(12) UK Patent Application (19) GB (11) 2542707 (13) A

(43) Date of A Publication 29.03.2017

(21) Application No: **1618768.4**

(22) Date of Filing: **03.07.2015**

Date Lodged: **07.11.2016**

(62) Divided from Application No
1511740.1 under section 15(9) of the Patents Act 1977

(51) INT CL:
H03M 7/40 (2006.01) ***H04L 9/06*** (2006.01)

(56) Documents Cited:
US 7773634 B1 **US 20020021754 A1**

(58) Field of Search:
INT CL **H03M, H04L**
Other: **WPI, EPODOC, INSPEC**

(71) Applicant(s):
SISP Technologies Ltd
47 Albemarle Street, Mayfair, LONDON, W1S 4JW, United Kingdom

(72) Inventor(s):
Stuart Marlow
Nicholas Stavrinou

(74) Agent and/or Address for Service:
Mathys & Squire LLP
The Shard, 32 London Bridge Street, LONDON, SE1 9SG, United Kingdom

(54) Title of the Invention: **Data processing method and apparatus**
Abstract Title: **Data compression or encryption by dividing input data, performing frequency analysis and assigning frequency-based labels**

(57) A method of processing data, for use in compression or encryption, is disclosed in which an input sequence of bits is divided into a plurality of portions. Each portion is sub-divided into a plurality of sub-divisions. Frequency analysis is performed to determine the number of occurrences of each sub-division permutation and new values are assigned, based on the frequency analysis, to each of the sub-division permutations. For each portion a label representing the permutation of bits in that portion is assigned. The label comprises a representation of a value resulting from adding the new values associated with the sub-division permutations of that portion. A processed sequence of bits is generated by replacing, within the input sequence of bits, bit portions with the respective label representing the permutation of bits in that portion.

Figure 5B

Frequency analysis of bit portion values having a bit length of 4 bits

$L_{BP} = 4$

Ranking	Bit Portion value	Number of occurrences	Occurrence-based Level (total occurences)	Occurrence-based New BP Value	New BP value in binary	Disambiguation value in binary	Initial label 403	Occurrence-based Label size	No. of bits used
Most occurring									
0	0 0 0 1	27369	Level 0 (64539)	0	000	-	000	3 bits	193617
1	1 0 1 1	18629		1	001	-	001		
2	0 0 1 1	18541		2	010	-	010		
3	1 0 0 1	12646	Level 1 (38803)	3	011	0	0110	4 bits	155212
4	1 1 1 1	11079		3	011	1	0111		
5	0 0 0 0	7900		4	100	0	1000		
6	0 1 0 1	7178		4	100	1	1001		
7	1 0 0 0	3923	Level 2 (27730)	5	101	00	10100	5 bits	128840
8	1 1 0 1	3857		5	101	01	10101		
9	0 1 1 1	3658		5	101	10	10110		
10	0 0 1 0	3414		5	101	11	10111		
11	1 1 0 0	3169		6	110	00	11000		
12	0 1 0 0	3074		6	110	01	11001		
13	1 0 1 0	2605		6	110	10	11010		
14	0 1 1 0	2068		6	110	11	11011		
Least occurring									
15	1 1 1 0	1962		7	111	-	111	3 bits	5886
	Total number of bits analysed: **524288**		Total number of 4-bit portions: **131072**						Total number of bits used: **483555**

GB 2542707 A

The claims were filed later than the filing date but within the period prescribed by Rule 22(1) of the Patents Rules 2007.

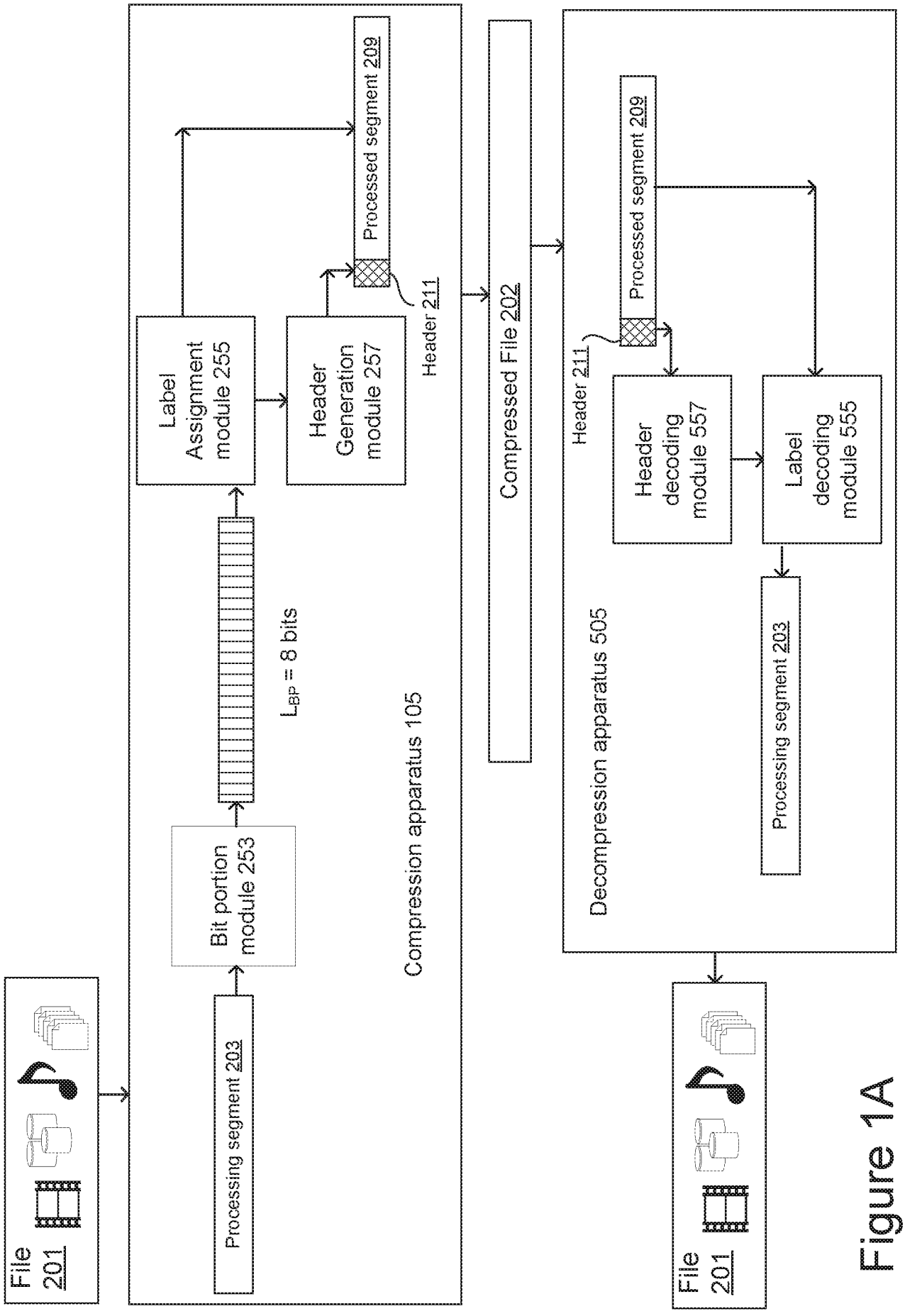
File 201
Processing segment 203
Bit portion module 253
L_{BP} = 8 bits
Label Assignment module 255
Header Generation module 257
Processed segment 209
Header 211
Compression apparatus 105
Compressed File 202
Decompression apparatus 505
Header 211
Processed segment 209
Header decoding module 557
Label decoding module 555
Processing segment 203
File 201

Figure 1A

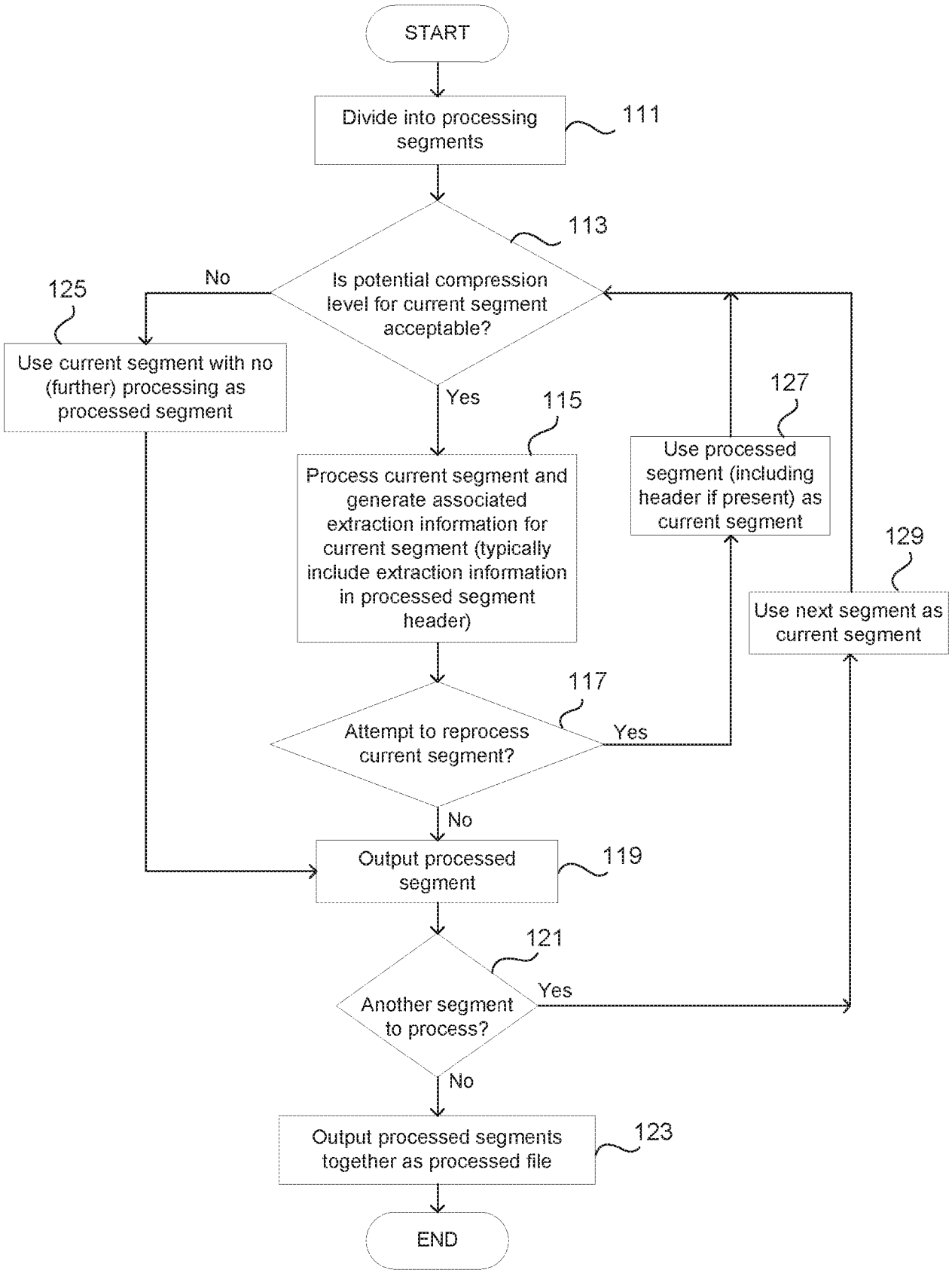
START
Divide into processing segments
111
113
Is potential compression level for current segment acceptable?
No
125
Use current segment with no (further) processing as processed segment
Yes
115
Process current segment and generate associated extraction information for current segment (typically include extraction information in processed segment header)
127
Use processed segment (including header if present) as current segment
129
Use next segment as current segment
117
Attempt to reprocess current segment?
Yes
No
Output processed segment
119
121
Another segment to process?
Yes
No
Output processed segments together as processed file
123
END

Figure 1B

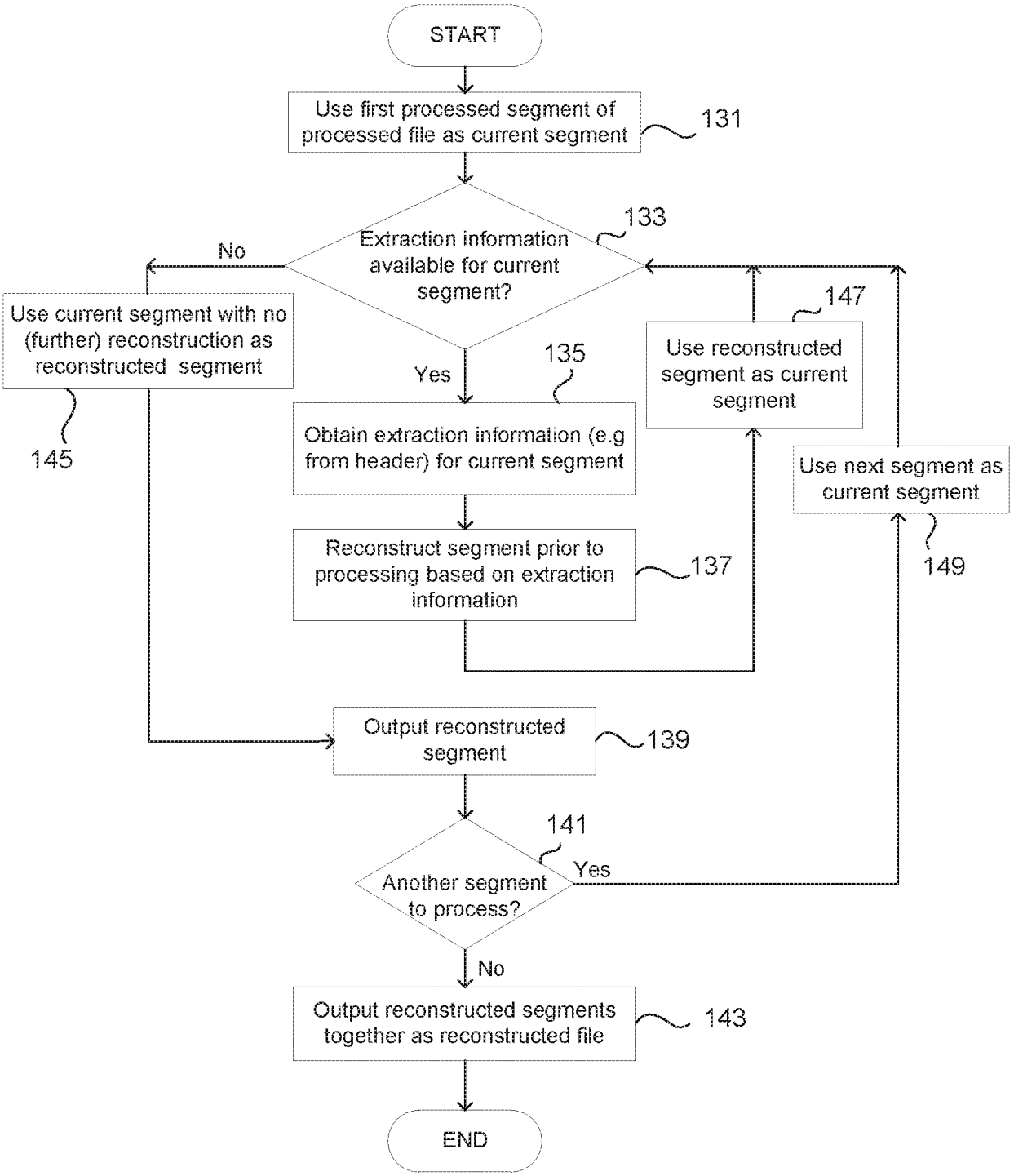
START
Use first processed segment of processed file as current segment
131
Extraction information available for current segment?
133
No
Yes
Use current segment with no (further) reconstruction as reconstructed segment
145
135
Obtain extraction information (e.g from header) for current segment
Reconstruct segment prior to processing based on extraction information
137
147
Use reconstructed segment as current segment
Use next segment as current segment
149
Output reconstructed segment
139
141
Another segment to process?
Yes
No
Output reconstructed segments together as reconstructed file
143
END

Figure 1C

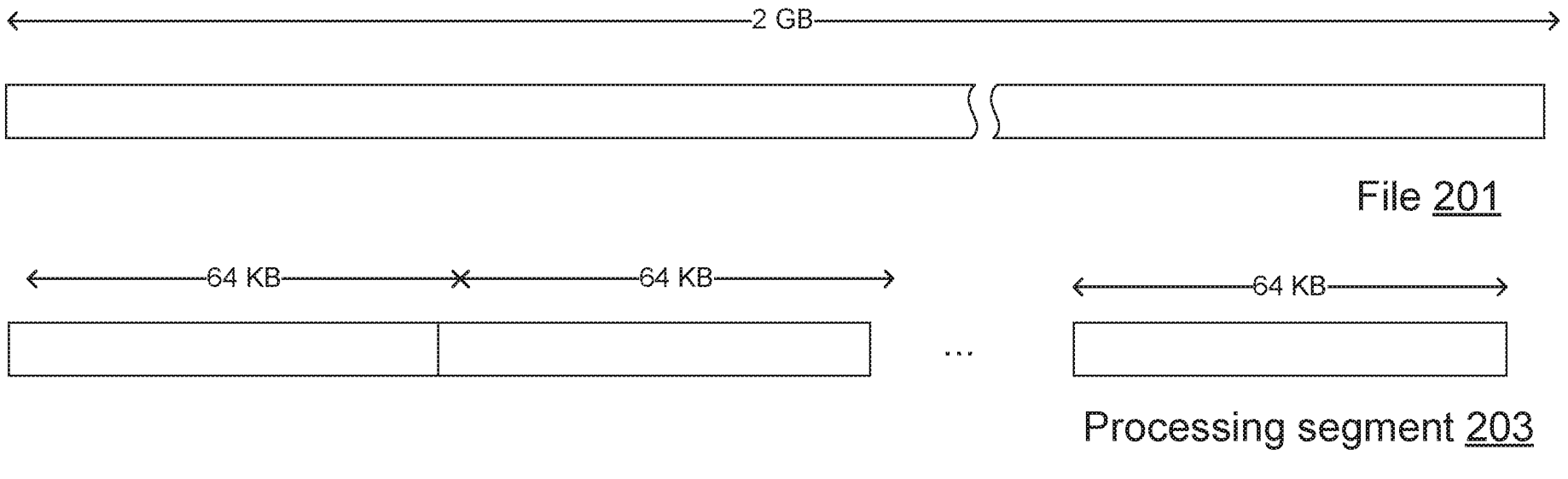
2 GB
File 201
64 KB
64 KB
64 KB
Processing segment 203
Bit length L_{BP}
6 bits
6 bits
6 bits
011100
100110
111100
Bit portion 205
0 1 1100 1 0 0110
1 1 1100
Combination arrays 207

Figure 2

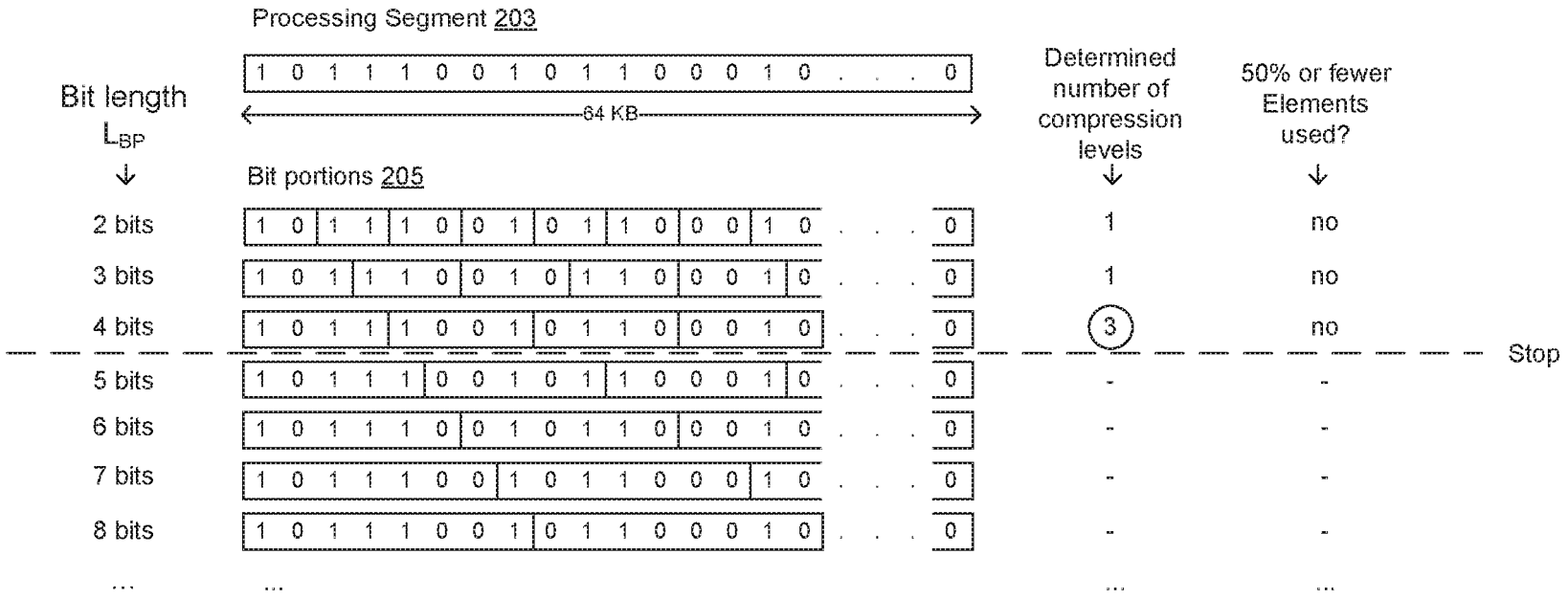
Processing Segment 203
1 0 1 1 1 0 0 1 0 1 1 0 0 0 1 0 . . . 0
64 KB
Bit length
L_{BP}
Bit portions 205
Determined number of compression levels
50% or fewer Elements used?
2 bits
1 0 | 1 1 | 1 0 | 0 1 | 0 1 | 1 0 | 0 0 | 1 0 . . . 0
1
no
3 bits
1 0 1 | 1 1 0 | 0 1 0 | 1 1 0 | 0 0 1 | 0 . . . 0
1
no
4 bits
1 0 1 1 | 1 0 0 1 | 0 1 1 0 | 0 0 1 0 . . . 0
3
no
Stop
5 bits
1 0 1 1 1 | 0 0 1 0 1 | 1 0 0 0 1 | 0 . . . 0
-
-
6 bits
1 0 1 1 1 0 | 0 1 0 1 1 0 | 0 0 1 0 . . . 0
-
-
7 bits
1 0 1 1 1 0 0 | 1 0 1 1 0 0 0 | 1 0 . . . 0
-
-
8 bits
1 0 1 1 1 0 0 1 | 0 1 1 0 0 0 1 0 . . . 0
-
-
... ...

Figure 3A

$L_{BP} = 2$

Frequency analysis of bit portion values having a bit length of 2 bits

	Ranking	Bit portion value	Number of occurences	Level
Most occuring	0	0 1	65,538	1st Level
	1	1 1	65,537	
	2	1 0	65,536	
Least occuring	3	0 0	65,533	
			Total number of 2-bit portions: **262144**	Total number of levels: 1

Figure 3B

More than 50% of elements in use

$L_{BP} = 3$

Frequency analysis of bit portion values having a bit length of 3 bits

	Ranking	Bit portion value	**Number of occurances**	**Occurrence-based Level**
Most occurring	0	0 1 1	21,851	Level 0
	1	0 0 1	21,849	
	2	0 0 0	21,848	
	3	0 1 0	21,847	
	4	1 0 0	21,846	
	5	1 1 1	21,845	
	6	1 1 0	21,844	
Least occurring	7	1 0 1	21,833	
			Total number of 3-bit portions: **174763**	Total number of levels: 1

Figure 3C

More than 50% of elements in use

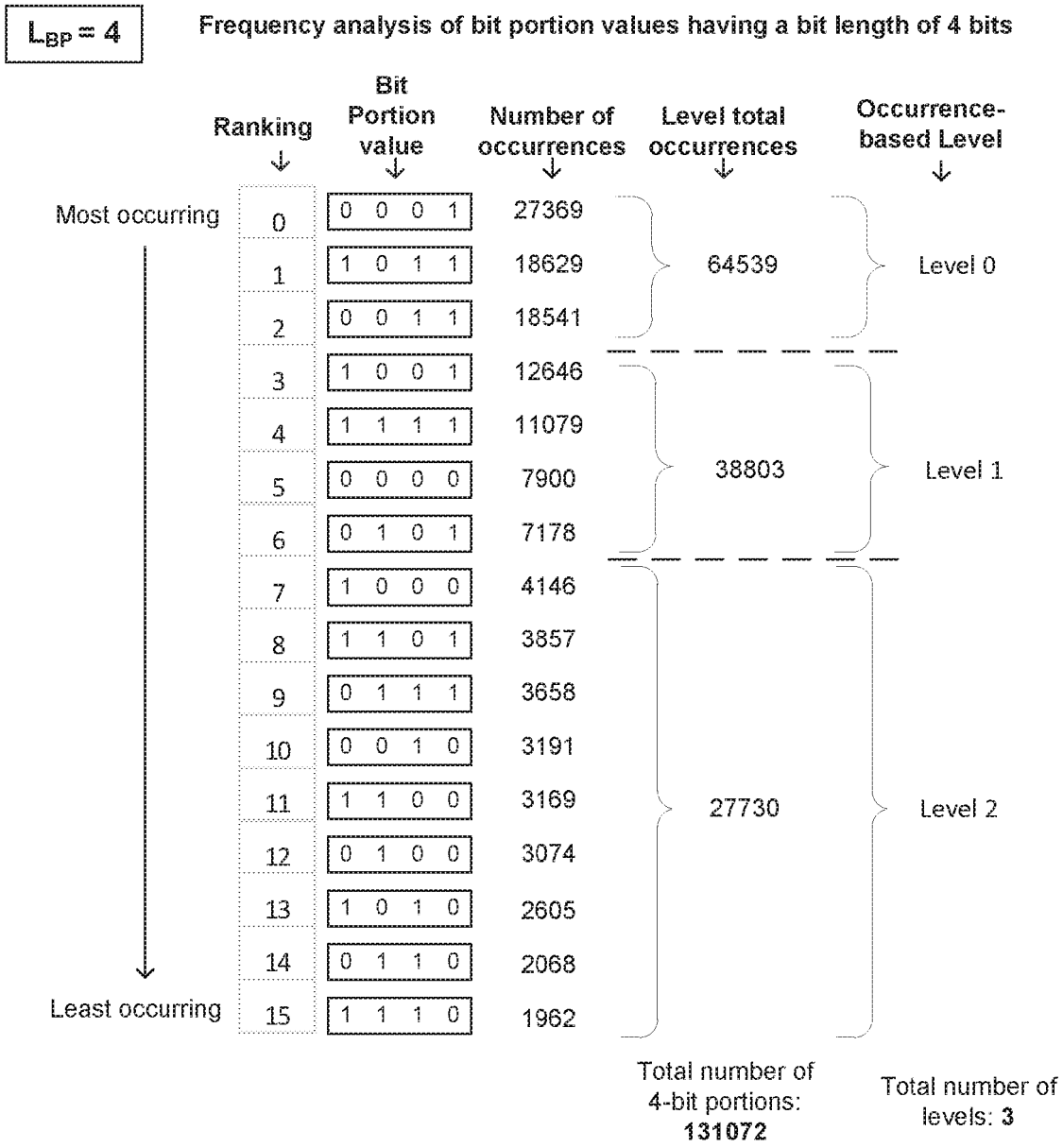
L_{BP} = 4
Frequency analysis of bit portion values having a bit length of 4 bits
Ranking
Bit Portion value
Number of occurrences
Level total occurrences
Occurrence-based Level
Most occurring
Least occurring
0 0 0 1 27369
1 0 1 1 18629
0 0 1 1 18541
1 0 0 1 12646
1 1 1 1 11079
0 0 0 0 7900
0 1 0 1 7178
1 0 0 0 4146
1 1 0 1 3857
0 1 1 1 3658
0 0 1 0 3191
1 1 0 0 3169
0 1 0 0 3074
1 0 1 0 2605
0 1 1 0 2068
1 1 1 0 1962
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
64539
38803
27730
Level 0
Level 1
Level 2
Total number of 4-bit portions: 131072
Total number of levels: 3

Figure 3D

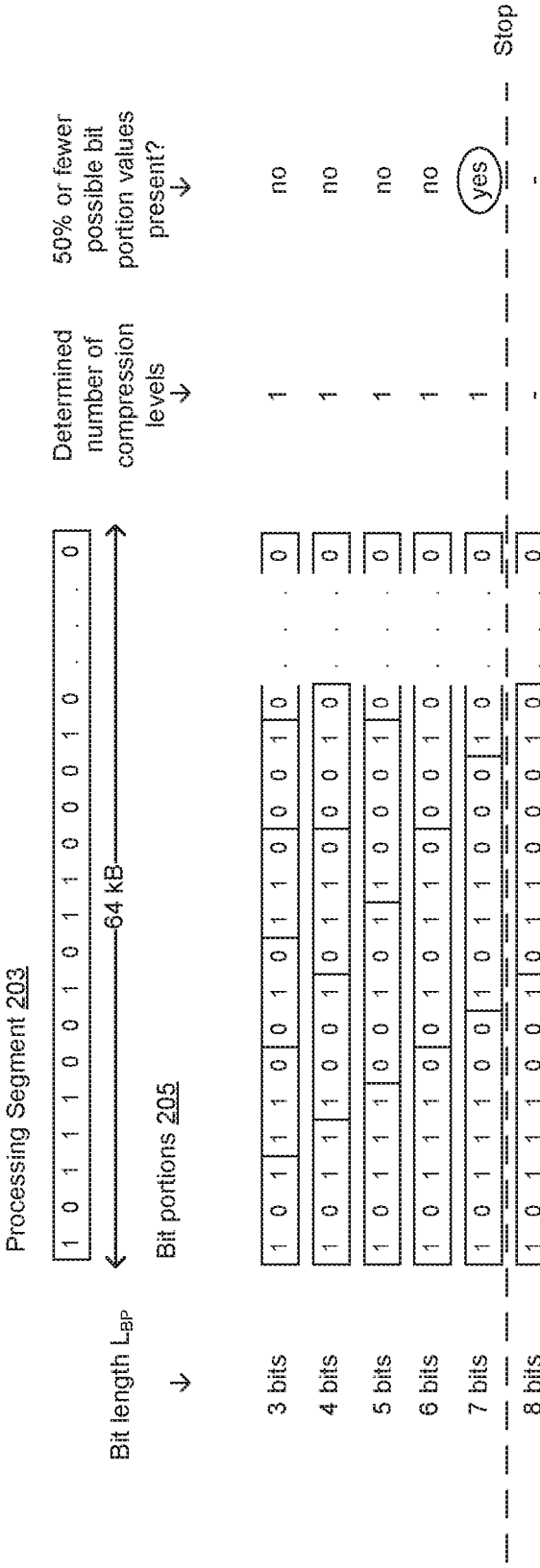
Processing Segment 203
64 kB
Bit portions 205
Bit length L_{BP}
Determined number of compression levels
50% or fewer possible bit portion values present?
3 bits
4 bits
5 bits
6 bits
7 bits
8 bits
no
yes
Stop

Figure 4A

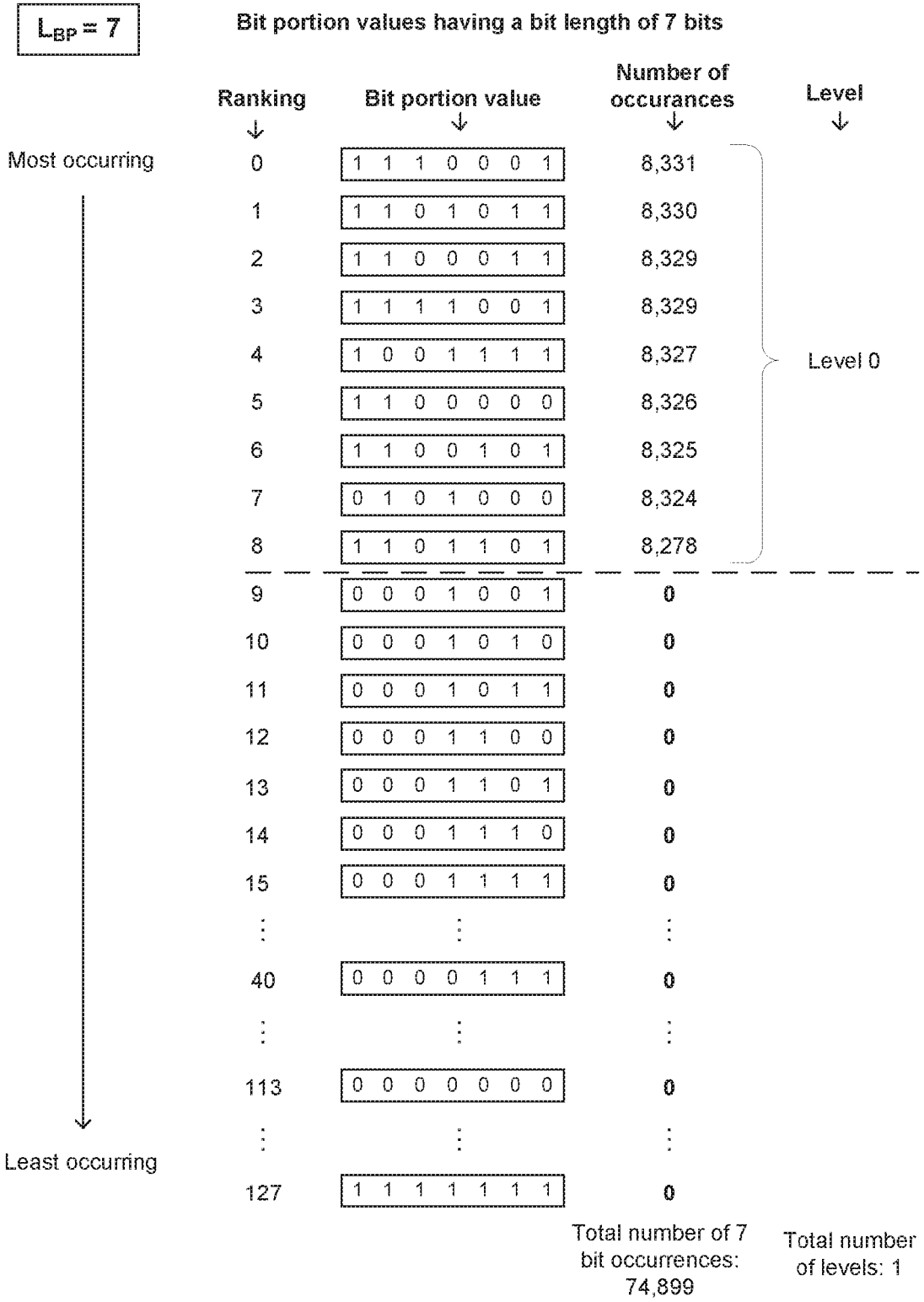

Figure 4B

Fewer than 50% of elements in use

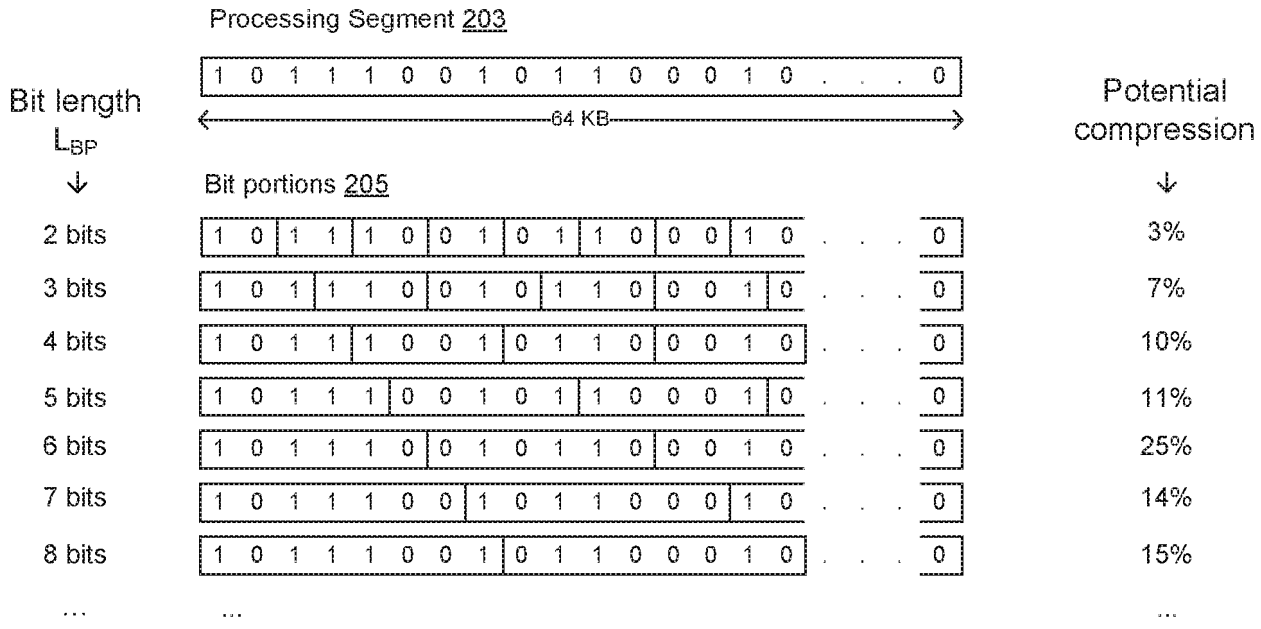
Processing Segment 203
1 0 1 1 1 0 0 1 0 1 1 0 0 0 1 0 . . . 0
64 KB
Bit length
L_{BP}
Bit portions 205
Potential
compression
2 bits
3 bits
4 bits
5 bits
6 bits
7 bits
8 bits
3%
7%
10%
11%
25%
14%
15%

Figure 5A

Frequency analysis of bit portion values having a bit length of 4 bits

$L_{BP} = 4$

Ranking	Bit Portion value	Number of occurrences	Occurrence-based Level (total occurences)	Occurrence-based New BP Value	New BP value in binary	Disambiguation value in binary	Initial label 403	Occurrence-based Label size	No. of bits used
Most occurring ↓									
0	0 0 0 1	27369	Level 0 (64539)	0	000	-	000	3 bits	193617
1	1 0 1 1	18629		1	001	-	001		
2	0 0 1 1	18541		2	010	-	010		
3	1 0 0 1	12646	Level 1 (38803)	3	011	0	0110	4 bits	155212
4	1 1 1 1	11079		3	011	1	0111		
5	0 0 0 0	7900		4	100	0	1000		
6	0 1 0 1	7178		4	100	1	1001		
7	1 0 0 0	3923	Level 2 (27730)	5	101	00	10100	5 bits	128840
8	1 1 0 1	3857		5	101	01	10101		
9	0 1 1 1	3658		5	101	10	10110		
10	0 0 1 0	3414		5	101	11	10111		
11	1 1 0 0	3169		6	110	00	11000		
12	0 1 0 0	3074		6	110	01	11001		
13	1 0 1 0	2605		6	110	10	11010		
14	0 1 1 0	2068		6	110	11	11011		
Least occurring ↓									
15	1 1 1 0	1962		7	111	-	111	3 bits	5886

Total number of bits analysed: **524288**

Total number of 4-bit portions: **131072**

Total number of bits used: **483555**

Figure 5B

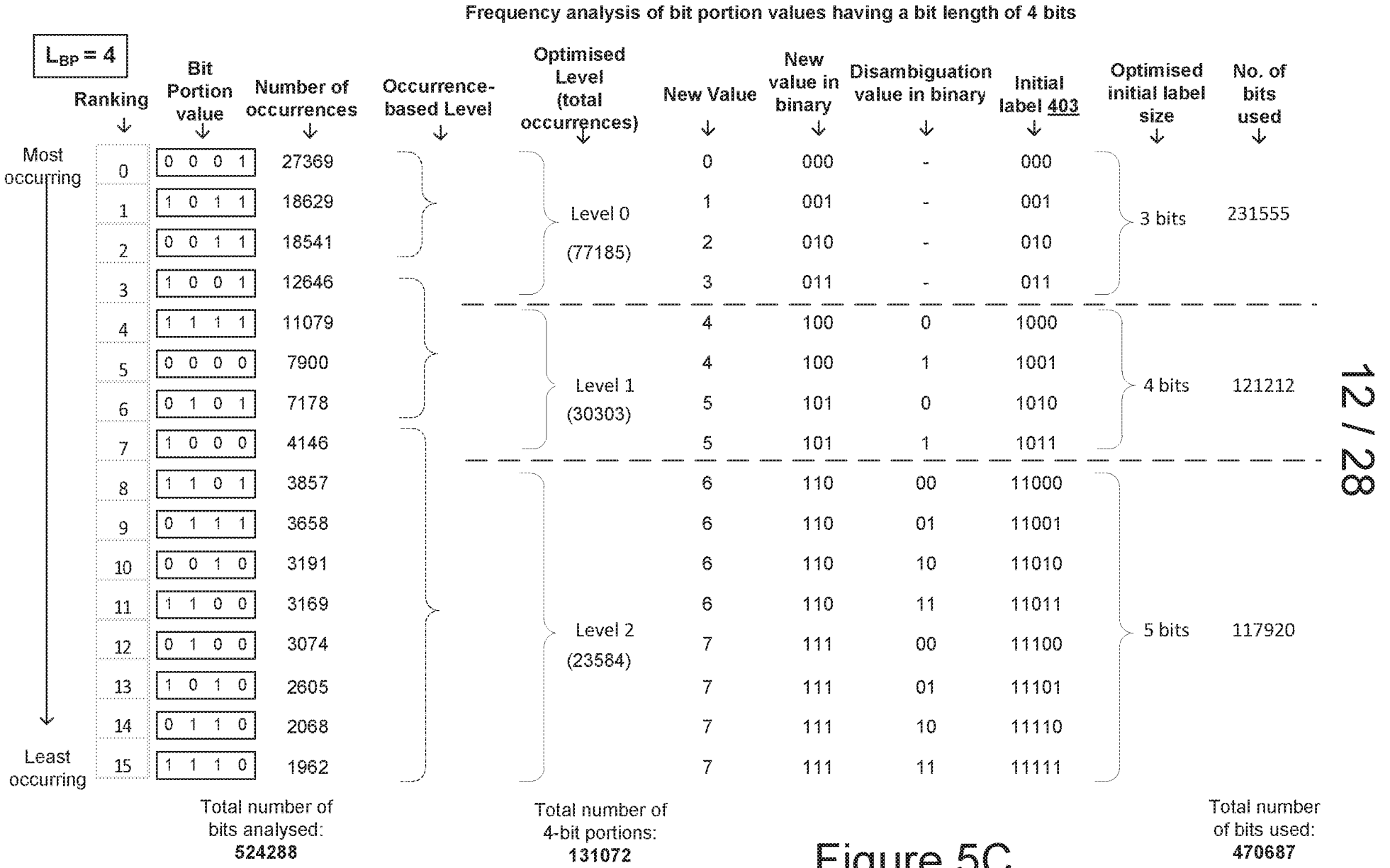
Frequency analysis of bit portion values having a bit length of 4 bits
L_{BP} = 4
Ranking
Bit Portion value
Number of occurrences
Occurrence-based Level
Optimised Level (total occurrences)
New Value
New value in binary
Disambiguation value in binary
Initial label 403
Optimised initial label size
No. of bits used
Most occurring
Least occurring
0 0 0 1 27369
1 0 1 1 18629
0 0 1 1 18541
1 0 0 1 12646
1 1 1 1 11079
0 0 0 0 7900
0 1 0 1 7178
1 0 0 0 4146
1 1 0 1 3857
0 1 1 1 3658
0 0 1 0 3191
1 1 0 0 3169
0 1 0 0 3074
1 0 1 0 2605
0 1 1 0 2068
1 1 1 0 1962
Level 0 (77185)
Level 1 (30303)
Level 2 (23584)
0 000 - 000
1 001 - 001
2 010 - 010
3 011 - 011
4 100 0 1000
4 100 1 1001
5 101 0 1010
5 101 1 1011
6 110 00 11000
6 110 01 11001
6 110 10 11010
6 110 11 11011
7 111 00 11100
7 111 01 11101
7 111 10 11110
7 111 11 11111
3 bits
231555
4 bits
121212
5 bits
117920
Total number of bits analysed: 524288
Total number of 4-bit portions: 131072
Total number of bits used: 470687

Figure 5C

Header Size For 4 Bits			
		Minimum bits required	Maximum bits required
Signature		32	32
Bit Length		5	5
CA Configuration		4	4
Level Counts		64	16
CA Value info		8	64
Total Bits		113	121

Figure 5D

Header			121
Data			470687
Total			470808
In Bytes			58851
% Compression			10.20%

Figure 5E

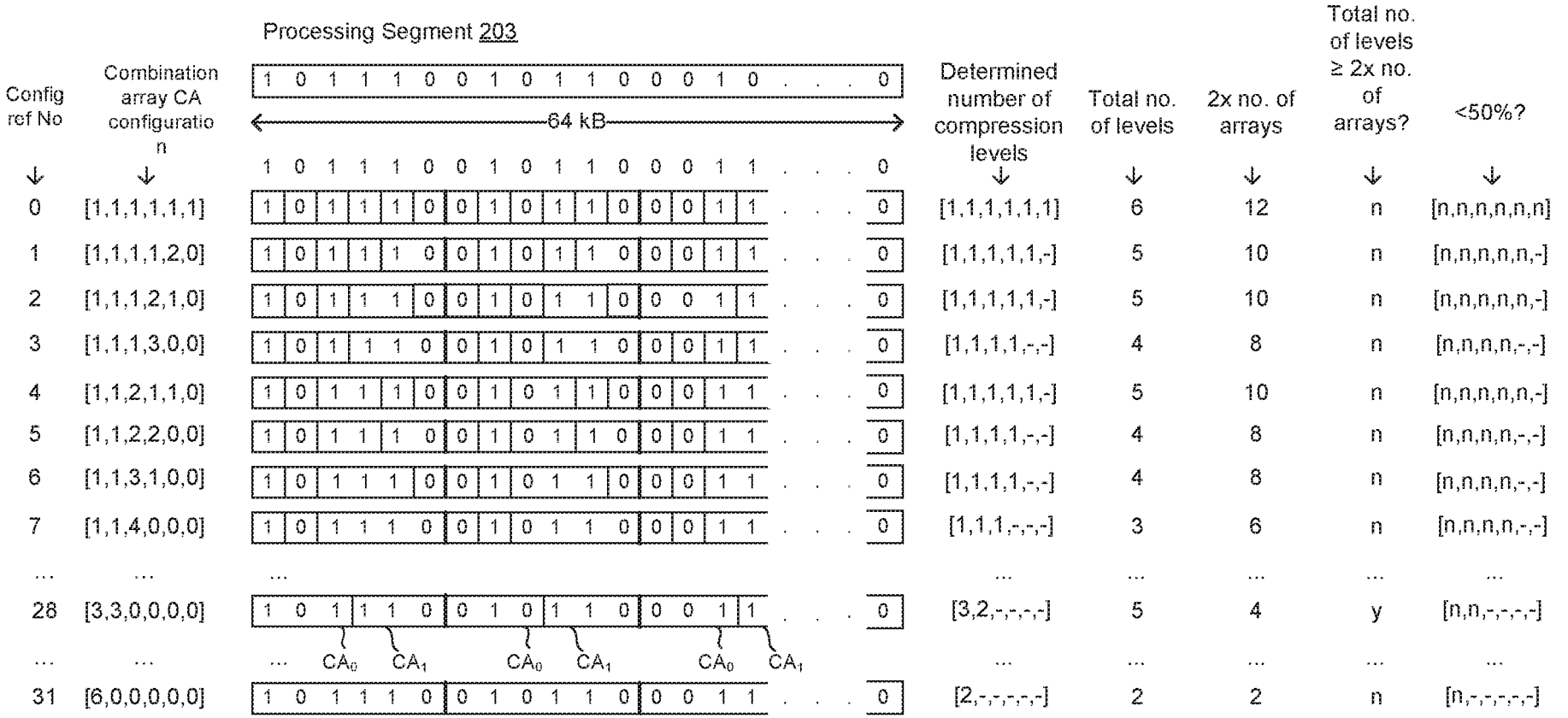
Processing Segment 203
1 0 1 1 1 0 0 1 0 1 1 0 0 0 1 0 . . . 0
64 kB
Config ref No
Combination array CA configuration
Determined number of compression levels
Total no. of levels
2x no. of arrays
Total no. of levels ≥ 2x no. of arrays?
<50%?
0 [1,1,1,1,1,1] [1,1,1,1,1,1] 6 12 n [n,n,n,n,n,n]
1 [1,1,1,1,2,0] [1,1,1,1,1,-] 5 10 n [n,n,n,n,n,-]
2 [1,1,1,2,1,0] [1,1,1,1,1,-] 5 10 n [n,n,n,n,n,-]
3 [1,1,1,3,0,0] [1,1,1,1,-,-] 4 8 n [n,n,n,n,-,-]
4 [1,1,2,1,1,0] [1,1,1,1,1,-] 5 10 n [n,n,n,n,n,-]
5 [1,1,2,2,0,0] [1,1,1,1,-,-] 4 8 n [n,n,n,n,-,-]
6 [1,1,3,1,0,0] [1,1,1,1,-,-] 4 8 n [n,n,n,n,-,-]
7 [1,1,4,0,0,0] [1,1,1,-,-,-] 3 6 n [n,n,n,n,-,-]
28 [3,3,0,0,0,0] [3,2,-,-,-,-] 5 4 y [n,n,-,-,-,-]
CA0 CA1 CA0 CA1 CA0 CA1
31 [6,0,0,0,0,0] [2,-,-,-,-,-] 2 2 n [n,-,-,-,-,-]

Figure 6A

$L_{BP} = 6$ | CA_0

Frequency analysis of 1st combination arrays

	Ranking	CA Value	Number of occurances	Occurrence-based Level
Most occurring	0	0 1 0	30,000	Level 0
	1	0 0 0	20,000	
	2	0 0 1	9,000	Level 1
	3	0 1 1	8,000	
	4	1 0 1	8,000	
	5	1 1 0	4,400	Level 2
	6	1 1 1	4,001	
Least occurring	7	1 0 0	3,981	

Total number of 3-bit portions: **87382**

Total number of levels: 3

Figure 6B

$L_{BP} = 6$ | CA_1

Frequency analysis of 2nd combination arrays

	Ranking	CA Value	Number of occurances	Occurrence-based Level
Most occurring	0	0 1 1	49,940	Level 0
	1	0 0 1	8,752	Level 1
	2	0 0 0	6,240	
	3	0 1 0	5,251	
	4	1 0 0	4,727	
	5	1 1 1	4,375	Level 2
	6	1 1 0	4,158	
Least occurring	7	1 0 1	3,938	

Total number of 3-bit portions: **87382**

Total number of levels: 3

Figure 6C

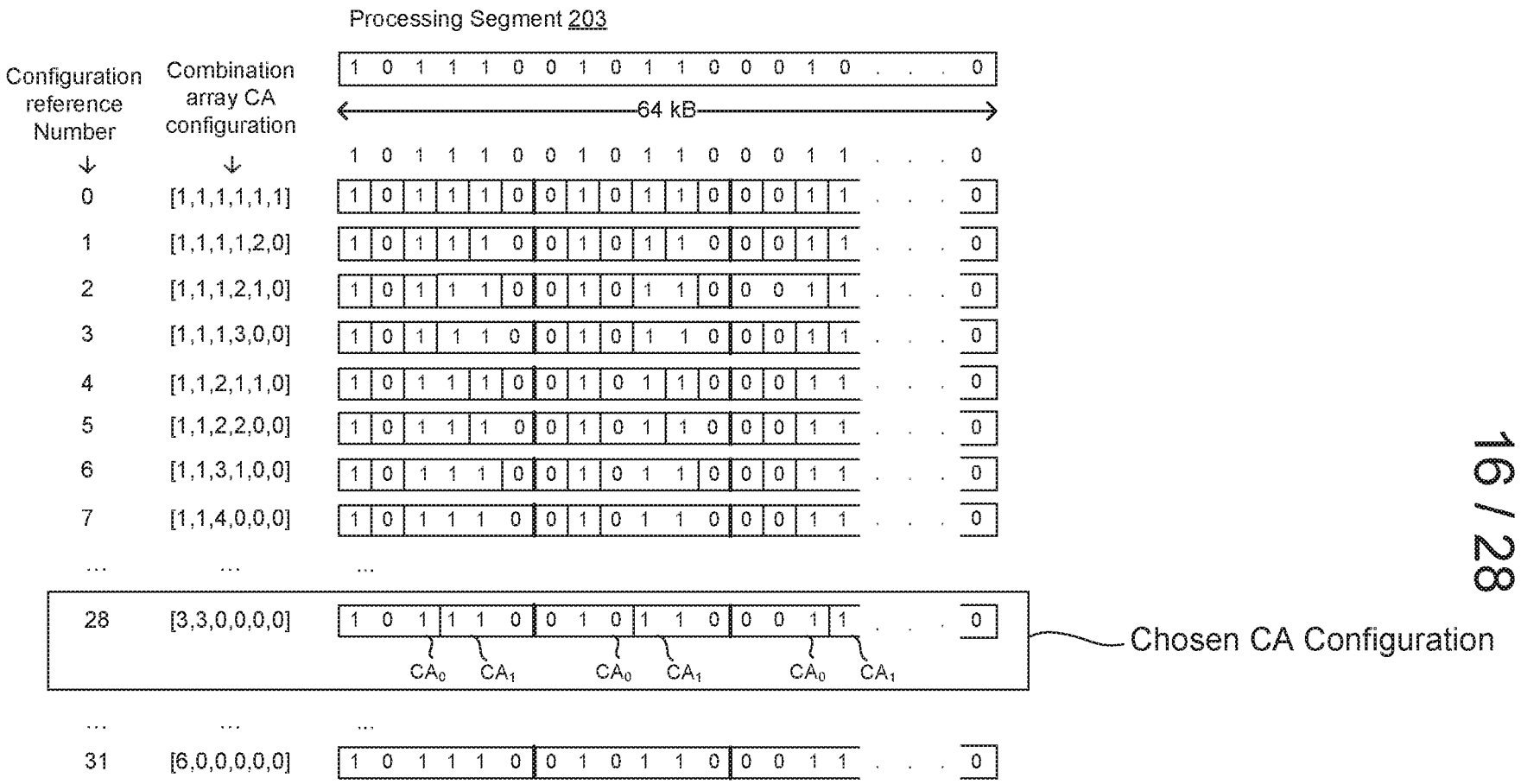
Processing Segment 203
Configuration reference Number
Combination array CA configuration
64 kB
0 [1,1,1,1,1,1]
1 [1,1,1,1,2,0]
2 [1,1,1,2,1,0]
3 [1,1,1,3,0,0]
4 [1,1,2,1,1,0]
5 [1,1,2,2,0,0]
6 [1,1,3,1,0,0]
7 [1,1,4,0,0,0]
28 [3,3,0,0,0,0]
31 [6,0,0,0,0,0]
CA0 CA1 CA0 CA1 CA0 CA1
Chosen CA Configuration

Figure 6D

Frequency analysis of 1st combination arrays

CA_0

Ranking	CA Value	Number of occurances	Occurrence-based Level
Most occurring			
0	0 1 0	30,000	Level 0
1	0 0 0	20,000	
2	0 0 1	9,000	Level 1
3	0 1 1	8,000	
4	1 0 1	8,000	
5	1 1 0	4,400	Level 2
6	1 1 1	4,001	
7	1 0 0	3,981	
Least occurring		Total number of 3-bit portions: **87382**	Total number of levels: 3

Optimised Level	Initial Value 690	New CA Value 701	Instance	Disambiguation value 703
Level 0	0	0	0	-
	1	1	0	-
Level 1	2	2	0	0
	2	2	1	1
Level 2	3	3	0	0
	3	3	1	1
	3	3	2	2
	3	3	3	3

Figure 7A

CA_1

Frequency analysis of 2nd combination arrays

Ranking	CA Value	Number of occurances	Occurrence-based Level
Most occurring			
0	0 1 1	49,940	Level 0
1	0 0 1	8,752	Level 1
2	0 0 0	6,240	
3	0 1 0	5,251	
4	1 0 0	4,727	
5	1 1 1	4,375	Level 2
6	1 1 0	4,158	
7	1 0 1	3,938	
Least occurring		Total number of 3-bit portions: **87382**	Total number of levels: 3

Optimised Level	Initial Value 690	New CA Value 701	Instance	Disambiguation value 703: combining with level 0 of CA_0	Disambiguation value 703: combining with level 1 of CA_0	Disambiguation value 703: combining with level 2 of CA_0
Level 0	0	0	0	-	-	-
	1	4	0	-	-	-
Level 1	2	8	0	0	0	0
	2	8	1	1	2	4
Level 2	3	12	0	0	0	0
	3	12	1	1	2	4
	3	12	2	2	4	8
	3	12	3	3	6	12

Figure 7B

		New CA_1 Values			
		0	4	8	12
New CA_0 Values	0	0	4	8	12
	1	1	5	9	13
	2	2	6	10	14
	3	3	7	11	15
		Combined New CA Values			

Figure 8A

		New CA_1 Values			
		0	4	8	12
New CA_0 Values	0	00000	0100	1000	1100
	1	00001	0101	1001	1101
	2	00010	0110	1010	1110
	3	00011	0111	1011	1111
		Combined New CA Values in binary			

Figure 8B

		New CA_1 Values					
		Y_0	Y_1	Y_2	Y_3	...	Y_n
New CA_0 Values	X_0	$X_0 + Y_0$	$X_0 + Y_1$	$X_0 + Y_2$	$X_0 + Y_3$	...	$X_0 + Y_n$
	X_1	$X_1 + Y_0$	$X_1 + Y_1$	$X_1 + Y_2$	$X_1 + Y_3$	...	$X_1 + Y_n$
	X_2	$X_2 + Y_0$	$X_2 + Y_1$	$X_2 + Y_2$	$X_2 + Y_3$	...	$X_2 + Y_n$
	X_3	$X_3 + Y_0$	$X_3 + Y_1$	$X_3 + Y_2$	$X_3 + Y_3$	...	$X_3 + Y_n$
	...	...	...	...	...	...	...
	X_n	$X_n + Y_0$	$X_n + Y_1$	$X_n + Y_2$	$X_n + Y_3$	...	$X_n + Y_n$
		Combined New CA Values using addition					

Figure 8C

		New CA_1 Values				
		y_0	y_1	y_2	y_3	y_4
New CA_0 Values	x_0	z_0	z_1	z_2	z_3	z_4
	x_1	z_5	z_6	z_7	z_8	z_9
	x_2	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}
	x_3	z_{15}	z_{16}	z_{17}	z_{18}	z_{19}
		Combined New CA Values				

Figure 8D

CA_0 (LEVEL 0)
CA_1 (LEVEL 0)

No Disambiguation Values

CA_0 (LEVEL 1)
CA_1 (LEVEL 0)

CA_0 Dis. Values	CA_1 Dis. Values: -
0	0
1	1
	Combined Dis. Values

CA_0 (LEVEL 2)
CA_1 (LEVEL 0)

CA_0 Dis. Values	CA1 Dis. Values: -
0	00
1	01
2	10
3	11
	Combined Dis. Values

CA_0 (LEVEL 0)
CA_1 (LEVEL 1)

CA_0 Dis. Values	CA_1 Dis. Values: 0	1
-	0	1
	Combined Dis. Values	

CA_0 (LEVEL 1)
CA_1 (LEVEL 1)

CA_0 Dis. Values	CA_1 Dis. Values: 0	2
0	0	10
1	1	11
	Combined Dis. Values	

CA_0 (LEVEL 2)
CA_1 (LEVEL 1)

CA_0 Dis. Values	CA_1 Dis. Values: 0	4
0	000	100
1	001	101
2	010	110
3	011	111
	Combined Dis. Values	

CA_0 (LEVEL 0)
CA_1 (LEVEL 2)

CA_0 Dis. Values	CA_1 Dis. Values: 0	1	2	3
-	00	01	10	11
	Combined Dis. Values			

CA_0 (LEVEL 1)
CA_1 (LEVEL 2)

CA_0 Dis. Values	CA_1 Dis. Values: 0	2	4	6
0	000	010	100	110
1	001	011	101	111
	Combined Dis. Values			

CA_0 (LEVEL 2)
CA_1 (LEVEL 2)

CA_0 Dis. Values	CA_1 Dis. Values: 0	4	8	12
0	0000	0100	1000	1100
1	0001	0101	1001	1101
2	0010	0110	1010	1110
3	0011	0111	1011	1111
	Combined Dis. Values			

Figure 9

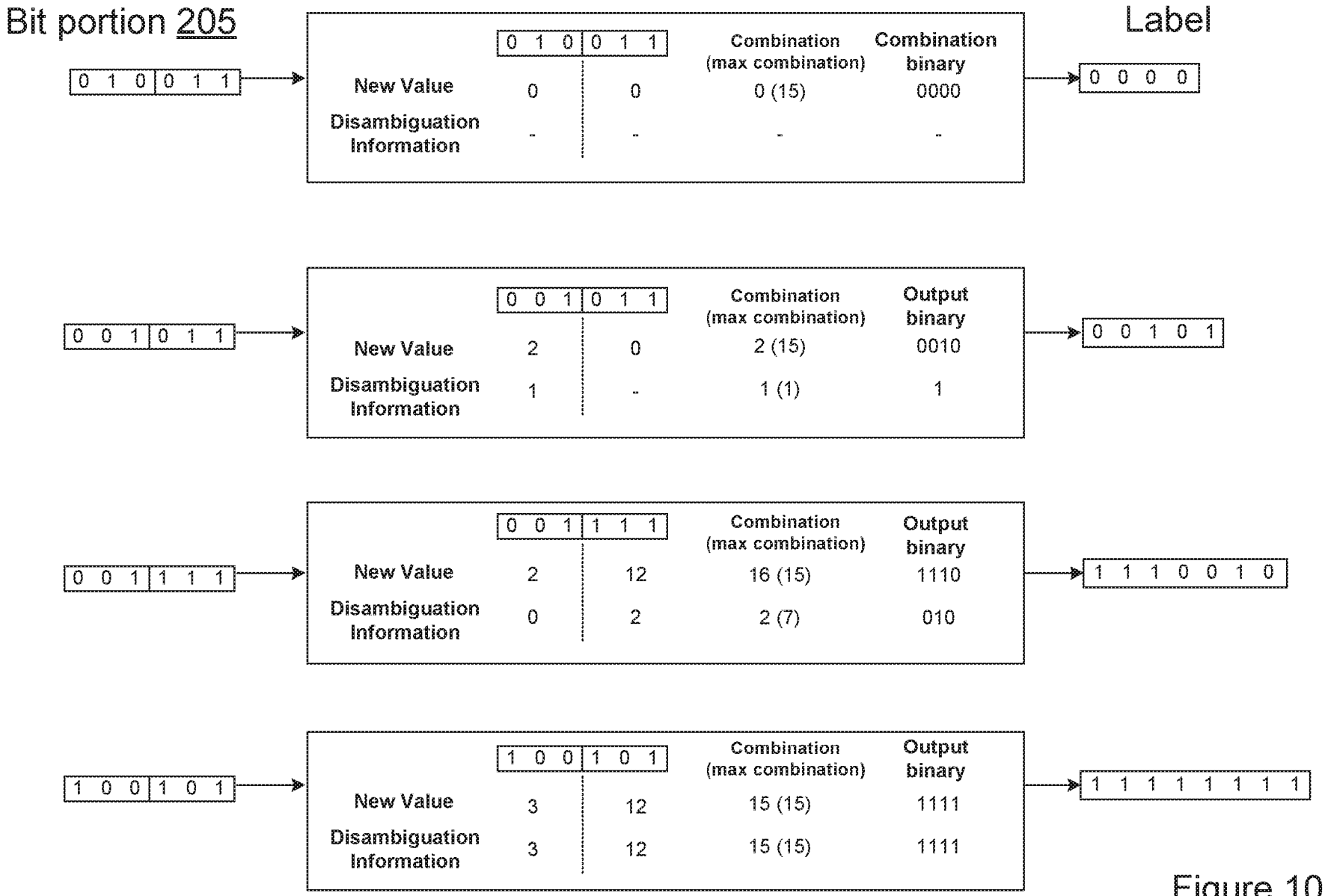
Bit portion 205
Label
0 1 0 0 1 1
0 1 0 0 1 1
Combination (max combination)
Combination binary
New Value
0
0
0 (15)
0000
Disambiguation Information
-
-
-
-
0 0 0 0
0 0 1 0 1 1
0 0 1 0 1 1
Combination (max combination)
Output binary
New Value
2
0
2 (15)
0010
Disambiguation Information
1
-
1 (1)
1
0 0 1 0 1
0 0 1 1 1 1
0 0 1 1 1 1
Combination (max combination)
Output binary
New Value
2
12
16 (15)
1110
Disambiguation Information
0
2
2 (7)
010
1 1 1 0 0 1 0
1 0 0 1 0 1
1 0 0 1 0 1
Combination (max combination)
Output binary
New Value
3
12
15 (15)
1111
Disambiguation Information
3
12
15 (15)
1111
1 1 1 1 1 1 1 1

Figure 10

Label outputs using CA configuration [3,3,0,0,0,0] (No Occurrence Sort)									
Value	Bit portion	CA0	CA1	Label	Value	Bit portion	CA0	CA1	Label
0	000000	000	000	10010	32	100000	100	000	1011011
1	000001	000	001	0101	33	100001	100	001	011111
2	000010	000	010	10011	34	100010	100	010	1011111
3	000011	000	011	0001	35	100011	100	011	001111
4	000100	000	100	110100	36	100100	100	100	11110011
5	000101	000	101	110111	37	100101	100	101	11111111
6	000110	000	110	110110	38	100110	100	110	11111011
7	000111	000	111	110101	39	100111	100	111	11110111
8	001000	001	000	101000	40	101000	101	000	1011000
9	001001	001	001	01100	41	101001	101	001	011100
10	001010	001	010	101010	42	101010	101	010	1011100
11	001011	001	011	00100	43	101011	101	011	001100
12	001100	001	100	1110000	44	101100	101	100	11110000
13	001101	001	101	1110110	45	101101	101	101	11111100
14	001110	001	110	1110100	46	101110	101	110	11111000
15	001111	001	111	1110010	47	101111	101	111	11110100
16	010000	010	000	10000	48	110000	110	000	1011001
17	010001	010	001	01000	49	110001	110	001	011101
18	010010	010	010	10001	50	110010	110	010	1011101
19	010011	010	011	0000	51	110011	110	011	001101
20	010100	010	100	110000	52	110100	110	100	11110001
21	010101	010	101	110011	53	110101	110	101	11111101
22	010110	010	110	110010	54	110110	110	110	11111001
23	010111	010	111	110001	55	110111	110	111	11110101
24	011000	011	000	101001	56	111000	111	000	1011010
25	011001	011	001	01011	57	111001	111	001	011110
26	011010	011	010	101011	58	111010	111	010	1011110
27	011011	011	011	00101	59	111011	111	011	001110
28	011100	011	100	1110001	60	111100	111	100	11110010
29	011101	011	101	1110111	61	111101	111	101	11111110
30	011110	011	110	1110101	62	111110	111	110	11111010
31	011111	011	111	1110011	63	111111	111	111	11110110

Figure 11

CA_0 - 5 Bits			CA_1 - 3 Bits		
Original	Original CA Value (Binary)	New CA value	Original	Original CA Value (Binary)	New CA value
0	00000	0	0	000	0
1	00001	1	1	001	16
2	00010	2	2	010	32
3	00011	3	3	011	32
4	00100	4	4	100	48
5	00101	4	5	101	48
6	00110	5	6	110	48
7	00111	5	7	111	48
8	01000	6			
9	01001	6			
10	01010	7			
11	01011	7			
12	01100	8			
13	01101	8			
14	01110	9			
15	01111	9			
16	10000	10			
17	10001	10			
18	10010	11			
19	10011	11			
20	10100	12			
21	10101	12			
22	10110	13			
23	10111	13			
24	11000	14			
25	11001	14			
26	11010	14			
27	11011	14			
28	11100	15			
29	11101	15			
30	11110	15			
31	11111	15			

Figure 12A

CA_0 Level	CA_1 Level	Disambiguation Bit Length	Label Bit Length
0	0	0	6
0	1	1	7
0	2	2	8
1	0	1	7
1	1	2	8
1	2	3	9
2	0	2	8
2	1	3	9
2	2	4	10

Figure 12B

5 Bits			3 Bits		
Original	Original CA Value (Binary)	New CA Value	Original	Original CA Value (Binary)	New CA Value
0	00000	0	0	000	0
1	00001	1	1	001	8
2	00010	2	2	010	16
3	00011	2	3	011	16
4	00100	3	4	100	24
5	00101	3	5	101	24
6	00110	3	6	110	24
7	00111	3	7	111	24
8	01000	4			
9	01001	4			
10	01010	4			
11	01011	4			
12	01100	5			
13	01101	5			
14	01110	5			
15	01111	5			
16	10000	6			
17	10001	6			
18	10010	6			
19	10011	6			
20	10100	6			
21	10101	6			
22	10110	6			
23	10111	6			
24	11000	7			
25	11001	7			
26	11010	7			
27	11011	7			
28	11100	7			
29	11101	7			
30	11110	7			
31	11111	7			

Figure 12C

CA_0 Level	CA_1 Level	Disambiguation Bit Length	Label Bit Length
0	0	0	5
0	1	1	6
0	2	2	7
1	0	1	6
1	1	2	7
1	2	3	8
2	0	2	7
2	1	3	8
2	2	4	9
3	0	3	8
3	1	4	9
3	2	5	10

Figure 12D

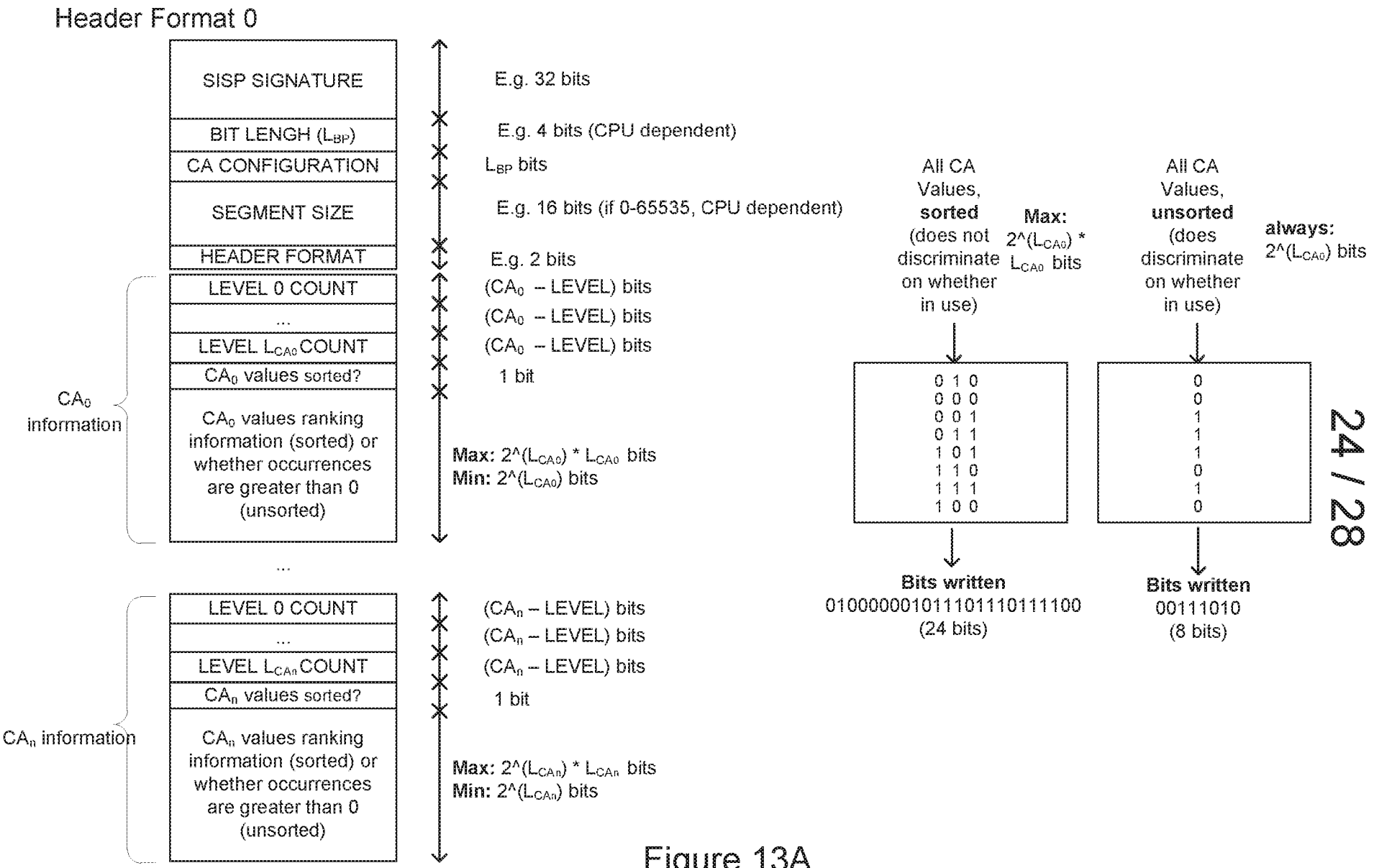
Header Format 0
SISP SIGNATURE
BIT LENGH (L_{BP})
CA CONFIGURATION
SEGMENT SIZE
HEADER FORMAT
LEVEL 0 COUNT
...
LEVEL L_{CA0} COUNT
CA_0 values sorted?
CA_0 values ranking information (sorted) or whether occurrences are greater than 0 (unsorted)
CA_0 information
...
LEVEL 0 COUNT
...
LEVEL L_{CAn} COUNT
CA_n values sorted?
CA_n values ranking information (sorted) or whether occurrences are greater than 0 (unsorted)
CA_n information
E.g. 32 bits
E.g. 4 bits (CPU dependent)
L_{BP} bits
E.g. 16 bits (if 0-65535, CPU dependent)
E.g. 2 bits
(CA_0 – LEVEL) bits
(CA_0 – LEVEL) bits
(CA_0 – LEVEL) bits
1 bit
Max: 2^(L_{CA0}) * L_{CA0} bits
Min: 2^(L_{CA0}) bits
(CA_n – LEVEL) bits
(CA_n – LEVEL) bits
(CA_n – LEVEL) bits
1 bit
Max: 2^(L_{CAn}) * L_{CAn} bits
Min: 2^(L_{CAn}) bits
All CA Values, sorted (does not discriminate on whether in use)
Max: 2^(L_{CA0}) * L_{CA0} bits
0 1 0
0 0 0
0 0 1
0 1 1
1 0 1
1 1 0
1 1 1
1 0 0
Bits written
010000001011101110111100
(24 bits)
All CA Values, unsorted (does discriminate on whether in use)
always: 2^(L_{CA0}) bits
0
0
1
1
1
0
1
0
Bits written
00111010
(8 bits)
24 / 28

Figure 13A

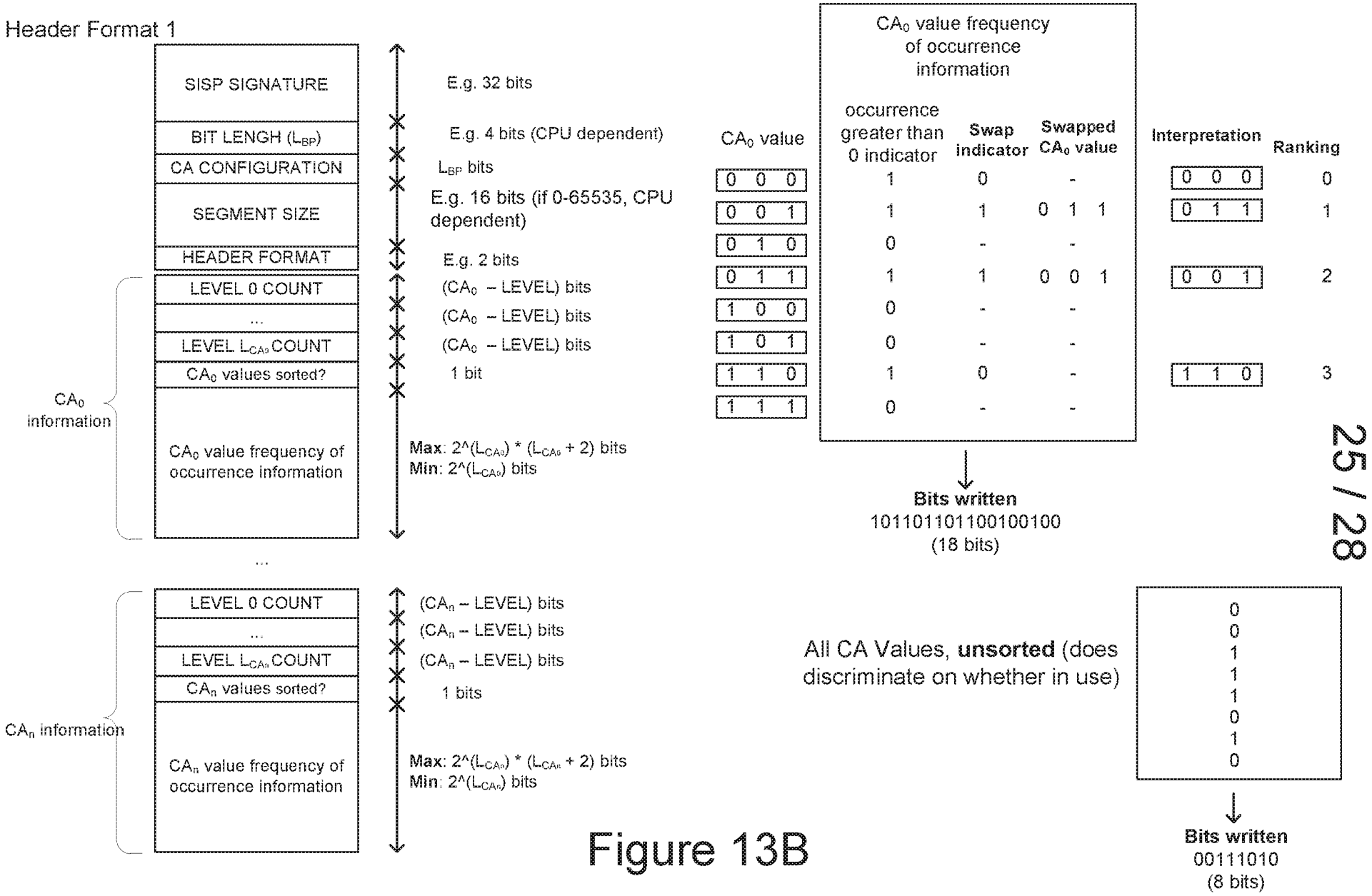
Header Format 1
SISP SIGNATURE
BIT LENGH (L_{BP})
CA CONFIGURATION
SEGMENT SIZE
HEADER FORMAT
LEVEL 0 COUNT
...
LEVEL L_{CA_0} COUNT
CA_0 values sorted?
CA_0 value frequency of occurrence information
CA_0 information
E.g. 32 bits
E.g. 4 bits (CPU dependent)
L_{BP} bits
E.g. 16 bits (if 0-65535, CPU dependent)
E.g. 2 bits
(CA_0 – LEVEL) bits
(CA_0 – LEVEL) bits
(CA_0 – LEVEL) bits
1 bit
Max: 2^(L_{CA_0}) * (L_{CA_0} + 2) bits
Min: 2^(L_{CA_0}) bits
...
LEVEL 0 COUNT
...
LEVEL L_{CA_n} COUNT
CA_n values sorted?
CA_n value frequency of occurrence information
CA_n information
(CA_n – LEVEL) bits
(CA_n – LEVEL) bits
(CA_n – LEVEL) bits
1 bits
Max: 2^(L_{CA_n}) * (L_{CA_n} + 2) bits
Min: 2^(L_{CA_n}) bits
CA_0 value
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
CA_0 value frequency of occurrence information
occurrence greater than 0 indicator
Swap indicator
Swapped CA_0 value
1 0 -
1 1 0 1 1
0 - -
1 1 0 0 1
0 - -
0 - -
1 0 -
0 - -
Interpretation
Ranking
0 0 0 0
0 1 1 1
0 0 1 2
1 1 0 3
Bits written
10110110110010 0100
(18 bits)
All CA Values, unsorted (does discriminate on whether in use)
0
0
1
1
1
1
0
1
0
Bits written
00111010
(8 bits)

Figure 13B

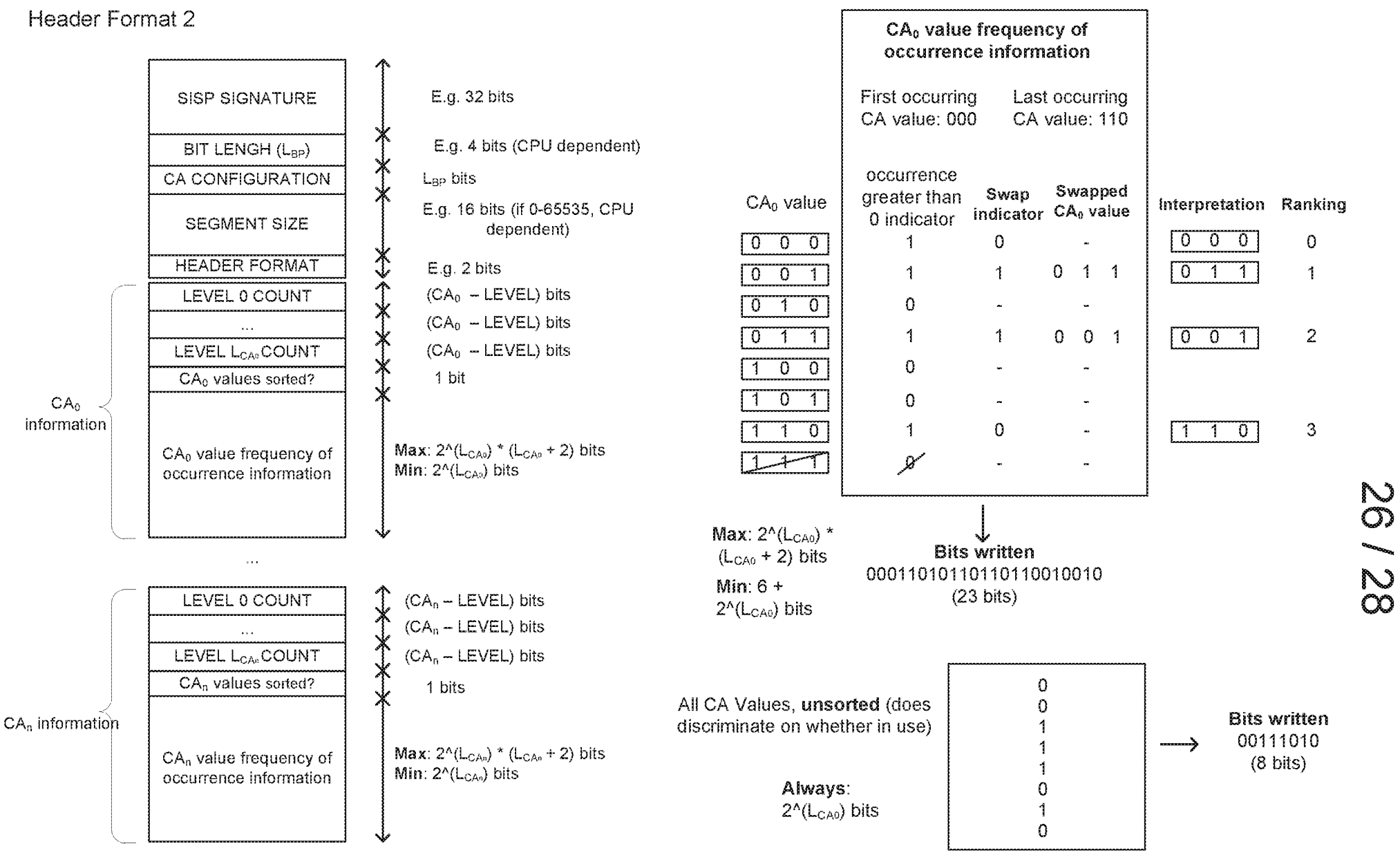
Header Format 2
SISP SIGNATURE
BIT LENGH (L_{BP})
CA CONFIGURATION
SEGMENT SIZE
HEADER FORMAT
LEVEL 0 COUNT
...
LEVEL L_{CA0} COUNT
CA_0 values sorted?
CA_0 value frequency of occurrence information
CA_0 information
LEVEL 0 COUNT
...
LEVEL L_{CAn} COUNT
CA_n values sorted?
CA_n value frequency of occurrence information
CA_n information
E.g. 32 bits
E.g. 4 bits (CPU dependent)
L_{BP} bits
E.g. 16 bits (if 0-65535, CPU dependent)
E.g. 2 bits
(CA_0 – LEVEL) bits
(CA_0 – LEVEL) bits
(CA_0 – LEVEL) bits
1 bit
Max: 2^(L_{CA0}) * (L_{CA0} + 2) bits
Min: 2^(L_{CA0}) bits
(CA_n – LEVEL) bits
(CA_n – LEVEL) bits
(CA_n – LEVEL) bits
1 bits
Max: 2^(L_{CAn}) * (L_{CAn} + 2) bits
Min: 2^(L_{CAn}) bits
CA_0 value frequency of occurrence information
First occurring CA value: 000
Last occurring CA value: 110
CA_0 value
occurrence greater than 0 indicator
Swap indicator
Swapped CA_0 value
Interpretation
Ranking
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
1 - - 0
1 1 0 1 1 0
0 - -
1 1 0 0 0 1
0 - -
0 - -
1 0 - 1 1 0
0 - -
0 0 0 0
0 1 1 1
0 0 1 2
1 1 0 3
Max: 2^(L_{CA0}) * (L_{CA0} + 2) bits
Min: 6 + 2^(L_{CA0}) bits
Bits written
00011010110110110010010
(23 bits)
All CA Values, unsorted (does discriminate on whether in use)
Always: 2^(L_{CA0}) bits
0 0 1 1 1 0 1 0
Bits written
00111010
(8 bits)

Figure 13C

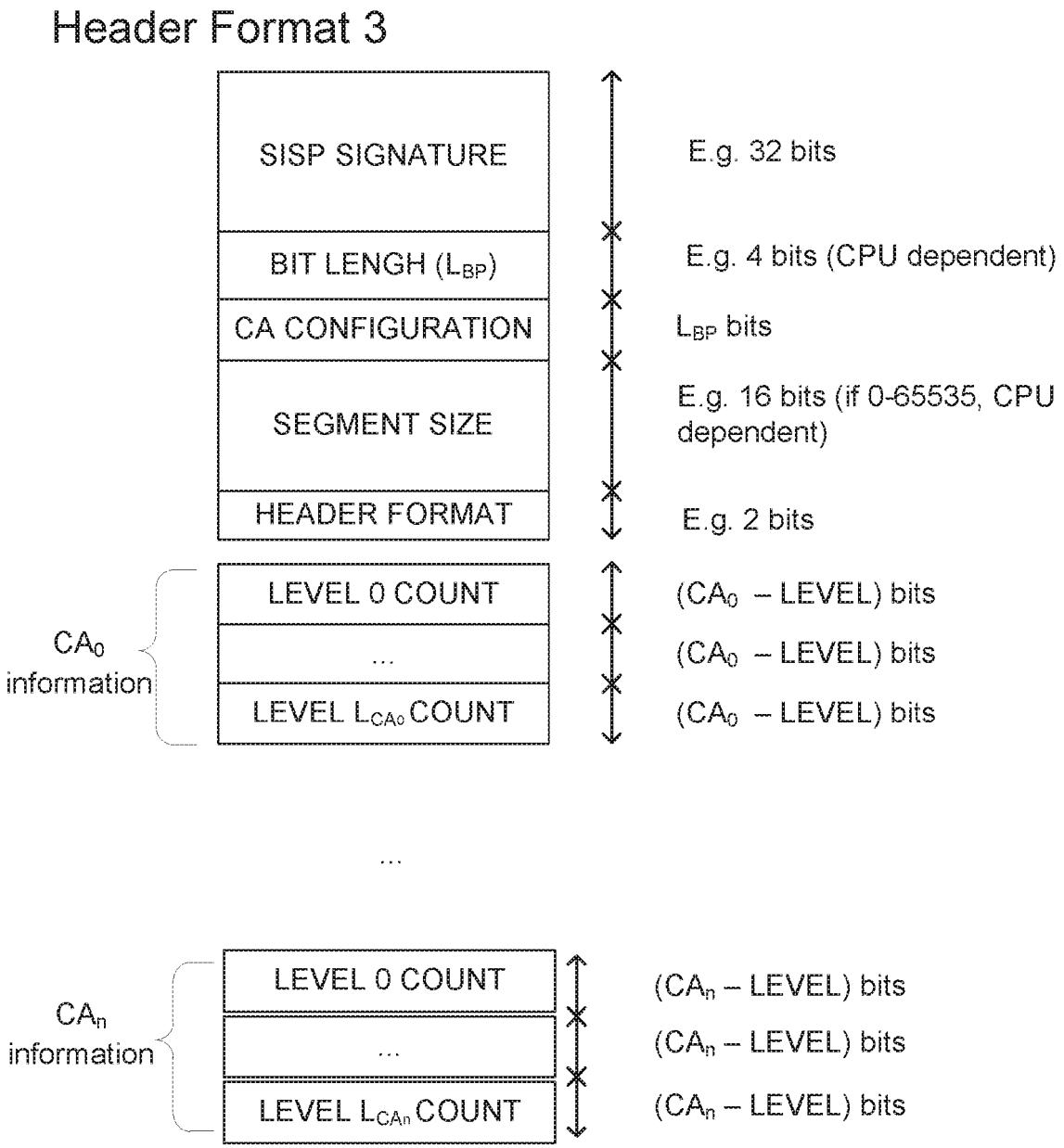
Header Format 3
SISP SIGNATURE
E.g. 32 bits
BIT LENGH (L_{BP})
E.g. 4 bits (CPU dependent)
CA CONFIGURATION
L_{BP} bits
SEGMENT SIZE
E.g. 16 bits (if 0-65535, CPU dependent)
HEADER FORMAT
E.g. 2 bits
CA_0 information
LEVEL 0 COUNT
(CA_0 – LEVEL) bits
...
(CA_0 – LEVEL) bits
LEVEL L_{CA0} COUNT
(CA_0 – LEVEL) bits
...
CA_n information
LEVEL 0 COUNT
(CA_n – LEVEL) bits
...
(CA_n – LEVEL) bits
LEVEL L_{CAn} COUNT
(CA_n – LEVEL) bits

Figure 13D

Number of BP values/CA values in processing segment with an occurrence greater than 0	Target Maximum CA/BP Value
1	1
2	1
3	1
4	3
5	3
6	3
7	3
8	3
9	3
10	3
11	3
12	3
13	3
14	3
15	3
16	7
17	7
...	...
30	7
32	7
33	7
....	....
64	7
....	....
127	7
128	7
...	...
254	7
255	7
256	15

Figure 14

Data Processing Method and Apparatus

The present invention relates to a method and apparatus of processing data, in particular for compressing (and/or encrypting) data.

Background

Currently, information held on a computer is stored as ones and zeros (bits) which are grouped into sets of eight bits which are referred to as bytes. Two bytes are referred to as a word (16 bits), and four bytes are referred to as a double word (32 bits) or can be used as the mathematical storage referred to as a 32-bit integer (int32 or Long). An integer which has a bit length of 32 can hold a value between -2147483648 and +2147483647; or by removing the sign and making it an unsigned 32-bit integer (UInt32), the longest number that can be stored is 4294967295 (2^{32}-1).

It is desirable to represent information using the smallest number of bits possible in order to reduce the space required for storage and to minimise the resources required for signalling information from one entity to another. In computer science and information theory, data compression (also referred to as source coding) involves encoding information using fewer bits than the original representation. Furthermore, it is important that sensitive data, represented using the American Standard Code Information Interchange (ASCII) standard or by other means, is protected, for example by preventing access to this data by unauthorised persons or machines. Therefore, methods of encrypting and decrypting data form an integral part of information technology.

Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. In contrast, lossy compression reduces the total number of bits by identifying marginally important information and removing it.

Once data has been compressed, it must subsequently decompressed in order for it to be used. Both compression and decompression require computer processing. Therefore, data compression/decompression must find a compromise between the level of compression achieved and the computer processing required for compression and decompression. For example, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough for it to be watched as it is being decompressed, and the option to decompress the video in full before watching it may be inconvenient and may require additional storage.

The present invention seeks to provide improved methods of compression and/or decompression and/or improved methods of encryption and/or decryption.

According to one aspect of the invention there is provided a method of processing data comprising an input sequence of bits, the method comprising the steps of: (i) identifying a processing bit length for use in processing said input sequence of bits; (ii) dividing the input sequence of bits into a plurality of portions wherein each portion has a respective portion bit length equal to said processing bit length and wherein the bits in each portion are arranged in a respective portion permutation; (iii) respectively sub-dividing each portion into a plurality of sub-divisions comprising at least a first sub-division and a second sub-division, wherein each sub-

division of the plurality of sub-divisions comprises at least one bit, wherein the at least one bit of each first sub-division is arranged in a respective first sub-division permutation, and wherein the at least one bit of each second sub-division is arranged in a respective second sub-division permutation; (iv) performing frequency analysis: to determine, for each of a plurality of possible first sub-division permutations, how many times, within said input sequence of bits, a portion comprises a first sub-division having bits arranged in that possible first sub-division permutation; and to determine, for each of a plurality of possible second sub-division permutations, how many times, within said input sequence of bits, a portion comprises a second sub-division having bits arranged in that possible second sub-division permutation; (v) assigning, based on said frequency analysis, a first respective sub-division value to each of said plurality of possible first sub-division permutations and assigning a second respective sub-division value to each of said plurality of possible second sub-division permutations; (vi) for each portion permutation of a plurality of possible portion permutations, generating a respective label representing that portion permutation, wherein said generating comprises combining: the first sub-division value assigned to the first sub-division permutation corresponding to the first sub-division of that portion permutation; with the second sub-division value assigned to the second sub-division permutation corresponding to the second sub-division of that portion permutation; wherein said respective label comprises a representation of a combined value resulting from said combining; and (vii) forming a processed sequence of bits by replacing, within said input sequence of bits, bit portions comprising bits arranged in one of said plurality of possible portion permutations, with the respective label representing that one of said plurality of possible portion permutations.

When generating, for each portion permutation, a respective label representing that portion permutation, said combining may comprise arithmetically adding said first sub-division value assigned to the first sub-division permutation corresponding to the first sub-division of that portion permutation to said second sub-division value assigned to the second sub-division permutation corresponding to the second sub-division of that portion permutation. The combined value may then comprise a result of the addition.

When generating, for each portion permutation, a respective label representing that portion permutation, said generating may comprise, when a particular first sub-division value is assigned for a plurality of different first sub-division permutations), generating, for each of said respective plurality of different first sub-division permutations having that particular first sub-division value, a different respective first additional value for use in discriminating between said respective plurality of first sub-division permutations having that particular first sub-division value.

When generating, for each portion permutation, a respective label representing that portion permutation, said generating may comprise, when a particular second sub-division value is to be assigned for a plurality of different second sub-division permutations, generating, for each of said respective plurality of different second sub-division permutations having that particular second sub-division value, a different respective second additional value for use in discriminating between said respective plurality of second sub-division permutations having that particular second sub-division value.

When generating, for each portion permutation, a respective label representing that portion permutation, said generating may comprise, when a first additional value and a second additional

value have been generated for a particular portion permutation: combining said first additional value and said second additional value to produce a combined additional value, wherein the label for that particular portion permutation comprises a representation of the combined value together with the combined additional value for that particular portion permutation.

When generating, for each portion permutation, a respective label representing that portion permutation, said generating may comprise, when one of a first additional value and a second additional value have been generated for a particular portion permutation, generating a label for that particular portion permutation that comprises a representation of the combined value together with that one of a first additional value and a second additional value.

When respectively sub-dividing each portion into a plurality of sub-divisions, said first sub-division may have a different number of bits to said second sub-division.

When generating, for each portion permutation, a respective label representing that portion permutation, each label generated may have a respective label bit length, and the labels are generated such that labels generated for portion permutations which occur a greater number of times within said input sequence of bits may generally have a smaller label bit length than labels generated for portion permutations which occur a lesser number of times within said input sequence of bits.

When generating, for each portion permutation, a respective label representing that portion permutation, each label generated may have a respective label bit length, and the labels are generated such that at least some of the labels may have a label bit length which may be smaller than the processing bit length.

The frequency analysis may comprise, for each one of said plurality of possible first sub-division permutations, determining a respective occurrence level which is the number of times, within said sequence of bits, that a portion occurs comprising that one of said plurality of possible first sub-division permutations. The frequency analysis may comprise may comprise, for each one of said plurality of possible second sub-division permutations, determining a respective occurrence level which is the number of times, within said sequence of bits, a portion occurs comprising that one of said plurality of possible second sub-division permutations.

For a given first sub-division value, the number of first sub-division permutations which are assigned the given first sub-division value may depend on the occurrence levels associated with the first sub-division permutations which are assigned the given first sub-division value; and for a given second sub-division value, the number of second sub-division permutations which are assigned the given second sub-division value may depend on the occurrence levels associated with the second sub-division permutations which are assigned the given second sub-division value.

When assigning, based on said frequency analysis, a first (or second) respective sub-division value to each of said plurality of possible first sub-division permutations, said assigning may comprise: grouping, based on said frequency analysis, said plurality of possible first (or second) sub-division permutations into a plurality of sets (or 'levels'). Each set may comprise at least one first (or second) sub-division permutation. The at least one first (or second) sub-division

permutation in each set may have a corresponding occurrence level that falls within a different respective range of occurrence levels associated with that set.

For a given first (or second) sub-division value, the number of first sub-division permutations which are assigned the given first sub-division value may depend on the set associated with the first (or second) sub-division permutation(s) which are assigned the given first sub-division value.

Forming a processed sequence of bits may further comprise including a header portion in the processed sequence, said header portion comprising extraction information for use in reconstructing said input sequence of bits from said processed sequence, and the extraction information being configured for use in identifying the respective portion permutation which each label represents.

The extraction information may be configured for use in identifying how the said plurality of possible first (or second) sub-division permutations are grouped into sets. The extraction information may identify how many first (or second) sub-division permutations each set comprises. The extraction information may be further configured to identify the processing bit length used in processing said input sequence of bits. The extraction information may be further configured to identify how each portion is sub-divided into a plurality of sub-divisions. The extraction information may be further configured to identify how many bits each first sub-division comprises and how many bits each second sub-division comprises. The extraction information may be further configured to identify how many bits the input sequence of bits comprises.

The process, may further comprise repeating steps (i) to (vii) at least one further time using said processed sequence as said input sequence.

According to one aspect of the invention there is provided a method of processing data, the method comprising the steps of: (i) dividing the data into a plurality of processing segments wherein each processing segment comprises an input sequence of bits; (ii) identifying a current processing bit length for use in processing a current processing segment of said data to form a processed segment meeting at least one predetermined processing criterion; (ii) dividing the current processing segment into a plurality of portions wherein each portion has a respective portion bit length equal to said current processing bit length and wherein the bits in each portion are arranged in a respective one of a number of possible permutations; (iv) assigning a respective label to each of a plurality of said possible permutations; and (v) forming a processed segment by replacing, within said current processing segment, bit portions comprising bits arranged in one of said plurality of possible permutations with the respective label assigned to that one of said possible permutations; (vi) identifying a new processing bit length for use in processing a next processing segment of said data to form a processed segment meeting at least one predetermined processing criterion; (vii) repeating, for each of said plurality of processing segments, steps (ii) to (vi) wherein the new processing bit length is used as the current processing bit length and the next processing segment of said data is used as the current processing segment, and wherein a processing bit length used for at least one of said processing segments of said data is different to a processing bit length used for at least one other of said processing segments of said data.

According to one aspect of the invention there is provided a method of processing data comprising an input sequence of bits, the method comprising the steps of: (i) setting a current

processing bit length, of at least one bit, for use in processing said input sequence of bits; (ii) dividing the input sequence of bits into a plurality of portions wherein each portion has a respective portion bit length equal to said current processing bit length and wherein the bits in each portion are arranged in a respective one of a number of possible permutations; (iii) for each of a plurality of possible permutations analysing the input sequence of bits to respectively identify how many times, within said input sequence of bits, a portion having that possible permutation occurs; (iv) determining whether at least one predetermined processing criterion has been achieved by comparing results of said analysing with the predetermined processing criterion; (v) processing said input sequence of bits based on said determining wherein said processing comprises: when the determining determines that the predetermined processing criterion has not been achieved performing at least one of: setting a new processing bit length that is different to the current processing bit length and repeating steps (ii) to (v) using said new processing bit length as the current processing bit length; and ending processing of said input sequence of bits; and when the determining determines that the at least one predetermined processing criterion has been achieved: assigning a respective label to each of said plurality of possible permutations; and forming a processed sequence of bits by replacing, within said sequence of bits, bit portions comprising bits arranged in one of said plurality of possible permutations with the respective label assigned to that one of said possible permutations.

The predetermined processing criterion may comprise whether 50% of the possible permutations which occur in the input sequence of bits occur at least twice as frequently as the other 50% of the possible permutations which occur in the input sequence of bits.

The predetermined processing criterion may comprise whether 50% of the possible permutations occur in the input sequence of bits.

According to one aspect of the invention there is provided a method of reconstructing a processed sequence of bits produced by a method according to any preceding claim, the method of reconstructing a processed sequence comprising the steps of: obtaining extraction information for use in reconstructing an original sequence of bits from said processed sequence; reconstructing said original sequence of bits from said processed sequence based on said extraction information.

Aspects of the invention extend to computer program products such as computer readable storage media having instructions stored thereon which are operable to program a programmable processor to carry out a method as described in the aspects and possibilities set out above or recited in the claims and/or to program a suitably adapted computer to provide the apparatus recited in any of the claims.

According to another aspect there is provided a method of compression in which an input sequence of bits is divided into a plurality of portions; each portion is sub-divided into a plurality of sub-divisions; frequency analysis is performed to determine the number of occurrences of each sub-division permutation and new values are assigned, based on the frequency analysis, to each of the sub-division permutations. For each portion a label, representing the permutation of bits in that portion, is assigned, wherein the label comprises a representation of a combined value resulting from combining the new values associated with the sub-division permutations of that

portion. A processed sequence of bits is generated by replacing, within the input sequence of bits, bit portions with the respective label representing the permutation of bits in that portion.

Embodiments of the invention will now be described, by way of example only, with reference to the attached figures in which:

Figure 1a is a simplified schematic block diagram illustrating a system for compressing and decompressing data;

Figure 1b is a flow chart illustrating an overview of a method of compression;

Figure 1c is a flow chart illustrating an overview of a method of decompression;

Figure 2 illustrates the main data groups used in the methods of compression described below, including exemplary data sizes/values for the purposes of explanation only;

Figures 3A to 3D illustrate how a bit portion length is selected in a first example;

Figures 4A and 4B illustrate how a bit portion length is selected in a second example;

Figures 5A to 5E illustrate an alternative method of selecting a bit portion length;

Figures 6A to 6D illustrate a method of determining which configuration of combination arrays to use once a bit portion length has been determined according to one or more of the methods of Figures 3A to 3D, 4A and 4B and 5A to 5E;

Figures 7A and 7B illustrate a first part of a method of assigning labels to bit portions once a combination array CA configuration has been selected according to the method illustrated in Figures 6A to 6D;

Figures 8A to 8D are tables detailing possible combined new CA values with their corresponding new CA_0 values and new CA_1 values;

Figure 9 is a table detailing possible combination of CA_0 disambiguation values and CA_1 disambiguation values, and the resulting combined disambiguation values, for the example illustrated in Figures 7A and 7B;

Figure 10 illustrates how labels are assigned to bit portions;

Figure 11 is a table listing all of the possible bit portions of length 6 bits and the labels assigned to each bit portion, based on the combination arrays CA_0 and CA_1 in Figures 7A and 7B;

Figures 12A to 12D are examples of generating new CA values (and disambiguation values) for bit portions having a bit portion length of 8 bits, using a particular CA configuration;

Figures 13A to 13D are simplified representations of four exemplary header structures; and

Figure 14 illustrates the target maximum BP and/or CA values calculated in accordance with an alternative embodiment.

Overview – System

Figure 1a is a simplified schematic block diagram illustrating a system for compressing and decompressing data, and Figures 1b and 1c show related methods. The system of Figure 1a comprises compression apparatus 105 for compressing a file 201 to produce a compressed file 202.

The system of Figure 1a also comprises decompression apparatus 505 for decompressing a compressed file 202, which has been compressed using the compression apparatus 105, in order to re-create the original file 201.

As indicated in Figure 1a the file 201 may comprise, for example, a text document, music data, the contents of a database or video data.

The compression apparatus 105 is configured to extract data comprising a sequence of bits from the file 201, the sequence of bits corresponding to a processing segment 203. The processing segments 203 can be configured, on the fly, to be any suitable size, therefore allowing the processing segment size to be selected adaptively based, for example, on the processing capabilities of the compression apparatus 105 or other relevant factors.

The compression apparatus 105 comprises a bit portion module 253, which is beneficially configured to analyse each of the processing segments 203 and select, based on this analysis, a bit portion length L_{BP} for use in dividing the processing segments into smaller data units referred to as 'bit portions' 205. As an example, Figure 1a illustrates a processing segment having been assigned a bit portion length L_{BP} of 8 bits by the bit portion module 253, however the bit portion module 253 is configured to select a respective bit portion length L_{BP} based on frequency analysis of each processing segment 203, and therefore different processing segments can be assigned different bit portion lengths. Using this frequency analysis, the bit portion module 253 is configured to select the bit portion length L_{BP} based on which bit portion length L_{BP} apparently provides the best (or among the best) prospects for compression. The bit portion module 253 can also be configured to select any bit portion length L_{BP} with acceptable prospects for compression, for example to optimise for speed as opposed to compression.

If the bit portion module 253 determines that no bit portion length will allow compression of the processing segment 203 (or the compression does not meet a predefined compression threshold, for example a greater than 5% reduction in size), it is configured to refrain from assigning a bit portion length to the processing segment, and the processing segment 203 will be output by the compression apparatus 105 in in its original (unprocessed) form.

Once a bit portion length is selected and the processing segment 203 sub-divided into bit portions 205 accordingly, the bit portions 205 may advantageously be further sub-divided into smaller data sub-divisions referred to as combination arrays (although, depending on requirements, such further sub-division may not be implemented). These combination arrays represent the smallest data unit used in processing the processing segment 203.

The way in which a file may be sub-divided into smaller data units to aid efficient processing is described in more detail below, in the section titled 'Overview - Main Data Groups', with reference to Figure 2.

The compression apparatus 105 further comprises a label assignment module 255 which is configured to assign a respective label to each permutation of bits represented by the bit portions 205, based on analysis of the frequency of occurrence of the bit portion value corresponding to that permutation, and/or frequency of occurrence of combination array values that form that permutation, within a processing segment.

The way in which a label for a bit portion permutation may be assigned is introduced below in the section titled 'Overview - Assigning Labels'.

Where a bit portion is sub-divided into combination arrays, the respective combination array values within each bit portion 205 are assigned a new value (or 'label'). The new values assigned to the combination array values within each bit portion are combined together and, if necessary, the resulting combination concatenated with any additional information required for transforming the resulting combination back into its original form. The respective combination for each bit portion 205, together with any information concatenated with that combination, form a bit portion label that is, in effect, assigned to a corresponding permutation bits represented by that bit portion 205. In so doing, each bit portion label is, in effect, also assigned to every bit potion 205 comprising bits arranged in the permutation associated with that label.

The concept of combining different data values is introduced below, in the section titled 'Overview - Combine Method'. The way in which combination array values may be labelled and combined to form a label for a bit portion permutation is described in more detail below in the section titled 'Method of Assigning labels to Bit Portion Permutations using Combination Arrays'.

The label assignment module 255 is configured to output a processed segment 209 corresponding to a processing segment 203 in which each bit portion 205 has been replaced with the bit portion label assigned to the permutation of bits represented by that bit portion 205. In this example, the resulting processed segment 209 is smaller than the processing segment and can thus be thought of as a 'compressed' segment. The processed segment 209 comprises each of the labels assigned to the bit portions 205 of the processing segment 203.

The compression apparatus 105 further comprises a header generation module 257 which is configured to generate a header 211 for each processing segment 203. The header 211 comprises extraction information which is used by the decompression apparatus 505 to extract the processing segment 203 from the processed segment 209. The extraction information preferably allows the decompression apparatus 505 to interpret the labels in the processed segment 209 in order to allow the decompression apparatus 505 to map the labels to their associated bit portion values.

Preferably, each header starts with a compression method signature, and provides information relating to the chosen bit portion length L_{BP}, the combination array configuration used, the size of the original processing segment 203, and information on how labels were assigned to each of the bit portions 205.

As visually indicated in Figure 1a, the total size of each of the processed segments 209 in combination with its header 211 is less than the size of the corresponding processing segment 203. Furthermore, the size of the processed segments 209 and their associated headers 211 may vary.

As shown in Figure 1a, the compression apparatus 105 outputs a compressed file 202, which comprises fewer bits in total than the original file 201. This is due to the fact that the size of each of the processed segments 209 in combination with its header 211 is less than the size of the corresponding processing segment 203.

The decompression apparatus 505 is configured to process each header 211 and associated processed segment 209 of the compressed file 202. Each header can be identified, for example, by the signature included in the header.

The decompression apparatus 505 comprises a header decoding module 557 and a label decoding module 555. The header decoding module 557 is configured to decode the information in the header 211, for use by the label decoding module 555 in decoding the labels in the processed segment 209 and thus map the labels to their associated bit portion values. The label decoding module is configured to output a processing segment 203 comprising all the bit portion values associated with the labels in the processed segment 209. The processing segment 203 therefore corresponds to the original processing segment 203.

The system for compressing and decompressing data illustrated in Figure 1a can alternatively or additionally be used to encrypt and decrypt data. Any file 202 produced by the apparatus 105 will exhibit some level of encryption, because the information contained in the file 202 is represented by different data to that used in the original file 201. In such embodiments where the system of Figure 1a is used to encrypt and/or decrypt data, the total size of each of the processed segments 209 in combination with its header 211 may be greater than the size of the corresponding processing segment 203. Accordingly, when the apparatus 105 is used as encryption/decryption apparatus, the encrypted file 202 output by the encryption side of the apparatus 105 may not always be a 'compressed' file.

Figure 1b is a flow chart illustrating, in overview, a method of compression that may be employed by the compression apparatus 105 of Figure 1a.

In the method of Figure 1a, at step 111 an input sequence of bits is divided into the processing segments. At step 113 the determination is made of whether there is a bit portion length that will allow compression of the processing segment 203 (or the compression does not meet a predefined compression threshold, for example a greater than 5% reduction in size). In other words it is determined whether the potential compression level for the current processing segment is acceptable, for example whether a predetermined processing criterion is satisfied.

If it is determined that the potential compression level for the current processing segment is acceptable, the method continues to step 115 in which the current segment is processed, as described above. Specifically, the possessing segment is analysed and a bit length is selected based on the analysis. The labels are then assigned to each of the bit portions. Extraction information for use in reconstructing the original processing segment is then generated and, in this example, placed in a header. More detail on how the current segment is processed is provided in Figures 2-10 and the associated description.

At step 117, it is determined whether to attempt to reprocess the current segment. If the current segment is to be reprocessed, the processed segment (including the header if present) is used

as the current segment, and the method returns to step 113. If the current segment is not reprocessed, the method continues to step 119 where a processed segment is output.

If at step 113 it is determined that the potential compression level for the current processing segment is not acceptable, the method continues to step 125 in which the current segment is used as the processed segment, without any processing (or further processing) of the current segment. Then, at step 119, the processed segment is output.

After the processed segment is output, it is determined at step 121 whether there is another processing segment in the input sequence of bits for processing. If yes, the next processing segment of the input sequence of bits is used as the current segment, and the method returns to step 113.

If it is determined at step 121 that there are no more processing segments in the input sequence of bits for processing, the processed segments are output together as a processed file at step 123.

Figure 1c is a flow chart illustrating in overview, a method of decompression that may be employed by the decompression apparatus 505 of Figure 1a.

At step 131 the first processed segment of processed file is used as the current segment.

At step 133 it is determined whether extraction information is available for the current segment. In this example, any extraction information is found in the header of the processed segment. If extraction information is available, the method proceeds to step 135 where extraction information is obtained for current segment, for example from an associated header.

Next, at step 137, the processing segment in its form prior to processing is reconstructed from the current segment, based on extraction information.

At step 147, the reconstructed segment is used as the current segment, and the method returns to step 133.

If, at step 133, extraction information is not available, the method proceeds to step 145 where the current segment is used as the reconstructed segment, without any reconstruction (or further reconstruction) of the current segment. Then, at step 139, the processed segment is output.

Next, at step 141, it is determined whether there is another processed segment of the processed file. If yes, the next processed segment of the processed file is used as the current segment, and the method returns to step 133.

If it is determined at step 141 that there are no more processed segments in the processed file, the reconstructed segments are output together as a reconstructed file at step 143.

It will be appreciated that the methods of compressing and decompressing data described herein can beneficially be used in various applications.

For example, compressing data using the methods described herein can allow larger amounts of data to be stored in a given storage medium, and larger amounts of data to be transmitted in any given transmission of data. This in turn will reduce the cost for data storage, which could be

particularly advantageous where large amounts of data need to be stored, such as in data farms. Cost saving can be made because, for example, data farms will require less power to maintain their data storing devices. Advantageously, even if different types of data are being stored (e.g. in a data farm) the methods of compression described allow compression to be achieved for generally any data, regardless of the data type (e.g. audio, text, video).

In the field of telecommunications, the described techniques can be used to compress data before transmission, which would allow a reduction in the amount of resources needed to make transmissions.

Devices can be configured to carry out both compression and decompression of data according to the described methods, or devices can be configured to carry out only one of compression and decompression. Media-playing devices, such as mobile phones and DVD players, may only be configured to decompress compressed media files using the methods described herein. In some cases such media-playing devices may be provided with a dedicated chip for this purpose, or the decompression may be performed by software modules in the device which are not tied to any specific hardware. Providing the processing power of a device is sufficient and enough storage space is available, entire files can be decompressed before use (for example a short video clip can be decompressed and then viewed). In other cases, files can be decompressed on the fly during use (for example a film can be decompressed and watched simultaneously). Considering, mobile phones, storing data in compressed form and then decompressing the data when required using the methods described herein would allow significant amounts of space to be saved on mobile phones, for example allowing multiple high quality films to be stored on the mobile phone memory.

Although the time and/or power taken to compress/decompress a given piece of data can vary, in many instances compression takes significantly longer (and/or requires more processing power) than decompression. In some applications this is not especially limiting, for example where films are compressed at a central internet server, and downloaded or streamed in compressed form and then decompressed at a user device for viewing.

In some cases the time and/or processing power required for compression using the methods described herein can be greater than existing compressions techniques. However, the methods described herein have the advantage that greater compression can be achieved, and additionally or alternatively substantial compression can be achieved more consistently across different types of data when compared to existing data compression techniques. The compression methods described can achieve this because the ability to use different bit lengths and different combination array configuration when processing data means that, in effect, different compression algorithms are applied, not only to different iterations of compression for the same file, but also to different parts of the same file.

As described below, the use of combination arrays allows header sizes to be reduced. This is advantageous because headers 211 are generally added to all compressed segments 209. This contrasts with many existing compression techniques in which files are analysed as a whole, and data for use in decompression, such as a hash table, relates to the file as a whole and is only included once in the compressed file.

The compression methods described herein advantageously analyse each processing segment 203 of a file 201 individually, unlike existing compression methods which analyse a file as a whole. Analysing the processing segments 203 individually (and analysing a processing segment in multiple different ways using bit portions and/or combination arrays) allows the described methods to achieve better and more consistent compression of data.

Overview - Main Data Groups

The way in which a file may be sub-divided into smaller data units to aid efficient processing will now be described, by way of example only with reference to Figure 2.

Figure 2 illustrates the main data groups used in the methods of compression described below, including exemplary data sizes/values for the purposes of explanation only.

A file 201 may comprise, for example, a text document, a music file, a database or a video file. The file 201 may have any size; in this example the file size is 2GB. As a further example, an ultra-high 4K definition DVD is approximately 100GB. A traditional high definition DVD is approximately 6GB. An hour of high definition downloadable video from the internet is approximately 1GB. As an example, using the compression techniques described below, it has been found that any of these types of file can be compressed, typically down to 1/64 of their original size.

In the compression methods described below, the file 201 is divided up into one or more processing segments 203, which are generally smaller in size than the file 201. In this example, the 2GB file 201 is broken up into a plurality of 64KB processing segments 203. Padding bits/bytes may be used to ensure a file 201 can be divided into an integer number of segments 203.

The processing segments 203 can be used where the size of the file 201 is too large for a computer processor to read and/or process the whole file at once. Generally most files fall into this category, however in some cases a whole file 201 may be read and/or processed without being divided into processing segments.

The size of the processing segments 203 is usually fixed and selected based on normal computer processing capabilities; however in some examples the size of processing segments 203 is not fixed (see Modifications and Alternatives section).

The method involves assigning labels to groups of bits in a processing segment 203, where the grouping of bits and corresponding labels are chosen in a way which ensures that the number of bits required to represent the information of the processing segment 203 is less than the original size of the processing segment 203 in bits. In overview, smaller labels (i.e. labels comprising fewer bits) are used to represent more frequently occurring groups of bits, while larger labels (i.e. labels comprising more bits) are used to represent less frequently occurring groups of bits.

In preferred embodiments, two or more main groupings of bits in the processing segment are used: bit portions 205, and combination arrays 207.

As illustrated in Figure 2, each bit portion 205 generally comprises a plurality of consecutive bits, and each combination array 207 generally comprises a sub-group of consecutive bits (or a single bit) from a bit portion 205.

In this example, a 64KB processing segment 203 is divided into a plurality of bit portions 205 each having a bit portion length L_{BP} of 6 bits. As shown in Figure 2, each of the bit portions 205 comprises a permutation of 6 bits, where the first three bit portions have permutations of 011100, 100110 and 111100 respectively. The first bit portion, comprising the bit permutation 011100, is considered to have a bit portion (BP) value of 011100, or 28 in base 10.

Dividing each processing segment 203 up into bit portions 205 provides a way of analysing the characteristics of the processing segment 203, where the results of this analysis are used to determine the prospects for compressing the segment 203 using a particular bit length.

Advantageously, the size of the bit portions 205 is not predetermined, and it can therefore be determined for each processing segment 203 what size of bit portion provides the best prospects for compressing the segment 203.

In this example, the bit portion 205 has a bit portion length L_{BP} of 6 bits, which are sub-divided into three combination arrays 207. The first two combination arrays each comprise a single bit, and the third combination array comprises four consecutive bits. As shown in Figure 2, all bit portions 205 are divided up into combination arrays of the same configuration – in this example the configuration is : [1 bit array][1 bit array][4 bit array]. As also shown in Figure 2, while the configuration (or pattern) of combination arrays 207 is the same for each bit portion 205 of a processing segment 203, the contents of the combination arrays 207 may vary between each bit portion 205, depending on the permutation of bits in each bit portion 205.

As shown in Figure 2, each of the combination arrays comprises permutation of any number of bits (including one bit), where the number of bits in the permutation depends on the combination array (CA) configuration. In Figure 2, the first three combination arrays have permutations of 0, 1 and 1100 respectively. These first three combination array permutations are considered to have combination array (CA) values of 0, 1 and 1100 respectively; or 0, 1 and 12 respectively in base 10.

In some alternative embodiments, processing segments are only divided up into groups of consecutive bits (or single bits) once, without these groups (e.g. bit portions 205) being sub-divided into further groups of consecutive bits or single bits (e.g. combination arrays 207).

Although in this example the bit portion 205 comprises three combination arrays 207, the bit portion can advantageously be divided into any number of combination arrays 207, each combination array 207 having any size. This means that the particular configuration of compression arrays can be selected to provide optimised compression for a particular segment. In this example, where the bit portion length L_{BP} of the bit portions 205 is 6 bits, there are 32 different possible configurations of the combination arrays 207, as set out below:

Configurations for a Combination Array Bit portion length L_{BP} of 6
{1,1,1,1,1,1},{1,1,1,1,2,0},{1,1,1,2,1,0},{1,1,1,3,0,0},{1,1,2,1,1,0},{1,1,2,2,0,0},
{1,1,3,1,0,0},{1,1,4,0,0,0},{1,2,1,1,1,0},{1,2,1,2,0,0},{1,2,2,1,0,0},{1,2,3,0,0,0},

{1,3,1,1,0,0},{1,3,2,0,0,0},{1,4,1,0,0,0},{1,5,0,0,0,0},{2,1,1,1,1,0},{2,1,1,2,0,0},
{2,1,2,1,0,0},{2,1,3,0,0,0},{2,2,1,1,0,0},{2,2,2,0,0,0},{2,3,1,0,0,0},{2,4,0,0,0,0},
{3,1,1,1,0,0},{3,1,2,0,0,0},{3,2,1,0,0,0},{3,3,0,0,0,0},{4,1,1,0,0,0},{4,2,0,0,0,0},
{5,1,0,0,0,0},{6,0,0,0,0,0},

Table 2

In Table 2, each set of six numbers within curly brackets represents a possible configuration of combination arrays 207. Each number represents the size of a combination array in bits, where 0 indicates that no array is used. For example, {1, 1, 3, 1, 0, 0} denotes dividing a bit portion 205 into four combination arrays 207, the first two combination arrays comprising a single bit each, followed by a 3 bit combination array, in turn followed by another single bit array.

It is noted that the total number of different possible configurations of combination arrays depends on the bit portion length, where the number of possible configurations is equal to $2^{L_{BP}-1}$.

As stated above, the configuration of combination arrays is selected to provide the best compression of a segment 203. Generally, all bit portions 205 of a particular processing segment 203 are divided into the same configuration of combination arrays and the combination array configuration exploits any patterns, repetition and/or redundancy in the processing segment 203 in order to achieve effective compression.

Overview - Combine Method

The concept of combining different data values will now be introduced and explained, by way of example only.

A byte can hold a value between 0 (00000000) and 255 (11111111). The ASCII standard provides for representation of characters, letters or symbols where each character, letter or symbol is represented using an ASCII code which has a value of between 0 and 255. As a result, each letter, character or symbol requires one byte of information to be represented, as Table 1, below, illustrates.

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011

l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Table 1

Considering, for example, the letters J and o, these have ASCII codes of 74 (01001010) and 111 (01101111) respectively. Therefore, a conventional representation of the name Jo would be 0100101001101111, which is 16 bits long.

The number of bits required to represent the name can be decreased by combining the respective ASCII values using at least one mathematical operation. For example, the two values can be added together:

74+111=185

Advantageously, the number 185 can be represented in binary using only 8 bits (10111001), therefore saving 8 bits on the 16 bit value of 0100101001101111.

However, the letters J and o are <u>not</u> the only combination of letters which would sum to give the total 185. For example, the letters l and p would also yield the total 185 when added together. This is referred to as a collision.

Therefore, in this example it is necessary to provide additional disambiguation information in order to indicate which of the potential combinations of ASCII characters is being represented.

The number of collisions (i.e. combinations resulting in the same total when combined using a mathematical operation such as addition) can be decreased by changing the numeric value used to represent the characters being combined.

For example, the first ASCII character value can be multiplied by 10 before the two values are combined. Taking the example of “Jo” again:

74×10+111 = 740+111 = 851

The number 851 can be represented in binary using only 10 bits (1101010011), therefore saving 6 bits on the 16 bit value of 0100101001101111.

In this example, it is also necessary to provide additional disambiguation information in order to indicate which of the potential combinations of ASCII characters is being represented.

However, multiplying the first ASCII character value by 10 before the two values are added has the effect of reducing the number of combinations yielding the same result ("*collisions*"). This means that less additional disambiguation information is required.

Collisions when combining bytes can be reduced still further by replacing the ASCII values used to represent characters with numeric labels. Labels can also reduce the number of bits used to represent the combined value. For example, if the letters J and O are represented by the labels 0 and 1 respectively, then combining the two labels using addition results in a combined value of 1. As long as no other characters are assigned the labels 0 or 1, the combined value of 1 will be unique, with no collisions occurring. Moreover, in this example the combined value can be represented using only 1 bit.

Although described with reference to ASCII characters for ease of understanding, the above-described methods of combining data can be applied to any data, comprising any number of bits.

The methods described herein allow data, such as a file, to be compressed by dividing the data into groups of bits, assigning labels to the groups of bits and then "combining" two or more of these groups of data together by combining their respective labels. In some embodiments, the combining comprises a mathematical operation such as addition.

In an e-book that uses letters and numbers (see Table 1), it is possible that either the first bit or the last bit is only ever 0 and the 1 is never used, or vice versa, depending on encoding.

Advantageously, in preferred embodiments the way in which a file is divided into groups of bits can be chosen in order to provide improved compression of the file. For example, when one part of the file is being processed it may be divided up in a different way to another part of the file.

Also, the preferred embodiments allow data from different types of media, and by extension having vary different characteristics, to be compressed effectively, due to the flexibility when dividing the data into groups of bits and assigning labels to the groups of bits. Existing compression techniques tend to be more effective in compressing particular types of media data (e.g. text, image data or the like) because they are better optimised for the inherent characteristics of that data. Advantageously, the preferred embodiments can achieve compression of files and/or data which would ordinarily be difficult to compress using such existing compression techniques.

Overview - Assigning Labels

The bits of the processing segment are analysed to determine a way of dividing the processing segment into groups of bits which will allow compression to be achieved when labels are assigned to the groups of bits. The processing segment is then divided into groups of bits according to the determined configuration. The groups of bits may comprise bit portions and/or combination arrays as introduced above.

Next, a label is assigned to each of the groups of bits, wherein each label is unique (although generally only unique for the processing segment being processed; labels may be reused

between processing segments). Some or all of the labels may comprise multiple parts. Preferably, all labels comprise a first part which acts as a primary identifier of the bit portion value (later referred to as "Combined new CA value").

The first part of the bit label may uniquely (i.e. unambiguously) identify a bit portion value, in which case the label need only comprise the first part. However, when the first part of the label does not unambiguously identify the bit portion value (i.e. multiple different bit portion values are associated with the same first part of the label), the label further comprises a second part (later referred to as "Combined disambiguation information DI").

The purpose of the second part of the label is to identify which of the multiple different bit portion values associated with the first part is being represented by the label.

In order to illustrate this with an example, consider the following four different bit portion values:

01011, 10110, 10111, 10010

Each of these four different bit portion values may be associated with the same first part of a label (e.g. 11):

01011, 10110, 10111, 10010
↓ ↓ ↓ ↓
11 11 11 11

In such a case, each bit portion value can be unambiguously identified using one of four second parts of the label (e.g. 00, 01, 10, 11):

01011, 10110, 10111, 10010
↓ ↓ ↓ ↓
11 11 11 11
00 01 10 11

In the examples provided here, the complete label for the bit portion values would be as follows:

01011, 10110, 10111, 10010
↓ ↓ ↓ ↓
1100 1101 1110 1111

Preferably, the length of the first part in bits remains constant for all bit portion values in a processing segment 203, while the length of the second part can vary, or the second part may not be used at all to identify some bit portion values.

It can therefore be seen that the label as a whole can vary in length of bits. All the labels used for the bit portion values of a particular processing segment can vary in length but share a common minimum length, corresponding to the length of the first part of the label. However, between different processing segments the length of the first part of the label can vary, as it is assigned based on frequency analysis of the processing segment (as described in further detail below).

Method of Selecting Bit portion length

Figures 3A to 3D illustrate a method of selecting what bit portion length L_{BP} should be used when dividing a processing segment 203 into a number of bit portions 205.

This is done by dividing the processing segment 203 up into bit portions 205 of different bit portion lengths L_{BP}, and performing frequency analysis for each of the different bit portion lengths used.

Some existing compression techniques use fixed bit portion lengths. It has been found that by using variable bit portion lengths, which can change depending on which part of a file is being processed, additional compression can be obtained which would otherwise not have been achievable.

Figures 3A and 4A provide overviews of the frequency analysis results obtained for bit portion lengths 2 to 4 and 3 to 7 respectively, with different exemplary results. Figures 3B, 3C, 3D and 4B illustrate frequency analysis performed on bit portion lengths of 2, 3, 4 and 7 respectively.

As shown in Figure 3A, in this example a bit portion length of 2 bits is tested first. The processing segment 203 is divided up into a plurality of bit portions 205, each having a bit portion length of 2 bits. As shown in Figure 3A, frequency analysis is performed on the bit portions 205 of this initial bit portion length $L_{BP} = 2$, and it is determined whether at least one of two criteria are fulfilled.

The first criterion is whether two or more compression "levels" (levels are described further below) are present within the analysed bit portion BP values, and the second criterion is whether 50% or fewer of the possible bit values are present in the processing segment 203.

If neither of the criteria are fulfilled, the bit portion length is incremented by one bit - to 3 bits - and the processing segment 203 is re-divided up into a plurality of bit portions 205, this time each having a bit portion length of 3 bits. For each bit portion length being tested, if the frequency analysis results fail to fulfil either of the two criteria, the next bit portion length is tested (i.e. the bit portion length is incremented by one bit and the processing segment 203 is re-divided up into a plurality of bit portions 205, each having the same number of bits as the current bit portion length).

Figure 3B illustrates the frequency analysis performed on the plurality of bit portions 205, in this case each having a bit portion length of 2 bits. As each bit portion 205 of the processing segment 203 is only made up of 2 bits, a bit portion 205 can only have one of four values – 00, 01, 10 or 11. Once the processing segment 203 has been divided into the plurality of bit portions 205, the number of occurrences of each possible bit portion value is determined (i.e. the frequency of each value).

The bit portion values are then sorted in order of most occurring to least occurring, as shown in Figure 3B. In this example, the bit portion value 01 occurs the greatest number of times, with 65,538 occurrences and the bit portion value 00 occurs the least number of times, with 65,533 occurrences.

The number of compression levels is then determined based on the number of occurrences of each of the bit portions values.

The level in which a bit portion (BP) value is placed determines how many bits the label assigned to the BP value will have. All BP values in the same level will be assigned the same number of bits. In preferred embodiments, the 1st level (level 0) is allocated labels with the minimum possible number of bits. Furthermore, in preferred embodiments, the labels allocated to each successive level are one bit longer than the previous level. An exemplary set of labels and associated labels are shown in Table 3 below.

Level	Label
0	00
0	01
1	100
1	101
2	1100
2	1101
2	1110
2	1111

Table 3

In preferred embodiments, a "level" is defined as being a group of bit portion values in which none of the bit portion values occur less than half as frequently as the most occurring bit portion value in that group. For example, in a group of bit portion values where the most occurring bit portion value occurs 28,000 times, all of the bit portion values in the group will have occurrences greater than 14,000. In the example shown in Figure 3B, the least occurring bit portion value occurs 65,533 times, and therefore all of the bit portion values are considered to occupy the same level. Bit portion length $L_{BP} = 2$ therefore fails to satisfy the first criterion.

Next, it is determined whether 50% of the possible bit portion values occur in the processing segment. For example, if only the bit portion values 01 and 11 occurred in the processing segment 203, and bit portion values 10 and 00 both never occurred, then exactly 50% of the possible bit portion values are present in the processing segment. This would be an indication that the processing segment 203 can be compressed using the selected bit portion length. However, in the example illustrated in Figure 3B all four of the possible bit portion values are present in the processing segment and therefore 100% of the possible bit portion values are present. As can be seen in Figure 3A, the bit portion length of 2 bits is listed as having one compression level and as not satisfying the requirement that 50% or fewer of the possible bit portion values are present. Bit portion length $L_{BP} = 2$ therefore fails to satisfy the second criterion.

Therefore, the processing segment 203 is divided into a plurality of bit portions each having a bit portion length of 3 bits instead of 2 bits and frequency analysis is again performed. This is illustrated in Figure 3C. Figure 3C shows that if bit portion length $L_{BP} = 3$ there are 8 possible bit portion values.

The bit portion values are then sorted in order of most occurring to least occurring, as shown in Figure 3C. In this example, the bit portion value 011 occurs the greatest number of times, with 21,851 occurrences and the bit portion value 101 occurs the least number of times, with 21,833 occurrences.

Bit portion length $L_{BP} = 3$ therefore fails to satisfy the first criterion.

Furthermore, in the example illustrated in Figure 3C all eight (i.e. 100%) of the possible bit portion values are present in the processing segment. Bit portion length L_{BP} = 2 therefore fails to satisfy the second criterion.

Next, the processing segment 203 will be divided into a plurality of bit portions 205 having a bit portion length of 4 bits. This is illustrated in Figure 3D.

Figure 3D shows that if bit portion length L_{BP} = 4 there are 16 possible bit portion values.

As shown in Figure 3D, bit portion values are sorted in order of most occurring to least occurring. In this example, the bit portion value 0001 occurs the greatest number of times, with 27,369 occurrences and the bit portion value 1110 occurs the least number of times, with 1,962 occurrences.

Therefore, unlike for bit portion lengths 2 and 3 described above, multiple compression levels are present within the analysed bit portion BP values. Specifically, the 4th BP value (1001) occurs 12,646 times, which is less than half of 27,369. Therefore, the 4th bit portion value belongs to a 2nd level (level 1).

Furthermore, the 8th BP value (1000) occurs 4,146 times, which is less than half of 12,646. Therefore, the 4th bit portion value belongs to a 3rd level (level 2).

This means that three levels are present, and bit portion length L_{BP} = 4 therefore satisfies the first criterion.

As a result, bit portion length L_{BP} = 4 would be selected as the chosen bit portion length in this example.

In the exemplary method of Figure 4A, the processing segment 203 is initially divided up into a plurality of bit portions 205 each having a bit portion length of 3 bits (rather than 2 bits as illustrated in Figure 3A). As the exemplary results of Figure 4A, none of bit portion lengths 3 to 6 satisfy either of the criteria.

Figure 4B shows exemplary frequency analysis results for bit portion length L_{BP} = 7. If bit portion length L_{BP} = 7, there are 128 possible bit portion values (some are omitted for legibility).

As shown in Figure 4B, bit portion values are sorted in order of most occurring to least occurring. In this example, all bit portion values from the 10th value onwards have an occurrence of 0, and therefore bit portion length L_{BP} = 7 satisfies the second criterion. BP values with an occurrence of 0 are not assigned to a level, and therefore the total number of levels present for L_{BP} = 7 is 1 (the first criterion is therefore not fulfilled).

As a result, bit portion length L_{BP} = 7 would be selected as the chosen bit portion length in this example.

It is noted that in the particular example illustrated in Figure 4B, it is possible to achieve improved compression by assigning levels according to alternative embodiments, such as those described below.

Alternative method of Selecting Bit portion length

Figures 5A to 5E illustrate an advantageous alternative method of selecting a bit portion length L_{BP}. The method involves testing multiple bit portion lengths and determining if compression of the processing segment can be achieved using the bit portion length being tested, and if so how much compression can be achieved.

The determination is made by assigning labels to each of the possible bit portion (BP) values, and then determining whether the processing segment 203 can be represented using fewer bits if the bit portions are represented using their respective labels (i.e. determining whether the processing segment 203 can be compressed using the labels). In order to assign the labels and make the determination as to whether compression can be achieved, frequency analysis is performed on the bit portion values to determine how many times each possible bit portion value occurs within the processing segment 203.

The frequency analysis results in a value for the achievable compression of the processing segment 203 for each bit portion length tested (i.e. the minimum compression that is known to be achievable for the processing segment based on the chosen bit portion length). In Figure 5A, bit portion lengths from 2 bits to 8 bits are tested, with achievable compressions ranging from 3% (2 bits) to 25% (6 bits). It is noted that the final compression achieved for the selected bit portion length, once the full compression method described below has been carried out, may be higher than the achievable compression value.

It can also been seen from Figure 5A that the bit portion length having the highest potential compression is 6 bits, whereas a bit portion length of 8 bits would, for this particular segment being processed, have a lower potential compression. Therefore, in this case any compression techniques which divide the processing segment into bytes (i.e. 8 bits) would fail to exploit potential additional compression.

As illustrated by the exemplary bit values in Figure 5A, the same processing segment 203 comprising the same bits may be analysed multiple times, being divided into bit portions 205 of different sizes each time.

As shown in Figure 5A, frequency analysis using different bit portion lengths is performed on a processing segment, in this example of size 64KB (only the first 16 bits and the final bit of the segment are shown for simplicity).

Figure 5B and 5C illustrate the frequency analysis performed on the processing segment 203 when divided up into a plurality of bit portions 205, each having a bit portion length of 4 bits. As each bit portion 205 of the processing segment 203 is made up of 4 bits, a bit portion 205 can have one of sixteen values – from 0 (0000) to 15 (1111).

Once the processing segment 203 has been divided into the plurality of bit portions 205, the number of occurrences of each possible bit portion value is determined (i.e. the frequency of each value). The bit portion values are then sorted in order of most occurring to least occurring, as shown in Figure 5B. In this example, the bit portion value 0001 occurs the greatest number of times, and the bit portion value 1110 occurs the least number of times.

The default order of bit portion values is from smallest to largest, and therefore when two bit portion values have the same number of occurrences within a processing segment (which may

be, for example, zero), the bit values are not sorted and accordingly will remain in size order. As shown in Figure 5B, each of the sorted bit portion values is associated with a ranking corresponding to their sorted position. As can be seen, the most occurring bit portion value is ranked 0 and the least occurring bit portion value is ranked 15.

In some embodiments, the sorted bit portion values are assigned new values which correspond to their ranking, with value 0000 corresponding to ranking 0, and value 1111 corresponding to value 15.

Referring to Figures 5B and 5C, in some embodiments the sorted bit portion values are not renumbered with new values, for example when fewer than 50% of the BP values occur in the processing segment being analysed.

The occurrences of the bit portions are then analysed in order to split the BP values into levels where possible. As explained above, a "level" is defined as being a group of bit portion values in which none of the bit portion values occur less than half as frequently as the most occurring bit portion value in that group. For example, in a group of bit portion values where the most occurring bit portion value occurs 28,000 times, all of the bit portion values in the group will have occurrences greater than 14,000.

In the example shown in Figure 5B, it is determined that the BP values can be grouped to create three levels. These levels are referred to as occurrence-based levels. As can be seen, in Level 0 the highest occurring bit portion value has 27369 occurrences; in Level 1 the highest occurring bit portion value has 12646 occurrences; and in Level 2 the highest occurring bit portion value has 3923 occurrences.

In some alternative embodiments, the levels can be defined using different methods. For example, the occurrences of the BP values may be analysed in order to determine whether the occurrences can be divided into two or more groups in which the total number of occurrences of one group (i.e. all occurrence counts in the group summed) of one group is less than or equal to half the total number of occurrences of another group.

If there are only two levels in a bit portion, compression cannot be achieved unless the bit portion is broken up into two or more combination arrays (see below for description of how bit portions are broken up into combination arrays). For example, if a bit portion length of 4 is used, and two levels are present within the bit portion values, the bit portion can then be broken into two combination arrays. It has been found that one combination array may have one level in its CA values, while the other CA may have three levels in its CA values (this becomes more likely the longer the bit portion length being used).

Once each of the bit portion values has been assigned to an occurrence-based level, each of the bit portion values can be assigned an initial label 403. However, in some preferred embodiments the BP values are first re-grouped into optimised levels before the initial labels 403 are assigned. This re-grouping of the BP values into optimised levels is illustrated in Figure 5C.

The initial labels are assigned to bit portion values to determine an achievable compression ratio for the processing segment 203, and whether compression can be achieved at all. They are

referred to as "initial labels" because the actual labels assigned to bit portions may be different once the full compression method as described below is carried out.

As can be seen in Figure 5B, the initial labels 403 have varied lengths, but in general bit portion values with a high frequency of occurrences are assigned a short initial label (e.g. 3 bits long) and bit portion values with a low frequency of occurrence are assigned a longer initial label (e.g. 5 bits long).

As can also be seen from Figure 5B, the initial labels 403 can comprise one or two parts: all initial labels 403 comprise a new bit portion (BP) value part; while some initial labels 403 additionally comprise a disambiguation part.

The new values act as primary identifiers of the bit portion values, and all new BP values assigned have the same length in bits – in the example shown in Figure 5B, all new BP values are three bits long. The size in bits of the new BP values is determined by the maximum new BP value. In this case the maximum new BP value is 7, which is represented in binary as 111, and as a result all new BP values comprise three bits. However, if the maximum new BP value was 8, this would be represented in binary as 1000, and as a results all new BP values would comprise 4 bits.

However, new BP values do not unambiguously identify an associated bit portion value in all cases because in some cases the same new BP value is assigned to multiple BP values. In such cases, a disambiguation value is used to identify a particular one of the multiple bit portion values associated with the same new BP value.

In order to ensure that the most frequently occurring bit portion values are assigned the shortest initial labels, the bit portion values in the first level (Level 0) are each assigned unique new values, as can be seen in Figure 5B. No disambiguation values are therefore used, and the initial label assigned to the bit portion values of level 0 only comprises the new value part.

When assigning new bit portion values to the bit portion values in level 1 onwards, the same new BP values can be assigned to multiple BP values. Where this re-use of new BP values occurs, the number of disambiguation values which are needed corresponds to the number of bit portion values which have been assigned the same new bit portion value.

For example, if four bit portion values have been assigned the same new bit portion value, four disambiguation values are required in order to unambiguously identify a particular bit portion value. This means that each disambiguation value will comprise two bits. It will be appreciated that, in general, the higher the number of BP values which are assigned the same new BP value, the larger the disambiguation value which is assigned to each BP value.

To achieve compression, bit portion values with a high frequency of occurrences should generally be assigned a short initial label and bit portion values with a with a low frequency of occurrence should generally be assigned a longer initial label. Since the new BP values comprise the same number of bits for all possible BP values, it is the disambiguation which principally affects the size of the initial label 403.

As a general rule, the lower the level (where Level 0 is the lowest), the fewer BP values are assigned the same new BP value. In this embodiment, the maximum number of repetitions of a

new bit portion value is set to be 2^{Lev}, where Lev is the level of the bit portion values being assigned new values. For example, in level 2, the same new bit portion value can be assigned to up to 4 bit portion values.

A more general example of new BP value repetition is shown in Table 4, below.

BP Level	New BP Value
Level 0	z_0
Level 0	z_1
Level 0	z_2
Level 0	z_3
Level 0	z_4
Level 1	z_5
Level 1	z_5
Level 1	z_6
Level 1	z_6
Level 2	z_7
Level 2	z_7
Level 2	z_7
Level 2	z_7

Table 4

As shown in Table 4, each new BP value is repeated 2^{Lev} times. In level 0, new BP values are repeated $2^0 = 1$ times each. In level 1, new BP values are repeated $2^1 = 2$ times each. In level 2, new BP values are repeated $2^2 = 4$ times each.

In Figure 5B, level 2 comprises 9 BP values. In this level new BP values can be assigned to up to four original BP values. Therefore, the four most-occurring BP values are assigned the new BP value 5, the next four most-occurring BP values are assigned the new BP value 6, and the remaining BP value in Level 2 is assigned the new BP value 7.

In such a situation, as can be seen from Figure 5B the new BP value 7 is unique, and therefore the least-occurring BP value in Level 2 is not assigned a disambiguation value. This means that the least-occurring BP value in Level 2 has an initial label of only 3 bits, while the rest of the (more-occurring) BP values in level 2 have initial labels of 5 bits. This is not optimum for compression, and therefore a method of level optimisation is used to move BP values between levels, as illustrated in Figure 5C.

Nevertheless, even without any level optimisation having been performed, it can be seen from Figure 5B that compression can be achieved. The size in bits of each occurrence-based label is shown in Figure 5B, and from this the number of bits used to represent the BP values in each level can be determined. This is given by the total number of occurrences for a level multiplied by the occurrence-based label size.

The total number of bits used to represent all of the BP values in the bit portion 203 can then be determined by summing the number of bits used for each level. As shown in Figure 5B, this is equal to 483555, which is less than the total number of bits in the processing segment (524288). Accordingly, assuming a header size of 121 bits, a 7.7% compression is possible. In some embodiments, the bit portion length may be selected based on this possible compression measure, without any optimisation of the levels (since compression is achieved without optimisation in some cases).

Figure 5C illustrates how the bit portion length is selected according to preferred embodiments, where levels are optimised before the potential compression is determined.

In Figure 5C, BP values are first re-grouped into optimised levels before the initial labels 403 are assigned. The occurrence-based levels determined in Figure 5B are indicated on Figure 5C using dashed braces. It can therefore be seen that the optimised levels are generally different to the occurrence based levels.

A general aim of level optimisation is to ensure that the number N_{Lev}^{BP} of BP values in each level is divisible by 2^{Lev} without remainder, where Lev is the level. This ensures efficient use of the assigned new BP values.

This can be represented mathematically as:

$$N_{Lev}^{BP} \bmod 2^{Lev} = 0 \qquad \text{Equation 1}$$

For example, as shown in Figure 5B, Level 2 includes 9 BP values, so $N_{Lev}^{BP} = 9$, and for Level 2, Lev = 2, therefore the number N_{Lev}^{BP} of BP values in the level is not divisible by 2^{Lev} without a remainder.

Specifically:

$$N_{Lev}^{BP} \bmod 2^{Lev} = 9 \bmod 2^2 = 9 \bmod 4 = 1$$

The result of $N_{Lev}^{BP} \bmod 2^{Lev}$ can be used to indicate how many BP values should be moved out of the level and into a different level. In this example, one BP value should be moved out of level 2.

In some examples, the condition $N_{Lev}^{BP} \bmod 2^{Lev} = 0$ is satisfied by moving the highest-occurring BP values in the level from level Lev to level Lev -1. In the present example, the most-occurring BP value, 1000, is moved from level 2 to level 1.

It will be appreciated that in other examples, the condition $N_{Lev}^{BP} \bmod 2^{Lev} = 0$ may be satisfied by adding additional BP values to the level (e.g. the lowest-occurring BP values from level Lev -1 are moved to level Lev).

In this way, the levels are optimised such that the number of BP values in each level is a multiple of 2^{Lev} or equal to 2^{Lev}, satisfying $N_{Lev}^{BP} \bmod 2^{Lev} = 0$.

This process of determining whether the number N_{Lev}^{BP} of BP values in a level is divisible by 2^{Lev} is repeated for each level, from the highest level to level 0.

It is noted that for level 0, $N_{Lev}^{BP} \; mod \; 2^{Lev}$ will always equal 0, because 2^0 is equal to 1.

Therefore, the condition $N_{Lev}^{BP} \; mod \; 2^{Lev} = 0$ is always fulfilled for level 0, regardless of how many bit portion values are present in level 0.

Preferred further conditions for optimising bit portion levels are described below.

An initial label 403 is assigned to each BP value based on its level, in a similar way to that shown in Figure 5B.

The size in bits of each optimised initial label 403 is shown in Figure 5C, and from this the number of bits used to represent the BP values in each level can be determined. This is given by the total number of occurrences for a level multiplied by the optimised initial label size.

As can be seen from Figure 5B, the total number of bits used to represent the bit portions 205 of the processing segment 203 is 483555 when labels are assigned to bit portions 205 based on occurrences, without any optimisation of the levels. In contrast, as can be seen from Figure 5C, the total number of bits used to represent the bit portions 205 of the processing segment 203 is 470687 when labels are assigned to bit portions 205 using optimised levels. This demonstrates that optimising levels results in a higher achievable compression.

Figures 5D and 5E illustrate how the achievable percentage compression of the processing segment is determined, based on a bit portion length L_{BP} of 4 bits and the frequency analysis shown in Figures 5B and 5C.

Figure 5D is a table which summarises the total possible bits used in the header 211 which is assigned to the compressed portion 209. As shown in Figure 5D, this calculation is based on the header 211 comprising a signature, and information on the bit portion length, combination array configuration, and two types of label assignment information – "level counts" and "CA value information". A minimum and maximum size of each of these parts is determined, and summed in order to provide minimum and maximum total sizes of the header 211.

Figure 5E is a table which shows the calculation of the achievable compression of the processing segment 203 as a percentage of its original size. The maximum header size is used in this calculation in order to ensure that the percentage compression is achievable.

As shown in Figure 5E, the determined achievable compression for the processing segment based on a bit portion length L_{BP} of 4 bits is 10.20%.

Preferred conditions for optimising bit portion levels

In preferred embodiments, in addition to the condition defined by equation 1, level optimisation is based on the following further conditions.

Firstly, the number of levels in a bit portion should not exceed the bit portion length:

$$N_{BP}^{LevelsMAX} = L_{BP} \qquad \text{Equation 2}$$

Secondly, the maximum new bit portion value should equal a target maximum new bit portion value

$$MaxNewBPVal = TargetMaxNewBPVal \qquad \text{Equation 3}$$

Where the target maximum new bit portion value assigned to one or more bit portion values in a processing segment is defined as follows:

$$TargetMaxNewBPVal = 2^{\lfloor log_2(N_{BP}^{Levels}) \rfloor + 1} - 1 \qquad \text{Equation 4}$$

And where the maximum new bit portion value is defined as follows:

$$MaxNewBPVal = \left(\sum_{Lev=0}^{Lev=N_{BP}^{Levels}-1} \frac{N_{Lev}^{BP}}{2^{Lev}} \right) - 1 \qquad \text{Equation 5}$$

L_{BP} is the bit portion length in bits;

N_{BP}^{Levels} is the number of levels into which the bit portions values of a bit portion 205 are divided;

$N_{BP}^{LevelsMAX}$ is the maximum number of levels into which the bit portion values of a bit portion 205 can be divided;

Lev is the level index, for example $Lev = 0$ for level 0 and $Lev = 1$ for level 1;

$MaxNewBPVal$ is the maximum new bit portion value assigned to one or more bit portion values in a processing segment

$TargetMaxNewBPVal$ is the target maximum new bit portion value assigned to one or more bit portion values in a processing segment;

N_{Lev}^{BP} is the number of bit portion values in a level;

Splitting the analysed BP values into more levels, while still fulfilling the conditions above, typically results in a smaller maximum new value and therefore smaller initial labels 403 being assigned to each of the BP values. This allows greater compression to be achieved.

Method of selecting configuration of combination arrays

Figures 6A to 6D illustrate a method of determining which configuration of combination arrays 207 to use once a bit portion length L_{BP} has been determined according to one or more of the methods described above. The method involves dividing the bit portions 205 into combination arrays 207 according to different configurations and performing frequency analysis on the combination arrays, in order to determine which configuration of combination arrays 207 has the best prospects for compressing the processing segment 203.

In Figures 6A to 6D an exemplary bit portion length L_{BP} of 6 bits is used. As illustrated in Table 2, above, a bit portion 205 having a bit portion length L_{BP} of 6 can be divided up into combination arrays using 32 different configurations. Figure 6A provides a visual overview of how each bit portion 205 of a processing segment 203 is divided into combination arrays 207 according to the first 8 combination array (CA) configurations, the 29th CA configuration and the final (32nd) CA configuration.

As shown in Figure 6A, each of the possible CA configurations is assigned a reference number, in this example starting at 0 for the CA configuration [1, 1, 1, 1, 1, 1] and continuing to 31 for the CA configuration [6, 0, 0, 0, 0, 0].

Frequency analysis is performed on each of the combination array CA configurations, and it is determined whether at least one of two criteria is fulfilled. The first criterion is whether the total number of levels is greater than or equal to twice the number of arrays. The second criterion is whether, for any of the combination arrays of a CA configuration, 50% or fewer of the possible combination array values occur in the processing segment 203. These criteria are explained in further detail below with reference to Figures 6B and 6C.

For the purpose of explanation, the combination array configuration [3, 3, 0, 0, 0, 0] (reference number 28) will be considered.

The configuration [3, 3, 0, 0, 0, 0] dictates that each bit portion 205 is divided into two arrays, each comprising 3 bits.

As indicated in Figure 6A, the first array is denoted CA_0, and the second array is denoted CA_1.

It will be appreciated that as CA_0 and CA_1 are each 3 bits long, each can have any of 8 different combination array values (CA values), as set out in Table 5, below.

Possible CA_0 values (L_{CA0} = 3)	Possible CA_1 values (L_{CA1} = 3)
000	000
001	001
010	010
011	011
100	100
101	101
110	110
111	111

Table 5

For each CA configuration (such as number 28 presently being considered), all equivalent combination arrays in the processing segment 203 are analysed collectively. For example, all the CA_0 arrays defined by CA configuration 28 are analysed to determine their values. The frequency of occurrence of each possible CA_0 value is determined, from which CA values can be assigned to levels. This is illustrated in Figure 6B. The same analysis is done on all CA_1 arrays, as shown in Figure 6C.

Considering Figure 6B in more detail, the number of occurrences of each possible CA value is determined (i.e. the frequency of occurrence of each CA value within the segment). As shown, the most occurring value 010 occurs 30,000 times in the processing segment 203 and the least occurring value 100 occurs 3,981 times in the processing segment 203.

The CA_0 values are sorted in order of most occurring to least occurring (as long as more than 50% of the CA_1 values have an occurrence greater than 0 within the processing segment). The default order of CA values is from smallest to largest, and therefore when two CA values have the same number of occurrences within a processing segment (which may be, for example, zero), the CA values are not sorted and accordingly will remain in size order.

The number of compression levels is then determined based on the number of occurrences of each of the CA_0 values. In preferred embodiments, a "level" is defined as being a group of bit portion values in which none of the bit portion values occur less than half as frequently as the most occurring bit portion value in that group. In the example shown in Figure 6B, the most occurring CA_0 value occurs 30,000 times, and the second most occurring CA_0 value occurs 20,000 times which is more than half of 30,000 and therefore both of the most occurring values are assigned to the same level (Level 0).

The third most occurring CA_0 value, 001, occurs 9,000 times within the processing segment 203. Since 9,000 is less than half of 30,000, the third most occurring CA_0 value 001 is assigned to a second level – Level 1.

As 9,000 is the highest occurring value in Level 1, any CA_0 values with an occurrence of less than 4,500 will be assigned to a different level. As shown in Figure 6B, the sixth most occurring CA_0 value, 110, has an occurrence of 4,400, and therefore it is assigned to a third level – Level 2. No CA_0 values have an occurrence of less than 2,200, and therefore CA_0 has three levels in total.

Next, considering Figure 6C in more detail, the second combination array CA_1 is analysed in the same way as for CA_0 in Figure 4B. The frequency of occurrence of each possible CA_1 value is determined. As shown, the most occurring value 011 occurs 19,000 times in the processing segment 203 and the least occurring value 101 occurs 9,000 times in the processing segment 203.

In a similar way as performed for CA_0, the CA_1 values are sorted in order of most occurring to least occurring (as long as more than 50% of the CA_1 values have an occurrence greater than 0 within the processing segment).

The number of compression levels is then determined based on the number of occurrences of each of the CA_0 values.

Using this technique for defining levels, it is found that the total number of levels for CA_1 is two levels.

The total number of levels for the CA configuration 28 [3, 3, 0, 0, 0, 0] is therefore 5 levels (3 levels for CA_0 + 2 levels for CA_1).

Turning back to Figure 6A, the total number of levels for CA configuration 28 can be seen in the "total no. of levels" column. The "2x number of arrays" column indicates 4 for CA configuration 28 (as there are two combination arrays), and therefore the first criterion is fulfilled - the total number of levels is greater than twice the number of arrays.

It can be seen from Figures 6B and 6C that all the possible CA_0 values and all the possible CA_1 values occur in the processing segment 203, and therefore the second criterion is not fulfilled – for both of the combination arrays, more that 50% (in fact 100%) of the possible combination array values occur in the processing segment 203.

Figure 6D illustrates that CA configuration 28 is treated at the chosen configuration, based on the analysis performed in Figures 6B and 6C. The chosen CA configuration is then used to compress the whole processing segment 203, by assigning labels to each of the bit portions 205 in the processing segment 203, where the labels are generated by splitting the bit portions 205 up into combination arrays 207 in accordance with CA configuration 28. This method of compressing the processing segment 203 is explained further below.

After the processing segment 203 is compressed using the chosen CA configuration, it is checked whether the compression has been successful (e.g. whether any compression has been achieved, or whether the compression is greater than a predefined threshold). If it is determined that compression has not been successful, the method will return to analysing CA configurations as shown in Figure 6A, and a new CA configuration is chosen for use in compressing the processing segment 203.

If none of the possible CA configurations fulfil either of the two criteria, then the processing segment 203 is not compressed and is output by the compression apparatus 105 in its original form.

In some alternative embodiments, if none of the possible CA configurations fulfil either of the two criteria, a new bit portion length is selected using one or more of the methods described above. A new CA configuration can then be chosen based on the two criteria for selecting CA configurations. In such cases, it is preferable to set a processing time limit for attempting to compress a single processing segment, where expiry of the time limit results in the processing segment 203 not being compressed and being output by the compression apparatus 105 in its original form.

Furthermore, if none of the chosen CA configurations are found to result in successful compression, the processing segment 203 is not compressed and is output by the compression apparatus 105 in its original form.

Why Combination Arrays are used

In a similar way to assigning levels to bit portions (explained above), the level in which a CA value is placed affects how large the disambiguation value assigned to the CA value can be.

In preferred embodiments, CA values in the 1st level (level 0) are not allocated disambiguation values, and therefore all new CA values assigned to CA values in Level 0 must be unique.

It is noted that a bit portion having only 2 levels may not be able to be compressed using only the bit portion, or using a single combination array comprising all the bits of the bit portion (unless not all bit portion values, or not all CA values, occur within the processing segment 203). In such cases dividing the bit portion up into a plurality of combination arrays can allow compression to be achieved.

It is also noted that the higher the number of levels, the more compression will be achieved, because the resulting label will be smaller.

Frequency analysis is performed on the bit portions 205. In preferred embodiments, the bit portions 205 are sub-divided up into smaller combination arrays 207 and frequency analysis is also performed on these combination arrays 207. For example, the bit portion 205 may be divided up into a left hand part and a right hand part, such as combination arrays CA_0 and CA_1 in Figures 4B and 4C. The frequency analysis of the left hand parts (the CA_0 values) allows the most occurring left hand part to be determined. Similarly, the frequency analysis of the right hand parts (the CA_1 values) allows the most occurring right hand part to be determined.

In preferred embodiments, the labels assigned to the bit portions are not only dependent on the frequency of occurrence of the whole bit portions, but also on the frequency of occurrence of the combination arrays which make up the bit portions. Therefore, in the example where the bit portion 205 is divided up into a left hand part and a right hand part, the most occurring left hand part of the bit portion will be associated with the smallest new CA_0 values, and the most occurring right hand part of the bit portion will be associated with the smallest new CA_1 value. Typically labels generated based on analysis of combination arrays will allow greater compression than labels generated only based on analysis of bit portions.

Also, breaking up bit portions 205 into combination arrays 207 allows the header 211 to use fewer bits. For example, consider a bit portion comprising 5 bits. Table 6 illustrates two possible CA configurations which can be used for a bit portion length of 5 bits – [5,0,0,0,0] and [2,3,0,0,0].

As shown in Table 6, if a CA of length 5 bits is used, the number of possible CA values (and BP values as in this case the combination array is the same as the bit portion) is:

$$2^{L_{CA}} = 2^5 = 32$$

For each of the 5 combination arrays, in the biggest header format all of the possible CA values are written out in order of occurrence, and therefore the maximum number of bits used for CA values within header is 32 * 5 = 160 bits in total.

As shown in Table 6, if two CAs of length 2 and 3 bits are used, the number of possible CA values (and BP values as in this case the combination array is the same as the bit portion) for CA length 2 is:

$$2^{L_{CA0}} = 2^2 = 4$$

The number of possible CA values (and BP values as in this case the combination array is the same as the bit portion) for CA length 3 is:

$$2^{L_{CA1}} = 2^3 = 8$$

For each of the 2 CA_0 combination arrays, in the biggest header format all of the possible CA values are written out in order of occurrence, and therefore the maximum number of bits used for CA values within header is 32 * 5 = 160 bits in total.

Table 6 illustrates the maximum number of bits used for CA values within the header for the bit portion alone (which can considered as a combination array comprising 5 bits) and for the bit portion being divided in to two combination arrays of 2 bits and 3 bits respectively.

CA configuration	Maximum no. of bits used for CA values within header
[5,0,0,0,0]	160 bits in total (32 * 5)
[2,3,0,0,0]	32 bits in total (4 * 2) + (8 * 3)

Table 6

As can be seen in Table 6, dividing the bit portion up into combination arrays results in fewer bit being used for the CA values in the header.

Method of Assigning labels to Bit Portion Permutations using Combination Arrays

Figures 7A and 7B illustrate a first part of a method of assigning labels to the permutations of bits represented by bit portions 205, and hence to the corresponding bit portions 205, once a combination array CA configuration has been selected according to the method illustrated in Figures 6A to 6D.

In this example, the CA configuration 28 [3,3,0,0,0,0] was selected (as shown in Figure 6D), which means that each bit portion 205 is split up into two combination arrays – CA_0 and CA_1. Figures 7A and 7B illustrate how, for each possible CA_0 value and each possible CA_1 value, a new CA value 701 and a disambiguation value 703 is assigned. Figure 10, described below, illustrates how these new CA values 701 and disambiguation values 703 are used to generate labels for bit portion permutations.

The way in which new CA values 701 and disambiguation values 703 are assigned to CA values is similar to the way in which new BP values and disambiguation values are assigned to bit portion values, as shown in Figures 5B and 5C.

As stated above, the level to which a CA value is assigned affects how large the disambiguation value 703 assigned to the CA value can be.

The CA_0 and CA_1 values are initially assigned occurrence based levels, as explained above in reference to Figures 6B and 6C. However, in preferred embodiments, before new CA values 701 and disambiguation values 703 are assigned, the division of the CA_0 and CA_1 values into levels is

optimised. The optimisation of levels for CA values follows a similar principle to optimisation of bit portion values, as described above.

A general aim of level optimisation is to ensure that the number N_{Lev}^{CA} of CA values in each level is divisible by 2^{Lev} without remainder, where Lev is the level. This ensures efficient use of the assigned new CA values and disambiguation values.

This can be represented mathematically as:

$$N_{Lev}^{CA} \, mod \, 2^{Lev} = 0 \qquad \text{Equation 6}$$

For example, as shown in Figure 6B, Level 2 includes 3 CA values, so $N_{Lev}^{CA} = 3$, and for Level 2, Lev = 2, therefore the number N_{Lev}^{CA} of CA values in the level is not divisible by 2^{Lev} without a remainder.

Specifically:

$$N_{Lev}^{CA} \, mod \, 2^{Lev} = 3 \, mod \, 2^2 = 3 \, mod \, 4 = 3$$

In some examples, the condition $N_{Lev}^{CA} \, mod \, 2^{Lev} = 0$ is satisfied by moving the highest-occurring CA values in the level from level Lev to level Lev -1. In other examples, the condition $N_{Lev}^{CA} \, mod \, 2^{Lev} = 0$ may be satisfied by adding additional CA values to the level (e.g. the least-occurring CA value(s) from level Lev -1 are moved to level Lev).

In the present example, the least-occurring CA value, 101, from level 1 is moved to level 2.

In this way, the levels are optimised such that the number of CA values in each level is a multiple of 2^{Lev} or equal to 2^{Lev}, thus satisfying $N_{Lev}^{CA} \, mod \, 2^{Lev} = 0$.

This process of determining whether the condition $N_{Lev}^{CA} \, mod \, 2^{Lev} = 0$ is satisfied is repeated for each level of each combination array, in this example combination arrays CA_0 and CA_1.

As described in relation to optimising levels of BP values, for level 0, $N_{Lev}^{CA} \, mod \, 2^{Lev}$ will always equal 0, because 2^0 is equal to 1. Therefore, the condition $N_{Lev}^{CA} \, mod \, 2^{Lev} = 0$ is always fulfilled for level 0, regardless of how many CA values are present in level 0.

Preferred further conditions for optimising CA levels are described below.

In Figures 7A and 7B, the levels of CA_0 and CA_1 are optimised using the condition above and the further preferred conditions for optimising described below.

Once the levels of CA_0 and CA_1 have been optimised, each of the CA_0 and CA_1 values can be assigned a new CA value 701 and a disambiguation value 703.

As described below, where a bit portion permutation is made up of a particular CA_0 value and a particular CA_1 value, the new CA values 701 and disambiguation values 703 associated with the

CA_0 value and the CA_1 value are combined to generate a label for the bit portion value represented by that permutation.

In a simplified example, the CA_0 value 011 may be assigned a new CA value of 2, and a disambiguation value of 1. The CA_1 value 101 may be assigned a new CA value of 3, and a disambiguation value of 2.

To generate a label for the bit portion permutation 011101, the new CA values and disambiguation values for the CA values 011, 101 are combined. Specifically, new CA values 2 and 3 are combined by addition to give a combined new CA value of 5. New disambiguation values 1 and 2 are combined by addition to give a combined disambiguation value of 3. The label for bit portion permutation 011101 is created using the combined new CA value and the combined disambiguation value, so the label is 5, 3 – which is preferably represented in binary, as 10111. The bit portion permutation 011101, comprising 6 bits, is therefore represented using the label 10111, which comprises 5 bits. As the label comprises fewer bits that the bit portion permutation it represents, compression is achieved for all bit portions 205 having bits arranged in that bit portion permutation.

As can be seen in Figures 7A and 7B, new CA values 701 are assigned based on the level a CA value is in, in a similar way to assigning new BP values as described above. As a general rule, the lower the level (where Level 0 is the lowest), the fewer CA values are assigned the same (repeated) new CA value. In this embodiment, maximum number of repetitions of a new CA value is set to be 2^{Lev}, where Lev is the level of the CA values being assigned new values. This ensures that the maximum instance of CV values with a high frequency of occurrences is small and the maximum instance of CV values with a low frequency of occurrence is larger. For example, in level 3, the same new CA value can be assigned to up to 8 CA values.

A more general example of new CA value repetition is shown in Table 7, below.

CA Value	CA Level	New CA Value
a_0	Level 0	j_0
a_1	Level 0	j_1
a_2	Level 0	j_2
a_3	Level 0	j_3
a_4	Level 0	j_4
a_5	Level 1	j_5
a_6	Level 1	j_5
a_7	Level 1	j_6
a_8	Level 1	j_6
a_9	Level 2	j_7
a_{10}	Level 2	j_7
a_{11}	Level 2	j_7
a_{12}	Level 2	j_7

Table 7

As shown in Table 7, each new CA value is repeated 2^{Lev} times. In level 0, new CA values are repeated $2^0 = 1$ times each. In level 1, new CA values are repeated $2^1 = 2$ times each. In level 2, new CA values are repeated $2^2 = 4$ times each.

Since the assigned new CA values do not unambiguously identify an associated CA value in all cases, a disambiguation value is used to identify a particular one of the multiple CA values associated with the same new CA value.

The condition that the maximum number of repetitions of a new CA value in a level is 2^{Lev} ensures that the most frequently occurring CA values are assigned the shortest disambiguation values. For example, the CA values in the first level (Level 0) are each assigned unique new values, since $2^0 = 1$ (as can be seen in Figures 7A and 7B). No disambiguation values are therefore used for CA values in level 0.

When assigning new CA values 701 to the CA values in level 1 onwards, the same new CA values 701 can be assigned to multiple CA values. Where this re-use of new CA values 701 occurs, the number of disambiguation values 703 which are needed corresponds to the number of bit portion values which have been assigned the same new bit portion value.

Considering Figures 7A in detail, in CA_0 the two CA values in level 0 are assigned new CA values of 0 and 1 respectively. The instance column in Figure 7A provides a count of new CA values, starting at 0. As can be seen from the instance column, there is only a single instance of each of the level 0 CA values. Therefore, no disambiguation information is assigned to either of the level 0 CA values.

The two CA values in level 1 are both assigned a new CA value of 2, and therefore the first (e.g. most occurring) level 1 CA value is assigned an instance value of 0 and the second (e.g. next most occurring) level 1 CA value is assigned an instance value of 1. Disambiguation values 703 are also assigned to the CA values. In this first combination array, CA_0, the disambiguation values can simply use the instance values, as there are no previous combination arrays to affect the disambiguation values.

In Figure 7A there are four CA values in level 2, and therefore these CA values are all assigned a new CA value of 3, and disambiguation values of 0, 1, 2 and 3, corresponding to their instance values.

Considering Figure 7B in detail, in CA_1 level 0 contains two CA values, and each is assigned a new CA value with a single instance - in this case the new CA values are 0 and 4 respectively. The two CA values in level 1 of CA_1 are assigned a new CA value of 8, while the four CA values in level 2 of CA_1 are assigned a new CA value of 12.

The new CA values assigned in CA_0 and CA_1 are selected such that any combination of new CA values from each of the combination arrays results in a unique combined new CA value.

Figures 8A to 8D are tables detailing possible combined new CA values with their corresponding new CA_0 values and new CA_1 values.

Figure 8A is a table detailing every possible combination of new CA_0 values and new CA_1 values for the example illustrated in Figures 7A and 7B. As can be seen, the resulting combined new CA

values contain no repetitions. Each combined value uniquely identifies a particular combination of a new CA_0 value and a new CA_1 value – for example the combined new CA value 7 can only be arrived at by combining new CA values 3 and 4 (using addition in this embodiment).

The new CA_0 values are consecutively numbered from 0 to 3, while the new CA_1 values are multiples of 4, from 4*0 to 4*3. As can be seen Figure 8A, this results in efficient assigning of combined new CA values, because all the resulting values are consecutive, thus ensuring that the largest combined new CA value is as small as it can be (15 in this example).

More generally, the new CA values assigned for a combination array are multiples of the highest new CA value in the previous array + 1.

Figure 8B shows the combined new CA values for the example illustrated in Figures 7A and 7B in binary. The number of binary bits used to represent each of the combined new CA values is based on the size of the maximum combined new CA value, which in this example is 15. The number 15 is represented using four bits in binary (1111) and therefore all combined new CA values are represented using four bits.

Figure 8C shows generalised new CA values for CA_0 and CA_1 where CA_0 is assigned new CA values from X_0 to X_n and CA_1 is assigned new CA values from Y_0 to Y_n. As can be seen in Figure 8C, in preferred embodiments the combined new CA values are generated by adding the corresponding new CA_0 and CA_1 values together.

Figure 8D shows a further generalised way of assigning combined new CA values. In this Figure, each of the combined new CA values are unique (represented by values z_0 to z_{19}), however these values can be generated using any method and are not necessarily generated by adding together new CA values of CA_0 and CA_1.

As can be seen from Figure 7A and 7B, the way in which the disambiguation values 703 are assigned for a configuration array depends on the disambiguation values 703 used in the previous combination array. In Figure 7B, the “Instance” column shows the same Instances as Figure 7A. However, the three disambiguation value columns in Figure 7B show how the disambiguation values of CA_1 change based on the previous combination array, CA_0.

In a similar way as described above in relation to new combination arrays, the disambiguation values of combination arrays, such as CA_0 and CA_1, are combined to generate a combined disambiguation value.

The disambiguation values assigned in CA_0 and CA_1 are selected such that any combination of disambiguation values from each of the combination arrays results in a unique combined disambiguation value. Furthermore, the disambiguation values are preferably selected such that the smallest possible integers are used as disambiguation values, while still resulting in unique combined disambiguation values.

This can be seen in Figure 9, which is a table detailing possible combination of CA_0 disambiguation values and CA_1 disambiguation values, and the resulting combined disambiguation values, represented in binary.

As can be seen, the disambiguation values associated with the combination array depends on the level of the CA values being combined.

If both of the new CA_0 and CA_1 values are in level 0, there are no disambiguation values to be combined. This means that the resulting label for the bit portion permutation corresponding to such CA values will include a combined new CA value (in this example comprising four bits) but will not include a combined disambiguation value. The new CA values in level 0 are the most occurring values and therefore this method of generating labels ensures that the bit portions comprising the most occurring CA values will be assigned the shortest labels.

In all other instances, Figure 9 shows the possible CA disambiguation values for each combination array and the resulting combined disambiguation values for the example illustrated in Figures 7A and 7B.

As can be seen, the disambiguation values for CA_0 are 0-1 for level 1 and 0-3 for level 2. The disambiguation values for CA_1 for level 1 can be 0-1, 0 and 2, or 0 and 4; while for level 2 the disambiguation values can be 0-3; 0, 2, 4 and 6; or 0, 4, 8 and 12.

This ensures that all the resulting combined disambiguation values contain no repetitions. Each combined disambiguation value uniquely identifies a particular combination of a CA_0 disambiguation value and a CA_1 disambiguation value. Furthermore, as can be seen Figure 9, this results in efficient assigning of combined disambiguation values, because all the resulting combined disambiguation values in each table of Figure 9 are consecutive, thus ensuring that for each possible combination of disambiguation values the largest combined disambiguation value is as small as it can be (maximums may be 1, 11, 111 or 1111 in this example).

More generally, the disambiguation values assigned for the CA_1 combination array are multiples of the highest disambiguation value in CA_0 + 1 (with the multiples starting at 0).

The number of bits used to represent each combined disambiguation value depends on the maximum combined disambiguation value for the levels being combined. For example, combining the disambiguation value 2 from level 2 of CA_0 and 4 from level 2 of CA_1 results in a combined disambiguation value of 6 which is represented in binary using 4 bits as 0110 because the maximum combined disambiguation value for combining CA_0 level 2 with CA_1 level 2 is 15 which in binary using 4 bits is 1111. It can be seen in Figure 9 that by adding together the levels associated with each combination array determines the length, in bits, of the combined disambiguation values, for example combining CA_0 level 1 with CA_1 level 2 results in a 3 bit disambiguation length.

Generally, the higher-occurring the CA values being combined are, the fewer bits will be present in the combined disambiguation value. As explained above, the bit portions comprising the most occurring CA values will be assigned the shortest labels, in which the labels do not include disambiguation information.

Figure 10 illustrates how labels are assigned to bit portions permutations, by dividing the bit portion into combination arrays according to the chosen CA configuration and combining the new CA values and instance values associated with the combination array values of each of the combination arrays 207 of the bit portion 205.

As shown in the example of Figure 10, the length of the first part (the "Combined new CA value") in bits remains constant for all bit portion values in a processing segment 203, while the length of the second part can vary, or the second part may not be used at all to identify some bit portion values.

Advantageously, using labels in which the length of the first part is constant means that during decompression the labels can be read more easily by the decompression apparatus 505, for example requiring less processing power, compared to existing compression methods which use labels which are based on prefix code alone.

This is because the decompression apparatus does not need to analyse each individual incoming bit in order to determine the division between labels. Instead, the decompression apparatus 505 can determine from the header 211 how many bits the first part of each label will comprise (for example in Figure 10 the first part always comprises 4 bits, for instance 0000). It can also determine from the header how many instance bits (if any) will follow a first part from the value of the first part itself (e.g. first part 0010 in Figure 10 is always followed by one bit - either a 0 or a 1).

Figure 11 is a table listing all of the possible bit portions of length $L_{BP} = 6\ bits$ and the labels assigned to each bit portion permutation, based on the combination arrays CA_0 and CA_1 in Figures 7A and 7B. As can be seen, the labels vary in length from 4 bits to 8 bits. The 4 bit labels are associated with the most occurring combined CA values, while the 8 bit labels are associated with the least occurring combined CA values. As explained above, the most occurring CA values do not have any disambiguation value assigned, and therefore the 4 bit labels associated with the most occurring combined CA values comprise only the combined new CA value part, without a combined disambiguation value part.

The labels made up of 5, 6, 7 and 8 bits all comprise a 4 bit combined new CA value part, along with a combined disambiguation value part which comprises 1, 2, 3 or 4 bits respectively.

It is noted that bit portions with 4 bit labels occur approximately twice as frequently as bit portions assigned a 5 bit label, four times as frequently as bit portions assigned a 6 bit label, eight times as frequently as bit portions assigned a 7 bit label and sixteen times as frequently as bit portions assigned an 8 bit label. This is because each additional bit in the disambiguation value represents an approximate halving of frequency of occurrence of the combined combination values. This is in turn due to the fact that the disambiguation value assigned to each CA value, for example as shown in Figures 7A and 7B, is based on the level of the CA value, which is determined based on frequency analysis. It is noted that the effect of optimising levels means that the halving of frequency between successive levels is only approximate.

Figures 12A to 12D are examples of generating new CA values (and disambiguation values) for bit portions having a bit portion length L_{BP} of 8 bits, using a CA configuration of [5,3] – a five bit combination array and a three bit combination array.

Figures 12A shows the new CA values assigned to the original CA values of the 5-bit CA_0 and the 3-bit CA_1 combination arrays. The CA values of the 5-bit CA_0 are divided into 3 levels, and the CA values of the 3-bit CA_1 are also divided into 3 levels.

Figure 12A shows that the resulting maximum combined new CA value would be 63. This value can be representing in binary using 6 bits, and therefore the minimum bits label length is 6 bits.

The combined new CA values are generated by combining the two combination arrays - 5-bit CA_0 and 3-bit CA_1 – which each have 3 levels, and therefore the total number of levels in the combination arrays is 6.

Figure 12B shows the possible disambiguation value lengths (in bits) in relation to the levels of the CA_0 and CA_1 values being combined in Figure 12A.

It can be seen in Figure 12B that each disambiguation value length (in bits) is the sum of levels of the CA values being combined – for example combining CA_0 level 1 with CA_1 level 2 results in a 3 bit disambiguation length.

The greatest number of bits used for the combined disambiguation values can also be determined by subtracting the number of arrays being combined from the total number of levels in the arrays. In this case, the total number of levels in the combination arrays is 6, and the number of arrays being combined is two, to the maximum combined disambiguation length is 4 (6 – 2 = 4).

As all combined CA values comprise 6 bits, the label length (in bits) is shown in Figure 12B as the disambiguation bit length + 6.

As can be seen from Figure 12B, only 3 combinations have an label bit portion length of more bits than the input bit portion length (8 bits), meaning that 67% of the labels are either the same size or smaller than the input bit portion length.

In in Figure 12C, the same CA configuration of [5,3] is used, however in this example the 5 bit combination array has been changed to use 4 levels instead of 3.

Figure 12C shows that the resulting maximum combined new CA value would be 31. This value can be representing in binary using 5 bits, and therefore the minimum bits label length is 5 bits.

The combined new CA values are generated by combining the two combination arrays - 5-bit CA_0 and 3-bit CA_1 – which have 4 and 3 levels respectively, and therefore the total number of levels in the combination arrays is 7.

Figure 12D shows the possible disambiguation value lengths (in bits) in relation to the levels of the CA_0 and CA_1 values being combined in Figure 12C.

It can be seen in Figure 12D that each disambiguation value length (in bits) is the sum of levels of the CA values being combined – for example combining CA_0 level 3 with CA_1 level 1 results in a 4 bit disambiguation length.

The greatest number of bits used for the combined disambiguation values can also be determined by subtracting the number of arrays being combined from the total number of levels in the arrays. In this case, the total number of levels in the combination arrays is 7, and the number of arrays being combined is two, to the maximum combined disambiguation length is 5 (7 – 2 = 5).

As all combined CA values comprise 5 bits, the label length (in bits) is shown in Figure 12B as the disambiguation bit length + 5.

As can be seen from Figure 12D, only 3 combinations have an label bit portion length of more bits than the input bit portion length (8 bits), meaning that 75% of the labels are either the same size or smaller than the input bit portion length.

It is noted that even though the number of levels used for CA_0 in Figures 12C and 12D has increased from 3 to 4, the maximum label size in bits remains at 10 bits.

Preferred conditions for optimising CA levels

In preferred embodiments, level optimisation is based on the following conditions.

Firstly, the number of levels in a combination array should not exceed the combination array length:

$$N_{CA}^{LevelsMAX} = L_{CA} \qquad \text{Equation 7}$$

Secondly, the number N_{Lev}^{CA} of combination array values in each level should be divisible by 2^{Lev}

$$N_{Lev}^{CA} \bmod 2^{Lev} = 0 \qquad \text{Equation 8}$$

Thirdly, the maximum new combination array value assigned to one or more values in a combination array should equal a target maximum new combination array value.

$$MaxNewCAVal = TargetMaxNewCAVal \qquad \text{Equation 9}$$

Where the target maximum new combination array value assigned to one or more combination array values is defined as follows:

$$TargetMaxNewCAVal = 2^{\left\lfloor log_2(N_{CA}^{Levels}) \right\rfloor + 1} - 1 \qquad \text{Equation 10}$$

And the maximum new combination array value is defined as follows:

$$MaxNewCAVal = \left(\sum_{Lev=0}^{Lev=N_{CA}^{Levels}-1} \frac{N_{Lev}^{CA}}{2^{Lev}} \right) - 1 \qquad \text{Equation 11}$$

L_{CA} is the combination array length in bits;

N_{CA}^{Levels} is the number of levels into which the combination array values of a combination array 207 are divided;

$N_{CA}^{LevelsMAX}$ is the maximum number of levels into which the combination array values of a combination array 207 can be divided;

Lev is the level index, for example $Lev = 0$ for level 0 and $Lev = 1$ for level 1;

$MaxNewCAVal$ is the maximum new combination array value assigned to one or more combination array values in a processing segment

$TargetMaxNewCAVal$ is the target maximum new combination array value assigned to one or more bit portion values in a processing segment;

N_{Lev}^{CA} is the number of combination array values in a level;

However, in some situations not all conditions can be met. For example, if only two levels are present, it may not be possible for the maximum new combination array value to equal the target maximum new combination array value, but all other conditions can be met. In such situations, for example in Figure 5B, the combination array configuration can still be used, as compression is still achievable.

Splitting the analysed CA values into more levels generally results in a smaller maximum new value and ultimately smaller labels being assigned to bit portions.

Hard-to-compress data

It is possible to achieve compression using the above described methods even if the frequency of occurrence of BP/CA values is substantially even across all possible BP/CA values and thus all BP/CA values exist in the same level, so long as at least one of the BP values and/or CA values has an occurrence of 0.

Compression can be achieved in such cases by assigning one of the BP values and/or CA values to a different level (e.g. assigning the first BP/CA value to level 0 and all others to level 1). This causes the first BP and/or CA value to be assigned fewer disambiguation value bits than the remaining BP and/or CA values (for example, the BP value in level 0 may not be assigned a disambiguation value, and as a result the label assigned to the BP value in level 0 will be 1 bit in length smaller than the BP values in level 1).

For example, using a bit portion length of 8 bits, the BP value of level 0 is assigned a new BP value of 0000000 with no disambiguation information. This level 0 BP value is therefore assigned a label which is 7 bits long; 1 bit shorter than the original 8 bits of the BP value. The remaining BP values are assigned new BP values of 1-127 (0000001 – 1111111), each with a disambiguation value of either 0 or 1. Therefore, the level 1 new BP values are assigned labels with 8 bits, which is the same number of bits as the original BP values.

Due to the relatively small size of the header in most situations, compression can still be achieved even if BP/CA value in level 0 has exactly the same number of occurrence as all other BP/CA values. This occurs more often when using smaller bit portion lengths because the occurrence values are higher and offset the header size. It is noted that that the BP/CA values are preferable not sorted to achieve this compression, in order to avoid having to use a larger

header to indicate how the BP/CA values have been sorted. Therefore, the BP/CA value assigned to a different level (e.g. level 0) need not be the most occurring.

It should be noted that for each additional BP value and/or CA value that is not in use (occurrence is 0), the compression which can be achieved increases.

For example, if two BP/CA values have an occurrence of 0, the first two BP/CA values can be assigned to level 0 and the remaining BP/CA values would be assigned to level 1. The result would be that the two new BP/CA values in level 0 are assigned labels which are 1 bit shorter than the original BP/CA values. All other BP/CA values would be assigned labels which are the same length as the original BP/CA values.

For a substantially evenly distributed processing segment, new BP/CA values and disambiguation values are assigned in the same way, until the point at which 50% of the available BP/CA values be not in use. At this point, all BP/CA values can be assigned to level 0 and compression can still be achieved.

If more than 50% of the BP/CA values have an occurrence of 0, higher compression can be achieved by assigning new BP/CA values and disambiguation values are in the same way as described above. At this point each additional BP/CA value with an occurrence of 0 can be assigned to level 0, resulting in labels which are two bits shorter. The other BP/CA values can be assigned to level 1, resulting in labels which are one bit shorter than the original BP/CA values.

Number of BP/CA values with occurrence greater than 0	**No. of bits saved in level 0**	**No. of bits saved in level 1**
128 to 255	1 bit less	0 bit less
64 to 127	2 bits less	1 bit less
32 to 63	3 bits less	2 bits less
16 to 31	4 bits less	3 bits less
8 to 15	5 bits less	4 bits less
4 to 7	6 bits less	5 bits less
1 to 3	7 bits less	6 bits less

Table 10

Table 10 shows the number of bits saved for a BP/CA with length of 8 bits, depending on the number of BP/CA values with occurrence greater than 0.

Bit Portion Length (in bits)	**Number of different possible BP values (i.e. permutations of bits)**	**Expected number of occurrences of each possible BP value in a processing segment of length 64KB**
1	2	262144
2	4	65536
3	8	21846
4	16	8192
5	32	3277
6	64	1366
7	128	586
8	256	256
9	512	114

10	1024	52
11	2048	24
12	4096	11
13	8192	5
14	16384	3
15	32768	2
16	65536	1

Table 11

Table 11 shows the expected number of occurrences of each possible BP value in a processing segment of length 64KB, where all possible BP values occur in the processing segment, and the frequency of occurrence of BP values is substantially even across all possible BP values. For example, if the bit portion length is 1, each of the possible BP values (0 and 1) would be expected to occur 262144 times in the processing segment of 65536 bytes (64KB).

In the case where the frequency of occurrence of BP values is substantially even across all possible BP values, but at least one BP value does not occur, the number of bits which can potentially be saved is the expected number of occurrences of each possible BP value in a processing segment shown in Table 11 above (less the size of the header). For example, for a bit portion length of 3, if only 7 of the 8 possible BP values occur in the processing segment, 21846 bits could potentially be saved (less the size of the header). As long as the header does not exceed the expected number of occurrences, compression can be achieved.

The expected number of occurrences of each possible BP value in Table 11 is given by:

$$N_{occurrences}^{BP} = \frac{L_{PS}}{L_{BP} \times 2^{L_{BP}}} \qquad \text{Equation 12}$$

Where:

L_{PS} is the processing segment length in bits.

Table 12 shows the expected number of occurrences of each possible CA value for a bit portion length of 4 bits.

Bit Portion Length (in bits)	**Combination array Length (in bits)**	**Number of different possible CA values (i.e. permutations of bits)**	**Expected number of occurrences of each possible CA value in a processing segment of length 64KB**
4	1	2	65536
4	2	4	32768
4	3	8	16384
4	4	16	8192

Table 12

The possible CA configuration for a bit portion length of 4 are [1,1,1,1], [1,1,2,0], [1,2,1,0], [1,3,0,0], [2,1,1,0], [2,2,0,0], [3,1,0,0], [4,0,0,0]. Therefore, as shown in Table 12, CA lengths of 1, 2, 3 and 4 are possible.

Table 12 shows the expected number of occurrences of each possible CA value in a processing segment of length 64KB, where all possible CA values occur in the processing segment, and the

frequency of occurrence of CA values is substantially even across all possible CA values. For example, if the CA length is 1, each of the possible CA values (0 and 1) would be expected to occur 65536 times in the processing segment of 65536 bytes (64KB).

In the case where the frequency of occurrence of CA values is substantially even across all possible CA values, but in one combination array at least one CA value does not occur, the number of bits which can potentially be saved is the expected number of occurrences of each possible CA value in a processing segment shown in Table 12 above (less the size of the header). For example, for a CA length of 3, if only 7 of the 8 possible CA values of a combination array occur in the processing segment, 16384 bits could potentially be saved (less the size of the header). As long as the header does not exceed the expected number of occurrences, compression can be achieved.

The expected number of occurrences of each possible CA value in Table 12 is given by:

$$N_{occurrences}^{CA} = \frac{L_{PS}}{L_{BP} \times 2^{L_{CA}}} \qquad \text{Equation 13}$$

Header Structure

Figures 13A to 13D are simplified representations of four exemplary header structures.

As stated previously, preferably, each header starts with a compression method signature, and provides information relating to the chosen bit portion length L_{BP}, the combination array configuration used, the size of the original processing segment 203, and information on how labels were assigned to each of the bit portions 205. Figures 13A to 13D are preferred header structures.

Header Format 0

As shown in Figure 13A, the header starts with a signature. In this example, the signature is referred to as a "SISP" signature, which is an exemplary trade name for the presently described compression method. The "SISP" signature is 32 bits long.

The header also specifies the bit portion length L_{BP}, which in this example is allocated 4 bits in the header (and therefore, in this example, the bit portion length L_{BP} can be a maximum of 16 bits). The number of bits allocated to the bit portion length L_{BP} in the header may be CPU dependent.

The CA configuration is also specified in the header, which uses L_{BP} bits. Preferably, the CA configuration is specified by its reference number, which (in combination with knowledge of the bit portion length L_{BP}) unambiguously identifies the CA configuration used to assign labels to the bit portions 205.

Furthermore, the size of the processing segment (in bytes) is specified, and in this embodiment the processing segment size can be between 0 and 65535 bytes because the length of the processing segment size part of the header is 16 bits as shown.

Also, in preferred embodiments, multiple different header formats can be used (e.g. 3). The header format can be chosen based on which will result in the smallest total header size for a

processing segment. Therefore, the header includes a part comprising two bits for indicating the header choice.

As described above, the CA configuration may use any number of arrays within a range, where the range is from one array to L_{BP} arrays (L_{BP} arrays would occur when all arrays are one bit in size). The header contains information relating to each of the combination arrays in the combination array configuration, and therefore as a minimum the header will contain CA_0 information if only one array is used by the CA configuration.

In the example given in Figure 13A, the CA configuration uses more than one array, and therefore the combination array information comprises CA_0 information through to CA_n information.

As shown, the CA_0 information comprises a count for each of levels 0 to L_{CA0}, where the count indicates how many CA_0 values are present in the respective level. The count can be from level 0 to level L_{CA0} because the maximum number of levels in a combination array is length of the combination array in bits (L_{CA}).

The CA_0 information further comprises a single bit indicator to indicate whether the CA_0 values are sorted (e.g. by frequency of occurrence).

The CA_0 information also comprises frequency of occurrence information, which indicates the rankings of CA_0 values and whether they are in use.

Specifically, if the CA values have been sorted, all possible CA values are written out in order of occurrence, including any CA values having an occurrence of 0.

If the CA values have not been sorted, then a single bit for each possible CA value is written out in unsorted order, where a value of 0 represents no occurrences of the CA value and a value of 1 represents one or more occurrences of the CA value.

As shown, the CA_n information comprises the same information fields as the CA_0 information.

It will be appreciated that equivalent information as the CA_0 information and CA_n information will be included in the header for any intervening combination arrays present in the CA configuration.

Header Format 1

Figure 13B illustrates header format 1. The header format contains the same parts as header format 0, with the exception of the frequency of occurrence information.

Specifically, for header format 1, if the CA values have been sorted, a single bit for each possible CA value is written out in unsorted order, where a value of 0 represents no occurrences of the CA value and a value of 1 represents one or more occurrences of the CA value. In addition to the occurrence indicating bit, additional bits may be included after this bit, depending on whether or not the occurrence is 0 and whether the CA value has been swapped with another CA value. If the occurrence of a CA value is greater than 0 and the CA value has not been swapped, a “swap indicator” bit of 0 is included after the occurrence indictor bit. If the occurrence of a CA value is greater than 0 and the CA value has been swapped, then a “swap indicator” bit is included after

the occurrence indictor bit along with the swapped CA value assigned to the CA value (where the swapped CA value is represented in bits).

If the CA values have not been sorted, then a single bit for each possible CA value is written out, where a value of 0 represents no occurrences of the CA value and a value of 1 represents one or more occurrences of the CA value.

As shown, the CA_n information comprises the same information fields as the CA_0 information.

It will be appreciated that equivalent information as the CA_0 information and CA_n information will be included in the header for any intervening combination arrays present in the CA configuration.

Advantageously, header format 1 does not write out every possible CA value in the CA value frequency of occurrence information part, and therefore the resulting header is smaller.

Header Format 2

Figure 13C illustrates header format 2. The header format contains the same parts as header format 1, with the exception that the frequency of occurrence information specifies the first occurring CA value (in this example 000) and the last occurring CA value (in this example 110), and that no information on CA values before and after these first and last occurring CA values is included.

As shown, the CA_n information comprises the same information fields as the CA_0 information.

It will be appreciated that equivalent information as the CA_0 information and CA_n information will be included in the header for any intervening combination arrays present in the CA configuration.

Advantageously, header format 2 does not write out every possible CA value in the CA value frequency of occurrence information part, and therefore the resulting header is smaller.

Header Format 3

Figure 13C illustrates header format 3. This header format contains the same parts as the other header formats, with the exception that no frequency of occurrence information is included. This means that header format 3 is preferably only be used if all CA_0 to CA_n values occur in the processing segment and no sort has occurred.

It is noted that the headers advantageously don't require the mapping between each CA (or BP) value and the corresponding new value (or label) to be written out specifically. The header instead just indicates how the permutations (BP or CA values) have been grouped, which allows a decompression apparatus to determine the mapping between each CA (or BP) value and the corresponding new value (or label).

Reprocessing processed segments and associated headers

Once all of the bit portions 205 of a processing segment 203 have been assigned labels, a processed_segment 209 is output in which the bit portions are represented using their respective labels. A header 211 is output with the processed_segment 209 in order to allow the processed segment 209 to be decompressed (e.g. by a decompression apparatus 505).

In preferred embodiments, the processed_segment 209 and associated header 211 are then reprocessed, using the methods described above, treating the processed_segment 209 and associated header 211 as a new processing segment 203. The reprocessing of the processed segment 209 and associated header 211 results in the generation of a new processed_segment 209 and new associated header 211, where the total size in bits of the new processed_segment 209 and new associated header 211 is less than the total size in bits of the processed_segment 209 and associated header 211.

In alternative embodiments, the compressed file 202 is reprocessed, by dividing the compressed file 202 into new processing segments and performing the methods for compression described above.

Although it is recognised that generally it is not possible to recompress data using the same compression method, the methods of compression described in this application advantageously allow the way in which data is processed to be significantly varied, on the fly, (for example changing bit length, changing CA configuration, changing grouping (levels) of permutations (BP or CA values), in order to allow data to be recompressed at least once.

Alternative Method for Calculating Target Maximum BP values and CA values

Figure 14 illustrates the target maximum BP and/or CA values calculated in accordance with an alternative embodiment. In this embodiment, the target maximum new bit portion value assigned to one or more bit portion values in a processing segment is defined as follows:

$$TargetMaxNewBPVal = 2^{\left\lfloor log_2\left\lfloor log_2(N_{BP}^{Present})\right\rfloor\right\rfloor+1} - 1 \qquad \text{Equation 14}$$

$N_{BP}^{Present}$ is the number of bit portions with an occurrence greater than 0 in the processing segment.

Similarly, in this embodiment, the target maximum new CA value assigned to one or more CA values in a processing segment is defined as follows:

$$TargetMaxNewCAVal = 2^{\left\lfloor log_2\left\lfloor log_2(N_{CA}^{Present})\right\rfloor\right\rfloor+1} - 1 \qquad \text{Equation 15}$$

$N_{CA}^{Present}$ is the number of combination arrays with an occurrence greater than 0 in the processing segment.

As can be seen from Figure 14, in an example where only 255 out of 256 possible bit portions are present in a processing segment (where a bit portion length L_{BP} = 8 is being used), the target maximum BP value is 7 rather than 15. Since 7 can be represented in binary using one less bit than 15, this means that the label assigned to the most occurring BP values will be one bit less.

This method of calculating the target maximum new bit portion value can advantageously achieve higher levels of compression; however it may require additional manipulation of the levels. For example, in some cases the number of levels determined using frequency analysis will be too low to achieve the target maximum new bit portion value. Therefore, in some embodiments where

this method of calculating the target maximum new bit portion value is used the bit portions values may be divided into levels using methods different to those described above.

Modifications and Alternatives

Detailed embodiments have been described above. As those skilled in the art will appreciate, a number of modifications and alternatives can be made to the above embodiments whilst still benefiting from the inventions embodied therein.

Although it is described above that a "file" is compressed, it will be appreciated that any data may be compressed using the same methods as described.

The processing segments 203 can have different sizes to one another even when forming part of the same file. In one example this allows the final segment 203 of a file 201 to have a smaller size than the other segments, avoiding the need to use padding bits/bytes. In other examples the size of the processing segments 203 can be chosen using a similar method to that used to choose the size of bit portions 205.

The bit portions 205 are generally all of the same size within a processing segment 203; however in some embodiments the bit portions 205 may be of different sizes (i.e. have different bit portion lengths) within a processing segment 203.

In some alternative embodiments, each bit portion corresponds to a byte (i.e. 8 bits) of the processing segment, and there is no further division of the bit portions into combination arrays.

In some embodiments, different bit portions of the same processing segment may be divided into different configurations of combination arrays, which allows further exploitation of patterns, repetition and/or redundancy in a processing segment 203.

Although Figure 3A shows the frequency analysis starting with a bit portion length of 2 bits, it can start at any bit portion length, for example 1 bit or 3 bits.

In other alternative embodiments, the target maximum new bit portion value assigned to one or more bit portion values in a processing segment is defined as follows:

$$TargetMaxNewBPVal = L_{BP} - \left\lfloor \frac{N_{BP}^{Levels}}{2} \right\rfloor$$

Similarly, in this embodiment, the target maximum new CA value assigned to one or more CA values in a processing segment is defined as follows:

$$TargetMaxNewCAVal = L_{CA} - \left\lfloor \frac{N_{CA}^{Levels}}{2} \right\rfloor$$

This method of calculating the target maximum new bit portion value and/or new CA value is simpler and therefore the calculation can be made more quickly and/or using less processing power, although the level of compression achieved may not always be as high.

Although in the description above "levels" are defined

In some alternative embodiments, the labels assigned to bit portions may be configured to be larger than the bit portions themselves, for example to increase the level of encryption.

It will be appreciated by those skilled in the art that binary can be written right to left or left to right. For example, the binary string 00010 would be considered to represent the number 2 if written right to left, but would be considered to represent the number 16 if written left to right. In preferred embodiments, the binary used in the methods described above is written left to right as this can make the decompression process quicker and easier when using variable bit length. For example, writing the binary left to right can make it easier to identify any padding bits included at the end of a processing segment.

In some alternative embodiments, no extraction information is included in a compressed file. In some cases, the same configurations are used to process all processing segments, and therefore the decompression apparatus can use information corresponding to a "static header" for decompressing all processed segments. In some alternative embodiments, extraction information is output separately to the processed segments.

Not including a header with processed segments can be particularly advantageous when encrypting data, as the "static header" acts as a key for compression, where the key is private and only available to the compression and decompression apparatus.

In some embodiments, for example when processing large amounts of similar data, all processing segments are processed in the same way, using the same configurations, and therefore no header is guaranteed by the compression apparatus 105, and the decompression apparatus 505 decodes all processed segments 209 in the same way.

It is noted that not all permutations may be assigned labels.

Various other modifications will be apparent to those skilled in the art and will not be described in further detail here.

The disclosure of this application also includes the following numbered clauses:

1. A method of processing data comprising an input sequence of bits, the method comprising the steps of:

(i) identifying a processing bit length for use in processing said input sequence of bits;

(ii) dividing the input sequence of bits into a plurality of portions wherein each portion has a respective portion bit length equal to said processing bit length and wherein the bits in each portion are arranged in a respective portion permutation;

(iii) respectively sub-dividing each portion into a plurality of sub-divisions comprising at least a first sub-division and a second sub-division, wherein each sub-division of the plurality of sub-divisions comprises at least one bit, wherein the at least

one bit of each first sub-division is arranged in a respective first sub-division permutation, and wherein the at least one bit of each second sub-division is arranged in a respective second sub-division permutation;

(iv) performing frequency analysis:

to determine, for each of a plurality of possible first sub-division permutations, how many times, within said input sequence of bits, a portion comprises a first sub-division having bits arranged in that possible first sub-division permutation; and

to determine, for each of a plurality of possible second sub-division permutations, how many times, within said input sequence of bits, a portion comprises a second sub-division having bits arranged in that possible second sub-division permutation;

(v) assigning, based on said frequency analysis, a first respective sub-division value to each of said plurality of possible first sub-division permutations and assigning a second respective sub-division value to each of said plurality of possible second sub-division permutations;

(vi) for each portion permutation of a plurality of possible portion permutations, generating a respective label representing that portion permutation, wherein said generating comprises combining:

the first sub-division value assigned to the first sub-division permutation corresponding to the first sub-division of that portion permutation; with

the second sub-division value assigned to the second sub-division permutation corresponding to the second sub-division of that portion permutation;

wherein said respective label comprises a representation of a combined value resulting from said combining; and

(vii) forming a processed sequence of bits by replacing, within said input sequence of bits, bit portions comprising bits arranged in one of said plurality of possible portion permutations, with the respective label representing that one of said plurality of possible portion permutations.

2. A method according to clause 1, wherein when generating, for each portion permutation, a respective label representing that portion permutation, said combining comprises:

arithmetically adding said first sub-division value assigned to the first sub-division permutation corresponding to the first sub-division of that portion permutation to said second sub-division value assigned to the second sub-division permutation corresponding to the second sub-division of that portion permutation;

wherein said combined value comprises a result of said addition.

3. A method according to any preceding clause, wherein when generating, for each portion permutation, a respective label representing that portion permutation, said generating further comprises:

when a particular first sub-division value is assigned for a plurality of different first sub-division permutations:

generating, for each of said respective plurality of different first sub-division permutations having that particular first sub-division value, a different respective first additional value for use in discriminating between said respective plurality of first sub-division permutations having that particular first sub-division value; and

when a particular second sub-division value is to be assigned for a plurality of different second sub-division permutations:

generating, for each of said respective plurality of different second sub-division permutations having that particular second sub-division value, a different respective second additional value for use in discriminating between said respective plurality of second sub-division permutations having that particular second sub-division value.

4. A method according to clause 3, wherein when generating, for each portion permutation, a respective label representing that portion permutation, said generating further comprises:

when a first additional value and a second additional value have been generated for a particular portion permutation:

combining said first additional value and said second additional value to produce a combined additional value, wherein the label for that particular portion permutation comprises a representation of the

combined value together with the combined additional value for that particular portion permutation.

5. A method according to clause 3, wherein when generating, for each portion permutation, a respective label representing that portion permutation, said generating further comprises:

when one of a first additional value and a second additional value have been generated for a particular portion permutation:

generating a label for that particular portion permutation that comprises a representation of the combined value together with that one of a first additional value and a second additional value.

6. A method according to any preceding clause, wherein, when respectively sub-dividing each portion into a plurality of sub-divisions, said first sub-division has a different number of bits to said second sub-division.

7. A method according to any preceding clause, wherein when generating, for each portion permutation, a respective label representing that portion permutation, each label generated has a respective label bit length, and the labels are generated such that labels generated for portion permutations which occur a greater number of times within said input sequence of bits generally have a smaller label bit length than labels generated for portion permutations which occur a lesser number of times within said input sequence of bits.

8. A method according to any preceding clause, wherein when generating, for each portion permutation, a respective label representing that portion permutation, each label generated has a respective label bit length, and the labels are generated such that at least some of the labels have a label bit length which is smaller than the processing bit length.

9. A method according to any preceding clause, wherein said frequency analysis comprises:

for each one of said plurality of possible first sub-division permutations:

determining a respective occurrence level which is the number of times, within said sequence of bits, that a portion occurs comprising that one of said plurality of possible first sub-division permutations; and

for each one of said plurality of possible second sub-division permutations:

determining a respective occurrence level which is the number of times, within said sequence of bits, a portion occurs comprising that one of said plurality of possible second sub-division permutations.

10. A method according to clause 9, wherein, for a given first sub-division value, the number of first sub-division permutations which are assigned the given first sub-division value depends on the occurrence levels associated with the first sub-division permutations which are assigned the given first sub-division value; and

wherein, for a given second sub-division value, the number of second sub-division permutations which are assigned the given second sub-division value depends on the occurrence levels associated with the second sub-division permutations which are assigned the given second sub-division value.

11. A method according to clause 9 or 10, wherein when assigning, based on said frequency analysis, a first (or second) respective sub-division value to each of said plurality of possible first sub-division permutations, said assigning comprises:

grouping, based on said frequency analysis, said plurality of possible first (or second) sub-division permutations into a plurality of sets (or 'levels');

wherein each set comprises at least one first (or second) sub-division permutation; and

wherein the at least one first (or second) sub-division permutation in each set has a corresponding occurrence level that falls within a different respective range of occurrence levels associated with that set.

12. A method according to clause 11, wherein for a given first (or second) sub-division value, the number of first sub-division permutations which are assigned the given first sub-division value depends on the set associated with the first (or second) sub-division permutation(s) which are assigned the given first sub-division value.

13. A method according to any preceding clause, wherein forming a processed sequence of bits further comprises including a header portion in the processed sequence of bits, said header portion comprising extraction information for use in reconstructing said input sequence of bits from said processed sequence of bits, and the extraction information being configured for use in identifying the respective portion permutation which each label represents.

14. A method according to clause 13, wherein said extraction information is configured for use in identifying how the said plurality of possible first (or second) sub-division permutations are grouped into sets.

15. A method according to clause 14, wherein said extraction information identifies how many first (or second) sub-division permutations each set comprises.

16. A method according to any of clauses 13 to 15, wherein the extraction information is further configured to identify said processing bit length used in processing said input sequence of bits.

17. A method according to any of clauses 13 to 16, wherein the extraction information is further configured to identify how each portion is sub-divided into a plurality of sub-divisions.

18. A method according to clause 17, wherein the extraction information is further configured to identify how many bits each first sub-division comprises and how many bits each second sub-division comprises.

19. A method according to any of clauses 13 to 18 wherein the extraction information is further configured to identify how many bits the input sequence of bits comprises.

20. A method according to any preceding clause, further comprising repeating steps (i) to (vii) at least one further time using said processed sequence of bits as said input sequence of bits.

21. A method of processing data, the method comprising the steps of:

(i) dividing the data into a plurality of processing segments wherein each processing segment comprises an input sequence of bits;

(ii) identifying a current processing bit length for use in processing a current processing segment of said data to form a processed segment meeting at least one predetermined processing criterion;

(ii) dividing the current processing segment into a plurality of portions wherein each portion has a respective portion bit length equal to said current processing bit length and wherein the bits in each portion are arranged in a respective one of a number of possible permutations;

(iv) assigning a respective label to each of a plurality of said possible permutations; and

(v) forming a processed segment by replacing, within said current processing segment, bit portions comprising bits arranged in one of said plurality of possible permutations with the respective label assigned to that one of said possible permutations;

(vi) identifying a new processing bit length for use in processing a next processing segment of said data to form a processed segment meeting at least one predetermined processing criterion;

(vii) repeating, for each of said plurality of processing segments, steps (ii) to (vi) wherein the new processing bit length is used as the current processing bit length and the next processing segment of said data is used as the current processing segment, and wherein a processing bit length used for at least one of said processing segments of said data is different to a processing bit length used for at least one other of said processing segments of said data.

22. A method of processing data comprising an input sequence of bits, the method comprising the steps of:

(i) setting a current processing bit length, of at least one bit, for use in processing said input sequence of bits;

(ii) dividing the input sequence of bits into a plurality of portions wherein each portion has a respective portion bit length equal to said current processing bit length and wherein the bits in each portion are arranged in a respective one of a number of possible permutations;

(iii) for each of a plurality of possible permutations analysing the input sequence of bits to respectively identify how many times, within said input sequence of bits, a portion having that possible permutation occurs;

(iv) determining whether at least one predetermined processing criterion has been achieved by comparing results of said analysing with the predetermined processing criterion;

(v) processing said input sequence of bits based on said determining wherein said processing comprises:

when the determining determines that the predetermined processing criterion has not been achieved performing at least one of: setting a new processing bit length that is different to the current processing bit length and repeating steps (ii) to (v) using said new processing bit length as the current processing bit length; and ending processing of said input sequence of bits; and

when the determining determines that the at least one predetermined processing criterion has been achieved: assigning a respective label to each of said plurality of possible permutations; and forming a processed sequence of bits by replacing, within said sequence of bits, bit portions comprising bits arranged in one of said plurality of possible permutations with the respective label assigned to that one of said possible permutations.

23. A method according to clause 21 or 22, wherein said predetermined processing criterion comprises whether 50% of the possible permutations which occur in the input sequence of bits occur at least twice as frequently as the other 50% of the possible permutations which occur in the input sequence of bits.

24. A method according to clause 21 or 22, wherein said predetermined processing criterion comprises whether 50% of the possible permutations occur in the input sequence of bits.

25. A method of reconstructing a processed sequence of bits produced by a method according to any preceding clause, the method comprising the steps of:

(i) obtaining extraction information for use in reconstructing an original sequence of bits from said processed sequence of bits;

(ii) reconstructing said original sequence of bits from said processed sequence of bits based on said extraction information.

26. A computer implementable instructions product comprising computer implementable instructions for causing a programmable communications device to perform the method according to any preceding clause.

27. Apparatus for processing data, the apparatus comprising at least one of an electronic circuit, an integrated circuit chip and a computer processor configured to implement the method of any of clauses 1 to 25.

Claims

1. A method of processing data comprising an input sequence of bits, the method comprising the steps of:

(i) setting a current processing bit length, of at least one bit, for use in processing said input sequence of bits;

(ii) dividing the input sequence of bits into a plurality of portions wherein each portion has a respective portion bit length equal to said current processing bit length and wherein the bits in each portion are arranged in a respective one of a number of possible permutations;

(iii) for each of a plurality of possible permutations analysing the input sequence of bits to respectively identify how many times, within said input sequence of bits, a portion having that possible permutation occurs;

(iv) determining whether at least one predetermined processing criterion has been achieved by comparing results of said analysing with the predetermined processing criterion;

(v) processing said input sequence of bits based on said determining wherein said processing comprises:

when the determining determines that the predetermined processing criterion has not been achieved performing at least one of: setting a new processing bit length that is different to the current processing bit length and repeating steps (ii) to (v) using said new processing bit length as the current processing bit length; and ending processing of said input sequence of bits; and

when the determining determines that the at least one predetermined processing criterion has been achieved: assigning a respective label to each of said plurality of possible permutations; and forming a processed sequence of bits by replacing, within said sequence of bits, bit portions comprising bits arranged in one of said plurality of possible permutations with the respective label assigned to that one of said possible permutations.

2. A method of processing data, the method comprising the steps of:

(i) dividing the data into a plurality of processing segments wherein each processing segment comprises an input sequence of bits;

(ii) identifying a current processing bit length for use in processing a current processing segment of said data to form a processed segment meeting at least one predetermined processing criterion;

(ii) dividing the current processing segment into a plurality of portions wherein each portion has a respective portion bit length equal to said current processing bit length and wherein the bits in each portion are arranged in a respective one of a number of possible permutations;

(iv) assigning a respective label to each of a plurality of said possible permutations; and

(v) forming a processed segment by replacing, within said current processing segment, bit portions comprising bits arranged in one of said plurality of possible permutations with the respective label assigned to that one of said possible permutations;

(vi) identifying a new processing bit length for use in processing a next processing segment of said data to form a processed segment meeting at least one predetermined processing criterion;

(vii) repeating, for each of said plurality of processing segments, steps (ii) to (vi) wherein the new processing bit length is used as the current processing bit length and the next processing segment of said data is used as the current processing segment, and wherein a processing bit length used for at least one of said processing segments of said data is different to a processing bit length used for at least one other of said processing segments of said data.

3. A method according to claim 1 or 2, wherein said predetermined processing criterion comprises whether 50% of the possible permutations which occur in the input sequence of bits occur at least twice as frequently as the other 50% of the possible permutations which occur in the input sequence of bits.

4. A method according to claim 1 or 2, wherein said predetermined processing criterion comprises whether 50% of the possible permutations occur in the input sequence of bits.

5. A method according to any preceding claim, wherein when generating, for each portion permutation, a respective label representing that portion permutation, each label generated has a respective label bit length, and the labels are generated such

that labels generated for portion permutations which occur a greater number of times within said input sequence of bits generally have a smaller label bit length than labels generated for portion permutations which occur a lesser number of times within said input sequence of bits.

6. A method according to any preceding claim, wherein when generating, for each portion permutation, a respective label representing that portion permutation, each label generated has a respective label bit length, and the labels are generated such that at least some of the labels have a label bit length which is smaller than the processing bit length.

7. A method of reconstructing a processed sequence of bits produced by a method according to any preceding claim, the method comprising the steps of:

(i) obtaining extraction information for use in reconstructing an original sequence of bits from said processed sequence of bits;

(ii) reconstructing said original sequence of bits from said processed sequence of bits based on said extraction information.

8. A computer implementable instructions product comprising computer implementable instructions for causing a programmable communications device to perform the method according to any preceding claim.

9. Apparatus for processing data, the apparatus comprising at least one of an electronic circuit, an integrated circuit chip and a computer processor configured to implement the method of any of claims 1 to 7.

Intellectual Property Office

Application No:	GB1618768.4	**Examiner:**	Mr Steven Davies
Claims searched:	1, 3-9	**Date of search:**	21 January 2017

Patents Act 1977: Search Report under Section 17

Documents considered to be relevant:

Category	Relevant to claims	Identity of document and passage or figure of particular relevance
A	-	US7773634 B1 (Machiraju)
A	-	US2002/0021754 A1 (Pian et al)

Categories:

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

Field of Search:

Search of GB, EP, WO & US patent documents classified in the following areas of the UKC^{X} :

Worldwide search of patent documents classified in the following areas of the IPC

H03M; H04L

The following online and other databases have been used in the preparation of this search report

WPI, EPODOC, INSPEC

International Classification:

Subclass	Subgroup	Valid From
H03M	0007/40	01/01/2006
H04L	0009/06	01/01/2006