Title	An interactive debugging system composed of a minicomputer and a microprocessor
Sub Title	
Author	Okada, Kenichi(Matsuo, Taiki) 松尾, 泰樹(Kitagawa, Misao) 北川, 節
Publisher	慶応義塾大学工学部
Publication year	1980
Jtitle	Keio engineering reports Vol.33, No.10 (1980. 12) ,p.131- 150
JaLC DOI	
Abstract	This paper describes a debugging system which has been developed on such a combined system of a minicomputer and a microprocessor that is microprogrammable for the user, and aims at an effective debugging of the errors that will be detected during the execution of the program written in a low-level language. The multiprocessor organization, adoption of firmware monitor and special hardwares yield such advantageous features as bilateral tracing, procedure extraction, eight kinds of event monitors, etc., which come to be effective for debugging. Debugging is performed in art interactive mode so that the program can be tested and modified easily by various kinds of editing and debugging commands. This system has four modes, that is, assemble, edit, debug, and test, which allow it to perform the debug processing consistently.
Notes	
Genre	Departmental Bulletin Paper
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO50001004-00330010-0131

KEIO ENGINEERING REPORTS
VOL. 33, NO. 10, pp. 131-150, 1980

AN INTERACTIVE DEBUGGING SYSTEM COMPOSED OF A MINICOMPUTER AND A MICROPROCESSOR

KEN'ICHI OKADA, TAIKI MATSUO and MISAO KITAGAWA

Dept. of Instrumentation Engineering, Keio University, Yokohama 223, Japan

(Received July 15, 1980)

ABSTRACT

This paper describes a debugging system which has been developed on such a combined system of a minicomputer and a microprocessor that is microprogrammable for the user, and aims at an effective debugging of the errors that will be detected during the execution of the program written in a low-level language. The multiprocessor organization, adoption of firmware monitor and special hardwares yield such advantageous features as bilateral tracing, procedure extraction, eight kinds of event monitors, etc., which come to be effective for debugging. Debugging is performed in an interactive mode so that the program can be tested and modified easily by various kinds of editing and debugging commands. This system has four modes, that is, assemble, edit, debug, and test, which allow it to perform the debug processing consistently.

1. Introduction

In contrast to the drastic reduction of the recent hardware prices, software cost in the computer system tends to rather increase its rate, since the software productivity will not largely be improved. [1] One of the factors that suppress the improvement of software productivity lies in the difficulty of program debugging; [9] especially, debugging the errors detected in the execution of programs is an urgent problem, and thus many researches for the debuggers have been studied up to the present. [2] [3] [4] [5] [6] [8].

It can be considered that the requirements for such a debugger include (1) an interactive processing, (2) time-sharing processing, (3) high-performance efficiency, (4) acquisition of high-level information, (5) feasible usage, and so forth. At the time of developing the currently proposed debugging system that can satisfy the above requirements, the following principles are established:

(1) Its object is laid in the assembly language, and is to debug the errors detected in the execution of user program.
(2) It is realized on the multiprocessors to distribute their functions and also to suppress the occurrence of secondary errors which might be made through debugging.
(3) It should employ a firmware monitor.
(4) The debugging is proceeded in an interactive mode, and thus the time-sharing system is not taken into account because the object is limited to a minicomputer only.

Following the above statement, a debugging system which is realized on the combined system composed of a HITAC-10 minicomputer and a mCOM-16 microprocessor is described, which is shown in Figure 1, where the console for debugging use and various I/O units are also illustrated. [7]

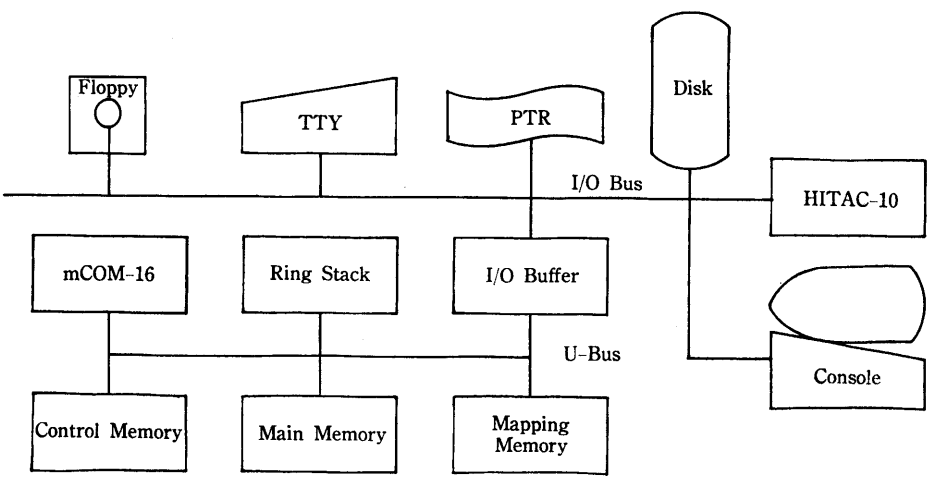

Figure 1. Hardware configuration.

2. Debugger

The debugger's software is composed of the processing programs of both the minicomputer side (nearly 4500 steps of the HITAC-10 assembly language) and the microprocessor side (about 2500 steps of the mCOM-16 microprogram), and collects and displays the debugging information obtained in the execution of the program in accordance with the commands delivered from the user. The user program is stored in the main memory of the mCOM-16 and carried out through the firmware monitor (three kinds of firmware monitors are registered in the disk and transferred to the mCOM-16 control memory). Since the mCOM-16 is controlled and managed by the HITAC-10, the user needs not be conscious of the existence of the mCOM-16. On the other hand, the processing program in the HITAC-10 is

divided into three; namely, (1) the command processing part which processes and analyzes the user's commands, (2) the debugging information part which arranges and displays the information obtained from the firmware monitor, and (3) the I/O service part which is prepared for the mCOM-16 providing no I/O units.

2.1 Firmware Monitor

For the purpose of collecting effective and high-level information without introducing any disturbances at the debug processing, the following are set as the monitor:

(1) Execution of instructions by the specified number of steps.
(2) Execution of the specified instructions by the specified number of times.
(3) Execution of the instruction at the specified location by the specified number of times.
(4) Execution of instructions at the specified area of the memory.
(5) Read out from the specified location or area.
(6) Write into the specified location or area.
(7) Comparison of the contents of the memory with the constants.
(8) Comparison of the contents between two different memory locations.

The multiple monitors can be designated simultaneously for those events, and even when only one of them is specified during the execution of user program, the execution is halted to display it to the user. Thus the user can recognize the executing state at the time when the individual conditions have been satisfied.

Furthermore, the following are prepared as the back-up functions in order to proceed the debug processing smoothly:

(1) Restoring function of the initial state:
Since the user program stored in the disk is transferred to the main memory of the mCOM-16, and at the same time the initialization of the firmware monitor is carried out, the reexecution of the user program can be made from the initial state arbitrarily.

(2) Read and write functions for the registers and memory:
The contents of all the registers and memory of the emulated machine can be read or written.

(3) Procedure extraction function:
A series of processes which affect a particular location are taken out as a unit of procedure. Namely, this is such a function that since a sequence of instructions composed of a program has a pattern that it may start with a transfer instruction and then end at another transfer instruction after processed by some arithmetic instruction(s), such a sequence is picked up from the program to display as a unit of procedure. It will be a powerful tool for the user in a debugging operation to recognize a series of instructions that affect the location, the value of which is found to be inadequate.

(4) Backtrack function:
Backtracking, which proceeds the execution of user program in the reverse direction, can be realized by stacking and monitoring the data information, which comprises the contents of memory and registers that

will be destroyed during the forward execution of the program and the address information concerning the program flow at the Jump, Call, Return instructions and the like in the ring stack. The number of steps that can be backtracked amounts to approximately 1000 steps at present, which will be determined by the ring stack capacity.

(5) Trace function:

This function provides a means of displaying the contents of the respective registers and effective addresses everytime when the specified operations, specified operands, or the instructions in the specified area are carried out, and consequently can catch the consecutive state transitions of user program. During the trace, both collection of the trace information due to the firmware monitor on the mCOM-16 and its display to the user through the HITAC-10 are performed in parallel.

(6) I/O change function:

This permits the change of the I/O units used in the user's program without altering its construction. Presently the TTY and PTR are provided.

(7) Disk dump function:

This allows the user program to be saved in the disk at any time and to return to that point, and comes to be effective to realize a check point and restart function.

These functions as well as the designation of events to be monitored are carried out using the commands, and the HITAC-10 generates the Reference Table which will be mentioned later in accordance with the given commands. The firmware monitor refers to the Reference Table and proceeds the execution of user program, collecting the useful information only. Events are checked at the following three processing parts:

(1) Fetch part, where the execution address, the number of steps, and the kind of instructions are checked.
(2) Address computation part, where the memory access is checked.
(3) Execution part, where the executed results are checked.

The result of individual checking, if an event specified by the user comes into existence, will be informed to the HITAC-10, which then displays the generated event, the contents of individual registers, and the contents of consecutively seven locations of the program placing the approved location in the center of them.

The following three kinds of runs are registered in the firmware monitor: (1) Emulation run, which proceeds the execution of user program while monitoring the event specified by the user's commands; (2) preparation run, where the stacking operation for backtrack use is added to the emulation run; and (3) back run, which executes the program in the reverse direction while monitoring the event. Those are stored in the disk and transferred to the control memory of the mCOM-16 through user commands, and also the mapping memory is rewritten. Figures 2 and 3 depict these three kinds of monitor flows.

2.2 Commands and Reference Table

For the purpose of easy input and of getting abundant, useful information, the commands listed in Table 1 are prepared. As the operand of command, one

can freely use the symbolic names in the user program, the direct address designation by decimal or hexadecimal representation, and the mnemonic names of registers. Multiple commands input from the console for debugging use are rearranged and compiled to generate the Reference Table.

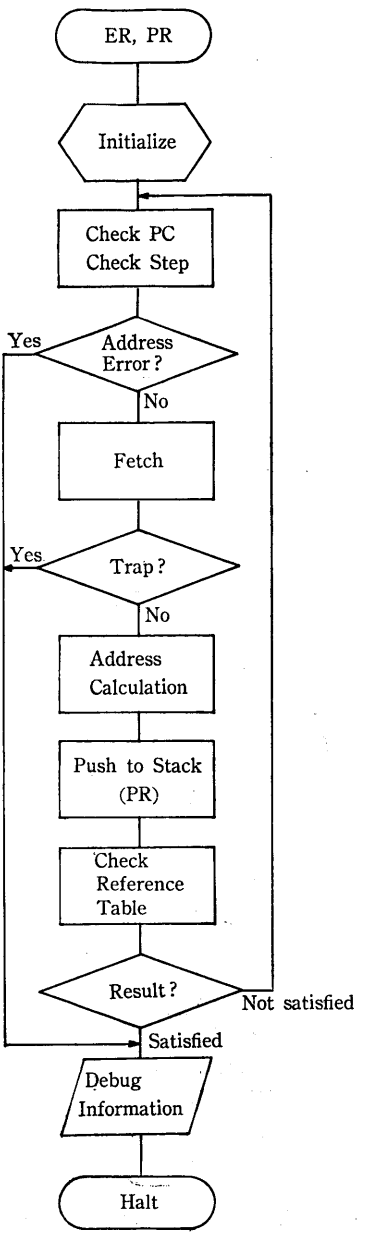

Figure 2. Flow of the firmware monitor (ER, PR).

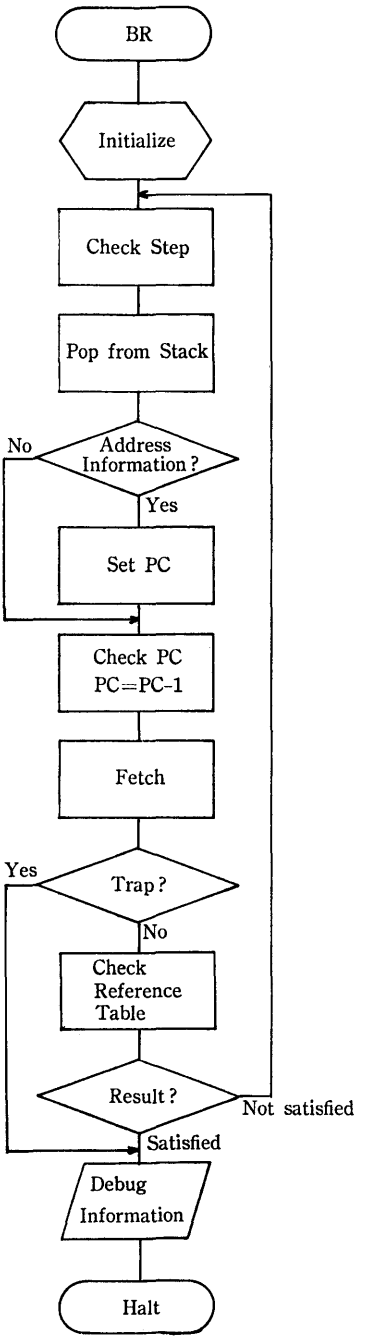

Figure 3. Flow of the firmware monitor (BR).

Table 1. Command of the debugger.

Format	Function
I	Initialize
DD v	Dump to disk
DM v	Dump to memory
SR n v —	Set register
SM a v —	Set memory
GR n v —	Get register
GM a v —	Get memory
BP a v —	Break point
PC r —	Program counter
ST v	Step
OC c v —	Operation code
OR a v / r —	Operand refer
OM a v —	Operand modify
CC a o v —	Compare with constant
CV a o a′ —	Compare with variable
E a	Extract
IO i	I/O change
CIO	Cancel I/O change
RA a / r —	Reverse assemble
T a / r / c —	Trace
UT a / r / c —	Untrace
ER	Emulation run
PR	Preparation run
BR	Back run
F	Finish
(ESC)	Go to command waiting
(Control-P)	Cancel current command

r: range
  address—address
a: address
  decimal number, hexadecimal number, or label
n: register name
  PC, AC, EC, CAR, or IR
v: value
  decimal number, or hexadecimal number
o: operator
  EQ, GT, GE, LT, LE, or NE
c: operation code
i: I/O device name
  PTR or TTY
—: repeatable

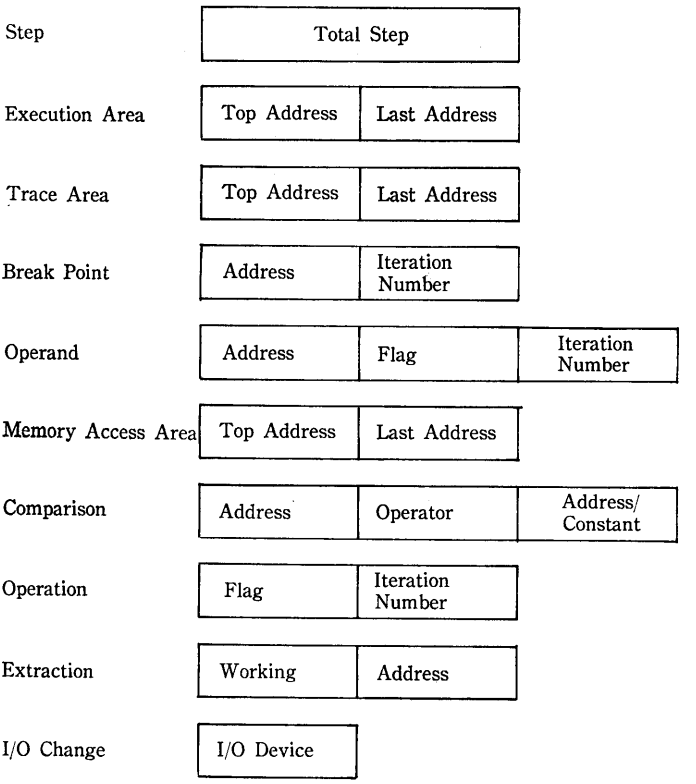

Figure 4. Format of the Reference Table.

The Reference Table determines the operation of the firmware monitor and has 10 kinds of formats which keep the following information, as given in Figure 4:

(1) Number of steps to be executed.
(2) Location or area where the execution is prohibited.
(3) Location or area to be traced.
(4) Number of times of executing a location till the time of activating the breakpoint and/or break mechanism.
(5) Specified number of accesses to a location and the necessity of tracing.
(6) Memory area where the access is to be monitored.
(7) 'Compare' operator and two locations to be compared or a location and a constant.
(8) Specified number of times of executing macroinstructions and the necessity of tracing.
(9) A location to be noticed by the procedure extraction function.
(10) Names of two I/O devices to be changed each other.

In the above stated Reference, (1), (9), and (10) include only one item each; (8) will have the same number of items as that of the macroinstructions; and (2)

through (7) will keep the necessary number of entries for themselves. This Reference Table will be transferred to the special area of the main memory of the mCOM-16 through the execution (run) commands of the user program.

2.3 Processing of the Minicomputer Side

While the firmware monitor on the mCOM-16 executes the user's program, the processing that the minicomputer performs in the debugger includes the following:

(1) Input of the user commands from the console.
(2) Analysis of the commands, generation and updating of the Reference Table.
(3) Read out from and write into the registers and the memory of the mCOM-16.
(4) Transfer of the microprogram to the control memory of the mCOM-16 and its initiation.
(5) I/O services for the mCOM-16.
(6) Rearrangement and display of the debugging information resulted from the firmware monitoring.
(7) Mode change of the system.

Figures 5 and 6 show the flow of processing by the minicomputer of the

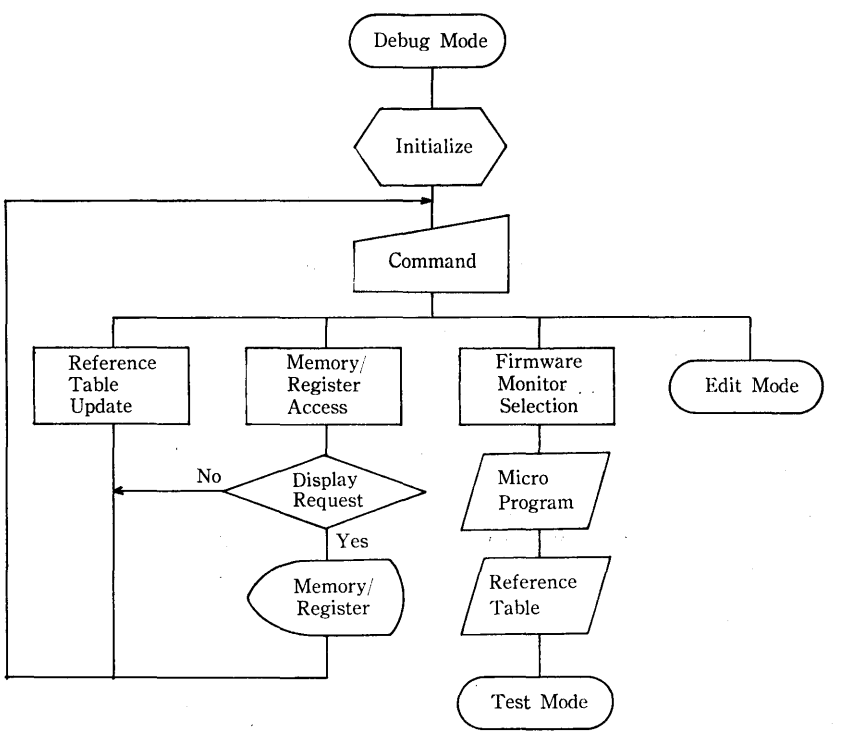

Figure 5. Flow of the debugger in debug mode.

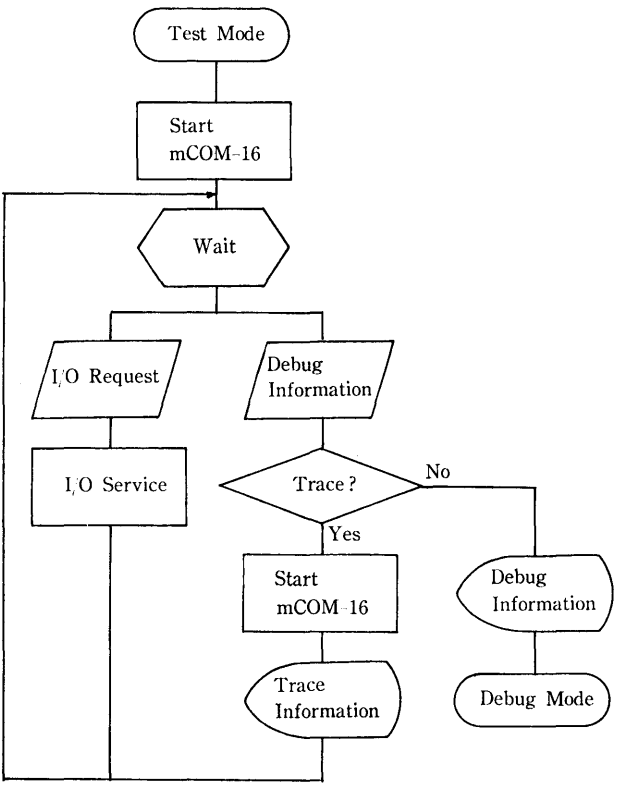

Figure 6. Flow of the debugger in test mode.

debugger.

This system has the four operation modes as follows:

(1) Assemble mode that converts the user's program to the object one.
(2) Edit mode that performs the program compilation and the file processing.
(3) Debug mode that arranges the condition for the debug processing and displays the debug information.
(4) Test mode that the firmware monitor performs the execution of user program.

Table 2 lists up the contents of the respective memory area in those four modes. On the HITAC-10, the assembler for debugging use runs in assemble mode, the text editor runs in edit mode, and the debugger runs in either debug or test mode; the distinction between them depends upon whether or not the firmware monitor in the control memory of the mCOM-16 is operating.

The role of the assembler for debugging use is to convert the user program to the object one and it is provided as the debugging version after having modified the macro-assembler delivered from the manufacturer. The principal modification is in that the address designation by symbols while debugging the program can be performed by way of saving the symbol table in a safety area at the end of assembling.

Table 2. Contents of the specified memories for four system modes.

	Edit Mode	Assemble Mode	Debug Mode	Test Mode
HITAC-10 Main Memory	Editor	Assembler	Debugger	Debugger
Common Area	—	—	Reference Table	Reference Table
mCOM-16 Main Memory	—	—	Object	Object
mCOM-16 Control Memory	—	—	—	Firmware Monitor

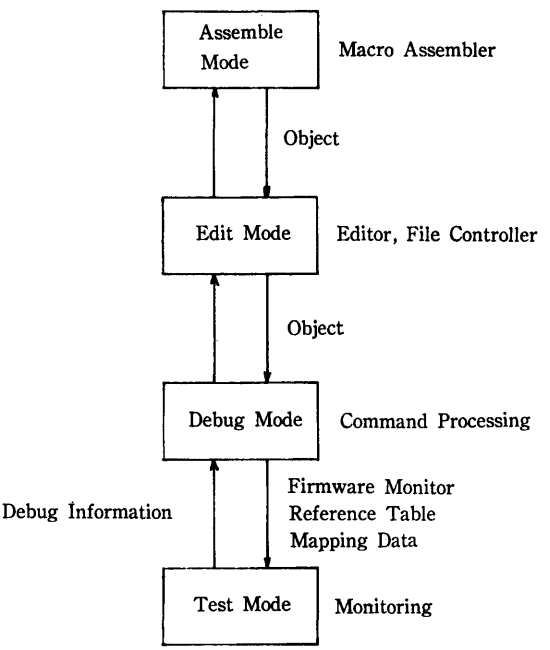

Figure 7. Mode transition.

The text editor has the function of various kinds of compilation and file processing, and is capable of linking the assembler with the debugger.

Figure 7 gives the operation and transition of these four modes, where the transition from assemble mode to edit is initiated automatically after terminating the assembling; the transition from test mode to debug is due to the generation of a specified event, and the other transitions are generated by the user's commands. In debug and test modes the following data transfer is carried out between the HITAC-10 and the mCOM-16.

(1) From HITAC-10 to mCOM-16,

(a) object codes of the user's program (in debug mode),

(b) firmware monitor (in debug mode),
(c) data to the mapping memory (in debug mode),
(d) Reference Table (in debug mode), and
(e) I/O service information (in test mode).

(2) From mCOM-16 to HITAC-10,
(a) debug information (in test mode), and
(b) I/O service request (in test mode).

These data transfer will be performed through the two buffer registers connecting the I/O Bus of the HITAC-10 with the U-Bus of the mCOM-16.

3. Examples of Debug Processing

This chapter describes some examples concerning the actual debug processing by making use of the results obtained from the execution of a sample program (see Appendix) through the debugger. The program is to convert a 4-digit hexadecimal (4 bytes) number of the character format to the decimal, using the TTY as an I/O unit. Each input hexadecimal digit of 1 byte is converted to a 4-bit hexadecimal digit after echobacking and stored in a word of 16 bits packed to the lefthand side, where the most significant bit is considered to be the sign. The 4-bit hexadecimal number thus obtained is output after converting it to the character type decimal number. The following lists are the output results obtained from the execution of the sample program with several kinds of commands picked out of those in this debugger. ($$ indicates the mode change, ** gives the system message in debug mode, mark # shows the address where the program is stopped after the HLT instruction has been carried out or some condition has been satisfied.)

(1) From edit to debug mode by Z command. First, initialization is carried out. (See List 1.)

List 1.

```
$$  TTY - EDITOR  $$
>Z
$$ HITAC-10 DEBUGGER $$

#I
```

(2) The output results obtained from the normal execution without any input of the commands for event monitor use. (See List 2.)

The Program Counter is set to the starting location and the user program is executed by the ER command. Since the input routines from the TTY (X'2005' to X'200A') is in the user program, the operation will be at the state of queue. When 0800 is input at this point, =␣02048 (␣ indicates a space) is output by the user program, and the debugger informs the user that the HLT instruction has been carried out. The indicated information includes the following: the location and the instruction in it at the time its execution ended, the contents of Accumulator (AC), Extended Accumulator (EC), Carry Register (CAR), Effective Address (EA) and its contents ((EA)) as well as the location where the instruction just finished

List 2.

```
#SR PC X"2000"

#ER
0800= 02048
** I'VE JUST EXECUTED "HLT" INSTRUCTION. **

LOC  IR   AC   EC   CAR EA   (EA)
2033 6F00 000A 0000 0

      2030  5266        KCT        ADR2 *+054 #2066
      2031  5259        KCT             *+040 #2059
      2032  422E        B               *-004 #202E
    * 2033  6F00        HLT
      2034  0A6B        L          X20  *+055 #206B
      2035  3A5E        ST         BUFF *+041 #205E
      2036  4200        B          BGN  *-054 #2000
** CONTINUE ? **
```

List 3.

```
#CC M7+1 EQ 0
** THE SPECIFIED LOCATION HAS THE APPOINTED VALUE NOW.**

#BP BGN+1

#ER
** I'VE JUST DONE THE SPECIFIED LOCATION.**

LOC  IR   AC   EC   CAR EA   (EA)
2001 5259 205E 0000 0   2059 FFFB => FFFA

      1FFE  F1C4        STE   Z            472 #01C4
      1FFF  9156        SRA                342 #0176
      2000  4A37   BGN  BAL         INIT *+055 #2037
    * 2001  5259        KCT              *+088 #2059
      2002  5259        KCT              *+087 #2059
      2003  5259        KCT              *+086 #2059
      2004  5266        KCT         ADR2 *+098 #2066

#CC M7+1 EQ 0

#ER
FFFB
** THE SPECIFIED LOCATION HAS THE APPOINTED VALUE NOW.**

LOC  IR   AC   EC   CAR EA   (EA)
201D 5259 FFFB 0000 0   2059 FFFF => 0000

      201A  8804        SLL                004 #0004
      201B  325B        O           HEXA *+064 #205B
      201C  3A5C        ST          PACK *+064 #205C
    * 201D  5259        KCT              *+060 #2059
      201E  4205        B           READ *-025 #2005
      201F  6864        KPA
      2020  4A51        BAL         NEGT *+049 #2051

#GR AC
 CONTENT OF AC  ;#FFFB

#GM PACK
 CONTENT OF LOCATION #205C IS # FFFB
```

List 4.

```
#BP X"2021"

#T NEGT,NEGT+6

#ER

LOC  IR   AC   EC   CAR EA   (EA)
2052 2A50 0004 0000 0   2050 FFFF

      2052  2A50        X              *-002 #2050

LOC   IR   AC   EC   CAR EA   (EA)
2053 3A5E 0004 0000 0   205C FFFB => 0004

      2053  3A5C        ST         PACK *+009 #205C

LOC  IR   AC   EC   CAR EA   (EA)
2054 525C 0004 0000 0   205C 0004 => 0005

      2054  525C        KCT        PACK *+008 #205C

LOC  IR   AC   EC   CAR EA   (EA)
2055 0A6A 002D 0000 0   206A 002D

      2055  0A6A        L          X2D  *+021 #206A

LOC  IR   AC   EC   CAR EA   (EA)
2056 3A5E 002D 0000 0   205E 0020 => 002D

      2056  3A5E        ST         BUFF *+008 #205E

LOC  IR   AC   EC   CAR EA   (EA)
2057 4651 002D 0000 0   2021 0A5A

      2057  4651        B     I    NEGT *-006 #2051

** I'VE JUST DONE THE SPECIFIED LOCATION.**

LOC  IR   AC   EC   CAR EA   (EA)
2021 0A5A 003D 0000 0   205A 003D

      201E  4205        B          READ *-025 #2005
      201F  6864        KPA
      2020  4A51        BAL        NEGT *+049 #2051
    * 2021  0A5A        L          EQ   *+057 #205A
      2022  4A48        BAL        WRTE *+038 #2048
      2023  4A3D        BAL        CVHD *+026 #203D
      2024  2710        N     I         *+236 #2110
```

its execution and the inversely assembled results of seven locations (2030—2036). At the end of the HLT instruction, the 'CONTINUE' message is issued and will start the execution of program from the next location when 'C' is input from the TTY. In case where the characters except 'C' are input, the debugger will be at the state of accepting the commands of debug mode.

(3) List 3 is an example to confirm that the 4-byte input data are packed and stored in the PACK location.

Since location M7+1 is a counter of the input characters, the command to halt is input when the contents of location M7+1 has come to be zero

List 5.

```
#BP X"202D"

#ER
=
** I'VE JUST DONE THE SPECIFIED LOCATION.**

LOC  IR   AC   EC   CAR EA   (EA)
202D 4A37 0035 0000 0   2037 2001 => 202E

      202A  000A
      202B  4A3D        BAL         CVHD *+018 #203D
      202C  0001
    * 202D  4A37        BAL         INIT *+010 #2037
      202E  0E66        L     I     ADR2 *+056 #2066
      202F  4A48        BAL         WRTE *+025 #2048
      2030  5266        KCT         ADR2 *+054 #2066

#GM BUFF,BUFF+5
 CONTENT OF LOCATION #205E IS # 002D
 CONTENT OF LOCATION #205F IS # 0030
 CONTENT OF LOCATION #2060 IS # 0030
 CONTENT OF LOCATION #2061 IS # 0030
 CONTENT OF LOCATION #2062 IS # 0030
 CONTENT OF LOCATION #2063 IS # 0035
```

(input of 4 characters finished). The message that the conditions have already been satisfied is issued (which will be checked before executing the CC and CV commands), so that the program is executed to the location where the initialization of it completed. Here again the command to halt is input when the contents of location M7+1 come to be zero and then the program is executed. As the the result, such a message is output that the conditions are satisfied after the input of hexadecimal date 'FFFB' of 4-byte form. It can be confirmed that the input data in the character format are stored in the form of hexadecimal numbers after reading the contents of the AC and PACK locations.

(4) List 4 gives an example to confirm the state transition that the minus datum FFFB stored in the PACK location is converted to a form of 2's complement.

While the program is executed to location X'2021', the command to trace locations NEGT to NEGT+6 (conversion subroutine to the 2's complement) is input. It can be confirmed below that the contents 'FFFB' of the PACK location comes to be 0005 and the minus code X'2D' is embedded in the BUFF location.

(4) List 5 is an example to confirm that the converted data to the decimal numbers are stored in locations BUFF to BUFF+5 in the character format.

Such a command is input that executes the program till the time when embedding the character type decimal data to the BUFF is completed. It can be confirmed that the contents of locations BUFF to BUFF+5 are read and '−00005' is embedded.

(6) Examples of the re-initialization setting, modify, step, and extract.

By making use of the I command, it is possible to execute the program again from its initial state at any arbitrary point of it. The program can

be executed by the input of the command to halt and by proceeding the stacking for the back trace use after the WORK location has been modified. At the state of input queue (due to the user program), when '8' is input, the message is output, whicn indicates that the read-in data X'38' has been stored in the WORK location. (See List 6.)

The following shows the results after executing 4 steps of the program

List 6.1.

```
#I

#SR PC X"2000"

#OM WORK

#PR
8
** I'VE MODIFIED THE SPECIFIED LOCATION.**

LOC  IR   AC   EC   CAR EA   (EA)
200E 3A67 0038 00B8 1   2067 0000 => 0038

      200B  4A48        BAL        WRTE *+061 #2048
      200C  8809        SLL             009 #0009
      200D  8009        SRL             009 #0009
    * 200E  3A67        ST         WORK *+089 #2067
      200F  1A69        S          X40  *+090 #2069
      2010  6864        KPA
      2011  4215        B                *+004 #2015
```

List 6.2.

```
#ST 4

#PR
** I'VE DONE THE NUMBER OF STEPS YOU ORDERED. **

LOC  IR   AC   EC   CAR EA   (EA)
2015 0A67 0038 00B8 1   2067 0038

      2012  0A67        L          WORK *+085 #2067
      2013  1268        A               *+085 #2068
      2014  4216        B               *+002 #2016
    * 2015  0A67        L          WORK *+082 #2067
      2016  880C        SLL             012 #000C
      2017  800C        SRL             012 #000C
      2018  3A5B        ST         HEXA *+067 #205B
```

List 6.3.

```
#E WORK

#BR

LOC  IR   AC   EC   CAR EA   (EA)
2009 7074 205E 00B8 0

      2009  7074        RTI   C
      200C  8809        SLL             009 #0009
      200D  8009        SRL             009 #0009
      200E  3A67        ST         WORK *+089 #2067
```

while stacking.

The following gives the results obtained after extracting a series of instructions that affect the WORK location while restoring the program inversely from the X'2015' location.

The extract instruction halts at the time when restoring the transfer instruction that affects the specified location. Accordingly, this example shows that the contents of the AC, EC, and CAR were X'205E', X'00B8', and 0, respectively, before executing the RTI C instruction.

To edit mode by the F command that follows,

List 6.4.

```
#F
$$  TTY - EDITOR  $$
```

4. Conclusion

Since the program used in the previous examples of debug processing is so much simple and required some hand manipulation of the user that the reduction of execution efficiency by the firmware monitor has not been found at all. Therefore, when a program that has no I/O instructions to compute the factorial of 1000 is carried out under the following three conditions, the execution time is as follows:

(1) Execution by the HITAC-10—42.0 msec.
(2) Execution by the emulator—420.1 msec.
(3) Execution by the firmware monitor—2032.0 msec.

These figures indicate that the result by the firmware monitor introduced approximately 50 times of speed reduction composed with that by the hardware, but the principal cause of it is due to the low performance of the microprocessor mCOM-16 and also the lack of the hardware such as multiplication, division, shifter, or the like. The evaluation of the overhead by means of supplementing the debugging function should be composed with the usual emulator, where the speed reduction amounts to about 5 times. In fact when this system was operated, the processing time for debugging information at the user side has come to be extremely large, and thus the decrease in execution speed of the program in the minicomputer does not give much trouble to the user. The most essential factor is that the profitable debugging information is obtained and thus the modification and testing of the program can easily be carried out. This system provides a simple mode change, many edit functions as well as monitor ones, which have all used to accomplish the purpose of debugging from the standpoint of improving the overall performance of the system.

REFERENCES

[1] Boehm, B. W., (1976): Software engineering, IEEE Trans. on Computers, Vol. C-25, No. 12, pp. 1226-1240.

[2] FLANAGAM, H., (1973): Program debugging system, IBM Technical Disclosure Bulletin, Vol. 16, No. 7, pp. 2322-2329.

[3] GOLDBERG, J., COOPERBAND, A., and GALLENSON, L., (1978): PRIM System—A Framework for emulation-based debugging tools, National Computer Conference, pp. 373-377.

[4] GRISHMAN, R., (1970): The debugging system AIDS, SJCC, pp. 59-64.

[5] OKADA, K, and KITAGAWA, M., (1978): BRAKEMAN—A debugging hardware system, KEIO Engineering Reports, Vol. 31, No. 13, pp. 139-149.

[6] OKADA, K., YOKOHAMA, T., and KITAGAWA, M., (1979): A dynamic debugging system by multiple processors (in Japanese), Technical Reports of Information Processing, IEE of Japan, IP-79-45, pp. 47-56.

[7] OKADA, K. and KITAGAWA, M., (1980): A general-purpose experimental computer system characterized by its architecture changeability, KEIO Engineering Reports, Vol. 33, No. 5.

[8] SAKAMURA, K., KITAFUSA, H., TAKEYARI, Y., and AISO, H., (1977): A debugging machine (in Japanese), Trans. IECE of Japan, Vol. J60-D, No. 9, pp. 671-678.

[9] SCHWARZ, J.T., (1970): An overview of bugs, Debugging Techniques in Large Systems, PRENTICEHALL, INC., Englewood Cliffs, New Jersey.

Appendix

```
      0000I
0800  0800G
                ////////////////////////////////////
                /   CONVERT HEXADECIMAL TO DECIMAL  /
                ////////////////////////////////////
      0000I
2000  2000G          ORG     X"2000"
2000  4A37@    BGN   BAL     INIT
2001  5259@          KCT     M7+1
2002  5259@          KCT     M7+1
2003  5259@          KCT     M7+1
2004  5266@          KCT     ADR2
2005  6D00@    READ  RIM             / READ HEXADECIMAL NUMBER
2006  7062@          STI
2007  7061@          KTI
2008  4207@          B       *-1
2009  7074@          RTI,C
200A  6C00@          SIM
200B  4A48@          BAL     WRTE    / ECHO BACK
200C  8809@          SLL     9       / REMOVE PARITY
200D  8009@          SRL     9
200E  3A67@          ST      WORK
200F  1A69@          S       X40     / A B C D E F ?
2010  6864@          KPA
2011  4215@          B       *+4
2012  0A67@          L       WORK
2013  1268@          A       WORK+1
2014  4216@          B       *+2
2015  0A67@          L       WORK
2016  880C@          SLL     12
2017  800C@          SRL     12      / REMOVE ZONE CODE
2018  3A5B@          ST      HEXA
2019  0A5C@          L       PACK
201A  8804@          SLL     4
201B  325B@          O       HEXA
201C  3A5C@          ST      PACK    / STORE INTO ONE WORD
201D  5259@          KCT     M7+1
201E  4205@          B       READ
201F  6864@          KPA
2020  4A51@          BAL     NEGT    / WHEN NEGATIVE
2021  0A5A@          L       EQ
2022  4A48@          BAL     WRTE    / PRINT CHAR. =
2023  4A3D@          BAL     CVHD
2024  2710@          DC      10000
2025  4A3D@          BAL     CVHD
2026  03E8@          DC      1000
2027  4A3D@          BAL     CVHD
2028  0064@          DC      100
2029  4A3D@          BAL     CVHD
202A  000A@          DC      10
202B  4A3D@          BAL     CVHD
202C  0001@          DC      1
202D  4A37@          BAL     INIT
202E  0E66@          L ,I    ADR2
202F  4A48@          BAL     WRTE    / PRINT DECIMAL NUMBER
2030  5266@          KCT     ADR2
2031  5259@          KCT     M7+1
2032  422E@          B       *-4
2033  6F00@          HLT
2034  0A6B@          L       X20
2035  3A5E@          ST      BUFF
2036  4200@          B       BGN
```

```
2037  0000@   INIT  DC              / INITIALIZE
2038  0A58@         L       M7
2039  3A59@         ST      M7+1
203A  0A65@         L       ADR1
203B  3A66@         ST      ADR2
203C  4637@         B ,I    *-5
                  /
203D  0000@   CVHD  DC              / CONVERT
203E  8800@         SLL
203F  C25C@         LE      PACK
2040  EE3D@         D ,I    *-3
2041  3A5C@         ST      PACK    / REMAINDER --> PACK
2042  A800@         SLDL
2043  125D@         A       X30
2044  3E66@         ST ,I   ADR2    / QUOTIENT --> (ADR2)
2045  5266@         KCT     ADR2
2046  523D@         KCT     CVHD
2047  463D@         B ,I    CVHD
                  /
2048  0000@   WRTE  DC              / PRINT ONE CHAR.
2049  6D00@         RIM
204A  7086@         WTO
204B  7081@         KTO
204C  424B@         B       *-1
204D  7082@         CTO
204E  6C00@         SIM
204F  4648@         B ,I    WRTE
                  /
2050  FFFF@         DC      X"FFFF"
2051  0000@   NEGT  DC
2052  2A50@         X       NEGT-1
2053  3A5C@         ST      PACK
2054  525C@         KCT     PACK    / TWO'S COMPLEMENT
2055  0A6A@         L       X2D
2056  3A5E@         ST      BUFF
2057  4651@         B ,I    NEGT
                  /
2058  FFF9@   M7    DC      -7
2059  0001K         DS      1
205A  003D@   EQ    DC      C"="
205B  0001K   HEXA  DS      1
205C  0000@   PACK  DC
205D  0030@   X30   DC      X"30"
205E  0020@   BUFF  DC      X"20"
205F  0005K         DS      5
2064  000A@         DC      X"0A"
2065  205E@   ADR1  DC      BUFF
2066  0001K   ADR2  DS      1
2067  0001K   WORK  DS      1
2068  0009@         DC      9
2069  0040@   X40   DC      X"40"
206A  002D@   X2D   DC      X"2D"   / MINUS -
206B  0020@   X20   DC      X"20"   / SPACE
      F352I
00B9  00B9G
00B9  3FC0@
00BA  3FC0@
      7F80I
009F  009FG
009F  0000@
      0000I
      2000H         END     BGN
```