



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2024-0159595
(43) 공개일자 2024년11월05일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2018.01) G06F 9/38 (2024.01)
(52) CPC특허분류
G06F 9/30036 (2023.08)
G06F 9/30109 (2013.01)
(21) 출원번호 10-2024-7033147
(22) 출원일자(국제) 2022년12월15일
심사청구일자 없음
(85) 번역문제출일자 2024년10월04일
(86) 국제출원번호 PCT/GB2022/053244
(87) 국제공개번호 WO 2023/170373
국제공개일자 2023년09월14일
(30) 우선권주장
2203431.8 2022년03월11일 영국(GB)

(71) 출원인
에이알엠 리미티드
영국 캠브리지 씨비1 9엔제이 체리턴 폴번로드 110
(72) 발명자
마르티네즈 비센테, 알레한드로
영국 씨비1 9엔제이 케임브리지셔 케임브리지 폴
본 로드 110 씨/오 에이알엠 리미티드
선, 펄
영국 씨비1 9엔제이 케임브리지셔 케임브리지 폴
본 로드 110 씨/오 에이알엠 리미티드
(74) 대리인
이화익

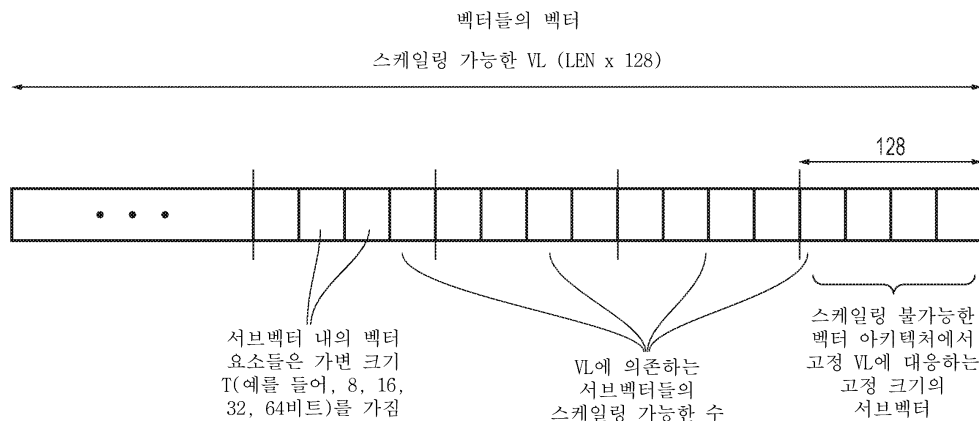
전체 청구항 수 : 총 24 항

(54) 발명의 명칭 스케일링 가능한 벡터 명령어 세트 아키텍처를 위한 서브벡터 지원 명령어

(57) 요약

장치는 데이터 처리를 수행하는 처리 회로부(16), 및 동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들의 디코딩에 응답하여 데이터 처리를 수행하도록 처리 회로부를 제어하는 명령어 디코딩 회로부(10)를 갖는다. 명령어 디코딩 회로부 및 처리 회로부는 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 지원하며, 각각의 서브벡터는 복수의 벡터 요소들을 포함한다. 서브벡터 지원 명령어에 응답하여, 명령어 디코딩 회로부는 서브벡터 입도에서 주어진 벡터에 대한 연산을 수행하도록 처리 회로부를 제어한다. 각각의 서브벡터는 동일한 서브벡터 길이를 갖는다.

대표도 - 도5



(52) CPC특허분류
G06F 9/3887 (2023.08)

명세서

청구범위

청구항 1

장치로서,

데이터 처리를 수행하는 처리 회로부; 및

동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들의 디코딩에 응답하여 상기 데이터 처리를 수행하도록 상기 처리 회로부를 제어하는 명령어 디코딩 회로부를 포함하며,

상기 명령어 디코딩 회로부 및 상기 처리 회로부는 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 지원하도록 구성되고, 각각의 서브벡터는 복수의 벡터 요소들을 포함하고, 각각의 서브벡터는 동일한 서브벡터 길이를 가지며,

상기 서브벡터 지원 명령어에 응답하여, 상기 명령어 디코딩 회로부는 서브벡터 입도에서 상기 주어진 벡터에 대한 연산을 수행하도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 2

제1항에 있어서, 각각의 서브벡터는 상기 서브벡터 지원 명령어를 사용하여 실행될 주어진 명령어 시퀀스에 대한 컴파일 시간에 알려지는 서브벡터 길이를 갖는, 장치.

청구항 3

제1항 또는 제2항에 있어서, 상기 주어진 벡터에 의해 얼마나 많은 서브벡터들이 포함되는지는 상기 주어진 명령어 시퀀스에 대한 컴파일 시간에 알려지지 않는, 장치.

청구항 4

제1항 내지 제3항 중 어느 한 항에 있어서, 상기 서브벡터 지원 명령어에 응답하여, 상기 명령어 디코딩 회로부는 상기 서브벡터 지원 명령어를 실행하는 동일한 인스턴스에 응답하여 상기 서브벡터들 각각을 처리하도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 5

제1항 내지 제4항 중 어느 한 항에 있어서, 각각의 서브벡터는 상기 주어진 벡터에 대해 사용되는 벡터 길이와는 무관한 아키텍처 정의 고정 크기의 서브벡터 길이를 갖는, 장치.

청구항 6

제5항에 있어서, 상기 아키텍처 정의 고정 크기는 미리 결정된 스케일링 불가능한 벡터 명령어 세트 아키텍처에 따라 처리되는 벡터 명령어들에 대해 규정된 아키텍처 정의 최대 벡터 길이에 대응하는, 장치.

청구항 7

제5항 또는 제6항에 있어서, 상기 아키텍처 정의 고정 크기는 128비트인, 장치.

청구항 8

제1항 내지 제7항 중 어느 한 항에 있어서, 각각의 서브벡터의 각각의 벡터 요소는 가변 요소 크기를 갖고, 상기 서브벡터 길이는 각각의 서브벡터 내의 각각의 벡터 요소에 대해 어떤 요소 크기가 사용되는지와는 무관한, 장치.

청구항 9

제1항 내지 제8항 중 어느 한 항에 있어서, 적어도 하나의 서브백터 지원 명령어에 대해, 서브백터 입도에서 수행되는 상기 연산은, 각각의 서브백터에 대해, 다른 서브백터들 내의 요소들과는 무관하게, 그 서브백터에 내의 백터 요소들에 대해 수행되는 연산인, 장치.

청구항 10

제1항 내지 제9항 중 어느 한 항에 있어서, 적어도 하나의 서브백터 지원 명령어에 대해, 서브백터 입도에서 수행되는 상기 연산은, 서브백터 내의 각각의 요소 위치에 대해, 상기 복수의 서브백터들 각각 내의 그 요소 위치에 있는 각자의 백터 요소들에 대해 수행되는 연산인, 장치.

청구항 11

제1항 내지 제10항 중 어느 한 항에 있어서, 적어도 하나의 서브백터 지원 명령어에 대해, 서브백터 입도에서 수행되는 상기 연산은 술어 값의 선택된 술어 비트들을 설정하거나, 상기 선택된 술어 비트들에 따라 연산을 수행하는 연산이며, 상기 선택된 술어 비트들은 백터의 서브백터 크기 부분들에 대응하는 술어 비트들인, 장치.

청구항 12

제1항 내지 제11항 중 어느 한 항에 있어서, 서브백터 지원 치환 명령어에 응답하여, 상기 명령어 디코더는, 백터 결과의 각각의 서브백터에 대해, 상기 서브백터를 적어도 하나의 백터 피연산자의 대응적으로 위치된 서브백터 내의 백터 요소들 중에서 선택된 하나 이상의 백터 요소들의 치환으로 설정하도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 13

제1항 내지 제12항 중 어느 한 항에 있어서, 서브백터 지원 감소 명령어에 응답하여, 상기 명령어 디코더는 서브백터 입도에서 적어도 하나의 감소 연산을 수행하도록 상기 처리 회로부를 제어하도록 구성되며, 각각의 감소 연산은 피연산자 백터의 복수의 백터 요소들을 결과 내의 단일 데이터 값으로 감소시키는, 장치.

청구항 14

제13항에 있어서, 인트라-서브백터 서브백터 지원 감소 명령어에 대해, 각각의 감소 연산에 대해 상기 복수의 백터 요소들은 상기 피연산자 백터의 대응하는 서브백터 내의 각자의 백터 요소들을 포함하는, 장치.

청구항 15

제13항 또는 제14항에 있어서, 인터-서브백터 서브백터 지원 감소 명령어에 대해, 각각의 감소 연산에 대해 상기 복수의 백터 요소들은 상기 피연산자 백터의 복수의 서브백터들 내의 대응하는 요소 위치들에 있는 상기 백터 요소들을 포함하는, 장치.

청구항 16

제1항 내지 제15항 중 어느 한 항에 있어서, 서브백터 지원 로드/저장 명령어에 응답하여, 상기 명령어 디코더는 메모리 시스템과 적어도 하나의 백터 레지스터 사이에서 하나 이상의 서브백터들을, 서브백터 입도에서, 전송하는 로드/저장 연산을 수행하도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 17

제16항에 있어서, 상기 서브백터 지원 로드/저장 명령어는 술어 값과 연관된 술어 명령어이고,

상기 서브백터 지원 로드/저장 명령어에 응답하여, 상기 명령어 디코더는, 서브백터 입도에서 상기 술어 값으로부터 선택된 술어 비트들에 기초하여, 상기 하나 이상의 서브백터들의 각각의 전송이 수행되는지 또는 마스킹되는지를 제어하도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 18

제1항 내지 제17항 중 어느 한 항에 있어서, 서브백터 지원 증가/감소 명령어에 응답하여, 상기 명령어 디코더

는 서브벡터 입도에서 상기 술어 값으로부터 선택된 술어 값의 비트들에 의해 벡터의 얼마나 많은 서브벡터 크기 부분들이 활성으로서 표시되는지에 기초하여 피연산자 값을 증가시키거나 감소시키도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 19

제18항에 있어서, 상기 술어 값은,

상기 서브벡터 지원 증가/감소 명령어에 의해 술어 피연산자로서 지정된 술어 값, 및

상기 서브벡터 지원 증가/감소 명령어에 의해 지정된 술어 패턴 식별자에 의해 암시되는 술어 값 - 상기 술어 패턴 식별자는 서브벡터 입도에서 술어 비트들의 미리 결정된 패턴을 지정함 -

중 하나인, 장치.

청구항 20

제1항 내지 제19항 중 어느 한 항에 있어서, 서브벡터 지원 술어 설정 명령어에 응답하여, 상기 명령어 디코더는, 벡터의 어떤 서브벡터들이 활성인지를 나타내기 위해, 서브벡터 입도에서 술어 값의 비트들을 설정하는 술어 설정 연산을 수행하도록 상기 처리 회로부를 제어하도록 구성되는, 장치.

청구항 21

제20항에 있어서, 상기 술어 설정 연산은,

서브벡터 입도에서 적용될 술어 비트들의 미리 결정된 패턴을 지정하는 술어 패턴 식별자, 및

제1 피연산자와 제2 피연산자의 비교에 기초한 서브벡터 입도 비교 연산들

중 하나에 기초하여 상기 술어 값을 설정하는 것을 포함하는, 장치.

청구항 22

방법으로서,

명령어 디코딩 회로부를 사용하여, 동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들을 디코딩하는 단계; 및

상기 프로그램 명령어들의 디코딩에 응답하여 데이터 처리를 수행하도록 처리 회로부를 제어하는 단계를 포함하며,

상기 명령어 디코딩 회로부 및 상기 처리 회로부는 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 지원하고, 각각의 서브벡터는 복수의 벡터 요소들을 포함하고, 각각의 서브벡터는 동일한 서브벡터 길이를 가지며,

상기 서브벡터 지원 명령어에 응답하여, 상기 명령어 디코딩 회로부는 서브벡터 입도에서 상기 주어진 벡터에 대한 연산을 수행하도록 상기 처리 회로부를 제어하는, 방법.

청구항 23

타겟 코드의 실행을 위한 명령어 실행 환경을 제공하도록 호스트 데이터 처리 장치를 제어하는 컴퓨터 프로그램으로서,

상기 타겟 코드의 명령어들에 응답하여 데이터 처리를 수행하도록 상기 호스트 데이터 처리 장치를 제어하기 위해 상기 타겟 코드의 상기 명령어들을 디코딩하는 명령어 디코딩 프로그램 로직을 포함하며,

상기 명령어 디코딩 프로그램 로직은 동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들의 디코딩을 지원하고,

상기 명령어 디코딩 프로그램 로직은 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 디코딩하는 서브벡터 지원 명령어 디코딩 프로그램 로직을 포함하고, 각각의 서브벡터는 복수의 벡터 요소들을 포함하고, 각각의 서브벡터는 동일한 서브벡터 길이를 가지며,

상기 서브벡터 지원 명령어에 응답하여, 상기 명령어 디코딩 프로그램 로직은 서브벡터 입도에서 상기 주어진 벡터에 대한 동작을 수행하도록 상기 호스트 데이터 처리 장치를 제어하도록 구성되는, 컴퓨터 프로그램.

청구항 24

제23항의 컴퓨터 프로그램을 저장하는 저장 매체.

발명의 설명

기술 분야

- [0001] 본 기법은 데이터 처리의 분야에 관한 것이다.
- [0002] 명령어 세트 아키텍처(ISA)는 특정 소프트웨어 프로그램을 개발할 때 소프트웨어 개발자 또는 컴파일러가 이용할 수 있는 명령어들의 세트를 정의하며, 하드웨어가 ISA에 따라 작성된 소프트웨어와 호환될 수 있게 하기 위해 하드웨어에서 프로세서 구현에 의해 지원될 필요가 있는 명령어들의 세트를 대응하는 방식으로 정의한다. 예를 들어, ISA는, 각각의 명령어에 대해, 명령어의 인코딩, 그의 입력 피연산자 및 결과 값의 표현, 및 입력 피연산자를 명령어의 결과에 매핑하기 위한 함수를 정의할 수 있다.
- [0003] 벡터 ISA는 단일 레지스터 내에 표현되는 2개 이상의 독립적인 벡터 요소를 포함하는 벡터 피연산자에 대해 동작하고/하거나, 2개 이상의 독립적인 벡터 요소를 포함하는 벡터 결과를 생성하는 적어도 하나의 벡터 명령어를 지원한다. 벡터 명령어는 단일 명령어 다중 데이터(SIMD) 방식으로 처리되어, 단일 명령어에 응답하여 상이한 데이터 값들에 대해 다수의 독립적인 계산이 수행될 수 있게 할 수 있다. 벡터 명령어는, 예를 들어, 고레벨 코드로 작성된 명령어들의 스칼라 루프가 벡터화될 수 있게 하여 다수의 루프 반복에 대응하는 처리가 벡터화된 루프의 단일 반복에 응답하여 수행될 수 있게 하는 데 유용할 수 있다. 이것은 소정량의 데이터 처리를 수행하기 위해 페치, 디코딩 및 실행될 필요가 있는 명령어들의 수를 감소시킴으로써 성능을 개선하는 데 도움이 된다.
- [0004] 적어도 몇몇 예들은, 장치로서, 데이터 처리를 수행하는 처리 회로부; 및 동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들의 디코딩에 응답하여 데이터 처리를 수행하도록 처리 회로부를 제어하는 명령어 디코딩 회로부를 포함하며, 명령어 디코딩 회로부 및 처리 회로부는 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 지원하도록 구성되고, 각각의 서브벡터는 복수의 벡터 요소들을 포함하고, 각각의 서브벡터는 동일한 서브벡터 길이를 가지며, 서브벡터 지원 명령어에 응답하여, 명령어 디코딩 회로부는 서브벡터 입도에서 주어진 벡터에 대한 연산을 수행하도록 처리 회로부를 제어하도록 구성되는, 장치를 제공한다.
- [0005] 적어도 몇몇 예들은, 방법으로서, 명령어 디코딩 회로부를 사용하여, 동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들을 디코딩하는 단계; 및 프로그램 명령어들의 디코딩에 응답하여 데이터 처리를 수행하도록 처리 회로부를 제어하는 단계를 포함하며, 명령어 디코딩 회로부 및 처리 회로부는 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 지원하고, 각각의 서브벡터는 복수의 벡터 요소들을 포함하고, 각각의 서브벡터는 동일한 서브벡터 길이를 가지며, 서브벡터 지원 명령어에 응답하여, 명령어 디코딩 회로부는 서브벡터 입도에서 주어진 벡터에 대한 연산을 수행하도록 처리 회로부를 제어하는, 방법을 제공한다.
- [0006] 적어도 몇몇 예들은, 타겟 코드의 실행을 위한 명령어 실행 환경을 제공하도록 호스트 데이터 처리 장치를 제어하는 컴퓨터 프로그램으로서, 타겟 코드의 명령어들에 응답하여 데이터 처리를 수행하도록 호스트 데이터 처리 장치를 제어하기 위해 타겟 코드의 명령어들을 디코딩하는 명령어 디코딩 프로그램 로직을 포함하며, 명령어 디코딩 프로그램 로직은 동일한 명령어 시퀀스가 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치들에

서 실행되는 것을 가능하게 하기 위해 스케일링 가능한 벡터 길이의 벡터들에 대해 동작하는 벡터 명령어들을 지원하는 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 정의된 프로그램 명령어들의 디코딩을 지원하고, 명령어 디코딩 프로그램 로직은 주어진 벡터를 복수의 서브벡터들을 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 디코딩하는 서브벡터 지원 명령어 디코딩 프로그램 로직을 포함하고, 각각의 서브벡터는 복수의 벡터 요소들을 포함하고, 각각의 서브벡터는 동일한 서브벡터 길이를 가지며, 서브벡터 지원 명령어에 응답하여, 명령어 디코딩 프로그램 로직은 서브벡터 입도에서 주어진 벡터에 대한 동작을 수행하도록 호스트 데이터 처리 장치를 제어하도록 구성되는, 컴퓨터 프로그램을 제공한다.

- [0007] 컴퓨터 프로그램은 저장 매체에 저장될 수 있다. 저장 매체는 비일시적 저장 매체일 수 있다.
- [0008] 본 기법의 추가의 태양들, 특징들 및 이점들은 첨부 도면과 관련하여 읽혀질 예들의 하기의 설명으로부터 명백해질 것이다.
- [0009] 도 1은 벡터 ISA를 지원하는 데이터 처리 장치의 예를 예시한다.
- [0010] 도 2는 스케일링 가능한 벡터 ISA에 대한 레지스터의 예를 도시한다.
- [0011] 도 3은 술어 및 가변 벡터 요소 크기의 예를 도시한다.
- [0012] 도 4는 상이한 최대 벡터 길이를 지원하는 상이한 하드웨어 구현들이 동일한 명령어 시퀀스를 실행할 수 있게 하는 스케일링 가능한 벡터 ISA의 예를 예시한다.
- [0013] 도 5는 서브벡터들의 벡터로서 취급되는 벡터의 예를 도시한다.
- [0014] 도 6은 서브벡터 지원 명령어를 처리하는 방법을 도시한다.
- [0015] 도 7 내지 도 11은 서브벡터 지원 치환 명령어의 여러 예를 예시한다.
- [0016] 도 12 및 도 13은 서브벡터 지원 감소 명령어의 예를 예시한다.
- [0017] 도 14 및 도 15는 서브벡터 지원 로드/저장 명령어의 예를 예시한다.
- [0018] 도 16 및 도 17은 서브벡터 지원 술어 설정 명령어의 예를 예시한다.
- [0019] 도 18은 서브벡터 지원 증가/감소 명령어의 예를 예시한다.
- [0020] 도 19는 시뮬레이터 구현을 예시한다.
- [0021] 장치는 데이터를 처리를 수행하는 처리 회로부, 및 스케일링 가능한 벡터 명령어 세트 아키텍처(ISA)에 따라 정의된 프로그램 명령어의 디코딩에 응답하여 데이터 처리를 수행하도록 처리 회로부를 제어하는 명령어 디코딩 회로부를 갖는다. 스케일링 가능한 벡터 ISA("벡터 길이 불가지론(vector length agnostic)" 벡터 ISA로도 알려짐)은 스케일링 가능한 벡터 길이의 벡터에 대해 동작하는 벡터 명령어를 지원하여, 상이한 최대 벡터 길이들을 지원하는 하드웨어를 갖는 장치에서 동일한 명령어 시퀀스가 실행되는 것을 가능하게 한다. 이것은 그것이 프로세서 구현의 상이한 하드웨어 설계자들이 그들의 설계 우선 순위가 고성능인지 또는 감소된 회로 면적 및 전력 소비인지에 따라 상이한 최대 벡터 길이들을 선택할 수 있게 하기 때문에 유용하지만, 소프트웨어 개발자들은, 특정 하드웨어 플랫폼에 의해 지원되는 특정 최대 벡터 길이에 관계없이, 스케일링 가능한 벡터 ISA에 따라 작성된 소프트웨어가 스케일링 가능한 벡터 ISA를 지원하는 임의의 하드웨어 플랫폼에 걸쳐 실행될 수 있으므로 특정 하드웨어 플랫폼에 그들의 소프트웨어를 맞춤 필요가 없다. 따라서, 스케일링 가능한 벡터 ISA의 특정 벡터 명령어에 사용될 벡터 길이는 컴파일 시간에 알려지지 않는다(ISA 자체에서 고정되도록 정의되지도 않고, 소프트웨어 자체의 파라미터에 의해 지정되지도 않음). 스케일링 가능한 벡터 ISA의 주어진 벡터 명령어에 응답하여 수행되는 연산은 특정 하드웨어 구현을 위해 선택된 벡터 길이에 따라 상이할 수 있다(예를 들어, 더 큰 최대 벡터 길이를 지원하는 하드웨어는 더 작은 최대 벡터 길이를 지원하는 하드웨어보다 주어진 벡터 명령어에 대해 더 많은 수의 벡터 요소를 처리할 수 있다). 따라서 더 짧은 벡터 길이를 갖는 구현은 더 긴 벡터 길이를 갖는 구현보다 특정 기능을 수행하기 위해 더 많은 수의 루프 반복을 요구할 수 있다.
- [0022] 스케일링 가능한 벡터 ISA는 상이한 최대 벡터 길이들을 갖는 프로세서 구현들 간에 쉽게 복사될 수 있는 플랫폼 독립적인 프로그램 코드의 개발을 가능하게 하는 데 매우 유용할 수 있지만, 알려진 벡터 길이를 가정하여 컴파일된 상당한 양의 레거시 코드가 있을 수 있다. 스케일링 가능한 벡터 ISA와 함께 사용하기 위해 레거시 소프트웨어를 재개발하려면 상당한 양의 소프트웨어 개발 노력이 들 수 있다. 이것은 특히 소프트웨어 파이프라이닝 또는 루프 언롤링(loop unrolling)과 같은, 벡터화된 코드에 대한 성능을 개선하기 위해 전형적으로 사

용되는 몇몇 소프트웨어 기술들이 컴파일 시간에 알려진 명령어에 사용되는 벡터 길이에 의존할 수 있기 때문에 그러하다. 따라서, 스케일링 불가능한 벡터 ISA의 명령어를 스케일링 가능한 벡터 ISA의 명령어에 리매핑하는 것은 간단하지 않을 수 있는데, 왜냐하면 레거시 코드에 사용되는 몇몇 기술들은 스케일링 가능한 벡터 ISA에서 이용 가능하지 않을 수 있기 때문이다. 이것은 스케일링 가능한 벡터 ISA의 채택에 대한 중요한 장벽일 수 있고, 몇몇 소프트웨어 개발자들이 스케일링 가능한 벡터 ISA를 사용하지 않기로 선택하는 결과를 초래할 수 있으며, 따라서 스케일링 가능한 벡터 ISA를 지원하는 더 새로운 프로세서에서 실행되는 상당한 양의 소프트웨어는, 스케일링 가능한 벡터 ISA를 사용하여 큰 최대 벡터 길이를 지원하는 프로세서 구현(그 프로세서 구현은 또한 역호환성 이유로 스케일링 불가능한 벡터 ISA를 지원할 수 있음)에서도, 상대적으로 짧은 최대 벡터 길이를 사용하는 덜 성능 효율적인 스케일링 불가능한 벡터 ISA를 여전히 사용할 수 있다. 이것은 하드웨어의 전체 수행 능력이 많은 프로그램에 대해 사용되지 않을 수 있다는 것을 의미한다.

[0023] 아래에서 논의되는 예들에서, 명령어 디코딩 회로부 및 처리 회로부는, 스케일링 가능한 벡터 ISA 내에서, 주어진 벡터를 복수의 서브벡터를 포함하는 것으로서 취급하는 서브벡터 지원 명령어를 지원하며, 이때 각각의 서브벡터는 복수의 벡터 요소를 포함하고 각각의 서브벡터는 동일한 서브벡터 길이를 갖는다. 서브벡터 지원 명령어에 응답하여, 명령어 디코딩 회로부는 서브벡터 입도에서 주어진 벡터에 대한 연산을 수행하도록 처리 회로부를 제어한다. 이것은 스케일링 가능한 벡터 ISA의 채택을 가능하게 하는 데 요구되는 소프트웨어 개발 노력을 감소시키는 데 도움이 되는데, 왜냐하면 스케일링 불가능한 벡터 ISA의 각각의 벡터가 스케일링 가능한 ISA에서의 벡터의 서브벡터들 중 하나에 매핑될 수 있기 때문이다. 이것은 스케일링 불가능한 소프트웨어를 스케일링 가능한 소프트웨어에 매핑하는 것을 더 간단하게 만들고, 그에 따라 스케일링 가능한 벡터 ISA의 사용에 대한 장벽을 감소시켜, 스케일링 가능한 벡터 ISA를 지원하는 장치에서 실행되는 코드의 더 많은 부분이 실제로 스케일링 가능한 벡터 ISA를 사용하고 있을 가능성을 더 높이며, 이는 다양한 프로세서 구현에 걸쳐 평균 성능을 개선하는 경향이 있을 것인데, 왜냐하면 더 긴 벡터 길이를 지원하는 그러한 고급 프로세서가 그러한 더 긴 벡터 길이를 이용하여 성능을 개선할 수 있을 가능성이 더 높을 수 있기 때문이다.

[0024] 각각의 서브벡터는 서브벡터 지원 명령어를 사용하여 실행될 주어진 명령어 시퀀스에 대한 컴파일 시간에 알려지는 서브벡터 길이를 가질 수 있다. 이것은 그것이 스케일링 가능한 벡터 ISA의 벡터 피연산자 또는 결과가 각각 알려진 벡터 길이의 다수의 더 작은 서브벡터를 포함하는 "벡터들의 벡터"로서 정의될 수 있게 하며, 따라서 스케일링 불가능한 벡터 ISA를 사용하여 작성된 소프트웨어에 따라 정의된 다수의 벡터가 스케일링 가능한 벡터 ISA에 따라 더 큰 벡터로 결합될 수 있기 때문에 도움이 된다. 각각의 서브벡터의 길이는 컴파일 시간에 알려지기 때문에, 컴파일 시간에서의 벡터 길이에 대한 지식에 의존하는 임의의 성능 개선 소프트웨어 기술이 서브벡터의 입도에서 적용될 수 있으며, 이는 그러한 소프트웨어 기술을 유지하면서 소프트웨어 개발자가 스케일링 불가능한 벡터 ISA에 따라 작성된 코드를 스케일링 가능한 벡터 ISA에 따라 정의된 코드에 매핑하는 것을 훨씬 더 쉽게 만든다.

[0025] 주어진 벡터에 의해 얼마나 많은 서브벡터가 포함되는지는 서브벡터 지원 명령어를 사용하여 실행될 주어진 명령어 시퀀스에 대한 컴파일 시간에 알려지지 않는다. 다시 말해서, 주어진 벡터에 대한 전체 벡터 길이는 (상이한 프로세서 구현들에 대해 허용될 수 있는 다양한 최대 벡터 길이를 정의할 수 있는) 스케일링 가능한 벡터 ISA에 의해 허용되는 바와 같은 스케일링 가능한 벡터 길이일 수 있다. 이것은 서브벡터 지원 명령어가 스케일링 가능한 벡터 ISA의 플랫폼 독립적인 특성으로부터 여전히 이익을 얻으면서 다양한 상이한 성능/전력 포인트를 지원할 수 있다는 것을 의미한다. 그럼에도 불구하고, 벡터화에 대한 입도로서 사용될 벡터 길이에 대한 지식에 의존하는 소프트웨어 최적화들이 여전히 사용될 수 있는데, 왜냐하면 그들은 서브벡터 지원 명령어를 사용하여 스케일링 가능한 벡터 길이의 더 큰 주어진 벡터에 가변적인 수의 서브벡터가 수용될 수 있기 전에 알려진 서브벡터 길이의 서브벡터를 참조하여 적용될 수 있기 때문이다. 예를 들어, 이전에 스케일링 불가능한 벡터 ISA를 사용하여 벡터화된 형태로 구현되었을 벡터화된 루프의 가변적인 수의 반복이 스케일링 가능한 벡터 ISA에서의 벡터화된 루프에 매핑될 수 있으며, 이때 원래의 벡터화된 루프의 각각의 반복은 스케일링 가능한 벡터 ISA의 서브벡터 지원 명령어에 의해 처리된 벡터들의 서브벡터들 중 하나에 대응한다. 이것은 코드의 컴파일 및 소프트웨어 개발을 훨씬 더 간단하게 만드는데, 왜냐하면 그것은 스칼라 루프를 ISA에 따라 벡터화된 스케일링 가능한 코드로 다시 변환하기 전에 벡터화된 스케일링 불가능한 코드로부터 스칼라 루프로 되돌아갈 필요성을 회피할 수 있기 때문이다 - 대신에 개재되는 역벡터화 단계 없이 스케일링 불가능한 ISA의 벡터 명령어를 스케일링 가능한 ISA의 스케일링 가능한 벡터 명령어에 직접 매핑하는 것이 더 간단할 수 있다(다른 컴파일러는 컴파일을 스케일링 불가능한 ISA를 위해 컴파일된 이전 코드에 기초함이 없이 서브벡터 처리 명령어를 포함하는 스케일링 가능한 ISA에 대해 직접 간단히 컴파일할 수 있다). 컴파일 시간에 알려진 서브벡터 길이는 주어진 벡터에 대해 사용된 벡터 길이와는 무관할 수 있으며, 따라서 서브벡터 길이는 주어진 하드웨어 구현에 의해 전

체 주어진 벡터에 대해 사용된 실제 벡터 길이에 관계없이 주어진 소프트웨어의 주어진 명령어에 대해 동일하다.

[0026] 서브벡터 지원 명령어에 응답하여, 명령어 디코딩 회로부는 서브벡터 지원 명령어를 실행하는 동일한 인스턴스에 응답하여 서브벡터들 각각을 처리하도록 처리 회로부를 제어할 수 있다. 예를 들어, 서브벡터 입도에서 수행되는 연산들 각각은 병렬로 처리될 수 있다. 대안적으로, 서브벡터 입도에서 수행되는 연산들은 순차적으로, 또는 일부는 순차적으로 그리고 일부는 병렬로, 또는 파이프라인 방식으로 처리될 수 있다. 다양한 연산이 서브벡터 입도에서 수행되는 정확한 타이밍에 관계없이, 서브벡터 입도에서의 연산들 각각은 서브벡터 지원 명령어의 실행의 단일 인스턴스에 응답하여 수행되며, 따라서 벡터 ISA의 SIMD 이익이 실현될 수 있다.

[0027] 서브벡터 길이가 컴파일 시간에 알려질 수 있는 다수의 방법이 있다. 스케일링 가능한 벡터 ISA의 몇몇 구현에 대해, ISA는 서브벡터 길이를 소프트웨어 코드 자체에 의해 지정될 수 있는 가변 소프트웨어 정의 파라미터로서 정의하여, 소프트웨어가 2개 이상의 상이한 서브벡터 크기 사이에서 선택할 수 있게 할 수 있다. 예를 들어, 이것은 상이한 스케일링 불가능한 벡터 길이들을 갖는 2개 이상의 상이한 스케일링 불가능한 벡터 ISA로부터의 코드를 리매핑하기 위한 지원을 제공하는 것을 도울 수 있다.

[0028] 그러나, 일례에서 각각의 서브벡터는 주어진 벡터에 대해 사용된 벡터 길이와는 무관한 아키텍처 정의 고정 크기(architecturally-defined fixed size)의 서브벡터 길이를 가질 수 있다. 이것은 아키텍처를 단순화하고, 소프트웨어가 서브벡터 길이를 지정할 어떠한 필요성도 회피할 수 있다. 대신에, 서브벡터 길이는 스케일링 가능한 벡터 ISA에 정의된 서브벡터 지원 명령어의 아키텍처 정의에서 고정될 수 있다.

[0029] 아키텍처 정의 고정 크기는 미리 결정된 스케일링 불가능한 벡터 ISA에 따라 처리되는 벡터 명령어에 대해 규정된 아키텍처 정의 최대 벡터 길이에 대응할 수 있다. 예를 들어, 미리 결정된 스케일링 불가능한 벡터 ISA는 영국 케임브리지 소재의 Arm® Limited에 의해 제공하는 "진보된 SIMD" 아키텍처(Neon™ 아키텍처로도 알려짐)일 수 있다.

[0030] 예를 들어, 서브벡터 길이의 아키텍처 정의 고정 크기는 128비트일 수 있다. 이것은 128비트의 최대 벡터 길이를 정의하는 진보된 SIMD 아키텍처와의 호환성에 유용하다.

[0031] 다른 스케일링 불가능한 벡터 ISA를 타겟으로 하는 서브벡터 지원 명령어를 구현하는 것이 또한 가능하며, 이 경우에 서브벡터 길이는 타겟이 되는 특정 ISA에 따라 달라질 수 있다.

[0032] 또한, 임의의 특정 스케일링 불가능한 벡터 ISA를 타겟으로 하려고 의도함이 없이 서브벡터 지원 명령어를 구현하지만, 단순히 특정 스케일링 불가능한 벡터 ISA에 사용되는 길이를 에뮬레이션하려는 임의의 목표와는 무관한 주어진 고정 서브벡터 길이를 선택하는 것이 가능하다. 타겟이 되는 어떠한 특정 스케일링 불가능한 벡터 ISA도 없는 경우에도, 아키텍처 정의 고정 크기의 서브벡터 길이를 갖는 서브벡터를 정의하여, 소프트웨어 개발자 및 컴파일러가 그러한 고정 크기의 컴파일 시간 지식에 의존하는 소프트웨어 성능 최적화를 이용할 수 있게 하는 것은 여전히 유용할 수 있다.

[0033] 서브벡터 지원 명령어는 가변 요소 크기를 지원하여서, 각각의 서브벡터의 각각의 벡터 요소가 스케일링 가능한 벡터 ISA에 의해 지원되는 2개 이상의 상이한 크기로부터 선택된 가변 요소 크기를 갖게 할 수 있다. 서브벡터 길이는 각각의 서브벡터 내의 각각의 벡터 요소에 대해 사용되는 요소 크기와는 무관할 수 있다. 따라서, 선택된 요소 크기가 더 큰 요소 크기인지 또는 더 작은 요소 크기인지에 관계없이 동일한 서브벡터 길이가 사용될 수 있다. 서브벡터별로 정의되는 벡터 요소들의 수는 서브벡터 길이와 선택된 요소 크기 사이의 비율에 대응할 수 있다. 선택된 요소 크기는 서브벡터 지원 명령어를 포함하는 명령어 시퀀스의 소프트웨어 지정 파라미터에 의해 정의될 수 있으며, 따라서 컴파일 시간에 알려진다. 따라서, 서브벡터 길이 및 요소 크기 둘 모두가 컴파일 시간에 알려질 수 있기 때문에, 서브벡터별로 제공되는 벡터 요소들의 수가 또한 컴파일 시간에 알려질 수 있지만, 전체로서의 주어진 벡터 내의 벡터 요소들의 총 수는 컴파일 시간에 알려지지 않을 수 있는데, 왜냐하면 주어진 벡터의 전체 벡터 길이는 스케일링 가능한 벡터 ISA에 따라 컴파일 시간에 알려지지 않기 때문이다.

[0034] 서브벡터 입도에서 수행되는 연산은 달라질 수 있다. 다수의 상이한 서브벡터 지원 명령어가 스케일링 가능한 벡터 ISA의 일부로서 정의되어 다수의 상이한 연산이 프로그래머 또는 컴파일러의 선택에 따라 서브벡터 입도에서 수행되는 것을 가능하게 할 수 있다.

[0035] 몇몇 예들에서, 적어도 하나의 서브벡터 지원 명령어에 대해, 서브벡터 입도에서 수행되는 연산은, 각각의 서브벡터에 대해, 다른 서브벡터 내의 요소와는 무관하게, 그 서브벡터 내의 벡터 요소에 대해 수행되는 연산이다. 따라서, 이것은 스케일링 불가능한 벡터 ISA에서 다수의 별개의 벡터 명령어(각각이 컴파일 시간에 알려진 스케

일링 불가능한 벡터 길이의 각자의 벡터에 대해 동작함)를 사용하여 수행되었을 연산이 단일 서브벡터 지원 명령어에서 수행될 수 있게 할 수 있다(예를 들어, 그러한 별개의 벡터 명령어들을 스케일링 불가능한 벡터 ISA의 코드에 작성된 루프의 상이한 반복들에 대응할 수 있음).

[0036] 몇몇 예들에서, 적어도 하나의 서브벡터 지원 명령어에 대해, 서브벡터 입도에서 수행되는 연산은, 서브벡터 내의 각각의 요소 위치에 대해, 복수의 서브벡터들 각각 내의 그 요소 위치에 있는 각자의 벡터 요소에 대해 수행되는 연산이다. 그러한 명령어는, 예를 들어, 단일 루프 반복 내에서 벡터 명령어들의 시퀀스를 사용하여 수행되었을 처리를 복제하는 데 유용할 수 있으며, 여기서 그러한 벡터 명령어들은 그러한 벡터 명령어들의 시퀀스에 대해 지정된 다수의 벡터 피연산자 내에서 대응하는 요소 위치들에서 데이터 값들을 결합했을 것이다.

[0037] 몇몇 예들에서, 적어도 하나의 서브벡터 지원 명령어에 대해, 서브벡터 입도에서 수행되는 연산은 술어 값의 선택된 술어 비트를 설정하거나, 그에 따라 연산을 수행하는 연산이며, 여기서 선택된 술어 비트는 벡터의 서브벡터 크기의 부분에 대응하는 술어 비트이다. 이것은 서브벡터 길이보다 더 작은 개별 벡터 요소들의 입도에서 술어를 설정하거나 해석할 수 있는 스케일링 가능한 벡터 ISA의 많은 명령어와는 상이할 수 있다. 그러한 서브벡터 지원 술어 설정 또는 술어 의존 명령어는 소정 인스턴스에서, 예를 들어 스케일링 불가능한 벡터 ISA의 대응하는 코드가 컴파일 시간에 알려진 특정 벡터 길이의 벡터에 대해 수행된 벡터화된 루프의 전체 반복을 마스킹했을 조건에서, 전체 서브벡터의 처리가 선택적으로 마스킹될 수 있게 하는 데 유용할 수 있다.

[0038] 몇몇 예들에서, 스케일링 가능한 벡터 ISA는 서브벡터 지원 치환 명령어의 적어도 하나의 변형을 지원할 수 있다. 서브벡터 지원 치환 명령어에 응답하여, 명령어 디코더는, 벡터 결과의 각각의 서브벡터에 대해, 서브벡터를 적어도 하나의 벡터 피연산자의 대응적으로 위치된 서브벡터 내의 벡터 요소들 중에서 선택된 하나 이상의 벡터 요소의 치환으로 설정하도록 처리 회로부를 제어한다. 서브벡터 지원 치환 명령어는 벡터 피연산자들 중 하나 내에서 상이한, 대응하지 않는, 서브벡터 위치에 있는 벡터 요소의 비트에 기초하여 벡터 결과의 주어진 서브벡터의 벡터 요소를 설정하는 것이 가능하지 않을 수 있다. 전체로서의 주어진 벡터의 전체 벡터 길이에 걸쳐서보다는, 서브벡터 입도에서 치환을 수행함으로써, 이것은 주어진 명령어에 대해 수행되는 그러한 서브벡터 입도 치환들의 수가 스케일링 가능한 벡터 ISA에 따라 하드웨어에서 지원되는 구현된 벡터 길이에 기초하여 스케일링되는 것을 여전히 가능하게 하면서, 서브벡터 지원 치환 명령어의 거동이 치환을 위해 알려진 벡터 길이를 가정하는, 스케일링 불가능한 벡터 ISA에 정의된 다수의 별개의 스케일링 불가능한 치환 명령어의 거동을 미러링할 수 있게 할 수 있다.

[0039] 몇몇 예들에서, 스케일링 가능한 벡터 ISA는 서브벡터 지원 감소 명령어의 적어도 하나의 변형을 지원할 수 있다. 서브벡터 지원 감소 명령어에 응답하여, 명령어 디코더는 서브벡터 입도에서 적어도 하나의 감소 연산을 수행하도록 처리 회로부를 제어할 수 있으며, 각각의 감소 연산은 피연산자 벡터의 복수의 벡터 요소를 결과 내의 단일 데이터 값으로 감소시키는 것이다. 그러한 감소 연산은 개별 서브벡터의 입도에서 수행될 때 전체 벡터 길이에 걸쳐 수행되는 대응하는 감소 연산과는 상이한 결과를 제공할 수 있는데, 왜냐하면 상이한 서브벡터들 내의 벡터 요소들이 서로 결합되지 않을 수 있기 때문이다. 그러한 감소들은 스케일링 불가능한 벡터화된 코드의 시퀀스에서 주어진 루프 반복 또는 일련의 루프 반복들 내에서 다수의 명령어들의 시퀀스에 의해 구현되었을 수 있는 처리를 에뮬레이션하는 데 유용할 수 있으며, 이들 각각은 벡터 피연산자의 각각의 요소를 임의의 선행 루프 반복들에서의 유사한 결합들의 결과를 추적하는 누산기 값의 대응하는 요소들과 결합하기 위한 명령어의 단일 인스턴스를 포함한다.

[0040] 서브벡터 지원 감소 명령어의 상이한 변형들이 가능하며, 이들은 감소가 서브벡터 입도에서 수행되는 방식에서 다르다.

[0041] 예를 들어, 인트라-서브벡터 서브벡터 지원 감소 명령어에 대해, 각각의 감소 연산에 대해 복수의 벡터 요소는 피연산자 벡터의 대응하는 서브벡터 내에 각자의 벡터 요소를 포함한다. ISA에서 그러한 명령어를 포함하는 것은 소프트웨어 개발자가 명령어를 사용하여, 컴파일 시간에 알려지지 않은 벡터 길이를 갖는 스케일링 가능한 아키텍처에서, 단일 벡터 피연산자의 모든 벡터 요소에 걸쳐 감소를 수행한 스케일링 불가능한 코드의 거동을 에뮬레이션할 수 있게 하는 데 유용할 수 있다. 각각의 서브벡터 입도 감소의 결과는 결과 값의 상이한 서브벡터에 배치될 수 있다. 대안적으로, 인트라-서브벡터 서브벡터 지원 감소 명령어의 변형은 각각의 서브벡터 입도 감소의 결과를 결과 값의 하나 이상의 서브벡터의 각각의 벡터 요소에 배치할 수 있다.

[0042] 다른 변형에서, 인터-서브벡터 서브벡터 지원 감소 명령어에 대해, 각각의 감소 연산에 대해 복수의 벡터 요소는 피연산자 벡터의 복수의 서브벡터 내에서 대응하는 요소 위치에 벡터 요소를 포함한다. 이러한 유형의 명령어에 대해, 단일 결과로 감소되는 벡터 요소는 서브벡터 길이의 간격에서 선택된 벡터 요소이다. ISA에서 그런

한 명령어를 포함하는 것은 소프트웨어 개발자가 명령어를 사용하여, 컴파일 시간에 알려지지 않은 벡터 길이를 갖는 스케일링 가능한 아키텍처에서, 다수의 연속 루프 반복에서 처리된 벡터 내의 동일한 요소 위치에 있는 벡터 요소들에 걸쳐 감소를 수행하는 스케일링 불가능한 코드의 거동을 에뮬레이션할 수 있게 하는 데 유용할 수 있다.

[0043] 스케일링 가능한 벡터 ISA는 이러한 유형의 서브벡터 지원 감소 명령어들 중 임의의 하나 이상을 지원할 수 있다.

[0044] 몇몇 예들에서, 스케일링 가능한 벡터 ISA는 서브벡터 지원 로드/저장 명령어의 적어도 하나의 변형을 지원할 수 있다. 서브벡터 지원 로드/저장 명령어에 응답하여, 명령어 디코더는 메모리 시스템과 적어도 하나의 벡터 레지스터 사이에서 하나 이상의 서브벡터를, 서브벡터 입도에서, 전송하기 위한 로드/저장 연산을 수행하도록 처리 회로부를 제어할 수 있다. 이것은 알려진 벡터 길이의 벡터에 대해 대응하는 로드/저장 연산을 수행했을 스케일링 불가능한 벡터 ISA의 로드/저장 명령어의 거동을 에뮬레이션하는 데 유용할 수 있다.

[0045] 서브벡터 지원 로드/저장 명령어는 술어 값과 연관된 술어 명령어일 수 있다. 서브벡터 지원 로드/저장 명령어에 응답하여, 명령어 디코더는, 서브벡터 입도에서 술어 값으로부터 선택된 술어 비트에 기초하여, 하나 이상의 서브벡터의 각각의 전송이 수행되는지 또는 마스킹되는지를 제어하도록 처리 회로부를 제어할 수 있다. 이것은 벡터의 벡터 요소에 사용되는 요소 크기(요소 크기는 서브벡터 길이보다 작음)의 입도에서 선택된 술어로 전체(스케일링 가능한) 벡터 길이에 걸쳐 로드/저장 연산을 수행할 수 있는 스케일링 가능한 벡터 ISA의 다른 로드/저장 명령어의 거동과는 상이할 수 있다.

[0046] 몇몇 예들에서, 스케일링 가능한 벡터 ISA는 서브벡터 지원 증가/감소 명령어의 적어도 하나의 변형을 지원할 수 있다. 서브벡터 지원 증가/감소 명령어에 응답하여, 명령어 디코더는 서브벡터 입도에서 술어 값으로부터 선택된 술어 값의 비트들에 의해 벡터의 얼마나 많은 서브벡터-크기의 부분이 활성으로서 표시되는지에 기초하여 피연산자 값을 증가시키거나 감소시키도록 처리 회로부를 제어할 수 있다. 이것은 루프 제어에 도움이 될 수 있으며, 따라서 적어도 하나의 추가 루프 반복이 여전히 수행될 필요가 있는지를 결정하기 위해 소프트웨어에 의해 사용되는 루프 제어 변수는 루프의 최신 반복에서 처리된 서브벡터들의 수에 따라 증가되거나 감소될 수 있다(그 루프 반복에서 처리된 서브벡터들의 수는 컴파일 시간에 알려지지 않을 수 있기 때문에, 처리된 서브벡터들의 수가 추론되고 사용되어 그에 맞춰 루프 제어 변수를 업데이트할 수 있게 하는 명령어를 제공하는 것이 유용할 수 있다).

[0047] 피연산자 값을 업데이트하는 방법을 결정하기 위해 서브벡터 지원 증가/감소 명령어에 의해 사용되는 술어 값은 서브벡터 지원 증가/감소 명령어에 의해 술어 피연산자로서 지정된 술어 값; 및 서브벡터 지원 증가/감소 명령어에 의해 지정된 술어 패턴 식별자에 의해 암시된 술어 값 중 하나일 수 있으며, 술어 패턴 식별자는 서브벡터 입도에서 술어 비트들의 미리 결정된 패턴을 지정한다.

[0048] 몇몇 예들에서, 스케일링 가능한 벡터 ISA는 서브벡터 지원 술어 설정 명령어의 적어도 하나의 변형을 지원할 수 있다. 서브벡터 지원 술어 설정 명령어에 응답하여, 명령어 디코더는 서브벡터 입도에서 술어 값의 비트들을 설정하기 위한 술어 설정 연산을 수행하여, 벡터의 어떤 서브벡터들이 활성인지를 나타내도록 처리 회로부를 제어할 수 있다. 그러한 명령어는 다른 서브벡터 지원 술어 명령어들에 의해 어떤 서브벡터들이 처리되는지를 제어하는 데 유용할 수 있다. 특정 값(예를 들어, 0)으로 설정된 술어 값의 비트들은 대응하는 서브벡터의 처리가 마스킹되게 할 수 있다. 그러한 술어 설정 명령어는 서브벡터 길이보다 작을 수 있는 벡터 요소 크기의 입도에서 술어 값을 설정할 수 있는 스케일링 가능한 벡터 ISA의 다른 술어 설정 명령어와는 상이한 명령어일 수 있다.

[0049] 서브벡터 지원 술어 설정 명령어에 대한 술어 설정 연산은 서브벡터 입도에서 적용될 술어 비트들의 미리 결정된 패턴을 지정하는 술어 패턴 식별자; 및 제1 피연산자와 제2 피연산자의 비교에 기초한 서브벡터 입도 비교 연산 중 하나에 기초하여 술어 값을 설정하는 것을 포함할 수 있다. 이러한 접근법의 경우, 술어의 새로운 값은 서브벡터 지원 술어 설정 명령어에서 명시적으로 지정되는 것이 아니라, 상이한 벡터 길이들에 대해 스케일링 가능할 수 있는 어떤 일반적인 패턴에 따라 정의될 수 있다. 이것은 설정될 술어 비트들의 수가 스케일링 가능한 벡터 길이로 인해 컴파일 시간에 알려지지 않을 수 있기 때문에 유용하다.

[0050] 서브벡터 지원 명령어의 다양한 예가 위에서 설명된다. 스케일링 가능한 벡터 ISA의 임의의 주어진 구현이 이러한 유형의 명령어들 모두를 지원할 필요는 없다는 것이 인식될 것이다. 본 출원에서 설명된 명령어들 중 임의의 하나 이상이 구현될 수 있다.

- [0051] 위에서 논의된 기법은 위에서 논의된 명령어 디코더 및 처리 회로부를 구현하기 위해 제공되는 하드웨어 회로부를 갖는 데이터 처리 장치 내에서 구현될 수 있다.
- [0052] 그러나, 동일한 기법은 또한 타겟 코드의 실행을 위한 명령어 실행 환경을 제공하기 위해 호스트 데이터 처리 장치 상에서 실행되는 컴퓨터 프로그램 내에서 구현될 수 있다. 그러한 컴퓨터 프로그램은 스케일링 가능한 벡터 명령어 세트 아키텍처에 따라 타겟 코드를 실제로 지원하는 하드웨어 장치 상에서 제공될 아키텍처 환경을, 호스트 데이터 처리 장치 자체가 그 아키텍처를 지원하지 않을지라도, 시뮬레이션하도록 호스트 데이터 처리 장치를 제어할 수 있다. 컴퓨터 프로그램은 위에서 논의된 명령어 디코딩 회로부의 기능을 에뮬레이션하는 명령어 디코딩 프로그램 로직을 가질 수 있다. 예를 들어, 명령어 디코딩 프로그램 로직은, 타겟 코드의 주어진 명령어에 응답하여, 호스트 데이터 처리 장치의 고유 명령어 세트에서 대응하는 코드 시퀀스를 생성하여, 디코딩된 명령어에 대응하는 기능을 수행하도록 호스트 데이터 처리 장치를 제어한다. 명령어 디코딩 프로그램 로직은 서브벡터 입도에서 주어진 벡터에 대한 연산을 수행하도록 호스트 데이터 처리 장치를 제어하기 위해, 위에서 논의된 바와 같이 서브벡터 지원 명령어를 디코딩하기 위한 서브벡터 지원 명령어 디코딩 프로그램 로직을 포함한다. 그러한 시뮬레이션 프로그램은, 예를 들어, 하나의 명령어 세트 아키텍처에 대해 작성된 레거시 코드가 상이한 명령어 세트 아키텍처를 지원하는 호스트 프로세서 상에서 실행되고 있을 때, 유용할 수 있다. 또한, 시뮬레이션은 그 새로운 아키텍처 버전을 지원하는 처리 하드웨어가 준비되기 전에 더 새로운 버전의 명령어 세트 아키텍처에 대한 소프트웨어 개발이 시작될 수 있게 할 수 있는데, 왜냐하면 시뮬레이션된 실행 환경 상에서의 소프트웨어의 실행은 새로운 아키텍처를 지원하는 하드웨어 디바이스의 진행 중인 개발과 병행하여 소프트웨어의 테스트를 가능하게 할 수 있기 때문이다. 시뮬레이션 프로그램은 비밀시적 저장 매체일 수 있는 저장 매체에 저장될 수 있다.
- [0053] 이제 특정 예들이 도면을 참조하여 설명된다. 청구항들은 이러한 특정 예들로 제한되지 않는다는 것이 인식될 것이다.
- [0054] 도 1은 데이터 처리 장치(2)의 예를 개략적으로 예시한다. 데이터 처리 장치는 다수의 파이프라인 스테이지를 포함하는 처리 파이프라인(4)을 갖는다. 이 예에서, 파이프라인 스테이지들은 명령어 캐시(8)로부터 명령어들을 폐지하기 위한 폐지 스테이지(6); 파이프라인의 나머지 스테이지들에 의해 처리될 마이크로 연산들(디코딩된 명령어들)을 생성하기 위해 폐지된 프로그램 명령어들을 디코딩하기 위한 디코드 스테이지(10)(명령어 디코딩 회로부의 예); 마이크로 연산들을 위해 요구되는 피연산자들이 레지스터들(14)에서 이용 가능한지를 체크하고, 일단 주어진 마이크로 연산을 위해 요구되는 피연산자들이 이용 가능하면 실행을 위한 마이크로 연산들을 발행하기 위한 발행 스테이지(12); 레지스터들(14)로부터 판독된 피연산자들을 처리하여 결과 값들을 생성함으로써, 마이크로 연산들에 대응하는 데이터 처리 연산들을 실행하기 위한 실행 스테이지(16)(처리 회로부의 예); 및 처리의 결과들을 레지스터들(14)에 후기입하기 위한 후기입 스테이지(18)를 포함한다. 이것은 단지 가능한 파이프라인 배열의 일례일 뿐이고, 다른 시스템들은 추가적인 스테이지들 또는 스테이지들의 상이한 구성을 가질 수 있다는 것이 인식될 것이다. 들어, 비순차적 프로세서에서, 프로그램 명령어들 또는 마이크로 연산들에 의해 지정된 아키텍처 레지스터들을 레지스터들(14)에서 물리적 레지스터들을 식별하는 물리적 레지스터 지정자들에 매핑하기 위한 레지스터 재명명 스테이지가 포함될 수 있다. 몇몇 예에서, 디코드 스테이지(10)에 의해 디코딩된 프로그램 명령어들과 실행 스테이지에 의해 처리된 대응하는 마이크로 연산들 사이에 일대일 관계가 있을 수 있다. 프로그램 명령어들과 마이크로 연산들 사이에 일대다 또는 다대일 관계가 있는 것이 또한 가능하며, 따라서, 예를 들어, 단일 프로그램 명령어가 2개 이상의 마이크로 연산으로 분할될 수 있거나, 2개 이상의 프로그램 명령어가 단일 마이크로 연산으로서 처리되도록 융합될 수 있다.
- [0055] 실행 스테이지(16)는 상이한 부류들의 처리 연산을 실행하기 위한 다수의 처리 유닛을 포함한다. 예를 들어, 실행 유닛들은 스칼라 처리 유닛(20)(예컨대, 레지스터들(14)로부터 판독된 스칼라 피연산자들에 대해 산술 또는 논리 연산들을 수행하기 위한 스칼라 산술/로직 유닛(ALU)(20)을 포함함); 다수의 벡터 요소들을 포함하는 벡터들에 대해 벡터 연산들을 수행하기 위한 벡터 처리 유닛(22); 및 메모리 시스템(8, 30, 32, 34) 내의 데이터에 액세스하기 위해 로드/저장 연산들을 수행하기 위한 로드/저장 유닛(28)을 포함할 수 있다. 실행 스테이지에서 제공될 수 있는 처리 유닛들의 다른 예들은 부동 소수점 포맷으로 표현된 값들을 수반하는 연산들을 수행하기 위한 부동 소수점 유닛, 또는 분기 명령어들을 처리하기 위한 분기 유닛을 포함할 수 있다.
- [0056] 레지스터들(14)은 스칼라 값들을 저장하기 위한 스칼라 레지스터들(25), 벡터 값들을 저장하기 위한 벡터 레지스터들(26), 및 술어 값들을 저장하기 위한 술어 레지스터들(27)을 포함한다. 술어 값들(27)은 벡터 명령어들을 처리할 때 벡터 처리 유닛(22)에 의해 사용될 수 있으며, 이때 주어진 술어 레지스터 내의 술어 값은 벡터 레지스터들(26)에 저장된 대응하는 벡터 피연산자의 어느 벡터 요소들이 활성 벡터 요소들인지 또는 비활성 벡

터 요소들인지를 표시한다(여기서 비활성 데이터 요소들에 대응하는 연산들은 억제될 수 있거나, 벡터 명령어에 응답하여 벡터 처리 유닛(22)에 의해 생성된 결과 값에 영향을 미치지 않을 수 있음).

[0057] 메모리 관리 유닛(MMU)(36)은, 메모리 시스템에 저장된 페이지 테이블 구조에 정의된 어드레스 매핑들에 기초하여, (페치 회로부(6)로부터의 명령어 페치들 또는 로드/저장 유닛(28)으로부터의 로드/저장 요청들에 의해 지정된) 가상 어드레스들과 메모리 시스템 내의 위치들을 식별하는 물리적 어드레스들 사이의 어드레스 변환들을 제어한다. 페이지 테이블 구조는 또한, 예를 들어 어드레스 공간의 영역들이 판독 전용인지 또는 판독가능/기입 가능인지를 지정하고, 어느 특권 레벨들이 영역에 액세스하도록 허용되는지를 지정하고, 그리고/또는 어드레스 공간의 대응하는 영역이 액세스될 수 있는 방법을 좌우하는 다른 특성들을 지정하는, 어드레스 공간의 대응하는 페이지들에 액세스하기 위한 액세스 허가들을 지정할 수 있는 메모리 속성들을 정의할 수 있다. 페이지 테이블 구조로부터의 엔트리들은 페이지 테이블 엔트리들 또는 메모리에 제시된 페이지 테이블 구조로부터의 페이지 테이블 엔트리들에 대한 액세스를 가속화하기 위한 다른 정보를 캐싱하기 위해 MMU(36)에 의해 유지되는 캐시인 변환 색인 버퍼(translation lookaside buffer, TLB)(38)에 캐싱될 수 있다.

[0058] 이 예에서, 메모리 시스템은 레벨 1 데이터 캐시(30), 레벨 1 명령어 캐시(8), 공유 레벨 2 캐시(32) 및 메인 시스템 메모리(34)를 포함한다. 이것은 가능한 메모리 계층 구조의 일례일 뿐이며, 캐시들의 다른 배열들이 제공될 수 있다는 것이 인식될 것이다. 실행 스테이지(16)에 도시된 특정 유형의 처리 유닛(20 내지 28)은 일례일 뿐이며, 다른 구현들은 상이한 세트의 처리 유닛들을 가질 수 있거나, 동일 유형의 다수의 마이크로 연산이 병렬로 핸들링될 수 있도록 동일 유형의 처리 유닛의 다수의 인스턴스를 포함할 수 있다. 도 1은 가능한 프로세서 파이프라인 배열의 몇몇 컴포넌트들의 단순화된 표현일 뿐이며, 프로세서는 간결함을 위해 예시되지 않은 많은 다른 요소를 포함할 수 있다는 것이 인식될 것이다.

[0059] 처리 파이프라인(4)은 스케일링 가능한 벡터 ISA를 지원하며, 이는 ISA의 벡터 명령어가 그것이 컴파일 시간(실행된 명령어가 컴파일러에 의해 컴파일된 시간)에 그러한 벡터 명령어의 실행에 어떤 벡터 길이가 사용될지를 알지 않고서 처리될 수 있다는 것을 의미한다. 이것은 상이한 최대 벡터 길이들을 지원하는 다양한 상이한 처리 장치들이 동일한 소프트웨어 코드를 실행하는 것을 가능하게 하여, 다양한 처리 플랫폼에 걸쳐 실행하기에 적합한 소프트웨어 코드를 생성함에 있어서 소프트웨어 개발자에 대한 부담을 회피한다. 이것은 주어진 처리 장치의 하드웨어 설계자가 구현되는 최대 벡터 길이를 설계자의 선호되는 성능/전력 요건에 따라 자유롭게 선택한다는 것을 의미한다(더 높은 성능을 목표로 하는 시스템은 더 나은 에너지 효율을 목표로 하는 시스템보다 더 긴 최대 벡터 길이를 선택할 수 있음).

[0060] 도 2는 스케일링 가능한 벡터 ISA에 대한 아키텍처 상태의 예를 예시한다. ISA는, 벡터 레지스터들(26)로서, 다수의 스케일링 가능한 벡터 레지스터(예를 들어, 32개의 벡터 레지스터, Z0 내지 Z31)를 정의하며, 이들은, 주어진 하드웨어 구현에 대해, 소정의 아키텍처 정의 최대 길이까지의 소정 단위 크기의 임의의 배수일 수 있는 구현 선택 벡터 길이를 갖는다. 예를 들어, 이 예에서 단위 크기는 128비트이고, 아키텍처는 128비트 내지 2048비트의 128비트의 임의의 배수인 최대 벡터 길이를 지원한다(즉, 주어진 프로세서에 대한 벡터 길이는 $LEN * 128$ 비트이며, 여기서 $1 \leq LEN \leq 16$ 임). 이것은 레지스터가, 말하자면, 128비트의 고정된 아키텍처 정의 벡터 길이를 가질 수 있는 스케일링 불가능한 벡터 아키텍처와는 상이하다(도 2에 V0 내지 V31로 라벨링된 레지스터들 참조). 도 2에 도시된 예는 Arm® Limited에 의해 제공되는 "스케일링 가능한 벡터 확장"(SVE)인 스케일링 가능한 벡터 ISA, 및 Arm® Limited에 의해 제공되는 "진보된 SIMD" 아키텍처(Neon™)인 스케일링 불가능한 벡터 ISA에 기초한다. 스케일링 가능한 벡터 ISA에 따라 128비트보다 큰 벡터 길이를 지원하더라도, 명령어 디코더(10) 및 처리 회로부(16)는 또한 역호환성 이유로 스케일링 불가능한 벡터 ISA를 지원할 수 있고, 스케일링 불가능한 벡터 ISA의 명령어는 고정 길이(예를 들어, 128비트)의 레지스터 V0-V31을 참조할 수 있다 - 실제로 이러한 레지스터들은 스케일링 가능한 벡터 ISA의 명령어에 액세스 가능한 더 긴 레지스터들의 일부를 사용하여 표현될 수 있으며, 따라서 스케일링 가능한 벡터 레지스터 Z0-Z31 및 스케일링 불가능한 벡터 레지스터 V0-V31은 하드웨어 저장 회로부를 공유할 수 있다.

[0061] 물론, 다른 예는 상이한 스케일링 가능한 그리고 스케일링 불가능한 아키텍처들에 기초할 수 있으며, 따라서 스케일링 가능한 벡터 길이에 대한 선택에 이용 가능한 크기들의 범위 및 스케일링 불가능한 벡터 길이에 대해 지정된 고정 크기는 도 2에 도시된 것과는 상이할 수 있다.

[0062] 벡터 레지스터(26)뿐만 아니라, 벡터 레지스터(26)를 사용하여 제공된 벡터 피연산자의 벡터 요소에 대해 수행되는 연산을 선택적으로 마스킹하는 데 사용되는 술어 값을 저장하기 위한 다수의 술어 레지스터(27)(P0 내지 P15로 라벨링됨)가 제공된다. 술어 레지스터(27)는 하드웨어에서 지원되는 최대 벡터 길이가 사용될 때 ISA에

의해 지원되는 최소 벡터 요소 크기에 따라 정의될 때 벡터 레지스터(26)에서 벡터 요소당 하나의 비트를 가질 수 있다(최대 벡터 길이는 여기서 임의의 하드웨어 구현에 대해 아키텍처에 의해 허용되는 최대 벡터 길이(예를 들어, 2048비트)가 아니라, 특정 하드웨어 구현에 의해 사용되는 선택된 크기 $LEN * 128$ 을 지칭한다). 예를 들어, 최소 요소 크기가 8비트인 경우, $LEN * 128$ 비트 벡터 레지스터의 예에 대해, 각각의 술어 레지스터는 크기 $LEN * 16$ 비트를 가질 수 있다. 도 3은 요소별로 벡터에 적용되는 술어의 예를 도시한다. 명령어는 가변 요소 크기를 정의할 수 있으며, 예를 들어 도 3은 256비트 벡터가 논리적으로 64비트 또는 32비트 요소로 분할될 수 있는 방법을 보여준다(다른 요소 크기가 또한 가능할 수 있음, 예를 들어 16비트 또는 8비트 요소). 벡터 레지스터(26)에 저장된 주어진 벡터에 대해 지정된 술어 레지스터(27)는 최소 요소 크기가 사용될 때 각각의 벡터 요소에 대해 별개의 술어 비트를 지정할 수 있기에 충분한 술어 비트들의 수를 지정하지만, 더 큰 요소 크기에서 모든 술어 비트가 판독될 필요는 없으며 대신에 술어 비트는 요소 크기에 대응하는 간격에서 판독될 수 있다. 예를 들어, 64비트 요소에 대해, 도 3의 상부 예에서 도시된 바와 같이 8개의 술어 비트마다 하나씩 판독될 수 있는 반면, 32비트 요소에 대해, 4개의 술어 비트마다 하나씩 판독될 수 있다(8비트 요소 크기에서, 각각의 술어 비트가 판독될 수 있음). 술어 비트가 1로 설정될 때, 대응하는 요소는 활성화로 간주되는 반면, 대응하는 술어 비트가 0으로 설정된 벡터 요소는 비활성으로 간주된다. 비활성 벡터 요소에 대응하는 결과 벡터의 요소는 벡터 연산의 결과에 기초하여 업데이트되는 것으로부터 마스킹될 수 있다(술어 명령어는 마스킹된 결과 요소가 0으로 설정되는 제로화 술어 변형(zeroing-predication variant), 또는 마스킹된 결과 요소가 벡터 명령어가 실행되기 전에 목적지 레지스터 내의 대응하는 요소와 동일한 값으로 설정되는 병합 술어 변형을 포함할 수 있음).

[0063] 도 2에 도시된 바와 같이, 레지스터들(14)은 또한 몇몇 제어 레지스터들(29)(ZCR로 라벨링됨)을 포함할 수 있으며, 이들은 더 적은 특권을 가진 상태에서 실행되는 소프트웨어에 의해 사용 가능한 최대 벡터 길이를 제한하기 위해 더 많은 특권을 가진 소프트웨어에 의해 사용될 수 있다. 예를 들어, 전력을 절약하기 위해 주어진 소프트웨어는 그것이 하드웨어에서 지원되는 전체 벡터 길이를 이용할 수 없도록 제한될 수 있다. 그럼에도 불구하고, 더 많은 특권을 가진 소프트웨어가 벡터 길이에 제한을 적용하더라도, 애플리케이션 소프트웨어에 대한 벡터 길이는 컴파일 시간에 여전히 알려지지 않는데, 왜냐하면 특정 프로세서에서의 실제 구현된 벡터 길이가 제어 레지스터(29)에 정의된 제한보다 크거나 작을지가 알려지지 않을 것이기 때문이다(제어 레지스터(29)에 정의된 제한보다 작은 최대 벡터 길이를 갖는 구현에 대해, 제한에 의해 표시된 것보다 작은 벡터 길이가 사용될 것임).

[0064] 스케일링 가능한 벡터 ISA의 벡터 길이 불가지론 특성은 유용한데, 왜냐하면 ISA의 명령어를 인코딩하는 데 이용 가능한 고정 인코딩 공간 내에서, 상대적으로 작은 에너지 효율적인 마이크로컨트롤러로부터 서버 및 다른 고성능 컴퓨팅 시스템으로 스케일링하는 광범위한 요건을 고려할 때, 프로세서 설계자에 의해 요구될 수 있는 모든 상이한 벡터 길이에 대한 상이한 명령어들을 생성하는 것이 실현 가능하지 않기 때문이다. 컴파일 시간에 알려진 고정 벡터 길이를 갖지 않는 것에 의해, 각각의 성능/전력/면적 포인트에 코드를 맞추려고 하는 소프트웨어 개발자로부터의 노력 없이, 동일한 ISA를 사용하여 다수의 시장이 어드레스될 수 있다.

[0065] 스케일링 가능한 벡터 ISA의 스케일링 가능한 특성을 달성하기 위해, 스케일링 가능한 벡터 ISA의 벡터 명령어의 기능은 (하드웨어에서 지원되는 최대 벡터 길이 및 제어 레지스터(29)를 사용하는 임의의 소프트웨어 정의 제한을 고려할 때) 사용 중인 벡터 길이를 나타내는 파라미터(예를 들어, 도 2에 도시된 바와 같은 LEN)를 참조하여 아키텍처에서 정의되며, 여기서 그 파라미터 LEN 은 컴파일 시간에 알려지지 않는다. 따라서, 상이한 시스템들에서의 동일한 벡터 명령어의 실행은 상이한 결과들을 생성할 수 있다(전형적으로, 생성된 벡터 요소들의 수의 면에서 다름, 그의 서브세트는 상이한 플랫폼들에서 동일한 결과 값들을 가질 수 있지만, 일반적으로 더 큰 벡터 길이를 구현하는 플랫폼들은 더 작은 벡터 길이를 구현하는 플랫폼에 비해 추가적인 벡터 요소를 생성할 수 있음). 술어는 명령어의 주어진 인스턴스에서 어떤 요소가 생성되는지를 제어하는 데 사용될 수 있으며, 예를 들어 비교 명령어를 사용하여 특정 루프 반복에 대한 술어의 값을 자동으로 생성하거나 상이한 벡터 길이들로 스케일링할 수 있는 어떤 일반 정의 술어 패턴을 적용함으로써, 벡터 길이 불가지론 원리에 기초하여 설정될 수 있다. 소정 명령어는 요소 카운트 값과 같은 루프 제어 파라미터를 업데이트하여 지금까지 얼마나 많은 벡터 요소가 처리되었는지를 추적할 수 있으며, 따라서 루프 전체에 걸쳐 더 넓은 벡터 길이를 갖는 구현 및 좁은 벡터 길이를 갖는 구현 둘 모두가 결국 동일한 결과 그러나 상이한 성능 레벨을 갖는 결과를 달성할 수 있는데, 왜냐하면 더 넓은 벡터 길이를 갖는 구현은 더 좁은 벡터 길이를 갖는 구현보다 더 적은 루프 반복을 요구할 수 있기 때문이다.

[0066] 도 4는 상이한 벡터 길이를 사용하는 상이한 하드웨어 구현들에서 동일한 명령어 시퀀스가 어떻게 상이하게 처

리될 수 있는지를 보여주는 실시예를 도시한다. 이 예에서, 코드 예는 배정밀도 부동 소수점 연산 'daxpy'(배정밀도 Ax 플러스 y)를 수행하여 입력 값들의 2개의 어레이 $x[i]$, $y[i]$ 에 대해 $y[i] = a \cdot x[i] + y[i]$ 를 계산하는 것이며, 여기서 $0 \leq i < n$ 이다. 이것은 다음과 같이 스케일링 가능한 벡터 ISA로부터의 명령어들의 세트를 사용하여 벡터화된 루프에서 구현될 수 있다:

```

1 // x0 = &x[0], x1 = &y[0], x2 = &a, x3 = &n
2 daxpy_:
3     ldrsw x3, [x3]           // x3=*n
4     mov x4, #0              // x4=i=0
5     whilelt p0.d, x4, x3     // p0=while(i++<n)
6     ld1rd z0.d, p0/z, [x2]   // p0:z0=broadcast(*a)
7 .loop:
8     ld1d z1.d, p0/z, [x0, x4, lsl #3] // p0:z1=x[i]
9     ld1d z2.d, p0/z, [x1, x4, lsl #3] // p0:z2=y[i]
10    fmla z2.d, p0/m, z1.d, z0.d // p0?z2+=x[i]*a
11    st1d z2.d, p0, [x1, x4, lsl #3] // p0?y[i]=z2
12    incd x4                  // i+=(VL/64)
13.latch:
14    whilelt p0.d, x4, x3     // p0=while(i++<n)
15    b.first .loop           // more to do?
16 ret

```

[0067]

[0068] 도 4는 128비트 벡터 길이를 사용하는 하나의 예와 256비트 벡터 길이를 사용하는 다른 예에 대한 이러한 명령어의 처리를 도시한다. 변수 a 와 어레이 $x[]$ 및 $y[]$ 는 메모리에 저장된 데이터이다. 레지스터 $x0$ - $x4$ 는 스칼라 레지스터이다. 레지스터 $p0$ 은 술어 레지스터이다. 레지스터 $z0$ - $z2$ 는 제1 예에서 128비트 레지스터이고 제2 예에서 256비트 레지스터인 벡터 레지스터이다(따라서, 술어 레지스터 $p0$ 은 128비트 예에 비해 256비트 예에서 더 긴 레지스터이다). 실선 윤곽 상자는 주어진 명령어에 의해 처리되거나 설정된 활성 요소를 나타내는 반면, 파선 윤곽 상자는 술어에 의해 마스킹된 비활성 요소 또는 주어진 명령어에 전혀 사용되지 않는 레지스터를 나타낸다. 도 4에 도시된 명령어 넘버링은 처리 회로부(16)에 의해 실행되는 디코딩된 명령어들의 시퀀스의 프로그램 순서를 나타낸다는 점에 유의한다(이는 위에 제시된 컴파일된 코드의 넘버링과는 상이한데, 왜냐하면 몇몇 명령어들이 여러 번 실행될 수 있고 위의 예에서 넘버링된 분기점 라벨들이 도 4에서는 넘버링되지 않기 때문이다). 그럼에도 불구하고, 위에 제시된 바와 같이, 도 4에는 동일한 명령어 시퀀스가 도시된다.

[0069]

128비트 예 및 256비트 예 둘 모두에서, 벡터 명령어는 대응하는 벡터 길이 내에 맞춰질 수 있는 벡터 요소들의 수로 스케일링한다. 이 예에서, 사용되는 요소 크기는 64비트이며, 따라서 128비트 예에는 벡터당 2개의 벡터 요소가 있고 256비트 예에는 벡터당 4개의 벡터 요소가 있다. 따라서, 128비트 예에서는 $p0$ 의 2개의 술어 비트만이 사용되고 256비트 예에서는 $p0$ 의 4개의 술어 비트만이 사용된다(실제로, 레지스터 $p0$ 은 더 작은 벡터 요소 크기가 지원되는 것을 가능하게 하기 위해 더 많은 수의 술어 비트를 포함할 수 있음).

[0070]

`whilelt`, `incd` 및 `b.first` 명령어는 술어를 설정하고 지원되는 벡터 길이에 따라 루프 카운트 값을 업데이트하고 루프 카운트 값의 비교에 따라 조건부로 루프의 시작 부분으로 다시 분기하는 루프 제어 명령어이다. 다른 명령어는 로드/저장 명령어, 또는 지원되는 벡터 길이에 따라 다수의 벡터 요소를 처리하는 벡터 처리 명령어이다.

[0071]

증가 명령어 `incd`는 대응하는 루프 반복에서 처리된 벡터 요소들의 수에 기초하여 루프 카운터를 증가시킨다(예를 들어, VL/ES , 여기서 VL 은 벡터 길이이고 ES 는 벡터 요소 크기임). 술어 설정 명령어 `whilelt`는 루프 카운터(i , 명령어 시퀀스에서 레지스터 $x4$ 에 의해 표현됨)와 종료 제한(n , 위의 코드 예의 라인 3에서 로드 명령어에 의해 메모리로부터 로드되는 레지스터 $x3$ 에 의해 표현됨) 간의 비교를 사용하여, 루프 카운터의 다양한 증가

버전이 제한 값(1t)보다 작은지 여부에 따라 술어를 설정한다. 증가된 루프 카운터가 여전히 제한 값보다 작은 그러한 요소에 대해, 술어는 참으로 설정되는 반면, 일단 증가된 루프 카운터가 종료 제한에 도달하면 술어는 거짓으로 설정된다(이 예에서, 이것은 256비트 예에서 명령어 2에 대해 처리된 제4 요소에 대해 발생하지만, 128비트 예에서의 2-요소 벡터에 대해, 이러한 제한은 아직 첫 번째 패스에서 도달되지 않았고, 처리할 다른 벡터 요소가 여전히 있는지를 결정하기 위해 실행되는 후속 술어 설정 명령어(위의 코드 예에서 라인 14에 제시된 whilelt 명령어)에 의해서만 도달될 것이며(도 4의 128 비트 예에 도시된 디코딩된 프로그램 순서에서의 9번째 명령어 참조), 따라서 분기 명령어(디코딩된 프로그램 순서에서의 명령어 10)는 다른 패스에 대해 루프백한다). 다음 패스에서, whilelt 명령어(디코딩된 프로그램 순서에서의 16번째 명령어)는 이제 모든 술어가 "거짓"이라고 결정하며, 따라서 이번에는 분기 명령어가 분기를 취하지 않으며 따라서 루프는 종료된다. 대조적으로, 256 비트 예에 대해, 각각의 벡터 레지스터에서 4번째 요소의 효과를 마스킹하는 하나의 패스만이 필요했다.

[0072] 궁극적으로 명령어 시퀀스에 의해 생성된 결과가 128비트 예 및 256비트 예 둘 모두에 대해 얼마나 동일한지에 주목한다(어레이 y[]에 대해 메모리에 저장된 데이터는 동일한 값 3, 50, 41, 32(최상위 어드레스로부터 최하위 어드레스까지)를 갖지만, 128비트 예는 두 번의 루프 반복에서 그의 결과를 획득한 반면, 256비트 예는 한 번의 루프 반복만을 필요로 함). 따라서, 위의 코드 예에서 라인 8-15에서의 명령어는 128비트 예에서 두 번 디코딩되고 실행되도록 요구되었다(도 4에서의 128비트 예에 대한 디코딩된 프로그램 순서에 넘버링된 명령어 4 내지 17 참조).

[0073] 스케일링 가능한 벡터 ISA는 다양한 상이한 하드웨어 구현을 지원하는 플랫폼 독립적인 코드가 개발될 수 있게 하는 데 매우 유용할 수 있지만, 그럼에도 불구하고 사용 중인 많은 소프트웨어 프로그램은 Arm® Limited의 Neon™ 스케일링 불가능한 벡터 ISA를 따르는 시스템을 위해 제공되는 128비트 벡터 단위와 같은 고정 길이 벡터 단위를 지원하는 스케일링 불가능한 벡터 ISA에 대해 특별히 최적화되었다. 역호환성을 위해, 스케일링 가능한 벡터 ISA를 지원하는 시스템은 또한 그러한 레거시 소프트웨어가 여전히 실행될 수 있는 것을 보장하기 위해 스케일링 불가능한 벡터 ISA의 명령어를 지원할 수 있다. 그러나, 이것은 하드웨어가 더 큰 벡터 길이를 지원하더라도, 스케일링 불가능한 소프트웨어는 하드웨어에서 이용 가능할 추가 성능으로부터 이익을 얻을 수 없다는 것을 의미한다.

[0074] 스케일링 불가능한 벡터 ISA에 작성된 프로그램 코드가 스케일링 가능한 벡터 ISA의 명령어를 사용하여 재개발되는 것이 바람직할 수 있는데, 그 이유는 이것이 스케일링 가능한 벡터 ISA의 많은 하드웨어 구현에서 이용 가능한 더 긴 벡터 길이를 이용하여 더 큰 성능 기회를 열 수 있기 때문이다. 그러나, 스케일링 불가능한 벡터 ISA(여기서 벡터 길이는 컴파일 시간에 알려짐)를 위해 작성된 프로그램을 스케일링 가능한 벡터 ISA(여기서 벡터 길이는 컴파일 시간에 알려지지 않음)를 위해 작성된 프로그램 코드로 재작성하는 것이 시도되는 경우, 이것은 프로그래머 또는 컴파일러 작성자에게 상당한 노력을 요구할 수 있다. 스케일링 불가능한 벡터 ISA의 명령어는 간단한 방식으로는 스케일링 가능한 벡터 명령어에 매핑되지 못할 수 있는데, 이는 종종 스케일링 불가능한 ISA에서의 프로그램 코드가 컴파일 시간에 알려진 고정 벡터 길이(예를 들어, 128비트)에 대한 지식에 따라 선택된 특정 최적화를 포함할 수 있기 때문이며, 컴파일 시간에 어떤 벡터가 다음에 사용될지를 알 수 없는 경우, 이것은 몇몇 그러한 최적화의 사용을 방해할 수 있으며, 이는 몇몇 경우들에서 스케일링 가능한 벡터화된 코드에서의 벡터화된 루프의 사용을 완전히 방해할 수 있다.

[0075] 예를 들어, 다음의 C 코드는 부분 합 감소를 구현할 수 있다(예를 들어, 이것은 디지털 신호 처리 애플리케이션으로부터의 연산일 수 있음):

```

for (int biquad = 0; sample < num_samples; ++sample) {
    tmp = acc;
    acc += (((int32_t) coeff[biquad].b1 * state[biquad].x1 +
            (int32_t) coeff[biquad].b2 * state[biquad].x2) >> 14) +
            (((int32_t) coeff[biquad].a1 * state[biquad].y1 +
            (int32_t) coeff[biquad].a2 * state[biquad].y2) >> 14);
    state[biquad].x2 = state[biquad].x1;
    state[biquad].x1 = tmp;
    state[biquad].y2 = state[biquad].y1;
    state[biquad].y1 = acc;
}

```

[0076]

[0077]

위의 코드에서, 각각의 반복에서의 곱셈-덧셈 연산들의 결과는 acc에서 누산되고 state[biquad].y1에 저장된다. 벡터화된 후에, state[biquad].y1의 값을 저장하는 데 사용되는 벡터의 각각의 요소는 누산의 부분 합을 저장해야 한다(요소 0은 0번째 반복에서 acc의 값을 저장해야 하고, 요소 1은 처음 두 번의 반복 후에 acc의 결과적인 값을 저장해야 하고, 등등이다).

[0078]

스케일링 불가능한 벡터 ISA에서, 이러한 부분 합 감소는, 아래에 나타낸 바와 같이, 4개의 요소 각각을 새로운 벡터 레지스터로 브로드캐스팅하고 mla(multiply-accumulate, 곱셈-누산) 명령어를 사용하여 이러한 4개의 벡터 레지스터를 부분적으로 누산함으로써 달성될 수 있다:

```

...
dup v17.4h, v16.h[0]
dup v18.4h, v16.h[1]
dup v19.4h, v16.h[2]
dup v20.4h, v16.h[3]
add v6.4h, v6.4h, v17.4h
mla v6.4h, v18.4h, v29.4h
mla v6.4h, v19.4h, v30.4h
mla v6.4h, v20.4h, v31.4h

```

[0079]

[0080]

여기서, 첫 번째 복제(dup) 명령어는 벡터 레지스터 v17의 모든 벡터 요소를 벡터 레지스터 v16의 요소 0과 동일하게 설정하고, 두 번째 dup 명령어는 벡터 레지스터 v18의 모든 벡터 요소를 벡터 레지스터 v16의 요소 1과 동일하게 설정하고, 등등이다. 레지스터 v16의 벡터 요소당 하나의 dup가 있다. 스케일링 불가능한 벡터 ISA에서의 벡터 길이는 컴파일 시간에 알려지기 때문에, 벡터 요소의 크기가 주어지면, 컴파일러는 포함할 dup 명령어들의 수를 계산할 수 있다(이 예에서는 128비트 벡터당 4개의 32비트 요소에 기초하여 4개).

[0081]

그러나, 스케일링 가능한 벡터 ISA에서 이러한 부분 합 감소는 동일한 방식으로 수행될 수 없는데, 왜냐하면 컴파일 시간에 벡터 길이가 알려지지 않고 그에 따라 컴파일러가 얼마나 많은 dup 명령어가 포함되어야 하는지를 모르기 때문이다. 이것은 C 코드에 정의된 스칼라 루프의 성공적인 벡터화를 방해할 수 있으며, 스케일링 가능한 벡터 코드가 스칼라 루프에 의존하도록 강요하여 벡터화의 이익이 실현될 수 없게 할 수 있다.

[0082]

이것은 단지 벡터 길이에 대한 컴파일 시간 지식에 의존할 수 있는 소프트웨어 최적화의 일례일 뿐이다. 다른 예는 소정 수의 원래 루프 반복을 더 많은 수의 명령어를 각각 포함하는 더 적은 수의 루프 반복에 매핑함으로써 실행될 필요가 있는 루프 제어 명령어들의 수를 감소시키는 루프 언롤링 - "언롤링된" 루프의 하나의 반복은

원래 루프의 다수의 반복에 대응함 -; 및 컴파일러가 루프의 명령어들의 실행을 재정리하여 나중 루프 반복의 몇몇 명령어가 이전 루프 반복으로부터의 명령어보다 먼저 실행될 수 있게 하는 소프트웨어 파이프라이닝을 포함할 수 있다.

[0083] 도 5는 이러한 문제에 도움이 될 수 있는 기술의 예를 개략적으로 예시한다. (컴파일 시간에 알려지지 않은) 스케일링 가능한 벡터 길이를 갖는 벡터는 논리적으로 스케일링 가능한 수의 서브벡터로 구성되는 것으로 취급되며, 각각의 서브벡터는 동일한 서브벡터 길이를 갖는다. 서브벡터 길이는 컴파일 시간에 알려진다. 예를 들어, 각각의 서브벡터는 Neon™과 같은 스케일링 불가능한 벡터 아키텍처에서 고정 벡터 길이에 대응하는 고정 크기(예를 들어, 128비트)일 수 있거나, (기존의 스케일링 불가능한 벡터 ISA와의 임의의 대응관계에 관계없이) 단순히 스케일링 가능한 벡터 ISA에 정의된 임의적인 고정 크기일 수 있다. 대안적으로, 스케일링 가능한 벡터 ISA의 몇몇 구현들에서, 소프트웨어가 2개 이상의 선택사항 중에서 서브벡터 길이를 지정하는 능력이 있을 수 있으며, 따라서 서브벡터 길이는 컴파일 시간에 알려지지만(소프트웨어에 의해 정의되지만), 이러한 서브벡터 접근법을 이용하는 명령어의 상이한 인스턴스들에 대해 다를 수 있다. 서브벡터 길이가 컴파일 시간에 알려지지만, 전체 벡터 길이는 알려지지 않기 때문에, 벡터 내에 맞춰지는 서브벡터들의 총 수가 또한 컴파일 시간에 알려지지 않으며, 서브벡터 접근법을 지원하도록 설계된 명령어를 실행하는 특정 하드웨어 구현에 대해 선택된 특정 벡터 길이에 따라 스케일링 가능할 수 있다. 각각의 서브벡터는 가변적인 수의 벡터 요소를 가지며, 각각의 벡터 요소의 크기는 가변적이고 다수의 선택사항(예를 들어, 8, 16, 32 또는 64비트)으로부터 소프트웨어에 의해 선택될 수 있다. 따라서, 서브벡터당 벡터 요소들의 크기 및 수가 컴파일 시간에 알려질 수 있지만, 전체 벡터 내의 효과 요소들의 수는 가변적이고 컴파일 시간에 알려지지 않을 수 있다. 서브벡터 길이는 하드웨어에 의해 사용되는 전체 벡터 길이 및 주어진 명령어에 대해 지정된 벡터 요소 크기와는 무관할 수 있다.

[0084] 이러한 접근법의 경우, 서브벡터 길이가 알려지기 때문에, (고정 벡터 길이를 가정하는) 스케일링 불가능한 벡터 ISA와 함께 사용하도록 컴파일된 코드를 (알려진 서브벡터 길이의 서브벡터를 가정하지만 스케일링 가능한 전체 벡터 길이를 허용하는 서브벡터 지원 명령어를 사용하는) 스케일링 가능한 벡터 ISA와 함께 사용하도록 컴파일된 코드로 변환하는 것이 훨씬 더 간단해진다. 또한, (스케일링 불가능한 벡터화된 코드부터 시작함이 없이) 서브벡터 지원 명령어를 지원하는 스케일링 가능한 벡터 ISA에 대해 직접 컴파일하더라도, 서브벡터 지원 명령어는 위에서 논의된 것과 같은 소프트웨어 성능 개선 기술이 적용될 수 있게 하는 데 유용할 수 있으며, 이는 그렇지 않으면 고정 크기의 서브벡터의 입도에서보다는, 스케일링 가능한 길이의 벡터에서 요소별로 또는 전체 벡터 입도에서 동작하는 스케일링 가능한 벡터 명령어에 대해 가능하지 않을 것이다.

[0085] 따라서, 아래의 예들에서 논의되는 바와 같이, 개별 요소의 입도 또는 전체 벡터의 입도에서보다는 서브벡터의 입도에서 연산을 수행하도록 처리 회로부(16)(예컨대, 벡터 처리 유닛(22) 및/또는 로드/저장 유닛(28))을 제어하는 다수의 서브벡터 지원 명령어가 정의될 수 있다. 서브벡터의 입도에서 수행되는 연산들은 서브벡터 지원 명령어의 실행의 단일 인스턴스에 응답하여 병렬로, 순차적으로, 일부는 병렬로 그리고 일부는 순차적으로, 또는 파이프라인 방식으로 수행될 수 있다(따라서, 서브벡터 지원 명령어를 실행하는 각자의 인스턴스들 사이에 설정된 슬롯을 사용하여 벡터를 서브벡터들 - 각각의 서브벡터는 별개의 패스에서 서브벡터 지원 명령어를 통해 처리됨 - 로 분할하는 것이 필요하지 않다).

[0086] 벡터 피연산자들에 대해 수행되도록 요구될 수 있는 모든 벡터 연산에 대응하는 서브벡터 지원 명령어를 제공하는 것은 필수적이지 아니다. 많은 연산(예를 들어, 덧셈 또는 곱셈)이 요소별 입도에서 적용될 수 있으며, 따라서 도 5에 도시된 벡터들의 벡터 접근법(vector-of-vectors approach)을 지원하도록 설계된 피연산자에 적용될 때에도 올바른 결과를 제공할 수 있다. 따라서, 서브벡터 입도 명령어는 소정 유형의 처리 연산에 대해 가능하지 않을 수 있다. 그러나, 통상적으로 전체 벡터에 걸쳐 적용될 치환 또는 감소와 같은 몇몇 유형의 처리 연산에 대해, 서브벡터의 입도에서 대응하는 연산을 적용하는 서브벡터 지원 명령어를 정의하여, 상이한 처리 결과를 제공하는 것이 유용할 수 있다. 유사하게, (예를 들어, 개별 요소보다는 서브벡터의 입도에서 슬롯 비트를 설정하거나 관독함으로써) 서브벡터 입도에서 동작하는 소정 유형의 로드/저장 명령어 및 루프 제어 또는 슬롯 설정 명령어를 구현하는 것이 유용할 수 있다. 스케일링 가능한 벡터 ISA에 그러한 명령어를 포함시킴으로써, 이것은 스케일링 불가능한 벡터 ISA에 대해 이전에 최적화된 프로그램 코드를 재개발하여 스케일링 가능한 벡터 ISA를 사용하는 것을 훨씬 더 간단하게 만들며, 이는 그 코드가 스케일링 가능한 벡터 ISA에 의해 지원되는 더 큰 벡터 길이를 사용하는 고급 프로세서 구현에서 실행될 때 더 나은 성능을 달성할 수 있게 한다.

[0087] 도 6은 서브벡터 지원 명령어의 처리를 예시하는 흐름도를 도시한다. 단계 100에서, 스케일링 가능한 벡터 ISA를 지원하는 명령어 디코더(10)는 실행 중인 프로그램의 다음 명령어를 디코딩한다. 단계 102에서, 명령어 디코더(10)는 디코딩된 명령어가 서브벡터 지원 명령어인지 여부를 결정한다. 만약 아니라면, 단계 104에서 명령

어 디코더(10)는 다른 유형의 명령어에 의해 표시된 바와 같은 처리 연산을 수행하도록 처리 회로부(16)를 제어한다. 명령어 디코더(10)에 의해 디코딩된 명령어가 서브벡터 지원 명령어인 경우, 단계 106에서 명령어 디코더(10)는 소정 수의 벡터 요소를 각각 포함하는 2개 이상의 서브벡터를 포함하는 것으로서 취급되는 주어진 벡터에 대해 서브벡터 입도에서 연산을 수행하도록 처리 회로부(16)를 제어하며, 여기서 각각의 서브벡터는 동일한 서브벡터 길이를 갖는다. 서브벡터 길이는 컴파일 시간에 알려진다. 그러나 전체 벡터 길이는 컴파일 시간에 알려지지 않으며, 따라서 명령어에 의해 처리되는 서브벡터들의 수는 컴파일 시간에 알려지지 않는다.

[0088]

도 7 내지 도 11은 벡터 피연산자의 각각의 서브벡터 내에서 치환 연산을 적용하기 위한 서브벡터 지원 치환 명령어의 다양한 예를 도시한다. 각각의 경우에, 명령어에 대한 입력 피연산자(들) 및 결과 둘 모두는 아키텍처에서 고정되거나 달리 컴파일 시간에 알려지는 서브벡터 길이(예를 들어, 128비트)를 각각 갖는 다수의 서브벡터로 논리적으로 분할되는 것으로 간주되는 벡터이다. 벡터 피연산자 및 결과의 전체 길이는 스케일링 가능하며 컴파일 시간에 알려지지 않는다. 설명의 용이함을 위해, 아래에서 논의되는 모든 예는 32비트 요소를 갖는 예를 보여주며, 따라서 서브벡터당 4개의 벡터 요소가 있지만, 다른 예는 상이한 요소 크기를 사용할 수 있다는 것이 인식될 것이다. 각각의 치환 명령어에 대해, 결과의 주어진 서브벡터 내의 모든 요소는 하나 이상의 피연산자 벡터의 대응하는 서브벡터로부터 선택된 비트들의 치환으로 설정된다. 결과의 주어진 서브벡터와 비교하여 상이한 상대적 서브벡터 위치에 있는 피연산자 벡터의 다른 서브벡터 내의 요소로부터 선택된 비트에 따라 결과의 주어진 서브벡터 내의 주어진 요소를 설정하는 것은 가능하지 않다. 따라서, 대응하는 치환 연산은 서브벡터 입도에서 여러 번 수행된다.

[0089]

도 7은 복제 치환 명령어 DUPQ의 제1 예를 도시하며, 여기서 결과 Zd의 주어진 서브벡터에 적용되는 치환은 피연산자 벡터 Zn의 대응하는 서브벡터의 선택된 요소를 결과의 주어진 서브벡터의 벡터 요소들 각각에 복제하는 것이다. 어느 요소가 복제될지는 즉시 인덱스 값 #imm에 의해 표시되며, 이는 서브벡터의 시작에 대한 요소 위치를 정의한다. 예를 들어, 도 7에서 즉시 값은 1의 값을 가지며, 이는 피연산자 Zn의 각각의 서브벡터의 요소 1이 결과 Zd의 대응하는 서브벡터 내의 모든 벡터 요소 위치에 복제될 것임을 나타낸다. 따라서, 값 A1은 Zd의 처음 4개의 요소 위치에 복사되고, 값 B1은 다음 4개의 요소 위치에 복사되고, 값 C1은 다음 4개의 요소 위치에 복사되고, 등등이다(왜냐하면 각각의 서브벡터가 이 예에서 4개의 벡터 요소를 포함하기 때문에). 치환 연산을 사용하여 처리되는 서브벡터들의 수는 스케일링 가능한 벡터 ISA에 의해 허용되는 바와 같은 하드웨어에 의해 사용되는 특정 벡터 길이에 의존할 것이다. 그럼에도 불구하고, 복제될 주어진 벡터 요소의 인스턴스들의 수가 컴파일 시간에 알려지기 때문에(예를 들어, 이 경우에 4개), 부분 합 감소에 대해 위에 제시된 것과 같은 코드 최적화가 여전히 사용되어, 컴파일 시간에서의 (서브)벡터 길이에 대한 지식에 의존하는 코드 최적화를 여전히 허용하면서 스케일링 가능한 벡터 ISA를 채택하는 것을 더 쉽게 만들 수 있다.

[0090]

도 7의 하부 부분은 종래의 벡터 명령어에 대해 예상될 수 있는 바와 같은, 전체 벡터의 입도에서 적용되는 경우의 대응하는 복제 치환을 도시한다. 이 경우에, 즉시 인덱스 값은 벡터 피연산자 Zn의 시작에 대한 벡터 요소 위치를 정의할 것이고, 벡터 피연산자 Zn의 이 위치에 있는 벡터 요소는 결과 벡터 Zd의 모든 요소에 복사될 것이다. 이것은 전체 벡터 치환 명령어와 비교하여 치환 명령어의 서브벡터 지원 변형을 사용하여 달성되는 결과에 있어서의 차이를 명확하게 보여준다.

[0091]

도 8은 서브벡터 지원 치환 명령어 EXTQ의 다른 예를 도시한다. 이 예에서, 각각의 서브벡터에 적용되는 치환은 결과 Zdn'의 각각의 서브벡터의 하위 부분을 제1 벡터 피연산자 Zdn의 대응하는 서브벡터의 상위 비트로 설정하고, 각각의 서브벡터의 상위 부분을 제2 벡터 피연산자 Zm의 대응하는 서브벡터의 하위 비트로 설정하는 추출 치환이다. 이 예에서, 인코딩은 파괴적이며, 따라서 결과는 제1 벡터 피연산자를 제공하는 데 사용되는 것과 동일한 레지스터 Zdn에 기입된다. 다른 예는 결과를 저장하는 데 사용되는 목적지 레지스터와는 별개로, 제1 벡터 피연산자를 제공하기 위해 추가 벡터 레지스터를 정의하는 건설적 인코딩을 사용할 수 있다. 즉시 인덱스 값은 2개의 피연산자 벡터로부터 추출된 부분의 크기를 정의한다(예를 들어, 인덱스는 서브벡터의 시작에 대한 위치를 정의할 수 있으며, 여기서 상위 부분이 제1 피연산자에 대해 추출되어 결과의 대응하는 서브벡터의 하위 비트로 복사되고, 결과의 대응하는 서브벡터의 나머지 비트는 제2 피연산자의 최하위 단부로부터 선택된 비트로 채워질 수 있다).이 특정 예에서, 즉시 값은 바이트 입도(8비트 단위)에서 비트 위치를 선택하며, 따라서 주어진 결과 서브벡터의 비트 $[127-(imm*8):0]$ 은 제1 피연산자 Zdn 내의 대응하는 서브벡터의 비트 $[127:imm*8]$ 과 동일하게 설정되고, 주어진 결과 서브벡터의 비트 $[127:128-(imm*8)]$ 은 제2 피연산자 Zm의 비트 $[(imm*8)-1:0]$ 과 동일하게 설정된다. 다른 예는 8 비트 이외의 다른 단위의 증가들에서 선택하기 위한 즉시 값을 정의할 수 있다는 것이 인식될 것이다. 이러한 치환은 서브벡터 입도에서 개별적으로 적용되지만, 이때 각각의 서브벡터 치환은 두 피연산자의 대응하는 서브벡터로부터 추출된 부분들 사이의 경계를 나타내는 동일한

즉시 값을 사용한다. 다시, 도 8은 서브벡터 입도에서 이러한 연산을 적용하는 것이 전체 벡터 치환과는 어떻게 상이한지를 도시한다.

[0092] 도 9는 테이블 룩업 치환이 서브벡터별로 적용되는, 서브벡터 지원 치환 명령어 TBLQ의 다른 예를 도시한다. 이 예에서, 제1 벡터 피연산자 Z_m 은 제2 벡터 피연산자 Z_n 의 주어진 서브벡터의 어떤 요소가 결과 벡터 Z_d 의 대응하는 서브벡터 내의 대응하는 요소 위치에 복사될 것인지를 나타내는 인덱스 값들의 세트를 정의한다. 인덱스 값은 전체 벡터에 대해서보다는, 대응하는 서브벡터의 시작에 대해 정의된다. 따라서, 제2 서브벡터(제2 피연산자 Z_n 에서의 값 B0, B1, B2, B3에 대응함)에 대해, 요소 인덱스 3, 3, 1, 2는 결과에서의 대응하는 서브벡터의 대응하는 벡터 요소에 대해 값 B3, B3, B1, B2(B3은 서브벡터의 최하위 요소에 배치되고 B2는 최상위 요소에 배치됨)를 선택한다. 특정 인덱스 값이 주어진 서브벡터에서의 최대 벡터 요소 인덱스보다 큰 값을 갖는 경우, 0은 결과 벡터 Z_d 내의 대응하는 위치에 기입된다(예를 들어, 주어진 서브벡터 내의 4개의 요소(0 내지 3으로 라벨링됨)에 대한 최상위 인덱스 3보다 더 큰, 제3 서브벡터 내의 최상위 벡터 요소 위치에 대한 인덱스 값 6의 예를 참조, 따라서 0은 결과 Z_d 내의 대응하는 위치에 기입됨). 다시, 이러한 서브벡터별 치환의 접근법은 인덱스 값이 전체 벡터에 대해 정의될 것이고 도 9에 도시된 동일한 두 피연산자 Z_m , Z_n 에 전체 벡터 치환을 적용하는 것이, 도 9의 하부 부분에 도시된 바와 같이, 서브벡터 입도에서 수행된 연산과는 상이한 결과를 제공할 전체 벡터 치환과는 상이하다.

[0093] 도 10은 서브벡터 지원 치환 명령어 ZIPQ1의 다른 예를 도시하며, 여기서 이 경우에 주어진 서브벡터에 적용된 치환은, 결과 벡터 Z_d 의 주어진 서브벡터 내에서, 두 벡터 피연산자 Z_n , Z_m 의 대응하는 서브벡터의 하부 절반으로부터의 요소들을 인터리브하는 것이다. 이 치환은 서브벡터들 각각에 대해 수행되며, 다시 도 10의 하부 부분에 도시된 바와 같이 전체 벡터 입도에서 유사한 치환이 수행된 경우와는 상이한 결과를 제공한다. 도 10은 각각의 서브벡터의 하부 절반으로부터의 요소들을 인터리브하는 예를 도시하지만, 대응하는 명령어 ZIPQ2가 또한 대신에 각각의 서브벡터의 상부 절반으로부터의 요소들을 인터리브하도록 정의될 수 있다.

[0094] 도 11은 서브벡터 지원 치환 명령어 UZPQ1의 다른 예를 도시하며, 여기서 이 경우에 주어진 서브벡터에 적용되는 치환은 결과 벡터 Z_d 의 대응하는 서브벡터 내에서 두 벡터 피연산자 Z_n , Z_m 의 대응하는 서브벡터의 짝수 번호 벡터 요소들을 연결하는 것이다. 다시, 이것은 전체 벡터의 입도에서 유사한 치환이 적용된 경우 달성될 것과는 상이한 순서로 요소들을 갖는 결과를 생성한다. 이 명령어의 대안 버전 UZPQ2는 짝수 번호 요소들 대신에 홀수 번호 요소들을 연결할 수 있다.

[0095] 도 12는 서브벡터 입도에서 감소 연산들을 수행하는 인터-서브벡터 감소 명령어의 예를 도시하며, 각각의 감소 연산은 피연산자 벡터의 다수의 요소를 결과 벡터 내의 단일 데이터 값으로 감소시킨다. 이 예는 명령어가 피연산자 벡터 Z_n 과 연관된 술어 값 P_g 를 지정하여 어느 요소가 마스킹되는지를 나타내는 술어 명령어를 보여준다. 마스킹된 요소는 감소 결과에 기여하지 않는다. 이 예에 대해, 감소는 각각의 서브벡터 내의 대응하는 번호의 요소들에 걸쳐 수행되며, 이 예에서 적용된 감소 연산자는 덧셈이다. 따라서, 결과 벡터 Z_d 의 요소 0은 벡터 피연산자 Z_n 의 서브벡터들 각각 내의 위치 0에 있는 임의의 마스킹되지 않은 요소들의 합으로 설정되고, 결과 벡터 Z_d 의 요소 1은 벡터 피연산자 Z_n 의 서브벡터들 각각 내의 위치 1에 있는 임의의 마스킹되지 않은 요소들의 합으로 설정되고, 다른 요소 위치들에 대해서 등등이다(각각의 서브벡터 내의 주어진 위치에 있는 모든 요소가 마스킹되는 경우, 결과의 대응하는 요소는 0으로 설정됨). 감소 연산은 요소들의 총 수를 감소시키기 때문에, 감소 결과로 채워지지 않은 결과 벡터 Z_d 의 나머지 요소는 0으로 채워질 수 있다.

[0096] 도 12는 덧셈 감소를 갖는 예를 도시하지만, AND, 배타적 OR(EOR), 부동 소수점 덧셈(FADD), 부동 소수점 최대값(FMAX, 이는 부동 소수점 숫자들의 세트의 최대값을 결정함), 부동 소수점 최소값(FMIN, 이는 부동 소수점 숫자들의 세트의 최소값을 결정함), OR, 부호 있는 최대값(SMAX, 이는 부호 있는 정수들의 세트의 최대값을 결정함), 부호 있는 최소값(SMIN, 이는 부호 있는 정수들의 세트의 최소값을 결정함), 부호 없는 최대값(UMAX, 이는 부호 없는 정수들의 세트의 최대값을 결정함), 및 부호 없는 최소값(UMIN, 이는 부호 없는 정수들의 세트의 최소값을 결정함)과 같은 다른 연산에 대해 서브벡터 입도에서 유사한 감소 연산이 수행될 수 있다.

[0097] 도 13에 도시된 바와 같이, 각각의 서브벡터 내에서 감소를 수행하는 인트라-서브벡터 감소 명령어를 제공하여, 입력 피연산자 Z_m 의 주어진 서브벡터의 모든 요소를 결과 벡터 Z_d 의 단일 데이터 값으로 감소시키는 것이 또한 가능하다. 그 단일 요소는 결과 벡터 Z_d 의 대응하는 서브벡터를 채우기 위해 부호 또는 제로 확장될 수 있다. 도 13에 도시된 바와 같이, 주어진 서브벡터 내의 감소 연산은 또한 제2 벡터 피연산자 Z_d 의 대응하는 서브벡터 내의 주어진 요소 위치(예를 들어, 요소 위치 0)로부터 추출된 요소에 의존할 수 있다. 다시, 술어는 Z_m 피연산자의 활성 또는 비활성 요소를 정의하는 술어 값 P_g 에 기초하여 적용될 수 있다(Z_d 로부터 취해진 추가 요소는

항상 활성화로 간주될 수 있다 - 이 요소는 일련의 이전 감소들의 결과를 추적하는 누산기 값으로서 사용될 수 있고, 따라서 결과가 누산기 값을 제공하는 데 사용되는 것과 동일한 레지스터 Zd에 기입되는 도 13에 도시된 바와 같은 파괴적인 인코딩을 사용하는 것이 유용할 수 있다).

[0098] 감소가 도 12에 도시된 바와 같이 각각의 서브벡터의 대응하는 요소들에 걸쳐 구현되는지 또는 도 13에 도시된 바와 같이 단일 서브벡터 내의 요소들에 대해 구현되는지에 관계없이, 이 감소는 대응하는 연산이 전체 벡터 입도에서 수행된 경우(벡터의 모든 요소를 단일 데이터 값으로 감소시킴), 또는 루프에 포함된 요소별 벡터 추가 명령어(또는 다른 감소 연산을 위한 유사한 요소별 명령어) 또는 벡터들의 세트 내의 대응하는 위치에 요소를 추가하거나 달리 감소시키는 명령어들의 시퀀스를 사용하여 동일한 감소 연산이 구현된 경우 달성될 결과와는 상이하게, 서브벡터 입도에서 수행되는 연산이다. 서브벡터 기반 접근법은 고정 벡터 길이를 가정했을 수 있는 스케일링 불가능한 벡터화된 코드를, 서브벡터가 그러한 고정 벡터 길이에 대응하고 전체 벡터 길이가 하드웨어 설계자의 설계 선택에 따라 스케일링 가능한 스케일링 가능한 벡터화된 코드에 매핑하는 것을 단순화하는 데 유용할 수 있다.

[0099] 도 14는 벡터들의 벡터 접근법을 지원하는 데 도움이 될 수 있는 서브벡터 지원 로드/저장 명령어의 예를 도시한다. 도 14는 메모리로부터 적어도 하나의 벡터 레지스터로 로드된 또는 적어도 하나의 벡터 레지스터로부터 메모리로 저장된 데이터 블록이 메모리 어드레스 공간 내의 연속 데이터 블록에 대응하는 연속 로드/저장 명령어를 도시한다. 도 14는 단일 벡터 레지스터를 로드/저장하는 로드/저장 명령어를 도시하지만, 다수의 벡터 레지스터의 로드/저장을 지원하기 위해 다른 로드/저장 명령어가 제공될 수 있다. 다른 형태의 벡터 로드/저장 명령어와는 달리, 서브벡터 지원 벡터 로드/저장은 서브벡터 내의 개별 벡터 요소보다는 서브벡터 길이를 벡터 로드/저장 연산의 기본 단위로서 취급한다. 따라서, 벡터 레지스터의 어떤 부분이 로드되거나 저장될 것인지를 나타내는 데 사용되는 술어 값 Pg는 개별 요소의 입도에서보다는 서브벡터 입도에서 적용된다. 0의 술어 비트에 대응하는 서브벡터는, 로드 명령어에 대해, 목적지 레지스터 Zt의 대응하는 서브벡터가 메모리로부터 로드된 대응하는 서브벡터의 값으로 설정되지 않고, 저장 명령어에 대해, 소스 레지스터 Zt의 대응하는 서브벡터 내의 데이터가 메모리 내의 대응하는 어드레스 위치에 기입되지 않도록 마스크될 수 있다.

[0100] 유사하게, 도 15는 메모리 내의 연속되지 않은 어드레스 블록들로부터의 다수의 서브벡터를 벡터 레지스터 Zt에 로드하거나 그로부터 저장할 수 있는 서브벡터 지원 로드/저장 명령어의 수집-산란 형태를 도시한다. 이 예에서, 벡터 레지스터 Zm은 주어진 서브벡터에 대응하는 어드레스 블록을 결정하는 데 사용되는 인덱스 값을 제공한다. 각각의 인덱스는 서브벡터 길이에 대응하는 크기의 데이터의 블록의 배수를 나타낸다. 다시, 술어는 서브벡터 입도에서 적용된다.

[0101] 도 14 및 도 15에 도시된 어드레싱 모드는 단지 하나의 예일 뿐이며, 다른 예들은 벡터 레지스터에 로드되거나 벡터 레지스터로부터 저장될 서브벡터의 어드레스를 결정하기 위해 상이한 기술을 사용할 수 있다는 것이 인식될 것이다. 예를 들어, 다른 수집-산란 명령어에 대해, 각각의 서브벡터에 대한 로드/저장 연산에 대한 어드레스를 계산하는 데 사용되는 벡터 피연산자는 오프셋들의 벡터보다는 기본 어드레스들의 벡터로서 사용될 수 있다. 일반적으로, 서브벡터 지원 로드/저장 명령어는 서브벡터의 입도에서 연산하는 임의의 로드/저장 명령어일 수 있다.

[0102] 도 16은 서브벡터 입도에서 주어진 벡터에 대응하는 술어를 설정하는 데 사용될 수 있는 술어 설정 명령어의 예를 예시한다. 이 예에서, 술어 설정 명령어는 'Rn'+i와 'Rm'의 비교가 참인 경우 서브벡터 i에 대응하는 술어 비트 Pi를 "참"(1)로 설정하는 비교 명령어이며, 여기서 'Rn'은 제1 스칼라 레지스터 Rn에 저장된 값이고 'Rm'은 제2 스칼라 레지스터 Rm에 저장된 값이다. 명령어의 상이한 변형들이 LO(부호 없는 하위), LS(부호 없는 하위 또는 동일), LT(부호 있는 미만), LE(부호 있는 이하), HI(부호 없는 상위), HS(부호 없는 이상), GT(부호 있는 초과), 또는 GE(부호 있는 이상)와 같은 상이한 비교 조건들에 대응하여 제공될 수 있다. 이 명령어는 도 4의 스케일링 가능한 코드 예에 도시된 whilelt 명령어와 유사하지만, 술어를 벡터 요소 입도보다는 서브벡터 입도에서 설정한다. 이것은 스케일링 불가능한 벡터화된 예에서 전체 벡터의 효과를 마스크할 코드의 거동을 에뮬레이션하는 데 유용할 수 있다.

[0103] 도 17은 서브벡터 지원 술어 설정 명령어의 제2 예를 도시하며, 여기서 다시 술어 값은 서브벡터 입도에서 설정되지만, 이 예에서 명령어는 서브벡터 입도에서 적용될 소정의 사전 정의된 패턴을 지정한다(예를 들어, 적용될 패턴에 따라 설정/소거될 술어 비트는 각각의 서브벡터의 시작에 대응하는 그러한 술어 비트이다). 이것은 개별 벡터 요소의 입도에서 패턴을 적용할 수 있는 대응하는 술어 설정 명령어와는 상이할 수 있다. 예를 들어, 술어 패턴은 지정된 수의 예측 비트를 활성화로서 설정하거나, N개의 술어마다 하나씩을 활성화로서 설정하거나

(여기서 N은 2, 3, 4 등과 같은 값임), 하드웨어에 대한 구현된 벡터 길이를 고려할 때 술어 값에 맞는 가장 큰 2의 거듭제곱에 대응하는 다수의 술어 비트를 활성으로서 설정하는 것일 수 있다.

[0104] 도 18은 미리 결정된 술어 패턴을 식별하는 패턴 식별자를 지정하고, 서브벡터 입도에서 적용될 때 지정된 술어 패턴에 의해 표시된 활성 서브벡터들의 수에 대응하는 수만큼 스칼라 피연산자 Xdn을 증가시키도록 처리 회로부(16)를 제어하는 서브벡터 지원 증가 명령어의 예를 예시한다. 술어 패턴에 대해 표시된 활성 서브벡터들의 수만큼 피연산자를 대신에 감소시키기 위해 대응하는 명령어가 제공될 수 있다. 술어 패턴은 도 17의 명령어에 대해 정의된 패턴에 대응하는 방식으로 정의될 수 있다. 이 명령어는 추가 루프 반복을 계속하는 것이 여전히 필요한지 여부 또는 모든 요구되는 서브벡터가 처리되었기 때문에 루프를 종료하는 것이 가능한지 여부를 추적하는 데 사용되는 루프 카운트 변수들의 증가 또는 감소를 제어하는 데 유용할 수 있다.

[0105] 전술된 모든 명령어는 고정 길이 벡터 아키텍처를 위해 작성된 코드를 스케일링 가능한 벡터 아키텍처에 적응시키는 것을 더 쉽게 만드는 데 도움이 될 수 있다. 이러한 명령어들 모두가 주어진 구현에서 구현될 필요는 없다는 것이 인식될 것이다. 또한, 다른 연산에 대해 유사한 서브벡터 입도 명령어가 정의될 수 있다.

[0106] 요컨대, Neon™과 같은 스케일링 불가능하고 벡터 길이를 규정하는 아키텍처로부터 SVE와 같은 스케일링 가능하고 벡터 길이 불가지론적인 아키텍처로 전환하는 소프트웨어 개발자를 위한 더 간단한 전환을 가능하게 하기 위해, 위의 예들은 서브벡터(예를 들어, 쿼드 워드(128비트) 크기) 요소를 추가하고 각각의 서브벡터를 스케일링 가능한 아키텍처에서의 요소로서 취급한다. 그렇게 함에 있어서, 스케일링 가능한 아키텍처의 벡터 길이 불가지론 스타일을 사용하여 스케일링 불가능한 아키텍처의 각각의 고정 길이 벡터를 벡터화하기 위해 벡터 내 벡터 스타일(vector-in-vector style)이 형성된다. 이것은 명령어들의 대략적인 1 대 1 매핑으로 스케일링 불가능한 코드를 스케일링 가능한 코드에 매핑하는 것을 허용하며, 따라서 벡터화는 스케일링 가능한 아키텍처에서 허용되는 더 길고 유연한 벡터 길이를 이용하면서도 컴파일 시간에 알려진 벡터 길이의 가정에 의존하는 스케일링 불가능한 아키텍처에 대해 도입된 코드 최적화를 유지할 수 있다.

[0107] 도 19는 사용될 수 있는 시뮬레이터 구현예를 예시한다. 앞서 설명된 실시예들은 관심이 있는 기술들을 지원하는 특정 처리 하드웨어를 동작시키기 위한 장치 및 방법들과 관련하여 본 발명을 구현하지만, 본 명세서에서 설명되는 실시예들에 따라 컴퓨터 프로그램의 사용을 통해 구현되는 명령어 실행 환경을 제공하는 것이 또한 가능하다. 그러한 컴퓨터 프로그램들은, 그들이 하드웨어 아키텍처의 소프트웨어 기반 구현예를 제공하는 한, 종종 시뮬레이터들로 지칭된다. 다양한 시뮬레이터 컴퓨터 프로그램들은 에뮬레이터들, 가상 머신들, 모델들, 및 동적 이진 변환기(dynamic binary translator)들을 포함한 이진 변환기들을 포함한다. 전형적으로, 시뮬레이터 구현예는 호스트 프로세서(330) 상에서 실행되어, 호스트 운영 체제(320)를 선택적으로 실행하여, 시뮬레이터 프로그램(310)을 지원할 수 있다. 몇몇 배열들에서, 하드웨어와 제공된 명령어 실행 환경 사이에 다수의 시뮬레이션 계층들이 있을 수 있고/있거나, 동일한 호스트 프로세서 상에 제공된 다수의 별개의 명령어 실행 환경들이 있을 수 있다. 역사적으로, 강력한 프로세서들이 합리적인 속도로 실행되는 시뮬레이터 구현예들을 제공하기 위해 요구되었지만, 그러한 접근법은 호환성 또는 재사용 이유들을 위해 다른 프로세서에 고유한 코드를 실행하려는 요구가 있을 때와 같은 소정 상황들에서 정당화될 수 있다. 예를 들어, 시뮬레이터 구현예는 호스트 프로세서 하드웨어에 의해 지원되지 않는 추가적인 기능을 명령어 실행 환경에 제공할 수 있거나, 또는 상이한 하드웨어 아키텍처와 전형적으로 연관된 명령어 실행 환경을 제공할 수 있다. 시뮬레이션의 개요가 문헌["Some Efficient Architecture Simulation Techniques", Robert Bedichek, Winter 1990 USENIX Conference, Pages 53 - 63]에서 주어진다.

[0108] 실시예들이 특정 하드웨어 구성들 또는 특징부들을 참조하여 앞서 설명된 범위 내에서, 시뮬레이션된 실시예에서, 동등한 기능이 적합한 소프트웨어 구성들 또는 특징부들에 의해 제공될 수 있다. 예를 들어, 특정 회로부는 시뮬레이션된 실시예에서 컴퓨터 프로그램 로직으로서 구현될 수 있다. 유사하게, 레지스터 또는 캐시와 같은 메모리 하드웨어는 시뮬레이션된 실시예에서 호스트 프로세서(330)의 호스트 저장소(예를 들어, 메모리 또는 레지스터)에 저장된 소프트웨어 데이터 구조로서 구현될 수 있다. 전술된 실시예들에서 언급된 하드웨어 요소들 중 하나 이상의 하드웨어 요소들이 호스트 하드웨어(예를 들어, 호스트 프로세서(330)) 상에 존재하는 배열들에서, 몇몇 시뮬레이션된 실시예들은, 적합한 경우, 호스트 하드웨어를 사용할 수 있다.

[0109] 시뮬레이터 프로그램(310)은 컴퓨터 판독가능 저장 매체(이는 비일시적 매체일 수 있음)에 저장될 수 있고, 타겟 코드(300)(이는 애플리케이션들, 운영 체제들 및 하이퍼바이저를 포함할 수 있음)에 프로그램 인터페이스(명령어 실행 환경)를 제공하는데, 이는 시뮬레이터 프로그램(310)에 의해 모델링되는 하드웨어 아키텍처의 인터페이스와 동일하다. 따라서, 타겟 코드(300)의 프로그램 명령어는 시뮬레이터 프로그램(310)을 사용하여 명령어

실행 환경 내로부터 실행될 수 있으며, 따라서 실제로 위에서 논의된 장치(2)의 하드웨어 특징들(예를 들어, 위에서 논의된 바와 같이 서버백터 지원 명령어를 지원하는 명령어 디코더(10) 및 처리 회로부(16))을 갖지 않는 호스트 컴퓨터(330)가 이러한 특징들을 에뮬레이션할 수 있다.

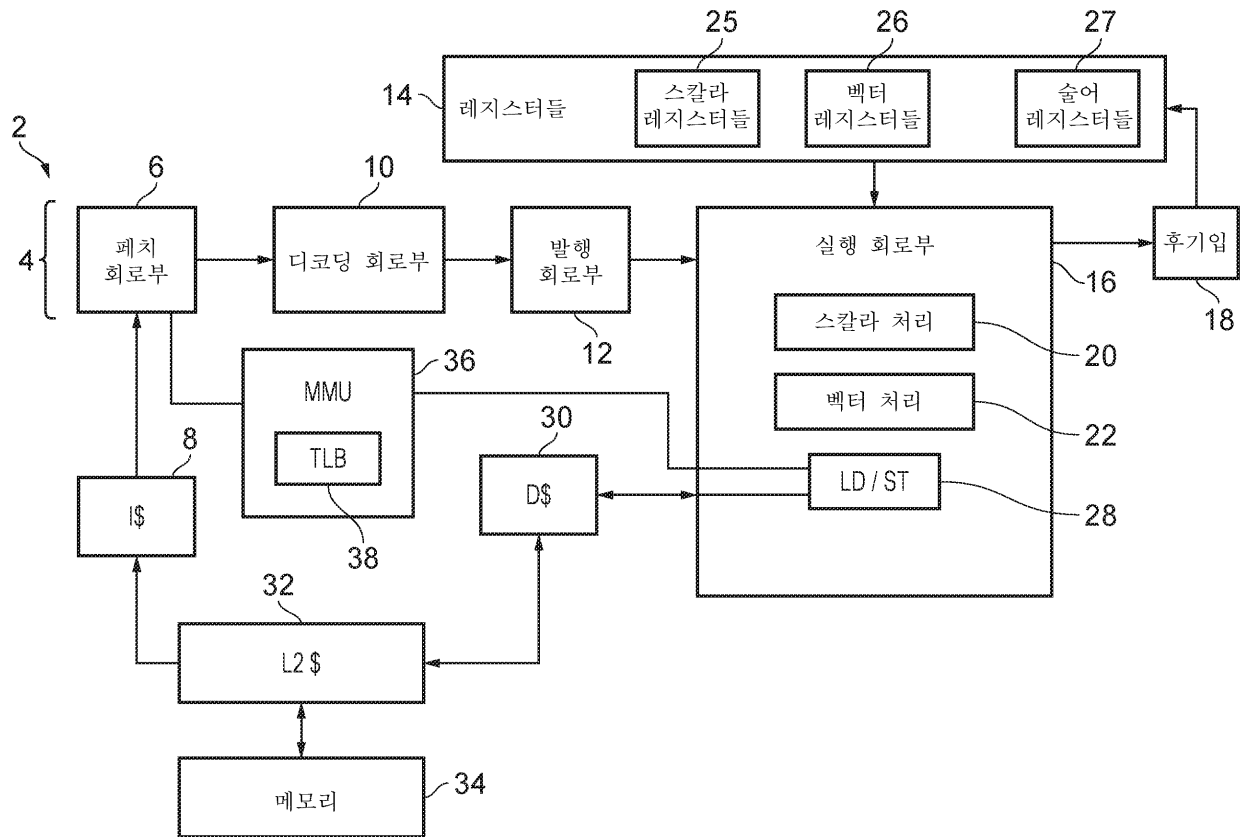
[0110] 따라서, 시뮬레이터 프로그램(310)은 타겟 코드(300)의 명령어들을 디코딩하고 이들을 호스트 장치(330)의 고유 명령어 세트 내의 대응하는 명령어 세트들에 매핑하기 위한 명령어 디코딩 프로그램 로직(312)을 가질 수 있다. 명령어 디코딩 프로그램 로직(312)은 전술된 서버백터 지원 명령어를 디코딩하기 위한 서버백터 지원 명령어 디코딩 프로그램 로직(313)을 포함한다. 레지스터 에뮬레이팅 프로그램 로직(314)은, 예컨대 호스트 장치(330)의 레지스터들 또는 메모리 내의 데이터에 액세스하는 것에 의해, 타겟 코드에 의해 요청된 레지스터 액세스를 호스트 장치(330)의 호스트 하드웨어 상에 유지된 대응하는 데이터 구조에 대한 액세스에 매핑한다. 메모리 관리 프로그램 로직(316)은 위의 하드웨어 구현 실시예에서 설명된 바와 같은 MMU(36)에 대응하는 방식으로, 타겟 코드(300)에 의해 시뮬레이션된 어드레스 공간에 대한 액세스를 시뮬레이션하기 위해 어드레스 변환, 페이지 테이블 워크(page table walk) 및 액세스 허가 체크를 구현한다. 메모리 어드레스 공간 시뮬레이션 프로그램 로직(318)은 타겟 프로그램 코드(300)의 소프트웨어에 의해 유지되는 페이지 테이블 정보를 사용하여 어드레스 변환에 기초하여 메모리 관리 프로그램 로직(316)에 의해 획득된 시뮬레이션된 물리 어드레스를, 호스트 프로세서(330)의 호스트 메모리에 액세스하는 데 사용되는 호스트 가상 어드레스에 매핑하기 위해 제공된다. 이러한 호스트 가상 어드레스들은 그들 자체가 호스트에 의해 지원되는 표준 어드레스 변환 메커니즘을 사용하여 호스트 물리적 어드레스로 변환될 수 있다(호스트 물리적 어드레스로의 호스트 가상 어드레스의 변환은 시뮬레이터 프로그램(310)에 의해 제어되는 것의 범위 밖에 있음).

[0111] 본 출원에서, 단어들 "... 하도록 구성된"은 장치의 요소가 정의된 동작을 수행할 수 있는 구성을 갖는다는 것을 의미하는 데 사용된다. 이러한 맥락에서, "구성"은 하드웨어 또는 소프트웨어의 상호접속의 배열 또는 방식을 의미한다. 예를 들어, 장치는 정의된 동작을 제공하는 전용 하드웨어를 가질 수 있거나, 프로세서 또는 다른 처리 디바이스가 기능을 수행하도록 프로그래밍될 수 있다. "하도록 구성된"은, 장치 요소가, 정의된 동작을 제공하기 위해 어떤 방식으로든 변경될 필요가 있음을 암시하지는 않는다.

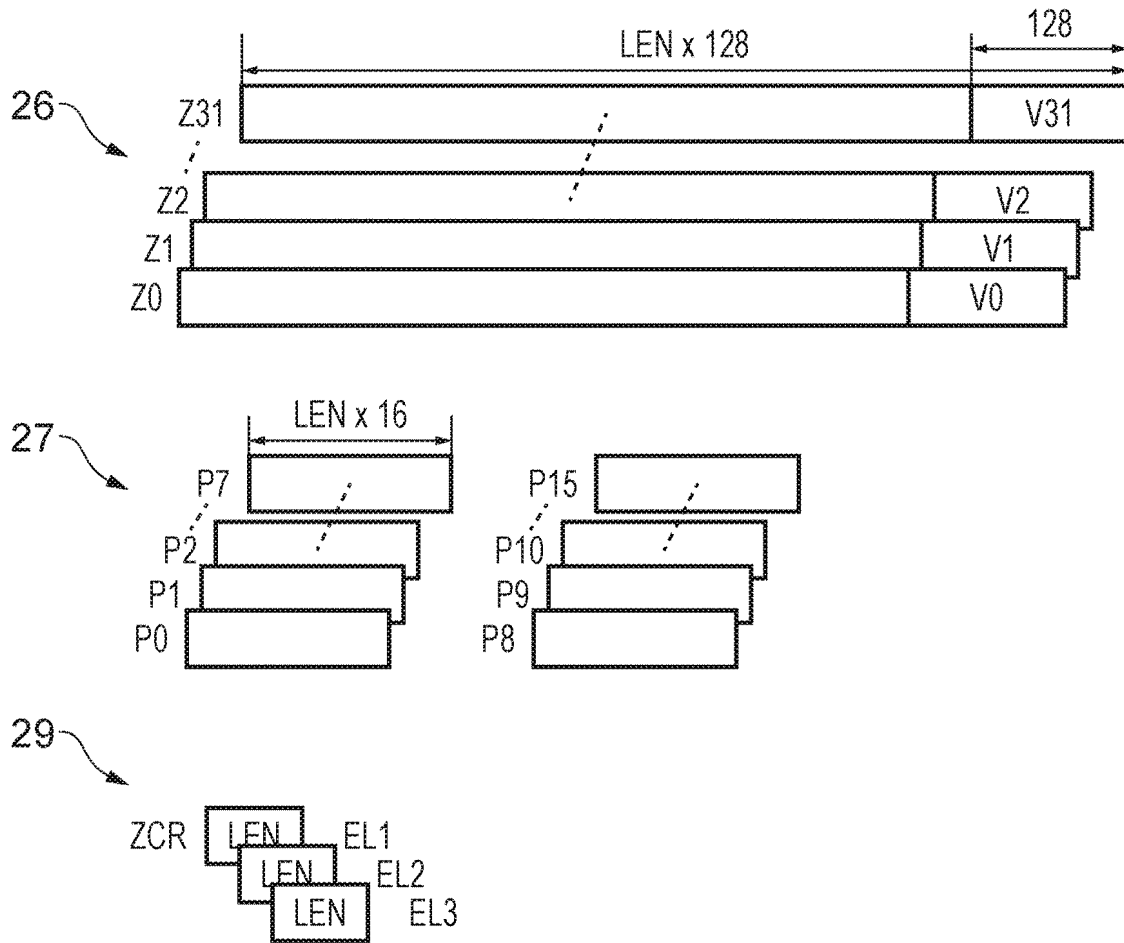
[0112] 본 발명의 예시적인 실시예들이 첨부 도면들을 참조하여 본 명세서에서 상세히 설명되었지만, 본 발명은 그러한 정확한 실시예들로 제한되지 않으며, 첨부된 청구항들에 의해 한정된 바와 같은 본 발명의 범위로부터 벗어남이 없이 실시예들에서 다양한 변경들 및 수정들이 당업자에 의해 이루어질 수 있다는 것이 이해되어야 한다.

도면

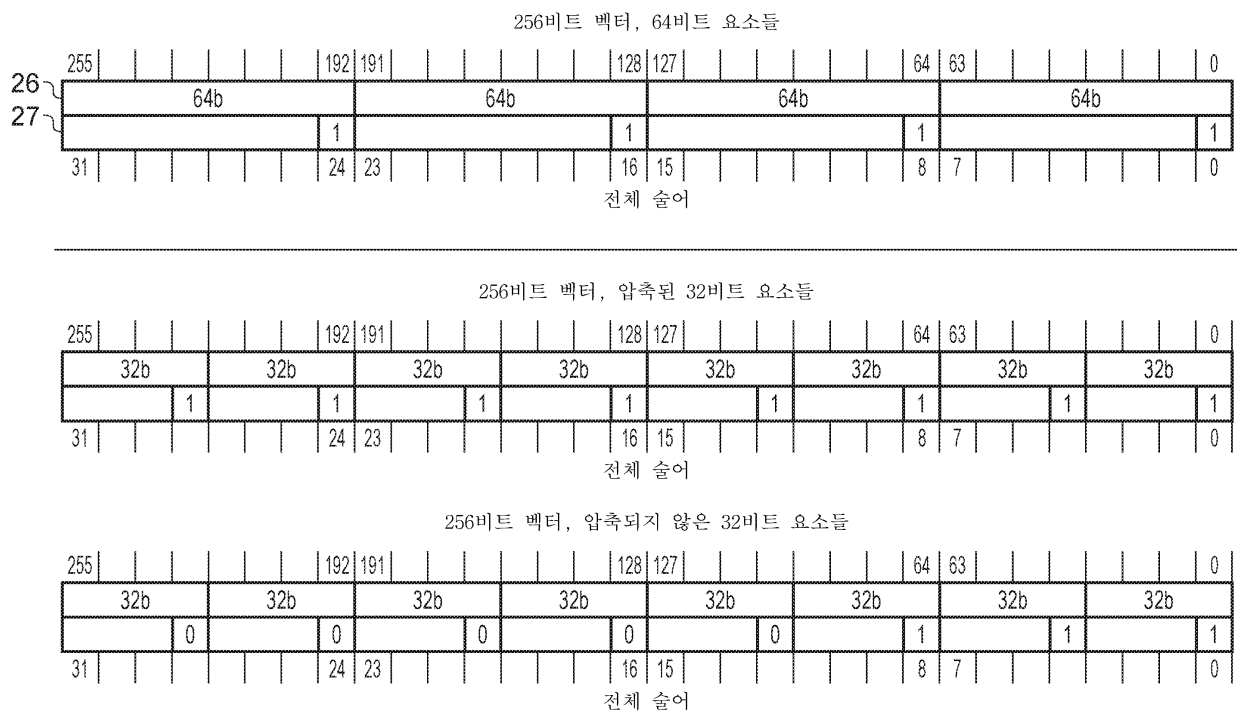
도면1



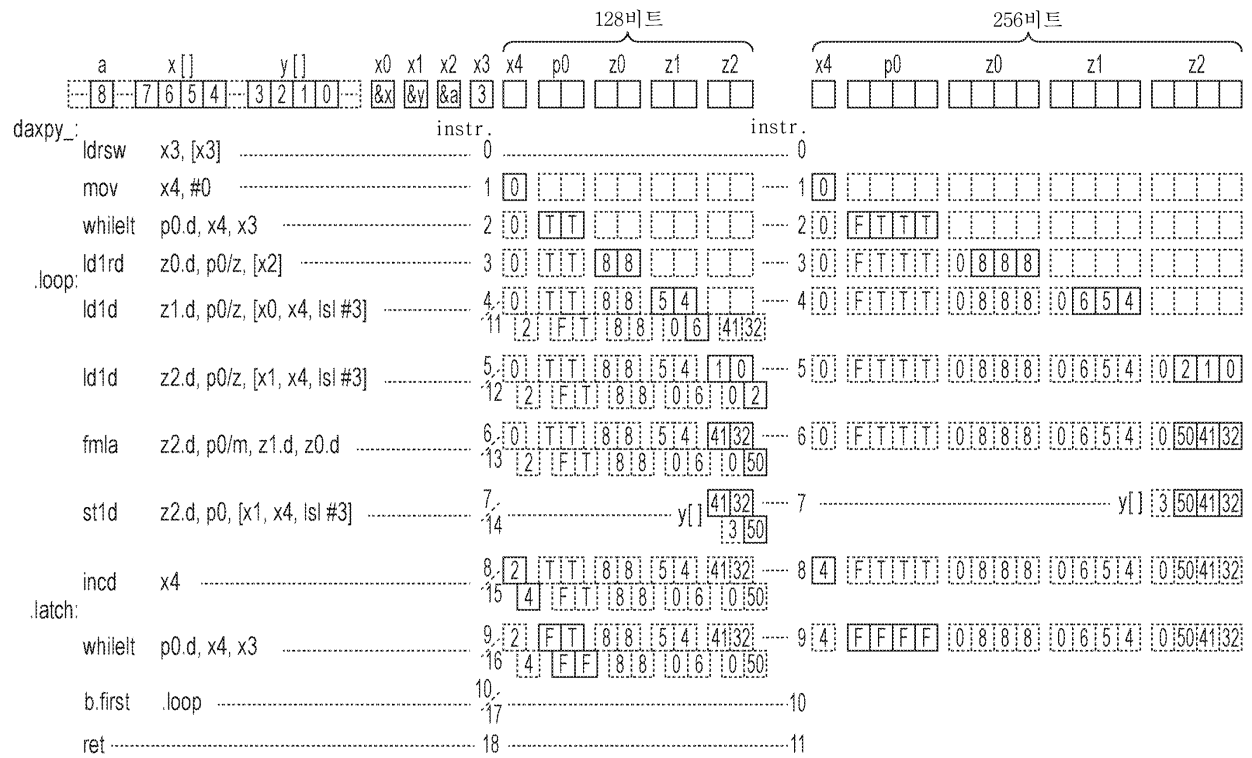
도면2



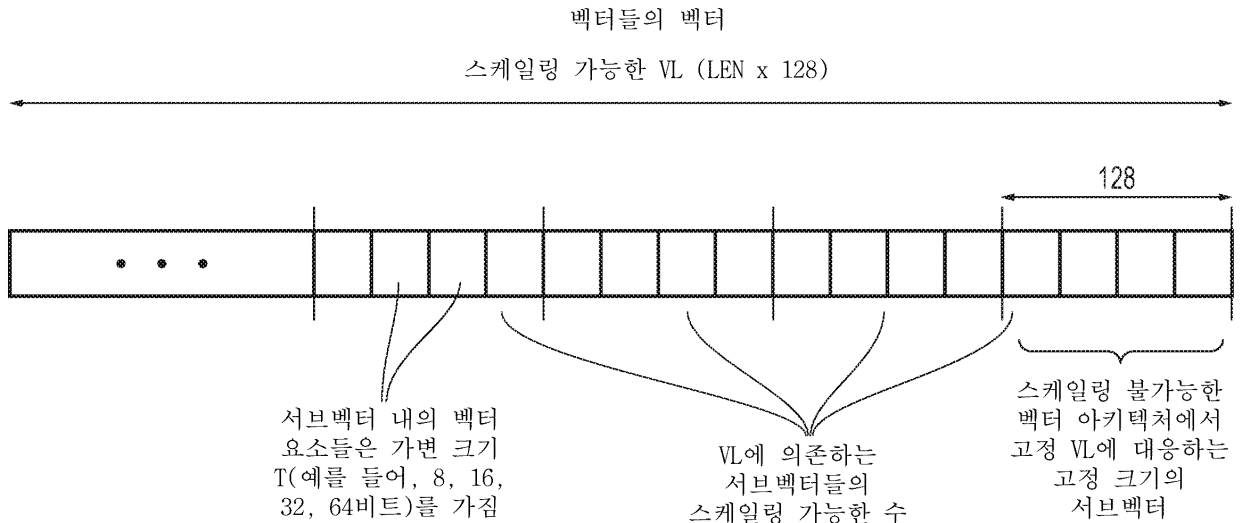
도면3



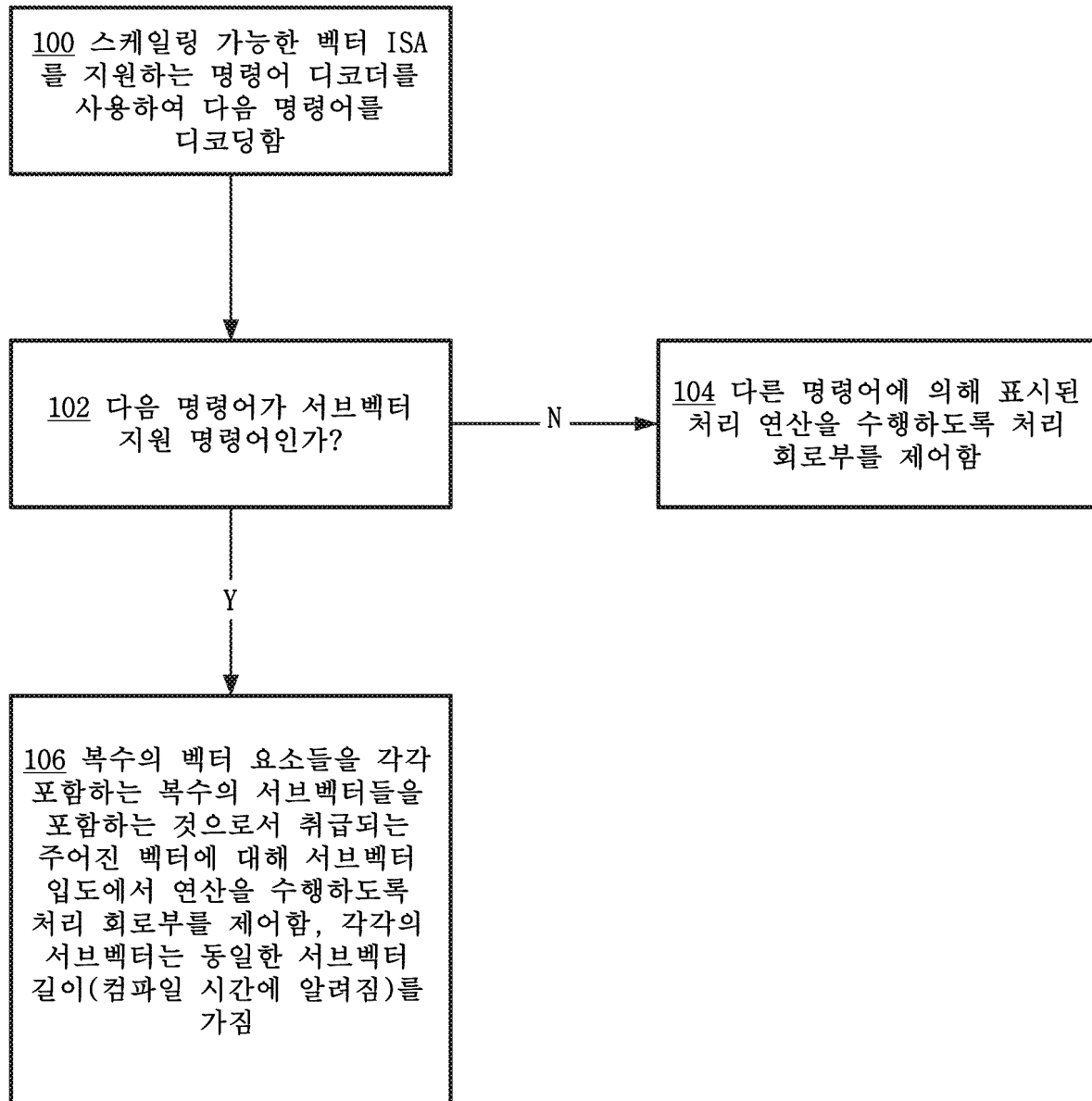
도면4



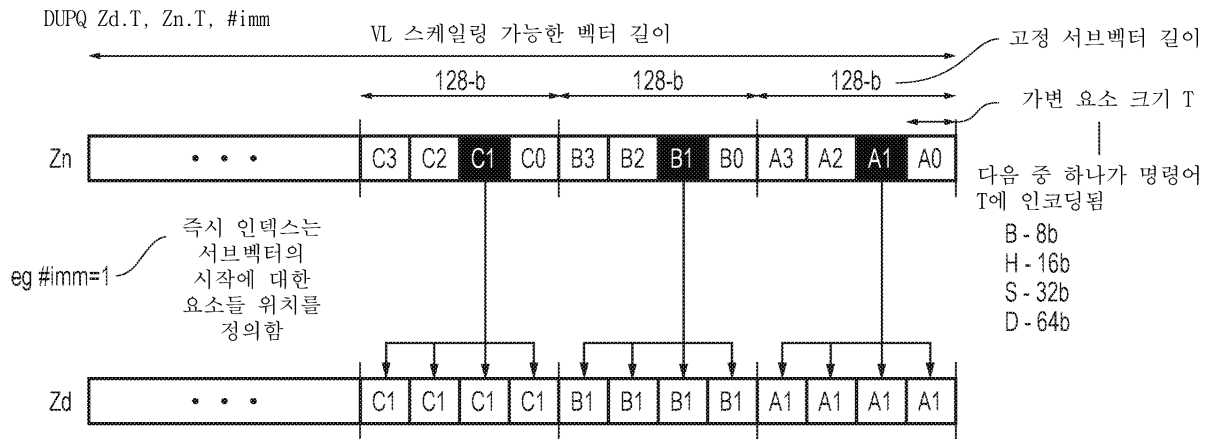
도면5



도면6



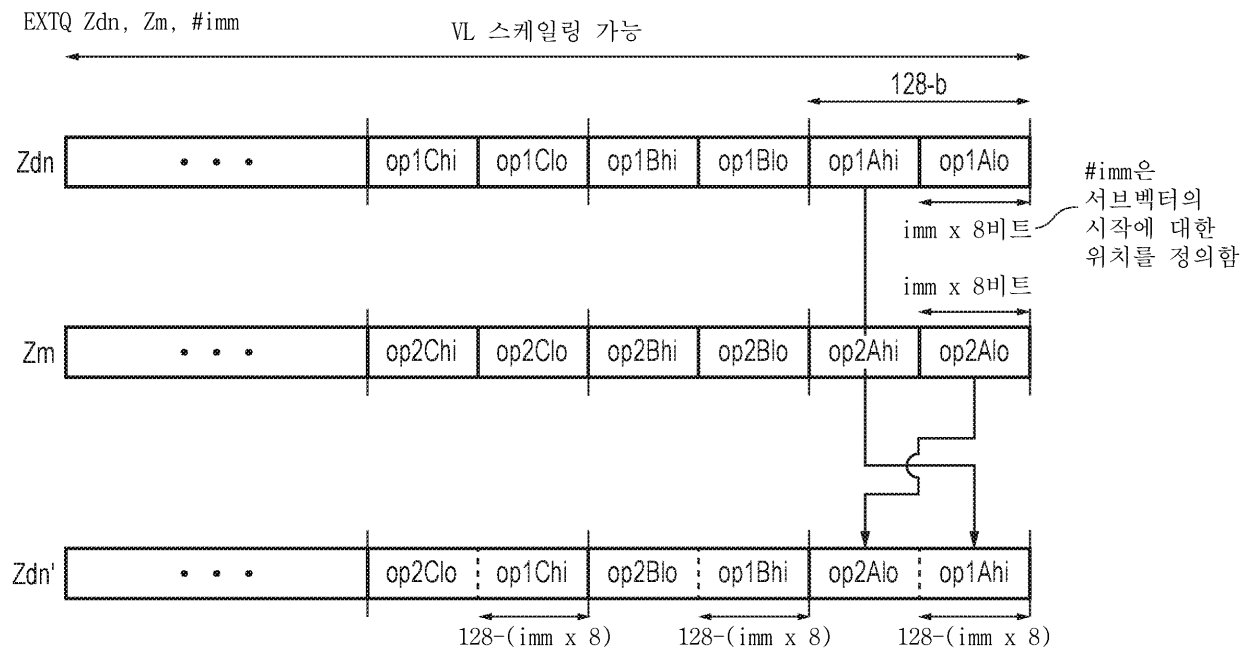
도면7



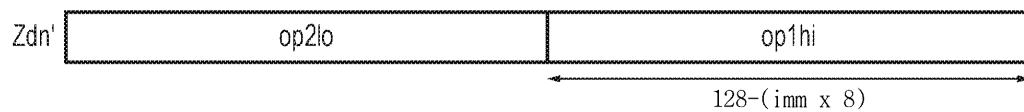
cf 전체 벡터 입도에서 수행되는 DUP는 다음의 결과를 제공할 것임:



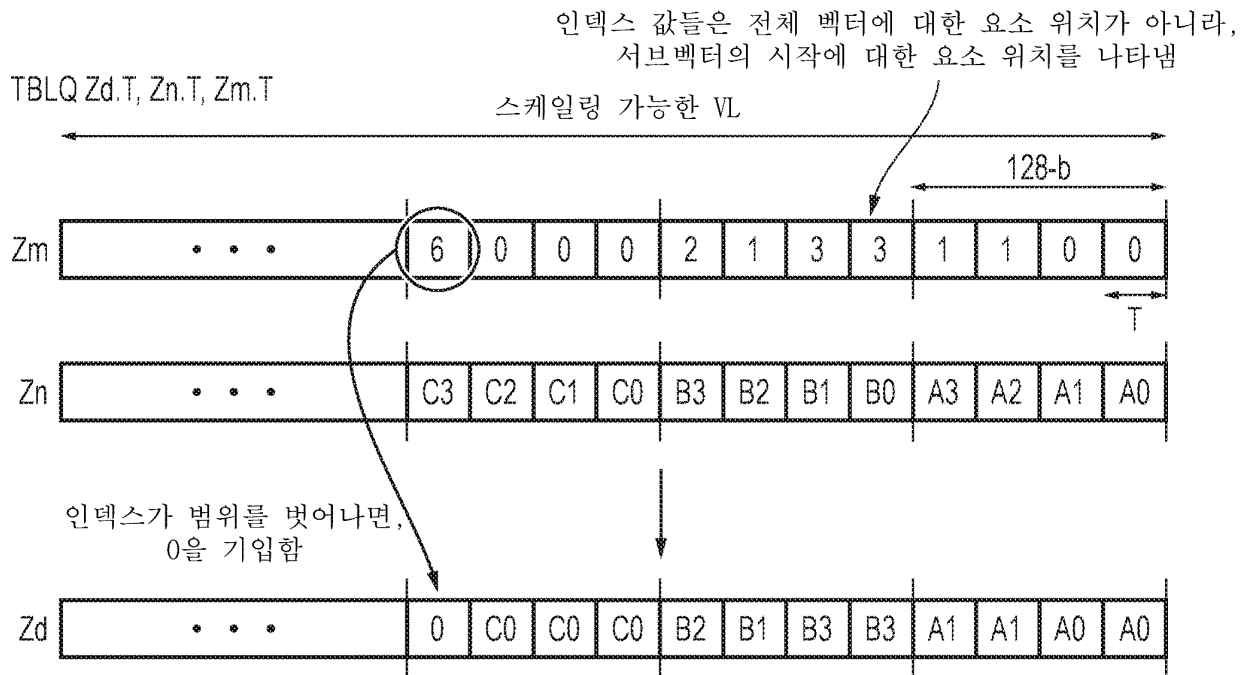
도면8



cf 전체 벡터 입도에서 수행되는 EXT는 다음의 결과를 제공할 것임:

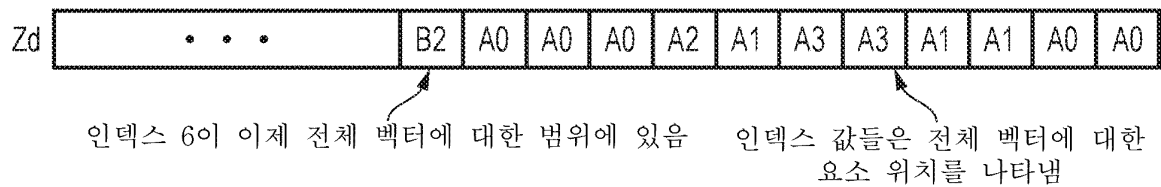


도면9



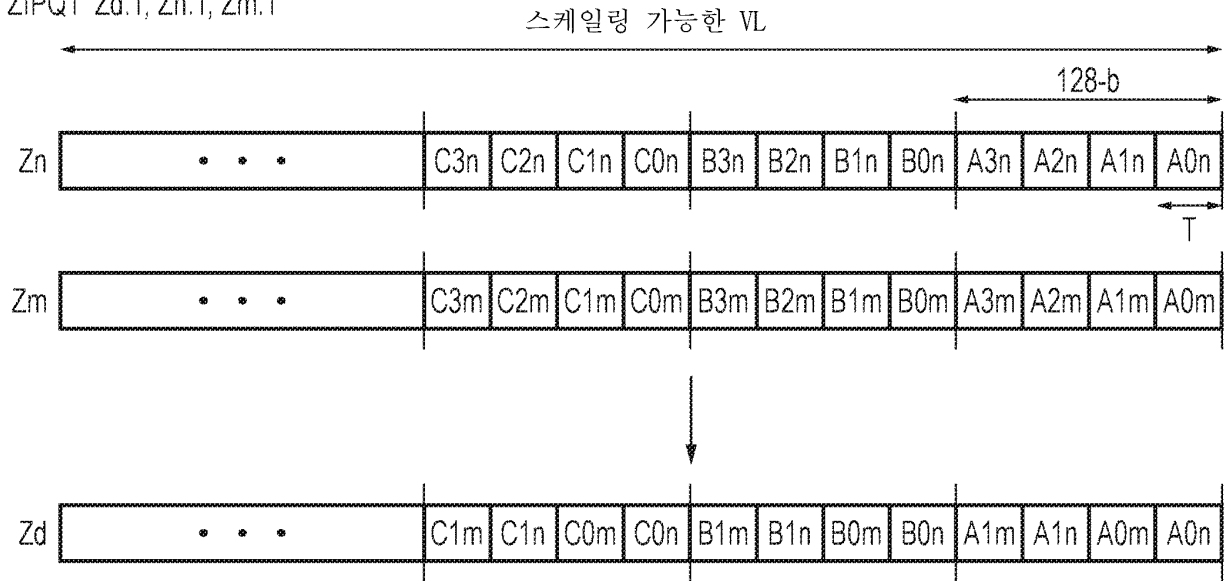
(대안: TBXQ - 인덱스가 범위를 벗어나면 목적지 요소를 변경하지 않은 채로 둠)

cf TLB는 동일한 Zn, Zm 입력들에 대해 전체 벡터 입도에서 수행됨:



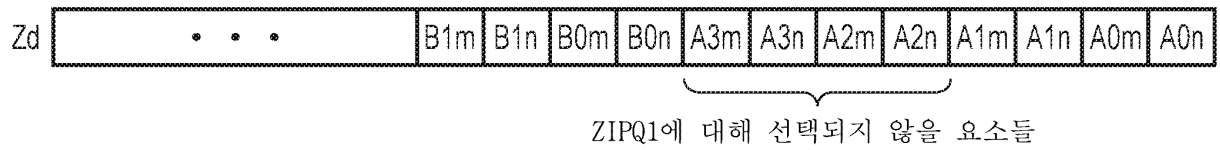
도면10

ZIPQ1 Zd.T, Zn.T, Zm.T



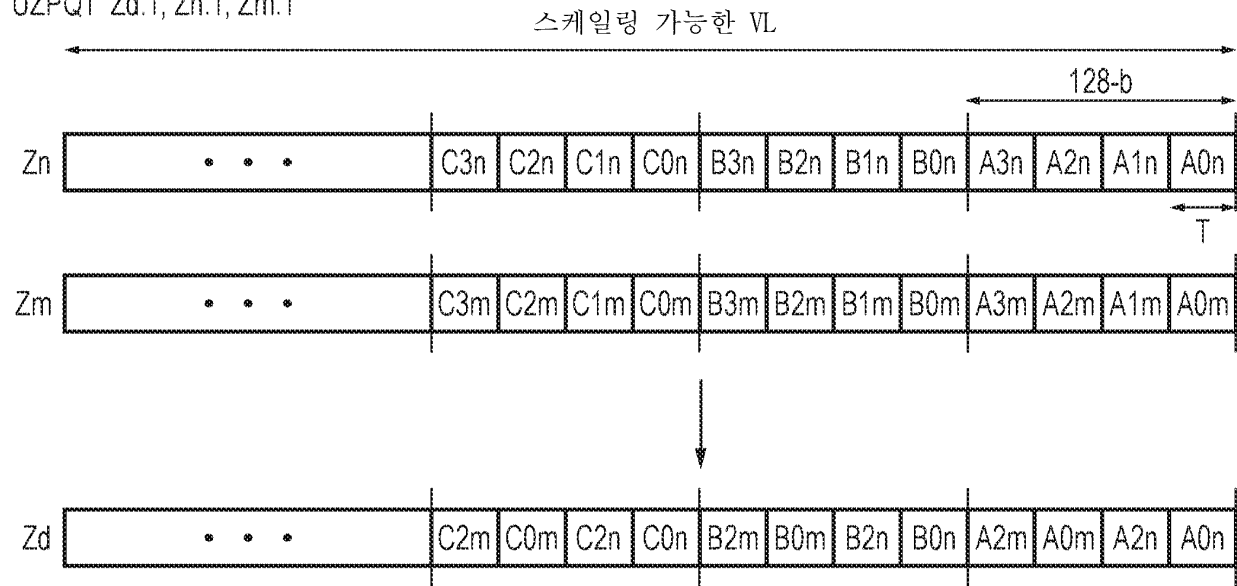
(대안: ZIPQ2는 각각의 서브벡터의 상부 절반들로부터의 요소들을 인터리브함)

ZIP1 명령어는 전체 벡터 입도에서 동일한 입력들에 대해 동작함



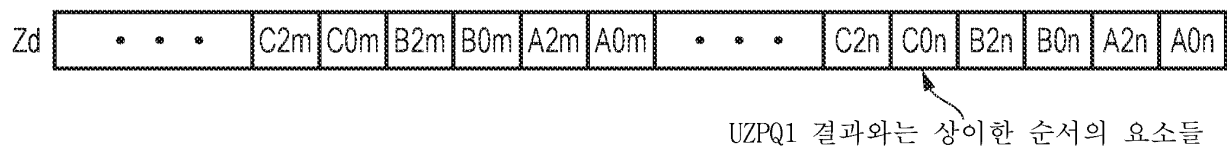
도면11

UZPQ1 Zd.T, Zn.T, Zm.T



(대안: UZPQ2는 홀수 요소들을 연결함)

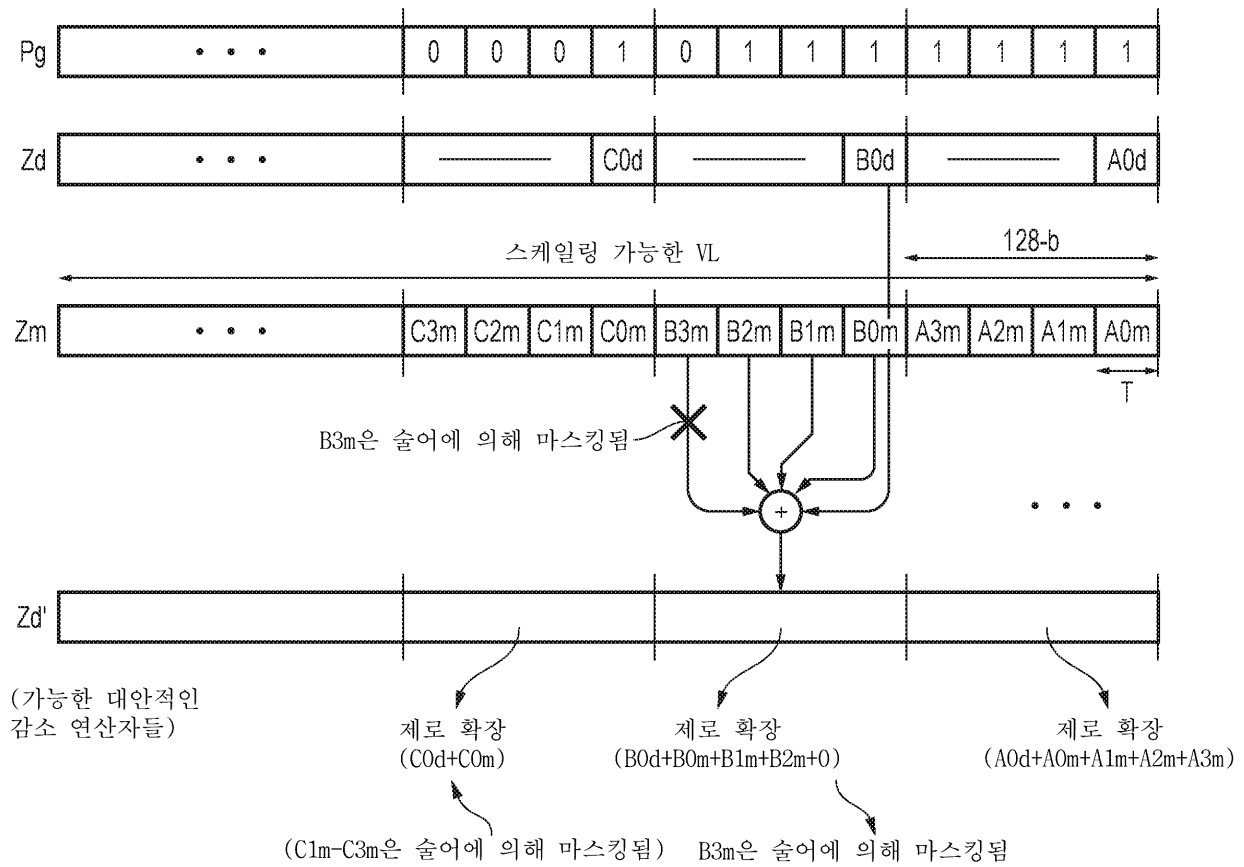
cf UZP1은 전체 벡터 입도에서 동일한 입력들에 대해 수행됨



도면 13

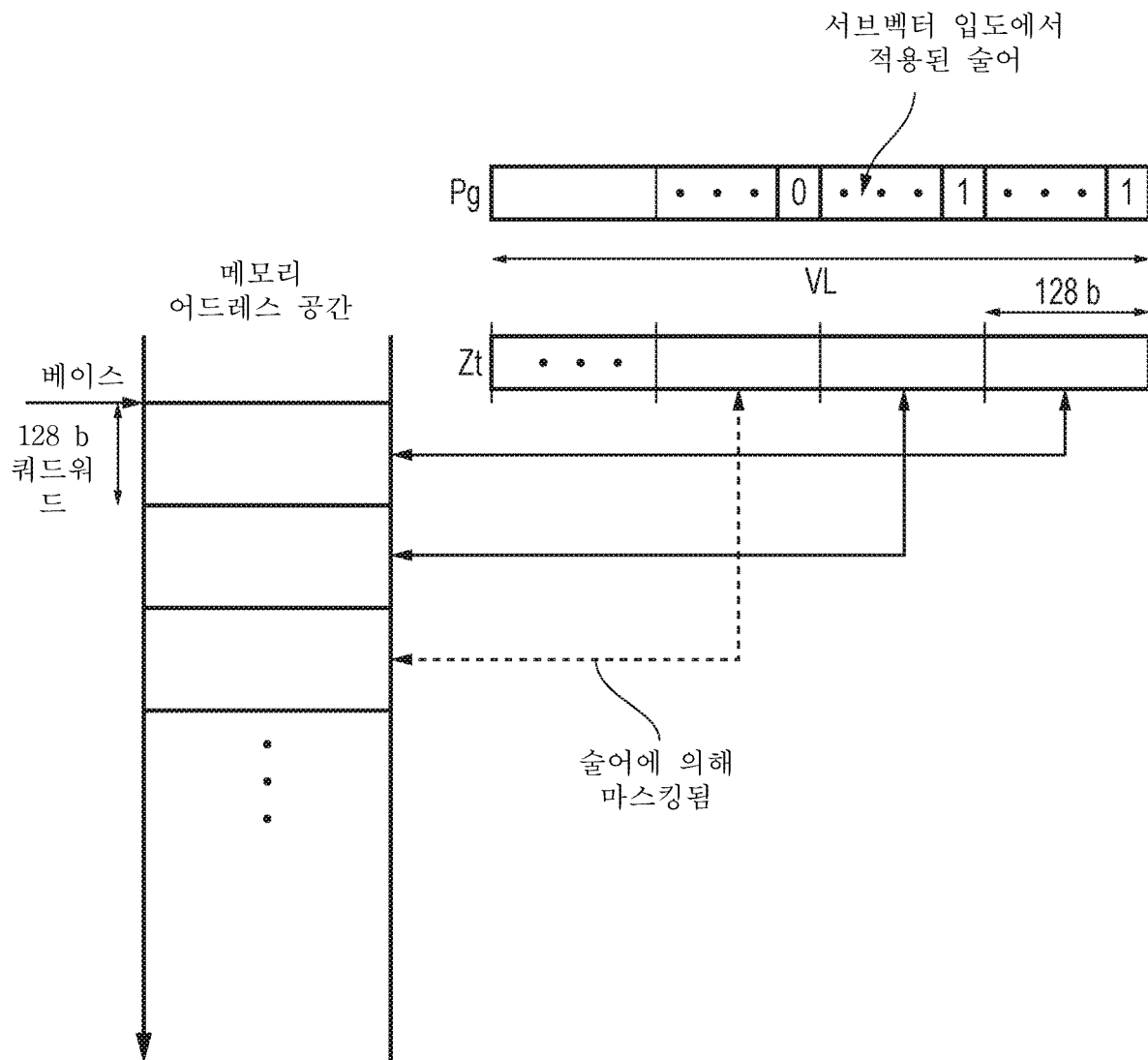
FADDQA Zd.T, Pg, Zm.T

(각각의 서브벡터 내에서의 감소)



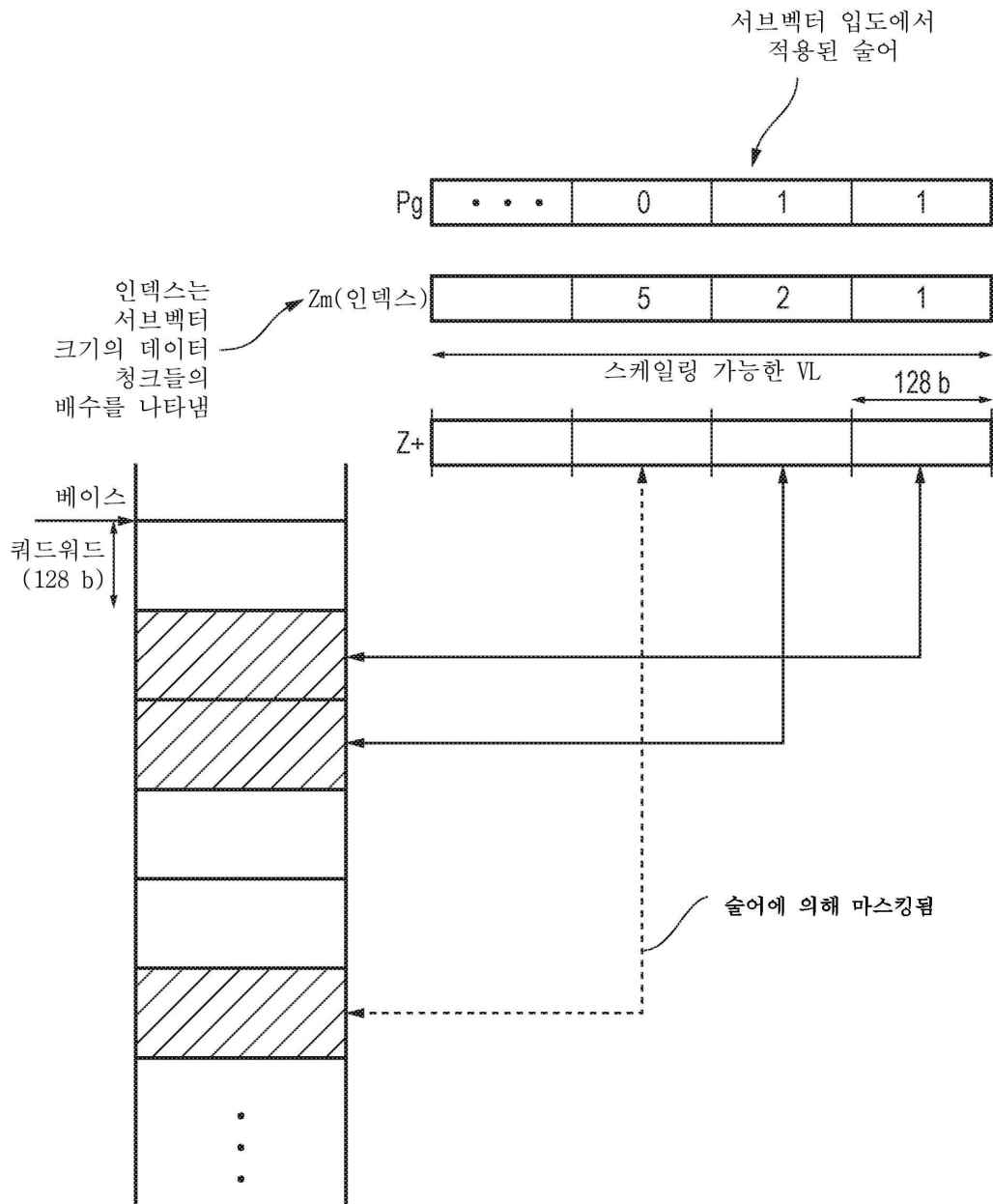
도면14

LD1Q / ST1Q (연속)



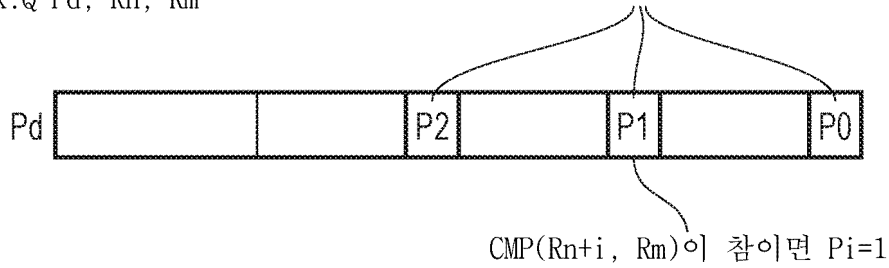
도면15

LD1Q / ST1Q (수집-산란)



도면16

WHILE xx.Q Pd, Rn, Rm 쿼드워드 (128b) 서브벡터들에 대응하는 술어 비트들을 설정함

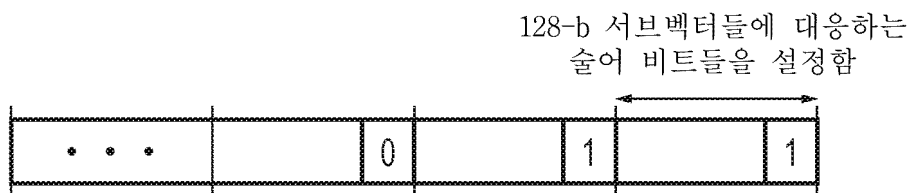


xx는 예를 들어 다음의 비교 조건을 나타냄:

-LO	-HI
-LS	-HS
-LT	-GT
-LE	-GE

도면17

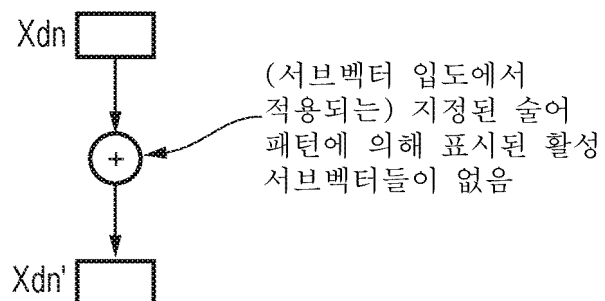
PTRUES Pd.Q, 패턴



패턴은 서브벡터 입도에서 적용될 술어 패턴을 정의함

도면18

INCQ Xdn, 패턴



(대안: DECQ는 피연산자를 감소시킴)

도면19

