ICE-51™ IN-CIRCUIT EMULATOR OPERATING INSTRUCTIONS FOR ISIS-II USERS

Manual Order Number: 9801004-01 Rev. A

A302/281/5K DD

PREFACE

This manual is the operating instructions for the ICE-51™ In-Circuit Emulator for the MCS-51™ family of microcontrollers. Here is a brief guide to the contents of the manual:

Chapter 1 includes a product overview, explanation of command format notation used in the manual, and definitions of terms used in more than one chapter.

Chapter 2 describes utility commands used to enter, set up, and exit the emulator system.

Chapter 3 presents the basic emulation control commands.

Chapter 4 describes an easy subset of the trace control commands. The intention of chapters 3 and 4 is to provide easy-to-use commands that access a significant percent of the emulator's functions, especially for assembler-level debugging.

Chapter 5 describes the various types of memory and how to access each type through the emulator. Included are commands to disassemble code memory into assembler mnemonics and to assemble mnemonic instructions into code memory.

Chapter 6 gives complete details on emulation and trace controls for users desiring maximum precision (e.g., using events with durations less than one complete instruction), including the use of the external sync lines.

Chapter 7 discusses the block commands and macro facility used to automate all or part of the emulation and test session.

Appendix A contains installation and checkout procedures for the emulator hardware.

Appendix B lists the error and warning messages with suggestions for corrective action.

Appendix C gives information on obtaining service and repair assistance.

Appendix D contains specifications for the emulator hardware.

Appendix E documents known product limitations that may affect performance in some applications.

Appendix F contains reference schematics.

This manual assumes the reader is a designer using the MCS-51™ family of microcontrollers in a new product or application. The reader should be familiar with the operation of the microcontroller, and with its assembler, as described in the following manuals:

MCS-51™ Family User's Manual, Manual Order No. 121517.
MCS-51™ Macro Assembler User's Guide, Manual Order No. 9800937.

In addition, many of the examples in this manual are based on the following Application Note:

An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family, AP-69.

The operating instructions are accompanied by the following reference and tutorial publications:

ICE-51™ In-Circuit Emulator Command Dictionary, Manual Order No. 9801005.
Getting Started with the ICE-51™ In-Circuit Emulator, Manual Order No. 121595.

The output file format used by the emulator is specified in the following document:

Hexadecimal Object File Format, Manual Order No. 9800183.

The emulator system uses the ISIS-II software operating system; we assume the user is familiar with the operating system as described in the following publication:

ISIS-II User's Guide, Manual Order No. 9800306.

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1
INTRODUCTION

The ICE-51™ In-Circuit Emulator for the MCS-51™ family of microcontrollers is a hardware/software product that resides in the Intellec® Microcomputer Development System. The emulator boards connect to the Multibus™ chassis in the development system; the software runs under ISIS-II. The user operates the emulator by entering commands at the console of the development system.

This chapter describes the emulator and its uses, gives details on entering commands, explains the notation used in subsequent chapters for command syntax, and defines several basic terms.

Uses for the Emulator

The emulator aids the design effort in several ways: software execution, hardware/software integration and debug, and diagnostic testing. The next sections present the features of the emulator in the context of these uses.

Software Execution and Assembler-Level Debug

The emulation processor, a special version of the Intel 8051 microcontroller, performs real-time and single-step execution of the user's object code. The system provides 8K of RAM for user code. The code is loaded into RAM from diskette, using the output file generated by the ASM51 assembler. Real-time execution can be halted at points of interest (breakpoints) specified by the user.

The system maintains a trace buffer that can collect data on up to 1000 instructions during emulation. Instructions from the buffer are displayed as assembler mnemonics. Trace also includes port values.

Console entry and display can use symbols instead of absolute addresses. The emulator offers system symbols corresponding to the predefined code addresses, register addresses, and bit addresses in the MCS-51 assembly language. Labels and other user-defined symbols are loaded with the user program if the output file contains a symbol table. The user can define additional symbols for debugging without affecting the code.

The user can assemble short programs or program patches directly into program memory from the console, using the ASM51 instruction mnemonics, predefined symbols, and user defined symbols. The user can copy the modified program and symbol table to an output file on diskette.

The emulator hardware includes a Crystal Power Accessory (CPA). When installed, the CPA provides clock and power to the emulation processor for stand-alone operation as a software execution and debug vehicle.

Integration of Prototype Software and Hardware

The emulator system interacts with the user hardware through the user cable. The user cable terminates in a 40-pin plug that fits the MCS-51 socket on the user system. When the user cable is connected to the user system, the emulation processor becomes the user system processor. In this configuration, clock and power for the emulation processor are provided by the user system.

The emulator system can simulate the 4K of on-chip program memory with RAM, and provide an additional 4K of RAM to represent a segment of external memory. (If needed, both 4K segments can be mapped to represent external memory). The user system may also include any amount of external program memory up to 64K. The emulator executes from any of these memories at full speed, without wait states.

The user can interrogate and change processor registers or memory from the console when the system is not emulating. The new values are immediately reflected at the pins of the emulation processor.

The emulator system provides external signals that can be used as inputs or outputs to allow the user to monitor and control emulation and trace with external test equipment.

Diagnostic or Production Testing

In diagnostic or production testing the goal is to exercise the product in as many modes as necessary to prove full functionality. To facilitate testing, the emulator permits command sequences to be entered as blocks.

A block of commands can be executed repetitively or conditionally. Blocks can be nested. Blocks can be named as command macros. A macro block executes only when called, and can be passed different text parameters each time it is called. Macros can contain repeat and conditional blocks and calls to other macros. Macro definitions can be copied to text files on diskette, then included from file into the command sequence when desired in later test sessions.

Macro blocks can serve as a programmable diagnostic facility. With macros, the user in effect can define new emulator commands for special purposes or to increase generality. Thus, the macro commands provide a way to expand the emulator command language itself.

Not only macro definitions, but any sequence of commands can be entered from a file on diskette, as discussed later in this chapter. The command sequence is entered to file using a text editor. Using command files the user can automate all or part of this phase of diagnostic testing.

The emulator can send a copy of the diagnostic session to a text file on diskette, or to a hard-copy device such as a line printer.

Description of the Emulator System

Reference Diagram

Figure 1-1 is a block diagram of the emulator system with its interfaces to the development system, to the user hardware system and other test equipment, and to the user's input and output controls. The shaded portion of the diagram shows the emulator hardware and software. Unshaded portions are development system resources.

Hardware Components

The hardware components of the emulator are as follows:

- Controller board.
- Trace board.

- Dual auxiliary connector kit containing connectors for model 800 and Series II development systems.
- Buffer box and user cable assembly.
- External sync lines assembly.
- Crystal power accessory.

The controller and trace boards communicate to each other through the dual auxiliary connector, and to the development system through the Multibus connector on each board. The controller and trace boards connect by ribbon cable to the buffer box assembly containing the trace buffer and RAM for the user program memory. A short user cable connects the buffer box assembly to the user plug that connects to the user system; the emulation processor is mounted on the user plug.

The crystal power accessory can be mounted on the user plug and connected to the buffer box for stand-alone operation (no connection to user hardware). The external signal cable assembly contains emulation (SY0), and trace (SY1) signals. The cable assembly uses the same buffer box connector as the crystal power accessory.

Details on installing the hardware components are given in appendix A.

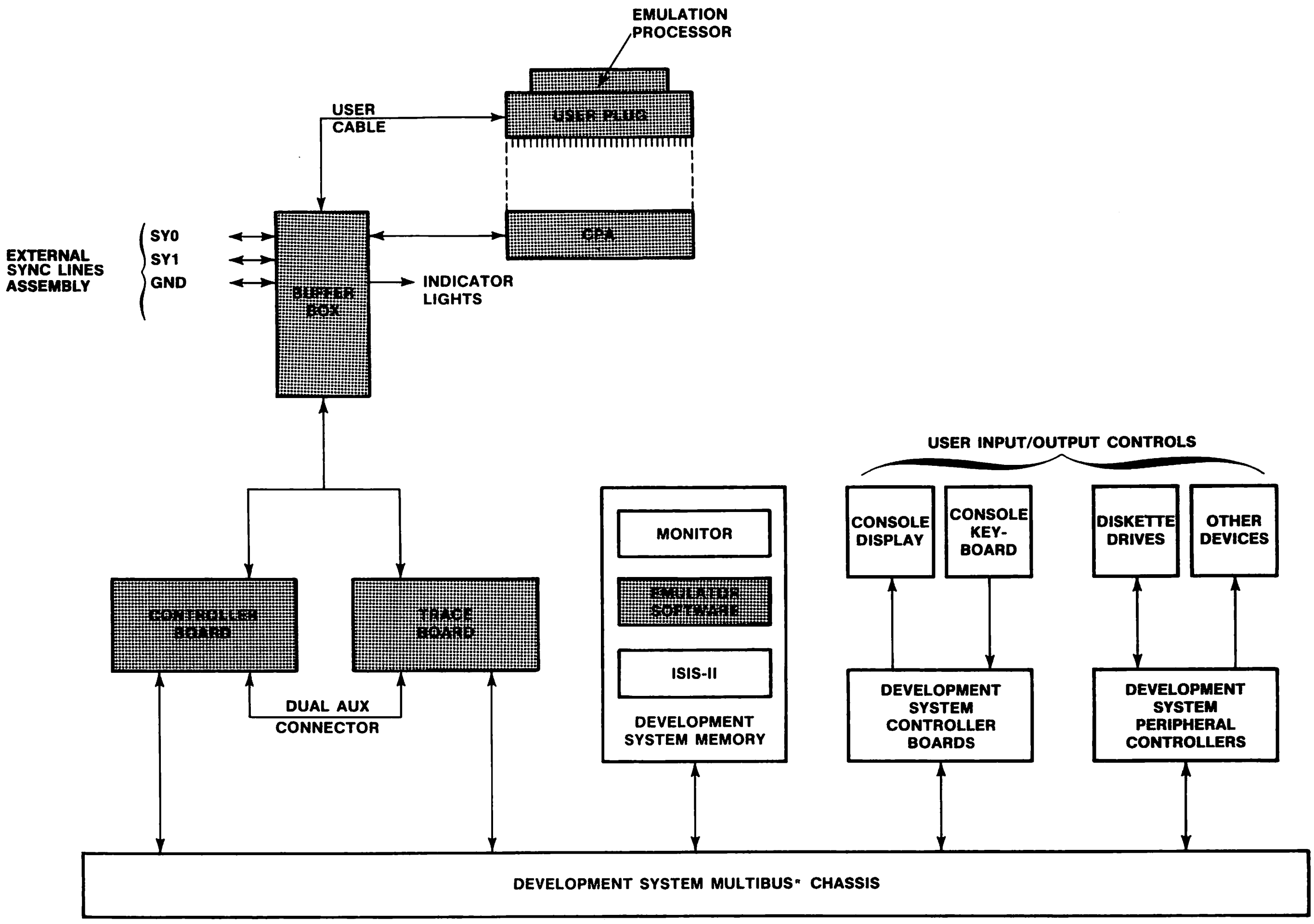


Figure 1-1. Emulator System Diagram

Software Components

The emulator software is furnished on diskette; the package contains both single-density and double-density versions. When the emulator in invoked, some of the software is loaded into RAM on the ICE boards; this software is the hardware control program to be executed by a processor on the controller board. Other software is loaded from diskette into RAM in the development system; this software handles console interactions, and is to be executed by the processor in the development system. The remaining software resides on the diskette to be called in when required; these are text files containing longer messages for display at the console, and the user confidence test software.

The emulator software is contained in the following files:

ICE51
ICE51.OVS
ICE51.OVH
ICE51.OVE

The user confidence test is contained in the following files:

CONF
CON51
CON51.OV1
CON51.OV2
CON51.OV3

The diskettes with the emulator software are not system diskettes (they do not have the ISIS-II utilities). You may wish to copy the emulator software to another diskette, either to have the system software and the emulator software on the same diskette or to create a working diskette with the emulator software and room for working files (user program modules, listings of diagnostic sessions, pre-programmed diagnostic command sequences).

Indicator Lights

Three LED lights on the top cover of the buffer box give a quick indication of emulator status:

Light Color	Status When Lit
Green	Emulation in progress.
Yellow	Ready for command from user.
Red	Hardware error detected.

Note that more than one light can be on at a given time, to indicate a combination of conditions. (Refer to appendix D, Hardware Specifications, for details on interpreting the buffer box lights.)

Development System Resources

Through the emulator, the user can access the hardware resources of the development system.

- Console display and keyboard. The emulator system is controlled by commands entered at the keyboard. The system displays prompts, messages, and the results of user commands.

- Development system memory (RAM). In addition to the ISIS-II and emulator software, development system RAM is available for the user symbol table, and a macro expansion workspace.
- Diskette drives. Access to the diskette drives is made through the ISIS-II diskette operating system.
- Other peripherals. The user can obtain a hard-copy listing of the diagnostic session on a line printer. Other development system peripherals of interest are PROM programmers and hard disk drives.

(Refer to appendix A, "Installation Procedures", for details on required and optional hardware.)

User Publications

The emulator package includes several kinds of instructional and reference material to assist the user in operating the system. These materials include:

- This *Operating Instructions* manual.
- *Getting Started With ICE-51,* a brief hands-on tutorial session designed for the less experienced user.
- A *Command Dictionary* listing each emulator command in alphabetical order with a summary of its syntax and usage.
- A file of HELP messages (part of the emulator software) that the user can request while the emulator is running.
- The package also contains a Dear Customer letter, including a list of sub-assemblies with part numbers.

Entering Commands

The ICE-51 system is controlled by commands typed in from the development system console. An ICE-51 command is a sequence of one or more command words; the words are separated by spaces. The sequence of words is terminated by a carriage return/linefeed. With two exceptions, the carriage return/linefeed causes the command to be interpreted, and if no syntax errors are detected the command is executed immediately. The exceptions are block commands and continuation lines. Continuation lines are discussed later in this section; refer to chapter 7 for details on block commands.

NOTE

Since all commands end with a final carriage return/linefeed, this entry is *not* shown explicitly in the command formats and examples. Carriage returns other than the final one are shown by the notation *"cr"*.

Prompts and Messages

The emulator displays an asterisk prompt (*) to show that it is ready to accept a command. Continuation lines and block commands precede the asterisk with other characters, as discussed later.

The emulator uses messages to inform the user about system operation, for example when beginning and terminating emulation and when an error or warning condition occurs. Most messages are self-explanatory, or are explained in this manual in connection with the commands that produce them; refer to appendix B for more information on the error and warning messages.

Correcting Errors in Commands

To correct errors in a command while the command is being entered, use the following control keys:

RUBOUT	Delete last character typed; repeat RUBOUT to delete more than one character.
CTRL X	Delete current line of command being entered.
ESC	Delete entire command being entered.
CTRL R	Echo command line being entered.
CR	Carriage return ends command line.
LF	Line feed also ends command line.

Once a command line has been ended with CR or LF that line can no longer be corrected. The following additional controls are used to pause and continue during lengthy displays:

CTRL S	Pause console display.
CTRL Q	Continue console display.

Continuation Lines

If you want to break a long command line into two or more input lines, use the ampersand (&) to request a continuation line. After the ampersand, the next carriage return is treated as an intermediate carriage return; nothing is executed until the final carriage return is encountered. Characters between the ampersand and the intermediate carriage return are ignored. The system begins the continuation line with a double asterisk (**) instead of the single asterisk.

Comments

You can add comments on any command line. To identify a comment, precede it with a semicolon (;). Any characters after the semicolon are not interpreted, but they will be "acknowledged" by the system (e.g., sent to LIST file, retained in MACRO definition). The semicolon can be the first non-blank character on a line, in which case no interpreting is done on that line. An ampersand within a comment does not act as a continuation character.

Command Entry From Files

The two ways to have ICE commands read in from an ISIS-II file are as follows:

1. The ICE-51 INCLUDE command causes a sequence of commands to be read in from a file. The INCLUDE command is executed from within the ICE system (i.e., after the ICE software has been invoked). Refer to chapter 7 for more details on the INCLUDE command.
2. The ISIS-II SUBMIT facility allows you to read in ICE commands from diskette file; the commands are executed as they are read in. When building the file to be run under SUBMIT, the first command should be the ICE51 invocation (to be executed by ISIS-II), and the last command should be EXIT (to return to ISIS-II). Refer to the *ISIS-II System User's Guide* for details on SUBMIT files.

Aborting Commands With the ESC Key

Pressing the ESC key aborts any command that is executing. Pressing ESC while a command is being entered (before the final carriage return) cancels that command. After the ESC has been processed, the prompt is issued. Refer to chapter 3 for details on using the ESC key to halt emulation.

Command Format Notation

Command format notation is a kind of diagram for ICE commands. The notation shows what command words to use, indicates parts of the command that can be omitted or included at your option, and shows the places in the command where you have a choice among several kinds of entry.

Elements of Commands

To present the notation, we need briefly to define some basic elements of commands.

character	Valid characters are the upper and lower case letters, the numerals, and a set of special characters.
token	A command word; one or more contiguous characters delimited by blanks or by context.
keyword	A token that is defined by the system; a command literal.
clause	A sequence of tokens that must be included or omitted as a unit.
entry	A token or clause.
user-entry	An entry whose exact form must be determined by the user; a class of entries.
menu	A choice of entries represented as a vertical list.
command	A sequence of entries terminated by a final carriage return.

Keywords and User Entries

A keyword is a command word with a fixed spelling and interpretation defined by the system. In the notation keywords are shown as ALL CAPS. All keywords may be abbreviated to the first three characters, and several may be shortened to one or two characters. The abbreviated forms of keywords are not shown in the format notation, but are used in some of the examples. (Refer to the *Command Dictionary* listed in the Preface for a list of keywords and their abbreviations.)

A user entry is a word or hyphenated phrase shown in *lower-case italic*. A user-entry represents a class of possible entries that contains too many variations to allow listing them all explicitly. The lower-case items themselves are not part of the command language.

This command format shows the notation for keywords and user entries:

EVALUATE *expression*

In this command format, EVALUATE is a keyword; it cannot be omitted from the command although it may be abbreviated. The term *expression* represents all the possible ways of obtaining a numeric result including single tokens and longer formulas. Here are some examples of this command:

```
*EVALUATE 10H
*EVAL .START + 25T
*E .TABLE + .INDEX * 10H
```

Here is another example; in this command format, the description of the user entry is a hyphenated phrase.

Format:

HELP *help-item*

Examples of command:

```
*HELP HELP
*HELP GO
```

Required and Optional Entries

Required entries are given in the notation without any enclosure. In the format EVALUATE *expression,* both the keyword EVALUATE and an entry corresponding to *expression* are required to make a valid command.

Optional entries are enclosed in square brackets. Any entry shown in square brackets can be omitted leaving a valid command. Here is an example of a command format containing required and optional entries:

Format

LOAD :*Fn*:*filename* [NOCODE] [NOSYMBOLS]

Examples of Command

```
*LOAD :F1:PROG01
*LOAD :F1:PROG01 NOCODE
*LOAD :F1:PROG01.HEX NOSYMBOLS
```

Repeatable Entries

Entries that can be repeated at user option are enclosed in square brackets and followed by an ellipsis (three adjacent periods). The most common use of this notational form is to represent lists of entries separated by commas. Here is an example of a format containing a list:

Format

REMOVE MACROS {:*macro-name* {, :*macro-name*} ...}

Examples of Command

```
*REMOVE :SKIP
*REM :SKIP, :MEM, :TEST
```

In this command format, the list of macro names can be omitted entirely, or it can have just one macro name, or it can be several macro names separated by commas.

Choices Of Entries

A choice of entries is indicated by a vertical list of the entries enclosed in curved braces or square brackets; such a list is called a "menu". A menu enclosed in square brackets means "select none or one". For example,

Format

$$\text{PRINT}\begin{Bmatrix}\text{ALL}\\ \text{[-] } \textit{number-of-lines}\end{Bmatrix}$$

Examples of Command

```
*PRINT
*PRINT ALL
*PRI 10
*P -25
```

The menu shows that you can omit the entry after PRINT, or you can select the keyword ALL, or you can enter a "number of lines" optionally with a minus sign preceding it.

A menu enclosed in square brackets and followed by an ellipsis (...) means "select none, one, or more than one, in any order". For examples of commands that use this format, see Chapter 7, "Programming With Block Commands".

A menu enclosed in curved braces means "select one and only one". In the following example, the choice is between the keyword SYMBOLS and a list of symbolic references; one of these two kinds of entries must be included to form a valid command.

Format:

```
        {SYMBOLS                                              }
REMOVE  {                                                     }
        {symbolic-reference [, symbolic-reference] ...        }
```

Examples of Command

```
*REMOVE SYMBOLS
*REMOVE .LOOP
*REM .LOOP, .DONE
```

A menu can contain entries that themselves contain menus. Here is an example of such a command format:

```
      {FOREVER                                  }
      {SY1                                      }
TR =  {  [      ]  {QR0                }        }
      {  [AFTER ]  {QR1                }        }
      {  [TILL  ]  {QR                 }        }
      {  [      ]  {match-condition    }        }
```

This command allows several kinds of entry after the equals sign:

1. The keyword FOREVER.
2. The keyword SY1.
3. One of the keywords QR0, QR1, or QR.
4. An entry specifying a match condition.
5. The entries described in 3 and 4 can be preceded by one of the keywords AFTER or TILL.

Here are a few examples of this command, but they do not demonstrate all possible "branches" of the syntax. More examples of this command are in chapters 4 and 6.

```
*TR =  FOREVER
*TR =  SY1
*TR =  QR0
*TR =  AFTER 0
*TR =  TILL QR1
```

Single-Character Tokens

The emulator command language contains single-character tokens that serve as delimiters, punctuation marks, operators, and other uses (table 1-1). These are shown in **bold** in the command formats.

For example:

```
:macro-name
```

Example of command:

```
:TEST1
```

In this command the colon (:) identifies the name that follows as the name of a macro to be executed.

NOTE

The one-line assembler uses several additional one-character tokens, as discussed in chapter 5.

Table 1-1. Single-Character Tokens

Token	Meaning
.	(Period); Identifies symbolic reference.
,	(Comma); Separates items in a list.
(and)	(Parentheses); Control order of evaluation in expressions.
+	(Plus sign); Unary plus and addition sign.
-	(Minus sign); Unary minus and subtraction.
*	(Asterisk); Multiplication and in HELP command.
/	(Slash); Division.
=	(Equals sign); Equal relation operator and assignment operator.
>	(Right angle bracket); Greater than operator.
<	(Left angle bracket); Less than operator.
"	(Quotation marks); Encloses mnemonic constants in expressions.
'	(Apostrophe); Encloses string characters.
:	(Colon); Identifies macro name.
%	(Percent); Identifies formal parameter in macro definition.

Definitions

The terms *address, partition, expression,* and *string* occur in several different places in the manual. Here are brief definitions for these terms.

Address

The term *address* in an emulator command format can be *numeric constant,* a *symbolic reference,* a *keyword reference,* a *trace reference,* or a calculation using mathematical operators.

Numeric Constant

A numeric constant is a number consisting of one or more digits, and (optionally) a one-character explicit radix (suffix) to identify the number base. The elements of numeric constants are summarized in Table 1-2.

A numeric constant entered from the console with an explicit radix is interpreted accordingly. If you omit the explicit radix, the system uses the current default radix; for addresses the initial default is hexadecimal (H). The digits in the number must be valid for the radix that is applied, or an error results. In this manual, most numeric constants are shown with explicit radixes for clarity. Numbers shown without explicit radixes are *decimal* (for example, 65,535), except for the numbers 0 and 1, to which any radix can apply.

Table 1-2. Elements of Numeric Constants

Number Base	Valid Digits	Explicit Radix	Example
Binary (base 2)	0, 1	Y	11110011Y
Octal (base 8)	0 - 7	Q, O	363Q
Decimal (base 10)	0 - 9	T	243T
Hexadecimal (base 16)	0 - 9, A - F	H	00F3H
Decimal multiple of 1024T	0 - 9	K	4K

Symbolic Reference

A system symbol or a user symbol preceded by a period (.). System symbols correspond to the predefined symbols for addresses in the assembly language; examples are .RESET for address 0000H in code memory, .SP for address 81H in register memory (Stack Pointer), and .CY for address 0D7H in bit-addressable memory (Carry Flag). User symbols are labels from the user program symbol table, and additional symbols defined during the emulation session. Refer to chapter 2 for discussion and examples of user symbols, and to chapter 5 for details on the system symbols.

Keyword Reference

An emulator command word for a register that contains an address. For example, the keyword PC refers to the code memory address pointed to by the program counter, and the keyword DPTR refers to the external address in the Data Pointer register. Other examples of keyword references to addresses appear in chapter 5.

Trace Reference

A construct with one of the two formats FRAME ADDR or FRAME XADDR. The *address* is the 16-bit Code Address field (for ADDR) or the 16-bit External Address field (for XADDR) from the frame of trace pointed to by the trace display pointer. (Refer to chapter 6 for details on trace references.)

Calculated Addresses

You can also enter an address as a formula with mathematical operators; the system performs the calculation and uses the result as the *address*. In this sense, an address is a form of *expression* as discussed later in this chapter. In an expression representing an address, only arithmetic operators (+, -, *, /, MOD) can appear outside parentheses.

Here are some examples:

```
.START + 7
.TABLE + 10H * 24T
```

Partition

The term *partition* in a command format means any of the following types of entry:

- A single *address*.
- A range of addresses expressed with the format:
  address1 TO *address2*

For example:

```
100H TO 110H
.START TO .START + 10H
```

In this type of partition, *address2* must be greater than or equal to *address1*, or an error results.

- A range of addresses expressed with the format:
  address LENGTH *number-of-addresses*

For example:

```
1100H LENGTH 10H
```

Valid entries for *number-of-addresses* can have any of the forms described above for *address*. However, an error results if the sum of *address* and *number-of-addresses* exceeds the size of the memory space addressed.

Expression

The term *expression* in a command format means any of the following types of entry:

- A single number, constant, or reference; for examples:

```
0       ;Number without explicit radix.
0100H   ;Hexadecimal numeric constant.
'RETI'  ;One-byte mnemonic constant.
'A'     ;String constant (one-character string).
.START  ;Symbolic reference.
DPTR    ;Keyword reference.
```

- A formula applying mathematical operators to numbers, constants, and references as operands. The system performs a 16-bit calculation, using operator precedence and left-to-right order to determine the sequence of operations. The precedence of operations can also be controlled with pairs of parentheses.

There are four kinds of operator: arithmetic operators, content operators, relational operators, and logical operators, in descending order of precedence.

Arithmetic Operators

The arithmetic operators are:

Operator	Operation	Precedence (1 = highest)
+	Unary plus	1
-	Unary minus (2's complement)	1
*	Multiplication	2
/	Integer division	2
MOD	Modulo reduction	2
+	Addition	3
-	Subtraction	3

Content Operators

Each of the types of memory space has a corresponding content operator. A content operator treats its operand as an address in its type of memory space; the result is the contents of that address. The content operators are as follows:

Content Operator	Type of Memory	Example	Precedence
CBYTE	Code memory	CBYTE 0100H	4
DBYTE	On-chip data memory	DBYTE 10H	4
RBYTE	Register memory	RBYTE .SP	4
XBYTE	External data memory, Verified	XBYTE .TABLE	4
PBYTE	External data memory, Unverified	PBYTE C000H	4
RBIT	Bit-addressable memory	RBIT .CY	4

Content operators have precedence next lower after MOD (corresponding to precedence "4" in the numbering scheme given above). Refer to chapter 5 for more examples of the use of content operators.

Relational Operators

The relational operators are:

Operator	Relation	Precedence
=	Is equal to	5
>	Is greater than	5
<	Is less than	5
>=	Is greater than or equal to	5
<=	Is less than or equal to	5
<>	Is not equal to	5

The relational operators have precedence "5", next lower after the content operators. The result of any relational operation is either TRUE (FFFFH) or FALSE (0000H).

Logical Operators

The logical operators are:

Operator	Operation	Precedence
NOT	1's complement	6
AND	Bitwise AND	7
OR	Bitwise OR	8
XOR	Bitwise exclusive OR	8

The logical operators work on 16-bit operands and produce 16-bit results. The NOT operator has precedence next lower after the relational operators; AND is next lower after NOT; OR and XOR have the lowest precedence of any operators.

How Expressions Are Evaluated

To evaluate an expression, the system scans the expression iteratively from left to right, one iteration for each operator in the expression. The series of scans terminates in two ways:

- When nothing remains except a single number, the result of the evaluation.
- When a syntax error or other error occurs.

On each iteration, the scanner identifies the operator that must be applied next. This operator can be unary (takes one operand) or binary (takes two operands). This next operator is always the *leftmost* operator with *highest precedence* that is enclosed in the innermost pair of parentheses.

If this next operator is unary, its operand must follow it and must be a number. If so, the operator is applied to produce a number as the result; if not, an error results.

If the next operator is binary, its two operands must both be numbers; the operations then produces a numeric result. If both operands are not numbers, an error results.

If the next operator does not have the required number of operands, an error results.

A pair of parentheses is cleared when it contains just a single number; that is:

(number) → number

After performing any operation, the numeric result becomes an operand for the next scan. Parentheses are cleared before the next scan begins.

NOTE

When an expression is evaluated, the result has 16 bits. If the number of bits in the result exceeds the number of bits required, the low-order bits are used. For example, when an expression is used to represent an 8-bit address, only the low 8 bits of the result are used. Please refer to the *Command Dictionary* for more information on expressions.

String

The term *string* in a command format means a sequence of one or more alphanumeric characters enclosed in apostrophes ('); for examples:

'ABCDE'

'X'

'THIS IS A STRING'

When a *string* is used to set the contents of memory, the value is the one-byte ASCII value of each character. If the string has more than one character, the system uses consecutive addresses to store the ASCII values after the first. Examples appear in chapter 5.

As the third example above shows, the string can contain blanks. It can also contain apostrophes, but they must be distinguished from the ones that delimit the string itself; use a double apostrophe ('') for this purpose; for example:

```
    'WHAT''S UP?'            ;As entered — stored as WHAT'S UP?
```

CHAPTER 2 UTILITY COMMANDS

The commands in this chapter are:

- *ICE51 Invocation:* Initiate the emulator system from ISIS-II.
- *EXIT Command:* Terminate the emulation session and return control of the console to ISIS-II.
- *HELP Command:* Obtain displays that briefly explain the emulator commands.
- *LOAD Command:* Load the user program (and symbol table if desired) into the emulator's memory from object file on diskette.
- *SAVE Command:* copy the contents of emulator code and symbol table memory to diskette file.
- *Symbolic Reference Commands:* Define, change, display, reference, and remove user symbols.
- *LIST Command:* Copy the output from the emulation session to a device other than the console.
- *RESET ICE Command:* Restore the emulator hardware to its initial state (as after invocation).
- *RESET CHIP Command:* Place emulation processor in its RESET condition.
- *SUFFIX Command:* Select the implicit radix for numbers entered at the console.
- *BASE Command:* Select the default radix for certain kinds of numeric displays on the console.
- *EVALUATE Command:* Display the result of any expression in four radixes, as ASCII, and symbolically.

ICE51 Invocation

The emulator software driver is contained in an executable ISIS-II file named ICE51. When this file is executed, the emulator hardware is initialized and the user can control the emulator through the console. To invoke the emulator system follow these steps:

- Install emulator hardware following the procedure in appendix A of this manual.
- Turn power on to the development system and diskette drives, and "boot" the ISIS-II system as described in your ISIS-II manual.

NOTE

Make sure your emulator software diskette has a Write-Enable tab installed before attempting to invoke the software.

- With a one drive system, copy the emulator software to a system diskette, and place the system diskette in drive 0. Enter the command ICE51.

NOTE

The emulator software files to copy are as follows: ICE51; ICE51.OVS; ICE51.OVE; ICE51.OVH; CONF; CON51; CON51.OV1, CON51.OV2, CON51.OV3.

- With a larger system, place a system diskette in drive 0 and the emulator software diskette in drive *n* (*n* = 1 to 9). Enter the command :F*n*:ICE51. (Refer to the note on WORKFILES below.)

NOTE

Drive numbers shown are for diskette-only systems. Hard disk users refer to the ISIS-II manual for drive numbers.

- In response to the ICE51 invocation, the ISIS system loads the main emulator software module into memory in the development system, then issues a sign-on message: ISIS-II ICE-51 Vn.n.
- Next, the system copies the emulator's hardware control program to RAM on the emulator boards, then issues the asterisk prompt.

Here is an example of the emulator invocation sequence, assuming the emulator software diskette is on drive 1.

```
ISIS-II Vn.n                          ;ISIS-II sign-on.
-:F1:ICE51                            ;Emulator invocation.
ISIS-II  ICE-51  Vn.n                 ;Emulator sign-on.
FOR COMMAND ENTRY ASSISTANCE, TYPE HELP
*                                     ;Emulator prompt.
```

NOTE

When you are using MACRO commands (chapter 7), the emulator system opens a file named MAC.TMP to hold the macro definitions. By default, the drive used for this file is the one named in the invocation command. To specify a drive for this temporary file, the format of the invocation is:

```
:Fn:ICE51 WORKFILES (:Fm:)
```

In this format, *n* is the number of the diskette drive containing the emulator software and *m* is the number of the drive that will be used for file MAC.TMP. The workspace file is temporary; it is removed at the EXIT command. However, if the workfile disk already contains a file named MAC.TMP, that previous file will be lost.

EXIT Command

To end the emulation session and return control of the console to ISIS-II, enter the command:

```
*EXIT
```

The EXIT command halts all emulations, resets the emulator hardware, closes all files opened by or through the emulator, then returns control to ISIS-II.

HELP Command

The emulator system includes a file with brief explanations of the emulator commands and other kinds of entries. The HELP commands display items from this file on the console. The HELP facility is designed to reduce the need for printed reference material by the somewhat experienced user.

Each item in the HELP file is identified by name. To display a menu of the names of the items in the HELP file, enter the command:

```
*HELP
```

To obtain an explanation of the notation used in the HELP command, type:

```
*HELP HELP
```

To obtain general information about entering commands at the console, type:

```
*HELP INFO
```

To display the explanation of any item, type HELP followed by the name of the item exactly as it appears in the menu. For example:

```
*HELP ADDRESS
```

To display several items, type HELP followed by the list of items separated by commas. For example:

```
*HELP GO,GR,BR
```

To display all the HELP messages, type HELP followed by an asterisk. For example:

```
*HELP *
```

The items are displayed in the order they appear on the file. This facility is useful for obtaining a hard-copy of the HELP explanations (see the LIST command later in this chapter for obtaining a hard-copy printout).

LOAD Command

The user program to be emulated typically resides in a hexadecimal object file generated by the ASM51 assembler. If the DEBUG assembler control has been used, the object file also contains the program's symbol table.

The LOAD command copies the hexadecimal object code from a file into code memory, using the emulator memory map to identify the physical location of every address (see Code Memory Access, chapter 5). When the load is complete, the program counter (PC) is set to the load address from the file (address of first executable instruction).

The LOAD command also copies the user symbol table (if present on the file) into the development system memory where it can be accessed by the emulator. The symbols are checked for duplications of the system symbols (which are not loaded if they occur in the user symbol table), but does not check for duplications of user symbols already in the emulator's symbol table.

The general syntax of the LOAD command is:

LOAD :F*n*:*filename* [NOCODE] [NOSYMBOLS]

In this command format, *n* is the number of the diskette drive (*n* = 0 to 9), and *filename* is the name of the hexadecimal object file (including the extension, if any). The object filename is assigned by the user; ASM51 assigns the extension .HEX to the object files it generates.

For example, assume the user code is in an object file named PROG.HEX and the diskette with the file is in drive 1. Then the command would be:

```
*LOAD :F1:PROG.HEX
```

To load only the object code and *not* the symbols from the file, add the modifier NOSYMBOLS to the command; for example:

```
*LOAD :F1:PROG.HEX NOSYMBOLS
```

To load just the symbol table but not the code, use the modifier NOCODE:

```
*LOAD :F1:PROG.HEX NOCODE
```

Object programs to be loaded must be in the hexadecimal file format generated by the ASM51 assembler (and by the SAVE command discussed below).

NOTE

Please see the Preface to this manual for references on assembler controls, ISIS filenames, and hexadecimal file format.

SAVE Command

The SAVE command copies user code from emulator memory to a hex format object file. The command also copies the user symbol table from memory to the object file if desired. The format of this command is:

SAVE :F*n*:*filename* $\begin{bmatrix}\textit{partition}\\ \text{NOCODE}\end{bmatrix}$ [NOSYMBOLS]

In this format, *n* is the number of the drive containing the target diskette (*n* = 0 to 9). The *filename* is assigned by the user, including an extension if desired. If the diskette does not have the given filename in its directory, ISIS-II creates the file and opens it for write. If the file does exist, it is overwritten and the previous contents are lost. The SAVE operation does not alter the program code or symbol table in memory. Programs are saved in hex file format, so any programs saved with SAVE can be loaded with LOAD.

A SAVE command with no modifiers (the modifiers are shown in brackets above) saves the user code in the ranges that were affected by the most recent LOAD or SAVE command within the current session. If no LOADs or SAVEs have been performed, nothing is saved by this form of the command. For example, assuming the command LOAD :F1:PROG.HEX from the previous section, we can save this code in another file named PROG01.HEX with the command:

```
*SAVE :F1:PROG01.HEX
```

To save the code in a range of your choosing, use a command with the format:

SAVE :F*n*:*filename partition*

See chapter 1 for details on partitions.

Here are some examples of this kind of SAVE command:

```
*SAVE :F1:PROG02.HEX 0 TO 1FFH
*SAVE :F1:PROG03.HEX 200H LENGTH 80H
```

The forms with *partition* also save the user symbol table, if one is present in the emulator memory.

To save the symbol table only, use the modifier NOCODE; for example:

```
*SAVE :F1:PROG.HEX NOCODE
```

The modifiers NOCODE and *partition* are mutually exclusive; if one is included, the other may not be included.

To save the program code only and not the symbol table, use the modifier NOSYMBOLS:

```
*SAVE :F1:PROG.HEX NOSYMBOLS
*SAVE :F1:PROG01.HEX 0 TO 1FFH NOSYMBOLS
```

Refer to chapter 1 for more about addresses and partitions.

Symbolic Reference Commands

A *symbol* is a name for a particular memory address or for some other value. A *symbolic reference* in the emulator language is the name of a symbol preceded by a period (.). For example, if the user program contains a symbol START for code address 0100H, the symbolic reference .START would represent 0100H when it is used in a command such as PC = .START.

The emulator maintains two kinds of symbol tables, one for user-defined symbols and another for system-defined symbols.

User-Defined Symbols

The emulator system maintains a symbol table for user-defined symbols. The user table contains the name of all user symbols in the order they were loaded or defined. The user table is initially empty.

The user table can receive the symbol table generated by the assembler when the user program is loaded into the emulator with the LOAD command. The assembler output file contains the user symbol table in loadable format when the DEBUG control is added to the assembler invocation. The system loads the symbol table in the order the symbols appear in the program. The segment type (CSEG, DSEG, etc.) for user symbols is *not* loaded; user symbols in the emulator system are untyped. The system checks for and rejects system symbols in the user table.

The user can also define symbols from the console to be added to the user symbol table. To define a symbol, use a command with the format:

DEFINE *.symbol-name = expression*

Examples:

```
*DEFINE .START = 0100H
*DEFINE .LOOP = 0200H
```

The name of the new symbol (*symbol-name*) can be defined with a maximum of 31 characters. The first character in the new *symbol-name* must be an alphabetic character, or one of the two characters @ or ?. The remaining characters after the first can be these characters or numeric digits.

The new symbol name in a DEFINE symbol cannot duplicate a symbol name already present either in the user table or in the system symbol table.

When you define a new symbol, you also specify the value corresponding to it in the table. You can treat the value you assign as an address or as a numeric value for use other than addressing.

Once a symbol has been defined or loaded, any reference to that symbol is equivalent to supplying its corresponding address or value. If the table contains duplicate symbol-names, the one that occurs nearest the top of the table is supplied.

To display the value from the symbol table corresponding to any symbol, enter a symbolic reference (the symbol-name preceded by a period). For example:

```
*.START
.START = 0100H
```

To display the entire user symbol table, enter the command SYMBOLS. For example:

```
*SYMBOLS
.START = 0100H
.LOOP = 0200H
```

You can change the value corresponding to an existing symbol by entering a command of the form:

.symbol-name = expression

Example:

```
*.LOOP = 0300H
```

To delete one or more symbols from the table, use a command of the form:

REMOVE *.symbol-name* [, *.symbol name*] ...

Example:

```
*REMOVE .START, .LOOP
```

Note that deleting a symbol from the user symbol table makes that symbol inaccessible to the emulator, but does not affect the program code.

To delete the entire user symbol table, enter the command:

```
*REMOVE SYMBOLS
```

Multiple Symbols

When symbols are loaded with the LOAD command, the system does not check the symbols being loaded for duplication of symbols already in the user table. By contrast, when you define a symbol using the DEFINE command, the system detects duplicate symbols as an error; the assembler also disallows duplicate symbol definitions. Thus the LOAD command is the only way multiple symbols can occur in the user table.

In normal operation, the system searches the user table from the earliest entry, and terminates the search when the first occurrence of the desired symbol is reached. Under this condition, access to the second or later instances of a multiple symbol will not be possible.

The emulator system allows you to combine symbolic references to user symbols; a multiple reference accesses a multiply-defined symbol. For example, suppose the table contains two symbols with the name .INDEX; the first .INDEX represents address 0100H and the second .INDEX represents address 0200H. To display the first symbol, the command is:

```
*.INDEX
.INDEX=0100H
```

To display the second version, the command is:

```
*.INDEX.INDEX
.INDEX=0200H
```

Any number of user symbols may be combined to produce a multiple reference of this kind. System symbols may not be used in multiple references.

System-Defined Symbols

The emulator system maintains tables of system symbols for addresses corresponding to the predefined symbols in the assembly language. The format of a system symbol in the emulation language is the name of the assembler symbol preceded by a period (.). For example, the assembler symbol "SINT" refers to the address of the serial port interrupt vector in code memory; in the emulator language, the corresponding system symbol is ".SINT".

The system symbols are loaded with the emulator software at invocation. They are not part of the user symbol table, and are not affected by the user symbol table commands. The system symbols may not be changed or removed. They are not displayed with the user table (SYMBOLS command), but can be displayed individually by entering the system symbol with its leading period; for example:

```
*.SINT
.SINT=23H
```

System symbols correspond to addresses in register memory, in bit-addressable memory, or in code memory. When you refer to a system symbol in an emulator command, the system uses the corresponding numeric address (for example, PC, = .SINT is equivalent to PC = 23H). Complete listings and more examples of system symbols appear in the corresponding sections of chapter 5 (see "Register Addresses", "Bit Addresses", and "Code Addresses").

Symbolic Displays of Addresses

To assist in symbolic debugging, the system can use a symbolic format for displays of addresses. The symbolic display is obtained by searching the user table or the system tables looking for a symbol whose value either matches the address or is closest to but less than the address. Table 2-1 shows the commands that can produce symbolic displays of addresses, and the details on how the symbolic display is produced in each case.

Most of the commands produce searches of the user table only. The remaining commands search the user symbol table first, then search one of the system symbol tables if no user symbol matches. The user symbol table does not preserve the segment types from the assembler; system tables are segmented according to type (code, register, bit). In case of a "tie" (two symbols match), the symbol occurring first in the table will be used. Symbols occur in the user table in the order they were defined or loaded. As shown in table 2-1, system symbols are used only for addresses appearing as operands in disassembled instructions (DASM command, INSTRUCTION mode trace displays). The system symbol table is searched only when the user table has been searched without finding a match. Since the disassembler "knows" what kind of operand address it is trying to match, only the appropriate system symbol table is searched (code, bit, or register).

Symbolic display applies only to the commands in table 2-1. All other addresses are displayed in hexadecimal. In addition, you can disable symbolic displays and have addresses display in hexadecimal; to disable symbolic displays enter the command:

```
DISABLE SYMBOLIC
```

To re-enable symbolic displays after they have been disabled, enter the command:

```
ENABLE SYMBOLIC
```

Symbolic displays are initially enabled.

NOTE

Symbolic display is always enabled in displays with the EVALUATE command. DISABLE SYMBOLIC has no effect on these displays.

Table 2-1. Types of Symbolic Displays

Symbolic Display Format	Comparison Rule	Symbol Tables Searched	Default If No Symbol Found	Command/Display Content	Notes
.symbol-name[+*nnnn*H]	Closest to but not greater than the address.	User symbols only.	Hexadecimal	*content-operator address* (Address portion of display) [ch. 5]	
				Display of PC after emulation breaks. [ch. 3]	
				EVALUATE *expression* (Right most column of display) [ch. 2]	5
.symbol-name	Exact match.	User symbols only.	No symbolic display	*content-operator partition* (more than one address) [ch. 5]	1,2
				DASM *partition* (address field of disassembled instruction) [ch. 5]	1,3
				PRINT (LOC field in INSTRUCTIONS mode, ADDR field in FRAMES mode) [ch. 4,6]	1,3
		User symbols first, then system symbols if no user symbol matches.	Hexadecimal	Addresses as operands in disassembled instructions (DASM, PRINT in INSTRUCTION mode) [ch. 4,5]	4

NOTES:
1. Display appears on line preceding address.
2. Display line for memory contents restarts after each symbolic display
3. The symbol must match the first address in the instruction.
4. A system symbol is displayed if it matches and is the same type (code address, register address, or bit address) as the operand.
5. DISABLE SYMBOLIC has no effect on this display.

LIST Command

The LIST command saves a record of the emulation session, including high-volume data such as trace data, on a hard-copy device or on a diskette file. Only one device or file other than the console can be specified (active) at a given time.

The format of the LIST command is:

```
     ⎧:CO:        ⎫
     ⎪:LP:        ⎪
LIST ⎨:TO:        ⎬
     ⎪:HP:        ⎪
     ⎩:Fn:filename⎭
```

Examples:

```
*LIST  :F1:MAY30.LOG
*LIST  :LP:
*LIST  :CO:
```

In this format, *n* is the number of the drive (*n* = 0 to 9), and *filename* is the name of the diskette file assigned by the user.

The initial device is :CO:, output to the console. Other devices that can be specified are a line printer (:LP:), high-speed paper tape punch (:HP:), and teletypewriter printer (:TO:).

Instead of a hard-copy device, a diskette file can be specified for LIST. If the output is to a diskette file, the file is opened when the LIST command is invoked, and output is stored from the beginning of the file, writing over any existing data. Specifying a new file or device in a later LIST command closes any existing open file and avoids over-writing any more data.

When LIST is in effect (with a device or file other than :CO:), all screen output from the emulation session including system prompts, commands, and error messages, is sent both to the device or file and to the console display. To restore output to the console only (no other device), use the command LIST :CO:.

RESET ICE Command

The command RESET ICE causes a reset of the emulator hardware to its initial state as after the ICE 51 invocation. RESET ICE resets the emulator map to 0000H, 1000H (see MAP command, chapter 5). The reset does not affect any other accessible emulator registers, and does not reset the emulation processor. However, during RESET ICE the emulation processor executes a few random instructions; these can affect on-chip registers or ports. This command is useful for attempting recovery from a hardware error in the emulator (refer to appendix B, "Error Conditions", for details).

RESET CHIP Command

The command RESET CHIP places the emulation processor in its reset state (see Table 2-2). Among the processor flags that are reset with this command, the Interrupt-In-Progress flag is reset. This is the only way the user can clear this flag interactively. This is a simulated reset as far as the user system is involved; no reset signal is sent to the user system. On-chip data memory is not affected.

Table 2-2. State of Emulation Processor After RESET

Register	Value
Accumulator	00H
Multiplication Register	00H
Data Pointer	0000H
Interrupt Enable	00H
Interrupt Priority	00H
Port 0	0FFH
Port 1	0FFH
Port 2	0FFH
Port 3	0FFH
Program Counter	0000H
Program Status Word	00H
Serial Port Control	00H
Serial I/O Buffer	undefined
Stack Pointer	07H
Timer Control	00H
Timer Mode	00H
Timer 0 Counter	0000H
Timer 1 Counter	0000H

SUFFIX Command

A numeric constant entered from the console without an explicit radix is interpreted according to the implicit radix that applies to the context. In most contexts, the initial implicit radix is hexadecimal (H); in these contexts, you can change the implicit input radix by entering a command with the format:

```
         ⎧ H ⎫        ;Hexadecimal
SUFFIX = ⎨ T ⎬        ;Decimal
         ⎨ Q ⎬        ;Octal
         ⎩ Y ⎭        ;Binary
```

For example, to enter a series of values in decimal radix, the commands would be:

```
*SUFFIX = T                        ;Change to decimal radix
*DBYTE 50 = 23, 67, 88, 23, 68, 45 ;Enter the decimal numbers.
*SUF = H                           ;Restore hexadecimal radix.
```

To display the current implicit radix, enter the command:

```
 *SUFFIX
H                                  ;One-character display, as described above.
```

Certain command contexts assume an implicit decimal (T) radix. The SUFFIX command has no effect on these contexts. In addition, the MAP command (chapter 5) requires hexadecimal (or a multiple of 1024 with explicit K radix), and is not affected by the SUFFIX command.

BASE Command

Numeric information such as addresses and memory contents is displayed in the current console output radix. The initial output radix is hexadecimal (H). To change to a new output radix, use a BASE command with the following format:

```
       ⎧ H     ⎫ ;Hexadecimal
       ⎪ T     ⎪ ;Decimal
BASE = ⎨ Q     ⎬ ;Octal
       ⎪ Y     ⎪ ;Binary
       ⎩ ASCII ⎭ ;ASCII character for each byte
```

Examples:

```
*BASE = T
*BAS = ASCII
*BASE = Y
```

To display the current output radix, enter the command:

```
*BASE
H
```

The display consists of a single character: H, T, Q, Y, or A for ASCII.

EVALUATE Command

The EVALUATE command performs mathematical computation and displays the results; the format is:

EVALUATE *expression*

Example:

```
*EVALUATE FFH = 1
100000000Y 400Q 256T 100H '' .START
```

The system evaluates the expression and displays the result in the four bases (binary, octal, decimal, and hexadecimal), as ASCII characters, and as an address. For ASCII, the characters are enclosed in apostrophes ('); printable characters are displayed (ASCII codes 20H through 7H after bit 7 is masked off), while non-printing characters are suppressed. As an address, the symbolic form is used if the system can identify a user-defined symbol that is less than or equal to the address; otherwise, the address is in hexadecimal. Refer to chapter 1 for details on expressions, and to chapter 2 for symbolic displays.

CHAPTER 3
EMULATION COMMANDS

Real-Time Emulation

The ICE-51 system emulates the microcontroller in the user prototype with a special version of the 8051 located at the end of the user cable. This special 8051 is called the *emulation processor.*

During real-time emulation, the emulation processor fetches instructions from code memory and executes them. Emulation begins at the address in the program counter (PC) and halts ("breaks") either by user abort (ESC key) or on a condition set by the user. Emulation breaks after completing the current instruction. An emulation timer records execution time for display after emulation breaks. Trace data, a picture of chip activity, is collected under conditions set by the user.

Real-time emulation operates at the 12.0 MHz frequency from the Crystal Power Accessory furnished with the emulator system, when this accessory is installed on the buffer box. In this mode, the emulator does not interact with the user system. When the user plug is inserted in the user system, the operating frequency is derived from the user hardware.

Basic GO Commands

To start emulation at the current program counter address, enter the command GO; the system begins emulation with a message; for example:

```
*GO
EMULATION BEGUN
```

To halt emulation at any time, press the ESC key on the keyboard. The system responds with two messages:

```
EMULATION TERMINATED, PC=address
PROCESSING ABORTED

*
```

The first message is displayed any time emulation breaks. The *address* in the program counter is the next instruction to be executed when emulation resumes. The second message is displayed whenever command processing is halted by pressing the ESC key. After the second message the system displays the prompt for the next command.

To start emulation at a particular address of your choice, use a GO command with a FROM clause; the format is:

```
GO FROM address
```

This one command is equivalent to the sequence:

```
PC = address          ;See chapter 5 for this command.
GO
```

Here are some examples of this form of GO command:

```
*GO FROM 100H
*GO FROM .START
*GO FROM .COUNTER + 10H
```

Using Breakpoints to Halt Emulation

A breakpoint is a halt-condition involving a particular state of the emulation processor that is of interest to you. In simplest form, a breakpoint is specified as the address of an instruction (more precisely, the address of the opcode or first byte of an instruction); if the breakpoint instruction is executed, emulation breaks after completing that instruction.

You can specify a breakpoint when you start emulation. The format of this command is:

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

For example:

```
*GO TILL 100H
```

This command starts emulation at the current PC; if the instruction at code address 0100H is executed, emulation halts after completing that instruction. If this instruction is never executed, you must press ESC to halt emulation. Here are some more examples of this kind of breakpoint:

```
*GO FROM 0 TILL 30H
*GO FROM .START TILL 100H
*GO FROM .LOOP TILL .DONE
*GO TILL .SUBR - 1
```

You can have two breakpoints active at the same time. To specify two breakpoints, use a command with the following format:

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

For example:

```
*GO TILL 0100H OR 0200H
```

During this emulation, if an opcode is fetched either from address 100H or from address 200H, emulation breaks after completing that instruction. Here are some more examples using two breakpoints:

```
*GO FROM 0 TILL .DONE OR 200H
*GO TILL .DONE OR .SUBR - 1
```

Initially, no breakpoints are active. To cancel all breakpoints and restore the initial condition, use a command with the format:

```
GO [FROM address] FOREVER
```

For example:

```
*GO FOREVER
*GO FROM .START FOREVER
```

Following either of these commands, you must use the ESC key to halt emulation.

NOTES ON BREAKPOINTS:

1. In the format TILL *address*, the address must be the address of an opcode, the first address in an instruction. Thus, if you enter GO TILL 101H and address 101H actually contains an operand byte, emulation will not break when address 101H is accessed, and you must press ESC to halt emulation. (Other command forms allow you to break on operand addresses, on opcode and operand values, on port values, and on external signal SY0 IN; refer to chapter 6 for these commands.)

2. Once activated, a breakpoint remains in effect until canceled. Setting a new breakpoint implicitly cancels any previous breakpoints.

3. If the program counter contains the breakpoint address when emulation begins (for example, *GO FROM 100H TILL 100H), emulation breaks after executing that one instruction.

Using a Mask as a Breakpoint

A mask is a hexadecimal, binary, or octal numeric constant in which one or more of the digits are "don't-cares"; a don't-care digit is represented with the character X. (The digits that are not X can be called "care" digits.) When used as a breakpoint, a mask represents the several addresses that share the care digits; the don't-care digits are ignored. For example, to request a break on any opcode address in the range of addresses from 0D000H to 0DFFFH inclusive, any of the three following examples will serve:

```
*GO TILL 0DXXXH                 ;"H" means "hexadecimal".
*GO TILL 15XXXXQ                ;"Q" means "octal".
*GO TILL 1101XXXXXXXXXXXXY      ;"Y" means "binary".
```

In a binary mask, each X digit represents one don't-care bit. In an octal mask, each X digit represents three don't-care bits. In a hexadecimal mask, each X represents four don't-care bits. If the don't-care digits in a mask are the low-order digits (as in the examples above), the mask represents a range of contiguous addresses. If the don't-care digits are not the low-order digits, the mask represents several single addresses distributed at intervals; for example, the mask 01X0H represents addresses 0100H, 0110H, 0120H, and so on up through 10F0H.

Since the mask in this form of breakpoint represents an address, the system assumes 16 bits; if the mask you enter specifies fewer than 16 bits the value is right-justified (placed in the low-order bits) and the leading bits not specified in the mask are filled with *zeros* (not don't-cares). For example, 1XH becomes 001XH.

You can use a mask instead of a single address in any of the forms of the GO command. Thus, the overall format of the GO command discussed in this chapter is as follows:

```
                      ┌                                              ┐
                      │ FOREVER                                      │
GO [FROM address]     │                                              │
                      │ TILL   {address}   [OR {address}]            │
                      └        {mask   }   [   {mask   }]            ┘
```

Here are some examples with combinations of addresses and masks:

```
*GO FROM .START TILL .DONE OR 2XXH
*GO TILL 1XXXY or 400H
```

Single-Step Emulation

Under single-step emulation the emulation processor executes the instruction whose address is in the program counter. Emulation breaks automatically after that instruction is completed, and the system returns control to the console. Trace data is always collected during single-step emulation.

Single-step is useful for exercising software logic in detail.

To begin single-step emulation enter the command STEP then press RETURN. At this command, the following occurs:

- The message EMULATION BEGUN is displayed.
- The instruction defined by PC is emulated.
- Emulation breaks automatically after that instruction is completed.
- The message EMULATION TERMINATED, PC=*address* is displayed just as for real-time emulation.

Thus, each time you enter the STEP command, one instruction is emulated.

The full format is:

```
STEP [FROM address]
```

The FROM clause controls the address of the instruction to be executed.

Single-step emulation differs from real-time emulation in several ways. With respect to the emulator system, trace information is always collected during single step; the emulation timer is inoperative and is cleared; the external sync line inputs are ignored. Signal SY0 OUT, if enabled, goes momentarily high during each single step. The on-chip timers, if running, count *two* times during each single step.

NOTE

To have several steps emulated as a unit, the STEP command can be entered within a COUNT or REPEAT command block. Refer to chapter 7 for the details on block commands.

CHAPTER 4
TRACE COMMANDS

The Trace Buffer

The emulator system maintains a trace buffer to record information collected during emulation. The buffer contains up to 1000 frames of information; each instruction cycle takes four frames. Thus, depending on the mix of instructions, 1000 frames represents about 100 to 200 instructions. A technique for increasing the maximum number of instructions to 1000 is presented later in this chapter.

The trace buffer is initially empty. The buffer is cleared whenever the program counter is changed by a PC command, by a FROM clause in a GO or STEP command, or by a LOAD command. Emulation can break and resume without clearing trace if the program counter is not changed between emulations.

Frames are numbered from 000 to 999 decimal. New frames are added at the end of the buffer, using successively higher frame numbers. After the first 1000 frames, the buffer "overflows"; each new frame added at the end (in frame 999) causes the oldest frame to be lost from the beginning of the buffer. Thus after overflow the buffer retains the most recent 1000 frames.

Displaying Trace Information

There are two modes of trace display, called INSTRUCTION mode and FRAME mode. INSTRUCTION mode is the initial mode. In INSTRUCTION mode each line of display contains the information collected during the execution of one complete instruction (several frames). In FRAME mode each display line contains the information for just one frame. INSTRUCTION mode displays are explained in the present chapter. FRAME mode displays are discussed in chapter 6; chapter 6 contains details on trace collection that may be required to interpret FRAME mode displays.

Display of information from the buffer is controlled by a "line pointer"; the pointer indicates the place in the buffer where the display will begin or end. The pointer always points at the beginning of a line. After emulation breaks, the pointer is just past the frame most recently collected.

OLDEST and NEWEST Commands

To move the display pointer to the first frame in the buffer, use the command:

```
*OLDEST
```

To move the pointer to just after the last frame in the buffer, use the command:

```
*NEWEST
```

MOVE Command

The MOVE command moves the pointer forward toward NEWEST or backward toward OLDEST. The format of this command is:

MOVE [[-] *number-of-lines*]

Examples:

```
*MOVE                           ;Same as "MOVE 1"
*MOVE 10
*MOVE -25
```

The entry *number-of-lines* can be a decimal constant, that is a decimal number without an explicit radix (example: MOVE 5). It can also be a symbol or keyword reference. Any numeric constant entered after MOVE without an explicit radix is assumed to be decimal; if the number contains digits that are invalid in decimal radix, an error occurs.

When *number-of-lines* is positive (for example, MOVE 5), movement is toward NEWEST. The pointer cannot move past NEWEST; if the pointer is already at NEWEST or if *number-of-lines* would move the pointer past NEWEST, the pointer ends up just past the last line in the buffer.

When *number-of-lines* is negative (example: MOVE -5), the pointer moves toward OLDEST. The pointer cannot move back past OLDEST; if the pointer is already at OLDEST or if the *number-of-lines* would move the pointer back past OLDEST, the pointer ends up at the first line in the buffer.

PRINT Command

The PRINT command displays one or more lines from the trace buffer, with a header to identify the information. The PRINT command does not move the line pointer. Details on the header and information appear later in this chapter. The format of the command is:

$$\text{PRINT} \begin{bmatrix} \text{[-] } \textit{number-of-lines} \\ \text{ALL} \end{bmatrix}$$

Examples:

```
*PRINT                          ;Same as "PRINT 1"
*PRINT 10
*PRINT -25
*P ALL
```

If the pointer is at NEWEST (end of the buffer), the PRINT command displays the header only; no lines are displayed.

The entry *number-of-lines* is as defined earlier for the MOVE command.

When *number-of-lines* is positive (no sign, for example, PRINT 5) display begins with the line at the current pointer. If the number of lines exceeds the distance to NEWEST, only the lines from the pointer to NEWEST are displayed; in other words, you cannot display any lines past NEWEST.

When *number-of-lines* is negative (for example, PRINT -5), the display contains that many lines up to and including the line right before the pointer (i.e., PRINT -5 is equivalent to MOVE -5, PRINT 5, MOVE 5). If the number of lines requested exceeds the distance to OLDEST, only the number of lines from OLDEST to the current pointer are displayed.

The command PRINT ALL displays all lines in the buffer from OLDEST to NEWEST.

Displays in INSTRUCTION Mode

In INSTRUCTION mode each line of the display contains information collected during the execution of one complete instruction. Figure 4-1 shows an example of an INSTRUCTION mode display. The display begins with a header to identify each type of information. The next several paragraphs briefly explain the items as they appear in the figure from left to right.

FRAME. The frame number of the first frame in each instruction from 000 to 999 decimal.

LOC. The address of the opcode (first byte) of the instruction in hexadecimal. When symbolic display is enabled, a symbol that exactly matches an address appears on the line above that address (frame number 008 in figure 4-1).

OBJ. Opcode and operand bytes in hexadecimal, like the OBJ field in an assembly listing.

INSTRUCTION. The assembler mnemonics for the opcode and operands as produced by the "disassembler." The disassembly includes symbols for addresses in the operand fields. The format is identical to that produced by the DASM command (see chapter 5). The display of the disassembled instruction may require more than one display line.

P1, P2, P0. The values of the three I/O ports in hexadecimal. In INSTRUCTION mode, the values displayed are the ones from the first frame in the instruction. The arrangement of the ports in the display allows you to read P2 and P0 as the high and low bytes (respectively) of an external address.

TOVF. Trace overflow flag (0 = no overflow, 1 = overflow). The trace buffer overflows after the first 1000 frames have been collected. In INSTRUCTION mode, the value of TOVF is the one from the first frame of the instruction.

```
FRAME   LOC     OBJ      INSTRUCTION                 P1    P2    P0    TOVF
0000:   0000H   802E     SJMP   .START               FFH   FFH   FFH    0
.START
0008:   0030H   439002   ORL    .P1,#02H             FFH   FFH   FFH    0
0016:   0033H   7830     MOV    R0,#.START           FFH   FFH   FFH    0
0020:   0035H   7A00     MOV    R2,#00H              FFH   FFH   FFH    0
0024:   0037H   C220     CLR    .Y                   FFH   FFH   FFH    0
0028:   0039H   D222     SETB   .Z                   FFH   FFH   FFH    0
0032:   003BH   758920   MOV    .TMOD,#.Y            FFH   FFH   FFH    0
0040:   003EH   75A888   MOV    .IE,#.CTABL          FFH   FFH   FFH    0
0048:   0041H   758DF0   MOV    .TH1,#.COUNT         FFH   FFH   FFH    0
0056:   0044H   758800   MOV    .CTABL,#00H          FFH   FFH   FFH    0
0064:   0047H   758BF0   MOV    .TL1,#.COUNT         FFH   FFH   FFH    0
0072:   004AH   D28E     SETB   .TR1                 FFH   FFH   FFH    0
0076:   004CH   8002     SJMP   .LOOP                FFH   FFH   FFH    0
.LOOP
0084:   0050H   A2D0     MOV    C,.V                 FFH   FFH   FFH    0
0088:   0052H   728F     ORL    C,.W                 FFH   FFH   FFH    0
0096:   0054H   8291     ANL    C,.U                 FFH   FFH   FFH    0
0104:   0056H   92D5     MOV    .F0,C                FFH   FFH   FFH    0
0112:   0058H   A2AB     MOV    C,.X                 FFH   FFH   FFH    0
0116:   005AH   B020     ANL    C,/.Y                FFH   FFH   FFH    0
0124:   005CH   72D5     ORL    C,.F0                FFH   FFH   FFH    0
0132:   005EH   A022     ORL    C,/.Z                FFH   FFH   FFH    0
0140:   0060H   92A3     MOV    .Q,C                 FFH   FFH   FFH    0
```

Figure 4-1. INSTRUCTION Mode Trace Display

Reconstruction and Verification

The intention in INSTRUCTION mode is to facilitate debug at the assembler level by providing a display that includes the OBJ values and disassembly for a complete instruction on every line. However, some of the ways to collect trace do not collect entire instructions. The system can use the contents of program memory to fill in "gaps" in trace, as described below.

In INSTRUCTION mode the system only displays an instruction if the first frame of the instruction has been collected in trace. The first frame of each instruction contains the address of the opcode; this type of frame is called a LOCATION frame. The address in the LOCATION frame is the one in the LOC field in ASM-51 listings and INSTRUCTION mode displays. All of the trace qualifier commands in this chapter collect the LOCATION frame of any instruction they qualify. (Refer to chapter 6 for more details on frames and frame types.)

The system reads the opcode address, then checks to see if the opcode value is present in trace. If it is, the one-byte opcode value is read from trace. If it is not, the byte value is read from the corresponding address in code memory instead. The system decodes the opcode to determine the number of bytes in the instruction, then checks to see if all the bytes have been collected in trace. If the entire instruction is in trace, the displays in the OBJ field and the disassembled instruction are based on the values from trace. If any byte is missing, the corresponding display is read from memory instead. Once the values of all bytes have been obtained, display of the OBJ field and the disassembled instruction proceeds as described above.

If a value is present in trace it is compared to the corresponding value in code memory. If the two values are different, a warning message UNEXPECTED TRACE is displayed. If the two values are the same, no message is displayed. Trace and memory can differ if the user changes code memory after emulation breaks; program memory does not usually change during emulation.

One warning message is displayed for each frame found to be different. The message or messages appear on the line(s) immediately preceding the instruction that produced the warning frames. (Refer to "Frame Mode Trace Displays" in chapter 6 for more information on this warning message.)

NOTE

Normally when trace and memory are found to differ, the value from trace is the one displayed in INSTRUCTION mode. However, when an opcode or value from trace equals FFH and the corresponding memory value is not FFH, the trace is assumed to be in error and the memory value is the one displayed.

Turning Trace On and Off

Trace information is always collected during single-step. The following trace controls thus apply only to real-time emulation.

Initially, no factors are enabled to control trace; every frame of every instruction is collected in the buffer, and the buffer retains the most recent 1000 frames. To restore this condition, the command is:

```
TR = FOREVER
```

To have trace turned off after collecting the first-frame of a particular instruction, the format is:

$$\text{TR = TILL} \begin{Bmatrix} \textit{address} \\ \textit{mask} \end{Bmatrix}$$

Examples:

```
*TR = TILL 0100H          ;Halt trace after opcode address 0100.
*TR = TILL .DONE          ;Halt trace after opcode address given by
                          label .DONE.
*TR = TILL 01XXH          ;Halt after any opcode address, 0100H
                          through 01FFH.
```

In these forms the *address* must be the first (opcode) address in an instruction to halt trace collection. Trace halts after collecting the first frame of the instruction; the buffer retains the 1000 frames up to and including that instruction.

You can use a *mask* to specify a range of addresses. A mask is a hexadecimal, octal, or binary numeric constant in which some of the digits are "don't-cares"; a don't-care digit is specified with the character X. Refer to chapter 3 for more details on masks.

You can specify two different addresses (or masks) to halt trace; the format is:

$$\text{TR = TILL} \left\{ \begin{array}{l} \textit{address} \\ \textit{mask} \end{array} \right\} \quad \text{OR} \quad \left\{ \begin{array}{l} \textit{address} \\ \textit{mask} \end{array} \right\}$$

Examples:

```
*TR = TILL .DONE OR 0100H
*TR = TILL 01XXH OR .START + 5
```

In these examples, trace collection halts if either of the two opcode addresses is executed. Addresses and masks can be combined in the same command, as shown in the second example above.

To turn trace on for one bufferful after the first frame of a particular instruction, the format is:

$$\text{TR = AFTER} \left\{ \begin{array}{l} \textit{address} \\ \textit{mask} \end{array} \right\} \left[\text{OR} \left\{ \begin{array}{l} \textit{address} \\ \textit{mask} \end{array} \right\} \right]$$

Examples:

```
*TR = AFTER 0100H
*TR = AFTER XXXXH
*TR = AFTER .START OR .TIMER0
```

With this form, trace collection does not start until the specified opcode address is executed. Then, all frames of trace are collected until the buffer overflows. The command allows you to specify one or two addresses, and to use a mask for either or both addresses. The second example, "AFTER XXXXH", collects one bufferful of information from the beginning of emulation (the first frame of the first instruction is not collected). If the trace buffer is cleared before starting emulation, 1000 frames are collected after trace turns on; if the buffer has not been cleared, only the frames from NEWEST to 1000 are collected.

To maximize the number of instructions collected in the buffer (and displayable in INSTRUCTION mode), the command is:

```
*TR = VALUE IS XXH
```

For reasons explained in chapter 6, this command collects the first frame of every instruction (along with some additional types of frames). Depending on the mix of instructions, this control allows you to capture the first frames of up to 1000 instructions. This information allows the instruction to be displayed in INSTRUCTION mode, but data on frames other than the first may be lost.

The trace controls described in this chapter are useful for controlling trace when the display mode is in INSTRUCTION. Refer to chapter 6 for further trace controls.

CHAPTER 5
MEMORY CONTENTS COMMANDS

Introduction

The commands in this chapter allow you to access the various memory spaces available to the user. Access to memory serves the following purposes:

- Displaying the contents of one or more locations.
- Changing the contents of one or more locations.
- Assembling short programs or program patches.
- Using addresses and contents as operands in assembler instructions.
- Using addresses and contents as operands in emulator commands.

Types of Memory Space

The emulator provides access to five kinds of user memory spaces, summarized in table 5-1. Each type of memory is referenced by an *address* of 8 or 16 bits, and contains a byte or bit contents, as shown in the table. The table also shows the *content operator* appropriate to each kind of memory; the content operators are one way to access the contents of memory in emulator commands.

Table 5-1. Types of User Memory

Type of Memory	Maximum Bits in Address	Maximum Bits in Contents	Content Operator
On-chip data memory	8	8	DBYTE
Register memory	8	8	RBYTE
Bit-addressable memory	8	1	RBIT
External data memory	16	8	XBYTE, PBYTE
Code memory	16	8	CBYTE

Using Content Operators

Commands to display and set memory contents with the content operators (DBYTE, RBYTE, RBIT, XBYTE, PBYTE, and CBYTE) are similar in pattern. The format for displaying memory content is:

content-operator partition

Examples:

```
*DBYTE 0 TO FH
*RBYTE .SP
*RBIT .PSW+7
*PBYTE 0100H LENGTH 30H
*CBYTE 0 TO 1K
```

The format for setting memory contents is:

$$\textit{content-op partition} = \left\{\begin{array}{l}\textit{expression}\\ \textit{content-op partition}\\ \textit{string}\end{array}\right\}\left[\,,\left\{\begin{array}{l}\textit{expression}\\ \textit{content-op partition}\\ \textit{string}\end{array}\right\}\right]\ldots$$

Examples:

```
*DBYTE 30H = 0
*RBYTE .SP = (RBYTE .SP) + 1
*RBIT .PSW+7 = 1
*XBYTE 0100H TO 0120H = DBYTE 30H, 'A', FOH
*CBYTE 50H = DBYTE 30H LENGTH 37T
```

More examples of these kinds of commands appear in the following sections. These additional examples illustrate operations that apply to each type of memory.

NOTE

The contents shown in the examples in this chapter may be 'random' in that they do not represent any particular program or data. The result you obtain by entering the command examples may produce different values in the display.

On-Chip Data Memory Access

Data Memory Addresses

Figure 5-1 diagrams the on-chip data memory. Addresses in this memory are 8-bit quantities, but only the addresses in the range from 00H to 7FH (0 to 127 decimal) are valid. (Addresses above 7FH are in register memory, discussed later in this chapter.) There are no system symbols for this memory area. The low 32 bytes of data memory contain the four register banks; use of the register banks is discussed later in this section. As shown in the figure, the stack is initially positioned to begin at location 08H (i.e., the stack pointer is initialized at 07H). Displays of the stack are described later in this section. The bits in data locations 20H through 2FH can be individually addressed; bit addressing is discussed in a later section of this chapter.

Contents of Data Memory

The content operator DBYTE is used to display and change the contents of data memory. All addresses after DBYTE must be in the range 00 to 7FH; addresses above this range produce an error message and no memory is displayed or changed.

To display the contents of a single address, the format is:

DBYTE *address*

Example:

```
*DBYTE 0
DBYTE 00H=00H
```

To display the contents of several contiguous addresses, the format is:

DBYTE *partition*

Example:

```
*DBYTE 20H LENGTH 10H
0020H=20H 9CH 92H 9AH F6H 99H B0H 7BH 08H 4CH CDH 27H ECH 8AH 07H D8H
```

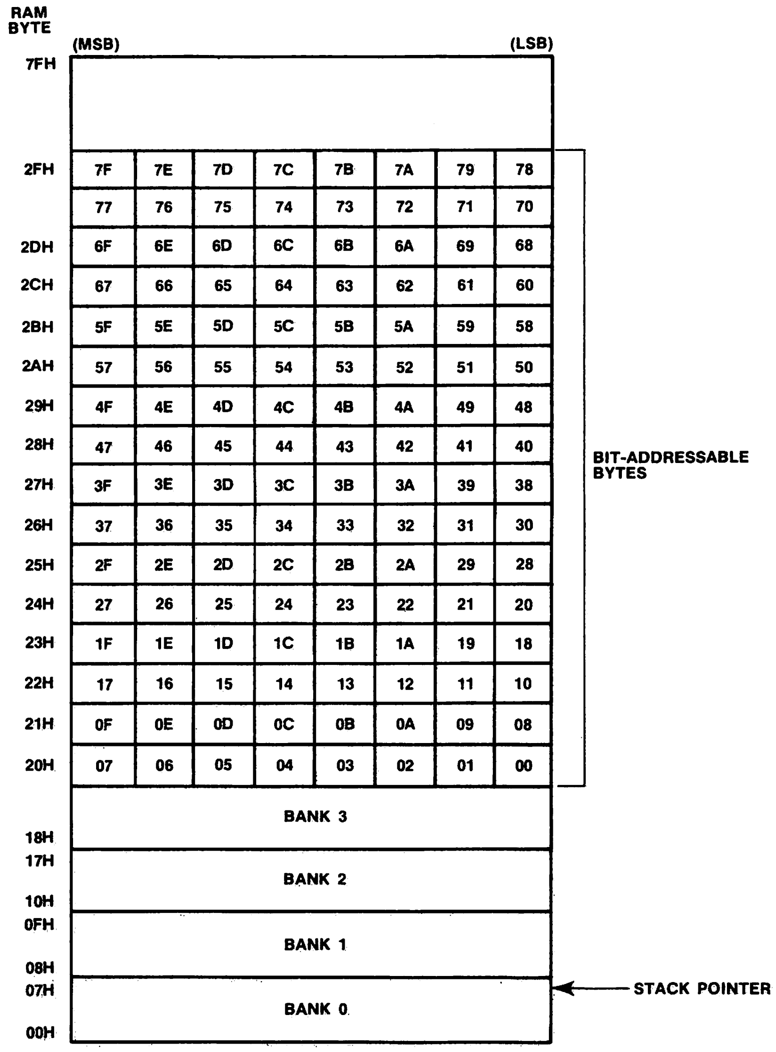

RAM BYTE
(MSB)
(LSB)
7FH
2FH 7F 7E 7D 7C 7B 7A 79 78
2EH 77 76 75 74 73 72 71 70
2DH 6F 6E 6D 6C 6B 6A 69 68
2CH 67 66 65 64 63 62 61 60
2BH 5F 5E 5D 5C 5B 5A 59 58
2AH 57 56 55 54 53 52 51 50
29H 4F 4E 4D 4C 4B 4A 49 48
28H 47 46 45 44 43 42 41 40
27H 3F 3E 3D 3C 3B 3A 39 38
26H 37 36 35 34 33 32 31 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
BIT-ADDRESSABLE BYTES
18H
BANK 3
17H
10H
BANK 2
0FH
08H
BANK 1
07H
00H
BANK 0
STACK POINTER

Figure 5-1. Data Memory

The display has a maximum of 16 bytes per line. The address of the first byte in the line is displayed at the beginning of each line.

To change the contents of a single data address, the format is:

DBYTE *address* = *expression*

Example:

```
*DBYTE 30H = 0                ;See chapter 1 for the forms of expression.
```

To change all bytes in a partition of data memory to a single value, the format is:

DBYTE *partition* = *expression*

Examples:

```
*DBYTE 0 to 7FH = 0           ;Clears data memory
```

To load a string of ASCII characters into data memory, the format is:

DBYTE *address* = *string*

Example:

```
*DBYTE 50H = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
*DBYTE 50H LENGTH 26T
0050H=41H 42H 43H 44H 45H 46H 47H 48H 49H 4AH 4BH 4CH 4DH 4EH 4FH 50H
0060H=51H 52H 53H 54H 55H 56H 57H 58H 59H 5AH
```

The address after DBYTE specifies where the first character in the string is to be stored; successive bytes receive the subsequent characters. Thus, the length of the string determines the number of bytes. An error occurs if the top of data memory is reached with characters yet to be stored.

To copy bytes from one area of memory to data memory, the format is:

DBYTE *address* = *content-operator partition*

For example, to copy a table from code to data memory:

```
*DBYTE 50H = CBYTE .TABLE LENGTH 32T
```

Another way to set a range of data memory is to use a list of new values; the format for this kind of command is:

$$\text{DBYTE } \textit{address} = \left\{\begin{array}{l}\textit{expression}\\ \textit{string}\\ \textit{content-op partition}\end{array}\right\}\left[\,,\left\{\begin{array}{l}\textit{expression}\\ \textit{string}\\ \textit{content-op partition}\end{array}\right\}\right]\ldots$$

Example:

```
*DBYTE 60H = 10H, RBYTE .PSW, 'HELLO', CBYTE .TABLE LENGTH 30T
```

The number of data bytes changed is determined by the number of bytes in the list (37 bytes in the example above).

The above form can be generalized to the following:

DBYTE *partition* = *list-of-new-values*

Where *list-of-new-values* is any number of the entries *expression, string, or content-operator partition* separated by commas.

For example, to fill a block of data memory with a repeating sequence of values:

```
*DBYTE 60H LENGTH 10H = 11H, 22H, 33H
*DBYTE 60H LENGTH 10H

0060H=11H 22H 33H 11H 22H 33H 11H 22H 33H 11H 22H 33H 11H 22H 33H 11H
```

In this form, the number of bytes in the *partition* after DBYTE determines the number of data bytes changed. If the number of bytes in the list of values is less than the length of the partition, the values are repeated as in the previous example. If the number of values exceeds the length of the partition, data bytes will be set as described earlier until the first excess value is reached; at this point a warning message is displayed.

Contents of Register Banks

The emulator system provides the keywords R0, R1, R2, R3, R4, R5, R6, and R7 to refer to the contents of the 8-bit working registers in data memory. The register bank is the one currently selected. For example, to display the contents of register 0 in the current bank, the command is:

```
*R0
R0=00H
```

To change the content of a working register, the format is:

```
Rn = expression                    ;Where n = 0 through 7.
```

Example:

```
*R5 = 21H
```

The emulator truncates the expression to its least significant byte, just as the assembler does.

To display the current register bank, enter the keyword RBS (Register Bank Select):

```
*RBS
RBS=00H
```

The value will be 00H, 01H, 02H, or 03H

To select another register bank, use a command with the format:

```
RBS = expression
```

The emulator requires the expression to contain one or two bits. Most of the time the expression will be a single digit, 0 through 3, for example:

```
RBS = 2
```

The Register Bank Select bits are part of the Program Status Word, a location in register memory. Refer to the section on "Register Memory Access" later in this chapter for more details.

STACK Display Command

The STACK command has the following format:

```
STACK [expression]
```

The STACK command displays the stack from the top in bytes, and the contents of the stack pointer. The *expression* is the number of bytes (entries) you wish to display. The stack pointer is not affected by the STACK command.

The command STACK without *expression* displays the stack from the top down to data location 00H. (In other words, you have to keep track of the initial stack pointer location.) The display also stops at location 00H when the *expression* is larger than (SP + 1).

Here are some examples:

```
*STACK              *STACK 10T
SP STACK            SP STACK
0CH  00H            0CH  00H
0BH  00H            0BH  00H
0AH  00H            0AH  00H
09H  00H            09H  00H
08H  00H            08H  00H
07H  00H            07H  00H
06H  00H            06H  00H
05H  00H            05H  00H
04H  00H            04H  00H
03H  00H            03H  00H
02H  00H
01H  00H
00H  00H
```

The stack pointer is a byte register in register memory. To display the contents of the stack pointer directly, use the command:

```
RBYTE .SP
```

Example:

```
*RBYTE .SP
RBYTE .SP=07H                    ;Initial default.
```

More information on register memory access follows in the next section.

Register Memory Access

Register Addresses

Register addresses are 8-bit quantities. In the register memory (addresses 80H through FFH), only certain addresses are valid. Each of the valid addresses in register memory has a predefined system symbol to represent it; table 5-2 shows the valid register addresses (in hexadecimal) and the corresponding system symbols.

If you try to read any of the invalid register addresses (above 80H but not in table 5-2), a byte of undefined data is returned. If you try to write to any invalid register address, the data is lost.

Table 5-2. Addresses and System Symbols for Registers

Register Address (Hex)	System Symbol	Meaning
80H	.P0	Port 0
81H	.SP	Stack Pointer
82H	.DPL	Data Pointer, Low Byte
83H	.DPH	Data Pointer, High Byte
88H	.TCON	Timer Control
89H	.TMOD	Timer Mode
8AH	.TL0	Timer 0, Low Byte
8BH	.TL1	Timer 1, Low Byte
8CH	.TH0	Timer 0, High Byte
8DH	.TH1	Timer 1, High Byte
90H	.P1	Port 1
98H	.SCON	Serial Port Control
99H	.SBUF	Serial Port Buffer
A0H	.P2	Port 2
A8H	.IE	Interrupt Enable
B0H	.P3	Port 3
B8H	.IP	Interrupt Priority
D0H	.PSW	Program Status Word
E0H	.ACC	Accumulator
F0H	.B	Multiplication Register

Contents of Register Memory

The content-operator RBYTE is used to display and change the contents of register memory. To display the contents of any register, the format is:

RBYTE *register-address*

Examples:

```
*RBYTE .PSW
RBYTE 00D0H=00H
*RBYTE .P1
RBYTE 0090H=FFH
*RBYTE 80 TO FF
0080H=FFH 07H 00H 00H FFH FFH FFH FFH 00H 00H 00H 00H 00H 00H 00H 00H
0090H=FFH FFH FFH FFH FFH FFH FFH FFH 00H 00H FFH FFH FFH FFH FEH FFH
00A0H=FFH FFH FFH FFH FFH FFH FFH FFH 00H FFH FFH FFH FFH FFH FFH FFH
00B0H=FFH FFH FFH FFH FFH FFH FFH FFH E0H FFH FFH FFH FFH FFH FFH FFH
00C0H=FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH
00D0H=00H FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH
00E0H=00H FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH
00F0H=00H FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH
```

To change the contents of any register, the format is:

RBYTE *register-address = expression*

The expression is truncated to its low byte.

Examples:

```
*RBYTE .PSW = F0H
*RBYTE .PSW
RBYTE 00D0H=F0H
*RBYTE .ACC = F7F7H
*RBY .ACC
RBYTE 00E0H=F7H
```

Reserved Keywords for Register Memory Contents

The following keywords refer to the contents of special parts of register memory:

DPTR	Data Pointer (16 bits).
TM0	Timer 0 (16 bits)
TM1	Timer 1 (16 bits)
RBS	Register Bank Select (2 bits)

These references produce the contents of memory directly; the RBYTE operator is not required. The keyword alone is a display command. For example:

```
*DPTR
DPTR=0000H
```

You can change the contents of these registers; the format is:

keyword-reference = expression

For example:

```
*DPTR = .TABLE + 5
*TM0 = FFF0H
*RBS = 1
```

REGISTERS Command

The command REGISTERS produces a display of the contents of the following registers:

PC	Program Counter
ACC	Accumulator
B	Multiplication Register
SP	Stack Pointer
DPTR	Data Pointer
R0	Working Register 0
R1	Working Register 1
PSW	Program Status Word

The PSW register is displayed in binary radix; all the others are displayed in hexadecimal. For example:

```
*REGISTERS
PC     ACC  B    SP   DPTR   R0   R1   PSW
0000H  00H  A1H  07H  0000H  00H  01H  00000000Y
```

The value of R0 and R1 reflect the current bank selected.

INTERRUPT Display Command

The INTERRUPT command displays the Interrupt Enable (IE) and Interrupt Priority (IP) registers, and the status of Interrupts In Progress for priority 0 (IIP0) and priority 1 (IIP1) in tabular format. For example:

```
*INTERRUPT
        EA       SINT     TIMER1     EXTI1     TIMER0     EXTI0
IIP0             0        0          0         0          0
IIP1             0        0          0         0          0
IE      1        0        0          0         0          1
IP               0        0          0         0          1
```

For IIP0 and IIP1, "1" means "in progress"; for IE, "1" means "enabled"; for IP, "0" and "1" are the priority levels assigned.

The column headers are:

EA	Enable All Interrupts (IE register only)
SINT	Serial Port Interrupt
TIMER1	Timer 1 Interrupt
EXTI1	External 1 Interrupt
TIMER0	Timer 0 Interrupt
EXTI0	External 0 Interrupt

Bit-Addressable Memories

Bit Addresses

A bit address is an 8-bit quantity that indicates either a bit in the data RAM bit-addressable space or a bit in a bit-addressable register in register memory. Bit addresses in the range from 0 to 7FH refer to the 128 bits in data bytes 20H through 2FH. Bit addresses between 80H and FFH refer to bits in bit-addressable registers. The bit-addressable registers are those whose register address has either a zero or an eight as the second hex digit (e.g., 90H, 98H). Many of the bit addresses in register memory can be represented by system symbols; all of them can be represented by an entry with the form

.bit-addressable-register + bit-position

For example, bit 7 of the Program Status Word would be addressed as:

.PSW+7

This form is the emulator language equivalent of the ASM-51 "bit-operator"; the PSW example would be "PSW.7" in ASM-51.

Table 5-3 shows all the bit addresses in register memory that are valid. The table shows the system symbols when applicable, as well as the "bit-operator" form of address. Any bit address in the range from 80H to FFH that is not in this table is invalid; bits read from invalid bit addresses are undefined, and bits written to invalid addresses are lost.

Table 5-3. Bit Addresses in Register Memory

Hex Address	System Symbol	Register + Bit Position	Meaning
80H		.P0+0	Port 0, Bit 0
81H		.P0+1	Port 0, Bit 1
82H		.P0+2	Port 0, Bit 2
83H		.P0+3	Port 0, Bit 3
84H		.P0+4	Port 0, Bit 4
85H		.P0+5	Port 0, Bit 5
86H		.P0+6	Port 0, Bit 6
87H		.P0+7	Port 0, Bit 7
88H	.IT0	.TCON+0	Timer 0 Interrupt, Type Control Bit
89H	.IE0	.TCON+1	Timer 0 Interrupt, Edge Flag
8AH	.IT1	.TCON+2	Timer 1 Interrupt, Type Control Bit
8BH	.IE1	.TCON+3	Timer 1 Interrupt, Edge Flag
8CH	.TR0	.TCON+4	Timer 0 Run Control Bit
8DH	.TF0	.TCON+5	Timer 0 Overflow Flag
8EH	.TR1	.TCON+6	Timer 1 Run Control Bit
8FH	.TF1	.TCON+7	Timer 1 Overflow Flag
90H		.P1+0	Port 1, Bit 0
91H		.P1+1	Port 1, Bit 1
92H		.P1+2	Port 1, Bit 2
93H		.P1+3	Port 1, Bit 3
94H		.P1+4	Port 1, Bit 4
95H		.P1+5	Port 1, Bit 5
96H		.P1+6	Port 1, Bit 6
97H		.P1+7	Port 1, Bit 7
98H	.RI	.SCON+0	Receive Interrupt Flag
99H	.TI	.SCON+1	Transmit Interrupt Flag
9AH	.RB8	.SCON+2	Receive Bit 8
9BH	.TB8	.SCON+3	Transmit Bit 8
9CH	.REN	.SCON+4	Receiver Enable
9DH	.SM2	.SCON+5	Serial Mode Control Bit 2
9EH	.SM1	.SCON+6	Serial Mode Control Bit 1
9FH	.SM0	.SCON+7	Serial Mode Control Bit 0

Table 5-3. Bit Addresses In Register Memory (Continued)

Hex Address	System Symbol	Register + Bit Position	Meaning
A0H		.P2+0	Port 2, Bit 0
A1H		.P2+1	Port 2, Bit 1
A2H		.P2+2	Port 2, Bit 2
A3H		.P2+3	Port 2, Bit 3
A4H		.P2+4	Port 2, Bit 4
A5H		.P2+5	Port 2, Bit 5
A6H		.P2+6	Port 2, Bit 6
A7H		.P2+7	Port 2, Bit 7
A8H	.EX0	.IE+0	Enable External Interrupt 0
A9H	.ET0	.IE+1	Enable Timer 0 Interrupt
AAH	.EX1	.IE+2	Enable External Interrupt 1
ABH	.ET1	.IE+3	Enable Timer 1 Interrupt
ACH	.ES	.IE+4	Enable Serial Port Interrupt
AFH	.EA	.IE+7	Enable All Interrupts
B0H	.RXD	.P3+0	Serial Port Receive Pin
B1H	.TXD	.P3+1	Serial Port Transmit Pin
B2H	.INT0	.P3+2	Interrupt 0 Input Pin
B3H	.INT1	.P3+3	Interrupt 1 Input Pin
B4H	.T0	.P3+4	Timer/Counter 0 External Flag
B5H	.T1	.P3+5	Timer/Counter 1 External Flag
B6H	.WR	.P3+6	Write Data (For External Memory)
B7H	.RD	.P3+7	Read Data (For External Memory)
B8H	.PX0	.IP+0	Priority of External Interrupt 0
B9H	.PT0	.IP+1	Priority of Timer 0 Interrupt
BAH	.PX1	.IP+2	Priority of External Interrupt 1
BBH	.PT1	.IP+3	Priority of Timer 1 Interrupt
BCH	.PS	.IP+4	Priority of Serial Interrupt
D0H	.P	.PSW+0	Parity Flag
D2H	.OV	.PSW+2	Overflow Flag
D3H	.RS0	.PSW+3	Register Bank Select Bit 0
D4H	.RS1	.PSW+4	Register Bank Select Bit 1
D5H	.F0	.PSW+5	Flag 0
D6H	.AC	.PSW+6	Auxiliary Carry Flag
D7H	.CY	.PSW+7	Carry Flag
E0H		.ACC+0	Accumulator, Bit 0
E1H		.ACC+1	Accumulator, Bit 1
E2H		.ACC+2	Accumulator, Bit 2
E3H		.ACC+3	Accumulator, Bit 3
E4H		.ACC+4	Accumulator, Bit 4
E5H		.ACC+5	Accumulator, Bit 5
E6H		.ACC+6	Accumulator, Bit 6
E7H		.ACC+7	Accumulator, Bit 7
F0H		.B+0	Multiplication Register, Bit 0
F1H		.B+1	Multiplication Register, Bit 1
F2H		.B+2	Multiplication Register, Bit 2
F3H		.B+3	Multiplication Register, Bit 3
F4H		.B+4	Multiplication Register, Bit 4
F5H		.B+5	Multiplication Register, Bit 5
F6H		.B+6	Multiplication Register, Bit 6
F7H		.B+7	Multiplication Register, Bit 7

Contents of Bit-Addressable Memory

The keyword RBIT is used to display and change the contents of bit-addressable memory. To display the contents of any bit address, the format is:

RBIT *bit-address*

Examples:

```
*RBIT .PSW
RBIT 00D0H=08H
```

To display a range of bit-addresses, the format is:

RBIT *partition*

This format applies particularly to the bit addresses in data memory (00H through 7FH), since the other bit addresses are somewhat discontinuous.

For example:

```
*RBIT 0 TO 7FH
```

To change the value of a bit in bit-addressable memory, the format is:

$$\textit{RBIT bit-address} = \left\{\begin{array}{l}\text{RBIT } \textit{bit-address}\\ \textit{boolean-expression}\end{array}\right\}$$

The term *boolean-expression* means the same kinds of entries as *expression* (see chapter 1). A boolean-expression is truncated to its least significant bit.

Examples:

```
*RBIT .PSW = 0
*RBIT .CY = RBIT .TF1
```

The emulator language allows several other forms with RBIT, including the use of strings and other byte quantities (truncated to the lowest bit), and allowing a partition of bit addresses to be set to a single value, or to a list of values just as with DBYTE. The general form of such commands is:

$$\text{RBIT } \textit{partition} = \left\{\begin{array}{l}\textit{boolean-expression}\\ \textit{string}\\ \textit{content-op partition}\end{array}\right\} \left[, \left\{\begin{array}{l}\textit{boolean-expression}\\ \textit{string}\\ \textit{content-op partition}\end{array}\right\} \right] \ldots$$

The form takes the least significant bit of each value.

Access To External Data Memory

Valid addresses in external data memory depend on what the user has installed in the prototype system. The emulator treats all external data addresses as 16-bit values.

The content-operators XBYTE and PBYTE are used to display and change the contents of external data memory. To display one or more locations:

$$\left\{\begin{array}{l}\text{XBYTE}\\ \text{PBYTE}\end{array}\right\} \textit{partition}$$

Examples:

```
*XBYTE 0
XBYTE 0000H=93H
*PBYTE 0 TO 5
0000H=93H A5H 52H 17H 54H 52H
```

To change the contents of one or more addresses in external data memory, the format is:

```
{XBYTE}             {expression          } [ {expression          } ]
{PBYTE} partition = {string              } [,{string              } ] ...
                    {content-op partition} [ {content-op partition} ]
```

Examples:

```
*XBYTE 0100 TO 0200 = 'X'
*PBYTE 07FF = DBYTE 30, 'A', F0H
```

When you display or change the contents of an external data location, the emulator reads or writes the external memory via P2 and P0 just as if a MOVX instruction had been executed.

With XBYTE, the system performs read-after-write verification. With PBYTE, no verification is performed.

Code Memory Access

Code Addresses

Code addresses are 16-bit quantities, representing addresses in the range from 0 through FFFFH (65,535 or 64K - 1). System symbols for (interrupt) locations in code memory are listed in table 5-4.

The emulator provides two 4K RAM segments for user code memory; the user system can also contain external code memory. The emulator uses a memory map to determine whether a given memory address is in emulator-supplied memory or in user-supplied memory. (The LOAD, CBYTE, and ASM commands can write only into emulator-supplied memory.) Controls over access to code memory include the program counter commands, the memory map commands, and the EA/ pin on the emulation processor.

Table 5-4. System Symbols for Code Addresses

System Symbol	Address	Type of Interrupt
.RESET	00H	Power-On Reset
.EXTI0	03H	External Interrupt 0
.TIMER0	0BH	Timer 0 Interrupt
.EXTI1	13H	External Interrupt 1
.TIMER1	1BH	Timer 1 Interrupt
.SINT	23H	Serial Port Interrupt

Program Counter Commands

The 16-bit program counter points to the location of the next instruction to be emulated. The emulator uses the keyword PC to refer to the contents of the program counter; to display the program counter, the command is:

```
*PC
```

To set the program counter to any code address, the format is:

PC = *address*

Examples:

```
*PC = 0
*PC = .START
```

Map Commands

The emulator map divides the 64K code memory space into sixteen blocks of 4K addresses. Each 4K block is identified by the lowest address in the block, as diagrammed in figure 5-2.

The MAP commands can locate the two 4K segments of emulator-supplied memory on any two blocks in the code space, or can map all code memory as user-supplied.

The format is:

```
    [  ( low-address, low-address ) ]
MAP [= <                          > ]
    [  ( USER                     ) ]
```

Examples:

*MAP	;Dispays current mapping.
*MAP=0, 1000H	;Initial setting.
*MAP=0, 20K	;See table 5-5.
*MAP=5000H, E000H	;See table 5-5.
*MAP=USER	;See table 5-5.

As shown in figure 5-2, the low address of a block can be entered as a multiple of 1000H, or as a multiple of 4K.

The initial map setting is 0, 1000H; one of the two 4K segments is mapped to start at address 0, and the other 4K segment is mapped to start at address 1000H. With this mapping, the emulator provides 8K of continuous memory starting with address 0000H; all references to addresses in this range are directed to emulator memory. Any addresses outside this range are assumed to be user-supplied, external memory. With this setting, the EA/ pin must be inactive (high), as discussed later in this chapter.

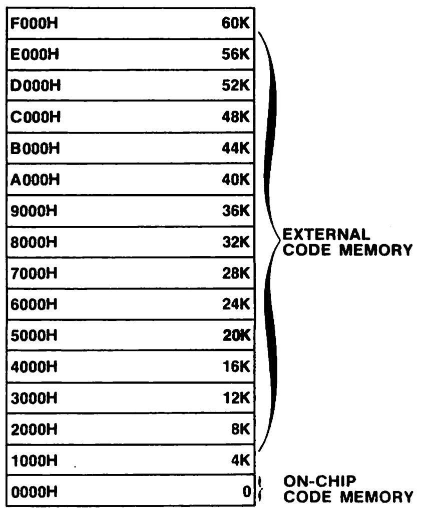


Figure 5-2. Code Memory Mapping

NOTE

The map is reset to 0000H, 1000H by the RESET ICE command.

The two emulator memory segments can be mapped to any two of the sixteen blocks in the map, or the map can be set to USER (no emulator memory is accessed). However, if the lowest block (starting with address 0000H) is not mapped to the emulator, the user must supply external memory for these addresses, and must set the EA/ pin active (low) to cause references to the lowest 4K block to be treated as external addresses rather than on-chip addresses.

Table 5-5 summarizes four combinations of emulator and user memory, showing how to map them, what user memory is required, and what state of the EA/ pin is required for correct operation.

Table 5-5. Mapping Emulator and User-Supplied Memory

Combination Desired	MAP Command (Example)	Emulator Memory	User-Supplied Memory	EA/ Level Required
Initial mapping. All addresses below 2000H are emulator memory.	*MAP=0, 1000H	0000H to 1FFFH, inclusive.	2000H or higher, optional.	Inactive (high)
'On-chip' memory is emulator, higher memory is a combination.	*MAP=0, 5000H	0000H to 0FFFH and 5000H to 5FFFH	2000H to 4FFFH and 6000H to FFFFH, optional.	Inactive (high)
'On-chip' memory is user-supplied, higher memory is a combination.	*MAP=5000H, 6000H	5000H to 6FFFH inclusive.	0000H to 0FFFH is required, 1000H to 4000H and 7000H to FFFFH, optional.	Active (low)
All memory is user-supplied.	*MAP=USER	None	0000H to 0FFFH is required, 1000H to FFFFH, optional.	Active (low)

Using the EA/ Pin

The EA/ (External Access) pin is inactive when it is high (TTL "1") and active when it is pulled to ground (TTL "0"). When EA/ is active, references to addresses in the lowest 4K range are treated as external addresses by the processor. When EA/ is inactive, references to the lowest 4K of addresses are directed to the 'on-chip' memory. As shown in table 5-5, the EA/ pin must be inactive when either of the emulator segments is mapped to start at address 0000H, and must be active when the lowest block of memory is user-supplied.

Any memory not mapped to the emulator is referred to external user-supplied memory. Execution from this memory is handled with the external address, ALE and PSEN/ lines in a normal manner. However, user-supplied memory is read-only to the emulator. Reads from non-existent user memory produce undefined results.

Contents of Code Memory

The emulator system provides access to code memory in two forms: numerically and as assembly language instructions.

Numeric Values

The keyword CBYTE is used to display and change the contents of code memory numerically. To display one or more code bytes, the format is:

CBYTE *partition*

Examples:

```
*CBYTE 0100H
CBYTE 0100H=00H
*CBYTE .START LENGTH 20
.START
0030H=43H 90H 02H 78H 30H 7AH 00H C2H 20H D2H 22H 75H 89H 20H 75H A8H
0040H=88H 75H 8DH F0H C2H 88H D2H 8EH 80H 06H 00H 00H 00H 00H 00H 00H

*CBYTE .TABLE1 TO .TABLE2 - 1
.TABLE1
00ABH=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00BBH=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00CBH=00H 00H 00H 00H 00H
```

To change the contents of one or more code bytes numerically, the format is:

$$\text{CBYTE } \textit{partition} = \left\{\begin{array}{l}\textit{expression}\\ \textit{content-op partition}\\ \textit{string}\end{array}\right\}\left[,\left\{\begin{array}{l}\textit{expression}\\ \textit{content-op partition}\\ \textit{string}\end{array}\right\}\right]\ldots$$

Examples:

```
*CBYTE .TABLE1 = DBYTE .ALPHA LENGTH 26T
*CBYTE 0 TO 8K = 0
*CBY 0700 = 1,2,3,4,5, 'ABCDE', PBYTE 0700 TO 0705
```

Disassembly From Code Memory

The DASM command displays code memory as assembly language instructions. The format is:

DASM *partition*

Examples:

```
*DASM 0
0000H=SJMP       .START
*DASM .START TO .LOOP
.START
0030H=ORL        .P1,#02H
0033H=MOV        R0,#.START
0035H=MOV        R2,#00H
0037H=CLR        .Y
0039H=SETB       .Z
003BH=MOV        .TMOD,#.Y
003EH=MOV        .IE,#.CTABL
```

(Display continues on next page)

```
0041H=MOV        .TH1,#.COUNT
0044H=MOV        .CTABL,#00H
0047H=MOV        .TL1,#.COUNT
004AH=SETB       .TR1
004CH=SJMP       .LOOP
004EH=NOP
004FH=NOP
.LOOP
0050H=MOV        C,.V
```

The display includes the opcode address of each instruction; if symbolic display is enabled, any user symbol matching an opcode address is also displayed on a separate line.

The disassembled instruction includes the opcode and operands as they would appear in an assembly listing. Address values in operands can appear as user or system symbols.

The display consists of complete instructions; an instruction is displayed if its first byte is within the *partition,* even though some subsequent bytes may lie outside the range.

Trace displays in INSTRUCTIONS mode use the same disassembly format for displaying instructions from the trace buffer.

Assembly into Code Memory

The ASM command assembles one instruction into one or more bytes of code memory, at a designated starting address. The code memory accessed must be mapped to the emulator. The system uses an "ASM pointer" to point to the starting address for the next instruction. The pointer automatically increments after assembling each instruction. The pointer is initially at 0000H. To display the location of the ASM pointer, enter the command ASM (for convenience, ASM may be abbreviated to "A"). For example:

```
*A
0000H
```

To change the ASM pointer to a new start location, the format is:

ASM ORG *address*

Example:

```
*A ORG 0100H
0100H
```

The system echoes the new start address.

To enter an instruction into code memory, the format is:

ASM *8051-instruction*

Example:

```
*A MOV A,@R0
0101H
```

To verify the correct assembly, we can examine the memory location:

```
*CBYTE 0100H
CBYTE 0100H=E6H

*D 0100H              ;DASM abbreviates to "D".
0100H=MOV A,@R0
```

The ASM command will assemble most of the 8051 instructions; exceptions are noted below.

Note the following differences between the ASM command and the ASM-51 assembler:

- Generic JMP and CALL instruction types cannot be assembled by the ASM command.
- References to undefined user symbols (forward references) are not allowed.
- The "$" operand (location counter) is not a valid character to the emulator. Display the current value of the ASM pointer to obtain the numeric equivalent.
- The only assembler directive accepted by the ASM command is the ORG *address* directive; no other directives are valid.
- Where the assembler instruction has an address-type operand (*code-address, data address, bit address*), the emulator will accept any *expression* (see chapter 1).
- Where the assembler has immediate data (*#data*), the emulator will accept *expression.*
- Where the assembler has *relative offset,* the emulator will accept *address* (not *expression*).
- The default radix for numbers in the ASM command is hexadecimal (H); in the assembly language, the default is decimal.
- The bit operator (.) is not valid in the ASM mmand; the addition operator has the same effect (+). See "Bit Addresses" earlier in this chapter.
- The emulator system does not accept the SHR, SHL, HIGH or LOW operators.
- Relational operators are accepted, in the algebraic form only (=, >, <, >=, <=, < >); the alphabetic forms (EQ, GT, etc.) are not recognized by the emulator.

One-Byte Mnemonic Constants

You can set a one-byte instruction into code memory by enclosing it in quotes. For example:

```
*CBYTE 0100H = "RETI"
```

If the instruction has more than one byte, only the first byte is set into memory.

[illegible]

CHAPTER 6 ADVANCED EMULATION AND TRACE COMMANDS

This chapter presents complete details on the controls over real-time emulation and trace collection, supplementing those introduced in chapters 3 and 4. The chapter begins with an overview of the software and hardware controls for emulation and trace, and describes the processor information available for matching. It gives details on the commands used to set and enable the hardware and software controls, including the sync lines. The chapter also gives explanations of FRAMES mode trace displays, emulation timer commands, and references to other results of emulation.

Emulation and Trace Controls

Reference Diagrams

This section contains brief but detailed descriptions of emulator and trace controls. The discussions are accompanied by modified logic diagrams. The diagrams show the possible combinations of hardware events and software flags, and their effects. Figure 6-1 explains the symbols used in the reference diagrams.

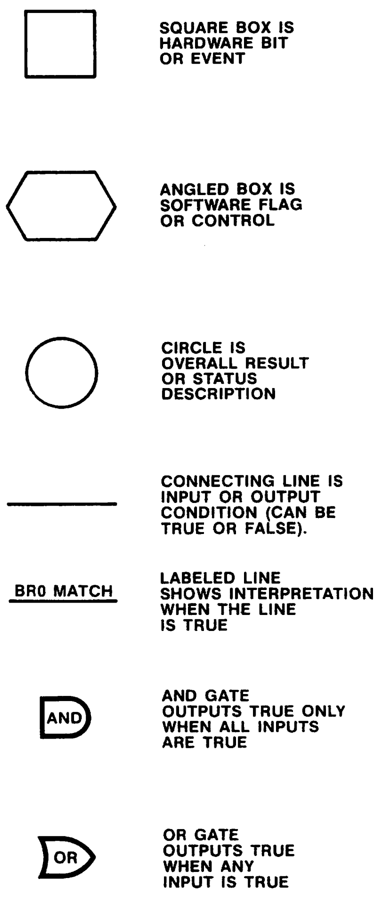

SQUARE BOX IS HARDWARE BIT OR EVENT
ANGLED BOX IS SOFTWARE FLAG OR CONTROL
CIRCLE IS OVERALL RESULT OR STATUS DESCRIPTION
CONNECTING LINE IS INPUT OR OUTPUT CONDITION (CAN BE TRUE OR FALSE).
BR0 MATCH
LABELED LINE SHOWS INTERPRETATION WHEN THE LINE IS TRUE
AND
AND GATE OUTPUTS TRUE ONLY WHEN ALL INPUTS ARE TRUE
OR
OR GATE OUTPUTS TRUE WHEN ANY INPUT IS TRUE

Figure 6-1. Key to Diagram Symbols

Starting Real-Time Emulation

The GO command starts real-time emulation. This command is introduced in simple form in chapter 3. When a GO command is executed, real-time emulation begins, subject to the following conditions (refer to figure 6-2):

- Emulation begins at the current program counter (PC). A FROM clause in the GO command sets PC to a new start location.
- If sync line SY0 IN is not enabled, emulation begins immediately.
- If SY0 IN is enabled and is high when the GO command is executed, emulation begins immediately.
- If SY0 IN is enabled and is low when the GO command is executed, emulation does not begin until SY0 IN goes high. In this way an external device can control the start of emulation. (Commands to enable SY0 IN are discussed later in this chapter).

Halting Real-Time Emulation

The factors that can halt real-time emulation (once it has started) are the ESC key, the GO register, the breakpoint registers, and the status (level/latch) of sync line SY0 IN (refer to figure 6-3). Emulation always halts after completing the instruction that was executing when the halt condition occurred. When emulation breaks for any reason, another GO command is required to resume real-time emulation.

ESC Key

If no other factors are enabled, emulation runs until the user aborts emulation by pressing the ESC key at the console. If other factors are enabled, but the halt conditions they specify have not occurred yet, the ESC key can be used to halt emulation. The ESC key overrides all other halt factors.

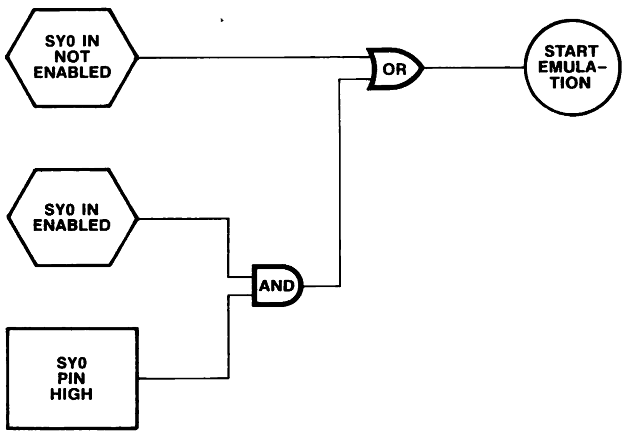


Figure 6-2. Starting Real-Time Emulation

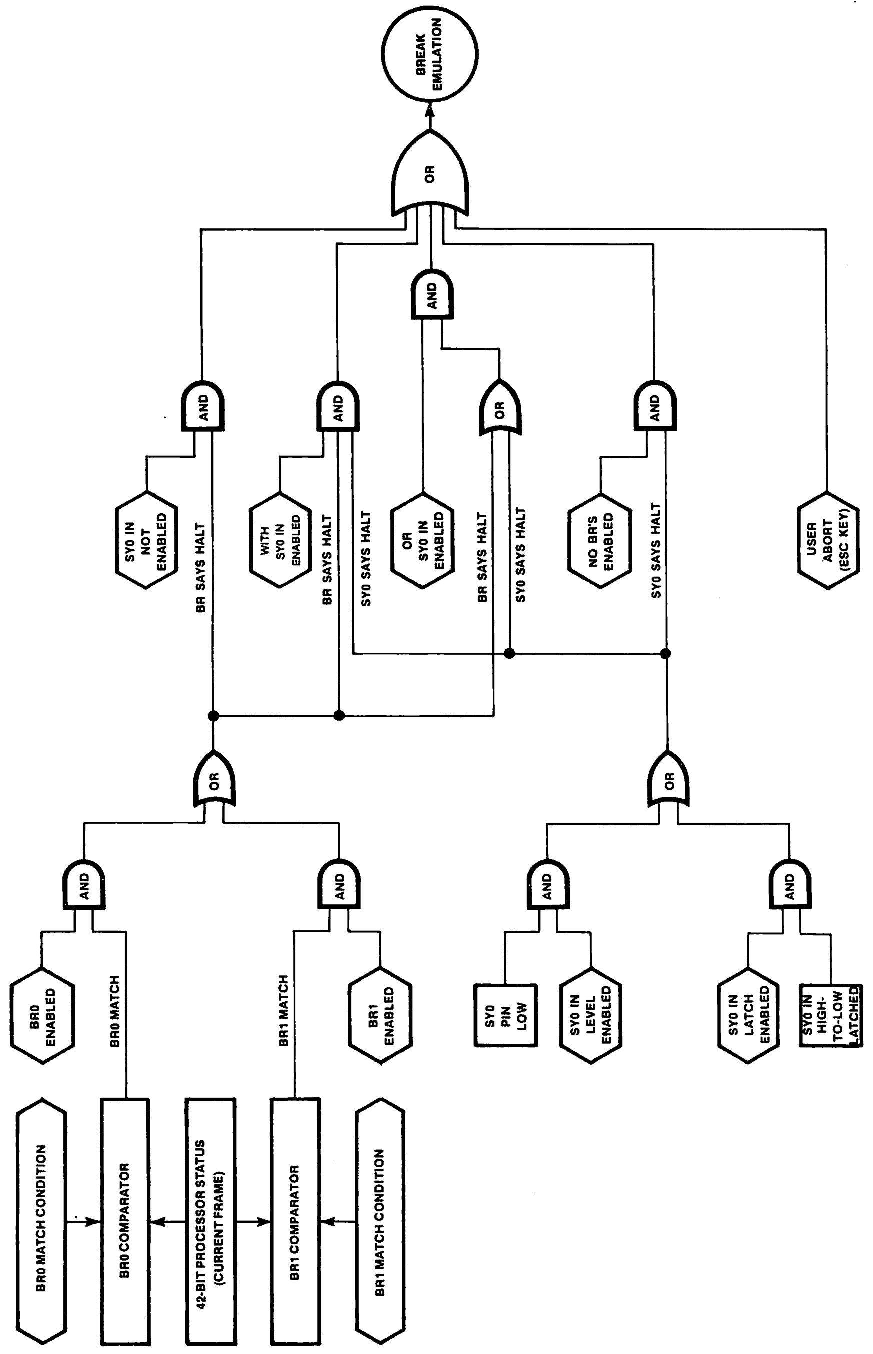


Figure 6-3. Halting Real-Time Emulation

GO Register

The GO register contains flags that indicate the combination of breakpoint register matches and sync line SY0 IN that is currently enabled to halt real-time emulation. Initially, no factors are enabled. The GO register flags are:

BR0 ENABLED
BR1 ENABLED
SY0 ENABLED
WITH SY0 IN ENABLED
OR SY0 IN ENABLED

The possible settings of the GO register flags are:

Setting	**Emulation Halts After:**
FOREVER	User abort (ESC key).
TILL BR0	BR0 match only.
TILL BR1	BR1 match only.
TILL BR	Either BR0 match or BR1 match.
TILL SY0	SY0 IN (going) low only; the timing of the sync signal required depends on the latch/level setting of SY0 IN, as discussed later in this chapter.
TILL BR0 OR SY0	Either BR0 match or SY0 IN low.
TILL BR1 OR SY0	Either BR1 match or SY0 IN low.
TILL BR OR SY0	BR0 match, BR1 match, or SY0 IN low.
TILL BR0 WITH SY0	BR0 match and SY0 IN low simultaneously.
TILL BR1 WITH SY0	BR1 match and SY0 IN low simultaneously.
TILL BR WITH SY0	Either BR0 match or BR1 match, and SY0 IN low simultaneously.

Breakpoint Registers

A breakpoint is a way to break emulation right after a condition of interest occurs. A breakpoint register contains the condition of interest, called a match condition. There are two breakpoint registers, BR0 and BR1; each of them can match on up to 42 bits of processor data, as described later in this chapter.

During emulation, the emulator system compares the match conditions with the current frame of processor data. This data is available for matching even when it is not collected in the trace buffer. On any given frame, BR0 may match, BR1 may match, both may match, or neither may match.

The setting of the GO register determines whether a breakpoint match can halt emulation. If the breakpoint that matched is not enabled in the GO register, the match has no effect.

When the breakpoint register that matched is enabled in the GO register, the condition shown in figure 6-3 as BR SAYS HALT becomes true. If WITH SY0 IN is not enabled in the GO register, emulation breaks when BR SAYS HALT is true. If WITH SY0 IN is enabled, both BR SAYS HALT and SY0 SAYS HALT must be true to break emulation (see next section for SY0 SAYS HALT).

Initially, both breakpoint registers are disabled; the match conditions in both registers are set to match all frames.

Sync Line SY0 IN

SY0 is an external signal accessible at the buffer box of the emulator. This signal can function as an input (SY0 IN) and as an output (SY0 OUT). This section describes SY0 IN; SY0 OUT is described later in this chapter.

SY0 IN is enabled or disabled according to the GO register setting. Initially, it is disabled. When SY0 IN is disabled, it has no effect on emulation. As discussed previously, SY0 IN can prevent emulation from starting if it is enabled and low when the GO command is executed.

If SY0 IN is enabled and is high so that emulation has started, the external device on SY0 IN can set SY0 IN low to request a break in emulation. SY0 IN can be conditioned for one of two modes. Initially, SY0 IN is level-sensitive; in this mode, the line must be held low for a brief period to be recognized. SY0 IN can also be conditioned to latch the low-going edge of the signal; in this mode, the system can recognize a momentary low pulse on SY0 IN.

When SY0 IN is enabled and the appropriate low state of the line is recognized, the condition shown in figure 6-3 as SY0 SAYS HALT becomes true. If WITH SY0 is not enabled in the GO register, SY0 SAYS HALT breaks emulation immediately. If WITH SY0 is enabled, both BR SAYS HALT and SY0 SAYS HALT must be true to break emulation (see previous section for BR SAYS HALT).

Once emulation breaks for any reason, another GO command is required to resume real-time emulation. If SY0 IN caused the break, emulation cannot be restarted by setting the line high again. Note that if SY0 IN is enabled in latch mode and the line is being held low when the GO command is executed, emulation starts when SY0 IN goes high; the initial low state is not latched.

Initially, SY0 IN is disabled, and is level sensitive. Commands to enable and condition SY0 IN are discussed later in this chapter.

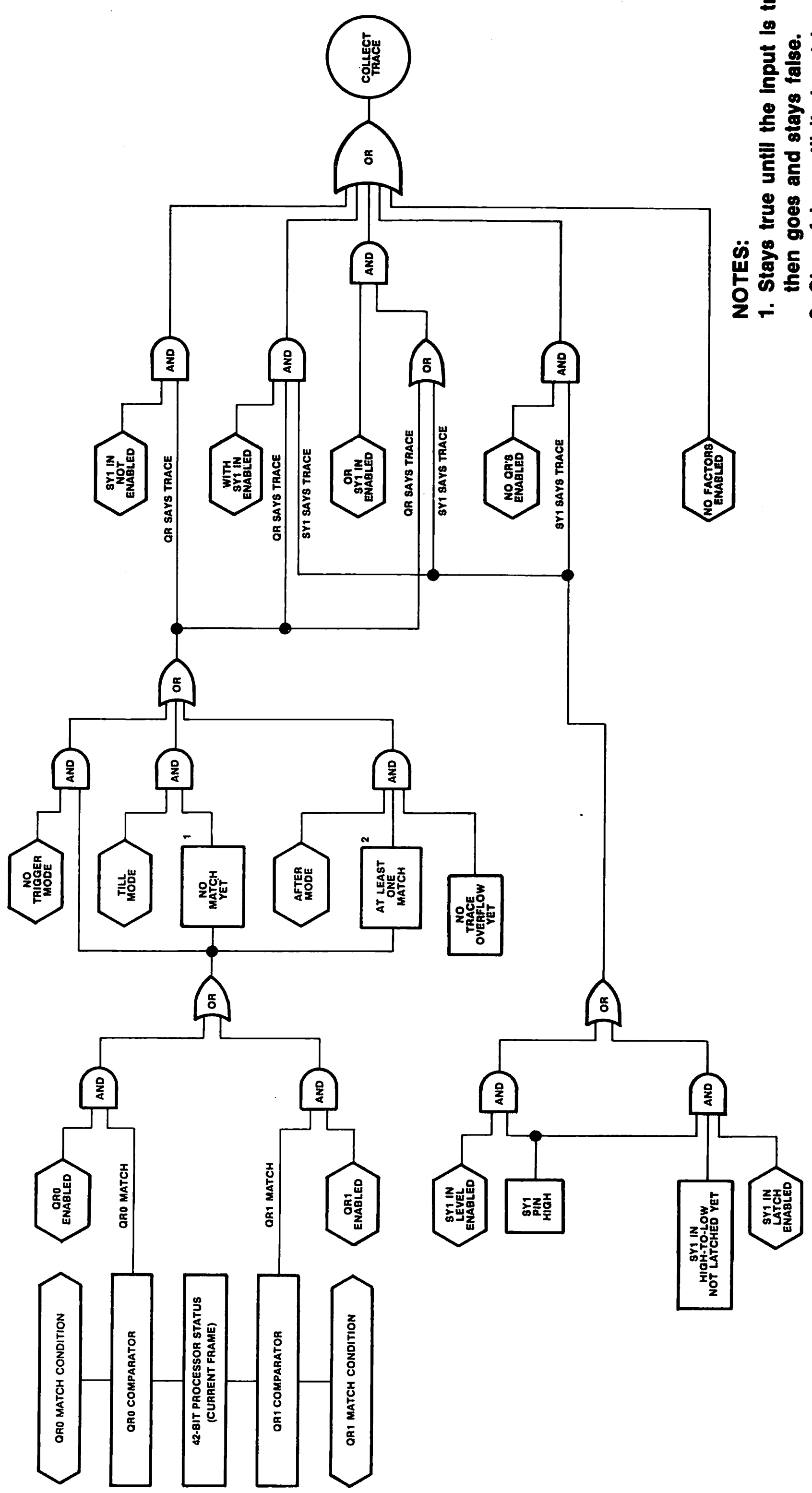
Controlling Trace Collection

The factors that can control trace collection are the trace register, the qualifier registers, and the status (level/latch) of sync line SY1 IN. (Refer to figure 6-4).

Trace Register

The trace register contains flags that indicate the combination of qualifier register matches, trigger mode, and external signal SY1 IN that is currently enabled to control trace collection. Initially, no factors are enabled; in this state, every frame is collected and the buffer retains the most recent 1000 frames. The trace register flags are:

QR0 ENABLED
QR1 ENABLED
TILL MODE ENABLED
AFTER MODE ENABLED
SY1 IN ENABLED
OR SY1 IN ENABLED
WITH SY1 IN ENABLED



NOTES:
1. Stays true until the input is true, then goes and stays false.
2. Stays false until its input is true, then goes and stays true.

Figure 6-4. Controlling Trace Collection

The possible settings of the trace register flags are:

Setting	Trace Collected
FOREVER	Every frame.
QR0	One frame after each match by QR0.
QR1	One frame after each match by QR1.
QR1	One frame after each match by either QR0 or QR1.
TILL QR0 TILL QR1 TILL QR	All frames up to and including the frame on which QR0, QR1, or (with QR) either QR0 or QR1 matched. The buffer retains the last 1000 frames.
AFTER QR0 AFTER QR1 AFTER QR	One bufferful (1000 frames) starting with the next frame after QR0 match, QR1 match, or (with QR) either QR0 or QR1 match.
SY1	All frames when SY1 IN is high. The exact effect of SY1 IN depends on the level/latch status of SY1 IN, as discussed later in this section.
QR0 OR SY1 QR1 OR SY1 QR OR SY1	One frame after each qualifier match, or all frames when SY1 IN is high.
TILL QR0 OR SY1 TILL QR1 OR SY1 TILL QR OR SY1	All frames until qualifier match, thereafter only when SY1 IN is high.
AFTER QR0 OR SY1 AFTER QR1 OR SY1 AFTER QR OR SY1	1000 frames following a qualifier match, otherwise when SY1 IN is high.
QR0 WITH SY1 QR1 WITH SY1 QR WITH SY1	One frame after each qualifier match that occurs while SY1 IN is high.
TILL QR0 WITH SY1 TILL QR1 WITH SY1 TILL QR WITH SY1	Until a qualifier matches, all frames when SY1 IN is high. After a qualifier matches, no further trace is collected.
AFTER QR0 WITH SY1 AFTER QR1 WITH SY1 AFTER QR WITH SY1	Before a qualifier matches, no trace is collected. After a qualifier match, the first 1000 frames that occur while SY1 IN is high.

Details on the settings using with SY1 and OR SY1 are given in the discussions of qualifier registers, trigger modes, and SY1 IN that follow.

Qualifier Registers and Trigger Modes

There are two qualifier registers, QR0 and QR1. Like the breakpoint registers, the qualifier registers contain match conditions that are compared to the current frame of processor data during real-time emulation. The qualifier registers use the same kinds of match conditions as the breakpoint registers. On any given frame, QR0 may match, QR1 may match, both may match, or neither may match.

The setting of the trace register determines the effect of a qualifier register match on trace collection. If the qualifier register that matches is not enabled in the trace register, the match has no effect.

A qualifier register can be enabled in one of three ways: using no trigger mode, using TILL mode, or using AFTER mode. Only one trigger mode (TILL or AFTER) may be in effect at a given time.

When a qualifier register is enabled without a trigger mode and matches the current frame, the condition shown in figure 6-4 or QR SAYS TRACE becomes true for a space of one frame. If WITH SY1 IN is not enabled at this point, the system collects the next frame after the one that matched. If WITH SY1 IN is enabled, the condition shown as SY1 SAYS TRACE (discussed in a later section) must also be true to collect the frame.

The TILL mode uses the qualifier registers to halt trace collection by matching (subject to the other trace register conditions). When one or both qualifier registers are enabled using TILL mode, the condition QR SAYS TRACE is set true when emulation begins, and remains true until an enabled qualifier register matches. If WITH SY1 IN is not enabled in the trace register, trace runs continuously, collecting every frame until an enabled qualifier matches; the buffer overflows after the first 1000 frames. When an enabled qualifier matches, QR SAYS TRACE becomes false; if OR SY1 IN is not enabled, trace halts unconditionally after collecting the frame that matched. (IF OR SY1 IN is enabled when the qualifier match in TILL mode sets QR SAYS TRACE to false, trace collection can continue as long as SY1 IN remains high.) If WITH SY1 IN is enabled and TILL mode has set QR SAYS TRACE to true (waiting for a qualifier to match), the condition SY1 SAYS TRACE must also be true for trace to run.

The AFTER mode uses the qualifier registers to turn trace on for one bufferful (1000 frames); trace then turns off automatically (subject to the other trace register conditions). When one or both qualifiers are enabled using AFTER mode, the condition QR SAYS TRACE is set false when emulation begins, and remains false until an enabled qualifier matches. (If OR SY1 IN is enabled, the condition SY1 SAYS TRACE can cause trace to be collected while QR SAYS TRACE is being held false.) When an enabled qualifier matches in AFTER mode, QR SAYS TRACE becomes true and remains true until the buffer overflows. If WITH SY1 IN is not enabled in the trace register, trace begins with the next frame after the one that matched and runs unconditionally until 1000 frames have been collected; then QR SAYS TRACE becomes false. When WITH SY1 IN is enabled, trace runs when SY1 SAYS MATCH is also true; the AFTER mode still keeps QR SAYS TRACE true as long as the buffer has not overflowed.

NOTE

The trigger modes pertain only to qualifier register matches; sync line SY1 IN may not be used to effect a trigger mode.

Sync Line SY1 IN

Sync line SY1 is accessible at the emulator buffer box. This signal can function as an input (SY1 IN) and as an output (SY1 OUT). This section describes SY1 IN; SY1 OUT is described in a later section.

SY1 IN is enabled or disabled according to the trace register setting. Initially, it is disabled and so has no effect on trace collection. When SY1 IN is enabled, its general effect is to allow trace collection when it is high and to prevent trace collection when it is set low; thus an external device can use SY1 IN to control trace. The exact effect depends on the level/latch status of SY1 IN, and on any qualifier registers or trace modes that may also be enabled.

Initially, SY1 IN is level-sensitive; in this mode, the external device must hold SY1 IN low for a brief period to be recognized. When SY1 IN is enabled and level-sensitive, a low period on the line sets the condition shown in figure 6-4 as SY1 SAYS TRACE to false. SY1 SAYS TRACE remains false as long as SY1 IN is low; if the line is subsequently set high, SY1 SAYS TRACE becomes true and trace collection can resume. When SY1 IN is in level-sensitive mode, trace can be toggled off and on by switching SY1 IN (assuming no other factors are in effect).

SY1 IN can also be conditioned to latch the low-going edge of the input signal. In this mode, the system can recognize a momentary low pulse on SY1 IN. However, when SY1 IN is enabled and in latch mode, and SY1 IN latches a high-to-low transition, SY1 SAYS TRACE is set false unconditionally; it is not set to true if SY1 IN subsequently goes high again.

SY1 IN can be enabled without any other factors, or it can be enabled in combination with any of the qualifier registers (with or without trigger modes). The net effect of the qualifier registers and trigger modes is shown by the condition QR SAYS TRACE in figure 6-4. The possible combinations can be described as follows:

SY1 IN only (no other factors): trace is collected when SY1 SAYS TRACE is true.

(QR SAYS TRACE) OR SY1: Trace is collected when SY1 SAYS TRACE is true, and also when the combination of qualifier registers and trigger mode enabled in the trace register has resulted in the condition QR SAYS TRACE to be true.

(QR SAYS TRACE) WITH SY1: Trace is collected when both QR SAYS TRACE and SY1 SAYS TRACE are true.

Match Conditions

This section gives details on match conditions. The material includes a description of the kinds of information from the 8051 emulation processor that are available for matching, and the formats for match conditions as entered in commands.

Processor Data for Matching

Forty-two bits of processor data are checked at frequent intervals during emulation. The processor data are available for breakpoint register matching (to halt emulation) and for trace qualifier register matching (to control trace collection). This section gives details on the types of processor data available for matching.

Instructions, Cycles, Fetches, and Frames

The unit of emulation is the *instruction.* The emulation always interprets the first byte it fetches as the opcode of the first instruction, and always breaks at the end of an instruction (right before the next opcode fetch).

The unit of execution is the *cycle* (also referred to as "instruction cycle", "minimum instruction cycle", "machine cycle", or "TCY"). The emulator decodes the first byte and determines the number of cycles in the instruction. MCS-51 instructions can require one, two, or four cycles to execute completely.

Each cycle produces two *memory fetches* (also called "memory cycles" or "ALE cycles"). The internal bus timing of a memory fetch is such that the address and the data read from the address are alternately valid, so they can be multiplexed. The 8-bit data appear on the low-order bits of the address lines inside the processor and in trace.

The unit of processor information is the *frame.* The emulator checks 42 bits of processor information for a match twice during each memory fetch. Within a fetch, the first frame is checked when the address is valid, and the second frame is checked when the data is valid. The frame most recently checked is called the *current frame.*

Figure 6-5 shows a diagram of one cycle in terms of two timing signals, ALE (Address Latch Enable) and PSEN/ (Program Store Enable, negative true). The diagram shows the points when frames of processor data are checked. Since there are two fetches, one cycle produces four consecutive frames.

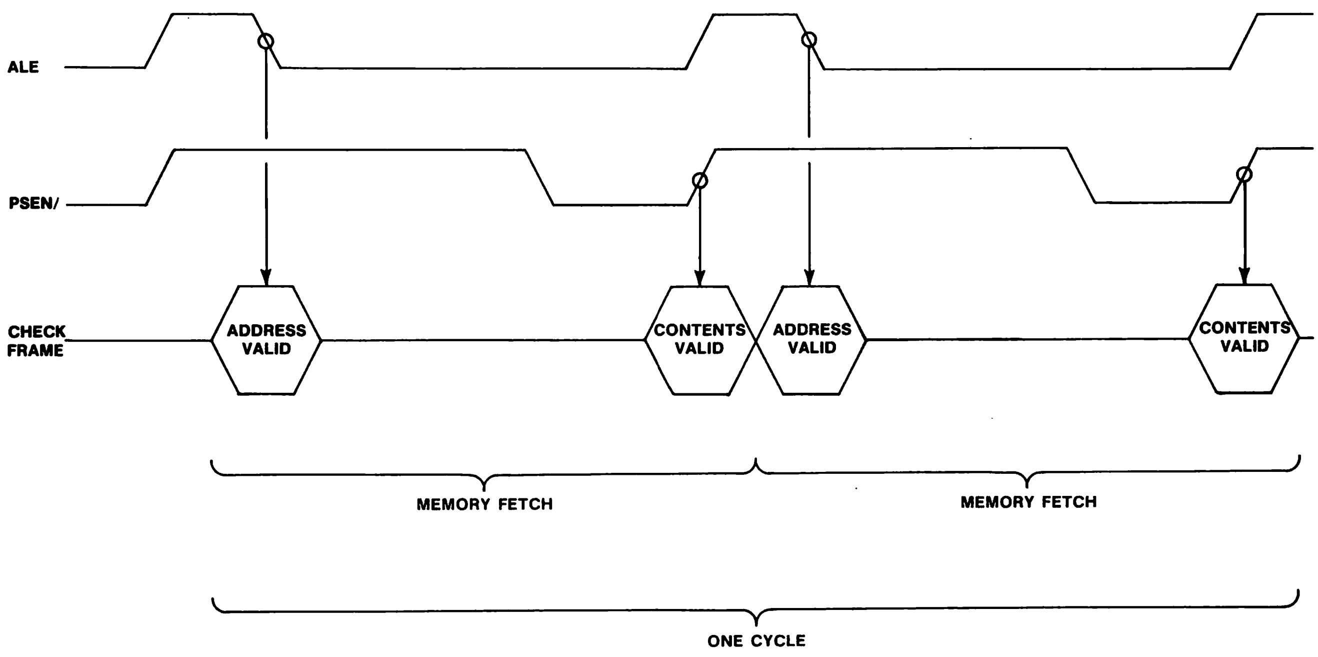


Figure 6-5. Instruction Cycle Timing

Fields Available For Matching

Each frame of trace contains the fields of information available for matching shown in figure 6-6. The following paragraphs give details on these fields of information.

Address in code memory: 16 bits, valid on LOCATION and VLOCATION frames.

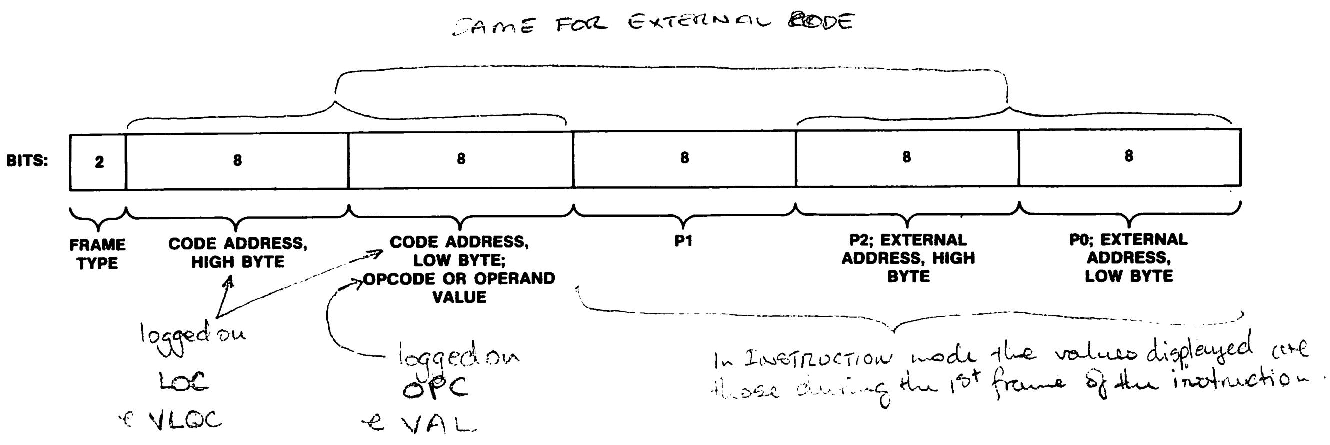
Contents of code memory: 8 bits, multiplexed with the low 8 bits of code address; valid on OPCODE and VALUE frames.

Ports: P0, P1, and P2, 8 bits each.

External address: 16-bit combination of P2 (high 8 bits) and P0 (low 8 bits).

Frame Type: The emulator uses two bits of trace to identify four types of frames. In effect, one bit distinguishes "address-valid" from "contents-valid" frames, as discussed previously; the second bit distinguishes between the first (opcode) fetch and all subsequent fetches (operand or other fetch). The resulting four frame types are described in table 6-1 and diagrammed in figure 6-7 in relation to a two-cycle instruction. Two instruction cycles represent four memory fetches and these in turn produce eight frames of trace. The first two frames, LOCATION and OPCODE, represent the opcode fetch (address, opcode). The remaining three pairs of frames (VLOCATION, VALUE) represent fetches other than the opcode fetch.

(Note that for most instructions the number of fetches exceeds the number of bytes in the instruction; the two-byte, one-cycle instructions have no extra frames. The extra frames are suppressed in INSTRUCTIONS mode trace displays, but can be examined in FRAMES mode as discussed later in this chapter.)


SAME FOR EXTERNAL MODE
BITS:
2
8
8
8
8
8
FRAME TYPE
CODE ADDRESS, HIGH BYTE
CODE ADDRESS, LOW BYTE; OPCODE OR OPERAND VALUE
P1
P2; EXTERNAL ADDRESS, HIGH BYTE
P0; EXTERNAL ADDRESS, LOW BYTE
logged on LOC & VLOC
logged on OPC & VAL
In INSTRUCTION mode the values displayed are those during the 1st frame of the instruction.

ALSO FORMAT OF FRAME i.e. also summarises the data that is captured by the ICE.

Figure 6-6. Match Fields

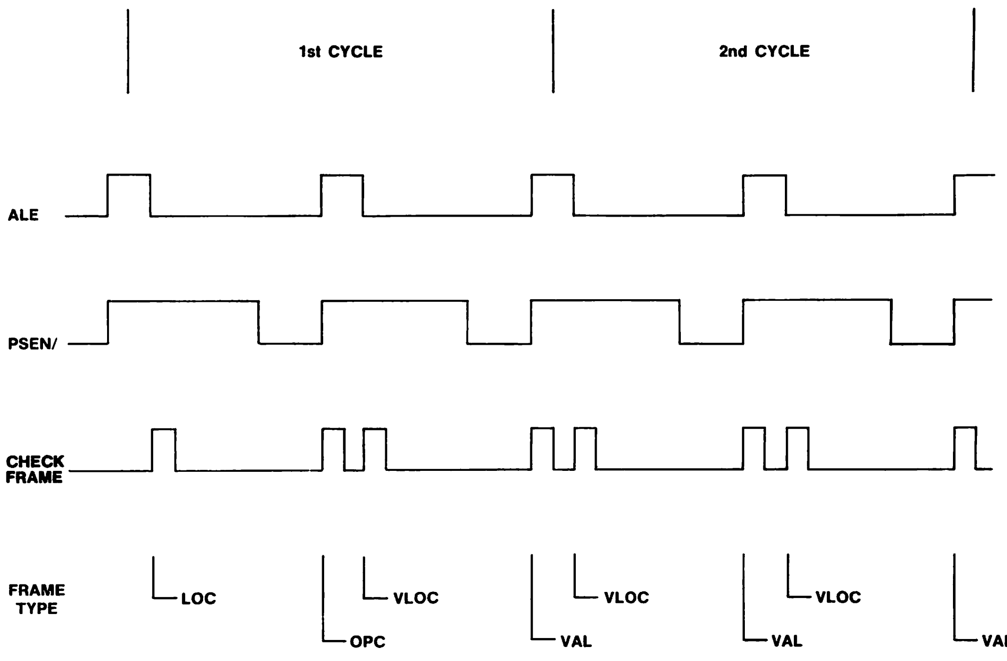


Figure 6-7. Two-Cycle Instruction with Frame Types

Table 6-1. Frame Types in Trace Information

Frame Type	Abbreviation	Information Currently Valid
LOCATION	LOC	Address of opcode, ports, external address.
OPCODE	OPC	Opcode, ports.
VLOCATION	VLO	Address of operand, ports, external address.
VALUE	VAL	Operand, ports.

Formats for Match Conditions

A match condition specifies one or more (non-overlapping) fields of processor data to use for matching, and tells the numeric value or values that are to produce a match. There are two forms of match conditions, "limited" and "unlimited". They differ in how they allow you to specify a range of addresses or values in a match condition; only some commands permit the "unlimited" form, see "Setting Breakpoint and Qualifier Registers" later in this chapter. In general, the term *match-condition* in this manual means the following form:

```
┌             ┐
 LOCATION IS
 VLOCATION IS                  ┌                                     ┐
 ADDR IS                       │     ┌ P0 IS    ┐                    │
 OPCODE IS      ┌ address ┐    │     │ P1 IS    │   ┌ address ┐      │
 VALUE IS       ┤         ├    │ AND ┤ P2 IS    ├   ┤         ├      │ ...
 P0 IS          └ mask    ┘    │     └ XADDR IS ┘   └ mask    ┘      │
 P1 IS                         └                                     ┘
 P2 IS
 XADDR IS
└             ┘
```

In this form, you must use a *mask* to specify a range of match addresses or values.

NOTE

The match conditions shown in chapters 3 and 4 are based on this form.

The "unlimited" match condition allows a *partition* of addresses or values (*address* TO *address* or *address* LENGTH *address*) in addition to a single *address* or *mask*. The form of an "unlimited" match condition is:

```
┌             ┐
 LOCATION IS
 VLOCATION IS                    ┌                                       ┐
 ADDR IS                         │     ┌ P0 IS    ┐                      │
 OPCODE IS      ┌ partition ┐    │     │ P1 IS    │   ┌ partition ┐      │
 VALUE IS       ┤           ├    │ AND ┤ P2 IS    ├   ┤           ├      │ ...
 P0 IS          └ mask      ┘    │     └ XADDR IS ┘   └ mask      ┘      │
 P1 IS                           └                                       ┘
 P2 IS
 XADDR IS
└             ┘
```

The elements of these forms are as follows:

LOCATION IS. Matches the code address field on LOCATION frames only. LOCATION is the default if no frame type is designated.

VLOCATION IS. Matches the code address field in VLOCATION frames only.

ADDR IS. Matches the code address field on both LOCATION and VLOCATION frames.

OPCODE IS. Matches the opcode field on OPCODE frames only.

VALUE IS. Matches the operand field on VALUE frames only.

P0 IS. Matches the P0 (port 0) field.

P1 IS. Matches the P1 field.

P2 IS. Matches the P2 field.

XADDR IS. Matches the combination of the P2 field (upper byte) and the P0 field (lower byte).

address. A number, reference, of expression. Although this entry is not literally an "address" when used with OPCODE, VALUE, or ports, the same kinds of entry are permitted as for addresses.

mask. A binary, octal, or hexadecimal number with one or more digits specified as don't-care (X). See chapter 3 for details on masks in match conditions.

AND. Combines a match on P0, P1, or P2 with the condition specified in the previous clause (or clauses) of the match condition. More than one port can be ANDed to form a combination. To produce a match, the values specified must all occur on the same frame.

partition. A range of addresses or other values. See chapter 1 for details on partitions.

NOTE

When matching on instructions that change ports P0, P1, or P2, note that the changed value of the port does not appear until the next OPCODE frame following the execution of the instruction that changed the port. This limitation applies only to the individual ports, not to ports used as an external address, and affects trace collection only, not breakpoints.

Emulation and Trace Commands

This section describes the GO register and Trace register commands, the breakpoint and qualifier register commands, and the external signal lines commands.

Command Formats

The following format describes the possible entries in the GO register command:

```
        [   { FOREVER                                                        } ]
        [   {          { BR0 }  [ OR SYO   ]                                 } ]
        [   { TILL     { BR1 }  [ WITH SYO ]                                 } ]
GR      [ = {          { BR  }                                               } ]
        [   { TILL match-condition [OR match-condition]  [ OR SYO   ]        } ]
        [   {                                            [ WITH SYO ]        } ]
        [   { TILL SYO                                                       } ]
```

NOTE

In addition to the GR commands, the GO command can set the GO register; the GO command also starts real-time emulation. To obtain the format for this command, substitute "GO" (with or without a FROM clause) for "GR =" in the above format.

The possible entries for the Trace register command are:

```
        ⎡ ⎧FOREVER                                                              ⎫⎤
        ⎢ ⎪⎡TILL ⎤   ⎧QR0⎫  ⎡OR SY1  ⎤                                           ⎪⎥
        ⎢ ⎪⎣AFTER⎦   ⎨QR1⎬  ⎣WITH SY1⎦                                           ⎪⎥
TR      ⎢=⎨          ⎩QR ⎭                                                       ⎬⎥
        ⎢ ⎪⎡TILL ⎤                                                   ⎡OR SY0  ⎤  ⎪⎥
        ⎢ ⎪⎣AFTER⎦  match-condition [OR match-condition]             ⎣WITH SY0⎦  ⎪⎥
        ⎣ ⎩SY1                                                                   ⎭⎦
```

Displaying the Go and Trace Registers

To display the Go register, enter the command GR:

```
*GR
GR = FOREVER
BR0 = RESET
BR1 = RESET
```

To display the trace register, enter the command TR:

```
*TR
TR = FOREVER
QR0 = RESET
QR1 = RESET
```

In both displays, the first line shows the flags that are enabled, or the keyword FOREVER if no flags are enabled.

The GR command also displays the settings of both breakpoint registers. Initially, both registers are RESET, as shown in the example; this condition matches every frame. Similarly, TR displays the settings of both qualifier registers.

Disabling All Factors in the Go and Trace Registers

Since the breakpoint and trace qualifier flags are independent of the match conditions, you can disable the match registers and other factors without changing the match conditions. The formats are:

```
GR = FOREVER                  ;Also, GO FOREVER
TR = FOREVER
```

Enabling Break and Qualifier Registers

To enable break or qualifier registers without changing the settings, the commands are:

```
GO TILL BR0                   TR = QR0
GO TILL BR1                   TR = QR1
GO TILL BR                    TR = QR
GR = TILL BR0                 TR = TILL QR0
GR = TILL BR1                 TR = TILL QR1
GR = TILL BR                  TR = TILL QR
                              TR = AFTER QR0
                              TR = AFTER QR1
                              TR = AFTER QR
```

The forms with GO also begin real-time emulation. In these commands, the token BR means "either BR0 or BR1", and QR means "either QR0 or QR1".

Setting Match Conditions in the GO and Trace Registers

This section reviews the kinds of match conditions permitted in the GO and trace registers.

Command Formats

The general formats for the GO register commands are as follows:

GO TILL *match-condition* [OR *match-condition*]

GR = TILL *match-condition* [OR *match-condition*]

The form with GO sets the GO register, then starts real-time emulation; with GR, no emulation is started. The first match condition sets and enables BR0; the second, if present, sets and enables BR1. Emulation halts when an enabled breakpoint register matches.

The formats for the trace register commands are as follows:

TR = *match-condition* [OR *match-condition*]

TR = TILL *match-condition* [OR *match-condition*]

TR = AFTER *match-condition* [OR *match-condition*]

In all three formats, the first match condition sets and enables QR0; the second, if any, sets and enables QR1.

In the first format (no trigger mode), a match by the enabled qualifier register collects one frame of trace, the one right after the one that matched. In the second format (TILL mode), a match halts trace after the frame that matched. In the third format (AFTER mode), a match turns trace on until the buffer overflows.

Matching On Addresses

To match on the address of an opcode, the format is:

[LOCATION IS] *address*

Examples:

```
*GR = TILL 0100H
*TR = TILL LOCATION IS .LOOP
```

As the format and examples show, LOCATION is the default when no explicit frame type is entered. (The match conditions in chapters 3 and 4 assume this default.)

Instead of a single address, you can use a *mask* to specify that some bits in the field are to be ignored in matching; the format is:

[LOCATION IS] *mask*

Examples:

```
*GR = TILL LOCATION IS 02XXH
*TR = AFTER 02XXH
```

In a mask, the "X" digits are ignored in the match. For hexadecimal masks such as the examples above, each X digit represents four consecutive bits (02XXH allows a match on any address from 0200H through 02FFH inclusively). Refer to chapter 3 for more information on *masks.*

To match on the address of an operand, the format is:

VLOCATION IS $\begin{Bmatrix} \textit{address} \\ \textit{mask} \end{Bmatrix}$

Examples:

```
*GR = TILL VLOCATION IS 0101H
*TR = AFTER VLOCATION IS .LOOP + 1
*GR = TILL VLOCATION IS 02XXH
```

To match on any address-valid frame, the format is:

ADDR IS $\begin{Bmatrix} \textit{address} \\ \textit{mask} \end{Bmatrix}$

Examples:

```
*GR = 1 TILL ADDR IS 02XXH
*TR = TILL ADDR IS 800H
```

A match condition with ADDR matches frame types LOCATION (opcode address) and VLOCATION (operand address).

Matching On Opcode and Operand Values

To match on an opcode, the format is:

OPCODE IS $\begin{Bmatrix} \textit{address} \\ \textit{mask} \end{Bmatrix}$

Examples:

```
*TR = AFTER OPCODE IS 34H
*GR = TILL OPCODE IS 4XH
*TR = TILL OPCODE IS "MOVC A, @A + DPTR"
```

In the format definition shown above, the entry *address* is an 8-bit quantity; although it is not literally an "address", the same kinds of entry are permitted as for an address. Refer to chapter 1 for the discussion of the "data type" *address.*

To match on an operand value, the format is:

VALUE IS $\begin{Bmatrix} \textit{address} \\ \textit{mask} \end{Bmatrix}$

Examples:

```
*GR = TILL VALUE IS 50H
*TR = VALUE IS XXH
```

The second example (TR = VALUE IS XXH) has the effect of maximizing the number of LOCATION frames collected (per 1000 frames). When a VALUE frame matches (with no trigger mode) the frame following the matching frame is collected in the buffer. Because each cycle contains a VALUE frame as the last frame (even on one-byte instructions), the effect is to collect the first frame of every cycle after the first. This collects the LOCATION frame of every instruction, so that the instruction can be displayed in INSTRUCTION mode (see chapter 3). It also collects some VLOCATION frames from two-cycle and four-cycle instructions. The extra VLOCATION frames are suppressed during INSTRUCTION mode displays. Frame types LOCATION, VLOCATION, and VALUE are discussed earlier in this chapter.

Matching on Ports and External Addresses

To match on a port setting, the format is:

```
{P0}
{P1}  IS  {address}
{P2}      {mask   }
```

Examples:

```
*GR = TILL P0 IS 33H
*TR = P1 IS 0FXH
*GR = TILL P2 IS 0
```

To match on an external address (concatenation of P2 and P0), the format is:

```
XADDR  IS {address}
          {mask   }
```

Examples:

```
*GR =TILL XADDR IS 8000H
*TR = XADDR IS 4XXXH
```

Matching On Combinations With Ports

The following repeatable form can be used in combination with any of the match conditions described above:

```
[{AND P0 IS     }                ]
[{AND P1 IS     }   {address}    ]...
[{AND P2 IS     }   {mask   }    ]
[{AND XADDR IS  }                ]
```

Examples:

```
*GR = TILL 0100H AND P0 IS F7H
*TR = AFTER ADDR IS 01FXH AND P1 IS 0A0H
*GR = TILL OPCODE IS 0E0H AND XADDR IS 35XXH AND P1 IS F0H
```

The system does not check these combinations for consistency; if the combination is such that the same port is mentioned more than once, the last setting is the one used in the match condition and the other settings are lost. For example:

```
*GR = TILL P0 IS 00H AND P0 IS FFH
```

This condition is equivalent to:

```
*GR = TILL P0 IS FFH
```

Using Two Match Conditions

All the forms of match condition described so far in this chapter require just one breakpoint register or one qualifier register, since they involve no overlapping fields. You can enter two breakpoint match conditions or two qualifier match conditions by using the format:

match-condition OR *match condition*

Examples:

```
*GR = TILL 0100H OR 0200H
*TR = LOCATION IS .LOOP OR OPCODE IS A3H
*GR = TILL P0 IS 0 AND P1 IS 1 OR P0 IS F0H AND P1 IS 0FH
```

The first match condition is loaded into BR0 (for GR) or QR0 (for TR), and the second match condition is loaded into BR1 or QR1.

Breakpoint and Qualifier Register Commands

The commands in this section display and change the match conditions in the breakpoint and qualifier registers directly. These commands do not change the GO register or trace register settings. These commands do not enable or disable the breakpoint or qualifier registers. They allow you to use unlimited match conditions (with partitions).

Displaying Breakpoint and Qualifier Registers

To display a breakpoint or qualifier register setting, enter the name of the register as a display command:

```
*BR0
BR0 = LOCATION IS 0030H

*BR1
BR1 RESET

*QR0
QR0 = VALUE IS XXXXXXXXY

*QR1
QR1 RESET
```

To display the settings of both breakpoint or both qualifier registers, enter the command:

```
*BR
BR0 = LOCATION IS 0030H
BR1 RESET

*QR
QR0 = VALUE IS XXXXXXXXY
QR1 RESET
```

The GR and TR display commands discussed above also display the breakpoint and qualifier register settings, respectively.

Setting Breakpoint and Qualifier Registers

To set a match condition into a breakpoint or qualifier register without enabling the register, the format is:

$$\left\{\begin{array}{l}\text{BR0}\\\text{BR1}\\\text{BR}\\\text{QR0}\\\text{QR1}\\\text{QR}\end{array}\right\} = \textit{unlimited-match-condition}$$

Examples:

```
*BR0 = LOCATION IS 0100H
*BR = XXXXH
*QR1 = VALUE IS 10H
*QR = P1 IS F0H AND P2 IS 0FH
*BR0 = 0100H TO 0200H
*BR1 = VALUE IS 10H TO 1FH
*QR0 = P0 IS 00H TO 7FH AND P1 IS 80H TO FFH
```

These forms set match conditions into the individual registers (or into both registers identically, when BR or QR is used), but do not affect the enabled/disabled status of the register.

NOTE

The breakpoint and qualifier register commands do not reset any values previously set in the specified register; only the bits specified in the match condition are affected. (See RESET command, next section.) By contrast, setting a match condition with the GO, GR, or TR commands resets any bits not mentioned to don't-care.

Adjustment of Partitions in Match Conditions

When you enter a partition in a BR or QR command, the system adjusts the partitions you enter so that a match can be made in units of four bits ("nibbles"). Each nibble is one hex digit, so it is easier to explain how adjusting works using hex values.

An address has at most four digits; to represent a partition of addresses the system uses two four-digit numbers indicating the low and high addresses in the partition. In adjusting the partition, the system first identifies the highest-order nibble for which the two corresponding digits in the two numbers are different. As examples, in partition 0100 TO 0102 the high and low addresses differ only in the least significant nibble; in partition 1234 TO 56789 all digits differ; the highest-order difference is in the most significant nibble.

After the system identifies the highest-order difference, it adjusts any lower-order nibbles in the low bound to 0, and any lower-order nibbles in the high bound to FH (1111Y). Using the two examples just given, the partition 0100 TO 0102 differs only in the least significant nibble; there are no lower-order nibbles and no adjustment is made. (A message may still be displayed.) In the partition 1234 to 5678, the most significant nibbles differ; the low bound (1234H) would be adjusted to 1000H and the high bound (5678H) would be adjusted to 5FFFH. The system informs you of the adjustment:

```
*BR0 = LOCATION IS 1234 TO 5678H
LOCATION ADJUSTED TO 1000H = 5FFFH
```

Resetting Break and Qualifier Registers

To reset the match conditions in the breakpoint and qualifier registers without changing the flags (enabled/disabled), the format is:

```
        ( BR0 )
        ( BR1 )
RESET  <  BR1  >                  ;BR here means "both BR0 and BR1".
        ( QR0 )
        ( QR1 )
        ( QR  )                   ;QR here means "both QR0 and QR1".
```

Examples:

```
*RESET BR0
*RESET BR
*RESET QR1
*RESET QR
```

If either breakpoint register is enabled while RESET, emulation will break after executing one instruction. If either qualifier register is enabled while RESET, all OPCODE and VALUE frames of every instruction will be collected.

Sync Line Commands

Sync lines SY0 and SY1 are accessible at the buffer box. A cable is provided for connecting these signals to external equipment; refer to appendix A for installation.

SY0 Commands

The SY0 line synchronizes the start or stop of emulation with external events. It can be used as an input signal (SY0 IN) or as an output signal (SY0 OUT) or both. The functions of SY0 IN and SY1 IN are discussed earlier in this chapter.

SY0 IN is enabled as a halt condition in the GO register. The command formats are:

GO TILL SY0

GO TILL *breakpoints* OR SY0

GO TILL *breakpoints* WITH SY0

GR = TILL SY0

GR = TILL *breakpoints* OR SY0

GR = TILL *breakpoints* WITH SY0

In these formats, the entry *breakpoints* means one of the keywords BR0, BR1 or BR, or any of the forms of *match-condition* discussed earlier in this chapter. The forms with GO also begin real-time emulation; the optional FROM clause is not shown in the above formats for clarity (refer to chapter 3 for details).

Examples:

```
*GO TILL SY0
*GO TILL BR0 OR SY0
*GR = TILL VALUE IS 10H OR SY0
*GR = TILL P0 IS 1FH WITH SY0
```

SY0 IN is disabled by another GO or GR command that does not include a reference to SY0 IN, such as GO FOREVER.

SY0 IN can be combined with one or more breakpoints. The clause OR SY0 in a GO or GR command means either a breakpoint match or SY0 IN low can halt emulation. The clause WITH SY0 means both a breakpoint match and SY0 IN low are required to halt emulation. (SY0 IN has no effect on single-step emulation.)

Initially, SY0 IN is level-sensitive; the external device must hold SY0 IN low for a period of time to produce a break. (See appendix D for SY0 timing parameters.) You can also specify that the system is to latch SY0 IN on the high-to-low transitional edge; thus a momentary low will cause a break in this condition. To cause latching instead of level-sensitivity, the command is:

```
ENABLE SY0 LATCH
```

To restore the level-sensitive condition, the command is:

```
DISABLE SY0 LATCH
```

The level-sensitive or latched condition is independent of the enabled/disabled status of SY0 IN in the GO register.

SY0 OUT is enabled or disabled with the commands:

```
ENABLE SY0 OUT
DISABLE SY0 OUT
```

When SY0 OUT is enabled, the emulator holds the line low and releases it only when emulation is in progress. If enabled, SY0 OUT is pulsed during single-step emulation.

SY1 Commands

The SY1 line synchronizes the start and stop of trace collection with external events. It can be used as an input (SY1 IN), as an output (SY1 OUT), or both.

SY1 IN is enabled as a qualifier in the trace register. The command formats are:

TR = SY1

TR = *tracepoints* OR SY1

TR = *tracepoints* WITH SY1

TR = TILL *tracepoints* WITH SY1

TR = TILL *tracepoints* WITH SY1

TR = AFTER *tracepoints* OR SY1

TR = AFTER *tracepoints* WITH SY1

Examples:

```
*TR = SY1
*TR = QR WITH SY1
*TR = VALUE IS 1XH OR SY1
*TR = TILL QR0 OR SY1
*TR = AFTER OPCODE IS 3CH OR SY1
```

In these formats, the entry *tracepoints* means one of the keywords QR0, QR1, or QR, or any of the forms of *match-condition* discussed earlier in this chapter.

SY1 IN is disabled by another TR command that does not include a reference to SY1 IN, such as TR = FOREVER.

Initially, SY1 IN is level-sensitive; after it has been high (starting trace), the external device must hold it low for a period of time to halt collection of trace. (See appendix D for SY1 timing parameters.) Under this condition, however, the external device can toggle trace on and off repeatedly by manipulating SY1 IN. Instead of level-sensitivity, you can also specify that the system is to latch SY1 IN on the high-to-low transitional edge; thus a momentary low will cause trace to halt. To enable or disable latching, the commands are:

```
ENABLE SY1 LATCH
DISABLE SY1 LATCH
```

The level-sensitive or latched condition is independent of the enabled/disabled status of SY1 IN in the trace register.

In the latched state, once trace has turned off by SY1 IN going low, it cannot restart until emulation breaks and resumes, even though SY1 IN may go high again.

SY1 OUT is enabled and disabled with the commands:

```
ENABLE SY1 OUT
DISABLE SY1 OUT
```

When SY1 OUT is enabled, the emulator holds the line low except when trace is being collected.

RESET SY Command

The command RESET SY is equivalent to the following four commands:

```
*DISABLE SY0 LATCH
*DISABLE SY0 OUT
*DISABLE SY1 LATCH
*DISABLE SY1 OUT
```

In other words, RESET SY restores the level-sensitive condition for both SY0 IN and SY1 IN, and disables SY0 OUT and SY1 OUT. It does not affect the enabled/disabled status of SY0 IN or SY1 IN (which require changing the GO and trace registers, respectively).

References to Sync Lines

The commands SY0 and SY1 display the current levels of the two lines (0 = low, 1 = high); for example:

```
*SY1
SY1=1

*SY1
SY1=1
```

The command SY displays the levels of SY0 and SY1, and also displays the enabled/disabled status of SY0 LATCH, SY0 OUT, SY1 LATCH and SY1 OUT; for example:

```
*SY
SY0 OUT   DISABLED
LATCH DISABLED
INPUT HIGH
SY1 OUT   DISABLED
LATCH DISABLED
INPUT HIGH
```

The keywords SY0 and SY1 can also be used as read-only references in expressions and other numeric contexts. SY0 and SY1 are 1-bit quantities (0 or 1).

For example:

```
*EVALUATE SY0
1Y 1Q 1T 1H '' 0001H
```

Note that the emulator languages uses context to decide whether the keywords SY0 or SY1 refer to enabled flags in the GO and trace registers or to 1-bit numeric quantities. One point of possible confusion is after the command word TILL. Consider the following sequence of GO register commands:

```
*GR = TILL SY0                ;Enables SY0 IN.

*GR                           ;Display GO register and
GR = TILL SY0                 ;breakpoint register settings.
BR0 RESET
BR1 RESET
```

In the GO register commands, TILL introduces one or more halt-factors; SY0 is interpreted as a valid halt factor, and its flag is set accordingly. Now compare the following trace register commands:

```
*TR = TILL SY1                    ;Enables QR0 using setting of SY1 as
                                  ;the address.
*TR
TR = TILL QR0
QR0 = LOCATION IS 0000H
QR1 RESET
```

In the trace register commands, TILL is a trigger mode in combination with the tracepoint that immediately follows it. In this context, the keyword SY1 is interpreted as a tracepoint address and, assuming SY0 is 0, the value is set into QR0 and QR0 is enabled as shown in the example.

Other Results of Emulation and Trace

Emulation Timer

The emulation timer is a 32-bit, 500-nanosecond counter maintained by the emulator system. The emulation timer runs during real-time emulation, providing an approximate measure of the duration of the emulation.

To display the timer value, enter the command SECONDS. The display gives the emulation timer value rounded to the nearest microsecond.

For example:

```
*SECONDS
2 MICROSECONDS

*SECONDS
4,090,769 MICROSECONDS
```

To display the low and high words of the emulation timer in the current output radix, use the commands TIMER and HTIMER.

Examples:

```
*TIMER                            ;Low 16 bits.
TIMER=D723H

*HTIMER                           ;High 16 bits.
HTIMER=007CH
```

The emulation timer is accurate to about .01%, or about 1 in 10,000 microseconds, however, breaking emulation produces a "roundoff" error of ±1 microsecond.

The emulation timer is reset to zero when the program counter is changed by a PC=*address*, GO FROM *address*, or *LOAD filename* command. The timer is also set to zero and remains at zero during single-step emulation.

CAUSE Command

The CAUSE command displays the reason for the last break in emulation. The CAUSE keyword can also be used as a read-only reference in expressions and other numeric contexts. As a display:

```
*CAUSE
CAUSE = 01H
BREAK CAUSED BY: BR0
```

The numeric byte displayed has the following interpretation:

Bit 0 on if BR0 match caused break.*

Bit 1 on if BR1 match caused break.*

Bit 2 on if SY0 caused break.

Bit 3 on if single-step caused break.

Bit 4 on if user aborted emulation.

(*On occasion both bits may be on when only one is true.) The display also includes the cause of break in message format.

As a numeric entry, CAUSE is a 5-bit quantity.

BUFFERSIZE Command

The trace buffer holds a maximum of 1000 frames, numbered from 000 to 999 decimal. A frame counter, BUFFERSIZE, points to the next frame available to receive the current new frame of information. The buffer is initially empty (BUFFERSIZE = 0); after overflow, BUFFERSIZE = 1000T.

To display the number of valid frames in trace, enter the command BUFFERSIZE; for example:

```
*BUFFERSIZE
BUF=0
```

The keyword BUFFERSIZE can also be used as a read-only reference in expressions and other numeric contexts.

PPC and OPCODE Commands

The command PPC displays the address of the last instruction in the trace buffer, and the command OPCODE displays the opcode of the last instruction.

For example, assuming the last instruction in trace would display as follows:

```
*NEWEST
*TRACE=INSTRUCTIONS
*PRINT -1
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
.START
0008:  0030H  439002  ORL   .P1,#02H           FFH  FFH  FFH
```

PPC and OPCODE then produce displays as follows:

```
*PPC
PPC=0030H
*OPCODE
OPCODE=43H
```

NOTE

The values of PPC and OPCODE are updated after each break in emulation if a LOCATION frame can be found within the last sixteen frames of trace. When no LOCATION frame occurs, a warning message is displayed to inform you that PPC and OPCODE have not been updated. If you then display PPC or OPCODE, the value is incorrect but no subsequent warning is given.

Trace References

A trace reference is an entry of the form:

FRAME *trace-group*

Where *trace-group* is one of the following entries:

Trace Group	**Meaning**
ADDR	16-bit code address (on LOC and VLOC frames)
DATA	8-bit quantity (on OPC and VAL frames)
DMUX	1-bit, 0 if address frame, 1 if data frame.
CYC	1-bit, 0 if first fetch, 1 if any other fetch.
P0	8-bit, Port 0
P1	8-bit, Port 1
P2	8-bit, Port 2
XADDR	16-bit, external address (concatenation of P2 and P0)
TOVF	1-bit, trace overflow flag (1 = overflow).

The value of a trace reference in an expression is the trace data corresponding to *trace-group* in the frame pointed at by the trace display pointer (see chapter 4). An error occurs if the pointer is at NEWEST or if the buffer is empty.

Here's how a trace reference to port P1 works. Suppose the buffer is at OLDEST, so that the PRINT command displays frame 0000:

```
*OLDEST
*
*PRINT 1
FRAME LOC    OBJ   INSTRUCTION P1   P2   P0   TOVF
0000:  0000H 2100  AJMP   .START FFH FFH  FFH   0
*
```

Now, to reference the value of P1 from this frame, we use a trace reference, FRAME P1. This reference is not a command, however, so we need to use EVALUATE to see the value:

```
*EVALUATE FRAME P1
1111111111111111Y 177777Q 65535T FFH '' 00FFH
```

FRAME Mode Trace Displays

Trace Display Mode

Display of trace information is controlled by the trace display mode. There are two modes, INSTRUCTIONS mode and FRAMES mode. Table 6-2 summarizes the differences between the two modes. Note that the trace display mode does not affect the manner in which trace is collected, only the manner of display of information already collected.

The trace display mode command changes the mode; the format is:

$$\text{TRACE} = \begin{bmatrix}\text{INSTRUCTION}\\ \text{FRAME}\end{bmatrix}$$

The initial mode is INSTRUCTION. INSTRUCTION mode displays are discussed in chapter 4. To obtain the FRAME mode displays discussed in this section, enter the command:

```
*TRACE = FRAME
```

To return to INSTRUCTION mode, the command is:

```
*TRACE = INSTRUCTION
```

To display the current trace display mode, the command is:

```
*TRACE
```

Example:

```
*TRACE
TRACE = INSTRUCTION

*TRACE
TRACE = FRAME
```

Table 6-2. Trace Display Modes

INSTRUCTION MODE	FRAME MODE
Each display line contains one complete instruction (opcode, operands).	Each display line contains one complete frame (opcode or operand).
An instruction is displayed only if its first frame was collected.	All frames are displayed, even if they do not form complete instructions.
The display can include information that was not actually collected.	Only frames that were collected are displayed.
The values of ports and TOVF are from the first frame in the instruction.	Ports and TOVF are sampled on every frame, and the display shows the value on each frame.
Opcodes and operands are reconstructed from program memory for comparison with the trace values.	Only opcodes are reconstructed from program memory.
Opcodes and operands are disassembled.	Only the first byte is disassembled (opcode, some operands).
The trace version is displayed if it exists, otherwise the reconstruction is displayed.	Both the reconstruction and the actual trace information are displayed.
All addresses can be displayed symbolically.	Code addresses can be displayed symbolically.
The user is informed of any discrepancy between trace and memory, with a message.	No message is displayed.

FRAME Mode Headers

Figure 6-8 shows an example of a display in FRAME mode. Each line of the display contains one frame. The display begins with a header to identify the fields of information. The next several paragraphs briefly describe the header items as they appear in the figure from left to right.

FRAME: The frame number. Frames in the trace buffer are numbered from 000 to 999 decimal.

TYPE: The frame type identifies the type of information currently valid (see table 6-1).

ADDR: For LOCATION and VLOCATION frames, the 16-bit code address in hexadecimal. On LOCATION frames, the system searches the user symbol table and displays a user symbol if it matches the LOCATION address; the display is on the line preceding the LOCATION frame.

```
FRAME   TYPE  ADDR   DATA   INSTRUCTION          P1   P2   P0   TOVF
0000:   LOC   0000H  (80H)  (SJMP       ---)     FFH  FFH  FFH   0
0001:   OPC          80H                         FFH  FFH  FFH   0
0002:   VLO   0001H                              FFH  FFH  FFH   0
0003:   VAL          2EH                         FFH  FFH  FFH   0
0004:   VLO   0001H                              FFH  FFH  FFH   0
0005:   VAL          2EH                         FFH  FFH  FFH   0
0006:   VLO   0002H                              FFH  FFH  FFH   0
0007:   VAL          00H                         FFH  FFH  FFH   0
.START
0008:   LOC   0030H  (43H)  (ORL        ---,#---) FFH  FFH  FFH   0
0009:   OPC          43H                         FFH  FFH  FFH   0
0010:   VLO   0031H                              FFH  FFH  FFH   0
0011:   VAL          90H                         FFH  FFH  FFH   0
0012:   VLO   0032H                              FFH  FFH  FFH   0
0013:   VAL          02H                         FFH  FFH  FFH   0
0014:   VLO   0032H                              FFH  FFH  FFH   0
0015:   VAL          02H                         FFH  FFH  FFH   0
0016:   LOC   0033H  (78H)  (MOV        R0,#---) FFH  FFH  FFH   0
0017:   OPC          78H                         FFH  FFH  FFH   0
0018:   VLO   0034H                              FFH  FFH  FFH   0
0019:   VAL          30H                         FFH  FFH  FFH   0
0020:   LOC   0035H  (7AH)  (MOV        R2,#---) FFH  FFH  FFH   0
0021:   OPC          7AH                         FFH  FFH  FFH   0
0022:   VLO   0036H                              FFH  FFH  FFH   0
0023:   VAL          00H                         FFH  FFH  FFH   0
```

Figure 6-8. FRAME Mode Trace Display

DATA: For OPCODE and VALUE frames, the 8-bit quantity read from trace. On LOCATION frames, the display has the 8-bit quantity read from code memory, in parentheses.

INSTRUCTION: For LOCATION frames, the disassembly for the first byte of the instruction read from code memory, in parentheses. For OPCODE frames, the disassembly for the first byte of the instruction read from trace; this display appears when the OPCODE frame is not preceded by a LOCATION frame and when the quantities read from code memory and trace do not agree. The disassembly shows the opcode mnemonic and any operands that can be determined from the first byte; subsequent operands are shown as dashes (---).

P1, P2, P0: The values of the three I/O ports in hexadecimal. The arrangement of the ports in this display allows you to read P2 and P0 as the high and low bytes (respectively) of an external address.

TOVF: Trace overflow flag (0 = no overflow, 1 = overflow). The trace buffer overflows after the first 1000 frames have been collected.

Verification of Trace in FRAME Mode

In FRAME mode, the value of the opcode read in trace is compared to the corresponding byte in code memory. If they differ, the value from memory is displayed on the line with the LOCATION frame. In INSTRUCTION mode, a warning message is displayed to inform you of the verify failure. The following example shows the result that occurs when code memory is changed after emulation breaks.

```
*A
0000H
*A INC R0
0001H
*GO FROM 0 TILL 0
EMULATION BEGUN
EMULATION TERMINATED, PC=0001H
*CBYTE 0 = 0
*TRA = INSTRUCTIONS
*P ALL
FRAME  LOC    OBJ    INSTRUCTION              P1   P2   P0   TOVF
WARN CC:UNEXPECTED TRACE
0000:  0000H  08     INC    R0                FFH  FFH  FFH   0
*TRA = FRAMES

*P ALL
FRAME  TYPE  ADDR   DATA   INSTRUCTION        P1   P2   P0   TOVF
0000:  LOC   0000H  (00H)  (NOP)              FFH  FFH  FFH    0
0001:  OPC          (08H)  (INC   R0)         FFH  FFH  FFH    0
0002:  VLO   0001H                            FFH  FFH  FFH    0
0003:  VAL           2EH                      FFH  FFH  FFH    0
```

In this example, one instruction is assembled into code memory, then executed (the command GO FROM 0 TILL 0 executes one instruction). After emulation breaks, the memory byte is set to 0 with a CBYTE command. In INSTRUCTION mode the instruction is displayed as emulated, with the warning message UNEXPECTED TRACE. The corresponding display in FRAME mode shows the byte read from memory in parentheses after the address in the LOC frame, along with the opcode that corresponds to this value. In the OPC frame that follows, the byte read from trace is displayed, with its corresponding opcode.

Another reason for the UNEXPECTED TRACE warning is related to interrupts. When an interrupt occurs, two non-executed cycles of trace can be generated while the system puts the return address on the stack. Here is an example of this type of occurrence, and the use of FRAME mode to obtain more detail; in this example, address 001BH is the interrupt vector (on-chip timer 1 interrupt), and 0130H is the return address for the interrupt.

Suppose the INSTRUCTION mode display contains the following sequence:

```
0127:  0134H  50FA    JNC   .LOOP                00H  01H  FFH   0
.LOOP
0135:  0130H  2401    ADD   A,#01H               00H  02H  FFH   0
0139:  0132H  F5A0    MOV   .P2,A                00H  02H  FFH   0
0143:  0134H  50FA    JNC   .LOOP                00H  02H  FFH   0
WARN CC:UNEXPECTED TRACE
.LOOP
0151:  0130H  2424    ADD   A,#24H               00H  03H  FFH   0
0159:  001BH  2140    AJMP  .TIMOUT              00H  03H  FFH   0
.TIMOUT
0167:  0140H  C28E    CLR   .TR1                 00H  03H  FFH   0
0171:  0142H  C0D0    PUSH  .PSW                 00H  03H  FFH   0
0179:  0144H  8890    MOV   .P1,R0               00H  03H  FFH   0
0187:  0146H  08      INC   R0                   00H  03H  FFH   0
0191:  0147H  B85B02  CJNE  R0,#5BH,.MIDOUT      41H  03H  FFH   0
.MIDOUT
0199:  014CH  D0D0    POP   .PSW                 41H  03H  FFH   0
0207:  014EH  D28E    SETB  .TR1                 41H  03H  FFH   0
.ENDOUT
0211:  0150H  32      RETI                       41H  03H  FFH   0
```

(Display continues on next page.)

```
.LOOP
0219:    0130H  2401    ADD    A,#01H              41H  03H  FFH   0
0223:    0132H  F5A0    MOV    .P2,A               41H  03H  FFH   0
```

Note the UNEXPECTED TRACE warning right before address 0130H appears (frame 0151), then the interrupt vector address 01BH (frame 0159H), then the interrupt service routine (addresses 0140H through 0150H, frames 0167 through 0211), then the return to address 0130H. To get more detail on the UNEXPECTED TRACE warning, we use FRAME mode:

```
*OLDEST
*TRACE = FRAME
*MOVE 140
*PRINT 25
FRAME   TYPE  ADDR   DATA   INSTRUCTION          P1   P2   P0   TOVF

0140:   OPC          F5H    (MOV     ---,A)      00H  02H  FFH   0
0141:   VLO   0133H                              00H  02H  FFH   0
0142:   VAL          A0H                         00H  02H  FFH   0
0143:   LOC   0134H  (50H)  (JNC     ---)        00H  02H  FFH   0
0144:   OPC          50H                         00H  03H  FFH   0
0145:   VLO   0135H                              00H  03H  FFH   0
0146:   VAL          FAH                         00H  03H  FFH   0
0147:   VLO   0135H                              00H  03H  FFH   0
0148:   VAL          FAH                         00H  03H  FFH   0
0149:   VLO   0136H                              00H  03H  FFH   0
0150:   VAL          A3H                         00H  03H  FFH   0
.LOOP
0151:   LOC   0130H  (24H)  (ADD     A,#---)     00H  03H  FFH   0
0152:   OPC          24H                         00H  03H  FFH   0
0153:   VLO   0130H                              00H  03H  FFH   0
0154:   VAL          24H                         00H  03H  FFH   0
0155:   VLO   0130H                              00H  03H  FFH   0
0156:   VAL          24H                         00H  03H  FFH   0
0157:   VLO   0130H                              00H  03H  FFH   0
0158:   VAL          24H                         00H  03H  FFH   0
0159:   LOC   001BH  (21H)  (AJMP    ---)        00H  03H  FFH   0
0160:   OPC          21H                         00H  03H  FFH   0
0161:   VLO   001CH                              00H  03H  FFH   0
0162:   VAL          40H                         00H  03H  FFH   0
0163:   VLO   001DH                              00H  03H  FFH   0
0164:   VAL          00H                         00H  03H  FFH   0
*
```

The eight frames (0151 through 0158) represent the incoming interrupt. The return address (0130H) is pushed on the stack. However, when the system compares the trace of this event with the instruction at 0130H, they are different, resulting in the warning message.

[illegible]

CHAPTER 7 PROGRAMMING WITH COMMAND BLOCKS

This chapter contains descriptions of commands that you can use to automate all or part of the emulation session. The chapter begins with the WRITE command; this command allows you to display text of your choosing on the system console. The chapter continues with discussions of the block commands REPEAT, COUNT, and IF. The chapter concludes with a presentation of the macro commands; the macro commands allow you to build a library of diagnostic command suites on disk files.

The WRITE Command

The WRITE command is included in this chapter because it is especially useful for displaying results from within command blocks. The WRITE command allows you to display several items, expressions or text strings, at the console.

The format of the WRITE command is:

$$\text{WRITE} \begin{Bmatrix} \textit{expression} \\ \textit{string} \end{Bmatrix} \left[, \begin{Bmatrix} \textit{expression} \\ \textit{string} \end{Bmatrix} \right] \ldots$$

Examples:

```
*WRITE 'THE VALUE IS', .VALUE
*WRITE 'THE NEW VALUE IS', .VALUE * 3 / 2
```

The WRITE command displays one or more elements on the console.

For example, if the content of the variable with address .TIME is 15, the following command:

```
WRITE DBYTE .TIME,'SECONDS ELAPSED.'
```

produces the following display:

```
0015H SECONDS ELAPSED
```

The WRITE command can display a string, a number, the result of evaluating a numeric expression, or a combination of any of these kinds of elements.

All the elements following WRITE are displayed on one line if possible; if the next element doesn't fit the remaining character space on the line, the system inserts a carriage return/line feed. No spaces are inserted between elements on the same line; if you want spaces before or after a text message, put spaces in the string. A string is displayed just as you enter it. A numeric constant is displayed as entered, using the current SUFFIX when the WRITE command is created. An expression is evaluated, and the result is displayed in the current BASE.

Block Commands

Block commands allow you to enter and execute several command lines as a unit. The block commands are:

IF....END	Conditional execution of a block of commands.
REPEAT....END	Repeat a block of commands.
COUNT....END	Repeat a block of commands a definite number of times.

(The DEFINE MACRO command in the next section is also a block command.)

Entering Block Commands

The block of command lines begins with an identifier line (for example, REPEAT or DEFINE MACRO) and ends with a terminator line (END or EM). The identifier line and all intermediate command lines are ended with a crriage return, but the system interprets these as intermediate carriage returns (intermediate carriage returns are shown as *cr* in the examples.) The carriage return after the END or EM is the final carriage return that terminates the entire command.

At the beginning of intermediate command lines, the system adds a period (.) before the asterisk to show that you are within a block. When blocks are "nested" (that is, when one block contains one or more other blocks), the number of leading periods shows the depth of block nesting.

Initially, the depth is "zero" (no periods preceding the asterisk prompt) this is called the "outermost level". When the system encounters an END line, it terminates the REPEAT, COUNT, or IF command and reverts to the next outer level. When the END line reverts to the outermost level, the command block is executed.

The EM terminator for MACRO definition blocks also reduces the level by one, and this must reach the outermost level (i.e., MACRO definitions cannot be nested within other command blocks). However, the EM terminates the definition only and does not cause any commands to execute.

Boolean Expressions in Block Commands

The IF, COUNT, and REPEAT commands use *boolean expressions* to control what commands in the block are to be executed. A boolean expression can have the same kinds of entries as *expression* (see chapter 1); however, in a boolean expression, only the least significant bit (LSB) of the result is tested. If the LSB = 0, the value of the boolean expression is FALSE; if the LSB = 1, the value of the boolean expression is TRUE.

Boolean expressions in block commands frequently involve *relational expressions* to form true or false conditions. A relational expression uses a relational operator (=, >, <, >=, <=, <>) to compare two values; the result of the comparison is either TRUE (FFFFH) or FALSE (0000H), giving a boolean TRUE or FALSE when the LSB is tested. Here are some examples of relational expressions:

```
PC < 7FFH
RBYTE .SP = 0
.COUNTER >= 10T
```

More examples of boolean expressions are given in the sections on the REPEAT, COUNT, and IF commands later in this chapter.

REPEAT Command

The REPEAT Command has the format:

```
REPEAT cr
 ┌                            ┐
 │command cr                  │
 │WHILE boolean-expression cr │...
 │UNTIL boolean-expression cr │
 └                            ┘
 ENDREPEAT
```

Examples:

```
*REPEAT
.*GO FROM .START TILL BR0
.*ENDREPEAT

*REPEAT
.*WHILE .VAR < .TOTAL
.*STEP
.*PRINT -1
.*ENDR

*REPEAT
.*.COUNTER = .COUNTER + 1

.*WRITE 'COUNTER = ', .COUNTER
.*UNTIL .COUNTER = .MAXIMUM
.*END
```

The REPEAT command consists of the REPEAT keyword, zero or more commands, zero or more exit conditions using WHILE or UNTIL, and the keyword END. Enter each of these elements on its own line of the console display; terminate each input line with an intermediate carriage return (shown as *cr* in the command syntax). You can mix these elements in any order, using any number of each type of element. If no elements are entered, the REPEAT is a "null" command.

After each intermediate carriage return, the system begins the next line with a period (giving an indented appearance), then the asterisk prompt to signal readiness to accept the next element. After the END keyword, enter a final carriage return to begin the sequence of execution. (The final carriage return after END is not shown in the syntax, since all commands terminate with a final carriage return.) The END keyword can be entered as ENDR or ENDREPEAT; the characters after END serve as a form of "comment" to indicate which loop is being terminated.

Each command is executed when it is encountered on each iteration. After the command has been completely executed, the loop proceeds to the next element.

The WHILE and UNTIL keywords introduce exit clauses. More details on WHILE and UNTIL clauses appear just after the discussion of the COUNT command.

COUNT Command

The format of the COUNT command is:

```
COUNT decimal-expression cr
 [command cr                    ]
 [WHILE boolean-expression cr   ] ...
 [UNTIL boolean-expression cr   ]
 ENDCOUNT
```

Examples:

```
*COUNT 10
.*GO FROM .START TILL BR0
.*ENDCOUNT

*COUNT .TESTTIMES + 50
.*WHILE .VAR < .TOTAL
.*STEP
.*PRINT -1
.*ENDC
```

```
*COUNT .COUNTER
.*WRITE 'COUNTER = ', COUNTER
.*COUNTER = .COUNTER + 1
.*END
```

Like REPEAT, the COUNT command sets up a loop. In addition to the WHILE and UNTIL exit clauses (see next section), the COUNT command includes a loop counter that terminates the loop if no exit condition is met before the counter runs out.

The *decimal-expression* after COUNT controls the maximum number of iterations to be performed. If a numeric constant is used (for example, COUNT 10 or COUNT .START + 50), the system interprets it in implicit decimal radix; in other words, any number entered after COUNT without an explicit radix is interpreted as a decimal number.

If the entry after COUNT is an arithmetic expression, it is evaluated to give the number of iterations. The COUNT expression is evaluated *once,* before any loop elements are encountered. It is not evaluated again on any iteration. The COUNT expression uses the values of any references it contains as they stand at the time of evaluation. For example, consider the following command sequence:

```
*DEFINE .XX = 2
*COUNT .XX
.* .XX = .XX + 1
.* .XX
.*ENDC
```

This loop goes through *two* iterations, although .XX has value 4 when the loop terminates.

Halting REPEAT and COUNT Loops

USER ABORT

To halt execution of a REPEAT or COUNT loop, press the ESC key at the console. The command currently executing halts where it happens to be; if you are emulating, the current instruction is completed before the break.

WHILE and UNTIL Clauses

WHILE and UNTIL clauses are used to halt the execution of REPEAT and COUNT loops. A WHILE clause halts a loop when its condition (boolean expression) is FALSE; an UNTIL clause halts a loop when its condition is TRUE.

In both the WHILE and UNTIL clauses, the boolean-expression is evaluated each time the clause is encountered; that is, once per iteration. Evaluation at each iteration involves looking up the values of any references in the expression. Thus, the result can change with each evaluation.

The choice of WHILE or UNTIL is usually a matter of convenience — there is always a way to convert one into the other. For example, "WHILE bool-expr" is equivalent to "UNTIL NOT (bool-expr)".

An exit can be made only when a condition is *tested,* not when it occurs. To cause an exit, the test must be placed at the point in the loop where the condition occurs. For example, consider the following command sequence:

```
*PC = .START
*REPEAT
.*UNTIL PC = 1000H
.*STEP
.*ENDR
```

In this command the condition PC = 1000H is tested after every STEP. If the sequence of STEPs reaches PC = 1000H as the next instruction, the loop will terminate. By contrast, consider this example:

```
*PC = .START
*REPEAT
.*UNTIL PC = 1000H
.*COUNT 10
..*STEP
..*ENDCOUNT
.*ENDR
```

In the second example, the condition PC = 1000H is tested after every *ten* STEPs. The loop exits only if PC = 1000H occurs at the *end* of some group of ten instructions. If PC = 1000H occurs *during* one of the groups of ten STEPs, the loop does not terminate because that condition is changed by subsequent STEPs before the test can be made.

If the command has more than one exit clause, each exit clause is tested when it is encountered. If the result at the moment of the test causes an exit, the loop terminates; otherwise, the loop proceeds to the next element.

In a COUNT loop, the loop terminates when the number of iterations given by the COUNT expression has been performed *or* when an exit condition is tested and causes the loop to terminate, *whichever comes first.*

IF Command

The IF command has the format:

```
IF boolean-expression [THEN] cr
  [command cr] ...
  [ORIF boolean-expression cr ]...
  [  [command cr] ...         ]
  [ELSE cr          ]
  [  [command cr] ...]
   ENDIF
```

Examples:

```
*IF PC < 0100H THEN
.*GO TILL BR0
.*ENDIF

*IF PC >=0 AND PC<1000H
.*GO TILL BR0
.*END

*IF PC>=0 AND PC<0100H THEN
.*GO TILL BR0
.*ORIF PC<0200H
.*GO TILL BR1
.*ORIF PC < 0300H
.*GO TILL SY0
.*ELSE
.*GO FOREVER
.*ENDIF
```

The IF command permits conditional execution of a command sequence. The command must have the IF clause; the ORIF and ELSE clauses are optional. The command can include as many ORIF clauses as desired. The IF and ORIF clauses each contain a single condition (boolean expression). Any clause can contain none, one, or more commands.

The system examines each boolean expression in turn, clause by clause, looking for the first TRUE condition. If a TRUE condition is found, the commands in that clause are executed and the IF command terminates. If none of the conditions is TRUE, the commands in the ELSE clause are executed and the IF command terminates. If the ELSE clause is omitted and no condition is TRUE, the IF command terminates with no commands executed.

The ENDIF keyword is required to close off the IF command; it can be abbreviated to END.

Nesting Command Blocks

The REPEAT, COUNT, and IF commands can be nested to provide a variety of control structures.

Each nested compound command must have its own END keyword. When entering a nested command sequence, you may wish to use the keywords ENDR, ENDC, and ENDIF, to help you keep straight which command you intend to close off. The system does not check nesting levels at entry, and if an END is omitted, the resulting error makes it necessary to enter the entire command again.

Each nested REPEAT or COUNT command can contain its own exit clauses (WHILE or UNTIL). Each such exit clause can terminate the loop that contains it, but has no effect on any outer loops or commands.

Macro Commands

A macro is a named block of commands. When a block of commands is defined as a macro, it is stored in a "macro table" so that it can be executed more than once without having to enter the commands each time. The macro commands described in this chapter allow you to perform the following functions:

- Define a macro, specifying the macro name, the command block, and any formal parameters (points where text can be filled in at the time of the macro call).
- Invoke (call) a macro by name, giving actual parameters to fill in the blank fields in the macro definition, and begin the execution of the command block.
- Display the text of any macro as it was defined.
- Display the names of all macros currently defined.
- Remove one or more macros.
- Save one or more macro definitions on an ISIS-II file.
- Bring one or more macro definitions (or other commands) in from a file for use in the current test sequence.

Defining and Invoking Macros

To define a macro, the format is:

DEFINE :*macro-name cr*
 [*command cr*] ...
 EM

Once a macro has been defined, you can invoke (call) the macro as often as desired. The format is:

```
:macro-name   actual-parameter-list
```

Example:

```
*DEFINE :LOOK                              ;Definition
.*WRITE 'VAR1 = ', DBYTE .VAR1
.*WRITE 'VAR2 = ', DBYTE .VAR2
.*EM
*
*:LOOK                                     ;Invocation
VAR1 = 00H
VAR2 = 00H
```

The macro definition command causes the macro name and the block of commands to be stored in a table of macro definitions.

NOTE

A macro definition may not be placed within any other compound command block.

A *macro-name* must begin with a letter, or with one of the characters "?" or "@". The characters after the first character can be letters, "?", "@", or numeric digits. The macro name must not duplicate a previously-defined macro name.

A macro definition may not appear with any other command (REPEAT, COUNT, IF, or another macro definition). The command block in the macro definition can include any command except another DEFINE MACRO command or a REMOVE MACRO command.

The macro name in the macro invocation must be the name of a previously-defined macro. The form of *actual-parameter-list* is discussed later in this chapter.

Here is a simple macro definition:

```
*DEF :GOER
.*REPEAT
..*GO FROM .START TILL BR1
..*ENDR
.*EM
```

To invoke this macro and cause its command block to begin executing, enter the macro name preceded by a colon (:). For example:

```
*:GOER
```

A macro definition can include commands that define user symbols. Macros that include user definitions can be used for initialization purposes. You should use some caution in placing user definitions within macro definitions, however, since multiple definitions cause errors.

NOTE

When you are using MACRO commands, the emulator system opens a file named MAC.TMP to hold the macro definitions. By default, the drive used for this file is the one named in the invocation command (see chapter 2).

A macro definition can include calls to other macros, but a macro cannot call itself recursively. A macro that calls itself in its command block expands indefinitely when the outer macro is called, without ever executing any commands (press ESC to terminate such an infinite expansion). Any macros called from within a macro must have been defined when the calling macro is invoked. Macro calls can be nested; i.e., one macro calls another, which calls another, and so on. The level of nesting is limited only by the memory space, as noted below.

NOTE

The emulator uses memory in the development system for its workspace. The workspace contains symbol tables and space for processing commands, including macro expansions. It grows dynamically to accommodate larger symbol tables and command structures. As workspace grows, it always reflects the largest space required for command processing during each test session; it does not "shrink" dynamically to accommodate smaller commands. If more macro space is required, you must have fewer user symbols.

When a macro is called as an outer level command the following operations occur:

- The default SUFFIX is saved in case a new default is set inside the macro.
- The text of each actual parameter in the call is substituted for the corresponding formal parameter in the definition.
- The expanded command block is executed if all commands are valid as expanded.
- When the last command has finished, the former SUFFIX is restored.
- The macro exits. Control returns to the console (asterisk prompt).

The next several sections provide details on these operations.

Formal and Actual Parameters

A formal parameter marks a place in a macro definition where variable text can be "filled in" when the macro is called. A formal parameter can represent part of a token or a field of one or more tokens. A macro definition can contain up to ten formal parameters. A formal parameter has the form:

%*n*

where *n* is a decimal digit, 0 to 9.

Formal parameters can appear in the macro definition in any order, and each one can appear any number of times. In most cases, the formal parameters form a complete numeric sequence with %0 as the lowest numbered parameters (even if %0 is not the first parameter to appear). However, one or more parameters can be omitted from the sequence; the effect of omitting a formal parameter from the sequence is to ignore the actual parameter in the call that corresponds to the omitted formal parameter.

The macro call can contain as many actual parameters as desired. Enter multiple parameters as a list, with entries separated by commas. The first actual parameter in the list is substituted at all points that %0 appears in the macro definition; the second parameter substitutes for %1, and so on.

An actual parameter can be "null", causing the system to substitute a null for the formal parameter to which it corresponds. You can pass a null parameter to a macro in two ways:

- Enter no actual parameter between consecutive commas.
- Omit one or more parameters from the end of the list.

If too few actual parameters are entered, the system supplies nulls for the extra formal parameters. If too many actual parameters are entered, the extra actual parameters are ignored. However, if more than ten actual parameters are entered, an error occurs and the call is aborted.

If any actual parameter contains a carriage return, a comma, or an apostrophe, the entire parameter must be enclosed in apostrophes to identify it as a single actual parameter. In other words, parameters with these characters must be entered as *strings*. An apostrophe within a string is entered as (").

Here are some examples to demonstrate the use of formal and actual parameters.:

Example 1:

```
*DEF :MEM
.*%0BYTE %1
.*EM
```

In the call to this macro, parameter %0 can become "C", "D", "R", "P", or "X". Parameter %1 can be any valid address or partition. Examples of calls to this macro:

Macro call	Expansion
:MEM X,20H	XBYTE 20H
:MEM D,20H LEN 5H	DBYTE 20H LEN 5H

Example 2:

```
*DEF :RPT
.*REPEAT
..*%0
..*T1
..*%2
..*%3
..*%4
..*%5
..*%6
..*%7
..*%8
..*%9
..*END
.*EM
```

Macro RPT can accept up to ten commands to be repeated. For example:

```
:RPT GO TILL BR0, PRINT -1, REGISTERS, GO TILL BR1, PRINT -10
```

If fewer than ten commands are given, as in the example above, the extra formal parameters are ignored (treated as nulls).

Details on Macro Expansion

The syntax and semantics of commands in a macro block are ignored at the point of definition; they are not determined until invocation, and may be different on each invocation through the use of formal parameters.

When a macro is called, its definition is expanded by adding the text of any actual parameters in the call at the points indicated by formal parameters in the definition. If the expanded macro contains any calls to other macros, the text of any such macro is also expanded, forming in effect one overall block of commands. Expansion continues until the last EM is reached. If the expansion results in a set of complete, valid commands, the commands are executed. An error results if any command is incomplete or invalid after expansion.

A macro invoked in a REPEAT, COUNT, or IF command is expanded immediately after the macro call command is entered. Thus, a macro called in a REPEAT or COUNT command is expanded only once, and a macro called in an IF command is expanded whether the condition in the IF or ORIF clause that contains the macro call is TRUE or FALSE.

Initially, macro expansion is "silent"; that is, the expansion is not displayed on the console. To have the expansion of macros displayed before the macros are executed, the command is:

```
*ENABLE EXPANSION
```

To have macro expansion "silent" again, the command is:

```
*DISABLE EXPANSION
```

Macro Table Commands

The macro table contains the name and text of all macros currently defined. The text is stored as it is defined, and does not contain any expansions.

To display the name of all macros in the table, the command is:

```
*DIR
```

For example, assuming the macros in this chapter have been defined:

```
*DIR
LOOK
GOER
MEM
RPT
```

To display the name and definition of one, several, or all macros in the table, the format is:

MACRO [:*macro-name* [, :*macro-name*] ...]

Examples:

```
*MACRO              ;Displays all macro definitions.
*MAC :MEM           ;Displays text of macro MEM only,
*MAC :GOER,:RPT     ;Displays text of macros in the list.
```

To remove one or more macro definitions from the table, the format is:

$$\text{REMOVE}\left\{\begin{array}{l}\text{MACRO}\\ :\textit{macro-name}\ [,\ :\textit{macro-name}]\ \ldots\end{array}\right\}$$

Examples:

```
*REMOVE MACRO    ;Removes all macros.
*REMOVE :LOOK    ;Removes LOOK only.
*REM :RPT,:LOOK  ;Removes all macros in the list.
```

The REMOVE macros command may not appear within any block command or macro definition.

Saving Macros

The PUT command causes one or more macro definitions to be copied from the temporary file to a "permanent" file. The format is:

```
                       {MACRO                              }
PUT :drive:filename    {                                   }
                       {:macro-name [, :macro-name] ...    }
```

Examples:

```
*PUT :F1:UTIL.MAC MACRO
*PUT :F1:INIT1.MAC :GOER, :LOOK
```

If any macro names are entered, those macro definitions are saved. If MACRO is used, all macros in the macro table are saved. The definitions in the macro table are not affected by the operation.

The file containing the saved macro can later be edited or brought into another session with the INCLUDE command, discussed below.

If the named file does not exist, it is created by the PUT command. If the file does exist on the diskette, the file is opened for input and the macros in the list are written on the file, destroying the previous contents of that file.

Including Commands From Files

The macro definition files created with the PUT command can be read into the ICE temporary macro table with the INCLUDE command. The format is:

```
INCLUDE :drive:filename
```

Examples:

```
*INCLUDE :F1:UTIL.MAC
*INCLUDE :F1:INIT.MAC
```

For example, suppose we had defined a macro INIT as follows:

```
*DEFINE :INIT
.*MAP = 0,2000H
.*LOAD :F1:PROG1
.*EM
```

Suppose further that we had saved this macro definition with a PUT command as follows:

```
*PUT :F1:INIT.MAC :INIT
```

Then, in a subsequent session we bring this macro definition from the file with the command:

```
*INCLUDE :F1:INIT.MAC
```

The result of the INCLUDE is to read in the definition of INIT as given earlier. The system issues a prompt at the beginning of each input line (i.e., the prompts are not saved on the file).

Although the PUT command can be used only to save macro definition, the INCLUDE command can refer to a file containing any valid emulator commands. For example, the file could contain macro invocations as well as definitions. To have an INCLUDE file contain commands other than macro definitions, edit the file with the ISIS text editor. Each command that you "edit in" should start at the left margin (do not "edit in" the prompt), and should terminate with a carriage return.
The INCLUDE command can be nested within any other command, including a macro definition.

APPENDIX A INSTALLATION AND CHECKOUT PROCEDURES

This appendix contains procedures for installing the emulator hardware in the development system, for performing the hardware confidence test, for connecting the user cable to the Crystal Power Accessory for stand-alone operation, for connecting the user cable to the user hardware system, and for connecting the external signal cable assembly to the buffer box. Refer to chapter 2 for the procedure for invoking the emulator software from diskette.

Required and Optional Equipment

The emulator requires the following minimum equipment for installation and operation:

Intellec Model 800 or Series II development system with 64K of RAM and available slots for two adjacent circuit boards.

CRT and keyboard for console display and command entry.

One diskette drive, single-density or double density.

Although one diskette drive is sufficient, a two-drive (or more) system provides considerable convenience in operating the emulator, as discussed in chapter 2. In addition, a printer may be added where hard-copy output is required by the application.

The following items are the parts of the emulator system contained in the shipping carton:

Controller board.

Trace board.

Dual auxiliary connector kit, containing dual auxiliary connectors for Model 800 (1000515) and Series II (1000751).

User cable and buffer box assembly.

Crystal power accessory (CPA).

Sync cable assembly.

Emulator software diskettes, one for single-density and one for double density operation.

NOTE

Make sure your emulator software diskette has a write-enable tab installed before attempting to invoke the software.

Literature kit, containing the following publications:

An introductory Dear Customer letter
ICE-51™ Operating Instructions
ICE-51™ Command Dictionary
Getting Started With the ICE-51™ Emulator

Hardware Installation Procedures

This procedure is recommended for installing the emulator hardware in the chassis of the development system. The procedure contains variations for installation in (a) the model 800, (b) the Series II main chassis, and (c) the Series II expander chassis.

WARNING

The chassis of the development system presents electrical power on several connectors. Installation of the emulator boards in the development system chassis should be performed by qualified persons only, and only with the power cords completely disconnected from the development system.

1. **Controller board device code.** Figure A-1 shows the locations and diagrams of jumpers J1, J2, J3, and J4 on the Controller board. Device code 32H is the required device code for a single ICE-51 system. Verify that shorting plugs are installed as follows to select device code 32H:

   J1 — No shorting plugs.

   J2 — Shorting plug in position 2 (connects pin labeled 2 to pin labeled C).

   J3, J4 — Shorting pin installed across the two pins of each jumper.

2. **Disconnect power cords.** Disconnect the power cord from the development system, and turn power off to diskette drives and other peripherals.

3. **Remove access panel.** Remove the chassis access panel on the development system.

   a. On the model 800, remove the top cover; it is secured by four half-turn fasteners. A blade screwdriver is required.

   b. On the Series II main chassis, remove the front panel below the screen. It is secured by two half-turn fasteners. A blade screwdriver is required.

   c. On the Series II expander chassis, remove the front panel. Like the main chassis, it is secured by two half-turn fasteners. A blade screwdriver is required.

4. **Prepare chassis slots.** Remove or rearrange peripheral controller boards in the chassis to leave two adjacent slots.

   a. On the model 800, the Controller board must reside in an odd-numbered slot. The Trace board can reside in the adjacent even-numbered slot on either side of the Controller board. If you follow the exact procedure given below, the Trace board ends up in the even-numbered slot to the left of the Controller board (next lower numbered slot).

   b. On the Series II main chassis, move peripheral controller boards to free up the two central slots in the set of six slots in the chassis.

   c. On the Series II expander chassis, move peripheral controller boards to free up the two central slots in the set of four slots in the chassis.

5. **Insert Controller board in dual auxiliary connector.** Refer to figure A-2. Insert the controller board so that the component side faces toward the empty slot on the connector. For the model 800, the connector is labeled "DUAL AUX CONN"; for the Series II, the connector is labeled "SBC DUAL AUX CONN".

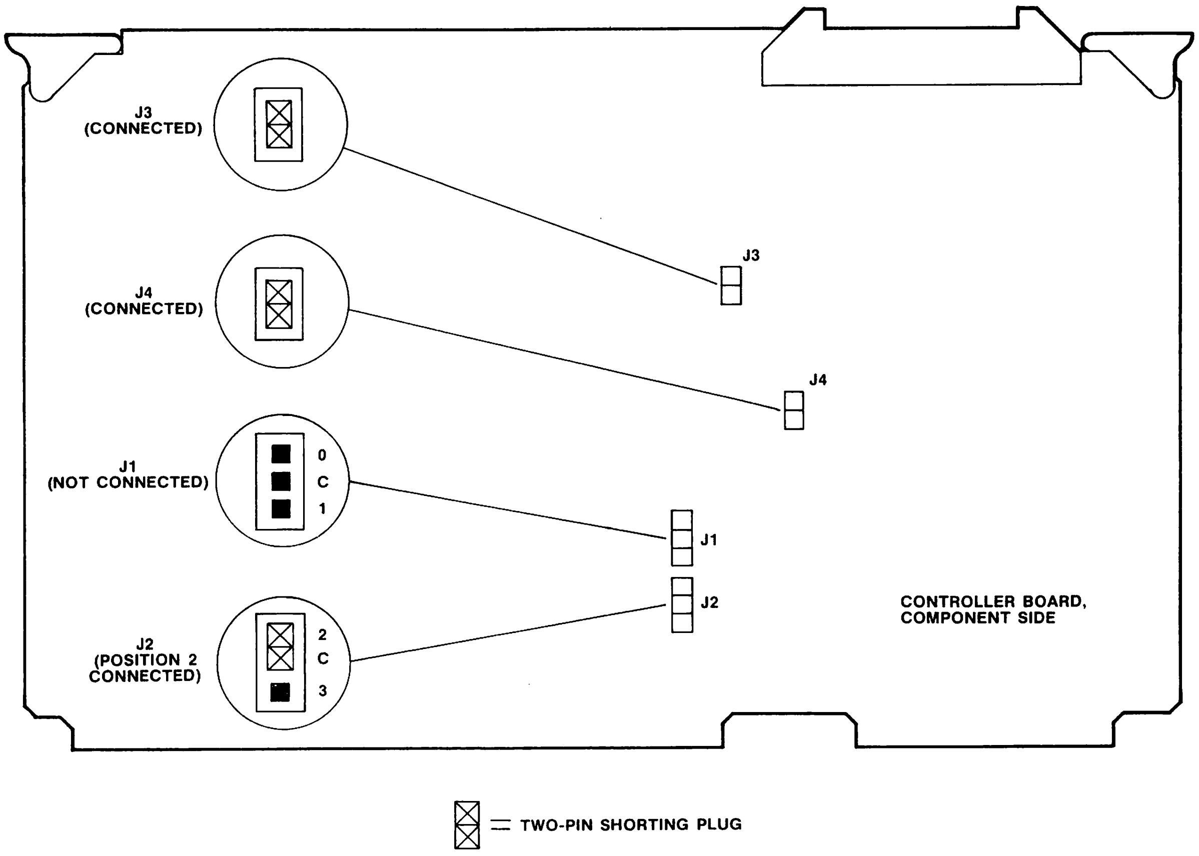


Figure A-1. Device Code Jumpers

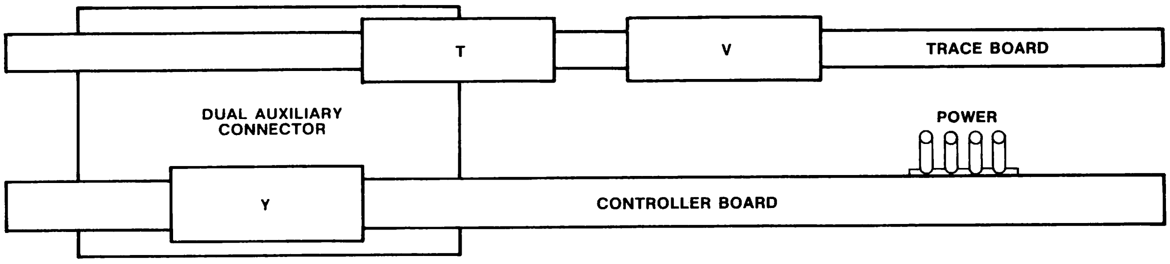


Figure A-2. Emulator Boards in Dual Auxiliary Connector

6. **Insert Controller board/connector assembly in chassis slot.**

   a. For the model 800, the Controller board must go in an odd-numbered slot. The component side of the board faces toward the power supply on the left end of the card cage.

   b,c For the Series II main chassis and expander chassis, insert the Controller board in the lower of the two adjacent slots, component side up.

7. **Insert Trace board in chassis.** The component side faces the same way as the component side on the Controller board. Press firmly on both sides to seat the board in the main backplane and in the dual auxiliary connector. Check that the Controller board has remained firmly seated.

NOTE

To assist in aligning the boards with the chassis slots, you may find it helpful to lift the boards gently in the center to relieve any flex.

8. **Route cable assembly from buffer box into chassis.** The assembly contains a power cable and three ribbon cables in a protective tubing.

   a. On the model 800, locate the connector panel at the top rear of the chassis. Remove the four screws that attach the panel to the frame. Lift the connector panel away from the frame, and insert the cable assembly through the slot. The corrugated sides of the ribbon cables should be facing down. Do not replace the screws until the cables have been connected (see next step).

   b,c. On the Series II main chassis and expander chassis, locate the cable relief slot and restraint bar at the right side of the chassis. Insert the cable assembly from the right, behind the restraint. The corrugated sides of the ribbon cables should be facing toward the chassis.

9. **Connect buffer box cables.** Figure A-3 shows the suggested cable routing and order of connection. Observe the pin 1 indicators on connectors.

   - Connect the ribbon cable marked V to the connector marked V on the Trace board. The ribbon cable descends from the connection. Fold the cable as needed to make the turn.
   - Connect the ribbon cable marked T to the connector marked T on the Trace board.
   - Connect the ribbon cable marked Y to the connector marked Y on the Controller board. If necessary, lift cable T out of the way while inserting cable Y; ensure T remains seated.
   - Connect the power cord to the four-pin connector on the Controller board. The flange on the front edge of the plug points up.

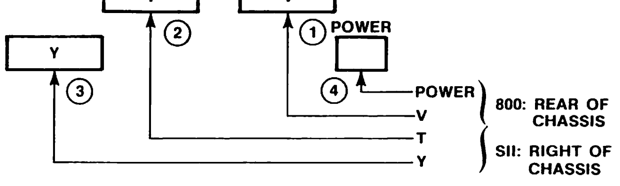


Figure A-3. Ribbon Cable Routing Diagram

10. **Route cables out of chassis.**

    a. On the model 800, fold the cables to lie flat when the top cover is replaced. Ensure the power cord is at the edge of the protective tube, not on top of the flat cables. Remove slack in the cable assembly to the rear, then hold the rear connector panel in place and replace the four screws. The cable assembly should fit snugly in the slot.

    b,c. On the Series II main chassis and expander chassis, fold the cables to lie flat when the front cover is replaced. Ensure the power cord is at the edge of the protective tube, not on top of the flat cables. Remove slack in the cable assembly to the right.

11. **Replace chassis access panel.** (See step 3.)
12. **Ensure proper grounding.** Locate the ground access points on the user plug (figure A-4). To enhance noise immunity, connect one or more of these to the user system electronic ground.

    For additional grounding, plug the external signal cable assembly into the slot on the buffer box (used for CPA or external cables), then connect the black cables to the system electronic ground. Refer to external signal installation procedure later in this chapter.

13. **Connect power cord to development system.**

This completes the chassis installation.

Installing Crystal Power Accessory

To install Crystal Power Accessory (CPA):

1. Remove power from the emulator system.
2. Leave pin protector on user plug (figure A-5).
3. Insert user plug in 40-pin socket on the CPA, aligning the pin 1 markers (figure A-6).

NOTE

As shipped, the emulator has the crystal power accessory installed at the end of the user cable. When this is the case, instead of steps 2 and 3 simply check that the pin protector is in place and that the pin 1 indicators on the user plug, pin protector, and CPA are aligned.

4. Insert the edge connector of the CPA in the socket on the buffer box; the CPA may be inserted in either direction.

Hardware Confidence Test

To perform the confidence test after installation or to check on a potential hardware error:

1. Complete hardware chassis installation as described earlier in this chapter.
2. Install Crystal Power Accessory as described in the previous section.
3. Turn power on to development system, diskette drives, and peripherals.
4. Insert a system diskette in drive 0, and the emulator software diskette (single or double density as appropriate for your system) in drive 1. In a one-drive system, copy the emulator software to a system disk (see "Invocation", chapter 2).

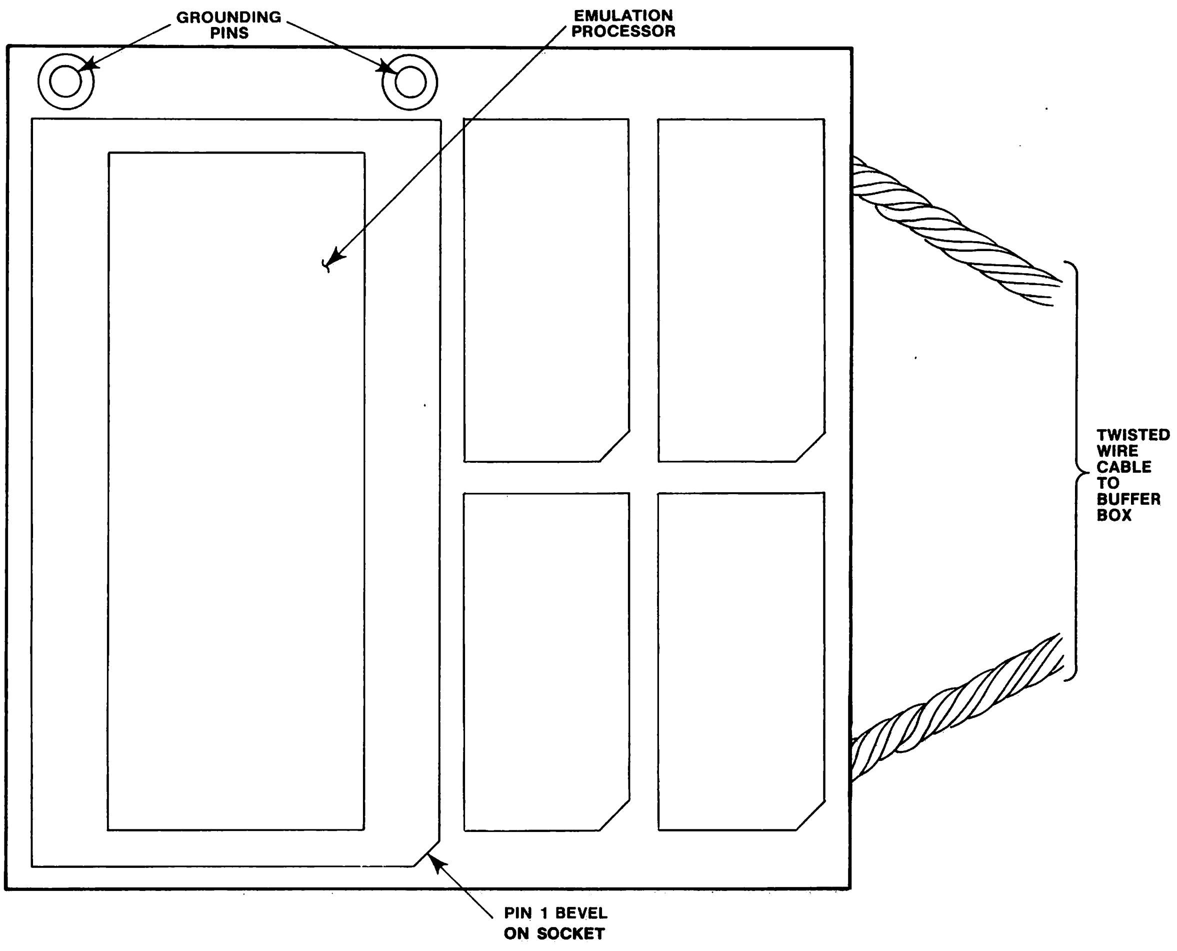


Figure A-4. User Plug, Top View

5. Boot the ISIS-II system, using the procedure in your development system operating manual.
6. Use the ISIS command DIR 1 to verify the emulator disk has files:

   ```
   CONF
   CON51
   CON51.OV1
   CON51.OV2
   CON51.OV3
   ```
7. Enter the following ISIS call:

   ```
   :F1:CONF
   ```

   One-drive system users can omit the :F1:.
8. In response to CONF command, ISIS-II loads and executes Confidence Test Manager program, which displays the following message:

   ```
   ISIS-II CONF, Vx.y
   ```

NOTE

In this and other system messages, Vx.y indicates software version number.

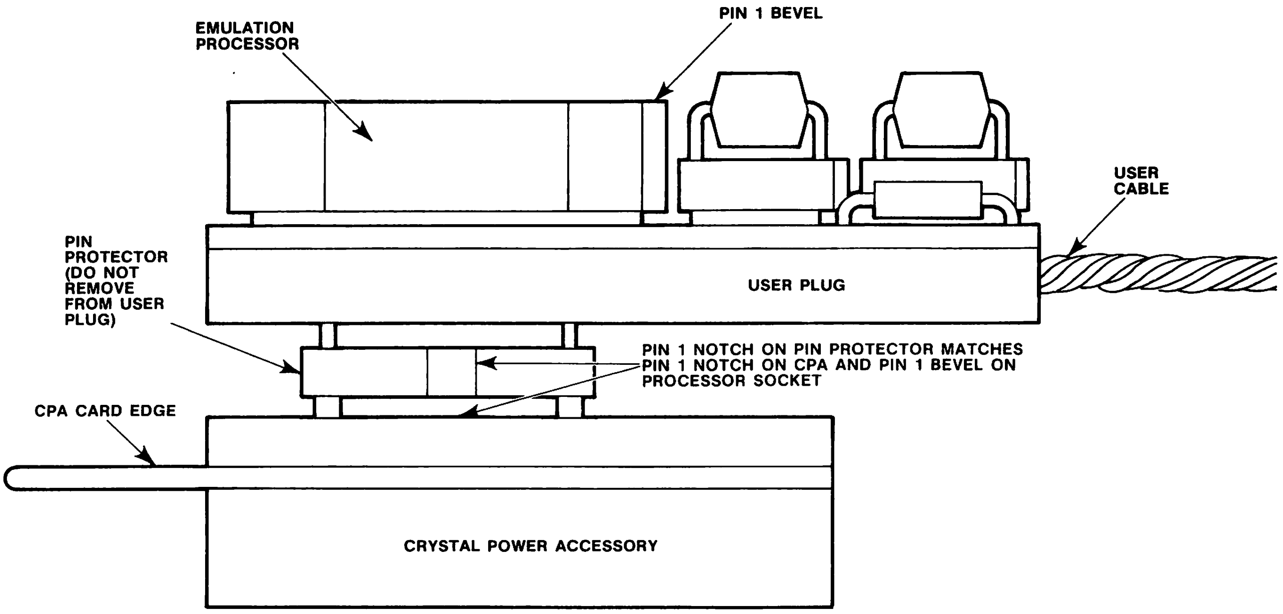

EMULATION PROCESSOR
PIN 1 BEVEL
USER CABLE
PIN PROTECTOR (DO NOT REMOVE FROM USER PLUG)
USER PLUG
PIN 1 NOTCH ON PIN PROTECTOR MATCHES PIN 1 NOTCH ON CPA AND PIN 1 BEVEL ON PROCESSOR SOCKET
CPA CARD EDGE
CRYSTAL POWER ACCESSORY

Figure A-5. User Plug Assembly, Side View

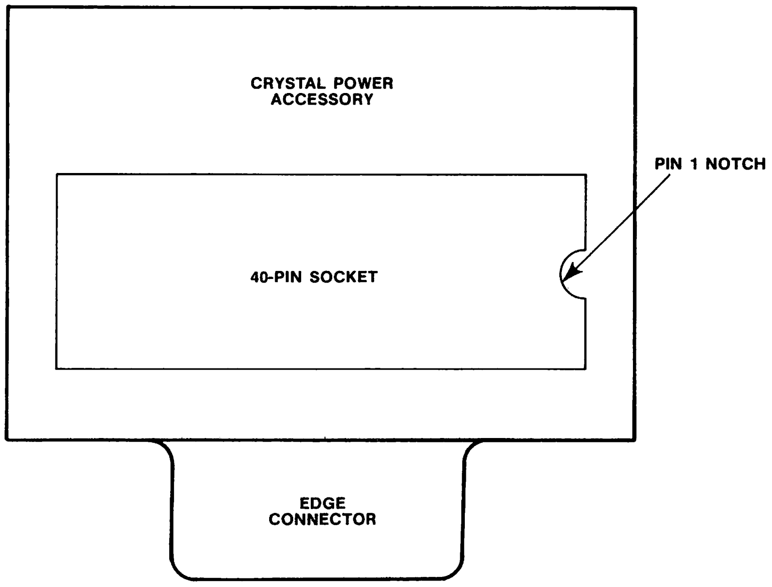


Figure A-6. CPA Socket Diagram

Following this message, the Confidence Test Manager displays asterisk (*) as prompt for next command.

CAUTION

Do not remove confidence test diskette from its drive while testing is in progress.

9. Load confidence test routines by entering the INITIALIZE command, followed by the appropriate disk drive designation, and the keyword CON51. For example:

```
*INITIALIZE :F1:CON51
```

In response, Confidence Test Manager displays following message:

```
ICE-51 CONFIDENCE TEST, Vx.y
USER RETURN
```

This message is followed by an asterisk prompt for another command.

10. To produce an optional copy of the confidence test input/output, enter the LIST command as follows, specifying a device such as the line printer (device name :LP:) or a diskette file as the destination of this copy. For instance:

```
*LIST :LP:
```

11. Begin the confidence test series by entering the TEST command:

```
*TEST
```

As the Confidence Test Manager concludes each individual confidence test, it displays a message indicating whether the emulator hardware passed or failed the test. For example, following the first test in the series, one of the following two messages appears:

```
RESET AND INVOKE ICE-51 "PASSED"
RESET AND INVOKE ICE-51 "FAILED"     <= = = =
```

If all displayed test messages denote "PASSED," the hardware has been installed correctly and is operating properly. But, if any displayed test message denotes "FAILED," the hardware is not operating properly; inspect the hardware for improper installation, reinstall it correctly, and rerun the confidence test to verify the installation. For additional assistance in correcting errors, contact your Intel Service Representative (appendix C).

12. Terminate the Confidence Test Manager by entering this command:

```
*EXIT
```

Connecting the Emulator to the User System

To connect the emulator to the user hardware:

1. Turn power off to the user prototype system.

CAUTION

Any wrong connection of the user plug can damage equipment. Check proper orientation before applying power.

2. Remove the Crystal Power Accessory from the end of the user cable (see CPA installation procedure earlier in this chapter). Leave the pin protector attached (figure A-5).
3. Insert the user plug in the 8051 socket on the user system, aligning pin 1 on the socket with the pin 1 indicator on the user plug (figure A-5).

NOTE

In its initialization phase, the emulation expects power and clock to be present at the user plug. If power and clock are present, however, and the emulator has not been invoked since the development system was powered up, the emulation processor on the user plug executes 'random' instructions until the emulator is invoked. In some applications, power and clock should not be applied until the emulator has been invoked. After the invocation, use RESET ICE (chapter 2) to permit the emulator to complete its initialization.

Installing Sync Line Cables

To install the sync line cable assembly:

1. Turn power off to the system.
2. Remove CPA if installed (see CPA installation procedure earlier in this appendix).
3. Plug edge connector of cable assembly into the same slot used by the CPA. Align the color-coded wires with the markings on the buffer box (blue is SY0, yellow is SY1, black is ground).

APPENDIX B ERROR MESSAGES AND WARNINGS

This appendix contains a list of error and warning messages produced by the emulator system. An error message has the following format:

```
ERR XX:xxxxxxxxxxxxxxxxxxxxxxxxxx
```

Where XX is the number of the error in hexadecimal, and the remainder is a short description of the error.

When an error occurs, the command that was executing is terminated. In some cases (e.g., syntax error), the command is never executed. In other cases (e.g., memory overflow), the command may have executed partially before the error occurred. The discussion of each error includes suggestions for corrective action, where possible.

Warnings have the following format:

```
WARN XX:xxxxxxxxxxxxxxxxxxxxxxxxxx
```

Where XX is the number of the warning in hexadecimal, and the remainder is a brief description of the warning. Warnings inform you of possible side effects of the command that was just executed, but do not abort any commands.

Overview of Corrective Actions

This section briefly reviews corrective actions that apply to more than one error condition.

RESET ICE and ERROR Commands

The command:

```
RESET ICE
```

resets the emulator hardware. The emulator map is reset to 0000H, 1000H. No other emulator registers accessible to the user are affected. RESET ICE is used to help the emulator hardware to recover from hardware errors. The discussions of the errors refer to this command where it applies.

The command:

```
ERROR
```

causes the emulator to display additional error information. If the most recent command resulted in a hardware error (errors 0 through 7FH, producing a red light signal at the buffer box), the ERROR command displays the results so that details on that error may be passed on to field service personnel. The version number and date code of the executing software and firmware are displayed at the beginning of the test.

Confidence Test

After any error, especially errors 0 through 7FH, from which the system does not recover after RESET ICE, terminate the emulation session with the EXIT command and run the confidence test. Appendix A describes the confidence test procedure.

Service Assistance

To obtain service assistance, refer to the service information in appendix C.

Incomplete and Undetermined Errors

In rare instances, an error number may appear without a description; for instance:

```
ERR 80:?
```

This display indicates that the error normally associated with this number was in fact detected, but some other problem prevented the emulator from printing the descriptive text. If this kind of display occurs, check that your emulator software diskette contains the file ICE51.OVE.

In other instances, the system may generate an error number for which the cause and corrective action are not predictable; in this event, the system displays:

```
ERR XX:UNDETERMINED ERROR
```

Where XX is the number of the error in hexadecimal. Substitute a fresh software diskette, or try the confidence test.

Error Messages and Warnings

Here is a list of the error messages and warnings in hexadecimal order, including brief discussions of each error and suggestions for corrective action.

NOTE

Corrective actions are numbered for emphasis and (in some cases) to indicate the seriousness of the error condition they are trying to correct. Within a given error, try corrective action 1; if it fails to correct the problem, proceed to higher numbered actions in sequence.

```
ERR 12:PAR BLK DVC CODE ERROR
ERR 13:PAR BLK FORMAT ERROR
ERR 14:NON-ZERO COMMAND ACK
ERR 15:NON-ZERO DONE FLAG
ERR 16:DVC CD FORMAT ERROR
ERR 17:DVC NOT IN DVC CD TABLE
```

Errors 12 through 17 arise through failure of communication between the software and hardware. (1) Use RESET ICE to attempt recovery. (2) Check to see that the emulator hardware installation has been performed correctly. (3) Run the confidence test. (4) Verify that the development system hardware operation is correct.

```
ERR 20:ILLEGAL INPUT COMMAND
```

The emulator software is not giving a correct command to the hardware. (1) Use the EXIT command to end the session, then invoke the emulator again. (2) Substitute a fresh software diskette. (3) Run the confidence test.

```
ERR 21:COMMAND NOT ALLOWED NOW
ERR 22:RSLTS LENGTH INADEQUATE
ERR 23:COMMAND FORMAT ERROR
```

Errors 21, 22, and 23 indicate a communication problem between the software and the hardware. A software error is the probable cause. (1) Use RESET ICE to attempt recovery. (2) EXIT, then invoke the emulator again. (3) Substitute a fresh software diskette. (4) Run the confidence test.

ERR 30:PGM MEMORY FAILURE

After command to change user program memory (in buffer box), the data read back did not agree with the data written. (1) Use RESET ICE to attempt recovery. (2) EXIT and run the confidence test.

ERR 31:DATA MEMORY FAILURE

After command that changed external data memory, data read back did not agree with data written. (1) Use RESET ICE to attempt recovery. (2) Check user system for memory operation.

ERR 38:CONTROL MEMORY FAILURE

During hardware reset, data read back from controller memory did not agree with data written. (1) EXIT and invoke the emulator again. (2) Run the confidence test.

ERR 39:FIRMWARE CHECKSUM ERR

During a load of downloadable firmware (at invocation of the emulator or after RESET ICE), a checksum error was encountered. Possibly the file ICE51.OVS is missing or damaged on the emulator software diskette. (1) EXIT and invoke the emulator again. (2) Substitute a fresh software diskette. (3) Run the confidence test.

ERR 40:NO USER CLOCK

No clock signal is present at the emulator plug. (1) If plug is connected to the user system, ensure that plug is installed correctly and that correct crystal is present; if the plug is connected to the Crystal Power accessory (CPA), ensure that plug is installed correctly. (2) Use RESET ICE to attempt recovery. (3) Check the user hardware. (4) Run the confidence test.

ERR 41:NO USER VCC

No power is present at the emulator plug. (1) If the plug is connected to the user system, ensure that plug is installed correctly and that power is present; if the plug is connected to the Crystal Power Accessory (CPA), ensure that plug is installed correctly. (2) Use RESET ICE to attempt recovery. (3) Check the user hardware. (4) Run the confidence test.

ERR 43:PROCESSOR NOT RUNNING

The EA/ pin is inactive although no emulator program memory is mapped to the first 4K block, or the RESET pin is being held high while a command is being processed (i.e., not in emulation and no prompt is displayed). (1) Check the user system operation of these two pins. (2) Use RESET ICE to attempt recovery. (3) Run the confidence test.

ERR 51:BANK SWITCHING HUNG

The emulator has lost synchronization between the emulator hardware and the 8051 processor. (1) Use RESET ICE to attempt recovery. (2) EXIT and invoke the emulator again. (3) Check the user system for EA or RESET changing during command processing. (4) Run the confidence test.

ERR 52:UNWRITEABLE MEMORY

The user attempted to write to external program memory (not supported by the 8051). The emulator can write only to the 8K of emulator memory as mapped (using ASM, CBYTE, or LOAD commands). Check the address of the code memory you are trying to write.

ERR 80:SYNTAX ERROR

The token flagged by a #-sign in the previous line is not allowed in this command context. The command is ignored. (1) Check command syntax and enter command again.

ERR 81:INVALID TOKEN

The token flagged by a #-sign in the previous line is not properly formed, or is inappropriate for the command context; for example, the entry CBYTE 300 when SUFFIX = Y (binary).

ERR 83:RANGE ERROR

The value entered is not appropriate in the current context, or lies outside the range permitted for the specified memory or register type.

ERR 84:PARTITION BOUNDS ERROR

Incorrect values have been used to define a memory partition. This error can result when the value entered for the left bound exceeds that for the right bound, or when either value is out of range in the current context.

ERR 85:ITEM ALREADY EXISTS

The previous DEFINE command referred either to a symbol already in the user or system symbol table, or to a macro already in the macro definition table.

ERR 86:ITEM DOES NOT EXIST

The symbol or macro referenced in the previous command is not defined. (1) For a symbol, check version of code loaded, or define the symbol in the current session. (2) For a macro, define or INCLUDE the macro definition in the current session.

ERR 88:MACRO PARAMETER ERROR

A macro call contained more than ten actual parameters, or a parameter contained too many characters. (1) Check the definition of the macro.

ERR 89:MISSING CR-LF IN FILE

The current INCLUDE file does not end with a carriage return/linefeed. (1) EXIT and use your editor to correct the file.

ERR 8E:TRACE FRAME EMPTY

A trace reference (e.g., FRAME ADDR) has been used in an expression, but either no trace has been collected in the buffer or the buffer pointer is at NEWEST. (1) Check the contents of the trace buffer (PRINT command), (2) Move the pointer to the desired frame (MOVE command).

ERR 8F:NON-NULL STRING NEEDED

A null string (apostrophes with no enclosed characters) was used where at least one character is required, such as DBYTE 1 = ' '.

ERR 90:MEMORY OVERFLOW

The emulator workspace exceeded the amount allocated to it. The workspace contains the user symbol table and space for expansion of macros prior to their execution. The command that produced the overflow is aborted, but the memory already written remains as written. (1) To reclaim workspace, remove some user symbols.

ERR 92:COMMAND TOO LONG

The command exceeds the capacity of the emulator's command buffer. Possibly caused by too many operators. (1) Break the command or expression into several smaller units.

ERR 94:NON-CHANGEABLE ITEM

The command attempted to change a read-only value such as PPC or BUFFER-SIZE.

ERR 95:INVALID OBJECT FILE

The hexadecimal object file referenced in a LOAD command is not written in the proper format. Perhaps it is a text file rather than a hexadecimal object file. (1) Select another file for loading. (2) EXIT and verify file type.

ERR 99:EXCESSIVE ITERATED DATA

The number of data items to be repeated in memory exceeds the buffer size for iterated data (128 bytes). Example: CBYTE 30 TO 1000H = RBYTE 0 TO 256T.

ERR 9D:LINE TOO LONG

The input line exceeds 120 characters. (1) Use continuation lines (ampersand at end of input line to identify intermediate carriage return) to divide the command line into several input lines. (2) Break command into two or more shorter commands.

ERR A4:MACRO FILE FULL

The temporary file MAC.TMP has used all the available space on the diskette specified at invocation as the WORKFILE diskette. (1) Save and remove macro definitions to make room for more, using the PUT and REMOVE MACRO commands.

ERR A9:MAP CONTENTS CHANGED

Data read back from the map does not agree with data previously written. Something has caused the hardware version of the map to change since the last map command from the console. (1) Set the map again, then display the setting (MAP Command). (2) Run the confidence test.

ERR B3:OFFSET TOO LARGE

An address in an 8051 instruction (for ASM or as a mnemonic constant) results in a relative offset that is larger than 8 bits. (1) Check the relative offset; you may need a different instruction type.

ERR B7:PARTITION NOT ALLOWED

Command specified a range of values for a match condition in a GO command, GR command, or TR command. Partitions for match conditions are allowed only in the BR and QR commands.

ERR B8:ASSEMBLY IMPOSSIBLE

A mnemonic instruction in an ASM command or mnemonic constant has been specified incorrectly. (1) Check the correct instruction format and enter the instruction again.

ERR B9:NO HELP AVAILABLE

No HELP message is available for the item requested. (1) Type HELP with no modifier to obtain a menu of available HELP items.

ERR BB:ILLEGAL MAP BOUNDARY.

The map boundary in the previous MAP command does not fall on a 4K boundary. Map blocks must begin at multiples of 1000H (4K).

ERR BC:SYSTEM SYMBOL ERROR

Invalid operation on system symbol table, such as attempting to change the value of a system symbol, or using a multiple reference (e.g., .P0.P1) involving system symbols.

ERR BD:INVALID REG BANK NUMBER

The command attempted to set the register bank select (RBS) to a value other than 0, 1, 2, or 3.

WARN CO;EXCESSIVE DATA

The command attempted to enter data into a partition of memory too small for the number of data items specified. For instance, a command may attempt to load 50 bytes into a partition only 40 bytes long; in this case, the first 40 bytes are loaded correctly but the last 10 bytes of data are ignored.

WARN C2:HARWARE MISSING

The device with the expected device code (displayed on the line previous to this error) did not respond to initialization. (1) Check that the controller board has been correctly set to device code 32H and that all boards are correctly installed (refer to appendix A for details).

WARN C3:MULTIPLE HARDWARE

Two or more emulators are installed with the same device codes. (1) Check installation and device code setting so that only one ICE-51 emulator (device code 32H) can respond to initialization.

WARN C4:EXTRA FRAMES

During trace display in INSTRUCTION mode, the system detected extra frames after the end of the instruction. The extra frames cannot be assigned to any instruction. Probable cause is that the trace register setting qualified trace collection so as to omit some LOCATION frames.

WARN C9:VERSIONS DO NOT MATCH

Version numbers of software and downloadable firmware are not correct. (1) Verify that files ICE51 and ICE51.OVS are present on software diskette. (2) Substitute a fresh software diskette.

WARN CA:PPC/OPCODE NOT VALID

In updating trace information after emulation, up to 16 of the newest frames are searched for a LOC frame to refresh the values of PPC and OPCODE. Warning CA occurs if a LOC frame is not found in those 16 frames or if no trace was collected. Consequently, PPC and OPCODE will contain invalid data.

WARN CB:TRUNCATED TO 8 BITS

Warning CB occurs if the user specifies an OPCODE, VALUE, or port value in a match condition, or an expression in a mnemonic instruction to be assembled, that is greater than 8 bits. The value is truncated to the low 8 bits, the warning is issued, and processing continues.

WARN CC:UNEXPECTED TRACE

Warn CC occurs if, during trace update after breaking emulation or during trace display, code memory does not match the data collected in trace.

WARN CD:TRUNCATED TO 11 BITS

Warning CD occurs if the user specifies an address in an ACALL or AJMP instruction for assembly that is outside the current 2K block. The value is truncated to 11 bits, the warning is displayed, and processing continues.

ERR E7:ILLEGAL FILENAME

The command specified a filename that does not conform to ISIS-II format specifications. (Same as ISIS-II error 4.) (1) Check the ISIS-II User's Guide for proper filename format.

ERR E8:ILLEGAL DEVICE

The command contained a reference to an ISIS-II device, but the reference is illegal or unrecognizable. (Same as ISIS-II error 5.) (1) Check the ISIS-II User's Guide for proper device names.

ERR E9:FILE OPEN FOR INPUT

Command attempted to write data to a file opened for input (read-only) operations. (Same as ISIS-II error 6.) (1) Select another file for writing. (2) Close file and reopen.

ERR EF:FILE ALREADY OPEN

Command attempted to open a file that is already open. (Same as ISIS-II error 12.)

ERR F0:NO SUCH FILE
Command specified file that does not exist on the designated diskette. (Same as ISIS-II error 13.) (1) Verify the drive number and filename. (2) Check to see that the correct diskette is inserted in the drive.

ERR F1:WRITE-PROTECTED FILE

The command attempted to open a write-protected file for write access. (Same as ISIS-II error 14.) (1) Select another file. (2) EXIT and remove write protection from the target file (refer to ISIS-II manual for details).

ERR F3:CHECKSUM ERROR

A checksum error in a hexadecimal object file was encountered during loading. (Same as ISIS-II error 16.) (1) Select another file. (2) EXIT and create a correct object file.

ERR F6:DISKETTE FILE REQUIRED

The command specified a device other than a diskette file, but the operation requires a diskette file. (Same as ISIS-II error 19.)

ERR F9:ILLEGAL ACCESS

Command attempted to open a read-only device for write access (e.g., :CI: as a LIST device), or attempted to open a write-only device for read access (e.g., :LP: in LOAD command). (Same as ISIS-II error 22.) (1) Check command syntax for list of appropriate devices.

ERR FA:NO FILE NAME

The command references a diskette device, but omitted the filename, for example: LOAD :LF1:. (Same as ISIS-II error 23.)

ERR FD:"DONE" TIMED OUT
ERR FE:"ACKNOWLEDGE" TIMED OUT

Errors FD and FE indicate the emulator was unable to complete a requested operation. (1) Check hardware installation. (2) Use RESET ICE to attempt recovery. (3) EXIT and invoke emulator again. (4) Run the confidence test.

ERR FF:NULL FILE EXTENSION

The command referenced a file terminated by a period, but the implied extension is missing. (1) Omit the period or include the extension (refer to the ISIS-II User's Guide for details on filenames and extensions).

APPENDIX C
SERVICE AND REPAIR ASSISTANCE

United States Customers can obtain service and repair assistance by contacting the Intel Product Service Hotline in Phoenix, Arizona. Customers outside the United States should contact their sales source (Intel Sales Office or Authorized Distributor) for service information and repair assistance.

Before calling the Product Service Hotline, you should have the following information available:

a. Date you received the product.

b. Complete part number of the product (including dash number). On boards, this number is usually silk-screened onto the board. On other MCSD products, it is usually stamped on a label.

c. Serial number of product. On boards, this number is usually stamped on the board. On other MCSD products, the serial number is usually stamped on a label.

d. Shipping and billing addresses.

e. If your Intel product warranty has expired, you must provide a purchase order number for billing purposes.

f. If you have an extended warranty agreement, be sure to advise the Hotline personnel of this agreement.

Use the following numbers for contacting the Intel Product Service Hotline:

TELEPHONE:

All U.S. locations, except Alaska, Arizona, and Hawaii:

(800) 528-0595

All other locations: (602) 869-4600

TWX NUMBER:

910 - 951 - 1330

Always contact the Product Service Hotline before returning a product to Intel for repair. You will be given a repair authorization number, shipping instructions, and other important information which will help Intel provide you with fast, efficient service. If you are returning the product because of damage sustained during shipment or if the product is out of warranty, a purchase order is required before Intel can initiate the repair.

In preparing the product for shipment to the Repair Center, use the original factory packing material, if possible. If this material is not available, wrap the product in a cushioning material such as Air Cap TH - 240, manufactured by the Sealed Air Corporation, Hawthorne, N.J. Then enclose in a heavy duty corrugated shipping carton, and label "FRAGILE" to ensure careful handling. Ship only to the address specified by Product Service Hotline personnel.

APPENDIX D HARDWARE SPECIFICATIONS

This appendix contains specifications and other information regarding the emulator hardware: environmental and power requirements, user plug characteristics, synchronization line characteristics, and buffer box indicator lights.

Environmental and Power Requirements

The power for the emulator is drawn from the development system. The amounts at the backplane are:

			max	typical
5V	+5%	-1%	13.2A	8.0A
12V	±5%		0.1A	0.05A
-10V	±5%		0.05A	0.01A

The emulator requires the following atmospheric environment:

- Buffer box and cables operating temperature 0 to 55 degrees C.
- Intellec resident boards operating temperature 0 to 55 degrees C.
- Operating humidity 0 to 85% RH.
- Storage temperature -65 to 85 degrees C.
- Storage humidity 0 to 85% RH

User Plug Characteristics

Sixteen of the user emulator plug pins are connected directly to the 8051E emulator chip. These user emulator pins are distinguishable from the corresponding 8051 pins only by the presence of an 8pf load attributable to the connections within the user emulator plug. Twenty four of the port pins (port 2, 1, and 0 are connected to actual 8051E processor pins, but have additional loading contributed by the cable to the Buffer Box and the input of the trace buffers. This amounts to a worst case additional load of 50 microamps and 25pf which is a three fold improvement over most previous emulators' 150 microamp to 250 microamp trace buffer loading.

Timing is exactly that of the 8051 (the 8051E is essentially an 8051 chip) for all pins.

Emulation does not break if user Vcc is absent or if the user clock stops; however, an error message is presented when a break occurs afterward. Thus it is possible to emulate the 8051 power low reset. Note however that the Vcc missing warning is issued when emulation breaks. The Vcc missing warning may be issued without the 8051 resetting itself or vice versa, due to the difference in threshold levels and the particular user system configuration.

Sync Lines Characteristics

Sync lines SY0 and SY1 are accessible at the buffer box. SY0 controls or reflects real-time emulation; SY1 controls or reflects trace collection.

SY0/SY1 output drive is a TTL open collector with an internal 3.3k pull-up resistor. Low level sink capability is 20 mA. Do not actively drive the SY0/SY1 lines high when output is enabled because of possible conflicts.

SY0/SY1 logic level is high (TRUE or 1) when the voltage is between 2.0V and Vcc. It is low (FALSE or 0) when the voltage is between ground and 0.8V.

Critical timing delay factors for SY0 and SY1 input/output appear in table D-1.

Buffer Box Indicator Lights

Buffer box lights indicate emulator status as described in table D-2.

Table D-1. Sync Lines Delay Factors

Condition Enabled	Period	Delay (Max/Min)
OUTPUT	From emulation termination until SY0 goes low.	1 cycle max.
	From emulation initiation until SY0 goes high.	1 cycle max.
	From GO command entry until SY0 output goes high (with SY0 input enabled).	0.5 sec min.
	From trace termination until SY1 goes low.	1 cycle max.
	From trace initiation until SY1 goes high.	1 cycle max.
INPUT, LEVEL AND LATCHED	From SY0 going low until emulation terminates.	4 cycles max.
	From SY0 going high until emulation initiaties.	200 μsec max.
	From SY1 going low until trace terminates.	1 cycle max.
	From SY1 going high until trace initiates.	1 cycle max.
INPUT, LEVEL ONLY	SY0 low hold time	4 cycles min.
	SY0 high hold time (level-sensitive)	200 μsec min.
INPUT, LATCHED ONLY	From SY0 or SY1 input valid until ALE signal signal positive edge.	500 nsec max.
	SY0 or SY1 low hold time.	30 nsec min.

Table D-2. Buffer Box Indicator Lights

Lights On	Emulator Status	User Action
Yellow	Interrogate Mode; no command in progress.	Enter any command.
None	Command in progress.	To abort command, press ESC key.
Green and Yellow	Emulation in progress.	To abort emulation, press ESC key.
Green	GO Command entered while SY0 IN enabled and SY0 has never been high; waiting for SY0 to go high so emulation can begin.	To abort command, press ESC key.
Red and Yellow	Hardware error, recovery has occurred. The red-yellow combination occurs momentarily during RESET ICE.	Enter any command.
Red, or Red and Green, or All three	Hardware error, no recovery has occurred.	Try ESC, then RESET ICE, or reset development system.

APPENDIX E OPERATING HINTS AND LIMITATIONS

The following shortcomings do not seriously impede usage of this product but are nonetheless worth notice:

- Port 3 is not traced.
- Symbols are not typed (RAM vs ROM). There is no provision in the absolute hex object file format for type information, and so types are not passed from the assembler and cannot be saved in the object file. As a result symbols from the two memory spaces overlap, and so symbolic disassembly and evaluation may produce inappropriate results.
- Internal data memory accesses are not available for break and trace.
- Map settings must agree with the EA/ pin or results will be unpredictable. (e.g., If EA/ is low, then the map must not contain a block at zero and vice versa.)
- The RESET pin should only be allowed to be active while in emulation, or else an error occurs.
- Ports are changed by the 8051 after the opcode fetch *following* an instruction to change a port. Therefore, with trace displayed in INSTRUCTIONS mode the port will appear to change on the second instruction following a port change. Also if a break occurs between the port change instruction and its appearance in trace, and more trace is then collected with no user change in PC the composite trace will reflect the change earlier than on an uninterrupted trace. This is because of the extra time spent in interrogation mode between the port change instruction and the second series of trace.
- Qualified trace or anytime trace is starting while emulation is in progress will start trace the frame *after* the qualifying event. (e.g., TR = LOC IS XXXXH will collect opcodes.) When the 8051 speed decreases to below approximately 3 mHz, qualified trace will capture the qualifying frame as well as the one frame after it.
- At power up, the 8051 processor will execute random instructions until the emulator software is invoked. In some applications this can be troublesome. For such applications, power and clock should not be applied to the user emulator plug from the prototype system until the emulator has been invoked. After doing so, the RESET ICE command must be issued to permit the emulator to complete its initialization.
- The CAUSE command or reference sometimes includes both BR0 and BR1 when only one breakpoint matched and caused the break in emulation. This situation can occur when the two breakpoints could match within five addresses of each other, or when either is set to a location that is within the first four addresses in a block that starts on an even multiple of 4K (e.g., 0 to 3, 2000H to 2003H, 4000H to 4003H). Trace collection and breakpoint function are not affected.
- The XADDR field in trace can contain an incorrect external opcode (not the one actually executed). The external address written is always valid, but the data read back for trace collection may differ depending on user hardware. Program execution by the emulation processor is not affected.
- During RESET ICE, the 8051 executes a few random instructions.
- If an interrupt occurs at the same time as a break, trace collection can be altered. If trace is in AFTER or qualify mode, four additional frames may be added to the end of the trace buffer, depending on the exact interrupt timing within the instruction cycle. This will exclude the first four frames that should have been collected.

Limitations

- Any CBYTE read in external memory (unmapped memory) will leave P0 floating and disturbs P2 as does any MOVC instruction executed by an 8051.
- On-chip timers which are running at the time of break or step will receive one extra count for every break.

- Stack locations 0 → 6 are reserved for emulator operation. These locations are saved & preserved for the user between emulations. The only limitation is that user application programs leave the stack pointer at the power-on location (07) or higher.

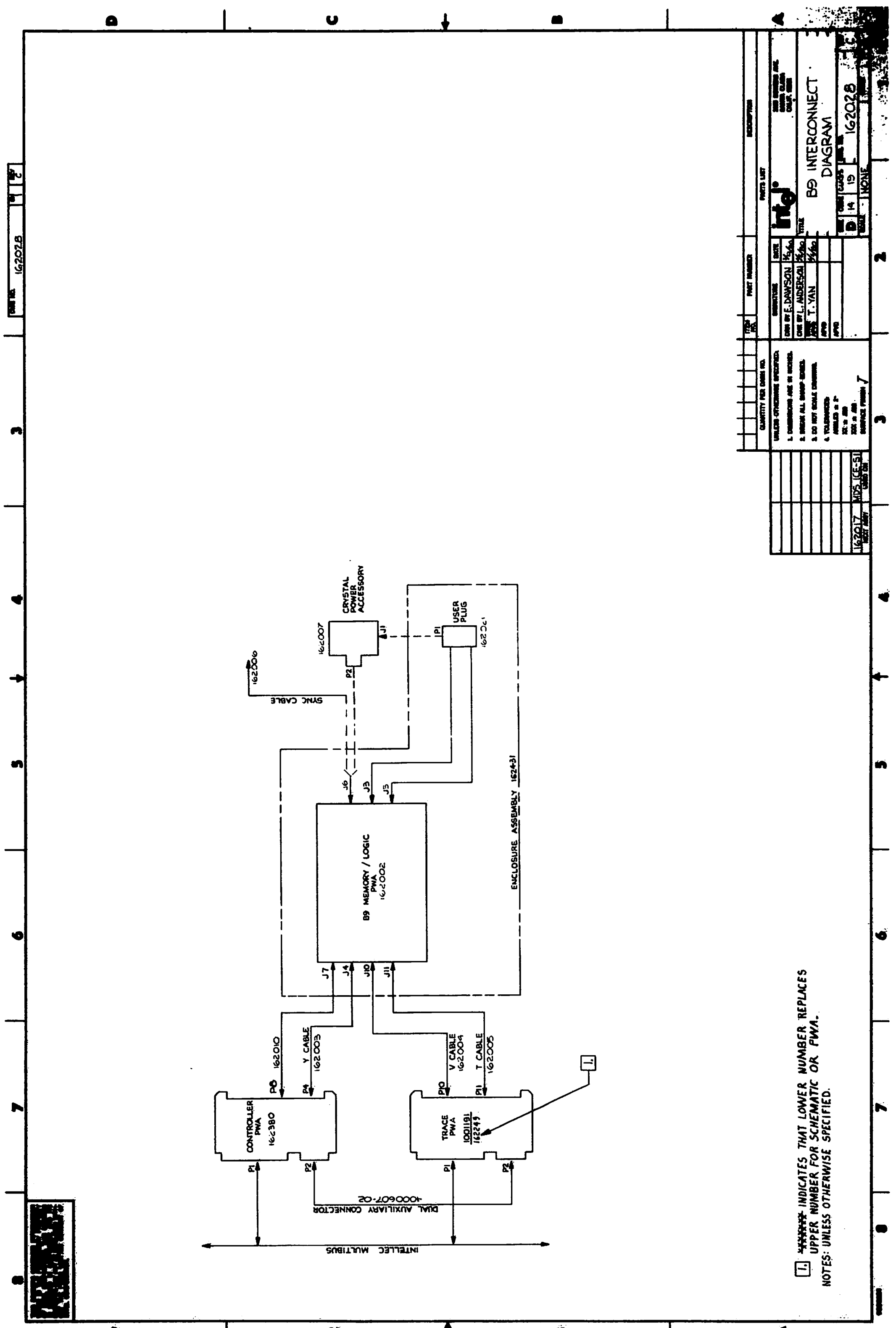
APPENDIX F
REFERENCE SCHEMATICS

This appendix contains the following schematic drawings, for user reference:

TITLE	NUMBER
Interconnect Diagram (2 sheets)	162028
B8 Controller Schematic (7 sheets)	162407
Trace Board Schematic (6 sheets)	162379
B9 Memory/Logic Schematic (6 sheets)	162000
User Plug Schematic (1 sheet)	162018
Crystal Power Accessory Schematic (1 sheet)	162009

NOTE

The documents in this appendix are for general reference only.



Interconnect Diagram (Sheet 1 of 2)

162028 SH 2 REV C

SEE SHEET 1

SIGNAL NAME CROSSREFERENCES

"Y" CABLE		
CONTROLLER P4	PIN NO.	MEMORY P4
A5-Y	1	A5Y
AØ-Y	2	AØY
A1-Y	3	A1Y
A2-Y	4	A2Y
A3-Y	5	A3Y
A4-Y	6	A4Y
CS-Y/	7	CSY/
DØ-Y	8	DØY
D1-Y	9	D1Y
D2-Y	10	D2Y
D3-Y	11	D3Y
D4-Y	12	D4Y
D5-Y	13	D5Y
D6-Y	14	D6Y
D7-Y	15	D7Y
—	16	+5V
RD-Y/	17	RDY/
GND	18	GND
WR-Y/	19	WRY/
GND	20	GND
TRACE OFF-Y/	21	TOFFY
GND	22	GND
B-EMULATE-Y	23	EMULY
TIMER ON	24	TIMON
—	25	GND
—	26	+5V
GND	27	GND
—	28	+5V
GND	29	GND
STOPPROC-Y/	30	STOPY/
—	31	+5V
—	32	+5V
OLYCKF-Y/	33	QLCKY/
—	34	+5V
—	35	+5V
CLOCK-Y/	36	CLKY/
—	37	+5V
GND	38	GND
STOPTR/	39	STTY/
GND	40	GND

SCHEMATIC 162407 | SCHEMATIC 162000

"T" CABLE		
TRACE P11	PIN NO.	LOGIC P11
GND	1	GND
GND	2	GND
C25C	3	P14T/
C26C	4	P15T/
C28C	5	P17T/
C27C	6	P16T/
C29C	7	RB4T/
C30C	8	RB5T/
C32C	9	RB7T/
C31C	10	RB6T/
C33C	11	RB12T/
C34C	12	RB13T/
C36C	13	RB15T/
C35C	14	RB14T/
C37C	15	RB8T/
C38C	16	RB9T/
—	17	+5V
C40C	18	RB11T/
—	19	+5V
C39C	20	RB1ØT/
—	21	+5V
C42C	22	ADDATT
-10V	23	—
C41C	24	CITRT
GND	25	GND
GND	26	GND

SCHEMATIC 1. 2002815 162377 | SCHEMATIC 162000

"V" CABLE		
TRACE P10	PIN NO.	LOGIC P10
GND	1	GND
GND	2	GND
C16C	3	P27V/
C14C	4	P25V/
C12C	5	PØ7V/
C1ØC	6	PØ5V/
CØ8C	7	P23V/
CØ6C	8	P21V/
CØ4C	9	PØ3V/
CØ2C	10	PØ1V/
C15C	11	P26V/
C13C	12	P24V/
C11C	13	PØ6V/
CØ9C	14	PØ4V/
CØ7C	15	P22V/
CØ5C	16	P2ØV/
CØ3C	17	PØ2V/
CØ1C	18	PØØV/
GND	19	GND
C18C	20	P11V/
GND	21	GND
C2ØC	22	P13V/
GND	23	GND
C22C	24	RB1V/
C17C	25	P1ØV/
C19C	26	P12V/
GND	27	GND
C21C	28	RBØV/
GND	29	GND
C23C	30	RB2V/
GND	31	GND
C24C	32	RB3V/
GND	33	GND
GND	34	GND

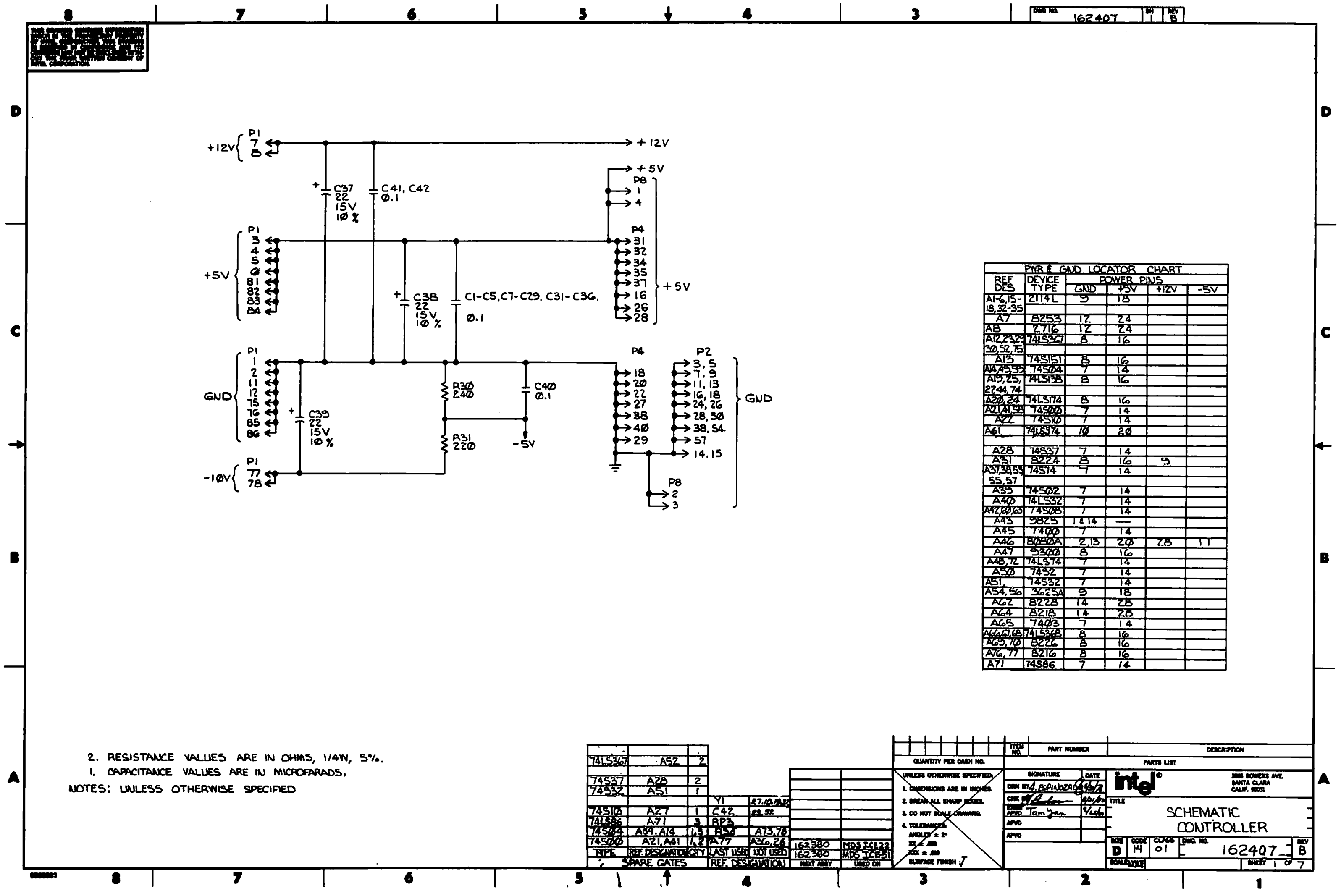
SCHEMATIC 1. 2002815 162377 | SCHEMATIC 162000

DUAL AUXILIARY CONNECTOR		
CONTROLLER P2	PIN NO.	TRACE P2
—	1	MHA1
—	2	MHA2
GND	3	GND
M1/	4	M1/
GND	5	GND
M2/	6	M2/
GND	7	GND
—	8	MHA3
GND	9	GND
—	10	MHA4
GND	11	GND
CLKB/	12	CLKB/
GND	13	GND
GND	14	TST2/
GND	15	TST1
GND	16	GND
WE-B/	17	WE-B/
GND	18	GND
QLYCLKB/	19	M34/
—	20	—
CTOVFLW/	21	CTOFLW/
—	22	—
INOQLY/	23	INOQLY/
GND	24	GND
B-CTOVFLW/	25	MTH-B/
GND	26	GND
IENWE/	27	IENWE/
GND	28	GND
STOPCT/	29	STOPCT/
GND	30	GND
MEMW-B/	31	MEMW/
SELCP3/	32	—
SELCP1/	33	SELCP1/
SELCP2/	34	SELCP2/
A6	35	A6
—	36	—
MEMR-B/	37	—
GND	38	—
LD MOD	39	LD MOD
A3	40	A3
A4	41	A4
A5	42	A5
DØ-Y	43	DAØ
D1-Y	44	DA1
D2-Y	45	DA2
D3-Y	46	DA3
D4-Y	47	DA4
D5-Y	48	DA5
D6-Y	49	DA6
D7-Y	50	DA7
AØ	51	—
A1	52	—
A2	53	—
GND	54	GND
CS1/	55	CS1/
CS2/	56	CS2/
GND	57	GND
ENRDM/	58	ENRDM/

SCHEMATIC 162407 | 1. SCHEMATIC 2002815 162377

9-25-80 SIZE D CODE 14 CLASS 19 DWG NO. 162028 REV C SCALE — SHEET 2

Interconnect Diagram (Sheet 2 of 2)



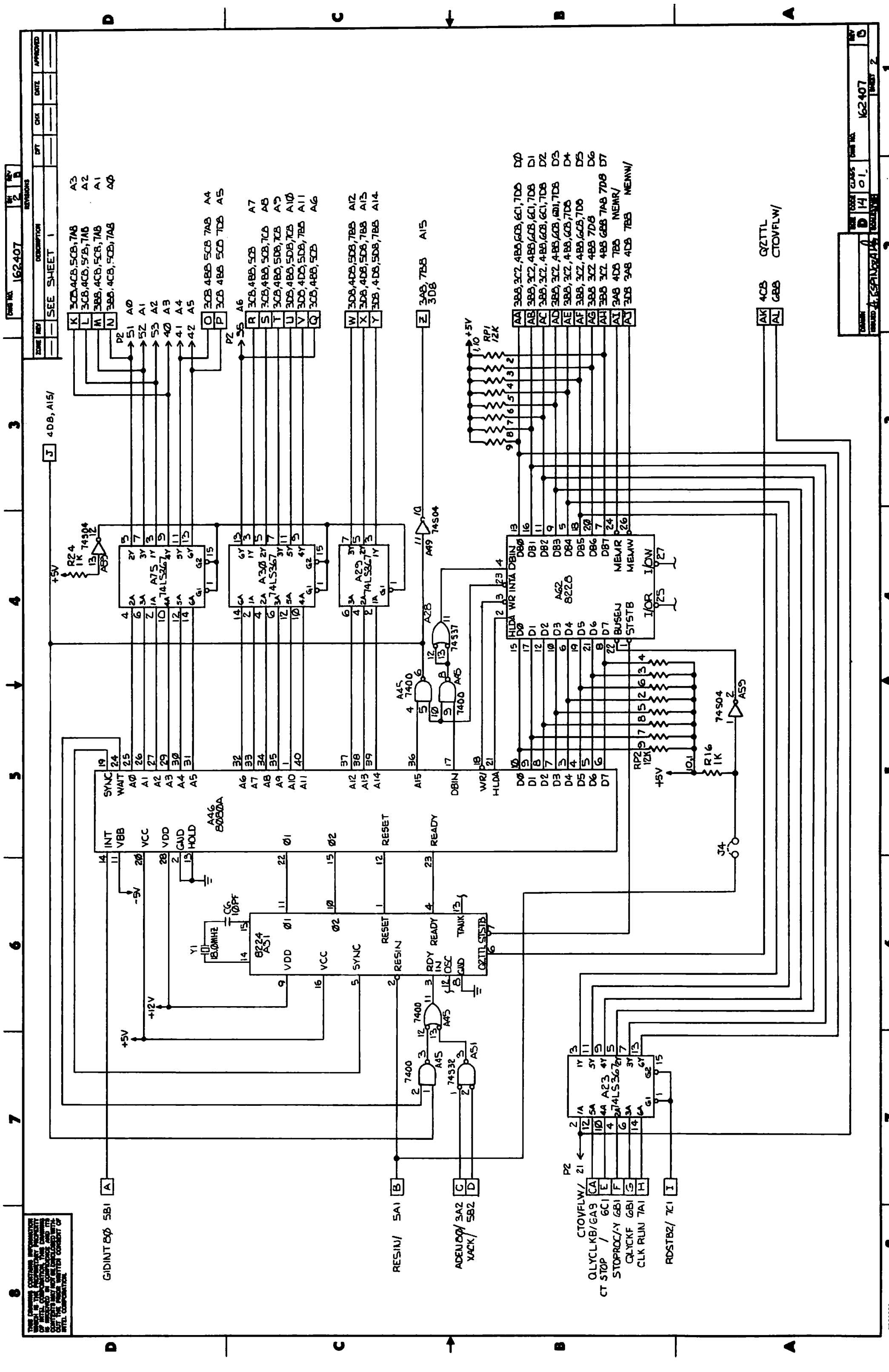
PWR & GND LOCATOR CHART					
REF DES	DEVICE TYPE	POWER PINS			
		GND	+5V	+12V	-5V
A1-6,15-18,32-35	2114L	9	18		
A7	8253	12	24		
A8	2716	12	24		
A12,23,29,30,52,75	74LS367	8	16		
A13	74S151	8	16		
A14,49,59	74S04	7	14		
A19,25,27,44,74	74LS138	8	16		
A20,24	74LS174	8	16		
A21,41,58	74S00	7	14		
A22	74S10	7	14		
A61	74LS374	10	20		
A28	74S37	7	14		
A31	8224	8	16	9	
A37,38,53,55,57	74S74	7	14		
A39	74S02	7	14		
A40	74LS32	7	14		
A42,60,63	74S08	7	14		
A43	9825	1 & 14	—		
A45	7400	7	14		
A46	8080A	2,13	20	28	11
A47	9300	8	16		
A48,72	74LS74	7	14		
A50	7432	7	14		
A51	74S32	7	14		
A54,56	3625A	9	18		
A62	8228	14	28		
A64	8218	14	28		
A65	7403	7	14		
A66,67,68	74LS368	8	16		
A69,70	8226	8	16		
A76,77	8216	8	16		
A71	74S86	7	14		

SPARE GATES			REF. DESIGNATION	
TYPE	REF. DESIGNATION	QTY	LAST USED	NOT USED
74LS367	A52	2		
74S37	A28	2		
74S32	A51	1		
			Y1	R7,10,18,20
74S10	A27	1	C42	21,52
74LS86	A71	3	RP3	
74S04	A59, A14	1,3	R58	A73,78
74S00	A21, A41	1,2	A77	A36,26

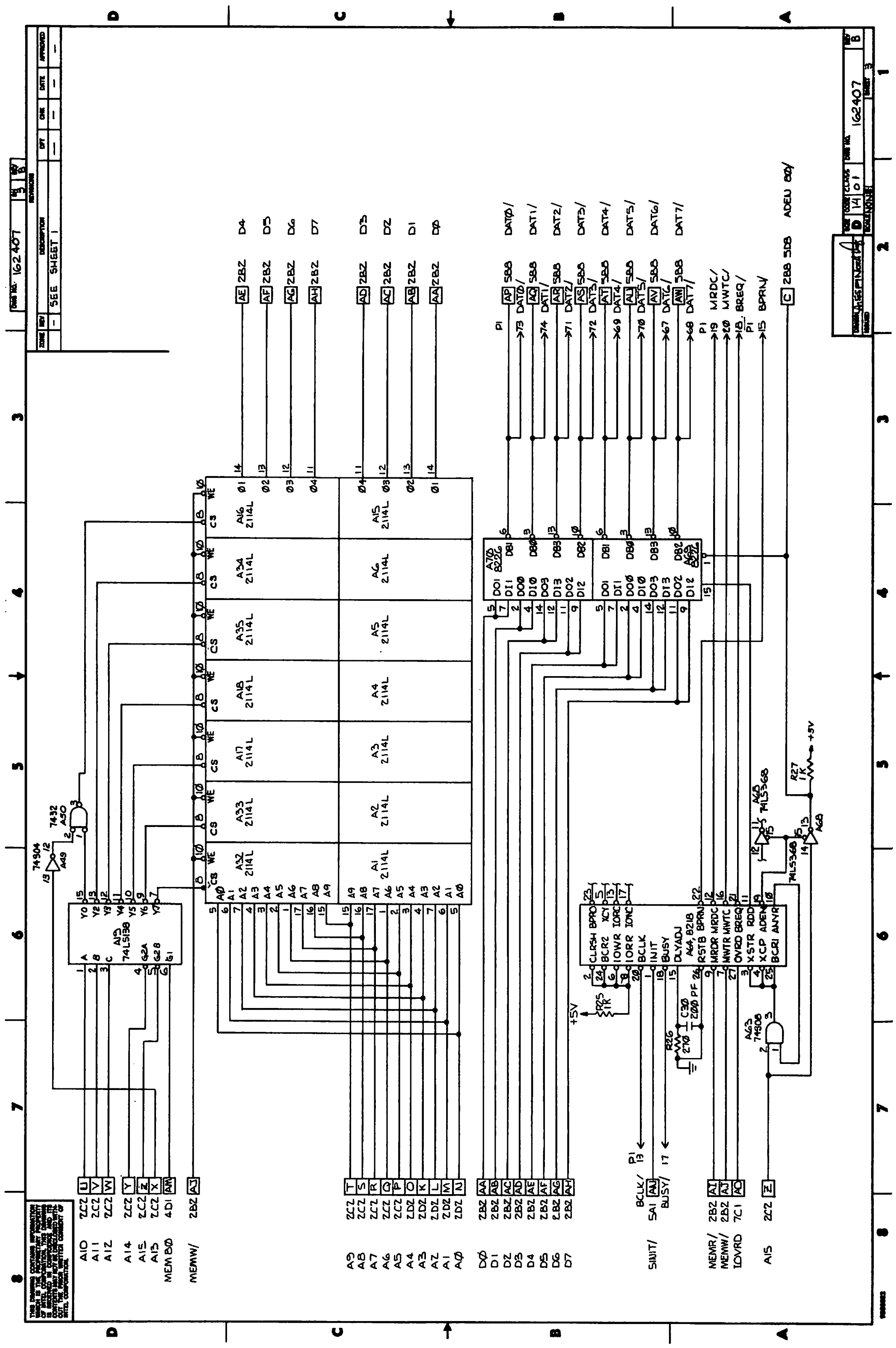
NEXT ASSY	USED ON
162380	MDS ICE22
162380	MDS ICE51

NOTES: UNLESS OTHERWISE SPECIFIED
1. CAPACITANCE VALUES ARE IN MICROFARADS.
2. RESISTANCE VALUES ARE IN OHMS, 1/4W, 5%.

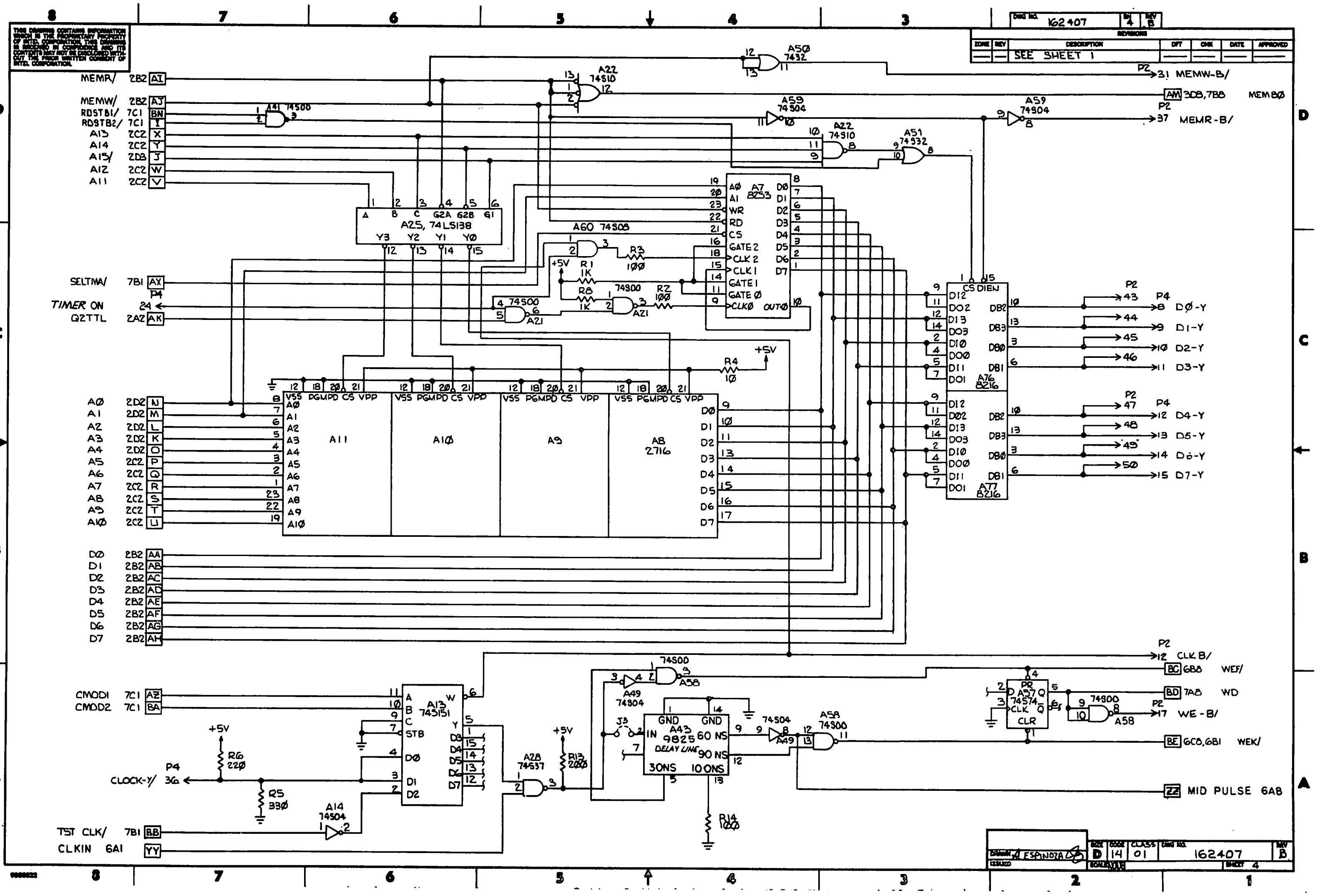
B8 Controller Schematic (Sheet 1 of 7)



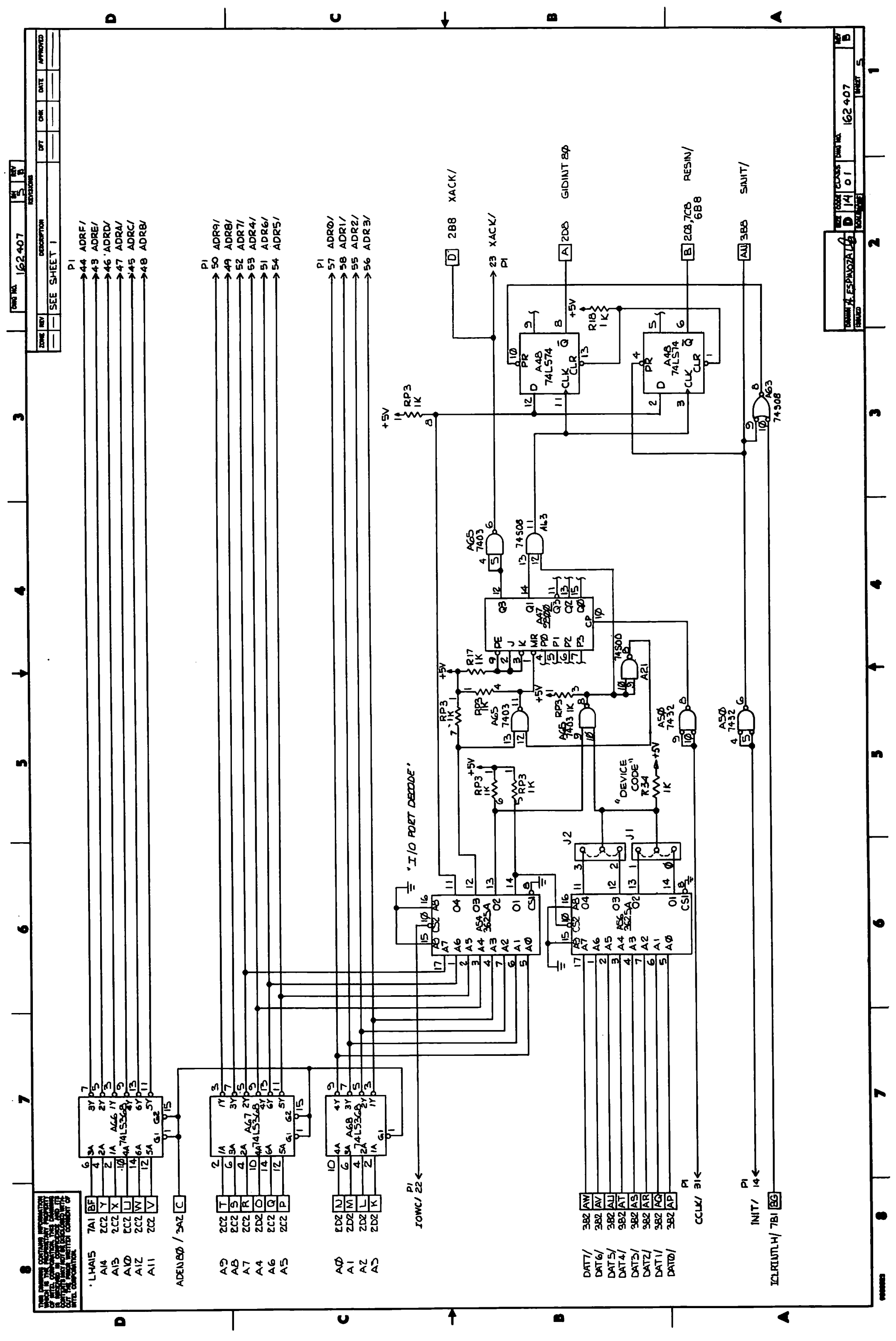
B8 Controller Schematic (Sheet 2 of 7)



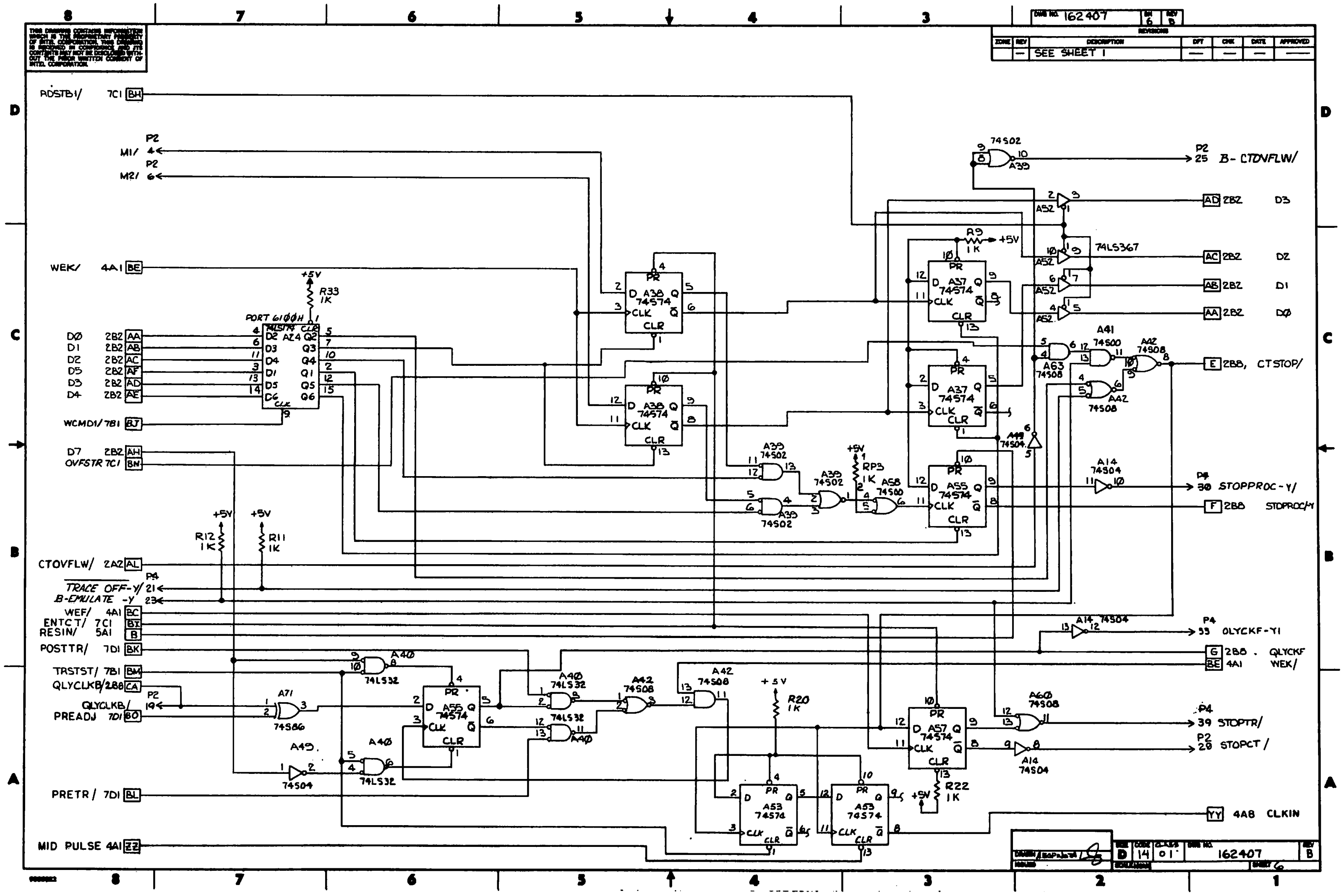
B8 Controller Schematic (Sheet 3 of 7)



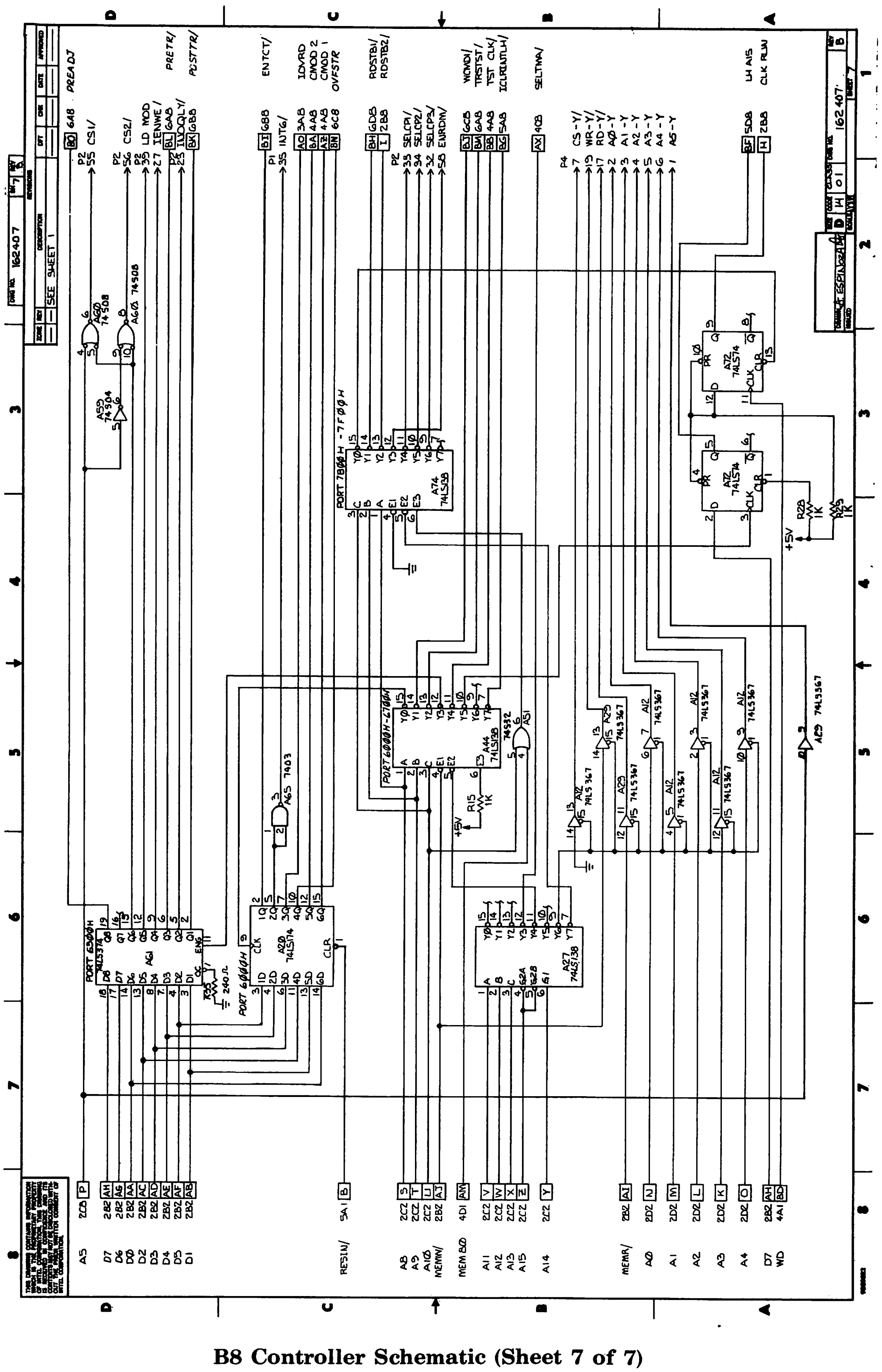
B8 Controller Schematic (Sheet 4 of 7)



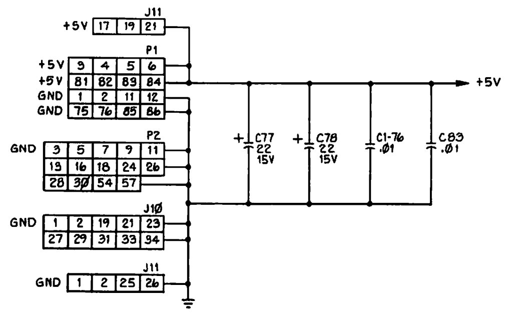
B8 Controller Schematic (Sheet 5 of 7)



B8 Controller Schematic (Sheet 6 of 7)



B8 Controller Schematic (Sheet 7 of 7)



NOTES: UNLESS OTHERWISE SPECIFIED
1. ALL RESISTOR VALUES ARE IN OHMS, 1/4W, 5%.
2. ALL CAPACITOR VALUES ARE IN MICROFARADS.
3. * = SPARE GATES "EBL" ALREADY USED

POWER & GROUND LOCATOR CHART			
REFERENCE DESIGNATOR	DEVICE TYPE	GROUND	POWER PINS +5V
A91	7432	7	14
A71-76,81-85	3101A	8	16
A86	34Ø4	8	16
A24,25,43,44	74SØ4	7	14
A87,90	74SØ8	7	14
A79	74S1Ø	7	14
A88	74LS32	7	14
A7	74LS74	7	14
A62,77,80	74LS138	8	16
A78	74S14Ø	7	14
A13-15	74S163	8	16
A1-6,8-12	74LS298	8	16
A92-94,51-53,70,89	74LS367	8	16
A16-23,26-31,	2115	8	16
A35-42,45-50,	2115	8	16
A54-61,63-69	2115	8	16

SPARE GATES		
TYPE	REF DES	QTY
74LS367	A70 *	2
74LS367	A52 *	2
74LS367	A94 *	1
74LS367	A92	2
74LS74	A7	1
7432	A91	3
74S08	A87	2
3404	A86	2
74S04	A44	1
74S04	A43	1
74S04	A25	1
1K RES. PK.	RP4	1
1K RES. PK	RP1	1

ASSEMBLIES USING THIS SCHEMATIC
162247-001
1001191-07

REF DESIGNATOR	
LAST USED	NOT USED
J11	J1-9
P2	
RP8	
A20	
C84	
A91	

NEXT ASSY	USED ON
	ICE 51/SI
	ICE 51
	ICE 22
	ICE 856/SI
	ICE 856

UNLESS OTHERWISE SPECIFIED:
1. DIMENSIONS ARE IN INCHES.
2. BREAK ALL SHARP EDGES.
3. DO NOT SCALE DRAWING.
4. TOLERANCES:
ANGLES ± 2°
XX ± .03
XXX ± .010
SURFACE FINISH √

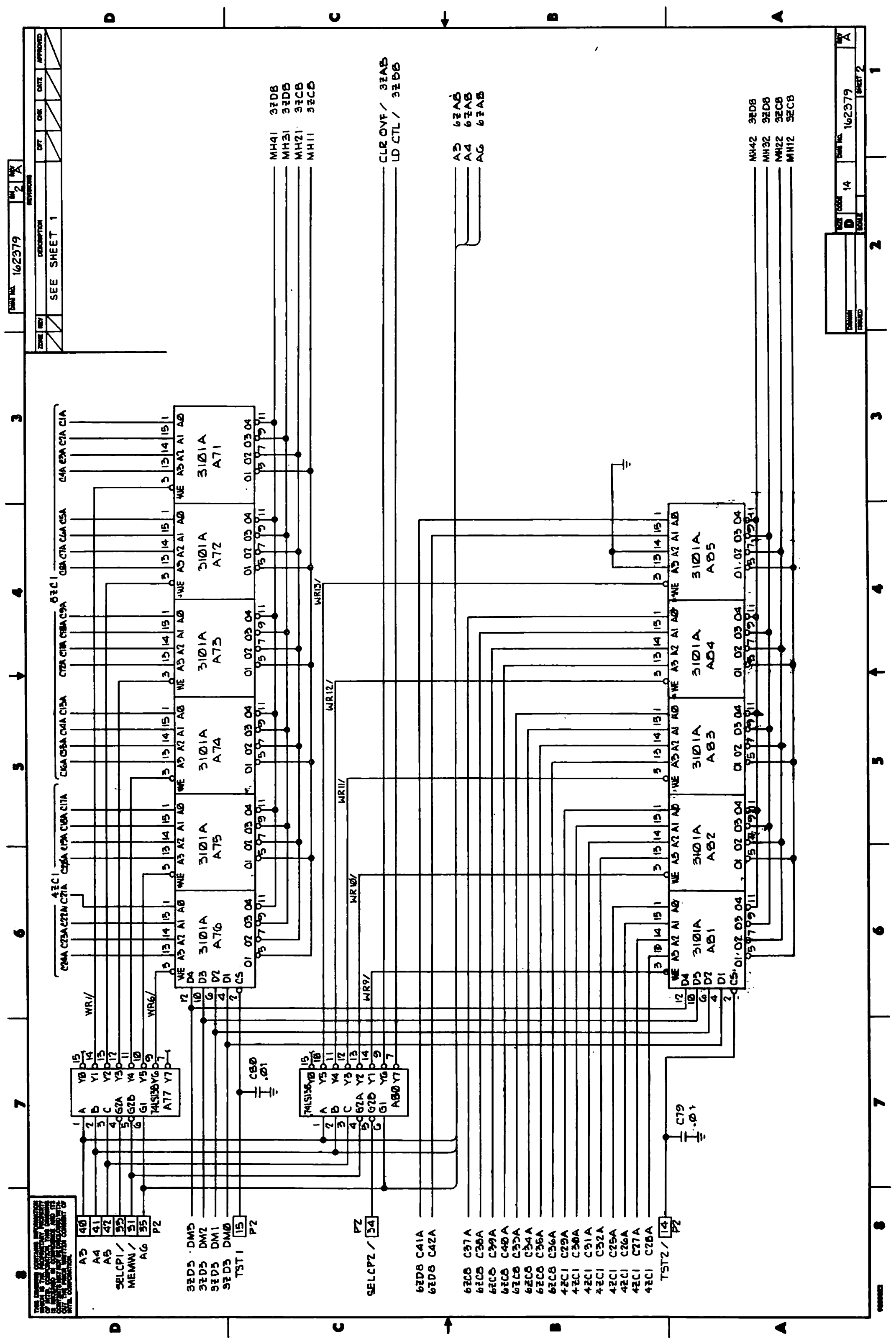
intel® 3065 BOWERS AVE. SANTA CLARA CALIF. 95051

TITLE SCHEMATIC, TRACE BOARD

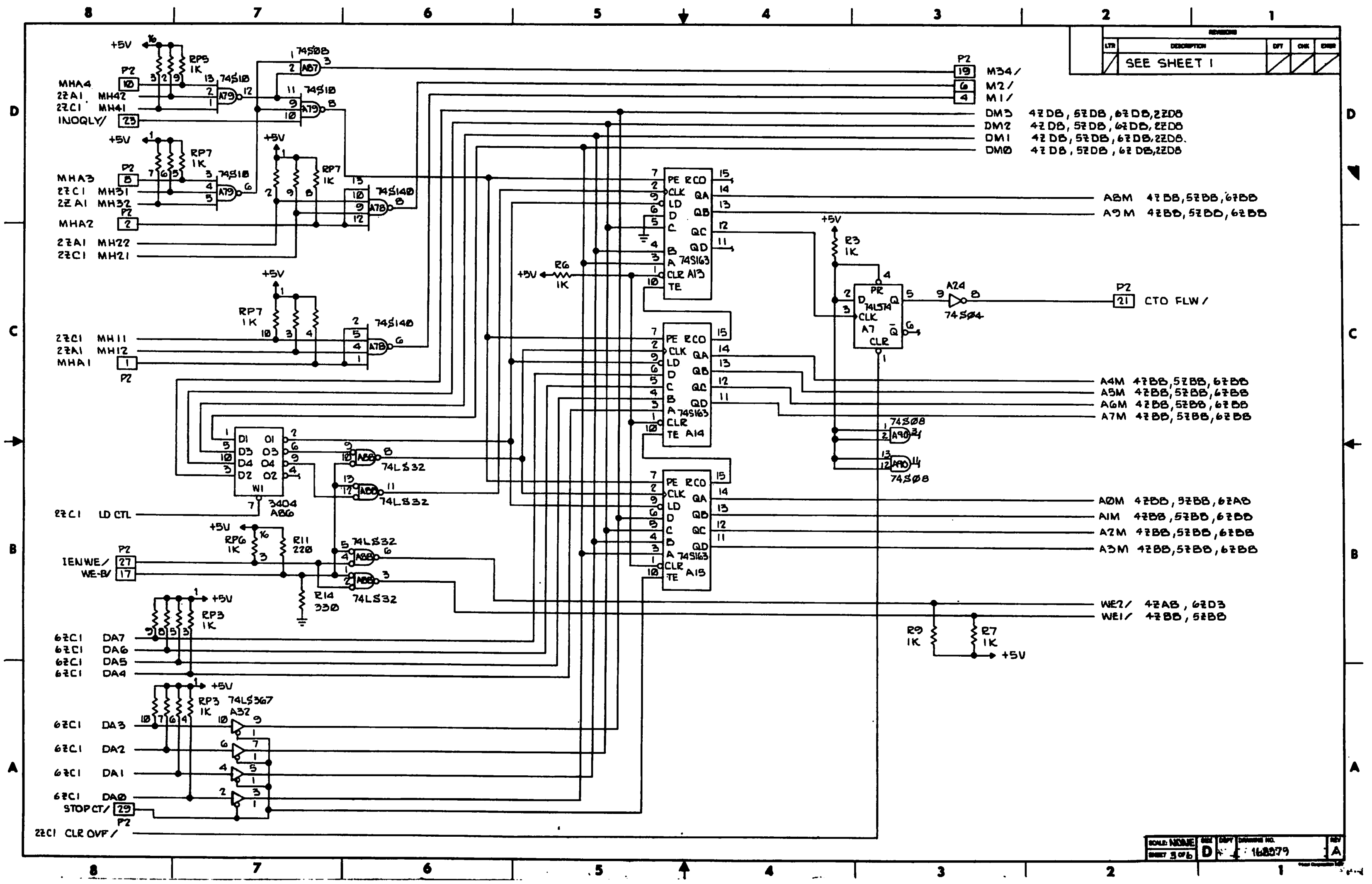
SIZE D CODE 14 CLASS Ø1 DWG. NO. 162379 REV A SHEET 1 OF 6

DWG NO. 162379 SH 1 REV A

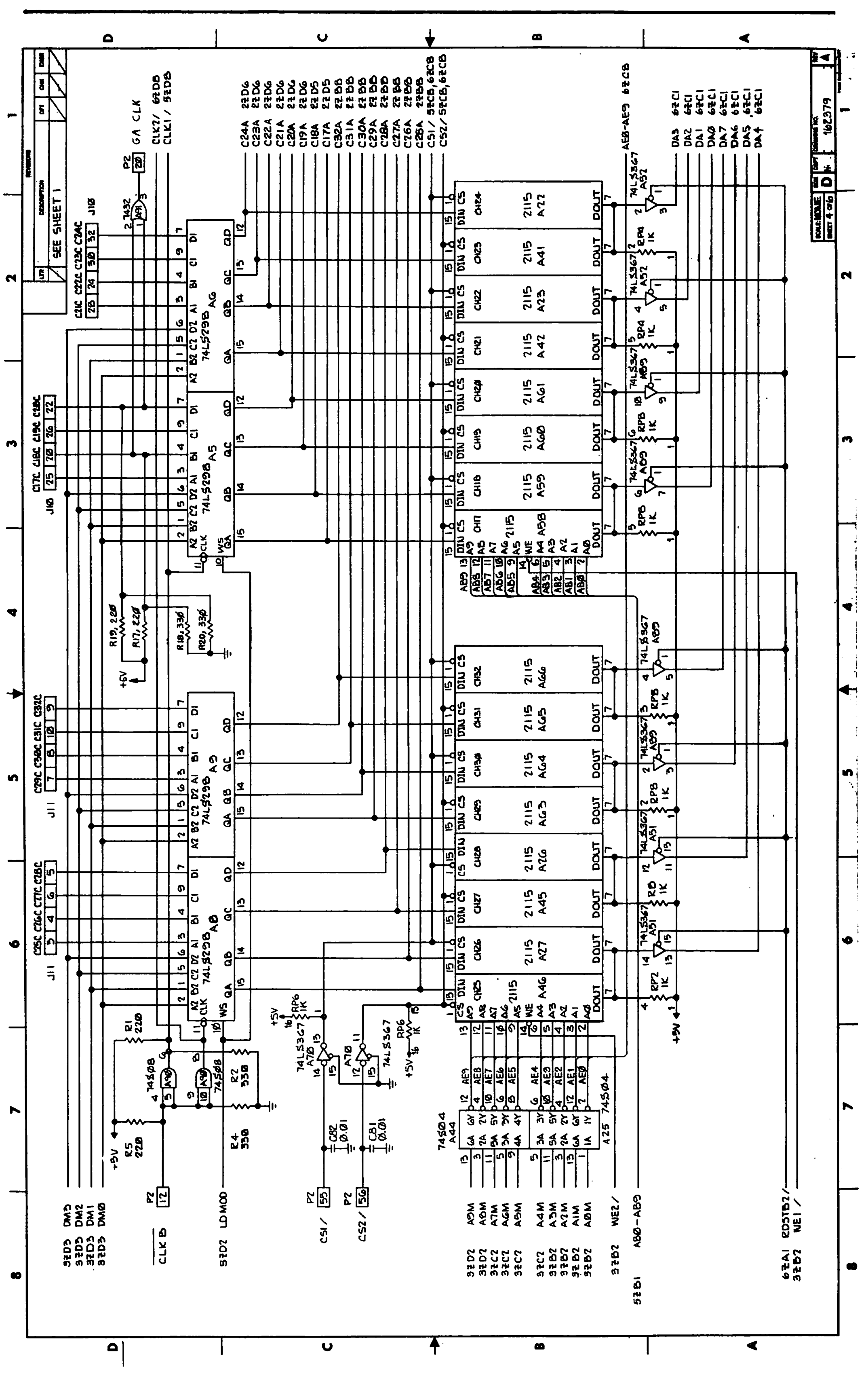
Trace Board Schematic (Sheet 1 of 6)



Trace Board Schematic (Sheet 2 of 6)

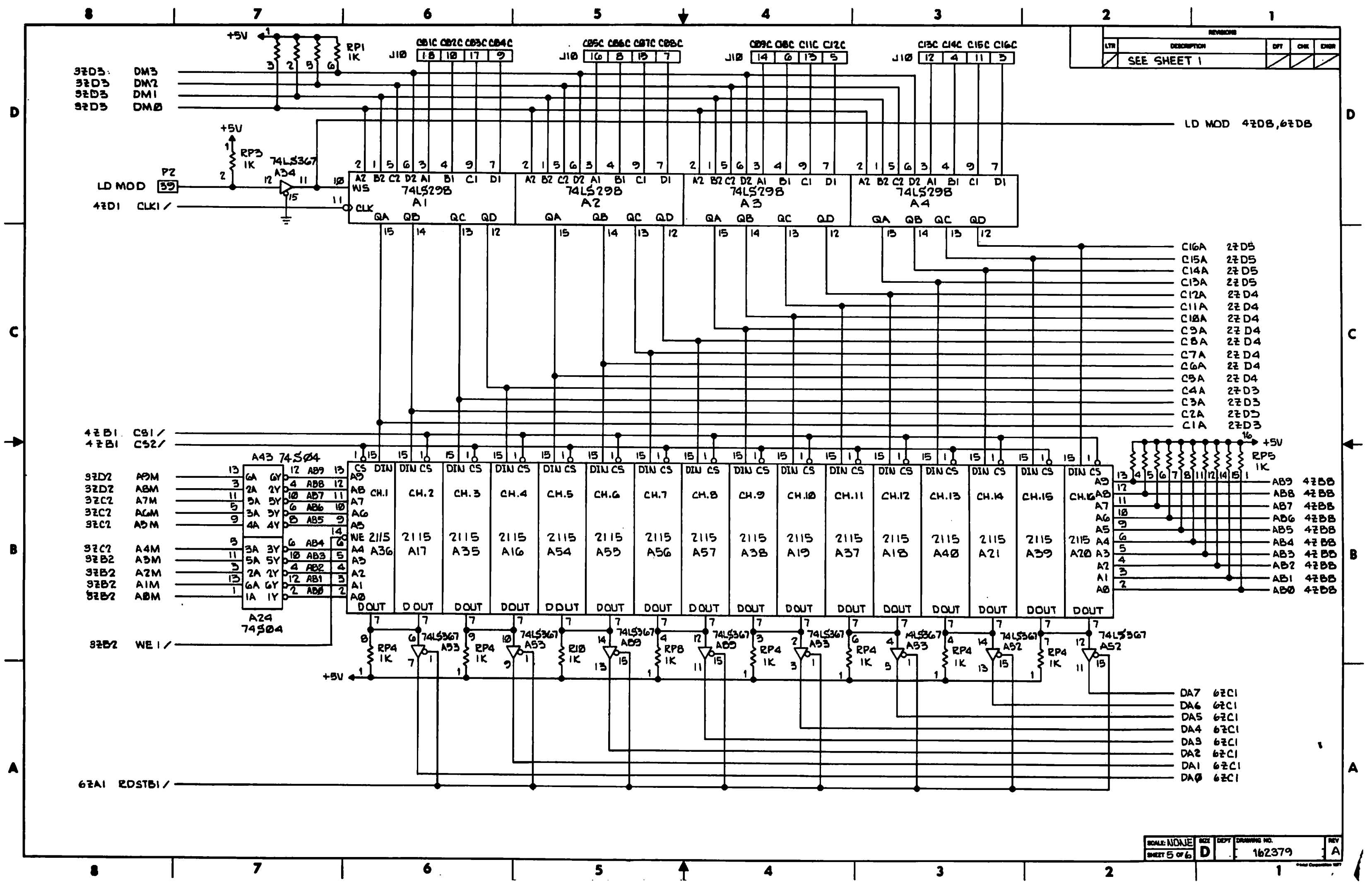


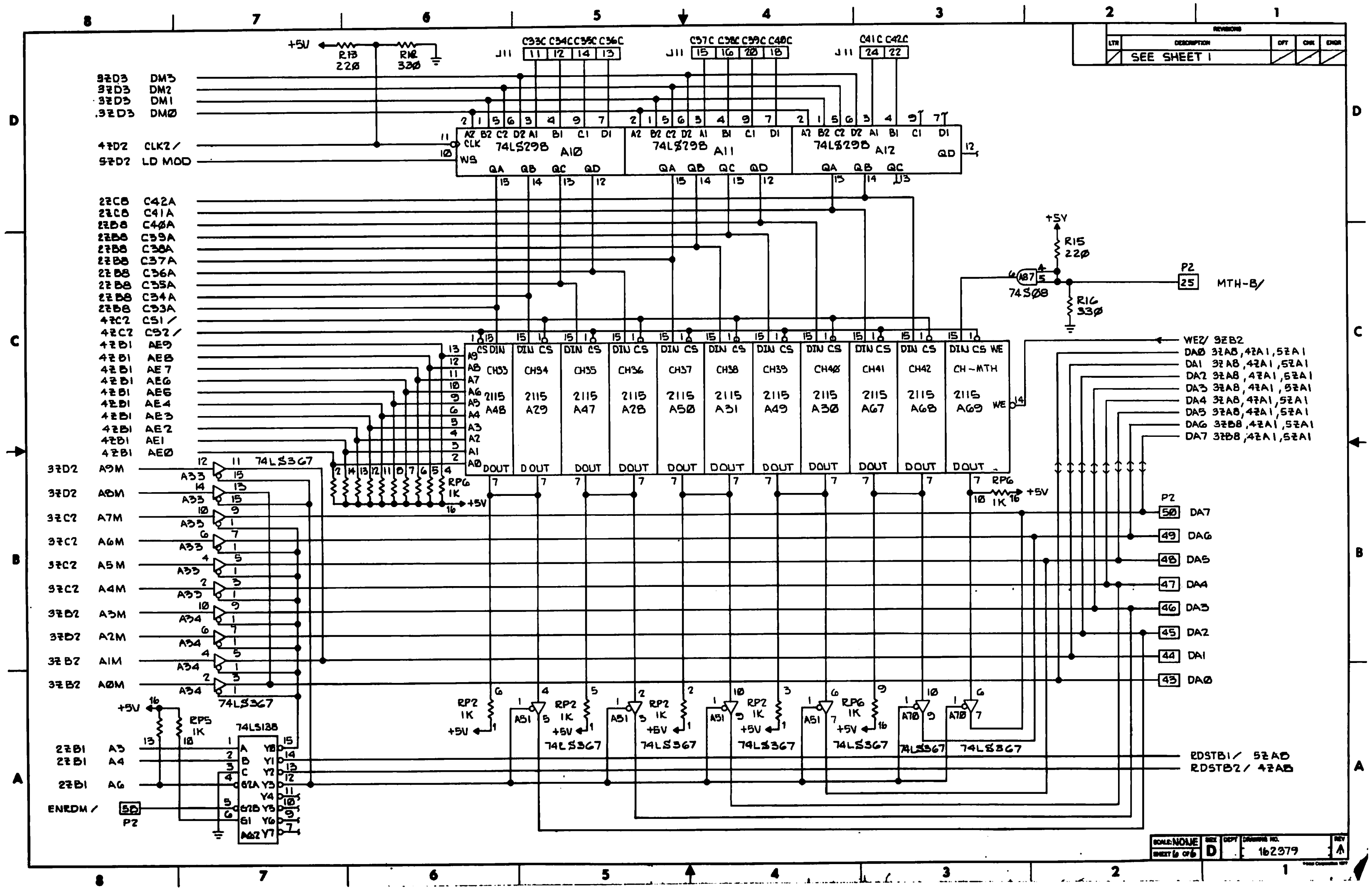
Trace Board Schematic (Sheet 3 of 6)



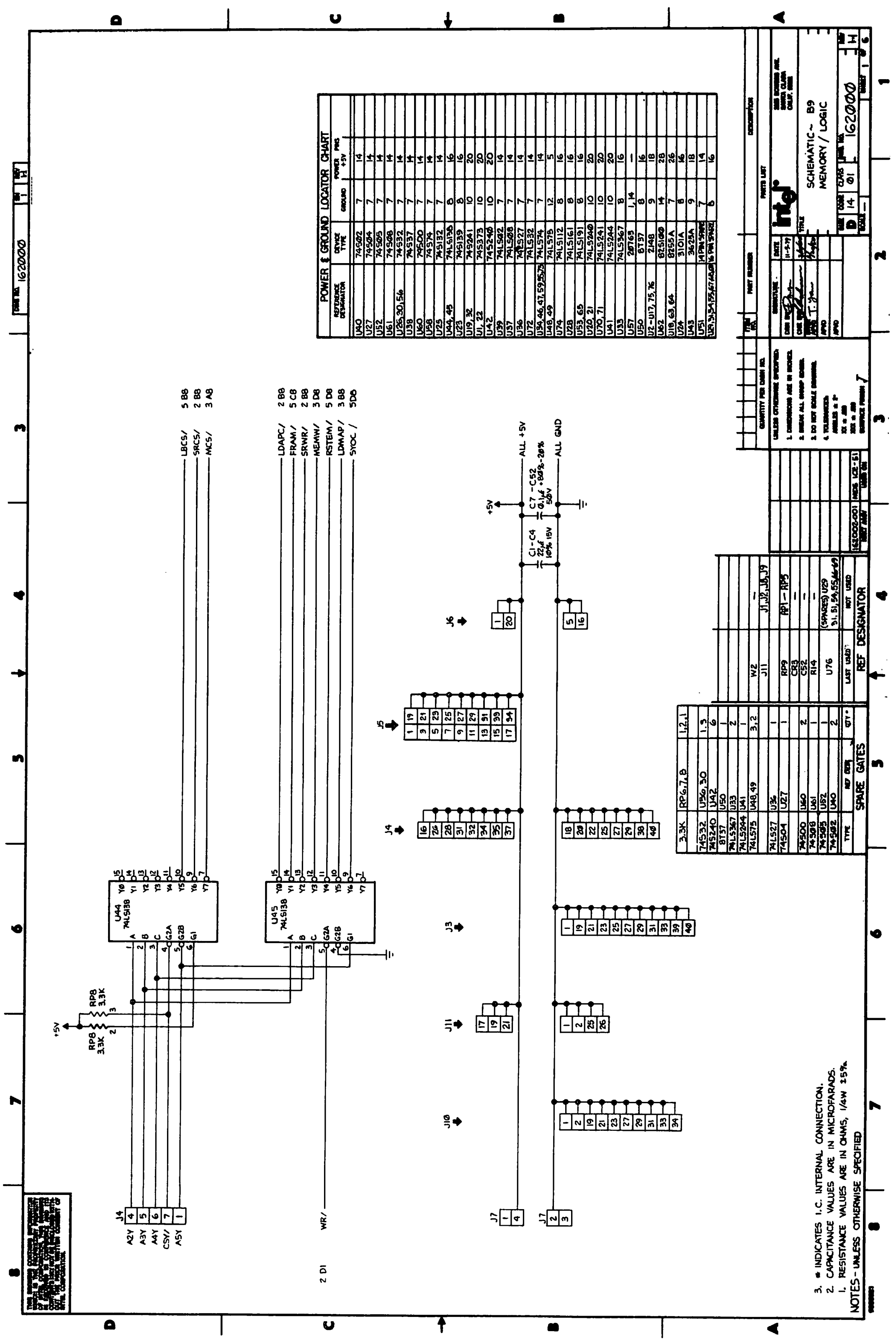
Trace Board Schematic (Sheet 4 of 6)

Trace Board Schematic (Sheet 5 of 6)

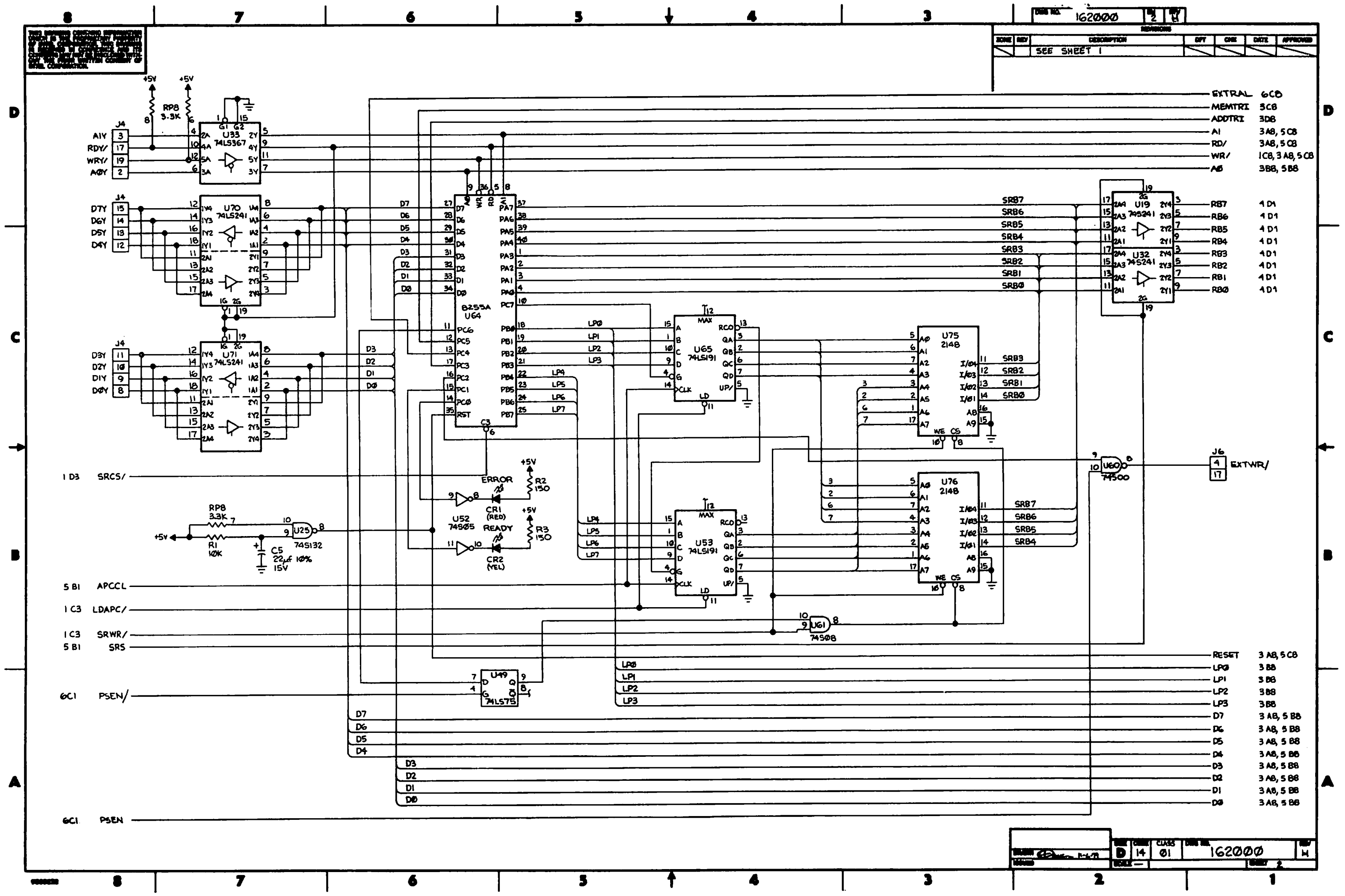




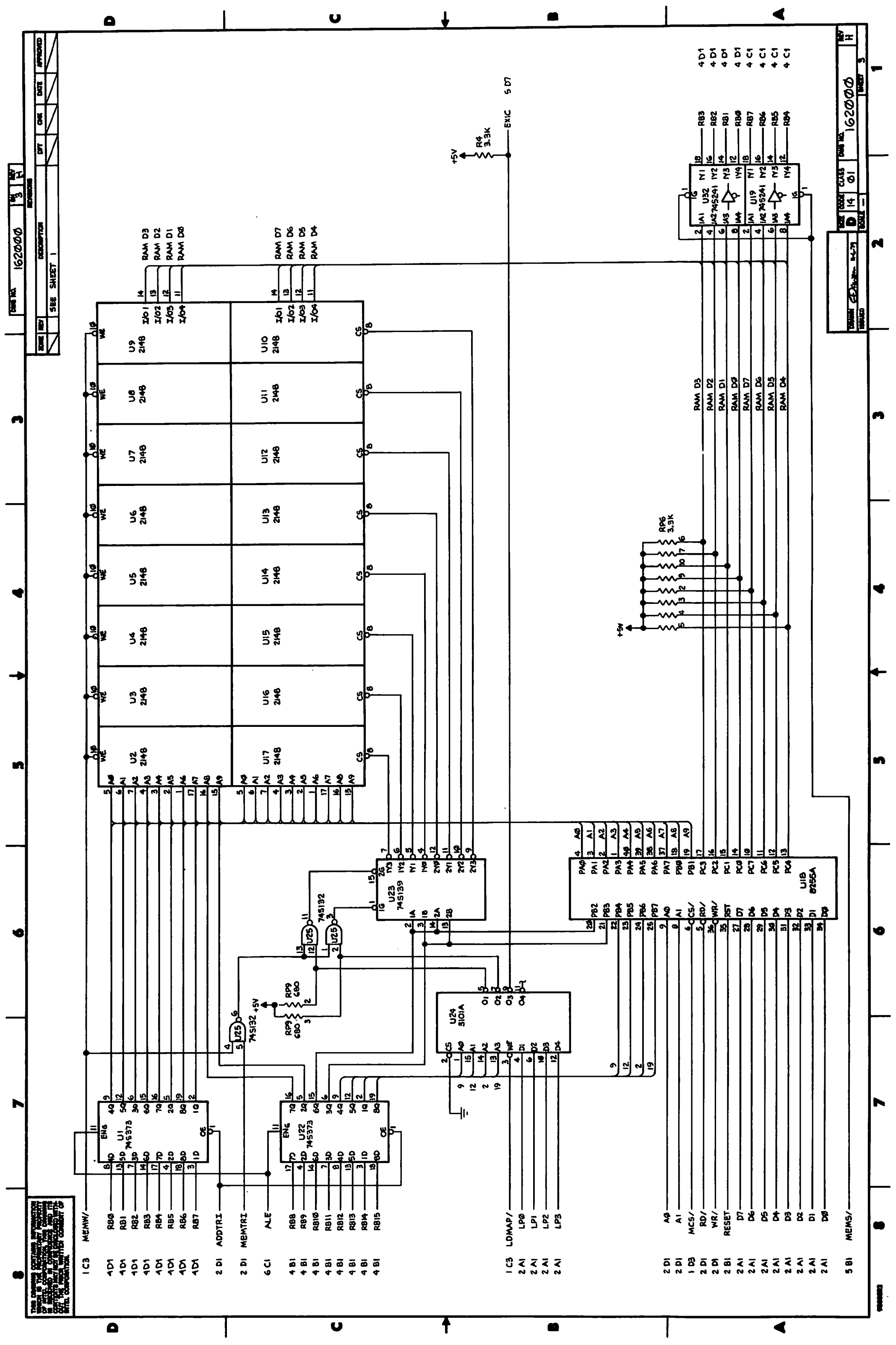
Trace Board Schematic (Sheet 6 of 6)


POWER & GROUND LOCATOR CHART
U44 74LS138
U45 74LS138
RP8 3.3K
LBCS/
SRCS/
MCS/
LDAPC/
FRAM/
SRWR/
MEMW/
RSTEM/
LDMAP/
SYOC/
WR/
A2Y
A3Y
A4Y
CSY/
A5Y
ALL +5V
ALL GND
C1-C4 22μf 10% 15V
C7-C52 0.1μf +80%-20% 50V
SPARE GATES
REF DESIGNATOR
SCHEMATIC - B9 MEMORY / LOGIC
162000
NOTES - UNLESS OTHERWISE SPECIFIED
1. RESISTANCE VALUES ARE IN OHMS, 1/4W ±5%.
2. CAPACITANCE VALUES ARE IN MICROFARADS.
3. * INDICATES I.C. INTERNAL CONNECTION.

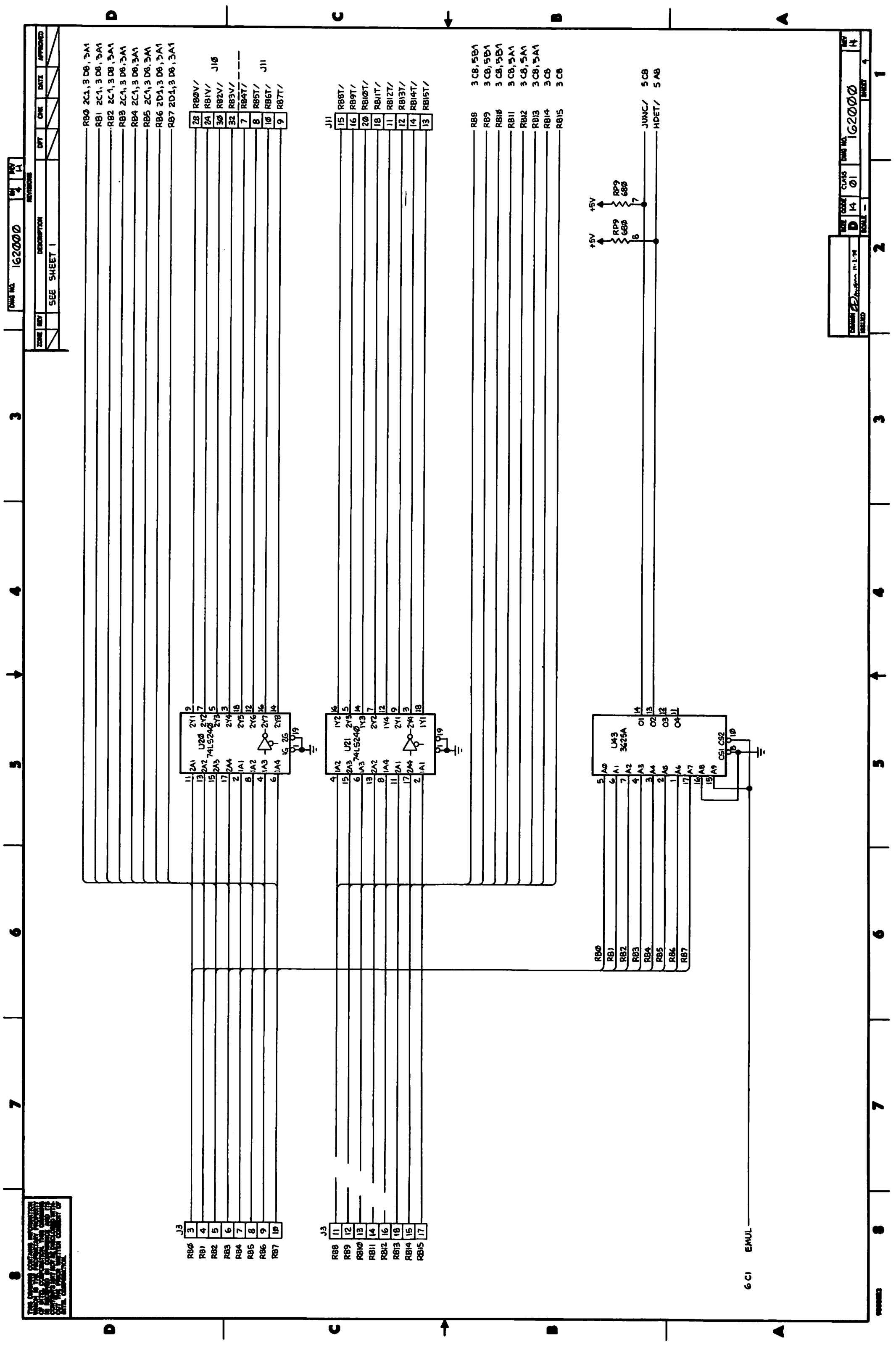
B9 Memory/Logic Schematic (Sheet 1 of 6)



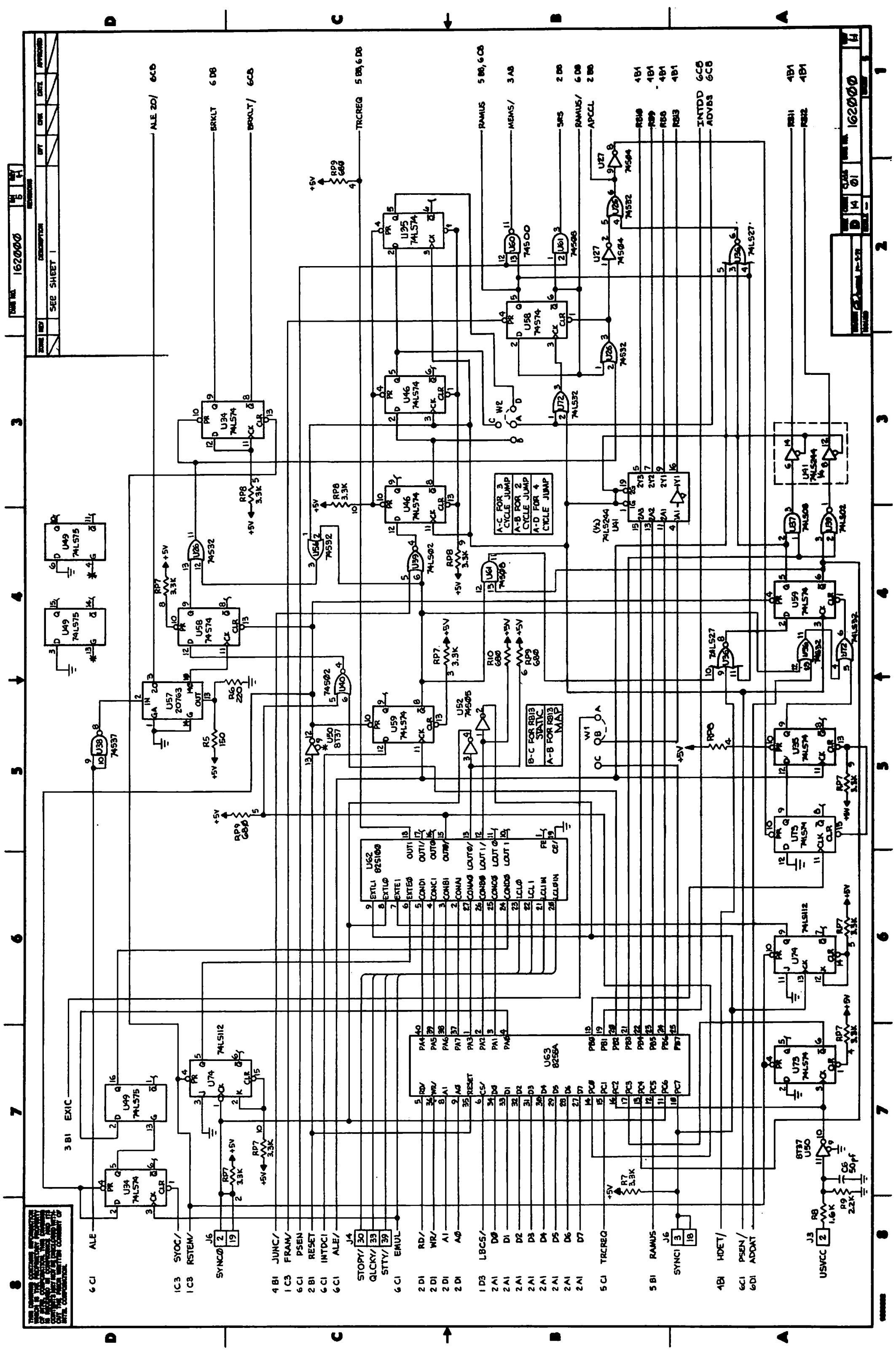
B9 Memory/Logic Schematic (Sheet 2 of 6)



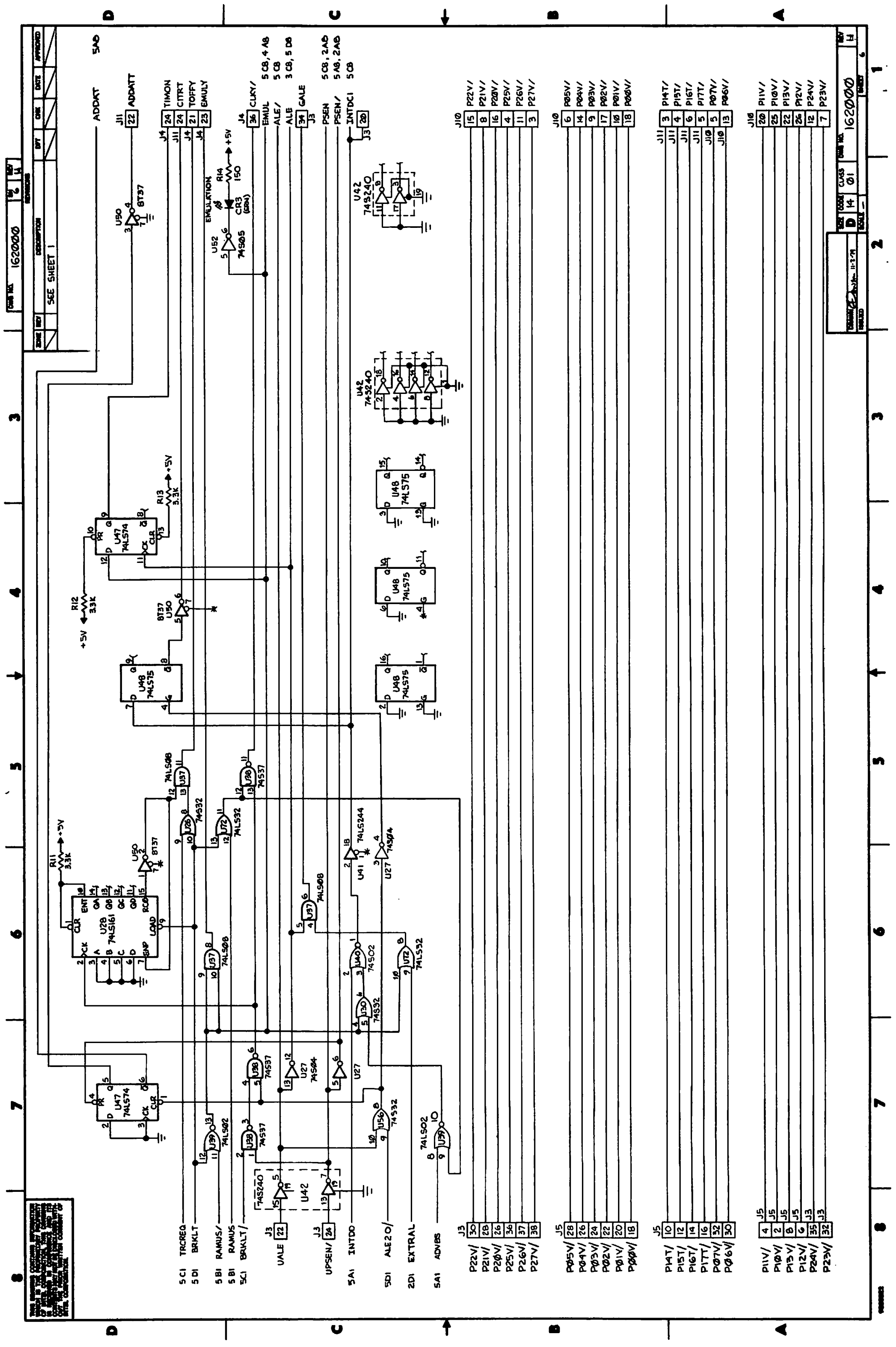
B9 Memory/Logic Schematic (Sheet 3 of 6)



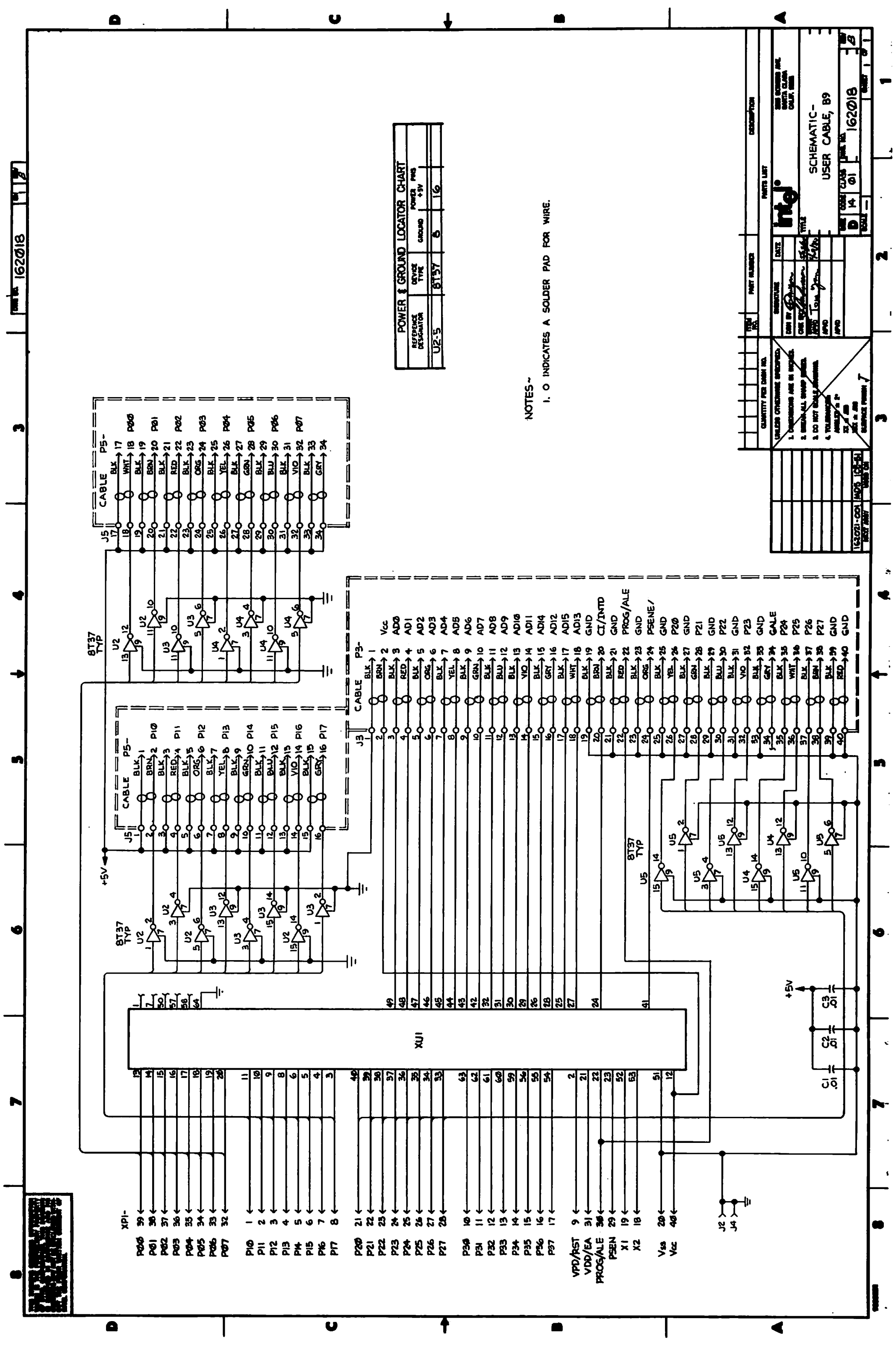
B9 Memory/Logic Schematic (Sheet 4 of 6)



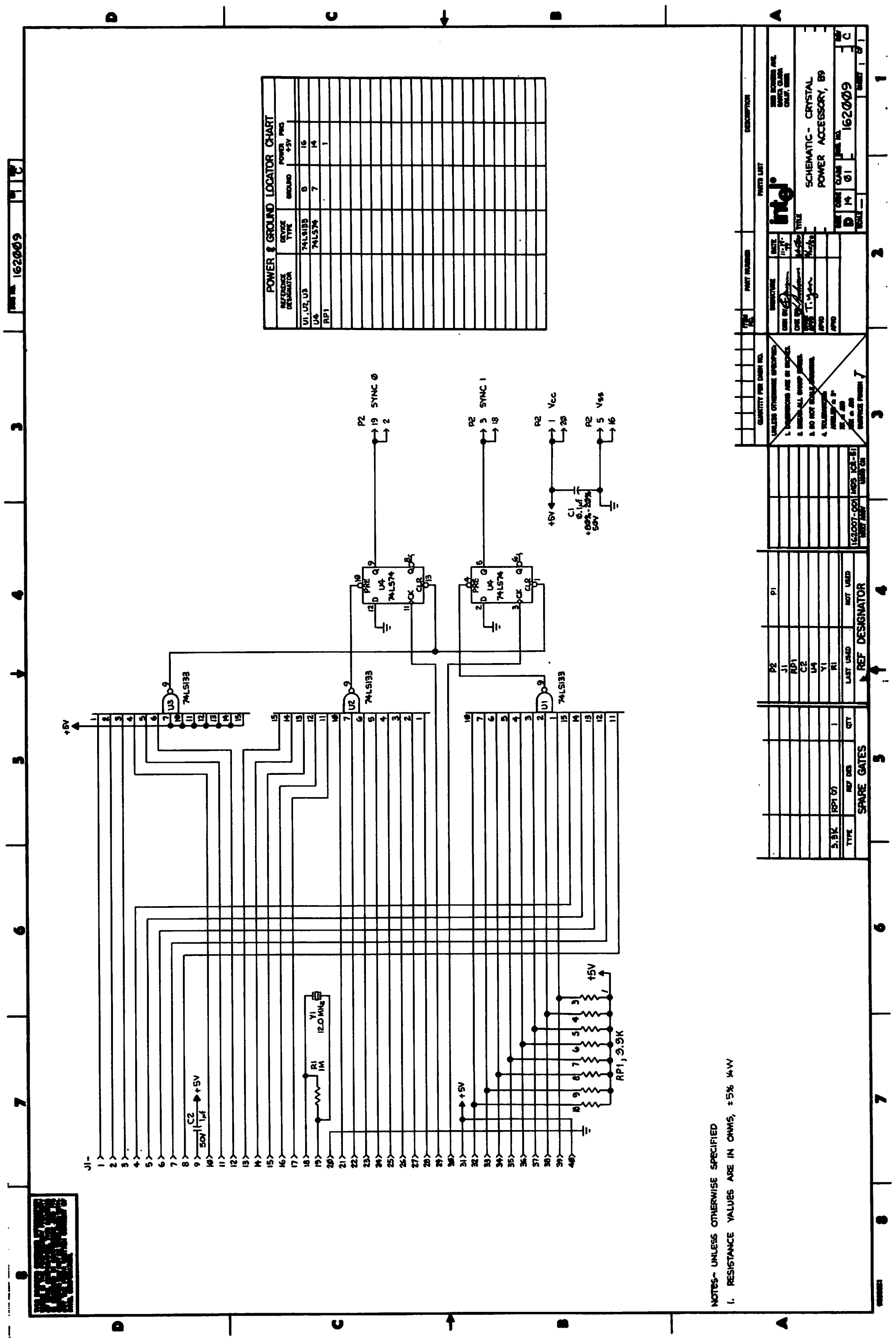
B9 Memory/Logic Schematic (Sheet 5 of 6)



B9 Memory/Logic Schematic (Sheet 6 of 6)



User Plug Schematic



Crystal Power Accessory Schematic

INDEX

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

M.C.S.O. Technical Publications

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

Printed in U.S.A.