(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2010년04월01일
(11) 등록번호 10-0950607
(24) 등록일자 2010년03월24일

(51) Int. Cl.
H03M 7/40 (2006.01)
(21) 출원번호 10-2004-7004806
(22) 출원일자 2002년10월11일
심사청구일자 2007년08월17일
(85) 번역문제출일자 2004년03월31일
(65) 공개번호 10-2004-0041651
(43) 공개일자 2004년05월17일
(86) 국제출원번호 PCT/IB2002/004198
(87) 국제공개번호 WO 2003/034597
국제공개일자 2003년04월24일
(30) 우선권주장
10/045,693 2001년10월19일 미국(US)
(56) 선행기술조사문헌
KR1020010053348 A
전체 청구항 수 : 총 30 항

(73) 특허권자
노키아 코포레이션
핀란드 핀-02150 에스푸 케이라라덴티에 4
(72) 발명자
칸가스잔느
핀란드핀-90570오울루칼러본티에9씨39
(74) 대리인
김창세, 장성구

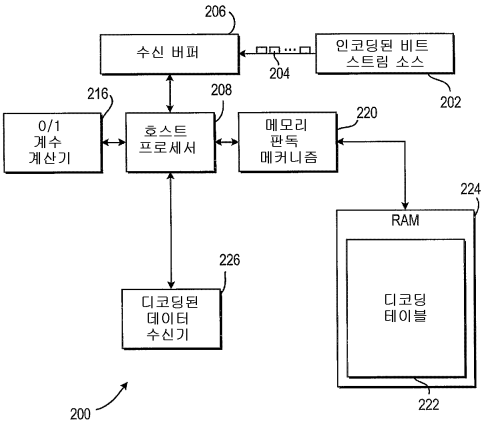
심사관 : 권성락

(54) 허프만 코딩

(57) 요 약

허프만 코드들의 디코딩은 높은 자리의 1들 또는 0들(216)의 연속적인 스트링들을 식별하고, 높은 자리의 1들 또는 0들의 연속적인 스트링들 다음에, 각각의 스트링에 대해 그의 런 계수 및 비트 값에 근거하여, 상기 검색된 엔트리가 디코딩 출력 심볼을 포함할 때까지, 또는 상기 코드 워드의 잔여 비트들이 소정의 한계내의 수에 달할 때까지, 테이블 엔트리(222)를 검색함으로써 달성된다. 상기 잔여 비트들은 검색 테이블내로의 오프셋으로서 이용되지만, 상기 테이블의 치수들은 선두의 1들 및 0들의 제거를 통하여 감소되었다. 상기 연속적인 스트링들은 바람직하게는 하드웨어 가속기에 의해 처리되어 반복된 비트를 식별하고, 상기 스트링내의 비트들을 카운트하고, 그리고 이 정보를 호스트 프로세서에 리턴한다. 정규 코드들을 디코딩하는 것이 효율성이 있다고 알려져 있지만, 비-정규 코드들이 디코딩될 수도 있다.

대 표 도 - 도2



특허청구의 범위

청구항 1

일련의 가변 길이 코드 워드들 중 임의의 코드 워드를 디코딩하는 방법으로서,

a) 상기 코드 워드에서의 비트의 값을 검출하는 단계와;

b) 상기 일련의 가변 길이 코드 워드들로부터, 상기 비트로 시작하며, 후속하는 동일한 값의 연속적인 비트들을 포함하는 현재의 계수(count)를 계산하는 단계와;

c) 상기 현재의 계수에 근거하여, 디코딩 테이블로부터 엔트리를 검색하는 단계와; 그리고

d) 상기 검색된 엔트리에 근거하여, 상기 단계(b)에서의 하나 또는 그 이상의 계수들에 포함된 비트들에 후속하는 비트들을 이용하여, 상기 코드 워드에 대해 상기 단계들 (a) 내지 (d)를 반복할지를 결정하는 단계를 포함하는 것을 특징으로 하는 디코딩 방법.

청구항 2

제 1 항에 있어서,

상기 단계(d)에서, 상기 단계들 (a) 내지 (d)를 반복할 필요가 없다고 결정한 경우, 상기 단계(d)는,

최종적으로 검색된 엔트리가 상기 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는지를 결정하는 단계와; 그리고

만일 상기 출력 심볼을 포함하고 있지 않다면, 상기 단계(b)에서 카운트된 비트들에 후속하는 적어도 하나의 비트에 근거하여, 디코딩 테이블로부터 상기 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는 엔트리를 검색하는 단계를 더 포함하는 것을 특징으로 하는 디코딩 방법.

청구항 3

제 1 항에 있어서,

상기 단계(d)는 상기 검색된 엔트리가 심볼 파운드 표시기(symbol found indicator)를 포함하는지를 검출하는 단계를 더 포함하며, 여기서, 상기 심볼 파운드 표시기의 검출은 상기 코드 워드에 대해 상기 단계들 (a) 내지 (d)를 반복할 필요가 없음을 나타내는 것임을 특징으로 하는 디코딩 방법.

청구항 4

제 1 항에 있어서,

단계(d)는 상기 검색된 엔트리가 분기 표시기(branching indicator)를 포함하는지를 검출하는 단계를 더 포함하며, 여기서, 상기 분기 표시기의 검출은 상기 코드 워드에 대해 상기 단계들 (a) 내지 (d)를 반복할 필요가 없음을 나타내는 것임을 특징으로 하는 디코딩 방법.

청구항 5

제 4 항에 있어서,

상기 분기 표시기가 검출되었다면,

e) 상기 단계(b)에서 카운트된 비트들에 후속하는 적어도 하나의 비트에 근거하여, 디코딩 테이블로부터 상기 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는 엔트리를 검색하는 단계를 더 포함하는 것을 특징으로 하는 디코딩 방법.

청구항 6

제 5 항에 있어서,

상기 단계(e)에 대하여, 상기 적어도 하나의 비트는 상기 코드 워드에 속하고, 소정의 한계(predetermined threshold)보다 많지 않은 수로 이루어지는 것을 특징으로 하는 디코딩 방법.

청구항 7

제 1 항에 있어서,

상기 방법은 각각의 반복들의 검출된 비트 값들이 모두 동일하지는 않은 코드 워드를 디코딩하도록 동작가능한 것을 특징으로 하는 디코딩 방법.

청구항 8

제 1 항에 있어서,

단계(c)에서의 검색은 하나의 반복으로부터 다음 반복까지 다른 디코딩 테이블로부터 행하는 것을 특징으로 하는 디코딩 방법.

청구항 9

제 1 항에 있어서,

상기 현재의 계수로 카운트된 비트들은 상기 일련의 코드 워드들내의 단일 비트에 의해 반복시마다(iteration-to-iteration) 분리되는 것을 특징으로 하는 디코딩 방법.

청구항 10

제 1 항에 있어서,

상기 일련의 가변 길이 코드 워드들은 이동 단말기에 의해 수신되어 상기 방법에 의해 디코딩되는 것을 특징으로 하는 디코딩 방법.

청구항 11

제 10 항에 있어서,

상기 이동 단말기는 이동 전화인 것을 특징으로 하는 디코딩 방법.

청구항 12

제 1 항에 있어서,

상기 가변 길이 코드 워드들은 허프만 인코딩된 코드 워드들인 것을 특징으로 하는 디코딩 방법.

청구항 13

연속적으로 배열된 가변 길이 코드 워드들 중에서 임의의 코드 워드를 디코딩하는 가변 길이 디코딩 장치로서,

상기 코드 워드에서의 비트의 값을 검출하여, 상기 일련의 가변 길이 코드 워드들로부터, 상기 비트로 시작하고, 후속하는 동일한 값의 연속적인 비트들을 포함하는 현재의 계수를 계산하기 위한 선두의 0/1 계수 계산기와;

상기 현재의 계수에 근거하여, 디코딩 테이블로부터 엔트리를 검색하고, 상기 검색된 엔트리에 근거하여, 상기 계산기에 의해 카운트된 비트들에 후속하는 비트들을 이용하여, 상기 계산기를 호출하는 단계와 상기 코드 워드에 대한 상기의 검색 및 결정을 수행하는 단계를 반복할 것인지를 결정하는 제어 블록을 포함하는 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 14

제 13 항에 있어서,

상기 제어 블록은 상기 계산기에 의해 카운트된 비트들에 후속하는 비트들을 이용하여, 상기 코드 워드에 대해 상기 호출, 검색 및 결정 단계들을 반복할 필요가 없다고 결정한 경우, 또한 최종적으로 검색된 상기 엔트리가 상기 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는지를 결정하며, 만약 상기 출력 심볼을 포함하지 않는다면, 상기 계산기에 의해 카운트된 비트들에 후속하는 적어도 하나의 비트에 근거하여, 디코딩 테이블로부터, 상기 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는 엔트리를 검색하는 것을 특징으로 하

는 가변 길이 디코딩 장치.

청구항 15

제 14 항에 있어서,

상기 적어도 하나의 비트는 상기 코드 워드에 속하고, 소정의 한계보다 많지 않은 다수의 비트들로 이루어지는 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 16

제 15 항에 있어서,

상기 장치는 각각의 반복들에서의 상기 검출된 비트 값들이 일반적으로 모두 동일하지는 않도록 동작 가능한 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 17

제 13 항에 있어서,

상기 장치는 각각의 반복들에서의 상기 검출된 비트 값들이 일반적으로 모두 동일하지는 않도록 되어있는 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 18

제 13 항에 있어서,

상기 디코딩 테이블로부터 엔트리를 검색하는 것은, 매 검색마다 다른 디코딩 테이블을 검색하는 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 19

제 13 항에 있어서,

상기 제어 블록으로부터 상기 코드 워드의 상기 디코딩을 수신하는 이동 단말기를 더 포함하는 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 20

제 19 항에 있어서,

상기 이동 단말기는 이동 전화를 포함하는 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 21

제 13 항에 있어서,

상기 가변 길이 코드 워드들은 허프만 인코딩된 코드 워드들인 것을 특징으로 하는 가변 길이 디코딩 장치.

청구항 22

스트링의 선두 비트의 값 및 상기 비트를 포함하는 런의 계수를 결정하는 장치로서,

상기 값을 검출하는 값 검출기와;

상기 검출된 값이 사전선택된 비트 값과 동일한 경우, 상기 스트링의 상기 비트들을 반전시키는 제 1 인버터와;

상기 사전선택된 비트 값과 다른 값의 상기 스트링의 비트와, 그리고 상기 사전선택된 비트 값을 갖는 최상위 비트의 중요도보다 낮은 중요도의 비트 모두를, 상기 사전선택된 비트 값으로 변환하는 디지트 확장기와;

상기 디지트 확장기로부터 출력된 비트들을 반전시키는 제 2 인버터와;

상기 제 2 인버터에 의해 반전된 상기 비트들의 순서를 반전시켜 반전된 스트링을 생성시키는 반전기와; 그리고

상기 사전선택된 값을 갖는 상기 반전된 스트링에서의 상기 비트들의 런 계수를 계산하는 지표 코드 평가기

(thermometer code evaluator)를 포함하는 것을 특징으로 하는 장치.

청구항 23

제 22 항에 있어서,

연속적으로 배열된 가변 길이 코드 워드들로부터 상기 스트링을 선택하는 수단과; 그리고

상기 계산된 런 계수 및 검출된 값을 이용하여 상기 코드 워드들 중 현재의 코드 워드를 디코딩하는 수단을 더 포함하는 것을 특징으로 하는 장치.

청구항 24

제 23 항에 있어서,

상기 가변 길이 코드 워드들은 허프만 인코딩된 코드 워드들인 것을 특징으로 하는 장치.

청구항 25

제 22 항에 있어서,

상기 계산된 런 계수 및 상기 검출된 값의 디지털 표시를 어셈블링(assembling)하는 런 캐릭터라이저(run characterizer)를 더 포함하는 것을 특징으로 하는 장치.

청구항 26

제 25 항에 있어서,

상기 디지털 표시는 상기 계산된 계수의 2진 코딩된 10진 표시를 포함하는 것을 특징으로 하는 장치.

청구항 27

컴퓨터 판독가능한 매체로서,

상기 컴퓨터 판독가능 매체는 연속적으로 배열된 가변 길이 코드 워드 중에서 임의의 코드 워드를 디코딩하는 프로그램 명령들을 저장하고 있으며, 상기 프로그램 명령들은,

a) 상기 코드 워드에서의 비트의 값을 검출하는 단계와;

b) 상기 비트로 시작하며, 상기 일련의 가변 길이 코드 워드들로부터, 후속하는 동일한 값의 연속적인 비트들을 포함하는 현재의 계수를 계산하는 단계와;

c) 상기 현재의 계수에 근거하여, 디코딩 테이블로부터 엔트리를 검색하는 검색하는 단계와; 그리고

d) 상기 검색된 엔트리에 근거하여, 상기 계산 단계 b)에서 카운트된 비트들에 후속하는 비트들을 이용하여 상기 코드 워드에 대해 상기 단계들 (a) 내지 (d)를 재호출할지를 결정하는 단계를 수행하는 것을 특징으로 하는 컴퓨터 판독가능 매체.

청구항 28

제 27 항에 있어서,

상기 단계 d)는 최종적으로 검색된 상기 엔트리가 상기 현재의 코드 워드의 디코딩을 포함하는지를 더 결정하고, 상기 프로그램 명령들은,

e) 상기 단계 d)에서 상기 현재의 코드 워드의 디코딩이 아직 검색되지 않았다고 결정하였다면, 상기 단계 b)에서 카운트된 비트들에 후속하는 적어도 하나의 비트에 근거하여, 디코딩 테이블로부터 상기 현재의 코드 워드의 디코딩을 포함하는 엔트리를 검색하는 단계를 수행하는 것을 특징으로 하는 컴퓨터 판독가능 매체.

청구항 29

스트링의 선두 비트의 값 및 상기 비트를 포함하는 런의 계수를 결정하는 방법으로서,

상기 값을 검출하는 단계와;

상기 검출된 값이 사전선택된 비트 값과 같은 경우 상기 스트링의 상기 비트들을 반전시키는 단계와;

상기 사전선택된 비트 값과 다른 값의 상기 스트링의 비트와, 그리고 상기 사전선택된 비트 값을 갖는 최상위 비트의 중요도보다 낮은 중요도의 비트 모두를, 상기 사전선택된 비트 값으로 변환하는 단계와;

상기 변환 단계후에 상기 스트링을 반전시키는 단계와;

상기 변환 단계후에 반전된 상기 비트들의 순서를 반전시켜 반전된 스트링을 생성시키는 단계와; 그리고

상기 사전선택된 값을 갖는 상기 반전된 스트링에서의 상기 비트들의 런 계수를 계산하는 단계를 포함하는 것을 특징으로 하는 결정 방법.

청구항 30

제 1 항에 있어서,

상기 단계(d)가 상기 코드 워드에 대하여 상기 단계들 (a) 내지 (d)을 반복할 필요가 있다고 결정한 경우, 상기 단계(b)에서의 하나 또는 그 이상의 계수들에 포함된 비트들에 후속하는 비트들을 이용하여 상기 코드 워드에 대하여 상기 단계들 (a) 내지 (d)을 반복하는 단계를 더 포함하는 것을 특징으로 하는 디코딩 방법.

명 세 서

기 술 분 야

[0001] 본 발명은 데이터 압축에 관한 것으로, 특히 허프만 인코딩된 코드 워드들을 디코딩하는 것에 관한 것이다.

배 경 기 술

[0002] 허프만 코드들은 데이터 압축 및 전기통신 영역에서 매우 광범위하게 이용되고 있다. 일부 응용들로서, JPEG 사진 압축과 MPEG 비디오 및 오디오 압축이 있다. 허프만 코드들은 가변 워드 길이로 되어 있으며, 이것은 메시지를 구성하는데 이용되는 개별 심볼들이 별개의 길이의 별개의 비트 시퀀스로 각각 표시(인코딩)됨을 의미한다. 코드 워드의 이러한 특성은 메시지 데이터에서 리던던시(redundancy)의 양을 감소시키는데 도움이 된다. 즉, 상기 특성으로 인해 데이터 압축이 가능하게 된다. 예를 들면, 심볼들(A, B, C 및 D)은 다음의 코드 워드들로 표시될 수 있다.

[0003] 표 1. 허프만 코드표의 예

[0004]

심볼	코드 워드
A	0
B	10
C	110
D	111

[0005] 모든 코드 워드들은 고유하게 디코딩가능하다. 예를 들면, 비트들의 시퀀스 "01101110100"는 'ACDABA'로 디코딩한다. 코드 워드들의 집합을 심볼 리스트 또는 알파벳이라 칭한다. 고유성(uniqueness)은 허프만 코드들의 "접두어 특성(prefix property)" 즉, 코드 워드의 임의의 극좌 또는 "선두(leading)" 서브스트링이 허프만 디코딩 표의 코드 워드와 일치하는 경우, 상기 선두 서브스트링 이외의 임의의 추가 비트들은 검사할 필요가 없다는 사실로부터 생겨난다. 예를 들면, 심볼 "B"에는 "10"의 코드 워드가 할당된다. 따라서, 다른 어떠한 코드 워드들도 "10"으로 시작하지 않는다.

[0006] 허프만 코드들을 이용하면 압축이 가능하게 된다. 그 이유는 별개의 심볼들은 별개의 발생 확률을 갖기 때문이다. 이러한 특성은 심볼들에 대응하는 코드 길이들을 그들 각각의 발생 확률에 따라 맞춤으로써 유리하게 하는데 이용된다. 높은 발생 확률을 갖는 심볼들은 짧은 코드 워드들로 코딩되고, 낮은 발생 확률을 갖는 심볼들은 긴 코드 워드들로 코딩된다. 긴 코드 워드들이 여전히 나타나지만, 그들의 낮은 발생 확률 때문에, 전형적인 비트 스트링내의 모든 코드 워드들의 전체 코드 길이는 허프만 코딩에 의해 더 작아지는 경향이 있다.

[0007] 허프만 코드 생성 알고리즘은 "코딩 트리"에 근거한다. 일반적으로 알려진 알고리즘 단계들은 다음과 같다.

[0008] 1. 심볼들을 확률이 높은 것에서부터 낮은 것으로 정렬시킨다.

[0009] 2. 가장 낮은 확률들을 갖는 2개의 심볼들을, 2개의 심볼들의 확률들의 합이 확률인 하나의 새로운 심볼로 결합시킨다.

[0010] 3. 단일 확률을 갖는 단지 1개의 심볼만이 남겨질 때까지 단계 2를 반복한다.

[0011] 4. 코딩 트리를 루트(1.0의 확률을 갖는 발생된 심볼)로부터 기원(origin) 심볼들까지 추적(trace)하여, 각각의 하위 브랜치에는 1을, 각각의 상위 브랜치에는 0을, 또는 이와 반대로 할당한다.

[0012] 예를 들면, 일부 문자들의 확률들이 표 2에 열거되어 있으며, 상기 알고리즘을 이 확률들에 적용함으로써 형성된 가능한 허프만 트리들 중 하나가 도 1에 도시되어 있다.

[0013] 표 2. 알파벳의 소집합의 확률 분포의 예

[0014]

E	2/25
X	1/25
A	2/25
M	2/25
P	1/25
L	1/25
O	2/25
F	2/25
H	1/25
U	1/25
C	1/25
D	1/25
I	1/25
N	2/25
G	1/25
여백	3/25

[0015] 코드 워드의 각각의 "0" 비트는 트리의 "0" 브랜치를 횡단(traversing)하는 것에 대응하며, 도 1에서 "0" 브랜치의 횡단은 상승하는 식으로, 그리고 "1" 브랜치의 횡단은 하강하는 식으로 행해진다. 코드 워드 "11000"는 루트에서 오른쪽에서 시작하여, 상기 코드 워드의 각각의 비트에 대한 브랜치를 하나씩 횡단하는 식으로 하여 상기 트리상에서 표시된다. 처음 2개의 비트들 "11"은 2개의 "1" 브랜치들 즉, 2개의 다운 스텝(down step)에 대응한다. 그 다음 비트 "0"은 화살표로 표시된 바와 같이, 위로 즉, "0" 브랜치를 따라서 이동하는 것에 대응한다. 잔여 비트들 "00"에 대하여 2개의 "0" 브랜치를 더 횡단함으로써, 완전한 코드 워드 "11000"에 대한 출력 심볼을 얻을 수 있게 된다. 본원에서는 도 1의 좌측에 놓인 문자 "P"가 코드 워드 "11000"에 대한 출력 심볼이 된다.

[0016] 따라서, 도 1로부터, 예를 들면, 문자 "P"에 대한 코드는 "11000"이고, 임의의 주어진 확률 분포에 대한 여러개의 가능한 허프만 테이블들이 존재함을 알 수 있다.

[0017] 허프만 코드의 디코딩에 있어서의 기본적인 어려움은, 디코더가 유입 코드 워드의 길이를 사전에 알 수 없다는 것이다.

[0018] 허프만 코드들은 막대한 양의 메모리를 제공함으로써 매우 빨리 검출될 수 있다. 최대 워드 길이가 N개의 비트들인 허프만 코드 워드들의 집합에 대하여, 2^N 개의 메모리 위치들이 필요하다. 그 이유는 대응하는 코드 워드들을 찾기 위하여 검색 테이블내의 어드레스로서 N개의 유입 비트들이 이용되기 때문이다. 예를 들면, 표 1의 디코딩 심볼들은 2^3 = 8 개의 메모리 위치들을 필요로 한다. "0"으로 시작하는 모든 어드레스들은 심볼 "A"를 저장하는데 이용되고, "10"으로 시작하는 모든 어드레스들은 심볼 "B"를 저장하는데 이용되며, 기타의 어드레스들도 이와 같은 방식으로 특정 심볼을 저장하는데 이용된다. 하나의 코드 워드가 검색 테이블에 적용될 때, 그 부분의 디코딩이 즉시 수행된다. 그 다음, 유입 비트 스트림은 디코딩된 코드 워드의 비트 길이만큼 시프트되어, 그 다음 코드 워드를 동작가능한 디코딩 포지션으로 가져간다. 예를 들면, 최대 길이가 19 비트인 코드들에 대해서는, 메모리 요구량이 매우 커지게 된다.

[0019] 메모리가 적게 요구되는 기술로는 비트별(bit-by-bit) 디코딩이 있으며, 이 디코딩 방법은 다음과 같이 진행한다. 1개의 비트가 취해져서, 워드 길이가 1인 모든 가능한 코드들과 비교된다. 일치하는 코드가 없는 경우, 다

른 비트가 시프트되어, 워드 길이가 2인 모든 코드 워드들 중에서 비트 쌍을 찾아본다. 일치하는 코드를 찾을 때까지 계속하여 이것을 행한다. 비록 이 방식은 매우 메모리 효율적이긴 하지만은, 또한 특히 디코딩된 코드 워드가 긴 경우 매우 느리다.

[0020] 다른 해결책으로서, 내용 주소화 메모리(CAM: content-addressable memory)들을 이용한다. "어드레스들"로서 모든 코드 워드들을, 그리고 "내용들"로서 메모리 포인터들을 포함하는 CAM의 입력에 비트 슬라이스(bit slice)(즉, 임의의 코드 워드를 수용하기에 충분히 길어서 최대 코드 워드와 길이가 동일한 비트 스트링)이 인가된다. 상기 CAM은 RAM 테이블에서 심볼들 및 관련 코드 워드 길이들을 가리키는 메모리 포인터들을 포함한다. 일단 코드 워드가 디코딩되면, 유입 비트 스트림은 디코딩된 코드 워드의 길이만큼 시프트되어, 디코딩을 재개한다. 효율적으로 구현된 CAM 방식은 고속이지만, 여전히 포인터들에 대한 여분의 메모리를 필요로 한다. 또한, CAM들은 모든 기술들에서 쉽게 이용가능하지 않다. CAM 기반의 방식은 미국 특허 번호 제 5,208,593 호에 기재되어 있으며, 하기에서 더 논의된다.

[0021] 상기 실시예들에서 지적된 바와 같이, 가변 코드 워드 길이를 이용함에 있어서의 문제는 속도와 적절한 메모리 사용 간에 균형을 이루는 것이다.

[0022] 정규 허프만 코드(Canonical Huffman code)들이 특별한 관심거리가 되고 있는데, 그 이유는 이 코드들이 디코딩을 더 용이하게 하기 때문이다. PKZip(파일 압축/압축해제 유틸리티), MPEG-1 계층 Ⅲ(Mp3) 및 JPEG 디폴트 기준선 인코더 모두는 정규 허프만 테이블들을 이용한다. 멀티미디어 및 전기통신의 다른 영역들에서도 역시 응용들을 찾아볼 수 있다.

[0023] 정규 허프만 코드들의 특성은, 테이블이 거의 모든 코드들이 선두 1 비트를 갖는 형태로 되어 있다면, 길이가 n인 최소의 허프만 코드의 최상위(n-1) 비트들은 길이가 (n-1)인 최대의 허프만 코드보다 값이 더 크다는 것이다. 선두 비트가 "0"인 코드들로 주로 이루어진 허프만 테이블 즉, 예를 들면, 모든 코드 워드 비트들을 역으로 함으로써 발생된 테이블에 대하여, 역(converse)의 법칙이 적용된다. 길이가 n인 최대의 허프만 코드의 최상위(n-1) 비트들은 길이가 (n-1)인 최소의 허프만 코드보다 값이 더 작다. 허프만 테이블들을 정규 형태로 변환하는 것에 의해 코딩 효율성이 감소되지는 않는다. 그 이유는 표 3의 다음의 실시예로부터 알 수 있는 바와 같이, 상기 변환은 코드 워드당 비트의 수를 변경시키지 않기 때문이다.

[0024] 표 3. 보통 코드 워드 대 정규 코드 워드

[0025]

	트리 코드	코드2	코드3	정규
A	001	111	100	010
B	0000	1101	0011	0000
C	1001	0010	1011	0001
D	101	000	000	011
E	01	01	11	10
F	0001	1100	1010	0010
G	1000	0011	0010	0011
H	11	10	01	11

[0026] 정규 코드들에 대한 상기 역의 법칙에 따르면, 길이가 3인 코드들(예를 들면, 010 및 011)은 길이가 4인 코드들(예를 들면, 0000, 0001, 0010, 0011)의 3개의 시작 비트들보다 항상 크다. 그렇지 않으면, 코드 길이들은 변경되지 않은 상태로 남게된다.

[0027] 또한, 정규 코드들은 종종 상기 특성으로 인해 "1"들(또는 "0"들)의 스트링으로 시작한다는 점에 주목할 필요가 있다. "1" 스트링들로 시작하는 것의 특징은 JPEG 디코딩과 관련하여 미국 특허 번호 제 5,208,593 호("Tong")에서 이용되었으며, 그 이유는 JPEG 허프만 테이블들은 "1"들의 스트링들로 시작하는 여러개의 코드들로 이루어지기 때문이다. 이 참조문헌은 JPEG에서 이용된 허프만 코드들에 "선두의 1들 검출(leading ones detection)"을 적용한다. 디코딩될 후속 코드 워드는 그 후속 코드 워드의 최상위 비트(MSB)(이하, "선두 비트"는 최상위 비트 또는 극좌 비트를 뜻하는 것으로 한다)에서 시작하는 "1"들의 연속적인 런의 길이에 대하여 검사된다. 이 길이 또는 계수가 알려진 후에, 주어진 최대 코드 워드 길이에 근거하여, 코드 워드의 잔여 비트들의 최대수가 몇개인지 역시 알게 된다. "1"들의 연속적인 런(그 다음에는 "0", 그 이유는 "0"이 항상 알려져 있기 때문이다)은 마스킹된다. 상기 잔여 비트들은 연속적인 (선두) "1"들의 수의 인식 외에도, 어드레스를, 심볼들을 포함하는 RAM 테이블로 형성하는데 이용된다.

[0028] Tong의 방법은 단지 "1"들의 선두 비트 스트링을 갖는 허프만 코드 워드들에 대해서만 효과적이다. Mp3 오디오 표준은 예를 들면, "0"들의 선두 스트링들을 갖는 코드 워드를 갖는 허프만 테이블들을 특정한다. 또한, Tong의 방법은 단지 정규 허프만 테이블들에 대해서만 동작하며, 다수의 메모리를 이용한다. Tong의 방법을 하기의 표 4(Hashemian, R. Memory Efficient and High-Speed Search Huffman Coding, IEEE Transactions on Communications, Vol. 43 No. 10, 1995년)에 도시된 허프만 테이블에 적용하게 되면, Tong의 방법은 특히 잘 동작하게 될 것이다. 그 이유는 상기 허프만 테이블은 단일-측 성장 테이블(single-side growing table) 즉, "서브트리들"을 작게 유지하도록 구성된 테이블이기 때문이다. 그러나, Tong의 방법은 36개의 엔트리들을 포함하는 제 2 테이블로의 어드레스들로 13개의 워드들을 이용함으로써, 총계, 13 + 36 = 49 워드를 필요로 한다. 또한, Tong의 방법은 선두 "1"들의 제거후에 최대 코드 워드 길이가 8 비트인 JPEG 표준 AC 테이블들에 적용되는 경우, 메모리에 비효율적으로 될 수 있다. 그 이유는 Tong의 방법은 잔여 8 비트들에 대해 검색 테이블에서 2^8 개의 메모리 위치들을 이용할 것이기 때문이다.

[0029] Potu에 대한 미국 특허 번호 제 6,219,457 호는 유입 코드 스트림이 각각 선두 "0"들로 코딩하는 MPEG 표준하에서 인코딩되었는지, 혹은 선두 "1"들로 코딩하는 JPEG 또는 디지털 비디오 표준(Digital Video Standard)하에서 인코딩되었는지에 따라서, 코드 워드의 연속적인 선두 "0"들의 수, 혹은 코드 워드의 선두 "1"들의 수를 카운트(count)하도록 실시되는 허프만 디코딩 사전-처리를 개시한다. 상기 계수는 가변 길이 코드(VLC: variable length code) 디코딩 테이블의 기준 어드레스를 결정하기 위하여 제 1 검색 테이블내로 인덱싱하는데 이용된다. 소정수의 비트들 다음에 코드 워드의 계산된 비트들이 상기 기준 어드레스와 함께 결합되어, 적절한 VLC 디코딩 테이블을 선택하며, 이 테이블로부터 출력 심볼이 검색된다.

[0030] 그러나, Potu는 Potu가 적용되는 응용에 따라서 선두 1 비트 스트링 또는 선두 0 비트 스트림에 대해 실시하고, 또한, Potu는 동일한 코드 워드내에서 일련의 비트 런들에 대해 효과가 없다. Tong의 경우에서와 같이, Potu는 허프만 코드들을, 이 코드들이 정규일 경우에만 처리할 수 있고, Potu가 동일한 코드 워드의 일련의 비트 런들을 디코딩할 수 없으므로, 디코딩 테이블을 더 크게 만든다.

표 4. Hashemian의 연구로부터의 허프만 테이블

CL	심볼	허프만 코드			
2	00	00			
2	01	01			
4	02	1000			
4	03	1001			
4	04	1010			
4	05	1011			
4	06	1100			
4	07	1101			
5	08	1110	0		
6	09	1110	10		
6	0a	1110	11		
6	0b	1111	00		
7	0c	1111	010		
7	0d	1111	011		
8	0e	1111	1000		
8	0f	1111	1001		
8	10	1111	1010		
8	11	1111	1011		
8	12	1111	1100		
9	13	1111	1101	0	
9	14	1111	1101	1	
9	15	1111	1110	0	
10	16	1111	1110	10	
10	17	1111	1110	11	
10	18	1111	1111	00	
10	19	1111	1111	01	
10	1a	1111	1111	10	
12	1b	1111	1111	1100	
12	1c	1111	1111	1101	
12	1d	1111	1111	1110	
13	1e	1111	1111	1111	0
13	1f	1111	1111	1111	1

[0031]

[0032] Hashemian의 디코딩 방식은 다음과 같이 유입 비트들을 "클러스터링(clustering)"하는 것을 근거로 한다. 처음 L 비트들은 테이블로의 포인터로서 사용하기 위해 "클러스터링"된다. 코드의 길이가 L 또는 그 보다 작은 비트인 경우, 현재의 테이블은 심볼을 포함하고, 그 코드는 즉시 디코딩된다. 코드의 길이가 더 긴 경우, 상기 테이블은 특정한 L 비트들로 시작하는 코드 워드들을 포함하는 다른 테이블들에 대한 포인터들을 갖는다. 이들 새로운 테이블들은 후속 L-비트 클러스터에 의해 다시 어드레스되고, 심볼이 최종적으로 발견될 때까지 계속하여 이것을 반복한다. L을 감소시키면 메모리 효율성은 개선되지만, 디코딩 단계의 수는 증가하게 된다.

[0033] 예를 들면, L=4에 대하여, 13-비트 워드는 심볼을 위치시키는데 4개의 스텝들(13/4=3.25)을 필요로 한다. 상기

13 비트들 중 처음 4개는 제 1 검색 테이블에서, 코드들 모두가 그들의 4개의 비트들로 시작하는 제 2 검색 테이블에 대한 포인터를 식별한다. 그러므로, 그들의 4개의 비트는 더이상 필요가 없다. 따라서, 제 2 검색을 위해 9 비트가 남게되고, 상기 제 2 검색 후에는, 제 3 검색을 위해 5 비트가 남게되고, 그리고 상기 제 3 검색 후에는, 제 4 단계를 요구하는 1 비트가 남게된다. 즉, 3개의 테이블 검색들은 디코딩시에 처음 3개의 단계들을 구성하고, 잔여 비트의 처리는 제 4 디코딩 단계를 구성한다. JPEG는 13 비트의 최대 길이를 이용하고, Mp3의 가장 긴 코드 워드들의 길이는 19 비트이다.

[0034] Hashemian의 방식에 대한 여러가지 단점이 존재한다. Hashemian의 방식은 비트 마스킹 및 비교 단계들에 의존한다. 또한, Hashemian의 방식은 정규 코드들의 특징들을 이용하지 않기 때문에, 이 알고리즘은 연속적인 "1"들 또는 "0"들을 간단하게 점프할 수 없지만, 한번에 많아야 L개의 비트들의 속도로 코드를 처리함으로써, 긴 코드들은 디코딩하는데 매우 긴 시간이 걸리게 된다. 또한, 상기 단일-측 성장 테이블 및 4인 클러스터 길이를 이용하는 Hashemian의 해결책은 메모리의 122 워드들을 차지한다.

[0035] 필요한 것은, 허프만 코드들이 정규인지 여부를 처리하는데 충분히 유연성있으면서도, 정규 형태로 코드들을 디코딩하는 것으로부터 실현가능한 효율성들을 이용하기에 충분히 강건한(robust), 고속의 메모리 효율적인 디코딩 방법이다.

[0036] 상기 문제를 더 가중시키는 것으로서, 일반적인 CPU들은 가변 길이의 코드 워드들을 처리하도록 잘 갖추고 있지 않지만, 16 또는 32 비트들과 같은 본래의 길이들(native lengths)에서 동작을 행한다는 사실이다. 임의의 마스크들로의 비트 필드들의 시프트 또는 마스크 및 그 결과들에 근거한 서치가 느리다. 또한, 허프만 디코딩 알고리즘들은 빈번한 비교들 및 그 결과들에 근거한 브랜치들을 요구하도록 구조화되어 있는데, 이것은 깊은 파이프라인을 갖는 CPU들에 대해 매우 비효율적이다. 일부 디지털 신호 처리기(DSP)들은 비트 필드 처리를 잘 수행할 수 있지만, 불행하게도 또한 긴 파이프라인들을 갖는다. 큰 if/then 또는 switch/case 구문들이 회피되어야 한다.

[0037] 순수한 소프트웨어 디코딩은 느리다. 예를 들면, 스트림에서 처음 "1"을 찾는 것은 2의 지수들을 이용하는 여러 개의 비교 동작들 또는, 대안적으로 다른 복잡한 작업들을 필요로 한다. 하드웨어에서 선두의 "1"을 찾는 것은 단지 복합 로직만을 필요로 하는 단순한 작업인 반면, 일반적인 CPU 명령들로는, 여러 시프트/마스크/비교 동작들이 필요하다.

[0038] 허프만 디코딩을 수행하는 것은 시프터(shifter)들 및 가산기 등과 같은 특화된 독립 하드웨어 구성요소들의 이용을 필요로 한다. 이 방식은 고선명 텔레비전(HDTV) 디코더들 등과 같은 주문형 디바이스들에서 실행가능하지만, 이 구성요소들이 이미 호스트에 존재하기 때문에 고성능 프로세서를 갖춘 시스템에서 이 방식은 자원의 낭비를 가져온다.

[0039] 가속기가 메모리에 그 자신의 접속을 갖고 있어 데이터를 출력하는 완전하게 독립된 디코더(루즈 커플링(loose coupling))로서 구현될 수 있어, 호스트 CPU가 그 자신의 작업들을 수행할 수 있게 된다. 비록 여러 자원들(가산기들, 메모리 인터페이스 유닛들, 시프터들 등)이 중복되어야 하지만은, 성능이 높다. 불행하게도, 허프만 디코딩은, 상기 디코더의 내부 메모리에 저장되는 경우 상기 메모리가 그에 대응하여 크고 비용이 많이 드는 다소 큰 테이블들을 필요로 한다. 상기 테이블들이 공통 메모리내에 있는 경우, 상기 디코더는 디코딩이 메모리 집약형 응용이기 때문에 메모리 버스들을 차단한다.

발명의 상세한 설명

[0040] 일 양상에서, 본 발명은 일련의 허프만 인코딩된 코드 워드들에서 현재의 코드 워드를 디코딩하는 방법, 장치 및 프로그램에 관한 것이다. 상기 코드 워드들에서의 비트의 값이 검출된다. 그 비트 및 동일한 값의 후속하는 연속적인 비트들에 대해 현재의 계수가 계산된다. 상기 현재의 계수에 근거하여, 디코딩 테이블로부터 엔트리가 검색된다. 상기 검출 및 계산은 최종적으로 검색된 엔트리가 더이상 반복들이 수행될 필요가 없음을 표시할 때까지, 이미 카운트된 비트들에 후속하는 비트들에 대하여 매번 되풀이하여 반복된다.

[0041] 본 발명의 다른 양상에서, 최종적으로 검색된 엔트리가 현재의 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하지 않는 경우, 카운트된 비트들에 후속하는 적어도 1개의 비트는 상기 현재의 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는 엔트리를 검색하는데 이용된다.

[0042] 다른 양상에서, 본 발명은 스트링의 선두 비트의 값 및 상기 비트를 포함하는 런의 계수를 결정하는 것에 관한 것이다. 밸류 검출기(value detector)가 상기 값을 검출하고, 제 1 인버터(inverter)가 검출된 값이 사전선택된

비트 값과 같은 경우 스트링의 비트들을 반전시킨다. 디지트 확장기(digit extender)가 사전선택된 비트 값과 다른 값의 스트링의 비트와, 그리고 상기 사전선택된 비트 값을 갖는 최상위 비트의 중요도보다 낮은 중요도의 비트 모두를, 사전선택된 비트 값으로 변환한다. 제 2 인버터가 상기 디지트 확장기로부터 출력된 비트들을 반전시킨다. 반전기(reversor)가 상기 제 2 인버터에 의해 반전된 비트들의 순서를 반전시켜 반전된 스트링을 생성시킨다. 지표 코드 평가기(thermometer code evaluator)가 이전에 선택된 값을 갖는 반전된 스트링에서의 비트들의 런 계수를 계산한다.

[0043] 대안적인 양상에서, 본 발명은 허프만 코드들을 디코딩하는 컴퓨터 판독가능 프로그램 코드 수단을 갖는 컴퓨터 사용가능 매체에 관한 것이다. 상기 수단은 허프만 디코딩 테이블을 포함하며, 이 허프만 디코딩 테이블은 엔트리로서, 연속적으로 배열된 허프만 인코딩된 코드 워드들로부터, 상기 테이블로의 말미 오프셋(tail offset)을 표시하는 잔여 비트들을 식별하는 오프셋을 갖는다. 상기 말미 오프셋을 표시하는 잔여 비트들의 수는 연속 정렬된 각각의, 연속적인, 동일한 값을 갖는 비트들의 다수의 계수들에 근거하여 미리 결정된다. 상기 동일한 값을 갖는 비트들은 상기 잔여 비트들의 그것보다 높은 중요도를 갖고, 그리고 일반적으로 모두, 계수별 동일한 비트 값을 갖지는 않는다.

[0044] 본 발명의 다른 목적들 및 특징들은 첨부 도면들을 참조로 한 다음의 상세한 설명으로부터 명백해질 것이다. 그러나, 상기 도면들은 단지 예시를 위한 것일 뿐이고 본 발명을 한정하고자 하는 것이 아니며, 본 발명의 권리 범위는 첨부된 청구항들에 의해 정의된다.

실 시 예

[0049] 본 발명은 허프만 코드 스트림을 디코딩하는 고속의 메모리 효율적인 방법을 제공한다. 디코딩은 코드 워드의 시작부에서 비트 런들 즉, "0000..." 및 "1111..." - 스트링들의 검출에 근거한다. 연속적인 "1"들 또는 "0"들의 처음 n개의 비트들의 길이가 구해진 후에, (n+1 위치로부터 포워드된) 코드 워드내의 잔여 비트들은 코드 워드에 단지 몇개의 비트들만이 남겨져 있다고 결정될 수 있을 때까지, "1"들 및 "0"들의 연속 스트림들에 대해 다시 서치되고, 이 시점에서, 잔여 비트들은 메모리 테이블로부터 대응하는 심볼을 검색하는데 이용될 수 있다. 이 프로세스는 허프만 트리에서 "연속된 런들"의 최대 길이들을 이동하고, 새로운 "서브트리"로 가는 "전환점(turn)"이 검출될 때마다 멈추는 것으로서 시각화될 수 있다. 장점적으로, 한번에 최소 2개의 비트로 코드 워드가 프로세스된다(선두의 "0" 또는 "1", 그리고 "전환점"을 표시했던 그 다음 비트).

[0050] 예를 들면, 도 1에서, 문자 "P -> 11000"에 대한 디코딩 프로세스는 다음과 같이 개략적으로 프로세스된다. 먼저, 트리의 다른 브랜치에 도달했음을 의미하는 모든 1들/0들 경로로부터의 이탈(deviation)이 검출된다. 그 다음, 더이상 필요하지 않은 처음 2+1개의 비트들("110")이 제거되어, "00XXXXX..."가 디코딩되도록 남겨진다. 여기서, X들은 후속 코드 워드에 속하는 비트들이다. 말미의 점(dot)들은 덜 특정적이며, 이후부터는, 이들의 후속 비트들이 후속 코드 워드에 속하는지 속하지 않는지에 관계없이, 후속 비트들로 언급한다. 도 1을 참조하면, 프로세스된 상기("110") 스트링은 2개의 "1" 브랜치들을 내려가고 1개의 "0" 브랜치를 올라가는 것을 의미한다.

[0051] 상기 잔여 비트들 "00XXXXX"은 선두의 1/0 검출기에 다시 제공된다. 이 경우, 상기 검출기는 "0000..." 방향으로 향하고 있음을 검출한다. 도 1을 참조하면, 모든 "0" 방향의 최대 잔여 코드 길이가 2이기 때문에(그리고 사실상 임의의 방향으로 2개이기 때문에), 루트 "11000"의 끝에 도달하여 이제 디코딩되었다.

[0052] 반대로, 잔여 비트들 "00XXXXX"에 직면했을 때, 미국 특허 번호 제 5,208,593 호(Tong)는 이미 연속적인 높은 자리 "1"들을 검출하였고, 높은 자리 "0"들을 검출하기 위한 어떠한 수단도 갖지 않는다. Tong은 그 대신 디코딩 테이블로의 검색키로서 "최대 잔여 길이"까지 잔여 높은 자리 비트들을 취한다. 상기 최대 잔여 길이는 2를 훨씬 초과할 수 있어, 본 발명에 따라 요구되는 것보다 큰 테이블을 필요로 하게 된다.

[0053] 또한, 본 발명의 방법은 (선두 비트로부터 다른 값의 비트까지) 2 비트를 한번에 최소로 코드 워드를 프로세스하기 때문에, 본 발명은, 특징적으로 같은 수의 선두의 동일한 값을 갖는 비트 스트링들을 갖지 않고 Tong의 방법은 큰 메모리에 의지하지 않고는 처리할 수 없는 비-정규 허프만 인코딩된 코드 워드들 조차도 빨리 디코딩한다.

[0054] Potu와 대조적으로, 본 발명은 제 2 테이블 검색에 의존하지 않으며, 그 대신 다수의 출력 심볼들이 제 1 검색에 억세스된다. 또한, Tong과 같이, Potu는 단지 단일 비트 런 계수가 사전 프로세스되기 때문에, 본 발명을 하는 것보다 큰 테이블 크기를 필요로 하며, Tong과 같이, 비-정규 허프만 코드들을 처리하는데 훨씬 더 큰 테이블 크기가 요구된다.

[0055] 도 2에 도시된 바와 같은 예시적인 디코딩 시스템(200)은 수신 버퍼(206)에 의한 직렬 수신을 위해 허프만 인코딩된 코드 워드들(204)을 전송하는 인코딩된 비트스트림 소스(202)를 포함한다. 호스트 프로세서 또는 제어 블록(208)은 마이크로프로세서일 수 있으며, 예를 들면, 상기 버퍼(206)의 출력을 수신하고 상기 버퍼(206)에 버퍼 제어 명령들을 전송한다. 상기 호스트 프로세서(208)는 하드웨어 가속기일 수 있는 선두의 0/1 계수 계산기(leading zero/one count calculator)(216)에 그 출력에서의 비트들의 그룹을 전송한다. 상기 계산기(216)는 상기 호스트 프로세서(208)에, 그 그룹에서의 연속적인 동일한 값을 갖는 비트들의 계수를 리턴한다. 상기 호스트 프로세서(208)는 상기 계수에 근거하여 어드레스를 결정하고, 메모리 판독 메커니즘(220)을 호출하여 RAM(224) 또는 ROM과 같은 다른 메모리에 존재하는 디코딩 테이블(222)내에서 그 어드레스를 판독한다. 그 어드레스로부터 판독된 데이터에 근거하여, 상기 호스트 프로세서(208)는 (1) 다른 그룹을 위해 다른 계산이 필요한지, (2) 이미 카운트된 비트들에 후속하는 비트들이 필요한지, 또는 (3) 상기 판독 데이터가 현재의 코드 워드가 디코딩되도록 하는 현재의 코드 워드에 대응하는 출력 심볼 즉, 상기 현재의 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하는지를 결정한다. (1)의 경우, 상기 호스트 프로세서(208)는 다른 반복을 위해 상기 선두의 0/1 계수 계산기(216)를 호출한다. (3)의 경우, 상기 현재의 코드 워드는 디코딩된 데이터 수신기(226)에 출력되고, 이 디코딩된 데이터 수신기(226)는 예를 들면, 역이산 코사인 변환 프로세서(reverse discrete cosine transform processor)에 대한 수신 버퍼일 수 있다. 이미 카운트된 비트들에 후속하는 비트들이 필요한 (2)의 경우, 상기 호스트 프로세서(208)는 그 값이 상기 테이블(222)내의 현재의 위치로부터 "말미 오프셋"의 역할을 하여 상기 현재의 워드를 디코딩하는 스트링으로서 그것들 중 소정의 수를 선택한다. 상기 오프셋은 본원에서 "말미 오프셋"이라 칭해지는데, 그 이유는 그것(이하, "말미 오프셋 비트들")을 제공하는 선택된 비트들이 디코딩될 코드 워드의 끝에 놓이기 때문이다.

[0056] 도 3은 본 발명에 따라 디코딩이 어떻게 실시될 수 있는지에 대해 상세한 설명을 제공하는 예시적인 흐름도이다. 예시의 목적을 위하여, 상기 수신 버퍼(206)가 비트 스트링 "00011111010"을 포함한다고 가정한다.

[0057] 프로세스는 현재의 비트 그룹 및 현재의 코드 워드로 시작한다. 이 프로세스의 특정 시점에서, 상기 현재의 그룹은 최하위 끝에서, 상기 현재의 코드 워드를 포함하기에 충분하게 확장할 수 있거나, 혹은 상기 현재의 코드 워드가 상기 현재의 그룹보다 더 긴 경우일 수 있다. 본 실시예에서, 한 그룹의 길이가 8 비트로 설정되는데, 그 이유는 상기 0/1 계수 계산기(216)가 8 비트의 서치 필드 길이로 구성되었기 때문이다. 따라서, 본 실시예에서의 상기 현재의 그룹은 "00011111"이고, 상기 계산기(216)에 전송된다. 상기 계산기(216)는 선두 비트의 값을 "0"으로서 검출하고(단계(S302)), 현재의 계수로서, 상기 선두 비트 "0"의 연속적인 비트 반복들의 수를 계산한다. 즉, 상기 현재의 계수는 상기 선두 비트로 시작하고 동일한 값의 후속하는 연속적인 비트들을 포함한다. 상기 계산기(216)는 상기 호스트 프로세서(208)에 3인 상기 현재의 계수를 리턴한다(단계(S304)).

[0058] 상기 호스트 프로세서(208)는 상기 디코딩 테이블(222)로의 오프셋으로서, 상기 현재의 계수 3을 이용하여, 상기 테이블(222)의 어드레스를 가리킨다. 상기 프로세서(208)는 상기 어드레스를 상기 메모리 판독 메커니즘(220)에 제공한다. 상기 메커니즘(220)은 그 어드레스에서 엔트리를 검색하여 상기 엔트리를 상기 프로세서(208)에 제공한다(단계(S306)). 그 엔트리의 내용들에 근거하여, 상기 프로세서(208)는 상기 현재의 코드 워드를 디코딩하기 위하여 연속적인 비트들을 카운트하는 다른 반복이 필요한지를 결정한다(단계(S308)). 다른 반복이 필요한 경우, 본 실시예에서와 같이, 후속 그룹이 현재의 그룹으로 되고(단계(S310)), 다른 반복(단계들(S302 내지 S306))이 수행된다. 상기 본 실시예에서, 상기 현재의 그룹에 대해 형성된 후속 그룹은 "1111010X"이고, 여기서 X는 상기 수신 버퍼(206)내에 다음에 올 비트를 표시하고, 따라서 상기 프로세서(208)에 의해 검색된다. 상기 현재의 그룹이 "1111010X"라고 결정할 때, 선행 비트 스트링 "0001"이 스킵되었다. 3개의 높은 자리 "0"들은 이미 카운트되었고, "1"은 "0"들의 스트링을 종결시키는 본질적으로 유일한 비트라는 사실로부터 알려져 있기 때문에, 이 4개의 비트들은 더이상 필요가 없다.

[0059] 후속 반복에서, 상기 선두 비트가 검출되고(단계(S302)), 상기 현재의 계수가 계산된(단계(S304)) 후에, 상기 프로세서(208)는 상기 디코딩 테이블(222), 또는 상기 테이블(222)을 대신하기 위해 분기되어 교환되는 다른 디코딩 테이블로부터 다른 엔트리를 검색한다(단계(S306)). 새로 검색된 엔트리에 근거하여, 상기 프로세서(208)는 다른 반복이 필요하지 않고(단계(S308)), 최후 검색된 엔트리(단계(S306))가 현재의 코드 워드의 디코딩을 구성하는 출력 심볼을 포함하지 않는다(단계(S312))고 결정한다. 본 실시예에서, 이것은 동일한 현재의 코드 워드를 디코딩하는 프로세스에서 두번째 반복이다. 상기 현재의 코드 워드가 디코딩되지 않았기 때문에(단계(S312)), 상기 현재의 코드 워드를 디코딩하기 위하여 말미 오프셋 비트들이 필요하다. 상기 프로세서(208)는 최후 검색된 엔트리(단계(S306))로부터, 상기 말미 오프셋이 2개의 비트들에 의해 제공되어, 후속하는 2개의 비트들을 검색하는 것(단계(S314))을 알고 있으며, 이 2개의 비트들은 상기 본 실시예에서 "10"인 것을 알 수 있

다. 상기 말미 오프셋 비트들의 최대 비트 길이는 본원에서 "말미 한계(tail threshold)"로 칭해지며, 이 길이는 상기 본 실시예에서 2이다. 상기 프로세서(208)는 상기 말미 오프셋을 이용하여 현재의 디코딩 테이블내의 그의 현재의 위치를 지나서, 출력 심볼이 존재하는 위치를 향하여 가리키고, 그 다음 상기 출력 심볼을 추출한다(단계(S314)). 상기 출력 심볼을 검색하는데 막 이용된 비트들은 더이상 필요가 없기 때문에, 상기 프로세서(208)는 후속 코드 워드 준비를 위해 이 비트들을 지나칠 것을 지시한다(단계(S316)).

[0060] 그 다음, 상기 소스(202)로부터의 비트 스트림의 디코딩이 완료되었는지가 결정된다(단계(S318)). 상기 수신 버퍼(206)로부터 수신된 모든 비트들이 디코딩을 거쳤다면, 프로세싱을 중단하여 대기했다가 상기 수신 버퍼(206)에 추가 비트들이 수신되면 재시작한다. 한편, 비트들이 모두 디코딩되지는 않았다면, 상기 프로세싱은 상기 현재의 그룹에 대해 후속 그룹이 형성되는(단계(S310)) 다른 반복에 대한 시작부로 돌아간다. 대안적으로, 상기 소스(202)로부터의 비트 스트림의 디코딩의 완료는 비트 스트림의 특정 코드 워드에 의해 표시될 수 있다. 이러한 실시예에서, 단계(S316)는 상기 특정 코드 워드와 막 디코딩된 것들에 후속하는 비트들을 비교하는 단계를 포함한다. 일치하는 경우, 프로세싱은 완료되고, 디코딩이 재시작될 것이라는 신호로 재시작한다. 한편, 일치하지 않는 경우, 프로세싱은 단계(S310)로 돌아간다.

[0061] 따라서, 상기 본 실시예에서, 코드 워드를 디코딩할 때, 0 비트들의 런이 카운트되고, 비트가 스킵되고, 그리고 1 비트들의 런이 카운트된다. 상기 코드 워드의 최종적인 2개의 비트들이 디코딩 테이블로의 말미 오프셋을 제공한다. 말미 오프셋 비트들의 수는 최대 2개의 비트로 한정되었고, 최종 반복전에 반복들의 수를 최대 결정하였고, 이 실시예에서는 2였다.

[0062] 아래의 표 5는 본 발명에 따른 디코딩의 다른 실시예에서 이용되며, 상기에서 표 4로서 번호 붙여진 허프만 테이블에 근거하여, 그리고 아래에서 논의된 포매팅(formatting)에 따라 형성된 예시적인 검색 테이블을 이용한다.

표 5. 예시적인 검색 테이블

행 #	엔트리 식별자	심볼/오프셋 어드레스	시프트 양/코드 길이
-13	S	0x1f	13
-12	S	0x1e	13
-11	S	0x1d	12
-10	B	+31	15
-9	S	0x 1a	10
-8	B	+29	14
-7	B	+25	14
-6	B	+21	14
-5	B	+17	14
-4	B	+13	14
-3	B	+9	14
-2	B	+5	15
-1	B	+3	14
0	2		13
1	S	0x 01	2
2	S	0x 00	2
3		0x 02	4
4		0x 03	4
5		0x 04	4
6		0x 05	4
7		0x 06	4
8		0x 07	4
9		0x 08	5
10		0x 08	5
11		0x 09	6
12		0x 0a	6
13		0x 0b	6
14		0x 0b	6
15		0x 0c	7
16		0x 0d	7
17		0x 0e	8
18		0x 0f	8
19		0x 10	8
20		0x 11	8
21		0x 12	8
22		0x 12	8
23		0x 13	9
24		0x 14	9
25		0x 15	9
26		0x 15	9
27		0x 16	10
28		0x 17	10
29		0x 18	10
30		0x 19	10
31		0x 1b	12
32		0x 1c	12

[0063]

[0064] 표 5(이 실시예에서, 단지 하나의 테이블만이 필요하지만, 후의 실시예는 여러개의 테이블들을 이용하는 경우를 도시함)는 46개의 엔트리들 즉, 행(row)들을 가지며, 각각은 이전에 논의된 바와 같이, Tong(49 워드) 또는 Hashemian(122 워드)에서 이용된 것보다 작은 공간인 16 비트 워드로 이루어진다. 32개의 가능한 코드 워드들이 존재하며, 여기서 효율성은 32/46 = 69.6% 라고 간주한다.

[0065] 표 5가 도 2에 도시된 디코딩 테이블(222)로 구현될 때, 열(column)은 사실상 메모리 공간에 존재하지 않지만, 표 5의 행(row)들은 본원에서 예시를 위하여 "행 #"으로 번호 붙여져서 도시된다. 표 5에서, 행 0은 이 실시예에서 값들 "2" 및 "13"을 포함하는 2개의 필드를 갖는다. 이 2개의 필드는 각각 8 비트이고, 전체 행 길이는 16 비트이다. 행 0 위에 도시된 행들은 음(negative)의 행 번호들로 라벨이 붙여지고, 행 0 아래에 도시된 행들은 양(positive)의 행 번호들로 라벨이 붙여진다. 행 0을 제외한 모든 행들은 3개의 필드를 갖는다. 이 3개의 필드 즉, "엔트리 식별자", "심볼/오프셋 어드레스" 및 "시프트 양/코드 길이"는 각각 2, 10 및 4 비트이고, 전체 행 길이는 16 비트이다.

[0066] "엔트리 식별자"라고 표제가 붙은 제 1 필드는 다음의 3개의 가능한 경우들에 대한 정보를 보유한다.

[0067] 1) 상기 필드가 "S"(예를 들면, 본원에서 "00"으로 표시되는 2개의 0 비트들에 의해 메모리에 표시되는 "심볼 파운드 표시기(symbol found indicator)")를 포함하는 경우, 엔트리 즉, 행은 출력 심볼(즉, 디코딩 결과) 및 그의 길이를 포함한다.

[0068] 2) 상기 필드가 "B"("01"로 표시되는 "분기 표시기(branching indicator)"를 포함하는 경우, 엔트리는 테이블 시작 어드레스(table starting address)로부터 오프셋을 보유한다. 표 4의 상기 테이블 시작 어드레스는 행 0의 위치에 대응한다. 상기 오프셋은 출력 심볼 및 그의 길이를 포함하는 메모리 엔트리를 가리키는 어드레스를 형성하는데 이용된다. 즉, 상기 오프셋은 그의 "엔트리 식별자"로서 "S"를 갖는 엔트리 즉, "S" 엔트리를 가리키는 어드레스를 형성하는데 이용된다.

[0069] 3) 상기 필드가 "N"("10"으로 표시되는 "후속 테이블 표시기(next table indicator)")를 포함하는 경우, 상기 엔트리는 메모리(224)의 다른 곳에 있는 새로운 테이블을 가리키는데 이용되는 테이블 시작 어드레스로부터 오프셋을 보유한다. 대안적으로, 상기 새로운 테이블은 이전의 테이블을 대신하기 위해 분기되어 교환될 수 있다. (현재의 실시예에서, 표 5로부터 명백해지는 바와 같이, 이 실시예의 다음에 바로 설명되는 바와 같이, "N" 엔트리는 존재하지 않는다.)

[0070] "심볼/오프셋 어드레스"라고 표제가 붙은 제 2 필드는 다음의 3개의 서로 다른 타입의 정보를 보유한다.

[0071] 1) 출력 심볼(본원에서, 각각 0 내지 31에 대한 16진수인 0x00 내지 0x1f 중 하나). 여기서, "0x"는 뒤따를 심볼이 16진수임을 의미한다.

[0072] 2) 테이블 시작 어드레스로부터의 오프셋. 상기 오프셋은 현재의 코드 워드에 대한 출력 심볼을 가리키는 어드레스를 형성하는데에 이용된다.

[0073] 3) 테이블 개시 어드레스로부터의 오프셋. 상기 오프셋은 새로운 테이블 시작 어드레스를 가리키는 어드레스를 형성하는데에 이용된다. (이 실시예에서, 표 5로부터 명백해지는 바와 같이, 새로운 테이블 시작 어드레스를 가리키는 엔트리는 존재하지 않는다.)

[0074] "시프트 양/코드 길이"라고 표제가 붙은 제 3 필드는 다음의 2가지 타입의 정보를 보유한다.

[0075] 1) 현재의 코드 워드가 출력 심볼을 가리키는 유효 어드레스를 형성하도록 말미 오프셋 비트들의 위치를 정하기 위해 오른쪽으로 시프트되어야 하는 양을 포함한다.

[0076] 2) 현재의 코드 워드의 길이를 포함한다. 출력 심볼을 찾은 후에 현재의 코드 워드의 밖으로 시프트하고 새로운 코드 워드내로 시프트되도록 이 길이 만큼 현재의 코드 워드를 시프트시킨다.

[0077] 테이블 시작 어드레스에서의 브랜치 엔트리는 양의 브랜치 계수 및 음의 브랜치 계수를 포함한다. 즉, 모두 0인 쪽(00 및 01)에 대해 2개의 가능한 브랜치들 및 모두 1인 쪽에 대한 13개의 가능한 브랜치들이 존재한다. 상기 양의 브랜치 계수 및 상기 음의 브랜치 계수는 일반적으로 현재의 디코딩 테이블에 대한 임의의 코드 워드에서 비록 하기에 논의된 바와 같이, 상기 테이블이 이들 한계들을 초과하는 코드 워드들로 구성될 수 있지만은, 연속적인 0 및 1 비트들의 최대수에 각각 대응한다.

[0078] 예를 들면, 유입 코드 스트림이 4개의 코드 워드들 "1111111110 00 01 111011"을 포함하고, 수신 레지스터

(206)에서 수신되는 경우:

[0079] 1) 호스트 프로세서(208)는 비-순환형 시프트 레지스터인 코드 워드 레지스터에서 16 비트의 현재의 그룹(CURR_GRP)을 수신한다. 가속기(216)는 상기 호스트 프로세서(208)로부터 16 비트의 현재의 그룹(CURR_GRP)을 수신하며, 상기 16 비트의 현재의 그룹(CURR_GRP)은 이 실시예에서는, 표 4에서 알 수 있는 바와 같이, 가장 큰 코드 워드는 13 비트이기 때문에, 상기 현재의 코드 워드를 포함해야 한다. 이에 따라 상기 현재의 코드 워드 플러스 다음의 코드 워드(들)의 비트들을 포함하는 CURR_GRP는 다음의 16 비트 즉, "1111111110 00 01 11"를 포함한다.

[0080] 본원에서, CURR_GRP는 처음 3개의 코드 워드들 및 4번째 코드 워드의 부분을 포함한다. 제 1 작업은 제 1 코드 워드를 디코딩하는 것이고, 상기 제 1 코드 워드는 이 시점에서 상기 현재의 코드 워드이다.

[0081] 2) 도 3을 참조하면, 가속기(216)는 그룹의 선두 비트를 1인 것으로 검출하여(단계(S302)), -9의 값을 리턴한다. 크기 9는 상기 그룹에서 가장 높은 9개의 비트들을 차지하는 연속적인 1들 후에 처음 0을 찾았음을 의미한다. 음의 부호는 단계(S302)에서 검출된 선두 비트의 값이 1임을 의미한다. 검출된 선두 비트가 0이었다면, 리턴된 값은 양의 부호였을 것이다. ACC_VALUE가 리턴값 즉, 상기 가속기(216)가 리턴하는 현재의 계수라면, 이 실시예에 대하여 ACC_VALUE = -9이다(단계(S304)).

[0082] 3) 호스트 프로세서(208)는 ACC_VALUE에 대해 리턴된 값이 유효한지를 검사한다. ZEROS_MAX 및 ONES_MAX가 양의 브랜치 계수 및 음의 브랜치 계수를 각각 표시하는 경우, ACC_VALUE를 유효화하도록 ACC_VALUE와 비교하기 위해 ZEROS_MAX가 선택될지 ONES_MAX가 선택될지는 ACC_VALUE의 부호에 따른다. 즉, ONES_MAX는 단지 ACC_VALUE가 1 비트들의 계수를 표시하는 음일 때만 이용된다. 이 실시예에서, ACC_VALUE가 음이기 때문에, ONES_MAX <= ACC_VALUE 인지에 대해 검사가 이루어진다. 표 5는 행 0에 상기 음의 브랜치 계수 필드의 내용들로서 13을 표시한다. 비록 기억장소를 보존하기 위해 부호없이 도시되어 있지만은, 상기 음의 브랜치 계수는 ONE_MAX가 상기 음의 브랜치 계수를 표시하고, 단지 ACC_VALUE가 음일 때만 ACC_VALUE와 비교되기 때문에 항상 음이다. 따라서, 상기 음의 브랜치 계수, 그리고 이에 따라 ONE_MAX는 이 실시예에서 -13과 같다. -13 <= -9 이기 때문에, ONES_MAX <= ACC_VALUE 이다. 그러므로, ACC_VALUE는 유효한 것으로 생각된다.

[0083] 4) 호스트는 엔트리 *(TABLE_STARTING_ADDRESS + ACC_VALUE)를 판독하고, 여기서 TABLE_STARTING_ADDRESS는 테이블 시작 어드레스를 칭하며, 본원에서는 행 0에 위치한다. 즉, 행 0 + (-9) = -9 에서의 테이블 엔트리는,

[0084]

S	0x 1a	10

[0085] 이고, 상기 엔트리를 비-순환형 시프트 레지스터인 검색 레지스터내로 복사하고(단계(S306)), 여기서 *(ADDRESS)는 ADDRESS에서의 메모리 위치의 내용을 가리킨다. 표 5를 참조하면, 상기 테이블 엔트리에서, ACC_VALUE = -9 이기 때문에, TABLE_STARTING_ADDRESS 위에 9개의 엔트리들이 위치된다.

[0086] 5) 호스트(208)는 이 엔트리에 대한 "엔트리 식별자" 필드가 S, B 또는 N인지를 검사한다. 이렇게 하기 위해서, 상기 호스트 프로세서(208)는 작업 레지스터(work register)내로 검색 레지스터의 내용들을 복사한다. "심볼/오프셋 어드레스" 필드는 이전에 설명된 바와 같이, 상기 테이블의 이 제 2 필드의 제 2 타입이다. 상기 제 2 타입은 상기 테이블 시작 어드레스, 이 필드에서 찾은 오프셋, 그리고 말미 오프셋의 합에 의해 표시된 어드레스로부터 출력 심볼을 찾을 수 있음을 표시한다. 상기 "심볼/오프셋 어드레스" 필드의 이러한 제 2 타입을 이하 "OFFSET"으로 칭하기로 한다. 이 실시예에서, 검색된 테이블 엔트리의 상기 도표의 중간 필드가 OFFSET이고, 이 필드의 길이는 10 비트이며, 가장 오른편의 필드가 "시프트 양/코드 길이" 필드이고, 이 필드의 길이는 4 비트이다. 이들 길이들은 불변이 아니며, 하기에 설명된 바와 같이 선택되어 다른 설계 파라미터들을 조절한다. 당면 목적은 가장 왼편의 필드 즉, "엔트리 식별자" 필드내의 값을 결정하는 것이다. 상기 작업 레지스터를 오른쪽으로 14 비트만큼 시프트하는 것은, 상기 "엔트리 식별자" 필드를 오른쪽으로 자리맞춤하여 가장 왼편의 14 비트를 "0"들로 채우는 작용을 한다. 상기 값들, "S", "B" 및 "N" 각각은 그들 각각의 저장 위치들의 가장 오른편에 존재한다. 따라서, 상기 작업 레지스터를 시프트하는 것은, 상기 작업 레지스터내의 "엔트리 식별자"와 상기 각각의 저장 위치들 각각과의 비트별(bit-to-bit) 비교를 용이하게 해주어, 상기 "엔트리 식별자"는 비교에 의해 상기 값들(S, B 및 N) 중 하나로서 식별될 수 있게 된다. 상기 비교들은 임의의 순서로, 예를 들면, N, S, 그 다음에 B로 이어지는 순서로 행해질 수 있다. 선택적으로, 비교들은 가능하게는 상기 3개의 값들 중 단지 2개의 값과 행해질 수 있는데, 그 이유는 제 3 값이 제거 프로세스에 의해 결정되기 때문이다.

[0087] 6) 시프트된 작업 레지스트와 상기 값("S")과의 비교는, 이 실시예에서, 상기 "엔트리 식별자"가 "S"임을 표시

하고, "S"는 다른 반복이 필요하지 않고(단계(S308)), 현재의 코드 워드가 디코딩되었음(단계(S312))을 표시하며, 이에 따라 검색 레지스터는 상기 현재의 코드 워드에 대한 출력 심볼을 보유함을 표시한다.

[0088] 7) 후속 코드 워드를 준비하기 위하여, 상기 호스트(208)는 상기 검색 레지스터의 상기 "시프트 양/코드 길이" 엔트리 즉, 마지막 4 비트를 검사하여, 막 디코딩된 코드 워드의 코드 길이(CW_LEN)를 알아낸다. 이 실시예에서는 상기 코드 길이가 10이다.

[0089] 8) 상기 검색 레지스터를 2 비트만큼 왼쪽으로 시프트하여 2 비트 길이의 상기 "엔트리 식별자"를 제거하고, 그 다음에 6 비트 만큼 오른쪽으로 시프트하여 상기 시프트 양/코드 길이 필드를 제거하고(상기 시프트 양/코드 길이 필드의 길이는 4 비트이지만, 6 비트만큼 오른쪽으로 시프트하여 왼쪽으로 2 비트 시프트한 것을 보상한다) 출력 심볼의 형태로 디코딩된 출력을 포함하는 OFFSET을 오른쪽 자리맞춤한다. 이 실시예에서, 상기 출력 심볼은 검색된 엔트리내에 상기 예시된 바와 같이, "0x1a"이다. 상기 출력 심볼은 이제 상기 검색 레지스터에서 분리되어 오른쪽 자리맞춤되었고, 이에 따라 상기 출력 심볼은 상기 디코딩된 출력의 후속 프로세싱을 위해 이용가능하게 된다.

[0090] 9) 상기 호스트(208)는 상기 코드 워드 레지스터를 그 값이 10인 CW_LEN 만큼 시프트함으로써 새로운 코드 워드를 디코딩할 준비를 한다. 이 실시예에서, 비트들 "1111111110"은 상기 코드 워드 레지스터의 왼쪽으로 시프트된다. 새로운 비트들이 이용가능한 경우, 이 새로운 비트들은 오른쪽으로부터 삽입된다. 이 실시예에서, "1011....."이 오른쪽으로부터 시프트된다. 말미의 점들은 존재하는 경우, 후속 비트들을 표시한다. 따라서, 상기 코드 워드 레지스터는 "00 01 111011 ....."을 포함한다. 예를 들면, 코드 스트림의 디코딩이 행해질 때와 같이, 수신 버퍼(206)로부터 다른 비트들이 이용가능하지 않았다면, 후속 비트들은 존재하지 않을 것이다. 상기 코드 워드 레지스터가 비어있지 않다면(단계(S318)), 이 경우와 같이, 디코딩이 계속되고, 후속 그룹이 현재의 그룹으로 된다(단계(S310)).

[0091] 10) 이 경우, 상기 현재의 그룹의 처음 10개의 비트들 "00 01 111011" 및 16 비트의 레지스터 한계까지의 임의의 다른 후속 비트들로 이루어지는 CURR_GRP는 비트 그룹으로서 가속기(216)에 전송되어, 이제 상기 현재의 코드 워드로 간주되는 새로운 코드 워드를 디코딩할 준비가 되어 있다.

[0092] 11) 상기 가속기(216)는 CURR_GRP를 수신한다. 이 실시예에서, 상기 CURR_GRP는 상기의 단계(9)에서 표시된 바와 같이, "00 01 111011....."이다.

[0093] 12) 상기 가속기(216)는 3개의 "0"들 다음의 첫번째 "1"이 발견되었음을 의미하는 ACC_VALUE = 3(즉, 현재의 계수는 3임)을 리턴한다.

[0094] 13) 상기 호스트(208)는 ACC_VALUE =< ZEROS_MAX 인지를 검사한다.

[0095] 14) ZEROS_MAX 즉, (표 5의 행 0에 있는) TABLE_STARTING_ADDRESS에 위치한 가장 왼편의 필드는 2이다. ACC_VALUE가 3이기 때문에, ACC_VALUE =< ZEROS_MAX 즉, ACC_VALUE가 "범위를 넘었음(out of bounds)"은 맞지 않다. 따라서, ACC_VALUE는 ZEROS_MAX의 값으로 설정되며, 이 경우 이 값은 2이다.

[0096] 15) 상기 호스트(208)는 엔트리 *(TABLE_STARTING_ADDRESS + ACC_VALUE) 즉, 행 0 + 2 = 2에서의 엔트리를 판독하며, 다음의 엔트리를 가리킨다.

[0097]

S	0x 00	2

[0098] 이고, 상기 호스트(208)는 이 엔트리를 상기의 단계들(4 및 5)에서와 같이, 검색 레지스터 및 작업 레지스터내로 복사한다.

[0099] 16) 상기 호스트(208)는 상기의 단계(5)에서와 같이, 이 때 이 실시예에서 "S"를 포함하는 "엔트리 식별자" 필드를 검사하여, 현재의 코드 워드가 디코딩되었는지를 결정한다(단계들(S308 및 S312)).

[0100] 17) 상기 호스트(208)는 상기의 단계(7)에서와 같이, 마지막 4개의 비트를 추출하여, 이 때 이 실시예에서 그 값이 2와 동일한 CW_LEN을 결정한다.

[0101] 18) 상기 호스트(208)는 2개의 최상위 비트들을 검출하여, 상기 "엔트리 식별자" 필드를 삭제하고, 상기의 단계(8)에서와 같이, OFFSET(길이가 4 비트임)을 4 + 2 = 6 비트만큼 오른쪽으로 시프트하여 이 때 이 실시예에서 "0x00"인 출력 심볼을 오른쪽 자리맞춤한다.

[0102] 19) 상기 코드 워드 레지스터는 상기의 단계(9)에서와 같이, CW_LEN 만큼 왼쪽으로 시프트된다. 이 때 이 실시예에서, CW_LEN = 2 비트이다. 상기 코드 워드 레지스터의 오른쪽에 새로운 비트들이 추가된다.

[0103] 20) 이제, CURR_GRP는 막 디코딩된 2개의 비트들이 상기의 단계(19)에서의 상기 코드 워드 레지스터의 밖으로 시프트되었기 때문에 "01 1110 11..."을 포함한다.

[0104] 21) 상기 가속기(216)는 그 그룹의 선두 비트를 0으로 검출하여(단계(S302)) 값 1을 리턴한다. 따라서, ACC_VALUE = 1 이다.

[0105] 22) ACC_VALUE는 ACC_VALUE =< ZEROS_MAX이고 그 값이 2이기 때문에 범위내에 있다.

[0106] 23) 상기 호스트(208)는 *(TABLE_STARTING_ADDRESS + ACC_VALUE) 즉, 행 0 + 1 = 1에서의 엔트리를 검사하여 다음을 수신하고, 이것은 상기 검색 레지스터 및 상기 작업 레지스터에 저장된다.

[0107]

S	0x 01	2

[0108] 24) 상기 작업 레지스터의 최하위 끝으로 시프트된 후에, 상기의 단계들(5 및 6)에서와 같이, 비교에 의해 "엔트리 식별자"가 "S"인 것으로 결정된다.

[0109] 25) 상기 호스트(208)는 상기 검색 레지스터의 마지막 4개의 비트를 검사하여 상기 "시프트 양/코드 길이" 필드로부터 값 2를 수신한다.

[0110] 26) 출력 심볼은 길이가 2이고 "0x01"이다. 상기에 나타낸 바와 같은 동일한 절차를 수행하여 후속 코드 워드를 준비한다.

[0111] 27) CURR_GRP = "111011..." 이다.

[0112] 28) ACC_VALUE = -3 이다.

[0113] 29) ONES_MAX = -13 이고 ONES_MAX <= ACC_VALUE 이기 때문에 ACC_VALUE 는 범위내에 있다.

[0114] 30) 상기 호스트(208)는 *(TABLE_STARTING_ADDRESS + ACC_VALUE) 즉, 행 0 + (-3) = -3 에서 테이블을 판독하여 다음을 수신하고, 이것은 상기 검색 레지스터 및 상기 작업 레지스터에 저장된다.

[0115]

B	+9	14

[0116] 31) 상기 작업 레지스터의 최하위 끝으로 시프트된 후에, 상기의 단계들(5 및 6)에서와 같이, 비교에 의해 상기 "엔트리 식별자"가 "B"인 것으로 결정된다. "B" 엔트리에 대하여, 다른 반복이 필요하지 않고(단계(S308)), 현재의 코드 워드는 아직 디코딩되지 않았다(단계(S312)).

[0117] 32) 상기 호스트(208)는 상기 검색 레지스터의 마지막 4개의 비트들을 검사하여, 상기 "시프트 양/코드 길이" 필드로부터 값 14를 수신한다. 이 값은 말미 오프셋 비트들을 오른쪽 자리맞춤하도록 임시 레지스터(이하 "TEMP"라고 칭해짐)가 오른쪽으로 시프트되는 비트의 수를 결정하는데 이용됨으로써, 말미 오프셋이 추가되어, 하기의 단계(35)에서, 출력 심볼을 가리키는 어드레스를 형성한다.

[0118] 33) 이 시점에서, 상기 호스트(208)는 상기 코드 워드 레지스터를 1 플러스 ACC_VALUE의 크기 즉, |ACC_VALUE|+ 1 = 3 + 1 = 4 비트만큼 왼쪽으로 시프트하여, 이미 카운트된 비트 런 및 말미 비트를 시프트하는데, 상기 말미 비트는 본래 알려져 있다. 그러나, 상기 코드 워드 레지스터는 "B" 엔트리가 비트들을 추가하는 말미 오프셋을 필요로 하여 코드 워드의 길이를 증가시키기 때문에 본원에서 시프트되지 않다. 상기 코드 워드 레지스터의 이용된 비트들을, 이 시점에서 시프트하는 대신, 전체의 현재의 코드 워드는 상기 현재의 코드 워드의 디코딩이 완료되어 코드 워드 길이가 검색될 때 즉시 시프트될 것이다. 이 실시예에서, 상기 코드 워드는 단계(38)에서 시프트된다.

[0119] 상기 호스트(208)는 임시 레지스터(이하, "TEMP"라고 칭함)에 CURR_GRP를 저장한다. 비록 상기 코드 워드 레지스터는 |ACC_VALUE|+ 1 비트만큼 왼쪽으로 시프트되지 않았지만은, TEMP는 |ACC_VALUE|+ 1 비트만큼 왼쪽으로 시프트된다. 결과적으로, 비트 스트링 "1110"이 제거된다. TEMP는 이제 "11 ...."을 포함한다. (TEMP에 저장된 선두의 2개의 비트들 "11"은 단계(35)에서 필요한 말미 오프셋 비트들을 포함하며, 여기서 TEMP는 14만큼 오른쪽으로 시프트되어 상기 스트링 "11"은 오른쪽으로 자리맞춤된다.)

[0120] 34) 상기 호스트(208)는 상기 검색 레지스터를 상기 레지스터의 최하위 즉, 가장 오른편 끝에서 위치 OFFSET(즉, 비트들 4 내지 13, 여기서 비트 0은 상기 레지스터의 가장 오른편의 비트 위치임)에 시프트하고, OFFSET = +9 임을 결정한다.

[0121] 35) 상기 호스트(208)는 어드레스 *(TABLE_STARTING_ADDRESS + OFFSET + (14만큼 오른쪽으로 시프트된 TEMP)) 즉, 어드레스 *(TABLE_STARTING_ADDRESS + 9 + 3) 즉, 행 0 + 9 + 3 = 행 12 에서 테이블을 판독하여 다음의 엔트리를 수신하고,

[0122]

	0x 0a	6

[0123] 이 엔트리는 상기 검색 레지스터에 저장된다(단계(S314) -- 행 12에서의 엔트리 위치를 결정하기 위해 상기에서 이용된 3의 가수(addend)는 말미 오프셋 비트들 "11"의 값이다).

[0124] 36) 상기 호스트(208)는 상기 엔트리의 마지막 4개의 비트를 판독하여 CW_LEN = 6 임을 결정한다.

[0125] 37) 상기 호스트(208)는 상기 엔트리를 4 비트만큼 오른쪽으로 시프트하여 심볼 "0x0a"을 오른쪽 자리맞춤한다.

[0126] 38) 상기 코드 워드 레지스터는 CW_LEN 만큼 왼쪽으로 시프트하고(단계(S316)), 새로운 비트들이 존재한다면, 상기 오른쪽으로부터 새로운 비트들이 추가된다. 이 경우, 새로운 비트들이 존재하지 않는다.

[0127] 39) 수신 버퍼(206)로부터 이용가능한 새로운 비트들이 존재하지 않기 때문에(단계(S318)), 상기 수신 버퍼(206)내의 비트들의 도착에 의해 재가동될 때까지 프로세싱을 중단한다.

[0128] 표 5의 검색 테이블은 필드 식별자 "N"을 갖지 않는데, 그 이유는 표 5가 근거로 하는 표 4의 코드 워드들은 계수 다음의 임의의 코드 워드의 잔여 부분 즉, 바로 다음에 오는 본래 알려져 있는 비트와 결합된 비트 런 이외의 부분의 길이가 항상 2개 또는 보다 적은 수의 비트들이 되도록 되어 있기 때문이다. 설계에 의해, 따라서, 이들 2개 또는 보다 적은 수의 비트들은 최초 그리고 최종 반복인 것을 표 5의 소정의 말미 오프셋으로서 즉시 평가된다. 이 제 1 실시예는, 따라서, 본 발명에서 0 및 1 스트링들의 순환적 계수를 위한 가능성을 완전하게 보여주지는 않는다.

[0129] 본 발명의 제 2 실시예는 선두의 1들 및 0들을 서치하는 반복을 보여준다. 상기 제 1 실시예와 같이, 상기 제 2 실시예는 도 1 내지 도 3에 예시되어 있지만, Mp3 오디오 표준으로부터 "허프만 테이블 번호 13"을 수정함으로써 생성된 테이블에 근거한다. 상기 수정된 테이블은 하기의 표 6에 도시된다.

[0130]

표 6

심볼	코드 워드	심볼	코드 워드
1	1	45	0001001010
2	011	46	0001001001
3	0101	47	0001001000
4	0100	48	0001000111
5	001111	49	0001000110
6	001110	50	000100010
7	001101	51	000100001
8	001100	52	0001000001
9	0010111	53	0001000000
10	0010110	54	000011111
11	0010101	55	000011110
12	0010100	56	000011101
13	0010011	57	00001110011
14	00100101	58	00001110010
15	00100100	59	0000111000
16	00100011	60	0000110111
17	00100010	61	0000110110
18	0010000	62	0000110101
19	00011111	63	0000110100
20	000111101	64	000011001
21	000111100	65	000011000
22	000111011	66	00001011111
23	000111010	67	00001011110
24	000111001	68	00001011101
25	000111000	69	00001011100
26	00011011	70	00001011011
27	00011010	71	00001011010
28	000110011	72	0000101100
29	000110010	73	0000101011
30	000110001	74	00001010101
31	0001100001	75	00001010100
32	0001100000	76	0000101001
33	000101111	77	0000101000
34	000101110	78	00001001111
35	000101101	79	00001001110
36	000101100	80	00001001101
37	000101011	81	00001001100
38	000101010	82	0000100101
39	00010100	83	00001001001
40	0001001111	84	00001001000
41	0001001110	85	0000100011
42	0001001101	86	00001000101
43	0001001100	87	00001000100
44	0001001011	88	0000100001

[0131]

[0132]

심볼	코드 워드	심볼	코드 워드
89	0000100000	136	000000101110
90	0000011111	137	000000101101
91	0000011110	138	000000101100
92	00000111011	139	00000010101
93	00000111010	140	000000101001
94	00000111001	141	0000001010001
95	00000111000	142	0000001010000
96	00000110111	143	000000100111
97	00000110110	144	0000001001101
98	00000110101	145	0000001001100
99	00000110100	146	0000001001011
100	0000011001	147	0000001001010
101	0000011000	148	000000100100
102	0000010111	149	000000100011
103	000001011011	150	000000100010
104	000001011010	151	000000100001
105	00000101100	152	0000001000001
106	000001010111	153	0000001000000
107	000001010110	154	000000011111
108	00000101010	155	000000011110
109	000001010011	156	0000000111011
110	000001010010	157	0000000111010
111	00000101000	158	0000000111001
112	000001001111	159	0000000111000
113	000001001110	160	0000000110111
114	00000100110	161	0000000110110
115	00000100101	162	0000000110101
116	000001001001	163	0000000110100
117	000001001000	164	0000000110011
118	000001000111	165	0000000110010
119	000001000110	166	0000000110001
120	00000100010	167	0000000110000
121	000001000011	168	000000010111
122	000001000010	169	000000010110
123	00000100000	170	0000000101011
124	00000011111	171	0000000101010
125	000000111101	172	000000010100
126	000000111100	173	0000000100111
127	00000011101	174	0000000100110
128	00000011100	175	0000000100101
129	00000011011	176	0000000100100
130	00000011010	177	0000000100011
131	000000110011	178	0000000100010
132	000000110010	179	000000010000
133	000000110001	180	000000001111
134	000000110000	181	000000001110
135	000000101111	182	00000000110111

심볼	코드 워드
183	00000000110110
184	0000000011010
185	0000000011001
186	00000000110001
187	00000000110000
188	0000000010111
189	00000000101101
190	00000000101100
191	0000000010101
192	00000000101001
193	00000000101000
194	0000000010011
195	00000000100101
196	00000000100100
197	0000000010001
198	0000000010000
199	00000000011111
200	00000000011110
201	0000000001110
202	00000000011011
203	0000000000110101
204	0000000000110100
205	00000000011001
206	00000000011000
207	00000000010111
208	00000000010110
209	00000000010101
210	00000000010100
211	00000000010011
212	00000000010010
213	00000000010001
214	00000000010000
215	00000000001111
216	0000000000011101
217	0000000000011100
218	0000000000011011
219	0000000000011010
220	00000000001100
221	00000000001011
222	0000000000101011
223	0000000000101010
224	0000000000010100
225	0000000000100111
226	0000000000100110
227	0000000000010010
228	0000000000010001
229	0000000000010000

심볼	코드 워드
230	0000000000001111
231	0000000000001110
232	0000000000001101
233	0000000000001100
234	0000000000001011
235	0000000000001010
236	0000000000001001
237	00000000000010001
238	00000000000010000
239	0000000000000111
240	0000000000000110
241	00000000000010111
242	00000000000010110
243	0000000000001010
244	0000000000001001
245	0000000000001000
246	0000000000000111
247	0000000000000110
248	0000000000000101
249	0000000000000100
250	0000000000000011
251	0000000000000010
252	0000000000000001
253	00000000000000001
254	00000000000000001
255	000000000000000001
256	000000000000000000

[0133]

[0134] 상기 Mp3 오디오 표준으로부터의 허프만 테이블 번호 13는 각각의 코드 워드를 번호 쌍과 결합시키는 반면, 상기의 표 6은 간단함을 위하여, 각각의 코드 워드를 "1" 내지 "256"의 범위에서 선택된 각각의 심볼에 할당한다. 역시 간단함을 위하여, 공통 선두 비트들을 갖는 코드 워드들이 그룹화되어, 그들 각각의 심볼들은 연속된다. 허프만 테이블 번호 13에서와 같이, 표 6의 임의의 코드 워드에 대한 최대 길이는 19 비트이다.

[0135] 하기에 도시된 표 7 및 표 8은 이 실시예에서 다중 계수들 즉, S302 내지 S310 루프의 다중 반복들을 요구하는 코드 워드들을 디코딩하는데 총체적으로 이용되는 표 6에 근거한 다수의 디코딩 테이블들 중 2개이다. 단지 상기 2개의 표 7과 표 8만이 본원에 제공되는데, 그 이유는 이 실시예에서 스트링을 디코딩함에 있어 단지 표 7과 표 8만이 필요하기 때문이다. 표 7은 주요 디코딩 테이블이고, 표 8은 서브테이블이다. 표 7은 또한, 다중의 다른 서브테이블들을 가지며, 서브테이블들은 그들 자신의 서브테이블들을 가질 수 있다. 실행 속도에 대하여, 모든 이들 (서브)테이블들은 바람직하게는 메모리내에, 그리고 서로 밀집하게 인접하여 존재한다. 따라서, 예를 들면, 메인 테이블 다음에 서브테이블이 오고, 이 서브테이블 다음에 그 자신의 서브테이블들이 오고, 이 서브테이블들 다음에 상기 메인 테이블의 제 2 서브테이블이 오고, 그리고 이 제 2 서브테이블 다음에 그 자신의 서브테이블들이 오고, 이와 같이 된다.

표 7. 검색 테이블의 예

행 #	엔트리 식별자	심볼/오프셋 어드레스	시프트 양/코드 길이
-1	S	0x1	1
0	19		1
1	B	+20	17
2	N	T1	
3	N	T2	
4	N	T3	
5	N	T4	
6	N	T5	
7	N	T6	
8	N	T7	
9	N	T8	
10	N	T9	
11	N	T10	
12	N	T11	
13	B	+24	17
14	B	+28	18
15	S	0x FC	16
16	S	0x FD	17
17	S	0x FE	18
18		0x FF	19
19		0x 100	19
20		0x 4	4
21		0x 3	4
22		0x 2	3
23		0x 2	3
24		0x F6	16
25		0x F7	16
26		0x F8	16
27		0x F9	16
28		0x FA	16
29		0x FB	16

[0136]

표 8. 검색 서브 테이블 T2의 예

행 #	엔트리 식별자	심볼/오프셋 어드레스	시프트 양/코드 길이
-4	S	0x13	8
-3	B	+11	18
-2	B	+7	17
-1	N	T2_1	
0	6		5
1	N	T2_2	
2	B	+13	16
3	B	+21	17
4	S	0x33	10
5	S	0x34	10
6	S	0x35	9
7		0x19	9
8		0x18	9
9		0x17	9
10		0x16	9
11		0x15	9
12		0x14	9
13		0x2f	10
14		0x2e	10
15		0x2d	10
16		0x2c	10
17		0x2b	10
18		0x2a	10
19		0x29	10
20		0x28	10
21		0x32	9
22		0x32	9
23		0x31	10
24		0x30	10

[0137]

[0138] 표 7은, 이전의 실시예의 디코딩 테이블, 표 5와 달리, "엔트리 식별자"가 "N"인 엔트리들을 갖는 디코딩 테이블이며, 각각의 "N" 엔트리는 새로운 디코딩 테이블을 가리킨다. 표 7의 행 3은 예를 들면, 상기에 표 8로서 나

타낸 서브테이블 T2을 가리키는 "심볼/오프셋 어드레스" 필드를 갖는다. 표 8의 후속 행들은 다른 서브테이블들(도시되지 않음)을 가리킨다. 코드 워드를 디코딩하는 과정에서, 다른 계수, 그리고 이에 따라 루프 S302 내지 S310의 다른 반복이 필요할 때 프로세싱은 서브테이블로 분기한다.

[0139] 인코딩된 비트스트림 소스(202)로부터 수신된 것이 다음의 비트스트림 "0001000111...."이고, 여기서 말미의 점들은 상기 비트스트림에서 다음에 올 비트들을 표시한다고 본원에서 가정한다.

[0140] 다음의 스트링 0001000111의 디코딩은 다음과 같이 진행한다.

[0141] 1) CURR_GRP = 0001000111...(최대 19 비트까지, 이 비트는 표 6이 19 비트의 최대 코드 길이를 갖기 때문에 이 실시예에서 서치 필드 길이임)

[0142] 2) ACC_VALUE = 3(도 3을 참조하면, 단계들(S302 및 S304)), 그 이유는 높은 자리의 스트링이 3 비트 즉, "000"으로 이루어져 있기 때문이다. ACC_VALUE는, ACC_VALUE가 양인지 혹은 음인지에 따라서, ZEROS_MAX 또는 ONES_MAX에 대해 유효화된다. ACC_VALUE는 이 실시예에서 양의 값을 갖기 때문에, 유효화를 위해 ZEROS_MAX가 이용된다. ZEROS_MAX과 ONES_MAX는 이 실시예에서의 경우와 같이, 그리고 이전의 실시예에서의 경우와 같이 일반적으로 양의 브랜치 양과 음의 브랜치 양과 각각 동일하게 설정된다. 상기 양의 브랜치 양은 디코딩 동안에 필요한 가장 큰 0 비트 계수와 동일하게 설정된다. 표 7의 심볼 "255"가 19개의 0들로 이루어져 있기 때문에, 상기 양의 브랜치 양은 행 0에 나타낸 바와 같이, 19로 설정된다. 이에 따라, ZEROS_MAX = 19 이다.

[0143] ZEROS_MAX가 19이고, ACC_VALUE가 3이기 때문에, ACC_VALUE =< ZEROS_MAX 이고, 따라서, ACC_VALUE는 범위내에 있다.

[0144] 3) 현재의 표 7의 위치는 TABLE_STARTING_ADDRESS(행 0)이다. 표 7에서의 ACC_VALUE = 3 의 오프셋이 현재의 위치로부터 만들어져서, 행 3에 도달한다. 그 다음, 행 3의 내용들이 검색된다. 심볼로 표시하여, *(TABLE_STARTING_ADDRESS + ACC_VALUE)는

[0145]

N	T2

[0146] 이고, 호스트(208)는 이 엔트리를 검색 레지스터 및 작업 레지스터내로 복사한다.

[0147] 4) 상기 호스트(208)는 상기 작업 레지스터를 시프트함으로써 상기 엔트리 식별자를 검사하여, 상기 식별자가 N인 것으로 결정함으로써, TABLE_STARTING_ADDRESS + OFFSET 에 위치한 새로운 테이블로의 브랜치가 발생할 것임을 의미한다. 상기 식별자가 N이기 때문에, 다른 반복이 필요하다(단계(S308)).

[0148] 5) 상기 호스트(208)는 상기 검색 레지스터의 엔트리 식별자 필드를 시프트하여, 값 "T2"을 포함하는 제 2 필드를 남겨둔다. 이 제 2 필드는 이전에 언급한 제 3 타입으로 되어 있다. 즉, TABLE_STARTING_ADDRESS 로부터 새로운 테이블 즉, 서브테이블 T2(표 8임)로의 오프셋을 포함한다. (표 8의 행 -1 및 행 1이 "심볼/오프셋 어드레스" 필드에 "T2_1" 및 "T2_2"를 각각 포함함을 주목한다. "T2_X"는 서브테이블 T2의 서브테이블의 어드레스 즉, 표 8의 서브테이블의 어드레스이다.)

[0149] 6) 빠른 검색을 위해 예를 들면, 상기 호스트 프로세서(208)에 의해 실행되는 프로그램의 선언된 파라미터로서 TABLE_STARTING_ADDRESS 이 이미 저장되어 있지 않은 경우, ADDRESS_SAVE에 TABLE_STARTING_ADDRESS를 세이브한 후에, TABLE_STARTING_ADDRESS = TABLE_STARTING_ADDRESS + T2를 갱신한다. 이제, TABLE_STARTING_ADDRESS는 표 8을 가리키며, 이 표 8은 표 8을 호출하는데 이미 이용되었던 선두의 코드 워드 비트들인 "0001"로 시작하는 코드 워드들에 대해 생성된다.

[0150] 7) TEMP(CURR_GRP와 같이, 이 실시예에서 길이가 19 비트임)가 CURR_GRT와 함께 적재되어, |ACC_VALUE|+ 1 비트만큼 왼쪽으로 시프트된다. 이에 의해, 이미 알려져 있는 4 비트가 시프트되게 된다.

[0151] 8) TEMP = "000111..."에서의 잔여 비트들은 이제 가속기(216)에 입력되는 CURR_GRP(단계(S312))를 포함한다.

[0152] 9) 길이 3의 높은 자리 비트 런으로 인해, ACC_VALUE = 3 이다(단계들(S302 및 S304)).

[0153] 10) 상기 호스트(208)는 이제 표 8의 행 0 엔트리에 대응하는 *(TABLE_STARTING_ADDRESS), 즉

[0154]

0	6	5

[0155] 를 검색한다. 상기 호스트(208)는 이제 ACC_VALUE와의 비교를 위해 양의 브랜치 계수를 갖는다.

[0156] 11) ACC_VALUE는 범위내에 있다(그 이유는 상기 양의 브랜치 계수가 6이기 때문임).

[0157] 12) 상기 호스트(208)는 *(TABLE_STARTING_ADDRESS + ACC_VALUE)를 판독하여,

[0158]

B	+21	17

[0159] 를 찾고, 상기 호스트(208)는 이 엔트리를 상기 검색 레지스터 및 상기 작업 레지스터내로 복사한다.

[0160] 13) 상기 호스트(208)는 상기 작업 레지스터를 시프트하여 이 엔트리를 "B" 엔트리로서 식별한다. "B" 엔트리에 대하여, 다른 반복은 필요하지 않고(단계(S308)), 현재의 코딩된 워드가 아직 디코딩되지 않았다(단계(S312)). 상기 호스트(208)는 상기 검색 레지스터의 "시프트 양/코드 길이"내의 값을 검출한 후에, 상기 검색 레지스터를 시프트함으로써, 비트들 4 내지 13(즉, OFFSET에 의해 표시된 심볼/오프셋 어드레스 필드)을 결정한다.

[0161] 14) 상기 호스트(208)는 CURR_GRP 즉, 코드 워드 레지스터의 내용들을 TEMP에 복사하여, TEMP를 |ACC_VALUE| + 1 비트만큼 왼쪽으로 시프트한다. TEMP는 이제 비트 스트링 "11....."을 포함한다. TEMP를, 이 실시예에서 "시프트 양/코드 길이"에 특정된 비트들의 수인 17 비트만큼 오른쪽으로 시프트하여, TEMP의 상기 비트 스트링 "11"을 오른쪽 자리맞춤한다. 이러한 최종 반복에서, 스트링 "11"의 값은 표 8로의 소정의 말미 오프셋으로서 작용한다.

[0162] 15) 상기 호스트(208)는 *(TABLE_STARTING_ADDRESS + OFFSET + (상기의 단계(14)에 특정된 바와 같이 SHIFT만큼 오른쪽으로 시프트한 후의 TEMP)) 즉, 행(0+21+3=24)의 내용들을 판독하여, 다음의 엔트리 즉,

[0163]

	0x30	10

[0164] 를 찾고, 상기 호스트(208)는 이 엔트리를 상기 검색 레지스터내로 복사한다(단계(S314)).

[0165] 16) 상기 호스트(208)는 4개의 최하위 비트들로부터 CL_LEN = 10을 추출한다.

[0166] 17) 상기 호스트는 상기 엔트리를 4 비트만큼 오른쪽으로 시프트하고, 16진수로 값이 "30" 또는 10진수로 48인 출력 심볼을 수신한다. 표 6에서 알 수 있는 바와 같이, 상기 출력 심볼 "48"은 기대했던 바와 같이, 이 실시예에서 "0001000111"인 디코딩된 코드 워드에 대응한다.

[0167] 18) CURR_GRP는 CL_LEN만큼 왼쪽으로 시프트된다(단계(S316)). 현재 수신 버퍼(206)에 임의의 비트가 존재한다면, 추가적인 새로운 비트들이 19 비트의 서치 필드 길이까지 코드 워드 레지스터를 채운다.

[0168] 19) ADDRESS_SAVE로부터 TABLE_STARTING_ADDRESS를 재저장한다.

[0169] 20) 디코딩은 코드 스트림이 완료되지 않은 경우 계속한다(단계(S318)).

[0170] 특정 비트 시퀀스로 시작하는 코드 워드들 중 임의의 것이 서브테이블의 이용을 요구한다면, 모든 이러한 코드 워드들은 디코딩 동안 그 서브테이블을 요구할 것임을 주목할 필요가 있다. 예를 들면, 만약 말미 임계가 적어도 3인 경우, 코드 워드 "0001100"는 선두 서브스트링 "000"에 대해 3의 계수를 결정하고, 그 후속 비트를 무시하고, 그리고 말미 오프셋으로서 마지막 3개의 비트들 "100"을 이용함으로써 디코딩된다. 더 많은 정보가 없이도, 이 경우에는, "B" 엔트리를 가리키기 위해 3의 계수가 메인 디코딩 테이블로 오프셋하여, 이것에 의해 상기 말미 오프셋이 이용될 수 있게 됨으로써, 출력 심볼이 얻어지고, 어떠한 서브테이블도 필요없게 되도록 메인 디코딩 테이블을 설계할 수 있다. 그러나, "001"로 시작하는 다른 코드 워드가 서브테이블을 요구하는 경우, 선두 스트링 "001"은 필요한 서브테이블을 참조하는 요구된 "N" 엔트리 보다는 "B" 엔트리에 이르게 된다. 일례로 "00101100"이 있다. 이 코드 워드는 1 비트 런 계수를 수반하는 서브스트링 "001" 으로 시작하지만, 상기 코드 워드는 각각의 후속하는 비트 런 계수에 대한 서브테이블을 요구한다.

[0171] 해결책은 선두 서브스트링 "001"이 "B" 엔트리가 아닌 "N" 엔트리를 가리키도록 메인 디코딩 테이블을 설계하여, 서브테이블로 분기되게 하여 이 지점에서 프로세싱을 처리하도록 구성하는 것이다. 따라서, 임의의 코드 워드에 대한 선두 서브스트링 "001"은 프로세싱이 메인 디코딩 테이블에서 "N" 엔트리를 가리키도록 한다.

[0172] 허프만 코드 워드들의 디코딩 동안, 양의 브랜치 계수와 음의 브랜치 계수는 비트 런 계수들을 유효화하는데 반복적으로 이용되지만, 특정 디코딩 테이블에 대해 고정되어 있다. 따라서, 디코딩의 시작에서, 양의 브랜치 계

수와 음의 브랜치 계수는 바람직하게는 특별한 CPU 레지스터들에서와 같이, 고속 억세스를 위해 저장되어, 상기 테이블로부터 그것들을 반복적으로 검색하는 오버헤드(overhead)를 피하게 된다.

[0173] 상기 양의 브랜치 계수와 음의 브랜치 계수는 계수에 필요로 되는 연속적인 0 또는 1 비트들 각각의 최대수 이외의 일부 길이로 한정될 수 있다. 예를 들면, 19 비트를 검사하는 대신, 단지 처음 16 비트만을 검사한다. TABLE_STARTING_ADDRESS + (ZEROS_MAX 또는 ONES_MAX)에서의 테이블 엔트리는 이 경우, 16 비트보다 긴 코드 워드들의 잔여 비트들을 다루는 새로운 테이블에 대한 포인터를 포함한다. 그러나, 디코딩 시간을 최소로 하기 위해서, 대다수의 카운트될 비트 런들은 디코더 윈도우에 맞춰야 하는데, 이 디코더 윈도우는 제 1 실시예에서는 16 비트이고 제 2 실시예에서는 19 비트이다.

[0174] 말미 한계의 길이는 설계상의 문제이다. 상기 제 1 디코딩 실시예에서, 상기 길이는 2 비트이다. 비록 1 비트가 다른 디코딩 테이블들에 이득이 될 수 있지만은, 1 비트를 이용하면, 메모리를 세이브하지 못하지만, 귀납적 단계들을 더 요구함으로써, 디코딩 시간이 길어지게 된다. 상기 말미 한계를 3으로 증가시키면, 상기 제 1 실시예에 이득이 없으며, 모든 서브트리들이 (0/1 스트링들을 프로세싱한 후에) 2 비트의 최대 잔여 코드 워드 길이를 갖기 때문에, JPEG 또는 MPEG 표준에서 이용되는 것들과 같은 더 복잡한 테이블들에 대해 3 또는 4와 같은 다른 한계들이 이용될 수 있고, 예를 들면, 제 2 실시예에 대해 3의 말미 한계가 이용되었다. 더 많은 비트들은 더 빠른 서치를 할 수 있지만, 여러개의 잉여 메모리 위치들을 갖는 디코딩 테이블들을 발생시킬 수 있다. 예를 들면, 비록 상기 위치들 중 단지 3개만이 코드 워드들에 의해 점유되는 경우가 될 수 있지만은, (3의 말미 한계에 대한) 3개의 잔여 비트들은 8개의 메모리 위치들을 요구한다.

[0175] 본원에 나타낸 디코딩 테이블들의 필드 크기들은 가변적이어서, 더 긴 디코딩 테이블 행들이 요구되는 경우, 상기 필드들의 상대적인 길이들은 조정될 수 있거나 더 긴 워드 길이가 이용될 수 있다. 10 비트의 심볼/오프셋 어드레스 필드 크기는 RAM 224에서 1024개의 메모리 위치들을 어드레스할 수 있으며, 이것은 제 1 또는 제 2 실시예에 대해 매우 충분하다. 상기 제 2 실시예에서, 예를 들면, 표 6은 256개의 심볼들을 포함하는 반면, 단지 50%의 효율성에서만, 참조될 필요가 있는 512개의 서로 다른 메모리 위치들이 존재한다. 상기 제 1 실시예에서, 표 4는 단지 32개의 심볼들을 가지며, 10 비트 메모리 어드레스들에 의해 쉽게 조절된다.

[0176] 상기 시프트 양/코드 길이 필드는 상기 제 1 실시예에서 4비트 길이이므로, 길이가 16 비트까지인 코드 워드들을 처리할 수 있으며, 반면, 상기 제 2 실시예는 5 비트 길이 시프트 양/코드 길이 필드를 이용하는데, 그 이유는 최대 코드 워드의 비트 길이가 19 이고, 이것은 5 비트로 표시되기 때문이다. 상기 디코딩 테이블들에서의 결과적인 전체 엔트리 길이는 17 비트이고, 여기서, 2 비트는 엔트리 식별자 필드를 위한 것이고, 10 비트는 심볼/오프셋 어드레스 필드를 위한 것이고, 그리고 5 비트는 시프트 양/코드 길이 필드를 위한 것이다. 대안적으로, 상기 전체 엔트리 길이는, 심볼/오프셋 어드레스 필드에서 1 비트를 없애고, 이에 따라 일부 서브테이블들이 메모리내의 비-상주(non-resident) 및 교환가능하게 하는 것에 의해 RAM 메모리 224를 줄임으로써 16 비트로 이루어질 수도 있다.

[0177] 검색 테이블(들)은 특정 허프만 트리에 대해 고정될 수 있다. 대안적으로, 인코딩된 비트스트림에서 선두하는 허프만 트리에 근거한 디코딩의 시작에서 "온더플라이(on the fly)"를 발생시킬 수 있거나, 또는 예를 들면, 적응 허프만 코딩 방식을 수용하기 위해서와 같이, 필요에 따라 갱신될 수 있다.

[0178] 비록 상기 도시된 실시예들이 "B" 엔트리들을 갖는 디코딩 테이블들을 가지며 말미 오프셋들을 이용하지만은, 본 발명의 범위는 디코딩 테이블이 "B" 엔트리들을 갖거나 말미 오프셋들이 이용되는 것을 요구하지는 않는다. 디코딩 테이블은 예를 들면, 단지 "S" 및 "N" 엔트리들만을 가질 수 있으며, 여기서 "S" 엔트리는 디코딩된 출력 심볼을 포함하는 반면, "N" 엔트리는 후속 디코딩 테이블을 가리킨다. 동작중에, 현재의 코드 워드내의 각각의 비트 런은 각각의 계수를 야기시키고, 이 계수로부터 호스트 프로세서(208)는 상기 현재의 테이블로의 각각의 오프셋을 결정한다. 그 오프셋에서, 각각의 "N" 엔트리는 각각의 후속 테이블을 가리키며, 최종 비트 런에 근거하여, 호스트 프로세서(208)가 현재의 테이블로 오프셋하여, 디코딩 결과를 포함하는 "S" 엔트리에 도달할 때까지 후속 비트 런에 대해 프로세스를 반복한다.

[0179] 본 발명은 바람직하게는 가속기(216)와 함께 실시되어, CPU가 수행하기 힘든 0/1 스트링의 길이 찾기, 비트 시프트 및 비교와 같은 어려운 비트 조작 기능들을 돕는다. 예를 들면, 상기에 논의된 바와 같이, 스트림에서 최초의 "1"을 찾는 것은 소프트웨어를 이용하면 어렵지만, 하드웨어를 이용하면 쉽게 처리된다.

[0180] 도 4는 허프만 디코딩 장치(400)에서 본 발명에 따른 선두의 0/1 계수 계산기 또는 가속기(404)의 하나의 가능한 실시예를 예시한다. 본원에서는 8 비트 서치 윈도우에 대한 동작이 설명되는바, 검출될 비트들은 레지스터

(412)에서의 호스트 CPU(410)에 의한 비트들의 현재의 그룹(408)의 부분으로서 상기 가속기(404)에 주어진다. 서치될 최초의 비트는 상기 레지스터의 MSB 끝에 있다. 유입 비트 슬라이스의 상기 MSB 또는 선두 비트(414)는 값 검출기(417)를 갖는 선택기(416)에 의해 검사되어 "모두-0" 또는 "모두-1" 경로를 향해 프로세싱이 되는지를 결정한다. 사전 선택된 비트 값 예를 들면, 1이 상기 MSB(414)로서 검출된다면, 제 1 인버터(418)가 상기 레지스터(412)의 출력을 비트 단위로 반전시키고, 그렇지 않고 상기 MSB(414)로서 0이 검출된다면 반전이 발생하지 않는다. 어느 한 경우에, 선택기(416)가 상기 사전 선택된 값(본원에서는 1임)으로, 디지털 확장된 출력(420)에서 상기 사전 선택된 값을 갖는 최상위 비트의 그것보다 낮은 중요도의 다른 값을 갖는 모든 비트를 변환하는 1 확장기(419)와 같은 디지털 확장기에 그 결과를 넘겨준다. 제 2 인버터(422)가 상기 출력(420)을 반전시킨다.

[0181] 이 제 2 반전된 결과는 반전기(424)에 의해 반전되어, 비트 x_L이 x_0로 교환되고, x_{L-1}이 x_1로 교환되고, 등등도 이와 같이 한다. 여기서, x_L은 최상위 비트이고 x_0은 최하위 비트이다. 이 반전된 결과 즉, 반전된 스트링은 2진 코딩된 디지털 변환에 대한 지표(thermometer)를 행하는 지표 코드 평가기(426)에 보내짐으로써, 모든 0들/1들 런의 길이를 결정한다. 지표 코드는 한쪽 끝에서 모든 동일한 값을 갖는 비트들을 갖는 비트 스트링이며, 상기 동일한 값을 갖는 비트들의 수는 상기 코드 즉, 4의 값을 갖는 4개의 말미 1들을 갖는 지표 코드의 값을 표시한다. 선두 비트(414)가 사전 선택된 값을 갖는다고 상기 값 검출기(417)에 의해 검출된다면, 상기 지표 코드 평가기(426)의 출력이 부정기(negator)(430) 및 선택기(432)를 포함하는 런 캐릭터라이저(run characterizer)(428)에 의해 부정된다. 따라서, 상기 런 캐릭터라이저(428)는 상기 지표 코드 평가기의 출력을 선택적으로 재포맷팅함으로써, 부호 확장하는 단계를 포함하여, 상기 호스트 CPU(410)에 의해 이러한 확장이 요구되는 경우, 그 출력을 상기 호스트 CPU(410)의 원래 길이로 부호 확장한다.

[0182] 최종 결과는 현재의 그룹(408)의 시작에서 모두 1들/0들 런의 길이를 제공하는 BCD 번호 즉, 상기 그룹(408)에 대한 현재의 계수이다. 상기 최종 결과는 선두 비트(414)가 1이면 음이고, 선두 비트(414)가 0이면 양이다.

[0183] 상기 가속기(404)에 이용될 수 있는 하드웨어의 예가 하기에 나타나 있다(16 비트 서치 필드가 이용되는데, 그 이유는 16 비트가 CPU의 공통된 음의 워드 길이이기 때문이다).

[0184] 1) 서치될 데이터를 저장하기 위한 16 비트 레지스터(412)

[0185] 2) 반전이 필요한 경우 상기 데이터를 반전시키기 위한 16개의 NOT 게이트들

[0186] 3) 32-대-16 선택기(416)

[0187] 4) 최초로 검출된 1을 오른쪽으로 확장하기 위한 하드웨어

[0188] 5) 이 결과를 반전시키기 위한 16개의 NOT 게이트들

[0189] 6) 16 비트를 반전시키기 위한 하드웨어

[0190] 7) BCD 컨버터에 대한 지표(16-대-4 BCD 인코더)

[0191] 8) 음의 2의 보수 컨버터(부정기(430))에 대한 BCD

[0192] 9) 직접 또는 2의 보수 부정된 결과를 선택하기 위한 32-대-16 선택기(선택기(432))

[0193] 10) 16 비트의 출력 버퍼 레지스터

[0194] "1들에 대해서는 음, 0들에 대해서는 양"의 규정은 필수가 아니다. 정보는 일부 다른 형태로 CPU에 전달될 수 있다. 상기 실시예들에서, 가속기 출력의 포맷은 ACC_VALUE와의 비교를 위한 ONES_MAX 또는 ZEROS_MAX를 선택하기 위해 현재의 계수와 베이시스(basis) 둘다를 제공한다. 그러나, 상기 결과의 MSB 끝의 모든-1들 정보 및 LSB 끝의 모든-0들을 전달하는 것과 같은 여러 다른 방법들이 존재하며, 이것은 지표 코드 가속기(426)에 대한 2의 보수 변환에 대한 BCD에 대한 요구 및 대응하는 요구를 세이브한다.

[0195] 대안적인 실시예에서, 호스트와의 인터페이스는 상기 가속기(404)가 상기 호스트(410)의 코어에의 엔트리를 허용받은 경우, 상기 호스트가 이미 갖고 있는 레지스터들을 통해 수행될 수 있다.

[0196] 비록 상기의 실시예들이 허프만 인코딩된 코드 워드들에 대하여 설명되었지만은, 본 발명의 범위는 다른 가변 길이 코드 워드들을 포함한다.

[0197] 따라서, 본 발명의 바람직한 실시예에 적용될 수 있는 기본적인 독창적인 특징들이 도시되어 설명되고 지적되어 있지만은, 예시된 디바이스들의 형태 및 세부항목 및 그의 동작에 있어서의 다양한 생략 및 치환 및 변경이 본

발명의 정신으로부터 벗어남이 없이 이 기술분야의 당업자들에 의해 이루어질 수 있음을 이해할 수 있을 것이다. 예를 들면, 동일한 결과들을 달성할 수 있는 실질적으로 동일한 방법으로 실질적으로 동일한 기능을 수행하는 그들의 요소들 및 방법 단계들의 모든 조합들이 본 발명의 범위내에 있도록 명시적으로 의도된다. 또한, 본 발명의 임의의 개시된 형태 및 실시예와 결합하여 도시 및/또는 설명된 구조들 및/또는 요소들 및/또는 방법 단계들이 설계 선택의 일반적인 문제로서 임의의 다른 개시 또는 설명 또는 제안된 형태 또는 실시예로 통합될 수 있음을 인식해야 한다. 따라서, 본 발명은 단지 본원에 첨부된 청구의 범위에 나타낸 바와 같이만 한정될 수 있다.

도면의 간단한 설명

[0045] 도 1은 허프만 트리의 선도이고;

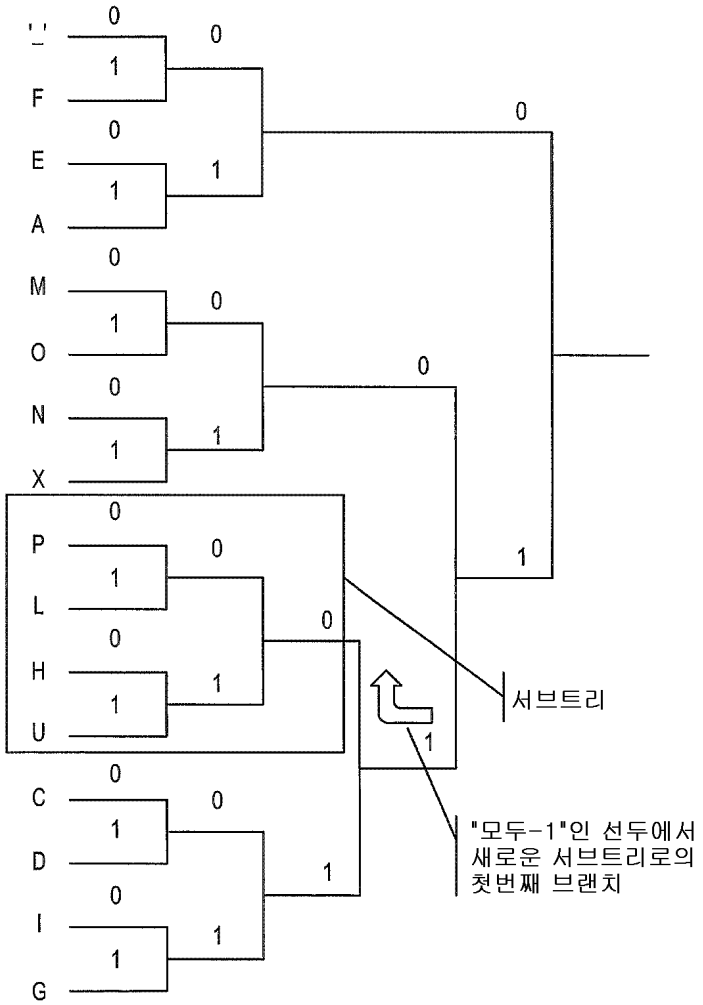
[0046] 도 2는 본 발명에 따른 예시적인 디코딩 시스템이고;

[0047] 도 3은 본 발명에 따른 허프만 인코딩된 코드 워드들을 디코딩하는 프로세스의 흐름도이고; 그리고

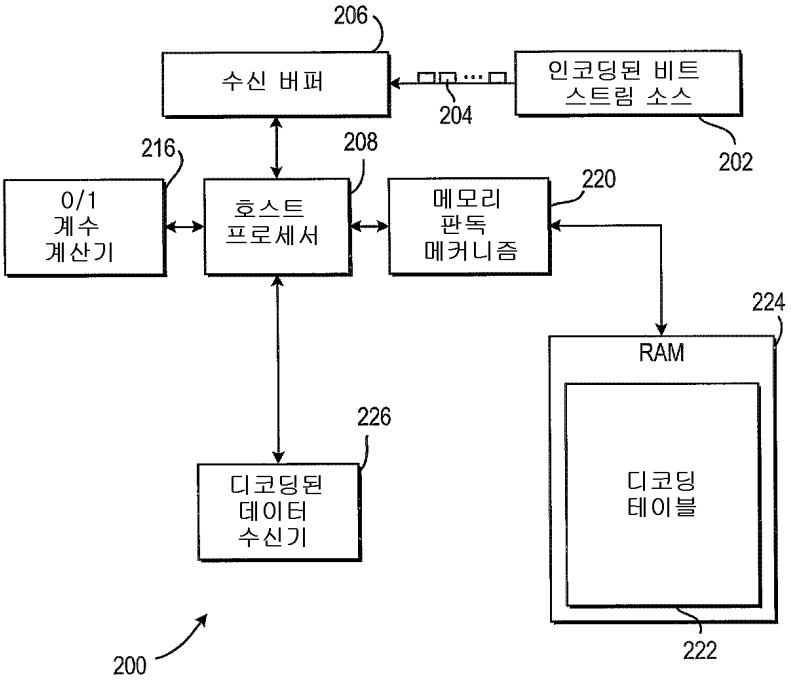
[0048] 도 4는 본 발명에 따른, 도 2의 디코딩 시스템의 구성요소의 선도이다.

도면

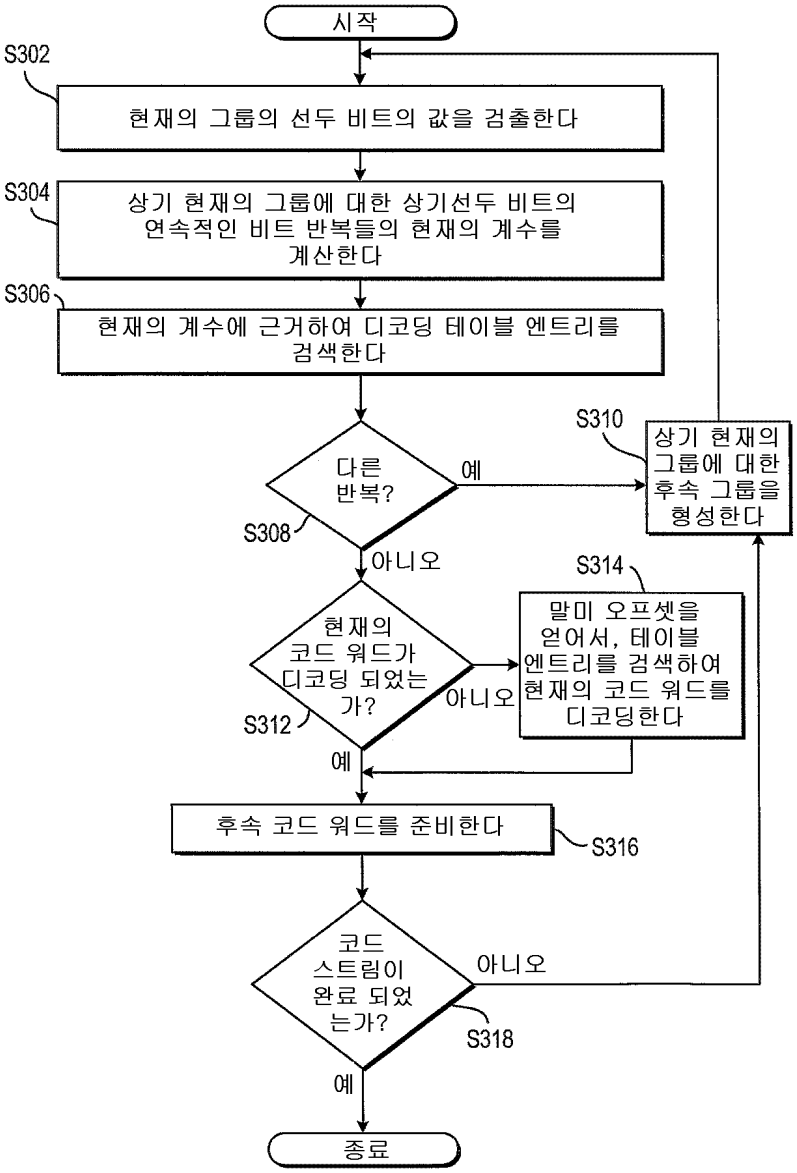
도면1



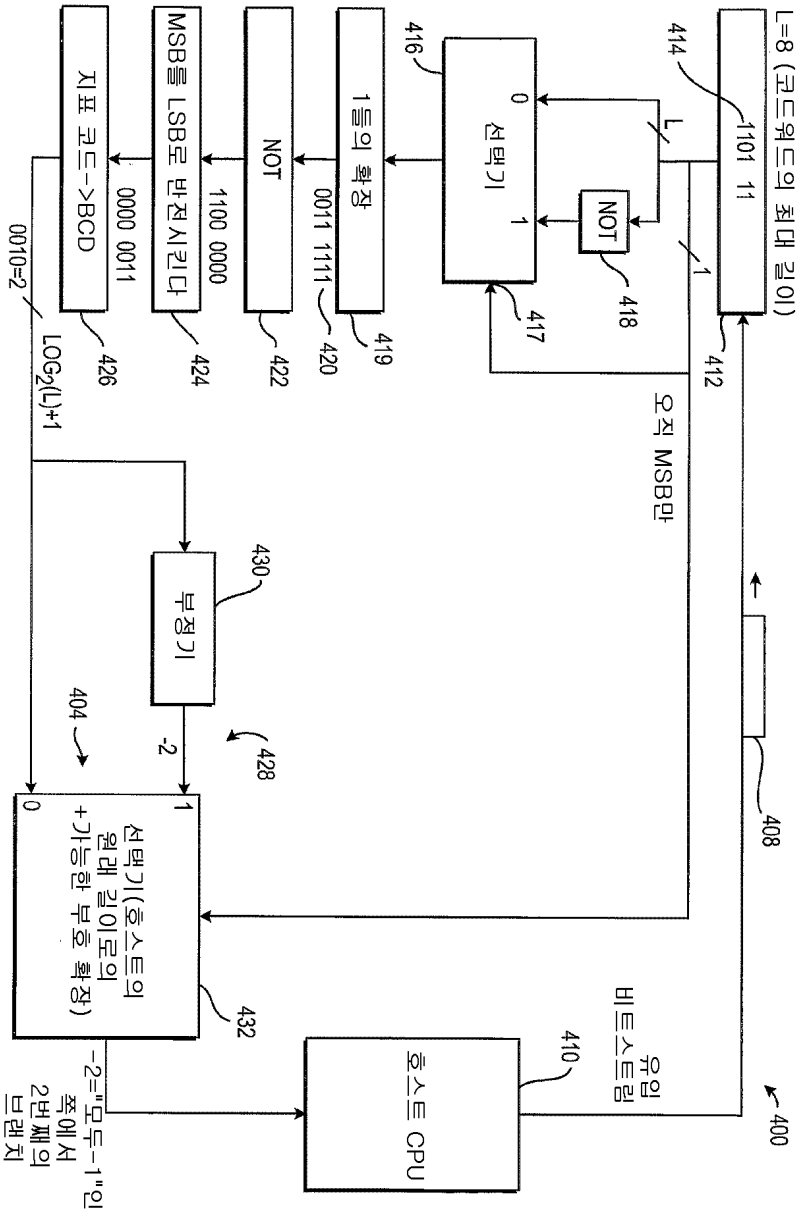
도면2


206
수신 버퍼
204
인코딩된 비트
스트림 소스
202
216
0/1
계수
계산기
208
호스트
프로세서
220
메모리
판독
메커니즘
224
RAM
디코딩
테이블
222
226
디코딩된
데이터
수신기
200

도면3


시작
S302
현재의 그룹의 선두 비트의 값을 검출한다
S304
상기 현재의 그룹에 대한 상기선두 비트의 연속적인 비트 반복들의 현재의 계수를 계산한다
S306
현재의 계수에 근거하여 디코딩 테이블 엔트리를 검색한다
다른 반복?
예
S310
상기 현재의 그룹에 대한 후속 그룹을 형성한다
S308
아니오
S314
현재의 코드 워드가 디코딩 되었는가?
아니오
말미 오프셋을 얻어서, 테이블 엔트리를 검색하여 현재의 코드 워드를 디코딩한다
S312
예
후속 코드 워드를 준비한다
S316
코드 스트림이 완료 되었는가?
아니오
S318
예
종료

도면4


L=8 (코드워드의 최대 길이)
1101 11
414
412
L
1
NOT
418
0
선택기
1
416
417
오직 MSB만
408
400
410
유일
비트스트림
호스트 CPU
1들의 확장
0011 1111
420
419
NOT
1100 0000
422
MSB를 LSB로 반전시킨다
424
지표 코드->BCD
0000 0011
426
0010=2
LOG2(L)+1
부정기
430
-2
428
404
선택기(호스트의
원래 길이로의
+가능한 부호 확장)
0
1
432
-2="모두-1"인
쪽에서
2번 째의
브랜치