(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) Int. Cl.[6]
G06F 9/44

(45) 공고일자 2000년08월01일
(11) 등록번호 10-0241063
(24) 등록일자 1999년11월01일

(21) 출원번호 10-1997-0012608
(22) 출원일자 1997년04월04일
(65) 공개번호 특1998-0018064
(43) 공개일자 1998년06월05일

(30) 우선권주장 8/699280 1996년08월19일 미국(US)

(73) 특허권자 삼성전자주식회사 윤종용
경기도 수원시 팔달구 매탄3동 416

(72) 발명자 성윤 피터 송
미국 캘리포니아 94022 로스 알토스 캐시타 웨이 717
모아타즈 에이. 모하메드
미국 캘리포니아 95050 산타 클라라 #7 와버톤 애비뉴 1721
르 누옌
미국 캘리포니아 95030 몬테 세레노 다니엘 플레이스15096
헌철 박
미국 캘리포니아 95014 쿠퍼티노 #7 파크우드 드라이브 10140
제리 반 아켄
미국 워싱턴 98052 레드몬드 엔이 99스 웨이 아이비 226
알렉산드로 포린
미국 워싱턴 98052 레드몬드 엔이 179스 애비뉴 2321
앤드류 라프만
미국 워싱턴 98072 우드인빌 엔이 205스 스트리트 17835

(74) 대리인 김능균

심사관 : 엄인권

(54) 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구장치 및 방법

요약

멀티타스킹 처리시스템 환경에서 하나의 프로그램이 정지되고 콘텍스트 변경됨으로써 프로세서가 실행을 위해 연속되는 프로그램내의 콘텍스트를 변경할 수 있다. 콘텍스트 변경되는 프로그램의 상태를 나타내는 프로세서 상태정보가 존재한다. 이 프로세서 상태정보의 저장에 의해 콘텍스트 변경된 프로그램이 성공적으로 재실행된다. 콘텍스트 변경된 프로그램이 연속적으로 콘텍스트 변경되면, 저장된 프로세서 정보는 실행이 정지된 지점에서 프로그램을 성공적으로 다시 시작하기 위한 준비로서 로드된다. 넓은 영역의 메모리가 프로세서 상태정보의 저장을 위해 할당될 수 있으나, 이중 일부분만이 콘텍스트 변경된 프로그램의 성공적인 저장 및 재실행을 위해 콘텍스트 변경되는 동안 보존될 필요가 있다. 불필요하게 모든 이용가능한 프로세서 상태정보를 저장 및 로드하는 것은 특히 비교적 많은 양의 프로세서 상태정보가 존재하는 곳에서 비효율적일 수 있다. 프로세서는 코프로세서가 현재 실행중인 프로그램을 콘텍스트 변경하도록 요구한다. 실행중인 프로그램내의 소정의 적절한 지점에서 프로그램 실행을 중지하고 성공적인 프로그램의 복구를 위해 필요한 최소량의 프로세서 상태정보만을 저장함으로써 코프로세서는 응답한다. 적절한 지점은 응용 프로그래머에 의해 콘텍스트 변경시 최소량의 프로세서 정보의 보존을 요구하는 실행중인 프로그램내의 위치에서 선택된다. 최소량의 프로세서 정보만을 저장함으로써 프로세서 시간이 콘텍스트 저장 및 복구 동작동안에 축적된다.

대표도

도1

명세서

도면의 간단한 설명

제1도는 본 발명의 일실시예에 의한 멀티미디어 멀티프로세서 시스템을 나타낸 블럭도.

제2도는 제1도에 나타낸 멀티미디어 멀티프로세서 시스템의 멀티미디어신호 프로세서를 나타낸 블럭도.

제3도는 제2도에 나타낸 멀티미디어신호 프로세서에 있어서의 코프로세서(co-processor)를 나타낸 블럭도.

제4도는 제3도에 나타낸 코프로세서의 코프로세서 실행 데이타경로를 나타낸 블럭도.

제5도는 멀티미디어신호 프로세서의 펌웨어 구조를 나타낸 블럭도.

제6도는 멀티프로세서구조에 있어서의 효율적인 프로그램 콘텍스트 저장 및 프로그램 복구와 관련된 동작을 나타낸 흐름도.

제7도는 군데군데 위치한 조건적인 콘텍스트 변경명령을 갖는 실행가능한 프로그램의 흐름부분을 나타낸 도면.

발명의 상세한 설명

발명의 목적

발명이 속하는 기술분야 및 그 분야의 종래기술

본 발명은 컴퓨팅시스템(computing system)에 관한 것으로, 특히 예컨대 멀티프로세서 구조를 갖는 멀티타스킹(multi-tasking)환경에서의 콘텍스트(context)변경작업에 관한 것이다.

일반적으로 멀티타스킹이란 컴퓨터에서 시분할 프로그램을 수행하여 여러개의 프로그램을 동시에 실행하는 것이다. 실제로 컴퓨터가 다수개의 프로그램들간을 번갈아가며 실행할때 컴퓨터 사용자는 프로그램들이 병렬로 실행된다고 생각한다. 프로그램 실행이 시분할되면, "콘텍스트의 변경(context switched out)"(타스크 변경)이 일어나기 전의 소정시간동안 한 프로그램이 실행된 다음 다른 프로그램이 실행된다.

콘텍스트 변경된 프로그램을 다시 시작할때, 이 프로그램은 그전에 실행이 종료되었던 정확한 위치에서부터 다시 프로그램의 실행이 시작되어야 한다. 콘텍스트 변경된 프로그램을 다시 실행할 것을 대비하여 프로세서는 다른 프로그램을 실행하기 전에 콘텍스트 변경되는 프로그램의 상태를 저장하는 과정을 거친다. 이 프로그램 상태는 프로그램이 콘텍스트 변경될때 프로세서의 상태에 의해 다양한 메모리 위치로 나타낸다.

프로그래머의 입장에서 볼때, 콘텍스트 변경은 임의의 시간에 프로그램내의 임의의 위치에 발생하기 때문에 콘텍스트 변경된 프로그램의 프로그래머에게는 명료한 것이다. 또한, 콘텍스트 변경을 수행하는 구동시스템은 콘텍스트 변경되는 프로그램의 상태를 알지 못한다. 결과적으로, 모든 프로세서의 상태정보는 프로그램이 콘텍스트 변경될때 반드시 저장되어야 한다. 이 프로세서의 상태정보에는 모든 구조적이고 그 소프트웨어를 알 수 있는 레지스터뿐 아니라 스크래치 패드(scratch pad)메모리와 같은 프로세서내에서만 알려진 어드레스들로 매핑되는 메모리 위치들이 포함된다.

많은 양의 상태정보를 갖는 프로세서 구조를 위한 모든 프로세서 상태정보의 저장 및 복구와 관련된 종래의 프로그램 콘텍스트 변경방법은 효율적이지 않으며, 프로세서의 동작에 부정적인 영향을 미칠 수 있다. 프로세서 상태정보 저장부에 할당된 억세스되는 다수의 메모리 위치들과, 상대적으로 낮은 대역폭의 버스를 통해 다른 메모리 위치로 프로세서의 상태정보를 기입하는 동작을 필요로 하는 프로세서 동작에 상기 종래방법에 있어서의 콘텍스트 변경과 관련된 비효율적이고 부정적인 영향이 부분적으로 기여할 수 있다. 종래의 콘텍스트 변경방법에 있어서의 이러한 비효율성은 또한 저장된 모든 프로그램 상태정보가 메모리 위치에 할당된 프로세서 정보상태로 다시 전달될때 프로세서 복구과정동안 프로세서의 동작에 부정적인 영향을 미친다. 이러한 비효율성은 각각의 프로그램 콘텍스트가 변경되는 동안 축적된다. 예를 들면, 멀티미디어 신호 프로세서는 100개이상의 레지스터와 메모리위치내의 7킬로바이트 이상으로 표현되는 프로세서 상태를 가질 수 있다. 각각의 콘텍스트 변경시에는 통상적으로 이러한 모든 정보가 프로세서 상태 메모리 저장위치로 전달되어야 한다.

멀티미디어신호 프로세서를 채용한 멀티미디어 시스템과 같은 멀티타스킹 시스템에 있어서의 콘텍스트 변경과 관련된 부정적인 영향을 줄이기 위해서는 프로그램들의 실행시 그들사이의 원치않는 지연을 감소시켜야 한다. 이는 프로그램들이 멀티미디어처리 환경에서와 같은 많은 양의 정보를 처리하는데 관련될 경우에 두드러진 효과를 나타낸다.

발명이 이루고자 하는 기술적 과제

본 발명은 프로그램들간의 콘텍스트 변경를 위해 필요한 처리시간을 줄인다. 콘텍스트 변경되는 프로그램에 있어서의 몇몇 지점은 나중에 프로그램 실행을 다시 성공정으로 시작하기 위해 다른 지점들보다 프로세서 상태정보를 더 많이 저장할 것을 요구한다. 본 발명의 일실시예에 있어서, 멀티프로세서구조는 현재 실행중인 프로그램의 콘텍스트 변경에 대비하여 프로세서가 현재의 프로그램을 실행하는 다른 프로세서를 요구하도록 하여 적절한 지점에서 그 자신을 인터럽트한다. 상기 적절한 지점은 콘텍스트 변경된 프로그램을 성공정으로 다시 실행시키기 위해 필요한 프로세서 상태정보를 줄일 수 있고 콘텍스트 변경요구를 반드시 즉각적으로 받아들이지 않을 수 있는 현재 실행중인 프로그램내의 지점이다.

본 발명의 다른 실시예에 있어서, 멀티타스킹 환경에 있어서의 콘텍스트 저장 및 복구는 프로세서와 현재 실행중인 응용프로그램이 담당한다. 이 프로그램은 예컨대, 프로그램의 실행을 성공적으로 다시 시작하기 위해 감소된 양의 프로세서 상태정보를 필요로 하는 프로그램내의 지점들에 상응하는 다수의 위치내의 조건적인 콘텍스트 변경 프로그램명령으로 표시된다. 프로그램이 상기 표시된 위치에 도달하고, 콘텍스트 변경요구의 접수가 검출되면, 프로그램은 프로그램의 성공적인 재실행을 위해 필요한 프로세서 상태정보만을 저장하기 시작한다. 프로그램은 콘텍스트 변경이 요구되고 이에 응답하여 콘텍스트가 변경될때 그 사이의 지연을 감소시키는데 충분할만큼 빈번하게 표시되어야 한다.

본 발명의 다른 실시예에 있어서, 제 1 프로세서상에서 제 1 프로그램을 실행하는 단계와, 제 2 프로세서

로부터 콘텍스트 변경요구를 받는 단계, 및 상기 제 1 프로그램의 적절한 지점에서 상기 콘텍스트 변경요구에 응답하는 단계를 포함하는 처리과정에 있어서, 상기 적절한 지점은 프로그램의 성공적인 복구를 위한 최소량의 프로세서 상태정보 저장을 필요로 하는 제 1 프로그램내의 가장 가까운 지점을 나타내는 프로그램내의 표시와 관련된 것이다.

또다른 실시예에 있어서, 본 발명의 컴퓨팅시스템은 복수개의 군데군데 배치된 콘텍스트 변경 표시를 구비한 프로그램을 실행하기 위한 멀티타스킹 환경내의 제 1 프로세서와, 상기 제 1 프로세서에 연결되고 프로세서 상태정보를 저장하는 역할을 하는 제 1 메모리를 포함한다. 상기 컴퓨팅시스템은 또한 제 1 프로세서에 연결된 제 2 메모리와, 프로세서가 어떠한 실행 프로그램내의 표시들중의 하나와 만난 후에 프로그램의 변경요구를 검출하기 위한 제 1 프로세서상에서 동작하는 콘텍스트 변경요구 검출부, 및 제 1 메모리내에 위치하는 프로세서 상태정보를 제 2 메모리에 저장함으로써 검출된 콘텍스트 변경요구에 응답하는 제 1 프로세서상에서 동작하는 콘텍스트 저장모듈을 포함한다.

발명의 구성 및 작용

이하, 첨부한 도면을 참조로 하여 본 발명의 바람직한 실시예를 상술하며, 도면 전체를 통하여 동일한 부분에는 동일한 도면부호를 사용하기로 한다.

도 1 은 호스트 프로세서(102)와 멀티미디어신호 프로세서(200)를 구비한 멀티미디어 멀티프로세서 시스템(100)을 도시한 블럭도이다. 도 1 을 참조하면, 호스트 프로세서(102)는 인텔사의 Pentium™ 또는 Pentium Pro™ 프로세서와 같은 x86 프로세서이다. 호스트 프로세서(102)는 시스템 메모리(104)와 캐쉬(105)에 본래 저장되어 있는 명령 및 데이타에 기초한 프로그램을 실행한다. 호스트 프로세서(102)는 PC 칩세트(107)와 PCI버스와 같은 시스템버스(106)을 통해 멀티미디어신호 프로세서(200)와 연결된다. 멀티미디어신호 프로세서(200)는 오디오 및 전화통신을 받아 들이기 위한 오디오 및 통신 CODEC, 비디오입력신호를 받아 들이기 위한 비디오 A/D컨버터(110), 비디오출력신호를 전송하기 위한 D/A컨버터(112) 및 프레임 버퍼 SDRAM메모리(114)와 같은 다양한 기능블럭들과 접속된다. 본 발명의 일실시예에 있어서, 상기 멀티미디어신호 프로세서는 삼성반도체 MSP계열의 멀티미디어신호 프로세서(삼성 MSP)중의 어느 하나일 수 있다.

도 2 에 상기 멀티미디어 멀티프로세서 시스템(100)의 멀티미디어신호 프로세서(200)를 블럭도로 나타내었다. 도 2 를 참조하면, 멀티미디어신호 프로세서(200)는 패스트버스(fast bus;FBUS)(210)를 통해 예컨대, 32비트 PCI버스 인터페이스(222), 64비트 SDRAM메모리콘트롤러(226), 8채널 DMA콘트롤러(220), ASIC 로직블럭(216), 및 호스트 프로세서(102)와 프레임버퍼 SDRAM메모리(114)간의 데이타 전송을 위한 메모리 데이타 무버(mover)(224)를 포함하는 복수개의 FBUS주변장치들에 접속된다. 상기 PCI버스 인터페이스(222)는 시스템버스(106)에 접속되면, 예컨대 33MHz에서 동작한다. ASIC로직블럭(216)은 원하는 기능의 수행을 위한 제어로직을 제공한다. 본 발명의 일실시예에 있어서, ASIC로직블럭(216)은 여러 아날로그 CODEC 및 주문용 I/O장치와 접속된 인터페이스를 포함하는 10K게이트를 제공한다. 메모리데이타 무버는 호스트프로세서(102)로부터의 DMA데이타를 멀티미디어신호 프로세서(200)와 연결되는 SDRAM메모리(114)로 전송한다. DSP코어(201)는 I/O버스를 통해 예컨대, 8254-호환(compatible) 프로그램가능한 인터벌 타이머(programmable interval timer)(228), 16450-호환 UART 직렬라인(230), 8259-호환 프로그램가능한 인터럽트 콘트롤러(232), 및 비디오 비트스트림을 처리하기 위한 비트스트림 프로세서(234)를 포함하는 복수개의 I/O버스장치에 접속된다. 비트스트림 프로세서(234)와 관련된 보다 상세한 내용은 미국특허출원 일련번호 xx/xxx,xxx (변리사 사건번호 M-4368)인 C.Reader의 "Methods and Appartus for Processing Video Data"에 기술되어 있다.

DSP코어(201)는 멀티미디어신호 프로세서(200)의 계산의 핵심부로서, 프로세서(202), 코프로세서(204), 캐쉬 서브시스템(208), FBUS(210), 및 I/O버스(212)를 포함한다. 본 발명의 일실시예에 있어서, 프로세서(202)는 콘텍스트 변경 요구, 실시간 구동시스템의 구동, 인터럽트 및 예외(exception)처리, 입출력장치 관리, 호스트프로세서(102)와의 접속등과 같은 일반적인 처리기능을 수행하는 ARM7™ RISC 제어프로세서이다. 프로세서(202)는 40MHz로 동작하며, 코프로세서 인터페이스(206)를 통해 코프로세서(204)와 접속된다.

프로세서(202)는 명령이 처리되는 동안 발생하는 조건인 예외(exception)에 응답하여 예외처리를 수행하여 프로그램실행의 제어흐름이 변경되도록 한다. 예외처리에 관한 보다 상세한 것은 미국특허출원 일련번호 xx/xxx,xxx (변리사 사건번호 M-4366)인 Song등의 "System Ad Method For Handling software Interrupt With Argument Passing"과, 일련번호 xx/xxx,xxx (변리사 사건번호 M-4367)인 Song등의 "System And Method For Handling Interrupt And Exception Events In An Asymmetric multiprocessor Architecture"에 기술되어 있다.

코프로세서(204)는 멀티미디어신호 프로세서(200)의 디지털신호 처리 구동부이다. 본 발명의 일실시예에 있어서, 코프로세서(204)는 삼성MSP계열의 벡터프로세서이다. 벡터프로세서로서 코프로세서(204)는 SIMD(Single-Instruction-Multiple-Data)구조를 가지며, DCT(Discrete Cosine Transforms), FIR필터링, 회선(convolution), 비디오 움직임평가 및 다른 프로세싱구동과 같은 신호처리기능을 수행하는 것과 병렬로 다수의 데이타요소들에 작용하는 경로연결된(pipelined) RISC구동부를 포함한다. 코프로세서(204)는 다수의 데이타요소들이 복수개의 벡터실행유니트들에 의해 벡터처리되는 식으로 병렬로 처리되는 벡터산술을 지원한다. 코프로세서(204)는 스칼라처리 및 조합된 벡터-스칼라 처리를 모두 실행한다. 코프로세서(204)의 다수의 데이타요소들은 주기당(예를 들어 12.5ns) 32개의 8/9비트 고정 지점 산술처리, 16개의 16비트 고정지점 산술처리, 또는 8개의 32비트 고정지점 또는 플로팅지점 산술처리의 비율로 계산되는 576비트 벡터에 패킹된다. 대부분의 32비트 스칼라처리는 1주기당 1명령 비율로 경로연결되고, 대부분의 576비트 벡터처리는 2주기당 1명령 비율로 경로연결된다. 로드(load) 및 저장처리는 산술 처리와 겹쳐지며, 별도의 로드 및 저장회로에 의해 독립적으로 수행된다.

도 3을 참조하면, 코프로세서(204)는 명령페치유니트(instruction fetch unit)(302), 명령디코더 및 이슈

어(issuer)(304), 명령실행 데이타경로(306) 및 로드 및 저장유니트(308)를 포함하는 4개의 기능블럭들을 구비한다. 명령페치유니트(302) 및 명령디코더 및 이슈어(304)는 코프로세서(204)에 포함되어 코프로세서(204)가 프로세서(202)와는 별도로 동작하도록 한다.

명령페치유니트(302)는 명령 및 서브루틴명령으로의 분기(Branch) 및 점프(Jump)와 같은 프로세스제어 흐름명령을 프리페치(prefetch)한다. 명령페치유니트(302)는 현재 실행중인 스트림을 위한 16열의 프리페치된 명령들과 브랜치 타겟 스트림(Branch target stream)을 위한 8열의 프리페치된 명령들을 포함한다. 명령페치유니트(302)는 명령캐쉬로부터 1주기동안 8개까지의 명령을 256비트넓이의 버스를 통해 입력받는다. 명령디코더 및 이슈어(304)는 코프로세서(204)에 의해 실행되는 모든 명령들을 디코딩하고 준비시킨다. 디코더는 명령페치유니트(302)로부터의 입력순서에 따라 1주기동안 1명령을 처리하고, 이슈어는 실행 수단 및 연산수데이타의 능력에 따라 순서에 관계없이 대부분의 명령들을 준비시킨다.

도 4를 참조하면, 명령실행 데이타경로(306)는 4포트 레지스터파일(402), 8개의 32×32 병렬 승산기(multiplier)(404), 8개의 36비트 ALU(406)를 포함한다. 레지스터파일(402)은 주기당 2개의 읽기동작과 2개의 쓰기동작을 지원한다. 병렬 승산기(404)는 정수 또는 플로팅포인트(floating point) 포맷으로 8개의 32비트 승산을 제공하거나 주기당 16개의 16비트 승산 또는 32개의 8비트 승산을 제공한다. ALU(406)는 주기당(예컨대 12.5ns) 정수 또는 플로팅포인트 포맷으로 8개의 36비트 ALU처리, 16개의 16비트 ALU처리, 또는 32개의 8비트처리를 실행한다.

레지스터파일(402)은 복수개의 특수목적 레지스터 및 복수개의 복귀어드레스(return address) 레지스터를 포함한다. 특수목적 레지스터는 벡터 제어 및 상태 레지스터(VCSR), 벡터프로그램 카운터(VPC), 벡터 예외프로그램 카운터(vector exception program counter;VEPC), 벡터 인터럽트 발생원 레지스터(vector interrupt source register;VISRC), 벡터 및 제어프로세서 동기와 레지스터(vector and control processor synchronization register;VASYNC) 및 다양한 카운트, 마스크, 오버플로우 및 브레이크포인트 레지스터등과 같은 다른 레지스터들을 포함한다. VPC는 코프로세서(204)에 의해 실행될 다음 명령의 어드레스이다.

VISRC는 프로세서(202)에 인터럽트 발생원을 나타낸다. VISRC의 적절한 비트들은 예외(exception)검출에 의해 하드웨어에 의해 설정된다. 이 비트들은 코프로세서(204)의 재실행전에 소프트웨어에 의해 리셋된다. VISRC내의 어떤 설정비트는 코프로세서(204)가 아이들상태(idle state)(VP_IDLE)로 들어가게 한다. 해당 인터럽트 인에이블 비트가 코프로세서 인터페이스(206)내의 VIMSK레지스터내에 설정되면, IRQ 인터럽트는 프로세서(202)로 신호를 알린다.

코프로세서(204)는 정확한 예외와 부정확한 예외를 포함하는 예외조건을 검출한다. 정확한 예외는 코프로세서(204)에 의해 검출되어 잘못된 명령전에 보고된다. 정확한 예외는 명령어드레스 브레이크포인트 예외, 데이타어드레스 브레이크포인트 예외, 부당한 명령예외, 단일스텝 예외, 복구어드레스 스택 오버플로우 예외, 복구어드레스 스택 언더플로우 예외, VCINT예외, 및 VCJOIN예외를 포함한다. 코프로세서(204)의 부정확한 예외는 잘못된 명령으로의 프로그램순서상 뒷쪽인 일정치 않은 수의 명령들의 실행후에 검출되고 보고된다. 부정확한 예외는 부당한 명령어드레스 예외, 부당한 데이타어드레스 예외, 정렬되지 않은 데이타억세스 예외, 정수 오버플로우 예외, 플로팅포인트 오버플로우 예외, 플로팅포인트 부당 연산수 예외, 플로팅포인트의 0에 의해 나눗셈 예외, 및 정수의 0에 의해 나눗셈 예외를 포함한다.

명령이 확장되어 프로세서(202)를 인터럽트할때 벡터 인터럽트 명령레지스터(VIINS)는 코프로세서 VCINT 또는 VCJOIN명령으로 업데이트된다.

프로세서(202)는 코프로세서(204)의 구동을 개시한다. 이에 대한 보다 상세한 설명은 미국특허출원 일련번호 xx.xxx,xxx (변리사 사건번호 M-4366)의 Song등의 "System and Methods for Handling Software Interrupts with Argument Passing"과 일련번호 xx/xxx,xxx (M-4367)의 Song등의 "System and Method for Handling Interrupt and Exception Events in an Asymmetric Multiprocessor Architecture"에 기술되어 있다.

벡터 인터럽트 마스크 레지스터(VIMSK)는 코프로세서(204)내에서 일어나는 예외를 프로세서(202)로 보고하는 것을 제어한다. VIMSK내의 비트들은 벡터 인터럽트 발생원 레지스터(VISRC)내의 해당비트를 따라 설정되면 예외를 인에이블시켜 프로세서(202)를 인터럽트한다. VISRC레지스터는 복수개의 예외들 및 명령중의 어느 것이 발생원인지를 나타내는 복수개의 비트들을 포함한다. VIMSK레지스터는 데이타 어드레스 브레이크포인트 인터럽트 인에이블(data address breadkpoint interrupt enable;DABE), 명령어드레스 브레이크포이트 인터럽트 인에이블(IABE), 및 단일스텝 인터럽트 인에이블(single-step interrupt enable;SSTPE)을 포함한다. VIMSK는 플로팅포인트 오버플로우(FOVE), 부당 연산수(FINVE) 및 0에 의한 나눗셈(divide-by-zero)(FDIVE) 인터럽트 인에이블 비트, 및 정수 오버플로우(IOVE) 및 0에 의한 나눗셈(IDIVE) 인터럽트 인에이블을 제어한다. VIMSK는 또한 VCINT인터럽트 인에이블(VIE), VCJOIN인터럽트 인에이블(VJE) 및 콘텍스트 변경 인에이블(CSE)을 제어한다.

코프로세서(204)는 프로세서(202)로 신호를 보냄으로써 프로세서(202)와 상호작용을 한다. 특히, 코프로세서(204)는 코프로세서(204)가 동기되는 명령을 실행했다는 것을 나타내는 사용자확장된 레지스터를 통해 간접적으로 프로세서(202)로 신호를 보낸다. 코프로세서(204)는 코프로세서(204)가 예외를 정지시키고 VP_IDLE상태로 들어갔다는 것을 나타내는 인터럽트 요구를 통해 프로세서(202)로 신호를 직접 보낸다. 코프로세서(204)는 프로세서(202)로의 신호전송을 위해 2개의 명령을 실행한다. VCJOIN명령(VCJOIN n)은 프로세서(202)와 조건적으로 합해지며, 코프로세서(204)가 정지하고 VP_IDLE상태로 들어가도록 한다. 코프로세서(204)내의 프로그램 카운터(도시하지 않음)는 VCJOIN명령에 따라 명령을 어드레스 한다. 코프로세서(204)에 의해 실행되는 VCJOIN명령은 제어흐름등급으로 분류된다. 제어흐름명령은 브랜치, 감소 및 브랜치, 점프, 서브루틴으로부터의 복귀, 콘텍스트 변경, 및 배리어명령과 같은 다양한 조건적 명령들을 포함한다.

도 2를 참조하면, 캐쉬 서브시스템(208)은 데이타캐쉬(236)(예컨대, 5KB), 명령캐쉬(238)(예컨대, 2KB), 및 캐쉬 ROM(240)(예컨대, 16KB)을 포함하며, 코프로세서(204)의 속도(80MHz)와 동일한 속도로 동작한다. 본 발명의 일실시예에 있어서, 캐쉬 서브시스템(208)은 프로세서(202)를 위한 1킬로바이트의 명령 저장영

역 및 1킬로바이트의 데이타 저장영역과, 코프로세서(204)를 위한 1킬로바이트의 명령 저장영역과 4킬로바이트의 데이타 저장영역, 및 프로세서(202)와 코프로세서(204)를 위한 공유 16킬로바이트의 집적화된 명령 및 데이타 캐쉬ROM을 구비하고 있다. 캐쉬 서브시스템(208)은 32비트 데이타버스를 통해 프로세서(202)와 접속되고, 128 비트 데이타버스를 통해 코프로세서(204)와 접속된다. 캐쉬ROM(204)은 μROM초기화 소프트웨어, 셀프테스트 진단 소프트웨어, 다양한 시스템관리 소프트웨어, 라이브러리 루틴 및 선택된 명령 및 데이타 상수들을 위한 캐쉬를 포함한다. 특히,캐쉬 ROM(240)은 프로세서(202)를 위한 명령 예외처리기 및 입출력장치 인터럽트 처리기(0,1,2,3)를 구비하고 있다. 또한 캐쉬ROM(240)은 프로세서(202)내에서 실행되는 벡터프로세서 인터럽트처리기와 벡터프로세서 브레이크포인트 배제 처리기를 포함한다.

도 5는 멀티미디어신호 프로세서(200)의 소프트웨어 및 펌웨어 구조(500)를 나타낸 것으로, 멀티미디어신호 프로세서는 멀티미디어신호 프로세서(200)상에서 실행되는 MSP시스템 요소 소프트웨어(502)와 호스트 프로세서(102)상에서 실행되는 PC응용 및 구동시스템 소프트웨어(508)를 포함한다. 멀티미디어신호 프로세서(200)는 코프로세서(204)상에서 실행되는 벡터화된 DSP 펌웨어 라이브러리와 프로세서(202)상에서 실행되는 시스템관리 기능블럭(506)을 포함하는 펌웨어에 의해 제어된다. 상기 벡터화된 DSP 펌웨어 라이브러리(504) 및 시스템관리 기능블럭(506)은 MSP시스템요소 소프트웨어(502)내에 포함된다. 상기 소프트웨어 및 펌웨어구조(500)는 호스트 응용 제어동작으로부터 신호처리 기능을 분리하여 소프트웨어 개발을 단순화시키고, 소프트웨어 설계 관계를 향상시키며, 응용 개발 및 유지비용을 감소시킨다.

MSP시스템요소 소프트웨어(502)는 프로세서(202)상에서 독자적으로 실행되며 MSP실시간 구동시스템 핵심부(510), 멀티미디어 라이브러리 모듈(512), 시스템 관리 기능블럭(506) 및 벡터화된 DSP 펌웨어 라이브러리(504)를 포함한다. 마이크로소프트상의 MMOSA 실시간 커널(Kernel)인 MSP실시간 핵심부(510)는 호스트프로세서(102), 자원관리, I/O장치 구동 및 대부분의 인터럽트 및 예외처리로의 접속을 관장한다. MSP 실시간 핵심부(510)는 호스트 프로세서(102)상에서 실행되는 Windows ™ 및 Windows NT™ 소프트웨어와의 접속을 위한 소프트웨어를 포함한다. MSP 실시간 핵심부(510)는 호스트 프로세서(102)로부터 소정의 응용펌웨어를 선택하고 다운로드하기 위한 소프트웨어, 프로세서(202)와 벡터프로세서(204)내에서의 실행을 위한 타스크준비를 위한 소프트웨어, 및 메모리 및 I/O장치를 포함하는 멀티미디어신호 프로세서(200)의 시스템자원을 관리하기 위한 소프트웨어를 포함한다. MSP실시간 핵심부(510)는 또한 멀티미디어신호 프로세서(200)의 타스크들간의 동기 접속을 위한 소프트웨어와, MSP관련 인터럽트, 예외 및 상태조건을 보고하기 위한 소프트웨어를 포함한다.

벡터화된 DSP펌웨어 라이브러리(504)는 모든 디지털신호 처리기능을 수행한다. 벡터화된 DSP펌웨어 라이브러리(504)는 또는 프로세서(202)에 의해 벡터프로세서(204)로 내려진 코프로세서 인터럽트, 또는 벡터프로세서(204)내에서 발생되는 하드웨어 스택 오버플로우 예외와 같은 특별한 인터럽트를 제어한다.

멀티미디어 라이브러리 모듈(512)은 데이타 통신, MPEG비디오 및 오디오, 스피치 코딩 및 합성, 사운드블래스터-호환 오디오등을 포함하는 통신처리 기능을 수행한다. MSP실시간 핵심부(510)은 멀티미디어신호 프로세서(200)상에서 실행되는 멀티미디어 응용을 용이하게 하는 견실한 실시간 멀티타스킹의 프리엠프티브(pre-emptive) 구동시스템이다.

호스트 프로세서(102)내에서 실행되는 PC응용 및 구동시스템 소프트웨어(508)는 시스템버스(106)를 통해 MSP제어 및 상태레지스터로의 읽기 및 쓰기동작, 및 시스템메모리(104)와 멀티미디어신호 프로세서(200)에 상주하는 공유 데이타구조로의 쓰기동작을 행함으로써 멀티미디어신호 프로세서(200)를 제어한다.

MSP프로그램실행은 제1실행 스트림을 수행하는 프로세서(202)로 시작된다. 프로세서(202)는 벡터 프로세서(204)내의 제2의 독립적인 실행스트림을 개시할 수 있다. 프로세서(202)와 벡터 프로세서(204)의 동작은 STARTVP, INTVP 및 TESTVP명령, 및 VJOIN 및 VINT명령을 포함하는 벡터 프로세서(204)내에서 실행되는 특수명령을 포함하는 프로세서(202)내에서 처리되는 특정한 코프로세서 명령들을 통해 동기된다. 프로세서(202)와 벡터 프로세서(204)간의 데이타전송은 프로세서(202)내에서 실행되는 데이타 이동명령에 의해 수행된다.

코프로세서(204)는 프로세서(202)로 신호를 보냄으로써 프로세서(202)와 상호작용을 한다. 특히, 코프로세서(204)는 코프로세서(204)가 동기되는 명령을 실행했다는 것을 나타내는 사용자확장된 레지스터를 통해 간접적으로 프로세서(202)로 신호를 보낸다. 코프로세서(204)는 코프로세서(204)가 예외를 정지시키고 VP_IDLE상태로 들어갔다는 것을 나타내는 인터럽트 요구를 통해 프로세서(202)로 신호를 직접 보낸다. 코프로세서(204)는 프로세서(202)로의 신호전송을 위해 2개의 명령을 실행하는데, VCJOIN명령과 프로세서(202)를 조건적으로 인터럽트하는 VCINT명령(VCJOIN n)은 코프로세서(204)가 정지하고 VP_IDLE상태로 들어가도록 한다. VCINT 및 VCJOIN명령은 제어흐름 등급으로 분류되는 코프로세서(204)에 의해 실행되는 명령들이다. 상기 제어흐름명령은 브랜치, 감소 및 브랜치, 점프, 서브루틴으로부터의 복귀, 콘텍스트 변경, 및 배리어명령과 같은 다양한 조건적 명령들을 포함한다.

멀티미디어 멀티프로세서 시스템(100)은 미국특허출원 일련번호 xx/xxx,xxx (변리사 사건번호M-4354)의 L. Nguyen의 "Mictoprocessor Operation in a Multimedia Signal Processor", 일련번호 xx/xxx,xxx (변리사 사건번호 M-4355)인 L. Nguyen의 "Single-Instruction-Multiple-Data Processing in a Multimedia Signal Processor", 일련번호 xx/xxx,xxx(M-4369)인 L. Nguyen등의 "Single-Instruction-Multiple-Data Processing Using Multiple Banks of Vector Registers", 일련번호 xx/xxx,xxx (M-4370)인 M.Mogamed등의 "Single-Instruction-Multiple-Data Processing With Combined Scalar/Vector Operations"에 상세히 기술되어 있다.

멀티미디어신호 프로세서(200)는 연속적인 프로그램실행과 관련된 멀티타스킹 동작을 수행할 수 있다. 프로그램들간의 콘텍스트 변경에 필요한 시간은 콘텍스트 변경되는 프로그램을 성공적으로 복구하는데 필요한 모든 프로세서 상태정보의 통상적인 저장에 비하여 관련된 저장된 프로세서 상태정보의 양을 감소시킴으로써 줄일 수 있다. 콘텍스트가 변경되는 동안 저장된 프로세서 정보량을 감소시킴으로써 다음에 설명하는 바와 같이 자원을 보다 효율적으로 이용할 수 있다.

도 6은 멀티미디어 멀티프로세서 시스템(100)에 채용된 것으로 삼성 MSP계열의 멀티미디어신호 프로세서

로부터 선택된 멀티미디어신호 프로세서(200)를 이용하는 효율적인 콘텍스트의 변경 및 프로그램복구 과정의 흐름(600)(콘텍스트 저장/복구 과정의 흐름(600))의 일실시예를 나타낸 것이다. 콘텍스트 저장/복구 과정의 흐름(600)은 프로그램실행 블럭(602)에서 시작된다. 프로그램 실행블럭(602)에 있어서, 코프로세서(204)(도 2 참조)는 응용프로그램과 같은 프로그램을 실행한다. 이 응용프로그램은 응용프로그램 전반에 걸쳐 프로그래머에 의해 군데군데 배치된 표시를 포함함으로써 이 표시가 대략 규칙적으로 나타날 수 있도록 한다. 이 실시예에 있어서, 상기 표시는 VCCS라고 하는 코프로세서(204)의 조건적인 콘텍스트 변경명령이다. 응용프로그램이 실행되는 동안 어느 한 지점에서 코프로세서(204)는 현재 실행중인 프로그램내의 상기 지점과 관련된 프로세서 상태정보를 포함한다. 응용프로그램내의 몇몇 지점들에서 다른 지점들에서보다 콘텍스트변경후 응용프로그램을 성공적으로 복구하는데 필요한 프로세서 상태정보가 적다. 콘텍스트 변경을 통해 최소량의 프로세서 상태정보의 보존을 필요로 하는 응용프로그램내의 이러한 "감소된 프로세서 상태" 지점를 인지하는 프로그래머는 VCCS명령을 응용프로그램내의 이들 감소된 프로세서 상태 지점들에 내린다. 그러나, 서로 마주친 VCCS명령들간의 간격은 콘텍스트 변경동작을 크게 지연시키지 않도록 크지 않은 것이 바람직하다. 이 간격은 프로그래머의 판단, 응용프로그램 및 코프로세서(204)의 클럭속도에 의해 결정된다. 일실시예에 의한 상기 간격은 요구된 콘텍스트 변경과 요구에 대한 응답 사이에 2μSEC이하의 지연을 발생시킨다.

콘텍스트 변경후에 프로그램을 성공정으로 재실행시키는데 필요한 프로세서 상태정보량을 감소시킴으로써 프로세서 상태정보의 잠재적인 크기가 매우 큰 경우에 특히 프로세서 동작의 이득을 크게 할 수 있다. 예를 들면, 삼성 MSP계열의 어떤 프로세서는 프로세서 상태정보에 할당된 매우 많은 양의 메모리 저장영역을 갖는 전형적인 구조이다. 삼성 MSP는 7킬로바이트이상의 프로세서 상태정보를 포함하고, 64개의 288비트 벡터 레지스터, 80개 이상의 32비트 스칼라 레지스터, 및 최고 4킬로바이트의 스크래치패드 메모리를 포함한다. 삼성 MSP와 같은 구조에 있어서의 콘텍스트 변경 및 프로그램 복구가 이루어지는 동안의 프로세서 상태정보의 전송량의 감소와 관련하여 성능의 향상이 이루어질 수 있다.

응용프로그램내의 감소된 프로세서 상태지점의 일예는 50개의 독립변수(argument)들의 합을 결정하는 ADD 명령과 관련이 있다. 응용프로그램이 독립변수들의 합을 결정하기 전에 콘텍스트 변경되면, 응용프로그램을 복구하는데 필요한 프로세서 상태정보는 50개 독립변수 모두를 반드시 가져야 한다. 그러나, ADD명령이 종료되고 상기 합이 결정되면, 상기 합만이 프로세서 상태정보의 일부분으로서 포함되어야 한다. 그러므로, 프로그램으로의 복귀상의 프로그램의 재실행을 위해 필요한 프로세서 상태정보의 양은 모든 가능한 프로세서 상태정보의 저장과 관련하여 감소된다. 코프로세서(204) 조건적 콘텍스트 변경명령인 VCCS명령은 상기 50개의 독립변수들의 합의 결정에 따른 지점에서 응용프로그램내에 위치한다. 이것은 성공적으로 콘텍스트를 변경하고 응용프로그램을 복구하기 위해 필요한 프로세서 상태정보를 감소시키기 위한 응용프로그램내의 적절한 지점의 일예이다. 응용프로그램내의 다른 많은 지점들이 콘텍스트 변경 및 이에 뒤따른 프로그램의 복구전에 프로세서 상태정보 저장량을 감소시키기 위해 선택될 수 있음은 당분야에서 통상의 지식을 가진에게 명백할 것이다.

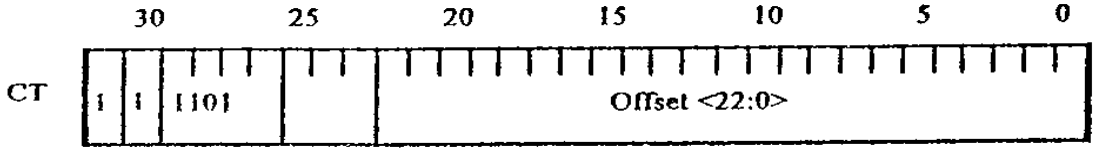
도 7을 참조하면, 응용프로그램의 부분(700)은 군데군데 위치한 VCCS명령들 (702)을 포함한다. VCCS명령들(702)은 응용프로그램내에서 성공적으로 정확하게 콘텍스트 변경이 일어나기 전에 최소량의 프로세서 상태정보저장을 요구하는 소정의 위치에 규칙적으로 군데군데 위치한다.

도 6을 참조하면, 프로그램 실행블럭(602)내에서 응용프로그램이 실행되는 동안 32비트 VCCS명령은 콘텍스트 변경 명령블록(604)에 나타낸 바와 같이 적절한 지점에서 만나게 될 것이다. 콘텍스트 변경 요구블럭(606)으로 진행하여 VCCS명령은 코프로세서(204)가 프로세서(202)가 현재 실행중인 응용프로그램이 콘텍스트 변경되어 다른 프로그램으로 대체되는 것을 요구했는지 하지 않았는지를 결정하도록 한다. 콘텍스트 변경이 요구되지 않았으면, 콘텍스트 저장/복구 과정의 흐름(600)은 프로그램 실행블럭(602)로 되돌아가서 마주친 VCCS명령에 따른 지점에서부터 응용프로그램을 계속 실행한다.

콘텍스트 변경을 요구하기 위하여 프로세서(202)는 코프로세서 인터페이스(206)내에 위치하는 VIMSK레지스터라고 하는 32비트 벡터 프로세서 인터럽트 마스크 레지스터(208)로 쓰기동작을 행한다. 특히, 프로세서(202)는 CSE비트에 1을 씀으로써 VIMSK레지스터(214)의 CSE또는 콘텍스트 변경 인에이블비트인 비트를 0로 설정한다. VIMSK레지스터에 대한 보다 상세한 내용은 미국특허출원 일련번호 xx/xxx,xxx (변리사 사건번호M-4366)의 Song등의 "System and Method for Handling Software Interrupts with Argument Passing"과 일련번호 xx/xxx,xxx (M-4367)의 Song등의 "System and Method for Handling Interrupt and Exception Events in an Asymmetric Multiprocessor Architecture"에 기술되어 있다.

표 1에 삼성 MSP 벡터 프로세서에서 사용되는 것과 같은 VCCS명령포맷을 나타내었다

[표 1]



VCCS포맷은 최상위비트 위치내의 111101 비트들을 포함하고, 최하위 23비트들내의 오프셋 필드(offset field)를 포함한다. 상기 오프셋필드는 콘텍스트 변경이 요구되었을때 코프로세서(204)가 VCCS명령의 성공적인 종료로 분기되는 콘텍스트 저장 서브루틴 위치를 인지한다. 비트들(25:23)은 VCCS명령에 의해 사용되지 않는다. VCCS명령의 어셈블러 신택스(assembler syntax)는 VCCS#Offset이다.

콘텍스트 변경요구 블럭(606)을 참조하면, 프로그램 실행블럭(602)내에서 현재 실행중인 응용프로그램이

콘텍스트 변경되어 다른 프로그램으로 교체될 것을 프로세서(202)가 요구했는지를 결정하기 위해 상기 마주친 VCCS명령에 의해 코프로세서(204)가 VIMSK레지스터(214)를 읽게 된다. VCCS명령이 예외처리가 일어나도록 하지 않는다고 가정한다면, VIMSK레지스터(214)의 CSE비트가 1로 설정되면 프로그램 실행블록(602)내의 응용프로그램의 복귀어드레스는 콘텍스트 변경되는 프로그램과 유일하게 관련된 소프트 복귀어드레스 스택에 저장된다. 응용프로그램의 현재 어드레스는 벡터 프로그램카운터(VPC)내에 위치한다. 복귀어드레스는 VPC+4이다.

코프로세서(204)가 VCCS명령을 마주칠때까지 프로세서(202)로부터 요구된 콘텍스트 변경을 점검하지 않는다는데 주목해야 한다. 결과적으로, 프로그래머는 실행중인 프로그램이 콘텍스트 변경될 수 있는 지점을 제어할 수 있다. 또한, 요구된 콘텍스트 변경과 이에 따른 응답사이에 지연이 발생한다 하더라도 콘텍스트의 변경요구, 검출 및 응답지연이 일어나는 동안 응용프로그램이 계속 실행되므로 프로그램 실행에 있어 문제가 발생하지 않는다.

다음의 의사코드(pseudocode)는 VCCS의 작용을 나타내는 것이다.

```
If(VIMSK 〈 〈CSE〉 ==1){
    if(VSP 〈4〉 〉 15){
        VISRC 〈RASO〉 =1;signal processor202 with
        RASO exception;
        VP_STATE=VP_IDLE;
   } else {
        RSTACK[VSP 〈3:0〉 ]=VPC+4;
        VSP 〈4:0〉 =VSP 〈4:0〉 +1;
        VPC=VPC+sex(Offset 〈22:0〉 *4);
  }
} else VPC = VPC+4;
```

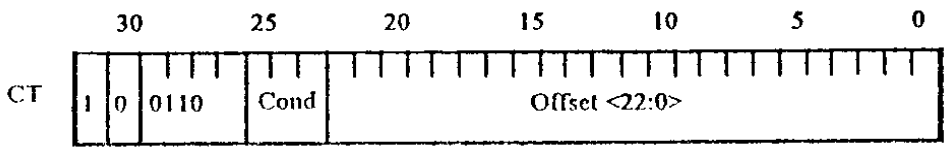
콘텍스트 변경 요구블럭(606)을 참조하면, VCCS명령에 의해 코프로세서(204)가 VIMSK레지스터(214)의 CSE비트가 설정되었는지를 결정한다. 비트가 설정되지 않았으면, 프로세서(202)가 콘텍스트 변경을 요구하지 않은 것이고, 코프로세서 (204) 프로그램 카운터(VPC)는 다음 명령까지 증가하며 프로그램 실행블럭(602)으로의 복귀가 이루어진다. 프로세서(202)가 콘텍스트 변경을 요구했다면, 코프로세서(204)는 응용프로그램과 관련된 5비트 벡터 스택포인터(VSP)의 비트 4개를 검사하고, VCCS명령의 실행이 스택 오버플로우를 초래할 것인지를 결정한다. 스택 오버플로우가 발생할 것이라면 벡터 인터럽트 발생원 레지스터(VISRC)의 복귀어드레스 스택오버플로우 예외 비트는 1로 설정되고, 코프로세서(204)의 상태(VP_STATE)는 아이들상태(VP-IDLE)내에 위치하게 된다. 스택 오버플로우가 일어나지 않을 것이라면, 콘텍스트 저장/복구 과정의 흐름(600)이 진행되어 복귀어드레스 블럭(608)을 저장하고, 코프로세서(204)는 프로그램 실행블럭(602)내에서 실행되는 응용프로그램의 복귀어드레스를 저장하고 소프트 스택포인터를 증가시킨다. 그런 다음 VPC는 VCCS 오프셋필드에 의해 표시되는 부호확장된 어드레스로 로드된다. 이제 VPC에 저장된 어드레스는 콘텍스트 저장 서브루틴 블럭(612)에 나타낸 서브루틴을 저장하는 콘텍스트의 어드레스이다.

그런 다음 코프로세서(204)는 콘텍스트 저장 서브루틴 블럭(610)으로의 실행 브랜치에 나타난 바와 같이 VPC내의 콘텍스트 저장 서브루틴 어드레스를 이용하여 콘텍스트 저장 서브루틴으로 분기된다. 콘텍스트 저장 서브루틴(612)은 프로그램 실행블럭(602)내에서 실행되는 응용프로그램의 프로그래머에 의해 씌여진다. 결과적으로, 프로그래머는 콘텍스트 변경된 응용프로그램의 성공정인 재실행을 위해 저장될 필요가 있는 프로세서 상태정보의 최소량을 인지하게 된다. 최소 프로세서 상태정보 서브블럭(614)에 나타난 바와 같이 콘텍스트 저장 서브루틴(612)은 예컨대 콘텍스트 변경된 프로그램을 복구하기 위해 필요하게 될 레지스터와 스트래치패드 메모리에 존재하는 필요한 정보만을 저장한다.

콘텍스트 복구 서브루틴 서브블럭(616)의 저장위치전에 코프로세서(204)는 VCJOIN이라고 하는 프로세서(202) 타스크명령의 32비트 조건적 결합(Conditional Join)을 실행함으로써 콘텍스트 저장 서브루틴(612)을 종료한다.

표 2는 삼성 MSP 벡터프로세서에서 사용되는 것과 같은 VCJOIN명령 포맷을 나타낸 것이다.

[표 2]



VCJOIN포맷은 최상위비트 위치내에 100110 비트들을 포함하고, 최하위 23비트내에 비트들(25:23)내의 조건 필드(condition field) 및 오프셋필드를 포함한다. 상기 조건필드는 무조건적으로 프로세서(202)를 인터럽트하기 위해 설정된다. 오프셋필드는 콘텍스트 변경 프로그램의 재실행에 따라 코프로세서(204)가 실

행될 코텍스트 복구 서브루틴 위치의 어드레스를 나타낸다. VCJOIN명령의 어셈블러 신택스는 VCJOIN.cond#Offset이다.

다음 의사코드는 VCJOIN의 작용을 나타낸 것이다.

```
If(Cond=un){
    VISRC〈VJP〉==1;
    VIINS=[VCJOIN.cond#Offset instruction];
    VEPC=VPC;
        if(VIMSK〈〈VJE〉=1;signal processor 202
        interrupt;
        VP_STATE=VP_IDLE;
}
else VPC=VPE+4;
```

콘텍스트 복구 서브루틴 서브블럭(616)의 저장위치를 참조하면, 코프로세서(204)에 의한 VCJOIN.un#Offset 명령의 실행에 의해 콘텍스트 복구 서브루틴(628)의 위치가 저장되고 프로세서(202)를 인터럽트함으로써 프로세서(202)가 연속되는 프로그램의 실행을 위해 코프로세서(204)를 구동할 수 있게 된다. 콘텍스트 복구서브루틴 서브블럭(616)의 저장위치에 있어서, VCJOIN의 조건은 무조건으로 설정되며, VIMSK레지스터(214)의 VCJOIN인터럽트 인에이블(VJE) 비트는 1로 설정된다. VIMSK레지스터(214)의 VJE비트는 비트 5이다. 콘텍스트 복구 서브루틴 서브블럭(616)의 저장위치를 참조하면, VCJOIN명령의 23비트 오프셋필드는 콘텍스트 복구서브루틴(628)의 위치를 나타내기 위해 설정된다.

VCJOIN명령이 인터럽트 프로세서 서브블럭(618)내에서 코프로세서(204)에 의해 실행될때 코프로세서(204)는 VCJOIN명령의 조건 필드를 점검하여 조건 필드의 무조건적 상태를 확증한다. 계속해서, 코프로세서(204)는 32비트의 벡터 인터럽트 발생원 레지스터(VISRC)의 예외중인 비트(VJP)를 1로 설정한다. VJP비트는 VISRC레지스터의 비트 5이다. 벡터 인터럽트 명령레지스터 1(VIINS)는 VCJOIN명령이 실행되어 프로세서(202)를 인터럽트하므로 VCJOIN명령으로 업데이트된다. 현재의 VPC는 벡터 예외 프로그램카운터(VEPC)에 저장된다. VEPC는 가장 최근의 예외를 초래한 명령의 어드레스를 규정한다. VCJOIN명령의 실행에 계속됨에 따라 VIMSK레지스터(214)의 VJE비트는 평가되고 1로 설정되도록 결정된다. VJE비트의 설정에 의해 프로세서(202)가 IRQ인터럽트되고, VJP비트의 설정에 의해 코프로세서(204)는 IDLE 상태로 들어가게 된다.

콘텍스트 저장/복구 과정의 흐름(600)은 콘텍스트 저장 서브루틴 블럭(612)에서 나와 프로세서 인터럽트 처리기 블럭(620)으로 진행된다. 프로세서(202)는 미국특허출원 일련번호 xx/xxx,xxx (변리사 사건번호 M-4366)의 Song등의 "System and Method for Handling Software Interrupt with Argument Passing"와 일련번호 xx/xxx,xxx(M-4367)의 Song등의 "System and Method for Handling Interrupt and Exception Events in an Asymmetric Multiprocessor Architecture"에 기술된 바와 같이 인터럽트를 처리한다.

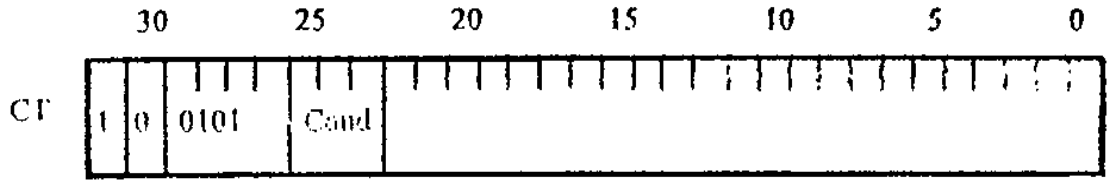
VCJOIN개시된 인터럽트를 처리한 후, 다음 프로그램 콘텍스트 변경 결정블럭(622)에 나타난 바와 같이 프로세서(202)는 실행될 다음 프로그램이 "새로운"프로그램, 즉, 전에 콘텍스크 변경이 일어나지 않거나 변경되지 않은 프로그램인지를 결정한다. 멀티미디어신호 프로세서(200) 구동시스템(도 5를 참조)는 프로그램 트랙을 유지하며 이 프로그램이 사전에 콘텍스트 변경되었는지 아닌지를 결정한다. 코프로세서(204)에 의해 실행될 다음 프로그램이 "새로운" 프로그램이면, 콘텍스트 저장/복구 과정(600)의 흐름은 새로운 프로그램블럭)(624)을 실행하기 위해 코프로세서를 구동하도록 진행된다. 프로세서(202)는 코프로세서(204)를 구동시켜 "새로운" 프로그램을 실행시키고, 콘텍스트 저장/복구 과정의 흐름(600)은 "새로운"프로그램이 실행되는 프로그램 실행블럭(602)으로 되돌아간다.

다음 프로그램 콘텍스트 변경 결정블럭(622)을 참조하면, 코프로세서(204)는 다음 프로그램이 이미 콘테스트 변경되었을 경우 콘텍스트 복구 서브루틴 블럭(626)으로의 브랜치로 들어간다. 그러면 프로세서(202)는 코프로세서(204)의 VPC를 상술한 바와 같이 콘텍스트 복구 서브루틴 서브블럭(616)의 저장위치내에서의 VCJOIN명령이 실행되는 동안 저장되었던 콘텍스트 복구 서브루틴(628)의 어드레스로 로드하고, 콘텍스트 저장/복구 과정의 흐름(600)은 콘텍스트 복구 서브루틴(628)으로 분기된다.

콘텍스트 복구 서브루틴(628)은 현재 콘텍스트 변경되는 응용프로그램의 프로그래머에 의해 씌여진다. 결과적으로, 프로그래머는 성공적인 콘텍스트 변경의 재실행전에 콘텍스트 저장 서브루틴(612)에 의해 사전에 저장되었고 현재는 응용프로그램내에서 콘텍스트 변경된 최소량의 프로세서 상태정보의 위치를 알게 된다. 콘텍스트 저장/복구 과정의 흐름(600)은 적절한 코프로세서(204)의 메모리위치, 예컨대 상술한 레지스터 및 스크래치 메모리내에 이미 저장된 최소량의 프로세서 상태정보를 코프로세서(204)가 읽는 최소 프로세서 상태정보 서브블럭(630)을 로드하기 위해 진행된다.

성공적인 콘텍스트 변경의 재실행을 위한 프로세서 상태정보를 로드함에 있어서, 콘텍스트 복구 서브루틴(628)은 코프로세서(204)를 준비시켜 콘텍스트 복구서브루틴 서브블럭(616)의 저장위치내에 이미 저장되어 있는 정확한 프로그램 위치에서 응용프로그램내에서의 콘텍스트 변경이 실행되도록 한다. 이 응용프로그램의 위치를 복구하기 위해 코프로세서(204)는 콘텍스트 복구 서브루틴(628)을 종료하는 서브루틴으로부터의 조건적 복귀(VCRSR)명령을 실행한다. 표 3은 삼성 MSP 벡터프로세서에서 사용되는 것과 같은 VCJOIN명령 포맷을 나타낸 것인다.

[표 3]



VCRSR포맷은 최상위 비트위치내에 100101비트들을 포함하며, 비트들(25:23)내에 조건 필드를 포함한다. 최하위 23비트는 사용되지 않으며 어떠한 값으로도 설정될 수 있다. 상기 조건 필드는 무조건적으로 프로세서(202)를 인터럽트하도록 설정된다. VCRSR명령의 어셈블리 신택스는 VCRSR.cond이다.

다음 의사코드는 VCRSR의 작용을 나타낸 것이다.

```
If(cond=un){
    if(VSP 〈4:0〉 ==0){
        VISRC 〈RASU〉 =1;
        signal processor 202 with RASU exception;
       VP_STATE=VP_IDLE;
    }else{
        VSP 〈4:0〉 =VSP 〈4:0〉 -1;
        VPC=RSTACK[VSP 〈3:0〉 ];
        VPC 〈1:0〉 =b'00;
    }
} else VPC=VPC+4;
```

프로그램 블럭(632)내의 콘텍스트 변경의 로드 복귀 어드레스를 참조하면, VCRSR.un 명령의 실행에 의해 코프로세서(204)가 저장 복귀 어드레스블럭(608)내에 이미 저장되어 있는 위치부터 프로그램내의 상기 콘텍스트 변경의 실행을 다시 시작한다. VCRSR.un명령이 실행되는 동안 코프로세서(204)는 무조건적 브랜치 코드를 포함하는 VCRSR명령의 조건 필드를 조사함으로써 무조건적 브랜치가 요구되었는지 결정한다. VCRSR명령의 실행으로 인해 조사된 VSP는 콘텍스트 변경중인 프로그램과 유일하게 관련된 소프트 스택 포인터이다. 이어서 코프로세서(204)는 VSP를 조사하여 VSP가 최하위 비트위치를 가르키고 있는지 아닌지를 결정한다. VSP가 최하위 위치를 가리키고 있으며, VISRC레지스터의 RASU비트는 1로 설정된다. RASU비트는 복귀 어드레스 스택 언더플로우(Return Address Stack Underflow)예외비트이며, 상기와 같이 설정되면 프로세서(202)로 예외가 존재한다는 신호를 보낸다. 이때, 코프로세서(204)는 아이들상태로 들어가게 된다. VSP가 최하위 위치를 가리키고 있지 않으면, 코프로세서(204)는 VSP를 1씩 감소시키고 이 선택된 VSP위치에 저장된 어드레스로 VPC를 로드한다. 이 VSP위치는 저장 복귀 어드레스블럭(608)에 이미 저장되어 있는 복귀 어드레스를 포함한다. VPC 〈1:0〉 =b'00 단계는 VPC의 최하위 2비트가 0로 로드되는 것을 확실하게 하는 것이다.

복귀어드레스가 VPC내에 로드된 후, 콘텍스트 변경 과정의 흐름(600)은 프로그램내의 콘텍스트 변경의 실행되기 시작한 프로그램 실행블럭(602)으로 되돌아간다. 콘텍스트 변경 과정의 흐름(600)은 프로그램 실행이 종료될때가지 반복된다.

발명의 효과

상술한 바와 같이 멀티타스킹 멀티프로세서 환경에서의 콘텍스트 변경의 효율성은 콘텍스트 변경 과정의 흐름(600)에 의해 향상된다. 프로그램의 콘텍스트 변경이 일어나는 동안 프로세서 상태정보의 최소량을 저장하는 것만으로 코프로세서(204)의 시간을 다른 동작에 이용할 수 있다. 또한, 최소량의 프로세서 상태정보만을 복구함으로써 프로그램이 효율적으로 콘텍스트 변경되도록 할 수 있다. 이러한 시간 절약은 콘텍스트 변경동작이 수행되는 동안 축적된다.

본 발명을 특정의 바람직한 실시예에 관련하여 도시하고 설명하였지만, 본 발명이 상기 실시예에 한정되는 것은 아니다. 예를 들면, 특정한 하드웨어 및 소프트웨어의 실시예들은 예시적인 것이며, 다른 많은 시스템구조 및/또는 소프트웨어 실시예들에 의해 상술한 콘텍스트 변경 및 복구 특징을 실현할 수 있다. 또한, 콘텍스트 변경 및 복구가 2개 이상의 타스크과 관련된 멀티타스킹 환경에서 사용될 수 있음은 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 명백할 것이다. 따라서, 이하의 특허청구의 범위에 의해 마련되는 본 발명의 정신이나 분야를 이탈하지 않는 한도내에서 본 발명이 다양하게 개조 및 변화될 수 있다는 것을 당업계에서 통상의 지식을 가진 자는 용이하게 알 수 있다.

(57) 청구의 범위

청구항 1

제 1 프로세서상에서 제 1 프로그램을 실행하는 단계와; 제 2 프로세서로부터 콘텍스트 변경요구를 받아들이는 단계; 및 상기 제 1 프로그램내의 적절한 지점에서 상기 콘텍스트 변경요구에 응답하는 단계를 포함하며, 상기 적절한 지점은 상기 제1프로그램내에서 복수의 표시들에 의해 지시되는 복수의 미리설정된 지점들중의 하나이며, 상기 복수의 미리설정된 지점들은 상기 제 1 프로그램내의 근사지점에 위치된 선택 설정지점들을 가지며, 상기 근사지점은 상기 제1프로세서가 상기 제1프로그램의 성공적인 복구를 위해 최소량의 프로세서 상태정보의 저장을 요구하는 지점이 되도록 설정되는 것임을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 2

제1항에 있어서, 제 1 프로그램의 성공적인 복구를 위해 최소량의 프로세서 상태정보의 저장을 요구하는 위치들에서 상기 제 1 프로그램내에 복수개의 표시들을 산재시키는 단계가 더 포함되는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 3

제1항에 있어서, 상기 적절한 지점에서 상기 제 1 프로그램의 상태에 상응하는 프로세서 상태 정보를 저장하는 단계와; 상기 제 1 프로그램을 바꾸는 단계; 상기 제 1 프로세서상에서 제 2 프로그램을 실행하는 단계 ; 상기 제 2 프로그램을 바꾸는 단계; 및 최소량의 저장된 프로세스 정보를 이용하여 상기 제 1 프로그램을 복구하는 단계가 더 포함되는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 4

제1항에 있어서, 상기 콘텍스트 변경요구에 응답하는 단계가, 상기 적절한 지점에서 상기 제 1 프로그램의 상태에 상응하는 프로세서 상태 정보를 저장하는 단계와; 상기 제 1 프로그램을 정확하게 복구하기 위해 콘텍스트 복구 루틴의 위치를 저장하는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 5

제1항에 있어서, 상기 제 1 프로그램을 실행하는 단계가, 멀티미디어 멀티프로세서 시스템내의 멀티미디어신호 프로세서의 벡터프로세서상에서 상기 제 1 프로그램을 실행하는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 6

제1항에 있어서, 복수개의 멀티미디어 장치들로부터 데이타를 제공받는 단계가 더 포함되며, 상기 제 1 프로그램을 실행하는 단계가 상기 제공된 데이타를 처리하는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 7

제1항에 있어서, 상기 콘텍스트 변경요구를 받아들이는 단계가 상기 제 2 프로세서에 의해 설정되는 콘텍스트 변경 인에이블 필드를 갖춘 레지스터를 읽는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 8

제1항에 있어서, 상기 콘텍스트 변경요구를 받아들인 후에 상기 제 1 프로그램을 계속해서 실행하는 단계가 더 포함되며, 상기 콘텍스트 변경요구에 응답하는 단계가, 상기 적절한 지점에서 상기 제 1 프로그램내의 표시와 만나는 단계와; 상기 제 1 프로세서를 인터럽트하는 단계; 상기 제 1 프로그램내의 표시를 만남에 따라 상기 제 1 프로세서로 상기 콘텍스트 변경요구를 읽어내는 단계; 상기 제 1 프로그램의 복귀 어드레스를 저장하는 단계; 상기 제 1 프로그램의 성공적인 복구를 위해 필요한 최소량의 프로세서 상태정보를 저장하는 단계; 및 상기 제 1 프로세서가 이용가능하다는 것을 제 2 프로세서에게 통지하여 제 2 프로그램이 실행되도록 하는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 9

제1항에 있어서, 상기 표시는 조건적인 콘텍스트 변경명령임을 특징으로 하는 멀티타스킹 처리시스템 환경에서의 효율적인 콘텍스트 저장 및 복구방법.

청구항 10

각기 산재한 복수개의 콘텍스트 변경 표시들을 갖춘 프로그램들을 수행하기 위한 멀티타스킹 환경내의 제 1 프로세서와; 상기 제 1 프로세서에 연결되고, 프로세서 상태정보를 저장하도록 할당된 제 1 메모리; 상기 제 1 프로세서에 연결된 제 2 메모리; 상기 프로세서가 실행중인 프로그램내의 표시들중의 어느 하나를 만난 후에 프로그램의 콘텍스트 변경 요구를 검출하기 위한 상기 제 1 프로세서상에서 동작하는 콘텍스트 변경요구 검출기; 및 상기 제 1 메모리내에 위치한 프로세서 상태정보를 상기 제 2 메모리에 저장함으로써 검출된 콘텍스트 변경요구에 응답하기 위한 제 1 프로세서상에서 동작하는 콘텍스트 저장 모듈을 포함하는 것을 특징으로 하는 컴퓨팅시스템.

청구항 11

제10항에 있어서, 상기 콘텍스트 저장모듈이 콘텍스트 복구모듈의 위치를 저장함으로써 검출된 콘텍스트

변경요구에 응답하는 것을 특징으로 하는 컴퓨팅시스템.

청구항 12

제10항에 있어서, 상기 제 1 프로세서와 상호연결된 제 2 프로세서가 더 포함되는 바, 상기 제 1 및 제 2 프로세서는 비대칭적인 특성을 가지며, 상기 제 2 프로세서는 제 1 프로세서가 실행중인 프로그램을 콘텍스트 변경하도록 요구하도록 하는 메카니즘을 갖는 것을 특징으로 하는 컴퓨팅시스템.

청구항 13

제12항에 있어서, 상기 제 1 프로세서와 제 2 프로세서사이에 연결되며, 제 1 및 제 2 프로세서에 의해 상호 억세스가능한 레지스터를 포함하는 인터페이스 유니트가 더 포함되는 바, 상기 제 2 프로세서는 콘텍스트 변경요구를 나타내도록 상기 레지스터에 쓰기동작을 행할 수 있으며, 상기 제 1 프로세서는 상기 제 2 프로세서로부터 콘텍스트 변경요구를 검출하도록 상기 레지스터를 읽을 수 있는 것을 특징으로 하는 컴퓨팅시스템.

청구항 14

제12항에 있어서, 상기 제 2 프로세서는 제어프로세서이고, 제 1 프로세서는 벡터프로세서임을 특징으로 하는 컴퓨팅시스템.

청구항 15

제10항에 있어서, 콘텍스트 변경된 프로그램과 관련된 상기 제2메모리내에 위치하는 상기 저장된 프로세서 상태정보를 상기 제 1 메모리로 복귀시킴으로서 콘텍스트 변경된 프로그램을 복구하는 상기 제 1 프로세서상에서 동작하는 콘텍스트 복구모듈이 더 포함되는 것을 특징으로 하는 컴퓨팅시스템.

청구항 16

제10항에 있어서, 상기 표시들은 상기 프로그램 전반에 걸쳐 성공적으로 프로그램을 복구하기 위해 최소 프로세서 상태정보를 필요로 하는 위치에 군데군데 위치하는 것을 특징으로 하는 컴퓨팅시스템.

청구항 17

제10항에 있어서, 상기 표시들은 조건적인 콘텍스트 변경명령들임을 특징으로 하는 컴퓨팅시스템.

청구항 18

프로그램에 조건적인 콘텍스트 변경명령들을 삽입하는 단계와; 상기 프로그램을 실행하는 단계; 콘텍스트 변경요구를 받는 단계; 상기 조건적인 콘텍스트 변경명령들중의 하나를 검출하는 단계; 상기 조건적인 콘텍스트 변경명령들중의 하나를 검출하는 단계후에 상기 콘텍스트 변경요구가 프로세서에 존재하는지를 결정하는 단계; 및 콘텍스트 변경요구의 존재의 결정에 따라 프로그램의 복귀어드레스를 저장하거나 프로그램을 계속 실행하는 단계와, 상기 조건적인 콘텍스트 변경명령을 검출하기 전에 프로그램의 상태에 해당하는 프로세서 상태정보를 저장하는 단계와 콘텍스트 복구 모듈의 위치를 저장하는 단계 및 상기 프로세서를 인터럽트하는 단계로 이루어진 콘텍스트 저장모듈을 실행하는 단계를 구비한 프로그램을 바꾸는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 멀티프로세서 처리시스템 환경에서의 효율적인 콘텍스트 저장방법.

청구항 19

제18항에 있어서, 상기 콘텍스트 저장모듈 실행단계에서 이미 저장된 프로세서 상태정보를 로딩하는 단계와, 상기 프로그램을 바꾸는 단계에서 이미 저장된 프로그램의 복귀를 로드하는 단계로 이루어진 콘텍스트 복구모듈을 실행하는 단계와; 상기 프로그램을 실행하는 단계가 더 포함되는 것을 특징으로 하는 멀티타스킹 멀티프로세서 처리시스템 환경에서의 효율적인 콘텍스트 저장방법.

청구항 20

제18항에 있어서, 코프로세서상에서 상기 프로그램을 실행하는 단계와; 제어프로세서로부터 인터페이스 유니트로 상기 콘텍스트 변경요구를 제공하는 단계가 더 포함되는 것을 특징으로 하는 멀티타스킹 멀티프로세서 처리시스템 환경에서의 효율적인 콘텍스트 저장방법.

청구항 21

제18항에 있어서, 상기 프로그램에 조건적인 콘텍스트 변경명령들을 삽입하는 단계가 프로그램 전반에 걸쳐 대략 규칙적으로 조건적인 콘텍스트 변경명령들을 삽입함으로써 이 조건적인 콘텍스트 변경명령들이 프로그램내의 콘텍스트 변경전에 최소량의 프로세서 상태정보 저장을 필요로 하는 프로그램내의 위치들에 삽입되도록 하는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 멀티프로세서 처리시스템 환경에서의 효율적인 콘텍스트 저장방법.

청구항 22

제21항에 있어서, 상기 삽입단계가 조건적인 콘텍스트 변경명령들은 대략 규칙적인 간격으로 삽입함으로써 상기 콘텍스트 변경요구를 받는 단계와 조건적인 콘텍스트 변경명령들중의 하나를 검출하는 단계사이에 두드러진 지연이 발생하는 것을 방지하는 단계를 더 포함하는 것을 특징으로 하는 멀티타스킹 멀티프로세서 처리시스템 환경에서의 효율적인 콘텍스트 저장방법.

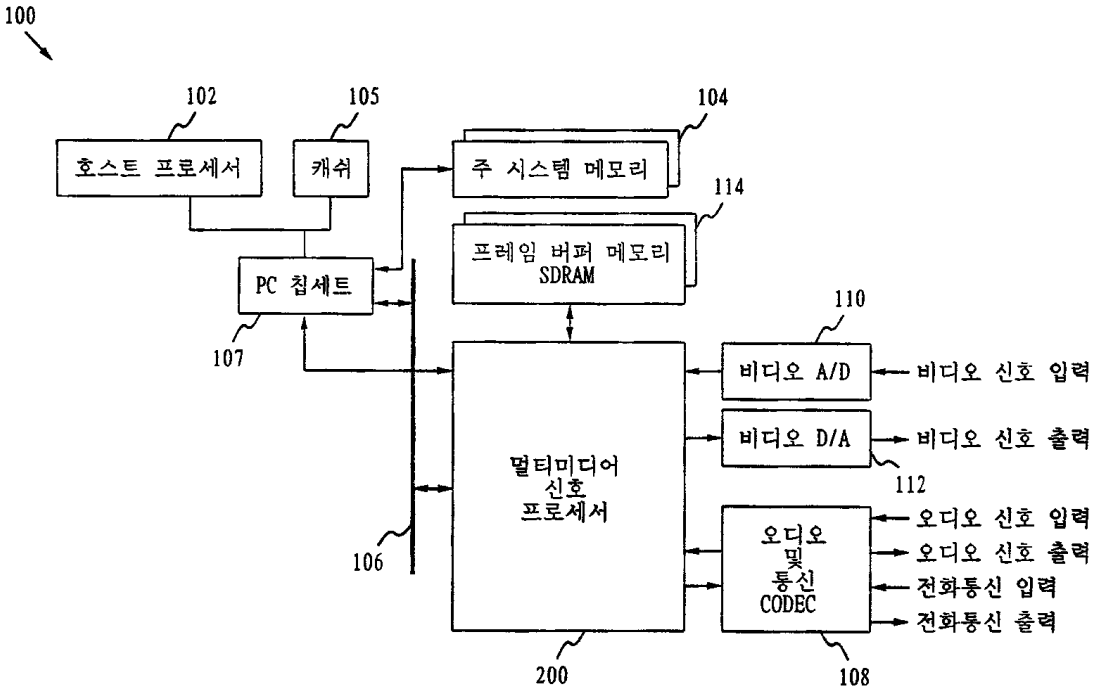
청구항 23

제18항에 있어서, 상기 조건적인 콘텍스트 변경명령들중의 하나를 검출함으로써 상기 제1프로세서를 대략

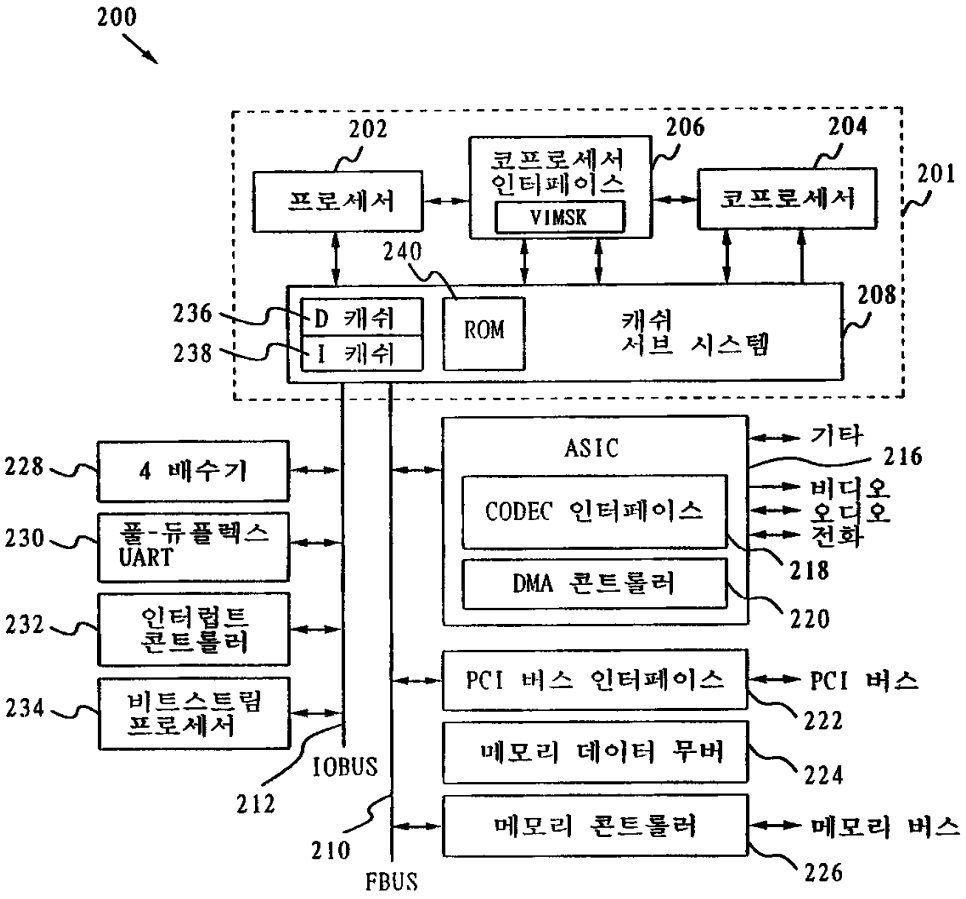
규칙적으로 인터럽트하는 단계가 더 포함되며, 상기 프로그램을 실행하는 단계가 제1프로세서상에서 상기 프로그램을 실행하는 단계를 포함하는 것을 특징으로 하는 멀티타스킹 멀티프로세서 처리시스템 환경에서의 효율적인 콘텍스트 저장방법.

도면

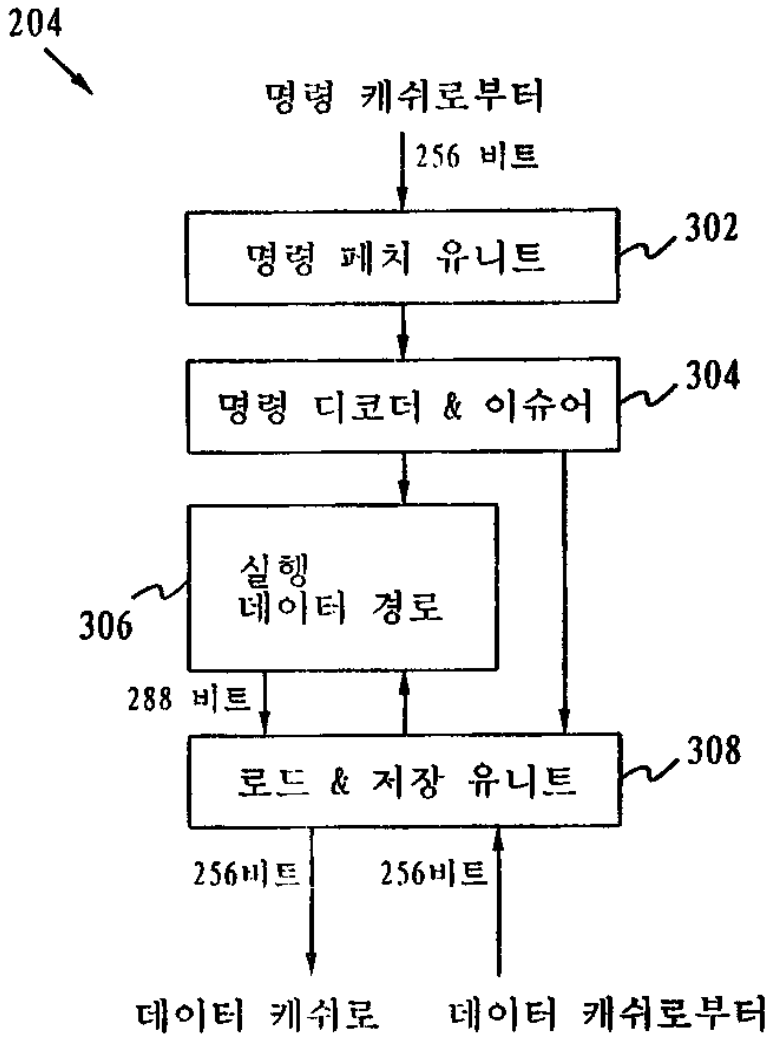
도면1



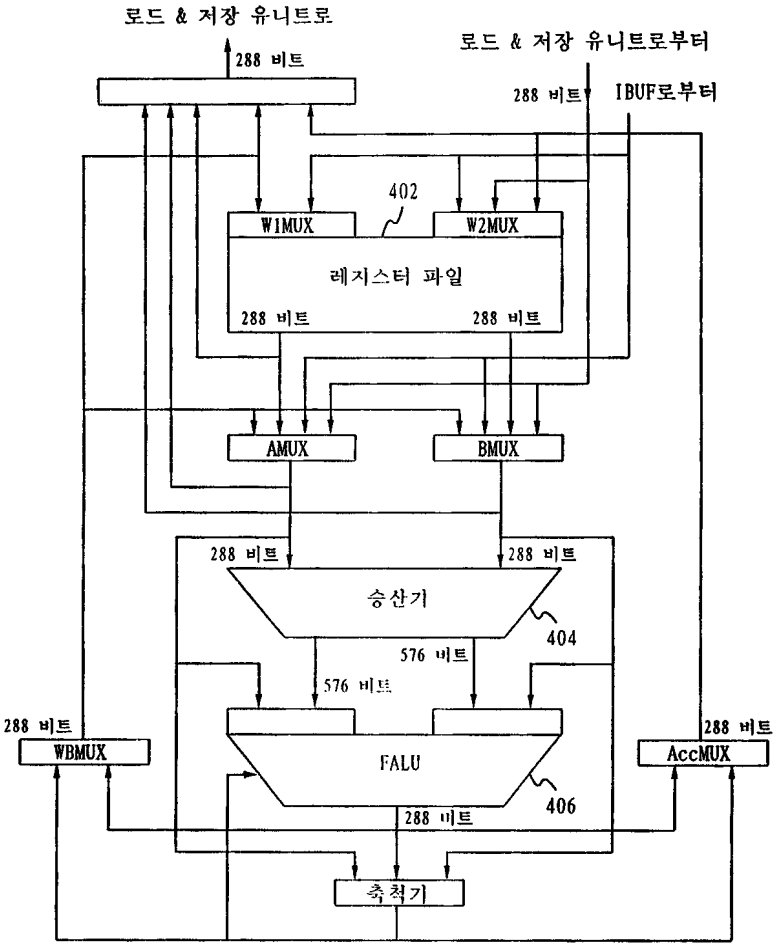
도면2


200
202
프로세서
206
코프로세서 인터페이스
VIMSK
204
코프로세서
201
240
236
D 캐쉬
238
I 캐쉬
ROM
캐쉬 서브 시스템
208
228
4 배수기
230
풀-듀플렉스 UART
232
인터럽트 콘트롤러
234
비트스트림 프로세서
IOBUS
212
210
FBUS
ASIC
기타
216
CODEC 인터페이스
비디오
오디오
전화
218
DMA 콘트롤러
220
PCI 버스 인터페이스
PCI 버스
222
메모리 데이터 무버
224
메모리 콘트롤러
메모리 버스
226

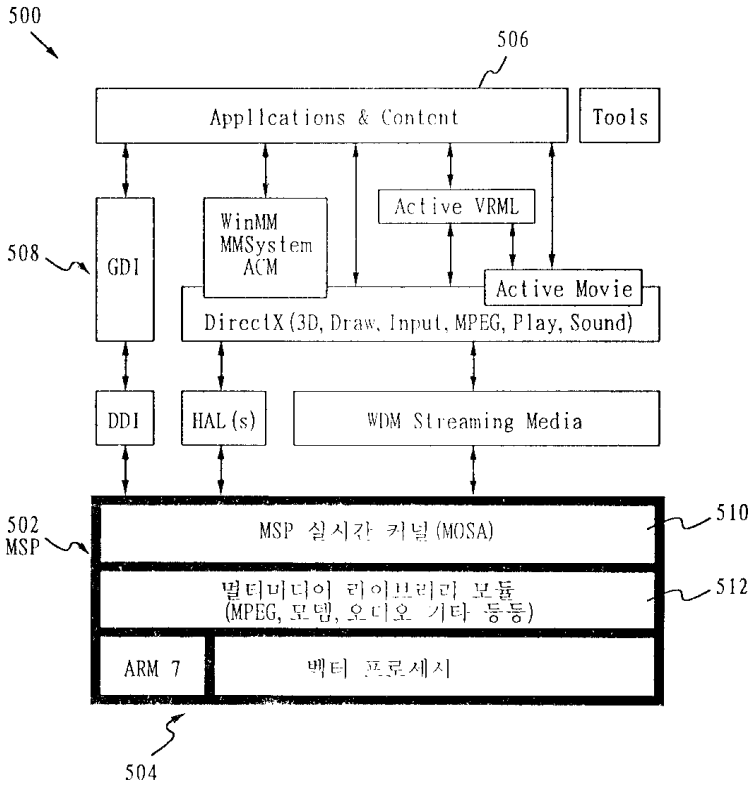
도면3


204
명령 캐쉬로부터
256 비트
명령 페치 유니트
302
명령 디코더 & 이슈어
304
실행
데이터 경로
306
288 비트
로드 & 저장 유니트
308
256비트
256비트
데이터 캐쉬로
데이터 캐쉬로부터

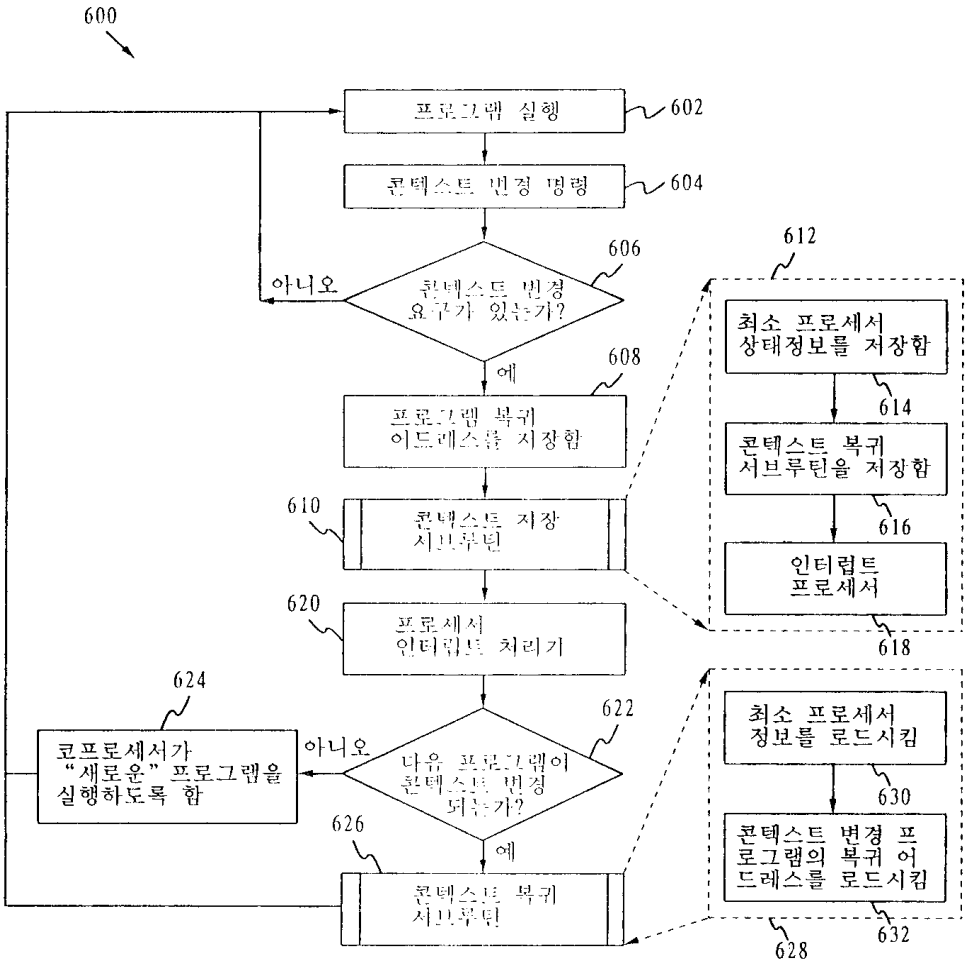
도면4


로드 & 저장 유니트로
288 비트
로드 & 저장 유니트로부터
288 비트
IBUF로부터
402
W1MUX
W2MUX
레지스터 파일
288 비트
288 비트
AMUX
BMUX
288 비트
288 비트
승산기
404
576 비트
576 비트
288 비트
WBMUX
FALU
288 비트
AccMUX
406
288 비트
축적기

도면5


500
506
Applications & Content
Tools
508
GDI
WinMM
MMSystem
ACM
Active VRML
Active Movie
DirectX(3D, Draw, Input, MPEG, Play, Sound)
DDI
HAL(s)
WDM Streaming Media
502
MSP
MSP 실시간 커널(MOSA)
510
멀티미디어 라이브러리 모듈
(MPEG, 모뎀, 오디오 기타 등등)
512
ARM 7
벡터 프로세서
504

도면6


600
프로그램 실행
602
콘텍스트 변경 명령
604
606
콘텍스트 변경 요구가 있는가?
아니오
예
608
프로그램 복귀 어드레스를 저장함
610
콘텍스트 저장 서브루틴
620
프로세서 인터럽트 처리기
622
다음 프로그램이 콘텍스트 변경 되는가?
624
코프로세서가 "새로운" 프로그램을 실행하도록 함
아니오
626
예
콘텍스트 복귀 서브루틴
612
최소 프로세서 상태정보를 저장함
614
콘텍스트 복귀 서브루틴을 저장함
616
인터럽트 프로세서
618
최소 프로세서 정보를 로드시킴
630
콘텍스트 변경 프로그램의 복귀 어드레스를 로드시킴
632
628

도면7