SDK-51 MCS™-51 SYSTEM DESIGN KIT USER'S GUIDE

Manual Order No.: 121588-002

intel®

SDK-51 MCS™-51 SYSTEM DESIGN KIT USER'S GUIDE

Manual Order No.: 121588-002

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

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue	2/81
-002	Minor Revisions	8/81

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

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

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP
CREDIT
i
ICE
iCS
im
INSITE
Intel
Intel
Intelevision
Intellec
iRMX
iSBC
iSBX
Library Manager
MCS
Megachassis
Micromainframe
Micromap
Multibus
Multimodule
Plug-A-Bubble
PROMPT
Promware
RMX/80
System 2000
UPI
μScope

and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or RMX and a numerical suffix.

PREFACE

This manual contains operating instructions, functional description, and application programming examples for the MCS-51 System Design Kit (SDK-51). The manual assumes the user has completed the assembly and initial checkout of the kit as described in the following manual included with the kit:

SDK-51 MCS™-51 System Design Kit Assembly Manual, order number 121589.

In addition, the kit contains the following documents to supplement the discussions in the user's manual:

SDK-51 MCS™-51 System Design Kit Monitor Listing Manual, order number 121590.

SDK-51 CPU Board Schematics, part number 162072.

SDK-51 Display Board Schematic, part number 162075.

MCS™-51 Family of Single-Chip Microcomputers User's Manual, order number 121517.

MCS™-51 Macro Assembler User's Guide, order number 9800937.

The following manuals are not included in the kit, but are recommended for reference:

Intel Component Data Catalog.

UPI-41A User's Manual, order number 9800504.

SERVICE INFORMATION

If, following assembly, you cannot get your kit to operate satisfactorily, the Intel Technical Support Center "Service Hotline" is available for assistance. This service is provided during the hours of 8AM to 5PM (Mountain Time), Monday through Friday. The toll-free Hotline telephone numbers are:

All U.S. locations except Alaska, Arizona, and Hawaii:
(800) 528-0595

All other locations:
(602) 869-4600

TWX Number:
910 - 951-1330

The Hotline is intended expressly to help you get your kit running and is *not* intended to assist you in circuit designs or applications. Telephone assistance is limited to *one* call per problem. If a problem cannot be remedied over the telephone, you may, at your discretion, return your assembled kit to Intel for repair. To return your kit, a Return Authorization Number *must be obtained* from the Technical Support Center prior to sending in your kit. Also, either a purchase order for the repairs must be furnished to the center or a money order (no personal checks please) must be included with the kit being returned. Repairs resulting from defective components supplied with your kit will be done at no charge, and all prepayments will be refunded. Repairs necessitated as a result of customer error, damage or misuse will be billed at a fixed, flat-rate charge which will be quoted by the Technical Support Center.

NOTE

The Technical Support Center will not repair an SDK-51 Kit that has been modified and, when circuitry has been added to the user design area, may request that the circuitry be disconnected prior to submitting the kit to the center for repair.

CONTENTS

CHAPTER 1
GENERAL INFORMATION PAGE

CHAPTER 2
OPERATING INSTRUCTIONS

CHAPTER 3
FUNCTIONAL DESCRIPTION PAGE

CHAPTER 4
APPLICATIONS

APPENDIX A
TELETYPEWRITER MODIFICATIONS

APPENDIX B
ERROR MESSAGES

ILLUSTRATIONS

TABLES

CHAPTER 1
GENERAL INFORMATION

Introduction

This manual contains operating instructions and circuit descriptions for the SDK-51 System Design Kit. It assumes you have completed the assembly and initial checkout of the SDK-51 as described in the Assembly Manual for the product (see Preface for reference). The User's Guide is organized into chapters as follows:

Chapter 1 introduces the SDK-51 concept and gives specifications for the basic configuration.

Chapter 2 contains procedures for operating the on-board features of the SDK-51. The chapter includes an overview of system operating modes, a description of the keyboard and display, and an explanation of the notation for command formats used in the chapter. Operating procedures include commands to be entered at the keyboard and configuration and interface considerations where they apply. The operations are grouped into four categories: Power Up, Command Entry, and Reset; Program Execution Commands; Memory Access Commands; and Input/Output Operations.

Chapter 3 provides background operating theory for the major functional blocks of the SDK-51. The chapter includes a functional block diagram keyed to the schematic drawings.

Chapter 4 gives information on programming the 8031 microprocessor, emphasizing the use of the on-board facilities. The chapter includes suggestions for accessing the keyboard and display, operating the 8031's on-chip serial port through the SDK-51's serial I/O interface, using the on-board parallel I/O interface, scanning the auxiliary keypad, installing circuits in the prototyping area provided on the board, and expanding the physical memory beyond the space provided on the board.

Appendix A describes the modification of a teletypewriter for use with the SDK-51.

Appendix B lists the error messages with brief explanations.

The Intel 8051 Microcontroller Family

The 8051 family has three members: the 8031, the 8051, and the 8751. The 8031 has no on-chip program memory; execution is from external program memory. The SDK-51 controller is an 8031. The 8051 has 4096 (4K) bytes of factory-masked ROM, and the 8751 has 4K bytes of EPROM.

NOTE

> In this manual, the term 8051 refers to the members of the 8051 family in general, while the term 8031 refers to the controller on the SDK-51 in particular.

The Intel 8051 Microcontroller is truly a "controller-on-a-chip". In a 40-pin package the 8051 combines:

12-MHz CPU;

Non-volatile 4Kx8 read-only program memory (8051 and 8751);

128x8 read-write data memory;

32 I/O lines;

Two 16-bit timer/event counters;

Five interrupts at two priority levels;

On-chip oscillator and clock circuits;

Serial I/O channel for multiprocessor communications, I/O expansion, or full duplex UART.

With the addition of a +5 volt supply and a timing crystal, the 8051 Microcontroller can perform all the computing, controlling and interfacing of a general purpose microcomputer.

Please see the *MCS-51™ Family of Single-Chip Microcomputers User's Manual* for details. The Preface contains a full reference to that manual.

The SDK-51 Concept

The SDK-51 is tool for evaluating the 8051 microcontroller and for illustrating types of applications for the 8051. It is furnished in kit form for low cost and to give hands-on experience with 8051 designs.

The SDK-51 (see Figure 1-1) includes a 60-character ASCII keyboard, a 24-character alphanumeric LED display, space for up to 16K bytes of RAM, space for up to 8K bytes of ROM, a parallel I/O interface with 22 I/O port lines, a serial I/O interface, and a cassette I/O interface,.

The address and data bus controllers separate the 8051 microcontroller's multiplexed address/data bus, creating a 16-bit address bus and an 8-bit data bus.

The Universal Peripheral Interface (UPI) controls the serial I/O interface, keyboard, display, and cassette interface, through the 8-bit UPI bus and other control signals.

The monitor program for the SDK-51 is contained in 8K bytes of EPROM. The monitor is located at the top of external memory to allow the lower part of memory to be used for program development.

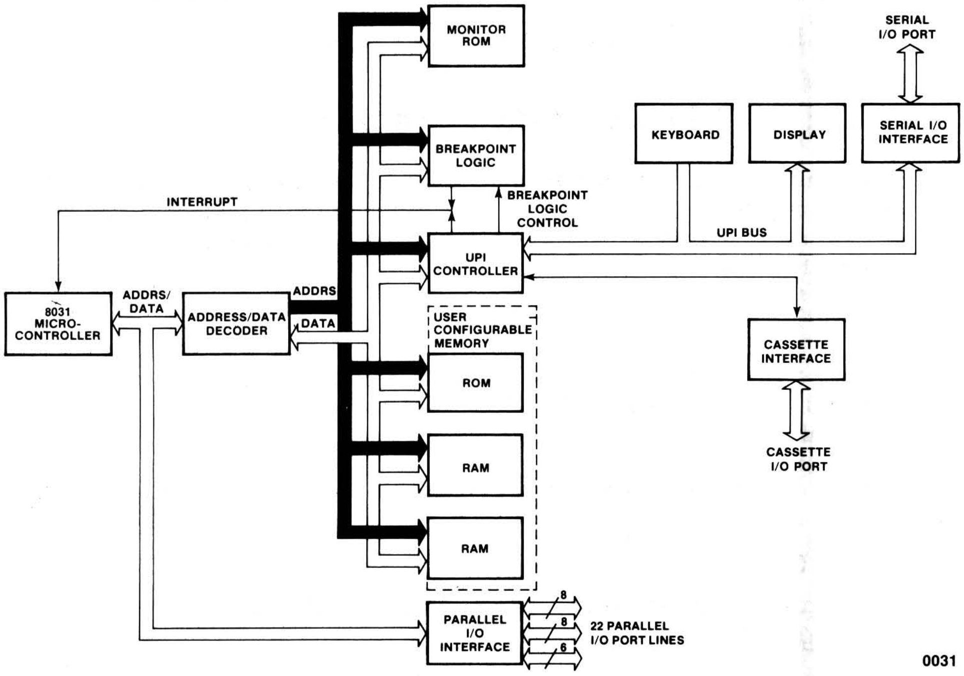


Figure 1-1. SDK-51 Block Diagram

The breakpoint logic circuit allows external interrupts to the 8051 microcontroller to be generated on specified data or program memory addresses. The breakpoint logic also interrupts the controller when the UPI requires the attention of the 8031.

SDK-51 Specifications

Table 1-1 gives specifications for the SDK-51 in the basic configuration furnished in the kit.

Table 1-1. SDK-51 Specifications

MICROPROCESSOR	
CPU	Intel 8031
Crystal	12 MHz
T_{CYC}	1 microsecond
MEMORY	
System Monitor ROM	8K bytes included (two 2732A EPROMs)
System Breakpoint RAM	4K bits included; provision for additional 4K bits
User-configurable RAM	1K byte supplied (two 2114s); provision for up to 16K bytes
User-configurable ROM	Provision for up to 8K bytes
On-chip Memory	Full read/write access to on-chip RAM and Register memory from SDK-51 console and from user program
ADDRESS SPACE	
User configurable	0000H through 7FFFH
System I/O	8000H through DFFFH
Monitor	E000H through FFFFH
POWER REQUIREMENTS	
V_{CC}	+5 VDC (±5%), 3 amperes (minimum configuration, 1K byte of RAM)
V_{TTY}	+12 VDC and -12 VDC, 0.1 ampere (required for serial I/O operation)
DIMENSIONS	
Length	13.75 inches
Width	12 inches
Height	4 inches (with display mounted)
OPERATING ENVIRONMENT	
Operating temperature	0 to 50 degrees Celsius
Operating humidity	0 to 90%, non-condensing
INTERFACE SIGNAL LEVELS	
Microcontroller Bus	All signals TTL compatible
Parallel I/O	All signals TTL compatible
UPI Bus	All signals TTL compatible
Serial I/O	RS-232 or current loop
Cassette Interface	Suitable for microphone or line input to audio cassette recorder
INTERRUPTS	
Interrupt 0	Reserved for system use.
Interrupt 1	User selectable; may be connected to INTR key on keyboard

CHAPTER 2
OPERATING INSTRUCTIONS

Introduction

This chapter describes procedures for entering, executing, and storing user programs on the SDK-51. Each procedure includes the monitor commands to be keyed in by the user, and, where applicable, configuration steps and interface considerations.

The SDK-51 operations are controlled by the monitor program stored in 8K (8192) bytes of read only memory (ROM) at the upper end of SDK-51 on-board memory (refer to discussion of SDK-51 memory map in Chapter 3). The system goes to the beginning of the monitor program whenever power is turned on or when both RESET buttons are pressed. The monitor program allows the user to perform the following operations:

- Communicate with the SDK-51 using the on-board keyboard and display.
- Execute user programs in real-time or single-step.
- Set breakpoints on program or data addresses, and display the cause of the most recent break.
- Assemble individual MCS-51 assembly language instructions into memory, and disassemble memory into assembler equivalents, line-by-line.
- Write-protect the portion of on-board memory designated by the user as program memory rather than data memory.
- Examine and modify memory locations, registers and bits in SDK-51 on-board program/data memory and in the 8031's on-chip data and register memories.
- Upload and download programs from an INTELLEC® development system.
- Read and write data to an audio cassette recorder.
- List the output of the SDK-51 to a printer or other external device.

The SDK-51 command language is modeled after that of the ICE™-51 In-Circuit Emulator, allowing a minimum of relearning to use the emulator system.

NOTE

Chapter 4, Developing Applications, contains information on developing user programs to run on the SDK-51 that most users will find essential.

Keyboard

The SDK-51 typewriter-like keyboard allows the user to enter commands and data. It includes 59 characters plus space key, shift keys, CNTRL key and RUBOUT key for line-by-line editing, a tab key, and an escape key (see Figure 2-1). An auxiliary keypad contains the two system RESET keys and an INTR key (the INTR key requires user configuration as described in chapter 4).

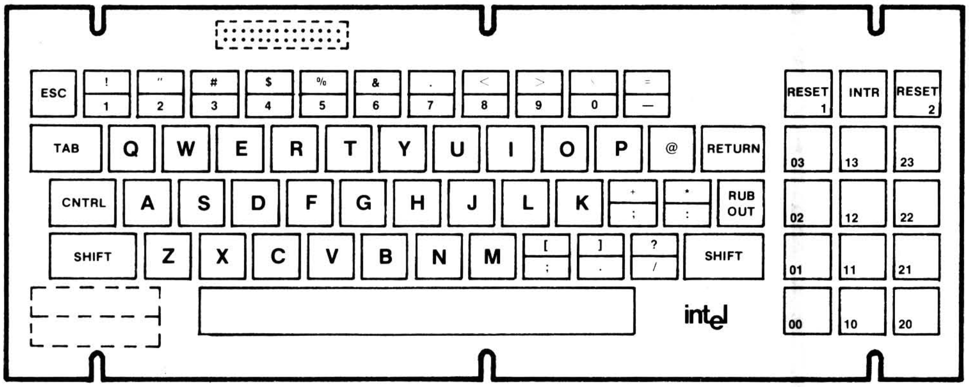


Figure 2-1. Keyboard Layout

Pressing any of the alphanumeric keys results in the corresponding letter, number or symbol being read out on the LED display. The functions of the other keys are as follows:

RETURN Terminates command entry. Clears display or error message so that next command can be entered.

ESC (Escape) Terminates current monitor operation and returns monitor program control to the interrogation mode. Deletes any line being entered on the display. (Exceptions: one ESC pauses autostepping, two ESC's are required to abort autostepping; ESC key has no effect in cassette I/O LOAD or SAVE commands.)

SHIFT Selects the upper characters for keys with two functions.

TAB Aligns cursor on four-character boundaries.

RESET (two) Pressing the two RESET keys simultaneously halts the operation of the SDK-51 at any stage of operation and returns the system to its start-up (initialized) state. Pressing just one of the RESET keys has no effect. See Power Up, Command Entry, and Reset later in this chapter for details.

RUBOUT Deletes the rightmost character on the display each time it is pressed. If the display is empty (that is, no characters after the prompt), RUBOUT has no effect.

Display

The SDK-51 display consists of a 24-character LED readout. Each character of the display is capable of displaying 64 different characters, including the space character; see Table 2-1. Table 2-1 also shows the six-bit value that the UPI-41A controller sends on the UPIBUS to the display. When characters are transmitted between the UPI-41A and the 8031 controller, the 7-bit ASCII values are used; to display the four characters without corresponding keys, program the 8031 to send the 7-bit code (see Chapter 4 for details).

When commands or data are entered from the keyboard, characters are displayed left to right starting with the leftmost LED readout (similar to the entry of data on a CRT terminal). Only 24 characters can be entered (23 command characters plus a RETURN, which does not count as a character); any characters entered following the 23rd are ignored. Note that the cursor overlays the 24th character position, but after 23 characters, only RETURN, RUBOUT, and ESC are valid.

Table 2-1. Display Character Set

6-Bit UPIBUS Value (Hex)	Display Character	Keyboard Character
00H	@	@
01H	A	A
02H	B	B
03H	C	C
04H	D	D
05H	E	E
06H	F	F
07H	G	G
08H	H	H
09H	I	I
0AH	J	J
0BH	K	K
0CH	L	L
0DH	M	M
0EH	N	N
0FH	O	O
10H	P	P
11H	Q	Q
12H	R	R
13H	S	S
14H	T	T
15H	U	U
16H	V	V
17H	W	W

Table 2-1. Display Character Set (Continued)

6-Bit UPIBUS Value (Hex)	Display Character	Keyboard Character
18H	X	X
19H	Y	Y
1AH	Z	Z
1BH	[	[
1CH	\	\
1DH	]	]
1EH	↗	None
1FH	←	None
20H		Space bar
21H	!	!
22H	"	"
23H	#	#
24H	$	$
25H	%	%
26H	&	&
27H	'	'
28H	<	<
29H	>	>
2AH	*	*
2BH	+	+
2CH	,	,
2DH	-	-
2EH	.	.
2FH	/	/

Table 2-1. Display Character Set (Continued)

6-Bit UPIBUS Value (Hex)	Display Character	Keyboard Character
30H	0	0
31H	1	1
32H	2	2
33H	3	3
34H	4	4
35H	5	5
36H	6	6
37H	7	7
38H	8	8
39H	9	9
3AH	:	:
3BH	;	;
3CH	<	None
3DH	=	=
3EH	>	None
3FH	?	?

Modes of Operation

The monitor has several modes of operation: interrogation mode (the default), assembler mode, continuation mode, real-time execution mode, and single-step execution mode.

The monitor enters the interrogation mode at power-up and following any system reset. A flashing hyphen prompt is displayed at the left margin to indicate that the system is in interrogation mode and is ready to accept a command. All commands are entered through interrogation mode.

The assembler mode allows the user to enter instructions in 8051 assembly language and assemble the instructions directly into on-board memory. Assembler mode is initiated by the ASM command, and is terminated by pressing RETURN instead of entering an instruction. The user can also disassemble on-board memory into 8051 instruction mnemonics.

Continuation mode allows the user to enter a list of values for setting memory contents without repeating the memory type keyword. See Memory Access Commands in this chapter for details.

The real-time execution mode allows you to run the user code stored in program memory. Execution begins when the user enters a GO command in interrogation mode. Real-time execution can be controlled by breakpoints set by the user. If breakpoints are used and a breakpoint is encountered, the program halts after executing the instruction that contained the breakpoint address, then returns to the interrogation mode. If breakpoints are not used (or encountered), the program runs until the user terminates execution.

The single step execution mode allows you to run the user program one instruction at a time, breaking between steps. The user can specify the stepping rate. Between steps the system displays register values.

Keywords and Numeric Entries in Commands

An SDK-51 command begins with a command keyword that indicates the type of operation to be executed; the command word may be followed by one or more qualifiers that limit or define the parameters of the operation. For example, the command:

```
BAUD = 600
```

sets the SDK-51 serial I/O transfer rate to 600 baud. In this command, BAUD is the command keyword, = is the assignment operator, and 600 is the baud rate parameter.

The term *keyword* refers to words with fixed spellings that the SDK-51 monitor interprets in defined ways. To save keystrokes and to allow you to enter the extended versions of some commands within the 24-character display, most keywords can be abbreviated to one, two, or three characters. Some of the command qualifiers are also keywords (for example, DATA and RESET). Table 2-2 summarizes the command keywords and other keywords used in the SDK-51 command language.

NOTE

The MCS-51 assembly language mnemonics such as MOV and CLR are also keywords, since their spelling and interpretation are system-defined. The mnemonics are not listed in Table 2-2, since they are defined in the MCS-51 assembler and component manuals. See the Preface for references.

NOTE

The minimum abbreviations of some SDK-51 keywords differ from those used for the same keywords in the ICE-51 command language. The three-character abbreviations used in ICE-51 also work with the SDK-51.

Command qualifiers can be numeric entries instead of keywords. Numeric entries are of the following kinds:

digit	Any hexadecimal digit from 0 to F
decimal-digit	Any decimal digit from 0 to 9 (for the STEP command)
byte	Hexadecimal number up to two digits (three digits when leading zero required)

file-number A cassette file number (for the LOAD and SAVE commands) consisting of up to four hexadecimal digits (up to five digits when leading zero required)

address Hexadecimal number up to four digits (five digits when leading zero required)

partition *address* [TO *address*]

With one exception, the monitor assumes hexadecimal (base sixteen) for numeric entries (the exception is the rate parameter for the STEP command, where base ten is assumed). In the text of this manual, hexadecimal numbers are identified with an "H" suffix; for example: 0134H. The monitor does not require you to use the H suffix; however, it may be entered without causing an error.

A *partition* contains one or more addresses. The second address (after the keyword TO) must be equal to or greater than the first address, or an error occurs.

NOTE

The Monitor requires all numbers to begin with a numeric digit 0 through 9. Hexadecimal numbers must be preceded by a zero unless the first digit is a numeral. Example: "7F" is acceptable, but "F7" must be entered "0F7".

Table 2-2. Keywords and Abbreviations

Keyword	Minimum Abbreviation
ABR	ABR
ACC	ACC
ASM	AS
B	B
BAUD	BA
BR	BR
CAUSE	CA
CBYTE	CB
DASM	D
DATA	DAT
DBYTE	DB
DOWNLOAD	DO
DPTR	DP
FROM	F
GO	G
LIST	LI
LOAD	LO
ON	ON
OR	OR
ORG	ORG
PC	PC
PROGRAM	PR
PSW	PS
RBIT	RBI
RBYTE	RBY
RESET	RES
SAVE	SA
SP	SP
STEP	ST
TILL	T
TM0	TM0
TM1	TM1
TO	TO
TOP	TOP
UPLOAD	UP
XBYTE	XB

Command Format Notation

The syntax or general format of each command is described with a simple notation system. The command notation shows what command keywords to use, indicates parts of the command that can be omitted or included at the user's option and shows the places in the command where the user has a choice of several kinds of entries. In the notation:

- Keywords are shown in ALL CAPS.
- Numeric entries are shown in *lower case italics or underlined.*
- Required entries are shown without any enclosures (brackets or braces), for example:

  UPLOAD *partition*

  Both the command keyword UPLOAD and the numeric entry *partition* are required in this command.
- Optional entries are enclosed in square brackets; for example:

  ASM [ORG *address*]

  The keyword ASM is required, the entry ORG *address* is optional.
- Entries that can be repeated at user option are enclosed in brackets and followed by an ellipsis (...); for example:

  ABR = *partition* [, *partition*] ...

  The first partition is required; thereafter the user can add partitions separated by commas to the end of the 24 character display.
- A choice of entries is shown by a vertical list (or "menu") of the entries enclosed in braces or brackets. A menu enclosed in square brackets means "select none or one", for example:

  $$\text{GO [FROM } \textit{address}\text{]} \begin{bmatrix} \text{FOREVER} \\ \text{TILL PROGRAM} \\ \text{TILL DATA} \\ \text{TILL PROGRAM OR DATA} \end{bmatrix}$$

  In this command, All the entries after GO are optional. Of the menu in brackets, select either none or one.

  A menu enclosed in curved braces means "select one and only one", for example:

  $$\text{LIST} = \begin{Bmatrix} \text{ON} \\ \text{RESET} \end{Bmatrix}$$

  An entry is required following "LIST =". Of the two possible entries in the brackets, you must select one and only one.

Display Commands and Change Commands

The equals sign (=) is the assignment operator used to change values. When a command keyword or phrase is entered without an equals sign, it indicates that the current contents of the referenced object (memory location, register, breakpoint memory) are to be displayed. When a command word or phrase is followed by an equals sign, it means that the contents of the referenced object are to be set to the values of the numeric entry or entries on the right of the sign. For example:

```
XBYTE 60F7
```

causes the information stored in memory location 60F7H to be displayed, whereas:

```
XBYTE 60F7 = 11
```

causes memory location 60F7H to be set to 11H.

Overview of Commands and Operations

The SDK-51 monitor commands and related operations are grouped for discussion as follows:

- Power-Up, Command Entry, and Reset Operations
- Program Execution Commands

  Set and display breakpoints (BR and ABR Commands)

  Real-time execution (GO Command)

  Display cause of last break in execution (CAUSE Command)

  Single-step execution (STEP Command)

- Memory Access Commands

  Display and change program and data memory (CBYTE, DBYTE, RBYTE, XBYTE, and RBIT Commands)

  Assemble and disassemble instructions (ASM and DASM Commands)

  Set and display the highest address in the write-protected portion of program memory (TOP Command)

- Serial I/O Interface Operations

  Jumper configuration and cable connection for serial I/O operations

  Setting baud rate (BAUDD Command)

  Development system interface (UPLOAD and DOWNLOAD Commands)

  List output to external device (LIST Command)

- Audio Cassette Interface Operations

  Cassette interface connections

  SAVE and LOAD operations

Power-Up, Command Entry, and Reset

Use the following procedures to apply power to the SDK-51 board, enter commands, and reset the system.

Power-Up

1. Connect the red wire on the SDK-51 power cable (pin 1) to the +5 VDC terminal on the power supply; connect the black wire adjacent (pin 2) to the 5 volt return terminal on the power supply.

CAUTION

Be sure that the + and - sides of the power supply are connected to the proper SDK-51 power cable terminals: +5 volts (red); 5 volt return (black). Applying reverse polarity to the power input terminals will damage system components.

2. If serial I/O operations are desired, connect the blue wire on the power cable (pin 4) to +12 VDC, and the yellow wire (pin 6) to -12 VDC. Connect the remaining black wires to ±12 volt return.
3. Place shorting plugs on the memory configuration jumpers to assign memory address blocks to the memory areas installed on the board and to the breakpoint memory (see Table 2-3).

4. If Memory 2 (ROM) is installed, check that jumper area U68 has been connected correctly for the type of ROM used . Jumper socket U68 allows jumper wires to be installed as required for the type of ROM devices used. Table 3-3 (in Chapter 3) shows the required jumper configurations and maximum crystal frequencies for five different types of ROMs.
5. Serial I/O operations require additional jumper settings, as discussed later in this chapter.
6. Connect the power cable and apply power to the SDK-51 board.
7. The message SDK-51 MONITOR VER. nnnn appears on the display. The SDK-51 is reset, program control goes to the beginning of the SDK-51 monitor program.
8. Press the RETURN key. The monitor program enters its interrogation mode; the prompt (flashing hyphen '-') is displayed in the leftmost display character position.

Table 2-3. Memory Configuration Jumpers

	Address Blocks			
Memory Areas	0000H-1FFFH	2000H-3FFFH	4000H-5FFFH	6000H-7FFFH
Breakpoint Memory	W20*	W24	W28	W32
Memory 2 (ROM)	W21	W25	W29*	W33
Memory 1 (RAM)	W22	W26*	W30	W34
Memory 0 (RAM)	W23*	W27	W31	W35

*NOTE: W20, W23, W26, and W29 are the standard settings for the minimum configuration supplied in the kit.

Command Entry

Enter a command of up to 23 characters from the keyboard, followed by a RETURN. If the command is in error, an error message "ERR = XX" is displayed. Refer to Appendix B for details on the error messages.

Once the command has been executed, the prompt is displayed, indicating that the system is ready for the next command.

To correct a command while it is being entered (before pressing RETURN), press the RUBOUT key to delete characters from the right then enter the desired characters.

To abort a command while it is being entered, press the ESC key. The display is cleared, and the prompt is displayed.

To abort a command while it is being executed, press the ESC key. The command halts *where it happens to be.* If the aborted command is a GO command, execution of the user program halts after executing the current instruction.

NOTE

The ESC key has no effect on the LOAD, SAVE, UPLOAD, or DOWNLOAD operations. To abort these data transfer operations, press the RESET keys as described in the next section. The ESC key pauses autostepping mode; see STEP command later in this chapter for details.

Reset

To reset the system to its initial state as after power-on, press both RESET keys at the same time. Press RETURN to clear the sign-on message and obtain the prompt.

During the reset process, the 8031 microcontroller is reset to the status specified in the MCS-51 User's Manual; on-chip data RAM is affected by the Monitor during reset, and is thus indeterminate after reset. The 8041A UPI, 8155-2 Parallel I/O Interface and breakpoint logic are reset. User-configurable memory is not affected.

At reset, the TOP address set to 0000H, the serial I/O baud rate is set to 2400, and LIST mode is set to RESET.

Program Execution Commands

The following commands are used to control the execution of user programs that have already been written. The BR and ABR commands set breakpoint addresses. The GO and STEP commands cause the system to enter execution mode from interrogation mode. The CAUSE command displays the cause of the last break in execution.

Breakpoint Commands

Function

Breakpoint commands BR and ABR set bits in breakpoint memory corresponding to addresses in an 8K byte block of user-configurable memory selected by the user. Breakpoints are enabled as program or data addresses with the GO command (next section). When a breakpoint address is executed, the system interrupts the 8031 and halts processing.

The BR command first resets the breakpoint RAM, clearing any previous match values, then sets the specified address values. BR can also be used to display the addresses in breakpoint memory where breakpoints have been set.

The ABR command sets breakpoints as well, but does not reset the breakpoint RAM, allowing breakpoint combinations to accumulate in breakpoint memory.

Formats

BR

BR = RESET

BR = *partition* [, *partition*] ...

ABR = *partition* [, *partition*] ...

Display Breakpoints

To display the match addresses currently set:

BR

Individual match addresses are displayed one address at a time. If a partition (two or more adjacent addresses) is set, the display has the format "*address* TO *address*". If more than one address or partition is set (not adjacent), the lowest address is displayed, followed by a flashing comma; press RETURN to display the remaining addresses one at a time. If no addresses are set, "BR=RESE" is displayed.

NOTE

> Breakpoint displays are always in the range from 0000H through 1FFFH. If the memory configuration jumpers are set to select a higher range, add 2000H, 4000H, or 6000H to the addresses displayed to obtain the actual breakpoints.

Reset Breakpoints

To clear the breakpoint memory, the command is:

```
BR = RESET
```

Set Breakpoints (With Reset)

To set one or more match addresses, or partitions of match addresses, (after first resetting any addresses already set), the format is:

BR = *partition* [, *partition*] ...

For example:

```
BR = 80, 90, 100 TO 110
```

The BR command clears any breakpoints previously set, then sets new breakpoints at addresses 80H, 90H, and 100H through 110H.

Add Breakpoints (Without Reset)

To add one or more match addresses or partitions of addresses (without clearing any previously set addresses), the format is:

ABR = *partition* [, *partition*]

For example:

```
ABR = 205, 280
```

The ABR command adds match addresses 205H and 280H to any breakpoints already set.

Operation

With the minimum SDK-51 configuration supplied with the kit, the breakpoint RAM is 4K by 1 bit. In this configuration, the highest breakpoint address (relative to the beginning of the 8K block) is 0FFFH. An additional 4K by 1 bit of RAM can be added, allowing breakpoints to be set on any address within an 8K byte block of user configurable memory; memory configuration jumpers specify which 8K block the breakpoint RAM represents. A reference to non-existent breakpoint memory is ignored (the BR command still clears the locations in breakpoint RAM not selected as breakpoints).

The user-configurable memory can contain both the user program instructions (opcodes and operands) and data segments (e.g., tables, arrays, text). A given breakpoint address must be one of the following:

- The address of an opcode in program memory (that is, the first address in an instruction). User program memory access involves the PSEN/ signal from the

8031 controller; see User Program Memory later in this chapter and Breakpoint Control in Chapter 4 for additional detail.

- The address of a byte of data in a MOVX instruction. Data memory access involves the RD/ and WR/ signals from the 8031 controller; see External Data Memory later in this chapter and Breakpoint Control in Chapter 4 for additional detail.

To break on an opcode, first set the opcode address in breakpoint memory (with a BR or ABR command), then enable program breakpoints (with a GO command).

To break on a data address, first set the address in breakpoint memory, then enable data breakpoints (with a GO command). The GO command is discussed in the next section.

Initially and after a reset, the breakpoint memory is cleared (no breakpoints are set), and breakpoints are disabled.

CAUTION

When a breakpoint occurs, the system writes four bytes onto the stack. The stack pointer is saved and restored, but you must allow for the four extra bytes in assigning space for your stack. Stack overflow (SP greater than 7FH) disrupts system operation.

NOTE

Since the system uses the interrupt structure to handle breakpoints (including single stepping), the following restrictions on breaking within interrupt routines apply:

1. The user program may not use or disable interrupt 0. See Interrupt Considerations in Chapter 4 for details.
2. If your program uses high-priority interrupts, system features are disabled while those interrupts are in progress.
3. If your program uses high-priority interrupts, breakpoints should not be used within the high-priority interrupt routine (FORCENOP circuit changes the user program).
4. After breaking in a low-priority interrupt routine, the Interrupt In Progress flag remains set until cleared by a RETI instruction. For proper operation, resume execution where it left off in the interrupt routine, or clear the flag by executing a RETI instruction before resuming execution at the outer level.

GO Command

Function

The GO command initiates program execution at real time (12 MHz crystal, 1 microsecond instruction cycle). The GO command can also include the starting address, and can enable or disable program and data breakpoints.

Format

```
                      ┌                        ┐
                      │ FOREVER                │
GO [FROM address]     │ TILL PROGRAM           │
                      │ TILL DATA              │
                      │ TILL PROGRAM OR DATA   │
                      └                        ┘
```

Begin Real-Time Execution

To begin real-time execution of the user program beginning with the instruction currently addressed by the program counter, the command is:

```
GO
```

When the GO command is executed, the SDK-51 enters execution mode. During program execution, the prompt is suppressed and the display contains the message EXECUTION BEGUN. Execution continues until one of the following occurs:

The user presses the ESC key.

A breakpoint is encountered (applies only when breakpoints are enabled)

The program attempts to write to an address lower than the specified top of program memory (TOP). See the TOP command for details.

The program attempts to execute across location 03H. This location is reserved for system operation.

After execution breaks, the message EXECUTION HALTED PC=*nnnn* appears on the display (*nnnn* is the value of the program counter after the last instruction executed). Enter any command to clear the display. The last condition, execution across location 03H, produces an error message instead of EXECUTION HALTED. See Appendix B for details.

When the GO command is entered without any modifiers, the following default conditions apply:

- The system uses the current program counter address as the start address.
- If program breakpoints or data breakpoints have been enabled by a previous GO command (and have not been disabled), they remain in effect for the current GO command.

Begin Execution, Specify Starting Address

To begin real-time execution at a specific starting address, the format is:

GO FROM *address*

For example, to begin at address 0100H:

```
GO FROM 100
```

This is equivalent to the commands:

```
PC = 100
GO
```

Begin Execution, Disable Breakpoints

To begin real-time execution and disable any previously enabled breakpoint types (program or data), the format is:

GO [FROM *address*] FOREVER

Examples:

```
GO FOREVER
GO FROM 100 FOREVER
```

Both examples disable breakpoints. Note that breakpoint memory is not affected; any addresses set remain set. When disabled, breakpoint address matches just have no effect on execution. The first example starts from the current program counter, while the second specifies address 0100H as the start address.

Breakpoints are disabled at power-on and after reset.

Begin Execution, Enable Program Breakpoints

To begin real-time execution and enable the breakpoint addresses in breakpoint memory that correspond to instruction addresses, the format is:

```
GO [FROM address] TILL PROGRAM
```

Examples:

```
GO TILL PROGRAM
GO FROM 100 TILL PROG          ; Note abbreviation to PROG
                               ; to fit in 23 characters
```

Both examples enable all addresses set in breakpoint memory. If an opcode is fetched from any of the addresses set in breakpint memory, execution breaks after completing that instruction. Operand fetches including data memory (MOVX) fetches do not break execution, even if the operand or data address is set in breakpoint memory; the user must ensure that the program addresses set in breakpoint memory are opcode addresses, not operand addresses.

The entry "TILL PROGRAM" implicitly disables data breakpoints; see below for ways to have both program and data breakpoints enabled.

Start Execution, Enable Data Breakpoints

To begin real-time execution and enable accesses to data addresses to break execution, the format is:

```
GO [FROM address] TILL DATA
```

Examples:

```
GO TILL DATA
GO FROM 100 TILL DATA
```

Both examples enable the data addresses that are set in breakpoint memory. A read or write to any of these addresses causes execution to break after that instruction. The entry "TILL DATA" implicitly disables program breakpoints.

Start Execution, Enable Program and Data Breakpoints

To begin real-time execution and enable both program and data breakpoints, the format is:

```
GO [FROM address] TILL PROGRAM OR DATA
```

Examples:

```
GO TILL PROGRAM OR DATA
G F 100 T PRO OR DAT
```

Both examples enable both program and data addresses to break execution. Note that the second example contains abbreviations rather than the full keywords (G for GO, F for FROM, T for TILL, PRO for PROGRAM, and DAT for DATA). The abbreviations are required to fit the command within 23 characters plus the RETURN.

Operation

With the GO command, the user has the option of specifying the starting address or using the current program counter address as the default.

Similarly, the user has the option of specifying the types of breakpoint to be enabled, or using the types previously enabled as the default. When program or data breakpoints have been enabled, they remain enabled until they are disabled.

Note that the setting of breakpoint memory is independent of the type of breakpoints enabled. This feature allows the user to set breakpoint addresses without starting execution, to halt execution on program breakpoints only, to halt on data breakpoints only, to halt on either program or data breakpoints (whichever occurs first), and to disable all breakpoints without affecting the setting of breakpoint memory.

NOTE

> Because locations 0003H through 0005H are overwritten for system interrupt handling, the user program is not allowed to execute across this location with GO or STEP. Thus if you wish to begin execution at location 0000H, that location must contain a jump to higher memory.

Display Break Cause Command

Function

The CAUSE command displays the reason for the most recent break in user program execution.

Format

```
CAUSE
```

Operation

The displays produced by the CAUSE command are as follows:

Display	Explanation
WHAT BREAK?	No program execution has occurred since the most recent power-on or reset.
USER ABORT	User pressed ESC key during program execution.
GUARDED ACCESS	The user program attempted to write to an address designated by the user as write-protected (see TOP command), or executed across location 3 (external interrupt 0 vector used by Monitor).
DATA BREAK	Data memory breakpoint occurred.
PROGRAM BREAK	Program memory breakpoint occurred.
SINGLE STEP	The last execution was single step; execution breaks after each instruction in single step.

Step Command

Function

Step command STEP executes one or more instructions at user selected rate, breaking after each instruction. After executing each instruction, the monitor displays the contents of the program counter (PC), the accumulator (ACC), the data pointer register (DPTR), and the stack pointer (SP). Optionally, the user can obtain a display of a specified byte (or bit) from one of the memory spaces accessible to the 8031.

Format

```
STEP [FROM address] [, memory-type address] [, decimal-digit]
```

Where *memory-type* means one of the following keywords:

CBYTE	Program memory
DBYTE	On-chip data memory
RBYTE	On-chip register memory
XBYTE	External data or system memory
RBIT	Bit-addressable memory

Please refer to Memory Access Commands later in this chapter for details on these memory types.

Execute One Instruction

To single-step one instruction, the format is:

```
STEP [FROM address]
```

Examples:

```
STEP
STEP FROM 100
```

Both examples cause one instruction to be executed. The first example executes the instruction at the address in the program counter; the second specifies the address of the instruction explicitly.

After each instruction, the system displays the values of the updated program counter, accumulator, data pointer register, and stack pointer. The format of the display is:

```
Pnnnn Ann Dnnnn Snn
```

Where P identifies the four-digit hexadecimal address in the program counter, A identifies the two-digit hexadecimal value of the accumulator, D identifies the four-digit hexadecimal value of the data pointer register, and S identifies the two-digit hexadecimal value of the stack pointer. In all cases, the value displayed is the one that occurs *after* the instruction has been executed.

Execute One Instruction With Memory Display

In addition to the program counter and accumulator, you can display the contents of one memory location after each instruction in single step. The format of the display is:

```
Pnnnn Ann Dnnnn Snn (nn)
```

Where the P, A, D, and S fields are the program counter, accumulator, data pointer, and stack pointer as described previously, and the value in parentheses is the data byte requested by the user.

The format of the command is:

```
                        ( CBYTE address )
                        | DBYTE address |
STEP [FROM address],   <  RBYTE address  >
                        | XBYTE address |
                        ( RBIT address  )
```

NOTE

> For CBYTE and XBYTE, the *address* may have up to four digits. For DBYTE, the range for *address* is 00H through 7FH. For RBYTE, the range is 80H through 0FFH. For RBIT, the range is 00H through 7FH for the portion of bit-addressable memory in the DBYTE space, and segments of the range 80H through 0F7H for bit addresses in the RBYTE area. See Memory Access Commands later in this chapter for details.

Examples:

```
STEP, DBYTE 0
```

This command executes one instruction, then displays working register R0 (data RAM address 00H) in addition to the program counter, accumulator, data pointer, and stack pointer normally displayed.

```
STEP FROM 100, CBYTE 100
```

This command executes the instruction at address 0100H, and displays the opcode at that address (in addition to the four standard registers).

```
STEP, XBYTE 7000
```

This command executes the intruction in the program counter, then displays the byte at external address 7000H (in addition to the four standard registers).

```
STEP FROM 100, RBYTE 0D0
```

This command executes the instruction at address 100H, then displays the value of the program status word (PSW register, address 0D0H in register memory), in addition to the four standard registers.

```
STEP, RBIT 0D7
```

This command executes the instruction in the program counter, then displays the (bit) value in the Carry (CY) flag (bit address 0D7H in register memory), along with the four standard registers.

Execute in Autostep Mode

Instead of just one instruction per command, you can use autostep mode to step through as many instructions as you wish. You can specify the rate at which stepping occurs, and use any of the kinds of memory display between steps discussed in the previous section. The format for initiating autostepping is:

STEP [FROM *address*] [, *memory-type address*], *decimal-digit*

The entry *decimal-digit* means a numeral, 0 through 9. The digit sets the autostepping rate.

For each instruction in autostep mode, the monitor performs the following operations:

1. Clears the display.
2. Enters execution mode.
3. Executes one instruction.
4. Breaks execution mode.
5. Displays updated program counter, accumulator, data pointer, and stack pointer, and (optionally) the value from memory specified by the user.
6. Checks for ESC key (pause or abort) entered by the user.

7. Inserts an interval of time (approximately 0.5 seconds times the decimal-digit entered by the user).
8. Begins the sequence again with operation 1 above. Operations 1 through 6 take about 0.05 second (50 milliseconds).

NOTE

If LIST mode is ON, the autostep displays can be captured as a form of non-real-time trace. Using LIST slows down the autostepping rate; the amount depends on the LIST baud rate. See the LIST command in this chapter for details

Pause and Abort in Autostep Mode

To halt the autostepping, press the ESC key. The system pauses execution with the most recent information on the display.

To continue autostepping from a pause, press any key except ESC. Autostepping continues with the instruction in the program counter, with the rate and display as specified in the STEP command that began the autostep mode.

NOTE

The character key used to resume autostepping cannot be processed as a character by the user program.

To abort the autostepping from a pause, press ESC again (i.e., press ESC twice to abort autostepping). The system returns to interrogation mode.

Examples:

```
STEP, 0
```

This command initiates autostepping at the fastest rate (no delay interval between operations), displaying the four standard registers after each instruction.

```
STEP FROM 100, 4
```

This command starts autostepping from address 0100H, with an interval of approximately 2 seconds (4 times 0.5 sec) between steps. The four standard registers are displayed after each instruction.

```
STEP F 100, RBY 0D0, 9
```

This command starts autostepping from address 0100H, with an interval of approximately 4.5 seconds between steps (9 times 0.5 seconds).

After each step, the system displays the four standard registers, and in addition the contents of PSW (address 0D0H in register memory).

NOTE

As explained in chapter 4, the user program can contain calls to the SDK-51 monitor in order to use the monitor's I/O facilities. However, any Monitor call or other access to addresses in system memory is treated as *one single step* in single-step or autostep mode. In other words, you cannot single step across system memory (higher than 7FFFH); a monitor call will cause single stepping to appear to pause with display blank while the I/O operation is completed.

NOTE

As explained in Chapter 4, the monitor routine PRINT STRING involves accesses to system memory and to user-configurable memory. The system memory is protected against multiple steps as described in the note above, but when user-configurable memory is accessed, the system attempts to resume single-stepping with unpredictable results. Therefore, the user is cautioned against attempting to single step through this routine. A system reset must be used to recover from this error.

NOTE

During autostepping, an attempt to write to an address lower than the TOP produces a guarded access error and autostepping halts. See TOP command for details.

Memory Access Commands

The memory access commands described in this section allow the user to examine and modify specific bytes and bits in the 8031's on-chip memory and in on-board external memory (user-configurable memory, system memory). The memory spaces accessible to the user are summarized in Table 2-4.

Table 2-4. User-Accessible Memory Spaces

Memory Space	Address Range	Commands
8031 On-chip Data Memory	00H - 7FH (Contiguous)	DBYTE commands.
8031 On-Chip Register Memory	80H - 0FFH (Not contiguous)	ACC, B, DPTR, PSW, SP, TM0, TM1; RBYTE commands.
8031 On-Chip Bit-Addressable Memory	00H - 7FH (Contiguous), 80H - 0F7H (Not Contiguous)	RBIT commands.
User-Configurable Memory	0000H - 7FFFH (Assigned in 8K blocks to Memory 0 (RAM), Memory 1 (RAM) and Memory 2 (ROM)).	CBYTE commands for program memory (below TOP), XBYTE commands for external data memory.
UPI Controller	A000H - AFFFH	Reserved for system use.
Parallel I/O Interface	B000H - B7FFH	Reserved for system use.
	B801H - B803H	XBYTE commands.
	B804H - BFFFH	Reserved for system use.
Breakpoint Memory	C000H - DFFFH	Reserved for system use.
Monitor	E000H - FFFFH	CBYTE commands (read-only).

Format Summary

To display memory contents, use one of the two following formats:

(1) *keyword-reference*

(2) *memory-type partition*

To set (change) memory contents (where allowed by the type of memory), use one of the four following formats:

(1) *keyword-reference* = *byte*

(2) *memory-type partition* = *byte*

(3) *memory-type address* = *byte*, [, [*cr*] *byte* ...

(4) CBYTE *partition* = CBYTE *address*

Where: *keyword-reference* means one of the register names listed in Table 2-5.

byte is any two-digit hexadecimal number (three digits if leading zero required).

memory-type means one of the following keywords:

CBYTE	Program memory
DBYTE	On-chip data memory
RBYTE	On-chip register memory
XBYTE	External memory
RBIT	Bit-addressable memory

partition is a block of addresses entered as *address* TO *address.*

address is a hexadecimal number up to four digits (five with leading zero).

cr means an intermediate carriage return. When entering a list of bytes (format 3 above), you can continue the list to the next display line by pressing RETURN after the comma; the system begins the next line with the next available address. See Example 4-1 (chapter 4) for an example of the continuation feature.

NOTE

Format 4 copies a block of program memory from one partition to another. See User Program Memory for details.

Table 2-5. Keywords for 8051 Registers

Keyword	Description
ACC	Accumulator
B	Multiplication/Division
DPTR	Data Pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program Status Word
SP	Stack Pointer
TMO	Timer 0 (16 bits)
TM1	Timer 1 (16 bits)

On-Chip Data Memory

Figure 2-2 shows a diagram of the 8031's on-chip data RAM. On-chip data addresses are in the range from 00H through 7FH.

There are four banks of general purpose registers (addresses 00H through 1FH).

The stack pointer is initialized (at power up or reset) to address 07H in RAM. The stack pointer is incremented before each stack write (PUSH), so the first stack write would be to address 08H. To display the address pointed to by the stack pointer, enter:

SP

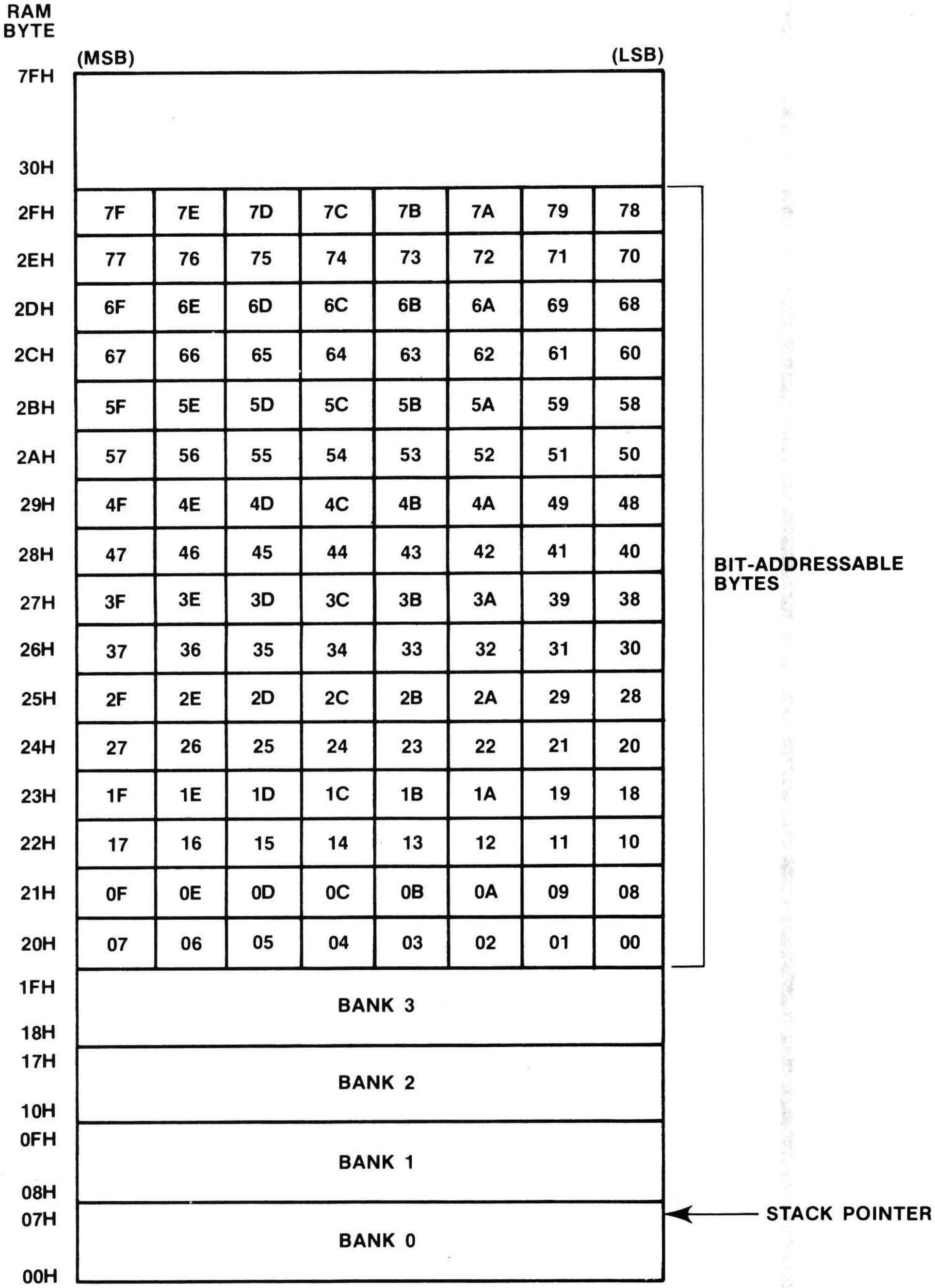
To change the stack pointer address, the format is:

SP = *byte*

Note that the value of *byte* must be an address in data RAM that leaves room for the stack such that SP never exceeds 7FH maximum.

To display the content of one or more addresses in on-chip data memory, the format is:

DBYTE *partition*



RAM BYTE	(MSB)							(LSB)
7FH 30H								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH 18H	BANK 3							
17H 10H	BANK 2							
0FH 08H	BANK 1							
07H 00H	BANK 0							

Figure 2-2. On-Chip Data Memory

Examples:

```
DBYTE 0
```

This command displays a single byte.

```
DBYTE 0 TO 10
```

This command displays seventeen bytes, in groups of four (maximum). When this command is entered, the system puts the first four bytes on the display and identifies the address of the first of the four bytes. The bytes are separated by commas. To indicate that there are more bytes to follow, the system flashes the final comma on the line. Press RETURN to obtain the next four bytes. The last byte (address 10H in this example) does not have a comma.

To fill one or more addresses in data RAM with a single value, the format is:

DBYTE *partition = byte*

For example, to change a single address:

```
DBYTE 5 = 55
```

To clear all of on-chip memory (that is, to fill a partition with the same value, zero in this example):

```
DBYTE 0 TO 7F = 0
```

To enter a list of values into data memory, the format is:

DBYTE *address = byte* [, [*cr*] *byte*] ...

Example:

```
DBYTE 30 = 0,1,2,3,4,5
```

This command sets addresses 30H through 35H with the six bytes in the list. With this type of command, the length of the partition affected is determined by the number of bytes in the list.

NOTE

When you use a list of bytes to set memory (with DBYTE, RBYTE, RBIT, or XBYTE), you can only specify the starting address. You cannot specify a partition of addresses (address TO address) and a list of bytes in the same command, or an error results.

The continuation feature allows you to enter a long list of bytes without repeating the DBYTE keyword. When you near the end of the display line, enter RETURN after the last comma in the line. The system begins the next line with DBYT *xxxx*=, then the prompt; *xxxx* is the address where the next byte will be placed. You can continue as many lines as necessary to enter all the bytes in the list. After the last byte has been entered, press RETURN to terminate the command. (See Example 4-1 in chapter 4 for an example of the continuation feature.)

Special Function Registers

8031 special function register addresses are in the range from 80H through 0FFH, but the registers do not fill the entire address space. Table 2-6 shows the addresses and interpretations of the 8031 registers.

To display the contents of any register in this space, the format is:

RBYTE *address*

Table 2-6. Register Addresses

Register Address (Hex)	Meaning
80H	Port 0
81H	Stack Pointer
82H	Data Pointer, Low Byte
83H	Data Pointer, High Byte
88H	Timer Control
89H	Timer Mode
8AH	Timer 0, Low Byte
8BH	Timer 1, Low Byte
8CH	Timer 0, High Byte
8DH	Timer 1, High Byte
90H	Port 1
98H	Serial Port Control
99H	Serial Port Buffer
A0H	Port 2
A8H	Interrupt Enable
B0H	Port 3
B8H	Interrupt Priority
D0H	Program Status Word
E0H	Accumulator
F0H	Multiplication Register

For example, to display the value of the accumulator:

```
RBYTE 0E0
```

The *address* for RBYTE must be one of the valid register addresses. If you try to read a non-assigned address, a byte of undefined data is returned.

NOTE

The non-assigned addresses in register memory are reserved in the SDK-51 for system use. If you try to write to a non-assigned address, the data is lost, and system operation may be adversely affected.

To change the contents of any of the registers (as bytes), the format is:

RBYTE *address* = *byte*

For example, to write the ASCII character A to the accumulator, the command is:

```
RBYTE 0E0 = 41
```

NOTE

Port 3 is partly reserved for system use. All 8 bits of port 3 are displayed, but bits 2, 6, and 7 are set to ones by the system when beginning execution or single-step.

In addition to the RBYTE commands, the keyword references in Table 2-5 can be used to display and change register memory. For example, to display the (16 bit) data pointer:

```
DPTR
```

To set the data pointer, the format is:

```
DPTR = address
```

For example, to set the pointer to location 0100H:

```
DPTR = 100
```

The keyword references DPTR, TM0, and TM1 allow you to access these registers as 16-bit quantities directly, rather than a byte at a time (as required when using RBYTE). It should be noted that the values in the timer registers (TM0 and TM1) may not reflect the settings of the timers at the moment of exit from execution mode. The monitor commands that stop the timers allow up to 500 microseconds execution time following the occurence of a break condition.

NOTE

> When port 1 or the non-reserved pins of port 3 are changed from the console, the pins are not set until the next user program execution begins. Port 0, port 2, and pins 2, 6, and 7 of port 3 are reserved for system use.

Bit-Addressable Memory

Portions of the on-chip data and special function register memories are bit-addressable. Bit addresses 00H through 7FH are in data memory, as shown in Figure 2-2; bit addresses in the range 80H through F7H refer to individual bits of registers in register memory, (Table 2-7).

Table 2-7. Bit Addresses in Register Memory

Hex Address	Meaning
80H	Port 0, Bit 0
81H	Port 0, Bit 1
82H	Port 0, Bit 2
83H	Port 0, Bit 3
84H	Port 0, Bit 4
85H	Port 0, Bit 5
86H	Port 0, Bit 6
87H	Port 0, Bit 7
88H	Timer 0 Interrupt, Type Control B
89H	Timer 0 Interrupt, Edge Flag
8AH	Timer 1 Interrupt, Type Control Bit
8BH	Timer 1 Interrupt, Edge Flag
8CH	Timer 0 Run Control Bit
8DH	Timer 0 Overflow Flag
8EH	Timer 1 Run Control Bit
8FH	Timer 1 Overflow Flag
90H	Port 1, Bit 0
91H	Port 1, Bit 1
92H	Port 1, Bit 2
93H	Port 1, Bit 3
94H	Port 1, Bit 4
95H	Port 1, Bit 5
96H	Port 1, Bit 6
97H	Port 1, Bit 7
98H	Receive Interrupt Flag
99H	Transmit Interrupt Flag
9AH	Receive Bit 8
9BH	Transmit Bit 8
9CH	Receiver Enable
9DH	Serial Mode Control Bit 2
9EH	Serial Mode Control Bit 1
9FH	Serial Mode Control Bit 0

Table 2-7. Bit Addresses in Register Memory (Continued)

Hex Address	Meaning
A0H	Port 2, Bit 0
A1H	Port 2, Bit 1
A2H	Port 2, Bit 2
A3H	Port 2, Bit 3
A4H	Port 2, Bit 4
A5H	Port 2, Bit 5
A6H	Port 2, Bit 6
A7H	Port 2, Bit 7
A8H	Enable External Interrupt 0
A9H	Enable Timer 0 Interrupt
AAH	Enable External Interrupt 1
ABH	Enable Timer 1 Interrupt
ACH	Enable Serial Port Interrupt
AFH	Enable All Interrupts
B0H	Serial Port Receive Pin
B1H	Serial Port Transmit Pin
B2H	Interrupt 0 Input Pin
B3H	Interrupt 1 Input Pin
B4H	Timer/Counter 0 External Flag
B5H	Timer/Counter 1 External Flag
B6H	Write Data (For External Memory)
B7H	Read Data (For External Memory)
B8H	Priority of External Interrupt 0
B9H	Priority of Timer 0 Interrupt
BAH	Priority of External Interrupt 1
BBH	Priority of Timer 1 Interrupt
BCH	Priority of Serial Interrupt
D0H	Parity Flag
D2H	Overflag Flag
D3H	Register Bank Select Bit 0
D4H	Register Bank Select Bit 1
D5H	Flag 0
D6H	Auxiiary Carry Flag
D7H	Carry Flag
E0H	Accumulator, Bit 0
E1H	Accumulator, Bit 1
E2H	Accumulator, Bit 2
E3H	Accumulator, Bit 3
E4H	Accumulator, Bit 4
E5H	Accumulator, Bit 5
E6H	Accumulator, Bit 6
E7H	Accumulator, Bit 7
F0H	Multiplication Register, Bit 0
F1H	Multiplication Register, Bit 1
F2H	Multiplication Register, Bit 2
F3H	Multiplication Register, Bit 3
F4H	Multiplication Register, Bit 4
F5H	Multiplication Register, Bit 5
F6H	Multiplication Register, Bit 6
F7H	Multiplication Register, Bit 7

To display the content of any of these bits, the format is:

RBIT *address*

To change the content of one of these bits:

RBIT *address* = *bit*

Where *bit* is 0 or 1 (or any hexadecimal number; the system uses the least significant bit of the number).

For example, to display the carry flag:

```
RBIT 0D7
Display: RBIT 00D7=00
```

The display has a leading zero (00 = bit value zero, 01 = bit value one).

To clear the carry flag from the console:

```
RBIT 0D7 = 0
```

To enter a list of bit values into bit-addressable memory, the format is:

RBIT *address* = *bit* [, [*cr*] *bit*] ...

Example:

```
RBIT 30 = 0,1,1,0,0,1
```

This command sets bit addresses 30H through 35H with the six bit values in the list. With this type of command, the length of the partition affected is determined by the number of bit values in the list.

NOTE

> When you use a list of values to set memory (with DBYTE, RBYTE, RBIT, CBYTE, or XBYTE), you can only specify the starting address; you cannot enter a partition (address TO address), since this form is used only to fill memory with a single value. You cannot combine a partition of addresses with a list of values in the same command, or an error results.

The continuation feature allows you to enter a long list of bits without repeating the RBIT keyword. When you near the end of the display line, enter RETURN after the last comma in the line. The system begins the next line with RBIT *xxxx*=, then the prompt; *xxxx* is the address where the next byte will be placed. You can continue as many lines as necessary to enter all the bytes in the list. After the last byte has been entered, press RETURN to terminate the command. (See Example 4-1 in chapter 4 for an example of the continuation feature.)

User Program Memory

User-configurable memory is external to the 8031, in logical addresses 0000H through 7FFFH. Not all this address space may have physical memory installed. User-configurable memory contains the user program and external data space.

Program memory is accessed during program execution by the address in the program counter and the PSEN/ signal from the microprocessor. The user can write protect the program memory space with the TOP command (discussed in this chapter).

To display the program counter, the command is:

```
PC
```

To change the program counter to a new address:

PC = *address*

To display the content of program memory (using the PSEN/ line), the format is:

CBYTE *partition*

For example, to display the bytes from addresses 10H through 20H:

```
CBYTE 10 TO 20
```

The bytes are displayed four per line; press RETURN to see the next line.

To fill a partition of program memory with the same value:

CBYTE *partition* = *byte*

NOTE

When memory is changed with a CBYTE command, the system performs a read-after-write verify. If the value read back does not agree with the value written, an error results.

For example, to clear the partition displayed earlier:

```
CBYTE 10 TO 20 = 0
```

To enter a list of opcode and operands as hexadecimal values:

CBYTE *address* = *byte* [, [*cr*] *byte*] ...

For example, to load the instruction LCALL E006H numerically starting at address 100H:

```
CB 100 = 12,0E0,06
```

The three bytes are loaded into three consecutive addresses (100H, 101H, 102H).

The continuation feature allows you to enter a long list of bytes without repeating the CBYTE keyword. When you near the end of the display line, enter RETURN after the last comma in the line. The system begins the next line with CBYT xxxx=, then the prompt; xxxx is the address where the next byte will be placed. You can continue as many lines as necessary to enter all the bytes in the list. After the last byte has been entered, press RETURN to terminate the command. (See Example 4-1 in Chapter 4 for an example of the continuation feature.)

To copy a block of bytes from one place in program memory to another, the format is:

CBYTE *partition* = CBYTE *address*

For Example:

```
CB 100 TO 1FF = CB 300
```

This command copies 256 bytes from addresses 300H through 3FFH into addresses 100H through 1FFH. The contents of 300H through 3FFH are not changed.

See the ASM instruction in the next section for an alternative way to load instructions into program memory.

External Data Memory

External data memory is accessed with the MOVX instructions; the RD/ and WR/ lines from the 8031 control the operation. For external data memory access, the command formats are the same but the keyword is XBYTE rather than CBYTE. The XBYTE command uses the RD/ and WR/ to access external data memory.

External data memory command format summary:

Display:

XBYTE *partition*

Fill:

```
XBYTE partition = byte
```

Load list of bytes:

```
XBYTE address = byte [, [cr] byte] ...
```

The continuation feature allows you to enter a long list of bytes without repeating the XBYTE keyword. When you near the end of the display line, enter RETURN after the last comma in the line. The system begins the next line with XBYT *xxxx*=, then the prompt; *xxxx* is the address where the next byte will be placed. You can continue as many lines as necessary to enter all the bytes in the list. After the last byte has been entered, press RETURN to terminate the command. (See Example 4-1 in chapter 4 for an example of the continuation feature.)

System Monitor ROM

The user may read the contents of the system monitor with the following type of command:

```
CBYTE partition
```

ASM Command

Function

The ASM command assembles a single instruction mnemonic into program memory. The assembler mode is entered from interrogation mode when the ASM command is executed. In assembler mode, the system maintains an assembly pointer to indicate the address where the next instruction is to begin.

Format

```
ASM ORG address
ASM
```

Operation

The ASM command begins assembler mode at the *address* specified after the ORG keyword. If no ORG address is specified, the current assembly pointer location is used; the assembly pointer is initially at 0000H.

The monitor displays the pointer address followed by the assembler mode prompt. The user then enters an 8051 instruction mnemonic with operands in hexadecimal; terminate each instruction by pressing the RETURN key. The opcode of each instruction is stored in the ORG address and operand bytes (if any) are stored in subsequent addresses. The assembly pointer is then updated to point to the next available memory location and the new address is displayed.

The user may then enter another instruction, or press RETURN after the prompt to exit assembler mode.

If the user enters an invalid instruction, a syntax error message is displayed. Press the RETURN key to clear the error and return to interrogate mode. The assembly pointer is not changed by the error.

To exit from assembler mode back to interrogation mode, press RETURN immediately after the prompt.

Certain restrictions are placed on the type of instructions that may be entered in this mode. Operands must be hexadecimal; expressions are not allowed. Symbols, labels, the generic CALL and the generic JMP instructions are not allowed. AJMP addresses can only be absolute (numeric values) and JMP is allowed only in the form "JMP @A+DPTR". The $ symbol for the current program counter is not allowed.

Please refer to the *MCS™-51 Microcontroller Family User's Manual* for details on the 8051 instruction set.

Example:

Enter assembler mode and assemble an instruction

Entry	ASM ORG 0010
Response	0010 -
Entry	MOV A,@R0
Response	0011 -

The user begins assembly mode with an ORG address of 0010H. The system displays the ORG address followed by the assembly mode prompt. The user enters a MOV instruction for assembly. The monitor assembles the instruction, stores the object code in program memory starting at address 0010H, then displays the next available address in program memory (0011H).

DASM Command

Function

The DASM command allows the user to disassemble memory values into 8051 instruction mnemonics.

Format

DASM *partition*

Operation

The DASM command displays the content of user-configurable memory (addresses 0000H through 7FFFH), and of the Monitor (E000H through FFFFH) as disassembled instructions; other addresses return undefined instructions. The system assumes that the beginning address of the partition is the first byte of an instruction, and it completes an instruction if the first byte of the instruction is within the partition. No assembler mode prompt is displayed with the disassembly command. The specific partition of code is disassembled and the monitor returns to the interrogation mode.

NOTE

If you request the disassembly of the undefined opcode, A5H, the DASM command aborts the display of any remaining instructions in the partition and displays an error message. Press ESC or both RESET keys to clear the error. Also, the operand @DPTR is abbreviated in the display to @DPT.

Example:

To disassemble a single instruction (at address 0100H):

```
DASM 100
```

To disassemble the several instructions that are in the partition from 1010H through 1025H:

```
DASM 1010 TO 1025
```

Top of Program Memory Command

Function

The top of program memory command (TOP) identifies the highest address in program memory. Once this address has been set, all memory locations below that address are protected from writes during the execution of the user program. Any attempt to write in this space during program execution causes a guarded access break.

The TOP command can also be used to display the top of memory address.

Format

```
TOP
TOP = address
```

Operation

The user may write-protect the memory containing the user program, to prevent the program from changing any instructions by mistake. The write-protection offered is contiguous starting with address 0000H. However, the write protection does not prevent the system from writing addresses 03H, 04H, and 05H.

For write protection, user-configurable memory is arbitrarily divided into 256-byte blocks. The user specifies the top of program memory at any address from 0000H to 7FFFH. The top of memory value is determined from the specified address in the following manner:

1. If the user specifies a TOP value is equal to zero, all memory space between 0000H and 7FFFH is treated as external data memory. Writes and reads are permitted; no errors are generated.
2. If the user specifies a TOP value between 0001H and 7FFFH, the top of program memory is set to the highest address within the 256-byte block in which the selected address falls.
3. If the user specifies a TOP value greater than 7FFFH, an error message is displayed and the TOP value remains unchanged from its previous value.

Note that the user specified value for top of program memory is written to the top of program memory comparators just prior to entering the execution mode. The user will thus be able to examine and modify all addresses while in interrogation mode.

NOTE

> On a normal data break (write to data address), instruction halts after the instruction that caused the break. The program counter points to the instruction immediately following the address that caused the data break, and the CAUSE command displays DATA BREAK. However, after a break due only to guarded access (no data break enabled), one more instruction is executed after the one that caused the guarded access. Thus the program counter does not point to the next instruction after the one that caused the break. The extra instruction does not affect the write protection, which is guaranteed by the hardware.
>
> Furthermore, if a data breakpoint is set on a write-protected address, and the program breaks on that address, no extra instruction is executed (program counter points to the next instruction after the one that caused the break), but the CAUSE display after the break shows a guarded access break, not a data break.

Default

Initially and after reset, TOP = 0; no user-configurable memory is write-protected, no guarded access errors are generated.

Examples:

Example 1: Set top of program memory.

```
TOP = 1F10
```

the user enters top of program memory address 1F10H. Monitor sets actual top of program memory to highest address in 256-byte memory block, which is 1FFFH.

Example 2: Display top of program memory

```
TOP
Display: TOP = 1FFF
```

Actual top of program memory is displayed.

Input/Output Operations

Figure 2-3 shows the locations of the serial I/O interface, the audio cassette I/O interface, and the parallel I/O interface. Serial I/O and audio cassette I/O operations are discussed in the remainder of this chapter. Parallel I/O operations involve the user program; see chapter 4 for the discussion of parallel I/O operations.

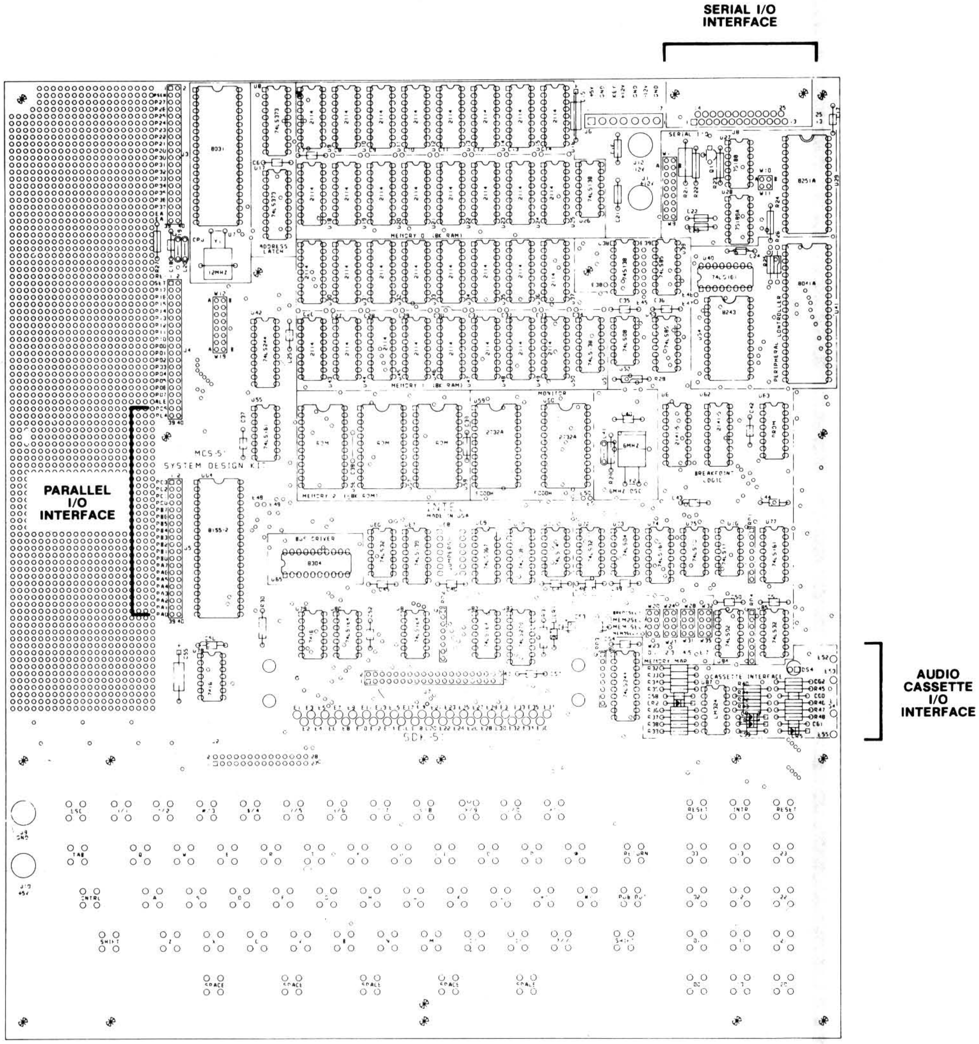


Figure 2-3. I/O Interface Locations

I/O Object Code Format

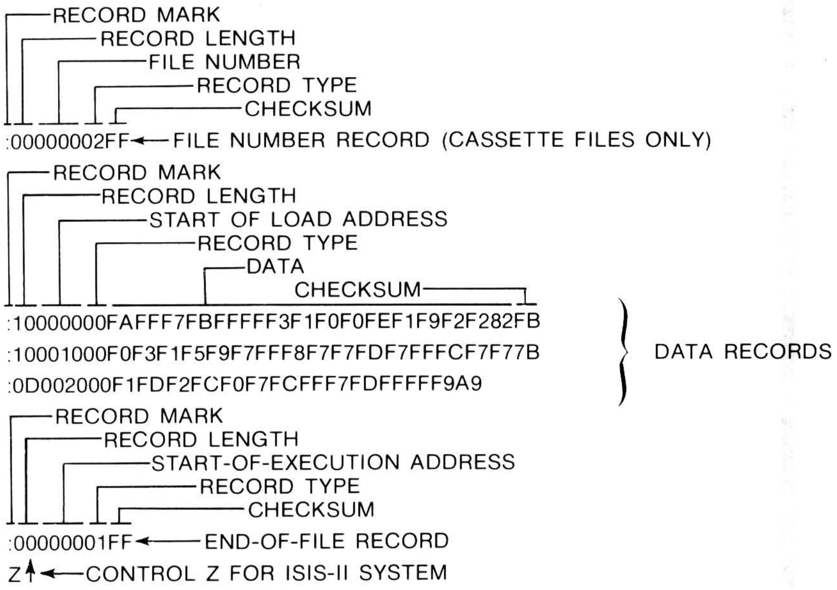
The object code generated and accepted by the development system assembler and by the SDK-51 is a series of data records in hexadecimal format terminated with an end of file record and CTRL Z character (Figure 2-4). The SDK uses much the same format for audio cassette I/O operations, except that the data is in binary instead of hexadecimal, the final CTRL Z is omitted, and a file number record is added as the first record.

The format of each data record is as follows (one frame is four bits; two frames equal one byte):

Field 0: Record mark (frame 0 is always a colon :)

Field 1: Record length (frames 1 and 2; the length of the data field in bytes)

Field 2: Start-of-load address (frames 3, 4, 5, and 6; the data bytes are loaded into consecutive memory addresses starting with this address)



```
:00000002FF
:10000000FAFFF7FBFFFFF3F1F0F0FEF1F9F2F282FB
:10001000F0F3F1F5F9F7FFF8F7F7FDF7FFFCF7F77B
:0D002000F1FDF2FCF0F7FCFFF7FDFFFFF9A9
:00000001FF
Z↑
```

Figure 2-4. I/O Object File in Hexadecimal Format

Field 3: Record type 00 (frames 7 and 8)

Field 4: Data (frames 9 through N; a record can contain a maximum of sixteen bytes, thus N is always less than or equal to forty)

Field 5: Checksum (frames N+1 and N+2; the data is summed byte by byte, and the least significant byte is taken as the checksum)

The end of file record has the following format:

Field 0: Record mark (frame 0)

Field 1: Record length 00 (frames 1 and 2)

Field 2: Start-of-execution address (frames 3, 4, 5, and 6); after a DOWNLOAD from the development system or LOAD from cassette, the SDK-51 program counter is set to this address.

Field 3: Record type 01 (frames 7 and 8)

Field 4: Checksum (frames 9 and 10)

The file number record (cassette files only) has the following format:

Field 0:	Record mark (frame 0)
Field 1:	Record length 00 (frames 1 and 2)
Field 2:	File number (frames 3, 4, 5, and 6)
Field 3:	Record type 02 (frames 7 and 8)
Field 4:	Checksum (frames 9 and 10)

Serial I/O Interface Operations

The following procedures and commands allow the user to send and receive data through the on-board serial I/O interface.

The SDK-51 provides a number of options for serial I/O communications with external peripherals such as a computer terminal, printer, teletypewriter or development system.

1. Either RS232 or 20 mA current loop interface is allowed.
2. Either the on-board 8251 (U29) USART or the 8051 internal UART can be selected for serial data transmission.

NOTE

> For serial I/O operations, +12-volt and -12-volt power supplies must be connected to the board. On the power cable, the BLUE wire connects to the +12V terminal and the YELLOW wire connects to the -12V terminal. Both remaining BLACK wires connect to 12 volt return.

Jumper Installation

Sheet 8 of the schematic drawings shows the jumper terminals that allow routing of the various serial I/O interface lines to serial I/O interface connector J8. Jumpers W1 and W2 select between the on-board 8251A and the 8031 on-chip UARTs; jumpers W3 through W11 select among RS-232 (slave mode), RS-232 (master mode), and current loop protocols. Tables showing the required jumper settings are given where appropriate to the UPLOAD, DOWNLOAD, and LIST procedures later in this chapter. Jumper settings applicable to the use of the 8031 on-chip UART appear where applicable in chapter 4.

NOTE

> Use of the 8031 on-chip UART requires the user program to interface to the SDK hardware. Please refer to Chapter 4 for details. The discussion in chapter 2 for the most part assumes the use of the on-board 8251A.

Serial I/O Interface Cable

Serial I/O data is transmitted to and from the SDK-51 through connector J8. J8 accepts a male, 25-pin, delta type connector. Refer to Table 2-8 for details on the J8 pinout.

Table 2-8. Serial I/O Connector J8

Pin	Signal
J8-1	Not used
J8-2	RS-232 TRANSMITTED DATA
J8-3	RS-232 RECEIVED DATA
J8-4	RS-232 REQUEST TO SEND
J8-5	RS-232 CLEAR TO SEND
J8-6	Not used
J8-7	GROUND
J8-8	Not used
J8-9	Not used
J8-10	Not used
J8-11	Not used
J8-12	CURRENT LOOP RECEIVED DATA (+)
J8-13	CURRENT LOOP TRANSMITTED DATA (+)
J8-14	Not used
J8-15	Not used
J8-16	Not used
J8-17	Not used
J8-18	Not used
J8-19	Not used
J8-20	Not used
J8-21	Not used
J8-22	Not used
J8-23	Not used
J8-24	CURRENT LOOP RECEIVED DATA (-)
J8-25	CURRENT LOOP TRANSMITTED DATA (-)

Serial I/O Baud Rate

The timer in the 8155-2 (U64) provides a baud clock for the 8251 (U29) serial I/O interface device. The BAUD command discussed in the next section sets the baud rate of the 8155 timer.

The 8031 supplies the baud clock for its internal UART. Refer to Chapter 4 for details on programming the 8031 internal baud rate.

Baud Command

Function

The BAUD command allows the user to set the baud rate of data transfer through the serial I/O port. The command also allows the current baud rate to be displayed.

Format

$$\text{BAUD} = \left\{ \begin{array}{l} 110 \\ 300 \\ 600 \\ 1200 \\ 2400 \\ 4800 \\ 9600 \end{array} \right\}$$

BAUD

Operation

The BAUD command causes the timer in the 8155-2 parallel I/O interface to be set to produce a timing signal (BAUDCLK) for the 8251 serial I/O interface at the selected baud rate. In order for this function to operate as specified, the UPI-41A must operate with the 6 MHz crystal supplied with the kit.

Default

Initially and after reset, the serial I/O data rate is set to 2400 baud.

Examples:

Example 1: Set baud rate.

BAUD = 600

Baud rate is set to 600.

Example 2: Display current baud rate.

BAUD

Display: BAUD = 600

Current baud rate setting of 600 is displayed.

Development System Interface

The SDK-51 can be interfaced to an Intel development system through the serial I/O port, allowing user programs to be uploaded and downloaded from the SDK-51. For this application, the SDK-51's serial I/O port should be jumpered to use the 8251 serial I/O interface and the RS-232 (slave mode) interface protocol (see Table 2-9).

NOTE

To ensure proper SDK-51 operation, remove the connection between the SKD and the development system before turning the development system power on or off. If the SDK appears to halt after a development system power on/off, press the RESET keys to restore operation.

Table 2-9. Serial I/O Jumpers for UPLOAD and DOWNLOAD

Jumper	Connection
W1	open
W2	W2A - W2B
W3	open
W4	open
W5	W5A - W5B
W6	W6A - W6B
W7	W7A - W8A
W8	W8A - W7A
W9	open
W10	W10B - W11B*
W11	W11B - W10B*

*NOTE: Alternately, connecting W10A - W10B and W11A to W11B allows normal operation of RTS/ and CTS/ when external handshake is required.

Table 2-8 shows the requirements for an interface cable to be connected between J8 of the SDK-51 and the development system. For Series II (and later) development systems, the interface cable must be connected to the development system's Serial Channel 1. For Model 800 development systems, the interface cable must be connected to the development system's RS232 CRT port; the development system's console must in this case be connected to the development system's TTY port.

The UPLOAD and DOWNLOAD commands, combined with the development system's COPY command, control the data transfer as described in the next section.

UPLOAD Command

Function

The UPLOAD command copies programs and data from the SDK-51 user-configurable memory to a disk file, using the development system file utility (COPY command).

Format

```
UPLOAD partition
```

Operation

The UPLOAD command transfers a memory partition from the SDK-51 to a file through the development system. The partition may include any address within the range 0000H through 7FFFH. The first address in the partition is stored as the start-of-load address in the file.

NOTE

The UPLOAD operation clears LIST mode to RESET.

The upload procedure is as follows:

1. Enter a command with the following format on the development system keyboard:

   COPY :TI: TO :F*x*:*filename*

   NOTE: Model 800 users substitute :VI: for :TI:.

   The entry :F*x*:*filename* specifies the destination drive and file name. (Make sure the disk containing the file is mounted on the drive before entering the COPY command.)
2. Set the SDK-51 program counter to the start of execution address desired for the program.
3. Enter the SDK-51 command:

   UPLOAD
4. The SDK-51 displays the message:

   LOADING

When the transfer of data is complete, the development system and the SDK-51 both return a command prompt.

Example:

Upload a program from locations 2000H through 2800H in SDK user-configurable memory to a development system disk file named PROG.HEX.

1. Development System Entry:

   COPY :TI: TO :F1:PROG.HEX

   NOTE: Model 800 users substitute :VI: for :TI:.
2. SDK-51 Entry:

   UPLOAD 2000 TO 2800

3. SDK-51 Response

   ```
   LOADING
   ```

DOWNLOAD Command

Function

The DOWNLOAD command transfers a file from the development system to a partition of memory locations in the SDK-51 user-configurable memory within the address range 0000H through 7FFFH. The download operation uses the start-of-load address in the file to determine the first address in the partition; the number of bytes of data in the file determines the length of the partition affected by the download.

Format

```
DOWNLOAD
```

Operation

The download procedure is as follows:

1. Enter the DOWNLOAD command on the SDK.
2. The SDK-51 responds with the following message:

   ```
   LOADING
   ```

3. Enter the following command on the development system:

   COPY :F*x*:*filename* TO :TO:

   NOTE: Model 800 users substitute :VO: for :TO:.

   The entry :F*x*:*filename* gives the drive and file name containing the program to be downloaded. (Make sure the disk containing the file is mounted in the drive before entering the COPY command.)

When transfer of data is complete, the development system and the SDK-51 both return a command prompt.

The DOWNLOAD command sets the SDK-51 program counter to the start-of-execution address in the file.

Example:

Download a program from development system to SDK-51.

1. SDK-51 Entry

   ```
   DOWNLOAD
   ```

2. SDK-51 Response

   ```
   LOADING
   ```

3. Development System Entry

   ```
   COPY :F1: PROG.HEX TO :TO:
   ```

   NOTE: Model 800 users substitute :VO: for :TO:.

   Code stored in development system under filename PROG.HEX is transferred to SDK-51. Check the 8051 program counter, register PC, for the starting address of the downloaded program.

LIST Command

Function

The LIST command causes any data being displayed on the SDK-51 display to be transmitted through the serial I/O port to an external I/O device such as a printer. This feature permits the user to obtain hard copy of data being displayed. To use the LIST command, the 8251 serial I/O interface and either RS-232 (master mode) or current loop protocol must be jumper selected (see Tables 2-10 and 2-11).

Format

```
LIST = ON
LIST = RESET
LIST
```

Table 2-10. Serial I/O Jumpers for LIST Using RS-232 Protocol

Jumper	Connection
W1	open
W2	W2A - W2B
W3	open
W4	open
W5	W5A - W6A and W5B - W6B
W6	W6A - W5A and W6B - W5B
W7	W7A - W8A
W8	W8A - W7A
W9	open
W10	W10B - W11B*
W11	W11B - W10B*

*NOTE: Alternatively, connecting W10A - W10B and W11A - W11B allows normal operation of RTS/ and CTS/ if external handshake is required.

Table 2-11. Serial I/O Jumpers for LIST Using Current Loop

Jumper	Connection
W1	open
W2	W2A - W2B
W3	W3A - W3B
W4	W4A - W4B
W5	open
W6	open
W7	open
W8	W8A - W9A
W9	W9A - W8A
W10	open
W11	open

Operation

Enabling the list operation (LIST = ON) alters the speed of the display. Normally the byte values are read out on the display one line at a time, so the user can examine each byte. When the list function is enabled, data is displayed continuously at the selected baud rate.

To disable the list function, enter LIST = RESET.

To display the list status (ON/RESET), type LIST.

NOTE

The UPLOAD operation clears the LIST mode to RESET.

Default

Initially and after reset, LIST = RESET (no listing is performed).

Examples:

To list a partition of user-configurable memory through the serial I/O port:

```
LIST = ON
CBYTE 2000 TO 2800
```

The bytes of data are flashed on the display and sent to the serial I/O port at the selected baud rate without pausing at the end of each line. (If desired, use the ESC key to halt the transfer.)

To list the output of autostepping mode:

```
LIST = ON
STEP FROM 100, 2
```

After each instruction executed in autostep mode (see STEP command), the system displays the program counter and other registers. With LIST on, the displays are sent to the list device as a form of non-real-time trace.

To disable the list function:

```
LIST = RESET
```

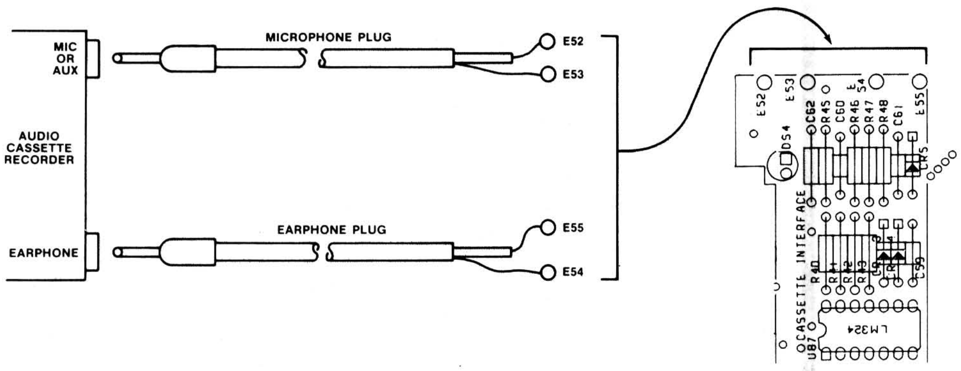
To display the state of the list function:

```
Entry       LIST
Response    LIST = RESE
```

The state of the list function is displayed; in this case it is reset (note screen abbreviation).

Audio Cassette Interface

To connect an audio cassette recorder/player to the SDK-51, use the following procedure (refer to Figure 2-5):



NOTE:
Connecting both the microphone plug and the earphone plug may introduce noise into the data transmission. Simultaneous connection is therefore not recommended.

Figure 2-5. Audio Cassette Interface Connections

1. Select two leads (preferably shielded) with jacks that fit your cassette recorder. Strip the shielding from the leads to expose the signal and ground wires.
2. Connect the microphone lead to E52 (signal wire) and E53 (ground wire).
3. Connect the earphone lead to E55 (signal wire) and E54 (ground wire).

NOTE

On some recorders, connecting both the microphone plug and the earphone plug may introduce noise into the data transmission. Simultaneous connection is therefore not recommended.

SAVE Command

Function

The SAVE command copies the data in a partition of user-configurable memory to a file on the audio cassette tape.

Format

SAVE *file-number, partition*

Where: The *file number* must be entered as hexadecimal digits, up to four, or five if leading zero is required (e.g., 5 for file number 5, 7510 for file number 7510, 0F7 for file number F7, 0FF30 for file number FF30)

partition is a block of addresses in the range from 0000H through 7FFFH.

Operation

The SAVE command copies a partition of SDK-51 user-configurable memory to the cassette interface. The procedure is as follows:

1. Connect the microphone lead to the microphone or auxiliary input on the recorder.
2. Place a tape cassette on the recorder, and position the tape to the point where the file is to be recorded.
3. If your recorder has manual record level adjustment, set the record volume level to midpoint (on some recorders, the record volume is adjusted automatically, not manually).
4. Set the SDK-51 program counter to the start of execution address desired for the file.
5. On the SDK-51, enter the SAVE command. After you press RETURN to enter the command, the system displays the message:

   START CASSETTE
6. Start the cassette recorder in RECORD mode.
7. Press the RETURN key again on the SDK-51. The display is blank while the transfer takes place.
8. When the partition has been transferred to tape, the system displays the prompt. Stop the recorder and disconnect the microphone lead.

NOTE

During a cassette operation, the system does not scan either the keyboard or the display. To abort the transfer, press both RESET keys

The system saves the partition as a single output file, with file number record, data records, and end-of-file record as discussed earlier in this chapter (see I/O Object Code Format). The data is stored as binary tone pulses (see chapter 3 for details on data encoding). The SAVE command uses the program counter setting as the start-of-execution address in the end-of-file record.

Example

Transfer a program in addresses 100H through 3FFH to a file numbered 05 on cassette, using PC = 100 as the start-of-execution address.

Entry:	PC = 100
Entry:	SAVE 05, 100 TO 3FF
Response:	START CASSETTE
Entry;	Press RETURN key again

LOAD Command

The LOAD command copies data from the designated file on cassette to the SDK-51 user-configurable memory, using the start-of-load address in the file to determine the beginning of the memory partition. The LOAD command can also be used to search for file numbers on the tape.

Format

```
LOAD file-number
LOAD
```

The file number must be entered as hexadecimal digits, up to four, or up to five if leading zero is required (e.g., 05 for file number 5, 7510 for file number 7510, 0F7 for file number F7, 0FF30 for file number FF30)

Operation

To load a file (program) from file to user-configurable memory, the procedure is as follows:

1. Connect the earphone plug to the earphone or monitor output on the recorder.
2. Place the cassette containing the file on the recorder.
3. Adjust the cassette volume by starting the cassette, watching the red LED on the SDK-51 near the cassette I/O connectors, and setting the recorder volume control so that the LED just begins to flicker on and off. Some trial and error may be required to obtain good results with a given recorder.

NOTE

If the signal level from the recorder is too high, noise errors are introduced into the data.

4. Position the tape ahead of the file to be read.
5. Enter the LOAD command with the desired file number. When you press RETURN to enter the command, the system displays the message:

   START CASSETTE

6. Start the recorder in PLAY or FORWARD mode.

7. Immediately press the RETURN key again (must occur before the recorder encounters the start of the file). The system enters cassette transfer mode and blanks the display while the transfer is taking place.
8. The system reads the tape until it locates the specified file number, then stores the data records in the partition of addresses that begins with the start-of-load address in the file. The transfer ends when the end-of-file record is read, and the system displays the message:

   LOADED FILE *file-number*
9. Stop and rewind the tape, and disconnect the earphone plug.
10. Press the RETURN key again to obtain the prompt.

The LOAD operation sets the SDK-51 program counter to the start-of-execution address on the file.

Example:

Transfer data from file 05 on cassette to user-configurable memory on the SDK-51.

Entry:	LOAD 05
Response:	START CASSETTE
Entry:	Start tape, press RETURN key again
Response:	LOADED FILE 0005
Entry:	Stop tape, press RETURN key again.

To search for the number of the next file on the cassette tape, enter the LOAD command without a file number, then start the tape. The tape is read forward until the next file number record is found, then the number of that file is displayed:

FIRST FILE FOUND = *file-number*

To display the next file number after the one displayed, you must type LOAD again. Stop the tape if files are short.

For example, suppose file 0006 is the next file on the tape.

Entry:	LOAD
Start tape forward.	
Response:	FIRST FILE FOUND = 0006

Summary of Command Formats

This section gives the formats of all the SDK-51 commands in alphabetical order.

ABR = *partition* [, *partition*] ...

ASM [ORG *address*]

B [= *byte*]

$$\text{BAUD} \left[= \left\{ \begin{matrix} 110 \\ 300 \\ 600 \\ 1200 \\ 2400 \\ 4800 \\ 9600 \end{matrix} \right\} \right]$$

BR = *partition* [, *partition*] ...

BR = RESET

BR

CAUSE

CBYTE *address* = *byte* [, [*cr*] *byte*] ...

CBYTE *partition* [= *byte*]

CBYTE *partition* = CBYTE *address*

DASM partition

DBYTE partition [= *byte*]

DBYTE *address* = *byte* [, [*cr*] *byte*] ...

DOWNLOAD

DPTR [= *address*]

GO [FROM *address*] [FOREVER
TILL PROGRAM
TILL DATA
TILL PROGRAM OR DATA]

LIST = ON

LIST = RESET

LIST

LOAD [*file-number*]

PC [= *address*]

PSW [= *byte*]

RBIT *partition* [= *bit*]

RBIT *address* = *bit* [, [*cr*] *bit*] ...

RBYTE *partition* [= *byte*]

RBYTE *address* = *byte* [, [*cr*] *byte*] ...

SAVE *file-number*, *partition*

SP [= *byte*]

STEP [FROM *address*] [, *memory-type address*] [, *decimal-digit*]

TM0 [= *address*]

TM1 [= *address*]

TOP [= *address*]

UPLOAD *partition*

XBYTE *partition* [= *byte*]

XBYTE *address* = *byte* [, [*cr*] *byte*] ...

CHAPTER 3
FUNCTIONAL DESCRIPTION

Introduction

This chapter describes the internal operation of the SDK-51 in terms of the integrated circuit devices and control signals used in the design. The description of the circuitry is at the block diagram level; a detailed block diagram (Figure 3-6) accompanies this chapter on a convenient fold-out page. In addition, you may wish to compare the discussion to the SDK-51 schematics furnished with the kit.

The circuit blocks and functions discussed are:

- Microcontroller
- Memory Mapping
- Address Bus Control
- Data Bus Control
- User-Configurable Memory (Memory Configuration, RAM, and ROM)
- Monitor
- Reset Operation
- 6 MHz Clock Generator
- Parallel I/O Interface
- UPI Control
- Keyboard and Display Timing Generation
- Keyboard Control
- Display Control
- Serial I/O Interface
- Audio Cassette Interface
- Top of Program (T.O.P.) Memory Protect Circuit
- Breakpoint Control

A glossary of the main signal lines in the SDK-51 is given at the end of the chapter.

Microcontroller

Detailed information on the Intel 8051 microcontroller appears in the *MCS-51 User's Guide* (see Preface for reference), and is not repeated here. Several points regarding the use of the 8051 in the SDK-51 should be noted.

- The controller device supplied with the SDK-51 is the 8031 version (no on-chip ROM).

NOTE

> In this chapter, references to the 8051 family in general use "8051"; references to the SDK-51 controller in particular use "8031".

- A 12-MHz crystal controlled oscillator provides the fundamental clock frequency for the 8031 microcontroller. Internally, this clock signal provides timing for internal controller operations and for the ALE, PSEN/, WR/, and RD/ signals generated by the 8031 to control the operation of various circuits in the SDK-51.

- The 8051 has two external interrupt lines, INT0 and INT1. The SDK-51 reserves INT0 exclusively for system operation, and assigns it highest priority. The user can place a jumper to allow the INTR switch on the keyboard to trigger the 8031's INT1 input. The user must supply the interrupt processing code (see Chapter 4 for details and limitations).
- The SDK-51 provides a power-on reset, causing the system to display a sign-on message as soon as power is applied. It also has two RESET switches; pressing the two switches simultaneously resets the system, including the 8031 (pressing just one switch has no effect). The RESET signal to the 8031 controller causes it to clear all internal registers (the stack pointer is reset to 07H, and ports 1 and 3 are reset to FFH), and to fetch address 0000H in system memory (see Reset Operation later in this chapter for details on reset handling involving the microcontroller and monitor ROM).
- Multiplexed address and data appears at I/O port pins P00 through P07 and P20 through P27 of the 8031. The parallel I/O interface (8155) on the SDK-51 has on-chip demultiplexing capabilities; the multiplexed address/data bus is thus connected directly to the parallel I/O interface block. The RAM, ROM, UPI controller, monitor ROM, and breakpoint logic require separate address and data buses; the address and data bus control sections demultiplex and decode the address and data buses for these devices.
- I/O port pins P10 through P16 from the 8031 can be used to operate the auxiliary keypad, or as an interface to user-designed circuitry. These operations require user programming (refer to Chapter 4 for details).
- The 8051 has an internal serial I/O port (UART). The SDK-51 monitor does not use this port. However, the SDK-51 may be jumpered to connect the 8031 serial I/O port with the serial I/O section on the board. User code is required to operate the on-chip UART; refer to Chapter 4 for details.
- The monitor provides facilities for reading and writing the 8051's internal RAM and hardware registers (see Chapter 2)
- The EA/ pin from the 8031 is tied to ground on the SDK-51 board.

Memory Mapping

The 8051 microcontroller can address two independent 64K memory spaces, external data memory and external program memory. The SDK-51 design offers a single 64K byte address space to be used for both program memory and data memory. The upper half of the address space (addresses 8000H through FFFFH) is reserved the monitor program and for interfacing the microcontroller with system functions. The lower half (addresses 0000H through 7FFFH) is the user-configurable memory space; it provides general purpose storage for user programs and data. Table 3-1 gives some details on the SDK-51 memory map.

Table 3-1. Memory Mapping

Address	Memory Block Assignment
0000H - 1FFFH	User-configurable 8K
2000H - 3FFFH	User-configurable 8K
4000H - 5FFFH	User-configurable 8K
6000H - 7FFFH	User-configurable 8K
8000H - 9FFFH	System memory, not used.
A000H - AFFFH	UPI controller, 4K
B000H - BFFFH	Parallel I/O interface, 4K
C000H - CFFFH	Lower half of breakpoint RAM, 4K
D000H - DFFFH	Upper half of breakpoint RAM, 4K
E000H - EFFFH	Monitor ROM 0, 4K
F000H - FFFFH	Monitor ROM 1, 4K

Address Bus Control

Eight-bit latches U8 and U17 demultiplex the address/data bus lines from the microcontroller. U8 and U17 latch the 16 address bits on the trailing edge of ALE (high-to-low transition).

Address decoder U38 decodes address lines A12 through A15 to produce six device select lines: UPISEL/, IOSEL/, LOBRKMEMSEL/, HIBRKMEMSEL/, MONLOSEL/, and MONHISEL/. Table 3-2 gives details on these select lines. (Compare to Table 3-1.)

Table 3-2. Address Decoder Select Lines

Hexadecimal Value on A12 - A15	Device Select Line Pulled Low	Device Selected
AH	UPISEL/	UPI Controller
BH	IOSEL/	Parallel I/O Interface
CH	LOBRKMEMSEL/	Breakpoint RAM, low 4K
DH	HIBRKMEMSEL/	Breakpoint RAM, high 4K
EH	MONLOSEL/	Monitor ROM, low 4K
FH	MONHISEL/	Monitor ROM, high 4K

Data Bus Control

The data bus control area contains two circuits:

Memory read and write

FORCENOP circuit

The RDBRKMEM/ circuit is an additional input to the data bus; this circuit is described in the section on Breakpoint control.

Memory Read and Write

The device at U65 is an 8-bit bidirectional bus driver that transmits data between the controller and data bus lines DB0 through DB7. The WR/ line (T input to U65) determines the direction of data flow through U65: during WR/ low (T = 0), data flows from the controller to the data bus; during WR/ high (T = 1), data flows from the data bus to the controller.

The OE/ input to U65 enables the device. This input is pulled low (enabled) during each read or write cycle, except when the controller is communicating with the parallel I/O interface (i.e., when IOSEL/ is active) or when the FORCENOP signal is active. When IOSEL/ = 0, the OE/ input is pulled high, inhibiting data transfer through U65.

FORCENOP Circuit

U42 is a unidirectional bus driver. Since all of U42's eight inputs are zeros, the circuit puts a NOP instruction (opcode 00H) on the data bus. This action is required by the breakpoint logic (see Breakpoint Control later in this chapter).

User Configurable Memory

The user configurable memory is the lower 32K bytes of system external memory, as shown in the memory map in Table 3-1. The user configurable memory area is divided into four address spaces of 8192 bytes (8K) each. Each of these address spaces may be filled either with RAM devices or with ROM devices, but the two types may not be mixed in a given address space. Printed circuitry is provided on the SDK board for two 8K blocks of RAM (memory 0 and memory 1) and one 8K block of ROM (memory 2). To make use of the remaining 8K of addressable memory space, the user must install memory devices either in the prototype area or off-board.

Memory Configuration

Address decoder U67, gate U72, and the jumper matrix W20 through W35 assign the three 8K on-board memory blocks to three out of the four 8K address spaces in the memory map.

Memory configuration jumpers W20 through W35 are used to assign address ranges to the three 8K memory blocks on-board. Lines MEM0SEL/, MEM1SEL/, and MEM2SEL/ select memory 0 (RAM), memory 1 (RAM), and memory 2 (ROM), respectively. Jumpers across these lines select address space 0000H to 1FFFH, 2000H to 3FFFH, 4000H to 5FFFH, or 6000H to 7FFFH; a given address space may be assigned to one and only one memory block. Figure 3-1 shows an example of memory configuration. In this example, address range 0000H to 1FFFH is assigned to memory 0, address space 2000H to 3FFFH is assigned to memory 2, and address space 4000H to 5FFFH is assigned to memory 1. Address space 6000H to 7FFFH is unassigned.

Address decoder U67 decodes the two address bits A13 and A14 to its four output lines: Y0 (designating address range 0000H - 1FFFH), Y1 (2000H - 3FFFH), Y2, (4000H - 5FFFH), or Y3 (6000H - 7FFFH). Address line A15 enables U67. Output Y0 is gated through U72 to allow reset signal RESETLCH/ to inhibit access to user-configurable memory addresses 0000H - 1FFFH during a reset operation (see the Reset Operation section later in this chapter).

The breakpoint logic allows program breakpoints to be established in one 8K block of user configurable memory. The jumper on the BRKPTSEL/ line determines the address space to which breakpoints can apply. In the example in Figure 3-1, breakpoints are assigned to the address range 0000H to 1FFFH, corresponding to memory 0's addresses.

RAM

1024 bytes (1K) of RAM is supplied with the SDK-51; additional RAM can be added in 1K units to fill the 16K of RAM positions available on the board.

One RAM address decoder is required for each 8K RAM block: U26 for block 0 and U51 for block 1. U26 is provided with the kit. MEM0SEL/ enables U26 and MEM1SEL/ enables U51. When enabled, the RAM decoders decode the three address lines A10, A11, and A12, enabling one of the eight 1K RAM segments in the 8K block. Each 1K segment consists of two 4096-bit (1Kx4) RAM devices (2114). Address lines A0 through A9 address the selected 1K of memory. The RAMWR/ line applied to the WE/ input on each device selects the read (high) or write (low) operation.

ROM

Up to 8K bytes of ROM can be installed in the user configurable ROM space (Memory 2) using two 4K ROM devices, up to 6K with 2K ROMs, or up to 3K with 1K ROMs. When jumpered by the user for the type of ROM being used, ROM jumper block U68 and decoder U67 decode the MEM2SEL/ memory select line and address lines A10, A11, and A12 to enable the appropriate ROM chip.

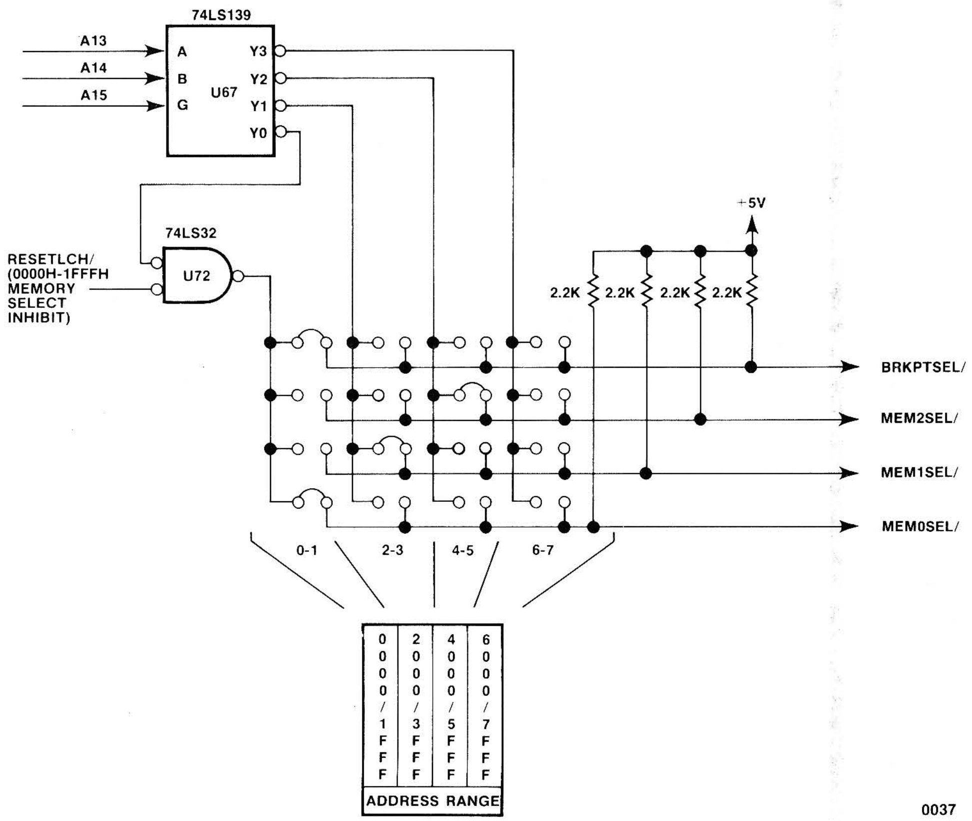


Figure 3-1. Memory Configuration Jumpers

Address lines A0 through A9 (A0 through A10 for 2K devices, A0 through A11 for 4K devices) select the byte to be read from the enabled ROM.

Jumper socket U68 allows jumper wires to be installed as required for the type of ROM devices used. Table 3-3 shows the required jumper configurations and maximum crystal frequencies for five different types of ROMs.

Table 3-3. ROM Jumper Configuration Chart

ROM/PROM:	3636		3628A		2716-1		2758		2732A	
	FROM	TO	FROM	TO	FROM	TO	FROM	TO	FROM	TO
	1	10	1	9	1	10	1	9	1	10
	2	13	2	11	2	7	2	11	2	7
	3	11	3	12	3	11	3	12	3	13
	4	12	4	10	4	12	4	10	4	11
	5	7,8	5	7,8	5	—	5	—	5	—
	6	9	6	13	6	8,9	6	7,8	6	8,9,12
	14	—	14	—	14	13	14	13	14	—
MAXIMUM CRYSTAL FREQUENCY	12 MHz		12 MHz		10 MHz		8 MHz		12 MHz	

Monitor

The monitor program is located in ROM in the upper 8K of the 64K external memory space (addresses E000H to FFFFH). Address decoder U38 receives address lines A12 through A15, pulling the MONLOSEL/ line low when the ExxxH address range is selected, and pulling MONHISEL/ low when the FxxxH address range is selected. A low on MONLOSEL/ enables ROM U59, while a low on MONHISEL/ enables ROM U60. Address lines A0 through A11 select the address within ROM to be read.

Reset Operation

A reset operation occurs at power on or when the two RESET keys on the keyboard are pressed simultaneously (pressing just one key has no effect). The RESET circuit generates two momentary signals, RESET and RESET/, and a latched status line, RESETLCH/.

On the SDK-51, the EA/ pin from the 8031 is tied to ground, causing external memory to be accessed. Upon a system reset, the 8031 microcontroller automatically attempts to access external memory location 0000H for its first instruction. During a reset, however, latch U83 in the Reset circuit pulls the RESETLCH/ line low. RESETLCH/ has two effects on memory selection: it disables accesses to user-configurable memory addresses 0000H through 1FFFH (through gate U72 in the Memory Configuration area), and it enables Monitor ROM 0 (U59). Thus, the first address read by the controller at reset is actually E000H in the Monitor rather than physical address 0000H.

The first instruction the monitor executes is a long jump instruction that moves the program counter to its actual address in high memory. Following this jump, the monitor performs a write to breakpoint memory; this sets the WRBRKMEM/ line low, which in turn pulls the RESETLCH/ line high. RESETLCH/ high allows normal access to the low 8K of user configurable memory.

(Other effects of RESET, RESET/, and RESETLCH/ are discussed in the sections to which they apply. For RESET, see Microcontroller, Parallel I/O Control, Serial I/O Control, and Breakpoint Control. For RESET/, see UPI Controller and Keyboard and Display Timing Generation. For RESETLCH/, see UPI Control and Breakpoint Control.)

6 MHz Clock Generator

The 6 MHz crystal Y2 drives an oscillator circuit to provide clock signals UPICLK and UPICLK/ for the UPI controller. Counter U40 divides UPICLK by 3 to generate SYSCLK, a 2 MHz timing signal for the Serial I/O Interface.

Parallel I/O Interface

A single Intel 8155-2 256-byte RAM with I/O ports and timer (U64) controls the parallel I/O ports. This device provides two general purpose 8-bit I/O ports and one 6-bit port that can be used either as a general purpose I/O port or as a status register to be operated in handshake mode.

The 8031 controller uses address range B000H through BFFFH to address the parallel I/O interface. When address lines A12 through A15 contain the value 0BH, address decoder U38 pulls the IOSEL/ line low, enabling U64 (see Address Bus Control earlier in this chapter). Address line A11 selects the mode of operation of

U64 by toggling the IO/M input to U64. When A11 is low, U64 operates as a memory device; when A11 is high, U64 operates as an I/O controller.

NOTE

> The memory space in the 8155 parallel I/O interface has been assigned to the monitor. Any attempt to access this space through a user program may interfere with the monitor's operation, causing a system malfunction.

When the I/O mode has been selected, address lines A0, A1, and A2 select the type of I/O operation to be performed. U64 receives this command information from the microcontroller (lines P00 through P02), and demultiplexes the address and data signals on these lines, using the ALE, RD/ and WR/ signals. On a high-to-low transition of ALE, U64 reads the address information from the bus to receive its command (e.g., select I/O port A, select command/status register); when ALE is low (before the next low-to-high transition) the controller instructs U64 either to transfer data to the bus (RD/ is low) or to accept data from the bus (WR/ is low).

The 8155 receives SYSCLK from the 6 MHz Clock Generator (the 6MHz UPICLK is divided by 3 to obtain the 2 MHz system clock) The timer output from U64 is BAUDCLK, the baud rate clock for the serial I/O interface. The rate for BAUDCLK can be selected by the user with a keyboard command (see BAUD command, Chapter 2).

The RESET signal initializes the 8155's three I/O ports to the input mode (see Reset Operation for the origin of the RESET signal). On reset, the BAUDCLK rate is set to 2400 baud.

Please refer to the *Intel Component Data Catalog* for details on the operation of the 8155-2.

UPI Control

The UPI control section (U41, U54) performs three functions in the SDK-51: it controls the input and output of commands and data between the 8031 controller and certain I/O peripheral devices (display, keyboard, serial I/O interface, cassette I/O interface); it scans the keyboard and maintains the display; and it provides control signals to the breakpoint logic and TOP circuit.

The UPI controller is an Intel UPI-41A. This device contains a CPU, a control program stored in ROM, a scratchpad RAM, three data bus registers, and two I/O ports. In addition to the UPI-41A, the UPI Control section includes an 8243 port expander (U54) to generate several of its outputs. The UPI Control section is diagrammed in Figure 3-2.

Data and commands are transmitted between the 8031 controller and the UPI controller through the data bus; control signals UPISEL/, A0, RD/, and WR/ are also involved in the communication. The UPI also receives timing signals UPICLK and UPICLK/ from the 6 MHz Clock Generator circuit, and the RESET/ signal from the Reset area.

The UPI in turn communicates with the peripherals through the UPIBUS, the DSPLYCLK timing signal, cassette data lines CASSIN and CASOUT, and signal RDKEYS/. Additional signals generated by or through this section are the TOP address lines; SSTEPEN/, DATABRKEN, PROGBRKEN and CLRBRK/ for the breakpoint logic,and UPIOBF for handshaking with the 8031 controller. Several of these signals are generated through the port expander U54 (8243).

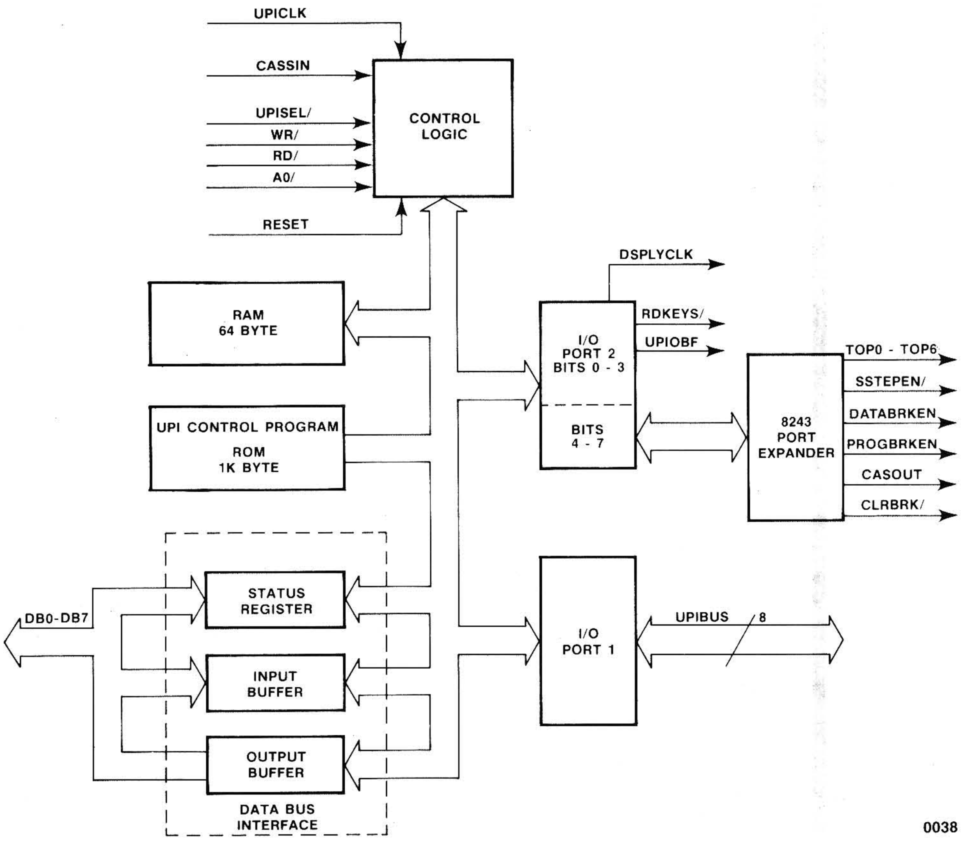


Figure 3-2. UPI Control, Block Diagram

The 8031 controller uses the address range A000H through AFFFH to address the UPI (U41). When the 8031 selects this address range (reflected as value 0AH on address lines A12 through A15), address decoder U38 pulls the UPISEL/ line low, enabling the UPI.

U41 has three externally addressable registers: the input buffer, the output buffer, and the status register. The 8031 writes both data and UPI controller commands to the input buffer; it reads data from the output buffer and reads status information from the status register. Address line A0, and the RD/ and WR/ signals, determine which of these functions to perform, as shown in Table 3-4. When the 8031 writes a command to the input buffer, the UPI performs the operation by referencing code in its internal control ROM. (See the *UPI-41A User's Manual* (referenced in the Preface) for more on the UPI commands.)

Table 3-4. 8031 Interface with UPI Controller

Address	WR/	RD/	Host Activity
Axx1H	0	1	Write data to input buffer
Axx1H	1	0	Read data from output buffer
Axx0H	0	1	Write UPI command to input buffer
Axx0H	1	0	Read UPI status register

In operation, the 8031 controller issues a command to the UPI controller, indicating the peripheral device it wishes to access. The 8031 then sends data to or receives data from the selected device, using the UPI status register and the 8031's interrupt logic for handshaking.

I/O port expander U54 decodes four I/O port lines from the UPI controller into 15 (out of 16 possible) output lines. Signals TOP0 through TOP6 are used by the Top of Program Memory Protect area. SSTEPEN/, DATABRKEN, and PROGBRKEN are used by the Breakpoint logic. An additional breakpoint signal from the 8243, CLRBRK/, can also be generated by a low on RESETLCH/. CLRBRK/ is used to clear the breakpoint logic (see Breakpoint Control and Top of Program Memory Protect Circuit for details on CLRBRK/, and Reset Operation for the origin of RESETLCH/). The remaining signal, CASOUT, is the serial data line to the cassette I/O interface.

The RESET/ signal to the UPI controller places the 8041A/8741A in its initial state, and it begins executing its internal program from the beginning. See Reset Operation for the origin of RESET/.

Keyboard and Display Timing Generation

Shift registers U80, U81, and U82 strobe the 24 KDTIME lines at a rate of approximately 60 times a second. At KDTIME0, a 1 is shifted from the UPIBUS6 line into the A input of U80, pulling the output at Q_A high. On each successive DSPLYCLK cycle, a 0 is shifted into input A, so that the 1 is shifted to each KDTIME line in turn. On the 25th clock cycle, a 1 is again input. The KDTIME lines are used to scan the keyboard and the LED display modules as discussed in the next two sections.

A RESET/ signal from the Reset circuit clears the KDTIME lines so that the sequence can begin again from KDTIME0. See Reset Operation for the origin of RESET/.

Keyboard Control

Strobe lines KDTIME0 through KDTIME6 scan the columns of the keyboard matrix, one KDTIME line being active (high) on each clock cycle. The rows of the matrix, KR0 through KR7, are connected through 8-bit buffer U86 to the UPIBUS lines. The UPI-41A enables U86 to drive KR0 through KR7 on the UPIBUS by pulling the RDKEYS/ line low. Through this system the UPI scans the keyboard matrix approximately 60 times a second. If the UPI detects the same key closed on two consecutive sweeps, it transmits the ASCII code for that key position to its output buffer and sends an interrupt to the host via the UPIOBF line. The UPIOBF line generates an interrupt over the same line (P32) used by the breakpoint logic; UPIOBF is also placed on the data bus (DB1) when RDBRKMEM/ goes low.

Display Control

The SDK-51's 24-character display consists of three 8-character modules, DS1, DS2, and DS3. Each character is formed by illuminating a pattern of segments on an 18-segment LED readout. The displayable character set consists of a subset of ASCII (see Chapter 2 for details).

Converter U2 on the display board translates ASCII code on UPIBUS0 through UPIBUS5 into the appropriate combination of segments on display modules DS1, DS2, and DS3 to create the selected character. The lines KDTIME0 through KDTIME23 enable the characters one at a time. A low on DSPLYCLK enables the input to U2 and, through one-shot U5, disables the output. On the rising edge of DSPLYCLK, the ASCII code on the UPIBUS lines is latched into U2, and U5 is triggered to enable output from U2.

Each character in the display is illuminated 1/24 (about 4%) of the time. The duration of each KDTIME pulse (equal to the DSPLYCLK high time) is about 700 microseconds. The entire display is refreshed approximately 60 times a second (700 microseconds times 24 equals about 16.8 milliseconds or approximately 1/60 of a second). To protect the LED readouts, U5 will disable the output if a circuit malfunction causes one of the LED characters to be selected for more than approximately 760 microseconds (i.e., DSPLYCLK remains high).

Serial I/O Interface

The Intel 8251A Programmable Communications Interface, U29, controls the serial I/O interface in the SDK-51. U29 operates in an asynchronous mode.

BAUDCLK, the baud rate clock input to the 8251A, is generated by the parallel I/O interface device, U64 (the 8155). The baud clock rate is user-selectable from 110 to 9600 baud, as described in chapter 3 (Baud command). Upon reset, the baud rate is set to 2400 baud.

The SDK-51 includes devices for both standard RS-232 and current loop interfaces. Jumpers on the board select one or the other interface standard. In addition, the 8031 controller's serial I/O pins can be jumpered to connect to the RS-232 or current loop interface.

The UPI controller communicates with U29 through the UPIBUS lines, and in addition uses port expander U54 to provide inputs to the RD/, WR/, and C/D inputs on U29. (See the *Intel Component Data Catalog* for details on the 8251A and its command set.

The RESET signal (see Reset Operation) places the 8251A in its initial "idle" state.

Audio Cassette Interface

The audio cassette interface allows the 8031 to read data from or write data to an audio cassette tape recorder. The 8031 communicates with the cassette interface through the UPI controller. The user selects the direction of data transfer, and the UPI performs the serial-to-parallel or parallel-to-serial conversion.

Data to be transmitted to the cassette recorder comes over the CASOUT line from the UPI. Data received from the recorder goes to the UPI over the CASSIN line.

The transmitted data is formatted at the byte level, and again at the bit level. At the byte level, the UPI begins the byte stream with a leader tone approximately 10 bits in duration. Then it sends a byte of eight data bits followed by another 10-bit leader tone, repeating leader and data "fields" until all the data has been sent (Figure 3-3).

At the bit level, bits are encoded in the CASOUT signal with a four-part, software generated data cycle. As diagrammed in Figure 3-4, the four divisions of one bit time consist of a START period (1/4 bit time), the DATA (middle 2/4 bit time), and a STOP period (1/4 bit time). The START period always contains a burst of tone; the STOP period is always "silent" (no tone). The DATA period in the middle half of the bit time can be a tone (data 1) or silent (data 0).

When the encoded data is read back from the cassette, shaping circuits invert, filter, and square the tone bursts, creating the CASSIN signal. Figure 3-4 shows the CASSIN waveform that corresponds to the CASOUT signal in the figure. The UPI then decodes the four-part data cycles using a sampling technique, and converts the serial data to parallel bytes, which can be transmitted to the 8031 over the data bus.

NOTE

The keyboard and display are disabled during cassette operation. The display is blanked. No keys are detected; in particular, the ESC key is not recognized.

Please Refer to Audio Cassette Interface in Chapter 2, for details on connecting and controlling the audio cassette interface.

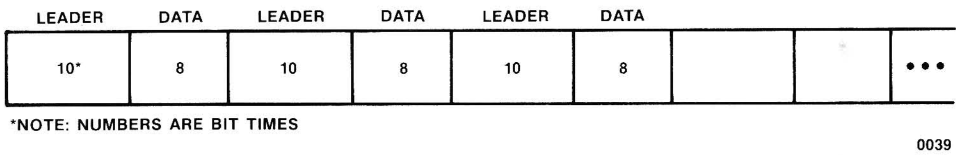


Figure 3-3. Cassette Output Byte Stream

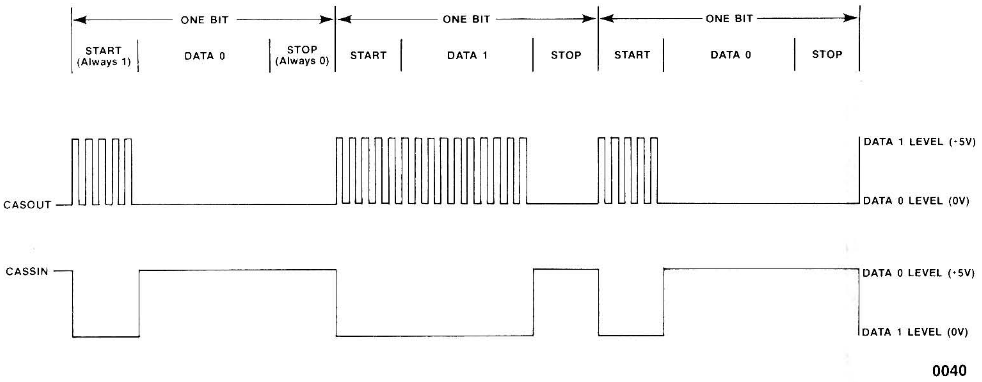


Figure 3-4. Cassette Interface Waveforms

Top of Program (T.O.P.) Memory Protect Circuit

The user can write-protect a portion of user-configurable memory that contains the user program, by specifying the address that is the top of program ("T.O.P.") (see Chapter 2, Top of Program Memory Command). The write protected segment always begins with address 0000H, and ends with the T.O.P. address. The TOP0 through TOP6 signals from the UPI area contain the T.O.P. address set by the user. U39 and U53 in the T.O.P. circuit compare address lines A8 through A15 with TOP0 through TOP6. If the address on the bus is below the top of memory address, the RAMWR/ line is inhibited from going low (write).

If the user program attempts to write to write-protected memory, the GUARDEDACC line is latched high. GUARDEDACC generates an interrupt to the 8031 controller via breakpoint signal P32 (breaking user program execution). A low on the CLRBRK/ line resets the GUARDEDACC line low; as discussed in the UPI Control section, CLRBRK/ is generated either by a reset or by command from the 8031 via the UPI.

Breakpoint Control

The breakpoint logic circuit allows the user to specify addresses in either program memory or data memory as breakpoint addresses; if the user program accesses a breakpoint address, the breakpoint logic sends an interrupt to the 8031 controller, breaking program execution. In addition, the breakpoint logic is used to break execution after each instruction in single-step. Figure 3-5 shows a simplified diagram of the breakpoint logic. (See Breakpoint Commands and STEP Command in Chapter 2.)

U74 latches breakpoint signals PROGBRK (break on program address), DATABRK (data address), and SSTEP (break after each instruction in single-step). Gating logic combines these signals to generate the interrupt to the 8031 (P32). U74 also generates the ONECYCLOP signal discussed later in this section.

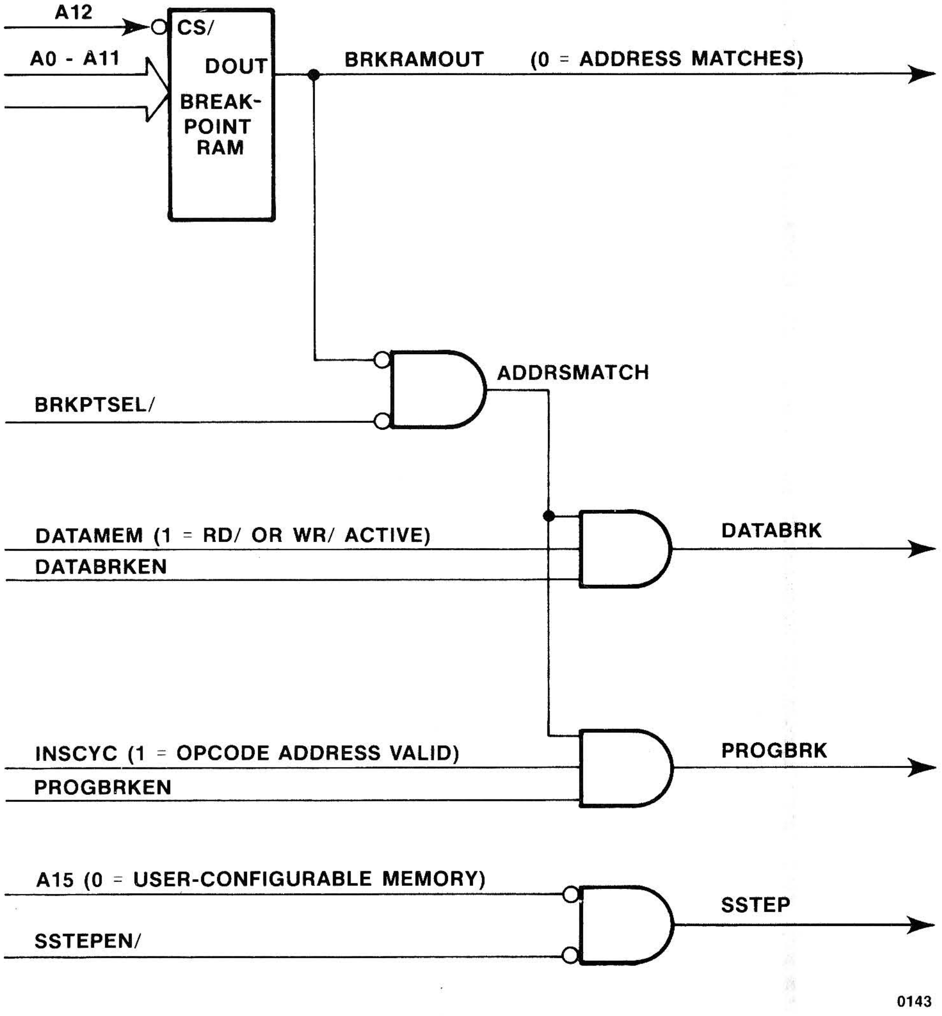

A12
A0 - A11
CS/
DOUT
BREAK-POINT RAM
BRKRAMOUT (0 = ADDRESS MATCHES)
BRKPTSEL/
ADDRSMATCH
DATAMEM (1 = RD/ OR WR/ ACTIVE)
DATABRKEN
DATABRK
INSCYC (1 = OPCODE ADDRESS VALID)
PROGBRKEN
PROGBRK
A15 (0 = USER-CONFIGURABLE MEMORY)
SSTEPEN/
SSTEP
0143

Figure 3-5. Breakpoint Logic, Simplified

Single-Step Breaks

When the SSTEPEN/ line from the UPI is active (low) and address line A15 is also low (indicating an access to user-configurable memory), a breakpoint interrupt is generated after every instruction executed. This operation does not involve the setting of breakpoint RAM. The signal SSTEP through U74 is gated onto signal P32 (processor INT0 input).

Breakpoint RAM

The breakpoint addresses for program breaks and data breaks are represented by a special breakpoint memory (U61, U62), consisting of one or two 4K x 1 bit RAMs (2141-5), or 8K bits maximum. (One RAM device is provided with the kit; the other may be added at the user's option.) Each bit in breakpoint memory represents a byte in user-configurable memory (which can be either program or data memory). The breakpoint memory can be mapped to represent any of the 8K blocks of user-configurable memory, by setting a jumper on the BRKPTSEL/ line (see Memory Configuration earlier in this chapter).

The 8031 controller uses addresses C000H through DFFFH in memory to address the breakpoint memory. To read or write breakpoint memory, U38 decodes address lines A12 through A15, generating a low either on LOBRKMEMSEL/ (C000H - CFFFH) or on HIBRKMEMSEL/ (D000H - DFFFH), as described earlier in the section on Address Bus Control. These two signals are then combined with the RD/ and WR/ lines to create the signals RDBRKMEM/ and WRBRKMEM/. When WRBRKMEM/ is low, data can be written to breakpoint memory. The breakpoint address is on lines A0 through A11, and the bit value is on data bus line DB0: 0 = set breakpoint, 1 = clear breakpoint.

Breakpoint memory is read each address cycle, except when WRBRKMEM/ is low. The breakpoint memory outputs the bit value of the current address on BRKRAMOUT; a low on BRKRAMOUT indicates an address that has a breakpoint set on it; this signal is combined with BRKPTSEL/ to generate signal ADDRSMATCH (since this line is internal to the breakpoint logic, it is not shown on the functional block diagram).

Data Memory Breaks

For data memory breaks, ADDRSMATCH is combined with DATABRKEN (from the UPI) and DATAMEM (from the Data Bus Control), to latch the DATABRK signal high.

The DATABRK signal is pulled high when the following three conditions are all true at the same time: (1) data memory is being read or written (DATAMEM is high); (2) breaks on data memory have been enabled (DATABRKEN high); and (3) the current address matches a breakpoint address (ADDRSMATCH high). The DATABRK line passes through U74 and generates an interrupt to the host controller at its P32 (INT0) input.

Program Memory Breaks

For program memory breaks (PROGBRK signal), ADDRSMATCH is combined with signals PROGBRKEN and INSCYC. PROGBRK is pulled high when the following three conditions are true at the same time: (1) breaks on program memory have been enabled; (2) the instruction cycle is valid; and (3) an address match occurs between the current program memory address and breakpoint memory. The PROGBRK line from U74 generates an external interrupt to the controller at P32 (INT0).

Instruction Cycle (INSCYC) Circuit

Look-up memory U63 is a ROM that contains a list of the number of bus cycles required to complete each instruction in the 8051 microcontroller instruction set. Each time the 8031 receives an instruction from program memory, U64 receives the instruction through the data bus, looks up the number of bus cycles required to complete the instruction, and transfers this count to counter U77. The counter then counts bus cycles (CMD/ low-to-high transition). Upon reaching the loaded count, U77 pulls the LOADINSCTR/ line low on the next bus cycle. LOADINSCTR/ and initialization latch U83 are combined to control the INSCYC signal. A high on INSCYC indicates that the current fetch is the address of the first byte of a valid instruction.

This circuit is provided to prevent the breakpoint logic from breaking on a memory fetch that is not a valid instruction. The 8051 always fetches two bytes on every cycle, even though the two bytes may not be part of the same instruction. If an opcode is fetched as the second byte of a cycle, therefore, it has not yet been executed, and thus is invalid as a breakpoint. The lookup ROM and counter assure that program breaks will only occur when an instruction that matches a breakpoint address is actually executed, not merely "pre-fetched."

NOTE

During any interrupt, the INSCYC circuit loses synchronization with the instructions executed. To resynchronize, a data memory reference is used to clear the U77 counter (program breaks are inhibited by U83 and U84 during this fetch). The next instruction fetch starts the counter at the right point again.

FORCENOP Circuit

The 8031 processor samples interrupt pin INT0 (P32) on the falling edge of the second ALE on any instruction cycle. The processor then delays one Tcyc before responding to the interrupt. If a "new" instruction begins execution during the one Tcyc delay, that instruction will be completed before the controller responds to the interrupt. The problem for the breakpoint logic is that this "new" instruction could affect register or data memory contents, including the program counter.

The solution is to cause the "new" instruction to be a NOP instruction, which only affects the program counter (PC is decremented to the correct break value by the Monitor). U74 in the breakpoint logic latches signal ONECYCLOP high to indicate that the instruction being fetched is a one-cycle instruction and will therefore fall through to a new instruction prior to servicing the interrupt.

There are three cases:

1. (PROGBRK or SSTEP) = 1, and ONECYCLOP = 0. In this case, the interrupt occurs at the end of the desired instruction, and all registers are valid.
2. (PROGBRK or SSTEP) = 1, and ONECYCLOP = 1. In this case, the "new" instruction must be forced to be a NOP. All registers then are valid except the program counter, which must be decremented by one.
3. DATBRK = 1 and ONECYCLOP = 0 or 1 (don't care). In this case the "new" instruction must be forced to be a NOP and the program counter must be decremented by one.

To force a NOP on the data bus, counter U55 generates a signal FORCENOP under the cases outlined above. This signal is received by the FORCENOP circuit in the Data Bus Control area; device U42 is enabled and puts a byte of all zeros on the bus.

Clearing a Break Condition

When the breakpoint logic generates an interrupt, the 8031 controller discontinues its current processing task to service the interrupt. Upon completion, the monitor interrupt service routine generates a low on the CLRBRK/ line through the UPI controller. A low on CLRBRK/ resets the breakpoint signal (SSTEP, DATABRK, or PROGBRK), clears the interrupt line (P32), and resets FORCENOP counter U55. CLRBRK/ also resets the GUARDEDACC line from the Top of Memory circuit.

A system reset also clears the breakpoint logic. The RESET signal clears cycle counter U77 and turns off the INSCYC line. The RESETLCH/ line sets CLRBRK/ active (low) when it is latched low by the Reset circuit (see Reset Operation).

Reading The Cause of the Last Break

Breakpoint signals BRKRAMOUT, GUARDEDACC, PROGBRK, DATABRK, SSTEP, and ONECYCLOP (also UPI status line UPIOBF and the RESETLCH/ signal) are connected to the data bus so they may be interrogated. The data bit set uniquely identifies the cause of the last break in program execution (see CAUSE Command in Chapter 2). Signal RDBRKMEM/ enables unidirectional drivers U69 and U70 to place these signals on the bus (see Table 3-5 for details).

NOTE

During the process of servicing a break, signal CLRBRK/ prevents the user from reading these values directly with a memory reference command. However, the CAUSE command (Chapter 2) provides the information.

Table 3-5. Breakpoint Signals on the Data Bus

Data Bus Line	Breakpoint Signal
DB0	BRKRAMOUT
DB1	UPIOBF
DB2	GUARDEDACC
DB3	PROGBRK
DB4	DATABRK
DB5	SSTEP
DB6	ONECYCLOP
DB7	RESETLCH/

Glossary of Signal Lines

This section contains brief descriptions of the signal lines shown on the SDK-51 schematic drawings or on the block diagram (Figure 3-6). Most of these signals are discussed in the previous sections of this chapter.

ADDRSMATCH — High indicates that breakpoints have been assigned to the block of user configurable memory being read and that a breakpoint has been assigned to the address being accessed.

ALE
Address Latch Enable. Demultiplex signal for address/data lines. When ALE = 1, P00 - P07 contains the low address bits; when ALE = 0, P00 - P07 contains the data byte. In most cases, the low address byte is latched on the high-to-low (trailing) edge of ALE, but details of timing vary with application.

ANYBRK
Logical "OR" combination of SSTEP, PROGBRK, and DATABRK, used by the Breakpoint logic to control the output of these signals and the FORCENOP signal.

A0 - A15
Address bus; high true.

BAUDCLK
Clock signal from timer in parallel I/O interface that sets the baud rate of the serial I/O interface.

BPSEN/
Buffered PSEN/ line.

BRKPTSEL/
Low indicates a read from the 8K byte block of user configurable memory to which the breakpoint logic has been assigned.

BRKRAMOUT
Output of the breakpoint RAM; low indicates that a breakpoint address has matched an address accessed by the program.

CASOUT
Serial output from the UPI controller to an audio cassette recorder.

CASSIN/
Serial input data from audio cassette recorder to UPI controller.

CTS/
Serial I/O (RS232) interface CLEAR TO SEND signal.

CLRBRK/
Low resets interrupt circuitry following an interrupt resulting from a breakpoint or guarded access interrupt.

CMD/
Control input to breakpoint logic. Signals activation of BPSEN/, WR/, or RD/.

CPU SERIAL DATA
Serial data received and transmitted, routed from serial I/O to CPU for use by customer program (optional).

CUR LP RX DATA (-)
Serial (TTY) I/O Interface signal (CURRENT LOOP RECEIVED DATA).

CUR LP RX DATA (+)
Serial (TTY) I/O interface signal (CURRENT LOOP RECEIVED DATA).

CUR LP TX DATA (+)
Serial (TTY) I/O interface signal (CURRENT LOOP TRANSMITTED DATA).

CUR LP TX DATA (-)
Serial (TTY) I/O interface signal (CURRENT LOOP TRANSMITTED DATA).

DATABRK
High indicates breakpoint interrupt on access to data memory.

DATABRKEN/
Low enables breakpoint logic for accesses to data memory.

DATAMEM
High indicates data memory is being accessed.

DB0 - DB7
Data bus; high true.

DSPLYCLK
Clock signal that UPI controller generates for use in scanning the keyboard and LED display modules.

EARPHONE Audio cassette I/O input (comes from EARPHONE output on cassette recorder).

FORCENOP Control input to data bus control from breakpoint logic. Forces a byte of 0's on the data bus (U42)

GUARDEDACC/ Low indicates a write has been attempted in the write protected section of memory; causes an interrupt to occur.

HIBRKMEMSEL/ Low selects upper half of breakpoint RAM in order to write breakpoints.

INSCYC High indicates valid instruction fetch cycle (used by Breakpoint logic).

IOSEL/ Low selects the parallel I/O interface device and inhibits data bus control.

KDTIME0 - KDTIME23 24 enable lines used to multiplex data to the LED display modules and to scan the keyboard; high is true.

KR0 - KR7 Low on one of these lines indicates that there is a key closure on the associated row of the keyboard matrix.

LOADINSCTR/ Low indicates that instruction cycle counter in breakpoint logic has reached its preset count and is ready for the next count to be loaded (i.e., next cycle is the beginning of the next instruction).

LOBRKMEMSEL/ Low selects lower half of breakpoint RAM in order to write breakpoints.

MEM0SEL/ Output from Memory Configuration area, selects Memory 0 when low.

MEM1SEL/ Output from Memory Configuration area, selects Memory 1 when low.

MEM2SEL/ Output from Memory Configuration area, selects Memory 2 when low.

MICROPHONE INPUT Audio cassette I/O output (goes to MIC input on cassette recorder).

MONHISEL/ Enables high 4K of Monitor ROM when low.

MONLOSEL/ Enables low 4K of Monitor ROM when low.

ONECYCLOP Control output from breakpoint logic. Indicates a one-cycle instruction was executing when a breakpoint was encountered; this forces a NOP on the data bus.

P00-P07, P20-P27 Address/Data ports from 8031 Microcontroller. Addresses and data are multiplexed through these ports each bus cycle: address followed by data.

P10 - P17 Microcontroller port 1 pins. Can be used to scan auxiliary keypad or for other optional uses. Requires user to generate program for the 8031.

P30 Serial data output from microcontroller. Can be used by customer program.

P31 Serial data input to microcontroller. Can be used by customer program.

P32

Logical "OR" of GUARDEDACC, UPIOBF, SSTEP, PROGBRK, and DATABRK sent from breakpoint logic to INT0 input on CPU. (also shown as "INTERRUPT" on functional block diagram.)

P33

INT1 interrupt input to 8031 microcontroller. Can be jumpered to connect INTR key to INT 1 input. Can be jumpered for other use by customer designed circuit.

PA0-PC5

Output lines from the Parallel I/O control.

PROG/ALE

See ALE (PROG applies only to the 8751 version of the microcontroller family).

PROGBRK

High indicates breakpoint interrupt on read from program memory.

PROGBRKEN/

Low enables breakpoint logic for accesses to external program memory.

PROGMEM

High indicates external program memory is being accessed.

PSEN/

Low indicates 8031 controller is reading from (external) program memory.

RAMWR/

Low enables RAMs for write operation. Logical AND of WR/ and output of T.O.P. address comparator; low indicates that address to be written to is above write protected area.

RD/

Low indicates a read from data memory. The line is the buffered read output from the 8051 Microcontroller.

RDBRKMEM/

Low causes BRKRAMOUT, UPIOBF, GUARDEDACC, ONECYCLOP, PROGBRK, DATABRK, SSTEP and RESETLCH/ lines to be connected to the data bus so they can be checked.

RDKEYS/

Low enables buffer U86 to drive KR0 through KR7 from the keyboard onto the UPIBUS.

RECEIVED DATA

Serial I/O (RS-232) interface signal (Note: The data on this line is received by the equipment connected to the line external to the SDK. Internal to the SDK, this line is TxD, transmitted data. It is shown on the block diagram as "RX" DATA to emphasize the reversal of names.

REQUEST TO SEND

Serial I/O (RS-232) interface signal. Same as RTS/.

RESETLCH/

Disables user configurable memory and selects lower half of monitor ROM when a system reset is initiated. Assures the program start begins at beginning monitor address (E000H).

SERIAL DATA/

Control line from UPI controller to 8251A = 0, UPI bus contains data for the serial I/O interface; when 1, bus contains a command for the 8251A.

SERIAL RD/

Control line from UPI to 8251A serial I/O control. When SERIAL RD/ = 0, 8251A is enabled to transfer a byte of data to the line.

SERIAL WR/ — Control line from UPI to 8251A serial I/O control. When SERIAL WR/ = 0, 8251A is enabled to transfer a byte of data from the line.

SSTEP — High indicates breakpoint interrupt on read to memory due to single step operation.

SSTEPEN/ — Low enables single step function (generates break in execution after each instruction).

SYSCLK — 2 MHz timing signal to Parallel I/O and Serial I/O areas from 6MHz clock generator (through divide by 3 counter).

TOP0 - TOP6 — Contain the uppermost address of the write-protected portion of user-configurable memory as specified by the user. Write-protected area starts with address 0000H.

TRANSMITTED DATA — Serial I/O (RS-232) interface signal. Note: the data on this line is transmitted by the equipment connected to the line external to the SDK. Internal to the SDK, this line is named RxD, Received Data. On the block diagram, this line is shown as "TX" DATA to emphasize the reversal in names.

TTLHI1, TTLHI2 — TTL HIGH inputs to integrated circuits.

UPIBUS0 - UPIBUS7 — Lines that transmit data and commands between UPI controller and the peripherals it controls.

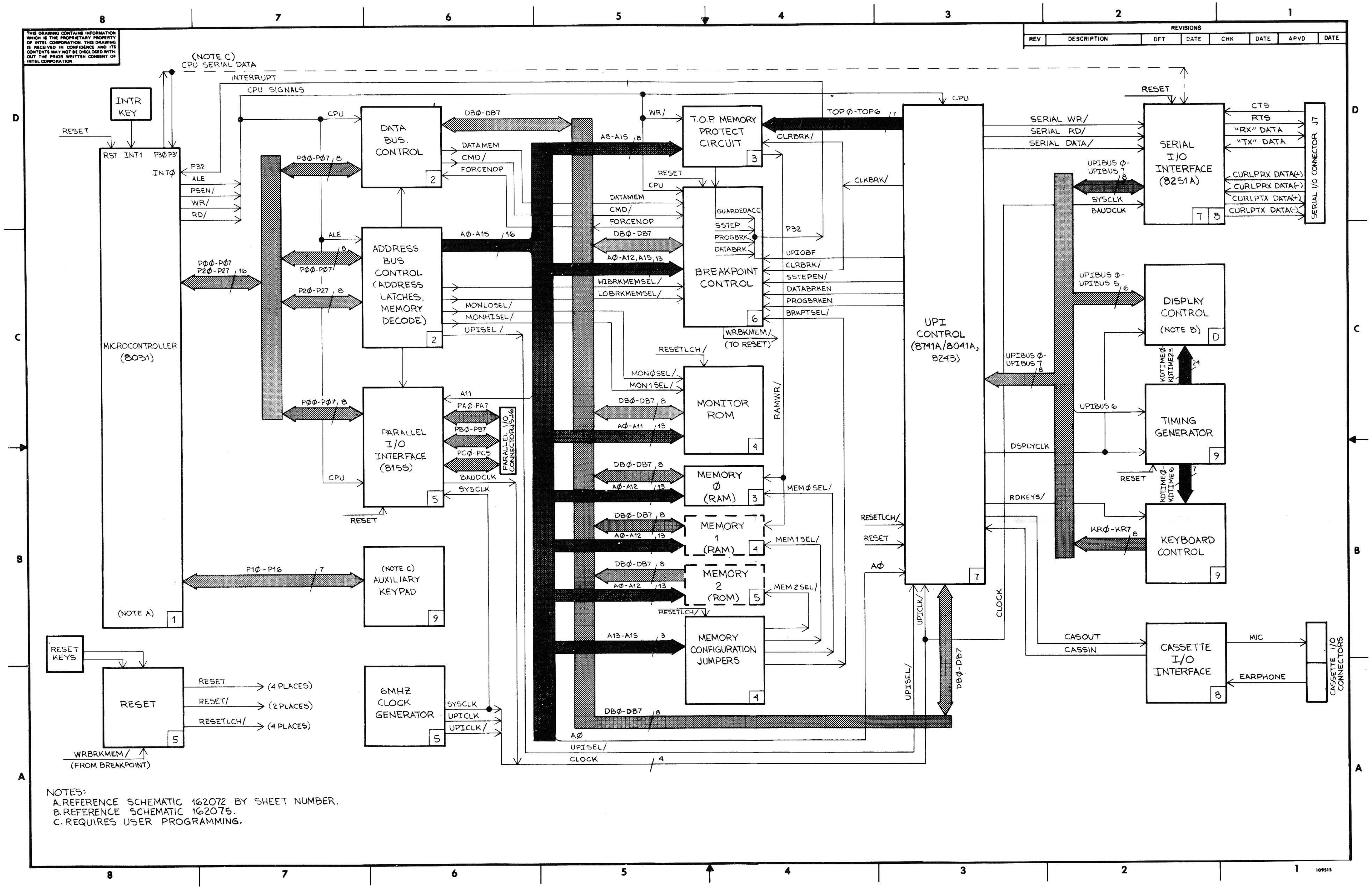
UPICLK, UPICLK/ — Timing signals from 6MHz clock generator to UPI controller.

UPIOBF — High causes an interrupt (through the breakpoint logic) to the 8031 microcontroller to request the 8031 to read the output buffer of the UPI controller.

UPISEL/ — Low selects UPI controller device.

WR/ — Low enables a write from the data bus to memory or I/O. The line is the buffered write output from the 8031 microcontroller.

WRBRKMEM/ — Low enables write to breakpoint memory; low also resets RESETLCH/ line to high.



NOTES:
A. REFERENCE SCHEMATIC 162072 BY SHEET NUMBER.
B. REFERENCE SCHEMATIC 162075.
C. REQUIRES USER PROGRAMMING.

Figure 3-6. Breakpoint Logic, Simplified

CHAPTER 4 APPLICATIONS

This chapter contains guidelines for writing programs to run on the SDK-51, for using the on-board parallel I/O ports and auxiliary keypad, and for expanding the hardware capability of the board. The information includes:

- Ways to call Monitor utility routines from within the user program
- Special considerations for interrupt routines.
- Using the 8031 on-chip UART (serial I/O port)
- Parallel I/O interfacing
- Using the on-board auxiliary keypad
- Adding circuitry to the breadboard area
- Expanding the SDK-51 memory capacity

Accessing Monitor Utilities

Several of the utility routines in the SDK-51 system monitor program can be called by the user program. The routines are accessed via a jump table near the beginning of the monitor. These routines are:

UCI	Reads a character from the SDK-51 main keyboard.
CO	Displays a character on the SDK-51 display.
NEWLINE	Clears the display (outputs a RETURN to the display).
PRINT_STRING	Displays a message on the display.
LSTBYT	Displays a byte value on the display.
LSTWRD	Displays a two-byte value on the display.
UCSTS	Returns console status bit (1 = character is waiting to be read).
TIME	Delays 100 microseconds times the 16-bit value passed to the routine.

NOTE

Most of the monitor utility routines described in this section affect the accumulator and registers R2 and R3 in register bank 0. See the SDK-51 Monitor Listing Manual for details.

Console Input (UCI)

Routine UCI reads a character from the main keyboard, and places the ASCII value (with bit 7 masked to zero) in the accumulator. To call this routine from the user program, the code is:

```
LCALL 0E009H
```

No other parameters are required. The routine waits until a valid key has been pressed on the keyboard, then reads the ASCII value from the UPI-41A. Bit 7 is masked to zero, then the 7-bit value for the key combination is moved into the accumulator of the 8031. Table 4-1 lists the key combinations and corresponding ASCII values as returned by the console input routine.

Table 4-1. Console Input Values

KEY LABEL	KEY ONLY	KEY AND SHIFT	KEY AND CONTROL
A	61H	41H	01H
B	62H	42H	02H
C	63H	43H	03H
D	64H	44H	04H
E	65H	45H	05H
F	66H	46H	06H
G	67H	47H	07H
H	68H	48H	08H
I	69H	49H	09H
J	6AH	4AH	0AH
K	6BH	4BH	0BH
L	6CH	4CH	0CH
M	6DH	4DH	0DH
N	6EH	4EH	0EH
O	6FH	4FH	0FH
P	70H	50H	10H
Q	71H	51H	11H
R	72H	52H	12H
S	73H	53H	13H
T	74H	54H	14H
U	75H	55H	15H
V	76H	56H	16H
W	77H	57H	17H
X	78H	58H	18H
Y	79H	59H	19H
Z	7AH	5AH	1AH
!/1	31H	21H	31H
"/2	32H	22H	32H
#/3	33H	23H	33H
$/4	34H	24H	34H
%/5	35H	25H	35H
&/6	36H	26H	36H
'/7	37H	27H	37H
</8	38H	28H	38H
>/9	39H	29H	39H
\/0	30H	5CH	30H
=/-	2DH	3DH	2DH
@	40H	NONE	NONE
+/;	3BH	2BH	3BH
*/:	3AH	2AH	3AH
[/,	2CH	5BH	2CH
]/.	2EH	5DH	2EH
?//	2FH	3FH	2FH
ESC	1BH	NONE	NONE
TAB	09H	NONE	NONE
CNTRL	N/A	N/A	N/A
SHIFT	N/A	N/A	N/A
RETURN	0DH	NONE	NONE
RUBOUT	7FH	NONE	NONE
SPACEBAR	20H	NONE	NONE

Console Output (CO)

The console output routine takes the 7-bit ASCII byte passed in register R2, converts it to a six-bit value for the UPIBUS, and displays the corresponding character on the display. To call this routine from the user program, the sequence is:

```
MOV R2,byte         ; Immediate, direct, or indirect.
LCALL 0E006H        ; Calls the monitor routine
```

The values for the characters are shown in Figure 4-1. The byte passed to R2 can be immediate data, a direct data address, or an indirect address.

For example, to display the character X:

```
MOV R2,#58H
LCALL 0E006H
```

As a more useful example, to clear the display then display the character read in with the console input routine:

```
MOV R2, #0DH        ; Carriage return character.
LCALL 0E006H        ; Output the return.
MOV R2, #0AH        ; Linefeed character.
LCALL 0E006H        ; Output the linefeed.

LCALL 0E009H        ; Inputs keyboard character into ACC.
MOV R2,A            ; Passes character to output routine.
LCALL 0E006H        ; Outputs character to display.
```

See NEWLINE in the next section for an easier way to clear the display.

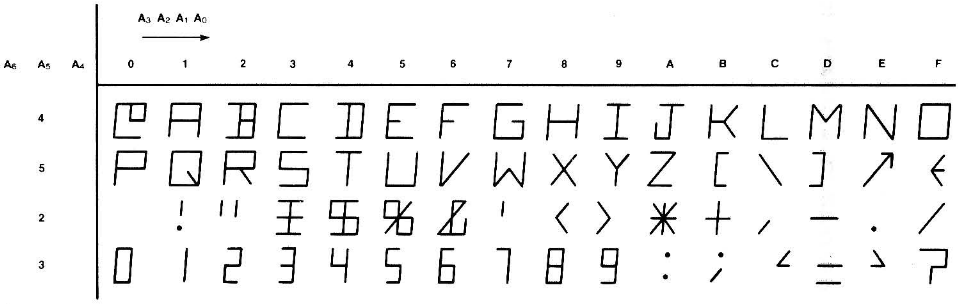


Figure 4-1. Console Output Values

Clear Display (NEWLINE)

To guarantee that the display is clear before displaying a character, message, or number, insert a call to the NEWLINE routine as follows:

```
LCALL 0E00FH
```

NEWLINE outputs a carriage return and linefeed to the display.

No parameters are required.

Display a Message (PRINT_STRING)

Routine PRINT_STRING displays a message stored as ASCII bytes in external memory. To use this routine, first load the string into memory. The first byte of the string must contain the number of characters in the string; the length can be any value from 00H through 0FFH. Figure 4-1 shows the ASCII values to use for the characters to be displayed.

To call the routine, move the high byte of the string's address into R2 and the low byte of the address into R3, then call the routine. The sequence in general is:

```
MOV R2,byte          ; High byte of string address.
MOV R3,byte          ; Low byte of string address.
LCALL 0E01EH         ; Call the routine.
```

The bytes passed to R2 and R3 can be immediate data, direct data addresses, or indirect addresses.

For example, suppose we wish to have the user program display the sign-on message READY (five characters). First, we load the string with that message into data memory from the console starting at address 300H:

```
XBY 300 = 5, 52,45,41,44,59
```

To display this message, the user program contains the following code sequence:

```
LCALL 0E00FH         ; Clear the display.
MOV R2,#03H          ; High byte of string address.
MOV R3,#00H          ; Low byte of string address.
LCALL 0E01EH         ; Displays the message.
```

CAUTION

Do not attempt to single step through a call to PRINT_STRING. Normally, the system hardware treats any access to system memory (above 7FFFH) as one step, without breaks between instructions. When PRINT_STRING attempts to read the string (in user-configurable memory), the hardware tries to resume single-stepping, with unpredictable results. To restore proper system operation, press both RESET keys.

Display a One-Byte Number (LSTBYT)

The LSTBYT routine displays the byte in register R2 as hexadecimal digits (using ASCII characters). The set-up and call to this routine is:

```
MOV R2, byte         ; Move one-byte value into R2.
LCALL 0E015H         ; Call the routine.
```

The byte passed to R2 can be immediate data, a direct data address, or an indirect address.

For example, to display the value of the accumulator in hexadecimal:

```
LCALL 0E00FH         ; Clear the display.
MOV R2,A             ; Copy the accumulator to R2.
LCALL 0E015H         ; Display the value.
```

Display a Two-Byte Number (LSTWRD)

Routine LSTWRD uses the value in register R2 as the high byte and the value in R3 as the low byte of a 16-bit number, and displays that number in hexadecimal (using ASCII characters). The call sequence is:

```
MOV R2, byte         ; High byte of number.
MOV R3, byte         ; Low byte of number.
LCALL 0E018H         ; Call to display routine.
```

The bytes passed to R2 and R3 can be immediate data, direct data addresses, or indirect addresses.

For example, to display the number A123H:

```
LCALL 0E00FH        ; Clear the display.
MOV R2,#0A1H        ; High byte.
MOV R3,#23H         ; Low byte.
LCALL 0E018H        ; Display hexadecimal value.
```

Read Console Status (UCSTS)

The UCSTS routine sets the carry flag to 1 if the UPI has a character waiting to be read by the 8031, or to 0 if no character is waiting. The call is:

```
LCALL 0E00CH
```

No parameters are required.

Time Delay (TIME)

Routine TIME introduces a delay in program execution. The duration of the delay is 100 microseconds multiplied by the 16-bit parameter passed to the routine in registers R2 (high byte) and R3 (low byte). The general sequence is:

```
MOV R2, byte        ;High byte of delay parameter.
MOV R3, byte        ;Low byte of delay parameter.
LCALL 0E012H        ;Call to delay routine.
```

The bytes passed to R2 and R3 can be immediate data, direct data address, or indirect addresses.

For example, to introduce a pause of one second (one million microseconds or ten thousand times the delay factor of 100 microseconds), the parameter would be 2710H, and the call sequence is:

```
MOV R2, #27H        ;High byte.
MOV R3, #10H        ;Low byte.
LCALL 0E012H        ;Call the delay routine.
```

Interrupt Considerations

Details on interrupt programming for the 8051 microcomputer family are given in the MCS-51 User's Guide; the Preface has a complete reference to that manual. The following considerations apply only to the use of interrupts by the user program on the SDK-51.

1. External interrupt 0 is reserved for system use. This interrupt uses location 0003H as its interrupt vector.

   NOTE

   The code at locations 0003H, 0004H, and 0005H must be the following instruction:

   ```
   LJMP E003H
   ```

   If the user program is RAM, the system will overwrite location 0003H with the required instruction. If the user program is in PROM or ROM, it must contain the required instruction for correct system operation. The user should refrain from executing (with GO or STEP) across this location.

NOTE

The user program should not use or disable external interrupt 0; both the EX0 bit (IE register bit 0) and the EA bit (IE register bit 7) must be 1 at all times to allow system interrupts (breakpoints, UPI operations, guarded access) to operate correctly.

2. All other interrupts may be enabled and used by the user program.
3. In particular, to connect the INTR key on the keyboard to the external interrupt 1 input (Port 3 pin 3), place a shorting plug across jumper pins W12A-W12B. The user program must then enable this interrupt and have code at location 0013H to handle the interrupt (or jump to a service routine higher in memory).
4. The SDK-51 breakpoint logic contains hardware to determine when the first byte of an instruction is being fetched from memory. This information is necessary for program breakpoint operation. Following an interrupt, however, this hardware loses synchronization with the instruction fetches.

NOTE

For proper breakpoint operation after an interrupt in the user program, the user program must resynchronize the instruction cycle counter by reading or writing external data memory (MOVX instruction) early in the interrupt service routine. The instruction need not affect program operation; for example, the following sequence "writes" the accumulator to monitor ROM:

```
MOV DPTR,#0FFFFH
MOVX @DPTR,A
```

This sequence resynchronizes the breakpoint logic, affecting only the DPTR.

NOTE

Since the system uses the interrupt structure to handle breakpoints (including single stepping), the following restrictions on breaking within interrupt routines apply:

1. Breakpoints should not be used in high-priority interrupt routines.
2. After breaking in a low-priority interrupt routine, the Interrupt In Progress flag remains set until cleared by a RETI instruction. For proper operation, resume execution where it left off in the interrupt routine, or clear the flag by executing a RETI instruction before resuming execution at the outer level.

Interrupt Program Example

Example 4-1 contains a short demonstration program for the SDK-51. This program illustrates the use of interrupt 1, and several of the Monitor calls discussed earlier in this chapter. The leftmost column shows the address where each instruction begins. The middle column shows the instruction to be entered in SDK-51 assembly mode. The rightmost column provides a brief explanation of each instruction.

To enter this program, begin assembler mode at the first address with the command:

```
ASM ORG 13
```

Then enter the instructions; end each instruction with the RETURN key. The system displays the new assembly address and prompts you for the next instruction. If you make a mistake that results in an error message, press RETURN to clear the error, then enter the ASM command again; the ORG address is not necessary in this case. If you enter the wrong instruction, press RETURN to enter interrogate mode, then enter ASM ORG address to enter the correct instruction.

When all instructions have been entered, press RETURN twice to begin interrogate mode.

Place a shorting plug on jumper W12.

Then enter the command:

```
GO FROM 2F
```

to begin executing. The program displays the message WAITING.

Press the INTR key on the keyboard. The interrupt routine displays the character I for two seconds, then returns to the main program and the WAITING message. To terminate the program press the ESC key.

Example 4-1. Interrupt Program

Enter the following commands on the SDK-51 to load the sample program.

```
ASM ORG 13                  ; Interrupt service routine
0013     MOV DPTR, #8000H   ; External data memory access is required to synchronize
0016     MOVX @DPTR, A      ; breakpoint hardware.
0017     ANL 0A8, #0FBH     ; Mask out Timer 0 interrupts
001A     LCALL 0E00FH       ; Call monitor routine NEWLINE to clear the display.
001D     MOV R2, #49H       ; 49H is the ASCII value for the letter I. The ASCII value is
                            ; loaded into R2 for the CO routine.
001F     LCALL 0E006H       ; Call monitor routine CO to display the character.
0022     MOV R2, #4EH       ; Load R2 and R3 with values to determine the time delay of
0024     MOV R3, #20H       ; the routine TIME. Value 4320H gives routine TIME. Value
                            ; 4E20H gives a delay of 2 seconds.
0026     LCALL 0E012H       ; Call monitor routine TIME.
0029     SETB 0             ; The main program uses bit address 00H as a flag to
                            ; indicate that an EXT1 interrupt has occurred.
002B     ORL 0A8H, #84H     ; Enable the interrupt that was masked out in line 0017.
002E     RETI               ; End of interrupt routine. The program returns to the point
                            ; in the main program where it was interrupted.
; Main program, initialization steps.
002F     ORL 0A8H, #84H     ; Enable interrupts. 0A8H is the address of the interrupt
                            ; enable (IE) register.
0032     SETB 00H           ; Initialize bit in data memory used as interrupt flag.
0034     CLR 0D3H           ; Select register bank 0 by clearing bits 3 and 4 of the
0036     CLR 0D4H           ; program status word (PSW).
; Main loop.
0038     JNB 00H, 0038H     ; Loops here until bit 00H is set to 1. This flag will be set
                            ; to 1 after initialization (first time through) and after
                            ; returning from the interrupt service routine.
003B     LCALL 0E00FH       ; Call NEWLINE to clear display.
003E     MOV R2, #00H       ; R2 and R3 get the high and low bytes of the address of
0040     MOV R3, #90H       ; the ; message string.
0042     LCALL 0E01EH       ; Call PRINT STRING to display the message.
0045     CLR 00H            ; Clear the flag now that the display is updated.
0047     SJMP 0038H         ; Jump back to the start of the loop.
```

(Continued on next page)

Example 4-1. Interrupt Program (Continued)

Press RETURN twice to exit the assembler mode. Enter the message string with the following CBYTE commands:

```
CBYT 90 = 07,57,41,49,54
CBYT 95 = 49,4E,57
```

To start the program, enter:

```
GO FROM 2F
```

8031 On-Chip UART

Details on using the 8031 on-chip serial I/O port are given in the MCS-51 User's Manual described in the Preface.

To interface directly to the serial I/O pins on the 8031, connect to jumpers P30 (Receive Data) and P31 (Transmit Data). Other pins of P3 have functions as follows:

- P32 is reserved for system use (interrupt 0)
- P33 (interrupt 1) is available to the user.
- P34 is available to the user.
- P35 is available to the user.
- P36 is reserved for system use (WR/)
- P37 is reserved for system use (RD/)

P33, P34, and P35 can have serial I/O functions as described in the *MCS-51*™ *User's Guide.*

On-Board Jumpers

To use the on-chip UART with the serial I/O interface on the SDK-51, the serial I/O jumpers on the board must be configured as shown in Tables 4-2, 4-3, and 4-4, depending on the transmission protocol.

Table 4-2. Serial I/O Jumpers for RS-232, Slave Mode

Jumper	Connection
W1	W1A - W1B
W2	open
W3	open
W4	open
W5	W5A - W5B
W6	W6A - W6B
W7	W7B - W8B
W8	W8B - W7B
W9	open
W10	W10B - W11B*
W11	W11B - W10B*

*NOTE: Alternately, connecting W10A - W10B and W11A - W11B allows normal operation of RTS/ and CTS/ when external handshake is required.

Table 4-3. Serial I/O Jumpers for RS-232, Master Mode

Jumper	Connection
W1	W1A - W1B
W2	open
W3	open
W4	open
W5	W5A - W6A and W5B - W6B
W6	W6A - W5A and W6B - W5B
W7	W7A - W8A
W8	W8A - W7A
W9	open
W10	W10B - W11B*
W11	W11B - W10B*

*NOTE: Alternatively, connecting W10A - W10B and W11A - W11B allows normal operation of RTS/ and CTS/ if external handshake is required.

Table 4-4. Serial I/O Jumpers for Current Loop

Jumper	Connection
W1	W1A - W1B
W2	open
W3	W3A - W3B
W4	W4A - W4B
W5	open
W6	open
W7	open
W8	W8B - W9B
W9	W9B - W8B
W10	open
W11	open

Setting Baud Rate

The serial I/O port on the 8031 uses the auto-reload mode of timer 1 to generate the serial port baud rate. To set the baud rate, load the reload value in TH1 (hex address 8DH), set the TMOD register (hex address 89H) for auto-reload mode on Timer 1, and start the timer by setting bit TR1 (hex address 8EH).

The reload values for five baud rates are shown in Table 4-5; the values assume the 12 MHz crystal as supplied in the kit.

Table 4-5. Baud Rate Auto-Reload Values

Baud Rate	Hex Value for TH1
150 baud	30H
300 baud	98H
600 baud	0CCH
1200 baud	0E6H
2400 baud	0F3H

For example, to set 2400 baud, the code is:

```
MOV 8DH, #0F3H     ; Set reload value in TH1.
MOV 89H, #20H      ; Select auto-reload mode, Timer 1.
SETB 8EH           ; Start Timer 1
```

The general formula for computing the reload value is:

```
TH1 Initial Value = - ((Osc. Clock Rate in Hz)/(384 * Baud Rate))
```

The unary minus means two's complementation, the / means integer division, and the * means multiplication. The value 384 is a decimal constant.

Hexadecimal Addresses

The SDK-51 assembler mode does not accept the symbolic register names used by ASM-51. For reference, Table 4-6 shows the byte and bit addresses of selected registers involved in serial I/O operations.

Table 4-6. Serial I/O Register Addresses

Register Name	Byte Address	Bit Address
TCON	88H	
TMOD	89H	
TH1	8DH	
SCON	98H	
SBUF	99H	
P3	B0H	
TR1		8EH
P30 (RXD)		B0H
P31 (TXD)		B1H

Parallel I/O Interfacing

The 8031 has 32 parallel I/O lines. In the SDK-51, sixteen bits (ports P0 and P2) are used as address/data lines and the use of P3 is restricted by the system use of P32 (INT0), P36 (WR/), and P37 (RD/) as control lines. Lines P10 through P17, however, are available for parallel I/O operations. In addition, the SDK-51 design includes the 8155 (U64), which provides an additional 22 parallel I/O lines arranged into two 8-bit ports, A and B, and one 6-bit port, PC. Figure 4-2 shows the location of the pinouts for these two parallel I/O interfaces on connectors J4 and J5.

Using Port P1

8031 source code is used to program P10 through P17 for read or write (drive) functions. The lines can be addressed as a byte (hexadecimal address 90H), or as individual lines (bit addresses 90H through 97H). To configure a line for input, write a 1 to the line before reading it (0FFH for all lines at once).

For example, to read port 1 as a byte:

```
ORL 90H, #0FFH     ; Configure P1 for input.
MOV A, 90H         ; Read P1 into accumulator.
```

When writing to P1, any instruction that has P1 as the destination may be used; the operation should preserve the input bits by writing ones to those bits. The user must execute or single step to have the value written appear on the port pins.

Using the 8155-2

The 8155-2 parallel I/O interface is addressed as part of the SDK-51 system memory: addresses B000H through BFFFH. Addresses below B800H access the 8155 on-chip RAM; this space is reserved for system use.

The 8155-2 has three parallel I/O ports A, B, and C. To access these ports, a command must be sent to the 8155 Command register (address B800H) to configure the ports as input or output. Table 4-7 shows the values to be loaded into the accumulator for each possible combination of inputs and outputs.

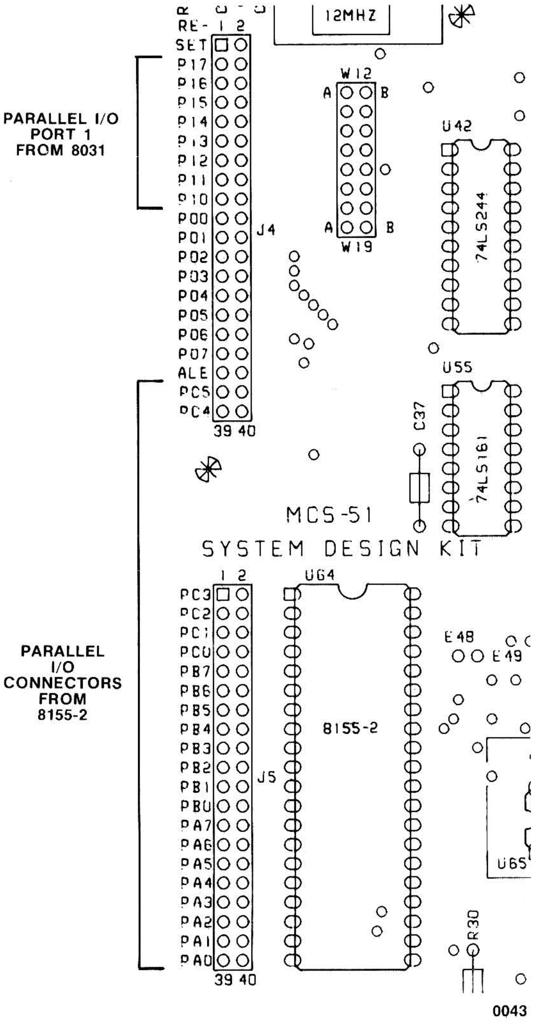


Figure 4-2. Parallel I/O Interface Connectors

Table 4-7. 8155 Command Register Values

Accumulator	Port A	Port B	Port C
00H	input	input	input
01H	output	input	input
02H	input	output	input
03H	output	output	input
0CH	input	input	output
0DH	output	input	output
0EH	input	output	output
0FH	output	output	output

To configure the 8155, the general code sequence is:

```
MOV A, #nnH         ; nnH is the value from Table 4-7.
MOV DPTR, #0B800H   ; B800H is the address of the 8155 command
                    ; register.
MOVX @DPTR, A       ; Write to the command register.
```

From the SDK-51 console, the command register can be configured with an XBYTE command:

```
XBYTE 0B800H = nnH
```

For example to set port A for input, and ports B and C for output:

```
MOV A, #62H MOV DPTR, #0B800H MOVX, @DPTR, A
```

or, from the console:

```
XBYTE 0B800 = 62
```

Ports A, B, and C have addresses shown in Table 4-8.

Table 4-8. Parallel I/O Port Addresses

Port	Address
Command	0B800H
Port A	0B801H
Port B	0B802H
Port C	0B803H
Other addresses on the 8155-2 are reserved for system use.	

To read a port, the code is:

```
MOV DPTR, #0B80nH  ; Select port address from Table 4-8.
MOVX A, @DPTR      ; Read selected port into accumulator.
```

To read the value from the console and display it on the SDK-51, the command is:

XBYTE 0B80*n* For example, to read port A:

```
MOV DPTR, #0B801H; ; Address of port A
MOVX A, @DPTR      ; Read byte into accumulator.
```

The console equivalent is:

```
XBYTE 0B801
```

To write to a parallel I/O port, the code sequence is:

```
MOV A, byte         ; Load the value to be output; can be immediate, data,
                    ; direct data address, or indirect address.
MOV DPTR, #B80≙H    ; Port address from Table 4-8.
MOVX @DPTR, A       ; Write selected port.
```

The console equivalent is:

```
XBYTE 0B80n = byte
```

For example, to transmit the value 41H to port B:

```
MOV A, #41H
MOV DPTR, #0B802H
MOVX @DPTR, A
```

The console equivalent is:

```
XBYTE 0B802 = 41
```

For more details on the 8155 operations, refer to the *Intel Component Data Catalog*.

Scanning A 3 x 4 Matrix Keypad

Printed wiring has been provided on the SDK-51 board to allow the seven 8051 I/O lines, P10 through P16, to be used to scan a 3 x 4 matrix keypad. The keypad is part of the SDK-51 keyboard. Install shorting plugs across jumpers W13 through W19 to connect P10 through P16 to J2, the keyboard connector. Figure 4-3 shows the hardware connections for the auxiliary keypad.

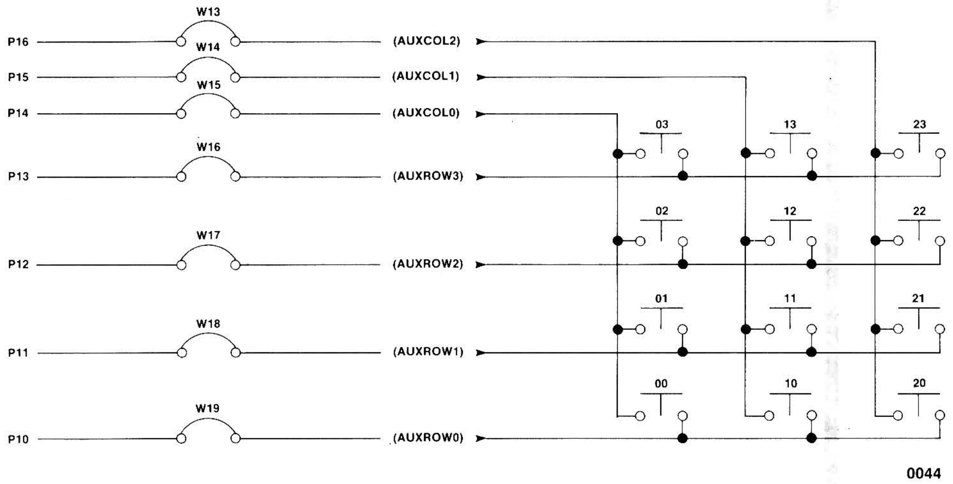


Figure 4-3. Auxiliary Keypad Hardware Connections

Examples 4-2 and 4-3 show a program that scans the 3 x 4 matrix keypad. Example 4-2 shows the lines as you might enter them using the SDK-51 ASM command; this figure also provides comments to indicate the program logic. (Note that all numeric values in Example 4-2 are hexadecimal.) Example 4-3 shows how to enter the program in hexadecimal with the CBYTE commands, using the continuation feature.

The scheme used to scan the keypad is as follows. Columns 0 through 2 are sequentially pulled low by writing a zero to the appropriate bit position of port 1. Bit 4 selects column 0; bit 5 selects column 1; bit 6 selects column 2. A table indexed by R0 contains byte values to be written to port 1 with the column select bits in place. During each of the time intervals when a column is enabled (low), rows 0 through 3 are checked by reading the corresponding bits of port 1 to see if any one (or more) has a low value, indicating a key has been pressed. The combination of column enabled and row sensed low identifies the key pressed. The program displays the number of any auxiliary key pressed.

The procedure is interrupt driven. Each time an interrupt is generated, the next column is enabled. The interrupts are generated by internal Timer 0 on the 8031 at intervals of about 4 milliseconds. Thus, the entire keypad is scanned approximately every 12 milliseconds. The delay between scans serves two purposes. First, a key closure is taken as valid only if the same key is detected on two successive scans, so the delay acts as a debounce mechanism. Second, since the keypad scans are infrequent (relative to the instruction cycle time of a few microseconds at most), the interrupt scheme frees the processor to carry out more important tasks most of the time with occasional pauses to service the keypad.

The program uses the following registers and memory locations. To incorporate this program into a larger program, these locations must be saved. One way to save them is to modify the interrupt service routine to PUSH the values at the start of the routine and POP them at the end.

Register bank 0:
- R0 is a pointer to the table (data locations 30H, 31H, and 32H) containing the settings for port 1 that enable columns 0, 1, and 2 in sequence and condition the row pins for input.
- R1 is the row counter.
- R2 is used to pass parameters to monitor system calls.

On-chip data memory locations 20H through 23H:
- DBYTE 20H, bit addresses 00H and 01H
  - Bit 00H is a flag indicating a key closure on the last scan.
  - Bit 01H is a flag indicating a key closure on the current scan.
- DBYTE 21H contains the key matrix value (00, 01, 02, 03, 10, 11, 12, 13, 20, 21, 22, or 23) of the key detected on the previous scan.
- DBYTE 22H contains the key value from the current scan.
- DBYTE 23H contains the key value of the last character displayed.

On-chip data memory locations 30H, 31H, and 32H:
- DBYTE 30H contains the value 0EFH; the zero in bit 4 selects column 0 when the byte is written to port 1.
- DBYTE 31H contains 0DFH; the zero in bit 5 selects column 1 when the byte is written to port 1.
- DBYTE 32H contains 0BFH; the zero in bit 6 selects column 2 when the byte is written to port 1.

Example 4-2. Keypad Scan Program, Source Code

NOTE: All numeric values are hexadecimal.

```
Addr    Instruction          Comments

; Begin assembly mode with the command ASM ORG 0.

0000    LJMP 0079            ; Jump to start of main program.

; Press RETURN to end assembly mode so you can change the assembly pointer with the
; command ASM ORG 0B. This portion is the Timer 0 interrupt service routine.

000B    CLR 0A9              ; Mask Timer 0 interrupt.
000D    CLR 8C               ; Stop timer.
000F    MOV DPTR, #8000      ; Special SDK-51 requirement to
0012    MOVX @DPTR, A        ; synchronize break hardware.
0013    MOV R1, #00          ; Initialize row counter.
0015    MOV A, 90            ; Read port 1.
0017    JNB 0E0, 0025        ; Test for 0 in LSB (key closure in current row). Location
                             ; 0025 begins the routine to process the key closure.
001A    INC R1               ; Increment row counter to point to next row.
001B    RR A                 ; Rotate so next bit can be checked.
001C    CJNE R1, #04, 0017   ; Have all rows been checked?
001F    INC R0               ; Point to next column.
0020    CJNE R0, #33, 006F   ; Have all columns been checked?
0023    SJMP 0046            ; If not, prepare to return from the current interrupt
                             ; (006FH) if all columns checked, jump to code which is
                             ; executed after a complete pass through the columns
                             ; (0046H).

; This portion of code is executed when a key closure is detected. It saves the value of the closed
; key for future use, and checks for the error condition of multiple keys pressed at the same time.

0025    SETB 01              ; Flag for key closure on current scan.
```

(Continued on next page)

Example 4-2. Keypad Scan Program, Source Code (Continued)

NOTE: All numeric values are hexadecimal.

```
Addr    Instruction          Comments

; This code saves the value of the key closed.

0027    PUSH 0E0             ; Save accumulator.
0029    MOV A, R0            ; Load column pointer.
002A    ANL A, #0F           ; Mask off most significant nibble, then swap the column
002C    SWAP A               ; value into the high nibble.
002D    ORL A, R1            ; Load row pointer in low nibble.
002E    MOV 22, A            ; Save the key value (column, row) in data byte 22H.
0030    POP 0E0              ; Restore accumulator.

; Begin error checks.

0032    JNB 00, 001A         ; If no key closure on last pass, no error is possible, so
                             ; continue with row scans.

; If there was a previous key closure, compare the key value of that closure with the current
; key value. If they are different, multiple key closures have occurred.

0035    PUSH 0E0             ; Save accumulator.
0037    MOV A, 21            ; Load previous key value.
0039    CJNE A, 22, 0040     ; Compare to current key value;
                             ; If they are different, jump to error section (0040H).
003C    POP 0E0              ; Otherwise, pop accumulator and
003E    SJMP 001A            ; continue with row scans.

; This error routine corrects for a multiple key closure by clearing the previous key closure
; flag, in effect "forgetting" about that closure.

0040    CLR 00               ; Clear the previous key flag.
0042    POP 0E0              ; Pop accumulator.
0044    SJMP 001A            ; Continue with row scans.

; This section of code is reached when one complete pass through the columns has been made.

0046    JNB 01, 006D         ; If no closure detected, jump to clean-up steps near end of
                             ; interrupt routine.
0049    JB 00, 0053          ; Closure detected on current pass; if also closure on previous
                             ; pass, jump to compare section. Otherwise,
004C    MOV 21, 22           ; Save the current key value,
004F    SETB 00              ; set the previous closure flag,
0051    SJMP 0068            ; and jump to the clean-up code near end of routine.

; This code is reached when key closures are detected on two successive passes.

0053    MOV A, 22            ; Load value of current key.
0055    CJNE A, 21, 004C     ; If the current value is not the same as the previous value
                             ; save the current value as the previous value (by jumping
                             ; back to the save routine at location 004C).
0058    CJNE A, 23, 005D     ; If the current value is not the same as the value now being
                             ; displayed, jump to the display routine (005D).
005B    SJMP 0068            ; Otherwise jump to the clean-up code near end of routine.

; This section displays the characters representing the key value.

005D    PUSH 0E0             ; Save accumulator; note that the accumulator contains the key
                             ; value to be displayed.
005F    LCALL 0E00F          ; Call monitor routine NEWLINE to clear the display.
0062    POP 0E0              ; Restore accumulator (NEWLINE uses the accumulator).
0064    MOV R2, A            ; Load R2 with the value to be displayed in hexadecimal.
0065    LCALL 0E015          ; Call monitor routine LSTBYT to display the hexadecimal
                             ; (ASCII) characters for the two nibbles.
```

(Continued on next page)

Example 4-2. Keypad Scan Program, Source Code (Continued)

NOTE: All numeric values are hexadecimal.

```
Addr    Instruction        Comments
; The remainder of the service routine performs clean-up prior to returning.
0068    CLR 01             ; Clear current closure flag.
006A    MOV 22, #0FF       ; Blanks current key value to all ones.
006D    MOV R0, #30        ; Begin scan again at column 0 via table pointer R0.
006F    MOV 90, @R0        ; Writes table value out to port 1, enabling column 0 and
                           ; conditioning row pins for input during interrupt.
0071    MOV 8C, #0F0       ; Reload TH0 with initial timer value (about 4 msec).
0074    SETB 8C            ; Start timer 0.
0076    SETB 0A9           ; Enable timer 0 interrupt.
0078    RETI               ; Return to main loop.
; Main routine, initialization section. This code is executed only once, at the beginning of
; program execution.
0079    ANL 0B8, #0E1      ; Set all interrupts for priority 0, except EXTI0 (reserved for
                           ; SDK).
007C    ANL 0A8, #0E1      ; Mask all interrupts except EXTI0.
007F    ANL 88, #03        ; Turn internal timer off.
0082    MOV 89, #01        ; Set timer mode for 16-bit internal timer.
0085    CLR 0D3            ; Select register bank 0 with
0087    CLR 0D4            ; these two instructions.
; The next five instructions set up the table of port 1 settings that enable columns 0, 1, and 2 in
; sequence. The column enable bits are P14 (column 0), P15 (column 1, and P16 (column 2);
; writing a 0 to the bit enables the corresponding column. The low nibble of the port 1 value
; (always 0FFH in the table) conditions pins P10 through P13 (corresponding to the four rows)
; for input during the interrupt routine.
0089    MOV R0, #30        ; Initialize R0, the index into the table of port 1 settings.
008B    MOV 30, #0EF       ; Enables column 0 when written to port 1.
008E    MOV 31, #0DF       ; Enables column 1 when written to port 1.
0091    MOV 32, #0BF       ; Enables column 2 when written to port 1.
0094    MOV 90, @R0        ; Writes table value to port 1, (enabling column 0 this time).
0096    MOV 22, #0FF       ; Blank current key value.
0099    MOV 21, #0FF       ; Blank previous key value.
009C    CLR 01             ; Clear current key closed flag.
009E    CLR 00             ; Clear previous key closed flag.
00A0    MOV 23, #0FF       ; Blank display value.
00A3    MOV 8C, #0F0       ; Set timer 0 for 4 msec
00A6    MOV 8A, #00        ; initially; the interrupt routine restarts the timer
                           ; before returning each time.
00A9    SETB 8C            ; Start timer 0.
00AB    SETB 0A9           ; Enable timer 0 interrupt.
; Now the program enters the main loop, waiting for an interrupt. In a normal program,
; the main loop would perform one or more processing tasks.
00AD    SJMP 00AD          ; Loop forever. Interrupts return here. To halt
                           ; execution, press ESC.
```

Example 4-3. Keypad Scan Program, Hexadecimal

This example uses the continuation feature for memory change commands. Enter the CBYTE command on the first two lines only. After the first line, end each line with a comma before the RETURN; the comma causes the system to continue the command to the next line, display the keyword CBYT, the next available address, the equals sign, and the hyphen prompt. The display for each line after the second line is shown in parentheses. After the = sign on each line, enter the hexadecimal values shown.

Each command line loads one instruction into code memory. The result is the same as the assembler mode commands, but the number of keystrokes is reduced.

```
CBYTE 0 = 02, 00, 79
CBYT 000B = 0C2, 0A9,
```

(Continued on next page)

Example 4-3. Keypad Scan Program, Hexadecimal (Continued)

```
(CBYT 000D =) 0C2, 8C,
(CBYT 000F =) 90, 80, 00,
(CBYT 0012 =) 0F0,
(CBYT 0013 =) 79, 00,
(CBYT 0015 =) 0E5, 90,
(CBYT 0017 =) 30, 0E0, 0B,
(CBYT 001A =) 09,
(CBYT 001B =) 03,
(CBYT 001C =) 0B9, 04, 0F8,
(CBYT 001F =) 08,
(CBYT 0020 =) 0B8, 33, 4C,
(CBYT 0023 =) 80, 21,
(CBYT 0025 =) 0D2, 01,
(CBYT 0027 =) 0C0, 0E0,
(CBYT 0029 =) 0E8,
(CBYT 002A =) 54, 0F,
(CBYT 002C =) 0C4,
(CBYT 002D =) 49,
(CBYT 002E =) 0F5, 22,
(CBYT 0030 =) 0D0, 0E0,
(CBYT 0032 =) 30, 00, 0E5,
(CBYT 0035 =) 0C0, 0E0,
(CBYT 0037 =) 0E5, 21,
(CBYT 0039 =) 0B5, 22, 04,
(CBYT 003C =) 0D0, 0E0,
(CBYT 003E =) 80, 0DA,
(CBYT 0040 =) 0C2, 00,
(CBYT 0042 =) 0D0, 0E0,
(CBYT 0044 =) 80, 0D4,
(CBYT 0046 =) 30, 01, 24,
(CBYT 0049 =) 20, 00, 07,
(CBYT 004C = ) 85, 22, 21,
(CBYT 004F =) 0D2, 00,
(CBYT 0051 =) 80, 15,
(CBYT 0053 =) 0E5, 22,
(CBYT 0055 =) 0B5, 21, 0F4,
(CBYT 0058 =) 0B5, 23, 02,
(CBYT 005B =) 80, 0B,
(CBYT 005D =) 0C0, 0E0,
(CBYT 005F =) 12, 0E0, 0F,
(CBYT 0062 =) 0D0, 0E0,
(CBYT 0064 =) 0FA,
```

(Continued on next page)

Example 4-3. Keypad Scan Program, Hexadecimal (Continued)

```
(CBYT 0065 =) 12, 0E0, 15,
(CBYT 0068 =) 0C2, 01,
(CBYT 006A =) 75, 22, 0FF,
(CBYT 006D =) 78, 30,
(CBYT 006F =) 86, 90,
(CBYT 0071 =) 75, 8C, 0F0,
(CBYT 0074 =) 0D2, 8C,
(CBYT 0076 =) 0D2, 0A9,
(CBYT 0078 =) 32,
(CBYT 0079 =) 53, 0B8, 0E1,
(CBYT 007C =) 53, 0A8, 0E1,
(CBYT 007F =) 53, 88, 03,
(CBYT 0082 =) 75, 89, 01,
(CBYT 0085 =) 0C2, 0D3,
(CBYT 0087 =) 0C2, 0D4,
(CBYT 0089 =) 78, 30,
(CBYT 008B =) 75, 30, 0EF,
(CBYT 008E =) 75, 31, 0DF,
(CBYT 0091 =) 75, 32, 0BF,
(CBYT 0094 =) 86, 90,
(CBYT 0096 =) 75, 22, 0FF,
(CBYT 0099 =) 75, 21, 0FF,
(CBYT 009C =) 0C2, 01,
(CBYT 009E =) 0C2, 00,
(CBYT 00A0 =) 75, 23, 0FF,
(CBYT 00A3 =) 75, 8C, 0F0,
(CBYT 00A6 =) 75, 8A, 00,
(CBYT 00A9 =) 0D2, 8C,
(CBYT 00AB =) 0D2, 0A9,
(CBYT 00AD =) 80, 0FE
```

Prototype Area Techniques

The prototype area of the SDK-51 is located on the left side of the board. It contains 1875 solder terminal holes, spaced at 0.01 inch for use with standard dual in-line package IC sockets or discrete components. Adjacent to the prototype area, connectors J3, J4 and J5 bring out the address, data, and control pins of the 8031 microcontroller and the 8155-2 parallel I/O interface. The pins of J3, J4, and J5 nearest the prototype area are the signals; the pins nearest the component area are grounded. Locations J9 (GND) and J10 (+5 VDC) bring out the power lines; the pads around these locations are designed to receive screw-on binding posts.

In order to make this area more versatile for evaluating user designed circuits, it is suggested that solder-tail or wire-wrap posts be installed, rather than making solder connections directly to the holes in the board. Wire connections can thus be removed easily without damaging the holes in the board. For installation of IC devices, sockets with wire-wrap posts or solder end terminals are again suggested, allowing easy modification of a circuit. When using the solder-tail posts, it may be desirable to install them on the back side of the board, to allow all interconnections to be kept on the bottom side of the board.

Memory Expansion

One use of the prototype area is to expand the user configurable memory space. As discussed in Chapter 3, printed wiring on the SDk-51 allows up to 24K bytes of RAM or ROM to be installed. The SDK-51 decoding logic, however, allows the entire 32K bytes of SDK-51 user configurable memory to be addressed. If the additional 8K bytes of memory is desired, it can be installed off-board or in the wire wrap area of the board. Note that besides the RAM an address decoder may be required, depending on the type of RAM device used. On the SDK-51, the 74LS138 circuits at U26 and U51 serve as decoders for memory 0 and memory 1 respectively. It is suggested that sockets be installed for these devices.

Data lines DB0 through DB7 (J4 pins P00 through P07) must also be jumpered to the expansion memory devices.

The additional memory can be assigned to any of the four address ranges of user configurable memory. Address range 6000H through 7FFF would be typical; for this range, attach a jumper wire from the A pin of W32, W33, W34, or W35 to the Chip Select input on each memory device. Refer to Memory Configuration in Chapter 3 for more details.

U67-9

For RAM devices, the WR/ signal is output at pin P36 on J3. If write-protection is desired (using the TOP circuit), the RAMWR/ signal (indicating a permitted write operation) is available at several points; pin 3 of U66 is the origin, and pin 10 of each 2114 RAM device is the destination.

APPENDIX A
TELETYPEWRITER MODIFICATIONS

Introduction

This appendix provides information required to modify a Model ASR-33 Teletypewriter for use with the SDK-51.

Internal Modifications

WARNING

> Hazardous voltages are exposed when the top cover of the teletypewriter is removed. To prevent accidental shock, disconnect the teleprinter power cord before proceeding beyond this point.

Remove the top cover and modify the teletypewriter as follows:

a. Remove the blue lead from 750-ohm tap on current source register, reconnect this lead to 1450-ohm tap. (Refer to figures A-1 and A-2).

b. On terminal block, change two wires as follows to create an internal full-duplex loop (refer to figures A-1 and A-3):

   1. Remove brown/yellow lead from terminal 3; reconnect this lead to terminal 5.
   2. Remove white/blue lead from terminal 4; reconnect this lead to terminal 5.

c. On terminal block, remove violet lead from terminal 8; reconnect this lead to terminal 9. This changes the receiver current level from 60 mA to 20 mA.

A relay circuit card must be fabricated and connected to the paper tape reader driver circuit. The relay circuit card to be fabricated requires a delay, a diode, a thyractor, a small 'vector' board for mounting the components, and suitable hardware for mounting the assembled relay card.

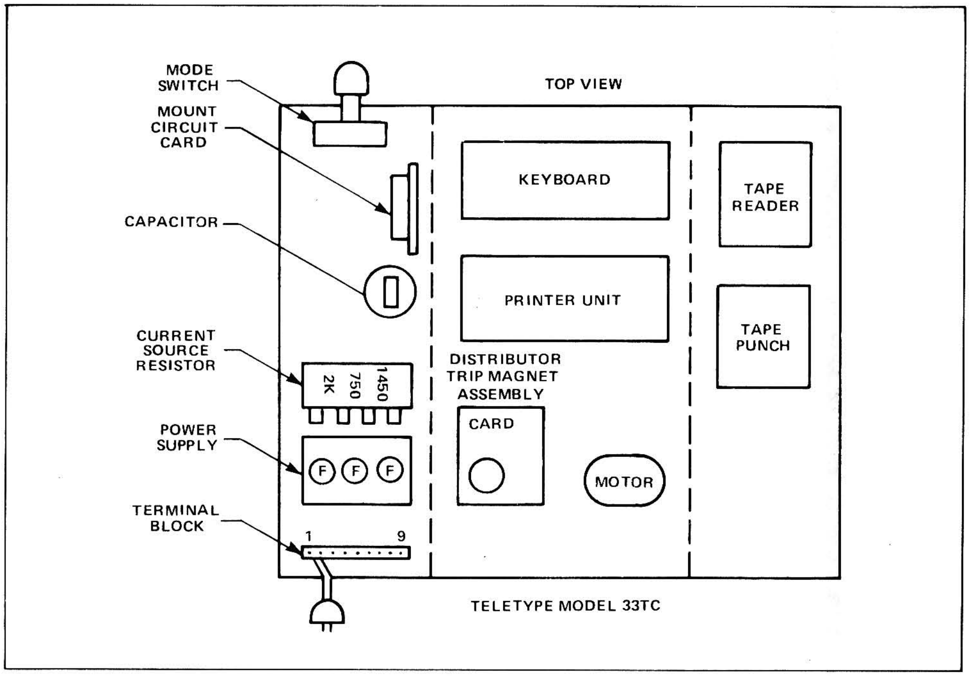


Figure A-1. Teletype Component Layout

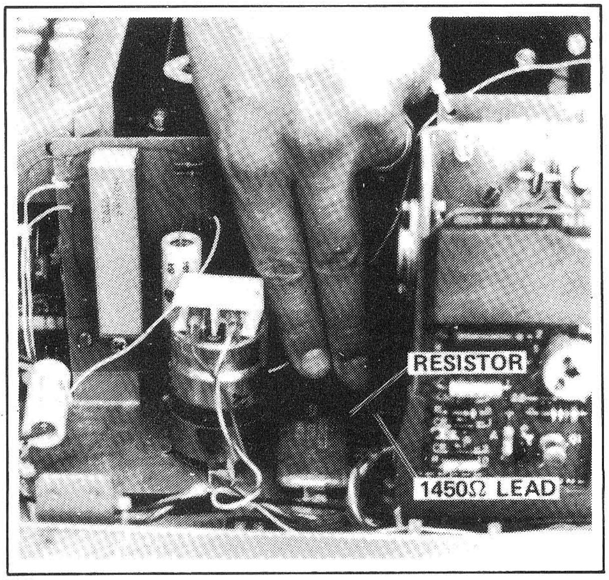


Figure A-2. Current Source Resistor

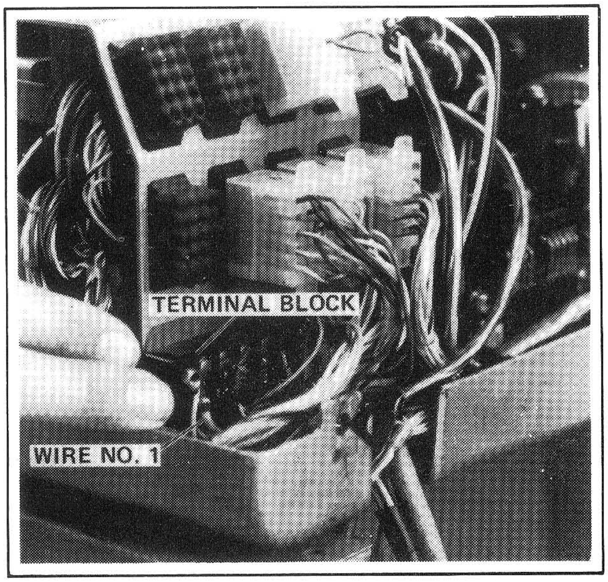


Figure A-3. Terminal Block

A circuit diagram of the relay circuit card is included in figure A-4; this diagram also includes the part numbers of the relay, diode, and thyractor. (Note that a 470-ohm resistor and a 0.1 μF capacitor may be substituted for the thyractor.) After the relay circuit card has been assembled, mount it in position as shown in figure A-5. Secure the card to the base plate using two self-tapping screws. Connect the relay circuit to the distributor trip magnet and mode switch as follows:

a. Refer to figure A-4 and connect a wire (Wire 'A') from relay circuit card to terminal L2 on mode switch. (See figure A-6.)

b. Disconnect brown wire shown in figure A-7 from plastic connector. Connect this brown wire to terminal 12 on mode switch. (Brown wire will have to be extended.)

c. Refer to figure A-4 and connect a wire (Wire 'B') from relay circuit board to terminal L1 on mode switch.

External Connections

Connect a two-wire receive loop, a two-wire send loop, and a two-wire tape reader control loop to the external device as shown in figure A-4. The external connector pin numbers shown in figure A-4 are for interface with an RS232C type, 25-pin connector.

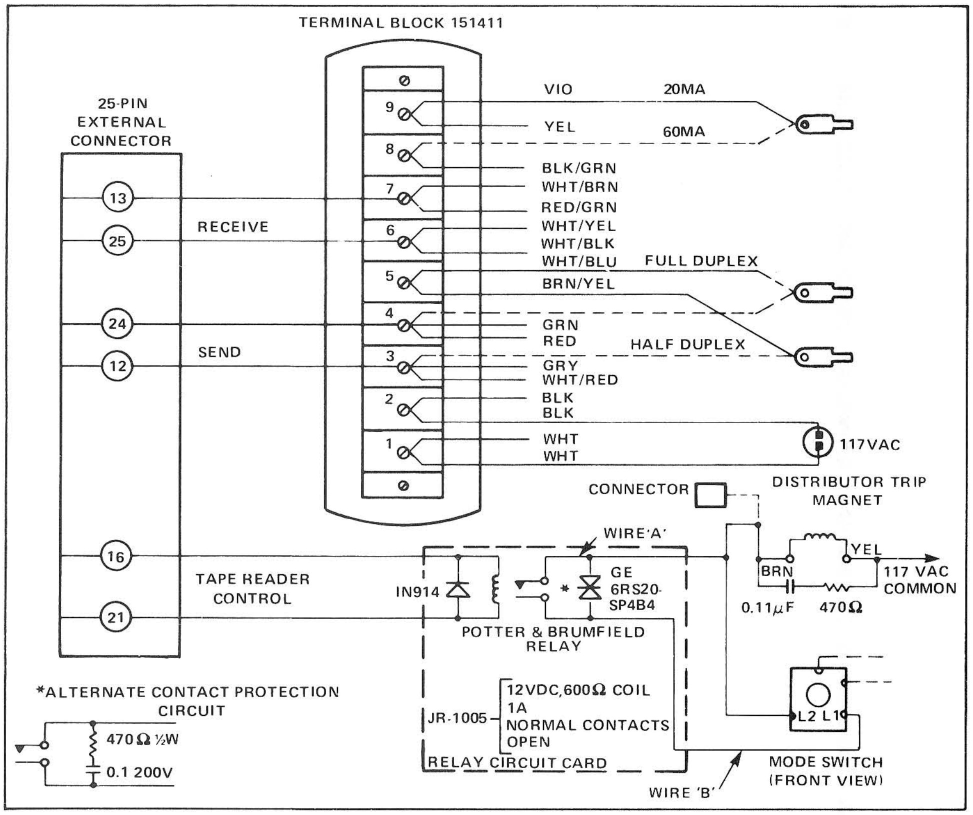


Figure A-4. Teletypewriter Modifications

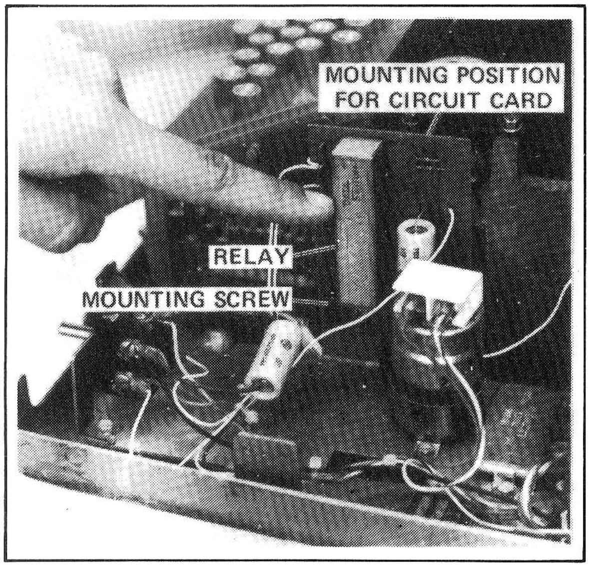


Figure A-5. Relay Circuit

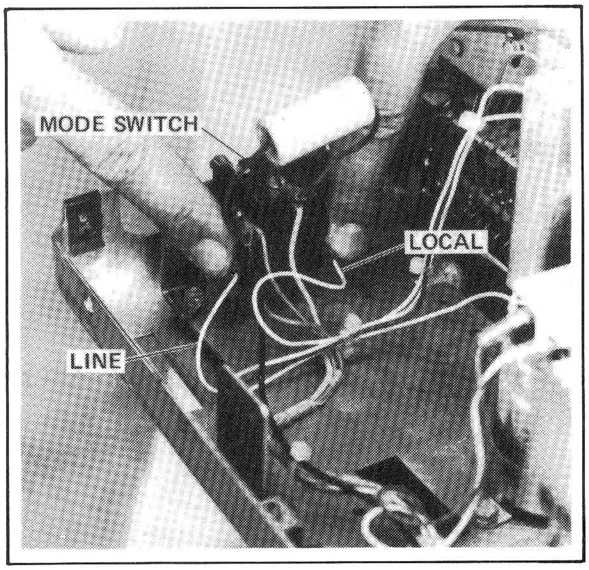


Figure A-6. Mode Switch

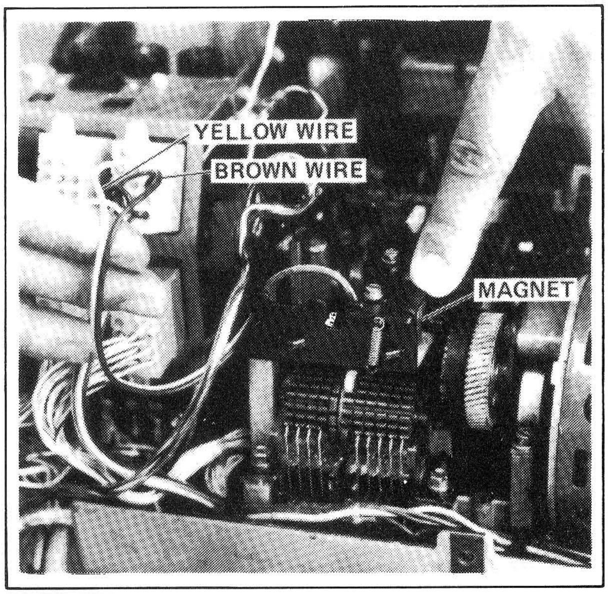


Figure A-7. Distributor Trip Magnet

APPENDIX B
ERROR MESSAGES

This appendix describes the error messages produced by the SDK-51 monitor, with suggestions for corrective action.

ERR=00 PROM CKSUM

Monitor firmware checksum error. Check that the monitor PROMs are correctly socketed. The PROM labeled E000H goes in socket U59, and the PROM labeled F000H goes in socket U60. Check for bent pins on the socket. If error persists, contact Intel field service (see Service Assistance at front of manual).

ERR=01 INVALID WORD

One or more entries in the command cannot be identified as keywords or abbreviations. Enter the command again.

ERR=02 INVALID COMMAND

The first entry in the command cannot be identified as a valid command keyword. Enter the command again.

ERR=03 NUMBER REQ

The command required a numeric entry at some point, but some other entry occurred instead. Note that hexadecimal numbers must begin with numerals; 5F is acceptable, but F5 must be entered with a leading zero: 0F5. Enter the command again.

ERR=04 RETURN REQ

The command contained entries after the last valid entry. The command was complete at some point, but instead of RETURN, some other entry was added. Check command syntax and enter the command again.

ERR=05 EQUAL OR RTRN REQ

The command begins with a keyword referring to some system element that can be displayed or changed. To display the value of the element, follow the keyword with a RETURN; to change the value, follow the keyword with an equals sign (=), then the new value. Any other kind of entry produces this error.

ERR=06 COMMA REQ

A comma is required to separate bytes in a list, or to separate operands in an instruction for the ASM command. Enter the command again. For ASM, press RETURN to clear the error, then enter ASM to begin assembler mode at the address where the error occurred.

ERR=07 PARTITION ADR

Ending address of partition in breakpoint or memory access command is less than beginning address. Enter the partition again, with the lower of the two bounding addresses first.

ERR=08 RESET OR ON REQ

A LIST command contained an entry after the equals sign (=) other than RESET or ON. Enter the command again; to display the LIST setting, enter LIST, then press RETURN.

ERR=09 DECIMAL NUM REQ

In the STEP command, the autostep delay value must be a decimal digit, 0 through 9. Enter the command again.

ERR=0A ILLEGAL BAUD VAL

The command attempted to set the baud rate to a value other than 110, 300, 600, 1200, 2400, 4800, or 9600. Enter the command again using one of the allowable values.

ERR=0B BRK ENABL SYNTAX

In a GO command, the final clause that disables/enables program and/or data breakpoints has one of the following forms: FOREVER, TILL PROGRAM, TILL DATA, or TILL PROGRAM OR DATA. Any other entry produces an error. An incorrect FROM clause after GO or STEP also produces ERR 0B. Enter the command again.

ERR=0C NUM OR RESET REQ

After the command entry BR = (or ABR =), you must enter an address, a partition, or (with BR) the keyword RESET. Enter the command again; to display the breakpoint settings, enter the keyword BR, then press RETURN.

ERR=0D TOP > 7FFFH

The command attempted to set a Top of Program memory address higher than the highest address in user-configurable memory. Addresses 8000H and higher are in system memory and cannot be write-protected by the user. Enter the command again with a lower value.

ERR=0E DISPLAY ONLY

The CAUSE command displays the cause of the last break in execution of the user program. The cause cannot be changed from the console. To display the break cause, enter the keyword CAUSE, then press RETURN.

ERR=0F UNDEFINED OPCODE

While disassembling memory with the DASM command, opcode 0A5H was encountered. This opcode value is undefined. Disassembly is terminated.

ERR=10 ASSEMBLY SYNTAX

In assembler mode, an instruction was entered with incorrect format. Check the instruction syntax. As a result of the error, nothing was assembled (the assembly pointer is unchanged). Press RETURN to clear the error, then enter ASM to begin assembler mode again at the address where the error occurred.

ERR=11 ADR OUT OF RANGE

In assembler mode, the address in an absolute jump or call instruction requires more than 11 bits. AJMP and ACALL addresses must be within a 2K byte page, since they are encoded in the three high-order bits of the opcode byte and the eight bits of the second byte. Check the address or use a LJMP/LCALL instead. As a result of the error, nothing has been assembled (the assembly pointer is unchanged). Press RETURN to clear the error, then enter ASM to begin assembler mode at the address where the error occurred.

ERR=12 ADR OUT OF RANGE

In assembler mode, the address in a short jump yielded a relative offset greater than seven bits plus sign bit. Check the address or use another form of jump that allows larger offsets. As a result of the error, nothing was assembled (the assembly pointer is unchanged). Press RETURN to clear the error, then enter ASM to begin assembler mode where the error occurred.

ERR=13 ASM PC > 0FFFFH

Assembling the current instruction (in assembler mode) would cause the assembly pointer to overflow from 0FFFFH to 0000H. In general, the user is cautioned against attempting to assemble instructions into system memory (addresses above 7FFFH); however, no automatic protection is provided for system memory. As a result of the error, nothing was assembled (the assembly pointer is unchanged). Press RETURN to clear the error, then enter ASM ORG *address* to begin assembly mode at a new address.

ERR=14 FILE RD OR WR

Hexadecimal file (on UPLOAD or DOWNLOAD) or binary file (on LOAD or SAVE) read or write error (including checksum). Try to access the file again.

ERR=15 MEMORY WRITE

Value read back after write with CBYTE fails to match value written. Possibly memory is ROM, or is non-existent (not installed).

ERR=16 EX ACROSS ADR 03

Execution over vector at location 3. Location 3 is reserved for the system interrupts (breakpoints, UPI operations, guarded access). The user may not execute (with GO or STEP) across this location. The stack pointer may have been changed; verify stack pointer value before continuing execution.

ERR=17 NO RAM AT ADR 03

No RAM at location 3. The system overwrites location 03H with the correct jump for the system interrupt. If the memory assigned to location 3 is ROM or PROM, or if no memory is installed at this location, the write fails. The system will operate correctly if the ROM or PROM contains the correct jump instruction (LJMP 0E003H); it will not operate correctly if no memory is installed. Begin execution from another address.

ERR=18 CBYTE TYPE REQ

A block move or copy command can be used with memory type CBYTE (user program memory) only. Refer to User Program Memory in Chapter 2 for details.

ERR=19 CHANGE ONLY

The ABR command (add breakpoints) has the syntax

ABR = *address* [, *address*]...

This command requires the equals sign and list of addresses. To display the breakpoint addresses currently set, enter the command BR.

ERR=1A CBY OR NUM REQD

A command that begins with CBYTE *partition* = must continue with either a number (list of bytes) or an entry of the form CBYTE *address*. Refer to User Program Memory commands in chapter 2 for details.

REQUEST FOR READER'S COMMENTS

Intel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. ______

NAME ______ DATE ______
TITLE ______
COMPANY NAME/DEPARTMENT ______
ADDRESS ______
CITY ______ STATE ______ ZIP CODE ______

Please check here if you require a written reply. □

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Pkwy.
Hillsboro, Oregon 97123

DSO-N Technical Publications

intel®

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

Printed in U.S.A.