(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 March 2001 (22.03.2001)

PCT

(10) International Publication Number
WO 01/20453 A1

(51) International Patent Classification⁷: G06F 11/00

(21) International Application Number: PCT/US00/25474

(22) International Filing Date: 14 September 2000 (14.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/398,912 14 September 1999 (14.09.1999) US

(71) Applicant: **QUALCOMM INCORPORATED** [US/US]; 5775 Morehouse Drive, San Diego, CA 92121-1714 (US).

(72) Inventors: **LEE, Way-Shing**; 8555 Foucaud Way, San Diego, CA 92129 (US). **FOERSTER, Gregory, B.**; 10149 Tilton Street, San Diego, CA 92126 (US). **ZHANG, Li**; 17116 Patina Street, San Diego, CA 92127 (US). **ZOU, Qiuzhen**; 5791 Rutgets Road, La Jolla, CA 92037 (US).

(74) Agents: **WADSWORTH, Philip, R.** et al.; Qualcomm Incorporated, 5775 Morehouse Drive, San Diego, CA 92121-1714 (US).

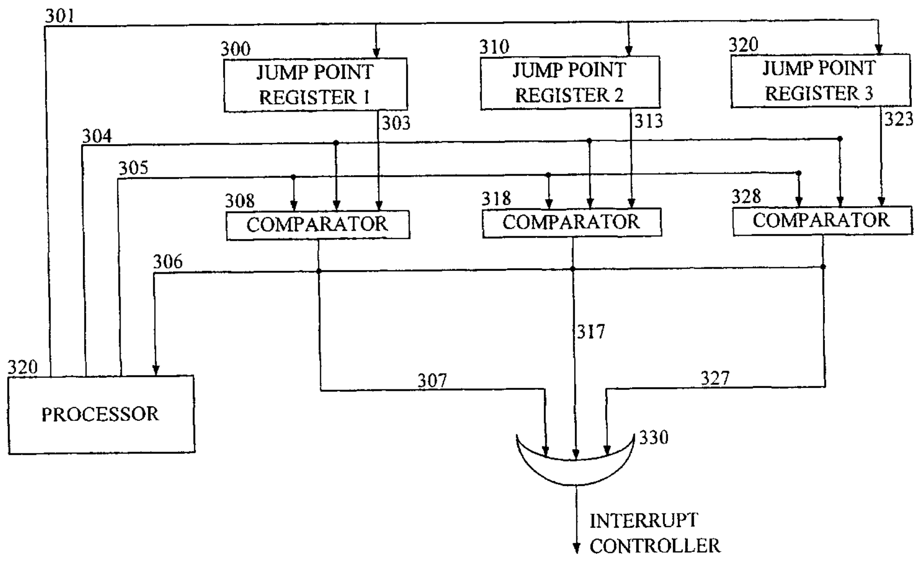
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*
— *Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.*

[Continued on next page]

(54) Title: METHOD AND APPARATUS FOR MODIFYING MICROINSTRUCTIONS IN A STATIC MEMORY DEVICE



(57) Abstract: Method and apparatus for modifying the program flow of microinstructions residing in a static memory device. When microinstructions from the static memory device need to be modified, a jump point register is used to hold a jump point address that triggers an interrupt event. When the current program counter contains an address equal to the jump point address and the jump point register is enabled, an interrupt event is generated that redirects the program flow away from the static memory device to a programmable memory device. Rather than using the interrupt to indicate the occurrence of an external event, the interrupt is used to bypass portions of the microinstructions residing in the static memory device.

METHOD AND APPARATUS FOR MODIFYING MICROINSTRUCTIONS IN A STATIC MEMORY DEVICE

BACKGROUND

I. Field of the Invention

The present invention relates to the field of control stores for microprocessors. More particularly, the invention relates to the modification of a control store apparatus that utilizes both a Read-Only Memory (ROM) and a Random Access Memory (RAM).

II. Background

Control stores contain executable microinstructions that control the data path of a microprocessor. On some machines, the control store consists of RAM, in others, the control store is ROM. The contents of a RAM can easily be rewritten with new information. However, RAM is volatile, i.e., the contents of RAM are retained only during the time period when power is supplied to the circuit. In contrast, the contents of a ROM are inserted at the time of the ROM's manufacture and cannot be changed or erased, even when powered off.

ROM is much cheaper than RAM when produced in large volumes. Due to cost considerations of RAM and ROM, microcode programmers generally design new circuitry with RAM so that programming mistakes can be easily corrected, but substitute RAM with ROM in the final design stage to minimize production costs. However, even the most rigorous design review can miss programming errors that will then be permanently embedded in the static ROM.

When programming mistakes are discovered in the microinstruction set stored in ROM, programmers create patches to correct the mistakes. "Patch" is a term of art that refers to new code introduced to fix prior code or to add functionality. Microinstructions as a whole are referred to as "code" and designed in a modular manner, wherein the entire code consists of separate subroutines. Hence, an error in one portion of the code can be isolated and corrected without having to rewrite the entire code. When a faulty subroutine is discovered, a programmer will create a duplicate subroutine, without the errors, which is called by the program flow instead of the faulty subroutine. This technique is possible by utilizing system RAM and ROM together for storing the microinstruction set. Subroutines are generally stored in ROM, whereas the main program code that calls the subroutines is generally stored in RAM. As the program flow proceeds down the microinstruction set in RAM, exit points from the main code allow the program flow to execute the microinstructions of the subroutines in ROM. When a subroutine has been executed, the program flow exits the subroutine and reenters the main code in RAM. However, when an error in a subroutine is discovered, the exit point corresponding to the faulty subroutine is disabled and the programmer must patch the mistake with a new subroutine. Since ROM is static, this new subroutine must be stored in RAM.

This method is inflexible due to the inability to enter and exit faulty subroutines except at predefined points in the main code. In addition, this method wastes space in RAM since an entire subroutine must be duplicated in RAM in order to fix a programming error within the subroutine, no matter how minor the error. There is a present need in the art to have more flexible exit and entry points in the program flow between ROM and RAM. In addition, there is a present need to minimize the size of RAM required to fix programming errors in ROM.

SUMMARY OF THE INVENTION

The present invention is directed to a method and apparatus for modifying the program flow of microinstructions residing in a ROM in order for programmers to have more direct access and control over the ROM microinstructions. Since the microinstructions in ROM cannot be altered, any changes to the microinstruction set must necessarily be made in RAM. The present invention allows a programmer to directly access a programming error within ROM without having to duplicate the entire subroutine in RAM. The present invention can also allow a programmer to add functionality to outdated ROM rather than replacing the outdated ROM with a newly designed ROM.

The exemplary embodiment of the present invention is a method for modifying the program flow in a static memory device, the method comprising the step of generating an interrupt that triggers a jump from the static memory device to a programmable memory device.

In one embodiment of the invention, a jump point register is used to hold a jump point address. This jump point address can trigger an interrupt event in the program flow. A program counter contains the address of the current microinstruction in the program flow. If the program counter holds an address that equals the jump point address in the jump point register, the interrupt event is generated. This interrupt event initiates an alteration in the program flow from a static memory device to a programmable memory device.

In one embodiment of the invention, the interrupt event can be used to repair programming errors in the static memory device. A programmer can create a patch for a defective portion of code residing in the static memory device. The programmer can then store an exit address in a register or other storage device wherein the exit address corresponds to an address of a predetermined microinstruction within the defective code portion. The exit address is compared to all the microinstructions being executed in the program flow. The microinstructions in the patch are executed when the predetermined microinstruction occurs in the program flow.

In another embodiment of the invention, the interrupt event can produce an alteration in the program flow that adds functionality to the static memory device. The alteration can be in the form of additional code stored in a programmable memory device, which can be executed in the midst of the execution of microcode from the static memory device.

In another embodiment of the invention, multiple jump point registers can be coupled to an interrupt port along with corresponding comparators, which can enable or disable each individual jump point register. Each individual jump point register can be associated with separate interrupt events. Hence, using multiple jump point registers gives flexibility to a programmer and allows her to advantageously allocate jump points according to future need.

This modification reduces the requirement of RAM size for fixing firmware mistakes in ROM and improves the functionality of ROM.

Still other objects and advantages of the present invention will become readily apparent to those skilled in the art from the following detailed description.

BRIEF DESCRIPTIONS OF THE DRAWINGS

FIG. 1 is a diagram showing a conventional program flow between RAM and ROM.

FIG. 2 is a diagram showing the program flow between RAM and ROM in an embodiment of the present invention.

FIG. 3 is a block diagram of a circuit for implementing program flow between RAM and ROM.

FIG. 4 is a block diagram of a data processing system.

DETAILED DESCRIPTION

FIG. 1 is a block diagram showing a prior art implementation of an error correcting method, i.e., "debugging" method, for microinstructions in a data processing system such as a computer system or a general purpose microcomputer. For illustrative purposes, the preferred embodiments of the invention are described using ROM and RAM. However, it will be readily apparent in the detailed description below that the methods described are applicable for use with any static storage device and volatile storage device. In FIG. 1, a RAM **100** has been programmed with code that calls subroutines in a ROM **110**. A program counter (not shown) containing the address of the microinstruction next to be executed proceeds down the RAM stack until the program counter encounters the address of a microinstruction in the ROM stack. At point **101**, the program flow exits the microinstruction set in RAM **100** and enters the microinstruction set in ROM **110**. The program counter proceeds down the ROM **110** until it encounters the address of a microinstruction in RAM **100**. The program flow exits the microinstruction set in ROM **110** and reenters the microinstruction set in RAM **100** at point **102**. This process repeats itself for the various subroutines stored in ROM **110**. However, if a programming error needs be corrected in ROM **110**, or a different functionality needs to be added, a programmer could reprogram the RAM **100** so that the subroutine stored in ROM **110** can be bypassed. The present practice among those skilled in the art is to debug faulty subroutines by duplicating the entire subroutine in RAM **100**, absent programming errors, and reprogramming the RAM **100** to carry the program flow to the duplicated subroutine in RAM **100**, rather than to the faulty subroutine stored in ROM **110**.

As illustrated in FIG. 1, if a programming error **106** is discovered in subroutine **105**, then the programmer would have to disable the data path **103** from RAM **100** to ROM **110** and create a new data path **104** to a replacement subroutine **107** residing in RAM **100**. When subroutine **107** is complete, the data path **108** flows back to any designated microinstruction located in RAM **100** after the disabled data path **103**.

This can be a large waste of RAM resources when the size of the programming error is minimal. Because ROM is static, a programmer cannot change the microinstructions in ROM to redirect the program flow to RAM from a different point in ROM, even if only a small portion of the subroutine need be rewritten.

FIG. 2 is a block diagram showing the program flow between RAM **200** and ROM **210** in an embodiment of the present invention that allows a programmer to debug a faulty subroutine stored in ROM **210** without having to replicate the entire subroutine in RAM **200**. In addition, a programmer may include additional features and functions within the subroutines stored in ROM

210. The program flow exits RAM **200** and enters ROM **210** as in FIG. 1 during an error-free portion of the microcode. However, when a bug is discovered in the ROM **210**, the program flow of FIG. 2 allows the programmer to create a patch for the bug without sacrificing a large portion of the RAM **200** in order to correct the bug.

A program counter (not shown) proceeds down the RAM stack until the program counter encounters the address of a microinstruction in the ROM stack. At point **201**, the program flow exits the microinstruction set in RAM **200** and enters the microinstruction set in ROM **210**. The program counter proceeds down the ROM **210** until it encounters the address of a microinstruction in RAM **200**. The program flow exits the microinstruction set in ROM **210** and reenters the microinstruction set in RAM **200** at point **202**. This process repeats itself for the various subroutines stored in ROM.

Program flow **203** continues to the subroutine **208** containing the programming error **205**. Error-free instructions in the subroutine are executed until point **204**, at which point the program flow returns to RAM **200** for a patch. At point **206**, the patch has been completed and the program flow returns to ROM **210**. When the subroutine is complete, the data path **207** returns to RAM **200**.

In an exemplary embodiment of the present invention, an interrupt circuit is introduced to a prior art data processing system. FIG. 3 is a block diagram of an interrupt circuit that will allow a programmer to create a program flow as shown in FIG. 2.

The interrupt circuit of FIG. 3 consists of a plurality of registers, or any other storage device capable of storing a microinstruction address, and are referred to generically as jump point registers. A jump point register holds a jump point address that triggers an interrupt event. As is well known in the art, an interrupt causes the temporary suspension of a process when an external event occurs outside of that process. An interrupt signal indicates the occurrence of an interrupt event so that the processor suspends the current process and performs the task requested by the interrupt signal. However, in this embodiment of the invention, interrupt signals are used in a very different way. Rather than using the interrupt to indicate the occurrence of an external event, the interrupt circuit of FIG. 3 uses an interrupt to bypass portions of the microcode residing in ROM.

The interrupt circuit embodied in FIG. 3 includes three (3) jump point registers **300, 310, 320.** However, it should be apparent to one skilled in the art that the number of jump point registers can vary according to a circuit designer's preference without affecting the scope of the present invention. Each of the three (3) jump point registers is individually coupled to one of three comparators **308, 318, 328,** respectively, through lines **303, 313, 323.** A comparator is a device that compares two input words and is generally composed of EXCLUSIVE-OR gates, but for this or any other embodiment of the invention, any device that can accomplish a comparison function can be used. However, for illustrative purposes, the term "comparator" will be used. Each comparator **308, 318, 328,** is coupled to the processor **320** through control lines **304, 305.**

Each jump point register **300, 310, 320,** is set with an address corresponding to an interrupt event. Line **301** loads the addresses into each jump point register from the processor **320.**

Control line **304** carries the contents of the program counter to the comparators **308, 318, 328.** The program counter is a register that contains the address of the microinstruction to be executed next. In some data processing systems, the program counter is designed to contain the current microinstruction being executed. It would be apparent to one skilled in the art that the contents of the program counter need not be limited to a predictive state or a current state in order to implement any embodiment of the invention.

Control line **305** carries a control signal from the processor **320** that enables or disables each comparator **308, 318, 328,** in order to achieve the desired functionality associated with each jump point register.

Control line **306** carries a status signal from comparators **308, 318, 328** to status registers (not shown) in the processor **320**, indicating which jump point register contained the same address as the program counter. Placement of the status registers in the processor **320** is merely a matter of design choice and does not effect the scope of the invention. It should be noted that in an alternative embodiment of the invention, the program counter could be used to identify which jump point register has the same address as the program counter. If a program counter is used, then all the bits in an address must be checked to identify the jump point register in question. If a status register is used, then only one bit need be checked. The choice of using the program counter or the status registers for the purpose of identifying which jump point register contains the jump point address associated with the current interrupt is merely a matter of design choice.

If the address in the program counter is equal to one of the addresses located in jump point registers **300, 310, 320** and a control signal is sent that enables the comparator corresponding to the aforementioned jump point register, then a status signal is sent to the processor **320** and a signal is sent to an interrupt controller (not shown) so that an interrupt occurs.

In this instance, the interrupt event is not an external event which calls for an suspension of the program flow, rather, the interrupt event redirects the program counter to a microinstruction stored in RAM. The program flow continues in RAM until redirected back to the ROM. Since RAM is dynamic, the RAM can redirect the program counter, hence, the program flow, to any microinstruction stored in ROM.

For the purposes of debugging the program code, a programmer can identify a bug in a portion of the microinstruction subroutine in ROM and store the address of the bug in a jump point register. If the bug is a minor error that can be fixed with just a few lines of code, a patch for the bug can easily be stored in RAM. When the program counter encounters the address of the bug, which has been stored in the jump point register, the comparator allows a signal to be sent to the interrupt controller, whereby an interrupt is generated which redirects the program flow to the patch stored in RAM. When the microinstructions contained in the patch have finished executing, the RAM microinstruction immediately following the patch can redirect the program flow to the ROM microinstruction following the erroneous section of the subroutine. In this manner, a subroutine containing a small programming error can be corrected without having to duplicate the entire subroutine in RAM.

For the purpose of adding functionality to the program code stored in ROM, a programmer can advantageously utilize the jump point registers to add further subroutines within the structure of already existing ROM subroutines. In one embodiment of the invention, a programmer can insert a microinstruction address into a jump point register wherein the microinstruction is part of a program subroutine residing in ROM. When the program counter contains the address of this microinstruction, the program flow will jump to a corresponding set of microinstructions stored in RAM. The last instruction of the corresponding set of microinstructions will redirect the program flow back to the subroutine at whichever point the programmer desires. In this manner, a programmer can add a new function to the ROM subroutine without replacing the old, programmed ROM. Hence, a ROM can be updated without having to be replaced by a reprogrammed ROM.

The interrupt circuit of Fig. 3 is one embodiment of the invention where three (3) jump points and corresponding comparators are connected through a OR gate **330** to an interrupt request (IRQ) pin in a processor, i.e., the interrupt controller. In yet another embodiment of the invention, multiple interrupt circuits can be used in a data processing system in order to minimize the amount of checking for new code that will occur whenever an interrupt is triggered. When there is a large number of jump point registers within a single interrupt circuit, a large amount of MIPS is consumed to determine which new code corresponds to the interrupt that was just triggered. However, when there are numerous interrupt circuits, each with only three or fewer jump point registers, and each connected to an individual IRQ pin, less MIPS are consumed in implementing the code corresponding to the triggered interrupt event. The use of multiple interrupt circuits or a single interrupt circuit is a matter of design choice according to the needs of the circuit board designer. However, any variation of the interrupt circuit as described herein falls within the scope of this invention.

FIG. 4 is a block diagram showing a data processing system. It will be apparent to one skilled in the art that the present invention may be practiced without specific details as to well-known circuits and control logic. In order to avoid obscuring the description, such specific details have been omitted from FIG. 4. The block diagram of FIG. 4 is representative of a system wherein the control logic is segregated from the operation core. The system may be a digital signal processor or an application specific integrated circuit. However, it should be noted that the present invention can be used in data processing systems with other architectural forms, e.g., where the control logic is combined with the operation core.

A program flow control device **400** is coupled to the control store RAM **430**, the control store ROM **440**, an interrupt circuit **450**, and an instruction-decoding device **410**. The interrupt circuit **450** may be the interrupt circuit of FIG. 3. The program flow control device **400** generates the contents of the program counter, generates the flags which show whether the current instruction has been executed or canceled, and handles all external events such as direct memory access (DMA) and interrupts. The instruction-decoding device **410** may or may not be integrated within the operation core **420** and is connected to the program flow control device **400** through line **405**. The instruction-decoding device **410** is also connected to the control store RAM **430** and control store ROM **440** through line **404**. The interrupt circuit **450** is coupled to the program flow control device **400**, control store RAM **430**, control store ROM **440**, and the operation core **420**. The program flow control device **400** generates the program counter based upon input from control store RAM **430**, control store ROM **440** or the interrupt circuit **450**. Lines RAM_CS **401**, ROM_CS **403**, and EXEC **412** are used by the program flow control device **400** to enable input from the control store RAM **430**, control store ROM **440**, or the interrupt circuit **450**. Line **422** loads jump point addresses into the jump point registers (not shown) in the interrupt circuit **450**. Line **402** carries the contents of the program counter to the control store RAM **430**, control store ROM **440** and the interrupt circuit **450**. When interrupt circuit **450** indicates that the current program counter contains an address equal to a jump point address contained in a jump point register (not shown), an interrupt is generated by an interrupt controller (not shown), which may or may not be integrated into the program flow control device **400**. When the interrupt circuit **450** generates an interrupt, the program flow control device **400** resets the program counter to hold the address of the next microinstruction specified by the interrupt event.

The data processing system of FIG. 4 is just one illustrative example of how an embodiment of the present invention may be used. It should be noted that the present invention may be realized using a variety of computer programming languages and hardware, and is not limited to any particular hardware and software configuration. For example, the functions of program flow control device **400**, the instruction decoder **410**, and the operation core **420** can be achieved through the use of a general-purpose processor, as illustrated in block **490**. The present invention may be utilized in any embodiment which has code stored in a static storage device such as a ROM, a magnetic tape storage unit, a compact disk or a floppy disk.

As will be realized, the invention is capable of other and different embodiments and its several details are capable of modifications in various respects, all without departing from the invention. Accordingly, the drawings and description are to be regarded as illustrative in nature, and not as restrictive.

WE CLAIM:

CLAIMS

1. A method for modifying the program flow in a static memory device, the method comprising the step of generating an interrupt that triggers a jump from the static memory device to a programmable memory device.

2. The method of Claim 1, wherein the step of generating the interrupt comprises the steps of:

storing a copy of an address of a first microinstruction, wherein the first microinstruction is a portion of a first set of microinstructions, and the first set of microinstructions is a subset of a subroutine residing in the static memory device;

storing a replacement set of microinstructions in the programmable memory device;

comparing the address of each microinstruction in the subroutine with the stored copy during the program flow; and

substituting the first set of microinstructions with the replacement set of microinstructions when the comparing step results in a match, wherein the substituting step produces a jump in the program flow from the static memory device to the programmable memory device.

3. The method of Claim 2, wherein the static memory device comprises a Read-Only Memory (ROM) device and the programmable memory device comprises a Random Access Memory (RAM) device.

4. The method of Claim 3, wherein the replacement set of microinstructions comprises a patch for a programming error present in the subroutine.

5. The method of Claim 1, wherein the step of generating an interrupt comprises the steps of:

storing a copy of an address of a first microinstruction, wherein the first microinstruction is a portion of a first set of microinstructions embedded in a first subroutine, the first subroutine residing in the static memory device;

storing an additional set of microinstructions in a programmable memory device;

comparing the address of each microinstruction in the subroutine with the stored copy during the program flow; and

adding the additional set of microinstructions when the comparing step results in a match, wherein the adding step produces a jump in the program flow from the static memory device to the programmable memory device.

6. The method of Claim 5, wherein the additional set of microinstructions is a second subroutine with an additional functionality.

7. The method of Claim 1, wherein the step of generating an interrupt comprises the steps of:

using a jump point register to hold a jump point address, the jump point address triggering an interrupt event;

executing a first set of microinstructions in a sequential order, wherein the first set of microinstructions resides in the static memory device, wherein each sequentially executed microinstruction in the first set of microinstructions has a corresponding address that is individually stored in a program counter during the sequential order of execution; and

interrupting the program flow with the interrupt event if the program counter holds an address that equals the jump point address in the jump point register.

8. The method of Claim 7 wherein the step of interrupting the program flow comprises the steps of:

setting a control signal to conditionally enable a comparator, wherein the comparator compares the address in the program counter with the address contained in the jump point register; and

executing a second set of microinstructions corresponding to the jump point address if the comparator receives an enable control signal, wherein the second set of microinstructions reside in a programmable memory device.

9. The method of Claim 8 wherein the programmable memory device is a Random Access Memory (RAM) device.

10. The method of Claim 1 wherein the step of generating an interrupt further comprises the steps of:

storing a plurality of jump point addresses, each jump point address stored in a corresponding jump point register, each jump point address triggering a corresponding interrupt event;

predetermining which jump point address of the plurality of jump point addresses is to be enabled;

executing a first set of microinstructions in a sequential order, said first set of microinstructions reside in the static memory device, wherein a program counter sequentially holds an address for each microinstruction being executed; and

implementing the corresponding interrupt event if the program counter holds an address that equals one of the plurality of jump point addresses and if the jump point address is enabled.

11. The method of Claim 10 wherein the step of implementing the interrupt event comprises the steps of:

predetermining a plurality of microinstruction sets corresponding to each of the plurality of jump point addresses;

setting a control signal to conditionally enable each member of a plurality of comparators, wherein each comparator is coupled to a corresponding jump point register and the program counter; and

executing a corresponding set of microinstructions associated with the jump point address if one of the plurality of comparators receives an enable control signal, wherein the jump point address is equal to the address in the program counter, wherein the corresponding set of microinstructions reside in a programmable memory device.

12. The method of Claim 11 wherein the step of executing the corresponding set of microinstructions is followed by the step of executing a subsequent microinstruction from the first set of microinstructions.

13. Apparatus for processing programmable machine instruction signals, comprising:

a jump point register containing at least one predetermined jump point;

a static memory device containing a first microinstruction set, the static memory device coupled to the jump point register;

a random access memory device containing a second microinstruction set, the random access memory device coupled to said jump point register and static memory device;

a program control unit coupled to the jump point register, the static memory device and the random access memory device, wherein the program control unit sends a plurality of control signals to the jump point register, the static memory device and the random access memory device; and

a comparator coupled to the jump point register and the program control unit, wherein the comparator makes a comparison between the predetermined jump point with at least one of the plurality of control signals and generates an interrupt signal according to the comparison.

14. The apparatus of Claim 13, further comprising:

a plurality of jump point registers, each of said plurality of jump point registers storing at least one predetermined jump point; wherein the program control unit sends a plurality of control signals to each of the plurality of jump point registers, the static memory device and the random access memory device; and

a plurality of comparators, each comparator coupled to one of the plurality of jump point registers and the program control unit, wherein each comparator makes a comparison between the corresponding jump point address and at least one of the plurality of control signals, and generates an interrupt signal according to the comparison.

15. Apparatus for modifying the program flow of microinstructions residing in a static memory device, comprising:

means for storing a predetermined microinstruction address apart from the static memory device;

means for storing a first set of microinstruction addresses, wherein said storage means is volatile;

means for comparing the predetermined microinstruction address with each address of the first set of microinstruction addresses; and means for generating a plurality of control signals, the means coupled to the predetermined microinstruction address storage means, the volatile storage means, and the comparison means, wherein the control signal means enable a modification of the program flow from the static memory device to the volatile storage means.

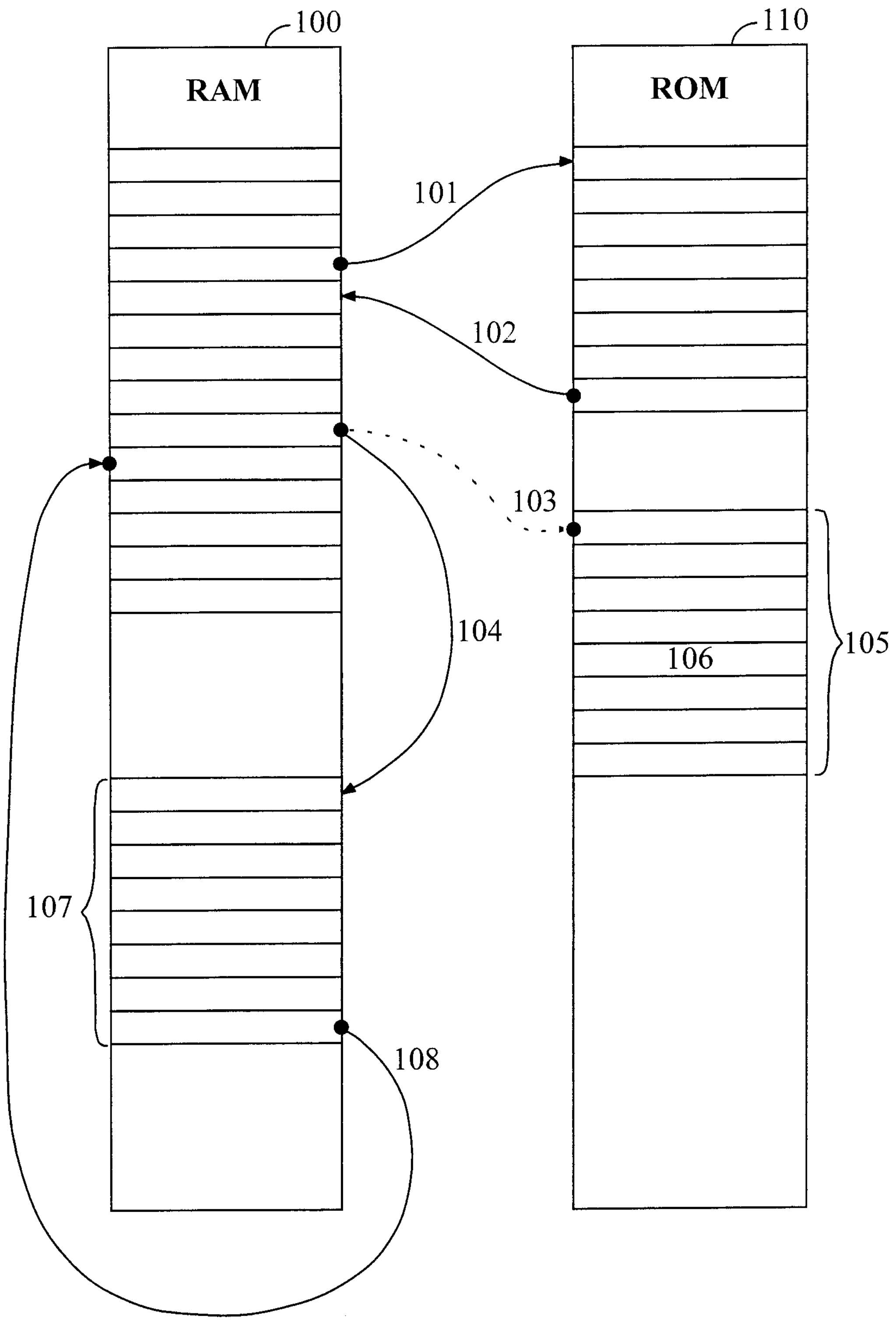


FIG. 1
(PRIOR ART)

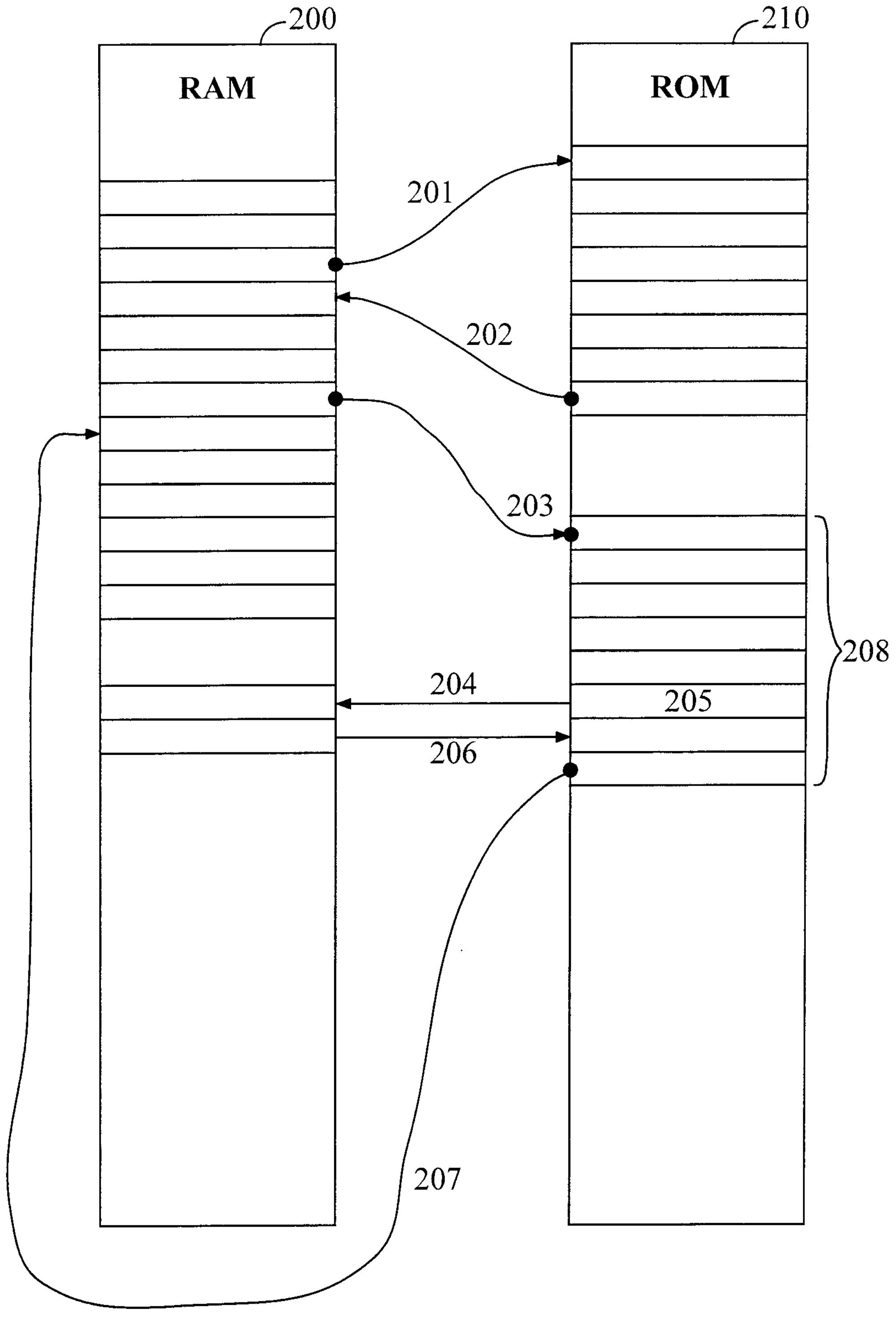


FIG. 2

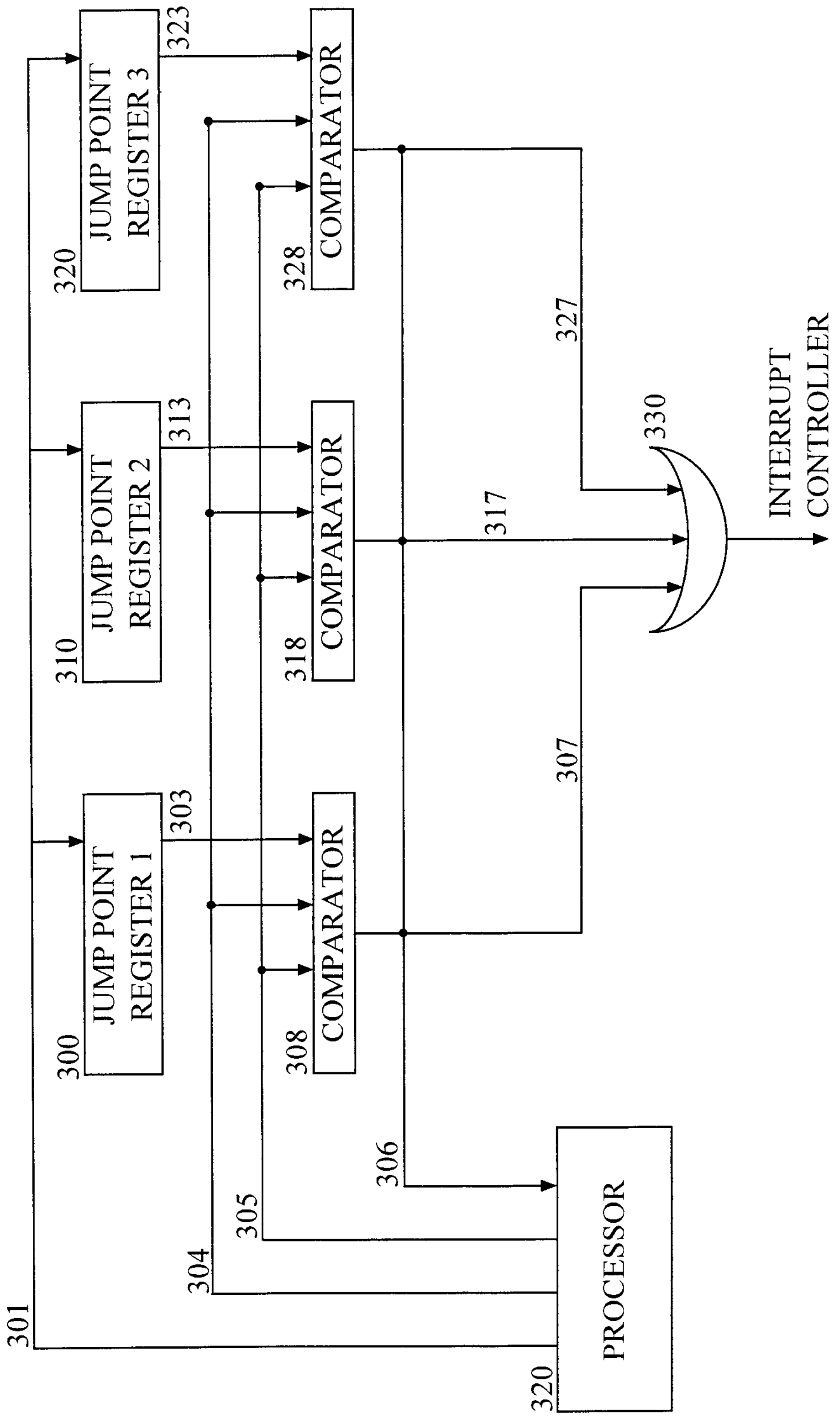

JUMP POINT REGISTER 3
320
323
COMPARATOR
328
JUMP POINT REGISTER 2
310
313
COMPARATOR
318
JUMP POINT REGISTER 1
300
303
COMPARATOR
308
327
330
317
307
INTERRUPT CONTROLLER
306
305
304
301
PROCESSOR
320

FIG. 3

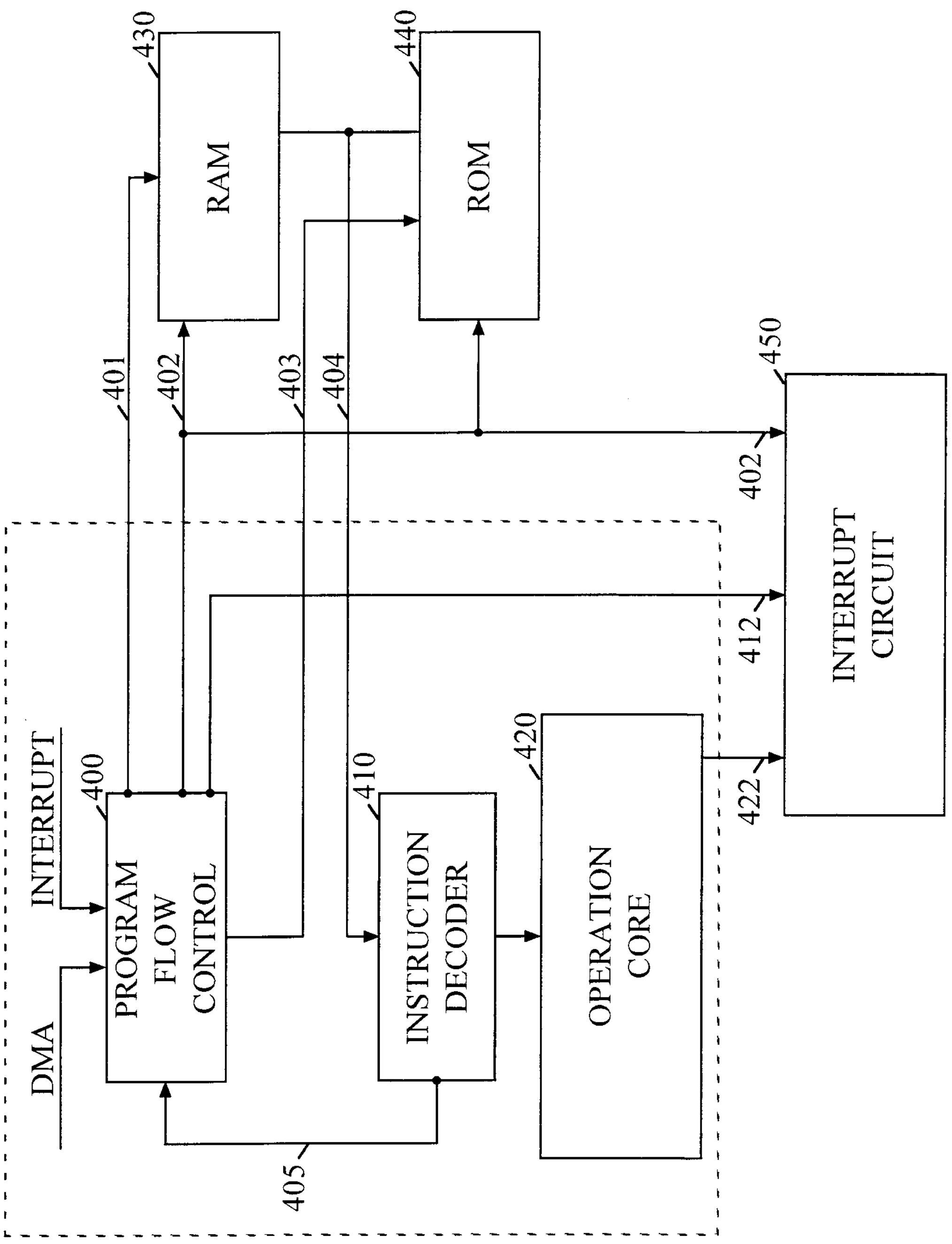

RAM
430
ROM
440
401
402
403
404
450
402
INTERRUPT CIRCUIT
412
422
INTERRUPT
DMA
400
PROGRAM FLOW CONTROL
410
INSTRUCTION DECODER
420
OPERATION CORE
405

FIG. 4

INTERNATIONAL SEARCH REPORT

Int' ational Application No
PC I/US 00/25474

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F11/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 901 225 A (IRETON MARK A ET AL) 4 May 1999 (1999-05-04) abstract column 2, line 54 -column 3, line 17 column 3, line 43 -column 3, line 64 column 4, line 38 -column 5, line 10 column 2, line 13 - line 27 column 7, line 39 -column 8, line 45 column 9, line 40 - line 54 column 10, line 3 - line 18 figures 1,2,6 ---	1-15

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search 8 February 2001	Date of mailing of the international search report 15/02/2001
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Authorized officer Leuridan, K

3

INTERNATIONAL SEARCH REPORT

Int tional Application No
PCT/US 00/25474

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	US 5 592 613 A (MIYAZAWA ET AL) 7 January 1997 (1997-01-07) abstract column 1, line 48 -column 2, line 22 column 3, line 26 - line 67 column 4, line 37 - line 60 column 6, line 9 -column 8, line 5 column 9, line 7 - line 65 column 11, line 46 -column 12, line 5 figures 6,12,15 ---	1-15
X	US 5 784 537 A (SUZUKI ET AL) 21 July 1998 (1998-07-21) abstract column 1, line 45 -column 2, line 56 column 3, line 59 -column 4, line 26 column 4, line 40 - line 49 column 5, line 41 - line 61 column 6, line 50 - line 64 figures 1,2 ---	1-4, 7-10,13, 15
X	US 5 051 897 A (HAYASHI KAZUO ET AL) 24 September 1991 (1991-09-24) the whole document -----	1,2,7,8

3

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int ıtional Application No
PCT/US 00/25474

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5901225 A	04-05-1999	WO 9825205 A	11-06-1998
US 5592613 A	07-01-1997	JP 3186927 A	14-08-1991
		US 5357627 A	18-10-1994
		JP 3033926 A	14-02-1991
US 5784537 A	21-07-1998	JP 8166877 A	25-06-1996
US 5051897 A	24-09-1991	JP 1232447 A	18-09-1989
		DE 3900187 A	28-09-1989
		KR 9300096 B	08-01-1993