Software implementations of the Q-Coder

by J. L. Mitchell
W. B. Pennebaker

The Q-Coder is an important new development in arithmetic coding. It combines a simple but efficient arithmetic approximation for the multiply operation, a new formalism which yields optimally efficient hardware and software implementations, and a new technique for estimating symbol probabilities which matches the performance of any method known. This paper describes implementations of the Q-Coder following both the hardware and software paths. Detailed flowcharts are given.

1. Introduction

A new binary arithmetic coding system, the Q-Coder, was developed as a joint effort between the authors of this paper and colleagues at the Almaden site of the IBM Research Division [1]. This paper describes software implementations of the Q-Coder. Companion papers describe in greater detail the probability-estimation technique used in the Q-Coder [2] and the arithmetic coding procedures which allow compatible yet optimal hardware and software structures [3]. The Q-Coder arithmetic coding technique is part of a proposal submitted to the Consultative Committee for International Telephony and Telegraphy (CCITT) and International Standards Organization (ISO) Joint Photographic Experts Group for photographic image compression [4].

This paper assumes a familiarity with the principles of arithmetic coding. For a tutorial on the Q-Coder, see [1]; for a more general tutorial on arithmetic coding, see [5]. Both [1] and [5] discuss the basic principles of arithmetic coding, some approximations to the multiply operation, and the general idea of bit stuffing to avoid carry propagation. Further relevant background information is found in [6].

The basic structure of a compression/decompression system is shown in **Figure 1**. The compression process is divided into three basic parts: a model which converts uncompressed data into binary decisions, a probability estimator, and an arithmetic coder. The decompression process has three similar parts: an arithmetic decoder, the same probability estimator, and the inverse model which converts the binary decisions back into uncompressed data. The dashed boxes enclose the parts of the system comprising the Q-Coder. The model is outside the scope of this paper.

The model in the encoder uses the uncompressed data to determine the state S (used to determine where the probability estimate Q_e for that state is stored) and YN, the binary (yes/no) decision that is to be encoded. These are the inputs to the Q-Coder. The compressed data are output a byte at a time and may be transmitted to a decoder immediately or stored for future use. The Q-decoder examines the compressed data a byte at a time. The inverse model in the decompression system supplies the decoder with exactly the same state S that the encoder used for each decision. The Q-decoder returns the decoded decision YN to the inverse model, which then regenerates the uncompressed data.

The arithmetic coding and decoding are described in detail in Section 2, and the probability-estimation process is explained briefly in Section 3. Section 4 explains how the carry and bit stuffing are handled. Section 5 lists results using the Q-Coder, and Section 6 summarizes the Q-Coder.

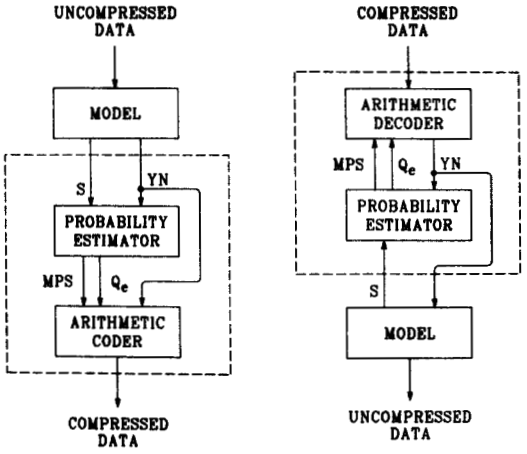


Figure 1

Basic structure of a generic comdec.

Appendix 1 contains detailed descriptions of flowcharts of the software and hardware versions of the Q-Coder/decoder. A test sequence is given in Appendix 2 to provide examples of the operation of both encoders and decoders.

2. Coding/decoding

Two possible arithmetic encoder/decoder structures are illustrated in **Figures 2(a)** and **2(b)**. Figure 2(a) illustrates the concept that the same code string can be constructed in two different ways in the encoder. The dashed line represents the final code string. The solid lines approaching it are the hardware and software code strings for a series of symbols beginning with M, M, L, and M, where M is the more probable symbol (MPS) and L is the less probable symbol (LPS). The hardware code string approaches the final code string from below. It is initialized to zero and always points to the base of the arithmetic coding interval. This interval is subdivided into two parts, with the LPS subinterval beneath the MPS subinterval. The relative size of each subinterval is determined by the estimated LPS probability Q_e and the estimated MPS probability P_e, which is equal to $1 - Q_e$. The hardware code string is shifted up by Q_e for each MPS symbol coded, as shown in the figure; the interval size is reduced to the MPS size P_e as the LPS subinterval is discarded from the full interval. The hardware code string is unchanged for an LPS symbol, but the interval is reduced to the LPS subinterval Q_e.

The software code string starts at the top of the initial interval, $A(0)$, and approaches the final code string from above. Therefore, it can stay unchanged for each M symbol while the range is reduced. For each L symbol the software code string moves down by the fraction of the interval assigned to the MPS; thus, the software code string changes by larger amounts less often. Note how the two code strings converge as the interval size between them decreases. When coding is complete, the software encoder is moved down by the final interval size to reach exactly the same point as the hardware version.

After decoding a given symbol, the decoders must, in general, remove from the code string any subinterval the encoder added to (or subtracted from) the code string. Figure 2(b) illustrates the decoding processes corresponding to the hardware and software encoders. The hardware decoder begins with the final code string and determines whether the LPS subinterval Q_e was added in at the corresponding step in the encoder by comparing the code string remaining at each step with Q_e. If an MPS was encoded, the code string will be at least as large as the Q_e interval. In that case, the MPS is decoded, the code string is reduced by Q_e in order to keep the bottom of the remaining code string positioned at zero, and the range is reduced to the MPS interval size P_e. If the code string is not as large as Q_e, an LPS is decoded and the interval is reduced to Q_e.

To decode a symbol, the software decoder must compare the difference between $A(0)$ and the remaining code string with the MPS interval. The calculation of the difference requires increasing arithmetic precision as decoding progresses. One solution to this problem is to subtract $A(0)$ from the code string at the beginning of the decoding process. The top of the interval, the reference level, is thus shifted to zero, and comparisons can be made between the modified code string and the magnitude of the MPS interval. The software code string is shifted up by P_e for each L symbol decoded, as shown in the figure; the interval size is reduced to the LPS subinterval (Q_e) as the MPS subinterval is discarded. The software code string is unchanged for an M symbol; the interval is reduced to the MPS subinterval (P_e). Note how once again the hardware and software code-string remainders in the decoder converge to the same point, zero in this case.

The following is a simplified mathematical description of the hardware encoder and decoder. Let A denote the present interval on the number line. Let C be a position on the number line that identifies the interval in question. In particular, let C point to the base of that interval. For the approximations used in the Q-Coder, the coding process for a single symbol is as follows:

if MPS is encoded

$$C \leftarrow C + Q_e$$

$$A \leftarrow A - Q_e$$

else (LPS is encoded)

$A \leftarrow Q_e$

end

if $A < 0.75$

renormalize A and C; update Q_e

end

and the matching decoder is

if $C \geq Q_e$

(MPS is decoded)

$C \leftarrow C - Q_e$

$A \leftarrow A - Q_e$

else

(LPS is decoded)

$A \leftarrow Q_e$

end

if $A < 0.75$

renormalize A and C; update Q_e

end

The simplified encoder and decoder described above are ideal for hardware implementation because the interval subtraction and the code-string addition can be done in parallel [1, 7]. However, software implementation is not as efficient because two arithmetic operations are required on the most frequently taken path. Therefore, a more efficient software implementation of the encoder is realized by pointing the code string, C, at the top of the current interval rather than the bottom [3]. C is therefore initialized at $A(0)$, the starting interval value. Then, the encoding process is as follows:

$A \leftarrow A - Q_e$

if LPS is encoded

$C \leftarrow C - A$

$A \leftarrow Q_e$

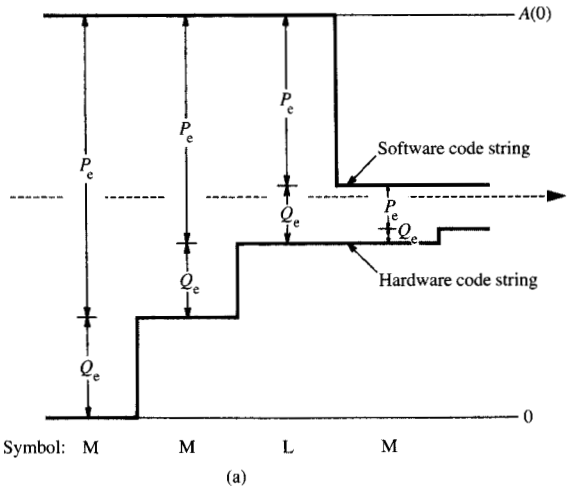
end

if $A < 0.75$

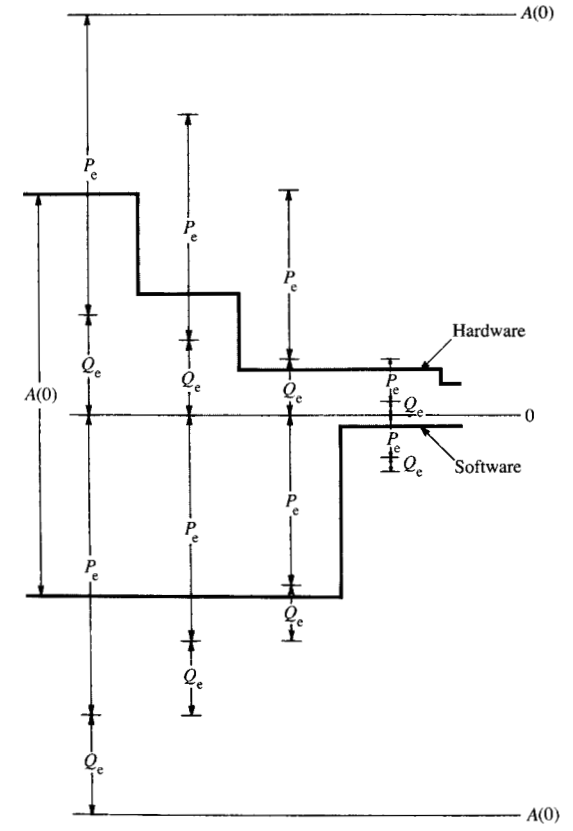
renormalize A and C; update Q_e

end

The software decoder differs from the hardware decoder in that both the interval and code string are initialized at $-A(0)$ in order to prevent problems with arithmetic precision. The



(a)



(b)

Figure 2

Two possible encoder/decoder procedures: (a) software and hardware code strings for the sequence MMLM; (b) software and hardware decoding procedures for the same sequence.

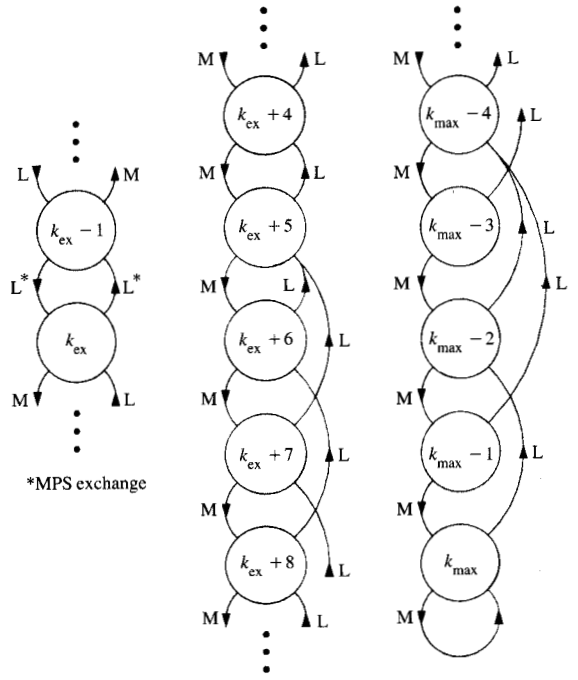


Figure 3

Sections of the state diagram for the probability estimator.

decoding is done, therefore, with negative intervals and negative code-string remainders:

$A \leftarrow A + Q_e$

if $C \geq A$

(MPS is decoded)

else

(LPS is decoded)

$C \leftarrow C - A$

$A \leftarrow -Q_e$

end

if $A > -0.75$

renormalize A and C; update Q_e

end

After the last symbol is encoded, the remaining range is subtracted from the code string to leave the code string pointing at the bottom of the interval. Although the hardware and software implementations are quite different, the two code strings then point precisely to the same point on the number line. The test results in Appendix 2 demonstrate how the two encoder versions generate identical code strings and how the two decoders correctly decode these compressed data.

The probability estimator provides the LPS probabilities and the MPS symbol senses to the arithmetic encoder/decoder. The Q_e values can be selected in advance and kept constant. In such a case the state S selects the desired Q_e for each decision. However, if the Q_e values are allowed to change and adapt to the particular statistics of the decisions being coded, better compression may be obtained over a wider range of data. The probability estimator used in the Q-Coder is adaptive.

3. Probability estimation

In the present coder/adapter system, the probability estimation is derived from the interval renormalization. The principles of this estimation process are described in [1, 2]. The key aspect, in terms of software efficiency, is the adjusting of the probability estimate only when the A register is shifted. When the A register is shifted, the inner loop of the software must be left, because bits are shifted into/out of the decoder/coder. Therefore, adjusting the probability estimate adds no overhead to the inner loop.

The estimator can be defined as a finite-state machine, that is, a table of Q_e values and associated next states for each type of renormalization (i.e., new table positions). The rate of change of Q_e is determined by the granularity of the table of Q_e values and by the new state associated with each Q_e value for the two types of renormalization. **Figure 3** diagrams sections of the actual finite-state machine used to estimate the probabilities. The leftmost section illustrates the exchange of MPS and LPS definitions at $Q_e \cong 0.5$ (k_{ex} is defined to be the index where this exchange occurs). As the LPS goes from k_{ex} to $k_{ex} - 1$, the LPS and MPS senses are reversed, as indicated by the asterisk. The center section shows a region where the finite-state machine changes from a single-state jump on LPS to a double-state jump. Some parts of the finite-state machine require a jump of more than one state in order to correctly estimate the probability. The rightmost section shows the diagram for the smallest values of Q_e. This last section shows how the transition at the MPS renormalization for the smallest Q_e value is returned to that state.

Figure 4 illustrates the sequencing of the probability estimator for an LPS followed by a sequence of MPSs. The ordinate shows interval (A-register) values, and the abscissa shows the discrete allowed values of Q_e. Solid lines indicate changes to the interval resulting from coding operations; dashed lines represent changes resulting from

renormalizations. The initial LPS renormalization (marked with an asterisk) causes a transition to a known A-register value and a known state in the finite-state machine (in this case, from a Q_e of 0.42206 to the appropriate starting A-register value at $Q_e = 0.46893$. (The particular Q_e values in the figure are scaled to a decimal representation from the actual 5-bit Q_e table.) As MPSs are coded, the interval decays until it drops below 0.75. At that point a transition is made to a smaller Q_e, and the interval is renormalized by doubling until it is greater than 0.75. In most cases only one doubling is needed. Thus, the pair of doublings shown at $Q_e = 0.32831$ is the exception rather than the rule. Whether one or two doublings occur is of no consequence for the probability estimation.

The Q_e values for each Q_e index are chosen to have mostly bit values of 0, except that the last bit is always a 1. This strategy simplifies the hardware implementation in the sense that each bit value of 1 (except the last) requires additional wiring and circuits. Also, in order to balance the Q_e values with the adaptation rule, several strategies may apply. For example, one may have fewer entries in the table (fewer indexes) by reducing the size of the index change following a renormalization. With more table entries, the adaptation may be faster if the index change is larger. Correct estimation requires the proper balancing of movement to larger and smaller Q_e indexes following renormalizations. The use of estimated Q_e values which renormalize to the minimum allowed value for the A register should be avoided, unless the index change is such that a single MPS renormalization cannot return the adapter to that Q_e value. In the given table, for simplicity the strategy is to always increase the Q_e index by one upon an MPS renormalization. An alternative strategy could be to always decrease the Q_e index by one upon an LPS renormalization. However, since fractional index change is not allowed, correct balance of the estimator could only be achieved either by conditional index change by means external to the table, or by replication of Q_e values in the table.

The above simplified description of the coding/decoding process shows renormalization every time the A register drops below 0.75. Thus, the normalized range is maintained in the interval from 0.75 to 1.5. This keeps A centered around 1.0 so that the arithmetic approximations are reasonably good. For ease of implementation in hardware, it is preferred that the test for renormalization be done on the most significant bit of A. Therefore, an integer representation is used in which X'1000' is equivalent to decimal 0.75 [1]. Thus, in the table of Q_e values X'0AC1' is a probability of about 0.5.

Table 1 gives the information necessary to perform the probability estimation. For each Q_e value in the table, an LPS causes the Q_e index to be decremented by the amount indicated in the "Decr LPS" column. An MPS renormalization increments the Q_e index by one unless the

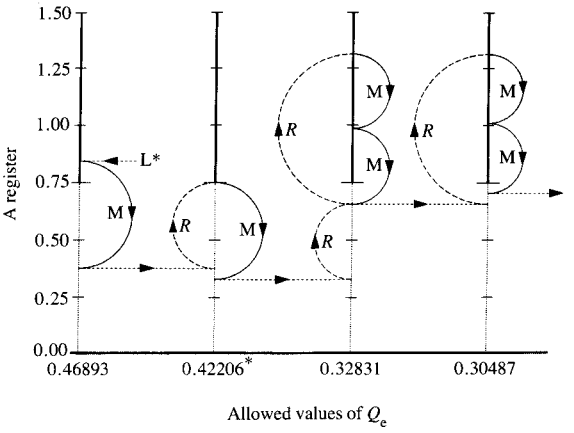


Figure 4

Examples of the sequence of probability estimation for an LPS followed by a sequence of MPS symbols.

Table 1 Probability estimation process.

Q_e value	Decr LPS	Incr MPS	MPSexch flag	Q_e index	Q_e value (binary)			Q_e value (decimal)
X'0AC1'	0	1	1	0	1010	1100	0001	0.50409
X'0A81'	1	1	0	1	1010	1000	0001	0.49237
X'0A01'	1	1	0	2	1010	0000	0001	0.46893
X'0901'	1	1	0	3	1001	0000	0001	0.42206
X'0701'	1	1	0	4	0111	0000	0001	0.32831
X'0681'	1	1	0	5	0110	1000	0001	0.30487
X'0601'	1	1	0	6	0110	0000	0001	0.28143
X'0501'	2	1	0	7	0101	0000	0001	0.23456
X'0481'	2	1	0	8	0100	1000	0001	0.21112
X'0441'	2	1	0	9	0100	0100	0001	0.19940
X'0381'	2	1	0	10	0011	1000	0001	0.16425
X'0301'	2	1	0	11	0011	0000	0001	0.14081
X'02C1'	2	1	0	12	0010	1100	0001	0.12909
X'0281'	2	1	0	13	0010	1000	0001	0.11737
X'0241'	2	1	0	14	0010	0100	0001	0.10565
X'0181'	2	1	0	15	0001	1000	0001	0.07050
X'0121'	2	1	0	16	0001	0010	0001	0.05292
X'00E1'	2	1	0	17	0000	1110	0001	0.04120
X'00A1'	2	1	0	18	0000	1010	0001	0.02948
X'0071'	2	1	0	19	0000	0111	0001	0.02069
X'0059'	2	1	0	20	0000	0101	1001	0.01630
X'0053'	2	1	0	21	0000	0101	0011	0.01520
X'0027'	2	1	0	22	0000	0010	0111	0.00714
X'0017'	2	1	0	23	0000	0001	0111	0.00421
X'0013'	3	1	0	24	0000	0001	0011	0.00348
X'000B'	2	1	0	25	0000	0000	1011	0.00201
X'0007'	3	1	0	26	0000	0000	0111	0.00128
X'0005'	2	1	0	27	0000	0000	0101	0.00092
X'0003'	3	1	0	28	0000	0000	0011	0.00055
X'0001'	2	0	0	29	0000	0000	0001	0.00018

Q_e index has reached the maximum value, as shown in the "Incr MPS" column. An LPS from the top entry causes an exchange in the sense of the MPS and LPS, as indicated by the "MPSexch flag." The Q_e values as binary bits are followed by the decimal form. The decimal fraction 0.75 was chosen to correspond to X'1000'; this determines the scaling in converting the first column to the last column.

4. Bit stuffing in the Q-Coder

The general principle of bit stuffing is reviewed in [8]. The version adopted for the Q-Coder is a composite of techniques used by Langdon and Rissanen [6] and Goertzel and Mitchell [9].

Carry propagation can only occur through a sequence of consecutive 1-bits. In principle, the sequence of 1s can be as long as the code string itself. To avoid arbitrary propagation of the carry, sequences of 1s are periodically interrupted by inserting or stuffing 0-bits. In the Q-Coder the bit stuffing is done only on byte boundaries. One 0 is stuffed into the high-order bit of the byte immediately following any byte which is all 1s (X'FF'). This 0-bit is the receiver for any carry which might subsequently occur.

One point which must be resolved is the number of carries that can occur in creating one byte of compressed data. Indeed, if the new byte of coded data is taken from bits 20 to 12 (using a numbering convention that bit 0 is the low-order bit and bit 31 is the high-order bit), more than one carry can occur. As a means of avoiding the double carry, Langdon [10] suggested leaving a spacer bit, and taking the byte from bits 21 to 13. This spacer bit guarantees that only one carry bit can be created in creating a single byte of coded data. For the integer representation giving $1 \le A < 2$, the upper bound on $C[0]$, the part of the code string remaining in the C register after removing a byte of coded data, is $C[0] < 2^s$, where s is the number of spacer bits.

The code string cannot be increased by more than A, as that is the probability interval remaining. Therefore, after eight renormalization shifts, the maximum value of the sum $A[8] + C[8]$ is bounded by $A[8] + C[8] \le 2^8(A[0] + C[0])$ for any possible symbol sequence. Given the bounds on $C[0]$ and $A[0]$, and given that $A[8] > 0$, $C[8] < 2^8(2 + 2^s)$, if only one carry bit is allowed in the C register, the condition $C[8] < 2^{(9+s)}$ must hold. If $s = 1$, both conditions are satisfied. Therefore, a single spacer bit is sufficient to prevent the double carry. However, taking the byte out of bits 21 to 13, as required if one spacer bit is used, can be awkward in software if C is a memory-mapped register. It is much more convenient to normally have the output byte-aligned in bits 23 to 16. This can be accomplished by using four spacer bits instead of just one spacer bit. For $s = 4$, the above relationship shows that any two-byte pattern in the range X'FF90' to X'FFFF' is illegal. These patterns are therefore available for use in inserting control characters in the compressed data string [10]. Furthermore, any byte with a stuff bit cannot be X'FF', which guarantees that any byte containing a stuff bit cannot be followed by another byte which contains a stuff bit. Thus, no more than one stuff bit per 15 code bits is possible.

The diagram below shows the 12 "x" bits to the right of a binary point aligned* with the fractional bits in A, the 4 "s" spacer bits, and the 8 "b" bits which will be output as a byte when the flag bit which starts in bit position 24 is shifted out of the register. A carry will sometimes be shifted into bit 24. If it is preceded by an X'FF' byte, the carry will be the leading bit of the output byte and will be followed by the next seven "b" bits. Otherwise, the carry is added to the preceding byte.

```
Encoder:31-24, 23 - 16, 15 - 8, 7 - 0        bits
        ffffffff, bbbbbbbb, ssss.xxxx, xxxxxxxx
                          C register          (4 bytes)
                            0001.aaaa, aaaaaaaa
                                  A register  (2 bytes)

Decoder: 31       -       16, 15 - 8, 7 - 0
        0000.xxxx, xxxxxxxx, nnnnnnnn, ffffffff
                          C register          (4 bytes)
                        Cx  ,                 (2 bytes)
                            ,  Cn  ,  Cf      (1 byte)
        0001.aaaa, aaaaaaaa
             A register                       (2 bytes)
```

For the decoder each new byte must be inserted to the right of the bits which are aligned with the A register. The diagram above illustrates the convention used in the decoder. The four highest-order bits of C will always be 0. For convenient comparison to the encoder structure, a binary point (the equivalent in binary arithmetic to the decimal point in decimal arithmetic) is defined just to the right of them. The next twelve bits are used in all comparisons and calculations with the A register. These high 16 bits are identified as a two-byte value called C_x. The new compressed data byte is inserted into the C register in bits 15 to 8 (byte C_n). The low-order byte, C_f, has a flag bit which normally starts in bit 0. After eight shifts it will be shifted out of C_f, indicating that a new byte is needed. This is equivalent to having an independent counter keep track of when a new byte is needed. The encoder uses its most significant bits (flagged with 'f') for a similar purpose of keeping track of when a byte is ready to be put into the code buffer.

If the carry (hardware) and borrow (software) are handled such that no more than the immediately preceding byte in the code string can be affected by the current coding operations, the two encoders generate identical code strings.

In the software encoder, at the start of a new byte of coded data a borrow bit is set in bit 23 of C. If the borrow bit is used, a bit is borrowed from the preceding byte in the code string. This leads to a problem with borrow propagation,

* Note that the binary value 1.000000000000 corresponds to the decimal fraction 0.75. As noted before, this representation was adopted to simplify hardware implementation [1].

which is analogous to carry propagation in the hardware encoder. To ensure that a borrow can be done, each time a byte is about to be placed in the code string, the current value of C is compared to the A register. If C is smaller than A, the byte being placed in the code string is zero and, in addition, a borrow may be needed. This test detects the condition where the software C register contains an X'00' and the hardware C register contains an X'FF' value for the byte being transferred to the code string. In addition, it detects a condition for which, if the software X'00' byte were simply placed in the code string, a borrow might be required from it which would propagate more than one byte. The borrow is instead taken immediately from the preceding byte, and the current byte in the code string is set to X'FF'. Just as in the hardware encoder, whenever an X'FF' is generated, the next byte must have a stuffed bit. In hardware the stuffed bit is a 0-bit; in software it is a 1-bit. As coding proceeds, either a carry will occur in hardware or a borrow will occur in software. At that point the two code strings become identical.

In principle, the C register containing the newest part of the code string must have 12 bits reserved for the fractional part of C (Q_e has 12-bit precision). Bits 31 to 24 are set to binary 00000001 at the start of each new byte of coded data; when the register sign bit (31) is set, the next shift of the register completes the byte of coded data. The remaining bits in the register, bits 23 to 12, must hold up to one byte of coded data and any carries.

5. Coding results

Table 2 gives some test results for some sample files. The gray-scale images all use the model described in [11] encoded with the Q-Coder. The facsimile and digital halftone files use the 7-pel predictor described in [6]. The column labeled "Bits" represents a complete bit count, including the overhead for carries and flushing the final bits out of the coder. The "Acnt" column only records the number of renormalizations of the A register. The entropy was calculated as a global static value for the relative frequency of 0s and 1s for each conditioning state for each file.

6. Summary

This paper describes the Q-Coder system, with an emphasis on the relationship between hardware and software implementations. The Q-Coder has been designed for efficient implementation both in hardware and software. It incorporates a robust optimized arithmetic coder with a probability estimator that requires only six bits of storage per state context, and which adjusts the probability estimates only when renormalization of the current interval is needed. It provides for two compatible implementations, one suitable for serial software structures, the other for parallel hardware structures. Carry and borrow propagation are treated in a systematic way such that software and hardware implementations produce identical code strings. Bit stuffing is on byte boundaries, and four spacer bits in the code register guarantee that no more than one carry can occur in a single byte of compressed data. In software, use of a flag bit in the code register allows for a simple implementation of bit stuffing on byte boundaries.

Table 2 Q-Coder coding performance.

File	*Q-Coder*		*Entropy* (bits)	*Comments*
	Bits	*Acnt*		
Gray-scale, teleconferencing model (512 × 480 × 8-bit images)				
t2	171 856	171 756	166 373	teleconferencing scene
courierf	94 432	94 370	95 327	courier text, filtered
courier	171 648	171 553	170 335	courier text
ieeef	175 536	175 445	173 443	IEEE chart, filtered
ieee	232 952	232 819	228 819	IEEE chart
handwrtf	98 832	98 760	97 563	handwriting, filtered
topletf	101 784	101 722	101 272	top of memo, filtered
densetf	248 232	248 096	245 042	dense text, filtered
marcosf	110 536	110 471	108 278	child's face, filtered
atomsf	171 408	171 306	164 753	diagram, filtered
fruitf	141 248	141 171	137 530	fruit, filtered
t4f	237 560	237 440	229 059	TEM photo, filtered
t5f	164 112	164 014	156 253	TEM photo, filtered
total:	2 120 136	2 118 923	2 074 047	
Facsimile, 7-pel predictor (1728 × 2376 × 1)				
CCITT 1	119 752	119 630	130 623	business letter
CCITT 2	71 568	71 497	71 884	circuit drawing
CCITT 3	187 728	187 574	200 927	French invoice
CCITT 4	446 816	446 522	494 249	dense text
CCITT 5	215 888	215 737	230 345	math book page
CCITT 6	112 256	112 151	117 002	graph
CCITT 7	468 232	467 978	465 156	Kanji
CCITT 8	124 768	124 667	131 735	memo and sign
total:	1 747 008	1 745 756	1 841 921	
Digital halftone, 7-pel predictor (various sizes)				
budking	966 480	965 195	1 267 429	clown (1976 × 2480 × 1)
boat2x	154 744	154 560	153 029	boats (768 × 488 × 1)
jphmesh	184 016	183 397	257 665	face (672 × 864 × 1)
total:	1 305 240	1 303 152	1 678 123	

Appendix 1: Description of flowcharts of the Q-Coder

Figures 5 through 25 detail the operation of hardware and software encoders and decoders. In some cases the same procedure is used for both hardware and software. In other cases parallel procedures are shown and labeled "-H" for hardware and "-S" for software. The hardware version may be somewhat simpler to implement and debug because the encoder and decoder A registers are identical at each step. Once correctly implemented, the changes required to convert to the more computationally efficient software version are minor.

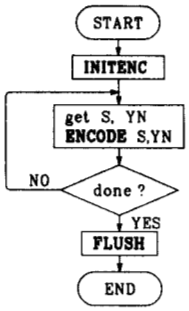


Figure 5

Coder for an arithmetic-coding comdec.

Figure 5 is a flowchart showing an encoder in an arithmetic coding compression system. Figures 6 to 19 show details for some of the functional blocks in Figure 5. Figure 20 is a flowchart showing a decoder in an arithmetic coding decompression system. Figures 21 to 25 show the more detailed functional blocks for the Q-decoder.

Definitions used in the flowcharts

S is the state generated by the model. The same state must be used for both the encoder and decoder for a given decision.

YN is the binary decision which is coded (0/1). In the encoder it is determined by the model. In the decoder, it is the output from the Q-decoder.

LEN is the length of the buffer for the code string. It is set to 256 bytes (an arbitrary but convenient choice).

BPST points to the start of the compressed data buffer.

BE points to the first byte beyond the compressed data buffer. When the pointer to the compressed data reaches BE, a new compressed data buffer is needed.

BP is the pointer to the current byte of compressed data.

B is the byte of compressed data pointed to by BP.

A is a 16-bit integer, but can be considered a binary fraction with the binary point positioned to provide 12 fractional bits and 4 integer bits.

A_{min} determines when renormalization is needed. It is set to X′1000′ (which is equivalent to 0.75).

Q_e(S) is defined as a fixed-point fraction with 12 significant bits. It is the current estimate of the less probable symbol probability for state S.

C contains the least significant bits of the compressed data string for the encoder. In the decoder C contains the most significant bits.

C_x is the 16 most significant bits of C in the decoder.

C_n is the byte representing bits 8–15 of C in the decoder.

C_f is the 8 least significant bits of C in the decoder.

MPS(S) is either a 0 or 1, the sense of the more probable symbol for the given state.

LPS(S) is either a 0 or 1, the sense of the less probable symbol for the given state. It must be saved for each state or derived by inverting MPS(S).

CT is the number of extra bits which must be flushed to guarantee that the decoder has enough bits to complete decoding of all decisions.

Detailed description of encoder operation

Referring back to Figure 1, the encoder model uses the uncompressed data to generate a sequence of states S and yes/no decisions *YN* which are input to the Q-encoder (shown in the dashed box). For each decision, the S is used to look up the MPS sense and less probable symbol probability Q_e. The arithmetic coder codes the *YN* decision using this information. The flowcharts show these processes in greater detail.

Figure 5 illustrates how after initializing the encoder, the *YN* decisions for the model-generated states S are coded one at a time until the coding is done. The modeling process is represented by the statement "get S, *YN*." The decision as to when all symbols have been encoded is provided by some external means. For example, for gray-scale TV images there is a fixed format such as 512 pels per line by 480 lines. If there is no agreed-upon convention, the encoder must supply the decoder with that information, either externally or as part of the compressed data string. When all symbols have been encoded, the final bytes are flushed out so that the decoder is guaranteed to have enough data to completely decode the decisions.

INITENC [**Figures 6(a)** and **6(b)**] does the initialization for the encoder. First the tables for the probability estimator are set up using the data in Table 1, and the storage area for the probability estimates is initialized. Our convention is to start each state with the largest Q_e (X′0AC1′) and MPS equal to 0. Both hardware and software versions initialize LEN to 256 bytes, point BE to the end of the compressed data buffer, and point BP to 1 byte before BPST, the actual start of the buffer to be sent. The pointer is updated before a byte is written; hence an offset of 1 is necessary. For the hardware version the byte B (addressed by BP) is initialized to X′80′ to guarantee that the special case of B = X′FF′ will not be triggered for the first byte in the compressed data string. For the software version this byte is incremented by 1 because C is initialized such that a bit will always be borrowed from this byte. The interval A is initialized to

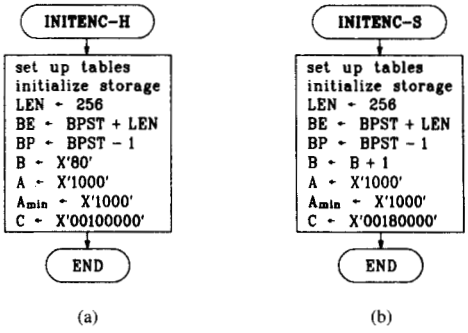


Figure 6

Initialization routines for the compression-system encoder: (a) hardware procedure; (b) software procedure.

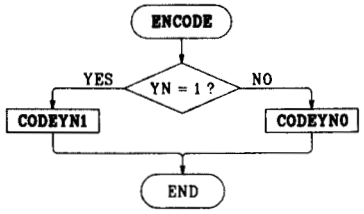
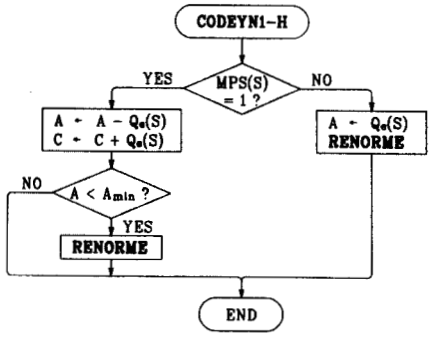


Figure 7

Generation of compressed data string.



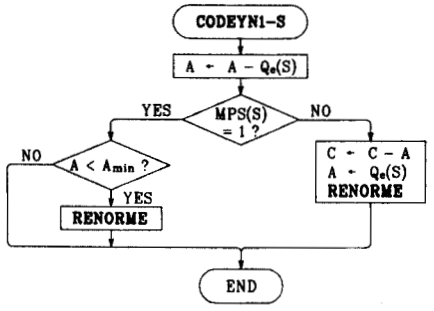


Figure 8

Encoding for $YN = 1$: (a) hardware; (b) software.

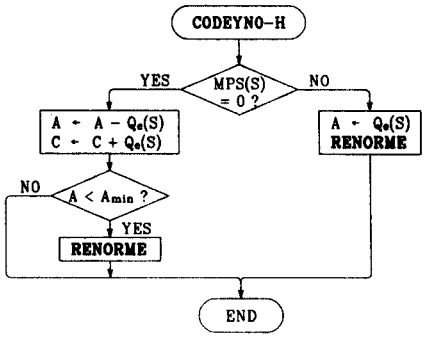
X'1000' and A_{min} is initialized to the same value. For the initialization of C both versions have the 12th most significant bit set to 1 to flag when 8 compressed bits are ready. The software version has the preborrow bit inserted just after the flag.

ENCODE (**Figure 7**) shows the two paths taken according to whether YN is 1 or 0.

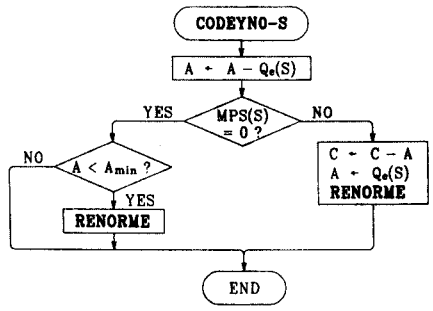
CODEYN1 [**Figures 8(a)** and **8(b)**] codes $YN = 1$. The hardware encoder [Figure 8(a)] first tests to determine whether MPS(S) = 1, in which case an MPS must be coded. A is decreased by subtracting Q_e(S) and C is increased by adding Q_e(S). If A is less than A_{min}, both A and C must be renormalized. If MPS(S) = 0, an LPS is coded by setting the interval to the Q_e(S); renormalization is always required for an LPS since Q_e is always less than A_{min}. For the software encoder [Figure 8(b)] the new MPS interval is calculated immediately. If an MPS is to be encoded, the revised interval is tested to determine whether renormalization is required. No other action is necessary. If an LPS is to be encoded, the revised interval is subtracted from C; the interval is reset to the LPS probability. Renormalization is always required.

CODEYN0 [**Figures 9(a)** and **9(b)**] shows the same operations as Figure 8 for the $YN = 0$ path. In this case, the MPS test is made with 0 instead of 1. Otherwise it is identical to the CODEYN1 procedure. Figures 8 and 9 could have been coalesced into one figure by comparing MPS(S) to YN. However, in practice the symbol to be encoded is often known, so it is convenient to be able to specify CODEYN0 or CODEYN1.

RENORME (**Figure 10**) renormalizes the A and C values one bit at a time. A is shifted first and then C is tested to see if the most significant bit is set. If so, the next shift of C



(a)



(b)

Figure 9

Encoding for $YN = 0$: (a) hardware; (b) software.

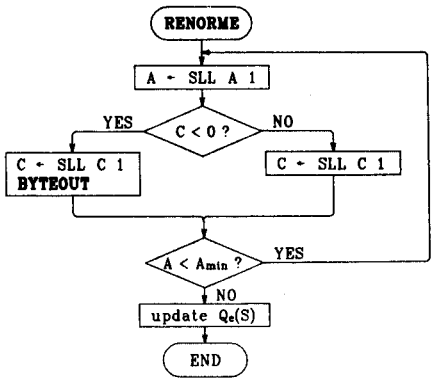


Figure 10

Encoder renormalization.

removes that flag bit and a byte is output. Otherwise C is just shifted one bit. This process is continued as long as A is less than A_{min}. The "update Q_e(S)" process is not shown. The Q_e value will be increased for an LPS renormalization and decreased for an MPS renormalization. Table 1 gives the necessary information to determine how to update the estimated probability Q_e. Different procedures must be used for MPS = YN and MPS ≠ YN.

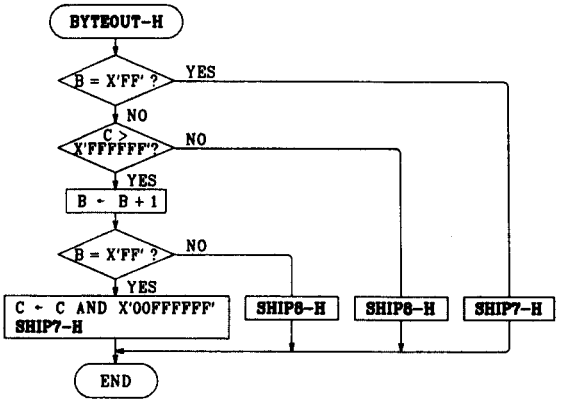
Figures 11(a) and **11(b)** show the procedure for moving a byte out of the C register into the compressed data string. The decoder expects every X′FF′ byte to be followed immediately by a leading stuffed bit in the next byte; if set, it contains the carry bit. The encoder must maintain this convention.

In Figure 11(a) the hardware version of BYTEOUT first looks at the last byte B and immediately outputs only 7 data bits if B is X′FF′. Any carry will appear in the most significant bit of the new byte. If B is less than X′FF′, C is tested for a carry, and if there is none, 8 bits can be output. If there is a carry, the last byte needs to be incremented by 1 and the result tested to see if it is now X′FF′. If so, then the carry in C which has already been added to B must be cleared before outputting the next 7 bits. Otherwise, 8 bits may be output into the new byte.

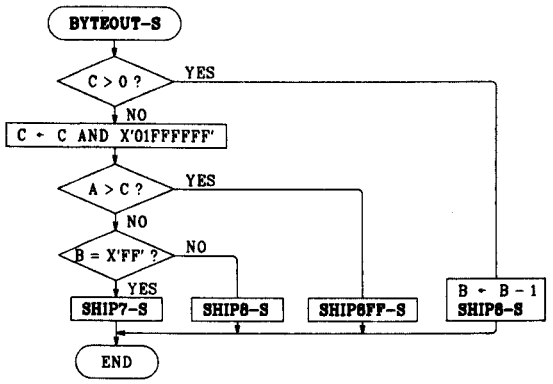
The software version BYTEOUT-S [Figure 11(b)] tests the sign of C. If positive, the borrow bit is zero (a borrow has occurred) and B must be decremented by 1 before outputting 8 bits. If the borrow bit is still set, it is cleared from C before A is compared to C. If C is smaller than A, a borrow could be needed in the future that would not be available if the new byte were output as X′00′. (A is at most X′1FFF′, so C has only zeros in the 8 output bits.) In this case SHIP8FF-S (Figure 14) does the preborrow, converts the new byte to X′FF′, and saves the borrowed bit into C. If B is X′FF′, then the 8 bits which are transferred from the C register are shifted to include the borrow bit plus 7 bits of data.

SHIP8 **[Figures 12(a)** and **12(b)]** is similar for both hardware and software versions. After the output byte pointer is incremented in NEXTBYTE (Figure 15), the 8 bits in C at bits 23–16 are stored as B. All but the 16 least significant bits are cleared in C and the flag is inserted at the 8th most significant bit. The software version also inserts a borrow bit after the flag.

SHIP7-H **[Figure 13(a)]** is similar to SHIP8-H except that only 7 bits and possibly a carry are removed from C. After incrementing the output byte pointer to the next output byte B, bits 24–17 of C are stored in the new B. The leading bit contains any carry. Only the trailing 17 bits are left in C before the flag is inserted at the 7th most significant bit. This



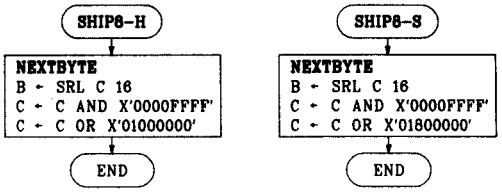
(a)



(b)

Figure 11

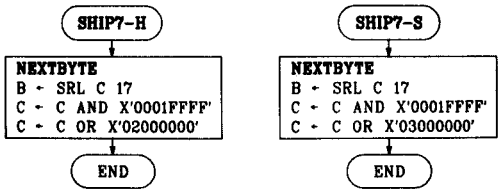
Procedure for shifting a byte of data from the C register to the compressed data string: (a) hardware; (b) software.



(a) (b)

Figure 12

Procedure for adding eight bits from the C register to the compressed data string: (a) hardware; (b) software.



(a) (b)

Figure 13

Procedure for adding seven bits from the C register plus one stuffed bit to the compressed data string: (a) hardware; (b) software.

causes the next byte to be output when 7 new bits are ready because 1 bit has been left in *C*. SHIP7-S **[Figure 13(b)]** is the same as SHIP7-H except that the borrow bit is set to follow the flag bit immediately.

Since the software encoder must guarantee that future values of B can be decremented if necessary, SHIP8FF-S **(Figure 14)** always decrements B before storing X'FF' in the next byte. The extra borrow is inserted into *C*, where it will be output in the next byte as a carry if it is not needed.

NEXTBYTE **(Figure 15)** moves BP to address the next byte in the compressed data buffer. If, after it is incremented, BP is not less than the end of the buffer, the buffer must be transferred and BP reset to the start of the buffer. It is

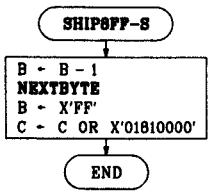


Figure 14

Procedure for adding eight bits from the C register to the compressed data string with preborrow (software only).

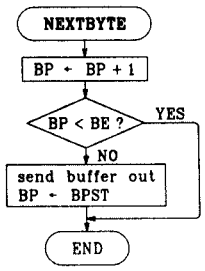


Figure 15

Compressed data buffer management for the encoder.

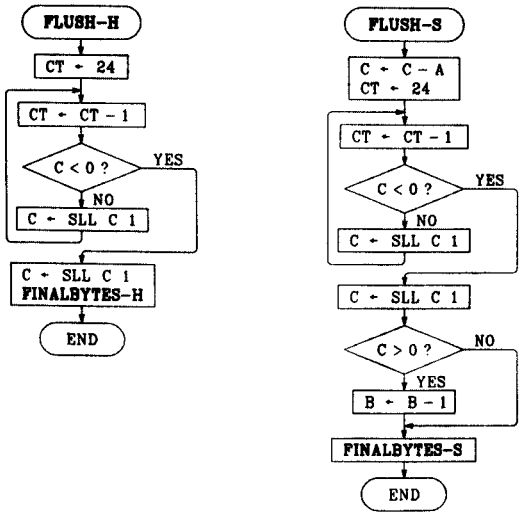


Figure 16

Procedure for flushing the final bits of data from the C register to the compressed data string: (a) hardware; (b) software.

assumed that BPST and BE will be appropriately changed as part of the "send buffer out" operation, if necessary.

After the final decision has been coded, the 24 compressed data bits still in C need to be flushed out. In FLUSH-H **[Figure 16(a)]** a counter CT is initialized to 24 and decremented for each shift in C until the flag is in the most significant bit. One more shift puts the compressed data bits on a byte boundary. Then FINALBYTES-H **[Figure 17(a)]** can output these last bytes.

FLUSH-S **[Figure 16(b)]** moves C to the bottom of the final coding interval, which positions it precisely to the value generated by the hardware version. After the bits have been byte-aligned, if the borrow has been used the last byte must be decremented before the final bytes are output.

FINALBYTES-H [Figure 17(a)] goes through the same type of operations as BYTEOUT-H [Figure 11(a)] within a loop until all bits have been flushed out. The blocks FLUSH7-H and FLUSH8-H include appropriate decrements of CT by 7 and 8 bits, respectively. When completed, BP is incremented past the last byte stored and the final buffer can be sent out.

The software version of FINALBYTES-S **[Figure 17(b)]** only determines whether to ship 7 or 8 bits according to whether the preceding byte is X'FF'. The preborrow was already handled in FLUSH-S. Since C was moved to the bottom of the interval, the comparison of A and C in BYTEOUT-S is irrelevant.

In FLUSH7 (**Figure 18**) 7 bits are output for both the hardware and software versions by pointing to the new byte, storing bits 24–17, saving only the 17 least significant bits of C, shifting the bits in C up by 7, and decrementing CT by 7.

In FLUSH8 (**Figure 19**) 8 bits are output for both the hardware and software versions by pointing to the new byte, storing bits 23–16, saving only the 16 least significant bits of C, shifting the bits in C up by 8, and decrementing CT by 8.

Detailed description of decoder operation

Referring back to Figure 1, the decoder uses the compressed data to decode a sequence of decisions *YN* based on the model-generated sequence of states S as indicated in the right-hand diagram. As in the encoder, for each decision, the S is used to look up the MPS sense and less probable symbol probability Q_e.

Figure 20 illustrates how after initializing the decoder, the *YN* decisions for the model-generated states S are decoded one at a time until the decoding is done. The modeling process is represented by the statement "get S." The decision as to when all symbols have been decoded is provided by some external means.

INITDEC **[Figures 21(a)** and **21(b)]** does the initialization for the decoder. After the tables are set up, all states are initialized as in the encoder. Both versions start by getting a new buffer of compressed data. This is assumed to initialize BPST and LEN. BP is pointed to the start of the compressed data buffer (BPST) and BE is set to address the end of the buffer. In the hardware version the interval A and the value A_{min} are initialized to X'1000' to match the encoder. In the software version they are both set to negative X'1000' because the decoding operations will be done in the negative domain. C is initialized with the first two bytes of the

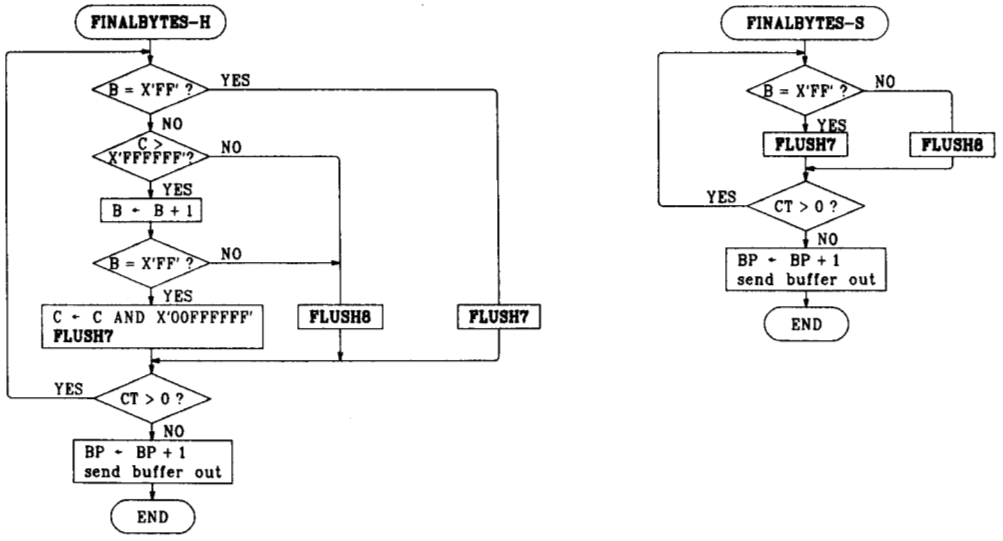


Figure 17

Adding the final compressed data bytes to the compressed data string: (a) hardware; (b) software.

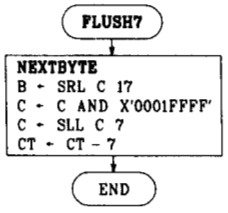


Figure 18

Adding seven bits of data plus one stuffed bit to the compressed data string during the final flushing of the C register.

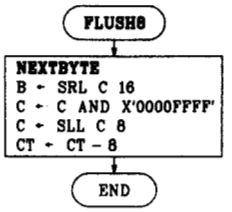


Figure 19

Adding eight bits of data to the compressed data string during the final flushing of the C register.

compressed data buffer. The first byte is shifted into bit positions 23–16, the pointer BP is incremented, and the second byte positioned in C by BYTEIN (Figure 24). Then C is shifted left 4 bits to align it with A. The decoding process only looks at the bits in C_x, the high two bytes of C (bits 31–16). In the software version a negative A is added to C to shift the code point below zero.

DECODE **[Figures 22(a) and 22(b)]** shows the procedure for decoding one *YN* decision. In the hardware version, if C_x is greater than or equal to Q_e(S), the MPS(S) path is

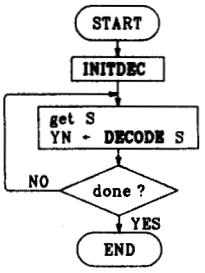
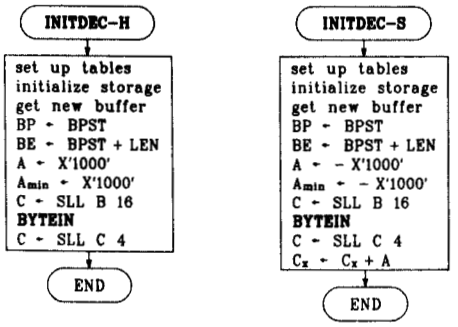


Figure 20

Decoder for an arithmetic-coding comdec.



(a) (b)

Figure 21

Initialization routines for the compression-system decoder: (a) hardware procedure; (b) software procedure.

followed. *YN* is set to MPS(S) and C_x and A are decremented by Q_e(S). A comparison of A with A_{min} determines whether renormalization is necessary. On the LPS path, *YN* is set to LPS(S) and A is set to the LPS interval size Q_e(S). Renormalization is always required. In the software version, Q_e(S) is added to (negative) A to decrease the interval to the MPS interval size. If C_x is greater than or equal to the revised A, the MPS path is followed. *YN* is set to MPS(S), and a comparison of A with A_{min} determines whether renormalization is necessary. On the LPS path, *YN* is set to LPS(S), C_x is increased (moved toward zero) by subtracting the (negative) A, and A is set to negative Q_e(S). Renormalization is always required. Note that A, A_{min}, and C_x are all negative throughout this process for the software version.

RENORMD **[Figures 23(a) and 23(b)]** renormalizes the A and C values one bit at a time. C_f, the least significant byte of C, is tested to see if any bit is set (if so, the flag bit has not yet been shifted out). If C_f is 0, it is time to get a new byte. Then, A and C are shifted. This process is continued as long as A is less than A_{min} for the hardware and as long as the (negative) A is greater than the (negative) A_{min} for the software. The "update Q_e(S)" is identical to the probability estimation update done in the encoder.

During the process of moving a new byte into C as shown in BYTEIN **(Figure 24)**, it is necessary first to test the last byte B to see if it was an X'FF' byte. If not, the new byte is inserted in C_n (bits 15–8 of C) and a flag bit is set as the least significant bit of C_f. If the last byte was an X'FF', the leading bit in the next byte was inserted during encoding and must be appropriately accounted for during decoding. In this case, C_f is set to a 2 to position the flag with a 1-bit shift. C_n is set to zero to clear the old flag bit. Then the new byte (which would normally be placed in C_n) is shifted up an extra bit and added to C.

GETBYTE **(Figure 25)** moves BP to address the next byte in the compressed data buffer. If, after it is incremented, BP is not less than the end of the buffer, a new buffer must be obtained and BP reset to the start of the buffer. It is assumed that BPST and BE will be appropriately changed if necessary.

Appendix 2: Test sequence for a small data set

The test file was generated using a random number generator for 256 binary decisions. The number of 1s in the file was 40. In the table the event count, EC, is listed first, followed by the value of Q_e upon completion of coding that event. The decision *YN* encoded (or decoded) is listed next. The A register and C register are listed as they are upon completion of that event. Note that the A register is always greater than X'0FFF'. The number of shifts during renormalizations is given under "bits" and does not include stuff bits.

In the following encoder tests, the "codebytes" are listed as they are output. Two bytes in that column list both a changed preceding byte and the new byte.

Test data (in hexadecimal form): 00020051 00000000 A0C02000 00094242 42023029 90311A00 10006040 821000C0

For this file the coded bit count is 192, including the overhead to flush the final data. The actual compressed data string for both encoders is (in hexadecimal form): FF390252 8116303C ED8E4008 C8D713A7 97D99694 8E3BB2C0

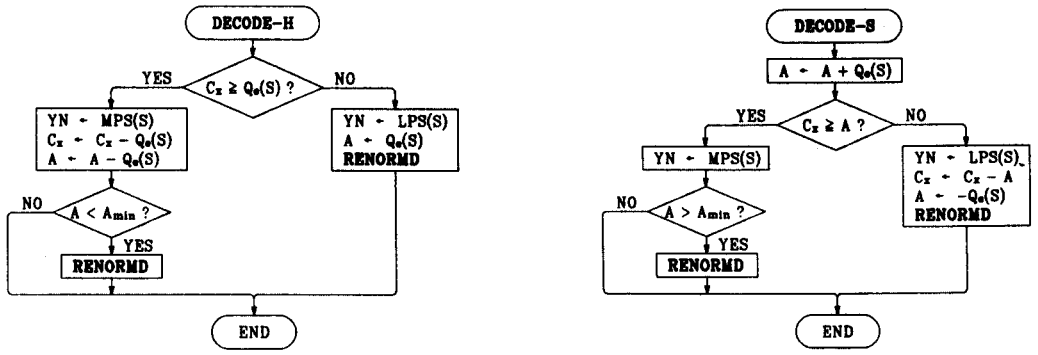


Figure 22

Decoding one decision: (a) hardware; (b) software.

Hardware encoder:

EC	Q_e	YN	A	C	bits	codebytes
0	0AC1		00001000	00100000	0	
1	0A81	0	000014FC	00402B04	2	
2	0A01	0	000014F6	00806B0A	3	
3	0901	0	000015EA	0100EA16	4	
4	0701	0	000019D2	0201E62E	5	
5	0701	0	000012D1	0201ED2F	5	
6	0681	0	000017A0	0403E860	6	
7	0681	0	0000111F	0403EEE1	6	
8	0601	0	00001534	0807EAC4	7	
9	0501	0	00001E76	100FE18A	8	
10	0501	0	00001975	100FE68B	8	
11	0501	0	00001474	100FEB8C	8	
12	0481	0	00001EE6	201FE11A	9	
13	0481	0	00001A65	201FE59B	9	
14	0481	0	000015E4	201FEA1C	9	
15	0601	1	00001204	807FA870	11	
16	0501	0	00001806	01005CE2	12	FF
17	0501	0	00001305	010061E3	12	
18	0481	0	00001C08	0200CDC8	13	
19	0481	0	00001787	0200D249	13	
20	0481	0	00001306	0200D6CA	13	
21	0441	0	00001D0A	0401B696	14	
22	0441	0	000018C9	0401BAD7	14	
23	0441	0	00001488	0401BF18	14	
24	0441	0	00001047	0401C359	14	
25	0381	0	0000180C	08038F34	15	
26	0481	1	00001C08	401C79A0	18	
27	0481	0	00001787	401C7E21	18	
28	0601	1	00001204	0201F884	20	38
29	0501	0	00001806	0403FD0A	21	
30	0501	0	00001305	0404020B	21	
31	0481	0	00001C08	08080E18	22	
32	0601	1	00001204	20203860	24	
33	0501	0	00001806	40407CC2	25	
34	0501	0	00001305	404081C3	25	
35	0481	0	00001C08	80810D88	26	
36	0481	0	00001787	80811209	26	
37	0481	0	00001306	8081168A	26	
38	0441	0	00001D0A	01003616	27	39 02
39	0441	0	000018C9	01003A57	27	
40	0441	0	00001488	01003E98	27	
41	0441	0	00001047	010042D9	27	
42	0381	0	0000180C	02008E34	28	
43	0381	0	0000148B	02009185	28	
44	0381	0	0000110A	02009536	28	
45	0301	0	00001B12	0401316E	29	
46	0301	0	00001811	0401346F	29	

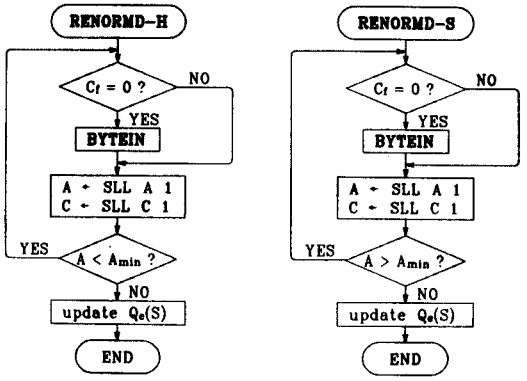


Figure 23

Decoder renormalization: (a) hardware; (b) software.

47	0301	0	00001510	04013770	29
48	0301	0	0000120F	04013A71	29
49	02C1	0	00001E1C	08027AE4	30
50	02C1	0	00001B5B	08027DA5	30
51	02C1	0	0000189A	08028066	30
52	02C1	0	000015D9	08028327	30
53	02C1	0	00001318	080285E8	30
54	02C1	0	00001057	080288A9	30
55	0281	0	00001B2C	100516D4	31
56	0281	0	000018AB	10051955	31
57	0281	0	0000162A	10051BD6	31

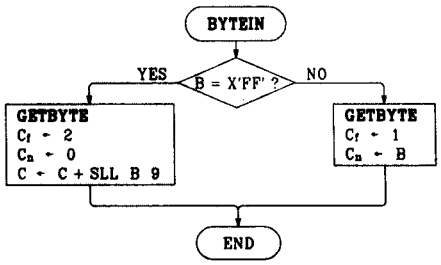


Figure 24

Adding a new byte of compressed data to the C register.

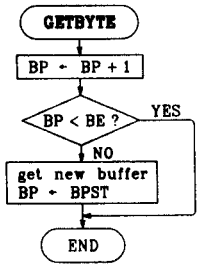


Figure 25

Compressed data buffer management for the decoder.

58	0281	0	000013A9	10051E57	31	
59	0281	0	00001128	100520D8	31	
60	0241	0	00001D4E	200A46B2	32	
61	0241	0	00001B0D	200A48F3	32	
62	0241	0	000018CC	200A4B34	32	
63	0241	0	0000168B	200A4D75	32	
64	0241	0	0000144A	200A4FB6	32	
65	02C1	1	00001208	01007DB0	35	52
66	0281	0	00001E8E	020100E2	36	
67	0301	1	00001408	10080710	39	
68	0301	0	00001107	10080A11	39	
69	02C1	0	00001C0C	20101A24	40	
70	02C1	0	0000194B	20101CE5	40	
71	02C1	0	0000168A	20101FA6	40	
72	02C1	0	000013C9	20102267	40	
73	0381	1	00001608	01001338	43	81
74	0481	1	00001C08	080099C0	46	
75	0481	0	00001787	08009E41	46	
76	0481	0	00001306	0800A2C2	46	
77	0441	0	00001D0A	10014E86	47	
78	0441	0	000018C9	100152C7	47	
79	0441	0	00001488	10015708	47	
80	0441	0	00001047	10015B49	47	
81	0381	0	0000180C	2002BF14	48	
82	0381	0	0000148B	2002C295	48	
83	0481	1	00001C08	010014A8	51	16
84	0481	0	00001787	01001929	51	
85	0481	0	00001306	01001DAA	51	
86	0441	0	00001D0A	02004456	52	
87	0441	0	000018C9	02004897	52	
88	0441	0	00001488	02004CD8	52	
89	0441	0	00001047	02005119	52	
90	0381	0	0000180C	0400AAB4	53	
91	0381	0	0000148B	0400AE35	53	
92	0381	0	0000110A	0400B1B6	53	
93	0301	0	00001B12	08016A6E	54	
94	0301	0	00001811	08016D6F	54	
95	0301	0	00001510	08017070	54	
96	0301	0	0000120F	08017371	54	
97	02C1	0	00001E1C	1002ECE4	55	
98	02C1	0	00001B5B	1002EFA5	55	
99	02C1	0	0000189A	1002F266	55	
100	02C1	0	00001509	1002F527	55	
101	02C1	0	00001318	1002F7E8	55	
102	02C1	0	00001057	1002FAA9	55	
103	0281	0	00001B2C	2005FAD4	56	
104	0281	0	000018AB	2005FD55	56	
105	0281	0	0000162A	2005FFD6	56	
106	0281	0	000013A9	20060257	56	
107	0281	0	00001128	200604D8	56	
108	0241	0	00001D4E	400C0EB2	57	
109	02C1	1	00001208	02007590	60	30
110	0281	0	00001E8E	0400F0A2	61	
111	0281	0	00001C0D	0400F323	61	
112	0301	1	00001408	20079918	64	
113	0301	0	00001107	20079C19	64	
114	0441	1	00001808	0100E0C8	67	3C
115	0441	0	000013C7	0100E509	67	
116	0381	0	00001F0C	0201D294	68	
117	0381	0	00001B8B	0201D615	68	
118	0381	0	0000180A	0201D996	68	
119	0481	1	00001C08	100ECCB0	71	
120	0481	0	00001787	100ED131	71	
121	0481	0	00001306	100ED5B2	71	
122	0601	1	00001204	403B56C8	73	
123	0501	0	00001806	8076B992	74	
124	0501	0	00001305	8076BE93	74	
125	0481	0	00001C08	01008728	75	ED
126	0481	0	00001787	01008BA9	75	
127	0601	1	00001204	04022EA4	77	
128	0501	0	00001806	0804694A	78	
129	0501	0	00001305	08046E4B	78	
130	0681	1	00001404	2011B92C	80	
131	0601	0	00001B06	40237F5A	81	
132	0601	0	00001505	4023855B	81	
133	0501	0	00001E08	804716B8	82	
134	0501	0	00001907	80471BB9	82	
135	0681	1	00001404	02006EE4	84	8E
136	0601	0	00001B06	0400EACA	85	
137	0601	0	00001505	0400F0CB	85	
138	0501	0	00001E08	0801ED98	86	
139	0501	0	00001907	0801F299	86	
140	0501	0	00001406	0801F79A	86	
141	0481	0	00001E0A	1003F936	87	
142	0481	0	00001989	1003FDB7	87	
143	0601	1	00001204	400FF6DC	89	
144	0501	0	00001806	801FF9BA	90	
145	0501	0	00001305	801FFEBB	90	
146	0481	0	00001C08	01000778	91	40
147	0601	1	00001204	04001DE0	93	
148	0681	1	00001804	10007780	95	
149	0681	0	00001183	10007E01	95	
150	0601	0	00001604	20010904	96	
151	0601	0	00001003	20010F05	96	
152	0501	0	00001404	40022A0C	97	
153	0481	0	00001E06	80045E1A	98	
154	0481	0	00001985	8004629B	98	
155	0601	1	00001204	02018A6C	100	08
156	0501	0	00001806	040320DA	101	
157	0681	1	00001404	100C8368	103	
158	0601	0	00001B06	201913D2	104	
159	0601	0	00001505	201919D3	104	
160	0681	1	00001804	8064674C	106	
161	0701	1	00001A04	02019D30	108	C8
162	0701	0	00001303	0201A431	108	
163	0681	0	00001804	04035664	109	
164	0701	1	00001A04	100D5990	111	
165	0701	0	00001303	100D6091	111	
166	0681	0	00001804	201ACF24	112	
167	0681	0	00001183	201AD5A5	112	

168	0601	0	00001604	4035B84C	113	
169	0601	0	00001003	4035BE4D	113	
170	0501	0	00001404	806B889C	114	
171	0681	1	00001404	02002270	116	D7
172	0701	1	00001A04	080089C0	118	
173	0701	0	00001303	080090C1	118	
174	0681	0	00001804	10012F84	119	
175	0681	0	00001183	10013605	119	
176	0701	1	00001A04	4004D814	121	
177	0701	0	00001303	4004DF15	121	
178	0681	0	00001804	8009CC2C	122	
179	0681	0	00001183	8009D2AD	122	
180	0701	1	00001A04	02014AB4	124	13
181	0901	1	00001C04	08052AD0	126	
182	0901	0	00001303	080533D1	126	
183	0A01	1	00001202	100A67A2	127	
184	0901	0	00001002	2014E346	128	
185	0701	0	00001C04	8053B11C	130	
186	0701	0	00001503	8053B81D	130	
187	0681	0	00001C04	01007E3C	131	A7
188	0681	0	00001583	010084BD	131	
189	0601	0	00001E04	0201167C	132	
190	0601	0	00001803	02011C7D	132	
191	0601	0	00001202	0201227E	132	
192	0501	0	00001802	040250FE	133	
193	0501	0	00001301	040255FF	133	
194	0481	0	00001C00	0804B600	134	
195	0481	0	0000177F	0804BA81	134	
196	0601	1	00001204	2012EA04	136	
197	0501	0	00001806	4025E00A	137	
198	0501	0	00001305	4025E50B	137	
199	0481	0	00001C08	804BD418	138	
200	0481	0	00001787	804BD899	138	
201	0481	0	00001306	804BDD1A	138	
202	0441	0	00001D0A	0100C336	139	97
203	0441	0	000018C9	0100C777	139	
204	0441	0	00001488	0100CBB8	139	
205	0441	0	00001047	0100CFF9	139	
206	0381	0	0000180C	0201A874	140	
207	0381	0	0000148B	0201ABF5	140	
208	0381	0	0000110A	0201AF76	140	
209	0301	0	00001B12	040365EE	141	
210	0441	1	00001808	201B2F70	144	
211	0501	1	00001104	806CBDC0	146	
212	0481	0	00001806	01008582	147	D9
213	0481	0	00001385	01008A03	147	
214	0441	0	00001E08	02011D08	148	
215	0441	0	000019C7	02012149	148	
216	0441	0	00001586	0201258A	148	
217	0441	0	00001145	020129CB	148	
218	0501	1	00001104	0804A72C	150	
219	0481	0	00001806	1009585A	151	
220	0481	0	00001385	10095CDB	151	
221	0441	0	00001E08	2012C2B8	152	
222	0441	0	000019C7	2012C6F9	152	
223	0441	0	00001586	2012CB3A	152	
224	0441	0	00001145	2012CF7B	152	
225	0501	1	00001104	804B3DEC	154	
226	0481	0	00001806	010085DA	155	96
227	0481	0	00001385	01008A5B	155	
228	0441	0	00001E08	02011DB8	156	
229	0441	0	000019C7	020121F9	156	
230	0441	0	00001586	0201263A	156	
231	0501	1	00001104	080498E8	158	
232	0481	0	00001806	10093BD2	159	
233	0481	0	00001385	10094053	159	
234	0441	0	00001E08	201289A8	160	
235	0441	0	000019C7	20128DE9	160	
236	0501	1	00001104	804A37A4	162	
237	0481	0	00001806	0100794A	163	94
238	0481	0	00001385	01007DCB	163	
239	0441	0	00001E08	02010498	164	
240	0441	0	000019C7	020108D9	164	
241	0441	0	00001586	02010D1A	164	
242	0441	0	00001145	0201115B	164	
243	0381	0	00001A08	04022B38	165	
244	0381	0	00001687	04022EB9	165	
245	0381	0	00001306	0402323A	165	
246	0301	0	00001F0A	08046B76	166	
247	0301	0	00001C09	08046E77	166	
248	0301	0	00001908	08047178	166	
249	0441	1	00001808	40238BC0	169	
250	0501	1	00001104	01002F00	171	8E
251	0481	0	00001806	02006802	172	
252	0481	0	00001385	02006C83	172	
253	0441	0	00001E08	0400E208	173	
254	0441	0	000019C7	0400E649	173	
255	0441	0	00001586	0400EA8A	173	
256	0441	0	00001145	0400EECB	173	
						3B B2 C0

Software encoder:

Note that the byte preceding the first compressed byte will always be modified.

EC	Q_e	YN	A	C	bits	codebytes
0	0AC1		00001000	00180000	0	
1	0A81	0	000014FC	00600000	2	
2	0A01	0	000014F6	00C00000	3	
3	0901	0	000015EA	01800000	4	
4	0701	0	000019D2	03000000	5	
5	0701	0	000012D1	03000000	5	
6	0681	0	000017A0	06000000	6	
7	0681	0	0000111F	06000000	6	
8	0601	0	0000153C	0C000000	7	
9	0501	0	00001E76	18000000	8	
10	0501	0	00001975	18000000	8	
11	0501	0	00001474	18000000	8	
12	0481	0	00001EE6	30000000	9	
13	0481	0	00001A65	30000000	9	
14	0481	0	000015E4	30000000	9	
15	0601	1	00001204	BFFFBA74	11	
16	0501	0	00001806	018074E8	12	80 FF
17	0501	0	00001305	018074E8	12	
18	0481	0	00001C08	0300E9D0	13	
19	0481	0	00001787	0300E9D0	13	
20	0481	0	00001306	0300E9D0	13	
21	0441	0	00001D0A	0601D3A0	14	
22	0441	0	000018C9	0601D3A0	14	
23	0441	0	00001488	0601D3A0	14	
24	0441	0	00001047	0601D3A0	14	
25	0381	0	0000180C	0C03A740	15	
26	0481	1	00001C08	601C95A8	18	
27	0481	0	00001787	601C95A8	18	
28	0601	1	00001204	03000A88	20	39
29	0501	0	00001806	06001510	21	
30	0501	0	00001305	06001510	21	
31	0481	0	00001C08	0C002A20	22	
32	0601	1	00001204	30004A64	24	
33	0501	0	00001806	600094C8	25	
34	0501	0	00001305	600094C8	25	
35	0481	0	00001C08	C0012990	26	
36	0481	0	00001787	C0012990	26	
37	0481	0	00001306	C0012990	26	
38	0441	0	00001D0A	01805320	27	02
39	0441	0	000018C9	01805320	27	
40	0441	0	00001488	01805320	27	
41	0441	0	00001047	01805320	27	
42	0381	0	0000180C	0300A640	28	
43	0381	0	0000148B	0300A640	28	
44	0381	0	0000110A	0300A640	28	
45	0301	0	00001B12	06014C80	29	
46	0301	0	00001811	06014C80	29	
47	0301	0	00001510	06014C80	29	
48	0301	0	0000120F	06014C80	29	
49	02C1	0	00001E1C	0C029900	30	
50	02C1	0	00001B5B	0C029900	30	
51	02C1	0	0000189A	0C029900	30	
52	02C1	0	000015D9	0C029900	30	
53	02C1	0	00001318	0C029900	30	
54	02C1	0	00001057	0C029900	30	
55	0281	0	00001B2C	18053200	31	
56	0281	0	000018AB	18053200	31	
57	0281	0	0000162A	18053200	31	
58	0281	0	000013A9	18053200	31	
59	0281	0	00001128	18053200	31	
60	0241	0	00001D4E	300A6400	32	
61	0241	0	00001B0D	300A6400	32	
62	0241	0	000018CC	300A6400	32	
63	0241	0	0000168B	300A6400	32	
64	0241	0	0000144A	300A6400	32	
65	02C1	1	00001208	01808FB8	35	52
66	0281	0	00001E8E	03011F70	36	
67	0301	1	00001408	18081B18	39	
68	0301	0	00001107	18081B18	39	
69	02C1	0	00001C0C	30103630	40	
70	02C1	0	0000194B	30103630	40	
71	02C1	0	0000168A	30103630	40	
72	02C1	0	000013C9	30103630	40	
73	0381	1	00001608	01802940	43	81
74	0481	1	00001C08	0C00B5C8	46	
75	0481	0	00001787	0C00B5C8	46	
76	0481	0	00001306	0C00B5C8	46	
77	0441	0	00001D0A	18016B90	47	
78	0441	0	000018C9	18016B90	47	
79	0441	0	00001488	18016B90	47	
80	0441	0	00001047	18016B90	47	
81	0381	0	0000180C	3002D720	48	
82	0381	0	0000148B	3002D720	48	
83	0481	1	00001C08	018030B0	51	16

84	0481	0	00001787	018030B0	51	
85	0481	0	00001306	018030B0	51	
86	0441	0	00001D0A	03006160	52	
87	0441	0	000018C9	03006160	52	
88	0441	0	00001488	03006160	52	
89	0441	0	00001047	03006160	52	
90	0381	0	0000180C	0600C2C0	53	
91	0381	0	0000148B	0600C2C0	53	
92	0381	0	0000110A	0600C2C0	53	
93	0301	0	00001B12	0C018580	54	
94	0301	0	00001811	0C018580	54	
95	0301	0	00001510	0C018580	54	
96	0301	0	0000120F	0C018580	54	
97	02C1	0	00001E1C	18030B00	55	
98	02C1	0	00001B5B	18030B00	55	
99	02C1	0	0000189A	18030B00	55	
100	02C1	0	000015D9	18030B00	55	
101	02C1	0	00001318	18030B00	55	
102	02C1	0	00001057	18030B00	55	
103	0281	0	00001B2C	30061600	56	
104	0281	0	000018AB	30061600	56	
105	0281	0	0000162A	30061600	56	
106	0281	0	000013A9	30061600	56	
107	0281	0	00001128	30061600	56	
108	0241	0	00001D4E	600C2C00	57	
109	02C1	1	00001208	03008798	60	30
110	0281	0	00001E8E	06010F30	61	
111	0281	0	00001C0D	06010F30	61	
112	0301	1	00001408	3007AD20	64	
113	0301	0	00001107	3007AD20	64	
114	0441	1	00001808	0180F8D0	67	3C
115	0441	0	000013C7	0180F8D0	67	
116	0381	0	00001F0C	0301F1A0	68	
117	0381	0	00001B8B	0301F1A0	68	
118	0381	0	0000180A	0301F1A0	68	
119	0481	1	00001C08	180EE8B8	71	
120	0481	0	00001787	180EE8B8	71	
121	0481	0	00001306	180EE8B8	71	
122	0601	1	00001204	603B68CC	73	
123	0501	0	00001806	C076D198	74	
124	0501	0	00001305	C076D198	74	
125	0481	0	00001C08	0180A330	75	ED
126	0481	0	00001787	0180A330	75	
127	0601	1	00001204	060240A8	77	
128	0501	0	00001806	0C048150	78	
129	0501	0	00001305	0C048150	78	
130	0681	1	00001404	3011CD30	80	
131	0601	0	00001B06	60239A60	81	
132	0601	0	00001505	60239A60	81	
133	0501	0	00001E08	C04734C0	82	
134	0501	0	00001907	C04734C0	82	
135	0681	1	00001404	030082E8	84	8E
136	0601	0	00001B06	060105D0	85	
137	0601	0	00001505	060105D0	85	
138	0501	0	00001E08	0C020BA0	86	
139	0501	0	00001907	0C020BA0	86	
140	0501	0	00001406	0C020BA0	86	
141	0481	0	00001E0A	18041740	87	
142	0481	0	00001989	18041740	87	
143	0601	1	00001204	601008E0	89	
144	0501	0	00001806	C02011C0	90	
145	0501	0	00001305	C02011C0	90	
146	0481	0	00001C08	01802380	91	40
147	0601	1	00001204	06002FE4	93	
148	0681	1	00001804	18008F84	95	
149	0681	0	00001183	18008F84	95	
150	0601	0	00001604	30011F08	96	
151	0601	0	00001003	30011F08	96	
152	0501	0	00001404	60023E10	97	
153	0481	0	00001E06	C0047C20	98	
154	0481	0	00001985	C0047C20	98	
155	0601	1	00001204	03019C70	100	08
156	0501	0	00001806	060338E0	101	
157	0681	1	00001404	180C976C	103	
158	0601	0	00001B06	30192ED8	104	
159	0601	0	00001505	30192ED8	104	
160	0681	1	00001804	C0647F50	106	
161	0701	1	00001A04	0301B734	108	C8
162	0701	0	00001303	0301B734	108	
163	0681	0	00001804	06036E68	109	
164	0701	1	00001A04	180D7394	111	
165	0701	0	00001303	180D7394	111	
166	0681	0	00001804	301AE728	112	
167	0681	0	00001183	301AE728	112	
168	0601	0	00001604	6035CE50	113	
169	0601	0	00001003	6035CE50	113	
170	0501	0	00001404	C06B9CA0	114	
171	0681	1	00001404	03003674	116	D7
172	0701	1	00001A04	0C00A3C4	118	
173	0701	0	00001303	0C00A3C4	118	
174	0681	0	00001804	18014788	119	
175	0681	0	00001183	18014788	119	
176	0701	1	00001A04	6004F218	121	
177	0701	0	00001303	6004F218	121	
178	0681	0	00001804	C009E430	122	
179	0681	0	00001183	C009E430	122	
180	0701	1	00001A04	030164B8	124	13
181	0901	1	00001C04	0C0546D4	126	
182	0901	0	00001303	0C0546D4	126	
183	0A01	1	00001202	180A79A4	127	
184	0901	0	00001002	3014F348	128	
185	0701	0	00001C04	C053CD20	130	
186	0701	0	00001503	C053CD20	130	
187	0681	0	00001C04	01809A40	131	A7
188	0681	0	00001583	01809A40	131	
189	0601	0	00001E04	03013480	132	
190	0601	0	00001803	03013480	132	
191	0601	0	00001202	03013480	132	
192	0501	0	00001802	06026900	133	
193	0501	0	00001301	06026900	133	
194	0481	0	00001C00	0C04D200	134	
195	0481	0	0000177F	0C04D200	134	
196	0601	1	00001204	3012FC08	136	
197	0501	0	00001806	6025F810	137	
198	0501	0	00001305	6025F810	137	
199	0481	0	00001C08	C04BF020	138	
200	0481	0	00001787	C04BF020	138	
201	0481	0	00001306	C04BF020	138	
202	0441	0	00001D0A	0180E040	139	97
203	0441	0	000018C9	0180E040	139	
204	0441	0	00001488	0180E040	139	
205	0441	0	00001047	0180E040	139	
206	0381	0	0000180C	0301C080	140	
207	0381	0	0000148B	0301C080	140	
208	0381	0	0000110A	0301C080	140	
209	0301	0	00001B12	06038100	141	
210	0441	1	00001808	301B4778	144	
211	0501	1	00001104	C06CCEC4	146	
212	0481	0	00001806	01809D88	147	D9
213	0481	0	00001385	01809D88	147	
214	0441	0	00001E08	03013B10	148	
215	0441	0	000019C7	03013B10	148	
216	0441	0	00001586	03013B10	148	
217	0441	0	00001145	03013B10	148	
218	0501	1	00001104	0C04B830	150	
219	0481	0	00001806	18097060	151	
220	0481	0	00001385	18097060	151	
221	0441	0	00001E08	3012E0C0	152	
222	0441	0	000019C7	3012E0C0	152	
223	0441	0	00001586	3012E0C0	152	
224	0441	0	00001145	3012E0C0	152	
225	0501	1	00001104	C04B4EF0	154	
226	0481	0	00001806	01809DE0	155	96
227	0481	0	00001385	01809DE0	155	
228	0441	0	00001E08	03013BC0	156	
229	0441	0	000019C7	03013BC0	156	
230	0441	0	00001586	03013BC0	156	
231	0501	1	00001104	0C04A9EC	158	
232	0481	0	00001806	180953D8	159	
233	0481	0	00001385	180953D8	159	
234	0441	0	00001E08	3012A7B0	160	
235	0441	0	000019C7	3012A7B0	160	
236	0501	1	00001104	C04A48A8	162	
237	0481	0	00001806	01809150	163	94
238	0481	0	00001385	01809150	163	
239	0441	0	00001E08	030122A0	164	
240	0441	0	000019C7	030122A0	164	
241	0441	0	00001586	030122A0	164	
242	0441	0	00001145	030122A0	164	
243	0381	0	00001A08	06024540	165	
244	0381	0	00001687	06024540	165	
245	0381	0	00001306	06024540	165	
246	0301	0	00001F0A	0C048A80	166	
247	0301	0	00001C09	0C048A80	166	
248	0301	0	00001908	0C048A80	166	
249	0441	1	00001808	6023A3C8	169	
250	0501	1	00001104	01804004	171	8E
251	0481	0	00001806	03008008	172	
252	0481	0	00001385	03008008	172	
253	0441	0	00001E08	06010010	173	
254	0441	0	000019C7	06010010	173	
255	0441	0	00001586	06010010	173	
256	0441	0	00001145	06010010	173	
C-A				0600EECB		3B B2 C0

Hardware decoder:

EC	Q_e	YN	A	C	bits	codebytes
0	0AC1		00001000	0FF72020	0	
1	0A81	0	000014FC	14088080	2	
2	0A01	0	000014F6	14AF0201	3	02
3	0901	0	000015EA	155C0402	4	
4	0701	0	000019D2	18B60804	5	
5	0701	0	000012D1	11B50804	5	
6	0681	0	000017A0	15681008	6	
7	0681	0	0000111F	0EE71008	6	
8	0601	0	0000153C	10CC2010	7	
9	0501	0	00001E76	15964020	8	
10	0501	0	00001975	10954020	8	
11	0501	0	00001474	0B944020	8	
12	0481	0	00001EE6	0D268040	9	
13	0481	0	00001A65	08A58040	9	
14	0481	0	000015E4	04248040	9	
15	0601	1	00001204	10925201	11	52
16	0501	0	00001806	1522A402	12	
17	0501	0	00001305	1021A402	12	
18	0481	0	00001C08	16414804	13	
19	0481	0	00001787	11C04804	13	
20	0481	0	00001306	0D3F4804	13	
21	0441	0	0000100A	117C9008	14	
22	0441	0	000018C9	0D3B9008	14	
23	0441	0	00001488	08FA9008	14	
24	0441	0	00001047	04B99008	14	
25	0381	0	0000180C	00F12010	15	
26	0481	1	00001C08	07890080	18	
27	0481	0	00001787	03080080	18	
28	0601	1	00001204	0C210202	20	81
29	0501	0	00001806	0C400404	21	
30	0501	0	00001305	073F0404	21	
31	0481	0	00001C08	047C0808	22	
32	0601	1	00001204	11F02020	24	
33	0501	0	00001806	17DE4040	25	
34	0501	0	00001305	12DD4040	25	
35	0481	0	00001C08	1BB88080	26	
36	0481	0	00001787	17378080	26	
37	0481	0	00001306	12B68080	26	
38	0441	0	00001D0A	1C6B1601	27	16
39	0441	0	000018C9	182A1601	27	
40	0441	0	00001488	13E91601	27	
41	0441	0	00001047	0FA81601	27	
42	0381	0	0000180C	16CE2C02	28	
43	0381	0	0000148B	134D2C02	28	
44	0381	0	0000110A	0FCC2C02	28	
45	0301	0	00001B12	18965804	29	
46	0301	0	00001811	15955804	29	
47	0301	0	00001510	12945804	29	
48	0301	0	0000120F	0F935804	29	
49	02C1	0	00001E1C	1924B008	30	
50	02C1	0	00001B5B	1663B008	30	
51	02C1	0	0000189A	13A2B008	30	
52	02C1	0	000015D9	10E1B008	30	
53	02C1	0	00001318	0E20B008	30	
54	02C1	0	00001057	0B5FB008	30	
55	0281	0	00001B2C	113D6010	31	
56	0281	0	000018AB	0EBC6010	31	
57	0281	0	0000162A	0C3B6010	31	
58	0281	0	000013A9	09BA6010	31	
59	0281	0	00001128	07396010	31	
60	0241	0	00001D4E	0970C020	32	
61	0241	0	00001B0D	072FC020	32	
62	0241	0	000018CC	04EEC020	32	
63	0241	0	0000168B	02ADC020	32	
64	0241	0	0000144A	006CC020	32	
65	02C1	1	00001208	03663001	35	30
66	0281	0	00001E8E	014A6002	36	
67	0301	1	00001408	0A530010	39	
68	0301	0	00001107	07520010	39	
69	02C1	0	00001C0C	08A20020	40	
70	02C1	0	0000194B	05E10020	40	
71	02C1	0	0000168A	03200020	40	
72	02C1	0	000013C9	005F0020	40	
73	0381	1	00001608	02F83C01	43	3C
74	0481	1	00001C08	17C1E008	46	
75	0481	0	00001787	1340E008	46	
76	0481	0	00001306	0EBFE008	46	
77	0441	0	00001D0A	147DC010	47	
78	0441	0	000018C9	103CC010	47	
79	0441	0	00001488	0BFBC010	47	
80	0441	0	00001047	07BAC010	47	
81	0381	0	0000180C	06F38020	48	
82	0381	0	0000148B	03728020	48	
83	0481	1	00001C08	1B94ED01	51	ED
84	0481	0	00001787	1713ED01	51	
85	0481	0	00001306	1292ED01	51	
86	0441	0	00001D0A	1C23DA02	52	
87	0441	0	000018C9	17E2DA02	52	
88	0441	0	00001488	13A1DA02	52	
89	0441	0	00001047	0F60DA02	52	
90	0381	0	0000180C	163FB404	53	
91	0381	0	0000148B	12BEB404	53	
92	0381	0	0000110A	0F3DB404	53	
93	0301	0	00001B12	17796808	54	
94	0301	0	00001811	14786808	54	
95	0301	0	00001510	11776808	54	
96	0301	0	0000120F	0E766808	54	
97	02C1	0	00001E1C	16EAD010	55	
98	02C1	0	00001B5B	1429D010	55	
99	02C1	0	0000189A	1168D010	55	
100	02C1	0	00001509	0EA7D010	55	
101	02C1	0	00001318	0BE6D010	55	
102	02C1	0	00001057	0925D010	55	
103	0281	0	00001B2C	0CC9A020	56	
104	0281	0	000018AB	0A48A020	56	
105	0281	0	0000162A	07C7A020	56	
106	0281	0	000013A9	0546A020	56	
107	0281	0	00001128	02C5A020	56	
108	0241	0	00001D4E	00894040	57	
109	02C1	1	00001208	044B1C02	60	8E
110	0281	0	00001E8E	03143804	61	
111	0281	0	00001C0D	00933804	61	
112	0301	1	00001408	0499C020	64	
113	0301	0	00001107	0198C020	64	
114	0441	1	00001808	0CC64001	67	40
115	0441	0	000013C7	08854001	67	
116	0381	0	00001F0C	08888002	68	
117	0381	0	00001B8B	05078002	68	
118	0381	0	0000180A	01868002	68	
119	0481	1	00001C08	0C340010	71	
120	0481	0	00001787	07B30010	71	
121	0481	0	00001306	03320010	71	
122	0601	1	00001204	0CC80040	73	
123	0501	0	00001806	0D8E0080	74	
124	0501	0	00001305	088D0080	74	
125	0481	0	00001C08	07180801	75	08
126	0481	0	00001787	02970801	75	
127	0601	1	00001204	0A5C2004	77	
128	0501	0	00001806	08B64008	78	
129	0501	0	00001305	03B54008	78	
130	0681	1	00001404	0ED50020	80	
131	0601	0	00001B06	10A80040	81	
132	0601	0	00001505	0AA70040	81	
133	0501	0	00001E08	094C0080	82	
134	0501	0	00001907	044B0080	82	
135	0681	1	00001404	112D9002	84	C8
136	0601	0	00001B06	15592004	85	
137	0601	0	00001505	0F582004	85	
138	0501	0	00001E08	12AE4008	86	
139	0501	0	00001907	0DAD4008	86	
140	0501	0	00001406	08AC4008	86	
141	0481	0	00001E0A	07568010	87	
142	0481	0	00001989	02D58010	87	
143	0601	1	00001204	0B560040	89	
144	0501	0	00001806	0AAA0080	90	
145	0501	0	00001305	05A90080	90	
146	0481	0	00001C08	0150D701	91	D7
147	0601	1	00001204	05435C04	93	
148	0681	1	00001804	150D7010	95	
149	0681	0	00001183	0E8C7010	95	
150	0601	0	00001604	1016E020	96	
151	0601	0	00001003	0A15E020	96	
152	0501	0	00001404	0829C040	97	
153	0481	0	00001E06	06518080	98	
154	0481	0	00001985	01D08080	98	
155	0601	1	00001204	07422602	100	13
156	0501	0	00001806	02824C04	101	
157	0681	1	00001404	0A093010	103	
158	0601	0	00001B06	07106020	104	
159	0601	0	00001505	010F6020	104	
160	0681	1	00001804	043D8080	106	
161	0701	1	00001A04	10F74E02	108	A7
162	0701	0	00001303	09F64E02	108	
163	0681	0	00001804	05EA9C04	109	
164	0701	1	00001A04	17AA7010	111	
165	0701	0	00001303	10A97010	111	
166	0681	0	00001804	1350E020	112	
167	0681	0	00001183	0CCFE020	112	
168	0601	0	00001604	0C9DC040	113	
169	0601	0	00001003	069CC040	113	
170	0501	0	00001404	01378080	114	
171	0681	1	00001404	04DF2E02	116	97
172	0701	1	00001A04	137CB808	118	
173	0701	0	00001303	0C78B808	118	
174	0681	0	00001804	0AF57010	119	
175	0681	0	00001183	04747010	119	
176	0701	1	00001A04	11D1C040	121	
177	0701	0	00001303	0AD0C040	121	
178	0681	0	00001804	079F8080	122	

179	0681	0	00001183	011E8080	122	
180	0701	1	00001A04	047BB202	124	D9
181	0901	1	00001C04	11EEC808	126	
182	0901	0	00001303	08EDC808	126	
183	0A01	1	00001202	11DB9010	127	
184	0901	0	00001002	0FB52020	128	
185	0701	0	00001C04	1AD08080	130	
186	0701	0	00001503	13CF8080	130	
187	0681	0	00001C04	199D9601	131	96
188	0681	0	00001583	131C9601	131	
189	0601	0	00001E04	19372C02	132	
190	0601	0	00001803	13362C02	132	
191	0601	0	00001202	0D352C02	132	
192	0501	0	00001802	0E685804	133	
193	0501	0	00001301	09675804	133	
194	0481	0	00001C00	08CCB008	134	
195	0481	0	0000177F	044BB008	134	
196	0601	1	00001204	112EC020	136	
197	0501	0	00001806	165B8040	137	
198	0501	0	00001305	115A8040	137	
199	0481	0	00001C08	18B30080	138	
200	0481	0	00001787	14320080	138	
201	0481	0	00001306	0FB10080	138	
202	0441	0	00001D0A	16609401	139	94
203	0441	0	000018C9	121F9401	139	
204	0441	0	00001488	0DDE9401	139	
205	0441	0	00001047	099D9401	139	
206	0381	0	0000180C	0AB92802	140	
207	0381	0	0000148B	07382802	140	
208	0381	0	0000110A	03B72802	140	
209	0301	0	00001B12	006C5004	141	
210	0441	1	00001808	03628020	144	
211	0501	1	00001104	0D8A0080	146	
212	0481	0	00001806	11128E01	147	8E
213	0481	0	00001385	0C918E01	147	
214	0441	0	00001E08	10211C02	148	
215	0441	0	000019C7	0BE01C02	148	
216	0441	0	00001586	079F1C02	148	
217	0441	0	00001145	035E1C02	148	
218	0501	1	00001104	0D787008	150	
219	0481	0	00001806	10EEE010	151	
220	0481	0	00001385	0C6DE010	151	
221	0441	0	00001E08	0FD9C020	152	
222	0441	0	000019C7	0B98C020	152	
223	0441	0	00001586	0757C020	152	
224	0441	0	00001145	0316C020	152	
225	0501	1	00001104	0C5B0080	154	
226	0481	0	00001806	0EB43B01	155	3B
227	0481	0	00001385	0A333B01	155	
228	0441	0	00001E08	0B647602	156	
229	0441	0	000019C7	07237602	156	
230	0441	0	00001586	02E27602	156	
231	0501	1	00001104	0B89D808	158	
232	0481	0	00001806	0D11B010	159	
233	0481	0	00001385	0890B010	159	
234	0441	0	00001E08	081F6020	160	
235	0441	0	000019C7	03DE6020	160	
236	0501	1	00001104	0F798080	162	
237	0481	0	00001806	14F1B201	163	B2
238	0481	0	00001385	1070B201	163	
239	0441	0	00001E08	17DF6402	164	
240	0441	0	000019C7	139E6402	164	
241	0441	0	00001586	0F5D6402	164	
242	0441	0	00001145	0B1C6402	164	
243	0381	0	00001A08	0DB6C804	165	
244	0381	0	00001687	0A35C804	165	
245	0381	0	00001306	0684C804	165	
246	0301	0	00001F0A	06679008	166	
247	0301	0	00001C09	03669008	166	
248	0301	0	00001908	00659008	166	
249	0441	1	00001808	032C8040	169	
250	0501	1	00001104	0CB2C001	171	C0
251	0481	0	00001806	0F638002	172	
252	0481	0	00001385	0AE28002	172	
253	0441	0	00001E08	0CC30004	173	
254	0441	0	000019C7	08820004	173	
255	0441	0	00001586	04410004	173	
256	0441	0	00001145	00000004	173	

Software decoder:

EC	Q_e	YN	A	C	bits	codebytes
0	0AC1		0000F000	FFF72020	0	
1	0A81	0	0000EB04	FFDC8080	2	
2	0A01	0	0000EB0A	FFB90201	3	02
3	0901	0	0000EA16	FF720402	4	
4	0701	0	0000E62E	FEE40804	5	
5	0701	0	0000ED2F	FEE40804	5	
6	0681	0	0000E860	FDC81008	6	
7	0681	0	0000EEE1	FDC81008	6	
8	0601	0	0000EAC4	FB902010	7	
9	0501	0	0000E18A	F7204020	8	
10	0501	0	0000E68B	F7204020	8	
11	0501	0	0000EB8C	F7204020	8	
12	0481	0	0000E11A	EE408040	9	
13	0481	0	0000E59B	EE408040	9	
14	0481	0	0000EA1C	EE408040	9	
15	0601	1	0000EDFC	FE8E5201	11	52
16	0501	0	0000E7FA	FD1CA402	12	
17	0501	0	0000ECFB	FD1CA402	12	
18	0481	0	0000E3F8	FA394804	13	
19	0481	0	0000E879	FA394804	13	
20	0481	0	0000ECFA	FA394804	13	
21	0441	0	0000E2F6	F4729008	14	
22	0441	0	0000E737	F4729008	14	
23	0441	0	0000EB78	F4729008	14	
24	0441	0	0000EFB9	F4729008	14	
25	0381	0	0000E7F4	E8E52010	15	
26	0481	1	0000E3F8	EB810080	18	
27	0481	0	0000E879	EB810080	18	
28	0601	1	0000EDFC	FA1D0202	20	81
29	0501	0	0000E7FA	F43A0404	21	
30	0501	0	0000ECFB	F43A0404	21	
31	0481	0	0000E3F8	E8740808	22	
32	0601	1	0000EDFC	FFEC2020	24	
33	0501	0	0000E7FA	FFD84040	25	
34	0501	0	0000ECFB	FFD84040	25	
35	0481	0	0000E3F8	FFB08080	26	
36	0481	0	0000E879	FFB08080	26	
37	0481	0	0000ECFA	FFB08080	26	
38	0441	0	0000E2F6	FF611601	27	16
39	0441	0	0000E737	FF611601	27	
40	0441	0	0000EB78	FF611601	27	
41	0441	0	0000EFB9	FF611601	27	
42	0381	0	0000E7F4	FEC22C02	28	
43	0381	0	0000EB75	FEC22C02	28	
44	0381	0	0000EEF6	FEC22C02	28	
45	0301	0	0000E4EE	FD845804	29	
46	0301	0	0000E7EF	FD845804	29	
47	0301	0	0000EAF0	FD845804	29	
48	0301	0	0000EDF1	FD845804	29	
49	02C1	0	0000E1E4	FB08B008	30	
50	02C1	0	0000E4A5	FB08B008	30	
51	02C1	0	0000E766	FB08B008	30	
52	02C1	0	0000EA27	FB08B008	30	
53	02C1	0	0000ECE8	FB08B008	30	
54	02C1	0	0000EFA9	FB08B008	30	
55	0281	0	0000E4D4	F6116010	31	
56	0281	0	0000E755	F6116010	31	
57	0281	0	0000E9D6	F6116010	31	
58	0281	0	0000EC57	F6116010	31	
59	0281	0	0000EED8	F6116010	31	
60	0241	0	0000E2B2	EC22C020	32	
61	0241	0	0000E4F3	EC22C020	32	
62	0241	0	0000E734	EC22C020	32	
63	0241	0	0000E975	EC22C020	32	
64	0241	0	0000EBB6	EC22C020	32	
65	02C1	1	0000EDF8	F15E3001	35	30
66	0281	0	0000E172	E2BC6002	36	
67	0301	1	0000EBF8	F64B0010	39	
68	0301	0	0000EEF9	F64B0010	39	
69	02C1	0	0000E3F4	EC960020	40	
70	02C1	0	0000E6B5	EC960020	40	
71	02C1	0	0000E976	EC960020	40	
72	02C1	0	0000EC37	EC960020	40	
73	0381	1	0000E9F8	ECF03C01	43	3C
74	0481	1	0000E3F8	FBB9E008	46	
75	0481	0	0000E879	FBB9E008	46	
76	0481	0	0000ECFA	FBB9E008	46	
77	0441	0	0000E2F6	F773C010	47	
78	0441	0	0000E737	F773C010	47	
79	0441	0	0000EB78	F773C010	47	
80	0441	0	0000EFB9	F773C010	47	
81	0381	0	0000E7F4	EEE78020	48	
82	0381	0	0000EB75	EEE78020	48	
83	0481	1	0000E3F8	FF8CED01	51	ED
84	0481	0	0000E879	FF8CED01	51	
85	0481	0	0000ECFA	FF8CED01	51	
86	0441	0	0000E2F6	FF19DA02	52	

87	0441	0	0000E737	FF19DA02	52	
88	0441	0	0000EB78	FF19DA02	52	
89	0441	0	0000EFB9	FF19DA02	52	
90	0381	0	0000E7F4	FE33B404	53	
91	0381	0	0000EB75	FE33B404	53	
92	0381	0	0000EEF6	FE33B404	53	
93	0301	0	0000E4EE	FC676808	54	
94	0301	0	0000E7EF	FC676808	54	
95	0301	0	0000EAF0	FC676808	54	
96	0301	0	0000EDF1	FC676808	54	
97	02C1	0	0000E1E4	F8CED010	55	
98	02C1	0	0000E4A5	F8CED010	55	
99	02C1	0	0000E766	F8CED010	55	
100	02C1	0	0000EA27	F8CED010	55	
101	02C1	0	0000ECE8	F8CED010	55	
102	02C1	0	0000EFA9	F8CED010	55	
103	0281	0	0000E4D4	F19DA020	56	
104	0281	0	0000E755	F19DA020	56	
105	0281	0	0000E9D6	F19DA020	56	
106	0281	0	0000EC57	F19DA020	56	
107	0281	0	0000EED8	F19DA020	56	
108	0241	0	0000E2B2	E33B4040	57	
109	02C1	1	0000EDF8	F2431C02	60	8E
110	0281	0	0000E172	E4863804	61	
111	0281	0	0000E3F3	E4863804	61	
112	0301	1	0000EBF8	F091C020	64	
113	0301	0	0000EEF9	F091C020	64	
114	0441	1	0000E7F8	F4BE4001	67	40
115	0441	0	0000EC39	F4BE4001	67	
116	0381	0	0000E0F4	E97C8002	68	
117	0381	0	0000E475	E97C8002	68	
118	0381	0	0000E7F6	E97C8002	68	
119	0481	1	0000E3F8	F02C0010	71	
120	0481	0	0000E879	F02C0010	71	
121	0481	0	0000ECFA	F02C0010	71	
122	0601	1	0000EDFC	FAC40040	73	
123	0501	0	0000E7FA	F5880080	74	
124	0501	0	0000ECFB	F5880080	74	
125	0481	0	0000E3F8	EB100801	75	08
126	0481	0	0000E879	EB100801	75	
127	0601	1	0000EDFC	F8582004	77	
128	0501	0	0000E7FA	F0B04008	78	
129	0501	0	0000ECFB	F0B04008	78	
130	0681	1	0000EBFC	FAD10020	80	
131	0601	0	0000E4FA	F5A20040	81	
132	0601	0	0000EAFB	F5A20040	81	
133	0501	0	0000E1F8	EB440080	82	
134	0501	0	0000E6F9	EB440080	82	
135	0681	1	0000EBFC	FD299002	84	C8
136	0601	0	0000E4FA	FA532004	85	
137	0601	0	0000EAFB	FA532004	85	
138	0501	0	0000E1F8	F4A64008	86	
139	0501	0	0000E6F9	F4A64008	86	
140	0501	0	0000EBFA	F4A64008	86	
141	0481	0	0000E1F6	E94C8010	87	
142	0481	0	0000E677	E94C8010	87	
143	0601	1	0000EDFC	F9520040	89	
144	0501	0	0000E7FA	F2A40080	90	
145	0501	0	0000ECFB	F2A40080	90	
146	0481	0	0000E3F8	E548D701	91	D7
147	0601	1	0000EDFC	F33F5C04	93	
148	0681	1	0000E7FC	FD097010	95	
149	0681	0	0000EE7D	FD097010	95	
150	0601	0	0000E9FC	FA12E020	96	
151	0601	0	0000EFFD	FA12E020	96	
152	0501	0	0000EBFC	F425C040	97	
153	0481	0	0000E1FA	E84B8080	98	
154	0481	0	0000E67B	E84B8080	98	
155	0601	1	0000EDFC	F53E2602	100	13
156	0501	0	0000E7FA	EA7C4C04	101	
157	0681	1	0000EBFC	F6053010	103	
158	0601	0	0000E4FA	EC0A6020	104	
159	0601	0	0000EAFB	EC0A6020	104	
160	0681	1	0000E7FC	EC398080	106	
161	0701	1	0000E5FC	F6F34E02	108	A7
162	0701	0	0000ECFD	F6F34E02	108	
163	0681	0	0000E7FC	EDE69C04	109	
164	0701	1	0000E5FC	FDA67010	111	
165	0701	0	0000ECFD	FDA67010	111	
166	0681	0	0000E7FC	FB4CE020	112	
167	0681	0	0000EE7D	FB4CE020	112	
168	0601	0	0000E9FC	F699C040	113	
169	0601	0	0000EFFD	F699C040	113	
170	0501	0	0000EBFC	ED338080	114	
171	0681	1	0000EBFC	F0DB2E02	116	97
172	0701	1	0000E5FC	F978B808	118	
173	0701	0	0000ECFD	F978B808	118	
174	0681	0	0000E7FC	F2F17010	119	
175	0681	0	0000EE7D	F2F17010	119	
176	0701	1	0000E5FC	F7CDC040	121	
177	0701	0	0000ECFD	F7CDC040	121	
178	0681	0	0000E7FC	EF9B8080	122	
179	0681	0	0000EE7D	EF9B8080	122	
180	0701	1	0000E5FC	EA77B202	124	D9
181	0901	1	0000E3FC	F5EAC808	126	
182	0901	0	0000ECFD	F5EAC808	126	
183	0A01	1	0000EDFE	FFD99010	127	
184	0901	0	0000EFFE	FFB32020	128	
185	0701	0	0000E3FC	FECC8080	130	
186	0701	0	0000EAFD	FECC8080	130	
187	0681	0	0000E3FC	FD999601	131	96
188	0681	0	0000EA7D	FD999601	131	
189	0601	0	0000E1FC	FB332C02	132	
190	0601	0	0000E7FD	FB332C02	132	
191	0601	0	0000EDFE	FB332C02	132	
192	0501	0	0000E7FE	F6665804	133	
193	0501	0	0000ECFF	F6665804	133	
194	0481	0	0000E400	ECCCB008	134	
195	0481	0	0000E881	ECCCB008	134	
196	0601	1	0000EDFC	FF2AC020	136	
197	0501	0	0000E7FA	FE558040	137	
198	0501	0	0000ECFB	FE558040	137	
199	0481	0	0000E3F8	FCAB0080	138	
200	0481	0	0000E879	FCAB0080	138	
201	0481	0	0000ECFA	FCAB0080	138	
202	0441	0	0000E2F6	F9569401	139	94
203	0441	0	0000E737	F9569401	139	
204	0441	0	0000EB78	F9569401	139	
205	0441	0	0000EFB9	F9569401	139	
206	0381	0	0000E7F4	F2AD2802	140	
207	0381	0	0000EB75	F2AD2802	140	
208	0381	0	0000EEF6	F2AD2802	140	
209	0301	0	0000E4EE	E55A5004	141	
210	0441	1	0000E7F8	EB5A8020	144	
211	0501	1	0000EEFC	FC860080	146	
212	0481	0	0000E7FA	F90C8E01	147	8E
213	0481	0	0000EC7B	F90C8E01	147	
214	0441	0	0000E1F8	F2191C02	148	
215	0441	0	0000E639	F2191C02	148	
216	0441	0	0000EA7A	F2191C02	148	
217	0441	0	0000EEBB	F2191C02	148	
218	0501	1	0000EEFC	FC747008	150	
219	0481	0	0000E7FA	F8E8E010	151	
220	0481	0	0000EC7B	F8E8E010	151	
221	0441	0	0000E1F8	F1D1C020	152	
222	0441	0	0000E639	F1D1C020	152	
223	0441	0	0000EA7A	F1D1C020	152	
224	0441	0	0000EEBB	F1D1C020	152	
225	0501	1	0000EEFC	FB570080	154	
226	0481	0	0000E7FA	F6AE3B01	155	3B
227	0481	0	0000EC7B	F6AE3B01	155	
228	0441	0	0000E1F8	ED5C7602	156	
229	0441	0	0000E639	ED5C7602	156	
230	0441	0	0000EA7A	ED5C7602	156	
231	0501	1	0000EEFC	FA85D808	158	
232	0481	0	0000E7FA	F50BB010	159	
233	0481	0	0000EC7B	F50BB010	159	
234	0441	0	0000E1F8	EA176020	160	
235	0441	0	0000E639	EA176020	160	
236	0501	1	0000EEFC	FE758080	162	
237	0481	0	0000E7FA	FCEBB201	163	B2
238	0481	0	0000EC7B	FCEBB201	163	
239	0441	0	0000E1F8	F9D76402	164	
240	0441	0	0000E639	F9D76402	164	
241	0441	0	0000EA7A	F9D76402	164	
242	0441	0	0000EEBB	F9D76402	164	
243	0381	0	0000E5F8	F3AEC804	165	
244	0381	0	0000E979	F3AEC804	165	
245	0381	0	0000ECFA	F3AEC804	165	
246	0301	0	0000E0F6	E75D9008	166	
247	0301	0	0000E3F7	E75D9008	166	
248	0301	0	0000E6F8	E75D9008	166	
249	0441	1	0000E7F8	EB248040	169	
250	0501	1	0000EEFC	FBAEC001	171	C0
251	0481	0	0000E7FA	F75D8002	172	
252	0481	0	0000EC7B	F75D8002	172	
253	0441	0	0000E1F8	EEBB0004	173	
254	0441	0	0000E639	EEBB0004	173	
255	0441	0	0000EA7A	EEBB0004	173	
256	0441	0	0000EEBB	EEBB0004	173	

Acknowledgments

As noted in the Introduction, this paper is part of a collaborative effort between IBM researchers at Almaden and Yorktown Heights to develop an arithmetic coding

implementation which is computationally efficient in both software and hardware. We have benefited greatly from the continuing interaction with R. Arps, J. Rissanen, and R. Pasco of the Almaden facility, and G. Langdon, formerly at Almaden and now retired from IBM. We have also benefited from interactions with C. Gonzales and G. Goertzel at the Yorktown facility, and from the continuing support and encouragement of our manager, K. Pennington.

References

1. W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM J. Res. Develop.* **32,** 717 (1988, this issue).
2. W. B. Pennebaker and J. L. Mitchell, "Probability Estimation for the Q-Coder," *IBM J. Res. Develop.* **32,** 737 (1988, this issue).
3. J. L. Mitchell and W. B. Pennebaker, "Optimal Hardware and Software Arithmetic Coding Procedures for the Q-Coder," *IBM J. Res. Develop.* **32,** 727 (1988, this issue).
4. J. L. Mitchell, W. B. Pennebaker, C. A. Gonzales, and C. F. Touchton, "A Predictive Image Coder/Decoder Using Adaptive Binary Arithmetic Coding," *Research Report RC-13423*, IBM T. J. Watson Research Center, Yorktown Heights, NY, January 1988.
5. G. G. Langdon, "An Introduction to Arithmetic Coding," *IBM J. Res. Develop.* **28,** 135 (1984).
6. G. G. Langdon and J. J. Rissanen, "Compression of Black-White Images with Arithmetic Coding," *IEEE Trans. Commun.* **COM-29,** 858 (1981).
7. G. G. Langdon and J. J. Rissanen, "High-Speed Arithmetic Compression Using Concurrent Value Updating," U.S. Patent 4,467,317, August 21, 1984.
8. J. Rissanen and G. G. Langdon, "Universal Modeling and Coding," *IEEE Trans. Info. Theory* **IT-27,** 12 (1981).
9. G. Goertzel and J. L. Mitchell, "Symmetrical Optimized Adaptive Data Compression/Transfer/Decompression System," U.S. Patent 4,633,490, December 30, 1986.
10. R. B. Arps, J. M. Cheng, and G. G. Langdon, "Control Character Insertion into Arithmetically Encoded Strings," *IBM Tech. Disclosure Bull.* **25,** 2051 (1982).
11. D. Anastassiou, J. L. Mitchell, and W. B. Pennebaker, "Gray-Scale Image Coding for Freeze-Frame Videoconferencing," *IEEE Trans. Commun.* **COM-34,** 382 (1986).

Received August 11, 1987; accepted for publication August 17, 1988

Joan L. Mitchell *IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Mitchell graduated from Stanford University with a B.S. in physics in 1969. She received her M.S. and Ph.D. degrees in physics from the University of Illinois at Champaign-Urbana in 1971 and 1974, respectively. She joined the Exploratory Printing Technologies group at the IBM T. J. Watson Research Center immediately after completing her Ph.D., and since 1976 has worked in the field of data compression. Dr. Mitchell received IBM Outstanding Innovation Awards for two-dimensional data compression in 1978, for teleconferencing in 1982, and for the Image View Facility and resistive ribbon thermal transfer printing technology in 1985. She is co-inventor on fifteen patents.

William B. Pennebaker *IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Pennebaker is a Research Staff Member at the IBM T. J. Watson Research Center and currently manages a group doing research in areas related to image processing and compression. He joined IBM's Research Division in 1962 and has worked in areas related to low-temperature physics, thin films, display technology, printing technology, and image processing. Dr. Pennebaker has received an Outstanding Contribution Award for work on strontium titanate films, an Outstanding Invention Award for work on silicon nitride films, and an Outstanding Innovation Award for work on image processing and compression. He has received ten IBM Invention Achievement Awards. Dr. Pennebaker received his B.S. in engineering physics from Lehigh University, Bethlehem, Pennsylvania, in 1957, and his Ph.D. in physics from Rutgers University, New Brunswick, New Jersey, in 1962. He is a member of the American Association for the Advancement of Science, the American Institute of Physics, the Institute of Electrical and Electronics Engineers, and the Society for Information Display.