(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2014년07월15일
(11) 등록번호 10-1419668
(24) 등록일자 2014년07월08일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2006.01) *G06F 9/318* (2006.01)
G06F 9/32 (2006.01) *G06F 9/45* (2006.01)
(21) 출원번호 10-2013-7023899(분할)
(22) 출원일자(국제) 2007년09월06일
심사청구일자 2013년09월10일
(85) 번역문제출일자 2013년09월10일
(65) 공개번호 10-2013-0109251
(43) 공개일자 2013년10월07일
(62) 원출원 특허 10-2013-7015566
원출원일자(국제) 2007년09월06일
심사청구일자 2013년06월17일
(86) 국제출원번호 PCT/NL2007/050434
(87) 국제공개번호 WO 2008/030093
국제공개일자 2008년03월13일
(30) 우선권주장
06120236.2 2006년09월06일
유럽특허청(EPO)(EP)
(56) 선행기술조사문헌
KR100160602 B1
KR1020000006302 A
전체 청구항 수 : 총 15 항

(73) 특허권자
실리콘 하이브 비.브이.
네덜란드 5656 에이쥐 에인트호벤 하이 테크 캠퍼스 83
(72) 발명자
레이흐텐, 제로엔, 안톤, 요한
네덜란드 엔엘-5096 비씨 허셀 부레인드 18
즈와르텐코트, 헨드리크, 트에르트, 요아네스
네덜란드 엔엘-5653 피에이 에인트호벤 아케렌담 7
(74) 대리인
김윤배

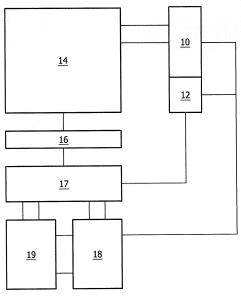
심사관 : 지정훈

(54) 발명의 명칭 **데이터 처리회로 및 데이터 처리방법**

(57) 요 약

데이터 처리회로는 복수의 기능적 유닛(20)을 구비하는 실행회로(18)를 가진다. 명령어 디코더(17)는 제1 및 제2 명령어 모드에서 작동가능하다. 제1 명령어 모드에서는 명령어들이 각 기능적 유닛(20)을 제어하기 위한 각각의 필드를 가지며, 제2 명령어 모드에서는 명령어들이 하나의 기능적 유닛을 제어한다. 모드제어회로(12)는 명령어 모드들의 선택을 제어한다. 일 실시예에서, 명령어 디코더는 실행회로(18)에 의해 실행될 연산들의 선택과 레지스터 세트(19)로부터 목적지 레지스터들의 선택의 시간 정적 디코딩을 사용한다. 모드 절환은 명령어들이 명령을 포함하는 기능적 유닛을 표시하는 것보다 시간 정적 프로세서들을 위한 명령어 시간을 감소시키는 더욱 효율적인 방법이다. 명령어들이 예약되면, 바람직하게는 연산 선택 및 목적지 레지스터 선택이 명령어 모드변경의 다른 쪽들에 포함되는 연산들에 제약들이 부여된다. 이러한 연산들이 실시예에서는 방지된다. 다른 실시예에서는, 연산 선택 및 목적지 레지스터 선택이 명령어 모드변경의 다른 쪽들에 포함되는 선택된 연산들이 예약된다. 점프를 포함하는 명령어들에 직면하면, 모드 제어회로는 점프 명령의 실행에 의해 제공된 정보와 일치하여 후속의 명령어들을 위한 명령어 모드를 설정한다.

대 표 도 - 도1



특허청구의 범위

청구항 1

한 세트의 레지스터(19);와,

이 한 세트의 레지스터에 결합된 실행회로(18);와,

실행회로(18)에 결합되고, 실행회로(18)에 의해 실행될 연산들의 선택을 위한 명령어들과 그리고 실행유닛(18)에 의해 결과들이 생성되면 연산들의 결과를 쓰기 위한 레지스터(19) 세트로부터의 목적지 레지스터들의 선택을 위한 명령어들의 시간 정적 디코딩을 사용하도록 구성되고, 적어도 제1 및 제2 명령어 모드에서 동작가능한 명령어 디코더(17)로서, 제1 및 제2 명령어 모드에서의 실행을 위한 명령어들은 서로 상이한 개수의 어드레싱 가능한 목적지 레지스터들에 대한 액세스를 가지는, 명령어 디코더(17);와,

명령어 디코더(17)에 결합되고, 프로그램이 실행되는 동안 명령어들에 대응하여 명령어 모드들 사이의 절환을 제어하도록 된 모드제어회로(12);를 구비하는, 데이터 처리회로.

청구항 2

청구항 1에 있어서,

제1 및 제2 명령어 모드 사이의 절환을 수행하는 절환 명령어들을 구비하는 프로그램으로 프로그램된 명령어 메모리(14)를 포함하고,

각 절환 명령어는 절환 명령어가 수행되기 전과 후 각각 실행된 프로그램의 제1 부분 및 제2 부분 사이의 경계를 규정하며,

프로그램의 상기 제1 부분은 그 결과가 프로그램의 대응하는 상기 제2 부분으로부터 명령어에 의해 선택된 목적지 레지스터에 기입되는 연산을 선택하는 명령어를 포함하지 않는, 데이터 처리회로.

청구항 3

청구항 1에 있어서,

제1 명령어 모드에서의 실행을 위한 제1 명령어로서, 연산의 선택을 위한 제1 필드를 포함하는 제1 명령어와, 제2 명령어 모드에서의 실행을 위한 제2 명령어로서, 그 결과가 생성될 때 제1 명령어에 의해 선택된 연산의 결과를 기입하기 위한 목적지 레지스터를 선택하기 위한 제2 필드를 포함하는 제2 명령어를 구비하는 프로그램으로 프로그램된 내부 메모리(14)를 포함하는, 데이터 처리회로.

청구항 4

청구항 3에 있어서,

상기 모드제어회로는 점프명령의 실행에 의해 제공된 명령어 모드 선택 정보에 대응하여 프로그램으로부터 점프명령에 따라 실행된 명령어들을 위한 명령어 모드들 사이의 선택을 제어하도록 구성되고, 제1 및 제2 명령어는 각각 점프명령이 취해지는 프로그램의 일지점 전 과 후에 위치되는, 데이터 처리회로.

청구항 5

청구항 4에 있어서,

상기 프로그램의 점프명령의 타겟은 프로그램 내에서 점프명령을 통해서만 도달될 수 있는, 데이터 처리회로.

청구항 6

청구항 1에 있어서,

실행유닛은 복수의 기능적 유닛(20)을 구비하고, 명령어 디코더(17)는 결과들에 생성되면 제1 명령어 모드에서의 실행을 위한 각 명령어로부터 각각의 기능적 유닛으로부터의 결과들을 기입하기 위한 목적지 레지스터 어드레스를 선택하기 위한 제1 필드들을 추출하도록 구성되고, 명령어 디코더(17)는 결과 또는 결과들이 생성되면

각각의 기능적 유닛들의 서브세트만으로부터의 결과 또는 결과들을 기입하기 위한 목적지 레지스터 어드레스 또는 목적지 레지스터 어드레스들을 선택하기 위한 하나의 제3 필드 또는 제3 필드들을 제2 명령어 모드에서의 실행을 위한 각 명령어로부터 추출하도록 구성되는, 데이터 처리회로.

청구항 7

청구항 1에 있어서,

명령어 디코더(17)는 각각 제1 및 제2 명령어 모드의 명령어들로부터 목적지 레지스터 어드레스들을 선택하기 위한 제1 및 제2 필드들을 추출하도록 구성되고, 제1 및 제2 필드는 제1 및 제2 범위의 어드레싱 가능한 레지스터들을 각각 제공하고, 제1 범위는 제2 범위에 포함되지 않은 레지스터들을 포함하는, 데이터 처리회로.

청구항 8

청구항 1에 있어서,

명령어 메모리(14)를 더 구비하는, 데이터 처리회로.

청구항 9

청구항 8에 있어서,

명령어 메모리(14)는 하나 이상의 캐시 레벨들을 포함하는 계층적 메모리 회로로 구현되는, 데이터 처리회로.

청구항 10

청구항 1에 있어서,

영상압축을 위한 블록 정합 연산들, 영상 압축 및 압축해제를 위한 DCT 변환 명령들, 컴퓨터 그래픽 처리를 위한 텍스쳐 맵핑 또는 비트 블리팅(blitting) 연산들, 및 가변길이 압축/압축해제 연산들로부터 선택된 하나 이상의 타입의 연산들을 실행하기 위한 전용 하드웨어를 더 구비하는, 데이터 처리회로.

청구항 11

청구항 1에 있어서,

상이한 명령어 모드들은 영상 압축, 영상 압축해제, 오디오 압축, 오디오 압축해제, 및 컴퓨터 그래픽 처리 중의 하나 이상을 위한 상이한 세트의 특화된 연산들로부터의 선택을 제공하는, 데이터 처리회로.

청구항 12

실행회로(18)에 의해 실행될 연산들의 선택을 위한 명령어들과 그리고 실행유닛(18)에 의해 결과들이 생성되면 연산들의 결과를 쓰기 위한 레지스터(19) 세트로부터의 목적지 레지스터들의 선택을 위한 명령어들의 시간 정적 디코딩을 사용하는 단계, 및

프로그램이 실행되는 동안 명령어들에 대응하여 적어도 제1 및 제2 명령어 모드 사이의 절환을 제어하는 단계를 포함하고,

제1 및 제2 명령어 모드에서의 실행을 위한 명령어들은 서로 상이한 개수의 어드레싱 가능한 목적지 레지스터들에 대한 액세스를 가지는, 데이터 처리방법.

청구항 13

청구항 12에 있어서,

제1 및 제2 명령어 모드 사이의 절환을 수행하는 절환 명령어들을 구비하는 프로그램을 디코딩하는 단계를 더 포함하고,

각 절환 명령어는 절환 명령어가 수행되기 전과 후 각각 실행된 프로그램의 제1 부분 및 제2 부분 사이의 경계를 규정하며,

프로그램의 상기 제1 부분은 그 결과가 프로그램의 대응하는 상기 제2 부분으로부터 명령어에 의해 선택된 목적

지 레지스터에 기입되는 연산을 선택하는 명령어를 포함하지 않는, 데이터 처리방법.

청구항 14

청구항 12에 있어서,

제1 명령어 모드에서의 실행을 위한 제1 명령어로서, 연산의 선택을 위한 제1 필드를 포함하는 제1 명령어와, 제2 명령어 모드에서의 실행을 위한 제2 명령어로서, 그 결과가 생성될 때 제1 명령어에 의해 선택된 연산의 결과를 기입하기 위한 목적지 레지스터를 선택하기 위한 제2 필드를 포함하는 제2 명령어를 구비하는 프로그램을 디코딩하는 단계를 더 포함하는, 데이터 처리방법.

청구항 15

청구항 13에 있어서,

명령어 모드들 사이의 절환을 제어하는 것은 점프명령의 실행에 의해 제공된 명령어 모드 선택 정보에 대응하여 프로그램으로부터 점프 명령에 따라 실행된 명령어들을 위한 명령어 모드들 사이의 선택을 제어하는 것을 포함하고, 제1 및 제2 명령어는 각각 점프명령이 취해지는 프로그램의 일지점 전 과 후에 위치되는, 데이터 처리방법.

명 세 서

기 술 분 야

[0001] 본 발명은 데이터 처리회로 및 데이터 처리방법에 관한 것이다.

배 경 기 술

[0002] 미국특허출원 제2002042909호는 VLIW 데이터 프로세서의 복수의 기능적 유닛을 위하여 상이한 명령어 세트들을 사용하여 프로그램의 여러 부분들을 수행하는 재목적 컴파일러(retargetable compiler)의 사용을 개시하고 있다. 이 프로그램은 연산을 위한 오퍼랜드(operand)들이 로딩되어야 하는 소스 레지스터들과 그리고 연산의 결과들이 저장되어야 할 목적지 레지스터의 가산, 시프트 연산 등과 같이, 실행될 연산을 특정하는 기계 명령어들로 컴파일된다.

[0003] 상이한 명령어 세트들이 사용되는데, 이 명령어 세트들은 기능적인 유닛들에 대하여 다양한 제어도를 제공한다. 제1 명령어 세트는 명령어들로 하여금 복수의 기능적 유닛에 의해 독립적으로 병렬 연산을 제어하게끔 한다. 제2 명령어 세트는 오직 하나의 기능적 유닛의 연산만이 각 명령어에 의해 제어되게끔 한다. 제1 명령어 세트와 제2 명령어 세트를 비교하자면, 제1 명령어 세트는 태스크(task)가 제1 명령어 모드를 위한 명령어들에 코딩되는 경우 태스크의 실행에 일반적으로 시간이 덜 걸리는 장점을 가지며 제2 명령어 모드는 테스크를 수행하는 프로그램을 저장하는데 일반적으로 메모리를 덜 필요로 한다는 장점을 갖는다.

[0004] 컴파일러는 상이한 명령어 세트들에 대응하는 하드웨어 명령들과 프로그램을 구비한다. 컴파일러는 예컨대 내부 루프들을 위한 제1 명령어 세트들과 내부 루프 외부의 프로그램 부분들을 위한 제2 명령어 세트를 사용하여 상이한 프로그램 부분들을 위한 명령어 세트들을 선택한다. 다음에 컴파일러는 프로그램 부분들을 선택된 명령 세트들로부터 기계 명령어들의 프로그램들로 컴파일한다.

[0005] 프로그램을 실행하는데 사용되는 데이터 처리회로는 복수의 각각의 기능적 유닛들을 위한 상이한 명령어 세트들에 대응하여, 상이한 명령어 모드들 사이에서 절환가능하도록 구성된다. 명령어 모드에서 작동하는 경우, 데이터 처리회로는 현재의 명령어 모드에 대응하는 명령어 세트로부터의 명령어들로서 데이터 처리회로에 공급되는 명령어들을 해석한다. 미국 특허출원 제2002042909호는 명령어 모드들 사이의 절환이 어떻게 수행되는지에 대해 관여하지 않고 다만 이를 위하여 전용의 재구성 명령어들을 사용하는 미국특허 제5,933,642호를 참조하고 있다.

[0006] 프로그램을 실행하는 이 방법은 실행속도와 프로그램 크기의 조합의 최적화를 제한한다. 게다가 특별한 재구성 명령어들의 사용에 의해 명령 세트가 확장된다.

발명의 내용

해결하려는 과제

[0007] 본 발명은 데이터 처리회로를 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0008] 본 발명의 일태양에 있어서, 그 목적은 특별한 명령어들을 사용하지 않고 명령어 모드 절환을 제공하는 것이다.

[0009] 다른 태양에서는, 실행속도와 프로그램 크기의 조합의 최적화 개선을 제공하는 것이 목적이다.

[0010] 또 다른 목적은 실행속도와 프로그램 크기의 조합의 최적화를 개선시킬 수 있도록 하는 데이터 처리회로를 제공하는 것이다.

[0011] 제1 태양에 따르면, 청구항1에 따른 데이터 처리회로가 제공된다. 명령어 내의 점프명령들이 상이한 명령어 모드들 사이의 절환을 달성하기 위하여 사용되는데, 명령어들은 더 많은 그리고 더 적은 기능적 유닛들을 제어한다. 따라서 이를 위해 전용의 모드절환 명령어가 필요하지 않다. 모드에 관한 정보는 점프 타겟 어드레스에 제공될 수 있는데, 예를 들어 그 어드레스의 다수의 최상위 비트로서 제공될 수 있다. 점프 타겟 어드레스는 점프 명령어에서 문자로 특정될 수 있다. 다른 실시예에서는, 점프 타겟 어드레스가 점프 명령의 오퍼랜드 레지스터로부터 제공될 수 있고, 이 오퍼랜드 레지스터로부터의 다수의 최상위비트들이 후속의 명령어들을 위한 모드를 선택하는데 사용된다.

[0012] 추가 실시예에서는, 상이한 모드의 명령어들은 서로 다른 길이를 가진다. 이는 명령어 어드레스의 상이하게 크기설정된 부분들은 상이한 명령어 모드들의 명령어 어드레스들의 최소 명령어 증가 보다 낮은 변환(resolution)을 제공하는 것을 의미한다. 일 실시예에서, 프로그램 카운터 값들은 모드에 의존하는 시프트 연산에 의해 예컨대 4 바이트의 명령어 모드에서 2 비트만큼 또는 2바이트 명령어 모드에서 1비트만큼 명령어 메모리 어드레스들로 번역된다. 따라서 모드에 의존하는 프로그램 카운터 증가가 사용될 수 있고 어떠한 명령어 어드레스 비트들도 여분의 어드레스 변환으로 잃지 않는다. 다른 실시예에서는, 명령어 모드에 의존하는 증가분들이 사용된다. 또 다른 실시예에서는, 메모리로부터의 새로운 인출이 필용한지 또는 새로 어드레싱된 명령어가 더 큰 인출 워드에서 초기 명령어와 함께 이미 인출되었는지를 더 짧은 명령어들을 가진 명령어 모드의 각 프로그램 카운터 증가분 이후에 판별하는데 비교기가 사용될 수 있다.

[0013] 다른 태양에서, 상이한 명령모드들은 연산 선택과 결과 레지스터 어드레스 선택의 시간 정적 디코딩을 사용하는 데이터 프로세서에 대해 사용된다. 시간 정적 명령어 인코딩과 디코딩은 이미 공지되어 있다. 각 명령어가 연산의 선택과 이 연산을 위한 결과 레지스터 어드레스의 선택 모두를 포함하는 더 일상적인 데이터 정적 인코딩 및 디코딩과 대비될 수 있다. 이는 연산이 결과를 생성할 때까지 결과 레지스터 어드레스가 예컨대 파이프라인에 유지되어야 하는 결과를 가진다. 시간 정적 명령어 인코딩 및 디코딩의 경우에는, 이와 대조적으로 연산의 선택과 이 연산을 위한 결과 레지스터의 어드레싱이 연산이 개시되는 시간에 그리고 데이터 프로세서가 연산의 결과를 생성한 시간에 각각 실행되는 상이한 명령어들에 포함된다. 따라서 각 명령어는 연산 선택 및 결과 레지스터 어드레싱을 위한 필드들을 포함하지만, 상이한 연산들에 대하여 시간 정적 명령어 인코딩에 포함한다. 결과 레지스터 어드레싱을 위한 필드는 초기 실행명령어에 의해 선택된 연산의 결과로서, 명령어를 실행할 때 기능적 유닛으로부터 이용할 수 있는 결과를 위해 사용된다.

[0014] 시간 정적 명령어 디코딩을 사용하는 데이터 프로세서에 있어서, 상이한 명령어 모드들의 사용은 프로그램 크기를 압축하는 다른 방법들보다 더 효율적이라는 장점을 가진다. 데이터 정적 VLIW 프로세서들을 위해 사용되는 종래의 압축 방법들은 명령들이 명령어에 포함되는 기능적 유닛들의 지시를 사용한다. 지시들의 갯수는 시간 정적 프로세서에서 상당히 증가할 수 있다. 상이한 명령어 모드들 사이의 절환의 사용은 이 오버헤드를 감소시키는데, 그 이유는 어드레싱될 수 있는 레지스터들의 갯수가 각 명령어에서 지시될 필요가 없기 때문이다. 따라서 예를 들면 복수의 기능적 유닛들로부터 결과들을 위한 결과 레지스터 어드레스들이 복수의 기능적 유닛들을 위하여 제1 명령어 모드의 각 명령어에 포함될 수 있고, 제2 명령어 모드에서는 하나의 결과 레지스터 어드레스만이 포함될 수 있다.

[0015] 일 실시예에서, 모드 절환 명령들은 대기시간을 가질 수 있는데, 이는 예를 들어 점프 명령어들이 이 목적을 위해 사용되는 경우이다. 이 경우에, 모드 절환 명령의 실행개시와 실제의 모드절환 사이에 지연이 있다. 이 지연은 모드절환명령들 뒤쪽의 추가 결과 레지스터 어드레스들을 가진 명령어들을 추가하는데 사용될 수 있다. 일 실시예에서는, 연산들이 모드 절환 전에 너무 늦게 선택되어 데이터 프로세서가 모드 절환 후 그 결과들을 생성하는 것이 방지된다. 따라서 그 결과 또는 결과들은 예컨대 같은 수의 연산들과 결과 레지스터 선택들을 제공하

는 명령어들을 사용하여, 연산을 선택하는데 사용되었던 것과 같은 명령어 모드에 있는 명령어의 제어 하에서 기입될 수 있다. 이는 프로그램 시퀀싱을 간단히 한다. 다른 실시예에서 상이한 명령어 모드들은 연산이 선택되는 제1 명령어와 레지스터가 그 결과를 기입하기 위해 어드레싱되는 제2 명령어를 위해 사용된다. 이는 데이터 프로세서를 더 효율적으로 사용할 수 있게 하는데, 명령어 모드들 사이의 절환들 근방의 예약 연산들에 대한 제약을 줄여주기 때문이다. 예를 들어 점프 명령어가 명령어 모드들 사이에서의 절환에 사용되고, 점프의 타겟은 점프를 매개로 도달할 수 있을 때에 적용될 수 있다. 이 경우에 어떠한 모호함도 결과의 소스에 존재하지 않는다. 그러나, 더 복잡한 실시예에서는, 상이한 경로들이 타겟에 존재할 때, 연산들이 그 각 경로에서 결과를 생성하기 위해 선택될 때 적용될 수도 있다. 다른 실시예에서, 프로그램 내의 상이한 지점들로부터 호출될 수 있는 서브루틴(일련의 명령어들의 서브프로그램)의 결과를 기입하는데 사용될 수 있다. 따라서 서브루틴으로부터의 모든 복귀를 위한 어떠한 소정의 레지스터도 결과들을 위하여 보존될 필요가 없다.

[0016] 다른 태양에서는, 이런 타입의 데이터 프로세서를 위한 명령어들을 예약하기 위한 예약 방법이 제공된다. 단일 모드 프로세서들과 데이터 정적 프로세서들의 예약은 이미 공지되어 있다. 전형적으로는, 프로그램이 고급언어 프로그램으로부터 프로세서의 기계 연산들로 먼저 컴파일된다. 예약하는 동안, 연산들의 (병렬) 실행을 위한 시퀀스가 선택되어, 프로세서 효율을 최대화한다. 이어서 연산들을 실행하는 명령어들은 예약의 결과에 따라 명령어들에 위치된 연산 선택 코드들을 가진 명령어 메모리로 프로그램된다 (프로그래밍은 요구되는 명령어들이 되는 마스크 세트로 제조하는 것, 또는 비휘발성 메모리 또는 심지어는 휘발성 메모리에 프로그래밍하는 것을 예컨대 포함할 수 있다). 다수의 명령 모드들을 가진 시간 정적 프로세서의 경우에, 예약은 모드변경 명령들을 명령어들에 삽입하고 대응하는 연산들의 선택들을 포함하는 명령어들로의 오프셋들에서 명령어들에 결과 레지스터 어드레스들을 위치시키는 것을 또한 포함한다. 결과 레지스터 어드레스들이 결과들을 생성하는 연산들의 선택과 같은 명령어 모드에서 수송될 수 있는 것을 보장하기 위하여 예약의 제약들이 여러 실시예들에서 부여될 수 있거나, 이와 달리 결과 레지스터 어드레스들과 그리고 결과들을 생성하는 연산들의 선택이 상이한 모드들에서 실행되는 명령어들에 포함되는 것을 위해 연산들이 선택될 수 있다. 후자의 경우에, 예약되는 연산 선택들의 갯수는 명령어 모드 절환 후 목적지 레지스터들을 선택하기 위한 용량에 맞추어질 수 있다. 즉, 용량이 감소하면 더 적은 연산 선택들이 모드 절환에 근접하여 예약될 수 있고 용량이 증가하면 더 많이 예약될 수 있다.

도면의 간단한 설명

[0017] 상기 목적들과 그리고 다른 목적들 및 장점을 갖는 태양들은 하기 도면들을 참조하여 주어진 예시적 실시예들의 설명으로부터 명백해 질 것이다.

도1은 데이터 처리회로를 나타낸다.

도2와 도2a는 실행 유닛을 나타낸다.

도3은 명령어 디코더를 나타낸다.

도4와 도4a는 시퀀서와 모드 제어 회로를 나타낸다.

도5는 프로그래밍 시스템을 나타낸다.

도6은 컴파일 절차의 흐름도를 나타낸다.

발명을 실시하기 위한 구체적인 내용

[0018] 도1은 데이터 처리회로를 나타낸다. 이 회로는 시퀀서(10)와, 모드 제어회로(12)와, 명령어 메모리(14)와, 명령어 레지스터(16)와, 명령어 디코더(17)와, 실행 유닛(18)과 레지스터 화일 세트(즉, 하나 이상의 레지스터 화일)(19)을 포함한다. 시퀀서(10)는 명령어 메모리(14)의 입력에 결합된, 프로그램 카운터 어드레스를 제공하기 위한 출력을 가진다. 명령어 메모리(14)는 명령어 레지스터(16)에 결합된 출력을 가진다. 명령어 레지스터(16)는 명령어 디코더(17)에 결합된 출력을 가지는데, 이 명령어 디코더는 실행 유닛(18)과 레지스터 화일 세트(19)에 결합된 출력들을 가진다. 실행 유닛(18)은 레지스터 화일 세트(19)의 읽기 포트들과 쓰기 포트들에 결합된 결과 출력들과 오퍼랜드 입력들 및 시퀀서(10)에 결합된 점프 제어 출력을 가진다. 모드 제어회로(12)는 명령어 디코더(17)의 제어입력에 결합된 출력을 가진다.

[0019] 동작에 있어서, 시퀀서(10)는 프로그램 카운터 값을 유지하며, 보통 연속 명령어 싸이클에서 프로그램 카운터 값을 증가시키며 점프 명령어가 실행 유닛(18)에 의해 실행되는 경우 성공적인 점프 명령어에 따라 프로그램 카운터 값을 변경시킨다. 시퀀서(10)는 명령어 모드 선택을 제어하기 위하여 모드 제어회로(12)에 신호를 제공한

다.

[0020] 프로그램 카운터 값의 적어도 일부는 명령어 메모리(14)의 명령어들을 어드레싱하는데 사용된다. 명령어 메모리(14)는 어드레싱된 명령어들을 검색하여 명령어 레지스터(16)에 제공하는데, 명령어 레지스터에는 명령어들이 실행을 제어하기 위하여 저장되어 있다. 간단한 메모리가 명령어 메모리(14)를 구현하는데 사용될 수 있다 하더라도, 명령어 메모리(14)는 계층적 메모리 회로로 구현될 수도 있는데, 이 계층적 메모리 회로는 사용될 것으로 예상되는 명령어들을 저장하기 위하여 임의 갯수의 캐시 레벨들을 포함한다.

[0021] 명령어 레지스터(16)는 명령어 디코더(17)에 연속적으로 명령어들을 제공한다. 모드 제어회로(12)는 명령어 디코더(17)에 모드 제어신호를 제공한다. 명령어 디코더(17)는 실행 유닛(18)과 레지스터 화일 세트(19)를 위한 실행 제어신호를 생성한다.

[0022] 도2는 실행 유닛(18)의 실시예를 나타낸다. 본 실시예에서, 실행 유닛(18)은 각기 하나 이상의 레지스터 화일(미도시)의 읽기 포트에 결합된 오퍼랜드 입력(22)들을 가지고 적어도 하나의 기능 유닛을 구비하는 복수의 기능적 유닛(20) 군과, 기능적 유닛(20) 군에 결합된 입력들과 하나 이상의 레지스터 화일(미도시)의 쓰기 포트들에 결합된 출력들(26)을 가진 결과 레지스터(24)들을 구비하는 회로이다. 기능적 유닛(20)의 각각의 군은 명령어 디코더(미도시)에 결합된 제어 입력(28)을 갖는다. 추가 실시예에서는, 내부 레지스터(미도시)들이 기능적 유닛 군에 제공될 수 있는데, 이는 기능적 유닛의 결과들이 작동 코드의 제어하에서 다른 기능적 유닛을 위한 오퍼랜드로서 내부적으로 통과되도록 하기 위하여 실행회로들에 결합된다.

[0023] 도2a는 실행 유닛(18)의 실시예를 나타내는데, 멀티플렉서(29)들이 실행 유닛(18)의 내부 결과 레지스터(24)들(임의선택적으로 선택가능)과 레지스터 화일들(미도시)의 읽기 포트들로부터의 오퍼랜드들사이의 선택을 위해 제공된다. 멀티플렉서(29)들은 명령어 디코더로부터 제어된다. 이는 레지스터 화일에 중간 결과들을 저장하지 않고 연산의 연결이 가능하게 한다.

[0024] 일 실시예에서, 오퍼랜드 레지스터들은 레지스터 화일들 및/또는 결과 레지스터들로부터의 오퍼랜드의 임시 저장소를 위하여 기능적 유닛 군(20)들의 입력들에서 제공될 수도 있다. 이는 오퍼랜드의 로딩과 그리고 다른 명령 싸이클들에서 오퍼랜드들을 사용한 연산들의 수행을 실행할 수 있게 한다.

[0025] 일 실시예에서, 명령어 레지스터(16)로부터의 명령어는 복수의 필드를 포함하는데, 각각 하나 이상의 비트 위치를 구비한다. 필드들의 일부분의 내용들은 실행 유닛(18)에 의해 수행될 연산들을 나타내며 필드들의 다른 부분의 내용들은 레지스터 화일(19) 세트의 레지스터 어드레스를 나타내는데, 이로부터 오퍼랜드 데이터가 읽기 포트로 제공되거나 또는 쓰기 포트들로부터 여기로 결과들이 저장되어야 한다. 이 실시예에서는, 모드제어신호는 명령어의 어떤 부분들이 연산을 나타내는 필드로서 작용하고 명령어의 어떤 부분들이 레지스터 어드레스 및/또는 이 부분들의 크기를 나타내는 필드로서 작용하는지를 판별하는데 사용된다.

[0026] 도3은 명령어 디코더(17)의 실시예를 나타낸다. 본 실시예에서, 명령어 디코더(17)는 명령어 레지스터(16)에 결합된 멀티 비트 입력을 가진 명령어 선택회로(31)와 이 명령어 선택회로(31)의 출력중 다른 부분들에 결합된 입력을 가진 복수의 멀티플렉서를 구비한다. 그리고, 디폴트 신호제공 도선들이 멀티플렉서(30)들의 선택된 입력들에 결합될 수 있다. 멀티플렉서(30)들의 제1 부분은 기능적 유닛 군들(미도시)의 제어입력들에 결합된 제1 출력(32)들을 가진다. 멀티플렉서(30)들의 제2 부분은 레지스터 화일 세트(미도시)의 읽기 포트들의 읽기 어드레스 입력들에 결합된 제2 출력(34)들을 가진다. 멀티플렉서(30)들의 제3 부분은 레지스터 화일 세트(미도시)의 쓰기 포트들의 쓰기 어드레스 입력들에 결합된 제3 출력(36)들을 갖는다.

[0027] 명령어 선택회로(31)는 시퀀서(10)와 모드제어회로(12)에 결합된 제어 입력들을 가진다. 모드제어회로(12)에 의해 선택된 명령어 모드가 완전 명령어 레지스터보다 명령어들이 덜 점유하는 것을 나타내는 경우, 명령어 선택회로(31)는 어떤 것이 명령어를 제공할지로부터 명령어 레지스터내의 위치를 선택하기 위하여 시퀀서로부터 프로그램 카운터 값의 비트들을 사용한다. 예를 들면, 명령어 모드가 절반크기의 명령들이 사용되는 것을 나타내는 경우, 1 비트의 프로그램 카운터 값이 명령어 레지스터의 내용의 제1 및 제2 절반부가 제공되는지를 제어한다. 다른 예로서, 명령어 모드는 1/4 크기의 명령어들이 사용되는 것을 나타내는 경우, 프로그램 카운터 값의 비트들은 명령어 레지스터의 내용의 제1, 제2, 제3 또는 제4의 연속 1/4이 제공되는지를 제어한다.

[0028] 멀티플렉서(30)들은 모드제어회로(12)에 결합된 제어 입력들을 가진다. 모드제어회로(12)에 의해 선택된 명령어 모드에 따라서, 멀티플렉서(30)들은 명령어 내의 다른 위치들로부터 비트들을 제공하고 및/또는 제1, 제2 및 제3 출력들에 디폴트 비트들을 제공한다.

[0029] 일 실시예에서, 명령어 디코더(17)는 모드제어신호에 의해 모드제어회로(12)로부터 선택된 명령어 모드에 따라

서 명령어로부터 실행 유닛(18)과 레지스터 화일 세트(19)로 필드들의 비트들을 라우팅한다. 레지스터 어드레스들을 나타내는 필드들은 결과들을 생성하는 연산들을 나타내는 필드들로부터 분리하여 라우팅되는 점을 주목하여야 한다. 따라서, 결과를 기록하기 위한 쓰기 어드레스를 나타내는 현재의 명령어로부터의 필드는 이전의 명령어에 의해 표시되었던 연산의 실행결과를 기록하기 위한 레지스터를 어드레스하는데 사용된다. 이 이전의 명령어의 결과는 현재의 명령어가 실행되고 있는 경우, 즉 현재의 명령어에 의해 선택된 하나 이상의 연산들이 실행되고 있는 동안 결과 레지스터(24)에서 이용가능하다.

[0030] 따라서, 이 연산들의 쓰기 결과들을 위한 레지스터 어드레스들과 연산 선택 코드들은 상이한 명령 싸이클들에서 제공되는, 다른 명령들로부터 제공된다. 일 실시예에서, 현재의 명령어에 의해 표시된 연산들은 현재 명령어의 필드들의 제어하에서 어드레싱된 레지스터들로부터 인출되는 오퍼랜드들에 적용되지만, 이와 달리 연산들은 이전 명령어의 필드들의 제어하에서 어드레싱된 레지스터들로부터 인출되는 오퍼랜드들에 적용될 수 있다. 이 경우 이 연산들을 위하여 오퍼랜드들을 인출하기 위한 레지스터 어드레스들과 연산 선택코드들은 다른 명령어들로부터 그리고 다른 명령어 싸이클에서 제공되기도 한다.

[0031] 실행 유닛이 기능적 유닛들로부터 결과들을 수용할 수 있는 내부 레지스터들을 포함하는 추가 실시예에서, 연산 코드들은 기능적 유닛들에 의해 이러한 내부 레지스터들의 내용에 적용될 연산들을 나타낼 수도 있다. 따라서, 복합 연산의 연속 단계들을 수행하는 연산 선택코드들이 다른 명령어들로부터 그리고 다른 명령어 싸이클에서 제공될 수 있다.

[0032] 명령어 디코더(17)는 제1 명령어 모드를 지원할 수 있는데, 예를 들어 하나의 명령어 내의 복수의 필드는 실행 유닛(18)의 다른 기능적 유닛(미도시)들에 의해 병렬로 실행되어야 하는 복수의 연산을 나타낸다. 제1 명령어 모드에서도 역시, 명령어의 다른 복수의 필드들은 명령어들을 위한 오퍼랜드들을 제공하는 레지스터들의 어드레스들을 나타낼 수 있다. 명령어의 다른 복수의 필드들은 초기 명령어 또는 초기 명령들로부터 연산들의 결과를 기록하는 레지스터들의 어드레스들을 나타낼 수 있다. 제1 명령어 모드에 추가하여, 명령어 디코더(17)는 하나의 필드만이 하나의 연산(또는 연산들의 하나의 소정의 조합)을 선택하기 위해 정의되는 제2 명령어 모드를 지원할 수 있고, 2개의 필드만이 오퍼랜드를 인출하기 위한 레지스터들의 어드레스를 나타내기 위해 정의되고, 하나의 필드만이 결과들을 기록하기 위한 어드레스를 나타내기 위하여 정의된다. 그리고, 또는 이와 달리 다른 명령어 모드는 필드들이 하나 이상의 연산을 그러나 제1 모드에서보다 더 적은 연산을, 오퍼랜드들을 인출하기 위한 2개 이상의 레지스터 어드레스들 및 결과를 기록하기 위한 하나 이상의 레지스터 어드레스를 나타내는 것을 제공할 수 있다.

[0033] 또한, 명령어의 상이한 필드들의 비트들의 수는 다른 명령어 모드들에서 상이할 수 있다. 예를 들어, 하나의 명령어 모드에서는 레지스터 어드레스들을 선택하기 위한 필드들의 비트들의 수가 다른 명령어 모드의 레지스터 어드레스를 선택하기 위한 필드들의 비트들의 수보다 더 클 수 있다. 따라서 많은 수의 레지스터들이 다른 명령어 모드에서보다 상기 하나의 모드에서 어드레싱될 수 있다. 예를 들어, 상기 하나의 모드에서 어드레싱될 수 있는 어드레스들의 서브세트만이 다른 모드에서 어드레싱 가능할 수 있다. 다른 예에서는, 레지스터들의 상이한 서로소 집합(disjoint set)들이 상기 하나의 모드와 다른 모드에서 또는 부분적으로 겹치는 집합들에서 어드레싱 가능할 수 있다.

[0034] 유사하게, 하나의 명령어 모드에서는 연산을 선택하기 위한 필드들의 비트들의 수가 다른 명령어 모드에서 연산들을 선택하기 위한 필드들의 비트들의 수보다 더 클 수 있다. 따라서 많은 수의 연산이 다른 명령어 모드에서보다 상기 하나의 모드에서 선택될 수 있다. 예를 들어, 상기 하나의 모드에서 선택될 수 있는 연산들의 서브세트만이 다른 모드에서 선택가능할 수 있다. 다른 예에서는, 연산들의 상이한 서로소 집합들이 상기 하나의 모드와 다른 모드에서 또는 부분적으로는 겹치는 세트들에서 선택가능할 수 있다.

[0035] 명령어 레지스터(16)로부터의 모든 비트들이 연산들과 오퍼랜드 레지스터 어드레스들 및 결과 레지스터 어드레스들을 나타내는데 사용될 필요가 없다는 점을 이해하여야 한다. 명령어 레지스터(16)로부터의 비트들의 일부는 명령어 모드들의 적어도 일부에서 사용되지 않고 남겨질 수 있다. 따라서, 더 짧은 명령어들이 이 명령어 모드들에서 사용될 수 있다.

[0036] 도4는 시퀀서(10)와 모드제어회로(12)의 실시예를 나타낸다. 이 실시예에서는, 시퀀서(10)가 프로그램 카운터 레지스터(40)와, 증가기(incrementer; 42)와, 멀티플렉서(44)와 비교기(46)를 포함한다. 프로그램 카운터 레지스터(40)는 프로그램 메모리(미도시)의 어드레스 입력과 그리고 증가기(42)에 결합된 출력을 가진다. 증가기(42)는 모드제어회로(12)에 결합된 제어입력을 가진다. 멀티플렉서(44)는 증가기(42)의 출력에 그리고 실행유닛(미도시)의 점프 어드레스 출력에 그리고 시작 어드레스 소스(미도시)에 결합된 입력들을 가진다. 추가하여 또

는 이와 달리, 멀티플렉서(44)는 서브루틴들로부터 복귀하기 위하여 복귀 어드레스 소스(미도시)에 결합된 입력을 가질 수 있다. 멀티플렉서(44)는 실행유닛(미도시)에 결합된 제어입력을 가진다. 멀티플렉서(44)는 프로그램 카운터 레지스터(40)의 입력에 결합된 출력을 가진다. 비교기(46)는 프로그램 카운터 레지스터(40)와 멀티플렉서(44)의 출력들에 결합된 입력들과 그리고 명령메모리(미도시)의 인에이블 입력에 결합된 출력을 가진다.

[0037] 모드 제어회로(12)는 모드 레지스터(48)와 모드 멀티플렉서(49)를 포함한다. 모드 멀티플렉서(49)는 출력모드 레지스터(48)에 그리고 점프 어드레스 출력을 동반하는 실행유닛(미도시)의 모드 부분 출력에 그리고 시작 모드 소스(미도시)에 결합된 입력들을 가진다. 모드 멀티플렉서(49)는 실행유닛(미도시)에 결합된 제어입력을 가진다. 모드 멀티플렉서(44)는 모드 레지스터(48)의 입력에 결합된 출력을 가진다.

[0038] 작동에 있어서, 프로그램 카운터 레지스터(40)는 각 실행 싸이클의 멀티플렉서(44)로부터 어드레스를 로딩한다. 실행유닛은 어드레스의 소스를 제어한다. 처음에, 시작 어드레스가 로딩된다. 이어서, 이전의 프로그램 카운터 값을 증가시키는 결과가 보통 로딩되지만, 실행 유닛이 성공적인 분기 명령어를 신호로 내보낸다면 실행유닛으로부터 제공된 분기 타겟 어드레스가 로딩된다. 이 실시예에서, 프로그램 카운터의 증가는 선택된 모드에 좌우된다. 예를 들어, 전체 크기, 절반크기, 1/4 크기 명령어들을 구비한 명령어 모드들이 사용된다면, 프로그램 카운터 값은 전체 크기, 절반크기 및 1/4 크기 명령어 모드가 각각 표시되는 경우 4, 2, 또는 1만큼 증가된다. 바람직하게는 명령어 메모리가 전체 크기 명령어를 로딩할 수 있다. 이 경우에 프로그램 카운터 값의 최하위 비트의 비트들은 전체 크기 명령어들보다 작은 것들을 식별하는데, 이 비트들은 명령어 메모리를 어드레싱하는데 필요하지 않다. 비교기(46)는 임의선택사항으로서, 명령어 메모리를 어드레싱하는데 필요한 현재와 장래의 프로그램 카운터 값들의 부분들을 비교하여 이 2개가 다르다면 읽기 인에이블 신호를 보낸다.

[0039] 도4a는 (예컨대 1씩 증가하는 것과 같이) 고정된, 명령어 모드에 독립한 증가가 사용되는 다른 실시예를 나타낸다. 이 다른 실시예에서는, 명령어 메모리에 적용하기 전에 모드에 의존하는 양만큼 프로그램 카운터 값을 시프트시키기 위해 시프트 회로(400)가 제공된다. 비교기(46)는 현재 모드에서 명령어 메모리를 어드레싱하는데 사용되는 어드레스들의 부분들만을 비교하기 위하여 대응하여 수정된다. 이 다른 실시예는 최하위비트들이 어떤 모드들에서 소정의 값들을 가질 필요가 있기 때문에 어떠한 최하위비트들도 쓸모없지 않다는 의미에서 프로그램 카운터 비트들을 더 효율적으로 사용하게 한다는 점을 이해하여야 한다.

[0040] 모드제어회로(12)의 동작은 시퀀서(10)의 동작과 유사하다. 모드값이 각 명령어 싸이클에서 모드 레지스터(48)에 로딩된다. 실행 유닛은 어드레스의 소스를 제어한다. 처음에, 시작 모드가 로딩된다. 이어서, 모드값은 통상 바뀌지 않지만 (예컨대 모드 레지스터(48)로부터 로딩됨), 실행유닛이 성공적인 분기 명령의 신호를 보내면 실행유닛으로부터 제공된 점프 타겟 어드레스를 수반하는 모드값이 로딩된다. 바람직하게는, 점프 타겟 모드값들은 예를 들어 명령의 일부를 형성하는 문자 데이터로서, 점프 명령들의 (절대 어드레스 또는 상대 어드레스의 형태인) 점프 타겟 어드레스들과 조합하여 특정된다.

[0041] 일 실시예에서, 점프 타겟 어드레스의 소정 개수의 최상위 비트들의 위치들이 이를 위하여 사용될 수 있어서, 점프 타겟 어드레스의 최상위 비트들은 모드 비트들이다. 절대 어드레스들을 구비한 점프 명령어들에 있어서, 이 최상위비트들은 점프 명령어가 실행되면 모드 레지스터(48)에 복사된다. 회로가 상대 어드레스(즉, 프로그램 카운터 어드레스에 대한 오프셋)들을 구비한 점프 명령들(분기 명령이라고도 명명될 수 있음)을 지원하는 일 실시예에서, 새로운 프로그램 카운터 어드레스를 얻기 위하여 성공적인 점프 명령어의 상대 어드레스의 최하위 비트들이 프로그램 카운터 어드레스에 더해지고, 최상위 비트들은 모드 레지스터(48)를 로딩하기 위해 독립적으로 취급될 수 있으며, 최하위 비트들로부터 모드 비트들로의 오버 플로우가 디스에이블된다.

[0042] 일실시예에서, 시퀀서(10)는 "서브루틴으로의 점프"와 "서브루틴으로부터 복귀" 명령어들 및/또는 레지스터에 기반을 둔 점프 타겟 어드레싱의 실행을 제공한다. 서브루틴으로의 점프가 수행되면, 현재의 증가된 프로그램 카운터 값은 점프가 수행될 때 저장된다. 나중에, "서브루린으로부터 복귀" 명령어가 수행되면 이 프로그램 카운터 값은 프로그램 카운터 레지스터에 로딩된다. 본 실시예에서는, 모드제어값이 서브루틴으로의 점프 명령어에 응하여 역시 기억되며 서브루틴으로부터 복귀 명령어에 응하여 재판독된다. 유사하게, 임의의 점프 타겟 어드레스가 레지스터로부터 제공되면, 모드 비트들이 물론 그 레지스터로부터 로딩될 수 있다.

[0043] 이런 식으로 선택될 수 있는 상이한 모드들의 수는 이를 위하여 사용된 모드 비트들의 수에 좌우되는 점을 이해하여야 한다. 상이한 모드들은 상이한 길이를 갖는 명령어들을 제공할 수 있다. 그러나 일 실시예에서는 모드들 중 복수의 상이한 모드가 같은 길이를 갖는 명령어들을 제공할 수 있고, 점프 타겟 어드레스의 최상위 비트들은 이 모드들 사이에서 선택하는데 사용된다.

[0044] 이와 같이 상이한 명령어 모드들 사이에서 절환하기 위한 점프 명령들의 사용은 여기서 연산 선택 코드, 결과 레지스터 어드레스, 및 연산을 위한 오퍼랜드 레지스터 어드레스들은 각기 같은 명령어에 포함되는 (즉, 파이프라이닝이 상이한 명령어 싸이클들에서 이 명령의 부분들을 사용할 수 있게 요구되도록) 프로세서들에 적용될 수도 있다는 점을 이해하여야 한다. 이러한 프로세서들에서, 명령어의 크기는 기능적 유닛들 명령 정보의 그룹들이 포함되는 각 명령어에 표시를 포함시킴으로써 전형적으로는 더 효과적으로 감소될 수 있어서, NOP(연산없음 명령)들이 기능적 유닛들의 다른 그룹들을 위하여 자동적으로 생성될 수 있다. 그러나, 이러한 표시들의 사용은 연산 선택코드, 결과 레지스터 어드레스 및 연산을 위한 오퍼랜드 레지스터 어드레스들 중 적어도 두개가 서로 상이한 명령들에 포함되는 프로세서들에 대하여 매우 비효율적으로 되는 것을 알아 내었다.

[0045] 다른 메카니즘들이 모드 선택을 위하여 사용될 수 있다는 점을 이해하여야 한다. 예를 들어, 모드 선택 비트들은 각 명령어의 일부 또는 예컨대 다음 명령어 또는 명령어들의 명령어 모드를 나타내는 명령어의 일부일 수 있다. 이 경우에, 이 비트들은 모드제어회로에 제공된다. 이는 모드에 대해 상세한 제어를 제공한다는 장점을 갖지만, 명령어의 크기가 커져서 메모리 사용의 효율성이 떨어진다는 단점도 갖는다. 다른 실시예에서, 전체 크기의 명령어 워드(word)를 위한 공간의 매 마지막 명령어만이 이러한 모드 선택 비트들을 포함하는데, 예를 들어 2개의 절반 크기 명령어들의 두번째 명령어만이 그러하다. 이는 메모리 효율이 더 높다. 모드 선택을 위한 점프 명령어들의 사용은 코드 크기를 줄인다는 장점을 갖는다.

[0046] 다른 예로서, 특별한 모드 변경 명령어들이 사용될 수 있는데, 실행유닛은 이 모드 변경 명령어들로부터 정보의 제어를 받으면서 모드값을 로딩하기 위하여 모드 제어회로를 제어한다. 모드변경명령은 예를 들어 새로운 모드를 표시하기 위하여 데이터를 포함할 수 있고 또는 명령어 모드들의 소정의 시퀀스가 정의될 수 있고, 모드 변경명령어는 이 소정의 시퀀스에서 현재의 명령어 모드를 따르는 명령어 모드에 단계를 명령한다. 모드변경 명령어들의 사용은 더 큰 점프 명령어들과 비교하여 코드 크기를 줄일 수 있지만, 모드 변경들이 종종 점프를 수반하는 경우 점프 명령어들은 더 작은 코드 크기를 가져올 수 있다.

[0047] 이하 이해하게 될 것처럼, 상술한 데이터 처리회로는 하나의 명령어에서 상이한 연산들을 처리하는 단계들을 특정할 수 있는 명령들을 사용한다. 따라서 예를 들면, 제1 연산의 연산선택코드는 제2 연산의 후기입 단계를 위한 쓰기 레지스터 어드레스에 의해 같은 명령어에서 수반될 수 있는데, 이를 위해 연산선택코드가 이전 명령어에 제공되었다. 또한, 상이한 기능적 유닛들을 위한 복수의 연산선택코드는 같은 명령어 및/또는 연산선택코드들이 이전 명령어에 또는 이전 명령어들에서 제공되었던 상이한 연산들을 위한 복수의 쓰기 레지스터 어드레스에 제공될 수 있다. 그리고, 오퍼랜드 레지스터 어드레스들은 그 명령어 및/또는 이전 명령어들에서 선택된 연산들 또는 연산을 위해 그 명령어에 제공될 수 있다. 문자 데이터가 명령어에서 선택된 연산들 또는 연산을 위하여 명령어에 제공될 수도 있다.

[0048] 일 실시예에서, 명령어에서 특정된 레지스터 어드레스의 갯수 및/또는 연산들의 갯수는 선택된 명령어 모드에 좌우된다. 일 실시예에서, 명령어에서 제공된 레지스터 어드레스의 갯수는 연산들의 갯수에 비례한다. 따라서 예를 들면 쓰기 레지스터 어드레스의 갯수는 연산선택코드들의 갯수와 같고 및/또는 오퍼랜드 레지스터 어드렛스들의 갯수는 연산 선택코드들의 갯수의 2배일 수 있다 (명령어 모드에 좌우됨).

[0049] 그러나 이와 달리 이러한 비례관계가 없는 명령어 모드들이 정의될 수 있다. 예를 들어, 하나 이상의 명령어 모드들에서, 쓰기 레지스터 어드레스들의 갯수는 연산선택코드들의 갯수보다 클 수 있다. 이는 예를 들어 상이한 연산들이 상이한 대기시간을 가지는 경우, 연산들이 결과를 제공하는 명령어 싸이클에서 연산들이 용량부족에 기인하여 쓰기 어드레스들을 특정하도록 예정될 수 없는 점을 피하기 위하여 사용될 수 있다. 또한 넓은 명령어 모드에서 시작된 연산들의 결과들을 취급하기 위하여 넓은 명령어 모드와 좁은 명령어 모드 사이에서 회로가 절환될 수 있는 전이모드(transition mode)를 규정하는데 사용될 수 있다.

[0050] 다른 예로서, 쓰기 레지스터 어드레스들의 갯수는 하나 이상의 명령어 모드들에서 연산 선택코드들의 갯수보다 적을 수 있다. 이는 예를 들어 상이한 연산들이 상이한 대기시간을 갖는 경우 그 결과들이 상이한 명령어 싸이클들에서 후기입될 복수의 연산을 병렬로 개시할 가능성을 개발하기 위해 사용될 수 있다. 또한 다른 예로서, 상이한 연산들이 중간결과를 레지스터 화일의 레지스터에 기입하지 않고 연결되는 프로그램 부분들을 위해 사용될 수 있다. 이 경우에 어떠한 기입 어드레스도 그러한 중간결과들을 생성하는 연산들을 위해 제공될 필요가 없다. 따라서 보다 적은 기입 어드레스들을 구비한 명령어들이 이 타입의 실행을 지원하는 명령어 모드들에서 사용될 수 있다. 추가 예로서, 그 결과들이 넓은 명령어 모드에서 기입될 더 많은 연산들을 개시할 수 있도록 하기 위하여 회로가 좁은 명령어 모드와 넓은 명령어 모드 사이에서 절환될 수 있는 전이모드에서 더 적은 기입 어드레스들이 사용될 수 있다.

[0051] 상이한 갯수의 쓰기 레지스터 어드레스 및/또는 오퍼랜드 레지스터 어드레스의 사양을 가능하게 하는 것은 별론으로 하고, 상이한 명령어 모드들은 레지스터 어드레스들의 상이한 사양 크기들을 제공할 수도 있다. 따라서 예를 들어 제1 명령어 모드에서는 4 비트 레지스터 어드레스들이 명령어들에서 사용될 수 있는데, 각각은 16개의 레지스터중 하나를 특정하고, 제2 명령어 모드에서는 6 비트 레지스터 어드레스들이 사용될 수 있는데, 각각은 64개의 레지스터중 하나를 특정한다. 일실시예에서, 제1 명령어 모드는 제2 명령어 모드에서 어드레싱될 수 있는 레지스터들의 서브세트만의 어드레싱을 허용한다.

[0052] 일 실시예에서, 회로를 위한 프로그램은 하나의 모드에서 동작하면서 개시될 임의의 연산과 그리고 다른 모드에서 동작하면서 기입될 연산의 결과를 초래하는 것이 방지된다. 즉, 프로그램을 컴파일하는 동안, 후속으로 분기가 취해질 지점 이후에 그 결과가 생성되도록 연산들이 예정되는 것이 방지된다 (분기는 프로그램 카운터 어드레스가 변경될 때 취해진다고 한다; 전형적으로는 이는 분기명령이 실행유닛에 적용되는 명령 싸이클 이후의 하나 이상의 명령어 싸이클이다), 회로를 위한 프로그램들이 다른 모드에서 오퍼랜드를 검색하기 위하여 어드레싱된 레지스터들로부터 오퍼랜드들을 사용하여 하나의 모드에서 동작하면서 개시될 임의의 연산을 낳는 이 타입의 실시예에서 유사하게 방지된다.

[0053] 그러나 다른 실시예에서, 그러한 제한이 부과되지 않을 수 있다. 예를 들어, 모드 전이가 연산을 선택하는 명령어들 내에서 가능한 연산을 위하여 넓은 범위의 결과 레지스터 어드레스들을 제공하기 위하여 사용될 수 있다. 이 경우에, 프로그램은 연산을 위한 연산선택코드를 포함하는 제1 명령어와, 연산을 위한 결과 레지스터 어드레스를 포함하는 제2 명령어와, 명령어 모드를 변경하는 명령어를 가져서, 제1 명령어는 제1 명령어 모드에서 실행되고 제2 명령어는 제2 명령어 모드에서 실행되는데, 제2 명령어 모드는 제1 명령어 모드에서 어드레싱될 수 없는 레지스터들의 어드레싱을 허용한다.

[0054] 다른 실시예에서, 상이한 모드들은 상이한 세트의 특화된 연산들로부터의 선택을 제공하지만 반드시 상이한 갯수의 연산 선택들 및/또는 상이한 명령어 모드들의 레지스터 선택들일 필요는 없다. 일 실시예에서, 예를 들어 하나 이상의 영상 압축, 영상 압축해제, 오디오 압축, 오디오 압축해제 및 컴퓨터 그래픽 처리를 위한 상이한 명령어 모드들이 제공될 수 있다. 특화된 연산들은 전형적으로는 전용 하드웨어에 의해 지원되는데 영상압축을 위한 블록 정합 연산, 영상 압축 및 압축해제를 위한 DCT 변환 명령들, 컴퓨터 그래픽 처리를 위한 텍스쳐 맵핑 또는 비트 블리팅(blitting) 연산들, 가변길이 압축/압축해제 연산 등으로 이 모드들의 각각을 위하여 규정될 수 있다. 상이한 명령어 모드들은 이 연산들의 각각의 상이한 서브세트들로부터 연산들의 선택을 제공한다.

[0055] 도5는 프로그램 개발 시스템을 나타낸다. 이 시스템은 컴파일러 프로그램을 실행하도록 프로그램된 프로세서(50)와, 소스 코드 저장장치(52)와, 컴파일된 코드를 명령어 메모리들에 프로그램하기 위한 프로그래밍 시스템(54)을 구비한다. 프로그램 가능한 비휘발성 메모리(예컨대 플래쉬 메모리)가 명령어 메모리를 위해 또는 그 일부를 위해 사용된다면, 프로그래밍 시스템(54)은 비휘발성 메모리 프로그래밍 회로를 사용할 수 있다. 이러한 회로는 복수의 명령어 메모리들을 프로그램하기 위하여 오프라인으로 사용될 수 있다. 그러나 집적회로의 마스크 프로그래밍 등의 다른 형태의 프로그래밍들이 사용될 수 있다.

[0056] 도6은 컴파일러의 연산을 나타낸다. 제1 단계(61)에서, 컴파일러는 예를 들어 C 또는 C++ 등의 고급언어나 중간언어로 된 소스 코드를 수신한다. 제2 단계(62)에서는, 컴파일러가 프로그램의 각각의 부분들을 위하여 명령어 모드들을 선택한다. 이는 예를 들어 명시적으로 프로그램된 모드선택들에 의거하여 또는 프로그램의 임시로 컴파일된 버전을 실행하여 얻어진 데이터를 프로파일링하는 것에 의거하여 행해질 수 있는데, 프로그램중 종종 실행되는 부분들은 넓은 명령어들을 가진 명령어 모드를 할당하고 덜 실행되는 프로그램 부분들은 좁은 명령어들을 가진 명령어 모드를 할당한다. 다른 실시예에서는, 상이한 명령어 모드들이 상이한 타입의 작업(예컨대 영상 압축, 영상 압축해제, 컴퓨터 그래픽, 오디오 압축, 오디오 압축해제 등)을 지원하는데, 상이한 명령어 모드들이 그 부분에서 수행된 작업에 따라 상이한 프로그램 부분들에 할당될 수 있다.

[0057] 이어서, 컴파일러는 소스 코드의 프로그램 부분들을 컴파일한다. 기본적으로, 각 부분은 프로그램 부분에 할당되었던 명령어 모드에 의해 정의된 상이한 가공의(notional) 프로세서를 대상으로 하여 컴파일된다. 제3 단계(63)에서는, 컴파일러가 프로그램 부분에 할당된 명령어 모드에서 이용가능한 명령어들로부터 프로그램 부분의 소스 코드를 구현하기 위한 연산들을 선택한다. 제4 단계(64)는 연산들과 이 연산들을 위한 레지스터 읽기 및 쓰기 동작들을 예정하고 있는데, 즉 연산들과 이 연산들을 위한 레지스터 읽기 및 쓰기 동작들의 실행에 명령어 싸이클들을 할당한다. 예약 기술은 이미 알려져 있다, 일 실시예에서, 예약은 연속적인 명령어들에 스테핑(stepping)하고 그리고 각 명령어에 대하여 데이터 의존성을 고려하여 명령어에 예약될 수 있는 연산들을 구하고 그리고 다른 연산들을 위한 오퍼랜드로서 후에 사용하기 위하여 결과들을 저장하기 위한 레지스터들을 할당

함으로써 수행된다. 결과들을 쓰기 위한 명령어들의 공간 가용성이나 레지스터들의 가용성에 관한 제약들이 충족되지 않는다면 되추적기법이 사용될 수 있다.

[0058] 제5 단계(65)에서는, 컴파일러가 명령어들엣 연산들을 선택하기 위한 연산코드들을 위치시킨다. 제6 단계(66)에서는, 컴파일러가 명령어들에 레지스터들에 결과들을 쓰기 위한 레지스터들을 어드레싱하기 위한 코드들을 위치시킨다. 전형적으로는, 레지스터들을 어드레싱하기 위한 코드들이 위치되는 명령어들은 명령어를 실행하는데 요구되는 지연을 추가함으로써, 연산코드들이 실행되는 명령들에 따라서 선택된다. 제7 단계(67)에서는, 컴파일러가 명령어들에서 레지스터들로부터 오퍼랜드들을 읽기 위한 레지스터들을 어드레싱하기 위한 코드들을 위치시킨다. 일 실시예에서, 이 코드들은 그 안에 연산선택코드가 위치된 같은 명령어에 위치된다. 다른 실시예에서는, 오퍼랜드 데이터의 버퍼링이 지원되거나 또는 연산코드를 발하기 전에 오퍼랜드들이 어드레싱될 필요가 있고, 이 오퍼랜드 선택 코드들과 연산선택코드들은 서로 상이한 명령어들에 배치될 수 있다. 임의선택사항으로서, 레지스터 어드레싱 코드를 위치시키기 위한 명령어들이 명령어들의 공간 가용성에 따라서 선택된다.

[0059] 제8 단계(68)에서는, 컴파일러가 상이한 명령 세트들을 가지고 그리고 하나의 프로그램 부분에서 다른 프로그램 부분으로의 점프 명령어에서 실행되는 프로그램 부분들의 변환부들에서 명령어들(예컨대 점프명령어)을 변경하는 명령어 모드를 삽입한다. 제9단계(69)에서는, 컴파일러가 모든 프로그램 부분들이 컴파일되었는지를 테스트하고 만약 아니라면, 다른 프로그램 부분들을 위하여 제3 단계(63)로 점프하여 돌아간다.

[0060] 일 실시예에서, 컴파일러는 점프(분기)명령어들에 대한 예약 연산들에 관한 제약을 부여한다. 특히, 이 제약은 점프(분기)가 취해진 후 연산의 결과가 이용가능해진다면 점프(분기)가 취해지기 전에 어떠한 연산도 예약될 수 있다는 점을 요구한다. 유사하게, 연산을 위하여 어드레싱되는 오퍼랜드들과 이 연산을 위한 연산코드선택이 상이한 명령어들에 포함된다면, 어떠한 연산도 점프(분기)가 취해진 후 너무 가깝게 예약될 수 없어서 점프(분기)가 취해지기 전에 연산의 오퍼랜드들이 어드레싱되어야 하는 제약이 부여될 것이다. 이런 식으로, 종래의 단일모드 예약은 연산이 모드에 할당되면 사용될 수 있다: 연산 선택. 오퍼랜드들의 로딩과 연산을 위한 결과의 쓰기는 선택모드에서 모두 수행된다.

[0061] 표1은 이런 방식으로 얻어질 수 있는 프로그램의 일부의 예를 나타낸다.

표 1

[0062]

필드1	필드2	필드3	필드4	필드5	필드6	필드7	필드8
OP1	A11	A12	R11	OP2	A13	A14	R12
JMP	1100000000		R21	OP3	A23	A24	R22
NOP	×	×	×	NOP	×	×	R32
OP4	A31	A32	×				

[0063] 각 열은 다른 명령어를 나타내고 다른 행은 명령어들에서 다른 필드를 나타낸다. 연속적인 열들의 명령어들이 연속적으로 실행된다. OP1, OP2, OP3, OP4 및 JMP가 연산 레지스터 선택 코드들이다. A11, A12, A13, A14, A23, A24, A31, A41은 오퍼랜드 선택코드들이다. R11, R12, R22 및 R32는 결과 레지스터 선택 코드들이다. 선택된 연산들이 1싸이클의 대기시간을 갖는 것으로 예에서는 가정된다. R21은 OP1에 의해 선택된 연산을 위한 결과 레지스터를 선택하고, R22는 OP2에 의해 선택된 연산을 위한 결과 레지스터를 선택하고, R32는 OP3에 의해 선택된 연산을 위한 결과 레지스터를 선택한다. R11과 R12는 이미 선택된 연산들(미도시)을 위한 결과 레지스터들을 선택한다.

[0064] 제2 명령어의 점프 연산 선택(JMP)은 1명령어의 대기시간으로 효력이 발생하는, 즉 제4 명령으로부터 시작하는 명령어 모드 절환을 제어한다. 모드절환 이전의 제1 명령어 모드는 8개의 필드 명령어를 가지고 모드 절환후의 제2 명령어 모드는 4개의 필드 명령어들을 가지는 것으로 가정한다 (간단히 하기 위하여 수직한 행에 필드들을 나타내었지만, 실제로 제1모드와 제2 모드의 필드들은 서로 다른 크기들을 가질 수 있다). 점프명령어는 레지스터들을 특정하기 위한 복수의 필드 크기들의 크기를 가진 점프 어드레스를 위한 필드를 사용하는 점을 주목하여야 한다. 이 필드들은 점프를 취급하기 위하여 시퀀서에 라우팅된다.

[0065] 예에서는 선택된 연산들이 1싸이클 대기시간을 갖는 것으로 가정한다. 따라서 어떠한 연산 선택(NOP)들도 모드 변경이 효과를 발생하기 전에 마지막 명령어에 예약되지 않는다. 이 명령어는 점프 명령의 뒤쪽에 연산(OP3)을 위한 결과 레지스터 선택을 포함하기만 한다. 따라서 점프 지연은 결과를 쓰기 위한 필드를 제공하는데 사용된다.

[0066] 연산 레지스터 선택 코드들(A11, A12, A13, A14, A23, A24, A31, A41)은 이 선택코드들이 그 안에 포함된 같은 명령어 내에서 선택된 연산을 위해 제공된다고 가정한다. 그러나 다른 실시예에서는 이러한 선택코드들이 초기 명령어에 제공될 필요가 있다는 것을 이해하여야 한다. 긴 대기시간의 연산들을 가진 실시예에서, 이들은 후의 명령어들에 포함될 수도 있다. 각 연산이 1싸이클 대기시간을 가지는 예가 사용되었지만, 실제로는 상이한 대기시간이 사용될 수 있으며 또는 상이한 연산들이 서로 다른 대기시간을 가질 수 있다는 점을 이해하여야 한다. 이 경우에 결과 레지스터 선택코드들은 대응하여 오프셋되는 명령어들에 위치된다. 어떤 경우에는, 결과 레지스터 선택코드들이 결과가 먼저 생성되는 명령어보다 심지어는 후에 위치될 수 있어서, 쓰기를 지연시키기 위하여 기능적 유닛의 버퍼링을 사용할 수 있게 한다.

[0067] 다른 실시예에서, 컴파일러는 연산들중 적어도 일부에 이러한 "단일모드" 제약을 부여하지 않는다. 따라서 예컨대 연산을 선택하기 위한 코드는 하나의 모드에서 실행되는 명령어에 포함될 수 있고 연산의 결과를 쓰기 위한 레지스터를 어드레싱하는 코드가 다른 모드에서 실행되는 명령어에 포함될 수 있고 및/또는 연산의 오퍼랜드를 판독하기 위한 레지스터를 어드레싱하기 위한 코드가 다른 모드에서 실행되는 명령어에 포함될 수 있다. 표2는 결과적인 프로그램 일부의 예를 나타낸다.

표 2

[0068]

필드1	필드2	필드3	필드4	필드5	필드6	필드7	필드8
OP1	A11	A12	R11	OP2	A13	A14	R12
JMP	1100000000		R21	OP3	A23	A24	R22
OP4	A31	A32×	×	NOP	×	×	R32
OP5	A41	A42	R41				

[0069] 본 명세서에서는, (JMP 연산에 의해 야기된) 제1명령어 모드에서 제2 명령어 모드로의 명령어 모드 변경이 제4 명령어(제4열)에서 효과를 발생하고, 제4열의 명령어는 제2 명령어 모드에서 실행되는 것으로 가정한다. 제1명령어 모드가 여전히 적용되는 동안 실행되는 제3 명령어는 제4명령어의 실행시 기록될 수 있는 결과를 생성하는 것으로 생각되는 연산의 연산선택(OP4)을 포함한다. 따라서 이 제4명령어는 OP4에 의해 선택된 연산의 결과를 위한 목적지 레지스터를 선택하기 위한 선택코드(R41)를 가진 필드를 포함한다.

[0070] 이는 예컨대 모드들 사이의 절환을 제어하는 점프명령어가 모드절환을 제어하는데에만 포함되는 경우 (즉, 프로그램이 점프 타겟에 도달하는 오직 하나의 방법만을 제공한다면) 행해진다. 이 경우에, 어떤 연산들이 점프명령어 전에 개시되어질 것인지가 유일하게 알려져 있고, 점프 타겟 후 명령어들 내에 있는 이 연산들을 위한 결과 레지스터 어드레스들의 포함을 예약하는데 문제가 없다. 이 경우에 컴파일러는 이러한 제약들 없이 작동하도록 배열될 수 있다. 이는 실행속도나 코드크기를 감소시킬 수 있는 제약들을 제거한다. 그리고, 이 경우에 모드 절환이 점프가 취해지기 전 적용된 모드에서 이용할 수 없었던 결과 레지스터들에 기록을 제공하는데 사용될 수도 있다. 연산 코드가 점프타겟 뒤쪽에서 선택되는 연산들을 위한 오퍼랜드 레지스터 어드레스들의 포함을 예약하는데에도 같은 것이 행해진다.

[0071] 일 실시예에서, 컴파일러는 프로그램 부분을 위한 연산들을 구현하는 명령어들의 끝 다음에 결과들이 아직 기록되어야 하는 연산들의 결과들의 리스트를 생성한다. 이 결과들이 이용가능해 질 명령어 싸이클의 표시들이 이 리스트에 추가될 수 있다. 이 리스트는 제3단계(63)로 통과되어 컴파일러가 다음 프로그램 부분을 위한 명령어들에 이 결과들을 위한 수신 어드레스들을 포함시킬 수 있게 하는데, 다음 프로그램 부분은 상이한 명령어 모드를 위해 컴파일된다. 예약 단계(제4단계; 64)를 실행함에 있어서, 컴파일러는 리스트 상에서 결과들을 위한 명령어들을 선택한다.

[0072] 일 실시예에서, 프로그램의 변환점 바로 전 및 바로 후의 명령어들에 특별한 주의를 기울이는데, 명령어 모드의 변경들이 발생한다. 이 실시예의 제3 단계(63)에서는, 컴파일러가 변환점 전 소정의 거리 내에서 제1 명령어 모드의 명령어들내의 배치를 위한 연산 코드들에 대한 검색을 제한하며, 선택이 명령어들의 공간 가용성에 의해 또는 변환점 후에 적용된 제2 명령어 모드를 위한 어드레싱 가능한 "자유로운" 레지스터들의 가용성에 의해 제약된다. 따라서 예를 들어 컴파일러는 제2 명령어 모드가 더 많은 결과 레지스터 어드레스들이나 더 큰 세트의 어드레싱 가능한 레지스터들을 제공한다면 변환점 전 소정의 거리 내에 있는 명령어들에 대하여 이러한 타입의 한정적인 제약을 덜 부여할 것이다. 다른 예에서는, 컴파일러는 제2 명령어 모드가 더 적은 결과 레지스터 어드레스들이나 더 작은 세트의 어드레싱 가능한 레지스터들을 제공한다면 변환점 전 소정의 거리 내에 있는 명령어들에 대하여 이러한 타입의 더 많은 한정적인 제약들을 부여할 것이다.

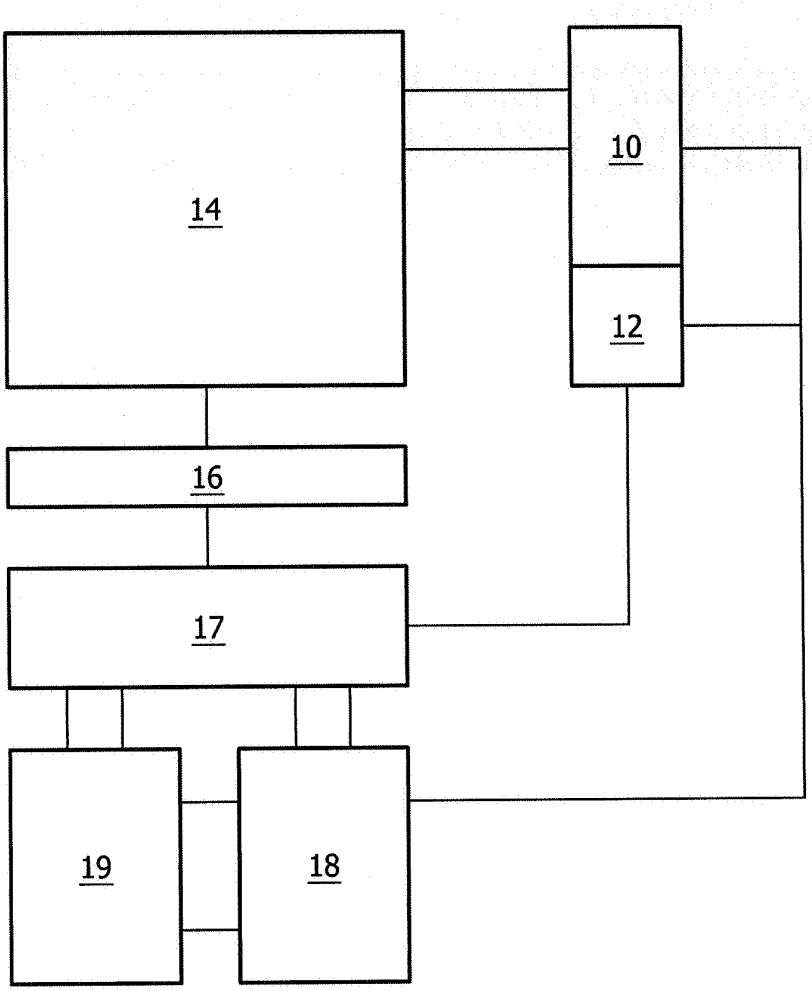
[0073] 다른 예에서는, 단일 모드 제약은 점프 명령어를 취하기 전 연산들을 개시하기 위한 표준규약을 사용함으로써 완화될 수 있다. 서브루틴에서 복귀하는 경우, 예를 들어 표준규약은 서브루틴 결과에 이르는 연산이 서브루틴으로부터의 복귀에 대응하는 분기에 대하여 소정의 오프셋을 가지고 개시되는 것일 수 있다. 이 경우에 제어가 복귀되는 호출코드는 서브루틴 결과를 위한 결과 레지스터의 어드레스의 선택을 포함할 수 있다. 이는 레지스터 사용효율을 증가시키는데 왜냐하면 어떠한 표준 서브루틴 결과 레지스터도 보존될 필요가 없기 때문이다. 더 일반적으로는, 점프 타겟이 프로그램의 상이한 부분들로부터 도달될 수 있다면, 이 모든 부분들이 분기가 취해지는 명령어 싸이클에 대한 같은 명령어 싸이클의 결과를 생성하는 연산들을 개시하는 규약이 사용될 수 있다. 이 경우에 프로그램의 타겟 부분은 결과 레지스터 어드레스를 선택하기 위한 명령어를 포함할 수 있다. 따라서 컴파일러는 예약에 대하여 제한을 덜 부여하도록 준비될 수 있다. 그리고 이는 상이한 모드들의 연산코드 선택과 결과 레지스터의 어드레싱을 허용한다. 서브루틴으로의 점프 명령어를 취한 후에 연산코드가 선택되는 연산들을 위한 오퍼랜드 레지스터 어드레스들의 포함을 예약하는데 같은 것이 행해진다. 이런 식으로 서브루틴 인수(argument)들이 인수 레지스터들을 유보하지 않고 통과될 수 있다.

[0074] 일 실시예에서, 컴파일러는 관련 서브루틴 명령어로부터 복귀에 대한 제1 소정의 위치에서 서브루틴 결과를 생성하는 연산의 선택을 예약하도록 그리고 상기 서브루틴 명령어로의 점프에 대하여 제2 소정의 위치에서 결과 레지스터 어드레스의 어드레싱을 예약하도록 구성되는데, 제1 및 제2 위치는 연산의 결과가 어드레싱된 결과 레지스터와 함께 사용하기 위하여 이용가능하도록 선택된다. 컴파일러는 상이한 서브루틴들을 위한 그러한 소정의 위치들의 상이한 세트들을 사용하도록 준비될 수도 있다. 소정의 위치들의 유사한 세트들이 서브루틴 명령어로의 점프와 그리고 서브루틴내의 연산들의 선택을 취하기 전에 오퍼랜드 레지스터들을 어드레싱하는데 사용될 수 있다.

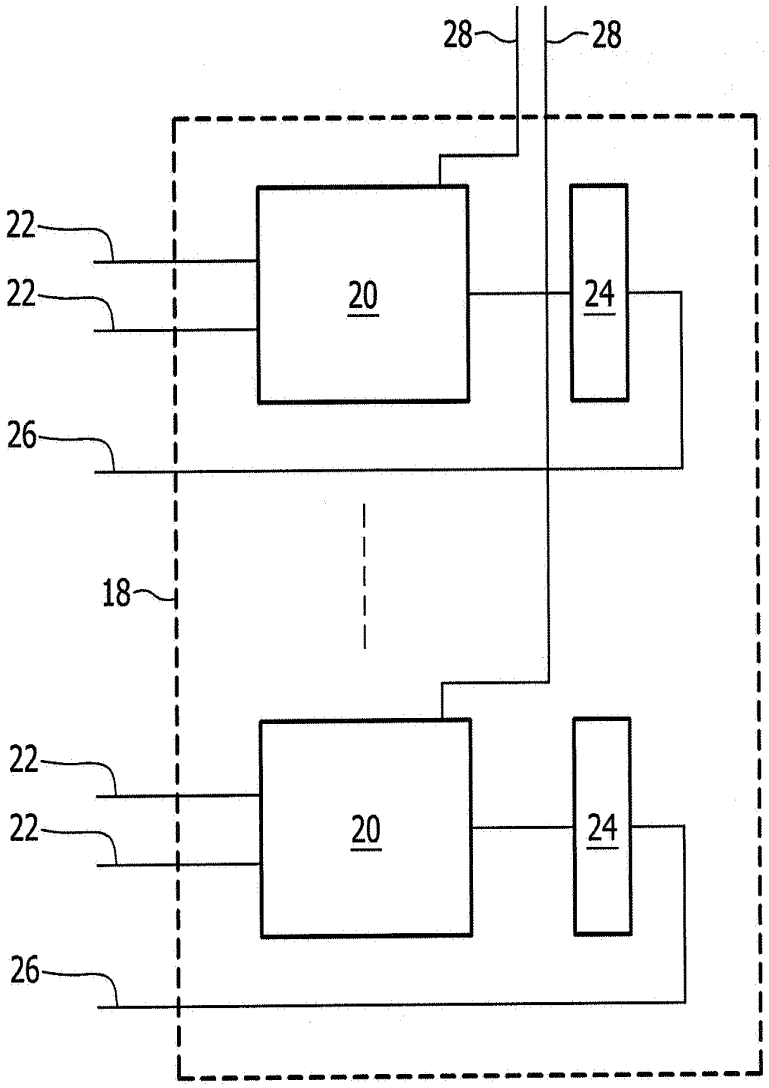
[0075] 다른 실시예에서, 컴파일러는 레지스터 화일로 중간 결과들의 쓰기 또는 읽기를 요구하지 않고 하나에서 다른 것으로 결과들을 통과시키는 연결된 연산들을 예약하도록 구성되기도 한다. 이 경우에 컴파일러는 연결을 방해하는 방식으로 다른 명령어들이 연결된 명령어들 사이에서 예약되는 것을 피하기 위하여 추가 제약들을 부여할 것이다. 이 경우에 레지스터 어드레싱의 예약을 덜 필요로 하여, 불필요한 결과 기록 동작들이 수행될 필요가 없다.

도면

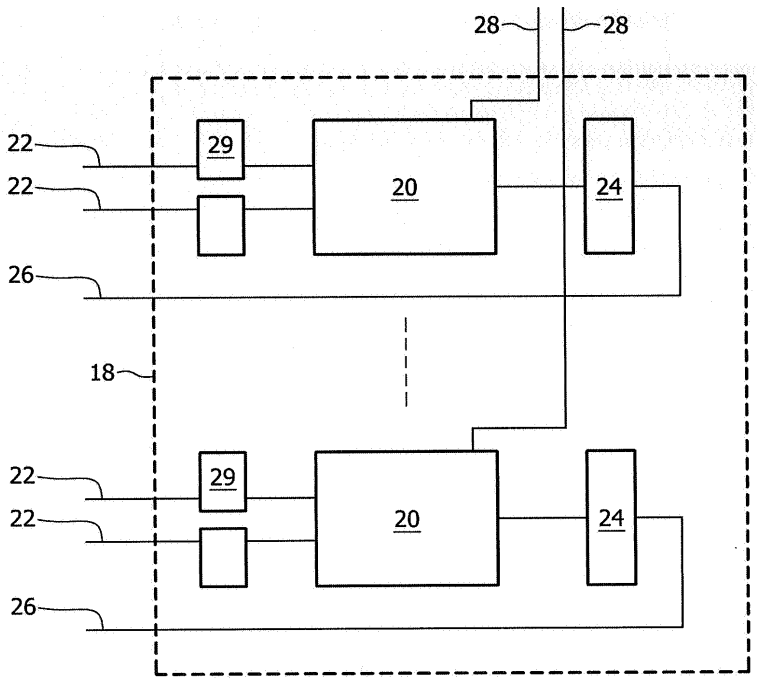
도면1



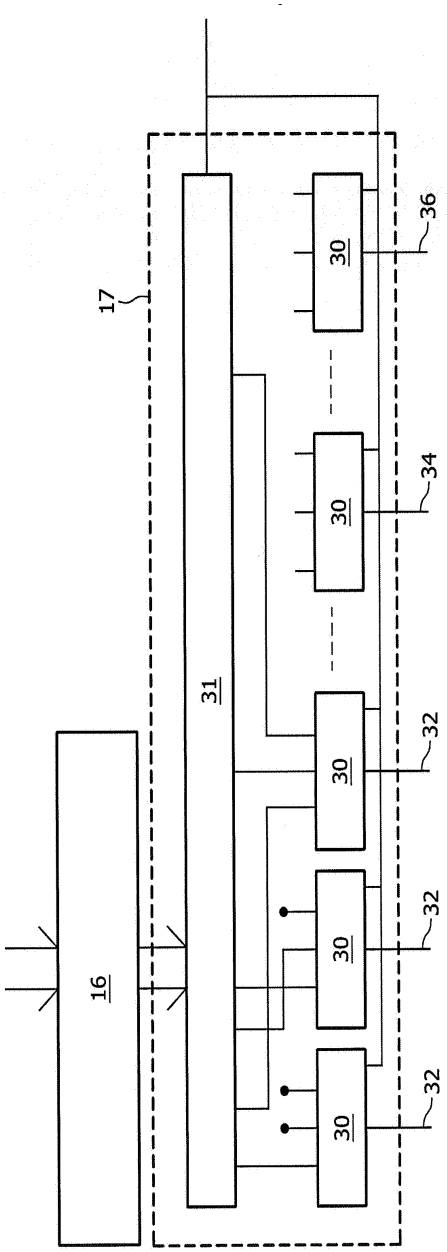
도면2



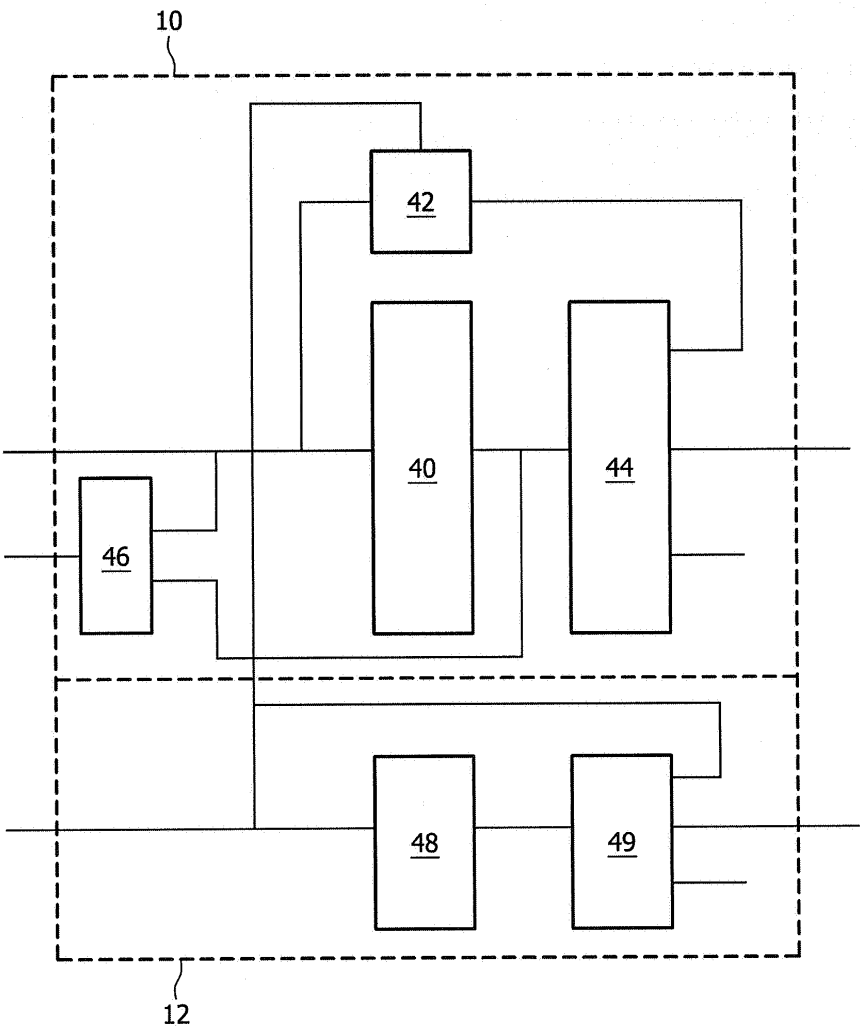
도면2a



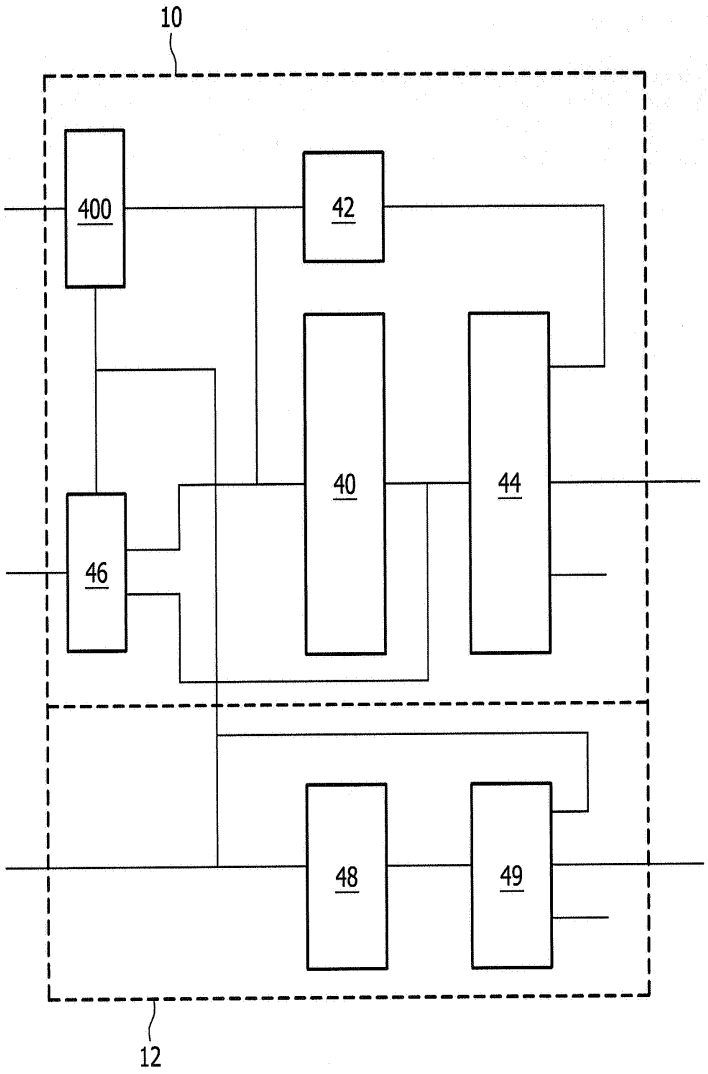
도면3



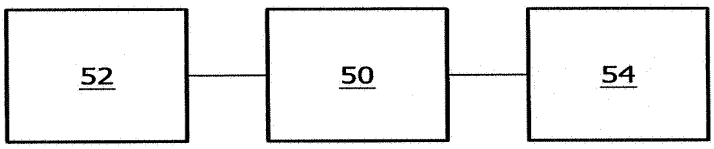
도면4



도면4a



도면5



도면6