CHAPTER 5

TRAP 14 HANDLER

An additional function contained within the MC68000 Educational Computer firmware is a function called the TRAP 14 handler. This function can be called by the user program to utilize various routines within TUTOR, to call special routines, and to return control to TUTOR. This chapter describes the TRAP 14 handler and how it is used.

CHAPTER 5

TRAP 14 HANDLER

5.1 WHAT IS THE TRAP 14 HANDLER?

The TRAP 14 handler is a function contained within TUTOR that allows system calls from user programs. The system calls can be used to access selected functional routines contained within the firmware, including ASCII/hex conversion, input routines, output routines, etc. The user is then not required to reproduce these functions in his own program.

5.1.1 Types of Exceptions

Exceptions are inputs to the MC68000 which change the "normal" flow of a program. They can be generated by either internal or external causes. There are three basic kinds of exceptions recognized by the MC68000:

a. Exceptions which cause the instruction currently being executed to be aborted. These consist of reset, bus error, and address error exceptions.

b. Exceptions which allow the current instruction to be completed before processing the exception. These are trace, interrupt, illegal instruction, and privilege violation exceptions. Illegal instructions and privilege violations cause the current instruction to be executed as an NOP instruction.

c. Exceptions which occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, DIVS, and DIVU (when dividing by zero) instructions are included in this group.

5.1.2 MC68000 Exception Processing

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the Supervisor mode (S) bit is asserted, putting the processor into the supervisor privilege state. Also, the Trace mode (T) bit is negated. For the reset and interrupt exceptions, the interrupt priority mask is also changed to match the level of the interrupt causing the exception.

In the second step, the vector number of the exception is determined. For interrupts other than auto-vectored interrupts, the vector number is obtained by a processor fetch from an external device, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked, using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction; however, for

bus error and address error, the value stacked for the program counter is unpredictable and may be incremented from the address of the instruction which caused the error. Additional information defining the instruction/operation causing the error is stacked for the bus error and address error exceptions.
The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution are started.

5.1.3 Trap 14 Handler

Traps are instructions which generate exceptions. The TRAP instruction can generate one of 16 exception vectors. Traps are useful for implementing system calls from user programs. The TRAP 14 handler within TUTOR serves this purpose. The TRAP 14 handler permits selected routines from TUTOR to be accessed by the user's target programs. In addition, it allows the user to append his own routines to the TRAP 14 handler and redefine the functions provided.

Up to 255 different functions can be accessed via the TRAP 14 handler. When using the TRAP 14 handler in TUTOR, the number of the desired function is passed to the TRAP 14 handler in the least significant byte of register D7. The handler uses this function number to find the address of the selected routine in a lookup table, and transfers control to that address. Most of the defined functions return to the user's program upon completion; the exceptions are function numbers 229 and 228, which return to TUTOR.

Of the 255 available functions, 127 are reserved by Motorola (numbered 128 through 254). It is suggested, therefore, that the user assign numbers 0 through 127 to user routines.

5.2 TRAP 14 CALLING SEQUENCE

The calling sequence is:

```
     MOVE.B   #<function number>,D7
     TRAP     #14
```

where <function number> is a number from 0 through 254, which represents the selected function. Calls to functions not defined result in the message 'UNDEFINED TRAP 14'; program control is passed to TUTOR.

The appropriate function number is placed in the least significant byte of register D7 before executing the TRAP instruction. A summary of the defined functions and the corresponding function numbers is shown in Table 5-1.

TABLE 5-1. TRAP 14 Function Summary

FUNCTION	FUNCTION NAME	FUNCTION DESCRIPTION
255	—	Reserved function - end of table indicator.
254	—	Reserved funciton - used to link tables.
253	LINKIT	Append user table to TRAP 14 table.
252	FIXDADD	Append string to buffer.
251	FIXBUF	Initialize A5 and A6 to 'BUFFER'.
250	FIXDATA	Initialize A6 to 'BUFFER' and append string to buffer.
249	FIXDCRLF	Move 'CR', 'LF', string to buffer.
248	OUTCH	Output single character to Port 1.
247	INCHE	Input single character from Port 1.
246	—	Reserved function.
245	—	Reserved function.
244	CHRPRINT	Output single character to Port 3.
243	OUTPUT	Output string to Port 1.
242	OUTPUT21	Output string to Port 2.
241	PORTIN1	Input string from Port 1.
240	PORTIN20	Input string from Port 2.
239	TAPEOUT	Output string to Port 4.
238	TAPEIN	Input string from Port 4.
237	PRCRLF	Output string to Port 3.
236	HEX2DEC	Convert hex value to ASCII encoded decimal.
235	GETHEX	Convert ASCII character to hex.
234	PUTHEX	Convert 1 hex digit to ASCII.
233	PNT2HX	Convert 2 hex digits to ASCII.
232	PNT4HX	Convert 4 hex digits to ASCII.
231	PNT6HX	Convert 6 hex digits to ASCII.
230	PNT8HX	Convert 8 hex digits to ASCII.
229	START	Restart TUTOR; perform initialization.
228	TUTOR	Go to TUTOR; print prompt.
227	OUT1CR	Output string plus 'CR', 'LF' to Port 1.
226	GETNUMA	Convert ASCII encoded hex to hex.
225	GETNUMD	Convert ASCII encoded decimal to hex.
224	PORTIN1N	Input string from Port 1; no automatic line feed.
223-128	—	Reserved.
127-0	—	User-defined functions.

5.3 TRAP 14 FUNCTIONS

There are five groups of functions defined by the TRAP 14 handler. These are:

a. Input/Output single character or character strings to or from I/O ports.

b. Conversion routines:

   Hex to decimal (ASCII format)
   Hex to ASCII - 1, 2, 4, 6, or 8 digits
   ASCII (one digit) to hex
   ASCII formatted hex to hex
   ASCII formatted decimal to hex

c. Buffer control routines.

d. Transfer control to TUTOR with/without performing initialization.

e. Routines to insert additional user functions into TRAP 14 lookup table.

NOTE: The expected convention when using the TRAP 14 handler is independent user and supervisor stacks.

5.3.1 Input/Output Functions

The Input/Output group of TRAP 14 functions includes routines to move information from/to the four available I/O ports to/from memory. They are useful for receiving commands from the terminal and displaying responses at the console and at the printer. Communication with a host is also possible.

There are five input routines which can be called via the TRAP 14 handler. A buffer string can be received from Port 1, 2, or 4. Input is not received from Port 3 because this port is typically connected to a printer. Single character input can also be received from Port 1. The four string input routines -- PORTIN1, PORTIN1N, PORTIN20, and TAPEIN -- receive input from Port 1, Port 1, Port 2, and Port 4, respectively. ASCII coded strings are typically used although this is not necessary.

The first three routines accept input until the ASCII code for a carriage return ($0D) is received signifying the end of the string. The last routine, TAPEIN, recognizes a line feed ($0A) as the end of string indicator. Because it is used exclusively with S-records, TAPEIN expects the first character of each string to be an ASCII 'S' ($53); characters prior to the 'S' are ignored.

All of the string input routines move characters from the appropriate port to a buffer pointed to by register A6. Before using a TRAP 14 call, the user must, in some cases, initialize parameters other than register D7. For the string input calls, A6 must be initialized to point to the next free location in the buffer where the characters will be stored. Register A5 must point to the start address of the buffer. A comparison is made betwen A5 and A6 each time a character is received, and the buffer size is not allowed to grow larger than 127 bytes (characters).

Upon completion, the routines PORTIN20 and TAPEIN leave A6 pointing to the last character in the buffer. PORTIN1 and PORTIN1N leave A6 pointing to the last character plus one (i.e., the next free location). All incoming bytes are masked to seven bits by ANDing them with the value $7F. Control characters (ASCII codes less than $20) are ignored by PORTIN20 and TAPEIN, while PORTIN1 and PORTIN1N ignore nulls ($00).

Both PORTIN1 and PORTIN1N will echo the characters back out to Port 1. The only difference between these two routines is that PORTIN1 will send both a carriage return and a line feed back to Port 1 upon receipt of only a carriage return. PORTIN1N sends only a carriage return. Table 5-2 summarizes the input functions.

Single character input is available only for Port 1. Using the routine INCHE, a single character is input from Port 1 and transferred to the lowest byte of register D0 (the remainder of D0 is unchanged). No additional parameters are required for this function other than the function number which is passed to the TRAP 14 handler in register D7. Upon completion, D0 contains the received character and A0 has the base address of the serial interface device, the MC6850, associated with Port 1. In addition, register D1 is used by INCHE and is not restored.

There are seven output routines corresponding to the five input functions with two additions. Both string and single character output is provided for Port 3; there are no corresponding input functions. The five string functions are OUTPUT, OUT1CR, OUTPUT21, PRCRLF, and TAPEOUT corresponding to Ports 1, 1, 2, 3, and 4, respectively;. Functions which send single character output to Ports 1 and 3 are named OUTCH and CHRPRINT, respectively.

In all of the string output routines, register A5 is used to point to the beginning of the string, and register A6 points to the byte immediately following the last byte in the string. The string that is outputted consists of all the bytes between the address contained in A5 and the one in A6, including the byte pointed to by A5 but not including the byte to which A6 points. In each case, with one exception, A5 is pointing to the last byte in the output string plus one when the output function is complete. The exception, PRCRLF, leaves register A5 unchanged.

Strings are generally collections of ASCII encoded characters. In the case of Port 4 (tape interface), the string are also generally in S-record format. OUTPUT, OUT1CR, and OUTPUT21 will echo the character being sent to their respective ports to Port 3 (printer) if it is attached (using the PA command). Any character sent to Port 3, by these functions or by PRCRLF, must be a printable ASCII character or a carriage return or line feed; all other characters are replaced by the ASCII code for a period ($2E) before they are sent to the printer. Printable characters include all ASCII codes between and including $20 through $7F.

The major difference between OUTPUT and OUT1CR is that OUT1CR appends a carriage return and line feed to the end of the string. None of the other output string functions append the CRLF to the end of the string. OUT1CR also destroys the contents of registers D0 and D1 and moves the base address of the Port 1 serial device into register A0. Additionally, the contents of register D0 are destroyed by PRCRLF. Refer to Table 5-3 for a summary of parameters required and registers used by these functions.

Two single character output functions can be called; OUTCH sends a character to Port 1 echoing it to Port 3 if it is attached, and CHRPRINT sends a character directly to Port 3. The character is generally ASCII coded and must be put in the lowest byte of register D0 before the output function is called. As with the string functions, any character sent to Port 3 must be a printable character or a carriage return or line feed.

OUTCH does not restore the contents of either D0 or D1. It initializes A0 to the base address of the Port 1 serial I/O device.

TABLE 5-2. Input Functions

FUNCTION NAME	FUNCTION NUMBER	PORT NUMBER	OUTPUT TYPE	TERMINATION CHARACTER	REGISTER A6 INITIAL	REGISTER A6 FINAL	REGISTER A5 INITIAL VALUE	OTHER REGISTERS FINAL VALUE
PORTIN1	241	1	String	CR = $0D	Next Free Loc.	Buf. End + 1	Buffer Start	---
PORTIN1N	224	1	String	CR = $0D	Next Free Loc.	Buf. End + 1	Buffer Start	---
PORTIN20	240	2	String	CR = $0D	Next Free Loc.	Buf. End	Buffer Start	---
TAPEIN	238	4	String	LF = $0A	Next Free Loc.	Buf. End	Buffer Start	---
INCHE	247	1	Single Char.	---	---	---	---	D0 = input char. A0 = ACIA #1 base address D1 destroyed

TABLE 5-3. Output Functions

FUNCTION NAME	FUNCTION NUMBER	PORT NUMBER	OUTPUT TYPE	REGISTER A6 INITIAL VALUE	REGISTER A5 INITIAL	REGISTER A5 FINAL	OTHER REGISTERS
OUTPUT	243	1	String	Buf. End + 1	Buf. Start	Buf. End + 1	---
OUT1CR	227	1	String	Buf. End + 1	Buf. Start	Buf. End + 1	D0, D1 destroyed A0-ACIA #1 base address
OUTPUT21	242	2	String	Buf. End + 1	Buf. Start	Buf. End + 1	---
PRCRLF	237	3	String	Buf. End + 1	Buf. Start	Buf. Start	D0 destroyed
TAPEOUT	239	4	String	Buf. End + 1	Buf. Start	Buf. End + 1	---
OUTCH	248	1	Single Char.	---	---	---	D0, D1 destroyed A0-ACIA #1 base address
CHRPRINT	244	3	Single Char.	---	---	---	---

5.3.2 Conversion Functions

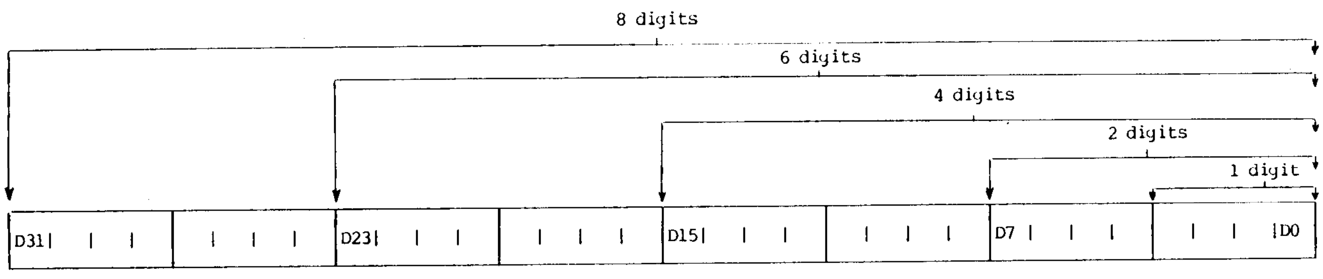
The conversion functions can be divided into two groups:

- Hex conversion routines which convert hexadecimal numbers to ASCII encoded decimal or ASCII encoded hex.
- ASCII conversion routines which convert ASCII encoded numbers to either decimal or hexadecimal.

Both groups are needed to interface with the various peripherals connected to the I/O ports. Most require all characters to be ASCII encoded.

There are five different routines to convert hex numbers to ASCII encoded hex digits, which are then transferred to a buffer. One, two, four, six, or eight hex digits can be converted to ASCII, depending on which of the five hex to ASCII conversion functions -- PUTHEX, PNT2HX, PNT4HX, PNT6HX, PNT8HX -- is used. Register A6 must point to the buffer where the character(s) will be stored. A6 is incremented by 1, 2, 4, 6, or 8, depending on the number of digits converted.

The hex number to be converted is passed to the conversion routine in register D0 (right-justified using four bits per digit); anywhere from four to 32 bits of D0 may be required, depending on the number of digits to be converted.



Register D0

Each digit requires four bits; up to eight digits can be represented. The most significant digit is converted and placed in the buffer first, followed by the other digits in descending order. All five routines destroy the contents of register D0. The contents of registers D1 and D2 may or may not be destroyed, depending on the function (see Table 5-4 for more details).

HEX2DEC converts the 8-digit hex number found in register D0 to the equivalent decimal number. This decimal number is converted to ASCII-encoded digits and placed in the buffer. All leading zeros are suppressed when the number is transferred to the buffer. The final value of register A6 (the buffer pointer) is therefore unpredictable; its value depends on the number of non-zero digits in the number. Specific information on this function is shown in Table 5-4.

The six conversion functions discussed thus far convert hex numbers to ASCII. The last three conversion functions are used to convert ASCII characters to hex or decimal digits. GETNUMA and GETNUMD take ASCII encoded numbers from a buffer pointed to by register A5 and convert them to hex and decimal digits, respectively. If the number of digits is too large to be represented in 32 bits or if an inappropriate digit is received (i.e., a digit which is too large for the base or is not a digit at all), the conversion will be aborted and an error message displayed at the terminal. Otherwise, the conversion will continue until all digits in the buffer have been processed. Register A6 should point one byte past the last digit to indicate the end of the buffer. The resultant value is returned in register D0; all 32 bits are used. A5 is left pointing one byte past the last digit.

TABLE 5-4. Hex Conversion Routines

FUNCTION NAME	FUNCTION NUMBER	NUMBER OF HEX DIGITS CONVERTED	PORTION OF D0 USED FOR DIGIT(S)	FINAL VALUE OF A6	REGISTERS USED AND NOT RESTORED
PUTHEX	234	one	bits 3-0	A6 init + 1	D0
PNT2HX	233	two	bits 7-0	A6 init + 2	D0,D2
PNT4HX	232	four	bits 15-0	A6 init + 4	D0,D1,D2
PNT6HX	231	six	bits 23-0	A6 init + 6	D0,D1,D2
PNT8HX	230	eight	bits 31-0	A6 init + 8	D0,D1,D2
HEX2DEC	236	eight	bits 31-0	*	D0

* unpredictable -- depends on original hex value

The final conversion function, GETHEX, converts the ASCII character in the lowest byte of register D0 to hex and returns the hex number in the same byte. If the ASCII code does not represent a valid hex digit, an error message is generated. The ASCII conversion routines are summarized in Table 5-5; all registers not included in the table are unchanged.

5.3.3 Buffer Control Functions

The buffer control functions are useful in conjunction with the various input and output functions which are available. The control functions are used primarily to initialize buffer pointers and to move ASCII strings into an output buffer.

FIXBUF initializes registers A5 and A6 to point to 'BUFFER'. This is a buffer within the system RAM and is used by TUTOR. No other registers are altered and no parameters are required. FIXBUF can be used to initialize A5 and A6 prior to calling the string input functions. Usage of this buffer while tracing or with breakpoints will produce erroneous results.

FIXDADD, FIXDATA, and FIXDCRLF are all used to transfer an ASCII string to a buffer and are summarized in Table 5-6; all registers not included in the table are unchanged. The first two, FIXDADD and FIXDATA, move a string pointed to by register A5 into the buffer. FIXDADD requires that register A6 point to the buffer, while FIXDATA always moves the string to the location in system RAM called 'BUFFER'. The ASCII end of transmission character (EOT = $04) is used to indicate the end of the string. It must be the last character in the string, but is not moved to the buffer. Registers A5 and A6 are left pointing to 'BUFFER' and the buffer end + 1, respectively.

FIXDCRLF is quite similar to FIXDATA -- a string pointed to by register A5 is moved to 'BUFFER'. The difference is that FIXDCRLF transfers the ASCII code for a carriage return and line feed to the beginning of the buffer before transferring the string. The termination character ($04) and parameters returned are the same.

TABLE 5-5. ASCII Conversion Routines

FUNCTION NAME	FUNCTION NUMBER	REGISTER A5 INITIAL	REGISTER A5 FINAL	REGISTER A6 INITIAL	REGISTER A6 FINAL	REGISTER D0 INITIAL	REGISTER D0 FINAL
GETNUMA	226	buffer start	buffer end + 1	buffer end + 1	---	---	hex value 32 bits
GETNUMD	225	buffer start	buffer end + 1	buffer end + 1	---	---	hex value 32 bits
GETHEX	235	---	---	---	---	ASCII char 8 bits	hex value 4 bits

TABLE 5-6. Buffer Control Functions

FUNCTION NAME	FUNCTION NUMBER	FUNCTION DESCRIPTION	PARAMETERS REQUIRED	PARAMETERS RETURNED
FIXDADD	252	Append a string to buffer; $04 signifies end of string	A5 - String start A6 - Next location in buffer	A5 - 'BUFFER' A6 - Buffer end + 1
FIXDATA	250	Same as FIXDADD but initializes A6 to 'BUFFER' before moving data	A5 - string start	A5 - 'BUFFER' A6 - Buffer end + 1
FIXBUF	251	Initializes A5 and A6 to 'BUFFER'	None	A5 - 'BUFFER' A6 - 'BUFFER'
FIXDCRLF	249	Move CR, LF, string to 'BUFFER'; $04 signifies end of string; initializes A6 to 'BUFFER' before moving data	A5 - string start	A5 - 'BUFFER' A6 - Buffer end + 1

5.3.4 Transfer Control to TUTOR

Using TRAP 14, control can be transferred from a target program to TUTOR in two ways. One path uses the TRAP 14 function START. After control is passed to TUTOR, the restart initialization procedure is executed. The stack pointer, register A7, is initialized to 'SYSSTACK'. The exception vectors, located in the lower portion of the system RAM, are initialized. A portion of the upper system RAM is zeroed. The status register is set to $2700 — interrupt mask level 7 and supervisor state. The prompt is then sent to Port 1.

Using the TRAP 14 function TUTOR, the other path performs only a portion of the initialization described above. The stack pointer and status register are initialized and the prompt is displayed. See Table 5-7.

TABLE 5-7. Transfer Control to TUTOR

FUNCTION NAME	FUNCTION NUMBER	DESCRIPTION	REGISTERS AFFECTED
START	229	Restart TUTOR Perform Initialization	Status Register=$2700 A7='SYSSTACK', Program Counter=0
TUTOR	228	Go to TUTOR Print Prompt	Status Register=$2700 A7='SYSSTACK'

NOTE: Calling START or TUTOR from the user state will result in a privilege violation because they write to the status register.

5.3.5 Inserting Additional Functions

User-defined functions can also be called using the TRAP 14 handler. The TRAP 14 handler uses the function number and a lookup table to determine the address of the selected function. User functions are included by inserting additional lookup tables. The format for entries in these tables is $UUSSSSSS, where UU is the function number in hex -- $0 through $7F for user-defined functions — and SSSSSS is the starting address of the function in hex. Each entry in the table requires one long word (32 bits).

The table is inserted quite simply; the TRAP 14 function LINKIT, function number 253, performs the insertion. Register A0 must point to the lookup table to be inserted when the TRAP instruction is executed. The new table is, in effect, placed in front of the old lookup table. However, since the new table is not addressed immediately in front of the old table (i.e., old table is in ROM and new table is in RAM), a link must be provided in the new table to connect it with the old table. This is accomplished by making the starting address of the old table preceded by $FE the last entry in the new table. The format is $FETTTTTT, where TTTTTT is the pointer to the old table. This is easily accomplished because LINKIT returns the required long word ($FETTTTTT) in register A0. The target program should store this value at the end of the new lookup table immediately after executing the TRAP 14 instruction. The following is an example of the procedure used to link a user-defined table. Location NEWTBL is the beginning of the user table and location ENDTBL is the end of the user table. After executing the TRAP instruction, the pointer to the old table must be saved at the end of the new table.

NOTE: Labels NEWTBL and ENDTBL are used here for clarity but will not, of course, be accepted by the interactive assembler.

```
         LEA       NEWTBL,A0          Register A0 points to new table
         MOVE.B    #253,D7            LINKIT
         TRAP      #14
         MOVE.L    A0,ENDTBL          A0 contains $FETTTTTT
           .                          where TTTTTT points to old table
           .
           .

NEWTBL   DC.W      $UUSS
         DC.W      $SSSS
         DC.W      $UUSS
         DC.W      $SSSS
           .
ENDTBL   DC.W      $XXXX              Link to old table will be stored here
         DC.W      $XXXX
```

LINKIT is summarized in Table 5-8. Additional tables can be inserted in the same way.

Each user-defined function requires a user-written software routine located somewhere within the user RAM. These routines should use an RTS (return from subroutine) instruction, not an RTE (return from exception) instruction, to return to the calling program. The TRAP 14 handler jumps to the user software routine, leaving only the program counter on the stack. The RTE instruction expects both the status register and the program counter to be on the stack. NOTE: The user must make sure that the stack pointers are pointing to locations within the user RAM and that the two stacks do not overlap before executing a TRAP 14 instruction.

TABLE 5-8. Inserting Additional Functions

FUNCTION NAME	FUNCTION NUMBER	DESCRIPTION	REQUIRED PARAMETERS	REGISTERS AFFECTED
---	255	Reserved - End of Table Indicator $FFXXXXXX - must be last entry in last table	---	---
---	254	Reserved - Use to link one table to another	---	---
LINKIT	253	Insert User Table	A0 - start add of table to be inserted	D0 - destroyed A0 - start add of old table

Because function numbers are compared with entries in the user table(s) first, it is possible to redefine function numbers which have been defined by TUTOR (i.e., 128 through 253; function numbers 254 and 255 cannot be redefined). For example, if a user does not have a host system and has nothing else tied to Port 2, that user may not require the Port 2 I/O functions numbered 242 and 240. The user can redefine the functions associated with these numbers by placing the appropriate function number and the address of the new routine ($UUSSSSSS) in the user function table. The user table will be searched first and the entry in the other table will be ignored.

Function number 255 is described as a reserved function in Table 5-1. Although this function is not called by target programs, it nevertheless performs a useful function as an end-of-table indicator. All 255 available function numbers may not be assigned to a function and, therefore, will not be included in the lookup table(s). Thus, it is possible to attempt to call a nonexistent function by using an unassigned function number. In such a situation, a match will not be found in the table(s). Whatever follows the table(s) in memory -- program, data, random bytes -- will automatically be used to extend the table in an attempt to find a match. A bus or address trap error is the usual end point of this sequence, although significant damage may occur prior to the trap error.

To avoid these problems, the end-of-table function number should always be the last entry in the last lookup table. Usually this is the table provided by TUTOR. An address is not required for the end-of-table entry; only the function number is needed. The format is $FFXXXXXX, where the X's represent 'don't care' characters. This entry is included as the last long word of the TRAP 14 lookup table provided by TUTOR. When the end-of-table entry is reached, the message 'UNDEFINED TRAP 14' is sent to Port 1 and control is transferred to TUTOR.