Standard ECMA-222

June 1995

ECMA

Standardizing Information and Communication Systems

Adaptive Lossless Data Compression Algorithm

Standard ECMA-222

June 1995

ECMA

Standardizing Information and Communication Systems

Adaptive Lossless Data Compression Algorithm

(ALDC)

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 optimal use of the resulting data capacity, lossless compression algorithms have been designed which allow a reduction of the number of bits required for the representation of user data.

These compression algorithms are registered by ECMA, the International Registration Authority established by ISO/IEC. The registration consists 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.

This ECMA Standard is the third ECMA Standard for compression algorithms. The two previous standards are:

ECMA-151	Data Compression for Information Interchange - Adaptive Coding with Embedded Dictionary - DCLZ Algorithm (June 1991)
ECMA-159	Data Compression for Information Interchange - Binary Arithmetic Coding Algorithm - (December 1991)

These ECMA Standards have been adopted under the fast-track procedure by ISO/IEC JTC 1 as International Standards 11558 and ISO/IEC 12042, respectively.

The present ECMA Standard has also been contributed to ISO/IEC for adoption under the fast-track procedure as an International Standard.

This ECMA Standard has been adopted by the ECMA General Assembly of June 1995.

Table of contents

1 Scope

This ECMA standard specifies a lossless compression algorithm to reduce the number of bytes required to represent data. The algorithm is known as Adaptive Lossless Data Compression algorithm (ALDC). The numerical identifiers according to ISO/IEC 11576 allocated to this algorithm are:

ALDC 512-Byte History Buffer: 3

ALDC 1024-Byte History Buffer: 4

ALDC 2048-Byte History Buffer: 5

2 Conformance

A compression algorithm shall be in conformance with this ECMA Standard if its output data stream satisfies the requirements of this ECMA Standard.

3 Reference

ISO/IEC 11576:1993, Information Technology - Procedure for the Registration of Algorithms for the Lossless Compression of Data

4 Definitions

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

4.1 Compressed Data Stream

The output stream after encoding.

4.2 Copy Pointer

A part of the Compressed Data Stream which represents a group of two or more consecutive bytes for which there already exists an identical group in the History Buffer. It comprises a Length Code Field and a Displacement Field.

4.3 Current Address

The location within the History Buffer where the Data Byte is written.

4.4 Data Byte

The current byte of incoming data which is written into the History Buffer and is compared to all data bytes previously written into the History Buffer.

4.5 Displacement Field

That part of the Copy Pointer which specifies the location within the History Buffer of the first byte of a Matching String.

4.6 End Marker

A string of 12 ONEs indicating the end of the Compressed Data Stream.

4.7 History Buffer

A data structure where incoming data bytes are stored for use in the compression and decompression process.

4.8 Literal

A Data Byte for which no match was found in the History Buffer.

4.9 Matching String

A sequence of bytes in the incoming data which is identical with a sequence of bytes in the History Buffer.

4.10 Match Count

The number of bytes in a Matching String.

4.11 Match Count Field

That part of the Copy Pointer which specifies the number of consecutive bytes for which a match was found in the History Buffer.

4.12 Pad Bits

Bits set to ZERO and included in the Compressed Data Stream, as required, to maintain an 8-bit byte boundary.

5 Conventions and Notations

5.1 Representation of numbers

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

- The setting of binary bits is denoted by ZERO and ONE.
- Numbers in binary notation and bit combinations are represented by ZEROs and ONEs with the most significant bit to the left.
- All other numbers shall be in decimal form.

5.2 Names

The names of entities are given with a capital initial letter.

6 ALDC compression algorithm

An overview of the ALDC encoding process is given in annex B, and a flow chart is contained in annex C.

6.1 Encoding description for a 512-byte History Buffer

At the start of encoding, all bytes of the History Buffer shall be reset to all ZEROs. Data bytes shall be stored in sequence in the History Buffer, starting with a Current Address of 0.

The encoder processes the incoming data stream one byte at a time. The current byte being processed is referred to as the Data Byte. When a Data Byte is received from the input data stream, it shall be written into the History Buffer at the Current Address. Then the Current Address shall be incremented by 1. If it exceeds the maximum address, which is 511 for a History Buffer size of 512 bytes, it shall be reset to 0.

Step 1

The Data Byte shall be compared with each byte previously written into the History Buffer to identify any identical bytes.

Step 2

If the Data Byte does not match any byte in the History Buffer, the process shall continue at step 6.

- If the Data Byte matches one or more bytes in the History Buffer, for every matching byte it shall be noted whether this matching byte is a continuation of a previous sequence of matching bytes or not.
- If it is not a continuation, the Displacement Field of the matching byte shall be noted and recorded as having a Match Count of one byte.
- If the matching byte is a continuation of a previous string, the Match Count for that string shall be incremented by 1.

Step 3

If a Match Count equals 271, the corresponding bytes shall be identified by a Copy Pointer, which shall be added to the Compressed Data Stream. Its Match Count Field and Displacement Field shall be specified as defined in 6.2. The next Data Byte shall then be read and the process shall continue at Step 1.

If there is no more data to be read, the process shall continue at Step 7.

Note: The value of 271 was chosen for implementation reasons.

Step 4

If the Match Count has not reached 271, any pending Matching Strings shall be checked to see if any are continued by the Data Byte.

– If none of the previous Matching Strings is continued and if any of the previous Matching Strings consists of two or more bytes, that Matching String having the lowest Displacement Field shall be identified by a Copy Pointer, which shall be added to the Compressed Data Stream. The next Data Byte shall then be read. If there is no more data to be read, the process shall continue at Step 7.

– If no previous Matching Strings are continued and the previous matches were only 1-byte matches, the previous 1-byte match shall be identified as a Literal and shall be added to the Compressed Data Stream as defined in 6.2. The next Data Byte shall then be read. If there is no more data to be read, the process shall continue at Step 7.

Step 5

If there are no Matching Strings with a Match Count of 271 and there is the continuation of at least 1 previous Matching String, the next Data Byte shall be read. The process shall continue at Step 1.

If there is no more data to be read, the pending Matching String shall be identified as a Copy Pointer, which shall be added to the Compressed Data Stream. The process shall then continue at Step 7.

Step 6

If the Data Byte does not match any bytes of the History Buffer, a check shall be made for any previous Matching Strings that may be pending.

If there are any previous Matching Strings of two or more bytes, they shall be identified by a Copy Pointer, which shall be added to the Compressed Data Stream. Then the Data Byte that did not match shall be added to the Compressed Data Stream as a Literal. The next Data Byte shall be read and the process shall continue at Step 1. If there is no more data to be read, the process shall continue at Step 7.

If there are no pending Matching Strings of two or more bytes and there are any pending 1-byte matches, the byte preceding the Data Byte shall be identified as a Literal, which shall be added to the Compressed Data Stream. Then the Data Byte that did not find a match shall be identified as a Literal, which shall be added to the Compressed Data Stream. The next Data Byte shall be read and the process shall continue at Step 1. If there is no more data to be read, the process shall continue at Step 7.

Step 7

An End Marker shall be added to the Compressed Data Stream and Pad Bits shall be included as required. This ends the encoding process.

6.2 Description of the Compressed Data Stream

As described above, the processing of the input data generates as its output the Compressed Data Stream. The completed Compressed Data Stream shall consist of:

– Literals, each preceded by a bit set to ZERO.
– Copy Pointers, each preceded by a bit set to ONE.
– An End Marker preceded by a bit set to ONE.
– Pad Bits.

Once all data has been read, the Compressed Data Stream shall be terminated by a bit set to ONE, followed by an End Marker, followed by Pad Bits.

During the encoding, if more than one Matching String of the same Match Count is found, the Copy Pointer with the lowest Displacement Field shall be used. The Match Count Field shall consist of 2, 4, 6, 8, or 12 bits, identifying Match Counts as specified in table 1. The length of the Displacement Field shall be 9 bits, 10 bits, or 11 bits for History Buffer sizes of 512 bytes, 1024 bytes, or 2048 bytes, respectively.

Table 1-Match Count Field

Setting of Match Count Field	Number of bytes
00	2
01	3
10 00	4
:	:
:	:
10 11	7
110 000	8
:	:
:	:
110 111	15
1110 0000	16
:	:
:	:
1110 1111	31
1111 0000 0000	32
:	:
:	:
1111 1110 1110	270
1111 1110 1111	271

Annex A
(informative)

ALDC encoding format

A.1 Nomenclature

The Compressed Data Stream encoding format is described using BNF-like language, the symbols being defined as follows :

Symbol	**Definition**
:=	the non-terminal on the left side of the ":=" can be replaced by the expression on the right side
<name>	non-terminal expression
[ ]	the expression inside the square brackets may occur 0 or more times
|	the logical disjunction OR
( )	the text enclosed within the parentheses is a comment only, added for clarity
0,1	the terminal binary digits ZERO or ONE

A.2 ALDC_1 Algorithm - encoding format (512-byte History Buffer)

<Compressed Data> := 0<Literal> [0<Literal> | <Copy Pointer>] | <End Marker> <pad>

<Literal> := <b><b><b><b><b><b><b><b> (8-bit byte data)

<b> := 0 | 1

<Copy Pointer> := <Match Count Field><Displacement Field>

<Match Count Field> := (the length can be 2, 4, 6, 8 or 12 bits)

Setting of Match Count Field	**Match Count in bytes**
00	2
01	3
10 00	4
:	:
:	:
10 11	7
110 000	8
:	:
:	:
110 111	15
1110 0000	16
:	:
:	:
1110 1111	31
1111 0000 0000	32
:	:
:	:
1111 1110 1110	270
1111 1110 1111	271

<Displacement Field> := <b><b><b><b><b><b><b><b><b> (9-bit)

<pad> := number of bits sets to ZERO required to maintain an 8-bit boundary

<End Marker> := 1111 1111 1111

A.3 ALDC_2 Algorithm - encoding format (1024-byte History Buffer)

(identical to ALDC_1 definition except for a 10-bit Displacement Field)

<Displacement Field> := <b><b><b><b><b><b><b><b><b><b> (10-bit)

A.4 ALDC_4 Algorithm - encoding format (2048-byte History Buffer)

(identical to ALDC_1 definition except for an 11-bit Displacement Field)

<Displacement Field> := <b><b><b><b><b><b><b><b><b><b><b> (11-bit)

Annex B
(informative)

ALDC Overview

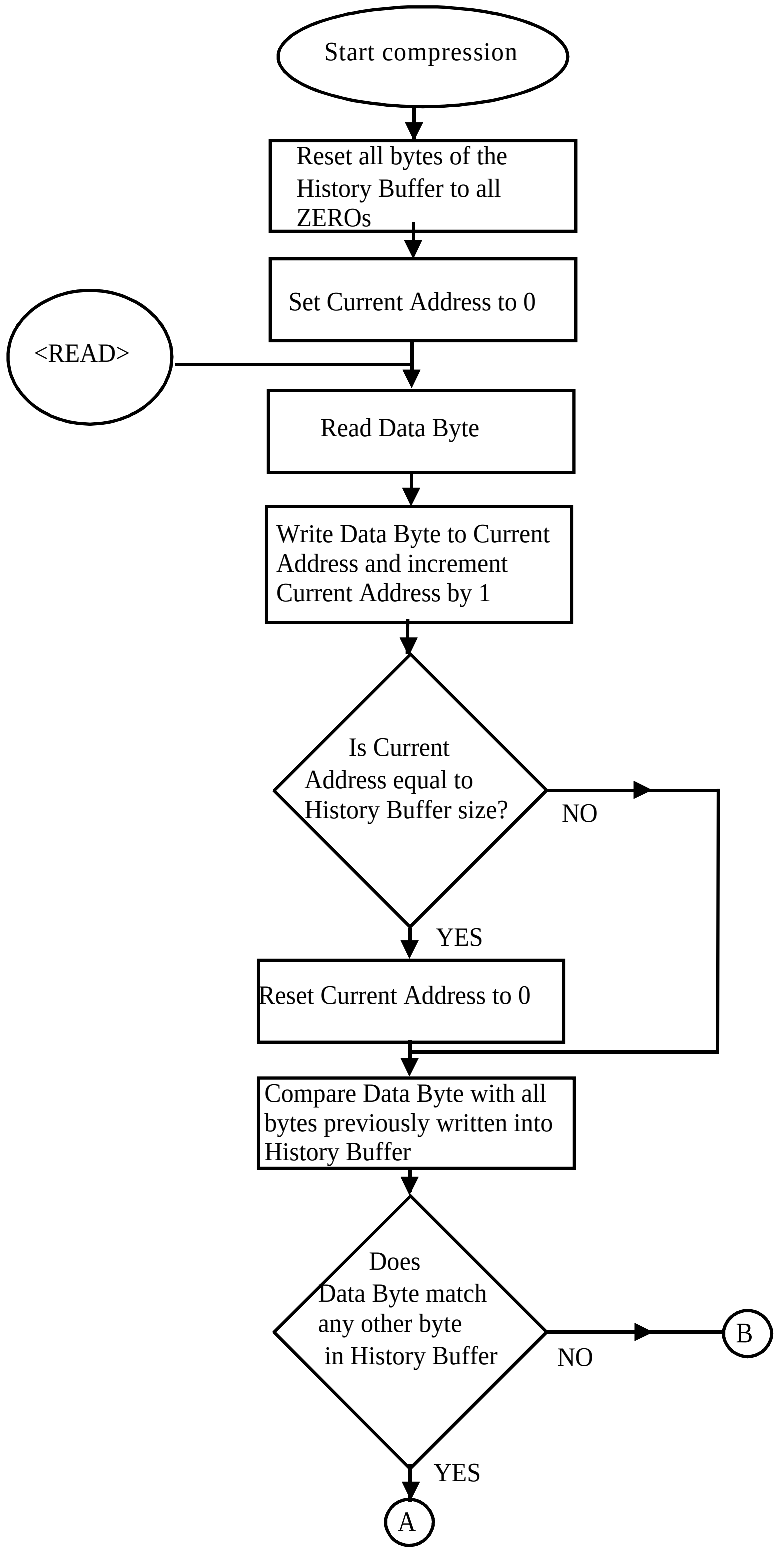
The ALDC algorithm accepts input in 8-bit data bytes and outputs a bit stream representing data in compressed form. The ALDC algorithm is one implementation of the Lempel-Ziv 1 (LZ1) class of data compression algorithms.

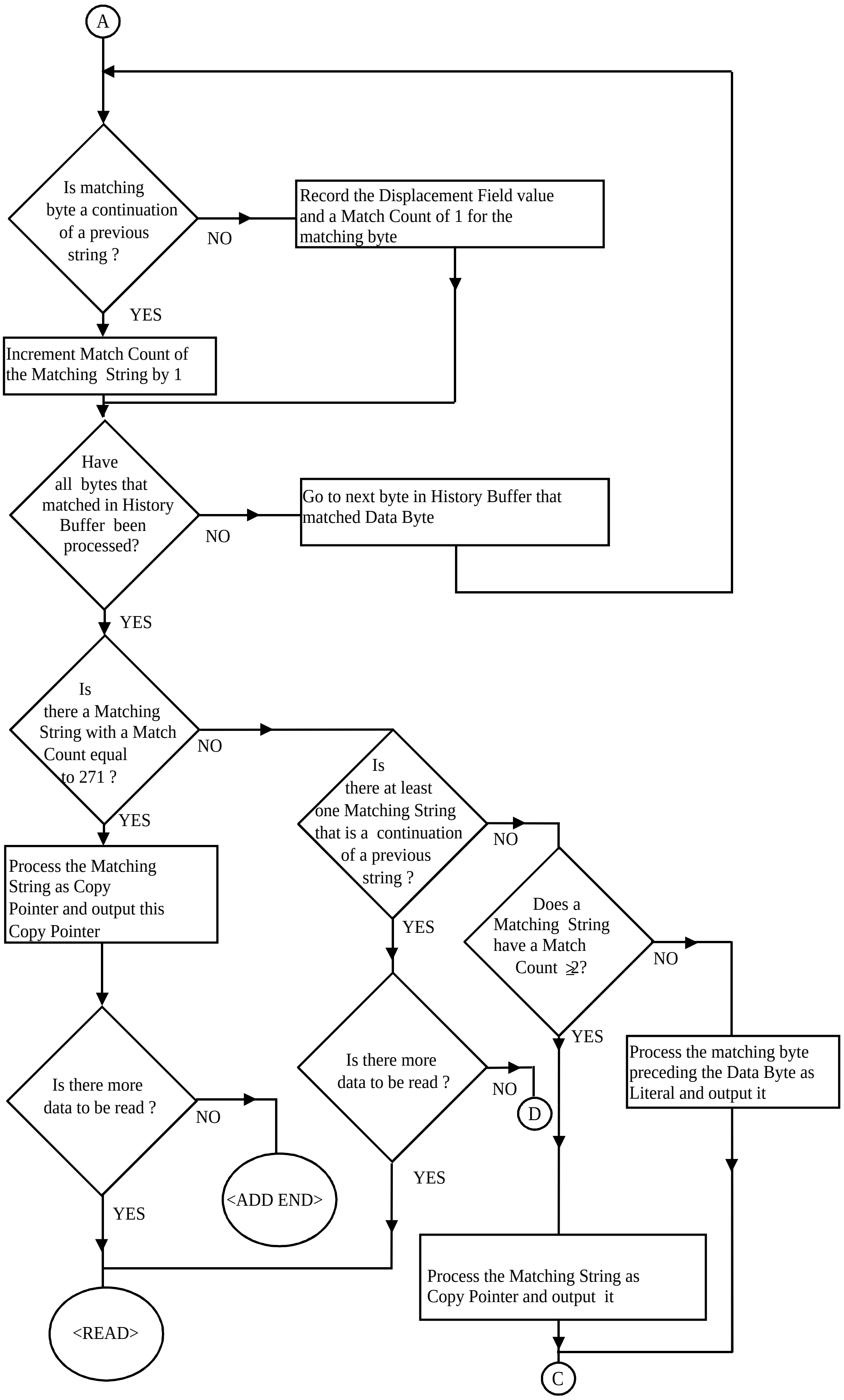
LZ1 algorithms achieve compression using a data structure called a History Buffer where incoming data is stored and compared to previous data in the same History Buffer. An LZ1 encoding process and an LZ1 decoding process both initialize this structure to the same known state and update it in an identical fashion. Consequently, these two histories remain identical, so it is not necessary to include history content information with in the compressed data stream.

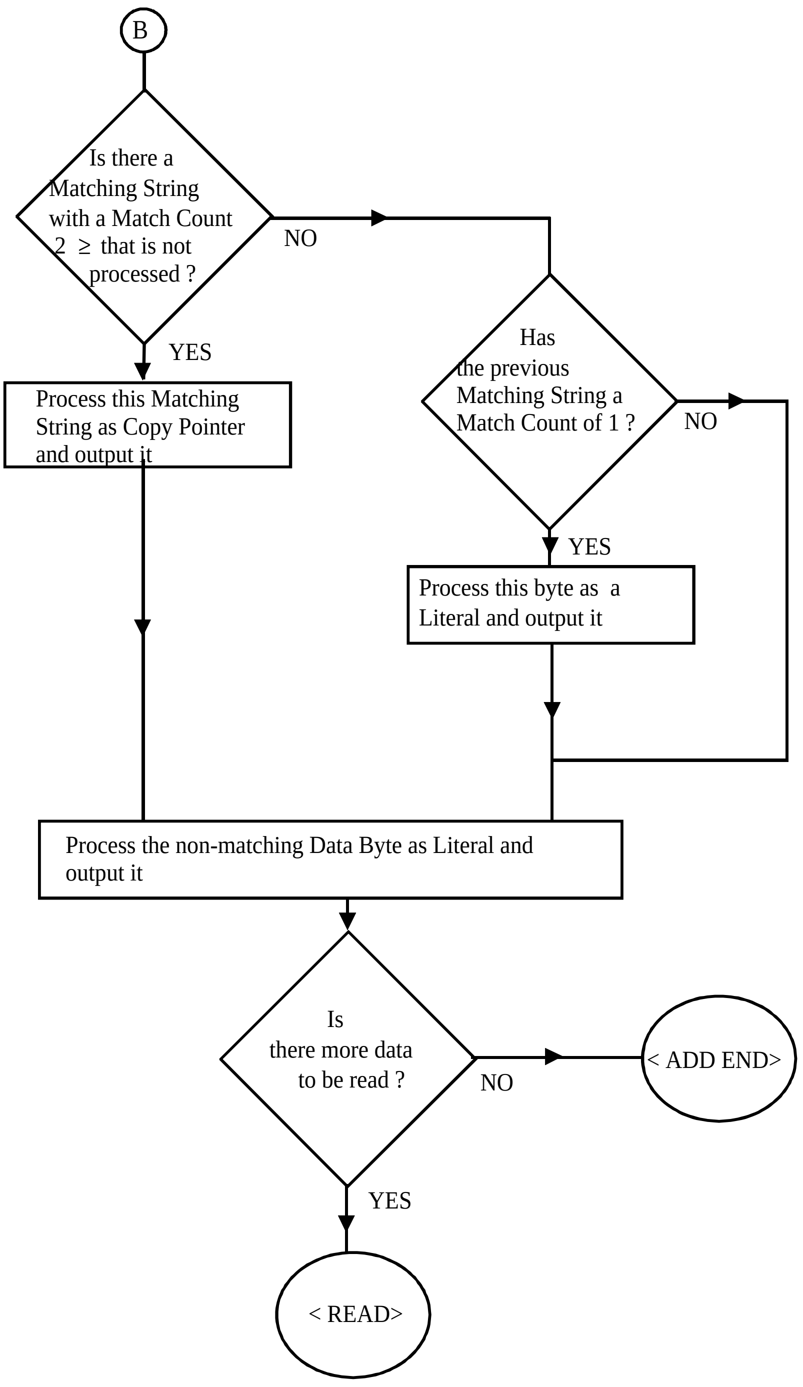
Incoming data is entered into the History Buffer. Each incoming byte is compared with all other bytes previously stored in the History Buffer. Compression results from finding sequential matches. At the beginning of the encoding process the History Buffer is empty. The result of not finding any matching bytes in the History Buffer is that the Compressed Data Stream contains only Literals. Bytes are encoded as Literals until a matching sequence of two or more bytes occurs. As the History Buffer fills, it becomes increasingly possible for the encoder to represent incoming data by encoding it as a Copy Pointer for a string already present in this History Buffer. This is the principal mechanism by which LZ1 algorithms are able to achieve compression.

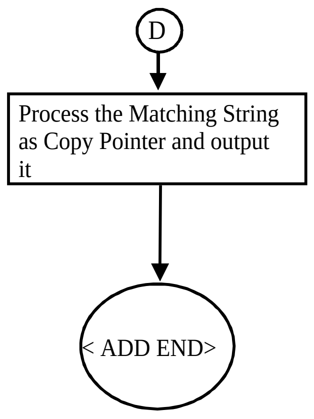
Annex C
(informative)

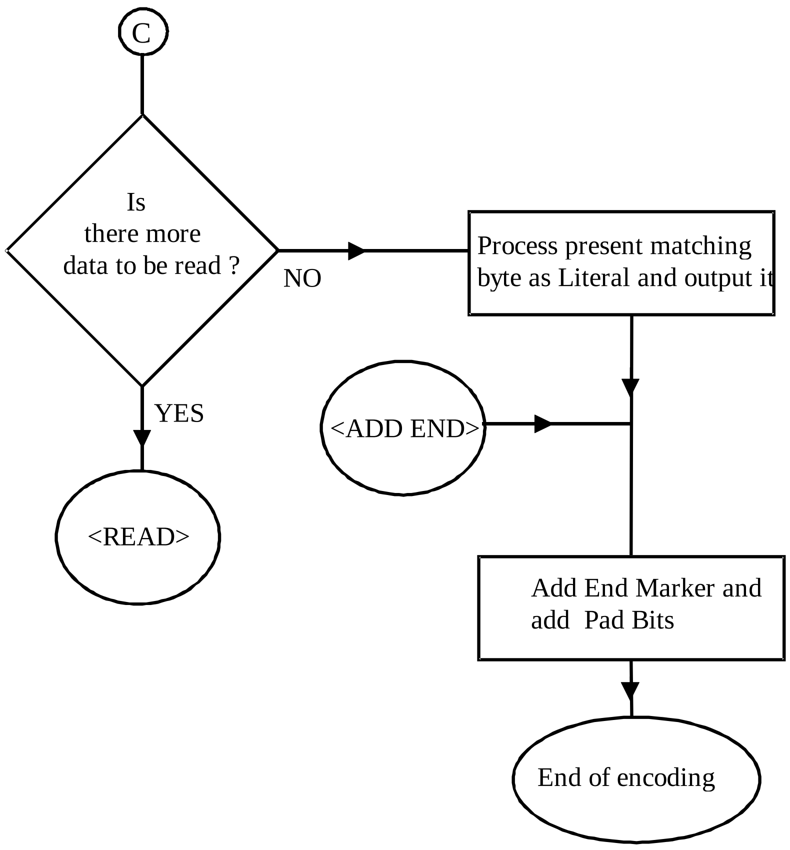
ALDC Encoding Flow Chart


Start compression
Reset all bytes of the History Buffer to all ZEROs
Set Current Address to 0
<READ>
Read Data Byte
Write Data Byte to Current Address and increment Current Address by 1
Is Current Address equal to History Buffer size?
NO
YES
Reset Current Address to 0
Compare Data Byte with all bytes previously written into History Buffer
Does Data Byte match any other byte in History Buffer
NO
B
YES
A


A
Is matching byte a continuation of a previous string ?
NO
Record the Displacement Field value and a Match Count of 1 for the matching byte
YES
Increment Match Count of the Matching String by 1
Have all bytes that matched in History Buffer been processed?
NO
Go to next byte in History Buffer that matched Data Byte
YES
Is there a Matching String with a Match Count equal to 271 ?
NO
YES
Process the Matching String as Copy Pointer and output this Copy Pointer
Is there at least one Matching String that is a continuation of a previous string ?
NO
YES
Does a Matching String have a Match Count ≥2?
NO
YES
Process the matching byte preceding the Data Byte as Literal and output it
Is there more data to be read ?
NO
<ADD END>
YES
Is there more data to be read ?
NO
D
YES
<READ>
Process the Matching String as Copy Pointer and output it
C


B
Is there a Matching String with a Match Count 2 ≥ that is not processed ?
NO
YES
Process this Matching String as Copy Pointer and output it
Has the previous Matching String a Match Count of 1 ?
NO
YES
Process this byte as a Literal and output it
Process the non-matching Data Byte as Literal and output it
Is there more data to be read ?
NO
< ADD END>
YES
< READ>


D
Process the Matching String as Copy Pointer and output it
< ADD END>


C
Is there more data to be read ?
NO
YES
<READ>
Process present matching byte as Literal and output it
<ADD END>
Add End Marker and add Pad Bits
End of encoding

95-0044-A

Annex D
(informative)

Bibliography

J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Trans. Inform. Theory, vol. IT-23, no.3, pp. 337-343, 1977.