in PROM. In this case, bootstrap loading commences -- that is, the first stage begins to run -- when you turn on the system hardware or press the RESET button.

- Place the first stage in secondary storage, and then load it by means of external commands. Doing this requires you to use the iSDM 86, iSDM 286, or iSBC 957B monitor, or an ICE in-circuit emulator, first to load the first stage into RAM and then to invoke the first stage.

- Augment the iSDM 86, iSDM 286, or iSBC 957B monitor by reconfiguring the first stage of the Bootstrap Loader to include the device driver(s) needed for bootstrap loading, and program new PROMs with the combination of the monitor and the first stage of the Bootstrap Loader. With this method, you initiate bootstrap loading by means of the B command of the monitor.

- Place the first stage in secondary storage, and then load it programmatically.

In the first method, you must add the BOOTSTRAP control to the LOC86 command used in the BS1.CSD file, as indicated in the last comment in that file. Otherwise, each of the first two methods is straightforward and therefore is not described in this manual.

The instructions for using the third method lie outside of this chapter. To use this method with the iSDM 86 monitor, follow the instructions given in the iSDM 86 SYSTEM DEBUG MONITOR REFERENCE MANUAL. In addition, Appendix B of this manual has a short section that describes a required modification of a file that is listed in the iSDM 86 manual.

To use the third method with the iSDM 286 monitor, refer to Appendix B of this manual, as well as the iSDM 286 SYSTEM DEBUG MONITOR REFERENCE MANUAL.

To use the third method with the iSBC 957B monitor, follow the directions given in the USER'S GUIDE FOR THE iSBC 957B iAPX 86, 88 INTERFACE AND EXECUTION PACKAGE.

Note that error handling, as described later in this chapter, does not take place when you use the third method of bootstrap loading with the iSBC 957B package.

The rest of this section gives instructions for using the fourth method.

Although bootstrap loading is performed usually in response to an external event, it can be initiated by an executing program by means of a call to the PUBLIC symbol BOOTSTRAP_ENTRY. To prepare for such a call, do the following:

- Place a call to BOOTSTRAP_ENTRY in the code of the invoking program, and define BOOTSTRAP_ENTRY as an EXTERNAL symbol there. The form of the call is:

  ```
  CALL BOOTSTRAP_ENTRY(@filename)
  ```

  where:

filename	An ASCII string containing either the pathname of the load file followed by a carriage return, or only a carriage return. If the string contains only a carriage return, then the default file, as defined by the %DEFAULTFILE macro in the BS1.A86 configuration file, is loaded. Otherwise, the file whose pathname is contained in the string is loaded.

  The call must follow the PL/M-86 LARGE model of segmentation. (Even though this is a call, rather than a jump, it does not return.)

- Link the calling program to a version of the first stage of the Bootstrap Loader. You can do this by following the BS1.CSD file as a model, with the following changes:

  - Place the calling program in the link sequence.
  - If appropriate, "comment out" the locate sequence.

OPERATOR'S ROLE IN BOOTSTRAP LOADING

Depending upon the method used for Bootstrap Loading, an operator might be required to enter the name of the bootstrap device, the name of the load file, or both names. (Another possibility, depending upon how the Bootstrap Loader is configured, is that the operator can enter neither device name nor file name. This section refers to what the operator enters as the specification of the load file.) Along with the specification of the load file, the operator can specify, by means of the

Debug switch, that control should pass to the monitor after loading has been completed. These are the subjects of this section.

SPECIFYING THE LOAD FILE

There are two times at which an operator can enter the specification of a load file. One time is when one of the monitors has issued a period (.) prompt. In that case, the operator can enter the monitor's B (bootstrap) command, followed by the specification. The other time is when the first stage of the Bootstrap Loader has issued an asterisk (*) prompt at the terminal. When this prompt appears on the screen, the first stage waits for an operator to enter the specification of the load file.

Once the period or asterisk prompt has been issued, the specification that the operator enters depends three things. They are:

- Which file is the load file.
- Which device unit contains the load file.
- Which of the %CONSOLE, %MANUAL, and %AUTO macros were in the BS1.A86 file when the present configuration of the Bootstrap Loader was defined.

For a discussion of the possible operator actions and their effects, see the description of the %CONSOLE, %MANUAL, and %AUTO macros in Chapter 2.

THE DEBUG SWITCH

Along with the specification of the load file, the operator can include the Bootstrap Loader's Debug (D) switch. When specified, the Debug switch instructs the second stage of the Bootstrap Loader to do the following immediately after loading has been completed:

- Set a breakpoint at the first instruction to be executed by the system.
- Pass control to the monitor, which displays a "*BREAK* at xxxx:xxxx" (iSDM 86 and iSBC 957B monitors) or an "Interrupt 3 at <xxxx:xxxx>" (iSDM 286 monitor) message at the terminal, issues its prompt, and waits for a command from the terminal. (To start up the loaded system, enter "G<cr>".)

One advantage of the Debug switch is that the monitor's message tells you that the loading process is successful. When a system fails, it is sometimes difficult to determine whether the bootstrap loading was unsuccessful or whether the system loaded successfully and then failed during initialization. The presence or absence of the message makes this clear when you use the Debug switch.

The Debug switch also allows you to alter the contents of specific memory locations before your system begins to run.

To use the Debug switch with a monitor's period (.) prompt, follow the B command in the command line by the letter "D", which, in turn, can be followed by the pathname of the load file. For example, any of the following command lines invokes the Bootstrap Loader with the Debug switch:

```
.bd

.b d

.bd /system/rmx86

.b  d  :w0:system/rmx86
```

Similarly, the way to use the Debug switch with the first stage's asterisk (*) prompt is to precede the load file specification with the letter "D." Examples of this are:

```
*d

*  d

*d /system/rmx86

* d  :w0:system/rmx86
```

The only restriction concerning the use of spaces in these command lines is that there must be a space between the letter "D" and the pathname of the load file.

Note that the Debug switch is available only in second stages residing on secondary storage volumes that have been formatted using the Release 6 (or later) versions of the iRMX 86 Format command. If you use the Debug switch with older second stages, the letter "D" is ignored, and the loadfile is loaded and run without the effects the Debug switch has when used with Release 6-compatible volumes.

ANALYZING BOOTSTRAP LOADING FAILURES

The Bootstrap Loader has the ability to display messages on the screen when bootstrap loading is not successful. As you saw in Chapter 2, the %CONSOLE, %TEXT, and %LIST macros in the BSERR.A86 file determine whether such messages are to be displayed, how detailed the messages are, and under what circumstances they are to be displayed. This section describes two ways of analyzing bootstrap loading errors: first, when messages are being displayed; and second, when there are no messages.

After responding to an error by pushing a word onto the stack and optionally displaying a message, the Bootstrap Loader either tries again, passes control to a monitor, or halts, depending upon whether your BSERR.A86 file contains a %AGAIN, %INT3, or %HALT macro.

WITH DISPLAYED ERROR MESSAGES

If your BSERR.A86 file contains the %CONSOLE, %TEXT, or %LIST macro, then the Bootstrap Loader displays an error message at your terminal whenever a failure occurs in the bootstrap loading process. The message consists of one or two parts. The first part, which is always displayed, is a numerical error code. The second part, which is displayed only if the %TEXT or %LIST macro was included, is a short verbal description of the error.

Each numerical error code has two digits. The first digit indicates, if possible, the stage of the bootstrap loading process where the error occurred. The second digit distinguishes among types of errors that can occur in a particular stage. There are three possible values for the first digit:

First Digit	Stage
0	Can't tell
1	First
2	Second

The error codes, their abbreviated display messages, and their causes and meanings are as follows.

Error Code: 01
Description: I/O error

An I/O error occurred at some undetermined time during the bootstrap loading process, but it is not clear when the error occurred. To help you further diagnose this problem if the %CONSOLE macro is included, the Bootstrap Loader places a code in the high-order byte of the word it pushes onto the stack. This byte identifies the driver for the device that produced the error, as follows:

Code	Driver
04H	204
06H	206
08H	208
15H	215 (with or without 218A) or 220
18H	218A on CPU board
51H	251
54H	254
0E0H	SCSI
0E1H	SASI
other (in range A0H-DFH)	driver for your custom device

Note that this device code is overwritten during the printing of the description in case the %TEXT or %LIST macro has been included.

The last entry in the list of device codes assumes that you have written a device driver for your device and have identified the driver by some code in the indicated range -- other values are reserved for Intel drivers. For information about how to incorporate this code into the driver, see Chapter 4.

Error Code: 11
Description: Device not ready.

The specific device designated for bootstrap loading is not ready. This error occurs only when your BSERR.A86 file does not contain the %AUTO macro. Therefore, either the operator has specified a particular device or there is only one device in the Bootstrap Loader's device list, and the device is not ready.

Error Code: 12
Description: Device does not exist. (If BSERR.A86 contains the %LIST macro, the display then shows the list of known devices.)

The device name entered at the console does not have an entry in the Bootstrap Loader's device list. This error occurs only when your BSERR.A86 file contains the %MANUAL macro and you enter a device name, but the device name you entered is not known to the Bootstrap Loader. After displaying the message, the Bootstrap Loader displays the names of the devices in its device list.

Error Code: 13
Description: No device ready.

None of the devices in the Bootstrap Loader's device list are ready. This error occurs only when your BSERR.A86 file contains the %AUTO or %MANUAL macro and you do not enter a device name at the console.

Error Code: 21
Description: File not found.

The Bootstrap loader was not able to find the indicated file on the designated bootstrap device. This is the default file if no pathname was entered at the console. Otherwise, it is the file whose pathname was entered.

Error Code: 22
Description: Bad checksum.

While trying to load the load file, the Bootstrap Loader encountered a checksum error. Each file consists of several records, and associated with each record is a checksum value that specifies the numerical sum (ignoring overflows) of the bytes in the record. When the Bootstrap Loader loads a file, it computes a checksum value for each record and compares that value to the recorded checksum value. If there is a discrepancy for any record in the file, it usually means that one or more bytes of the file have been corrupted, so the Bootstrap Loader returns this message instead of continuing the loading process.

Error Code: 23
Description: Premature end of file.

The Bootstrap Loader did not find the required end-of-file records at the end of the load file.

Error Code: 24
Description: No start address found in input file.

The Bootstrap Loader successfully loaded the load file but was unable to transfer control to the file because when it got to the end of the file it still had not found initial CS and IP values.

WITHOUT DISPLAYED ERROR MESSAGES

In most cases, by observing the behavior of the Bootstrap Loader when it fails to load the application successfully, you can determine the cause of the failure and take steps to correct it. Table 3-1 shows the correlation between the behavior of the Bootstrap Loader and most of the possible causes of its failure. The table assumes that the Bootstrap Loader is set up to halt if it detects an error. Before halting, the Bootstrap Loader places the error code into the CX register.

Another possible cause of failure of the Bootstrap Loader, the effects of which are completely unpredictable, is that the device controller block (as determined by the device's wake-up address) can be corrupted. To avoid this kind of failure, see that neither the second stage nor the load file overlaps the device controller block for the device.

Table 3-1. Postmortem Analysis Of Bootstrap Loader Failure

Behavior Of Loader	Possible Causes
Bootstrap loading fails in the first stage.	The indicated device is not ready or is not known to the Bootstrap Loader. An I/O error occurred during the first stage operation.
Bootstrap loading fails in the second stage.	The indicated file is not on the device. The file had no end-of-file record or no start address. The file contained a checksum error. An I/O error occurred during the second stage operation.
Bootstrap Loader enters second stage, but does not halt or pass control to the loaded file.	The Bootstrap Loader is attempting to load the system on top of the second stage. The Bootstrap Loader is attempting to load the system into nonexistent memory.

***

CHAPTER 4
WRITING A DRIVER FOR A BOOTSTRAP LOADING DEVICE

The iRMX 86 Bootstrap Loader can be configured to run with many kinds of devices. If you plan to use one of the devices for which Intel supplies a device driver, you may skip this chapter.

If you want to use the Bootstrap Loader with a device other than those supported by Intel, you must write your own device driver. The purpose of this section is to provide you with guidelines for writing a customized driver.

Two procedures must be included in every device driver for the Bootstrap Loader. The initialization procedure initializes the bootstrap device. The reading procedure loads information from the device into RAM.

The rest of this chapter refers to the two procedures as DEVICE$INIT, and DEVICE$READ. However, you can give them any names you want during configuration. You must specify each of their names in a %DEVICE macro in the BS1.A86 file.

Both device driver procedures must conform to the LARGE model of segmentation of the PL/M-86 programming language. This means that the procedures must be FAR (not NEAR) and all pointers must be 32 bits long.

You may write the procedures in assembly language, rather than in PL/M-86, but if you do, you must adhere to the interfacing and referencing conventions of the PL/M-86 LARGE model.

DEVICE$INIT PROCEDURE

The DEVICE$INIT procedure must present the following PL/M-86 interface to the Bootstrap Loader:

```
DEVICE$INIT: PROCEDURE (unit) WORD PUBLIC;
          DECLARE unit      WORD;
     .
     . (code)
     .
END DEVICE$INIT;
```

where:

unit	The device's unit number, as defined during Bootstrap Loader configuration.

The WORD value returned by the procedure must be the device granularity, in bytes, if the device is ready, or zero if the device is not ready.

The following outline shows the steps that the DEVICE$INIT procedure must perform to be compatible with the Bootstrap Loader:

1. Test to see if the device is present. If it is not, return the value zero.

2. Initialize the device for reading. This is a device-dependent operation. For guidance in initializing the device, refer to the hardware reference manual for the device.

3. Test to see if device initialization was successful. If it was not, return the value zero.

4. Obtain the device granularity. For some devices, only one granularity is possible, while for others several granularities are possible. This is a device-dependent issue that is explained in the hardware reference manual for your device.

5. Return the device granularity.

DEVICE$READ PROCEDURE

The DEVICE$READ procedure must present the following PL/M-86 interface to the Bootstrap Loader:

```
DEVICE$READ: PROCEDURE (unit, blk$num, buf$ptr) PUBLIC;
          DECLARE   unit          WORD,
                    blk$num       DWORD,
                    buf$ptr       POINTER;
     .
     . (code)
     .
END DEVICE$READ;
```

where:

unit	The device's unit number, as defined during configuration.
blk$num	A 32-bit number specifying the number of the block that the Bootstrap Loader wants the procedure to read.
buf$ptr	A 32-bit POINTER to the buffer that is to receive the information from the secondary storage device.

The DEVICE$READ procedure does not return a value to the caller.

The following outline shows the steps that the DEVICE$READ procedure must perform to be compatible with the Bootstrap Loader:

1. Read the block specified by the blk$num parameters from the bootstrap device specified by the unit parameter into the memory location specified by the buf$ptr parameter.

2. Check for I/O errors. If none occurred, return to the caller. Otherwise, combine the device code, if any, for the device with 01 (in the form <device code>01), push the resulting word value onto the stack, and call the BSERROR procedure. For example, if the device code is 0B3H, push B301H onto the stack. If there isn't a device code, use 00.

   In PL/M-86, adding the following statements will accomplish this:

```
DECLARE BSERROR EXTERNAL;
DECLARE IO_ERROR LITERALLY '0B301H';

CALL BSERROR(IO_ERROR);
```

   If you are calling the BSERROR procedure from assembly language, note that BSERROR follows the PL/M-86 LARGE model of segmentation; that is, declare BSERROR as:

```
extrn     bserror:far
```

***

APPENDIX A
AUTOMATIC BOOT DEVICE RECOGNITION

Automatic boot device recognition allows the iRMX 86 Operating System to recognize the device from which it was bootstrap loaded and to assign a logical name (normally :SD:) to represent that device.

If you use this feature, you can configure versions of the Operating System that are device independent, that is, versions you can load and run from any device your system supports.

This section describes the automatic boot device recognition feature in detail. It consolidates the information found in the other iRMX 86 manuals and answers the following questions:

- How does automatic boot device recognition work?
- How do you configure a version of the Operating System that includes this feature?

HOW AUTOMATIC BOOT DEVICE RECOGNITION WORKS

The Nucleus, the Extended I/O System, and the second stage of the Bootstrap Loader combine to provide the automatic boot device recognition feature, as follows:

1. The second stage of the Bootstrap Loader, after loading the Operating System, places a pointer in the DI:SI register pair. This pointer points to a string containing the name of device from which the system was loaded. The name it uses is the one you (or whoever performed the configuration) supplied as a parameter in the %DEVICE macro when configuring the Bootstrap Loader.

2. The second stage sets the CX and DX registers to the value 1234H. This value signifies that the pointer contained in the DI:SI register pair is valid.

3. The root job checks CX and DX and then, if both contain 1234H, uses the pointer in DI:SI to obtain the device name. The root job sets a Boolean variable to indicate whether it found the name of the boot device.

4. The Nucleus checks the root job's Boolean variable and, if it is true (equal to 0FFH), places the device name in a segment and catalogs that segment in the root job's object directory under the name RQBOOTED.

5. The Extended I/O System looks up the name RQBOOTED and, if successful, obtains the device name from the segment cataloged there. If the name RQBOOTED is not cataloged in the root directory, the Extended I/O System uses a default device name that you must have specified during the configuration of the Extended I/O System (DPN prompt of the "EIOS" screen).

6. The Extended I/O System attaches the device as the system device, assigning it the logical name that you must have specified during the configuration of the Extended I/O System (DLN prompt on the "EIOS" screen).

HOW TO INCLUDE AUTOMATIC BOOT DEVICE RECOGNITION IN YOUR SYSTEM

This section describes the operations you must perform to include the automatic boot device recognition feature in your application. The operations include:

- The "EIOS" screen (see Figure A-1) contains several prompts that affect the automatic boot device recognition feature. They are: ABR, DLN, DPN, DFD, and DO.

  With the ABR prompt, you specify whether you want to include the automatic boot device recognition feature in your system. If you set ABR to "yes," the Extended I/O System automatically attaches the system device using the characteristics specified in the DLN, DFD, and DO prompts. You must not supply this information later in the "Logical Names" screen.

  If you set ABR to "no," the Extended I/O System does not attach a system device. In this case, the ICU does not display the DLN, DPN, DFD, and DO prompts.

```
     EIOS
     (ASC) All Sys Calls in EIOS                                  Req
---> (ABR) Automatic Boot Device Recognition [Yes/No]              Yes
---> (DLN) Default System Device Logical Name [1-12 characters]   SD
---> (DPN) Default System Device Physical Name [1-12 characters]  w0
---> (DFD) Default System Device File Driver [Phys/Str/Named]     Named
---> (DO)  Default System Device Owners ID [0-0FFFFH]             0000H
     (EBS) Internal Buffer Size [0-0FFFFh]                        0400H
     (DDS) Default IO Job Directory Size [5-0FF0h]                0032H
     (ITP) Internal EIOS Task's Priorities [0-FFH]                0083H
     (PMI) EIOS Pool Minimum [0-0FFFFH]                           0180H
     (PMA) EIOS Pool Maximum [0-0FFFFH]                           0180H
     (EIR) Extended I/O System in ROM [Yes/No]                    No
```

Figure A-1. EIOS Configuration Screen (ABR, DLN, DPN, DFD, and DO)

With the DLN prompt, you can specify the logical name for your system device. If you change this value from the default (SD), you must change all other references to the :SD: logical name to the new name you specify. The Extended I/O System creates the logical name you specify only if you set ABR to "yes."

With the DPN prompt, you specify the physical name of a device that you want to use as your system device in case the Extended I/O System cannot find the name RQBOOTED cataloged in the root object directory. This situation normally occurs when you load your system using a means other than the Bootstrap Loader. For example, if you transfer the Operating System to your target system via the iSBC 957B load package or iSDM 86 or iSDM 286 monitor, there is no bootstrap device. In this case, the Extended I/O System uses the device name specified in the DPN prompt as the system device.

With the DFD and DO prompts, you set other characteristics associated with the system device. For most cases, the defaults (DFD=Named and DO=0000H) are the preferred values.

- During configuration of the Basic I/O System, you must specify device-unit information for the devices you wish to support. One of the prompts on each "Device-Unit Information" screen (NAM) requires you to specify the name of the device-unit. Another parameter (UN) requires you to specify the unit number. (See Figure A-2 for an example of these prompts.) To enable the automatic boot device recognition feature to work correctly, assign device-unit names and unit numbers that match the device names and unit numbers assigned during the configuration of the Bootstrap Loader.

- You assign the Bootstrap Loader device names and unit numbers by including or modifying %DEVICE macros in the first-stage configuration file (BS1.A86) or in the iSBC 957B configuration file. With the ICU, you can define device-unit names and unit numbers other than those that are valid for the Bootstrap Loader. But each Bootstrap Loader device name must have a corresponding device-unit name, and the unit numbers must be the same.

Before you can use the automatic boot device recognition feature, you must format your system device using the Release 6 version of the FORMAT command. The iRMX 86 CONFIGURATION GUIDE describes how to set up your system device for use with Release 6.

HOW TO EXCLUDE AUTOMATIC BOOT DEVICE RECOGNITION

To configure a system that does not include the automatic boot device recognition feature, set the ABR prompt in the "EIOS" screen to "No" (see Figure A-1). This disables the automatic boot device recognition feature.

```
    Intel iSBC 215/iSBX 218 Device-Unit Information
--->(NAM) Device-Unit Name [1-13 chars]
    (PFD) Physical File Driver Required [Yes/No]                    Yes
    (NFD) Named File Driver Required [Yes/No]                       Yes
    (SDD) Single or Double Density Disks [Single/Double]            Single
    (SDS) Single or Double Sided Disks [Single/Double]              Single
    (EFI) 8 or 5 inch Disks [8/5]                                   8
    (GRA) Granularity [0-0FFFFH]                                    0080H
    (DSZ) Device Size [0-0FFFFFFFFH]                                0003E900H
--->(UN)  Unit Number on this Device [0-0FFH]                       0000H
    (UIN) Unit Info Name [1-17 Chars]
    (UDT) Update Timeout [0-0FFFFH]                                 0064H
    (NB)  Number of Buffers [nonrand = 0/rand = 1-0FFFFH]           0006H
    (FUP) Fixed Update [True/False]                                 True
    (MB)  Max Buffers [0-0FFH]                                      00FFH
```

Figure A-2. Device-Unit Information Screen (NAM and UN)

When you set ABR to "No", the ICU deletes the DLN, DPN, DFD, and DO prompts from the EIOS screen. Therefore, you must provide this information as input to the "Logical Names" screen. Figure A-3 shows an example of this screen after it has been filled in to include a logical name for the system device. The underlined information in Figure A-3 is the information you would supply if you set the ABR prompt in Figure A-1 to "No" and you want the system device to be a flexible diskette drive controlled by an iSBC 208 board.

```
    Logical Names
    Logical Name : logical_name,device_name,file_driver,owners-id
           [1-12 Chars, 1-14 Chars, Physical/Stream/Named, 0-0FFFFH]

    Logical Name : BB, BB, Physical, 0000H
    Logical Name : STREAM, STREAM, Stream, 0000H
--->Logical Name : SD, AF0, Named, 0000H
```

Figure A-3. Logical Names Screen

***

APPENDIX B
PROMMING THE BOOTSTRAP LOADER WITH A SYSTEM DEBUG MONITOR

In Chapter 3, it was mentioned that one of the ways in which you can prepare to use the Bootstrap Loader is to combine it with one of the Intel monitor packages and burn the combined code into PROM. This appendix supplies information that is not contained in the reference manuals for the iSDM 86 and iSDM 286 monitors.

COMBINING WITH THE iSDM 86 SYSTEM DEBUG MONITOR

The iSDM 86 SYSTEM DEBUG MONITOR REFERENCE MANUAL correctly describes the procedure for combining the iRMX 86 Bootstrap Loader with the iSDM 86 monitor. However, if you are going to combine the Release 6 version of the Bootstrap Loader with the iSDM 86 monitor, you must first modify one of the files described in that manual. This file, called SDMGNB.CSD, must be changed to the read as is shown in Figure B-1. In the figure, the lines that are different from the listing in the iSDM 86 manual are marked with comments on the right-hand side of the figure.

COMBINING WITH THE iSDM 286 SYSTEM DEBUG MONITOR

The iSDM 286 SYSTEM DEBUG MONITOR REFERENCE MANUAL does not describe the procedure for combining the iRMX 86 Bootstrap Loader with the iSDM 286 monitor. This section gives the instructions required to burn the first stage and the iSDM 286 monitor into two 2764 EPROMs. You can modify this example to suit your own purposes, or you can follow it exactly. The step-by-step procedure is as follows:

Enter the name of the (version 1.3 or later) software used with the iUPP Universal Prom Programmer:

```
:f0:ipps
```

Specify that the PROMs are 2764 EPROMs:

```
type 2764
```

Initialize the file type to be loaded:

```
initialize 86
```

This says that the load file is an 8086 Object Module Format file.

```
run
;
asm86 :f1:bsl.a86 macro(90)                    ;changed line
asm86 :f1:bserr.a86 macro(50)
;asm86 :f1:b204.a86 macro(50)
;asm86 :f1:b206.a86 macro(50)
;asm86 :f1:b208.a86 macro(50)
;asm86 :f1:b215.a86 macro(50)
;asm86 :f1:b218a.a86 macro(50)                 ;new line
;asm86 :f1:b251.a86 macro(50)                  ;new line
;asm86 :f1:b254.a86 macro(50)
;asm86 :f1:bsasi.a86 macro(50)                 ;new line
;asm86 :f1:bscsi.a86 macro(50)                 ;new line
;
link86 &
     :f1:sdm86.lib(monitor), &
     :f1:%0.obj, &
     :f1:bsl.obj, &
     :f1:bserr.obj, &
     :f1:sdm86.obj, &
&    :f1:b204.obj, &
&    :f1:b206.obj, &
&    :f1:b208.obj, &
&    :f1:b215.obj, &
&    :f1:b218a.obj, &                         ;new line
&    :f1:b251.obj, &                          ;new line
&    :f1:b254.obj, &
&    :f1:bsasi.obj, &                         ;new line
&    :f1:bscsi.obj, &                         ;new line
     :f1:bsl.lib, &
     8087.lib to :f1:%0.lnk
;
loc86 &
     :f1:%0.lnk &
     addresses(classes(sdm86_data(400h), &
     stack(3c000h), &
     code(0f8000h))) &
     order(classes(sdm86_data, sdm86_stack, &
                   stack, data, boot, &
                   code)) &
     segsize(boot(1800h)) &
     start(%1) bootstrap noinitcode
;
exit
```

Figure B-1. Contents Of SDMGNB.CSD

Specify that the even-numbered bytes of the BS1 (first stage) file are to go into EPROM 0 and the odd-numbered bytes are to go into EPROM 1. (The address FD800H is an example value for a particular configuration. The numbers 3, 2, and 1 match ipps prompts for defining the information.)

```
format :f1:bs1(FD800H)
3
2
1
0 to :f1:bs1.evn
1 to :f1:bs1.odd
<cr>
```

Tell the software to program one EPROM with even-addressed bytes. (The address 0C00H is correct for the Bootstrap Loader-iSDM 286 combination.)

```
copy :f1:mbs1.evn to prom(0C00h)
```

Do the same thing for the odd-numbered bytes.

```
copy :f1:mbs1.odd to prom (0C00h)
```

Exit the ipps program.

```
exit
```

***

INDEX

Underscored entries are primary references.