(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2020년01월30일
(11) 등록번호 10-2071272
(24) 등록일자 2020년01월22일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2018.01) *G06F 8/40* (2018.01)
G06F 9/38 (2006.01)
(52) CPC특허분류
G06F 9/30036 (2013.01)
G06F 8/4451 (2013.01)
(21) 출원번호 10-2015-7002617
(22) 출원일자(국제) 2013년06월11일
심사청구일자 2018년06월04일
(85) 번역문제출일자 2015년01월30일
(65) 공개번호 10-2015-0037951
(43) 공개일자 2015년04월08일
(86) 국제출원번호 PCT/GB2013/051530
(87) 국제공개번호 WO 2014/009689
국제공개일자 2014년01월16일
(30) 우선권주장
13/546,227 2012년07월11일 미국(US)
(56) 선행기술조사문헌
US20100042789 A1
US20060155964 A1
WO2007039837 A1
US6988183 B1

(73) 특허권자
에이알엠 리미티드
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110
(72) 발명자
라이드 엘라스테어 데이비드
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110, 에이알엠 리미티드
(74) 대리인
이화익

전체 청구항 수 : 총 20 항 심사관 : 유진태

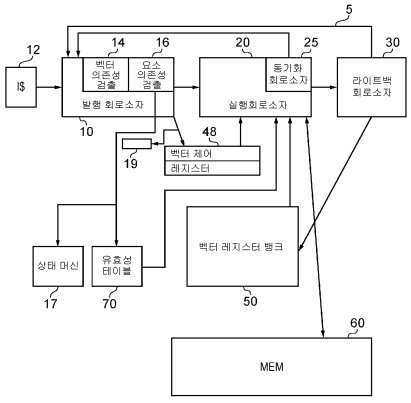
(54) 발명의 명칭 **벡터처리중 데이터 요소들의 처리순서 제어**

(57) 요 약

각각 복수의 데이터 요소를 포함하는 벡터들에 관해 연산을 행하는 벡터 명령어들의 스트림을 처리하는 데이터 처리장치가 개시되어 있다. 이 데이터 처리장치는, 처리중인 상기 벡터들을 기억하는 복수의 레지스터를 갖는 레지스터 뱅크; 상기 벡터 명령어들의 스트림을 처리하는 파이프라인식 프로세서를 구비하고, 상기 파이프라인식

(뒷면에 계속)

대 표 도 - 도1



프로세서는, 상기 벡터 명령어들의 스트림에 의해 처리되고 상기 복수의 레지스터에 기억된 상기 벡터들에 대한 데이터 의존성을 검출하고, 레지스터 데이터 해저드가 일어나지 않게 상기 벡터 명령어의 실행의 타이밍에 관한 제약을 판정하도록 구성된 회로소자를 구비하고, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 상기 명령어 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 명령어 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 명령어 스트림의 순서와 다른 순서로 생기는 경우에, 일어난다. 상기 파이프라인식 프로세서는, 데이터 의존성이 식별되어 있는 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하여, 존재하지 않는 경우 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키도록 구성된 데이터 요소 해저드 판정회로소자를 구비한다.

(52) CPC특허분류
G06F 9/30018 (2013.01)
G06F 9/3838 (2013.01)
G06F 9/384 (2013.01)

명 세 서

청구범위

청구항 1

각각 복수의 데이터 요소를 포함하는 벡터들에 관해 연산을 행하는 벡터 명령어들의 스트림을 처리하는 데이터 처리장치로서,

처리중인 상기 벡터들을 기억하는 복수의 레지스터로 이루어진 레지스터 뱅크;

상기 벡터 명령어들의 스트림을 처리하는 파이프라인식 프로세서를 구비하고,

상기 파이프라인식 프로세서는, 상기 벡터 명령어들의 스트림에 의해 처리되고 상기 복수의 레지스터에 기억된 상기 벡터들에 대한 데이터 의존성을 검출하고, 레지스터 데이터 해저드가 일어나지 않게 상기 벡터 명령어들의 실행의 타이밍에 관한 제약을 판정하도록 구성된 회로소자를 구비하고, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 상기 벡터 명령어들의 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 벡터 명령어들의 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 벡터 명령어들의 스트림의 순서와 다른 순서로 생기는 경우에, 일어나고,

상기 파이프라인식 프로세서는, 데이터 의존성이 식별되어 있는 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하여, 데이터 의존성이 존재하지 않는 경우 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키도록 구성된 데이터 요소 해저드 판정회로소자를 구비한, 데이터 처리장치.

청구항 2

제 1 항에 있어서,

상기 데이터 처리장치는 벡터내의 데이터 요소들을 소정의 액세스 순서로 액세스하도록 구성되고, 상기 데이터 요소 해저드 판정회로소자는, 벡터 명령어에 따라 벡터 내에서 적어도 하나의 데이터 요소를 갱신할 때 상기 벡터 내의 상기 소정의 액세스 순서로 이후의 데이터 요소들이, 변경되어도 되고 폐기될 수 있는 값들을 현재 포함하는지를 판정하고, 상기 이후의 데이터 요소들 중 적어도 하나를 갱신하기 위해 적어도 하나의 또 다른 벡터 명령어에 따라, 상기 데이터 요소를 갱신하기 위해 상기 연산에 대해 임의의 순서로 적어도 하나의 연산을 행할 수 있는 신호를 실행 회로소자에 보내도록 구성된, 데이터 처리장치.

청구항 3

제 2 항에 있어서,

상기 벡터는 파티션들로 분할되고, 각 파티션은 적어도 하나의 데이터 요소를 포함하고, 여기서 파티션들은 인접하는 데이터 요소들인 다수의 데이터 요소를 포함하며, 상기 데이터 처리장치는 소정의 액세스 순서로 벡터내의 상기 파티션들을 액세스하도록 구성된, 데이터 처리장치.

청구항 4

제 1 항 내지 제 3 항 중 어느 한 항에 있어서,

상기 데이터 요소 해저드 판정 회로소자는 상태 머신을 포함하고, 상기 상태 머신은, 명령어에 의해 상기 벡터를 액세스할 때 상기 벡터내의 데이터 요소들에 의해 기억된 값들의 상태를 나타내고, 이 상태는 보존되는 데이터 값들이나 폐기될 수 있는 데이터 값들을 상기 데이터 요소들이 기억하고 있는 중인지를 나타내는, 데이터 처리장치.

청구항 5

제 3 항에 있어서,

상기 데이터 요소 해저드 판정 회로소자는 상태 머신을 포함하고, 상기 상태 머신은, 명령어에 의해 상기 벡터를 액세스할 때 상기 벡터내의 데이터 요소들에 의해 기억된 값들의 상태를 나타내고, 이 상태는 보존되는 데이터 값들이나 폐기될 수 있는 데이터 값들을 상기 데이터 요소들이 기억하고 있는 중인지를 나타내고,

상기 상태 머신은 3개의 상태, 즉 모든 데이터 요소들이 보존되는 데이터 값을 포함하는 것을 나타내는 미지의 상태와, 상기 벡터 명령어들이 현재 액세스중인 현재의 파티션 이후의 모든 데이터 요소들이 폐기될 수 있고 그 밖의 모든 데이터 요소들이 보존되는 것을 나타내는 액세스 현재 상태와, 상기 현재의 파티션내의 모든 데이터 요소들과 이후의 모든 데이터 요소들이 폐기될 수 있고 이전의 모든 데이터 요소들의 데이터 값이 보존되는 것을 나타내는 이전의 파티션에 의해 액세스된 상태로, 이루어지는, 데이터 처리장치.

청구항 6

제 5 항에 있어서,

상기 데이터 처리장치는, 상기 명령어가 어느 쪽의 파티션을 연산할지를 제어하는 복수의 벡터 제어 레지스터와, 상기 복수의 벡터 제어 레지스터 중 하나가 현재의 파티션을 나타내고 있는 것을 나타내는 값을 기억하는 스토어를 구비하고, 다음 파티션을 나타내는 상기 벡터 제어 레지스터 중 하나가 이전의 파티션을 나타낸 상기 벡터 제어 레지스터와 다른 상기 기억된 값으로부터 검출하는 것에 따라, 상기 상태 머신은 미지의 상태로 전환되는, 데이터 처리장치.

청구항 7

제 5 항에 있어서,

상기 이전의 파티션에 의해 액세스된 상태는, 현재의 파티션에 관한 벡터 연산과 이전의 파티션에 관한 벡터 연산을 임의의 순서로 행할 수 있는 것을 상기 데이터 요소 해저드 판정 회로소자에 나타내는, 데이터 처리장치.

청구항 8

제 1 항 내지 제 3 항 중 어느 한 항에 있어서,

상기 벡터 명령어들의 스트림이 처리한 상기 벡터에 대한 데이터 의존성을 검출하도록 구성된 상기 회로소자는, 상기 데이터 의존성을 판정할 때 데이터 의존성이 없는 것을 나타내는 상기 벡터 명령어들에 관련된 주석에 응답하는, 데이터 처리장치.

청구항 9

제 2 항에 있어서,

상기 데이터 처리장치는, 상기 벡터 명령어들내에서 레지스터 식별자들에 의해 식별된 레지스터를 상기 레지스터 뱅크내에서 상기 복수의 레지스터 중 하나에 의해 매핑하는 레지스터 리네이밍 회로소자를 더 구비하고;

벡터 명령어에 따라 벡터내에서 적어도 하나의 데이터 요소가 갱신되는 것과, 상기 벡터내에 상기 소정의 액세스 순서로 이후의 데이터 요소들이 현재 변경되어도 되고 폐기될 수 있는 값들을 포함하는 것을, 상기 데이터 요소 해저드 판정 회로소자가 판정하는 것에 따라, 상기 파이프라인식 프로세서는 상기 벡터 명령어들내에서 상기 레지스터 식별자들에 의해 식별된 상기 레지스터에 현재 매핑된 상기 레지스터를 갱신하도록 구성된, 데이터 처리장치.

청구항 10

제 9 항에 있어서,

상기 파이프라인식 프로세서는, 벡터의 제1 데이터 요소가 액세스 순서로 갱신될 때를 판정하고, 상기 벡터가 임의의 현재 미사용된 레지스터에 기록될 수 있는 것을 나타내는 신호를 송신하는 추가의 회로소자를 구비한, 데이터 처리장치.

청구항 11

제 10 항에 있어서,

상기 추가의 회로소자는, 미지의 상태를 나타내는 제1 상태와 액세스되는 상기 벡터의 제1 요소가 갱신되는 것을 나타내는 적어도 하나의 또 다른 상태를 갖는, 제1 요소 상태 머신을 구비한, 데이터 처리장치.

청구항 12

제 1 항 내지 제 3 항 중 어느 한 항에 있어서,

상기 데이터 처리장치는, 상기 데이터 요소가 보존되어야 하는 데이터 값이나 폐기될 수 있는 데이터 값을 현재 기억중인가를 나타내는 벡터내의 데이터 요소마다 지시하는 지시자를 기억하기 위한 데이터 스토어를 더 구비하고, 상기 데이터 요소 해저드 판정 회로소자는 상기 지시자를 기억하기 위한 데이터 스토어를 갱신하도록 구성된, 데이터 처리장치.

청구항 13

제 12 항에 있어서,

상기 데이터 처리장치는, 벡터 레지스터 요소를 판독하는 명령어에 따라, 폐기될 수 있는 데이터 값을 기억하는 것으로서 표시되는 파티션내의 데이터 요소마다 상기 데이터 요소에 현재 기억된 값 대신에 소정의 값을 기억하도록 구성된, 데이터 처리장치.

청구항 14

제 1 항 내지 제 3 항 중 어느 한 항에 있어서,

상기 파이프라인식 프로세서는, 상기 벡터 명령어들을 발행하여 실행하는 발행 회로소자 및 실행 회로소자를 구비하고, 상기 발행 회로소자는 상기 데이터 요소 해저드 검출 회로소자를 구비한, 데이터 처리장치.

청구항 15

제 14 항에 있어서,

상기 데이터 처리장치는 동기화 회로소자를 구비하고, 상기 동기화 회로소자는, 상기 실행 회로소자가 현재 처리중인 데이터 요소들의 상태를 주기적으로 판정하여 상태 정보를 상기 발행 회로소자에 송신하도록 구성된, 데이터 처리장치.

청구항 16

제 1 항 내지 제 3 항 중 어느 한 항에 있어서,

상기 벡터는, 벡터 연산에 의해 연산될 데이터 요소들을 식별하도록 구성된 벡터 마스크들을 포함하고, 상기 데

이터 요소 해저드 판정 회로소자는, 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 적어도 일부마다 존재하는지를 상기 벡터 마스크들로부터 판정하여, 존재하지 않는 경우 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키도록 구성된, 데이터 처리장치.

청구항 17

제 5 항에 있어서,

상기 벡터는, 벡터 연산에 의해 연산될 벡터 파티션들내의 마스킹된 데이터 요소들을 식별하도록 구성된 벡터들에 관련된 벡터 마스크들을 갖고, 상기 데이터 처리장치는, 벡터 마스크의 제어하에 벡터 파티션에 관한 하나의 연산과, 상기 벡터 마스크의 반대의 제어하에 상기 벡터 파티션에 관한 또 다른 연산을 행하도록 구성되고, 상기 상태 머신은, 상기 벡터 명령어들에 의해 현재 액세스중인 상기 현재의 파티션 이후의 데이터 요소 모두와 상기 벡터 파티션내의 상기 마스킹된 데이터 요소들이 폐기될 수 있고, 그 밖의 데이터 요소 모두가 보존되는 것을 나타내는 마스킹된 액세스 현재 상태를 포함하는 적어도 하나의 또 다른 상태를 포함한, 데이터 처리장치.

청구항 18

데이터 처리장치내에서 각각이 복수의 데이터 요소들을 포함하고 복수의 레지스터에 기억되는 벡터들에 관해 연산을 행하는 벡터 명령어들의 스트림을 처리하는 처리 방법으로서,

레지스터 데이터 해저드가 일어나지 않게 상기 벡터 명령어들의 실행의 타이밍에 관한 제약을 판정하기 위해서 상기 벡터 명령어들의 스트림에 의해 처리되고 상기 복수의 레지스터에 기억된 상기 벡터들에 대한 데이터 의존성을 검출하는 단계로서, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 상기 벡터 명령어들의 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 벡터 명령어들의 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 벡터 명령어들의 스트림의 순서와 다른 순서로 생기는 경우에, 일어나는, 상기 검출하는 단계;

데이터 의존성이 식별되어 있는 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해, 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하는 단계;

데이터 의존성이 존재하지 않는 경우, 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키는 단계를 포함하는, 처리 방법.

청구항 19

제 18 항에 있어서,

상기 데이터 처리장치는 벡터내의 데이터 요소들을 소정의 액세스 순서로 액세스하도록 구성되고, 상기 실행의 타이밍에 관한 상기 판정된 제약을 완화시키는 단계는,

벡터 명령어에 따라 벡터 내에서 적어도 하나의 데이터 요소를 갱신할 때 상기 벡터 내의 상기 소정의 액세스 순서로 이후의 데이터 요소들이, 현재 변경되어도 되고 폐기될 수 있는 값들을 포함하는 것을 판정하는 단계; 및

상기 이후의 데이터 요소들 중 적어도 하나를 갱신하기 위해 적어도 하나의 또 다른 벡터 명령어에 따라, 상기 데이터 요소를 갱신하기 위해 상기 연산에 대해 임의의 순서로 적어도 하나의 연산을 행할 수 있는 신호를 실행 회로소자에 보내는 단계를 포함하는, 처리 방법.

청구항 20

각각 복수의 데이터 요소를 포함하는 벡터들에 관한 연산을 행하는 복수의 벡터 명령어로 이루어진 컴퓨터 프로그램을 변환시키는 변환 방법으로서,

상기 복수의 벡터 명령어를 해석하는 단계;

레지스터 데이터 해저드가 일어나지 않게 상기 복수의 벡터 명령어의 실행의 타이밍에 관한 제약을 판정하기 위해서 상기 복수의 벡터 명령어에 의해 처리되는 상기 벡터들에 대한 데이터 의존성을 검출하는 단계로서, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 명령어 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 명령어 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 명령어 스트림의 순서와 다른 순서로 생기는 경우에, 일어나는, 상기 검출하는 단계;

데이터 의존성이 식별되어 있는 상기 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해, 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하는 단계;

데이터 의존성이 존재하지 않는 경우, 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키는 단계;

상기 컴퓨터 프로그램을 데이터 처리 시스템상에서 실행하는데 적합한 코드로 변환시키는 단계를 포함하는, 변환방법.

청구항 21

삭제

발명의 설명

기 술 분 야

[0001] 본 발명은, 데이터 처리 분야에 관한 것이고, 특히 벡터 연산 처리에 관한 것이다.

배 경 기 술

[0002] 벡터 처리는, 많은 데이터 요소를 포함하는 벡터 처리를 포함한다. 상기 벡터내의 모든 데이터 요소나 선택된 데이터 요소에 대해 연산을 행하되, 이들 연산이 서로 병렬로 동시에 행해진다. 다른 요소는, 상기 벡터를 하나 이상의 요소들의 파티션으로 분할하고 나서 특별한 연산에서 처리하기 위한 파티션을 선택하여서 선택되어도 된다. 이와는 달리, 벡터내의 일부의 요소를 선택하거나 상기 연산에서 처리하기 위한 그 밖의 요소들을 선택하지 않도록, 마스크를 이용할 수 있다.

[0003] 벡터 파티셔닝 및 상기 분할된 데이터의 처리의 성능은, 병렬로 연산을 행하게 함으로써 향상시킬 수 있지만, 상당한 데이터 해저드(hazard)를 일으킬 가능성이 있다. 이들은 피해야 하고, 상기 데이터를 병렬로 처리하게 하면서 이들 연산의 데이터 일관성(consistency)과 코히어런시(coherency)를 확보하는 것은 추가의 하드웨어, 지연시간 및 전력으로 매우 비쌀 수 있다.

[0004] 벡터를 처리할 때 코히어런시와 일관성 연산의 비용을 감소시키고, 또 병렬 처리에 연관된 상기 향상된 성능으로부터 이익을 얻을 수 있는 것이 바람직할 것이다.

발명의 내용

[0005] 본 발명의 제1 국면은, 각각 복수의 데이터 요소을 포함하는 벡터들에 관해 연산을 행하는 벡터 명령어들의 스트림을 처리하는 데이터 처리장치를 제공하고, 상기 데이터 처리장치는,

[0006] 처리중인 상기 벡터들을 기억하는 복수의 레지스터로 이루어진 레지스터 뱅크;

[0007] 상기 벡터 명령어들의 스트림을 처리하는 파이프라인식 프로세서를 구비하고,

[0008] 상기 파이프라인식 프로세서는, 상기 벡터 명령어들의 스트림에 의해 처리되고 상기 복수의 레지스터에 기억된 상기 벡터들에 대한 데이터 의존성을 검출하고, 레지스터 데이터 해저드가 일어나지 않게 상기 벡터 명령어들의 실행의 타이밍에 관한 제약을 판정하도록 구성된 회로소자를 구비하고, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 상기 벡터 명령어들의 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 벡터 명령어들의 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 벡터 명령어들의 스트림의 순서와 다른 순서로 생기는 경우에, 일어나고,

[0009] 상기 파이프라인식 프로세서는, 데이터 의존성이 식별되어 있는 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하여, 존재하지 않는 경우 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키도록 구성된 데이터 요소 해저드 판정회로소자를 구비한다.

[0010] 본 발명에 의해 안 것은, 분할된 벡터 연산이 많은 데이터 해저드의 가능성을 제공한다는 것이다. 이들을 정확히 취급하면 상당한 비용을 초래할 수 있다. 또한, 본 발명에 의해 안 것은, 전형적인 이용 패턴에 있어서, 이들 데이터 해저드가 벡터간에 존재하기도 하지만, 그 해저드가 상기 벡터내의 개개의 데이터 요소간의 해저드라고 한다면, 이들 해저드들 중 많은 해저드가 사라질 수도 있다는 것이다. 이렇게 아는 것에 의해, 벡터들을 고려할 때 생성되고, 최적화 하는 경우 그들이 안전하고 벡터들내의 데이터를 반드시 변조하지 않도록, 타이밍 제약의 일부를 완화시킬 수 있다. 따라서, 본 발명은, 상기 벡터들내의 데이터 요소들의 적어도 일부 사이에서의 데이터 해저드를 판정하고, 실제의 데이터 요소 일관성을 위해 필요하지 않는 벡터간의 데이터 일관성을 확보하는데 필요한 타이밍 제약을 완화한다. 이와 관련하여, 상기 판정된 타이밍 제약을 완화한다는 것은, 특정 연산을 교대로 행하는 것이 필요한 대신에, 서로 중복시킬 수도 있거나, 다른 순서로 행할 수도 있는 것을 의미한다.

[0011] 상술한 것처럼, 데이터 해저드는, 동일한 레지스터에 적어도 하나가 기록인 2개의 액세스가 상기 명령어 스트림의 순서와 다른 순서로 일어나는 경우에 일어난다. 이와 관련하여, 명령어 스트림에서의 나중의 액세스가 앞선 액세스가 완료하기 전에 시작하는 경우에도 심지어 상기 나중의 액세스가 상기 앞선 액세스보다 나중에 완료하는 경우에도 상기 명령어 스트림의 순서와 다른 순서로 일어나는 액세스들을 고려하고, 상기 앞선 액세스가 완료하기 전에 시작하는 것에 의해, 잘못된 데이터가 액세스될 수도 있다는 점에 데이터 해저드가 생길 수도 있다.

[0012] 일부의 실시예에서, 상기 데이터 처리장치는 벡터내의 데이터 요소들을 소정의 액세스 순서로 액세스하도록 구성되고, 상기 데이터 요소 해저드 판정회로소자는, 벡터 명령어에 따라 벡터 내에서 적어도 하나의 데이터 요소를 갱신할 때 상기 벡터 내의 상기 소정의 액세스 순서로 이후의 데이터 요소들이, 현재 변경되어도 되고 폐기될 수 있는 값들을 포함하는 것을 판정하고, 상기 이후의 데이터 요소들 중 적어도 하나를 갱신하기 위해 적어도 하나의 또 다른 벡터 명령어에 따라, 상기 데이터 요소를 갱신하기 위해 상기 연산에 대해 임의의 순서로 적어도 하나의 연산을 행할 수 있는 신호를 실행 회로소자에 보내도록, 구성된다.

[0013] 본 발명의 실시예들에 의해 안 것은, 레지스터를 갱신하는데 사용된 코드는, 특정한 순서로 그렇게 하는 것이 일반적이어서, 제1 데이터 요소를 갱신한 후 이후의 데이터 요소들 등을 갱신하는 경우도 많다는 것이다. 따라서, 다른 벡터 연산을 행하고 있는 경우에도, 그 다른 벡터 연산은 동일한 벡터에 관해 행해지고 있을 수도 있고, 제1 연산은 제1의 3개의 데이터 요소를 채울지도 모르고 다음의 연산이 다음 3개의 데이터 요소를 채울지도 모른다. 이것을 염두해두고, 데이터 처리장치는, 데이터 요소들의 갱신을 검출하고 나서, 상기 벡터내의 상기 이후의 데이터 요소들에 기억된 값들이 나중에 갱신되기 때문에 나중에 필요하지 않아 폐기될 수 있다는 것을 인식하도록 구성된다. 이렇게 인식함으로써 이들 값을 겹쳐쓰는 벡터 연산을, 선행하는 데이터 요소의 갱신을 기다리지 않고 즉시 행할 수 있다. 따라서, 특정한 벡터를 갱신하는 이후의 연산은, 서로 같은 시간에 서로 병렬로, 또는 상기 명령어 스트림의 순서와 다른 순서로 행해질 수 있다.

[0014] 일부의 실시예에서, 상기 벡터는 파티션들로 분할되고, 각 파티션은 적어도 하나의 데이터 요소를 포함하고, 여기서 파티션들은 인접하는 데이터 요소들인 다수의 데이터 요소를 포함하며, 상기 데이터 처리장치는 소정의 액세스 순서로 벡터내의 상기 파티션들을 액세스하도록 구성된다.

[0015] 벡터 연산이 단일의 데이터 요소에 관해 행해지지만, 일부의 실시예에서, 상기 벡터는 분할되고, 각 연산은 특정한 파티션에 관해 동작한다. 파티션은 단일의 데이터 요소를 포함하여도 되거나, 다수의 인접한 데이터 요소들을 포함하여도 된다.

[0016] 일부의 실시예에서, 상기 데이터 요소 해저드 판정 회로소자는 상태 머신을 포함하고, 상기 상태 머신은, 명령어에 의해 벡터를 액세스할 때 상기 벡터내의 데이터 요소들에 의해 기억된 값들의 상태를 나타내고, 이 상태는 보존되는 데이터 값들이나 폐기될 수 있는 데이터 값들을 상기 데이터 요소들이 기억하고 있는 중인지를 나타낸다.

[0017] 상기 데이터 요소 해저드를 판정하는 일 방식은, 명령어가 벡터를 액세스할 때에 상기 벡터의 데이터 요소들내에 기억된 값들의 상태를 나타내는 상태 머신을 사용하는 방식이 있다. 이에 관련해서, 실제 이루어지

고 있는 액세스보다 앞의 시점에서 무슨 상태인지를 판정한다. 그 액세스보다 앞에서 그것을 함으로써, 수집되는 정보는, 그 밖의 연산의 타이밍에 영향을 미치는데 사용될 수 있고, 최적화시킬 수 있다. 상기 상태는, 기억되는 데이터 요소들이 보존될 필요가 있거나 폐기될 수 있는지를 나타낸다. 그것은, 벡터에서 어느 요소들이 계속 액세스되므로 폐기될 수 있고 이미 갱신된 요소들이 보존될 필요가 있는지를 판정하는, 액세스 순서 제약을 사용하여 행한다.

[0018] 일부의 실시예에서, 상기 벡터 명령어들의 스트림이 처리한 상기 벡터에 대한 데이터 의존성을 검출하도록 구성된 상기 회로소자는, 상기 데이터 의존성을 판정할 때 데이터 의존성이 없는 것을 나타내는 상기 벡터 명령어에 관련된 주석에 응답한다.

[0019] 본 발명의 실시예들은, 실제로 데이터 의존성이 없는 데이터 의존성을 예상할지도 모르는 명령어에 대해 나타내는 명령어들에 관련된 주석을 갖는 명령어들을 제공한다. 이러한 경우에, 그 예상된 데이터 의존성에 관련하는 상기 명령어들의 실행 계획에 관한 제약이 완화될 수 있다. 이와 관련하여, 벡터 내에 개개의 요소를 갱신하는 명령어들이 있고, 비록 표면상으로는 동일한 벡터를 갱신하는 이후의 명령어가 순서적으로 실행되어야 하지만, 각각이 서로 다른 개개의 요소를 갱신하는 경우, 이것이 필요하지 않고 그 명령어 내의 주석이 이것을 나타낼 수 있고 상기 타이밍 제약이 완화될 수 있다.

[0020] 일부의 실시예에서, 상기 상태 머신은 3개의 상태, 즉 모든 데이터 요소들이 보존되는 데이터 값을 포함하는 것을 나타내는 미지의 상태와, 상기 벡터 명령어가 현재 액세스중인 상기 현재의 파티션 이후의 모든 데이터 요소들이 폐기될 수 있고 그 밖의 모든 데이터 요소들이 보존되는 것을 나타내는 액세스 현재 상태와, 상기 현재의 파티션내의 모든 데이터 요소들과 이후의 모든 데이터 요소들이 폐기될 수 있고 이전의 모든 데이터 요소들의 데이터 값이 보존되는 것을 나타내는 이전의 파티션에 의해 액세스된 상태로, 이루어진다.

[0021] 그 상태 머신이 다수의 상태를 포함하여도 되지만, 일부의 실시예에서 필요한 정보를 제공하는데는 3개의 상태이면 충분하다. 상태가 미지이어도 되고, 그 미지의 상태에서는 안전상 모든 데이터 요소들이 보존되어야 하는 것으로서 표시된다. 이것은, 접수 불가능할 경우에 이들 요소들이 겹쳐써지지 않게 한다. 그리고, 상기 명령어가 상기 벡터내의 특정한 요소나 파티션을 액세스하고 있을 때 일어나는 액세스 현재 상태가 있고, 이 시점에서 현재의 요소나 파티션 이후의 모든 데이터 요소들이 폐기될 수 있고 그 밖의 모든 데이터 요소들이 갱신되었거나 갱신중일 때 보존될 필요가 있다는 것을 안다. 다음 파티션이 갱신될 것이지만 아직 갱신되지 않은 것을 나타내는 이전의 파티션 상태에 의한 액세스인 또 다른 상태가 있다. 따라서, 이들 상태는 최적화를 판정할 때 유용하다. 상기 미지의 상태에서는 최적화가 불가능하다. 상기 이전의 파티션 상태에서 현재의 파티션에 관한 벡터 연산과 이전의 파티션에 관한 벡터 연산을 임의의 순서로 행할 수 있지만, 이전의 파티션에 관한 상기 연산을 행하는데 필요한 어떠한 타이밍 제약도 우선 완화될 수 있다.

[0022] 일부의 실시예에서, 상기 데이터 처리장치는, 상기 명령어가 어느 쪽의 파티션을 연산할지를 제어하는 복수의 벡터 제어 레지스터와, 상기 복수의 벡터 제어 레지스터 중 하나가 현재의 파티션을 나타내고 있는 것을 나타내는 값을 기억하는 스토어(store)를 구비하고, 다음 파티션을 나타내는 상기 벡터 제어 레지스터 중 하나가 상기 이전의 파티션을 나타낸 상기 벡터 제어 레지스터와 다른 상기 기억된 값으로부터 검출하는 것에 따라, 상기 상태 머신은 미지의 상태로 전환된다. 파티션이 액세스되도록 제어할 수 있는 하나의 벡터 제어 레지스터보다 많이 있을 수도 있다. 이것은 그 자신의 문제점을 나타내고, 이러한 경우에, 데이터 스토어를 이용하여, 현재의 파티션을 나타내는 상기 벡터 제어 레지스터를 가리키는 값을 기억한다. 이렇게 하여, 다른 벡터 제어 레지스터의 제어하에 하나의 레지스터의 하나의 다음 파티션이 액세스되면, 이것을 상기 기억된 데이터 값으로부터 판정할 수 있고, 그 상태 머신에서의 값들은 더 이상 새로운 벡터 제어 레지스터에 관련되지 않기 때문에, 그 상태는 미지의 상태로 설정되어야 한다.

[0023] 일부의 실시예에서, 상기 데이터 처리장치는, 상기 벡터 명령어내에서 레지스터 식별자들에 의해 식별된 레지스터를 상기 레지스터 뱅크내에서 상기 복수의 레지스터 중 하나에 의해 매핑하는 레지스터 리네이밍(renaming) 회로소자를 더 구비하고; 벡터 명령어에 따라 벡터내에서 적어도 하나의 데이터 요소가 갱신되는 것과, 상기 벡터내에 상기 소정의 액세스 순서로 이후의 데이터 요소들이 현재 변경되어도 되고 폐기될 수 있는 값들을 포함하는 것을, 상기 데이터 요소 해저드 판정회로소자가 판정하는 것에 따라, 상기 파이프라인식 프로세서는 상기 벡터 명령어내에서 상기 레지스터 식별자에 의해 식별된 상기 레지스터에 현재 매핑된 상기 레지스터를 갱신하도록 구성된다.

[0024] 데이터 처리장치가 벡터 연산을 사용하여 레지스터를 갱신할 때 레지스터 리네이밍 회로소자를 갖는 경우, 새로운 레지스터는, 그 벡터 연산을 위해 개개의 요소가 다른 시간에 갱신되므로 리네이밍될 필요가 있고,

이전에 리네이밍된 레지스터로부터의 데이터는 그 연산이 행해질 때 이 레지스터에 통합될 필요가 있다. 이것은 매우 비용이 많이 드는 연산이다. 본 발명의 실시예들에 의해 안 것은, 특정 데이터 요소를 폐기할 수 있다는 판정은, 실제로 이전에 리매핑된 레지스터에 기억된 값들이 필요하지 않을 수도 있기 때문에, 그것을 통합해봤자 아무 소용 없고, 그 값들과 새로운 값을 간단히 겹쳐쓸 수 있다는 것을 의미한다는 것이다. 이에 따라 상당히 절감된다.

[0025] 따라서, 벡터 연산에 따라 이후의 데이터 요소들을 갱신할 때 상기 리네이밍 회로소자에 의해 현재 매핑된 물리 레지스터에 간단히 기록될 수 있고, 데이터를 추가로 리네이밍하거나 복사할 필요가 없다. 실제로, 이들 연산에 대한 레지스터 리네이밍을 억제하고 시간과 전력을 상당히 절약한다.

[0026] 일부의 실시예에서, 상기 파이프라인식 프로세서는, 벡터의 제1 데이터 요소가 액세스 순서로 갱신될 때를 판정하고, 상기 벡터가 임의의 현재 미사용된 레지스터에 기록될 수 있는 것을 나타내는 신호를 송신하는 추가의 회로를 구비한다.

[0027] 벡터의 제1 데이터 요소가 갱신될 때를 판정하는 추가의 회로소자를 갖는 상기 데이터 처리장치를 제공한다면, 레지스터 리네이밍이 존재하는 경우, 이 제1값을 새로운 레지스터에 기록하여 이 새로운 레지스터를 리네이밍할 수 있다. 그리고, 이전에 매핑된 레지스터는, 내부의 값들이 액세스되고 있을 때 폐기될 수 있다. 이것은, 상기 이전에 매핑된 레지스터에 기억된 값들이 새로운 처리에 의해 겹쳐써지지 않고, 통상의 방식으로 이들 값을 마무리할 때 그 레지스터가 해제되므로, 타이밍 제약을 향상시킬 수도 있다. 따라서, 나머지 값들이 사용되도록 그 새로운 값을 기다릴 필요가 없다. 실제로, 본 발명의 실시예들은, 제1 데이터 요소 갱신을 위한 리네이밍을 기동하고 이후의 갱신을 위해서는 억제한다.

[0028] 일부의 실시예에서, 상기 추가의 회로소자는, 미지의 상태를 나타내는 제1 상태와 액세스되는 상기 벡터의 제1 요소가 갱신되는 것을 나타내는 적어도 하나의 또 다른 상태를 갖는, 상태 머신을 구비한다.

[0029] 상기 제1 요소가 갱신중일 때를 판정하기 위해서, 적어도 2개의 상태, 즉 하나가 제1 미지의 상태인 상태와 다른 하나가 상기 벡터의 상기 제1 요소가 액세스되는 것을 나타내는 상태를 갖는 추가의 상태 머신은 사용될 수 있다. 일부의 실시예에서, 추가의 상태 머신은, 구현에 따라 상태를 더 가질 수도 있다.

[0030] 일부의 실시예에서, 상기 데이터 처리장치는, 상기 데이터 요소가 보존되어야 하는 데이터 값이나 폐기될 수 있는 데이터 값을 현재 기억중인가를 나타내는 벡터내의 데이터 요소마다 지시하는 지시자를 기억하기 위한 데이터 스토어를 더 구비하고, 상기 데이터 요소 해저드 판정회로소자는 상기 지시자 스토어를 갱신하도록 구성된다.

[0031] 일부의 경우에, 데이터 요소들이 보존되어야 하고 폐기될 수 있는 레코드를 유지하는 것이 바람직할 수도 있다. 이 레코드는, 레지스터 리네이밍이 있는 경우 레지스터 리네이밍 회로소자에 유지되어도 되거나, 레지스터 리네이밍이 없는 경우 벡터 레지스터와 계속 관련되어도 된다. 상기 지시는, 유효 데이터나 유효하지 않은 데이터가 기억되는 것, 즉 보존될 수 있거나 폐기될 수 있는 것을 나타내는 데이터 요소마다 지시자의 형태이어도 되거나, 요소 3까지의 요소들이 보존되어야 하고 요소 3이상의 요소들이 폐기될 수 있는 것과 같은 다른 형태로 행해져도 된다.

[0032] 일부의 실시예에서, 상기 데이터 처리장치는, 벡터를 메모리에 기억하는 명령어에 따라, 폐기될 수 있는 데이터 값을 기억하는 것으로서 표시되는 파티션내의 데이터 요소마다 상기 데이터 요소에 현재 기억된 값 대신에 소정의 값을 기억하도록 구성된다.

[0033] 상기 데이터의 유효성인지 아니면 다른 것인지의 지시를 기억하는 것은, 상기 벡터가 판독될 때와 같은 경우에 유용하기도 하다. 이 경우에, 상기 레지스터로부터 폐기될 수 있는 값들을 판독하기보다는, 간단히 0이나 1과 같은 소정의 값을 판독할 수 있다. 이것은 상기 레지스터로부터 판독되는 것이 필요하지 않은 이들 데이터 값을 피하고, 이것은 보안이 쟁점이고 현재 필요로 하지 않고 있는 데이터 값들이 민감한 정보를 갖는 경우에 출력되지 않아야 하는 경우에 중요할 수도 있다.

[0034] 일부의 실시예에서, 상기 파이프라인식 프로세서는, 상기 벡터 명령어를 발행하여 실행하는 발행 회로소자 및 실행 회로소자를 구비하고, 상기 발행 회로소자는 상기 데이터 요소 해저드 검출 회로소자를 구비한다.

[0035] 상기 파이프라인식 프로세서는, 발행 회로소자와 실행 회로소자를 구비하여도 된다. 상기 발행 회로소자는, 상기 프로세서의 프론트 엔드에 있고 디코딩과 데이터 의존성 체크의 명령어들의 인출을 행한다. 이들 데이터의존성 체크를 파이프라인에서 초기에 행하는 것은, 프로시져에서 초기에 최적화 가능성이 판정되고 이에

따라 보다 최적화 할 수 있다는 것을 의미한다.

[0036] 일부의 실시예에서, 상기 데이터 처리장치는 동기화 회로소자를 구비하고, 상기 동기화 회로소자는, 상기 실행 회로소자가 현재 처리중인 데이터 요소들의 상태를 주기적으로 판정하여 상기 상태 정보를 상기 발행 회로소자에 송신하도록 구성된다.

[0037] 일부의 실시예에서, 상기 데이터 처리장치는 현재 처리중인 데이터 요소들의 상태를 주기적으로 판정하고 이 정보를 다시 상기 발행 회로소자에 보내는데 사용될 수 있는 동기화 회로소자를 구비한다. 상기 상태 머신은 어떤 이유로 그 데이터의 상태가 정확히 후속하지 않고 용이하게 그 정보를 되돌릴 수 없을지도 모른다. 이것이 긴 코드의 길이로 인해 일어나는 경우, 최적화 가능성이 가능하지 않을 것이다. 따라서, 주기적으로 기동하거나, 동기화 오류를 일으킬 가능성이 있는 특정 이벤트에 따라 기동하거나 최근에 최적화가 행해지지 않은 것을 검출하는 것에 따라 기동하는, 동기화 회로소자를 갖는 것이 바람직할 수도 있다. 이 동기화 회로소자는, 상기 요소들의 상태를 상기 실행 회로소자로부터 검출하여 상기 상태 머신을 리셋트하고 계속 최적화 시키는 상기 발행 회로소자에 되먹인다.

[0038] 일부의 실시예에서, 상기 벡터는, 벡터 연산에 의해 연산될 데이터 요소들을 식별하도록 구성된 벡터 마스크들을 포함하고, 상기 데이터 요소 해저드 판정회로소자는, 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 적어도 일부마다 존재하는지를 상기 벡터 마스크들로부터 판정하여, 존재하지 않는 경우 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키도록 구성된다.

[0039] 벡터 연산에 의해 연산될 데이터 요소들과 연산되지 않는 데이터 요소들을 식별하는 벡터 마스크들이 있을 수도 있다. 이러한 경우에, 상기 데이터 요소 해저드 검출 회로소자는, 상기 벡터들에 대해 식별된 데이터 의존성이 연산중인 벡터내의 상기 데이터 요소들의 일부만으로 인해 일어나지 않을 이들 마스크로부터 판정할 수 있다. 따라서, 이들 마스크들을 해석함으로써, 상기 데이터 요소 해저드 검출 회로소자는, 그 타이밍의 일부를 완화시킬 수도 있다.

[0040] 일부의 실시예에서, 상기 벡터는, 벡터 연산에 의해 연산될 벡터 파티션들내의 데이터 요소들을 식별하도록 구성된 벡터 마스크들을 구비하고, 상기 데이터 처리장치는, 벡터 마스크의 제어하에 벡터 파티션에 관한 하나의 연산과, 상기 벡터 마스크의 반대의 제어하에 상기 벡터 파티션에 관한 또 다른 연산을 행하도록 구성되고, 상기 상태 머신은, 상기 벡터 명령어에 의해 현재 액세스중인 상기 현재의 파티션 이후의 데이터 요소 모두와 상기 데이터 파티션내의 상기 마스킹된 데이터 요소들이 폐기될 수 있고, 그 밖의 데이터 요소 모두가 보존되는 것을 나타내는 마스킹된 액세스 현재 상태를 포함하는 적어도 하나의 또 다른 상태를 포함한다.

[0041] 데이터 의존성을 완화시킬 수 있는 경우의 일례는, 분할된 벡터내에서, 마스크의 반대가 나머지 서브세트를 선택하면서 상기 파티션의 서브세 트를 선택하는데 벡터 마스크들이 사용되는 경우가 있다. 이러한 경우에, 분할된 벡터에 사용된 3개의 상태를 갖는 것과 같은 상태 머신이 사용될 수 있지만, 상기 마스크에 의해 제어되는 상기 파티션의 일부가 폐기될 수 있고 일부분이 보존될 필요가 있는 것을 나타내는 일부의 추가의 상태를 갖는 상태 머신에서 사용될 수 있다. 이렇게 하여, 연산의 오더링은, 어떤 데이터가 겹쳐써질 수 있는지를 인식함으로써 판정될 수 있다.

[0042] 본 발명의 제2 국면은, 데이터 처리장치내에서 각각이 복수의 데이터 요소들을 포함하고 복수의 레지스터에 기억되는 벡터들에 관해 연산을 행하는 벡터 명령어들의 스트림을 처리하는 방법을 제공하고,

[0043] 레지스터 데이터 해저드가 일어나지 않게 상기 벡터 명령어들의 실행의 타이밍에 관한 제약을 판정하기 위해서 상기 벡터 명령어들의 스트림에 의해 처리되고 상기 복수의 레지스터에 기억된 상기 벡터들에 대한 데이터 의존성을 검출하는 단계로서, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 상기 명령어 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 명령어 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 명령어 스트림의 순서와 다른 순서로 생기는 경우에, 일어나는, 상기 단계;

[0044] 데이터 의존성이 식별되어 있는 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해, 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하여, 존재하지 않는 경우,

[0045] 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키는 단계를 포함한다.

[0046] 본 발명의 제3 국면은, 각각 복수의 데이터 요소를 포함하는 벡터들에 관한 연산을 행하는 복수의 벡터 명령어로 이루어진 컴퓨터 프로그램을 변환시키는 방법을 제공하고,

[0047] 상기 복수의 벡터 명령어를 해석하는 단계;

[0048] 레지스터 데이터 해저드가 일어나지 않게 상기 벡터 명령어들의 실행의 타이밍에 관한 제약을 판정하기 위해서 상기 복수의 벡터 명령어에 의해 처리되는 상기 벡터들에 대한 데이터 의존성을 검출하는 단계로서, 상기 레지스터 데이터 해저드는, 동일한 레지스터에 적어도 하나의 액세스가 기록인 2개의 액세스가, 명령어 스트림에서 앞서 일어나는 액세스가 완료되기 전에 상기 명령어 스트림에서 나중에 일어나는 액세스가 시작하도록 상기 명령어 스트림의 순서와 다른 순서로 생기는 경우에, 일어나는, 상기 단계;

[0049] 데이터 의존성이 식별되어 있는 상기 벡터들내의 상기 데이터 요소들 중 적어도 일부에 대해, 상기 벡터들에 대해 식별된 상기 데이터 의존성이 상기 데이터 요소들의 상기 적어도 일부마다 존재하는지를 판정하여, 존재하지 않는 경우,

[0050] 상기 데이터 요소를 처리하는 명령어의 실행의 타이밍에 관한 상기 판정된 제약을 완화시키는 단계;

[0051] 상기 컴퓨터 프로그램을 데이터 처리 시스템상에서 실행하는데 적합한 코드로 변환시키는 단계를 포함하는, 컴파일링 방법을 제공한다.

[0052] 본 발명의 제4 국면은, 컴퓨터상에서 실행될 때 상기 컴퓨터에 본 발명의 제3 국면에 따른 방법의 단계들을 실행시키는 컴퓨터 프로그램을 제공한다.

도면의 간단한 설명

[0053] 본 발명을, 아래의 첨부도면에 도시된 것과 같은 본 발명의 실시예들을 참조하여 예시로만 더욱 설명하겠다:

도 1은 본 발명의 실시예에 따른 데이터 처리장치의 개략도,

도 2는 본 발명의 실시예에 따른 레지스터 리네이밍 회로소자를 갖는 데이터 처리장치의 개략도,

도 3은 본 발명의 실시예에 따른 레지스터 리네이밍 회로소자의 개략도,

도 4는 본 발명의 실시예에 따른 벡터와 데이터 의존성 검출 회로소자의 개략도,

도 5는 본 발명의 실시예에 따른 상태 머신의 개략도,

도 6은 본 발명의 실시예에 따른 레지스터 리네이밍을 행할 때를 판정하기 위한 상태 머신의 개략도,

도 7은 상기 상태 머신의 코드와 대응한 상태들의 예시도,

도 8은 본 발명의 실시예에 따른 파티션들과 마스크들 모두를 갖는 코드에 대한 상태 머신의 개략도,

도 9는 본 발명의 실시예에 따른 방법의 단계들을 나타내는 흐름도,

도 10은 본 발명의 실시예에 따른 컴파일러를 갖는 데이터 처리장치를 나타내고,

도 11은 주석이 달린 명령어의 예시도다.

발명을 실시하기 위한 구체적인 내용

[0054] 도 1은 본 발명의 실시예에 따른 데이터 처리장치의 개략도다. 이 데이터 처리장치는 아주 개략적으로 도시되어 있고, 상기 회로소자와 실제 추가의 회로소자와의 사이에 추가의 상호연결이 있을 수도 있다는 것을 당업자에게는 자명하다. 도시된 데이터 처리장치에는, 발행 회로소자(10), 실행 회로소자(20) 및 라이트백 회로소자(30)를 구비하는 파이프라인식 프로세서가 있다. 발행 회로소자(10)는, 명령어 캐시(12)로부터 명령어들을 검색하여 디코딩하고, 또한 상기 벡터들의 처리로 인해 일어나는 데이터 의존성을 검출하고 그 디코딩된 명령어를 상기 실행 회로소자에 발행한다.

[0055] 따라서, 발행 회로소자(10)는, 벡터들간의 데이터 의존성을 판정하고, 여러 가지의 연산이 완료되었을 때 알게 할 수 있는 라이트백 회로소자(30)로부터 정보를 수신하는 벡터 데이터 의존성 회로소자(14)를 갖는다. 또한, 발행 회로소자(10)는, 개개의 요소들간의 데이터 의존성을 검출하고, 또한, 비록 데이터 요소들 자체가 액세스되는 방식으로 인해 벡터들간에 상기 연산이 데이터 의존성의 원인이 될지도 모르지만, 이들 데이터 의존

성이 실제로 일어나지 않아서, 실제로 특정한 순서로 실행될 필요가 있는 벡터 데이터 의존성으로부터 나타나는 벡터 연산이 다른 순서로 실행될 수 있으므로 실제로 상기 벡터 데이터 의존성 검출 회로소자(14)에서 부과한 타이밍 제약을 완화시킬 수 있는 경우를 판정하는, 데이터 요소 의존성 검출 회로소자(16)를 구비한다. 이 데이터 요소 의존성 검출 회로소자(16)는, 타이밍 제약을 완화시킬 수 있는 경우를 나타내는 명령어들에 관련된 주석에 응답한다(도 11 참조).

[0056] 또한, 데이터 처리장치(5)는, 벡터 제어 레지스터(40)와 벡터 레지스터 뱅크(50)를 구비한다. 데이터를 기억하는 메모리(60)도 있다. 상기 벡터 레지스터 뱅크(50)는, 상기 실행 회로소자(20)가 명령어 캐시(12)로부터 검색된 벡터 명령어들의 제어하에 연산을 행하는 복수의 데이터 요소들을 갖는 벡터들을 포함한다. 각 벡터 연산은, 상기 벡터들내에 제로, 하나 이상의 데이터 요소들에 관해 행해져도 된다. 상기 벡터 제어 레지스터(40)는, 어느 데이터 요소에 관해 벡터 연산을 행할지를 제어한다. 이들 벡터 제어 레지스터(40)는, 현재의 연산에 의해 요소들이 연산되는 것을 나타내는 마스크의 형태를 취하여도 되거나, 상기 벡터의 파티션이 연산되는 것을 나타내는 벡터 파티션 레지스터의 형태이어도 된다. 후자의 경우에, 벡터는, 각각을 후술하는 다수의 파티션으로 분할되어도 된다. 각 파티션은, 인접한 요소들인 하나 이상의 데이터 요소를 포함하여도 된다. 상기 벡터 제어 레지스터들은, 갱신되는 제1 데이터 요소를 추적하는데 사용되고, 레지스터 리네이밍이 적절한 경우와, 이것이 도 4 및 도 6에 대해 설명되지 않은 경우를 판정하는데 사용될 수 있는 것들에 관련된 상태 머신(19)을 갖는다.

[0057] 일부의 실시예에서, 파티셔닝은 하나보다 많은 벡터 제어 레지스터에 의해 제어되어도 되고, 이러한 경우에 특정한 벡터 연산을 위한 벡터 제어 레지스터에의 변경을 추적하는데 충분하지 않고, 상기 벡터 제어 레지스터 모두에의 변경을 추적하는 것이 필요하고, 벡터 레지스터마다 그 레지스터에 관해 최후에 분할된 벡터 연산에서 어느 벡터 제어 레지스터를 사용하였는지를 추적하는 것이 필요하다. 이것을 추적하기 위해서, 특정한 레지스터에 기록하는데 사용되었던 최후의 벡터 제어 레지스터를 나타내는 값을 기억하는 데이터 스토어를 사용하여도 된다. 따라서, 어떤 벡터 제어 레지스터의 제어하에 특정한 레지스터에의 기록에 따라, 상기 데이터 스토어는, 이것이 이 레지스터에의 이전의 기록을 제어한 벡터 제어 레지스터인 경우를 판정하기 위해 체크되고, 그러한 경우 상태 머신에서의 현재의 값은 체크될 수 있고, 그렇지 않은 경우, 그 레지스터에 대한 상태 머신이 미지의 상태로 갈 필요가 있다.

[0058] 벡터 레지스터 뱅크(50)에 관련된 유효성 테이블(70)과 상태 머신(17)도 있고, 상기 유효성 테이블(70)은, 그 레지스터내의 데이터 요소들이 유효하고 보존될 필요가 있고 폐기될 수 있는 것을 나타내며, 또 상태 머신(17)은, 현재의 파티션에 대한 파티션들이 연산중이고, 보존될 필요가 있는 데이터를 기억하고 폐기될 수 있는 데이터를 기억하는 것을 나타낸다. 이 상태 머신에 대한 추가의 상세내용은, 도 4 및 도 5의 설명에서 제공된다. 상기 유효성 테이블(70)과 상태 머신(17)용 정보는, 이들 데이터 요소와 액세스의 패턴을 액세스하는 명령어를 해석할 때 상기 데이터 요소 의존성 검출 회로소자(16)에서 제공된다. 이 정보는, 데이터 요소들이 판독될 때 폐기될 수 있는 것을 나타내는 무효인 것으로서 나타내어지는 경우에, 간단히, 실제의 기억된 값이라기보다는 임의의 경우에 무효인 소정의 값을 사용할 수 있도록 벡터를 판독할 때 사용될 수 있다. 이것은, 절전할 수 있고, 또한 비록 더 이상 유효하지 않지만 이전에 기억된 민감한 값을 포함하기도 하는 상기 데이터의 출력을 방지할 수도 있다. 따라서, 이것은 그 데이터의 보안을 보호할 수 있다.

[0059] 중요한 것은, 상기 데이터 요소 의존성 회로소자는 상기 명령어 스트림을 감시하는데 필요할 뿐이다는 것이고, 그 이유는 이것에 의해 명령어 디코드/발행에 의해 실행과 상관없이 진행될 수 있으므로 일시 중지(stall)가 일어나는 것을 피하기 때문이다. 그렇지만, 이것은, 최적화가 안전한 경우에도 행해지지 않는다는 것을 의미하기도 하는 약간의 근사를 도입한다. 이들 근사는 결코 부정확한 작용을 일으키지 않는다는 것을 주목해야 한다.

[0060] 특히, 상기 상태에 대한 정보는, VP(벡터 파티션) 또는 벡터 레지스터가 보관되고 나서 그 스택으로부터 복원될 때 발생할 가능성이 있다. 그 상태에 대한 정보를 잃어버리는 것은, 심하게 벡터 연산을 이용하지만 파티셔닝을 이용하지 않는 코드에 있어서는 아주 해롭다(즉, 상기 제어 레지스터는, 모든 벡터의 요소를 변경시키고 나서 많은 벡터 연산을 행하는 주어진 값들이다). 이것은, 그 정보를 정확한 정보로 갱신할 기회가 전혀 없거나 거의 없기 때문에, 또한 상기 정밀성의 손실이 불필요한 비용을 초래하기 때문에 해롭다.

[0061] 이를 감소시키기 위해서, 동기화 회로소자(25)는, 상기 실행 회로소자(20)에 존재하여도 되고, 이것은 필요에 따라 동기화 신호를 발행 회로소자(10)에 반대로 송신한다. 상기 실행 회로소자로부터 상기 발행 회로소자에의 반대의 이러한 또 다른 경로는 그 2개의 회로소자의 재동기에 사용된다. 상기 파이프라인이 재동기화가

발생하는 동안 일시 중지되어야 하거나 그 파이프라인이 유휴 또는 적어도 안정한(즉, 벡터 파티션 VP 레지스터를 변경하는 연산이 행해지고 있지 않다) 경우 행해질 수 있을 뿐이기 때문에 재동기화에 비용이 든다.

[0062] 이러한 재동기화는, 특별한 명령어 및/또는 데이터 값들(예를 들면, 데이터 처리 파이프라인에 의해 상기 벡터 제어 레지스터가 최적화가 허용되었을 값들을 가졌었지만 그 최적화가 적용되지 않았다는 것을 검출할 때)에 따라 주기적으로, 또는 가끔(즉, 재동기화의 비용에 따라) 행해질 수 있다.

[0063] 도 2는 도 1의 데이터 처리장치와 동일하지만 추가로 레지스터 리네이밍 회로소자(75)를 갖는 또 다른 데이터 처리장치를 나타낸 것이다. 이때, 본 발명의 실시예들은, 순차적 처리와 비순차적 처리 양쪽에서 사용될 수 있다. 이에 관해서는, 순차적 프로세서들로서 생각되는 처리장치는 명령어 스트림의 순서로 명령어들을 실행하는 것이 일반적이다. 그러나, 그 순차적 프로세서들에 의해 이들 명령어의 중복 및 병렬 실행이 실행속도를 높이기도 한다. 본 발명의 실시예들은, 벡터간의 잠재적 데이터 의존성이 실제로 일어나지 않도록 벡터에서 데이터 요소들을 액세스하는 경우에 상기 병렬성을 향상시키는데 사용될 수 있다.

[0064] 레지스터 리네이밍 회로소자(75)는, 비순차적 프로세서에 존재하여도 되고, 그 레지스터 뱅크에 있는 물리 레지스터의 수가 명령어 세트가 리네밍할 수 있는 레지스터의 수보다 큰 경우에 사용된다. 따라서, 레지스터 뱅크에서의 특정한 레지스터는, 상기 레지스터 리네이밍 회로소자(75)에 의해 하나의 명령어에 지정된 레지스터에 매핑되고, 그 매핑은 리네이밍 테이블(72)에 기억된다. 본 실시예에서도, 상기 레지스터 리네이밍 회로소자는, 벡터에서의 요소들이 미정의된 정보를 기억하는 유효성 테이블(70)을 포함한다.

[0065] 레지스터 리네이밍 회로소자(75)는, 도 3에 보다 상세히 도시되어 있다. 레지스터 리네이밍 회로소자(75)는, 현재 사용된 레지스터의 명령어 식별자들을 그 레지스터 뱅크(50)에서 그들의 물리적 위치에 매핑하는 리네이밍 테이블(72)을 포함한다. 또한, 본 실시예에서, 레지스터 리네이밍 회로소자(75)는, 레지스터내에 상기 데이터 요소 각각의 유효성을 나타내는 유효성 테이블(70)을 갖는다. 이는 도 1의 유효성 테이블과 같지만, 레지스터 리네이밍 회로소자일 경우, 레지스터 리네이밍이 일어나는 것과 같은 시간에 얻어지므로 그 유효성 테이블에 상기 유효성 정보를 기억하기가 편리하다.

[0066] 본 발명의 실시예들은, 리네이밍을 사용하는 데이터 처리장치의 효율성을 거의 향상시킬 수 있다. 레지스터 리네이밍은, 일 레지스터에 기억된 벡터내의 데이터 요소들에 관한 연산이 일반적으로 리네이밍될 새로운 레지스터와, 이 연산에 의해 연산되지 않고 그 새롭게 리네이밍된 레지스터와 통합된 이전에 리네이밍된 레지스터에 기억된 데이터 요소들을 필요로 하므로, 벡터 처리동안에 매우 비싸진다. 이후의 데이터 요소들이 폐기될 수 있다는 것을 인식함으로써 이러한 데이터의 리네이밍과 복사는 억제될 수 있고 실질적으로 전력 및 시간이 절감될 수 있다. 실제로, 몇몇의 연산이 특정한 레지스터를 갱신하는 경우, 한번만 리네이밍될 필요가 있고, 이후의 벡터 연산은 간단히 동일한 레지스터에 액세스할 수 있고, 폐기될 수 있는 것으로서 인식되는 상기 요소들을 갱신할 수 있다. 또한, 벡터에서 제1 요소가 액세스되는 경우에, 리네이밍이 장려될 수 있고 새로운 레지스터는 상기 제1 요소의 그 새로운 갱신을 위해 매핑될 수 있다. 그리고, 이 레지스터는, 또 다른 요소의 갱신에 사용되고, 이전에 매핑된 레지스터는 겹쳐써지지 않을 것이고, 이는 이러한 이전의 레지스터가 판독동작에 사용되는 중일 경우 타이밍 제약을 향상시킬 수 있다.

[0067] 도 4는 벡터 데이터 의존성 회로소자와 요소 의존성 회로소자를 보다 상세히 나타낸 것이다. 벡터 데이터 의존성 회로소자(14)는, 발행 스테이지로부터 또한 라이트백 스테이지로부터 정보를 수신하고, 이들 벡터 데이터 의존성으로부터 판정한다. 그 후, 이들은, 본 실시예에서 벡터 레지스터 뱅크(50)의 벡터들에 관련된 상태 머신(17)과 상기 벡터 제어 레지스터(40)에 관련된 상태 머신(19)의 형태인, 상기 데이터 요소 의존성 회로소자(16)에 의해 판정된 의존성들을 감안하여 수정된다. 본 실시예에서 상태 머신(17)은, 3개의 상태, 즉 미지의 상태UN, 현재의 파티션에의 기록상태WC 및 이전의 파티션에의 기록상태WP를 갖는다. 이들을 도 5에 대해 보다 상세히 설명한다.

[0068] 동작상 벡터 연산이 해석될 때, 상기 상태 머신은 현재의 연산에 관한 상태로 갱신되고, 이것은 상기 판정된 벡터 데이터 의존성에 대한 최적화가 가능한지 아닌지를 판정하는데 사용될 수 있다. 따라서, 상기 상태 머신에서의 값은, 현재의 파티션에 대한 파티션들이 연산중이고 보존될 필요가 있는 데이터를 기억하고 폐기될 수 있는 데이터를 기억하는 것을 나타낸다. 이러한 정보는, 벡터 연산간에 데이터 해저드를 피하기 위해 일어나는 타이밍 제약이, 개개의 요소들을 고려할 때 이들 해저드가 일어나지 않을 것이기 때문에 필요하지 않은 경우를 판정하는데 사용될 수 있다.

[0069] 본 실시예에서는, 벡터 제어 레지스터(40)에 관련된 상태 머신(19)도 있다. 이것을 이용하여, 갱신되는

제1 데이터 요소를 추적할 수 있고, 레지스터 리네이밍이 적절하고 이것이 도 6에 대해 설명하지 않은 경우를 판정할 수 있다.

[0070] 벡터들을 갖는 파티션을 이용하여, 초기의 파티션P0, 즉 벡터 레지스터A에 약간의 값들을 계산하는데 사용된 용도를 생성하는 것이 일반적일 경우, P0를 사용하여 P0와 겹치지 않고 벡터 레지스터에 통합된 일부의 또 다른 값들을 계산하는데 사용되는 후계자 파티션P1을 생성하고, 충분한 요소가 처리될 때까지 어떠한 초기의 파티션과 겹치지 않는 또 다른 파티션들을 계속 생성한다.

[0071] 파티셔닝은 다양한 방식으로 행해질 수 있지만, 일 방식은 '벡터 파티션' 레지스터(VP)가 현재의 파티션을 식별하는 한 쌍의 요소 지수값들을 보유하기 위한 방식이다. 예를 들면, VP가 그 한 쌍[0,3]을 포함하는 경우, 요소 위치 0, 1, 2 및 3은 현재의 파티션에 있다. 파티셔닝이 VPCLR(VP 레지스터를 '클리어'함)과 VPNXT(VP를 다음의 파티션으로 변경함)일 때 2개의 키 조작을 사용하였다.

[0072] VPCLR은 상기 VP 레지스터를 액티브 요소[0,0]를 포함하지 않는 초기의 파티션에 설정한다(이것은 일례일 뿐이고 다른 설계는 VP를 [0,7]에 설정할지도 모른다).

[0073] 상기 VPNXT 연산은, 현재의 파티션 후 일어나는 요소들의 시퀀스인 다음 파티션을 식별하기 위해 VP를 변경한다. 예를 들면, VP의 초기값이 [0,0]이면, VPNXT를 한번 사용하면 VP를 [0,2]로 변경하고, VPNXT를 두 번 사용하면 VP를 [3,5]를 변경하고, VPNXT를 세 번 사용하면 VP를 [6,7]로 변경할지도 모른다. 여기서 알 수 있듯이, 파티션은 낮은 수로 번호가 매겨진 요소로부터 보다 높은 수로 번호가 매겨진 요소들로 이동하고, VPNXT에서 생성한 연속적인 파티션은 서로 겹치지 않는다. (파티션들의 정확한 수와 시퀀스는 데이터 의존적인 경우가 있다)

[0074] 상기 VP 레지스터의 값은, 그 밖의 벡터 연산(이를테면, VADD, VMUL, VLOAD 등)은 액티브 요소들만이 변경되어도 된다는 점에서 적용되는 요소들에 영향을 미친다. 추가로, 보다 낮은 수로 번호가 매겨진 요소들의 값들은 변경되지 않고, 보다 높은 수로 번호가 매겨진 요소들의 값들은 나중에 변경되어도 되고 그들의 값이 겹쳐써질 수 있고 보존될 필요가 없다는 점에서 미정의되어 있다.

[0075] 아래에는 일부의 코드의 일례를 나타내고 있다:

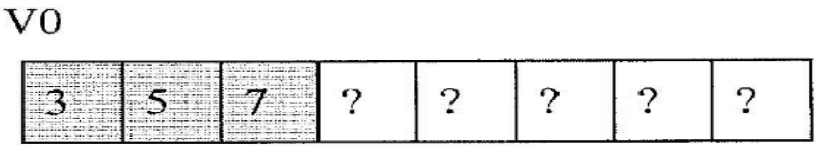
[0076] VPCLR

[0077] ; VP==[0..0]

[0078] VPNXT V5 ; V5에서의 정보를 사용하여 다음 파티션을 판정한다(설명을 위해 사용된 0..2)

[0079] ; VP=[0..2]

[0080] VADD V0,V1,V2; V1[0..2]의 요소들을 V2[0..2]에 가산하여, V0[0..2]가 된다

[0081] 

[0082] VPNXT V5 ; V5에서의 정보를 사용하여 다음 파티션을 판정한다(설명을 위해 사용된 3..5)

[0083] ; VP==[3..5]

[0084] VSUB V0,V4,V6 ; V4[3..5]의 요소들을 V6[3..5]에 감산하여, V0[3..5]가 되어, V0[0..2]를 보존한다

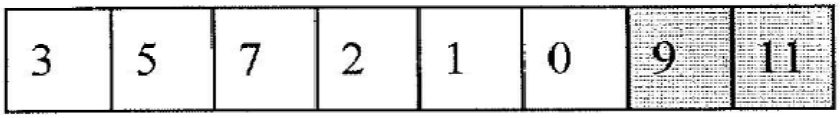
[0085] 

[0086] VPNXT V5 ; V5에서의 정보를 사용하여 다음 파티션을 판정한다(설명을 위해 사용된 6..7)

[0087] ; VP=[6..7]

[0088] VADD V0,V8,V9 ; V8[6..7]의 요소들을 V9[6..7]에 가산하여, V0[6..7]가 된다

V0



[0089]

[0090] 상기 코드 예에서 관찰하는 2개의 키 상황은 아래에 있다:

[0091] 1) 각 벡터 연산은 이전의 연산으로부터 다른(비중복) 세트의 요소들에 기록한다. 이 특성은, 상기 파티션들의 위치와 상관없고, 하나의 벡터 연산이 행해질 때마다 참이고, VPNXT는 다음 파티션에 이동하는데 사용되고 나서 또 다른 벡터 연산이 행해진다. 이러한 관찰은, 상기 레지스터 레벨에서 RAW 해저드가 있지만, 상기 요소 레벨에서 RAW 해저드가 없다는 것을 의미한다.

[0092] 2) 현재의 파티션 이후의 요소들은, 미정의되고(?로 도시됨) 다음의 VPNXT와 이후의 벡터 연산 후까지 상기 프로그램에 의해 어떠한 특정한 값도 갖는데 좌우될 수 없다. 이것은 상기 요소 레벨에서 잘못된 WAR 해저드를 제공한다.

[0093] 따라서, 다수의 최적화가 적용될 수 있다:

[0094] 1) 원리상, V0에 기록하는 3개의 모든 벡터 연산은, 실제의 해저드 또는 그들간의 의존성이 없으므로 병렬로 또는 다른 순서로 실행할 수 있었다. 필요한 것 모두는, 그 결과를 단일의 결과 레지스터에 통합할 것이다.

[0095] 2) 병렬로 또는 비순차적 순서로 동일한 목적지 레지스터를 포함하는 연산들을 실행할 때, 통상, 정확한 예외를 지원하거나 일시 중지를 일으켜 WAW 해저드를 핸들링하기 위해서 레지스터 리네이밍을 사용하는 것이 필요하다. 파티션에서의 초기에 기록이 나중의 요소에 미정의된 채로 있으므로, 레지스터 리네이밍 또는 일시 중지의 사용은, 본 예시에서는 필요하지 않다: 모든 연산은 단일의 물리 레지스터에 기록할 수 있다.

[0096] 이러한 최적화를 구현할 수 있는 방법을 설명하기 전에, 우리는 이들 최적화의 가능성을 검출할 필요가 있다. 이는, 예를 들면 도 4에 도시된 것과 같이, 상태 머신(17)을 사용하여 상기 명령어 스트림을 감시하여서 행해질 수 있다.

[0097] 이러한 상태 머신의 상태들이, 도 5에 도시되어 있다. 이들은 상기 도시된 것과 같은 코드를 처리할 때 프로세서가 가지는 상기 검출된 상태들이 후속한다. 상기 프로세서가 초기에 미지의 상태에 있고, 안전한 이 상태에서, 최적화를 적용할 수 없다. 상기 도시된 VADD와 같은 분할된 벡터 연산이 행해질 때, 상기 목적지 레지스터 Vd에 기록되고, 상기 프로세서는 현재의 파티션 상태에의 상기 기록인 새로운 "WC" 상태에 이동한다. 이 상태에서, 현재의 파티션들과 이전의 파티션들의 값들은, 보존되어야 하지만 나중의 파티션들은 미정의되고 겹쳐써져도 된다. 임의의 또 다른 분할된 벡터 연산이 행해짐으로써 상기 프로세서가 이 상태로 유지하지만, VPNXT명령어는 상기 프로세서가 다음 파티션으로 이동하고, 상기 프로세서가 반대로 WC상태로 이동하는 분할된 벡터 연산이 행해지기 전에, 상기 프로세서는 현재의 파티션에서와 이후의 파티션에서의 요소 모두가 미정의되는 이전의 파티션 상태에 의해 기록된 것인 "WP" 상태에 있다. 이 상태는, 미정의된 값들이 겹쳐써질 뿐이고 이러한 데이터 해저드가 일어나지 않으므로, 상기 이전의 기록과 조합하여서 최적화될 수 있다. 따라서, 이 상태의 인식은, 최적화가 행해질 수 있는 트리거다.

[0098] 새로운 벡터 파티션이 수신될 경우 상기 미지의 상태로 전환된다.

[0099] 이 상태 머신에 더하여, 또 다른 상태 머신(도 4에서 19)은 일부 최적화에 도움이 될 수도 있다. 이러한 추가의 상태 머신은, 갱신되는 제1 데이터 요소를 추적하는데 사용될 수 있고, 레지스터 리네이밍이 적절하고 그렇지 않은 경우를 판정하는데 사용될 수 있다. 따라서, 도 6으로부터 알 수 있듯이, 상기 상태는, 미지의 VPUN상태로서 시작한다. VPCLR 연산의 접수는 그 상태를 VPZERO로 이동하고 여기서 하나는 초기의 파티션 앞이고 요소들은 액티브가 아니다. VPNXT는 제1 파티션으로 상태를 이동시키고, 이것은 레지스터 리네이밍이 행해져야 하는지 아닌지를 판정할 때 중요하다. 본 발명의 실시예에서, 레지스터 리네이밍은, 상기 상태가 VPFIRST로서 식별될 때 목적지 레지스터를 위해 구현된다. 상기 상태 머신(19)과 상기 WP상태에 의해 식별된 동일한 목적지 레지스터의 어떠한 이후의 연산도 레지스터 리네이밍을 억제시켜야 한다.

[0100] 상기 파티션을 변경하는 어떠한 또 다른 연산도 미지의 상태로 취하고, 상기 VPFIRST 상태는 VPCLR 후에 VPFIRST에 따라 다시 도달될 뿐이다.

[0101] 도 7은 도 4의 상태 머신의 코드의 시퀀스의 예시 및 상태 머신(17, 19)의 대응한 상태를 나타낸다. 따라서, 이 경우에, 양쪽의 상태 머신은 미지의 상태에서 시작하고, 상태 머신(19)은 VPCLR 명령어에 따라 VPZERO 상태에 이동하고, VPNXT 명령어에 따라 VPFIRST에 이동하고, 이 시점에서 상기 목적지 레지스터는 레지스터 리네이밍 회로소자를 사용하여 리네이밍된다. 상태 머신(17)은, 벡터 연산 VADD가 제1 파티션에 관해 행해질 때까지 상기 미지의 상태에 머무르는 한편, 현재의 파티션 또는 WC상태에 기록이 들어간다. 이 시점에서, 상태 머신(19)은, 상기 미지의 상태에 들어간다. 다음의 파티션을 나타내는 VPNXTV5 명령어에 따라, 상태 머신(17)은 WP 상태에 들어가고, 다음 연산은 최적화되고 이전의 연산과 병렬로 행해질 수 있고, 레지스터 리네이밍은 이 연산에 대해 동일한 목적지 레지스터에 기록되도록 억제된다.

[0102] 요약하면, 본 발명의 실시예들은, WAW 요소 해저드가 없는 경우(즉, 상기 목적지 레지스터의 상태 머신이, WAW 해저드가 없는, WP상태를 나타내는 경우)를 판정하고, 이 시점에서 프레쉬 물리 레지스터의 할당을 억제하고(레지스터 리네이밍) 새로운 분할된 벡터 연산에 의해 현재의 물리 레지스터에 기록하게 하는 것이 안전하다. 이에 따라 현재의 파티션의 외부의 요소들을 하나의 물리 레지스터로부터 다른 물리 레지스터에 반복적으로 복사시키지 않게 하고, 동일한 레지스터에 관해 분할된 연속적인 연산을 직렬화하는 문제점을 피한다. 디코드 및 발행 로직에서 변경시키는 것에 더하여, 이러한 최적화는 벡터 레지스터 파일에서의 변경을 필요로 한다: 벡터 레지스터의 요소 모두에 기록하는 대신에, 연산은 현재의 파티션에서의 그들의 요소들에 기록할 뿐이다. 이를 달성하기 위해서, 벡터 레지스터 파일은, 요소 3,4,5(예를 들면)에 기록하기 위해서만 선택할 수 있도록 벡터 레지스터의 요소마다 "기록 선택" 신호를 사용하여도 된다. 또한, 이것은, 기록할 어느 요소가 라이트백 스테이지에 역으로 전파되어야 하는지를 선택하는 신호들을 제어하는 것이 필요하다.

[0103] RAW 요소 해저드가 없는 경우(즉, 상기 VP 레지스터에 관한 상태 머신은 현재의 파티션이 제1 요소를 포함하는 것을 나타내는 경우), 프레쉬 물리 레지스터를 할당(레지스터 리네이밍을 기동)하는 것이 필요하지만 상기 이전의 물리 레지스터로부터 어떠한 요소들을 복사하는 것이 필요하지 않다. 이것은, 벡터 연산간의 의존성이 추가하는 문제점의 일부를 제거한다.

[0104] 이들 최적화는, 프레쉬 물리 레지스터를 할당하고, 레지스터 네임들을 물리 레지스터에 변환하고 하나의 레지스터의 이전의 값을 판독하도록 구성되는 처리들에 영향을 미쳐서, 이들의 처리를 행하는 상기 명령어 디코드 및 발행 로직에서 가장 잘 행해진다.

[0105] 이 기술은, 모든 복사 및 모든 의존성을 제거하지 않고: 상기 복사 또는 의존성이 필요한 경우도 있고, 상기 기술이 필요하지 않는 것을 검출할 수 없는 경우도 있다. 복사가 필요한 경우, 2가지 방식 중 하나로 행해질 수 있다.

[0106] 하나의 접근법은, 상기 실행 로직이 분할된 벡터 연산의 2개의 변형예: 통합을 행하는 것과 통합을 행하지 않는 최적화된 형태를 지원하기 위한 방법이다. 상기 명령어 발행 로직은, 상기 상태 머신의 값들에 의거하여 상기 적절한 변형예를 선택할 수 있다.

[0107] 또 다른 접근법은, 상기 실행 로직이 분할된 벡터 연산의 하나의 변형예만을 지원하지만 단일의 벡터에 2개의 벡터의 요소들을 통합할 수 있는 추가의 '통합' 연산을 포함하기 위한 방법이다. 상기 기술에 의해 WAW 해저드가 없거나 RAW 해저드가 없는 것을 검출하는 경우, 상기 분할된 벡터 연산만이 발행되는 것이 필요하다. 그 밖의 모든 경우에 있어서, 상기 디코드/발행 로직은, 추가로 이러한 '통합' 연산을 발행하여 상기 레지스터의 이전의 값을 판독하여 새로운 결과와 통합하고, 그것을 역으로 새로운 물리 레지스터 파일에 기록할 필요가 있다.

[0108] 도 8은 마스크들을 갖는 프로세서의 상태가 파티션들에 어떻게 사용되었는지를 나타낸 것이다. 특정한 파티션에 관해 연산을 행하는 것에 더하여, 상기 프로세서는 그렇게 상기 연산이 행해져야 하는 파티션내의 요소들을 식별하는 마스크의 제어하에서 한다는 것일지도 모른다. 예를 들면, 벡터 비교 연산을 사용하여 벡터 마스크를 세트하고, 그 마스크를 사용하여 벡터 가산을 행하여 벡터 레지스터 V0에 기록하고 나서, 그 마스크의 반대를 이용하여 벡터 감산을 행하여 벡터 레지스터V0에 기록할지도 모른다. 마스킹 정의 방법에 따라, 이것은 단일 벡터에의 벡터 가산 및 감산의 결과를 통합하여도 된다. 벡터 파티셔닝과 같이, 벡터 마스킹은 단순한 상태 머신을 사용하여 감시될 수 있는 특정 공통 모드 시퀀스를 포함한다. 특히, 벡터 파티셔닝과 조합될 경우, 우리는 WC상태(보다 위에)를 몇 개의 서브상태로 분할할 수 있다.

[0109] 도 8은 추가의 2개의 상태 WC+ 및 WC-가 마스크들의 사용으로 인해 생길 때 일어나는 도 5의 상태 머신의 확대도다. 이 경우에, 제1 연산이 상기 마스크에서 선택한 요소들에 관해 행해지고 이것은 현재의 파티션의 요소들 중 일부가 미정의되는 WC+ 경우이고, 그리고, VMNOT 연산은, 그 밖의 요소들을 선택하도록 행해지고 나서, 이들은 연산을 위해 선택되지만 그 연산이 행해지기 전에 그들이 액티브(선택)되지만 아직 정의(갱신)되어 있지 않다. 일단 상기 연산이 행해졌다면 상기 프로세서는, WC상태에 들어가고, VPNXT에 따라 다음의 연산이 이전의 연산과 병렬로 행해져서 최적화될 수 있는 것을 나타내는 WP상태에 들어간다.

[0110] 도 9는 본 발명의 실시예에 따른 방법에서의 단계들을 나타내는 흐름도다. 본 방법에서, 데이터 의존성은 벡터 명령어의 스트림에 의해 처리되는 벡터들내에서 검출되고 그 벡터 명령어들의 실행의 타이밍에 관한 제약이 판정된다. 그후, 식별된 데이터 의존성이 상기 데이터 요소들에 대해 존재하는지와 상기 타이밍 제약이 이들 데이터 요소들의 처리에 완화될 수 없는지 판정된다. 이 방법은, 컴퓨터 프로그램에 따라 컴퓨터에 의해 행해져도 되고, 상기 벡터 명령어들의 스트림을 컴파일링중인 컴파일러에서 행해져도 된다. 이에 관해서, 이들 명령어의 처리중에 최적화를 판정하는 것이라기보다는, 그들은 컴파일 시간에 판정된다. 상기 컴파일러는, 시스템 외부에 있어도 되거나, 도 10에 도시된 것과 같이 처리장치 내에 있어도 된다.

[0111] 도 10은 런타임 컴파일러(90)를 사용하여 명령어 캐시(12)로부터 수신된 코드를 컴파일링하는 경우의 데이터 처리장치를 나타낸 것이다. 상기 컴파일러는, 그 코드를 해석하고, 데이터 해저드를 확실히 없애기 위해 상기 벡터 데이터 의존성과 명령어의 실행의 타이밍에 관한 상기 연관된 제약을 판정한다. 그리고, 컴파일러는 개개의 데이터 요소 데이터 의존성을 보고, 이전에 판정된 상기 타이밍 제약이 완화될 수 있는 경우를 판정한다. 그리고, 이렇게 최적화된 상기 컴파일링된 코드는, 발행 스테이지(10)에 보내지고 상기 파이프라인을 통해 진행된다. 이때, 상기 컴파일러가 여기서는 런타임 컴파일러로서 도시되어 있지만, 코드를 처리하기에 앞서 상기 코드를 컴파일링하는데 사용된 원격 컴파일러이어도 된다.

[0112] 요약하면, 본 발명의 실시예들에 의해, 해저드가 존재하지 않는 경우를 식별하고 해저드가 존재할 때의 작용을 억제하는 것에 의거하여 최적화할 수 있다. 비순차적 프로세서에서는, 필요 이상의 레지스터 리네이밍을 억제함에 따라서, 필요 이상의 레지스터를 판독한다. 또한, 필요 이상의 파이프라인 일시 중지가 억제된다.

[0113] 부분 순차 방식으로(예를 들면, 요소 모두가 처리될 때까지 한번에 2개의 요소) 벡터 연산을 행하는 프로세서에서는, 동일한 벡터에 기록하는 다수의 연산을 병렬로 진행시킬 수 있다.

[0114] 본 발명의 실시예들에서는, 상기 명령어 디코더내에서, 벡터 파티션들을 계산하는 명령어들과 벡터 파티션들에 의해 변경된 명령어들을 찾는 상기 명령어 스트림을 감시하는 감시회로를 이용한다. 이 감시회로는, 작고 간단한 상태 머신을 사용하여, 최적화가 가능하고 안전하며 상기 디코드/발행 로직을 다르게 작동시키는 경우들을 식별하기 위해서 각 벡터 레지스터와 벡터 제어 레지스터들의 상태를 추적한다.

[0115] 비순차적 프로세서는, 레지스터 리네이밍을 종종 사용하여 가짜 레지스터 의존성을 제거하고 정밀한 예외를 지원한다. 레지스터 리네이밍은 명령어들에 사용된 레지스터명들을 물리 레지스터명으로의 변환을 확립한다. 각 명령어가 발행을 위해 디코딩되어 준비되므로, 미사용 물리 레지스터들은 상기 명령어의 결과를 보유하도록 할당되고, 그 입력 레지스터명들은 그들의 현재의 물리적 매핑으로 변환된다. 이것은 스칼라 명령어들에 대해서는 잘되지만, 벡터 분할된 벡터 연산에 사용될 때 아래의 2종류의 비효율성을 일으킨다:

[0116] 1. 분할된 벡터 연산이 상기 목적지 레지스터의 이전의 값과 새롭게 계산된 현재의 파티션의 값들을 통합하므로, 이 레지스터에 이전에 매핑된 물리 레지스터로부터 상기 이전의 값을 판독하는 것이 필요하다. 이는, 판독 포트가 추가로 필요하고, 벡터 연산이 부분 순차 실행을 하면 상당한 시간이 걸릴 수도 있다. 이러한 복사처리는, 요소마다 여러번 반복되는 경우도 있다: 벡터가 N개의 별개의 파티션으로 분할되면, 제1 파티션의 요소들은 N-1번 복사될 것이다.

[0117] 2. 그것은 동일한 레지스터에 관한 연산끼리 추가의 의존성을 야기한다: 그들이 요소들의 분리 집합에 관해 연산하는 경우에도, 제2 연산은 동일한 레지스터에 관한 제1 연산이 그 결과를 생성한 후까지 완료할 수 없다. 이 직렬화는 분할된 벡터 연산을 사용하여 프로그램에서의 병렬 가능성을 감소시킨다.

[0118] 이들 문제점에 대한 우리의 해결책은, 상술한 기술을 사용하여 요소 해저드가 없는 상황을 식별하고 그 기술을 사용하여 이들 비효율성 각각을 억제하는 것이다.

[0119] 파이프라인식 순차적 프로세서는, 독립 명령어들의 실행을 중복하려고 하고 해저드 검출 로직(연동장치라고도 알려짐)을 이용하여 종속 명령어들을 검출하고 상기 제1 명령어의 결과가 이용가능할 때까지 제2 종속

명령어의 발행을 일시 중지시킨다.

[0120] 상술한 것처럼, 분할된 벡터 연산은, 본 기술을 사용하지 않고 명령어 발행을 일시 중지시키는 레지스터 해저드로부터 고통을 받는 경우도 있다. 본 기술을 파이프라인식 순차적 프로세서에 가산하는 것에 의해 상기 해저드 검출로직이 요소 해저드가 없는 상황을 식별하므로 불필요한 일시 중지가 일어나는 것을 피할 수 있다.

[0121] 비순차적 프로세서와 같이, 상기 추가의 요구사항은, 요소가 기록되도록 제어하고 VP 레지스터로부터의 신호들을 라이트백 스테이지에 역으로 전파하게 배치되도록, 기록 선택 신호들에 의해 벡터 레지스터 파일을 변경하는 것이다.

[0122] 도 11은 다른 명령어간의 데이터 의존성이 존재하지 않는 것을 나타내기 위해 주석이 달려진 명령어들의 예시도다. 이러한 예시는, 주석이 달리지 않은 변형예에 존재하는 일부의 데이터 의존성의 부재를 나타내기 위해서, 각각 '_first', '_partial' 및 '_last'로 주석이 달린 8개의 명령어 중 첫 번째 2개의 명령어와 최후 2개의 명령어를 나타낸 것이다. 그들의 주석이 달리지 않은 변형예는, 벡터 레지스터 V0의 요소번호 0을 스칼라 레지스터 R0의 내용으로, V0의 요소번호 1을 스칼라 레지스터 R1의 내용으로,... 및 V0의 요소번호 7을 스칼라 레지스터 R7의 내용으로 설정한다. 또한, 그들의 주석이 달리지 않은 변형예는, 상기 명령어들을 순차로 실행시키는 기록 후 기록(WAW) 해저드가 생기게 될 것이다. 또한, 주석이 달리 변형예는 대응한 요소 번호를 스칼라 레지스터의 내용으로 설정하지만 이들 명령어가 데이터 요소 해저드를 야기하지 않아서 그 명령어들이 병렬로 또는 비순차로 실행할 수 있는 것을 나타낸다.

[0123] 아래의 레지스터 리네이밍을 사용하는 프로세서에서:

[0124] - Vset_first 명령어는, 벡터 레지스터 V0에 매핑된 프레쉬 물리 레지스터P를 할당되게 하지만 V0에 이전에 매핑된 물리 레지스터의 내용이 P에 복사될 필요가 없는 것을 나타낸다.

[0125] - Vset_partial 명령어는, V0에 매핑된 물리 레지스터P의 적절한 요소를 갱신한다(이 명령어는, 프레쉬 물리 레지스터의 할당이 필요하지 않고, 상기 레지스터의 이전의 내용을 새로운 요소값과 통합한다).

[0126] - Vset_last 명령어는, V0에 매핑된 물리 레지스터P의 최후 요소를 갱신하고(또, 이 명령어는, 프레쉬 물리 레지스터의 할당 또는, 오래된 값들의 통합을 필요로 하지 않는다) 상기 레지스터가 지금 완전히 갱신된 것을 나타낸다.

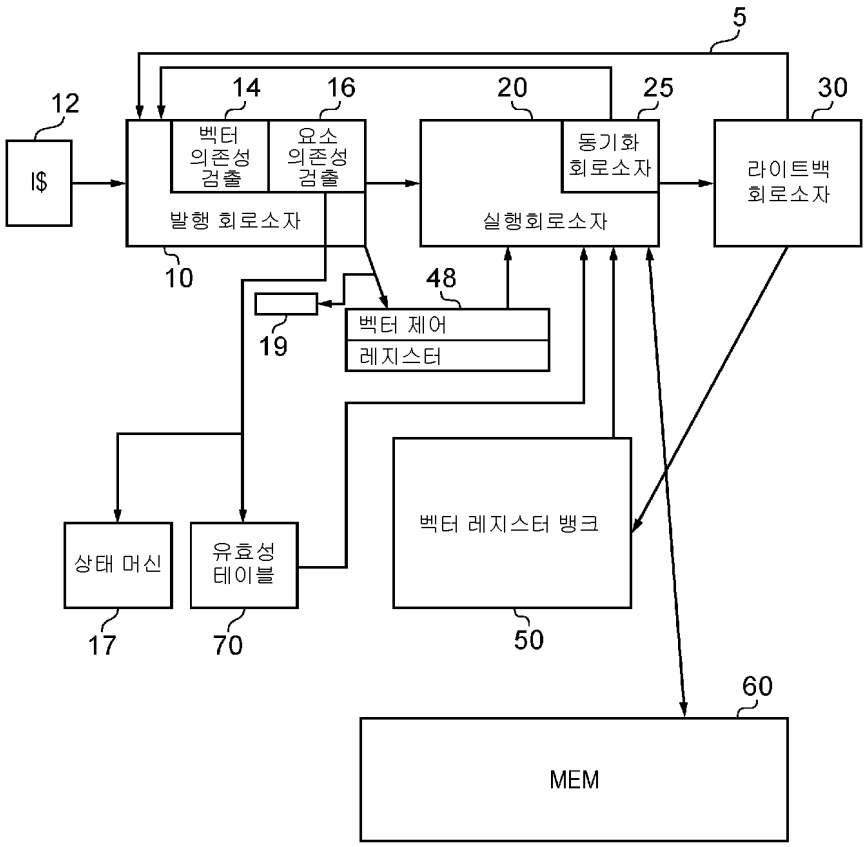
[0127] 순차적 프로세서에서는, 상기 Vset_first 명령어는 V0를 사용하여 이전의 명령어 후에 실행해야 하고, Vset_last 명령어는 V0를 사용하여 이후의 명령어 이전에 실행해야 하지만 Vset_partial 명령어는, Vset_first 명령어, 다른 Vset_partial 명령어 및 Vset_last 명령어의 순서로 임의의 명령어와 병렬로 실행할 수 있다.

[0128] '_first' 주석은 요소 번호 0으로 설정할 때 사용되고, '_partial' 주석은 벡터에 있어서 최저로 번호가 매겨진 요소도 최고로 번호가 매겨진 요소도 아닌 요소 1 내지 6에 사용되고, '_last' 주석은 요소 7에 사용된다. 이에 따라, _first, _partial 및 _final간의 차이가 요소 번호로 판정되는지를 효율적인 명령어로 인코딩할 수 있다.

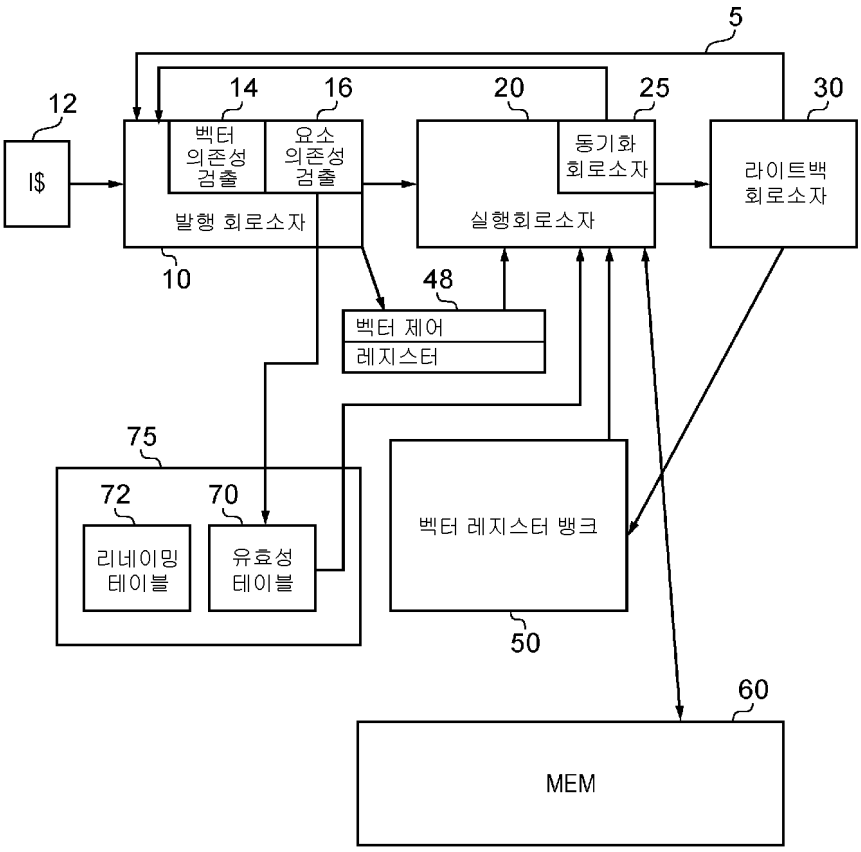
[0129] 본 발명의 여러 가지 또 다른 국면 및 특징은 첨부된 청구항에 기재되어 있다. 본 명세서에서는 본 발명의 범위를 벗어나지 않고 이전에 설명한 실시예들을 여러 가지로 변형할 수 있다.

도면

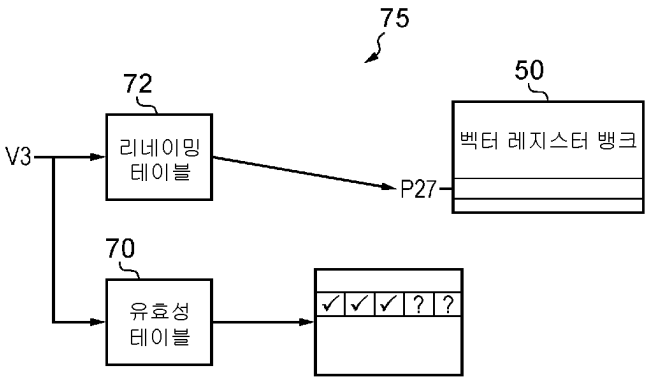
도면1


5
12
14
16
20
25
30
I$
벡터
의존성
검출
요소
의존성
검출
동기화
회로소자
라이트백
회로소자
발행 회로소자
실행회로소자
10
48
19
벡터 제어
레지스터
상태 머신
유효성
테이블
벡터 레지스터 뱅크
17
70
50
60
MEM

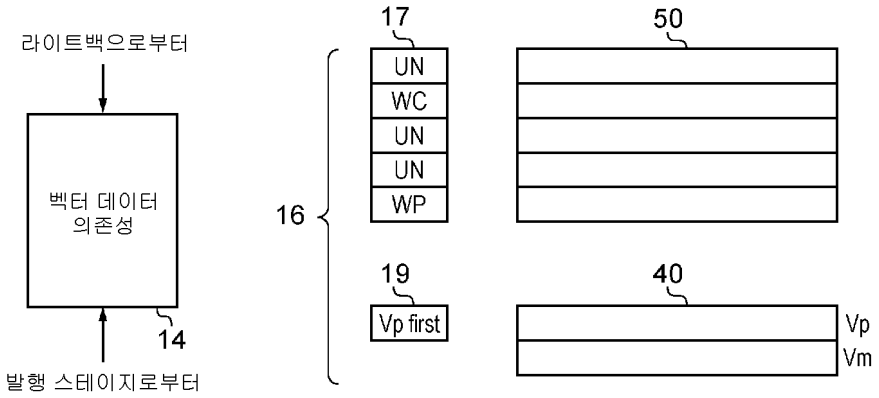
도면2


5
12
I$
14
16
벡터
의존성
검출
요소
의존성
검출
발행 회로소자
10
20
25
동기화
회로소자
실행회로소자
30
라이트백
회로소자
48
벡터 제어
레지스터
75
72
70
리네이밍
테이블
유효성
테이블
벡터 레지스터 뱅크
50
60
MEM

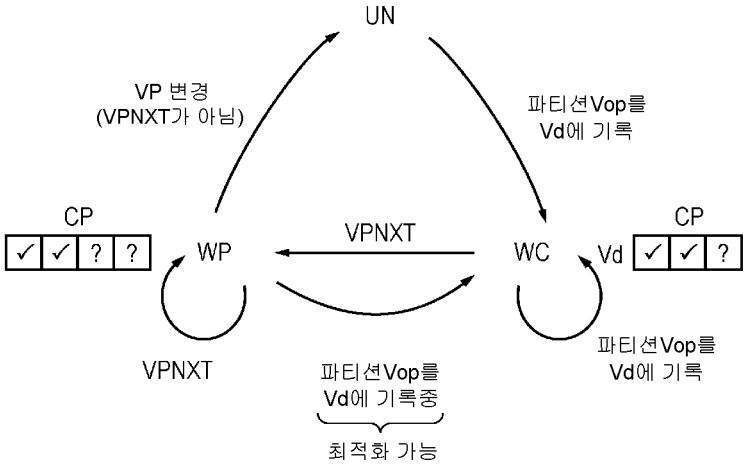
도면3


75
72
50
V3
리네이밍
테이블
벡터 레지스터 뱅크
P27
70
유효성
테이블
✓ ✓ ✓ ? ?

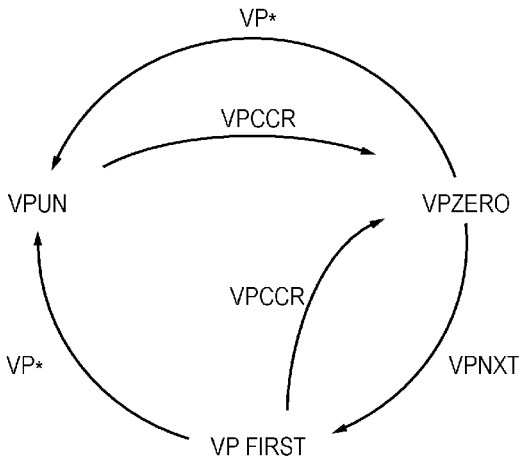
도면4


라이트백으로부터
벡터 데이터
의존성
14
발행 스테이지로부터
16
17
UN
WC
UN
UN
WP
50
19
Vp first
40
Vp
Vm

도면5


UN
VP 변경
(VPNXT가 아님)
파티션Vop를
Vd에 기록
CP
WP
VPNXT
WC
Vd
CP
VPNXT
파티션Vop를
Vd에 기록중
파티션Vop를
Vd에 기록
최적화 가능

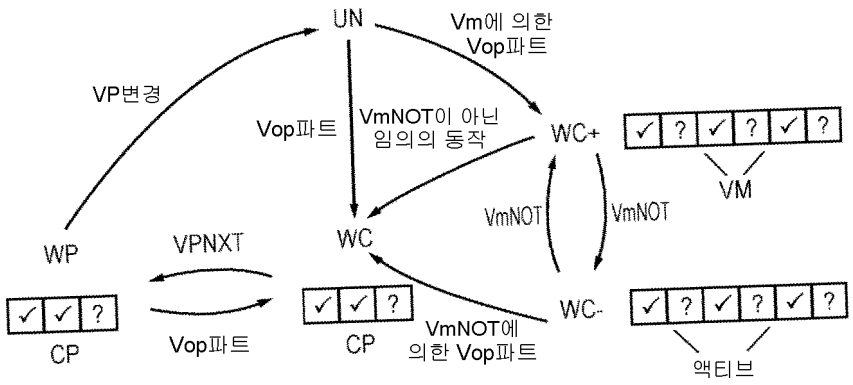
도면6


VP*
VPCCR
VPUN
VPZERO
VPCCR
VP*
VPNXT
VP FIRST

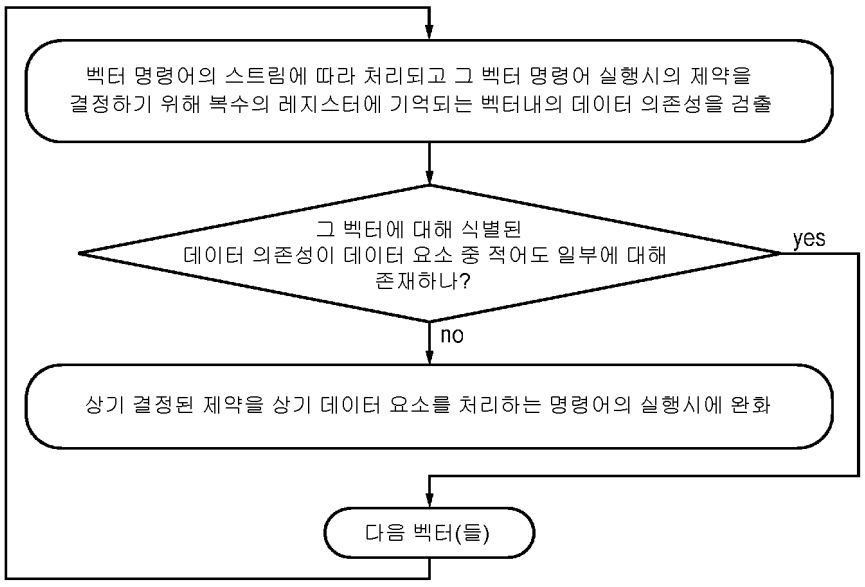
도면7

		상태 머신17	상태 머신19
		UN	UN
VPCLR		UN	
iVP == [0...0]		UN	
VPNXT V5			VPfirst
iVP == [0...2]			
VADD V0, V1, V2	✓ ✓ ✓ ? ? ? ?	WC	UN
VPNXT V5		WP (최적화 가능)	
iVP == [3...5]			
VSUB V0, V4, V6	✓ ✓ ✓ ✓ ✓ ✓ ? ?	WC	

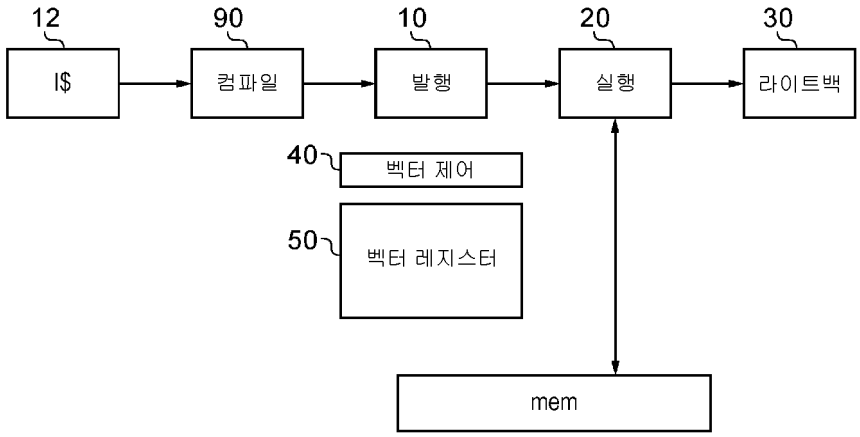
도면8



도면9



도면10



도면11

```
Vset_first      V0, R0, #0
Vset_partial    V0, R1, #1
.
.
.
Vset_partial    V0, R6, #6
Vset_last       V0, R7, #7
```