US 20050246506A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2005/0246506 A1

Ukai (43) Pub. Date: Nov. 3, 2005

(54) INFORMATION PROCESSING DEVICE, PROCESSOR, PROCESSOR CONTROL METHOD, INFORMATION PROCESSING DEVICE CONTROL METHOD AND CACHE MEMORY

(75) Inventor: **Masaki Ukai**, Kawasaki (JP)

Correspondence Address:
STAAS & HALSEY LLP
SUITE 700
1201 NEW YORK AVENUE, N.W.
WASHINGTON, DC 20005 (US)

(73) Assignee: **FUJITSU LIMITED**, KAWASAKI (JP)

(21) Appl. No.: **10/937,253**

(22) Filed: **Sep. 10, 2004**

(30) **Foreign Application Priority Data**

Apr. 30, 2004 (JP) 2004-135875

Publication Classification

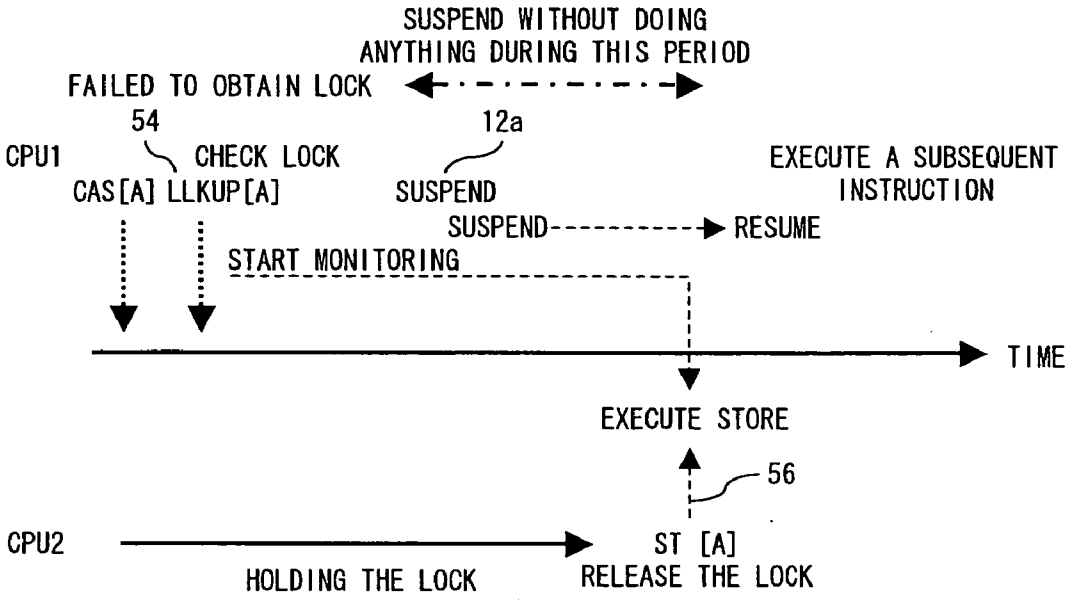
(51) **Int. Cl.**[7] **G06F 12/00**

(52) **U.S. Cl.** **711/152**

(57) **ABSTRACT**

In a method for controlling a processor which accesses information of a storage device through cache memory, when reading information stored in a target address or an address range of the storage device, it is monitored whether there is an update access to the address or address range from another processor, and also the processor is entered into a suspense status, which is released using the occurrence of the update access to the storage device from another processor as a trigger.

IN THE CASE OF THE PRESENT INVENTION



```
                                    // COMMENTS
try:
     mov   [ID],    %l0             // MOVE THE MEMORY CONTENTS OF [ID] TO A REGISTER
                                       %l0.
     cas   [lock], %g0,  %l0        // COMPARE THE MEMORY CONTENTS OF [LOCK] WITH THE
                                       VALUE OF A ZERO REGISTER %0.
                                    // IF THEY COINCIDE WITH EACH OTHER, [LOCK] AND THE
                                       VALUE OF %l0 ARE EXCHANGED.
     tst   %l0                      // CHECK WHETHER THE VALUE OF THE REGISTER %l0 IS 0.
     bne   rty                      // IF THE CHECK RESULT OF TST IS NOT 0, THE
                                       EXECUTION POINT IS BRANCHED TO TRY:. HOWEVER,
                                    // EXECUTE AN INSTRUCTION IMMEDIATELY AFTER (DELAY
                                       INSTRUCTION PROCESS) REGARDLESS OF THE
                                       EXISTENCE/NON-EXISTENCE OF A BRANCH.
       nop                          // AN INSTRUCTION TO DO NOTHING. DUE TO THE DELAY
                                       INSTRUCTION PROCESS OF BNE ABOVE
out:
     - - -  ! Lock held             //SUCCEEDED IN LOCKING
```

FIG. 1

```
loop:
        ld    [lock], %l0       // LOAD THE MEMORY CONTENTS OF [LOCK] ONTO THE
                                   ZERO REGISTER %10.
        tst   %l0
        bne   loop
        nop
try:
        mov   [ID],   %l0
        cas   [lock], %g0, %l0
        tst   %l0
        bne   loop
        nop
out:
        . . .         ! Lock held              //SUCCEEDED IN LOCKING
```

FIG. 2

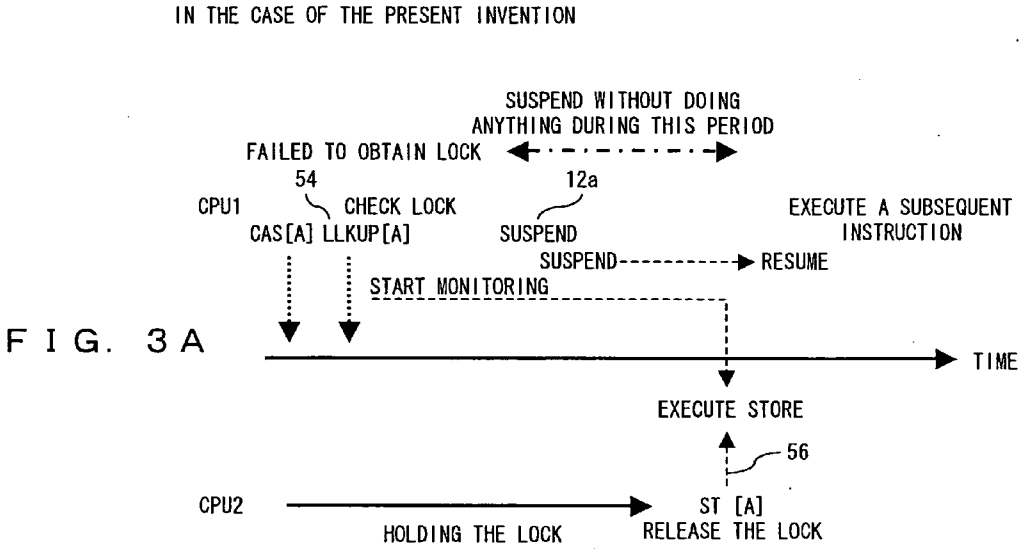

IN THE CASE OF THE PRESENT INVENTION
SUSPEND WITHOUT DOING ANYTHING DURING THIS PERIOD
FAILED TO OBTAIN LOCK
54
12a
CPU1
CHECK LOCK
CAS[A] LLKUP[A]
SUSPEND
EXECUTE A SUBSEQUENT INSTRUCTION
SUSPEND
RESUME
START MONITORING
TIME
EXECUTE STORE
56
CPU2
ST [A]
HOLDING THE LOCK
RELEASE THE LOCK

FIG. 3A

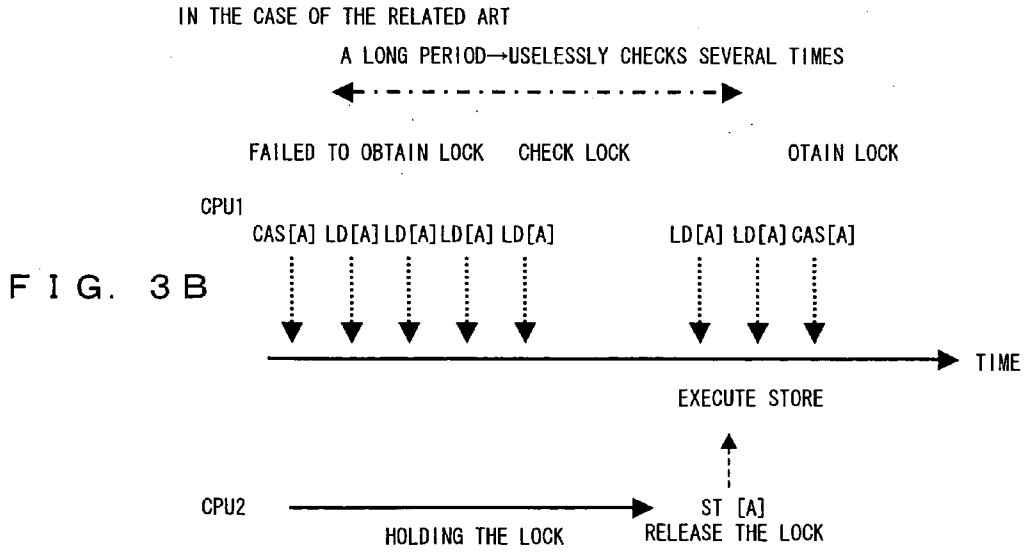

IN THE CASE OF THE RELATED ART
A LONG PERIOD→USELESSLY CHECKS SEVERAL TIMES
FAILED TO OBTAIN LOCK
CHECK LOCK
OTAIN LOCK
CPU1
CAS[A] LD[A] LD[A] LD[A] LD[A]
LD[A] LD[A] CAS[A]
TIME
EXECUTE STORE
CPU2
ST [A]
HOLDING THE LOCK
RELEASE THE LOCK

FIG. 3B

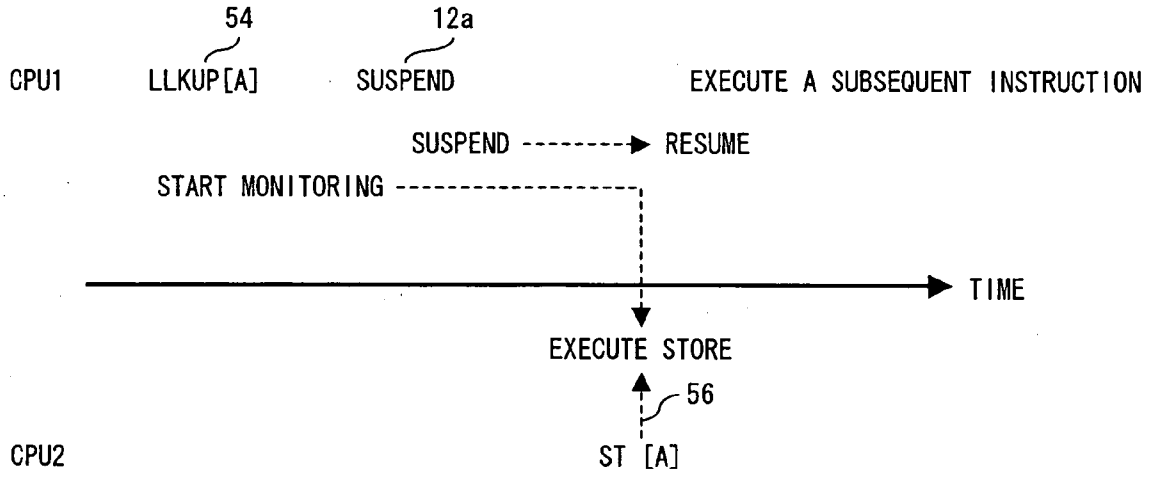

54
12a
CPU1
LLKUP[A]
SUSPEND
EXECUTE A SUBSEQUENT INSTRUCTION
SUSPEND
RESUME
START MONITORING
TIME
EXECUTE STORE
56
CPU2
ST [A]

FIG. 4

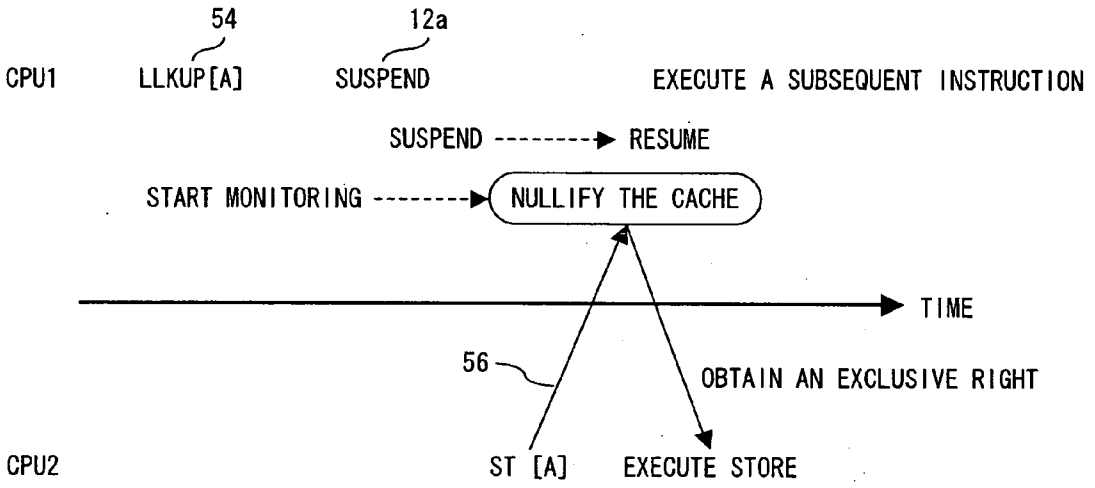

54
12a
CPU1
LLKUP[A]
SUSPEND
EXECUTE A SUBSEQUENT INSTRUCTION
SUSPEND
RESUME
START MONITORING
NULLIFY THE CACHE
TIME
56
OBTAIN AN EXCLUSIVE RIGHT
CPU2
ST [A]
EXECUTE STORE

FIG. 5

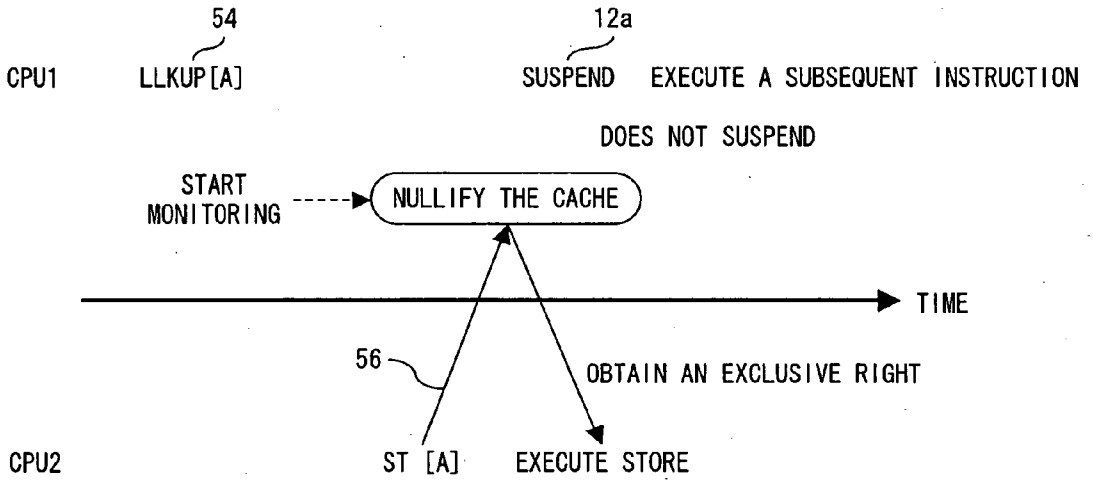

54
12a
CPU1
LLKUP[A]
SUSPEND
EXECUTE A SUBSEQUENT INSTRUCTION
DOES NOT SUSPEND
START MONITORING
NULLIFY THE CACHE
TIME
56
OBTAIN AN EXCLUSIVE RIGHT
CPU2
ST [A]
EXECUTE STORE

FIG. 6

```
ld    [lock], %l0
ld    [lock], %l0
ld    [lock], %l0
```

FIG. 7

```
ld   [lock], %l0
mov  [ID],   %l0
cas  [lock], %g0, %l0        //%0 IS ALWAYS 0 REGISTER
ld   [lock], %l0
ld   [lock], %l0
ld   [lock], %l0
 . . .
```

FIG. 8

```
mov   [ID],   %l0
cas   [lock], %g0,  %l0
ld    [lock], %l0
ld    [lock], %l0
ld    [lock], %l0
 . . .
```

FIG. 9

```
cas   [lock], %g0, %l0
ld    [lock], %l0
```

FIG. 10

```
cas   [lock], %g0, %l0
cas   [lock], %g0, %l0
cas   [lock], %g0, %l0
 . . .
```

FIG. 11

```
loop:
     llkup   [lock], %l0
     tst     %l0
     bne,a   loop                  // IF BRANCH CONDITIONS ARE MET, "SUSPEND"
                                      IMMEDIATELY AFTER IS EXECUTED.
     suspend                       // IF THEY ARE NOT MET, "SUSPEND" IMMEDIATELY
                                      AFTER IS NOT EXECUTED.
try:
     mov     [ID],   %l0
     cas     [lock], %g0, %l0
     tst     %l0
     bne     loop
     nop
out:
     ...          ! Lock held              //SUCCEEDED IN LOCKING
```

FIG. 12

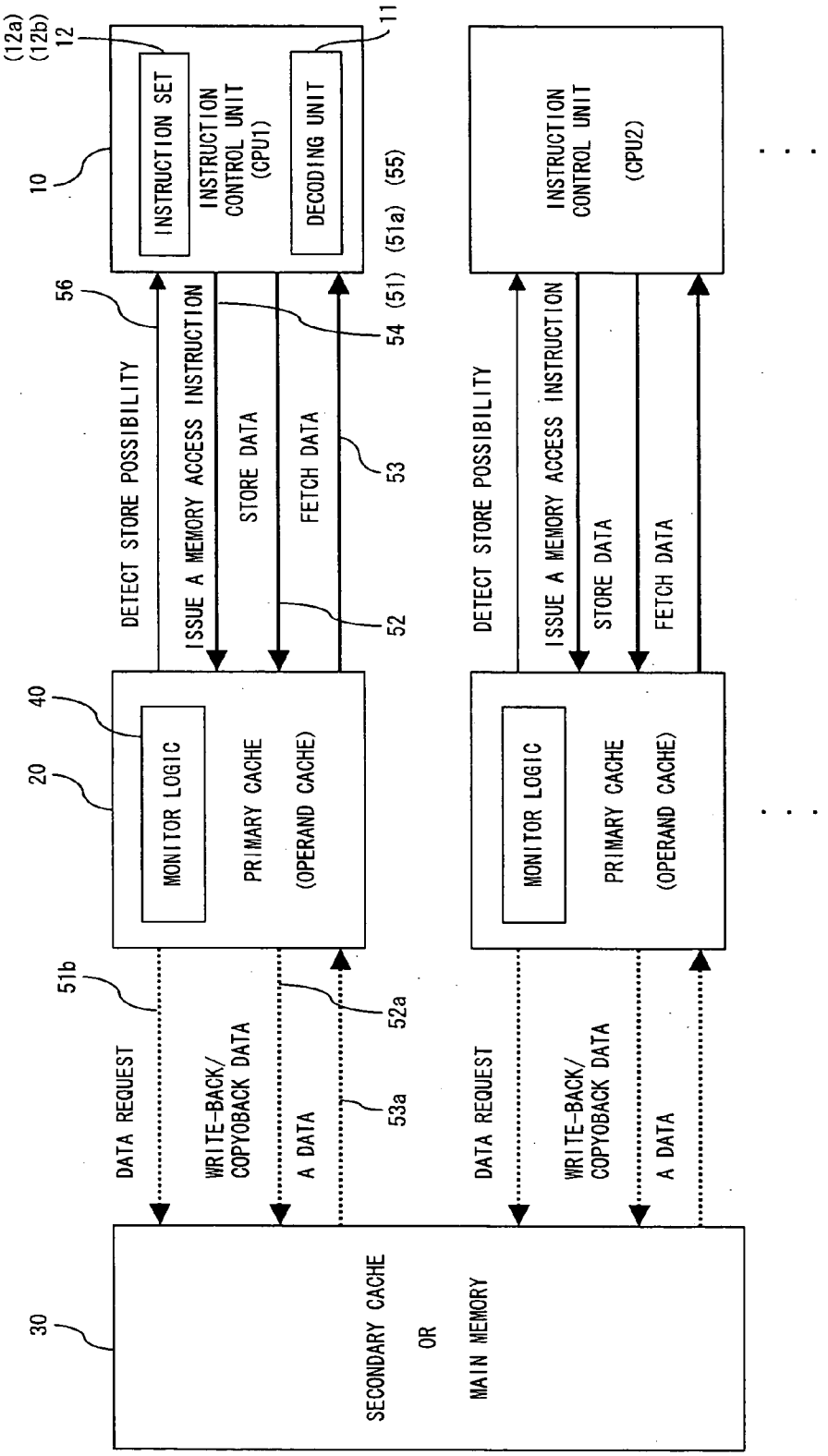

(12a)
(12b)
12
11
10
INSTRUCTION SET
INSTRUCTION CONTROL UNIT (CPU1)
DECODING UNIT
INSTRUCTION CONTROL UNIT (CPU2)
56
54 (51) (51a) (55)
DETECT STORE POSSIBILITY
ISSUE A MEMORY ACCESS INSTRUCTION
STORE DATA
FETCH DATA
53
52
40
20
MONITOR LOGIC
PRIMARY CACHE (OPERAND CACHE)
51b
52a
53a
DATA REQUEST
WRITE-BACK/ COPYBACK DATA
A DATA
30
SECONDARY CACHE OR MAIN MEMORY

FIG. 13

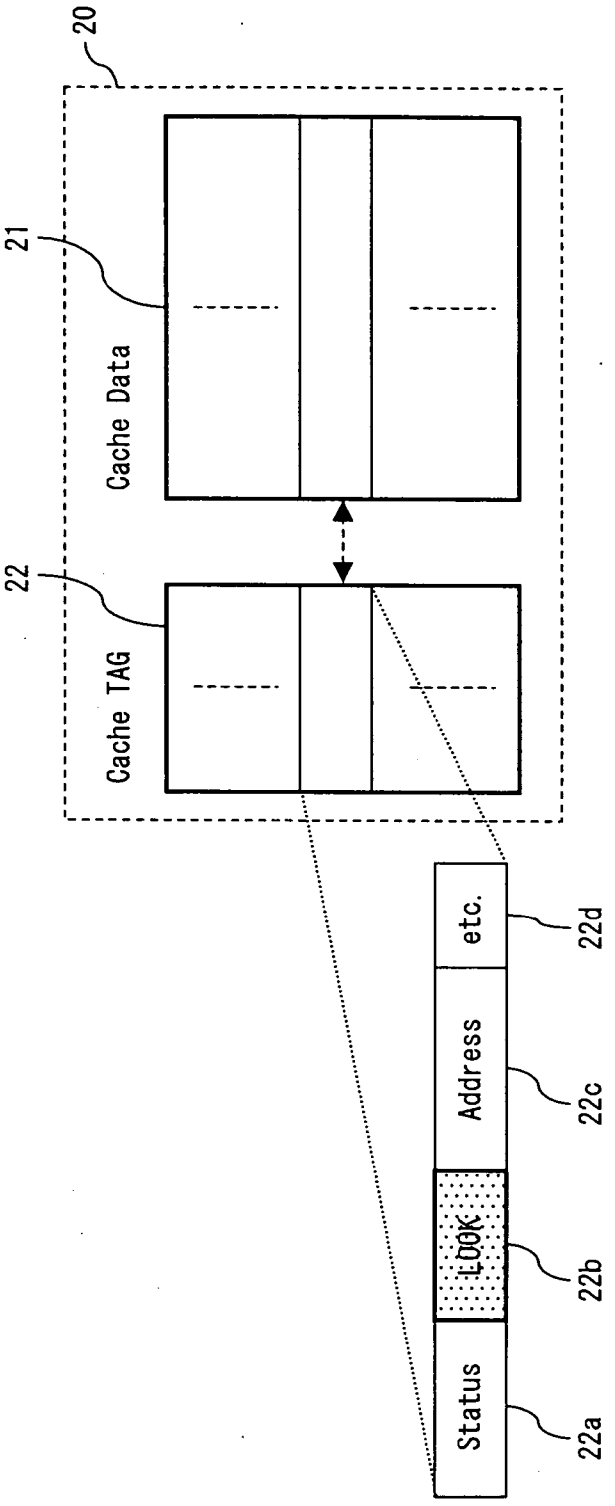

20
21
Cache Data
22
Cache TAG
Status
LOOK
Address
etc.
22a
22b
22c
22d

FIG. 14

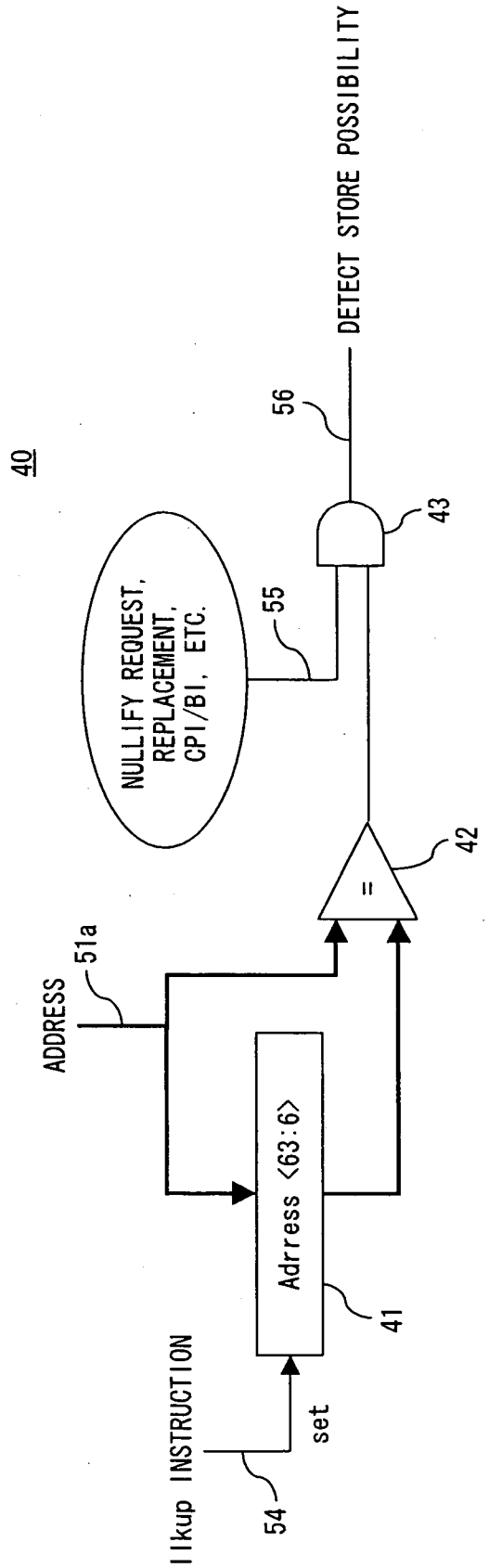

40
DETECT STORE POSSIBILITY
56
43
NULLIFY REQUEST,
REPLACEMENT,
CPI/BI, ETC.
55
=
42
ADDRESS
51a
Adrress <63:6>
41
Ilkup INSTRUCTION
54
set

FIG. 15

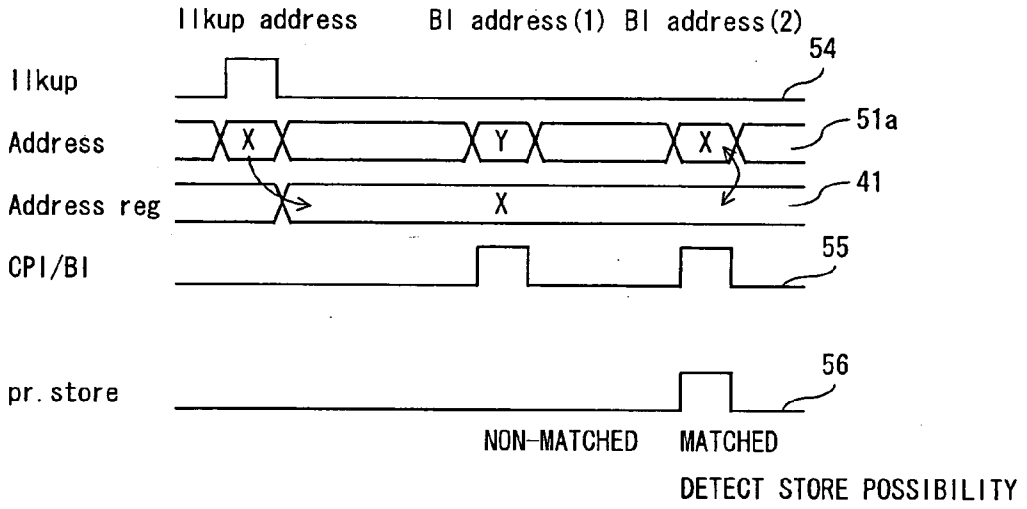

Ilkup address
BI address(1)
BI address(2)
54
Ilkup
Address
X
Y
X
51a
Address reg
X
41
CPI/BI
55
pr.store
56
NON-MATCHED
MATCHED
DETECT STORE POSSIBILITY

FIG. 16

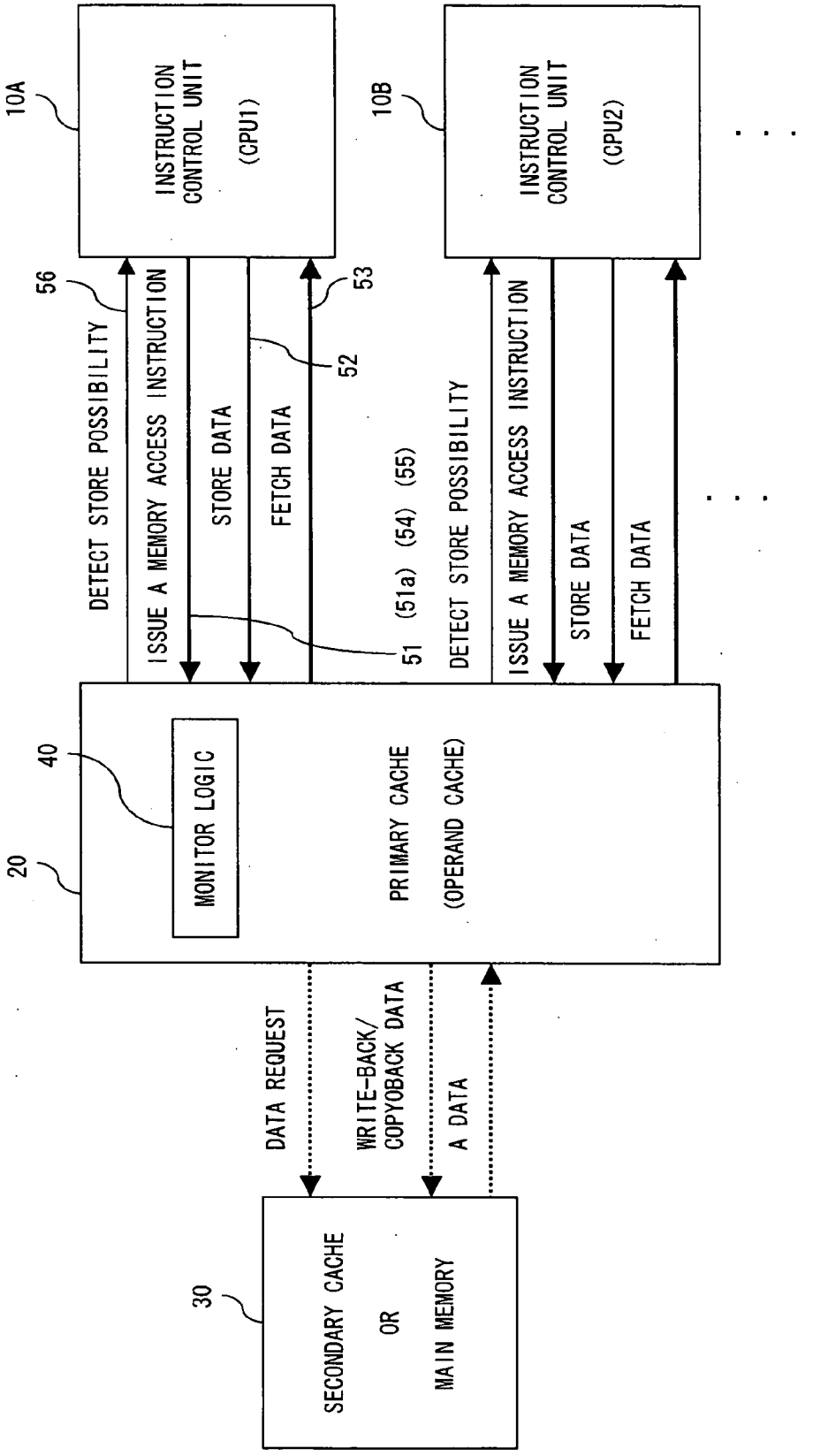

10A
INSTRUCTION CONTROL UNIT (CPU1)
10B
INSTRUCTION CONTROL UNIT (CPU2)
56
DETECT STORE POSSIBILITY
ISSUE A MEMORY ACCESS INSTRUCTION
STORE DATA
FETCH DATA
53
52
51 (51a) (54) (55)
DETECT STORE POSSIBILITY
ISSUE A MEMORY ACCESS INSTRUCTION
STORE DATA
FETCH DATA
40
MONITOR LOGIC
20
PRIMARY CACHE (OPERAND CACHE)
DATA REQUEST
WRITE-BACK/ COPYOBACK DATA
A DATA
30
SECONDARY CACHE OR MAIN MEMORY

FIG. 17

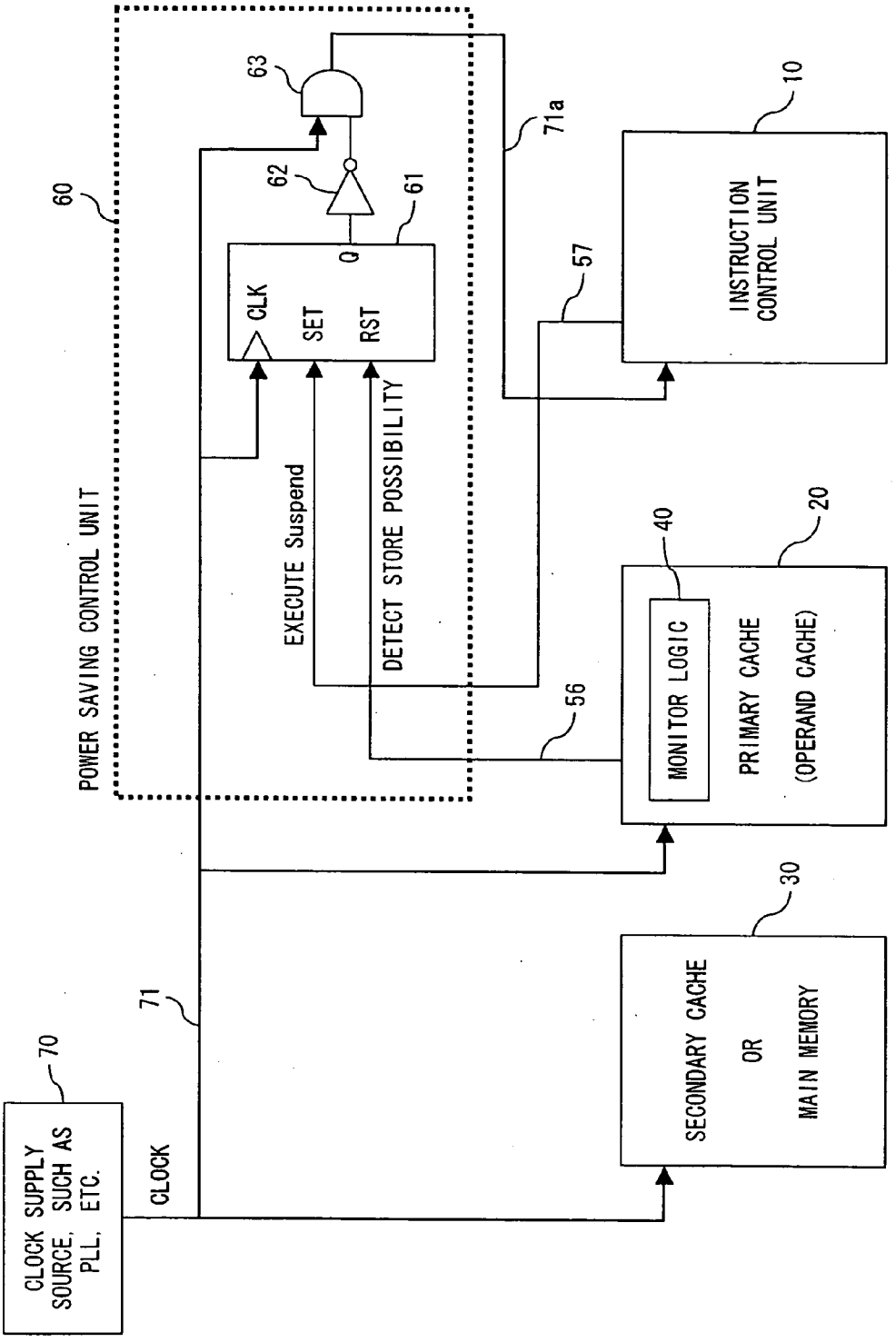

CLOCK SUPPLY SOURCE, SUCH AS PLL, ETC.
70
CLOCK
71
POWER SAVING CONTROL UNIT
60
63
62
61
CLK
SET
RST
Q
EXECUTE Suspend
DETECT STORE POSSIBILITY
71a
57
56
INSTRUCTION CONTROL UNIT
10
MONITOR LOGIC
40
PRIMARY CACHE
(OPERAND CACHE)
20
SECONDARY CACHE
OR
MAIN MEMORY
30

FIG. 18

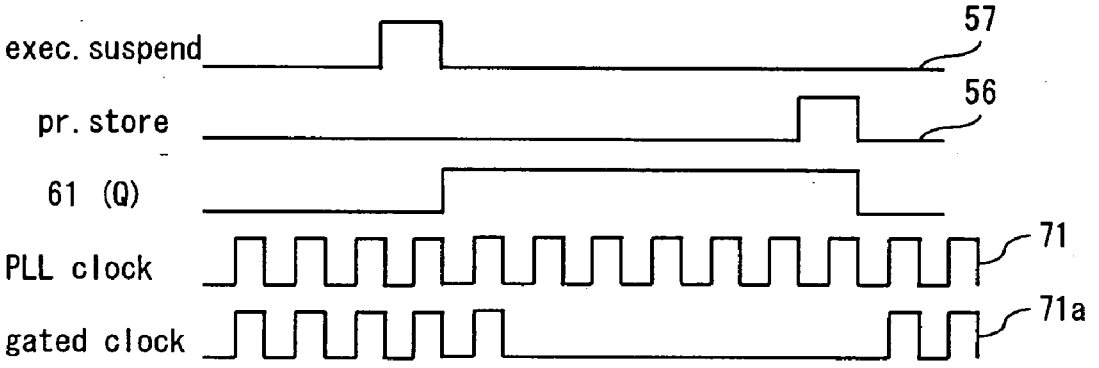


FIG. 19

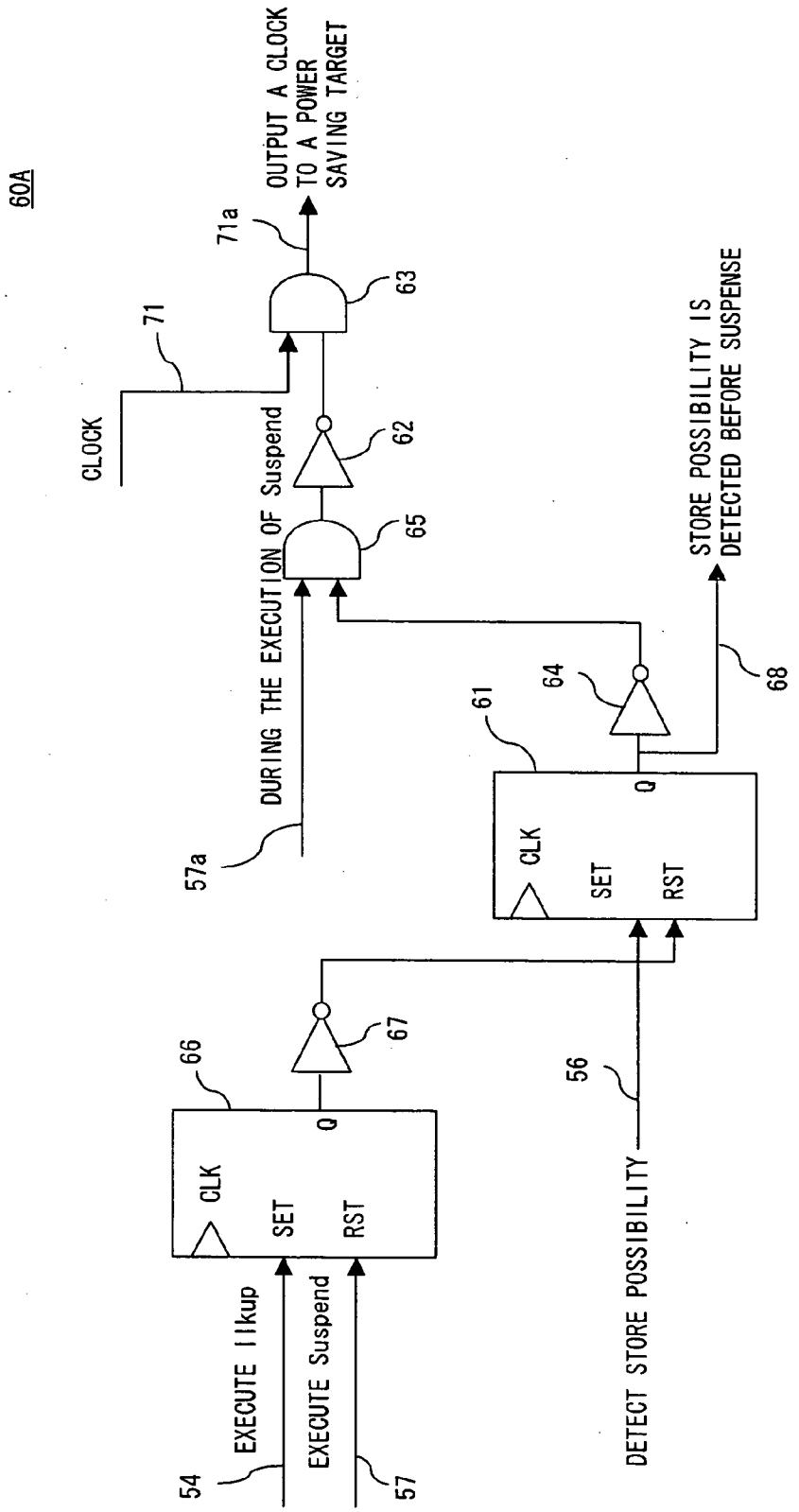

60A
OUTPUT A CLOCK TO A POWER SAVING TARGET
71a
63
71
CLOCK
62
DURING THE EXECUTION OF Suspend
65
STORE POSSIBILITY IS DETECTED BEFORE SUSPENSE
68
64
61
57a
CLK
SET
RST
Q
67
66
56
54
EXECUTE Ilkup
EXECUTE Suspend
57
DETECT STORE POSSIBILITY

FIG. 20

FIG. 21A

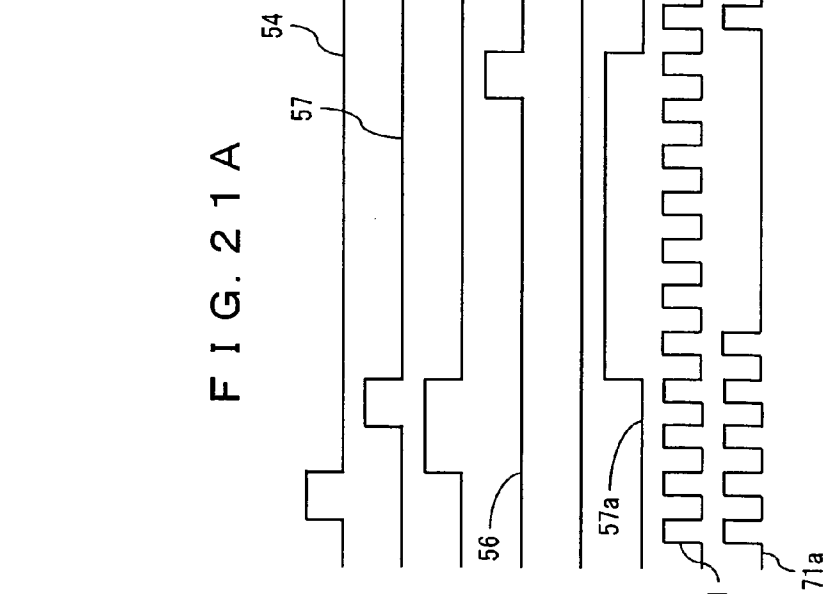
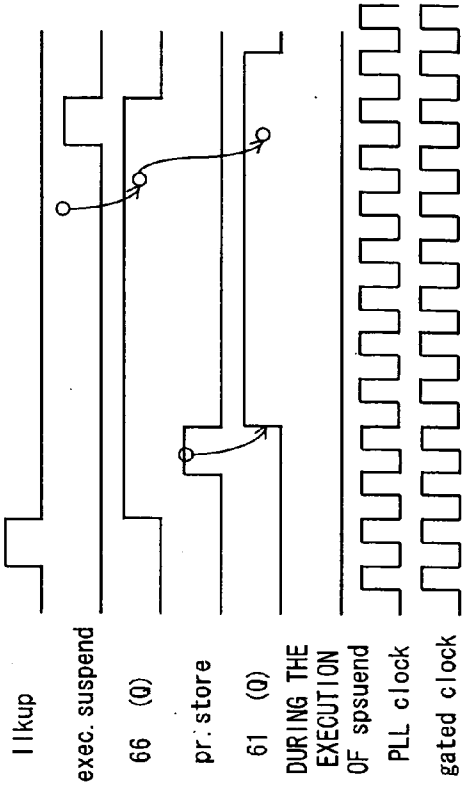


FIG. 21B



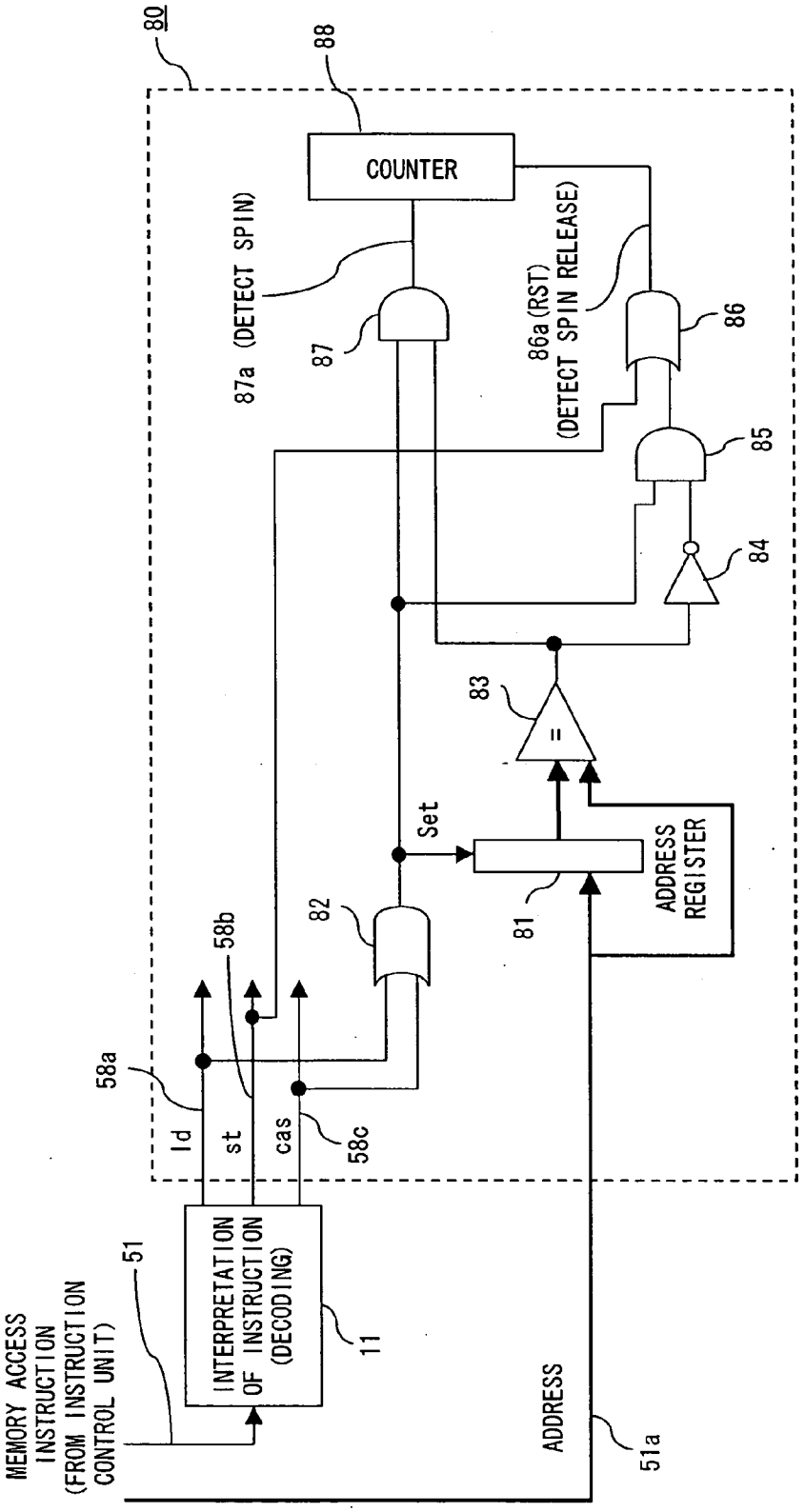

80
88
COUNTER
87a (DETECT SPIN)
87
86a (RST)
(DETECT SPIN RELEASE)
86
85
84
83
=
Set
82
81
ADDRESS REGISTER
58b
58a
ld
st
cas
58c
INTERPRETATION OF INSTRUCTION (DECODING)
11
51
MEMORY ACCESS INSTRUCTION (FROM INSTRUCTION CONTROL UNIT)
ADDRESS
51a

FIG. 22

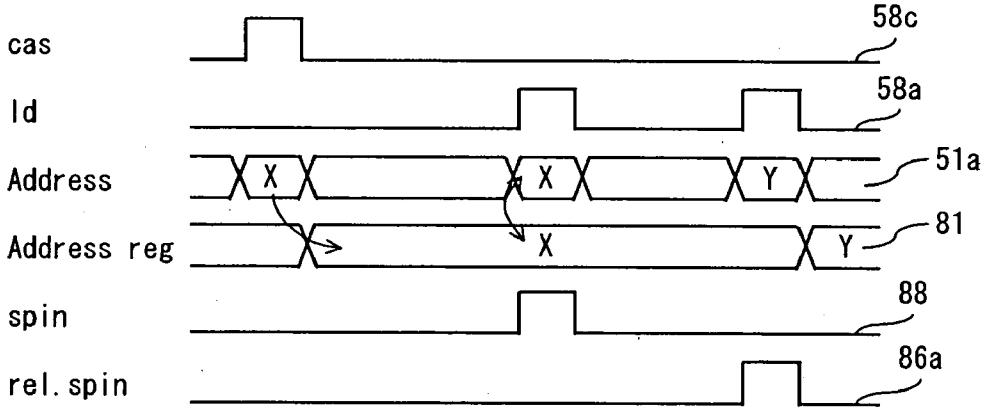

cas
ld
Address
Address reg
spin
rel.spin
X
X
Y
X
Y
58c
58a
51a
81
88
86a

FIG. 23

INFORMATION PROCESSING DEVICE, PROCESSOR, PROCESSOR CONTROL METHOD, INFORMATION PROCESSING DEVICE CONTROL METHOD AND CACHE MEMORY

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] The present invention relates to an information processing technology, and more particularly relates to the instruction processing technology of a system with shared memory, such as a symmetrical multi-processor (SMP) in which a plurality of processors are combined, cc-non-uniform memory architecture (cc-NUMA) and the like.

[0003] 2. Description of the Related Art

[0004] In a shared memory system by a multiprocessor, as a method for securing an exclusive right, a lock method, mutex-lock and the like are known. A spin loop is one of the typical methods for obtaining lock. In this case, a "lock variable" is provided for main memory, and each processor repeats the reference/update trial of the "lock variable" and spin loop (no-load running waiting) in order to secure lock. If lock is obtained, the lock is displayed only during lock. If the lock is released, lock release is displayed. Thus, an exclusive right can be secured among a plurality of processors.

[0005] An example of a conventional spin loop instruction string is described below. An address block (4 bytes) indicated by the "lock" of the main memory corresponds to the above-mentioned "lock variable". If the contents are 0, it is indicated that the process is in a lock-release status. If the contents are its own ID (contents of ID to which each process belongs), it is indicated that the process is being locked.

[0006] Although in the following example, description is made using a SPARC V9 instruction set, the description is not peculiarly applied to a specific instruction set, but is common to all instruction sets.

[0007] As the simplest spin loop, the instruction string shown in **FIG. 1** can be used.

[0008] In **FIG. 1**, the function of each instruction is described on its right side as comments. In an instruction string structured as shown in **FIG. 1**, since in a cas instruction, there is always a possibility that an access to the main memory of another processor (store) may occur, the processor always operates to obtain a cache block exclusive right. In this case, a plurality of processors tries to obtain the same lock variable and competition among a plurality of segments of cache becomes tough, which is a problem.

[0009] **FIG. 2** shows an example of an instruction string by which this problem is solved. In **FIG. 2**, the cache is kept in a shared status unless the lock variable is rewritten (as long as a processor that obtained lock maintains the lock). Therefore, there is no above-mentioned cache competition.

[0010] In such a configuration, the loop must be always executed and checked. However, since the recent high-speed tendency of processors is faster than the high-speed tendency of memory system, the difference in speed between a processor and a memory system is getting large.

[0011] In such a situation, although instruction strings are interpreted and executed by a great number of idle running by a spin loop, substantially no job is made and power is uselessly consumed, which is a problem. More particularly, in a large-scale SMP system, a specific lock variable is often collectively scrambled for. In this case, equipment other than a specific CPU does no useful job, and the power cost of system operation increases, which is a problem.

[0012] In a processor core adopting a multi-thread processing method, if this spin loop occurs in a specific thread processing part, idle running due to a spin loop process with no substantial job hinders the progress of other thread processes of the processor core, which is also a problem.

[0013] The same problem occurs in other processes using a lock variable, such as barrier synchronization, a processor synchronization process (synchronization waiting), general processor synchronization, I/O synchronization, an idle loop and the like.

[0014] As conventional exclusive control and synchronous control technologies in a multi-processor system, Patent References 1, 2 and 3 are known.

[0015] Specifically, in Patent Reference 1, a mechanism for realizing exclusive control by storing a shared variable in main memory and collectively monitoring processors on the main memory, is disclosed. In a recent processor with cache memory, rewriting in the cache is not promptly reflected on the main memory. More particularly, in a write-back cache method, it usually takes a fairly long time to reflect rewriting. Since in a recent processor with write-through cache, memory latency is very short and reflection loss is long. Accordingly, performance degrades.

[0016] Therefore, as in Patent Reference 1, the above-mentioned spin loop problems cannot be solved simply by collectively monitoring a plurality of segments of main memory. Therefore, a method for solving them within cache memory, which does not affect memory latency, is desired.

[0017] In Patent Reference 2, a technology for realizing the exclusive control of shared memory among CPUs by providing an access control signal wire (pin) for the exclusive control among CPUs in addition to a system bus shared by a plurality of the CPUs, is disclosed. Recently, since connection between processors (for example, the number of input/output pins of an LSI) has been costly, the use of one pin as a data line is more effective in performance improvement than the exclusive use of one pin for the purpose of exclusive control. Otherwise, the deletion of even one pin can contribute to the reduction of CPU manufacturing cost more. Therefore, a method for realizing exclusive control among CPUs without increasing the number of pins, is demanded.

[0018] In Patent Reference 3, a synchronous control circuit used to control synchronization between a processor and a co-processor, which are in the relationship between a master and a servant, is disclosed. However, it is difficult to apply the circuit to a system in which each processor equally handles shared memory.

[0019] Specifically, a processor can voluntarily catch the operation status of a co-processor since the processor is in a position to issue instructions to the co-processor. However, since in an SMP system, each processor does not logically

store information about the operation statuses of other processors, it is difficult to apply the technology of Patent Reference 3 in order to solve the above-mentioned problems.

[0020] The present invention is made to solve such problems. The present invention can also be applied to a co-processor system.

[0021] Patent Reference 1: Japanese Patent Laid-open Application No. 3-164964

[0022] Patent Reference 2: Japanese Patent Laid-open Application No. 61-229150

[0023] Patent Reference 3: Japanese Patent Laid-open Application No. 2002-41489

SUMMARY OF THE INVENTION

[0024] It is an object of the present invention to reduce the waste of power consumption and processor resources due to a spin loop for exclusive control among a plurality of logical and physical processors.

[0025] It is another object of the present invention to prevent the performance degradation of other logical processors due to the spin loop of one logical processor in a multi-processor system where one physical processor is provided with a plurality of hardware threads and which can be operated as if there were a plurality of logical processors.

[0026] It is another object of the present invention is to realize an exclusive control mechanism in which the rewriting from the existing program can be easily made and to which software can be ported with a low cost.

[0027] In the present invention, the above-mentioned problems can be solved by predicting the possibility of the rewriting of a lock variable for the exclusive control of memory access and suspending the processor or thread located in a part where conventionally a spin loop occurs due to release waiting.

[0028] Specifically, in the present invention, a processor can be suspended and resumed by providing both a new load instruction to set the monitor start trigger of a memory block, including a memory block to be loaded (hereinafter called "LOAD-WITH-LOOKUP instruction") and a writing detection function to monitor a memory block, and by executing/releasing a suspense instruction, such as a SUSPEND instruction and the like, in accordance with both the LOAD-WITH-LOOKUP instruction and the detection result of the writing detection function, in order to realize the possibility prediction of the rewriting of a lock variable.

BRIEF DESCRIPTION OF THE DRAWINGS

[0029] **FIG. 1** shows an instruction string in the conventional exclusive control;

[0030] **FIG. 2** shows an instruction string in the conventional exclusive control;

[0031] **FIGS. 3A and 3B** compare the control method of the information processing device with the conventional one;

[0032] **FIG. 4** shows an example of the function of the information processing device, which is one preferred embodiment of the present invention;

[0033] **FIG. 5** shows an example of the function of the information processing device, which is one preferred embodiment of the present invention;

[0034] **FIG. 6** shows an example of the function of the information processing device, which is one preferred embodiment of the present invention;

[0035] **FIG. 7** shows an example of an instruction string to be handled in the control method of the information processing device of the present invention;

[0036] **FIG. 8** shows an example of an instruction string to be handled in the control method of the information processing device of the present invention;

[0037] **FIG. 9** shows an example of an instruction string to be handled in the control method of the information processing device of the present invention;

[0038] **FIG. 10** shows an example of an instruction string to be handled in the control method of the information processing device of the present invention;

[0039] **FIG. 11** shows an example of an instruction string to be handled in the control method of the information processing device of the present invention;

[0040] **FIG. 12** shows an instruction string including a use example of a LOAD-WITH-LOOKUP instruction provided for the information processing device of the present invention;

[0041] **FIG. 13** conceptually shows a configuration of the information processing device executing the control method of the information processing device, which is one preferred embodiment of the present invention;

[0042] **FIG. 14** conceptually shows in detail a part of the configuration of the information processing device shown in **FIG. 13**, as an example;

[0043] **FIG. 15** is the circuit diagram showing a configuration of a store monitor logic provided for the information processing device shown in **FIG. 13**;

[0044] **FIG. 16** is a timing chart showing the function of the store monitor logic shown in **FIG. 15**;

[0045] **FIG. 17** conceptually shows a variation of the information processing device, which is one preferred embodiment of the present invention;

[0046] **FIG. 18** is a circuit diagram showing a configuration of a power-saving control unit provided for the information processing device, which is one preferred embodiment of the present invention of the present invention;

[0047] **FIG. 19** is a timing chart showing the operation of the power-saving control unit shown in **FIG. 18**;

[0048] **FIG. 20** is the circuit diagram of a variation of the power-saving control unit provided for the information processing device, which is one preferred embodiment of the present invention;

[0049] **FIGS. 21A and 21B** are timing charts showing the operation of the power-saving control unit shown in **FIG. 16**;

[0050] **FIG. 22** is a circuit diagram showing a configuration of a spin loop discrimination unit provided for the

information processing device, which is one preferred embodiment of the present invention; and

[0051] **FIG. 23** is a timing chart showing the operation of the spin loop discrimination unit shown in **FIG. 22**.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0052] The preferred embodiments of the present invention are described in detail below with reference to the drawings.

[0053] As shown in **FIG. 3B**, conventionally, when obtaining the lock of a lock variable [A] on a storage device, a useless spin loop repeating LD[A] after the failed acquisition by CAS[A] and checking the change of the lock variable [A] (release from other processors) is executed.

[0054] However, in the present invention, as shown in **FIG. 3A**, a CPU **1** issues a LOAD-WITH-LOOKUP instruction after the failed acquisition by CAS[A] and monitors store to lock variable [A] (possibility of release from another CPU **2**). Simultaneously, the CPU **1** is entered into a suspense status by a SUSPEND instruction. Then, the CPU **1** can be restored using the detection of the store possibility of the lock variable [A] by another CPU **2** as a trigger, and the re-acquisition of the lock variable [A] can be attempted. Thus, no useless spin loop occurs.

[0055] Specifically, as shown in **FIG. 4**, generally the CPU **1** starts monitoring a target lock variable [A] by a LOAD-WITH-LOOKUP instruction and then enters into SUSPENSE (suspense status). If the CPU **1** detects an access for the release of the lock variable [A] by the other CPU **2**, the CPU **1** is restored from the suspense status and starts executing a subsequent instruction.

[0056] The present invention analyzes the instruction string of the existing program, predicts the generation area of a spin loop and suspends/resumes the processor. Specifically, the present invention executes the step of detecting an instruction string that will become a spin loop used to detect the possibility of rewriting a lock variable from the existing instruction string and the step of suspending the relevant processor or hardware thread instead of the conventional spin loop.

[0057] However, since the recent processor is usually provided with cache, it is very difficult to monitor main memory without any process. Therefore, the present invention is provided with a writing detection function to monitor and detect the possibility of rewriting a lock variable in cache memory.

[0058] In other words, as shown in **FIG. 5**, in order to detect the possibility of rewriting a lock variable [A], the nullification on the cache memory of the lock variable [A] by the locked CPU **2** is used as a detection trigger, and the CPU **1** is restored from the suspense status.

[0059] As shown in **FIG. 6**, there is a possibility of detecting the nullification (release) on the cache memory of the lock variable [A] before the CPU **1** enters into the suspense status after the issuance of a LOAD-WITH-LOOKUP instruction. In that case, the CPU **1** does not enter into a suspense status and continues to access the lock variable [A] without any process.

[0060] The higher the detection accuracy of the possibility of rewriting a lock variable is, the higher the utilization efficiency of a processor becomes. The situation in which detection is not possible although in reality there is rewriting, must be avoided so that there may not be unreasonable hang-up.

[0061] Sometimes a suspense method for permitting resumption only by the conventional offering without address monitor, is required. Therefore, it is convenient if this can be selected.

[0062] Sometimes an instruction cannot be added to the existing instruction set. Sometimes a program cannot be revised (or it is difficult to revise a program) from old instruction codes. In such cases, no benefit can be obtained by instruction addition. Therefore, a solution method without instruction addition is desired.

[0063] As described above, in a typical lock chain, when lock cannot be secured, a memory access instruction as shown in **FIG. 7** can be observed in the execution range of several tens of instructions. Even if it is unlucky, it can be observed as shown in **FIG. 8**.

[0064] Alternatively, in the example shown in Appendix J.6 of SPARC-V9 Manual, a method in which a try routine and a loop routine are reversely arranged (if a trial fails, first of all a loop is executed) is introduced. In this case, it can be observed as shown in **FIG. 9**.

[0065] A common fact is that if two instructions accessing the same address is issued closely, as shown in **FIG. 10**, when lock acquisition fails, the processor falls into a spin loop. In other words, if there are a function to detect the approach of these two instructions, a function to suspend the instruction when detecting it and a function to start monitoring at the time point of cas and to recognize the rewriting by another system, a method for solving the conventional problems in which LOAD-WITH-LOOKUP and SUSPEND instructions are almost combined, can be obtained.

[0066] A basic example is shown in **FIG. 11**. However, it can also be applied to a series of cas instructions to the same address. Alternatively, it can also be applied to a series of ld instructions to the same address.

[0067] There can be a case where such a hardware supplementary mechanism operates as unexpected (for example, there is a possibility that it may be mistaken for a spin loop since even a semaphore mechanism uses cas). Therefore, even an expected operation must be prevented in order to remove an unexpected operation or reduce its frequency.

[0068] In order to truly improve it, it is preferable to add a LOAD-WITH-LOOKUP instruction and to explicitly designate it using an additional instruction. In other words, although both a method for adding a LOAD-WITH-LOOKUP instruction and a method for analyzing the existing instruction string can be improved, it is the best to combine them.

[0069] The conventional spin loop problem can be improved by using the LOAD-WITH-LOOKUP (llkup) instruction of the present invention. One of the improvement methods is shown in **FIG. 12**. Specifically, firstly, a lock variable [lock] is loaded onto a register %**10** by an llkup instruction, and simultaneously the monitor of store possibility of the lock variable [lock] is started. Then, the contents

of the register %**10** are evaluated by a tst instruction. If the contents are not 0 (another processor obtains its exclusive right), the execution point attempts to be branched to loop:. However, since an SUSPEND instruction that is restored by the detection of store possibility to the lock variable [lock] is executed, immediately the processor enters into a waiting status, and the processor attempts to access the lock variable [lock] again by a cas instruction or the like using the detection of the store possibility as a trigger. Thus, the processor can wait to obtain the lock variable [lock] without falling into a spin loop.

[**0070**] As described above, the fact that the rewriting frequency of the existing codes is very few is also one of the great advantages of the present invention.

[**0071**] **FIG. 13** conceptually shows a configuration of the information processing device executing the control method of the information processing device, which is one preferred embodiment of the present invention.

[**0072**] The information processing device of the preferred embodiment comprises a plurality of instruction control units **10**, a storage device **30** storing data accessed by these instruction control units and cache memory **20** which is inserted between this storage device **30** and each instruction control unit **10**, temporarily storing data to be transmitted/received between them and capable of accessing data faster than the storage device **30**.

[**0073**] Specifically, both the cache memory **20**, such as (primary operand) cache and the like, and the storage device **30** as the lower-order memory of the secondary cache, main memory or the like are connected to the instruction control unit **10** as followers. The storage device **30** is shared by he plurality of instruction control units **10**.

[**0074**] In this case, each instruction control unit **10** is in charge of the interpretation/execution of instructions and operation. Each instruction control unit **10** comprises a register. Each instruction control unit **10** also issues a memory access instruction **51** and an address **51***a* to the cache memory **20**, outputs store data **52** to be stored to the storage device **30** and inputs fetch data **53** read from the storage-device **30** through the cache memory **20**.

[**0075**] Information, such as a data request **51***b* issued from the cache memory **20** to the storage device **30**, write-back/copy-back data **52***a* used to reflect data in the cache memory of the storage device **30**, data **53***a* read into the cache memory **20** from the storage device **30** and the like, are transmitted/received between the cache memory **20** and the storage device **30**.

[**0076**] This preferred embodiment further comprises a control interface, such as a LOAD-WITH-LOOPUP instruction **54** (hereinafter called "llkup instruction **54**"), which is described later, a nullify request **55** to designate the nullification of specific data in the cache memory **20** and the like, as an memory access instruction **51**.

[**0077**] In other words, the instruction set **12** of the instruction control unit **10** of this preferred embodiment includes an llkup instruction **54**. The instruction set **12** also includes a specific SUSPEND instruction **12***a* to restore the processor from the suspense status using the detection of a store possibility detection signal **56**, which is described later, and a regular SUSPEND instruction **12***b* to permit its resumption by the conventional offering, which are properly used as requested.

[**0078**] As shown in **FIG. 14**, the cache memory **20** comprises a cache data unit **21** temporarily storing data to be transmitted/received between the instruction control unit **10** and the storage device **30**, for example, in units of lines or the like, and a cache tag unit managing data stored in this cache data unit **21**. In the cache tag unit **22**, a status **22***a*, an address **22***c* of the storage device **30**, other management information **22***d* are set for each data storage unit, such as a line in the cache data unit **21**.

[**0079**] Furthermore, in this preferred embodiment, the cache memory **20** comprises store monitoring logic **40**. This store monitoring logic **40** monitors whether data in the cache memory **20**, corresponding to the access area of the storage device **30**, of the relevant llkup instruction **54** is rewritten by another instruction control unit **10** different from the instruction control unit **10** that issued the relevant llkup instruction **54**, and transmits a store possibility detection signal **56** to the instruction control unit **10** in response to it.

[**0080**] In this preferred embodiment, the operand cache part of the cache memory **20**, such as a primary cache and the like, is important, and it can handle the target memory access of a load instruction (fetch instruction) and the like, regardless of which the cache configuration is, unified or separate, that is, whether or not instruction cache and operand cache (data cache) are separated. Although the cache memory **20** can be write-back (store-in) cache or write-through (store-through) cache, here the description is made assuming that the cache memory **20** is write-back cache.

[**0081**] The operation principle of this preferred embodiment is described below. In a shared memory system, such as SMP, cc-NUMA or the like, store order is restricted so that there will be no inconvenience when handling shared memory among processors (instruction control units **10**), and cache coherency is also maintained.

[**0082**] As to store order, there are order provisions called "TSO" in SPARC V9 of Sun Micro-System Corporation (although there are other order provisions, in reality only this is used). In short, in store provisions, a processor is specified to move so that other observers may observe it in stored order (although it is not limited to that).

[**0083**] If cache coherency is maintained in this status, a processor which performs store when the store should be performed must obtain an exclusive right to the memory area to be stored (it can also be called "writing right", that is, a status where no other processor than the processor has its memory area on the cache. Therefore, when storing, the store provisions can be observed).

[**0084**] As the status **22***a* of the cache tag unit **22**, a configuration in which cache statuses, such as MCI (modified, clean and invalid), MESI (modified, exclusive, sharable and invalid), MOEDI (modified, owned, exclusive, sharable and invalid) and the like, are stored, is known. Essentially, there are three statuses; "shared (C) (equivalent to no exclusive right since only the relevant processor may share it)", "monopolized (M)" and "invalid (I)", and MESI and MOSEI are obtained simply by adding an auxiliary status to each of them, which are described later with reference to

MCI. As to MESI and MOESI, E and M are statuses having an exclusive right, and O and S are shared statuses. Therefore, they can be similarly handled.

[0085] In order to maintain cache coherency, the cache memory **20** of the execution processor obtains an exclusive right to the relevant memory area when performing store. When the exclusive right is obtained, the corresponding memory areas of other processors enter into "invalid" statuses.

[0086] Therefore, when there is memory rewriting (store), the storing right to the relevant memory area is released (nullified) in cache memory **20** other than one to be rewritten if it has the memory area. In this preferred embodiment, access to the relevant lock variable in exclusive control by the update of data, such as a lock variable or the like, set in a storage device **30** shared by a plurality of instruction control units **10** is controlled utilizing the control mechanism of this cache memory **20**.

[0087] Specifically, a llkup instruction **54** (LOAD-WITH-LOOKUP instruction **54**) is newly set between an instruction control unit **10** and cache memory **20** as described above, according to the basic control principle of this cache memory. Then, if this llkup instruction **54** is executed, the following control operations are performed.

[0088] (1) Data is loaded onto a designated memory area.

[0089] (2) The data is registered in its own cache memory **20** accompanying item (1) above (usually shared (C)).

[0090] (3) The store monitoring logic **49** monitors this memory area, and also the instruction control unit **10** executes a special SUSPEND instruction **12***a* to enter into a suspense status.

[0091] (4) If a memory area designated by the llkup instruction **54** cannot be held for some reason, it is regarded that there is a store possibility and a store possibility detection signal **56** is returned to an instruction control unit **10** that issued the llkup instruction **54**.

[0092] (5) The instruction control unit **10** is restored from the suspense status using the reception of this store possibility detection signal **56** as a trigger.

[0093] The monitoring of a memory area by the store monitoring logic **40** can be realized on a cache tag or a register.

[0094] A method for realizing it on a cache tag is shown in **FIG. 14**. As described above, usually, the cache tag unit **22** is paired with data, and includes a cache status (one of the above-mentioned M, C, I, etc.) indicated by a status **22***a*, an address **22***c* and other management information **22***d* (privilege flag, context, etc.). LOOK flag information is provided here. Specifically, in this preferred embodiment, the cache tag unit **22** is provided with a LOOK flag **22***b* for the llkup instruction **54** for each data storage unit.

[0095] At the time of cache registration usually generated when executing a regular load, this LOOK flag **22***b* is not set. If the llkup instruction **54** is executed, the following load process is performed accompanying the registration the llkup instruction **54**.

[0096] (1) If corresponding data is stored in the cache data unit **21**, the LOOK flag **22***b* of the relevant line of the cache tag unit **22** is set.

[0097] (2) If corresponding data is not stored in the cache data unit **21**, the LOOK flag **22***b* of the relevant line is set in the cache tag unit **22** when registering the data of the relevant line.

[0098] The store possibility of other processors is determined by the nullification of a cache line in which this LOOK flag **22***b* is set. It is generally determined by the following events.

[0099] (1) A cache line becomes a replacement target.

[0100] (2) A nullify request comes from another processor than the relevant processor by an obtain request for an exclusive right from another processor or the like. Since nullification equals to entry of a cache status (status **22***a*) from M or C (in another cache status indication method, S, O, E are also possible; in any way, from all storing statuses) to I, the rewriting of the cache tag unit **22** is limited to the above-mentioned cases. At this moment, the store monitoring logic **40** checks the LOOK flag **22***b*. If the flag is set, it means a monitor status. Since this is the detection target of store possibility, it is regarded that store possibility was detected, and a store possibility detection signal is returned to the instruction control unit **10** that issued the llkup instruction **54**.

[0101] The cache method includes several methods, such as a write-through/write-back method in store management, an association method (set associative method for cache hit rate improvement), an address management method (virtually indexed physically tagged (VIPT), physically indexed physically tagged (PIPT)) and the like. However, in the present invention, their different combination can also be similarly realized without bad influences.

[0102] The circuit configuration in which a register realizes the store monitoring logic **40** and the timing chart of its operation are shown in **FIGS. 15 and 16**, respectively. Specifically, a register **41** is provided instead of providing the cache tag unit **22** with a LOOK flag **22***b*, and an address **51***a* is set by executing the llkup instruction **54**. When cache nullification is performed by a nullify request **55**, the address is compared with the contents of the set register **41** by a comparator **42**, and the output of the AND by a logic control circuit **43**, of a comparison result outputted from the comparator **42** and the nullification request **55**, is returned to the instruction control unit **10** as a store possibility detection signal **56**.

[0103] By combining the issuance of a llkup instruction **54** and the return of a store possibility detection signal **56** thus, the spin loop of the instruction control unit **10** can be avoided in exclusive control by the reference/update of a shared variable (lock variable) in the storage device **30**.

[0104] More accurately, an address range can also be set in the register **41**. For example, if a target fetch size is an llkup instruction **54** of four bytes, it can be recorded that the designated address is four bytes. If a plurality of llkup instructions **54** is executed, the range can be recorded.

[0105] In this example, a cache line size is used for the range, for convenience' sake. Even if a cache line size is

four-byte fetch, its target address range is regarded to be 64 bytes (there is no problem at all if cache is not shared by a plurality of processors. More particularly it is in a case where the cache is shared by a plurality of processors that a small address range has a meaning).

[0106] As shown in **FIG. 17**, a method in which an address range of four bytes can be used, such as a global buffer storage (GBS) method, a chip multi-processor (CMP) method and a hardware multi-thread method (a processor that executes one hardware thread in this case is called "logical processor") has an important meaning in a configuration such that one segment of cache memory **20** can be shared by a plurality of (logical) processors (instruction control units **10**A and **10**B). If an address range is used, address range determination by store executed by a shared logical processor is added to address matching determination accompanying the above-mentioned nullification. There is not always a need to limit the range, and it can also be address matching determination by a cache line. The difference is whether highly accurate determination is required or easy possibility determination is sufficient, which is also a trade-off with cost.

[0107] An example of restoration (resumption) from suspense by using a store possibility detection means, such as the above-mentioned store monitoring logic **40**, LOOK flag **22***b* or the like, is described below. A simple method for first of all entering a processor into a suspense status is to generate a SUSPEND instruction to instruct the processor to enter into a suspense status and to execute the instruction. Namely, if there is an instruction, suspense can be realized by interpreting and executing the instruction. Typically, the clock control system (power-saving control unit **60**) is configured as shown in **FIG. 18**, and is operated according to the timing chart shown in **FIG. 19**.

[0108] Specifically, in a clock control system for supplying an operation clock **71** to the instruction control unit **10**, cache memory **20** and storage device **30** from a clock supply source **70**, a power-saving control unit **60** is inserted in the supply route of the operation clock **71** to the instruction control unit **10**. This power-saving control unit **60** comprises a flip-flop circuit **61** using a suspense execution signal **57** from the instruction control unit **10** and a store possibility detection signal **56** from the store monitoring logic **40** of the cache memory **230** as set input (SET) and reset input (RESET), respectively, and an AND circuit **63** generating an operation clock **71***a* by the AND of a signal obtained logically inverting the Q output of the flip-flop circuit **61** by a logic inversion circuit **62**, and the operation clock **71**, and inputting the clock to the instruction control unit **10**.

[0109] Thus, as shown in the timing chart of **FIG. 19**, the clock supply of the execution control unit can be suspended at the time of suspense, and the clock supply can be resumed at the time of the detection of store possibility. Alternatively, the existing suspense/resumption methods can be combined.

[0110] Since the order of the execution of a SUSPEND instruction and the store possibility detection signal **56** is not guaranteed, the following execution order can be expected (**FIG. 5**).

[0111] 1. Execution of an llkup instruction through monitor start

[0112] 2. Execution of a SUSPEND instruction

[0113] 3. Detection of store possibility

[0114] However, in reality, there is a possibility that the order is reversed (**FIG. 6**) as follows.

[0115] 1. Execution of an llkup instruction through monitor start

[0116] 2. Detection of store possibility

[0117] 3. Execution of a SUSPEND instruction

[0118] More particularly, the larger the set number of monitor targets is, the higher the reverse possibility becomes. Therefore, control which takes this into consideration is needed. The power-saving control unit **60**A shown in **FIG. 20** and the timing chart shown in **FIG. 21** realize this.

[0119] In the power-saving control unit **60**A, both a flip-flop circuit **66** using the execution signal of the llkup instruction **54** as the suspense execution signal **57** as set input (SET) and reset input (RESET), respectively, and a logic inversion circuit **67** inverting the Q output of the flip-flop circuit **66** are disposed in the reset input route of the flip-flop circuit **61** shown in **FIG. 18**. Furthermore, both a logic inversion circuit **64** inverting the Q output of the flip-flop circuit **61** and an AND circuit **65** calculating the AND of the output of the logic inversion circuit **64** and a suspense during-execution signal **57***a* which always becomes true during suspense execution are inserted between the flip-flop circuit **61** and logic inversion circuit **62**.

[0120] Thus, if a store possibility detection signal **56** is regularly detected after suspense execution as shown in **FIG. 21A**, the input of the operation clock **71***a* to the instruction control unit **10** is suspended during suspense execution. However, if a store possibility detection signal **56** is detected before suspense execution as shown in **FIG. 21B**, the input of the operation clock **71***a* to the instruction control unit **10** is not suspended during suspense execution and the instruction control unit **10** continues to operate.

[0121] In the power-saving control unit **60**A, the Q output of the flip-flop circuit **61** can be used as an early store detection signal **68** for detecting that a store possibility detection signal **56** was detected before suspense execution, as requested.

[0122] As described above, the useless power consumption due to the idle-running waiting of the instruction control unit **10**, such as the conventional spin loop, can be avoided by monitoring the change of a shared variable for exclusive control due to the access of another instruction control unit **10**, using the issuance of an llkup instruction **54** as a trigger, and also entering the instruction control unit **10** into a suspense (operation clock suspense) status by a special suspend instruction **12***a* and restoring the instruction control unit **10** using the input to the power-saving control unit **60** of a store possibility detection signal **56** as a trigger, if the change of the shared variable is anticipated.

[0123] So far the effects of this preferred embodiment have been described from the viewpoint of power saving. However, in a configuration where a plurality of logical processors is mounted on one physical chip, the idle-running waiting of each logical processor means the suspense of

other processors sharing the hardware. Therefore, the application of the technology of this preferred embodiment, for precisely suspending a processor that falls into an idle-running waiting status can contribute not only to power saving but also to the effective utilization of the throughput (resources) of each logical processor.

[0124] In other words, if the present invention is applied to a processor corresponding to a hardware multi-thread, improving throughput by pretending as if a plurality of logical processors were mounted on one physical chip, not only power saving but also the effective utilization of processor throughput (processor resources) can be realized.

[0125] For example, although in a vertical multi-thread (VMT) method, a thread process is time-divisionally performed, in this preferred embodiment, a thread (logical processor) is suspended instead of idle-running waiting for exclusive control. Therefore, when the thread is suspended, its instruction processing can be suspended and another thread process can be performed with priority. This also applies to a simultaneous multi-thread (SMT) method.

[0126] The realization of a similar function by detecting a spin loop or the like without an llkup/suspend instruction or without using the instruction is described below. A similar function can be realized by operating a processor according to the result of spin-loop detection, instead of setting a LOOK flag **22***b* by an llkup instruction **54** as described above. Therefore, description is omitted after flag setting.

[0127] As to how to detect a spin loop, as described above in **FIGS. 7 through 9**, cache access (instruction string) which is characteristic of a status that easily falls into a spin loop is detected. The features are shown in **FIGS. 7 through 9**, and an example of a spin loop discrimination unit **80** catching the features is shown in **FIG. 22**.

[0128] Specifically, the spin loop discrimination unit **80** comprises a register **81** storing an address **51***a* accompanying a memory access instruction **51**, an OR circuit **82** calculating the OR of a load instruction detection signal **58***a* outputted from the decoding unit **11** of the instruction control unit **10** and a compare/swap instruction detection signal **58***c*, a comparator **83** comparing the stored contents of the register **81** with the address **51***a*, an AND circuit **87** calculating the AND of the output of the comparator **83** and the output of the OR circuit **82** and outputting the AND as a spin detection signal **87***a*, a logic inversion circuit **84** inverting the output of the comparator **83**, an AND circuit **85** calculating the AND of the output of the logic inversion circuit **84** and the output of the OR circuit **82**, an OR circuit **86** calculating the OR of the output of the AND circuit **85** and the store instruction detection signal **58***b* of the decoding unit **11** and outputting the OR as a spin release signal **86***a*, and a counter **88** which is counted up by the spin detection signal **87***a* and is reset by the spin release signal **86***a*.

[0129] Then, according to the timing chart shown in **FIG. 23**, the address **51***a* is taken in the register **81** when a compare/swap instruction detection signal **58***c* is detected. Then, if the contents of the register **81** coincide with the address **51***a* accompanying the load instruction when the load instruction detection signal **58***a* is detected, the spin detection signal **87***a* is outputted. If the address **51***a* does not coincide with the contents of the register **81** at the time of the detection of the store instruction detection signal **58***b* or at the time of the detection of the load instruction detection signal **58***a* or a compare/swap instruction detection signal **58***c*, the spin release signal **86***a* is outputted.

[0130] Then, the counter **88** is counted up by the spin detection signal **87***a*, and if the count reaches a specific value (1 acceptable), the then ld/cas instruction is regarded to be equivalent to an llkup instruction. If the counting continues and the count exceeds the specific value, the processor is suspended after the then ld/cas instruction is completed. If the spin release signal **86***a* is detected, mis-detection can be suppressed by resetting the counter **88**. Once the processor is suspended, the counter **88** is reset. Otherwise, when restoring from the suspense, the processor promptly enters into the next suspense status.

[0131] By providing such a spin loop discrimination unit **80**, a spin loop can be detected regardless of whether or not an llkup instruction **54** is mounted. Power saving can also controlled regardless of whether or not there is a suspend instruction.

[0132] As described above, according to the preferred embodiment of the present invention, the bad influence on other thread processes of a processor core can be eliminated by removing the useless idle running of a spin loop. By suspending a processor while waiting for lock acquisition and reducing the useless operation, such as idle-running waiting and the like, both power cost and operation cost can be suppressed. Furthermore, when porting software to a processor provided with a LOAD-WITH-LOOKUP instruction, a program written in the existing instruction codes can be easily rewritten, and accordingly the porting cost of software can be suppressed.

[0133] In other words, the wasteful operation cost of a processor can be suppressed, and performance can be remarkably improved. Since the present invention can be applied to any information processing device, its applicable scope is wide. Furthermore, a program written in the existing instruction codes can be easily rewritten by a LOAD-WITH-LOOKUP instruction, which is useful.

[0134] According to the present invention, the waste of both power consumption and processor resources which are due to a spin loop for exclusive control among a plurality of logical or physical processors can be reduced.

[0135] In a multi-processor system in which one physical processor core is provided with a plurality of hardware threads and which can operate as if it were provided with a plurality of logical processors, the performance degradation of other logical processors due to the spin loop of one logical processor can also be prevented.

[0136] In the multi-processor system, since the existing program can be easily rewritten, an exclusive control mechanism with the low porting cost of software can be realized.

[0137] The present invention is expressed as follows from a different viewpoint.

[0138] Note 1:

[0139] A method and architecture for monitoring memory blocks including a memory block to be loaded in an instruction set architecture having a load instruction to read a value from main memory mounted on an information processing device with cache memory.

[0140] Note 2:

[0141] A method and architecture with a load instruction (LOAD-WITH-LOOKUP instruction) with a function to monitor memory blocks including a memory block to be loaded.

[0142] Note 3:

[0143] A process method and an information processing device with a means for detecting store possibility (of other logical processors, I/O devices and the like) in the memory block monitoring range set forth in note 2.

[0144] Note 4:

[0145] A method and a device for providing the detection means, more particularly a cache tag, set forth in note 3, with a flag indicating a during-monitor status and monitoring a cache line with the flag.

[0146] Note 5:

[0147] A method and a device for providing the detection means, more particularly one or more registers, setting a monitor address and monitoring by checking an address range, based on this set address.

[0148] Note 6:

[0149] A process method and a device with cache memory for storing the data of an address to be monitored by the cache as the detection means set forth in note 3, in the cache, and determining store possibility when receiving a cache line nullify instruction (Buffer Invalidate or Copy-back and invalidate).

[0150] Note 7:

[0151] A process method and a device with cache memory for storing the data of an address to be monitored by the cache as the detection means set forth in note 3, in the cache, and determining store possibility when there is replacement.

[0152] Note 8:

[0153] A method and architecture provided with an instruction to suspend a logical processor (logical CPU-)(hereinafter called "SUSPEND instruction"), capable of restoring the processor from the suspense by the store possibility detection means set forth in note 3, as a means for restoring the processor from the suspense.

[0154] Note 9:

[0155] The method and device according to note 8, for preventing the processor from resumption by address monitor by providing a means for instructing the SUSPEND instruction not to monitor an address in order to temporarily suspend without address monitor.

[0156] Note 10:

[0157] A method and a device for restoring the processor immediately after suspense entry or interpreting a SUSPEND instruction as a non-operation instruction when detecting store possibility before suspense.

[0158] Note 11:

[0159] A store possibility detection means according to note 3, with a mechanism for detecting a case where a specific logical processor writes data in the memory block monitoring range set forth in note 2, of shared cache in a configuration where one segment of cache can be shared by a plurality of logical processors, such as shared cache (GBS method), a processor in which one physical chip includes a plurality of processor cores (CMP method) and a processor in which one physical processor core is provided with a plurality of hardware threads (strand) and which can operate as if there were a plurality of logical processors (such as, MT method HMT, SMT, VMT, etc.).

[0160] Note 12:

[0161] The method and device according to note 1, which operates regarding the set of a fetch/store instruction to operate the same address block (such as a cas instruction, an instruction to atomically load data from memory and store memory) and a load instruction as an monitor/suspend instruction, instead of the LOAD-WITH-LOOKUP/SUSPEND instruction set forth in note 2 (can also be provided).

[0162] Note 13:

[0163] The method and device according to note 1, for regarding the fetch/store instruction of the set set forth in note 12 to be equivalent to a LOAD-WITH-LOOKUP instruction, instead of the SUSPEND instruction set forth in note 8 (can also be provided).

[0164] Note 14:

[0165] The method and device according to note 1, for regarding the load instruction set forth in note 12 to be equivalent to an SUSPEND instruction, instead of the SUSPEND instruction set forth in note 8 (can also be provided).

[0166] Note 15:

[0167] The method and device according to note 1, which operates regarding a plurality of fetch/store instruction in the neighborhood to operate the same address block (such as cas instruction, etc.) as an instruction to memory blocks, instead of the LOAD-WITH-LOOKUP/SUSPEND instruction set forth in note 2(can also be provided).

[0168] Note 16:

[0169] The method and device according to note 1, for regarding the first fetch/store instruction of the set set forth in note 12 to be equivalent to a LOAD-WITH-LOOKUP instruction, instead of the SUSPEND instruction set forth in note 8 (can also be provided)

[0170] Note 17:

[0171] The method and device according to note 1, for regarding the second fetch/store instruction of the set set forth in note 15 to be equivalent to a load-with-lookup instruction, instead of the suspend instruction set forth in note 8 (can also be provided)

What is claimed is:

1. A method for controlling a processor which accesses information of a storage device through cache memory, comprising:

a first step of reading information stored in a target address or an address range of the storage device and monitoring whether there is an update access to the address or the address range from another processor;

a second step of entering the processor into a suspense status; and

a third step of releasing the suspense status using the occurrence of the update access as a trigger.

2. The processor control method according to claim 1, wherein

if the occurrence of the update access is detected between said first and second steps, the execution of said second step is suppressed.

3. The processor control method according to claim 1, wherein

part of cache management information for managing the cache memory comprises discrimination information for discriminating the storage area of the cache memory corresponding to the specific storage area of the storage device, and

in said first step the storage area of the cache memory, corresponding to the address or address range of the storage device accessed is designated as a monitor target by attaching the discrimination information, and

in said third step the suspense status is released using the occurrence of the nullification or rewriting of the information about the storage area of the cache memory as a trigger.

4. The processor control method according to claim 1, wherein

in said second step, the clock supply to the processor is suspended, and

in said third step, the clock supply is resumed.

5. A method for controlling a processor which accesses information of a storage device through cache memory, comprising:

a first step of detecting an instruction string composed of at least one of a first instruction to collectively read and update information in a target address or an address range of the storage device and a second instruction to read information about the address or address range;

a second step of monitoring whether there is an update access to the address or the address range from another processor and also entering the processor into a suspense status using detection of the instruction string as a trigger; and

a third step of releasing the suspense status using the occurrence of the update access to the address or address range from another processor as a trigger.

6. The processor control method according to claim 5, wherein

part of cache management information for managing the cache memory comprises discrimination information for discriminating the storage area of the cache memory corresponding to the specific storage area of the storage device, and

in the first step, the storage area of the cache memory, corresponding to the address or address range of the storage device accessed is designated as a monitor target by attaching the discrimination information, and

in said third step, the suspense status is released using the occurrence of the nullification or rewriting of the information about the storage area of the cache memory as a trigger.

7. The processor control method according to claim 5, wherein

in said second step, the clock supply to the processor is suspended, and

in said third step, the clock supply is resumed.

8. An information processing device, comprising:

an instruction control unit;

cache memory inserted between the instruction control unit and a storage device;

a load instruction to read information from the storage device into the instruction control unit; and

a monitor trigger setting function to set the monitor start trigger of a specific storage area of the storage device, including a target access area of the load instruction.

9. An information processing device, comprising:

an instruction control unit;

cache memory inserted between the instruction control unit and a storage device;

a load instruction to read information from the storage device into the instruction control unit; and

a writing detection function to monitor a specific storage area of the storage device, including a target access area of the load instruction and to detect a possibility that information may be written into the specific storage area.

10. The information processing device according to claim 9, wherein

part of cache management information for managing said cache memory comprises discrimination information for discriminating the storage area of said cache memory corresponding to the specific storage area of the storage device, and the storage area of the cache memory, to which the discrimination information is attached, is designated as the monitor target of said writing detection function.

11. The information processing device according to claim 9, wherein

said writing detection function comprises at least one register, and

a monitor address indicating the specific storage area is set in said register, and the specific storage area is monitored by comparing the monitor address with an access address to all segments of cache memory.

12. The information processing device according to claims **9**, wherein

if there is an instruction to nullify or rewrite information stored corresponding to the specific storage area of said cache memory, it is determined that there is a possibility of writing into the specific storage area.

13. An information processing device, comprising:

an instruction control unit;

cache memory inserted between the instruction control unit and a storage device;

a load instruction to read information from the storage device into the instruction control unit; and

a suspense instruction to enter the instruction control unit into a suspense status and to release the suspense status using detection of a possibility of writing the information into the specific storage area of the storage device, including a target access area of the load instruction as a trigger.

14. The information processing device according to claim 13, further comprising

a clock supply control function to suspend and resume clock supply to said instruction control unit in synchronization with entry to the suspense status by said suspense instruction and release from the suspense status.

15. An information processing device provided with an exclusive control mechanism for exclusively controlling access to shared memory of a plurality of instruction control units by rewriting specific information in a specific area of the shared memory, comprising

a clock supply control function to suspend clock supply to the relevant instruction control unit while one of the plurality of instruction control units is waiting for rewriting of the specific information by another instruction control unit as a trigger.

16. An information processing device, comprising

a plurality of logical or physical instruction control units;

cache memory which is inserted between each of the instruction control unit and a storage device and is shared by the plurality of instruction control units; and

a writing detection function to notify other instruction control units of a fact that one of the plurality of instruction control units has written information into a specific storage area of the storage device.

17. The information processing device according to claim 16, wherein

each of said instruction control units comprises

a suspense instruction to enter the relevant instruction control unit into a suspense status and to release the suspense status using notification from said writing detection function as a trigger.

18. A processor with an instruction set, said instruction set comprising:

a load instruction to access a storage device through cache memory and to read information from the storage device; and

a specific instruction to set the existence/non-existence of writing in a specific storage area of the storage device, including a target access area of the load instruction as a monitor start trigger.

19. An information processing device which accesses information of a storage device through cache memory, comprising:

a first function to detect an instruction string composed of at least one of a first instruction to collectively read and update information in a target address or address range of the storage device and a second instruction to read information about the address or address range; and

a second function to monitor whether there is an update access to the address or address range from another processor and also to enter the information processing device into a suspense status, using detection of the instruction string as a trigger; and

a third function to release the suspense status using the occurrence of the update access to the address or address range from another processor as a trigger.

20. Cache memory which is inserted between an instruction control unit and a storage device and temporarily stores information transmitted/received between the storage device and the instruction control unit, comprising

a writing detection function to detect a possibility that information may be written into the specific storage area of the storage device, including a target access area of a load instruction to read the information from the storage device into the instruction control unit.

21. The cache memory according to claim 20, wherein

part of cache management information for managing said cache memory comprises discrimination information for discriminating the storage area of said cache memory corresponding to the specific storage area of the storage device, and the storage area of the cache memory, to which the discrimination information is attached, is designated as the monitor target of said writing detection function.

22. The cache memory according to claim 20, wherein

said writing detection function comprises at least one register, and

a monitor address indicating the specific storage area is set in said register, and the specific storage area is monitored by comparing the monitor address with an access address to all segments of cache memory.

* * * * *