(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2014년12월02일
(11) 등록번호 10-1467467
(24) 등록일자 2014년11월25일

(51) 국제특허분류(Int. Cl.)
H03M 13/41 (2006.01)
(21) 출원번호 10-2012-7024288(분할)
(22) 출원일자(국제) 2008년10월27일
심사청구일자 2013년10월25일
(85) 번역문제출일자 2012년09월17일
(65) 공개번호 10-2012-0120448
(43) 공개일자 2012년11월01일
(62) 원출원 특허 10-2010-7011361
원출원일자(국제) 2008년10월27일
심사청구일자 2010년05월25일
(86) 국제출원번호 PCT/EP2008/064530
(87) 국제공개번호 WO 2009/053490
국제공개일자 2009년04월30일
(30) 우선권주장
07119378.3 2007년10월26일
유럽특허청(EPO)(EP)

(56) 선행기술조사문헌
Hui-Ling Lou, "Implementing the Viterbi Algorithm", IEEE Signal Processing Magazine, September 1995, pp.42-52.
JP2006041616 A
W. Grass, "Higher performance and lower power enhancements to VLIW architectures", IEEE Workshop on Signal Processing Systems, September 2001, p.157.
K. Y. Lin 외 1인, "A parallel Viterbi algorithm for decoding convolutional codes on SIMD machines", Proceedings of 35th Midwest Symposium on Circuits and Systems, 1권, August 1992, pp.361-364.

(73) 특허권자
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
영, 필
영국 엔엔45에이디 노르트햄프톤 노르트햄프톤셔 그랜지 파크 퀸톤사이드 43
(74) 대리인
특허법인 남앤드남

전체 청구항 수 : 총 15 항 심사관 : 성경아

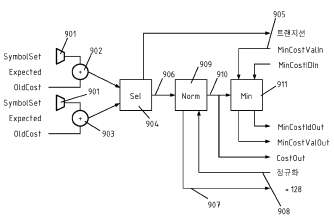
(54) 발명의 명칭 **최적화된 비터비 디코더 및 GNSS 수신기**

(57) 요 약

본 발명은 프로세서 내에서 구현된 특수 명령 세트에 기초하며, 하드웨어 복잡도를 현저하게 증가시키지 않고 훨씬 더 낮은 CPU 로딩으로 비터비 프로세싱을 관리할 수 있게 하는 비터비 디코더를 개시한다. SV 네비게이션에 대해 특정한 적절한 설계 제약의 주의 깊은 애플리케이션 및 비터비 알고리즘의 분석에 의해, 최적화된 아키텍처

(뒷면에 계속)

대 표 도 - 도9



는 비터비 가속화 로직을 효율적으로 GNSS 칩셋으로 포함시킴으로써 달성될 수 있다.

특허청구의 범위

청구항 1

컨볼루션 인코딩된 데이터를 디코딩하기 위해 최적화된 프로세서로서,

상기 프로세서는, 연관된 상태 심볼 해석 비용들이 루프 불변(loop invariant)들로서 표현되어 하나의 32비트 코어 레지스터에 저장되게 하며 비터비 디코딩 알고리즘의 비터비 루프에 대한 트랜지션 정보가 제 2 레지스터에 저장되는 순서로, 상기 비터비 루프의 8회 반복들 – 각각의 반복은 8개의 연속하는 현재 상태들을 프로세싱함 – 로 64 비터비(Viterbi) 상태들을 프로세싱하고,

상기 프로세서는 명령에 의해 어드레스지정되는 경우, 상기 명령에 의해 명시적으로 어드레스지정되지 않은 레지스터들 및/또는 신호들로부터 계산된 값을 리턴시키는 팬텀(phantom) 레지스터들을 사용하여 상기 프로세서의 레지스터 세트를 확장시키도록 구성되는,

프로세서.

청구항 2

제1항에 있어서,

상기 팬텀 레지스터들은 루프 제어 레지스터로서 사용되는 코어 레지스터 필드, 및 데이터 구조들에 대한 베이스 어드레스 및 현재 상태 비용 세트 할당을 표시하고 현재 및 다음 상태 테이블들의 매핑을 선택하는 상태 레지스터 필드에 기초하여, 소스 및 목적지 상태 비용 오퍼랜드들에 대한 어드레스들을 리턴시키도록 구성되는,

프로세서.

청구항 3

컨볼루션 인코딩된 데이터를 디코딩하기 위한 프로세서로서,

상기 프로세서는, 연관된 상태 심볼 해석 비용들이 루프 불변들로서 표현되어 하나의 32비트 코어 레지스터에 저장되게 하며 비터비 디코딩 알고리즘의 비터비 루프에 대한 트랜지션 정보가 제 2 레지스터에 저장되는 순서로, 상기 비터비 루프의 8회 반복들 – 각각의 반복은 8개의 연속하는 현재 상태들을 프로세싱함 – 로 64 비터비(Viterbi) 상태들을 프로세싱하고,

상기 프로세서는 코어 데이터 레지스터들에 저장된 다수의 오퍼랜드들 상에서 동작하는 가산-비교-선택 동작을 구현하기 위한 명령을 실행하도록 구성되고,

상기 명령은 상기 명령에 의해 묵시적으로 어드레스지정되는 부가적인 CPU 레지스터들을 이용하며, 프로세서들의 제약들인 명령 세트 아키텍처(Instruction Set Architecture)를 위배하지 않고 SIMD 아키텍처가 구현되게 하는,

프로세서.

청구항 4

제3항에 있어서,

상기 명령은 하나 이상의 가산-비교-선택 동작들을 동시에 수행하는 것을 포함하고 다수의 목적지 레지스터들에 저장될 수 있는 적절한 상태들에 대한 관련 상태 비용 필드들을 업데이트시키는,

프로세서.

청구항 5

제4항에 있어서,

상기 명령은 상기 비터비 디코딩 알고리즘의 상기 비터비 루프에 대한 최저 누적 비용 상태의 계산을 수행하는,

프로세서.

청구항 6

제4항에 있어서,

상기 명령은 상기 비터비 루프에 대한 최저 비용 상태 결과가 미리 결정된 임계치보다 큰지의 여부를 계산하는,

프로세서.

청구항 7

제4항에 있어서,

상기 명령은 계산된 상태 비용들을 저장하기 이전에 상기 계산된 상태 비용들로부터 미리 결정된 임계값을 감산하는,

프로세서.

청구항 8

제6항 또는 제7항에 있어서,

상기 임계치들은 2의 거듭제곱(power)으로 표현되고, 상기 감산은 상태 플래그에 기초하여 관련 비트를 0으로 마스킹(mask)함으로써 수행되고, 상기 상태 플래그는 CPU 레지스터에 저장될 수 있는,

프로세서.

청구항 9

제1항 내지 제7항 중 어느 한 항에 있어서,

상기 명령들은 비터비 엔진 내의 묵시적(implicit) 레지스터들의 로드(load)들 및 저장(store)들에 기초하여 자동으로 코어 레지스터 또는 상태 레지스터 내 플래그를 세팅하고,

상기 플래그는 비터비 동작이 부분적으로 완료되었는지의 여부를 표시하여, 상기 프로세서로 하여금, CPU 레지스터들을 저장/복원하는 경우 상기 비터비 엔진이 사용되는지의 여부를 결정하게 할 수 있는,

프로세서.

청구항 10

복수의 무선 로컬화 비컨들에 의해 수신되는 신호에 기초하여 지리적 위치를 결정하기 위한 GNSS 수신기로서,

로컬화(localization) 위성들을 포함하고,

상기 신호 내에 포함된 데이터의 일부는 컨볼루션 코드에 따라 인코딩되고,

상기 수신기는 제1항 내지 제7항 중 어느 한 항에 따른 프로세서를 포함하고, 상기 프로세서는 상기 컨볼루션 코드를 디코딩하기 위한 특수 명령들을 포함하는 소프트웨어 프로그램들을 실행하도록 구성되는,

GNSS 수신기.

청구항 11

복수의 무선 로컬화 비컨들에 의해 수신되는 신호에 기초하여 지리적 위치를 결정하기 위한 GNSS 수신기로서,

로컬화 위성들을 포함하고,

상기 신호 내에 포함된 데이터의 일부는 컨볼루션 코드에 따라 인코딩되고,

상기 수신기는 제1항 또는 제2항에 따른 프로세서를 포함하고, 상기 프로세서는 상기 컨볼루션 코드를 디코딩하기 위한 특수 명령들을 포함하는 소프트웨어 프로그램들을 실행하도록 구성되며,

상기 특수 명령들 중 적어도 일부는 비터비 디코딩 알고리즘의 가산-비교-선택 동작을 동시에 구현하기 위한 명

령을 포함하고,

상기 명령은 다수의 코어 데이터 레지스터들 상에서 동작하며, 묵시적으로 어드레스지정되는 부가적인 코어 CPU 레지스터들을 사용하는,

GNSS 수신기.

청구항 12

제11항에 있어서,

상기 명령은 상기 비터비 디코딩 알고리즘의 비터비 루프에 대한 최저 누적 비용 상태의 계산을 동시에 수행하는,

GNSS 수신기.

청구항 13

복수의 무선 로컬화 비컨들에 의해 수신되는 신호에 기초하여 지리적 위치를 결정하기 위한 GNSS 수신기로서,

로컬화 위성들을 포함하고,

상기 신호 내에 포함된 데이터의 일부는 컨볼루션 코드에 따라 인코딩되고,

상기 수신기는 제1항 또는 제2항에 따른 프로세서를 포함하고, 상기 프로세서는 상기 컨볼루션 코드를 디코딩하기 위한 특수 명령들을 포함하는 소프트웨어 프로그램들을 실행하도록 구성되며,

상기 명령은:

비터비 엔진에서 묵시적 레지스터들의 로드들 및 저장들에 기초하여 자동으로 코어 레지스터 또는 상태 레지스터 내 하나 이상의 플래그들을 세팅하는 것 – 상기 하나 이상의 플래그들은 비터비 동작이 부분적으로 완료되었는지의 여부를 표시함 – 을 포함하여, 상기 프로세서로 하여금, CPU 레지스터들을 저장/복원할 때 상기 비터비 엔진이 사용되는지의 여부를 결정할 수 있게 하는,

GNSS 수신기.

청구항 14

제10항에 있어서,

비터비 알고리즘의 역추적 상태 및 시스템 메모리에서의 데이터 구조의 어드레스에 기초하는 팬텀 레지스터를 포함하고, 상기 레지스터는 판독될 다음 32비트 역추적 히스토리 워드에 대한 정확한 어드레스를 리턴시키며,

상기 팬텀 레지스터는 간접 로드에 대한 어드레스 레지스터로서 CPU에 의해 사용될 수 있으며,

상기 어드레스는 현재 역추적 상태 및 역추적 샘플 인덱스의 함수인,

GNSS 수신기.

청구항 15

제10항에 있어서,

상기 컨볼루션 코드의 상기 디코딩은,

데이터 상태들 간의 트랜지션 비용들의 계산을 포함하고, 여기서 트랜지션은 0 내지 M의 범위에 있는 음이 아닌 값들로서 표현되고 – M은 K 단계들에서 획득될 수 있는 최대 비용이고, K는 상기 컨볼루션 인코딩된 데이터 내의 메모리 비트들의 수임 –,

누적(cumulative) 경로 비용 당 N개 비트들을 할당하는 것– (2^(K+1) * M)- 1 < 2^N-1임 – 을 포함하는,

GNSS 수신기.

명 세 서

기 술 분 야

[0001] 본 발명은 컨볼루션-인코딩된 데이터를 디코딩하기 위한 수신기를 포함하는 글로벌 네비게이션 위성 시스템(GNSS)용 수신기에 관한 것이다. 특히, 그러나 배타적이지 않게, 본 발명의 실시예들은 WAAS/EGNOS 위성들과 유사한 지구 정지 궤도(geostationary) 위성들에 의해 전송되는 증대(augmentation) 데이터를 디코딩할 수 있는 GPL 수신기, 및 GALILEO 표준에 따른 컨볼루션-인코딩된 데이터를 위한 GALILEO 수신기에 관한 것이다.

배 경 기 술

[0002] GPS에서, 정지 궤도(Geosynchronous) 위성들이 상이한 국가들에 의해 동작되는 기존의 위상 기반 증대 시스템(SBAS)은 휴대용 GPS 수신기 디바이스들로부터 이용가능한 GPS 위치 고정 품질 및 무결성을 증대시키기 위한 부가 정보를 제공하기 위해 사용된다.

[0003] 이러한 정보는 고정 품질을 개선하기 위해 적용될 수 있는 기압 정정(atmospheric correction)들에 대한 부가 정보, 및 위성 무결성에 대한 정보를 제공한다.

[0004] 더 큰 궤도 반경, 및 SBAS 위성들의 지구 정지 궤도로 인해, 더 복잡한 컨볼루션 인코딩이 데이터 전송을 위해 사용되어 왔으며, 이는 시스템 상에 증가된 프로세싱 로드를 부과하는 수신기, 통상적으로 비터비 디코더에서의 더 복잡한 디코딩 성능을 요구한다. SBAS 구현예들의 경우, 비터비 디코딩은 통상적으로 단일 SV에 적용되며, 범용 프로세서에 의해 실행되는 소프트웨어 디코더에 의해 그것을 구현하는 것이 알려져 있다. 이러한 해법은 단순하지만, 속도와 전력 절감(power economy)에 관심이 있는 한 차선책이다.

[0005] 또한, 제안되는 Galileo 시스템은, 동일한 기본 컨볼루션 코딩 기법을 이용하여, 그러나 가능하게는 상이한 생성기 다항식(generator polynomial) 및 심볼 레이트들을 사용하여, 성상도(constellation)에서 모든 SV들 상의 데이터 채널들에 대해 이러한 컨볼루션 인코딩 메커니즘을 사용할 것이다.

[0006] 이러한 새로운 네비게이션 신호 포맷의 영향은 프로세싱 오버헤드를 상당히 증가시킬 것인데, 왜냐하면 상기 해법에서 사용되는 모든 Galileo SV들이 비터비 디코딩 알고리즘이 동시에 동작될 것을 요구하므로, 시스템 자원들에 대한 비터비 디코더들의 로드가 매우 커지기 때문이다. 소프트웨어에서 이들 모두를 수행하는 것이 이론적으로는 가능하지만, 상기 프로세싱 오버헤드 및 추가 메모리 요건들은 GNSS 네비게이션 해법들에 추가 비용 및 전력 요건들을 부과할 것인데, 이는 작은 사이즈, 저전력, 및 저비용의 시장 목표들과 직접적으로 상충한다.

[0007] 이에 대한 일 해법은 프로세서로부터 완전히 프로세싱을 오프로드(offload) 시키기 위해 하드웨어 내에 비터비 디코더들을 구현하는 것이다. 이 수행 방법은 메인(main) 프로세서에 대한 로드를 감소시키지만, 비용 및 전력은 단순히 다른 서브시스템으로 전가된다.

[0008] 따라서, 시스템 자원들에 높은 로드를 부과하지 않고 컨볼루션 인코딩된 신호들을 디코딩할 수 있는 저전력 GNSS 수신기를 제공할 필요성이 존재한다. 더욱이 본 발명은 컨볼루션-인코딩 신호를 보다 신속하게 디코딩할 수 있으며 당해 분야에 알려진 수신기보다 더 낮은 전력을 사용하는 저전력 GNSS 수신기를 제공하는 것을 목적으로 한다.

발명의 내용

[0009] 본 발명의 목적들은 첨부된 청구항들의 대상에 의해 획득된다. 본 발명의 변경예들에서, 이러한 목적은 프로세서 내에 구현되는 특수 명령 세트에 기초하며, 하드웨어 복잡도를 크게 증가시키지 않고 상기 프로세서로 하여금 훨씬 더 낮은 CPU 로딩으로 비터비 프로세싱을 처리하게 할 수 있는 소프트웨어 비터비 디코더에 의해 달성된다.

[0010] 비터비 알고리즘의 분석 및 SV 네비게이션에 대해 특정한 적절한 설계 제약들의 주의깊은 애플리케이션에 의해, 비터비 가속화 로직을 GNSS 칩셋으로 효율적으로 포함시키기 위한 최적화된 아키텍처가 달성될 수 있다.

[0011] 그러나, 하드웨어 및 소프트웨어 간의 가까운 통합에 의해, 이는 시스템 자원들의 현저한 증가 없이도 종래의 RISC 프로세서의 명령 세트를 확장시킴으로써 달성될 수 있다.

도면의 간단한 설명

[0012] 도 1은 트렐리스(trellis) 상태도를 나타낸다.

도 2는 도 1의 다이어그램의 최저-비용 경로를 예시한다.

도 3은 해밍 거리(Hamming distance)의 계산을 나타낸다.

도 4는 비용 분포를 예시한다.

도 5는 누적 상태 비용 다이어그램을 나타낸다.

도 6은 비터비 알고리즘에서의 상이한 데이터 액세스 방법을 개략적으로 도시한다.

도 7은 본 발명의 실시예에 관한 명령 아키텍처를 예시한다.

도 8 및 9는 본 발명에 관한 명령의 묵시적 어드레스지정에 관한 것이다.

도 10은 전송된 비트들의 시퀀스를 예시한다.

도 11 및 12는 도 10의 비트들의 시퀀스의 디코딩을 나타낸다.

도 13은 본 발명의 양상들에 따른 특정 명령 세트를 사용하는 프로세서의 동작을 개략적으로 예시한다.

발명을 실시하기 위한 구체적인 내용

[0013] 비터비 인코딩 및 디코딩 알고리즘들은 일반적으로 당해 기술분야에 알려져 있으며 기술 문헌에서 설명된다. 이하에서 오직 GNSS 구현예로 특정된 양상들만이 논의될 것이다.

[0014] 몇몇 기준들, 즉 심볼 사이즈, 심볼 해석, 생성기 다항식 및 길이, 및 심볼 레이트가 비터비 구현예에서 고려되어야 한다. GNSS 애플리케이션에 대해, 제약 길이 K = 7(메모리 m = 6)이며 이는 전송된 심볼이 현재 데이터 비트 및 이전의 6개 데이터 비트들의 함수임을 의미하고, 코드 레이트는 2이며 이는 각각의 데이터 비트에 대해 2개의 심볼들이 전송됨을 의미하고, 심볼 레이트는 일반적으로 약 250 또는 500 심볼/초이고, 생성기 다항식은 변할 수 있다.

[0015] 비터비는 이전에 수신된 심볼들의 잠재적 해석들에 기초하여 특정 상태에서의 최대 우도비(maximum likelihood)를 계산하는 경로 비용 테이블 상에서 동작하는 컨볼루션 코딩 및 디코딩 기법이다. 다항식이 길이 7이므로 상기 구현예는 이전 6개 데이터 비트들을 나타내는 62개의 상태들을 요구한다.

[0016] 사용된 컨볼루션 코드는, 2개의 상태들 간의 트랜지션으로서 표현되고, 64개의 상태들(K=7)을 가지는, 전송된 각각의 데이터 비트에 대해 2개의 비트 심볼들을 생성하고, 각각의 상태로부터 2개의 잠재적 심볼들에 의해 표현되는 2개의 가능한 다음 상태들이 존재하며, 각각의 상태에 대해, 결코 전송될 수 없는 2개의 잠재적 심볼들이 존재한다. 데이터 비트에 대해 전송된 실제 심볼들은 생성기 다항식에 의해 결정되며, 이전의 6개 데이터 비트들 및 전송중인 비트의 함수이다.

[0017] 따라서, 각각의 데이터 상태에 대해, 심볼은 다항식에 의해 결정되고, 상태들은 다음과 같이 예측가능하다: 비트가 "1"인 경우 다음 상태는 32 + (현재 상태/2)이고, 그렇지 않은 경우 그것은 단순히 현재 상태/2이므로 어떠한 상태 트랜지션 테이블도 요구되지 않는다.

[0018] 각각의 심볼이 전송됨에 따라, 따라서 그 표현은 7개 데이터 비트들에 의해 결정되며, 그것이 수신될 때 다항식 및 이전 6개 데이터 비트들에 기초하여 디코더에 의해 해석된다. 각각의 심볼이 수신될 때, 그것은 해석되어 잠재적으로 유효한 심볼들에 대한 유사성에 기초하여 확률값을 받게 되며(award), 특정 심볼 시퀀스 해석이 정확할 확률을 계산하는데 사용된다.

[0019] 이를 달성하기 위해, 비터비 디코더는 모든 가능한 상태들에 대해 발견된 최저 비용 경로의 기록을 유지하고, 각각의 가능한 상태들에 대해, 그것은 수신 중인 심볼들의 이전에 가장 유사한 시퀀스 및 그들의 누적 비용을 기록한다.

[0020] 충분한 심볼들이 수신되었으면, 데이터는 최저 비용 상태를 선택(pick)하고 상기 상태를 초래하기 위해 가장 유사하게 전송되었던 데이터 비트들의 시퀀스를 결정하기 위해 그 히스토리를 통해 역으로 작용함으로써 디코딩된다.

[0021] 비터비 인코딩 및 디코딩의 원리들은 일반적으로 컨볼루션 코딩 메커니즘에 익숙한 사람들에 의해 이해되며, 사용가능한 분석은 이들 방식들에 의해 획득된 전체 SNR 개선들이 단순한 바이너리 결정이 아닌, 각각의 심볼에

잠재적 값들의 범위를 수여함으로써 이익을 얻으며, 통상적으로 상당한 이득들이 각각의 심볼에 대한 최대 8 레벨의 해석을 통해 달성될 수 있음을 보여준다.

[0022] K = 7에 대해, 보통의 비터비 디코딩 알고리즘들은 최적 성능을 위해 제 1 데이터 비트를 추출하기 전에 적어도 35개의 심볼들의 프로세싱을 요구하고, 이 프로세싱동안, 특정 상태에서의 누적 비용들을 나타내는 상태 정보의 적어도 2개 세트들이 유지될 필요가 있으며, 각각의 상태에 대해, 최적 디코딩 성능을 위해 상기 상태를 초래하는 통상적으로 이전의 34개의 트랜지션들(데이터 비트들)의 기록이 유지될 필요가 있다.

[0023] 이들 제약들이 주어지면, 우선 비터비 디코딩 상태들을 표현하는 적절한 최소 데이터 구조들을 선택하고 이후 포함된 구현을 위해 이들을 최적화시킬 수 있으며, HW/SW 상호작용 및 프로세서 서브시스템 아키텍처에 대한 특정 고려를 부여한다.

[0024] 각각의 잠재적 현재 상태에 있어서, 2개의 심볼 인코딩들에 의해 표현되는 2개의 유효 상태 트랜지션들(브랜치들)이 존재하며, 무효(invalid) 트랜지션들을 나타내는 2개의 심볼 인코딩들이 존재하며, 그 결과, 각각의 상태에 대해, 오직 2개의 잠재적 후임자(successor) 상태들 및 2개의 전임자(predecessor) 상태들만이 존재한다. 이들은 도 1의 트렐리스 다이어그램에 의해 표현될 수 있다.

[0025] 각각의 잠재적 현재 상태에 따라, 유효 및 무효 브랜치들이 상이한 심볼들에 의해 표시되고, 따라서, 각각의 상태에 대해, (자명하게(trivially) 계산될 수 있는) 2개의 잠재적 다음 상태들을 알 필요가 있으며, 적절한 심볼들은 전용 로직을 사용하여 계산될 수도 있는 브랜치들을 연관시키지만, 테이블로부터 더 용이하게 로딩되어 그 결과 다항식과는 독립적인 구현예를 생성할 수 있다.

[0026] 소프트 결정(연판정) 디코더를 구현하고 있으므로, 수신된 심볼들의 강도(strength)에 따라 가중된(weighted) 모든 잠재적 심볼 쌍들에 대한 비용 인자를 먼저 결정할 필요가 있는데, 모든 상태들에 대한 모든 유효 상태 트랜지션들이 동일한 확률을 가지므로, 이러한 가중치는 현재 상태와는 독립적이며 수신된 심볼 쌍 및 브랜치와 연관된 실제 심볼 쌍과 수신된 심볼 쌍 간의 해밍 거리에만 의존한다. 이는 각각의 심볼에 대해 소프트웨어가 해밍 거리와 연관된 가중치 테이블을 계산해야 함을 의미하며, 이는 이후 해밍 거리들 및 상태들 간의 매핑을 사용하는 모든 상태 및 브랜치 비용 계산들에 적용될 수 있다.

[0027] 따라서, 기본 비터비 알고리즘은 수신된 각각의 심볼 쌍에 대한 계산들의 후속 시퀀스를 수행할 것이며, 실제 구현예들은 연관된 데이터 구조들의 주의깊은 설계 및 적절한 커스텀(custom) 명령들의 구현예에 의해 로드들/저장들 및 복잡한 명령 시퀀스들을 최소화하도록 이들 프로세스들을 병렬화할 것이다.

[0028] 수신된 심볼 쌍에 대해, 심볼 쌍들 해석과 4개의 가능한 전송된 심볼 쌍들 간의 해밍 코드와 연관된 4개의 비용들의 세트가 계산된다. 실질적으로, 해밍 비용 테이블은 각각의 가능한 방식으로 실제 수신된 입력 심볼 쌍을 해석하는 것과 연관된 비용을 나타내는 4개 값들, 즉 00, 01, 10, 11의 테이블이다.

[0029] 각각의 잠재적인 현재 상태(64개 상태들)에 대해, 수신된 심볼 쌍의 2개의 잠재적 해석 각각에 대한 다음 상태들 각각에 도달하는 비용들(브랜치 비용)이 계산되며, 이는 수신된 심볼에 대한 사전-계산된 비용 테이블 및 상태들의 현재 비용에 기초하는데, 즉, 각각의 상태에 대해, 트랜지션과 연관되는 심볼로서 요구되는 심볼 쌍 및 그것의 재-해석에 기초하여 연관된 비용들을 가지는 2개의 트랜지션들 및 2개의 잠재적 다음 상태들이 존재한다.

[0030] 후임자 상태들 각각에 대해, 최저 비용을 가지는 상태로의 트랜지션을 선택하고 상기 상태에 도달하는 비용 및 전송된 데이터 비트를 나타내는 거기에 도착한 트랜지션("1" 또는 "0")을 레코딩하기 위해 상기 상태에 대한 추적 히스토리를 업데이트한다.

[0031] 그 결과는 도 2에 도시된 바와 같이 각각의 상태에 도달하기 위한 경로 및 각각의 상태에 대해 달성가능한 최저 비용의 테이블이다.

[0032] 마지막으로, 충분한 심볼들이 디코딩되었으면, 최저 누적 비용들을 가지는 상태를 선택하고 각각의 상태로의 트랜지션들을 통해 역추적하여, 해당 상태에 도달한 최저 비용 데이터 시퀀스를 복원하는데, 이것이 수신된 비트 스트림의 비터비 디코딩이다.

[0033] 이것으로부터, 비터비 디코딩이 데이터 집약형(intensive) 동작이며, 또한 데이터 액세스 순서가 정해졌을 때(drawn) 형성되는 패턴으로 인해 일반적으로 비터비 버터플라이(Viterbi butterfly)라고 지칭되는 비 순차적인

순서로 데이터를 액세스 하는 것을 포함한다는 점이 명백하다.

[0034] 일반적으로, 비터비 가속화는 효율성을 위해 다수의 메모리들, 및 공지된 가산-비교-선택 동작을 수행하는 전용 하드웨어를 사용하는 비터비 버터플라이 동작을 사용하여 수행될 수 있다. 가산-비교-선택 동작은 용이하게 가속될 수 있으며 종종 DSP에서 커스텀 명령들에 의해 지원되지만, 이를 단독으로 가속하는 것은 속도를 많이 향상시키지는 않는데, 왜냐하면, 상기 동작이 데이터 액세스에 우세해지며, 일반적으로, 유효(effective) 어드레스 계산이 데이터 액세스 자체만큼이나 많은 시간을 소요하기 때문이다.

[0035] 통상적으로 DSP들은 어드레스들의 계산을 위한 커스텀 명령들 및 경로 비용 계산들을 위한 가산-비교-선택 명령들을 가진다.

[0036] 최적화 및 데이터 표현

[0037] 컨볼루션 알고리즘의 속성으로 인해, 상태 트랜지션들이 예측가능하며, 부분적으로 독립적인 데이터 세트들로 그룹화될 수 있는데, 이는 유도될 수 있었던 정확히 4개의 현재 상태들이 존재하는 잠재적 후임자 상태들의 각각의 쌍에 대한 것이며, 따라서, 메모리 액세스를 위한 최적의 아키텍처는 메모리 액세스들을 최소화하기 위해 모든 다른 데이터 세트들과는 독립적으로 이 데이터 세트 상에서 동작할 수 있어야 한다.

[0038] 각각의 잠재적인 후임자 상태를 유도하는 2개의 잠재적인 현재 상태들이 존재하고 임의의 현재 상태와 연관된 2개의 잠재적 심볼들이 존재하므로, 그 결과, 계산될 최대 128개의 브랜치들이 존재한다. 각각의 심볼 쌍에 대해, 4개의 가능한 해석들이 존재하며, 따라서, 각각의 잠재적 브랜치와 연관된 2개 비트들은 대응하는 심볼 쌍을 표시하고, 그 결과 1개 워드 내에 8개의 상태들에 대한 브랜치 비용 매핑(16 브랜치들)을 저장할 수 있다.

[0039] 임의의 브랜치에 대해, 4개의 가능한 심볼들이 존재하고, 각각의 상태에 대해, 2개의 가능한 브랜치들이 존재하며, 그 결과 상기 브랜치들은 적어도 1개 심볼만큼 차이가 나야 하는데, 왜냐하면 최대 심볼 비용이 결과적으로 14이기 때문이다. 채널이 완벽하지 않아서, 불가피하게 심볼 에러들이 존재할 것이므로(컨볼루션 코드가 사용되는 이유), 사용되는 블록 길이에 의해 결정되는 최대값을 가지는 시간상에서의 누적 에러들이 존재할 것이다.

[0040] 전송된 심볼과 수신된 심볼 간의 최대 차이가 14이므로, 모든 심볼들에 대해, N개 데이터 비트들 다음의 최저 비용 상태에 대한 최대값은, 사용되는 다항식에 따라, 8N 미만일 것이다.

[0041] 가능한 4개의 조합들로부터 전송되는 2개 심볼 비트들이 존재하는 주어진 상태에서 전송되는 주어진 데이터 비트에 대해, 그 원리가 소프트 결정으로 확장될 수 있지만, 이제 이들이 바이너리 표현이라고 가정하자. 수신된 심볼을 전송된 심볼과 비교하고 각각의 가능한 해석에 대해 차이들을 카운팅하면, 해밍 거리가 계산되는데, 정확한 해석에 대해서는 0, 심볼 비트들 중 하나가 부정확하면 1, 두 비트들 모두가 부정확하면 2이다. 도 3이 이를 도시한다.

[0042] 각각의 가능한 상태에 대해, 각각의 가능한 후임자 상태에서의 잠재적인 비용을 계산하고 최저 비용을 유지하며, 시간상으로, 가능한 상태 비용들의 범위는 신호 및 가능한 해석들에서의 에러들로 인해 확장하며, 이들 중 하나는 정의에 의해 항상 잘못된 것이다(wrong).

[0043] 완벽한 신호를 가진다고 가정하면, 도 4에 도시된 바와 같은 비용 분포 변경을 알 것이다.

[0044] 그러나, 디코더에서 유한 개수의 메모리 비트들만이 존재하므로, K 비트가 존재하는 경우, 심지어 그것이 모든 수신된 심볼을 잘못 해석하는 것을 의미할지라도, 모든 상태가 모든 이전 상태로부터 K개 단계들로 도달가능하게 되며, 이는 도 5의 다이어그램에 도시된 바와 같은 누적 비용들을 구성하게 한다.

[0045] 간략함을 위해, 오직 2개 메모리 비트들만이 존재한다고 가정하면, 그 결과 좌측 삼각부는 K개 단계들에서 획득될 수 있고 K는 메모리 비트들의 수인 최대 비용이 이 경우 M=4임을 보여주는 시간 T+2에서 현재 상태 S0로부터 유도된 임의의 상태 S1에 대한 비용들의 최대값 분포를 도시한다. 신호 상태 트랜지션 다이어그램에서, 모든 심볼이 적어도 하나의 에러(H=I)를 가지도록 상태 T1 다음에 잡음의 존재를 표시하였으며, 이후 2비트 메모리의 제약들을 적용하고, 이전 2개 심볼들의 가능한 최악의 해석에 기초하여 모든 상태들에 대한 비용들의 잠재적 범위를 마스킹하였다.

[0046] T4에서 모든 비용들은 M=4의 최대 상태 비용 미만이 아니며, 따라서, T5에서, 공통 오프셋을 감산함으로써 모든

새로운 비용들을 조정할 수 있으며, 그 결과 비용 결과들의 범위를 제약할 수 있다. 이에 의해, 본 발명의 방법은 범위 0 내지 M 내의 음이 아닌 값들로 트랜지션 비용들을 세팅하는 단계를 포함하며, 여기서 M은 K개 단계들에서 얻어질 수 있는 최대 비용이며, K는 컨볼루션 인코딩된 데이터 내의 메모리 비트들의 수이다. 바람직하게는, N 비트들은 누적 경로 비용마다 할당되며, 여기서 (2^(K+1) * M)- 1 < 2^N-1이다.

[0047] 임의의 상태 Sn에 대해, 전임자 상태들은 S(2n) 및 S(2n+1)이다. Hn0/1를 상태 n에서 각각 0/1로서 심볼을 해석하는 해밍 비용들이라 하자. C(N)은 상태 N의 최저 누적 경로 비용을 나타낸다고 하자. KT는 시간 T에서, C0..C63의 세트를 나타낸다고 하자.

[0048] 시간 T0에서 상태 비용들의 세트 KT0를 가지고, 이후 시간 T1에서, 관련 비용들 KT1을 가진다고 가정하자. 이는 Min(KT1) >= Min(KT0) 및 Max(KT1) <= Min(KT0) + 84를 따른다. 데이터 복원을 시작하기 위해, KT의 실제 값들이 아닌 KTn의 최소 멤버의 인덱스를 고려해야 하는데, 이는 비용들 KT를 Min(KT)보다 작거나 같은 일부 값으로 정규화시킴으로써 이들을 제약할 수 있다는 점을 따른다.

[0049] 따라서, 누적된 상태 트랜지션 비용들을 나타내기 위해 7비트가 필요하므로, CPU 아키텍처를 가지는 정렬을 위해, 누적된 비용들을 8비트 값들로서 저장하는 것이 편리한데, 이는 심볼 쌍의 32비트 워드로의 4가지 가능한 해석에 대한 비용들을 패킹하게 한다.

[0050] 따라서, 누적 비용들의 오버플로우를 회피하기 위해, KT 내의 모든 값들을 고유하게 표현하고 이들의 순서(ordering)를 보존하기 위해 충분한 범위를 가진다고 가정하면, 임의의 상태가 오버플로우에 근접할 때 상기 누적 비용들을 스케일링할 필요가 있다. 이를 달성하기 위해, 각각의 누적된 비용은 8비트 값들로서 표현될 것이며, 비용들이 모든 상태 트랜지션들에 대해 누적됨에 따라 이들은 84를 초과하여 분기(diverge)할 수 없는데, 이는 비용 조정이 256 미만 사이에 모든 값들이 있도록 제약함으로써 수행되게 할 수 있고, 이는 새로운 트랜지션 비용의 누적 이전에 최저 상태 비용이 128에 도달하는 경우 모든 상태 비용들의 MSB를 리셋함으로써 용이하게 달성된다.

[0051] 이제, 상기 데이터 표현은 메모리 액세스들의 오버헤드를 최소화하도록 데이터의 프로세싱을 최적화하는 것이 바람직할 수 있다고 결정한다.

[0052] 임의의 상태에 대한 브랜치 비용들의 계산은 Cn = 이 되게 하는 것이 아니라 2개 경로들의 잠재적 비용들의 계산을 포함한다.

[0053] 각각의 후임자 상태로의 2개의 잠재적 브랜치들이 존재하므로, 이들 트랜지션들을 동시에 계산하여 최저 비용 브랜치가 중간 값들을 계산할 필요 없이 검출되고 저장될 수 있는 것이 바람직하다.

[0054] 비터비는 다음 반복을 위한 소스 데이터로서 그 결과들을 사용하여 동작하며, 결과적으로 오퍼랜드들로서 동일한 순서로 결과를 정렬할 필요가 있는데, 이는 64바이트(16개의 32비트 워드들)을 점유하는 상태 비용들의 선형 어레이를 내포한다.

[0055] 4개의 상태 비용들을 1개 워드에 저장할 수 있으므로, 폐기하기 전에 상기 워드를 판독하고 상기 워드 내의 모든 상태들을 처리하며, 입력과 출력인 구조의 2개 복사본을 사용하고 교번시켜서 동일한 타입의 구조로 상기 결과들을 직접적으로 기록하는 것이 바람직할 수 있다.

[0056] 이러한 구조를 사용하여, 처리된 각각의 워드는 4개의 새로운 상태들을 생성하고, 상기 4개의 새로운 상태들은 출력 배열내의 동일한 워드로 피팅하는 2개 그룹들 내에 존재하지만, 2개의 연속적인 소스 워드들의 프로세싱은 각각의 그룹이 단일 워드로 피팅하는 2개 그룹들 내에 8개의 결과들을 생성할 것이지만, 2개 워드들은 인접하지 않는다.

[0057] 따라서, 비터비 동작을 위해, 2개 워드들의 8개 그룹들로서 8개 상태들을 처리할 필요가 있으며, 각각의 상태가 매핑 레지스터 내의 4비트를 요구하므로, 매핑 레지스터를 패킹할 수 있으며, 따라서, 각각의 그룹 역시 이러한 매핑을 포함하기 위해 1개의 32비트 레지스터를 요구한다.

[0058] 이는 루프 당 8개 상태들을 처리하는 8회 반복 루프에 잘 피팅되며, 여기서 요구되는 모든 데이터는 CPU 레지스터를 채우며, 각각의 오퍼랜드는 루프 당 한번 판독되며, 각각의 결과는 어떤 부분적 워드 업데이트들도 요구되지 않는 경우 1로 기록되며, 따라서, 메모리 효율성이 최적화된다.

[0059] 각각의 새로운 상태에 대해, 최저 비용 브랜치가 레코딩되어야 하는데, 이는 32비트 레지스터들의 쌍으로 잘 피

팅될 64비트들을 요구하며, 이들은 각각의 루프 반복동안 부분적으로 업데이트되고 상기 루프의 완료 이후 히스토리 또는 '역추적' 테이블에 저장될 필요가 있을 것이다.

[0060] 또한, 최저 비용 상태의 인덱스를 계산하고, 언제 모든 상태들이 최소 임계를 초과하는지를 검출하여 다음 루프 반복 상에서 이들을 조정할 수 있도록 할 필요가 있는데, 이는 모든 상태들이 128 초과임을 표시하기 위해 플래그를 사용하고 다음 반복에서 MSB를 마스킹함으로써 용이하게 달성된다.

[0061] 루프 반복 동안, 소스 및 결과 데이터를 효율적으로 어드레스지정(address)할 필요가 있지만, 상기 어드레스지정은 루프 인덱스의 간단한 함수이며, 필드가 역추적 레지스터들 내에서 업데이트할 것임에 따라, 이상적인 확장은 루프 인덱스, 어드레스지정, 최저 비용 상태 인덱스, 및 정규화 플래그를 위한 레지스터들을 포함할 것이다. 도 6은 2개 소스 값들(56)이 하나의 결과(66)를 생성하기 위해 사용되는 다이어그램(61)에서 비터비 루프를 위해 액세스될 수 있는 상이한 방법들을 도시하는데, 이는 모든 소스 데이터를 이용하지 않으므로 불충분하다. 다이어그램(62)에 예시된 경우에 있어서, 하나의 소스 오퍼랜드(65)는 2개 비용들(66)을 계산하기 위해 사용되지만, 이들은 상이한 워드들 내에서의 상이한 상태들과 관련되며, 제3의 다이어그램(63)에서, 2개 워드들 내의 8개의 상태들(65)은 2개 워드들 내의 8개의 새로운 상태들(66)을 계산하기 위해 이용되며, 모든 데이터는 완벽하게 정렬되고 어떤 데이터도 낭비되지 않는다.

[0062] 전체적으로, 이는 아래 도 7에 도시되는 바와 같이, 통상적인 명령 세트 아키텍처를 사용하여 용이하게 구현될 수 없는 다수의 소스 및 목적지 레지스터들을 요구하는 이상적인 구현예를 도출한다.

[0063] 도 7은 4개 소스 변수들로부터 3개 결과들을 계산하기 위한 동작들의 통상적인 시퀀스를 예시한다. 상위 다이어그램은 이들이 통상적으로 표준 3 오퍼랜드 CPU 상에서 어떻게 구현되는지를 도시하고, 여기서 3개의 오퍼랜드들은 2개의 소스 오퍼랜드들 및 1개의 목적지 오퍼랜드를 나타낼 것이다. 여기서, 상기 동작은 적어도 3개의 순차적 명령들로 분할되어야 한다. 이는 충분한 레지스터들의 인코딩을 위한 충분한 비트들을 허용하지 않는 명령 오퍼랜드 포맷 제한들로 인한 것이며, 또한, 일반적으로, 사이클 내 둘 이상의 기록 동작을 허용하기 위한 레지스터 파일 능력에 대한 제한일 것이다. 이러한 인코딩에 있어서, 모든 레지스터들은 각각의 op코드 내에서 명시적으로 인코딩되며, 모든 레지스터는 범용 레지스터 파일에 존재한다.

[0064] 하위 부분에서, 동일한 동작이 레지스터들의 일부의 묵시적 인코딩, 및 범용 레지스터 파일과는 별개인 전용 레지스터들로서의 이들 특정 필드들의 구현예를 사용하여 예시된다.

[0065] 도 8 및 9는 도 3의 최적 방법이 명령들에서의 레지스터 어드레스지정을 묵시적이 되게 함으로써 얻어지는 본 발명의 양상에 관한 것이다.

[0066] 도 9의 다이어그램은 비터비 명령들에 의해 구현되는 실제 기능을 도시한다. 이 도면에서, 이전 상태들의 쌍(Sa, Sb)은 새로운 상태 Sc에 대한 잠재적인 전임자 상태들의 쌍에 대응하여 선택된다. 새로운 상태를 도출하는 이러한 방식으로 새로운 심볼을 해석하는 대응 비용들은 멀티플렉서(901)에 의해 선택되고 가산기들(902)에 의해 이들 이전 상태들의 대응 비용들에 가산되어(903) 상태 Sc에 도달하는 잠재적인 비용들의 쌍을 생성한다. 이들 비용들은 비교되고 선택기(904)에 의해 최저 비용 전임자 상태가 선택되며, 대응 비트 트랜지션이 표시되고(905), 대응 비용이 출력된다(906). 이러한 대응 비용들은 각각의 새로운 심볼이 적용됨에 따라 각각의 상태에 대해 잠재적으로 증가할 것이지만, 상태 메모리가 유한 개수의 상태들(이 경우 6개 상태들)로 제한됨에 따라, 모든 상태들의 비용들에 대응하는 임의의 경우에서의 값들의 세트가 존재하는데, 여기서 최대값과 최소값 간의 범위는 최대 상태 트랜지션 비용들 및 최대 상태 메모리에 의해 제한된다. 이 예에서, 이는 상태 비용들이 무한히 증가하는 것을 방지하기 위해 64 미만이며, 표시는 임의의 상태 비용이 128인 경우 출력된다(907).

[0067] 이전 상태 비용들 중 적어도 하나가 128보다 컸다는 표시(908)가 정규화 블록(909)으로 공급되고, 이는 64를 감산함으로써 상태 비용을 정규화하는데, 이는 모든 상태 비용들이 유한 정수 범위 내에 유지될 것임을 보장한다.

[0068] 결과적인 새로운 상태 비용(910)은 명령으로부터 출력되며 또한 최소 비용 블록(911)에 의해 사용된다. 이 블록의 목적은 새로운 심볼 프로세싱 동안 계산된 모든 상태 비용을 모니터링하고, 새로운 상태 비용들의 모든 결과 세트의 최저 비용들을 결정하는 것이다.

[0069] 상태를 도출했던 트랜지션이 결정되면, 상기 상태에 대한 새로운 비용이 계산되고 상태 비용 테이블이 업데이트된다.

[0070] 새로운 심볼이 공급되고, 상기 심볼을 1 또는 0으로서 해석하는 연관된 비용들 각각이 멀티플렉서들(901)에 의해 선택되고, 이들 비용들은 심볼의 2개 가능한 해석들과 연관된 비용들의 쌍을 생성하는(903) 가산기들(902)에

서 이전 심볼 비용들에 가산된다.

[0071] 상기 결과들 및 제어 변수들 중 몇몇이 길이에 있어서 32비트 미만이므로, 이들은 단일 32비트 코어 레지스터의 서브 필드들로 병합될 수 있다.

[0072] 다수의 선제(pre-emptive) 작업들에서 명령 세트 확장의 동작을 인에이블시키기 위해, 이들 레지스터들 및 임의의 상태를 저장(save)하는 것 역시 필요한데, 이는 이들 상태 플래그들이 동일한 제어 레지스터로 병합되는 경우 더 간략화된다.

[0073] 충분한 심볼들이 프로세싱되었으면, 현재 상태를 도출하는 브랜치들의 시퀀스를 결정하기 위한 역추적 동작을 수행하는 것이 필요하다. 이는 비교적 간단한 동작이지만, 다시 상당한 어드레스 계산, 비트 조작, 및 데이터 로드 동작들을 포함한다. 또한, 이들은 각각의 역추적 단계를 수행하기 위한 전용 명령 및 역추적 히스토리의 어드레스지정을 위한 전용 레지스터들을 사용하여 효율적으로 구현될 수 있다.

[0074] 도 10의 트렐리스 다이어그램에서 전송된 비트들의 시퀀스를 고려하자. 이러한 인코딩에서, Rx 및 Ry는 범용 레지스터 파일의 외부에 있는 특수 레지스터들 내에 존재하는 4개의 요구되는 소스 오퍼랜드들 중 2개를 나타내고, Rp, Rq는 레지스터 파일의 외부에 있는 특수 레지스터들에 저장되며, 따라서 레지스터의 서브필드들이며 Rx 및 Ry와 오버랩할 수 있는 결과들 중 2개를 나타낸다.

[0075] 데이터 전송 후, 수신기는 비용들 및 역추적을 계산하고, 도 2의 비용 및 역추적 히스토리를 달성한다.

[0076] 역추적이 T=6에서 시작하고 상태 10이 최저 비용 상태임을 발견하면, 수신기는 시퀀스 101101를 디코딩하는데, 데이터가 전송되었던 것과 역순서이며 이후 도 11에서와는 역으로 진행될(reverse) 필요가 있다.

[0077] 또다른 심볼을 수신하고 상태 "10"을 다시 가정하는 최저 비용 경로로부터 다시 시작한 이후, 이제 수신기는 도 12에 도시된 바와 같이 1000101을 디코딩하는데, 명백하게 수개의 심볼들 다음에 경로들이 수렴하며, 상태 T=3보다 더 지나서 역으로 추적할 필요는 없었다.

[0078] 역추적 동안, 1비트만큼 오프셋되어 떨어져서 각각의 새로운 심볼이 추가된 후 상기 경로가 종종 현저하게 변하지는 않을 것이며, 또한 다시 분기되지 않을 이전 히스토리 내의 상태에서 새로운 역추적 히스토리가 수렴하면 결과적으로 이전 역추적 동작과의 컨버전스를 검출함으로써 역추적에 드는 노력을 감소시킬수 있다는 점 역시 명백하다. SW에서, 이는 비교 이전에 비트 필드들의 마스킹 및 로테이션을 수반하는 복잡한 작업이며, 이것의 오버헤드는 그것의 이점들이 부정되는 각각의 반복에 충분한 추가 사이클들을 부가하지만, 필드를 선택하기 위해 역추적 인덱스를 사용하여 이들 비트 필드에 액세스하도록 전용 하드웨어를 사용하는 것은 그것이 투명하게 수행될 수 있으며, 역추적 동작에 의해 소모되는 전력 및 사이클들을 크게 감소시킬 수 있음을 의미한다.

[0079] 역추적이 수행되면, 가장 오래된 데이터 비트들은 한번에 보통 1 바이트씩 추출되지만, 초기 단계(phase)동안 통상적으로 동기화 패턴이 탐색되고 비트 정렬은 알려지지 않으며, 이는 추가적으로 SW에 대한 초기 역추적 동작을 복잡하게 하지만, 또한 CPU 레지스터로 정렬된 올바른 순서로 가장 오래된 비트들을 추출하기 위한 특정 명령들, 및 사용되었을 때 다수의 비트들을 "사용"하기 위한 다른 별개의(separate) 명령을 추가함으로써 최적화될 수 있으며, 이러한 방식으로, 바이트들이 추출되고 비교되어 동기화가 검출될 때까지 한번에 1비트씩 폐기하며, 이후 애플리케이션에 의해 처리되는 레이트로 비트들을 사용한다.

[0080] 마지막 기능으로서, 비터비 프로세스 각각은 비터비 상태, 역추적 히스토리, 역추적 테이블 및 상태 비용 테이블들을 포함하는 데이터 구조를 요구한다. 이들은 각각의 채널에 대해 분리될 필요가 있으며, 따라서 모든 어드레스들은 얼마간의 오프셋을 필요로 할 것이다. 이는 다시 비터비 제어/상태 워드의 일부분으로서 비터비 데이터 구조 베이스 어드레스를 구현함으로써 프로세서로부터 숨겨질(hidden) 수 있다.

[0081] 다른 프로세스들에 대한 영향을 최소화하기 위해, 묵시적 비터비 레지스터들이 코어 cpu 레지스터들에 추가하여 존재하는 것 및 SW가 상황 스위칭시에 그들의 상태를 효율적으로 저장할 수 있는 것이 바람직하지만, 대부분의 프로세스들은 이들 레지스터들을 사용하지 않을 것이며, 따라서, 이들을 저장할 필요가 없다.

[0082] 저장되는 횟수를 최소화하기 위해, OS가 각각의 프로세스가 그들이 요구되는지 여부를 알 필요 없이, 추가적인 상태 비트들은 언제 비터비 블록이 사용되는지를 표시하기 위해 비터비 상태/제어 레지스터에 추가될 수 있으며, 이들은 프로세스의 상태에 따라 비터비 명령들에 의해 자동으로 세팅/클리어될 수 있고, 이러한 방식으로, 상황 스위칭은 저장될 임의의 상황이 존재하는지 여부를 결정하기 위해 이들 플래그들을 테스트할 수 있으며, 이러한 방식으로 이들은 그 기능을 사용하지 않는 프로세스들로부터 중요한 스택 공간을 취하지 않으며, 이들은 이들이 활성적으로 사용되지 않는 경우 저장되지 않아서 비터비가 멀티-스레드 환경에서 효율적으로 사용

될 수 있다.

[0083] 본 발명의 또다른 양상에 따라, 본 발명은 전술된 본 발명의 방법에 따라 컨볼루션 인코딩된 데이터의 디코딩에 최적화된 프로세서에 관한 것이다. 상기 프로세서는 물리적으로 임의의 레지스터 저장소를 가지지는 않지만, 현재 상태 비용 세트 할당 및 데이터 구조들에 대한 베이스 어드레스를 표시하고 현재 및 다음 상태 테이블들의 매핑을 선택하는 상태 레지스터 필드 및 루프 제어 레지스터로서 사용되는 코어 레지스터 필드에 기초하여 소스 및 목적지 상태 비용 오퍼랜드들에 대한 어드레스들을 리턴시키는 팬텀(phantom) 레지스터들로 프로세서의 레지스터 세트를 확장시키도록 구성된다.

[0084] 도 13을 참조하면, 본 발명의 프로세서의 동작은 다수의 필드들을 포함하는 단일 워드들(또는 op코드들)의 형태로 메모리에 저장되는 명령에 의해 제어된다. 예시된 예에 있어서, 예를 들어, op코드(147)는 2개의 소스 오퍼랜드들(148), 명령 필드(150) 및 목적지 레지스터 어드레스(149)로 구성되는 통상적인 3 오퍼랜드 명령을 예시한다.

[0085] 오퍼랜드들(OP1, OP2)은, ALU(146)로 전달되며, 멀티플렉서(142)에 의해 레지스터 파일(141)로부터 2개 소스 오퍼랜드들을 선택하도록 사용된다. ALU의 출력은 통상적으로 도시되지 않은 경로를 통해 어드레스(149)를 사용하여 레지스터 파일(141)로 역으로 기록된다(write back).

[0086] 예시된 예에서, 레지스터 파일(141)은 32개의 위치들 또는 레지스터들(R0-R31)을 포함하지만, 이는 본 발명에 대한 제한이 아니다. 또한, 도 14가 프로세서에서의 논리적 데이터 흐름을 예시하는 것을 의미하는 간략화된 기능도이며, 본 발명의 프로세서의 구조를 제한하지 않음이 이해되어야 한다. 특히, 본 발명은 또한 이들 동작들이 상이한 시간들에서 수행되는 파이프라인화된 아키텍처들을 포함한다.

[0087] 본 발명에 따라, op코드(147)는 레지스터 파일(141) 내의 별도의 저장 공간에 대응하는 것이 아니라, 레지스터들(R0-R31) 내의 값들 및/또는 CPU 내부 또는 외부의 추가적인 값들(151, 152)의 논리 함수인 "팬텀 레지스터들"(143)에 대한 어드레스들을 포함할 수 있다. 명령에 의해 어드레스지정될 때, 팬텀 레지스터 또는 레지스터들은 명령에 의해 명시적으로 어드레스지정되지 않는 레지스터들 및/또는 신호들로부터 계산되는 값을 리턴시킨다.

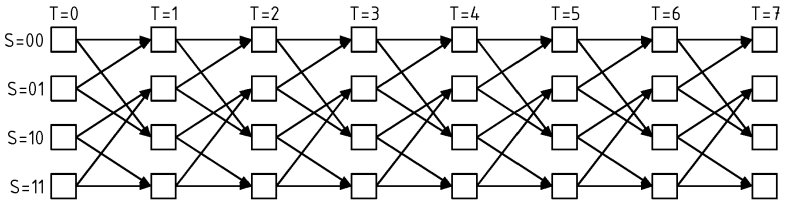
[0088] op코드(147)는 각각 6비트를 가지는 오퍼랜드 어드레스들(148)을 저장하며, 따라서 이 예에서, 32개의 물리 레지스터들보다 더 많은 오퍼랜드들을 어드레스지정하는 능력을 제공한다. 팬텀 레지스터들은, 예를 들어, 레지스터 파일(141) 내에 어떠한 대응부(counterpart)도 가지지 않는 어드레스들 중 하나를 인코딩함으로써 액세스 가능하다.

[0089] 도 14의 다이어그램이 산술 명령의 처리 및 통상적인 포맷을 도시하지만, CPU는 통상적으로 또한 비-산술 명령, 예를 들어, 로드/저장 명령을 가질 수도 있다. 본 발명에 따라, 이들 명령들의 오퍼랜드 역시 팬텀 레지스터(143)일 수 있다. 또한, 본 발명이 복수의 팬텀 레지스터들을 포함할 수 있으며, 이들이 예를 들어, 특히 다른 팬텀 레지스터들의 값으로부터 의존하는 일부 팬텀 레지스터들을 사용하여 서로 상호작용할 수 있도록 의도된다. 본 발명은 또한, 일부 팬텀 레지스터들이 예를 들어, 추가적인 숨겨진 레지스터들과 같은 상태를 유지할 수 있는 독립적인 메모리 엘리먼트들과 관련되는 경우를 포함한다.

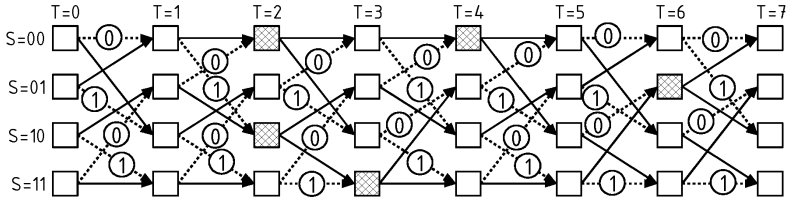
[0090] 유리하게는, 팬텀 레지스터들은 데이터 구조들에 대한 베이스 어드레스 및 현재 상태 비용 세트 할당을 표시하고 현재 및 다음 상태 테이블의 매핑을 선택하는 상태 레지스터 필드 및 루프 제어 레지스터로서 사용되는 코어 레지스터 필드에 기초하여 소스 및 목적지 상태 비용 오퍼랜드들에 대한 어드레스들을 리턴시키도록 구성된다.

도면

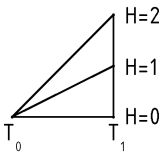
도면1



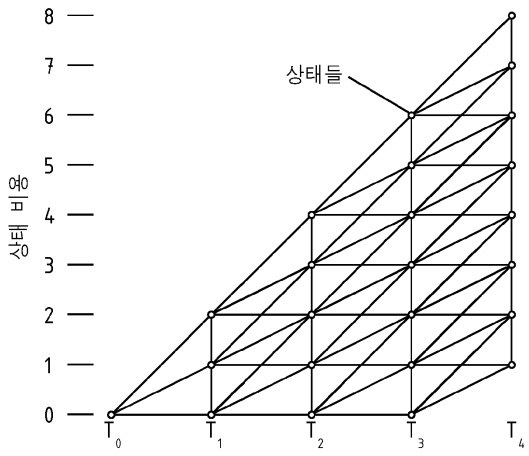
도면2



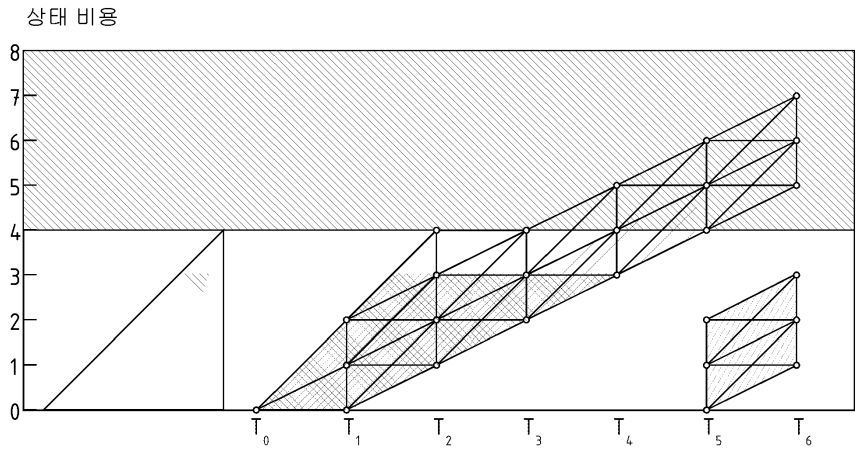
도면3



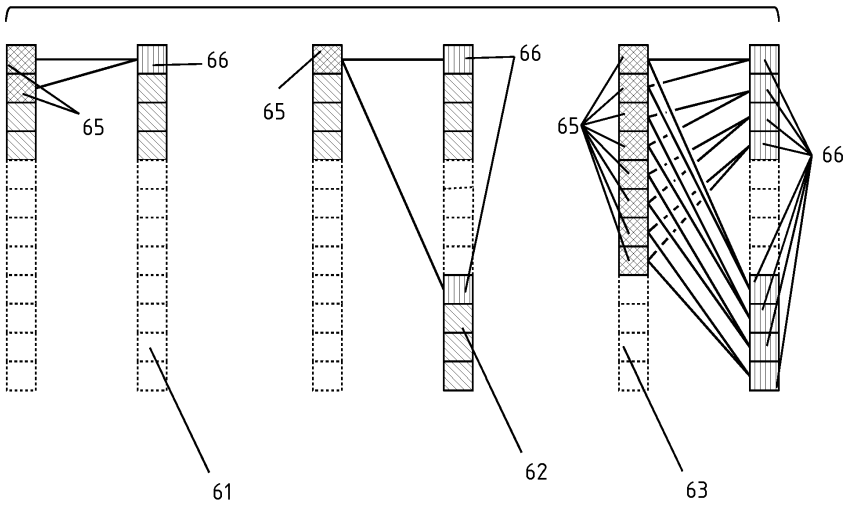
도면4



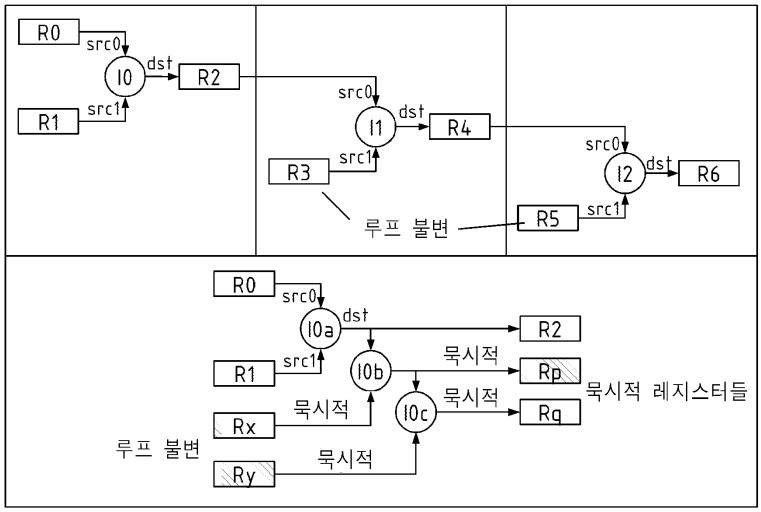
도면5


상태 비용
8
7
6
5
4
3
2
1
0
T0
T1
T2
T3
T4
T5
T6

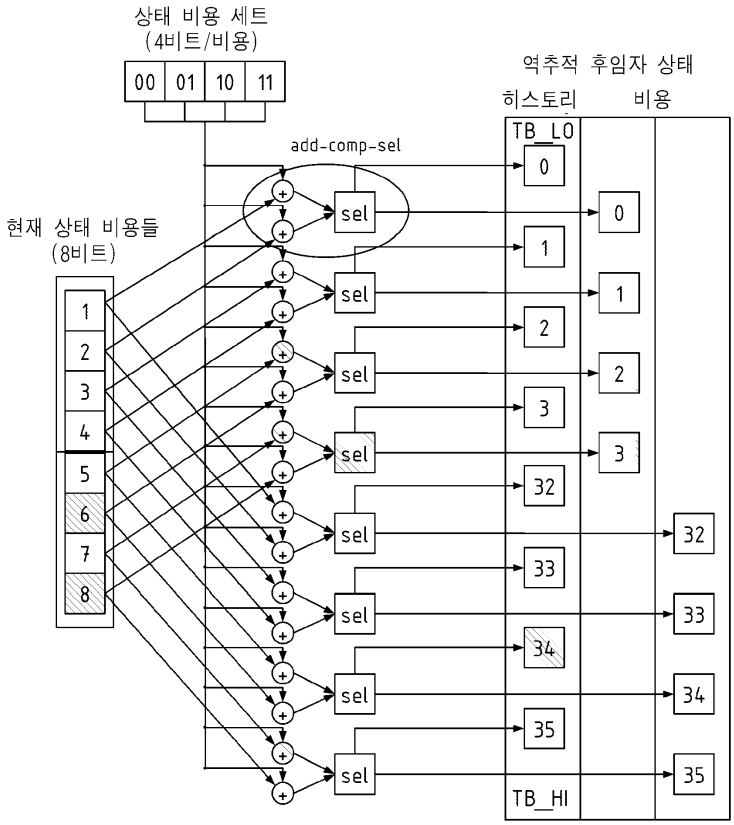
도면6


66
65
66
65
65
66
61
62
63

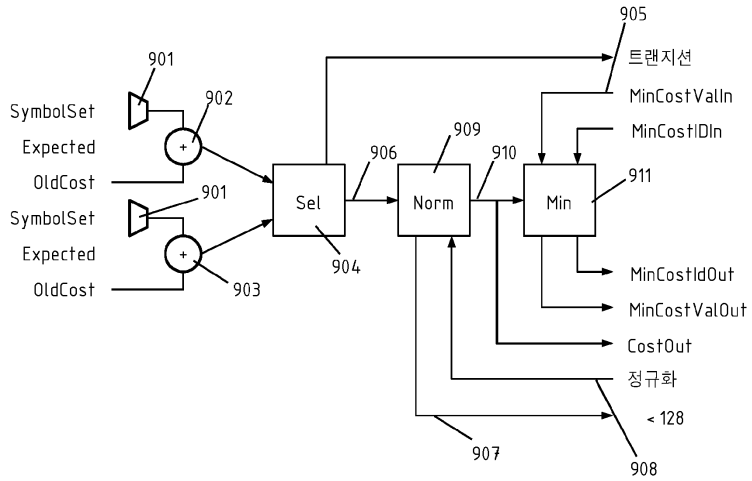
도면7


R0
src0
I0
dst
R2
R1
src1
src0
I1
dst
R4
R3
src1
src0
I2
dst
R6
R5
src1
루프 불변
R0
src0
I0a
dst
R2
R1
src1
I0b
묵시적
Rp
묵시적 레지스터들
묵시적
I0c
Rx
묵시적
Rq
루프 불변
Ry
묵시적

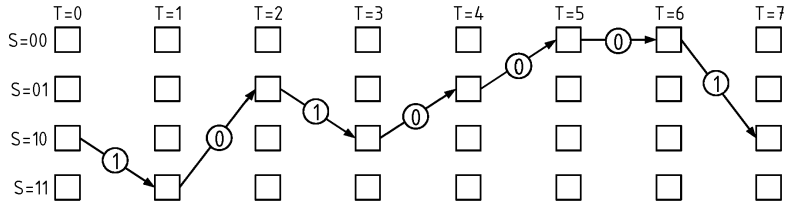
도면8


상태 비용 세트
(4비트/비용)
00
01
10
11
역추적 후임자 상태
히스토리
비용
TB_LO
TB_HI
add-comp-sel
현재 상태 비용들
(8비트)
sel
0
1
2
3
32
33
34
35

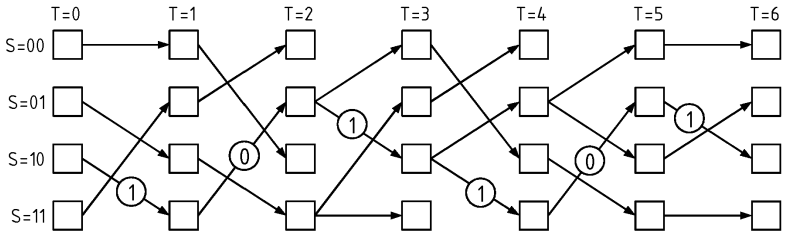
도면9


901
902
903
904
905
906
907
908
909
910
911
SymbolSet
Expected
OldCost
SymbolSet
Expected
OldCost
Sel
Norm
Min
트랜지션
MinCostValIn
MinCostIDIn
MinCostIdOut
MinCostValOut
CostOut
정규화
< 128

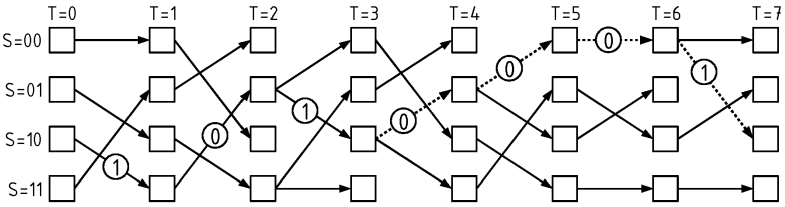
도면10



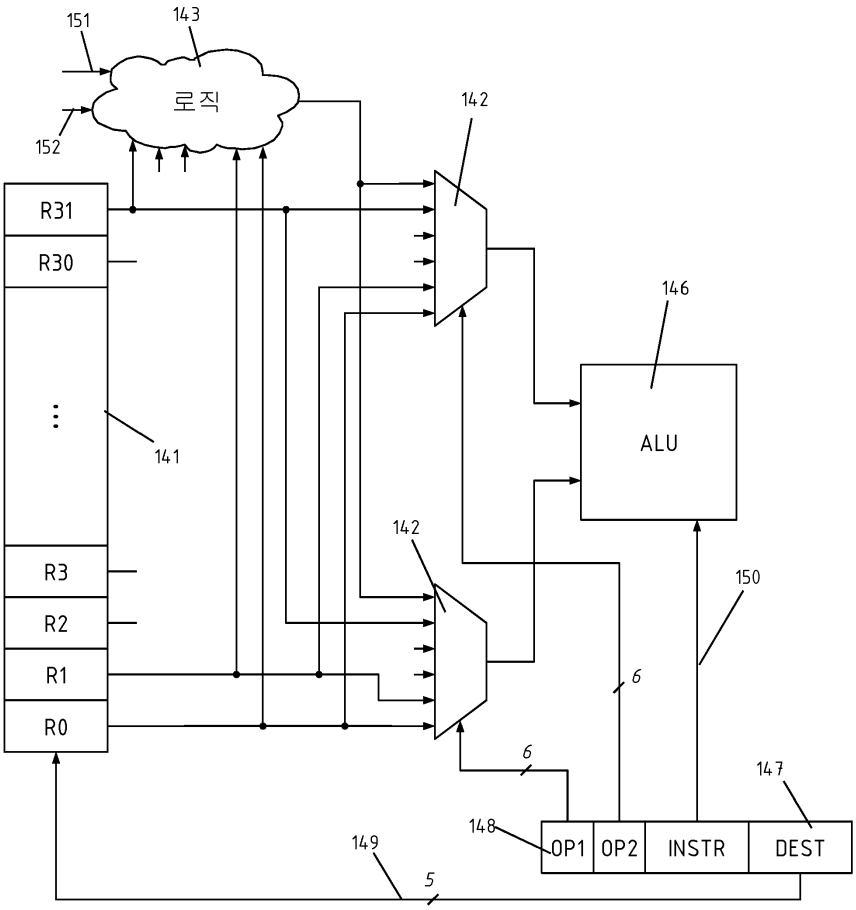
도면11



도면12



도면13


151
143
로직
152
142
R31
R30
141
146
ALU
142
R3
R2
R1
R0
150
6
6
147
148
OP1
OP2
INSTR
DEST
149
5