(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) Int. Cl.[6]
G06T 1/00

(45) 공고일자 2002년06월20일
(11) 등록번호 10-0331351
(24) 등록일자 2002년03월22일

(21) 출원번호	10-1998-0701282	(65) 공개번호	특1999-0044049
(22) 출원일자	1998년02월21일	(43) 공개일자	1999년06월25일
번역문제출일자	1998년02월21일		
(86) 국제출원번호	PCT/US1996/12719	(87) 국제공개번호	WO 1997/09665
(86) 국제출원일자	1996년08월05일	(87) 국제공개일자	1997년03월13일
(81) 지정국	국내특허 : 아일랜드 오스트레일리아 캐나다 일본 EP 유럽특허 : 오스트리아 벨기에 스위스 독일 덴마크 스페인 프랑스 영국 그리스 이탈리아 룩셈부르크 모나코 네덜란드 포르투칼 스웨덴		

(30) 우선권주장 518031 1995년08월22일 미국(US)

(73) 특허권자 휴렛-팩커드 컴퍼니(델라웨어주법인) 파트릭 제이. 바렛트
미합중국 캘리포니아주 (우편번호:94304) 팔로 알토 하노버 스트리트 3000
(72) 발명자 프래져 알렌 엘
미국 켄사스주 67230 위치타 윌로우 벤드 코트 14602
브래드번 브렌트 엠
미국 켄사스주 67206 위치타 비츠 로드 12
(74) 대리인 김창세, 장성구

심사관 : 전상현

(54) 이진입력데이터스트림압축방법

명세서

기술분야

<1> 본 발명은 전반적으로 데이터 압축 분야에 관한 것으로서, 더욱 구체적으로는, 무손실 이미지 데이터 압축 및 압축해제(lossless image data compression and decompression)에 관한 것이다.

배경기술

<2> 이미지 데이터 정보의 전송 및 저장은 오늘날의 컴퓨터화된 세상에서 그 중요성이 날로 증가하고 있다. 이미지 스캐너, 사진복사 머신(photocopy machine), 전화 팩시밀리 머신 등은 한장의 인쇄된 텍스트(text)나 도면과 같은 이미지를 디지털화한 후 그 이미지를 비트맵으로서 저장 또는 전송한다.

<3> 이렇게 이미지 데이터를 디지털화할 때에는, 처리 속도를 증가시키기 위해서, 필요한 메모리 저장량이나 전송 대역폭을 줄이기 위해서, 저장된 이미지의 해상도를 높이기 위해서, 또는 이 중 두 가지 이상을 목적으로, 데이터를 압축하는 것이 바람직하다. 이미지 데이터는 막대한 양의 중복 정보(redundant inforamation)를 포함하는 것이 보통이므로, 종종, 중복 정보의 상당량을 제거함으로써, 이미지를 표현하는 데 필요한 비트수를 줄이는 것이 가능하다.

<4> 압축은 중복 정보를 제거하는 프로세스이다. 압축해제는 압축된 데이터로부터 원래의 데이터를 복원하는 프로세스이다.

<5> 기존의 기법들은 여러 가지로 응용가능한 다수의 유용한 데이터 압축을 제공한다. 예컨대, MPEC이라고 알려진 ISO/IEC 표준 11172-3은 컬러 비디오 동화상(moving color video image)용으로 화질(picture quality) 및 압축율(compression rate)의 밸런스를 맞추도록 최적화되었다. 이것은 "손실" 압축 기법으로서, 적어도 몇몇 상황의 정보들이 인코딩 프로세스 도중 손실되어 버려, 이미지를 완전히 정확하게 복원할 수 없게 된다.

<6> "LZ" 시스템으로 알려진 무손실 압축 기법이 렘펠(Lempel) 및 지브(Ziv)에 의한 "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Volume IT-23, NO. 3, May 1977에 기술되어 있다. LZ 시스템은 가변 길이의 입력 데이터 심볼 스트링(string)들을 나타내는 고정 길이 코드 값들을 사용한다.

<7> 이 시스템은 단일 코드 값을 사용하여 가능한한 많은 데이터 심볼들을 표현하고자 한다. LZ 시스템은 변환 테이블, 히스토리 어레이(history array), 혹은 사전 등과 같이 다양하게 알려진 테이블-코드 심볼들의 블럭들은 자신이 처리될 히스토리 어레이내에 저장되어 있음-을 사용한다. 동일한 데이터

값들의 블럭들의 장래 어커런스들은 그 블럭이 저장되어 있는 히스토리 테이블내에서의 위치를 지정하는 인덱스로 대체될 수 있다.

<8> 따라서, LZ 알고리즘은 "적응성(adaptive)"이 있어, 히스토리 어레이가 구축 및 수정될수록 점점 더 큰 압축율을 얻을 수 있게 된다. LZ 알고리즘에 대한 변형예가 웽(Weng)에게 허여된 미국 특허 제 4,881,075 호에 기술되어 있다. 이 시스템은 2 개의 변환 테이블을 사용하는데, 하나는 데이터를 압축 및 압축해제하는 데 사용되며, 다른 하나는 구축에 사용되어, 인코딩이 항상 전체 히스토리 테이블을 가지고 수행되도록 한다.

<9> 사전-기반(dictionary-based) 인코딩 방법에 있어 중요한 문제점 중 하나는 이러한 기법들이 대개의 경우 막대한 양의 메모리를 필요로 한다는 것이다. 예컨대, 웽의 압축기에 대한 바람직한 실시예에서는 4096 개의 엔트리(entry)를 갖는 사전을 사용한다. 사전이 엔트리를 많이 포함하고 있지 않거나, 입력 스트림이 가능한 심볼들 및 심볼 조합들을 조금 포함하고 있다면, 결과적으로 압축율도 작은 값, 즉, 최소 값으로 될 것이다.

<10> 큰 사전들을 하드웨어로 구현하는 경우에는, 대용량의 메모리가 필요하게 되어, 회로가 차지하는 면적(circuit real estate)과 탐색 시간 모두에 있어서 비용 부담이 크게 된다. 소프트웨어로 큰 사전을 구현하는 것 역시, 액세스 시간이 길어진다고 하는 문제가 있다.

<11> 따라서, 이미지 데이터에 대해 고압축율을 얻을 수 있고, 사전을 사용하는 경우에도 비교적 작은 사전을 가지고 구현할 수 있는 무손실 이미지 압축 기술이 요망되고 있다.

<12> 발명의 개요

<13> 본 발명은 고압축율을 얻을 수 있고, 사전을 사용하는 경우에도 비교적 작은 사전만으로 구현가능하며, 인쇄된 텍스트, 선도(line drawing), 하프톤(half tone)들을 나타내는 이미지 데이터를 압축하는 데 특히 적합한, 무손실 데이터 압축 방법 및 장치를 제공하는 것이다.

<14> 본 발명의 하나의 특징은, 인코더가 복수의 코딩 단계(phase)－각각의 스테이지(stage)는 이전 단계로부터 출력된 데이터를 압축하는 데 특히 적합함－들을 갖는다는 것이다. 제 1 단계는 비트-런 길이 코딩(bit-run length coding)이다. 입력 데이터 스트림을 검사(examine)하여, 연속된 값(value)들로 이루어진 런(run)들을 그 런내에 있는 값들의 개수로 바꾼(replace) 다음, 이렇게 압축된 데이터를 더 압축하기 위해 제 2 단계로 전달한다.

<15> 제 2 단계는 페어 반복("pair-rep") 코딩이다. 한 쌍의 값들(페어)이 연속해서 반복되고 있으면, 이를 그 반복되고 있는 페어의 복사(copy)와 복사 카운터(repetition counter)로 바꾼다.

<16> 본 발명의 다른 특징은 사전-기반 인코딩 단계의 구현에 있다. 이 인코딩 단계는 다단계 인코더(multiple phase encoder)의 제 3 단계이자 선택사양적인 단계이다. 이 제 3 단계에서는, 적응적 사전을 사용하여 남아있는 중복부분(redundancy)들을 더 제거한다.

<17> 인코딩의 처음 두 단계들에서 초기 데이터 스트림내에 존재하는 대부분의 중복부분들이 추출되기 때문에, 마이크로-테이블이라고 불리는 매우 작은 테이블을 사용하여 사전을 구현할 수 있으며, 따라서 하드웨어 구현시 필요한 회로 소자의 양을 상당히 줄일 수 있다. 마이크로-테이블은 가중화(weight)되어 있는 데, 이는 새로운 엘리먼트를 이 테이블내에 첨가할 때, 가장 낮은 우선순위를 갖는 엘리먼트가 이 마이크로-테이블에서 삭제된다는 것이며, 여기서 우선순위는 사용 빈도 및 최근에 사용되었는 지의 여부에 근거하여 부여된 것이다.

<18> 전체 인코더내에 있어서, 비트-런 인코더는 단순히 하나의 압축 메카니즘 이상의 중요성을 갖는다. 비트-런 인코더는 데이터의 성질을 변형시킨다. 자신의 비트-런 길이로서 비트맵을 인코딩함으로써, 비트-런 단계는, 대개의 경우 순(raw) 비트맵보다 압축하기가 더 쉬운 데이터 스트림을 생성한다. 이미지 성분들이 픽셀 레벨에서 분석되어, 비트맵내에서의 바이트 경계 위치들이 더 이상 중요하지 않게 된다.

<19> 솔리드 컬러 하프톤(solid color halftone)들을 포함하는, 다수의 간단한 패턴들은 반복된 런 카운트들의 페어로서 인코드되는 데, 여기서 제 1 카운터는 화이트(white) 런 길이를 표시하고 제 2 카운트는 블랙(black) 런 길이를 표시하며, 혹은 그 반대로 된다. 제 2 인코딩 단계는 반복된 페어 데이터를 압축할 수 있도록 최적화되므로, 디코더 단계들간에는 독특한 시너지즘(synergism)이 존재한다. 이 세 단계들을 개별적으로 구현하면, 어느 것에 의해서도 안정적인 고압축율을 얻을 수 없다.

<20> 이 단계들을 조합함으로써, 각각의 단계들의 단순 합(sum)보다 더 좋은 결과－광범위한 문서들에 대해 효율적인 압축을 얻을 수 있다고 하는 결과－를 만들어 낼 수 있게 된다.

<21> 따라서, 본 발명의 특징 중 하나는 다단계 인코딩－제 1 단계는 비트-런 길이 인코딩임－을 제공하는 것이다.

<22> 본 발명의 다른 특징은 비트-런 압축기, 페어-반복 압축기, 사전-기반 압축기를 포함하는 다단계 인코딩을 제공하는 것이다.

<23> 본 발명의 또 다른 특징은 다단계 인코더－제 1 단계는 매우 작은 사전을 사용하는 사전-기반 압축임－를 제공하는 것이다.

<24> 본 발명의 또 다른 특징은 다층(multiple tier)들을 갖는 마이크로-테이블－여기서, 여러번 반복된 엘리먼트는 이보다 덜 반복된 엘리먼트보다 높은 우선순위를 가짐－을 제공하는 것이다.

<25> 본 발명의 또 다른 특징은 다층들을 갖는 마이크로-테이블을 갱신(update) 및 가중화하는 간단한 방법을 제공하는 것이다.

<26> 본 기술 분야의 당업자라면, 첨부한 도면-동일 참조 부호는 동일 부분을 지칭함-과 함께 후술하는 발명의 상세한 설명을 참조함으로써, 본 발명의 상기한 사항은 물론이고 그 이외의 잇점 및 특징들도 명확히 이해할 수 있을 것이다.

도면의 간단한 설명

<27> 도 1은 본 발명의 이미지 데이터 압축기의 전반적인 블럭도이다,

<28> 도 2는 본 발명의 일 실시예에 따른 인코딩 프로세스의 예이다,

<29> 도 3은 본 발명에 따른 단일-층 마이크로-테이블의 물리적 구성을 도시한다,

<30> 도 3a는 도 3의 마이크로-테이블의 논리적 구성을 도시한다,

<31> 도 4는 마이크로-테이블 히트(hit)의 물리적 효과를 도시한다,

<32> 도 4a는 마이크로-테이블 히트 이전의 도 4의 마이크로-테이블의 논리적 구성을 도시한다,

<33> 도 4b는 마이크로-테이블 히트 이후의 도 4의 마이크로-테이블의 논리적 구성을 도시한다,

<34> 도 5는 도 4의 마이크로-테이블상에서의 마이크로-테이블 미스(miss)의 물리적 효과를 도시한다,

<35> 도 5a는 도 4의 마이크로-테이블상에서의 마이크로-테이블 미스의 개념적 효과를 도시한다,

<36> 도 6은 3개의 층(tier)를 갖는 마이크로-테이블을 도시한다,

<37> 도 7은 간단한 교체(swap) 마이크로-테이블 갱신 기법에 따라, 다층화된(multi-tiered) 마이크로-테이블상에서의 마이크로-테이블 히트(hit)의 물리적 효과를 도시한다,

<38> 도 8은 도 7에서 도시한 마이크로-테이블 히트의 개념적인 효과를 도시한다,

<39> 도 9는 무브-투-탑(move-to-top) 마이크로-테이블 갱신 기법에 따라, 다층화된 마이크로-테이블상에서의 마이크로-테이블 히트의 물리적 효과를 도시한다,

<40> 도 10은 도 9에서 도시한 마이크로-테이블 히트의 개념적인 효과를 도시한다,

<41> 도 11은 마이크로-테이블 내의 모든 위치들을 동시에 탐색하는 하드웨어 비교기를 가지고, 하드웨어로 구현한 마이크로-테이블을 도시한다,

<42> 도 12는 본 발명에 따른 데이터 압축해제기를 도시한다,

<43> 도 13은 비트-런 압축 단계의 소프트웨어 구현을 기술한다,

<44> 도 14는 페어-반복 압축 단계의 소프트웨어 구현을 기술한다,

<45> 도 15는 마이크로-테이블 압축 단계의 소프트웨어 구현을 기술한다,

<46> 도 16는 마이크로-테이블 압축해제 단계의 소프트웨어 구현을 기술한다,

<47> 도 17은 페어-반복 압축해제 단계의 소프트웨어 구현을 기술한다,

<48> 도 18은 비트-런 압축해제 단계의 소프트웨어 구현을 기술한다.

발명의 상세한 설명

<49> 다음의 설명은 설명을 위해 사용된 특정 실시예에 대한 것이다. 본 기술 분야의 당업자라면, 본 발명의 범주를 벗어나지 않고도, 다음의 설명에서 사용된 파라미터들 중, 코드 워드 길이, 사전 어드레스 범위, 데이터 심도(data depth)와 같은, 다수의 파라미터들이 변경될 수 있음을 알 것이다.

<50> Ⅰ. 압축

<51> 도 1은 본 발명에 따른 데이터 압축기를 개략적으로 도시한다. 스캐닝 장치(12)(본 발명의 범주에 속하지 않음)가 이미지(10)를 스캐닝하여, 이미지를 순 이미지 데이터의 스트림으로 변환시킨다. 스캐닝 장치는 개인용 컴퓨터들과 함께 사용되는 핸드 헬드(hand-held) 스캐너일 수 있으며, 또는 팩시밀리 머신 또는 사진복사기 등에 내장될 수 있다.

<52> 순 이미지 데이터는 비트-런 압축기(16)에서 구현되는 비트-런 길이 압축 알고리즘을 이용하여 압축된 다음, 페어-반복 압축기(20)들내에서 구현되는 페어-반복 알고리즘을 이용하여 더 압축되고, 선택사양적으로, 마이크로-테이블 압축기(24)내에서 사전-기반 마이크로-테이블 압축 알고리즘을 이용하여 더욱 압축된다. 압축기가 하드웨어로 구현되는 경우에는, 선택사양적인 선입선출형(FIFO) 메모리들(18, 22)을 일련의 압축 단계들 사이에서 데이터들을 전달하는 데 사용할 수 있다.

<53> FIFO들은 일련의 압축 단계들 사이의 대기 시간을 제거함으로써 처리를 증가시킨다. 압축기를 소프트웨어로 구현하는 경우에는, FIFO_top 및 FIFO_bottom 포인터들이 FIFO 동작에 대응되게 작동한다. 코드 팩커(code packer)(26)는 페어-반복 데이터나, 마이크로-테이블에 의해 더 압축된 페어-반복 데이터 중 하나를, 덴서(denser)가 의존하는 출력으로 스위칭한다.

<54> 압축된 출력 데이터는 자성 하드 디스크(28)와 같은 영구 메모리에 저장되거나, RAM과 같은 임시 메모리에 저장되거나, 혹은 원격의 위치로 전송된다. 데이터 압축의 장점은, 이와 대응하여, 메모리 요구의 감소나 전송 시간의 단축으로서 나타난다.

<55> 도 2는 바람직한 실시예에서 이미지 데이터가 압축 프로세스내의 각 단계에서 어떻게 압축되는지를 도시한다. 스캐닝된 이미지를 나타내는 데이터는 이진 데이터 스트림(30)-"0"은 화이트 비트를 나타내고 "1"은 블랙 비트를 나타내며, 혹은 그 반대로 됨-을 포함한다. 스캐닝된 이미지가 텍스트나 선도를 포함하는 인쇄면일 때에는, 데이터 스트림(30)이 다수의 연속된 1들의 런들 혹은 연속된 0들의 런들을 포함할 것이다.

<56> a. 비트-런 압축

<57> 비트-런 압축기(32)는 각각의 런의 길이를 카운트하여, 런의 길이값들로 된 시리즈인 데이터 스트림(34)을 생성한다. 각각의 런의 극성은 이전의 런의 극성과 반대가 된다. 바람직한 실시예에서, 런 값들은 니블(nibble : 4 비트)들로서 인코딩된다.

<58> 그러나, 본 기술 분야의 당업자라면, 런-길이 값들을 바이트(8 비트들), 워드들(16 비트들), 롱 워드들(32 비트들), 또는 압축될 특정 데이터에 대해 최선의 압축율을 얻기 위하여 요망되는 임의의 길이로서, 달리, 표현할 수 있음을 알 것이다. 런 길이값들이 길수록 고 해상도(예컨대, 인치당 1200 도트) 데이터의 처리와 같은 특정한 구현에 더욱 적합할 수 있다. 반대로, 런 길이값들이 짧다는 것은 짧은 런들만을 포함하는 보다 효율적인 인코드 데이터임을 의미한다.

<59> 바람직한 실시예에서는, 4-비트 값들이 사용된다. 1 내지 15의 니블값들은 제각기 런 길이가 1 내지 15 임을 표시한다. 니블값 0은, 런 길이가 15를 초과하는 경우, 그 런의 일부로서 길이가 15이며 다음 니블에서 이 런이 계속되고 있음을 나타낸다. 따라서, 16 비트 또는 그 보다 긴 런들은 이 런의 매 15 비트마다 니블값 0을 출력하며, 이 런의 최종 니블은 나머지 런 길이를 표시하는 1과 15 사이의 값을 갖게 된다. 표 1은 비트-런 인코딩의 예들을 기술한다.

[표 1]

비트-런 길이	니블 인코딩
1	1
15	F
16	01
30	0F
31	001

<61> 표 2는 순 픽셀 데이터의 스트림이 비트-런 길이 압축기에 의해 어떻게 인코딩되는지를 기술한다.

[표 2]

비트-런 길이 압축												
입력 데이터 스트림	11111111011111111000111111110001111111100011110000011110000											
니블을 사용한 압축된 스트링	1000	0001	0111	0011	0111	0011	0111	0011	0100	0101	0100	0101
10진수 표현	8	1	7	3	7	3	7	3	4	5	4	5

<63> 도 2에서, 입력 데이터 스트림(30)은 비트-런 길이 압축 스트림(34)으로 도시한 바와 같이 인코딩된다.

<64> 도 13은 비트-런 인코더의 소프트웨어 구현을 기술한다. 블럭(200)에서 제 1 비트를 읽고, 비트-런 카운트를 1로 세트(set)한다. 블럭(202)에서, 데이터 비트가 전송 또는 기록의 마지막 비트가 아니면, 블럭(204)에서 다음 비트를 읽어들인다. 블럭(206)에서 판단되는 바에 따라, 비트가 동일하면, 블럭(208)에서 런 카운트를 증가시킨다. 런 카운트가 최대값보다 작고(본 바람직한 실시예에서는 15가 최대값임), 블럭(202)에서 아직 데이터의 끝에 도달하지 않았다면, 블럭(204)에서 다음 비트를 읽어들인다.

<65> 반면에, 카운트가 허용가능한 최대 카운트를 초과하면, 카운트를 1로 리셋하고, "0"을 출력하여,

런 카운트가 15를 초과하였음을 표시한다. 블럭(206)에서 비교된 비트들이 서로 상이하면, 런이 끝난 것이므로, 현재의 카운트를 출력하고, 카운트를 1로 리셋한 다음, 비트 페치 및 카운트 프로세스를 다시 시작한다. 블럭(202)에서 데이터 전송 또는 기록의 마지막이 나타나면, 블럭(216)에서 그 때까지의 런 카운트 업을 출력하고, 비트-런 인코더를 종료한다.

<66> 니블값들을 사용하는 바람직한 실시예에서, 4-비트 코드값으로 압축될 수 있는 최대 런은 15 비트이다. 따라서, 이 제 1 단계에서 얻을 수 있는 최대 압축율은 15:4, 즉 3.75:1이다. 런 카운트들을 저장하는 데 더 큰 코드들을 사용하더라도, 안정된 양질의 압축율을 기대할 수는 없을 것이다. 비트-런 압축을 개선시키기 위해 제 2 압축 단계를 첨가한다.

<67> b. 페어-반복 압축

<68> 제 2 압축 단계는 페어-반복(페어-반복) 압축기(36)이다. 이것은 비트-런 코드들의 반복되는 페어들을 압축한다. 여러가지 특수한 인코딩 체계들이 페어-반복 단계에 사용될 수 있다. 바람직한 실시예에서, 페어-반복 압축기(36)는 반복되고 있는 패턴들을 식별하여, 이들을 반복 카운트 규칙자(repeat count specifier)와 이 반복값들의 복사로 인코딩한다. 페어-반복의 일부로서 인코딩되지 않은 데이터는 변경되지 않은 채로 남는다.

[표 3]

pairs-rep 압축	
비트-런 데이터	8 1 7 3 7 3 7 3 4 5 4 5
pairs-rep 압축된 데이터	1 8 1 3 7 3 2 4 5

<70> 페어-반복 압축기(36)의 출력(38)은 바이트로 구성된다. 각각의 바이트는 코드나 데이터 중 하나이다. 제 1 바이트는 코드이다. 각각의 코드는 얼마나 많은 데이터 바이트들이 뒤따라오는지를 표시한다. 코드할 데이터가 없으면, 다른 코드값이 출력된다. 코드의 상위 비트는 자신의 타입을 표시하며, 하위 7 비트들은 카운트를 표시한다. 타입 비트가 클리어이면, 카운트는 후속하는 문자(literal)(페어되지 않은) 데이터 바이트들의 수를 표시한다.

<71> 예를 들면, 도 2의 데이터 스트림(38)에서, 처음의 2-바이트 워드는 다음의 정보를 포함한다. 즉, 타입 필드(44)의 0 비트는 후속하는 데이터가 문자임을 표시하고, 카운트 필드(46)의 값 1은 이 문자 데이터가 1 바이트 길이(2 니블)임을 표시하며, 데이터 필드(48)의 값 81(16진수(hex))은 런-길이 데이터(8 비트가 한 극성을 갖고, 후속하는 1 비트가 다른 극성을 가짐)를 표시한다.

<72> 스캐닝된 텍스트에서, 흑색의 수평 스캔 라인들과 여백들은, 화이트 비트들이 연속되어 있는 긴 스트링으로 나타난다. 이것들은, 테이블 1에 기술한 바와 같이, 비트-런 인코더에 의해 0인 니블값들의 시리즈로서 인코딩될 것이다. 이 데이터들은 페어-반복 인코더에 의해 블럭들의 시리즈－여기서, 각각의 블럭은 "7F 0 0" 값(16진수)들을 포함함－로서 더 인코딩될 것이다. 7-비트 값 "7F"는 127번의 반복을 표시하고, "0 0"은 15 개의 화이트 비트들 다음에 또 15 개의 화이트 비트들이 더 있음을 표시한다. 따라서, 16 비트인 각각의 블럭은 127×30=3810개의 연속된 화이트 화소들을 표시하며, 따라서 3810:116(238:1)의 최대 압축율을 얻게 된다.

<73> 도 14는 페어-반복 인코더의 소프트웨어 구현을 기술한다. 블럭(220)에서, 이전의 (비트-런) 단계로부터 제 1 바이트를 읽어들인다. 이 바이트가 전송 또는 기록의 마지막이 아니라면, 블럭(224)에서 페어 카운트를 0으로 세트한다. 블럭(226, 228, 230)에서, 이 데이터 스트림내의 반복되고 있지 않는 바이트들의 수를 카운트하며, 블록(228)에서 연속한 두 바이트들이 동일한 것으로 검출될 때에만 이 루프로부터 제어가 빠져나온다.

<74> 일단 제어가 루프로부터 빠져나오면, 블럭(232)에서 반복되고 있지 않는 바이트 카운트(문자 카운트)를 출력하고, 블럭(234)에서 판단되는 바에 따라, 문자 스트림이 끝날 때까지 블럭(236)에서 문자들을 계속해서 시프트 아웃(shift out)한다. 일단, 블럭(228)에서 문자들이, 검출된 반복 바이트들에 응답하여, 시프트 아웃되면, 블럭(238)에서 반복 카운트가 2로 세트되고, 바이트가 반복된 횟수는 블럭(240, 242, 244)으로 이루어진 루프에서 카운트된다.

<75> 블럭(242)에서 바이트(니블 페어)의 반복이 중단되었음을 검출하면, 곧, 블럭(246)에서 반복 카운트와 대응하는 니블 페어를 출력하고, 블럭(222)에서 전송 또는 기록의 마지막이 나타났음을 검출할 때까지, 이 프로세스를 전부 다시 재개한다.

<76> 처음 두 단계가 스캐닝된 데이터내의 중복성을 다수 제거하였더라도, 많은 경우, 어느 정도는 여전히 중복부분이 남아 있을 것이다.

<77> c. 마이크로-테이블 압축

<78> 제 3 압축 단계에서, 마이크로-테이블 사전 인코더(50)(도 2 참조)가 비트-런 단계 및 페어-반복 단계가 완료된 후에도 남아있는 중복을 찾는 데 사용된다. 최종 단계로 전달된 데이터는 별로 압축할 것이 없을 것이다. 이 제 3 단계에서의 성공 열쇠는 압축되지 않은 부분들을 확장하는 일 없이 압축될

수 있는 부분들을 압축하는 것이다.

<79> 최종 압축 단계는 이미 압축된 데이터에 대한 작용이므로, 작용될 데이터량은 보다 적고, 따라서 이 최종 단계들에 대한 요건들이 줄어들 것이다. 제 3 단계는 소량의 데이터에 대한 오퍼레이션이므로, 더 느린 속도가 요구된다. 또한, 제 3 단계의 압축율에는 이전의 두 단계들의 압축율이 곱해질 것이므로, 작은 압축률로도 전체적으로는 양호한 압축율을 얻을 수 있다.

<80> 예컨대, 이전의 두 단계들에서 5:1 압축을 얻었다면, 제 3 단계에서 2:1로만 압축하더라도 전체 시스템 압축은 10:1이 될 것이다. 이로써, 테이블이 압축되지 않은 순 데이터에 대해 작용되어야 할 때보다 훨씬 저렴한 비용으로 이 테이블을 하드웨어로 구현하는 것이 가능하게 된다.

<81> 마이크로-테이블(50)은 데이터 스트림(38)에 나타난 데이터 패턴들을 저장함으로써 작업한다. 데이터 스트림(38)과 이전에 마이크로-테이블(50)내에 저장되어 있던 스트링간에 패턴 매치가 발생하면, 마이크로-테이블(50)은 자신의 최상위 비트(most significant bit;MSB) 포지션에 "1"이 있고, 나머지 비트들은 그 패턴이 이전에 저장되어 있던 마이크로-테이블(50)에 대한 인덱스를 표시한다.

<82> 도 2에 도시한 예에서, 마이크로-테이블(50)은 각각 16 비트로된 2^5=32 개의 위치들을 포함한다. 코드(56)는 6 개의 비트－MSB 포지션내의 1-비트 플래그와 5 개의 나머지 인덱스 비트들－로 이루어져 있다. 데이터 스트림과 테이블간에 패턴 매치(pattern match)가 없으면, MSB 포지션은 문자를 표시하는 "0"이고, 나머지 8 개의 비트들은 문자 데이터와 동일한 9-비트 값(58)이 출력된다. 도 2에서, 패턴 3, 7, 3은 이미 마이크로-테이블내의 위치 1D(16진수)에 저장되어 있으며, 이 패턴이 다시 나타나면, 마이크로-테이블 압축기가 출력 코드(56)내에서 인덱스값이 1D인 그 패턴을 이동시킨다.

<83> 전술한 바와 같이, 스캐닝된 이미지 라인이 모두 화이트인 경우에는, 페어-반복 인코더로부터의 한 개의 16-비트 데이터 블럭은 3810 개 모두 화이트 비트임을 표시할 것이다. 이 16-비트 데이터 블럭은 마이크로-테이블에 의해 6-비트 코드 워드로 더 압축됨으로써, 블랭크 라인들에 대해서는 3810:6(635:1)의 최대의 전체 압축율을 얻게 될 것이다.

<84> 이 테이블은 가중된 MRU(Most Recently Used) 테이블로서 간주된다. 가장 최근에 발생한 패턴들을 저장하고 가장 예전에 발생한 패턴들을 제거한다. 이 메카니즘은, 패턴의 발생 빈도를 고려함으로써 가중화될 수 있다. 한 번 이상 발생한 패턴들은 단 한번 발생한 패턴들보다 이 테이블내에서 더 오래 유지될 것이다. 이 가중화된 MRU 메카니즘에 의해 인코더가 극소수의 테이블 엔트리들을 가지고 작업할 수 있게 된다. 마이크로-테이블은, 새로운 엔트리를 위한 여유 공간을 만들기 위해서 이 테이블로부터 엔트리를 제거할 때, 자주 사용한 엔트리들이 자주 사용되지 않는 엔트리들보다 높은 우선순위를 갖도록 하는 방식으로 유지보수된다.

<85> 1. 단일 자격층(qualification layer)

<86> 마이크로-테이블(50)은 여러가지 방식으로 구성될 수 있다. 한 개의 자격층만을 갖는 간단한 제 1 마이크로-테이블 실시예를 도 3 내지 도 5b에 도시한다. 도 3은 마이크로-테이블의 물리적(실제) 구성을 도시하며, 도 3a는 논리적(개념적) 구성을 도시한다. 도 3을 참조하면, 마이크로-테이블(60)은 8 개의 위치를 갖는 스택으로 구성된다. 소프트웨어 혹은 하드웨어로 구현되는 스택 포인터 bottom_ptr은 이 스택에 대한 인덱스를 포함한다.

<87> bottom_ptr(62)이 가리키는 위치는 이 스택의 논리상 최하부로서 정의되며, 이 스택의 물리적인 최하부는 이 스택의 물리적인 최상부와 연결되어 있다. bottom_ptr(62)을 증가시키면, 논리적으로는 스택이 아래 방향으로 시프트되는 효과가 나타나, 최하부 값이 최상부로 가게 된다.

<88> 도 4에서, 테이블 탐색 결과, 매치(테이블 히트)가 발생하면, 히트 엘리먼트(68)가, 이 테이블의 논리적인 최상부로 이동되어 가장 최근에 사용되었음을 표시한다. 이것은, 물리적으로는, 히트 엘리먼트(68)를 bottom_ptr(62)가 가리키는 엘리먼트와 교체(swap)한 후, bottom_ptr(62)을 증가시킴으로써 수행된다. 도 4a를 참조하면, bottom_ptr(62)를 증가시킨 데 따른 논리적 효과는, 이전에 이 포인터가 가리키고 있던 위치에 있던 데이터－이 경우에는 히트 엘리먼트(68)(교체 후)－가 새로운 논리적인 최상부 엘리먼트로 된다는 것이다. bottom_ptr(62)은 또한 이 테이블내의 다른 모든 엘리먼트들의 우선순위를 낮추는 효과도 갖는다.

<89> 순수한 LRU(least recently used)와 상이한 본 메카니즘의 특징 중 하나는, 교체 작용의 결과, 이전에 이 테이블의 논리상 최하부에 있었던 엘리먼트가, 히트 엘리먼트가 논리적인 최상부로 이동함으로 인해 비게 된 슬롯(slot)을 채우기 위하여 이동된다는 것이다. 이로써, 테이블 히트가 있을 때마다, 테이블내에서 히트 엘리먼트 아래에 있던 모든 엘리먼트들이 순환(cycle)되는 결과를 낳는다.

<90> 테이블의 최하부에 있는 엘리먼트가, 임의의 시점에, 최근 사용 빈도가 가장 최소인 것일 필요는 없다. 그러나, 최하부 엘리먼트가 인위적으로 상당히 상향된 우선순위를 갖게 될 수 있는 한편, 다른 엘리먼트들은 테이블 액세스당 한 엘리먼트씩 우선순위가 감소하기만 할 것이다. 엘리먼트가 자주 히트되기만 하면, 테이블의 최하부는 결코 도달되지 않을 것이다.

<91> 이와 달리, 본 메카니즘은, 마이크로-테이블의 최상부 바로 근처의 히트를 특별히 처리함으로써 개선될 수 있다. 예컨대, 최상부 엘리먼트에 히트가 발생하면, 어떠한 이동도 필요없고, 제 2 엘리먼트에 히트가 발생하면 처음의 두 엘리먼트들만 서로 교체하여 제 2 엘리먼트를 최상부로 이동시키면 된다. 이것을 특별 경우로 처리함으로써, 최하부 엘리먼트가 최상부 바로 근처로 이동된 경우, 마이크로-테이블의 기능성을 개선시킬 수 있다.

<92> 테이블 미스

<93> 테이블 미스(miss)의 물리적 및 논리적 효과를 제각기 도 5 및 도 5a에 도시한다. 테이블 탐색 결과, 매치가 발생하지 않으면(테이블 미스), 새로운 (탐색) 엘리먼트(72)가 이 테이블의 논리상 최상부에 삽입되는데, 이를 위한 공간은 이 테이블의 최하부에 있던 엘리먼트(74)를 제거함으로써 확보한다. 이것은, 물리적으로는, bottom_ptr(62)이 가리키고 있던 엘리먼트(74)를 새로운 엘리먼트(72)로 바꾼 후, bottom_ptr(62)이 이보다 한 단계 높은 우선순위를 갖는 엘리먼트(76)를 가리키도록 조정함으로써 달성될 수 있다. 테이블 히트의 경우와 마찬가지로, bottom_ptr(62)이 가리키는 포지션(position)을 증가시킴으로써, 최하부 엘리먼트(74)와 교체된 히트 엘리먼트(72)가 새로운 상부로 되고 다른 모든 엘리먼트들의 우선순위는 낮아진다.

<94> 미스의 결과로 마이크로-테이블에 새로운 엘리먼트를 첨가할 때, 채용되는 MRU 메카니즘은 LRU 메카니즘과 동일하게 기능한다. LRU 메카니즘을 이용하여 마이크로-테이블을 구현하여도 매우 유사한 결과를 얻을 수 있지만, 특히 인코더가 소프트웨어로 구현되는 경우에는, 기술한 MRU 메카니즘을 사용하는 것이 훨씬 더 효과적이다. MRU 메카니즘에 있어서의 최근 사용된 패턴을 보존하는 특성은, 타고난 압축 사전으로서 이것을 더욱 유용하게 만든다. 그러나, 후술하는 바와 같이, MRU 메카니즘을 확장함으로써 극소량의 테이블 공간을 훨씬 더 많이 활용할 수 있게 된다.

<95> 2. 복수의 자격층들

<96> 복수의 자격층들을 갖는 제 2 마이크로-테이블 실시예를 도 6에 도시한다. 본 실시예는 극소량의 테이블 공간을 훨씬 더 많이 활용할 수 있게 하는 MRU 메카니즘의 확장을 기술하고 있다. 자격층은 단순히 별개의 자격 상태(가중치)가 주어진 테이블의 섹션(층)이다. 마이크로-테이블(80)은, 한 층의 최상부에 다른 층이 놓인 식으로 적층된 다층ㅡ여기서, 최하부 층에는 가장 낮은 자격 상태가 주어지고, 각각의 다음 층은 그 아래에 놓인 층보다 높은 자격 상태를 가짐ㅡ들로 구성되어 있다.

<97> 바람직한 실시예에서는, 마이크로-테이블(80)이 세 개의 자격층들로 나뉘어 있다. 최상위층(86)과 중간층(84)은 각각 8 개의 위치들을 포함하고, 최하위 층(82)은 16 개의 위치들을 포함하여, 총 32 개의 위치들이 있다. 마이크로-테이블(80)의 각 층은 별개의 MRU 메카니즘으로 유지보수된다. 층내의 엘리먼트는, 그 엘리먼트의 "히트"가 발생한 때, 한 단계 높은 층으로 상향이동된다. 다른 엘리먼트들이 그 레벨로 상향이동된 결과, 그 레벨내의 한 엘리먼트를 범프 오프(bump off)함으로써 한 단계 낮은 층으로 하향이동시킨다.

<98> 하기 내용을 설명하기 위하여, 마이크로-테이블(80)을 탐색하는 데 대한 참조(reference)들을 집합 테이블(collective table)이라고 지칭한다. 바꾸어 말하면, 마이크로-테이블(80)의 각 층내의 각 엘리먼트는, 목표 패턴이 발견되거나 모든 층들(82, 83, 86)내의 모든 엘리먼트들이 체크될 때까지, 그 목표 패턴에 대해 탐색된다.

<99> 테이블 탐색의 결과, 매치가 발견되면, 앞서 기술한 MRU 첨가 프로세스를 이용하여, 히트 엘리먼트를 현재 포지션에서 제거하고 한 단계 높은 층의 최상부에 삽입한다. 한 단계 높은 층의 최하부에서 제거된 엘리먼트는, 다음의 두 방법 중 하나를 이용하여, 그 히트 엘리먼트의 원래 층내에 놓이게 된다.

<100> 테이블 히트

<101> 도 7 및 도 8은 제 1 (간단한 교체) 방법을 도시한다. 도 7을 참조하여 물리적인 측면에서 보면, 히트가 발생한 경우, 하위 층(92)내의 매치된 엘리먼트(90)를 상위 층(94)내의 최하부 엘리먼트와 교체하고, 상위 층의 bottom_ptr(98)을 증가시킨다.

<102> 도 8을 참조하여 개념적인 측면에서 보면, 히트가 발생한 경우, 하위 층(92)내의 히트 엘리먼트(90)를 상위 층(94)의 최상부로 이동시키고, 상위 층(94)내의 모든 나머지 엘리먼트들을 한 포지션씩 범프 오프시킨다. 상위 층을 범프 오프시킨 후, 상위 층(94)의 최하부 엘리먼트였던 것을 히트 엘리먼트(9)에 의해 비게 된 하위 층내의 포지션으로 이동시킨다. 단일 교체 방법의 잇점 중 하나는 구현하는 것이 쉽고 간단하다는 것이다.

<103> 도 9 및 도 10은 제 2 (무브-투-탑) 방법을 도시한다. 도 9를 참조하여 물리적인 측면에서 보면, 히트가 발생한 경우, 히트 엘리먼트(90)를 상위 층(94)의 최하부 스폿(spot)으로 이동시키고, 상위 층(94)의 최하부로부터 제거된 엘리먼트(102)를 하위(히트) 층(92)의 최상부에 삽입하며, 이 히트 엘리먼트(90)에 의해 비게 된 스폿을 동일한 층(92)의 논리상 최하부 엘리먼트로 바꾼 후, 이 두 층들에 대한 bottom_ptr들(98, 100)을 모두 증가시킨다.

<104> 도 10을 참조하여 개념적인 측면에서 보면, 이 무브-투-탑 방법은 다음의 단계들을 포함한다. 히트 엘리먼트(90)를 한 단계 높은 층(94)의 최상부로 이동시키고, 그 층내의 모든 엘리먼트들을 범프 오프시킨 다음, 층(94)의 최하부에 있던 엘리먼트(96)를, 그 최하부를 범프 오프하여, 층(92)의 최상부 포지션으로 이동시킨다. 개념상, 이것은 앞서 기술한 간단한 교체 방법을 수행한 후 원래 층의 MRU 갱신을 수행함으로써 히트 포지션으로부터의 값(지금은 상위 층의 최하부로부터의 엘리먼트)을 원래 층의 논리적인 최상부로 이동시키는 것으로 생각될 수 있다.

<105> 마이크로-테이블의 최상위 층에서 매치가 발견되면, 그 히트 엘리먼트를 그 층의 최상부로 이동시키도록 MRU 갱신을 수행할 수 있다. 이와 달리, 히트 엘리먼트는 이미 최상위 층에 있고 그 층내에서의 정확한 포지션은 별로 중요하지 않으므로, 인코더가 아무런 행동을 취하지 않고, 간단히 마이크로-테이블을 변경하지 않은 채로 내버려 두어도 무방하다.

<106> 테이블 미스

<107> 마이크로-테이블 탐색 결과, 매치가 발견되지 않으면(즉, 목표 패턴이 테이블내의 어느 층에도 없는 경우), MRU 첨가 오퍼레이션을 사용하여, 목표 패턴을 최하부 층에 간단히 첨가할 수 있다.

<108> 도 15는 두 개의 층들을 갖는 마이크로-테이블 인코더의 소프트웨어 구현을 기술한다. 블럭(250)에서 페어-반복 인코더로부터 처음 16-비트 심볼을 읽어들인다. 이 심볼이 전송 및 기록의 마지막이 아니면, 블럭(254)에서 마이크로-테이블 탐색을 수행하여 이 심볼이 이전에 마이크로-테이블 사전내에 저장되었 있었는지 여부를 판정한다. 심볼이 마이크로-테이블내에서 발견되면, 블럭(258)에서 그 심볼을 가리키는 테이블 인덱스를 출력한다.

<109> 테이블 히트가 하위 층에서 발생한 경우에는 블럭(262)에서 다층 테이블 히트 갱신 작용을 수행하며, 그렇지 않은 경우에는 테이블을 변경하지 않는다. 블럭(256)에서 심볼이 테이블에서 발견되지 않으면, 블럭(266)에서 심볼(문자)을 직접 출력하고, 블럭(266)에서 테이블 미스 갱신 작용을 수행한다.

<110> 다층화된 마이크로-테이블 사용의 결과는 다음과 같다. 엘리먼트가 히트되어 한 단계 높은 층으로 이동될 때마다, 마이크로 테이블내에서의 기반이 더욱 확고하게, 즉 고도로 가중화된다. 층들이 복수 개 있으므로, 단순히 사용되지 않았다는 이유로 엘리먼트가 제거되지는 않는다. 마이크로-테이블의 상위 층으로 올라온 엘리먼트는, 몇개의 다른 엘리먼트가 동일한 레벨에 도달하여 그것과 바뀌는 경우에만 제거될 것이다.

<111> 마이크로-테이블 인코더가 인코드불가능한 긴 데이터 시리즈를 만나면, 각각의 입력 패턴에 대해 자신의 최하부 층에 엘리먼트를 첨가한다. 패턴들 중 어느 것도 다시 히트되지 않는 한, 그것들 중 어느 것도 테이블의 상위 층으로 올라가지 않을 것이다. 이 때문에 상위 층들에 도달한 패턴들이 최근에 히트되지 않았더라도 보존될 수 있다. 인코드되지 않은(한번 발생한) 데이터의 버스트(burst)가 마이크로-테이블의 상위 층들내에 있는 패턴들을 치환(displace)할 수 없다는 것은 중요하면서도 유용한 결과이다. 마찬가지로, 단 두 번 발생하고, 이에 의해 제 2 층에 도달한 데이터는 제 3 층내의 패턴들을 치환할 수 없다.

<112> 인코더가 이렇게 작은 테이블을 사용하기 때문에, (입력 데이터에 비해) 이 테이블내에 저장된 패턴들의 질이 양적인 부족을 메울 수 있어야 한다는 것이 중요하다. MRU 메카니즘을 이용하여 양질의 패턴들을 제공하고, 이 데이터를 다층들내에 저장함으로써, 마이크로-테이블내에서 발생 빈도가 낮은 패턴들이 보다 유용한 패턴들을 대체할 가능성을 줄일 수 있다.

<113> 작은 테이블을 사용하는 한 가지 잇점은 구현을 위해 요구되는 하드웨어가 큰 테이블을 사용하는 경우에 비해 저렴할 것이라는 점이다. 테이블 엔트리가 적기 때문에 테이블이 적은 레지스터에 의해 인덱스 될 수 있다고 하는 부차적인 잇점도 있다. 마찬가지로, 압축된 데이터 스트림내의 코드로서 사용되는 인덱스도 적은 수의 비트들을 포함할 것이다. 이것은, 다시말하면, 컴팩트한 인코딩 체계, 즉 양호한 압축을 얻을 수 있음을 의미한다.

<114> 마이크로-테이블을 탐색하는 데에는 여러가지 방법이 가능하다. 마이크로-테이블을 하드웨어로 구현하는 경우, 한 가지 방법은 간단히, 각각의 테이블 위치를 순차적으로 공통의 버스상에 출력하여, 이 값을 탐색 값과 비교하는 것이다. 이것은 느리다. 더 복잡한 기법을 필요로 하기는 하지만, 훨씬 빠른 방법으로서, 테이블 위치마다 하드웨어 비교기를 구현하여, 탐색 값을 테이블내의 모든 위치와 동시에 비교하는 방법이 있다.

<115> 도 11에 이 방법을 도시한다. 탐색 값을 탐색 레지스터(100)로 탑재하여, 마이크로-테이블(108)내의 각 사전 위치에 저장되어 있는 히스토리 데이터와 동시에 비교한다. 예컨대, 탐색 데이터가 최상부 위치(102)에 저장되어 있는 데이터와 매치되면, 하드와이어드(hardwired) 레지스터(104)내의 대응하는 5-비트 하드와이어드 인덱스 값 1F(16진수)가 테이블_인덱스 버스상으로 출력될 것이다. 5-비트 값이, 1인 MSB(106)과 결합되어, 마이크로-테이블 압축기로부터 출력되는 6-비트 코드 값(110)이 될 것이다.

<116> 이와 달리, 마이크로-테이블을 소프트웨어로 구현할 수 있다. 마이크로-테이블을 소프트웨어로 구현하는 경우에는, 해시 테이블(hash table)을 사용하여 테이블 탐색 속도를 증가시킬 수 있다. 해시 테이블은 랜덤 액세스 메모리(RAM)이며, 마이크로-테이블내에 저장될 수 있는 모든 가능한 데이터 값은 이 해시 테이블내에 대응하는 어드레스를 갖는다. 예컨대, 16-비트 심층(deep) 마이크로-테이블은 64K(2^{16}=65,536 개의 위치들) 어드레스들을 갖는 해시 테이블을 요구할 것이다.

<117> 16-비트 데이터 값 8373(16진수)이 마이크로-테이블내의 위치 1D(16진수)에 저장되어 있다면, 해시 테이블내의 어드레스 8373에 값 1D가 저장된다. 다음에 코드 값 8373이 마이크로-테이블내에 저장되어 있는지를 알기 위해, 마이크로-테이블 인코더가 테이블 탐색을 수행할 때에는, 간단히 해시 테이블내의 어드레스 8373을 액세스하면 된다.

<118> 이 어드레스에 있는 해시 데이터는 1D이며, 따라서 인코더는 탐색 데이터(8373)가 마이크로-테이블내의 어드레스 1D에서 발견될 수 있다는 것을 알게 된다. 마이크로-테이블로부터 값을 범프(bump)할 때는 언제나, 그 위치에 있는 해시 데이터가 더이상 유효하지 않음을 표시하기 위하여, 이 범프할 데이터에 대응하는 해시 테이블에서의 어드레스를 기입하고, 적당한 플래그(flag)를 그 위치에 세트한다. 해시 테이블을 사용하기 전에, 모든 위치들을 예약 값으로 초기화하여야 한다. 이 예약 값은, 마이크로-테이블 압축기에 의해 읽혀지면, 그 해시 테이블 어드레스와 동일한 유효한 패턴이 마이크로-테이블내에 저장되어 있지 않다는 것을 표시한다.

<119> 새로운 압축 기록이 생성되거나 전송되고, 해시 테이블이 사용되지 않는 경우는 언제나, 인코딩을 개시하기 전에 모든 위치에 초기 값을 줌으로써 마이크로-테이블을 초기화하여야 한다.

<120> 전술한 바에 따르면, 본 발명의 마이크로 메카니즘은 압축 및 압축해제를 목적으로 사용되어 왔다. 그러나, 본 기술 분야의 당업자라면, 바로 이 메카니즘을 LRU 스택 대신에 다수의 실제 응용에 적용할 수 있음을 알 것이다. 여기에는 디스크 캐시 및 메모리 캐시와 같은 캐시 메카니즘, 다른 유형의

압축기 및 압축해제기에 사용되는 LRU 메카니즘, 가상 메모리 메카니즘 등이 포함되지만, 이에 국한되는 것은 아니다. 스택의 엘리먼트들에 대한 효과는 전형적인 LRU에 있어서와는 상당히 다르다. 그러나, 마이크로-테이블이 간단하기 때문에, 하드웨어로 구현시에는 비용이 저렴하게 되고 소프트웨어로 구현시에는 높은 성능을 얻을 수 있게 된다는 점은, 많은 경우 잇점이 된다.

<121> D. 코드 팩커

<122> 도 1을 다시 참조하면, 스캐닝될 이미지의 특성과 마이크로-테이블(24)내에 충분한 데이터 히스토리가 구축되어 있는지의 여부에 의존하여, 마이크로-테이블 압축기(24)가 페어-반복 압축기(20)로부터의 데이터를 압축시키지 않고 오히려 실제적으로는 확장시킬 수도 있을 것이다. 따라서, 마이크로-테이블(24)을 디스에이블시키는 것이 보다 효과적일 수 있다. 또한, 마이크로-테이블(24)을 소프트웨어로 구현하는 경우에는, 마이크로-테이블 압축을 수행하는 데 관련된 오버헤드(overhead)가 있을 것이다.

<123> 또한, 데이터 전송 시간이 매우 빠른 시스템에서는, 마이크로-테이블(24)이 필요하지 않을 수도 있다. 이러한 이유로, 전형적인 문서들을 압축할 때 마이크로-테이블(24)을 디스에이블시킴으로써 실제적으로 전체 시스템의 처리를 향상시킬 수 있다. 그러나, 마이크로-테이블 단계(24)가 없으면 만족할만한 압축이 이루어지지 않는 페이지(page)들이 있을 수 있으므로, 필요에 따라 선택할 수 있도록 해야 한다.

<124> 코드 팩커(26)는 마이크로-테이블(24)을 인에이블시킬지 여부를 판단한다. 도 2에 도시한 바와 같이 하드웨어로 구현하는 경우, 코드 팩커(26)는 페어-반복 압축기(20)의 출력이나 마이크로-테이블 압축기(24)의 출력 중 하나를 사용할 것이다. 소프트웨어로 구현하는 경우에는, 코드 팩커(26)가 마이크로-테이블(24)을 인에이블 혹은 디스에이블시킬 것이다.

<125> 코드 팩커(26)를 작동시킬 수 있는 알고리즘은 매우 다양하다. 제 1 방법으로, 코드 팩커(26)는, 처음의 두 단계(16, 20)에 의해 생성된 압축율이 사전선택된 임계 레벨과 맞지 않는 경우에만, 마이크로-테이블(24) 출력을 사용할 수 있다. 그 점에서, 페이지의 압축이 완료되었기 때문에, 래스터화(rasterization)가 느려져 압축 시간이 상당히 줄어들게 된다.

<126> 제 2 방법으로, 코드 팩커(26)는, 처음의 두 단계들(20, 16)에 의해 생성된 압축율이 이전 밴드(band)에 대한 것과 상당히 다를 경우(비트맵 데이터의 성질이 변했음을 표시함)에는 언제나 마이크로-테이블(24) 출력을 사용하도록 할 수 있다. 테스트 밴드에 대해 상당한 이득을 얻는다면, 다음 밴드들에도 역시 최종 단계를 적용할 것이다. 코드 팩커(26)를 작동시키는 다른 알고리즘들도 본 기술 분야의당업자에게는 명백할 것이다.

<127> II. 압축해제

<128> 도 12를 참조하면, 본 발명에 따른 압축해제는 기본적으로 압축의 역순이 된다.

<129> 자성 디스크(120)와 같은 데이터 소스로부터, 압축된 비트맵을 나타내는 데이터 스트림을 수신한다. 데이터를 먼저 마이크로-테이블 압축해제기(122), 다음으로 페어-반복 압축해제기(126), 마지막으로 비트-런 압축해제기(130)에 의해 압축해제한 다음, 레이저 프린터(132)와 같은 출력 장치로 전달한다. 도 1의 압축기와 같이, 선택사양적인 FIFO(124, 128)들이 일련의 압축해제기 단계들 사이에서 데이터를 완충하여 대기 시간을 제거함으로써 처리 속도를 증가시킨다. 압축해제의 여러 단계들에 대해 다음에 보다 상세히 기술한다.

<130> A. 마이크로-테이블 압축해제

<131> 마이크로-테이블 압축해제 단계(122)에서, 압축된 데이터 스트림은 압축기로부터 출력될 때와 동일한 순서로 처리되며, 문자 값들로부터 매치 코드들을 구별하도록 데이터를 파싱(parse)한다. 데이터가 처리됨에 따라, 테이블 정보가 재구성되어 압축해제기는 각각의 가능한 입력 코드 값에 대응하는 패턴 값이 무엇인지를 알 수 있게 된다.

<132> 마이크로-테이블 압축해제(122) 동안, 테이블은, 압축시와 동일한 방식으로 조작된다. 기본적인 차이점은, (압축시와 같은 탐색을 요구하는 대신) 각각의 입력값이, 매치가 발생하였는지, 만약 발생했다면, 테이블의 어느 위치에서 발생하였는지를 직접적으로 가리키고 있다는 점이다. 테이블 매치가 있으면, 마이크로-테이블 압축해제(122)는, 마이크로-테이블 압축기(24)(도 1 참조)가 동일한 포지션에서의 매치에 대해 행하는 것과 동일한 방식으로 테이블을 갱신한다.

<133> 마찬가지로, 테이블 미스에 대해서는 마이크로-테이블의 최하위 층을 압축 동안 수행한 것과 유사한 MRU-첨가 작용으로 갱신된다. 마이크로-테이블 압축기(24)에 의해 수행된 테이블 조작을 재실행함으로써, 마이크로-테이블 압축해제기(122)가 등가의 테이블을 만들어 낸다.

<134> 따라서, 마이크로-테이블 압축기(24)가 매치를 발견하여 테이블 포지션을 표시하는 코드를 출력할 경우, 마이크로-테이블 압축해제기(122)는 이 테이블의 복사내에서 그 포지션을 찾아봄으로써 매칭 패턴이 무엇인지를 알아낼 수 있다.

<135> 마이크로-테이블 압축해제에는 한 가지 복잡한 문제가 있다. 마이크로-테이블 압축기(24)가 패턴에 대한 매치를 발견하지 못한 경우에는, 문자값으로서 그 패턴의 위치를 출력할 뿐이다. 나머지는 입력 데이터 스트림으로부터의 부가적인 데이터에 첨부되며, 테이블이 다시 탐색된다.

<136> 문자들은 원래 패턴의 위치를 표시하고 있을 뿐이므로, 마이크로-테이블 압축해제기(122)는 문자가 발견될 때 전체 패턴을 MRU내에 즉시 저장할 수 없다. 대신, 마이크로-테이블 압축해제기는, 이전

리터랄과 함께 마이크로-테이블에 첨가될 패턴의 나머지를 드러낸 다음 코드나 문자(character)를 수신할 때까지 대기한다.

<137> 이와 달리, 테이블 미스가 발생한 경우, 전체 리터랄 패턴이 압축기의 출력 스트림에서 인코드되도록 마이크로-테이블 압축기(24)(도 1)를 구성할 수도 있다. 이 경우, 패턴이 즉시 마이크로-테이블 압축해제기(122)에 이용가능하게 될 것이며, 따라서, 수신되자마자 테이블에 첨가될 수 있을 것이다.

<138> 도 16은 이층의 사전을 갖는 마이크로-테이블의 소프트웨어 구현을 기술한다. 블럭(272)에서 코드의 블럭을 읽어들인다. 블럭(274)에서, 코드가 테이블 히트를 표시하면(즉, 도 2의 코드 워드(56 또는 58)의 MSB가 "1"이면), 제어가 블럭(276)으로 넘어간다.

<139> 종전의 코드가 테이블 미스를 표시했었던 경우에는(즉, 이전 플래그가 -1로 세트되지 않았었다면), 블럭(278)에서, 이 코드를 하위 테이블의 최상부의 최하위 바이트로서 삽입하며, 다른 경우에는, 그 히트 엘리먼트를 출력하고 테이블 히트를 표시하도록 이전 플래그를 세트한다.
블럭(282)에서, 히트가 상위 층에서 발생한 경우에는, 아무것도 행해지지 않는다. 블럭(282)에서 히트가 하위 층에서 발생한 경우에는, 블럭(284)에서 그 히트 엘리먼트를 상위 층의 최상부로 이동시키고, 다음 바이트를 읽어들인다.

<140> 반면에, 코드가 테이블 미스를 표시하면, 제어가 블럭(286)으로 넘어간다. 종전의 코드가 테이블 미스를 표시했었던 경우에는 블럭(288)에서 하위 테이블의 최상부에 코드를 삽입하며, 다른 경우에는 이 코드를 하위 테이블의 최상부 엘리먼트의 최상위 바이트로서 삽입하고 다음 바이트를 읽어들인다.

<141> B. 페어-반복 압축해제

<142> 페어-반복 압축해제기(126)는 반복된 페어를 카운터 표시기에 의해 지정된 횟수만큼 복제한다. 반복되지 않은 데이터는 변경되지 않고 페어-반복 압축해제기로부터 출력된다.

<143> 도 17은 페어-반복 압축해제기(126)의 소프트웨어 구현을 기술한다. 블럭(294)에서 마이크로-테이블 압축해제기로부터 제 1 코드 바이트를 읽는다. 블럭(296)에서 이 코드 바이트가 반복된 데이터를 나타낸다고 표시하는 경우(예컨대, 도 2에서 코드의 MSB(44)가 "1"인 경우), 블럭(298)에서 다음 바이트(반복된 데이터임)를 읽어들여, 블럭(302)에서 적당한 횟수－이 횟수는 제 1 코드 바이트의 7 개의 최하위 비트들(LSB들)에 의해 지정됨－만큼 출력된 다음, 블럭(300)에서 카운트 다운된다.

<144> 코드 바이트가, 후속하는 데이터가 문자 데이터임을 표시하는 경우(예컨대, 도 2 에서 코드의 MSB(44)가 "0"인 경우), 블럭(306)에서 바이트의 수를 읽어 출력하는 한편, 판독 및 출력되고, 블럭(306)에서 문자 데이터 바이트 카운트를 카운트 다운한다.

<145> C. 비트-런 압축해제

<146> 비트-런 압축해제기(130)는 페어-반복 압축해제기(126)로부터의 런 길이 표시기들의 스트림을 1과 0들의 스트림으로 변환시킨다. 각각의 런 카운트마다, 현재의 극성을 갖는 비트들을 그 개수만큼 출력한 다음 극성을 토글(toggle)한다. 카운트가 0이면, 최대 런 길이를 출력하고 극성은 토글하지 않는다.

<147> 도 18은 비트-런 압축해제기(130)의 소프트웨어 구현을 기술한다. 블럭(312)에서 니블인 코드를 읽어들인다. 코드 니블이 0이 아니면, 블럭(318)에서 런 길이는 니블의 값으로 되고, 니블이 0이면, 블럭(316)에서 런 길이는 15보다 크며 다음 니블에서 계속된다.

<148> 블럭(320, 322)에서, "1" 또는 "0"이 런 길이와 동일한 횟수만큼 출력된다. 블럭(324)에서, 코드가 0이 아니면(데이터 런이 다음 니블에서 계속되지 않음), 출력 비트의 극성이 0에서 1로, 혹은 그 반대로 변한다. 이어서, 블럭(310)에서 다음 니블에 대한 프로세스를 다시 시작한다.

<149> 본 발명은 블랙 비트와 화이트 비트를 나타내는 비트 스트림을 구성하는, 단색 이미지 데이터에 대하여 기술된 것이다. 그러나, 본 발명은 단색 이미지 데이터의 압축 및 압축해제에 국한된 것은 아니다.

<150> 비트맵 데이터 스트림은 블랙/화이트 픽셀 데이터를 표시하는 데에만 사용되는 것이 아니라, 저장 또는 전송 효율에 있어서의 증가에 대응하여, 일 성분의 RGB 컬러 데이터나 일 성분의 밝기(luminance)/명암(chrominance) 컬러 데이터와 같은, 일 채널의 다색 픽셀 데이터를 표시할 수도 있다.

<151> 본 발명은 16 비트의 고정 길이 엘리먼트들 갖는 사전에 관하여 기술된 것이다. 그러나, 사전들은 가변 길이 스트링을 인코딩하는 것으로 구성될 수도 잇다. 예컨대, 란가나단(Ranganathan) 등에게 허여된 미국 특허 제 5,179,378 호는 사전 테이블내의 데이터 스트링－여기서 인코드된 데이터 스트링들은 가변 길이임－을 표시하는 데 코드 워드들을 사용하는 방법을 기술하고 있다. 본 기술 분야의 당업자라면, 다수의 이러한 가변 길이 인코딩 기술이 본 발명과 결합되어 사용될 수 있다는 것을 알 것이다.

<152> 본 발명은 바람직한 실시예 및 그에 대한 도면들에 관하여 상세히 기술하고 있지만, 본 기술 분야의 당업자라면, 본 발명의 사상 및 범주를 벗어나지 않고도, 본 발명의 다양한 응용 및 변형이 가능하다는 것을 알 것이다.

<153> 따라서, 앞서 상술된 바와 같은 상세한 설명 및 첨부한 도면은 본 발명을 제한하는 것으로 해석되어서는 안되며, 단지 다음의 청구의 범위 및 그의 등가 범위에 의해서만 본 발명이 한정되는 것임을 알아야 한다.

(57) 청구의 범위

청구항 1

이진 비트(binary bit)들로 이루어진 이진 입력 데이터 스트림(binary input data stream)을 압축하는 방법에 있어서,

(a) 제 1 압축 데이터 스트림을 만들어내기 위하여 상기 이진 입력 데이터 스트림을 비트-런 인코딩(bit-run encoding)하는 단계로서,

(a1) 상기 입력 데이터 스트림내의 다수의 연속적인 동일 비트들로 이루어진 제 1 런(run)을 상기 제 1 런내의 다수의 비트들을 나타내는 제 1 심볼로 바꾸는 단계와,

(a2) 상기 입력 데이터 스트림내의 다수의 연속적인 동일 비트들로 이루어진 제 2 런을 상기 제 2 런내의 다수의 비트들을 나타내는 제 2 심볼로 바꾸는 단계－상기 입력 데이터 스트림내에서 상기 제 2 런은 상기 제 1 런에 곧바로 이어지고, 상기 제 2 런은 상기 제 1 런내의 비트들의 극성과 반대되는 극성을 갖는 비트들을 가짐－

를 포함하는 비트-런 인코딩 단계와,

(b) 제 2 압축 데이터 스트림을 만들어내기 위해 상기 제 1 압축 데이터 스트림을 페어-반복 압축(pairs-rep compressing)하는 단계로서,

(b1) 상기 제 1 압축 데이터 스트림내에서 연속해서 반복되어 있는 한 쌍의 심볼들(페어)을 식별하는 단계와,

(b2) 상기 페어가 연속해서 반복되어 있는 횟수를 판정하는 단계와,

(b3) 상기 반복되는 페어와 상기 페어의 반복 횟수에 대응하는 제 1 값들의 셋트를 생성하는 단계

를 포함하는 페어-반복 압축 단계

를 포함하는 이진 입력 데이터 스트림 압축 방법.

청구항 2

제 1 항에 있어서,

상기 부단계(substep) (a1)에서 만들어진 상기 제 1 심볼은 상기 제 1 런내에 있는 상기 비트들의 예(example)를 포함하지 않고,

상기 부단계 (a2)에서 만들어진 상기 제 2 심볼은 상기 제 2 런내에 있는 상기 비트들의 예를 포함하지 않는

이진 입력 데이터 스트림 압축 방법.

청구항 3

제 1 항에 있어서,

(c) 제 3 압축 데이터 스트림을 만들어내기 위해 사전-기반 압축(dictionary-based compression)을 이용하여 상기 제 2 압축 데이터 스트림을 압축하는 단계를 더 포함하되, 상기 사전-기반 압축을 하는 단계는,

(c1) 상기 사전내의 위치에 사전 엘리먼트를 저장하는 부단계－상기 사전 엘리먼트는 상기 제 1 값들의 셋트에 대응함－와,

(c2) 상기 위치에 대응하는 인덱스를 생성하는 부단계와,

(c3) 상기 제 2 압축 데이터 스트림내의 상기 제 1 값들의 셋트를 상기 인덱스로 바꾸는 부단계를 포함하는

이진 입력 데이터 스트림 압축 방법.

청구항 4

제 3 항에 있어서,

상기 사전내의 상기 사전 엘리먼트들은 최대 32 개인 이진 입력 데이터 스트림 압축 방법.

청구항 5

제 3 항에 있어서,

상기 사전은 다수의 상기 사전 엘리먼트들을 포함하는 스택과, 상기 스택내에 있는 최하위 우선순위 엘리먼트를 가리키는 포인터를 포함하며, 상기 스택은 다음의 단계, 즉

(d1) 상기 최하위 우선순위 엘리먼트를 가장 최근에 사용된 엘리먼트로 바꿈으로써 상기 스택내의 엘리먼트들을 하향이동시키는 단계와,

(d2) 상기 포인터를 증가시키는 단계에 의해 갱신되는

이진 입력 데이터 스트림 압축 방법.

청구항 6

제 3 항에 있어서,

상기 사전은 다층화된 스택이고, 상기 다층화된 스택은 제 1 부스택과 제 2 부스택을 포함하며, 각각의 부스택은 식별된 최하위 우선순위 엘리먼트를 포함하고, 상기 스택은 다음의 단계, 즉

(e1) 제 1 사전 엘리먼트가 상기 제 2 압축 데이터 스트림에서 검출된 것에 응답하여, 상기 제 1 엘리먼트를 상기 제 1 부스택에서 상기 제 2 부스택으로 이동시킴으로써 상향이동시키는 단계와,

(e2) 상기 제 1 엘리먼트가 상기 제 2 부스택내의 최하위 우선순위 엘리먼트일 때, 제 2 엘리먼트가 상기 제 1 부스택에서 상기 제 2 부스택으로 상향이동된 것에 응답하여, 상기 제 1 엘리먼트를 상기 제 2 부스택에서 상기 제 1 부스택으로 이동시킴으로써 하향이동시키는 단계에 따라 갱신되는

이진 입력 데이터 스트림 압축 방법.

청구항 7

제 3 항에 있어서,

상기 사전은 다층화된 스택이고, 상기 다층화된 스택은 제 1, 제 2, 제 3 부스택을 포함하며, 각각의 부스택은 식별된 최하위 우선순위 엘리먼트를 포함하고, 상기 제 1, 제 2, 제 3 부스택들은 각각 최하위부터 최상위까지로서 정의되며, 상기 스택은 다음의 단계, 즉

(e1) 제 1 사전 엘리먼트가 하위 부스택에 위치한 경우, 상기 제 1 사전 엘리먼트가 상기 제 2 압축 데이터 스트림내에서 검출된 것에 응답하여, 상기 제 1 엘리먼트를 상기 하위 부스택에서 상위 부스택으로 이동시킴으로써 상향이동시키는 단계와,

(e2) 상기 제 1 엘리먼트가 상위 부스택내의 최하위 우선순위 엘리먼트인 경우, 제 2 엘리먼트가 상기 하위 부스택에서 상기 상위 부스택으로 상향이동된 것에 응답하여, 상기 제 1 엘리먼트를 상기 상위 부스택에서 상기 하위 부스택으로 이동시킴으로써 하향이동시키는 단계에 따라 갱신되는

이진 입력 데이터 스트림 압축 방법.

청구항 8

제 7 항에 있어서,

상기 제 1 부스택은 8 개의 사전 엘리먼트들을 포함하고,

상기 제 2 부스택은 8 개의 사전 엘리먼트들을 포함하며,

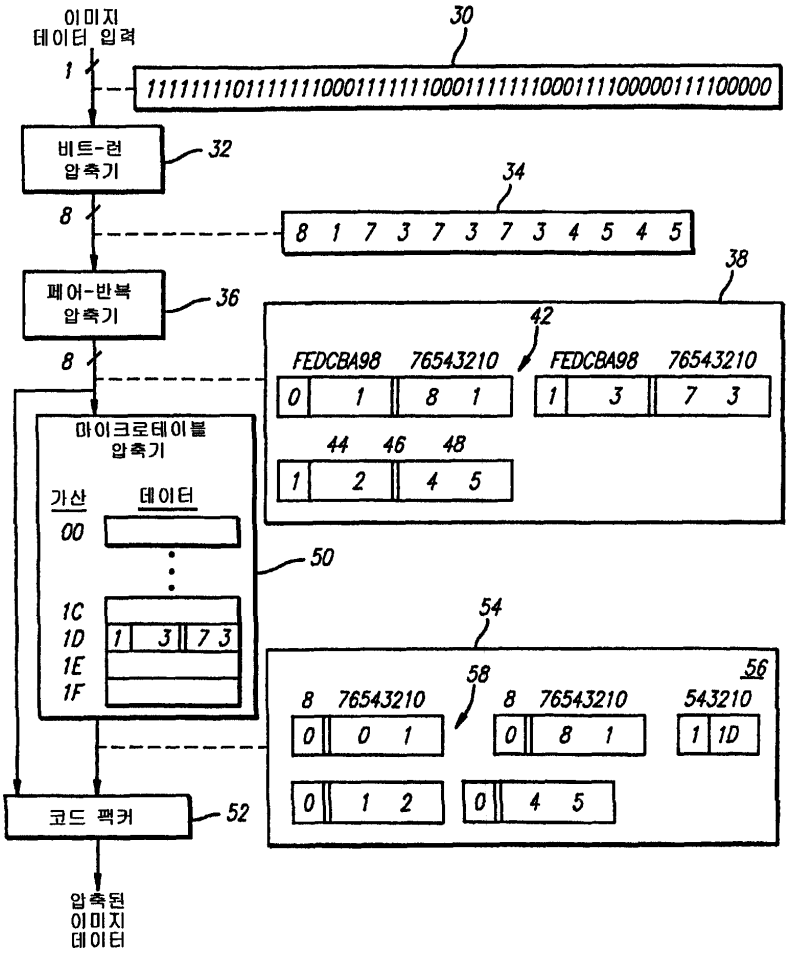
상기 제 3 부스택은 16 개의 사전 엘리먼트들을 포함하는

이진 입력 데이터 스트림 압축 방법.

요약

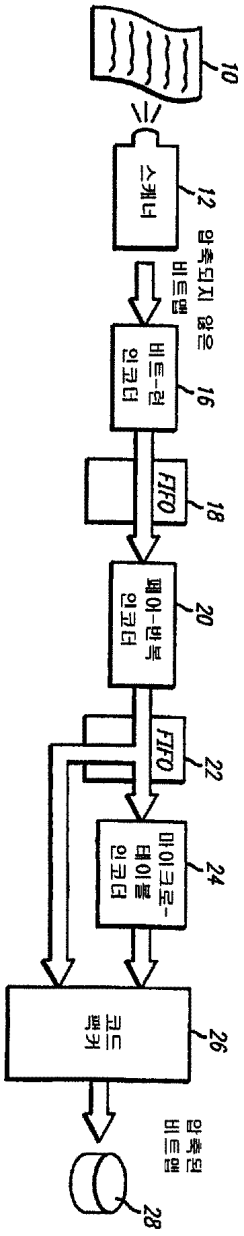
텍스트나 하프톤 이미지와 같은 이미지 데이터를 압축 또는 압축해제한다. 압축기는 연속해 있는 동일한 픽셀들로 이루어진 각각의 런의 길이를 카운트하는 비트-런 길이 단계(32)와, 픽셀 런 값들의 반복된 페어들을 반복된 페어의 복사 및 반복 카운트로 압축하는 페어-반복 단계(36)와, 선택사양적인 사전-기반 마이크로-테이블 인코더(50)의, 세 개의 단계들을 갖는다. LRU 메카니즘을 필요로하는 임의의 애플리케이션에서 사용될 수 있는 마이크로-테이블(80)은 복수의 자격층들(82, 84, 86)－이 자격층내의 엘리먼트들은 그 엘리먼트에 대한 테이블 히트가 발생하면 한 단계 높은 자격층으로 상향이동되고, 하위 자격층으로부터 상향이동된 엘리먼트들로 인해 범프됨으로써 한 단계 낮은 자격층으로 하향이동됨－을 갖는다. 이 결과 테이블은 히트의 빈도 및 최근성 모두에 의해 가중된다.

대표도

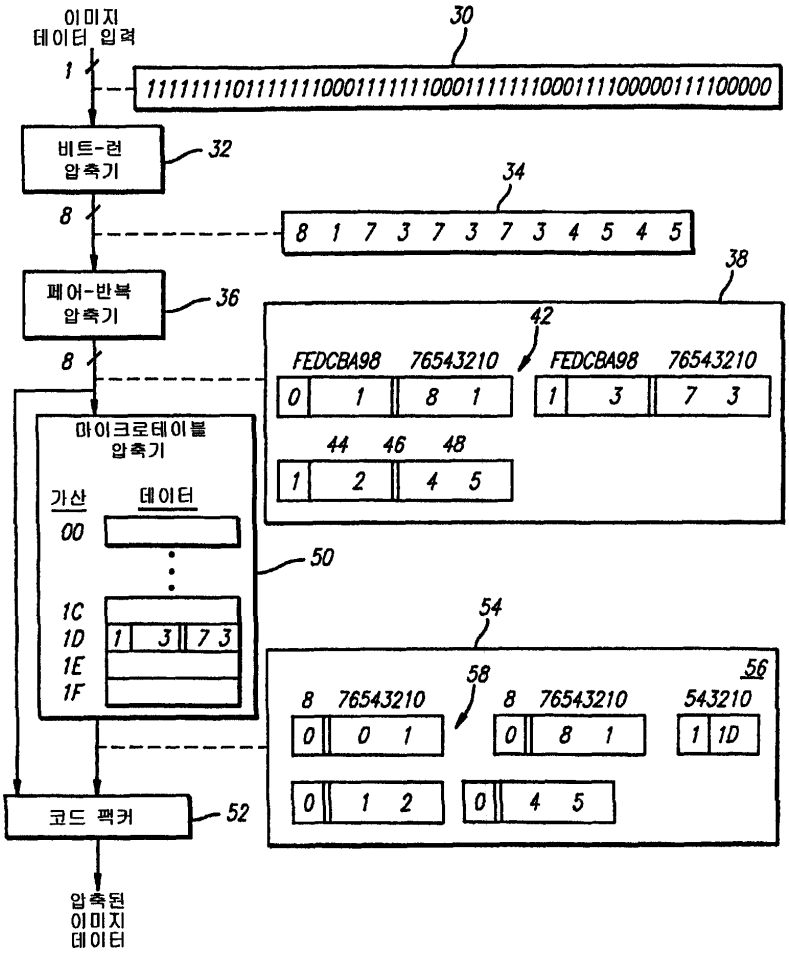

이미지
데이터 입력
1
30
11111111011111110001111111000111111100011110000011110000
비트-런
압축기
32
8
34
8 1 7 3 7 3 7 3 4 5 4 5
폐어-반복
압축기
36
8
38
42
FEDCBA98 76543210 FEDCBA98 76543210
0 1 8 1
1 3 7 3
44 46 48
1 2 4 5
마이크로테이블
압축기
가산
데이터
00
1C
1D
1 3 7 3
1E
1F
50
54
56
58
8 76543210
0 0 1
8 76543210
0 8 1
543210
1 1D
0 1 2
0 4 5
코드 팩커
52
압축된
이미지
데이터

도면

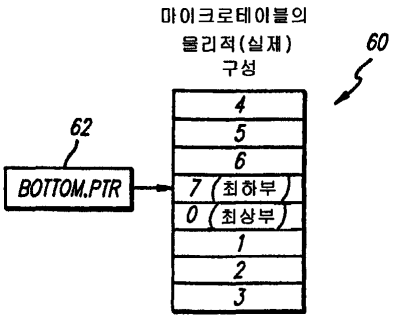
도면1


10
스캐너
12
압축되지 않은 비트맵
비트-런 인코더
16
FIFO
18
페어-반복 인코더
20
FIFO
22
마이크로-테이블 인코더
24
코드 북커
26
압축된 비트맵
28

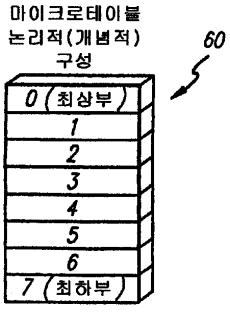
도면2


이미지
데이터 입력
30
1
11111111011111110001111111000111111100011110000011110000
비트-런
압축기
32
34
8
8 1 7 3 7 3 7 3 4 5 4 5
폐어-반복
압축기
36
38
42
FEDCBA98 76543210 FEDCBA98 76543210
0 1 8 1
1 3 7 3
44 46 48
1 2 4 5
마이크로테이블
압축기
50
가산
데이터
00
1C
1D
1E
1F
1 3 7 3
54
56
58
8 76543210
0 0 1
8 76543210
0 8 1
543210
1 1D
0 1 2
0 4 5
코드 팩커
52
압축된
이미지
데이터

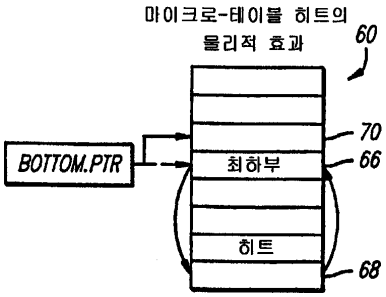
도면3


마이크로테이블의
물리적(실제)
구성
60
62
BOTTOM.PTR
4
5
6
7 (최하부)
0 (최상부)
1
2
3

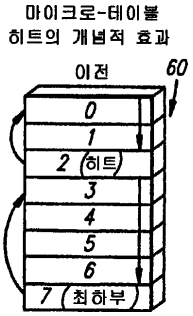
도면3a


마이크로테이블
논리적(개념적)
구성
60
0 (최상부)
1
2
3
4
5
6
7 (최하부)

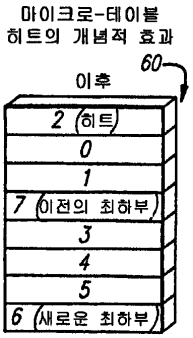
도면4



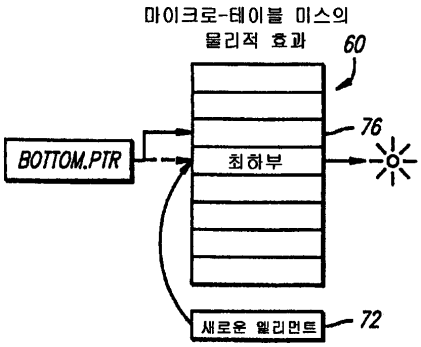
도면4a



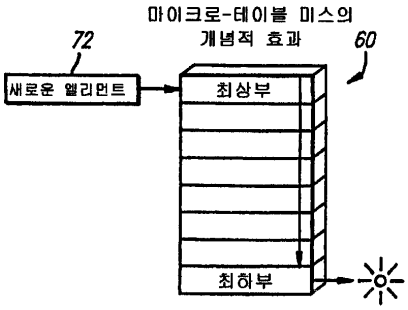
도면4b



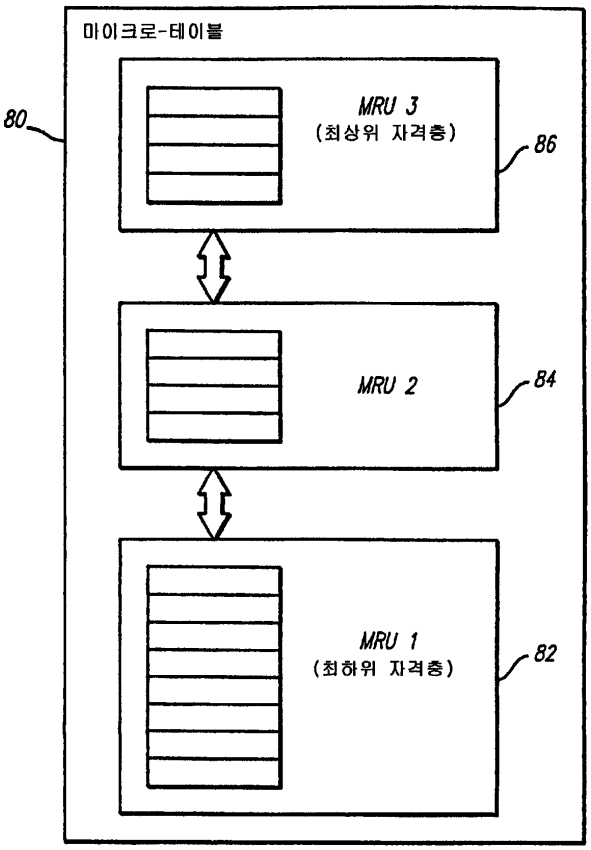
도면5



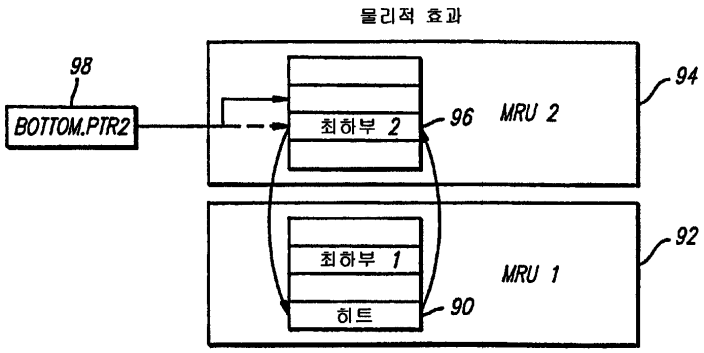
도면5a


마이크로-테이블 미스의
개념적 효과
72
60
새로운 엘리먼트
최상부
최하부

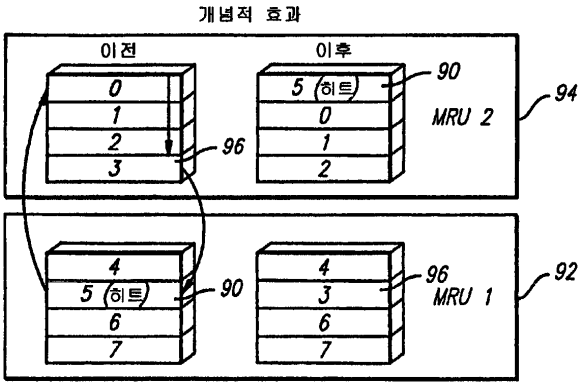
도면6


마이크로-테이블
80
MRU 3
(최상위 자격층)
86
MRU 2
84
MRU 1
(최하위 자격층)
82

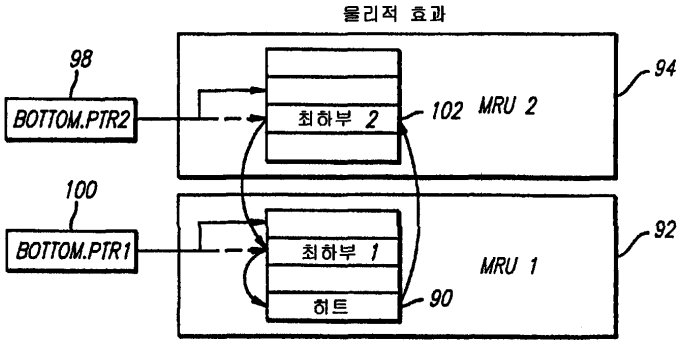
도면7


물리적 효과
98
BOTTOM.PTR2
최하부 2
96
MRU 2
94
최하부 1
히트
90
MRU 1
92

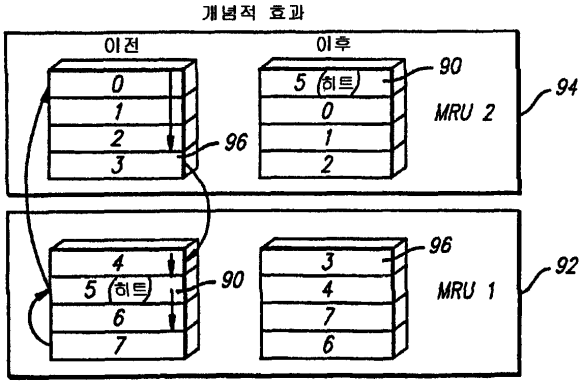
도면8


개념적 효과
이전
이후
0
1
2
3
96
5 (히트)
0
1
2
90
MRU 2
94
4
5 (히트)
6
7
90
4
3
6
7
96
MRU 1
92

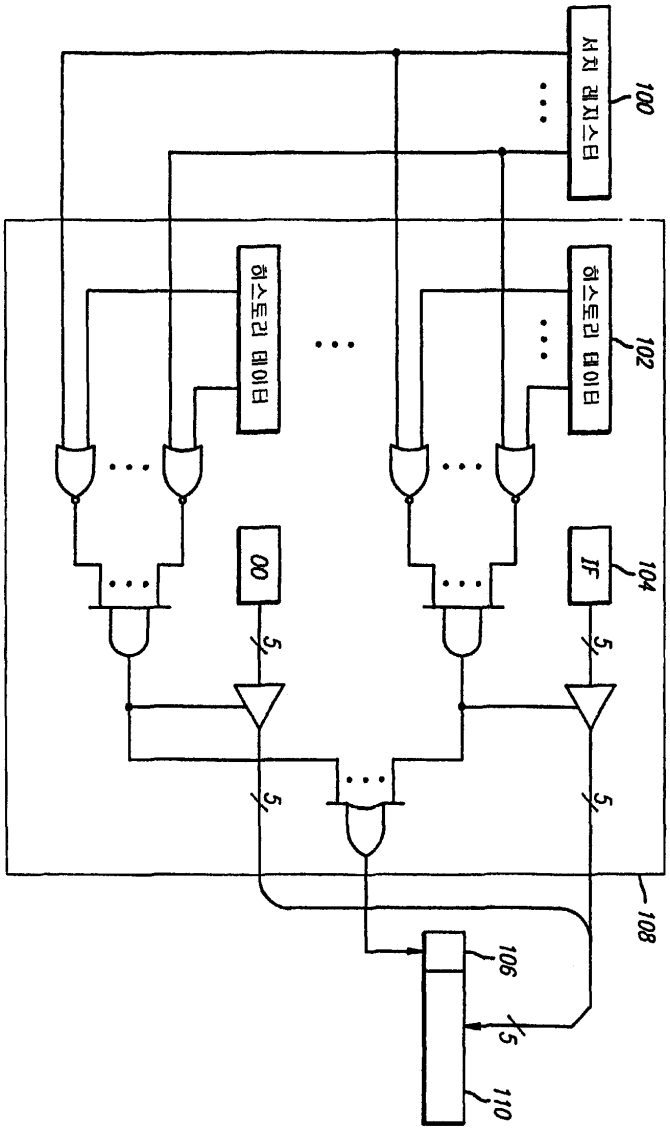
도면9


물리적 효과
98
BOTTOM.PTR2
최하부 2
102
MRU 2
94
100
BOTTOM.PTR1
최하부 1
히트
90
MRU 1
92

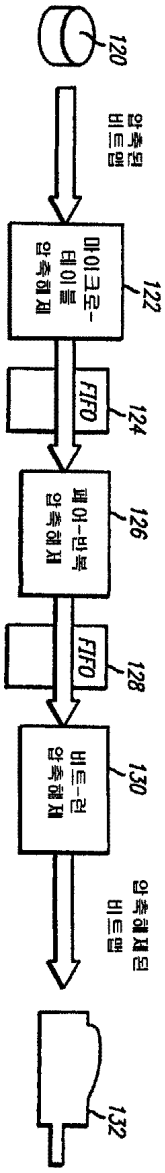
도면10


개념적 효과
이전
이후
0
1
2
3
96
5 (히트)
0
1
2
90
MRU 2
94
4
5 (히트)
6
7
90
3
4
7
6
96
MRU 1
92

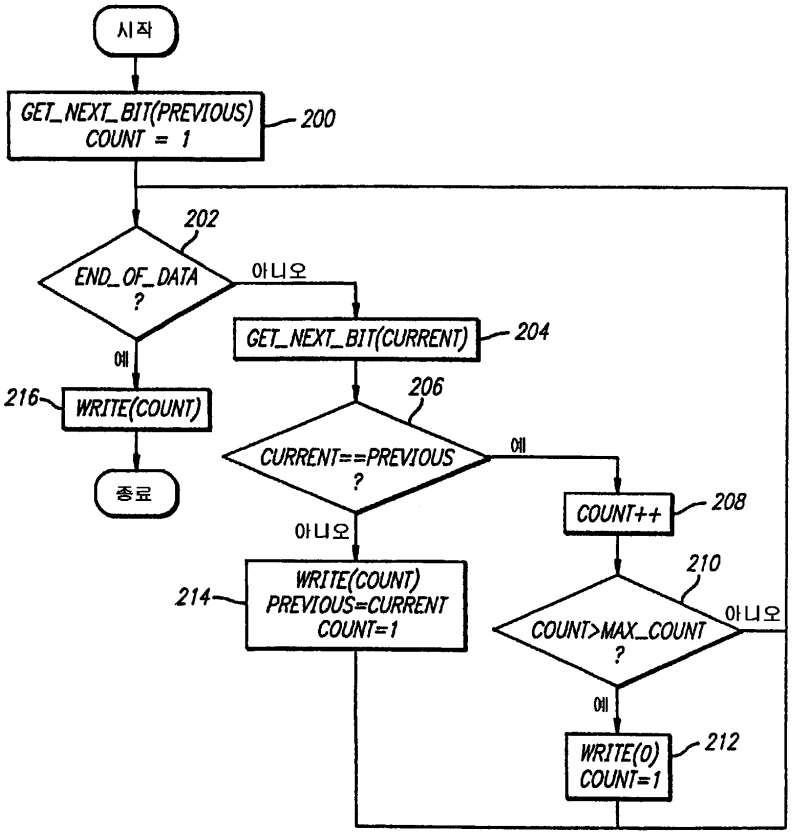
도면11



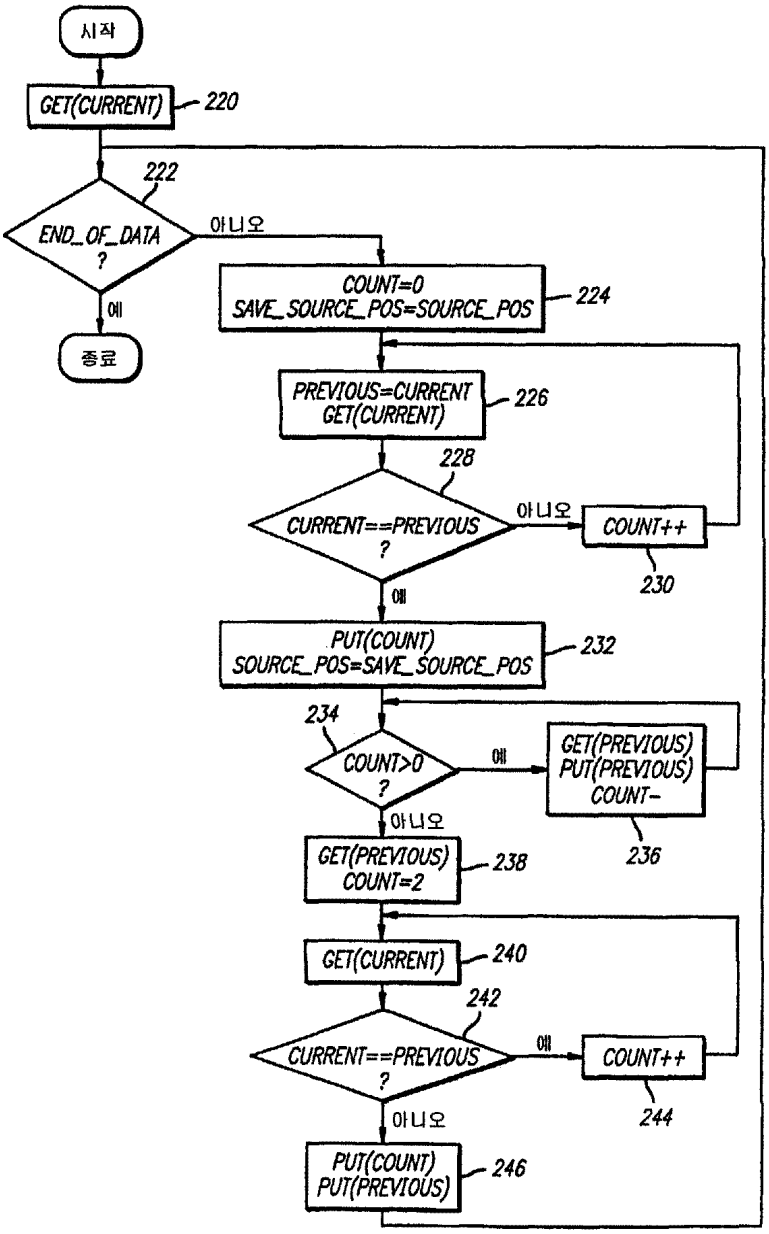
도면12


120
압축된 비트맵
마이크로-테이블 압축해제
122
FIFO
124
페어-반복 압축해제
126
FIFO
128
비트-런 압축해제
130
압축해제된 비트맵
132

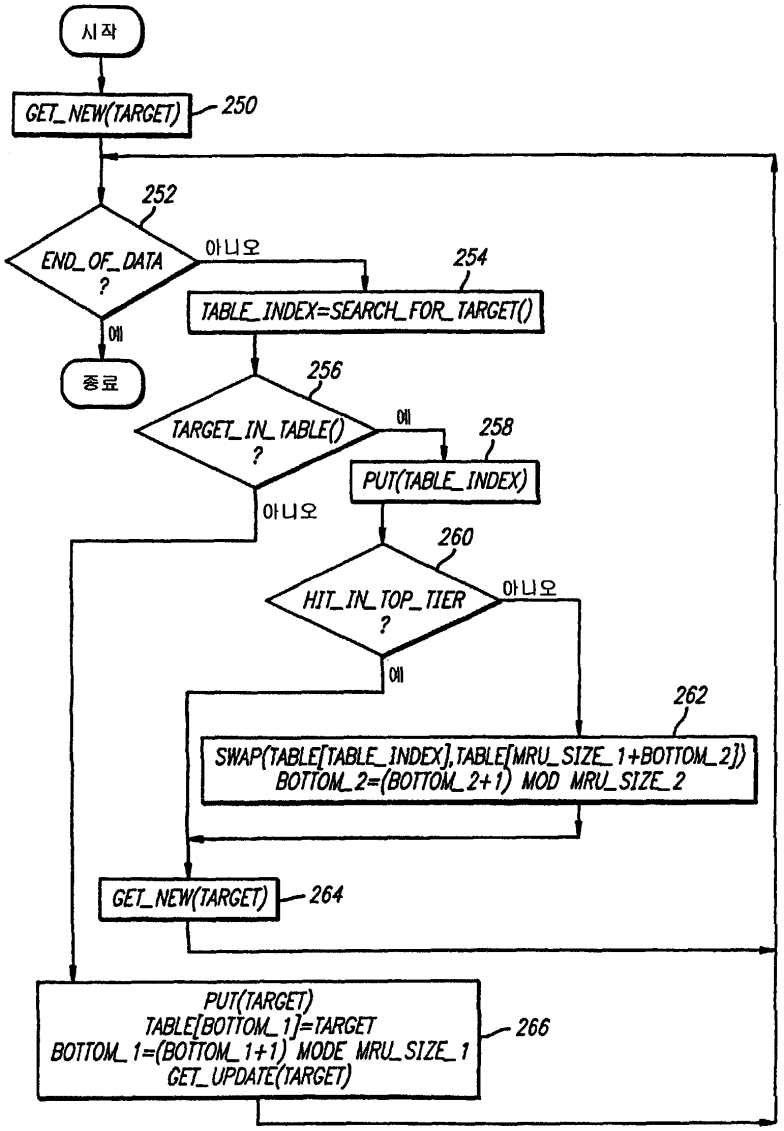
도면13


시작
GET_NEXT_BIT(PREVIOUS)
COUNT = 1
200
202
END_OF_DATA
?
아니오
예
GET_NEXT_BIT(CURRENT)
204
206
CURRENT==PREVIOUS
?
예
아니오
COUNT++
208
210
COUNT>MAX_COUNT
?
아니오
예
WRITE(0)
COUNT=1
212
214
WRITE(COUNT)
PREVIOUS=CURRENT
COUNT=1
216
WRITE(COUNT)
종료

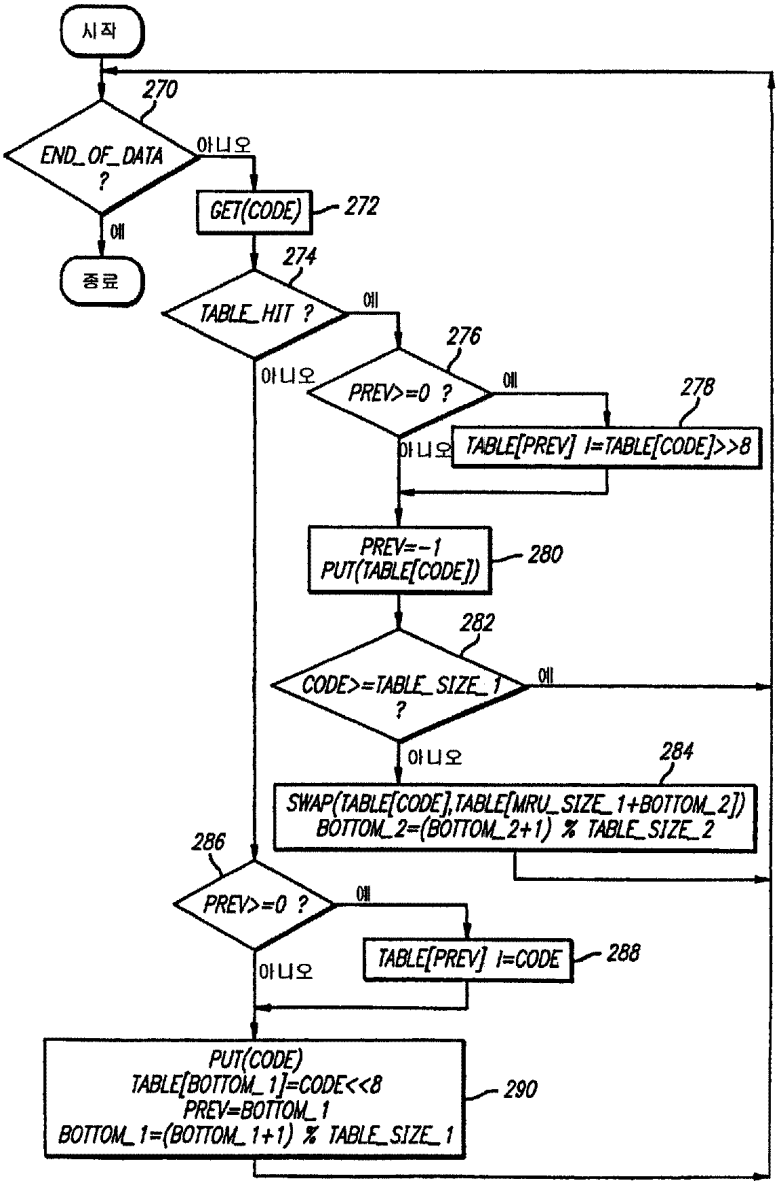
도면14


시작
GET(CURRENT)
220
222
END_OF_DATA ?
아니오
예
종료
COUNT=0
SAVE_SOURCE_POS=SOURCE_POS
224
PREVIOUS=CURRENT
GET(CURRENT)
226
228
CURRENT==PREVIOUS ?
아니오
COUNT++
230
예
PUT(COUNT)
SOURCE_POS=SAVE_SOURCE_POS
232
234
COUNT>0 ?
예
GET(PREVIOUS)
PUT(PREVIOUS)
COUNT-
236
아니오
GET(PREVIOUS)
COUNT=2
238
GET(CURRENT)
240
242
CURRENT==PREVIOUS ?
예
COUNT++
244
아니오
PUT(COUNT)
PUT(PREVIOUS)
246

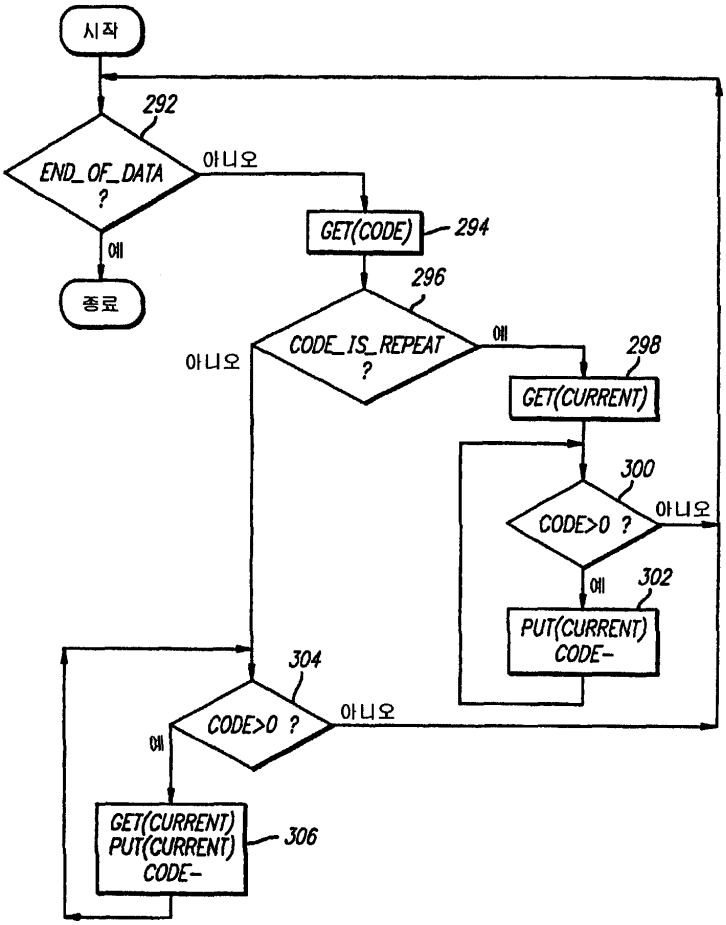
도면15


시작
GET_NEW(TARGET)
250
252
END_OF_DATA ?
아니오
254
TABLE_INDEX=SEARCH_FOR_TARGET()
예
종료
256
TARGET_IN_TABLE() ?
예
258
PUT(TABLE_INDEX)
아니오
260
HIT_IN_TOP_TIER ?
아니오
예
262
SWAP(TABLE[TABLE_INDEX],TABLE[MRU_SIZE_1+BOTTOM_2])
BOTTOM_2=(BOTTOM_2+1) MOD MRU_SIZE_2
GET_NEW(TARGET)
264
PUT(TARGET)
TABLE[BOTTOM_1]=TARGET
BOTTOM_1=(BOTTOM_1+1) MODE MRU_SIZE_1
GET_UPDATE(TARGET)
266

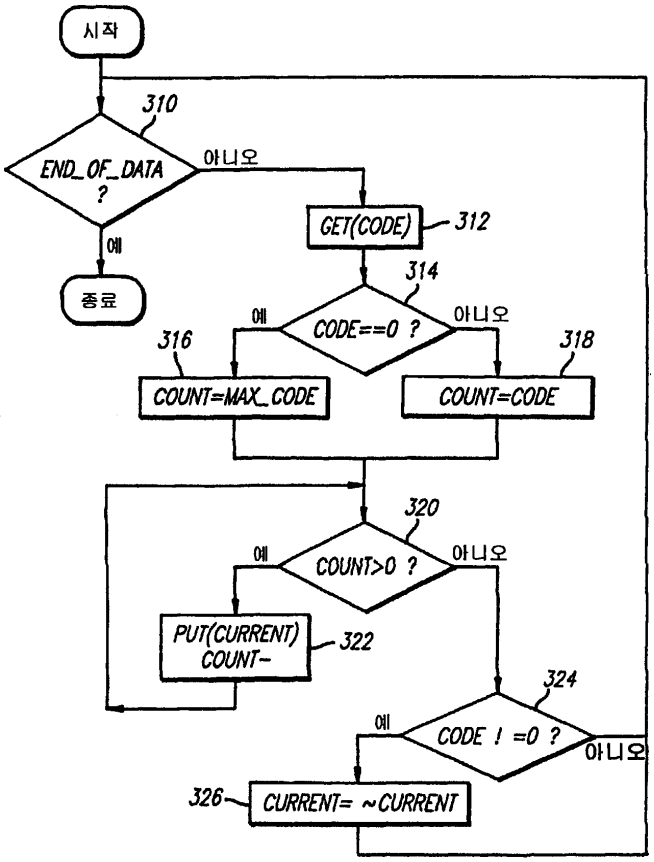
도면16


시작
270
END_OF_DATA ?
아니오
예
GET(CODE)
272
종료
274
TABLE_HIT ?
예
276
아니오
PREV>=0 ?
예
278
아니오
TABLE[PREV] !=TABLE[CODE]>>8
PREV=-1
PUT(TABLE[CODE])
280
282
CODE>=TABLE_SIZE_1 ?
예
아니오
284
SWAP(TABLE[CODE],TABLE[MRU_SIZE_1+BOTTOM_2])
BOTTOM_2=(BOTTOM_2+1) % TABLE_SIZE_2
286
PREV>=0 ?
예
아니오
TABLE[PREV] !=CODE
288
PUT(CODE)
TABLE[BOTTOM_1]=CODE<<8
PREV=BOTTOM_1
BOTTOM_1=(BOTTOM_1+1) % TABLE_SIZE_1
290

도면17


시작
292
END_OF_DATA ?
아니오
예
종료
GET(CODE)
294
296
CODE_IS_REPEAT ?
예
아니오
298
GET(CURRENT)
300
CODE>0 ?
아니오
예
302
PUT(CURRENT)
CODE-
304
CODE>0 ?
아니오
예
GET(CURRENT)
PUT(CURRENT)
CODE-
306

도면18


시작
310
END_OF_DATA ?
아니오
예
종료
GET(CODE)
312
314
CODE==0 ?
예
아니오
316
318
COUNT=MAX_CODE
COUNT=CODE
320
COUNT>0 ?
예
아니오
PUT(CURRENT)
COUNT-
322
324
CODE ! =0 ?
예
아니오
326
CURRENT= ~CURRENT