Powerful trace facility for the MC68000 microprocessor system

Trace facilities can be useful debugging aids. **S M Said and K R Dimond** present a package for the 68000

Software providing a trace facility for 68000-based systems has been written and installed in a monitor program. The commands in the software package are of two types: for the unconditional single-step function, and for the conditional trace function. Both types of command are described. The package is intended for use in nonrealtime applications only.

microsystems system debugging 68000

The advantages of using a trace facility as a debugging aid have long been acknowledged. Many computer systems now provide such facilities for high-level languages. However, at the machine level these functions are less common. Generally, 8-bit microprocessor systems use either software trace[1] or hardware circuitry[2] to provide these facilities.

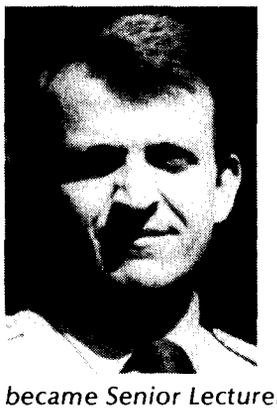
Most monitor programs written for the MC68000 microprocessor system[3] have a form of trace command. This command executes one instruction under user control and then prints the contents of the processor's registers on the screen. Initialization of the registers is also available. The package described in this paper, however, goes further.

- It allows the user to initialize the trigger conditions. Thus the user can trace automatically through a program until the trigger conditions (any combination of registers and memory contents) are met.
- It enables the user to define the parameters to be displayed (registers or memory contents) when the trigger parameters are valid.

The trace facility package described has been written in M68KASM assembler, and installed in a monitor program for use on the 16-bit 68000 microprocessor. The package does not use any of the processor interrupts nor require any hardware modifications.

Electronics Laboratory, University of Kent at Canterbury, Kent CT2 7NT, UK

S M Said was born in El-Fayom, Egypt, in 1949. He received his BSc in electrical engineering from the Military Technical College of Cairo, Egypt, in 1971. Between 1971 and 1976 he was involved in several communications projects for the Egyptian Army. Then from 1977 to 1981 he was a lecturer at the Military College. Since 1982 he has been working at the University of Kent at Canterbury, UK, for his PhD in the application of 16-bit microprocessors in digital communication.



Keith Dimond graduated in electrical engineering from the University of Manchester Institute of Science and Technology, UK, where he also undertook research. He was then a scientific officer at Government Communications Headquarters, Cheltenham, UK. Dimond joined the Electronics Laboratory at the University of Kent in October 1971. He became Senior Lecturer there in 1981. His interests are in CAD tools for digital and system design.

0141–9331/85/01003–05 $03.00

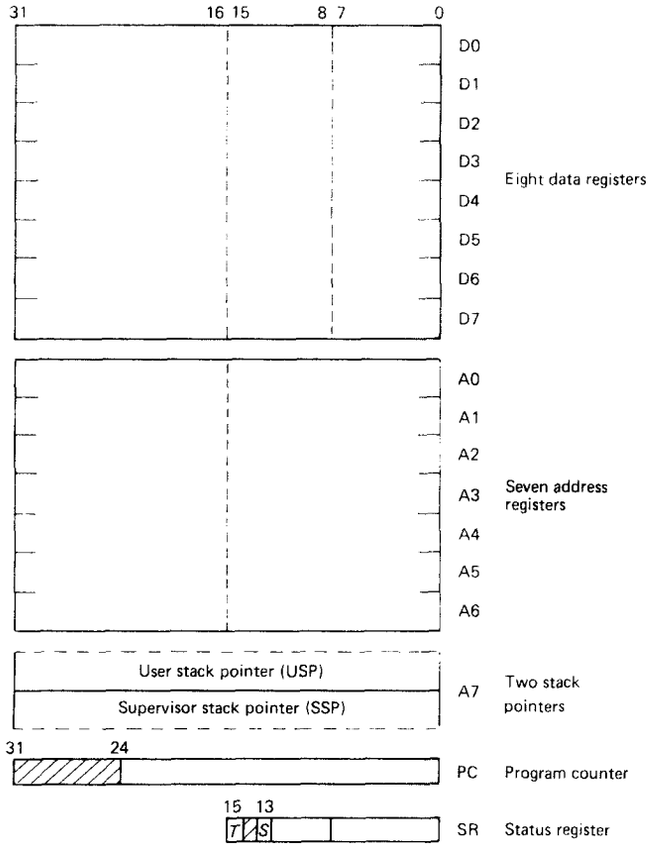


Figure 1. MC68000 programming model

REGISTER STRUCTURE OF THE PROCESSOR

The 68000 offers eight data registers, seven address registers, two stack pointers (one for user programs and the other for supervisor programs), a program counter and a status register. Figure 1 shows the processor's internal registers.

To support multiuser and multitasking applications the 68000 operates in two different states: a user state for normal functions and a supervisor state for system control (eg an operating system). In the status register, bit 13(S) indicates whether the processor is operating in the supervisor state ($S = 1$) or the user state ($S = 0$). When the system is in the supervisor state, all machine code instructions can be executed without any restrictions. However, when the system is in the user state some privileged instructions cannot be executed (eg instructions involving the manipulation of the system stack pointer).

The 68000 has an integral single-step facility, controlled by bit 15 (T) of the status register. When the T bit is set to 1, the 68000 will execute one program instruction. After this instruction is terminated, the processor will enter the supervisor state ($S = 1$), switch off the trace mode ($T = 0$) and then proceed to execute a program whose starting address is stored in the trace exception vector[4].

SOFTWARE IMPLEMENTATION

The trace facility software is based on the processor's single-step ability to provide the user with two useful debugging functions

- unconditional single-step command
- conditional trace command

User's view of the system

Since the trace package has been designed to be added to the system monitor, the commands have a similar format. The commands needed to use the package are

- for the unconditional single-step command
  SS : single step
- for conditional trace command
  ID : initialize display parameters
  IT : initialize trigger parameters
  KD : kill display parameters
  KT : kill trigger parameters
  PP : print parameters
  TR : trace command

UNCONDITIONAL SINGLE-STEP COMMAND

This function permits a user program to be executed one instruction at a time. The contents of the processor's registers are displayed on the screen after each instruction has been executed.

As an understanding of this command is essential for comprehending the trace command, the flowchart of the single-step program is shown in Figure 2. The detailed procedure is as follows.

Stage 1 When the single-step command is entered, a routine is called to set bit 15 ($T = 1$) and to clear bit 13 ($S = 0$) of a reserved memory location (XSR). This memory location will be subsequently loaded into the processor's status register (SR).

Stage 2 Another routine is then called to load the processor's registers from a reserved memory area as shown in Figure 3. As these memory locations have been previously initialized by the user, the single-step starting conditions are defined by the user. In other words, the executed instruction will be fetched from an address contained in the memory location XPC. The

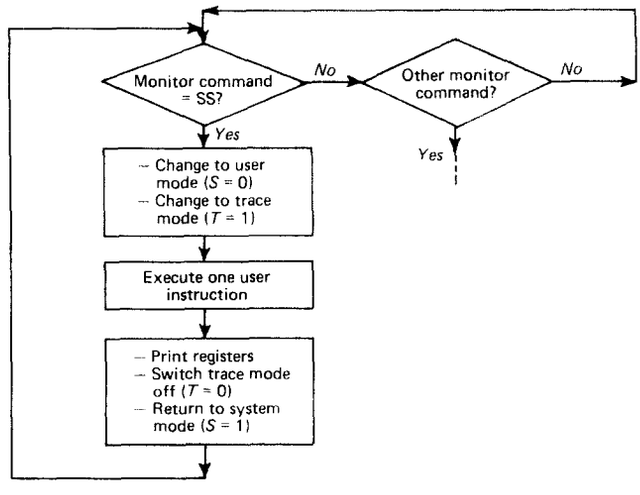


Figure 2. Flowchart of the single-step process

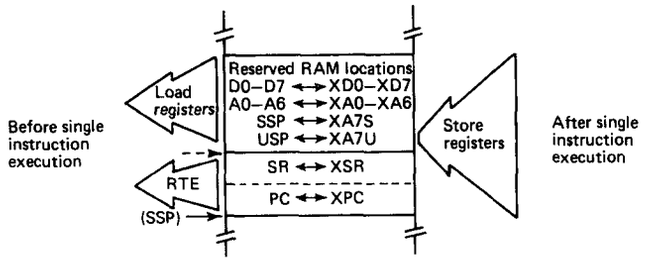


Figure 3. Reserved memory locations for register values

routine will initialize the system stack pointer (SSP) to point at the memory location XSR, and then execute a return-from-exception (RTE) instruction. This instruction pulls both the status register (SR) and the program counter (PC) values from XSR and XPC in the supervisor stack as shown in Figure 3.

Stage 3 The instruction is now pointed to by the program counter and is executed. After the termination of the single instruction, the processor traps through the trace exception vector[4]. The trace exception vector contains the address of another routine which again pushes all the register values back into the reserved memory area. It then clears bit 15 in the memory location XSR and also loads the address of the next routine to be executed into the memory location XPC. The system stack pointer (SSP) is initialized again to point at the memory location XSR, and executes an RTE instruction. This instruction will load both the status register (SR) and the program counter (PC) from the memory locations XSR and XPC respectively.

Stage 4 The final routine to be executed (pointed at by XPC in stage 3) will print all the memory locations containing register values as a result of the single executed instruction. Control is then returned to the system monitor to wait for the next command.

CONDITIONAL TRACE COMMAND

A single-stepping facility such as that described above will trace each instruction executed. In cases where the number of instructions being executed is small, such a facility is very useful. However, if the number of instructions executed is large it can be rather cumbersome. In this case, what is required is a mechanism whereby the tracing facility can be switched on and off automatically.

The term 'conditional trace' refers to the ability of this command to single step through a program, and on each step to perform comparisons between preinitialized registers and/or memory values (trigger values) and the current program values. When the trigger values are met, the trace program prints the registers and/or the memory parameters defined previously.

Initialization of parameters

To use the tracing facility, the user must first initialize the trace parameters. There are two groups of parameters that must be set up before the trace command can be used

- display parameters
- trigger parameters

Display parameter format

The user can interactively define any combination of registers and/or memory locations to form a display parameter. This parameter can be displayed at any time or after the trace trigger conditions have been met.

The display parameters needed are stored in a buffer. The mechanism of display initialization is shown in Figure 4, where the display buffer (DISBUF) has a pointer (DISPTR) and works in conjunction with two other tables: the string table (STRTBL) and a subroutine table (SUBTBL1). The software represents each display parameter as a set of numbers according to its type as follows.

- *Index value* is an offset value which is added to a pointer register that has been initialized previously to point at the start of either the SUBTBL1 or the STRTBL table. Therefore the resultant value of the pointer register will point to either the starting address of a string in the STRTBL table or a subroutine to be executed in the SUBTBL1 table.
- The *address* parameter is only inserted in the buffer table (DISBUF) if the display parameter is a memory location.

After storing all the display parameters, the software inserts a terminator at the end of the buffer DISBUF. Figure 4 shows an example of two different display parameters stored in the display buffer.

Trigger parameter format

As in the display parameter initialization, any combination of registers and/or memory locations can form the trace trigger condition. These parameters are stored in a trigger buffer (TRGBUF). As shown in Figure 5, the buffer has a pointer (TRGPTR), which works in con-

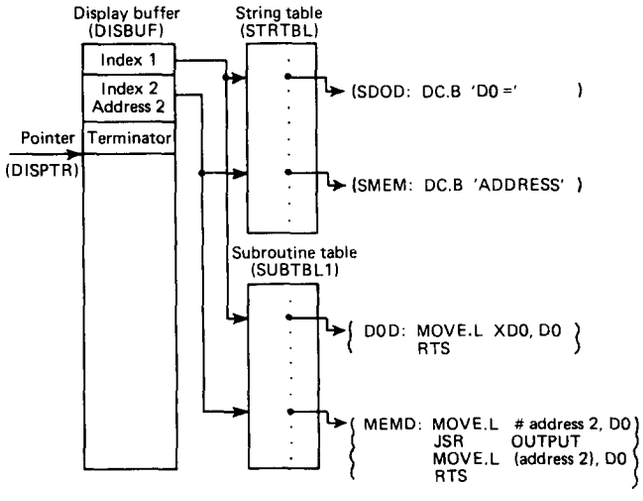


Figure 4. Display buffer (DISBUF) initialized with two conditions: the processor register (D0) and a memory location

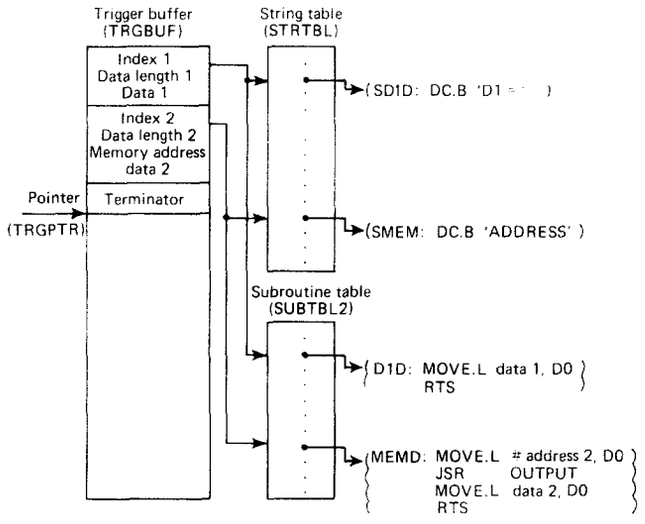


Figure 5. Trigger buffer (TRGBUF) initialized with two conditions: the processor register (D1) and a memory location

junction with the two tables STRTBL and SUBTBL2. Each trigger parameter is represented as follows.

- *Index value* is an offset value from the start of either the STRTBL or the SUBTBL2 table, as with the display parameter.
- *Data length* identifies the data length (byte, word or long word).
- *Data value* shows what the data trigger value is.
- *Address* is only inserted if the memory location is a trigger parameter.

Again, a terminator is used to identify the buffer end in the same way as for the display buffer.

Kill parameters

The KP command enables the user to remove previously defined parameters. As mentioned before, both the display parameters and the trigger parameters are stored in buffers (DISBUF and TRGBUF respectively) ended with a terminator. The reason for using this terminator is to simplify the process of adding new parameters or killing old ones. New parameters can be added to the buffers simply by moving the terminator down, while moving the terminator to the top of the buffer will remove all the previous parameters.

The two kill commands mentioned above enable either the display parameters or the trigger parameters to be deleted. This is done by inserting the terminator at the beginning of either DISBUF or TRGBUF.

Print parameters

The print command is used either as a command by the user or as a subroutine by the system.

When employed by the user the command acts as an interactive monitor of the preinitialized conditions. Once the display and trigger parameters have been initialized, the user can display them at any time using the 'PP' command.

The command is employed by the system as a result of a successful trigger condition occurring. Thus when the trigger values are met the display parameters are printed automatically.

To display all the initialized conditions, the print routine interprets the stored information in both the display and trigger buffers into printed messages. It begins with the display buffer DISBUF, moves its pointer DISPTR to the start, and reads the buffer elements. Using the processor's indirect addressing mode with index and offset, the print routine uses the index values stored in the buffer to form two pointers. As explained in the listing shown in Figure 6, the two pointers will point to the addresses of a string to be printed in the STRTBL table and a subroutine to be executed from the SUBTBL1 table. The pointer DISPTR is then moved to the next buffer element to get the index value and to reform the two pointers. This process is repeated until the buffer terminator is reached. The complete sequence is then repeated using the trigger buffer TRGBUF. The output format of the examples described in Figures 4 and 5 is

Trace Display Parameters:
D0 = 00001200 Address 00100000 = 0000FFF1
Trace Trigger Parameters:
D1 = FFFFFFFF Address 0002F000 = 000000FF

Trace command

As mentioned above, this command enables the user to trace through program instructions until the predefined conditions are met.

The flowchart of the trace command is shown in Figure 7. When this routine is entered, it changes to the user state and executes one user instruction as explained before in the unconditional single step. After storing all register values in memory, it starts to test the trigger conditions. This trigger testing is performed by comparing the trigger values in the trigger buffer TRGBUF with the current values. The testing time is minimized as follows: if a trigger condition is not met, the rest of the conditions are not tested; if the tested condition is valid, it proceeds to check the next entry in the buffer table. This process of single stepping and checking of trigger conditions is repeated until the trigger values are met.

As the trace process takes many instructions, the following points were taken into account.

- The search process for the trigger conditions should be as fast as possible. This is achieved by using the powerful addressing modes of the processor for fast access of both the trigger buffer and the subroutine table.

SUBROUTINE (PRINT) TO PRINT DISPLAY AND TRIGGER PARAMETERS

```
            LEA     STRTBL, A2      * STRING TABLE POINTER
            LEA     SUBTBL1, A4     * SUBROUTINE TABLE POINTER
            LEA     DISBUF, A5      * DISPLAY BUFFER POINTER
PRINTDIS:   MOVE.L  (A5) +, D3      * GET INDEX VALUE
            CMP.L   #SFF, D3        * TERMINATOR CODE ?
            BEQ     PRINTTRG        * GO TO PRINT TRIGGER VALUE
            MOVEA.L 0 (A2, D3.L), A0 * GET STRING ADDRESS
            JSR     STROUTPUT       * PRINT STRING
            MOVEA.L 0 (A2, D3.L), A0 * GET SUBROUTINE ADDRESS
            JSR     (A0)            * EXECUTE ROUTINE
            JSR     OUTPUT          * PRINT VALUE IN D0
            BRA     PRINTDIS        * NEXT DISPLAY PARAMETER
PRINTTRG:   LEA     TRGTBL, A5      * TRIGGER BUFFER POINTER
              .         .
              .         .
              .         .
```

Figure 6. Listing of part of the print routine

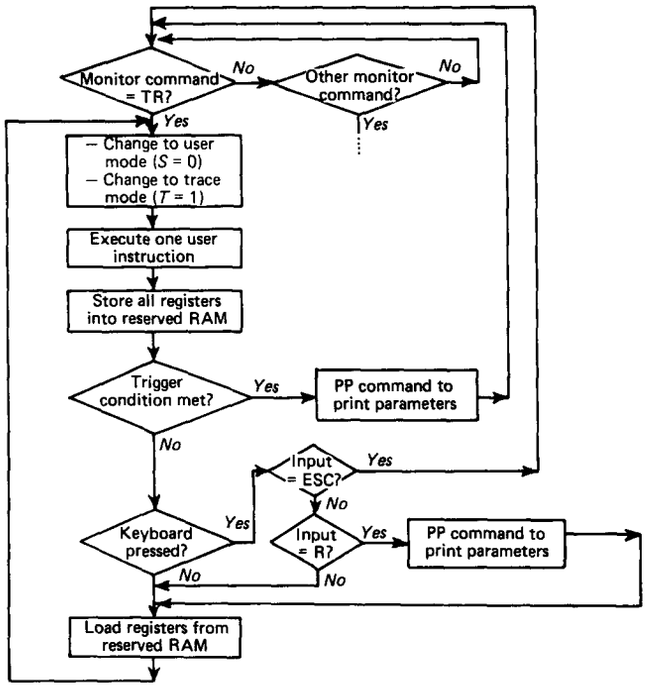


Figure 7. Flowchart of the conditional trace process

- During the tracing time the user is unaware of what is going on, while the trigger conditions may be unrealizable. Consequently, the software should include some sort of reveal function to allow the user to monitor the current processor status, registers and/or memory defined previously in the initialization commands. This reveal command is 'control — R' during the trace process (as shown in Figure 7).
- A break character (ESC) can be used at any time during program tracing to terminate and re-enter the monitor at the command level (as shown in Figure 7).

CONCLUSIONS

The software package described here has been written for the 68000 microprocessor and installed in a monitor program for the 68000 system. The package provides the user with two powerful debugging facilities: single stepping and tracing. The package is only intended for use as a debugging aid on nonrealtime software applications. This is mainly because of the overheads of the software executed on every user instruction traced.

The main ideas of the software, and how it was implemented, have been explained. A complete listing of the package is available from the authors.

ACKNOWLEDGEMENTS

S M Said would like to acknowledge the Egyptian Government for its financial support, and to thank H Twyman for his invaluable help and cooperation.

REFERENCES

1 **Williams, M H** 'A useful trace facility for microcomputers' *Microprocessors Microsyst.* Vol 5 No 3 (1981) pp 99–102
2 *MEK6800D2 evaluation kit II manual* Motorola, USA (1977)
3 Mostek Matrix 68K system with BOOT68K monitor (1983)
4 *MC68000 user manual* Motorola, USA (1982)