(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.
G06F 9/38 (2006.01)

(45) 공고일자 2006년03월10일
(11) 등록번호 10-0559455
(24) 등록일자 2006년03월03일

(21) 출원번호 10-2003-7001229(분할)
(22) 출원일자 2003년01월27일
(62) 원출원 특허10-2001-7015019
원출원일자 : 2001년11월23일
번역문 제출일자 2003년01월27일
(86) 국제출원번호 PCT/JP1992/000868
국제출원일자 1992년07월07일

(65) 공개번호 10-2004-0000381
(43) 공개일자 2004년01월03일
심사청구일자 2001년11월23일
(87) 국제공개번호 WO 00000
국제공개일자 0000년00월00일

(30) 우선권주장 07/727,006 1991년07월08일 미국(US)

(73) 특허권자
세이코 엡슨 가부시키가이샤
일본 도쿄도 신주쿠구 니시신주쿠 2초메 4-1

(72) 발명자
엥구엔레트롱
미국캘리포니아95030몽트세레노다니엘플레이스15096

렌쯔데렉제이.
미국캘리포니아95032로스가토스필립스애비뉴17400

미야야마요시유끼
미국캘리포니아95050산타클라라란쵸맥코믹블러바드2171

가르그산지브
미국캘리포니아94539프리몬트센티널드라이브46820

하기와라야스아끼
미국캘리포니아95050산타클라라아파트274먼로스트리트2250

왕요하네스
미국캘리포니아94062레드우드시티킹스트리트25

라우테이-라이
미국캘리포니아94306팔로알토아파트E칼리지애비뉴411

트랑쿠앙에이취.
미국캘리포니아95130산호세메이필드애비뉴2045

(74) 대리인
특허법인 신성

심사관 : 성경아

(54) 고성능 알아이에스씨 마이크로프로세서 구조

요약

고성능 RISC 코어 기반 마이크로프로세서 구조는 메인 프로그램 명령어 스트림, 목표 조건부 분기 명령어 스트림 및 절차 명령어 스트림을 고려한 복수의 프리펫치 경로를 가진 명령어 프리펫치 유니트를 통해 메모리로부터 얻어지는 명령어의 동시 실행을 허용한다.

목표 조건부 분기 프리펫치 경로는 조건부 분기 명령어에 대한 가능한 명령어 스트림 둘 다 프리펫치될 수 있도록 허용한다. 절차 명령어 프리펫치 경로는 메인 또는 목표 프리펫치 버퍼를 클리어링하지 않고 추가 명령어 스트림이 억세스될수 있도록 허용한다.

각각의 명령어 셋트는 다수의 일정길이의 명령어를 포함한다. 제 1 및 제 2 버퍼를 포함하는 다수의 명령어 셋트 버퍼에 명령어 셋트를 버퍼링하기 위해 명령어 FIF0가 제공된다.

하나의 레지스터 파일 및 다수의 기능 유니트를 포함하는 명령어 실행 유니트에는 제 1 및 제 2 버퍼내의 명령어 셋트를 검사할 수 있고 이용가능한 기능 유니트에 의한 실행을 위해 명령어 중 소정의 명령어를 예정할 수 있는 명령어 제어 유니트가 제공된다. 기능 유니트와 레지스터 파일사이의 복수의 데이터 경로는 각각의 명령어의 실행에 필요할 때 기능 유니트에 의한 레지스터 파일에의 복수의 독자적인 억세스를 허용한다.

레지스터 파일은 임시 데이터 레지스터의 추가 셋트를 포함한다. 이들 임시 데이터 레지스터는 조건부 분기 명령어 또는 완료를 위해 추가의 기능 유니트 처리 사이클을 요하는 소정의 명령어의 완전한 실행에 앞서 명령어의 비순서적 실행에 의해 기능 유니트에 의해 처리되는 데이터를 수신하기 위해 명령어 실행 제어 유니트에 의해 이용된다. 임시 데이터 레지스터에 저장된 데이터는 모든 이전의 명령어가 실행 완료된 지점에서 명령어 스트림의 실제 상태에 따라 선택적으로 홀드 또는 클리어 되거나, 또는 레지스터 파일로 회수된다.

대표도

도 1

색인어

버퍼, 마이크로프로세서, 레지스터, 펫치, 명령어, 동시실행, 데이터 경로

명세서

도면의 간단한 설명

본 발명의 전술한 장점과 다른 장점 및 특징은 첨부 도면과 관련한 다음의 본 발명의 상세한 설명의 고찰에 의해 잘 이해될 것이다. 도면에서 비슷하 참조번호는 비슷한 부품을 나타낸다.

제1도는 본 발명을 구현하는 양호한 마이크로프로세서 구조의 간략한 블록도;

제2도는 본 발명에 따라 구성된 명령어 펫치 유니트의 상세 불럭도;

제3도는 본 발명에 따라 구성된 프로그램 카운터 로직 유니트의 블록도;

제4도는 프로그램 카운터 데이터 및 제어 경로 로직의 상세 블록도;

제5도는 본 발명의 명령어 실행 유니트의 간략한 블록도;

제6a도는 본 발명의 한 양호한 실시예에 사용되는 레지스터 파일 구조의 간략한 블록도;

제6b도는 본 발명의 한 양호한 실시예에 사용되는 임시 버퍼 레지스터 파일의 저장 레지스터 포맷의 그래픽 도면;

제6c도는 본 발명의 명령어 FIFO 유니트의 마지막 두 스테이지에 존재하는 바와 같은 1차 및 2차 명령어 셋트의 그래픽 도면;

제7a도 내지 제7c도는 본 발명의 한 양호한 실시예에 따라 제공되는 바와 같은 1차 정수 레지스터 셋트의 재현성 가능(reconfigurable) 스테이트의 그래픽 도면;

제8도는 본 발명의 양호한 실시예에 따라 제공되는 바와 같은 재현성 가능 부동 소수점 및 2차 정수 레지스터 셋트의 그래픽 도면;

제9도는 본 발명의 양호한 실시예에 제공되는 바와 같은 3차 부울 레지스터 셋트의 그래픽 도면;

제10도는 본 발명의 양호한 실시예에 따라 구성된 명령어 실행 유니트의 1차 정수 처리 데이터 경로 부분의 상세 블록도;

제11도는 본 발명의 양호한 실시예에 따라 구성된 명령어 실행 유니트의 1차 부동 소수점 데이터 경로 부분의 상세 블록도;

제12도는 본 발명의 양호한 실시예에 따라 구성된 명령어 실행 유니트의 부울 연산 데이터 경로 부분의 상세 블록도;

제13도는 본 발명의 양호한 실시예에 따라 구성된 로드/스토어 유니트의 상세 블록도;

제14도는 본 발명에 따라 다양한 명령어를 실행하는데 있어서의 본 발명의 양호한 실시예의 양호한 연산 시퀀스를 도시하는 타이밍도:

제15도는 본 발명의 양호한 실시예에 따라 구성된 가상(virtual) 메모리 제어 유니트의 간략한 블록도;

제16도는 본 발명의 양호한 실시예에 이용되는 가상 메모리 제어 알고리즘의 그래픽 도면;

제17도는 본 발명의 양호한 실시예에 이용되는 캐쉬 제어 유니트의 간략한 블록도.

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 일반적으로 RISC형 마이크로프로세서 구조의 설계에 관한 것으로서, 특히, 복수 명령어를 동시에 실행할 수 있는 RISC 마이크로프로세서 구조에 관한 것이다.

관련출원의 상호 참조

본 출원은 그 모두가 본원출원의 양수인에게 양도된 다음의 출원들과 관련된다:

1. SMOS 7985 MCF/GBR의 Le T. Nguyen 등에 의해 발명되고, 미합중국 출원번호 07/727,058, 출원일 1991. 7. 8.인 "EXTENSIBLE RISC MICROPROCESSOR ARCHITECTURE";

2. SMOS 7987 MCF/GBR의 Le T. Nguyen 등에 의해 발명되고, 미합중국 출원번호 07/726,744, 출원일 1991. 7. 8.인 "RISC MICROPROESSOR ARCHITECTURE WITH ISOLATED ARCHITECTURAL DEPENDENCIES";

3. SMOS 7988 MCF/GBR/RCC의 Sanjiv Garg 등에 의해 발명되고, 미합중국 출원번호 07/726,733, 출원일 1991. 7. 8.인 "RISC MICROPROCESSOR ARCHITECTURE IMPLEMENTING MULTIPLE TYPED REGISTER SETS";

4. SMOS 7989 MCF/GBR/WSW의 Le T. Nguyen 등에 의해 발명되고, 미합중국 출원번호 07/726,942, 출원일 1991. 7. 8.인 "RISC MICROPROCESSOR ARCHITECTURE IMPLEMENTING FAST TRAP AND EXCEPTION STATE";

5. SMOS 7991 MCF/GBR/HKW의 Derek J. Lentz 등에 의해 발명되고, 미합중국 출원번호 07/726,929, 출원일 1991. 7. 8.인 "SINGLE CHIP PAGE PRINTER CONTROLLER";

6. SMOS 7992 MCF/WMB의 Derek J. Lentz 등에 의해 발명되고, 미합중국 출원번호 07/726,893, 출원일 1991. 7. 8.인 "MICROPROCESSOR ARCHITECTURE CAPABLE OF SUPPORTING MULTIPLE HETEROGENEOUS PROCESSORS".

관련 기술의 설명:

최근에, 마이크로프로세서 구조의 설계는 복합 명령어 컴퓨터(Complex Instruction Set Computer : CISC)의 사용으로부터 보다 간단한 명령어 제한 컴퓨터(Reduced Instruction Set Computer : RISC) 구조로 발전되어 왔다. CISC 구조는 명령어 실행 파이프라인을 구현 및 지원하기 위한 실질적인 하드웨어의 설비에 특히 저명하다. 통상적인 파이프라인 구조는 정해진 순서대로, 명령어펫치(fetch), 명령어 디코드, 데이터 로드(load), 명령어 실행 및 데이터 기억 스테이지를 포함한다. 파이프라인의 각 스테이지를 통한 명령어의 상이한 부분의 동시 실행에 의해 성능의 장점이 얻어진다. 파이프라인이 길수록, 이용가능한 실행 스테이지의 수가 커지고, 동시에 실행될 수 있는 명령어의 수가 커진다.

일반적인 문제점들이 CISC 파이프라인 구조의 유효성을 제한한다. 그 첫 번째 문제점은 이전의 조건 코드 셋팅 명령어가 파이프라인을 통해 실질적으로 실행을 완료할 때까지 조건부 분기(conditional branch) 명령어가 적절하게 평가되지 않을 수도 있다는 것이다.

그러므로, 조건부 분기 명령어의 후속 실행이 지연되거나 스톨(stall) 상태로 되어, 몇몇의 파이프라인 스테이지가 다수의 프로세서 사이클동안 비활성 상태로 있도록 한다.

통상적으로, 조건 코드는 조건 코드 레지스터에 기록되는데, 이 레지스터를 실행 스테이지를 통해 명령어의 처리를 완료한 경우에만 프로세서 상태 레지스터(processor status register : PSR) 로서도 언급된다. 그러므로, 파이프라인은 분기 조건 코드의 결정을 연기하는 복수의 프로세서 사이클 동안 디코더 스테이지에서의 조건부 분기 명령어에 의해 스톨(stall) 상태로 되어야만 한다. 파이프라인의 스톨상태(stalling)는 처리량의 상당한 손실을 초래한다. 더욱이, 컴퓨터의 평균 처리량은 실질적으로 프로그램 명령어 스트림에서 조건 코드 셋팅 명령어 후에 바로 나타나는 조건부 분기 명령어의 단순한 횟수에 의존한다.

두 번째 문제점은 프로그램 명령어 스트림에서 근접하여 발생하는 명령어는 프로세서 레지스터 파일의 동일한 레지스터를 참조하기 쉽다는 사실로부터 발생한다. 데이터 레지스터는 종종 연속한 명령어의 저장(store) 및 로드(load) 스테이지에서 데이터의 목적지(destination) 또는 원천지(source)로서 사용된다. 일반적으로, 데이터를 레지스터 파일에 저장하는 명령어는 후속 명령어의 로드 스테이지 처리가 레지스터 파일을 억세스하도록 허용될 수 있게 되기 전에 최소한 실행 스테이지를 통한 처리를 완료해야한다. 많은 명령어의 실행은 저장 데이터를 발생하기 위해 단일 실행 스테이지에서 다양한 프로세서 사이클을 요하기 때문에, 통상적으로 실행 스테이지 연산의 지속기간 동안에 전체 파이프라인이 스톨상태로 된다. 결과적으로 컴퓨터의 실행 처리량은 거의 실행되고 있는 명령어 스트림의 내부 순서(order)에 의존한다.

세 번째 문제점은 명령어 자체의 실행보다는 오히려, 마이크로프로세서 자체의 하드웨어 지원 명령어 실행 환경, 즉 머신 스테이트(state-of-the-machine)의 관리로부터 발생한다. 현재의 CISC 마이크로프로세서 하드웨어 서브-시스템은 명령어 실행중에 트랩(trap) 상황의 발생을 검출할 수 있다. 트랩은 하드웨어 인터럽트, 소프트웨어 트랩, 예외(exceptions)를 포함한다. 각각의 트랩은 프로세서에 의한 대응하는 트랩 처리 루틴의 실행을 요한다. 트랩의 검출시, 실행 파이프라인은 트랩 처리 루틴의 즉시 실행을 허용하도록 클리어되어야 한다. 동시에, 머신 스테이트는 트랩이 발생하는 정확한 지점에서 설정되어야 하는데, 정확한 지점은 인터럽트 및 트랩에 대한 현재 실행중이 명령어의 종결시와 예외로 인해 실패한 명령어 바로전에 발생한다. 계속해서, 머신 스테이트 및 또한 트랩의 특성에 따라 실행 명령어 자체가 처리 루틴의 완료시에 복원되어야 한다. 결과적으로, 각각의 트랩 또는 관련 사건에 있어, 처리 루틴의 발단 및 종결과 프로세서의 처리량에서의 대응하는 감소가 있는 정확한 머신 스테이트의 저장 및 복귀 모두에서 파이프라인의 클리어에 의해 대기시간(latency)이 도입된다. 이들 문제점은 CISC 구조의 잠재적인 처리량을 개선하기 위한 노력으로 다양하게 제기되어 왔다. 조건부 분기 명령어의 적절한 실행에 관한 가정을 설정하여, 분기 조건 코드의 최종 결정에 앞서 파이프라인 실행이 시험적으로 진

행하도록 허용 할 수 있다. 또한, 레지스터가 변형될 것인지에 관한 후속 가정을 설정하여 후속 명령어도 또한 시험적으로 실행되도록 허용할 수 있다. 마지막으로, 처리 루틴의 실행을 필요로 하는 예외의 발생을 최소화하여 프로그램 명령어 스트림의 처리를 인터럽트하는 예외 빈도수를 감소시키기 위한 상당한 추가 하드웨어가 제공 될 수 있다.

이들 해결책은 상당한 추가 하드웨어 복잡성을 초래하며, 또한 그 자체의 독특한 문제점을 초래한다. 분기 조건이나, 레지스터 파일 저장 억세스의 최종 해결 이전의 명령어의 연속적인 실행은 머신 스테이트가 조건부 분기의 위치 및 각 레지스터 파일의 각 변형을 포함하는 프로그램 명령어 스트림내의 다양한 지점중 한 지점으로 복원될 수 있도록 요구하며, 또한 예외의 발생에 대해서는 마지막 몇몇의 명령어의 완전히 완료된 실행 이전의 한 지점으로 복원되는 것을 필요로 한다. 결과적으로, 더욱 많은 지원 하드웨어가 요구되며, 파이프라인 스테이지의 사이클 시간을 현저하게 증가시키지 않도록 특별히 설계되어야 한다.

RISC 구조는 마이크로프로세서의 하드웨어 구현을 철저하게 단순화함으로써 전술한 많은 문제점을 회피하고자 추구되어 왔다. 극단적으로, 각각의 RISC 명령어는 로드 사이클, 실행 사이클, 저장 사이클을 포함하는 오직 3개의 파이프라인 프로그램 사이클에서 수행한다. 로드 및 저장 데이터 바이패스의 이용을 통해, 통상적인 RISC 구조는 근복적으로 3-스테이지 파이프라인에서 사이클당 단일 명령어를 실행할 수 있다.

가능하다면 언제든지, 필요한 기능을 수행하기 위한 소프트웨어 루틴을 위해 RISC 구조에서의 하드웨어 지원은 최소화된다. 결과적으로, RISC 구조는 최적으로 정합된 파이프라인에 의해 실행되는 단순한 로드/저장 명령어 셋트의 이용을 통해 실질적인 유연성(flexibility) 및 높은 속도의 희망을 계속 유지한다. 또한 실제적으로, RISC 구조는 모든 원하는 기능을 구현하기 위한 상당한 많은 수의 명령어를 실행할 필요성과 짧고 고성능의 파이프라인 사이의 균형으로부터 이익이 되는 것으로 알려졌다.

RISC 구조의 설계는 일반적으로 분기(branches), 레지스터 참조(references) 및 예외(exceptions)에 관해 CISC 구조에 의해 부딪히는 문제점들을 없애거나 최소화 한다. RISC 구조에 관련된 파이프라인은 짧으며 속도에 있어 최적이다. 파이프라인 짧음으로써 파이프라인인 스톨 또는 클리어의 결과가 최소화되며, 또한 머신 스테이트를 초기 실행 지점(point)으로 복원(restore)하는데 있어서의 문제점이 최소화된다.

그러나, 종래의 RISC 구조에 의해서는 일반적으로 실현된 현재 레벨 이상의 뚜렷한 처리량 성능 이득이 쉽게 실혈될 수 없다. 결과적으로, 대체의 소위 수퍼-스칼라(super-scaler) 구조가 다양하게 제안되어 왔다. 이들 구조는 일반적으로 다양한 명령어를 동시에 실행하여 프로세서의 처리량을 비례적으로 증가시키기 위한 시도를 한다. 불행하게도, 이들 구조도 역시 CISC 구조에 의해 부딪히는 문제점과 동일하진 않지만 비슷한 조건부 분기, 레지스터 참조, 예외 처리 문제점에 종속된다.

발명이 이루고자 하는 기술적 과제

발명의 요약

그러므로 본 발명의 일반적인 목적은 종래의 CISC 및 RISC 구조보다 실질적인 성능 이익을 얻을 수 있고, 마이크로프로세서 구현에 적합한 고성능 RISC 기반 수퍼-스칼라 프로세서 구조를 제공하는 것이다.

상기 목적은 본 발명에서 명령어 기억장치로부터 얻어지는 명령어의 동시 실행을 할 수 있는 마이크로프로세서 구조의 제공을 통해 이루어진다. 마이크로프로세서 구조는 명령어 기억장치로부터 명령어 셋트를 펫치(fetch) 하기 위한 명령어 프리펫치(prefetch) 유니트를 포함한다. 각각의 명령어 셋트는 다수의 일정 길이의 명령어를 포함한다. 명령어 FIFO는 제1 및 제2 버퍼를 포함하는 다수의 명령어 셋트 버퍼에 명령어 셋트를 버퍼링하기 위해 제공된다. 하나의 레지스터 파일과 다수의 기능 유니트를 포함하는 명령어 실행 유니트는 제1 및 제2 버퍼내의 명령어 셋트를 검사(examine) 할 수 있고 이용가능한 기능 유니트에 의한 실행을 위해 이들 명령어중 임의 명령어를 발생할 수 있는 명령어 제어 유니트를 갖추고 있다. 기능 유니트와 레지스터 파일 사이의 다양한 데이터 경로는 각각의 명령어의 동시 실행을 위해 필요할 때 레지스터 파일로의 다양한 독자적인 억세스를 허용한다.

레지스터 파일은 레지스터 데이터의 임시(temporary) 저장을 위해 사용되는 추가적인 데이터 레지스터 셋트를 포함한다. 이들 임시 데이터 레지스터는 명령어의 비순서(out-of-order) 실행시 기능 유니트에 의해 처리된 데이터를 수신하기 위해 명령어 실행 제어 유니트에 의해 이용된다. 임시 데이터 레지스터에 저장된 데이터는 선택적으로 유지되며(held), 모두 이전의 정순서(in-order) 명령어가 완전하게 실행되어 회수(retired)된 명령어 스트림내의 명령어의 위치로 정확한 머신 스테이트가 진행하는 경우에 클리어 되거나 레지스터 파일로 회수된다.

마지막으로, 명령어 기억장치로부터의 명령어 셋트의 프리펫치는 메인 프로그램 명령어 스트림, 목표(target) 조건부 분기 명령어 스트림 및 절차적 명령어 스트림의 프리펫치를 허용하는 다양한 프리펫치 경로에 의해 용이하게 된다. 목표 조건부 분기 프리펫치 경로는 조건부 분기 명령어에 대한 가능한 두 개 모두, 즉 메인 및 목표 명령어 스트림이 동시에 프리펫치될 수 있도록 한다. 절차적(procedural) 명령어 프리펫치 경로는 메인 또는 목표 명령어 스트림에 존재하는 유일의 (singular) 명령어를 구현하는 확장된 절차의 실행을 허용하는데 효과적인 추가(supplementary) 명령어 스트림을 허용하는데, 절차적 프리펫치 경로는 적어도 메인 프리펫치 버퍼없이 이들 확장된 절차가 펫치되어 실행될 수 있도록 한다.

결과적으로, 본 발명의 장점은 본질적으로 RISC 형태의 코어구조를 이용하는 극도의 고성능의 처리량을 실현하는 구조를 제공한다는 것이다.

본 발명의 다른 장점은 한 사이클당 다양한 명령어의 실행을 제공한다는 것이다.

본 발명의 또다른 장점은 다양한 명령어를 동시에 최적으로 실행하는데 필요한 기능 유니트의 동적(dynamic) 선택 및 이용을 제공한다는 것이다.

본 발명의 또다른 장점은 정확한 머신 스테이트 복귀 능력을 지원하기 위한 메카니즘을 총체적으로 포함하는 레지스터 파일 유니트를 제공하는 것이다.

본 발명의 또다른 장점은 독자적인 병렬의 정수 레지스터 파일과 같은 연산, 부동 소수점(floating point) 및 정수 레지스터 파일과 같은 연산 및 전용 부울 레지스터 파일과 같은 연산을 포함하는 다양한 레지스터 파일을 할 수 있는 일반화되고 전형화된 다양한 레지스터 파일을 레지스터 파일 유니트내에 포함하고 있다는 것이다.

본 발명의 또다른 장점은 효율적인 명령어 최소 메카니즘 및 로드/스토어 순서 동기화 장치를 포함하여 정확한 머신 스테이트 능력의 이용을 통해 로드 및 스토어 연산과 예외 및 인터럽트 처리가 정확한 방식으로 수행될 수 있다는 것이다.

본 발명의 또다른 장점은 대기시간을 최소화하고 처리량을 향상시키기 위해 트랩 스테이트를 지원하는 전용 레지스터 파일 유니트를 제공하는 것이다.

본 발명의 또다른 장점은 부정확한 목표 분기 스트림 실행 조차도 본 발명에 의해 얻어질 수 있는 전체 처리양에 최소로 영향을 미치도록 하는 메인 및 목표 분기 명령어 프리펫치 큐(queues)을 제공하는 것이다. 더욱이, 절차적 명령어 프리펫치 큐는 절차적 루틴의 실행을 통한 새로운 명령어의 효과적인 구현 및 특히 내장된(built-in) 절차적 명령어를 구현하는 절차적 루틴의 외부에서 제공되는 수정(revision)을 허용하도록 하기 위한 메인 및 목표 분기 명령어의 실행에의 효율적인 방식의 삽입을 허용한다.

발명의 구성 및 작용

본 발명의 상세한설명

Ⅰ. 마이크로프로세서 구조의 개략적 설명

Ⅱ. 명령어 펫치 유니트

A) IFU 데이터 경로

B) IFU 제어 경로

C) IFU/IEU 제어 인터페이스

D) PC 로직 유니트 상세

1) PF 및 ExPC 제어/데이터 유니트 상세

2) PC 제어 알고리즘 상세

Ⅰ. 마이크로프로세서 구조의 개략적 설명

제1도에는 본 발명의 구조(100)가 일반적으로 도시되어 있다. 이 구조(100)의 주요 동작 구성요소는 명령어 펫치 유니트(Instruction Fetch Unit ; IFU)(102)와 명령어 실행 유니트 (Instruction Execution Unit ; IEU)(104)이다. 가상 메모리 유니트(Virtual Memory Unit ; VMA)(108)과 캐쉬 제어 유니트(Cache Control Unit ; CCU)(106)는 IFU(102)와 IEU(104)의 기능을 직접 지원하기 위해 제공된 것이다. 메모리 어레이 유니트(Memory Array Unit ; MAU)(112)는 상기 구조(100)의 연산을 위한 일반적으로 필수적인 구성요소로서 제공되는데, 이 MAU(112)는 상기 구조(100)의 절대 필요한 소자로서 존재하는 것은 아니다. 즉, 본 발명의 양호한 실시예에서, IFU(102), IEU(104), VMU(108), CCU(106), 및 MCU(110)는 종래의 0.8 미크론 설계 규정 저전력 CMOS 공정을 이용하고 약 1,200,000 트랜지스터를 포함하는 단일 실리콘 다이(die) 상에서 제조된다. 상기 구조(100)의 표준 프로세서 또는 시스템 블록 속도는 40㎒이다. 그러나, 본 발명의 한 양호한 실시예에 따라, 내부 프로세서 클럭 속도는 160㎒이다.

IFU(102)는 1차적으로 명령의 펫치를 해야하며, IEU(104)에 의해 실행이 연기된 명령어의 버퍼링, 그리고 일반적으로 다음 명령어의 펫치에 사용될 다음의 가상 어드레스의 계산을 수행해야 한다.

본 발명의 양호한 실시예에서, 명령어는 각각 32 비트의 길이로 고정되어 있다. 4개의 명령어로 된 명령어 셋트 또는 "버킷(buckets)이 128 비트 폭(wide)의 명령어 버스(114)를 통해 CCU(106)내의 명령어 캐시(132)로부터 동시에 IFU(102)에 의해 펫치된다. 명령어 셋트의 전달은 제어 버스(116)를 통해 제공되는 제어 신호에 의해 IFU(102)와 CCU(106) 사이에서 동등하게 이루어진다. 펫치될 명령어 셋트의 가상 어드레스는 IFU에 연결된 중재(arbitration), 제어 및 어드레스 버스(118)을 통해, IEU(104)와 VMU(108) 사이에 연결된 공유 중재 제어, 어드레스 버스(120) 상으로 IFU(102)에 의해 제공된다. VMU(108)에의 억세스를 위한 중재는 IFU(102)와 IEU(104) 모두가 공통의 공유 자원(resource)으로서 VMU(108)을 이용한다는 사실로부터 발생한다. 상기 구조(100)의 양호한 실시예에서, 가상 어드레스의 물리적 페이지(physical page) 내의 어드레스를 정의하는 저순위(low order) 비트는 IFU(102)에 의해 제어 라인(116)을 통해 직접 캐쉬 제어 유니트(106)으로 전달된다. IFU(102)에 의해 공급되는 가상 어드레스의 가상화한 고순위 비트는 버스(118,120)의 어드레스 부분에 의해 대응하는 물리적 페이지 어드레스로의 번역을 위해 VMU(108)으로 제공된다. IFU(102)에 있어서, 상기 물리적 페이지 어드레스는 번역 요청이 VMU(108)에 이루어진후 내부 프로세서 사이클의 1/2 사이클 동안에 VMU(108)로부터 어드레스 제어 라인(122)을 통해 캐쉬 제어 유니트(106)로 직접 전달된다.

IFU(102)에 의해 펫치된 명령어 스트림은 다음에 명령어 스트림 버스(124)를 통해 IEU(104)로 제공된다. 제어신호는 제어라인(126)을 통해 IFU(102)와 IEU(104) 사이에서 교환된다. 또한, 통상적으로 IEU(104)내에 있는 레지스터 파일내의 억세스를 요구하는 명령어 펫치 어드레스는 제어 라인(126) 내의 목표 어드레스 복귀 버스를 통해 IFU로 역 제공된다.

IEU(104)은 80-비트 폭의 양방향 데이터 버스(130)를 통해 CCU(106)내에 제공된 데이터 캐쉬(134)에 관한 데이터를 저장 및 검색한다. IEU 데이터 억세스에 대한 전체 물리적 어드레스는 제어 버스(128)의 어드레스 부분을 통해 CCU(106)로 제공된다. 제어 버스(128)는 데이터 전달을 관리하기 위한 IEU(104)와 CCU(106) 사이에서 제어신호의 교환을 제공한다. IEU(104)는 가상 데이터 어드레스를 CCU(106)에 제공하기 적합한 물리적 데이터 어드레스로 변환하기 위해 자원으로서 VMU(108)를 이용한다. 데이터 어드레스를 가상화한 부분은 중재, 제어 및 어드레스 버스(120)를 통해 VMU(108)에 제공된다. IFU(102)에 관한 연산과는 달리, VMU(108)는 대응하는 물리적 어드레스를 버스(120)를 통해 IEU(104)로 복귀시킨다. 상기 구조(100)의 양호한 실시예에서, IEU(104)는 로드/스토어 연산이 적절한 프로그램 스트림 순서로 발생하도록 보장하는데 사용하기 위한 물리적 어드레스를 필요로 한다.

CCU(106)는 데이터에 대한 물리적 어드레스로 정의된 요청이 명령어 및 데이터 캐쉬(132,134)로부터 만족될 수 있는지를 결정하는 일반적으로 통상적인 고레벨 기능을 수행한다. 명령어 또는 데이터 캐쉬(132,134)에의 억세스에 의해 억세스 요청이 적절하게 충족되는 경우에 CCU(106)느 데이터 버스(114,128)를 통해 데이터 전달을 통합하여 수행한다.

데이터 억세스 요청이 명령어 또는 데이터 캐쉬 (132,134)로부터 충족될 수 없는 경우에, CCU(106)는 MAU(112)의 판독 또는 기록 억세스가 요구되는 것인지 식별하기 위한 충분한 제어 정보, 각각의 요청에 대한 CCU(106)의 원천지 또는 목적지 캐쉬(132,134), 및 요청 동작이 IFU(102) 또는 IEU(104)에 의해 발생된 바와 같은 궁극적인 데이터 요청과 상관될 수 있도록 허용하기 위한 추가적인 식별 정보와 함께 대응하는 물리적 어드레스를 MCU(110)에 제공한다.

MCU(110)는, 단방향 데이터 버스(136)에 의해 CCU(106)의 명령어 캐쉬(132)와 결합되고 양방향 데이터 버스(138)에 의해 데이터 캐쉬(134)에 결합된 포트 스위치 유니트(142)를 포함하는 것이 바람직하다. 포트 스위치(142)는 본질적으로,

제어 버스(140)로부터 얻어지는 물리적 어드레스가 다수의 포트 P_0 내지 P_N(146_{0-n})한 포트로 루팅될 수 있도록 하고 포트로부터 데이터 버스(136,138) 까지의 양방향 전달을 허용하는 대형 멀티플렉서이다. MCU(110)에 의해 처리되는 각각의 메모리 억세스 요청은 MAU(112)의 억세스에 요구되는 바와 같은 메인 시스템 메모리 버스(162)에의 억세스를 위한 중재 목적을 위해 포트(146_{0-n})중 한 포트와 연관된다. 일단 데이터 전달 접속이 설정되면, MCU는 포트 스위치(142) 및 포트(146_{0-n}) 중 대응하는 포트를 통해 명령어 캐쉬(132)나 데이터 캐쉬(134)중 하나와 MAU(112) 사이의 데이터 전달을 표시하기 위해 제어 버스(140)를 통해 CCU(106)에 제어 정보를 제공한다. 상기 구조(100)의 양호한 실시예에 따라, MCU(110)는 실제적으로, CCU(106)와 MAU(112) 사이의 이동중인 데이터를 저장하거나 랫치하지 않는다. 이것은 MCU(110)에 유일하게 존재할 수 도 있는 데이터를 트래킹 또는 관리할 필요성을 없애고 전달중의 대기시간을 최소화하기 위한 것이다.

Ⅱ. 명령어 펫치 유니트:

제2도에는 명령어 펫치 유니트(102)의 기본 구성 요소가 도시되어 있다. 이들 구성요소의 동작 및 상호관계를 IFU 데이터 및 제어 경로를 고찰함으로써 잘 이해될 수 있다.

A)IFU 데이터 경로

IFU 데이터 경로는 프리펫치 버퍼(260)에서의 임시 저장을 위해 명령어 셋트를 수신하는 명령어 버스(114)로부터 시작된다. 프리펫치 버퍼(260)로부터의 명령어 셋트는 IDecode 유니트(262)를 통해 IFIFO 유니트(264)로 통과된다. 명령어 FIFO(264)의 마지막 두 스테이지에 저장된 명령어 셋트는 계속해서 데이터 버스(278,280)를 경유해 IEU(104)에 이용 가능하다.

프리펫치 버퍼 유니트(260)는 명령어 버스(114)로부터 한번에 하나의 명령어 셋트를 수신한다. 일반적으로, 전체 128 비트 폭의 명령어 셋트가 프리펫치 버퍼(260)의 메인 버퍼(MBUF)(188)부분내의 4개의 128 비트 폭의 프리펫치 버퍼 위치중 한 위치에 병렬로 기록된다. 비슷하게, 최대 4개의 추가 명령어 셋트가 2개의 128비트 폭의 목표(Target) 버퍼(TBUF)(190) 프리펫치 버퍼 위치나 2개의 128 비트 폭의 절차((Procedural) 버퍼(EBUF)(192) 프리펫치 버퍼 위치에 기록될 수도 있다. 상기 구조(100)의 양호한 실시예에서, MBUF(188), TBUF(190) 또는 EBUF(192) 내의 프리펫치 버퍼 위치중 한 위치내의 명령어 셋트는 프리펫치 버퍼 출력 버스(196)로 전달될 수도 있다. 부가적으로, 명령어 셋트 버스(194)를 프리펫치 버퍼 출력 버스(196)에 직접 연결하여 MBUF, TBUF 및 EBUF(188,190,192)를 바이패스 시키기 위해 비실현(fall through) 명령어 셋트 버스(194)가 제공될 수도 있다.

양호한 실시예의 구조(100)에서, MBUF(188)는 정상 또는 메인 명령어 스트림내의 명령어 셋트를 버퍼시키기 위해 이용된다. TBUF(190)는 시험적인 목표 분기 명령어 스트림으로부터 펫치된 명령어 셋트를 버퍼하는데 이용된다. 결과적으로, 프리펫치 버퍼 유니트(260)는 조건부 상기 명령어 다음에 오는 모든 가능한 명령어 스트림이 프리펫치 될 수 있도록 허용한다. 이것은 조건부 분기 명령어의 해결시 결국 선택되는 특정 명령어 스트림에 관계없이 조건부 분기 명령어 다음에 오는 실행을 위한 정당한 다음 명령어 셋트를 얻기 위해 적어도 CCU(106)에의 더 이상의 억세스를 위한 대기tl간을 없애는데, 그렇지 않을 경우에는 MAU(112)의 대기시간이 상당히 더 커지게 된다. 본 발명의 양호한 구조(100)에서, MBUF(188)와 TBUF(190)의 제공으로 명령어 펫치 유니트(102)가 2개 모두의 가능한 명령어 스트림을 프리펫치할 수 있게 되며, 또한 명령어 실행 유니트(104)와 관련하여 후술되게 되는 바와 같이, 추정된 정당한 명령어 스트림의 실행을 허용한다. 조건부 분기 명령어의 해결시 정당한 명령어 스트림이 MBUB(188)로 프리펫치된 경우에, TBUF(190)내의 임의 명령어 셋트가 단순히 무효화될 수도 있다. 정당한 명령어 스트림의 명령어 셋트가 TBUF(190)에 존재하는 경우에 명령어 프리펫치 버퍼 유니트(260)는 TBUF(190)으로부터 MBUF(188)내의 각 버퍼 위치로의 상기 명령어 셋트의 직접 측방향(lateral) 전달을 제공한다. 이전에 MBUF(188)에 저장된 명령어 셋트는 TBUF(190)에 전달되는 명령어 셋트에 의해 중복기록됨으로써 효과적으로 무효화된다. MBUF 위치에 전달되는 TBUF 명령어 셋트가 없는 경우에 ,상기 위치는 단수히 무효 표시된다.

비슷하게, EBUF(192)는 프리펫치 버퍼(260)를 통하는 다른 교호의 (alternate) 프리펫치 경로로서 제공된다. EBUF(192)는 바람직하게, MBUF(188) 명령어 스트림에서 나타나는 "절차(procedural)" 명령어인 단일 명령어에 의해 지정된 동작을 구현하기 위해 사용되는 교호의 명령어 스트림을 프리펫칭하는데 사용된다. 이와 같은 방식으로, 복합 또는 확장 명령어가 소프트웨어 루틴 또는 절차를 통해 구현될 수 있으며, MBUF(188)에 이미 프리펫치된 명령어 스트림을 방해하지 않고 프리펫치 버퍼 유니트(260)를 통해 처리될 수 있다. 비록 본 발명은 일반적으로 TBUF(190)에서 제일 먼저 만나게 되는 절차 명령어의 처리를 허용하지만, 절차 명령어 스트림의 프리펫치는 모든 이전의 보류된 조건부 분기 명령어가

해결될 때까지 보류된다. 이것은 절차 명령어 스트림에서 발생하는 조건부 분기명령어가 TBUF(190)의 이용을 통해 일관성 있게 처리될 수 있도록 허용한다. 절차스트림에서 분기가 이루어지는 경우에, 목표 명령어 셋트는 TBUF(190)로 펫치되게 되며, 단순히 EBUF(192)로 측방향 전달될 수 있다.

마지막으로, MBUF(188), TBUF(190) 및 EBUF(192) 각각은 프리펫치 유니트에 의해 저장된 명령어 셋트를 출력 버스(196)에 제공하기 위해 프리펫치 버퍼 출력 버스(196)에 결합된다. 부가적으로, 명령어 버스(114)로부터 출력 버스(196)로 직접 명령어 셋트를 전달하기 위해 흐름 관통 버스(194)가 제공된다.

본 발명의 양호한 구조(100)에서, MBUF(188), TBUF(190) 및 EBUF(192) 내의 프리펫치 버퍼들이 직접 FIFO 구조를 형성하지는 않는다. 대신에, 출력 버스(196)에 임의 버퍼 위치 연결을 제공함으로써, 명령어 캐쉬(132)로부터 검색되는 명령어 셋트의 프리펫치 순서화가 실질적으로 자유로워질 수 있다. 즉, 명령어 펫치 유니트(102)는 일반적으로 명령어의 적절한 명령어 스트림 순서로 명령어 셋트를 판단 및 요청한다. 그러나, 명령어 셋트가 IFU(102)에 복귀되는 순서가 몇몇의 요청된 명령어 셋트는 CCU(106)로부터 단독으로 이동 및 억세스 가능하게 되고 다른 셋트는 MAU(112)의 억세스를 요청하는 환경에 부합하도록 적절하게 순서를 벗어나 발생하도록 허용된다.

비록 명령어 셋트가 프리펫치 버퍼 유니트(260)에 순서대로 복귀되지 않을 수도 있지만, 출력 버스(196) 상에 출력되는 명령어 셋트의 순서는 일반적으로 IFU(102)에 발생되는 명령어 셋트 요청의 순서에 따라야 하며, 순서적(in-order) 명령어 스트림 순서는 예를 들어 목표 분기 스트림의 시험적 실행에 종속된다.

IDecode 유니트(262)는 프리펫치 버퍼 출력 버스(196)로부터, IFIFO 유니트(264)가 공간을 허용하는 일반적으로 한 사이클당 한 셋트씩 명령어 셋트를 수신한다. 하나의 명령어 셋트를 형성하는 4개 명령어로된 각각의 셋트는 IDecode 유니트(262)에 의해 병렬로 디코드된다. IFU(102)의 제어 경로부를 위해 라인(318)을 통해 관련 제어 흐름 정보가 추출되지만, 명령어 셋트의 내용은 IDecode 유니트(262)에 의해 변경되지 않는다.

IDecode 유니트(162)로부터의 명령어 셋트는 IFIFO 유니트(264)의 128 비트 폭의 입력 버스(198) 상에 제공된다. 내부적으로 IFIFO 유니트(264)는 일련의 마스터/슬레이브 레지스터(200,204,208,212,216,220,224)로 구성된다. 각각의 레지스터는 마스터 레지스터(200,208,216)의 내용이 FIFO 연산의 내부 프로세서 사이클의 제 1 반(half)-사이클 동안에는 슬레이브 레지스터(204,212,220)에 전달되고, FIFO 연산의 후속 반-사이클 동안에는 다음의 연속된 마스터 슬레이브(208,216,224)에 전달될 수 있도록 그 후속 레지스터에 결합된다. 입력 버스(198)는 FIFO 연산의 제 2 반-사이클 동안에 IDecode 유니트(262)로부터 직접 마스터 레지스터로의 명령어 셋트의 로딩을 허용하도록 마스터 레지스터(200,208,216,224) 각각의 입력에 연결된다. 그러나, 입력 버스(198)로부터의 마스터 레지스터의 로딩은 IFIFO 유니트(264) 내에서의 데이터의 FIFO 시프트와 동시에 이루어질 필요는 없다. 결과적으로, IFIFO 유니트(264)는 명령어 FIFO 유니트(264) 내에 저장된 명령어 셋트의 현재의 깊이에 관계없이 또한 IFIFO 유니트 (264)를 통한 데이터의 FIFO 시프팅에 무관하게 입력 버스(198)로부터 연속적으로 채워질 수 있다.

각각의 마스터/슬레이브 레지스터(200,204,208,212,216,220,224)는 128 비트 폭의 명령어 셋트의 완전 병렬 저장을 제공할 뿐만 아니라, 또한 각각의 제어 레지스터(202,206,210,214,218,222,226)내의 수 비트의 제어 정보의 저장도 제공한다. 양호한 제어 비트 셋트는 예외 미스(miss) 및 예외 수정(modify), (VMU), 노 메모리(no memory)(MCU), 분기 바이어스, 스트림, 오프셋트(IFU)를 포함한다. 이 제어 정보는 입력 버스(198)로부터의 새로운 명령어로 IFIFO 마스터 레지스터를 로딩하는 것과 동시에 IFU(102)의 제어 경보 부분으로부터 기원한다. 그러므로, 제어 레지스터 정보는 IFIFO 유니트(264)를 통해 명령어 셋트와 병렬로 동시에 시프트 된다.

마지막으로, 본 발명의 양호한 구조(100)에서, IFIFO 유니트(264)로부터의 명령어 셋트의 출력은 Ⅰ_버킷_0(Ⅰ_BUCKET 0) 및 Ⅰ_버킷_1(Ⅰ_BUCKET_1) 명령어 셋트 출력 버스(278,280) 상에서 마지막 2개의 마스터 슬레이브(216,224)로부터 동시에 얻어진다. 부가적으로, 대응하는 제어 레지스터 정보는 IBASVO 및 IBASV1 제어 필드 버스(282,284)에 제공된다. 이들 출력 버스(278,282,280,284)는 모두 IEU(104)에 명령어 스트림 버스(124)로서 제공된다.

B) IFU 제어 경로

IFU(102)에 대한 제어 경로는 프리펫치 버퍼 유니트(260), IDecode 유니트(262) 및 IFIFO 유니트(264)의 연산을 직접적으로 지원한다. 프리펫치 제어로직 유니트(266)는 1차적으로 프리펫치 버퍼 유니트(260)의 연산을 관리한다. 프리펫치 제어로직 유니트(266) 및 IFU(102)는 일반적으로, IEU(104), CCU(106) 및 VMU(108)의 연산과 IFU 연산을 동기화시키기 위해 클럭 라인(290)을 통해 시스템 클럭 신호를 수신한다. MBUF(188), TBUF(190) 및 EBUF(192)에 명령어 셋트를 선택 및 기록하기에 적합한 제어신호는 제어라인(304) 상에 제공된다.

제어라인(316) 상의 다수의 제어신호는 프리펫치 제어로직 유니트(266)에 제공된다. 특히, 펫치 요청 제어신호는 펫치 연산을 개시하기 위해 제공된다. 제어라인(316) 상에 제공되는 다른 제어신호는 MBUF(188), TBUF(190) 또는 EBUF(192)와 같은 요청된 프리펫치 연산의 의도된 목적지를 식별한다. 프리펫치 요청에 응답하여, 프리펫치 제어 로직 유니트(266)는 ID 값을 발생하고, 프리펫치 요청이 CCU(106)으로 보내질 수 있는지 결정한다. ID 값의 발생은 순화(circular) 4비트 카운터를 이용하여 수행된다.

4-비트 카운터의 이용은 3가지 점에서 중요하다.

첫 번째 중요한 점은 최대 9개의 명령어 셋트가 프리펫치 버퍼 유니트(260)에서 한번에 활성화 될 수 있다는 것인데, 그중 4개의 명령어 셋트는 MBUF(188)에, 2개의 명령어 셋트는 EBUF(192)에 2개의 명령어 셋트는 TBUF(190)에, 그리고 1개의 명령어 셋트는 흐름 관통 버스(194)를 통해 IDecode 유니트(262)에 직접 제공된다. 두 번째는 명령어 셋트가 각각 4개의 바이트로된 4개의 명령어를 포함한다는 것이다. 결과적으로, 펫치를 위해 명령어 셋트를 선택하는 어드레스중 최하위 4개의 비트는 여분이 된다. 마지막으로, 프리펫치 요청 ID 값이 프리펫치 요청 어드레스의 최하위 4비트로서 삽입에 의해 프리펫치 요청과 쉽게 연결될 수 있으며, 그래서 CCU(106)와의 인터페이스에 필요로 되는 어드레스 라인의 총 수효를 감소시킨다.

IFU(102)에 의해 발생된 프리펫치 요청의 순서에 관해 비순서(out-of-order) 적으로, 명령어 셋트가 CCU(106)에 의해 복귀될 수 있도록 하기 위해, 상기 구조(100)는 CCU(106)로부터의 명령어 세트의 복귀와 함께 ID 요청값의 복귀를 제공한다. 그러나, 비순서적 명령어 셋트 복귀 능력은 16개의 유일한 ID의 소모를 초래할 수도 있다. 추가적은 프리펫치가 요청되었지만 복귀되지 않은 명령어 셋트를 초래하는 비순서적으로 실행된 조건부 명령어의 조합은 ID 값의 잠재적인 재사용(re-use)을 유도할 수 있다. 그러므로 4비트 카운터가 바람직하게 유지되며, 더 이상의 명령어 세트 프리펫치 요청이 발생되지 않는데, 여기서 다음 ID 값은 프리펫치 버퍼(260)에 보류중인 아직 미해결의 펫치 요청이나 또는 다른 명령어 셋트와 관련된 값과 동일하게 된다.

프리펫치 제어 로직 유니트(266)은 MBUF(188), TBUF(190) 및 EBUF(192)내의 각각의 명령어 셋트 프리펫치 버퍼 위치에 논리적으로 대응하는 상태 저장 위치를 포함하는 프리펫치 상태 어레이(268)를 직접적으로 관리한다. 프리펫치 제어 로직 유니트(266)는 선택 및 데이터 라이(306)을 경유하여 상태 레지스터 어레이(268)로 판독 및 기록 데이터를 주사할 수 있다. 상기 어레이(268) 내에서, 메인 버퍼 레지스터(308)는 4개의 4-비트 ID 값(MBID), 4개의 단일 비트 예외 플래그(MB RES) 및 4개의 단일 비트 유효 플래그(MB VAL)의 저장을 제공하는데, 이것은 각각 논리적 비트-위치에 의해 MBUF(180) 내의 각각의 명령어 셋트 저장 위치에 대응한다. 비슷하게, 목표 버퍼 레지스퍼(310)와 확장 버퍼 레지스터(312)는 각각 2개의 4-비트 ID 값(TB ID, EB ID), 2개의 단일-비트 예비 플래그(TB RES, EB RES) 및, 2개의 단일-비트 유효 플래그(TB VAL, EB VAL)의 저장을 제공한다. 마지막으로, 흐름 관통 상태 레지스터(314)는 단일 4-비트 ID 값(FT ID), 단일 예비 플래그 비트(FT RES) 및, 단일 유효 플래그 비트(FT VAL)의 저장을 제공한다.

상태 레지스터 어레이(268)는 먼저 CCU(106)에 프리펫치 요청이 이루어질 때 주사가 이루어지고 프리펫치 제어 로직 유니트(266)에 의해 적절하게 갱신되며, 계속해서 명령어 셋트가 복귀될 때마다 주사 및 갱신된다. 특히, 제어 라인(316)을 통해 프리펫치 요청 신호를 수신했을 때, 프리펫치 제어 로직 유니트(216)는 현재 순환 카운터에 의해 발생된 ID 값을 증분시키고, ID 값이 이용가능한 것인지 또한 프리펫치 요청 신호에 의해 지정된 형태의 프리펫치 버퍼 위치가 이용가능한 것인지를 판단하기 위해 상태 레지스터 어레이(268)를 주사하고, CCU(106)가 프리펫치 요청을 받아 들일 수 있는지 판단하기 위해 CCU IBUSY 제어 라인(300)의 상태를 검사하고 만일 그렇다면 제어 라인(298) 상의 CCU IREAD 제어 신호를 표명(assert) 하고, CCU ID 출력 버스(294) 상의 증분된 ID 값을 CCU(106)로 보낸다. 프리펫치 저장 위치는 대응하는 예비 및 유효 상태 플래그가 거짓(false)인 경우에 이용 가능하다. 프리펫치 요청 ID 는 CCU(106)에의 요청의 배치와 동시에 MBUF(188), TBUF(190) 또는 EBUF(192) 내의 의도된 저장 위치에 대응하는 상태 레지스터 어레이(268) 내의 ID 저장 위치로 기록된다. 부가적으로 대응하는 예비 상태 플래그는 참(true)으로 셋트된다.

CCU(106)가 이전에 요청된 명령어 셋트를 IFU(102)로 복귀시킬 수 있을 때, CCU IREADY 신호가 제어 라인(302)상에 표명되고, 대응하는 명령어 셋트 ID가 CCU ID 제어 라인(296)상에 제공된다.

프리펫치 제어 로직 유니트(266)는 프리펫치 버퍼 유니트(260)내의 명령어 셋트의 의도된 목적지를 식별하기 위해 상태 레지스터 어레이(268) 내의 ID 값 및 예비 플래그를 주사한다. 만일 매치만이 가능하다. 일단 식별되면, 명령어 셋트는 프리펫치 버퍼 유니트(260)내의 적절한 위치로 버스(114)를 통해 기록되거나, 또는 흐름 관통 요청으로서 식별되는 경우에는 IDecode 유니트 (262)로 직접 제공된다. 어느 경우에도, 대응하는 상태 레지스터 어레이내의 유효 상태 플래그는 참(true)으로 셋트된다.

다음에 더 상세하게 설명되게 되는 바와 같이, PC 로직 유니트(270)는 IFU(102) 전체를 통해 MBUF(188), TBUF(190) 및 EBUF(192) 명령어 스트림의 가상 어드레스를 추적(track)한다. 이 기능을 수행하는데 있어서, PC 로직 블록(270)은 디코드 유니트9262)로부터 제어 및 동작한다. 특히, 프로그램 명령어 스트림 흐름에서의 변화에 잠재적으로 관계되어 있는 IDecode 유니트(262)에 의해 디코드된 명령어의 일부는 버스(318) 상에서 제어 흐름 검출 유니트(274)에 제공되고 또한, 직접 PC 로직 블록(270)에 제공된다. 제어 흐름 검출 유니트(274)는 조건부 및 무조건 분기 명령어, 호출 형태 명령어, 소프트웨어 트랩 절차 명령어 및 다양한 복귀 명령어를 포함하는 제어 흐름 명령어를 구성하는 디코드된 명령어 셋트내의 각각의 명령어를 식별한다. 제어 흐름 검출 유니트(274)는 IDecode 유니트(262)내에 존재하는 명령어 셋트내의 제어 흐름 명령어의 위치 및 특성을 식별하기 위해 제어 신호를 라인(322)을 통해 PC 로직 유니트(270)에 제공한다. PC 로직 유니트(270)는 다음에, 통상적으로 명령어 내에 제공되어 라인(318)을 통해 PC 로직 유니트로 전달된 데이터로부터 제어 흐름 명령어의 목표 어드레스를 판단한다. 예를 들어, 분기 로직 바이어스가 앞으로 조건부 분기 명령어를 실행하도록 선택된 경우에, PC 로직 유니트(270)는 조건부 분기 명령어 목표 어드레스로부터 명령어 셋트의 프리펫칭을 직접 및 분리적으로 추적하기 시작한다.

그러므로, 제어 라인(316) 상의 프리펫치 요청의 다음 표명(assertion)으로, PC 로직 유니트(270)는 이전의 프리펫치 명령어 셋트가 MBUF(188)나 또는 EBUF(192)로 향하는 것으로 가정하여, 프리펫치의 목적지가 TBUF(190)가 되도록 선택하는 제어신호는 라인(316)을 통해 표명하게 된다. 일단, 프리펫치 제어 로직 유니트(266)가 프리펫치 요청 CCU(106)로 공급될 수 있는 것으로 판단하면, 프리펫치 제어 로직 유니트(266)는 어드레스 라인(324)을 통해 직접 CCU(106)에 목표 어드레스의 페이지 오프셋 부분(CCU PADDR[13:4])의 제공을 가능하게 하기 위해 다시 제어 라인(316)을 통해 PC 로직 유니트(270)에 인에이블링 신호를 제공한다. 이와 동시에, PC 로직 유니트(270)는 새로운 가상-물리적(virtual to physical) 페이지 번역이 요구되는 경우에, VMU 요청 신호는 제어 라인(328)을 통해 제공하고, 물리 어드레스의 번역을 위해 어드레스 라인(326)을 통해 VMU(108)에 목표 어드레스의 가상 부분(VMU VADDR[31:14])을 제공한다. 페이지 번역이 요구되지 않는 경우에는, VMU(108)에 의한 연산이 요구되지 않는다. 이전의 번역 결과는 CCU(106)에 의한 즉시 이용을 위해 버스(122)에 결합된 출력 랫치에서 유지된다.

PC 로직 유니트(270)에 의해 요청된 가상-물리적 번역을 수행하는데 있어서 VMU(108)에서의 연산 에러는 VMU 예외 및 VMU 미스 제어 라인(332,334)을 통해 보고 된다. VMU 미스 제어 라인(334)은 번역 룩 어사이드(lookaside) 버퍼(TLB) 미스를 보고한다. VMU 예외라인(332)상의 VMU 예외 제어 신호는 모든 다른 예외에 대해서 제기된다. 두 경우 모두에 있어서, PC 로직 유니트는 명령어 스트림내의 현재 실행 지점을 저장하고 무조건 분기에 응답하여 에러 조건의 진단 및 처리를 위한 전용 예외 처리 루틴 명령어 스트림을 프리펫치함으로써 에러 조건을 처리한다. VMU 예외 및 미스 제어 신호는 마주치게되는 예외의 일반적 성질을 식별하여, PC 로직 유니트(270)가 대응하는 처리 루틴의 프리펫치 어드레스를 식별할 수 있도록 한다.

IFIFO 제어 로직 유니트(272)는 FIFIO 유니트(264)를 직접적으로 지원하기 위해 제공된다. 특히, PC 로직 유니트(270)는 명령어 셋트가 IDecode 유니트(262)로부터 입력 버스(198) 상에서 이용 가능하다는 것을 IFIFO 제어 로직 유니트(272)에 신호하기 위한 제어 신호를 제어 라인(336)을 통해 제공한다. IFIFO 제어 유니트(272)는 명령어 셋트의 수신을 위해 가장 깊은(deepest) 이용 가능한 마스터 레지스터(200,208,216,224)를 선택할 책임이 있다. 마스터 제어 레지스터(202,210,218,226)의 각각의 출력은 제어 버스(338)를 통해 IFIFO 제어 유니트(272)에 제공된다. 각각의 마스터 제어 레지스터에 의해 저장된 제어 비트는 2-비트 버퍼 어드레스(IF_B_X_ADR), 단일 스트림 인디케이터(indicator) 비트(IF_B_X_STRM) 및 단일 유효 비트(IF_B_X_VLD)를 포함한다. 2-비트 버퍼 어드레스는 대응하는 명령어 셋트내의 제 1 유효 명령어를 식별한다. 즉, CCU(106)에 의해 복귀되는 명령어 셋트는 예를들어 분기 연산의 목표 명령어가 명령어 셋트내의 초기 명령어 위치에 위치되도록 정렬되지 않을 수도 있다. 그러므로, 버퍼 어드레스 값은 실행을 위해 고려될 명령어 셋트내의 초기 명령어를 유일하게 식별하기 위해 제공된다.

스트림 비트는 근본적으로 조건부 제어 흐름 명령어를 포함하고 IFIFO 유니트(264)를 통하여 명령어 스트림에서의 잠재적인 제어 흐름 변화를 발생하는 명령어 셋트의 위치를 식별하기 위한 표시로서 사용된다. 메인 명령어 스트림은 일반적으로 0의 스트림 비트값으로 MBUF(188)를 통해 처리된다. 예를 들어, 상대적 조건부 분기 명령어의 발생시, 대응하는 명령어 셋트는 1의 스트림 비트 값으로 표시된다. 조건부 분기 명령어는 IDecode 유니트(262)에 의해 검출된다. 명령어 셋트에는 최대 4개의 조건부 제어 흐름 명령어가 존재할 수도 있다. 명령어 셋트는 다음에 IFIFO 유니트(264)의 가장 깊은 이용 가능한 마스터 레지스터에 저장된다.

조건부 분기 명령어의 목표 어드레스를 결정하기 위해 현재 IEU(104) 실행 지점 어드레스(DPC), 스트림 비트에 의해 식별된 바와 같은 명령어 셋트를 포함하는 조건부 명령어의 상대적 위치 및 제어 흐름 검출기(274)에 의해 제공되는 바와 같은

명령어 셋트내의 조건부 명령어 위치 오프셋이 제어 라인(318)을 통해 대응하는 분기 명령어 필드로부터 얻어지는 바와 같은 상대적 분기 오프셋 값과 조합된다. 이 결과는 PC 로직 유니트(270)에 의해 저장되는 분기 목표 가상 어드레스가 된다. 이 때, 목표 명령어 스트림의 초기 명령어 셋트는 상기 어드레스를 이용하는 TBUF(190)로 프리펫치될 수도 있다.

PC 로직 유니트(270)에 대해 선택된 사전선택 분기 바이어스에 따라, IFIFO 유니트(264)는 MBUF(188)나 또는 TBUF (190)중 하나로부터 계속해서 로드되게 된다. 하나 또는 그 이상의 조건부 분기 명령어를 포함하는 제 2 명령어 셋트와 조우(encountered) 하게 되면, 명령어 셋트는 0의 스트림 비트 값으로 표시된다. 제 2 목표 어드레스는 PC 로직 유니트 (270)에 의해 계산되어 저장되지만, 프리펫치는 수행되지 않는다. 부가적으로, 더 이상의 명령어 셋트가 IDecode 유니트 (262)를 통해 처리될 수 없으며, 또는 적어도 조건부 흐름 제어 명령어를 포함하는 것은 없다.

본 발명의 양호한 실시예에서 PC 로직 유니트(270)는 최대 2개의 명령어 셋트에서 발생하는 최대 8개의 조건부 흐름 명령어를 관리할 수 있다. 스트림 비트 변화에 의해 표시되는 2개의 명령어 셋트의 각각에 대한 목표 어드레스는 명령어 셋트 내의 대응하는 조건부 흐름 명령어의 위치에 관해 논리적으로 위치되는 각각의 목표 어드레스로 4개의 어드레스 레지스터로된 어레이에 저장된다. 제 1 정순서(in-order) 조건부 흐름 명령어의 분기 결과가 해결되면, PC 로직 유니트(270)는 분기가 취해진 경우에 MBUF(188)로 TBUF(190)의 내용을 전달하고, TBUF(190)의 내용을 무효 표시하기 위해 라인(316) 상의 제어 신호를 통해 프리펫치 제어 유니트(260)를 관리하게 된다. 부당한(incorrect) 명령어 스트림으로부터의 IFIFO 유니트(264) 내의 명령어 셋트와 분기가 이루어지지 않은 경우 목표 스트림과 분기가 이루어진 경우 메인 스트림은 IFIFO 유니트(264)로부터 클리어 된다. 만일 제 1 스트림 비트 표시된 명령어 셋트내에 제 2 또는 후속 조건부 흐름 제어 명령어가 존재한다면, 그 명령어는 일관된 방식으로 처리되는데, 즉, 목표 스트림으로부터의 명령어 셋트는 프리펫치되고, MBUF(188)나 TBUF(190)로부터의 명령어 셋트는 분기 바이어스에 따라 IDecode 유니트(262)를 통해 처리되고, IFIFO 유니트(264)는 조건부 흐름 명령어가 마지막으로 해결될 때 부당한 스트림 명령어 셋트로부터 클리어된다.

일단 IFIFO 유니트(264)가 부당한 스트림 명령어 셋트로부터 클리어되면, IFIFO 유니트(264)에 2차 조건부 흐름 명령어 셋트가 남아 있고, 제 1 조건부 흐름 명령어 셋트가 더 이상의 조건부 흐름 명령어를 포함하고 있지 않으며, 제 2 스트림 비트 표시 명령어 세트의 목표 어드레스는 어드레스 레지스터의 제 1 어레이로 진전된다. 어느 경우에도, 조건부 흐름 명령어를 포함하는 다음 명령어 셋트는 IDecode 유니트(262)를 통해 평가될 수 있다. 그러므로, 스트림 비트의 토글 용법은 분기 목표 어드레스를 계산하고, 분기 바이어스가 계속해서 특정 조건부 흐름 제어 명령어에 대해 부당한 것으로 결정된 경우에 클리어시키기 위해 명령어 세트 위치를 표시(marking) 하기 위한 목적으로 잠재적으로 제어 흐름 변화가 IFIFO 유니트(264)를 통해 표시되어 추적(tracked)될 수 있도록 허용한다.

실제적으로 마스터 레지스터로부터 명령어 셋트를 클리어시키는 것이 아니라, IFIFO 제어 로직 유니트(272)는 단순히 IFIFO 유니트(264)의 대응하는 마스터 레지스터의 제어 레지스터에 유효 비트 플래그를 리셋트시킨다. 클리어 연산은 라인9336)에 제공된 제어 신호에 PC 로직 유니트(270)에 의해 인스티게이트(instigated) 된다. 마스터 제어 레지스터 (202,210,218,226)의 가각의 입력은 상태 버스(230)를 통해 IFIFO 제어 로직 유니트 (272)에 의해 직접 억세스 가능하다. 본 발명의 양호한 구조(100)에서, 이들 마스터 제어 레지스터(202,210,218,226) 내의 비트는 IFIFO 유니트(264)에 의한 데이터 시프트 연산과 동시에 또한 무관하게 IFIFO 제어 유니트(272)에 의해 셋트될 수도 있다. 이 능력은 명령어 셋트가 마스터 레지스터(200,208,216,224) 중 어느 레지스터로 기록될 수 있도록 허용하고, 대응하는 상태 정보가 IEU (104)의 연산과 비동기식으로 마스터 제어 레지스터(202,210,218,226)로 기록될 수 있도록 허용한다.

마지막으로, 제어 및 상태 버스(230) 상의 부가적인 제어 라인을 IFIFO 유니트(264)의 FIFO 연산을 인에이블시키고 관리한다. IFIFO 시프트는 제어 라인(336)을 통해 PC 로직 유니트(270)에 의해 제공되는 시프트 요청 제어 신호에 응답하여 IFIFO 제어 로직 유니트(272)에 의해 수행된다. 명령어 셋트를 수신하기 위한 마스터 레지스터(200,208,216,224)의 유효성에 기반을 둔 IFIFO 제어 유니트(272)는 프피펫 버퍼(260)로부터의 다음의 적절한 명령어 셋트의 전달을 요청하기 위한 제어 신호를 라인(316)을 통해 프리펫치 제어 유니트(266)에 제공한다. 명령어 셋트의 전달시, 어레이(268)내의 대응하는 유효 비트는 리셋트된다.

C) IFU/IEU 제어 인터페이스

IFU(102)와 IEU(104) 사이의 제어 인터페이스는 제어 서브(126)에 의해 제공된다. 이 제어 버스(126)는 PC 로직 유니트 (270)에 결합되며, 다수의 제어 라인, 어드레스 라인 및 특수 데이터 라인으로 이루어진다. 제어 라인(340)을 통해 통과되는 인터럽트 요청 및 승인 제어신호는 IFU(102)로 하여금 IEU(104)에 신호하고 그 인터럽트 동작과 동기화시킬 수 있도록 한다. 외부에서 발생된 인터럽트 신호는 라인(292)을 통해 로직 유니트(270)에 제공된다. 이에 응답하여 라인(340)에 제공되는 인터럽트 요청 제어 신호는 IEU(104)로 하여금 시험적으로 실행된 명령어를 취소(cancel) 하도록 한다. 인터럽트의 특성에 관한 정보는 인터럽트 정보 라인(341)을 통해 교환된다. IEU(104)가 PC 로직 유니트(270)에 의해 결정되는

인터럽트 서비스 루틴 어드레스로부터 프리펫치된 명령어 셋트를 수신하기 시작한 준비가 되어 있을 때, IEU(104)는 라인(340) 상에 인터럽트 승인 제어 신호를 발한다. 그러면, IFU(102)에 의해 프리펫치된 바와 같은 인터럽트 서비스 루틴의 실행이 시작되게 된다.

IFIFO 판독(read)(IFIFO RD) 제어 신호는 가장 깊은 마스터 레지스터(224)에 존재하는 명령어 셋트가 완전하게 실행되었고 다음 명령어 셋트가 요구된다는 신호 표시를 하기 위해 IEU(104)에 의해 제공된다. 제어 신호를 수신하자마자, PC 로직 유니트(270)는 IFIFO 유니트(264)에 대한 IFIFO 시프트 연산을 수행하도록 IFIFO 제어 로직 유니트(272)에 지시한다.

PC 증분 요청 및 사이즈 값(size value)(PC INC/SIZE)을 명령어의 대응하는 사이즈 수만큼 현재 프로그램 카운터 값을 갱신하도록 PC 로직 유니트(270)에 지시하기 위해 제어 라인(344) 상에 제공된다. 이것은 PC 로직 유니트(270)로 하여금 현재 프로그램 명령어 스트림내의 제 1 정순서 실행 명령어의 위치에 정확한 실행 프로그램 카운터(DPC)의 지점을 유지할 수 있도록 한다.

목표 어드레스(TARGET ADDR)는 어드레스 라인(346) 상에서 PC 로직 유니트(270)로 복귀된다. 목표 어드레스는 IEU(104)의 레지스터 파일내에 저장된 데이터에 의존하는 분기 명령어의 가상 목표 어드레스이다. 그러므로, 목표 어드레스를 계산하기 위한 IEU(104)의 연산이 요구된다.

제어 흐름 결과(CF RESULT) 제어신호는 임의의 현재 보류중인 조건부 분기 명령어가 해결되었는지 또한 결과가 분기를 취할 것인지 아닌지를 식별하기 위해 제어 라인(348) 상에서 PC 로직 유니트(270)에 제공된다. 이들 제어 신호에 근거하여, PC 로직 유니트(270)는 프리펫치 버퍼(260)와 IFIFO 유니트(264)내의 명령어 셋트중 조건부 흐름 명령어 실행의 결과로서 취소되어야 할 명령어 셋트를 결정할 수 있다.

IEU(104)에 의한 일정 명령어의 실행에 대해 IFU(102)에 경보(alert) 하기 위해 다수의 IEU 명령어 복귀 형태 제어신호(IEU Retrun)가 제어 라인(350) 상에 제공된다. 이들 명령어는 절차 명령어로부터의 복귀, 트랩으로부터의 복귀 및 서브루틴 호출로부터의 복귀를 포함한다. 트랩으로부터의 복귀 명령어는 하드웨어 인터럽트 및 소프트웨어 트랩 처리 루틴에서 동일하게 이용된다. 서브루틴 호출 복귀 명령어도 또한 점프-링크 형태 호출과 함께 이용된다. 각 경우에, 이전에 인터럽트된 명령어 스트림에 관한 그 명령어 펫치 연산을 재개시하도록 IFU(102)에 알리기 위한 복귀 제어 신호가 제공된다. IEU(104)로부터의 이들 신호의 발생은 시스템(100)의 정확한 연산이 유지될 수 있도록 하여, "인터럽트된" 명령어 스트림의 재개시(resumption)가 복귀 명령어의 실행 지점에서 수행될 수 있도록 한다.

현재 명령어 실행 PC 어드레스(Current IF-PC)는 어드레스 버스(352)상에서 IEU(104)에 제공된다. 이 어드레스 값, DPC는 IEU(104)에 의해 실행되고 있는 정확한 명령어를 식별한다.

즉, IEU(104)는 현재 IF-PC 어드레스를 지난 명령어를 앞서서 시험적으로 실행할 수도 있지만, 이 어드레스는 인터럽트, 예외 및, 정확한 머신 스테이트를 알기를 원하는 다른 사건(event)의 발생에 관해 본 구조(100)의 정확한 제어를 위한 목적으로 유지되어야 한다. 현재 실행중인 명령어 스트림에서 정확한 머신 스테이트로 진행될 수 있는 것으로 IEU(104)가 판단 했을 때, PC Inc/Size 신호가 IFU(102)에 제공되어, 현재 IF-PC 어드레스 값에 바로 반영된다.

마지막으로, 특수 레지스터 데이터의 전달을 위해 어드레스 및 양방향 데이터 버스(354)가 제공된다. 이 데이터는 IEU(104)에 의해 IFU(102)내의 특수 레지스터에 프로그램되어, 이 레지스터로부터 판독될 수 있다. 특수 레지스터 데이터는 일반적으로 IFU(102)에 의한 이용을 위해 IEU(104)에 의해 로드되어 계산된다.

D) PC 로직 유니트 상세:

제3도에는 PC 제어 유니트(362), 인터럽트 제어 유니트(373), 프리펫치 PC 제어 유니트(364) 및 실행 PC 제어 유니트(366)를 포함하는 PC 로직 유니트(270)의 상세도가 도시되어 있다. PC 제어 유니트(362)는 프리펫치 제어 로직 유니트(266), IFIFO 제어 로직 유니트(272) 및 IEU(104)로부터 인터페이스 버스(126)를 통해 제공되는 제어 신호에 응답하여 프리펫치 및 실행 PC 제어 유니트(364,366)에 대한 타이밍 제어를 제공한다. 인터럽트 제어 유니트(363)는 트랩의 각 형태를 처리하기 위한 적절한 처리 루틴을 선택하는 프리펫치 트랩 어드레스 오프셋의 판단을 포함하여 언터럽트 및 예외의 정확한 처리를 관리할 책임이 있다. 프리펫치 PC 제어 유니트(364)는 특히, 트랩 처리 및 절차 루틴 명령어 흐름에 대한 복귀 어드레스를 저장하는 것을 포함하여 프리펫치 버퍼(188,190,192)를 지원하는데 필요한 프로그램 카운터를 관리할 책임이 있다. 이와같은 연산의 지원으로, 프리펫치 PC 제어 유니트(364)는 물리적 어드레스 버스 라인(324) 상의 CCM PADDER 어드레스와 어드레스 라인(326)상의 VMU VMADDR 어드레스를 포함하는 프리펫치 가상 어드레스를 발생할 책임이 있다. 결과적으로, 프리펫치 PC 제어 유니트(364)는 현재의 프리펫치 PC 가상 어드레스 값을 유지할 책임이 있다.

프리펫치 연산은 일반적으로 제어 라인(316)에 제공되는 제어 신호를 통해 IFIFO 제어 로직 유니트(272)에 의해 갱신된다. 이에 응답하여, PC 제어 유니트(362)는 PADDR 어드레스와 필요한 경우에 어드레스 라인(324,326) 상의 VMADDR 어드레스를 발생하도록 프리펫치 PC 제어 유니트(364)를 동작시키기 위해 제어 라인(372)상에 제공되는 다수의 제어신호를 발생한다. 또한, PC 제어 유니트(362)가 현재의 프리펫치 어드레스에서 명령어 셋트 펫치를 재실행하고 있는지, 일련의 프리펫치 요청에서의 두 번째 요청에 대한 정렬(aligning) 중인지, 또는 프리펫치를 위한 다음의 정식 순차 명령어를 선택하고 있는지에 따라 제어 라인(374) 상에 0 내지 4의 값을 가진 증가 신호가 제공될 수도 있다. 마지막으로, 현재의 프리펫치 어드레스 PF_PC가 버스(370)상에서 실행 PC 제어 유니트(366)에 제공된다.

새로운 프리펫치 어드레스는 다수의 소스(sources)로부터 기원한다. 어드레스의 1차적인 소스는 버스(352)를 통해 실행 PC 제어 유니트(366)으로부터 제공되는 현재 IF_PC 어드레스이다. IF_PC 어드레스는 주로, 초기 호출, 트랩 또는 절차 명령어가 발생할 때 프리펫치 PC 제어 유니트(364)에 의한 후속 이용을 위한 복귀 어드레스를 제공한다. IF_PC 어드레스는 이들 각각의 명령어의 발생시 프리펫치 PC 제어 유니트(364)내의 레지스터에 저장된다. 이런 방식으로, PC 제어 유니트(362)는 제어 라인(350)을 통해 IEU 복귀 신호를 수신했을 때, 단지 새로운 프리펫치 가상 어드레스의 소스를 위해 프리펫치 PC 제어 유니트(364) 내의 대응하는 복귀 어드레스 레지스터를 선택함으로써, 원래의 프로그램 명령어 스트림을 재개시할 필요가 있다.

프리펫치 어드레스의 다른 소스는 실행 PC 제어 유니트(366)로부터 상태 목표 어드레스 버스(382) 및 IEU(104)로부터 제공되는 절대 목표 어드레스 버스(346) 상에 제공되는 목표 어드레스 값이다. 상대 목표 어드레스는 실행 PC 제어 유니트(366)에 의해 직접 계산될 수 있는 것들이다. 절대 목표 어드레스는 IEU 레지스터 파일에 포함된 데이터에 의존하기 때문에 IEU(104)에 의해 발생되어야 한다. 목표 어드레스는 프리펫치 가상 어드레스로서 사용하기 위해 목표 어드레스 버스(384)를 통해 프리펫치 PC 제어 유니트(364)로 루팅된다. 상대 목표 어드레스를 계산하는데 있어서, IDecode 유니트(262)로부터 버스(318)의 오퍼랜드 변위(operand displacement) 부분에는 또한 대응하는 분기 명령어의 오퍼랜드 부분이 제공된다.

프리펫치 가상 어드레스의 다른 소스는 실행 PC 제어 유니트(366)이다. 복귀 어드레스 버스(352)는 현재 IF_PC 값(DPC)을 프리펫치 PC 제어 유니트(364)에 전달하기 위해 제공된 것이다. 이 어드레스는 인터럽트, 트랩, 또는 호출과 같은 다른 제어흐름 명령어가 명령어 스트림내에서 발생된 경우에 복귀 어드레스로서 이용된다. 이 때, 프리펫치 PC 제어 유니트(364)는 새로운 명령어 스트림을 언제라도 프리펫치 할 수 있도록 허용한다. PC 제어 유니트(362)는 일단 대응하는 인터럽트 또는 트랩 처리 루틴 또는 서브루틴이 실행되면, IEU(104)로부터 라인(350)을 통해 IEU 복귀 신호를 수신한다. 다음에, PC 제어 유니트(362)는 라인(350)을 통해 제공되는 바와 같은 실행된 복귀 명령어의 식별에 근거하여 라인(372)상의 PFPC 제어 신호중 한 신호를 통해 현재 복귀 가상 어드레스를 포함하는 레지스터를 선택한다. 이 때 이 어드레스는 PC 로직 유니트(270)에 의해 프리펫치 동작을 진행하는데 사용된다.

마지막으로, 프리펫치 가상 어드렛의 다른 소스는 특수 레지스터 어드레스 및 데이터 버스(354)로부터 나온다. IEU(104)에 의해 계산되거나 로드된 어드레스 값 또는 적어도 기본 어드레스 값이 버스(354)를 통해 프리펫치 PC 제어 유니트(364)로 데이터로서 전달된다. 기본(base) 어드레스는 트랩 어드레스 테이블, 고속 트랩 테이블 및 기본 절차 명령어 디스패치 테이블 및 기본 절차 명령어 디스패치 테이블에 대한 기본 어드레스를 포함한다. 버스(354)는 머신 스테이트의 대응하는 애스팩트(aspects)가 IEU(104)를 통해 조작될 수 있도록 프리펫치 및 실행 PC 제어 유니트(364,366) 내의 래지스터가 판독되도록 허용한다.

PC 제어 유니트(362)의 제어를 받는 실행 PC 제어 유니트(366)는 1차적으로 현재 IF_PC 어드레스 값을 계산할 책임이 있다. 이런 역할에서, 실행 PC 제어 유니트(366)는 E_XPC 제어 라인(378)상의 PC 제어 유니트(362)에 의해 제공되는 제어신호와, IF_PC 어드레스를 조정하기 위해 제어 라인(380)상에 제공되는 증분/사이즈 제어신호에 응답한다. 이들 제어 신호는 1차적으로 라인(342)상에 제공되는 IFIFO 판독 제어 신호와 IEU(104)로부터 제어 라인(344)상에 제공되는 PC 증분/사이즈 값에 응답하여 발생된다.

1) PF 및 E_X PC 제어/데이터 유니트 상세:

제4도는 프리펫치 및 실행 PC 제어 유니트(364,366)의 상세 블록도이다. 이들 유니트는 기본적으로 레지스터, 증분기, 선택기 및 가산기 블록으로 구성된다. 이들 블록 사이의 데이터 전달을 관리하기 위한 제어는 PFPC 제어 라인(372), ExPC

제어 라인(378) 및 증분 제어 라인(374,380)을 통해 PC 제어 유니트(362)에 의해 제공된다. 명료성을 위해, 이들 특정 제어 라인은 제4도의 블록도에는 도시되지 않는다. 그러나, 이들 제어 신호가 본 명세서에 설명된 바와 같이 도시된 블록에 제공된다는 것을 이해될 수 있다.

프리펫치 PC 제어 유니트(364)의 중심이 되는 것은 현재 프리펫치 가상 어드레스의 중앙 선택기로서 작용하는 프리펫치 선택기(PF_PC SEL)(390)이다. 이 현재 프리펫치 어드레스는 다음 프리펫치 어드레스를 발생하기 위해 프리펫치 선택기로부터 증분기(incrementor) 유니트(394)까지의 출력 버스(392)에 제공된다. 상기 다음 프리펫치 어드레스는 레지스터 MBUF PFnPC(398), TBUF PFnPC(400) 및 EBUF PFnPC(402)의 병렬 어레이로 가는 증분기 출력 1버스(396) 상에 제공된다. 이들 레지스터(398,400,402)는 다음 명령어 프리펫치 어드레스를 효과적으로 저장한다. 그러나, 본 발명의 양호한 실시예에 따라, 개별 프리펫치 어드레스는 MBUF(188), TBUF(190) 및 EBUF(192)를 위해 홀드된다. MBUF, TBUF 및 EBUF PFnPC 레지스터(398,400,402)에 의해 저장된 바와 같은 프리펫치 어드레스는 각각 어드레스 버스(404,408,410)에 의해 프리펫치 선택기(390)에 제공된다. 그러므로 PC 제어 유니트(362)는 단지 프리펫치 레지스터(398,400,402) 중 다른 한 레지스터의 프리펫치 선택기(390)에 의한 선택을 지시함으로써 프리펫치 명령어 스트림의 즉시 스위치를 지시할 수 있다. 어드레스 값이 증분기(394)에 의해 증분된 후, 스트림내의 다음 명령어 셋트가 프리펫치될 것이면, 그 값은 프리펫치 레지스터(398,400,402) 중 적당한 레지스터로 복귀된다. 간략성을 위해 단일 특수 레지스터 블록(412)으로서 도시된 다른 병렬 레지스터 어레이는 다수의 특수 어드레스를 저장하기 위해 제공된다. 이 레지스터 블록(412)은 트랩 복귀 어드레스 레지스터, 절차 명령어 복귀 어드레스 레지스터, 절차 명령어 디스패치 테이블 기본 어드레스 레지스터, 트랩 루틴 디스패치 테이블 기본 어드레스 레지스터 및, 고속 트랩 루틴 테이블 기본 어드레스 레지스터를 포함하고 있다. PC 제어 유니트(362)의 제어하에서, 이들 복귀 어드레스 레지스터는 버스(352')를 통해 현재 IF_PC 실행 어드레스를 수신할 수도 있다. 레지스터 블록(412) 내의 복귀 및 기본 어드레스 레지스터에 의해 저장된 어드레스 값은 IEU(104)에 의해 독자적으로 판독 및 기록될 수도 있다. 특수 레지스터 어드레스 및 데이터 버스(354)를 통해 레지스터가 선택되고 그 값이 전달된다.

PC 제어 유니트(362)에 으해 제어되는 특수 레지스터 블록(412) 내의 선택기는 레지스터 블록(412)의 레지스터에 의해 저장된 어드레스가 특수 레지스터 출력 버스(416)에 실려 프리펫치 선택기(390)로 전달될 수 있도록 한다. 복귀 어드레스는 프리펫치 선택기(390)에 직접 제공된다. 기본 어드레스 값은 인터럽트 제어 유니트(363)으로부터 인터럽트 오프셋 버스(373) 상에 제공되는 오프셋 값과 조합된다. 일단 버스(373')를 통해 프리펫치 선택기(390)에 대한 소스로 되면, 특수 어드레스는 이후 증분기(394)와 프리펫치 레지스터(398,400,402) 중 한 레지스터를 통해 어드레스의 증분 루프를 연장함으로써 새로운 프리펫치 명령어 스트림에 대한 초기 어드레스로서 사용될 수 있다.

프리펫치(390)에 대한 어드레스의 다른 소스는 목표 어드레스 레지스터 블록(414) 내의 레지스터 어레이이다. 이 블록(414) 내의 목표 레지스터는 본 발명의 양호한 실시예에서 8개의 잠재적인 분기 목표 어드레스의 저장을 제공한다. 이들 8개의 저장 위치는 IFIFO 유니트(264)의 가장 낮은 2개의 마스터 레지스터(216,224)에 홀드된 8개의 잠재적으로 실행 가능한 명령어에 논리적으로 대응한다. 이들 명령어중 몇몇 명령어 및 잠재적으로는 모든 명령어가 조건부 분기 명령어가 될 수 있기 때문에, 목표 레지스터 블록(414)은 그 사전 계산된 목표 어드레스가 TBUF(190)을 통한 목표 명령어 스트림의 펫치를 위한 이용에 대기하여 저장될 수 있도록 허용한다. 특히, 만일 PC 제어 유니트(362)가 목표 명령어 스트림의 프리펫치를 즉시 시작하도록 조건부 분기 바이어스가 셋트되면, 목표 어드레스는 목표 레지스터 블록(414)에 의해 어드레스 버스(418)를 통해 프리펫치 선택기(390)에 즉시 공급된다. 일단 증분기(394)에 의해 증분되면, 어드레스는 다시 목표 명령어 스트림의 후속 프리펫치 연산에 사용하기 위해 TBUF PFnPC(400)에 저장된다. 만일 추가 분기 명령어가 목표 명령어 스트림내에서 발생하면, 이와 같은 2차적인 분기의 목표 어드레스는 제 1 조건부 분기 명령어의 해결시 이용을 보류하는 목표 레지스터 어레이(414)에서 계산되어 저장된다. 목표 레지스터 블록(414)에 의해 저장된 바와 같은 계산된 목표 어드레스는 어드레스 라인(382)를 통해 실행 PC 유니트(366) 내의 목표 어드레스 계산 유니트로부터 또는 절대 목표 어드레스 버스(346)를 토해 IEU(104)로부터 전달된다.

프리펫치 PF_PC 선택기(390)로부터 전달된 어드레스 값은 완전한 32-비트 가상 어드레스 값이다. 본 발명의 양호한 실시예에서 페이지 사이즈는 최대 페이지 오프셋 어드레스 값[13:0]에 대응하는 16KBytes로 고정된다. 그러므로, 현재 프리펫치 가상 페이지 어드레스[27:14]에서의 변화가 존재하지 않는한 VMU 페이지 번역은 요구되지 않는다. 프리펫치 선택기(390) 내의 비교기는 이 상황을 검출한다. 페이지 경계를 지나는 증분이나 또는 다른 페이지 어드레스로의 제어 흐름 분기로 인해 가상 페이지 어드레스에서 변화가 존재할 때에는 VMU 번역 요청 신호(VMXLAT)가 라인(372')을 통해 PC 제어 유니트(362)로 제공된다. 다음에, PC 제어 유니트(362)는 VMU 가상-물리적 페이지 번역을 얻기 위해, 버퍼 유니트(420)을 통해 라인(324) 상의 CCU PADDR 어드레스에 더하여 라인(326) 상의 VM VADDR 어드레스와 VMU 제어 라인(326,328,330) 상의 적절한 제어신호의 배치를 지시한다. 페이지 번역이 요구되지 않는 경우에, 버스(122) 상의 VMU 유니트(108)의 출력에서 랫치에 의해 현재 물리적 페이지 어드레스[31:14]가 유지된다.

버스(370) 상에 제공된 가상 어드레스 증분 제어 라인(374)상에 제공된 신호에 응답하여 증분기(394)에 의해 증분된다. 증분기(394)는 다음 명령어 셋트를 선택하기 위해 명령어 셋트(4개의 명령어 또는 16바이트)를 나타내는 값만큼 증분한다. CCU 유니트(106)에 제공되는 바와 같은 프리펫치 어드레스의 저순위 4-비트는 0이다. 그러므로, 제 1 분기 목표 명령어 셋트내의 실제 목표 어드레스 명령어는 제 1 명령어 위치에 위치되지 않을 수도 있다. 그러나, 어드레스의 저순위 4-비트는 제 1 분기 명령어 위치를 IFU(102)가 알수 있도록 허용하기 위해 PC 제어 유니트(362)에 제공된다. 비정렬 목표 명령어 셋트에서의 실행을 위한 적당한 제 1 명령어를 선택하기 위해 2-비트 버퍼 어드레스로서의 목표 어드레스의 저순위 비트[3:2]를 복귀함으로써, 새로운 명령어 스트림의 제 1 프리펫치, 즉 명령어 스트림내의 제 1 비-순차적 명령어 셋트에 대해서만 검출 및 처리가 수행된다. 명령어 셋트내의 제 1 명령어의 어드레스와 명령어 셋트를 프리펫치하는데 사용되는 프리펫치 어드레스 사이의 비-정렬 관계는 현재 순차적 명령어 스트림의 지속기간동안 무시될 수 있다.

제4도에 도시한 기능 블록도의 잔여부는 실행 PC 제어 유니트(366)를 포함하고 있다. 본 발명의 양호한 실시예에 따라, 실행 PC 제어 유니트(366)는 그 고유의 독자적으로 작용하는 프로그램 카운터 증분기를 포함하고 있다. 이 기능의 중심부는 실행 선택기(DPC SEL)(430)이다. 어드레스 버스(352')상에 실행 선택기(430)의 출력된 어드레스는 상기 구조(100)의 현재 실행 어드레스(DPC)이다. 이 실행 어드레스는 가산기 유니트(434)에 제공된다. 라인(380)상에 제공되는 증분/사이즈 제어 신호는 가산기 유니트(434)가 선택기(430)로부터 얻어지는 어드레스에 가산하는 1 내지 4의 명령어 증분값이다. 가산기(432)가 추자적으로 출력 랫치 기능을 수행할 때, 증분된 다음 실행 어드레스는 어드레스 라인(436)상에서 다음 실행 증분 사이클에 사용하기 위해 다시 실행 선택기(430)에 직접 제공된다.

초기 실행 어드레스 및 모든 후속의 새로운 스트림 어드레스는 어드레스 라인(440)을 경우하여 새로운 스트림 레지스터 유니트(438)를 통해 얻어진다. 새로운 스트림 레지스터 유니트(438)는 프리펫치 선택기(390)으로부터 PFPC 어드레스 버스(370)상에 제공되는 바와 같은 새로운 현재 프리펫치 어드레스가 직접 어드레스 버스(440)로 통과 될 수 있도록 하거나 또는 후속 이용을 위해 저장될 수 있도록 한다. 즉, 프리펫치 PC 제어 유니트(364)가 새로운 가상 어드레스에서의 프리펫치를 시작하도록 결정하는 경우에, 새로운 스트림 어드레스는 새로운 스트림 레지스터 유니트(438)에 의해 임시적으로 저장된다. PC 제어 유니트(362)는 프리펫치 및 실행 증분 사이클 모두에서의 참여에 의해 실행 어드레스가 새로운 명령어 스트림을 유발하는 제어 흐름 명령어에 대응하는 프로그램 실행 지점에 도달할 때까지 새로운 스트림 레지스터 유니트(438)에 새로운 스트림 어드레스를 유지한다. 새로운 스트림 어드레스는 이 때 새로운 명령어 스트림내의 실행 어드레스의 독자적 발생을 개시하기 위해 새로운 스트림 레지스터 유니트(438)로부터 실행 선택기(430)로 출력한다.

본 발명의 양호한 실시예에 따라, 새로운 스트림 레지스터 유니트(438)는 2개의 제어 흐름 명령어 목표 어드레스의 버퍼링을 제공한다. 새로운 스트림 어드레스의 즉시 유효성에 의해, 근본적으로 실행 어드레스이 현재 시퀀스의 발생으로부터 실행 어드레스의 새로운 스트림 시퀀스까지의 실행 PC 제어 유니트(366)의 스위칭에 있어서 대기시간이 존재하지 않는다.

마지막으로, IF_PC 선택기(IF_PC SEL)(442)는 궁극적으로 IEU(104)로 가는 어드레스 버스(352)상에서 현재 IF_PC 어드레스를 발생하기 위해 제공된다. IF_PC 선택기(442)로의 입력은 실행 선택기(430)나 또는 새로운 스트림 레지스터 유니트(438)로부터 얻어지는 출력 어드레스이다. 대개의 경우에, IF_PC 선택기(442)는 실행 선택기(430)에 의해 출력되는 실행 어드레스를 선택하도록 PC 제어 유니트(362)에 의해 명령을 받는다. 그러나, 새로운 명령어 스트림의 실행을 개시하는데 사용되는 새로운 가상 어드레스로 스위칭하는데 있어서의 대기시간을 더욱 감소시키기 위해, 새로운 스트림 레지스터 유니트(438)로부터 제공되는 선택된 어드레스는 현재 IF_PC 실행 어드레스로서의 제공을 위해 버스(440)를 통해 직접 IF_PC 선택기(442)로 바이패스될 수 있다.

실행 PC 제어 유니트(366)는 모든 상대 분기 목표 어드레스를 계산할 수 있다. 현재 실행 지점 어드레스 및 새로운 스트림 레지스터 유니트(438) 제공 어드레스는 어드레스 버스(352',440)를 통해 제어 흐름 선택기(CF_PC)(446)에 의해 수신된다. 결과적으로, PC 제어 유니트(362)는 목표 어드레스를 계산하기 위해 정확한 초기 어드레스를 선택하는데 있어 실질적인 유연성(flexibility)을 갖고 있다. 이와 같은 초기 또는 기본 어드레스는 어드레스 버스(454)를 통해 목표 어드레스 ALU(450)에 제공된다. 목표 어드레스 ALU(450)로의 제 2 입력값은 버스(458)를 통해 제어 흐름 변위 계산 유니트(452)로부터 제공된다. 본 발명의 양호한 구조(100)에 따라, 상대 분기 명령어는 상대 새로운 목표 어드레스를 지정하는 즉시 모드 상수의 형태로 변위 값을 포함한다. 제어 흐름 변위 계산 유니트(452)는 IDecode 유니트 오퍼랜드 출력 버스(318)를 통해 초기에 얻어지는 오퍼랜드 변위 값을 수신한다. 마지막으로, 오프셋 레지스터 값은 라인(456)을 통해 목표 어드레스 ALU(450)에 제공된다. 오프셋 레지스터(448)는 PC 제어 유니트(362)로부터 제어 라인(378')을 통해 오프셋 값을 수신한다. 오프셋 값의 크기는 어드레스 라인(454)상에 제공되는 기본 어드레스와 상태 목표 어드레스가 계산되고 있는 현재 분기 명령어의 어드레스 사이의 어드레스 오프셋에 근거하여 PC 제어 유니트(362)에 의해 결정된다. 즉, IFIFO 제어 로직 유니

트(272)의 제어를 통해 PC 제어 유니트(362)는 현재 실행 지점 어드레스 (CP_PC에 의해 요청됨)에서의 명령어와 IDecode 유니트(262)에 의해 현재 처리되고 있는, 즉 그 명령어의 목표 어드레스를 결정하기 위해 PC 로직 유니트(270)에 의해 처리되고 있는 명령어를 구별하는 명령어 수를 추적한다.

상대 목표 어드레스가 목표 어드레스 ALU(450)에 의해 계산되면, 그 목표 어드레스는 어드레스 버스(382)를 통해 목표 레지스터(414)중 대응하는 레지스터로 기록된다.

2) PC 제어 알고리즘 상세:

1. 메인 명령어 스트림 처리: MBUF PF_nPC

1.1 다음 메인 흐름 프리펫치 명령어의 어드레스는 MBUF PF_nPC에 저장된다.

1.2 제어 흐름 명령어가 없는 경우에, 32비트 증분기는 각각 프리펫치 사이클에 대해 16바이트 만큼($_X16$) MBUF PF_nPC 내의 어드레스 값을 조정한다.

1.3 무조건 제어 흐름 명령어가 디코드될 때, 그 명령어 셋트 다음에 펫치되는 모든 프리펫치된 데이터는 플러쉬(flushed)되고 MBUF PF_nPC는 새로운 메인 명령어 스트림 어드레스로 목표 레지스터 유니트, PF_PC 선택기 및 증분기를 통해 로드되게 된다. 새로운 어드레스는 또한 새로운 스트림 레지스터에 저장된다.

1.3.1 상태 무조건 제어 흐름의 목표 어드레스는 제어 흐름 명령어 다음에 오는 오퍼랜드 데이터와 IFU에 의해 유지되는 레지스터 데이터로부터 IFU에 의해 계산된다.

1.3.2 절대 무조건 제어 흐름 명령어의 목표 어드레스는 결국 레지스터 참조 값, 기본 레지스터 값 및 인덱스 레지스터 값으로부터 IEU에 의해 계산된다.

1.3.2.1 절대 어드레스 제어 흐름 명령어에 대해 IEU에 의해 목표 어드레스가 복귀될때까지 명령어 프리펫치 사이클링이 스톨된다. 명령어 실행 사이클링은 계속된다.

1.4 무조건 제어 흐름 명령어로부터 생기는 다음 메인 흐름 프리펫치 명령어의 어드레스는 목표 어드레스 레지스터 유니트, PF_PC 선택기 및 증분기를 통해 바이패스되고, MBUF PF_nPC에서의 궁극적인 저장을 위해 루팅된다; 프리펫치는 1.2에서 계속된다.

2. 절차 명령어 스트림 처리; EBUF PF_nPC

2.1 메인 또는 분기 목표 명령어 스트림에서 절차 명령어가 프리펫치될 수도 있다. 목표 스트림에서 펫치되면 조건부 제어 흐름 명령어가 해결되고 절차 명령어가 MBUF로 전달될때까지 절차 스트림의 프리펫치를 스톨한다. 이것은 절차 명령어 스트림에서 발생하는 조건부 제어 흐름의 TBUF가 이용될 수 있도록 한다.

2.1.1 절차 명령어가 절차 명령어 스트림에 나타나지 말아야 한다. 즉, 절차 명령어가 내포(nested) 되지 않아야 된다: 절차 명령어로부터의 복귀는 메인 명령어 흐름으로 실행을 복귀시키게 된다. 내포를 허용하기 위해, 내포된 절차 명령어로부터의 추가의 전용 복귀가 요구되게 된다. 상기 구조는 이와 같은 명령어를 쉽게 지원할 수 있지만, 내포된 절차 명령어 능력에 대한 필요성이 상기 구조의 성능을 향상시키지 않게 된다.

2.1.2 메인 명령어 스트림에서, 제 1 및 제 2 조건부 제어 흐름 명령어를 내포하는 명령어 셋트를 포함하는 절차 명령어 스트림은 상기 제 1 명령어 셋트내의 조건부 제어 흐름 명령어가 해결되고 제 2 조건부 제어 흐름 명령어 셋트가 MBUF로 전달될때까지 제 2 조건부 제어 흐름 명령어에 대한 프리펫치를 스톨한다.

2.2 절차 명령어는 절차 루틴 개시 어드레스를 식별하기 위해 명령어의 즉시 모드 오퍼렌드 필드로서 포함되는 상대 오프셋을 제공한다:

2.2.1 절차 명령어에 의해 제공되는 오프셋 값은 IFU에서 유지되는 절차 기본 어드레스(PBR) 레지스터에 포함된 값과 조합된다. 이 PBR 레지스터는 특수 레지스터 이동 명령어의 실행에 응답하여 특수 어드레스 및 데이터 버스를 통해 판독 및 기록 가능하다.

2.3 절차 명령어와 조우할 때, 다음의 메인 명령어 스트림 IF_PC 어드레스 $_{U}$PC 리턴 어드레스 레지스터에 저장되고 프로세서 상태 레지스터(PSR) 내의 진행중 절차(procedure-in-progress) 비트는 셋트된다.

2.4 절차 스트림의 개시 어드레스는 PBR 레지스터(절차 명령어 오퍼랜드 오프셋 값을 더함)로부터 PF_PC 선택기로 루팅된다.

2.5 절차 스트림의 개시 어드레스가 새로운 스트림 레지스터 유니트와 증분($_{X}$16)을 위한 증분기로 동시에 제공된다;

증분된 어드레스는 EBUF PF$_{n}$PC에 저장된다.

2.6 제어흐름 명령어가 없는 경우에, 32 비트 증분기는 각각의 절차 명령어 프리펫치 사이클에 대해 EBUF PF$_{n}$PC 내의 어드레스 값($_{X}$16)을 조정한다.

2.7 무조건 제어 흐름 명령어가 디코드(IDecode) 될 때, 분기 명령어 다음에 펫치되는 모든 프리펫치된 데이터는 플러쉬 도고 EBUF PF$_{n}$PC 는 새로운 절차 명령어 스트림 어드레스로 로드된다.

2.7.1 상대 무조건 제어 흐름 명령어의 목표 어드레스는 IFU에서 유지되는 레지스터 데이터와 제어 흐름 명령어의 즉시 모드 오퍼랜드 필드내에 제공되는 오퍼랜드 데이터로부터 IFU에 의해 계산된다.

2.7.2 절대 무조건 분기의 목표 어드레스는 레지스터 참조 값, 기본 레지스터 값 및, 인덱스 레지스터 값으로부터 IEU에 의해 계산된다.

2.7.2.1 목표 어드레스가 절대 어드레스 분기에 대해 IEU에 의해 복귀될때까지 명령어 프리펫치 사이클링 스톨된다; 실행 사이클링은 계속된다.

2.8 다음 절차 흐름 프리펫치 명령어 셋트의 어드레스는 EBUF PFnPC 저장되고 프리펫치는 1.2에서 계속된다.

2.9 절차 명령어로부터의 복귀가 디코드(IDecode)될 때, 프리펫치는 $_{U}$PC 레지스터에 저장된 어드레스로부터 계속 되는데, 이 어드레스는 증분되어($_{X}$16), 후속 프리펫치를 위해 MBUF PFnPC로 복귀된다.

3. 분기 명령어 스트림 처리: TBUF PFnPC

3.1 MBUF 명령어 스트림내의 제 1 명령어 셋트에서 발생하는 조건부 제어 흐름 명령어가 디코드(IDecoded) 될 때, 목표 어드레스는 그 목표 어드레스가 현재 어드레스에 관한 것일 경우에는 IFU에 의해 결정되고 절대 어드레스에 대해서는 IEU에 의해 결정된다.

3.2 "바이어스를 취한 브랜치"에 대해서:

3.2.1 만일 분기가 절대 어드레스에 대한 것이면, 목표 어드레스가 IEU에 의해 복귀될때까지 명령어 프리펫치 사이클링을 스톨한다; 실행 사이클링은 계속된다.

3.2.2 PF_PC 선택기 및 증분기를 통한 전달에 의해 분기 목표 어드레스로 TBUF PF$_{n}$PC를 로드한다.

3.2.3 목표 명령어 스트림 명령어들이 TBUF로 프리펫치되고 후속 실행을 위해 IFIFO로 루팅된다; 만일 IFIFO 및 TBUF가 가득찬 상태가 되면 프리펫치를 스톨한다.

3.4.4 32비트 증분기는 각각의 프리펫치 사이클에 대해 TBUF PF_nPC 내의 어드레스 값을 조정한다.(x16).

3.2.5 제 1 (1차) 셋트내의 모든 조건부 분기 명령어가 해결될때까지 목표 명령어 스트림내의 제 2 명령어 셋트에서 발생하는 조건부 제어 흐름 명령어의 디코드(IDecode)에 대한 프리펫치 연산을 스톨한다(그러나 계속해서 상대 목표 어드레스를 계산하고 목표 레지스터에 저장한다).

3.2.6 만일 제 1 명령어 셋트내의 조건부 분기가 "바이어스를 취한"것으로 해결되면 ;

3.2.6.1 분기의 소스가 진행중 절차 비트로부터 결정된 바와 같은 EBUF 명령어 스트림인 경우에, MBUF 또는 EBUF 내의 제 1 조건부 흐름 명령어 셋트 다음에 오는 명령어 셋트를 플러쉬한다.

3.2.6.2 진행중 절차 비트의 상태에 근거하여 TBUF PFnPC 값을 MBUF PFnPC 또는 EBUF로 전달한다.

3.2.6.3 진행중 절차 비트의 상태에 근거하여 프리펫치된 TBUF 명령어를 MBUF 또는 EBUF로 전달한다.

3.2.6.4 만일 제 2 조건부 분기 명령어 셋트가 디코드(IDecoded) 되지 않았다면, 진행중 절차 비트의 상태에 근거하여 MBUF 또는 EBUF 프리펫치 연산을 계속한다.

3.2.6.5 만일 제 2 조건부 분기 명령어가 디코드(IDecoded) 되었다면, 그 명령어 처리를 시작한다(단계 3.2.1 로 진행).

3.2.7 만일 제 1 조건부 명령어 셋트내의 명령어에 대한 조건부 제어가 "바이어스를 취하지 않은"것으로 해결되면:

3.2.7.1 명령어 셋트 및 목표 명령어 스트림으로부터의 명령어의 IFIFO 및 IEU를 플러쉬한다.

3.2.7.2 MBUF 또는 EBUF 프리펫치 연산을 계속한다.

3.3 "바이어스를 취하지 않은 분기"에 대하여

3.3.1 MBUF로의 명령어의 프리펫치를 스톨한다; 실행 사이클링은 계속된다.

3.3.1.1 만일 제 1 조건부 명령어 셋트내의 조건부 제어 흐름 명령어가 상대적인(relative) 것이면, 목표 어드레스를 계산하고 목표 어드레스에 저장한다.

3.3.1.2 만일 제 1 조건부 명령어 셋트내의 조건부 제어 흐름 명령어가 절대적인(absolute) 것이면, 목표 어드레스 계산을 위해 IEU를 대기하고 어드레스를 목표 레지스터로 복귀시킨다.

3.3.1.3 제 1 조건부 명령어 셋트 명령어내의 조건부 제어 흐름 명령어가 해결될때까지 제 2 명령어 셋트내의 조건부 제어 흐름 명령어의 디코드(IDecode)에 대한 프리펫치 연산을 스톨한다.

3.3.2 제 1 조건부 분기의 목표 어드레스가 계산되면, TBUF PFnPC로 로드하고 또한 메인 명령어 스트림의 실행과 동시에 TBUF로의 명령어 프리펫치를 시작한다.

3.3.3 만일 제 1 명령어 셋트내의 조건부 제어 흐름 명령어가 "바이어스를 취한 것"으로 해결되면:

3.3.3.1 분기 소스가 진행중 절차 비트의 상태로부터 결정되는 바와 같이 EBUF 명령어 스트림인 경우에 MBUF 또는 EBUF를 플러쉬하고, 제 1 조건부 분기 명령어 셋트 다음에 오는 메인 스트림으로부터의 명령어 IFIFO 및 IEU를 플러쉬한다.

3.3.3.2 진행중 절차 비트의 상태로부터 결정되는 바와 같이, MBUF PFnPC 또는 EBUF로 TBUF PFnPC 값을 전달한다.

3.3.3.3 진행중 절차 비트의 상태로부터 결정되는 바와 같이, 프리펫치된 TBUF 명령어를 MBUF 또는 EBUF로 전달한다.

3.3.3.4 진행중 절차 비트의 상태로부터 결정되는 바와 같이 MBUF 또는 EBUF 프리펫치 연산을 계속한다.

3.3.4 제 1 셋트내의 조건부 제어 흐름 명령어가 "바이어스를 취하지 않은"것으로 해결되면:

3.3.4.1 목표 명령어 스트림으로부터 명령어 셋트의 TBUF를 플러쉬한다.

3.3.4.2 만일 제 2 조건부 분기 명령어가 디코드(IDecoded)되지 않았다면, 진행중 절차 비트의 상태로부터 결정되는 바와 같이, MBUF 또는 EBUF 프리펫치 연산을 계속한다.

3.3.4.3 만일 제 2 조건부 분기 명령어가 디코드(IDecoded)되었다면, 그 명령어 처리를 시작한다(단계 3.4.1로 진행)

4. 인터럽트, 예외, 및 트랩 명령어

4.1 트랩은 일반적으로 다음을 포함한다:

4.1.1. 하드웨어 인터럽트

4.1.1.1. 비동기(외부)로 발생하는 사건, 내부 또는 외부.

4.1.1.2 언제라도 발생할 수 있으며 존속할 수 있다.

4.1.1.3 극소의(atomic) (통상의: ordinary) 명령어 사이의 우선 순서로 서비스되고, 절차 명령어를 중지할 수도 있다.

4.1.1.4 인터럽트 조정기(handler)의 개시 어드레스는 트랩 조정기 엔트리 지점의 미리 정해진 테이블로의 벡터 넘버 오프셋으로서 결정된다.

4.1.2 소프트웨어 트랩 명령어

4.1.2.1 비동기(내부) 발생 명령어

4.1.2.2. 예외로서 실행하는 소프트웨어 명령어

4.1.2.3 트랩 조정기의 개시 어드레스는 TBR 또는 FTB 레지스터에 저장된 기본 어드레스 값과 조합된 트랩 넘버 오프셋으로부터 결정된다.

4.1.3 예외

4.1.3.1 명령어와 비동기로 발생하는 사건

4.1.3.2 명령어가 실행될 때 처리된다.

4.1.3.3 예외의 결과로 인해, 예외로된 명령어 및 모든 후속 실행된 명령어는 취소된다.

4.C.3.4 예외 조정기의 개시 어드레스는 트랩 조정기 엔트리지점의 미리 정해진 테이블로의 트랩 넘버 오프셋으로부터 결정된다.

4.2 트랩 명령어 스트림 연산을 그때 동시에 실행하는 명령어 스트림과 조화하여 발생한다.

4.3 트랩 처리 루틴이 다음에 허용되는 트랩 이전에 xPC 어드레스를 세이브한다는 조건으로 트랩이 내포(nest) 할 수도 있다 -- 그렇게 하는 것을 실패하는 것은 현재 트랩 연산의 완료 이전에 트랩이 발생하는 경우에 머신 스테이트를 약화시키게 한다.

5. 트랩 명령어 스트림 처리: xPC

5.1 트랩을 조우할 때:

5.1.1. 만일 비동기 인터럽트이면, 현재 실행하고 있는 명령어의 실행이 중지된다.

5.1.2 만일 비동기 예외이면, 예외로된 명령어의 실행시 트랩이 처리된다.

5.2 트랩이 처리될 때:

5.2.1 인터럽터는 디스에이블된다.

5.2.2. 현재 IF_PC 어드레스가 xPC 트랩 상태 복귀 어드레스 레지스터에 저장된다.

5.2.3 IF_PC 어드레스 및 그 후속의 IFIFO 및 MBUF 프리펫치 버퍼가 플러쉬된다.

5.2.4 어드레스 IF_PC 및 그 후속의 명령어를 실행하고, 그 명령어의 결과는 IEU로부터 플러쉬된다.

5.2.5 MBUF PFnPC가 트랩 조정기 루틴의 어드레스로 로드된다.

5.2.5.1 트랩 어드레스의 소스는 특수 레지스터 셋트에 제공되는, 트랩 넘버에 의해 결정되는 바와 같은 트랩의 형태에 따라 TBR 또는 FTB 레지스터가 된다.

5.2.6 명령어가 프리펫치되어, 정상적인 방식으로 실행을 위해 IFIFO로 들어온다.

5.2.7 트랩 루틴의 명령어가 실행된다.

5.2.7.1 트랩 처리 루틴이 미리 정해진 위치에 세이브될 xPC 어드레스 및 재 인에이블된 인터럽트를 제공할 수도 있다; xPC 어드레스는 특수 레지스터 이동 명령어와 특수 레지스터 어드레스 및 데이터 버스를 통해 판독/기록된다.

5.2.8 트랩 명령어로부터의 복귀의 실행에 의해 트랩 상태를 빠져나와야 된다.

5.2.8.1 만일 이전에 세이브되었으면, xPC 어드레스는 트랩 명령어로부터의 복귀를 실행하기 전에 그 미리 정해진 위치로부터 복원되어야 한다.

5.3 트랩으로부터의 복귀가 실행될 때:

5.3.1 인터럽트가 인에이블된다.

5.3.2 xPC 어드레스가 진행중 절차 비트의 상태로부터 결정되는 바와 같이, 현재 명령어 스트림 레지스터 MBUF 또는 EBUF PFnPC로 복귀되고, 프리펫치는 그 어드레스로부터 계속된다.

5.3.3. xPC 어드레스가 새로운 스트림 레지스터를 통해 IF_PC 레지스터로 복원된다.

E) 인터럽트 및 예외 처리:

1) 개요:

인터럽트 및 예외는 프로세서가 메인 명령어 스트림이나 또는 절차 명령어 스트림으로부터 실행중인지에 관계없이 그것이 인에이블되어 있는 한 처리되게 된다. 인터럽트 및 예외는 우선순위 순서로 서비스되며 클리어될 때까지 존속된다. 트랩 조정기의 개시 어드레스는 후술되는 바와 같이 트랩 조정기 어드레스의 미리 정해진 테이블의 벡터 넘버 오프셋으로서 결정된다.

인터럽트 및 예외는 실시예에서 2가지 기본 형태로 이루어지는데, 명령어 스트림내의 특정 명령어와 동기하여 발생하는 형태와 명령어 스트림내의 특정 명령어와 비동기로 발생하는 형태가 있다. 본 명세서에서 용어 인터럽트, 예외 트랩 및 고

장(fault)는 상호 교환가능하게 이용된다. 비동기 인터럽트는 명령어 스트림과 동기식으로 동작하지 않는 온-칩 또는 오프-칩의 하드웨어에 의해 발생된다. 예를 들어, 온-칩 타이머/카운터에 의해 발생되는 인터럽트는 오프-칩으로부터 제공되는 마스크 불가능(non-maskable) 인터럽트(NMI) 및 하드웨어 인터럽트와 같이 비동기이다. 비동기 인터럽트가 발생할 때, 프로세서 문맥(context)은 동결되고, 모든 트랩은 디스에이블되며, 일정한 프로세서 상태 정보는 저장되고, 프로세서는 수신된 특정 인터럽트에 대응하는 인터럽트 조정기로 자동 분기된다. 인터럽트 조정기와 그 처리를 완료한 후, 프로그램 실행은 인터럽트가 발생했을 때 실행중이던 스트림내의 마지막으로 완료된 명령어 다음에 오는 명령어로 계속된다.

동기 예외는 명령어 스트림내의 명령어와 동기하여 발생하는 예외이다. 이들 예외는 특정 명령어에 관련하여 발생하며, 관련 명령어가 실행될때까지 홀드된다. 양호한 실시예에서, 동기 예외는 프리펫치하는 동안, 명령어 디코드하는 동안, 및 명령어 실행중에 발생한다. 프리펫치 예외는 예를 들어 TLB 미스 또는 다른 VMU 예외를 포함한다. 디코더 예외는 예를 들어, 디코드되고 있는 명령어가 불법(illegal) 명령어이거나 또는 프로세서의 현재 특권(privilege) 레벨에 부합하지 않을 경우에 발생한다. 실행 예외는 예를 들어 0으로 나누는 것과 같은 산술 에러로 인해 발생한다. 이들 예외가 발생할 때마다, 본 양호한 실시예는 그것을 명령어가 회수될 때까지 예외를 유발한 특정 명령어와 일치하여 유지한다. 이 때, 모든 이전의 완료된 명령어는 회수되며, 예외를 유발한 명령어로부터의 시험적 결과는 다음의 시험적으로 실행되는 명령어의 시험적 결과와 같이, 플러쉬된다. 다음에, 그 명령어에 대해 발생한 최고 우선순위 예외에 대응하는 예외 조정기로 제어가 전달된다.

소프트웨어 트랩 명령어는 CF_DET(274)(제2도)에 의해 디코드 (IDecode) 스테이지에서 검출되며, 무조건 호출 명령어 및 다른 동기 트랩 모두와 비슷하게 처리된다. 즉, 목표 어드레스가 계산되고, 프리펫치는 그때 현재의 프리펫치 큐(queue)(EBUF 또는 MBUF)로 계속된다. 동시에, 예외는 또한 명령어와 일치하여 노트되며, 명령어가 회수될 때, 처리된다. 동기 예외의 모든 다른 형태는 단지 그것을 유발하여 실행시간에 처리된 특정 명령어와 일치하여 노트(noted) 되고 누산된다.

2)비동기 인터럽트:

비동기 인터럽트는 인터럽트 라인(292)을 통해 PC 로직 유니트(270)로 신호된다. 제3도에 도시된 바와 같이, 이들 라인은 PC 로직 유니트(270)내의 인터럽트 로직 유니트(363)에 제공되며, NMI 라인, IRQ 라인 및 인터럽트 레벨 라인(LVL) 셋트를 포함한다. NMI 라인은 마스크 불가능 인터럽트를 신호하며, 외부 소스로부터 유래한다. 이것은 하드웨어 리셋트를 제외하고 최고 우선순위 인터럽트이다. IRQ 라인도 또한 외부소스로부터 유래하며, 외부 장치가 하드웨어 인터럽트를 요청하는 때를 표시한다. 본 양호한 실시예에서는 최대 32개의 사용자-정의 외부 공급하드웨어 인터럽트까지 허용되며, 인터럽트를 요청하는 특정 외부 장치는 인터럽트 레벨 라인(LVL)에 대한 인터럽트 넘버(0-31)를 제공한다. 메모리 에러 라인은 여러 종류의 메모리 에러를 신호하도록 MCU(110)에 의해 활성화 된다. 타이머/카운터 인터럽트, 메모리 I/O 에러 인터럽트, 머신 검사 인터럽트 및 성능 모니터 인터럽트를 요청하기 위한 라인을 포함하여, 다른 비동기 인터럽트 라인(도시안됨)도 또한 인터럽트 로직 유니트(363)에 제공된다. 후술되는 동기 예외뿐 아니라 각각의 비동기 인터럽트는 그 인터럽트와 관련된 대응하는 소정의 트랩넘버를 갖는데, 32가지의 이들 트랩 넘버는 32개의 이용 가능한 하드웨어 인터럽트 레벨과 관련된다. 이들 트랩 넘버의 테이블은 인터럽트 로직 유니트(363)에서 유지된다. 일반적으로 트랩 넘버가 높을수록 트랩의 우선수가 높아진다.

비동기 인터럽트 중 한 인터럽트가 인터럽트 로직 유니트(363)로 신호될 때, 인터럽트 제어 유니트(363)는 INT REQ/ACK 라인(340)을 통해 IEU(104)에 인터럽트 요청을 보낸다. 인터럽트 제어 유니트(363)는 또한 PC 제어 유니트(262)로 하여금 프리펫치 명령어를 정지시키도록 하는 중지 프리펫치 신호를 라인(343)을 경유하여 PC 제어 유니트(362)로 보낸다. IEU(104)는 그때 실행되는 모든 명령어를 취소하고 모든 시험적 결과를 플러쉬하거나, 또는 몇 개 또는 모두의 명령어가 완료되되록 허용할 수도 있다. 양호한 실시예에서, 그때 실행중인 명령어는 취소되면 그래서 비동기 인터럽트에 대한 가장 빠른 응답을 허용한다. 어느 경우에도, 실행 PC 제어 유니트(356) 내의 DPC는 IEU(104)가 인터럽트를 승인하기 전에 완료되어 회수된 최종 명령어에 일치하도록 갱신된다. MBUF, EBUF, TBUF 및 IFIFO(264) 내의 모든 다른 프리펫치된 명령어도 또한 취소된다.

IEU(104)는 인터럽트 조정기로부터 명령어를 수신할 준비가 되어 있을때에만, INT REQ/ACK 라인(340) 상의 인터럽트 승인 신호를 인터럽트 제어 유니트(363)로 보내다.

3) 동기 예외:

동기 예외에 있어서, 인터럽트 제어 유니트(363)는 각 명령어 셋트에 대해 4개의 내부 예외 비트로된 셋트(도시안됨)를 유지하는데, 그 중 한 비트는 셋트내의 각 명령어에 대응한다. 인터럽트 제어 유니트(363)는 또한 각각의 명령어에 대해 검출된 것이 있다면, 특정 트랩 넘버의 표시를 유지한다.

특정 명령어 셋트가 프리펫치되고 있는 동안 VMU가 TLB 미스 또는 다른 VMU 예외를 신호하면, 이 정보는 VMU 제어 라인(332,334)을 통해, PC 로직 유니트(270) 및 특히 인터럽트 제어 유니트(363)로 전달된다. 인터럽트 제어 유니트(363)가 이와 같은 신호를 수신하면, 이 유니트는 더 이상의 프리펫치를 중지하도록 라인(343)을 통해 PC 제어 유니트(362)에 신호한다. 동시에, 인터럽트 제어 유니트(363)는 명령어 셋트가 행하는 프리펫치 버퍼와 관련하여 적절하게 VM_Miss 또는 VM_Excp 비트를 셋트한다. 다음에, 인터럽트 제어 유니트(363)는 셋트내의 명령어가 유효한 것이 없기 때문에 그 명령어 셋트에 대응하는 모든 4개의 내부 예외 인디케이터 비트를 셋트하고, 오류가 있는 명령어 셋트내의 4개의 명령어 각각에 일치하여 수신되는 특정 예외에 대한 트랩 넘버를 저장한다. 이 때, 오류 셋트가 IFIFO(264) 내의 최저 레벨에 도달할 때까지 오류 명령어 셋트 이전의 명령어의 시프팅 및 실행은 통상적으로 계속된다.

유사하게, 만일 다른 동기 예외가 프리펫치 버퍼(260), IDecode 유니트(262) 또는 IFIFO(264)를 통해 명령어의 시프팅 동안에 검출되면, 이 정보는 예외를 발생하는 명령어에 대응하는 내부 예외 인디케이터 비트를 셋트하고 그 예외와 일치하는 트랩 넘버를 저장하는 인터럽트 제어 유니트(363)로 전달된다. 프리펫치 동기 예외에서와 같이, 오류 명령어 이전의 명령어의 시프팅 및 실행은 오류 셋트가 IFIFO(264)에서 최저 레벨에 도달할 때까지 통상적으로 계속된다.

양호한 실시예에서, 프리펫치 버퍼(260), IDecode 유니트(262) 또는 IFIFO(264)를 통해 명령의 시프팅 중에 검출되는 예외의 유일한 형태는 소프트웨어 트랩 명령어이다. 소프트웨어 트랩 명령어는 CF_DET 유니트(274)에 의해 IDecode스테이지에서 검출된다. 몇몇 실시예에서는 다른 형태의 동기 예외가 IDecode 유니트(262)에서 검출될 수도 있지만, 다른 동기 예외의 검출은 명령어가 실행 유니트(104)에 도달할 때까지 대기하는 것이 바람직하다. 이것은 특권 명령어의 처리로부터 발생하는 바와 같은 어떤 예외가 명령어의 효과적인 정순서 실행 이전에 변화할 수 있는 프로세서 상태에 근거하여 신호될 수도 있는 가능성을 없앤다. 불법 명령어와 같이 프로세서 상태에 의존하지 않는 예외가 IDecode 스테이지에서 검출될 수 있지만, 그러나 하드웨어는 동일한 로직이 모든 사전 실행 동기 예외(VMU 예외와는 별문제임)를 검출하는 경우에 최소화된다. 이와 같은 예외의 처리는 거의 시간적으로 중대하지 않기 때문에 명령어가 실행 유니트(104)에 도달할 때까지 대기하는 것에 의해 부과되는 어떠한 시간적 불리한 조건도 없다.

전술한 바와같이, 소프트웨어 트랩 명령어는 CF_DET 유니트(274)에 의해 IDecode 스테이지에서 검출된다. 인터럽트 로직 유니트(363) 내의 그 명령어에 대응하는 내부 예외 인디케이터 비트는 셋트되고, 소프트웨어 트랩 명령어의 즉시 모드 오퍼랜드 필드에서 지정되는 0 내지 127 중 어떤 수가 될 수 있는 소프트웨어 트랩 넘버는 트랩 명령어에 일치하여 저장된다. 그러나, 프리펫치 동기 예외와는 달리, 소프트웨어 트랩이 제어 흐름 명령어로서 또한 동기 예외로서 처리되기 때문에, 인터럽트 제어 유니트(363)는 소프트웨어 트랩 명령어가 검출될 때, 프리펫치를 중지하도록 PC 제어 유니트(362)에 신호하지 않는다. 오히려, 명령어가 IFIFO(264)를 통해 시프트하는 것과 동시에, IFU(102)는 트랩 조정기를 MBUF 명령어 스트림 버퍼로 프리펫치한다.

명령어 셋트가 IFIFO(264)의 최저 레벨에 도달할 때, 인터럽트 로직 유니트(363)는 명령어 셋트내의 명령어중 만일 있다면 어느 것이 동기 예외의 소스가 되는 것으로 이미 결정되었는지 표시하기 위해 그 명령어에 대한 예외 인디케이터 비트를 4비트 벡터로서 SYNCH-INT-INFO 라인(341)을 통해 IEU(104)에 전달한다. IEU(104)는 즉시 응답하지 않고, 명령어 셋트내의 모든 명령어가 정상적인 진행으로 예정될 수 있도록 허용한다. 실행중에 정수산술 예외와 같은 더 이상의 예외가 발생될 수도 있다. 특권 명령어의 실행으로 인한 바와 같이 머신의 현재 상태에 의존하는 예외도 또한 이 때 검출되며, 머신의 상태와 명령어 스트림내의 모든 이전의 명령어에 관해 통용되도록 보장하기 위해, PSR(특수 이동 및 트랩 명령어로부터의 복귀와 같은)에 영향을 줄 가능성이 있는 모든 명령어는 순서대로 실행하도록 강요된다. 동기 예외의 소스가 되는 명령어가 회수될 것일 때에만, 예외의 발생이 인터럽트 로직 유니트(363)에 신호된다.

IEU(104)는 동기 예외를 갖는 제 1 명령어 이전의 명령어 스트림에서 발생하는 시험적으로 실행된 모든 명령어를 회수하며, 명령어 스트림에서 계속해서 발생하는 시험적으로 실행된 명령어로부터의 시험적 결과를 플러쉬 한다. 예외를 유발한 특정 명령어도 또한 플러쉬되는데, 이것은 그 명령어가 통상적으로 트랩으로부터의 복귀시 재실행되게 되기 때문이다. 다음에 실행 PC 제어 유니트(366)내의 IF_PC가 실제적으로 회수된 최종 명령어에 일치하도록 갱신되고, 이전의 예외가 인터럽트 제어 유니트(363)로 신호된다.

예외의 소스가 되는 명령어가 회수되면, IEU(104)는 회수 명령어 셋트(레지스터 224) 내의 명령중, 만일 있다면 동기 예외를 갖고 있는 명령어를 표시하는 새로운 4비트 벡터와 또한 명령어 셋트내의 제 1 예외의 소스를 표시하는 정보를 SYNCH-INT-INFO 라인(341)을 통해 인터럽트 로직 유니트(363)로 복구시킨다. IEU(104)에 의해 복귀되는 4비트 예외 벡터내의 정보는 IEU(104)에서 발생되는 예외 뿐만 아니라 인터럽트 로직 유니트(363)에 의해 IEU(104)에 제공되는 4비

트 예외 벡터의 누산값이다. 프리펫치 도는 IDecode에서 검출되는 예외로 인한 인터럽트 제어 유니트(363)에 기존에 저장된 정보와 함께 IEU(104)로부터 인터럽트 제어 유니트(363)로 복귀된 정보의 잔여 정보는 인터럽트 제어 유니트(363)가 최고 우선순위 동기 예외 및 그 트랩 넘버의 특성을 결정하는 데에 충분하다.

4)조정기 디스패치 및 복귀:

IEU로부터 라인(340)을 통해 인터럽트 승인 신호가 수신된 후, 또는 라인(341)을 통해 0이 아닌(non-zero)예외 벡터가 수신된 후, 현재 DPC는 특수 레지스터(412)(제4도) 중 하나인 xPC 레지스터에 복귀 어드레스로서 임시적으로 저장된다. 현재 프로세서 상태 레지스터(PSR)는 또한 이전의 PSR(PPSR) 레지스터에 저장되고, 현재 비교 상태 레지스터(CSR)는 특수 레지스터(412)내의 이전의 비교상태 레지스터(PCSR)에 세이브 된다.

트랩 조정기의 어드레스는 트랩 기본 레지스터 어드레스와 오프셋을 더한 값으로서 계산된다. PC 로직 유니트(270)는 트랩을 위한 2개의 기본 레지스터를 유지하는데, 이들 두 레지스터 모두는 특수 레지스터(제4도)의 일부이며, 이전에 실행된 특수 이동 명령어에 의해 초기화된다. 대부분의 트랩에 대해서, 조정기의 어드레스를 계산하기 위해 이용되는 기본 레지스터는 트랩 기본 레지스터 TBR이다.

인터럽트 제어 유니트(363)는 현재 보류중인 최고 우선순위 인터럽트 또는 예외를 결정하며, 조사 테이블(look-up-table)을 이용하여 관련된 트랩 넘버를 결정한다. 이것은 선택된 기본 레지스터에 대한 오프셋으로서 INT-OFFSET 라인(373) 셋트를 통해 프리펫치 PC 제어 유니트(364)로 제공된다. 유익하게도, 벡터 어드레스는 단지 TBR 레지스터로부터 얻어진 고순위 비트에 저순위 비트로서의 오프셋 비트를 연결(concatenating) 시킴으로써 계산된다. 이것은 가산기의 지연 필요성을 없앤다.(본 명세서에 사용된 바와 같이, 2비트가 i'번째 순위 비트로서 언급된다). 예를 들어 만일 트랩이 8비트 값으로서 표현되는 0 내지 255까지 넘버링되면, 조정기 어드레스는 22-비트 TBR 저장 값의 끝(end)에 8비트 트랩 넘버를 연결함으로써 계산될 수도 있다. 트랩 조정기 어드레스가 항상 워드 경계상에서 발생하는 것을 보장하기 위해 트랩 넘버에 2개의 저순위 제로 비트가 부가될 수도 있다. 이렇게 해서 구성된 연결된 조정기 어드레스는 프리펫치 선택기 PF_PC Sel(390)(제4도)에 한 입력(373)으로서 제공되며, 명령어가 프리펫치될 다음 어드레스로서 선택된다.

TBR 레지스터를 이용하는 트랩에 대한 벡터 조정기 어드레스는 모두 단지 한 워드 떨어져 있다. 그러므로, 트랩 조정기 어드레스에서의 명령어는 보다긴 트랩 처리 루틴으로의 예비 분기 명령어가 되어야 한다. 그러나, 소정의 트랩은 시스템 성능의 저하를 방지하기 위해 매우 신중한 처리를 요한다. 예를 들어, TLB 트랩은 매우 빠르게 실행되어야 한다. 이런 이유로 인해, 본 발명의 양호한 실시예는 상기 예비 분기의 희생없이 작은 트랩 조정기의 호출을 허용하도록 설계된 고속(fast) 트랩 메카니즘을 포함한다. 또한, 고속 트랩 조정기는 예를 들어 RAM 위치와 관련된 메모리 시스템의 불리한 조건(penalties)을 없애기 위해 온-칩 ROM 내의 메모리에 독자적으로 위치될 수 있다.

양호한 실시예에서, 결과적으로 고속 트랩이 되는 트랩만이 전술한 VMU 예외가 된다. 고속 트랩은 다른 트랩과 분리하여 넘버링되며, 0 내지 7의 범위를 갖는다. 그러나, 이들 트랩은 MMU 예외와 동일한 우선순위를 갖는다. 인터럽트 제어 유니트(363)가 그때 보류중인 최고 우선순위 트랩으로서 고속 트랩을 인식하면, 고속 트랩 기본 레지스터(FTB)가 특수 레지스터(412)로부터 선택되어 트랩 오프셋과 조합되기 위해 라인(416)상에 제공되도록 한다. 이 때, 라인(373')을 통해 프리펫치 선택기 PF_PC Sel(390)에 제공된 생성 벡터 어드레스는 FTB 레지스터로부터의 고순위 22비트를 연결한 것이 되는데, 이들 비트 뒤에는 고속 트랩 넘버를 나타내는 3개의 비트가 뒤따르고, 그 뒤에는 7개의 0비트가 뒤따른다. 그러므로, 각각의 고속 트랩 어드레스는 128 바이트, 즉 32워드 떨어져 있다. 호출되었을 때, 프로세서는 개시 워드로 분기하며, 그 블록내의 프로그램을 실행하거나 그 밖으로 분기할 수도 있다. 32개 또는 그보다 적은 명령어로 구현될 수도 있는 표준 TLB 처리 루틴과 같은 소형 프로그램의 실행은 실제 예외 처리 루틴으로의 예비 분기가 없어지기 때문에 통상적인 트랩보다 빠르게 이루어진다.

양호한 실시예에서 비록 모든 명령어가 4바이트의 동일한 길이(즉, 4개의 어드레스 위치를 차지함)를 가졌지만, 고속 트랩 메카니즘은 또한 그 명령어의 길이가 가변적인 마이크로프로세서에서도 유용하다는 것을 주지해야 한다. 이 경우에, 고속 트랩 벡터 어드레스는 마이크로프로세서에서 이용 가능한 가장 짧은 명령어중 적어도 2개의 명령어, 바람직하게는 약 32 평균 크기의 명령어를 수용하기에 충분한 간격으로 분리되어야 한다는 것을 알 수 잇다. 만일 마이크로프로세서가 트랩 명령어로부터의 복귀를 포함한다면, 벡터 어드레스는 그 명령어가 조정기에서 적어도 하나의 다른 명령어가 선행되도록 허용하도록 최소한의 충분한 간격으로 분리되어야 한다는 것은 확실하다.

또한, 트랩 조정기로의 디스패치에 관해서, 프로세서는 커널(kernel) 모드 및 인터럽트 상태로 들어간다. 동시에, 비교 상태 레지스터(CSR)의 카피(copy)가 이전의 캐리(carry) 상태 레지스터(PCSR)에 배치되고, PSR의 카피는 이전의 PSR(PPSR) 레지스터에 저장된다. 커널 및 인터럽트 상태 모드는 프로세서 상태 레지스터(PSR)에서 비트로 표현된다. 현재

PSR내의 인터럽트 상태 비트가 셋트되면, 전술한 바와 같이 또한 제7b도에 도시된 바와 같이 새도우(shadow) 레지스터 또는 트랩 레지스터 RT[24] 내지 RT[31]가 가시적(visible)이 된다. 인터럽트 조정기는 단지 PSR에 새로운 모드를 기록함으로써 커널 모드로부터 전환될 수도 있지만, 그러나 인터럽트 상태를 벗어나는 유일한 방법은 트랩(RTT) 명령어로부터의 복귀를 실행함으로써 이루어진다.

IEU(104)가 RTT 명령어를 실행할 때, PCSR은 CSR 레지스터로 복원되고 PPSR 레지스터는 PSR 레지스터로 복원되며, 그렇게 함으로써 PSR 레지스터내의 인터럽트 상태 비트를 자동적으로 클리어한다. PF_PC SEL 선택기(390)는 또한 특수 레지스터 셋트(412)내의 특수 레지스터 xPC는 증분기(394) 및 버스(396)를 통해 MBUF PFnPC나 또는 EBUF PFnPC로 적절하게 복원된다. xPC를 EBUF나 또는 MBUF PFnPC로 복원할 것인지에 관한 결정은 일단 복원된 PSR의 "진행중 절차" 비트에 따라 이루어진다.

프로세서는 트랩 및 절차 명령어 두가지 모두에 대한 복귀 어드레스를 저장하기 위해 동일한 특수 레지스터 xPC를 이용하지 않는다는 것을 주목해야 한다. 트랩에 대한 복귀 어드레스는 전술한 바와 같이 특수 레지스터 xPC에저장되지만, 절차 명령어 후로의 복귀를 위한 어드레스는 다른 특수 레지스터 uPC에 저장된다. 그러므로, 인터럽트 상태는 절차 명령에 의해 야기된 모방(emulation) 스트림을 실행하고 있는 동안에도 이용가능한 상태로 유지된다. 한편, 모방 스트림이 완료된 후 예외 조정기로의 복귀를 위해 어드레스를 저장하기 위한 특수 레지스터가 없기 때문에 예외 처리 루틴은 어떠한 절차 명령어도 포함해서는 안된다.

5) 내포(Nesting):

비록 소정의 프로세서 상태 정보는 트랩 조정기, 특히 CSR, PSR, 복귀 PC 및 어떤점에서는 "A" 레지스터 셋트 ra[24] 내지 ra[31]로의 디스패치에 대해 자동적으로 백-업 되지만, 다른 문맥(context) 정보는 보호되지 않는다. 예를 들어, 부동 소수점 상태 레지스터(FSR)의 내용은 자동적으로 백-업 되지 않는다. 만일 조정기가 이들 레지스터를 변경하고자 한다면, 그 자체의 백-업을 수행해야 한다.

트랩 조정기로의 디스패치에 대해 자동적으로 수행되는 제한된 백-업으로 인해, 트랩의 내포가 자동적으로 허용되지 않는다. 트랩 조정기는 소정의 원하는 레지스터를 백-업해야 하고, 소정의 인터럽트 조건을 클리어해야 하며, 시스템 레지스트로부터 트랩을 처리하는데 필요한 소정의 정보를 판독해야 하고, 그 정보를 적절하게 처리해야 한다. 트립 조정기로의 디스패치시 인터럽트는 자동적으로 디스에이블 된다. 처리 후에, 조정기는 백-업된 레지스터를 복원하고, 인터럽트를 재-인에이블 시키고, 인터럽트로부터의 복귀를 위한 RTT 명령어를 실행할 수 있다.

내포된 트랩이 허용되는 것이면, 트랩 조정기는 제 1 및 제 2 부분으로 분할되어야 한다. 제 1 부분에서는 인터럽트가 디스에이블되지만, xPC는 특수 레지스터 이동 명령어를 이용하여 카피되어야 하며, 트랩 조정기에 의해 유지되는 스택(stack)에 푸시(pushed) 되어야 한다. 이 때 트랩 조정기의 제 2 부분의 개시 어드레스는 특수 레지스터 이동 명령어를 이용하여 xPC로 이동되어야 하며, 트랩 명령어(RTT)로의 복귀가 실행되어야 한다. RTT는 인터럽트 상태를(PSR로의 PPSR의 복원을 통해) 재이동하고, xPC 내의 어드레스로 제어를 옮기게 되는데, 이 어드레스는 이제 조정기의 제 2 부분의 어드레스를 포함한다. 제 2 부분은 이 시점에서 인터럽트를 인에이블시키고, 인터럽트 모드에서의 실행 처리를 계속할 수도 있다. 새도우 레지스터 RT[24] 내지 RT[31]는 오직 상기 조정기의 제 1 부분에서만 가시적(visible) 이고 제 2 부분에서는 가시적이지 않다는 것을 주목해야 한다. 그러므로, 제 2 부분에서, 조정기는 "A" 레지스터 값중 조정기에 의해 변경될 가능성이 있는 레지스터 값을 보전(preserve) 해야 한다. 트랩 처리 절차가 종료되면, 모든 백-업된 레지스터를 복원해야 하고, 트랩 조정기로부터 원래의 xPC를 팝(pop)한 후, 그것을 특수 레지스터 이동 명령어를 이용하여 xPC 특수 레지스터로 다시 이동시켜야 하며, 다른 RTT를 실행해야 한다. 이것은 메인 또는 에뮬레이션 명령어 스트림내의 적당한 명령어로 제어를 복귀시킨다.

6) 트랩 리스트:

다음의 테이블 I은 양호한 실시예에서 인식되는 트랩 넘버, 우선순위 및 트랩의 처리 모드를 기재하고 있다.

테이블 I

트랩 #	처리모드	동기/비동기	트랩명
0-127	정상	동기	트랩 명령어
128	정상	동기	FP 예외
129	정상	동기	정수 산술 예외
130	정상	동기	MMU(TLB 미스 또는 변형제외)
135	정상	동기	비정렬 메모리 어드레스
136	정상	동기	불법 명령어
137	정상	동기	특권 명령어
138	정상	동기	디버그 예외
144	정상	비동기	성능 모니터
145	정상	비동기	타이머/카운터
146	정상	비동기	메모리 I/O 에러
160-191	정상	비동기	하드웨어 인터럽트
192-253	예비용		
254	정상	비동기	머신 검사
255	정상	비동기	NMI
0	고속트랩	동기	고속 MMU TLB 미스
1	고속트랩	동기	고속 MMU TLB 변형
2-3	고속트랩	동기	고속 MMU(예비용)
4-7	고속트랩	동기	고속 (예비용)

Ⅲ. 명령어 실행 유니트:

제5도에서는 IEU(104)의 조합된 제어 및 데이터 경로 부분이 도시되어 있다. 1차 데이터 경로는 IFU(102)로부터의 명령어/오퍼랜드 데이터 버스(124)로 시작된다. 데이터 버스로서, 즉시 오퍼랜드가 오퍼랜드 정렬 유니트(470)에 제공되어 레지스터 파일(REG ARRAY)(472)로 통과된다. 레지스터 데이터는 레지스터 파일(472)로부터 레지스터 파일 출력 버스(476)를 경유해 바이패스 유니트(474)를 통해, 분배 버스(480)를 경유하여 기능 계산 엘리멘트(functional computing elements)(FUo-n)(478o-n)의 병렬 어레이로 제공된다. 기능 유니트(478o-n)에 의해 발생된 데이터는 다시 출력 버스(482)를 통해 바이패스 유니트(474)나 또는 레지스터 어레이(472), 또는 상기 유니트(474)와 어레이(472) 모두에 제공된다.

로드/스토어 유니트(484)는 IEU(104)의 데이터 경로 부분을 완료한다. 로드/스토어 유니트(484)는 IEU(104)와 CCU(106) 사이의 데이터 전달을 관리할 책임이 있다. 특히, CCU(106)의 데이터 캐쉬(134)로부터 얻어지는 로드 데이터는 로드/스토어 유니트(484)에 의해 로드 데이터 버스(486)를 통해 레지스터 어레이(472)의 입력으로 전달된다. CCU(106)의 데이터 캐쉬(134)에 저장될 데이터는 기능 유니트 분배 버스(480)로부터 수신된다.

IEU(104)의 제어 경로 부분은 IEU 데이터 경로를 통한 정보의 처리를 발생, 관리, 완료할 책임이 있다. 본 발명의 양호한 실시예에서, IEU 제어 경로는 복수 명령어의 동시 실행을 관리할 수 있으며, IEU 데이터 경로는 IEU(104)의 근본적으로 모든 데이터 경로 엘리멘트 사이의 다수의 독자적인 데이터 전달을 제고한다. IEU 제어 경로는 명령어/오퍼랜드 버스(124)를 통해 수신되는 명령어에 응답하여 동작한다. 특히, 명령어 셋트는 EDecode 유니트(490)에 의해 수신된다. 본 발명의 양호한 실시예에서, EDecode(490)는 IFIFO 마스터 레지스터(216,224)에 의해 홀드된 2개 모두의 명령어 셋트를 수신 및 디코드한다. 모두 8개의 명령어의 디코딩의 결과는 캐리 검사(CRY CHKR) 유니트(492), 의존도 검사(DEP CHKR) 유니트(494), 레지스터 리네임(renaming) 유니트(REGRENAME)(496), 명령어 발생(ISSUER) 유니트(498) 및 회수(retirement) 제어 유니트(RETIRE CTL)(500)에 다양하게 제공된다.

캐리 검사 유니트(492)는 제어 라인(502)을 통해 EDecode 유니트(490)로부터 8개의 보류 명령어에 대한 디코드된 정보를 수신한다. 캐리 검사 유니트(492)의 기능은 보류 명령어 중에서 프로세서 상태 워드의 캐리 비트에 영향을 주거나 또는 캐리 비트의 상태에 의존하는 보류 명령어를 식별하는 것이다. 이 제어 정보는 제어 라인(504)을 통해 명령어 발생 유니트(498)에 제공된다.

8개의 보류 명령어에 의해 이용되는 레지스터 파일(472)의 레지스터들을 식별하는 디코드된 정보는 제어 라인(506)을 통해 레지스터 리네임 유니트(496)에 직접 제공된다. 이 정보는 또한 의존도 검사 유니트(494)에도 제공된다. 의존도 검사

유니트(494)의 기능은 데이터에 대한 목적지로서 레지스터를 어느 보류 명령어가 참조하는지 결정하고 만일 있다면 어느 명령어가 상기 목적지 레지스터중 어느 레지스터에 의존하는지를 결정하는 것이다. 레지스터 의존도를 갖는 명령어는 제어 라인(508)에 의해 레지스터 리네임 유니트(496)에 제공되는 제어 신호에 의해 식별된다.

마지막으로, EDecode 유니트(490)는 8개의 보류 명령어 각각의 특성 및 기능을 식별하는 제어정보를 제어 라인(510)을 통해 명령어 발생 유니트(498)에 제공한다. 명령어 발생 유니트(498)는 특히 보류 명령어의 실행을 위해 특정 기능 유니트의 유효성에 관한 데이터 경로 자원(resources)를 결정할 책임이 있다. 본 구조 (100)의 양호한 실시예에 따라, 명령어 발생 유니트(498)는 데이터 경로 자원의 유효성과 레지스터 의존도 제한조건(constraints)에 종속되는 8개의 보류 명령어 중 소정의 명령어의 비순서(out-of-order) 실행을 허용한다. 레지스터 리네임 유니트(496)는 실행을 허용하도록 적당하게 속박되지 않는 명령어의 제어 라인(512)을 통해 명령어 발생 유니트(498)에 비트 맵(map)을 제공한다. 이미 실행된 명령어 및 레지스터 또는 캐리 의존도를 가진 명령어는 비트 맵으로부터 논리적으로 재이동된다.

요구되는 기능 유니트(478o-n)의 유효성에 의존하여, 명령어 발생 유니트(498)는 각 시스템 클럭 사이클 동안에 복수 명령어의 실행을 개시할 수도 있다. 기능 유니트(478o-n)의 상태는 상태버스(514)를 경유해 명령어 발생 유니트(498)로 제공된다. 명령어의 실행을 개시하고 계속해서 관리하기 위한 제어신호는 명령어 발생 유니트(498)에 의해 제어 라인(516)을 통해 레지스터 리네임 유니트(496) 및 선택적으로 기능 유니트(478o-n)에 제공된다. 이에 응답하여, 레지스터 리네임 유니트(496)는 레지스터 파일 억세스 제어 버스(518) 상에 레지스터 선택 신호를 제공한다. 버스(518)상에 제공되는 제어 신호를 통해 인에이블된 특정 레지스터는 실행되고 있는 명령어의 선택과, 그 특정 명령어의 의해 참조되는 레지스터의 레지스터 리네임 유니트(496)에 의한 결정에 의해 결정된다.

바이패스 제어 유니트(BYPASS CTL)(520)는 일반적으로 제어 라인(524)상의 제어 신호를 통해 바이패스 데이터 루팅(routing) 유니트(474)의 동작을 제어한다. 바이패스 제어 유니트(520)는 기능 유니트(478o-n) 각각의 상태를 모니터하고, 제어 라인(522)을 통해 레지스터 리네임 유니트(496)로부터 제공되는 레지스터 참조와 관련하여, 데이터가 레지스터 파일(472)로부터 기능 유니트(478o-n)로 루팅될 것인지 결정하고, 또는 기능 유니트(478o-n)에 의해 발생되는 데이터가 명령어 발생 유니트(498)에 의해 선택된 새로 발생된 명령어의 실행에 이용하기 위해 바이패스 유니트(474)를 통해 기능 유니트 목적지 버스(480)로 즉시 루팅될 수 있는지를 결정한다. 어느 경우에도, 명령어 발생 유니트(498)는 각각의 기능 유니트(478o-n)로의 특정 레지스터 데이터를 선택적으로 인에이블시킴으로써 분배 버스(480)로부터 기능 유니트(478o-n)까지의 데이터 루팅을 직접 제어한다.

IEU 제어 경로의 나머지 유니트는 회수 제어 유니트(500), 제어흐름 제어(CF CTI) 유니트(528), 및 완료(done) 제어(DONE CTL) 유니트(536)를 포함한다. 회수 제어 유니트(500)는 비순서적으로 실행된 명령어의 실행을 무효로 하거나, 또는 유효로 확인하여 동작을 한다. 명령어가 비순서적으로 실행된 경우에, 그 명령어는 모든 이전의 명령어도 역시 회수된 후 유효로 확인되거나 회수될 수 있다. 제어 라인(532)상에 제공되는 현재 셋트의 8개의 보류 명령어중 실행된 명령어의 식별을 기초로 하여, 회수 제어 유니트(500)는 비순서 실행된 명령어의 이전의 실행의 결과로서 레지스터 어레이(472)에 의해 저장된 결과 데이터를 효과적으로 확인하기 위해 버스(518)에 결합된 제어 라인(534) 상에 제어 신호를 제공한다.

회수 제어 유니트(500)는 그것이 각 명령어를 회수할 때 PC 증분/사이즈 제어신호를 제어 라인(344)을 통해 IFU(102)에 제공한다. 복수의 명령어가 비순서적으로 실행될 수도 있고, 그러므로 동시 회수를 위한 준비를 할 수도 있기 때문에, 회수 제어 유니트(500)는 동시에 회수되는 명령어의 수에 근거하여 사이즈 값을 결정한다. 마지막으로, IFIFO 레지스터(224)의 모든 명령어가 실행되어 회수된 경우에, 회수 제어 유니트(500)는 IFIFO 유니트(264)의 시프트 연산을 개시하기 위해 IFIFO 판독 제어 신호를 제어 라인(342)를 통해 IFU(102)에 제공하며, 그렇게 함으로써 추가의 4개의 명령어를 실행 보류 명령어로서 EDecode 유니트(490)에 제공하게 된다.

제어 흐름 제어 유니트(528)는 각각의 조건부 분기 명령어의 논리적 분기 결과를 검출하는 보다 특수한 기능을 수행한다. 제어 흐름 제어 유니트(528)는 제어 라인(510)을 통해 EDecode 유니트(490)으로부터 현재 보류중인 조건부 분기 명령어의 8비트 벡터 아이덴티피케이션(identification)을 수신한다. 8비트 벡터 명령어 완료(done) 제어 신호도 비슷하게 완료 제어 유니트(540)로부터 제어 라인(538)을 통해 수신된다. 상기 완료 제어 신호는 제어 흐름 제어 유니트(528)로 하여금 조건부 분기 명령어가 적어도 조건부 제어 흐름 상태를 결정하기에 충분한 지점으로 이행되는 때를 식별할 수 있도록 허용한다. 보류중인 조건부 분기 명령어에 대한 제어 흐름 상태 결과는 그 명령어가 실행될 때 제어 흐름 제어 유니트(528)에 의해 저장된다. 조건부 제어 흐름 명령어 결과를 결정하는데 필요한 데이터는 제어 라인(530)을 통해 레지스터 어레이(472)내의 임시 상태 레지스터로부터 얻어진다. 각각의 조건부 제어 흐름 명령어가 실행될때, 제어 흐름 제어 유니트는 새로운 제어 흐름 결과 신호를 제어 라인(348)을 통해 IFU(102)에 제공한다. 상기 제어 흐름 결과 신호는 바람직하게, 8개의 잠재적으로 보류중인 제어 흐름 명령어의 각각의 비트 위치에 의한 상태 결과가 알려져 있는지를 정의하고 역시 비트 위치의 일치에 의해 주어지는 대응하는 상태 결과 상태를 정의하는 2개의 8비트 벡터를 포함한다.

마지막으로, 완료 제어 유니트(540)는 기능 유니트(478o-n)의 각각의 연산 실행 상태를 모니터하기 위해 제공된다. 기능 유니트(478o-n) 중 어느 유니트가 명령어 실행 연산의 완료를 신호하면, 완료 제어 유니트(540)는 레지스터 리네임 유니트(496), 명령어 발생 유니트(498), 회수 제어 유니트(500) 및 바이패스 제어 유니트(520)에 경보하기 위해 대응하는 완료 제어 신호를 제어 라인(542) 상에 제공한다.

기능 유니트(478o-n)의 병렬 어레이 구성은 IEU(104)의 제어 일관성(consistency)을 향상시킨다. 개별 기능 유니트(478o-n)의 특성은 명령어가 실행을 위해 적당하게 인식되고 예정되도록 하기 위해 명령어 발생 유니트(498)가 알아야 한다. 기능 유니트 (478o-n)는 그 필요한 기능을 수행하는데 필요한 그 특수 제어 호름 연산을 결정 및 구현할 책임이 있다. 그러므로, 명령어 발생 유니트(498)와는 달리, IEU 제어 유니트 중 어느 것도 명령어의 제어 흐름 처리를 독자적으로 인식할 필요는 없다. 명령어 발생 유니트(498) 및 기능 유니트(478o-n)는 함께, 잔여 제어 흐름 관리 유니트(496,500,520,528,540)에 의해 수행될 기능의 필요한 제어 신호 프롬프팅(prompting)을 제공한다. 그러므로, 기능 유니트(478o-n)의 특정 제어 흐름 연산에서의 변경은 IEU(104)의 제어 연산에 영향을 주지 않는다. 더욱이, 기존의 기능 유니트(478o-n)의 기능적 증대와 또한, 확장된 정밀 부동 소수점 승산기(multiplier) 및 확장된 정밀 부동 소수점 ALU, 고속 퓨리어 계산 기능 유니트, 및 삼각법 계산 유니트와 같이 하나 또는 그 이상의 새로운 기능 유니트(478o-n)를 추가하는 것 조차도 단지 약간의 명령어 발생 유니트(498)의 변형만을 필요로 한다. 요구되는 변형은 필요한 기능 유니트(478o-n)에 대한 명령어의 상관성과 EDecode 유니트(490)에 의해 분리되는 대응하는 명령어 필드에 근거하여 특정 명령어의 인식(recognition)을 제공해야 한다. 레지스터 데이터의 선택, 데이터의 루팅, 명령어 완료 및 회수에 대한 제어는 모든 다른 기능 유니트(478o-n)에 관해 실행된 모든 다른 명령어의 처리와 일관성을 유지한다.

A) IEU 데이터 경로 상세:

IEU 데이터 경로의 중심 구성 요소는 레지스터 파일(472)이다. 그러나, IEU 데이터 경로내에서, 본 발명은 특정 기능에 대해 일반적으로 최적화된 다수의 병렬 데이터 경로를 제공한다. 2개의 주요 데이터 경로는 정수와 부동 소수점이다. 각각의 병렬 데이터 경로내에서 레지스터 파일(472)의 일부는 그 데이터 경로내에서 발생하는 데이터 조작(manipulations)을 지원하기 위해 제공된다.

1)레지스터 파일 상세:

제6a도에는 데이터 경로 레지스터 파일의 양호한 일반적 구조가 도시되어 있다. 데이터 경로 레지스터 파일(550)은 임시 버퍼(552), 레지스터 파일 어레이(564), 입력 선택기(559) 및 출력 선택기(556)을 포함하고 있다. 궁극적으로 레지스터 어레이(564)를 목적지로 하는 데이터는 통상적으로 먼저 조합 데이터 입력 버스(558')를 통해 임시 버퍼(552)에 의해 수신된다. 즉, 데이터 경로 레지스터 파일(550)로 향하는 모든 데이터는 다수의 입력 버스(558), 바람직하게는 2개의 입력 버스로부터 입력 버스(558')로 입력 선택기(559)에 의해 멀티플렉스 된다. 제어 버스(518)상에 제공되는 레지스터 선택 및 인에이블 제어신호는 임시 버퍼(552)내의 수신 데이터에 대한 레지스터 위치를 선택한다. 임시 버퍼에 저장된 데이터를 발생한 명령어의 회수시, 다시 제어 버스(518)에 제공되는 제어 신호는 데이터 버스(560)를 통해 임시 버퍼(552)로부터 레지스터 파일 어레이(564)내의 논리적으로 대응하는 레지스터로의 데이터 전달을 인에이블시킨다. 그러나, 명령어의 회수 이전에, 임시 버퍼(552)의 레지스터에 저장된 데이터는 임시 버퍼 저장 데이터를 데이터 버스(560)의 바이패스 부분을 통해 출력 데이터 선택기(556)로 루팅함으로써 후속 명령어의 실행에 이용될 수도 있다. 제어 버스(518)를 통해 제공되는 제어 신호에 의해 제어되는 선택기(556)는 임시 버퍼(552)의 레지스터로부터 제공되는 데이터와 레지스터 파일 어레이(564)의 레지스터로부터 제공되는 데이터 사이에서 선택한다. 그 최종 데이터는 레지스터 파일 출력 버스(564)상에 제공된다. 또한, 실행 명령어가 완료시 회수되게 되는 경우에, 즉 명령어가 완료시 회수되게 되는 경루에, 즉 명령어가 정순서(in-order)로 실행된 경우에, 입력 선택기(559)는 바이패스 연장 라인(558")을 통해 레지스터 어레이(554)로 직접 결과 데이터를 루팅하도록 조작될 수 있다.

본 발명의 양호나 실시예에 따라, 각각의 데이터 경로 레지스터 파일(550)은 2개의 동시 레지스터 연산이 이루어질 수 있도록 허용한다. 그러므로, 입력 버스(558)는 임시 버퍼(552)에 기록될 2개의 완전(full) 레지스터 폭(width) 데이터 값을 제공한다. 내부적으로, 임시 버퍼(552)는 임시 버퍼(552) 내의 2개의 레지스터로의 입력 데이터의 동시 루팅을 허용하는 멀티플렉서 어레이를 제공한다. 비슷하게, 내부 멀티플렉서는 임시 버퍼(552)의 임의의 5개의 레지스터가 버스(560) 상으로 데이터를 출력하도록 선택될 수 있도록 허용한다. 레지스터 파일 어레이(564)도 또한 동시에 각각의 데이터를 버스(560) 상에서 수신하도록 2개의 레지스터가 선택될 수도 있도록 하고 5개의 레지스터는 버스(562)를 통한 각 데이터의 소스가 되도록 선택될 수 있도록 허용하는 입력 및 출력 멀티플렉서를 포함하고 있다. 마지막으로, 레지스터 파일 출력 선택기(556)는 버스(560,562)를 통해 수신되는 10개의 레지스터 데이터 값 중 임의의 5개의 값이 동시에 레지스터 파일 출력 버스(564)상에 출력될 수 있도록 구현되는 것이 바람직하다.

제6도b에는 임시 버퍼내의 레지스터 셋트가 일반적으로 도시되어 있다. 레지스터 셋트(552')는 8개의 단일(single) 워드(32비트) 레지스터 IORD, I1RD...I7RD로 이루어져 있다. 레지스터 셋트(552')는 4개의 더블(double)워드 레지스터 IORD, IORD + 1(IORD4), I1RD, I1RD + 1(ISRD)...I3RD, I3RD + 1(I7RD)의 셋트로서 이용될 수도 있다.

본 발명에 따라, 레지스터 파일 어레이(564)내의 각각의 레지스터에 대해 이중의 레지스터를 제공하는 것이 아니라, 임시 버퍼 레지스터 셋트(552) 내의 레지스터는 IFIFO 마스터 레지스터(216,224) 내의 각 명령어의 상대적 위치에 근거하여 레지스터 리네임 유니트(496)에 의해 참조된다. 본 구조(100)에 의해 구현되는 각각의 명령어는 그 명령어의 실행에 의해 발생되는 데이터의 목적지를 위해 2개 까지의 레지스터 또는 하나의 더블 워드 레지스터를 참조할 수도 있다. 통상적으로, 한 명령어는 오직 하나의 단일 출력 레지스터만을 참조하게 된다. 그러므로, 단일 출력 레지스터를 참조하는 제6c도에 도시된 바와 같이 위치상으로 식별되는 8개의 보유 명령어 중 명령어 2(I_2)에 대해, 데이터 목적지 레지스터 I2RD는 명령어의 실행에 의해 발생되는 데이터를 수신하도록 선택되게 된다. 명령어 I_2에 의해 발생된 데이터가 예를 들어 후속 명령어 I_5에 의해 이용되는 경우에, I2RD 레지스터에 저장된 데이터는 버스(560)를 통해 전달되게 되며, 그 생성 데이터는 임시 버퍼(552)로 다시 전달되어 I5RD로 식별되는 레지스터로 저장된다. 현저하게, 명령어 I_5는 명령어 I_2에 의존한다. 명령어 I_5는 I_2로부터의 결과 데이터가 유효하게 될 때까지 실행될 수 없다. 그러나, 알 수 있는 바와 같이, 명령어 I_5는 임시 버퍼(552')의 명령어 I_2 데이터 위치로부터 그 필요한 입력 데이터를 얻음으로서 명령어 I_2 의 회수 이전에 실행될 수 있다.

마지막으로, 명령어 I_2가 회수될 때, 레지스터 I2RD로부터의 데이터는 회수 지점에서 명령어의 논리적 위치에 의해 결정되는 바와 같은 레지스터 파일 어레이(564)내의 레지스터 위치에 기록된다. 즉, 회수 제어 유니트(560)는 제어 라인(510)상에 EDecode 유니트(490)로부터 제공되는 레지스터 참조 필드 데이터로부터 레지스터 파일 어레이 내의 목적지 레지스터의 어드레스를 결정한다. 일단 명령어 I_{0-3}가 회수되면, I4RD 내지 I7RD 내의 값은 IFIFO 유니트(264)와 동시에 IORD내지 I3RD로 시프트된다.

명령어 I_2가 더블 워드 결과 값을 제공하는 경우에 복잡화가 발생한다. 본 발명의 양호한 실시에에 따라, 위치 I2RD 및 I6RD의 조합은 그 명령어가 회수되거나 아니면 취소될 때 까지 명령어 I_2로부터 생성되는 데이터를 저장하기 위해 이용된다. 양호한 실시예에서, 명령어 I_{4-7}의 실행은 명령어 I_{0-3} 중 어느 명령어에 의한 더블 워드 출력 참조가 레지스터 리네임 유니트(496)에 의해 검출되는 경우에 홀드된다. 이것은 전체 임시 버퍼(552')가 더블 워드 레지스터의 단일 랭크로서 이용될 수 있도록 허용된다. 일단 명령어 I_{0-3}가 회수되면, 임시 버퍼(552')는 다시 단일 레지스터의 2개의 랭크로서 이용될 수 있다. 더욱이, 소정의 명령어 I_{4-7}의 실행은 명령어가 대응하는 I_{0-3} 위치로 시프트될 때 까지 더블 워드 출력 레지스터가 요구되는 경우에 홀드된다.

제7a도 및 제7b도에는 레지스터 파일 어레이(564)의 논리적 구성이 도시되어 있다. 본 발명의 양호한 실시예에 따라, 정수 데이터 경로에 대한 레지스터 파일 어레이(564)는 40개의 32-비트 폭 레지스터로 이루어진다. 레지스터 셋트 "A"를 형성하는 레지스터 셋트는 기본 레지스터 셋트 ra[0 ..... 23](565), 일반 목적 레지스터로된 상부 셋트 ra[24 ..... 31](566) 및 8개의 일반 목적 트랩 레지스터로된 새도우 레지스터 셋트 rt[24 ..... 31]로서 구성된다. 정상적인 동작시, 일반 목적 레지스터 ra[0...31](565,566)는 정수 데이터 경로에 대한 레지스터 파일 어레이의 활성 "A" 레지스터 셋트를 형성한다.

제7b도에 도시된 바와 같이 트랩 레지스터 rt[24....31](567)는 활성 기본 레지스터 셋트 ra[0....23](565)와 함께 억세스를 허용하도록 활성 레지스터 셋트 "A" 로 스왑(swapped)될 수도 있다. 이와 같은 "A" 레지스터 셋트의 구성은 인터럽트의 승인 또는 예외 트랩 처리 루틴의 실행시 선택된다. 레지스터 셋트 "A"의 이와 같은 상태는 인에이블 인터럽트 명령어의 실행 또는 트랩 명령어로부터의 복귀의 실행에 의해 제7a도에 도시된 상태로 특별히 복귀될때까지 유지된다.

본 구조(100)에 의해 구현되는 바와 같은 본 발명의 양호한 실시에에 따라, 부동 소수점 데이터 경로는 제8도에 일반적으로 도시된 바와 같이 확장 정밀 레지스터 파일 어레이(572)를 이용한다. 레지스터 파일 어레이(572)는 64-비트 폭을 각각 갖고 있는 32개의 레지스터 rf[0.....31]로 구성된다. 부동 소수점 레지스터 파일(572)은 또한 "B" 셋트의 정수 레지스터 rb[0.....31]로서 논리적으로 참조될 수도 있다. 본 구조(100)에서 상기 레지스터 "B" 셋트는 부동 소수점 레지스터 rf[0....31] 각각의 저순위 32 비트와 동등하다.

제 3 데이터 경로를 기술하면, 제9도에 도시된 바와 같이, 부울 조합(boolean combinatorial) 연산의 논리적 결과를 저장하기 위해 부울 연산자(operator) 레지스터 셋트(574)가 제공된다. 상기 "C" 레지스터 셋트(574)는 32개의 단일 비트 레

지스터 rc[0.....31]로 구성된다. 부울 레지스터 셋트(574)의 연산은 부울 연산의 결과가 부울 레지스터 셋트(574)의 명령어 선택 레지스터로 향할 수 있기 때문에 유일하다. 이것은 동일한(equal) 단순 부울 상태 값, 동일하지 않은(not equal) 단순 부울 상태 값, 보다 큰(greater than) 단순 부울 상태 값 및 다른(other) 단순 부울 상태 값과 같은 조건에 대한 단일 비트 플래그를 저장하는 단일 프로세서 상태 워드 레지스터를 이용하는 것과 대조적이다.

부동 소수점 레지스터 셋트(572)와 부울 레지스터 셋트(574)는 둘 다 제6b도에 도시된 정수 임시 버퍼(552)와 구조적으로 동일한 임시 버퍼에 의해 상호 보완된다. 근본적인 차이점은 임시 버퍼 레지스터의 폭이 상보적인 레지스터 파일 어레이(572,574)의 폭, 양호한 구현에 있어서는 각각 64비트 1비트인 폭과 동일하게 되도록 정의된다는 것이다.

레지스터 어레이(472)에는 적어도 논리적으로는 다수의 추가의 특수 레지스터가 존재한다. 제7c도에 도시된 바와 같이, 레지스터 어레이(472)에 물리적으로 존재하는 레지스터는 커널 스택 포인터(568), 프로세서 상태 레지스터(PSR)(569), 이전의 프로세서 상태 레지스터(PPSR)(570) 및 8개의 임시 프로세서 상태 레지스터(tPSR [0 ..... 7])의 어레이(571)를 포함한다. 잔여 특수 레지스터는 본 구조(100)의 여러 부분에 걸쳐 분배된다. 특수 어드레스 및 데이터 버스(354)는 특수 레지스터와 "A" 및 "B" 레지스터 셋트 사이의 선택 및 데이터 전달을 위해 제공된다. "A" 또는 "B" 레지스터 셋트 중 어느 하나로부터 레지스터를 선택하고, 전달 방향을 선택하고 특수 레지스터의 어드레스 식별자(identifier)를 지정하기 위해 특수 레지스터 이동 명령어가 제공된다.

커널 스택 포인터 레지스터 및 임시 프로세서 상태 레지스터는 다른 특수 레지스터와 다르다. 커널 스택 포인터는 커널 상태에 있을때 레지스터 이동 명령어에 대한 표준 레지스터의 실행을 통해 억세스 될 수도 있다. 임시 프로세서 상태 레지스터는 직접 억세스 될 수 없다. 오히려, 이 레지스터 어레이는 비순서(out-of-order) 실행 명령어에 의한 이용을 위해 프로세서 상태 레지스터의 값을 전파하기 위한 계승(inheritance) 메카니즘을 구현하기 위해 이용된다. 초기 전파(propagation) 값은 최종 회수된 명령어에 의해 제공되는 값인 프로세서 상태 레지스터의 값이다. 이 초기 값은 비순서 실행 명령어가 위치적으로 대응하는 임시 프로세서 상태 레지스터내의 값을 억세스하게 되도록 임시 프로세서 상태 레지스터를 통해 앞으로 전파된다. 명령어의 특정 성질은 명령어가 의존하고 변화시킬 수도 있는, 만일 있다면, 조건 코드 비트를 정의한다. 명령어가 레지스터 의존도 검사 유니트(494) 및 캐리 의존도 검사 유니트(492)에 의해 결정되는 바와 같은 의존도, 레지스터, 또는 조건 코드에 의해 제한되지 않는 경우에, 그 명령어는 비순서적으로 실행될 수 있다. 프로세서 상태 레지스터의 조건 코드 비트의 변형은 논리적으로 대응하는 임시 프로세서 상태 레지스터로 향한다. 특히, 변화될 수도 있는 그런 비트만이 임시 프로세서 상태 레지스터내의 값으로 인가되고, 모든 보다 높은 순위의 임시 프로세서 상태 레지스터로 전파된다. 결과적으로, 모든 비순서적으로 실행된 명령어는 어떤 삽입되는 PSR 변형 명령어에 의해 적절하게 변형된 프로세서 상태 레지스터 값으로부터 실행된다. 명령어의 회수는 단지 대응하는 임시 프로세서 상태 레지스터 값을 PSR 레지스터(569)로 전달한다.

나머지 특수 레지스터는 테이블 Ⅱ에 기재되어 있다.

테이블 II		
특수 레지스터		
특수 이동		
Reg	R/W	설 명
PC	R	프로그램 카운터: 일반적으로, PC는 현재 실행중인 프로그램 명령어 스트림의 다음 어드레스를 유지한다.
IF_PC	R/W	IFU 프로그램 카운터: IF_PC는 정확한 다음 실행 어드레스를 유지한다.
PFnPCs	R	프리펫치 프로그램 카운터: MBUF, TBUF 및 EBUF PFnPC는 각각의 프리펫치 명령어 스트림에 대한 다음 프리펫치 명령어 어드레스를 유지한다.
uPC	R/W	마이크로-프로그램 카운터: 절차 명령어 다음에 오는 명령어의 어드레스를 유지한다. 이것은 절차 루틴으로부터의 복귀시 실행될 제 1 명령어의 어드레스이다.
xPC	R/W	인터럽트/예외 프로그램 카운터: 인터럽트 및 예외의 복귀 어드레스는 홀드한다. 이 복귀 어드레스는 트랩시의 IF_PC의 어드레스이다.
TBR	W	트랩 기본 레지스터: 트랩 처리 루틴 디스패칭을 위해 사용되는 벡터 테이블의 기본 어드레스 각각의 엔트리는 한 워드의 길이이다. 인터럽트 로직 유니트(363)에 의해 제공되는 트랩 넘버는 상기 어드레스에 의해 지시되는 테이블로의 인덱스로서 이용된다.
FTB	W	고속 트랩 기본 레지스터: 즉시 트랩 처리 루틴 테이블의 기본 어드레스. 각각의 테이블 엔트리는 32워드이며, 트랩 처리 루틴을 직접 구현하는데 이용된다. 인터럽트 로직 유니트(363)에 의해 제공되는 트랩 넘버에 32를 곱한 값은 이 어드레스에 의해 지시되는 테이블로의 오프셋으로서 이용된다.
PBR	W	절차 기본 레지스터: 절차 루틴 디스패칭을 위해 이용되는 벡터 테이블의 기본 어드레스. 각각의 이용되는 벡터 테이블의 기본 어드레스. 각각의 엔트리는 4개의 워드 경계에 정렬된 1워드 길이이다. 절차 명령어 필드로서 제공되는 절차 넘버는 이 어드레스에 의해 지시되는 테이블로의 인덱스로서 이용된다.
PSR	R/W	프로세서 상태 레지스터: 프로세서 상태 워드를 유지한다. 상태 데이터 비트는, 캐리, 오버플로우, 제로, 네가티브, 프로세서 모드, 현재 인터럽트 레벨, 실행되고 있는 처리 루틴, 0으로 나눔,오버플로우 예외, 하드웨어 기능 인에이블, 절차 인에이블, 인터럽 인에이블을 포함한다.
PPSR	R/W	이전의 프로세서 상태 레지스터: 명령어의 성공적인 완료시, 또는 인터럽트나 트랩이 취해질 때 PSR로부터 로드된다.
CSR	R/W	비교 상태(부울) 레지스터: 단일 워드로서 억세스 가능한 부울 레지스터 셋트.
PCSR	R/W	이전의 비교 상태 레지스터: 명령어의 성공적인 완료시 또는 인터럽트나 트랩이 취해질 때 CSR로부터 로드된다.

2) 정수 데이터 경로 상세:

제10도에는 본 발명의 양호한 실시예에 따라 구성된 IEU(104)의 정수 데이터 경로가 도시되어 있다. 명료성을 위해, 정수 데이터 경로(580)에 대한 많은 제어 경로 접속은 도시되지 않았다.

그러한 접속은 제5도에 정해져 있다.

데이터 경로(580)에 대한 입력 데이터는 정렬(alignment) 유니트(582,584) 및 정수 로드/스토어 유니트(586)로부터 얻어진다. 본래적으로 명령어 내장(embedded) 데이터 필드로서 제공되는 정수 즉시(immediate) 데이터 값은 버스(588)를 통해 오퍼랜드 유니트(470)로부터 얻어진다. 정렬 유니트(582)는 정수 데이터 값을 분리하고, 그 생성 값을 출력 버스(590)를 통해 멀티플렉서(592)로 제공한다. 멀티플렉서(592)로의 제 2 입력은 특수 레지스터 어드레스 및 데이터 버스(354)이다.

명령어 스트림으로부터 얻어지는 즉시 오퍼랜드 또한 데이터 버스(594)를 통해 오퍼랜드 유니트(570)로부터도 얻어진다. 이들 값은 다시 출력 버스(596)에 제공하기 전에 정렬 유니트(584)에 의해 바로 조정된다.

정수 로드/스토어 유니트(586)는 외부 데이터 버스(598)을 통해 CCU(106)와 양방향으로 통신한다. IEU(104)로 들어오는 데이터는 정수 로드/스토어 유니트(586)에 의해 입력 데이터 버스(600)를 통해 입력 랫치(602)에 전달된다. 멀티플렉서(592) 및 랫치(602)로부터 출력된 데이터는 멀티플렉서(608)의 멀티플렉서 입력 버스(604,606) 상에 제공된다. 기능 유니트 출력 버스(482`)로부터의 데이터는 또한 멀티플렉서(608)에 의해서도 수신된다. 본 구조(100)의 양호한 실시예에서, 상기 멀티플렉서(608)는 출력 멀티플렉서 버스(610)로의 2개의 동시 데이터 경로를 제공한다. 더욱이, 멀티플렉서(608)를 통한 데이터의 전달은 시스템 클럭의 각각의 반-사이클내에서 완료될 수 있다. 본 구조(100)에 의해 구현되는 대부분의 명령어는 단일 목적지 레지스터를 이용하며, 최대 4개의 명령어가 각 시스템 클럭 사이클 동안에 임시 버퍼(612)에 데이터를 제공할 수 있다.

임시 버퍼(612)로부터의 데이터는 임시 레지스터 출력 버스(616)를 통해 정수 레지스터 파일 어레이(614)로 전달되거나, 교호(alternate) 임시 버퍼 레지스터 버스(618)를 통해 출력 멀티플렉서(620)로 전달될 수 있다. 정수 레지스터 어레이 출력 버스(622)는 정수 레지스터 데이터가 멀티플렉서(620)로 전달될 수 있도록 한다. 임시 버퍼(612)와 정수 레지스터 파일 어레이(614)에 접속된 출력 버스는 각각 5개의 레지스터 출력값이 동시에 출력될수 있도록 허용한다. 즉, 총 5개까지의 소스 레지스터를 참조하여 2개의 명령어가 동시에 발생될 수 있다. 임시 버퍼(612), 레지스터 파일 어레이(614) 및 멀티플렉서(620)는 외부로 향하는(outbound) 레지스터 데이터 전달이 매 시스템 클럭 반-사이클마다 발생하도록 허용한다. 그러므로, 각 클럭 사이클 동안에 4개까지의 정수 및 부동 소수점 명령어가 발생될 수도 있다.

멀티플렉서(620)는 레지스터 파일 어레이(614)로부터 또는 직접 임시 버퍼(612)로부터 아웃바운드 레지스터 데이터 값을 선택하도록 동작한다. 이것은 이전에 비순서적으로 실행된 명령어에 대한 의존도를 갖는 비순서로 실행된 명령어가 IEU(104)에 의해 실행될 수 있도록 한다. 이것은 실행되고 회수된 명령어에 의해 발생된 데이터 결과로부터 비순서 데이터 결과를 정확하게 분리하면서, 보류 명령어의 비순서 실행에 의한 IEU 정수 데이터 경로의 실행 처리 능력을 최대화하는 이중 목적의 실현을 용이하게 한다. 머신의 정확한 상태가 복원될 필요가 있는 인터럽트 또는 다른 예외조건이 발생할 때마다, 본 발명은 임시 버퍼(612)에 존재하는 데이터 값이 단순히 클리어될 수 있도록 허용한다. 그러므로, 레지스터 파일 어레이(614)는 인터럽트 또는 예외 조건의 발생 이전에 완료되고 회수된 명령어의 실행에 의해서만 발생되는 바로 그 데이터 값을 포함하는 상태로 한다.

멀티플렉서(620)의 각각의 시스템 클럭 반-사이클 동안에 선택되는 5개까지의 레지스터 데이터 값을 멀티플렉서 출력 버스(624)를 통해 정수 바이패스 유니트(626)로 제공된다. 이 바이패스 유니트(626)는 본질적으로, 그 입력중 어느 입력에 제공되는 데이터를 그 출력 중 어느 출력으로 루팅하는 병렬 멀티플렉서 어레이이다. 바이패스 유니트(626)의 입력은 출력 버스(604)를 통한 멀티플렉서(592)로부터의 특수 레지스터 어드레스 지정 데이터 값, 버스(624) 상에 제공되는 최대 5개의 레지스터 데이터 값, 더블 정수 버스(600)를 통한 정수 로드/스토어 유니트(586)로부터의 로드 오퍼랜드 데이터, 정렬 유니트(584)로부터 그 출력 버스(596)를 통해 얻어지는 즉시 오퍼랜드 값 및 마지막으로, 기능 유니트 출력 버스(482)로부터의 바이패스 데이터 경로를 포함한다.

상기 바이패스 데이터 경로 및 데이터 버스(482)는 시스템 클럭 사이클당 4개의 레지스터 값의 동시 전달을 제공한다.

바이패스 유니트(626)에 의해 출력된 데이터는 부동 소수점 데이터 경로에 연결된 정수 바이패스 버스(628)와, 최대 5개의 레지스터 데이터 값의 동시 전달을 제공하는 2개의 오퍼랜드 데이터 버스 및, 정수 로드/스토어 유니트(586)에 데이터를 제공하는데 이용되는 스토어 데이터 버스(632)에 제공된다.

기능 유니트 분배 버스(480)는 루팅 유니트(634)의 연산을 통해 구현된다. 또한 루팅 유니트(634)는 그 입력에서 수신되는 5개의 레지스터 값이 정수 데이터 경로에 제공되는 기능 유니트를 루팅할 수 있도록 허용하는 멀티플렉서의 병렬 어레이에 의해 구현된다. 특히, 루팅 유니트(634)는 바이패스 유니트(626)로부터 버스(630)를 통해 제공되는 5개의 레지스터 데이터 값, 버스(352)를 통한 현재 IF_PC 어드레스 값 및 라인(378`) 상에 제공되는 바와 같이 PC 제어 유니트(362)에 의해 결정되는 제어 흐름 오프셋 값을 수신한다. 루팅 유니트(634)는 부동 소수점 데이터 경로 내에 제공되는 바이패스 유니트로부터 기원된 오퍼랜드 데이터 값을 데이터 버스(636)를 통해 선택적으로 수신할 수도 있다.

루팅 유니트(634)에 의해 수신되는 레지스터 데이터 값은 특수 레지스터 어드레스 및 데이터 버스(354)와 기능 유니트(640,642,644)로 전달될 수도 있다. 특히, 루팅 유니트(634)는 최대 3개의 레지스터 오퍼랜드 값을 루팅 유니트 출력 버스

(646,648,650)를 통해 각각의 기능 유니트(640,642,644)로 제공할 수 있다. 본 구조(100)의 일반적인 구조와 일치하여, 최대 2개의 명령어가 동시에 기능 유니트(640,642,644)로 발생될 수 있다. 본 발명의 양호한 실시예는 프로그램 가능 시프트 기능과 2가지 산술 로직 유니트 기능을 구현하는 3개의 전용 정수 기능 유니트를 제공한다.

ALU0 기능 유니트(644), ALU1 기능 유니트(642) 및 시프터 기능 유니트(640)는 각각의 출력 레지스터 데이터를 기능 유니트 버스(482`)에 제공한다. ALU0 및 시프터 기능 유니트(644,640)에 의해 발생되는 출력 데이터는 또한 부동 소수점 데이터 경로에 결합된 공유 정수 기능 유니트 버스(650)에도 제공된다. 유사한 부동 소수점 기능 유니트 출력값 데이터 버스(652)는 부동 소수점 데이터 경로로부터 기능 유니트 출력 버스(482`)에 제공된다.

ALU0 기능 유니트(644)는 또한 IFU(102)의 프리펫치 연산 및 정수 로드/스토어 유니트(586)의 데이터 연산 모두를 지원하기 위해 가상 어드레스 값의 발생에도 이용된다. ALU0 기능 유니트(644)에 의해 계산된 가상 어드레스 값은 실행 유니트 물리적 어드레스(EX PADDR)를 제공하기 위해 IFU(102)의 목표 어드레스 버스(346)와 CCU(106)에 연결된 출력 버스(654)에 제공된다. ALU0 기능 유니트(644)에 의해 발생되는 어드레스의 가상화(virtualizing) 부분을 저장하기 위해 랫치(656)가 제공된다. 상기 어드레스의 가상화 부분은 출력 버스(658)를 통해 VMU(108)에 제공된다.

3) 부동 소수점 데이터 경로 상세:

이제 제11도를 참조하면, 부동 소수점 데이터 경로(660)가 도시되어 있다. 초기 데이터는 즉시 정수 오퍼랜드 버스(588), 즉시 오퍼랜드 버스(594) 및 특수 레지스터 어드레스 데이터 버스(354)를 포함하는 다수의 소스로부터 수신된다. 외부 데이터의 마지막 소스는 외부 데이터 버스(598)를 통해 CCU(106)에 결합된 부동 소수점 로드/스토어 유니트(662)이다.

즉시 정수 오퍼랜드는 정렬 출력 데이터 버스(668)를 통해 멀티플렉서(666)에 제공하기 전에 정수 데이터 필드를 정확하게 자리 맞춤하는(justify) 기능을 하는 정렬 유니트(664)에 의해 수신된다. 멀티플렉서(666)는 또한 특수 레지스터 어드레스 데이터 버스(354)를 수신한다. 즉시 오퍼랜드는 출력버스(672) 상에 제공되기 전에 정확한 자리맞춤(justification)을 위해 제 2 정렬 유니트(670)에 제공된다. 부동 소수점 로드/스토어 유니트(662)는 로드 데이터 버스(676)로부터의 랫치(674)에 의해 수신된다. 멀티플렉서(666), 랫치(674) 및 기능 유니트 데이터 복귀 버스(482")로부터의 데이터는 멀티플렉서(678)의 입력으로 수신된다. 멀티플렉서(678)는 2개의 레지스터 데이터 값이 시스템 클럭의 각각의 반-사이클마다 멀티플렉서 출력 버스(682)를 통해 임시 버퍼(680)로 기록될 수 있도록 허용하기에 충분한 선택가능 데이터경로를 제공한다. 임시 버퍼(680)는 제6B도에 도시된 바와 같은 임시 버퍼(552`)와 논리적으로 동일한 레지스터 셋트를 포함한다. 상기 임시 버퍼(680)는 또한, 그 임시 버퍼(680)로부터 판독되는 5개까지의 레지스터 데이터 값을 데이터 버스(686)를 통해 부동 소수점 레지스터 파일 어레이(684)에 제공하고, 또한 출력 버스(690)을 통해 출력 멀티플렉서(688)에 제공한다. 상기 멀티플렉서(688)는 또한 부동소수점 레지스터 파일 어레이(684)로부터 5개까지의 레지스터 데이터 값을 동시에 데이터 버스(692)를 통해 수신한다. 상기 멀티플렉서(688)는 데이터 버스(696)를 통한 바이패스 유니트(694)로의 동시 전달을 위해 5개까지의 레지스터 데이터 값을 선택하는 기능을 한다. 바이패스 유니트(694)는 또한, 데이터 버스(672)를 통해 정렬 유니트(670)에 의해 제공되는 즉시 오퍼랜드 값을 수신하며, 멀티플렉서(666)로부터의 출력 데이터 버스(698)와 기능 유니드 데이터 복귀 버스(482")의 데이터 바이패스 연장라인으로부터의 데이터도 수신한다. 바이패스 유니트(694)는 바이패스 유니트 출력 버스(700), 부동 소수점로드/스토어 유니트(662)에 연결된 스토어 데이터 버스(702) 및, 정수 데이터 경로(580)의 루팅 유니트(634)를 연결하는 부동 소수점 바이패스 버스(636)로 출력하기 위해 5개까지의 동시 레지스터 오퍼랜드 데이터 값을 선택한다.

부동 소수점 루팅 유니트(704)는 바이패스 유니트 출력 버스(700) 및 정수 데이터 경로 바이패스 버스(628)와, 각각의 기능 유니트(712,714,716)에 결합된 기능 유니트 입력 버스(706,708,710) 사이에 동시 선택가능 데이터 경로를 제공한다. 본 발명의 양호한 실시예(100)에 따라 각각의 입력 버스(706,708,710)는 3개까지의 레지스터 오퍼랜드 데이터 값이 각각의 기능 유니트(712,714,716)로 동시 전달이 되도록 허용한다. 이러한 기능 유니트(712,714,716)의 출력 버스는 레지스터 파일 입력 멀티플렉서(678)로 데이터를 복귀시키기 위한 기능 유니트 데이터 복귀 버스(482")에 연결된다. 정수 데이터 경로 기능 유니트 출력 버스(650)도 또한 기능 유니트 데이터 복귀 버스(482")에 연결되도록 제공될 수도 있다. 본 구조(100)는 부동 소수점 데이터 경로 기능 유니트 버스(652)를 통해 정수 데이터 경로(580)의 기능 유니트 데이터 복귀 버스(482")에 결합되게 되는 승산기 기능 유니트(712)와 부동 소수점 ALU(714)의 기능 유니트 출력 버스의 접속을 제공한다.

4) 부울 레지스터 데이터 경로 상세:

제12도에는 부울 연산 데이터 경로(720)가 도시되어 있다. 이 데이터 경로(720)는 근본적으로 2가지 형태의 명령어의 실행을 지원하는데 이용된다. 제 1 형태는 정수 레지스터 셋트나 부동 소수점 레지스터 셋트로부터 선택되거나 즉시 오퍼랜드로서 제공되는 2개의 오퍼랜드가 정수 및 부동 소수점 데이터 경로의 ALU 기능 유니트중 한 유니트에서 감산에 의해 비

교된다. 비교는 ALU 기능 유니트(642,644,714,716)중 어느 유니트에 의한 감산 연산에 의해 수행되며, 그 결과로서 생성된 사인(sign) 및 제로 상태 비트는 조합된 입력 선택기 및 비교 연산 유니트(722)에 제공된다. 이 유니트(722)는 EDecode 유니트(490)로부터 수신되는 명령어 식별 제어 신호에 응답하여, ALU 기능 유니트(642,644,714,716)의 출력을 선택하고, 부울 비교 결과 값을 추출하기 위해 사인 및 제로 비트를 조합한다. 출력 버스(723)는 비교 연산의 결과가 입력 멀티플렉서(726)와 바이패스 유니트(742)에 동시에 전달될 수 있도록 허용한다. 정수 및 부동 소수점 데이터 경로에서처럼, 바이패스 유니트(742)는 복수 출력으로 바이패스 유니트(742)의 입력 사이의 복수의 선택가능한 데이터 경로를 제공하는 멀티플렉서의 병렬 어레이로서 구현된다. 바이패스 유니트(742)의 다른 입력은 부울 연산 결과 복귀 데이터 버스(724)와 데이터 버스(744) 상의 2개의 부울 오퍼랜드이다. 상기 바이패스 유니트(742)는 2개까지의 동시 실행 부울 명령어를 나타내는 부울 오퍼랜드가 오퍼랜드 버스(748)를 통해 부울 연산 기능 유니트(746)로 전달될 수 있도록 허용한다. 바이패스 유니트(746)는 또한 제어 흐름 결과 제어 라인(750,752)에 동시에 제공될 2개까지의 단일 비트 부울 오퍼랜드 비트(CF0, CF1)의 전달을 허용한다.

부울 연산 데이터 경로(720)의 잔여부는 비교 결과 버스(723)와 부울 결과 버스(724)에 제공되는 비교 및 부울 연산 결과 값을 그 입력으로서 수신하는 입력 멀티플렉서(726)를 포함한다. 상기 버스(724)는 2개 까지의 동시 부울 결과 비트가 멀티플렉서(726)로 전달될 수 있도록 허용한다. 부가적으로 2개까지의 비교 결과 비트가 버스(723)를 통해 멀티플렉서(726)로 전달될 수도 있다. 멀티플렉서(726)는 그 멀티플렉서 입력에 제공된 2개의 단일 비트가 멀티플렉서 출력 버스(730)를 통해 시스템 클럭의 각각의 반-사이클 동안에 부울 연산 임시 버퍼(728)로 전달될 수 있도록 허용한다. 임시 버퍼(728)는 비록 2가지 중요한 관점에서는 상이하지만, 제6B도에 도시된 바와 같이, 임시 버퍼(752")와 논리적으로는 동등하다. 첫번째 관점은 임시 버퍼(728) 내의 각각의 레지스터 엔트리(entry)가 단일 비트로 이루어진다는 것이다. 두번째로 구별되는 것은 부울 연산의 결과가 정의상 단일 결과 비트에 의해 전적으로 정의 되기 때문에 8개의 보류 명령어 슬롯 각각에 대해 단일 레지스터가 제공된다는 것이다.

상기 임시 버퍼(728)는 4개까지의 출력 오퍼랜드 값을 동시에 제공한다. 이것은 2개의 소스 레지스터에 대한 억세스를 각각 요구하는 2개의 부울 명령어의 동시 실행을 허용한다. 4개의 부울 레지스터 값이 시스템 클럭의 각각의 반-사이클 동안에 오퍼랜드 버스(736)를 통해 멀티플렉서(738)로 전달되거나 부울 오퍼랜드 데이터 버스(734)를 통해 부울 레지스터 파일 어레이(732)로 전달될 수도 있다. 제9도에 논리적으로 묘사된 바와 같이 부울 레지스터 파일 어레이(732)는 단일 4개까지의 단일 비트 위치의 분리된 조합이 임시 버퍼로부터의 데이터로 변형되어 시스템 클럭의 각각의 반-사이클 동안에 부울 레지스터 파일 어레이(732)로부터 출력 버스(740)로 판독될 수 있도록 허용하는 단일 32비트 폭 데이터 레지스터이다. 멀티플렉서(738)는 버스(736,740)를 통해 그 입력으로서 수신되는 2쌍의 부울 오퍼랜드를 오퍼랜드 출력 버스(744)로 전달하여 바이패스 유니트(742)에 제공한다.

부울 연산 기능 유니트(746)는 2개의 소스 값에 대한 넓은 범위의 부울 연산을 수행할 수 있다. 비교 명령어의 경우에, 소스 값은 정수 및 부동 소수점 레지스터 셋트 중 소정의 셋트로부터 얻어지는 한쌍의 오퍼랜드와 IEU(104)에 제공되는 소정의 즉시 오퍼랜드이며, 부울 명령어의 경우에는 부울 레지스터 오퍼랜드중 소정의 2개의 오퍼랜드이다. 테이블 Ⅲ와 Ⅳ에 의해서 본 구조(100)의 양호한 실시예에 의해 제공되는 논리적 비교 연산을 식별 할 수 있다. 테이블 Ⅴ에 의해서는 본 구조(100)의 양호한 실시예에 제공되는 직접 부울 연산을 식별할 수 있다. 테이블Ⅲ-Ⅴ에서 지정된 명령어 조건 코드 및 기능 코드는 대응하는 명령어의 세그먼트(segment)를 나타낸다. 상기 명령어는 또한 오퍼랜드 레지스터의 소스 쌍 및 대응하는 부울 연산 결과의 저장을 위한 목적지 부울 레지스터의 아이덴티피케이션을 제공한다.

테이블 Ⅲ

정수비교

조건	심볼	명령어 조건코드
rs1이 rs2보다 큼	>	0000
rs1이 rs2보다 크거나 동일함	>=	0001
rs1이 rs2보다 작음	<	0010
rs1이 rs2보다 작거나 동일함	>=	0011
rs1이 rs2와 동일하지 않음	!=	0100
rs1이 rs2와 동일함	==	0101
예비용		0110
무조건		1111

rs = 레지스터 소스(register source)

테이블 Ⅳ		
부동 소수점 비교		
조건	심볼	명령어 조건코드
rs1이 rs2보다 큼	>	0000
rs1이 rs2보다 크거나 동일함	>=	0001
rs1이 rs2보다 작음	<	0010
rs1이 rs2보다 작거나 동일함	>=	0011
rs1이 rs2와 동일하지 않음	!=	0100
rs1이 rs2와 동일함	==	0101
순서가 정해지지 않음	?	1000
순서가 정해지지 않거나, rs1이 rs2보다 큼	?>	1001
순서가 정해지지 않거나, rs1이 rs2보다 크거나 동일함	?>=	1010
순서가 정해지지 않거나, rs1이 rs2보다 작음	?<	1011
순서가 정해지지 않거나, rs1이 rs2보다 작거나 동일함	?<=	1100
순서가 정해지지 않거나, rs1이 rs2와 동일함	?=	1101
예비용		1110-1111

테이블 Ⅴ		
부울 연산		
연산	심볼	명령어 기능코드
0	Zero	0000
bs1&bs2	AND	0001
bs1&-bs2	AND2	0010
bs1	bs1	0011
-bs1&bs2	ANN1	0100
bs2	bs2	0101
bs1^bs2	XOR	0110
bs1:bs2	OR	0111
-bs1and-bs2	NOR	1000
-bs1^bs2	XNOR	1001
-bs2	NOT2	1010
bs1:-bs2	ORN2	1011
-bs1	NOT1	1100
-bs1:bs2	ORN1	1101
-bs1:-bs2	NAND	1110
1	ONE	1111

bs = 부울 소스 레지스터 (boolean source register)

B) 로드/스토어 제어 유니트:

제13도에는 한 예시적인 로드/스토어 유니트(760)가 도시되어 있다. 비록 데이터 경로 (580,660)에서는 분리적으로 도시되어 있지만, 로드/스토어 유니트(586,662)는 단일 공유 로드/스토어 유니트(760)로서 구현되는 것이 바람직하다. 각각의 데이터 경로(580,660)로부터의 인터페이스는 어드레스 버스(762)와 로드 및 스토어 데이터 버스(764(600,676), 766 (632,702))를 통해 이루어진다.

로드/스토어 유니트(760)에 의해 이용되는 어드레스는 IFU(102)와 IEU(104)의 잔여부에 의해 이용되는 가상 어드레스와 반대되는 물리적 어드레스이다. IFU(102)는 물리적 어드레스를 발생하기 위해 CCU(106)와 VMU(108) 사이의 코디네이션(coordination)에 의지하여 가상 어드레스에 대해 동작하지만, IEU(104)는 로드/스토어 유니트(760)가 물리적 어드레스 모드에서 직접 동작하는 것을 요구한다. 이 요구는 병행(overlapping) 물리적 어드레스 데이터 로드 및 스토어 연산을 포함할 수도 있는 비순서적으로 실행된 명령어가 존재하는 경우 및 CCU(106)으로부터 로드/스토어 유니트(760)로의 비순서 데이터 복귀가 존재하는 경우에 데이터 보전성(integrity)을 확실하게 보증하는데 필요하다. 데이터 보전성을 확실하게 보증하기 위해, 로드/스토어 유니트(760)는 스토어 명령어가 IEU(104)에 의해 회수될때까지 스토어 명령어에 의해 제공되는 데이터의 버퍼링을 수행한다. 결과적으로, 로드/스토어 유니트(760)에 의해 버퍼링된 스토어 데이터는 로드/스토

어 유니트(760)내에만 유일하게 존재한다. 실행되었지만 회수되지 않은 스토어 명령어와 동일한 물리적 어드레스를 참조하는 로드 명령어는 스토어 명령어가 실제적으로 회수될 때까지 지연된다. 이 지점에서 스토어 데이터는 로드/스토어 유니트(760)에 의해 CCU(106)로 전달되어, CCU 데이터 로드 연산의 실행에 의해 즉시 로드될 수도 있다.

특히, 완전(full) 물리적 어드레스는VMU(108)로부터 로드/스토어 어드레스 버스(762)로 제공된다. 로드 어드레스는 일반적으로 로드 어드레스 레지스터(768_{0-3})에 저장된다. 스토어 어드레스는 스토어 어드레스 레지스터(770_{3-0})로 랫치된다. 로드/스토어 제어 유니트(774)는 레지스터(768_{3-0}, 770_{3-0})로의 로드 및 스토어 어드레스의 랫칭을 코디네이트하기 위해 명령어 발생 유니트(498)로부터 수신되는 제어신호에 응답하여 동작한다. 상기 로드/스토어 제어 유니트(774)는 제어 라인(778) 상에 로드 어드레스를 랫치하기 위한 제어 신호를 제공하고, 제어 라인(780) 상에는 스토어 어드레스를 랫치하기 위한 제어신호를 제공한다. 스토어 데이터는 스토어 데이터 레지스터 셋트(782_{3-0})의 논리적으로 대응하는 슬롯에서 스토어 어드레스를 랫치하는 것과 동시에 랫치된다. 4x4x32 비트 폭의 어드레스 비교기 유니트(772)에는 로드 및 스토어 어드레스 레지스터(768_{3-0}, 770_{3-0})내의 각각의 어드레스가 동시에 제공된다. 시스템 클럭의 각각의 반-사이클 동안의 완전 매트릭스 어드레스 비교의 실행은 제어 라인(776)을 통해 로드/스토어 제어 유니트(774)에 의해 제어된다. 스토어 어드레스에 부합하는 로드 어드레스의 존재 및 논리적 위치는 제어 라인(776)을 통해 로드 스토어 제어 유니트(774)에 복귀되는 제어 신호에의해 제공된다.

로드 어드레스가 VMU(108)로부터 제공되고 보류중인 스토어가 없는 경우에, 로드 어드레스는 CCU 로드 연산의 초기화와 동시에 버스(762)로부터 어드레스 선택기(786)로 직접 바이패스된다. 그러나, 스토어 데이터가 보류중인 경우에는, 로드 어드레스는 이용 가능한 어드레스 랫치(768_{0-3})에서 랫치되게 된다. 대응하는 스토어 데이터 명령어가 회수중이라는 것을 나타내는 회수 제어 유니트(500)로부터의 제어 신호의 수신시, 로드/스토어 제어 유니트(774)는 CCU(106)로의 억세스를 위해 제어 라인(784)을 통해, 중재(arbitrating)에 의해 CCU 데이터 전달 동작을 초기화한다.

CCU(106)가 준비되었다는 신호를 하면, 로드/스토어 유니트(774)는 선택기(786)로 하여금 CCU PADDR 어드레스 버스(788) 상에 CCU 물리적 어드레스를 제공하도록 지시한다. 이 어드레스는 어드레스 버스(790)를 통해 대응하는 스토어 레지스터(770_{3-0})로부터 얻어진다. 대응하는 스토어 데이터 레지스터(782_{3-0})로부터의 데이터는 CCU 데이터 버스(792) 상에 제공된다.

명령어 발생기(498)에 의한 로드 명령어의 발생시, 로드/스토어 제어 유니트(774)는 요청된 로드 어드레스를 랫치하기 위해 로드어드레스 랫치(768_{3-0})중 하나를 인에이블시킨다. 선택된 특수랫치(768_{0-3})는 관련 명령어 셋트내의 로드 명령어의 위치에 논리적으로 대응한다. 명령어 발생기(498)는 2개의 가능한 보류 명령어 셋트중 한 셋트내의 로드 명령어를 식별하는 5-비트 벡터를 로드/스토어 제어 유니트(774)에 제공한다. 비교기(772)가 부합되는 스토어 어드레스를 식별하지 못하는 경우에, 로드 어드레스는 CCU PADDR 어드레스 버스(788)로의 출력을 위해 어드레스 버스(794)를 통해 선택기(786)로 루팅된다. 어드레스의 제공은 로드/스토어 제어 유니트(774)와 CCU(106) 사이에서 교환되는 CCU 요청 및 준비 제어 신호와 제휴하여 수행된다. 실행 ID 값(Ex ID)은 또한 CCU(106)가 계속해서 Ex ID 값을 포함하는 요청된 데이터를 복귀시킬때 로드 요청을 식별하기 위해 CCU(106)로 로드/스토어 제어 유니트(774)에 의해 준비 및 발생된다. 상기 ID 값은 현재 로드 요청이 발생되는 각각의 로드 어드레스 랫치(768_{0-3})를 식별하기 위해 고유의 비트를 이용하는 4비트 벡터로 이루어진다. 5번째 비트는 로드 명령어를 포함하는 명령어 셋트를 식별하기 위해 이용된다. 그러므로, ID 값은 명령어 발생기 유니트(498)로부터의 로드 요청이 제공된 비트 벡터와 동일하다.

CCU(106)로부터 로드/스토어 제어 유니트(774)로 이전의 요청된 로드 데이터의 유효성에 관한 후속 신호시, 로드/스토어 제어 유니트(774)는 데이터를 수신하고 그 데이터를 로드 데이터 버스(764) 상에 제공하도록 정렬 유니트를 인에이블시킨다. 정렬 유니트(798)는 로드 데이터를 정확하게 자리 맞추는 동작을 한다.

CCU(106)로부터의 데이터의 복귀와 동시에, 로드/스토어 제어 유니트(774)는 CCU(106)로부터 Ex ID 값을 수신한다. 로드/스토어 제어 유니트(774)는 다음에, 로드 데이터가 로드 데이터 버스(764) 상에 제공되고 있다는 것을 식별하는 제어 신호를 명령어 발생기 유니트(498)에 제공하고, 또한 로드 데이터가 복귀되고 있는 로드 명령어를 식별하는 비트 벡터를 복귀시킨다.

C) IEU 제어 경로 상세:

다시 제5도를 참조하면, 이제 IEU 제어 경로의 동작이 제14도에 제공된 타이밍도에 관해 상세하게 설명되게 된다. 제14도에 도시된 명령어 실행의 타이밍은 본 발명의 동작의 예시적인 것이며, 실행 타이밍 순열(permutations)의 전부는 아니다.

제14도의 타이밍도는 일련의 프로세서 시스템 클럭 사이클 P_{0-6}을 도시하고 있다. 각각의 프로세서 사이클은 내부 T 사이클 T_0로 시작된다. 본 구조(100)에 의해 제공되는 바와 같이, 본 발명의 양호한 실시예에서는 프로세서 사이클당 2개 T사이클이 있다.

프로세서 사이클 제로에서, IFU(102) 및 VMU(108)는 물리적 어드레스를 발생하도록 동작한다. 이 물리적 어드레스는 CCU(106)에 제공되고, 명령어 캐쉬 억세스 동작이 개시된다. 요청된 명령어 셋트가 명령어 캐쉬(132)에 존재하는 경우에, 명령어 셋트는 프로세서 사이클 1의 약 중간지점에서 IFU(102)로 복귀된다. 다음에, IFU(102)는 프리펫치 유니트(260)와 IFIFO(264)를 통한 명령어 셋트의 전달을 관리하며, 그 후에 명령어 셋트는 실행을 위해 IEU(104)로 먼저 제공된다.

1) 디코드(EDecode) 유니트 상세:

디코드(EDecode) 유니트(490)는 프로세서 사이클 1의 종료 이전의 디코딩을 위해 병렬로 완전 명령어 셋트를 수신한다. 양호한 실시예에서 EDecode 유니트(490)는 버스(124)를 통해 수신되는 모든 유효 명령어의 직접 병렬 디코딩을 제공하는 순수 조합 로직 블록으로서 구현된다. 명령어, 레지스터 요구 조건 및, 자원 요구(needs)의 명세를 포함하여 본구조(100)에 의해 인식되는 명령어의 각각의 형태는 테이블 VI에서 식별된다.

테이블 VI

명령어	제어 및 오퍼랜드 정보
레지스터-레지스터 이동	논리/산술 기능 코드: 가산, 감산, 승산, 시프트등을 지정한다. 목적지 레지스터 PSR만을 셋트 소스 레지스터 1 소스 레지스터 2 또는 즉시 상수 값 레지스터 셋트 A/B 선택
즉시-레지스터 이동	목적지 레지스터 즉시 정수 또는 부동 소수점 상수 값 레지스터 셋트 A/B 선택
로드/스토어 레지스터	연산기능 코드: 로드 또는 스토어를 지정한다. 즉시, 값, 기본 및 즉시 값, 또는 기본 및 오프셋 이용. 원천지/목적지 레지스터 기본 레지스터 인덱스 레지스터 또는 즉시 상수 값 레지스터 셋트 A/B 선택
즉시 호출	신호된 즉시 변위
제어 흐름	연산 기능 코드: 분기 형태 및 트리거링을 지정한다. 기본 레지스터, 즉시 상수 변위 값, 또는 트랙 넘버 레지스터 셋트 A/B 선택
특수 레지스터 이동	연산 기능 코드: 특수/정수 레지스터로 /로부터 이동을 지정 특수 레지스터 어드레스 식별자 원천지/목적지 레지스터 레지스터 셋트 A/B 선택
변환 정수 이동	연산 기능 코드: 부동 소수점-정수 형태 변환을 지정한다. 원천지/목적지 레지스터 레지스터 셋트 A/B 선택
부울 기능	부울 기능 코드: 가산, 논리합 등을 지정한다. 목적지 부울 레지스터 원천지 레지스터 1 원천지 레지스터 2 레지스터 셋트 A/B 선택
확장 절차	절차 지정자: 절차 기본값으로부터 어드레스 오프셋을 지정한다. 연산: 절차 루틴으로 통과되는 값
아토믹 절차	절차 지정자: 어드레스 값을 지정한다.

* 명령어는 명령어를 식별하기 위해 디코드한 필드뿐 아니라 이러한 필드들을 포함한다.

디코드(EDecode) 유니트(490)는 명령어 셋트의 각각의 명령어를 병렬로 디코드한다. 명령어, 명령어 기능, 레지스터 참조, 기능 요구조건(requirements)의 최종 아이덴티피케이션은 디코드(EDecode) 유니트(490)의 출력에서 이용 가능하다. 이 정보는 명령어 셋트내의 모든 명령어가 회수될때까지 각각의 프로세서 반-사이클 동안에 디코드 유니트(490)에 의해 재생 및 랫치된다. 그러므로, 모든 8개의 보류 명령어에 관한 정보는 디코드 유니트(490)의 출력에서 일정하게 유지된다. 이 정보는 각 벡터의 비트 또는 서브-필드가 2개의 보류 명령어 셋트내의 대응하는 명령어의 물리적 위치에 논리적으로 대응하도록 되어 있는 8개의 요소(element) 비트 벡터의 형태로 제공된다. 그러므로, 8개의 벡터는 제어 라인(502)을 통해 캐리 검사 유니트(492)에 제공되며, 여기서 각각의 벡터는 대응하는 명령어가 프로세서 상태 워드의 캐리 비트에 영향을 주거나 또는 의존하는지를 지정한다. 8개의 벡터는 각각의 명령어의 특정 성질 및 기능 유니트 요구조건을 식별하기 위해 제어 라인(510)을 통해 제공된다. 8개의 보류 명령어의 각각에 의해 이용되는 레지스터 참조를 지정하는 8개의 벡터는 제어 라인(506)을 통해 제공된다. 이들 벡터는 프로세서 사이클 1의 종료 이전에 제공된다.

2) 캐리 검사 유니트 상세:

캐리 검사 유니트(492)는 제14도에 도시된 연산의 데이터 의존상태 단계동안에 의존도 검사 유니트(494)와 병렬로 동작한다.

상기 캐리 검사 유니트(492)는 양호한 구조(100)에서 순수 조합로직으로서 구현된다. 그러므로, 캐리 검사 유니트(492)에 의한 각각의 반복 연산중에, 모든 8개의 명령어는 그것이 프로세서 상태 레지스터의 캐리 플래그를 변형하는지에 관해 고려된다. 이것은 이전의 명령어에 의해 셋트되는 바와 같은 캐리 비트의 상태에 의존하는 명령어의 비순서적 실행을 허용하기 위해 필요하다. 제어 라인(504) 상에 제공되는 제어 신호는 캐리 검사 유니트(492)로 하여금 캐리 플래그에 관해 이전의 명령어의 실행에 의존하는 특정 명령어를 식별할수 있도록 허용한다.

또한, 캐리 검사 유니트(492)는 8개의 보류 명령어 각각에 대해 캐리 비트의 임시 카피(copy)를 유지한다. 캐리 비트를 변형시키지 않는 그런 명령어에 대해, 캐리 검사 유니트(492)는 프로그램 명령어 스트림의 순서에 있어서 앞으로 다음 명령어로 캐리 비트를 전파한다. 그러므로, 캐리 비트를 변형시키는 비순서적 실행 명령어가 실행될 수 있으며, 더욱이, 이와 같은 비순서적 실행 명령어에 의존하는 후속 명령어도 비록 캐리 비트를 변형시키는 명령어 다음이긴 하지만 실행이 허용될 수도 있다. 또한, 이와 같은 명령어의 회수 이전에 발생하는 예외가 단지 캐리 검사 유니트(492)로 하여금 내부 임시 캐리 비트 레지스터를 클리어하도록 하는 것을 요구하기 때문에 캐리 검사 유니트(492)에 의한 캐리 비트의 유지는 비순서적 실행을 용이하게 한다. 결과적으로, 프로세서 상태 레지스터는 비순서적으로 실행되는 명령어의 실행에 의해 영향받지 않는다. 캐리 검사 유니트(492)에 의해 유지되는 임시 비트 캐리 레지스터는 각각의 비순서적으로 실행되는 명령어의 완료시 갱신된다. 비순서적으로 실행되는 명령어의 회수시, 프로그램 명령어 스트림내의 최종 회수된 명령어에 대응하는 캐리 비트는 프로세서 상태 레지스터의 캐리 비트 위치로 전달된다.

3) 데이터 의존도 검사 유니트 상세:

데이터 의존도 검사 유니트(494)는 제어 라인(506)을 통해 디코드 유니트(490)으로부터 8개의 레지스터 참조 아이덴티피케이션 벡터를 수신한다. 각각의 레지스터 참조는 한번에 32개의 레지스터중 하나를 식별하기 위해 적합한 5비트 값과, "A", "B" 또는 부울 레지스터 셋트내에 위치된 바와 같은 레지스터 뱅크를 식별하는 2비트 값에 의해 표시된다. 부동 소수점 레지스터 셋트는 "B" 레지스터 셋트와 동등하게 식별된다. 각각의 명령어는 3개의 레지스터 참조 필드, 즉 2개의 원천지 레지스터 필드 및 하나의 목적지 레지스터 필드를 가질 수도 있다. 비록 몇몇의 명령어, 그중에서도 특히 레지스터-레지스터 이동 명령어는 목적지 레지스터를 지정할 수도 있지만, 디코드 유니트(490)에 의해 인식되는 명령어 비트 필드는 발생될 실제 출력 데이터가 없다는 것을 나타낼 수도 있다. 오히려, 이 명령어의 실행은 오직 프로세서 상태 레지스터의 값의 변경을 결정하는 목적을 위한 것이다.

본 양호한 구조(100)에서 역시 순수 조합 로직으로서 구현되는 데이터 의존도 검사 유니트(494)는 프로그램 명령어 스트림에서 후속의 명령어의 원천지 어드레스 참조와 비교적 이전의 명령어의 목적지 레지스터 참조 사이의 의존도를 동시에 결정하도록 동작한다. 비트 어레이는 어느 명령어가 다른 명령어에 의존하는지 식별할 뿐만 아니라 각각의 의존도가 발생하는 레지스터를 식별하는 데이터 의존도 검사 유니트(494)에 의해 발생된다.

캐리 및 레지스터 데이터 의존도는 제 2 프로세서 사이클의 개시 이후에 바로 식별된다.

4) 레지스터 리네임 유니트 상세:

레지스터 리네임 유니트(496)는 제어 라인(506)을 통해 모든 8개의 보류 명령어의 레지스터 참조의 아이덴티피케이션을 수신하고, 제어 라인(508)을 통해 레지스터 의존도를 수신한다. 실행이 완료된 보류 명령어의 현제 셋트내의 이와 같은 명령어를 식별하는 8개의 요소의 매트릭스도 또한 제어 라인(542)을 통해 수신된다. 이 정보로부터, 레지스터 리네임 유니트(496)는 제어 신호의 8 요소 어레이를 제어 라인(512)을 통해 명령어 발생 유니트(498)에 제공한다. 이렇게 해서 제공되는 제어 정보는 이미 실행이 완료되지 않은 현재 보류중인 명령어중 어느 것이 식별된 데이터 의존도의 현재 셋트내에서 지금 실행에 이용가능한지에 관한 레지스터 리네임 유니트(496)에 의해 이루어지는 결정을 반영한다. 레지스터 리네임 유니트(496)는 실행을 위해 동시에 발생되게 될 6개까지의 명령어, 즉 2개의 정수, 2개의 부동 소수점 및 2개의 부울을 식별하는 선택 제어신호를 라인(516)을 통해 수신한다.

레지스터 리네임 유니트(496)는 버스(518) 상에서 레지스터 파일 어레이(472)에 제공되는 제어 신호에 의해, 식별된 명령어의 실행에서의 억세스를 위해 원천지 레지스터를 선택하는 추가 기능을 수행한다. 비순서적으로 실행되는 명령어에 대한 목적지 레지스터는 대응하는 데이터 경고의 임시 버퍼(612.680.728)에 있는 것으로서 선택된다. 정순서적으로 실행되는 명령어는 결과 데이터가 레지스터 파일(614,684,732)로 저장되면서 완료되었을때 회수된다. 원천지 레지스터의 선택은 레지스터가 목적지로서 이전에 선택되었는지 또한 대응하는 이전의 명령어가 아직 회수되지 않았는지에 의존한다. 이와같은 경우에, 원천지 레지스터는 대응하는 임시 버퍼(612,680,728)로부터 선택된다. 이전의 명령어가 회수되면, 대응하는 레지스터 파일(614,684,732)의 레지스터가 선택된다. 결과적으로, 레지스터 리네임 유니트(496)는 비순서적으로 실행되는 명령어의 경우에 레지스터 파일 레지스터 참조를 임시 버퍼 레지스터 참조로 효과적으로 대체하도록 동작한다.

본 구조(100)에서 구현되는 바와 같이, 임시 버퍼(612,680,728)는 그 대응하는 레지스터 파일 어레이의 이중 레지스터 구조가 아니다. 오히려, 8개의 보류 명령어 각각에 대해 단일 목적지 레지스터 슬롯이 제공된다. 결과적으로, 임시 버퍼 목적지 레지스터 참조의 대체는 보류 레지스터 셋트내의 대응하는 명령어의 위치에 의해 결정된다. 후속 원천지 레지스터 참조는 원천지 의존도가 발생하는 명령어에 관해 데이터 의존도 검사 유니트(494)에 의해 식별된다. 그러므로, 임시 버퍼 레지스터내의 목적지 슬롯은 레지스터 리네임 유니트(496)에 의해 쉽게 결정될 수 있다.

5) 명령어 발생 유니트 상세:

명령어 발생 유니트(498)는 디코드(EDecode) 유니트(490)에 의해 식별되는 바와 같은 명령어의 기능 요구조건 및 레지스터 리네임 유니트(496)의 출력에 근거하여, 발생될 수 있는 명령어 셋트를 결정한다. 상기 명령어 발생 유니트(498)는 제어 라인(514)을 통해 보고되는 바와 같은 기능 유니트(478_{0-n})의 각각의 상태를 기초로 하여 그 결정을 한다. 그러므로, 명령어 발생 유니트(498)는 레지스터 리네임 유니트(496)로부터 발생을 위해 명령어의 이용 가능한 셋트를 수신했을때 동작을 시작한다. 각각의 명령어의 실행을 위해 레지스터 파일 어레이가 요구된다고 가정하면, 명령어 발생 유니트(498)는 명령어를 현재 실행하고 있을 수도 있는 기능 유니트(478_{0-n})의 유효성을 기대한다. 레지스터 리네임 유니트(496)로 발생될 명령어를 식별하는데 있어서의 지연을 최소화 시키기 위해, 명령어 발생 유니트(498)는 전용 조합 로직으로 구현된다.

발생할 명령어의 식별시, 레지스터 리네임 유니트(496)는 제3 프로세서 사이클 P_2의 종료시까지 계속되는 레지스터 파일 억세스를 개시한다. 프로세서 사이클 P_3의 개시점에서, 명령어 발생 유니트(498)는 레지스터 파일 어레이(472)로부터 제공되는 원시 데이터를 수신 및 처리하기 위해 "실행0"로서 도시된 바와 같이 기능 유니트(478_{0-n})중 하나 또는 그 이상의 유니트에 의한 동작을 개시한다.

통상적으로, 본 구조(100)에서 처리되는 대부분의 명령어는 당일 프로세서 사이클에서 한 기능 유니트를 통해 실행된다. 그러나 몇몇의 명령어는 동시에 발생되는 명령어를 "실행1" 로서 도시된 바와 같이 완료하기 위해 복수의 프로세서 사이클을 필요로 한다.

실행0 및 실행1 명령어는 예를 들어, ALU 및 부동소수점 승산기 기능 유니트에 의해 각각 실행될 수도 있다. 제14도에 도시된 바와 같이 ALU기능 유니트는 제5 프로세서 사이클 P_4동안에 다른 명령어의 실행에 이용가능한 단순한 출력 랫칭의 제공에 의해 한 프로세서 사이클내에서 출력 데이터를 발생한다. 부동 소수점 승산 기능 유니트는 내부적으로 파이프 라인된 기능 유니트가 되는 것이 바람직하다. 그러므로, 다음 프로세서 사이클에서 다른 추가의 부동 소수점 승산 명령어가 발생될 수 있다.

그러나, 제1명령어의 결과는 데이터에 의존하는 수의 프로세서사이클에 이용가능하지 않게 된다. 제14도에 도시된 명령어는 기능 유니트를 통한 처리를 완료하기 위해 3개의 프로세서 사이클을 필요로 한다.

각각의 프로세서 사이클 동안에 명령어 발생 유니트(498)의 기능이 반복된다. 결과적으로, 기능 유니트(478_{0-n})의 전 셋트의 유효성 상태 뿐아니라 보류 명령어의 현재 셋트의 상태도 각 프로세서 사이클 동안에 재평가 된다. 그러므로 최적의 조건하에서, 상기 양호한 구조(100)는 프로세서 사이클당 최대 6개의 명령어를 실행할 수 있다. 그러나, 통상적인 명령어 믹스(mix)는 프로세서 사이클당 1.5내지 2.0 명령어의 전체적인 평균 실행을 초래하게 된다.

명령어 발생 유니트(498)의 기능에서 마지막으로 고려 할 점은 트랩 조건의 처리 및 특정 명령어의 실행에의 참여이다. 트랩 조건의 발생은 IEU(104)가 아직 회수되지 않은 모든 명령어에 대해 클리어되는 것을 필요로한다. 이와같은 상황은 산술오류에 응답하여 기능 유니트(478_{0-n})중 어느 유니트로부터, 또는 불법 명령어의 디코딩시 디코드 유니트(490)로부터 인터럽트 요청/승인 제어 라인(340)을 통해 IEU(104)로 중계되는 외부적으로 수신된 인터럽트에 응답하여 발생할 수 도 있다. 트래 조건의 발생시, 명령어 발생 유니트(498)는 IEU(104)에서 현재 보류중인 모든 회수되지 않은 명령어를 정지(halting)시키거나 무효화시킬(voiding) 책임이 있다. 동시에 회수될 수 없는 모든 명령어는 무효로 되게 된다.

이 결과는 프로그램 명령어 스트림의 통상적인 정순서 실행에 관해 인터럽트 발생이 정확성을 유지하는데 필수적이다. IEU(104)가 트랩 처리 프로그램 루틴의 실행을 시작할 준비가 되어 있으면, 명령어 발생기(498)는 제어 라인(340)에 따른 복귀 제어 신호를 통해 인터럽트를 승인한다. 또한 명령어가 표준적인 순수 정순서 루틴에서 실행을 완료하게 되기저에 변화되게 되는 프로세서 상태 비트에 근거하여 한 명령어에 대한 예외 조건이 인식될 수도 있는 가능성을 피하기위해, 명령어 발생기(498)는 PSR을 변경시킬 수 있는 모든 명령어(특수 이동, 트랩으로부터의 복귀등과 같음)가 정순서로 엄격하게 실행될 수 있도록 보장할 책임이 있다.

프로그램 제어 흐름을 변경시키는 명령어는 디코드(IDecode) 유니트(262)에 의해 식별되지 않는다. 이런 형태의 명령어는 서브 루틴 복귀, 절차 명령어로부터의 복귀, 및 트랩으로부터의 복귀를 포함한다. 명령어 발생 유니트(498)는 식별 제어 신호를 IEU 복귀 제어 라인(350)을 통해 IFU(102)에 제공한다. 호출 명령, 트랩의 발생 또는 절차 명령어를 조우했을때의 지점에서 존재하는 IF_PC 실행 어드레스를 제공하기위해 특수 레지스터(412)중 대응하는 레지스터가 선택된다.

6) 완료 제어 유니트 상세:

완료 제어 유니트(540)는 그 현재 연산의 완료 상태에 대해 기능 유니트(478_{0-n})를 모니터한다. 본 발명의 양호한 구조(100)에서, 완료 제어 유니트(540)는 명령어의 현재 보류중인 셋트 내의 각각의 명령어의 실행 상태를 반영하는 완료 벡터를 기능 유니트(478_{0-n})에 의한 명령어의 실행 완료 이전에 대략 프로세서 사이클의 반-사이클동안 레지스터 리네임 유니트(496), 바이패스 제어 유니트(520) 및 회수 제어 유니트(500)로 제공하기에 충분한 각 기능 유니트에 의한 연산의 완료를 기대한다.

이것은 명령어 발생 유니트(498)로 하여금 레지스터 리네임 유니트(496)를 통해 다음 명령어 발생 사이클 동안 이용가능한 자원으로서 명령어 완료 기능 유니트를 고려할 수 있도록 허용한다.

바이패스 제어 유니트(520)는 기능 유니트에 의해 출력되는 데이터를 바이패스 유니트(474)를 통해 바이패스 하기위한 준비를 하도록 허용한다. 마지막으로, 회수 제어 유니트(500)는 기능 유니트(478_{0-n})로부터 레지스터 파일 어레이(472)로의 데이터의 전달과 동시에 대응하는 명령어를 회수하도록 동작할 수도 있다.

7) 회수 제어 유니트 상세:

완료 제어 유니트(540)로부터 제공되는 명령어 완료 벡터에 더하여, 회수 제어 유니트(500)는 디코드 출력(490)으로부터 출력되는 가장 오래된 명령어 셋트를 모니터한다. 명령어 스트림 순서의 각 명령어가 완료 제어 유니트(500)에 의해 완료 표시됨에 따라, 회수 제어 유니트(500)는 제어 라인(534)상에 제공되는 제어 신호에 의해, 임시 버퍼 슬롯으로부터 레지스터 파일 어레이(472)내의 대응하는 명령어 지정 레지스터 파일 레지스터 위치로의 데이터의 전달을 지시한다. PC 증분/사이즈 제어 신호는 동시에 회수되는 하나 또는 그 이상의 명령어 각각에 대해 제어 라인(344)상에 제공된다. 매 프로세서 사이클 당 4개까지의 명령어가 회수될 수도 있다. 전체 명령어 셋트가 회수될 때마다, IFIFO판독제어 신호가 IFIFO(264)를 진척시키기 위해 제어 라인(342)상에 제공된다.

8) 제어 흐름 제어 유니트 상세:

제어 흐름 제어 유니트(528)는 현재 보류중인 명령어 셋트 내의 제어 흐름 명령어가 해결되었는지, 또한 분기 결과가 취해졌는지의 여부를 지정하는 정보를 IFU(102)에 연속적으로 제공하도록 동작한다. 상기 제어 흐름 제어 유니트(528)는 제어 라인(510)을 통해 디코드 유니트(490)에 의한 제어 흐름 분기 명령어의 아이덴티피게이션을 얻는다. 레지스터 의존도를 갖는 현재 셋트는 제어 흐름 제어 유니트(528)가 분기 명령어의 결과가 의존도에 의해 제한되는지 또는 현재 알려져 있는지를 결정하도록 허용하기위해 제어 라인(536)을 통해 데이터 의존도 검사 유니트(494)로부터 제어 흐름 제어 유니트(528)에 제공된다. 레지스터 리네임 유니트(496)로부터 버스(518)를 통해 제공되는 레지스터 참조는 분기결정을 정의하게되는 부울 레지스터를 식별하기 위해 제어 흐름 제어 유니트(528)에 의해 모니터 된다. 그러므로, 분기 결정은 제어 흐름 명령어의 비순서적 실행 훨씬 이전에 결정될 수도 있다.

제어 흐름 명령어의 실행과 동시에, 바이패스 유니트(472)는 제어 흐름 0 및 제어 흐름 1 제어 라인(750,752)으로 이루어진 제어 라인(530)을 통해 제어 흐름 제어 유니트(528)로 제어 흐름 결과를 제공하도록 바이패스 제어 유니트(472)에 의해 제어된다. 마지막으로, 제어 흐름 제어 유니트(528)는 8비트로된 2개의 벡터를 각각 제어 라인(348)을 통해 IFU(102)에 연속해서 제공한다. 이들 벡터는 벡터내의 비트에 대응하는 대응 논리적 위치에서의 분기 명령어가 해결되었는지, 또한 분기 결과가 취해졌는지 또는 취해지지 않았는지를 정의한다.

본 양호한 구조(100)에서, 제어 흐름 제어 유니트(528)는 제어유니트(528)로의 입력 제어 신호에 응답하여 연속적으로 동작하는 순수 조합 로직으로서 구현된다.

9) 바이패스 제어 유니트 상세:

명령어 발생 유니트(498)는 레지스터 파일 어레이(472) 및 기능 유니트(478_{0-n}) 사이의 데이터의 루팅을 제어하기위해 바이패스 제어 유니트(520)와 밀접하게 협력하여 동작한다.

바이패스 제어 유니트(520)는 제 14도에 도시된 연산의 레지스터 파일 억세스, 출력 및 스토어단계와 관련하여 동작한다. 레지스터 파일 억세스동안에, 바이패스 제어 유니트(520)는 명령어의 실행의 출력 단계동안에 기록되고있는 과정중에 있는 레지스터 파일 어레이(472)내의 목적지 레지스터의 억세스를 제어 라인(522)을 통해 인식할 수도 있다. 이 경우에, 바이패스 제어 유니트(520)는 기능 유니트 출력 버스(482)상에 제공되는 데이터의 선택이 기능 유니트 분배버스(480)로 되돌려 바이패스되도록 한다.

바이패스 유니트(520)에 대한 제어는 제어 라인(542)을 통해 명령어 발생 유니트(498)에 의해 제공 된다.

Ⅳ. 가상 메모리 제어 유니트:

제 15도에는 VMU(108)에 대한 인터페이스 정의가 제공되어 있다. VMU(108)는 주로 VMU 제어 로직 유니트(800)와 내용 어드레스 가능 메모리(content addressable memory)(CAM)(802)로 이루어 진다. 제 16도에는 VMU(108)의 일반적 기능이 그래프식으로 도시되어 있다. 여기서, 가상 어드레스의 표기는 공간 식 별자(sID[31:28]), 가상 페이지 넘버(VADDR[27:14]), 페이지 오프셋(PADDR[13:4]), 및 요청 ID(rID[3:0])으로 분할되어 도시되어 있다. 물리적 어드레스를 발생하기 위한 알고리즘은 공간 테이블(842)내의 레지스터 중 하나를 선택하기 위해 공간 ID를 이용하는 것이다. 가상 페이지 넘버와 조합하여 선택된 공간 레지스터의 내용은 테이블 룩 어사이드(look aside)버퍼(TLB)(844)를 억세스하기 위한 어드레스로서 이용된다. 32비트 어드레스는 버퍼(844)내의 대응하는 버퍼레지스터를 식별하는데 이용되는 내용 어드레스 태그(tag)로서 작용한다. 태그 매치(tag match)의 발생시, 18비트 폭의 레지스터 값이 물리적 어드레스(846)의 고순위 18비트로서 제공된다. 페이지 오프셋 및 요청 ID는 물리적 어드레스(846)의 저순위 14비트로서 제공된다.

테이블 룩 어사이드 버퍼(844)에 태그 미스(miss)가 있는 경우에, VMU 미스가 신호된다. 이것은 MAU(112)에서 유지되는 완전 페이지 테이블 데이터 구조를 억세스하는 통상적인 해쉬(hash) 알고리즘(848)을 구현하는 VMU 고속 트랩 처리 루틴의 실행을 필요로 한다. 상기 페이지 테이블(850)은 본 구조(100)에 의해 현재 사용 중인 모든 메모리 페이지에 대한 엔트리를 포함한다. 해쉬 알고리즘(848)은 현재의 가상 페이지 번역 연산을 만족시키는데 필요한 페이지 테이블(850)내의 이와같은 엔트리를 식별한다. 이와같은 페이지 테이블 엔트리는 MAU(112)로부터 레지스터 셋트 "A"의 트랩 레지스터에 로드되며, 다음에 특수 레지스터 이동 명령어에 의해 테이블 룩 어사이드 버퍼(844)로 전달된다. 예외 처리 루틴으로부터의 복귀시, VMU 미스 예외를 유발하는 명령어가 IEU(104)에 의해 재실행된다. 다음에, 가상-물리적 어드레스 번역 연산이 예외없이 완료되어야 한다.

VMU 제어 로직(800)은 IFU(102)와 IEU(104)모두에 대해 이중 인터페이스를 제공한다. VMU(108)가 어드레스 번역에 이용가능하다는 것을 표시하기 위한 준비 신호는 제어 라인(822)을 통해 IEU(104)에 제공된다. 본 발명의 양호한 실시예에서, VMU(108)는 IFU(120) 번역 요청을 수용하도록 항상 준비되어 있다. IFU(102) 및 IEU(104)는 둘다 제어 라인(324,804)을 통해 요정을 발할 수도 있다. 본 발명의 양호한 구조(100)에서, IFU(102)는 VMU(108)에 대한 우선순위 억세스를 갖는다. 결과적으로, 오직 단일 비지(busy) 제어 라인(820)만이 IEU(104)에 제공된다.

IFU 및 IEU(102,104) 모두는 공간 ID 및 가상 페이지 넘버 필드를 제어 라인(302,808)을 통해 VMU 제어 로직(800)에 각각 제공한다. 또한, IEU(104)는 어드레스가 참조되는 가상 메모리의 메모리 억세스 보호 어트리뷰트(attributes)를 변형시키는데 필요한 만큼 로드 또는 스토어 연산에 사용될 것인지를 정의하기 위해 판독/기록 제어 신호를 제어 신호(806)를 통해 제공한다. 가상 어드레스의 공간 ID 및 가상 페이지 필드는 실제적인 번역 연산을 수행하기위해 CAM 유니트(802)로 통과된다. 페이지 오프셋 및 Ex ID 필드는 결국 IEU(104)에 의해 CCU(106)로 직접 제공된다. 물리적 페이지 및 요청 ID 필드는 어드레스 라인(836)을 통해 CAM 유니트(802)에 제공된다. 테이블 룩 어사이드 버퍼 매치의 발생은 히트(hit)라인 및 제어 출력 라인(830)을 통해 VMU 제어 로직 유니트(800)로 신호된다. 길이가 18비트인 생성된 물리적 어드레스는 어드레스 출력 라인(824)에 제공된다.

VMU 제어 로직 유니트(800)는 라인(830)상의 히트 및 제어 출력 제어 신호에 응답하여 라인(334,332)상에서 가상 메모리 미스 및 가상 메모리 예외 제어 신호를 발생한다. 가상 메모리 번역 미스는 테이블 룩 어사이드 버퍼(844)내의 페이지 테이블 식별자의 매치 오류로서 정의 된다. 모든 다른 번역 에러는 가상 메모리 예외로서 보고된다.

마지막으로 CAM 유니트(802)내의 데이터 테이블은 IEU(104)에 의해 특수 레지스터-레지스터 이동 명령어의 실행을 통해 변형될 수도 있다. 판독/기록, 레지스터 선택, 리셋트, 로드 및 클리어제어 신호는 제어 라인(810,812,814,816,818)을 통해 IEU(104)에 의해 제공된다. CAM 유니트 레지스터로 기록될 데이터는 IEU(104)로부터 특수 어드레스 데이터 버스(354)에 결합되는 어드레스 버스(808)를 통해 VMU 제어 로직 유니트(800)에 의해 수신된다. 이 데이터는 초기화, 레지스터 선택 및, 판독 또는 기록 제어 신호를 제어하는 제어 신호(828)와 동시에 버스(836)를 통해 CAM 유니트(802)로 전달된다. 결과적으로, CAM 유니트(802)내의 데이터 레지스터는 고레벨 운용 시스템에 의해 정의되는 문맥(context)스위치의 처리를 위해 필요할 때 저장을 위한 판독을 포함하여 본 구조(100)의 동적 연산중에 필요할 때 쉽게 기록될 수도 있다.

V. 캐쉬 제어 유니트:

제 17도에는 CCU(106)에 대한 데이터 인터페이스에 대한 제어가 도시되어 있다. 또한, IFU(102)와 IEU(104)에 대한 분리 인터페이스가 제공된다. 더욱이, 논리적으로 분리된 인터페이스가 명령어 및 데이터 전달에 관해 CCU(106)에 의해 MCU(110)로 제공된다.

IFU 인터페이스는 어드레스 라인(324)상에 제공되는 물리적 페이지 어드레스와, 어드레스 라인(824)상에 제공되는 바와 같은 VMU 변화 페이지 어드레스 및, 제어 라인(294,296) 상에 분리적으로 전달되는 바와 같은 요청 ID로 이루어진다.

무조건 데이터 전달 버스(114)는 전체 명령어 셋트를 병렬로 IFU(102)로 전달하기 위해 제공된다. 마지막으로 판독/비지 및 준비 제어 신호는 제어 라인(298,300,302)을 통해 CCU(106)로 제공된다.

유사하게, 완전 물리적 어드레스는 물리적 어드레스 버스(788)를 통해 IEU(102)에 의해 제공된다. 요청 Ex ID는 제어 라인(796)을 통해 IEU(104)의 로드/스토어 유니트로 또는 유니트로부터 분리적으로 제공된다. 80비트 폭의 양방향 데이터 버스는 CCU(106)에 의해 IEU(104)로 제공된다. 그러나, 본 구조(100)의 양호한 실시예에서는, 오직 하위의 64비트만이 IEU(104)에 의해 이용된다.

전 80비트 데이터 버스의 CCU(106)내의 유효성 및 지원은 부동 소수점 데이터 경로(660)의 변형을 통해 IEEE 표준(754)에 따라 부동 소수점 연산을 지원하는 본 구조(100)의 계속되는 구현을 지원하기 위해 제공된다.

요청, 비지, 준비, 판독/기록 및 제어 신호(784)를 통해 설정되는 IEU 제어 인터페이스는 IFU(102)에 의해 이용되는 대응하는 제어 신호와 실질적으로 동일하다. 예외는 로드 및 스토어 연산 사이를 구별하기위한 판독/기록 제어 신호의 규정이다. 폭 제어 신호는 IEU(104)에 의한 각각의 CCU(106) 억세스 동안에 전달되는 바이트 수를 지정하며, 이와 대조적으로, 명령어 캐쉬(132)의 모든 억세스는 일정한 128 비트 폭의 데이터 펫치연산이다.

CCU(106)는 분리된 명령어 및 데이터 캐쉬(132,134)에 관한 실질적으로 통상적인 캐쉬 제어기 기능을 구현한다. 본 양호한 구조(100)에서, 명령어 캐쉬(132)는 256개의 128비트 폭의 명령어 셋트의 저장을 제공하는 고속 메모리이다. 데이터 캐쉬(134)는 데이터의 1024개의 32비트 폭의 워드의 저장을 제공한다. 명령어 및 데이터 캐쉬(132,134)의 내용으로부터 즉시 만족될 수 없는 명령어 및 데이터 요청은 MCU(110)로 통과된다. 명령어 캐쉬미스에 대해서는 28비트 폭의 물리적 어드레스가 어드레스 버스(860)를 통해 MCU(110)로 제공된다. CCU(106)와 MCU(110)의 연산을 코디네이트 하기위한 추가 제어 신호 및 요청 ID는 제어 라인(862)상에 제공된다. 일단 MCU(110)가 MAU(112)의 필요한 판독 억세스를 코디네이트하면, 2번의 연속적인 64비트 폭의 데이터 전달이 MAU(112)로부터 명령어 캐쉬(132)로 직접 수행된다. 2번의 전달은 데이터 버스(136)가 본 양호한 구조(100)에서 64비트 폭의 버스인것을 가정하여 요구된다. 요청된 데이터가 MCU(110)를 통해 복귀될때, 요청 연산의 미결중에 유지되는 요청 ID도 또한 제어 라인(862)을 통해 CCU(106)로 복귀된다.

데이터 캐쉬(134)와 MCU(110)사이의 데이터 전달 연산은 명령어 캐쉬 연산과 실질적으로 동일하다. 데이터 로드 및 스토어 연산이 단일 바이트를 참조할 수도 있기 때문에, 전 32비트 폭의 물리적 어드레스가 어드레스 버스(864)를 통해 MCU(110)로 제공된다. 인터페이스 제어 신호 및 요청 Ex ID는 제어 라인(866)을 통해 전달된다. 양방향 64비트 폭의 데이터 전달은 데이터 캐쉬 버스(138)를 통해 제공된다.

VI. 요약/ 결론

이렇게해서, 고성능 RISC기반 마이크로프로세서 구조가 기술되었다. 이 구조는 명령어의 비순서적 실행, 분리된 메인 및 목표 명령어 스트림 프리펫치 명령어 전달 경로, 및 절차 명령어 인식 및 전용 프리펫치 경로를 효과적으로 구현한다. 최적화된 명령어 실행 유니트는 정수, 부동 소수점, 및 부울 연산을 지원하는 복수의 최적화된 명령어 실행 경로를 제공하며, 쉽게 설정되는 정확한 머신 스테이트를 유지하면서 비순서적 실행 및 명령어 취소를 용이하게하는 각각의 임시 레지스터 파일을 포함한다.

그러므로, 본 발명의 양호한 실시예가 설명되었지만, 본 발명의 범위 내에서 보통의 숙련자에 의해 다른 변형 및 수정이 쉽게 이루어질 수도 있다는 것으로 이해되어야 한다.

발명의 효과

전술한 바와 같은 본 발명에 따르면, 본질적으로 RISC 형태의 코어구조를 이용하는 극도의 고성능의 처리량을 실현하는 구조를 제공할 수 있다.

(57) 청구의 범위

청구항 1.

데이터를 처리하기 위해 명령어 스토어로부터 얻어지는 명령어들을 실행하기 위한 수퍼-스칼라 마이크로프로세서를 포함하는 컴퓨터 시스템에 있어서,

상기 명령어 스토어로부터 다수의 명령어들을 펫치하기 위한 펫치 회로 - 상기 다수의 명령어들은 프로그램 순서로 이루어짐 - ;

상기 펫치 회로로부터 상기 다수의 명령어들을 버퍼링하기 위한 FIFO 버퍼;

상기 FIFO 버퍼에 의해 버퍼링된 상기 다수의 명령어들 중 하나 이상의 명령어들을 동시에 디코드하고 디스패치하기 위한 디스패치 회로; 및

실행 유니트

를 포함하고,

상기 실행 유니트는,

상기 프로그램 순서를 벗어나 상기 디스패치 회로에 의해 디스패치된 상기 다수의 명령어들 중 하나의 명령어를 각각 실행하는 다수의 기능 유니트;

상기 다수의 기능 유니트로부터의 비순서적 실행 결과를 임시 저장하기 위한 임시 레지스터 및 상기 임시 레지스터로부터의 결과를 저장하기 위한 레지스터 어레이를 포함하는 레지스터 파일; 및

다수의 데이터 루팅 경로를 포함하고,

여기서, 상기 레지스터 파일은 상기 다수의 기능 유니트 중 하나 이상의 기능 유니트로 데이터를 제공하여, 상기 다수의 기능 유니트에 의해 상기 다수의 명령어들 중 하나 이상의 명령어들의 동시 실행을 가능하게 하기 위해, 상기 다수의 데이터 루팅 경로를 통해 상기 다수의 기능 유니트와 통신하는

컴퓨터 시스템.

청구항 2.

제1항에 있어서,

상기 다수의 비순서적으로 실행된 명령어들이 프로그램 순서로 완료되도록 하기 위한 회수 회로를 더 포함하고,

여기서, 상기 회수 회로는 상기 임시 레지스터로부터 상기 레지스터 어레이로 데이터 값들이 전달되도록 하는

컴퓨터 시스템.

청구항 3.

제2항에 있어서,

상기 회수 회로는 하나의 프로세서 사이클에서 최대 4개의 명령어들을 회수할 수 있는

컴퓨터 시스템.

청구항 4.

제1항에 있어서,

상기 FIFO 버퍼는 하나의 프로세서 사이클에서 4개 또는 그 이상의 명령어들을 상기 디스패치 회로로 제공할 수 있는

컴퓨터 시스템.

청구항 5.

제1항에 있어서,

상기 디스패치 회로는 하나의 프로세서 사이클에서 최대 4개의 명령어를 상기 실행 유니트로 디스패치할 수 있는

컴퓨터 시스템.

청구항 6.

제1항에 있어서,

상기 FIFO 버퍼는 제1 및 제2 레지스터를 포함하고,

상기 각각의 레지스터는 하나 또는 그 이상의 명령어들을 상기 디스패치 회로로 동시에 제공하는

컴퓨터 시스템.

청구항 7.

제1항에 있어서,

상기 FIFO 버퍼는 제1 및 제2 레지스터를 포함하고,

상기 각각의 레지스터는 4개의 명령어들의 명령어 셋트를 상기 디스패치 회로로 동시에 제공하는

컴퓨터 시스템.

청구항 8.

제1항에 있어서,

상기 디스패치 회로는 상기 다수의 기능 유니트의 이용가능성을 조건으로 하여 상기 실행 유니트로 프로그램 순서를 벗어나 상기 명령어들을 발생하기 위한 회로를 더 포함하는

컴퓨터 시스템.

청구항 9.

제1항에 있어서,

상기 명령어 스토어는 상기 마이크로프로세서에 의해 실행될 명령어들을 저장하는 외부 캐쉬에 연결되어 있는

컴퓨터 시스템.

청구항 10.

제1항에 있어서,

상기 디스패치 회로는 하나의 프로세서 사이클에서 다수의 상기 버퍼링된 명령어들을 상기 실행 유니트로 제공함으로써 상기 명령어들을 동시에 디스패치하는

컴퓨터 시스템.

청구항 11.

명령어 스토어로부터 얻어지는 명령어들을 실행하기 위한 컴퓨터 시스템에 있어서,

상기 명령어 스토어로부터 다수의 명령어들을 펫치하기 위한 펫치 회로 - 상기 다수의 명령어들은 프로그램 순서로 이루어짐 - ;

상기 다수의 명령어들을 버퍼링하기 위한 명령어 버퍼;

상기 명령어 버퍼에 응답하는 분기 디코더;

상기 분기 디코더로부터의 상기 다수의 명령어들을 버퍼링하기 위한 FIFO 버퍼;

상기 FIFO 버퍼에 의해 버퍼링된 상기 다수의 명령어들 중 하나 이상의 명령어들을 동시에 디코드하고 디스패치하기 위한 디스패치 회로; 및

실행 유니트

를 포함하고,

상기 실행 유니트는,

상기 프로그램 순서를 벗어나 상기 디스패치 회로에 의해 디스패치된 상기 다수의 명령어들 중 하나의 명령어를 각각 실행하는 다수의 기능 유니트;

상기 다수의 기능 유니트로부터의 비순서적 실행 결과를 임시 저장하기 위한 임시 레지스터 및 상기 임시 레지스터로부터의 결과를 저장하기 위한 레지스터 어레이를 포함하는 레지스터 파일; 및

다수의 데이터 루팅 경로를 포함하고,

여기서, 상기 레지스터 파일은 상기 다수의 기능 유니트 중 하나 이상의 기능 유니트로 데이터를 제공하여, 상기 다수의 기능 유니트에 의해 상기 다수의 명령어들 중 하나 이상의 명령어들의 동시 실행을 가능하게 하기 위해, 상기 다수의 데이터 루팅 경로를 통해 상기 다수의 기능 유니트와 통신하는

컴퓨터 시스템.

청구항 12.

제11항에 있어서,

상기 다수의 비순서적으로 실행된 명령어들이 프로그램 순서로 완료되도록 하기 위한 회수 회로를 더 포함하고,

여기서, 상기 회수 회로는 상기 임시 레지스터로부터 상기 레지스터 어레이로 데이터 값들이 전달되도록 하는

컴퓨터 시스템.

청구항 13.

제12항에 있어서,

상기 회수 회로는 하나의 프로세서 사이클에서 최대 4개의 명령어들을 회수할 수 있는

컴퓨터 시스템.

청구항 14.

제11항에 있어서,

상기 FIFO 버퍼는 하나의 프로세서 사이클에서 4개 또는 그 이상의 명령어들을 상기 디스패치 회로로 제공할 수 있는

컴퓨터 시스템.

청구항 15.

제11항에 있어서,

상기 디스패치 회로는 하나의 프로세서 사이클에서 최대 4개의 명령어를 상기 실행 유니트로 디스패치할 수 있는

컴퓨터 시스템.

청구항 16.

제11항에 있어서,

상기 디스패치 회로는 상기 다수의 기능 유니트의 이용가능성을 조건으로 하여 상기 실행 유니트로 프로그램 순서를 벗어나 상기 명령어들을 발생하기 위한 회로를 더 포함하는

컴퓨터 시스템.

청구항 17.

제11항에 있어서,

상기 명령어 스토어는 마이크로프로세서에 의해 실행될 명령어들을 저장하는 외부 캐쉬에 연결되어 있는

컴퓨터 시스템.

청구항 18.

제11항에 있어서,

상기 디스패치 회로는 하나의 프로세서 사이클에서 다수의 상기 버퍼링된 명령어들을 상기 실행 유니트로 제공함으로써 상기 명령어들을 동시에 디스패치하는

컴퓨터 시스템.

청구항 19.

명령어 스토어로부터 얻어지는 명령어들을 실행하기 위한 컴퓨터 시스템에 있어서,

상기 명령어 스토어로부터 다수의 명령어들을 펫치하기 위한 펫치 회로 - 상기 다수의 명령어들은 프로그램 순서로 이루어짐 - ;

상기 다수의 명령어들을 버퍼링하기 위한 명령어 버퍼;

상기 명령어 버퍼에 응답하여 상기 명령어 버퍼에 버퍼링된 상기 다수의 명령어들로부터의 분기 정보를 디코딩하기 위한 분기 디코더;

상기 명령어 버퍼에 이전에 버퍼링된 상기 다수의 명령어들을 다음에 계속해서 버퍼링하기 위한 FIFO 버퍼;

상기 FIFO 버퍼에 의해 버퍼링된 상기 다수의 명령어들 중 하나 이상의 명령어들을 동시에 디코드하고 디스패치하기 위한 디스패치 회로; 및

실행 유니트

를 포함하고,

상기 실행 유니트는,

상기 프로그램 순서를 벗어나 상기 디스패치 회로에 의해 디스패치된 상기 다수의 명령어들 중 하나의 명령어를 각각 실행하는 다수의 기능 유니트;

상기 다수의 기능 유니트로부터의 비순서적 실행 결과를 임시 저장하기 위한 임시 레지스터 및 상기 임시 레지스터로부터의 결과를 저장하기 위한 레지스터 어레이를 포함하는 레지스터 파일; 및

다수의 데이터 루팅 경로를 포함하고,

여기서, 상기 레지스터 파일은 상기 다수의 기능 유니트 중 하나 이상의 기능 유니트로 데이터를 제공하여, 상기 다수의 기능 유니트에 의해 상기 다수의 명령어들 중 하나 이상의 명령어들의 동시 실행을 가능하게 하기 위해, 상기 다수의 데이터 루팅 경로를 통해 상기 다수의 기능 유니트와 통신하는

컴퓨터 시스템.

청구항 20.

제19항에 있어서,

상기 다수의 비순서적으로 실행된 명령어들이 프로그램 순서로 완료되도록 하기 위한 회수 회로를 더 포함하고,

여기서, 상기 회수 회로는 상기 임시 레지스터로부터 상기 레지스터 어레이로 데이터 값들이 전달되도록 하는

컴퓨터 시스템.

청구항 21.

제20항에 있어서,

상기 회수 회로는 하나의 프로세서 사이클에서 최대 4개의 명령어들을 회수할 수 있는

컴퓨터 시스템.

청구항 22.

제19항에 있어서,

상기 FIFO 버퍼는 하나의 프로세서 사이클에서 4개 또는 그 이상의 명령어들을 상기 디스패치 회로로 제공할 수 있는

컴퓨터 시스템.

청구항 23.

제19항에 있어서,

상기 디스패치 회로는 하나의 프로세서 사이클에서 최대 4개의 명령어를 상기 실행 유니트로 디스패치할 수 있는

컴퓨터 시스템.

청구항 24.

제19항에 있어서,

상기 디스패치 회로는 상기 다수의 기능 유니트의 이용가능성을 조건으로 하여 상기 실행 유니트로 프로그램 순서를 벗어나 상기 명령어들을 발생하기 위한 회로를 더 포함하는

컴퓨터 시스템.

청구항 25.

제19항에 있어서,

상기 명령어 스토어는 마이크로프로세서에 의해 실행될 명령어들을 저장하는 외부 캐쉬에 연결되어 있는

컴퓨터 시스템.

청구항 26.

제19항에 있어서,

상기 디스패치 회로는 하나의 프로세서 사이클에서 다수의 상기 버퍼링된 명령어들을 상기 실행 유니트로 제공함으로써 상기 명령어들을 동시에 디스패치하는

컴퓨터 시스템.

청구항 27.

제19항에 있어서,

상기 FIFO 버퍼는 상기 명령어 버퍼로부터의 상기 다수의 명령어를 버퍼링하는

컴퓨터 시스템.

청구항 28.

컴퓨터 시스템에 있어서,

메모리;

명령어 스토어로부터 얻어지는 명령어들을 실행하기 위한 수퍼-스칼라 마이크로프로세서; 및

상기 마이크로프로세서와 상기 메모리 사이에 연결된 버스

를 포함하고,

상기 마이크로프로세서는,

상기 명령어 스토어로부터 다수의 명령어들을 펫치하기 위한 펫치 회로 - 상기 다수의 명령어들은 프로그램 순서로 이루어짐 - ;

상기 다수의 명령어들을 버퍼링하기 위한 명령어 버퍼;

상기 명령어 버퍼에 응답하는 분기 디코더;

상기 분기 디코더로부터의 상기 다수의 명령어들을 버퍼링하기 위한 FIFO 버퍼;

상기 FIFO 버퍼에 의해 버퍼링된 상기 다수의 명령어들 중 하나 이상의 명령어들을 동시에 디코드하기 위한 제2 디코더 및 동시에 디스패치하기 위한 디스패치 부분을 포함하는 디스패치 회로; 및

실행 유니트

를 포함하고,

상기 실행 유니트는,

상기 프로그램 순서를 벗어나 상기 디스패치 회로에 의해 디스패치된 상기 다수의 명령어들 중 하나의 명령어를 각각 실행하는 다수의 기능 유니트;

상기 다수의 기능 유니트로부터의 비순서적 실행 결과를 임시 저장하기 위한 임시 레지스터 및 상기 임시 레지스터로부터의 결과를 저장하기 위한 레지스터 어레이를 포함하는 레지스터 파일; 및

다수의 데이터 루팅 경로를 포함하고,

여기서, 상기 레지스터 파일은 상기 다수의 기능 유니트 중 하나 이상의 기능 유니트로 데이터를 제공하여, 상기 다수의 기능 유니트에 의해 상기 다수의 명령어들 중 하나 이상의 명령어들의 동시 실행을 가능하게 하기 위해, 상기 다수의 데이터 루팅 경로를 통해 상기 다수의 기능 유니트와 통신하는

컴퓨터 시스템.

청구항 29.

컴퓨터 시스템에 있어서,

메모리;

명령어 스토어로부터 얻어지는 명령어들을 실행하기 위한 수퍼-스칼라 마이크로프로세서; 및

상기 마이크로프로세서와 상기 메모리 사이에 연결된 버스

를 포함하고,

상기 마이크로프로세서는,

상기 명령어 스토어로부터 다수의 명령어들을 펫치하기 위한 펫치 회로 - 상기 다수의 명령어들은 프로그램 순서로 이루어짐 - ;

상기 다수의 명령어들을 버퍼링하기 위한 명령어 버퍼;

상기 명령어 버퍼에 응답하여 상기 명령어 버퍼에 버퍼링된 상기 다수의 명령어들로부터의 분기 정보를 디코딩하기 위한 분기 디코더;

상기 명령어 버퍼에 이전에 버퍼링된 상기 다수의 명령어들을 다음에 계속해서 버퍼링하기 위한 FIFO 버퍼;

상기 FIFO 버퍼에 의해 버퍼링된 상기 다수의 명령어들 중 하나 이상의 명령어들을, 동시에 디코드하기 위한 제2 디코더 및 동시에 디스패치하기 위한 디스패치 부분을 포함하는 디스패치 회로; 및

실행 유니트

를 포함하고,

상기 실행 유니트는,

상기 프로그램 순서를 벗어나 상기 디스패치 회로에 의해 디스패치된 상기 다수의 명령어들 중 하나의 명령어를 각각 실행하는 다수의 기능 유니트;

상기 다수의 기능 유니트로부터의 비순서적 실행 결과를 임시 저장하기 위한 임시 레지스터 및 상기 임시 레지스터로부터의 결과를 저장하기 위한 레지스터 어레이를 포함하는 레지스터 파일; 및

다수의 데이터 루팅 경로를 포함하고,

여기서, 상기 레지스터 파일은 상기 다수의 기능 유니트 중 하나 이상의 기능 유니트로 데이터를 제공하여, 상기 다수의 기능 유니트에 의해 상기 다수의 명령어들 중 하나 이상의 명령어들의 동시 실행을 가능하게 하기 위해, 상기 다수의 데이터 루팅 경로를 통해 상기 다수의 기능 유니트와 통신하는

컴퓨터 시스템.

청구항 30.

삭제

청구항 31.

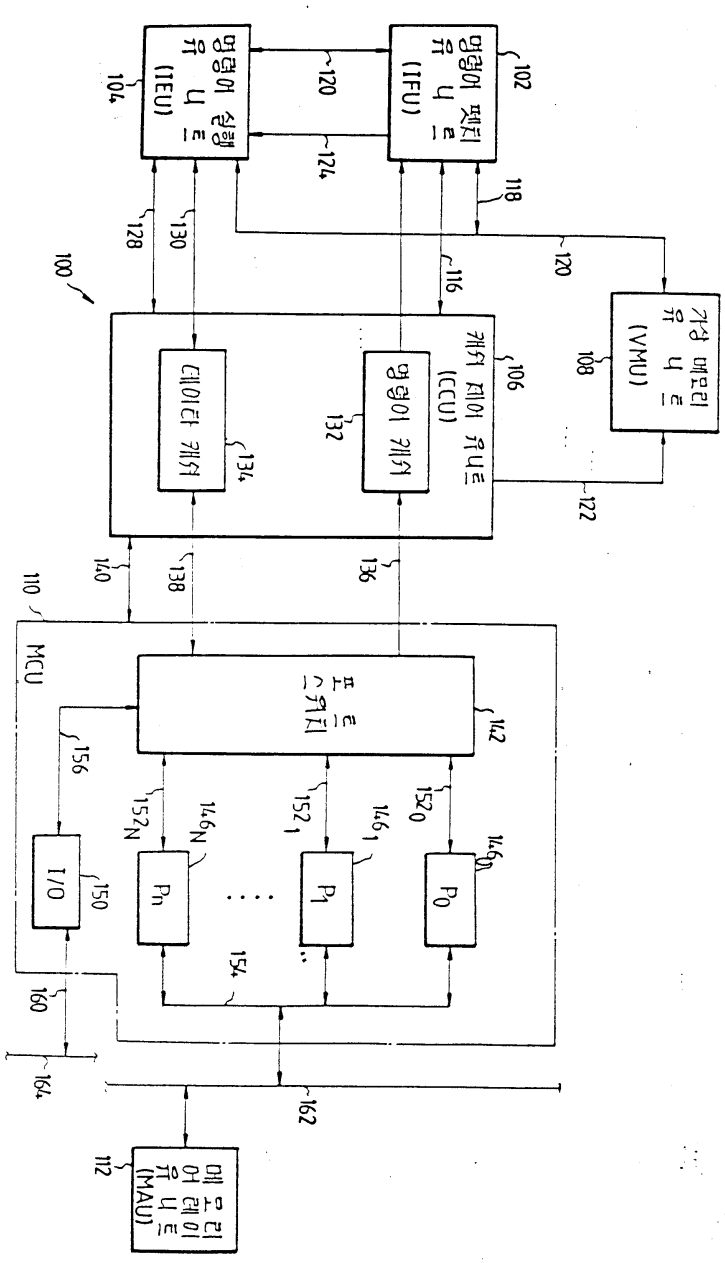
삭제

청구항 32.

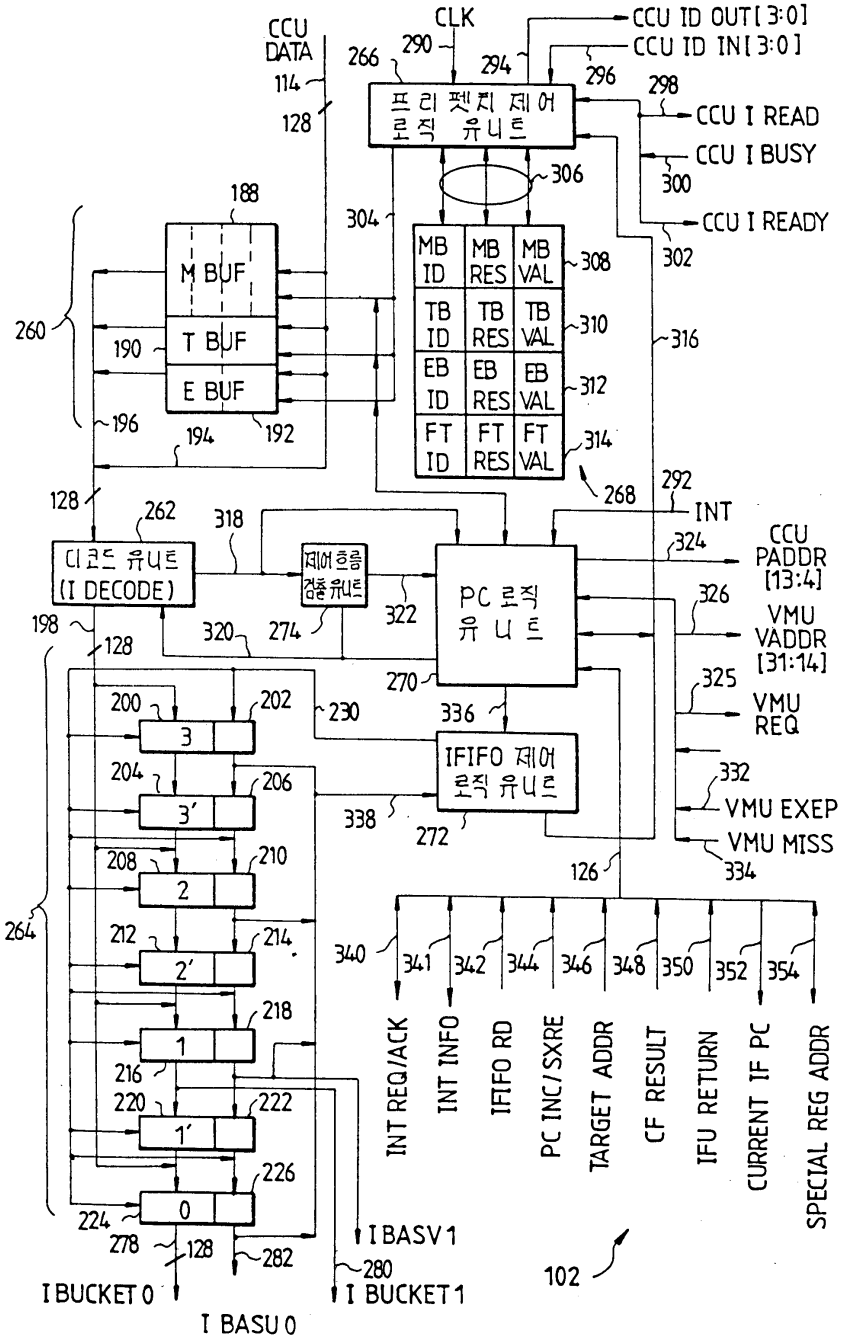
삭제

도면

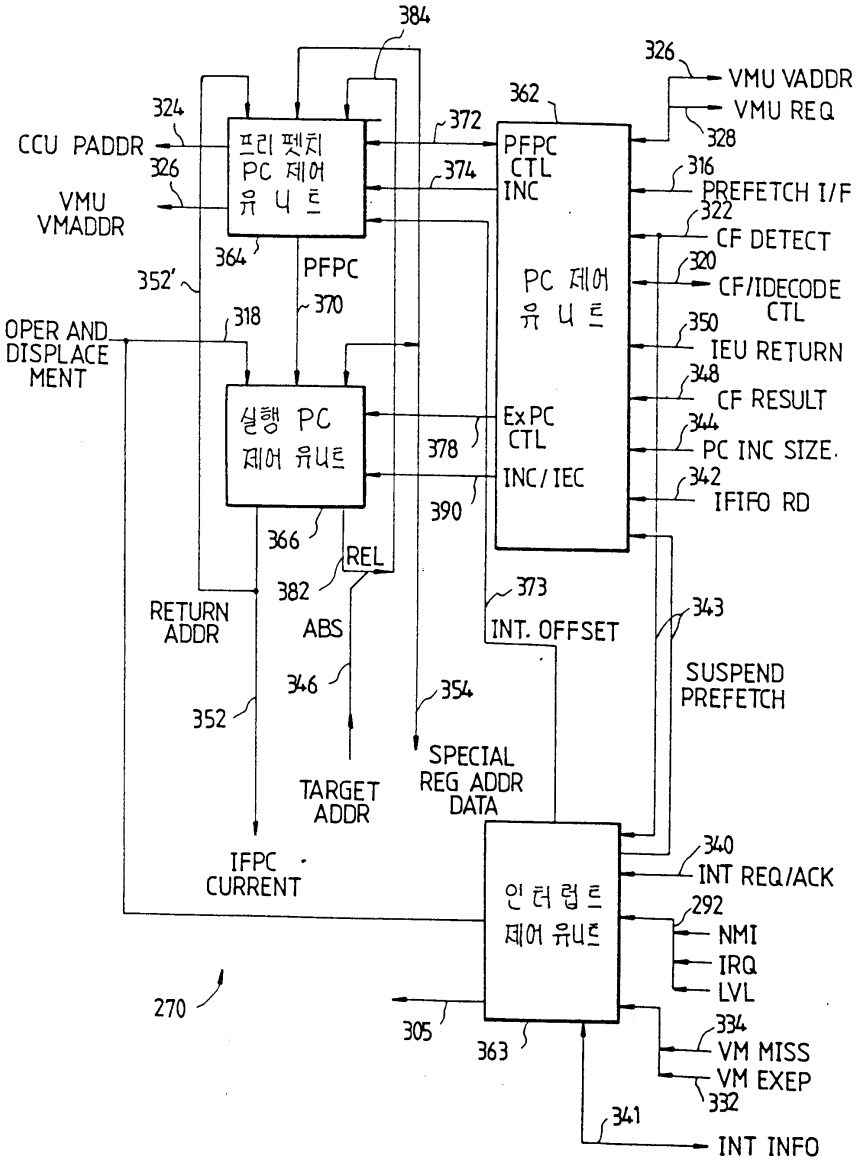
도면1



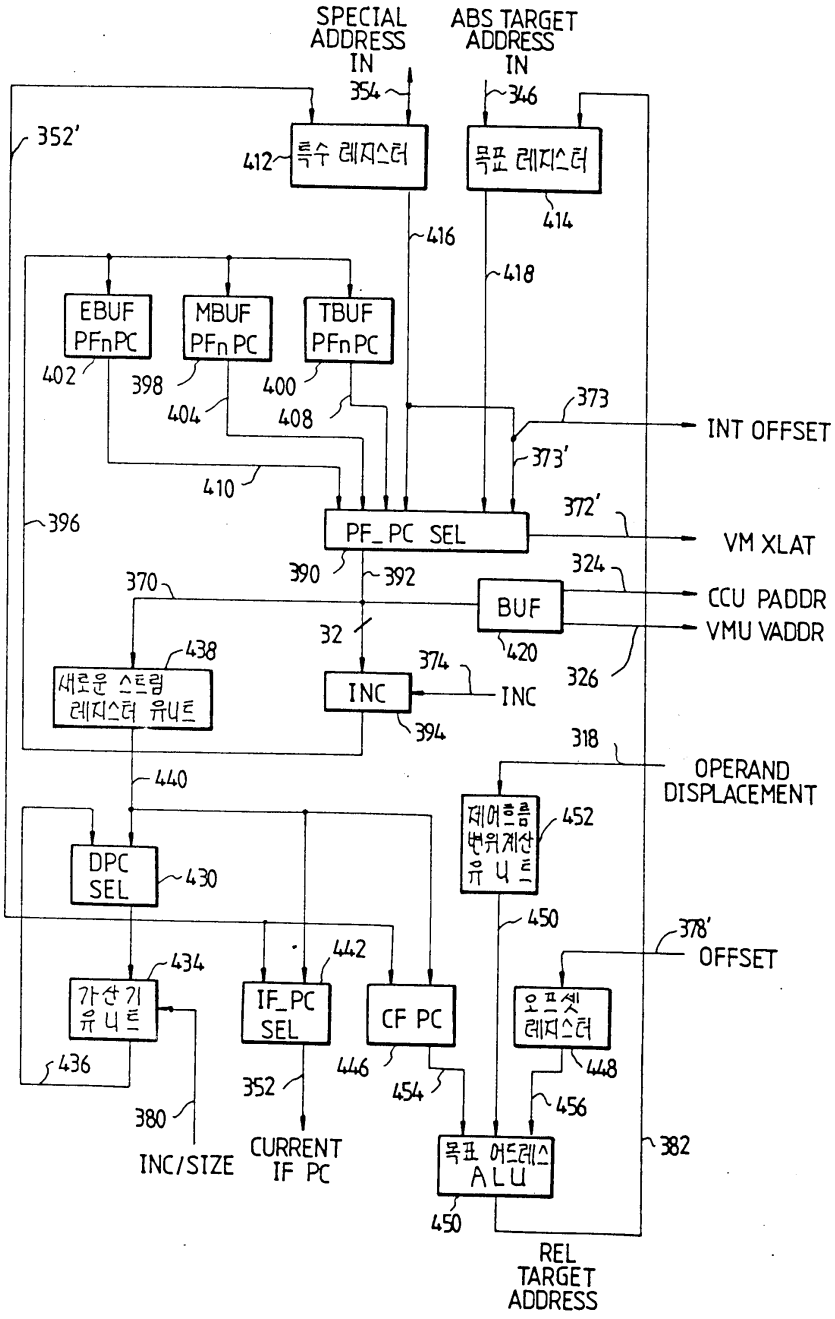
도면2


CCU DATA
CLK
CCU ID OUT[3:0]
CCU ID IN[3:0]
프리 펫치 제어 로직 유니트
CCU I READ
CCU I BUSY
CCU I READY
M BUF
T BUF
E BUF
MB ID
MB RES
MB VAL
TB ID
TB RES
TB VAL
EB ID
EB RES
EB VAL
FT ID
FT RES
FT VAL
INT
디코드 유니트 (I DECODE)
제어흐름 검출유니트
PC 로직 유니트
CCU PADDR [13:4]
VMU VADDR [31:14]
VMU REQ
VMU EXEP
VMU MISS
IFIFO 제어 로직 유니트
INT REQ/ACK
INT INFO
IFIFO RD
PC INC/SXRE
TARGET ADDR
CF RESULT
IFU RETURN
CURRENT IF PC
SPECIAL REG ADDR
I BASV 1
I BUCKET 0
I BUCKET 1
I BASU 0
102

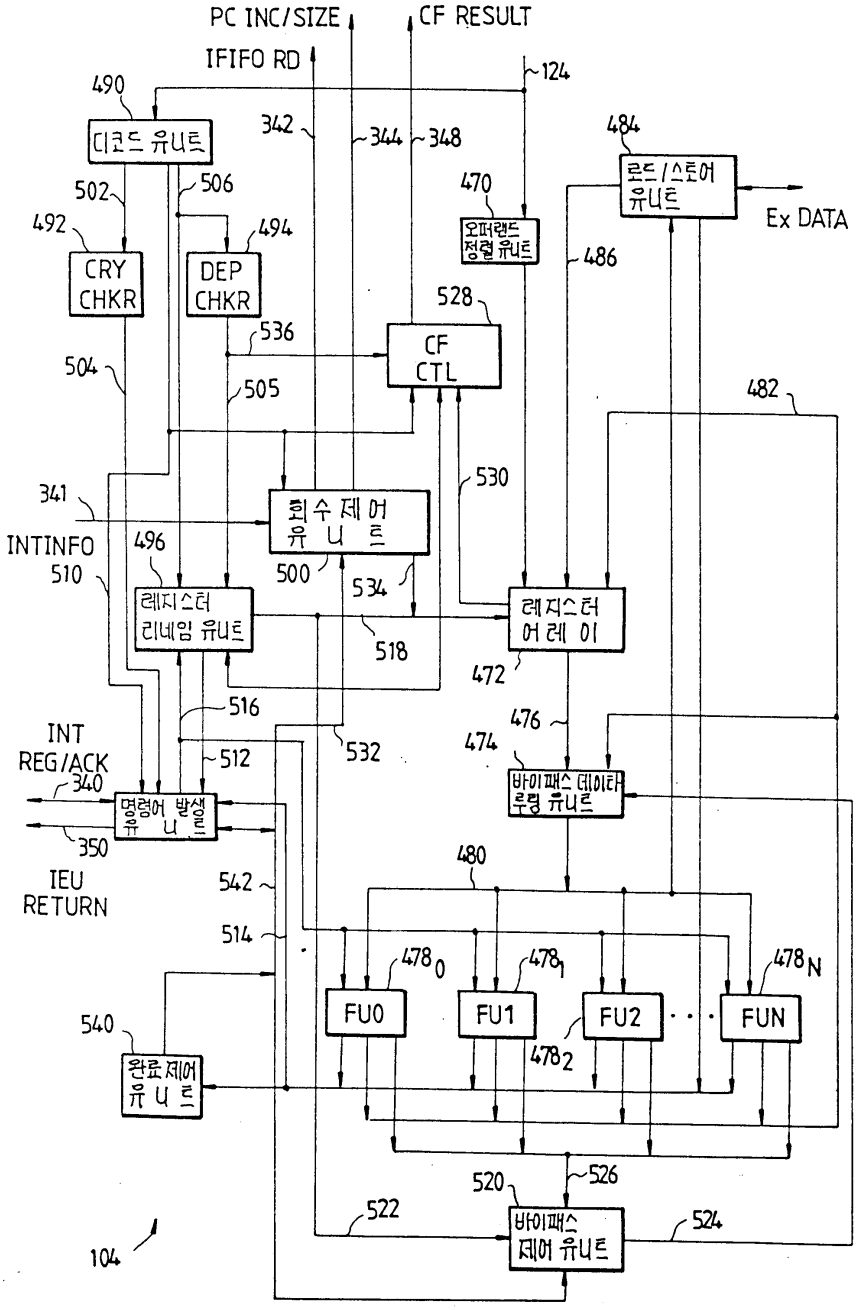
도면3


384
326
VMU VADDR
VMU REQ
328
362
324
CCU PADDR
326
VMU VMADDR
프리 펫치 PC 제어 유 닛 트
372
374
PFPC CTL INC
316
PREFETCH I/F
322
CF DETECT
320
CF/IDECODE CTL
350
IEU RETURN
348
CF RESULT
344
PC INC SIZE
342
IFIFO RD
PC 제어 유 닛 트
364
352'
PFPC
370
318
OPER AND DISPLACE MENT
실행 PC 제어 유닛트
ExPC CTL
INC/IEC
378
390
366
REL
382
RETURN ADDR
ABS
373
INT. OFFSET
343
SUSPEND PREFETCH
346
352
354
SPECIAL REG ADDR DATA
TARGET ADDR
IFPC CURRENT
340
INT REQ/ACK
인 터 럽 트 제어 유닛트
292
NMI
IRQ
LVL
270
305
363
334
VM MISS
VM EXEP
332
341
INT INFO

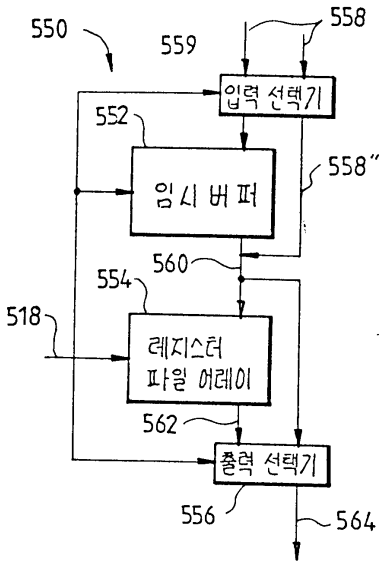
도면4


SPECIAL ADDRESS IN
354
ABS TARGET ADDRESS IN
346
352'
412
특수 레지스터
목표 레지스터
414
416
418
EBUF PFnPC
MBUF PFn PC
TBUF PFn PC
402
398
400
404
408
373
INT OFFSET
373'
410
372'
VM XLAT
396
PF_PC SEL
390
392
324
370
BUF
CCU PADDR
VMU VADDR
32
420
326
438
374
INC
INC
새로운 스트림 레지스터 유니트
394
318
OPERAND DISPLACEMENT
440
제어흐름 변위계산 유니트
452
DPC SEL
430
450
378'
OFFSET
434
442
가산기 유니트
IF_PC SEL
CF PC
오프셋 레지스터
436
446
448
352
454
456
380
382
INC/SIZE
CURRENT IF PC
목표 어드레스 ALU
450
REL TARGET ADDRESS

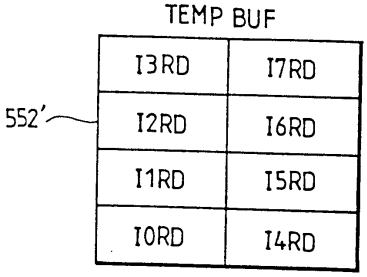
도면5


PC INC/SIZE
CF RESULT
IFIFO RD
124
490
디코드 유니트
342
344
348
484
로드/스토어 유니트
Ex DATA
502
506
470
오퍼랜드 정렬 유니트
492
494
CRY CHKR
DEP CHKR
486
528
536
CF CTL
504
505
482
341
530
INTINFO
510
496
인터럽트 제어 유니트
500
534
레지스터 리네임 유니트
518
레지스터 어레이
472
516
INT REG/ACK
340
512
532
476
474
바이패스 데이타 로딩 유니트
명령어 발생 유니트
350
480
IEU RETURN
542
514
478 0
478 1
478 N
FU0
FU1
FU2
FUN
478 2
540
완료제어 유니트
520
526
522
바이패스 제어 유니트
524
104

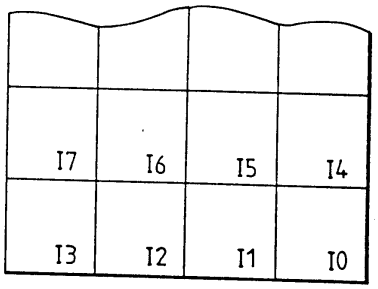
도면6a



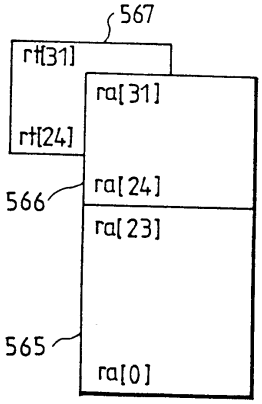
도면6b



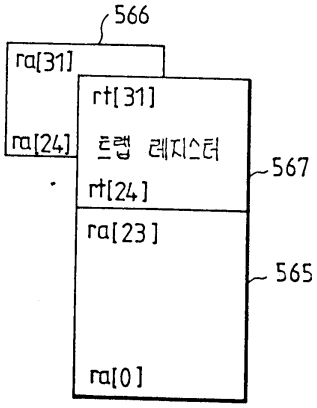
도면6c



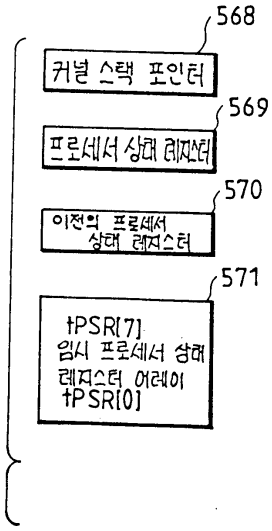
도면7a



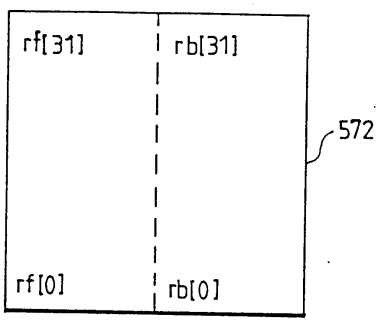
도면7b



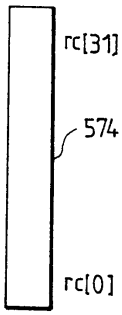
도면7c



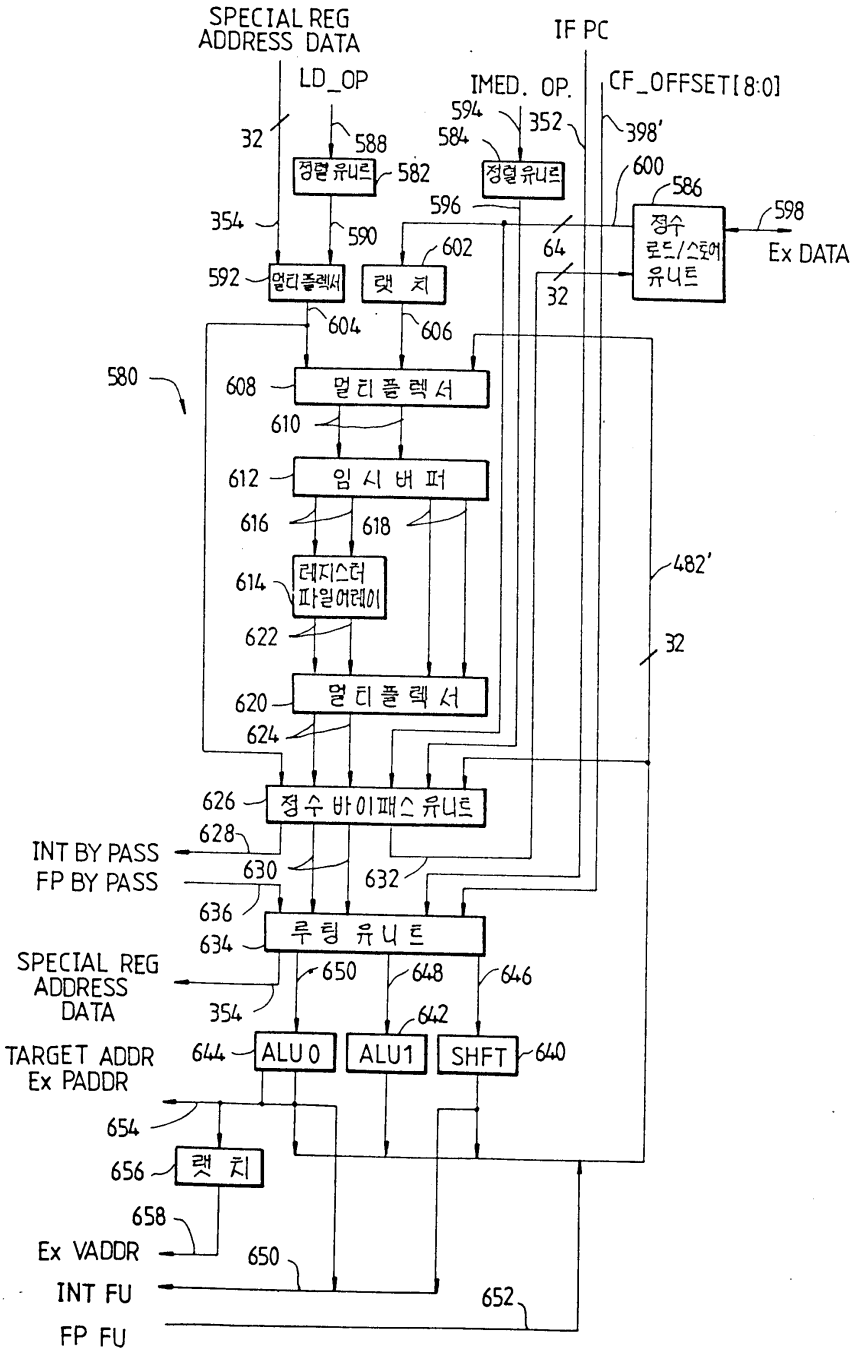
도면8



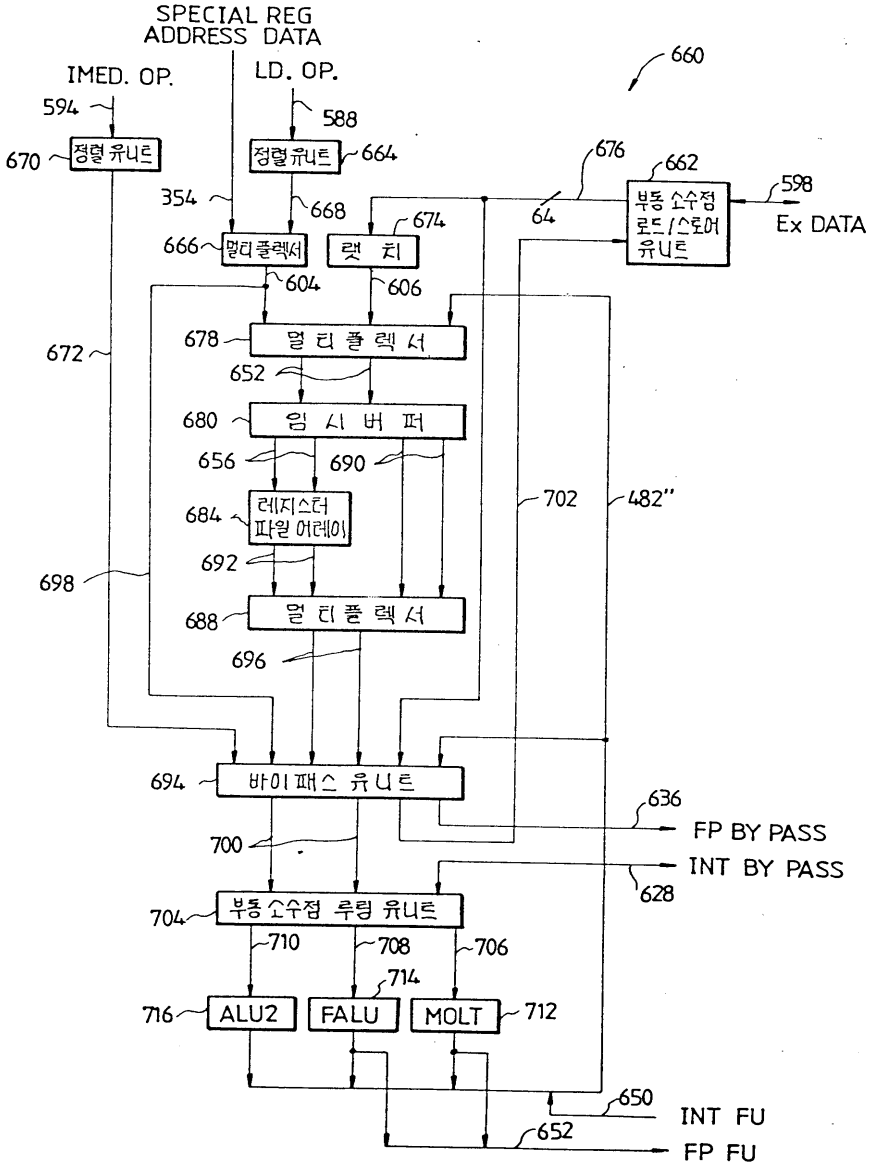
도면9



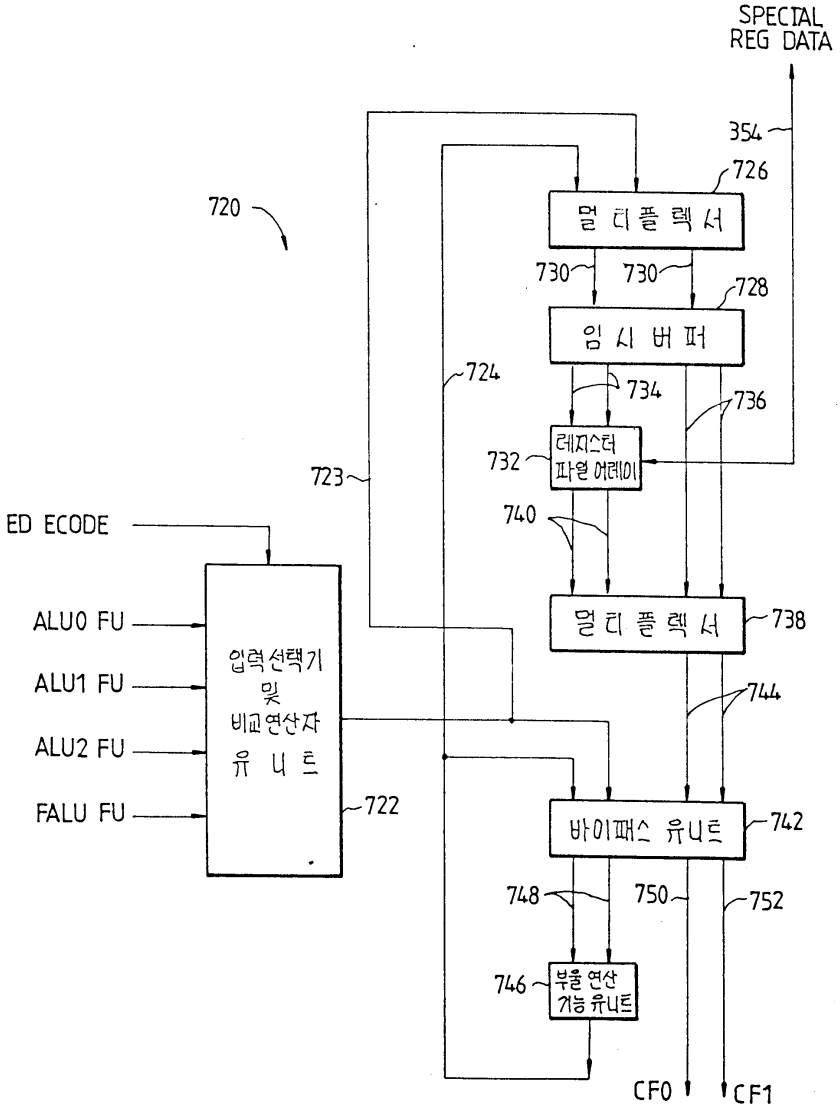
도면10


SPECIAL REG ADDRESS DATA
LD_OP
IMED. OP.
IF PC
CF_OFFSET[8:0]
32
588
정렬 유니트
582
594
584
352
정렬 유니트
398'
600
586
596
354
590
602
64
정수 로드/스토어 유니트
598
Ex DATA
592
멀티플렉서
랫 치
32
604
606
580
608
멀 티 플 렉 서
610
612
임 시 버 퍼
616
618
614
레지스터 파일어레이
482'
622
32
620
멀 티 플 렉 서
624
626
정 수 바 이 패 스 유 니 트
628
INT BY PASS
FP BY PASS
630
632
636
루 팅 유 니 트
634
SPECIAL REG ADDRESS DATA
650
648
646
354
642
TARGET ADDR
Ex PADDR
644
ALU 0
ALU1
SHFT
640
654
656
랫 치
658
Ex VADDR
650
INT FU
652
FP FU

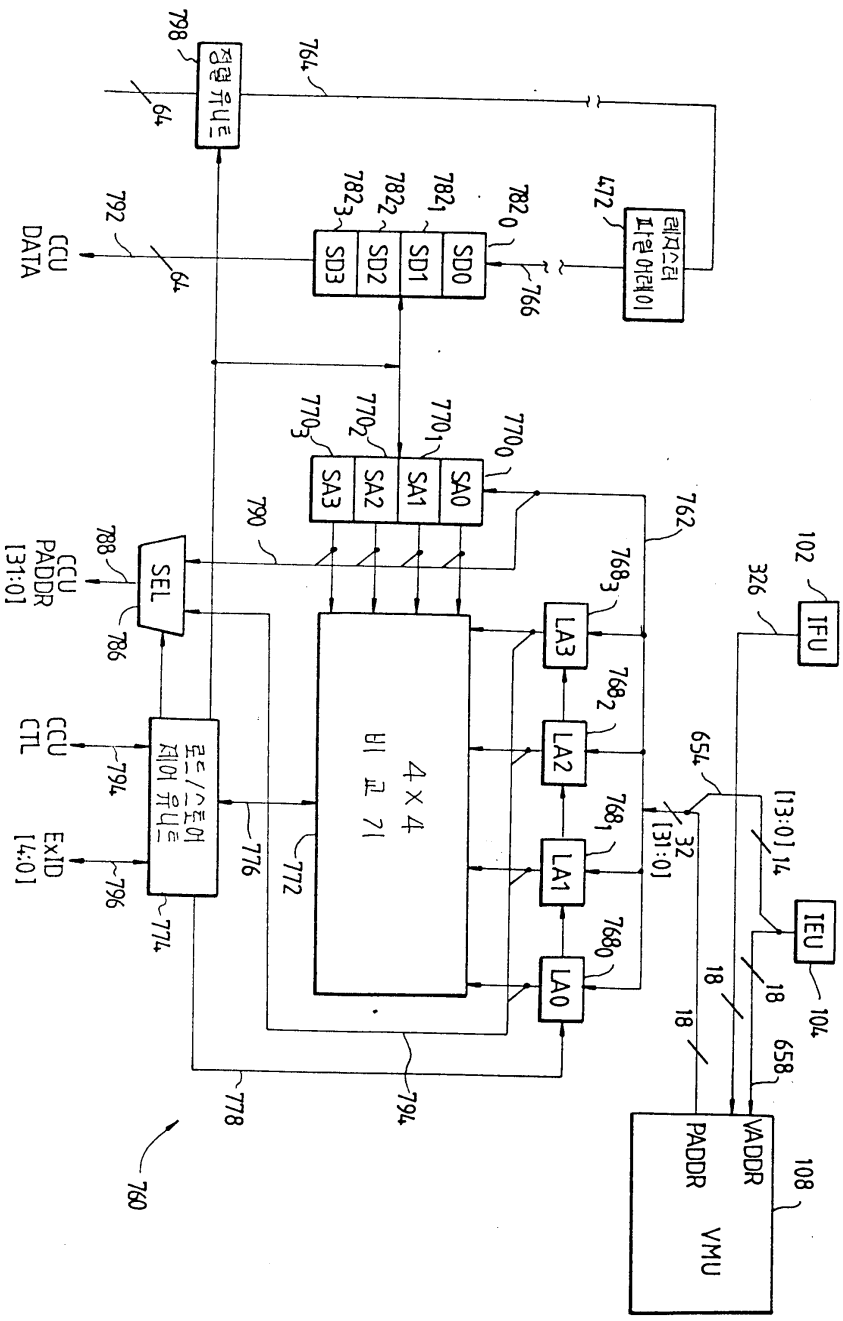
도면11


SPECIAL REG
ADDRESS DATA
IMED. OP.
LD. OP.
660
594
588
670
정렬 유니트
정렬유니트
664
676
662
354
668
64
부동 소수점
로드/스토어
유니트
598
Ex DATA
666
멀티플렉서
랫 치
674
604
606
672
678
멀 티 플 렉 서
652
680
임 시 버 퍼
656
690
684
레지스터
파일 어레이
702
482"
692
698
688
멀 티 플 렉 서
696
694
바이패스 유니트
636
FP BY PASS
700
INT BY PASS
628
704
부동 소수점 루팅 유니트
710
708
706
714
716
ALU2
FALU
MOLT
712
650
INT FU
652
FP FU

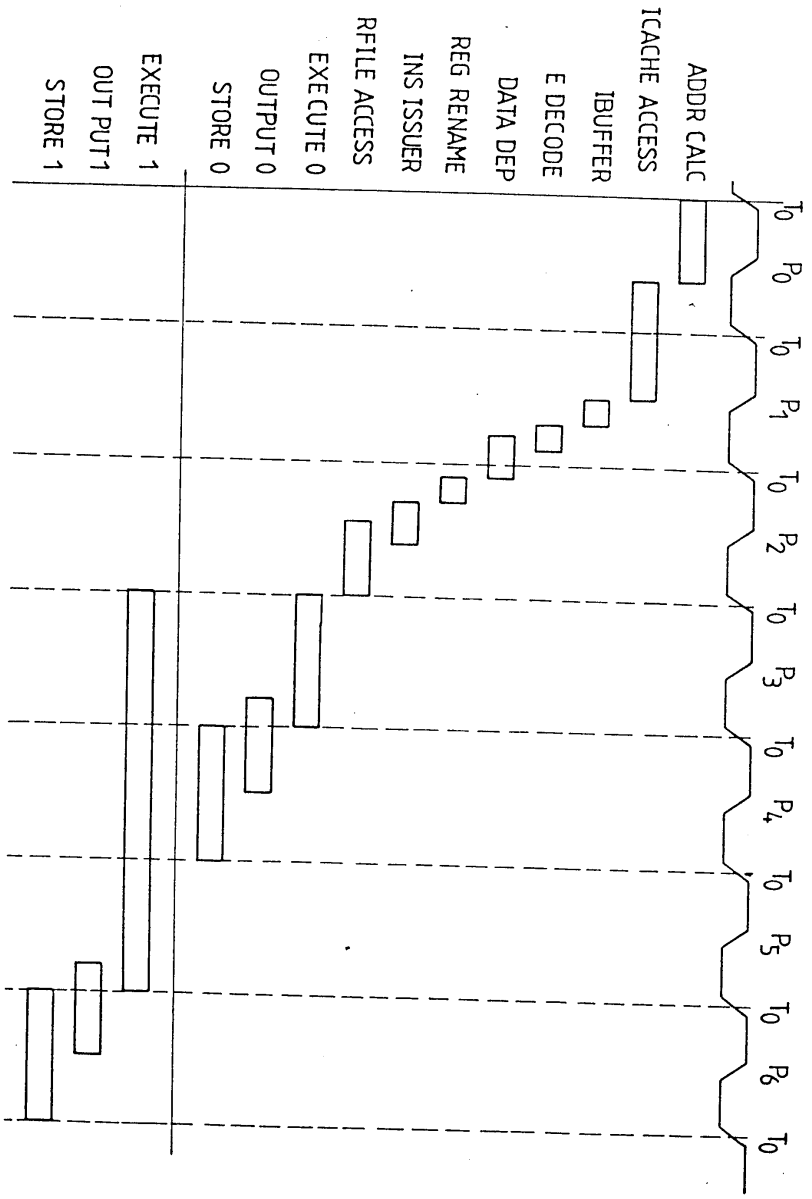
도면12


SPECIAL
REG DATA
354
726
720
멀 티 플 렉 서
730
730
728
임 시 버 퍼
724
734
736
레지스터
파일 어레이
732
723
740
ED ECODE
멀 티 플 렉 서
738
ALU0 FU
ALU1 FU
ALU2 FU
FALU FU
입력선택기
및
비교연산자
유 닛 트
722
744
바이패스 유닛트
742
748
750
752
746
부울 연산
기능 유닛트
CF0
CF1

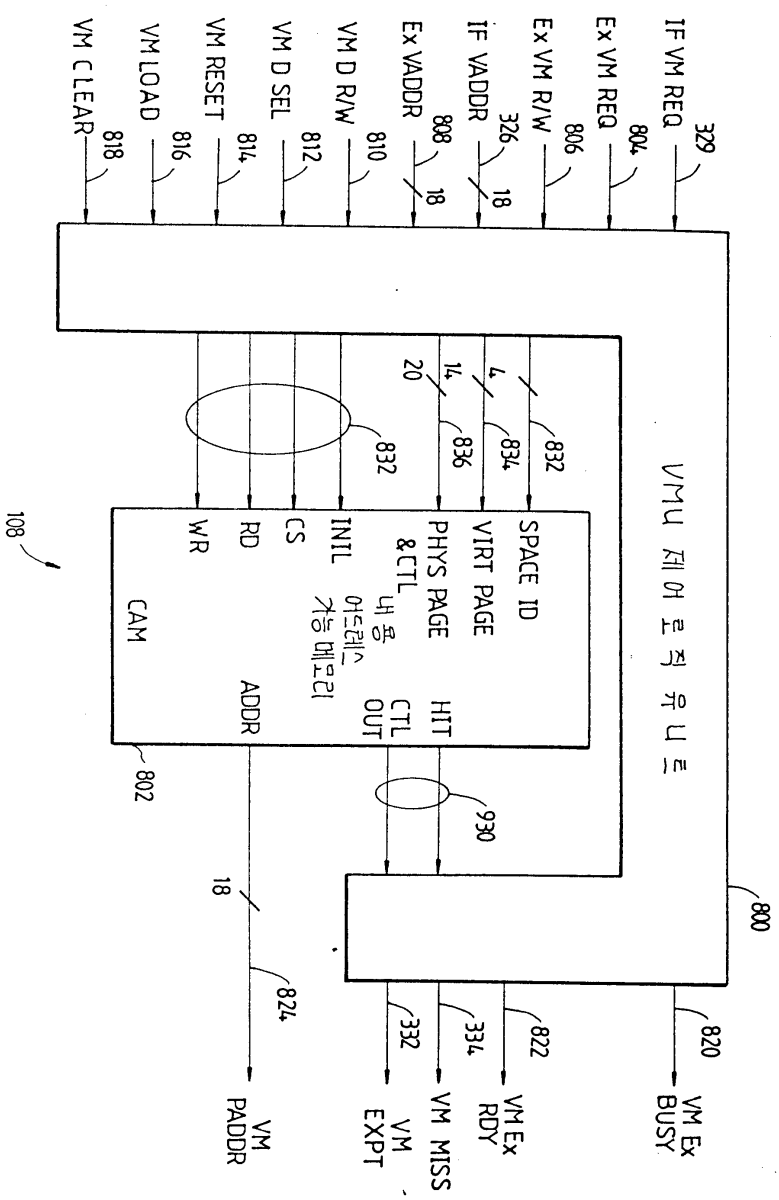
도면13


정렬 유니트
레지스터 파일 어레이
SD0
SD1
SD2
SD3
SA0
SA1
SA2
SA3
CCU DATA
CCU PADDR [31:0]
SEL
CCU CTL
ExID [4:0]
모드/상태 제어 유니트
4 × 4 비 교 기
LA0
LA1
LA2
LA3
IFU
IEU
PADDR
VADDR
VMU
32 [31:0]
[13:0] 14
18
64
760
762
764
766
768
770
772
774
776
778
782
786
788
790
792
794
796
798
102
104
108
326
472
654
658

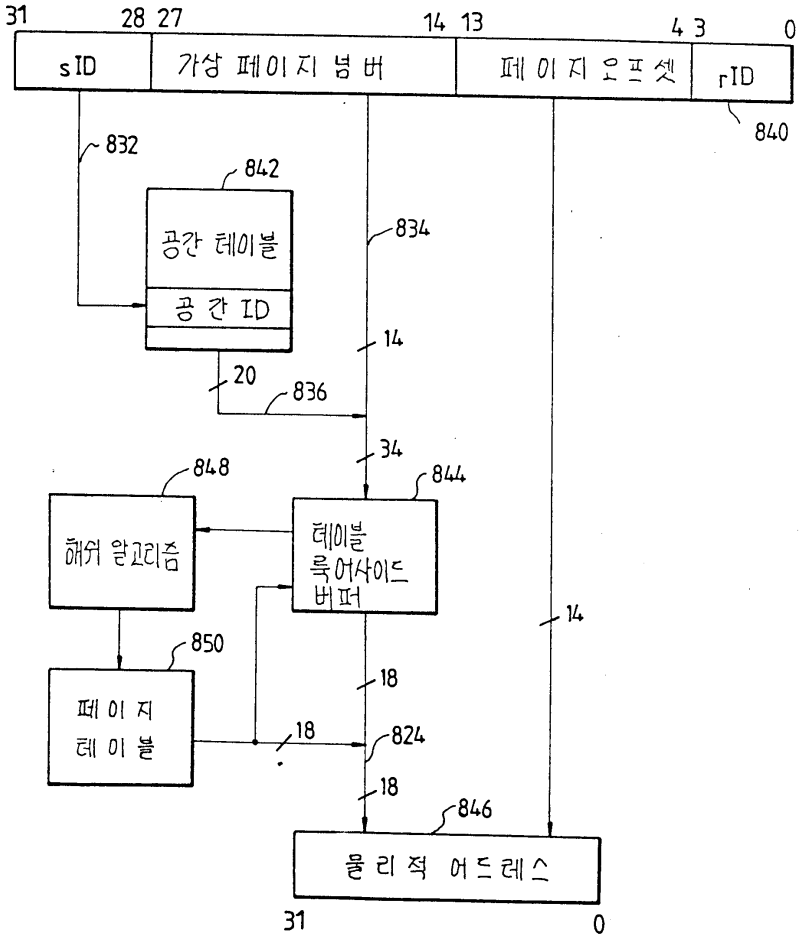
도면14


ADDR CALC
ICACHE ACCESS
IBUFFER
E DECODE
DATA DEP
REG RENAME
INS ISSUER
RFILE ACCESS
EXE CUTE 0
OUTPUT 0
STORE 0
EXECUTE 1
OUT PUT1
STORE 1
T0 P0 T0 P1 T0 P2 T0 P3 T0 P4 T0 P5 T0 P6 T0

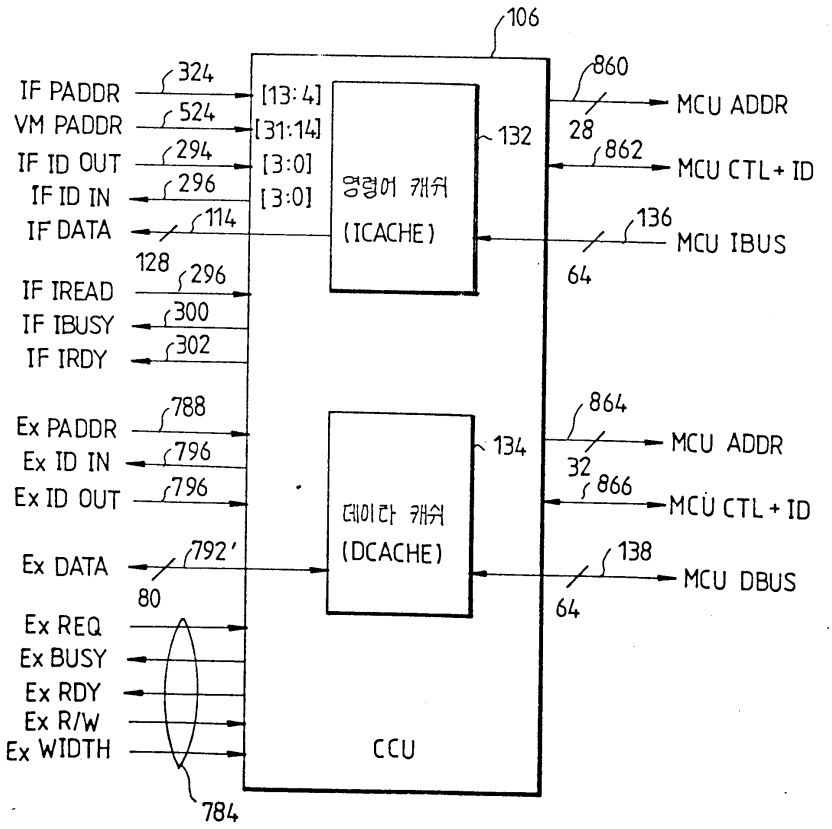
도면15


IF VM REQ
329
Ex VM REQ
804
Ex VM R/W
806
IF VADDR
326
18
Ex VADDR
808
18
VM D R/W
810
VM D SEL
812
VM RESET
814
VM LOAD
816
VM CLEAR
818
VM 제어 로직 유니트
800
SPACE ID
VIRT PAGE
PHYS PAGE &CTL
HIT
CTL OUT
INIL
CS
RD
WR
ADDR
CAM
내용 어드레스 가능 메모리
802
108
832
834
836
20
14
4
930
18
824
VM PADDR
332
VM EXPT
334
VM MISS
822
VM Ex RDY
820
VM Ex BUSY

도면16


31
28 27
14 13
4 3
0
sID
가상 페이지 넘버
페이지 오프셋
rID
840
832
842
공간 테이블
공간 ID
834
14
20
836
34
848
844
해쉬 알고리즘
테이블 룩어사이드 버퍼
14
850
18
페이지 테이블
18
824
18
846
물리적 어드레스
31
0

도면17


106
IF PADDR
324
[13:4]
VM PADDR
524
[31:14]
IF ID OUT
294
[3:0]
IF ID IN
296
[3:0]
IF DATA
114
128
IF IREAD
296
IF IBUSY
300
IF IRDY
302
명령어 캐쉬
(ICACHE)
132
860
MCU ADDR
28
862
MCU CTL+ID
136
MCU IBUS
64
Ex PADDR
788
Ex ID IN
796
Ex ID OUT
796
Ex DATA
792'
80
데이타 캐쉬
(DCACHE)
134
864
MCU ADDR
32
866
MCU CTL+ID
138
MCU DBUS
64
Ex REQ
Ex BUSY
Ex RDY
Ex R/W
Ex WIDTH
784
CCU