E C M A

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA-159

DATA COMPRESSION FOR INFORMATION INTERCHANGE - BINARY ARITHMETIC CODING ALGORITHM -

December 1991

E C M A

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA-159

DATA COMPRESSION FOR INFORMATION INTERCHANGE - BINARY ARITHMETIC CODING ALGORITHM -

December 1991

BRIEF HISTORY

In the past decades ECMA have published numerous ECMA Standards for magnetic tapes, magnetic tape cassettes and cartridges, as well as for optical disk cartridges. Those media developed recently have a very high physical recording density. In order to make an optimal use of the resulting data capacity, compression algorithms have been designed which allow a reduction of the number of bits required for the representation of user data in coded form.

In future, these compression algorithms will be registered by an International Registration Authority to be set up by ISO/IEC. The registration will consist in allocating to each registered algorithm a numerical identifier which will be recorded on the medium and, thus, indicate which compression algorithm(s) has been used.

ECMA has undertaken work on a series of ECMA Standards for compression algorithms. The first of these ECMA Standards was published in June 1991:

ECMA-151:	Data Compression for Information Interchange - Adaptive Coding with Embedded Dictionary - DCLZ Algorithm

The present ECMA Standard is the next one of this series. Both Standard ECMA-151 and the present Standard ECMA-159 have been contributed to ISO/IEC for adoption as International Standards under the fast-track procedure.

Adopted by the General Assembly of ECMA in December 1991.

Table of Contents

1 Scope

This ECMA Standard specifies an algorithm for the reduction of the number of bits required to represent information. This process is known as Data Compression. The algorithm uses binary arithmetic coding. The algorithm provides lossless compression and is intended for use in information interchange.

2 Conformance

A compression algorithm shall be in conformance with this Standard if it satisfies all mandatory requirements of this Standard.

3 References

International Register of Algorithms for Lossless Compression of Data

(to be established).

4 Conventions and Notations

The following conventions and notations apply in this Standard unless otherwise stated:

- In each field the bytes shall be arranged with Byte 1, the most significant, first. Within each byte the bits shall be arranged with Bit 1, the most significant bit, first and Bit 8, the least significant bit, last.
- Letters and digits in parentheses represent numbers in hexadecimal notation.
- The setting of bits is denoted by ZERO or ONE.
- Numbers in binary notation and bit combinations are represented by strings of ZEROs and ONEs.
- Numbers in binary notation and bit combinations are shown with the most significant bit to the left.

5 Algorithm Identifier

The numeric identifier of this algorithm in the International Register shall be 16.

6 Definitions

For the purposes of this Standard, the following definitions apply.

6.1 Block

A portion of the Logical Data Record, usually having a length of 512 bytes (see 8.2).

6.2 Code Block

A Block after compression with a Trailer appended.

6.3 Code String

The encoded Logical Data Record.

6.4 Encoding

The process of generating Code Blocks from Blocks.

6.5 Input Event

The sample of the input to an encoder currently being examined; in Run Mode it is a byte; in Normal Mode it is a bit.

6.6 Logical Data Record

The data entity that is the input to the data compressor.

6.7 Trailer

Data appended to a Block after compression and addition of pad bits.

6.8 Unique Table Pair

The last of the 256 Table Pairs, used only in Run Mode.

7 List of Acronyms

CV	Current Value
EV	Estimated Value
LDR	Logical Data Record
TP	Table Pair

8 Compression Algorithm

8.1 General

The LDR is transformed to a Code String by a one-pass, adaptive encoding technique designed to provide lossless data compression. By the use of a suitable decoding technique the exact original LDR can be recovered from the Code String.

8.2 Encoders

The LDR shall be divided into 512-byte Blocks, except for the last Block, which may be of any length less than, or equal to, 512 bytes. The Blocks shall be routed sequentially to eight encoders, numbered from 0 to 7, commencing with encoder 0. If the LDR contains more than 4 096 bytes the compressor shall return to encoder 0 and repeat the process (see figure 2).

8.3 Formation of a Code Block

The output of each encoder is a Code Block (see figure 1).

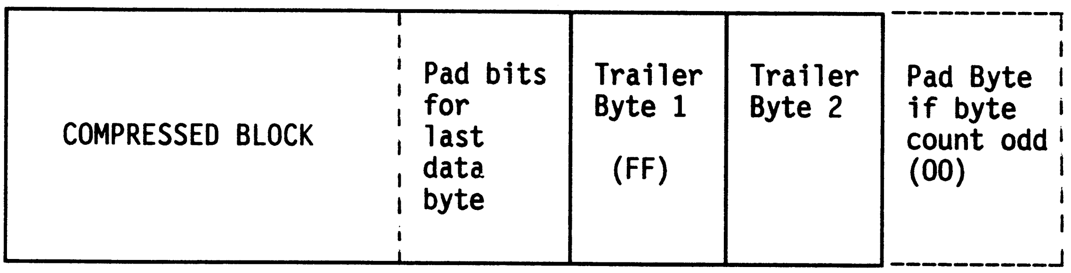

Figure 1 - Code Block

Since the degree of compression achieved in an encoder depends upon the relative frequency of the bit patterns in the LDR, and upon the presence of sequences of identical bytes, the length of the compressed data cannot be predicted. Pad bits set to ZERO shall be added at the end to form an integral number of 8-bit bytes.

The Code Block shall be completed by appending a Trailer. The Trailer shall consist of two or three bytes.

Byte 1	shall be set to (FF).
Byte 2	
Bits 1 to 4	shall be set to 1100 if the Code Block has been generated from the last Block of the LDR. shall be 1001 for all other Code Blocks.
Bit 5	shall be set to ZERO if the number of bytes after encoding is even. shall be set to ONE if the number of bytes after encoding is odd.
Bits 6 to 8	shall specify the number of pad bits that have been added to form an integral number of bytes.

If the number of bytes in the Compressed Block plus the pad bits, is odd, a third byte set to (00) shall be appended after Trailer Byte 2 to give an even number of bytes.

8.4 Code String

The Code String shall be assembled from the outputs of the encoders, with the first portion being that generated by encoder 0, the second that generated by encoder 1, and so on (see figure 2).

8.5 Table Pairs

Each encoder shall be allocated a table of 256 pairs of numbers, numbered from 1 to 256. The first number of each Table Pair shall be the estimated value (EV) of the Input Event to be encoded; it shall be 1 or 0. The second number (K) shall be a measure of the probability of the Input Event being equal to the EV. K shall have the value 1, 2, 3 or 4, with the probability shown in table 1.

Table 1- Probability values of K

K	Probability
1	1 - 2
2	2 - 4
3	4 - 8
4	8 - 16

The probabilities shall be a measure of how much more likely it is that the value of the Input Event is equal to the EV rather than being unequal (e.g. for K=2 the probability that the Input Event is equal to the EV shall be 2 to 4 times as great as the probability that it would not be equal).

Before commencing the encoding of the LDR all EVs shall be set to ZERO and all values of K shall be set to ONE.

8.6 Encoding

The data shall first be examined on a byte basis. Bytes shall be fetched sequentially from the Block, starting with the first byte, and compared with the previous byte. The first byte in a Block shall be compared with (40).

Run Mode (see 8.6.2) shall be disabled when the first byte is fetched.

If the current byte differs from the previous byte and Run Mode is not enabled, then the byte shall be encoded, bit by bit, in Normal Mode (see 8.6.1).

If the current byte differs from the previous byte and Run Mode is enabled, then encoding shall proceed as defined in 8.6.2.2.

If the two bytes are identical and Run Mode is not enabled, then Run Mode shall be enabled and the byte shall be encoded, bit by bit, in Normal Mode.

If the two bytes are identical and Run Mode is enabled, then encoding shall proceed as defined in 8.6.2.1.

8.6.1 Normal Mode

The first (most significant) bit of the byte shall be compared with the EV in the first Table Pair. Depending upon the result of this comparison, one of two actions, which are described in 8.6.1.1, shall result. The choice of which Table Pair to select for the remaining bits of the byte shall be determined by the bits previously encoded in the byte.

The first bit of each byte shall always use the first Table Pair.

The second bit shall use the second or third Table Pair, depending upon whether the first bit was ZERO or ONE, respectively.

The third bit shall use one of the next four Table Pairs depending upon the first two bits. The fourth Table Pair shall be used if the first two bits were 00, and so on (see figure 3).

This procedure requires the first 255 Table Pairs. The remaining Table Pair is the Unique Table Pair; it shall be used in the Run Mode only (see 8.6.2).

The process of encoding is then performed in two stages:

- The bit is encoded as in 8.6.1.1.
- The values of K and EV are revised as described in 8.6.1.2.

8.6.1.1 Bit Encoding

Two values shall be developed during the bit comparison portion of the compression process. Both are fractional binary numbers to four binary places. One value, termed the Current Value (CV), shall be used to generate the output (Code Block).

The second value, termed the Width, shall have values in the range 0.0000 to 1.1111.

For each Block the CV shall be initialized to 0.0000 and the Width value shall be initialized to 1.0000.

Each Input Event shall cause the two values to be modified according to the following rules:

i) The Input Event is equal to the EV

The CV shall be increased by 2^{-K}

The Width shall be decreased by 2^{-K}

where K is the measure of the probability of the Input Event (see 8.5).

If the Code Block is not null the bit to the left of the point in the CV shall be logically added to the last bit appended to the Code Block and replaced in the CV by a ZERO. If this addition causes the most recently generated complete byte in the Code Block to become (FF), then (0) shall be appended to the Code Block at the end of that byte to prevent a carry from propagating beyond the last complete byte.

If the Code Block is null the bit to the left of the point in the CV will already be a ZERO.

The Width shall then be compared with 1.0000.

If it is equal to, or greater than, 1.0000, then the Table Pair shall be revised (see 8.6.1.2) and the next bit fetched for encoding.

If it is less than 1.0000, then all the bits of the Width shall be shifted left one position; the leftmost bit, set to ZERO, shall be dropped, and the rightmost position shall be filled with a ZERO. The bit in the first position to the right of the point in the CV shall be appended to the encoder output as part of the Code Block. The remaining bits to the right of the point in the CV shall be shifted left one position and the rightmost position shall be filled with a ZERO.

To prevent a carry from propagating beyond the last complete byte, each byte in the Code Block shall be examined as it is completed. If it equals (FF) then (0) shall be appended to the Code Block.

ii) The Input Event is not equal to the EV.

The Width shall be reset to 1.0000.

The K bits to the right of the point in the CV shall be shifted out and appended to the Code Block, the leftmost bit first. As each bit is appended to the Code Block a check shall be made to determine whether a Byte boundary has been reached. If it has, then the byte just completed shall be checked; if it equals (FF) then (0) shall be appended to the Code Block.

The rightmost K bit positions of the CV shall be set to ZEROs.

8.6.1.2 Revision of K and EV (see figure 4)

As each Input Event is compared with the EV the values of K and the EV for that Table Pair shall be amended.

A four-stage binary counter (Mc counter) shall be set to 0000 at the beginning of each Block and incremented as described below.

i) The Input Event is equal to the EV

K shall be incremented according to table 2, where bits marked X shall be ignored.

Table 2 - Rules for incrementing K

Current value of K	State of Counter	New value of K
1	XX11	2
2	X111	3
3	1111	4
4	XXXX	4

For all other states of the counter K shall be unchanged.

The counter shall then be incremented by 1. If the counter value was 1111 incrementing by 1 shall result in a counter value of 0000.

The EV shall be unchanged.

ii) The Input Event is not equal to the EV

For K greater than 1	:	The value of K shall be decremented by 1 the counter shall not be incremented the EV shall be unchanged
For K equal to 1	:	the value of K shall be unchanged the counter shall not be incremented the EV shall be inverted

8.6.2 Run Mode

Run Mode shall have been enabled if the last two bytes to be encoded were identical.

8.6.2.1 If the current byte is identical with these two bytes, then the Unique Table Pair is selected and the EV is compared with ONE; the values of the Width, CV, K and EV are amended, as defined in 8.6.1.1 and 8.6.1.2. This may result in the Code Block being extended.

8.6.2.2 If the current byte differs from these two bytes then the Unique Table Pair is selected and the EV is compared with ZERO; the values of the Width, CV, K and EV are amended, as defined in 8.6.1.1 and 8.6.1.2. This may result in the Code Block being extended.

The byte is then encoded in Normal Mode and the values of the Width, CV, K and EV are amended as defined in 8.6.1.1 and 8.6.1.2. This may result in the Code Block being extended. Run Mode shall then be disabled.

8.7 Completion of the Encoding of a Block

If, when the final byte of the Block has been encoded and Run Mode is enabled, action shall be taken as defined in 8.6.2.2, except that no byte remains to be encoded in Normal Mode.

When this action has been taken, or if Run Mode was not enabled, the encoder shall be cleared as follows:

The four bits to the right of the point in the CV shall be appended to the Code Block, starting with the leftmost bit. As each bit is appended to the Code Block a check shall be made to determine whether a byte boundary has been reached. If it has, then the byte just completed shall be checked; if it equals (FF) then (0) shall be appended to the Code Block. Any remaining bits from the CV shall be appended to the Code Block. Pad bits and the Trailer shall then be appended to the Code Block as described in 8.3.

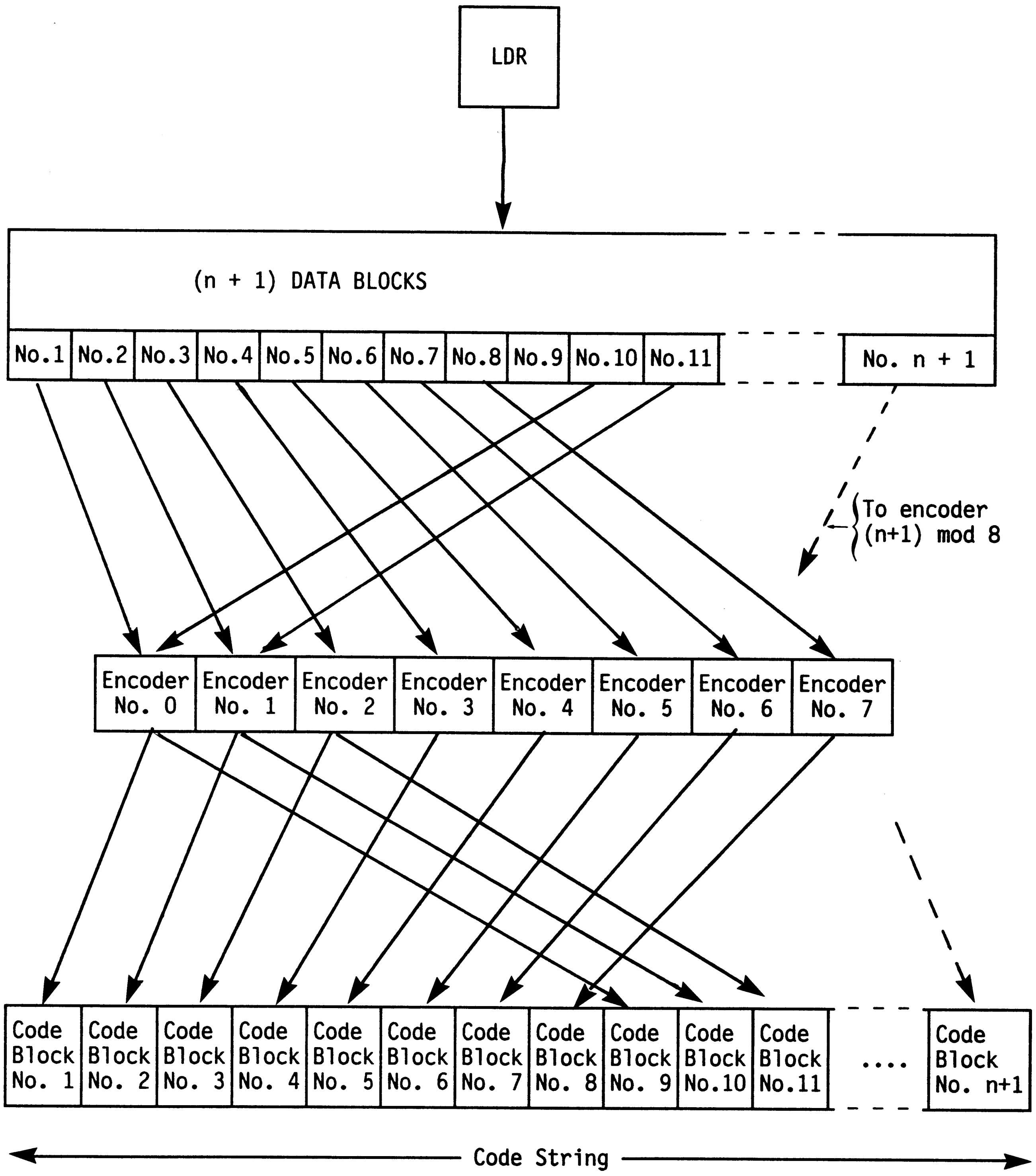

Figure 2 - Sequence of Encoding

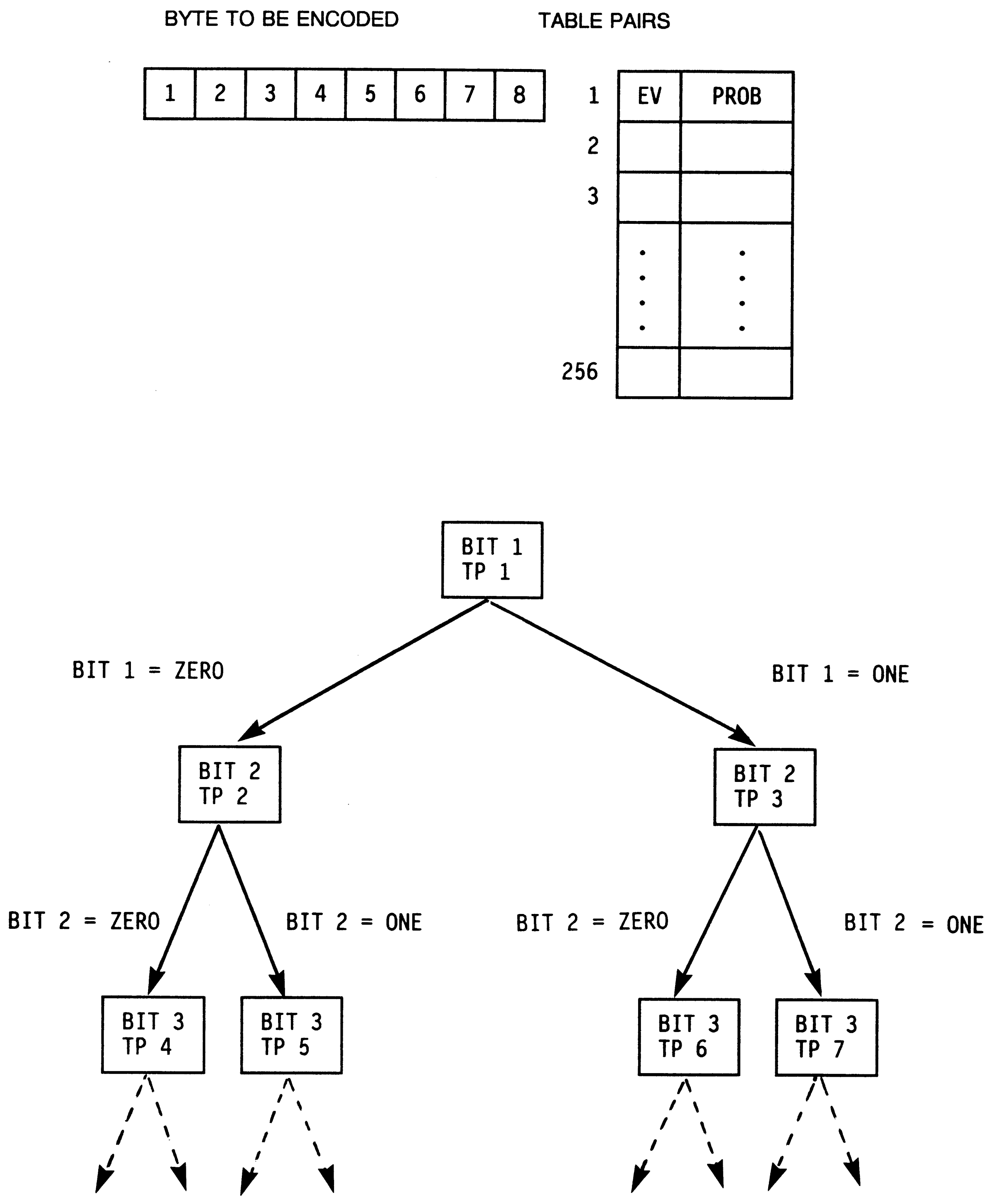

Figure 3 - Choice of Table Pairs

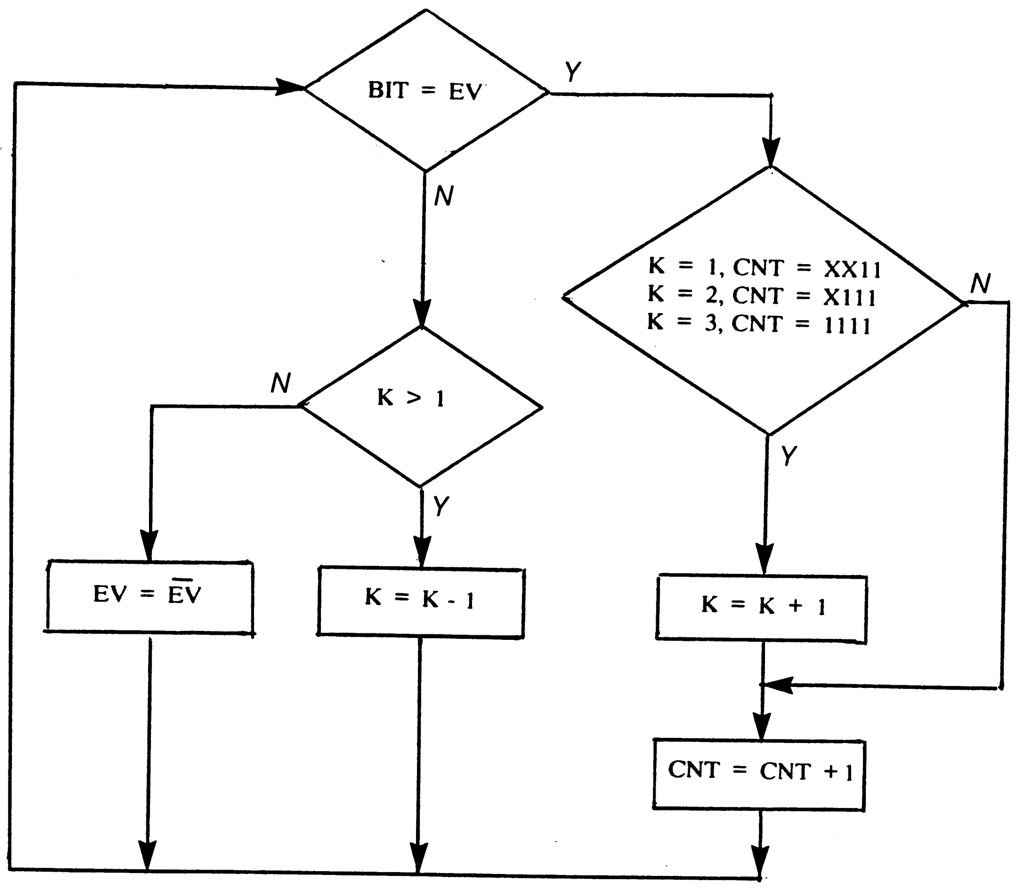

Figure 4 - Determination of K

Annex A

(informative)

Example of a binary arithmetic coding algorithm

A.1 Introduction

The following is a description of a Binary Arithmetic Coding Algorithm, expressed in structured English, also known as pseudo code. It is intended to give an overall understanding of the algorithm; it is not intended to be a complete and accurate description of the particular implementation described in this Standard.

The compression of a data record requires several successive processes. This example defines nine separate processes, which are linked together by Calls, that is, when a particular process requires the action of another process to be completed, the required process is called from within the calling process. These Calls are denoted by the called process being written in upper case.

The pseudo code structure uses basic operators such as IF, THEN, ELSE, SET, and DO WHILE. The only exception is the CASE statement which is used to replace several IF, THEN, ELSE statements. Indentation is used to clarify which operators are related. For instance, the IF, THEN, and ELSE operators of a given IF operation are all indented by the same distance.

A.2 Description of the Processes

A.2.1 Compact

Compact divides the Logical Data Record into Blocks of 512 bytes or less, and directs them sequentially to one of eight encoders. The Blocks are encoded and then concatenated to form the output, called the Code String.

A.2.2 Block Encode

This process covers the actual encoding of the individual Blocks. The Width and Current Value are introduced. The Current Value is used to generate the actual encoded output whilst the Width controls the level of compression achieved. Compression is achieved by deciding, based on the previous content of the Logical Data Record, what is the most probable state of the next bit in the byte. If this estimate is correct and the probability of the estimate being correct is high, the Width changes very little and one bit, or no bit is added to the Code Block. If the estimate is incorrect, then one to four bits are added to the Code Block.

Obviously, the more repetitive the data is, the more likely it is that the actual data will match the estimate and the greater will be the compression achieved.

Another mode of operation is also defined in this process. If the current byte is equal to the previous two bytes, then the whole byte is handled at once in Run Mode. The Unique Table Pair is used and the estimate is compared with 1. Width and Current Values are modified in the same way as they are for the comparison of individual bits.

A.2.3 Compute

Compute determines how the Width and Current Value are changed when the bit being examined matches the Estimated Value.

A.2.4 Bit Encode

This details how the individual bits within a byte of a Block are processed, and the order in which they are handled.

A.2.5 Check

This process is used to move data from the Current Value to the Code Block. There are two ways of doing this; the first is to take the value to the left of the point in the Current Value and logically add it to the Code Block. Since this may cause a carry, a mechanism has been added to limit how much of the Code Block can be modified by the carry. The most recently formed complete byte of the Code Block is examined; if it is equal to (FF), four ZERO bits are appended after this byte. Any further carries caused by logically adding the value to the left of the point in the Current Value will thus be prevented from propagating further by the (0).

The second way to move data is to append one or more bits from the right of the point in the Current Value to the Code Block. The number of bits to add is determined by whether the bit in question matches the estimate and, if not, what was the probability that the two would match. Again, as bits are appended to the Code Block, if an (FF) byte is formed, (0) is appended to prevent any subsequent carries from propagating more than one byte into the Code Block.

A.2.6 Table Pairs Update

The manner in which the estimates and probabilities are updated is defined. The CASE statement can be read as a cascading IF statement; that is, each set of conditions is checked until a match is found or until the end of the CASE statement is reached. If a match is found, the probability is set as described for that set of conditions.

A.2.7 Next Table Pair

This process determines which Table Pair will be used to evaluate the next bit in the byte. The first Table Pair is always used for the first bit of each byte. The second or third Table Pair is used for the second bit, depending upon whether the first bit was a ZERO or a ONE, respectively, In a similar fashion, the proper Table Pair to be used for each bit is determined, based on all the previously processed bits of the byte.

A.2.8 Clear Encoder

This terminates the encoding when the complete Block has been processed. If Run Mode is enabled, then the estimate in the Unique Table Pair is used to close out Run Mode.

If Run Mode is not enabled, or once Run Mode has been closed out, then the four bits to the right of the point in the Current Value are appended to the Code Block, again checking for an (FF) byte. ZEROs are appended to the Code Block to reach a byte boundary.

A.2.9 Trailer

This final process adds the Trailer, which is composed of an (FF) byte, an information byte, and, if necessary, an all ZEROs byte to make the total number of bytes in the completed Code Block an even number.

A.3 Description of the Process in Pseudo Code

A.3.1 COMPACT - Compact describes the overall process of breaking the Logical Data Record into Blocks and encoding them in the appropriate Encoder.

```
PROCESS   = COMPACT
RECEIVE   Logical__Data__Record
RESET     Code__String
SET       Encoder__Number = 0
SET       256 Table__Pairs (1:256) to {0,1} in each of 8 Tables (0:7)
IF        Logical__Data__Record > 512 bytes
THEN      ROUTE first 512 byte Block from the Logical__Data__Record to Encoder 0 and
                BLOCK__ENCODE (see A.3.2) using Table 0. The Code__Block (output of
                Encoder) is the first portion of the Code__String (compressor output).
          DO WHILE      remainder of Logical__Data__Record is > 512 bytes
             SET        Encoder__Number = Encoder__Number + 1
             IF         Encoder__Number is ≤ 7
             THEN       ROUTE next 512-byte Block to Encoder (Encoder__Number) and
                              BLOCK__ENCODE using Table (Encoder__Number).
             ELSE       SET   Encoder__Number = 0
                        ROUTE next 512-byte Block to Encoder 0 and BLOCK__ENCODE using
                              Table 0.
             APPEND     Code__Block to Code__String.
          END DO
          SET     Encoder__Number = Encoder__Number + 1
          IF      Encoder__Number is ≤ 7
          THEN    ROUTE remainder of Logical__Data__Record to Encoder (Encoder__Number) and
                        BLOCK__ENCODE using Table (Encoder__Number).
          ELSE    SET   Encoder__Number = 0
                  ROUTE remainder of Logical__Data__Record to Encoder 0 and
                        BLOCK__ENCODE using Table 0.
          APPEND Code__Block to Code__String.
ELSE      ROUTE Logical__Data__Record to Encoder 0 and BLOCK__ENCODE using Table 0.
                Code__Block (output of Encoder) is the Code__String (compressor output).
END PROCESS = COMPACT
```

A.3.2 BLOCK__ENCODE - Encoding Blocks is the process of converting Blocks of 512 bytes or less into Code__Blocks. Code__Blocks are terminated with Trailers.

```
PROCESS  = BLOCK__ENCODE
SET        Width = 1.0000
SET        Current__Value (CV) = 0.0000
SET        Previous__Byte = (40)
SET        Run__Mode = 0
SET        Mc__Count = 0000
SET        Block__Bytes__Remaining = number of bytes in Block
SET        Byte__Number = 1
DO WHILE Block__Bytes__Remaining > 0
           SET     Current__Byte = byte (Byte__Number) of Block
           SET     Byte__Number = Byte__Number + 1
           SET     Block__Bytes__Remaining = Block__Bytes__Remaining - 1
           IF      Current__Byte = Previous__Byte
           THEN    IF   Run__Mode = 1,
                   THEN    IF    first value of Unique__Table__Pair = 1
                           THEN    COMPUTE (see A.3.3)
                           ELSE    SET        Width = 1,0000
                                   SET        Compare = false
                                   SET        Shift__Left = second value in Unique__Table__Pair
                                   CHECK
                                   TABLE__PAIRS__UPDATE
                   ELSE    BIT__ENCODE (see A.3.4)
                           SET        Run__Mode = 1
           ELSE    SET     Previous__Byte = Current__Byte
                   IF      Run__Mode = 1
                   THEN IF first value of Unique__Table__Pair = 0
                           THEN       COMPUTE
                           ELSE       SET      Width = 1,0000
                                      SET      Compare = false
                                      SET      Shift__Left = second value of Unique__Table__Pair
                                      CHECK
                                      TABLE__PAIRS__UPDATE
                           SET        Run__Mode = 0
                   ELSE    continue
                   BIT__ENCODE
END DO
CLEAR__ENCODER (see A.3.8)
END PROCESS = BLOCK__ENCODE
```

A.3.3 COMPUTE - Computes the correct value to add to the Current__Value and subtract from the Width. It also determines how much of the Current__Value to shift to the Code__Block.

```
PROCESS = COMPUTE
SET       Width = Width - (2 raised to negative power of second value in Table__Pair (Number))
SET       CV = CV + (2 raised to negative power of second value in Table__Pair (Number))
SET       Compare = true
IF        Width < 1.0000
THEN      SHIFT     Width left one place
          APPEND    zero to rightmost end of Width
          SET       Shift__Left = 1
ELSE      SET       Shift__Left = 0
CHECK (see A.3.5)
TABLE__PAIRS__UPDATE (see A.3.6)
END PROCESS = COMPUTE
```

A.3.4 BIT__ENCODE - When the Encoders are not in Run Mode, Block bytes are encoded on a bit by bit basis.

```
PROCESS = BIT__ENCODE
    SET Number = 1
    SET Bit__Count = 1
    DO WHILE Bit__Count < 9
        IF      bit (Bit__Count) of Current__Byte = first value in Table__Pair (Number)
        THEN    COMPUTE
                    NEXT__TABLE__PAIR (see A.3.7)
        ELSE    SET Width = 1.0000
                SET Shift__Left = second value in Table__Pair (Number)
                SET Compare = false
                CHECK
                TABLE__PAIRS__UPDATE
                NEXT__TABLE__PAIR
        SET     Bit__Count = Bit__Count + 1
    END DO
END PROCESS = BIT__ENCODE
```

A.3.5 CHECK - The Current__Value is checked and the proper data is appended to the Code__Block.

```
PROCESS = CHECK
IF          Code__Block = null
THEN        continue
ELSE        SET         Count = 0
            ADD         bit (Count) of CV to rightmost end of Code__Block
            SET         bit (Count) of CV to 0
IF          Rightmost integral 8 bits of Code__Block = (FF)
THEN        INSERT (0) after last integral 8 bits of Code__Block
ELSE        continue
SET         Count = Count + 1
DO WHILE    Shift__Left > 0
            APPEND      bit (Count) of CV to rightmost end of Code__Block
            SET         bit (Count) of CV to 0
            SHIFT       CV left one place
            APPEND      ZERO to rightmost end of CV
            SET         Shift__Left = Shift__Left - 1
            IF          Code__Block = integral multiple of 8 bits
            THEN        IF          rightmost 8 bits of Code__Block = (FF)
                        THEN        APPEND    0000 to Code__Block
                        ELSE        continue
            ELSE        continue
END DO
END PROCESS = CHECK
```

A.3.6 TABLE__PAIRS__UPDATE - The Table__Pairs values are updated to the correct values, if required. If Mc__Count = 1111 and 1 is added to it, Mc__Count = 0000.

```
PROCESS  = TABLE__PAIRS__UPDATE
IF         Compare = true
THEN       CASE      second value of Table__Pair (Number) AND Mc__Count
           [second value of Table__Pair (Number) = 1 AND Mc__Count (3:4) = 11]
           second value of Table__Pair (Number) = 2
           [second value of Table__Pair (Number) = 2 AND Mc__Count (2:4) = 111]
           second value of Table__Pair (Number) = 3
           [second value of Table__Pair (Number) = 3 AND Mc__Count (1:4) = 1111]
           second value of Table__Pair (Number) = 4
           END CASE
           SET       Mc__Count = Mc__Count + 1
ELSE       IF        second value of Table__Pair (Number) > 1
           THEN      second value of Table__Pair (Number) = second value of Table__Pair (Number)
                     - 1
           ELSE      invert first value of Table__Pair (Number)
END PROCESS = TABLE__PAIRS__UPDATE
```

A.3.7 NEXT__TABLE__PAIR - The next Table__Pair to be used in the encoding process is determined.

```
PROCESS = NEXT__TABLE__PAIR
IF     Bit__Count < 8
THEN   IF     bit (Bit__Count) of Current__Byte = 1
       THEN   Number = Twice Number + 1
       ELSE   Number = Twice Number
ELSE   SET    Number = 1
END PROCESS = NEXT__TABLE__PAIR
```

A.3.8 CLEAR__ENCODER - The Encoder must be cleared whenever a Block encoding is complete.

```
PROCESS = CLEAR__ENCODER
SET         Zero__Count = 000
IF          Run__Mode = 1
THEN        IF          first value of Unique__Table__Pair = 0
            THEN        COMPUTE
            ELSE        SET         Width = 1.0000
                        SET         Compare = false
                        SET         Shift__Left = second value in Table__Pair (Number)
                        CHECK
                        TABLE__PAIRS__UPDATE
ELSE        continue
SET         Shift__Left = 4
CHECK
IF          Code__Block = integral multiple of 8 bits
THEN        TRAILER (see A.3.9)
ELSE        DO WHILE Code__Block ≠ integral multiple of 8 bits
               APPEND 0 to Code__Block
               SET         Zero__Count = Zero__Count + 1
            END DO
            TRAILER
END PROCESS = CLEAR__ENCODER
```

A.3.9 TRAILER - A trailer needs to be appended to each Code__Block to be used during the decoding process.

```
PROCESS = TRAILER
SET         Trailer__Byte = (00)
IF          Number of bytes in Code__Block is odd
THEN        SET         Trailer__Byte, bit 5 = 1
ELSE        SET         Trailer__Byte, bit 5 = 0
APPEND      (FF) to Code__Block
SET         Trailer__Byte, bits 6:8 to Zero__Count
IF          Block is last Block in Logical__Data__Record
THEN        SET         Trailer__Byte, bits 1:4 to 1100
ELSE        SET         Trailer__Byte, bits 1:4 to 1001
APPEND      Trailer__Byte to Code__Block
IF          Trailer__Byte, bit 5 = 1
THEN        APPEND      (00) to Code__Block
ELSE        Continue
END PROCESS = TRAILER
```