

618269

FORM 1
REGULATION 9

COMMONWEALTH OF AUSTRALIA

PATENTS ACT 1952-1973

APPLICATION FOR A PATENT

We TANDEM COMPUTERS INCORPORATED

of 19333 Vallco Parkway, Cupertino, CALIFORNIA 95014, U.S.A.

hereby apply for the grant of a Patent for an invention entitled:

METHOD AND APPARATUS FOR MEASURING PROGRAM
EXECUTION

which is described in the accompanying complete specification. This Application is a Convention Application and is based on the Application(s) numbered: 072,811 for a Patent or similar protection made in U.S.A. on 10 July 1987.

Our address for service is:

GRIFFITH HACK & CO.
71 YORK STREET
SYDNEY N.S.W. 2000
AUSTRALIA

DATED this 7th day of July, 1988.

TANDEM COMPUTERS INCORPORATED
By their Patent Attorneys


GRIFFITH HACK & CO.

TO: THE COMMISSIONER OF PATENTS
COMMONWEALTH OF AUSTRALIA
6433Ark

000835 07/07/88

PATENT DECLARATION FORM (CONVENTION)
COMMONWEALTH OF AUSTRALIA
Patents Act 1952

Regulation
12 (2)

DECLARATION IN SUPPORT OF A CONVENTION APPLICATION
FOR A PATENT

To be signed by the applicant(s) or in the case of a body corporate to be signed by a person authorised by the body corporate.

In support of the Convention application made for a patent for an invention entitled

(a) Insert title of invention.

(a) METHOD AND APPARATUS FOR MEASURING PROGRAM EXECUTION

(b) Insert full name(s) of declarant(s).

I, Patricia E. Hart

~~by~~ ^{C/O} (b) Tandem Computers Incorporated

(c) Insert address(es) of declarant(s).

of (c) 19333 Vallco Parkway, Cupertino, California 95014

do solemnly and sincerely declare as follows:-

~~1. I am/We are the applicant(s) for the patent~~

(OR, IN THE CASE OF AN APPLICATION BY A BODY CORPORATE.)

1. I am/We are authorised by TANDEM COMPUTERS INCORPORATED

....., the applicant for the patent to make this declaration on its behalf.

2. The basic application(s) as defined by Section 141 of the Act was/were made in the following country or countries on the following date(s) namely:-

(d) Insert country in which basic application(s) was/were filed.

in (d) the United States of America on (e) 10 July 1987

(e) Insert date of basic application(s).

by (f) Bruce William Bailey

(f) Insert full names of basic applicant(s).

in (d) on (e)

by (f)

in (d) on (e)

by (f)

~~3. I am/We are the actual inventor(s) of the invention referred to in the basic application.~~

(OR, WHERE A PERSON OTHER THAN THE INVENTOR IS THE APPLICANT)

(g) Insert full name(s) of actual inventor(s)

3. (g) Bruce William Bailey

(h) Insert address(es) of actual inventor(s).

of (h) 7771 Squirehill Court, Cupertino, California 95014

is/are the actual inventor(s) of the invention and the facts upon which the applicant(s) is/are entitled to make the application are as follows:

(i) Set out how applicant(s) derive(s) title from actual inventor(s) i.e., assignee of the invention from the actual inventor(s).

(i) assignment dated 29 June 1987.

Attestation or legalization not required.

4. The basic application(s) referred to in paragraph 2 of this Declaration was/were the first application(s) made in a Convention country in respect of the invention the subject of the application.

Declared at Cupertino this 7th day of July 1988

To:

Tandem Computers Incorporated

The Commissioner of Patents

By Patricia E. Hart

Signature of Declarant(s)

Name: Patricia E. Hart

Title: Assistant Secretary

(12) PATENT ABRIDGMENT **(11) Document No. AU-B-18790/88**
(19) AUSTRALIAN PATENT OFFICE **(10) Acceptance No. 618269**

- (54) Title
METHOD AND APPARATUS FOR MEASURING PROGRAM EXECUTION
- International Patent Classification(s)
(51)⁴ **G06F 011/23** **G05F 011/32**
- (21) Application No. : **18790/88** (22) Application Date : **07.07.88**
- (30) Priority Data
- (31) Number (32) Date (33) Country
072811 **10.07.87** **US UNITED STATES OF AMERICA**
- (43) Publication Date : **12.01.89**
- (44) Publication Date of Accepted Application : **19.12.91**
- (71) Applicant(s)
TANDEM COMPUTERS INCORPORATED
- (72) Inventor(s)
BRUCE WILLIAM BAILEY
- (74) Attorney or Agent
GRIFFITH HACK & CO., GPO Box 4164, SYDNEY NSW 2001
- (56) Prior Art Documents
EP 087064
US 3707725
- (57) Claim

6. A method for measuring the execution of machine-level instructions in a computer program, comprising the steps of:

selecting a set of the instructions from the computer program;

replacing each of the instructions in the selected set with a measurement instruction;

executing the program;

in response to the execution of one of the measurement instruction, performing the steps of:

identifying the measured instruction replaced by the executed measurement instruction;

indicating that the measured instruction has executed;

restoring the measured instruction by replacing the measurement instruction; and

executing the restored instruction.

(11) AU-B-18790/88
(10) 618269

-2-

18. An apparatus for measuring computer program instruction execution in a computer, the computer having at least one central processing unit for executing instructions, the apparatus comprising:

memory means which is arranged to be coupled to the central processing unit for storing data and instructions;
means coupled to said memory means for replacing a selected instruction with a measurement instruction; and

means coupled to said memory means and responsive to the execution of the measurement instruction and being arranged to

replace that the selected instruction has executed;

replace the measurement instruction with the selected instruction; and

the selected instruction.

618269

COMMONWEALTH OF AUSTRALIA

PATENTS ACT 1952

Form 10

COMPLETE SPECIFICATION

FOR OFFICE USE

Short Title:

Int. Cl:

Application Number:
Lodged:

Complete Specification--Lodged:
Accepted:
Lapsed:
Published:

Priority:

Related Art:

TO BE COMPLETED BY APPLICANT

Name of Applicant: TANDEM COMPUTERS INCORPORATED

Address of Applicant: 19333 Vallco Parkway, Cupertino,
CALIFORNIA 95014, U.S.A.

Actual Inventor: Bruce William Bailey

Address for Service: GRIFFITH HACK & CO.
71 YORK STREET
SYDNEY NSW 2000
AUSTRALIA

Complete Specification for the invention entitled:

METHOD AND APPARATUS FOR MEASURING
PROGRAM EXECUTION

The following statement is a full description of this invention,
including the best method of performing it known to me/us:-

6433A:rk

METHOD AND APPARATUS FOR MEASURING PROGRAM EXECUTION

5

10

Background Of The Invention

15

This invention relates to computer software testing tools, and more particularly, it relates to a method and apparatus for determining which program statements have been executed.

20

An important aspect of software quality assurance is the comprehensiveness of the testing. Without the assistance of special tools, it is difficult, if not impossible, to accurately determine "coverage", i.e., which program instructions are executed when test cases are run. It is therefore difficult to know which test cases to run or when to stop test case development.

25

30

35

In the prior art, there are two principal types of tools for measuring the coverage of test cases. The first type are tools that use pre-processors to modify source code before it is compiled. Procedure calls are inserted at the beginning of program blocks for keeping track of whether a block has been executed. There are several drawbacks to this approach. These tools are language-specific; that is, the preprocessors must be tailored to the syntax of the source language. The size of the object code is changed by the procedure



calls, which can change the execution characteristics of the program. Programs must be pre-processed and recompiled for measurement, and then restored or recompiled for production use or product release. The use of preprocessors not only
5 requires an extra step, it also makes these tools impractical for measuring operating system code and time-critical programs.

The second type of coverage measurement tool in the prior art uses sampling techniques to periodically read the program counter and accumulate statistics on how much time
10 is spent in predetermined memory address ranges. Histograms are developed which can be matched up with the programs load map. These tools are useful for evaluating program performance, but, because they use sampling, they cannot be
15 used to determine which instructions have not been executed.

What is needed is a true coverage measurement tool that does not require source code preprocessing, can be used on any type of program, and does not require modification of existing programs or test suites.

20 SUMMARY OF THE INVENTION

According to one aspect of the present invention there is provided in a computer system having a memory for storing a computer program and a processor for executing the stored computer program, said computer system having at least one interrupt-causing instruction and the computer program
25 comprising a first set of executable instructions stored in said memory, a method for determining which ones of a subset of said first set of executable instructions are not executed when the program is executed, comprising the steps
30 of:

making a program copy from said stored computer program, said program copy comprising a second set of executable instructions identical to said first set of executable instructions;

35 storing said program copy in said memory;



initiating the execution of the modified program copy;
in response to the execution of each one of the
interrupt-causing instructions in the program copy,
performing the steps of:

5 an instruction identical to the instruction replaced by
the executed one of the interrupt-causing instruction in
said program copy; and

restoring the replaced instruction by writing said
located instruction in place of the executed interrupt-
10 causing instruction in said program copy; and
executing the restored instruction.

According to another aspect of the present invention
there is provided an apparatus for measuring computer
program instruction execution in a computer, the computer
having at least one central processing unit for executing
instructions, the apparatus comprising:

15 memory means which is arranged to be coupled to the
central processing unit for storing data and instructions;

20 means coupled to said memory means for replacing a
selected instruction with a measurement instruction; and

means coupled to said memory means and responsive to
the execution of the measurement instruction and being
arranged to:

25 indicate that the selected instruction has executed;
replace the measurement instruction with the selected
instruction; and
execute the selected instruction.



statements. The size of the object code is not changed. The measurement tool does not noticeably degrade execution performance of the tested program.

5 According to the invention, machine-level instructions are selected for measurement and each of those instructions is replaced by a measurement instruction. Preferably, the measurement instruction is an interrupt-causing instruction. The program containing the measurement instructions is then
10 executed. When a measurement instruction is executed, the measured (replaced) instruction is identified and a value is stored to indicate that the measured instruction has been executed. Then the measured instruction is restored by replacing the measurement
15 instruction with the original instruction, and execution resumes with the restored instruction. The effect on execution performance (as measured by elapsed time) is minimal, because the replacement is performed only once for each selected instruction, even if the
20 selected instruction is executed many times.

In one embodiment of the invention, the first instruction of each block of executable object code compiled from a source language statement is selected for measurement. This provides statement level
25 coverage measurement. Each selected instruction is replaced by a breakpoint instruction, which causes non-maskable interrupts. To accommodate multiple users of the measured program, the measurement is run on a duplicate copy of the program. The results of the
30 measurement are stored in a bitmap which can be used to generate a variety of useful outputs, such as a marked-up copy of the source code indicating which source-language statements have or have not been
35 executed. These outputs can significantly increase the efficiency and effectiveness of the quality assurance effort.

Brief Description of the Drawings

Fig. 1 is a block diagram of a computer memory apparatus for the preferred embodiment.

Fig. 2 is a flow chart illustrating the operation of the ENABLE function.

Fig. 3 is a flow chart illustrating the operation of the MEASURE function.

Fig. 4 is a flow chart illustrating the operation of the program in response to a breakpoint interrupt.

Fig. 5 is a flow chart illustrating the operation of the SAVE function.

Fig. 6 is a flow chart illustrating the operation of the RESET function.

Fig. 7 is a flow chart illustrating the operation of the DISABLE function.

Fig. 8 is an example of a MARKUP report.

Fig. 9 is an example of the DISPLAY output.

Fig. 10 is an example of the SORT output.

Fig. 11 is an example of the ORDER output.

Fig. 12 is a pseudo-code representation of the algorithm used to generate the ORDER report.

Detailed Description of the Preferred Embodiment

In the preferred embodiment the method of the invention is practiced by a computer program written in the "TAL" language and executed on a "Tandem" "NonStop" computer under the control of the "GUARDIAN 90" operating system. ("Tandem", "TAL", "NonStop", and "GUARDIAN 90" are trademarks of Tandem Computers, Inc., of Cupertino California. Reference manuals for the "TAL" language and the "GUARDIAN 90" operating system, and the "NonStop" System Description Manual are available from Tandem Computers, Inc.) The following detailed description, with reference to the drawings, will enable those of ordinary skill in the art to make and use the invention.

COVER is a set of program modules, some of which are in the COVER command interpreter (COVERCOM), some in the monitor (CVRMON) and some in the operating system (in the breakpoint interrupt handler).

5

10

The preferred embodiment measures statement coverage. Statement coverage is a measure of which source code statements were (and were not) executed. Other types of coverage (such as code coverage, which is a measure of the execution of machine instructions) are within the scope of the invention, because the actual measurement is performed on selected machine instructions in the object code.

15

20

Referring to Fig. 1, a simplified block diagram illustrates the logical organization of a computer memory apparatus 4 for implementing the preferred embodiment of the invention. Two Process Control Blocks (PCBs) 6,7 are shown. In this computer system, there are up to 256 active PCBs, each controlling a process, or task. A process is a running invocation of a program associated with a single user. A single program can execute multiple processes for multiple users. PCB1 6 contains a pointer 6a pointing to program code segment 8 and a pointer 6b pointing to data area 9. PCB2 7 contains a pointer 7a which initially (before a COVER measurement is started) points (dotted arrow) to code segment 8, and a pointer 7b pointing to data area 11. When the COVER program 12 is enabled, it establishes Cover Routine Control Block (CRCB) 14. When COVER initiates a measurement on a particular process (PIN), it establishes a Cover PIN Data Block (CPDB) 16 for that process, and writes the CPDB address into one of the 256 pointers 14a in the

25

30

35



CRCB. COVER then makes a copy 10 of code segment 8, and saves pointers to code segment 8 and copy 10 into CPDB 16. COVER inserts measurement-causing instructions at selected locations in code copy 10, establishes a bitmap 18 for storing results of the measurement, and stores a pointer to copy 10 in pointer 7a of PCB2 7. The system will execute Copy 10.

COVER must be enabled by the system operator for a particular cpu (central processing unit) before users can invoke the measurement facility on that cpu. (This embodiment is implemented on a system having multiple cpu's.) The system operator issues the ENABLE command to allow measurements on a particular CPU. Referring to Fig. 2, flow chart 20 illustrates the operation of COVER in response to the ENABLE command.

At block 22, memory is allocated for a Cover Routine Control Block (CRCB). The format of the CRCB data structure is shown in Table I.

At block 24, the CRCB memory space is locked to make it memory resident and to allow absolute addressing. The CRCB is initialized at block 26. The CRCB is cleared (to zeroes and then the verifier ("CovR" in this embodiment) and other header fields are set. At block 28, the address of the CRCB is stored in a fixed predetermined location in memory (global pointer). The address of the global pointer is known to the operating system, including the breakpoint interrupt handler. COVER is now enabled, and users may perform measurements.



VERIFIER "CovR"
 VERSION
 XCVRN ^ PIN
 PrivEnabled
 MemoryRatio
 TraceBuffer
 Files to Measure [0:127]
 Control by PIN [0:255]

TABLE I

The measurement is performed as follows. The program "COVERCOM" is run to invoke the COVER command interpreter. To start a measurement of a process, the command "MEASURE" is issued with a process identification number ("PIN") identifying the process to be measured. Referring to Fig. 3, a flow chart 30 illustrates the operation of COVER in response to the MEASURE command. At block 32, a test is made to determine whether COVER is enabled. The address in the global pointer predetermined location in memory is read. If the address is non-zero, then COVER is enabled. If COVER is not enabled, then no measurement can be performed and the program exits (block 34) and informs the user with an error message.

If COVER is enabled, then at block 36, a test is made to determine whether there is a measurement in progress on the specified process. (Only one measurement is allowed at a time on a process). As seen in Table I, there is an array of 256 ControlByPin fields (one for each possible process on a cpu) in the CRCB. Each field contains a pointer to a Cover PIN Data Block, if that process is being



measured. If a process is not being measured, then the field corresponding to that PIN is zero. Thus, if the entry for the requested process is non-zero, then, at block 38, the COVER program exits and informs the user with an error message. If the entry for the requested process is zero, then COVER initiates a measurement.

At block 40, a Cover Pin Data Block (CPDB) is established for the requested process and the address is written into the CRCB ControlByPin entry for this process. (Alternatively, the Cover Pin Data Blocks may be established when COVER is enabled; in that case, the state field is used to indicate whether a measurement is in progress for a PIN.) Table II shows the format of CPDB. At block 42, the segment number for the program code is read from the code.seg field in the PCB for this process and written into the CPDB at field UC.SEG. (The segment number is equivalent to the starting address of the code because programs start on memory segment boundaries. The program counter measures the offset into a segment.) At block 44, a block of memory is allocated for a copy of the program code. At block 46, the memory is block "locked" to ensure that it will remain addressable at all times for the duration of the measurement.



5 State
 Eversaved
 Save Vol
 UC. Seg
 UC. CopySeg
 UC. Name
 UC. Timestamp
 UC. SpaceCount
 UC. Space Size
 10 UC. Bitmap Ptr.

TABLE II

15

At block 48, the program code is copied into the allocated memory block. At block 50, the segment number for this memory block is written into the CPDB at location UC.CopySeg. At block 20 52, memory is obtained (and locked down) for a "bitmap". The bitmap is used to store the results of the measurement of the process. Each bit in the bitmap corresponds to a word (2-bytes, the length of an instruction) in the object code. The bitmap is 25 initialized to zeroes and its address is stored in the CPDB at field UC.Bitmap.Ptr.

25

30

The machine instructions to be measured are now selected. Programs that are compiled on Tandem compilers with the "'Inspect' symbols" option contain a list of "statement starters" pointing to the first machine code instruction of each compiled source language statement. ("Inspect" is a trademark of Tandem Computers Inc.) For the statement coverage of this embodiment, these statement starting machine 35 instructions are selected as the set of instructions to be measured. In other embodiments, other means may be used to select the measured instructions. For example,

35



all of the code instructions may be selected (code coverage), so long as instructions can be distinguished from data in the object file.

At block 54, the statement starters
 5 are read and measurement instructions are written in place of the statement-starting machine code instructions. In this embodiment, Breakpoint instructions are used as the measurement instructions. Breakpoint instructions are used in this computer
 10 system to cause interrupts and turn control over to the corresponding interrupt handler in the operating system. In other embodiments, other measurement instructions may be used, such as an SVC (Supervisor call) or an instruction with an invalid opcode. Subroutine calls may also be used, but it is preferable
 15 to use non-maskable interrupt-causing instructions to perform the measurement from within the operating system.

At block 56, the segment number of
 20 the code copy (UC.CopySeg) is written into the PCB for this process, at the code.seg field. This field in the PCB stores the segment number for the program used to execute the process, and is the same field from which the original code segment number was copied to UC.seg. UC.seg thus stores the address of the "original" code,
 25 and the PCB has the address of the code copy. This causes the system to use the code copy, with the breakpoints, to execute this process. Other processes may use the unmodified original code while the
 30 measurement takes place on the copy.

As the code copy executes, whenever a selected instruction would be executed a breakpoint instruction is executed first. This causes a breakpoint interrupt.

35 Referring to Fig. 4, a flow chart 62 shows the effect of the breakpoint interrupt. At block 64, the breakpoint interrupt handler calls the



COVER procedure. At block 66, a test is made to determine whether COVER is enabled. This is done by reading the global pointer where the CRCB address is stored and determining whether the address is non-zero.

5 If COVER is not enabled, the called program returns (block 68) to the "standard" interrupt handler. If COVER is enabled, then, at block 70, the interrupted run environment is made addressable. Then, at block 72, a test is made to determine whether

10 the interrupted process is a measured process. The PIN of the interrupted PCB is read from a global memory location and used to index into the CRCB to determine whether there is a non-zero address pointer to a CPDB for this PIN. If the pointer for this PIN is zero, then this process is not being measured and, at block 74, control is returned to the standard interrupt handler. If the pointer for this PIN is non-zero, then the STATE field in the CPDB is checked. This field indicates whether the measurement is actually running.

15 (While the program is being copied and breakpoints are being inserted, the state field is used to indicate this interim state).

20

If the process is being measured, then, at block 76, the appropriate bit in the bitmap is set to indicate that the instruction has been executed.

25 The bit corresponding to the interrupted (measured) instruction is found as follows: The interrupted instruction is identified by reading the program counter in the program environment register to get the offset (in words) into the segment for the breakpoint instruction that caused the interrupt; this word offset is used as a bit offset from the bitmap starting address stored (at UC.BitmapPtr) in the CPDB. The bit at that offset is set to one to indicate execution of

30 the corresponding measured instruction.

35

After setting the bit in the bitmap, the breakpoint instruction in the code copy is replaced



with the actual instruction to be executed (block 78.

5 The offset from the program counter is concatenated to the segment number in the UC.Seg field in the CPDB. A low order 0 is added on (to indicate 0 byte offset into the instruction) to arrive at the 32 bit address of the instruction. The address of the instruction in the code copy is obtained in the same manner, using the UC.copy.seg field in the CPDB and the offset from the program counter. The measured
10 instruction from the original program is copied into the executing code copy, replacing the breakpoint instruction. At block 80, control is passed back to the program for execution of the actual instruction.

15 In this embodiment, the breakpoint instruction is also used for debugging. Therefore, before replacing the breakpoint instruction with the actual instruction, a test is made to determine whether a debug break point has been set. If the debugger has set a breakpoint, then COVER will find a breakpoint in the original code. A breakpoint table stores the instruction replaced by a debug breakpoint. COVER
20 will use the instruction in the breakpoint table to replace the breakpoint in the copy. Control returns to the interrupt handler instead of to the program. The interrupt handler will execute the debug routine for this breakpoint. (It will be understood that this routine is necessary only because this embodiment uses the same breakpoint instruction that is used by the
25 debugger. In other embodiments, a unique interrupt-causing instruction can be used, or concurrent debugging can be prohibited.)

30 The user can issue the SAVE command at any point to save the bitmap data. Referring to Fig. 5, flow chart 88 illustrates the operation of COVER in response to the SAVE command. At block 90, the
35 bitmap for the specified PIN is copied to another



location in memory. At block 92, a disk file is created for saving the bitmap. Then at block 94

a header is written in the bitmap file, with the measured program's name, its time stamp, etc. using the data in the CPDB for the measured process. Then at block 96 the bitmap is written to this disk file. The "eversaved" field in the CPDB is updated to indicate that the bitmap has been saved. Also, when a process terminates, if that process has been measured, the system monitor will save the bit map data.

The RESET command can be issued by the user or by the operating system after a save operation. Referring to Fig. 6, a flowchart 108 illustrates the operation of the program in response to the RESET command. At block 110, the original code segment number is copied from the CPDB field UC.SEG. into the PCB, causing the original code to be executed instead of the copy. At block 112, the code copy memory space is unlocked. At block 114, the memory space is freed (deallocated), and at block 116 the bitmap space is freed. The measurement is thus terminated.

Referring to Fig. 7, the DISABLE command is illustrated. The DISABLE command performs the opposite functions of the ENABLE command. At block 122, the program tests to determine whether COVER is enabled, and if not it exits. At block 126, zeroes are written to the global pointer containing the address of the CRCB. At block 128, the CRCB memory is unlocked. At block 130, the CRCB memory space is freed (deallocated).

The bitmap is used to generate various useful outputs.

One useful output report ("MARKUP")

generated in the preferred embodiment is a marked source code listing such as the one shown in



Fig. 8. Statements that were not executed are marked with an asterisk (*). If more than one statement occurs on the same line, a plus (+) indicates that a statement has been executed but some other statement has not executed. This report is generated using the same symbols table in the object file which was used to insert the breakpoint instructions. The symbols table indicates the offsets into the object code at which compiled source statements start. These same offsets (in bits instead of words) are used to index into the bitmap to test whether a statement-starting instruction has been executed.

The "DISPLAY" command displays a listing of all statements' source file line numbers in the measured program. All unexecuted statements are marked with an asterisk. For each procedure and for the total program, statistics are included indicating the number of statements and the percentage of those not executed. An example is shown in Fig. 9.

Another useful report ("SORT") is shown in Fig. 10. This report reports all sequences of over n unexecuted contiguous statements, where n is specified by the user/requestor. The statement starters are read and compared to the bitmap. Unexecuted, contiguous statements are tallied until an executed statement is found, and a new tally is started. The tallies are then sorted.

The "ORDER" report is shown in Fig. 11. Given an existing set of K test cases, it is useful to order the test cases so that the test with the highest coverage is run first. After this test, the next test case is selected so that it adds the most to the total coverage., etc. This will expose the program to the broadest testing in the shortest time.

First the bit maps are compressed to include only the bits corresponding to source code statement starters. If the program has N statements, the



compressed bit maps for the K test cases form a bit matrix $X(K,N)$. Referring to Fig. 12, a pseudo-code version of the algorithm for ordering the test cases is given. First, the test case J giving the most coverage is identified. Then each of the remaining bit maps is OR'ed (inclusive or) with the bit map for each of the K-1 remaining test cases, to find the combination giving the highest coverage. The combined bit map for these two cases is then or'ed with each of the K-2 remaining cases, etc.

Other boolean operations, such as AND, XOR, can also be performed on specified bit maps on request. The OR and exclusive or (XOR) can be used on bitmaps A and B to indicate the incremental increase provided by test cases. The function $(A \text{ or } B) \text{ XOR } A$ determines what test B covers that test A does not cover. The AND operation indicates which statements were tested by all test cases.

In summary, a novel technique has been described for measuring the execution of a computer program. As will be apparent to those skilled in the art, the above-described embodiment can be modified without departing from the scope of the invention. It will be understood, therefore, that the invention is defined not by the above description, but by the appended claims.

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. In a computer system having a memory for storing a computer program and a processor for executing the stored computer program, said computer system having at least one
5 interrupt-causing instruction and the computer program comprising a first set of executable instructions stored in said memory, a method for determining which ones of a subset of said first set of executable instructions are not executed when the program is executed, comprising the steps
10 of:

making a program copy from said stored computer program, said program copy comprising a second set of executable instructions identical to said first set of executable instructions;

15 storing said program copy in said memory;

modifying the stored program copy by replacing each one of a subset of said second set of executable instructions in said program copy with an interrupt-causing instruction;

initiating the execution of the modified program copy;

20 in response to the execution of each one of the interrupt-causing instructions in the program copy, performing the steps of:

25 an instruction identical to the instruction replaced by the executed one of the interrupt-causing instruction in said program copy; and

restoring the replaced instruction by writing said located instruction in place of the executed interrupt-causing instruction in said program copy; and
executing the restored instruction.

30 2. The method of claim 1 wherein the interrupt-causing instruction is a breakpoint instruction.

3. The method of claim 1 wherein the interrupt-causing instruction is an invalid instruction.



4. The method of claim 1 further comprising the step of reporting which ones of said selected subset of instructions have been restored and executed.

5 5. The method of claim 1 further comprising the step of reporting which ones of the selected subset of instructions have not been restored and executed.

6. A method for measuring the execution of machine-level instructions in a computer program, comprising the steps of:

10 selecting a set of the instructions from the computer program;

replacing each of the instructions in the selected set with a measurement instruction;

executing the program;

15 in response to the execution of one of the measurement instruction, performing the steps of:



identifying the measured instruction
replaced by the executed measurement instruction;
indicating that the measured instruction
has executed;

5 restoring the measured instruction by
replacing the measurement instruction; and
executing the restored instruction.

10 7. The method of claim 6 wherein said
identifying step comprises the step of ascertaining the
location in the program of the executed measurement
instruction.

15 8. The method of claim 7 wherein said
indicating step comprises the step of storing a
predetermined value in a storage location corresponding
to the ascertained location in the program.

20 9. The method of claim 6 further comprising
the step of making a copy of the program before
performing said replacing step.

25 10. The method of claim 9 wherein said
replacing step comprises the step of replacing the
selected instructions in the program copy and said
executing step comprises the step of executing the
program copy.

30 11. The method of claim 10 wherein said
restoring step comprises the steps of:
ascertaining the location in the program copy
of the executed measurement instruction; and
35 copying the instruction from the ascertained
location in the program to the ascertained location in
the program copy.

12. The method of claim 6 wherein said selecting step comprises the step of selecting instructions corresponding to source language statement starters.

5

13. The method of claim 6 wherein the measurement instruction is an interrupt-causing instruction.

10

14. The method of claim 13 wherein the measurement instruction is a breakpoint instruction.

15

15. The method of claim 13 wherein the measurement instruction is an invalid instruction.

16. The method of claim 6 wherein the measurement instruction is a subroutine call.

20

17. A method for measuring the execution of machine-level instructions in a computer program comprising the steps of:

making a copy of the computer program;

selecting a set of the instructions from the computer program;

25

replacing each of the selected instructions in the program copy with an interrupt-causing instruction;

30

initializing a map having a set of storage locations corresponding to the selected set of instructions;

executing the program copy;

in response to an interrupt, performing the steps of:

35

ascertaining the location in the program of the interrupted instruction;

storing a predetermined value in the map storage location corresponding to the instruction replaced by the interrupted instruction;

restoring the instruction by copying the instruction at the ascertained location in the program to the ascertained location in the program copy; and
executing the restored instruction.

18. An apparatus for measuring computer program instruction execution in a computer, the computer having at least one central processing unit for executing instructions, the apparatus comprising:

memory means which is arranged to be coupled to the central processing unit for storing data and instructions;

means coupled to said memory means for replacing a selected instruction with a measurement instruction; and

means coupled to said memory means and responsive to the execution of the measurement instruction and being arranged to

replace that the selected instruction has executed;

replace the measurement instruction with the selected instruction; and

the selected instruction.

19. The apparatus of claim 18 wherein the measurement instruction is an interrupt-causing instruction.

20. The apparatus of claim 19 wherein the measurement instruction is a breakpoint instruction.

21. The apparatus of claim 19 wherein the measurement instruction is an invalid instruction.



22. The apparatus of claim 18 wherein the measurement instruction is a subroutine call.

5 23. In a computer having at least one central processing unit for executing instructions, an apparatus for measuring the execution of machine-level instructions in a computer program, comprising:

memory means coupled to the central processing unit for storing instructions and data;

10 means coupled to said memory means for selecting a set of instructions from the computer program;

15 means coupled to said memory means for replacing each of the instructions in the selected set with a measurement instruction;

means coupled to said memory means for executing the program; and

20 means coupled to said memory means and responsive to the execution of one of the measurement instructions for:

identifying the measured instruction replaced by the executed measurement instruction;

25 indicating that the measured instruction has executed;

restoring the measured instruction by replacing the measurement instruction; and
executing the restored instruction.

30 24. The apparatus of claim 23 wherein said identifying means comprises means for ascertaining the location in the program of the executed measurement instruction.

35 25. The apparatus of claim 24 wherein said indicating means comprises means for storing a predetermined value in a storage location in said

memory corresponding to the ascertained location in the program.

5 26. The apparatus of claim 23 further comprising means coupled to said memory for making a copy of the program.

10 27. The apparatus of claim 26 wherein said replacing means comprises means for replacing the selected instructions in the program copy and said executing means comprises means for executing the program copy.

15 28. The apparatus of claim 27 wherein said restoring means comprises:

 means for ascertaining the location in the program copy of the executed measurement instruction; and

20 means for copying the instruction from the ascertained location in the program to the ascertained location in the program copy.

25 29. The apparatus of claim 23 wherein said selecting means comprises means for selecting instructions corresponding to source language statement starters.

30 30. The apparatus claim 23 wherein the measurement instruction is an interrupt-causing instruction.

 31. The apparatus of claim 30 wherein the measurement instruction is a breakpoint instruction.

35 32. The apparatus of claim 30 wherein the measurement instruction is an invalid instruction.

33. The apparatus of claim 23 wherein the measurement instruction is a subroutine call.

34. In a computer having at least one central processing unit, an apparatus for measuring the execution of machine-level instructions in a computer program comprising:

memory means coupled to the central processing unit for storing instructions and data;

means coupled to said memory means for making a copy of the computer program;

means coupled to said memory means for selecting a set of instructions from the computer program;

means coupled to said memory means for replacing each of the selected instructions in the program copy with an interrupt-causing instruction;

means coupled to said memory means for initializing a map having a set of storage locations corresponding to the selected set of instructions;

means coupled to said memory means for causing the central processing unit to execute the program copy; and

means coupled to said memory means and responsive to an interrupt for:

ascertaining the location in the program of the interrupted instruction;

storing a predetermined value in the map storage location corresponding to the instruction replaced by the interrupted instruction;

restoring the instruction by copying the instruction at the ascertained location in the program to the ascertained location in the program copy; and
executing the restored instruction.

35. A method for measuring computer program instruction execution substantially as hereinbefore described with reference to the accompanying drawings.

5 36. Apparatus for measuring computer program instruction execution in a computer substantially as hereinbefore described with reference to the accompanying drawings.

10 37. The method of claim 1 further comprising the step of, in response to the execution of an interrupt-causing instruction, indicating that the instruction replaced by the interrupt-causing instruction will be executed.

15 38. The method of claim 6 wherein said indicating step comprises storing in said computer system an indication that the interrupt-causing instruction has executed.

20 39. The method of claim 7 further comprising the step of reporting that each restored instruction has been executed.

Dated this 19th day of September 1991.

TANDEM COMPUTERS INCORPORATED

By their Patent Attorney

GRIFFITH HACK & CO.



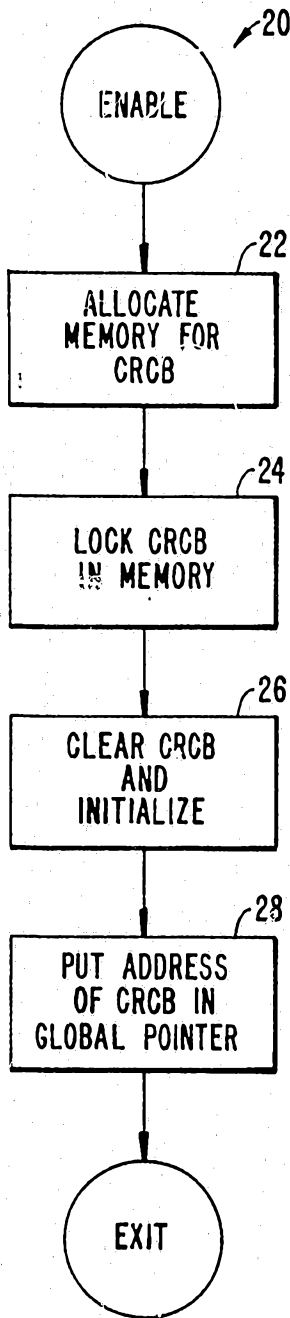


FIG. 2.

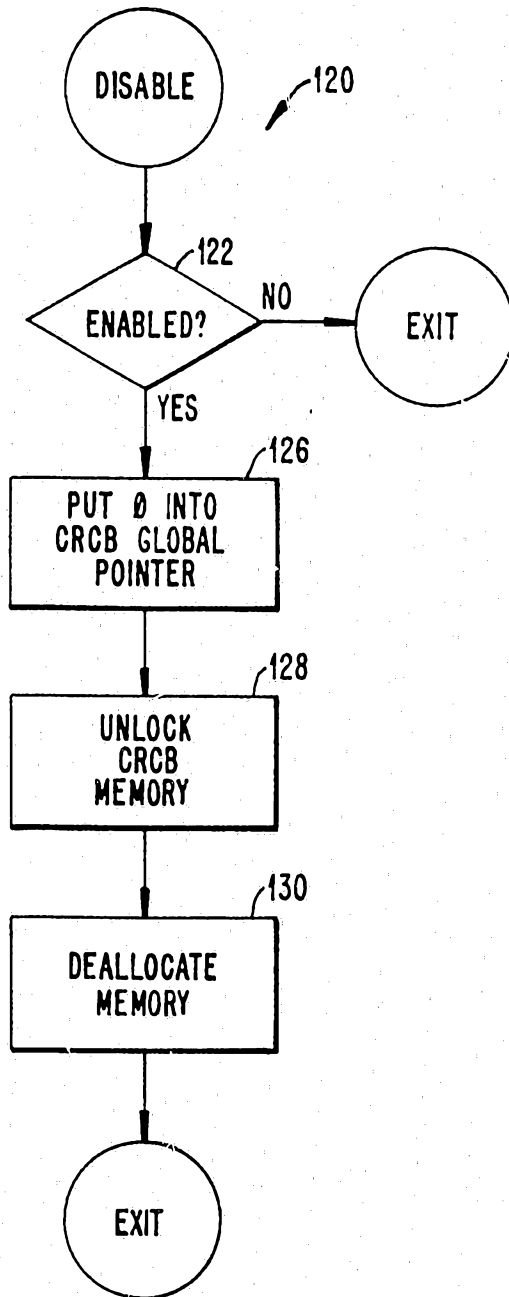


FIG. 7.

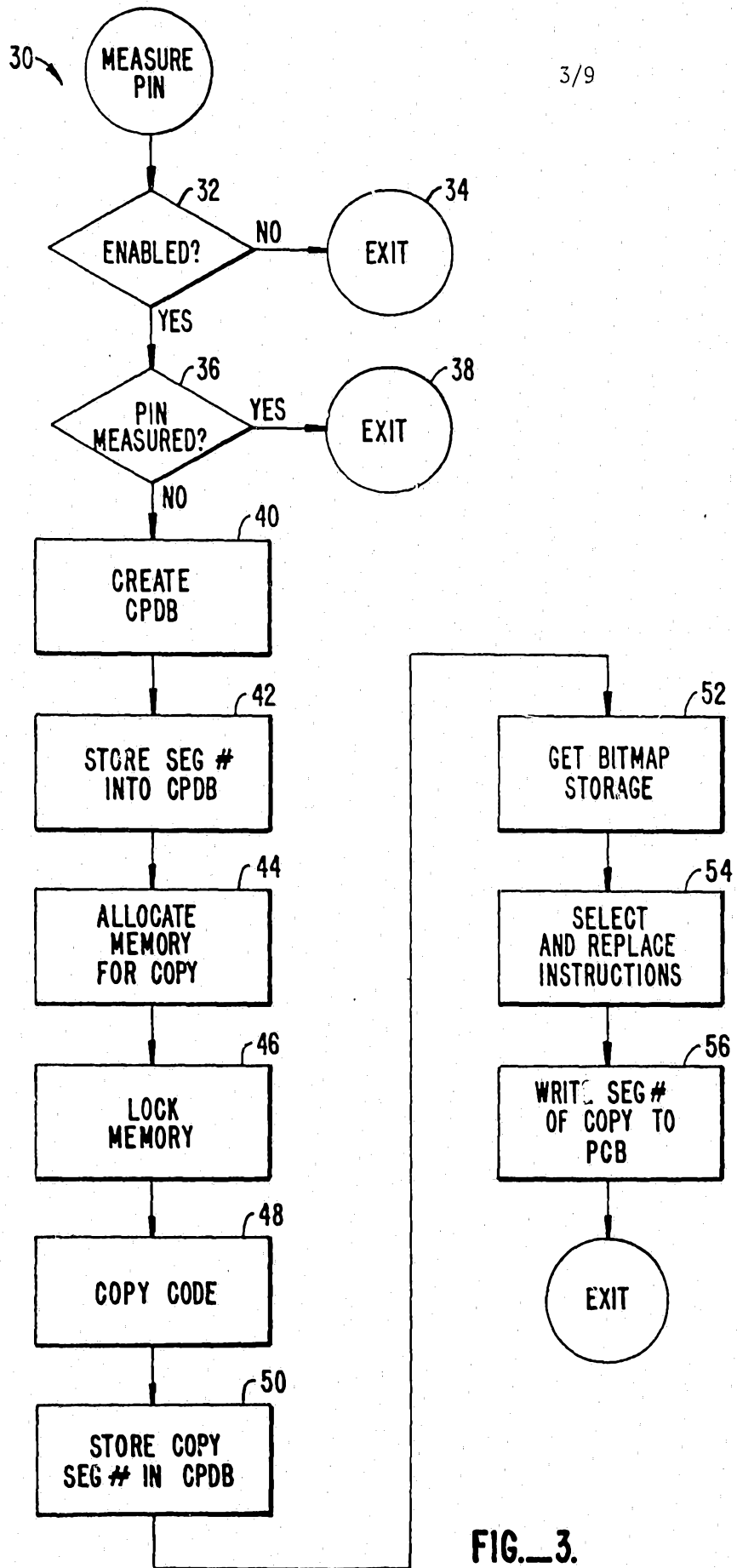


FIG. 3.

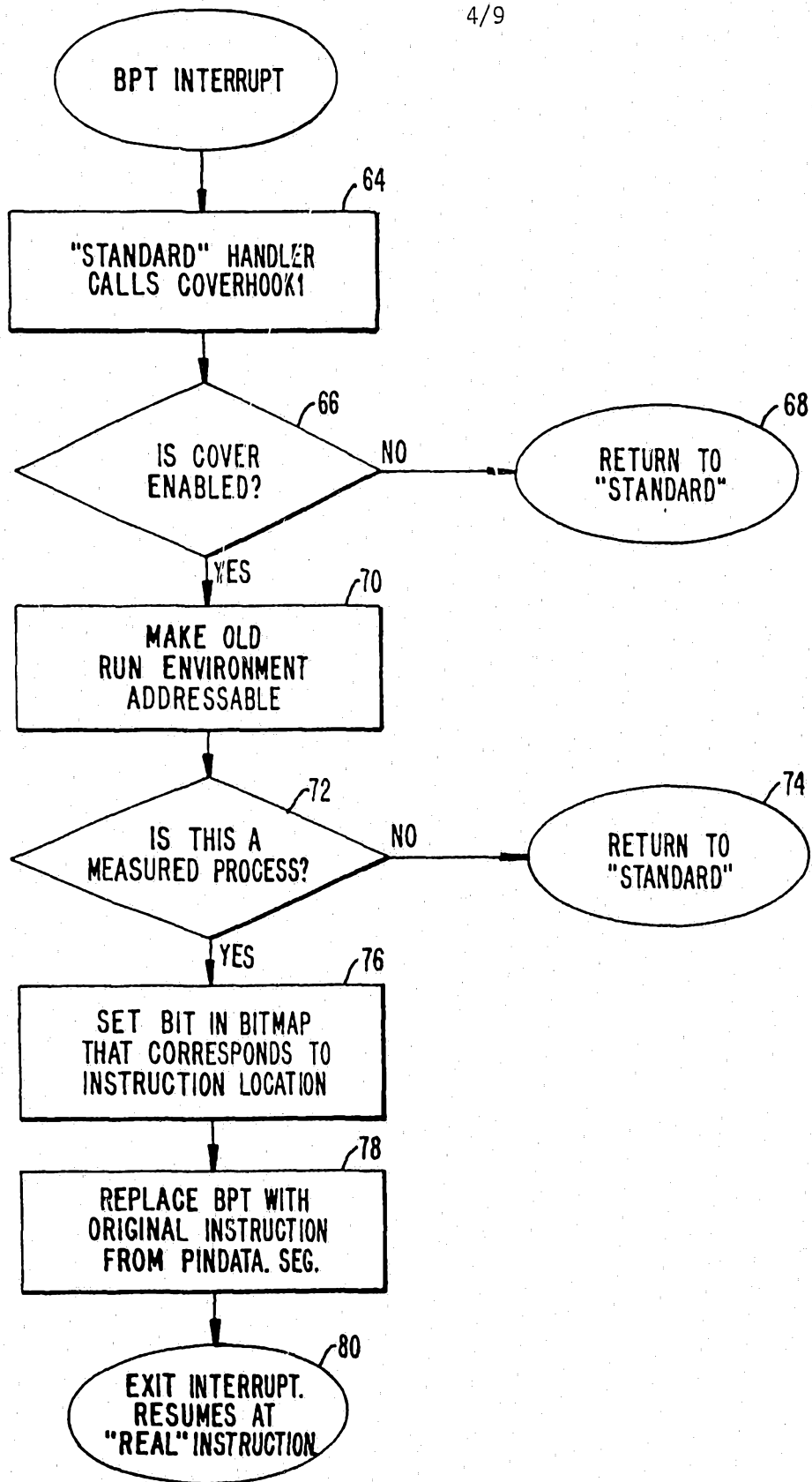


FIG. 4.

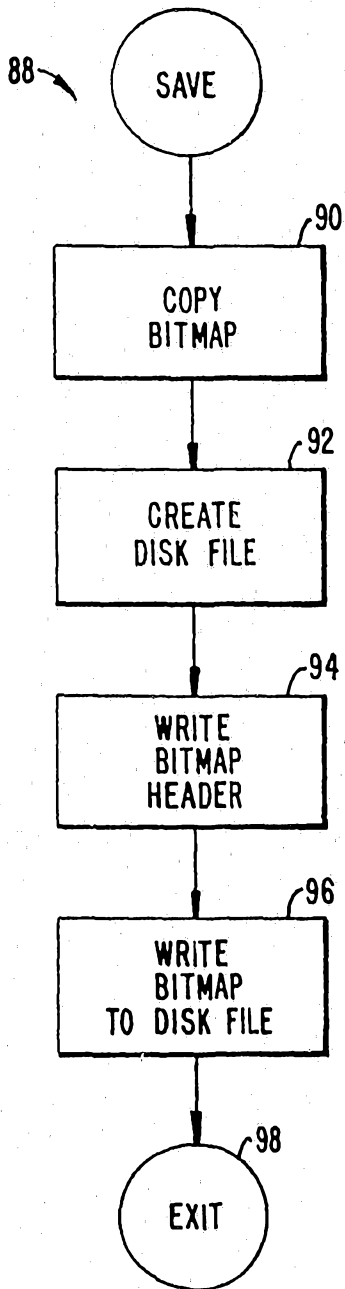


FIG._5.

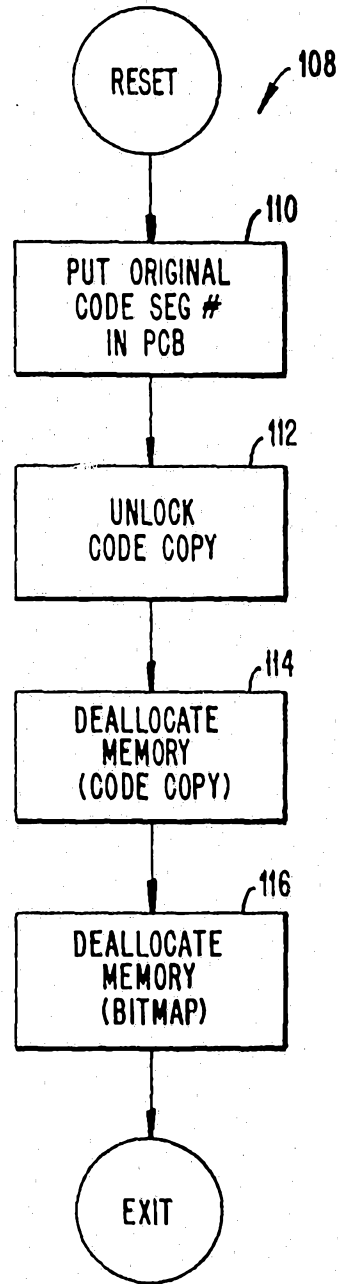


FIG._6.

PAGE 1 \$OVER.DEMO.SAMP [1]

TAL - T9250B40 - (15SEP86)
DATE - TIME: 04FEB87 - 08:56:54

6.	000000	INT PROC ABS(ARG); !FUNCTION DEFINITION	
7.	000000	INT ARG;	
8.	000000	BEGIN	
9.	000000	IF ARG > 0 THEN	
10.	*000003	RETURN ARG	
11.	000003	ELSE	
12.	000006	RETURN -ARG;	
13.	000011	END;	
14.	000000		
15.	000000	PROC MAINSTREET MAIN;	
16.	000000	BEGIN	
17.	000000	INT I, J;	
18.	000000		
19.	000000	CALL INITIALIZER; !EXTERNAL SYSTEM PROCEDURE	
20.	000006	I: = 0;	
21.	000010		
22.	000010	WHILE I DO	
23.	*000012	I: = 0;	!NEVER EXECUTED
24.	000015		
25.	+000015	IF I THEN J: = -J;	!SECOND STATEMENT NOT
26.	000022		
27.	000022	J: = ABS(I);	!FUNCTION CALL
28.	000026		
29.	000026	I: = 2; J: = 3;	!BOTH STATEMENTS EXECUTED
30.	000032		
31.	+000032	CALL STOP; J: = 0;	!SECOND STATEMENT NOT
32.	000041		
33.	*0000	END;	!IMPLICIT CALL STOP

FIG. 8.

EXECUTION REPORT. STATEMENTS MARKED "*" WERE NOT EXECUTED.
 BITMAP IS FROM \$SYSTEM.BWBDEMO.ZZCOAMP
 PROGRAM FILE IS \$SYSTEM.BWBDEMO.AMP DATED 4 FEB 1987, 8:56:54

SPACEID 00 WORD OFFSET 000004 IS BASE OF ABS.

ENTRY POINT AT WORD OFFSET 000000 IS ABS

PROCEDURE HEADER AT LINE 6. IN \$OVER.BWBDEMO.SAMP

SOURCE FILE TIMESTAMP WAS 4 FEB 1987, 8:56:30.

WORD OFFSET 000000 IS LINE 9. IN \$OVER.BWBDEMO.SAMP

*WORD OFFSET 000003 IS LINE 10. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000006 IS LINE 12. IN \$OVER.BWBDEMO.SAMP

OF 3 STATEMENTS, 1 (33.3%) WERE NOT EXECUTED.

SPACEID 00 WORD OFFSET 000015 IS BASE OF MAINSTREET.

ENTRY POINT AT WORD OFFSET 000000 IS MAINSTREET

PROCEDURE HEADER AT LINE 15. IN \$OVER.BWBDEMO.SAMP

SOURCE FILE TIMESTAMP WAS 4 FEB 1987, 8:56:30.

WORD OFFSET 000000 IS LINE 15. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000001 IS LINE 19. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000006 IS LINE 20. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000010 IS LINE 22. IN \$OVER.BWBDEMO.SAMP

*WORD OFFSET 000012 IS LINE 23. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000015 IS LINE 25. IN \$OVER.BWBDEMO.SAMP

*WORD OFFSET 000017 IS LINE 25. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000022 IS LINE 27. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000026 IS LINE 29. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000030 IS LINE 29. IN \$OVER.BWBDEMO.SAMP

WORD OFFSET 000032 IS LINE 31. IN \$OVER.BWBDEMO.SAMP

*WORD OFFSET 000037 IS LINE 31. IN \$OVER.BWBDEMO.SAMP

*WORD OFFSET 000041 IS LINE 33. IN \$OVER.BWBDEMO.SAMP

OF 13 STATEMENTS, 4 (30.8%) WERE NOT EXECUTED.

OF 16 STATEMENTS, 5 (31.3%) WERE NOT EXECUTED.

FIG.—9.

BITMAP IS FROM \$A.BWB.ZZC00006
PROGRAM FILE IS \$A.BWB.BRANDY DATED 26 JAN 1987, 15:47:38

PROCEDURE NAME	SOURCE FILE	LINE NUMBER	OFFSET	STTMNTS
CREATE ^ BACKUP	SBRANDY	696.	000000	55
SET ^ RUN ^ TIME ^ OPTIONS	SBRANDY	572.	000253	54
DOSHOW ^ MENU	SBRANDY	1369.	000000	54
ANALYZE ^ CHECK ^ STATUS	SBRANDY	749.	000000	26
ANALYZE ^ SYSTEM ^ MES	SBRANDY	794.	000000	23
SET ^ RUN ^ TIME ^ OPTIONS	SBRANDY	547.	000001	18
CHECKRECEIVE	SBRANDY	832.	000000	15
OPEN ^ BACKUPS ^ FILES	SBRANDY	672.	0000 22	11
CHECK ^ VERSION	SBRANDY	656.	0000 21	9
GET ^ TOKEN	SBRANDY	464.	0000 27	6
OPEN ^ PRIMARYS ^ FILES	SBRANDY	636.	0000 25	6
EXECUTE	SBRANDY	1443.	0000 71	6
BRANDY	SBRANDY	1469.	0000 33	5
GOTO ^	SBRANDY	1033.	0000 55	4
GOTO ^	SBRANDY	1051.	0001 25	3
XCAL ^	SBRANDY	1179.	0001 05	3
OUTPUT ^	SBRANDY	345.	0000 24	2
IF ^	SBRANDY	1201.	0000 42	2
BRANDY	SBRANDY	1485.	0001 24	2

FIG. 10.

PROGRAM FILE IS \$A.BWB.BRANDY DATED 26 JAN 1987, 15:47:38
MEASUREMENT OF \$A.BWB.BRANDY WHICH COMPRISES 771 STATEMENTS

SEQUENCE	FILENAME	% COVERAGE	% CUMULATIVE
1	ZZC00006	57.1%	57.1%
2	ZZC00004	25.3%	64.3%
3	ZZC00002	55.3%	69.9%
4	ZZC00005	53.7%	71.6%
5	ZZC00003	56.5%	71.7%
6	ZZC00001	49.8%	71.7%

FIG. 11.

```

PROC ORDER( BIGK, BIGN, X);
  INTEGER BIGK, BIGN;                --ARRAY DIMENSIONS.
  BIT ARRAY X( BIGK, BIGN);          --TEST MEASUREMENT RESULTS.
  BIT ARRAY V( BIGN);                --CUMULATIVE MEASUREMENT RESULT
-- TEMPORARY VARIABLES
  INTEGER SUMJ, SUMK, SUMMAX, j, k, MAX;
  BIT ARRAY SWAP( BIGN);

-- THE NOTATION "*" AS AN ARRAY SUBSCRIPT MEANS "ALL ELEMENTS
-- ALONG THAT COORDINATE TAKEN ONE AT A TIME."
-- THE FUNCTION "LOR" IS THE BOOLEAN INCLUSIVE OR.
-- THE FUNCTION "ADDELEMENTS" COMPUTES THE SUM OF THE ELEMENTS
-- OF ITS VECTOR ARGUMENT.
V(*) := 0;
FOR k: = 1 TO BIGK-1 DO
  BEGIN
    SUMMAX := SUMK := ADDELEMENTS( V(*) LOR X(k,*));
    MAX := k+1;
    --FIND THE TEST CASE WHICH HELPS THE MOST.
    FOR j: = MAX TO BIGK DO
      BEGIN
        SUMJ := ADDELEMENTS ( V(*) LOR X(j,*));
        BEGIN
          --ESTABLISH A NEW BOUND
          SUMMAX := SUMJ;
        END;
      END; --j
    IF SUMMAX > SUMK THEN
      BEGIN
        --SWAP X(k,*) WITH X(MAX,*)
        SWAP(*) := X(k,*);
        X(k,*) := X(MAX,*);
        X(MAX,*) := SWAP(*);
      END;
    V(*) := X(k,*) LOR V(*);
  END; --k

```

FIG. 12.