(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2024년03월22일
(11) 등록번호 10-2649933
(24) 등록일자 2024년03월18일

(51) 국제특허분류(Int. Cl.)
*G06F 9/30* (2018.01)
(52) CPC특허분류
*G06F 9/30036* (2023.08)
*G06F 9/30094* (2013.01)
(21) 출원번호 10-2020-7018974
(22) 출원일자(국제) 2018년11월15일
심사청구일자 2021년10월26일
(85) 번역문제출일자 2020년07월01일
(65) 공개번호 10-2020-0094771
(43) 공개일자 2020년08월07일
(86) 국제출원번호 PCT/EP2018/081444
(87) 국제공개번호 WO 2019/115142
국제공개일자 2019년06월20일
(30) 우선권주장
17386048.7 2017년12월13일
유럽특허청(EPO)(EP)
(56) 선행기술조사문헌
US06295597 B1*
US20060259737 A1*
US20150227367 A1*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
**에이알엠 리미티드**
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110
(72) 발명자
**율 엠보우**
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드
**스테펜스 니겔 존**
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드
*(뒷면에 계속)*
(74) 대리인
**이화익**

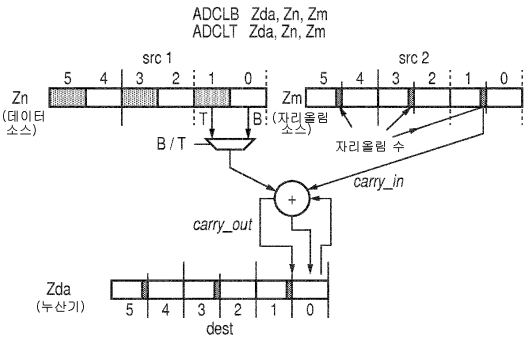
전체 청구항 수 : 총 15 항 심사관 : 지정훈

(54) 발명의 명칭 **벡터 자리올림이 있는 가산 명령**

## (57) 요 약

목적지 벡터 레지스터의 일부 성분들 또는 프레디케이트 레지스터의 대응하는 필드들을 사용하여 자리올림이 있는 가산 연산의 결과에 대응하는 자리올림 정보를 제공하는 벡터 자리올림이 있는 가산 명령이 기술된다. 이것은 긴 정수 값들의 승산을 포함하는 계산을 가속화하는데 유용하다.

*대 표 도* - 도3



(72) 발명자

**버지스 네일**

영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드

**맥클리스 그리고리오스**

영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드

# 명 세 서

## 청구범위

### 청구항 1

데이터 처리를 행하는 처리회로와,

명령을 디코드하고 상기 처리회로를 제어하여 데이터 처리를 행하게 하는 명령 디코딩 회로와,

복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들을 구비하고,

상기 명령 디코딩 회로는, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터 및 자리올림 소스 벡터 레지스터를 지정하는 - 이때 적어도 상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 각각은 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정한다 - 벡터 자리올림이 있는 가산(vector add-with-carry) 명령에 응답하여, 상기 처리회로를 제어함으로써, 상기 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해,

상기 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분으로부터 얻어진 자리올림 값(carry value)의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분을 갱신하고,

상기 가산의 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분을 갱신하고,

상기 가산의 결과에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분 및 상기 가산의 자리올림 출력에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분은 동일한 벡터 레지스터의 데이터 성분들인, 장치.

### 청구항 2

제 1항에 있어서,

상기 제1 데이터 소스 벡터 레지스터는 상기 목적지 벡터 레지스터와 동일한 레지스터이고, 상기 제1 소스 데이터 값은 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분의 이전 값을 포함하는 장치.

### 청구항 3

제 1항 또는 제 2항에 있어서,

각 쌍의 데이터 성분들은 한 쌍의 인접한 데이터 성분들을 포함하는 장치.

### 청구항 4

제 1항 또는 제 2항에 있어서,

상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 내부의 데이터 성분들의 쌍들의 수가 적어도 2일 때, 이들 쌍의 상기 제1 데이터 성분들이 이들 쌍의 상기 제2 데이터 성분들과 인터리브되는 장치.

**청구항 5**

제 1항 또는 제 2항에 있어서,

상기 벡터 자리올림이 있는 가산 명령에 응답하여, 상기 명령 디코딩 회로는, 상기 처리회로를 제어하여,

상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 상기 제2 데이터 성분의 최하위 비트로부터 상기 자리올림 값을 취득하고,

상기 가산의 상기 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분의 최하위 비트를 갱신하도록 구성된 장치.

**청구항 6**

제 5항에 있어서,

상기 벡터 자리올림이 있는 가산 명령에 응답하여, 상기 자리올림 소스 벡터 레지스터 및 상기 목적지 벡터 레지스터에 모두에 대해, 상기 최하위 비트 이외의 상기 제2 데이터 성분의 나머지 비트들이 사용되지 않는 장치.

**청구항 7**

제 1항 또는 제 2항에 있어서,

각각의 데이터 성분은 $2^N$ 비트를 포함하고, 상기 제2 소스 데이터 값은 $2^N$ 비트를 포함하고, 이때 N이 정수인 장치.

**청구항 8**

제 1항 또는 제 2항에 있어서,

상기 제2 데이터 소스 벡터 레지스터도 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정하고,

제1 변종의 상기 자리올림이 있는 가산 명령에 응답하여, 상기 제2 소스 데이터 값은 상기 제2 데이터 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중에서 제1 데이터 성분으로부터 얻어진 값을 포함하고,

제2 변종의 상기 자리올림이 있는 가산 명령에 응답하여, 상기 제2 소스 데이터 값은 상기 제2 데이터 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중에서 제2 데이터 성분으로부터 얻어진 값을 포함하는 장치.

**청구항 9**

제 1항 또는 제 2항에 있어서,

상기 목적지 벡터 레지스터의 적어도 한 쌍의 데이터 성분들 중에서 한 개에 각각 대응하는 적어도 한 개의 프레디케이트 표시를 지정하는 프레디케이트 값과 관련된 상기 벡터 자리올림이 있는 가산 명령의 프레디케이트된 변종에 응답하여, 상기 명령 디코딩 회로는, 상기 처리회로를 제어하여,

대응하는 프레디케이트 표시가 제1 값을 갖는 데이터 성분들의 쌍에 대해, 상기 가산의 결과에 대응하는 값으로 쌍의 상기 제1 데이터 성분의 갱신과 상기 가산의 상기 자리올림 출력으로 쌍의 상기 제2 데이터 성분의 갱신을 행하고,

대응하는 프레디케이트 표시가 제2 값을 갖는 데이터 성분들의 쌍에 대해, 상기 가산의 결과에 대응하는 값으로 쌍의 상기 제1 데이터 성분의 갱신과 상기 가산의 상기 자리올림 출력으로 쌍의 상기 제2 데이터 성분의 갱신을 금지하도록 구성된 장치.

**청구항 10**

제 1항 또는 제 2항에 있어서,

상기 벡터 자리올림이 있는 가산 명령의 가산 변종에 응답하여, 상기 가산은, 상기 제1 소스 데이터 값, 상기 제2 소스 데이터 값과 상기 자리올림 값의 가산을 포함하고,

상기 벡터 자리올림이 있는 가산 명령의 감산 변종에 응답하여, 상기 가산은 상기 제1 소스 데이터 값으로부터 상기 제2 소스 데이터 값의 감산을 포함하고, 상기 자리올림 값은 상기 감산에 대한 자리빌림(borrow) 값을 나타내고, 상기 자리올림 출력은 상기 감산의 자리빌림 출력을 포함하는 장치.

**청구항 11**

호스트 처리장치를 제어하여 타겟 프로그램 코드의 명령들을 실행하기 위한 명령 실행 환경을 제공하는, 컴퓨터 판독 가능한 기억 매체에 기억된, 컴퓨터 프로그램으로서,

상기 타겟 프로그램 코드의 명령들을 디코드하고 처리 프로그램 로직을 제어하여 데이터 처리를 행하는 명령 디코딩 프로그램 로직과,

복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들을 나타내는 데이터를 기억하는 벡터 레지스터 데이터 구조를 포함하고,

상기 명령 디코딩 프로그램 로직은, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터 및 자리올림 소스 벡터 레지스터를 지정하는 - 이때 적어도 상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 각각은 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정한다 - 벡터 자리올림이 있는 가산 명령에 응답하여, 상기 처리 프로그램 로직을 제어함으로써, 상기 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해, 상기 벡터 레지스터 데이터 구조를 갱신함으로써,

상기 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분으로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분을 갱신하고,

상기 가산의 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분을 갱신하고,

상기 가산의 결과에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분 및 상기 가산의 자리올림 출력에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분은 동일한 벡터 레지스터의 데이터 성분들인, 컴퓨터 프로그램.

**청구항 12**

데이터 처리방법으로서,

목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터 및 자리올림 소스 벡터 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령을 디코딩하는 단계를 포함하고, 이때 적어도 상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 각각은 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정하고,

상기 데이터 처리방법은,

상기 벡터 자리올림이 있는 가산 명령의 디코딩에 응답하여, 처리회로를 제어함으로써, 상기 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해,

상기 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 상기 제2 데이터 성분으로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 새로운 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분을 갱신하고,

상기 가산의 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분을 갱신하는 단계를 더 포함하고,

상기 가산의 결과에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분 및 상기 가산의 자리올림 출력에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분은 동일한 벡터 레지스터의 데이터 성분들인, 데이터 처리방법.

**청구항 13**

데이터 처리를 행하는 처리회로와,

명령들을 디코드하고 상기 처리회로를 제어하여 데이터 처리를 행하는 명령 디코딩 회로와,

복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들과,

상기 처리회로에 의해 행해진 연산들의 마스킹을 제어하기 위한 프레디케이트(predicate) 값들을 기억하는 복수의 프레디케이트 필드들을 포함하는 복수의 프레디케이트 레지스터들을 구비하고,

상기 명령 디코딩 회로는, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령에 응답하여, 상기 처리회로를 제어하여, 상기 목적지 벡터 레지스터의 지정된 데이터 성분에 대해,

상기 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 2 소스 데이터 값과, 상기 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 상기 지정된 데이터 성분을 갱신하고,

상기 가산의 자리올림 출력에 대응하는 값으로 상기 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드를 갱신하는 장치.

**청구항 14**

호스트 처리장치를 제어하여 타겟 프로그램 코드의 명령들을 실행하기 위한 명령 실행 환경을 제공하는, 컴퓨터 판독 가능한 기억 매체에 기억된, 컴퓨터 프로그램으로서,

상기 타겟 프로그램 코드의 명령들을 디코드하고 처리 프로그램 로직을 제어하여 데이터 처리를 행하는 명령 디코딩 프로그램 로직과,

복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들과, 상기 처리 프로그램 로직에 의해 행해진 벡터 연산들의 마스킹을 제어하는 프레디케이트 값들을 기억하는 복수의 프레디케이트 레지스터들을 나타내는 데이터를 기억하는 레지스터 데이터 구조를 포함하고,

상기 명령 디코딩 프로그램 로직은, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령에 응답하여, 처리회로를 제어하여, 상기 목적지 벡터 레지스터의 지정된 데이터 성분에 대해, 상기 레지스터 데이터 구조를 갱신함으로써,

상기 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 2 소스 데이터 값과, 상기 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 결과에 대응하는 값으로 상기 목적

지 벡터 레지스터의 상기 지정된 데이터 성분을 갱신하고,

가산의 자리올림 출력에 대응하는 값으로 상기 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드를 갱신하는 컴퓨터 프로그램.

**청구항 15**

데이터 처리방법으로서,

목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령을 디코딩하는 단계를 포함하고, 상기 입력 프레디케이트 레지스터와 상기 출력 프레디케이트 레지스터는 벡터 연산들의 마스킹을 제어하기 위한 프레디케이트 값들을 기억하는 복수의 프레디케이트 레지스터들 중에서 선택되고,

상기 데이터 처리방법은,

상기 벡터 자리올림이 있는 가산 명령의 디코딩에 응답하여, 처리회로를 제어하여, 상기 목적지 벡터 레지스터의 지정된 데이터 성분에 대해,

상기 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 2 소스 데이터 값과, 상기 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 상기 지정된 데이터 성분을 갱신하고,

상기 가산의 자리올림 출력에 대응하는 값으로 상기 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드를 갱신하는 단계를 더 포함하는 데이터 처리방법.

## *발명의 설명*

### *기 술 분 야*

[0001] 본 발명은 데이터 처리 분야에 관한 것이다.

### *배 경 기 술*

[0002] 일부 데이터 처리 시스템은, 복수의 데이터 성분을 포함하는 벡터 피연산자들에 작용하거나 벡터 피연산자를 발생하는 벡터 명령들의 처리를 지원한다. 한 개의 명령에 응답하여 다수의 별개의 데이터 성분들의 처리를 지원함으로써, 코드 밀도를 향상시킬 수 있으며, 스칼라 명령들을 사용하여 동일한 연산은 행하는 것과 비교하여 명령들의 페칭 및 디코딩의 오버헤드를 줄일 수 있다.

### *발명의 내용*

### *해결하려는 과제*

### *과제의 해결 수단*

[0003] 적어도 일부 실시예는,

[0004] 데이터 처리를 행하는 처리회로와,

[0005] 명령을 디코드하고 상기 처리회로를 제어하여 데이터 처리를 행하게 하는 명령 디코딩 회로와,

[0006] 복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들을 구비하고,

[0007] 상기 명령 디코딩 회로는, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터 및 자리올림(carry) 소스 벡터 레지스터를 지정하는 - 이때 적어도 상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 각각은 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정한다 - 벡터 자리올림이 있는 가산(vector add-with-carry)

명령에 응답하여, 상기 처리회로를 제어함으로써, 상기 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해,

[0008] 상기 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분으로부터 얻어진 자리올림 값(carry value)의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분을 갱신하고,

[0009] 상기 가산의 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분을 갱신하고,

상기 가산의 결과에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분 및 상기 가산의 자리올림 출력에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분은 동일한 벡터 레지스터의 데이터 성분들인, 장치를 제공한다.

[0010] 적어도 일부 실시예는, 호스트 처리장치를 제어하여 타겟 프로그램 코드의 명령들을 실행하기 위한 명령 실행 환경을 제공하는 컴퓨터 프로그램으로서,

[0011] 상기 타겟 프로그램 코드의 명령들을 디코드하고 처리 프로그램 로직을 제어하여 데이터 처리를 행하는 명령 디코딩 프로그램 로직과,

[0012] 복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들을 나타내는 데이터를 기억하는 벡터 레지스터 데이터 구조를 포함하고,

[0013] 상기 명령 디코딩 프로그램 로직은, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터 및 자리올림 소스 벡터 레지스터를 지정하는 - 이때 적어도 상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 각각은 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정한다 - 벡터 자리올림이 있는 가산 명령에 응답하여, 상기 처리 프로그램 로직을 제어함으로써, 상기 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해, 상기 벡터 레지스터 데이터 구조를 갱신함으로써,

[0014] 상기 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분으로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분을 갱신하고,

[0015] 상기 가산의 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분을 갱신하고,

상기 가산의 결과에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분 및 상기 가산의 자리올림 출력에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분은 동일한 벡터 레지스터의 데이터 성분들인, 컴퓨터 프로그램을 제공한다.

[0016] 적어도 일부 실시예는, 데이터 처리방법으로서,

[0017] 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터 및 자리올림 소스 벡터 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령을 디코딩하는 단계를 포함하고, 이때 적어도 상기 목적지 벡터 레지스터와 상기 자리올림 소스 벡터 레지스터 각각은 각 쌍이 제1 데이터 성분 및 제2 데이터 성분을 포함하는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정하고,

[0018] 상기 데이터 처리방법은,

[0019] 상기 벡터 자리올림이 있는 가산 명령의 디코딩에 응답하여, 처리회로를 제어함으로써, 상기 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해,

[0020] 상기 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 상기

제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 상기 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분으로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 새로운 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분을 갱신하고,

[0021] 상기 가산의 자리올림 출력에 대응하는 값으로 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분을 갱신하는 단계를 더 포함하고,

상기 가산의 결과에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제1 데이터 성분 및 상기 가산의 자리올림 출력에 대응하는 값으로 갱신된, 상기 목적지 벡터 레지스터의 데이터 성분들의 쌍 중에서 상기 제2 데이터 성분은 동일한 벡터 레지스터의 데이터 성분들인, 데이터 처리방법을 제공한다.

[0022] 적어도 일부 실시예는,

[0023] 데이터 처리를 행하는 처리회로와,

[0024] 명령들을 디코드하고 상기 처리회로를 제어하여 데이터 처리를 행하는 명령 디코딩 회로와,

[0025] 복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들과,

[0026] 상기 처리회로에 의해 행해진 연산들의 마스킹(masking)을 제어하기 위한 프레디케이트(predicate) 값들을 기억하는 복수의 프레디케이트 필드들을 포함하는 복수의 프레디케이트 레지스터들을 구비하고,

[0027] 상기 명령 디코딩 회로는, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령에 응답하여, 상기 처리회로를 제어하여, 상기 목적지 벡터 레지스터의 지정된 데이터 성분에 대해,

[0028] 상기 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 2 소스 데이터 값과, 상기 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 상기 지정된 데이터 성분을 갱신하고,

[0029] 상기 가산의 자리올림 출력에 대응하는 값으로 상기 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드를 갱신하는 장치를 제공한다.

[0030] 적어도 일부 실시예는,

[0031] 호스트 처리장치를 제어하여 타겟 프로그램 코드의 명령들을 실행하기 위한 명령 실행 환경을 제공하는 컴퓨터 프로그램으로서,

[0032] 상기 타겟 프로그램 코드의 명령들을 디코드하고 처리 프로그램 로직을 제어하여 데이터 처리를 행하는 명령 디코딩 프로그램 로직과,

[0033] 복수의 데이터 성분들을 포함하는 벡터 피연산자들을 기억하는 복수의 벡터 레지스터들과, 상기 처리 프로그램 로직에 의해 행해진 벡터 연산들의 마스킹을 제어하는 프레디케이트 값들을 기억하는 복수의 프레디케이트 레지스터들을 나타내는 데이터를 기억하는 레지스터 데이터 구조를 포함하고,

[0034] 상기 명령 디코딩 프로그램 로직은, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령에 응답하여, 처리회로를 제어하여, 상기 목적지 벡터 레지스터의 지정된 데이터 성분에 대해, 상기 레지스터 데이터 구조를 갱신함으로써,

[0035] 상기 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 2 소스 데이터 값과, 상기 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 상기 지정된 데이터 성분을 갱신하고,

[0036] 가산의 자리올림 출력에 대응하는 값으로 상기 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드를 갱신하는 컴퓨터 프로그램을 제공한다.

[0037] 적어도 일부 실시예는, 데이터 처리방법으로서,

[0038] 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령을 디코딩하는 단계를 포함하고, 상기 입력 프레디케이트 레지스터와 상기 출력 프레디케이트 레지스터는 벡터 연산들의 마스킹을 제어하기 위한 프레디케이트 값들을 기억하는 복수의 프레디케이트 레지스터들 중에서 선택되고,

[0039] 상기 데이터 처리방법은,

[0040] 상기 벡터 자리올림이 있는 가산 명령의 디코딩에 응답하여, 처리회로를 제어하여, 상기 목적지 벡터 레지스터의 지정된 데이터 성분에 대해,

[0041] 상기 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 1 소스 데이터 값, 상기 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제 2 소스 데이터 값과, 상기 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 상기 목적지 벡터 레지스터의 상기 지정된 데이터 성분을 갱신하고,

[0042] 상기 가산의 자리올림 출력에 대응하는 값으로 상기 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드를 갱신하는 단계를 더 포함하는 데이터 처리방법을 제공한다.

[0043] 본 발명의 또 다른 발명내용, 특징 및 이점은 첨부도면을 참조하여 주어지는 이하의 실시예의 설명으로부터 명백해질 것이다.

## *발명의 효과*

## *도면의 간단한 설명*

[0044] 도 1은 벡터 명령들의 실행을 지원하는 데이터 처리장치의 일례를 개략적으로 나타낸 것이다.

도 2는 프레디케이션을 갖는 벡터 연산의 일례를 나타낸 것이다.

도 3 및 도 4는 제1 실시예에 따른 벡터 자리올림이 있는 가산 명령의 가산 및 감산 변종을 나타낸 것이다.

도 4a는 벡터 자리올림이 있는 가산을 갖는 가산 명령의 프레디케이트된 변종을 나타낸 것이다.

도 5는 벡터 자리올림이 있는 가산 명령의 처리방법을 나타낸 흐름도이다.

도 6은 카라츠바 알고리즘에 따른 긴 정수 값의 승산의 벡터화의 일례를 나타낸 것이다.

도 7은 64비트의 벡터 성분 사이즈를 사용하여, 2개의 256비트 값들의 승산을 행할 때 가산되는 부분곱들의 일례를 나타낸 것이다.

도 8은 도 3에 나타낸 벡터 자리올림이 있는 가산 명령을 포함하는 명령들의 예시적인 시퀀스를 나타낸 것이다.

도 9는 벡터 자리올림이 있는 가산 명령을 사용한 명령 시퀀스의 제2 실시예를 나타낸 것이다.

도 10a 내지 도 10i는 도 9의 명령의 처리를 나타낸 일련의 도면을 나타낸 것이다.

도 11은 프레디케이트 레지스터들을 사용하여 가산 또는 감산에 대한 자리올림 값을 옮기는 벡터 자리올림이 있는 가산 명령의 제 2 실시예를 나타낸 것이다.

도 12는 도 11의 실시예에 따른 벡터 자리올림이 있는 가산 명령의 처리를 나타낸 흐름도이다.

도 13은 사용될 수 있는 시뮬레이터 실시예를 나타낸 것이다.

## *발명을 실시하기 위한 구체적인 내용*

[0045] 일부의 처리 작업량은 1024비트, 2048비트 또는 4096비트 등의 매우 큰 정수값에 수학적 연산이 적용되는 것을 요구한다. 예를 들어, 암호화 작업량은 큰 수를 인수분해하는 난이도에 의존하므로, 메시지를 암호화

또는 복호화하기 위해 이와 같은 큰수를 곱할 필요가 있다. 예를 들어, RSA 2048에서는, 4096비트의 곱을 생성하기 위해 2048비트 수들의 다수의 연속된 승산이 존재한다. 일부 금융 어플리케이션도 긴 정수값이 처리되는 것을 요구한다. 한 개의 피연산자의 각 숫자를 나머지 피연산의 각 숫자로 곱하는 표준 교과서적인 접근방법을 사용하여 곱하는 경우, 승산을 행하는 복잡도가 보통 숫자들의 개수의 제곱에 비례하여 증가할 것이다. 그러나, 숫자들의 수에 따라 계산 오버헤드가 더 느리게 비례하여 증가하도록 복잡성을 줄이는 일부 알고리즘이 알려져 있다. 예를 들어, 카라츠바 알고리즘은 곱해질 긴 숫자를 개별적으로 곱해질 더 적은 부분들로 분할하여, 필요한 승산의 수와 일부 추가적인 가산, 감산 또는 자리옮김 연산의 타협을 가능하게 한다. 이 때문에, 긴 정수들의 승산이 다수의 더 작은 승산들과 다수의 가산, 감산 및 자리옮김으로 분해된다. 벡터 명령들은, 서로 다른 서브 승산들을 서로 다른 벡터 레인(lane)에 매핑하여 한 개의 명령이 다수의 독립적인 서브 승산들의 부분들의 계산을 제어함으로써, 이와 같은 계산의 속도를 올리는데 사용될 수도 있다. 이와 같은 경우, 승산의 다양한 부분 곱들의 일련의 가산 또는 삼산을 행할 때 많은 작업량이 효율적으로 발생된다. 그러나, 한 개의 가산에서 발생되는 자리올림 정보가 동등한 가중치를 갖는 값에 행해진 다음 가산에 입력될 필요가 있으므로, 한 개의 명령으로부터의 자리올림 정보를 또 다른 명령에 대한 입력으로서 사용하기 위해 유지할 필요가 잇기 때문에, 부분 곱 가산들 사이에서의 자리올림 정보의 옮김을 관리함에 있어서 설계상의 문제가 발생한다.

[0046] 한가지 접근방법은, 각각의 벡터 레인 내에서 처리되고 있는 실제 데이터의 성분 사이즈를 효율적으로 줄이고, 이 레인에서 가산으로부터 발생되는 자리올림 정보를 기억하기 위해 각각의 레인의 상단에 한 개 이상의 비트를 예약함으로써, 추후의 명령에 의해 판독될 수 잇도록 하는 것일 수 있다. 그러나, 본 발명자는, 이와 같은 접근방법에서는, (벡터 레인들을 완전히 점유하는 성분들을 갖는) 데이터의 입력 벡터들을 언팩(unpack)하고 각각의 레인 내부에 추가적인 자리올림 정보를 포함시킬 필요성으로 인해 이들 성분들의 사이즈를 조정하여 이들 성분이 줄어든 유효 데이터 성분 사이즈를 갖고 벡터 레인들에 걸쳐 분할하기 위해 다수의 추가적인 명령들이 필요할 수도 있다는 것을 인식하였다. 또한, 입력 벡터가 지정된 성분 사이즈를 갖는 다수의 성분들을 포함하는 경우, 자리올림을 위해 예약된 공간을 갖는 레인들에 걸쳐 분할될 때, 동일한 양의 입력 데이터가 더 큰 총수의 벡터 레인들을 필요로 하게 된다. 예를 들어, 4개의 64비트 성분들의 입력 벡터들이, 실제 데이터에 대해 56비트를 갖고 자리올림을 위해 예약된 8비트를 갖는 벡터 레인들에 걸쳐 분할될 때, 56비트의 4개의 레인들 각각이 224비트만 제공하여 최종 32비트를 위해 5번째 레인이 필요하므로, 입력 데이터의 동일한 수의 비트(256비트)를 수용하기 위해 5개의 레인이 할당되는 것을 요구한다. 계산할 부분 곱의 수는 벡터의 성분들의 수의 제곱에 비례하여 증가하므로, 이것은 필요한 승산 및 부분 곱 가산의 수를 상당히 증가시킬 수 있어, 성능을 더욱 더 줄일 수 있다.

[0047] 이하에서 설명하는 기술은, 명령이 전체 벡터 레인을 점유하는 소스 데이터 성분들에 대해 직접 작용할 수 있기 때문에, 입력 데이터의 언팩킹의 필요가 없도록 벡터 자리올림이 있는 가산 명령을 구현하는 기술을 제공한다.

[0048] 한가지 접근방법에서, 벡터 자리올림이 있는 가산 명령은 목적지 벡터 레지스터, 제1 및 제 2 데이터 소스 레지스터들 및 자리올림 소스 벡터 레지스터를 지정한다. 적어도 목적지 벡터 레지스터와 자리올림 소스 벡터 레지스터 각각은 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자들을 지정하고, 이때 각 쌍은 제 1 데이터 성분 및 제 2 데이터 성분을 포함한다. 명령에 응답하여, 명령 디코더는 처리회로를 제어하여, 목적지 벡터 레지스터의 각 쌍의 데이터 성분들에 대해, 제1 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 제2 데이터 소스 벡터 레지스터의 선택된 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분으로부터 얻어진 자리올림 값의 가산의 결과에 대응하는 값으로 목적지 벡터 레지스터의 데이터 성분들의 쌍 중 제1 데이터 성분을 갱신한다. 또한, 목적지 벡터 레지스터의 데이터 성분들의 쌍 중 제2 데이터 성분은 가산의 자리올림 출력에 대응하는 값으로 갱신된다.

[0049] 이 때문에, 목적지 벡터 레지스터의 각 쌍의 성분들 중에서 제2 데이터 성분이 가산의 자리올림 출력을 제공하기 위해 예약되고, 마찬가지로 자리올림 소스 벡터 레지스터 내부의 각 쌍의 제2 데이터 성분이 가산에 대한 자리올림 입력을 제공하기 위해 사용된다. 따라서, 목적지 및 자리올림 소스 벡터 레지스터들은 대응하는 레이아웃을 갖고, 실제로 이들 명령이 실제로 사용될 때 자주, 벡터 자리올림이 있는 가산 명령의 한 개의 인스턴스(instance)로부터 발생된 목적지 벡터 레지스터가 벡터 자리올림이 있는 가산 명령의 다음의 인스턴스에 대한 자리올림 소스 벡터 레지스터로서 사용된다.

[0050] 이와 같은 접근방법은, 소스 데이터와 가산의 결과값들을 전달하는데 이용 가능한 데이터 성분들의 수를 효율적으로 절반으로 줄여, 벡터 레지스터 파일의 이용을 가능한한 증가시킴으로써 계산의 효율을 향상시킨

다는 목적의 벡터 프로세서의 통상의 설계 원리에 반대되므로, 직관에서 벗어난다. 즉, 자리올림 값을 제공하기 위한 절반의 벡터 레인을 이용하는 것은, 벡터 프로세서와 벡터 레지스터 파일의 용량의 절반을 소모하는 것처럼 보일 것이며, 지정된 수의 데이터 성분들을 처리하는데 필요한 명령들의 수를 두배로 만들 것이다.

[0051] 그러나, 놀랍게도 본 발명자들은, 주어진 작업량을 실행하기 위해 더 많은 수의 자리올림이 있는 가산 명령이 필요하지만, 각 쌍의 성분들 중에서 제2 데이터 성분에 자리올림을 배치하는 것의 이점은, 제1 데이터 성분이 전체 레인 사이즈를 점유할 수 있다는 것으로, 이것은 입력 벡터들로부터의 성분들의 언팩킹이나 사이즈 조정이 필요하지 않으므로, 벡터 자리올림이 있는 가산 명령이 팩킹된 소스 데이터에 직접 작용할 수 있다는 것을 의미한다는 것을 인식하였다. 실제로, 긴 정수의 승산을 포함하는 작업량에 대해, 벡터 자리올림이 있는 가산 명령에 의해 처리된 사실상 줄어든 수의 성분들을 보상하기 위해 추가적인 명령들이 필요하지만, (a) 언팩킹/사이즈 조정 오버헤드를 피하는 것과, (b) 전술한 인레인(in-lane) 자리올림 접근방법으로 유효 레인 사지를 줄임으로써 발생되는 부분 곱의 수의 증가를 피하는 것에 의해, 전체적인 성능이 더 높다는 것이 밝혀졌다. 이 때문에, 성능이 전체적으로 향상된다. 어쨌든, 일부 마이크로 아키텍처에서는, 한 개의 벡터 자리올림이 있는 가산 명령이 아키텍처 레벨에서 절반의 성문들에 대해 작용하더라도, 조합하여 모든 성분들에 대해 작용하는 2개의 인스턴스의 명령들이 함께 "융합되어(fused)" 처리 파이프라인에 의해 한 개의 마이크로연산으로서 처리되거나, 병렬로 처리됨으로써, 마이크로 연산 당 처리되는 성분들의 수를 절반으로 줄임으로써 발생되는 외견상의 성능 손실을 피한다. 이것은, 예를 들어, 마이크로 아키텍처가 결과 버스트에 대한 다수의 동시 기록을 지원하는 경우에 가능하다. 또한, 독립적인 자리올림 체인(carry-chain)의 형태로 병렬화도 이용가능하다.

[0052] 일부 구현예에서, 목적지 벡터 레지스터는 제1 및 제2 소스 벡터 레지스터들 및 자리올림 소스 벡터 레지스터와 별개로 명령에 의해 지정되어, 결과가 모든 소스 레지스터들과 다른 레지스터들에 기록될 때, 명령이 실행된 후 레지스터 파일에 모든 소스 벡터를 유지하는 비파괴적인 명령 포맷을 제공할 수도 있다.

[0053] 그러나, 다른 구현예에서는, 제1 데이터 소스 벡터 레지스터는 목적지 벡터 레지스터와 동일한 레지스터이며, 각각의 쌍에 대해, 제1 소스 데이터 값은 목적지 벡터 레지스터의 데이터 성분들의 쌍 중 제 1 데이터 성분의 이전 값을 포함한다. 실제로, 오래된 누산기 값이 제2 소스 데이터 값과 자리올림 값에 가산되어 동일한 레지스터에 다시 기록되는 누산 연산은, (승산의 각각의 부분 곱들을 가산하기 위해) 긴 정수의 승산을 포함하는 다수의 작업량에 유용하므로, 제1 소스 데이터 벡터 레지스터가 목적지 벡터 레지스터에 독립적으로 지정되는 것이 필수적인 것은 아니다. 한 개의 레지스터를 목적지 및 제1 데이터 소스 벡터 레지스터 양쪽으로서 지정함으로써, 명령 세트 아키텍처의 명령 인코딩 공간을 다른 용도를 위해 보존할 수 있다. (동일한 레지스터를 목적지 및 소스 레지스터들 중에서 한 개로서 사용하는) 파괴적인 명령 포맷의 또 다른 이점은, 때때로 명령이 필요로 하는 소스 레지스터 포트들의 수를 제한하는 것이 필요하고, 일부 마이크로 아키텍처에서는 (목적지 레지스터의 프레디케이트된 레인들이 그들의 이전 값을 유지하는) 병형 프레디케이션을 갖는 연산의 경우에 목적지 레지스터가 이미 판독될 필요가 있을 때, (제2 소스 벡터 레지스터 및 자리올림 벡터 레지스터에 대한) 단지 2개의 추가적인 레지스터 액세스를 갖는 것이 유익하다는 것이다.

[0054] 벡터 피연산자의 벡터 성분 사이즈 및/또는 전체 길이가 변동될 수 있다. 일부 구현예는 성분 사이즈 및/또는 벡터 길이를 특정한 상수값으로 고정할 수도 있다. 그러나, 다른 시스템은 가변의 벡터 성분 사이즈 및 벡터 길이를 지원한다. 일부의 벡터 성분 사이즈 또는 벡터 길이에 대해, 벡터가 단지 2개의 성문만 포함할 수도 있으며, 이 경우에 목적지 벡터 레지스터와 자리올림 소스 벡터 레지스터는 한 쌍의 데이터 성분, 즉 한 개의 제1 데이터 성분 및 한 개의 제 2 데이터 성분을 포함할 수도 있다.

[0055] 그러나, 명령 디코딩 회로와 처리회로가, 목적지 및 자리올림 소스 벡터 레지스터들이 다수의 쌍의 데이터 성분들을 포함하는, 즉 이들이 적어도 4개의 성분들(제1 및 제2 데이터 성분들 각각의 2개 이상)을 포함하는 벡터 자리올림이 있는 가산 명령의 적어도 한 개의 인스턴스의 실행을 지원하는 것이 유용하다. 이것은 시스템이 한 개의 명령에 응답하여 복수의 가산의 계산을 지원할 수 있도록 하며, 이것은 긴 정수 피연산자들의 승산을 포함하는 계산을 가속화하는데 유용할 수 있다. 예를 들어, 각 쌍의 성분들은 카라츠바 알고리즘의 다른 서브 승산에 대한 부분 곱들의 가산을 표시할 수도 있다.

[0056] 벡터 피연산자의 성분들에 대한 데이터 성분들의 쌍들의 매핑은 다양하게 행해질 수 이다. 예를 들어, 일 실시예에서는, 각 쌍 중 제1 성분들이 레지스터의 한 개의 절반부 내에 배치되고 각 쌍의 제2 성분들이 나머지 절반부 내에 배치될 수도 있다. 예를 들어, 4 쌍의 제1 및 제2 성분들을 갖는 예에서, 이들은 순서 2d, 2c, 2b, 2a, 1d, 1c, 1b, 1a로 배치될 수도 있다(이때, 1a는 제2 성분 2a에 대응하는 제1 성분이고, 쌍들 b, c 및 d에 대해서도 마찬가지이다).

[0057] 그러나, 일 구현예에서 각 쌍의 데이터 성분들은 한쌍의 인접한 데이터 성분들을 포함한다. 이 때문에, 목적지 벡터 레지스터와 자리올림 소스 벡터 레지스터의 데이터 성분들의 쌍들의 수가 적어도 2일 때, 이들 쌍들의 제1 데이터 성분들이 이들 쌍의 제2 데이터 성분들과 인터리브된다. 예를 들어, 4 쌍의 제1 및 제2 성분들에 대해, 이들이 순서 2d, 1d, 2c, 1c, 2b, 1b, 2a, 1a로 배치될 수도 있다(이와 달리, 각 쌍의 제1 및 제2 성분의 순서가 뒤바뀔 수도 있다). 이와 같은 접근방법은, 벡터 명령의 주어진 서브 계산에서 결합되는 소스 피연산자들과 결과값들이, 2개 이상의 벡터 레인들에 걸치는 더 길이가 긴 교차 레인 신호 경로를 필요로 하는 것이 아니라, 동일한 벡터 레인 내에만, 또는 곧바로 인접하는 벡터 레인 내부에만 가로질러 제한될 수 있기 때문에, 하드웨어 설계를 더욱 효율적으로 만들 수 있으므로, 하드웨어로 구현하는데 더욱 효율적일 수 있다. 이 때문에, 각 쌍의 제1 및 제2 데이터 성분들을 인터리브함으로써 필요한 배선의 복잡도와 길이를 줄일 수 있다.

[0058] 자리올림 값은 자리올림 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제2 데이터 성분의 최하위 비트로부터 얻어진다. 벡터 자리올림이 있는 가산 명령에 응답하여, 처리회로는 가산의 자리올림 출력에 대응하는 값으로 목적지 벡터 레지스터의 데이터 성분들의 쌍 중 제2 데이터 성분의 최하위 비트를 갱신한다. 자리올림을 제공하기 위해 제2 데이터 성분의 최하위 비트를 사용하는 것은, 이 최하위 비트가 대응하는 쌍 중 제1 데이터 성분의 최상위 비트에 인접하여, 자리올림 및 데이터 값들이 목적지 벡터 레지스터의 인접한 부분에 기록되게 할 수 있으므로, 더욱 효율적일 수 있다. 최하위 비트 이외의 제2 데이터 성분의 나머지 비트들은 자리올림 소스 벡터 레지스터와 누산기 벡터 레지스터에서 사용되지 않는다. 다수의 비트를 사용되지 않은 상태로 남기는 것은 벡터 레지스터 파일 내부의 공간을 소모하고 벡터 처리 유닛 내부의 처리 자원을 소모하는 것처럼 보이지만, 전술한 것과 같이, 이와 같은 접근방법에 의해 긴 정수의 승산을 행할 때 전체적인 성능이 향상될 수 있다.

[0059] 자리올림이 있는 가산 명령에 의해 처리된 벡터들은 $2^N$ 비트를 갖는 데이터 성분들을 갖고, 제2 소스 데이터 값도 $2^N$ 비트를 포함하고, 이때 N은 정수이다. 이 때문에, 제2 소스 데이터 값이 2의 거듭제곱의 비트 수에 대응하는 벡터 레인들 그 자체와 동일한 사이즈를 가지므로, 제2 소스 데이터 값이 전체 벡터 레인을 점유하여, 벡터 자리올림이 있는 가산 명령이 입력에 대해 동작하도록 하기 위해 입력 벡터 또는 이전의 벡터 명령들의 결과로부터의 데이터 성분들의 사이즈 조정이나 언패킹이 필요하지 않다. 그 대신에, 벡터 자리올림이 있는 가산 명령은 입력 벡터로부터의 선택된 성분에 대해 직접 작용할 수 있다. 이것은 데이터 결과와 별개의 벡터 성분 내에 자리올림이 유지되므로 가능하다.

[0060] 제2 소스 데이터 값은 제2 데이터 소스 벡터 레지스터의 모든 성분으로부터 얻어진다. 제2 소스 데이터 벡터 레지스터의 다른 성분들로부터 소스 데이터 값을 선택하기 위해 명령의 서로 다른 변종들이 제공될 수도 있다.

[0061] 일 실시예에서, 제2 데이터 소스 벡터 레지스터는, 목적지 및 자리올림 소스 벡터 레지스터들 내부의 성분들의 쌍들과 대응하게 배치되는 적어도 한 쌍의 데이터 성분들을 포함하는 피연산자를 제공한다. 예를 들어, 각 쌍의 제1 및 제2 데이터 성분들이 인터리브될 수도 있다. 벡터 자리올림이 있는 가산 명령의 제1 및 제2 변종이 제공될 수 있다. 제1 변종에 대해, 제2 소스 데이터 값은 제2 데이터 소스 벡터 레지스터의 대응하는 쌍의 데이터 성분들 중 제1 데이터 성분을 포함한다(그리고, 명령의 결과는 각 쌍의 제2 데이터 성분에 무관하다). 제2 변종에 대해, 제2 소스 데이터 값은 대응하는 쌍의 제2 데이터 성분을 포함한다(그리고, 이 명령의 결과는 각 쌍의 제1 성분에 무관하다). 서로 다른 성분들을 제2 소스 데이터 값으로서 선택하기 위한 명령의 2가지 변종을 제공함으로써, 명령의 2가지 변종이 조합하여 제2 소스 벡터의 각각의 성분을 처리하므로, 2개의 벡터 자리올림이 있는 가산 명령들(제1/제2 변종들 각각 중 한 개)을 실행하기 전에 제2 소스 벡터의 성분들의 추가적인 재배열 또는 언패킹을 행할 필요가 없이 입력 벡터의 모든 성분들이 처리되게 할 수 있다.

[0062] 제1 데이터 소스 벡터 레지스터가 명령 벡터 레지스터와 별개로 명령에 의해 지정되는 실시예에서, 제1 소스 데이터 값은 마찬가지로 (제1 변종의 명령에 대해) 각 쌍의 제1 성분으로부터 또는 (제2 변종의 명령에 대해) 각 쌍의 제2 성분으로부터 추출될 수도 있다.

[0063] 한편, 제1 데이터 소스 벡터 레지스터가 목적지 벡터 레지스트와 같은 레지스터이면, 목적지 벡터 레지스터의 제2 데이터 성분이 자리올림 정보를 표시하는데 사용될 것이므로, (실행중인 명령 변종이 제1 변종인지 제2 변종인지에 무관하게) 각각의 가간에 대해 제1 데이터 소스 값이 목적지 벡터 레지스터 내부의 대응하는 쌍의 성분들 중 제1 데이터 성분으로부터 추출된다.

[0064] 또한, 자리올림이 있는 가산 명령의 프레디케이트된 변종이 제공되는데, 이것은 적어도 한 개의 프레디

케이트 표시를 지정하는 프레디케이트 값과 관련되며, 각각의 프레디케이트 표시는 목적지 벡터 레지스터의 적어도 한 쌍의 데이터 성분들 중에서 한 개에 대응한다. 프레디케이트된 변종에 응답하여, 명령 디코딩 회로는, 처리회로를 제어하여, 대응하는 프레디케이트 표시가 제1 값을 갖는 목적지 벡터 레지스터의 한 쌍의 데이터 성분들에 대해 전술한 것과 같은 제1/제2 데이터 성분들의 갱신을 행하고, 대응하는 프레디케이트 표시가 제2 값을 갖는 한 쌍의 데이터 성분들에 대해 갱신을 금지한다. 프레디케이트 값은 프레디케이트 값을 기억하는 프레디케이트 레지스터를 식별하는 프레디케이트 레지스터 지정자를 포함하는 명령 인코딩에 의해 명령과 관련되거나, 디폴트 프레디케이트 레지스터를 사용하여 명령의 인코딩에 무관하게 자리올림이 있는 가산 명령의 프레디케이트 변종의 모든 인스턴스들에 대해 프레디케이트 값을 제공할 수도 있다. 대응하는 프레디케이트 표시가 제2 값을 갖는 목적지 벡터 레지스터의 한 쌍의 성분들에 대해, 이 쌍의 성분들은 목적지 벡터 레지스터의 이 부분에 기억된 이전 값들을 유지하거나, 제로값이나 다른 소정의 값으로 클리어될 수도 있다. 이 때문에, 자리올림이 있는 가산 명령이 프레디케이트될 때, 벡터의 개별적인 데이터 성분들의 입상도(granularity)에 작용하는 것이 아니라, 프레디케이션이 데이터 성분들의 쌍들의 입상도에 작용한다.

[0065] 벡터 자리올림이 있는 가산 명령의 가산 및 감산 변종이 제공될 수 있다. 가산 변종에 대해, 가산은, 제1 및 제2 소스 데이터 값들과, 자리올림 소스 벡터 레지스터의 제2 데이터 성분으로부터 얻어진 자리올림 값의 가산을 포함한다. 감산 변종에 대해, 이 연산은, 제1 소스 데이터 값으로부터 제2 소스 데이터 값의 감산을 포함하고, 자리올림 값은 감산에 대한 자리빌림(borrow) 값을 표시한다. 감산 변종에 대해, 자리올림 출력은 감산의 자리빌림 출력을 표시한다. 이때, 피연산자들 중에서 한 개가 가산을 행하기 전에 2의 보수일 때 2개의 피연산자들의 감산은 2개의 피연산자들의 가산과 같기 때문에, 감산은 여전히 가산으로 간주할 수 있다. 마찬가지로, 가산에 대한 +1의 자리올림 수 대신에 자리빌림 값이 단순히 -1의 자리올림 수에 대응하므로, 감산에 대한 자리빌림 값은 가산에 대한 자리올림 값으로서 간주할 수 있다.

[0066] 벡터 자리올림이 있는 가산 명령의 한 개보다 많은 수의 변종이 제공되는 경우, 변종들(제1/w[2, 또는 가산/감산)은 다양하게 구별될 수 있다. 예를 들어, 자리올림이 있는 가산 명령의 다양한 변종들은 다양한 명령 옵코드들을 가질 수 있다. 이와 달리, 변종들은 공통된 옵코드를 공유하지만, 변종들을 구별하는 명령 인코딩 내부에 또 다른 필드를 가질 수도 있다. 또 다른 예에서, 명령의 다양한 변종들은 동일한 명령 인코딩을 가질 수 있지만, 다음의 벡터 자리올림이 있는 가산 명령을 마주칠 때 명령의 어떤 변종을 사용해야 할 것인지 선택하기 위해 이전의 명령에 의해 설정되는, 장치의 제어 레지스터에 기억된 모드 파라미터에 의해 구별될 수도 있다. 제1/제2 변종에 대해, 제1/제2 성분 선택은 명령에 의해 판독된 프레디케이트 또는 마스크 레지스터에 표시될 수도 있다.

[0067] 가산/감산을 위한 자리올림 정보를 옮기기 위해 자리올림 소스 벡터 레지스터 및 목적지 벡터 레지스터의 일부 성분들을 사용하는 전술한 벡터 자리올림이 있는 가산 명령의 형태의 이점은, 명령에 응답하여 단지 한 개의 레지스터(목적지 벡터 레지스터)만 기록될 필요가 있으므로, 이것이 마이크로 아키텍처로 구현하기 위해 비교적 효율적이라는 것이다. 또한, 누산기 레지스터가 목적지 및 제1 소스 벡터 레지스터 양쪽으로서 지정되는 실시예에서는, 단지 한 개의 목적지 레지스터 지정자와 2개의 벡터 소스 레지스터가 명령에 의해 지정될 필요가 있다.

[0068] 그러나, 큰 정수값에 대해 계산을 구현하는 문제에 대한 대안적인 해결책은, 프레디케이트 레지스터를 사용하여 자리올림 정보를 옮기는 벡터 자리올림이 있는 가산 명령을 제공하는 것일 수 있다. 데이터 처리장치는, 처리회로에 의해 행해진 연산의 마스킹을 제어하기 위한 프레디케이트 값들을 기억하는 프레디케이트 필드들을 포함하는 다수의 프레디케이트 레지스터들을 갖는다. 다른 종류의 벡터 명령에 대해서는 프레디케이트 필드가 벡터 처리의 레인들의 마스킹을 제어하지만, 벡터 자리올림이 있는 가산 명령에 대해서는, 프레디케이트 레지스터를 재사용하여 가산으로부터 자리올림 출력을 표시할 수 있으며, 가산에 대한 자리올림 입력도 프레디케이트 레지스터의 프레디케이트 필드로부터 얻어진다.

[0069] 이 때문에, 명령 디코딩 회로는, 목적지 벡터 레지스터, 제1 데이터 소스 벡터 레지스터, 제2 데이터 소스 벡터 레지스터, 입력 프레디케이트 레지스터 및 출력 프레디케이트 레지스터를 지정하는 벡터 자리올림이 있는 가산 명령에 응답하여, 처리회로를 제어하여, 목적지 벡터 레지스터의 지정된 데이터 성분에 대해, 제1 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제1 소스 데이터 값, 제2 데이터 소스 벡터 레지스터의 대응하는 데이터 성분으로부터 얻어진 제2 소스 데이터 값과, 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드로부터 얻어진 자리올림 값의 가산 결과에 대응하는 값으로 목적지 벡터 레지스터의 지정된 데이터 성분을 갱신한다. 또한, 출력 프레디케이트 레지스터의 대응하는 프레디케이트 필드가 상기 가산의

자리올림 출력에 대응하는 값으로 갱신된다.

[0070] 이와 같은 접근방법은 더욱 복잡한 마이크로 아키텍처(예를 들면, 프레디케이트 레지스터 파일에 대한 더 빠른 판독/기록 경로와 동일한 명령에 응답하여 벡터 레지스터 파일과 프레디케이트 레지스터 파일 모두에 기록할 수 있는 능력)를 요구하지만, 이와 같은 접근방법의 이점은, 자리올림 수가 목적지 벡터 레지스터의 성분들 내부에 기억되지 않으므로, 목적지 벡터 레지스터의 모든 성분을 사용하여 추가적인 결과값을 기억할 수 있어(그리고 제1/제2 데이터 소스 벡터 레지스터들의 모든 성분이 소스 입력으로서 사용될 수 있어), 한 개의 명령에 응답하여 행해질 수 있는 연산의 수를 효율적으로 배가하므로, 성능을 향상시킬 수 있다.

[0071] 요약하면, 전술한 실시예들 모두는, 자리올림 정보가 데이터 결과와 같은 레인에 기억되지 않고, 벡터 레지스터의 또 다른 레인에 기억되거나, 프레디케이트 레지스터의 대응하는 프레디케이트 필드 내부에 기억되기 때문에, 이들 실시예는 언패킹 또는 사이즈 조정 동작이 필요가 없이 입력 벡터들로부터의 전체 데이터 성분들에 대해 직접 동작할 수 있다. 이것은 긴 정수들의 승산을 가속화하는데 매우 유용하다.

[0072] 전술한 모든 형태의 명령은, 처리회로를 제어하여 벡터 자리올림이 있는 가산 명령에 응답하여 필요한 연산을 행하는 명령 디코더를 사용하여 하드웨어로 구현될 수 있다. 예를 들여, 명령 디코더는, 벡터 자리올림이 있는 가산 명령의 인코딩을 해독하여, 처리회로를 제어하여 전술한 것과 같이 입력에 근거하여 결과 레지스터들을 갱신하기 위한 적절한 제어 신호 경로를 선택적으로 활성화하는 논리 게이트들을 구비한다.

[0073] 그러나, 이 기술은, 물리적인 하드웨어 대신에, 프로세서 아키텍처의 시뮬레이션으로 구현될 수도 있다. 이 때문에, (그 자체가 전술한 명령을 지원하지 않는) 호스트 처리장치를 제어하여, 타겟 프로그램 코드의 명령들을 실행하여 이들 명령을 지원하지 않은 타겟 처리장치 상에서 타겟 프로그램 코드의 실행을 시뮬레이션하는 명령 실행 환경을 제공하는 시뮬레이터 컴퓨터 프로그램이 제공된다. 명령 디코더와 처리회로의 기능은 그 대신에 시뮬레이터 프로그램에서의 명령 디코딩 프로그램 로직과 처리 프로그램 로직에 의해 행해질 수 있으며, 레지스터들은 시뮬레이션된 타겟 처리장치의 레지스터들을 표시하는 데이터를 기억하는 메모리 내부의 시뮬레이션된 레지스터 데이터 구조로서 구현될 수도 있다. 이 컴퓨터 프로그램은 기억매체에 기억된다. 기억매체는 비일시적인 기억매체이어도 된다.

[0074] 도 1은 명령 디코더(6)에 의해 디코드된 명령들에 응답하여 데이터 처리 연산을 행하는 처리회로(4)를 갖는 데이터 처리장치(2)의 일례를 개략적으로 나타낸 것이다. 프로그램 명령들은 한 개 이상의 캐시 및 메모리를 포함하는 메모리 시스템(8)으로부터 페치된다. 이들 명령은 명령 디코더(6)에 의해 디코드되어 제어신호를 발생하고, 제어신호는 처리회로(4)를 제어하여 명령들을 처리하여 명령 세트 아키텍처에 의해 정의된 대응하는 연산을 행한다. 예를 들어, 디코더(6)는, 옵코드들과 명령들의 추가적인 제어 필드들을 해석하여 처리회로(4)에게 산술 연산, 논리 연산, 또는 로드/스토어 연산 등의 연산을 행하기 위한 적절한 하드웨어 유닛들을 기동시키는 제어신호를 발생하는 로직 게이트들을 구비한다. 레지스터들(10)은, 명령들에 응답하여 처리회로(4)에 의해 처리할 데이터 값들과, 처리회로의 동작을 환경설정하기 위한 제어정보를 기억한다. 산술 또는 논리 명령들에 응답하여, 처리회로(4)는 레지스터들(10)로부터 피연산자들을 판독하고 명령들의 결과를 다시 레지스터들(10)에 기록한다. 로드/스토어 명령들에 응답하여, 데이터 값들이 처리 로직(4)을 거쳐 레지스터들(10)과 메모리 시스템(8) 사이에서 전달된다.

[0075] 레지스터들(10)은 한 개의 데이터 성분을 포함하는 스칼라 값들을 기억하는 다수의 스칼라 레지스터들을 구비한다. 스칼라 레지스터들은 정수 피연산자들을 기억하는 레지스터들과 부동소수점 값들을 기억하는 부동소수점 레지스터들을 구비할 수도 있다. 이와 달리, 정수 및 부동소수점 값들이 동일한 세트의 레지스터들에 기억될 수도 있다. 명령 세트 아키텍처에서 명령 디코더(6)에 의해 지원되는 일부 명령들은 처리회로(4)를 제어하여 스칼라 레지스터들(12)로부터 판독된 스칼라 피연산자들을 처리하여 스칼라 레지스터(12)에 다시 기록할 스칼라 결과를 발생하는 스칼라 명령들이다.

[0076] 레지스터들은 벡터 레지스터 파일(14)과 프레디케이트 레지스터 파일(16)을 더 포함한다. 벡터 레지스터 파일(14)은 복수의 데이터 성분들을 포함하는 벡터 피연산자의 기억을 지원하는 다수의 벡터 레지스터들을 구비한다. 명령 디코더(6)는 처리회로(4)를 제어하여 벡터 레지스터들로부터 판독된 벡터 피연산자의 각각의 성분들에 대해 다수의 레인들의 벡터 처리를 행하여, 스칼라 레지스터들(12)에 기록할 스칼라 결과 또는 벡터 레지스터(14)에 기록할 추가적인 벡터 결과를 발생하는 벡터 명령들을 지원한다. 일부 벡터 명령들은 한 개 이상의 스칼라 피연산자들로부터 벡터 결과를 발생하거나, 벡터 레지스터 파일로부터 판독된 피연산자들에 대한 벡터 처리 이외에 스칼라 레지스터 파일에 있는 스칼라 피연산자에 대한 추가적인 스칼라 연산을 행해도 된다. 이 때문에, 일부 명령이 혼합된 스칼라 및 벡터 명령이 될 수 있다. 처리회로 내에서 산술 또는 논리 연산을 기동

하는 벡터 산술 또는 논리 명령들 이외에, 디코더(6)는 벡터 레지스터(14)와 메모리 시스템(8) 사이에서 데이터를 전달하는 벡터 로드/스토어 명령들을 더 지원한다.

[0077] 도 2는 프레디케이트 레지스터들(16) 중에서 한 개에 있는 프레디케이트 정보에 의해 제어되는 벡터 연산의 일례를 나타낸 것이다. 본 실시예에서, 벡터 명령은, 처리회로(4)를 제어하여, 2개의 입력 벡터 Zm, Zn의 대응하는 데이터 성분들의 쌍들을 각각 가산하고 가산 결과를 목적지 벡터 Zd의 대응하는 성분에 각각 기록하는 다수의 독립된 가산 연산을 행하는 벡터 가산 명령이다. 프레디케이트 레지스터 P의 각각의 프레디케이트 필드는 목적지 레지스터의 대응하는 성분이 가산 결과로 갱신되는지 아닌지 제어한다. 예를 들어, 본 실시예에서, 벡터 레인들 1, 2 및 5가 (이들 레인에 대한 대응하는 프레디케이트 필드가 0이므로) 마스킹되므로, 목적지 레지스터 Zd의 대응하는 성분들이 가산 결과로 설정되지 않는다. 예를 들어, 마스킹된 레인들은 그들의 이전의 값들을 유지할 수도 있다. 이와 달리, 클리어링(clearing)/제로잉(zeroing) 형태의 프레디케이션은 목적지 레지스터의 마스킹된 성분들을 제로값 또는 다른 소정의 값으로 클리어하는 한편, 마스킹되지 않은 레인들은 이 레인에서의 가산 결과로 기록된다. 이 때문에, 일반적으로 프레디케이트 레지스터들(16)은 특정한 레인에서의 벡터 연산의 마스킹을 제어하는데 사용된다. 예를 들어, 조건부 벡터 명령은, 비교를 행하여 주어진 벡터의 각 레인에 있는 입력 성분들이 어떤 조건을 만족하는지 여부를 테스트하고, 대응하는 레인의 비교 결과에 근거하여 프레디케이트 레지스터에 프레디케이트 값들을 설정한 후, 그후 각각의 벡터 레인에서의 이 명령의 결과가 이전의 조건부 벡터 명령에 의해 행해진 비교를 조건으로 하도록 프레디케이트에 따라 이후의 벡터 명령들이 마스킹될 수 있다.

[0078] 도 2는 주어진 벡터에 대해 병렬로 행해지는 전체 가산을 나타내었지만, 이것이 필수적인 것은 아니다. 처리회로(4)는 다양한 다른 데이터 성분 사이즈들과 다양한 벡터 길이를 갖는 벡터들의 처리를 지원한다. 예를 들어, 256비트의 벡터 레지스터(14)가 예를 들어 32개의 8비트 데이터 성분들, 16개의 16비트 데이터 성분들, 8개의 32비트 데이터 성분들, 4개의 64비트 데이터 성분들, 또는 2개의 128비트 데이터 성분들로 분할되거나, 한 개의 256비트 성분으로 해석될 수도 있다. 또한, 예를 들어, 128, 256 또는 512비트의 다양한 사이즈를 갖는 벡터 레지스터들이 지원된다. 예를 들어, 벡터 레지스터 뱅크(14)는 다양하게 분할되어, 다양한 벡터 레지스터 사이즈들 제공할 수도 있다(예를 들어, 고정된 사이즈의 다수의 물리 레지스터들이 결합되고 한 개의 벡터 명령에 의해 동작하여 더 큰 레지스터 파일을 표시할 수도 있다). 이 때문에, 도 1에 나타낸 것과 같이, 레지스터들(10)은, 벡터 연산을 위해 사용될 데이터 성분 사이즈(벡터의 한 개의 성분의 비트 수)를 지정하는 성분 사이즈 파라미터(18)와, 벡터 레지스터의 길이(모든 성분들을 포함하는, 벡터 내부의 전체 비트수)를 지정하는 벡터 길이 파라미터(20) 등의, 벡터 처리를 제어하기 위한 제어정보를 더 포함한다. 일부 구현예에서, 이들 값은 고정되어 모든 명령이 대해 이들이 항상 동일하도록 할 수 있다. 서로 다른 성분 사이즈들 또는 벡터 길이들을 갖는 다양한 플랫폼 상에서 실행하기 위해 기록된 코드가 현재 이 코드를 실행하고 있는 시스템 상에서 구현된 특정한 성분 사이즈 또는 벡터 길이를 판독할 수 있도록 성분 사이즈(18) 및 벡터 길이(20)를 지정하는 레지스터들을 설치하는 것이 여전히 유용할 수 있다. 그러나, 다른 실시예에서는, 명령들이 주어진 연산에 대해 어떤 성분 사이즈 또는 벡터 길이를 사용할 것인지 지정하도록 성분 사이즈 및/또는 벡터 길이가 프로그래밍 가능하다. 이와 달리, 성분 사이즈 및/또는 벡터 길이는, 제어 레지스터 내부의 파라미터가 아니라, 벡터 명령 인코딩에 의해 지정될 수도 있다.

[0079] 따라서, 성분 사이즈 또는 벡터 길이가 변동할 수 있다. 선택된 특정한 사이즈에 따라, 벡터 처리회로가 전체 벡터를 병렬로 처리하는데 충분한 처리 하드웨어를 항상 갖지 않을 수도 있다. 처리 로직이 주어진 연산에 대해 사용되고 있는 벡터 길이보다 좁으면, 벡터가 복수의 사이클로, 더 좁은 처리 로직을 통한 별개의 패스(pass)들로 처리될 수도 있다. 이 때문에, 벡터 명령이 한개의 명령에 응답하여 처리회로(4)를 기동하여 복수의 레인들의 처리로 연산을 행하지만, 이것은 이들 모든 레인들이 병렬로 처리되어야 한다는 것을 반드시 의미하는 것은 아니다. 극단적인 예로, 일부 벡터 구현예는 한 개의 레인에 대응하는 처리 로직만을 제공한 후, 모든 벡터 레인들을 순차적으로 처리할 수도 있다. 또 다른 극단적인 예로, 더 높은 성능의 구현예는 다수의 병렬 실행 유닛들을 사용하여 모든 벡터 레인들을 병렬로 처리할 수도 있다. 다른 구현예는 다수의 레인들을 병렬로 처리하지만, 전체적으로 복수의 순차적인 청크들로 벡터를 처리한다.

[0080] 이때, 성분 사이즈 및 벡터 길이 표시들(18, 20)은 레지스터들(10)에 기억될 수 있는 제어 정보의 일부예에 지나지 않는다는 것은 자명하다. 다른 실시예는 현재의 실행 지점을 표시하는 명령의 어드레스를 표시하기 위한 프로그램 카운터 레지스터와, 예외를 처리시에 상태를 저장하거나 복원하기 위한 스택 데이터 구조를 갖는 메모리(8) 내부의 위치의 어드레스를 표시하는 스택 포인터 레지스터와, 함수의 실행 후에 처리가 분기되는 함수 복귀 어드레스를 기억하는 링크 레지스터를 구비해도 된다.

[0081] 도 3 및 도 4는 일 실시예에 따른 벡터 자리올림이 있는 가산 명령의 가산 및 감산 변종을 나타낸 것이다. 가산 및 감산 변종들 모두에 대해, 명령의 추가적인 제1 및 제2 변종을 나타내면 다음과 같다:

[0082] ADCLB Zda,Zn,Zm//자리올림 롱(long)을 갖는 가산(하단)-제1 변종, 가산 변종

[0083] ADCLT Zda,Zn,Zm//자리올림 롱을 갖는 가산(상단)-제2 변종, 가산 변종

[0084] SBCLB Zda,Zn,Zm//자리올림 롱을 갖는 감산(하단)-제1 변종, 감산 변종

[0085] SBCLT Zda,Zn,Zm//자리올림 롱을 갖는 감산(상단)-제2 변종, 감산 변종

[0086] 각각의 변종에 대해, Zda는 목적지 벡터 레지스터(이것은 제1 소스 데이터 벡터 레지스터로서의 역할도 하고, 누산기 벡터 레지스터로도 부를 수 있다)를 나타내고, Zn은 제2 소스 데이터 벡터 레지스터를 나타내고, Zm은 자리올림 소스 벡터 데이터 레지스터를 나타낸다. 이들 명령은, 자리올림 또는 자리빌림 입력 정보가 누산기 벡터 레지스터 및 자리올림 소스 벡터 레지스터 내부의 실제 데이터 값들과 인터리브되도록 가산 또는 감산을 행함으로써 큰 정수들의 처리를 가속화하는데 사용될 수 있다.

[0087] 이 때문에, 자리올림 정보는 소스 및 목적지 벡터 레지스터들의 필수적인 부분이다. 소스 및 목적지 벡터들 내부의 "자리올림 수"(또는 "자리빌림 수") 정보의 위치를 "자리올림 수"를 입력으로 이용하거나 "자리올림 수"를 발생하는 수학적 연산을 행하는 레인들에 인접한 레인들에 제한하는 것이 유용하다. 레지스터들 Zda, Zn, Zm이 성분들의 쌍들로 분할되는 도 3 및 도 4를 확인하기 바란다. 각 쌍은 짝수 성분에 대응하는 "하단" 성분(제1 성분)과 홀수 성분에 대응하는 "상단" 성분(제2 성분)을 포함한다. 레지스터들 Zda, Zm에 대해 자리올림 정보는 각 쌍의 상단 성분들 내부에 배치되는 한편, 하단 성분들은 대응하는 데이터를 제공한다. 다른 실시예는 각 쌍의 데이터/자리올림 성분들을 이와 다르게 배치할 수도 잇지만(예를 들어, 모든 데이터 성분들을 레지스터의 절반부에, 대응하는 자리올림 성분들을 나머지 절반부에 배치), Zda에 있는 인접한 쌍의 성분들을 발생하기 위한 입력들이 Zn, Zm의 대응하는 쌍의 성분들로부터 선택되어, 벡터의 2개 이상의 성분들에 걸치는 길이가 거긴 성분을 교차하는 경로들의 필요성을 없애, 처리회로의 마이크로 아키텍처 구현을 더 효율적으로 만들 수 있기 때문에, 데이터와 자리올림 성분들을 인터리브하는 것이 더 효율적일 수 있다. 이 때문에, 이들 연산은 각각의 벡터를 벡터 성분들보다 2개의 사이즈를 갖는 "과립(granule)들" 또는 세그먼트들로 효율적으로 분할한다.

[0088] 도 3은 ADCLB 및 ADCLT 명령들의 연산을 나타낸 것이다. 명료하게 하기 위해, 단지 한 개의 "과립"(인접한 상단 및 하단 성분들의 쌍)의 상세한 동작을 나타낸다. 이때, 대응하는 동작이 각각의 다른 과립(성분들의 쌍)에 대해 행해진다는 것은 자명하다. ADCLB 명령은, 제2 소스 Zm에 있는 각각의 과립 중 상단 절반부(홀수 성분)의 최하위 비트를 자리올림 입력으로 사용하여, 한 개의 소스 Zn의 짝수 성분들 B(또는 ADCLT 변종의 경우에는 홀수 성분들 T)를 목적지 벡터 레지스터의 짝수 성분들에 가산한다. 그 결과 얻어지는 합은 목적지 벡터 내부의 각각의 과립의 하단 절반부에 기록되는 한편, 생성된 자리올림 정보는 목적지 내부의 각각의 과립의 상단 절반부의 최하위 비트에 기록된다. Zda 내의 각각의 과립의 상단 절반부의 나머지 비트들은 사용되지 않은 채로 유지된다. 이 결과는 자리올림 소스 레지스터 Zm 내부의 각각의 과립의 하단(짝수) 성분의 비트들에 무관하다. 또한, ADCLB에 대해 결과는 데이터 소스 레지스터 Zn의 선택되지 않은 (상단/홀수) 성분들에 무관하고, ADCLT에 대해, 결과는 Zn의 선택되지 않은(하단/짝수) 성분들에 무관하다.

[0089] 이와 같이 자리올림 정보와 입력 또는 결과 데이터를 인터리브함으로써, 자리올림 체인이 정규 데이터 흐름의 일부인 것을 보장할 수 있다. 이것은 다수의 병렬도를 가능하게 한다. 첫째, 벡터 레지스터들의 입상도에서 의존성에 해결되기 때문에 한 개보다 많은 수의 동시에 보류중인 자리올림 체인이 존재할 수 있다. 둘째, 명령의 B/T 변종들이 독립적으로(일부 마이크로 아키텍처에서는, B/T 변종들이 한 개의 마이크로 연산으로 융합될 수 있거나, 또는 2개의 변종들이 파이프라인에 의해 병렬로 처리될 수 있는 경우에는 동시에) 산출될 수 있다. ADCL 명령들은 보통 이전의 승산 단계들에 의해 발생된 부분 곱들을 소모하기 때문에, 이들 명령은 승산 및 가산 단계가 "능률화"되도록 프로그램 시퀀스를 설계하는 것을 가능하게 하며, 자리올림 수들이 기본적으로 누산기에 구속되기 때문에 자리올림 수들을 처리하기 위한 여분의 연산이 존재하지 않는다. 이것은 자리올림 정보를 정렬시키는 오버헤드를 줄인다.

[0090] 도 3 및 도 4에서는, 각 쌍의 성분들 중에서 "제1 성분"이 제2 성분보다 낮은 성분 위치에 있으므로, 결과가 각 쌍의 하단 성분에 기록되지만, 다른 구현예에서는 제1 및 제2 성분들이 레지스터들 내에서 뒤바뀌어, 결과가 각 쌍의 상단 성분을 갱신하고 자리올림 수들이 하단 성분에 기록될 수도 있다. 그러나, 마이크로 아키텍처에서는 자리올림 수가 각 쌍의 최상위 성분에 설치되도록 하는 것이 더 효율적인데, 이것이 자리올림 수가 가산의 데이터 결과의 최상위 비트보다 1 비트 위치가 더 상위에 있다는 것을 표시할 때 비트들이 가산회로에

의해 출력되는 순서와 일치하기 때문이다.

[0091] 도 4는 벡터 자리올림이 있는 가산 명령의 감산 변종의 제1/제2 변종들의 대응하는 처리를 나타낸 것이다. 핵심적인 연산이 "감산"(즉 Zn으로부터 선택된 성분이 가산하기 전에 2의 보수인 것을 제외하고는 동일한 가산)이고 "자리올림" 정보 대신에 "자리빌림" 정보가 입력으로서 사용되고 출력에서 발생된다(자리빌림은 -1의 자리올림을 나타낸다)는 것을 제외하고는,SBCLB 및 SBCLT 명령들의 연산 또는 데이터 흐름이 각각 ADCLB 및 ADCLT와 유사하다. Zn으로부터 입력된 데이터의 2의 보수화 및 자리빌림 반전은 단순히 Zn 데이터 소스 입력과 가산기에 대한 borrow_in 값을 반전시킴으로써 구현될 수 있다. borrow_in 값의 반전이 이것을 효율적으로 제공하므로, Zn 데이터 소스 입력에 대한 1의 별도의 가산이 필요하지 않다. 1의 자리빌림 값에 대해, 0의 반전된 borrow_in은, 자리빌림 수에 대한 1의 감산이 2의 보수에 대한 1의 가산을 상쇄한다는 것을 반영한다. 이 때문에, 데이터 입력과 borrow_in 값이 반전되면, 가산 변종에 대한 것과 동일하게 가산이 행해질 수 있다.

[0092] 도 4a는 벡터 자리올림이 있는 가산 명령의 프레디케이트된 변종의 일례를 나타낸 것이다. 본 실시예에서, 프레디케이션이 도 3에 도시된 ADCLB 변종에 적용되지만, 이와 유사한 ADCLT, SBCLB 및 SBCLT 변종들의 프레디케이트된 버전들도 제공될 수 있다. 목적지 레지스터 Zda의 인접한 상단/하단 성분들의 각 쌍에 대해 행해진 핵심적인 자리올림이 있는 가산 연산은 도 3에서와 같다. 그러나, 명령이 예를 들어 프레디케이트 레지스터들(16) 중에서 한 개에 기억된 프레디케이트 값 P와 더 관련된다. 프레디케이트 값은 도 2와 유사하게 다수의 프레디케이트 필드들을 포함하지만, 도 3 및 도 4에 나타낸 종류의 벡터 자리올림이 있는 가산 명령에 대해서는, 각각의 성분에 대해서가 아니라, 인접한 성분들의 쌍들의 입상도에 프레디케이션이 적용되므로, 프레디케이트 필드들 중 절반만 유효하다. 본 실시예에서는, 유효한 프레디케이트 필드들은 각 쌍의 하단(짝수) 성분들에 대응하는 필드들이다(따라서 결과는 각 쌍의 상단 성분들에 대응하는 프레디케이트 필드들에 무관하며, 이들 필드들의 값들이 결과에 영향을 미치지 않는다는 것을 표시하기 위해 이들 필드들은 X로 표기한다). 그러나, 다른 구현예는 각 쌍의 성분들에 대한 상단(홀수) 프레디케이트 필드들을 유효한 프레디케이트 필드로 사용하도록 선택할 수도 있다.

[0093] 대응하는 프레디케이트 표시가 1인 목적지 벡터의 성분들의 쌍들에 대해, 자리올림이 있는 가산 연산이 도 3에 도시된 것과 동일하게 행해져, 쌍의 제1 성분을 가산으로부터 발생된 데이터 D로 갱신하고, 쌍의 제2 성분을 가산의 자리올림 출력 C로 갱신한다. 그러나, (본 실시예에서 성분 4 및 5에 대해 나타낸 것과 같이) 대응하는 프레디케이트 표시가 0인 성분들의 쌍에 대해서는, 목적지 벡터 Zda'의 이들 성분들의 갱신이 금지된다. 그 대신에, 목적지 벡터 Zga'의 이들 성분은 그들의 이전 값을 유지하거나(병합 프레디케이션) 제로값이나 다른 소정 값으로 클리어된다(제로잉 프레디케이션). 이 때문에, 프레디케이트가 각각의 개별 성분이 아니라 각각의 과립(성분들의 쌍)에 적용된다. 이때, 도 4a는 프레디케이트 레지스터(16)의 한가지 예시적인 포맷을 나타낸 것이지만, 목적지 레지스터 Zda'의 각각의 쌍의 성분들이 마스킹되어야 하는지 여부를 표시하는 프레디케이트 표시를 제공하는 다른 기술이 사용될 수도 있다는 것이 자명하다.

[0094] 도 5는 목적지 (누산기) 벡터 레지스터 Zda, 제2 데이터 소스 벡터 레지스터 Zn 및 자리올림 소스 벡터 레지스터 Zm(본 실시예에서 목적지 벡터 레지스터 Zda는 제1 데이터 소스 벡터 레지스터로서의 역할도 한다)를 지정하는 벡터 가산 자리올림 명령의 처리를 나타낸 흐름도이다. 스텝 50에서, 명령 디코더는 이와 같은 벡터 자리올림이 있는 가산 명령이 검출되었는지 여부를 판정하고, 검출되지 않은 경우에는 이 명령에 의해 표시된 연산에 따라 현재의 명령이 처리된다. 벡터 자리올림이 있는 가산 명령이 마주치면, 스텝 52에서 명령 디코더는 이 명령이 "상단"(제2) 변종인지 또는 "하단"(제1) 변종인지 판정한다. 명령이 상단 변종인 경우, 스텝 54에서 제2 데이터 소스 벡터 레지스터 Zn 내부의 각각의 쌍의 성분들로부터 선택된 성분이 이 쌍의 상단 성분, 즉 홀수 성분(제2 성분)이 되는 반면에, 명령이 하단(제1) 변종인 경우에는, 스텝 56에서, 각 쌍의 하단(제1 또는 짝수) 성분이 제2 데이터 소스 벡터 레지스터 Zn으로부터 선택된다.

[0095] 스텝 58에서, 명령 디코더는 이 명령이 가산 변종인지 또는 감산 변종인지 판정한다. 명령이 가산 변종이면, 스텝 60에서, 목적지 벡터 레지스터 Zda 내부의 각 쌍의 성분들에 대해, 이 쌍의 하단(제1) 성분의 새로운 값 Zda(B)'이 (i) 목적지 벡터 레지스터 내의 이 쌍의 성분들의 하단 성분 Zda(B)의 이전 값, (ii) 스텝 64 또는 56에서 선택된 것과 같이 제2 데이터 소스 레지스터 Zn 내부의 대응하는 쌍의 성분들의 선택된 성분(상단 또는 하단), 및 (iii) 자리올림 소스 벡터 레지스터 Zm 내부의 대응하는 쌍의 성분들의 상단 성분 Zm(T)로부터 추출되는 자리올림 입력의 가산 결과로 설정된다. 또한, 목적지 벡터 레지스터 Zda 내부의 대응하는 쌍의 성분들의 상단 성분 Zda(T)'의 최하위 비트가 가산의 자리올림 출력으로 설정된다.

[0096] 명령이 감산 명령인 경우, 스텝 62에서, 목적지 벡터 레지스터 내부의 각 쌍의 성분들의 하단 성분

Zda(B)"가 이 하단 성분의 이전값 - 스텝 54 또는 56에서 선택되었던 데이터 소스 벡터 Zn 내부의 대응하는 쌍의 성분들 중에서 한 개 - 자리올림 소스 벡터 레지스터 소스 Zm 내부의 대응하는 쌍의 성분들의 상단 성분 Zm(T)로부터 추출된 자리올림 값으로 표시되는 자리빌림 값의 결과로 설정된다. 또한, 또한, 목적지 벡터 레지스터의 대응하는 쌍의 성분들의 상단 성분 Zda(T)'의 최하위 비트는 감산으로부터의 자리빌림 출력으로 설정된다(즉, Zda(T)의 lsb는 감산을 행하고 있는 가산회로의 자리올림 출력 상에 출력된 값으로 설정된다).

[0097] 도 3 내지 도 5의 실시예는 동일한 레지스터 Zda가 목적지 레지스터 및 제1 데이터 소스 레지스터 양쪽으로서의 역할을 하는 파괴적인 명령 포맷을 나타내지만, 목적지 레지스터와 별개로, 또 다른 벡터 레지스터를 제1 데이터 소스 레지스터로서 지정하는 비파괴적인 명령 포맷이 제공되어, 목적지 레지스터에 결과를 기록할 때 제1 소스 벡터의 오버라이팅을 피할 수도 있다.

[0098] 이하, 이들 명령에 대한 예시적인 유스 케이스에 대해 설명한다.

[0099] 다수의 중요한 암호화 작업량의 가장 시간이 많이 걸리는 부분은 큰 정수값에 대해 수학적 연산을 행하는 루틴을 포함한다. 암호화 시나리오 이외에, 큰 수를 포함하는 수학적 연산은 금융 소프트웨어 및 일부 과학 응용분야에서 사용되는 GNU Multiprecision Arithmetic(GMP) 등의 라이브러리를 뒷받침한다. 이와 같은 수는 보통 재현성에 관한 문제 또는 수학적 엄밀함으로 인해 부동소수점 연산이 (그것의 범위에도 불구하고) 적절하지 않은 경우에 사용된다. RSA 알고리즘은 가장 얼리 사용되는 공개 키 암호화 알고리즘이다. 그것의 보안은 큰 수를 인수분해하는 인지된 난이도에 의존한다. 메시지를 암호화 또는 복호화하기 위해서는, 프로세서가 모듈러 지수로 알려진 기술로 큰 수를 가능한한 빨리 승산해야 하는 요구가 존재한다. 예를 들어, RSA2048에서는, 2048비트 수의 다수의 연속된 승산이 존재하여, 4096 비트 곱을 발생하고 그후 이 곱이 줄어든다.

[0100] 정수들을 승산하는 가장 간단한 방법(이것은 보통 "교과서적인" 방법으로 불린다)은, n -숫자 소스에 대해, 한 개의 소스의 모든 숫자가 나머지 소스의 모든 숫자와 승산될 필요가 있기 때문에, $O(n^2)$ 단계를 필요로 한다. 또한, 다음 예에서 볼 수 있는 것과 같이 "가산" 및 "데이터 재편성"(자리옮김) 연산이 필요하다.

[0101] 111*543 = 333 + (444 << 1 숫자) + (555 << 2 숫자) = 60273

[0102] 그러나, 카라츠바 알고리즘은 분할 정복 기반의 접근방식을 이용하여 2개의 n 숫자 수치의 승산을 최대 $O(n^{\log 23})$ 단일 숫자 승산으로 줄일 수 있다. 이와 같은 방법은, 승산과 "가산", "감산" 등 사이에서 타협안을 채용하여, "n"이 클 때 주목할만한 성능상의 이점을 제공한다. 사이즈 "n"을 갖는 입력 피연산자들을 사용한 각각의 승산은 사이즈 "n/2"를 갖는 입력 피연산자들을 사용한 다수의 더 작은 승산들과 보조 연산들 - "가산", "감산" 등으로 나뉜다. 공식적으로, 도 6의 상단에 있는 수식에서 나타낸 것과 같이, 2개의 n 비트 정수들 A 및 B를 승산하여 2n 비트의 결과 C를 발생하기 위한 수식을 작성할 수 있다. 이 예에서는, 부분 곱이 다루기 불편한 사이즈를 갖지 않도록 보장하기 위해 카라츠바 알고리즘의 감산형 변종을 사용한다. 이 때문에, 도 6에 나타낸 것과 같이, 2개의 2048 비트 정수들의 승산이 1024 비트 정수들의 3개의 서브 승산으로 분해될 수 있다. 각각의 1024 비트의 승산은 그 자체가 동일하게 3개의 512 비트 승산들로 분해될 수 있다. 결국, 전체적인 결과를 다시 만들기 위해 추가적인 가산, 감산 또는 자리옮김이 필요하기 때문에 승산을 더 분해하는 것이 더 큰 전체 오버헤드를 일으키는 지점에 도달한다.

[0103] 서브 승산들이 독립적이므로, 적절한 데이터 레이아웃과 조정을 사용하면, 도 6에 도시된 것과 같이 카라츠바 알고리즘을 벡터화할 수 있다. 예를 들어, 카라츠바 트리의 리프 노드가 256 비트 서브 승산들을 제공하는 경우, 카라츠바 트리의 주어진 분기의 3개의 256 비트 서브 승산들 각각은 도 6의 상측 예에서 나타낸 것과 같이 벡터의 별개의 레인 A, B, C에 할당될 수도 있다. 또한, 도 6의 하측 예에서 나타낸 것과 같이, 더 높은 레벨의 카라츠바 트리에 있는 다양한 512x512 비트 승산들에 대한 6개의 256비트 서브 승산들은 8 성분 벡터의 6 레인들에 할당될 수 있다(3개의 레인들 A, B, C의 각각의 세트는 대응하는 512 비트 노드의 3개의 256x256 비트 승산들을 나타낸다).

[0104] 예를 들어, 64비트의 성분 사이즈를 선택하면, 2개의 256 비트 정수들 A 및 B를 다음과 같이 승산할 수 있다: $A \times B = \{a_3, a_2, a_1, a_0\} \times \{b_3, b_2, b_1, b_0\}$. 이와 같은 승산에 속하는 부분 곱들을 도 7에 나타낸다. 이때, 각각의 부분 곱이 적절한 열에서 누산되도록 "재조정"될 필요가 있기 때문에, 이들 곱은 우측으로부터 좌측으로 엇갈려 배치된다.

[0105] 도 8은 부분 곱 $a_1 \times \{b_3, b_2, b_1, b_0\}$을 계산하기 위해 도 7의 두 번째 "가지"의 연산을 행하기 위해 필요한 명령들의 시퀀스의 한가지 가능한 예를 나타낸 코드 시퀀스이다. 이것은 다양한 부분 곱들의 하위 및 상위

64 비트 부분들을 생성하기 위해 승산들로 개시한다. 이때, 256 비트x256 비트 승산의 각각의 인스턴스는 벡터 처리시에 한 개의 레인을 차지하고, 각각의 256 비트 피연산자의 64 비트 성분들은 각각의 벡터를 가로지르는 것이 아니라 (별개의 명령들에 의해 작용한 별개의 레지스터들에) 수직으로 배치된다. 이것은, 계산 $a_1$x{$b_3$, $b_2$, $b_1$, $b_0$}에 대한 부분 곱들을 생성하기 위해서는, (다양한 소스 피연산들로부터 $a_1$을 포함하는) 레지스터 z1이 (다수의 다양한 소스 피연산자들로부터 $b_0$을 제공하는) z4, z5($b_1$), z6($b_2$) 및 z7($b_3$)의 각각과 승산할 필요가 있다(도 8의 라인 1 내지 8).

[0106] 도 8에서, adc1b 및 adc1t 명령을 사용하여 부분 곱들을 각각의 레인에 대한 누산기 zacc1LB, zacc1LT, zacc0HB 등에 추가한다. adc1b 및 adc1t 명령들의 대응하는 쌍들을 실행하여 각각의 승산으로부터의 부분 곱들의 짝수 및 홀수 성분들에 작용한다. 이들 명령은 4방향 병렬화를 노출하도록 사용되고, B & T 변종들은 독립적이며 그 결과 유리하게 스케줄에 넣을 수 있다. 더구나, 동시에 처리될 수 있는 2개의 자리올림 체인이 존재한다. 한 개의 자리올림 체인은 라인 10에서 시작하고 나머지는 라인 12에서 시작한다. 그후, 라인 14, 18 및 26의 명령들 거쳐 "B" 변종에 대한 첫 번째 자리올림 체인을 추적할 수 있다. "T" 변종에 대해서도 유사한 체인이 존재한다. 라인 32 및 33에 있는 명령을 사용하여, 최소 부분 곱($a_1$x$b_3$(H))를 동시에 누산기 내부에 가산하면서 이전의 독립적인 자리올림 체인들을 병합한다.

[0107] 도 9와 도 10a 내지 도 10l은 벡터 자리올림이 있는 가산 명령의 또 다른 예시적인 사용을 나타낸 것이다. 도 9는 큰 정수를 승산하기 위한 카라츠바 승산 트리의 일부인 더 큰 승산의 한 개의 가지를 나타내는 승산 {$a_3$, $a_2$, $a_1$, $a_0$}x$b_i$의 부분 곱들의 가산을 행하기 위한 ADCLB 명령들의 예시적인 명령을 나타낸 것이다. 이때, 도 9에서는, 부분 곱들이 이미 생성되어 벡터 레지스터들 $P_{0i}$lo, $P_{1i}$lo , …, $P_{3i}$hi에 기억된 것으로 가정한다(이때, $P_{ji}$lo 및 $P_{ji}$hi는 곱 $a_j$ 및 $b_i$의 하부 절반부 및 상부 절반부를 나타낸다). AccB[i] 내지 AccB[i+4]는 열 0 내지 4에 각각 나타낸 곱 {$a_3$, $a_2$, $a_1$, $a_0$}x$b_i$의 5개 부분들을 기억하기 위한 5개의 누산기 레지스터들을 나타낸다. 벡터 레인들의 각각의 쌍은 다른 벡터 레인들과 비교하여 서로 다른 입력값에 적용되는 완전히 독립적인 계산을 나타내므로, 도 7에 나타낸 종류의 주어진 계산에 필요한 부분 곱들의 각각의 가산은 도 9의 명령들의 시퀀스에 의해 한 개의 쌍의 레인들 내에서 행해진다(예를 들어, 각 쌍의 레인들에서의 계산은 카라츠바 트리의 서로 다른 노드에 대응한다). 간략을 위해, 도 10a 내지 도 10l은 벡터들의 짝수 레인들에 작용하는 ADCLB 명령들만 도시하고 있다 - 도시되지 않은 대응하는 ADCLT 명령은 실행되어 홀수 레인에 작용한다. 간략을 위해, 첫 번째 체인의 자리올림 수를 별표가 없게 표시하고(예를 들어, $C_{AccBi}$) 두 번째 체인의 자리올림 수를 별표로 표시(예를 들어, $C_{AccBi}$*)함으로써, 2개의 독립된 자리올림 체인을 구별한다.

[0108] · 도 10a: AccB[i]는 열 0의 누산을 나타낸다. AccB[i]의 각 쌍의 성분들에 대해, 이 쌍의 하부(짝수) 성분은, AccB[i]의 하부 성분의 이전 값, $P_{0i}$lo의 대응하는 성분 쌍의 하단 성분과, 자리올림 입력으로서의 0(이것은 결과의 최하위 부분이므로 하부 가산으로부터의 자리올림이 필요하지 않다)의 합으로 설정된다. 추가적인 자리올림이 결과의 최하위 부분에 주입될 수 없으므로, 가산 결과는 이미 결과의 이 부분에 대한 최종 결과값 Res[i]를 나타낸다. 가산으로부터의 자리올림 출력 CaccBi는 AccB[i]의 쌍의 상단 성분의 최하위 비트에 기록되고, 열 1의 최하위 비트와 동등한 가중치를 갖는 자리올림 비트를 나타낸다.

[0109] · 도 10b: AccB[i+1]은 열 1의 누산을 나타낸다. AccB[i+1]에 있는 각 쌍의 성분들에 대해, 이 쌍의 하부(짝수) 성분은, AccB[i+1]의 하부 성분의 이전 값, $P_{1i}$lo의 대응하는 성분 쌍의 하단 성분, 및 자리올림 입력으로서의 $C_{accBi}$의 합으로 설정된다. 즉, 이전의 ADCLB 명령의 목적지 레지스터 AccB[i+1]가 다음 ADCLB 명령의 자리올림 소스 벡터 레지스터로서 지정되므로, 행 0 내의 이전 가산에서 생성된 자리올림 수 $C_{accBi}$가 행 1의 가산에 대한 자리올림 입력으로서 입력된다. 행 1 누산으로부터의 자리올림 출력 $C_{accBi+1}$은 AccB[i+1]의 각 쌍의 상단 성분의 최하위 비트에 기억되어, 행 2의 누산의 최하위 비트에 적용될 자리올림 수를 표시한다.

[0110] · 도 10c: AccB[i+2]는 행 2의 누산을 나타낸다. 마찬가지로, 이전의 명령에 대한 결과 벡터 AccB[i+1]이 다음 명령에 대한 자리올림 소스 벡터 레지스터가 되므로, 각 쌍의 성분들에 대해, 행 1의 이전의 가산에 의해 생성된 자리올림 수 $C_{accBi+1}$이 AccB[i+1]의 대응하는 쌍의 상단 성분으로부터 추출되어 AccB[i+2]의 쌍의 성분의 하단에 기록되는 AccB[i+2]와 $P_{2i}$lo의 누산값에 가산되고, 결과적으로 얻어지는 자리올림 출력 $C_{accBi+2}$가 AccB[i+2] 내부의 각 쌍의 성분들의 상단 성분에 기억된다.

[0111] · 도 10d: 이 명령은 도 10a-도 10c를 거쳐 얻어지는 자리올림들의 체인에 무관한, 제2 자리올림 체인의 개시를 표시한다. 이 명령은 도 10b에서 얻어진 AccB[i+1]에 부분 곱 $P_{oi}$hi의 행 1에의 가산을 나타낸다. 이 때, AccB[i+1]의 홀수 성분들의 자리올림 값은 도 10c에서 이미 소모되었으므로, AccB[i+11의 짝수 성분들과 $P_{oi}$hi의 짝수 성분들의 각각의 가산에서 발생된 새로운 자리올림 값들 $C_{accBi+1}$에 의해 홀수 성분들에 오버라이트 될 수 있다.

[0112] · 도 10e: 이 명령은 제1 자리올림 체인의 일부이며, (열 3의 가산을 나타내는) AccB[i+3] 및 $P_{3i}$lo의 각 쌍의 성분들의 하부 성분들의 가산에 대한 자리올림 입력으로서 (AccB[i+2]의 각 쌍의 성분들의 상부 성분에 기억된) 도 10c로부터의 자리올림 출력 $C_{accBi+2}$를 사용한다. 그 결과 얻어지는 자리올림 출력 $C_{accBi+2}$는 AccB[i+3]의 각 쌍의 상부 성분에 기억된다.

[0113] · 도 10f: 제2 자리올림 체인 내부의 이와 같은 명령은 열 2의 AccB[i+2] 및 $P_{1i}$hi의 가산에 대한 자리올림 입력으로서 (열 2의 최하위 비트와 동등한 가중치를 갖는) 도 10d로부터의 자리올림 출력 $C_{accBi+1}$*를 사용한다.

[0114] · 도 10g: 제1 자리올림 체인 내부의 이와 같은 명령은 AccB[i+4]의 짝수 레일들에서 벡터 가산을 행하고, 각각의 가산에 대한 자리올림 입력은 (도 10e에서 발생되는) AccB[i+3]의 홀수 레인들에서 발생된다. 이 명령이 AccB[i+4]의 홀수 성분들의 최하위 비트에 각각의 가산의 자리올림 출력을 여전히 기록하지만, 실제로, 이와 같은 누산이 결과의 최상위 부분을 표시하므로, 이들 자리올림 수들은 다음의 명령에 대해 필요하지 않다.

[0115] · 도 10h: 제2 자리올림 체인 내부의 이와 같은 명령은 AccB[i+3]과 $P_{2i}$hi의 짝수 레인들에서 벡터 가산을 행하고, 각각의 가산에 대한 자리올림 입력 $C_{accBi+2}$*는 (도 10f에서 발생되는) AccB[i+2]의 홀수 레인들로부터 발생된다.

[0116] · 도 10i: 마지막으로, 제2 자리올림 체인 내부의 이와 같은 명령은 제로값이 된 데이터 소스 레지스터와 AccB[i+4]의 짝수 레인들의 벡터 가산을 행한다(도 10g에서 명령에 의해 이미 가산된 최상위 행에 가산할 단지 한 개의 부분 곱이 존재하기 때문에, 두 번째 데이터 입력은 제로이다). 각각의 가산에 대한 자리올림 입력 $C_{accBi+2}$*는 (도 10g에서 발생된) AccB[i+3]의 홀수 레인들로부터 발생된다. 각각의 가산의 결과는 누산기 레지스터 AccB[i+4]의 짝수 레인들에 기록된다. 마찬가지로, 짝수 레인들이 명령의 정상 거동의 일부로서 각각의 가산의 자리올림 입력으로 기록되지만, 이들 자리올림 수들은 추가적인 명령들에 대해서 필요하지 않다.

[0117] 이 때문에, 본 실시예는, ADCLB 명령들이 승산의 부분 곱들의 가산에 대한 자리올림 정보가 효율적으로 데이터 결과 그 자체와 함께 이동하도록 함으로써, 2의 거듭제곱의 비트 수의 성분들을 갖는 입력 벡터들(예를 들어, $P_{01}$lo 내지 $P_{3i}$hi)에 적용된 이와 같은 승산의 효율적인 계산을 가능하게 하여, 가산을 행하기 전에 벡터 승산 명령의 결과의 언패킹이 필요하지 않게 되는 방법을 나타낸다. 이것은, 예를 들어, 카라츠바 알고리즘을 사용하여, 긴 정수의 승산을 포함하는 계산에 대한 성능을 향상시킨다.

[0118] 도 11은 자리올림이 있는 가산 명령을 구현하는 또 다른 방법을 나타낸 것이다. 벡터 레지스터의 교번하는 성분들로부터 자리올림 입력을 취하는 것 대신에, 프레디케이트 레지스터 파일(16)로부터의 입력 프레디케이트 레지스터 Pm의 대응하는 프레디케이트 필드에서 자리올림 입력이 얻어진다. 마찬가지로, 자리올림 출력은 목적지 프레디케이트 레지스터 Pd2의 대응하는 프레디케이트 필드에 기록된다. 이와 같은 접근방법에 따르면, 하나 걸러서의 성분에 자리올림 수를 표시할 필요가 없으므로, 벡터가 실제 데이터 값들로 완전히 점유될 수 있다. 이 때문에, 8 성분 벡터에 대해서, 이것은 동일한 명령에 응답하여 8번의 별개의 누산이 행해질 수 있도록 할 것이다. 결과 벡터 Zda1은 이 성분의 이전 값과 데이터 소스 벡터 Zn의 대응하는 성분의 합 또는 차로 설정된 각각의 성분을 갖는다. 가산에 대한 자리올림 입력은 입력 프레디케이트 레지스터 Pm의 대응하는 프레디케이트 필드로부터 추출되고, 가산으로부터의 자리올림 출력은 출력 프레디케이트 레지스터 Pd2의 대응하는 프레디케이트 필드에 기록된다. 마찬가지로, 도 3 및 도 4에 도시된 것과 유사한 가산 및 감산 변종이 제공될 수 있으므로, 자리올림 입력/출력은 자리올림 또는 자리빌림 정보를 표시할 수 있다(자리빌림은 다음의 최상위 비트로부터 공제가 행해지는 것을 나타내는 반면에, 자리올림은 다음의 최상위 숫자에 대해 가산이 행해지는 것을 나타낸다 - 즉, 자리빌림은 -1의 자리올림이다).

[0119] 도 11에 도시된 접근방법은 더 꽉 채워진 벡터 레지스터 파일을 발생하여 명령들의 수의 절반으로 누산이 행해질 수 있게 하지만(예를 들어, 소스 데이터 벡터의 모든 성분들이 한 개의 명령으로 처리될 수 있기 때

문에 별개의ADCLB 및 ADCLT 명령이 필요하지 않다), 다수의 벡터 마이크로 아키텍처에 대해서는 드물지만, 벡터 레지스터 파일에 기록하는 것 이외에 프레디케이트 레지스터 파일에 대한 추가적인 기록이 명령에 의해 필요하므로, 도 복잡한 마이크로 아키텍처의 수정을 필요로 한다. 이 때문에, 도 3 및 도 4의 접근방법을 취할 것인지 도 11의 접근방법을 취할 것인지는 소프트웨어의 효율과 하드웨어 마이크로아키텍처의 효율 사이의 균형에 의존한다.

[0120] 도 12는 자리올림이 있는 가산 명령의 프레디케이트 형태의 처리를 나타낸 흐름도이다. 스텝 100에서, 이와 같은 자리올림이 있는 가산 명령과 마주쳤는지 여부를 검출하고, 마주치지 않은 경우에는 명령 디코더(6)가 처리회로(4)를 제어하여 검출된 명령의 종류에 의해 표시된 다른 연산을 행한다. 명령이 자리올림이 있는 가산 명령인 경우, 스텝 102에서 이 명령이 가산 변종인지 감산 변종인지 검출한다. 이때, 별도의 프레디케이트 레지스터로 자리올림 정보를 이동함으로써, 자리올림 수를 위해 성분들의 절반을 예약할 필요가 없어 한 개의 명령으로 데이터 소스 레지스터의 모든 성분들에 대해 작용할 수 있기 때문에, 이 명령에 대해 제1 및 제2(상단 또는 하단) 변종이 존재하지 않는다.

[0121] 자리올림이 있는 가산 명령이 가산 변종인 경우, 스텝 104에서, 목적지 레지스터의 각각의 성분 I의 새로운 값이, 이 데이터 성분 Zda1(i)의 이전 값, 데이터 소스 벡터 레지스터 Zn(i)의 대응하는 데이터 성분과, 입력 프레디케이트 레지스터 Pm(i)의 대응하는 프레디케이트 필드 I에서 얻어진 자리올림 입력의 합으로 설정된다. 또한, 출력 프레디케이트 레지스터 Pd2(i)의 대응하는 프레디케이트 필드가 가산으로부터 자리올림 출력과 같게 설정된다.

[0122] 명령이 감산 변종인 경우, 스텝 106에서 목적지 레지스터의 각각의 성분 Zda1(i)'이 이 성분 Zda1(i)의 이전의 값 - 데이터 소스 레지스터의 대응하는 성분 Zn(i) - 입력 프레디케이트 레지스터 Pm(i)의 대응하는 프레디케이트 필드에 의해 표시된 자리빌림 값에 대응하는 새로운 값으로 설정된다. 마찬가지로, 입력 프레디케이트 레지스터의 대응하는 프레디케이트 필드 Pd2(i)는 감산의 자리빌림 출력과 같게 설정된다. 이때, 스텝 104 및 106은 각각의 성분 위치에 대해 별도로 행해진다(즉, i=0…N-1, 이I N은 벡터 내부의 성분들의 총수이다).

[0123] 도 11 및 도 12는 동일한 레지스터 Zda1이 목적지 벡터 레지스터와 제1 소스 벡터 레지스터 양쪽으로서의 역할을 하는 것으로 도시하지만, 마찬가지로, 제1 소스 벡터 레지스터로서 별개의 레지스터가 지정되는 비파괴적인 변종을 제공하는 것도 가능하다.

[0124] 프레디케이션이 도 2에 도시된 것과 마찬가지로 성분 단위 기준으로 적용되는, 도 11에 도시된 자리올림이 있는 가산 명령의 프레디케이트된 변종이 제공될 수도 있다.

[0125] 도 13은 사용될 수 있는 시뮬레이터 구현예를 나타낸 것이다. 전술한 실시예는 해당 기술을 지원하는 특정한 처리 하드웨어를 작동하기 위한 장치 및 방법에 관해 본 발명을 구현하지만, 컴퓨터 프로그램의 사용을 통해 구현되는 본 발명에서 설명한 실시예에 따라 명령 실행 환경을 제공하는 것도 가능하다. 이와 같은 컴퓨터 프로그램은, 하드웨어 아키텍처의 소프트웨어 기반의 구현을 제공하는 한, 시뮬레이터로 부르는 경우가 많다. 다양한 시뮬레이터 컴퓨터 프로그램은 에뮬레이터, 가상머신, 모델, 및 동적 이진 변환기를 포함하는 이진 변환기를 포함한다. 보통, 시뮬레이터 구현은, 옵션으로 시뮬레이터 프로그램(710)을 지원하는 호스트 운영체계(720)를 실행하는 호스트 프로세서(730) 상에서 실행된다. 일부 구성에서는, 하드웨어와 제공된 명령 실행 환경 사이에 복수 층의 시뮬레이션이 존재하고, 및/또는 동일한 호스트 프로세서 상에서 복수의 별개의 명령 실행 환경이 제공된다. 역사적으로, 합당한 속도에서 실행되는 시뮬레이터 구현을 제공하기 위해 강력한 프로세서들이 요구되었지만, 이와 같은 접근방법은, 호환성이나 재사용 이유로 인해 다른 프로세서에 대해 네이티브한 코드를 실행하려는 요구가 있을 때 등과 같은, 특정한 상황에서 정당화된다. 예를 들어, 시뮬레이터 구현은, 호스트 프로세서 하드웨어에 의해 지원되지 않는 추가적인 기능을 갖는 명령 실행 환경을 제공하거나, 보통 다양한 하드웨어 아키텍처와 관련된 명령 실행 환경을 제공한다. 시뮬레이터의 개관에 대해서는 "Some Efficient Architecture Simulation Techniques", Robert Bedichek, Winter 1990 USENIX Conference, Pages 53-63에 기재되어 있다.

[0126] 본 실시예를 특정한 하드웨어 구성 또는 특징을 참조하여 설명하였지만, 시뮬레이션된 실시예에서는, 적절한 소프트웨어 구성 또는 특징에 의해 동등한 기능이 제공된다. 예를 들어, 특정한 회로가 시뮬레이션된 실시예에서는 컴퓨터 프로그램 논리로 구현된다. 마찬가지로, 레지스터 또는 캐시 등의 메모리 하드웨어도 시뮬레이션된 실시예에서는 소프트웨어 데이터 구조로 구현된다. 전술한 실시예에서 참조한 한 개 이상의 하드웨어 구성요소들이 호스트 하드웨어(예를 들어, 호스트 프로세서(230) 상에 존재하는 구성에서는, 적절한 경우에, 일부 시뮬레이션된 실시예가 호스트 하드웨어를 이용한다.

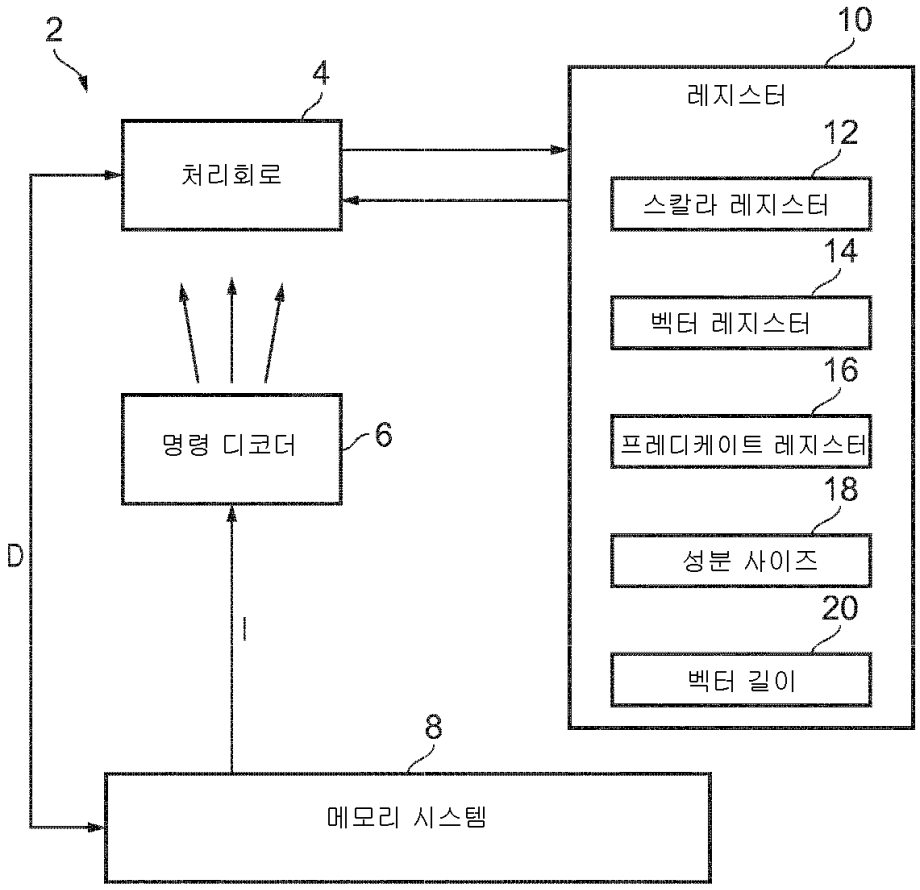
[0127] 시뮬레이터 프로그램(210)은, 컴퓨터 판독가능한 기억매체(이것은 비일시적인 매체일 수도 있다)에 기억되고, 시뮬레이터 프로그램(210)에 의해 모델링되고 있는 하드웨어 아키텍처의 응용 프로그램 인터페이스와 동일한 타겟 코드(200)에 대한 프로그램 인터페이스(명령 실행 환경)를 제공한다. 따라서, 타겟 코드(200)의 프로그램 명령들은 시뮬레이터 프로그램(210)을 사용하여 명령 실행 환경 내에서 실행됨으로써, 전술한 장치(2)의 하드웨어 특징을 실제로 갖지 않는 호스트 컴퓨터(230)가 이들 특징을 에뮬레이트할 수 있다. 예를 들어, 시뮬레이터 프로그램(210)은 명령 디코더(6), 처리회로(4) 및 레지스터들(10)과 각각 기능적으로 대응하는 명령 디코딩 프로그램 로직(212), 처리 프로그램 로직(214) 및 레지스터 데이터 구조(216)를 구비한다. 예를 들어, 디코딩 프로그램 로직(212)은 타겟 코드(200)의 명령의 명령 인코딩을 검사하여 행하려고 하는 연산을 결정하는 시뮬레이터 프로그램(210)의 일련의 "if" 문을 포함할 수도 있으며, 처리 프로그램 로직(214)은 특정한 명령들에 대해 활성화될 "then" 루틴들에 대응하여 이들 명령을 호스트 운영체계(220)에 의해 실행할 대응하는 명령들에 매핑한다. 레지스터 데이터 구조(216)는 시뮬레이터 프로그램(210)에 의해 시뮬레이트되고 있는 시뮬레이트된 장치(2)의 레지스터들을 에뮬레이트하기 위해 할당된 메모리 영역을 포함할 수도 있다.

[0128] 본 발명에서, 단어 "하도록 구성된"은 장치의 구성요소가 정의된 동작을 행할 수 있는 구성을 갖는다는 것을 의미하기 위해 사용된다. 이와 관련하여, "구성"은 하드웨어 또는 소프트웨어의 배치 또는 상호접속 방식을 의미한다. 예를 들어, 장치는 정의된 동작을 제공하는 전용 하드웨어를 갖거나, 프로세서 또는 기타의 처리 장치가 기능을 행하도록 프로그래밍되어도 된다. "하도록 구성된"은 이 장치의 구성요소가 정의된 동작을 제공하기 위해 어떤 식으로 변경될 필요가 있는 것을 시사하는 것은 아니다.

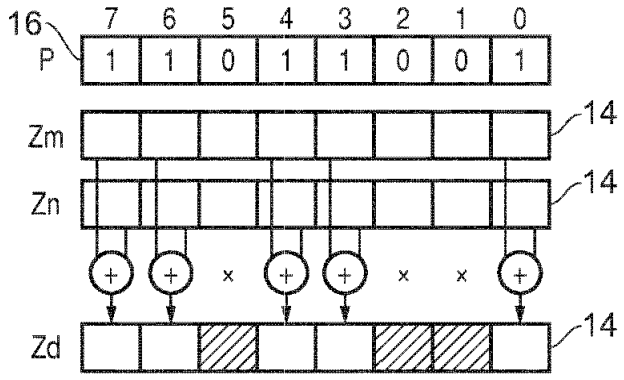
[0129] 첨부도면을 참조하여 본 발명의 예시적인 실시예들을 상세히 설명하였지만, 본 발명은 이들 실시예에 한정되지 않으며, 첨부된 청구범위의 보호범위 및 사상을 벗어나지 않으면서 본 발명이 속한 기술분야의 당업자에 의해 다양한 변경, 부가 및 변화가 행해질 수 있다는 것은 자명하다. 예를 들면, 종속항들의 특징들의 다양한 조합이 독립항들의 특징과 행해질 수도 있다.

## 도면

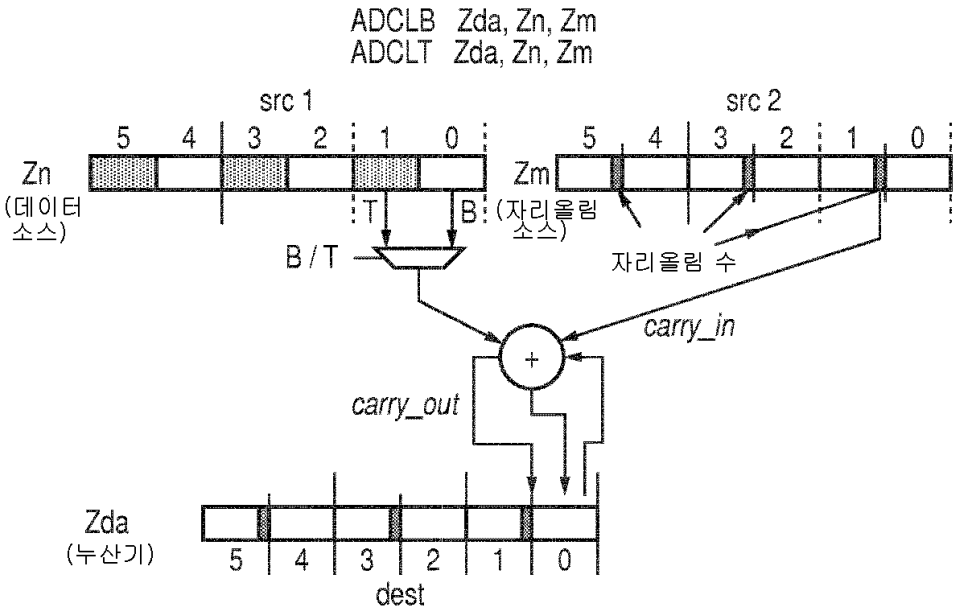
### 도면1


2
4
처리회로
10
레지스터
12
스칼라 레지스터
14
벡터 레지스터
16
프레디케이트 레지스터
18
성분 사이즈
20
벡터 길이
명령 디코더
6
D
I
8
메모리 시스템

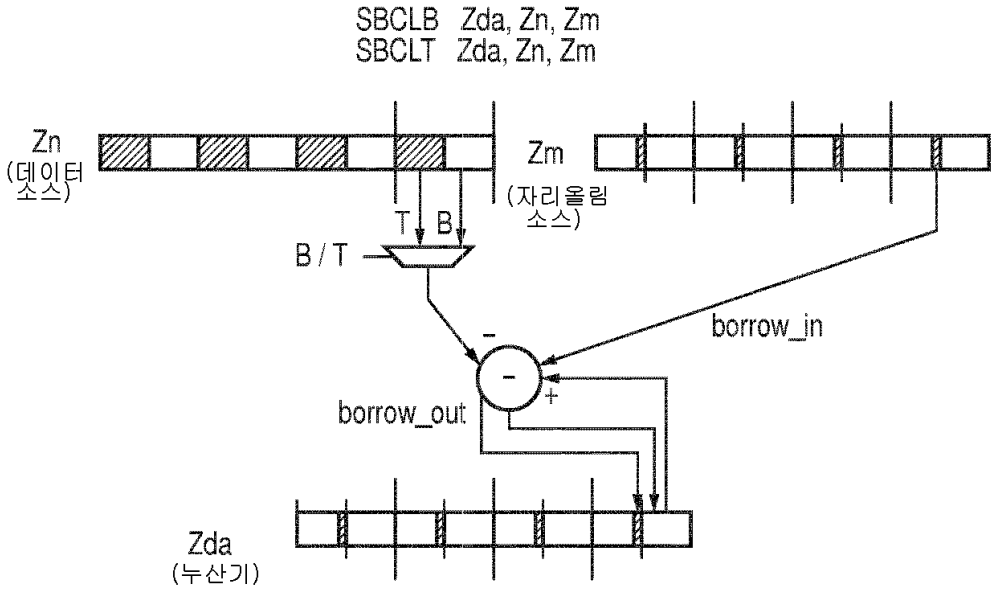
### 도면2


16
P
7 6 5 4 3 2 1 0
1 1 0 1 1 0 0 1
Zm
Zn
Zd
14
14
14

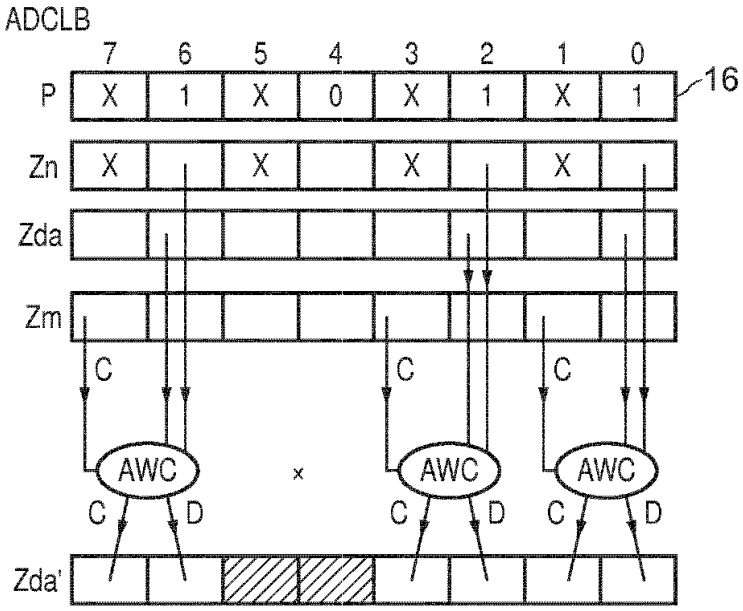
도면3


ADCLB Zda, Zn, Zm
ADCLT Zda, Zn, Zm
src 1
src 2
5 4 3 2 1 0
Zn
(데이터 소스)
Zm
(자리올림 소스)
T
B
B / T
자리올림 수
carry_in
+
carry_out
Zda
(누산기)
5 4 3 2 1 0
dest

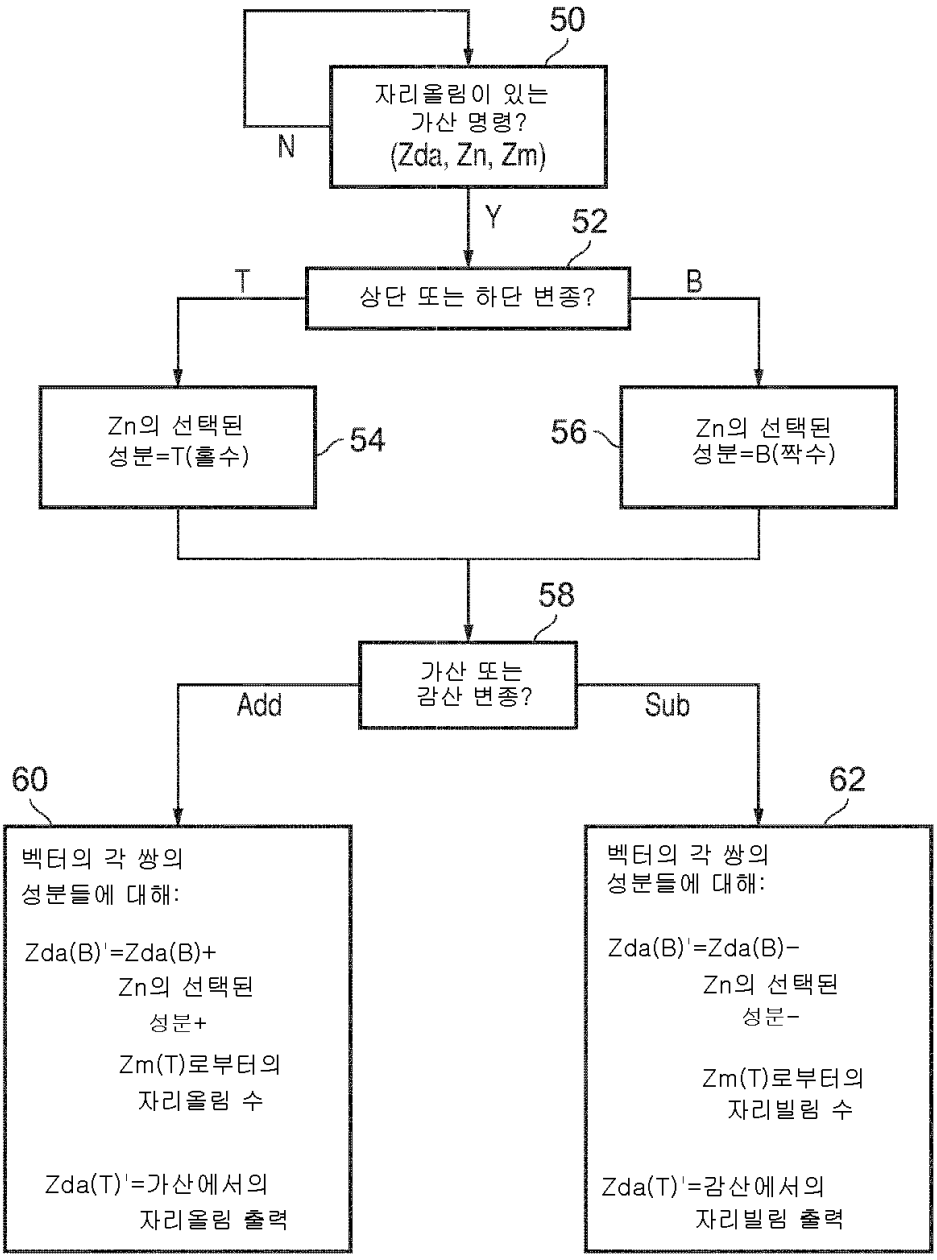
도면4


SBCLB Zda, Zn, Zm
SBCLT Zda, Zn, Zm
Zn
(데이터 소스)
Zm
(자리올림 소스)
T
B
B / T
borrow_in
−
+
borrow_out
Zda
(누산기)

도면4a


ADCLB
7 6 5 4 3 2 1 0
P X 1 X 0 X 1 X 1
16
Zn X X X X
Zda
Zm
C C C
AWC AWC AWC
C D C D C D
Zda'

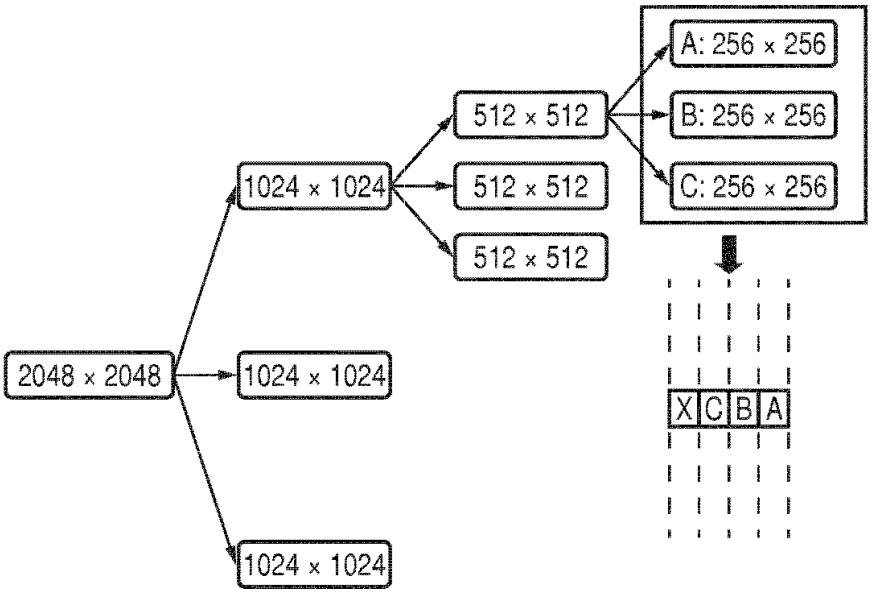
도면5

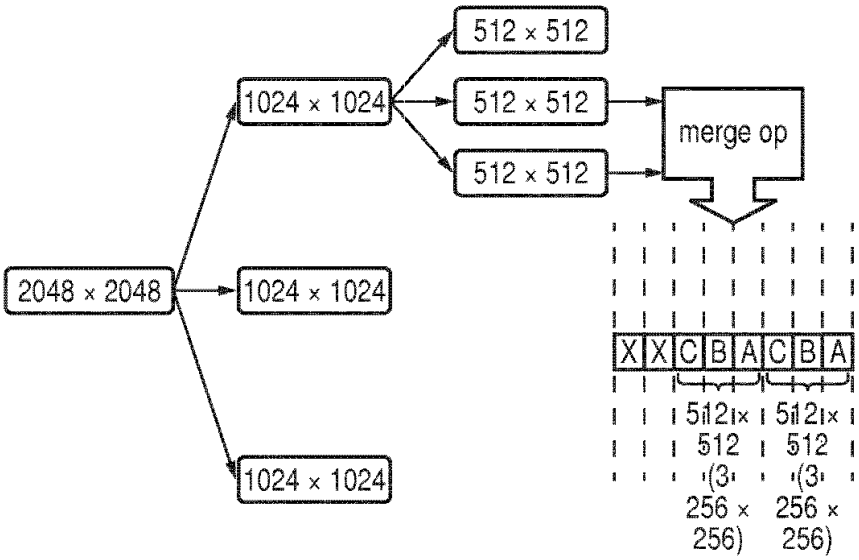

50
자리올림이 있는
가산 명령?
(Zda, Zn, Zm)
N
Y
52
상단 또는 하단 변종?
T
B
Zn의 선택된
성분=T(홀수)
54
56
Zn의 선택된
성분=B(짝수)
58
가산 또는
감산 변종?
Add
Sub
60
벡터의 각 쌍의
성분들에 대해:
Zda(B)'=Zda(B)+
Zn의 선택된
성분+
Zm(T)로부터의
자리올림 수
Zda(T)'=가산에서의
자리올림 출력
62
벡터의 각 쌍의
성분들에 대해:
Zda(B)'=Zda(B)-
Zn의 선택된
성분-
Zm(T)로부터의
자리빌림 수
Zda(T)'=감산에서의
자리빌림 출력

도면6

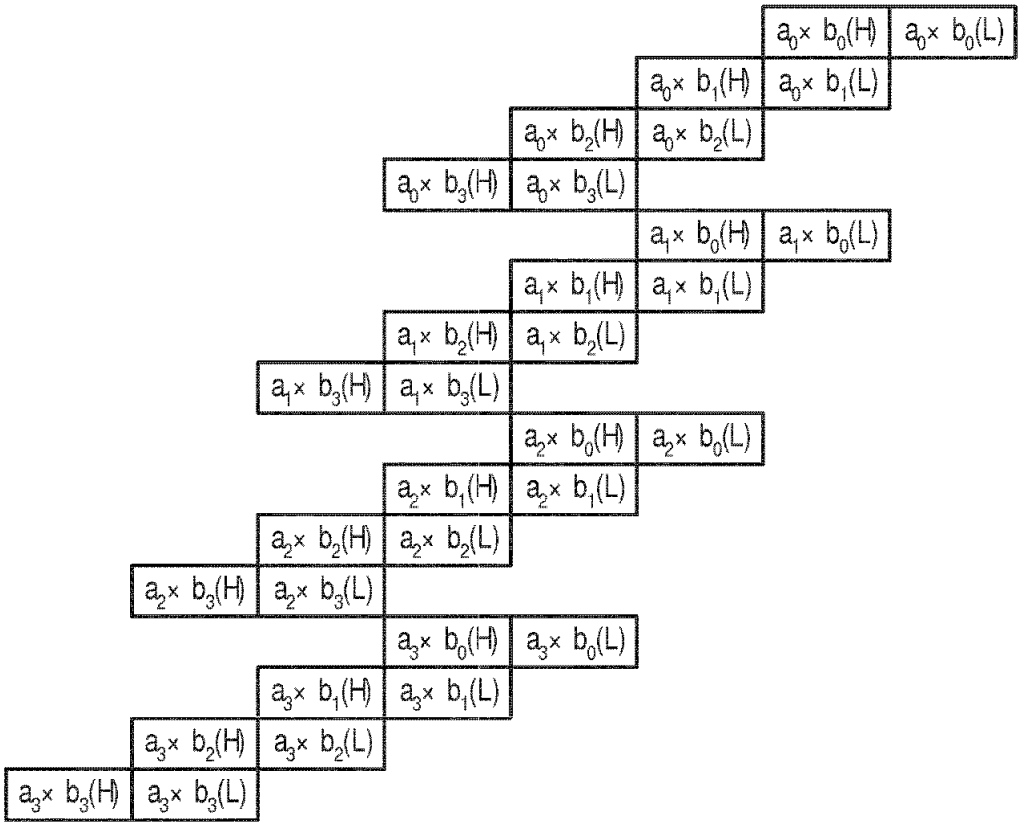
$$C = A_H \cdot B_H 2^n + ((A_L + A_H) \cdot (B_L + B_H) - A_L \cdot B_L - A_H \cdot B_H) 2^{n/2} + A_L \cdot B_L$$

$$(A_L + A_H) \cdot (B_L + B_H) - A_L \cdot B_L - A_H \cdot B_H = A_L \cdot B_L + A_H \cdot B_H - |A_H - A_L| \cdot |B_H - B_L|$$



도면7


$a_0 \times b_0(H)$
$a_0 \times b_0(L)$
$a_0 \times b_1(H)$
$a_0 \times b_1(L)$
$a_0 \times b_2(H)$
$a_0 \times b_2(L)$
$a_0 \times b_3(H)$
$a_0 \times b_3(L)$
$a_1 \times b_0(H)$
$a_1 \times b_0(L)$
$a_1 \times b_1(H)$
$a_1 \times b_1(L)$
$a_1 \times b_2(H)$
$a_1 \times b_2(L)$
$a_1 \times b_3(H)$
$a_1 \times b_3(L)$
$a_2 \times b_0(H)$
$a_2 \times b_0(L)$
$a_2 \times b_1(H)$
$a_2 \times b_1(L)$
$a_2 \times b_2(H)$
$a_2 \times b_2(L)$
$a_2 \times b_3(H)$
$a_2 \times b_3(L)$
$a_3 \times b_0(H)$
$a_3 \times b_0(L)$
$a_3 \times b_1(H)$
$a_3 \times b_1(L)$
$a_3 \times b_2(H)$
$a_3 \times b_2(L)$
$a_3 \times b_3(H)$
$a_3 \times b_3(L)$

도면8

```
1       mul         za1b0L, z1.d, z4.d
2       mulh        za1b0H, z1.d, z4.d
3       mul         za1b1L, z1.d, z5.d
4       mulh        za1b1H, z1.d, z5.d
5       mul         za1b2L, z1.d, z6.d
6       mulh        za1b2H, z1.d, z6.d
7       mul         za1b3L, z1.d, z7.d
8       mulh        za1b3H, z1.d, z7.d
9
10      adclb       zacc1LB, za1b0H, z_zero
11      adclt       zacc1LT, za1b0H, z_zero
12      adclb       zacc0HB, za1b0L, z_zero
13      adclt       zacc0HT, za1b0L, z_zero
14      adclb       zacc1HB, za1b1H, zacc1LB
15      adclt       zacc1HT, za1b1H, zacc1LT
16      adclb       zacc1LB, za1b1L, zacc0HB
17      adclt       zacc1LT, za1b1L, zacc0HT
18      adclb       zacc2LB, za1b2H, zacc1HB
19      adclt       zacc2LT, za1b2H, zacc1HT
20      adclb       zacc1HB, za1b2L, zacc1LB
21      adclt       zacc1HT, za1b2L, zacc1LT
22      //store
23      trn1        ztemp, zacc0HB, zacc0HT
24      st1d        {ztemp}, p1, [zptr, #8]
25      //**
26      adclb       zacc2HB, z_zero, zacc2LB
27      adclt       zacc2HT, z_zero, zacc2LT //no carryout possible
28
29      adclb       zacc2LB, za1b3L, zacc1HB
30      adclt       zacc2LT, za1b3L, zacc1HT
31      //fixup carry
32      adclb       zacc2HB, za1b3H, zacc2LB
33      adclt       zacc2HT, za1b3H, zacc2LT
34      trn2        zacc3LB, zacc2HB, z_zero
35      trn2        zacc3LT, zacc2HT, z_zero
```

도면9

$$ADCLB(AccB[i], P_{0i}lo, 0) \rightarrow AccB[i], c_{AccBi}$$

$$ADCLB(AccB[i+1], P_{1i}lo, c_{AccBi}) \rightarrow AccB[i+1], c_{AccBi+1}$$

$$ADCLB(AccB[i+2], P_{2i}lo, c_{AccBi+1}) \rightarrow AccB[i+2], c_{AccBi+2}$$

$$ADCLB(AccB[i+1], P_{0i}hi, 0) \rightarrow AccB[i+1], c_{AccBi+1}$$

$$ADCLB(AccB[i+3], P_{3i}lo, c_{AccBi+2}) \rightarrow AccB[i+3], c_{AccBi+3}$$

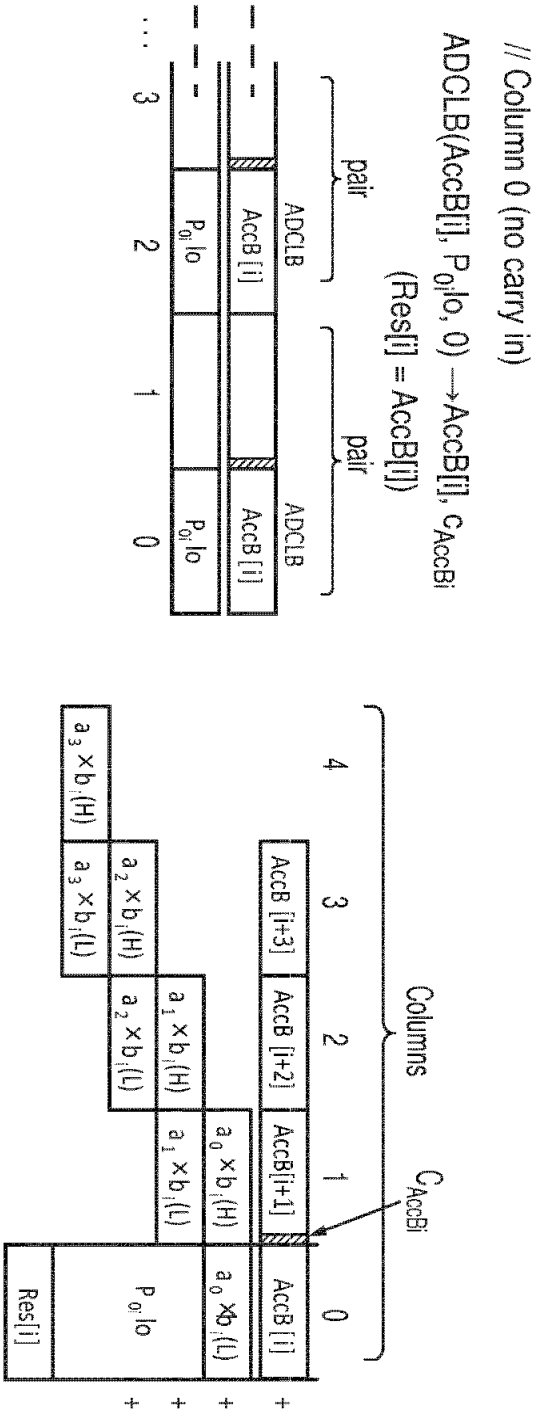
$$ADCLB(AccB[i+2], P_{1i}hi, c_{AccBi+1}) \rightarrow AccB[i+2], c_{AccBi+2}$$

$$ADCLB(0, P_{3i}hi, c_{AccBi+3}) \rightarrow AccB[i+4], 0$$

$$ADCLB(AccB[i+3], P_{2i}hi, c_{AccBi+2}) \rightarrow AccB[i+3], c_{AccBi+3}$$

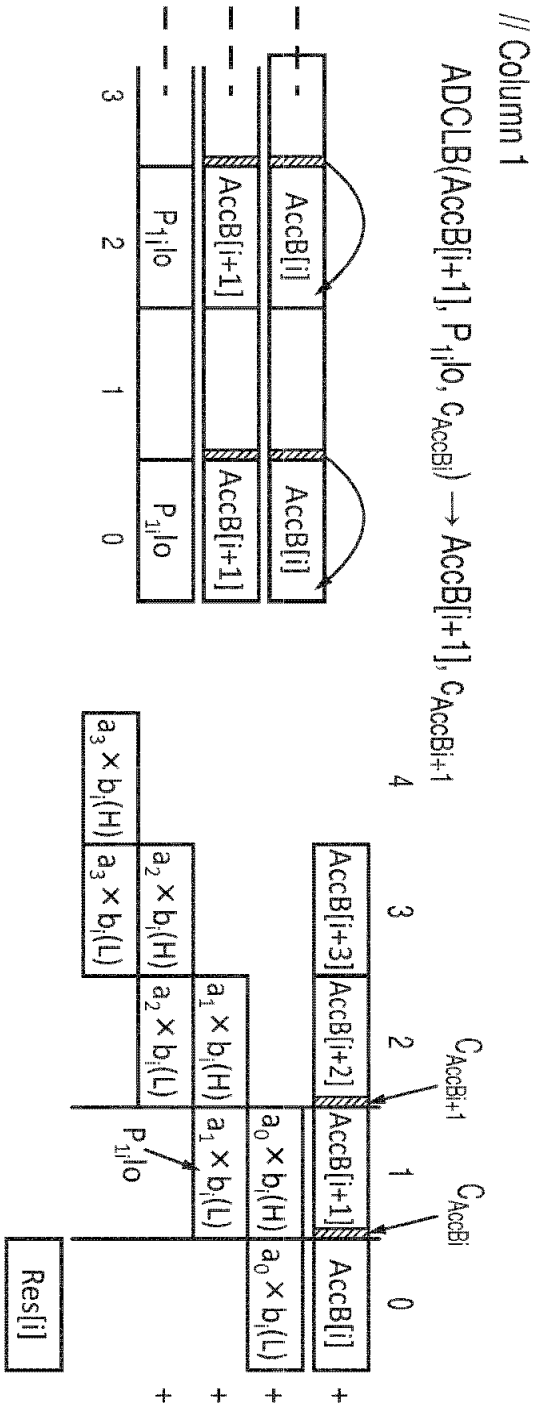
$$ADCLB(AccB[i+4], 0, c_{AccBi+3}) \rightarrow AccB[i+4], 0$$

도면10a

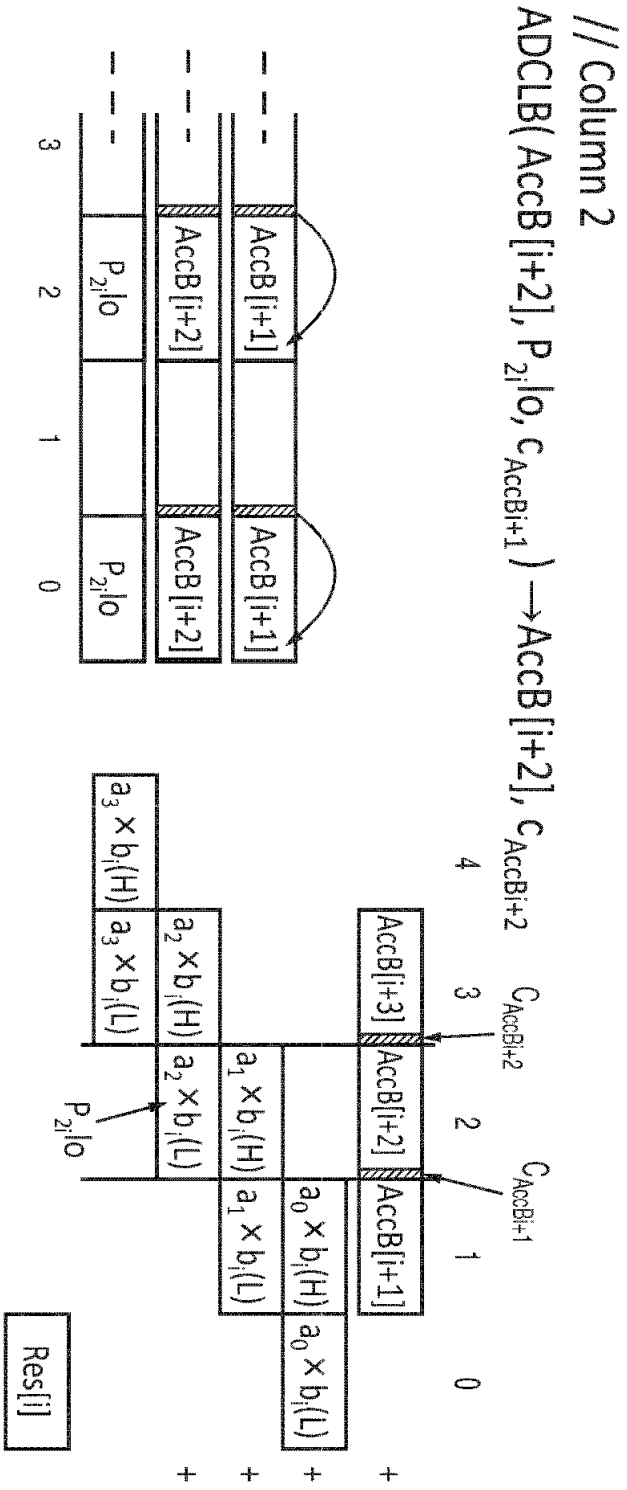

// Column 0 (no carry in)
ADCLB(AccB[i], P0i lo, 0) → AccB[i], cAccBi
(Res[i] = AccB[i])
pair
pair
ADCLB
ADCLB
AccB [i]
AccB [i]
P0i lo
P0i lo
3
2
1
0


Columns
cAccBi
4
3
2
1
0
AccB [i+3]
AccB [i+2]
AccB[i+1]
AccB [i]
a3 × b1(H)
a3 × b1(L)
a2 × b1(H)
a2 × b1(L)
a1 × b1(H)
a1 × b1(L)
a0 × b1(H)
a0 × b1(L)
P0 lo
Res[i]
+
+
+
+

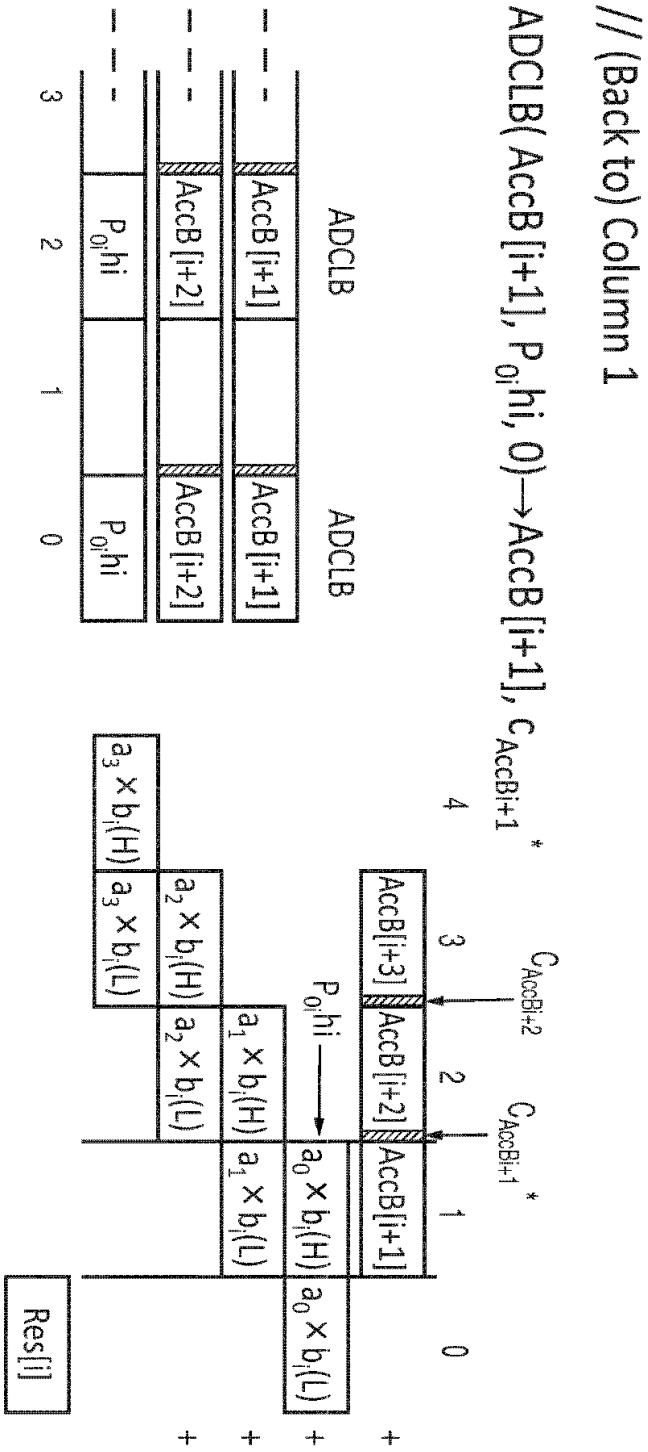
도면10b


// Column 1
ADCLB(AccB[i+1], P1i-1o, cAccBi) → AccB[i+1], cAccBi+1
AccB[i]
AccB[i+1]
P1i-1o
AccB[i]
AccB[i+1]
P1i-1o
3
2
1
0
4
3
2
1
0
AccB[i+3]
AccB[i+2]
AccB[i+1]
AccB[i]
cAccBi+1
cAccBi
a3 × bi(H)
a3 × bi(L)
a2 × bi(H)
a2 × bi(L)
a1 × bi(H)
a1 × bi(L)
a0 × bi(H)
a0 × bi(L)
P1i-1o
Res[i]
+
+
+
+

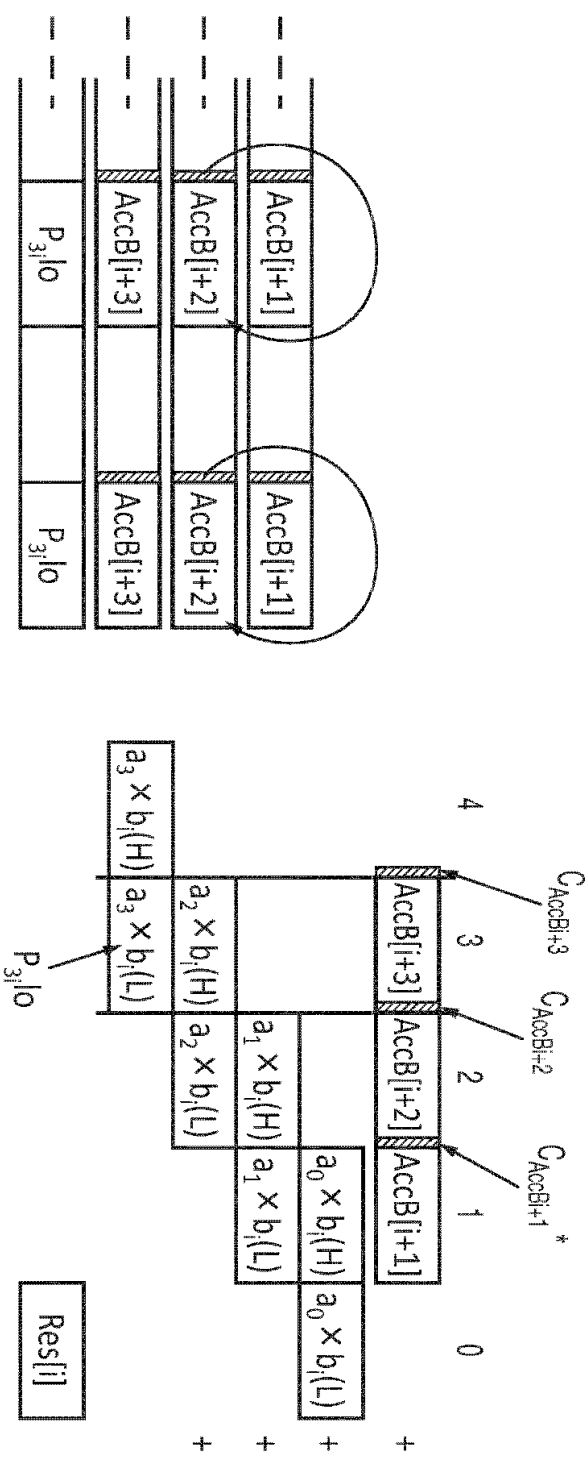
도면10c


// Column 2
ADCLB( AccB [i+2], P2i-lo, cAccBi+1 ) →AccB [i+2], cAccBi+2
AccB [i+1]
AccB [i+2]
P2i-lo
AccB[i+3]
AccB[i+2]
AccB[i+1]
cAccBi+2
cAccBi+1
a3 × b1(H)
a3 × b1(L)
a2 × b1(H)
a2 × b1(L)
a1 × b1(H)
a1 × b1(L)
a0 × b1(H)
a0 × b1(L)
Res[i]
4
3
2
1
0
+

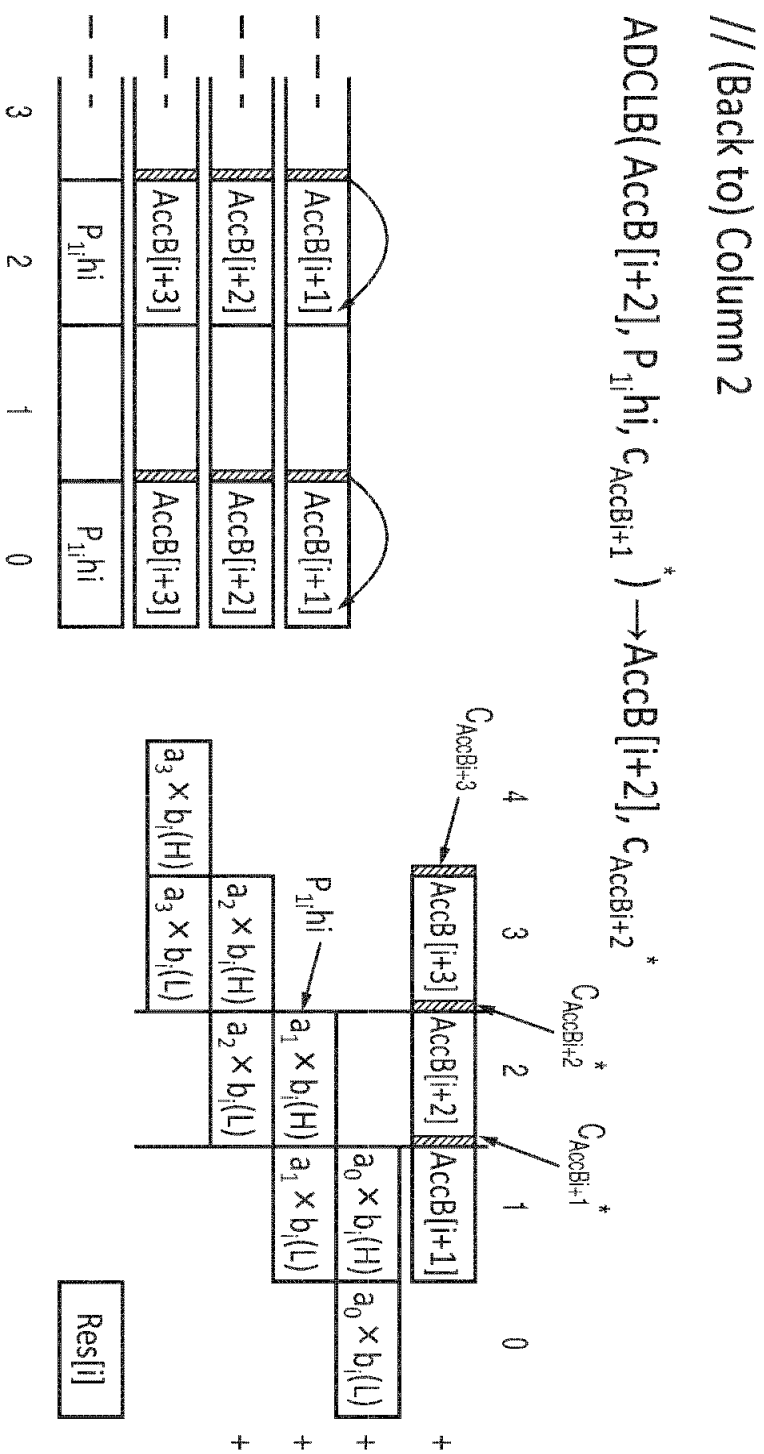
도면10d


// (Back to) Column 1
ADCLB( AccB [i+1], P0lhi, 0)→AccB [i+1], cAccBi+1
ADCLB
ADCLB
AccB [i+1]
AccB [i+2]
P0lhi
AccB [i+1]
AccB [i+2]
P0lhi
3
2
1
0
4
3
2
1
0
AccB[i+3]
AccB [i+2]
AccB [i+1]
cAccBi+2
cAccBi+1 *
P0lhi
a0 × b1(H)
a0 × b1(L)
a1 × b1(H)
a1 × b1(L)
a2 × b1(H)
a2 × b1(L)
a3 × b1(H)
a3 × b1(L)
+
+
+
+
Res[i]

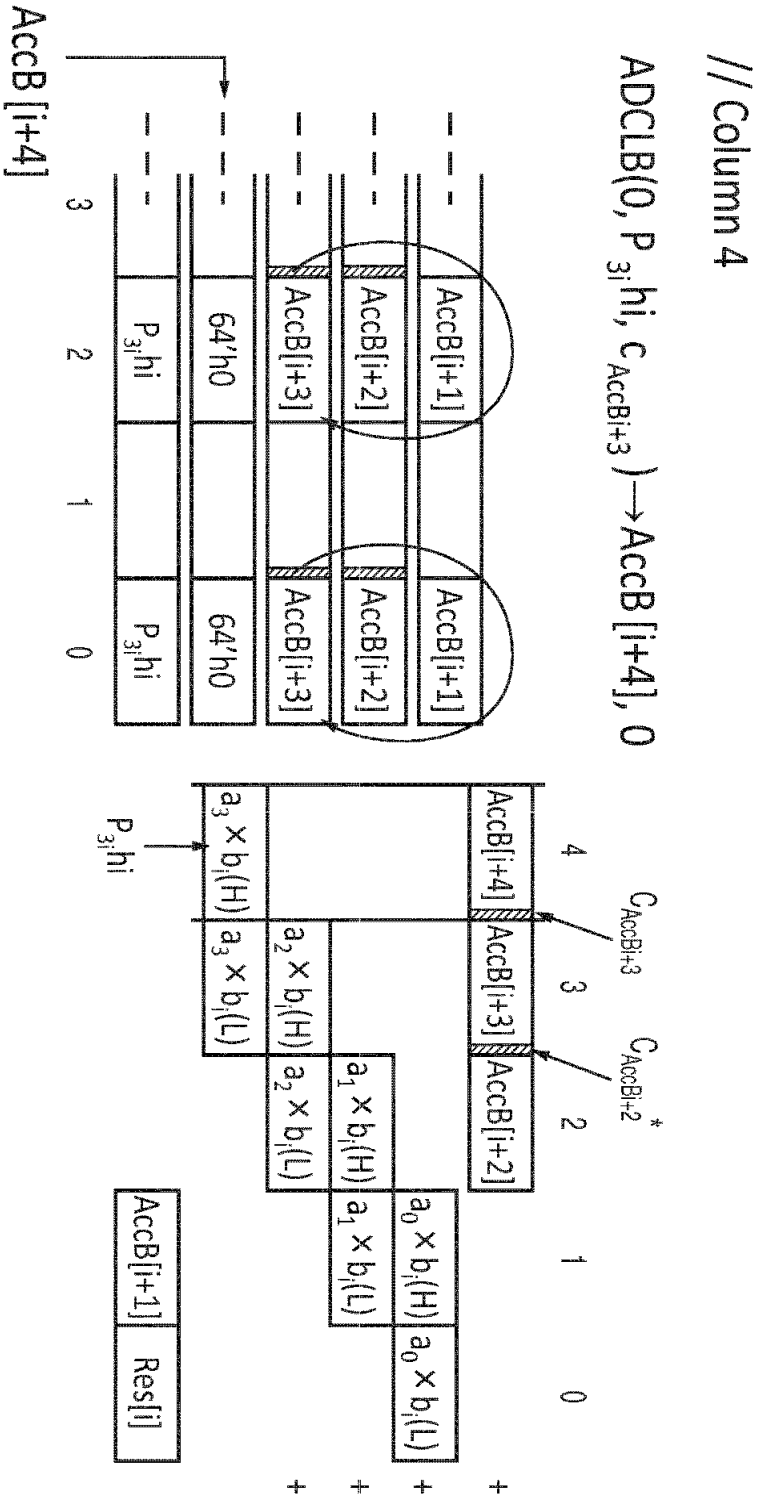
도면10e


// Column 3
ADCLB( AccB [i+3], P3:lo, cAccBi+2 )→AccB [i+3], cAccBi+3
AccB[i+1]
AccB[i+2]
AccB[i+3]
P3:lo
3
2
1
0
4
cAccBi+3
cAccBi+2
cAccBi+1
*
a3 × b:(H)
a3 × b:(L)
a2 × b:(H)
a2 × b:(L)
a1 × b:(H)
a1 × b:(L)
a0 × b:(H)
a0 × b:(L)
+
Res[i]

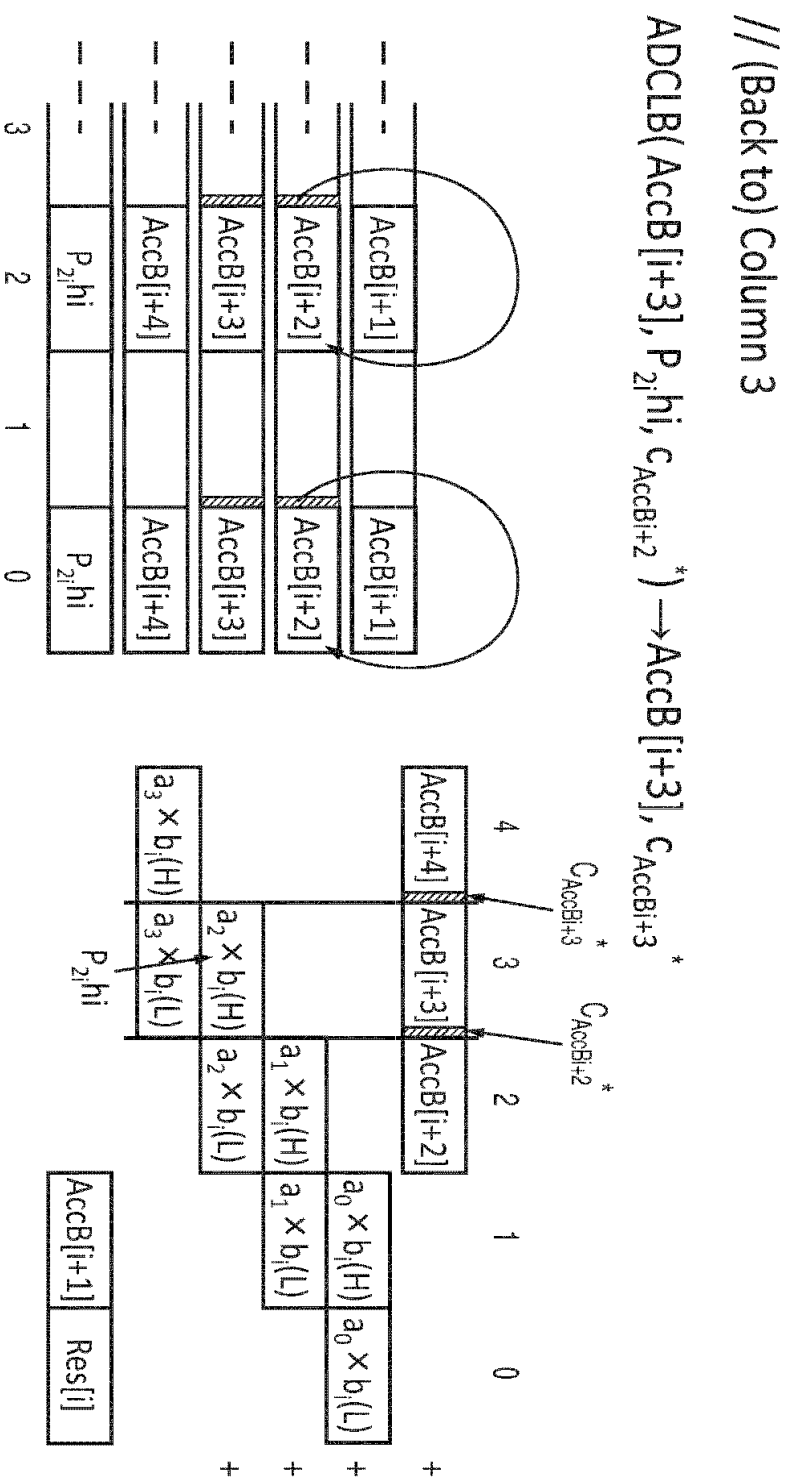
도면10f


// (Back to) Column 2
ADCLB( AccB [i+2], P1hi, CAccBi+1 * ) →AccB [i+2], CAccBi+2 *
CAccBi+3
CAccBi+2 *
CAccBi+1 *
P1hi
AccB[i+3]
AccB[i+2]
AccB[i+1]
a3 × b1(H)
a3 × b1(L)
a2 × b1(H)
a2 × b1(L)
a1 × b1(H)
a1 × b1(L)
a0 × b1(H)
a0 × b1(L)
Res[i]
4
3
2
1
0
+

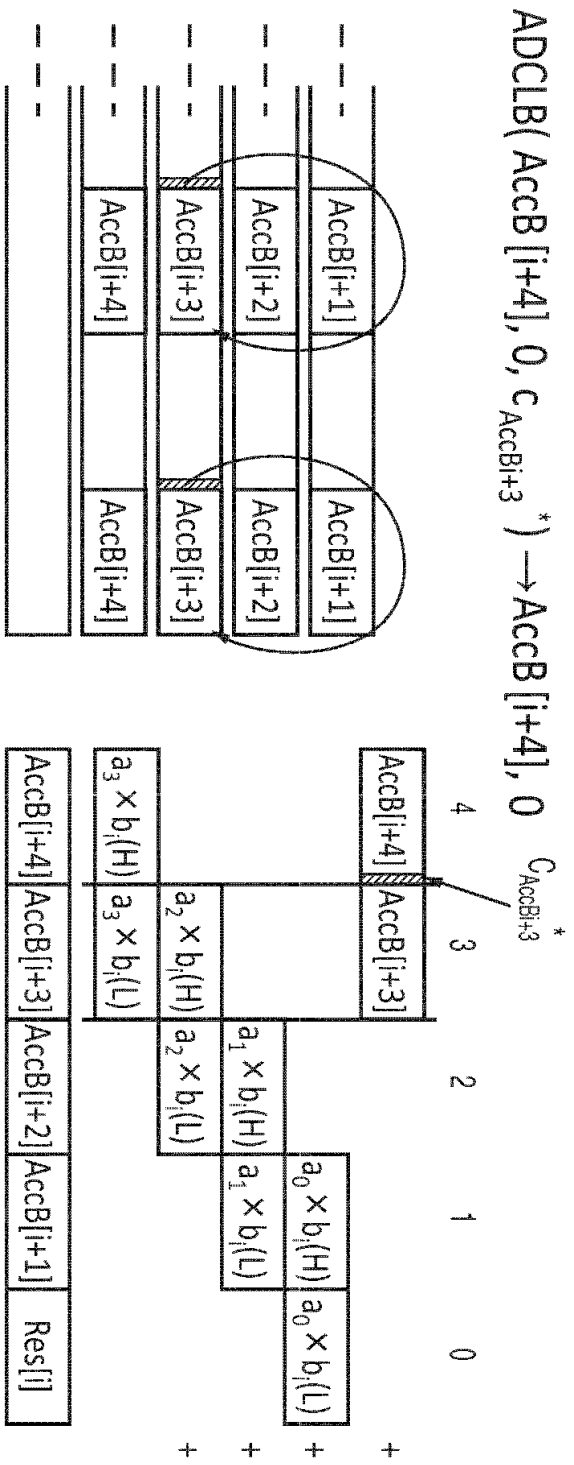
도면10g


// Column 4
ADCLB(0, P 3i hi, c AccBi+3 )→AccB [i+4], 0
AccB [i+4]
AccB[i+1]
AccB[i+2]
AccB[i+3]
64'h0
P3i hi
3
2
1
0
AccB[i+4]
AccB[i+3]
AccB[i+2]
c AccBi+3
c AccBi+2 *
4
3
2
1
0
a3 × b i(H)
a3 × b i(L)
a2 × b i(H)
a2 × b i(L)
a1 × b i(H)
a1 × b i(L)
a0 × b i(H)
a0 × b i(L)
+
AccB[i+1]
Res[i]

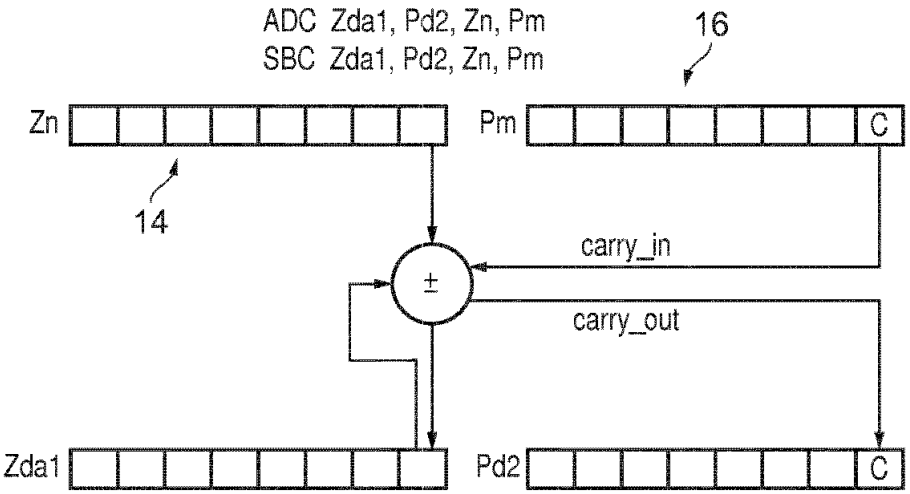
도면10h


// (Back to) Column 3
ADCLB( AccB [i+3], P2i hi, cAccBi+2 *) →AccB [i+3], cAccBi+3 *
AccB[i+1]
AccB[i+2]
AccB[i+3]
AccB[i+4]
P2i hi
3
2
1
0
cAccBi+3 *
cAccBi+2 *
4
3
2
1
0
a0 × b1(H)
a0 × b1(L)
a1 × b1(H)
a1 × b1(L)
a2 × b1(H)
a2 × b1(L)
a3 × b1(H)
a3 × b1(L)
Res[i]
+

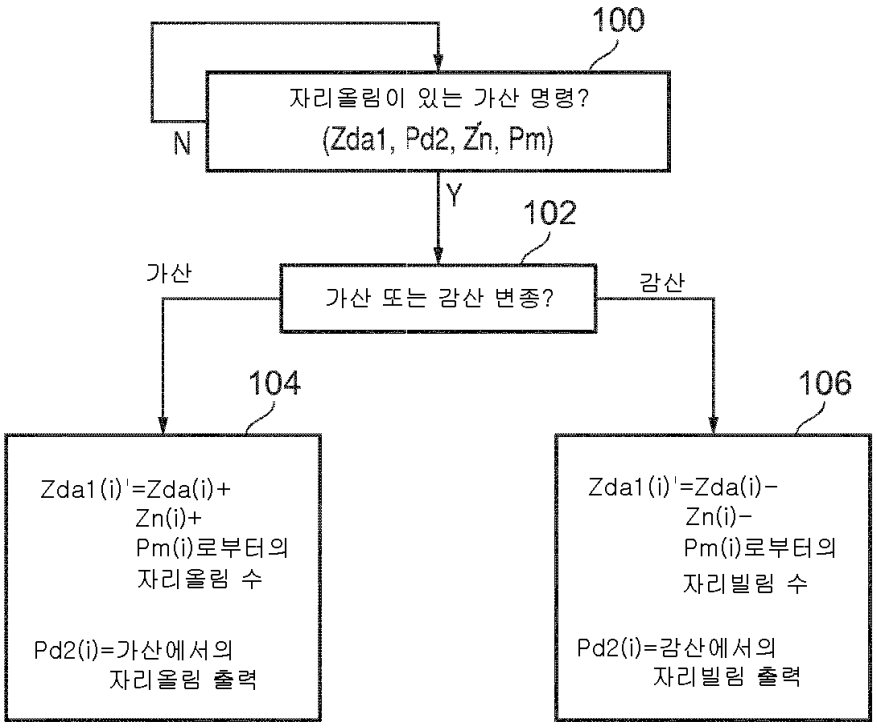
도면10i


// Column 4
ADCLB( AccB [i+4], 0, $c_{AccBi+3}^{*}$ ) → AccB [i+4], 0
$c_{AccBi+3}^{*}$
AccB[i+1]
AccB[i+2]
AccB[i+3]
AccB[i+4]
$a_0 \times b_i(L)$
$a_0 \times b_i(H)$
$a_1 \times b_i(L)$
$a_1 \times b_i(H)$
$a_2 \times b_i(L)$
$a_2 \times b_i(H)$
$a_3 \times b_i(L)$
$a_3 \times b_i(H)$
Res[i]
4
3
2
1
0
+

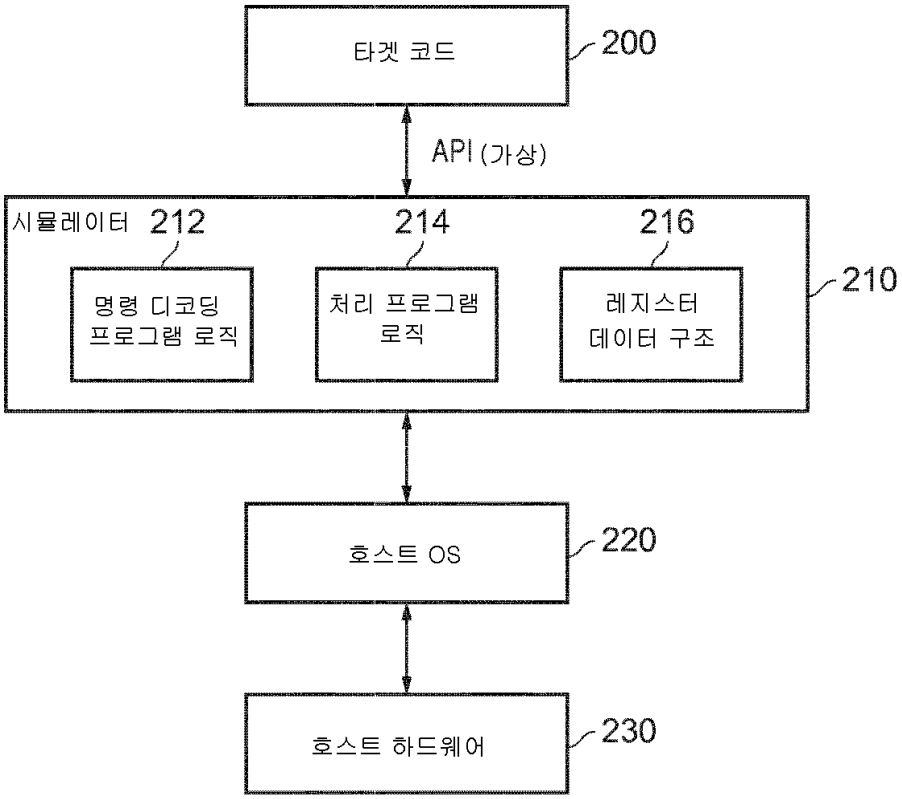
도면11


ADC Zda1, Pd2, Zn, Pm
SBC Zda1, Pd2, Zn, Pm
16
Zn
Pm
C
14
carry_in
±
carry_out
Zda1
Pd2
C

도면12


100
자리올림이 있는 가산 명령?
(Zda1, Pd2, Zn, Pm)
N
Y
102
가산
가산 또는 감산 변종?
감산
104
106
Zda1(i)'=Zda(i)+
Zn(i)+
Pm(i)로부터의
자리올림 수
Pd2(i)=가산에서의
자리올림 출력
Zda1(i)'=Zda(i)-
Zn(i)-
Pm(i)로부터의
자리빌림 수
Pd2(i)=감산에서의
자리빌림 출력

도면13


타겟 코드
200
API (가상)
시뮬레이터
212
214
216
명령 디코딩 프로그램 로직
처리 프로그램 로직
레지스터 데이터 구조
210
호스트 OS
220
호스트 하드웨어
230