Title	Brakeman' : a debugging handware system
Sub Title	
Author	岡田, 謙一(Okada, Kenichi) 北川, 節(Kitagawa, Misao)
Publisher	慶応義塾大学工学部
Publication year	1978
Jtitle	Keio engineering reports Vol.31, No.13 (1978. 10) ,p.139- 149
JaLC DOI	
Abstract	This paper presents a method of debugging the computer program with some hardware implementation attached purposely and exclusively to the computer in order to process the debugging faster and more efficient than with usual debugging systems, especially in a small-scale computer system.
Notes	
Genre	Departmental Bulletin Paper
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO50001004-00310013-0139

KEIO ENGINEERING REPORTS
VOL. 31, NO. 13, pp. 139-149, 1978

'BRAKEMAN'—A DEBUGGING HARDWARE SYSTEM

KEN'ICHI OKADA AND MISAO KITAGAWA

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

(Received December, 12, 1977)

ABSTRACT

This paper presents a method of debugging the computer program with some hardware implementation attached purposely and exclusively to the computer in order to process the debugging faster and more efficient than with usual debugging systems, especially in a small-scale computer system.

1. Introduction

When programming with a computer, it is very difficult, if not impossible, for anybody to make a program without any errors in his only one trial. It is, of course, largely concerned with the scale and structure of the program as well as the language used within it, so that plenty of time and labors are usually required to find out the errors. It is in fact said that when a program is to be developed, the cost for debugging is nearly 40 percent of the whole cost that will be spent for completing it.

When a user makes a program and then lets the computer execute it, he recognizes the bugs (that is, errors in the algorithm, mistaken punches, and the similar erroneous things) in the program at the time of either input, output, or execution phase. Those bugs can be grouped into the following three types; (1) bugs to be found out at the input stage, (2) bugs at the output, and (3) bugs during the execution stage.

Type (1) is a sort of errors concerning the grammar of the used language, so that such bugs can be precisely found out at their positions by using the programming language such as the compiler or assembler. Therefore, the errors of this type can easily be debugged in general.

Type (2) belongs to a logical error which can be found when the output results are examined and compared with the expected data. Thus the bugs of this kind can not be checked by the computer during the execution of the program. It is a very important theme in the future that the computer can have the capability of finding automatically the logical errors in a program by itself, although it is strongly related to the automatic programming.

Now the most troublesome bug is due to type (3), which often occurs when a program is made using a low level language such as an assembler[1], and the existence of such errors can be recognized by the user as well as the CPU through the hardware interruption such as an address error, operation code error, or operation error, for example. Debugging is usually made effective, provided that all kinds of information given from the CPU that has been stopped after an interruption is utilized, which are the contents of every register, and the cause and operation address of the interruption just occurred, for example.

When a bug is located considerably apart from the operation address of the interruption occurred, it is then very difficult to debug the program because the information to be used from the CPU might have already been destroyed at the time of the interruption[3]. The debugging system provided presently against this type of bugs contains the supporting systems of both such software as tracer and memory dump routine, and the hardware like the conditional stops[2]. A utility routine for debugging has the drawback that it will generally take a lot of time and require an extra memory area for reserving the debugging routine. The former drawback is actually an important problem for the large-scale computers, while the latter is also true to the small-scale ones. On the other hand, when a conditional stop like an address stop is used, for instance, it could not uniquely be determined which address should be appropriate to be selected absolutely. Moreover, if a bug is in a loop of a program and an error breaks out after the loop has repeatedly been executed several times, then it will become very troublesome or even impossible to continue to operate the CPU.

The Brakeman, a debugging hardware system, presented here is based on the idea that if the CPU can be stopped just before an error breaks out, it will provide a strong means for debugging[4]. The Brakeman, therefore, has to execute twice the same user's program in order to stop the CPU just before an error condition breaks out, and then the status change of the CPU is used for the purpose of debugging from the stopped time. Namely, this system intends to shut a bug with respect to the time, while the existing debugging systems of software are usually trying to shut a bug spatially. The following are the features of this system :

(1) The Brakeman needs no extra area to be used for debugging in the memory.
(2) The user's program need not be changed for debugging.
(3) The CPU can operate in such a high-speed mode as intrinsic to itself through the part of the program where any error never breaks out, and in a lower speed where some errors may break out, so that useful information for debugging can be obtained quickly.
(4) Since the Brakeman doesn't affect the normal execution of the CPU, the other debugging systems can be used cooperating with this system.

By making a good use of the Brakeman, the user can debug his program without changing it in a shorter time. The performance of the Brakeman is affected by the structure of the user's program, the type of bugs, and the condition when an error breaks out.

Section 2 presents the structure and usage of the Brakeman. Section 3 describes the implementation of the trial Brakeman connected to the minicomputer HITAC-10

and the result obtained out of some experiments. Section 4 deals with a further problem of the Brakeman.

2. The Outline of the Brakeman

If a program has a repeatability, the execution time of the program is always fixed. Accordingly, when a program has any bugs, the execution time that elapsed from the beginning to the time of finding errors is also fixed. This system takes notice of this viewpoint that it shuts a bug up in the time.

First, a user's program which possibly has any sort of bugs is executed normally, and the Brakeman measures the execution time passed from the beginning to the time of finding errors. Then the same program is executed once again, where this time the Brakeman can stop the CPU at the time just before an error breaks out. In other words, of the twice execution times, the first trial is carried out for the preliminary searching of errors, and the second is devoted to debug the program through the result obtained from the first examination so as to stop the CPU in order to watch the condition of it by means of the step-by-step method, if an error is possibly to break out. The Brakeman is a kind of timer, and stops the CPU before an error breaks out and then the useful information for debugging may not be destroyed.

Fig. 1 shows the block diagram of the Brakeman and Fig. 2 the logic circuits of it. The HITAC-10 minicomputer is used to make an experiment on the Brakeman system in order to verify the effect of debugging. The Brakeman consists of an indicator, a detector, a counter, and a controller. The indicator transmits the signals for the conditions of the CPU, that is, run, idle, waiting, I/O processing, or hardware interruptions, and the fetch pulse of the CPU through a sort of cable. The detector examines the condition of the CPU and decides to either open or close the gate of the counter. The counter measures the execution time in the first trial of the user's program by the fetch pulses of the CPU and decides when to stop the CPU in the second trial. The controller starts, stops, or resets the CPU by operating the appropriate keys on the front panel and/or sending a control signal pulse through the cable.

As the Brakeman uses a counter pulse in order to measure the execution time of the CPU, the counting pulse has to synchronize with the execution of the CPU. If not so, the Brakeman could not stop the CPU before the error breaks out, or could not appoint an exact interval of time between the stop and error points. It is very exact and convenient to use the cycle time pulse of the CPU as the counting pulse. It takes one cycle time to fetch, modify, or execute the simple instructions used, while it takes several cycle times for the execution of the shift, multiply, and divide operations. Therefore, the stop point will be fluctuated to some extent by the individual instructions just before the error point, and there may include a risk unable to stop the CPU before the error breaks out. The Brakeman takes out the fetch pulses of the CPU and counts them up, so it finds the error point exactly.

The behavior and usage of the Brakeman are explained as follows:

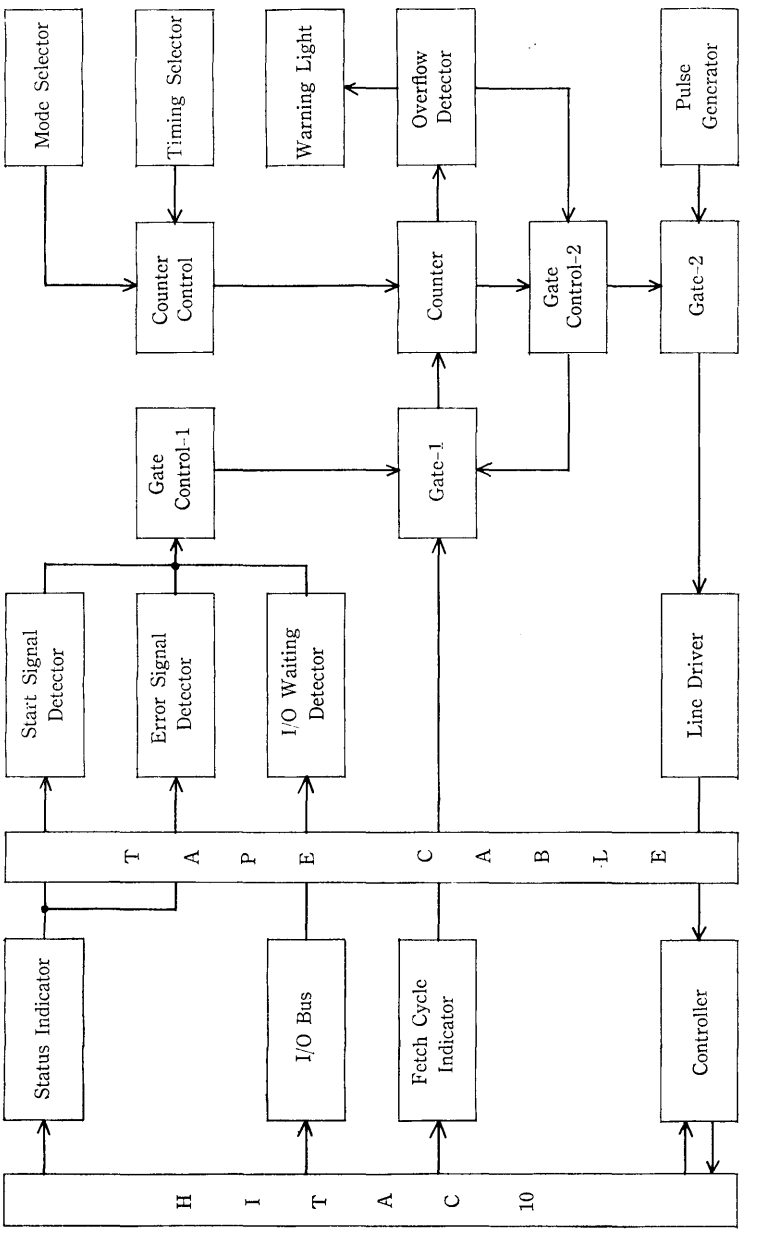


Fig. 1. The block diagram of the Brakeman.

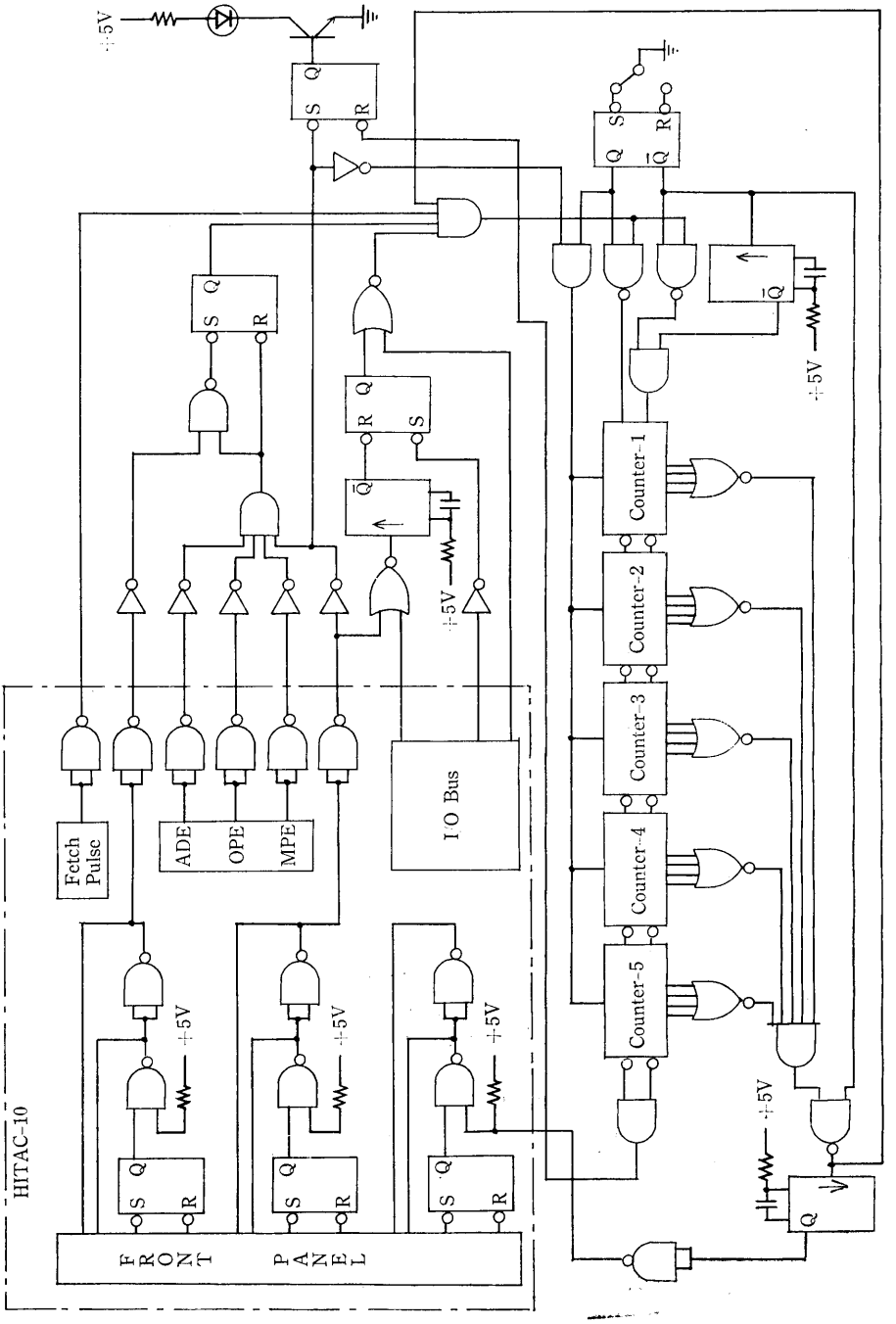


Fig. 2. The logic circuit of the Brakeman.

(1) Set the Mode Selector to 'normal' mode and clear the Counter.
(2) Initiate the execution of the user's program. When starting the CPU, the Brakeman can detect the start signal to open the Gate-1, and just after then the Counter starts counting by the fetch pulses out of the CPU.
(3) When an error breaks out, the Brakeman detects the error signal to close the Gate-1. The contents of the Counter then indicate the number of steps from the beginning of the program execution.
(4) Set the Mode Selector to 'debug' mode. Set the interval of the stop point and the error point by the Timing Selector. The value is subtracted from the Counter.
(5) Re-execute the user's program from its initial state. The Brakeman detects the start signal again and opens the Gate-1, but this time the Counter is counted down.
(6) When the contents of the Counter become zero, the Gate-1 is closed, and the Gate-2 is opened, the Brakeman transmits a stop signal to the CPU.

The Brakeman monitors the CPU at all time, so there is no necessity for doing items (1) to (3). After an error breaks out, the user shall only carry out items (4) to (6). As the Brakeman is independent of the CPU, the execution time of the CPU is not changed by the Brakeman.

Input/output equipments such as the readers, printers, data typewriters, disks, and so forth are generally used in almost all programs, and have no repeatability of their behavior in the I/O control (namely, the sequences of I/O instructions), so that the Brakeman detects the I/O instructions to close the Gate-1 until the input/output processing is finished.

This trial system can process a program of the maximum number of execution steps of, say, one million except waiting for the input/output execution. When a program exceeds this value of steps, a warning light is put on to signal the user. This value stated above can, of course, be altered by changing the length of the Counter.

It is necessary for the Brakeman to detect the condition of the CPU and to control the operation of it, this system is required to modify some logic circuits within the HITAC-10 and to add purposely several logic circuits to it. The modification is explained as follows:

As the Brakeman regards the erroneous state when an error is recognized by the CPU through the hardware interruption as the occurrence of error, the output of the flip-flop that indicates an address error (ADE), operation code error (OPE), or memory parity error (MPE), is sent to the Brakeman by the respective line drivers. The fetch pulse that indicates the fetch phase of the CPU delivered by the timing circuits is also sent to the Brakeman in the same way.

In order to detect the I/O instructions, the Brakeman uses the SERVICE QUALIFIER, STROBE, and IOFLAG that are prepared for the HITAC-10 as the standard I/O signals. Only the operation keys on the front panel of the CPU can control its operation. Whereas to make possible an external control of the CPU, that is, to stop the CPU for the Brakeman before an error breaks out, the key circuits are modified to set the external control circuits so as to facilitate the external and key operations for controlling start, stop, and reset. This operation control circuit is called the Controller. The external stop signal is used by the

Brakeman and stops the CPU at several steps before an error breaks out on the second execution of the user's program. Both start and reset signals are taken out from the above Controller to be transferred to the Brakeman by the line drivers. Thus the Brakeman can control and monitor the operation of the CPU.

3. Debugging by the Brakeman

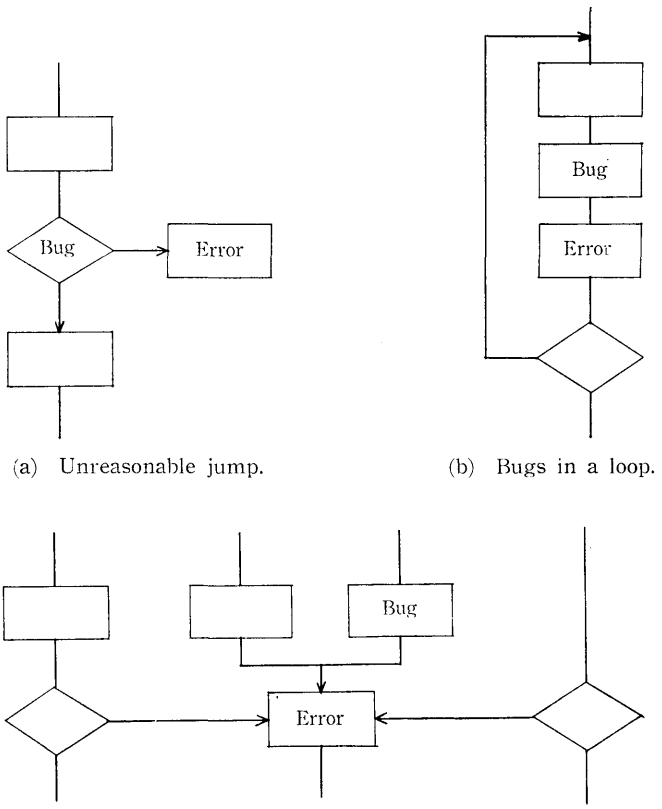
There will exist either direct or true cause which causes an abnormal state to occur, that is erroneous state, during the time when the CPU is executing a program that may have some bugs. The direct one that causes a hardware interruption is due to a result of executing an instruction, and the true one is a bug itself which causes a direct error condition. After an error breaks out during the execution of a program, the user refers to the condition of every register and counter in order to debug the program, and then he can find the direct cause and, tracing up the program, tries to back to the true cause intending to remove it. Such an error will be a simple one when the direct and true ones are the same, while it will be a complex one when those two are different. Further, if they are in general located in the distance to each other, the debugging will become more difficult. The distance between the direct and true causes can be considered to have two meanings, that is, one is the time distance (or difference) and the other the spatial distance. The latter is concerned to the difference of addresses corresponding to the locations of the two causes occurred and the former is to the time interval elapsed from the instant when a true cause occurred to the time when the direct cause was derived from the true one that brought about an abnormal condition.

The Brakeman is not affected by the spatial distance between the true and direct causes, but its effective range of adaptability will be varied within the time interval between them, and as the range is widened, the time required for debugging will increase accordingly.

The application of the Brakeman directly connected to the HITAC-10 minicomputer to the debugging of various kinds of the user's programs results in that the Brakeman is considered very effective for such bugs as shown in Fig. 3(a), (b), and (c), comparing with the usual debugging systems such as conditional stops, tracers, and so forth. Some explanations for them are given in the following:

(a) Unreasonable jump operation. This is the case when an operation code error breaks out due to the jump to such locations or area that are unable to be executed, as data or unused area. Then the contents of each register, buffer, and counter may often be destroyed before the error is generated, so the user could not be told where the jump occurred, and it may result in a possibility of the runaway of the program (or computer) or the system breakdown.

In case of the conditional stops where no information for the check points is obtained, the user has to put forward the debugging with repeated trial and error. Moreover, if he failed to set the check point location, the process should be repeated again. On the other hand, the tracer is effective



(a) Unreasonable jump. (b) Bugs in a loop.

(c) Errors at the confluence of many passes.

Fig. 3. Several examples of the bug difficult to be detected by the other existing systems.

for such a bug as stated above, but the executing speed of the program will be extremely slowed down, so that it causes a large amount of time losses for the program that may require a long execution time before an error breaks out after the start of the execution.

The Brakeman can stop the CPU at an arbitrarily specified step before the error breaks out without any information for the check point. Thus the useful information for the debugging can easily be obtained to be available.

(b) Bugs in a loop. This corresponds to the case when an error breaks out in a loop after the loop has been executed many times. It is troublesome to use such conditional stops as the address stop, for instance, because the CPU should be operated as many times as the number of loop executions. In the tracer it takes a lot of time to trace up the program even if the tracing area is selected to a limited region to speed up the processing time, because, if the tracing area is part of a loop, it should be traced as

many times as the number of looping.

The Brakeman, on the other hand, can stop the CPU just before an error breaks out, while in the normal condition the CPU executes the loop with an inherent high speed of its own.

(c) Errors at the confluence of many passes. This is the case when an error breaks out at the confluence (or junction) of many passes and a bug exists in any one of them that could not be specified to which pass it belongs. The similar cases will also occur when all the passes of a program could not be recognized by the user, or when an error breaks out in a subroutine only at a certain condition that the control has been transferred from the main program to the subroutine. In this case the check point can not be decided uniquely like case (a), but the Brakeman can stop the CPU while the control is maintained in any one of the passes or before jumping to a subroutine; thus the effective information available for debugging can be obtained.

When the Brakeman is used for debugging in order to provide for a useful information, it should be the most important problem to decide at what number of steps before an error breaks out to stop the CPU. As the time interval is longer from the stop point specified by the Brakeman to the point where an error occurred, the possibility of getting the useful information for debugging will become higher and the time required to get it longer. Namely, if a user intends to check his program by step-by-step method from the beginning of its execution, he will be able to catch the cause of errors provided they exist, but it will take a lot of time to carry out. On the other hand, however, if the Brakeman stops the CPU just before an error breaks out, it will be considered dangerous that the cause of the error has already been made at that time and could not be picked up because the useful information has already been destroyed. The optimal size of the time interval can not be decided uniquely, but depends upon the rate of error occurrences and the kind of the user's programs.

First, the unreasonable jump operation is considered. E denotes the probability with which an error breaks out within i steps after the execution of an unreasonable jump instruction, and p denotes the probability with which an error breaks out when an instruction in the location except the normal user's locations is executed. Thus the following is obtained.

$$E=p\cdot\sum_{j=0}^{i-1}(1-p)^j=1-(1-p)^i$$

where p is different with respect to the computers used, and is also different with the maximum size of the memory capacity even for the same type of computers; further, different with the internal state of the computer with the same hardware implementation.

In case of the HITAC-10 minicomputer (16 bits/word) used for the experiment, if it is assumed that the random numbers are stored in the locations except the user's area, it can be considered that $p=0.3$ or so. Therefore, if 10 steps are backed from the point of error occurrence, the error operation will be caught with the probability of 93 percent before losing the control of the CPU.

Errors in cases (b) and (c) are dependent upon the kinds of programs that the

number of steps may be left to the choice or judgement of the user, while the information concerning the point of the error occurrence being obtained, it should become very easy for the user to set up the check point or to trace up the changes of the CPU state.

The purpose of the Brakeman is to support the debugging of the program written by an assembly language at its execution stage. Namely, the user must grasp the correspondence between the addresses in the memory to be used and in his program. From this point of view it is impossible or extremely difficult for the trial Brakeman system to debug the program written by a high level language at present. On a large-scale computer system, even the system program is often coded using a high level language, but in a minicomputer programs are usually written by an assembly language and thus the user is often troubled with bugs peculiar to the assembly language used. Most bugs which cause various kinds of erroneous states during the execution stage have their primary origin at the flexibility or freedom of precisely manipulating every bit in the memory and also at the capability of making easy access to the whole area of memory by the user. Accordingly, when an error breaks out, not only the true cause of the error but also the flow of his program will be lost.

The most essential function of the Brakeman is to allow the user who has lost the control of the CPU to retry his program by means of tracing his program back to the state where he could control the CPU again. The Brakeman has some similar characteristics to the backtrace which can maintain all the information pertaining to the state changes of the CPU within the limit of its ability. The Brakeman need not have any responsibility for saving such information as those in the backtrace, but instead when any error breaks out, the user can only initiate again his program from its beginning. This offers no problem if the program is simple, but it takes a lot of time to re-execute such a program that is intended to pile up many data. Thus for such programs that have some troubles for re-execution of them, memory dump made at a certain proper time will bring the most effective way of saving the situation, and the program under debugging need only be re-started from the last dumped state, when any error breaks out.

When the true cause of an error is different from its direct one and further they both are located in the distance to each other, the Brakeman can discover the latter, but is rather difficult to recognize the former. In this case a monitor is required to look out for the place of the direct cause of an error, and to catch the moment when the true cause leads to make that place the direct one. This monitor is considered to be a special form of the conditional stop to which the Brakeman can give the information.

4. Conclusion

The bugs that can be removed only by the Brakeman are limited, whereas it gives a very important factor for the debugging to realize the condition before an error breaks out at all times and easily. The Brakeman will become more useful when it is used cooperating with the other debugging routines already developed

and is able to proceed the debugging interractively. In the case of a small-scale computer, however, such a cooperating system will be difficult and troublesome because it has usually a small size of memory as well as poor I/O equipments.

A marked tendency in the software field of data processing is to have a huge size of its capacity and a variety of complexity and hierarchy of its organization, so that it is considered to be keenly required that the CPU implemented with a debugging hardware should be brought into existence. The Brakeman can serve an effective function to support the debugging as somewhat a different approach from the usual ones.

REFERENCES

[1] AWATA, T., et al., (1974): A Simple Error Detection System in an Assembler, The Journal of the Annual Conference of the IECE in Japan, p. 1745, (Japanese).

[2] CARON, R., (1975): A Hardware Breakpoint Debugging Aid for the PDP-8/E, DECUSCOPE, **14**, 2.

[3] DEMACEDO, J., et al., (1975): Diagnostic Trace, IBM Technical Disclosure Bulletin, **17**, 9.

[4] FURUKAWA, K., et al., (1974): On a Debugging System Using the Timer, The Journal of the 15th Annual Conference of the Information Processing Society of Japan, pp. 111-112, (Japanese).

[5] SCHWARTZ, J. T., (1970): An Overview of Bugs, Debugging Techniques in Large Systems, PRENTICE-HALL, INC., Englewood Cliffs, New Jersey.