(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) Int. Cl.[6]
G06F 9/34

(11) 공개번호 특2000-0064489
(43) 공개일자 2000년11월06일

(21) 출원번호 10-1998-0704687
(22) 출원일자 1998년06월19일
번역문제출일자 1998년06월19일
(86) 국제출원번호 PCT/US1996/20573 (87) 국제공개번호
(86) 국제출원출원일자 1996년12월17일 (87) 국제공개일자
(81) 지정국 AP ARIPO특허 : 케냐 레소토 말라위 수단 스와질랜드 케냐
EA 유라시아특허 : 아르메니아 아제르바이잔 벨라루스
EP 유럽특허 : 오스트리아 벨기에 스위스 독일 덴마크 스페인 핀랜드 프랑스 영국 그리스 이탈리아 룩셈부르크 모나코 네덜란드 포르투칼 오스트리아 스위스 독일 덴마크 스페인 핀랜드 영국
국내특허 : 아일랜드 알바니아 오스트레일리아 보스니아-헤르체고비나 바베이도스 불가리아 브라질 캐나다 중국 쿠바 체코 에스토니아 그루지야 헝가리 이스라엘 아이슬란드 일본
(30) 우선권주장 8/574,500 1995년12월19일 미국(US)
(71) 출원인 인텔 코오퍼레이션 피터 엔. 데트킨
미합중국 캘리포니아 산타클라라 미션 칼리지 블러바드 2200
(72) 발명자 글류 앤드류 에프.
미국 오리건 97124 힐스보로 노스이스트 캐스린 825
바칼라가다 라마모한 알.
미국 캘리포니아 94539 프레몬트 우드뷰 테라스 459
린 데릭
미국 캘리포니아 94404 포스터 시티 바켄틴 스트리트 113
멘네메이어 래리 엠.
미국 캘리포니아 95006 볼더 크리크 피. 오. 박스587
펠렉 알렉산더 디.
이스라엘 하이파 카멜리아 한나 스트리트 38
비스트리 데이비드
미국 캘리포니아 95014 쿠퍼티노 파크우드 드라이브#6 10243
미탈 밀린드
미국 캘리포니아 94080 사우스 샌프란시스코 힐사이드 불러바드 1149
듀롱 캐로리
미국 캘리포니아 95070 사라토우거 할리프 드라이브18983
고와시 에이이치
일본 301 이바라키켄 류가사키시 구보다이 2-8-10
에이탄 베니
이스라엘 하이파 스테판 와이즈 25
(74) 대리인 장용식

심사청구 : 있음

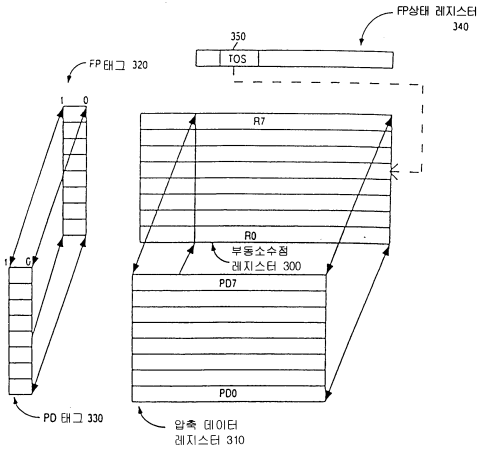
(54) 여러운영체제기술에서볼수없는상이한데이터형연산을수행하는방법

요약

여러 운영 체제 기술에서 볼 수 없도록 프로세서(505)가 상이한 데이터 타입을 수행하게 하는 상이한 명령 세트를 실행하는 방법. 본 발명의 일실시예에 따라서, 데이터 처리 장치(505)는 단일 논리 레지스터 파일(300,310)로서 소프트웨어에 최소한 논리적으로 나타나는 하나 이상의 물리 레지스터 파일을 이용하

여 제 1 데이터 타입의 제 1 명령 세트와 제 2 데이터 타입의 제 1 명령을 모두 실행한다. 제 1 명령 세트를 실행하는 동안에, 단일 논리 레지스터 파일(300,310)은 플랫 레지스터 파일로서 동작된다. 제 2 데이터 타입의 제 1 명령을 실행하는 동안에, 단일 논리 레지스터 파일(300,310)은 스택 레퍼런스(340) 레지스터 파일로서 동작된다. 더욱이, 데이터 처리 장치는 논리 레지스터 파일(300,310)에 대응하는 태그 세트(320,330)내의 모든 태그를, 제 1 명령 세트의 실행을 시작하는 때와 제 1 명령의 실행을 완료하는 때사이에서 비공백 상태로 바꾼다.

대표도



명세서

기술분야

본 발명은 컴퓨터 시스템 분야에 관한 것이고, 더 상세하게는 프로세서에 의한 압축 데이터 명령과 부동점 실행에 관한 것이다.

배경기술

전형적인 컴퓨터 시스템에 있어서, 하나이상의 프로세서는 다수의 비트(예, 16,32,64 등)로 표시된 데이터 값으로 연산하고, 프로그램 명령에 응답하여 결과를 만든다. 예를 들어, 가산 명령의 실행은 제 1 데이터 값과 제 2 데이터 값을 가산하고, 제3데이터 값으로서 그 결과를 저장한다. 그러나, 멀티미디어 애플리케이션(예, 컴퓨터 지원 협동에 목표를 둔 애플리케이션(CSC--혼합 미디어 데이터 조작으로 원격 전자 회의하는 집적체), 2D/3D 그래픽, 영상 처리, 비디오 압축/압축 해제, 인식 알고리즘, 및 오디어 조작)은 다수의 비트로 종종 표시되어 있는 다량의 데이터 조작을 필요로 한다. 예를 들어, 멀티미디어 데이터는 전형적으로 64 비트수로 표시되지만, 소수의 비트만이 유효 정보를 운반할 수 있다.

멀티미디어 애플리케이션(같은 특성을 가진 다른 애플리케이션뿐만 아니라)의 효율성을 향상시키기 위해, 종래의 프로세서는 압축 데이터 포맷을 제공한다. 압축 데이터 포맷은, 단일 값을 표시하는데 사용된 비트가 각각이 개별 값을 표시하는 다수의 고정 사이즈의 데이터 요소로 나뉘어 지는 것이다. 예를 들어, 64 비트 레지스터내의 데이터는 각각이 개별 32 비트값을 표시하는 두 개의 32 비트 요소로 나뉘어 질 수 있다.

휼렛패커드의 기본 32 비트 아키텍처 머신은, 구현하는 멀티 미디어 데이터 타입에서 이러한 접근법을 얻었다. 즉 그 프로세서는 64 비트 데이터 타입을 구현하기 위해 병렬로 32 비트의 일반적인 정수 레지스터를 이용했다. 이러한 단순한 접근법의 주 결점은 이용가능한 레지스터 영역을 엄격히 제한한다는 것이다. 추가로, 현재의 아키텍처를 확장시키는 데 필요한 노력면에서, 이러한 방식으로 멀티 미디어 데이터로 연산하는 수행 잇점은 최소한도로 고려된다.

모토롤라® 88110™ 프로세서에 채택된 어느 정도 유사한 접근법은 정수 레지스터를 쌍으로 조합시키는 것이다. 32 비트 레지스터를 쌍으로 하는 생각은 단일 연산 또는 명령에 대한 특정 레지스터의 랜덤 조합을 연결시키는 것을 포함하고 있다. 다시 한번, 그러나 쌍의 레지스터를 이용한 64 비트 멀티 미디어 타입을 구현하는 것의 주 단점은 이용할 수 있는 레지스터 쌍의 수가 제한되어 있다는 것이다. 추가 레지스터 공간을 아키텍처에 가산하는 것을 제외하고, 멀티 미디어 타입을 구현하는 다른 기술이 필요하다.

큰 소프트 웨어와 하드웨어 베이스를 가진 프로세서의 한 라인은, 캘리포니아 산타 클라라 소재 인텔 코포레이션에 의해 제조된, 펜티엄® 프로세서를 포함하여, 인텔 아키텍처 패밀리의 프로세서이다. 도1은 펜티엄 프로세서가 사용되는 전형적인 컴퓨터 시스템(100)을 설명하는 블록도를 도시하고 있다. 여기에 제시된 것보다 더 상세한 펜티엄 프로세서의 설명에 대해서는, 캘리포니아 산타 클라라 소재의 인텔 코포레이션 1994 판, 펜티엄 프로세서의 사용자 매뉴얼 - 권3 : 아키텍처 및 프로그래밍 매뉴얼을 참조하라. 전형적인 컴퓨터 시스템(100)은 프로세서(105), 기억 장치(110), 및 버스(115)를 포함하고 있다. 프로세서(105)는 버스(115)에 의해 기억 장치(110)에 연결되어 있다. 추가로, 키보드(120)와 디스플레이(125)와 같은 다수의 사용자 입력/출력 장치는 버스(115)에 또한 연결되어 있다. 네트워크(130)는 또한 버스(115)에 연결될 수 있다. 프로세서(105)는 펜티엄 프로세서를 나타낸다. 기억 장치(110)는 데이터를 저장하는 하나이상의 메카니즘을 나타낸다. 예를 들어, 기억 장치(110)는 판독 전용 메모리(ROM), 임의 접근 메

모리(RAM), 자기 디스크 기억 매체, 광학 기억 매체, 플래시 메모리 장치, 및/또는 다른 기계 판독가능한 매체를 포함할 수 있다. 버스(115)는 하나 이상의 버스(예, PCI, ISA, X-버스, EISA, VESA 등)와 브리지(버스 제어기로도 부름)를 나타낸다.

도1은 기억 장치(110)가 프로세서(105)상의 실행을 위해 그 안에 운영 체제(OS)(132)을 저장하고 있는 것을 설명하고 있다. 물론, 기억 장치(110)는 바람직하게 추가 소프트웨어(도시 생략)를 포함하고 있다. 도1은 프로세서(105)가 부동점 유닛(135)과 부동점 상태 레지스터(155)(여기서, 부호'FP'는 부동점 용어)를 포함하고 있는 것을 추가로 설명하고 있다. 물론, 프로세서(105)는 본 발명을 이해하는 데 필요가 없는 추가 회로 소자를 포함하고 있다.

부동점 유닛(135)은 부동점 데이터를 저장하기 위해 사용되고, 부동점 레지스터 세트(부동점 레지스터 파일로도 부름)(145), 태그 한 세트(150), 및 부동점 상태 레지스터(155)를 포함하고 있다. 부동점 레지스터 세트(145)는 R0 에서 R7로 표시된 8개의 레지스터를 포함하고 있다(여기서, 부호 Rn 은 부동점 레지스터의 물리적인 위치). 8개의 레지스터 각각은 80 비트 폭이고, 부호 필드(비트 79), 지수 필드(비트[78:64]), 및 소수부 필드(비트[63:0])를 포함하고 있다. 부동점 유닛(135)은 스택으로서 부동점 레지스터(145)를 연산시킨다. 환언하면, 부동점 유닛(135)은 스택 기준 레지스터 파일을 포함하고 있다. 레지스터 세트가 스택으로서 연산할 때, 부동점 레지스터 세트(145)(여기서, 부호 STn은 스택의 상부에서의 논리 부동점 레지스터(n)의 상대적인 위치)내의 레지스터의 물리적인 위치보다는, 스택의 상부를 기준으로 연산이 수행된다. 부동점 상태 레지스터(155)는 부동점 레지스터(145) 세트내의 어느 레지스터가 현재 부동점 스택의 상부에 있는지를 식별하는 스택 필드(160)의 상부를 포함하고 있다. 도1에서, 스택 표시의 상부는 스택의 상부로서 물리적인 위치(R4)에서 레지스터(165)를 식별한다.

태그 세트(150)는 8개의 태그를 포함하고 있고, 단일 레지스터로 저장되어 있다. 각각의 태그는 상이한 부동점 레지스터에 대응하고, 두 개의 비트를 구성하고 있다. 도1에 도시된 바와 같이, 태그(170)는 레지스터(165)에 대응한다. 태그는 태그에 대응하는 부동점 레지스터의 전류 크기에 관련된 정보를 식별한다 -- 00 = 유효; 01 = 제로; 10 = 특수; 및 11 = 공백. 공백과 비공백 레지스터 위치사이를 구별하기 위해 부동점 유닛(135)은 이러한 태그를 사용한다. 따라서, 태그는 두 가지 상태: 11에 의해 표시된 공백과, 00, 01, 10중 어느 하나에 의해 표시된 비공백을 식별하는 것으로서 생각될 수 있다.

이러한 태그는 이벤트(event)를 제공하는데 사용될 수 있다. ' 이벤트'는 하드웨어 인터럽트, 소프트웨어 인터럽트, 예외, 고장, 기계 체크, 어시스트, 및 디버그 이벤트를 포함하여, 컴퓨터가 응답하는 어떠한 연산이나 어커런스이다. 이벤트를 받았을 때, 프로세서의 이벤트 조정 메카니즘은 프로세서가 현재의 프로세스의 실행을 인터럽트하게 하고, 그 인터럽트된 프로세스의 실행 환경을 저장하게 하고, 그 이벤트를 제공하는 적절한 이벤트 핸들러를 시동하게 한다. 이벤트를 제공한 후, 이벤트 핸들러는 프로세서가 앞서 저장된 프로세스의 실행 환경을 이용하여 그 인터럽트된 프로세스를 재개시하게 한다. 이벤트 핸들러의 프로그래머는 이벤트를 더 제공하기 위해, 이러한 태그를 사용하여 상이한 부동점 레지스터의 내용을 체크할 수 있다.

태그의 각각은 2 비트를 포함하고 있는 것처럼 설명되었지만, 대체 실시예는 각각의 태그에 대하여 1 비트만을 저장할 수 있다. 이러한 1 비트 태그의 각각은 공백인지 미공백인지를 식별한다. 이러한 실시예에서, 이러한 1 비트 태그는 태그 값이 필요할 때 적절한 2 비트값을 결정함으로써 2 비트를 구성하고 있는 것처럼 사용자에게 보여질 수 있다.

상태 레지스터(140)는 각각 EM 지시와 TS 지시를 저장하는 EM 필드(175)와 TS 필드(180)를 포함하고 있다. EM 지시가 1이고, 또는 TS 지시가 1이면, 프로세서 하드웨어는 '디바이스 이용불가능' 예외를 발생시킴으로써 부동점 지시의 실행으로 운영 체제에 트랙을 야기시킨다. 소프트웨어 규정에 따라서, EM 지시와 TS지시는 각각 부동점 명령을 에뮬레이트하는 데, 그리고 다중 작업을 이행하는데 사용된다. 그러나. 이러한 지시의 사용은 단순히 소프트웨어 규정이다. 따라서, 지시 모두나 어느 하나는 어느 한 목적을 위해 사용될 수 있다. 예를 들어, EM 지시는 다중 작업을 이행하는 데 사용될 수 있다.

상기 소프트웨어 규정에 따라서, EM 필드(175)는 부동점 유닛이 소프트웨어를 이용하여 에뮬레이트되어야 하는지를 식별하는 부동점 에뮬레이트 지시('EM 지시')를 저장하는데 사용된다. 부동점 유닛이 존재하는지를 판단하기 위해, 그리고 필요하다면 EM 지시를 바꾸기 위해, 시스템이 부트되었을 때 일련의 지시 또는 단일 지시(예, CPUID)가 전형적으로 실행된다. 따라서, 한가지 이행에서, 부동점 유닛이 에뮬레이트되어야 할 때 EM 지시는 1인 반면, 대체 이행은 다른 값을 사용할 수 있다.

이 운영체제의 사용에도 불구하고, 다수의 프로세서는 협동 다중 작업, 타임 슬라이스 다중 작업등의 기술을 이용하여 일부 프로세스(이하, 태스크)를 다중 작업할 수 있다. 하나의 프로세서는 동시에 하나의 태스크만을 수행할 수 있기 때문에, 프로세서는 여러 태스크사이의 처리 시간을 여러 태스크사이의 스위칭 시간으로 나누어야 한다. 프로세서가 하나의 태스크에서 다른 태스크로 스위칭할 때, 태스크 스위치('콘텍트(context) 스위치' 또는 '프로세스 스위치' 라고도 함)는 발생된 것으로 알려졌다. 태스크 스위치를 수행하기 위해, 프로세서는 하나의 태스크 실행을 중지하여야 하고 다른 태스크의 실행을 재개시하거나 시작하여야 한다. 태스크 실행후 태스크의 실행을 재개시하도록 내용이 보존되어야 하는 다수의 레지스터(부동점 레지스터를 포함)가 있다. 태스크의 실행동안의 소정 시간에의 이러한 레지스터의 내용은 태스크의 '레지스터 상태'로서 불리운다. 일부 프로세스를 다중 작업하는 동안에, 태스크의 '레지스터 상태'는 프로세서 외부의 메모리에 포함되어 있는 데이터 구조('콘텍트 구조'라 함)로 저장함으로써 다른 프로세서의 실행동안에 보존된다. 하나의 태스크의 실행이 재개시되었을 때, 태스크의 레지스터 상태는 태스크의 콘텍트 구조를 이용하여 재저장된다(예, 프로세서로의 역올림).

태스크의 레지스터 상태 보존 및 재저장은 다수의 상이한 기술을 이용하여 실행될 수 있다. 예를 들어, 하나의 운영 체제는 이전 태스크의 전체 레지스터 상태를 저장하고, 각각의 테스크 스위치상에서 다음 태스크의 전체 레지스터를 재저장한다. 그러나, 전체 레지스터 상태를 저장하고 재저장하는 데에는 시간이 소모되 기 때문에, 태스크 스위치동안에 불필요한 부분을 저장 및/또는 재저장하는 것을 피하는 것이 바람직하다. 태스크가 부동점 유닛을 사용하지 않는다면, 태스크의 레지스터 상태의 부분으로서 부동점 레지스터의 내용을 저장하고 재저장하는 것이 불필요하다. 결론적으로, 앞서 설명된 소프트웨어 규정에 따

라서, 태스크 스위치(일반적으로, '부분 콘텍트 스위칭' 또는 '요구 콘텍트 스위칭' 라 함)동안에 부동점 레지스터의 내용을 저장하고 재저장하는 것을 피하기 위해, TS 지시는 운영 체제에 의해 종래에 사용되어 왔다.

부분 콘텍트 스위칭을 이행하는 TS 지시의 사용이 잘 알려져 있다. 그러나, 본 발명을 위해, TS 지시가 부분 콘텍트 스위치를 지시하는 동안의 부동점 명령의 시도 실행이 수행되었고(즉, 그 부동점 유닛은 '이용불가능' 또는 '무력화'된다), 그 결과 '디바이스 이용불가능' 예외로 된다. 이러한 예외에 반응하여, 이벤트 핸들러는 프로세서상에서 실행하여, 현재의 태스크가 부동점 유닛의 오너인지를(부동점 유닛에 저장된 데이터가 현재의 태스크에 속하는지 앞서 실행된 태스크에 속하는지를) 판단한다. 현재의 태스크가 오너가 아니다면, 이벤트 핸들러는 프로세서가 부동점 레지스터의 내용을 이전 태스크의 콘텍트 구조로 저장하게 하고, 현 태스크의 부동점 상태(이용가능하다면)를 저장하게 하고, 현재의 태스크를 오너로서 인식한다. 그러나. 현재의 태스크가 부동점 유닛의 오너이면, 현재의 태스크는 부동점 유닛을 이용하는 최종 태스크이었고(현 태스크의 레지스터 상태는 부동점 유닛에 이미 저장되어 있다), 부동점 유닛에 관한 연산은 일어날 필요가 없고, TS은 설정되지 않을 것이고, 어떠한 예외도 일어나지 않을 것이다. 또한 핸들러의 실행은 프로세서가 현재의 태스크에 의해 점유된 부동점 유닛을 지시하는 TS 지시를 바꾸게 한다('이용가능' 또는 '가능상태'라고 함).

이벤트 핸들러의 완료와 동시에, 현 태스크의 실행은 디바이스가 예외를 이용할 수 없게 하는 부동점 명령을 재시작함으로써 재개시된다. TS 지시는 부동점 유닛이 이용가능한 지시로 바뀌었기 때문에, 다음 부동점 명령의 실행은 추가 디바이스가 예외를 이용할 수 없게 하지는 않을 것이다. 그러나, 다음 부분 콘텍트 스위치동안에, TS 지시는 부분 콘텍트 스위치가 수행되었음을 지시하는 것으로 바뀌게 된다. 따라서, 다른 부동점 명령의 실행이 시도되었을 때, 그리고 시도된다면, 예외를 이용할 수 없는 다른 디바이스가 발생될 것이고, 이벤트 핸들러는 다시 실행될 것이다. 이러한 방식으로, TS 지시는 운영 체제가 부동점 레지스터 파일의 보관과 로딩을 딜레디하게 하고, 가능한 한 피하게 한다. 그렇게 함으로써, 태스크 스위치 오버헤드는 보관되고 로드되어야 하는 레지스터의 수를 줄임으로써 감소된다.

부동점 상태가 태스크 스위치동안 저장되지 않거나 재저장되지 않은 하나의 운영 체제가 설명되어 있지만, 대체 실시예는 임의 수의 다른 기술을 이용할 수 있다. 예를 들어, 앞서 설명된 바와 같이, 운영 체제는 각각의 태스크 스위치에서 전체 레지스터 상태를 항상 저장하고 재저장하도록 이행될 수 있다.

프로세스의 부동점 상태가 저장될 수 있는 상이한 시간에 추가로(예, 이벤트를 이용할 수 없는 디바이스에 응답하여 콘텍트 스위치하는 동안에), 부동점 상태를 저장하는 상이한 기술이 또한 있다. 예를 들어, 운영 체제는 전체 부동점 상태('단일 태스크 스위치'라고 함)를 저장하도록 이행될 수 있다. 대안으로, 운영 체제는, 대응 태그가 비공백 상태('최소 태스크 스위치'라고 함)를 지시하는 부동점 레지스터만의 내용을 저장하도록 이행될 수 있다. 그렇게 행할 때, 운영 체제는 유용한 데이터를 포함하고 있는 부동점 레지스터만의 내용을 저장하고 있다. 이러한 방식으로, 부동점 상태를 저장하는 오버헤드는 보존되어야 하는 레지스터의 수를 줄임으로써 감소될 수 있다.

도2는 펜티엄 프로세서에 의한 명령 실행을 설명하는 흐름도이다. 이 흐름도는 단계(200)에서 시작하여; 단계(205)로 진행한다.

단계(205)에서 보는 바와 같이, 비트 세트는 명령으로서 액세스되고 단계(210)으로 진행한다. 이 비트 세트는 그 명령에 의해 수행될 연산을 식별하는 연산 코드를 포함하고 있다.

단계(210)에서, 이 연산 코드가 유효한지를 판단한다. 이 연산이 유효하지 않다면, 단계(215)로 진행한다. 그 반대면, 단계(220)로 진행한다.

단계(215)에서 보는 바와 같이, 유효하지 않은 연산 코드의 예외가 발생되고, 적절한 이벤트 핸들러가 실행된다. 이 이벤트 핸들러는 프로세서가 메시지를 디스플레이하게 하고, 현 태스크의 실행을 중단시키고, 다른 태스크를 실행하기 위해 진행하게 하도록 구현될 수 있다. 물론, 대체 실시예는 이러한 이벤트 핸들러를 여러 방법으로 구현될 수 있다.

단계(220)에서, 명령이 부동점 명령인지를 판단한다. 명령이 부동점 명령이 아니다면, 단계(225)로 진행한다. 그 반대면, 단계(230)로 진행한다.

단계(225)에서 보는 바와 같이, 프로세서는 그 명령을 실행한다. 이 단계는 본 발명을 설명하는데 필요하지 않기 때문에, 추가로 설명하지 않는다.

단계(230)에서 보는 바와 같이, EM 지시가 1과 같은지를(상기된 소프트웨어 규정에 따라, 부동점 유닛이 에뮬레이트되어야 하는지를), 그리고 TS 지시가 1과 같은지를(상기 소프트웨어 규정에 따라, 부분 콘텍트 스위치가 수행되었는지를) 판단한다. EM 지시 및/또는 TS 지시가 1과 같다면, 단계(235)로 진행한다. 그 반대면, 단계(240)로 진행한다.

단계(235)에서, '디바이스 이용 불가능' 예외가 발생되고, 대응 이벤트 핸들러가 실행된다. 이 이벤트에 반응하여, 그 대응 이벤트 핸들러는 EM 지시와 TS 지시를 폴링하도록 구현될 수 있다. EM 지시가 1과 같다면, 이벤트 핸들러는 프로세서가 부동점 유닛을 에뮬레이트하여 그 명령을 실행하게 하고, 다음 명령(단계(205)에서 수신된 명령 다음에 논리적으로 뒤따르는 명령)으로 실행을 재개시하게 하도록 구현될 수 있다. TS 지시가 1과 같다면, 이벤트 핸들러는 부분 콘텍트 스위치(부동점 유닛의 내용을 저장하고, 필요하다면 올바른 부동점 상태를 재저장하기 위해)를 기준으로 앞서 설명된 바와 같이 작용하도록, 그리고 프로세서가 단계(205)에서 수신된 명령의 실행을 재시작함으로써 실행을 재개시하도록 구현될 수 있다. 물론, 대체 실시예는 이러한 이벤트 핸들러를 여러 방법으로 구현할 수 있다.

특정 숫자 오류가 부동점 명령의 실행동안에 발생된다면, 그 오류는 그 실행의 다음 부동점 명령의 시도된 실행이 그 보류한 부동점 숫자 오류에 도움이 되도록 인터럽트될 수 있을 때까지 보류한다.

단계(240)에서 보는 바와 같이, 이러한 보류 오류가 있는지를 판단한다. 이러한 보류 오류가 있다면, 단계(245)로 진행한다. 그렇지 않으면, 단계(250)로 진행한다.

단계(245)에서, 보류한 부동점 오류 이벤트가 발생된다. 이러한 이벤트에 반응하여, 프로세서는 부동점 오류가 마스크되는지를 판단한다. 그러하다면, 프로세서는 마이크로코드를 내부적으로 이용하여 그 이벤트 조정을 시도하고, 부동점 명령은 '마이크로 재시작' 상태이다. 마이크로 재시작 용어는 비마이크로코드 핸들러를 실행함이 없이 이벤트를 제공하는 기술에서 적용된다(운영 체제 이벤트 핸들러라고도 함). 이러한 이벤트는 그 이벤트가 프로세서에 의해 내부적으로 조정되어 외부 운영 체제 시스템 핸들러의 실행을 필요로 하지 않기 때문에, 이러한 이벤트는 내부 이벤트라고 불리운다(운영 체제 이벤트 핸들러라고도 함). 반대로, 부동점 오류가 마스크되지 않으면, 그 이벤트는 외부 이벤트('소프트웨어 가시 이벤트'라고 함)이고, 그 이벤트의 대응 이벤트 핸들러가 실행된다. 이 이벤트 핸들러는 그 오류에 적합하도록, 그리고 프로세서가 단계(205)에서 수신된 명령의 실행을 재시작함으로써 실행을 재개시하게 하도록 구현될 수 있다. 하나의 명령을 재시작하는 이러한 기술은 '매크로 재시작' 또는 명령 레벨 재시작'으로서 불리운다. 물론, 대체 실시예는 이러한 마이크로코드 이벤트 핸들러를 여러 방식으로 구현할 수 있다.

단계(250)에서 보는 바와 같이, 부동점 명령이 실행된다. 이러한 실행동안에, 태그는 필요한 만큼 바뀌고, 여기에 제공될 수 있는 숫자 오류는 기록되고, 다른 숫자 오류는 보류가 유지된다.

다른 일반적인 프로세서뿐만 아니라, 인텔 아키텍처 프로세서 패밀리(펜티엄 프로세서를 포함)의 한가지 한계는 압축 데이터로 연산하는 명령 세트를 포함하고 있지 않다는 것이다. 따라서, 압축 데이터로 연산하는 명령 세트를 현재 소프트웨어 및 하드웨어와 호환성이 있는 프로세서에 부가시키고자 한다. 더욱이, 압축 데이터 명령 세트를 지원하며, 운영 체제를 포함하여 현재의 소프트웨어와 호환성이 있는 새로운 프로세서를 만들고자 한다.

발명의 개요

본 발명은 여러 운영 체제 기술에서 볼 수 없도록 프로세서(505)가 상이한 데이터 타입을 수행하게 하는 상이한 명령 세트를 실행하고 좋은 프로그래밍 실행을 진전시키고, 그리고 현재의 소프트웨어 규정에서 볼 수 없는 방법을 제공한다.본 발명의 일측면에 따라서, 데이터 처리 장치는 단일 논리 레지스터 파일로서 소프트웨어에 최소한 논리적으로 나타나는 제 1 명령 타입의 제 1 명령 세트를 실행한다. 데이터 처리 장치는 제 1 명령 세트를 실행하고 있지만, 단일 논리 레지스터 파일은 플랫 레지스터 파일로서 동작되도록 나타난다. 더욱이, 데이터 처리 장치는 논리 레지스터 파일을 사용하여 제 2 명령 타입의 제 1 명령을 실행한다. 그러나, 그 데이터 처리 장치는 제 1 명령을 실행하고 있지만, 논리 레지스터 파일은 스택 레퍼런스 레지스터 파일로서 동작하도록 나타난다. 더욱이, 데이터 처리 장치는 단일 논리 레지스터 파일에 대응하는 태그 세트내의 모든 태그를, 제 1 명령 세트의 실행을 시작하는 때와 제 1 명령 세트의 실행을 완료하는 때사이에서 비공백 상태로 바꾼다. 태그는 단일 레지스터 파일내의 레지스터가 공백인지 비공백인지를 식별한다.

본 발명의 다른 측면에 따라서, 스칼라 및 압축 데이터 명령을 실행할 때 부분 콘텍트 스위칭을 구현하는 방법이 설명되어 있다. 이 방법에 따라서, 데이터 처리 장치는 제 1 루틴에 속하는 명령을 수신한다. 그 명령의 실행은 스칼라 연산 또는 압축 데이터 연산중 하나를 필요로 한다. 그러면, 데이터 처리 장치는 스칼라 및 압축 데이터 연산 모두를 실행하는 단일 논리 레지스터 파일로서 소프트웨어에 최소한 부분적으로 나타나는 것이 부분 콘텍트 스위치에 인해 이용할 수 없는지를 판단한다. 그 논리 레지스터 파일이 이용할 수 없다면, 제 1 루틴의 실행은, 논리 레지스터 파일내의 내용이 메모리에 저장되게 하는 제 2 루틴의 실행을 위해 인터럽트된다. 그러나, 논리 레지스터 파일이 이용할 수 있다면, 명령은 논리 레지스터 파일상에서 실행된다.

본 발명의 다른 측면에 따라서, 압축 데이터 명령을 실행하는 방법이 설명되어 있다. 이 방법에 따라서, 실행이 압축 데이터 항목이 부동점 데이터를 저장하는데 또한 사용되는 논리 레지스터 파일내의 레지스터로서 소프트웨어에 최소한 부분적으로 나타나는 것에 기록되게 하는 압축 데이터 명령이 수신된다. 이 명령을 실행하는 결과로서, 압축 데이터 항목은 논리 레지스터의 소수 필드에 기록되고, 숫자가 아닌 또는 무한대를 표시하는 값이 논리 레지스터의 부호 및 지수 필드에 기록된다.

본 발명은 다음 설명과 본 발명을 예시한 첨부 도면을 참조하여 잘 이해될 것이다.

도면의 간단한 설명

도 1은 펜티엄 프로세서가 사용된 컴퓨터 시스템의 일예를 설명하는 블록도;

도 2는 펜티엄 프로세서에 의한 명령 실행을 설명하는 흐름도;

도 3A는 본 발명의 일실시예에 따라 부동점 상태와 압축 데이터 상태의 에일리어싱을 설명하는 기능도;

도 3B와 도3C는 논리 부동점 레지스터에 관한 압축 데이터 레지스터와 물리 부동점의 매핑을 설명하는 도면;

도 3D는 압축 데이터와 부동점 명령을 포함한 실행 스트림;

도 4A는 본 발명의 일실시예에 따라서 현재의 소프트웨어와 호환성이 있고, 여러 운영 체제 기술에서 볼 수 없고, 그리고 효율적인 프로그래밍 기술을 도모할 수 있도록 부동점과 압축 데이터 명령을 실행하는 방법의 일부를 설명하는 흐름도;

도 4B는 도4A에 부분적으로 설명된 방법의 나머지를 설명하는 흐름도;

도 5는 본 발명의 일실시예에 따른 컴퓨터 시스템의 일예를 설명하는 블록도;

도 6A는 본 발명의 일실시예에 따라 두 개의 물리 레지스터 파일을 이용하여 부동점 상태로 압축 데이터 레지스터 상태를 에일리어스하는 장치를 설명하는 블록도;

도 6B는 본 발명의 일실시예에 따라 도6A에서 확대된 부동점 스택 레포런스 파일의 일부를 설명하는 블록도;

도 7A는 본 발명의 일실시예에 따라서, 현재의 소프트웨어와 호환성이 있고, 여러 운영 체제 기술에서 볼 수 없고, 좋은 프로그래밍 실행을 촉진시키고, 그리고 도6A의 하드웨어 배열을 이용하여 실행될 수 있도록 부동점 레지스터 세트상에서 에일리어스되는 레지스터 세트상에서 압축 데이터 명령을 실행하는 방법의 일부를 설명하는 흐름도;

도 7B는 도7A에 부분적으로 설명된 방법의 다른 일부를 설명하는 흐름도;

도 7C는 도7A와 도7C에 부분적으로 설명된 방법의 나머지를 설명하는 흐름도;

도 8은 본 발명의 일실시예에 따라 도7C의 단계(734)를 수행하는 방법을 설명하는 흐름도;

도 9는 본 발명의 일실시예에 따라 도7B의 단계(728)를 수행하는 방법을 설명하는 흐름도;

도 10은 본 발명의 다른 실시예에 따라 단일 레지스터 파일을 이용하여 압축 데이터 상태를 부동점 상태에서 에일리어스하는 장치를 통한 데이터 흐름을 설명하는 블록도;

도 11A는 본 발명의 다른 실시예에 따라서, 현재의 소프트웨어와 호환성이 있고, 여러 운영 체제 기술에서 볼 수 없고, 좋은 프로그래밍 실행을 촉진하고, 그리고 도10의 하드웨어 배열을 이용하여 실행될 수 있도록 압축 데이터와 부동점 명령을 에일리어스된 단일 레지스터 파일에서 실행하는 방법의 일부를 설명하는 도면;

도 11B는 도11A에 부분적으로 설명된 방법의 다른 일부를 설명하는 흐름도;

도 11C는 도11A와 도11B에 부분적으로 설명된 방법의 나머지를 설명하는 흐름도;

도 12A는 도10을 기준으로 설명된 본 발명의 일실시예에 따른 부동점 기억 포맷을 설명하는 도면;

도 12B는 도10을 기준으로 설명된 본 발명의 일실시예에 따른 압축 데이터에 대한 기억 포맷을 설명하는 도면;

도 13은 본 발명의 일실시예에 따라서, 도12A, 도12B, 도12C를 기준으로 설명된 기억 포맷이 구현될 때, 도11B의 단계(1138)를 수행하는 방법을 설명하는 도면;

도 14는 본 발명의 일실시예에 따른 태그를 소거하는 방법을 설명하는 흐름도;

도 15A는 에일리어스되는 별개의 물리 레지스터 파일이 갱신될 수 있는 동안의 시간 간격을 설명하기 위해 압축 데이터와 부동점 명령을 포함한 실행 스트림을 도시한 도면; 및

도 15B는 에일리어스되는 별개의 물리 레지스터 파일이 갱신될 수 있는 동안의 시간 간격을 설명하기 위해 압축 데이터와 부동점 명령을 포함한 다른 실행 스트림을 도시한 도면.

실시예

다음 명세서에서, 본 발명의 완전한 이해를 위해 여러 특정 세목이 설명되어 있다. 그러나. 본 발명은 이러한 여러 특정 세목없이도 실행될 수 있다는 것을 알 수 있다. 다른 예로, 공지된 회로, 구조, 및 기술은 본 발명을 모호하지 않게 하기 위해 상세히 설명되어 있지 않다.

본 발명의 일실시예에 따라서, 본 출원은 프로세서가, 여러 운영 체제 기술에서 볼 수 없고, 좋은 프로그래밍 실행을 촉진하고, 그리고 현재의 소프트웨어에서 볼 수 없는 방식으로 상이한 데이터 타입 작동을 수행하게 하는 상이한 명령 세트를 실행하는 방법과 장치를 설명하고 있다. 이것을 실현하기 위해, 프로세서가 상이한 데이터 타입 작동을 수행하게 하는 상이한 명령 세트는 에일리어스된 단일 레지스터 파일로서 소프트웨서에 최소한 논리적으로 나타나는 것으로 실행된다. 상이한 명령 세트를 실행하는 결과로서 수행되는 데이터 타입 작동은 어느 하나의 타입이 될 수 있다. 예를 들어, 하나의 명령 세트는 프로세서가 스칼라 연산(부동점 및/또는 정수)을 수행하게 할 수 있고, 다른 명령 세트는 프로세서가 압축 연산(부동점 및/또는 정수)을 수행하게 할 수 있다. 다른 예로서, 하나의 명령 세트는 프로세서가 부동점 연산(스칼라 및/또는 압축)을 수행하게 할 수 있고, 다른 명령 세트는 프로세서가 정수 연산(스칼라 및/또는 압축)을 수행하게 할 수 있다. 다른 예로서, 에일리어스된 단일 레지스터 파일은 스택 레퍼런스 레지스터 파일로서 그리고 플랫 레지스터 파일로서 연산될 수 있다. 추가로, 이 출원은 에일리어스된 단일 레지스터 파일로서 소프트웨어에 논리적으로 나타나는 별개의 물리 레지스터 파일을 이용하여 이러한 상이한 명령 세트를 실행하는 장치 및 방법을 설명하고 있다. 더욱이, 이 출원은 단일 물리 레지스터 파일을 이용하여 이러한 상이한 명령 세트를 실행하는 방법 및 장치를 설명하였다.

명백하게, 본 발명은 부동점 명령과 압축 데이터 명령(부동점 및/또는 정수)의 실행을 기준으로 설명될 수 있다. 그러나, 임의 수의 상이한 데이터 타입 작동이 수행될 수 있고, 본 발명은 부동점과 압축 데이터 연산에 제한을 두지 않는다는 것을 알게 될 것이다.

도3A는 본 발명의 일실시예에 따라서 압축 데이터 상태와 부동점 상태의 에일리어싱을 설명하는 기능도이다. 도3A는 부동점 데이터를 저장하는 부동점 레지스터 세트(300)(부동점 상태라 함)와 압축 데이터를 저장하는 압축 데이터 레지스터 세트(310)(압축 데이터 상태라 함)를 나타내고 있다. 여기서, 기호(PDn)는 압축 데이터 레지스터의 물리 위치로서 사용된다. 도3A는 또한 압축 데이터 상태가 부동점 상태에서 에일리어스되는 것을 보여준다. 즉, 최소한 부동점 명령과 압축 데이터 명령은 동일 논리 레지스터 세트에서 실행되도록 소프트웨어에 나타난다. 다수의 별개 물리 레지스터 파일 또는 단일 물리 레지스터 파일을 이용하는 것을 포함하여, 이러한 에일리어싱을 구현하는 다수의 기술이 있다. 이러한 기술의 예는 도4-13을 참조하여 추후 설명될 것이다.

상술된 바와 같이, 현재의 운영 체제는 프로세서가 다중 작업의 결과로서 부동점 상태를 저장하게 하도록 구현된다. 압축 데이터 상태는 부동점 상태에서 에일리어스되기 때문에, 이러한 동일 운영 체제는 부동점 상태에서 에일리어스되는 임의의 압축 데이터 상태를 프로세서가 저장하게 할 수 있다. 결과적으로, 본 발명은 구 운영 체제 태스크 스위치 루틴(물론, 태스크 스위치 루틴은 하나이상의 이벤트 핸들러로서 구

현될 수 있다) 또는 변경된 이벤트 핸들러 또는 기록된 신 운영 체제 이벤트 핸들러를 필요로 하지 않는다. 그러므로, 새로운 또는 변경된 운영 체제는 다중 작업시 압축 데이터 상태를 저장하도록 설계될 필요가 없다. 그것만으로, 이러한 운영 체제를 개발하는데 필요한 비용과 시간이 필요하지 않다. 추가로, 일실시예에서, 압축 데이터 명령의 실행에 의해 발생된 임의의 이벤트는 프로세서에 의해 내부적으로 제공되고, 대응 운영 체제 이벤트 핸들러가 그 이벤트를 제공할 수 있는 현재의 이벤트에서 매핑된다. 결과적으로, 압축 데이터 명령은 보이지 않는 시스템을 작동하는 방식으로 실행된다.

또한 도3A는 부동점 태그 세트(320)와 압축 데이터 태그 세트(330)를 보여준다. 부동점 태그(320)는 도1를 참조하여 설명된 태그(150)에서와 유사한 방식으로 작동한다. 따라서, 각각의 태그는 대응 부동점 레지스터의 내용이 공백인지 비공백인지(예, 유효, 스페셜, 또는 제로)를 나타내는 2 비트를 포함하고 있다. 압축 데이터 태그(330)는 압축 데이터 레지스터(310)에 대응하고, 부동점 태그(320)에서 에일리어스된다. 태그의 각각은 2 비트를 사용하여 구현될 수 있지만, 대체 실시예는 각각의 태그에 대해 단지 1 비트만을 저장할 수 있다. 이러한 1 비트 태그의 각각은 공백인지 비공백인지를 식별한다. 이러한 실시예에서, 이러한 1 비트 태그는 태그 값이 필요할 때 적절한 2 비트 태그 값을 결정함으로써, 2 비트를 포함하고 있는 소프트웨어에서 나타나도록 만들어 질 수 있다. 최소 태그 스위칭을 구현하는 운영 체제는 대응 태그가 비공백 상태를 지시하는 그 레지스터만의 내용을 기억한다. 그 태그는 에일리어스되기 때문에, 이러한 운영 체제는 필요한 압축 데이터와 부동점 상태를 기억할 것이다. 그 반대로, 단순한 태그 스위칭을 구현하는 운영 체제는 태그의 상태와 무관하게 논리적으로 에일리어스된 레지스터 파일의 전체 내용을 기억할 것이다.

일실시예에서, 부동점 레지스터(300)는 도1에서 설명된 부동점 레지스터(145)에서와 유사한 방식으로 작동된다. 따라서, 추가로 도3A는 스택 필드(350)의 상부를 포함한 부동점 상태 레지스터(340)를 보여준다. 스택 필드(350)의 상부는 부동점 레지스터(300)중 하나를 식별하는 스택 지시의 상부(TOS)를 기억하는데 사용된다. 부동점 레지스터(300)이 스택으로서 작동될 때, 그 레지스터의 물리 위치에 대향하는 스택 레지스터의 상부를 기준으로 동작이 수행된다. 그 반대로, 압축 데이터 레지스터(310)는 고정 레지스터 파일(직접 접근 레지스터 파일이라 함)로서 작동된다. 따라서, 그 압축 데이터 명령은 사용된 레지스터의 물리 위치를 나타낸다. 그 압축 데이터 레지스터(310)는 부동점 레지스터(300)의 물리 위치에서 매핑되고, 이 매핑은 스택의 상부가 변할 때 변하지 않는다. 결과적으로, 단일 논리 레지스터 파일이 존재하는 소프트웨어는 스택 레퍼런스 레지스터 파일로서 또는 플랫 레지스터 파일로서 작동될 수 있도록 최소한 나타난다.

도3B와 도3C는 도3A에 도시된 바와 같이, 압축 데이터 레지스터(310)와 압축 데이터 태그(330)를 기준으로, 에일리어스된 부동점 레지스터(300)와 부동점 태그(320)의 매핑을 설명하고 있다. 상술된 바와 같이, 부동점 환경에서, 각각의 레지스터(n)는 TOS 포인터에 의해 식별된 부동점 레지스터에 관하여 지정된다. 도3B와 도3C에 두가지 경우가 도시되어 있다. 그 도면의 각각은 논리 또는 프로그래머-가시 부동점 레지스터(스택)와 논리 또는 프로그래머-가시 압축 데이터 레지스터와의 관계를 나타낸다. 도3B와 도3C에 도시된 내부 원(360)은 물리 부동점/압축 데이터 레지스터 및 대응 태그를 표시하고, 외부 원은 스택 포인터(370)의 상부에 의해 레퍼런스된 논리 부동점 레지스터를 표시한다. 도3B에 도시된 바와 같이, 스택 포인터(370)의 상부는 물리 부동점/압축 데이터 레지스터(0)를 지시한다. 따라서, 물리 부동점/압축 데이터 레지스터와 논리 부동점 레지스터의 대응이 있다. 도면에 도시된 바와 같이, 스택 포인터(370)의 상부가 푸시 또는 팝중 하나를 유발하는 부동점 명령에 의해 수정됨에 따라, 스택 포인터(370)의 상부는 따라서 변한다. 푸시는 도면에서 반시계방향으로의 스택 포인터 상부의 회전에 의해 나타나고, 부동점 팝 동작은 스택 포인터의 상부를 시계방향으로 회전하게 한다.

도3C에 도시된 예에서, 논리 부동점 레지스터(ST0)와 물리 레지스터(0)는 대응하지 않는다. 따라서, 설명된 도3C의 예에서, 스택 포인터(370)의 상부는 물리 부동점 레지스터(ST0)를 지시하고, 이것은 논리 부동점 레지스터(ST0)와 대응한다. 다른 모든 논리 부동점 레지스터는 TOS(370)를 기준으로 접근된다. 부동점 레지스터는 고정 레지스터 파일로서 작동되는 일실시예가 설명되었지만, 대체 실시예는 임의적으로 이러한 레지스터 세트를 구현할 수 있다. 추가로, 일실시예는 부동점과 압축 데이터 연산을 기준으로 설명되었지만, 본 기술은 그것에서 수행되는 연산 타입과 무관하게, 임의의 스택 레퍼런스 레지스터 파일에서 임의의 고정 레지스터 파일을 에일리어스하는데 사용될 수 있다는 것을 알 수 있다.

압축 데이터 상태는 부동점 상태의 일부분 또는 전체에서 에일리어스될 수 있다. 일실시예에서, 압축 데이터 상태는 부동점 상태의 소수부에서 에일리어스된다. 더욱이, 에일리어싱은 부분 또는 전체적일 수 있다. 전체 에일리어싱은 그 레지스터의 전체 내용이 에일리어스되는 실시예를 참조하는데 사용된다. 부분적인 에일리어싱은 도6A를 기준으로 추가 설명된다.

도3D는 본 발명의 일실시예에 따라서 시간외의 부동점과 압축 데이터 명령의 실행을 설명하는 블록도이다. 도3D는 실행의 연대순으로 부동점 명령의 제 1 세트(380), 압축 데이터 명령 세트(382), 및 부동점 명령의 제 2 세트(384)를 도시하고 있다. 압축 데이터 명령 세트(382)의 실행은 시간(T1)에서 시작하여 시간(T2)에서 끝나지만, 부동점 명령 세트의 실행은 시간(T3)에서 시작한다. 다른 명령이 상기 압축 데이터 명령 세트(382)의 실행과 부동점 명령의 제 2 세트(384)사이에서 실행될 수 있거나 실행될 수 없다. 제 1 간격(386)은 시간(T1)과 시간(T3)사이의 시간을 지시하고, 제 2 간격(388)은 시간(T2)와 시간(T3)사이의 시간을 지시한다.

부동점과 압축 데이터 상태는 에일리어스된 레지스터 파일에 저장되기 때문에, 태그는 부동점 명령의 제 2 세트(384)의 실행전에 공백으로 바뀔 수 있다. 그렇지 않으면, 스택 오버플로의 예외가 발생될 수 있다. 따라서, 제 1 간격(386)중에 태그는 공백으로 바뀌게 된다. 이것은 다수의 상이한 방법으로 이루어질 수 있다. 예를 들어, 실시예는 1) 압축 데이터 명령 세트(382)내의 제 1 압축 데이터 명령의 실행이 태그를 공백 상태로 바꾸게 함으로써; 2) 압축 데이터 명령 세트(382)내의 각각의 압축 데이터 명령 실행이 태그를 공백 상태로 바꾸게 함으로써; 3) 실행이 에일리어스된 레지스터 파일을 수정하는 제 1 부동점 명령을 실행하려고 시도할 때 태그를 공백 상태로 바꿈으로써 이것을 이룰 수 있다. 이러한 실시예는, 압축 데이터 상태가 레지스터 파일의 나머지에 따라서 저장되고 재저장될 수 있기 때문에, 단일 콘텍트 스위칭을 지원하는(전체 레지스터 상태를 각각의 태스크 스위치에 저장하고 재저장하는) 현재의 운영 체제

에서 보이지 않는 운영 체제가 남는다.

다른 실시예에서, 단일 및/또는 최소 콘텍트 스위치를 지원하는 운영체제와 호환할 수 있도록, 압축 데이터 명령 세트(382)의 실행은, 블록(390)으로 표시된 이행 명령 세트가 시간(T2)후와 시간(T3)(부동점 명령의 제 2 세트(382)가 시작되는 시간)전에 실행되지 않으면, 제 1 간격(386)에서 태그가 비공백 상태로 바뀌게 한다. 예를 들어, 압축 데이터 명령 세트(382)가 태스크(A)에 속한다고 가정하자. 또한, 태스크(A)는 이행 명령(390)의 실행전에 완전 태스크 스위치(즉, 부분적인 테스크 스위치가 아님)에 의해 인터럽트된다고 가정하자. 이것은 완전 태스크 스위치를 수행하기 때문에, 태스크 스위치 핸들러는 부동점/압축 데이터 상태를 기억하는 부동점 명령(부동점 명령의 제 2 세트(384)로 설명되고, 이 실시예에서는 'FP 태스크 스위치 루틴'이라 함)을 포함할 수 있다. 이행 명령 세트(390)는 실행되지 않았기 때문에, 프로세서는 FP 태스크 스위치 루틴의 실행에 앞서 태그를 비공백 상태로 바꿀 수 있다. 결과적으로, 단일 또는 최소이든지, FP 태스크 스위치 루틴은 전체 에일리어스 레지스터 파일(이 실시예에서는 태스크(A)의 압축 데이터 상태)의 내용을 기억할 수 있다. 그 반대로, 이행 명령 세트(390)가 실행되면, 프로세서는 태그를 제 2 간격(388)에서 공백 상태로 바꾼다. 따라서, 이행 명령 세트(390)의 실행후 태그 스위치가 태스크(A)를 인터럽트하든지 하지 않던지, 프로세서는 부동점 명령의 제 2 세트(384)의 실행에 앞서 태그를 공백 상태로 바꿀 수 있다(부동점 명령의 제 2 세트(384)가 태스크 스위치 핸들러, 태스크(A), 또는 다른 프로그램에 속하는지에 무관하게).

다른 실시예로서, 이행 명령 세트(390)의 실행에 앞서 태스크(A)가 태스크 스위치에 의해 인터럽트되고, 압축 데이터 명령 세트(382)가 태스크(A)에 속한다고 다시 가정하자. 그러나, 이때, 태스크 스위치는 부분 태스크 스위치이다(즉, 부동점/압축 데이터 상태는 저장되지 않거나 재저장되지 않는다). 부동점 또는 압축 데이터 명령을 이용한 어떠한 태스크도 실행되지 않는다면, 프로세서는 결과적으로 태스크(A)를 실행하는 것으로 반전하고, 이행 명령 세트(390)는 실행될 것이다. 그러나, 다른 태스크(즉, 태스크(B))가 부동점 또는 압축 데이터 명령을 이용한다면, 이러한 명령 실행의 시도는 운영 체제 핸들러 콜(call)이 태스크(A)의 부동점/압축 데이터 상태를 저장하게 하고, 그리고 태스크(B)의 부동점/압축 데이터 상태를 재저장하게 한다. 이러한 핸들러는 부동점/압축 데이터 상태를 저장하는 FP 태스크 스위치 루틴(이 실시예에서는, 부동점 명령(384)로서 설명)을 포함할 수 있다. 이행 명령 세트(390)가 실행되지 않았기 때문에, 프로세서는, FP 태스크 스위치 루틴의 실행에 앞서, 태그를 비공백 상태로 바꿀 수 있다.
결과적으로, FP 태스크 스위치 루틴은, 최소 또는 단일이던지, 에일리어스된 전체 레지스터 파일(즉, 태스크(A)의 압축 데이터 상태)의 내용을 기억할 것이다. 이러한 방식으로, 이 실시예에서는, 에일리어스된 레지스터의 상태를 저장하는데 사용된 기술과 무관하게 볼 수 있는 운영 체제가 남는다.

이행 명령 세트는 여러 방법으로 구현될 수 있다. 일실시예에서, 이 이행 명령 세트는 여기서 EMMS(공백 멀티미디어 상태)명령으로서 언급된 새로운 명령을 포함하고 있다. 이 명령은 부동점/압축 데이터 태그가 차후 실행되는 코드에서 모든 부동점 레지스터(300)는 실행될 수 있는 차후 부동점 명령에 이용가능하다는 것을 지시하게 한다. 이것은 EMMS 명령이 압축 데이터 명령후에 실행되는 것이 아니라 부동점 명령전에 실행된다면, 일어날 수 있는 스택 오버플로 조건 발생을 방지한다.

인텔 아키텍처 프로세서를 이용한 종래의 부동점 프로그래밍 실행에서, 부동점 상태를 소거하는 연산에 의해 부동점 코드의 블록을 종결시키는 것이 일반적이다. 부분 및/또는 최소 콘텍트 스위칭이 사용되는 것과는 무관하게, 부동점 상태는 부동점 코드의 제 1 블록을 종결시키는 소거 조건으로 남게 된다. 그러므로, EMMS 명령은 압축 데이터 코드를 소거하기 위해 압축 데이터 시퀀스에 사용되도록 되어 있다. EMMS 명령은 압축 데이터 코드의 블록후에 실행될 수 있다. 따라서, 여기서 설명된 방법 및 장치를 구현한 프로세서는 인텔 아키텍처 프로세서를 이용한 종래의 부동점 프로세서와 완전 호환성을 가지고 있지만, 좋은 프로그래밍 기술로 프로그램되고 적절히 보조 관리(압축 데이터 코드와 부동점 코드사이에서의 이행전에 그 상태를 소거하는 것)한다면, 부동점 또는 압축 데이터 상태중 어느하나에 역영향을 미치지 않고 압축 데이터와 부동점 코드사이에서의 이행을 가능하게 하는 압축 데이터 명령의 실행 능력을 가지고 있다.

다른 실시예에서, 이행 명령 세트는, 프로세서가 실행될 때 태그를 공백 상태로 바꾸게 하는 현재의 부동점 명령을 이용하여 구현될 수 있다.

일실시예에서, 압축 데이터 명령과 부동점 명령사이의 스위칭은 시간을 소모한다. 따라서, 좋은 프로그래밍 기술은 이러한 이행의 수를 최소화하는 것이다. 부동점과 압축 데이터 명령사이의 이행 수는 압축 데이터 명령에서 분리한 부동점 명령을 그룹화함으로써 줄일 수 있다. 이러한 좋은 프로그래밍 기술을 진전시키는 것이 바람직하기 때문에, 이러한 좋은 프로그래밍 기술을 무시하기 어려운 프로세서를 구현하고자 한다. 따라서, 일실시예는 또한 제 1 간격(386)동안의 초기 상태(즉, 레지스터를 지시하는 제로(R0))에서 스택 지시의 상부를 바꾼다. 이것은 상이한 여러 방법으로 이루어 질 수 있다: 1) 제 1 압축 데이터 명령의 실행이 스택 지시의 상부를 바꾸게 하는 방법; 2) 압축 데이터 명령 세트(382)내의 각각의 압축 데이터 명령 실행이 스택 지시의 상부를 바꾸게 하는 방법; 3) EMMS 명령의 실행이 스택 지시의 상부를 세트시키게 하는 방법; 4) 도3D에서의 시간(T3)에 부동점 명령 실행을 시도하여 스택 지시의 상부를 바꾸는 방법등. 다시, 이것은 압축 데이터 명령을 부동점 명령과 혼합하는 코드내의 완전한 호환성을 유지하는 것이다. 또한 좋은 프로그래밍 기술을 진전시키는 측면에서, 일실시예는 또한, 제 1 간격(386)동안, 압축 데이터가 기록되는 에일리어스 레지스터의 부호 및 지수 필드내의 수를 지시하지 않는 값을 저장한다.

도4A와 도4B는 여러 운영 체제 기술에서 볼 수 없고 본 발명의 일실시예에 따른 효율적인 프로그래밍 기술을 진전시키는 방식으로 부동점과 압축 데이터 명령을 실행하는 방법을 설명하는 일반적인 흐름도이다. 이 흐름도는 단계(400)에서 시작한다. 단계(400)에서, 단계(402)로 진행한다.

단계(402)에서 보는 바와 같이, 비트 세트는 하나의 명령으로서 액세스되고, 단계(404)로 진행한다. 이러한 비트 세트는 그 명령에 의해 수행된 연산을 식별하는 연산 코드를 포함하고 있다.

단계(404)에서, 그 연산 코드가 유효한지를 판단한다. 그 연산 코드가 유효하지 않다면, 단계(405)로 진행한다. 그 반대면, 단계(408)로 진행한다. 압축 데이터 명령을 포함한 루틴의 실행은 압축 데이터 명령을 지원하지 않은 프로세서에서 시도된다고 가정하면, 압축 데이터 명령에 대한 연산 코드는 유효하지 않

을 것이고, 단계(406)로 진행할 것이다. 그 반대로, 그 프로세서가 압축 데이터 명령을 실행할 수 있다면, 이러한 명령에 대한 연산 코드는 유효하게 될 것이고, 단계(408)로 진행할 것이다.

단계(406)에서 보는 바와 같이, 유효 연산 코드의 예외가 발생되고, 적절한 이벤트 핸들러는 실행된다. 도2의 단계(215)를 기준으로 앞서 설명된 바와 같이, 이 이벤트 핸들러는 프로세서가 메시지를 디스플레이하게 하고, 현 태스크의 실행을 중단하게 하고, 그리고 다른 태스크를 실행하게 진행하도록 구현될 수 있다. 물론 이 이벤트 핸들러는 여러 방법으로 구현될 수 있다. 예를 들어, 이 이벤트 핸들러는 프로세서가 압축 데이터 명령을 실행할 수 없는지를 식별하도록 구현될 수 있다. 이러한 동일 이벤트 핸들러는, 프로세서가 압축 데이터 명령을 실행할 수 없다는 것을 식별하는 지시를 세트하도록 구현될 수 있다. 그 프로세서를 통해 실행하는 다른 응용예는, 스칼라 루틴 세트 또는 압축 데이터 루틴 복제 세트를 이용하여 실행할 수 있는지를 판단하는 지시를 이용할 수 있다. 그러나 이러한 구현은 현재의 운영 체제의 교체 또는 새로운 운영 체제의 개발을 필요로 할 수 있다.

단계(408)에서, 어떤 타입의 명령이 수신되었는지를 판단한다. 그 명령이 부동점 명령도 아니고 압축 데이터 명령도 아니다면, 단계(410)로 진행한다. 그러나 그 명령이 부동점 명령이다면, 단계(412)로 진행한다. 그 반대로, 그 명령이 압축 데이터 명령이다면, 단계(414)로 진행한다.

단계(410)에서 보는 바와 같이, 프로세서는 그 명령을 실행한다. 이 단계는 본 발명을 이해하는데 필요하지 않기 때문에, 여기서 추가 설명하지 않는다.

단계(412)에서 보는 바와 같이, EM 지시가 1인지를 판단하고(설명된 소프트웨어 규정에 따라, 부동점 유닛이 대리 실행될 수 있다면), 그리고 TS 지시가 1인지를 판단한다(설명된 소프트웨어 규정에 따라, 부분 콘텍트 스위치가 수행되었다면). EM 지시 및/또는 TS 지시가 1이다면, 단계(416)로 진행한다. 그 반대면, 단계(420)로 진행한다. 일실시예는 EM 지시 및/또는 TS 지시가 1일 때 디바이스가 예외를 활용할 수 없도록 구현되지만, 대체 실시예는 임의수의 다른 값을 이용하도록 구현될 수 있다.

단계(416)에서, 예외를 활용할 수 없는 디바이스가 생성되고, 대응 이벤트 핸들러는 실행된다. 도2의 단계(235)를 기준으로 앞서 설명된 바와 같이, 대응 이벤트 핸들러는 EM 지시와 TS 지시를 폴링하도록 구현될 수 있다. EM 지시가 1이다면, 이벤트 핸들러는 그 명령을 실행하기 위해 부동점 유닛을 대리 실행하고, 프로세서가 다음 명령(단계(402)에서 수시된 명령을 논리적으로 뒤따르는 명령)으로 실행을 재개시하게 한다. TS 지시가 1이다면, 이벤트 핸들러는 프로세서가 부분 콘텍트 스위치를 기준으로 앞서 설명된 바와 같이 동작하게 하고(부동점 유닛의 내용을 저장 및 필요하다면 올바른 부동점 상태를 재저장), 그리고 프로세서가 단계(402)에서 수신된 명령 실행을 재시작함으로써 실행을 재개시하게 한다. 물론, 대체 실시예는 여러 방법으로 이러한 이벤트 핸들러를 구현할 수 있다. 예를 들어, EM 지시는 다중 작업을 실행하는데 사용될 수 있다.

압축 데이터 상태는 부동점 상태에서 에일리어스되기 때문에, 그리고 EM 및 TS 지시는 부동점 상태를 변하게 하기 때문에, 프로세서는, 완전한 소프트웨어 호환성을 갖기 위해서 압축 데이터 명령을 실행할 때, EM 및 TS 지시에 응답하여야 한다.

단계(414)에서, EM 지시가 1인지를 판단한다. 앞서 설명된 바와 같이, 예외를 활용할 수 없는 디바이스를 제공하도록 실행된 이벤트 핸들러는 EM 지시를 폴링하도록 구현될 수 있고, EM 지시가 1이다면 부동점 유닛의 대리 실행을 시도한다. 현재의 이벤트 핸들러는 압축 데이터 명령을 대리 실행하도록 기록되어 있지 않기 때문에, EM 지시가 1인 동안에 압축 데이터 명령 실행의 시도는 이러한 이벤트 핸들러에 의해 제공될 수 없다. 더욱이, 보이지 않는 운영 체제를 갖기 위해, 그 프로세서는 이러한 이벤트 핸들러의 대체를 필요로 하지 않는다. 결과적으로, 단계(414)에서 EM 지시가 1이다고 결정되면, 단계(416)보다는 단계(406)로 진행한다. 그 반대면, 단계(418)로 진행한다.

앞서 설명된 바와 같이, 단계(406)에서 무효 연산 코드의 예외는 생성되고, 대응 이벤트 핸들러는 실행된다. 무효 연산 코드의 예외에서 EM=1인 동안에 압축 데이터 명령 실행의 시도를 전환함으로써, 이 실시예는 보이지 않는 운영 체제를 갖는다.

일실시예는 보이지 않는 시스템을 동작하는 방식으로 EM 지시를 조정하는 것에 대해 설명되었지만, 대체 실시예는 다른 기술을 사용할 수 있다. 예를 들어, 대체 실시예는 EM 지시가 1일 때, 압축 데이터 명령 실행의 시도에 응답하여 예외를 이용할 수 없는 디바이스, 현재의 상이한 이벤트, 새로운 이벤트중 어느 하나를 생성할 수 있다. 더욱이, 그 운영 체제에서 약간의 수정이 가능하다면, 그 선택된 이벤트 핸들러는 이러한 상황에 응답하여 임의의 작용이 적절하다고 생각되도록 바꿀 수 있다. 예를 들어, 이벤트 핸들러는 압축 데이터 명령을 대리 실행하도록 기록될 수 있다. 다른 대체 실시예는 압축 데이터 명령을 실행할 때 EM 지시를 묵살할 수 있다.

단계(418)에서 보는 바와 같이, TS 지시가 1(현재의 소프트웨어 규정에 따라 부분 콘텍트 스위치가 수행되었다면)인지를 판단한다. TS 지시가 1이다면, 단계(416)로 진행한다. 그 반대면, 단계(422)로 진행한다.

앞서 설명된 바와 같이, 단계(416)에서, 예외를 활용할 수 없는 디바이스가 생성되고, 그 대응 이벤트 핸들러가 실행된다. 따라서, 이러한 이벤트에 대응하여, 그 대응 이벤트 핸들러는 EM 지시와 TS 지시를 폴링하도록 구현될 수 있다. 단계(414)가 유효 연산 코드의 예외에서 EM 지시가 1인 경우를 전환하였기에, EM 지시는 0이 되어야 하고, TS 지시는 1이 되어야 한다. TS 지시는 1이기 때문에, 이벤트 핸들러는 부분 콘텍트 스위치를 기준으로 앞서 설명된 바와 같이 작용하고(부동점 유닛의 내용을 저장 및 필요하다면 올바른 부동점 상태를 재저장), 그리고 단계(402)에서 수신된 명령 실행을 재시작함으로써 실행을 재개시하게 한다. 압축 데이터 상태는 부동점 상태에서 에일리어스되기 때문에, 이러한 이벤트 핸드러는 부동점과 압축 데이터 상태 모두에 대하여 작용한다. 결과적으로, 이 방법은 보이지 않는 운영체제를 가지고 있다. 물론, 대체 실시예는 여러 방법으로 이 이벤트 핸들러를 구현할 수 있다. 예를 들어, 압축 데이터가 부동점 상태에서 에일리어스되지 않은 대체 실시예는 부동점과 압축 데이터 상태 모두를 저장하는 새로운 핸들러를 이용한다.

일실시예는 보이지 않은 시스템을 작동하는 방식으로 TS 지시를 조정하는 것에 대하여 설명하였지만, 대체 실시예는 다른 기술을 사용할 수 있다. 예를 들어, 대체 실시예는 TS 지시를 구현할 수 없다. 이러한 대체 실시예는 부분 콘텍트 스위칭을 구현하기 위해 TS 지시를 이용하는 운영 체제와 호환되지 않을 수 있다. 그러나, 이러한 대체 실시예는 TS 지시를 이용한 부분 콘텍트 스위칭을 지원하지 않는 현재의 운영 체제와 호환될 수 있다. 다른 예로서, TS 지시가 1일 때의 압축 데이터 명령 실행의 시도는 새로운 이벤트 핸들러에서 전환될 수 있거나, 수정된 현재의 이벤트 핸들러에서 전환될 수 있다. 이러한 이벤트 핸들러는 이러한 경우에 대응하여 임의의 작동이 적절하다고 생각되도록 구현될 수 있다. 예를 들어, 압축 데이터 상태가 부동점 상태에서 에일리어스되는 일실시예에서, 이 이벤트 핸들러는 압축 데이터 상태 및/또는 부동점 상태를 저장할 수 있다.

도2를 참조하여 설명된 바와 같이, 임의 숫자 오류가 부동점 명령의 실행동안에 발생된다면, 이러한 오류는, 실행이 그 오류를 제공하는 것을 막을 수 있는 다음 부동점 명령 실행의 시도때까지 보류를 유지한다. 단계(420)와 단계(422)모두에 도시된 바와 같이, 현재 제공될 수 있는 임의의 이러한 보류 오류가 있는지를 판단한다. 따라서, 이 단계는 도2의 단계(240)와 유사하다. 이러한 보류 오류가 있다면, 단계(420)과 단계(422)모두에서 단계(424)로 진행한다. 그러나, 이러한 보류 오류가 없다고 단계(420)에서 결정되면, 단계(426)로 진행한다. 그 반대면, 이러한 보류 오류가 없다고 단계(422)에서 결정되면, 단계(430)로 진행한다. 대체 실시예에서, 이러한 오류는 압축 데이터 명령의 실행동안에 보류된다.

단계(424)에서, 보류한 부동점 오류의 예외가 발생된다. 도2의 단계(245)를 기준으로 설명된 바와 같이, 이 이벤트에 대응하여, 프로세서는 부동점 오류가 마스크되는지를 판단한다. 만일 그러하다면, 프로세서는 내부적으로 그 이벤트 조정을 시도하고, 부동점 명령은 지극히 작게 재시작된다. 부동점 오류가 마스크되지 않는다면, 그 이벤트는 외부 이벤트이고, 그 대응 이벤트 핸들러가 실행된다. 이러한 이벤트 핸들러는 그 오류를 제공하도록 구현될 수 있고, 단계(402)에서 수신된 명령의 실행을 재시작함으로써 실행을 재개시하게 한다. 물론, 대체 실시예는 여러 방법으로 이러한 이벤트 핸들러를 구현할 수 있다.

단계(426)에서 보는 바와 같이, 부동점 명령이 실행된다. 보이지 않는 운영 체제를 갖기 위해, 또한 일실시예는 필요한 태그를 바꾸고, 현재 제공될 수 있는 숫자 오류를 기록하고, 다른 숫자 오류를 보류로 유지한다. 부동점 유닛의 내용을 저장하는 다수의 운영 체제가 있기 때문에, 모든 이러한 운영 체제 기술에서 볼 수 없는 방식으로 압축 데이터와 부동점 명령을 실행하는 것이 바람직하다. 그 태그를 유지함으로써, 이 실시예는, 대응 태그가 비공백 상태를 지시하는 그 부동점 레지스터만의 내용을 저장하는 임의의 운영 체제 기술에서 볼 수 없는 운영 체제를 갖는다. 그러나, 대체 실시예는 이러한 운영 체제 기술보다는 적게 호환되도록 구현될 수 있다. 예를 들어, 현재의 운영 체제가 태그를 이용할 수 없다면, 태그를 구현할 수 없는 프로세서는 그 운영 체제와 여전히 호환될 수 있다. 더욱이, 본 발명에서 숫자 부동점의 예외가 보류될 필요가 없어서, 그렇게 할 수 없는 대체 실시예는 여전히 본 발명의 범위내에 있다.

단계(430)에서 보는 바와 같이, 압축 데이터 명령이 EMMS 명령(또한, 이행 명령이라 함)인지를 판단한다. 압축 데이터 명령이 EMMS 명령이다면, 단계(432)로 진행한다. 그 반대면, EMMS 명령은 부동점 태그를 초기 상태로 바꾸는 데 사용된다. 따라서, 압축 데이터 상태가 부동점 상태에서 에일리어스된다면, 이 명령은, 압축 데이터 명령을 실행하는 것에서 부동점 명령을 실행하는 것으로 이행될 때 실행될 수 있다. 이러한 방식으로, 부동점 유닛은 부동점 명령의 실행을 위해 초기화된다. 부동점 상태에서 압축 데이터 상태를 에일리어스하지 않는 대체 실시예는 단계(430)와 단계(432)를 수행할 필요가 없을 것이다. 추가로, EMMS 명령이 대리 실행된다면, 단계(430)와 단계(432)는 필요가 없다.

단계(432)에서 보는 바와 같이, 모든 태그는 공백 상태로 바뀌고, 스택 지시의 상부는 초기 상태로 바뀐다. 태그를 공백 상태로 바꿈으로써, 부동점 유닛은 초기화되었고, 부동점 명령의 실행을 준비한다. 스택 지시를 초기값(일실시예에서는 레지스터를 식별하기 위해 제로이다(R0))으로 바꾸는 것은 부동점과 압축 데이터 명령을 개별적으로 그룹화하게 하고, 그래서 프로그래밍 기술을 좋게 한다. 대체 실시예는 스택 지시의 상부를 초기화할 필요가 없다. 단계(432)의 완료와 동시에, 그 시스템은 다음 명령(단계(402)에서 수신된 명령을 논리적으로 뒤따르는 명령)을 자유롭게 실행한다.

단계(434)에서 보는 바와 같이, 압축 데이터 명령이 실행되고(임의의 숫자의 예외를 발생시키지 않고), 그리고 스택 지시의 상부는 초기값으로 바뀐다. 숫자의 예외를 발생시키는 것을 방지하기 위해, 일실시예는 데이터 값이 최대 또는 최소값으로 포화 및/또는 클램프되도록 압축 데이터 명령을 구현한다. 임의 숫자의 예외를 발생시키지 않음으로써, 이벤트 핸들러는 그 예외를 제공할 필요가 없다. 결과적으로, 본 발명의 실시예는 보이지 않는 시스템을 작동시킨다. 대안으로, 실시예는 이러한 숫자의 예외에 응답하여 마이크로코드 이벤트 핸들러를 실행하도록 구현될 수 있다. 보이지 않는 시스템을 완전히 작동시키지 않는 대체 실시예는 추가 이벤트 핸들러가 그 운영 체제에 포함되거나, 현재의 이벤트 핸들러가 오류를 제공하기 위해 바뀌도록 구현될 수 있다. 스택의 상부는 상술된 바와 같은 이유로 바뀌게 된다. 대체 실시예는 임의의 상이한 시간에 스택의 상부를 바꾸도록 구현될 수 있다. 예를 들어, 대체 실시예는 EMMS를 제외한 모든 압축 데이터 명령 예외의 실행과 동시에 스택 지시의 상부를 바뀌도록 구현될 수 있다. 다른 대체 실시예는 EMMS를 제외한 어떠한 다른 압축 데이터 명령의 실행없이 스택 지시의 상부를 바꾸도록 구현될 수 있다. 어떤 메모리 이벤트가 압축 데이터 명령을 실행하는 것을 시도하는 결과로서 발생된다면, 실행은 인터럽트되고, 스택의 상부는 바뀌지 않고, 그리고 이벤트는 제공된다. 그 이벤트를 제공하는 것을 완료함과 동시에, 단계(402)에서 수신된 명령은 재시작된다. 단계(434)에서, 단계(436)로 진행한다.

단계(436)에서 보는 바와 같이, 압축 데이터 명령이 프로세서가 에일리어스된 레지스터에 기록되게 하는지를 판단한다. 그러하다면, 단계(438)로 진행한다. 그렇지 않으면, 단계(440)로 진행한다.

단계(438)에서, 압축 데이터 명령이 프로세서를 기록하게 하는 각각의 에일리어스된 레지스터의 부호 및 지수 필드에 1이 저장되어 있다. 단계(438)에서, 단계(440)로 진행한다. 이러한 단계를 수행하는 것은 부동점과 압축 데이터 명령의 개별 그룹화하게 하는 좋은 프로그래밍 기술을 진전시킨다. 물론, 이 결과와 관련이 없는 대체 실시예는 이러한 단계를 실행하는 것을 피할 것이다. 일실시예에서, 부호 및 지수 필드에 1이 기록되지만, 대체 실시예는 NAN(숫자가 아닌) 또는 무한대를 표시하는 임의의 값을 사용할 수 있

다.

단계(440)에서 보는 바와 같이, 모든 태그는 비공백 상태로 바뀌게 된다. 모든 태그를 비공백 상태로 바꾸는 것은 부동점과 압축 데이터 명령의 개별 그룹화하게 하는 좋은 프로그래밍 기술을 진전시킨다. 추가로, 운영 체제 호환성면에서, 특정 운영 체제 기술은 대응 태그가 비공백 상태를 지시하는 부동점 레지스터만의 내용을 기억한다(최소 콘텍트 스위칭). 따라서, 압축 데이터 상태가 부동점 상태에서 에일리어스되는 실시예에서, 모든 태그를 비공백 상태로 바꾸는 것은 이러한 운영 체제를 부동점 상태로 있는 것처럼 압축 데이터 상태를 보존하게 한다. 대체 실시예는 대응 레지스터가 유효 압축 데이터 항목을 포함한 태그만을 바꿀 수 있다. 더욱이, 대체 실시예는 이 운영 체제 기술보다 적게 호환되게 구현될 수 있다. 예를 들어, 현재의 운영 체제가 그 태그를 이용하지 않는다면(예, 전체 레지스터 상태를 저장 및 재저장하는 운영 체제), 그 태그를 실행하지 않은 실시예는 그 운영 체제와 여전히 호환될 수 있다. 단계(440)의 완료와 동시에, 그 시스템은 다음 명령(단계(402)에서 수신된 명령을 논리적으로 뒤따르는 명령)을 자유롭게 실행한다.

따라서, 이 실시예에서, 메모리내의 태그의 내용은 부동점 상태 세이브(FSAVE)를 바꾸거나, 부동점 환경 기억(FSTENV) 명령은 아래의 표1를 기준으로 도시되어 있다.

[표 1]

태그 워드에서의 압축 데이터/FP 명령의 효과

명령 타입	명령	태그 비트	FSAVE/FSTENV후 메모리내의 계산된 태그 워드
압축 데이터	임의 (EMMS를 제외)	비공백 (00,01,또는 10)	비공백 (00,01,또는 10)
압축 데이터	EMMS	공백(11)	공백(11)
부동 소수점	임의	00,11	00,11,01,또는 10
부동 소수점	FRSTOR, FLDENV	00,11,01,또는 10	00,11,01,또는 10

도시된 바와 같이, EMMS를 제외한 압축 데이터 명령중 어느 하나는 태그(320)를 비공백 상태(00)로 설정되게 한다. EMMS는 부동점 태그 레지스터를 공백(11)로 설정되게 한다. 추가로, EMMS를 포함한 어느 하나의 압축 데이터 명령은 스택 필드(350)의 상부에 저장된 스택 지시의 상부를 0으로 재설정되게 한다.

인텔 아키텍처 프로세서내의 제어 및 상태 워드(TOS를 제외한)와 같은 그외 환경 레지스터는 변하지 않는다. 압축 데이터 판독 또는 EMMS중 어느 하나는 부동점 레지스터(300)의 소수부와 지수부를 불변 상태로 유지한다. 그러나, 일실시예에서, 압축 데이터 레지스터에서의 하나의 압축 데이터 기록은, 에일리어싱 메카니즘으로 인해, 대응 부동점 레지스터의 소수부를, 수행되는 연산에 따라 변경시킨다. 더욱이, 이 실시예에서, 압축 데이터 레지스터(310)의 수정에 의한 부동점 레지스터의 소수부내의 데이터 기록은 부동점 레지스터(300)의 부호부와 지수부내의 모든 비트를 1로 세팅시킨다. 그 압축 데이터 명령은 부동점 레지스터의 부호부와 지수부를 사용하지 않기 때문에(압축 데이터 레지스터의 부호부와 지수부에서 압축 데이터의 에일리어싱이 없다), 이것은 압축 데이터 명령에 어떠한 영향을 주지 않는다. 앞서 설명된 바와 같이, 대체 실시예는 압축 데이터 상태를 부동점 상태의 일부에서 에일리어스할 수 있다. 추가로, 대체 실시예는 다른 값을 기록하거나 레지스터의 부호부 및/또는 지수부를 바꾸지 않는 것을 선택할 수 있다.

[표 2]

FPU에서의 압축 데이터 명령의 효과

명령 타입	태그 워드	TOS (SW 13..11)	다른 FPU환경 (CW Data ptr, Code ptr, 다른 SW 필드)	압축 데이터 레지스터의 지수비트 +부호 비트 (압축데이터)	압축 데이터 레지스터의 소수부 (압축데이터)
압축 데이터 레지스터로 부터 압축 데이터 판독	00(비공백)으로 설정된 모든 필드	0	불변	불변	불변
압축 데이터 레지스터로 부터 압축 데이터 기록	00(비공백)으로 설정된 모든 필드	0	불변	1로 설정	변형
EMMS	11(공백)로 설정된 모든 필드	0	불변	불변	불변

압축 데이터 명령의 실행을 추가 지시하기 위해, 기록된 부동점 레지스터의 부호부와 지수부는 모두 1로 설정된다. 부동점 레지스터가 부동점 레지스터의 지수부를 사용하기 때문에 이렇게 되고, 그 레지스터의 이 부분은 압축 데이터 명령의 실행후 행렬식 상태로 남아 있는 것이 바람직하다. 인텔 아키텍처 마이크로프로세서에서, 모두 1로 설정된 부동점의 지수부는 숫자가 아닌 것(NAN)으로 해석된다. 따라서, 압축

데이터 태그(330)를 비공백 상태로 설정하는 것에 추가로, 부동점 레지스터의 지수부는 모두 1로 설정되고, 이것은 압축 데이터 명령이 이미 실행되었다는 것을 지시하는데 사용될 수 있다. 이것은 그 데이터를 수정할 수 있는 부동점 명령과 압축 데이터 명령으로부터 데이터의 혼합을 할 수 없고, 적당하지 않은 결과를 야기한다. 따라서, 부동점 코드는, 부동점 레지스터가 부동점 데이터를 포함하고 있을 때와 압축 데이터를 포함하고 있을 때를 분별하는 추가 방법을 가지고 있다.

따라서, 현재의 운영 체제(워싱턴 레드몬드 소재, 마이크로소프트웨어사의 MS 윈도우 브랜드 운영 환경과 같은)와 호환성이 있고, 좋은 프로그래밍 기술을 진전시키는 압축 데이터 명령을 실행방법이 설명되어 있다. 압축 데이터 상태는 부동점 상태에서 에일리어스되기 때문에, 압축 데이터 상태는 부동점 상태인것처럼 현재의 운영 체제에 의해 보존되고 재저장될 것이다. 더욱이, 압축 데이터 명령의 실행에 의해 발생된 이벤트는 현재의 운영 체제 이벤트 핸들러에 의해 제공될 수 있기 때문에, 이 이벤트 핸들러는 수정될 필요가 없고, 새로운 이벤트 핸들러가 추가될 필요가 없다. 결과적으로, 그 프로세서는 역으로 호환성이 있고, 업그레이드는 운영 체제를 수정하거나 개발하는데 필요한 비용과 시간을 필요로 하지 않는다.

또한 현재의 운영 체제와 호환할 수 있는 본 방법의 상이한 실시예는 도7A-C, 도8, 도9, 도11A-C를 참조하여 설명되어 있다. 이러한 실시예는 다르지만, 다음 사항은 이 실시예(도4A-B에 도시된 실시예; 도7A-C, 도8, 도9에 도시된 실시예; 및 도11A-C에 도시된 실시예)의 모두에서 공통이다: 1) 부동점 상태와 압축 데이터 상태는 최소한 단일 레지스터 논리 파일에 저장되어 있는 소프트웨어에 나타난다; 2)EM 비트가 '부동점 명령은 대리 실행되어야 한다'는 것을 지시할 때의 압축 데이터 명령의 실행은, 예외를 이용할 수 없는 디바이스보다는 무효 연산 코드의 예외가 된다; 3) TS 지시가 '부분 콘텍트 스위치은 수행되었음'을 지시할 때의 압축 데이터 명령의 실행은 예외를 이용할 수 없는 디바이스가 된다; 4) 보류한 부동점 이벤트는 어느 하나의 압축 데이터 명령의 시도된 실행에 의해 제공된다; 5) 어느 하나의 압축 데이터 명령의 실행은 다음 부동점 명령의 실행에 앞서 0으로 바뀌게 되는 스택 지시의 상부가 될 것이다; 6) 어느 하나의 압축 데이터 명령의 실행 다음에 EMMS 의 실행이 오지 않으면, EMMS 명령의 실행은, 다음 부동점 명령의 실행에 앞서 모든 태그를 공백 상태로 바뀌게 할 것이다; 7) EMMS 명령의 실행 다음에 어느 하나의 압축 데이터 명령의 실행이 오지 않으면, 태그는 다음 부동점 명령의 실행에 앞서 비공백 상태로 바뀌게 된다; 8) NAN(숫자가 아님) 또는 무한대를 나타내는 일부 값은 압축 데이터 명령의 실행에 대응하여 프로세서에 의해, 기록된 어느 하나의 FP/PD 레지스터의 부호부와 지수부에 저장된다; 및 9) 새로운 비 마이크로코드 이벤트 핸들러가 필요하지 않다.

일부가 설명된, 도4A-B에 도시된 실시예의 변화는 이러한 운영 체제와 전체 또는 부분적으로 호환성이 있을 수 있고, 또는 좋은 프로그래밍 기술을 진전시킬 수 있다. 예를 들어, 본 발명의 대체 실시예는 도4A-B에 도시된 흐름도내의 상이한 위치에서 특정 단계로 이동할 수 있다. 본 발명의 다른 실시예는 하나이상의 단계를 제거하거나 바꿀 수 있다. 예를 들어, 대체 실시예는 EM 비트를 지원할 수 없다. 물론, 본 발명은 다수의 시스템 아키텍처에 유용할 수 있고, 여기서 설명된 아키텍처에 제한되지 않는다.

상기 방법을 부동점과 압축 데이터 명령에 이용하여, 본 발명의 실시예를 사용하는 프로그래머는 도3D에 도시된 압축 데이터 명령과 부동점 명령의 개별 블록으로 구성되어 있는 부분으로 그 코드를 분할할 것을 권한다. 이것은 부동점 연산의 시퀀스에서 압축 데이터 연산으로의 이행과 그 반대 이행에 앞서, 압축 데이터 상태의 저장과 소거를 가능하게 한다. 이것은 또한 태스크 스위치동안에 콘텍트를 저장하는 것을 포함한 종래의 태스크 스위칭 메카니즘과 호환이 가능하다.

압축 데이터 명령은 부동점 레지스터(300)(도3A)에 영향을 주고, 임의의 단일 압축 데이터 명령은 모든 부동점 태그를 비공백 상태로 설정하기 때문에, 코드 타입의 블록으로 코드를 분할하는 것을 적당한 부기를 위해 권한다. 블록내의 혼합된 부동점과 압축 데이터 명령의 실행의 예는 도3D에 예시되어 있다. 이것은 단일 응용예에, 혼합된 부동점과 압축 명령 응용 코드 또는 협력 다중 작업 운영 체제내의 연산을 포함할 수 있다. 어느 하나의 경우에 있어서, 부동점 레지스터(300)의 적당한 부기, 대응 태그, 및 스택 지시의 상부는 기능을 부동점 코드와 압축 데이터 코드의 개별 블록으로 분할함으로써 확실히 된다.

예를 들어, 도3D에 예시된 바와 같이, 실행 스트림은 부동점 명령의 제 1 세트(380)를 포함할 수 있다. 부동점 명령 블록(380)의 종료후, 부동점 상태는 원한다면, 응용예에 의해 저장될 수 있다. 이것은 부동점 스택을 ,팝하는 것을 포함한 다수의 공지된 기술을 이용하여, 인텔 아키텍처 프로세서내의 FSAVE/FNSAVE 명령을 이용하여 수행될 수 있다. 이것은, 부동점 환경을 저장하고 그 대응 부동점 레지스터가 유효 데이터를 포함하고 있다는 지시에 대해 개별 태그를 체크하는 최소 콘텍트 스위치동안에 또한 수행될 수 있다. 대응 부동점 데이터가 유효 데이터를 포함하고 있다는 것을 지시하는 각각의 태그에 대해, 그 대응 부동점 레지스터가 저장될 것이다. 추가로, 이러한 상황에서, 부동점 레지스터 수의 지시는 저장하는데 또한 필요하다.

부동점 명령의 제 1 세트(380)의 실행 다음에, 압축 데이터 명령의 제 2 세트(382)는 실행 스트림으로 실행된다. 각각의 압축 데이터 명령의 실행은 이행 명령 세트(390)가 실행되지 않으면, 모든 압축 데이터 태그(330)를 간격(386)에서 비공백 상태로 설정되게 한다는 것을 상기하라.

어떠한 태스크 스위치도 일어나지 않으면, 압축 데이터 명령 세트(382)의 실행 다음에, 이행 명령 세트(390)가 실행된다. 이러한 이행 명령 세트(390)는 압축 데이터 상태를 저장하도록 실행될 수 있다. 이것은 상기된 바와 같이 종래의 부동점 명령, 또는 압축 데이터 상태만을 저장하는 전용 명령을 포함한 임의의 메카니즘을 이용하여 수행될 수 있다. 그 압축 데이터 상태는 부분 및 최소 콘텍트 스위칭 메카니즘을 포함한 종래의 방식으로 저장될 수 있다. 압축 데이터 상태가 저장되거나 되지 않거나, 이행 명령 세트(390)는 압축 데이터 상태를 비어 둔다. 이 이벤트에서, 압축 데이터 상태는 압축 데이터 태그(330)와 대응 에일리어스된 부동점 태그(320)에 영향을 미친다. 앞서 설명된 바와 같이, 압축 데이터 상태를 비어두는 것은, 아래의 도14를 참조하여 설명된 바와 같이, 일련의 부동점 연산 또는 단일 명령 EMMS 의 실행에 의해 수행된다. 결과적으로, 프로세서는 압축 데이터 상태를 간격(388)에서 비어 두고, 부동점 명령의 실행을 위해 초기화된다.

이행 명령 세트(390)의 실행 다음에, 부동점 명령의 제 2 세트(384)가 실행된다. 태그는 비어져 있고 스택 지시의 상부는 제 2 간격(388)동안에 제 1 물리 레지스터(0)에서의 포인트로 바뀌어져 있기 때문에,

모든 부동점 레지스터는 이용가능하다. 이것은 부동점 명령을 실행하는 것과 동시에 발생될 수 있는 부동점 스택 오버플로 예외의 발생을 방지한다. 일부 소프트웨어 구현에 있어서, 스택 오버플로 조전은 인터럽트 핸들러가 압축 데이터 상태를 저장하고 비어두게 할 수 있다. 따라서, 본 발명의 구현 실시예에서, 혼합된 압축 데이터와 부동점 명령의 블록이 허용된다. 그러나, 적절한 부기는 응용 프로그래머 또는 협력 다중 작업에 의해 수행되어야 하고, 태스크 상태가 이행동안에 훼손되지 않게 하기 위해서, 압축 데이터 명령과 부동점 명령사이의 이행동안에 소정의 부동점 또는 압축 데이터 상태를 저장한다. 추가로, 본 방법은 본 발명의 구현된 실시예를 이용하여, 권하지 않은 프로그래밍 기술을 사용하여 일어날 수 있는 불필요한 예외를 피할 수 있다.

EMMS 명령은 압축 데이터 명령 스트림과 부동점 명령 스트림사이에서의 자연스러운 이행을 가능하게 한다. 앞서 설명된 바와 같이, 일어날 수 있는 부동점 오버플로 조건을 피하기 위해, 더욱이 스택 필드(350)의 상부에 저장된 스택 지시의 상부를 리셋하기 위해 부동점 태그를 소거한다. 이러한 연산을 수행하는 전용 명령이 구현될지라도, 본 발명의 범위내에서, 이러한 연산이 현재의 부동점 명령의 결합을 이용하여 구현될 수 있다는 것이 기대된다. 이러한 예가 도14에 도시되어 있다. 더욱이, 이것은 기능적으로 압축 데이터 명령의 실행다음의 제 1 부동점 명령의 실행으로 중복될 수 있다. 이 실시예에서, 압축 데이터 명령의 실행 다음의 제 1 부동점 명령의 실행(부동점/압축 데이터 상태의 환경을 저장한 것이외)은 프로세서가 암시 EMMS 연산을 수행하게 할 수 있다(모든 태그를 공백 상태로 설정).

도5는 본 발명의 일실시예에 따라 전형적인 컴퓨터 시스템(500)을 예시하는 블록도이다. 전형적인 컴퓨터 시스템(500)은 프로세서(505), 기억 디바이스(510), 및 버스(515)를 포함하고 있다. 프로세서(505)는 버스(515)에 의해 기억 디바이스(510)에 연결되어 있다. 추가로, 키보드(520) 및 디스플레이(525)와 같은 다수의 사용자 입력/출력 디바이스는 또한 버스(515)에 연결되어 있다. 네트워크(530)는 버스(515)에 또한 연결될 수 있다. 프로세서(505)는 CISC, RISC, VLIW, 또는 하이브리드 아키텍처와 같은 어느 하나의 아키텍처 타입의 중앙 처리 장치를 나타낸다. 추가로, 프로세서(505)는 하나이상의 칩에 구현될 수 있다. 기억 디바이스(510)는 데이터를 저장하는 하나이상의 메카니즘을 나타낸다. 예를 들어, 기억 디바이스(510)는 판독 전용 메모리(ROM), 임의 접근 메모리(RAM), 자기 디스크 기억 매체, 광학 기억 매체, 플래시 메모리 디바이스, 및/또는 다른 기계 판독가능 매체를 포함할 수 있다. 버스(515)는 하나이상의 버스(예, PCI, ISA, X-버스, EISA, VESA 등)와 브리지(버스 제어기라고 함)를 나타낸다. 이 실시예는 단일 프로세서 컴퓨터 시스템에 관하여 설명되어 있지만, 본 발명은 멀티 프로세서 컴퓨터 시스템에 구현될 수 있다. 추가로, 이 실시예는 32 비트와 64 비트 컴퓨터 시스템에 관하여 설명되어 있지만, 본 발명의 구현은 이러한 컴퓨터 시스템에 제한을 두지 않는다.

도5는, 프로세서(505)는 버스 유닛(545), 캐시(550), 명령 세트 유닛(560), 메모리 관리 유닛(565), 및 이벤트 조정 유닛(570)을 포함하고 있다는 것을 추가로 예시하고 있다. 물론, 프로세서(505)는 본 발명의 구현을 이해하는데 필요하지 않은 추가 회로 소자를 포함하고 있다.

버스 유닛(545)은 캐시(550)에 연결되어 있다. 버스 유닛(545)은 프로세서(505)외부에 발생된 신호의 값을 구하고 모니터하는데, 그리고 프로세서(505)내의 다른 유닛과 메카니즘으로부터의 입력 신호와 내부 요구에 응답하여 출력 신호를 좌표화하는데 사용된다.

캐시(550)는 프로세서(505)에 대하여 사용하기 위한 하나이상의 기억 영역을 명령 캐시와 데이터 캐시로서 나타낸다. 예를 들어, 일실시예에서, 캐시(550)는 명령용과 데이터용의 두 개의 별개의 캐시로서 구현된다. 그 캐시(550)는 명령 세트 유닛(560)과 메모리 관리 유닛(565)에 연결되어 있다.

명령 세트 유닛(560)은 적어도 하나의 명령 세트를 디코드하고 실행하기 위해 하드웨어 및/또는 펌웨어를 포함하고 있다. 도5에 도시된 바와 같이, 명령 세트 유닛(560)은 디코드/실행 유닛(575)을 포함하고 있다. 이 디코드 유닛은 프로세서(505)에 의해 수시된 명령을 제어 신호 및/또는 마이크로코드 엔트리 포인트로 디코딩하는데 사용된다. 이러한 제어 신호 및/또는 마이크로코드 엔트리 포인트에 응답하여, 실행 유닛은 적절한 연산을 수행한다. 그 디코드 유닛은 다수의 상이한 메카니즘(예, 룩-업 테이블, 하드웨어 구현물, PLA 등)을 이용하여 구현될 수 있다. 디코드 및 실행 유닛의 여러 명령 실행은 여기서는 일련의 if/then 명령문에 의해 표시되지만, 그 명령 실행은 if/then 명령문의 연속 처리를 필요로 하지 않는다. 오히려, 이 if/then 명령문을 논리적으로 수행하는 임의의 메카니즘이 본 발명의 구현 범위내에서 고려된다.

디코드/실행 유닛(575)은 압축 데이터 명령을 포함한 명령 세트(580)를 포함하여 도시되어 있다. 이러한 압축 데이터 명령은 다수의 상이한 연산을 수행하도록 구현될 수 있다. 예를 들어, 이러한 압축 데이터 명령은, 실행될 때, 프로세서가 압축 부동점 연산 및/또는 압축 정수 연산을 수행하게 할 수 있다. 일실시예에서, 이러한 압축 데이터 명령은 1995년 8월 31일에 출원된 08/521,360의 '압축 데이터를 통해 연산하는 명령 세트'에 설명되어 있다. 압축 데이터 명령에 추가로, 그 명령 세트(580)는 현재의 보통 프로세서에 있는 것과 같거나 유사한 명령 및/또는 새로운 명령을 포함하고 있다. 예를 들어, 일실시예에서의 프로세서(505)는 펜티엄 프로세서와 같은 현재의 프로세서에 의해 사용된 인텔 프로세서 아키텍처 명령과 호환할 수 있는 명령을 지원한다.

도5는 또한 메모리 유닛(585)을 포함한 명령 세트 유닛(560)을 도시하고 있다. 메모리 유닛(585)은, 부동점 데이터, 압축 데이터, 정수 데이터, 및 제어 데이터(예, EM 지시, TS 지시, 스택 지시의 상부 등)를 포함한 정보를 저장하는 프로세서(505)에서 하나이상의 레지스터 세트를 나타낸다. 특정 실시예에서, 여기서 일부는 추가 설명되어 있고, 메모리 유닛(585)은 부동점 상태에서 압축 데이터 상태를 에일리어스한다.

메모리 관리 유닛(565)은 페이징 및/또는 세그먼트 기법과 같은 하나이상의 메모리 관리법을 구현하는 하드웨어 및 펌웨어를 표시한다. 메모리 관리법중 일부가 사용될 수 있지만, 인텔 프로세서 아키텍처와 호환성이 있는 일실시예에서의 메모리 관리법이 구현된다. 이벤트 핸들링 유닛(570)은 메모리 관리 유닛(565)과 명령 세트 유닛(560)에 연결되어 있다. 그 이벤트 핸들링 유닛(570)은 하나이상의 이벤트 핸들링법을 구현하는 하드웨어 및 펌웨어를 나타낸다. 이벤트 핸들링법의 일부가 사용될 수 있지만, 인텔 프로세서 아키텍처와 호환성이 있는 일실시예에서의 이벤트 핸들링법이 구현된다.

도5는, 기억 디바이스(510)가 컴퓨터 시스템(500)에 실행하기 위해 압축 데이터 루틴(540)과 운영 체제(535)를 저장하고 있음을 예시하고 있다. 압축 데이터 루틴(540)은 하나이상의 압축 데이터 명령을 포함하고 있는 명령의 시퀀스이다. 물론, 기억 디바이스(510)는, 본 발명을 이해하는데 필요하지 않은 추가 소프트웨어(도시생략)를 바람직하게 포함하고 있다.

일실시예에서의 여러 지시는 프로세서(505)에서 레지스터내의 비트를 사용하여 구현되지만, 대체 실시예는 여러 기술을 사용할 수 있다. 예를 들어, 대체 실시예는 이러한 지시를 칩으로부터 떨어져 저장될 수 있고(기억 디바이스(510)내에), 또는 각각의 지시에 대하여 다중 비트를 사용할 수 있다. 기억 영역의 용어는 여기서 기억 디바이스(510)내의 위치, 프로세서(505)내의 하나이상의 레지스터 등을 포함하여, 데이터를 저장하는 임의의 메카니즘으로서 사용된다.

도6A는 본 발명의 일실시예에 따른 두 개의 별개 물리 레지스터 파일을 이용하여 압축 데이터 레지스터 상태를 부동점 상태에서 에일리어스하는 장치를 예시하는 블록도이다. 이러한 두 개의 물리 레지스터 파일이 에일리어스되기 때문에, 단일 논리 레지스터 파일로서 프로세서에서 실행하는 소프트웨어에 논리적으로 나타난다. 도6A는 이행 유닛(600), 부동점 유닛(605), 및 압축 데이터 유닛(610)을 도시하고 있다. 부동점 유닛(605)은 도1의 부동점 유닛(135)과 유사하다. 부동점 유닛(605)은 부동점 레지스터 세트(615), 태그 세트(620), 부동점 상태 레지스터(625), 및 부동점 스택 레퍼런스 유닛(630)을 포함하고 있다. 일실시예에서, 부동점 유닛(605)는 8개의 레지스터(R0에서 R7로 표시)를 포함하고 있다. 이 8개의 레지스터의 각각은 80 비트 폭이고, 부호 필드, 지수 필드, 및 소수부 필드를 포함하고 있다. 부동점 스택 레퍼런스 유닛(630)은 스택으로서 부동점 레지스터 세트(615)를 동작시킨다. 부동점 상태 레지스터(155)는 스택 지시의 상부를 저장하는 스택 필드의 상부(635)를 포함하고 있다. 앞서 설명된 바와 같이, 스택 지시의 상부는 부동점 레지스터 세트(615)내의 어느 레지스터가 현재의 부동점 스택의 상부인지를 식별한다. 도6A에서, 스택 지시의 상부는 스택의 상부-ST(0)로서 물리 위치(R4)에서 레지스터(640)를 식별한다.

일실시예에서, 태그 세트(620)는 8개의 태그를 포함하고 있고, 단일 레지스터에 저장되어 있다. 각각의 태그는 상이한 부동점 레지스터에 대응하고, 2 비트로 구성되어 있다. 대안으로, 태그의 각각은, 결과적으로 에일리어싱을 형성하는 논리 레지스터 파일내의 상이한 레지스터에 대응하는 것으로서 생각될 수 있다. 도6A에서 보는 바와 같이, 태그(645)는 레지스터(640)에 대응한다. 앞서 설명된 바와 같이, 이 태그는 공백과 비공백 레지스터 위치를 구별하기 위해 부동점 유닛(605)에 사용된다. 앞서 설명된 바와 같이, 실시예는 공백 또는 비공백 상태중 하나를 식별하는 1 비트 태그를 사용할 수 있지만, 태그값이 필요할 때 적절한 2 비트 태그값을 결정함으로써 2 비트를 구성하는 바와 같이 이 1비트 태그를 소프트웨어에 나타나게 한다. 물론, 대체 실시예는 2 비트 태그를 구현할 수 있다. 어느 것이든, 태그는 두 가지 상태: 11로 표시된 공백 상태와 00,01,10중 하나로 표시된 비공백 상태를 식별하는 것으로서 생각될 수 있다.

압축 데이터 유닛(610)은 압축 데이터를 저장하는데 사용되고, 압축 데이터 레지스터(압축 데이터 레지스터 파일이라 함) 세트(650), 압축 데이터 상태 레지스터(655), 및 압축 데이터 비스택 레퍼런스 유닛(660)을 포함하고 있다. 일실시예에서, 압축 데이터 레지스터 세트(650)는 8개의 레지스터를 포함하고 있다. 이 8개의 레지스터의 각각은 부동점 레지스터 세트(615)내의 상이한 레지스터에 대응한다. 8개의 압축 데이터 레지스터의 각각은 64 비트폭이고, 이것이 대응하는 부동점 레지스터의 64 비트 소수부 필드상에서 매핑된다. 압축 데이터 비스택 레퍼런스 유닛(660)은 고정 레지스터 파일로서 압축 데이터 레지스터(650)를 동작시킨다. 따라서, 압축 데이터 명령은 압축 데이터 레지스터 세트(650)내의 어느 레지스터가 이용되는지를 명백하게 표시한다.

이행 유닛(600)은, 이 두 개의 물리 레지스터 파일사이에서 데이터를 복사함으로써 압축 데이터 레지스터(650)를 부동점 레지스터(615)상에서 에일리어스한다. 따라서, 이행 유닛(600)은 물리 부동점 레지스터(615)와 물리 압축 데이터 레지스터(650)이 단일 논리 레지스터 파일로서 사용자/프로그래머에게 논리적으로 나타나게 한다. 이러한 방식으로, 이것은 단일 논리 레지스터만이 부동점과 압축 데이터 명령을 실행하는데 이용가능한 것처럼 소프트웨어에 나타난다. 이행 유닛(600)은 하드웨어 및/또는 마이크로코드를 포함하여, 다수의 기술을 이용하여 구현될 수 있다. 물론, 대체 실시예에서, 이행 유닛(600)은 프로세서상의 어느 곳에서도 위치될 수 있다. 더욱이, 대체 실시예에서, 이행 유닛(600)은 프로세서의 외부에 저장된 비마이크로코드 이벤트 핸들러가 될 수 있다.

이행 유닛(600)은 완전 또는 부분 에일리어싱을 위해 제공되도록 구현될 수있다. 모든 물리 부동점 레지스터의 내용은 압축 데이터 모드에서의 이행동안에 압축 데이터 레지스터 파일에 복사된다면, 물리 부동점 레지스터 파일은 압축 데이터 레지스터 파일에서 완전히 에일리어스된다. 유사하게, 모든 물리 압축 데이터 레지스터의 내용은 부동점 모드에서의 이행동안에 부동점 레지스터 파일에 복사된다면, 물리 압축 데이터 레지스터 파일은 물리 부동점 레지스터 파일에서 완전히 에일리어스된다. 그 반대로, 부분 에일리어스하면, '유용한' 데이터를 포함하고 있는 그 레지스터만의 내용이 복사된다. 임의수의 기준을 근거로 하여 어느 레지스터가 유용한 데이터를 포함하고 있는지를 판단할 수 있다. 예를 들어, 부분 에일리어싱은, 대응 태그가 비공백 상태를 지시하는 물리 부동점 레지스터에만 저장된 데이터를 물리 압축 데이터 레지스터로 복사함으로써 구현될 수 있다. 물론, 실시예는 압축 데이터 명령을 실행할 때 부동점 태그를 사용할 수 있거나, 물리 부동점 레지스터에서 물리 압축 데이터 레지스터를 부분적으로 에일리어스하는 별개의 압축 데이터 태그를 포함할 수 있다. 대안으로, 터치(판독 및/또는 기록)된 그 압축 데이터 레지스터 및/또는 부동점 레지스터는 유용한 데이터가 포함되어 있는 것으로 간주될 수 있다. 부동점 태그는 공백 또는 비공백을 추가로 지시하는데 사용되기 보다는 이러한 목적에 사용될 수 있다. 대안으로, 추가 지시는 어느 레지스터가 터치되었는지를 기록하는 부동점 및/또는 압축 데이터 레지스터에 포함될 수 있다. 부분적인 에일리어싱을 구현할 때, 좋은 프로그래밍 기술은, 데이터가 이행동안에 복사되지 않은 레지스터는 미정의된 값을 포함하도록 간주되어야 하는 것을 가정하는 것이다.

압축 데이터 상태 레지스터(655)는 압축 데이터 더티(dirty) 필드(665), 추리(speculative) 필드(670), 모드 필드(675), 예외 상태 필드(680), 및 EMMS 필드(685)를 포함하고 있다. 압축 데이터 더티 필드(665)의 각각은 압축 데이터 레지스터(650)의 상이한 하나에 대응하고, 더티 지시를 저장하는 데 사용된다. 압축 데이터 레지스터(650)와 부동점 레지스터(615)사이에 대응 관계가 있기 때문에, 더티 지시의 각각은 상이한 하나의 부동점 레지스터(615)와 대응 관계를 가지고 있다. 하나의 값은 압축 데이터 레지스터

(650)의 하나에 기록될 때, 더티 지시에 대응하는 그 레지스터는 더티 상태를 지시하도록 바뀌게 된다. 이행 유닛(600)이 압축 데이터 유닛(610)에서 부동점 유닛(605)로 이행하게 될 때, 대응하는 더티 지시가 더티 상태를 지시하는 그 부동점 레지스터(615)의 부호 필드와 지수 필드에 1이 기록된다. 이러한 방식으로, 도4B의 단계(430)가 실행된다.

모드 필드(675)는 프로세서가 현재 동작하고 있는 모드, 부동점 유닛(605)가 현재 사용되고 있는 부동점 모드, 또는 압축 데이터 유닛(610)이 사용되고 있는 압축 데이터 모드를 식별하는 모드 지시를 저장하는데 사용된다. 프로세서가 부동점 모드에 있고, 압축 데이터 명령이 수신되면, 부동점 모드에서 압축 데이터 모드로의 이행이 수행되어야 한다. 그 반대로, 프로세서가 압축 데이터 모드에 있고, 부동점 명령이 수신되면, 압축 데이터 모드에서 부동점 모드로의 이행이 수행되어야 한다. 따라서, 압축 데이터 또는 부동점 명령중 어느 하나를 수신할 때, 그 모드 지시는 이행이 필요한지를 판단하기 위해 폴링될 수 있다. 이행이 필요하다면, 이행이 수행되고, 따라서 그 모드 지시는 바뀌게 된다. 모드 지시의 연산은 도7A-9를 참조하여 추가 설명될 것이다.

예외 상태 필드(680)는 예외 상태 지시를 저장하는데 사용된다. 그 예외 상태 지시는, 앞의 부동점 명령의 실행으로부터의 보류 예외가 있는지를 식별하기 위해 압축 데이터 명령의 실행동안에 사용된다. 일실시예에서, 그 예외 상태 지시가 이러한 예외가 보류되어 있음을 지시하면, 그 예외는 압축 데이터 모드로의 이행에 앞서 제공된다. 일실시예에서, 이러한 목적으로 부동점 유닛(605)에 의해 사용된 지시는 예외 상태 지시로서 그 예외 상태 필드에 직접 복사되거나 인코드된다.

EMMS 필드(685)는, 실행된 최종 압축 데이터 명령이 EMMS 명령이었는지를 식별하는 EMMS 지시를 저장하는데 사용된다. 일실시예에서, EMMS 명령이 실행되면, 그 EMMS 지시는, 실행된 최종 압축 데이터 명령이 EMMS 명령이었음을 지시하기 위해 1로 바뀌게 된다. 그 반대로, 모든 압축 데이터 명령이 실행될 때, EMMS 지시는 0으로 바뀌게 된다. 이행 유닛(600)은, 최종 압축 데이터 명령이 EMMS 명령이었는지를 판단하기 위해, 압축 데이터 모드에서 부도 소수점 모드로 이행될 때 EMMS 지시를 폴링한다. 최종적으로 실행된 압축 데이터 명령이 EMMS 명령이었다면, 이행 유닛(600)은 모든 태그(620)를 공백 상태로 바꾼다. 그러나, 그 EMMS가, 최종적으로 실행된 압축 데이터 명령이 EMMS가 아님을 지시하면, 그 이행 유닛(600)은 모든 태그(620)를 비공백 상태로 바꾼다. 이러한 방식으로, 태그는 도4B의 단계(432)에서 단계(440)로 유사하게 바뀌게 된다. 추리 필드(670)는 부동점 모드에서 압축 데이터 모드로의 이행이 추리적인지를 식별하는 추리 지시를 저장하는데 사용된다. 그 이행이 추리적이다면, 부동점 유닛(605)으로의 역이행이 필요하든지 시간은 저장될 수 있다. 그 모드 지시의 연산은 도7A-9를 참조하여 추가 설명될 것이다.

도6B는 본 발명의 실시예에 따른 도6A의 부동점 레퍼런스 파일의 부분 확대를 예시하는 블록도이다. 도6B는 태그 세트(620)에서 태그를 선택적으로 바꾸는 태그 수정자를 포함하고 있는 부동점 스택 레퍼런스 유닛(630)을 도시하고 있다. 도6B에 도시된 실시예에서, 태그 세트(620)의 각각은 공백인지 비공백인지를 지시하는 1 비트만을 포함하고 있다. 그 태그 수정자 유닛(690)은 TOS 조정 유닛 세트(696)와 체크/수정 유닛(698)을 포함하고 있다. TOS 조정 유닛(696)의 각각은 구현에 따라 하나이상의 마이크로 연산을 수신하는 마이크로 연산 라인(692)에 연결되어 있다(예, 단지 마이크로 연산만을 수신하는 하나의 TOS 조정 유닛이 될 수 있다). 태그를 바꿀 필요가 있는 부동점 명령에서의 최소한의 마이크로 연산은 TOS 조정 유닛(696)에 의해 수신된다. 물론, 부동점 스택 레퍼런스 유닛(630)은 각각의 마이크로 연산의 모두 또는 관련 부분만이 TOS 조정 유닛(696)에 의해 수신되도록 구현될 수 있다.

마이크로 연산을 수신하는 것에 응답하여, TOS 조정 유닛은 체크/수정 유닛(698)에: 1)마이크로 연산에 의해 식별된 태그 세트(620)내의 태그의 어드레스; 및 2) 그 태그상에서 수행되는 동작을 지시하는 신호(예, 0 또는 1로 바뀌고, 폴링된다)를 전송한다. 태그의 폴링은 본 발명을 이해하는데 필요하지 않기 때문에, 여기서 추가 설명되지 않는다. TOS 조정 유닛(696)의 각각은 현재의 TOS 값을 수신하고 따라서 그 태그 어드레스를 조정하는 라인(694)에 또한 연결되어 있다. 체크/수정 유닛(698)은 적어도 하나의 기록 라인에 의해 태그(620)의 각각에 연결되어 있다. 예를 들어, 체크/수정 유닛(698)은 하나의 기록 라인에 의해 태그(645)에 연결되어 있다. 태그 어드레스를 수신하고 신호를 대응시키는 것에 응답하여, 체크/수정 유닛(698)은 필요한 체크 및/또는 수정을 수행한다. 다중 마이크로 연산이 동시에 수신될 수 있는 구현에 있어서, 체크/수정 유닛(698)은 마이크로 연산이 동일 태그를 수정하고 있는지를 판단하기 위해 그 마이크로 연산을 또한 비교한다(예, 마이크로 연산 1은 1로 바뀐 태그 1을 필요로 하고, 마이크로 연산 하나와 동일한 시간에 수신된 마이크로 연산 2는 0으로 바뀐 태그 1을 필요로 한다고 가정하자). 동일 태그가 수정되어 있다면, 체크/수정 유닛(698)은 어느 마이크로 연산이 최종적으로 실행되는지를 판단하고, 그 마이크로 연산에 따라 태그를 바꾼다. 상기 예에서, 마이크로 연산 2가 마이크로 연산 1다음에 실행된다고 가정하면, 체크/수정 유닛(698)은 태그1을 0으로 바꾼다.

예를 들어, 공백 상태로 바뀐 태그(예, 태그(645))를 필요로 하는 부동점 연산이 실행된다면, TOS 조정 유닛은 태그를 식별하는 마이크로 연산 라인을 통해 마이크로 연산과 현재의 TOS 값을 수신할 것이다. TOS 조정 유닛은 그 태그(예, 태그(645))의 어드레스를 결정할 수 있고, 태그가 공백 상태로 바뀔 수 있음을 지시하는 신호뿐만 아니라 그 어드레스를 체크/수정 유닛(698)에 전송할 수 있다. 응답으로, 체크/수정 유닛(698)은 태그(645)에 연결된 기록 라인을 통해 0을 전송함으로써 그 태그(645)를 공백 상태로 바꿀 수 있다.

일실시예에서, 부동점 명령은 모든 태그가 동시에 수정될 필요가 없도록 구현될 수 있기 때문에, 태그 수정자 유닛(690)은 모든 태그를 동시에 수정될 수 없도록 구현된다. 회로의 복잡성을 피하기 위해, 부동점 모드로의 이행에 대한 태그의 글로벌 바꿈은 이러한 현재의 메카니즘을 이용하여 구현될 수 있다. 이점에 관하여, 이행 유닛(600)이 마이크로코드에서 구현된다면, 마이크로코드 명령 세트는 디코드 유닛이 8개의 태그를 바꾸는 현재의 일부 마이크로 연산을 나타나게 한다. 따라서, EMMS 지시가 EMMS 명령은 최종적으로 실행된 압축 데이터 명령이었음을 지시하는 동안에 압축 데이터 모드로의 이행을 수행하는 것에 반응하여, 디코드 유닛은 이행 유닛(600)에 접근할 수 있고, 현재의 일부 마이크로 연산을 나타나게 할 수 있다. 이러한 마이크로 연산에 응답하여, 태그 수정자 유닛(690)은 그 대응 태그를 공백 상태로 수정할 수 있다. 그 반대로, EMMS 지시가 EMMS 명령은 최종적으로 실행된 압축 데이터 명령이 아님을 지시하는 동안에 압축 데이터 모드로의 이행을 수행하는 것에 반응하여, 디코드 유닛은 이행 유닛(600)에 접근할 수 있

고, 태그 수정자 유닛(690)이 태그의 각각을 비공백 상태로 바꾸게 할 수 있는 현재의 일부 마이크로 연산을 발생시킬 수 있다. 이러한 실시예에서, 태그의 글로벌 바꿈은 약 4-8 클록 사이클을 필요로 할 수 있다.

일실시예는 압축 데이터 모드로의 이행에 반응하여 모든 태그를 바꾸는 것에 대하여 설명되었지만, 대체 실시예는 임의 수의 메카니즘을 사용할 수 있다. 예를 들어, 모든 태그를 공백 상태 또는 비공백 상태로 바꾸는 것은, 새로운 마이크로 연산에 반응하는 태그를 글로벌로 바꿀 수 있도록 태그 수정자 유닛(690)을 구현하고 새로운 마이크로 연산을 포함시킴으로써 단일 클록 사이클로 완료될 수 있다. 이 실시예에서, 이행 유닛(600)은, 모든 태그를 공백 상태 또는 비공백 상태로 바꾸기 위해 디코드 유닛이 이러한 단일 마이크로 연산(일부 분할 마이크로 연산보다는)을 발생시키도록 구현될 수 있다. 다른 예로서, 디코드 유닛은 태그(620)에 연결될 수 있고, EMMS 명령을 수신하는 것에 반응하여 모든 태그(620)를 바꾸는 추가 하드웨어를 포함할 수 있다.

앞서 설명된 바와 같이, 태그 세트(620)이 1 비트 태그를 가지고 있는 것처럼 설명되었지만, 태그 세트(620)는 각각의 태그에 2 비트가 있는 것처럼 나타나게 만들어질 수 있다. 대체 실시예는 태그가 바뀌게 되는 여러 상태(예, 00,01,10,11)를 지시하는 추가 인코드 라인 또는 비인코드 라인을 포함함으로써 각각의 태그에 대해 2 비트를 구현할 수 있다.

도7A,7B,7C,8, 및 도9는 본 발명의 일실시예에 따라서, 보이지 않는 시스템을 동작시키고, 좋은 프로그래밍 실행을 진전시키고, 그리고 도6A의 하드웨어 배열을 이용하여 실행될 수 있는 방식으로 부동점 레지스터 세트상에서 에일리어스되는 레지스터 세트를 통해 압축 데이터 명령을 실행하는 방법을 예시하고 있다. 이 흐름도는 도4A와 도4B를 참조하여 설명된 흐름도와 유사하다. 도4A와 도4B를 참조하여, 단계를 바꾸고, 이동시키고 또는 제거하는 다수의 대체 실시예가 설명되어 있다. 도4A와 도4B에서 수행된 단계와 유사한 도7A,7B,7C,8 및 도9를 참조하여 설명된 단계는 이러한 대체 실시예를 이용하여 최소한 수행될 수 있다는 것을 알 수 있다. 그 흐름도는 단계(700)에서 시작한다. 단계(700)에서 단계(702)로 진행한다.

단계(702)에서 보는 바와 같이, 비트 세트는 하나의 명령으로서 접근되고 단계(704)로 진행한다. 이러한 비트 세트는 그 명령에 의해 수행되는 연산을 식별하는 연산 코드를 포함하고 있다. 따라서, 단계(702)는 도4A의 단계(402)와 유사하다.

단계(704)에서, 연산 코드가 유효한지를 판단한다. 연산 코드가 유효하지 않으면, 단계(706)로 진행한다. 그 반대면, 단계(708)로 진행한다. 단계(704)는 도4A의 단계(404)와 유사하다.

단계(706)에서 보는 바와 같이, 무효 연산 코드 예외가 발생되고, 적절한 잉벤트 핸들러가 실행된다. 따라서, 단계(706)는 도4A의 단계(406)와 유사하다.

단계(708)에서, 어떤 타입의 명령이 수신되었는지를 판단한다. 그 명령이 부동점 명령도 압축 데이터 명령도 아니면, 단계(710)로 진행한다. 그러나, 그 명령이 부동점 명령이면, 단계(712)로 진행한다. 그 반대로, 그 명령이 압축 데이터 명령이면, 단계(714)로 진행한다. 따라서, 단계(708)는 도4A의 단계(408)와 유사하다.

단계(710)에서 보는 바와 같이, 프로세서는 그 명령을 실행한다. 이 단계는 본 발명을 이해하는데 필요하지 않기 때문에, 추가 설명되어 있지 않다. 단계(710)는 도4A의 단계(410)와 유사하다.

단계(712)에서 보는 바와 같이, EM 지시가 1인지(설명된 소프트웨어 규정에 따라, 부동점 유닛이 대리 실행되어야 한다면), 그리고 TS 지시가 1인지(설명된 소프트웨어 규정에 따라, 부분 콘텍트 스위치가 수행되었다면)를 판단한다. EM 지시가 1이고 또는 TS 지시가 1이면, 단계(716)로 진행한다. 그 반대면, 단계(720)로 진행한다. 따라서, 단계(712)는 도4A의 단계(412)와 유사하다.

단계(716)에서, 예외를 이용할 수 없는 디바이스가 발생되고, 그 대응 이벤트 핸들러가 실행된다. 따라서, 단계(716)는 도4A의 단계(416)와 유사하다. 앞서 설명된 바와 같이, 이러한 이벤트 핸들러는 부동점 명령 및/또는 부분 콘텍트 스위치가 수행되었는지를 판단하는 EM 및 TS 지시를 사용하도록 구현될 수 있다.

단계(714)에서, EM 지시가 1인지를 판단한다. 따라서, 단계(714)는 도4A의 단계(414)와 유사하다. 결과적으로, 단계(714)에서 EM 지시가 1이다고 판단되면, 단계(716)보다는 단계(706)로 진행한다. 그 반대면, 단계(718)로 진행한다.

앞서 설명된 바와 같이, 단계(706)에서, 유효 연산 코드 예외가 발생되고, 그 대응 이벤트 핸들러가 실행된다. 유효 연산 코드 예외에서 EM=1일 때 압축 데이터 명령의 시도된 실행을 분할함으로써, 그 실시예는 도4A의 단계(406)를 참조하여 앞서 설명된 바와 같이 보이지 않는 시스템을 동작하고 있다.

일실시예는 보이지 않은 시스템을 동작하는 방식으로 EM 지시를 핸들링하는 것에 대해 설명되었지만, 대체 실시예는 다른 기술을 사용할 수 있다. 예를 들어, 대체 실시예는 예외를 이용할 수 없는 디바이스, 현재의 상이한 이벤트, 또는 EM 지시가 1일 때 압축 데이터 명령의 시도된 실행에 반응한 새로운 이벤트 중 어느 하나를 발생시킨다. 다른 예로서, 대체 실시예는 압축 데이터 명령이 실행될 때 EM 지시를 무시할 수 있다.

단계(718)에서 보는 바와 같이, TS 지시가 1인지를 판단한다(설명된 소프트웨어 규정에 따라, 부분 콘텍트 스위치가 수행되었다면). TS 지시가 1이다면, 단계(716)로 진행한다. 그 반대면, 단계(722)로 진행한다. 따라서, 단계(718)은 도4A의 단계(418)와 유사하다.

앞서 설명된 바와 같이, 단계(716)에서 예외를 이용할 수 없는 디바이스가 생성되고, 그 대응 이벤트 핸들러는 실행된다. 단계(716)는 도4A의 단계(418)와 유사하다. 단계(714)는 EM 지시가 무효 연산코드 예외에서 1인 상황으로 분할되어 있기 때문에, EM 지시는 0이여야 하고, TS 지시는 1이어야 한다. TS 지시가 1이기 때문에, 그 이벤트 핸들러는 프로세서가 부분 콘텍트 스위치(부동점 유닛의 내용을 저장하고 필요하다면 그 올바른 부동점 상태를 재저장한다)를 참조하여 앞서 설명된 바와 같이 작용하게 하고, 프로세서가 단계(702)에서 수신된 명령의 실행을 재시작함으로써 실행을 재개시하게 한다. 압축 데이터 상태는

부동점 상태에서 에일리어스되기 때문에, 이 이벤트 핸들러는 부동점과 압축 데이터 상태 모두에 작용한다. 결과적으로, 이 방법은 보이지 않은 시스템을 동작시킨다. 물론, 대체 실시예는 여러 방법으로 이 이벤트 핸들러를 구현할 수 있다.

일실시예는 보이지 않은 시스템을 동작하는 방식으로 TS 지시를 핸들링하는 것에 대해 설명되었지만, 대체 실시예는 다른 기술을 이용할 수 있다. 일예에서, 대체 실시예는 TS 지시를 구현할 수 없다. 이러한 대체 실시예는 부분 콘텍트 스위칭을 구현하기 위해 TS 지시를 이용하는 운영 체제와 호환될 수 없다. 그러나, 이러한 실시예는 TS 지시를 이용하여 부분 콘텍트 스위칭을 지원하지 않는 현재의 운영 체제와 호환할 수 있다. 다른 예로서, TS 지시가 1일 때 압축 데이터 명령의 시도된 실행은 새로운 이벤트 핸들러로 분할될 수 있거나, 수정되어진 현재의 이벤트 핸들러로 분할될 수 있다. 이 이벤트 핸들러는 이러한 상황에 반응하여 임의의 작동이 적절한 것으로 간주되도록 구현될 수 있다. 예를 들어, 압축 데이터 상태가 부동점 상태에서 에일리어스되는 일실시예에서, 이 이벤트 핸들러는 압축 데이터 상태 및/또는 부동점 상태를 저장할 수 있다.

앞서 설명된 바와 같이, 부동점 명령의 실행동안에 특정 숫자 오류가 발생되면, 이 오류는, 실행은 오류를 제공하는 것이 인터럽트될 수 있는 다음 부동점 명령의 실행이 시도될 때까지 보류된다. 앞서 설명된 바와 같이, 도4의 단계(420)와 단계(422)모두에서, 제공될 수 있는 이러한 보류 오류가 있는지를 판단한다. 도4A의 단계(420)와 유사하게, 단계(720)에서, 제공될 수 있는 이러한 보류 오류가 있는지를 판단한다. 이러한 보류 오류가 있다면, 단계(720)에서 단계(724)로 진행한다. 그러나, 단계(720)에서 이러한 보류 오류가 없다고 판단되면, 단계(726)로 진행한다. 그 반대로, 압축 데이터 명령의 시도된 실행동안에, 이전의 부동점 명령으로부터의 보류 오류가 있는지의 판단은 추후에 추가 설명되는 다른 단계에서 수행된다. 결과적으로, 단계(722)와 단계(422)는 다르다.

단계(724)에서, 보류한 부동점 오류 이벤트가 발생된다. 따라서, 단계(724)는 도4A의 단계(424)와 유사하다. 도4A의 단계(424)를 참조하여 앞서 설명된 바와 같이, 이 이벤트는 내부 또는 외부 이벤트로서 취급되고, 따라서 제공된다.

단계(726)에서 보는 바와 같이, 그 모드 지시는, 프로세서가 부동점 모드에서 동작하는 것을 지시하는지를 판단한다. 따라서, 단계(726)는 도4B의 단계(426)와 다르다. 그 프로세서가 부동점 모드에 있지 않다면, 프로세서는 부동점 명령을 실행하기 위해 압축 데이터 모드에서 부동점 모드로 이행되어야 할 것이다. 따라서, 프로세서가 부동점 모드에 있지 않다면, 단계(728)로 진행한다. 그 반대면, 단계(732)로 진행한다.

단계(728)에서, 프로세서는 압축 데이터 모드에서 부동점 모드로 이행되, 단계(730)로 진행한다. 단계(728)는 도6A의 이행 유닛(600)에 의해 수행되고, 도9를 참조하여 추가 설명될 것이다.

단계(730)에서 보는 바와 같이, 단계(702)에서 수신된 명령은 '마이크로 재시작'을 수행함으로써 재시작된다. 일실시예에서 단계(728)는 마이크로코드를 이용하여 수행되고 그 명령은 마이크로 재시작되기 때문에, 실행될 필요가 있는 운영 체제 이벤트 핸들러는 없다. 결과적으로, 현 태스크의 실행은 프로세서의 외부에서 작동이 일어남이 없이 재개시될 수 있다 -- 운영 체제 이벤트 핸들러와 같은 비마이크로코드 이벤트 핸들러는 실행될 필요가 없다. 따라서, 그 운영 체제를 포함하여, 소프트웨어에서 볼 수 없는 방식으로 프로세서는 압축 데이터 모드에서 부동점 모드로 이행할 수 있다. 이러한 방식으로, 이 실시예는 현재의 운영 체제와 호환성이 있다. 대체 실시예는 보다 적게 호환성이 있게 구현될 수 있다. 예를 들어, 추가 이벤트는 그 프로세서에 추가될 수 있고, 추가 이벤트 핸들러는 이 이행을 수행하기 위해 그 운영 체제에 더해질 수 있다.

단계(732)에서 보는 바와 같이, 부동점 명령이 실행된다. 단계(732)는 도4B의 단계(426)와 유사하다. 보이지 않을 시스템을 동작시키기 위해, 일실시예는 또한 필요한 것으로 그 태그를 바꾸고, 현재 제공될 수 있는 숫자 오류을 기록하고, 그리고 다른 숫자 오류를 보류시킨다. 앞서 설명된 바와 같이, 태그를 바꾸는 것은, 대응 태그가 비공백 상태를 지시하는 그 부동점 레지스터만의 내용을 저장하는 이러한 운영 체제 기술에서 볼 수 없는 운영 체제를 이 실시예가 유지하게 하는 것이다. 그러나, 앞서 설명된 바와 같이, 대체 실시예는 보다 적게 특정 운영 체제 기술과 호환되게 구현될 수 있다. 예를 들어, 현재의 운영 체제가 그 태그를 활용할 수 없다면, 그 태그를 구현할 수 없는 프로세서는 여전히 그 운영 체제와 호환될 수 없다. 더욱이, 본 발명에서 숫자 부동점 예외가 보류되어야 할 필요는 없고, 따라서, 그렇게 하지 않은 대체 실시예는 여전히 본 발명의 범위내에 있다.

단계(722)에서 보는 바와 같이, 프로세서가 압축 데이터 모드에 있다는 것을 지시하는 모드 지시가 있는지를 판단한다. 따라서, 단계(722)는 도4A의 단계(422)와 다르다. 단계(722)는 압축 데이터 명령을 실행하기 위해 프로세서가 적당한 모드에 있는지를 판단하도록 수행된다. 프로세서가 그 압축 데이터 모드에 있지 않다면, 프로세서는 그 압축 데이터 명령을 실행하기 위해 부동점 모드에서 압축 데이터 모드로 이행되어야 할 것이다. 따라서, 그 프로세서가 압축 데이터 모드에 있지 않다면, 단계(734)로 진행한다. 그 반대면, 단계(738)로 진행한다.

단계(734)에서, 그 프로세서는 부동점 모드에서 압축 데이터 모드로 이행되고, 단계(736)로 진행한다. 단계(734)는 도6A의 이행 유닛(600)에 의해 수행되고, 도8를 참조하여 추가 설명될 것이다.

단계(736)에서 보는 바와 같이, 단계(702)에서 수신된 명령은 마이크로 재시작을 수행함으로써 재시작된다. 따라서, 단계(736)는 단계(730)와 유사하다.

단계(738)에서, 그 압축 데이터 명령이 EMMS 명령인지를 판단한다. 그 압축 데이터 명령이 EMMS 명령이다면, 단계(740)로 진행한다. 그 반대면, 단계(742)로 진행한다. 그 압축 데이터 명령은 개별 유닛(즉, 압축 데이터 유닛)에서 실행되기 때문에, 특정 연산을 실제로 수행하는 것(예, 그 EMMS 명령을 실행하는 것에 반응하여 그 태그를 공백 상태로 바꾸는 것, 및 다른 압축 데이터 명령을 실행하는 것에 반응하여 그 태그를 비공백 상태로 바꾸는 것)보다 부동점 모드로의 역이행할 때 어느 것이 단계(728)에서 행해져야 하는지를 식별하는 지시(예, EMMS 지시)를 저장하는 것이 더 효율적이다. 다른 지시뿐만 아니라, EMMS 지시의 사용은 압축 데이터 모드에서 부동점 모드로 이행하는 단계를 참조하여 설명될 것이고, 그것은 도9

에서 추가 설명될 것이다.

단계(740)에서 보는 바와 같이, EMMS 지시는, 최종 압축 데이터 명령이 EMMS 명령이었음을 지시하는 것으로 바뀌게 된다. 단계(740)의 완료와 동시에, 프로세서는 자유롭게 다음 명령(단계(702)에서 수신된 명령을 논리적으로 뒤따르는 명령)을 실행한다.

단계(702)에서 보는 바와 같이, 압축 데이터 명령이 어떠한 숫자 예외를 발생시키지 않고 실행된다. 따라서, 단계(742)는 스택 지시의 상부가 바뀌지 않는 것을 제외하고 도4B의 단계(434)와 유사하다. 앞서 설명된 바와 같이, 보이지 않는 시스템을 완전히 동작시키지 않는 대체 실시예는, 추가 이벤트 핸들러가 그 운영 체제에 추가되거나 또는 현재의 이벤트 핸들러가 그 오류를 제공하도록 바뀌게 되도록 구현될 수 있다. 그 압축 데이터 명령을 실행하는 것을 시도하는 결과로서 한 메모리 이벤트가 발생된다면, 예외는 인터럽트되고, 그 이벤트는 제공된다.

단계(744)에서 보는 바와 같이, 추리 지시는 부동점 모드에서 압축 데이터 모드로의 이행이 더 이상 추리할 수 없음을 지시하도록 바뀌게 된다. 단계(744)에서, 단계(746)로 진행한다. 그 추리 지시의 연산은 도8을 참조하여 추가 설명될 것이다.

단계(746)에서 보는 바와 같이, 그 압축 데이터 명령은 프로세서가 어느 하나의 에일리어스된 레지스터에 기록하게 하는지를 판단한다. 그러하다면, 단계(748)로 진행한다. 그 반대면, 단계(750)로 진행한다. 따라서, 단계(746)는 도4B의 단계(736)와 유사하다.

단계(748)에서, 그 에일리어스된 레지스터의 대응 더티 지시는 그 더티 상태로 바뀌게 되고, 단계(750)로 진행한다. 이 더티 지시는 압축 데이터 모드에서 부동점 모드로 이행될 때 단계(728)에서 사용된다. 앞서 설명된 바와 같이, 이 더티 지시는 부호 필드와 지수 필드가 1로 기록되어야 하는 부동점 레지스터를 식별하는데 사용된다. 일실시예에서 부호 필드와 지수 필드에 1이 기록되는 반면에, 대체 실시예는 NAN(수가 아닌) 또는 무한대를 나타내는 값을 사용할 수 있다. 단계(746)와 단계(748)는 부호와 지수 필드가 바뀌지 않는 대체 실시예에서 필요로 될 수 없다.

단계(750)에서 보는 바와 같이, EMMS 지시는 최종 압축 데이터 명령이 EMMS 명령이 아니었음을 지시하도록 바뀌게 된다. 단계(750)의 완료와 동시에, 그 시스템은 다음 명령을 자유롭게 실행한다. 물론, EMMS 명령을 활용하지 않았던 실시예는 단계(738), 단계(740), 및 단계(750)를 필요로 하지 않는다.

따라서, 현재의 운영 체제(워싱턴 레드몬드 소재의 마이크로소프트웨어사에서 이용할 수 있는 MS-DOS 윈도우 브랜드 운영 환경과 같은)와 호환성이 있고, 좋은 프로그래밍 기술을 진전시키는 압축 데이터 명령을 실행하는 방법과 장치가 설명되어 있다. 압축 데이터 상태는 부동점 상태에서 에일리어스되기 때문에, 압축 데이터 상태는 부동점 상태인 것 처럼 현재의 운영 체제에 의해 보존되고 재저장될 수 있다. 더욱이, 압축 데이터 명령의 실행에 의해 발생된 이벤트는 현재의 운영 체제 이벤트 핸들러에 의해 제공될 수 있기 때문에, 이 이벤트 핸들러는 수정될 필요가 없고, 새로운 이벤트 핸들러가 추가될 필요가 없다. 결과적으로, 그 프로세서는 역으로 호환성이 있고, 업그레이드는 운영 체제를 수정하거나 발전시키는데 필요한 비용과 시간을 필요로 한다.

여기서 일부 설명된 이 실시예의 변화는 이러한 운영 체제와 완전히 또는 부분적으로 호환성이 있을 수 있고, 또는 좋은 프로그래밍 기술을 진전시킬 수 있다. 예를 들어, 본 발명의 대체 실시예는 흐름도에서 상이한 단계로 특정 단계를 이동시킬 수 있다. 본 발명의 다른 실시에는 하나이상의 단계를 바꾸거나 제거할 수 있다. 도7A, 도7B, 및/또는 도7C에서 특정 단계가 제거된다면, 도6A에서 특정 하드웨어는 필요하지 않을 것이다. 예를 들어, EMMS 지시가 활용되지 않는다면, EMMS 지시는 필요하지 않다. 물론, 본 발명은 다수의 시스템 아키텍처에서 유용할 수 있고,여기서 설명된 아키텍처에 제한을 두지 않는다.

더욱이, 두 개의 물리 레지스터 파일을 에일리어스하는 방법과 장치가 설명되었지만, 대체 실시예는 다수의 상이한 타입의 명령을 실행하기 위해 다수의 물리 레지스터 파일을 에일리어스할 수 있다. 추가로, 본 실시예는 부동점 명령을 실행하는 물리 스택 레지스터 파일과 압축 데이터 명령을 실행하는 물리 플랫 레지스터 파일을 참조하여 설명되었지만, 여기서의 티칭은 이 레지스터 파일상에서 실행되는 명령의 타입에 무관하게, 적어도 하나의 물리 스택 레지스터 파일과 적어도 하나의 물리 플랫 레지스터 파일을 에일리어스하는데 사용될 수 있다.

추가로, 부동점과 압축 데이터 명령을 실행하는 방법과 장치가 설명되었지만, 대체 실시예는 다수의 상이한 타입의 명령을 실행하도록 구현될 수 있다. 예를 들어, 앞서 설명된 바와 같이, 압축 데이터 명령은 프로세서가 압축 정수 연산 및/또는 압축 부동점 연산을 수행하게 하도록 구현될 수 있다. 다른 예로서, 압축 데이터 명령보다는 또는 추가로, 대체 실시예는 스칼라 부동점과 스칼라 정수 명령의 실행에 대해 물리 레지스터 파일을 에일리어스할 수 있다. 다른 예로서, 압축 데이터 명령을 부동점 레지스터상에서 에일리어스하기 보다는, 대체 실시예는 그 압축 데이터 명령을 정수 레지스터상에서 에일리어스할 수 있다. 다른 예로서, 대체 실시예는 스칼라 부동점, 스칼라 정수, 및 압축 데이터 명령(정수 및/또는 부동점)의 실행을 단일 논리 레지스터 파일상에서 에일리어스할 수 있다. 따라서, 여기서의 티칭은, 단일 논리 레지스터 파일이 상이한 데이터 타입에서 동작하는 명령의 실행에 대하여 이용가능하다는 것은 소프트웨어에 논리적으로 나타나게 하는데 사용될 수 있다.

도8은 본 발명의 일실시예에 따라 도7C의 단계(734)를 수행하는 방법을 예시하는 흐름도이다. 앞서 설명된 바와 같이, 단계(754)에서 프로세서는 부동점 모드에서 압축 데이터 모드로 이행된다. 단계(722)에서, 단계(800)로 진행한다.

단계(800)에서 보는 바와 같이, 이전의 부동점 명려으로부터의 보류 오류가 있는지를 판단한다. 그러하다면, 단계(724)로 진행한다. 그 반대면, 단계(804)로 진행한다. 따라서, 단계(800)는 도7의 단계(720) 및 도4A의 단계(422)와 유사하다.

앞서 설명된 바와 같이, 단계(724)에서, 보류 부동점 오류 예외가 발생되고, 적절한 이벤트 핸들러가 실행된다. 도4A의 단계(424)를 참조하여 앞서 설명된 바와 같이, 이 이벤트는 내부 또는 외부 이벤트로서 취급될 수 있고, 따라서 제공될 수 있다. 대체 실시예에서, 이러한 오류는 압축 데이터 명령의 실행동안

에 보류된다.

단계(804)에서 보는 바와 같이, 부동점 레지스터의 소수부 필드내에 저장된 데이터는 압축 데이터 레지스터에 복사된다. 그렇게 하면, 부동점 레지스터내에 저장된 데이터는 압축 데이터로서 연산될 수 있다. 완전한 에일리어싱이 구현되면, 모든 부동점 레지스터의 소수부 필드내에 저장된 데이터는 그 대응 압축 데이터 레지스터에 복사된다. 그 반대로, 부분 에일리어싱이 구현되면, 일실시예는, 대응 태그가 비공백 상태를 지시하는 부동점 레지스터만의 소수부 필드내에 저장된 데이터가 적절한 대응 압축 데이터 레지스터에 복사되도록 구현될 수 있다. 부동점 레지스터에 저장된 데이터가 압축 데이터로서 연산되지 않게 하는 대체 실시예는 단계(804)를 수행할 필요가 없다. 단계(804)에서, 단계(806)로 진행한다.

단계(806)에서, EMMS 지시는 최종 압축 데이터 명령이 EMMS 명령이 아니었음을 지시하도록 바뀌게 되고, 단계(808)로 진행한다. 이 단계는 압축 데이터 모드를 개시하도록 수행된다.

단계(808)에서, 더티 지시의 각각은 클린 상태를 지시하도록 바뀌게 되고, 단계(810)로 진행한다. 단계(806)와 단계(808)는 압축 데이터 모드를 개시하도록 수행된다.

단계(810)에서 보는 바와 같이, 추리 지시가 부동점에서 압축 데이터로의 이행이 추리적이다는 것을 지시하도록 바뀌게 된다. 부동점 레지스터내에 저장된 데이터는 단계(804)에서 압축 데이터 레지스터로 복사됨에도 불구하고, 부동점 유닛의 상태는 바뀌지 않았다. 따라서, 그 부동점 상태는 여전하다(예, 부동점 레지스터의 소수부 필드내에 저장된 데이터는 압축 데이터 레지스터에 저장된 것과 등가이고; 그 태그는 바뀌지 않고; 스택 지시의 상부는 바뀌지 않았다). 압축 데이터 명령이 계속하여 실행되면, 압축 데이터 레지스터에 저장된 데이터는 바뀌게 될 것이고, 부동점 상태는 더 이상 현재의 것이 아닐 것이다. 결과적으로, 압축 데이터 모드에서 부동점 모드로의 이행은 부동점 상태가 갱신되어야 할 필요가 있을 것이다(예, 압축 데이터 레지스터내에 저장된 데이터는 부동점 레지스터의 소수부 필드에 복사되어야 할 것이고; 스택 지시의 상부는 0으로 바뀌게 되어야 할 것이고; 그 태그는 공백 상태로 바뀌게 되어야 할 것이다). 그러나, 부동점 명령의 실행은 어느 압축 데이터 명령의 실행에 앞서 시도된다면(부동점 모드에서 압축 데이터 모드로 이행하게 하는 압축 데이터 명령의 실행에 앞서 이벤트가 발생된다면 이런 결과가 될 수 있다--예, 압축 데이터 명령의 실행이 시도되는 동안 메모리 결함이 일어났다면), 부동점 상태는 여전히 통용하고 있는 바와 같이 갱신될 필요가 없다. 이러한 갱신을 피함으로써, 압축 데이터 모드에서 부동점으로의 역 이행하는 오버헤드는 상당히 감소된다. 이러한 것을 이용하기 위해, 추리 지시는 이 단계에서 부동점 유닛에서 압축 데이터 유닛으로의 이행이 추리적이다는 것을 지시하도록 바뀌게 된다-- 부동점 상태는 여전히 통용된다. 압축 데이터 명령이 연속하여 실행되면, 추리 지시는 그 이행이 도7의 단계(744)를 참조하여 앞서 설명된 바와 같이 더 이상 추리적이지 않다는 것을 지시하도록 바뀌게 된다. 추리 지시의 사용은 도9를 참조하여 추가 설명되어 있다. 추리 지시가 사용된 일실시예가 설명되어 있지만, 대체 실시예는 이러한 추리 지시를 구현하는 것을 피할 수 있다.

단계(812)에서, 그 모드 지시는 프로세서가 현재 압축 데이터 모드에 있음을 지시하도록 바뀌게 된다. 단계(812)에서, 단계(736)로 진행한다.

도9는 본 발명의 일실시예에 따라 도7의 단계(728)를 수행하는 방법을 예시하는 흐름도이다. 앞서 설명된 바와 같이, 단계(728)에서 프로세서는 압축 데이터 모드에서 부동점 모드로 이행된다. 단계(726)에서, 단계(900)로 진행한다.

단계(900)에서, 추리 지시가 압축 데이터 모드로의 이행이 여전히 추리적이다는 것을 지시하는지를 판단한다. 앞서 설명된 바와 같이, 추리 지시는 압축 데이터 모드에서 부동점 모드로 이행하는 오버헤드를 감소시키는데 사용될 수 있다. 단계(900)에서, 부동점에서 압축 데이터로의 이행이 추리적이다는 것이 결정되면, 단계(902)내지 단계(912)는 피하게 되고, 단계(914)로 직접 진행하고, 그 이행한 오버헤드는 감소된다. 그렇지 않으면, 단계(902)로 진행한다.

단계(902)에서 보는 바와 같이, EMMS 지시가 최종 압축 데이터 명령이 EMMS 명령이었음을 지시하는지를 판단한다. 그러하면, 단계(904)로 진행한다. 그렇지 않으면, 단계(906)로 진행한다. 앞서 설명된 바와 같이, 그 압축 데이터 명령이 별개의 유닛(즉, 압축 데이터 유닛)상에서 실행된다는 사실은, 특정 연산(예, 태그를 바꾸는 것)을 수행하는 것보다 부동점 모드로 역 이행할 때 무엇을 행해야 하는지를 식별하는 지시(예, EMMS 지시)를 저장하는 것을 더 효과있게 한다. 따라서, 그 EMMS 명령에 반응하여 그 태그가 바뀌기 보다는 EMMS 지시가 바뀌였다. 그 다음, 부동점 모드로의 역 이행을 수행할 때, 여기서 보는 바와 같이 따라서 그 태그는 바뀌게 된다.

단계(904)에서, 모든 태그는 공백 상태로 바뀌게 되고, 단계(908)로 진행한다. 이러한 방식으로, 그 태그는 도4B의 단계(440)와 유사한 방식으로 바뀌게 된다.

단계(906)에서, 모든 태그는 비공백 상태로 바뀌게 되고, 단계(908)로 진행한다. 이러한 방식으로, 그 태그는 도4B의 단계(440)와 유사한 방식으로 바뀌게 된다.

단계(908)에서 보는 바와 같이, 압축 데이터 레지스터의 내용은 부동점 레지스터의 소수부 필드에 복사되고, 단계(910)로 진행한다. 이러한 방식으로, 압축 데이터 레지스터에 저장된 데이터는 부동점 데이터로서 연산될 수 있다. 더욱이, 현재의 운영 체제는 다중 작업을 수행할 때 이미 부동점 상태를 저장하고 있기 때문에, 압축 데이터 상태는, 부동점 상태인 것처럼 여러 콘텍트 구조로부터 저장되고 재저장된다. 이러한 방식으로, 물리 압축 데이터 레지스터는 물리 부동점 레지스터상에서 에일리어스되고, 그 프로세서는 단일 논리 레지스터 파일을 가지게 논리적으로 나타난다. 결과적으로, 그 실시예는 그 운영체제를 포함한 소프트웨어에서 볼 수 없다. 완전한 에일리어싱이 구현되면, 모든 압축 데이터 레지스터에 저장된 데이터는 그 대응 부동점 레지스터의 소수부 필드에 복사된다. 그 반대로, 부분 에일리어싱이 구현되면, 실시예는 터치된 그 압축 데이터 레지스터에만 저장된 데이터가 적절한 대응 부동점 레지스터의 소수부 필드에 복사되도록 구현될 수 있다.

단계(910)에서 보는 바와 같이, 스택의 상부는 초기값으로 바뀌게 된다. 일실시예에서, 이 값은 0이다. 대체 실시예에서, 어느 압축 데이터 명령의 실행은 스택 지시의 상부를 초기값에 설정한다. 단계(910)에

서, 단계(912)로 진행한다.

단계(912)에서 보는 바와 같이, 대응 더티 지시가 더티 상태에 있는 그 부동점 레지스터의 부호 필드와 지수 필드에 1이 저장된다. 이러한 방식으로, 도4B의 단계(438)가 수행된다. 단계(912)에서 단계(914)로 진행한다.

단계(914)에서, 그 모드 지시는 프로세서가 부동점 모드에서 동작하고 있음을 지시하도록 바뀌게 되고, 단계(736)로 진행한다. 이러한 방식으로, 압축 데이터 모드에서 부동점 모드로의 이행이 수행된다.

도10은 본 발명의 다른 실시예에 따라 단일 물리 레지스터 파일을 이용하여 그 압축 데이터 상태를 부동점 상태상에서 에일리어스하는 장치를 통해 데이터 흐름을 예시하는 블록도이다. 도10에 도시된 장치는 도5의 명령 세트 유닛(560)으로서 사용될 수 있다. 일실시예에서, 도10의 장치는 최소한 명령 세트(580)를 실행할 수 있다. 도10은 디코드 유닛(1002), 재명명(rename) 유닛, 회수(retirement) 유닛(1006), 발생 유닛(1008), 실행 유닛(1010), 상태 레지스터 세트(1012), 및 마이크로코드 ROM(1014)를 도시하고 있다.

디코드 유닛(1002)은 프로세서에 의해 수신된 명령을 제어 신호 및/또는 마이크로코드 엔트리 포인트로 디코드하는데 사용된다. 이 마이크로코드 엔트리 포인트는 디코드 유닛(1002)에 의해 프로세서내의 여러 유닛에 전송되는 마이크로 연산(일명 'uops')의 시퀀스를 식별한다. 특정 마이크로 연산은 디코드 유닛(1002)에 저장될 수 있지만, 일실시예에서는, 대다수의 마이크로 연산은 마이크로 코드 ROM(1014)에 저장되어 있다. 이 실시예에서, 디코드 유닛(1002)은 마이크로코드 엔트리 포인트를 마이크로코드 ROM(1014)에 전송하고, 필요한 마이크로 연산을 디코드 유닛(1002)에 역으로 전송함으로써 응답한다.

디코드 유닛(1002)에 의해 수신된 대부분의 명령은 그 명령의 연산이 수행되게 하는 하나이상의 피연산자(데이터, 레지스터 위치, 메모리내의 위치중 어느 하나)를 포함하고 있다. 레지스터를 식별하는 그 피연산자는 재명명 유닛(1004)에 전송된다.

재명명 유닛(1004)과 회수 유닛(1006)은 레지스터 재명명을 구현하는데 사용된다. 레지스터 재명명의 기술은 공지되어 있고, 레지스터와 같은 제한된 수의 기억 위치 사용을 시도한 상이한 명령으로부터 초래된 기억 콘플릭트(conflict)를 피하기 위해 수행된다. 그 콘플릭트 명령이 독립적임에도 불구하고 서로 방해할 때 기억 콘플릭트가 일어났다고 말한다. 기억 콘플릭트는 레지스터와 값사이의 대응을 재정립하는데 사용되는 추가 레지스터를 제공함으로써 제거될 수 있다. 레지스터 재명명을 구현하기 위해서, 전형적으로 프로세서는 생성된 모든 새로운 값에 대하여 버퍼 레지스터중 상이한 하나에 할당한다: 즉, 레지스터를 기록하는 모든 명령에 대하여. 그 값을 판독하기 위해서, 오리지날 레지스터를 식별하는 명령은 할당된 버퍼 레지스터내의 값대신 얻는다. 따라서, 그 하드웨어는 그 명령을 식별하는 오리지날 레지스터를 재명명하고, 버퍼 레지스터와 올바른 값을 식별한다. 상이한 일부 명령내의 동일 레지스터 식별자는 레지스터 할당에 관한 레지스터 레퍼런스의 위치에 따라서, 상이한 하드웨어 레지스터에 액세스할 수 있다. 레지스터 재명명의 추가 설명에 대하여, 뉴저지소재의 PTR 프렌티스-홀 사의 1991년판 존슨 마이크의 초스칼라 마이크로 프로세서 설계 ; 콜웰외 다수에 의한 미국 특허 08/204,521 레지스터 에일리어스 테이블내의 플래그 마스크와 플래그 재명명 ; 클리프트외 다수에 의한 미국 특허 08/129,678 프로세서 디바이스내의 정수 및 부동점 레지스터 에일리어스 테이블 ; 및 콜웰외 다수에 의한 미국 특허 08/174,841 레지스터 에일리어스 테이블내의 부분 폭 스톨(stall)을 참조하라. 하나의 명령이 완전히 실행되었을 때(보류를 유지하지 않는 이벤트를 유발하지 않고), 명령 할당 버퍼 레지스터는 '회수된다'-- 그 값은 버퍼 레지스터에서, 그 명령으로 식별된 오리지날 레지스터로 이동된다. 대체 실시예는 인터록, 부분 재명명등과 같은 기억 콘플릭트을 제거하는 다수의 기술을 구현할 수 있다.

회수 유닛(1006)은 FP/PD 레지스터 세트(1022)로서 버퍼 레지스터 세트(1020)와 정수 레지스터 세트(1024)를 포함하고 있다. 그 버퍼 레지스터 세트(1020)는 레지스터 재명명에 사용되는 추가 레지스터를 제공한다. 일실시예에서는, 버퍼 레지스터 세트(1020)가 40개의 레지스터를 포함하고 있지만, 대체 실시예는 다수의 레지스터를 구현할 수 있다. 이러한 실시예에서, 버퍼 레지스터 세트(1020)는 리오더(reorder) 버퍼로서 동작된다.

일실시예에서, FP/PD 레지스터(1022)와 정수 레지스터는 소프트웨어에서 볼 수 있다: 즉, 그 명령으로 식별되는 레지스터가 있고, 따라서 부동점 데이터, 압축 데이터, 및 정수 데이터를 실행하는 레지스터만이 있음이 그 소프트웨어에 나타난다. 그 반대로, 그 버퍼 레지스터(1020)는 그 소프트웨어에서 볼 수 없다. 따라서, FP/PD 레지스터(1022)는 단일 논리 레지스터 파일로서 소프트웨어에 나타나는 단일 물리 레지스터 파일이다. 일실시예에서, FP/PD 레지스터 세트(1022)와 정수 레지스터 세트(1024)는 각각 8개의 레지스터를 포함하고 있고, 현재의 인텔 아키텍처 소프트웨어와 호환성이 있다. 그러나, 대체 실시예는 다수의 레지스터를 구현할 수 있다.

재명명 유닛(1004)은 FP/PD 매핑 유닛(1030), FP/PD 매핑 테이블(1032), 태그 세트(1034), 정수 매핑 유닛(1040), 및 정수 매핑 테이블(1042)을 포함하고 있다. 재명명 유닛(1004)가 피연산자를 받을 때, 피연산자가 부동점 피연산자, 압축 데이터 피연산자, 또는 정수 피연산자인지를 판단한다.

정수 매핑 유닛(1040)은 정수 피연산자를 받는다. 그 정수 매핑 유닛(1040)은 정수 매핑 테이블(1042)을 제어한다. 일실시예에서, 정수 매핑 테이블(1042)은 정수 레지스터(1024)에 레지스터가 있는 것과 동수의 엔트리를 포함하고 있다. 정수 매핑 테이블(1042)내의 각각의 엔트리는 상이한 하나의 정수 레지스터(1024)에 대응한다; 도10에서, 엔트리(1050)는 정수 레지스터(1052)에 대응한다. 프로세서가 정수 레지스터(예, 정수 레지스터(1052))에 기록하게 할 수 있는 명령이 수신되면, 정수 매핑 유닛(1040)은, 버퍼 레지스터 세트(1020)(예, 버퍼 레지스터(1054))내에서 이용가능한 레지스터를 식별하여 정수 매핑 테이블(1042)내의 정수 레지스터의 대응 엔트리(예, 엔트리(1050))에 포인터를 저장함으로써 하나의 버퍼 레지스터(1020)를 할당한다. 그 데이터는 선택된 버퍼 레지스터(예, 버퍼 레지스터(1054))에 기록된다. 피연산자를 만드는 명령 실행이 중단없이(발생된 어떠한 이벤트없이) 완료되었을 때, 회수 유닛(1006)은 선택된 버퍼 레지스터(예, 버퍼 레지스터(1054))에서 적절한 정수 레지스터(예, 정수 레지스터(1052))로 복사함으로써 그 데이터를 '기억'시키고, 그 데이터가 그 엔트리의 대응 정수 레지스터에 저장되어 있음을 지

시하기 위해 그 엔트리(예, 엔트리(1050))의 내용을 정수 매핑 유닛(1040)가 갱신하게 한다.

프로세서가 정수 레지스터를 판독하게 할 수 있는 명령을 받을 때, 그 프로세서는 FP/PD 매핑 유닛(1030)을 이용하여 정수 매핑 테이블(1042)내의 정수 레지스터의 대응 엔트리(예, 엔트리(1050))의 내용을 액세스한다. 그 엔트리가 버퍼 레지스터(예, 버퍼 레지스터(1054))에 포인터를 포함하고 있다면, 프로세서는 그 버퍼 레지스터의 내용을 판독한다. 그러나, 엔트리의 내용이 그 데이터는 그 엔트리의 대응 정수 레지스터(예, 정수 레지스터(1052))에 저장되어 있음을 표시한다면, 프로세서는 그 엔트리의 대응 정수 레지스터의 내용을 판독한다. 따라서, 그 정수 레지스터(1024)는 본 발명의 실시예에서 고정된 레지스터 파일로서 구현된다.

FP/PD 매핑 유닛(1030)은 FP/PD 매핑 테이블(1032)과 태그(1034)를 제어한다. 앞서 설명된 바와 같이, 이 태그의 각각은 복수의 비트를 사용하여 구현될 수 있다. 정수 매핑 유닛(1040)과 유사하게, FP/PD 매핑 테이블(1032)은 FP/PD 레지스터(1022)에 레지스터가 있는 것과 동수의 엔트리를 포함하고 있다. FP/PD 매핑 테이블(1032)내의 각각의 엔트리는 상이한 하나의 FP/PD 레지스터(1022)에 대응한다. 부동점과 압축 데이터 피연산자는 FP/PD 매핑 유닛(1030)에 의해 받아들여지고, 버퍼 레지스터(1020)에서 매핑되고, FP/PD 레지스터(1022)에서 회수된다. 따라서, 부동점 상태와 압축 데이터 상태는 단일 유저 가시 레지스터 파일상에서 에일리어스된다. 현재의 운영 체제는, 다중 작업을 할 때, 프로세서가 부동점 상태를 저장하게 하도록 구현되기 때문에, 이러한 동일 운영 체제는 부동점 레지스터상에서 에일리어스되는 하나의 압축 데이터 상태를 프로세서가 저장하게 할 수 있다.

일실시예에서, 그 압축 데이터 피연산자는 정수 피연산자에서와 유사한 방식으로 조정된다-- 압축 데이터 레지스터는 고정된 레지스터 파일로서 구현된다. 따라서, 프로세서가 FP/PD 레지스터에 기록하게 할 수 있는 압축 데이터 명령이 수신될 때, FP/PD 매핑 유닛(1030)은, 버퍼 레지스터 세트(1020)에서 이용가능한 레지스터를 식별하여 FP/PD 매핑 테이블(1032)내의 그 FP/PD 레지스터의 대응 엔트리를 저장함으로써 하나의 버퍼 레지스터(1020)를 할당한다. 그 데이터는 그 선택된 버퍼 레지스터에 기록된다. 그 피연산자를 발생시킨 명령의 실행이 어떠한 인터럽트없이(발생된 어떠한 이벤트없이) 완료되었을 때, 그 회수 유닛(1006)은 선택된 버퍼 레지스터에서 적절한 FP/PD 레지스터(FP/PD 매핑 테이블(1032)내의 엔트리에 대응하는 FP/PD 레지스터)로 복사함으로써 그 데이터를 '기억'시키고, FP/PD 매핑 유닛(1030)이, 그 데이터가 그 엔트리의 대응 FP/PD 레지스터에 저장되어 있음을 표시하기 위해 FP/PD 매핑 테이블(1032)내의 엔트리를 갱신하게 한다.

압축 데이터 명령을 실행할 때, 그 레지스터는 고정된 레지스터 파일로서 구현되지만, 본 발명의 일실시예는 현재의 인텔 아키텍처 소프트웨어(운영 체제를 포함하여)와 호환성이 있도록 부동점 명령을 실행할 때 스택 레퍼런스 레지스터 파일로서 그 레지스터를 구현한다. 결과적으로, FP/PD 매핑 유닛(1030)은 압축 데이터 피연산자에 대한 양 고정 레지스터 파일로서, 그리고 부동점 피연사자에 대한 스택으로서 FP/PD 매핑 테이블을 동작시킬수 있어야 한다. 결론적으로, FP/PD 매핑 유닛(1030)은 스택 필드(1072)의 상부를 가진 부동점 상태 레지스터(1070)를 포함하고 있다. 스택 필드(1072)의 상부는 부동점 스택의 상부에서 현재 그 레지스터를 나타내는 FP/PD 매핑 테이블(1032)내의 엔트리를 식별하는 스택 표시의 상부를 저장하는데 사용된다. 물론, 대체 실시예는 부동점 명령을 실행할 때 플랫 레지스터 파일로서 그 레지스터를 동작시킬 수 있다.

프로세서가 FP/PD 레지스터를 기록하게 할 수 있는 부동점 명령이 수신될 때, FP/PD 매핑 유닛(1030)은 스택 표시의 상부를 바꾸고, 버퍼 레지스터 세트(1020)내에서 이용가능한 레지스터를 식별하는 포인터를 FP/PD 매핑 테이블(1032)내의 스택 레지스터의 대응 엔트리의 상부에 저장함으로써 하나의 버퍼 레지스터(1020)에 할당한다. 그 데이터는 그 선택된 버퍼 레지스터에 기록된다. 피연산자를 발생한 명령 실행이 어떠한 인터럽트없이(발생된 이벤트없이)완료되었을 때, 그 회수 유닛(1006)은 그 선택된 버퍼 레지스터에서 적절한 FP/PD 레지스터(FP/PD 매핑 테이블(1032)내의 엔트리에 대을하는 FP/PD 레지스터)로 복사함으로써 그 데이터를 '기록' 하고, FP/PD 매핑 유닛(1030)이, 그 데이터가 그 엔트리의 대응 FP/PD 레지스터에 저장되어 있음을 표시하는 FP/PD 매핑 테이블(1032)내의 엔트리를 갱신하게 한다.

따라서, FP/PD 매핑 유닛(1030)은 스택 레퍼런스 레지스터 파일상에서 부동점 피연산자를 매핑하기 때문에, FP/PD 매핑 테이블(1032)내의 엔트리는 그 스택의 상부에 관하여 액세스되어야 한다. 그 반대로, FP/PD 매핑 유닛(1030)은 고정 레지스터 파일상에서 압축 데이터 피연산자를 매핑하기 때문에, FP/PD 매핑 테이블(1032)내의 엔트리는 레지스터(R0)에 관하여 액세스되어야 한다. 프로세서가 레지스터(R0)에 관한 FP/PD 매핑 테이블내의 엔트리에 액세스하게 하기 위해서, 스택 표시의 상부는 레지스터(R0)를 표시하도록 바뀌여야 한다. 그러므로, 그 프로세서가 압축 데이터 명령을 실행하고 있는 동안에 스택 표시의 상부는 레지스터(R0)를 표시하도록 바뀌여야 한다. 이것은 부동점 모드에서 압축 데이터 모드로의 이행동안에 레지스터(R0)를 표시하는 것으로 스택 표시의 상부를 바꿈으로써, 그리고 압축 데이터 명령의 실행동안에 스택 표시의 상부를 바꾸지 않음으로써 실행될 수 있다. 이러한 방식으로, 부동점 스택을 매핑하는데 사용된 동일 회로 소자는 고정된 압축 데이터 레지스터 파일을 매핑하는데 사용될 수 있다. 결과적으로, 도6A를 참조하여 설명된 실시예보다 회로의 복잡성이 감소되고, 다이의 면적이 절약된다. 동일 회로 소자가 압축 데이터와 부동점 피연산자 모두를 매핑하는데 사용된 일실시예가 설명되어 있지만, 대체 실시예는 별개의 회로 소자를 이용할 수 있다.

실행되는 명령의 유형과는 무관하게, 일실시예에서, 버퍼 레지스터의 할당과 비할당은 동일 방식으로 조정된다. 회수 유닛(1006)은 할당 필드(1062)와 회수 필드(1064)를 가진 상태 레지스터(1060)를 포함하고 있다. 그 할당 필드(1062)는 사용될 다음 버퍼 레지스터를 식별하는 할당 포인터를 저장하고 있다. FP/PD 매핑 유닛(1030) 또는 정수 매핑 유닛(1040)중 어느 하나가 하나의 레지스터를 필요로 할 때, 현재의 할당 포인터는 적절한 매핑 테이블(즉, FP/PD 매핑 유닛(1030) 또는 정수 매핑 테이블(1042))에 저장되고, 그 할당 포인터는 증가된다. 추가로, 재명명 유닛(1004)은 그 명령이 압축 데이터 명령인지를, 그리고 그 프로세서가 압축 데이터 모드에 있는지를 지시하는 신호를 회수 유닛(1006)에 전송한다.

할당 버퍼 레지스터에서, 회수 유닛(1006)은 준비 필드(1082)에 준비 지시를 저장한다. 그 준비 지시는 그 버퍼 레지스터가 회수 준비 상태가 아님을 지시하는 것으로 초기에 바뀌게 된다. 그러나, 그 데이터가 버퍼 레지스터의 데이터 필드(1080)에 기록될 때, 그 버퍼 레지스터의 준비 지시는 그 버퍼 레지스터가

회수 준비 상태에 있음을 지시하는 것으로 바뀌게 된다.

상태 레지스터(1060)의 회수 필드(1064)는 회수될 다음 버퍼 레지스터를 식별하는 회수 포인터를 저장하고 있다. 그 버퍼 레지스터의 준비 지시는 준비 상태로 바뀌게 될 때, 그 회수 유닛(1006)은 그 버퍼 레지스터내의 데이터가 기록될 수 있는지를 판단하여야 한다. 추후 추가 설명되는 바와 같이, 회수 유닛(1006)의 일실시예는 어떠한 예외가 발생되어야 한다면(예, 예외를 이용할 수 없는 디바이스, 보류한 부동점 오류 예외, 무효 연산 코드 예외,등) 또는, 압축 데이터와 부동점 모드사이의 이행이 필요하다면, 그 데이터를 기록하지 않는다. 그 데이터가 기록될 수 있다면, 그 데이터는 적절한 FP/PD 또는 정수 레지스터에 복사되고, 그 회수 포인터는 다음 버퍼 레지스터에서 증가된다. 그 회수 및 할당 포인터가 제어 레지스터에 저장되어 있는 것처럼 설명되었지만, 대체 실시예는, 플립 플롭와 같이 순차 요소의 형태로, 여기서 설명된 어떤 다른 정보(예, EMMS 지시, 모드 지시등)뿐만 아니라, 그 포인터를 저장할 수 있다.

회수 유닛(1006)이 3개의 별개 레지스터 세트를 포함하고 있고, 데이터는 버퍼 레지스터에서 FP/PD 레지스터 또는 정수 레지스터로 기록되는 일실시예가 설명되어 있지만, 대체 실시예는 다수의 상이한 레지스터 세트를 포함하도록 구현될 수 있다. 예를 들어, 하나의 대체 실시예는 단일의 레지스터 세트를 포함하고 있을 수 있다. 이러한 실시예에서, 이러한 레지스터 세트내의 각각의 레지스터는 여기에 저장된 데이터가 기록되었는지를 식별하는 지시를 포함할 수 있다.

일실시예에서, 그 프로세서는 부동점 모드 또는 압축 데이터 모드중 어느 하나의 모드에 있다. 그 프로세서가 압축 데이터 모드에 있지 않으면, 그 프로세서는 어떠한 압축 데이터 명령을 적당히 실행할 수 없고, 그 역도 동일하다. 결과적으로, 버퍼 레지스터에 저장된 데이터를 기록하기에 앞서, 그 회수 유닛(1006)은 그 데이터가 압축 데이터인지를 그리고, 그 프로세서가 압축 데이터 모드에 있는지를 판단한다. 그 데이터가 압축 데이터이고 그 프로세서가 압축 데이터 모드에 있지 않으면, 마이크로코드 ROM(1014)에 포함되어 있는 이행 유닛(1036)은 그 압축 데이터 모드에서 이행하도록 되어 있다. 일실시예에서, 그 프로세서가 스택 지시의 상부가 초기값(예, 레지스터(R0)를 지시하는 것)으로 바뀌게 되는지를 그리고 모든 태그(1034)가 비공백 상태에 있는지를 판단함으로써 압축 데이터 모드에 있는지를 판단하게 된다.

그 프로세서가 스택 지시의 상부를 폴링하게 하고, 태그(1034)가 그 프로세서가 압축 데이터 모드에 있는지를 판단하게 하는 다수의 기술이 있다. 예를 들어, 앞서 설명된 바와 같이, 그 디코드 유닛(1002)이 마이크로코드 ROM(1014)에서 마이크로연산에 액세스한다. 이 마이크로 연산은 그 적당한 매핑이 FP/PD 매핑 유닛(1030)에 의해 수행되는지를 식별하는 인코드 필드를 포함하고 있다(예, 스택 지시의 상부를 증가, 스택 지시의 상부를 감소, 등). 일실시예에서, 적어도 하나의 추가 인코드 비트 패턴('압축 데이터 비트 패턴'이라 함)은 압축 데이터 비트 패턴에 대한 매핑을 식별하기 위해 포함되어 있다. 따라서, 그 디코드 유닛(1002)은 압축 데이터 명령을 수신하여 마이크로코드 ROM(1014)에 액세스할 때, 그 디코드 유닛(1002)에 전송된 적어도 하나는 압축 데이터 비트 패턴을 포함하고 있다.

그 압축 데이터 비트 패턴을 포함하고 있는 마이크로 연산을 수신함과 동시에, FP/PD 매핑 유닛(1030)은: 1) 스택 지시의 상부와 태그(1034)의 상태를 판단하고; 2) 압축 데이터 모드로의 이행이 필요한지를 지시하는 신호를 회수 유닛(1006)에 전송한다(일실시예에서, 그 프로세서의 모드와 명령 유형이 전송된다). 반응하여, 회수 유닛(1006)은 그 명령에 의해 할당된 임의의 버퍼 레지스터로 하나의 이행 필드(1084)에 이행 지시를 저장하고 있다(일실시예에서, 이행 지시는 그 프로세서의 모드를 지시하는 제 1 비트와 명령 유형을 지시하는 제 2 비트를 포함하고 있다). 따라서, 그 명령이 압축 데이터 명령이고, 그 프로세서가 압축 데이터 모드에 있지 않으면, 그 적당한 버퍼 레지스터의 모드 지시는 이행이 필요함을 지시하는 것으로 바뀌게 된다. 그 반대면, 그 모드 지시는 이행이 필요하지 않음을 지시하는 것으로 바뀌게 된다. 회수 포인터에 의해 식별된 버퍼 레지스터의 준비 지시는 준비 상태로 바뀌게 되고, 회수 유닛(1006)은 이행 지시를 체크한다. 그 이행 지시가 이행이 필요없음을 지시한다면, 그리고 그 데이터가 반대로 회수될 수 있다면(예, 제공되어야 하는 이벤트가 없다), 그 데이터는 회수된다. 그 반태로, 그 이행 지시가 이행이 필요함을 지시한다면, 그 회수 유닛(1006)은 이행 유닛(1036)에 대한 마이크로코드 엔트리 포인트를 마이크로코드 ROM(1014)에 전송한다. 반응하여, 그 마이크로코드 ROM(1014)은 프로세서에서 압축 데이터 모드로의 이행을 위해 필요한 마이크로 연산을 전송한다.

이러한 방식으로, 압축 데이터 모드로의 이행 혼합은 복잡성이 약간 증가할 수 있다. 물론, 대체 실시예는 1) 재명명 유닛(1004)이 스택 지시의 상부와 태그를 폴링하게 하는 압축 데이터 명령을 수신하여 디코드 유닛(1002)이 특정 신호를 전송하게 하는 방법; 2) 스택의 상부와 태그가 폴링될 수 있는지를 지시하기 위해 비트를 모든 마이크로 연산에 추가하는 방법; 3) 버퍼 레지스터가 할당될 때마다 FP/PD 매핑 유닛(1030)이 스택 지시의 상부와 태그를 폴링하게 하는 방법; 4) 압축 데이터 항목이 기록될 준비가 되어 있을 때 회수 유닛(1006)이 FP/PD 매핑 유닛(1030)에 지시하게 하는 방법, 및 그 프로세서가 압축 데이터 모드에 있지 않으면 FP/PD 매핑 유닛(1030)이 이행 유닛(1036)을 호출하게 하는 방법등 여러 방법으로 이러한 기능을 구현할 수 있다. 일실시예에서, 스택 지시의 상부와 태그(1034)를 근거로 하여, 그 프로세서가 압축 데이터 모드에 있는지를 판단하지만, 대체 실시예는 앞서 설명된 모드 지시를 포함하여, 다수의 기술을 사용할 수 있다.

앞서 설명된 바와 같이, 이행 유닛(1036)은 부동점 모드에서 압축 데이터 모드로 그 프로세서를 이행하는데 사용된다. 그 이행 유닛(1036)은 그 프로세서가 스택 지시를 초기값으로 바꾸게 하고, 그리고 모든 태그(1034)를 비공백 상태로 바꾸게 한다. 이러한 방식으로, 재명명 유닛(1004)은 압축 데이터 명령의 실행을 위해 초기화된다. 그 이행의 완료와 동시에, 부동점 모드에서 압축 데이터 모드로의 이행을 유발한 그 명령은 마이크로 재시작된다. 결과적으로, 비마이크로코드 이벤트 핸들러(운영 체제 이벤트 핸들러를 포함하여)는 필요하지 않고, 그 실시예는 보이지 않는 시스템을 작동시키고 있다. 그 이행 유닛(1036)이 마이크로코드 ROM(1014)에 위치되게 도시되어 있지만, 대체 실시예는 프로세서의 어디든지 이행 유닛(1036)을 위치시킬 수 있다. 다른 대체 실시예에서, 이행 유닛(1036)은 부동점 모드에서 압축 데이터 모드로의 이행을 수행하도록 구현될 수 있다. 이러한 이행동안에, 이행 유닛(1036)은 기억 영역에서 현재의 스택 지시의 상부를 보존할 것이고, 스택 지시의 상부를 초기값으로 바꿀 것이다. 그 이행 유닛(1036)이 다시 역으로 부동점 모드로 이행될 때, 이행 유닛(1036)은 이전의 스택 지시의 상부를 재저장할 것이다. 더욱이, 대체 실시예에서, 이행 유닛(1036)은 그 프로세서의 외부에 저장된 비마이크로코드 이벤트 핸들러

로서 또는 하드웨어에 구현될 수 있다.

일실시예를 참조하여 앞서 설명된 바와 같이, 압축 데이터 명령의 각각의 그룹화는 EMMS 명령으로 종결시키는 것이다. EMMS 명령을 실행하는 것에 응답하여, 그 실행 유닛(1010)은 재명명 유닛(1004)이 태그(1034)를 공백 상태로 바꾸게 한다. 따라서, EMMS 명령을 실행한 후, 그 프로세서는 부동점 모드에 있다: 즉, 모든 태그(1034)는 공백 상태에 있고, 스택 지시의 상부는 초기 상태(앞서 설명된 바와 같이, 스택 지시의 상부는 압축 데이터 모드로 이행할 때 초기값으로 바뀌게 되고 압축 데이터 명령동안에 바뀌지 않았다)에 있다. 결과적으로, 이행 유닛은 납축 데이터 모드에서 부동점 모드로의 이행을 수행하는데 필요하지 않다. 이것은 부동점 모드와 압축 데이터 모드사이에서 전후로 프로세서를 이해하도록 호출되어야 하는 도6A를 참조하여 설명된 이행 유닛과 같지 않다. 추가로, 단일 에일리어스된 레지스터 파일은 부동점 상태와 압축 데이터 상태에 사용되기 때문에, 이러한 이행은 두 개의 별개 레지스터 파일사이에 데이터를 복사하는데 필요하지 않는다. 결과적으로, 회로의 복잡성이 감소되고, 프로세서상의 다이 면적은 줄여 든다.

다른 대체 실시예에서, 스택 지시의 상부와 태그의 변경은 압축 데이터 명령의 실행과 동시에 완전히 또는 부분적으로 수행될 수 있다. 예를 들어, 그 이행 유닛에 필요한 것은 1) EMMS 명령이 아닌 각각의 압축 데이터 명령의 실행이 스택 지시의 상부를 초기값으로 바꾸게 함으로써; 그리고 2) EMMS 명령의 실행이 태그를 공백 상태로 바꾸게 함으로써 피할 수가 있다. 다른 대체 실시예에서, EMMS 명령은 구현되는 것이 아니라, 도14를 참조하여 후술되는 바와 같이 부동점 명령을 이용하여 대리 실행된다.

발생 유닛(1008)은 피연산자와 명령을 저장하는 버퍼를 나타낸다. 발생 유닛(1008)은 일련의 예약 스테이션, 중앙 명령 윈도우, 또는 그 둘의 하이브리드로서 구현될 수 있다. 예약 스테이션을 이용할 때, 그 함수 유닛(예, ALU)의 각각은 대응 피연산자를 식별하는 정보와 명령을 저장하는 자체의 버퍼를 가지고 있다. 그 반대로, 중앙 명령 윈도우를 사용할 때, 모든 함수 유닛에서 일반적인 중앙 버퍼는 대응 피연산자를 식별하는 정보와 명령을 저장하는데 사용된다. 명령의 대응 피연산자는 어떤 정보가 이용되는가에 따라 일부 상이한 형태가 될 수 있다. 실제 데이터가 이용할 수가 없다면, 명령의 대응 피연산자는, 그 데이터가 기록되었는지 그리고 데이터의 타입에 따라 FP/PD 레지스터 세트(1022), 정수 레지스터 세트(1024), 또는 버퍼 레지스터 세트(1020)중 하나의 레지스터를 식별한다. 실제 데이터가 이용가능하게 될 때, 그 데이터는 버퍼에 저장된다. 일실시예에서, 발생 유닛(1008)은 재명명 유닛(1004)으로부터 정보를 또한 수신한다. 그러나, 이 정보는 본 발명을 이해하는데 필요하지 않다. 발생 유닛(1008)은 필요한 정보가 입수될 때 그 명령을 실행 유닛(1010)에 발생시킨다.

실행 유닛(1010)은 그 명령을 실행한다. 실행 유닛(1010)은 앞서 설명된 바와 같이 기억을 위해 회수 유닛(1006)에 저장되어야 하는 하나의 피연산자 정보를 전송한다. 일실시예에서, 피연산자 정보의 부족으로 발생 유닛(1008)에서 딜레이될 수 있기 때문에, 실행 유닛(1010)은 또한 임의의 피연산자 정보를 발생 유닛(1008)에 전송한다. 이러한 방식으로, 그 피연산자 정보를 회수 유닛(1006)에 그리고 그 다음 발생 유닛(1008)에 전송함으로써 유발될 수 있는 임의의 추가 딜레이를 피하게 된다. 그 실행 유닛(1010)은 상태 레지스터(1012)에 연결되어 있다. 그 상태 레지스터(1012)는 실행 유닛(1010)이 사용하기 위한 제어 정보를 저장한다. 이러한 제어 정보는 여기서 앞서 설명된 EM 지시와 TS 지시를 포함할 수 있다. 실행 유닛(1010)은 회수 유닛(1006)으로부터 액세스된 여러 유형의 데이터를 정렬하는 데이터 정렬 유닛(1090)('로드/기억 변환 유닛'이라 함)을 포함한다. 그 데이터 정렬 유닛의 동작은 도12와 도13를 참조하여 추가 설명될 것이다.

태그(1034)를 바꾸는 것은 상이한 여러 메카니즘을 이용하여 구현될 것이다. 예를 들어, 도10은 그 태그를 바꾸는 태그 수정자 유닛(1092)을 또한 포함한 FP/PD 매핑 유닛(1030)을 도시하고 있다. 그 태그 수정자 유닛(1092)은 도6B를 참조하여 설명된 방법을 포함하여 여러 방법으로 구현될 수 있다.

예를 들어, 일실시예에서, 부동점 명령은 모든 태그가 동시에 수정될 필요가 없도록 구현될 수 있기 때문에, 태그 수정자 유닛(1092)은 모든 태그를 동시에 수정할 수 있도록 구현되어 있다(하나의 이러한 실시예는 도6B를 참조하여 앞서 설명된 바와 같다). 회로의 복잡성을 피하기 위해, 압축 데이터 상태로의 이행에 응답하여, 또는 EMMS 명령의 실행에 응답하여 그 태그를 전체적으로 바꾸는 것은 현재의 메카니즘을 이용하여 구현될 수 있다. 이러한 관점에서, EMMS 유닛(1094)에 의해 표시된 마이크로코드 명령 세트는 EMMS 명령을 구현하는 마이크로코드 ROM(1014)내에 저장될 수 있다. EMMS 유닛(1094)과 이행 유닛(1036)내의 마이크로코드 명령은 8개의 태그 각각을 바꾸는 일부 현재의 마이크로코드 연산을 디코드 유닛이 발생시킬 수 있다. 따라서, EMMS 명령을 수신하는 것에 응답하여, 디코드 유닛(1002)은 EMMS 유닛(1094)에 액세스할 수 있고, 일부 현존 마이크로 연산을 발생시킬수 있다. 이러한 각각의 마이크로 연산에 대응하여, 태그 수정자 유닛(1092)은 그 대응 태그를 공백 상태로 수정할 수 있다. 그 반대로, 이행 유닛(1036)에 액세스하는 것에 대응하여, 디코드 유닛(1002)은, 태그 수정자 유닛(1092)이 각각의 태그를 비공백 상태로 바꾸게 할 수 있는 일부 현재의 마이크로 연산을 발생시킬 수 있다. 이러한 실시예에서, 태그를 전체적으로 바꾸는 것은 약 4-8 클록 사이클을 필요로 할 수 있다.

일실시예는 EMMS 명령 또는 이행에 대응하여 모든 태그를 바꾸는 것에 대하여 설명되었지만, 대체 실시예는 여러 메카니즘을 이용할 수 있다. 예를 들어, 모든 태그를 공백 또는 비공백 상태로 바꾸는 것은, 새로운 마이크로 연산에 대응하여 전체적으로 그 태그(태그 수정자 유닛(1092)에 대해 하나의 이러한 실시예는 도6B를 참조하여 설명되어 있다)를 바꿀 수 있도록 새로운 마이크로 연산을 포함시키고 그 태그 수정자 유닛(1092)을 구현함으로써 1 클록 사이클내에서 완료될 수 있다. 이러한 실시예에서, EMMS 유닛(1094)은 모든 태그를 공백 상태로 바꾸기 위해 디코드 유닛(1002)이 이러한 단일 마이크로 연산(일부 개별 마이크로 연산보다는)을 발생시키도록 구현되어 있다. 그 반대로, 그 이행 유닛(1036)은 모든 태그를 비공백 상태로 바꾸기 위해 디코드 유닛(1002)이 이러한 단일 마이크로 연산(일부 개별 현재의 마이크로 연산보다는)을 발생시키도록 구현되어 있다. 다른 실시예로서, 대체 실시예는 실행 유닛(1010)을 태그(1034)와 회수 유닛(1006)에 연결시키는 버스를 포함할 수 있다. 이러한 대체 실시예는, EMMS 명령에 대응하여, 프로세서가 직렬화되고(이것은 재명명 유닛(1004)에 의해 수행될 수 있다), 그 신호는 그 태그가 바뀌도록 버스를 통해 전송되고(이것은 실행 유닛(1010)에 의해 수행될 수 있다), 그리고 프로세서가 다시 직렬화(이것은 재명명 유닛(1004)에 의해 수행될 수 있다)되도록 구현될 수 있다. 이러한 실시예는 모

든 태그를 바꾸는 데 약 10-20 클록 사이클을 필요로 할 수 있다. 그 반대로, 이러한 대체 실시예는 전 직렬화 및/또는 후 직렬화가 다른 유닛에 의해 수행되거나 필요하지 않도록 구현될 수 있다. 다른 실시예로서, 디코드 유닛(1002)은 태그(1034)에 연결될 수 있고, EMMS 명령을 수신하는 것에 대응하여 모든 태그(1034)를 바꾸는 추가 하드웨어를 포함한다.

따라서, 도10에 도시된 실시예는 도6A를 참조하여 앞서 설명된 개별 부동점과 압축 데이터보다는 부동점과 압축 데이터 명령을 실행하는 단일 레지스터 세트를 이용한다. 추가로, 도6A의 실시예는 부동점 레지스터를 스택으로서 그리고, 압축 데이터 레지스터를 고정 레지스터 파일로서 액세스하기 위한 개별 회로 소자를 필요로 하지만, FP/PD 매핑 유닛(1030)은 동일 회로 소자를 이용한다. 더욱이, 부동점과 압축 데이터 모드 전후에서 프로세서를 이행하게 할수 있는 도6A를 참조하여 설명된 이행 유닛과 같지 않게, 도10을 참조하여 설명된 이행 유닛은 부동점 모드에서 압축 데이터 모드로의 프로세서의 이행만을 필요로 한다. 더욱이, 단일의 에일리어스된 레지스터 파일은 부동점과 압축 데이터 상태에 사용되기 때문에, 이러한 이행은 두 개의 별개 레지스터 파일사이에서 데이터를 복사하는데 필요하지 않다. 결과적으로, 도10에 도시된 실시예는 회로의 복잡성을 줄이고, 프로세서의 다이 공간을 줄일 필요가 있다.

앞서 설명된 바와 같이, 일실시예는 부동점과 압축 데이터 연산을 수행하는 명령을 포함하고 있는 것에 대해 설명되어 있지만, 대체 실시예는 프로세서가 상이한 데이터 타입 연산을 수행하게 하는 상이한 명령 세트를 구현할 수 있다. 예를 들어, 하나의 명령 세트는 프로세서가 스칼라 연산(부동점 및/또는 정수)을 수행하게 할 수 있고, 다른 명령 세트는 프로세서가 압축 연산(부동점 및/또는 정수)을 수행하게 할 수 있다. 다른 예로서, 하나의 명령 세트는 프로세서가 부동점 연산(스칼라 및/또는 압축)을 수행하게 할 수 있고, 다른 명령 세트는 프로세서가 정수 연산(스칼라 및/또는 압축)을 수행하게 할 수 있다. 다른 예로서, 단일의 에일리어스된 레지스터 파일은 스택 레퍼런스 레지스터 파일로서 그리고 플랫 레지스터 파일로서 동작될 수 있다. 더욱이, 일실시예는 완전 에일리어싱이 구현된 것에 대해 설명되어 있지만, 단일 물리 레지스터 파일을 가진 대체 실시예는 부분적으로 에일리어스된 것으로서 동작하도록 구현될 수 있다. 이것은 어떠한 데이터가 단일의 에일리어스된 물리 레지스터 파일에 저장될 수 있는지의 트랙을 유지하는 일부 메카니즘(예, 테이블)을 필요로 할 수 있다.

도11A, 11B, 및 11C는 본 발명의 다른 실시예에 따라 보이지 않는 시스템을 동작하고, 좋은 프로그래밍 실행을 촉진시키고, 그리고 도10의 하드웨어 정렬을 이용하여 실행될 수 있는 방식으로 단일의 에일리어스된 레지스터 파일상에서 압축 데이터와 부동점 명령을 실행하는 방법을 예시하고 있다. 이 흐름도는 도4A-B와 도7A-C, 도9 및 도10을 참조하여 설명된 흐름도와 유사하다. 이러한 앞선 흐름도를 참조하여, 다수의 대체 실시예는 단계가 바뀌고, 이동되고, 제거되는 것을 설명하고 있다. 앞서 설명된 흐름도에서 수행되는 단계와 유사한 도11A-C를 참조하여 설명된 단계는 이러한 대체 실시예를 이용하여 수행될 수 있음을 알 수 있다. 그 흐름도는 단계(1100)에서 시작한다. 단계(1100)에서 단계(1102)로 진행한다.

단계(1102)에서 도시된 바와 같이, 비트 세트는 명령으로서 액세스되고 단계(1104)로 진행한다. 이러한 비트 세트는 그 명령에 의해 수행하는 연산을 식별하는 연산코드를 포함하고 있다. 따라서, 단계(1102)는 도4A의 단계(402)와 유사하다.

일실시예에서, 그 다음 단계는 파이프라인의 디코드 스테이지에서 수행된다.

단계(1104)에서, 연산 코드가 유효한지를 판단한다. 이 연산 코드가 유효하지 않다면, 단계(1106)로 진행한다. 그 반대면, 단계(1108)로 진행한다. 단계(1104)는 도4의 단계(404)로 진행한다.

단계(1106)에서, 하나 이상의 이벤트 단일 마이크로 연산은, 무효 연산 코드 예외가 발생될 수 있음을 지시하게 삽입된다. 이벤트 신호 마이크로 연산은 파이프라인의 회수 스테이지전까지 오류 제공을 방지하는데 사용된다. 하나의 명령이 이벤트 신호 마이크로 연산이면, 디코드 스테이지, 레지스터 재명명 스테이지, 및 실행 스테이지를 지나 진행한다. 그러나, 이벤트 신호 마이크로 연산이 회수 스테이지에서 수신될 때, 버퍼 레지스터의 상태는 기록되지 않고, 적절한 이벤트가 발생된다. 이벤트 신호 마이크로 연산은 그 이벤트를 유발하는 명령 대신 또는 앞서 삽입된다. 마이크로 연산의 사용은 다렐 디. 보그외 다수의 발명자에 의한 미국 특허 08/203,790 '프로세서로 이벤트의 발생을 신호화하는 장치 및 방법'을 참조하여 추가 설명되어 있다. 단계(1106)에서 단계(1108)로 진행한다.

단계(1108)에서, 어떤 유형의 데이터가 수신되었는지를 판단한다. 그 명령이 부동점 명령도 압축 데이터 명령도 아니다면, 단계(1110)로 진행한다. 따라서, 하나 이상의 이벤트 신호 마이크로 연산은 단계(1106)에 삽입되면, 단계(1110)로 진행한다. 그러나, 그 명령이 부동점 명령이면, 단계(1112)로 진행한다. 그 반대로, 그 명령이 압축 데이터 명령이면, 단계(1114)로 진행한다. 따라서, 단계(1108)는 도4A의 단계(408)와 유사하다.

단계(1110)에 도시된 바와 같이, 프로세서는 그 명령을 실행한다. 단계(1106)에서 무효 연산 코드 예외가 발생될 수 있음을 지시하는 하나 이상의 마이크로 연산이 삽입되면, 그 마이크로 연산은 디코드 스테이지, 레지스터 재명명 스테이지, 및 실행 스테이지를 지나 진행한다. 그러나, 그 이벤트 신호 마이크로 연산이 회수 스테이지에 도달할 때, 버퍼 레지스터의 상태는 기록되지 않고, 무효 연산 코드 예외가 발생된다. 도2의 단계(215)를 참조하여 앞서 설명된 바와 같이, 이 이벤트 핸들러는 프로세서가 메시지를 디스플레이하게 하고, 현재의 태스크의 실행을 중지하게 하고, 그리고 다른 태스크를 실행하게 진행하게 하도록 구현될 수 있다. 물론, 대체 실시예는 앞서 설명되었던 여러 방법으로 이러한 핸들러를 구현할 수 있다. 다른 명령의 실행은 본 발명을 이해하는 데 필요하지 않기 때문에, 추가 설명되어 있지 않다.

단계(1112)에 도시된 바와 같이, EM 지시가 1 인지를(설명된 소프트웨어 규정에 따라, 부동점 유닛이 대리 실행되면), 그리고 TS 지시가 1 인지를(설명된 소프트웨어에 따라, 부분 콘텍트 스위치가 수행되었다면) 판단한다. EM 지시 및/또는 TS 지시가 1 이면, 단계(1116)로 진행한다. 그 반대면, 단계(1120)로 진행한다. 따라서, 단계(1112)는 도4A의 단계(412)와 유사하다.

단계(1116)에서, 하나 이상의 이벤트 신호 마이크로 연산은 예외를 이용불가능한 디바이스가 발생될 수 있음을 지시하도록 삽입된다. 단계(1116)에서, 단계(1120)로 진행한다.

단계(1114, 1120)에 도시된 바와 같이, 레지스터 재명명이 수행된다. 단계(1120)에서, 단계(1122)로 진행한다. 그 반대로, 단계(1114)에서, 단계(1134)로 진행한다. 일실시예에서, 단계(1114, 1120)는 파이프라인의 재명명 스테이지에서 수행된다.

일실시예에서, 그 다음 스테이지는 파이프라인의 실행 스테이지에서 수행된다.

단계(1122)에서 도시된 바와 같이, 부동점 명령이 실행된다. 단계(1122)는 도4B의 단계(426)와 유사하다. 보이지 않는 시스템을 동작시키기 위해, 일실시예는 필요한 것으로 그 태그를 바꾸고, 현재 제공될 수 있는 숫자 오류를 기록하고, 그리고 보류한 임의의 다른 숫자 오류를 유지한다. 앞서 설명된 바와 같이, 그 태그를 바꾸는 것은, 대응 태그가 비공백 상태를 지시하는 부동점 레지스터만의 내용을 저장하는 이러한 운영 체제 기술에서 볼수 없는 시스템 동작을 이러한 실시예가 가능하게 한다. 그러나, 대체 실시예는 특정 운영 체제 기술과 호환되도록 구현될 수 있다. 예를 들어, 현재의 운영 체제가 태그를 이용하지 않는다면, 그 태그를 구현하지 않는 프로세서는 그 운영 체제와 여전히 호환성이 있을 수 있다. 더욱이, 본 발명에서, 숫자 부동점 예외가 보류되어야 할 필요는 없고, 따라서, 보류하지 않은 대체 실시예는 여전히 본 발명의 범위내에 있다. 단계(1122)에서, 단계(1124)로 진행한다.

단계(1134)에서, 압축 데이터 명령이 EMMS 명령인지를 판단한다. 따라서, 단계(1134)는 도4B의 단계(430)와 유사하다. 압축 데이터 명령이 EMMS 명령이면, 단계(1138)로 진행한다. 그 반대면, 단계(1138)로 진행한다. 앞서 설명된 바와 같이, EMMS 명령은 부동점 태그를 초기 상태로 바꾸는 데 사용되고, 프로세서를 부동점 모드로 이행하기 위해 임의의 압축 데이터 명령을 실행한 후 및/또는 임의의 부동점 명령을 실행하기 전 실행될 수 있다.

단계(1136)에 도시된 바와 같이, 모든 태그는 공백 상태로 바뀌게 된다. 이러한 방식으로, 그 태그는 초기화되었고, 부동점 명령의 실행을 준비한다. 단계(1136)의 완료와 동시에, 단계(1144)로 진행한다. EMMS 명령이 구현되지 않은 실시예에서, 단계(1134,1136)가 생략될 수 있고, 단계(1114)에서 단계(1138)로 진행될 수 있다.

단계(1138)에 도시된 바와 같이, 압축 데이터 명령이 실행된다. 이러한 단계동안에, 압축 데이터가 기록되는 FP/PD 레지스터로서 작동하는 임의의 버퍼 레지스터 또는 FP 레지스터의 부호와 지수 필드에 1이 저장된다. 따라서, 단계(1138)는 도4B의 단계(434, 436, 438)와 유사하다. 그렇게 하면는 부동점과 압축 데이터 명령의 분리시킴으로써 좋은 프로그래밍 기술을 진전시킨다. 그러나, 앞서 설명된 바와 같이, 대체 실시예는 이러한 특성을 구현하는 것을 피할 수 있다. 일실시예에서는 부호 필드와 지수 필드에 1이 기록되지만, 대체 실시예는 NAN(숫자가 아님) 또는 무한대를 표시하는 임의의 값을 사용할 수 있다. 더욱이, 이 단계는 임의의 숫자 예외를 발생시킴이 없이 수행된다. 임의의 메모리 이벤트는 압축 데이터 명령 실행을 시도한 결과로서 발생되면, 예외가 인터럽트되고 그 이벤트는 제공된다. 단계(1138)에서, 단계(1144)로 진행한다.

일실시예에서, 그 다음 단계는 파이프라인의 회수 스테이지에서 수행된다.

단계(1124)에서, 그 명령이, 예외를 이용할 수 없는 디바이스를 지시하는 이벤트 신호 마이크로 연산인지를 판단한다. 그러하다면, 단계(1112)에서, TS 및 EM 지시 모두 또는 어느 하나가 1 인지를 판단한다. 따라서, 그 명령이, 예외를 이용할 수 없는 디바이스를 지시하는 이벤트 신호 마이크로 연산이면, 단계(1126)로 진행한다. 그 반대면, 단계(1128)로 진행한다. 이러한 방식으로, 예외를 이용할 수 없는 디바이스는 레지스터 재명명을 이용하는 프로세서에 포함될 수 있다.

단계(1126)에서, 예외를 이용할 수 없는 디바이스가 발생되고, 그 대응 이벤트 핸들러가 실행된다. 따라서, 단계(1126)는 도4A의 단계(416)와 유사하다. 앞서 설명된 바와 같이, 이러한 이벤트 핸들러는 부동점 명령을 대리 실행하는지를 또는 부분 콘텍트 스위치가 수행되었는지를 판단하기 위해 EM 및 TS 지시를 사용하도록 구현될 수 있다. 또한 앞서 설명된 바와 같이, EM 및 TS 지시의 사용은 소프트웨어 규정이고, 그러므로 다른 목적으로 사용될 수 있다.

단계(1144)에 도시된 바와 같이, EM 지시가 1 인지를 판단한다. 따라서, 단계(1144)는 도4A의 단계(414)와 유사하다. 단계(1144)에서 EM 지시가 1 이다고 판단되면, 단계(1126)보다는 단계(1146)로 진행한다. 그 반대면, 단계(1146)로 진행한다.

단계(1146)에서, 무효 연산 코드 예외가 발생되고, 적절한 이벤트 핸들러가 실행된다. 이것은 도11A의 단계(1110)를 참조하여 설명된 무효 연산 코드 예외와 동일하다. 무효 연산코드 예외의 발생은 도4A의 단계(406)에서 발생된 무효 연산 코드 예외와 유사하다. 도2의 단계(215)를 참조하여 앞서 설명된 바와 같이, 이 이벤트 핸들러는 프로세서가 메시지를 디스플레이하고, 현재의 태스크 실행을 중지하고, 그리고 다른 태스크를 실행하는 것으로 진행한다. 물론, 대체 실시예는 앞서 설명되었던 다수의 방법을 구현될 수 있다. EM 이 1인 동안 압축 데이터 명령 실행의 시도를 무효 연산 코드 예외로 전환함으로써, 그 실시예는 보이지 않는 시스템을 동작시킨다.

일실시예는 보이지 않는 시스템을 동작하는 방식으로 EM 지시를 조정하는 것에 대해 설명되었지만, 대체 실시예는 다른 기술을 사용할 수 있다. 예를 들어, 대체 실시예는, EM 지시가 1 인 동안 압축 데이터 명령의 시도된 실행에 응답하여. 새로운 이벤트, 예외를 이용할 수 없는 디바이스, 상이한 현재의 이벤트 중 하나를 발생시킬 수 있다. 다른 예로서, 대체 실시예는 압축 데이터 명령을 실행할 때, EM 지시를 무시할 수 있다.

단계(1148)에 도시된 바와 같이, TS 지시가 1 인지를 판단한다(설명된 소프트웨어 규정에 따라, 부분 콘텍트 스위치가 수행되었다면). 부분 콘텍트 스위치가 수행되었다면, 단계(1126)로 진행한다. 그 반대면, 단계(1150)로 진행한다.

앞서 설명된 바와 같이, 단계(1126)에서 예외를 이용할 수 없는 디바이스가 발생되고, 그 대응 이벤트 핸들러가 실행된다. 따라서, 이러한 이벤트에 대응하여, 그 대응 이벤트 핸들러는 EM 지시와 TS 지시를 폴링하도록 구현될 수 있다. 그러나, 압축 데이터 명령이 실행될 때, 단계(1144)로 진행하고, EM 지시가 1인 상황은 무효 연산 코드 예외로 전환된다. 결과적으로, 압축 데이터 명령이 실행되게 되고, 단계(112

6)에 도달될 때, EM 명령은 0 이여야 하고, TS 지시는 1이여야 한다. TS 지시가 1이기 때문에, 그 이벤트 핸들러는 부분 콘텍트 스위치를 참조하여 앞서 설명된 바와 같이 작용하고, 단계(1102)에서 수신된 명령의 실행을 재시작함으로써 프로세서가 실행을 재개시하게 한다. 압축 데이터 상태가 부동점 상태에서 에일리어스되기 때문에, 이 이벤트 핸들러는 부동점 상태와 압축 데이터 상태 모두에 작용한다.
결과적으로, 이 방법은 보이지 않는 시스템을 동작시킨다. 물론, 대체 실시예는 앞서 설명된 다수의 방법으로 이러한 이벤트 핸들러를 구현할 수 있다. 일실시예는 보이지 않는 시스템을 동작시키는 방식으로 TS 지시를 조정하는 것에 대해 설명되었지만, 대체 실시예는 앞서 설명된 바와 같은 다른 기술을 사용할 수 있다.

앞서 설명된 바와 같이, 특정 숫자 오류가 부동점 명령의 실행동안에 발생되면, 그 오류는 실행이 오류가 제공되는 것을 인터럽트할 수 있는 다음 부동점 명령의 시도된 실행전까지 보류된다. 단계(1128,1150)에 도시된 바와 같이, 제공될 수 있는 이러한 보류 오류가 있는지를 판단한다. 따라서, 이러한 단계는 도4A의 단계(420, 422)와 유사하다. 그러나, 단계(1128)에서 이러한 보류 오류가 없다고 판단되면, 단계(1132)로 진행한다. 그 반대로, 단계(1150)에서 이러한 보류 오류가 없다고 판단되면, 단계(1152)로 진행한다. 대체 실시예에서, 단계(1150)는 수행되지 않고, 부동점 오류는 압축 데이터 명령의 실행동안에 보류된다.

단계(1130)에서, 보류한 부동점 오류 이벤트가 발생된다. 따라서, 단계(1130)는 도4A의 단계(424)와 유사하다. 도2의 단계(424)를 참조하여 앞서 설명된 바와 같이, 이러한 이벤트는 내부 이벤트 또는 외부 이벤트중 하나로서 취급되어서 제공될 수 있다.

단계(1152)에 도시된 바와 같이, 프로세서가 압축 데이터 모드에 있는지를 판단한다. 프로세서가 압축 데이터 모드에 있으면, 압축 데이터 명령의 실행은 성공적으로 완료되었고, 단계(1132)로 진행한다. 그러나, 프로세서가 압축 데이터 모드에 있지 않으면, 그 압축 데이터 명령은 부동점 모드에서 실행되었다. 결과적으로, 압축 데이터 명령의 실행은 정확하지 않다. 이것을 고치기 위해, 프로세서는 부동점 모드에서 압축 데이터 모드로 전환되어야 하고, 압축 데이터 명령은 재실행되어야 한다. 결과적으로, 프로세서가 압축 데이터 모드에 있지 않으면, 단계(1154)로 진행한다. 단계(1152)에서의 판단은 다수의 방법으로 수행될 수 있다. 예를 들어, 도6A를 참조하여 앞서 설명된 모드 지시가 사용될 수 있다. 다른 예로서, 스택 지시의 상부와 태그는 폴링될 수 있다. 스택 지시의 상부가 초기 상태에 있고 모든 태그가 비공백 상태에 있다면, 프로세서는 압축 데이터 모드에 있다. 그러나, 스택 지시의 상부가 초기 상태에 있지 않거나 모든 태그가 비공백 상태에 있지 않으면, 프로세서는 압축 데이터 모드에 있지 않다.

단계(1154)에서, 프로세서는 부동점 모드에서 압축 데이터 모드로 이행되고, 단계(1156)로 진행한다. 단계(1154)에서, 프로세서는 모든 태그를 비공백 상태로 바꿈으로써 그리고 스택 지시의 상부를 초기값으로 바꿈으로써 부동점 모드에서 압축 데이터 모드로 이행된다. 모든 태그를 비공백 상태로 바꾸는 것은 부동점 명령과 압축 데이터 명령을 개별적으로 그룹화하게 하는 좋은 프로그래밍 기술을 진전시킨다. 추가로, 운영 체제 호환성 측면에서, 특정 운영 체제 기술은 대응 태그가 비공백 상태를 지시하는 부동점 레지스터의 내용만을 저장한다. 따라서, 압축 데이터 상태가 부동점 상태에서 에일리어스된 실시예에서, 모든 태그를 비공백 상태로 바꾸는 것은, 부동점 상태였던 것처럼 이러한 운영 체제가 압축 데이터 상태를 보존하게 한다. 대체 실시예는 이러한 운영 체제 기술보다는 적게 호환되게 구현될 수 있다. 예를 들어, 운영 체제가 그 태그를 이용하지 않으면, 그 태그를 구현하지 않는 실시예는 그 운영 체제와 여전히 호환될 수 있다. 스택 지시의 상부를 0으로 바꾸는 것은 앞서 설명된 효과적인 프로그래밍 기술을 수행하는 데 사용된다. 추가로, 스택 지시의 상부를 초기값으로 바꾸는 것과 스택 지시의 상부를 압축 데이터 명령의 실행동안에 바꾸지 않는 것은, 도10을 참조하여 앞서 설명된 고정 레지스터 파일로서 그리고 부동점 스택으로서 FP/PD 레지스터를 동일 회로 소자가 동작하게 하는데 사용된다. 부동점과 압축 데이터 상태가 단일 레지스터 파일상에서 에일리어스되기 때문에, 이행하는 것은 별개의 부동점과 압축 데이터 레지스터 파일사이에서 복사될 데이터를 필요로 하지 않는다. 이것은 부동점과 압축 데이터 모드사이의 이행에 필요한 시간의 양을 감소시킨다. 앞서 설명된 바와 같이, 부동점에서 압축 데이터로의 이행은 마이크로모드에서 구현될 수 있다. 대체 실시예에서, 각각의 압축 데이터 명령의 실행은 스택 지시의 상부를 초기값으로 바꾼다.

단계(1156)에 도시된 바와 같이, 단계(1102)에서 수신된 명령은 마이크로 재시작을 수행함으로써 재시작된다. 마이크로-재시작이 사용되기 때문에, 현재의 태스크의 실행은 프로세서 외부에 어떠한 동작도 발생함이 없이 재실행될 수 있다 -- 어떠한 비마이크로코드 이벤트 핸들러는 실행될 필요가 없다. 이러한 방식으로, 이러한 실시예는 현재의 운영 체제와 호환될 수 있다. 대체 실시예는 보다 적게 호환되게 구현될 수 있다. 예를 들어, 추가 이벤트는 프로세서에 포함될 수 있고, 추가 이벤트 핸들러는 이러한 이행을 수행하기 위해 운영 체제에 추가될 수 있다.

단계(1132)에서, 버퍼 레지스터의 상태는 그 대응 FP/PD 또는 정수 레지스터에 기록된다. 단계(1132)의 완료와 동시에, 프로세서는 연속 실행이 허용된다.

따라서, 현재의 운영 체제와 호환성이 있으며, 좋은 프로그래밍 기술을 진전시키는 압축 데이터 명령를 실행하는 방법이 설명된다. 압축 데이터 상태는 부동점 상태상에서 에일리어스되기 때문에, 그 압축 데이터 상태는, 부동점 상태이었던 것처럼 현재의 운영 체제에 의해 보존되고 재저장될 것이다. 더욱이, 압축 데이터 명령의 실행에 의해 발생된 이벤트는 현재의 운영 체제 핸들러에 의해 제공되기 때문에, 이 이벤트 핸들러는 수정될 필요가 없고, 새로운 이벤트 핸들러는 추가될 필요가 없다. 결과적으로, 프로세서는 역으로 호환성이 있고, 업그레이드하는 것은 운영 체제를 발달시키고 수정하는데 필요한 비용과 시간을 필요로 하지 않는다.

일부가 설명된 이러한 실시예의 변화는 이러한 운영 체제와 완전히 또는 부분적으로 호환될 수 있고, 좋은 프로그래밍 기술을 진전시킨다. 예를 들어, 대체 실시예는 이러한 흐름도에서 하나 이상의 단계를 이동시키고, 바꾸고, 제거할 수 있다. 특정 단계가 도11A, 도11B, 및/또는 도11C에서 제거되면, 도10에서 특정 하드웨어가 필요하지 않을 수 있다. 예를 들어, TS 지시가 이용되지 않으면, TS 지시는 필요하지 않다. 물론, 본 발명은 다수의 시스템 아키텍처에 유용할 수 있고, 여기서 설명된 아키텍처에 제한을 두지

않는다.

도12A, 도12B, 및 도12C는 도10을 참조하여 설명된 실시예에 따라 부동점 데이터, 압축 데이터, 정수 데이터를 저장하는 기억 포맷을 예시하고 있다. 물론, 대체 실시예는 부동점 데이터, 압축 데이터, 정수 데이터를 저장하는 다수의 상이한 기억 포맷을 이용할 수 있다.

도12A는 도10을 참조하여 설명된 본 발명의 일실시예에 따른 부동점 기억 포맷을 예시하고 있다. 도12A는 비트[85]로 구성된 부호 필드(1202), 비트[84:68]로 구성된 지수 필드(1204), 비트[67:3]로 구성된 소수 필드(1206), 비트[2:0]로 구성된 라운딩 필드(1208)를 포함하고 있는 부동점 기억 포맷(1200)을 도시하고 있다. 앞서 설명된 바와 같이, 태스크 스위치를 수행할 때 메모리에 부동점 상태를 저장하는데 사용된 동일 부동점 명령은 부동점 레지스터상에서 에일리어스된 임의의 압축 데이터 상태를 저장하는데 또한 작용하여야 한다. 일실시예에서, 프로세서는 라운딩 필드(1028)에 그 라운딩 비트를 저장할 수 없다. 결과적으로, 그 압축 데이터는 부동점 기억 포맷(1200)의 소수부 필드(1206)내에서 어디든지 저장되어야 한다.

도12B는 도10을 참조하여 설명된 본 발명의 실시예에 따른 압축 데이터용 기억 포맷을 예시하고 있다. 도12B는 비트[85:68]로 구성된 부호/지수 필드(1212), 비트[67]로 구성된 제 1 예약 필드(1214), 비트[66:3]로 구성된 압축 데이터 필드(1216), 및 비트[2:0]로 구성된 제 2 예약 필드(1218)를 포함하고 있는 압축 데이터 기억 포맷(1210)을 도시하고 있다. 앞서 설명된 바와 같이, 압축 데이터가 레지스터에 기록될 때 부호/지수 필드(1212)에 모든 1이 저장된다. 또한 앞서 설명된 바와 같이, 압축 데이터 필드(1216)는, 현재의 부동점 명령이 압축 데이터 상태로 저장될 수 있도록 소수부 필드(1206)상에서 에일리어스된다. 일실시예에서, 제 1 및 제 2 예약 필드(1214,1218)는 압축 데이터가 레지스터에 기록될 때 0으로 기록된다. 본 발명의 일실시예는 압축 데이터 기억 포맷의 압축 데이터 필드(1216)이 부동점 기억 포맷(1200)의 소수부 필드(1206)와 동일 비트 위치에서 시작하는 것을 설명하였지만, 대체 실시예는 이러한 관계를 바꿀 수 있다.

도12C는 도10을 참조하여 설명된 본 발명의 실시예에 따라 정수 데이터용 기억 포맷을 예시하고 있다. 도12C는 비트[85:32]로 구성된 예약 필드(1222)와 비트[31:0]로 구성된 정수 데이터 필드(1224)를 포함하고 있는 정수 데이터 기억 포맷(1220)을 도시하고 있다. 일실시예는 정수 데이터가 32비트에 저장되어 있는 것을 설명하지만, 대체 실시예는 다수의 비트를 사용한 하나 이상의 포맷에 정수 데이터를 저장하도록 구현될 수 있다. 예를 들어, 대체 실시예는 64비트 포맷을 지원할 수 있다. 일실시예에서, 볼 수 있는 소프트웨어인 정수 레지스터(1024)의 각각은 단지 32비트만을 포함하고 있다. 결과적으로, 정수 기억 포맷(1220)은 버퍼 레지스터(1020)에서만 사용된다.

도13은, 본 발명의 일실시예에 따라, 도12A, 도12B, 도12C를 참조하여 설명된 기억 포맷이 구현될 때 도11B의 단계(1138)를 수행하는 방법을 예시하고 있다. 단계(1138)에서 단계(1300)로 진행한다.

단계(1300)에서, 압축 데이터 명령이 FP/PD 레지스터처럼 작동하는 버퍼 레지스터를 통해 임의의 FP/PD 레지스터로부터 압축 데이터를 검색하는지를 판단한다. 그러하다면, 단계(1302)로 진행한다. 그 반대면, 단계(1308)로 진행한다.

단계(1302)에서 보는 바와 같이, 이 에일리어스된 버퍼 또는 FP/PD 레지스터로부터의 비트[66:3]는 검색되고, 단계(1308)로 진행한다. 이 단계는 압축 데이터가 비트(0)에서 시작하여 저장되는 것이 아니라 도12B에 도시된 비트(3)에서 시작하여 저장되는 점에서 필요하다. 결과적으로, 비트[2:0]는 제거되어야 한다. 일실시예에서, 이 단계는 도10의 데이터 정렬 유닛(1090)에 의해 수행된다. 이러한 실시예에서, 그 데이터는 도12B에 도시된 포맷으로, 발생 유닛(1008)을 경유하여, 회수 유닛(1006)에서 실행 유닛(1010)으로 전송된다. 따라서, 데이터는 실행 유닛(1010)에 의해 도12B에 도시된 포맷으로 수신되고, 데이터 정렬 유닛(1090)은 비트[66:3]를 추출하는 것이 가능하다. 도10은 단일 데이터 정렬 유닛을 도시하고 있지만, 일실시예에서는, 압축 데이터를 통해 동작하는 실행 유닛(1010)내의 각각의 함수 유닛은 비트[63:3]를 추출하는 데이터 정렬 유닛(1010)을 포함하고 있다. 그 데이터는 실행 유닛(1010)에서 정렬되기 때문에, 프로세서의 나머지에서 압축 데이터 포맷은 분명히 사용된다. 그 데이터 정렬 유닛은 다수의 기술을 사용하여 비트[66:3]에 액세스하도록 구현될 수 있다. 예를 들어, 일실시예에서, 데이터 정렬 유닛은 FP/PD 레지스터처럼 작용하는 버퍼 레지스터 또는 FP/PD 레지스터로부터 검색된 모든 압축 데이터를 3 비트만큼 오른쪽으로 이동시키도록 설계되어 있다. 대체 실시예에서, 회수 유닛 또는 발생 유닛은 비트[2:0] 및/또는 비트[85:67]를 제거하도록 구현될 수 있다. 다른 예로서, 대체 실시예는 압축 데이터가 비트(0)에서 시작하여 저장되도록 구현될 수 있다.

단계(1304)에서, 압축 데이터 명령이 임의의 정수 레지스터 또는 정수 레지스터처럼 작용하는 임의의 버퍼 레지스터로부터 압축 데이터를 검색하는지를 판단한다. 그러하면, 단계(1306)로 진행한다. 그 반대면, 단계(1308)로 진행한다.

단계(1306)에서 보는 바와 같이, 그 에일리어스된 버퍼 또는 정수 레지스터로부터의 비트[31:0]가 검색되고, 단계(1308)로 진행한다. 이 단계는 그 데이터가 비트(0)에서 시작하여 저장되는 점에서 필요하다. 앞서 설명된 바와 같이, 일실시예에서, 이 단계는 도10의 데이터 정렬 유닛(1090)에 의해 수행된다. 이러한 실시예에서, 데이터는 발생 유닛(1008)을 경유하여, 회수 유닛(1006)에서 실행 유닛(1010)로 전송된다. 그 데이터가 버퍼 레지스터(1020)으로부터 액세스되면, 그 데이터는 도12C에 도시된 포맷으로 실행 유닛(1010)에 의해 수신되고, 그 데이터 정렬 유닛은 비트[31:0]를 추출하는 것이 가능하게 된다. 그러나, 그 데이터가 그 정수 레지스터(1024)가 32 비트인 실시예내의 정수 레지스터(1024)로부터 액세스되면, 그 데이터는 32 비트 포맷으로 실행 유닛(1010)에 의해 수신된다. 한가지 경우에서, 32 비트 데이터는 64 비트의 압축 데이터 항목중 하나로서 취급될 수 있다. 예를 들어, 제 1 이동 명령은 정수 레지스터에서 압축 데이터 항목중 상위 비트로 32비트를 이동하도록 구현될 수 있지만, 제 2 이동 명령은 정수 레지스터에서 압축 데이터 항목중 하위 32 비트로 32 비트를 이동하도록 구현될 수 있다.

단계(1308)에서 보는 바와 같이, 그 명령이 필요로 하는 동작이 수행되고 단계(1310)로 진행한다.

단계(13010)에서, 압축 데이터 명령이 프로세서를 FP/PD 레지스터로서 작동하는 버퍼 레지스터 또는 임의의 FP/PD 레지스터에 기록하게 하는지를 판단한다. 그러하면, 단계(1312)로 진행한다. 그 반대면, 단계

(1314)로 진행한다.

압축 데이터 명령이 프로세서를 FP/PD 레지스터로서 작동하는 버퍼 레지스터 또는 임의의 FP/PD 레지스터에 기록하게 하면, 그 데이터는 적당한 포맷으로 저장되어야 한다. 따라서, 단계(1312)에서, 그 압축 데이터는 FP/PD 또는 버퍼 레지스터의 비트[66:3]로 저장된다. 일실시예에서, 도10의 데이터 정렬 유닛(1090)은 재사용된다. 다시, 이러한 기능을 수행하는 다수의 기술이 있다. 예를 들어, 데이터 정렬 유닛은 3 비트만큼 왼쪽으로 그 데이터를 이동하고, 비트[2:0]를 0으로 채우고, 비트[67]를 0으로 채우고, 그리고 비트[85:68]에 1를 저장하도록 구현될 수 있다. 대체 실시예에서, 회수 유닛은 이러한 포맷으로 그 데이터를 저장하도록 구현될 수 있다.

단계(1314)에서, 압축 데이터 명령이 프로세서를 임의의 정수 레지스터 또는 정수 레지스터처럼 작동하는 임의의 버퍼 레지스터에 기록하게 하는지를 판단한다. 그러하다면, 단계(1316)로 진행한다. 그 반대면, 단계(1144)로 진행한다.

압축 데이터 명령이 프로세서를 임의의 정수 레지스터 또는 정수 레지스터처럼 작동하는 임의의 버퍼 레지스터에 기록하게 한다면, 압축 데이터는 적당한 정수 기억 포맷으로 저장되어야 한다. 따라서, 단계(1316)에서 그 데이터는 비트[31:0]로서 정수 레지스터내에 있거나, 비트[63:0]또는 [31:0](구현에 따라)로서 버퍼 레지스터내에 있다. 64비트 데이터가 있기 때문에, 그 데이터의 임의의 32비트는 그 레지스터에 저장될 수 있다. 예를 들어, 제 1 이동 명령은 압축 데이터 항목의 상위 비트를 정수 레지스터로 이동하도록 구현될 수 있지만, 제 2 이동 명령은 압축 데이터 항목의 하위 32 비트를 정수 레지스터로 이동하도록 구현될 수 있다. 일실시예에서, 이 단계는 도10의 데이터 정렬 유닛(1000)에 의해 다시 수행된다. 물론, 다수의 기술은 앞서 설명된 것을 포함하여, 단계(1316)를 구현하는데 사용될 수 있다.

이러한 방식으로, 상이한 타입의 데이터에 의해 사용된 기억 포맷은 프로세서의 레지스터내에서 적당히 정렬된다. 일실시예에서, 동일 기억 포맷은 FP/PD 레지스터(1022)와 정수 레지스터(1024)에서 사용되는 버퍼 레지스터(1020)에서 사용된다. 물론, 대체 실시예는 다수의 상이한 기억 포맷을 사용할 수 있고, 따라서, 이러한 대체 실시예는 여전히 본 발명의 범위내에 있을 수 있다. 예를 들어, 하나의 대체 실시예는 버퍼 레지스터 세트(1020)에서 이 데이터 기억 포맷을 사용하고, 소프트에어에서 볼 수 있는 레지스터(예, FP/PD 레지스터(1022) 및 정수 레지스터(1024))에서 상이한 데이터 기억 포맷을 사용한다.

앞서 설명된 바와 같이, 부동점 모드와 압축 데이터 모드사이의 이행은 시간을 소모할 수 있고, 이것은 효율적인 프로그래밍 실행이 아니다. 프로그래머가 이러한 다수의 이행을 수행하고 있는지를 판단할 때 돕기 위해, 상이한 성능 모니터링 기술을 사용할 수 있다. 예를 들어, 일실시예에서 성능 모니터 카운터가 사용된다. 성능 모니터 카운터는 프로그래머에서 볼 수 있고, 상이한 조건이 그 프로세서에서 충족되는 시간의 수를 카운트한다. 본 발명의 일실시예에서, 이 조건중 하나는 부동점과 압축 데이터 모드사이의 이행이다. 이러한 방식으로, 프로그래머는 얼마나 많은 이행이 프로그램을 필요로 하는지를 알 수 있을 것이다. 프로그램 카운터에 관한 추가 정보에 대하여, 로버트 에스.드레이어외 다수에 의한 미국 특허 07/883,845의 '프로세서의 성능을 모니터하는 장치'를 참조하라.

종래의 부동점 프로세서는 부동점 태그의 직접 조작이 불가능하기 때문에, 부동점 명령을 이용한 EMMS 명령의 대리 실행이 수행된다.

도14는 본 발명의 일실시예에 따른 태그를 소거하는 방법을 설명하는 흐름도이다. 이 흐름도는 메모리내의 소정된 위치에 부동점 환경을 저장함으로써 단계(1402)에서 시작한다. 이것은 인텔 아키텍처 프로세서내의 FNSAVE 또는 FSAVE 명령을 이용하여 수행된다. 이것이 일단 수행되면, 그 환경이 저장되어 있는 소정된 메모리 위치의 태그 및/또는 TOS 부분은 단계(1404)에서 공백 상태로 수정될 수 있다. 이것은 태그와 TOS 비트에 대한 적절한 비트 패턴용 즉 피연산자를 가진 MOV 명령을 포함한 다수의 종래 명령을 이용하여 수행된다. 소정의 메모리 위치의 태그와 TOS 부분을 공백 상태로 설정할 수 있는 다른 적절한 명령이 사용될 수 있다. 그 다음, 그 환경은 수정된 소정 메모리 위치로부터 단계(1406)에서 재장전될 수 있다. 그 환경의 다른 부분(제어 워드, 상태 워드등)은 수정되지 않을 수 있기 때문에, 부동점 태그를 수정하면, 환경의 나머지는 기억 환경 동작(1402)으로부터 변하지 않는다. 예기치 않은 인터럽트가 발생되는 것을 방지하기 위해 프로세스의 실시예는 인터럽트를 불가능하게 하는 명령의 사용(예, FNSTENV)을 포함한 공지된 종래 기술을 이용하여 구현될 수 있다는 것을 추가로 알 수 있다. 하여간, 그 환경은 FRSTOR 또는 FLDENV 와 같은 종래의 기술을 사용하여 재장전되었기 때문에, 현재 그 환경은 공백 상태로 수정된 부동점 태그만으로 재장전되었다. 단계(1404)는 스택 필드(350)의 상부에 저장된 스택 지시의 상부를 포함한 부동점 환경 부분을 소거하는 추가 단계를 더 포함할 수 있다는 것을 알 수 있다.

또 다른 대체 실시예에서, EMMS 명령은 모든 태그가 공백일 때까지 다수의 충분한 시간동안 부동점 레지스터를 팝핑함으로써 대리 실행될 수 있다. 어느 하나의 이벤트에서, EMMS는 전용 명령으로서 수행될 수 있고, 대리 실행될 수 있고, 그 방법은 본 발명의 기술내에 있다.

도15A는 에일리어스된 별개의 물리 레지스터가 갱신될 수 있는 동안의 시간 간격을 설명하기 위해 압축 데이터와 부동점 명령을 포함하여, 실행 스트림을 도시하고 있다. 도15A는 압축 데이터 명령 세트(1510)에 의해 수반되는 부동점 명령(1500)을 도시하고 있다. 추가로, 도15A는, 부동점 명령(1500)은 시간(T1)에서 실행되지만, 압축 데이터 명령 세트(1510)의 실행은 시간(T2)에서 시작한다는 것을 도시하고 있다. 부동점 명령 세트(1500)의 실행은 프로세서가 부동점 레지스터에 하나의 값을 기록하게 한다. 간격(1520)은 이 값이 에일리어스되어야 하는 동안의 시간(T1)과 시간(T2)사이의 시간을 표시한다. 예를 들어, 별개의 물리 레지스터 파일은 부동점과 압축 데이터 명령을 실행하는 데 사용되는 도6A-9를 참조하여 설명된 일실시예에서, 부동점 상태는 시간(T2)전까지 물리 부동점 레지스터에서 그 대응 물리 압축 데이터 레지스터로 복사되지 않는다(다른 값이 시간(T2)에 앞서 동일 물리 레지스터 파일에 기록되지 않는다고 가정하면). 그 반대로, 단일 물리 레지스터 파일이 사용될 때(도10-11C를 참조하여 설명된 실시예), 부동점 값은 시간(T1)에 그 에일리어스된 레지스터에 저장된다.

따라서, 그 간격(1520)의 두 개의 극단이 설명되어 있다. 그러나, 간격(1520)동안에 그 레지스터를 에일리어스하는 대체 실시예는 구현될 수 있다. 예를 들어, 부동점과 압축 데이터 명령을 실행하는 개별 물리 레지스터 파일을 사용하는 대체 실시예는 부동점 물리 레지스터 파일에 기록된 데이터가 시간(T1)에 압축

데이터 물리 레지스터 파일에 또한 기록되도록 구현될 수 있다. 동시에(예, 시간(t1)에) 양 물리 레지스터 파일에 그 값을 기록하는 일실시예에서, 그 데이터를 부동점 레지스터에서 압축 데이터 레지스터로 복사하는 이행 유닛의 그 부분은 하드웨어로서 구현될 수 있다(물론, 대체 실시예는 소프트웨어, 펌웨어 및/또는 하드웨어를 사용할 수 있다). 다른 예로서, 부동점과 압축 데이터 명령을 실행하는 별개 물리 레지스터 파일을 사용하는 대체 실시예는 부동점 물리 레지스터 파일에 기록된 데이터가 자유 프로세싱 시간이 간격(1520)(가끔 시간(t2)전에)동안에 이용할 수 있을 때 압축 데이터 물리 레지스터 파일에 기록되도록 구현될 수 있다. 이러한 방식으로, 이러한 실시예는 이행 시간을 줄일 수 있다.

도15B는 에일리어스된 별개 물리 레지스터 파일이 갱신될 수 있는 동안의 시간 간격을 설명하기 위해 압축 데이터와 부동점 명령을 포함하여, 실행 스트림을 도시하고 있다. 도15A는 부동점 명령 세트(1540)에 의해 압축 데이터 명령(1530)이 수반되는 것을 제외하고 도15B와 유사하다. 도15A는, 압축 데이터 명령(1530)이 시간(T1)에 실행되고, 부동점 명령 세트(1540)의 실행은 시간(T2)에 시작되는 것을 도시하고 있다. 압축 데이터 명령(1530)의 실행은 프로세서가 압축 데이터 레지스터에 하나의 값을 기록하게 한다. 간격(1550)은 이 값이 에일리어스되어야 하는 동안의 시간(T1)과 시간(T2)사이의 시간을 표시한다. 도15A(압축 데이터 명령에 의해 수반되는 부동점 명령을 참조하여)를 참조하여 설명된 모든 대체 실시예는 도15B(부동점 명령에 의해 수반되는 압축 데이터 명령을 참조하여)를 참조하여 또한 구현될 수 있다.

본 발명은 일부 실시예에 관하여 설명되었지만, 당업자는 본 발명이 설명된 실시예에 제한되지 않는다는 것을 알 것이다. 본 발명의 방법과 장치는 첨부된 청구 범위와 사상내에서 수정과 변경이 가능하다. 따라서, 본 명세서는 본 발명상에 제한을 두는 대신에 예시적이다.

(57) 청구의 범위

청구항 1

데이터 처리 장치에서, 명령을 실행하는 방법에 있어서,

단일 논리 레지스터 파일로서 소프트웨어에 최소한 논리적으로 나타나는 것으로 제 1 명령 타입의 제 1 명령 세트를 실행하는 단계로서, 여기서 상기 단일 논리 레지스터 파일은 상기 제 1 명령 세트를 실행하는 동안에 플랫 레지스터 파일로서 동작되는 단계;

상기 단일 논리 레지스터 파일로서 소프트웨어에 최소한 나타나는 것으로 제 2 명령 타입의 제 1 명령을 실행하는 단계로서, 여기서 상기 단일 논리 레지스터 파일은 상기 제 1 명령을 실행하는 동안에 스택 레퍼런스 레지스터 파일로서 동작하는 단계; 및

상기 단일 논리 레지스터 파일에 대응하는 태그 세트내의 모든 태그를, 상기 제 1 명령 세트를 실행하는 상기 단계를 시작할 때와 상기 제 2 명령 타입의 상기 제 1 명령 세트를 실행하는 상기 단계를 완료할 때 사이에서 비공백 상태로 바꾸는 단계로서, 여기서 상기 태그 세트는 상기 단일 논리 레지스터 파일내의 레지스터가 공백인지 비공백인지를 식별하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 2

제 1 항에 있어서, 상기 태그 세트내의 모든 태그를 바꾸는 상기 단계는 상기 제 1 명령 세트의 제 1 명령을 실행하는 것에 대응하여 수행되는 것을 특징으로 하는 방법.

청구항 3

제 1 항에 있어서, 상기 태그 세트내의 모든 태그를 바꾸는 상기 단계는 상기 제 1 명령 실행을 시도하는 것에 대응하여 수행되는 것을 특징으로 하는 방법.

청구항 4

제 1 항에 있어서, 상기 태그 세트내의 모든 태그를 바꾸는 상기 단계는 상기 제 1 명령 세트를 실행하는 상기 단계와 상기 제 2 명령 타입의 상기 제 1 명령을 실행하는 단계사이에서 수행되는 것을 특징으로 하는 방법.

청구항 5

제 1 항에 있어서, 상기 태그 세트내의 모든 태그를 바꾸는 상기 단계는 상기 제 1 명령 세트의 제 1 명령 실행을 시도하는 것에 대응하여 수행되는 것을 특징으로 하는 방법.

청구항 6

제 1 항에 있어서,

상기 제 1 명령 세트를 실행하는 상기 단계를 시작할 때와 상기 제 2 명령 타입의 상기 제 1 명령을 실행하는 상기 단계를 완료할 때 사이에서 스택 지시의 상부를 초기값으로 바꾸는 단계로서, 여기서 상기 스택 지시의 상부는 현재의 스택 지시의 상부로서 상기 단일 논리 레지스터 파일내에서 하나의 레지스터를 식별하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 7

제 1 항에 있어서,

상기 제 1 명령 세트를 실행하는 단계동안에 기록되는 상기 단일 논리 레지스터 파일내의 각각의 레지스터의 부호 및 지수 필드에, 상기 제 1 명령 세트를 실행하는 상기 단계를 시작하는 때와 상기 제 2 명령 타입의 상기 제 1 명령을 실행하는 상기 단계를 시작하는 때사이에서 숫자가 아니거나 무한대 중 어느 하나를 지시하는 값을 기록하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 8

제 1 항에 있어서, 상기 태그 세트내의 각각의 태그는 상기 단일 논리 레지스터 파일내의 상이한 레지스터에 대응하며, 상기 대응 레지스터가 공백인지 비공백인지를 식별하는 것을 특징으로 하는 방법.

청구항 9

제 1 항에 있어서, 상기 제 2 명령 타입의 상기 제 1 명령을 실행하는 상기 단계는 상기 단일 논리 레지스터 파일에 포함된 데이터를 메모리에 복사하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 10

제 1 항에 있어서, 상기 태그 세트내의 모든 태그를 바꾸는 상기 단계는 상기 제 1 명령 세트내의 각각의 명령을 실행하는 것에 대응하여 수행되는 것을 특징으로 하는 방법.

청구항 11

제 1 항에 있어서, 상기 제 1 명령 세트를 실행하는 상기 단계는 압축 연산을 수행하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 12

제 1 항에 있어서, 상기 제 1 명령 세트를 실행하는 상기 단계는 압축 정수 연산을 수행하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 13

제 1 항에 있어서, 상기 제 1 명령 세트를 실행하는 상기 단계는 압축 부동점 연산을 수행하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 14

제 1 항에 있어서, 상기 제 2 명령 타입은 부동점 연산이 수행되게 하는 명령인 것을 특징으로 하는 방법.

청구항 15

제 1 항에 있어서, 상기 제 2 명령 타입은 스칼라 연산이 수행되게 하는 명령인 것을 특징으로 하는 방법.

청구항 16

제 1 항에 있어서, 상기 제 2 명령 타입은 압축 연산이 수행되게 하는 명령인 것을 특징으로 하는 방법.

청구항 17

데이터 처리 장치에서, 명령을 실행하는 방법에 있어서,

최소한 부분적으로 에일리어스되는 단일 논리 레지스터 파일로서 소프트웨어에 최소한 논리적으로 나타나는 것으로 압축 데이터 명령 세트와 부동점 명령 세트를 실행하는 단계로서, 여기서 상기 압축 데이터 명령 세트는 상기 부동점 명령에 앞서 실행되는 단계; 및

상기 단일 논리 레지스터 파일에 대응하는 태그 세트내의 모든 태그를, 상기 압축 데이터 명령 세트의 제 1명령 실행을 시도하는 때와 상기 부동점 명령의 제 1명령 실행을 완료하는 때 사이에서 비공백 상태로 바꾸는 단계로서, 여기서 상기 태그 세트는 상기 단일 논리 레지스터 파일내의 레지스터가 공백인지 비공백인지를 식별하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 18

제 17 항에 있어서, 실행하는 상기 단계는:

고정된 레지스터 파일로서 상기 단일 논리 레지스터 파일상에서 상기 압축 데이터 명령 세트를 실행하는 단계; 및

스택 레퍼런스 레지스터 파일로서 상기 단일 논리 레지스터 파일상에서 상기 부동점 명령 세트를 실행하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 19

제 17 항에 있어서, 실행하는 상기 단계는:

압축 정수 연산을 수행하기 위해 상기 압축 데이터 명령 세트를 실행하는 단계; 및

스칼라 부동점 연산을 수행하기 위해 상기 부동점 명령세트를 실행하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 20

제 17 항에 있어서, 실행하는 상기 단계는:

압축 부동점 연산을 수행하기 위해 상기 압축 데이터 명령세트를 실행하는 단계; 및

스칼라 부동점 연산을 수행하기 위해 상기 부동점 명령세트를 실행하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 21

데이터 처리 장치에서, 스칼라 및 압축 데이터 명령을 실행할 때 부분 콘텍트 스위칭을 구현하는 방법에 있어서,

상기 스칼라 또는 상기 압축 데이터 명령중 어느 하나인 제 1 루틴에 속하는 명령을 수신하는 단계;

상기 스칼라 및 압축 데이터 명령을 모두 실행하는 단일 논리 레지스터 파일로서 소프트웨어에 최소한 논리적으로 나타나는 것이 부분 콘텍트 스위치로 인해 이용불가능한지를 판단하는 단계; 및

상기 단일 논리 레지스터 파일이 이용불가능하다면,

상기 제 1 루틴의 실행을 중단시키는 단계; 및

상기 단일 논리 레지스터 파일의 내용을 메모리에 복사하기 위해 제 2 루 틴을 실행하는 단계를 수행하는 단계;를 수행하는 단계;

그 반대면, 상기 단일 논리 레지스터 파일상에서 상기 명령을 실행하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 22

제 21 항에 있어서,

스칼라 대리 실행 지시가 대리 실행 상태에 있는지를 판단하는 단계;

상기 스칼라 대리 실행 지시가 상기 대리 실행 상태에 있다면,

상기 제 1 루틴의 실행을 중단시키는 단계;

상기 명령이 상기 스칼라 명령중 하나라면, 상기 제 2 루틴을 실행하는 단계; 및

그 반대면, 상기 명령은 상기 압축 데이터 명령중 하나이고, 상기 제 2 루틴보다 오히려 제3루틴을 실행하는 단계를 수행하는 더 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 23

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 압축 데이터 명령중 하나이면,

상기 명령이 압축 데이터 항목을 상기 단일 논리 레지스터 파일에 기록하게 하는지를 판단하는 단계;

상기 명령이 상기 압축 데이터 항목을 상기 단일 논리 레지스터 파일에 기록하게 한다면,

상기 단일 논리 레지스터 파일내의 레지스터의 소수부 필드에 상기 압축 데이터 항목을 기록하는 단계; 및

숫자가 아닌 또는 무한대를 나타내는 값을 상기 레지스터의 부호 필드와 지수 필드에 기록하는 단계;를 수행하는 단계;를 수행하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 24

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 압축 데이터 명령중 이행 명령이면, 태그 세트내의 각각의 태그를 공백 상태로 바꾸는 단계로서, 여기서 상기 태그 세트의 각각의 태그는 상기 단일 논리 레지스터 파일내의 상이한 레지스터에 대응하고 상기 대응 레지스터가 공백인지 비공백인지를 식별하는 단계; 및

상기 명령이 상기 이행 명령이 아니라 상기 압축 데이터 명령중 하나이면, 상기 태그 세트를 비공백 상태로 바꾸는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 25

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 스칼라 명령중 하나이면,

상기 압축 데이터 명령중 이행 명령이 상기 스칼라 명령중 하나보다 최근에, 그리고 상기 압축 데이터 명령중 기타 다른 하나보다 최근에 실행되었다면, 태그 세트를 비공백 상태로 바꾸는 단계로서, 여기서 상기 태그 세트내의 각각의 태그는 상기 단일 논리 레지스터 파일내의 상이한 레지스터에 대응하고 상기 대응 레지스터가 공백인지 비공백인지를 식별하는 단계; 및

상기 압축 데이터 명령중 하나가 상기 스칼라 명령중 하나보다 최근에, 그리고 상기 이행 명령보다 최근에 실행되었다면 상기 태그 세트를 공백 상태로 바꾸는 단계;를 수행하는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 26

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 압축 데이터 명령중 하나이면, 상기 스칼라 명령중 하나가 상기 압축 데이터 명령중 하나보다 최근에 실행되었으면 태그 세트를 비공백 상태로 바꾸는 단계로서, 여기서 상기 태그 세트내의 각각의 태그는 상기 단일 논리 레지스터 파일내의 상이한 레지스터에 대응하고 상기 대응 레지스터는 공백인지 비공백인지를 식별하는 단계; 및

상기 명령이 상기 압축 데이터 명령중 이행 명령이면, 상기 태그 세트를 공백 상태로 바꾸는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 27

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 압축 데이터 명령중 하나이면, 스택 지시의 상부를 초기값으로 바꾸는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 28

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 스칼라 명령중 하나이면, 상기 압축 데이터 명령중 하나가 상기 스칼라 명령중 하나보다 최근에 실행되었으면, 스택 지시의 상부를 초기값으로 바꾸는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 29

제 21 항에 있어서, 상기 명령을 실행하는 상기 단계는:

상기 명령이 상기 압축 데이터 명령중 하나이면, 상기 스칼라 명령중 하나가 상기 압축 데이터 명령중 하나보다 최근에 실행되었으면 스택 지시의 상부를 초기값으로 바꾸는 단계를 더 포함하고 있는 것을 특징으로 하는 방법.

청구항 30

제 21 항에 있어서, 상기 스칼라 명령은 스칼라 부동점 연산을 수행하기 위한 것이고, 상기 압축 데이터 명령은 압축 정수 연산을 수행하기 위한 것인 것을 특징으로 하는 방법.

청구항 31

제 21 항에 있어서, 상기 스칼라 명령은 스칼라 부동점 연산을 수행하기 위한 것이고, 상기 압축 데이터 명령은 압축 부동점 연산을 수행하기 위한 것인 것을 특징으로 하는 방법.

청구항 32

데이터 처리 장치에서, 스칼라 및 압축 데이터 명령을 실행하는 방법에 있어서,

압축 데이터 명령이나 스칼라 명령중 하나인 명령을 수신하는 단계;

스칼라 명령의 실행을 지시하는 지시가 대리 실행되어야 하는지 및/또는 상기 스칼라 및 압축 데이터 명령 모두를 실행하는 단일 논리 레지스터 파일로서 소프트웨어에 최소한 논리적으로 나타나는 것이 부분 콘텍트 스위치로 인하여 이용불가능한지를 판단하는 단계;

상기 명령이 상기 스칼라 명령이면, 상기 단일 논리 레지스터 파일중 어느 하나가 이용불가능하거나 스칼라 명령의 실행이 대리 실행될 수 있다면 제 1 루틴을 실행하는 단계; 및

그 반대면, 상기 명령은 상기 압축 데이터 명령이며, 상기 단일 논리 레지스터 파일이 이용불가능하면 상기 제 1 루틴을 실행하지만, 스칼라 명령의 실행이 대리 실행될 수 있으면 상기 제 1 루틴 대신에 제 2 루틴을 실행하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

청구항 33

제 32 항에 있어서, 상기 스칼라 명령은 스칼라 부동점 연산을 수행하기 위한 것이고, 상기 압축 데이터 명령은 압축 정수 연산을 수행하기 위한 것인 것을 특징으로 하는 방법.

청구항 34

제 32 항에 있어서, 상기 스칼라 명령은 스칼라 부동점 연산을 수행하기 위한 것이고, 상기 압축 데이터 명령은 압축 부동점 연산을 수행하기 위한 것인 것을 특징으로 하는 방법.

청구항 35

데이터 처리 장치에서, 압축 데이터 명령을 실행하는 방법에 있어서,

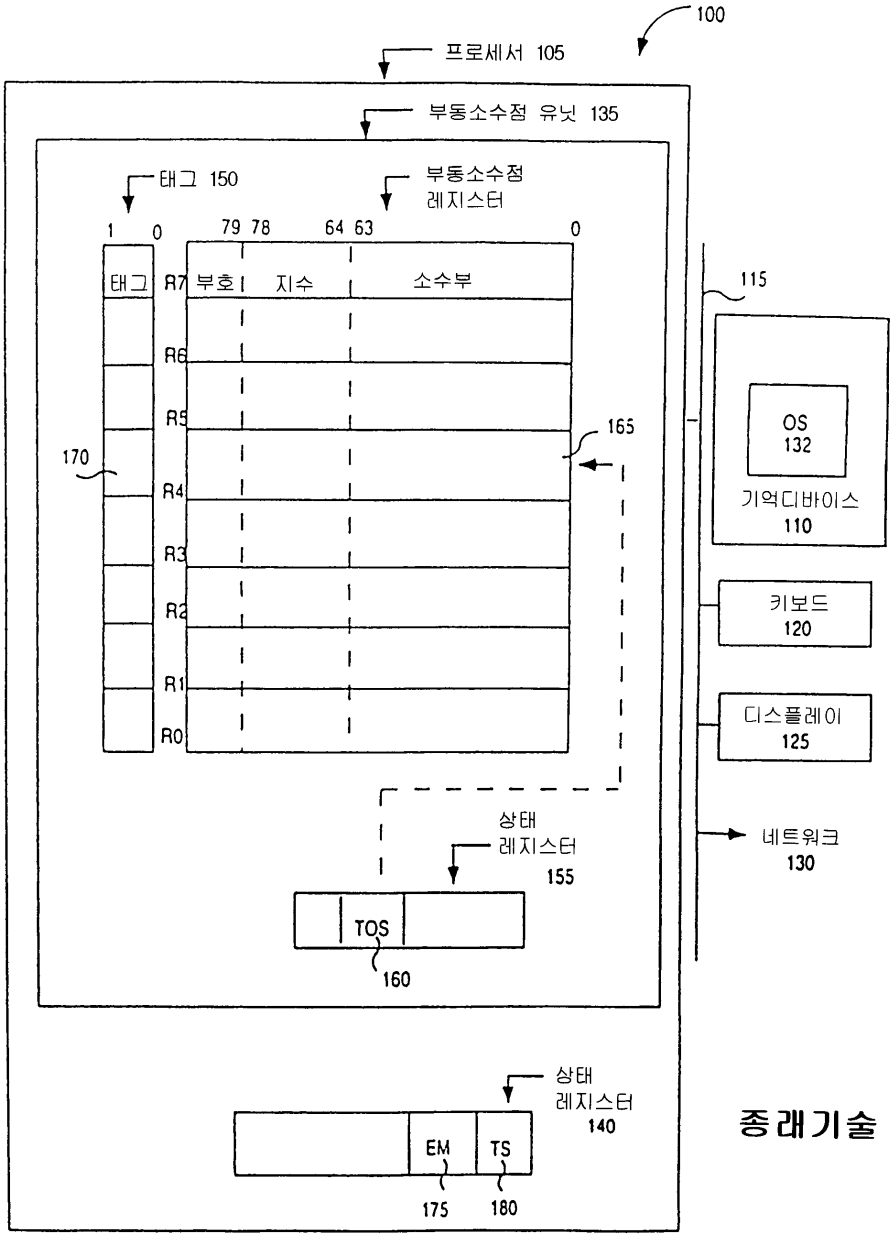
부동점 데이터를 저장하는 데 또한 사용되는 논리 레지스터 파일내의 레지스터로서 소프트웨어에 최소한 논리적으로 나타나는 것에 압축 데이터 항목이 기록되게 하는 압축 데이터 명령을 수신하는 단계;

상기 논리 레지스터의 소수부 필드에 상기 압축 데이터 항목을 기록하는 단계; 및

숫자가 아닌 또는 무한대를 나타내는 값을 상기 논리 레지스터의 부호 필드와 지수 필드에 기록하는 단계를 포함하고 있는 것을 특징으로 하는 방법.

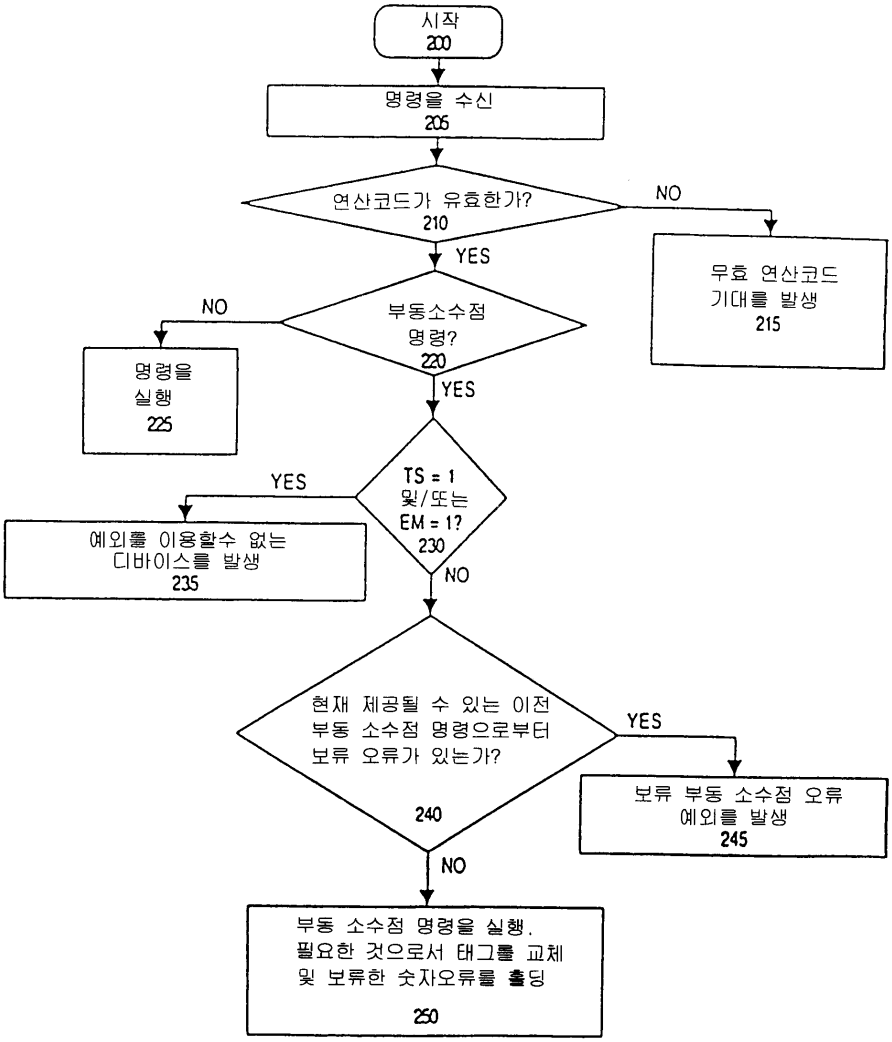
도면

도면1


100
프로세서 105
부동소수점 유닛 135
태그 150
부동소수점
레지스터
1 0
79 78
64 63
0
태그
R7
부호
지수
소수부
R6
R5
R4
R3
R2
R1
R0
170
165
115
OS
132
기억디바이스
110
키보드
120
디스플레이
125
네트워크
130
상태
레지스터
155
TOS
160
상태
레지스터
140
EM
TS
175
180

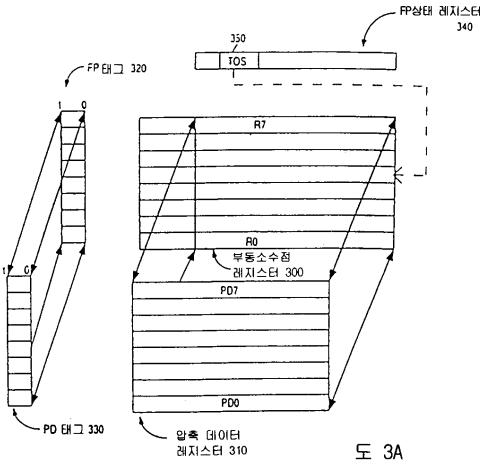
종래기술

도면2

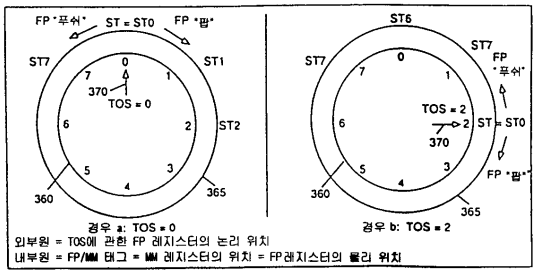

시작
200
명령을 수신
205
연산코드가 유효한가?
210
NO
무효 연산코드
기대를 발생
215
YES
부동소수점
명령?
220
NO
명령을
실행
225
YES
TS = 1
및/또는
EM = 1?
230
YES
예외를 이용할수 없는
디바이스를 발생
235
NO
현재 제공될 수 있는 이전
부동 소수점 명령으로부터
보류 오류가 있는가?
240
YES
보류 부동 소수점 오류
예외를 발생
245
NO
부동 소수점 명령을 실행.
필요한 것으로서 태그를 교체
및 보류한 숫자오류를 홀딩
250

종래기술

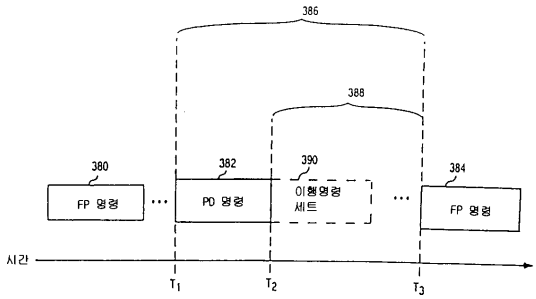
도면3



도 3A

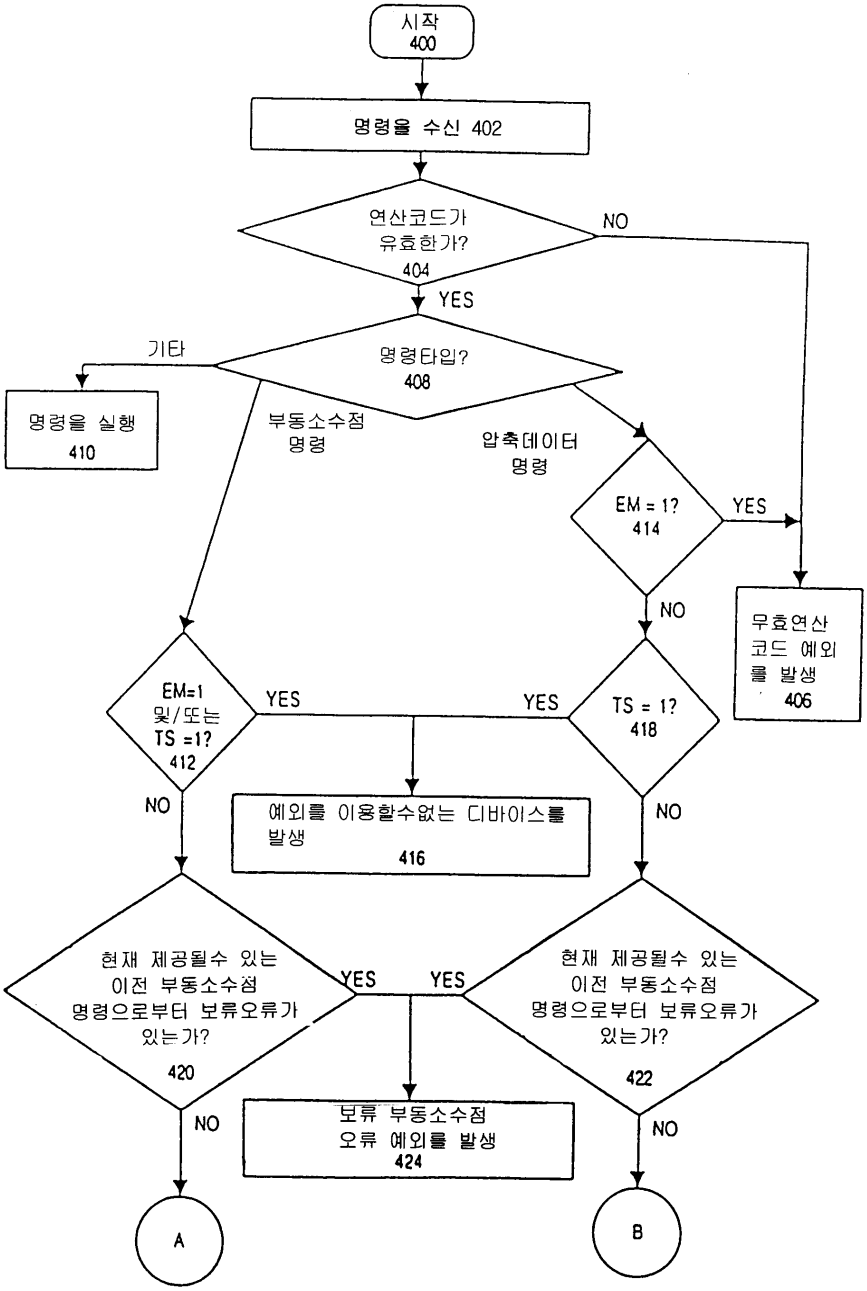


도 3B 도 3C

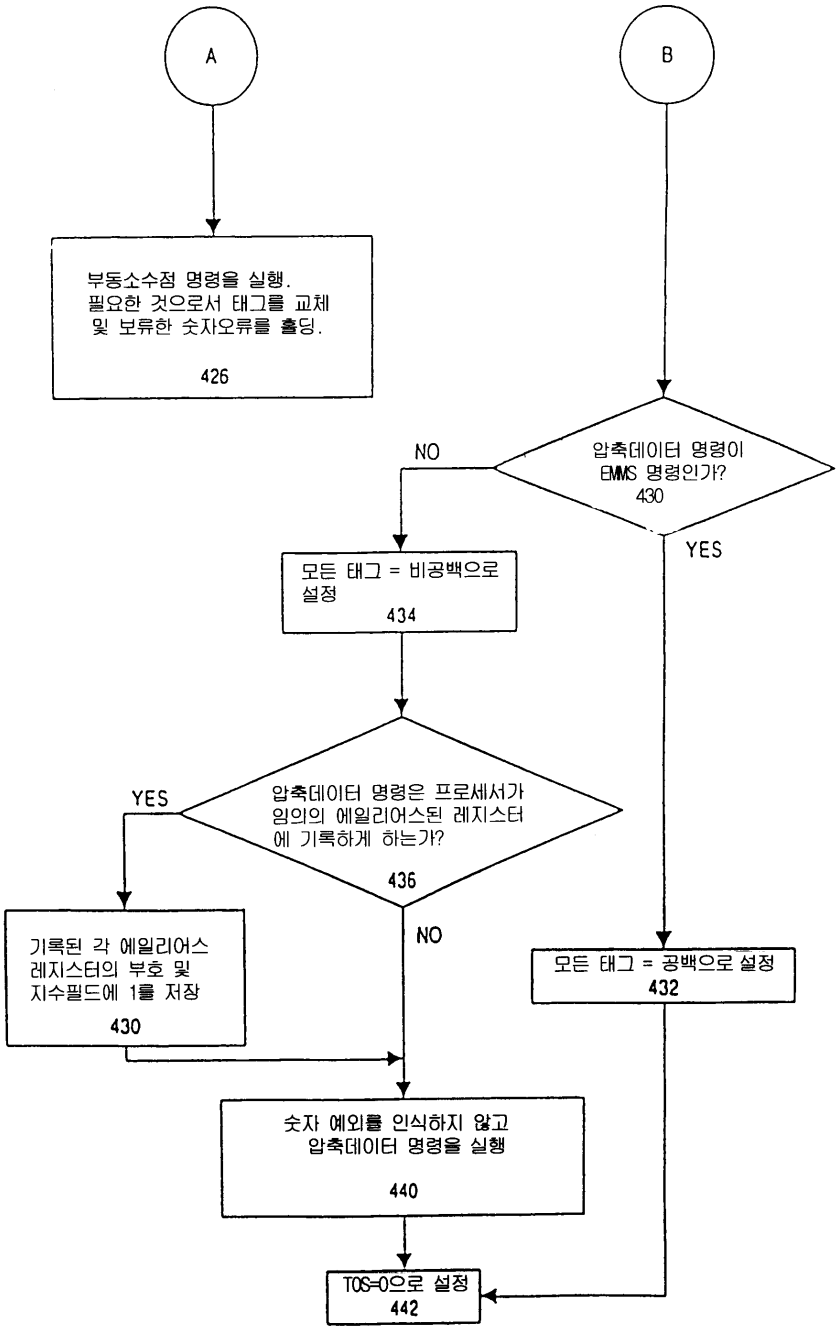


도 3D

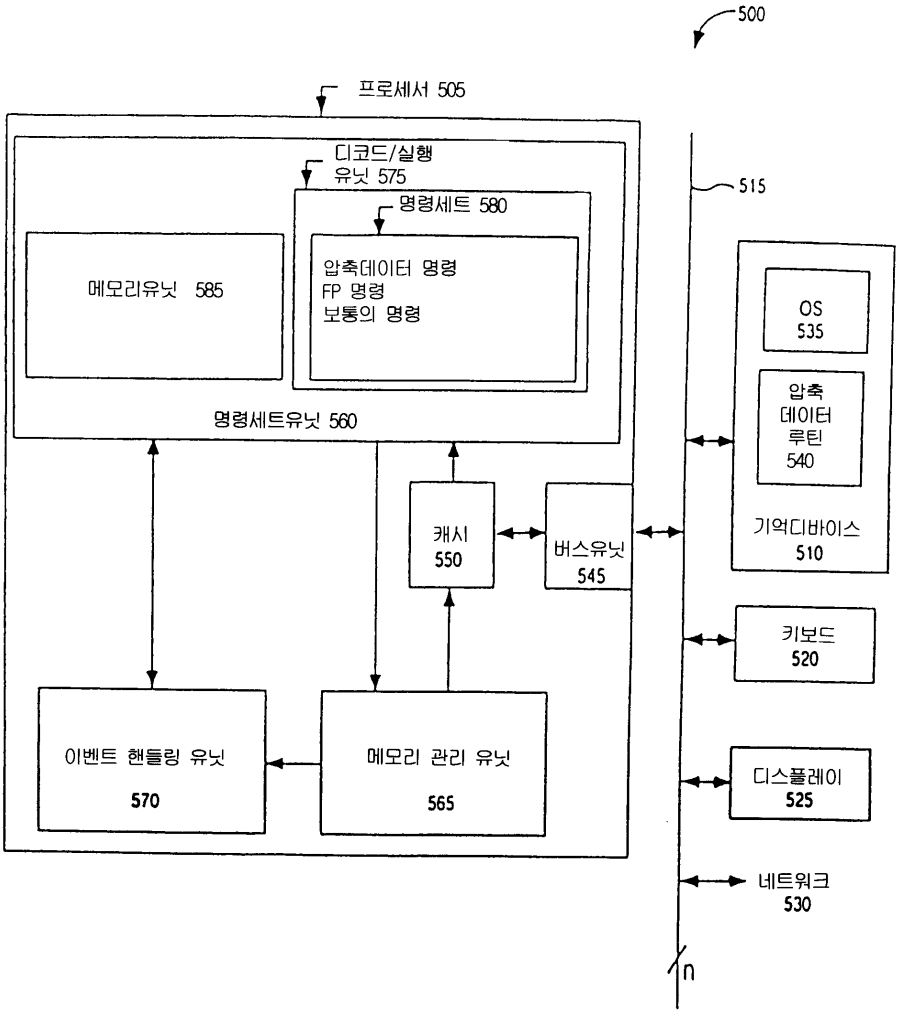
도면4A


시작
400
명령을 수신 402
연산코드가
유효한가?
404
NO
YES
기타
명령타입?
408
명령을 실행
410
부동소수점
명령
압축데이터
명령
EM = 1?
414
YES
NO
무효연산
코드 예외
를 발생
406
EM=1
및/또는
TS =1?
412
YES
YES
TS = 1?
418
NO
NO
예외를 이용할수없는 디바이스를
발생
416
현재 제공될수 있는
이전 부동소수점
명령으로부터 보류오류가
있는가?
420
YES
YES
현재 제공될수 있는
이전 부동소수점
명령으로부터 보류오류가
있는가?
422
NO
NO
보류 부동소수점
오류 예외를 발생
424
A
B

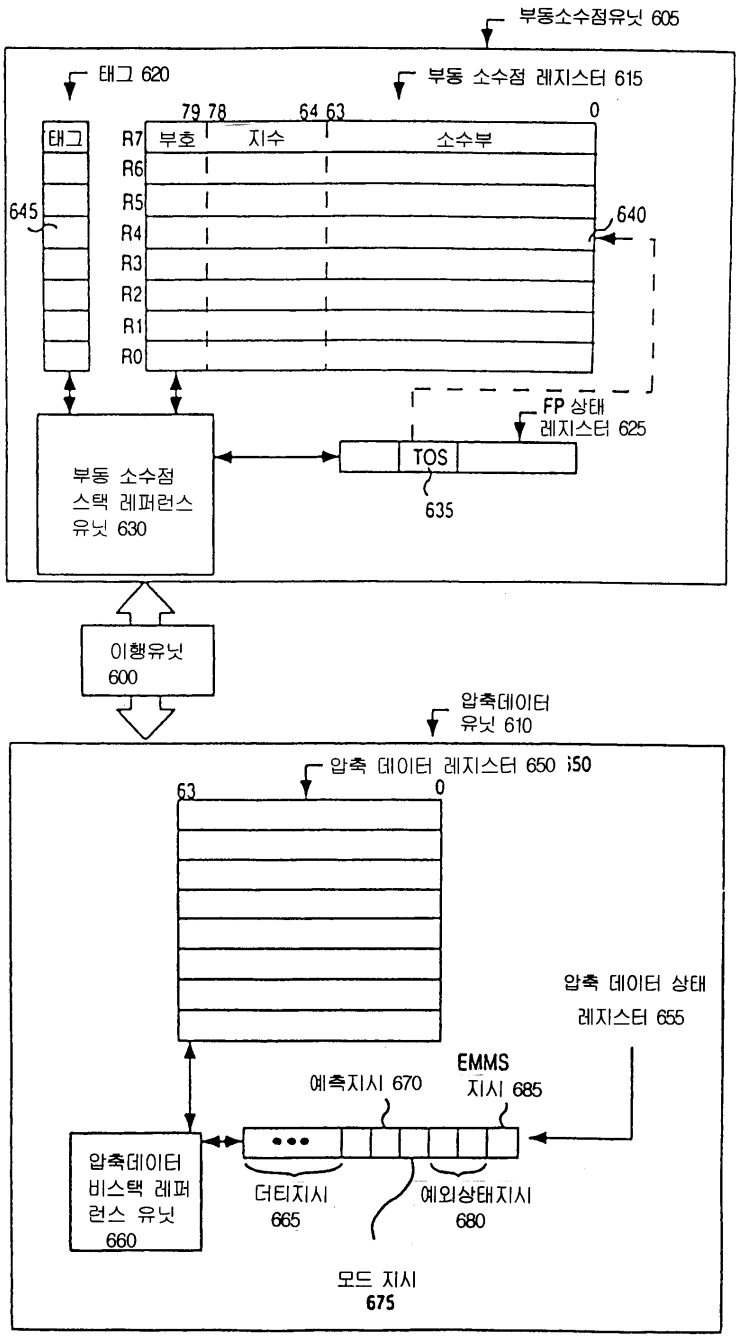
도면4B


A
B
부동소수점 명령을 실행.
필요한 것으로서 태그를 교체
및 보류한 숫자오류를 홀딩.
426
압축데이터 명령이
EMMS 명령인가?
430
NO
YES
모든 태그 = 비공백으로
설정
434
압축데이터 명령은 프로세서가
임의의 에일리어스된 레지스터
에 기록하게 하는가?
436
YES
NO
기록된 각 에일리어스
레지스터의 부호 및
지수필드에 1를 저장
430
모든 태그 = 공백으로 설정
432
숫자 예외를 인식하지 않고
압축데이터 명령을 실행
440
TOS=0으로 설정
442

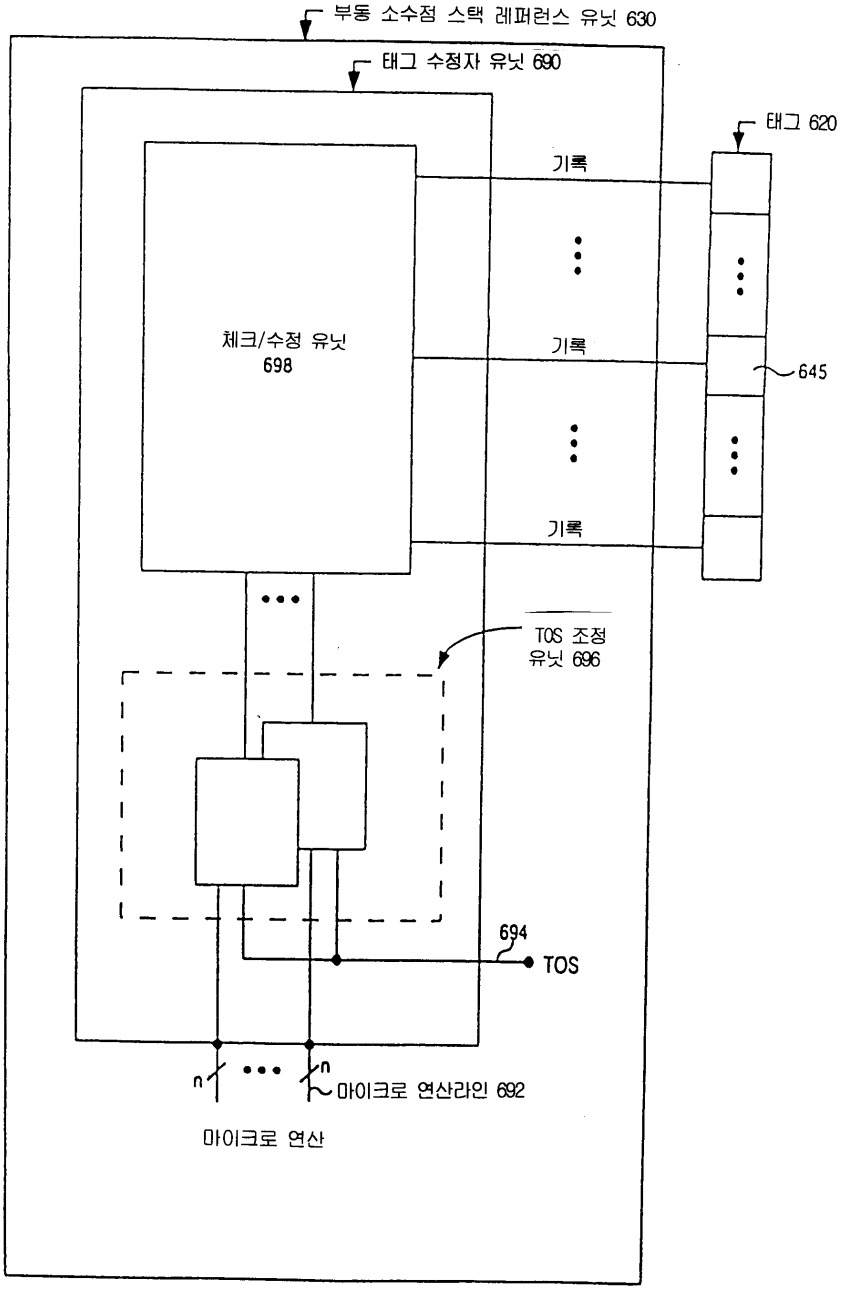
도면5


500
프로세서 505
디코드/실행
유닛 575
명령세트 580
메모리유닛 585
압축데이터 명령
FP 명령
보통의 명령
명령세트유닛 560
515
OS
535
압축
데이터
루틴
540
기억디바이스
510
캐시
550
버스유닛
545
키보드
520
이벤트 핸들링 유닛
570
메모리 관리 유닛
565
디스플레이
525
네트워크
530
n

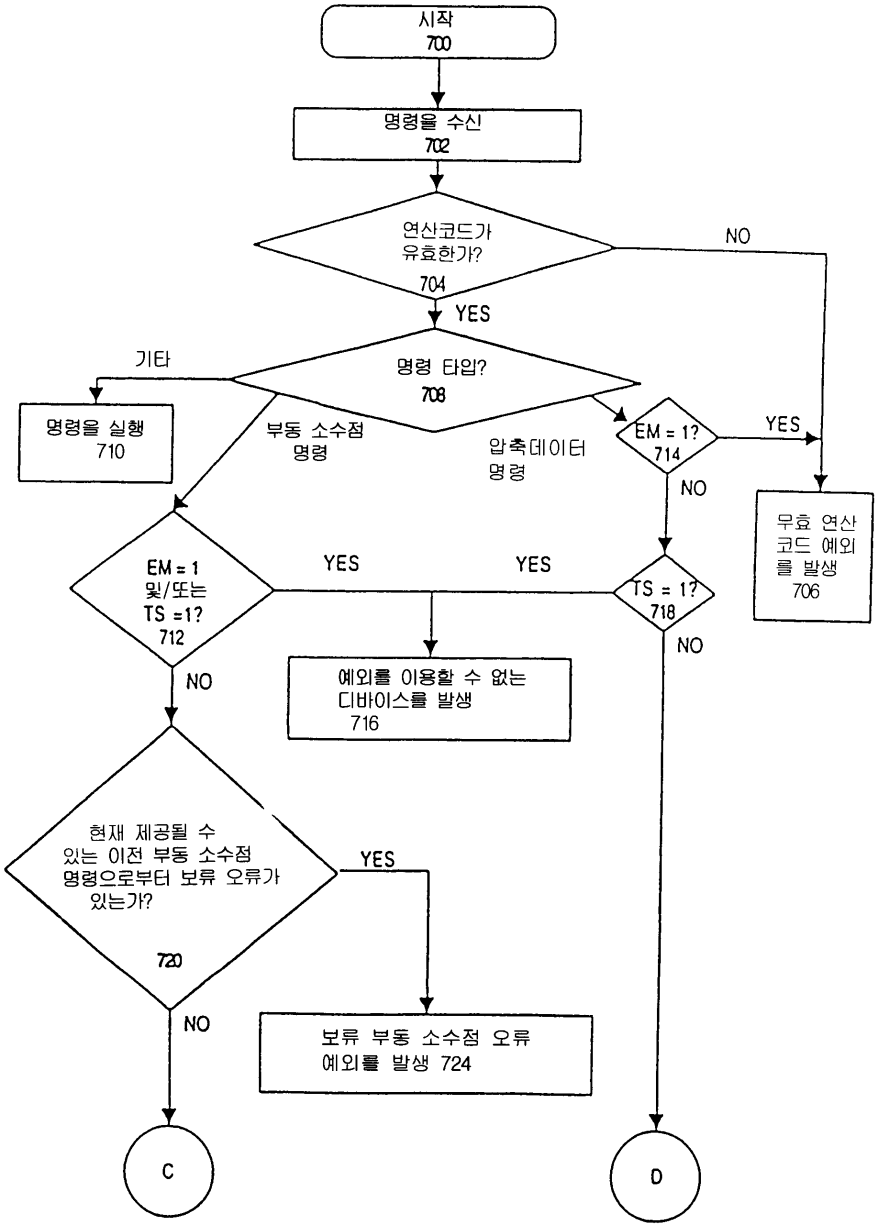
도면6A


부동소수점유닛 605
태그 620
부동 소수점 레지스터 615
79 78
64 63
0
태그
R7
R6
R5
R4
R3
R2
R1
R0
부호
지수
소수부
645
640
FP 상태
레지스터 625
TOS
635
부동 소수점
스택 레퍼런스
유닛 630
이행유닛
600
압축데이터
유닛 610
압축 데이터 레지스터 650 i50
63
0
압축 데이터 상태
레지스터 655
EMMS
지시 685
예측지시 670
압축데이터
비스택 레퍼
런스 유닛
660
더티지시
665
예외상태지시
680
모드 지시
675

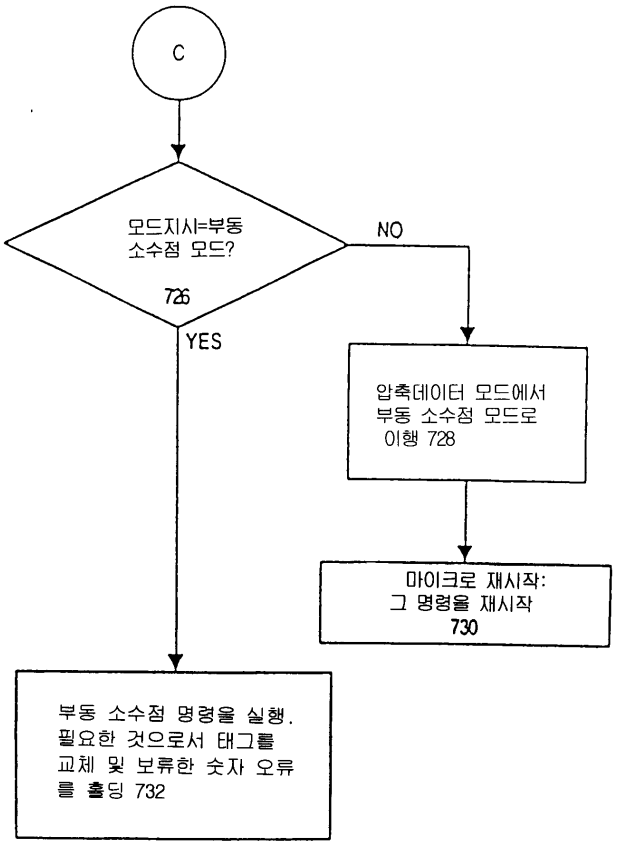
도면6B


부동 소수점 스택 레퍼런스 유닛 630
태그 수정자 유닛 690
태그 620
기록
기록
기록
645
체크/수정 유닛
698
TOS 조정
유닛 696
694
TOS
n
n
마이크로 연산라인 692
마이크로 연산

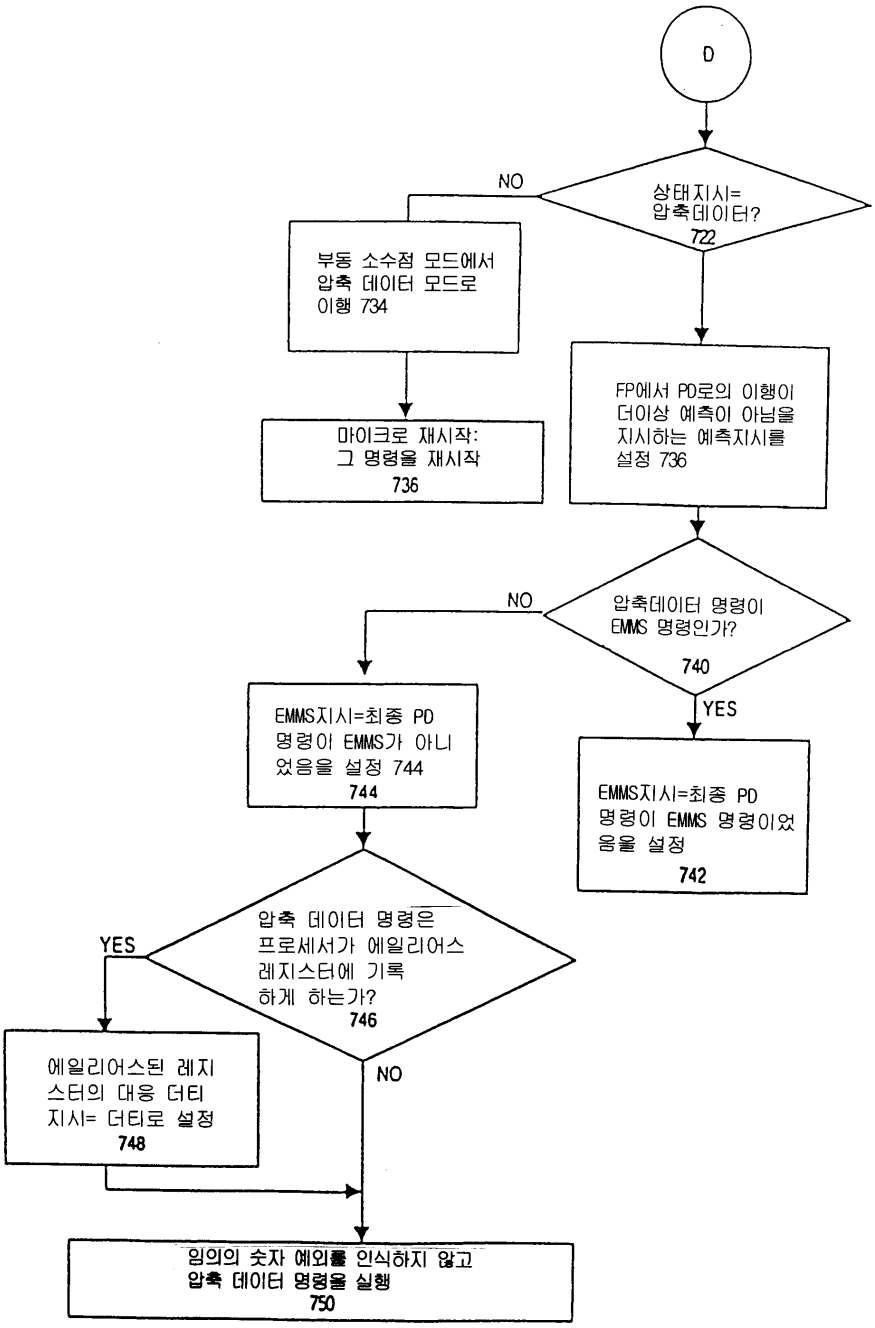
도면7A


시작
700
명령을 수신
702
연산코드가
유효한가?
704
NO
YES
명령 타입?
708
기타
명령을 실행
710
부동 소수점
명령
압축데이터
명령
EM = 1?
714
YES
NO
무효 연산
코드 예외
를 발생
706
EM = 1
및/또는
TS =1?
712
YES
YES
TS = 1?
718
NO
예외를 이용할 수 없는
디바이스를 발생
716
NO
현재 제공될 수
있는 이전 부동 소수점
명령으로부터 보류 오류가
있는가?
720
YES
NO
보류 부동 소수점 오류
예외를 발생 724
C
D

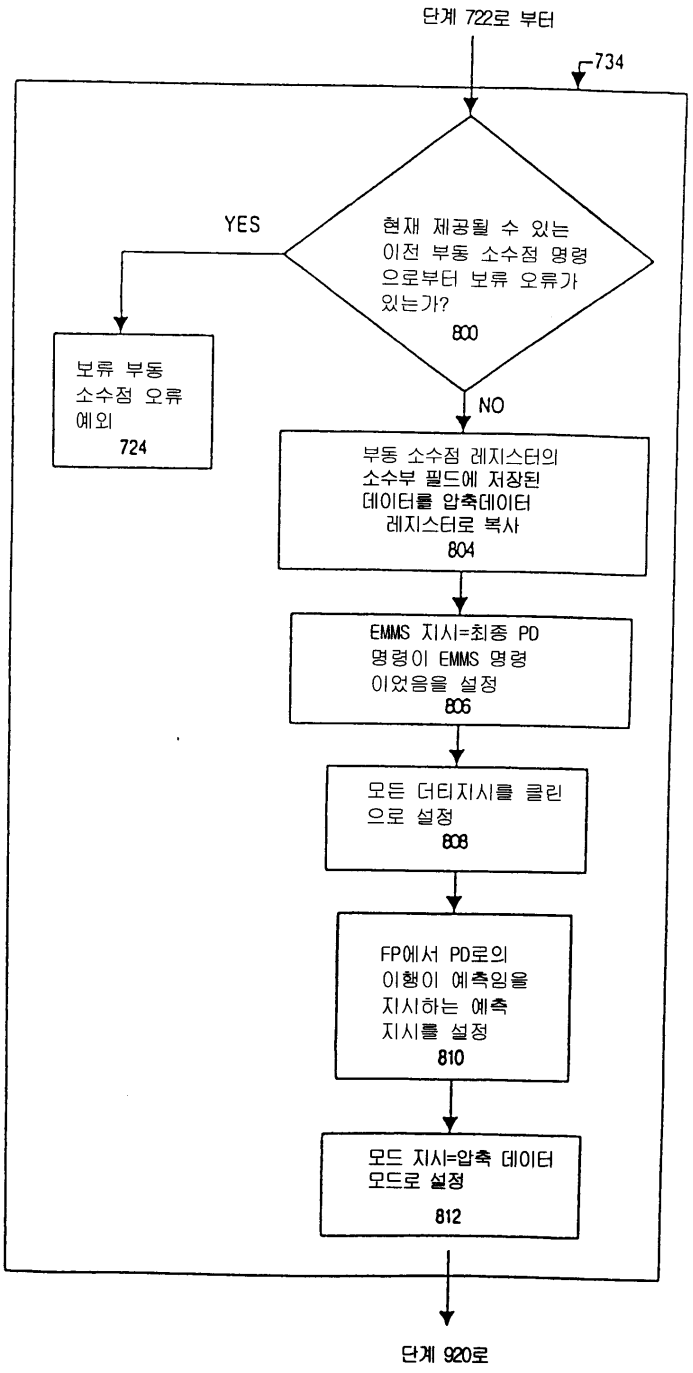
도면78


C
모드지시=부동
소수점 모드?
726
NO
YES
압축데이터 모드에서
부동 소수점 모드로
이행 728
마이크로 재시작:
그 명령을 재시작
730
부동 소수점 명령을 실행.
필요한 것으로서 태그를
교체 및 보류한 숫자 오류
를 홀딩 732

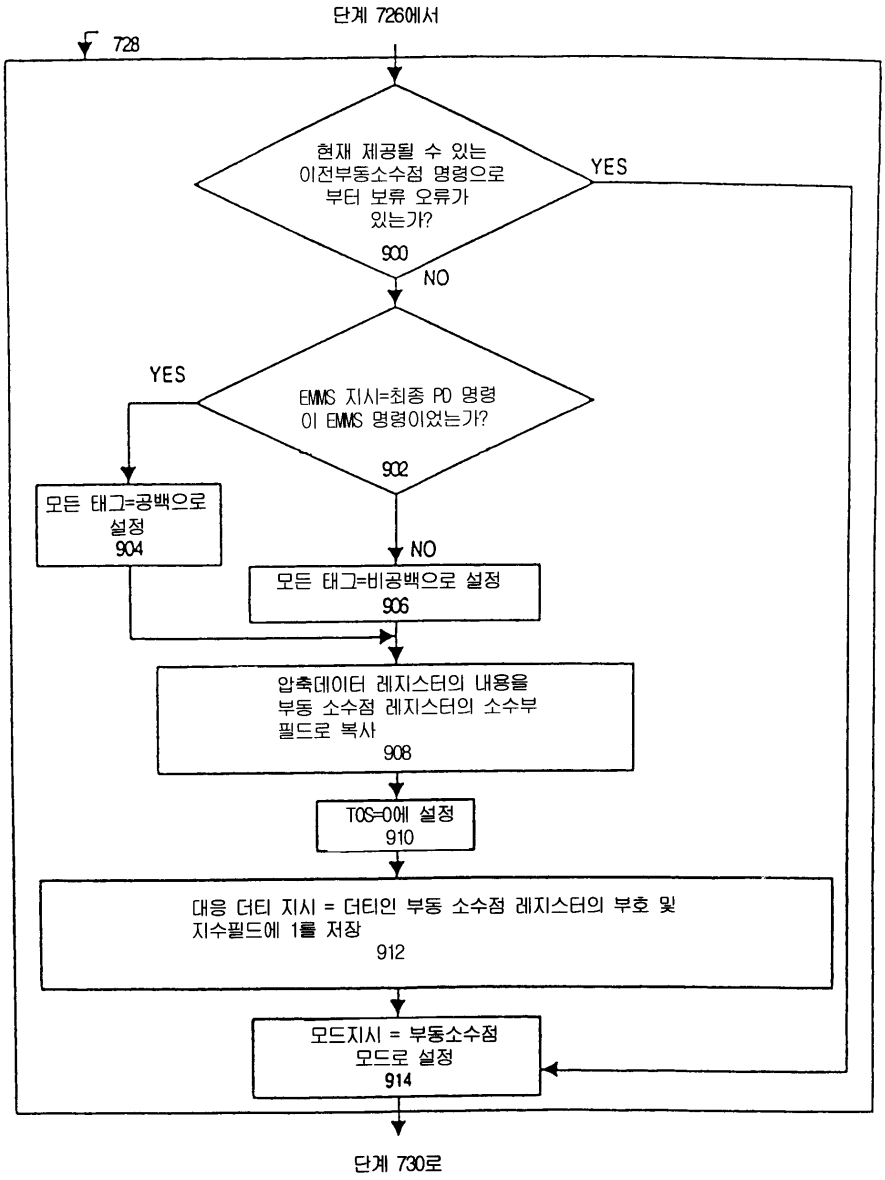
도면7C


D
상태지시=
압축데이터?
722
NO
부동 소수점 모드에서
압축 데이터 모드로
이행 734
마이크로 재시작:
그 명령을 재시작
736
FP에서 PD로의 이행이
더이상 예측이 아님을
지시하는 예측지시를
설정 736
압축데이터 명령이
EMMS 명령인가?
740
NO
YES
EMMS지시=최종 PD
명령이 EMMS가 아니
었음을 설정 744
744
EMMS지시=최종 PD
명령이 EMMS 명령이었
음을 설정
742
압축 데이터 명령은
프로세서가 에일리어스
레지스터에 기록
하게 하는가?
746
YES
NO
에일리어스된 레지
스터의 대응 더티
지시= 더티로 설정
748
임의의 숫자 예외를 인식하지 않고
압축 데이터 명령을 실행
750

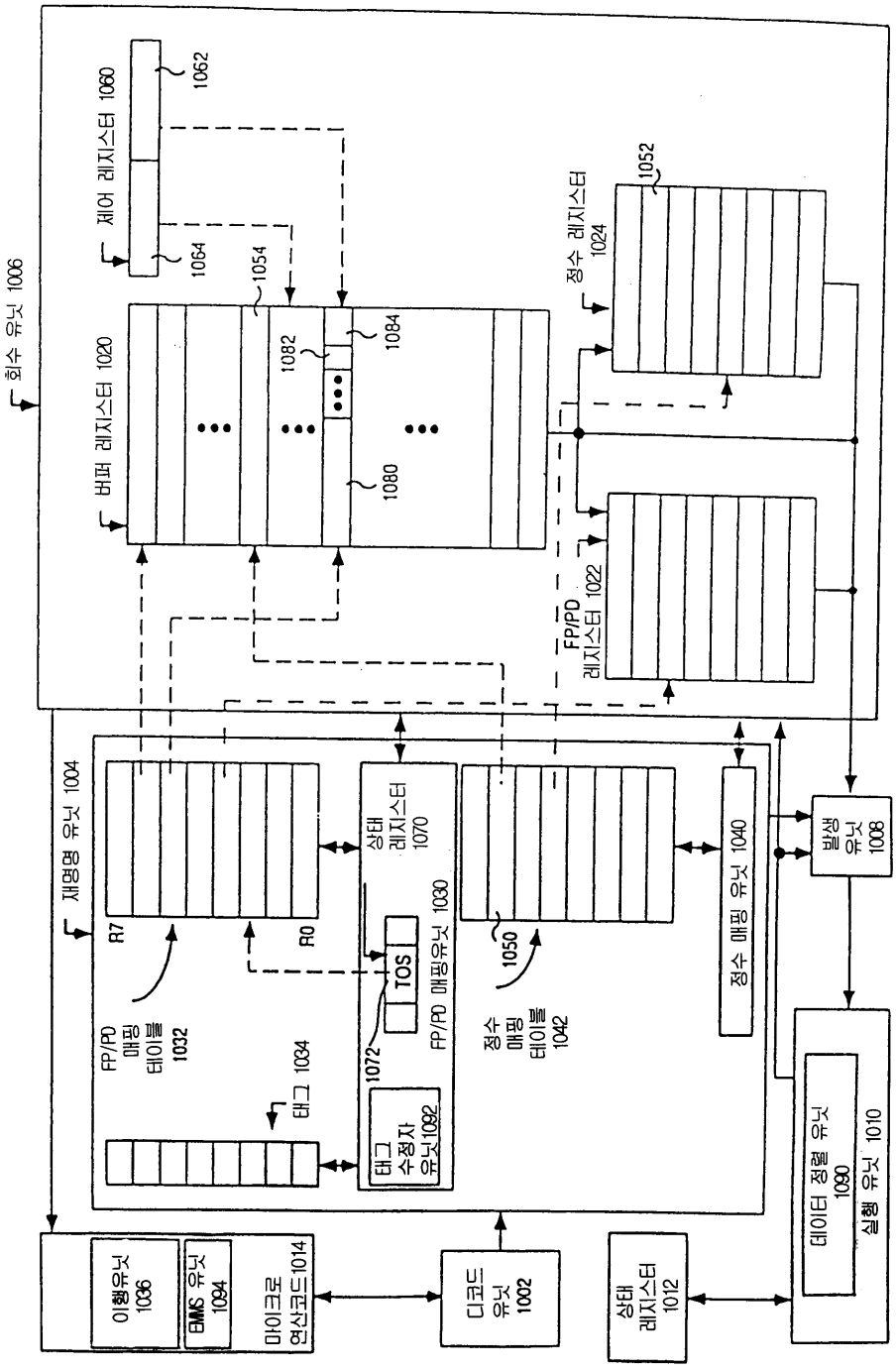
도면8


단계 722로 부터
734
현재 제공될 수 있는 이전 부동 소수점 명령으로부터 보류 오류가 있는가?
800
YES
NO
보류 부동 소수점 오류 예외
724
부동 소수점 레지스터의 소수부 필드에 저장된 데이터를 압축데이터 레지스터로 복사
804
EMMS 지시=최종 PD 명령이 EMMS 명령이었음을 설정
806
모든 더티지시를 클린으로 설정
808
FP에서 PD로의 이행이 예측임을 지시하는 예측 지시를 설정
810
모드 지시=압축 데이터 모드로 설정
812
단계 920로

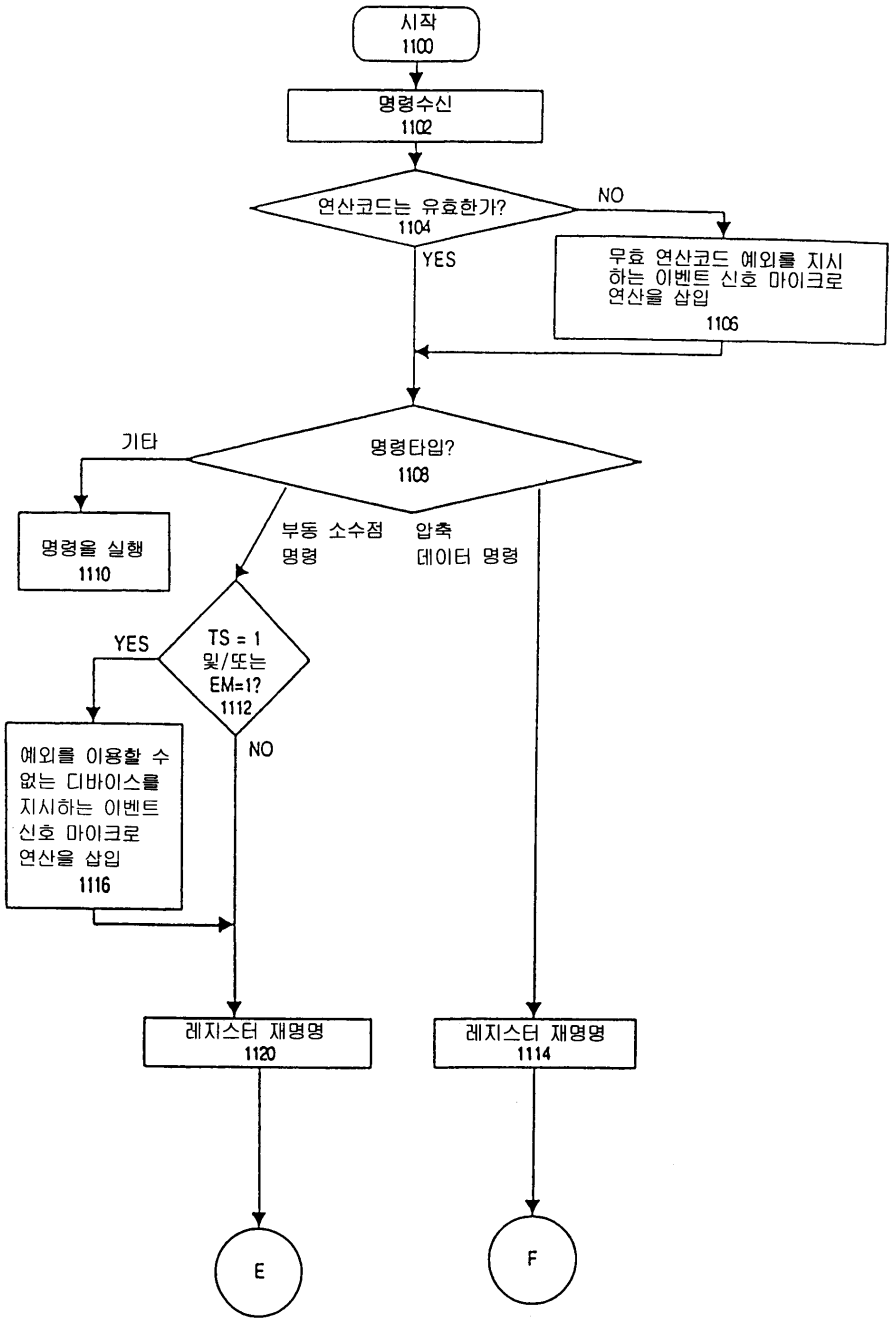
도면9


단계 726에서
728
현재 제공될 수 있는
이전부동소수점 명령으로
부터 보류 오류가
있는가?
900
YES
NO
EMMS 지시=최종 PD 명령
이 EMMS 명령이었는가?
902
YES
NO
모든 태그=공백으로
설정
904
모든 태그=비공백으로 설정
906
압축데이터 레지스터의 내용을
부동 소수점 레지스터의 소수부
필드로 복사
908
TOS=0에 설정
910
대응 데티 지시 = 데티인 부동 소수점 레지스터의 부호 및
지수필드에 1를 저장
912
모드지시 = 부동소수점
모드로 설정
914
단계 730로

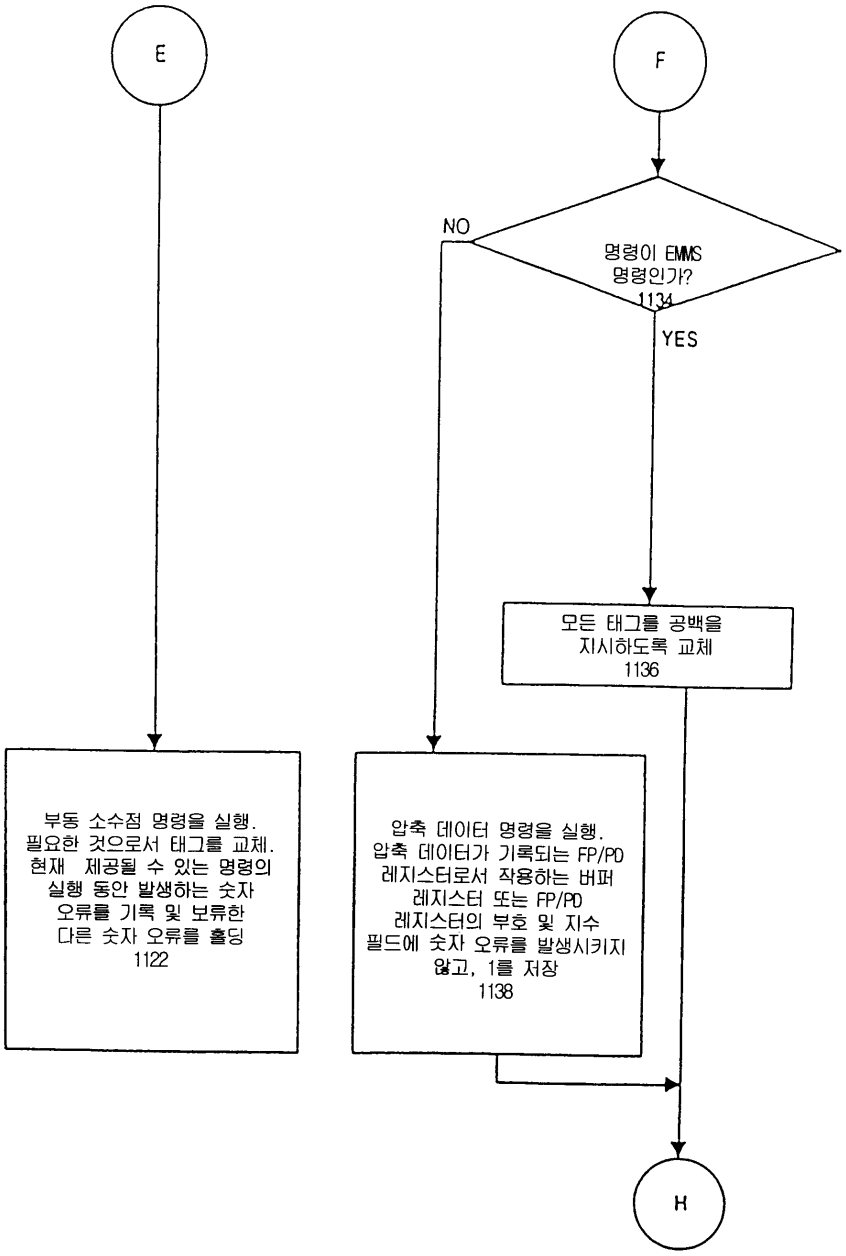
도면10


회수 유닛 1006
제어 레지스터 1060
1062
1064
1054
버퍼 레지스터 1020
1082
1084
1080
정수 레지스터 1024
1052
FP/PD 레지스터 1022
재명명 유닛 1004
R7
R0
FP/PD 매핑 테이블 1032
태그 1034
상태 레지스터 1070
TOS
1072
FP/PD 매핑유닛 1030
태그 수정자 유닛1092
1050
정수 매핑 테이블 1042
정수 매핑 유닛 1040
발생 유닛 1008
데이터 정렬 유닛 1090
실행 유닛 1010
이행유닛 1036
BMS 유닛 1094
마이크로 연산코드1014
디코드 유닛 1002
상태 레지스터 1012

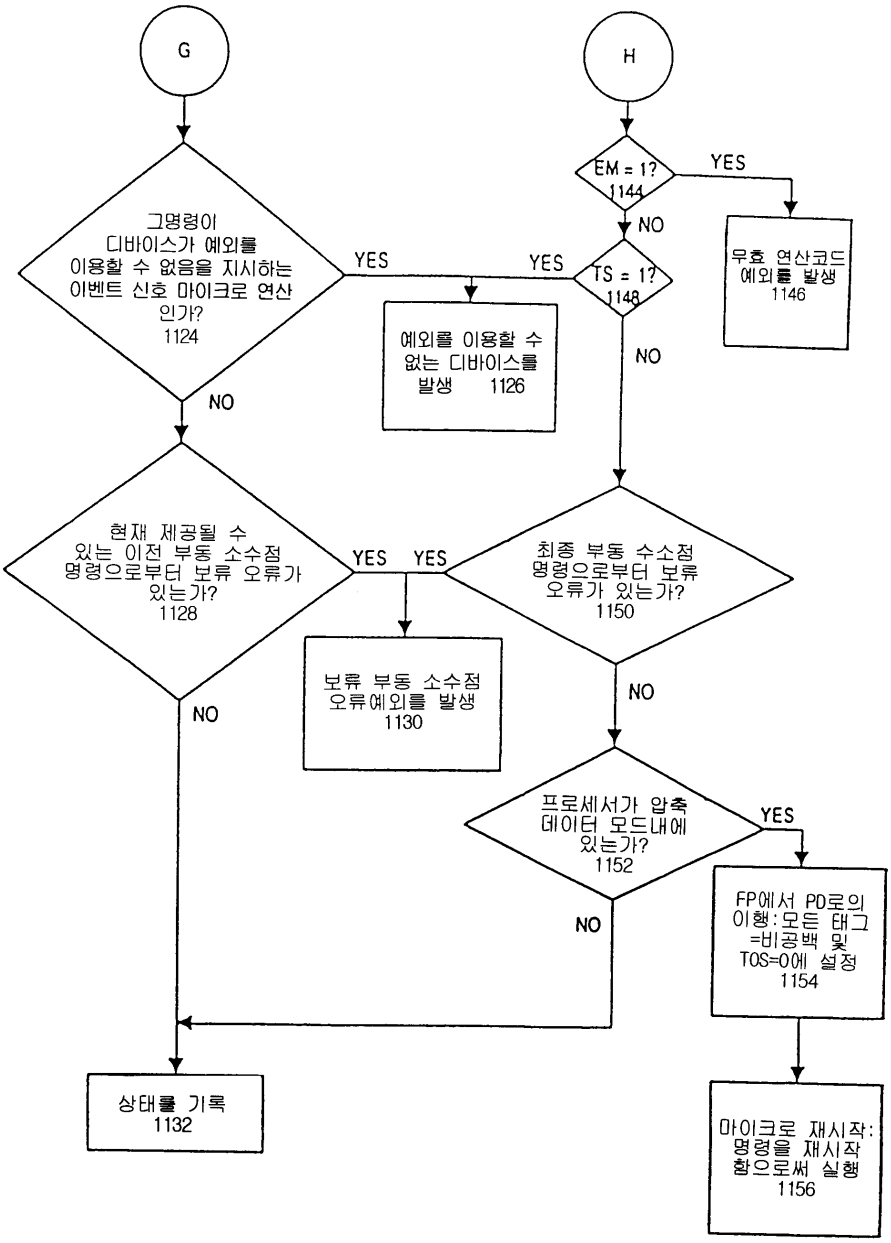
도면11A


시작
1100
명령수신
1102
연산코드는 유효한가?
1104
NO
YES
무효 연산코드 예외를 지시하는 이벤트 신호 마이크로 연산을 삽입
1106
명령타입?
1108
기타
명령을 실행
1110
부동 소수점 명령
압축 데이터 명령
TS = 1 및/또는 EM=1?
1112
YES
NO
예외를 이용할 수 없는 디바이스를 지시하는 이벤트 신호 마이크로 연산을 삽입
1116
레지스터 재명명
1120
레지스터 재명명
1114
E
F

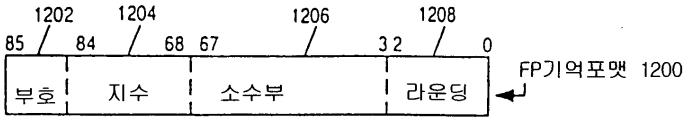
도면11B


E
F
NO
명령이 EMMS
명령인가?
1134
YES
모든 태그를 공백을
지시하도록 교체
1136
부동 소수점 명령을 실행.
필요한 것으로서 태그를 교체.
현재 제공될 수 있는 명령의
실행 동안 발생하는 숫자
오류를 기록 및 보류한
다른 숫자 오류를 홀딩
1122
압축 데이터 명령을 실행.
압축 데이터가 기록되는 FP/PD
레지스터로서 작용하는 버퍼
레지스터 또는 FP/PD
레지스터의 부호 및 지수
필드에 숫자 오류를 발생시키지
않고, 1를 저장
1138
H

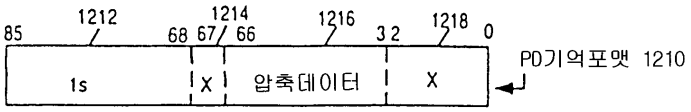
도면11C



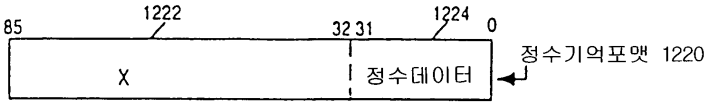
도면12A



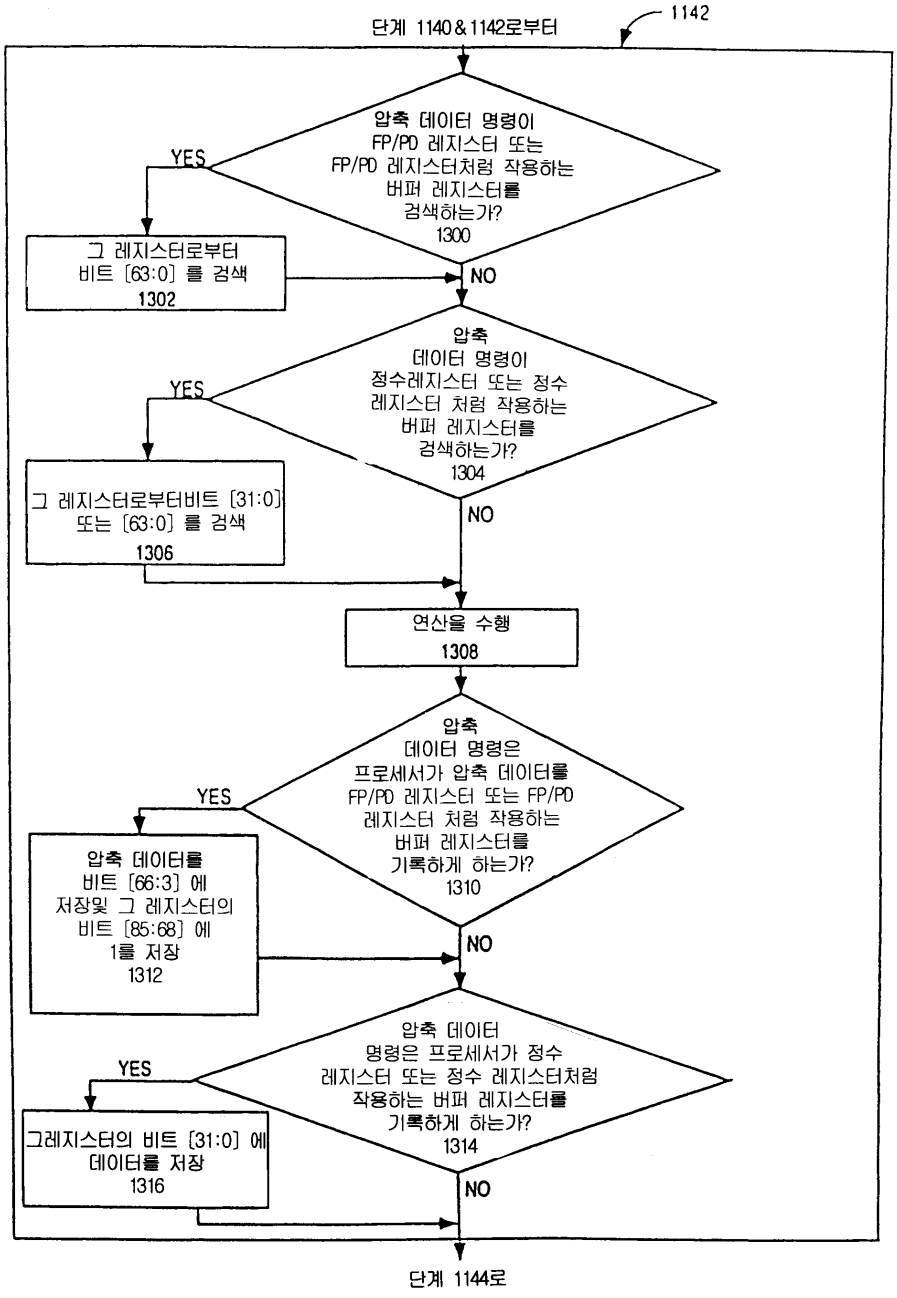
도면12B



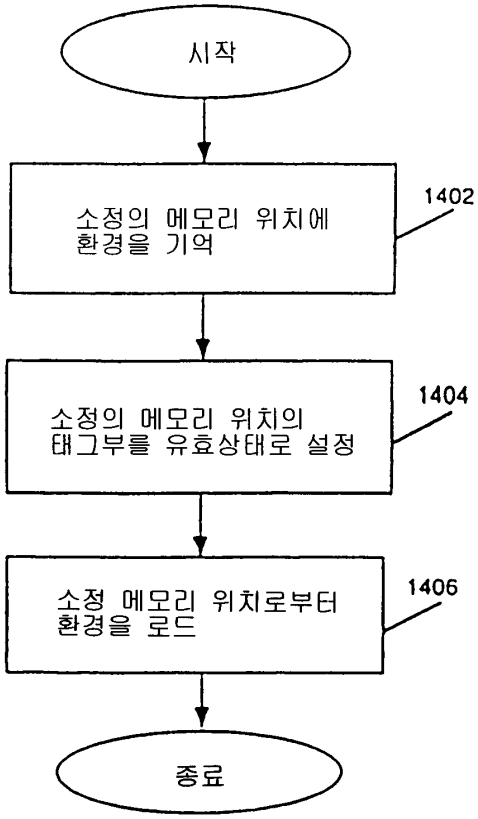
도면12C


85
1222
32 31
1224
0
X
정수데이터
정수기억포맷 1220

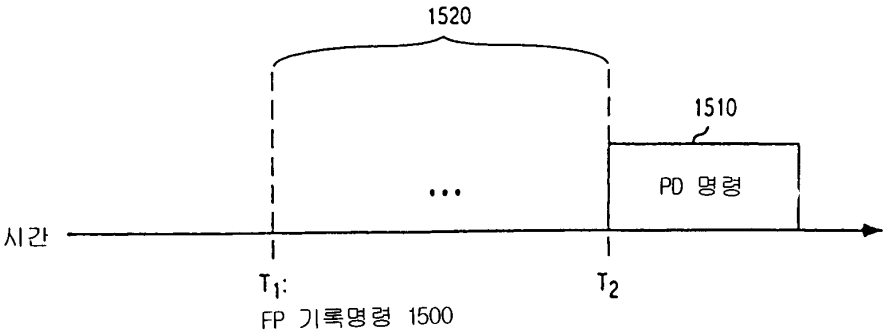
도면13


단계 1140 & 1142로부터
1142
압축 데이터 명령이 FP/PD 레지스터 또는 FP/PD 레지스터처럼 작용하는 버퍼 레지스터를 검색하는가? 1300
YES
NO
그 레지스터로부터 비트 [63:0] 를 검색 1302
압축 데이터 명령이 정수레지스터 또는 정수 레지스터 처럼 작용하는 버퍼 레지스터를 검색하는가? 1304
YES
NO
그 레지스터로부터비트 [31:0] 또는 [63:0] 를 검색 1306
연산을 수행 1308
압축 데이터 명령은 프로세서가 압축 데이터를 FP/PD 레지스터 또는 FP/PD 레지스터 처럼 작용하는 버퍼 레지스터를 기록하게 하는가? 1310
YES
NO
압축 데이터를 비트 [66:3] 에 저장및 그 레지스터의 비트 [85:68] 에 1를 저장 1312
압축 데이터 명령은 프로세서가 정수 레지스터 또는 정수 레지스터처럼 작용하는 버퍼 레지스터를 기록하게 하는가? 1314
YES
NO
그레지스터의 비트 [31:0] 에 데이터를 저장 1316
단계 1144로

도면14



도면15A



도면15B