

USER'S MANUAL

EPROM PROGRAMMER

Congratulations for selecting the EP-1 EPROM Programmer, a compact highly versatile EPROM programmer. Intended for use in development, production, or the field, the unit can be software configured to program over 600 different devices in its 28-pin socket. The list of devices includes almost every 24- and 28-pin EPROM and EEPROM used today. The EP-1 can also program microcontrollers with optional heads. The EP-1 is operated from the keyboard and CRT on your microcomputer.

The exact memory device to be supported is selected from a menu by manufacturer and specific part number. The algorithm specified in the manufacturer's data sheet is then recalled by the EP-1's database to program or read your part. This eliminates confusion caused by incompatible parts having the same generic part number, thus avoiding costly mistakes. The EP-1 automatically configures its hardware to use the correct voltage, pin out, and timing for your chip.

Any supported part may be read, programmed, or verified without setting any switches or changing personality modules. Parts may be duplicated by first reading the original chip into a file, then programming a blank chip from that file.

If you are using a PC or compatible and you want to skip over the details, turn to Chapter 2, Quick Start.

COPYRIGHT AND LEGAL DISCLAIMER

EP-1 User's Manual, V3.12 1/4/90

<189> 1987, 1990 by BP Microsystems, 10681 Haddington #190, Houston, TX 77043. (713)461-9430
(800)225-2102 FAX(713) 461-7413 BBS(713)461-4958.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of BP Microsystems.

EP-1 and EPEDIT are trademarks of BP Microsystems. IBM, and PS/2 are registered trademarks of International Business Machines Corporation. Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation.

The information in this manual is subject to change without notice and, except for the warranty, does not represent a commitment on the part of BP Microsystems. BP Microsystems cannot be held liable for any mistakes in this manual and reserves the right to make changes to the product in order to make improvements.

TABLE OF CONTENTS

<i>User's Manual</i>	1
<i>EPROM Programmer</i>	1
<i>Copyright and Legal Disclaimer</i>	3
<i>Table of Contents</i>	4
<i>CHAPTER 1 Introduction</i>	7
<i>CHAPTER 2 Quick Start</i>	9
<i>Serial Port and Cable</i>	9
<i>Installation</i>	9
<i>Operation</i>	9
<i>Selecting a Chip</i>	10
<i>Placing a Chip in the Socket</i>	10
<i>Reading a Chip</i>	10
<i>Programming a Chip</i>	10
<i>Returning to DOS</i>	10
<i>CHAPTER 3 Operation</i>	12
<i>Selecting a Chip</i>	12
<i>Inserting Chips</i>	12
<i>CHAPTER 4 Command Reference</i>	13
<i>General Information</i>	13
<i>Examples</i>	13
<i>Commands That Use Ranges</i>	14
<i>Examples</i>	14
<i>Command Summary</i>	14
<i>Addr</i>	16
<i>Blank</i>	17
<i>Base</i>	18
<i>Chip</i>	19
<i>Echo</i>	20
<i>Fill</i>	22
<i>Find</i>	23
<i>Help</i>	24
<i>List</i>	25
<i>Manual</i>	26
<i>Page</i>	27
<i>Parts</i>	28
<i>Program, PB</i>	29

<i>Protect</i>	31
<i>RB, RH, RI, RM, RT</i>	32
<i>Quit</i>	33
<i>Stat</i>	34
<i>Sum</i>	35
<i>Test</i>	36
<i>Verify, VB</i>	37
<i>;</i>	38
CHAPTER 5 Programming Heads	39
<i>Using a Head</i>	39
APPENDIX A Introduction to EPROMs	41
APPENDIX B Erasing EPROMs	42
APPENDIX C Copying EPROMs	43
<i>Changing the Data in an EPROM</i>	45
APPENDIX D Hex Files	46
<i>Hex Files</i>	46
APPENDIX E Compilers & Assemblers	51
APPENDIX F Addressing Modes	54
<i>The Base Command</i>	54
APPENDIX G EP.EXE Program	56
<i>Batch Mode Operation</i>	57
<i>Environment Variable</i>	57
APPENDIX H RS-232 Cable	59
APPENDIX I Non-PCDOS Operation	60
<i>Handshaking</i>	60
<i>Establishing Communications</i>	60
APPENDIX J Error Messages	62
<i>Error Messages Generated by the EP-1</i>	62
<i>Error Messages Generated by EP.EXE</i>	63
APPENDIX K Troubleshooting	66
<i>EP-1 Will Not Sign On</i>	66
<i>A Chip Will Not Program Correctly</i>	66
<i>Characters Are Lost in Communication</i>	66
APPENDIX L Specifications	68
<i>General</i>	68
<i>Communications</i>	68
Appendix M Limited Warranty	70

CHAPTER 1

INTRODUCTION

The EP-1 EPROM programmer is a standalone unit containing its own microcomputer and specialized hardware to program EPROM's. There are no operating controls on the unit (except the power switch); the unit is connected to a computer so commands may be typed at the computer's console to control the programmer. The unit communicates with the host computer via an RS-232 (serial) port. The programmer in conjunction with your computer gives you the ability to:

- Program EPROMs and EEPROMs
- Read existing devices
- Copy ROMS and EPROMS
- Program Microcontrollers (with optional heads)
- Calculate and set checksums
- Manually program EPROMs and EEPROMs

Most of this manual assumes you are familiar with EPROMs, their programming and erasing, your computer's operating system, and file formats. If you want to learn more about these concepts, you may read the tutorials in the appendices.

The EP-1 programmer can be used with the communication software we provide (EP.EXE) on IBM compatible computers (PC's, XT's, AT's, and their clones) running PC DOS. It can also be used on other computers with other communication programs. Since most users use the software we provide with the EP-1, the manual assumes you are using the software then notes the differences that apply if you do not use EP.EXE.

Memory Chips

All the devices supported by the EP-1 contain a block of memory that can be read. Most of the devices can also be programmed (you cannot program a ROM). EPROM's can be programmed by the EP-1 and erased using an ultraviolet light source, but only if the chip has a window (some parts are made of plastic and don't have a window). EEPROM's are automatically erased when the program command is activated. Zero-power RAM chips use internal batteries to retain data and can be programmed without erasing.

The data in the chip may represent any information: a program, a character generator, a lookup table, a logic pattern, etc. This data consists of some number of bits organized as bytes (8 bits at a time). For example, the 2764 has 65536 bits organized as 8192 bytes. The data pattern can be represented in your computer in many ways. One way is a binary file--8192 bytes that are exactly the same as the chip's data pattern. The difficulty with binary files is that they contain many characters that cannot be printed (only 95 ASCII characters print, while the file may have 256 different characters). The solution to this problem lies in hex files.

Hex Files

Hex files represent the data in your chip in ordinary lines of text. Each data byte is converted to two "hex" characters (0-9,A-F); additionally, address, checksum, and other information may be included in the file.

The EP-1 supports 6 hex file formats: straight hex, hex-space format, Intel MCS-80, Intel MCS-86, Motorola, and Tektronix. When using EP.EXE, you can also use binary files. These different formats give you the flexibility to communicate with almost any system generating data for EPROM's. Any file format can be used with the data from any chip. It doesn't matter which format you choose to store the data.

The EP-1 communicates in ASCII over the RS-232 connection. The communication program acts as a terminal emulator most of the time; in other words, it sends characters from the keyboard to the EP-1 while displaying characters that come back from the EP-1. It can also send a file (while programming or verifying) and save files to disk (while reading chips).

The EP-1 does not read your entire file into memory before programming; instead, it programs the chip while the data is being sent to the programmer. This is why the maximum device size is not limited by available memory and you do not have to buy expensive expansion memory. Further, operation is not slowed down because data can be transferred faster than chips can be programmed.

The Basics

The EP-1 is operated by commands you type at your keyboard. Most people will only need the following commands:

- CHIP
- RI
- BLANK
- PROGRAM
- QUIT

These commands are described in Chapter 2 along with a simple guide to installation for PC users.

Thank you for choosing the EP-1.

CHAPTER 2

QUICK START

This chapter is a simplified guide to installing and operating the EP-1 with an IBM PC, XT, AT, PS/2, or compatible. Users of other computers should refer to Appendix I.

SERIAL PORT AND CABLE

Look at the back of your computer and identify the serial port. Most computers have a male DB-25 connector (with 25 exposed pins in two rows). Some computers, like the AT and PS/2 computers, use a DB-9 connector with 9 pins. If your computer does not have a serial port, you must install one before you can operate the EP-1. Most computers can use up to two serial ports. If you do not have a free serial port you can switch cables back and forth or use an RS-232 switch box.

You also need a cable to connect the EP-1 to the computer. The cable is a "straight" RS-232 cable with a male DB-25 connector for the EP-1. See Appendix H for more information.

INSTALLATION

Plug the EP-1 power cord into a power outlet. Plug the serial cable into the EP-1 and your computer. Turn on both the computer and the EP-1. The EP-1 will perform a ROM test, RAM test, and serial port test when power is applied. The LED's will flash in sequence as these tests are performed, leaving just the green power LED on.

OPERATION

Insert the PCDOS software diskette into drive A:. If the EP-1 is connected to COM1:, type A:EP 1 and press ENTER. If the EP-1 is connected to COM2:, type A:EP 2 and press ENTER. The EP-1 can be connected to other serial ports, but you will need to read the detailed instructions in Appendix G.

The EP-1 programmer should sign on with its copyright notice and software version number:

A:EP 1
EP-1 EPROM Programmer 3.14
Copyright 1985-1989 BP Microsystems

You are now at command mode<197>the programmer is waiting on you to type a command.

SELECTING A CHIP

You must tell the EP-1 what kind of chip you are going to place in the socket. Use the chip command to select both the manufacturer and part number:

- Type C ENTER to activate the chip command.
- Type the number or name of the chip manufacturer and press ENTER.
- Select the part number the same way.

The EP-1 is now configured to use the proper pin configuration, programming algorithm, and programming voltage associated with your chip.

PLACING A CHIP IN THE SOCKET

The EP-1 can program either 24- or 28-pin chips in its 28-pin zero insertion force (ZIF) socket. Identify pin 1 on your chip. It will have a dot or notch near it. Lift the handle on the socket, place the chip in the socket with pin 1 away from you, and, on 24-pin parts, leave four holes empty at the top of the socket. Press the socket lever down.

✓ Parts must not be inserted or removed when the ACTIVE LED is on. Parts must be removed before turning the programmer power on or off.

READING A CHIP

The part will be read and its data stored in a hex or binary file. The command you use here will determine the file format. In this example, we will create an Intel format hex file:

- Press RI ENTER
- Specify a file name to hold the data and press ENTER.

If the file name already exists in the current directory, it will be overwritten. You can specify the disk drive and directory in the file name (i.e. C:\HEX\BIOS.HEX).

PROGRAMMING A CHIP

Before programming an EPROM, you should always make sure it is blank:

- Type B ENTER.

The chip is read to make sure it is properly erased.

When programming, a hex or binary file is read from the disk and used to program the part.

- Type P ENTER. The programming and active LED's will come on.
- Press ENTER to use the file name you entered above, or type a new file name.

Hex file formats are automatically determined. If errors occur during programming, they will be listed on the screen.

RETURNING TO DOS

To return to DOS, use the Quit command.

CHAPTER 3

OPERATION

SELECTING A CHIP

Before you can read or program a chip, you must tell the programmer what kind of chip you have. You will need to know the manufacturer and part number of your chip.

Most manufacturers mark their trade mark and the device number on the chip package. If you don't know the manufacturer by name, you should use the FIND command for help.

Prefixes and suffixes are usually added to part numbers indicating speed and package type. These suffixes generally should be ignored; however, it is very important to notice the suffix "A" or "B." These indicate that a different programming algorithm must be used with the part.

You will select the manufacturer and part number from menus using the C command.

Example: Suppose you have an Intel "D2732A-3." The "D" and "-3" indicate package type and speed, respectively. You would ignore these and choose the selection "2732A" from the Intel menu.

INSERTING CHIPS

The EP-1 can program either 24- or 28-pin chips in its 28-pin zero insertion force (ZIF) socket. Identify pin 1 on your chip. It will have a dot or notch near it. Lift the handle on the socket, place the chip in the socket with pin 1 away from you, and, on 24-pin parts, leave four holes empty at the top of the socket. Press the socket lever down.

- ✓ **You should follow these rules when using the EP-1. If you break these rules, you run the risk of destroying your chips:**
 - ✓ Chips should not be left in the socket when power is turned ON or OFF.
 - ✓ Chips should not be placed in the socket if the wrong part is selected (select new part using C command, then place the part in the socket).
 - ✓ Chips should not be inserted or removed while the "Active" LED is on.

CHAPTER 4

COMMAND REFERENCE

GENERAL INFORMATION

The programmer is ready to accept a command when you see the command prompt (i.e. "Am27256>>"). The command prompt always consists of the part number of the currently selected chip and the ">>" character.

The following information applies to all commands:

You only need to enter enough characters to make the command unique: (i.e. H activates the HELP command)

- Parameters are typed following the command name.
- Command names and Parameters must be separated by one or more spaces.
- Commands are entered when you type Carriage Return or Enter.
- You can type Control-S at any time to pause the display (PCDOS only)
- You can type Control-C (^C) at any time to terminate the current command and return to the command prompt.
- Pressing return for any menu selection causes the command to be canceled with no action taken.
- All addresses and other numeric values are specified in hex format (base 16).
- The lower case letters in the command SYNOPSIS indicate parameters.
- When a file name is required (when using EP.EXE), DOS file naming conventions are used. You can specify drive names and paths (i.e. C:\HEX\BIOS.ROM).

Some commands let you choose a selection from a menu. You can respond to a menu with either the number of the selection you desire, or the name of the selection you desire. If you specify by name, you only have to specify enough to make your selection unique.

The menu response can be typed when the menu asks for it, or you can type your response on the command line.

EXAMPLES

ADDR

this produces a menu; you can then make your selection

>>C MOTOROLA MCM68764

-this selects both manufacturer and part number at once, suppressing both menus

>>C MOT

-this selects Motorola, but gives a menu to choose the part from.

>>C 1

-this takes the first selection, AMD, and produces a menu of parts

- When you write software or batch files to use with the EP-1, be sure to use the full name of the selection, not its number. As we add new parts to the menus in future software releases, the menu numbers will change. If you use the full name for your selection, your software will remain upwardly compatible with future releases of the EP-1.

COMMANDS THAT USE RANGES

A number of commands can use a range of addresses. Ranges can be used to make the command operate on specific addresses. The addresses here refer to addresses at the chip, not in your file.

Ranges are optional and are specified as two hex numbers. If no range is explicitly given, the command will operate on the whole chip. If a range is given, only those bytes specified will be used by the command. Ranges can also be used to exclude a sequence of bytes from the command.

EXAMPLES

2732>>P

-this programs the entire chip

2732>>L 3E0 43F

-this lists the bytes from 3E0 to 43F inclusive.

2732>>P 30

-this programs all location from 30 to the end of the chip. When the second parameter is omitted, the range continues to the end of the chip.

2732>>P 580 - 57B

-this excludes the four bytes in the range 57C to 57F (57C,57D,57E,57F).

When excluding a range, the larger address comes first, the smaller second. The two addresses specified are always used while the bytes between them are not. Only the program (P) and verify (V) commands can use exclusive ranges.

COMMAND SUMMARY

Setup commands change a mode that affects programming and reading parts later.

CHIP = Select a chip

BASE = Set base address

ADDR = Set addressing mode

PAGE = Select page (on paged parts only)

Informative commands display the status of the EP-1 and provide help:

HELP = Get a quick command summary

STAT = Display current status

FIND = Find a specific part from its name

PARTS = List all parts supported

? = Programmer info

These commands read chips and produce a file or tell you about the contents of the chip:

LIST = List the contents of the chip

RB = Read the chip, Binary file

RH = Read chip, producing a binary or straight hex file

RI = Read the chip, Intel hex file

RM = Read the chip, Motorola hex file

RT = Read the chip, Tektronix hex file

VB = Verify the contents of a chip (by reading a binary file)

VERIFY = Verify the contents of a chip (by reading a file)

SUM = Calculate the checksum of a chip

BLANK = See if a chip is Blank (erased)

Commands that program chips:

PROGRAM = Program the chip (from a hex file)

PB = Program the chip (from a binary file)

MANUAL = Manually program bytes

FILL = Programs a range with a specified value

SUM = Set the chip's checksum

PROTECT = Protect a chip (Only on certain parts)

Miscellaneous commands:

ECHO = Select echo mode (only used in non-PCDOS environment)

; = Comment (useful in batch files)

TEST = Perform a hardware test of the EP-1

Many of the commands support complicated programming techniques, non-PCDOS operation, and testing. Therefore, you can ignore most of the commands until you have a special need.

ADDR

Description

Split up data files during programming for use with 16- and 32-bit systems.

Synopsis

A mode

Application

Use this command when you have one data file that will be programmed into multiple EPROMs to be used in a 16- or 32-bit system.

Operation

When generating EPROMs for a 16-bit system, half the data (even addresses) is programmed in one chip, while the other half (odd addresses) is programmed into another chip. Both chips are read simultaneously in the target system to yield a 16-bit word.

When generating EPROMs for a 32-bit system, you must make 4 EPROMs: the first EPROM gets the first out of each 4 bytes, the second gets the second out of each 4, and so forth.

The parameter selects the desired mode:

ALL: = Program all bytes (default)

EVEN: = Program even bytes only

ODD: = Program odd bytes only

1/4: = Program using the first byte of each four

2/4: = second byte of four

3/4: = third byte of four

4/4: = fourth byte of four

See Also

Appendix F

BLANK

Description

Read the chip and see if it is erased.

Synopsis

B aaaa aaaa

Application

You should always blank check an EPROM before programming.

Operation

The EPROM is searched for any bytes that are not erased. When EPROMs are erased, their bytes are all FF.

The two parameters may be used to specify a range to check.

If the EPROM is not erased, an error message is generated.

See Also

Appendix B

BASE

Description

Tell the programmer at what address to start programming (from the file).

Synopsis

BA aaaaaaaaa

Application

This command lets you make several EPROM's from one file. It also lets you program from files that don't start at address 0.

Operation

If no parameter is given, the current BASE setting is shown.

When programming, the byte at the base address will program byte 0 of the EPROM.

Specifying a parameter will change the current BASE setting. The parameter is an 8-digit positive or negative hex number. Negative BASE values are displayed in two's complement form.

The BASE is subtracted from incoming file addresses when programming. When using it with one of the addressing modes (ADDR command), the BASE is subtracted before the file is split.

See Also

Appendix F

CHIP

Description

Select a particular device before programming or reading.

Synopsis

C mfgr dev

Application

You have to tell the EP-1 what part you want to read or program. This sets the programming algorithm, pin-out, and programming voltage.

Use this command before you place a different chip in the socket.

Operation

Both parameters are optional. If a parameter is omitted, a menu will appear listing all options.

The first parameter specified the manufacturer of the chip. The second specifies the manufacturer's part number, as printed on the device.

Example:

```
C NEC
1) uPD2716
2) uPD2732
3) uPD2732A
4) uPD2764
5) uPD27C64
6) uPD27128
7) uPD27256
8) uPD27C256
9) uPD27C256A
10) uPD27C512
Selection: 8
```

Chip Type: NEC uPD27C256

File Address Range: 0 - 7FFF

uPD27C256>>

See Also

Chapter 3

ECHO

Description

The echo command gives users that cannot use EP.EXE the ability to control what file transfer protocol is used with the host computer.

Synopsis

E p

Application

If you are not using EP.EXE, you will probably want to set the protocol to XMODEM or CRC settings.

Operation

The parameter p selects what is echoed back to the computer:

PCDOS

(default)

File is not echoed, but special control sequences activate communication program. Used with EP.EXE

V2.xx

(ASCII transfers)

PCDOS3

File is not echoed, but special control sequences activate communication program.

Used with EP.EXE V3.xx (XMODEM transfers)

ON

Files are echoed back to the host while programming.

OFF

Nothing is sent to the host during programming, except error messages.

XMODEM

Uses XMODEM file protocol.

CRC

Uses CRC XMODEM file protocol.

The PCDOS mode is used with the EP.EXE program. During programming, the EP-1 sends one byte of the programming address (in hex) to indicate progress, followed by two backspace characters. When certain commands are activated, the EP-1 also sends special sequences:

P = ESC p

PB = ESC P

V = ESC v

VB = ESC V

RB = ESC b

RH = ESC h

RI = ESC i

RM = ESC m

RT = ESC t

QUIT = ESC q

See Also

Appendix I

FILL

Description

Program a range with a value.

Synopsis

F aaaa aaaa dd

Application

Use this command when you want to program a range with a hex value.

Operation

Fill can be used to change the contents of an EPROM to a specified value. The first parameter is the starting address. The second parameter is the ending address. The data is read as a hex pair.

Example:

The first 1000 hex bytes of an EPROM contain data that you want to change to 00.
MBM27C256>>FILL 0000 1000 00

<F1P8M>MBM27C256>><F255P255D>

FIND

Description

List every supported part that fits a description.

Synopsis

F string

Application

This command is useful when you don't know the manufacturer of a part, or you want to know which manufacturers make a specific part.

Operation

All parts are listed that contain the string in their names.

Example:

```
@EXAMPLE = FIND MBM27C256
Fujitsu MBM27C256
Fujitsu MBM27C256A
2 Found
>>CHIP FUJITSU MBM27C256
MBM27C256>>
```

HELP

Description

The HELP command gives a brief summary of the commonly used commands.

Synopsis

H

LIST

Description

This command shows you the contents of the chip in the socket.

Synopsis

L aaaa aaaa

Application

To quickly identify what is in a chip.

Operation

The contents of a chip are listed over the range of addresses specified.

Both parameters are optional; the default values will list the entire EPROM.

The listing shows addresses, hex values of bytes, and ASCII character equivalents. If the byte is not a printing ASCII character, a period ('.') is printed.

You can pause a listing using Control-S (^S); continue by pressing Control-Q (^Q).

You can obtain a hard-copy listing by turning your printer (LPT1:) on and off with Control-P (^P).

MANUAL

Description

Manually program bytes.

Synopsis

M aaaa aaaa

Application

This command is useful when patching an EPROM or entering a small amount of data.

Operation

The chip is manually programmed in the range of addresses given.

Both parameters are optional. The default will start at address 0.

The command will show you an address and its current contents. You may program a new value by typing two hex characters. You may enter an ASCII character by entering a single quote and the character (e.g. 'X).

The byte may be skipped (not programmed) by typing RETURN.

The command may be terminated using a period ('.') or Control-C (^C).

PAGE

Description

Select the active page on a paged part.

Synopsis

PAG n

Application

The pages of a paged EPROM behave like separate EPROM's. Selecting a page will let you read and program it.

Operation

The page number n will be activated.

The STAT command indicates the total number of pages on the part. The parameter n may be in the range 0 to number of pages - 1.

PARTS

Description

List all the parts that the programmer can program.

PROGRAM, PB

Description

Program an EPROM from a file on disk.

Synopsis

P aaaa aaaa (program from a hex file)

PB aaaa aaaa (program from a binary file)

Application

Use this command to load data into a memory chip. It is used to make an original chip or a copy.

Operation

- ✓ **The EPROM type must be selected with the Chip command before using this command.**
- ✓ **The EPROM is programmed from a file over the range of addresses given. Both parameters are optional; the default values will program the entire chip. Only addresses included in the range will be programmed.**

The program command will automatically recognize straight hex, hex-space format, Intel (MSC80), Intel extended (MCS86), Tekhex, and Motorola S-Records (S0-S9).

A binary file may be specified by using the PB command.

The command will proceed to program the chip. You will see two digits of the current address being programmed.

Bytes are verified after they are written. It is not necessary to use the verify command. If an error occurs, you get an error message immediately showing the address and data written and read back.

TO PROGRAM A RANGE:

You can specify which bytes are to be programmed. To program only the bytes from 143 to 7F0, use:

EXAMPLE = 2764>>P 143 7F0

To prevent the bytes in the range 302 to 348 from being programmed, use:

EXAMPLE = 2764>>P 349 301

Like the previous example, both addresses specified (349, 301) will be programmed. The addresses that fall between them are not programmed. The larger address comes first in this case.

The two addresses specified refer to offsets within the EPROM. If BASE or ADDR modes are set, these addresses will differ from the file addresses.

EPROMs must be properly erased before programming. Even though the PROGRAM command will verify, it is still a good idea to use the BLANK command before programming because it is more sensitive to unerased bits than the verify-while-programming (because verify-while-programming uses Vcc=6.0V on most parts and BLANK uses Vcc=4.8V).

EEPROMS are erased a byte at a time while programming, in general. A few EEPROMs do not support byte erase: a chip erase is performed before programming the first byte.

To copy or change an EPROM: See Appendix C.

When not using PCDOS

Activate the PROGRAM command then send a hex file. If you are using a communication program, you should type the keystrokes that will activate the file send function. After the file has been transferred, press Control-Z (^Z). Programming error messages will be displayed while the file is being sent to the programmer. You can tell the programmer to echo the file while it is being sent using the ECHO command.

Some communication programs throw in a few random characters before sending a file. If you have this problem, try placing this at the start of your file:

```
EXAMPLE = <<Carriage Return>>
PROGRAM<<Carriage Return>>
...hex data follows
```

The first return will generate an innocuous error message if there were some random characters sent. The PROGRAM command and the hex data will then be entered.

See Also

Appendix B, Appendix F

PROTECT

Description

Set the protection bits in a chip. Not all chips have this feature.

Synopsis

PROT

Application

To protect your proprietary design from unauthorized people attempting to read the part.

Operation

The PROTECT command sets the protection bits in chips that support this feature. The security bit makes some microcontrollers impossible to read (copy) while still operating normally. Once you have used this command, you cannot read or verify the part.

 *When protecting an Intel 87C51, both security bits are programmed rendering the part unreadable.*

RB, RH, RI, RM, RT

Description

Read a chip and produce a file on the disk in the specified format:

RB = Binary file

RH = straight hex file

RI = Intel hex file

RM = Motorola hex file

RT = Tekhex file

Synopsis

```
RB aaaa aaaa
RH aaaa aaaa
RI aaaa aaaa
RM aaaa aaaa
RT aaaa aaaa
```

Application

Used when copying or modifying chips.

Operation

The device is read over the range of addresses specified (range is optional).

You must specify a DOS filename to create. If there was an existing file with that name, it is erased and the new file takes its place.

To copy or change an EPROM, see Appendix C.

When not using PCDOS:

The command immediately produces the hex file. After the file has been transferred, an EOT (^D) is sent before returning to the command prompt. Binary files cannot be generated. Also refer to the ECHO command and Appendix I.

When using a modem program, you should type the command (i.e. RI), start the file capture mode, and press ENTER.

QUIT

Description

Return to DOS.

Synopsis

Q

Operation

The QUIT command stops the communication program and returns to DOS.

The currently selected chip, the base and address modes, and the active page will be saved as long as the EP-1 power remains on.

An alternate way to quit is to use Control-Break. This method stops immediately--even in the middle of a command.

In batch file mode, the quit command will go to the next file if there is one (see Appendix G).

When not using PCDOS

The QUIT command causes the EP-1 to send ESC q. If you are writing your own communication program, this could make it return to DOS.

STAT

Description

Gives a status report showing the current configuration.

Synopsis

S

Application

This is handy for troubleshooting because it shows all the modes that can affect the program read or verify commands.

Operation

The current programmer status is shown. This includes the current chip manufacturer and part number, file address range, addressing mode, page number, and baud rate.

The addressing mode is omitted when it is the default (ALL). The page number only appears when it is relevant.

See Also

CHIP, BASE, ADDR, and PAGE commands, Appendix G

SUM

Description

Read a part and calculate the checksum. Program a part so its checksum is known.

Synopsis

SU

SU aaaa ss

Application

Reading a checksum can tell you if a chip is not programmed correctly (this is useful when you have an original chip you are about to duplicate). Checksums are often set on BIOS EPROM's and similar applications so the microprocessor can make sure its own EPROM is valid.

Operation

SUM calculates the checksum of the currently selected part. The value of each byte is added up and the sum is displayed. The 8-bit sum is a modulus 256 sum, while the 16-bit sum is a modulus 65536 sum.

You can set a checksum by specifying an address and a sum. You must specify what you want the 8-bit sum to be (ss) as well as the address of a blank byte that can be programmed (aaaa). The EP-1 will calculate a sum for the chip and program the location with the byte required to make the desired sum.

Example:

The last byte of a 2764 is unprogrammed and you want the chip's checksum to be 0. Use this command:

2764>>SUM 1FFF 0

TEST

Description

Perform a test on the programming hardware. The test command will completely test the output circuitry on the EP-1.

Synopsis

TEST

Application

To diagnose a failure and to verify proper calibration.

Operation

To perform the test, you need an accurate digital voltmeter.

Pins are numbered starting with the top left pin (1), to the bottom left (14) and back up the right side (15 at bottom, 28 at top).

Remove any chip from the EP-1 socket. Type TEST and connect the negative lead of your DVM to pin 14 of the 28-pin ZIF socket. Set the voltmeter to the 20V DC scale. The program will ask you to test the voltage at each pin. Press RETURN to go to the next pin. You may need to switch to a higher scale when measuring 25V.

Logical High (H) is any voltage between 2.8 and 5.0. Logical Low (L) is any voltage between 0 and 1.4. Any voltage stated specifically (e.g. 25V) should be within 0.3 V (e.g. 24.7 to 25.3).

The EP-1 employs a highly stable IC voltage reference. It is highly unlikely that it will drift significantly with temperature or age. If all the voltages appear too high or too low, you might want to try another voltmeter before panicking.

Each output pin driver is protected by current limiting so the drivers are very reliable and you will probably never need this command.

If you are getting errors because a pin produces the wrong voltage, contact BP Microsystems.

VERIFY, VB

Description

Read a chip and compare the data to a file.

Synopsis

V aaaa aaaa (program from a hex file)

VB aaaa aaaa (program from a binary file)

Application

To be certain that a chip is programmed correctly.

Operation

See the program command (P) for details on parameters and file formats.

If the data in the chip does not match the data from the file, an error message is generated:

ERROR: (aaaa) = rr SHOULD BE ff.

The address aaaa did not have the correct data. It is rr; it should be ff.

- EPROMs and EEPROMs are automatically verified during programming. It is not necessary to verify manually following programming.*

;

Description

This is the comment command. It performs no action.

Synopsis

; line of comments

Application

To place comments in a batch file.

Operation

Anything following the ";" is ignored. There must be a space following the ";".

CHAPTER 5

PROGRAMMING HEADS

The EP-1 can program and read a number of single-chip microcontrollers. These are 40-pin chips with a CPU, RAM, and EPROM (or ROM) all on a single chip.

Chips that require a programming head are marked in the parts lists with an asterisk and the head number required (i.e. Intel 8751H *1C).

HEAD-1A supports the 8041A, 8042, 8048AH, 8049AH, 8050AH, 8741A, 8742A, MBL8742H, 8748, 8748H, and 8749H.

HEAD-1C supports the 8751H and 87C51 parts.

USING A HEAD

Lift the lever on the EP-1 socket. Place the head over the socket and press the socket lever down.

You may select the chip type before or after placing the head on the EP-1; however you must select the chip type before placing the chip in the socket. Use the EP-1 just like you would with standard parts.

Most of the microcontrollers may be secured. When the 8751 parts are secured, both security bits are set rendering the part unreadable.

APPENDIX A

INTRODUCTION TO EPROMS

EPROM is an acronym for Erasable Programmable Read Only Memory. EPROMs, EEPROMs (Electrically Erasable Programmable Read Only Memory), and ROMs (Read Only Memory) are all nonvolatile memory chips. Each stores data in an array of bytes (8 bits in a byte). Data is stored in ROMs using a mask during manufacturing. EPROMs and EEPROMs are programmed electrically. They are called "Read Only" because in most applications they are programmed once (or occasionally in the case of EEPROMs) but read many times. The size of a memory chip is usually referred to in bits. A 512K EPROM contains 524,288 (512 <F128M>"<F255D> 1024) bits, 65,536 (524,288 <F128M>~<F255D> 8) bytes or 64k bytes.

During programming, the EPROM stores bits of information by placing a charge on the gate of a transistor (one for each bit). When you read an EPROM, the pattern of charged transistors gives you the data back. The charge remains on the transistor until the chip is erased.

EPROMs are erased by exposing the chip to short-wave ultraviolet light. The high-energy light knocks electrons out of the gate, removing the charge.

When an EPROM is erased, all bits are returned to a 1 state (all bytes are FF). When a bit is programmed, its value is set to 0. Thus, bits can be programmed low at any time, but they can only be set high through erasure.

APPENDIX B

ERASING EPROMS

In order to clear the data in an EPROM (in preparation for programming), the chip is exposed to shortwave ultraviolet light through the quartz window on the top of the package. A certain dosage is required to fully erase the part, so brighter light sources erase the part more quickly. Most erasers require between 3 and 30 minutes to erase an EPROM.

EPROMs are usually erased using a mercury vapor bulb emitting 2537 Angstroms. The bulbs most commonly used are like fluorescent tubes, but without the phosphor; they are often called germicidal bulbs. The ultraviolet light can be harmful to eyes, so erasers are equipped with shields to prevent light leakage.

You can use the BLANK command to find out if your chip is erased. Use this procedure to determine erasing time: start with a programmed chip; erase the part in one minute increments and use the BLANK command to test it each minute; once the chip passes the BLANK test, double the erase time to give you an adequate margin for voltage and temperature changes.

- Some types of parts take longer to erase than others. You may need to experiment with the parts you use.
- The adhesive used on labels often blocks UV light. Try cleaning the window with alcohol or another solvent if the chip erases slowly.
- Sunlight and fluorescent light can erase chips; however, it usually takes months or years. You should cover the window with an opaque label to make the data permanent.
- Production EPROMs are available packaged in plastic instead of ceramic. These parts do not have a window and cannot be erased.

APPENDIX C

COPYING EPROMS

The data can be read from an existing ROM or EPROM and placed in a file. Any number of EPROMs can then be programmed from that file. You cannot copy a chip without making a file first because the EP-1 has no memory to store the data. When you copy an EPROM, you will first select the chip type you want to read and read the data into a file. You can then select the chip type you want to program and program that data into the chip. The source and target chips must be the same size but they can be different types. For example, you can read a Fujitsu MBM27C64 and program an Intel 2764. If the chips are different sizes, you could leave part of the target chip blank (if the target is larger) or place the data into more than one chip.

Install your programmer according to Chapter 2 before continuing.

Use the C command to select the chips, the RH command to read the data, and the P command to program the new part.

Start the EP-1 like you did during installation (p. 5):

```
EXAMPLE = A>>EP 1
EP-1 EPROM Programmer V3.14
Copyright 1985-1989 BP Microsystems
>>
```

Select the chip you want to read:

```
EXAMPLE = >>C
1) AMD          16) Ricoh
2) ATMEL        17) Rockwell
3) EXEL          18) Samsung
4) Fujitsu       19) Seeq
5) Generic        20) SGS
6) General Instrument 21) Signetics
7) Hitachi        22) SMOS
8) Intel          23) Synertek
9) Mitsubishi      24) Thomson
10) Mostek         25) TI
11) Motorola       26) Toshiba
12) National        27) VLSI Technology
13) NEC            28) White Technology
14) OKI             29) Xicor
15) Quick Pulse
```

Selection: 13

NEC:

- 1) uPD2716
- 2) uPD2732
- 3) uPD2732A
- 4) uPD2764
- 5) uPD27C64
- 6) uPD27128
- 7) uPD27256
- 8) uPD27C256
- 9) uPD27C256A
- 10) uPD27C512

Selection: 8

Chip Type: NEC uPD27C256

File Address Range: 0 - 7FFF

uPD27C256>>

-or, you could have selected both on the command line:

>>C NEC UPD27C256

Chip Type: NEC uPD27C256

File Address Range: 0 - 7FFF

uPD27C256>>

Insert the chip into the socket and read it into a file:

EXAMPLE = uPD27C256>>RB

-specify a filename now:

Binary file to write: DATA.ROM

uPD27C256>>

The file now contains the data from the EPROM. The file size should be 32768 bytes in this case.

Select the chip you want to program, if it is different:

EXAMPLE = uPD27C256>>C INTEL 27256

Chip Type: INTEL 27256

File Address Range: 0 - 7FFF

Insert the new chip and see if it is blank then program:

EXAMPLE = 27256>>B

O.K.

27256>>PB

Binary file to read [DATA.ROM]:

8000 Bytes programmed correctly; No errors.

27256>>

-now you can program more parts with the P command, or go back to DOS:

QUIT
A>>

CHANGING THE DATA IN AN EPROM

The best approach is to read the original EPROM into a hex file, edit the file, and program a blank chip from the edited file. The hex file can be edited with an ordinary text editor program or a special hex file editor program.

BP Microsystems sells an inexpensive hex and binary file editor program called EPEDIT. The program will let you load the file and edit it as easily you can edit text files with a text editor.

The alternative to using a special hex file editor is to use a standard text editor or word processor in "non-document" mode.

Straight hex files (RH command) contain only hex data--16 bytes to a line (32 ASCII characters). If you edit a straight hex file, you must calculate the file address from the line number (because the file contains no address records). To calculate the address of the first byte on a line, use the line number (from your editor), subtract one, convert to hex, and multiply by 10h (16 bytes/line).

You can also edit Intel, Motorola, and Tekhex format hex files. These file formats include an address and checksum on each line. If you do not update the checksum when you change the data on a line, you will get a warning message when you use the PROGRAM command. You can ignore this message and the data will be programmed anyway. If you move or copy the lines in one of these files, it will not change the data being programmed because each line contains its own address information.

APPENDIX D

HEX FILES

HEX FILES

Hex files and binary files are used to store data that can be programmed into a chip. The hex files are ASCII files containing hex characters (0-9, A-F) and other information. Hex files on the BP-4x00 can have a maximum of 80 characters on a line. The BP-4x00 supports six hex file formats. Information on editing hex files can be found in Chapter 4 – Using the Data Editors.

Straight-hex Files

The straight-hex file format is the simplest. It consists of two hex characters for each data byte in the file. The hex characters are separated into lines with a Carriage Return, Line Feed sequence. The file contains no address information nor any checksums.

Hex-space Files

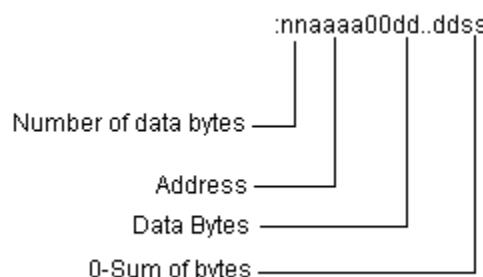
The hex-space format is identical to straight-hex format except spaces may be placed between hex pairs representing bytes.

Intel Hex Files

The BP-4x00 supports both MCS80 and MCS86 style Intel hex files. The MCS80 file format uses 16-bit addresses and is therefore limited to 64k bytes (2^{16} bytes). The newer MCS86 format (extended Intel format) adds an address offset record that extends the file addresses up to 20 bits, 1 Megabyte.

MCS80

Each MCS80 data record is formatted as follows:

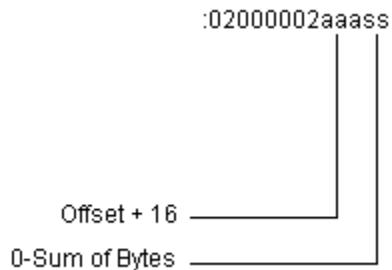


The end record is:

:00000001FF

MCS86

MCS86 files use an address offset record. The record specifies an offset to add to subsequent data records:

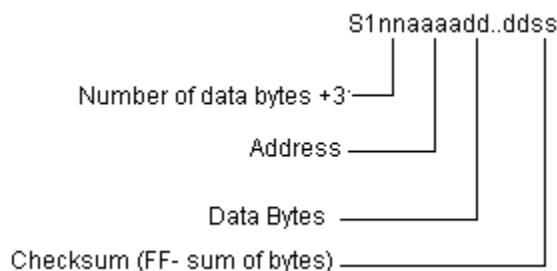


Motorola Hex Files

Motorola hex files support 16-, 24- and 32-bit addresses. The size limitation of this file format is 4 gigabytes. The BP-4x00 supports the full range of record numbers, S0-S9.

S1

S1 data records (64KB limit) are:

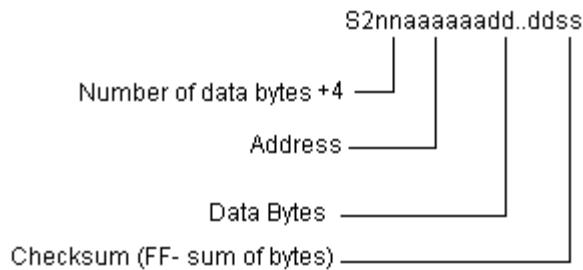


The end record is:

S9

S2

S2 data records (16MB limit) are:

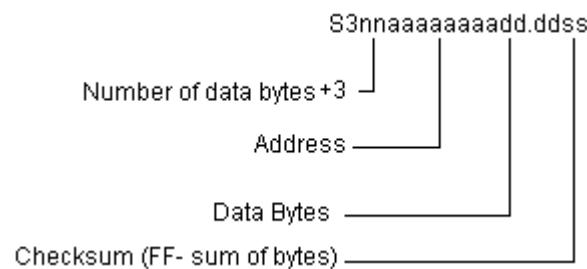


The end record is:

S8

S3

S3 data records (4GB limit) are:

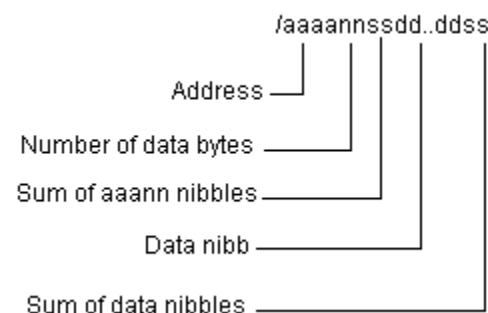


The end record is:

S7

Tekhex Files

Tektronix has their own hex format, Tekhex. It supports only 16-bit address records.



The end record is:

/00000000

Addresses

The Intel, Motorola and Tekhex files contain an address in each line of data. This address tells the BP-4X00 which bytes to program with the data on the line. Straight hex files and binary files do not contain addresses. The BP-4x00 counts each byte to generate an address, i.e., the first byte is 0, the next is 1 and so forth. Thus if you move or append lines to a straight hex file, the resultant EPROM will be different. If you move the lines around in an MCS80 Intel, Motorola or Tekhex file the data will not be changed.

Checksums

Intel, Motorola and Tektronix hex files include a checksum in every line. This code is calculated from the data on the line. When the BP-4x00 receives a line, it recalculates the checksum and compares it to the one from the file. If there is an error in the file (such as missing characters) the sum will not be correct and you will get the message:

Warning--Checksum Error:

line causing error is printed here

In this case, the programmer assumes the address and data information is correct anyway and program normally. If you edit the data fields in a hex file without updating the checksum, you will get this message.

APPENDIX E

COMPILERS & ASSEMBLERS

The data in an EPROM most often originates from an assembler or compiler. This section tells a little about how they work and the file types usually found. The descriptions here are very general<197>there are exceptions to everything.

Compilers take an ASCII source file (e.g. .C, .PAS) in a high-level language and convert it into an assembler language source file.

Assemblers take an assembler language source file (.ASM) and produce an object file (.OBJ). Many compilers skip the assembler language phase and produce an object file directly.

Object files (object modules) contain machine language instructions, data, addresses, and references. Multiple object files may be combined together (linked) to produce one program. When the object files are combined, they are essentially placed end-to-end one after the other. Addresses and references used in the object files cannot be known until all the object files are being linked.

The linker takes a set of object files and libraries and combines them to produce a loadable file. When a routine in an object file references a library routine, the linker extracts an object file (containing the referenced routine) from the library and combines it with the rest of the object files. The linker resolves all references and generates addresses relative to the loading address of the whole program. The loadable file it produces is ready to be loaded anywhere in memory and executed.

The loader takes the loadable (relocatable) file and makes a memory image. The loader decides where to place the loadable file in memory, then adjusts all address references by adding the new offset address. This is a quick process<197>it takes place every time you execute a .EXE file in PCDOS.

The output of the loader is a memory image, ready to be executed. This is what gets programmed into an EPROM. All of the references and addresses have been resolved and are unchangeable. The loader's output may go directly into memory, go into a binary file (.COM), or be translated into a hex file (.HEX).

When the file is loaded, the code (text segment or instructions) is copied into memory. Some of the memory (RAM) is initialized to the values specified in the file. This area is used as variables with pre-defined values. Another area of RAM is initialized to all zeros to be used for variables.

If you are writing a program to be placed in an EPROM, you must be certain that your RAM will be initialized properly. Some compilers will do this for you (such as Manx Aztec C); on assemblers, you should not use DB, DW, etc. in the RAM area to initialize variables (use DS or DB ?? instead).

Some linkers do not produce re-locatable (loadable) output, so they do not need a loader. Instead, they produce the hex (.HEX) or binary (.COM) file directly.

This executable file must be programmed into the EPROM and the EPROM placed in the target system so that the correct data will appear at the correct address in the target system. If you have a binary file, you simply program the EPROM from the file. The first byte of data in the binary file will appear at the starting address of the EPROM in the target system.

If you have a hex file, you must subtract the starting address of the EPROM from the addresses in the hex file before programming. This is accomplished using the BASE command.

Example:

You have an assembler program that starts like this:

```
EXAMPLE = org 4000h
```

You link and load the program, producing an Intel hex file ready to load at 4000h. The hex file starts like this:

```
EXAMPLE = :104000000...
```

The underlined characters are the load address of the data on that line. You can see that we must subtract 4000 to place the first data byte in the EPROMs first byte.

In order to program the chip, use these commands:

```
EXAMPLE = BASE 4000
```

```
P
```

- You must have a binary or hex file before you can program an EPROM.
- .OBJ files must be linked and loaded to produce a hex or binary file.
- Use the BASE command to specify the starting address of the EPROM

APPENDIX F

ADDRESSING MODES

The EP-1 gives you the ability to translate files while programming. This gives you the ability to:

- Program multiple chips from a file that will not fit in a single EPROM
- Take a file and produce 2 EPROMs for a 16-bit system
- Take a file and produce 4 EPROMs for a 32-bit system
- Skip the start of a file while programming
- Leave the beginning of an EPROM blank

THE BASE COMMAND

The BASE command stores the base offset. The base offset is subtracted from file addresses before programming:

EXAMPLE = EPROM addr = file addr - BASE

Example

You want to program a 32768 byte file into two 27128 parts.

```
27128>>BASE 0  
-first EPROM starts at 0 (default)  
27128>>P  
-specify your file  
27128>>BASE 4000  
-each EPROM will contain 16384 (4000h) bytes, so the second one starts at 4000h.  
27128>>P  
-specify the same file here
```

Example

You have a file with 65536 bytes to be placed into two 27256 EPROMs to be used in a 16-bit system. In the 16-bit system both chips are read simultaneously; hence, data splits as follows:

EPROM address	EPROM1	EPROM2
0	file byte 0	file byte 1
1	file byte 2	file byte 3
2	file byte 4	file byte 5

You can see that the EPROM addresses are half the file addresses and that each EPROM contains only even or odd file bytes.

Use the ADDR command to select modes:

```
27256>>ADDR EVEN  
27256>>P  
-EPROM1 contains only even bytes  
27256>>ADDR ODD  
27256>>P  
-EPROM2 contains only odd bytes
```

Example

You want to program an EPROM leaving the first 100 bytes blank. Use a BASE of negative 100. This will add 100 to each address before programming.

```
27128>>BASE -100  
27128>>P
```

Example

You have a 128k file with a starting address of E0000. You want to program it into four 27256's to be read in a 16-bit system:

```
Program first even/odd set  
27256>>BASE E0000  
-the base is subtracted first,  
27256>>ADDR EVEN  
-then the file is split even/odd  
27256>>P  
27256>>ADDR ODD  
27256>>P  
-Program second even/odd set  
27256>>BASE F0000  
-Each 27256 contains 8000 bytes  
-E0000 + 2*8000 = F0000  
27256>>ADDR EVEN  
27256>>P  
27256>>ADDR ODD  
27256>>P
```

The programmer ignores any data in the file that does not translate into a valid EPROM address.

If you have trouble be sure to check the number of bytes programmed, indicated when the PROGRAM command finishes. Zero bytes programmed means you told the EP-1 to ignore the whole file.

- When you want to leave the start of an EPROM blank, use a negative BASE.
- When you want to skip the start of a file, use a positive BASE.
- Intel, Motorola, and Tekhex files contain addresses generated by the assembler.
- Binary and Straight Hex files do not contain addresses, therefore, they always start at address 0. The addressing modes still work using these implicit addresses.
- Use STAT or BASE commands to determine what addressing mode is currently in effect.

APPENDIX G

EP.EXE PROGRAM

The EP.EXE communication program is provided with the EP-1 on a floppy disk. It is compatible with PCDOS and MSDOS V2.00 and later. The program can normally be invoked by typing EP 1 or EP 2. It has a number of other features described here.

 *Please read Chapter 2 before reading this appendix.*

The program normally communicates with a standard (IBM-like) serial port COM1: or COM2:. These are normally configured with interrupt 4 and 3, respectively. You can use the program with serial ports configured differently as noted below. Serial ports must use an NS8250, NS16450, or NS16550 chip or emulate these chips.

The program also expects to find an IBM-compatible video adapter. Use the -v flag if you are not getting video.

Synopsis

EP p [-Iint] [-Bbaud] [-V] [-Q] [-S] file...

p

P is a number specifying which serial port to use. To use standard COM ports, specify 1, 2, 3, or 4. The port addresses are read from the BIOS and interrupts default to 4, 3, 11, and 12, respectively. Otherwise, you can specify the hex address of the port (e.g. 2F8). In this case, you must specify the interrupt.

Iint

Specifies which interrupt to use. Be careful to specify the correct interrupt number, as the wrong number can crash your system. Interrupts 0-7 are the normal PC interrupts, while 8-15 are the extended interrupts on the AT.

Bbaud

Specifies what baud rate to use. Valid rates are 300, 600, 1200, 2400, 4800, 9600, 19200, 38400. The default rate is determined by your CPU speed.

-V

Tells the program to use BIOS video services. This is slower than the default.

-Q

Is normally used with batch mode operation. It forces the program to quit when an error occurs. The program returns the value 1. This option is especially useful in makefiles.

-S

Silent mode: nothing is sent to the screen. Use with -Q in batch mode.

file

Any DOS filename. This invokes batch mode operation<197>input is taken from the file instead of the keyboard. The special filename "CON:" will take input from the keyboard. When the user types QUIT, the next file in the list is used. If "CON:" is specified at the end of the command line, control returns to DOS.

BATCH MODE OPERATION

The programmer may be operated from a batch file, providing automated programming. You must prepare a data file to control the programmer. This data file will contain exactly what you would type from the keyboard to control the programmer.

Use a text editor to prepare the control file. The name of the control file is specified on the command line when invoking EP.EXE. Execution proceeds as if you were typing the commands in the control file. You may specify more than one file on the command line. When the commands are exhausted in one file, the next file is read. If you specify the special filename "CON:", input will be taken from the console; execution continues when the QUIT command is used.

ENVIRONMENT VARIABLE

You can set the environment variable EP1 to the default parameters for your system. EP.EXE will read arguments from the variable first, then the command line. For example, if you always use COM2: and the v parameter, you can use the DOS command:

EXAMPLE = SET EP1=2 -V

You can now start EP without any parameters:

EXAMPLE = C>>EP

You can place the SET command in your AUTOEXEC.BAT file so it will be executed every time you start your system.

- Press Control-S (^S) to pause the program at any time. Press any key to continue.
- Press Control-Break to return to DOS immediately.
- Do not type during read commands (RH, RI, RM, RT). Typing makes it more likely to lose characters.
- If you lose characters, you will get a warning message. You are using a baud rate that is faster than your computer can handle.
- Some load-and-stay-resident programs can slow down system interrupts, causing you to lose characters. If you suspect this, try booting from your ORIGINAL DOS diskette (with no

config.sys or autoexec.bat files). Now run EP.EXE. This insures no load-and-stay resident programs are in memory (because no config.sys file or autoexec.bat file placed them there).

- Version V3.xx of EP.EXE communications program fully supports the XMODEM and the CRC XMODEM communications file protocol. This protocol detects and corrects any errors in the file transfers between the host and EPROM Programmer

APPENDIX H

RS-232 CABLE

These cable specifications are meant to be used with IBM compatible computers in conjunction with the EP.EXE program. If you are not using this program, see APPENDIX I.

The IBM PC, XT, and most compatibles use a DB-25 for RS-232. Use the following cable or a straight 25 pin male-to-female cable:

The IBM AT uses a DB-9 connector for RS-232.

If you are not using EP.EXE and you want to use XON/XOFF handshaking, use this connection:

Other cable configurations can make the programmer appear to lock up.

If you are not using EP.EXE and you want to use hardware handshaking, you must connect to TxD, RxD, RTS, CTS, GND, and DTR. See Appendix I for more information.

DTR must be active to make the EP-1 operate.

APPENDIX I

Non-PCDOS OPERATION

If you are operating the EP-1 on a computer other than the PC, you need special instructions. EP.EXE is an ordinary communication program except that it is integrated with the EP-1 so it automatically asks you for filenames and reads/writes files.

You will need to use a general-purpose communication program (sometimes called a modem program). The program should have terminal emulation; it must be able to upload and download files in order to read and program. The program also needs handshaking (the ability to stop transmitting a file when the receiving end is busy).

HANDSHAKING

Handshaking is necessary to allow the programmer to control the flow of data while programming to prevent overflowing the input character buffer. Either hardware or software handshaking can be used.

Hardware handshaking uses the CTS and RTS lines. When the EP-1 cannot receive any more characters, it sets CTS low to stop the computer. CTS goes high when the EP-1 is ready for more characters. The EP-1 will not transmit characters if you put RTS low.

The programmer can also use XON/XOFF protocol. The programmer will send an XOFF (^S) when it can no longer accept characters, and an XON (^Q) when it is again ready to receive characters. If the programmer receives an XOFF, it ceases transmitting and resumes when it gets an XON. Even if you send the XOFF, the EP-1 can still generate XON and XOFF to control the computer.

Connect your computer to the EP-1 using an appropriate cable. See Appendix H.

- Both handshaking modes are in effect at all times. When the EP-1 receive buffer is close to full, it sets CTS low. If the characters continue to be received from the computer, it sends an XOFF. When the buffer again empties, it sends an XON first then sets CTS high again. If you use CTS, the XOFF/XON will never be transmitted. If you ignore CTS, you can just use XOFF/XON.

ESTABLISHING COMMUNICATIONS

The EP-1 uses 8 data bits, no parity bit, one stop bit. It will ignore parity, so you can also use 7 data bits, even or odd parity. The baud rate is automatically determined by the programmer when you sign on:

- activate DTR, RTS
- send a BREAK
- delay 300mS
- send a space character (20H)

- the programmer will respond with the sign on message.

See the ECHO, RH, PROGRAM commands for more information about using communication programs with the EP-1.

APPENDIX J

ERROR MESSAGES

ERROR MESSAGES GENERATED BY THE EP-1

All error messages generated by the EP-1 will sound the beeper (^G).

Argument Error

You have specified an invalid argument. Arguments are typed on the command line after the command name.

Bad Command

The command name was not recognized.

Cannot program this device.

The currently selected device is a ROM; it is not programmable. These parts are programmed by the factory when they are manufactured.

Cannot Protect this chip.

The PROTECT command does not apply to this chip type.

Cannot set page

The PAGE command does not apply to this chip type.

EP-1 BUFFER OVERFLOW--HANDSHAKING ERROR

This is a communications problem. The computer has sent the EP-1 data faster than the EP-1 can accept it. Check your cable using an ohmmeter and Appendix H or Appendix I.

Error reading hex--Ignored:

The line indicated is not a valid hex record. The line is ignored and programming continues.

ERROR: (AAAA) = RR SHOULD BE WW.

During programming: the location AAAA did not program correctly. The EP-1 programmed the data WW into that byte, but the chip now contains RR.

During verifying: the data (RR) read from address AAAA does not match the file's data (WW).

ERROR: (XXXX) = XX SHOULD BE XX; CHIP NOT ERASED PROPERLY.

Same as above, but the error probably resulted because the chip was not erased well enough before programming. See BLANK command, Appendix B.

Error: try again

Reenter the line again.

Hex record not recognized--Ignored:

The hex record is not a known type. It has been ignored.

No chip has been selected.

You must select which chip you want to use. See CHIP command.

This EPROM is not erased

At least one of the bytes in the EPROM is not FF (00 in the case of microcontrollers). See Appendix B.

Timeout

You have waited more than a minute to enter the next line. The programmer automatically cancels the command.

Too many arguments

You have specified too many arguments for this command. See manual page on this command.

Warning--Checksum Error:

The hex file you are using contains a checksum error on the line shown. Each line in the hex file has a checksum so the EP-1 can verify that the line is valid. This message warns you that the line has been changed (this might be an error if you did not intend to change the line). The error is ignored automatically; the data is assumed to be correct and is used to program the device. See RH command for more information.

ERROR MESSAGES GENERATED BY EP.EXE

Cannot access programmer.

The communications line is active but the EP-1 did not respond. Check cable and connections.

Cannot create

The file cannot be written. Check for write protect, disk full, or bad directory name.

Cannot open

The file does not exist.

Cannot write file.

Disk may be full.

Communications error

This may be caused by noise on the line or a defective serial port.

EP-1 not present.

The EP-1 is not connected to this port or the power is off.

Error detected... quitting

You have specified the -Q option (see Appendix G) and there has been some sort of error.

Hex transfer error

The conversion from straight hex to binary format encountered an error. Probably due to using too high a baud rate or a memory resident program.

NNNN is not a valid baud rate.

You have specified a baud rate that is not compatible with the EP-1. Valid rates are 300, 600, 1200, 2400, 4800, 9600, 19200, 38400.

Not receiving any data...

The read command is not receiving any data.

Not recognized:

You have specified an option that is not valid; see appendix G.

There is no port at NNNN.

The program did not find a properly installed serial port at the given address. See INSTALLATION section.

You have not selected a port.

You must specify which serial port the EP-1 is connected to. See INSTALLATION section.

You have selected a port, but no interrupt.

If you specify a nonstandard serial port, you must also specify which interrupt line it uses. See the instructions for your serial card.

You have selected an invalid interrupt.

Interrupt numbers range from 0 to 15.

You have selected interrupt N. Are you sure you want to use it?

Normally, this interrupt is reserved for some other purpose. Using it may cause your system to crash. These are the reserved interrupts: 3, 4, 5, 7, 10, 11, 12, 15.

WARNING: You are losing characters... you should use a lower baud rate

Warning: lost more characters... try a lower baud rate

The computer is not responding to interrupts quickly enough. Characters are being lost. This can be caused by:

- Using a baud rate that is faster than your computer can support.
- Typing while the EP-1 is sending characters.
- Terminate-and-stay-resident programs
- Programs that place the time on the screen
- Background communications programs
- Disk caching programs using extended memory
- Programs that pop up with special keystrokes

The last five problems are all load-and-stay-resident programs that use interrupts or disable interrupts. These programs can slow down interrupts so much that the EP-1 communications program cannot work properly. These programs are most often invoked in an AUTOEXEC.BAT or CONFIG.SYS file. You can either try using a slower baud rate (don't use the DOS mode command; see Appendix G) or try booting your computer without starting the program: boot from your ORIGINAL DOS diskette. Your DOS diskette should not have the files AUTOEXEC.BAT or CONFIG.SYS present.

APPENDIX K

TROUBLESHOOTING

Refer to Appendix J for a list of error messages and related information. Here are some common areas of trouble:

EP-1 WILL NOT SIGN ON

If you get an error message, look up the message in Appendix J. If the program locks up and leaves the cursor on the line below the DOS prompt you should suspect your serial cable. Here's what to look for:

Active LED flashes continuously

The TxD line (pin 2) is not making connection on the serial cable.

Active LED flashes once or twice

There are two possible problems: the interrupt jumper on the serial card is not set correctly or RxD line (pin 3) is not making connection on your serial cable.

Active LED does not flash

This indicates that the RTS (pin 4) or DTR (pin 20) lines are not making connection on the serial cable.

A CHIP WILL NOT PROGRAM CORRECTLY

Make sure the chip was erased properly before programming--see Appendix B. If you have trouble with a single chip, that part may be defective. If you have trouble with a number of parts from the same lot (check date code on chip), the whole lot may be bad, even if it is brand new. If you have trouble with several chips of the same type, contact BP Microsystems<197>there could be a problem with your programmer, a problem with the algorithm, or the manufacturer may have a new programming algorithm for that chip.

CHARACTERS ARE LOST IN COMMUNICATION

You may have a memory resident program that is slowing down the interrupts. Try formatting a new floppy and making it bootable (FORMAT A: /S). Copy EP.EXE onto the floppy (COPY EP.EXE A:). Now reset your system (Control-Alt-Del) and run EP as usual. If this clears up the problem, try to figure out what is causing the interference (look in CONFIG.SYS and AUTOEXEC.BAT).

If you are having problems with communications in general check the UART in your computer's serial port. This is typically a forty pin chip or a large PLCC type chip connecting to the RS-232 port in your computer.

National Semiconductor is the only manufacture that we recommend if you are having problems. Some second source chips will work at low baud rates or without interrupts but will not function at high baud rates with interrupts. New versions of these UARTs allow higher baud rates with fewer problems by offering FIFO mode operation. The EP.EXE program automatically sets the FIFO mode on if it detects this capability of the UART. Most UARTs in most computers work fine, but if you are having problems please refer to the table below.

PC or XT = NS8250B

AT or 386 = NS16450

ANY TYPE
(FIFO) = NS16550

APPENDIX L

SPECIFICATIONS

GENERAL

Architecture = Z-80 microprocessor controlled

Socket = gold Textool ZIF socket

Display = power, active, programming

Algorithms = fast (Intelligent), slow, paged fast, quick pulse, EEPROM

Self-test = automatic RAM, ROM, and communications test at power up

Manual test = pin drivers

Voltage-Reference = Stable I.C. reference; no pots or zeners

Size = 10 1/8" W x 6" D x 2 5/16" H

Construction = 3/32" aluminum

Weight = 4 lbs.

Power = 105-125 VAC (or 210-250VAC), 50-440Hz, 20 Watts, grounded US line cord

Environment = 0-40<198>C; 10-90% RH, non-condensing

COMMUNICATIONS

Type = RS-232 DCE

Connector = DB-25S

Baud Rates = 300; 600; 1200; 2400; 4800; 9600; 19,200; 38,400

Selection = Automatic, from a single space (ASCII 20H)

Format = 8 data bits, no parity, 1 stop bit

Handshaking = XON/XOFF or hardware

Buffer = 256 character receive buffer

Protocols = Xmodem or CRC Xmodem

APPENDIX M

LIMITED WARRANTY

BP Microsystems warrants this product against defects in material or workmanship, as follows:

For a period of one year from date of purchase, BP Microsystems will repair the defective product and provide new or rebuilt replacements at no charge.

After the one-year period, you must pay for all parts and labor.

This warranty does not cover any damage due to accident, misuse, abuse or negligence. This warranty is valid only in the United States.

You should retain your dated bill of sale as evidence of the date of purchase.

REPAIR OR REPLACEMENT AS PROVIDED UNDER THIS WARRANTY IS THE EXCLUSIVE REMEDY OF THE PURCHASER. BP MICROSYSTEMS SHALL NOT BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR BREACH OF ANY EXPRESS OR IMPLIED WARRANTY ON THIS PRODUCT. EXCEPT TO THE EXTENT PROHIBITED BY APPLICABLE LAW, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ON THIS PRODUCT IS LIMITED IN DURATION TO THE DURATION OF THIS WARRANTY.

Some states do not allow the exclusion or limitation of incidental or consequential damages, or allow limitations on how long an implied warranty lasts, so the above limitations or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights, which vary from state to state.