(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.[7]
G06F 12/00

(45) 공고일자 2004년04월06일
(11) 등록번호 10-0404672
(24) 등록일자 2003년10월27일

(21) 출원번호 10-1995-0055838
(22) 출원일자 1995년12월21일

(65) 공개번호 10-1996-0025043
(43) 공개일자 1996년07월20일

(30) 우선권주장 08/361,290 1994년12월22일 미국(US)

(73) 특허권자 선 마이크로시스템즈 인코퍼레이티드
미국 94303 캘리포니아주 팔로 알토 산안토니오 로드 901

(72) 발명자 데일 그린레이
미합중국 캘리포니아주 로스가토스 오버룩 로드 18401

레슬리 콘
미합중국 캘리포니아주 프리몬트 로즈미어 드라이브 43967

(74) 대리인 김명신

심사관 : 임동우

# (54) 슈퍼스칼라프로세서의로드버퍼와저장버퍼우선순위동적전환방법및장치

## 요약

본 발명은 슈퍼스칼라 프로세서에서 외부메모리 자원에 관한 로드버퍼 및 저장버퍼의 상대적인 우선순위를 동적으로 전환하기 위한 방법 및 장치에 관한 것이다. 제 1 실시예에 따르면, 프로토콜은 저장버퍼가 임의의 '고수위표'에 도달하기까지는 항상 로드버퍼가 우선순위를 갖도록 명령어하고, 저장버퍼가 임의의 고수위표에 도달하는 시간에 저장버퍼가 우선순위를 얻는다. 저장버퍼가 우선순위를 얻은 후, 저장버퍼가 '저수위표'로 소모될 때까지는 메모리에 계속해서 엑세스하고, 그 시간에 로드버퍼는 다시 우선순위를 얻는다. 저장버퍼가 고수위표에 도달할 때마다, 그것이 저수위표로 떨어질 때까지 우선순위를 얻는다. 이것은 저장버퍼가 꽉 차서 프로세서를 차단하는 경향을 감소시킨다. 제 2 실시예에 따르면, 로드버퍼가 로드버퍼의 고수위표 또는 그 위에 있으면 우선순위를 유지한다. 로드버퍼가 로드버퍼의 고수위표 아래에 있으면, 로드버퍼는 저장버퍼가 저장버퍼의 고수위표에 도달할 때까지 우선순위를 얻고, 저장버퍼가 저장버퍼의 고수위표에 도달한 시간에 저장버퍼가 우선순위를 얻는다. 저장버퍼가 우선순위를 얻은 후, 로드버퍼가 로드버퍼의 고수위표에 도달할 때까지 저장버퍼가 우선순위를 유지한다. 이것은 어느한 쪽의 버퍼가 꽉 차서 프로세서를 차단하는 경향을 감소시킨다.

## 대표도

도 3

## 명세서

## 도면의 간단한 설명

도 1은 종래의 슈퍼스칼라 프로세서의 구성을 단순화한 블록도.

도 2는 로드버퍼, 저장버퍼, 및 외부캐시 사이의 인터페이스를 특수하게 도시하는 로드/저장유닛을 단순화한 블록도.
도 3은 본 발명의 제 1 실시예에 따른 저장버퍼를 도시하는 도면.
도 4는 본 발명의 제 1 실시예에 따른 우선순위 할당 방법을 도시하는 흐름도.
도 5는 본 발명의 제 1 실시예를 구현하는데 사용될 수 있는 회로의 구조도.
도 6은 본 발명의 제 2 실시예에 따른 로드버퍼 및 저장버퍼를 도시하는 도
면.
도 7은 본 발명의 제 2 실시예에 따른 우선순위 할당 방법을 도시하는 흐름도.

## 발명의 상세한 설명

### 발명의 목적

#### 발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 마이크로프로세서 로드/저장 실행 유닛에 관한 것이다. 특히 , 본 발명은 마이크로프로세서의 로드/저장유닛에서 메모리 자원에 대해 경쟁하는 로드버퍼와 저장버퍼 사이에서 우선순위를 동적으로 제어하는 것에 관한 것이다.
RISC(Reduced Instruction Set Computing)는 감소된 명령어 집합 계산을 나타낸다. RISC프로세서는 통상적으로 CISC(Complex Instruction Set Computing) 프로세서보다 사용하는 명령어가 적으므로, RISC 기술의 사용에 따라 적어도 어느 정도까지는 RISC 프로세서를 위한 컴파일러를 쓰는 작업을 간소화시킨다. 더욱이 프로세서 설계에서 최대 동작 주파수를 제한하는 다소 복잡하고 거의 사용되지 않는 명령어을 갖는 것보다, 중요하고 빈번하게 사용되는 명령어를 구현하고 최적화하는데 중점을 둘 수 있다. 이러한 이유 및 기타 이유로, RISC 프로세서는 워크스테이션 및 하급-목적의 컴퓨터 제조업자 사이에서도 인기를 얻고 있다.
RISC 프로세서에서는 매우 적은 수의 명령어가 메모리에 실제로 엑세스하는 것이 통상적이다. 사실상 일부 구현에서는 단 2개의 명령어, LOAD 및 STORE만이 메모리에 엑세스할 수 있다. 전형적으로 몇 개의 전문화된 '지극히 작은' 동작이 동시성과 동시 프로세스에 의한 메모리 업데이트를 위해 프로세서에 의해 또한 지지될 수 있다. SPARC 구조 사용법 제9판(SPARC 인터내셔날 주식회사, 저자 D. L. Weaver 및 T. Germond, 편집자 PTR Prentice Hall, 1994년)의 17페이지를 보라. 그러한 경우조차 LOAD 및 STORE는 가장 빈번하게 사용되는 메모리 엑세스 명령어가다. LOAD 명령어의 실행은 메인메모리 번지에 상주하는 데이터를 프로세서 레지스터에 기록한다. STORE 명령어의 실행은 프로세서 레지스터에 상주하는 데이터를 메 인메모리 번지에 기록한다. 그 때문에 LOAD 및 STORE 명령어들은 하나의 레지스터 또는 다수 레지스터 및 메모리 번지를 오퍼런드(operand)로서 지정할 필요가 있다. 일부 구현에 있어서, 2개의 레지스터나 하나의 레지스터 및 즉시값(immediate value)은 메모리 번지를 계산하는데 사용된다. 대다수의 구현에서는 하나의 명령어를 사용하여 하나 이상의 데이터 워드(data word)를 전송하는 변형의 다양한 LOAD 및 STORE를 제공한다. 그러므로 하나 이상의 소스(source) 또는 목적지(destination) 레지스터가 존재할 수 있다. 이러한 소스 또는 목적지 레지스터는 각각 STORE 및 LOAD 명령어에서 오퍼런드로서 또한 표시된다.
도 1에 슈퍼스칼라 프로세서(12)가 도시되어 있다. 슈퍼스칼라 프로세서(12)는 몇개의 병렬 기능유닛(10, 11)을 구비한다. 일부 전형적인 슈퍼스칼라 구현은 부동소수점, 정수, 분기 및 로드/저장 기능유닛을 구비한다. 따라서 4개의 명령어까지 병렬적으로 실행될 수 있다. 또한 대부분의 슈퍼스칼라 프로세서는 최소한 어느 정도까지는 파이프라인으로 되어 있다. 파이프라이닝은 명령어의 처리가 몇 개의 단계로 이루어지도록 하여 하나의 명령어의 처리를 완료하는데 몇 개의 클록 사이클이 소요된다. 그렇지만, 파이프라인된 프로세서 구조에서는 각각의 단계가 하나의 명령어를 항상 처리하고 있으므로, 몇 개의 명령어들이 동시에 처리될 수 있다.
도 2에 로드/저장유닛(10)의 블록도가 도시되어 있다. 로드/저장 기능유닛(10)의 전체 처리량을 극대화하기 위해서 로드버퍼(20)와 저장버퍼(21)가 제공되며, 이로써 즉시 서비스될 수 없는 로드 및 저장이 여전히 디스패치(dispatch)될 수 있으며 또한 프로세서(12)의 동작을 지연시키지 않고 서비스되는 것을 기다릴 수 있다. 로드버퍼(20)는 LOAD 소스 메모리(22) 및 목적지레지스터 번지(23)를 보유하는 선입선출(FIFO) 대기열(queue)이다. 저장버퍼(21)는 STORE 번지(24) 및 이 번지에 저장되는 데이터(25)를 보유하는 FIFO 대기열이다. LOAD 및 STORE용 대기열을 보유함으로써 현재의 동작이 끝난 후에 다른 메모리 동작이 바로 이용가능하기 때문에 LOAD 및 STORE 명령어에 의해 사용되는 메모리 자원의 활용이 극대화된다.
다층 메모리 시스템의 통상적인 형태는 캐시메모리, 또는 루카사이드 버퍼 메모리(lookaside buffer memory)이다. 캐시메모리는 프로세서(12)와 메인메모리 시스템 사이에 배치된 다소 작고 전문화된 메모리 장치이다. 캐시메모리는 프로세서(12)에 의해 엑세스될 가능성이 있는 메인메모리의 워드 복사본을 보유한다. 캐시메모리는 메인메모리보다 더 빠르기 때문에 빈번하게 엑세스되는 위치가 캐시 안에서 발견되면 평균 메모리 엑세스 시간은 작아질 것이다. 캐시메모리 위치의 각각은 느린 메인메모리 안에 또한 저장된 정보의 더 엑세스 가능한 복사본을 제공하기위해 사용된다는 점에서 중복적이다. 프로그래머에 의해 보여지는, 엑세스 가능한 전체 메모리 크기는 캐시의 존재로 인해 증가하지는 않는다 많은 프로세서(12)는 작은 내부캐시와 별개의 집적회로 위에 구현되는 보다 큰 외부캐시를 구비하고 있다. 다른 시스템에서는 프로세서(12)와 메인메모리 사이에 오직 하나의 캐시만 존재한다.

도 2는 내부캐시(29)와 로드버퍼(20) 및 저장버퍼(21)와 인터페이스하는 로 드/저장 프리프로세싱 블록(pre-processing block)(19)을 도시한다. 로드 및 저장버퍼(20, 21)는 외부캐시(26)와 인터페이스한다. 외부캐시(26)가 별도의 집적회로 위에 있기 때문에 통상적으로 단지 하나의 읽기/쓰기 포트만 외부캐시(26) 안으로 존재한다. 현대의 프로세서에서 메모리 전송 유닛 또는 서브 블록은 144 비트 이상인 반면, 서브 블록용 번지는 20 비트 이상이다. 대다수의 시스템은 외부캐시용 이중 포트(dual-port)가 너무 비싸기 때문에 제공하지 않는다. 2개의 포트가 제공되면, 기본적으로 외부캐시 메모리칩 위의 기존의 수많은 핀을 2배로 해야 할 필요가 있다. 이러한 변경은 패키지를 더 복잡하게 하고, 외부캐시 RAM(random access memory)을 더 복잡하게 하며, 회로기판 위의 전선 수량을 증가시킨다. 잠정적으로는 외부캐시의 외형 크기도 증가한다. 더욱이 출력 패드 구동기(output pad drivers)는 상대적으로 많은 양의 총 전력을 소비하는 경향이 있으므로 시스템의 전력소비가 현저하게 증가한다. 상기한 모든 이유로, 외부캐시(26)는 통상적으로 한 세트의 번지회선(27)과 한 세트의 쌍방향 데이터회선(28)을 구비한다.
외부캐시 RAM(26)이 오직 하나의 읽기/쓰기 포트만을 구비하기 때문에, 로드버퍼(20) 안의 엔트리와 저장버퍼(21) 안의 엔트리는 외부캐시 RAM 자원을 경쟁하는 경향이 있다. 임의의 소정 시간에, 로드버퍼(20) 및 저장버퍼(21)는 모두 엔트리를 갖는다. 오직 하나의 LOAD 또는 STORE 명령어만이 임의의 소정 시간에 외부캐시(26)에 의해 처리되기 때문에, 로드 및 저장버퍼(20, 21)가 모두 엔트리를 갖는다면 버퍼 중 어느 하나는 외부캐시(26)로 그의 엔트리를 디스패치(dispatch)시킬 수 없을 것이다.
LOAD는 차후 명령어가 오퍼런드로서 사용할 데이터를 인출하는 요청이다. 따라서 LOAD는 상대적으로 빠르게 일어나야만 한다. 컴파일러는 전형적으로 로드된 데이터를 활용하는 명령어로부터 LOAD를 충분히 긴 시간으로 분리시키는 명령어 스트림을 만든다. 명령어가 사용하는 로드 데이터가 로드되기 전에 명령어가 실행될 준비가 된다면 프로세서(12)는 실속(stall)한다. 따라서 LOAD가 발생해야 할 때의 대기시간(latency)은 명령어 스트림 안에 얼마나 많은 명령어들이 로드되는 데이터의 사용으로부터 LOAD를 분리시키는가에 의존한다. 컴파일러에 제약조건이 보다 적고, 프로세서를 실속시키지 않는 것이 바람직스럽다. 로드 대기시간을 감소시킴으로써, 프로세서를 실속시키는 위험성없이 LOAD와 로드데이터 사용 사이에 개재된 명령어의 수가 감소될 수 있다.
로드버퍼(20)와 관련하여 이론적으로, 너무나 커서 내부캐시(29)에 적합하지않은 작업 세트(working set)를 갖는 코드의 경우 LOAD와 그것의 후속 사용은 적어도 외부캐시(26)의 대기시간을 충분히 포괄할 정도로 분리될 수 있다. 만일 LOAD 및 그의 후속 사용이 분리될 수 있다면, 몇 개의 개재된 명령어는 LOAD가 발생하는 동안 실행될 수 있다. 데이터를 즉시 귀환시킬 수 없는 LOAD가 파이프라인을 실속시키지 않고 오히려 그것들의 데이터를 귀환시킬 때까지 버퍼되는 점에서 로드버퍼(20)는 로드/저장 기능유닛(10)과 (도 1의) 다른 실행 파이프라인(11)이 다소 분리되도록 한다. 전형적인 예는 LOAD가 내부캐시(29)를 미스(miss)하고, 외부캐시(26)에 엑세스 해야만 하는 때이다. 이러한 경우에 LOAD는 로드버퍼(20) 안에 배치될 것이고, 그 외의 기능유닛 파이프라인(11)은 '사용' 명령어를 만나지 않 는 한 계속해서 동작할 것이다.
반면에, STORE 명령어는 LOAD가 갖는 절대적인 시간 제약조건을 갖지 않는다. STORE가 발행될 때, 프로세서(12)는 메인메모리로 데이터를 복귀시킨다. 저장된 데이터는 아마도 가까운 미래의 임의의 시간에서 필요하지 않을 것이다. 만일 저장된 데이터가 가까운 미래에 필요하게 되면, 메인메모리에 저장되기 보다는 레지스터에 보유될 것이다. LOAD용 대기시간이 STORE용 보다 더 많이 요구되기 때문에, 기존 설계는 2개의 버퍼가 외부캐시(26)를 경쟁할 때 저장버퍼 엔트리보다 로드버퍼 엔트리에 우선순위를 준다.
저장버퍼(21) 안의 엔트리보다 로드버퍼(20) 안의 엔트리가 항상 우선순위를 가짐으로써 저장버퍼(21)는 꽉 차게 되는 경향이 있다. 만일 로드버퍼(20)의 엔트리가 항상 저장버퍼(21)의 엔트리에 우선하여 서비스되면 저장버퍼(21) 안의 엔트리는 외부캐시(26)에 엑세스해야 하는 로드버퍼(20)의 엔트리가 걸 때까지 서비스될 수 없다. 로드버퍼(20)가 외부캐시(26)를 엑세스하는 엔트리를 갖는 동안은 STORE는 처리되지 않는다.
대개 일 프로그램에서 STORE의 동작의 빈도는 LOAD의 동작의 그것보다 현저하게 낮다. 그럼에도 불구하고 만일 저장버퍼의 STORE가 로드버퍼가 빌 때까지 외부캐시에 엑세스하지 못한다면 저장버퍼(21)는 그것의 모든 엔트리가 사용되는 점까지 채워진다. 일단 저장버퍼가 꽉 차면 프로세서(12)는 다른 STORE 명령어를 디스패치하는 것으로부터 차단된다. 일단 저장버퍼가 꽉 차므로써 저장버퍼 안으로 STORE 명령어가 들어가지 못하면 (도 1에서) 로드/저장 기능유닛 파이프라인(10)은 차단된다. 현재의 많은 슈퍼스칼라 프로세서(12)는 명령어 실행의 순서에 어느 정도 융통성을 제공하도록 컴파일된 코드로 작업하도록 되어 있다. 그러나 잠정적으로는 거의 모든 프로그래밍 모델이 명령어 실행의 순서에 어느 정도의 제약조건을 두기 때문에 전체 프로세서(12)가 차단될 수 있다. 명령어 스트림에서 STORE 명령어가 처리될 수 없을 때, 약간의 차후 명령어가 실행되는 것은 가능하지만, STORE 명령어가 처리되지 않는다면, 결국 전체 프로세서(12)가 실속할 것이다.
따라서 로드버퍼(20)가 항상 우선순위를 갖는 종래기술은 저장버퍼(21)가 꽉 차서 그 동작으로부터 프로세서(12)를 차단하는 문제점을 갖는다.

## 발명이 이루고자 하는 기술적 과제

따라서 본 발명은 메모리 자원에 대해 경쟁하는 로드버퍼 및 저장버퍼에 대해 저장버퍼가 꽉 차게 되어 프로세서를 차단하는 경향을 감소시키도록 메모리 자원에 대한 우선순위를 할당하는 방법 및 장치를 제공하는 것을 일 목적으로 한다.
또한 본 발명은 메모리 자원에 대해 경쟁하는 로드버퍼 및 저장버퍼에 대해 어느 한 쪽의 버퍼가 꽉 차게 되어 프로세서를 차단하는 경향을 감소시키도록 메모리 자원에 대한 우선순위를 할당하는 방법 및 장치를 제공하는 것을 다른 목적으로 한다.

## 발명의 구성 및 작용

현대의 슈퍼스칼라 마이크로프로세서에 있어서, 로드버퍼 및 저장버퍼는 보류 중인 LOAD 및 STORE를 큐잉(queueing)하는데 각각 활용된다. 이러한 2개의 버퍼 는 종종 메모리에 엑세스하기 위하여 경쟁한다.
본 발명의 제 1 실시예에 따르면, 프로토콜은 저장버퍼가 소정의 '고수위표(high water mark)'에 도달하기까지는 로드버퍼가 항상 우선하고, 저장버퍼가 상기 '고수위표'에 도달하면 그 시간에서 저장버퍼가 우선순위를 얻도록 지시한다. 저장버퍼가 우선순위를 얻은 후에 '저수위표(low water mark)'까지 소모되기 까지는 메모리에 계속해서 엑세스하고, '저수위표'에 도달하면 로드버퍼가 우선순위를 다시 얻는다. 저장버퍼가 고수위표에 도달할 때마다 저장버퍼는 저수위표로 떨어질 때까지 우선순위를 얻는다. 이것은 저장버퍼가 꽉 차게 되어 프로세서를 차단하는 경향을 감소시킨다.
본 발명의 제 2 실시예에 따르면, 로드버퍼는 로드버퍼의 고수위표 위이면 항상 우선한다. 로드버퍼가 고수위표 아래이면 저장버퍼가 저장버퍼의 고수위표에 도달하기까지는 로드버퍼가 우선하고, 저장버퍼가 저장버퍼의 고수위표에 도달하면 저장버퍼가 우선순위를 얻는다. 저장버퍼가 우선순위를 얻은 후, 로드버퍼가 로드버퍼의 고수위표에 도달하기까지는 저장버퍼가 우선순위를 유지한다. 이것은 어느 한 쪽의 버퍼가 꽉 차게 되어 프로세서를 차단하는 경향을 감소시킨다.
도 2는 M개 엔트리의 로드버퍼(20)와 N개 엔트리의 저장버퍼(21)를 도시한다. 로드버퍼(20)와 저장버퍼(21)를 사용하는 현대의 마이크로프로세서에 있어서, 이러한 2가지의 버퍼는 동일한 메모리 자원, 외부캐시(26)에 대해 경쟁한다. 본 발명에 따르면, 저장버퍼(21)가 거의 찼을 때 저장버퍼(21)가 우선순위를 얻도록 하는 우선순위 프로토콜이 사용된다 로드버퍼(20) 엔트리의 깊이(또는 수량)가 저장 버퍼(21)의 깊이와 같아야 할 필요는 없다. 이러한 버퍼들에서 엔트리의 수량은 여러가지 요인에 따라 결정된다. 이러한 버퍼들에서의 엔트리의 최적 수량을 결정하는데 고려될 수 있는 몇 가지 요인 중에, 전형적인 명령어 스트림에서 기대되는 LOAD 및 STORE의 빈도, 예상되는 내부캐시 적중률, 외부캐시 대기시간, 더 많은 엔트리를 제공하는데 필요한 하드웨어 경비 등이 있다. 본 실시예에서 로드버퍼(20)는 9개의 엔트리, 저장버퍼(21)는 8개의 엔트리다.
본 발명의 제 1 실시예에 따르면 우선순위 프로토콜은 다음과 같다. (도 3의) 저장버퍼(30)가 '고수위표(31)'에 도달하기 전까지는 (도 2의) 로드버퍼(20)가 우선순위를 유지한다. 저장버퍼(30)가 고수위표(31)에 도달하면 저장버퍼(30)가 '저수위표(32)'까지 소모될 때까지 저장버퍼(30)가 우선순위를 갖는다. 저장버퍼(30)가 저수위표(32)에 도달하면 우선순위는 로드버퍼(20)로 반환된다.
일례로서 도 3에 도시되어 있는 바와 같이, 저장버퍼(30)는 10개 엔트리의 깊이를 갖는다. 각 엔트리의 색인은 각 엔트리의 왼쪽에 표시한다. 로드버퍼(20) 및 저장버퍼(30) 모두 선입선출(FIFO) 버퍼다. 도 3에 도시된 바와 같이, 저장버퍼의 입력은 (도 2에서의) 로드/저장 프리프로세싱 유닛(19)으로부터 온다. STORE가 저장버퍼(30)로 입력될 때 서비스되기를 기다리는 STORE에 의해 미리 사용되지 않는 최하위 색인을 갖는 엔트리 안에 쓰여진다. 만일 저장버퍼(30)가 비고, 로드버퍼(20)가 외부캐시(26)를 차지하고 있다면, STORE는 제 1 저장버퍼 엔트리(도 3의 색인 1) 안으로 들어간다. 저장버퍼(30)가 외부캐시(26)를 사용하고 있을 때, 색인 1의 엔트리가 항상 서비스된다. 상기 엔트리가 처리된 후에는 나머지 엔트리(색인i)가 전부 한 위치씩 (색인 i-1의 엔트리로) 아래로 이동한다.
이러한 버퍼들은 그레인 사일로(grain silo)와 유사하다. 그레인이 덤프되었을 때, 그것은 앞서 덤프된 그레인의 바로 위로 오도록 가능한 최하위 높이로 떨어진다. 저장버퍼가 외부캐시에 엑세스할 때 사일로의 하부는 개방 상태로 되고 그레인 중 약간은 밖으로 떨어지고, 나머지 그레인은 약간씩 아래로 이동한다. 이러한 FIFO 동작은 엔트리 1에서 9까지의 사용 여부와 무관하게 새로운 STORE가 10번째(마지막) 엔트리 안에 쓰여지는 FIFO 형태와는 구별된다. 상기의 경우에 저장버퍼는 진정한 시프트 레지스터와 유사하고, STORE가 서비스되기까지는 최소 10 클록 빈도를 필요로 한다.
실제 구현에 있어서, 이러한 버퍼들은 점유되지 않은 최하의 엔트리를 통해 기록하는 능력을 제공하는 실제의 시프트 레지스터로서 구현될 수 있다. 상기 경우에 고수위표 포인터 및 저수인표 포인터는 고정값이다. 하위 엔트리가 외부캐시에 의해 서비스될 때 각각의 LOAD 또는 STORE는 버퍼의 하부에 더 가까운 한 레지스터로 이동한다.
그러나 더욱 일반적으로 이러한 버퍼들은 작은 메모리 레지스터 파일(memory register file)로서 구현된다. 그 경우에 엔트리들은 LOAD 및 STORE가 처리될 때 실제로 이동하지 않는다. 대신에 '최상부 스택 포인터' 및 '바닥부 스택 포인터'가 각 버퍼에 대해 보유된다. 바닥부 스택 포인터는 그 버퍼에서 다음에 서비스될 엔트리를 지정한다. 최상부 스택 포인터는 대기열에 들어갈 다음 LOAD 또는 STORE가 쓰여지는 엔트리를 지정한다. LOAD 또는 STORE가 서비스될 때 바닥부 스택 포인터 가 증가된다. 새로운 LOAD 또는 STORE가 버퍼 안에 도달했을 때, 최상부 스택 포인터가 증가된다. 레지스트 파일에서 가장 높은 번지로부터 증가된 포인터는 가장 낮은 번지를 지정한다. 그 때문에 버퍼의 깊이가 N이고 P가 포인터라면, 포인터가 증가될 때 새로운 P는 (예전 P + 1)을 N으로 나눈 나머지이다. 때문에 각 버퍼는 순환하는 한 쌍의 포인터를 갖는 레지스터 파일처럼 구현될 수 있다.
어떻게 물리적으로 구현하더라도 버퍼의 레벨을 놓치지 않기 위한 편리한 방법으로 버퍼 내의 엔트리의 수를 쫓는 카운터를 각각의 버퍼에 할당하는 방법이 있다. 버퍼가 빌 때, 카운터는 0을 표시한다. 엔트리가 버퍼로 들어가면 카운터는 증가된다. 엔트리에 대한 서비스가 완료되어 그래서 버퍼로부터 제거되는 경우 카운터는 감소된다. 고수위표 및 저수위표는 버퍼 우선순위의 상태를 결정하기 위하여 카운터 값과 계속적으로 비교되는 고정값으로 유지될 수 있다. 레지스터 파일의 회전 포인터 구현(curculating pointer implementation)에 적용되는 경우, 최상부 스택 포인터가 증가할 때마다 카운터는 증가되며, 바닥부 스택 포인터가 증가될 때마다 카운터는 감소된다.
본 발명에 대한 설명을 단순화하기 위하여 도면들과 발명에 대한 설명들은 동일한 레지스터가 스택의 바닥부에 항상 있는 제1 유형의 구현예와 관련된다. 본 발명은 전술한 바와 같은 다른 유형의 구현예에도 쉽게 적용되며, 특허청구범

위는 모든 구현예들을 포함한다.
또한 본 발명에서 로드 및 저장버퍼는 외부캐시와 인터페이스하는 것으로 설명된다. 그러나 여기서 설명되는 기술들은 메인메모리 또는 프로세서 메모리 구현 에서 어떤 다른 모듈과 직접 인터페이스하는 로드 및 저장버퍼에도 동일하게 적용될 수 있다. 외부캐시와 관한 논의는 설명적인 목적으로 제공된다. 특허청구범위는 다른 메모리 모델과 인터페이스하는 로드 및 저장버퍼를 포괄한다.
로드 및 저장버퍼 각각이 우선순위를 가질 때만 외부캐시에 엑세스하는 것에 LOAD 및 STORE 모두 제한되지 않는 것에 주목할 필요가 있다. 동일한 시간에 외부캐시를 요청하는 로드버퍼 내의 LOAD가 없는 동안에는 로드버퍼가 우선순위를 가질 때에도 저장버퍼 내의 STORE는 외부캐시에 엑세스할 수 있다. 이것이 종래기술의 방법에 따라 저장버퍼가 서비스되는 유일한 길이다.
주어진 프로세서 시스템의 역학을 이해함으로써, 저장버퍼(30)가 실행 파이프라인을 차단하는 것에 위험스럽게 근접하는 때가 결정될 수 있다. 외부캐시로의 STORE 동작의 대기시간은 일반적으로 일 빈도 이상이다. 만일 STORE 명령어가 매빈도 실행된다면, 저장버퍼(30)는 꽉 차게 되고, 심지어 저장버퍼가 우선순위를 가질 때조차도 넘치게 될 것이다. 그러나 만일 명령어 스트림 내에서의 STORE 명령어의 빈도에 소정의 제약조건이 주어진되면 저장버퍼(30)가 꽉 차서 발생하는 차단은 제거될 수 있다.
예를 들면, 외부캐시(26)로의 STORE가 S 클록 사이클을 소요하고, STORE 명령어가 매 S 사이클에 한번씩 보다 더욱 빈전하게 발생하지 않도록 스케쥴링된 것으로 알려져 있는 경우, 저장버퍼(30)는 실행 파이프라인을 차단하지 않도록 보장될 수 있다. LOAD 대기시간이 또한 S 클록 사이클인 것으로 가정하면 고수위표(31)는 N 엔트리 저장버퍼(30)의 엔트리 (N-1)에 있을 수 있는데, 엔트리 N-1은 저장버 퍼(30)의 '최상부'에 매우 가깝다. 고수위표는 저장버퍼(30) 내의 가장 오래된 STORE가 완료되기 전에 디스패치될 수 있는 STORE 명령어의 최대 개수이므로 단지 하나의 엔트리가 고수위표(31) 위의 저장버퍼의 최상부에서 요청된다. 저장버퍼(30)가 고수위표(31)에 도달할 때 우선순위는 저장버퍼(30)으로 전환된다. 고수위표(31)를 표시하는 저장버퍼 엔트리 내에 STORE 명령어가 도달할 때 LOAD가 외부캐시(26)를 엑세스하고 있을 수 있다. STORE가 외부캐시(26)를 엑세스 하는 것을 시작하기 전에 LOAD는 종결되도록 허용되어야 한다. 저장버퍼(30) 내의 STORERK 외부캐시(26)를 엑세스 하는 것을 시작할 수 있기 전에 LOAD는 S 클록 사이클을 소요할 수 있다. LOAD가 완결된 후에 일 이상의 STORE가 가장 오래된 STORE를 서비스하는데 필요한 S 클록 사이클 동안에 디스패치될 수 있다. 따라서 이전의 가정 하에서 단지 하나의 엔트리만이 고수위표 위에서 요청된다.
그러나 STORE 및 LOAD의 대기시간이 알려져있을지라도, STORE 동작 빈도에 아무런 제약조건이 없는 것이 보다 일반적이다. 그러한 경우에, 프로세서실속(processor stalls)이 가능하다. 예를 들어 만일 STORE가 매 클록 사이클로 발생되고, STORE를 외부캐시(26)에서 처리하는데 한 클록 사이클 이상이 걸리면, 저장버퍼(30)는 꽉 차게 되고 저장버퍼(30)가 우선순위를 가지고 있을지라도 프로세서를 차단한다. 이러한 시나리오에서 고수위표(31) 위의 엔트리의 수는, 확률적으로 모델링된(probabilisticly modeled) 명령어 분포가 주어지는 경우 최선의 가능성 있는 성능과 경비의 트레이드오프(tradeoff)를 제공하기 위해 최적유형 검사 기술을 통해 선택될 수 있다. 고수위표(31) 위의 엔트리가 많을수록 실행 파이 프라인은 꽉 찬 STORE 버퍼(30)에 의해 차단될 가능성이 줄어든다.
고수위표(31) 위의 엔트리 수를 증가시키는 데는 몇가지가 필요하다. 주어진 저장버퍼의 깊이에 대해 고수위표(31) 위의 엔트리 수가 증가될 때, 고수위표(31)아래의 엔트리 수는 감소된다. 그러므로 고수위표(31) 위의 엔트리 수를 증가시키므로써 저장버퍼 차단의 빈도가 감소됨에도 불구하고, 로드버퍼(20)가 우선순위를 더 빨리 잃기 때문에 평균 LOAD 대기시간은 증가할 것이다. 평균 LOAD 대기시간이 증가하면 차후 명령어의 오퍼런드로서 사용되기 위하여 필요한 시간까지 데이터가 로드되지 않기 때문에 더 자주 프로세서가 실속된다. 또한 저장버퍼(21)가 우선순위를 갖는 동안 로드버퍼(20)는 꽉 찰 수 있다. LOAD 명령어가 꽉 찬 로드버퍼(20)안으로 디스패치될 수 없으므로 꽉 찬 로드버퍼(20)는 프로세서를 또한 차단한다. 따라서 저장버퍼(30) 안에서 고수위표(31) 아래에 너무 적은 엔트리를 갖는 것은 로드 대기시간에 의한 실속 빈도를 증가시킨다.
본 발명의 제 1 실시예에 따라 일단 저장버퍼(30)가 우선순위를 가지면, 그것의 엔트리들은 '저수위표(32)'에 도달할 때까지 처리된다. 저장버퍼(21)가 저수위표(32)에 도달할 때, 로드버퍼(20)는 다시 우선순위를 얻는다 저수위표(32)의 적절한 위치는 많은 변수에 다시 의존한다. 저장버퍼(30)가 우선순위를 갖는 동안에, STORE는 외부캐시(26)를 계속적으로 이용한다. 따라서 저장버퍼가 우선순위를 유지하고 외부캐시(26)를 차지하는 최소 시간이 계산될 수 있다. 고수위표(31)에 도달하자마자 외부캐시(26)에 저장버퍼(30)은 그것의 엔트리를 제출하기 시작하므로, 그리고 저수위표에 도달할 때까지 외부캐시(26)로 엔트리를 제공하는 것을 계속하 므로 저장버퍼가 외부캐시(26)를 차지하는 최소 기간은 STORE 대기시간에 고수위표(31)로부터 저수위표(32)까지의 엔트리의 수를 곱한 것과 같다. 따라서 저수위표(32) 위치의 가장 중요한 파라미터는 고수위표(31)와 저수위표(32) 사이의 엔트리의 수이다. 저장버퍼가 우선순위를 가지는 동안 STORE는 저장버퍼에 진입할 수 있으므로, 따라서 저장버퍼가 외부캐시를 차지하는 시간을 증가시킨다. 더 많은 엔트리를 저수위표(32) 아래에 가지는 것이 평균 STORE 대기시간을 증가시키는 정도로 저수위표(32) 아래의 엔트리의 수가 중요한데, 통상 평균 STORE 대기시간은 대부분 타이밍에 치명적이지 않다.
실제 구현에 있어서는 전술한 일반 규칙에 약간의 예외가 있을 수 있다. 어떤 경우에 저장버퍼가 고수위표에 도달한 후이나 저수위표에 도달하기 전에 저장버퍼가 우선순위를 로드버퍼에 반환할 수 있다. 제1 예로서, 프로세서는 LOAD에 논리적으로 연결된 STORE타입을 지원할 수 있다. 이러한 경우에 LOAD가 로드버퍼를 떠나 닫힐 때까지는 STORE는 저장버퍼를 떠날 수 없다. 만일 이 경우에 로드버퍼가 다시 우선순위를 얻을 수 없다면, 어느쪽 버퍼도 외부캐시에 엑세스할 수 없기 때문에, 교착상태(deadlock)에 빠진다. 제 2 예로서, 때때로 STORE는 먼저 캐시에 엑세스할 수 없는 STORE 같은 다른 사건이 종료될 때까지는 저장버퍼를 떠날 수 없다. 캐시에 엑세스할 수 없는 STORE가 외부캐시에 엑세스하지 않기 때문에, 먼저 캐시에 엑세스할 수 없는 STORE는 메인메모리나 그 밖의 다른 곳에서 처리

되어 매우 오랜 시간을 소요하게 된다. 이러한 경우에 저수위표에 도달하지 않았다면 저장버퍼는 우선순위를 포기할 수 있다.
실제 구현예에서, 저장버퍼가 고수위표에 도달하기 전에 우선순위를 갖는 상황도 있다. 예를 들어 다중 프로세서 구조에서 저장버퍼가 로드버퍼에 '분실' 중재('lost' arbitration)를 수차례 취할 때, 저장버퍼가 우선순위를 갖는 것이 허용될 수 있다. 예를 들어, 여하한 새로운 데이터가 '메일박스' 위치('mailbox' location)로 전송되었는지 여부를 알기 위하여 연속적으로 그 위치에 LOAD 하는 매우 빡빡한 루프를 하나의 프로세서가 실행하고 있을 때, 이러한 상황이 발생할 수 있다. 제2 프로세서가 메일박스에 데이터를 저장하기 위해 시도하고 있는 경우, 제1 프로세서가 연속적으로 메일박스를 연속적으로 잠깐씩 들르고 있으므로 STORE 가 그것의 '메일'을 전송하는 것이 허용되지 않으면 교착상태가 발생할 것이다. 여기에서 설명된 일반 규칙에 대한 이러한 예외가 존재하며, 특허청구범위는 이러한 변경도 포괄한다.
도 4는 로드 및 저장버퍼의 제어를 도시한다. 시스템이 초기화되고 로드버퍼와 저장버퍼 모두가 비었을 때, 시스템은 로드버퍼가 우선순위를 가지는 상태(41)로 동작을 시작한다(40). 로드버퍼가 우선순위를 갖는 동안, 저장버퍼가 고수위표 또는 그 위로 채워졌는지를 계속적으로 검사한다(42). 저장버퍼 내의 STORE의 수가 고수위표 아래에 있는 동안은 제어는 분기(43)를 취하고, 로드버퍼는 우선순위를 유지한다(41). 다음으로 단계(42)에서 저장버퍼가 고수위표 또는 그 위로 STORE로 채워졌다고 결정되면, 분기(44)를 취하고, 저장버퍼는 우선순위를 갖는다(45). 저장버퍼가 우선순위를 갖는 동안은 단계(46)에서 저장버퍼가 저수위표 또는 그 아래로 떨어졌는지를 계속해서 검사한다. 저장버퍼 내의 STORE의 수가 저수위표 위에있 는 동안은 제어는 분기(47)를 취하고, 저장버퍼가 우선순위를 유지한다(45). 단계(46)에서 저장버퍼가 저수위표 또는 그 아래로 소모되었다고 결정되면 분기(48)를 취하고, 로드버퍼가 다시 우선순위를 갖는다(41).
도 5는 이러한 방안에 대한 제어과정으로 가능한 하나의 구현예를 도시한다. 외부캐시 RAM은 한 세트의 번지 입력(51)을 갖는다. 멀티플렉서(52)는 로드버퍼 번지(53) 또는 저장버퍼 번지(54)가 외부캐시 RAM 번지 입력(51)을 구동할 것인지 여부를 선택한다. 결정 블록(decision block : 55)은 극히 단순한 알고리즘을 구현한다. '로드 엑세스' 신호(56)가 제공되면 로드버퍼 번지(53)가 선택된다. 로드 엑세스 신호(56)가 제공되지 않고 '저장버퍼 요청' 신호(57)가 제공되면 저장버퍼번지(54)가 선택된다. 제 1 실시예에 따른 제어 알고리즘을 구현하기 위해서 로드엑세스 신호(56)를 계산하는 '억제' 블록(58)이 도입된다. 고수위표(31)에 도달할 때 억제 블록(58)은 저장버퍼 우선순위 신호(50)에 의해 로드 엑세스 신호(56)를 제공하지 못한다. 이것이 발생할 때마다, (도 3에서) 저장버퍼(30)에서의 엔트리가 있기 때문에, 저장버퍼 요청 신호(57)의 제공이 보증된다. 저장버퍼(30)가 서비스된 후에 저장버퍼(30)에서의 엔트리 수가 저수위표(32)에 도달할 때, 로드 엑세스신호(56)를 만들어내는 억제 블록(58)은 '로드버퍼 요청' 신호(59)가 제공되면 다시 그 신호가 제공되도록 허용한다.
실제로는 로드버퍼 및 저장버퍼를 제외하고도 로드 및 저장버퍼보다 우선순위가 낮으면서 동일한 메모리 자원을 경쟁하는 유닛들이 있다. 예를 들면, 명령어 인출 유닛(instruction fetch unit) 또한 동일한 자원을 이용한다. 그러한 경우에, 결정 블록(55)은 보다 많은 입력을 갖고, 멀티플렉서(52) 또한 추가입력을 갖는다.
이러한 제 1 실시예는 (도 2에서의) 로드버퍼(20)의 상태에 전혀 주의를 기울이지 않는다. 따라서 로드버퍼(20)는 저장버퍼(21)가 우선순위를 갖는 동안 꽉 찰 수 있다. 만일 저장버퍼(21)가 우선순위를 갖고 소모되는 동안 몇 개의 LOAD가 로드버퍼(20) 안으로 발행되면 로드버퍼(20)는 실행 파이프를 차단할 수 있다.
도 6는 본 발명의 제 2 실시예에 따른 로드버퍼(60) 및 저장버퍼(61)를 예시한다. 이 실시예에 따르면 우선순위 프로토콜은 다음과 같다. 로드버퍼(60)는 저장버퍼(61)가 '고수위표'(62)에 도달하기까지 우선순위를 유지한다. 저장버퍼(61)가 그것의 고수위표(62)에 도달할 때, 저장버퍼(62)는 로드버퍼(60)가 그 자신의 고수위표(63)에 도달하기까지 우선순위를 얻는다. 로드버퍼(60)가 그것의 고수위표(63)에 도달할 때, 우선순위는 로드버퍼(60)로 복귀한다. 만일 양쪽 버퍼(60, 61)가 각자의 고수위표(63, 62) 보다 위라면, 로드버퍼(60)가 우선순위를 갖는다. 도 6는 8개 엔트리의 로드버퍼(60) 및 10개 엔트리의 저장버퍼(61)를 도시한다. 고수위표(63, 62)의 레벨은 독립적으로 설계된다.
도 7는 본 발명의 제 2 실시예에 따른 제어의 진행을 도시하는 흐름도이다. 시스템이 초기화될 때, 시작 단계(70)에서 시스템은 로드버퍼가 우선순위를 갖는 상태(71)로 된다. 단계(72)에서 로드버퍼가 그것의 고수위표 또는 그 위에 있는지 여부를 검사한다. 만일 로드버퍼가 고수위표 또는 그 위로 채워지면, 분기(73)를 취하여 로드버퍼가 우선순위를 유지한다(71). 만일 로드버퍼가 고수위표 아래이면, 분기(74)를 취하여 단계(75)를 처리한다. 단계(75)에서 저장버퍼가 그것의 고수위 표 또는 그 위에 있는지 여부를 검사한다. 만일 저장버퍼가 그것의 고수위표 아래에 있으면, 분기(76)을 취하여 다시 로드버퍼가 우선순위를 갖는다(71). 만일 저장버퍼가 그것의 고수위표 또는 위에 있으면, 분기(77)를 취하여 로드버퍼가 그것의 고수위표에 다시 도달(72)할 때까지 저장버퍼가 우선순위를 갖는다(78).
이러한 방안은 버퍼 중 어느 한 쪽이 꽉 차서 발생하는 프로세서의 차단을 없애는 것에 가장 큰 중점을 두고 있다. 차단 방지의 관점에서, 이 방안은 제 1 실시예의 어느 하나 보다 성능이 우수하다. 그러나 이 방안은 장시간 동안 저장버퍼가 우선순위를 얻고 유지하는 것을 허용한다. 따라서 이 실시예에서는 차단 방지를 위해서 LOAD 명령어에 대한 대기시간 요건을 필요로 한다. 그러므로 프로세서는 현재 명령어에 의해 사용될 필요가 있는 로드 데이터를 사용할 수 없는 것에 기인하여 실속될 수 있다.
(도 6의) 저장버퍼(61)가 고수위표(62)에 도달하여 우선순위를 얻는 경우, 로드버퍼(60)는 그것의 고수위표(63)에 도달할 때까지 다시 우선순위를 얻지 못한다. 로드버퍼(60) 안에 몇 개의 엔트리가 있는 동안에도 저장버퍼(61)가 우선순위를 얻는 것이 가능하다. 다음에 일련의 STORE가 디스패치되더라도 저장버퍼(61)가 비지 않고 로드버퍼(60)가 그것의 고수위표(63)에 도달되지 않는 한 저장버퍼(61)는 외부캐시에 대한 제어를 유지할 것이다. 추가적인 LOAD가 고수위표(63) 위로 로드버퍼(60)에 넣기 위해 디스패치되지 않는 경우, 로드버퍼(60) 안에 현재 존재하는 LOAD는 점점 더 오래된 것이 되면서 그곳에 머문다. STORE가 모두 비거나 또는 로드버퍼(60)가 그것의 고수위표(63)에 도

달하도록 로드버퍼(60) 안으로 새로운 엔 트리가 디스패치되기까지는 로드버퍼(60)는 다시 서비스되지 않을 것이다. 로드 데이터를 필요로 하는 후속 '사용' 명령어가 실행될 준비가 되기 전에 LOAD가 서비스 되지 않는 경우, 프로세서는 실속한다. 이러한 방안을 사용하면 저장버퍼가 우선순위를 한번 얻으면 심지어 그것이 비어도 우선순위를 유지한다. 로드버퍼가 다시 우선순위를 얻기 위해서는 그것의 고수위표에 도달해야만 한다.

어느 실시예의 경우에도 시스템에 융통성을 추가하기 위해 또 다른 변경이 가능할 수 있다. 이러한 변경에 의해서, 수위표는 최적의 성능이 가능하도록 수위표의 레벨을 세팅하기 위해 운영 시스템 또는 컴파일러에 의해 프로그램 가능하게 만들어진다. 따라서 제 1 실시예에서 저장버퍼의 고수위표 및 저수위표는 동적으로 프로그램 가능하다. 제 1 실시예의 경우 저장버퍼 내의 고수위표 및 저수위표의 레벨은 실행되고 있는 프로그램의 명령어 스트림의 성질에 따라 동적으로 변경될 수 있다. 비슷하게 저장버퍼의 고수위표 및 로드버퍼의 고수위표의 레벨은 동적으로 변경 가능하다.

본 발명의 방법 및 장치가 바람직한 몇 개의 실시예들을 참조하여 위에 개시되어 있으나, 당업자들은 이러한 개시로부터 첨부된 특허청구범위의 정신 및 범위내에서 다양한 변형이나 변경을 할 수 있다. 따라서 상세한 설명 및 도면은 제한적인 의미보다는 설명적인 것으로 고려되어야 한다.

## 발명의 효과

전술한 바와 같은 본 발명에 의하면 메모리 자원에 대해 경쟁하는 로드버퍼 및 저장버퍼에 대해 우선순위를 합리적으로 할당하여 저장버퍼가 꽉 차게 되어 프 로세서를 차단하는 경향을 감소시킬 수 있다. 또한 어느 한 쪽의 버퍼가 꽉 차게 되어 프로세서를 차단하는 경향을 감소시킬 수 있다.

## (57) 청구의 범위

### 청구항 1.

메모리 자원에 대해 경쟁하는 로드버퍼(load buffer) 및 저장버퍼(store buffer)에 우선순위를 할당하는 방법에 있어서,
(a) 상기 로드버퍼에 우선순위를 부여하는 단계와,
(b) 상기 저장버퍼가 잠재적이고 임박한 실속 조건(potential impending stall condition)을 표시하는 상위 임계값(upper threshold)의 위에 있는지 여부를 검사하는 단계와,
(c) 단계 (b)에서 상기 저장버퍼가 상기 상위 임계값의 위에 있는 것으로 판정되면 잠재적인 실속를 회피하기 위하여 상기 저장버퍼에 우선순위를 부여하는 단계와,
(d) 단계 (b)에서 상기 저장버퍼가 상기 상위 임계값에 있거나 상기 상위 임계값의 아래에 있는 것으로 판정되면 단계 (a)로 복귀하는 단계를
포함하는 것을 특징으로 하는 우선순위 할당 방법.

### 청구항 2.

메모리 자원에 대해 경쟁하는 로드버퍼(load btlffer) 및 저장버퍼(store buffer)에 우선순위를 할당하는 방법에 있어서,
(a) 상기 로드버퍼에 우선순위를 부여하는 단계와,
(b) 상기 저장버퍼가 상위 임계값(upper threshold)의 위에 있는지 여부를 검사하는 단계와,
(c) 단계 (b)에서 상기 저장버퍼가 상기 상위 임계값의 위에 있는 것으로 판정되면 상기 저장버퍼에 우선순위를 부여하는 단계와,
(d) 단계 (b)에서 상기 저장버퍼가 상기 상위 임계값에 있거나 상기 상위 임계값의 아래에 있는 것으로 판정되면 단계 (a)로 복귀하는 단계와,
(e) 상기 저장버퍼가 하위 임계값의 아래에 있는지 여부를 검사하는 단계와,
(f) 단계 (e)에서 상기 저장버퍼가 상기 하위 임계값에 있거나 상기 하위 임계값의 위에 있는 것으로 판정되면 상기 저장버퍼에 우선순위를 부여하고 단계 (e)로 복귀하는 단계와,
(g) 단계 (e)에서 상기 저장버퍼가 상기 하위 임계값의 아래에 있는 것으로 판정되면 단계 (a)로 복귀하는 단계를
포함하는 것을 특징으로 하는 우선순위 할당 방법.

### 청구항 3.

메모리 자원에 대해 경쟁하는 로드버퍼(load buffer) 및 저장버퍼(store buffer)에 우선순위를 할당하는 방법에 있어서,
(a) 상기 로드버퍼에 우선순위를 부여하는 단계와,
(b) 상기 로드버퍼가 잠재적이고 임박한 실속 조건을 표시하는 로드버퍼 상위 임계값의 위에 있는지 여부를 검사하는 단계와,
(c) 단계 (b)에서 상기 로드버퍼가 상기 로드버퍼 상위 임계값의 위에 있는 것으로 판정되면 잠재적인 실속를 회피하기 위하여 단계 (a)로 복귀하는 단계를
포함하는 것을 특징으로 하는 우선순위 할당 방법.

### 청구항 4.

메모리 자원에 대해 경쟁하는 로드버퍼(load buffer) 및 저장버퍼(store buffer)에 우선순위를 할당하는 방법에 있어서,

(a) 상기 로드버퍼에 우선순위를 부여하는 단계와,
(b) 상기 로드버퍼가 로드버퍼 상위 임계값의 위에 있는지 여부를 검사하는 단계와,
(c) 단계 (b)에서 상기 로드버퍼가 상기 로드버퍼 상위 임계값의 위에 있는 것으로 판정되면 단계 (a)로 복귀하는 단계와,
(d) 단계 (b)에서 상기 로드버퍼가 상기 로드버퍼 상위 임계값에 있거나 상기 로드버퍼 상위 임계값의 아래에 있는 것으로 판정되면 상기 저장버퍼가 저장버퍼 상위 임계값의 위에 있는지 여부를 검사하는 단계와,
(e) 단계 (d)에서 상기 저장버퍼가 상기 저장버퍼 상위 임계값에 있거나 상기 저장버퍼 상위 임계값의 아래에 있는 것으로 판정되면 단계 (a)로 복귀하는 단계와,
(f) 단계 (d)에서 상기 저장버퍼가 상기 저장버퍼 상위 임계값의 위에 있는지 것으로 판정되면 상기 저장버퍼에 우선순위를 부여하는 단계와,
(g) 단계 (b)로 복귀하는 단계를
포함하는 것을 특징으로 하는 우선순위 할당 방법.

**청구항 5.**

메모리 번지를 제공하는 장치에 있어서,
로드버퍼 요청 입력, 저장버퍼 우선순위 입력, 및 로드 액세스 출력을 갖는 억제회로(inhibit circuit)와,
선택입력과 상기 메모리 번지를 전달하는 출력을 갖는 2 경로 멀티플렉서(two-way multiplexer)-여기서, 하나의 입력으로 로드버퍼 번지가 제공되고, 다른 하나의 입력으로 저장버퍼 번지가 제공됨-와,
제1 및 제2 입력과 출력을 갖는 결정회로-여기서, 상기 결정회로의 제1 입력으로 저장버퍼 요청 신호가 제공되고, 상기 제2 입력으로 상기 억제회로의 상기 로드 액세스 출력이 제공되며, 상기 출력은 상기 2 경로 멀티플렉서의 선택입력을 제공함-를
는 것을 특징으로 하는 메모리 번지 제공 장치

**청구항 6.**

메모리 번지를 제공하는 장치에 있어서,
로드버퍼 요청 입력, 저장버퍼 우선순위 입력, 및 로드 액세스 출력을 갖는 억제회로(inhibit circuit)와,
선택입력과 상기 메모리 번지를 전달하는 출력을 갖는 2 경로 멀티플렉서(two-way multiplexer)-여기서, 하나의 입력으로 로드버퍼 번지가 제공되고, 다른 하나의 입력으로 저장버퍼 번지가 제공됨-와,
제1 및 제2 입력과 출력을 갖는 결정회로를 구비하며,
상기 결정회로의 제1 입력으로 저장버퍼 요청 신호가 제공되고, 상기 제2 입력으로 상기 억제회로의 상기 로드 액세스 출력이 제공되며, 상기 출력은 상기 2 경로 멀티플렉서의 선택입력을 제공하고,
상기 로드버퍼 요청 입력과 상기 저장버퍼 우선순위 입력이 모두 제공될 때 상기 로드 액세스 출력이 제공되며,
상기 로드버퍼 요청 입력 또는 상기 저장버퍼 우선순위 입력이 제공되지 않을 때 상기 로드 액세스 출력이 제공되지 않는
것을 특징으로 하는 메모리 번지 제공 장치.

**청구항 7.**

메모리 번지 제공 장치에 있어서,
로드버퍼 요청 입력, 저장버퍼 우선순위 입력, 및 로드 액세스 출력을 갖는 억제회로(inhibit circuit)와,
선택입력과 상기 메모리 번지를 전달하는 출력을 갖는 2 경로 멀티플렉서(two-way multiplexer)-여기서, 하나의 입력으로 로드버퍼 번지가 제공되고, 다른 하나의 입력으로 저장버퍼 번지가 제공됨-와,
제1 및 제2 입력과 출력을 갖는 결정회로-여기서, 상기 결정회로의 제1 입력으로 저장버퍼 요청 신호가 제공되고, 상기 제2 입력으로 상기 억제회로의 상기 로드 액세스 출력이 제공되며, 상기 출력은 상기 2 경로 멀티플렉서의 선택입력을 제공함-를
구비하며,
상기 결정회로는 상기 로드 액세스 신호가 제공될 때 상기 로드버퍼 번지를 선택하는 출력을 만들고, 상기 로드 액세스 신호가 제공되지 않거나 상기 저장버퍼 요청 신호가 제공될 때 상기 저장버퍼 번지를 선택하는 출력을 만드는
것을 특징으로 하는 메모리 번지 제공 장치.

**청구항 8.**

메모리 번지 출력을 제공하는 방법에 있어서,
저장버퍼가 우선순위를 가지는 경우 로드버퍼에 의해 출력되는 로드버퍼 요청이 발생하는 것을 억제하는 단계와,
로드버퍼 요청이 발생하는 경우 출력용 로드버퍼 번지를 선택하는 단계와,
상기 저장버퍼에 의해 출력되는 저장버퍼 요청이 로드버퍼 요청이 없는 상태에서 발생하는 경우 출력용 저장버퍼 번지를 선택하는 단계를
포함하는 것을 특징으로 하는 메모리 번지 제공 방법.

**청구항 9.**

제 8 항에 있어서,
상기 억제 단계는 상기 로드버퍼로부터의 신호와 상기 저장버퍼 우선순위 신호를 논리곱 함으로써 수행되는 것을 특징으로 하는 메모리 번지 제공 방법.

**청구항 10.**

메모리 자원에 대해 경쟁하는 로드버퍼와 저장버퍼에 우선순위를 할당하는 장치에 있어서,

저장버퍼 상위 임계값 신호를 출력으로 가지며, 상기 저장버퍼가 잠재적이고 임박한 실속 조건을 표시하는 상위 임계값의 위에 있는지 여부를 검사하는 제1 테스트 회로와,
상기 저장버퍼 상위 임계값 신호를 입력으로 취하며, 상기 저장버퍼가 상위 임계값의 위에 경우 상기 저장버퍼에 우선순위를 부여하는 저장버퍼 우선순위 회로를 구비하고,
상기 장치는 잠재적인 실속를 회피하기 위하여 상기 로드버퍼 및 상기 저장버퍼에 상대적인 우선순위를 할당하도록 동작 가능한
것을 특징으로 하는 우선순위 할당 장치.

**청구항 11.**

메모리 자원에 대해 경쟁하는 로드버퍼와 저장버퍼에 우선순위를 할당하는 장치에 있어서,
저장버퍼 상위 임계값 신호를 출력으로 가지며, 상기 저장버퍼가 상위 임계값의 위에 있는지 여부를 검사하는 제1 테스트 회로와,
상기 저장버퍼 상위 임계값 신호를 입력으로 취하며, 상기 저장버퍼가 상위임계값의 위에 있는 경우 상기 저장버퍼에 우선순위를 부여하는 저장버퍼 우선순위회로와,
저장버퍼 하위 임계값 신호를 출력으로 가지며, 상기 저장버퍼가 하위 임계값의 아래에 있는지 여부를 검사하는 제2 테스트 회로와,
상기 저장버퍼 하위 임계값 신호를 입력으로 취하며, 상기 저장버퍼가 상기 하위 임계값의 아래에 있는 경우 상기 로드버퍼에 우선순위를 부여하는 로드버퍼 우선순위 회로를
포함하는 것을 특징으로 하는 우선순위 할당 장치

**청구항 12.**

메모리 자원에 대해 경쟁하는 로드버퍼와 저장버퍼에 우선순위를 할당하는 장치에 있어서,
로드버퍼 상위 임계값 신호를 출력으로 가지며, 상기 로드버퍼가 잠재적이고 임박한 실속 조건을 표시하는 로드버퍼 상위 임계값의 위에 있는지 여부를 검사하는 제1 테스트 회로와,
상기 로드버퍼 상위 임계값 신호를 입력으로 취하며, 상기 로드버퍼가 상기 로드버퍼 상위 임계값의 위에 있는 경우 잠재적인 실속를 회피하기 위하여 상기 로드버퍼에 우선순위를 부여하는 로드버퍼 우선순위 회로를
포함하는 것을 특징으로 하는 우선순위 할당 장치.

**청구항 13.**

메모리 자원에 대해 경쟁하는 로드버퍼와 저장버퍼에 우선순위를 할당하는 장치에 있어서,
로드버퍼 상위 임계값 신호를 출력으로 가지며, 상기 로드버퍼가 로드버퍼 상위 임계값의 위에 있는지 여부를 검사하는 제1 테스트 회로와,
상기 로드버퍼 상위 임계값 신호를 입력으로 취하며, 상기 로드버퍼가 상기 로드버퍼 상위 임계값의 위에 있는 경우 상기 로드버퍼에 우선순위를 부여하는 로드버퍼 우선순위 회로와,
저장버퍼 상위 임계값 신호를 출력으로 가지며, 상기 저장버퍼가 저장버퍼 상위 임계값의 위에 있는지 여부를 검사하는 제2 테스트 회로와,
상기 저장버퍼 상위 임계값 신호를 입력으로 취하며, 상기 저장버퍼가 상기 저장버퍼 상위 임계값의 위에 있고, 상기 로드버퍼가 상기 로드버퍼 상위 임계값의 아래에 있는 경우 상기 저장버퍼에 우선순위를 부여하는 저장버퍼 우선순위 회로를 포함하는 것을 특징으로 하는 우선순위 할당 장치.

**청구항 14.**

메모리 번지 출력을 제공하는 장치에 있어서,
억제 신호를 출력으로 가지며, 저장버퍼가 우선순위를 가지는 경우 로드버퍼에 의해 출력된 로드버퍼 요청이 발생하는 것을 억제하는 억제회로와,
상기 억제 신호를 입력으로 취하며, 로드버퍼 요청이 발생하는 경우 출력을 위해 로드버퍼 번지를 선택하고, 로드버퍼 요청이 없는 상태에서 상기 저장버퍼에 의해 출력된 저장버퍼 요청이 발생하는 경우 출력을 위해 저장버퍼 번지를 선택하는 선택회로를
포함하는 것을 특징으로 하는 메모리 번지 출력 제공 장치.

**청구항 15.**

제 14 항에 있어서,
상기 억제회로는 상기 로드버퍼로부터의 신호와 저장버퍼 우선순위 신호에 대한 논리곱을 수행하는 것을 특징으로 하는 메모리 번지 출력 제공 장치.

**청구항 16.**

제 1 항에 있어서,
상기 상위 임계값은 프로그램 가능한 것을 특징으로 하는 우선순위 할당 방 법.

**청구항 17.**

제 2 항에 있어서,
상기 하위 임계값은 프로그램 가능한 특징으로 하는 우선순위 할당 방법.

**청구항 18.**

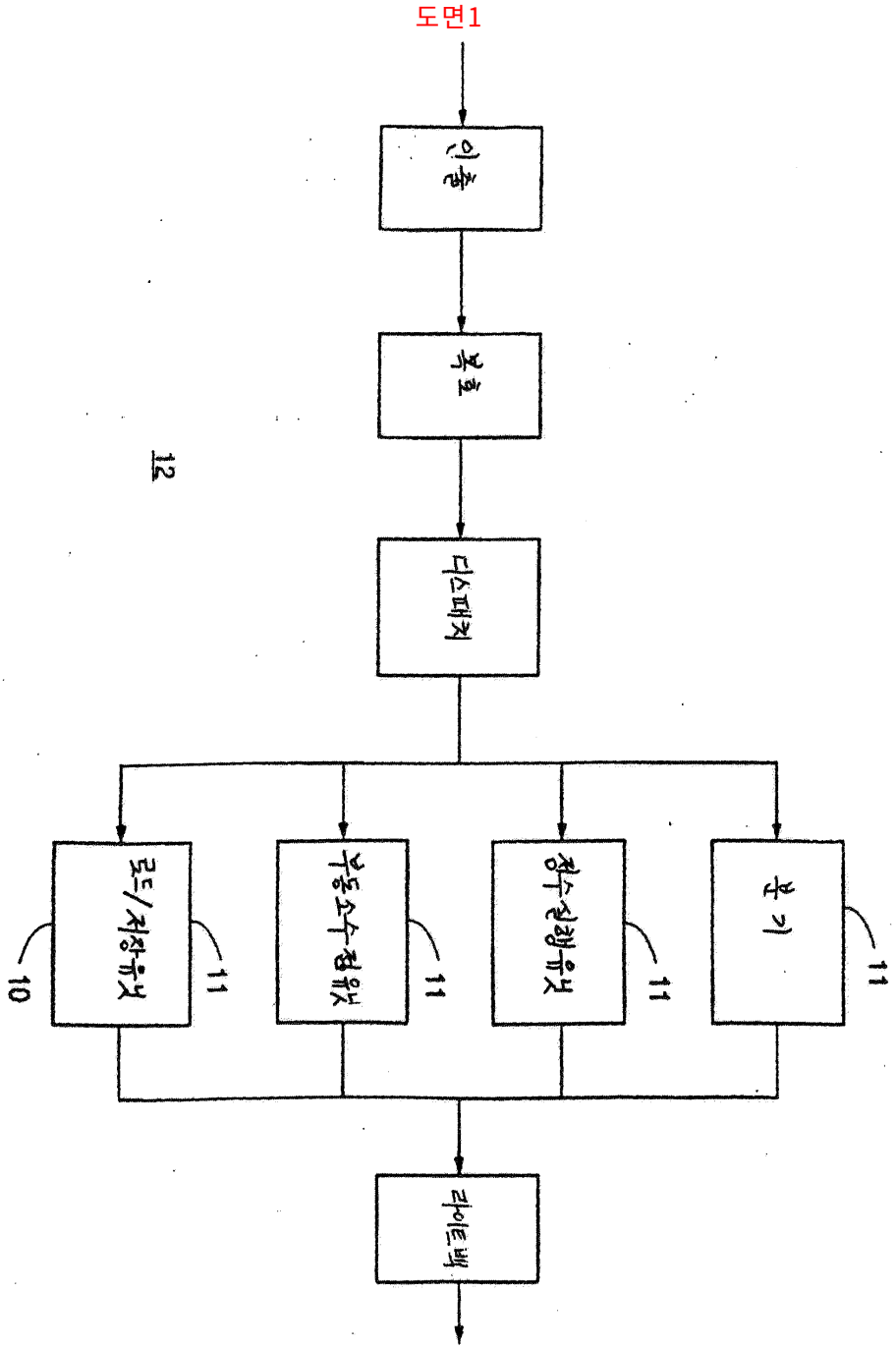
제 3 에 있어서,
상기 로드버퍼 상위 임계값은 프로그램 가능한 것을 특징으로 하는 우선순위 할당 방법.

**청구항 19.**

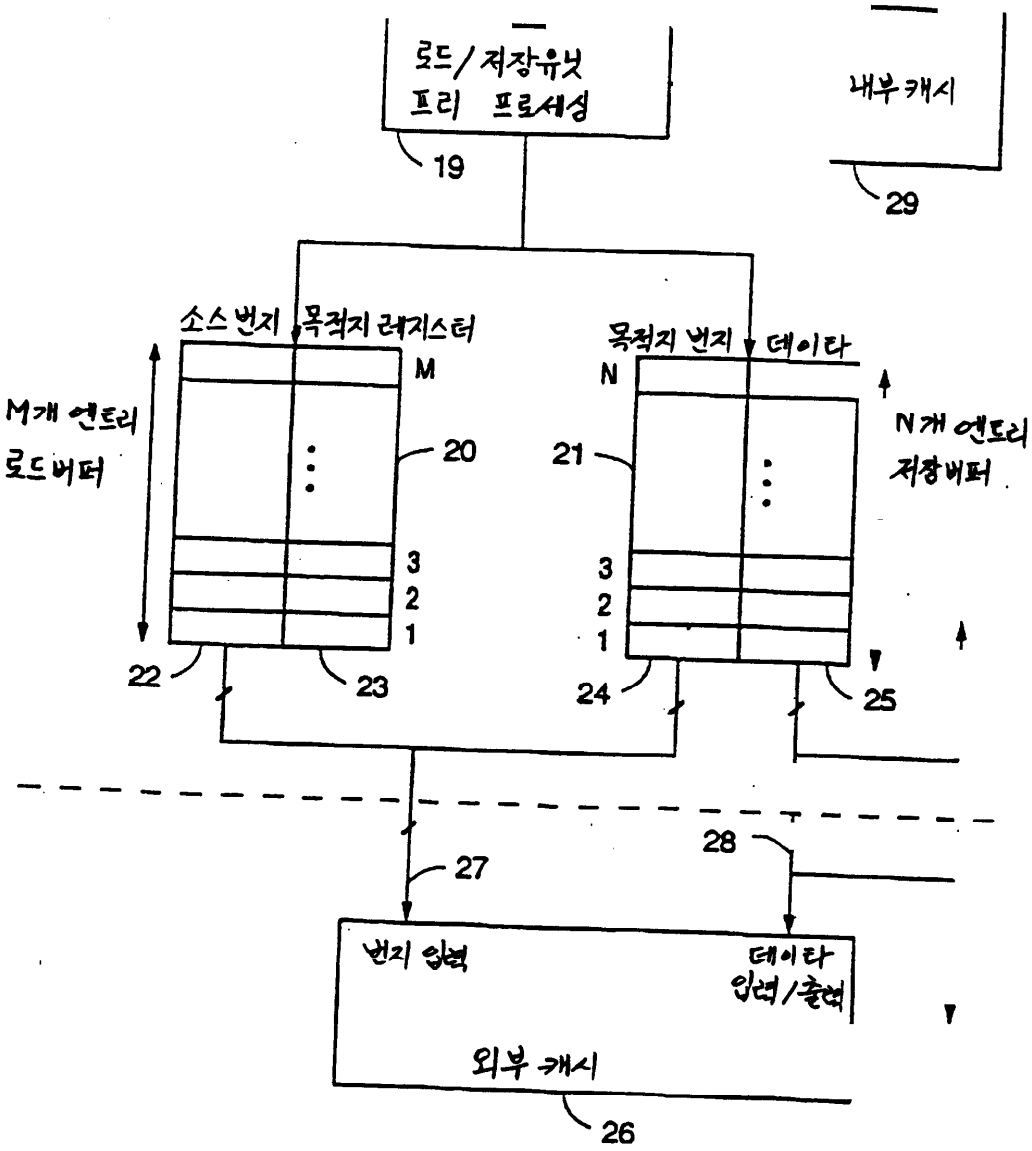
제 4 항에 있어서,
상기 저장버퍼 상위 임계값은 프로그램 가능한 것을 특징으로 하는 우선순위 할당 방법.

도면

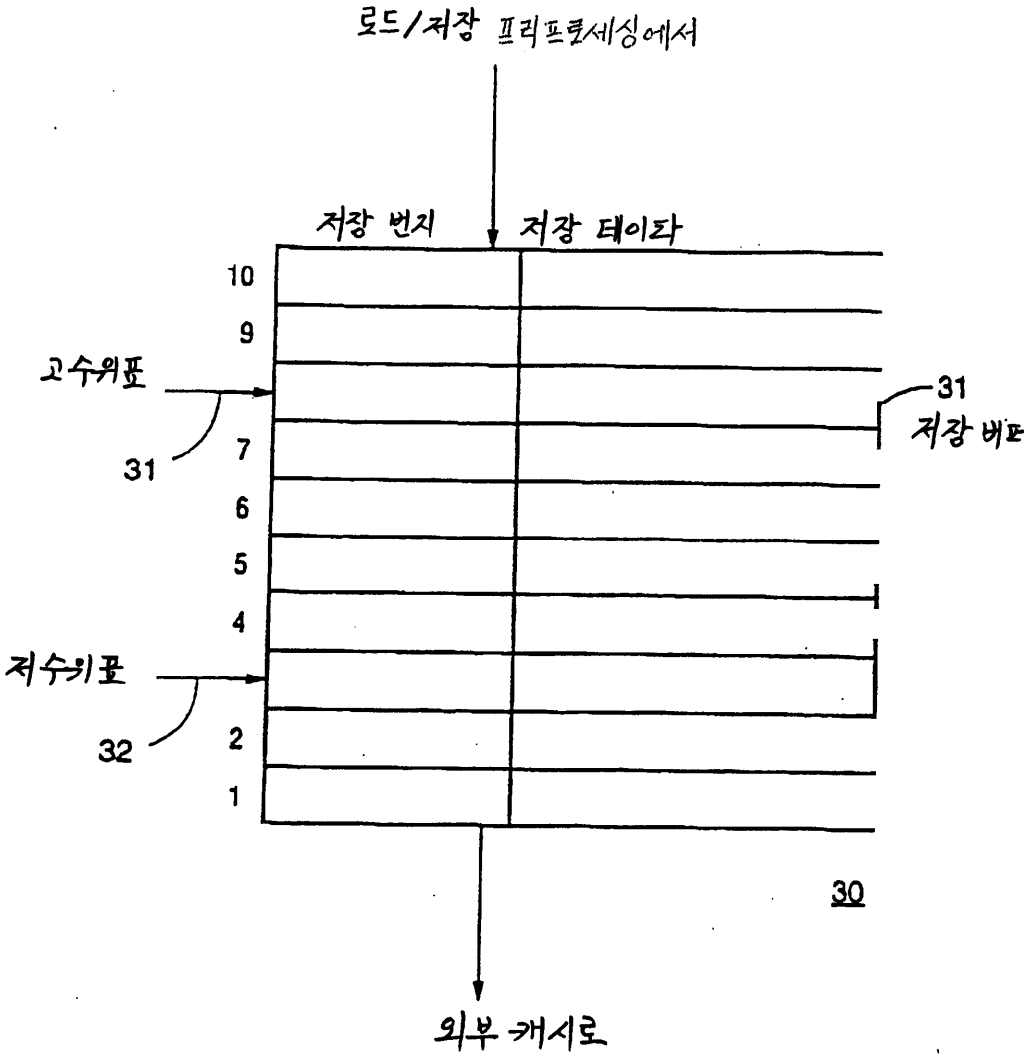
도면1



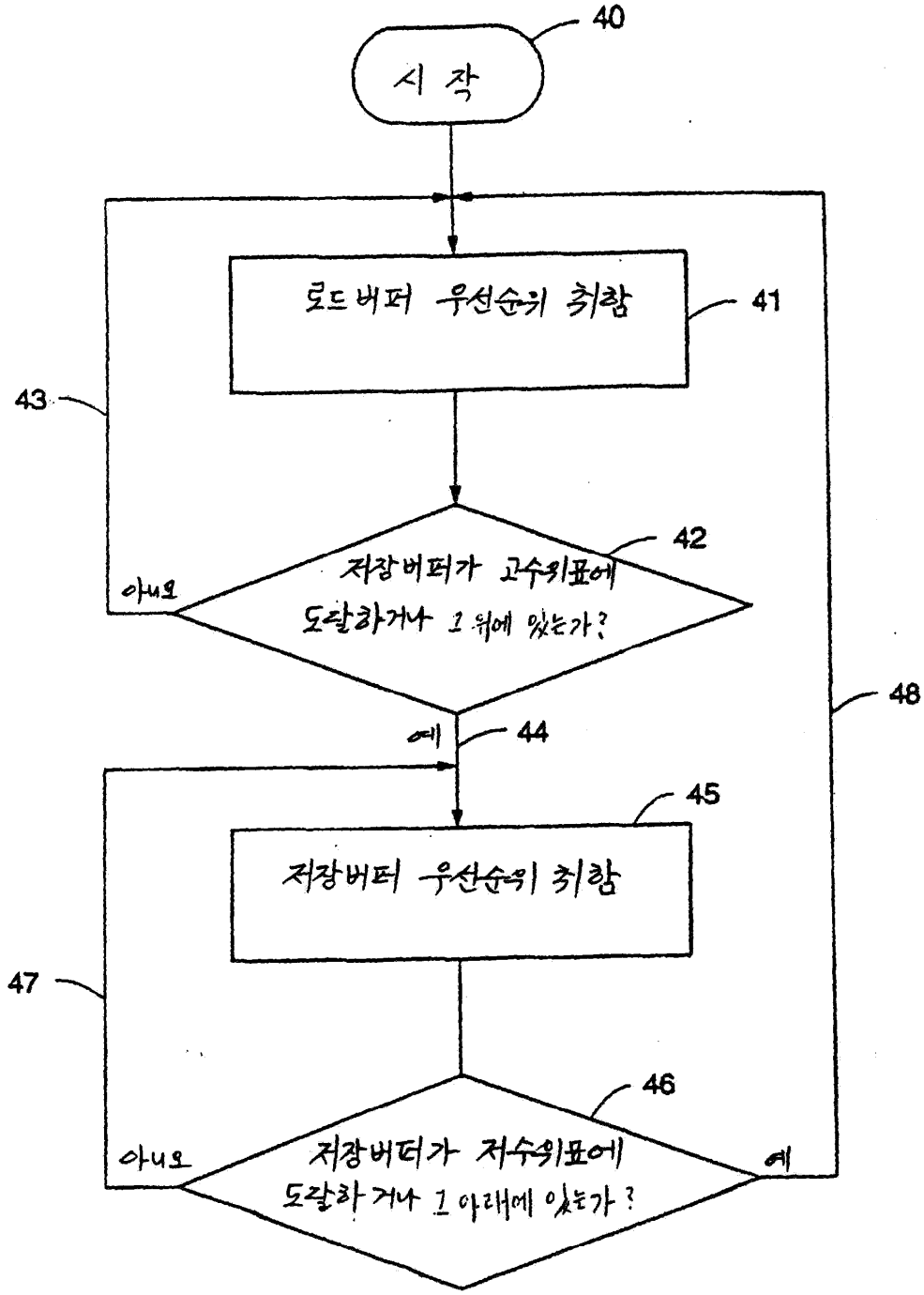
도면2


로드/저장유닛
프리 프로세싱
19
내부캐시
29
소스번지
목적지 레지스터
목적지 번지
데이타
M
N
M개 엔트리
로드버퍼
N개 엔트리
저장버퍼
20
21
3
2
1
22
23
24
25
27
28
번지 입력
데이타
입력/출력
외부 캐시
26

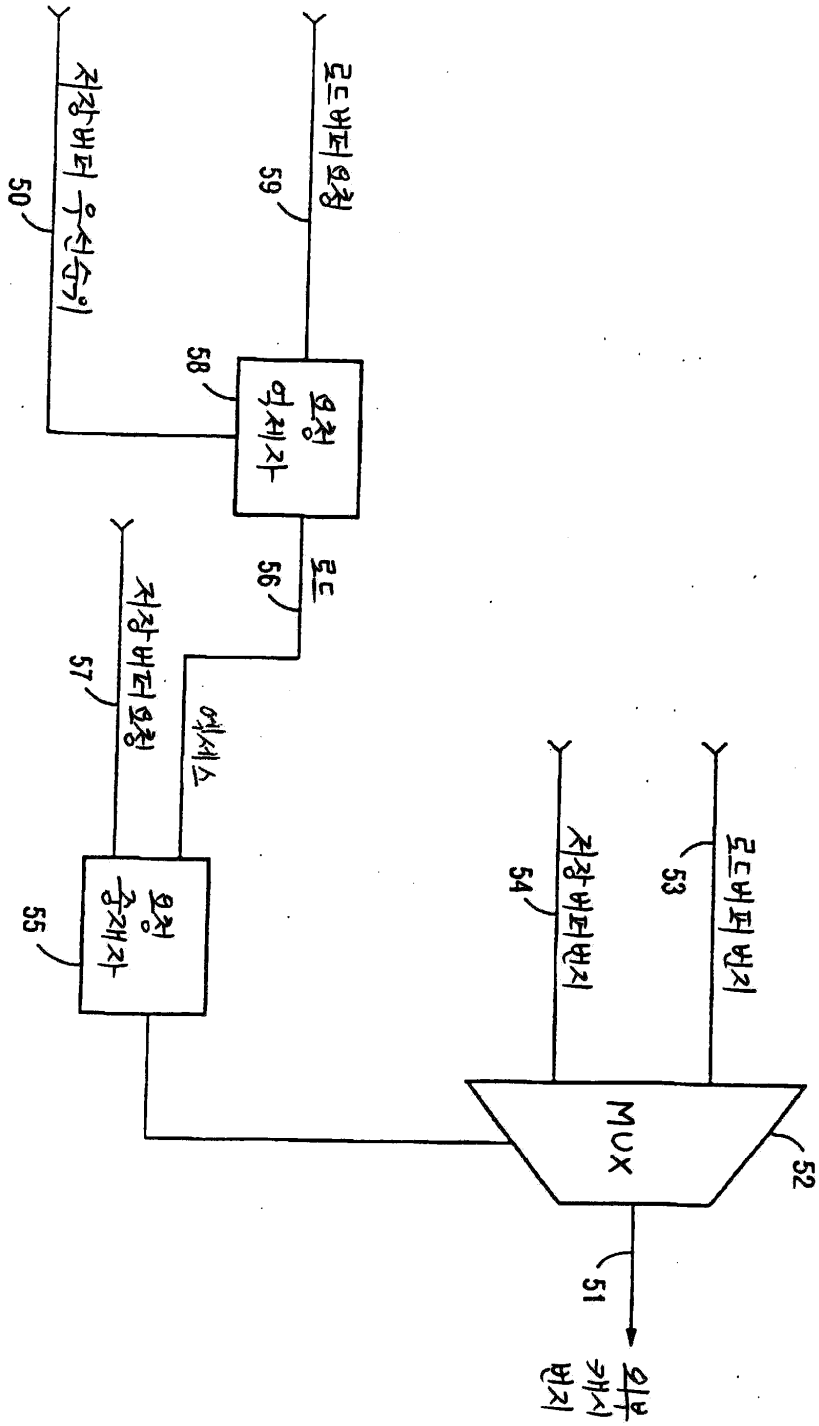
도면3


로드/저장 프리프로세싱에서
저장 번지
저장 데이타
10
9
고수위표
31
7
6
5
4
저수위표
32
2
1
31
저장버퍼
30
외부 캐시로

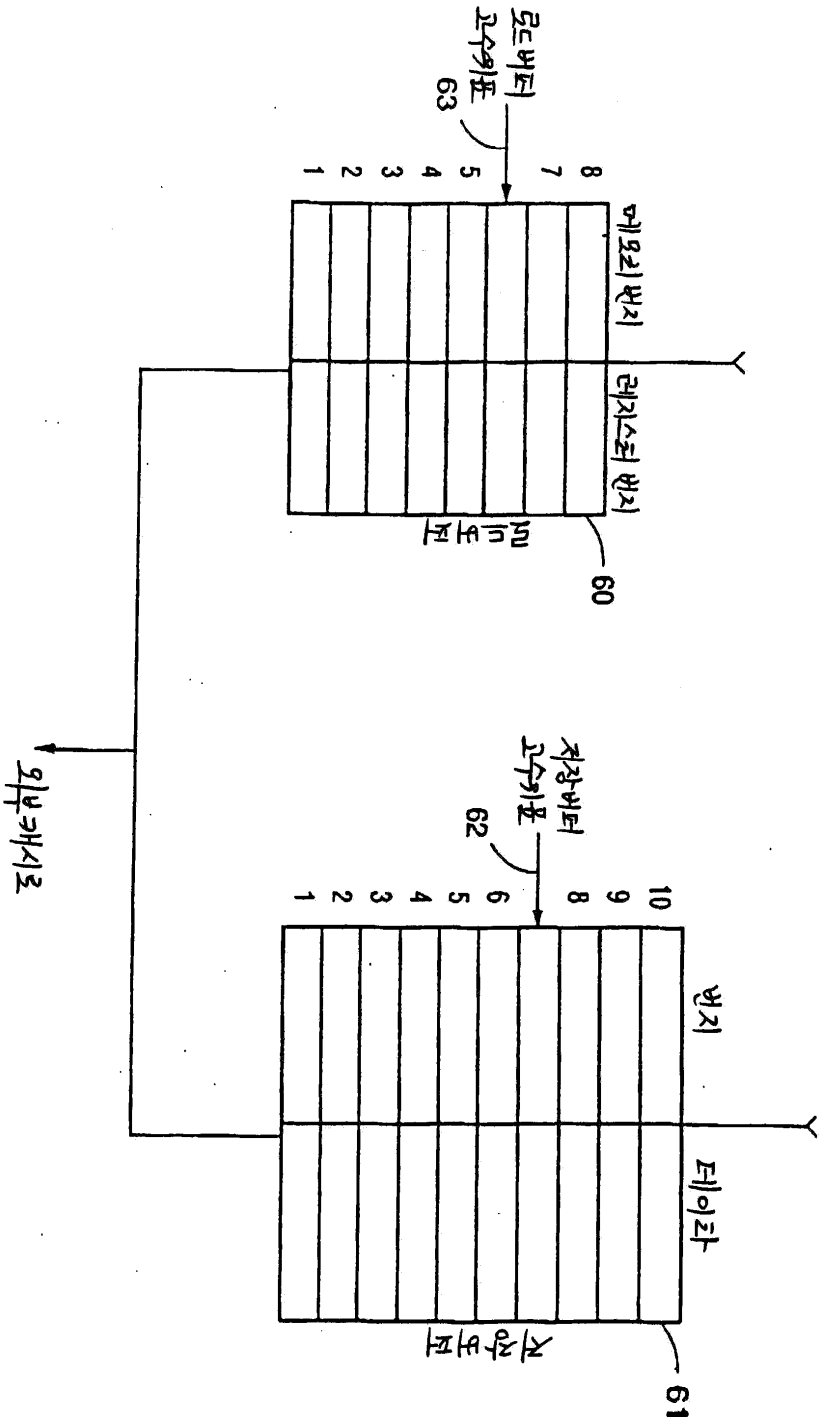
도면4


40
시 작
41
로드버퍼 우선순위 취함
43
42
저장버퍼가 고수위표에
도달하거나 그 위에 있는가?
아니오
예
44
48
45
저장버퍼 우선순위 취함
47
46
저장버퍼가 저수위표에
도달하거나 그 아래에 있는가?
아니오
예

도면5

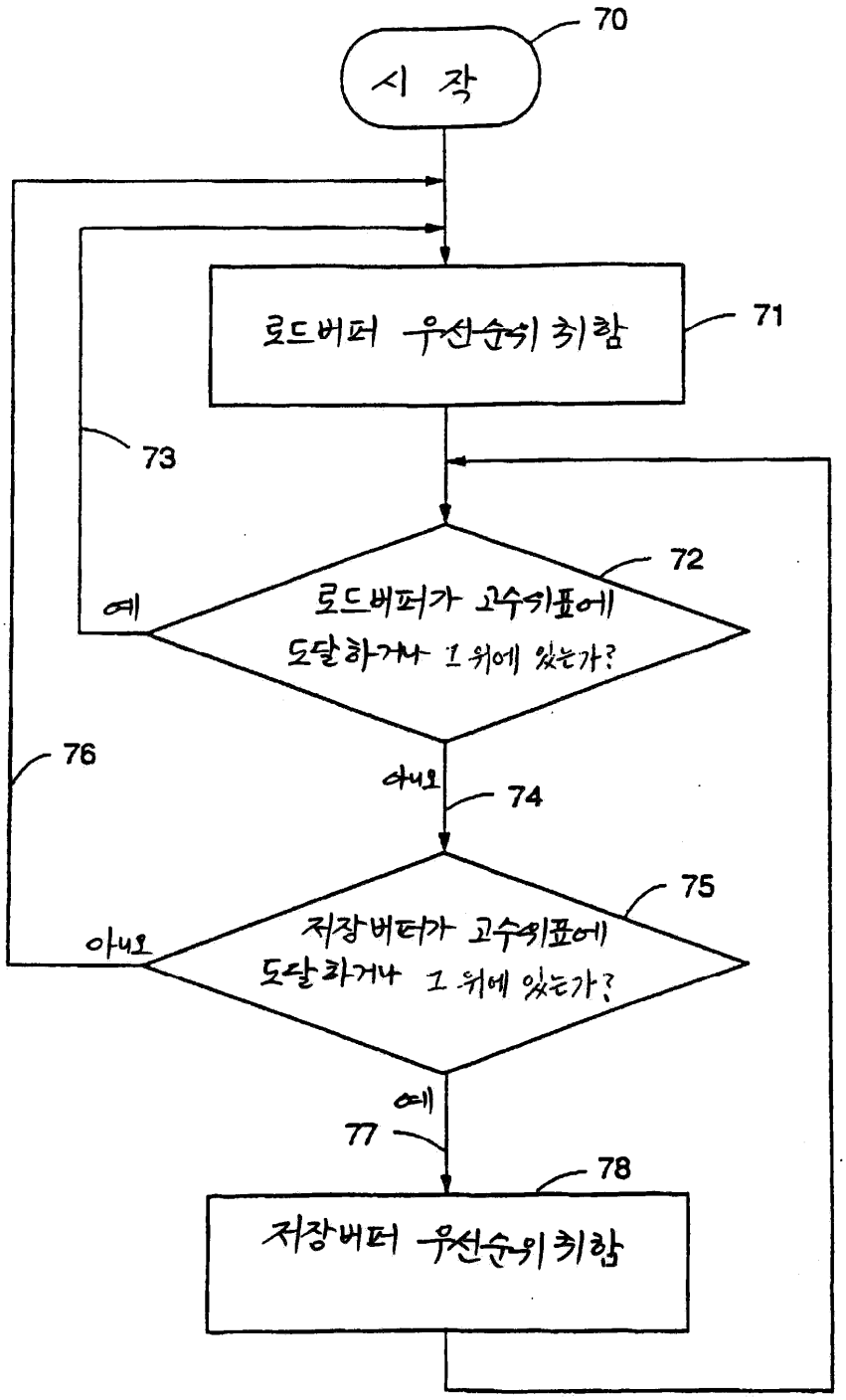

로드버퍼 요청
59
저장버퍼 우선순위
50
요청 억제자
58
로드
56
저장버퍼 요청
57
액세스
요청 중재자
55
저장버퍼번지
54
로드버퍼번지
53
MUX
52
51
외부 캐시 번지

도면6



로드버퍼 고수위표
63
1
2
3
4
5
7
8
메모리 번지
레지스터 번지
로드버퍼
60
외부 캐시로
저장버퍼 고수위표
62
1
2
3
4
5
6
8
9
10
번지
테이타
저장버퍼
61

도면7


70
시 작
71
로드버퍼 우선순위 취함
73
72
예
로드버퍼가 고수위표에
도달하거나 그 위에 있는가?
76
아니오
74
75
아니오
저장버퍼가 고수위표에
도달하거나 그 위에 있는가?
예
77
78
저장버퍼 우선순위 취함