



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

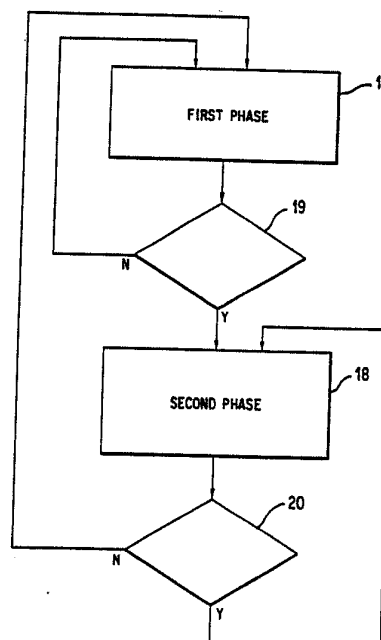
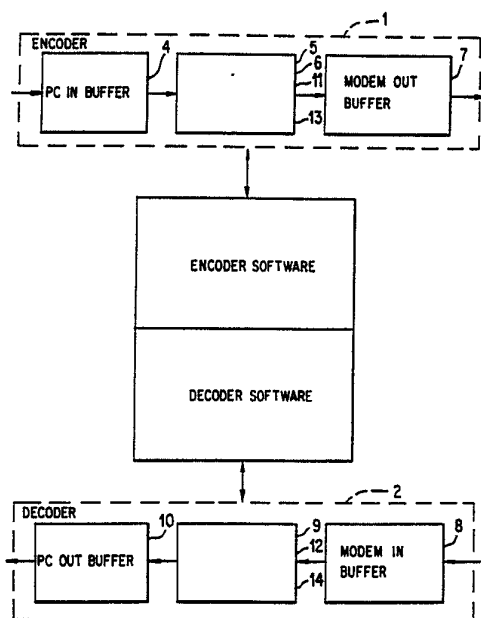
(51) International Patent Classification ⁵ : H03M 7/42, 7/30, 7/48	A1	(11) International Publication Number: WO 92/02989 (43) International Publication Date: 20 February 1992 (20.02.92)
--	----	--

(21) International Application Number: PCT/US91/05659

(22) International Filing Date: 8 August 1991 (08.08.91)

(30) Priority data:
565,155 9 August 1990 (09.08.90) US(71) Applicant: TELCOR SYSTEMS CORPORATION [US/
US]; 4 Strathmore Road, Natick, MA 01760 (US).(72) Inventors: BACON, Francis, L. ; 166 Pelham Island Road,
Wayland, MA 01778 (US). PRICE, Ernest, R. ; 30 Soren
Street, Randolph, MA 02368 (US).(74) Agents: SUNSTEIN, Bruce, D. et al.; Bromberg & Sun-
stein, 10 West Street, Boston, MA 02111 (US).(81) Designated States: AT (European patent), BE (European
patent), CH (European patent), DE (European patent),
DK (European patent), ES (European patent), FR (Eu-
ropean patent), GB (European patent), GR (European
patent), IT (European patent), LU (European patent),
NL (European patent), SE (European patent).**Published***With international search report.**Before the expiration of the time limit for amending the
claims and to be republished in the event of the receipt of
amendments.*

(54) Title: COMPOUNDS ADAPTIVE DATA COMPRESSION SYSTEM



(57) Abstract

A system for the dynamic encoding of a character stream has a single character encoder that includes a plurality of fonts, a string encoder that includes a history buffer, and an output selector that compares encodings from the single character encoder and the string encoder and selects the least cost encoding for output. The single character encoder generates and stores hash codes used for font access and the string encoder retrieves these same hash codes and uses them for history buffer access.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MN	Mongolia
BE	Belgium	GA	Gabon	MR	Mauritania
BF	Burkina Faso	GB	United Kingdom	MW	Malawi
BG	Bulgaria	GN	Guinea	NL	Netherlands
BJ	Benin	GR	Greece	NO	Norway
BR	Brazil	HU	Hungary	PL	Poland
CA	Canada	IT	Italy	RO	Romania
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland	KR	Republic of Korea	SN	Senegal
CI	Côte d'Ivoire	LI	Liechtenstein	SU ⁺	Soviet Union
CM	Cameroon	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
DE	Germany	MC	Monaco	US	United States of America
DK	Denmark				

⁺ It is not yet known for which States of the former Soviet Union any designation of the Soviet Union has effect.

COMPOUND ADAPTIVE DATA COMPRESSION SYSTEM

5

DESCRIPTIONTechnical Field

The invention relates to the field of data compression systems and particularly to apparatus and methods for compressing data signals and reconstituting the data
10 signals.

Background ArtData Compression System Requirements

Data compression systems are known in the prior art that encode a stream of digital data signals into compressed
15 digital signals and decode the compressed digital data signals back into the original data signals. Data compression refers to any process that converts data in one format into another format having fewer bits than the original. The objective of data compression systems is to
20 reduce the amount of storage required to hold a given body of digital information or to increase the speed of data transmission by permitting an effective data transmission rate that is greater than the rated capacity of a given data communication link. Compression effectiveness is
25 characterized by the compression ratio of the system. Compression ratio is herein defined as the ratio of the number of bits in the input data to the number of bits in the encoded output data. The larger the compression ratio, the greater will be the reduction in storage space or
30 transmission time.

In order for data to be compressible, the data must contain redundancy. Compression effectiveness is determined by how effectively the compression procedure matches the forms of redundancy in the input data. In typical computer
35 stored data, e.g. English text, computer programs, arrays of integers and the like, redundancy occurs both in the nonuniform usage of individual symbols, e.g. characters, bytes, or digits, and in frequent recurrence of symbol sequences, such as common words, blank record fields, and

- 2 -

the like. An effective data compression system should respond to both types of redundancy. A typical data stream contains both types of redundancy in varying portions resulting in varying statistics. An example of a data
5 stream of varying statistics is a data stream wherein "normal" English text is immediately followed by a computer program, for example source code in the "C" programming language.

To be of practical and general utility, a digital data
10 compression system must possess the property of reversibility, i.e. it must be possible to reexpand or decompress the compressed data back into its original form without any alteration or loss of information. The decompressed and the original information must be identical
15 and indistinguishable with respect to each other. In addition, it should satisfy several performance criteria.

First, the compression effectiveness should be high, and therefore the compression ratio should be large.

Second, the system should provide high data rate
20 performance with respect to the data rates provided by and accepted by the equipment with which the data compression and decompression systems are interfacing. For real time, switched network, data communications applications, preferably the rate at which data should be compressed
25 should match the output data rate from the compression system. Because it should match the output (compressed) rate, it should be higher in proportion to the compression effectiveness, typically 6:1. The higher the compression effectiveness, the faster the input data must be processed
30 to provide sufficient output data to fully utilize the capacity of the output channel. Thus high data rate performance of data compression processing is necessary to match the line speed of today's communication systems and the compression effectiveness of modern data compression
35 methods. The data rate performance of data compression and decompression systems is typically limited by the time required to perform the processing steps associated with

- 3 -

encoding each incoming character, which in turn is limited by serial processing and the speed of the compression processor. High performance for a given compression processor is achieved by a compression method that uses
5 fewer processing steps, on average, to encode each incoming character. The fewer processing steps, the higher the performance. However, complex methods are needed to achieve high compression effectiveness for data streams of varying statistics. Such methods tend to increase the
10 number of processing steps and therefore tend to reduce data compression processing performance.

Third, the system should be adaptable, that is, capable of achieving high compression effectiveness and high performance on data having a variety of statistical
15 characteristics. Many prior art data compression procedures require prior knowledge of the statistics of the data being compressed. Some prior art procedures adapt to the statistics of the data as it is received. Adaptability in the prior art processes has either been limited to a narrow
20 range of variation e.g. character-by-character encoding or has required a high degree of complexity with resultant severe penalty in data rate performance. The requirement for data compression systems suitable for use in modems in high speed data communication links is to accommodate a wide
25 range of data characteristics without prior knowledge of data statistics and achieve both high compression ratios and high data rate performance. Data compression and decompression systems and modems currently available are either not adaptable over a wide range of data
30 characteristics or are severely limited in compression efficiency or data-rate performance and so are not suitable for general purpose usage.

Finally, the system should be responsively adaptable, that is, capable of reestablishing a high compression ratio
35 quickly after the beginning of a new data file from a stream of data files, wherein each file has different statistical properties from the data in the immediately proceeding file.

- 4 -

Prior Art Systems

U.S. Patent 4,612,532 to Bacon et al., which is hereby incorporated herein by reference, discloses a system for adaptive compression and decompression of a data stream designed to compress redundancy resulting from non-uniform usage of individual symbology. The Bacon invention uses an adaptive character-by-character compression technique wherein dynamically updated "followset" tables having Huffman codes are used to encode characters, using, on average, far fewer bits per character than is required by ASCII or EBCDIC encoding. Each incoming character is encoded using information from the three preceding characters (character type, character type, character identity), i.e. (two bits, two bits, seven bits). Thus, for each incoming character, information from the three preceding characters is used to select the appropriate followset table. The Bacon invention has a high compression efficiency on a character-by-character basis and achieves high performance by using fewer processing steps, on average, to encode each character than other character-by-character encoding techniques.

U.S. Patent 4,558,302 to Welsh discloses a string search system designed to compress redundancy resulting from frequent recurrence of symbol sequences. The Welsh invention includes a compressor which compresses a stream of data character signals into a compressed stream of code signals. The compressor stores strings of data character signals parsed from the input data stream and searches the stream of data character signals by comparing the stream to the stored strings to determine the longest match. Having found the longest match, the compressor stores an extended string comprising the longest match plus the next data character signal following the longest match and assigns a code signal thereto. A compressed stream of code signals is provided from the code signals corresponding to the stored longest matches.

U.S. Patent 4,464,650 to Eastman et al. discloses an

- 5 -

adaptive string search system designed to compress redundancy resulting from frequent recurrence of symbol sequences. The Eastman invention uses the Lempel-Ziv algorithm to encode strings of characters without constraint
5 on the length of the input or output word. However, the Eastman invention suffers the disadvantage of requiring numerous RAM cycles per input character and utilizing time consuming and complex mathematical procedures such as multiplication and division to effect compression and
10 decompression. These disadvantages tend to render the Eastman invention unsuitable for on-line data communications applications.

U.S. Patent 4,730,348 to McCrisken discloses a system for adaptive compression and decompression of a data stream
15 using a combination of techniques to compress redundancy from non-uniform usage of individual symbols and frequent recurrence of symbol sequences. The McCrisken implementation uses an adaptive character-by-character compression technique described as "bigram encoding" based
20 on "pruned tree" Huffman and "running bigrams" to compress redundancy resulting from non-uniform usage of individual symbology. As part of his adaptive character-by-character compression technique, McCrisken uses a plurality of encoding tables, on-line analysis of compression efficiency,
25 an on-line table builder, a table changer and a table change code to permit rapid adaptation to changes when compressing data streams having varying statistics. McCrisken also uses a history buffer and a string substitution technique which identifies and further compresses matching strings of up to
30 eighteen characters to compress redundancy resulting from frequent recurrence of symbol sequences. Both techniques are adaptive and therefore do not need prior knowledge of data statistics. In a preferred embodiment, some of the data stream is encoded on a character-by-character basis and
35 some of the data stream is encoded with a string substitution code. McCrisken also uses protocol emulation and packet size control to improve performance. The

- 6 -

McCriskin character-by-character compression technique has a low compression efficiency and a poor data rate performance compared to the Bacon method. This is partly because the encoding tables of McCriskin's character-by-character
5 compression technique are updated on the basis of on-line explicit analysis of compression efficiency and this technique is very inefficient compared with the transposition heuristic used by Bacon to update his followset tables. McCriskin's use of a string substitution
10 technique compensates to a large extent for the low compression efficiency and poor performance of the adaptive updating of the McCriskin encoding tables. However, the processing required to perform the search for the longest list in the McCriskin is time-consuming and the search is
15 limited, in McCriskin's preferred embodiment, to the first twenty items in the list. Also, because of the McCriskin string substitution code, the longest matching string that can be encoded as such is eighteen characters long (column 14, lines 13-18). Because of these disadvantages, McCriskin
20 does not achieve as good a compression ratio as the Eastman implementation of the Lempel-Ziv algorithm which uses no character-by-character encoding of any kind. Furthermore, because of its complexity the McCriskin implementation is inherently slow.

25 James A. Storer, in his book Data Compression: Methods and Theory, Computer Systems Press, 1988, which is hereby incorporated herein by reference, discusses methods and theories pertaining to lossless data compression over a noiseless channel with serial I/O.

30 Storer describes a family of character-by-character techniques (p.20) and notes (p.21) that (i) the performance of Huffman codes has been well studied and can serve as a useful benchmark on which to judge the effectiveness of more complex methods and (ii) for several applications it will be
35 useful to combine more sophisticated techniques with Huffman codes. A dynamic Huffman codes method is discussed (p.40) in which "tries" (special tree structures - see p.15) are

- 7 -

built dynamically and maintained based on characters appearing in the data stream. Storer describes the "unseen leaf" (the equivalent of "new character" in the Bacon patent) but does not describe the floating position

- 5 characteristic of Bacon's "new character." Higher order Huffman codes are described (p.44) along with the "transposition" heuristic (p.45), correctly attributed to Bacon (p.52).

- Storer discusses in detail three on-line textual
- 10 substitution methods (p.54), all of which use dynamically updated local dictionaries. The three methods are the sliding dictionary method, the improved sliding dictionary method and the dynamic dictionary method. The local dictionary of strings is stored in a "trie" structure (p.15)
- 15 which is a tree where the edges are labeled by elements of the alphabet in such a way that children of a given parent are connected via edges that have distinct labels, all leaf nodes are labelled as "marked", and all internal nodes are labeled as either "marked" or "unmarked". The set of
- 20 strings represented by a trie are those that correspond to all root to marked node paths. The sliding dictionary method (p.64) contains within its local dictionary all strings contained within a portion of the source string defined by a "sliding window" technique well known (but used
- 25 for other purposes) in data communications systems. This method is similar to the method using a history buffer described by McCrisken except for the method of storing pointers to strings. It is a practical realization of the first of two universal data compression algorithms proposed
- 30 by Lempel and Ziv designated by Storer (p.67) as LZ1. The LZ1 algorithm works as follows. At each stage, the longest prefix of the (unread portion of the) input stream that matches a substring of the input already seen is identified as the current match. Then a triple (d, l, c) is
- 35 transmitted where d is the displacement back to a previous occurrence of this match, l is the length of the match, and c is the next input character following the current match

- 8 -

(the transmission of c is pointer guaranteed progress). The input is then advanced past the current match and the character following the current match. The sliding dictionary method can be viewed as a practical
5 implementation of LZ1 that uses fixed size pointers; instead of remembering the entire input stream the system remembers only a fixed number of characters back, and instead of pointer guaranteed progress, the system uses dictionary guaranteed progress by reserving codes for the characters of
10 the alphabet. The improved sliding dictionary method (p.67) contains a heuristic that eliminates duplicate strings. It too requires that the alphabet be added initially in the local dictionary. Storer also suggests using Huffman coding of output pointers. The dynamic dictionary method (p.69)
15 uses update and deletion heuristics that maintain a collection of strings that do not, in general, form a contiguous portion of the input stream. Various update and delete heuristics (i.e. mechanisms which provide learning capability) are described which are used to implement the
20 methods. Both the improved sliding dictionary method and the dynamic dictionary method create and maintain a dictionary that is different from the history buffer of McCrisken. Apart from the heuristic for locating the longest match (Storer's "greedy match heuristic") most of
25 the heuristics described by Storer are directed to the maintenance of pointer sets for the special dictionaries. Difficulties encountered by the use of heuristics such as "pruning" to remove "dead strings" relate also to the special nature of these dictionaries. Storer's experimental
30 data shows that sliding dictionary methods provide significantly better compression ratios than Huffman coding methods especially on spread-sheet data; the improved sliding dictionary method provides a higher compression ratio by 1% to 2% over the sliding dictionary method; and
35 the best performance of the dynamic dictionary methods is better than the best performance of the sliding dictionary and the improved sliding dictionary methods. Storer textual

- 9 -

substitution methods provide compression ratios of typically between 3-to-1 and 2-to-1 on English text and between 5-to-1 and 2.5-to-1 on programming language text.

U.S. Patent No. 4,876,541 to Storer discloses and
5 claims the AP (all-prefixes) heuristic, modifications of the LRU (least recently used) strategy, limited look ahead, and the use of the MaxChildren parameter.

Textual substitution methods achieve higher compression ratios with large files and dictionaries. However, as the
10 files and dictionaries grow, so too does the time taken to access and update them. Storer, in his patent, describes a string search data compression system that uses a sliding dictionary that is stored as a tree ("trie") structure. This approach provides fast access to dictionary entries but
15 updating the tree structure loads the processor heavily so Storer uses sophisticated update heuristics. McCrisken describes a string search data compression system that uses a history buffer. The McCrisken approach provides fast updating of the history buffer but, in this case, string
20 matching loads the processor heavily. McCrisken resolves this with arbitrary cut-off of his search process. Practical on-line, prior art, textual substitution techniques are thus limited by the trade-off between the size of the files and dictionaries on the one hand and the
25 speed of the access algorithms and update heuristics on the other. To the extent that access and update processing can be done more efficiently, i.e. faster, then larger files and dictionaries can be maintained with a corresponding improvement in compression ratios for a given data rate.

30 Disclosure of Invention

The invention provides a system for the dynamic encoding of a character stream. A preferred embodiment of the system comprises a single character encoder which includes a plurality of fonts, a string encoder which
35 includes a history buffer, and an output selector which compares encodings from the single character encoder and the string encoder and selects the least cost encoding for

- 10 -

output. The single character encoder generates and stores hash codes which it uses for font access. The string encoder retrieves these same hash codes and uses them for history buffer access. The hash codes are generated by
5 applying a CRC algorithm to a character pair and are given the name "CRC hash". The single character encoder maintains a position in a font for all characters not otherwise listed in the font, such characters herein called "new character", and four tables are maintained for the encoding of such
10 characters. The single character encoder also maintains a position in a font for a symbol representing a string, which position directly follows the position of new character in the font. Three or more consecutive like characters are represented in the history buffer by three characters only.
15 A pair encoder is provided that encodes character pairs using the font number. The pair encoder may be active at the same time as the string encoder. Two string encoding modes are provided. A switch controls activation and deactivation of string search processes based on a
20 comparison of the average bit cost of new character encoding with a predetermined value. A hash-link/hash-test table is provided in the string search encoder having entries corresponding to every second character position in the history buffer. This table uses properties of the CRC hash
25 to access matching strings in the history buffer. String match testing starts "n" characters beyond the current character where "n" is the length of the longest match found so far. Accordingly, the string search encoder, in addition to searching forward, also searches back. The string search
30 encoder discards a string match that has less than a predetermined number of characters. Linked lists of pointers to candidate strings are maintained and the end of the linked list is determined using a property of the CRC hash.

35 Brief Description of the Drawings

Fig. 1A is a block diagram and overview of the main buffers, tables and processes of the preferred embodiment of

- 11 -

the present invention.

Fig. 1B shows the two phases of the encoding process of Fig. 1A.

Fig. 2A illustrates the loading of the data stream into
5 the CC buffer.

Fig. 2B illustrates the relationships among the encoding buffers, tables and processes of the preferred embodiment of the present invention.

Fig. 3 illustrates the fonts used in the font encoder.

10 Fig. 4A shows the global (Huffman) font encoding tables.

Fig. 4B shows the Huffman Tables used for encoding New Character and for encoding String Length in Mode A.

Fig. 4C shows the Huffman Tables used for encoding
15 String Length in Mode B.

Fig. 4D shows the Huffman Tables used for encoding Zone Code in Mode A and Mode B.

Fig. 5 illustrates the font access tables used in the font encoding process.

20 Fig. 6 illustrates the generation and use of the CRC hash.

Fig. 7A illustrates the new character encoding process.

Fig. 7B illustrates the Pair Encoding, Mode A process.

Fig. 7C illustrates the String Encoding, Mode A
25 process.

Fig. 8A illustrates the use of the history buffer access tables for mode A string encoding.

Fig. 8B illustrates the use of the history buffer access tables for mode B string encoding.

30 Fig. 9 shows the start points for string searches.

Figs. 10A and 10B show the decoding logic.

Fig. 11 shows the dual processor configuration.

Fig. 12 shows the prior art processor configuration.

Fig. 13 shows a conventional two-processor
35 configuration.

Detailed Description of Specific Embodiments

The present invention in a preferred embodiment

- 12 -

combines a novel adaptive font encoding single-character compression technique with a repeat character compression technique and several novel string encoding compression techniques. It includes an adaptive font encoding process
5 that is an improved version of the efficient, high performance font encoding process disclosed by Bacon et al. in U.S. Patent 4,612,532. It includes several novel string encoding processes. It further includes a novel data compressibility trending function which is used to select
10 the most effective encoding process according to the compressibility of the data. The font encoding process and the string encoding process of a preferred embodiment share memory and processes associated with the generation of a novel "CRC hash" using a CRC algorithm, a portion of the CRC
15 hash being used as a hash code for font and dictionary addressing and another portion being used for identification. The present invention achieves superior compression ratios and superior performance over the prior art described above.

20 A copy of the source code of the preferred embodiment of the present invention, expressed in the assembly language of the Rockwell C-19 processor, is attached hereto as Appendix 1. A guide to the source code listing is given in Appendix 2.

25 A general overview of a preferred embodiment of the system is shown in Fig. 1A. The system provides full duplex operation and it is generally divided into an encoder 1 and a decoder 2 such that each contains its own set of buffers (encoder: PC In Buffer 4, Process Buffer 5, History Buffer
30 6, and Modem Out Buffer 7; decoder: Modem In Buffer 8, History Buffer 9, and PC Out Buffer 10), character fonts 11 and 12 and access tables 13 and 14. Both the encoder and the decoder are operated by control software 3 that runs on a single, shared Rockwell C-19 processor. Fig. 1A shows the
35 main tasks performed by the encoder software (Load Process Buffers, Do Font Encoding, Update Fonts, Do String Encoding, Select Least-Code Encoding, Update History Buffers, and

- 13 -

Format and Output) 15 and the decoder software (Receive Bit Stream, Interpret Escape Codes, Decode Single Characters and Strings, Load PC Out Buffer, Update History Buffer, and Update Fonts) 16. Fig. 1B shows the two phases of encoding. 5 Phase 1 processes (steps 1-10) 17, including Loading Process Buffer, Doing Font Encoding and Repeat Character Encoding and Updating Font, are performed once for each character of input. Phase 2 processes (steps 11-20) 18 including String Encoding, Selecting Least-Cost Encoding, Formatting For 10 Output, and Updating Buffers are performed, typically, when the process buffer is full. Test 19 following Phase 1 is "Process Buffer Full or Flush". Test 20 following Phase 2 is "Flush and Process Buffer not Empty". String encoding includes string encoding mode A, or string encoding mode B 15 which combines string encoding with pair encoding. The decoder performs corresponding decoding processes.

The character stream 20 enters the CC buffer as shown in Fig. 2A. The CC buffer consists of ECChar 21 which contains 256 bytes representing the most recent characters 20 from the data stream and ECCharCopy 22 which contains an identical copy of the content of ECChar. ECCharCopy is provided to remove the necessity for boundary checking in the string matching process. Fig. 2A shows string continuation for searching 23 extending into ECCharCopy. 25 ECCharCopy is contiguous with ECChar in memory. Fig. 2A also shows the next store location in ECChar 24 and in ECCharCopy 25, and old data 26.

The ECChar and ECCharCopy buffers are two of nine process buffers, shown in Fig. 2B, which operate in parallel 30 and share input and output pointers. These buffers are used by the font encoding and string encoding processes.

Fonts 31 are shown in Fig. 3. Fig. 3 shows a table of fonts 31 having 1024 font numbers 32, an FTLink field 33, an FTMatch field 34, an FTNC field (NewCharPosition) 35, an 35 FTSize field 36, and Font Character fields (6 per Font max) 37. Huffman encoding tables are shown in Figs. 4A-4D. Fig. 4A shows global (Huffman) font encoding tables including an

- 14 -

Access Table 41 having an index 42, a Font Code table 43 and a Font Bits table 44. Fig. 4B shows the Huffman Global Code (Frequency) Tables, used for encoding New Character and for encoding String Length in Mode A. The tables have 256 Table
5 Entries, a Code Length of 4-13 bits and are referenced as "Global Code High; Global Code Low" in the source code. Fig. 4C shows the Huffman Tables used for encoding String Length in Mode B. These tables have 10 table entries, a code length of 1-6 bits, and are referenced as "LengthBCode"
10 in the source code. Fig. 4D shows the Huffman tables used for encoding Zone Code in Mode A and Mode B. These tables have 32 table entries, a Code Length of 2-7 bits and are referenced as "ZoneCode" in the source code. Font access tables 51 along with a font table 31 and an input data
15 stream 52 are shown in Fig. 5. The font access table include a CRC Hash Table 53 having CRC Hash 54, a MatchVal data 55 and RoughAdr data 56. The font access tables also include an FTRough Table 59 having an index 57 and FTRough data 58. $CRC(\wedge v)=2963$, $CRC(in)=05D6$ and $CRC(\wedge d)=7DD6$
20 provide entry points 501, 502 and 503 respectively to the CRC Hash Table from the input data stream. The history buffer and history buffer access tables used for string search are shown in Figs. 8A and 8B.

The entire compound encoding process includes:

- 25 1. Repeat character encoding;
2. Font (single-character) encoding;
3. Monitoring compressibility of data stream;
4. Selecting encoding processes dynamically (mode A or mode B);
- 30 5. String encoding (longest match, Mode A);
6. String encoding (longest match, Mode B);
7. Pair encoding;
8. Anti-expansion process (mode B only);
9. Selecting and concatenating encodings having
35 fewest bits.

These processes will now be described in detail starting with font encoding.

- 15 -

Font Encoding

In a preferred embodiment of the present invention, font encoding uses a set of fonts having character symbols stored in approximate order of the frequency of occurrence of such character after the occurrence of a pair of characters with which the font is associated. For example, if the input data stream contained the words "this" and "those", then a font would exist associated with the pair of characters "th" and the font would contain the letter "i" and the letter "o". A single font consists of pointers, links, characters, etc. whose selection (font number) is based on the prior two characters in the input stream and which contains a list of historically occurring candidate characters to be matched with Encoder Current Character.

Fig. 3 shows an array of fonts. A single font is illustrated by a single row. "New Character", i.e., any character that is not otherwise listed in the table, is also assigned a position in the table in approximate order of such characters local frequency of occurrence after the occurrence of a pair of characters with which the table is associated. "New Character", is hereinbelow referred to as "NewChar" and sometimes abbreviated as "NC". Just as the occurrence of a particular character in the data stream is a font encoding event, so the occurrence of NewChar is a font encoding event. NewChar is a font encoding event wherein either the Encoder Current Character is not found in the selected font or the selected font does not exist. The value of NewChar Position is a dynamic value in the range of 0 through n (where n is the maximum number of characters per font) meaning "Character Not in Table". NewChar does not occupy a character position in the font: it is assigned a "virtual position". Fig. 3 shows how the position of NewChar is stored in field FTNC in the font. In mode A, each font includes a virtual position for a string directly following the NewChar position. In mode B, each font includes a virtual position for a "pair encoding" directly following the NewChar and includes another virtual position

- 16 -

for string encoding following the pair encoding.

Font Encoding, CRC Hash and Font Access

Font access tables are shown in Fig. 5. Fig. 6 shows how the hash pointer (RoughAdr) and the match value

5 (MatchVal) are derived from the CRC hash.

Font encoding includes the following steps:

1. Computing a CRC hash using a CRC algorithm applied to the prior two characters;
2. Using a portion of the CRC hash (RoughAdr) as a
10 rough selector for a linked list of fine entries and using the remaining portion of the CRC hash (MatchVal) to identify a font;
3. Determining and storing the position of the current character in the selected font;
- 15 4. Selecting a global Huffman table according to the current size of the font. FTSIZE from Fig. 5 is used to enter the Access Table of Fig. 4A.

The Font Encoding process occurs once for each character of input data. Fig. 6 shows the data stream
20 including the current character to be encoded "N" and its two predecessors "P" and "S". Encoder Current Character "N" is the most recent character from the input stream which is being processed by the font encoder. At the end of each encoder cycle "Encoder Current Character" becomes Char1Prior
25 and the fetch and encoding process continues with the next character from the input stream as the new Encoder Current Character. In the example of Fig. 6, in the input data stream, 61, Encoder Current Character is "N", Char1Prior (character immediately prior to Encoder Current Character)
30 is "P" and Char2Prior is "S".

After the initial value of the CRC hash is seeded to zero, the CRC hash for the two prior characters ("S" and "P") is created as follows. A CRC function (CCITT polynomial $x^{16} + x^{12} + x^5 + 1$) is performed on the character
35 S and then on P yielding 65 a sixteen-bit CRC result (64 see Fig. 6) (herein below referred to as "the CRC hash" indicative of its function in the present invention).

- 17 -

The CRC hash is used as follows:

- a) The ten least significant bits of the CRC hash are extracted and stored as RoughAdr (62 see Fig. 6) for use as a hash pointer.
- 5 b) The six most significant bits of the CRC hash are extracted and stored as MatchVal (63 see Fig. 6) to be used as a match value with the selected font.
- 10 c) The CRC hash is also stored for later use in constructing hashes for string encoding.

The CRC hash has two very important properties:

- i) Its ten least significant bits provide a hash code having excellent statistical properties for use in hashing.
- 15 ii) The sixteen-bit result produced by every possible two-byte combination is unique. No two-byte combination shares a sixteen-bit result with another two-byte combination so the sixteen-bit result may be used to provide one-to-one mapping
- 20 with the original two bytes.

Accordingly, the ten least significant bits may be used as a hash code to access a table and the remaining six bits may be used to test if this is the specific font assigned to that exact character pair. The CRC hash is used in font

25 encoding and for history buffer access in string encoding mode A and string encoding mode B. It provides significant benefit in reducing the average amount of processor time consumed in accessing the fonts and history buffer, thereby enabling a given processor to handle higher encoding

30 throughput rates. The use of the CRC hash, as described herein below, by virtue of the throughput rate benefits, also provides a practical realization of trigram font encoding. The combination of the CRC hash and MatchVal will always identify uniquely the font associated with the prior

35 two characters.

We found experimentally that the use of all sixteen bits of the prior two characters to identify a font gives an

- 18 -

8%-12% improvement in font encoding compression efficiency on "normal" English text when compared with the Type/Type/Prior Character method described in U.S. Patent 4,612,532 to Bacon et al. We also found experimentally that
5 use of ten bits from the CRC hash, in the manner described hereinabove, produces less synonyms and therefore reduces execution time. This benefit is achieved because less time is spent linking through the fonts via the FTLink fields (see Fig. 5).

10 Huffman Encoding Tables

Fig. 4A shows a set of global Huffman tables and the associated access table. The Access Table 41 is indexed by Font Size 42 and contains pointers to the several Huffman tables for Font Code 43 and Font Bits 44 (the bit cost of
15 the encoding). The Access Table is "Encoding Table" in the source code. Index 0, and the corresponding Font Code (1,0) and Font Bits (1,1) are not used. ECFontIndex is computed and stored during font encoding. Later, during string encoding, FontBits is retrieved and, during the output
20 process, FontCode is retrieved. Figs. 4B, 4C and 4D each show a single Huffman table. Fig. 4B shows the table used for new character encoding, for string length encoding and for repeats encoding. Fig. 4C shows the tables used for string length, mode B encoding. Fig. 4D shows the table
25 used for the zone portion of string address encoding, mode A and mode B.

Font Encoding, Example 1, Finding the Current Character in the Current Font

Referring now to Fig. 5, let us consider the encoding
30 of the following string:

"^Veni,^Vidi,^Vinci.^A^do"

In this string the caret character "^" has been substituted for the space character " " to reduce ambiguity. Fig. 5 shows the static state of the Font Encoding and
35 Access Tables directly after processing the string. Beginning at an initial state having empty fonts, the process of encoding the first character proceeds as follows.

- 19 -

1. Initialization and Assignment of the first Font.

As described above, each new character to be encoded is associated with a CRC hash. The ten least significant bits of the CRC hash 56 are used as a pointer to the

5 ECFTRoughTable 59 (Encoder Font Rough Table). Since all fonts are empty at the outset, the ECFTRoughTable is initially null indicating the need for new font creation. A font number is assigned and stored in the ECFTRoughTable in the position pointed to by the hash ("000" in the example

10 given in Fig. 5). This font number is either the next available not-in-use font or an old font selected as described later.

The newly created font is initialized as follows:

	FTLink	= NULL
15	FTMatch	= MatchVal from CRC calculation
	FTNC	= 0 (Most frequent)
	FTSize	= 1
	First Font Character	= Encoder Current Character
	Other Font Characters	= N/A

20 Following table reset, the first character to be processed is the "^". The prior two characters and the CRC are assumed to be 0. Thus a MatchVal and ten-bit RoughAdr of 0 are used. This points to FTRoughTable entry number 0 (which was initially null) and font number 1 was assigned.

25 Font number 1 was initialized as specified above and has not changed since, as indicated by Fig. 5.

2. Finding the Current Font and the Current Character in the Font

a. The current font is accessed as follows. When "e"

30 becomes the current character, a CRC hash is performed on "^" and "V". The result is hexadecimal 2963 (third row of the hash table in Fig. 5) giving a MatchVal of 28 and a RoughAdr of 163.

b. The RoughAdr of 163 is used to enter the FTRough

35 Table and yield the font number 0003, the font to be tested to determine if it is the font "^V".

c. To test if the selected font is the font "^V",

- 20 -

MatchVal is compared with the FTMatch from the selected font. If these are equal, the font is searched for the occurrence of Encoder Current Character.

3. Storing the Current Character

5 a. If the current character is found, its position, the size of the font and other pertinent data are stored in the process buffers for later use by the encoding selection process. The character matching Encoder Current Character is promoted towards the top of the table (higher frequency)
10 by exchange with the next higher frequency entity (character or NC).

b. If the current character is not found, it is added to the table in the next available position (overwriting the last character when the table is full) and the table size is
15 incremented (if not full). The NC value is promoted one position towards the top of the table unless already at the top (highest frequency).

Font Encoding, Example 2, Finding the Current Font Using the Link Table

20 If FTMatch does not equal MatchVal, FTLink is examined. If FTLink is null, then the Ftlink field is assigned the next font number and the flow joins step 1 above for the creation of a new font. If FTLink is not null, control proceeds to FTMatch comparison in step 2 with the FTLink
25 field as the new font number. Linking and match comparison repeat until either the desired font is found or a new one is created.

The last line of the input data stream in Fig. 5 details the "o" character from the sequence "A^do". The
30 calculated CRC hash for "^d" is 7DD6 which yields a MatchValue of 7C and a RoughAdr of 1D6. Note that the sequence "in", seven characters earlier, produced a CRC hash of 05D6, MatchValue of 04 and RoughAdr of 1D6. Access to entry 1D6 in the ECFTHashRough Table yields a pointer to
35 font number 000C but comparison of the FTMatch field in font 000C does not equal the desired value of 7C. At that point in time, the FTLink field of font number 000C was set to

- 21 -

NULL. Consequently, font number 0013 was assigned, set to initial state and the character "o" was added to it. A future occurrence of the sequence "^d" can link to font number 0013 via font number 000C and search for or add 5 characters as required.

Font Encoding, Example 3, New Character

When a character is encountered in the data stream that does not appear in the font defined by its prior two characters, it is encoded using one of four frequency 10 encoding tables.

Fig. 7A shows the encoding of character "w" which follows, in the character stream 701, "No". As shown in Fig. 7A, looking at Font (No) 702, "w" does not appear, and NC 703 = 2, indicating that "New Character" has a virtual 15 position between the position of "v" and the position of "n" in the font. Also Font (No) contains four characters so SZ 704 = 4. The two Global Font encoding Tables shown in Fig. 7A 710 are two of the tables from Fig. 4A, corresponding to font size SZ = 4 (from Font (No)) + 2 (for NC and ST in mode 20 A) 713 or SZ = 4 + 3 (for NC, PE and ST in mode B) 714. Mode A font size = (SZ) + 2. Mode B font size = (SZ) + 3. Position "2" 715 in these tables yields 709 the bit string "000" in the Global Font Encoding Tables 710 for either Mode A 711 or Mode B 712. String "000" will be transmitted by 25 the encoder and will be recognized by the decoder as the "new character escape". This will indicate to the decoder that the next bits to be received are the encoding of a new character. In a preferred embodiment, there are four NC to Frequency Encoding tables 705, identified as 00, 01, 10 and 30 11. Bits 5 and 6 706 from the prior character (in this example "o", and "o" = 6F in hexadecimal) are used to select one of these four NC to frequency tables (in this example NC to FreqTable 11 707). The binary value of "w" (77 in hexadecimal, 708 in Fig. 7A) is used to enter the selected 35 NC to Frequency table, yielding a position (or frequency) of 15, which defines an entry into the Global Code High/Low Table 716. This table in turn, yields the Huffman code

- 22 -

01111, the font encoding of new character "w" following "No". The output bit stream sequence 717 is therefore 000 (font) followed by 01111 (frequency). The use of four tables, instead of the one table described in U.S. Patent 4,612,532 to Bacon et al, is found to improve compression efficiency. Of course, more or less than four tables could be used.

Process Buffers

The process buffers, shown in Fig. 2B, consist of nine "First In/First Out" buffers 201-209, each having 256 locations, which operate in parallel and share input and output pointers. These buffers are used by the font encoding and string encoding processes. Fig. 2B shows the flow of font encoding data among the process buffers and various tables. The contents and significance of the several buffers are as follows:

The ECChar buffer 203 contains the most recent 256 characters from the input stream to be encoded. Characters are received singly from the input stream, placed in rotation in ECChar, font encoded, and later string encoded. Least-cost selection and output formatting follow. The value range of ECChar is 0 - 255.

The ECCharCopy 202 buffer contains an exact copy of the ECChar buffer. It is contiguous with ECChar to facilitate string searching. The value range of ECCharCopy is 0 - 255.

ECType 209 is a steering value which is set by the font encoding and/or the string encoding process. ECType is used by the output format process to control the output bit stream. ECType may have any one of the following values:

- 0 - String or pair continuation (the second or subsequent character of a mode A string or a mode B string or the second character of a pair encoding).
- 2 - Font encoding. The encoding is the relative offset of the character in the selected Font.
- 4 - New character.
- 6 - First character of a pair encoding.

- 23 -

8 - First character of a string encoding.

ECFontIndex 207 is the zero relative index into the FontCode or FontBits tables for this character. By using the value of ECFontIndex as an index, either the encoding size in bits or the actual encoding bit pattern can be accessed quickly. The value range of ECFontIndex is 2 - 43 as shown in Fig. 4A.

ECFrequency 208 is the frequency value of the character. It is obtained by using the character as an index into the NC to FreqTables (Fig. 7A). The value range of ECFrequency is 0 - 255.

ECHashRaw0 2040 contains the eight least significant bits of the CRC hash computed from the prior two characters in the input stream. The value range of ECHashRaw0 is 0 - 255. Data is shown in hexadecimal in Fig. 2B.

ECHashRaw1 2041 contains the eight most significant bits of the CRC hash computed from the prior two characters in the input stream. The value range of ECHashRaw1 is 0 - 255.

ECHashX20 2050 contains the eight least significant bits of zero relative font number multiplied by two. This value is maintained for quick access to the ECFTHashNext table. The value range of ECHashX20 is 0 - 254, even numbers. Data is shown in hexadecimal in Fig. 2B.

ECHashX21 2051 contains the eight most significant bits of zero relative font number multiplied by two. This value is maintained for quick access to the ECFTHashNext table. The value range of ECHashX21 is $0 - ((\text{MaxFontTable} - 1) * 2) / 256$.

ECNewIndex 206 is the zero relative index into FontCode or FontBits representing the New Character position in this Font. The value of ECNewIndex is derived from Font Size and font-relative new character position. (During font encoding, ECNewIndex is computed and stored. Later, during string encoding, FontBits is retrieved and, during the output process, FontCode is retrieved. See Fig. 4A.) Similarly for pair encoding and/or string escapes, the value

- 24 -

of ECNewIndex is incremented by 1 or 2 and the bit cost or pattern quickly determined. The value range of ECNewIndex is 2 - 41.

ECRepeats 201 is the count of repeats of this character beyond two. That is, the two prior characters are the same as this one. The buffer pointer will not advance as long as subsequent input characters remain the same and ECRepeats is less than or equal to 255. The value range of ECRepeats is 0 - 255.

10 Font Encoding Process Flow

Font encoding process flow is shown in Fig. 1B, first phase, steps 1 through 10. Font encoding and font update processing are performed in steps 1 through 10. This series of steps occurs once for each character of input. In this process, known as "refill", a character is added to the process buffer and the current input pointer is advanced by one. The steps (shown in Fig. 2B as S1, S2, S3, etc. corresponding to step 1, step 2 step 3, etc.) are as follows:

- 20 1. A character from the input stream is fetched and stored in the current input ECChar field 210.
2. The same character is stored in the current ECCharCopy field 211. (The relationship of ECChar and ECCharCopy is shown in Fig. 2A).
- 25 3. The current character is compared with the two prior characters in the input stream. If equal, the ECRepeats field is incremented (e.g. 212 in Fig. 2B) and, if the ECRepeats field is less than or equal to 255, flow proceeds to step 1 above.
- 30 This loop insures that no more than three consecutive like characters are stored in the history buffer (except when the number of consecutive like characters exceeds 258).
4. The CRC hash is computed on the two prior
- 35 characters in the input stream (as described under "Font Encoding, CRC Hash and Font Access" hereinabove) and the result is stored in the low

- 25 -

- and high bytes of EChashRaw 213 for later use.
5. The appropriate font 214 is accessed or created (as described hereinabove). If the font exists, the font number from FT Rough Table 215 is stored in the EChashX2 table high and low bytes (217 and 216) and the character fetched in step 1 above is looked up in the font 31. If the font is created (new font), EChashX2 is set to NULL.
6. Using SZ (the number of characters in the font) from the accessed font, the Access Table of Fig. 4A 41 is accessed for a pointer 218 to be used as an index value. Neither the FontCode or the FontBits tables are used at this time.
7. The index value fetched in step 6 is added to the NC (NewChar position) 219 from the font accessed in step 5. The result 220 is stored in the ECNewIndex for later use as a NewChar or String Escape. If the current character (from step 1) was not found in the accessed font, the ECType field is set to 4 denoting a NewChar encoding.
8. If the current character (from step 1) was found in the accessed font, the raw position 221 of that character in the font is added to the index value 222 fetched in step 6 and the result 223 stored in the current ECFontIndex field. If the character position is greater than or equal to the NC (NewChar) value from the font, the ECFontIndex field is incremented by two if in Mode A and 3 if in Mode B allowing for the virtual positions of the NewChar, Pair Encoding and/or String Escapes. The ECType field is set to 2 224 denoting that the character was found in the font, a "Font Encoding".
9. If in Mode B or if the current character (from step 1) was not found in the font, the appropriate one of four ECNCFrequency tables 225 (selected from bits 5 and 6 of the immediately prior

- 26 -

character) is selected (Fig. 7A). The frequency value 226 corresponding to the current character is fetched from the selected table and stored in the current input position of the ECFrequency field 227. This is for later use as a new character encoding or for 8-bit output in antiexpansion mode.

10. The current input pointer 228 into the process buffer is incremented by one. If the number of characters in the process buffer array is now 256 or, the Font Trending Switch changed from Mode A to B (or vice versa), or a timer-initiated flush occurs, flow proceeds to step 11 below for string processing and output. Otherwise flow proceeds to step 1 above.

Steps 11 through 20, including string search, least cost encoding selection and output are described hereinbelow under "Second Phase Processing".

Font Reallocation:

As input context changes, old fonts go out of use and new ones are created. Since there is a limit to the number of practical (actual) fonts in a preferred embodiment (e.g. 1024), a method for reassigning fonts is required. In the preferred embodiment this is a circular (low to high then back to low) replacement heuristic. An alternative embodiment may also include a "less recently used" heuristic. The next three paragraphs describe the combination. (The source listing of Appendix 1 details the circular heuristic only).

Since the fonts are linked in chains starting at FTRoughTable and forward-only linked via FTLink, the circular reallocation process points into the FTRoughTable advancing from 0 through 1023 and back to 0. The selected font, and subsequently linked fonts (if any) as indicated by FTLink are examined for potential reuse.

Each time a font is accessed by the previously described Font Encoding Process, an unused bit of the

- 27 -

FTHashNext field is set to 1 indicating activity. As the reallocation process traverse the fonts, it will reset the activity bit if it is set and link to the next candidate font. If the activity bit is reset, the font will be
5 reallocated as a new font. By the use of the single activity bit, any given font has the opportunity to survive permanently provided that it is used at least once per pass of the reallocation search process.

For example, referring to Fig. 5, assume that the main
10 reallocation pointer is pointing to the FTRoughTable at hexadecimal 1D6. The FTLink field of font 000C will be examined for the activity bit. Assuming it to be reset, font 000C will be the next assigned for a new font. This is done by copying the contents of the FTLink field (in this
15 case 0013) into the FTRoughTable at 1D6 thus freeing font 000C. The reallocation pointer is moved to 0013 for use in the next allocation cycle.

String Encoding

The string encoder of the present invention uses a
20 circular history buffer to store a sliding dictionary. The history buffer is a dictionary of all the strings it contains. String encoding may operate in one of two modes, mode A (using the tables in Fig. 8A) for use on relatively compressible text or mode B (using the tables in Fig. 8B)
25 for use on less compressible text. In both modes, string encoding is designed to achieve near-optimum compression efficiency under the time constraints of on-line operation. The history buffer is tagged at regular intervals and, in a preferred embodiment, is tagged every second character
30 position. The string encoder of the present invention also uses a novel dictionary access structure having a set of tables for accessing the history buffer. Updating the history buffer involves very little processing because it involves no more than accepting the next character and
35 incrementing a pointer. However, updating the dictionary access structure is as challenging a problem as updating the sliding dictionary in string encoding systems which store

- 28 -

the sliding dictionary as a tree structure. The present invention addresses this problem by the use of a novel history buffer access method. The method is based on the structure of the history buffer access tables as shown in Figs. 8A and 8B and it retrieves and uses the same CRC hash codes created and used in the font update process during font encoding. Accordingly, by use of this method, updating of the dictionary access structure is faster and requires less processing than updating a tree structure would require.

The use of a tagged history buffer provides additional benefit for accessing and matching strings. String encoding mode A, using a tagged history buffer, locates longer strings in a shorter time than earlier methods. While the process searches the same number of candidates, the process encounters shorter linked lists in the access buffers than would otherwise occur. Processing time spent building and searching access tables is beneficially reduced.

The history buffer/dictionary access structure, in a preferred embodiment, includes a history buffer and access tables. The history buffer and the access tables shown in Figs. 8A and 8B are used by the string encoding process of mode A and the string encoding process of mode B respectively. Both Figs. 8A and 8B show an ECRR (History) Buffer (1 byte wide) 81 with a Next Available Buffer Position 82 and an ECRR Suffix (256 positions) 83. Both Figs. show an ECRR Hash Head Buffer (2 bytes wide) 84. Both Figs. show an ECRR Hash Buffer containing an ECRR Hash Link portion (2 bytes wide) 85 and an ECRR Hash Test Portion (2 bytes wide) 86. Both Figs. show the derivation 87 and 88 of the CRC hash used as entry to the ECRR Hash Head Buffer 84.

Both string encoding mode A and string encoding mode B use the CRC hash created earlier during font encoding and stored in the ECHashRaw table (see Fig. 2B). However, each of these processes uses the CRC hash in a slightly different way. String encoding mode B uses the CRC hash (a hash based on two consecutive characters) directly. String encoding

- 29 -

mode A uses a novel algorithm (which includes the CRC hash) to create a hash based on two consecutive pairs of characters (four consecutive characters) as illustrated by the following example for the four characters "THEY":

5 "TH" [CRC hash] yields XXXX (16 bits)

 "EY" [CRC hash] yields YYYY (16 bits)

 XXXX \oplus (0-YYYY) yields ZZZZ (16 bits)

 where \oplus is exclusive OR, 0-YYYY is zero minus YYYY and ZZZZ is the resultant hash.

10 String Encoding, Mode A

 Every second character position in the history buffer is tagged and the tags are used to index the string search process. Each tagged position has corresponding Hash Link and Hash Test field. String encoding for mode A includes

15 the following steps:

1. Set LookAhead = 3 (Fig. 9 shows a Data Stream 91, a History Buffer ECRR 92 with a Next Available Buffer Position 93, A CC Buffer 94 with a Current Character 95, and a Pointer "p" 96. The pointer 96 is shown for Mode A to have a First Start Point for String Search 901 displaced 3 characters from the position of the current character and a Second Start Point for String Search 902 displaced 2 characters from the position of the current character.) Set pointer p to CCBuffer pointer (ECNextChar pointer in Fig. 2) plus a number of characters equal to LookAhead.
2. Create the hash for the string of four characters starting at the "p"th character as described hereinabove.
3. Use the least significant eleven bits of the hash (ZZZZ in the example above) as a pointer (e.g. 1811 in Fig. 8A) to enter ECRR Hash Head Table of Fig. 8A. Set pointer "h" to the first potential match by using the contents of ECRR Hash Head field (e.g. 7300 in Fig. 8A) to point to the most recent four-character string in the history

- 30 -

buffer, starting at a tagged location, that hashes to that same hash.

4. Find the longest match:

- a) Set $n = 3$
- 5 b) Set $x = 0$
- c) Compare (one character at a time) the character at $(p + n - x)$ in the CC buffer with the character at $(h + n - x)$ in the history buffer, incrementing x by 1 until
10 $x = n$ or no match. The "fast reject step" is when $x = 0$.
- d) Increment n by 1 and compare the character at $(p + n)$ in the CC buffer with the character at $(h + n)$ in the history buffer until no
15 match.

Continue to search for the longest match as follows. Use pointer "h" to enter the ECRRHashLink table (at 7300 in Fig. 8A). Reset pointer "h" from the content of the ECRRHashLink
20 table so that pointer "h" points to the next most recent four-character string in the history buffer (7284 in Fig. 8A). In each search, using steps b through d above, begin comparing characters for match starting at character n , where n is the
25 length of the current longest match. Continue until the end of the linked list, as indicated by a non-match of the hash with the corresponding entry in the ECRR Hash Test field or, to prevent looping, until MaximumASearches (eight in the
30 preferred embodiment) have been performed. Store length and location of longest match if n (length of longest match) > 3 .

5. Backmatch, as follows, to maximize the length of the string:

- 35 a) First time through (LookAhead = 3), check until no match: character preceding 1st character, the character preceding that and

- 31 -

then the character preceding that (the current character).

- 5 b) Within repeat steps (from step 6, LookAhead =
 2) check until no match: character preceding
 first character and then the character
 preceding that (the current character).
6. Repeat steps 1-5 with LookAhead = 2.
7. Select from the outputs of steps 5a and 5b the
 string which:
- 10 a) is the longest;
 b) if the strings are equal, the one that is
 most recently stored.

The following advantages follow from the structure and method of string encoding mode A:

- 15 a) History buffer update processing time is reduced
when the history buffer is accessed at fewer entry points
than every character. In the preferred embodiment, the
history buffer update processing time is reduced by a factor
of two because the history buffer access table update takes
20 place every second character instead of every character.
- b) String search processing time is reduced when the
history buffer is accessed at fewer entry points than every
character. In the preferred embodiment, the linked list to
be searched is, on average, only one-half the size it would
25 otherwise be (the list is drawn from a population of
candidates only one-half the size it would otherwise be).
- c) Less memory is required for the ECRR Hash-
Link/Hash-Test Table because, in the preferred embodiment,
it is only one-half the size it would otherwise be.
- 30 d) The end of the linked list is determined
dynamically by comparing the current hash code with the
content of the ECRR Hash Test field. Thus the need to
maintain end of list pointers or link length pointers or the
like is eliminated. Because the end of the linked list is
35 determined dynamically, no maintenance is required for the
overwritten string.
- e) Non-matches are eliminated faster and with fewer

- 32 -

processing steps because each search starts at $p + n$. This "fast reject" technique ensures that the candidate string is rejected immediately if it cannot be at least one character longer than the previous longest match.

5 String Encoding, Mode B

Every second character position in the history buffer is tagged and the tags are used to index the string search process. Each tagged position has corresponding Hash Link and Hash Test fields. String encoding for mode B includes
10 the following steps:

1. Set pointer p to CCBuffer pointer + 1 (Fig. 9 shows a Data Stream 91, a History Buffer ECRR 92 with a Next Available Buffer Position 93, a CC Buffer 94 with a Current Character 95, and a
15 Pointer "p" 96. The pointer 96 is shown for Mode B to have a First Start Point for String Search 903 displaced 1 character from the position of the current character and a Second Start Point for String Search 904 coincident with the position of
20 the current character).
2. Retrieve the CRC hash from ECHashRaw (Fig. 2B) for the string of two characters starting at the "p"th character.
3. Use the least significant eleven bits of the (16
25 bit) CRC hash as a pointer (2048 positions) to enter ECRR Hash Head Table (0012 in Fig. 8B). Set pointer "h" to the start of the first potential match by using the contents of ECRR Hash Head field (0006 in Fig. 8B) to point to the most
30 recent two-character string in the history buffer, starting at a tagged location that hashes to a CRC hash that has the same least significant eleven bits (ZQ in Fig. 8B).
4. Find the longest match having three or more
35 characters:
 - a) Compare the character at $(p - 1)$ in the CC buffer with the character at $(h - 1)$ in the

- 33 -

history buffer and terminate if no match.
This is the "fast reject step".

b) Set $n = 0$

c) Compare (one character at a time) the
character at $(p + n)$ in the CC buffer with
the character at $(h + n)$ in the history
buffer, incrementing n by 1 until no match.

Continue to search for the longest match as
follows. Use pointer "h" to enter the
ECRRHashLink table (at 0006 in Fig. 8B). Reset
pointer "h" from the content of the ECRRHashLink
table so that pointer "h" points to the next most
recent two-character string in the history buffer
(3750 in Fig. 8B). In each search, use steps 4a
through 4c above (or steps 5a through step 5c
below). Continue until the end of the linked
list, as indicated by a non-match of the hash with
the corresponding entry in the ECRR Hash Test
field or, to prevent looping, until
MaximumBSearches (sixteen in the preferred
embodiment) have been performed.

Store length and location of longest match if n
(length of longest match) > 2 .

5. Set p to CCBuffer pointer (the current character)
and repeat steps 2 through 4, using the following
steps a, b and c instead of steps 4a, 4b and 4c to
find the longest match:

a) Compare the character at $(p + 2)$ in the CC
buffer with the character at $(h + 2)$ in the
history buffer and terminate if no match.
This is the "fast reject step".

b) Set $n = 0$

c) Compare (one character at a time) the
character at $(p + n)$ in the CC buffer with
the character at $(h + n)$ in the history
buffer, incrementing n by 1 until no match.

6. Select from the outputs of step 4 and step 5 the

- 34 -

string which:

- a) is the longest;
- b) if the strings are equal, the one that is most recently stored.

5 String Length Encoding

String lengths are encoded differently for mode A string encoding and mode B string encoding.

In mode A, string lengths are encoded using the GlobalHigh/Low table. Further, the encoding is slightly
10 different depending upon the method of string escape. i) If the escape follows creation of a font, MinimumAString (which is 6) is subtracted from the actual length of the string and the result is used to index the GlobalHigh/Low table. ii) If the escape follows an old (existing) font,
15 MinimumAString (which is 6) is subtracted from the actual length of the string, four is added, and the result is used to index the GlobalHigh/Low table. This latter operation is because the bit pattern 11, which begins the first four entries in the GlobalHigh/Low table is reserved to signify a
20 Pair Encoding. The selected Huffman pattern from the GlobalHigh/Low table is placed into the output stream. String length encoding, mode A, old font, is illustrated as the second operation in Fig. 7C.

In mode B, string lengths are encoded by subtracting
25 MinimumBString (which is 3) from the actual length of the string. If the result is less than 9, the LengthBCode table is used to encode the string length. If the result is greater than or equal to nine, the further escape 0010 is output, an additional nine is subtracted from the result
30 above, and the new result is used to index the GlobalHigh/Low table. The selected Huffman pattern from the GlobalHigh/Low or LengthBCode table is placed into the output stream.

String Pointer Encoding

35 String pointer encoding for both mode A and mode B proceeds as follows:

The history buffer location of the first string

- 35 -

character is subtracted from the Next Buffer Store location. Buffer wraparound, if any, is corrected such that the result is the displacement from the found string to the Next Buffer Store location and is in the range 0 through BufferSize-1.

- 5 Note that strings closest in recent history (newer) have lesser displacements than do older strings.

Example 1. (using 8192 character buffer)

		Decimal	Hexadecimal
	Next Store location	3152	0C50
10	Found string location	1511-	05E7-
		-----	-----
		1641	0669

Example 2. (using 8192 character buffer)

	Next Store location	0052	0034
15	Found string location	8157-	1FDD-
		-----	-----
		8105-	1FA9-
	Correction	8192+	2000+
		-----	-----
20	String Displacement	87	57

With the BufferSize in the preferred embodiment selected as 8192, the calculated displacement can be expressed in thirteen bits.

The displacement is further broken into two components.

- 25 A) A zone portion from the most significant five bits. B) An offset portion from the least significant eight bits. In the proper string encoding context (i.e. after appropriate string encoding escapes) the five bit zone is Huffman coded using the ZoneCode table and the eight bit offset is
- 30 inserted directly into the output stream. Thus the Zone may be encoded using from 2 to 7 bits depending upon zone value with the strings closest in recent history getting favorably shorter encodings.

- 36 -

Example.	Hexadecimal	Binary
String Disp	0057	0000 0000 0101 0111
		0000 0000 = Offset = 57
5		
		Z ZZZZ = Zone = 0

String offset encoding is also illustrated as the third and fourth operations in Fig. 7C.

Fig. 7C illustrates string encoding, mode A, and shows
 10 a Character Stream 751 with a character string beginning
 with "w" 752, Font (No) 742 and Global Font Encoding Table
 743 yielding, for a font size value SZ =4, at entry point 3
 (3 = String Escape = NC + 1) 744, a Font String code 001
 724. Fig. 7C shows a History Buffer 753 having a character
 15 string beginning with "w" at location 933 (hexadecimal) 754.
 Fig. 7C shows that a string of nine characters 752 in the
 character stream match the nine characters in the history
 buffer beginning at location 933 754. The "Global Code" or
 Global Frequency Encoding Table 745 is entered at entry
 20 point 7 ($9 - 6 + 4 = 7$) 755 to create a Length Code of 1011
 756. The string location 933 (hexadecimal) 754 is
 subtracted from the location of the Next History Buffer
 Location 1201 (hexadecimal) 757 to yield 8CE (hexadecimal)
 whose 13 least significant bits 758 comprise the
 25 displacement which is broken into two components: i) a zone
 portion from the most significant five bits 759 and ii) an
 offset portion from the least significant eight bits 760.
 The five bit zone portion is Huffman coded using the Zone
 Code table 761 and the eight bit offset is inserted directly
 30 in the output stream. The Output Bit Stream Sequence 752
 includes 1st: Font (001), 2nd: Length (1011), 3rd: Zone
 (01001) and 4th: Offset (11001110).

Minimum String Length and Search Advance

In both mode A and mode B string encoding, the string
 35 search process discards matches having less than a
 predetermined number of characters, the predetermined number
 being greater than the hash length. Thus, we define a

- 37 -

minimum string length. The minimum string length can be greater than the hash length and it is advantageous to make it so. In mode A the hash length is 4 and the predetermined number is 6. In mode B the hash length is 2 and the
5 predetermined number is 3. Setting a lower limit on the length of the string reduces the bit-cost of encoding longer strings because the top (shortest code) entry into the Huffman table is used to represent a string of the minimum length.

10 On completion of a string search, if no match is found, a predetermined number of characters (3 if mode A and 1 if mode B) are released (in font encoded or pair encoded form) and the search pointer is advanced by a corresponding number of positions before the next search.

15 Pair Encoding

Pair Encoding is a novel method for encoding character pairs. Up to 1024 fonts, those associated with recently encountered character pairs, are maintained in memory principally for the purpose of font encoding. Pair encoding
20 takes advantage of the unambiguous one-to-one mapping between the input character pairs and the fonts effected using the CRC hash and the MatchVal. Since there are 1024 fonts maximum, ten bits ($2^{10} = 1024$) may be used to encode any of the character pairs that these fonts represent.
25 Thus, other than escape bit sequences, ten bits is all that is required to encode many character pairs. Assuming an average escape sequence of three bits, the resulting thirteen bit encoding compares quite favorably with the sixteen bits for two uncompressed characters especially in
30 computer binary codes files (eg .COM and .EXE).

In addition to the fonts and access structure maintained by the encoder, the decoder maintains a table of the actual two characters which are associated with each font. Thus it can do a direct lookup when directed by the
35 encoded bit stream.

Example. Refer to Fig. 7B which shows an input character stream 731 with a character pair "w^" 732, Font

- 38 -

(No) 722 (font address hex 195) and a Global Font Encoding Table 723 yielding, at entry point 3 725, a Font String Code 001 724. Assume that the character pair "w^" (lower case w and caret) has occurred previously in the input character stream and has font number 215 (hexadecimal) assigned to it by the font encoding process. The sequence "w^" 732 has occurred again following "No" in the input stream and is next to be processed for output by the encoder. After determining that the pair "w^" exists, and that Pair Encoding is the least cost, the encoder, entering the Global Font Encoding Table 723 of Font Size 6 (Font Size = SZ + 2 for Mode A) at NC+1 (String Escape 724), emits the String Escape "001" from font "No" 735, followed by the Pair Encoding Escape "11", 736 and the ten bit value "1000010101" (from binary of hex 215, the font number of "w^") 737 creating output bit stream 738.

Font Escapes

A font escape is a bit encoded sequence which serves as a signal from the encoder that the subsequent item is to be treated differently from that normally expected. A font encoded sequence that signifies that a NewChar follows in the data stream is an escape. It is used as an Escape to signal GlobalNC encoding. Another escape is String Escape. This is a bit sequence specifically to condition the decoder for reception of a string. When used in the context of a Font encoding/decoding, String Escape has a value equal to NewChar Escape + 1 when String "A" mode is active.

In string mode B the font has 3 escapes:

- 1) New character. Value = Font NC.
- 2) Pair encoding. Value = Font NC + 1.
- 3) String mode B. Value = Font NC + 2

Other escapes are described under Detail of Specific Encodings hereinbelow.

Second Phase Processing

Second Phase Processing, steps 11 through 20, includes string search, least cost encoding selection, formatting and output. Throughout these steps, the pointer into the

- 39 -

process buffer is the current output pointer which is from 1 to 256 characters behind (older than) the current input pointer.

- 5 11. According to the state of the mode switch, the correct string search routine is invoked, String Search Mode A or String Search Mode B.
12. If a less than a minimum length string (3 if mode B and 6 if Mode A) is found in step 11, proceed to step 15. Otherwise, the bit cost of the string is
10 computed by summing the costs of String Escape, String Length, Zone Code and the String Offset of 8, as follows:
 - 15 a. Fetch the ECNewIndex value corresponding to the first character of the string and add 1 if Mode A or 2 if Mode B. Use the result to access the FontBits section of the Global Font Encoding Table of Fig. 4A. The retrieved value from FontBits is the bit cost for the String Escape.
 - 20 b. If Mode A is active, subtract 6 and add 4 to the string length and use this result to access the GlobalBits table. If Mode B is active, subtract 3 from the string length and use this result to access the LengthBBits
25 table. This is the bit cost for the string length.
 - 30 c. Subtract the position of the first character of the string from the next history buffer store location and divide the result by 256 giving the zone. Using the computed zone, access the ZoneBits table. This is the bit cost for the Zone encoding.
 - d. The bit cost of the String Offset is 8.
 - e. Add items a through d. This sum is the total
35 bit cost of the string.
13. Compute the bit cost of equivalent font encoding for each position corresponding to a character in

- 40 -

- the string using step a or b below. Subtract this bit cost from the total from step 12. If underflow (the result goes negative) at any point, exit step 13 since the string encoding wins over the font encoding. If all corresponding positions are examined without underflow occurring, font encoding has a lesser or equal bit cost and will be used so proceed to step 15.
- 5
- a. If the ECType field is 4, use the ECNewindex field to access the FontBits table for the bit cost of NewChar Escape. Use the ECFrequency field to access the GlobalBits table for the bit cost of the NewChar.
- 10
- b. If the ECType field is 2, use the ECFontIndex field to access the FontBits table for the bit cost of a font encoding.
- 15
14. If string encoding wins as indicated in step 13, change the ECType field corresponding to the first character of the string to an 8 (denoting String Encoding) and then change the ECType field corresponding to all remaining characters of the string to a 0 (denoting string continuation). Set UpdateLength to string length. Proceed to step 19.
- 20
15. Examine the ECHashX2 field corresponding to the character at the current output position + 2. If NULL (the font exists in the encoder but does not yet exist in the decoder) proceed to step 18, otherwise compute the cost of a Pair Encoding as follows:
- 25
- 30
- a. Fetch the ECNewIndex value corresponding to the current output position and add 1. The result is used to index the FontBits table. This is the bit cost for the Pair Encoding Escape.
- 35
- b. Add 10 to the result of step a. This is the total Pair Encoding cost.

- 41 -

16. Compute the bit cost of equivalent font encoding for each of the two characters in the pair (current output position and current output position +1) using step a or step b below.
- 5 Subtract this bit cost from the total in step 15. If underflow (the result goes negative) at any point, exit step 16 since the pair encoding wins over the font encoding. If the two positions are examined without underflow occurring, font
- 10 encoding has a lesser or equal bit cost and will be used so proceed to step 18.
- a. If the ECType field is 4, use the ECNewindex field to access the FontBits table for the bit cost of NewChar Escape. Use the
- 15 ECFrequency field to access the GlobalBits table for the bit cost of the NewChar.
- b. If the ECType field is 2, use the ECFontIndex field to access the FontBits table for the bit cost of a font encoding.
- 20 17. If pair encoding wins as indicated in step 16, change the ECType field corresponding to the first character of the string to a 6 (denoting Pair Encoding) and then change the ECType field corresponding to the next character of the pair to
- 25 a 0 (denoting string/pair continuation). Set UpdateLength to 2. Proceed to step 19.
18. Set UpdateLength to 1. This is to be a font or NewChar encoding.
19. Access the ECType field at current output
- 30 position, format and output the bit sequences illustrated in Figs. 10A through 10D. Access the ECRepeats field at current output position, if greater than 0, output the repeat count using the GlobalCode (High and Low) table. Add each output
- 35 character to the history buffer and associated access tables. Increment current output position, decrement UpdateLength. Repeat step 19 while

- 42 -

UpdateLength is greater than 0.

20. If a Flush or Mode change operation is in process, repeat steps 11 through 19 until the process buffer is empty (current output position equals current input position). Otherwise proceed to step 1.

Compressibility and Encoding Process Switching

The following process is found to provide a useful measure of data compressibility. Every forty-eight new characters (i.e. characters not found in the font associated with the previous two characters), the cumulative bit-cost of encoding the previous ninety-six such characters is compared with a preset value. It is of no consequence to this process that such character might later be encoded as part of a string.

Every forty-eight NewChars (which may be more than forty-eight input characters), the current sum in NCBitsNew is added to the previous forty-eight character sum from NCBitsPrior and the result compared to the constant $96 * 7.5$ (representing 96 characters at 7.5 bits per character). If there are less than $96 * 7.5$ bits in the result, the Compressibility Trending Switch is turned OFF (or remains OFF). If the result is $96 * 7.5$ or greater, the Compressibility Trending Switch is turned ON (or remains ON). After the calculation, the current NCBitsNew is stored in NCBitsPrior in preparation for the next cycle forty-eight NewChars later.

If the compressibility trending switch is on, the following are in effect:

1. Font encoding is active.
2. String mode B is active.
3. Pair encoding is active.
4. Anti-expansion mode is active.

If the compressibility trending switch is off, the following are in effect:

1. Font encoding is active.
2. String mode A is active.

- 43 -

3. Pair encoding is active.

4. Anti-expansion mode is inactive.

Detail of Specific Encodings

The several encodings produced by the present invention, in addition to font encoding (NewChar, Pair and String) are shown in Tables 1A through 1D below. Table 2 provides the key to the data in these tables.

<u>Preconditions</u>	<u>Font Position</u>
10 Old Font, Mode A	EEE
Old Font, Mode B	FFF

Table 1A - Font Encodings

<u>Preconditions</u>	<u>Escapes</u>	<u>Frequency</u>
15 Old Font, Mode A	NES	NNN
New Font, Mode A	00 e	PPP
New Font, Mode A	10 or 11	PPP
Old Font, Mode B	NES	ffff ffff
New Font, Mode B	10	hhh hhhh
20 New Font, Mode B	0	iii iiii
Mode B, Antiexpansion		ffff ffff

Table 1B - New Character Encodings

<u>Preconditions</u>	<u>Escapes</u>	<u>10 Bit Font Number</u>
25 Old Font, Mode A	SEA 11	bb bbbb bbbb
New Font, Mode A	01 0	bb bbbb bbbb
Old Font, Mode B	PEB	bb bbbb bbbb
New Font, Mode B	11 0	bb bbbb bbbb

Table 1C - Pair Encodings

- 44 -

	<u>Preconditions</u>	<u>Escapes</u>	<u>Length</u>	<u>Buffer Position</u>
	Old Font, Mode A	SEA	SSS	ZZZ 0000 0000
	New Font, Mode A	01 1	GGG	ZZZ 0000 0000
	Old Font, Mode B	SEB	LLL	ZZZ 0000 0000
5	New Font, Mode B	11 1	LLL	ZZZ 0000 0000
	Old Font, Mode B	SEB 0010	GGG	ZZZ 0000 0000
	New Font, Mode B	11 1 0010	GGG	ZZZ 0000 0000

Table 1D - String Encodings

10	<u>Key to Tables 1A through 1D</u>	
	bb bbbb bbbb	A ten bit number representing the font number with which the encoded pair is associated.
	e	A single bit emitted to comprise the second of two bits which serve as a prefix to the PPP encoding.
15	ffff ffff	Eight bits representing a character frequency in the range 0 - 255.
	hhh hhhh	Seven bits representing a character frequency in the range 128 - 255.
20	iii iiii	Seven bits representing a character frequency in the range 0 - 127.
	oooo oooo	The eight least significant bits of the buffer (relative to the Next Buffer Store Location) displacement of the first character of a string. Used with a ZZZ encoding to identify a string position.
25	EEE	A Huffman pattern from the FontCode table from one to four bits in length encoding a value not equal to the Font NewChar or Font NewChar plus one and representing the font relative position of the encoded character in the Font.
30	FFF	A Huffman pattern from the FontCode table from one to five bits in length encoding a value not equal to the Font NewChar, Font NewChar plus one, or Font NewChar plus two and representing the font relative position
35		

- 45 -

of the encoded character in the Font.

GGG A Huffman pattern from the GlobalHigh/
GlobalLow table from four to thirteen bits in
length, in mode A, encoding a value from 0 to
5 249 and representing a string length of 6 -
255 characters; in mode B, encoding a value
from 0 to 243 and representing a string
length of 12 - 255 characters.

LLL A Huffman pattern from the LengthBBits table,
10 from one to six bits in length, encoding a
value from 0 to 8 and representing a string
length from three to eleven characters.

NES A Huffman pattern from the FontCode table
from one to four bits in length encoding a
15 value equal to the Font NewChar and
representing a NewChar Escape.

NNN A Huffman pattern from the GlobalHigh/
GlobalLow table from four to thirteen bits in
length, encoding a value from 0 to 255 and
20 representing a character frequency.

PEB A Huffman pattern from the FontCode table
from two to four bits in length encoding a
value equal to the Font NewChar plus one and
representing a Pair Encoding Escape, Mode B.

25 PPP The remainder of a Huffman pattern from the
GlobalHigh/GlobalLow table, excepting the
first two bits which are emitted separately,
from two to eleven bits in length, encoding a
30 value (in consideration of the prior two
bits) from 0 to 255 and representing a
character frequency.

SEA A Huffman pattern from the FontCode table
from two to four bits in length encoding a
value equal to the Font NewChar plus one and
35 representing a String Escape, Mode A.

SEB A Huffman pattern from the FontCode table
from three to five bits in length encoding a

- 46 -

		value equal to the Font NewChar plus two and representing a String Escape, Mode B.
	SSS	A Huffman pattern from the GlobalHigh/GlobalLow table, excepting the first four
5		entries (those beginning with 11), from four to thirteen bits in length, encoding a value from 4 to 255 and representing a string length of 6 - 253 characters.
	ZZZ	A Huffman pattern from the ZoneCode table, from four to thirteen bits in length, encoding a value from 0 to 31 and representing the five most significant bits of the string displacement. Used with the 0000 0000, described above to identify a
10		
15		string position in the history buffer.

Table 2 - Key to Encodings of Tables 1A through 1D

Selecting and Assembling Encodings

The least-cost encoding is built by selecting the
 20 encoding that has the fewest bits. If the bit cost of the two encodings are the same, font encoding is chosen.

If string encoding is selected, there may be up to three prefix characters not included in the string (e.g., the current character to the character immediately prior to
 25 the beginning of the string). Any such prefix characters are font encoded or pair encoded and their code is transmitted ahead of the string encoding.

Anti-expansion

Whereas it is possible for certain data streams to
 30 exhibit very little patterning, data expansion is a possible outcome of font encoding and string encoding systems. To counter this possibility, a running computation of the output bit count for Mode B minus 8 (bits per character) is maintained, i.e., for each equivalent character output,
 35 $SUM = SUM + BitCost - 8$. Thus a positive result indicates poor compression and a negative result indicates good compression. A switch is maintained which controls the

- 47 -

output stream such that, when the switch is on, the eight-bit frequency is output instead of the normal font, string, or pair encoding for mode B. A command (frequency 0FEh followed by a single 1 bit) is used to signal the decoder to
5 change state.

Table 3 indicates the action taken for each character output.

Switch On (Transparent Mode)		Switch Off (Mode B Encoding)	
10	SUM >= 0 No Change	SUM < 0 No Change	
	SUM -1 to -19 No Change	SUM 0 to 19 No Change	
	SUM < -19 Set Switch Off	SUM >19 Set Switch On	
15	Table 3 - Anti-expansion Actions		

Decoder Process

Figs. 10A and 10B provide a flowchart of the decoding process. Table 4 provides the key to the flowchart of Figs.
20 10A and 10B.

	F<n>	Fetch the next <n> bits from the input stream (where n is an integer).
25	D G	Decode Global. Decode a Huffman pattern which was selected and encoded from the GlobalCode encoding table.
	D F	Decode Font. Decode a Huffman pattern which was selected and encoded from the appropriate Font Encoding tables (Fig. 4A).
30	D S	Decode Short. Decode a Huffman pattern which was selected and encoded from the GlobalCode encoding table. Same as the DG (Decode Global) except that two bits have already been fetched (F2) and are in DCCode. Used
35		for length of 'A' type strings.
	D L	Decode Length. Decode a Huffman pattern which was selected and encoded from the

- 48 -

LengthBCode encoding table. Used for length of 'B' type strings.

D Z Decode Zone. Decode a Huffman pattern which was selected and encoded from the lowest 64 entries in the GlobalCode encoding table. Used for length of 'B' type strings.

Table 4 - Key to Decoder Flow

In the decoder flow, there are only four possible endpoints to a single decoder (and implicitly encoder) cycle. These four different endpoints are shown in Figs. 10A and 10B by an integer inside a triangle. They correspond to the four methods of encoding, shown in Tables 1A through 1C hereinabove: Font, NewChar, Pair and String.

Dual Processor Configuration

As discussed under Background Art hereinabove, the combination of higher data rates in data transmission systems, the achievement of high data compression efficiencies and the use of complex process-intensive algorithms for data compression increases the processing throughput required to perform modem control and data compression/decompression tasks. In a preferred embodiment, referring to Fig. 11, the system uses two processors connected in series between the computer (the DTE interface, 111) and the telephone line (the DCE interface, 112), each processor having its own memory. One processor, the DCE Interface Processor 113, a Zilog Z80180, performs DCE interface processes (modem control and data flow management). The other processor, the Compression/Decompression and DTE Interface Processor 114, a Rockwell C19, performs data compression, data decompression and DTE interface processes (data interchange with the PC). This configuration is shown for duplex operation in Fig. 11. Fig. 11 shows a data rate of 11,500 characters/second at the DTE Interface and a data rate of 1,500 characters/second at the DCE Interface. The conventional (prior art) approach using a single processor 121 to perform all functions (DTE

- 49 -

interface, DCE interface and Compression/Decompression) is shown in Fig. 12. The single processor approach involves using a more powerful, albeit more expensive, processor.

The general problem of sharing tasks among multiple processors is known to be a difficult problem in computer science. A conventional solution that might be applied to data compression modem applications is shown in Fig. 13. Fig. 13 shows a conventional two-processor configuration having a DTE/DCE Interface Processor 131 and a Compression/Decompression Processor 132. The present invention achieves the sharing of tasks by a simple but, nonetheless, unexpectedly effective configuration.

The preferred embodiment, shown in Fig. 11, achieves efficient control over all processes occurring in the system. This configuration utilizes the insight that compression and decompression and interface with the terminal all occur at a high error-free data rate, whereas modem control and the data line interface processes operate at a lower data rate and involves error detection and repeat transmission to cope with transmission errors. Accordingly, a first relatively high speed processor is used for both control of the terminal interface and for data compression and decompression; and a second processor is used for the processes involved in control of the data line interface including error detection and retransmission. Thus loading peaks occurring in either processor cannot interfere with the other.

Glossary

Encoder Current Character

30 The most recent character from the input stream which is being processed by the encoder font system. At the end of each encoder cycle, encoder current character, "ECChar", becomes Char1Prior and the fetch and encode process
35 continues with the next character from the input stream as encoder current

- 50 -

	character.
Escape	A bit encoded sequence which serves as a
	signal from the encoder that the
	subsequent item is to be treated
5	differently from that normally expected.
	Example: a font encoded sequence that
	signifies that a NewChar follows in the
	data stream.
Font	One record of an array of records, each
10	record consisting of pointers, links,
	characters, etc., each record having an
	address based on the prior two
	characters in the input stream, each
15	record containing a list of historically
	occurring candidate characters to be
	matched with characters from the input
	stream.
Huffman Codes	As used in this document, this term
	refers to any variable length bit
20	representation having fewer bits
	corresponding to higher frequency of
	occurrence, including but not limited to
	codes created by a tree algorithm.
NewChar	The occurrence of "NewChar" is a font
25	encoding event wherein either the
	encoder current character, "ECChar", is
	not found in the selected font or there
	is no font in existence (and it thus
30	contains 0 characters based on the font
	selection scheme).
NewChar Symbol	A dynamic value in the range of 0
	through n (where n is the maximum number
	of characters per font) which represents
35	the current virtual position in the font
	which represents "character not in
	table". It is used as an escape to
	signal GlobalNC encoding.

- 51 -

NewChar Escape	Specifically an encoding representing the NewChar Symbol.
String Escape	An escape sequence specifically to condition the decoder for reception of a string. When used in the context of a font encoding or a font decoding, a value equal to NewChar Escape + 1.
5	

- 52 -

APPENDIX 1

SOURCE CODE

```

; StringA, StringB with full separation in StringTime calls
;
; From TC90F2K2.MAC
5 ;

        IF2
        .printx /C19 Encoder and Decoder/
        ENDIF
        .xlist
10      .C18          ; assembler, please do C18
        instructions
        lodr6        equ      1      ; 6.144mhz clock
        .sfcond
        include      ITEc19
15      include      TCdfm001
;
        printstat    macro      a,b,c,d
        if2
        .printx /a b c d/
20      endif
        endm
        .list
        pagealign    macro
        if          (tblofs and 255) ne 0
25      fred          defl (0-tblofs) and 255
        printstat    <Page Align Waste =>,%fred
        tblofs      defl tblofs + fred
        endif
        endm
30 ;
;***** A S S E M B L Y   O P T I O N S *****
;
;   OPTIONS WHICH CHANGE COMPRESSION/SPEED
;
35 AHashX2            EQU 0      ; / (0) 0 - no; 1 - yes
MaximumASearches     EQU 8      ; / (8) maximum A hashes

```

- 53 -

APPENDIX 1

```

searched
MaximumBSearches    EQU 16    ; / (16) maximum B hashes
searched
NC8BitCycle         EQU 64    ; / (64) controls
5 A-String/B-String
TwoBytes            EQU 1      ; / (1) 0 - one-byte font controls
                        ;      1 - two-byte font controls
ZoneTestA           EQU 1      ; / (1) 0 - HIGH; 1 - HIGH &
LOW
10 ZoneTestB         EQU 1      ; / (1) 0 - HIGH; 1 - HIGH &
LOW
;
; * * * * *
;
15 ;   OPTIONS WHICH PROBABLY ARE NOT GOING TO CHANGE
;
FontSize            EQU 8      ; only 8,16 are supported; this
                        ; keeps fonts on page boundaries
FontTables          EQU 1024   ; may be 512-1024 provided
20 that
                        ; (FontTables*FontSize) MOD 256 =
0
                        IF   FontSize EQ 16
CharsPerFont        EQU 13     ; otherwise need 17,18-index
25 tables
                        ELSE
CharsPerFont        EQU FontSize-TwoBytes-1
                        ENDIF
SetLength           EQU 128    ; refill to SetLength*2 bytes
30 after
                        ; SetLength(+) bytes have been
encoded
AntiEx              EQU 1      ; 0 - off; 1 - on
BufferSize          EQU 8192   ; size of Round Robin buffer
35 BufferHashes      EQU 2048   ; # of Round Robin 4-byte
hashes

```

- 54 -

APPENDIX 1

```

BufferSuffix      EQU 1      ; 0 - nulls; 1 - maintained
FailSafe          EQU 0      ; 0 - no failsafe; 1 - output
failsafe

                    IF FailSafe
5 FailSafeSets     EQU 4      ; output every (n * 256)
encodings

                    ENDIF

FTHashes          EQU 2048 ; # of Round Robin 2-byte hashes
MinimumAString    EQU 6      ; minimum length A string
10 MinimumAUpdate  EQU 3      ; bytes advanced if no A
string found
MinimumBString    EQU 2      ; minimum length B string
MinimumBUpdate    EQU 1      ; bytes advanced if no B
string found
15 NCFreqSets      EQU 4      ; uses 256*2*2*Sets bytes
NCFreqSetsHigh    EQU 0      ; if used, 0 gives best
result ?????
NCFreqSetsReset   EQU 1      ; 0 - no; 1 - reset on B
to A change
20 Repeats         EQU 1      ; 0 - no repeat logic; 1 -
repeat logic
;

                    IF FontTables GT 512
MatchMask         EQU 0FC00H
25 NextMask        EQU 7FEH    ; after ASL A
ELSE
MatchMask         EQU 0FE00H
NextMask          EQU 3FEH    ; after ASL A
ENDIF
30 ;
; * * * * *
;
;   DEBUG AND TEST OPTIONS
;
35 Debug           EQU 1      ; set to 0 to skip statistics
DbgDum            EQU Debug XOR 1

```


- 55 -

APPENDIX 1

```
EOFControl      EQU 1      ; 0 - endless data flow;1 -
file by file
Macros          EQU 1      ; 0 - use subroutines; 1 - use
macros
5 Prodder       EQU 0      ; 0 - no prods; 1 - force
prods
                IF Prodder
ProdCycle       EQU 67     ; prod every ProdCycle
characters
10             ENDIF
Test           EQU 0      ; 0 - no test code; 1 - test
code
;
;***** OS EQUATES and ASSEMBLY OPTIONS *****
15 ;
Load8250 EQU      0      ; serial loader/debugger
;Load8250 EQU      1      ; parallel loader/debugger
DecBankSelect   MACRO
                SMB      2,PortB
20             ENDM
EncBankSelect   MACRO
                RMB      2,PortB
                ENDM
;
25 ;***** H O S T   I N T E R F A C E   M A P *****
;
;   HOST INTERFACE MAP definition (16450 mode)
;
                w8250_RXD equ 00020h
30             w8250_TXD equ 00021h
                w8250_LCR equ 00023h
                w8250_MCR equ 00024h
                ln_stat   equ 00030h
                mdm_stat  equ 00031h
35             HostContrl equ 00032h
;
;
```

- 56 -

APPENDIX 1

```

;***** FONT TABLE STRUCTURE *****
;
;   ENCODER / DECODER STRUCTURE MAPS
;
5 ; Map of 1 FONT entry
;
    tblbgn
    IF   TwoBytes
        tbyte   NCIndex
10        tbyte   Characters
    ELSE                                ; Bits 7-4 = Characters
        tbyte   CharsNCIndex          ; Bits 3-0 = NCIndex
    ENDIF
    tstor      CharTable,CharsPerFont
15    tblend    TestFontSize           ; size of a font
table
;
    if         TestFontSize NE FontSize
        db     256,Font size not 16
20    else
        printstat    <Font Size =>,%FontSize
        printstat    <Chars per font =>,%CharsPerFont
    endif
;
25 ;***** PAGE 1 VARIABLES *****
;
;   ENCODER / DECODER RAM PAGE 1 VARIABLES
;   (48H through 07fH inclusive)
;
30 ; Miscellaneous Variables
;
    tblbgn    RamPtr1
    IF   EOFControl          ; !!!!! must be in 48h
        tbyte   HostLCR      ; BBS,BBR
35    ENDIF
    tbyte     FetchPtr

```

- 57 -

APPENDIX 1

```

        tbyte    StorePtr
        tbyte    DCByte1
        tbyte    DCByte2
        tbyte    DCByte3
5       tword    DCWord1
        tword    DCWord2
        tword    DCWord3
        tbyte    ECByte1
        tbyte    ECByte2
10      tbyte    ECByte3
        tbyte    ECByte4
        tword    ECWord1
        tword    ECWord2
        tword    ECWord3
15      tword    ECWord4
        ;

        IF      EOFControl
            tstor    BytesIn,3
            tstor    BytesOut,3
20      tbyte    DCStack
            tbyte    ECStack
            tbyte    OutFetch
            tbyte    OutStore
        ENDIF
25      ;

        IF      Prodder
            tbyte    ProdCounter
        ENDIF
        ;

30      if      tblofs gt 80h
            db      256,Ram Window Error
        else
            MemoryOne    equ tblofs - RamPtr1
            printstat    <Page 1 Window Free =>,%080h-tblofs
35      endif
        ;

```

- 58 -

APPENDIX 1

```
;***** PAGE 0 VARIABLES *****
;
;   ENCODER / DECODER RAM PAGE 0 VARIABLES
;   (83h through 0ffh inclusive)
5 ;
      tblbgn    RamPtr0
;
; Decoder Variables
;
10      tbyte    DCABStatus
      tbyte    DCBuffer
      tbyte    DCCharacters
      tbyte    DCCharCount
      tbyte    DCChar1Prior
15      tbyte    DCChar2Prior
      tbyte    DCCommand
      tbyte    DCCurrentChar
      tbyte    DCCurrentFreq
      tword    DCCurrentHash
20      IF      FailSafe
      tword    DCFailSafe
      ENDIF
      tword    DCFontBase
      tbyte    DCFontIndex
25      tword    DCFTLastHash
      tword    DCFTNextRough
      tword    DCFTPParent
      tword    DCFTChild
      tword    DCNCBitsNew
30      tword    DCNCBitsPrior
      tbyte    DCNCCounter
      tbyte    DCNCIndex
      tword    DCRRPtr
;
35 ; Encoder Variables
;
```

- 59 -

APPENDIX 1

```

    tbyte    ECABStatus
    tbyte    ECBuffer
    tbyte    ECCharacters
    tbyte    ECChar2Prior
5   tbyte    ECChar1Prior
    IF      Repeats
        tbyte    ECCharSave
    ENDIF
    tbyte    ECCCommand
10   tbyte    ECCurrentChar
    tbyte    ECCurrentFreq
    tword    ECCurrentHash
    IF      FailSafe
        tword    ECFailSafe
15   ENDIF
    tbyte    ECFlush
    tword    ECFontBase
    tword    ECFTChild
    tword    ECFTLastHash
20   tword    ECFTNextRough
    tword    ECFTPParent
    tword    ECNCBitsNew
    tword    ECNCBitsPrior
    tbyte    ECNCCounter
25   tbyte    ECNCIndex
    IF      Repeats
        tbyte    ECRRepeatCount
    ENDIF
    tbyte    ECPriorHash0
30   tbyte    ECPriorHash1
    tword    ECRRPtr
;

    tbyte    ECABChange
    IF      AntiEx
35   tbyte    ECAntiEStatus
    ENDIF

```

- 60 -

APPENDIX 1

```

        tbyte    ECAvailable
        tbyte    ECExcessBits
        tbyte    ECFindHash
        tword     ECFound
5         tbyte    ECMaxLength
        tbyte    ECNextChar
        tbyte    ECNextOut
        tbyte    ECNextOutSave
        tbyte    ECNextOutStart
10        tbyte    ECStringLength
        tbyte    ECStringOrigin
        tbyte    ECZone
;
        if      tblofs gt 100h
15         db      256,Page 0 Ram Error
        else
            MemoryZero    equ tblofs - RamPtr0
            printstat      <Page 0 Ram    Free
=>,%0100h-tblofs
20         endif
;
;***** 0 8 0 0 - 4 0 0 0 h   M E M O R Y   B L O C K   *****
;
;    ENCODER / DECODER TABLES
25 ;
        tblbgn    0800h
;
        tstor     ECChar,256
256
30        tstor     ECCharCopy,256
256
        IF    Repeats
            tstor     ECrepeats,256
256
35        tstor     ECrepeatSW,256
256

```

- 61 -

APPENDIX 1

```

        ENDIF
        tstor      ECType,256                                ;
256
        tstor      FTHashMatch,FontTables*2                ;
5 2048
        tstor      ECRRHashHead,BufferHashes*2
        ; 4096
        tstor      InBuffer,256                            ;
256
10      IF      EOFControl;{
        tstor      OutBuffer,256                            ;
256
        ENDIF      ;}
        tstor      DCGlobalHigh,4                          ;
15 4
        tstor      ECGlobalHigh,4                          ;
4
                                                ; 7944
;
20      IF      Test ;{
        tstor      FSEntries,3
        tstor      FSNoHash,3
        tstor      FSSearches,3
        tstor      FSSkips,3
25      tword     AStringsOn
        tword     BStringsOn
        tword     SwitchToA
        tword     SwitchToB
        tword     AHashX2s
30      tword     BHashX2s
        tword     AStringsFound
        tword     BStringsFound
        tword     AStringsUsed
        tword     BStringsUsed
35      tword     AntiExOff
        tword     AntiExOn

```


- 63 -

APPENDIX 1

```

; 8192
    tstor    ECRRSuffix,256          ;
256
    tstor    ECNCChar,256            ;
5 256
    tstor    ECNCFreq,256            ;
256
    tstor    ECNCCandF,(NCFreqSets-1)*512 ; 4
sets    ; 1536
10      tstor    ECFontIndex,256
; 256
    tstor    ECFrequency,256
; 256
    tstor    ECHashRaw0,256          ;
15 256
    tstor    ECHashRaw1,256          ;
256
    tstor    ECHashX20,256           ;
256
    tstor    ECHashX21,256           ;
20 256
    tstor    ECNewIndex,256          ;
256
;32768
25      IF      tblofs GT 0C000h
        DB      256,Addr 4000h-C000h Block Error
      ELSE
        fred     defl 0C000h-tblofs
        printstat <Encoder Main Ram Free =>,%fred
30      ENDIF
;
;**** 4 0 0 0 - C 0 0 0 h   D E C O D E R   B L O C K   ****
;
; Decoder Font Tables:
35 ;
    tblbgn    4000h

```

- 64 -

APPENDIX 1

```

;
; MAX
; 8192
; ECRRHashTest,BufferSize/2*2
; 8192
; DCFontTables,FontTables*FontSize
5 ; 8192
IF FontTables GT 512;{
; 2048
; DCFTHashRough,1024*2
; 2048
ELSE
; {}
10 ; 2048
; DCFTHashRough, 512*2
ENDIF
; {}
; DCFTHashNext,FontTables*2
; 2048
; DCFTHashChars,FontTables*2
15 ; 2048
; DCRRBuffer,BufferSize
; 8192
; DCNCChar,256
256
; DCNCFreq,256
20 256
; DCNCCandF, (NCFreqSets-1)*512
sets ; 1536
; 32768
25 IF tblofs GT 0C000h
DB 256,Addr 4000h-C000h Block Error
ELSE
fred defl 0C000h-tblofs
printstat <Decoder Main Ram Free =>,%fred
30 ENDIF
;
;***** B A S E O F P R O G R A M *****
;
; BASE OF PROGRAM
35 ;
cb equ $

```

- 65 -

APPENDIX 1

```

;
;***** E N C / D E C   I N T E R F A C E   S U B S   *****
;
          IF   EOFControl XOR 1;{  all code in this Section
5  is
;
          active only in modem operation
;
; * * * * *
;
10 ;   READ FROM PC VIA INTERRUPT
;
HostInt:
;
;
15          RTI
;
; * * * * *
;
;   ENCODER SENDS PROD/COMMAND TO REMOTE
20 ;
SendProdCommand:
;
;          STI   #0A0h,ECCCommand          ;   Prod is 10
;          STI   #0C8h,ECCCommand          ; Command is 11nn
25 ;
          JSR   ECProdCommand
;
;          etc.
;
30 ; * * * * *
;
;   DECODER PROCESSES COMMAND FROM REMOTE
;
ProcessCommand:
35 ;
;
;          ; process command and then

```

- 66 -

APPENDIX 1

```

;                                     ; return to DCFontParams
    STI #000h,DCCommand
    JMP DCFontParams
;
5  ;* * * * *
;
;    DO WHATEVER IS REQUIRED WHEN DECODER FAILS
;
FailSafeFailed:
10 ;
;
;
; * * * * *
;
15 ;    ENCODER READ FROM PC
;
ECReadCharacter:
;
;
20     RTS
;
; * * * * *
;
;    ENCODER WRITE TO PC
25 ;
ECWriteCharacter:
;
;
    RTS
30 ;
; * * * * *
;
;    DECODER READ FROM PC
;
35 DCReadCharacter:
;

```

- 67 -

APPENDIX 1

```

;
        RTS
;
; * * * * *
5 ;
;   DECODER WRITE TO PC
;
DCWriteCharacter:
;
10 ;
        RTS
;
; * * * * *
;
15         ENDIF                ;} end of modem enc/dec
routines
;
;***** I N I T I A L I Z E   M E M O R Y *****
;
20 ; Initialize Working Storage
;
Initialize:
        LDX  #MemoryZero
        LDA  #0
25 ClearMemory0:
        STA  RamPtr0-1,X
        DEX
        BNE  ClearMemory0
        LDX  #MemoryOne
30         DEX
ClearMemory1:
        STA  RamPtr1,X          ; leave HostLCR unreset
        DEX
        BNE  ClearMemory1
35 ;
;   CLEAR 800h-3FFFh

```

- 68 -

APPENDIX 1

```

;
    STI    #008h,DCWord1+1
    LDY    #038h
    JSR    BlockReset
5 ;
;    CLEAR 4000h-BFFFh - DECODER
;
    DecBankSelect
;
10    STI    #040h,DCWord1+1
    LDY    #080h
    JSR    BlockReset
;
    STI    #HIGH(DCNCChar),DCWord1+1
15    JSR    NCCharFreqReset
;
    IF    FailSafe
        STI    #000h,          DCFailSafe+0
        STI    #FailSafeSets,  DCFailSafe+1
20    ENDIF
;    STI    #001h,          DCABStatus
    STI    #HIGH(DCFontTables),  DCFontBase+1
    IF    FontSize EQ 8
        STI    #008h,          DCFontBase+0
25    ELSE
        STI    #010h,          DCFontBase+0
    ENDIF
    LDA    #HIGH(DCFTHashRough)
    STA          DCFTNextRough+1
30    STA          DCFTParent+1
    LDA    #HIGH(DCFTHashNext)
    STA          DCFTHashRough+1
    STA          DCFTLastHash+1
    LDA    #002h
35    STA          DCFTHashRough+0
    STA          DCCurrentHash+0

```

- 69 -

APPENDIX 1

```

        STA                DCFTLastHash+0
        LDA  #080h
        STA                DCFontIndex
        STA                DCCurrentHash+1
5      STA                DCBuffer
        STI  #HIGH(DCRRBuffer),  DCRRPtr+1
        STI  #HIGH(NC8BitCycle*8),  DCNCBitsPrior+1
        STI  #LOW (NC8BitCycle*8),  DCNCBitsPrior+0
        STI  #NC8BitCycle,          DCNCCounter
10 ;
    ;  CLEAR 4000h-BFFFh - ENCODER
    ;
        EncBankSelect
    ;
15      STI  #040h,DCWord1+1
        LDY  #080h
        JSR  BlockReset
    ;
        STI  #HIGH(ECNCChar),DCWord1+1
20      JSR  NCCharFreqReset
    ;
        IF  Prodder
            STI  #ProdCycle,          ProdCounter
        ENDIF
25      IF  FailSafe
            STI  #000h,          ECFailSafe+0
            STI  #FailSafeSets,    ECFailSafe+1
        ENDIF
    ;
        STI  #001h,          ECABStatus
30      IF  AntiEx
    ;
            STI  #001h,          ECAntiEStatus
        ENDIF
        LDA  #HIGH(ECFTHashRough)
        STA                ECFTNextRough+1
35      STA                ECFTParent+1
        STI  #HIGH(ECFTHashNext),    ECFTLastHash+1

```

- 70 -

APPENDIX 1

```

        STI    #002h,          ECFTLastHash+0
        LDA    #HIGH(ECFTHashNext)
        STA          ECFTHashRough+1
        LDA    #002h
5         STA          ECFTHashRough+0
        STA          ECCurrentHash+0
        STA          ECHashX20+255
        LDA    #080h
        STA          ECCurrentHash+1
10        STA          ECHashX21+255
        STI    #001h,          ECBuffer
        STI    #HIGH(ECRRBuffer), ECRRPtr+1
        STI    #HIGH(NC8BitCycle*8), ECNCBitsPrior+1
        STI    #LOW (NC8BitCycle*8), ECNCBitsPrior+0
15        STI    #NC8BitCycle, ECNCCounter
        ;
        RTS
        ;
BlockReset:
20        STI    #000h,DCWord1+0
        LDA    #000
        BRLoop:
        STA    (DCWord1)
        INC    DCWord1+0
25        BNE    BRLoop
        DEY
        BEQ    BRExit
        INC    DCWord1+1
        !JMP    BRLoop
30 BRExit:
        RTS
        ;
NCCharFreqReset:
        LDA    #NCFreqSets
35        STA    DCByte1
        STI    #000h,DCWord1+0

```


- 71 -

APPENDIX 1

```
NCCFRLoop0:
    LDX  #000h
NCCFRLoop1:
    LDA  Best128,X
5      STA  (DCWord1)
    INC  DCWord1+0
    INX
    BPL  NCCFRLoop1
NCCFRLoop2:
10     TXA
    STA  (DCWord1)
    INC  DCWord1+0
    INX
    BMI  NCCFRLoop2
15     LDA  DCWord1+1
    ADD  #001h
    STA  DCWord2+1
    STI  #000h,DCWord2+0
    LDY  #000h
20  NCCFRLoop3:
    LDA  (DCWord1)
    TAX
    TYA
    STA  (DCWord2),X
25     INC  DCWord1+0
    INY
    BNE  NCCFRLoop3
    DEC  DCByte1
    BEQ  NCCFRExit
30     INC  DCWord1+1
    INC  DCWord1+1
    !JMP NCCFRLoop0
NCCFRExit:
    RTS
35 ;
;***** M A I N   D A T A   F L O W *****
```

- 72 -

APPENDIX 1

```

;
StrtUp:
    LDX #0FFh                ; set stack pointer
    TXS
5    mask_gen <bcr_fast_es2,bcr_fast_es1>
    STI #mask,bcr            ; set C18/C19 to fast
    execution
        mask_gen <cir_fast_es3>
        STI #mask,clint
10    STI #007h,HostContrl    ; enable 16450 mode +
    interrupts
        STI #05Fh,ln_stat      ; 8250.THRE = 1
ResetMemory:
    BBS 4,HostLCR,ResetHostLCR
15    JSR Initialize
ResetHostLCR:
    STI #080h,HostLCR        ; set bit 7
LCRLoop:
    STI #000h,FetchPtr
20    STI #000h,StorePtr
    BBR 7,HostLCR,SetBreak    ; reset if host wrote LCR
    !JMP LCRLoop
SetBreak:
    LDA w8250_LCR            ; save host command info
25    STA HostLCR
    BBR 5,HostLCR,SetBreakCont
    BBS 6,HostLCR,SetBreakCont
    BBR 2,HostLCR,SetBreakCont
    STI #0F6h,HostContrl      ; no ints during Memory
30 Load
SetBreakCont:
    BBS 4,HostLCR,SetBreakNoReset
    JSR Initialize            ; HostLCR(4) - 0 reset
memory
35 SetBreakNoReset:
    BBS 6,ln_stat,SetBreakTSRE

```

- 73 -

APPENDIX 1

```

        STI    #02Fh,ln_stat      ; set 4, leave 6 at 0
        !JMP   WhichProcess

SetBreakTSRE:
        STI    #06Fh,ln_stat      ; set 4, leave 6 at 1
5  WhichProcess:
        BBS    6,HostLCR,LoopBack
        BBS    5,HostLCR,DumpLoadMemory
        BBR    2,HostLCR,ECStart   ; HostLCR(2) - 0 Encoder
DCStart:
        ;                - 1 Decoder
10      DecBankSelect
        JMP    DCFontParams
ECStart:
        EncBankSelect
        JMP    ECRefill
15  DCorECEOF:
        LDX    #0FFh              ; reset primary stack
        TXS
        BBS    6,ln_stat,EOfTSRE
        STI    #02Fh,ln_stat      ; set 4, leave 6 at 0
20      !JMP   EOFStats
EOFTSRE:
        STI    #06Fh,ln_stat      ; set 4, leave 6 at 1
EOFStats:
        BBS    3,HostLCR,EOFAcked ; set if host set LCR bit
25  3
        !JMP   EOFStats
EOFAcked:
        LDA    BytesIn+0
        JSR    SubWriteToPC
30      LDA    BytesIn+1
        JSR    SubWriteToPC
        LDA    BytesIn+2
        JSR    SubWriteToPC
        LDA    #000h
35      JSR    SubWriteToPC
        LDA    BytesOut+0

```

- 74 -

APPENDIX 1

```

        JSR  SubWriteToPC
        LDA  BytesOut+1
        JSR  SubWriteToPC
        LDA  BytesOut+2
5       JSR  SubWriteToPC
        LDA  #000h
        JSR  SubWriteToPC
        LDA  #000h
        STA  BytesIn+0
10      STA  BytesIn+1
        STA  BytesIn+2
        STA  BytesOut+0
        STA  BytesOut+1
        STA  BytesOut+2
15      JMP  ResetMemory
;
;***** P C   L O O P B A C K   C O D E   *****
;
        IF  EOFControl      ;{  all code in this Section
20  is
;                               active only in loopback
operation
;
; * * * * *
25 ;
    DumpLoadMemory:
        BBS  2,HostLCR,LoadMemory
    DumpMemory:
        JSR  MemoryDump
30      !JMP DCOrECEOF
    LoadMemory:
        JSR  MemoryLoad
        !JMP DCOrECEOF
;
35  LoopBack:
        BBR  5,HostLCR,LoopBackNoDump

```

- 75 -

APPENDIX 1

```

        JSR  MemoryDump
        BBS  6,ln_stat,LoopBackTSRE
        STI  #02Fh,ln_stat      ; set 4, leave 6 at 0
        !JMP LoopBackWait
5  LoopBackTSRE:
        STI  #06Fh,ln_stat      ; set 4, leave 6 at 1
        LoopBackWait:
        BBS  3,HostLCR,LoopBackAked ; set if host set
        LCR bit 3
10      !JMP LoopBackWait
        LoopBackAked:
        LDA  HostLCR
        AND  #0F7h
        STA  HostLCR
15  LoopBackNoDump:
        LDX  #07Fh
        TXS
        LDA  #HIGH(ECRefill)
        PHA
20      LDA  #LOW(ECRefill)
        PHA
        PSH
        TSX
        STX  ECStack
25      LDX  #0FFh
        TXS
        DecBankSelect
        JMP  DCFontParams
;
30  ;* * * * *
;
        SwitchToDecode:
        PSH
        TSX
35      STX  ECStack
        LDX  DCStack

```

- 76 -

APPENDIX 1

```

        TXS
        DecBankSelect
        PUL
        RTS
5 ;
    SwitchToEncode:
        PSH
        TSX
        STX  DCStack
10        LDX  ECStack
        TXS
        EncBankSelect
        PUL
        RTS
15 ;
    ; * * * * *
    ;
    ;    MISC READ FROM PC; USED ONLY FOR MEMORY LOAD; INTS ARE
    OFF
20 ;
    SubReadFromPC:
        LDA  Hostcontrl
        BPL  SubReadFromPC
        LDA  w8250_TXD
25        STI  #076h,HostContrl
        STI  #01Fh,ln_stat
        RTS
    ;
    ; * * * * *
30 ;
    ;    MISC WRITE TO PC
    ;
    SubWriteToPC:
        BBS  0,ln_stat,SubWriteToPC
35        BBS  0,ln_stat,SubWriteToPC    ; twice for SPERRY
    et al

```

- 77 -

APPENDIX 1

```

        STA  w8250_RXD
        BBS  6,ln_stat,SubWritePCTSRE
        STI  #03Eh,ln_stat      ; set 0, leave 6 at 0
        !JMP SubWritePCCont
5  SubWritePCTSRE:
        STI  #07Eh,ln_stat      ; set 0, leave 6 at 1
        SubWritePCCont:
        RTS
;
10 ;* * * * *
;
        MemoryDump:
        LDX  #000h
        LDY  #((RamPtr1+1)-000h)
15 MDLoop1FF:
        TXA
        JSR  SubWriteToPC
        INX
        DEY
20        BNE  MDLoop1FF
        LDY  #(080h-(RamPtr1+1)) ; X = #RamPtr1+1
        MDLoop1:
        LDA  PortA,X
        JSR  SubWriteToPC
25        INX
        DEY
        BNE  MDLoop1
        LDY  #(RamPtr0-080h)
        MDLoop2FF:
30        TXA
        JSR  SubWriteToPC
        INX
        DEY
        BNE  MDLoop2FF
35        LDY  #(100h-RamPtr0)      ; X = #RamPtr0
        MDLoop2:

```

- 78 -

APPENDIX 1

```

    LDA  PortA,X
    JSR  SubWriteToPC
    INX
    DEY
5      BNE  MDLoop2
    LDA  ECWord1+0
    STA  MDSave+0
    LDA  ECWord1+1
    STA  MDSave+1
10     STI  #008h,ECWord1+1
    STI  #000h,ECWord1+0
    MDLoop3:
    LDA  (ECWord1)
    JSR  SubWriteToPC
15     INC  ECWord1+0
    ifEQ
        INC  ECWord1+1
        LDA  ECWord1+1
        CMP  #040h
20     BEQ  MDLoop3Exit
    fi
    !JMP  MDLoop3
    MDLoop3Exit:
    EncBankSelect
25     STI  #040h,ECWord1+1
    STI  #000h,ECWord1+0
    MDLoop4:
    LDA  (ECWord1)
    JSR  SubWriteToPC
30     INC  ECWord1+0
    ifEQ
        INC  ECWord1+1
        LDA  ECWord1+1
        CMP  #0C0h
35     BEQ  MDLoop4Exit
    fi
```


- 79 -

APPENDIX 1

```

                !JMP MDLoop4
MDLoop4Exit:
                DecBankSelect
                STI    #040h,ECWord1+1
5              STI    #000h,ECWord1+0
MDLoop5:
                LDA    (ECWord1)
                JSR    SubWriteToPC
                INC    ECWord1+0
10             ifEQ
                INC    ECWord1+1
                LDA    ECWord1+1
                CMP    #0C0h
                BEQ    MDLoop5Exit
15             fi
                !JMP MDLoop5
MDLoop5Exit:
                LDA    MDSave+0
                STA    ECWord1+0
20             LDA    MDSave+1
                STA    ECWord1+1
                RTS
;
MemoryLoad:
25             LDX    #000h
                LDY    #((RamPtr1+1)-000h)
MLLoop1FF:
                JSR    SubReadFromPC
                INX
30             DEY
                BNE    MLLoop1FF
                LDY    #(080h-(RamPtr1+1))    ; X = #RamPtr1+1
MLLoop1:
                JSR    SubReadFromPC
35             STA    PortA,X
                INX

```

- 80 -

APPENDIX 1

```

        DEY
        BNE  MLLoop1
        LDY  #(RamPtr0-080h)
MLLoop2FF:
5         JSR  SubReadFromPC
        INX
        DEY
        BNE  MLLoop2FF
        LDY  #(100h-RamPtr0)      ; X = #RamPtr0
10  MLLoop2:
        JSR  SubReadFromPC
        STA  PortA,X
        INX
        DEY
15         BNE  MLLoop2
        LDA  ECWord1+0
        STA  MDSave+0
        LDA  ECWord1+1
        STA  MDSave+1
20         STI  #008h,ECWord1+1
        STI  #000h,ECWord1+0
        MLLoop3:
        JSR  SubReadFromPC
        STA  (ECWord1)
25         INC  ECWord1+0
        ifEQ
        INC  ECWord1+1
        LDA  ECWord1+1
        CMP  #040h
30         BEQ  MLLoop3Exit
        fi
        !JMP  MLLoop3
        MLLoop3Exit:
        EncBankSelect
35         STI  #040h,ECWord1+1
        STI  #000h,ECWord1+0
```

- 81 -

APPENDIX 1

```
MLLoop4:
    JSR    SubReadFromPC
    STA    (ECWord1)
    INC    ECWord1+0
5    ifEQ
        INC    ECWord1+1
        LDA    ECWord1+1
        CMP    #0C0h
        BEQ    MLLoop4Exit
10    fi
        !JMP    MLLoop4
MLLoop4Exit:
    DecBankSelect
    STI    #040h,ECWord1+1
15    STI    #000h,ECWord1+0
MLLoop5:
    JSR    SubReadFromPC
    STA    (ECWord1)
    INC    ECWord1+0
20    ifEQ
        INC    ECWord1+1
        LDA    ECWord1+1
        CMP    #0C0h
        BEQ    MLLoop5Exit
25    fi
        !JMP    MLLoop5
MLLoop5Exit:
    LDA    MDSave+0
    STA    ECWord1+0
30    LDA    MDSave+1
    STA    ECWord1+1
    STI    #0F7h,HostContrl
    RTS
;
35 MDSave:
    ORG    $+2
```

- 82 -

APPENDIX 1

```

;
;*****
;
;   READ FROM PC VIA INTERRUPT
5 ;
HostInt:
    PSH
    LDA Hostcontrl
    ifMI
10    LDX StorePtr
    LDA w8250_TXD
    STA InBuffer,X
    INX
    STX StorePtr
15    INX
    CPX FetchPtr
    ifNE
        STI    #01fh,ln_stat
    fi
20    fi
    BBR 5,Hostcontrl,HostInt1
    LDA w8250_LCR      ; save host command info
    STA HostLCR

HostInt1:
25    STI #007h,HostContrl
    PUL
    RTI

;
;*****
30 ;
;   ENCODER READ FROM PC
;
EReadCharacter:
    IF Test    ;{
35    LDA BytesIn+0
    CMP #050h

```

- 83 -

APPENDIX 1

```

        BNE ECRReadChar
        LDA BytesIn+1
        CMP #002h
        BNE ECRReadChar
5         LDA BytesIn+2
        CMP #000h
        BNE ECRReadChar
        NOP                ; set breakpoint here
        ENDIF              ;}

10  ECRReadChar:
        LDX  FetchPtr
        CPX  StorePtr
        ifEQ
        BBR 1,HostLCR,ECRReadChar      ; HostLCR is read in
15  interrupt
        SMB 7,HostLCR
        !JMP      ECRReadCharExit      ; bit 1 set when EOF
        and ptrs =
        fi
20         BBS  5,ln_stat,ECRReadCharLS
        STI  #01fh,ln_stat
        ECRReadCharLS:
        INC  FetchPtr
        INC  BytesIn+0
25         ifEQ
        INC  BytesIn+1
        ifEQ
        INC  BytesIn+2
        fi
30         fi
        BBR  6,HostLCR,ECRReadCharNLB
        LDA  ECCommand      ; flush input when Decoder
        BMI  ECRReadChar    ; has FailedSafe
        (LoopBack)
35  ECRReadCharNLB:
        LDA  InBuffer,X      ; A = char from PC

```

- 84 -

APPENDIX 1

```

ECReadCharExit:
    RTS

;
; * * * * *
5 ;
;   ENCODER WRITE TO PC
;
ECWriteCharacter:
    PSH
10    IF    Test    ;{
        LDA BytesOut+0
        CMP #06Bh
        BNE ECWriteSearched
        LDA BytesOut+1
15    CMP #001h
        BNE ECWriteSearched
        LDA BytesOut+2
        CMP #000h
        BNE ECWriteSearched
20    NOP          ; set breakpoint here
ECWriteSearched:
    ENDF          ;}
    BBR 6,HostLCR,ECWriteChar
    LDX OutStore
25    LDA ECBuffer
    STA OutBuffer,X
    INX
    STX OutStore
    INX
30    CPX OutFetch
    ifEQ
        JSR SwitchToDecode
    fi
    !JMP ECWriteCont
35 ECWriteChar:
    BBS 0,HostLCR,ECWriteCont

```

- 85 -

APPENDIX 1

```

        BBS  0,ln_stat,ECWriteChar
        BBS  0,ln_stat,ECWriteChar      ; twice for SPERRY
et al

        LDA  ECBuffer
5       STA  w8250_RXD
        BBS  6,ln_stat,ECWriteTSRE
        STI  #03Eh,ln_stat              ; set 0, leave 6 at 0
        !JMP ECWriteCont
ECWriteTSRE:
10      STI  #07Eh,ln_stat              ; set 0, leave 6 at 1
ECWriteCont:
        PUL
        STI  #001h,ECBuffer
        INC  BytesOut+0
15      ifEQ
        INC  BytesOut+1
        ifEQ
        INC  BytesOut+2
        fi
20      fi
        RTS
;
;* * * * *
;
25 ;   DECODER READ FROM PC
;
DCReadCharacter:
        PSH
        IF   Test      ;{
30      LDA BytesIn+0
        CMP  #0A0h
        BNE  DCReadChar
        LDA  BytesIn+1
        CMP  #005h
35      BNE  DCReadChar
        LDA  BytesIn+2

```

- 86 -

APPENDIX 1

```

        CMP #002h
        BNE DCReadChar
        NOP                ; set breakpoint here
        ENDIF              ;}

5  DCReadChar:
        BBR 6,HostLCR,DCReadCharNLB
        LDH OutFetch
        CPX OutStore
        ifEQ
10      JSR SwitchToEncode
        !JMP DCReadChar
        fi
        INC OutFetch
        LDA OutBuffer,X      ; A = char from Encoder
15      STA DCBuffer
        !JMP DCReadCharExit

DCReadCharNLB:
        LDH FetchPtr
        CPX StorePtr
20      ifEQ
        BBR 1,HostLCR,DCReadChar      ; HostLCR is read in
interrupt
        SMB 7,HostLCR      ; NOTE: not a normal EOF
        !JMP DCReadCharExit      ; bit 1 set when EOF

25  and ptrs =
        fi
        BBS 5,ln_stat,DCReadCharLS
        STI #01fh,ln_stat

DCReadCharLS:
30      INC FetchPtr
        INC BytesIn+0
        ifEQ
        INC BytesIn+1
        ifEQ
35      INC BytesIn+2
        fi

```


- 87 -

APPENDIX 1

```

        fi
        LDA  ECCCommand      ; flush input when Decoder
        BMI  DCReadChar      ; has FailedSafe
        LDA  InBuffer,X      ; A = char from PC
5         STA  DCBuffer
        DCReadCharExit:
        PUL
        RTS

;
10  ;* * * * *
;
;   DECODER WRITE TO PC
;
        DCWriteCharacter:
15         IF   Test      ;{
            PHA
            LDA  BytesOut+0
            CMP  #027h
            BNE  DCWriteSearched
20         LDA  BytesOut+1
            CMP  #017h
            BNE  DCWriteSearched
            LDA  BytesOut+2
            CMP  #000h
25         BNE  DCWriteSearched
            NOP                      ; set breakpoint here
        DCWriteSearched:
            PLA
            ENDIF                  ;}
30  DCWriteChar:
        BBS  0,HostLCR,DCWriteCont
        BBS  0,ln_stat,DCWriteChar
        BBS  0,ln_stat,DCWriteChar      ; twice for SPERRY
        et al
35         STA  w8250_RXD
        BBS  6,ln_stat,DCWriteTSRE

```

- 88 -

APPENDIX 1

```

        STI    #03Eh,ln_stat      ; set 0, leave 6 at 0
        !JMP   DCWriteCont
DCWriteTSRE:
        STI    #07Eh,ln_stat      ; set 0, leave 6 at 1
5  DCWriteCont:
        STI    #001h,ECBuffer
        BBS    6,HostLCR,DCWriteCharNLB
        INC    BytesOut+0
        ifEQ
10         INC  BytesOut+1
        ifEQ
            INC    BytesOut+2
        fi
        fi
15 DCWriteCharNLB:
        RTS
;
;*****
;
20         ENDIF                      ;} end of loopback enc/dec
routines
;
;***** ENCODER MACROS *****
;
25 Write17  MACRO
        IF    Macros
            MSWrite17
        ELSE
            JSR MSWrite17
30         ENDIF
        ENDM
;
Write8      MACRO
        IF    Macros
35         MSWrite8
        ELSE

```


- 90 -

APPENDIX 1

```

;*****
;
;   WRITE ONE BYTE OF BITS
;
5       IF    Macros
MSWrite8    MACRO
            LOCAL    Write8Loop,Write8Skip,Write8Exit
            ELSE
MSWrite8:
10        ENDIF
            ASL    A
            ORA    #001h
            !JMP Write8Skip
Write8Loop:
15        ASL    A
            BEQ    Write8Exit
Write8Skip:
            ROL    ECBuffer
            BCC    Write8Loop
20        JSR    ECWriteCharacter
            !JMP Write8Loop
Write8Exit:
            IF    Macros
            ENDM
25        ELSE
            RTS
            ENDIF
;
;*****
30 ;
;   WRITE ONE+ BYTE(S) OF BITS
;
            IF    Macros
MSWrite817    MACRO
35        LOCAL
Write817Loop1,Write817Skip,Write817Exit1,Write817Loop2,Write

```

- 91 -

APPENDIX 1

```

      817Exit2
          ELSE
      MSWrite817:
          ENDIF
5          ASL  A
          ORA  #001h
          !JMP Write817Skip
      Write817Loop1:
          ASL  A
10         BEQ  Write817Exit1
      Write817Skip:
          ROL  ECBuffer
          BCC  Write817Loop1
          JSR  ECWriteCharacter
15         !JMP Write817Loop1
      Write817Exit1:
          TXA
      Write817Loop2:
          ASL  A
20         BEQ  Write817Exit2
          ROL  ECBuffer
          BCC  Write817Loop2
          JSR  ECWriteCharacter
          !JMP Write817Loop2
25  Write817Exit2:
          IF  Macros
          ENDM
          ELSE
          RTS
30         ENDIF
      ;
      ; * * * * *
      ;
      ;   SET POINTER TO NCChar,NCFreq TABLES
35  ;
      SetCharFreq    MACRO      ED,BW
                                ; NCFreqSets =

```

- 92 -

APPENDIX 1

```

4
    LDA ED&Char1Prior
    CLC
    AND #060h
5    ROL A
    ROL A
    ROL A
    ROL A
    IF "&BW" EQ "WB"
10    STA ED&Byte3          ; 0-3
        ASL A              ; HIGH((0-3)*512)
        ADD #HIGH(ED&NCChar)
        STA ED&Word1+1      ; Word1+1 = HIGH(base of
NCChar)
15    ADD #001h
        STA ED&Word2+1      ; Word2+1 = HIGH(base of
NCFreq)

        STI #000h,ED&Word1+0
        STI #000h,ED&Word2+0
20    ENDIF
    IF "&BW" EQ "W1"
        ASL A              ; HIGH((0-3)*512)
        ADD #HIGH(ED&NCChar)
        STA ED&Word1+1      ; Word1+1 = HIGH(base of
25 NCChar)
    ENDIF
    IF "&BW" EQ "W2"
        ASL A              ; HIGH((0-n)*512)
        ADD #(HIGH(ED&NCChar)+1)
30    STA ED&Word2+1      ; Word2+1 = HIGH(base of
NCFreq)
    ENDIF
    ENDM
;
35 ;***** FONT UPDATE MACRO *****
;

```

- 93 -

APPENDIX 1

```

;   FONT UPDATE MACRO
;
;   IF EC, Y = ECNextChar
;
5  FontUpdate      MACRO      XX,YY
        LOCAL
        Font1stUse,FontActive,ECNC8Bit,ECNCGlobal,ECNCCoded
        LDA  XX&CurrentHash+1      ; HIGH of prior hash
        IF   "&XX" EQ "DC"      ;{
10         BPL FontActive
        JMP DCNewCharacter
        ELSE                      ;{}
        STA ECFontBase+1          ; (stored as * 2)
        LDA ECCurrentHash+0        ; LOW  of prior hash
15         ASL A
        ROL ECFontBase+1          ; now = * 4
        ASL A
        ROL ECFontBase+1          ; now = * 8
        IF   FontSize EQ 16
20         ASL      A
        ROL      ECFontBase+1      ; now = * 16
        ENDIF
        STA ECFontBase+0
        TAX                      ; save for PHX
25         LDA ECCurrentHash+1      ; HIGH of prior hash
        BPL FontActive
        Font1stUse:
        IF   "&YY" EQ "FU"      ;{
        LDA ECFontBase+1
30         ADD #HIGH(ECFontTables)
        STA ECFontBase+1
        LDA #000h
        STA ECCharacters
        STA ECNCIndex
35         ENDIF                      ;}
        LDA #000h

```

- 94 -

APPENDIX 1

```

        STA ECNewIndex,Y          ; zero when new font
    IF    "&YY" EQ "FU"          ;{
        JMP ECNewCharacter        ; Characters = 0
    ELSE          ;{}
5      JMP ECNewCharCommand
    ENDIF          ;}
        ENDIF          ;}

FontActive:
    IF    "&XX" EQ "DC"          ;{
10     LDA DCFontIndex
        ifPL          ;{
            LDA    DCFontIndex
            STI    #080h,DCFontIndex
            CMP    DCNCIndex
15     BCC    DCKCharLTNCIndex
        BEQ    DCKCharEQNCIndex

DCKCharGTNCIndex:
            !JMP    DCCharSwap

DCKCharEQNCIndex:
20     INC    DCNCIndex          ; bump NCIndex
        JMP    DCFontUpdated

DCKCharLTNCIndex:
            AND    #0FFh
            BNE    DCCharSwap
25     JMP    DCFontUpdated
        els ;{}
            LDA    DCFontBase+1
            LDX    DCFontBase+0
            PHA                      ; push address back on
30     stack
            PHX                      ; and pull to I
            PLI
            LAN                      ; NCIndex or CharsNCIndex
            IF TwoBytes ;{
35     LAN                      ; Characters
            ENDIF          ;}

```


- 95 -

APPENDIX 1

```

        fi ;}
ELSE      ;{}
        LDA ECFontBase+1
        ADD #HIGH(ECFontTables)
5         STA ECFontBase+1
        PHA                ; push address back on stack
        PHX                ; and pull to I
        PLI
        LAN                ; NCIndex or CharsNCIndex
10        IF TwoBytes ;{
        STA XX&NCIndex
        LAN                ; Characters
        STA XX&Characters
        ELSE                ;{}
15        TAW                ; W = CharsNCIndex
        AND #00Fh
        STA XX&NCIndex      ; NCIndex in bits
        3-0
        TWA
20        ASR A
        ASR A
        ASR A
        ASR A
        AND #00Fh
25        STA XX&Characters ; Characters in bits
        7-4
        ENDIF                ;}
        ADD ECABStatus        ; +1 if 8-bit active
        TAX                ; X = # of font indices (base
30 0)
        LDA EncodingTable,X    ; A = FontCode(Bits)
        offset
        STA ECByte1            ; Byte1 = EncodingTable
        base ptr
35        ADD ECNCIndex        ; Byte1 + NCIndex
        STA ECNewIndex,Y

```

- 96 -

APPENDIX 1

```

                                ENDIF                                ;}
                                IF      "&YY" EQ "FU"                ;{
XX&FontSearch:                                ; I = ptr to 1st
font character
5          LDX  XX&Characters                ; X = # of characters to
match
XX&FontSearchLoop:
          LAN
          CMP  XX&CurrentChar
10         BEQ  XX&CharFound
          DEX
          BNE  XX&FontSearchLoop
XX&CharNotFound:
          JMP  XX&NewCharacter
15 XX&CharFound:
          TXA
          NEG  A
          ADD  XX&Characters
          IF   "&XX" EQ "EC"                                ; W = A = character
20         TAW                                ; table index, base 0
          ENDIF
          CMP  XX&NCIndex                        ; (= character encoding
index
          BCC  XX&CharLTNCIndex                ; if < NCIndex)
25         BEQ  XX&CharEQNCIndex
XX&CharGTNCIndex:
          IF   "&XX" EQ "EC"
          ADD  XX&ABStatus                        ; 0 or 1
          ADD  #002h                            ; A = character encoding
30 index
          ADD  XX&Byte1                        ; + character table index
          STA  ECFontIndex,Y                    ; W = A = character table
index
          TWA                                ; for table swap
35         ENDIF
          !JMP XX&CharSwap

```

- 97 -

APPENDIX 1

XX&CharEQNCIndex:

```

        IF    "&XX" EQ "EC"
            ADD XX&ABStatus          ; 0 or 1
            ADD #002h                ; A = character encoding
5   index
            ADD XX&Byte1            ; + character table index
            STA ECFontIndex,Y
        ENDIF
        INC  XX&NCIndex              ; bump NCIndex
10      !JMP XX&FontEncoding

```

XX&CharLTNCIndex:

```

        IF    "&XX" EQ "EC"
            ADD XX&Byte1            ; A = character encoding index
            STA ECFontIndex,Y        ; + character table
15   index
            TWA                      ; W = character table index
        ELSE
            AND #0FFh
        ENDIF
20      BEQ  XX&FontEncoding          ; no swap if already
        index 0
        XX&CharSwap:                  ; A = character index,
        base 0

```

```

            ADD  #(CharTable-1)      ; ptr to previous
25   character
            TAX
            LDA  (XX&FontBase),X
            INX
            STA  (XX&FontBase),X
30      LDA  XX&CurrentChar
            DEX
            STA  (XX&FontBase),X

```

XX&FontEncoding:

```

        IF    "&XX" EQ "DC"
35      JMP  DCFontUpdated
        ELSE

```

- 98 -

APPENDIX 1

```

        LDA #002h
        STA ECType,Y          ; Type 2 - normal font
encoding
        BBS 0,ECABStatus,ECFontSaveEight
5        JMP ECFontUpdated
ECFontSaveEight:
        SetCharFreq  XX,W2          ; output is NC
frequency value
        LDA ECCurrentChar          ; save for consistent
10 'strings
        STA ECWord2+0          ; off' ouput write code
        LDA (ECWord2)
        STA ECFrequency,Y
        JMP ECFontUpdated
15        ENDIF
XX&NewCharacter:
        LDA XX&NCIndex
        ifNE
        DEC XX&NCIndex
20        fi
        SetCharFreq  XX,WB
        LDX XX&CurrentChar
        LDA (XX&Word2),X          ; frequency of current
character
25        STA XX&CurrentFreq
        BEQ XX&NewCharOK
        LDX XX&Byte3          ; 0-n where n = 0, 3, 7 or 15
        LDA XX&GlobalHigh,X
        CMP XX&CurrentFreq
30        BCS XX&NewCharSwap      ; CurrentFreq <=
GlobalHigh
XX&NewCharExchange:
        INC XX&GlobalHigh,X
        TAX          ; X = high frequency
35        LDA (XX&Word1),X
        TAW          ; W = high character

```

- 99 -

APPENDIX 1

```

        LDA  XX&CurrentChar
        STA  (XX&Word1),X           ; current char > high
char
        TXA
5        LDX  XX&CurrentChar
        STA  (XX&Word2),X           ; high freq > char freq
        LDX  XX&CurrentFreq
        TWA
        STA  (XX&Word1),X           ; high char > current
10 char
        TAX
        LDA  XX&CurrentFreq
        STA  (XX&Word2),X           ; current freq > high
freq
15        !JMP XX&NewCharOK
        XX&NewCharSwap:
        LDX  XX&CurrentFreq
        DEX                      ; X = lower freq
        LDA  (XX&Word1),X
20        TAW                      ; W = lower char
        LDA  XX&CurrentChar
        STA  (XX&Word1),X           ; current char > lower
char
        TXA
25        LDX  XX&CurrentChar
        STA  (XX&Word2),X           ; lower freq > char freq
        LDX  XX&CurrentFreq
        TWA
        STA  (XX&Word1),X           ; lower char > current
30 char
        TAX
        LDA  XX&CurrentFreq
        STA  (XX&Word2),X           ; current freq > lower
freq
35 XX&NewCharOK:
        LDA  XX&Characters

```

- 100 -

APPENDIX 1

```

        CMP  #CharsPerFont
        BEQ  XX&NewCharOverflow ; check for font table
full
        INC  XX&Characters      ; if not, add to char
5 count
        ADD  #001H
        XX&NewCharOverflow:
        ADD  #(CharTable-1)
        TAX
10      LDA  XX&CurrentChar      ; store current char in
font
        STA  (XX&FontBase),X
        LDX  XX&CurrentFreq      ; X = ECCurrentFreq
        LDA  GlobalBits,X
15      ADD  XX&NCBitsNew+0
        STA  XX&NCBitsNew+0      ; update NC trending
total
        ifCS
        INC  XX&NCBitsNew+1
20      fi
        ELSE      ;{}
        ECNewCharCommand:
        LDX  #0FFh              ; X = ECCurrentFreq for
command
25      ENDIF      ;}
        IF  "&XX" EQ "EC"
        BBR  0,ECABStatus,ECNCGlobal
        ECNC8Bit:
        TXA              ; output is NC frequency value
30      STA  ECFrequency,Y      ; save for consistent
'strings
        !JMP  ECNCCoded      ; off' ouput write code
        ECNCGlobal:
        TXA              ; global index for write
35      STA  ECFontIndex,Y
        ECNCCoded:              ; prod/commands do not

```

- 101 -

APPENDIX 1

```

affect
        LDA #004h                ; any of the trending
totals,
        STA ECType,Y            ; tables or hashes
5      ENDIF
        IF "&YY" EQ "FU"        ;{
XX&NCTrending:
        DEC XX&NCCounter
        BNE XX&FontUpdated
10     STI #NC8BitCycle,XX&NCCounter
        LDY #000h                ; Word1 =
        LDX XX&NCBitsNew+0        ; NCBitsPrior +
NCBitsNew
        TXA
15     ADD XX&NCBitsPrior+0        ; NCBitsPrior set to
        STA XX&Word1+0            ; NCBitsNew
        STX XX&NCBitsPrior+0
        STY XX&NCBitsNew+0        ; NCBitsNew set to 0
        LDX XX&NCBitsNew+1
20     TXA
        ifCS
        ADD #001h                ; if low order carry
        fi
        ADD XX&NCBitsPrior+1
25     STA XX&Word1+1
        STX XX&NCBitsPrior+1
        STY XX&NCBitsNew+1
        BBR 0,XX&ABStatus,XX&NCOff
XX&NCon:
30     LDA #HIGH(NC8BitCycle*15)
        CMP XX&Word1+1
        BCC XX&FontUpdated        ; HIGH(Word1) > A
        BNE XX&TurnNCOff          ; HIGH(Word1) < A
        LDA #LOW(NC8BitCycle*15)
35     CMP XX&Word1+0
        BCC XX&FontUpdated

```

- 102 -

APPENDIX 1

```

XX&TurnNCOff:
    STI #000h,XX&ABStatus
    IF "&XX" EQ "EC"
        STI #001h,ECABChange
5       IF Test
            INC SwitchToA+0
            ifEQ
            INC SwitchToA+1
            fi
10      ENDIF
    ENDIF
    !JMP XX&FontUpdated

XX&NCOff:
    LDA #HIGH(NC8BitCycle*15)
15     CMP XX&Word1+1
        BCC XX&TurnNCon ; HIGH(Word1) > A
        BNE XX&FontUpdated ; HIGH(Word1) < A
    LDA #LOW(NC8BitCycle*15)
    CMP XX&Word1+0
20     BCS XX&FontUpdated

XX&TurnNCon:
    IF "&XX" EQ "EC"
        STI #001h,ECABStatus
        STI #001h,ECABChange
25     IF Test
            INC SwitchToB+0
            ifEQ
            INC SwitchToB+1
            fi
30     ENDIF
    ELSE
        STI #001h,DCABStatus
    ENDIF
    IF NCFreqSetsReset
35     LDA #NCFreqSetsHigh
        LDX #NCFreqSets

```


- 103 -

APPENDIX 1

```

XX&ResetGlobalHigh:
    STA XX&GlobalHigh-1,X
    DEX
    BNE XX&ResetGlobalHigh
5    ENDIF
XX&FontUpdated:
    IF    TwoBytes
        LDA XX&NCIndex
        STA (XX&FontBase)
10    LDX #001h
        LDA XX&Characters
        STA (XX&FontBase),X
    ELSE
        LDA XX&Characters
15    ASL A
        ASL A
        ASL A
        ASL A
        ORA XX&NCIndex
20    STA (XX&FontBase)
    ENDIF
XX&PlusHash:
    LDY XX&Char1Prior
    STY XX&Char2Prior
25    LDX CRC_TH,Y
    LDA XX&CurrentChar
    STA XX&Char1Prior
    ;    NEG    A                ; extra NEG over 1st try ?????
    EOR CRC_TL,Y
30    TAY
    TXA
    EOR CRC_TL,Y
    TAW                ; W = LOW(rough hash)
    IF    "&XX" EQ "EC"
35    LDX ECNextChar    ; ECHashRaw is bits 15-0

```

of

- 104 -

APPENDIX 1

```

        STA ECHashRaw0,X          ; the CRC
    ENDIF
    LDA  CRC_TH,Y                ; A = HIGH(rough hash)
    IF   "&XX" EQ "EC"
5      STA ECHashRaw1,X
    ENDIF
    STA  XX&Word1+1
    AND  #HIGH(MatchMask)
    STA  XX&Byte1                ; Byte1 = match bits
10    TWA
    ASL  A
    ROL  XX&Word1+1
    STA  XX&Word1+0
    LDA  XX&Word1+1
15    AND  #HIGH(NextMask)
    ADD  #HIGH(XX&FTHashRough)    ; Word1 = ptr to
    STA  XX&Word1+1              ; FTHashRough
    LDX  #001h                  ; X = 1 for all of
        PlusHash
20    XX&PlusFineLoop:
        LDA  (XX&Word1),X        ; direct ptr, can't be 0
        BEQ  XX&PlusNewHash
        TAY                          ; Word1 may be either Rough
        LDA  (XX&Word1)          ; or Next; is always the
25    AND  #0FEh                ; predecessor to new
        hash
        STA  XX&Word1+0
        TYA
        ADD  #(HIGH(FTHashMatch)-HIGH(XX&FTHashNext))
30    STA  XX&Word1+1            ; Word1 = ptr to
        IF   "&XX" EQ "EC"        ; FTHashMatch
            LDA  (ECWord1),X
        ELSE                      ; HashMatch values are inter-
            LDA  (DCWord1)        ; mixed Decoder/Encoder
35    ENDIF
        CMP  XX&Byte1

```

- 105 -

APPENDIX 1

```

        BEQ  XX&PlusFineFound
        STY  XX&Word1+1          ; Word1 = ptr to
        !JMP XX&PlusFineLoop    ;      FTHashNext
XX&PlusFineFound:
5         TYA
        STA  XX&Word1+1
        ADD  #(0-HIGH(XX&FTHashNext))
        STA  XX&CurrentHash+1
        IF   "&XX" EQ "EC"
10        LDY ECNextChar
        STA  ECHashX21,Y
        ENDIF
        LDA  XX&Word1+0
        STA  XX&CurrentHash+0
15        IF   "&XX" EQ "EC"
        STA  ECHashX20,Y
        ENDIF
        JMP  XX&PlusHashExit
XX&PlusNewHash:
        ; Byte1 = match bits
20        LDY  XX&FTLastHash+1    ; Word1 = rough hash
        * 2
        BEQ  XX&PlusNewSearch
XX&PlusNew1stPass:
        LDA  XX&FTLastHash+0      ; LastHash
25 initialized to 2
        ADD  #002h                ; + HIGH(FTHashNext)
        STA  XX&FTLastHash+0
        BNE  XX&PlusNew1stCont
        INY
30        CPY  #(HIGH(XX&FTHashNext)+HIGH(FontTables*2))
        ifEQ
        STI  #000h,XX&FTLastHash+1 ; 0 when wrapped to
force search
        !JMP  XX&PlusNewSearch
35        els
        STY  XX&FTLastHash+1

```

- 106 -

APPENDIX 1

```

fi
XX&PlusNew1stCont:
    STA XX&Word2+0          ; Word2 = ptr to new's
    IF "&XX" EQ "DC"        ; FTHashMatch
5      STA DCWord3+0
    ENDIF                  ; Word3 = ptr to new's
    TYA                    ; DCFTHashChars
    STA (XX&Word1),X
    ADD #(HIGH(FTHashMatch)-HIGH(XX&FTHashNext))
10   STA XX&Word2+1
    IF "&XX" EQ "EC"
        ADD #(0-HIGH(FTHashMatch))
    ELSE
        ADD #(HIGH(DCFTHashChars)-HIGH(FTHashMatch))
15   STA DCWord3+1
        ADD #(0-HIGH(DCFTHashChars))
    ENDIF
    ORA #080h              ; set bit 7 for new hash
    STA XX&CurrentHash+1
20   IF "&XX" EQ "EC"
        LDY ECNextChar      ; store new hash in
EHashX2
        STA EHashX21,Y      ; or in DCCurrentHash
    ENDIF
25   LDA XX&FTLastHash+0
    STA (XX&Word1)
    STA XX&CurrentHash+0
    IF "&XX" EQ "EC"
        STA EHashX20,Y
30   ELSE
        LDA DCChar2Prior    ; store prior/current
chars
        STA (DCWord3)       ; in DCFTHashChars
        LDA DCChar1Prior
35   STA (DCWord3),X
    ENDIF

```

- 107 -

APPENDIX 1

```

        LDA  XX&Byte1
        IF   "&XX" EQ "EC"
            STA (ECWord2),X           ; HashMatch values are
inter-
5         ELSE                       ; mixed Decoder/Encoder
            STA (DCWord2)
        ENDIF
        JMP  XX&PlusHashExit
XX&PlusNewSearch:
10        LDA  (XX&FTPParent),X       ; initialized to
        FTHashRough
            BNE  XX&PlusNewParent
XX&PlusNewRoughAdvance:
            LDA  XX&FTNextRough+0     ; initialized to
15  FTHashRough
            ADD  #002h
            STA  XX&FTNextRough+0
            ifEQ
                INC XX&FTNextRough+1   ; memory allocation
20  dependent
            LDY  #HIGH(XX&FTHashNext) ; i.e. FTHashRough
        table must
            CPY  XX&FTNextRough+1     ; be right before
        FTHashNext
25        ifEQ
            STI   #HIGH(XX&FTHashRough),XX&FTNextRough+1
            fi
            fi
            LDA  (XX&FTNextRough),X
30        BEQ  XX&PlusNewRoughAdvance
            LDY  XX&FTNextRough+0
            STY  XX&FTPParent+0
            LDY  XX&FTNextRough+1
            STY  XX&FTPParent+1
35  XX&PlusNewParent:
            STA  XX&FTChild+1

```

- 108 -

APPENDIX 1

```

        LDA  (XX&FTPParent)
        STA  XX&FTChild+0
        TAY                      ; Y = FTChild+0
        !JMP XX&PlusNewNext2Found

5  XX&PlusNewNextAdvance:
        LDA  XX&FTChild+1
        STA  XX&FTPParent+1
        LDA  XX&FTChild+0
        STA  XX&FTPParent+0
10      !JMP XX&PlusNewSearch

XX&PlusNewNext2Found:
        CPY  XX&Word1+0
        ifEQ
        LDA  XX&FTChild+1
15      CMP  XX&Word1+1
        BEQ  XX&PlusNewNextAdvance
        fi
        LDA  (XX&FTChild)
        STA  (XX&FTPParent)
20      LDA  (XX&FTChild),X
        STA  (XX&FTPParent),X
        STY  XX&Word2+0          ; Word2 = ptr to new's
        IF   "&XX" EQ "DC"      ;      FTHashMatch
        STY  DCWord3+0
25      ENDIF                    ; Word3 = ptr to new's
        TYA                      ;      DCFTHashChars
        STA  (XX&Word1)
        LDA  XX&FTChild+1
        STA  (XX&Word1),X
30      ADD  #(HIGH(FTHashMatch)-HIGH(XX&FTHashNext))
        STA  XX&Word2+1
        IF   "&XX" EQ "EC"
        ADD  #(0-HIGH(FTHashMatch))
        ELSE
35      ADD  #(HIGH(DCFTHashChars)-HIGH(FTHashMatch))
        STA  DCWord3+1

```

- 109 -

APPENDIX 1

```

        ADD #(0-HIGH(DCFTHashChars))
    ENDIF
    ORA  #080h                ; set bit 7 for new hash
    STA XX&CurrentHash+1
5      IF  "&XX" EQ "EC"
        LDY ECNextChar        ; store new hash in
    EHashX2
        STA EHashX21,Y        ; or in DCCurrentHash
    ENDIF
10     LDA XX&Word2+0
        STA XX&CurrentHash+0
        IF  "&XX" EQ "EC"
            STA EHashX20,Y
        ELSE
15         LDA DCChar2Prior    ; store prior/current
    chars
            STA (DCWord3)      ; in DCFTHashChars
            LDA DCChar1Prior
            STA (DCWord3),X
20     ENDIF
        LDA XX&Bytel
        IF  "&XX" EQ "EC"
            STA (ECWord2),X    ; HashMatch values are
    inter-
25     ELSE                ; mixed Decoder/Encoder
        STA (DCWord2)
    ENDIF
        LDA  #000h
        STA  (XX&FTChild),X
30     STA  (XX&FTChild)
    XX&PlusHashExit:
        IF  "&XX" EQ "DC";{
            LDA DCCurrentHash+0    ; LOW of prior hash
            STA DCFontBase+0
35         LDA DCCurrentHash+1    ; HIGH of prior hash
            AND #07Fh

```

- 110 -

APPENDIX 1

```

CLC
ROL DCFontBase+0
ROL A          ; now = * 4
ROL DCFontBase+0
5  ROL A          ; now = * 8
IF FontSize EQ 16;{
    ROL    DCFontBase+0
    ROL    A          ; now = * 16
ENDIF          ;}
10  ADD #HIGH(DCFontTables)
    STA DCFontBase+1
    LDA DCCurrentHash+1      ; HIGH of prior hash
    ifMI
        LDA    #000h          ; new font
15  STA    DCCharacters
    STA    DCNCIndex
    els
        LDA    (DCFontBase)      ; old font
        IF TwoBytes      ;{
20  STA    DCNCIndex      ; NCIndex
        LDX    #001h
        LDA    (DCFontBase),X      ; Characters
        STA    DCCharacters
        ELSE          ;{}
25  TAW          ; W = CharsNCIndex
        AND    #00Fh
        STA    DCNCIndex      ; NCIndex in bits 3-0
        TWA
        ASR    A
30  ASR    A
        ASR    A
        ASR    A
        AND    #00Fh
        STA    DCCharacters      ; Characters in bits
35 7-4
    ENDIF          ;}

```


- 111 -

APPENDIX 1

```

        fi
        ENDIF                                ;}
    ENDIF                                ;}
    ENDM

5  ;
    ;***** E N C O D E R   R E F I L L *****
    ;
    ;   ENCODER REFILL
    ;
10  ECR refill:
        IF   Prodder
            DEC ProdCounter
            ifEQ
                STI    #ProdCycle,ProdCounter
15         STI    #0A0h,ECCCommand
            JMP    ECProdCommand
        fi
    ENDIF
    JSR    ECReadCharacter    ; A = char from PC
20  IF    EOFControl
        BBR 7,HostLCR,ECRefillRepeats
        STI #0C8h,ECCCommand
        JMP ECProdCommand
    ENDIF
25  ECR refillRepeats:
        IF   Repeats    ;{
            CMP ECChar2Prior
            BNE ECR refillUpdate
            CMP ECChar1Prior
30         BNE ECR refillUpdate
            LDY ECRepeatCount    ; 3 in a row are equal
            BEQ ECR refill1stRepeat
            INC ECRepeatCount
            BEQ ECR refill256thRepeat ; ECR repeats = 100h
35         !JMP    ECR refill
    ECR refill1stRepeat:

```

- 112 -

APPENDIX 1

```

        STI #001h,ECRepeatCount
        !JMP     ECTRefill
ECTRefill1256thRepeat:
        STI #0FFh,ECRepeatCount
5         ENDIF          ;}
ECTRefillUpdate:
        LDY  ECNextChar
        IF   Repeats
            LDX ECRepeatCount          ; ECRepeatCount = 0 to
10  0FFh
            ifNE
                STA      ECCharSave
                TXA
                STA      ECRepeats,Y
15         LDA      #008h
                STA      ECRepeatSW,Y          ; repeat character
            is
                LDA      ECChar1Prior          ; swapped with new
            character
20         fi
        ENDIF
        STA  ECChar,Y          ; A = new character
        STA  ECCharCopy,Y
        STA  ECCurrentChar
25         IF   Test
            LDA  ECABStatus
            ifEQ
                INC      AStringsOn+0
                ifEQ
30         INC      AStringsOn+1
            fi
        els
            INC      BStringsOn+0
            ifEQ
35         INC      BStringsOn+1
            fi

```

- 113 -

APPENDIX 1

```

        fi
    ENDIF
    FontUpdate      EC,FU
    LDA  ECABChange          ; either condition
5  requires
        ORA  ECCommand      ; flushing the ECChar buffer
        ifEQ ;{
            INC ECAvailable
            ifEQ              ; 256 characters
10  available
                LDY      ECABStatus
                ifEQ
                JSR      StringATime
                els
15                JSR      StringBTime
                fi
                fi
            els
                STA ECFlush
20                INC ECAvailable
                LDA ECABChange
                ifNE              ; if ABStatus change,
this pass
                EOR      ECABStatus          ; is cleaning up
25  remaining
                ifEQ              ; characters from prior
status
                IF      AntiEx
                STI      #000h,ECAntiEStatus
30                ENDIF
                JSR      StringATime
                els
                JSR      StringBTime
                fi
35                els
                LDA      ECABStatus

```

- 114 -

APPENDIX 1

```

        ifEQ
        JSR    StringATime
        els
        JSR    StringBTime
5         fi
        fi
        STI #000h,ECABChange
        STI #000h,ECFlush
        LDA ECCommand
10        BEQ ECTRefillReturn
        RTS
        fi    ;}
ECTRefillReturn:
        INC ECNextChar          ; INC here to avoid
15 ECChar
        IF Repeats              ; buffer advance on
        LDA ECRepeatCount       ; prods/commands
        ifNE
        STI    #000h,ECRepeatCount
20        LDA    ECCharSave          ; use saved
character which
        JMP    ECTRefillRepeats    ; forced repeat
output
        fi
25        ENDIF
        JMP ECTRefill
;
ECTProdCommand:
        LDA ECAvailable
30        ifNE
        STI #0FFh,ECFlush
        LDY ECABStatus
        ifEQ
        JSR    StringATime
35        els
        JSR    StringBTime

```

- 115 -

APPENDIX 1

```

        fi
        STI #000h,ECFlush
    fi
    IF Repeats
5        LDY ECRepeatCount
        ifNE
            LDA    ECChar1Prior        ; set up repeat
            character
                JSR    ECRefillUpdate    ; as new character
10        STI    #000h,ECRepeatCount
            INC    ECNextChar
        fi
        ENDIF
        LDY ECNextChar
15        FontUpdate    EC,PC
        LDA ECABStatus
        ifNE
            JMP ECProdB
        fi
20 ECProdA:
        LDX ECNewIndex,Y
        ifEQ
            JMP ECProdANCF
        fi
25        LDA FontCode,X
        Write17
        ECProdANCOF:
            LDA GlobalCodeHigh+255
            LDX GlobalCodeLow+255
30        BEQ ECProdANCOFHigh
            Write817
            JMP ECProdShift
        ECProdANCOFHigh:
            Write17
35        JMP ECProdShift
        ECProdANCF:

```

- 116 -

APPENDIX 1

```

    LDA GlobalCodeLow+255
    STA ECByte4
    LDA GlobalCodeHigh+255
    ASR A
5    ROR ECByte4
    LDX ECByte4
    Write817
    !JMP ECProdShift
    IF AntiEx
10 ECProdNoStrings:
    LDA #0FFh ; plain 0FFh
    Write8
    JMP ECProdShift
    ECProdB:
15    LDA ECAntiEStatus
    BMI ECProdNoStrings
    ELSE
    ECProdB:
    ENDIF
20    LDX ECNewIndex,Y
    BEQ ECProdBNF
    LDA FontCode,X
    Writel7
    LDA #0FFh ; FFh = 11111111
25    Write8
    !JMP ECProdShift
    ECProdBNF:
    LDA #0BFh ; FFh = 10111111
    LDX #0C0h
30    Write817
    ECProdShift:
    LDA ECCommand
    STI #000h,ECCommand
    Writel7
35 ECProdShiftLoop:
    LDA ECBuffer
```

- 117 -

APPENDIX 1

```

        CMP    #001h
        ifEQ
            JMP ECProdDone
        fi
5         LDA    #040h
        Write17
        !JMP ECProdShiftLoop
ECProdDone:
        IF     EOFControl
10         BBR 7,HostLCR,ECToRefill
            BBR 6,HostLCR,ECProdNLB
            JSR SwitchToDecode
ECProdNLB:
            JMP DCoreCEEOF
15 ECToRefill:
            JMP ECToRefill
            ELSE
            RTS
            ENDIF
20 ;
;***** A - S T R I N G   M A C R O S *****
;
;   A-STRING HASH HEAD AND SEARCH MACRO
;
25 FindAString    MACRO                                ; A = 1st ECChar ptr
                LOCAL
                FindABackOK,FindAMoreLoop,FindAUpdate,FindASkip,FindABackMat
                ch,FindABackLoop,FindAReturn
                STA    ECByte1
30         STI    #003h,ECByte3
                ADD    #001h
                TAX                                ; Y = ECNextOut+ECFind4s+1
                ADD    #002h
                TAY                                ; Y = ECNextOut+ECFind4s+3
35         LDA    ECHashRaw0,X
                NEG    A

```

- 118 -

APPENDIX 1

```

EOR  ECHashRaw0,Y
STA  ECFindHash
STA  ECWord3+0      ; Word3 = ptr to
LDA  ECHashRaw1,X      ;      RRHashHead
5   NEG  A
EOR  ECHashRaw1,Y
AND  #HIGH(BufferHashes-1)
ASL  ECWord3+0
ROL  A
10  ADD  #HIGH(ECRRHashHead)
STA  ECWord3+1
LDX  #001h          ; Word2 = ptr to 1st
LDA  (ECWord3),X      ; RRBuffer location
ifNE          ; for this hash
15  STA  ECWord4+1
ADD  #(HIGH(ECRRBuffer)-HIGH(ECRRHashLink))
STA  ECWord2+1
LDA  (ECWord3)      ; Word4 = ptr to 1st
STA  ECWord2+0      ; RRHashLink location
20  STA  ECWord4+0      ; for this hash
STI  #MaximumASearches,ECWord1+1
!JMP  FindABackMatch
fi
IF    Test
25  ifEQ
INC      FSNoHash+0
ifEQ
INC      FSNoHash+1
ifEQ
30  INC  FSNoHash+2
fi
fi
fi
ENDIF
35  !JMP FindAReturn
FindASkip:

```


- 119 -

APPENDIX 1

```

DEC  ECWord1+1
BEQ  FindAReturn
LDX  #001h
LDA  (ECWord4)      ; use RRHashLink to find
5    TAY              ; next RRHashLink and
    LDA  (ECWord4),X      ; next RRBUFFER offset
    ; BEQ  FindAReturn
    STY  ECWord4+0
    STA  ECWord4+1
10   DecBankSelect
    LDA  (ECWord4)
    EncBankSelect
    CMP  ECFindHash
    BNE  FindAReturn
15   STY  ECWord2+0
    LDA  ECWord4+1
    ADD  #(HIGH(ECRRBuffer)-HIGH(ECRRHashLink))
    STA  ECWord2+1
FindABackMatch:
20   LDX  ECByte3
FindABackLoop:
    LDA  (ECWord2),X      ; check byte at longest
    CMP  (ECByte1),X      ; string + 1 and work
    IF   Test              ; backwards to origin
25   ifNE
        INC  FSSkips+0
        ifEQ
            INC  FSSkips+1
            ifEQ
30         INC  FSSkips+2
            fi
        fi
        !JMP  FindASkip
        fi
35   ELSE
        BNE  FindASkip

```

- 120 -

APPENDIX 1

```

ENDIF
DEX
BNE FindABackLoop
LDA (ECWord2)
5 CMP (ECByte1)
IF Test
BEQ FindABackOK
INC FSSkips+0
ifeq
10 INC FSSkips+1
ifeq
INC FSSkips+2
fi
fi
15 !JMP FindASkip
ELSE
BNE FindASkip
ENDIF
FindABackOK:
20 IF Test ; Word1 = RRHashCount (+0)
INC FSSearches+0 ; Word2 = RRBuffer offset
ifeq ; Word3 = RRBuffer best
string
INC FSSearches+1 ; Word4 = 1st
25 RRHashLink
ifeq ; Byte1 = 1st unmatched
ECChar
INC FSSearches+2 ; Byte3 = best
string length
30 fi
fi
ENDIF
LDX ECByte3
FindAMoreLoop:
35 INX
ifNE

```

- 121 -

APPENDIX 1

```

        LDA (ECWord2),X
        CMP (ECByte1),X
        BEQ FindAMoreLoop
    els
5       LDX #0FFh
        fi
    FindAUpdate:
        LDA ECWord2+0
        STA ECWord3+0      ; Word3 = RRBUFFER offset of
10      LDA ECWord2+1      ;      best string
        STA ECWord3+1
        CPX ECMaxLength
        ifCC
        STX ECByte3        ; Byte3 = best string
15  length
        !JMP      FindASkip
        fi
        LDA ECMaxLength    ; string length at
maximum
20      STA ECByte3
    FindAReturn:
        ENDM
;
;***** A - S T R I N G   S E A R C H *****
25 ;
    SkipAStrings:
        CMP #(MinimumAUpdate+1)
        ifCS
        JMP NoAFound
30      fi
        LDY ECNextOut
        INC ECNextOutSave
        DEC ECAvailable
        JMP WriteAEncodings      ; Y = ECNextOut
35  StringATime:
        LDY ECNextOut

```

- 122 -

APPENDIX 1

```

        STY  ECNextOutSave
        !JMP StringASearch1st
StringASearch:
        LDY  ECNextOut
5  StringASearch1st:
        LDA  ECAvailable
        ifEQ
        LDA  #0FFh
        els
10        CMP  #007h
        BCC  SkipAStrings
        fi
        ADD  #(0-003h)
        STA  ECMaxLength          ; 255 - 3 is MaxLength
15        STI  #0FFh,ECStringOrigin
        STI  #HIGH(ECChar),ECByte2
        IF   Test
        INC  FSEntries+0
        ifEQ
20        INC  FSEntries+1
        ifEQ
        INC  FSEntries+2
        fi
        fi
25        ENDIF
FindA43:
        LDA  ECNextOut
        ADD  #003h
        FindAString
30        LDY  ECByte3          ; Y = string length
        CPY  #004h
        BCC  FindA42
        LDA  ECNextOut
        STA  ECByte1
35        LDX  #002h          ; X = ECOrgin - 1
FindA43Loop:
```

APPENDIX 1

```

    LDA ECWord3+0
    ADD #0FFh
    STA ECWord3+0
    ifCC
5      LDA ECWord3+1
    ADD #0FFh
    CMP #HIGH(ECRRBuffer)
    ifCC
    ADD #HIGH(BufferSize)
10     fi
    STA ECWord3+1
    fi
    LDA (ECWord3)
    CMP (ECByte1),X
15     BNE FindA43Adjust
    INY
    DEX
    BPL FindA43Loop
    !JMP FindA43Done
20 FindA43Adjust:
    INC ECWord3+0
    ifEQ
    INC ECWord3+1
    fi
25 FindA43Done:
    STY ECStringLength
    INX
    STX ECStringOrigin
    LDA ECWord3+0
30     STA ECFound+0
    LDA ECWord3+1
    STA ECFound+1
    SEC
    IF ZoneTestA
35     LDA ECRRPtr+0
    SBC ECWord3+0
```

- 124 -

APPENDIX 1

```

ENDIF
LDA  ECRRPtr+1
SBC  ECWord3+1
AND  #HIGH(BufferSize-1)
5   STA  ECZone
FindA42:
INC  ECMaxLength
LDA  ECNextOut
ADD  #002h
10  FindAString
LDY  ECByte3          ; Y = string length
CPY  #004h
BCC  FindA4Exit
;   TYA          ; gives a little more
15  ;   ADD  #002h          ; compression if string
;   CMP  ECUpdateLength    ; length governs ?????
;   BCC  FindA4Exit
LDA  ECNextOut
STA  ECByte1
20  LDX  #001h          ; X = EOrigin - 1
FindA42Loop:
LDA  ECWord3+0
ADD  #0FFh
STA  ECWord3+0
25  ifCC
LDA  ECWord3+1
ADD  #0FFh
CMP  #HIGH(ECRRBuffer)
ifCC
30  ADD  #HIGH(BufferSize)
fi
STA  ECWord3+1
fi
LDA  (ECWord3)
35  CMP  (ECByte1),X
BNE  FindA42Adjust

```

- 125 -

APPENDIX 1

```

        INY
        DEX
        BPL FindA42Loop
        !JMP FindA42Done
5 FindA42Adjust:
        INC ECWord3+0
        ifEQ
        INC ECWord3+1
        fi
10 FindA42Done:
        INX
        STX ECByte1                ; ECOrgin
        SEC
        IF ZoneTestA
15        LDA ECRRPtr+0
        SBC ECWord3+0
        ENDIF
        LDA ECRRPtr+1
        SBC ECWord3+1
20        AND #HIGH(BufferSize-1)
        TAX
        LDA ECStringOrigin
        ifPL ;{
        CPY ECStringLength
25        BCC FindA4Exit
        ifEQ      ;{
        CPX      ECZone
        BCS      FindA4Exit
        fi ;}
30        fi ;}
        STY ECStringLength
        STX ECZone
        LDA ECByte1
        STA ECStringOrigin
35        LDA ECWord3+0
        STA ECFound+0
```

- 126 -

APPENDIX 1

```

        LDA  ECWord3+1
        STA  ECFound+1
FindA4Exit:
        LDA  ECStringOrigin
5         BMI  NoAFound
        LDA  ECStringLength
        CMP  #MinimumAString
        BCC  NoAFound
StringAOverlap:
10        LDX  ECRRPtr+1
        LDA  ECStringOrigin
        ADD  ECRRPtr+0
        ifCS
            INX
15        fi
        SEC
        SBC  ECFound+0
        STA  ECWord1+0      ; Word1+0 = LOW(Diff)
        TAW
20        TXA
        SBC  ECFound+1
        AND  #HIGH(BufferSize-1)
        STA  ECWord1+1
        ifEQ ;{              ; Delta(ECWord1) < 256
25        LDA ECStringOrigin
        ifNE      ;{
            ADD    ECStringLength
            STA    ECByte4
            TWA                    ; W = ECWord1+0 (saved
30 later)
            ifNE      ;{
                CMP    ECByte4      ; UL =
StringLength+StringOrigin
                BCC    NoAFound
35        fi ;}
        fi ;}

```


- 127 -

APPENDIX 1

```

        fi    ;}
        JMP   ProcessAString
NoAFound:
        IF    AHashX2 XOR 1
5         LDA  ECNextOut
          ADD  #MinimumAUpdate
          STA  ECNextOut
          JMP  ResetACharCounts
        ELSE
10        LDA  #MinimumAUpdate
NoAFoundHashX2:
          STA  ECByte4
          JSR  HashAX2          ; A = -1 or -2
          ADD  ECByte4
15        BMI  NoAFoundHashX2Negative
          BNE  NoAFoundHashX2
NoAFoundHashX2Negative:
          JMP  ResetACharCounts
HashAX2:
20        LDY  ECNextOut          ; Y = index to reach
          INY                      ; ECNextOut+1 data items
          LDA  ECHashX21,Y
          BMI  HashAX2Null
          LDX  ECNextOut
25        STA  ECWord2+1
          ORA  #080h
          CMP  ECHashX21,X
          ifEQ
            LDA  ECHashX20,Y
30          CMP  ECHashX20,X
            BEQ  HashAX2Null
          els
            LDA  ECHashX20,Y
          fi
35        STA  ECWord2+0
HashAX2Bits:

```

- 128 -

APPENDIX 1

```

LDY ECNewIndex,X
ifNE
    INY
    LDA    FontBits,Y
5      ADD    #00Ch          ; 2(11 length) + 10
    els
    LDA    #00Dh          ; 3(001 length) + 10
    fi
    STI #002h,ECByte3
10     SEC
HashAX2SumBits:
    LDY ECType,X
    CPY #002h
    ifEQ    ;{          ; Type 2
15      LDY    ECFontIndex,X
    SBC    FontBits,Y
    els ;{}          ; Type 4
    LDY    ECNewIndex,X
    ifNE    ;{
20      SBC    FontBits,Y
    LDY    ECFontIndex,X
    SBC    GlobalBits,Y
    els    ;{}
    LDY    ECFontIndex,X
25      SBC    GlobalBits,Y
    TAW
    LDA    GlobalCodeHigh,Y
    ifPL    ;{
    TWA
30      SBC    #001h
    els    ;{}
    TWA
    fi      ;}
    fi    ;}
35      fi    ;}
    BMI HashAX2OK

```

- 129 -

APPENDIX 1

```

        INX
        DEC ECByte3
        BNE HashAX2SumBits
HashAX2Null:
5          INC ECNextOut
          LDA #(0-001h)
          RTS
HashAX2OK:
          LDX ECNextOut
10         LDY ECNextOut
          INY
          IF Test
          INC      AHashX2s+0
          ifEQ
15         INC      AHashX2s+1
          fi
          ENDIF
          LDA #006h                ; Type 6
          STA ECType,X
20         LDA ECWord2+1
          ASR A
          ROR ECWord2+0
          AND #003h
          CLC
25         ROR A
          ROR A
          ROR A
          ORA #020h
          STA ECHashX21,Y          ; ECHashX21 of 2nd
30 character =
          LDA ECWord2+0            ; 2 high-order hash bits
          STA ECHashX20,Y
          LDA ECNextOut            ; ECHashX20 of 2nd character =
          ADD #002h                ; 8 low-order hash bits
35         STA ECNextOut
          LDA #(0-002h)

```

- 130 -

APPENDIX 1

```
        RTS
    ENDIF
ProcessAString:
    IF    AHashX2 XOR 1
5        LDA ECNextOut
        ADD ECStringOrigin
        STA ECNextOut
    ELSE
        LDA ECStringOrigin
10        !JMP    ProcessABestX1st
ProcessABestXLoop:
        ADD ECStringOrigin
        STA ECStringOrigin
ProcessABestX1st:
15        CMP #002h
        BCC ProcessABestX
        JSR HashAX2          ; A = -1 OR -2
        !JMP    ProcessABestXLoop
ProcessABestX:
20        AND #0FFh
        ifNE
            INC    ECNextOut
        fi
    ENDIF
25        IF    Test
            INC AStringsFound+0
            ifEQ
                INC    AStringsFound+1
            fi
30        ENDIF
DirectAString:
        LDY ECNextOut
        LDA ECStringLength
        STA ECByte3          ; Byte3 = StringLength
35        LDX ECNewIndex,Y
        ifNE
```

- 131 -

APPENDIX 1

```

        IF AHashX2
            ADD      #(0-(MinimumAString-4))
        ELSE
            ADD      #(0-MinimumAString)
5      ENDIF
        STA ECByte4          ; Byte4 = Global length
        index

        INX
        LDA FontBits,X
10     els
        ADD #(0-MinimumAString)
        STA ECByte4          ; Byte4 = Global length
        index

        IF AHashX2
15         LDA      #003h          ; 011
        ELSE
            LDA      #002h          ; 01
        ENDIF
        fi
20     CLC
        LDX ECByte4
        ADC GlobalBits,X
        LDX ECWord1+1
        ADC ZoneBits,X
25     ADD #008h          ; A = total string
        encoding bits
        SEC
        DirectASumBits:
        LDX ECType,Y
30     CPX #002h
        ifEQ ;{          ; Type 2
            LDX ECFontIndex,Y
            SBC FontBits,X
        els ;{}          ; Type 4
35     LDX ECNewIndex,Y
        ifNE ;{

```

- 132 -

APPENDIX 1

```

        SBC      FontBits,X
        LDX      ECFontIndex,Y
        SBC      GlobalBits,X
    els ;{}
5      LDX      ECFontIndex,Y
        SBC      GlobalBits,X
        TAW
        LDA      GlobalCodeHigh,X
        ifPL     ;{
10      TWA
        SBC      #001h
        els     ;{}
        TWA
        fi ;}
15      fi ;}
        fi ;}
        BMI     DirectAUse
        INY
        DEC     ECByte3
20      BNE     DirectASumBits
    DirectAReject:
        LDA     ECNextOut
        ADD     #MinimumAUpdate
        STA     ECNextOut      ; try HashX2'S ?????
25      JMP     ResetACharCounts
    DirectAUse:
        LDY     ECNextOut
        IF      Test
            INC  AStringsUsed+0
30      ifEQ
            INC  AStringsUsed+1
            fi
        ENDIF
        LDA     #008h          ; Type 8
35      STA     ECType,Y
        LDA     ECByte4        ; Global or LengthB index

```

- 133 -

APPENDIX 1

```

        STA  ECHashX20,Y          ; saved for ECWrite
        INY
        LDA  ECWord1+0            ; save Zone codes for ECWrite
        STA  ECHashX20,Y          ; in 2nd character's
5  ECHashX2
        LDA  ECWord1+1
        STA  ECHashX21,Y
        LDA  ECStringLength
        ADD  ECNextOut
10      STA  ECNextOut
ResetACharCounts:
        LDA  ECAvailable
        ADD  ECNextOutSave        ; update ECAvailable
        SEC
15      SBC  ECNextOut
        STA  ECAvailable
        LDY  ECNextOutSave        ; interchange ECNextOut
and
        LDA  ECNextOut            ;          ECNextOutSave
20      STA  ECNextOutSave
        STY  ECNextOut            ; Y = ECNextOut
;
;***** A - S T R I N G   O U T P U T *****
;
25 ;   ENCODER WRITE ROUTINE
;
WriteAEncodings:
                                ; Y = ECNextOut
        LDA  ECType,Y
        ORA  ECRepeatSW,Y        ; bit 3 on if repeats
30      TAX
        JMP  (WriteAJumps),X
WriteAJumps:
        IF   AHashX2
            DW  WriteANull        ; 0 - HashX2(2) - no
35 repeats
        ELSE
                                ;          or repeats

```

- 134 -

APPENDIX 1

```

        DW 0                ; 0 - HashX2 inactive
    ENDIF
    DW  WriteA0Font          ; 2 - Font char - no
repeats
5      DW  WriteA0NewChar    ; 4 - New char - no
repeats
        IF  AHashX2
            DW  WriteAHashX2        ; 6 - HashX2(1) - no
repeats
10     ELSE
        DW 0                ; 6 - HashX2 inactive
    ENDIF
    DW  WriteAString        ; 8 - String(1) - no
repeats
15     IF  Repeats          ; or repeats
        DW  WriteA1Font        ; 10 - Font char -
repeats
        DW  WriteA1NewChar    ; 12 - New char -
repeats
20     IF  AHashX2
        DW  WriteAHashX2        ; 14 - HashX2(1) -
repeats
        ELSE
            DW 0                ; 14 - HashX2 inactive
25     ENDIF
    ENDIF
;
    IF  AHashX2
WriteANull:
30     IF  Repeats
        LDA  ECRepeats,Y
        ifNE
            JMP  WriteA0Repeats
        els
35     JMP  UpdateA0Buffer
    fi

```


- 135 -

APPENDIX 1

```

        ELSE
        JMP      UpdateA0Buffer
        ENDIF
    ENDIF

5  WriteA0Font:
        LDX  ECFontIndex,Y      ; char encoding index -
font
        LDA  FontCode,X
        Write17
10      JMP  UpdateA0Buffer
        IF   Repeats
        WriteA1Font:
            LDX  ECFontIndex,Y      ; char encoding index -
font
15      LDA  FontCode,X
            Write17
            JMP  WriteA0Repeats
        ENDIF
        WriteA0NewChar:
20      LDX  ECNewIndex,Y          ; NC encoding index
            BEQ  WriteA0NCNF
            LDA  FontCode,X
            Write17
        WriteA0NCOF:
25      LDX  ECFontIndex,Y          ; char encoding index -
global
            LDA  GlobalCodeHigh,X
            TAW
            LDA  GlobalCodeLow,X
30      BEQ  WriteA0NCOFHigh
            TAX
            TWA
            Write817
            !JMP WriteA0Command
35  WriteA0NCOFHigh:
            TWA

```

- 136 -

APPENDIX 1

```

Write17
    JMP WriteA0Command
WriteA0NCNF:
    LDX ECFontIndex,Y      ; char encoding index -
5  global
    LDA GlobalCodeHigh,X
    BMI WriteA0NCNFHigh2
    LDA #040h              ; leading 0 bit
    Write17
10   LDA GlobalCodeHigh,X
    TAW
    LDA GlobalCodeLow,X
    BEQ WriteA0NCNFHigh1
    TAX
15   TWA
    Write817
    !JMP WriteA0Command
WriteA0NCNFHigh1:
    TWA
20  WriteA0NCNFHigh2:
    Write17
    WriteA0Command:
    LDA ECFontIndex,Y      ; char encoding index -
    global
25   CMP #0FEh
    ifCC
        JMP UpdateA0Buffer
    fi
    LDA #040h
30   Write17
    JMP UpdateA0Buffer
    IF Repeats
WriteA1NewChar:
    LDX ECNewIndex,Y      ; NC encoding index
35   BEQ WriteA1NCNF
    LDA FontCode,X

```

- 137 -

APPENDIX 1

```

Write17
WriteA1NCOF:
        LDX ECFontIndex,Y           ; char encoding index -
global
5        LDA GlobalCodeHigh,X
        TAW
        LDA GlobalCodeLow,X
        BEQ WriteA1NCOFHigh
        TAX
10       TWA
        Write817
        !JMP      WriteA1Command

WriteA1NCOFHigh:
        TWA
15       Write17
        JMP WriteA1Command

WriteA1NCNF:
        LDX ECFontIndex,Y           ; char encoding index -
global
20       LDA GlobalCodeHigh,X
        BMI WriteA1NCNFFHigh2
        LDA #040h                   ; leading 0 bit
        Write17
        LDA GlobalCodeHigh,X
25       TAW
        LDA GlobalCodeLow,X
        BEQ WriteA1NCNFFHigh1
        TAX
        TWA
30       Write817
        !JMP      WriteA1Command

WriteA1NCNFFHigh1:
        TWA
WriteA1NCNFFHigh2:
35       Write17
WriteA1Command:

```

- 138 -

APPENDIX 1

```

                                LDA ECFontIndex,Y      ; char encoding index -
global
                                CMP #0FEh
                                ifCC
5                                JMP      WriteA0Repeats
                                fi
                                LDA #040h
                                Write17
                                JMP WriteA0Repeats
10                               ENDIF
                                IF      AHashX2
WriteAHashX2:
                                LDX ECNewIndex,Y
                                BEQ WriteAHNF
15 WriteAHOF:
                                INX
                                LDA FontCode,X
                                Write17
                                LDA #0E0h              ; 11 length
20                               Write17
                                !JMP      WriteAHMain
WriteAHNF:
                                LDA #050h              ; 010
                                Write17
25 WriteAHMain:
                                INY
                                LDA ECHashX21,Y
                                Write17
                                LDA ECHashX20,Y
30                               Write8
                                LDA #000h
                                STA ECType,Y          ; Type 0 (2nd byte)
                                STA ECRpeatSW,Y
                                DEY
35                               IF Repeats
                                LDA      ECRpeats,Y
```

- 139 -

APPENDIX 1

```

        ifNE
        JMP    WriteA0Repeats
    els
        JMP    UpdateA0Buffer
5      fi
    ELSE
        JMP    UpdateA0Buffer
    ENDIF
ENDIF
10 WriteAString:
        LDX    ECNewIndex,Y
        BEQ    WriteASNF
    WriteASOF:
        INX
15      LDA    FontCode,X
        Write17
        !JMP WriteASLength
    WriteASNF:
        IF     AHashX2
20      LDA    #070h                ; 011
        ELSE
            LDA    #060h                ; 01
        ENDIF
        Write17
25 WriteASLength:
        LDX    ECHashX20,Y
        LDA    GlobalCodeHigh,X
        TAW
        LDA    GlobalCodeLow,X
30      BEQ    WriteASLHigh
        TAX
        TWA
        Write817
        !JMP WriteASMain
35 WriteASLHigh:
        TWA

```

- 140 -

APPENDIX 1

```

                                Write17
WriteASMain:
                                INY
                                LDX  ECHashX21,Y
5                                LDA  ZoneCode,X
                                Write17
                                LDA  ECHashX20,Y
                                Write8
                                DEY
10                               IF   Repeats
                                LDA  ECREpeats,Y
                                ifNE
                                JMP    WriteA1Repeats
                                els
15                               JMP    UpdateA1Buffer
                                fi
                                ELSE
                                JMP UpdateA1Buffer
                                ENDIF
20 ;
;*****  A - S T R I N G   B U F F E R   U P D A T E   *****
;
                                IF   Repeats
WriteA0Repeats:
25                               LDA  ECREpeats,Y
                                CMP  #001h
                                BNE  WriteA0AreRepeats
WriteA0NoRepeats:
                                LDA  #040h
30                               Write17
                                !JMP  WriteA0RClear
WriteA0AreRepeats:
                                LDA  #0C0h
                                Write17
35                               LDA  ECREpeats,Y
                                ADD  #0FEh
```

- 141 -

APPENDIX 1

```

        TAX
        LDA GlobalCodeHigh,X
        TAW
        LDA GlobalCodeLow,X
5       BEQ WriteA0RHigh
        TAX
        TWA
        Write817
        !JMP      WriteA0RClear
10      WriteA0RHigh:
        TWA
        Write17
        WriteA0RClear:
        LDA #000h
15      STA ECRRepeats,Y
        STA ECRRepeatsSW,Y
        ENDIF
        ;
        UpdateA0Buffer:                                ; Y = ECNextOut
20      LDA ECChar,Y
        STA (ECRRPtr)
        IF BufferSuffix
        LDX ECRRPtr+1
        CPX #HIGH(ECRRBuffer)
25      ifEQ
        STI
        # (HIGH(ECRRBuffer)+HIGH(BufferSize)),ECRRPtr+1
        STA      (ECRRPtr)
        STI      #HIGH(ECRRBuffer),ECRRPtr+1
30      fi
        ENDIF
        BBS 0,ECRRPtr+0,UpdateA0Head
        JMP UpdateA0BufferPtr
        UpdateA0Head:
35      LDX #001h
        LDA ECRRPtr+0      ; Word3 = ptr to RRHashLink

```

- 142 -

APPENDIX 1

```

ADD #(0-003h)      ; at location RRPTr-3
STA ECWord3+0
LDA ECRRPtr+1
ifCC
5   ADD #0FFh
    CMP #HIGH(ECRRBuffer)
    ifCC
      LDA      #(HIGH(ECRRBuffer)+HIGH(BufferSize)-1)
    fi
10  fi
    ADD #(HIGH(ECRRHashLink)-HIGH(ECRRBuffer))
    STA ECWord3+1
    LDA ECHashRaw0,Y      ; Word4 = ptr to
    EOR ECPriorHash0      ;      RRHashHead
15  STA ECWord4+0
    DecBankSelect
    STA (ECWord3)          ; store LOW(Hash) in
    EncBankSelect          ; RRHashTest table
    LDA ECHashRaw1,Y
20  EOR ECPriorHash1
    AND #HIGH(BufferHashes-1)
    ASL ECWord4+0
    ROL A
    ADD #HIGH(ECRRHashHead)
25  STA ECWord4+1
    LDA ECHashRaw0,Y
    NEG A
    STA ECPriorHash0
    LDA ECHashRaw1,Y
30  NEG A
    STA ECPriorHash1
UpdateA0Link:
    LDA (ECWord4)          ; transfer RRHashHead to
    STA (ECWord3)          ; RRHashLink table
35  LDA (ECWord4),X
    STA (ECWord3),X

```


- 143 -

APPENDIX 1

```

        LDA  ECWord3+0      ; reset RRHashHead to new
        STA  (ECWord4)      ; RRHashLink ptr
        LDA  ECWord3+1
        STA  (ECWord4),X
5  UpdateA0BufferPtr:
        INC  ECRRPtr+0
        ifEQ
            INC  ECRRPtr+1
            LDA  ECRRPtr+1
10       CMP  #(HIGH(ECRRBuffer)+HIGH(BufferSize))
        ifEQ
            STI   #HIGH(ECRRBuffer),ECRRPtr+1
        fi
        fi
15       IF   FailSafe
            DEC  ECFailSafe+0
            ifEQ
                DEC    ECFailSafe+1
            ifEQ
20         STI   #FailSafeSets,ECFailSafe+1
            LDA   #008h
            Writel7
            fi
            fi
25       ENDIF
;
        OutputA0Control:
            INY
            STY  ECNextOut
30         CPY  ECNextOutSave
            ifNE
                JMP WriteAEncodings          ; Y = ECNextOut
            fi
            LDA  ECFlush
35         BNE  OutputA0Flush
            LDA  #SetLength

```

- 144 -

APPENDIX 1

```

        CMP  ECAvailable
        ifCS
        RTS
        fi
5      JMP  StringASearch
OutputA0Flush:
        LDA  ECAvailable
        ifEQ
        RTS
10     fi
        JMP  StringASearch
;
        IF  Repeats
WriteA1Repeats:
15     CMP  #001h
        BNE WriteA1AreRepeats
WriteA1NoRepeats:
        LDA  #040h
        Write17
20     !JMP  WriteA1RClear
WriteA1AreRepeats:
        LDA  #0C0h
        Write17
        LDA  ECrepeats,Y
25     ADD  #0FEh
        TAX
        LDA  GlobalCodeHigh,X
        TAW
        LDA  GlobalCodeLow,X
30     BEQ  WriteA1RHigh
        TAX
        TWA
        Write817
        !JMP  WriteA1RClear
35 WriteA1RHigh:
        TWA
```

- 145 -

APPENDIX 1

```

Write17
WriteA1RClear:
    LDA #000h
    STA ECRepeats,Y
5    STA ECRepeatsSW,Y
    ENDIF

;
UpdateA1Buffer:                                ; Y = ECNextOut
    LDA ECChar,Y
10    STA (ECRRPtr)
    IF BufferSuffix
        LDX ECRRPtr+1
        CPX #HIGH(ECRRBuffer)
        ifEQ
15        STI
        # (HIGH(ECRRBuffer)+HIGH(BufferSize)),ECRRPtr+1
        STA (ECRRPtr)
        STI #HIGH(ECRRBuffer),ECRRPtr+1
        fi
20    ENDIF
    BBS 0,ECRRPtr+0,UpdateA1Head
    JMP UpdateA1BufferPtr

UpdateA1Head:
    LDX #001h
25    LDA ECRRPtr+0      ; Word3 = ptr to RRHashLink
    ADD #(0-003h)      ; at location RPtr-3
    STA ECWord3+0
    LDA ECRRPtr+1
    ifCC
30    ADD #0FFh
    CMP #HIGH(ECRRBuffer)
    ifCC
        LDA # (HIGH(ECRRBuffer)+HIGH(BufferSize)-1)
        fi
35    fi
    ADD # (HIGH(ECRRHashLink)-HIGH(ECRRBuffer))

```

- 146 -

APPENDIX 1

```

        STA  ECWord3+1
        LDA  ECHashRaw0,Y          ; Word4 = ptr to
        EOR  ECPriorHash0          ;      RRHashHead
        STA  ECWord4+0
5      DecBankSelect
        STA  (ECWord3)             ; store LOW(Hash) in
        EncBankSelect             ; RRHashTest table
        LDA  ECHashRaw1,Y
        EOR  ECPriorHash1
10     AND  #HIGH(BufferHashes-1)
        ASL  ECWord4+0
        ROL  A
        ADD  #HIGH(ECRRHashHead)
        STA  ECWord4+1
15     LDA  ECHashRaw0,Y
        NEG  A
        STA  ECPriorHash0
        LDA  ECHashRaw1,Y
        NEG  A
20     STA  ECPriorHash1
      UpdateAllLink:
        LDA  (ECWord4)             ; transfer RRHashHead to
        STA  (ECWord3)             ; RRHashLink table
        LDA  (ECWord4),X
25     STA  (ECWord3),X
        LDA  ECWord3+0             ; reset RRHashHead to new
        STA  (ECWord4)             ; RRHashLink ptr
        LDA  ECWord3+1
        STA  (ECWord4),X
30     UpdateAllBufferPtr:
        INC  ECRRPtr+0
        ifEQ
        INC  ECRRPtr+1
        LDA  ECRRPtr+1
35     CMP  #(HIGH(ECRRBuffer)+HIGH(BufferSize))
        ifEQ

```

- 147 -

APPENDIX 1

```

        STI      #HIGH(ECRRBuffer),ECRRPtr+1
    fi
fi
IF    FailSafe
5      DEC ECFailSafe+0
    ifEQ
        DEC      ECFailSafe+1
    ifEQ
        STI      #FailSafeSets,ECFailSafe+1
10      LDA      #008h
        Write17
    fi
    fi
ENDIF
15 ;
    OutputA1Control:
        INY
        STY  ECNextOut
        CPY  ECNextOutSave
20      ifNE
            LDA ECrepeats,Y
            ifEQ
                JMP      UpdateA1Buffer
            fi
25      JMP WriteA1Repeats      ; Y = ECNextOut
    fi
        LDA  ECFlush
        BNE  OutputA1Flush
        LDA  #SetLength
30      CMP  ECAvailable
        ifCS
            RTS
        fi
        JMP  StringASearch
35 OutputA1Flush:
        LDA  ECAvailable

```

- 148 -

APPENDIX 1

```

        ifEQ
        RTS
    fi
    JMP StringASearch
5 ;
;***** B - S T R I N G   M A C R O S   *****
;
;   B-STRING SEARCH MACRO - BYTE 2
;
10 FindB2String  MACRO
        STI #002h,ECByte3
        INX                               ; Word3 = ptr to
        LDA ECHashRaw0,X                 ;   FTHashHead
        STA ECFindHash
15        STA ECWord3+0
        LDA ECHashRaw1,X                 ; Word2 = ptr to 1st
        AND #HIGH(BufferHashes-1)       ; (RRBuffer+1)
        location
        ASL ECWord3+0                   ; for this hash
20        ROL A
        ADD #HIGH(ECRRHashHead)
        STA ECWord3+1                   ; Word4 = ptr to 1st
        LDX #001h                       ; (FTHashLink+1)
        location
25        LDA (ECWord3),X                ; for this hash
        ifNE
        STA ECWord4+1
        LDA (ECWord3)
        STA ECWord4+0
30        TAY
        STI #MaximumBSearches,ECWord1+1
        !JMP FindB2Skip1st
        fi
        !JMP FindB2Return
35 FindB2Skip:
        DEC ECWord1+1

```

- 149 -

APPENDIX 1

```

        BEQ  FindB2Return
        LDX  #001h
        LDA  (ECWord4)      ; use FTHashLink to find
        TAY          ; next FTHashLink and
5         LDA  (ECWord4),X      ; next RRBUFFER offset
        BEQ  FindB2Return
        STY  ECWord4+0
        STA  ECWord4+1
        DecBankSelect
10        LDA  (ECWord4)
        EncBankSelect
        CMP  ECFindHash
        BNE  FindB2Return
FindB2Skip1st:
15        LDA  ECWord4+1
        ADD  #(HIGH(ECRRBuffer)-HIGH(ECRRHashLink))
        STA  ECWord2+1
        TYA          ; Y = ECWord4+0
        ADD  #0FFh
20        STA  ECWord2+0
        ifCC
        LDA  ECWord2+1
        ADD  #0FFh
        CMP  #HIGH(ECRRBuffer)
25        ifCC
        ADD  #HIGH(BufferSize)
        fi
        STA  ECWord2+1
        fi
30        LDA  (ECWord2)
        CMP  (ECByte1)
        IF  Test
        BEQ  FindB2More
        INC  FSSkips+0
35        ifEQ
        INC  FSSkips+1

```

- 150 -

APPENDIX 1

```

        ifEQ
        INC    FSSkips+2
        fi
        fi
5      !JMP    FindB2Skip
      ELSE
        BNE FindB2Skip
      ENDIF
FindB2More:
10      IF    Test                ; Word1 = emergency max hashes
        INC  FSSearches+0        ; Word2 = RRBUFFER ptr
        ifEQ                                ; Word3 = RRBUFFER best
string
        INC    FSSearches+1        ; Word4 = FTHashLink
15 ptr
        ifEQ                                ; Byte1 = 1st unmatched
ECChar
        INC    FSSearches+2        ; Byte3 = best
string length
20      fi
        fi
      ENDIF
      LDX    #000h
FindB2MoreLoop:
25      INX
        ifNE
          LDA (ECWord2),X
          CMP (ECByte1),X
          BEQ FindB2MoreLoop
30      els
          LDX #0FFh
        fi
FindB2Update:
        CPX  ECByte3
35      BCC  FindB2Skip            ; reset ECChar offset
used

```


- 151 -

APPENDIX 1

```

        BEQ  FindB2Skip          ; in FindnnStart routine
        LDA  ECWord2+0
        STA  ECWord3+0          ; Word3 = RRBUFFER offset of
        LDA  ECWord2+1          ; best string
5       STA  ECWord3+1
        CPX  ECMaxLength
        ifcc
            STX ECByte3          ; Byte3 = best string
length
10      !JMP  FindB2Skip
        fi
        LDA  ECMaxLength          ; string length at
maximum
        STA  ECByte3
15 FindB2Return:
        ENDM

;
; * * * * *
;
20 ; B-STRING SEARCH MACRO - BYTE 1
;
FindB1String MACRO
        STI  #002h,ECByte3
        INX                      ; Word3 = ptr to
25      LDA  ECHashRaw0,X          ; FTHashHead
        STA  ECFindHash
        STA  ECWord3+0
        LDA  ECHashRaw1,X          ; Word2 = ptr to 1st
        AND  #HIGH(BufferHashes-1) ; (RRBuffer+1)
30 location
        ASL  ECWord3+0          ; for this hash
        ROL  A
        ADD  #HIGH(ECRRHashHead)
        STA  ECWord3+1          ; Word4 = ptr to 1st
35      LDX  #001h              ; (FTHashLink+1)
location

```

- 152 -

APPENDIX 1

```

    LDA (ECWord3),X          ; for this hash
    ifNE
        STA ECWord4+1
        LDA (ECWord3)
5      STA ECWord4+0
        TAY
        STI #MaximumBSearches,ECWord1+1
        !JMP FindB1Skip1st
    fi
10     !JMP FindB1Return
FindB1Skip:
        DEC ECWord1+1
        BEQ FindB1Return
        LDX #001h
15     LDA (ECWord4)          ; use FTHashLink to find
        TAY                  ; next FTHashLink and
        LDA (ECWord4),X       ; next RRBUFFER offset
        BEQ FindB1Return
        STY ECWord4+0
20     STA ECWord4+1
        DecBankSelect
        LDA (ECWord4)
        EncBankSelect
        CMP ECFindHash
25     BNE FindB1Return
FindB1Skip1st:
        STY ECWord2+0          ; Y = ECWord4+0
        LDA ECWord4+1
        ADD #(HIGH(ECRRBuffer)-HIGH(ECRRHashLink))
30     STA ECWord2+1
        LDX #002h
        LDA (ECWord2),X
        CMP (ECByte1),X
        IF Test
35     BEQ FindB1More
        INC FSSkips+0

```

- 153 -

APPENDIX 1

```

        ifEQ
        INC      FSSkips+1
        ifEQ
        INC      FSSkips+2
5         fi
        fi
        !JMP      FindB1Skip
        ELSE
        BNE FindB1Skip
10        ENDIF
        FindB1More:
                IF      Test                ; Word1 = emergency max hashes
                INC      FSSearches+0        ; Word2 = RRBuffer ptr
                ifEQ                ; Word3 = RRBuffer best
15 string
                INC      FSSearches+1        ; Word4 = FTHashLink
                ptr
                ifEQ                ; Byte1 = 1st unmatched
                ECChar
20                INC      FSSearches+2        ; Byte3 = best
                string length
                fi
                fi
                ENDIF
25                LDX      #000h
                !JMP      FindB1More1st
        FindB1MoreLoop:
                INX
                ifNE
30 FindB1More1st:
                LDA      (ECWord2),X
                CMP      (ECByte1),X
                BEQ      FindB1MoreLoop
                els
35                LDX      #0FFh
                fi

```

- 154 -

APPENDIX 1

```

FindB1Update:
    CPX  ECByte3
    BCC  FindB1Skip          ; reset ECChar offset
used
5      BEQ  FindB1Skip          ; in FindnnStart routine
    LDA  ECWord2+0
    STA  ECWord3+0          ; Word3 = RRBUFFER offset of
    LDA  ECWord2+1          ; best string
    STA  ECWord3+1
10     CPX  ECMaxLength
    ifCC
    STX  ECByte3          ; Byte3 = best string
length
    !JMP  FindB1Skip
15     fi
    LDA  ECMaxLength          ; string length at
maximum
    STA  ECByte3
FindB1Return:
20     ENDM
;
;***** B - S T R I N G   S E A R C H *****
;
SkipBStrings:
25     CMP  #(MinimumBUpdate+1)
    ifCS
    JMP  NoBFound
    fi
    LDY  ECNextOut
30     INC  ECNextOutSave
    DEC  ECAvailable
    IF  AntiEx
    JMP  TotalBBits
    ELSE
35     JMP  UpdateBBuffer      ; Y = ECNextOut
    ENDIF

```

- 155 -

APPENDIX 1

```

StringBTime:
    LDY  ECNextOut
    STY  ECNextOutSave
    IF   AntiEx
5       STY  ECNextOutStart
        STI  #000h,ECExcessBits
    ENDIF
    !JMP StringBSearch1st
StringBSearch:
10      LDY  ECNextOut
StringBSearch1st:
        LDA  ECAvailable
        ifEQ
        LDA  #0FFh
15      els
        CMP  #003h
        BCC  SkipBStrings
        fi
        STA  ECMaxLength          ; 255 is MaxLength
20      STY  ECByte1
        STI  #HIGH(ECChar),ECByte2
        STI  #000h,ECStringLength
        IF   Test
        INC  FSEntries+0
25      ifEQ
        INC  FSEntries+1
        ifEQ
        INC  FSEntries+2
        fi
30      fi
    ENDIF
FindB2:
        LDX  ECNextOut
        INX
35      FindB2String
        LDX  ECByte3

```

- 156 -

APPENDIX 1

```
CPX  #(MinimumBString+1)
BCC  FindB1
SEC
IF   ZoneTestB
5    LDA  ECRRPtr+0
    SBC  ECWord3+0
ENDIF
LDA  ECRRPtr+1
SBC  ECWord3+1
10   AND  #HIGH(BufferSize-1)
STA  ECZone
STX  ECStringLength
LDA  ECWord3+0
STA  ECFound+0
15   LDA  ECWord3+1
STA  ECFound+1
LDA  ECWord3+0

FindB1:
LDX  ECNextOut
20   FindB1String
LDX  ECByte3
CPX  #(MinimumBString+1)
BCC  FindB1Exit
CPX  ECStringLength
25   BCC  StringBOverlap
    ifEQ
    SEC
    IF   ZoneTestB
        LDA  ECRRPtr+0
        SBC  ECWord3+0
30   ENDIF
    LDA  ECRRPtr+1
    SBC  ECWord3+1
    AND  #HIGH(BufferSize-1)
35   CMP  ECZone
    BCS  StringBOverlap
```

- 157 -

APPENDIX 1

```

        fi
        STX  ECStringLength
        LDA  ECWord3+0
        STA  ECFound+0
5         LDA  ECWord3+1
        STA  ECFound+1
        !JMP StringBOverlap
FindB1Exit:
        LDA  ECStringLength
10        BEQ  NoBFound
StringBOverlap:
        LDA  ECRRPtr+0
        SEC
        SBC  ECFound+0
15        STA  ECWord1+0      ; Word1+0 = LOW(Diff)
        LDA  ECRRPtr+1
        SBC  ECFound+1
        AND  #HIGH(BufferSize-1)
        STA  ECWord1+1
20        JMP  ProcessBString
NoBFound:
        JSR  HashBX2
        JMP  ResetBCharCounts
HashBX2:
25        LDY  ECNextOut      ; Y = index to reach
        INY                      ; ECNextOut+1 data items
        LDA  ECHashX21,Y
        BMI  HashBX2Null
        LDX  ECNextOut
30        STA  ECWord2+1
        ORA  #080h
        CMP  ECHashX21,X
        ifEQ
        LDA  ECHashX20,Y
35        CMP  ECHashX20,X
        BEQ  HashBX2Null

```

- 158 -

APPENDIX 1

```

    els
        LDA ECHashX20,Y
    fi
    STA ECWord2+0
5 HashBX2Bits:
    LDY ECNewIndex,X          ; NC encoding index
    ifNE
        INY
        LDA FontBits,Y        ; ST bits + Hash(=10)
10    ADD #00Ah
    els
        LDA #00Dh              ; 3(110 length) + 10
    fi
    IF AntiEx
15    TAW
    ENDIF
    STI #002h,ECByte3
    SEC
HashBX2SumBits:
20    LDY ECType,X
    CPY #002h
    ifEQ ;{                    ; Type 2
        LDY ECFontIndex,X
        SBC FontBits,Y
25    els ;{}                  ; Type 4
        LDY ECNewIndex,X      ; NC encoding index
        ifNE ;{
            SBC FontBits,Y
            SBC #008h
30    els ;{}
        LDY ECFrequency,X
        ifPL ;{
            SBC #008h
        els ;{}
35    SBC #009h
    fi ;}

```


- 159 -

APPENDIX 1

```

        fi    ;}
        fi    ;}
        BMI   HashBX2OK
        INX
5         DEC   ECByte3
        BNE   HashBX2SumBits
HashBX2Null:
        INC   ECNextOut
        RTS
10      HashBX2OK:
        LDX   ECNextOut
        LDY   ECNextOut
        INY
        IF    AntiEx                ; total bits for 2-byte
15      hash
        TWA                ; less cost of 8-bit
        frequency
        STA   ECHashX21,X        ; save bit length for
        AntiEx
20      STI   #002h,ECStringLength ; in 2nd character
        position
        ENDIF
        IF    Test
        INC   BHashX2s+0
25      ifEQ
        INC   BHashX2s+1
        fi
        ENDIF
        LDA   #006h                ; Type 6
30      STA   ECType,X
        IF    AntiEx XOR 1
        LDA   #000h                ; Type 0 (skip)
        STA   ECType,Y
        ENDIF
35      LDA   ECWord2+1
        ASR   A

```

- 160 -

APPENDIX 1

```

ROR  ECWord2+0
AND  #003h
CLC
ROR  A
5   ROR  A
    ROR  A
    ORA  #020h
    STA  ECHashX21,Y          ; ECHashX21 of 2nd
character =
10   LDA  ECWord2+0          ; 2 high-order hash bits
    STA  ECHashX20,Y
    LDA  ECNextOut          ; ECHashX20 of 2nd character =
    ADD  #002h              ; 8 low-order hash bits
    STA  ECNextOut
15   RTS
ProcessBString:
    IF  Test
    INC BStringsFound+0
    ifEQ
20   INC  BStringsFound+1
    fi
    ENDIF
DirectBString:
    LDY  ECNextOut
25   LDA  ECStringLength
    STA  ECByte3            ; Byte3 = StringLength
    ADD  #(0-(MinimumBString+1))
    STA  ECByte4            ; Byte4 = LengthB index
    CMP  #009h
30   ifCC
    TAX
    LDA  LengthBBits,X      ; length bits from table
    els
    ADD  #(0-009h)
35   TAX
    LDA  GlobalBits,X

```

- 161 -

APPENDIX 1

```

        ADD #004h
        fi
        LDX ECNewIndex,Y
        ifNE
5         INX
          INX
          CLC
          ADC FontBits,X
        els
10        ADD #003h
          fi
          LDX ECWord1+1
          CLC
          ADC ZoneBits,X
15        ADD #008h                ; A = total string
        encoding bits
          IF AntiEx
            TAW
          ENDIF
20        SEC
        DirectBSumBits:
          LDX ECType,Y
          CPX #002h
          ifEQ                ; Type 2
25          LDX ECFontIndex,Y
            SBC FontBits,X
          els                ; Type 4
            LDX ECNewIndex,Y        ; NC encoding index
            ifNE                ;{
30              SBC FontBits,X
                SBC #008h
            els ;{}
              LDX ECFrequency,Y
              ifPL                ;{
35                SBC #008h
                  els                ;{}

```

- 162 -

APPENDIX 1

```

        SBC    #009h
        fi ;}
        fi ;}
        fi
5       BMI    DirectBUse
        INY
        DEC    ECByte3
        BNE    DirectBSumBits
DirectBReject:
10      JSR    HashBX2
        JMP    ResetBCharCounts
DirectBUse:
        LDY    ECNextOut
        IF     AntiEx
15      TWA
        STA    ECHashX21,Y          ; save bit length for
AntiEx
        ENDIF
        IF     Test
20      INC    BStringsUsed+0
        ifEQ
        INC    BStringsUsed+1
        fi
        ENDIF
25      LDA    #008h                ; Type 8
        STA    ECType,Y
        LDA    ECByte4              ; Global or LengthB index
        STA    ECHashX20,Y          ; saved for ECWrite
        INY
30      LDA    ECWord1+0            ; save Zone codes for ECWrite
        STA    ECHashX20,Y          ; in 2nd character's
ECHashX2
        LDA    ECWord1+1
        STA    ECHashX21,Y
35      IF     AntiEx XOR 1
        LDX    ECStringLength

```

- 163 -

APPENDIX 1

```

        DEX
        LDA #000h
UseBStringLoop:
        STA ECType,Y      ; set Type to 0 in the
5         INY              ; (ECStringLength-1) chars
        DEX              ; which generate no output
        BNE UseBStringLoop
        ENDIF
        LDA ECStringLength
10        ADD ECNextOut
        STA ECNextOut
ResetBCharCounts:
        LDA ECAvailable
        ADD ECNextOutSave ; update ECAvailable
15        SEC
        SBC ECNextOut
        STA ECAvailable
        LDY ECNextOutSave ; interchange ECNextOut
and
20        LDA ECNextOut      ;          ECNextOutSave
        STA ECNextOutSave
        STY ECNextOut      ; Y = ECNextOut
;
;***** B - S T R I N G   O U T P U T *****
25 ;
;   TOTAL B BITS FOR AntiExpansion
;
        IF   AntiEx
TotalBBits:
; Y = ECNextOut
30        LDA ECExcessBits
        ADD #(0-008h)
        LDX ECType,Y
        JMP (TotalBJumps),X
TotalBJumps:
35        DW TotalBDone
        DW TotalBFont

```

- 164 -

APPENDIX 1

```

        DW TotalBNewChar
        DW TotalBHashX2
        DW TotalBString

TotalBFont:
5          CLC
          LDX ECFontIndex,Y      ; char encoding index -
font
          ADC FontBits,X
          !JMP      TotalBDone

10 TotalBNewChar:
          CLC
          LDX ECNewIndex,Y      ; NC encoding index
          BEQ TotalBNCMainNF
          ADC FontBits,X

15 TotalBNCMainOF:
          ADD #008h              ; char encoding index -
8-bit
          !JMP      TotalBDone

TotalBNCMainNF:
20          LDX ECFrequency,Y    ; char encoding index -
8-bit
          ifPL
          ADD      #008h
          els
25          ADD      #009h
          fi
          !JMP      TotalBDone

TotalBHashX2:
TotalBString:
30          CLC
          ADC ECHashX21,Y
          IF AntiEx
          STA      ECExcessBits
          JMP      UpdateB1Buffer
35          ENDIF

TotalBDone:

```

- 165 -

APPENDIX 1

```

        AND #0FFh
        ifPL      ;{
            CMP      #040h
            ifCS      ;{
5              LDA      #040h
            fi ;}
        els ;{}
            CMP      #(0-040h)
            ifCC      ;{
10             LDA      #(0-040h)
            fi ;}
        fi ;}
        STA ECExcessBits
    ELSE
15         JMP WriteBEncodings
    ENDIF

;
;*****  B - S T R I N G   B U F F E R   U P D A T E   *****
;
20 ;   UPDATE BUFFER
;
    UpdateBBuffer:
                                           ; Y = ECNextOut
        LDA  ECChar,Y
        STA  (ECRRPtr)
25         IF  BufferSuffix
            LDX ECRRPtr+1
            CPX #HIGH(ECRRBuffer)
            ifEQ
                STI
30         #(HIGH(ECRRBuffer)+HIGH(BufferSize)),ECRRPtr+1
                STA      (ECRRPtr)
                STI      #HIGH(ECRRBuffer),ECRRPtr+1
            fi
        ENDIF
35         BBS  0,ECRRPtr+0,UpdateBHead
        JMP  UpdateBBufferPtr

```

- 166 -

APPENDIX 1

UpdateBHead:

```

LDX #001h
LDA ECRRPtr+0      ; Word3 = ptr to RRHashLink
ADD #(0-001h)      ; at location RRPTr-1
5 STA ECWord3+0
LDA ECRRPtr+1
ADD #(HIGH(ECRRHashLink)-HIGH(ECRRBuffer))
STA ECWord3+1
LDA ECHashRaw0,Y   ; Word4 = ptr to
10 STA ECWord4+0    ; RRHashHead
DecBankSelect
STA (ECWord3)      ; store LOW(Hash) in
EncBankSelect      ; RRHashTest table
LDA ECHashRaw1,Y
15 AND #HIGH(BufferHashes-1)
ASL ECWord4+0
ROL A
ADD #HIGH(ECRRHashHead)
STA ECWord4+1

```

20 UpdateBLink:

```

LDA (ECWord4)      ; transfer RRHashHead to
STA (ECWord3)      ; RRHashLink table
LDA (ECWord4),X
STA (ECWord3),X
25 LDA ECWord3+0    ; reset RRHashHead to new
STA (ECWord4)      ; RRHashLink ptr
LDA ECWord3+1
STA (ECWord4),X

```

UpdateBBufferPtr:

```

30 INC ECRRPtr+0
   ifEQ
   INC ECRRPtr+1
   LDA ECRRPtr+1
   CMP #(HIGH(ECRRBuffer)+HIGH(BufferSize))
35 ifEQ
   STI #HIGH(ECRRBuffer),ECRRPtr+1

```


- 167 -

APPENDIX 1

```

        fi
        fi
    ;
OutputBControl:
5          INY
          STY  ECNextOut
          CPY  ECNextOutSave
          ifNE                      ; Y = ECNextOut
          IF  AntiEX
10          JMP      TotalBBits
          ELSE
          JMP      WriteBEncodings
          ENDIF
          fi
15          IF  AntiEx XOR 1      ;{
          LDA  ECFlush
          BNE  ECOutputBFlush
          LDA  #SetLength
          CMP  ECAvailable
20          ifCS
          RTS
          fi
          JMP  StringBSearch
OutputBFlush:
25          LDA  ECAvailable
          ifEQ
          RTS
          fi
          JMP  StringBSearch
30          ELSE      ;{}
          LDA  ECAntiEStatus      ; saved at start of
StringTime
          BMI  OutputBSTOff
OutputBSTOn:
35          LDY  ECExcessBits
          IF  Macros

```

- 168 -

APPENDIX 1

```

        ifMI
        JMP      OutputBCurrent      ; if minus, write
current
        fi
5        ELSE
        BMI      OutputBCurrent      ; if minus, write
current
        ENDIF
        CPY #014h
10       IF Macros
        ifCC
        JMP      OutputBDefer        ; if a bit plus,
defer writing
        fi
15       ELSE
        BCC      OutputBDefer        ; if a bit plus,
defer writing
        ENDIF
        ORA #080h
20       STA ECAntiEStatus            ; if too plus, turn off
        LDX ECNextOutStart
        LDY ECNewIndex,X             ; NC encoding index
        ifNE      ;{
        LDA      FontCode,Y
25       Write17
        LDA      #0FEh                ; 0FEh = 11111110
        LDX      #0C0h                ; + 1
        Write817
        els ;{}
30       LDA      #0BFh                ; 0FEh = (10)111111
        LDX      #060h                ; + 01
        Write817
        fi ;}
        IF Test ;{
35       INC      AntiExOn+0
        ifEQ      ;{

```

- 169 -

APPENDIX 1

```

        INC      AntiExOn+1
        fI ;}
    ENDIF      ;}
    !JMP      OutputBCurrent

5  OutputBSTOff:
        LDY      ECEXCESSBits
        BPL      OutputBDefer      ; if losing, no change
        CPY      #(0-013h)      ; should be 9 ?????
        BCS      OutputBDefer      ; if a bit minus, no
10  change
        AND      #07Fh
        STA      ECAntiEStatus      ; if too minus, turn on
        LDA      #0FEh      ; strings and write
        current
15      LDX      #0C0h
        Write817      ; 0FEh, 1 to turn on
        IF      Test ;{
            INC      AntiExOff+0
            ifEQ      ;{
20      INC      AntiExOff+1
            fI ;}
        ENDIF      ;}

    OutputBCurrent:
        JSR      OutputBWrite
25      LDA      ECFlush
        ifEQ      ;{
            RTS
        els ;{}
        LDA      ECAvailable
30      ifEQ      ;{
            RTS
            fi ;}
        fi ;}
        JMP      StringBSearch

35  OutputBDefer:
        LDA      ECAvailable

```

- 170 -

APPENDIX 1

```

        BEQ OutputBWrite
        LDY ECFlush
        ifEQ      ;{
            CMP      #SetLength
5          BCC      OutputBWrite
        fi      ;}
        JMP StringBSearch
OutputBWrite:
        LDA ECNextOut
10       TAX
        SEC
        SBC ECNextOutStart
        STA ECBytel
        LDY ECNextOutStart
15       STX ECNextOutStart
        STI #000h,ECExcessBits
        ENDIF      ;}

;
WriteBEncodings:                                ; Y = ECNextOut
20       IF      AntiEx
            LDA ECAntiEStatus
            BPL WriteBStringsOn
WriteB8Bit:
        LDA ECFrequency,Y                        ; character frequency
25       Write8
        LDA ECFrequency,Y
        CMP #0FEh
        ifCC
            JMP      WriteB0TestRepeats
30       fi
        LDA #040h
        Write17
        JMP WriteB0TestRepeats
WriteBStringsOn:                                ; Y = ECNextOut
35       LDA ECType,Y
        ORA ECRepeatSW,Y                        ; bit 3 on if repeats

```

- 171 -

APPENDIX 1

```

        TAX
        ELSE
        LDX ECType,Y
        ENDIF
5      JMP  (WriteBJumps),X
      WriteBJumps:
      repeats
        DW  WriteBNull
        repeats)
10     DW  WriteB0Font
        repeats
        DW  WriteB0NewChar
        repeats
        DW  WriteBHashX2
15    repeats
        DW  WriteBString
        repeats
        IF  AntiEx
        IF  Repeats
20     DW  WriteB1Font
        repeats
        DW  WriteB1NewChar
        repeats
        DW  WriteBHashX2
25    repeats
        ENDIF
        ENDIF
      ;
      WriteBNull:
30     IF  Repeats
        JMP WriteB0TestRepeats
        ELSE
        JMP WriteB0Done
        ENDIF
35    WriteB0Font:
        LDX ECFontIndex,Y
        ; char encoding index -

```

- 172 -

APPENDIX 1

```

font
    LDA  FontCode,X
    Write17
    IF  AntiEx
5      JMP WriteB0Done
    ELSE
        IF  Repeats
            JMP  WriteB0TestRepeats
        ELSE
10      JMP  WriteB0Done
        ENDIF
    ENDIF
    IF  AntiEx
        IF  Repeats
15 WriteB1Font:
            LDX  ECFontIndex,Y      ; char encoding
            index - font
            LDA  FontCode,X
            Write17
20      LDA  ECrepeats,Y
            JMP  WriteB0Repeats
        ENDIF
    ENDIF
    WriteB0NewChar:
25      LDX  ECNewIndex,Y          ; NC encoding index
            BEQ  WriteB0NCMainNF
            LDA  FontCode,X
            Write17
    WriteB0NCMainOF:
30      LDA  ECFrequency,Y          ; char encoding index -
            8-bit
            Write8
            !JMP WriteB0Command
    WriteB0NCMainNF:
35      LDA  ECFrequency,Y          ; char encoding index -
            8-bit

```

- 173 -

APPENDIX 1

```
        ifPL
        Write8
    else
        STI #080h,ECByte4
5        ASR A
        ROR ECByte4
        AND #0BFh
        LDX ECByte4
        Write817
10        fi
        WriteB0Command:
            LDA ECFrequency,Y          ; character frequency
            CMP #0FEh
            ifCC
15        WriteB0CommandX:
            IF AntiEx
                JMP WriteB0Done
            ELSE
                IF Repeats
20                JMP WriteB0TestRepeats
            ELSE
                JMP WriteB0Done
            ENDIF
        ENDIF
25        fi
        LDA #040h
        Write17
        BRA WriteB0CommandX
        IF AntiEx
30        IF Repeats
            WriteB1NewChar:
                LDX ECNewIndex,Y          ; NC encoding index
                BEQ WriteB1NCMainNF
                LDA FontCode,X
35        Write17
        WriteB1NCMainOF:
```

- 174 -

APPENDIX 1

```

        LDA      ECFrequency,Y          ; char encoding
index - 8-bit
        Write8
        !JMP     WriteB1Command
5 WriteB1NCMMainNF:
        LDA      ECFrequency,Y          ; char encoding
index - 8-bit
        ifPL
        Write8
10      els
        STI      #080h,ECByte4
        ASR      A
        ROR      ECByte4
        AND      #0BFh
15      LDX      ECByte4
        Write817
        fi
WriteB1Command:
        LDA      ECFrequency,Y          ; character
20 frequency
        CMP      #0FEh
        ifCC
WriteB1CommandX:
        LDA      ECrepeats,Y
25      JMP      WriteB0Repeats
        fi
        LDA      #040h
        Write17
        BRA      WriteB1CommandX
30      ENDIF
      ENDIF
WriteBHashX2:
        LDX      ECNewIndex,Y          ; NC encoding index
        BEQ      WriteBX2NF
35 WriteBX2OF:
        INX
```


- 175 -

APPENDIX 1

```
        LDA  FontCode,X
        Write17
        !JMP WriteBX2Main
WriteBX2NF:
5         LDA  #0D0h                ; 110
        Write17
WriteBX2Main:
        INY
        LDA  ECHashX21,Y
10        Write17
        LDA  ECHashX20,Y
        Write8
        IF   AntiEx
        LDA  #000h
15        STA  ECType,Y
        IF   Repeats
        STA   ECRepeatsSW,Y
        ENDIF
        ENDIF
20        DEY
        IF   Repeats
        JMP  WriteB0TestRepeats
        ELSE
        JMP  WriteB0Done
25        ENDIF
WriteBSXtraLength:
        LDA  LengthBCode+9
        Write17
        LDA  ECHashX20,Y            ; length index
30        ADD  #(0-009h)
        TAX
        LDA  GlobalCodeHigh,X
        TAW
        LDA  GlobalCodeLow,X
35        BEQ  WriteBSXLHigh
        TAX
```

- 176 -

APPENDIX 1

```

        TWA
        Write817
        JMP WriteBSMain
WriteBSXLHigh:
5         TWA
        Write17
        JMP WriteBSMain
WriteBString:
        LDX ECNewIndex,Y          ; NC encoding index
10        BEQ WriteBSNF
WriteBSOF:
        INX
        INX
        LDA FontCode,X
15        Write17
        !JMP WriteBSLength
WriteBSNF:
        LDA #0F0h                ; 111
        Write17
20 WriteBSLength:
        IF AntiEX
        LDA ECHashX20,Y          ; length index
        TAX
        ADD #(MinimumBString+1)
25        STA ECByte4
        NEG A
        ADD ECByte1
        STA ECByte1
        ELSE
30        LDX ECHashX20,Y          ; length index
        ENDIF
        CPX #009h
        ifCS
        JMP WriteBSXtraLength
35        fi
        LDA LengthBCode,X
```

- 177 -

APPENDIX 1

```

                                Write17
WriteBSMain:
                                INY
                                LDX  ECHashX21,Y
5                                LDA  ZoneCode,X
                                Write17
                                LDA  ECHashX20,Y
                                Write8
                                DEY
10                               IF   AntiEX
                                IF   Repeats
                                LDA    ECrepeats,Y
                                ifEQ
                                JMP     WriteB1Done
15                               els
                                JMP     WriteB1Repeats
                                fi
                                ELSE
                                JMP     WriteB1Done
20                               ENDIF
                                ELSE
                                JMP WriteB0TestRepeats
                                ENDIF
                                ;
25                               IF   Repeats
WriteB0AreRepeats:
                                LDA #0C0h
                                Write17
                                LDA ECrepeats,Y
30                               ADD #0FEh
                                TAX
                                LDA GlobalCodeHigh,X
                                TAW
                                LDA GlobalCodeLow,X
35                               BEQ WriteB0RHigh
                                TAX
```

- 178 -

APPENDIX 1

```

        TWA
        Write817
        JMP WriteB0RClear
WriteB0RHigh:
5         TWA
        Write17
        JMP WriteB0RClear
WriteB0TestRepeats:
        LDA ECrepeats,Y
10        BEQ WriteB0Done
WriteB0Repeats:
        CMP #001h
        BNE WriteB0AreRepeats
WriteB0NoRepeats:
15        LDA #040h
        Write17
WriteB0RClear:
        LDA #000h
        STA ECrepeats,Y
20        STA ECrepeatsSW,Y
        ENDIF
WriteB0Done:
        IF FailSafe
        DEC ECFailSafe+0
25        ifEQ
        DEC      ECFailSafe+1
        ifEQ
        STI      #FailSafeSets,ECFailSafe+1
        LDA      #008h
30        Write17
        fi
        fi
        ENDIF
        IF AntiEX
35        DEC ECBytel
        ifEQ
```

- 179 -

APPENDIX 1

```

        RTS
        fi
        INY
        JMP WriteBEncodings
5      ELSE
        JMP UpdateBBuffer
      ENDIF
    ;

    IF AntiEX
10      IF Repeats
        WriteB1AreRepeats:
            LDA    #0C0h
            Write17
            LDA    ECrepeats,Y
15          ADD    #0FEh
            TAX
            LDA    GlobalCodeHigh,X
            TAW
            LDA    GlobalCodeLow,X
20          BEQ    WriteB1RHigh
            TAX
            TWA
            Write817
            JMP    WriteB1RClear
25 WriteB1RHigh:
            TWA
            Write17
            JMP    WriteB1RClear
        WriteB1Repeats:
30          CMP    #001h
            BNE    WriteB1AreRepeats
        WriteB1NoRepeats:
            LDA    #040h
            Write17
35 WriteB1RClear:
            LDA    #000h
```

- 180 -

APPENDIX 1

```

        STA      ECRpeats,Y
        STA      ECRpeatSW,Y
    ENDIF
WriteB1Done:
5          IF  FailSafe
        DEC      ECFailSafe+0
        ifEQ
        DEC      ECFailSafe+1
        ifEQ
10         STI    #FailSafeSets,ECFailSafe+1
        LDA      #008h
        Writel7
        fi
        fi
15        ENDIF
        DEC      ECByte4
        ifNE
        INY
        IF Repeats
20         LDA      ECRpeats,Y
        BEQ      WriteB1Done
        JMP      WriteB1Repeats
        ELSE
        JMP      WriteB1Done
25        ENDIF
        fi
        LDA      ECByte1
        ifEQ
        RTS
30        fi
        INY
        JMP WriteBEncodings
    ENDIF
;
35        IF  AntiEx
UpdateB1Buffer:
; Y = ECNextOut
```

- 181 -

APPENDIX 1

```

        LDA ECChar,Y
        STA (ECRRPtr)
        IF BufferSuffix
            LDX      ECRRPtr+1
5          CPX      #HIGH(ECRRBuffer)
            ifEQ
            STI
            # (HIGH(ECRRBuffer)+HIGH(BufferSize)),ECRRPtr+1
            STA      (ECRRPtr)
10         STI      #HIGH(ECRRBuffer),ECRRPtr+1
            fi
        ENDIF
        BBS 0,ECRRPtr+0,UpdateB1Head
        JMP UpdateB1BufferPtr
15 UpdateB1Head:
        LDX #001h
        LDA ECRRPtr+0      ; Word3 = ptr to RRHashLink
        ADD #(0-001h)      ; at location RRPTr-1
        STA ECWord3+0
20        LDA ECRRPtr+1
        ADD #(HIGH(ECRRHashLink)-HIGH(ECRRBuffer))
        STA ECWord3+1
        LDA ECHashRaw0,Y      ; Word4 = ptr to
        STA ECWord4+0      ; RRHashHead
25        DecBankSelect
        STA (ECWord3)      ; store LOW(Hash) in
        EncBankSelect      ; RRHashTest table
        LDA ECHashRaw1,Y
        AND #HIGH(BufferHashes-1)
30        ASL ECWord4+0
        ROL A
        ADD #HIGH(ECRRHashHead)
        STA ECWord4+1

UpdateB1Link:
35        LDA (ECWord4)      ; transfer RRHashHead to
        STA (ECWord3)      ; RRHashLink table

```

- 182 -

APPENDIX 1

```

    LDA (ECWord4),X
    STA (ECWord3),X
    LDA ECWord3+0      ; reset RRHashHead to new
    STA (ECWord4)      ; RRHashLink ptr
5    LDA ECWord3+1
    STA (ECWord4),X

UpdateB1BufferPtr:
    INC ECRRPtr+0
    ifEQ
10    INC      ECRRPtr+1
    LDA      ECRRPtr+1
    CMP      #(HIGH(ECRRBuffer)+HIGH(BufferSize))
    ifEQ
    STI      #HIGH(ECRRBuffer),ECRRPtr+1
15    fi
    fi

;
OutputB1Control:
    DEC ECStringLength
20    ifNE
    INY
    LDA      ECExcessBits
    ADD      #(0-008h)
    ifPL     ;{
25    CMP      #040h
    ifCS     ;{
    LDA      #040h
    fi      ;}
    els      ;{}
30    CMP      #(0-040h)
    ifCC     ;{
    LDA      #(0-040h)
    fi      ;}
    fi ;}
35    STA      ECExcessBits
    JMP      UpdateB1Buffer

```


- 183 -

APPENDIX 1

```

        fi
        JMP OutputBControl
    ENDIF
;
5 ;***** D E C O D E R   M A C R O S   *****
;
    DCGlobalShort  MACRO
        IF  Macros
            MSDCGlobalShort
10        ELSE
            JSR MSDCGlobalShort
        ENDIF
    ENDM
;
15 ;* * * * *
;
;   Enter:      A = guard bit at proper shift point (i.e. 80h
for 1 bit)
;   Exit:       A = fetched bits (right justified)
20 ;           Z flag properly set(reset) for [A]
;
    DecodeNBits  MACRO
        LOCAL    DecodeNB1
    DecodeNB1:
25        ASL  DCBuffer
        ifEQ
            JSR DCReadCharacter
            SEC
            ROL DCBuffer
30        fi
            ROL  A
            BCC  DecodeNB1
        ENDM
;
35 ;* * * * *
;

```

- 184 -

APPENDIX 1

```

DCFreqToChar    MACRO
    LOCAL      DCFTToCExit
    STA  DCWord1+0
    CMP  #0FEh
5      ifCS
        LDA  #080h                ; get 1 bit
        DecodeNBits
        ifNE                ; set DCCommand = 1
        STA  DCCommand        ; NOTE: this is the valid
10 EOF
        !JMP  DCFTToCExit
        fi
        fi
        SetCharFreq  DC,W1        ; sets HIGH(NCFreq)
15 in DCWord1+1
        LDA  (DCWord1)
DCFTToCExit:
        ENDM
;
20 ;* * * * *
;
;  Enter:      A = # of font table indices (2-16) - 2
;  Exit:       A = font index (0-15)
;
25 DCRReadFontFreq    MACRO
    LOCAL      DCRNextBit
    TAY
    LDX  DCFontTblIndex,Y
DCRNextBit:
30      ASL  DCBuffer
        ifEQ
            JSR DCRReadCharacter
            SEC
            ROL DCBuffer
35      fi
        ifCS

```

- 185 -

APPENDIX 1

```

        INX
        fi
        LDA  DCFontNext,X
        ifNE
5         TXA
        CLC
        ADC  DCFontNext,X
        TAX
        !JMP      DCRNextBit
10        fi
        LDA  DCFontValue,X      ; A = font index
        ENDM
;
; * * * * *
15 ;
        DCGlobalLong  MACRO
                LDA  #040h      ; get 2 bits
                DecodeNBits     ; DCGlobalShort expects
that
20        DCGlobalShort      ; 1st 2 bits of the
Global
                ENDM          ; code are in A and that
;                          ; the Z flag is based on [A]
; * * * * *
25 ;
        IF  Macros
        MSDCGlobalShort  MACRO
                LOCAL
        DGS00Zones,DGS0001,DGS00001,DGS000001,DGS000000,DGSExit
30        ELSE
        MSDCGlobalShort:
                ENDDIF
                BEQ  DGS00Zones      ; Z flag set for [A]
                CMP  #002h
35        ifCC      ; zone = 01
                LDA  #020h      ; get 3 bits

```

- 186 -

APPENDIX 1

```

        STI #008h,DCByte1      ; DCByte1 = base      (DL)
    els
        ifNE                    ; zone = 11
        STI #000h,DCByte1
5      els                      ; zone = 10
        STI #004h,DCByte1
        fi
        LDA #040h              ; get 2 bits
        fi
10     DecodeNBits
        ADD DCByte1
        !JMP DGSExit

DGS00Zones:
        LDA #010h              ; get 4 bits
15     DecodeNBits
        CMP #008h
        BCC DGS0001
        AND #007h
        ADD #010h              ; base 16
20     !JMP DGSExit

DGS0001:
        CMP #004h
        BCC DGS00001
        ORA #080h              ; append 1 bit
25     DecodeNBits
        AND #007h
        ADD #018h              ; base 24
        !JMP DGSExit

DGS00001:
30     CMP #001h
        BCC DGS000000
        BEQ DGS000001
        ORA #010h              ; append 4 bits
        DecodeNBits
35     AND #01Fh
        ADD #020h              ; base 32

```

- 187 -

APPENDIX 1

```

                                !JMP DGSExit
DGS000001:
                                LDA  #004h                ; get 6 bits
                                DecodeNBits
5                                ADD  #040h                ; base 64
                                !JMP DGSExit
DGS000000:
                                LDA  #002h                ; get 7 bits
                                DecodeNBits
10                               ADD  #080h                ; base 128
DGSExit:
                                IF    Macros
                                ENDM
                                ELSE
15                               RTS
                                ENDIF

;
; * * * * *
;

20 DCLengthLong  MACRO
                                LOCAL  DCLNextBit,DCLExit
                                LDX  #000h
DCLNextBit:
                                ASL  DCBuffer
25                                ifEQ
                                JSR  DCReadCharacter
                                SEC
                                ROL  DCBuffer
                                fi
30                                ifCS
                                INX
                                fi
                                LDA  LengthBNext,X
                                ifNE
35                                TXA
                                CLC

```

- 188 -

APPENDIX 1

```

        ADC LengthBNext,X
        TAX
        !JMP      DCLNextBit
    fi
5      LDA  LengthBValue,X
        CMP  #009h
        ifNE
            JMP DCLExit
        fi
10     STA  DCByte2
        DCGlobalLong
        ADD  DCByte2
DCLExit:
        ENDM
15  ;
    ;*****
    ;
DCZoneLong      MACRO
        LOCAL      DCZNextBit
20      LDX  #000h
DCZNextBit:
        ASL  DCBuffer
        ifEQ
            JSR DCReadCharacter
25      SEC
        ROL  DCBuffer
        fi
        ifCS
            INX
30      fi
        LDA  ZoneNext,X
        ifNE
            TXA
            CLC
35      ADC  ZoneNext,X
            TAX

```

- 189 -

APPENDIX 1

```

                !JMP      DCZNextBit
                fi
                LDA      ZoneValue,X
                STA      DCWord1+1
5               ENDM

;
;***** D E C O D E R   R E F I L L *****
;
DCFontParams:
10              LDY      DCABStatus
                BPL      DCFontsActive
                LDA      #001h                ; get 8 bits
                DecodeNBits
                JMP      DCNewAChar
15 DCFontsActive:
                LDA      DCCurrentHash+1
                BPL      DCOldFont
                TYA                ; Y = DCABStatus
                ifEQ
20              JMP      DCNewAFont
                fi
                JMP      DCNewBFont
DCOldFont:
                LDA      DCCharacters
25              ADD      DCABStatus
                ; ADD      DCSTIndex        ; always 1 if strings on
                ; ADD      #0FFh            ; A is # of font indices
                (0-16)
                DCReadFontFreq        ; [A] is returned as
30 index
                LDY      DCABStatus
                ifNE
                CMP      DCNCIndex
                ifEQ
35              LDA      #001h                ; get 8 bits
                DecodeNBits

```

- 190 -

APPENDIX 1

```

        JMP      DCNewAChar
    fi
    BCC DCReadOldChar
    ADD #0FFh
5     CMP DCNCIndex
    ifEQ
        JMP      DC2ByteHash
    fi
    ADD #0FFh
10    CMP DCNCIndex
    ifEQ
        JMP      DCReadBString
    fi
    ADD #0FFh
15    els
        CMP DCNCIndex
        BCC DCReadOldChar
        ifEQ
            JMP      DCNewACharLong
20        fi
        ADD #0FFh
        CMP DCNCIndex
        ifEQ
            JMP      DCReadAString
25        fi
        ADD #0FFh
    fi
DCReadOldChar:
    STA DCFontIndex          ; used in FontUpdate
30    ADD #(TwoBytes+1)
    ADD DCFontBase+0
    STA DCWord1+0
    LDA DCFontBase+1
    STA DCWord1+1
35    LDA (DCWord1)
    STA (DCRRPtr)

```


- 191 -

APPENDIX 1

```

        STI    #001h,DCCharCount
        JMP    DCRResetFont

DCNewAFont:
        LDA    #040h                ; get 2 bits
5         DecodeNBits
        CMP    #001h
        ifCC                ; 00
        LDA    #080h                ; get 1 bit
        DecodeNBits
10        !JMP    DCNewACharShort
        fi
        BNE    DCNewACharShort        ; 10,11
        IF     AHashX2 XOR 1        ; 01
        JMP    DCReadAString
15        ELSE
        LDA    #080h                ; get 1 bit
        DecodeNBits
        ifEQ
        JMP    DC2ByteHash
20        fi
        DCGlobalLong
        ADD    #MinimumAString
        JMP    DCDirectString
        ENDIF
25 DCNewACharShort:
        AND    #0FFh                ; reset Z flag for [A]
        DCGlobalShort
        JMP    DCNewAChar

DCNewACharLong:
30        DCGlobalLong

DCNewAChar:
        DCFreqToChar                ; [A] is input
        LDY    DCCCommand
        ifEQ
35        STA    (DCRRPtr)            ; character is output
        STI    #001h,DCCharCount

```

- 192 -

APPENDIX 1

```

        fi
        JMP  DCRResetFont
DCNewBFont:
        LDA  #040h                ; get 2 bits
5       DecodeNBits
        CMP  #002h
        ifCC                ; 00,01
            ORA  #004h                ; append 6 bits
            DecodeNBits
10      JMP  DCNewAChar
        els
            ifEQ                ; 10
            LDA  #003h                ; append 7 bits to 1
            DecodeNBits
15      JMP  DCNewAChar
        fi
        fi                ; 11
        LDA  #080h                ; get 1 bit
        DecodeNBits
20      ifEQ
            JMP  DC2ByteHash
        fi
DCReadBString:
        DCLengthLong
25      ADD  #(MinimumBString+1)
        JMP  DCDirectString
DCReadAString:
        IF  AHashX2
            LDA  #040h                ; get 2 bits
30      DecodeNBits
            CMP  #003h
            ifEQ
                JMP  DC2ByteHash
            fi
35      AND  #0FFh                ; reset Z flag for [A]
        DCGlobalShort

```

- 193 -

APPENDIX 1

```

        ADD #(MinimumAString-4)
    ELSE
        DCGlobalLong
        ADD #MinimumAString
5      ENDIF
    DCDirectString:
        STA DCCharCount
        DCZoneLong          ; DCWord1 = offset from
RRPtr
10      LDA #001h          ; get 8 bits
        DecodeNBits
        STA DCByte1
        LDA DCRRPtr+0
        STA DCWord2+0
15      SEC
        SBC DCByte1
        STA DCWord3+0      ; DCWord3 = source string
        offset
        TAX                ; X = save DCWord3+0 for LAN
20      LDA DCRRPtr+1
        STA DCWord2+1      ; DCWord2 = object string
        offset
        SBC DCWord1+1
        CMP #HIGH(DCRRBuffer)
25      ifCC
        ADD #HIGH(BufferSize) ; A = save DCWord3+1 for
LAN
        fi
        LDY DCWord1+1      ; DCWord1+1: 0 - Right to
30 Left
        BEQ DCDirectBackward ; <> 0 - Left to
Right
    DCDirectForward:
        LDY DCCharCount
35 DCDirectForward1:
        CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize)-1)

```

- 194 -

APPENDIX 1

```

        BEQ   DCDirectForward2
DCDirectForwardOK:
        PHA
        PHX
5         PLI
DCDirectFLoop1:
        LAN
        STA   (DCWord2)
        INC   DCWord2+0
10        ifEQ
            LDA DCWord2+1
            ADD #001h
            CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize))
            ifEQ
15            ADD      #(0-HIGH(BufferSize))
            fi
            STA DCWord2+1
            fi
            DEY
20        BNE   DCDirectFLoop1
        JMP   DCResetFont
DCDirectForward2:
        TAW
        TYA                      ; A = Y = DCCharCount
25        ADD   #(0-001h)
        ADD   DCWord3+0
        TWA
        BCC   DCDirectForwardOK
        STA   DCWord3+1
30 DCDirectFLoop2:
        LDA   (DCWord3)
        STA   (DCWord2)
        INC   DCWord3+0
        ifEQ
35        LDA   DCWord3+1
        ADD   #001h
```

- 195 -

APPENDIX 1

```

        CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize))
        ifEQ
            ADD      #(0-HIGH(BufferSize))
        fi
5       STA DCWord3+1
        fi
        INC  DCWord2+0
        ifEQ
            LDA DCWord2+1
10      ADD #001h
        CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize))
        ifEQ
            ADD      #(0-HIGH(BufferSize))
        fi
15     STA DCWord2+1
        fi
        DEY
        BNE  DCDirectFLoop2
        JMP  DCResetFont
20    DCDirectBackward:
        LDY  DCCharCount
        CPY  DCByte1
        BCC  DCDirectForward1
        BEQ  DCDirectForward1
25     STA  DCWord3+1
        DEY
        TYA
        ADD  DCWord3+0
        STA  DCWord3+0
30     ifCS
        LDA  DCWord3+1
        ADD  #001h
        CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize))
        ifEQ
35     ADD      #(0-HIGH(BufferSize))
        fi

```

- 196 -

APPENDIX 1

```

        STA DCWord3+1
        fi
        TYA
        ADD DCWord2+0
5       STA DCWord2+0
        ifCS
        LDA DCWord2+1
        ADD #001h
        CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize))
10      ifEQ
        ADD      #(0-HIGH(BufferSize))
        fi
        STA DCWord2+1
        fi
15      INY
        DCDirectBLoop:
        LDA (DCWord3)
        STA (DCWord2)
        LDA DCWord3+0
20      ADD #0FFh
        STA DCWord3+0
        ifCC
        LDA DCWord3+1
        CMP #HIGH(DCRRBuffer)
25      ifEQ
        ADD      #HIGH(BufferSize)
        fi
        ADD #0FFh
        STA DCWord3+1
30      fi
        LDA DCWord2+0
        ADD #0FFh
        STA DCWord2+0
        ifCC
35      LDA DCWord2+1
        CMP #HIGH(DCRRBuffer)
```

- 197 -

APPENDIX 1

```

        ifEQ
        ADD    #HIGH(BufferSize)
        fi
        ADD #0FFh
5         STA DCWord2+1
        fi
        DEY
        BNE   DCDirectBLoop
        JMP   DCResetFont
10  DC2ByteHash:
        LDA   #020h                ; get 3 bits
        DecodeNBits
        ADD   #HIGH(DCFTHashChars)
        STA   DCWord1+1
15         LDA   #002h                ; get 7 bits
        DecodeNBits
        ASL   A
        STA   DCWord1+0
        LDA   DCRRPtr+0            ; DCWord2 = object string
20  offset
        STA   DCWord2+0
        LDA   DCRRPtr+1
        STA   DCWord2+1
        LDA   (DCWord1)
25         STA   (DCWord2)
        INC   DCWord2+0
        ifEQ
        LDA   DCWord2+1
        ADD   #001h
30         CMP   #(HIGH(DCRRBuffer)+HIGH(BufferSize))
        ifEQ
        ADD    #(0-HIGH(BufferSize))
        fi
        STA   DCWord2+1
35         fi
        LDX   #001h

```

- 198 -

APPENDIX 1

```

        LDA  (DCWord1),X
        STA  (DCWord2)
        STI  #002h,DCCharCount
        JMP  DCRResetFont

5  DCProdCommand:
        LDA  #080h                ; get 1 bit
        DecodeNBits
        BNE  DCIsCommand          ; 0 = prod, 1 = command

    DCIsProd:
10      STI  #080h,DCBuffer        ; Prod resets DCBuffer,
        clears
        STI  #000h,DCCCommand     ; DCCCommand and returns
        to
        JMP  DCFontParams         ; DCFontParams

15  DCIsCommand:
        LDA  #040h                ; get next 2 bits and
        store as
        DecodeNBits              ; DCCCommand (right
        justified)
20      STA  DCCCommand            ; ProcessCommand does as named
        STI  #080h,DCBuffer        ; and then JMP's back to
        IF   EOFControl            ; DCFontParams
        STI  #000h,DCCCommand
        JMP  DCoreCEEOF

25      ELSE
        JMP  ProcessCommand
        ENDIF

    DCRResetFont:
        LDA  DCCCommand            ; prod/command encountered
30      ifNE
        IF   AntiEX
        LDA  DCWord1+0
        CMP  #0FFh
        ifEQ
35      JMP  DCProdCommand
        fi

```


- 199 -

APPENDIX 1

```

        LDA    DCABStatus
        ifPL
            ORA    #080h
        els
5         AND    #07Fh
        fi
        STA    DCABStatus
        STI    #000h,DCCommand
        JMP    DCFontParams
10      ELSE
        JMP    DCProdCommand
        ENDIF
    fi
    LDA    (DCRRPtr)
15      STA    DCCurrentChar
    JSR    DCWriteCharacter
    IF    Repeats
        CMP    DCChar2Prior
        ifNE
20          JMP    DCUpdateFont
        fi
        CMP    DCChar1Prior
        ifNE
25          JMP    DCUpdateFont
        fi
        LDA    #080h                ; get 1 bit
        DecodeNBits
        ifEQ
30          JMP    DCUpdateFont
        fi
        DCGlobalLong
        TAY
        INY
        LDA    DCCurrentChar
35  DCWriteRepeatsLoop:
        JSR    DCWriteCharacter

```

- 200 -

APPENDIX 1

```

        DEY
        BNE DCWriteRepeatsLoop
    ENDIF
DCUpdateFont:
5      FontUpdate      DC,FU
      IF  FailSafe
        DEC DCFailSafe+0
        ifEQ
          DEC      DCFailSafe+1
10      ifEQ
          STI      #FailSafeSets,DCFailSafe+1
          LDA      #010h                ; get 4 bits
          DecodeNBits
          ifNE
15      IF      EOFControl
FailSafeTrap:
          STI #0FFh,ECCCommand
          BBR 6,HostLCR,FailSafeNLB
          JSR ECReadCharacter
20      STI #000h,ECCCommand
          JMP DCoreCEOF
FailSafeNLB:
          STI #0FFh,ECCCommand
          JSR DCReadCharacter
25      STI #000h,ECCCommand
          JMP DCoreCEOF
          ELSE
          JMP FailSafeFailed
        ENDIF
30      fi
FailSafeOK:
      fi
      fi
    ENDIF
35      INC  DCRRPtr+0
      ifEQ
```

- 201 -

APPENDIX 1

```

        LDA DCRRPtr+1
        ADD #001h
        CMP #(HIGH(DCRRBuffer)+HIGH(BufferSize))
        ifEQ
5          ADD      #(0-HIGH(BufferSize))
        fi
        STA DCRRPtr+1
        fi
        DEC  DCCharCount
10        ifEQ
          JMP DCFontParams
        fi
        JMP DCResetFont
;
15        printstat Code,size,is,$$-cb
;
;***** INCLUDE TABLES *****
;
        tb          equ $
20 ;
          include      TCTab011
;***** ENCODER TABLES *****
;
        EncodingTable:
25 ;
          plantl      macro          q,r
          q&r:
              endm
;
30          IF FontSize EQ 8
              sepn0          defl 0
              irp            y,<e0,e1,e2,e3,e4,e5,e6,e7,e8>
              db              y-FontCode
              endm
35          ENDIF
;

```

- 202 -

APPENDIX 1

```

IF FontSize EQ 16
    sepno      defl 0
    irp
y,<e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,e12,e13,e14>
5          db      y-FontCode
          endm
ENDIF

;

etbase     macro
10          plantl      e,%sepno
          sepno      defl sepno+1
          endm

;
FontCode:
15          etbase                      ; 2
          db 11000000b,01000000b
          etbase                      ; 3
          db 11000000b,00100000b,01100000b
          etbase                      ; 4
20          db 11000000b,00100000b,01010000b,01110000b
          etbase                      ; 5
          db
          11000000b,00100000b,01010000b,01101000b,01111000b
          etbase                      ; 6
25          db
          10100000b,11100000b,00010000b,00110000b,01010000b
          db 01110000b
          etbase                      ; 7
          db
30          10100000b,11100000b,00010000b,00110000b,01010000b
          db 01101000b,01111000b
          etbase                      ; 8
          db
          10100000b,11100000b,00010000b,00110000b,01001000b
35          db 01011000b,01101000b,01111000b
          etbase                      ; 9

```

- 203 -

APPENDIX 1

```

        db
10100000b,11100000b,00010000b,00110000b,01001000b
        db 01011000b,01101000b,01110100b,01111100b
        etbase ; 10
5        db
10100000b,11100000b,00010000b,00110000b,01001000b
        db
01011000b,01100100b,01101100b,01110100b,01111100b
        IF FontSize EQ 16
10        etbase ; 11
        db
10100000b,11100000b,00010000b,00110000b,01001000b
        db
01011000b,01100100b,01101100b,01110100b,01111010b
15        db 01111110b
        etbase ; 12
        db
10100000b,11100000b,00010000b,00110000b,01001000b
        db
20 01011000b,01100100b,01101100b,01110010b,01110110b
        db 01111010b,01111110b
        etbase ; 13 ERP
        db
10100000b,11100000b,00010000b,00101000b,00111000b
25        db
01001000b,01011000b,01100100b,01101100b,01110010b
        db 01110110b,01111010b,01111110b
        ; etbase ; 13 FLB
        ; db
30 10100000b,11100000b,00010000b,00101000b,00111000b
        ; db
01001000b,01010100b,01011100b,01100100b,01101100b
        ; db 01110100b,01111010b,01111110b
        etbase ; 14
35        db
10100000b,11100000b,00010000b,00101000b,00111000b

```

- 204 -

APPENDIX 1

```

        db
01001000b,01010100b,01011100b,01100100b,01101100b
        db 01110010b,01110110b,01111010b,01111110b
        etbase ; 15
5        db
10100000b,11100000b,00010000b,00101000b,00111000b
        db
01000100b,01001100b,01010100b,01011100b,01100100b
        db
10 01101100b,01110010b,01110110b,01111010b,01111110b
        etbase ; 16
        db
10100000b,11100000b,00010000b,00101000b,00111000b
        db
15 01000100b,01001100b,01010100b,01011100b,01100100b
        db
01101010b,01101110b,01110010b,01110110b,01111010b
        db 01111110b
        ENDIF
20 ;
        fontesz equ $-FontCode
        ;

;***** D E C O D E R   T A B L E S *****
25 ;
DCFontTblIndex:
        DB 000,002,006,012
        DB 020,030,042,056
        DB 072
30 IF FontSize EQ 16
        DB 090,110,132
        DB 156,182,210
        ENDIF
        ;
35 DCFontNext:
        DB 0, 0 ; 2

```

- 205 -

APPENDIX 1

	DB	2, 0, 0, 0	; 3
	DB	2, 0, 0, 1, 0, 0	; 4
	DB	2, 0, 0, 1, 0, 1, 0, 0	; 5
	DB	4, 1, 0, 0, 2, 3, 0, 0	; 6
5	DB	0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 0	; 7
	DB	0, 1, 0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 0	; 8
	DB	2, 3, 0, 0, 0, 0	
10	DB	4, 1, 0, 0, 2, 3, 0, 0	; 9
	DB	2, 3, 0, 0, 0, 1, 0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 0	; 10
	DB	2, 3, 0, 0, 2, 3, 0, 0	
	DB	0, 0	
15	IF	FontSize EQ 16	
	DB	4, 1, 0, 0, 2, 3, 0, 0	; 11
	DB	2, 3, 0, 0, 2, 3, 0, 0	
	DB	0, 1, 0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 0	; 12
20	DB	2, 3, 0, 0, 2, 3, 0, 0	
	DB	2, 3, 0, 0, 0, 0	
	DB	4, 1, 0, 0, 2, 5, 0, 1	; 13 ERP
	DB	0, 0, 2, 3, 0, 0, 2, 3	
	DB	0, 0, 2, 3, 0, 0, 0, 0	
25 ;	DB	4, 1, 0, 0, 2, 3, 0, 0	; 13 FLB
;	DB	4, 5, 0, 0, 0, 3, 4, 5	
;	DB	0, 0, 0, 0, 0, 1, 0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 3	; 14
	DB	4, 5, 0, 0, 0, 3, 4, 5	
30	DB	0, 0, 0, 0, 2, 3, 0, 0	
	DB	0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 3	; 15
	DB	4, 5, 0, 0, 4, 5, 6, 7	
	DB	0, 0, 0, 0, 0, 0, 2, 3	
35	DB	0, 0, 0, 0	
	DB	4, 1, 0, 0, 2, 3, 0, 3	; 16

- 206 -

APPENDIX 1

```

        DB 4, 5, 0, 0, 4, 5, 6, 7
        DB 0, 0, 0, 0, 0, 3, 4, 5
        DB 0, 0, 0, 0, 0, 0, 0
    ENDIF

5 ;
    DCFontValue:
        DB 1, 0 ; 2
        DB 0, 0, 1, 2 ; 3
        DB 0, 0, 1, 0, 2, 3 ; 4
10 DB 0, 0, 1, 0, 2, 0, 3, 4 ; 5
    DB 0, 0, 0, 1, 0, 0, 2, 3 ; 6
    DB 4, 5
    DB 0, 0, 0, 1, 0, 0, 2, 3 ; 7
    DB 4, 0, 5, 6
15 DB 0, 0, 0, 1, 0, 0, 2, 3 ; 8
    DB 0, 0, 4, 5, 6, 7
    DB 0, 0, 0, 1, 0, 0, 2, 3 ; 9
    DB 0, 0, 4, 5, 6, 0, 7, 8
    DB 0, 0, 0, 1, 0, 0, 2, 3 ; 10
20 DB 0, 0, 4, 5, 0, 0, 6, 7
    DB 8, 9
    IF FontSize EQ 16
        DB 0, 0, 0, 1, 0, 0, 2, 3 ; 11
        DB 0, 0, 4, 5, 0, 0, 6, 7
25 DB 8, 0, 9, A
    DB 0, 0, 0, 1, 0, 0, 2, 3 ; 12
    DB 0, 0, 4, 5, 0, 0, 6, 7
    DB 0, 0, 8, 9, A, B
    DB 0, 0, 0, 1, 0, 0, 2, 0 ; 13 ERP
30 DB 3, 4, 0, 0, 5, 6, 0, 0
    DB 7, 8, 0, 0, 9, A, B, C
    ; DB 0, 0, 0, 1, 0, 0, 2, 0 ; 13 FLB
    ; DB 0, 0, 3, 4, 5, 0, 0, 0
    ; DB 6, 7, 8, 9, A, 0, B, C
35 DB 0, 0, 0, 1, 0, 0, 2, 0 ; 14
    DB 0, 0, 3, 4, 5, 0, 0, 0

```


- 207 -

APPENDIX 1

```

        DB    6, 7, 8, 9, 0, 0, A, B
        DB    C, D
        DB    0, 0, 0, 1, 0, 0, 2, 0          ; 15
        DB    0, 0, 3, 4, 0, 0, 0, 0
5       DB    5, 6, 7, 8, 9, A, 0, 0
        DB    B, C, D, E
        DB    0, 0, 0, 1, 0, 0, 2, 0          ; 16
        DB    0, 0, 3, 4, 0, 0, 0, 0
        DB    5, 6, 7, 8, 9, 0, 0, 0
10      DB    A, B, C, D, E, F
        ENDIF

;
;***** S H A R E D   T A B L E S   *****
;

15 Best128:
        DB
        20h,30h,45h,65h,0Ah,0Dh,31h,54h,74h,52h,32h,61h,49h,53h,41h,
        4Fh
        DB
20      72h,43h,4Eh,6Eh,4Ch,6Fh,69h,73h,09h,2Ch,44h,4Dh,35h,2Dh,33h,
        64h
        DB
        46h,2Eh,68h,50h,6Ch,38h,34h,29h,28h,39h,63h,55h,2Fh,3Dh,48h,
        36h
25      DB
        75h,66h,6Dh,42h,37h,70h,47h,57h,67h,58h,56h,62h,59h,77h,22h,
        79h
        DB
        2Ah,2Bh,5Fh,76h,27h,4Bh,25h,3Eh,21h,3Bh,5Ah,3Ch,24h,40h,3Ah,
30      6Bh
        DB
        4Ah,78h,26h,51h,5Bh,5Dh,23h,71h,7Ah,1Ah,6Ah,19h,3Fh,5Ch,00h,
        01h
        DB
35      02h,03h,04h,05h,06h,07h,08h,0Bh,0Ch,0Eh,0Fh,10h,11h,12h,13h,
        14h

```

- 208 -

APPENDIX 1

```

        DB
15h,16h,17h,18h,1Bh,1Ch,1Dh,1Eh,1Fh,5Eh,60h,7Bh,7Ch,7Dh,7Eh,
7Fh
;
5 FontBits:
        db 1,1                      ; 2
        db 1,2,2                    ; 3
        db 1,2,3,3                  ; 4
        db 1,2,3,4,4                ; 5
10      db 2,2,3,3,3,3              ; 6
        db 2,2,3,3,3,4,4            ; 7
        db 2,2,3,3,4,4,4,4          ; 8
        db 2,2,3,3,4,4,4,5,5        ; 9
        db 2,2,3,3,4,4,5,5,5,5      ; 10
15      IF FontSize EQ 16
        db 2,2,3,3,4,4,5,5,5,6,6    ; 11
        db 2,2,3,3,4,4,5,5,6,6,6,6  ; 12
        db 2,2,3,4,4,4,4,5,5,6,6,6,6 ; 13 ERP
;        db 2,2,3,4,4,4,5,5,5,5,5,6,6 ; 13 FLB
20      db 2,2,3,4,4,4,5,5,5,5,6,6,6,6 ; 14
        db 2,2,3,4,4,5,5,5,5,5,5,6,6,6,6 ; 15
        db 2,2,3,4,4,5,5,5,5,5,6,6,6,6,6,6 ; 16
        ENDIF
;
25 GlobalBits:
        DB
        04,04,04,04,04,04,04,04,05,05,05,05,05,05,05,05
        DB
        06,06,06,06,06,06,06,06,07,07,07,07,07,07,07,07
30      DB
        10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
        DB
        10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
        DB
35      12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12
        DB

```

- 209 -

APPENDIX 1

```

12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12
      DB
12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12
      DB
5 12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12
      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
10      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
      DB
15 13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
20      DB
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
;
GlobalCodeHigh:
      DB 11001000B,11011000B ; 0- 7
25      DB 11101000B,11111000B
      DB 10001000B,10011000B
      DB 10101000B,10111000B
      DB 01000100B,01001100B ; 8- 15
      DB 01010100B,01011100B
30      DB 01100100B,01101100B
      DB 01110100B,01111100B
      DB 00100010B,00100110B ; 16- 23
      DB 00101010B,00101110B
      DB 00110010B,00110110B
35      DB 00111010B,00111110B
      DB 00010001B,00010011B ; 24- 31

```

- 210 -

APPENDIX 1

	DB	00010101B,00010111B	
	DB	00011001B,00011011B	
	DB	00011101B,00011111B	
	DB	00001000B,00001000B ;	32- 47
5	DB	00001000B,00001000B	
	DB	00001001B,00001001B	
	DB	00001001B,00001001B	
	DB	00001010B,00001010B	
	DB	00001010B,00001010B	
10	DB	00001011B,00001011B	
	DB	00001011B,00001011B	
	DB	00001100B,00001100B ;	48- 63
	DB	00001100B,00001100B	
	DB	00001101B,00001101B	
15	DB	00001101B,00001101B	
	DB	00001110B,00001110B	
	DB	00001110B,00001110B	
	DB	00001111B,00001111B	
	DB	00001111B,00001111B	
20	DB	00000100B,00000100B ;	64- 79
	DB	00000100B,00000100B	
	DB	00000100B,00000100B	
	DB	00000100B,00000100B	
	DB	00000100B,00000100B	
25	DB	00000100B,00000100B	
	DB	00000100B,00000100B	
	DB	00000100B,00000100B	
	DB	00000101B,00000101B ;	80- 95
	DB	00000101B,00000101B	
30	DB	00000101B,00000101B	
	DB	00000101B,00000101B	
	DB	00000101B,00000101B	
	DB	00000101B,00000101B	
	DB	00000101B,00000101B	
35	DB	00000101B,00000101B	
	DB	00000110B,00000110B ;	96-111

- 211 -

APPENDIX 1

	DB	00000110B,00000110B
	DB	00000110B,00000110B
	DB	00000110B,00000110B
	DB	00000110B,00000110B
5	DB	00000110B,00000110B
	DB	00000110B,00000110B
	DB	00000110B,00000110B
	DB	00000111B,00000111B ; 112-127
	DB	00000111B,00000111B
10	DB	00000111B,00000111B
	DB	00000111B,00000111B
	DB	00000111B,00000111B
	DB	00000111B,00000111B
	DB	00000111B,00000111B
15	DB	00000111B,00000111B
	DB	00000000B,00000000B ; 128-143
	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B
20	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B ; 144-159
25	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000000B,00000000B
30	DB	00000000B,00000000B
	DB	00000000B,00000000B
	DB	00000001B,00000001B ; 160-175
	DB	00000001B,00000001B
	DB	00000001B,00000001B
35	DB	00000001B,00000001B
	DB	00000001B,00000001B

- 212 -

APPENDIX 1

	DB	00000001B,00000001B
	DB	00000001B,00000001B
	DB	00000001B,00000001B
	DB	00000001B,00000001B ; 176-191
5	DB	00000001B,00000001B
	DB	00000001B,00000001B
	DB	00000001B,00000001B
	DB	00000001B,00000001B
	DB	00000001B,00000001B
10	DB	00000001B,00000001B
	DB	00000001B,00000001B
	DB	00000010B,00000010B ; 192-207
	DB	00000010B,00000010B
	DB	00000010B,00000010B
15	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000010B,00000010B
20	DB	00000010B,00000010B ; 208-223
	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000010B,00000010B
25	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000010B,00000010B
	DB	00000011B,00000011B ; 224-239
	DB	00000011B,00000011B
30	DB	00000011B,00000011B
	DB	00000011B,00000011B
	DB	00000011B,00000011B
	DB	00000011B,00000011B
	DB	00000011B,00000011B
35	DB	00000011B,00000011B
	DB	00000011B,00000011B ; 240-255

- 213 -

APPENDIX 1

```

      DB  00000011B,00000011B
      DB  00000011B,00000011B
      DB  00000011B,00000011B
      DB  00000011B,00000011B
5      DB  00000011B,00000011B
      DB  00000011B,00000011B
      DB  00000011B,00000011B
      ;
GlobalCodeLow:
10      DB  00000000B,00000000B ; 0- 7
      DB  00000000B,00000000B
      DB  00000000B,00000000B
      DB  00000000B,00000000B
      DB  00000000B,00000000B ; 8- 15
15      DB  00000000B,00000000B
      DB  00000000B,00000000B
      DB  00000000B,00000000B
      DB  00000000B,00000000B ; 16- 23
      DB  00000000B,00000000B
20      DB  00000000B,00000000B
      DB  00000000B,00000000B
      DB  00000000B,00000000B ; 24- 31
      DB  00000000B,00000000B
      DB  00000000B,00000000B
25      DB  00000000B,00000000B
      DB  00100000B,01100000B ; 32- 47
      DB  10100000B,11100000B
      DB  00100000B,01100000B
      DB  10100000B,11100000B
30      DB  00100000B,01100000B
      DB  10100000B,11100000B
      DB  00100000B,01100000B
      DB  10100000B,11100000B
      DB  00100000B,01100000B ; 48- 63
35      DB  10100000B,11100000B
      DB  00100000B,01100000B

```

- 214 -

APPENDIX 1

	DB	10100000B,11100000B	
	DB	00100000B,01100000B	
	DB	10100000B,11100000B	
	DB	00100000B,01100000B	
5	DB	10100000B,11100000B	
	DB	00001000B,00011000B ;	64- 79
	DB	00101000B,00111000B	
	DB	01001000B,01011000B	
	DB	01101000B,01111000B	
10	DB	10001000B,10011000B	
	DB	10101000B,10111000B	
	DB	11001000B,11011000B	
	DB	11101000B,11111000B	
	DB	00001000B,00011000B ;	80- 95
15	DB	00101000B,00111000B	
	DB	01001000B,01011000B	
	DB	01101000B,01111000B	
	DB	10001000B,10011000B	
	DB	10101000B,10111000B	
20	DB	11001000B,11011000B	
	DB	11101000B,11111000B	
	DB	00001000B,00011000B ;	96-111
	DB	00101000B,00111000B	
	DB	01001000B,01011000B	
25	DB	01101000B,01111000B	
	DB	10001000B,10011000B	
	DB	10101000B,10111000B	
	DB	11001000B,11011000B	
	DB	11101000B,11111000B	
30	DB	00001000B,00011000B ;	112-127
	DB	00101000B,00111000B	
	DB	01001000B,01011000B	
	DB	01101000B,01111000B	
	DB	10001000B,10011000B	
35	DB	10101000B,10111000B	
	DB	11001000B,11011000B	

- 215 -

APPENDIX 1

	DB	11101000B,11111000B
	DB	00000100B,00001100B ; 128-143
	DB	00010100B,00011100B
5	DB	00100100B,00101100B
	DB	00110100B,00111100B
	DB	01000100B,01001100B
	DB	01010100B,01011100B
	DB	01100100B,01101100B
	DB	01110100B,01111100B
10	DB	10000100B,10001100B ; 144-159
	DB	10010100B,10011100B
	DB	10100100B,10101100B
	DB	10110100B,10111100B
	DB	11000100B,11001100B
15	DB	11010100B,11011100B
	DB	11100100B,11101100B
	DB	11110100B,11111100B
	DB	00000100B,00001100B ; 160-175
	DB	00010100B,00011100B
20	DB	00100100B,00101100B
	DB	00110100B,00111100B
	DB	01000100B,01001100B
	DB	01010100B,01011100B
	DB	01100100B,01101100B
25	DB	01110100B,01111100B
	DB	10000100B,10001100B ; 176-191
	DB	10010100B,10011100B
	DB	10100100B,10101100B
	DB	10110100B,10111100B
30	DB	11000100B,11001100B
	DB	11010100B,11011100B
	DB	11100100B,11101100B
	DB	11110100B,11111100B
	DB	00000100B,00001100B ; 192-207
35	DB	00010100B,00011100B
	DB	00100100B,00101100B

- 216 -

APPENDIX 1

```

5      DB  00110100B,00111100B
      DB  01000100B,01001100B
      DB  01010100B,01011100B
      DB  01100100B,01101100B
      DB  01110100B,01111100B
      DB  10000100B,10001100B ; 208-223
      DB  10010100B,10011100B
      DB  10100100B,10101100B
      DB  10110100B,10111100B
10     DB  11000100B,11001100B
      DB  11010100B,11011100B
      DB  11100100B,11101100B
      DB  11110100B,11111100B
      DB  00000100B,00001100B ; 224-239
15     DB  00010100B,00011100B
      DB  00100100B,00101100B
      DB  00110100B,00111100B
      DB  01000100B,01001100B
      DB  01010100B,01011100B
20     DB  01100100B,01101100B
      DB  01110100B,01111100B
      DB  10000100B,10001100B ; 240-255
      DB  10010100B,10011100B
      DB  10100100B,10101100B
25     DB  10110100B,10111100B
      DB  11000100B,11001100B
      DB  11010100B,11011100B
      DB  11100100B,11101100B
      DB  11110100B,11111100B
30 ;
      IF  1 EQ 0
LengthABits:
      DB
02,03,03,03,04,04,04,05,05,06,06,07,07,07,07,08
35     DB
08,08,08,09,09,09,09,09,09,10,10,10,10,10,10

```

- 217 -

APPENDIX 1

```

        DB
    10,10,10,11,11,11,11,11,11,11,11,11,11,12,12,12
        DB
    12,12,12,12,12,12,12,12,12,12,12,12,13,13,13,13,06
5 ;
    LengthACodeHigh:
        DB    11100000B,10110000B,10010000B,01110000B
        DB    01011000B,01001000B,00111000B,00101100B
        DB    00100100B,00011110B,00011010B,00010011B
10        DB    00010001B,00001111B,00001101B,00001011B
        DB    00001010B,00001001B,00001000B,00000111B
        DB    00000111B,00000110B,00000110B,00000101B
        DB    00000101B,00000100B,00000100B,00000100B
        DB    00000011B,00000011B,00000011B,00000011B
15        DB    00000010B,00000010B,00000010B,00000010B
        DB    00000010B,00000001B,00000001B,00000001B
        DB    00000001B,00000001B,00000001B,00000001B
        DB    00000001B,00000000B,00000000B,00000000B
        DB    00000000B,00000000B,00000000B,00000000B
20        DB    00000000B,00000000B,00000000B,00000000B
        DB    00000000B,00000000B,00000000B,00000000B
        DB    00000000B,00000000B,00000000B,00010100B ; no
guard bit
;
; on index 63
25 LengthACodeLow:
        DB    00000000B,00000000B,00000000B,00000000B
        DB    00000000B,00000000B,00000000B,00000000B
        DB    00000000B,00000000B,00000000B,00000000B
        DB    00000000B,00000000B,00000000B,10000000B
30        DB    10000000B,10000000B,10000000B,11000000B
        DB    01000000B,11000000B,01000000B,11000000B
        DB    01000000B,11000000B,01100000B,00100000B
        DB    11100000B,10100000B,01100000B,00100000B
        DB    11100000B,10100000B,01100000B,00110000B
35        DB    00010000B,11110000B,11010000B,10110000B
        DB    10010000B,01110000B,01010000B,00110000B

```

- 218 -

APPENDIX 1

```

DB 00010000B,11111000B,11101000B,11011000B
DB 11001000B,10111000B,10101000B,10011000B
DB 10001000B,01111000B,01101000B,01011000B
DB 01001000B,00111000B,00101000B,00011100B
5 DB 00010100B,00001100B,00000100B,00000000B
;
LengthANext:
DB 6, 1, 2, 0, 0, 0, 6, 1
DB 2, 0, 0, 0, 6, 1, 2, 0
10 DB 0, 0,10, 1, 4, 1, 0, 0
DB 2, 0, 0, 0,12, 1, 4, 1
DB 0, 0, 4, 1, 0, 0, 0, 0
DB 18, 1, 8, 1, 4, 1, 0, 0
DB 0, 0, 4, 1, 0, 0, 2, 0
15 DB 0, 0,18, 1, 8, 1, 4, 1
DB 0, 0, 0, 0, 4, 1, 0, 0
DB 2, 0, 0, 0,16, 1, 8, 1
DB 4, 1, 0, 0, 0, 0, 4, 1
DB 0, 0, 0, 0,16, 1, 8, 1
20 DB 4, 1, 0, 0, 0, 0, 4, 1
DB 0, 0, 0, 0, 8, 1, 4, 1
DB 0, 0, 0, 0, 4, 1, 0, 0
DB 4, 1, 0, 0, 0, 0, 0
;
25 LengthAValue:
DB 0, 0, 0, 0, 2, 1, 0, 0
DB 0, 3, 5, 4, 0, 0, 0, 6
DB 8, 7, 0, 0, 0, 0,10, 9
DB 0,63,12,11, 0, 0, 0, 0
30 DB 14,13, 0, 0,16,15,18,17
DB 0, 0, 0, 0, 0, 0,20,19
DB 22,21, 0, 0,24,23, 0,25
DB 27,26, 0, 0, 0, 0, 0, 0
DB 29,28,31,30, 0, 0,33,32
35 DB 0,34,36,35, 0, 0, 0, 0
DB 0, 0,38,37,40,39, 0, 0

```

- 219 -

APPENDIX 1

```

        DB    42,41,44,43, 0, 0, 0, 0
        DB     0, 0,46,45,48,47, 0, 0
        DB    50,49,52,51, 0, 0, 0, 0
        DB    54,53,56,55, 0, 0,58,57
5       DB     0, 0,60,59,62,61
        ENDIF

;
LengthBBits:
        DB    01,03,03,04,05,05,05,06,06,04
10      ;
LengthBCode:
        DB
        11000000B,01110000B,01010000B,00111000B,00011100B
        DB
15      00010100B,00001100B,00000110B,00000010B,00101000B
;
LengthBNext:
        DB     2, 0, 4, 1, 0, 0, 4, 1, 0, 0
        DB     4, 1, 0, 0, 2, 0, 0, 0, 0
20      ;
LengthBValue:
        DB     0, 0, 0, 0, 2, 1, 0, 0, 9, 3
        DB     0, 0, 5, 4, 0, 6, 8, 7
;
25      IF    BufferSize EQ 8192
ZoneBits:
        DB
        02,03,03,04,05,05,05,05,05,06,06,06,06,06,06
        DB
30      06,06,06,06,07,07,07,07,07,07,07,07,07,07,07
;
ZoneCode:
        DB    11100000B,10110000B,10010000B,01111000B
        DB    01101100B,01100100B,01011100B,01010100B
35      DB    01001100B,01000100B,00111110B,00111010B
        DB    00110110B,00110010B,00101110B,00101010B

```

- 220 -

APPENDIX 1

```

DB 00100110B,00100010B,00011110B,00011010B
DB 00010111B,00010101B,00010011B,00010001B
DB 00001111B,00001101B,00001011B,00001001B
DB 00000111B,00000101B,00000011B,00000001B
5 ;
ZoneNext:
DB 6, 1, 2, 0, 0, 0,14, 1
DB 6, 1, 2, 0, 0, 0, 4, 1
DB 0, 0, 0, 0,16, 1, 8, 1
10 DB 4, 1, 0, 0, 0, 0, 4, 1
DB 0, 0, 0, 0,12, 1, 4, 1
DB 0, 0, 4, 1, 0, 0, 0, 0
DB 8, 1, 4, 1, 0, 0, 0, 0
DB 4, 1, 0, 0, 0, 0, 0, 0
15 ;
ZoneValue:
DB 0, 0, 0, 0, 2, 1, 0, 0
DB 0, 0, 0, 3, 5, 4, 0, 0
DB 7, 6, 9, 8, 0, 0, 0, 0
20 DB 0, 0,11,10,13,12, 0, 0
DB 15,14,17,16, 0, 0, 0, 0
DB 19,18, 0, 0,21,20,23,22
DB 0, 0, 0, 0,25,24,27,26
DB 0, 0,29,28,31,30
25 ;
ELSE
ZoneBits:
DB
02,02,03,04,04,05,05,05,05,06,06,06,06,06,06
30 ;
ZoneCode:
DB 11100000B,10100000B,01110000B,01011000B
DB 01001000B,00111100B,00110100B,00101100B
DB 00100100B,00011100B,00010110B,00010010B
35 DB 00001110B,00001010B,00000110B,00000010B
;
```

- 221 -

APPENDIX 1

ZoneNext:

```

5      DB  4, 1, 0, 0, 6, 1, 2, 0
      DB  0, 0, 8, 1, 4, 1, 0, 0
      DB  0, 0, 6, 1, 2, 0, 0, 0
      DB  4, 1, 0, 0, 0, 0, 0

```

;

ZoneValue:

```

10      DB  0, 0, 1, 0, 0, 0, 0, 2
      DB  4, 3, 0, 0, 0, 0, 6, 5
      DB  8, 7, 0, 0, 0, 9, 11, 10
      DB  0, 0, 13, 12, 15, 14

```

ENDIF

;

;

15 CRC_TH:

```

      DB  000H, 011H, 023H, 032H, 046H, 057H, 065H, 074H ;000
      DB  08CH, 09DH, 0AFH, 0BEH, 0CAH, 0DBH, 0E9H, 0F8H ;008
      DB  010H, 001H, 033H, 022H, 056H, 047H, 075H, 064H ;010
      DB  09CH, 08DH, 0BFH, 0AEH, 0DAH, 0CBH, 0F9H, 0E8H ;018
20      DB  021H, 030H, 002H, 013H, 067H, 076H, 044H, 055H ;020
      DB  0ADH, 0BCH, 08EH, 09FH, 0EBH, 0FAH, 0C8H, 0D9H ;028
      DB  031H, 020H, 012H, 003H, 077H, 066H, 054H, 045H ;030
      DB  0BDH, 0ACH, 09EH, 08FH, 0FBH, 0EAH, 0D8H, 0C9H ;038
      DB  042H, 053H, 061H, 070H, 004H, 015H, 027H, 036H ;040
25      DB  0CEH, 0DFH, 0EDH, 0FCH, 088H, 099H, 0ABH, 0BAH ;048
      DB  052H, 043H, 071H, 060H, 014H, 005H, 037H, 026H ;050
      DB  0DEH, 0CFH, 0FDH, 0ECH, 098H, 089H, 0BBH, 0AAH ;058
      DB  063H, 072H, 040H, 051H, 025H, 034H, 006H, 017H ;060
      DB  0EFH, 0FEH, 0CCH, 0DDH, 0A9H, 0B8H, 08AH, 09BH ;068
30      DB  073H, 062H, 050H, 041H, 035H, 024H, 016H, 007H ;070
      DB  0FFH, 0EEH, 0DCH, 0CDH, 0B9H, 0A8H, 09AH, 08BH ;078
      DB  084H, 095H, 0A7H, 0B6H, 0C2H, 0D3H, 0E1H, 0F0H ;080
      DB  008H, 019H, 02BH, 03AH, 04EH, 05FH, 06DH, 07CH ;088
      DB  094H, 085H, 0B7H, 0A6H, 0D2H, 0C3H, 0F1H, 0E0H ;090
35      DB  018H, 009H, 03BH, 02AH, 05EH, 04FH, 07DH, 06CH ;098
      DB  0A5H, 0B4H, 086H, 097H, 0E3H, 0F2H, 0C0H, 0D1H ;0A0

```

- 222 -

APPENDIX 1

```

5      DB  029H,038H,00AH,01BH,06FH,07EH,04CH,05DH ;0A8
      DB  0B5H,0A4H,096H,087H,0F3H,0E2H,0D0H,0C1H ;0B0
      DB  039H,028H,01AH,00BH,07FH,06EH,05CH,04DH ;0B8
      DB  0C6H,0D7H,0E5H,0F4H,080H,091H,0A3H,0B2H ;0C0
      DB  04AH,05BH,069H,078H,00CH,01DH,02FH,03EH ;0C8
      DB  0D6H,0C7H,0F5H,0E4H,090H,081H,0B3H,0A2H ;0D0
      DB  05AH,04BH,079H,068H,01CH,00DH,03FH,02EH ;0D8
      DB  0E7H,0F6H,0C4H,0D5H,0A1H,0B0H,082H,093H ;0E0
      DB  06BH,07AH,048H,059H,02DH,03CH,00EH,01FH ;0E8
10     DB  0F7H,0E6H,0D4H,0C5H,0B1H,0A0H,092H,083H ;0F0
      DB  07BH,06AH,058H,049H,03DH,02CH,01EH,00FH ;0F8
;
CRC_TL:
15     DB  000H,089H,012H,09BH,024H,0ADH,036H,0BFH ;000
      DB  048H,0C1H,05AH,0D3H,06CH,0E5H,07EH,0F7H ;008
      DB  081H,008H,093H,01AH,0A5H,02CH,0B7H,03EH ;010
      DB  0C9H,040H,0DBH,052H,0EDH,064H,0FFH,076H ;018
      DB  002H,08BH,010H,099H,026H,0AFH,034H,0BDH ;020
      DB  04AH,0C3H,058H,0D1H,06EH,0E7H,07CH,0F5H ;028
20     DB  083H,00AH,091H,018H,0A7H,02EH,0B5H,03CH ;030
      DB  0CBH,042H,0D9H,050H,0EFH,066H,0FDH,074H ;038
      DB  004H,08DH,016H,09FH,020H,0A9H,032H,0BBH ;040
      DB  04CH,0C5H,05EH,0D7H,068H,0E1H,07AH,0F3H ;048
      DB  085H,00CH,097H,01EH,0A1H,028H,0B3H,03AH ;050
25     DB  0CDH,044H,0DFH,056H,0E9H,060H,0FBH,072H ;058
      DB  006H,08FH,014H,09DH,022H,0ABH,030H,0B9H ;060
      DB  04EH,0C7H,05CH,0D5H,06AH,0E3H,078H,0F1H ;068
      DB  087H,00EH,095H,01CH,0A3H,02AH,0B1H,038H ;070
      DB  0CFH,046H,0DDH,054H,0EBH,062H,0F9H,070H ;078
30     DB  008H,081H,01AH,093H,02CH,0A5H,03EH,0B7H ;080
      DB  040H,0C9H,052H,0DBH,064H,0EDH,076H,0FFH ;088
      DB  089H,000H,09BH,012H,0ADH,024H,0BFH,036H ;090
      DB  0C1H,048H,0D3H,05AH,0E5H,06CH,0F7H,07EH ;098
      DB  00AH,083H,018H,091H,02EH,0A7H,03CH,0B5H ;0A0
35     DB  042H,0CBH,050H,0D9H,066H,0EFH,074H,0FDH ;0A8
      DB  08BH,002H,099H,010H,0AFH,026H,0BDH,034H ;0B0

```


- 223 -

APPENDIX 1

```

        DB    0C3H,04AH,0D1H,058H,0E7H,06EH,0F5H,07CH ;0B8
        DB    00CH,085H,01EH,097H,028H,0A1H,03AH,0B3H ;0C0
        DB    044H,0CDH,056H,0DFH,060H,0E9H,072H,0FBH ;0C8
        DB    08DH,004H,09FH,016H,0A9H,020H,0BBH,032H ;0D0
5       DB    0C5H,04CH,0D7H,05EH,0E1H,068H,0F3H,07AH ;0D8
        DB    00EH,087H,01CH,095H,02AH,0A3H,038H,0B1H ;0E0
        DB    046H,0CFH,054H,0DDH,062H,0EBH,070H,0F9H ;0E8
        DB    08FH,006H,09DH,014H,0ABH,022H,0B9H,030H ;0F0
        DB    0C7H,04EH,0D5H,05CH,0E3H,06AH,0F1H,078H ;0F8
10
        ;

        printstat Data,size,is,$$-tb

        ;
        ;***** D E B U G G E R *****
15      ;
        ;   DEBUGGER or DUMMY INCLUSION
        ;

        include      TCdbg001

        ;
20      ; * * * * *
        ;
        unplanned_int:
            brk
        ;break:
25      ;break_out:
            nop
            nop
            nop
            rti
30      ;

        ;***** V E C T O R   T A B L E *****
        ;
        ;   VECTOR TABLE
        ;
35      ;   ds      0-progaddr-($-cb)-32,0
        ; Js b 0

```

- 224 -

APPENDIX 1

```

        dw    unplanned_int
; Jsb 1
        dw    unplanned_int
; Jsb 1
5        dw    unplanned_int
; Jsb 3
        dw    unplanned_int
; Jsb 4
        dw    unplanned_int
10 ; Jsb 5
        dw    unplanned_int
; Jsb 6
        dw    unplanned_int
; Jsb 7
15        dw    unplanned_int
;
; Irq6,break,PTGA,PTGb,bE
        dw    break
; Irq5,SerIn Stat, TimerA
20        dw    break_out
; Irq4,PA3,Edge/bF
        dw    unplanned_int
; Irq3,Host/Timerb
        dw    HostInt
25 ; Irq2,Pb0 Edge
        dw    unplanned_int
; Irq1,Pd7 Edge
        dw    unplanned_int
; NMI
30        dw    unplanned_int
; unplanned_int
reset:
        dw    dbginit          ; start in debugger
        printstat    <C000-FFFFh Block Free
35 =>,%16384-($-cb)
;          end

```

- 225 -

APPENDIX 2

SOURCE LISTING GUIDE

- Page 1, lines 1 through 11
Define the assembly environment.
- Page 1, line 12
- 5 The "include ITEC19" statement copies a source file which uses the MACRO facility in the assembler to provide some higher level language type constructs.
- Page 1, line 13
- 10 The "include TCDFM001" statement copies a source file which defines the internal register and I/O structure in the C19.
- Page 1, lines 14 through 28
Assembly macros used to manage/display the assembly environment and status.
- 15 Page 1, lines 33 through 40
Setting of symbols which control some assembly time features of the algorithm. These are used to enable/disable various structures and code to evaluate compression effectiveness.
- 20 Page 1, line 46 through Page 2, line 32
More assembly time controls which affect compression mechanisms and establish sizes of certain memory structures.
- Page 2, line 38 through Page 3, line 8
- 25 Assembly time controls for diagnostics and speed of execution having little or no effect on compression effectivity.
- Page 3, lines 12 through 42
- 30 Definition (mapping) of some structures used by the algorithm.
- Page 3, line 47 through Page 6, line 36
- 35 Declaration of byte (8 bit) and word (16 bit) variables used by the encoding and decoding processes. Variables beginning with "EC" are used by the encoder (compression process) and beginning with "DC" are the decoder (decompression process). Other prefixes are

- 226 -

APPENDIX 2

general use.

Page 6, line 42 through Page 7, line 36

Declaration of encoder/decoder structures which are a multiple of 256 bytes in length.

5 Page 7, line 40 through Page 8, line 27

Declaration of more encoder structures that are a multiple of 256 bytes in length. This block is from absolute address 4000h to 0c000h (size = 32768 bytes) and is bank switched alternating with the next described block.

10

Page 8, line 33 through Page 9, line 5

The decoder bankswitched block (size = 32768).

Page 9, line 10 through Page 11, line 48

Interface points to the operating system code for the production implementation of the algorithm. In production, these hooks replace development environment code on pages 22 through 25 inclusive.

15

Page 11, line 11 through Page 14, line 14

Table initialize code. All compression/decompression tables and variables are set to initial conditions.

20

Page 14, line 18 through Page 16, line 46

Program startup code which sets environment and initializes stacks for alternate execution of encoder/decoder.

25 Page 17, lines 1 through 19

Context switch subroutines.

Page 17, line 25 through Page 26, line 3

Development environment routines for Memory Dump to PC and character transfer to/from PC bus. Characters transferred are to be compressed or decompressed. The PC interface is an emulation of a standard PC asynchronous communications IC an INS16450.

30

Page 26, line 11 through Page 28, line 24

Macro declarations which facilitate the generation of certain microcode routines for bit stream output as either in-line code or as subroutines.

35

- 227 -

APPENDIX 2

Page 28, line 30 through Page 29, line 11

5 A macro which embodies the microcode to select the appropriate one of four NCToFrequency tables based on the prior character of the input stream, leaving the base address of the table ECNCChar in ECWord1 and the base address of the table ECNCFreq in ECWord2.

Page 29, line 17 through Page 41, line 21

10 The body of the FontUpdate Macro. This generates all of the microcode to perform the processes of CRC Hash generation, Font access, Font creation, etc. In general, all of the processes (steps) 3 through 9 as described with Figure 2B.

Page 41, line 27 through Page 45, line 44

15 The Encode main loop first phase. This is the Refill process which accepts characters from the input stream, stores them in the process buffer (ECChar, ECCharCopy), invokes the FontUpdate Macro (process). As required by flush operations and ProcessBuffer full conditions, this process invokes the second phase of execution.

20 Page 46, line 1 through Page 48, line 37

25 The Mode A string search macro. This embodies the code to locate the longest string in the history buffer matching the string beginning at the position of the ECChar buffer at position A (the value in the C19 accumulator register).

Page 48, line 41 through Page 52, line 43

30 The Mode A string find routines. These routines perform two iterations of the above macro, reject strings overlapping the next history buffer stream location, select the longer of the two if two were found.

Page 52, line 44 through Page 54, line 44

35 The Mode A Pair Encoding and bit cost comparison routines.

Mode A String bit cost computation and comparison with

- 228 -

APPENDIX 2

Font encoding.

Page 57, line 25 through Page 62, line 8

Mode A bit stream format and output routines.

Page 62, line 12 through Page 67, line 34

- 5 Mode A repeats output, history buffer and access table update routine, and phase 2 iteration (flush mode or normal) control.

Page 67, line 40 through Page 72, line 3

Mode B String search macro routines.

- 10 Page 72, line 6 through Page 74, line 16

Mode B String find routines. Performs Mode B string search macros and rejects strings which overlap next history buffer store location.

Page 74, line 17 through Page 76, line 13

- 15 Mode B Pair encoding and bit cost comparison subroutines.

Page 76, line 14 through Page 78, line 27

Mode B string bit cost comparison routines.

Page 78, line 32 through Page 79, line 40

- 20 Mode B antiexpansion summing routines.

Page 79, line 46 through Page 80, line 49

Mode B history buffer and access table update.

Page 81, line 1 through Page 92, line 18

Mode B bit stream format and output.

- 25 Page 92, line 17 through Page 96, line 29

Decoder macros for character input/output and bit stream fetch.

Page 96, line 33 through Page 105, line 17

Decoder main body.

- 30 Page 105, line 26 through Page 107 line 9

Encoding Table and FontCode (Huffman Font codes) tables. Used to emit Font encoding bit patterns.

Page 107, line 14 through Page 109, line 9

Decoder Huffman Font decoding trees.

- 35 Page 109, line 13 through Page 109, line 21

New Character to Frequency preload tables.

- 229 -

APPENDIX 2

Page 109, line 23 through Page 109, line 41

Font Bits table. Used for computing bit cost of Font encodings.

Page 109, line 43 through Page 110, line 10

5 Global bits Table. Used to compute the bit cost of NewChar and any other encodings which use the GlobalBits tables.

Page 110, line 12 through Page 115, line 25

10 The GlobalBits High and Low Huffman tables, used as a pair to encode any items such as NewChar, Mode A String length, and Repeat count.

Page 115, line 27 through Page 117, line 7

The LengthA encoding tables. Not in use by the preferred embodiment.

15 Page 117, line 9 through Page 117, line 22

The LengthBBits LengthBCode, LengthBValue and LengthBNext tables. LengthBBits and LengthB value are used for encoding the Mode B string length.

20 LengthBNext and LengthBValue are used for decoding Mode B string lengths.

Page 117, line 24 through Page 118, line 32

The ZoneBits, ZoneCode, ZoneNext and ZoneValue tables. ZoneBits and ZoneCode are used for encoding the Zone portion of Mode A and Mode B string location offsets.

25 ZoneNext and ZoneValue are used for decoding the Zone portion of Mode A and Mode B string location offsets.

Page 118, line 34 through Page 120, line 3

The precalculated CRC table. Used for rapid CRC hash calculations in the FontAccess Routines.

30 Page 120, line 11

Inclusion of the C19 debugger (soft monitor) file.

Page 120, line 15 through Page 121, line 17

Vector jump tables for the C19 hardware vectoring system.

- 230 -

What is claimed is:

1. A system for the dynamic encoding of a character stream, the system comprising:
 - an input for receiving the character stream;
 - an output for providing encoded data;
 - single character encoding means, connected to the input, for providing, for a given character, an encoded signal indicative of the given character, including
 - a) means, hereinafter referred to as "font means,"
10 connected to the input, associated with a character pair, hereinafter referred to as "the given character pair", for storing, accessing and updating for each given character of a plurality of characters, a table listing the set of candidates for the character that may follow the given
15 character pair in the stream, such table hereinafter referred to as a "font"; wherein all the candidates in such font are stored in approximate order of their local frequency of occurrence after the given character pair with which the font is associated;
 - 20 b) font identification means, connected to the input, for identifying the font, hereinafter referred to as the "given font", for that character in the stream at the input; and
 - c) position encoding means for providing, for one
25 given character, a signal indicative of the position, occupied by the given character, in the given font;
 - string encoding means, connected to the input, for providing, for a given string of characters, an encoded signal indicative of the given string of characters,
30 including:
 - a) a history buffer;
 - b) history buffer access means for finding a candidate string in the history buffer; and
 - c) longest match search means for searching for
35 longest match by comparing an object string in the character stream with a candidate string in the history buffer; and
 - output selection means for accepting encoded signals

- 231 -

from the single character encoding means and encoded signals from the string encoding means and selectively sending these encoded signals to the output;

wherein the font identification means further includes
5 hash encoding means for producing hash codes and hash code storage means for storing hash codes and the history buffer access means further includes means for retrieving hash codes from the hash code storage means, such that a common hash code is used by both the font encoding means and the
10 string encoding means.

2. A system according to claim 1, wherein the hash encoding means includes means for applying a CRC algorithm to an ordered character pair to produce a hash code

3. A system according to claim 1, further including:
15 means for maintaining a value for the position of any character that is not otherwise listed in the font, such character hereinafter referred to as "new character" or "NC", in relation to other candidates in a given font, in approximate order of such new character's local frequency of
20 occurrence after the given character pair;

such that new character is assigned a "virtual position" in the font, as distinct from a position that is associated with a location in the font capable of storing a specific candidate character; and

25 such that the address of the position of each candidate character below the new character position in the table is incremented by 1.

4. A system according to claim 3, further including:

a plurality of NC fonts, each font listing the
30 candidates for the new character which may follow a given set of characters in the character stream wherein all the candidates in such font are stored in approximate order of their local frequency of occurrence after the given set of characters with which the font is associated; and

35 NC font selection means for selecting the NC font to be used to encode a given new character based on predefined bits from the set of characters preceding the given

- 232 -

character in the character stream.

5. A system according to claim 4, wherein the number of NC fonts is four and the predefined bits are bits 5 and 6 from the character prior to the given character.

5 6. A system according to claim 1, further including:
means for maintaining a value for the position of a string in a given font.

7. A system according to claim 1, further including:
means for maintaining a value for the position of a new
10 character, i.e., any character that is not otherwise listed in the font, in relation to other candidates in a given font, in approximate order of such new character's local frequency of occurrence after the given character pair; and
means for maintaining a value for the position of a
15 string;

such that the value for the position of a string is one greater than the value for the position of a new character;

such that the string is assigned a "virtual position" in the font as distinct from a position associated with a
20 location in the font capable of storing a specific candidate character; and

such that the address of the position of each candidate character below the new character position in the font is incremented by 2.

25 8. A system according to claim 1, further including:
repeat character encoding means for encoding repeat character sequences, i.e. characters all alike, found in the character stream;

wherein the history buffer stores characters found in
30 the character stream; and

wherein repeat character sequences having three or more characters are represented in the history buffer by three characters only.

9. A system according to claim 3, wherein the string
35 encoding means has a plurality of modes of operation, the system further including:

means for summing, over a predetermined number of new

- 233 -

character occurrences, the bit-count of the code for each new character encoded;

means for comparing the sum with a predetermined value; and

5 switch means for switching modes whenever the bit-count exceeds the predetermined value.

10. A system according to claim 9, wherein the predetermined value has a value between seven bits per character and eight bits per character.

10 11. A system according to claim 9, wherein the predetermined value is 7.5 bits per character.

12. A system for providing, for a given string of characters, an encoded signal indicative of the given string of characters, comprising:

15 a) a history buffer tagged at regular intervals;

b) history buffer access means for finding a candidate string in the history buffer; and

c) longest match search means for searching for longest match by comparing an object string in the character stream with a candidate string in the history buffer;

20 d) a hash head table, which may be entered by a hash code derived from consecutive characters;

e) a hash link/test table, having a number of records equal to the number of tagged entries in the history buffer, each record having a link field and a test field and an address related to the address of the corresponding tagged entry in the history buffer;

wherein the hash head table contains pointers, consisting of part of the hash code, each pointing to the first candidate match in a linked list of candidates in the Hash Link field and the Hash Test field contains a match value consisting of another part of the hash code.

13. A system according to claim 12, wherein the longest match search means includes means for testing for a match beginning at a character in the candidate string at least one character ahead of the first character in such string.

14. A system according to claim 13, wherein the longest

- 234 -

match search means includes means for testing for a match beginning at character "n" ahead of the first character of the candidate string in the history buffer, where "n" is the length of the longest match found so far, and searching
5 forward to identify the longest match.

15. A system according to claim 14, wherein the longest match search means further includes means for searching back for the longest match.

16. A system according to claim 12, further including means
10 for discarding string matches having less than a predetermined number of characters.

17. A system according to claim 16, wherein the predetermined number is 3.

18. A system according to claim 12, wherein the linked list
15 is terminated by non-match of the contents of the Hash Test field with its corresponding part of the hash code.

19. A system according to claim 1, further including pair encoding means for encoding two characters by presenting the two characters in sequence to a CRC algorithm.

20 20. A system according to claim 19, wherein pair encoding processes and string encoding processes may be active at the same time.

21. An improved data compression modem of the type having terminal interface control means for controlling an
25 interface with a terminal, data compression means for compressing data from the terminal, line control means for controlling data flow over a data line, line interface means for interfacing with a data line, wherein the improvement comprises:

30 (a) first processor means for controlling both flow of data over the interface with the terminal and for compressing data from the terminal, and

(b) second processor means for controlling flow of data over the data line.

35 22. An improved data compression modem of the type having terminal interface control means for controlling an interface with a terminal, data compression and

- 235 -

decompression means for compressing data received from the terminal and for decompressing data going to the terminal, line control means for controlling data flow over a data line, line interface means for interfacing with a data line,

5 wherein the improvement comprises:

(a) first processor means for controlling both flow of data over the interface with the terminal and for compressing data received from the terminal, and for decompressing data going to the terminal, and

10 (b) second processor means for controlling flow of data over the data line.

23. An improved data compression modem according to claim 21, wherein the first processor and the second processor access a common memory.

15 24. A method for dynamically encoding a character stream, in an encoder having a history buffer and fonts, comprising the following steps:

a) receiving the character stream;

20 b) creating, from a two-character string, having a first character and a second character, a hash code;

c) associating each font with a pair of characters;

d) maintaining the position of a candidate character in a font in approximate order of the local frequency of occurrence of the candidate character in the character stream after the pair of characters with which the font is associated;

25 e) encoding a given character using the hash code to access the font associated with the pair of characters immediately preceding the given character in the character stream; and

30 f) encoding a given string of characters using the hash code to access a matching string in the history buffer.

25. A method for dynamically encoding a character stream, in an encoder having a history buffer and having fonts that are dynamically created and updated, comprising:

35 a) receiving the character stream;

b) creating, from a two-character string, having a

- 236 -

first character and a second character, a hash code having desirable statistical properties and a match code;

- c) associating each font with a pair of characters;
- d) maintaining the position of a candidate character
5 in a font in approximate order of the local frequency of occurrence of the candidate character in the character stream after the pair of characters with which the font is associated;

- e) encoding a given character using the hash code and
10 the match code to access the font associated with the pair of characters immediately preceding the given character in the character stream; and

- f) encoding a given string of characters using the hash code and the match code to access a matching string in
15 the history buffer.

26. A method for creating, from a two-byte string having a first byte and a second byte, a hash code having desirable statistical properties, comprising:

- encoding the two-byte string by presenting the two
20 bytes in sequence to a CRC algorithm to produce a CRC hash; and

designating selected bits from the CRC hash for use as a hash code.

27. A method for creating, from a two-byte string having a
25 first byte and a second byte, a hash code having desirable statistical properties and a match code for resolving ambiguity, comprising:

- encoding the two-byte string by presenting the two
bytes in sequence to a CRC algorithm to produce a CRC hash;
30 designating selected bits from the CRC hash for use as a hash code; and

designating the remaining bits from the CRC hash for use as a match code.

28. A method according to claim 27, wherein ten bits are
35 selected for use as a hash code.

29. A method for accessing a specific font within a data processing system, the system having a link table and a

- 237 -

plurality of fonts, each font being uniquely associated with a specific character pair, comprising:

accepting a pair of characters, having a first character and a second character, each character represented
5 by a single byte;

encoding the pair of characters using a CRC algorithm to produce a CRC hash;

selecting a first part of the CRC hash as a look-up code;

10 linking, in the link table, those fonts that are associated with pairs of characters whose encoding produces the same first part of the CRC hash;

entering the hash table with the look-up code to access a linked list of fonts; and

15 identifying, from among the fonts in the linked list, the specific font corresponding to the pair of characters, by matching the remainder of the CRC hash.

30. A method according to claim 29, wherein the method of encoding the pair of characters includes:

20 encoding the two-byte string by presenting the two bytes in sequence to a CRC algorithm to produce a CRC hash; and

designating selected bits from the CRC hash for use as a hash code.

25 31. A method according to claim 29, wherein the first part of the CRC hash consists of ten bits.

32. A method, for accessing a specific pair of characters in a history buffer within a system for the dynamic encoding of a character stream, the system having a history buffer
30 containing characters from the character stream, and a link table, comprising:

accepting a pair of characters from the character stream, hereinbelow referred to as "the given pair of characters", each pair having a first character and a second
35 character, each character represented by a single byte;

encoding the given pair of characters using a CRC algorithm to produce a CRC hash;

- 238 -

selecting a first part of the CRC hash as a look-up code;

linking, in the link table, history buffer entry points that have pairs of characters in the history buffer whose
5 encoding produces the same first part of the CRC hash;

entering the hash table with the look-up code to access a linked list of history buffer entry points; and

identifying, from among the history buffer entry points in the linked list, points corresponding to the given pair
10 of characters, by matching the remainder of the CRC hash.

33. A method according to claim 32, wherein the method of encoding the given pair of characters using the CRC algorithm to produce the CRC hash comprises:

encoding the byte representing the first character
15 using the CRC algorithm to produce an intermediate CRC hash;

encoding the second character using the CRC algorithm and the intermediate CRC hash to produce the CRC hash.

34. A method according to claim 32, wherein the first part of the CRC hash consists of ten bits.

20 35. A method, for accessing a specific sequence of four characters in a history buffer within a system for the dynamic encoding of a character stream, the system having a history buffer containing characters from the character stream, and a link table, comprising:

25 accepting four consecutive characters, hereinbelow referred to as "the given four characters", comprising a first pair of consecutive characters and a second pair of consecutive characters, each pair having a first character and a second character, each character represented by a
30 single byte, from the character stream;

encoding the given four characters using a CRC algorithm to produce a hash code;

selecting a first part of the hash code as a look-up code;

35 linking, in the link table, history buffer entry points that have four sequential characters in the history buffer whose encoding produces the same first part of the hash

- 239 -

code;

entering the link table with the look-up code to access a linked list of history buffer entry points; and

identifying, from among the history buffer entry points
5 in the linked list, points corresponding to the given four characters, by matching the remainder of the hash code.

36. A method according to claim 35, wherein the method of encoding the given four characters using a CRC algorithm to produce a hash code comprises:

10 encoding the first pair of characters by presenting the characters in sequence to a CRC algorithm to produce a first pair CRC hash;

encoding the second pair of characters by presenting the characters in sequence to a CRC algorithm to produce a
15 second pair CRC hash;

subtracting the second pair CRC hash from zero to produce a negated second pair CRC hash; and

performing an Exclusive OR operation on the first pair CRC hash and the negated second pair CRC hash to produce a
20 hash code.

37. A method according to claim 35, wherein the first part of the hash code consists of ten bits.

38. A method for controlling the selection of alternative string encoding modes in a system for the dynamic encoding
25 of a character stream, comprising:

maintaining a set of fonts, each font being associated with a pair of characters, wherein all the candidates in such font are stored in approximate order of their local frequency of occurrence after the given character pair with
30 which the font is associated, the fonts further including means for maintaining the position of a symbol for a new character, i.e., any character that is not otherwise listed in the font, in relation to other candidates in a given font in approximate order of such symbol's local frequency of
35 occurrence after the given character pair;

maintaining a new character encoding table;

encoding new characters from the character stream,

- 240 -

according to the position of the new character in the new character encoding table;

summing the bit cost of encoding each new character over a predetermined plurality of new character occurrences;

5 comparing the sum with a predetermined value; and
switching modes whenever the bit-count exceeds the predetermined value.

39. A method for encoding a character pair, within a system for the dynamic encoding of a character stream, the system
10 having a link table and a plurality of fonts, each font being uniquely associated with a specific character pair, comprising:

accepting a pair of characters, having a first character and a second character, each character represented
15 by a single byte;

encoding the pair of characters using a CRC algorithm to produce a CRC hash;

selecting ten bits of the CRC hash as a look-up code;

linking, in the link table, those fonts that are
20 associated with pairs of characters whose encoding produces the same first part of the CRC hash;

entering the hash table with the look-up code to access a linked list of fonts;

identifying, from among the fonts in the linked list,
25 the specific font corresponding to the pair of characters, by matching the remaining six bits of the CRC hash; and

encoding the pair of characters as the relative address of the identified font.

40. In a system for dynamic encoding of a character stream
30 having a link table and a plurality of fonts, each font being associated with a unique, ordered character pair, the system encoding a given character by means of the font associated with the pair of characters immediately preceding a given character in the character stream, a method for
35 maintaining fonts that are most recently used, comprising:

accepting a pair of characters, having a first character and a second character, each character represented

- 241 -

by a single byte;
encoding the pair of characters using a CRC algorithm
to produce a CRC hash;
selecting a first part of the CRC hash as a look-up
5 code;
linking, in the link table, fonts that are associated
with pairs of characters whose encoding produces the same
first part of the CRC hash;
entering the hash table with the look-up code to access
10 a linked list of fonts;
identifying, from among the fonts in the linked list,
the specific font corresponding to the pair of characters,
by matching the remainder of the CRC hash; and
discarding a font, when a font must be discarded, whose
15 associated character pair was least recently encountered in
the character stream.

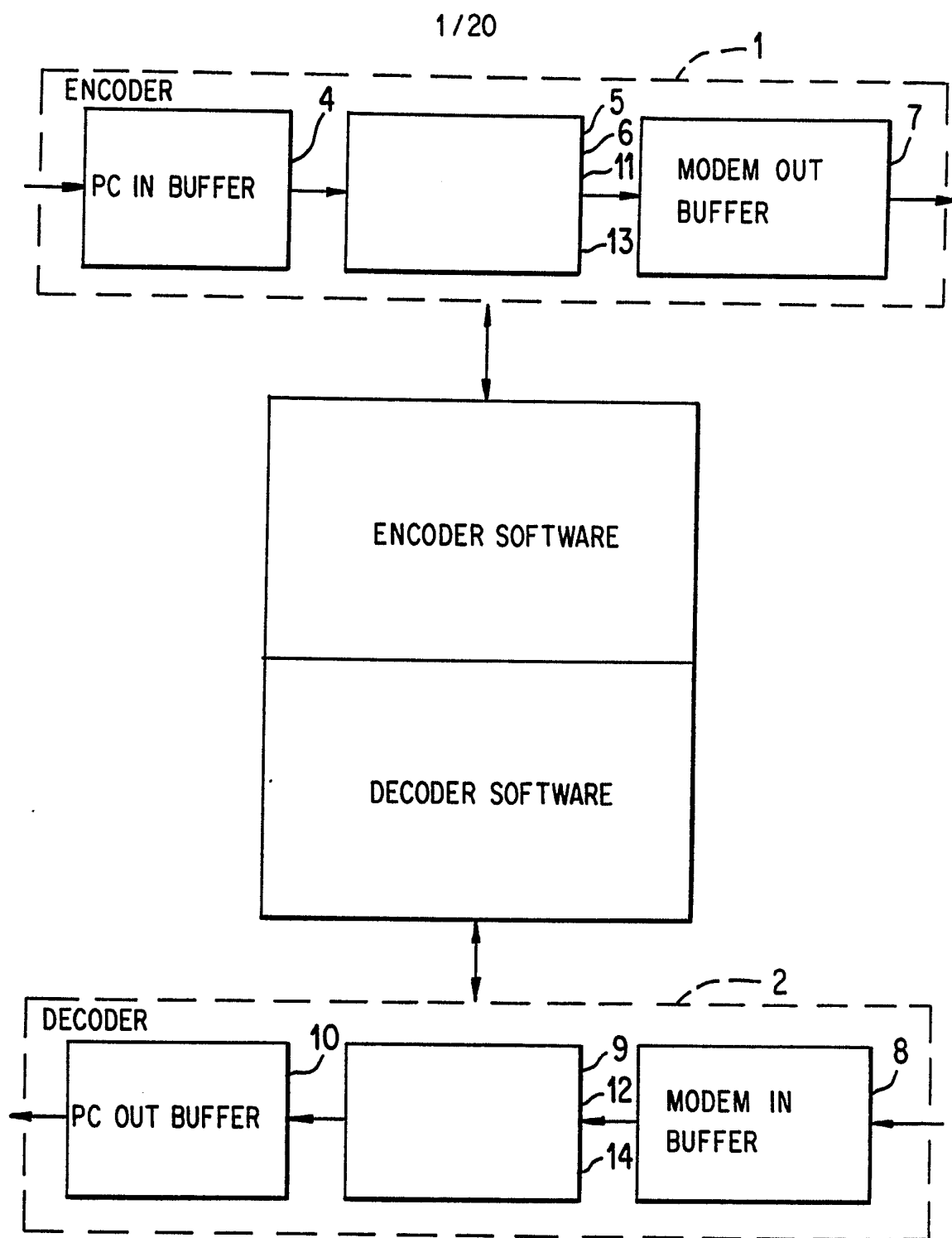
41. A method for use in a data processing system for
finding an object string within a data string comprising:
presenting the object string to a CRC algorithm to
20 produce an object string hash code;
presenting each of a plurality of candidate strings
within the data string to a CRC algorithm to produce a
candidate string hash code for each candidate string;
identifying candidate strings whose hash code matches
25 the object string hash code;
testing a candidate string, whose hash code matches the
object string hash code, for a match with the object string.

42. A method for finding the longest match between an
object string in a stream of characters and candidate
30 strings in a buffer, comprising:
comparing a character in the object string with a
character in a first candidate string;
comparing, if the prior comparison yields a match, each
next character in the object string with each next character
35 in the first candidate string until the comparison fails to
yield a match;
storing the number of characters so matched as the

- 242 -

length of the longest match;

comparing a character in the object string with a character in a second candidate string, starting at a character ahead of the origin of each string by a number of 5 characters substantially equal to the length of the longest match.



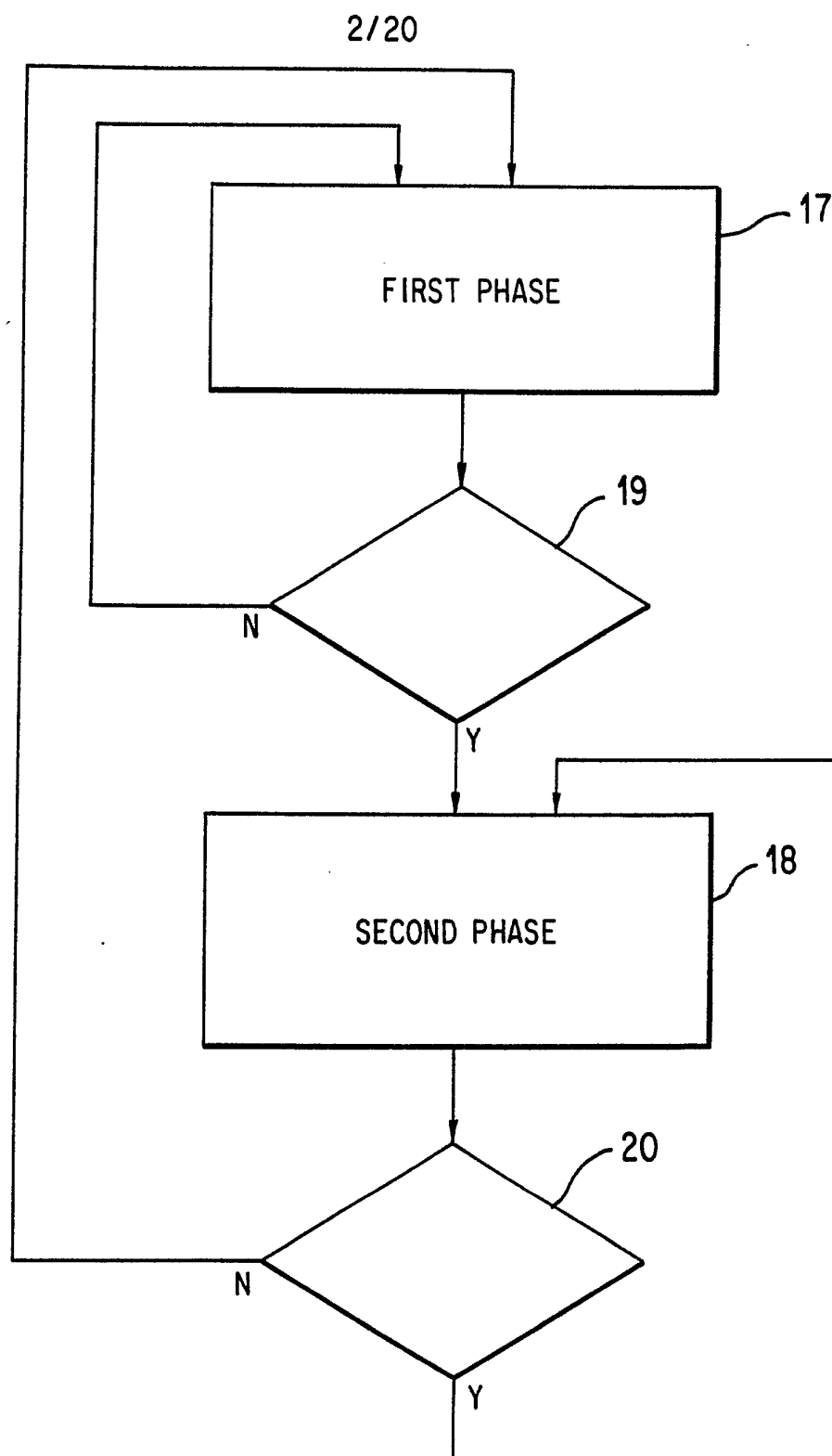


FIG. 1B

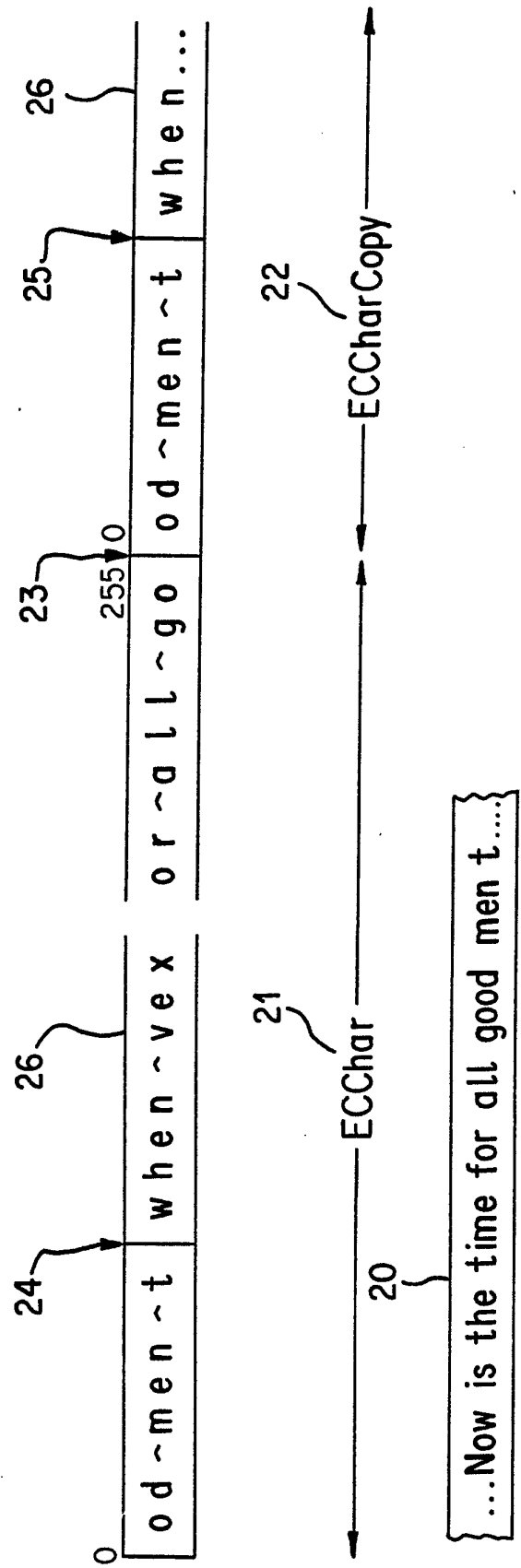
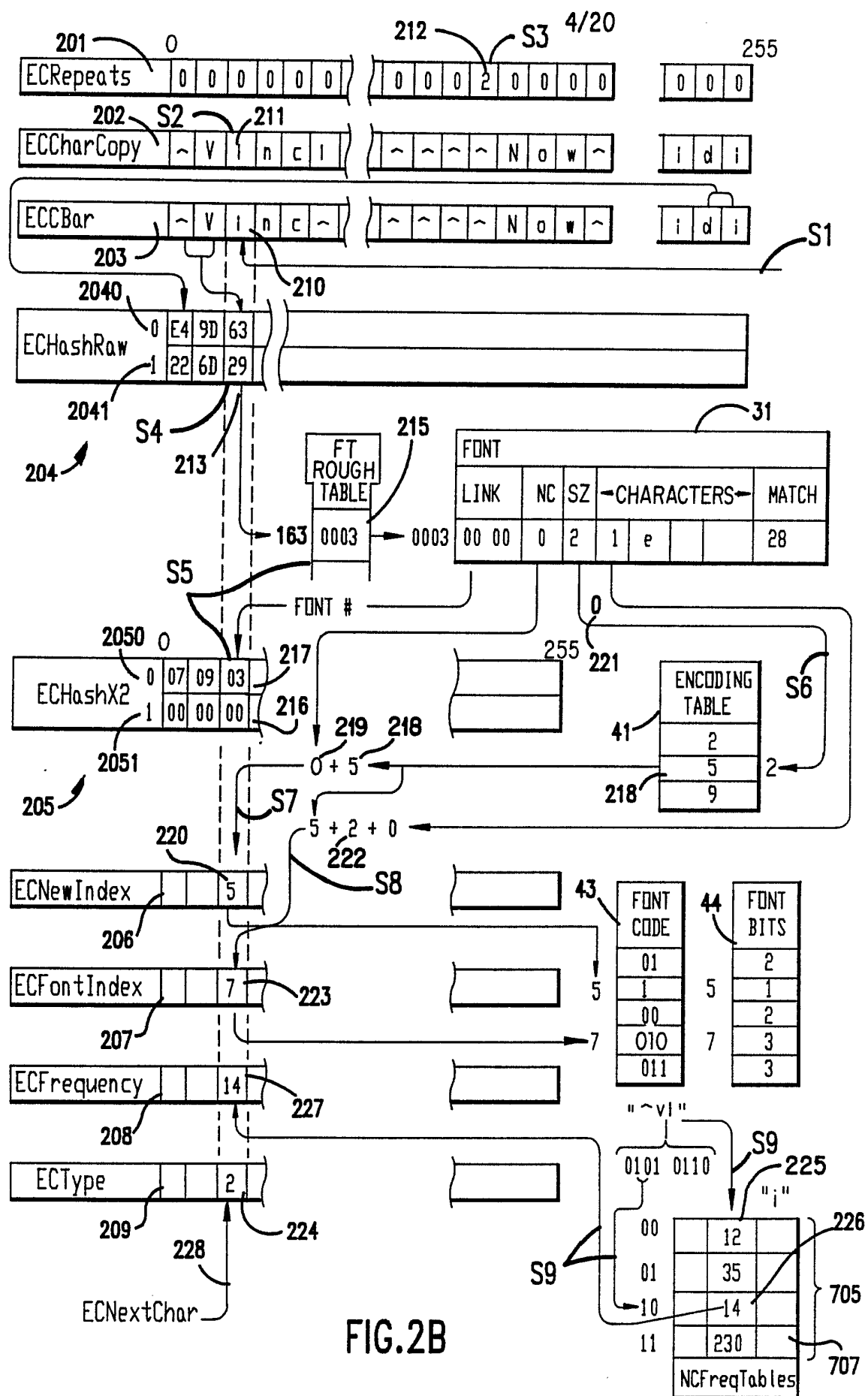


FIG. 2A



32

33 34 35 36 37

31

	LINK	MATCH	NC	SZ	CHARACTERS					
0000	0000	00	00	00						
0001	0000	00	00	01	^					
	0000	E4	00	01	V					
0003	0000	28	00	02	i					
	0000	AC	00	01	n					
	0000	AC	00	01	i	e				
	0000	1C	00	01	.					
	0000	20	01	01	^					
	0000	6C	01	01	V					
	0000	64	00	02	d	n				
	0000	EC	00	01	i					
	0000	E0	00	01	.					
000C	0013	04	00	01	c					
	0000	B0	00	01	i					
	0000	AC	00	01	.					
	0000	44	00	01	^					
	0000	5C	00	01	A					
	0000	6C	00	01	^					
	0000	B8	00	01	d					
0013	0000	7C	00	01	o					
⋮										
01FF	00	00	00	00	00					

FIG.3

6/20

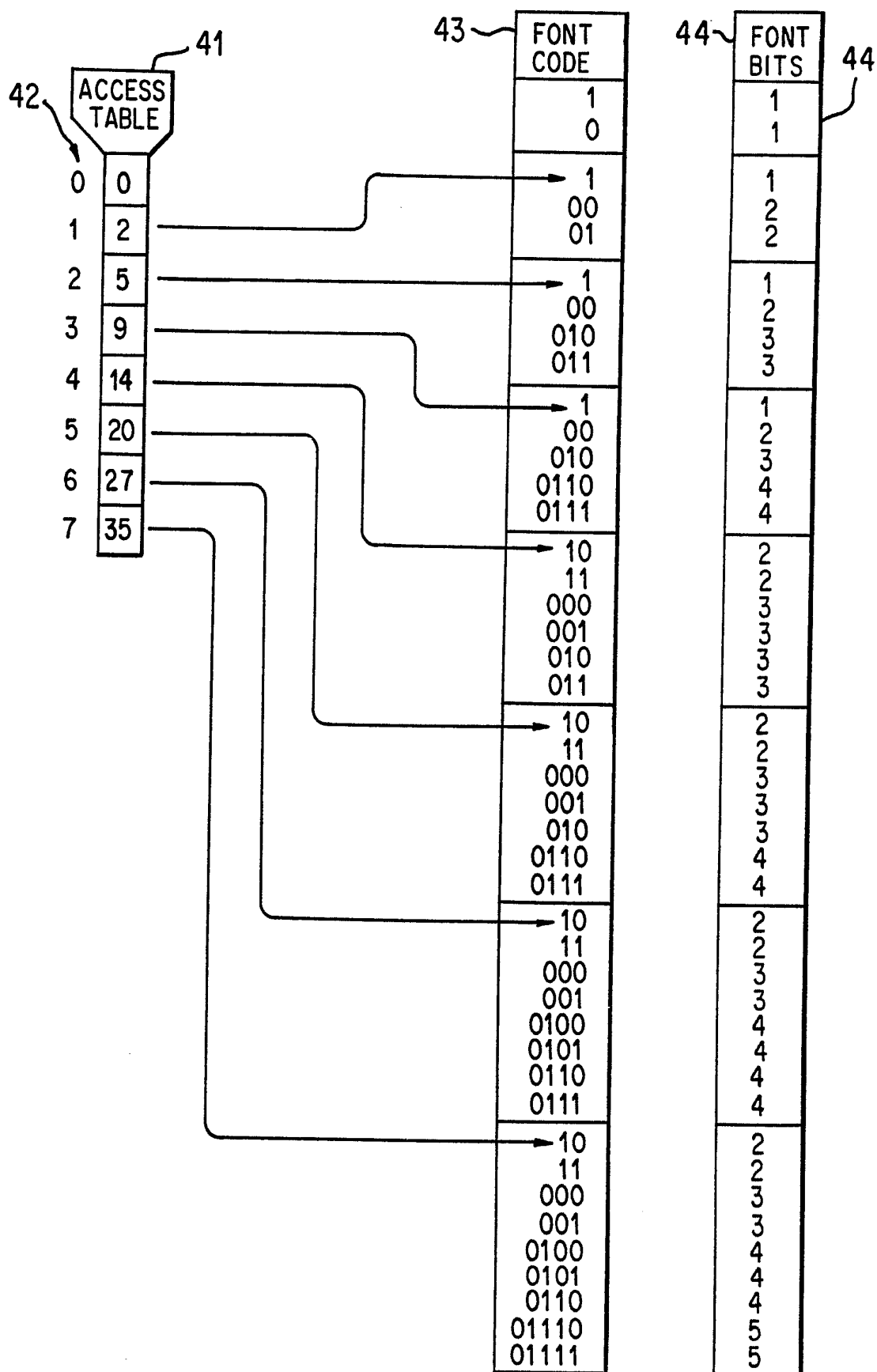


FIG. 4A

7/20

1100
1101
1110
1111
1000
.
.
.
.
00000011 11101
00000011 11110
00000011 11111

FIG. 4B

1
011
010
0011
00011
00010
00001
000001
000000
0010

FIG. 4C

11
101
100
.
.
.
.
.
.
.
.
0000010
0000001
0000000

FIG. 4D

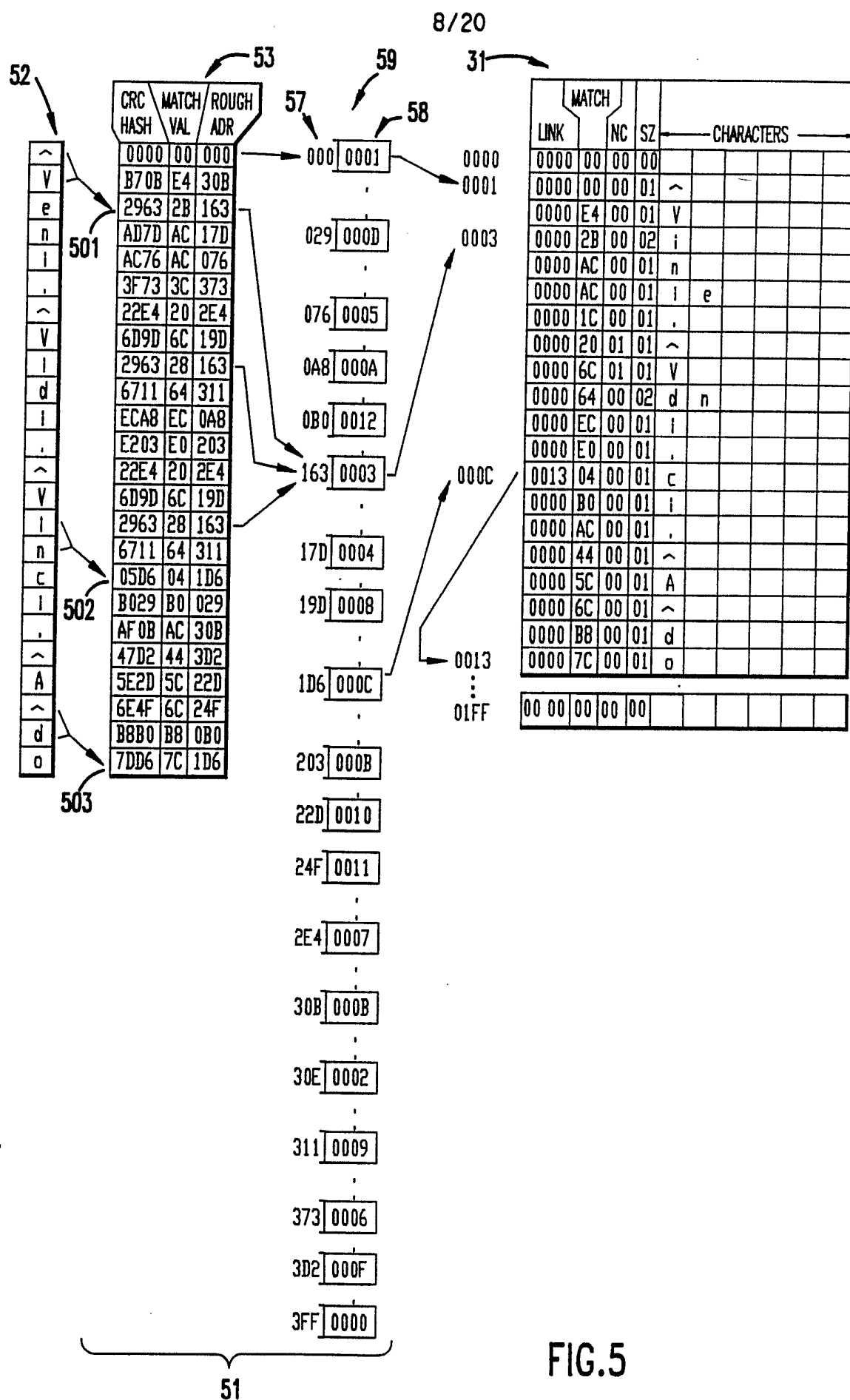


FIG.5

9/20

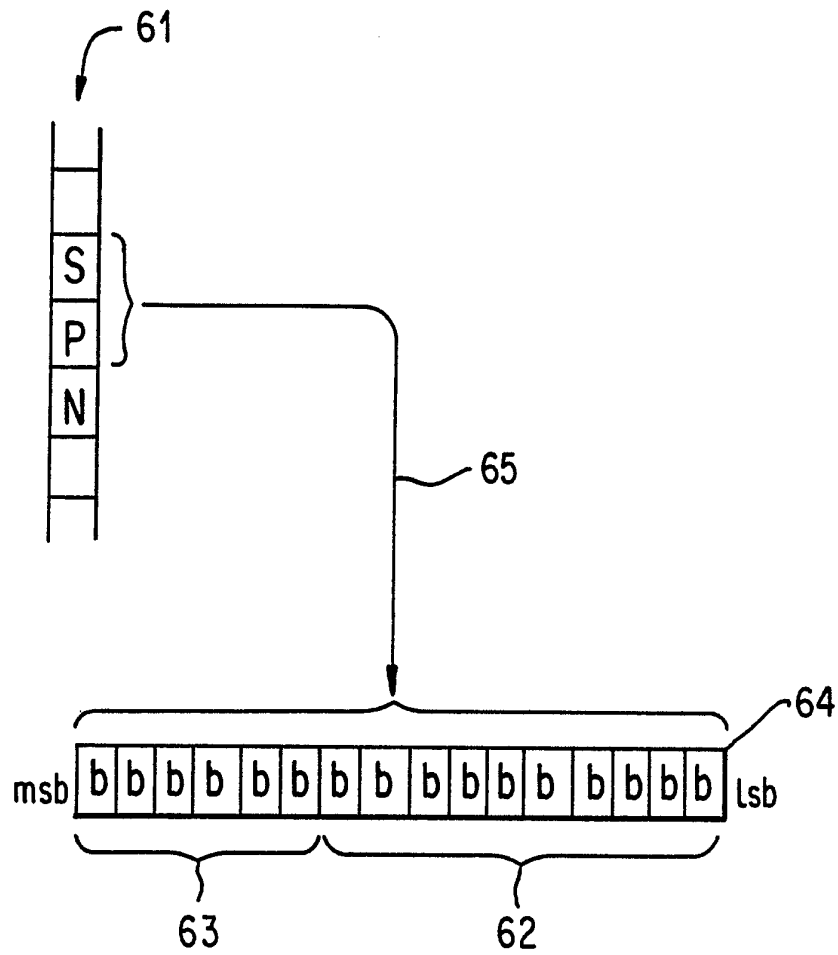


FIG. 6

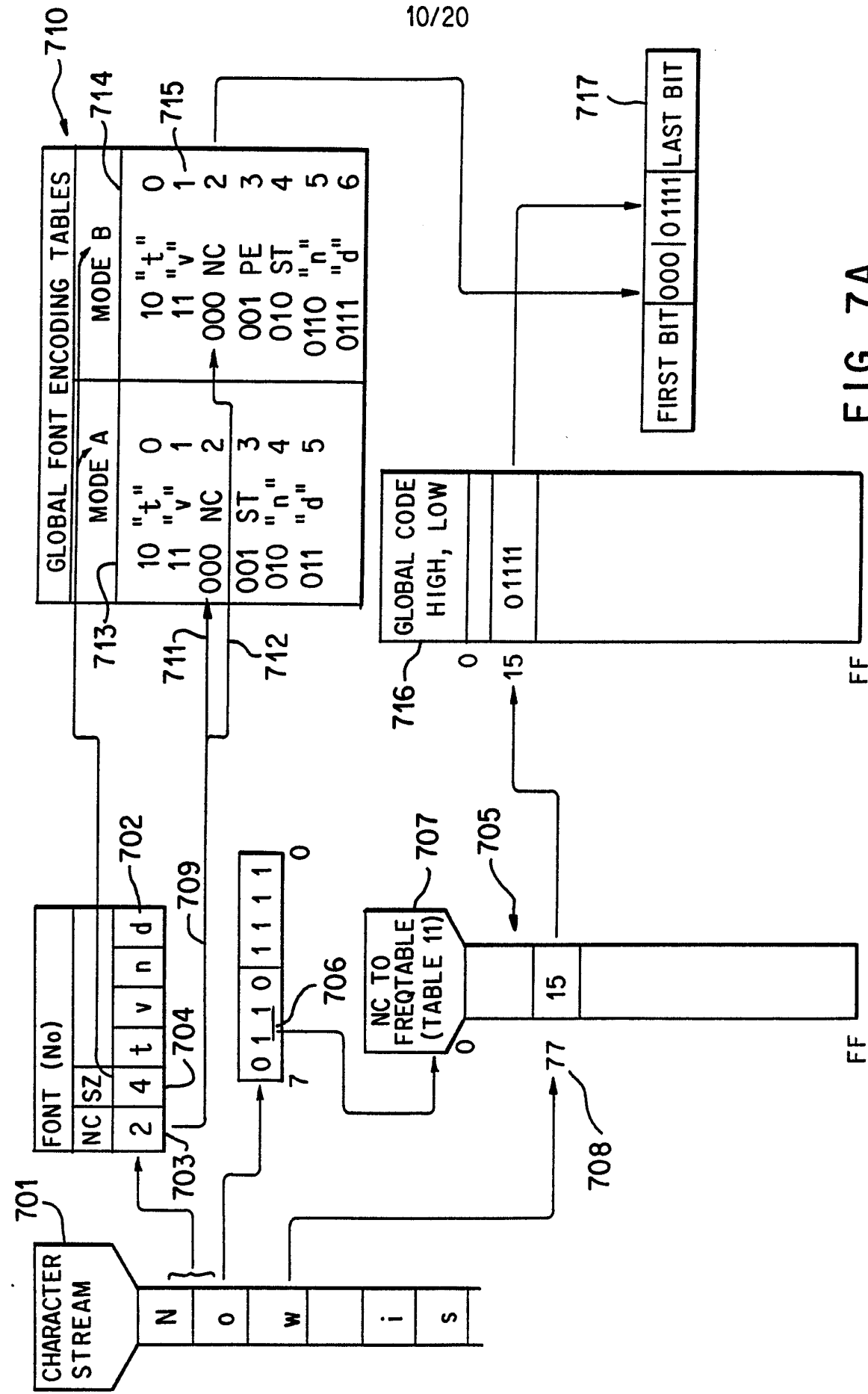
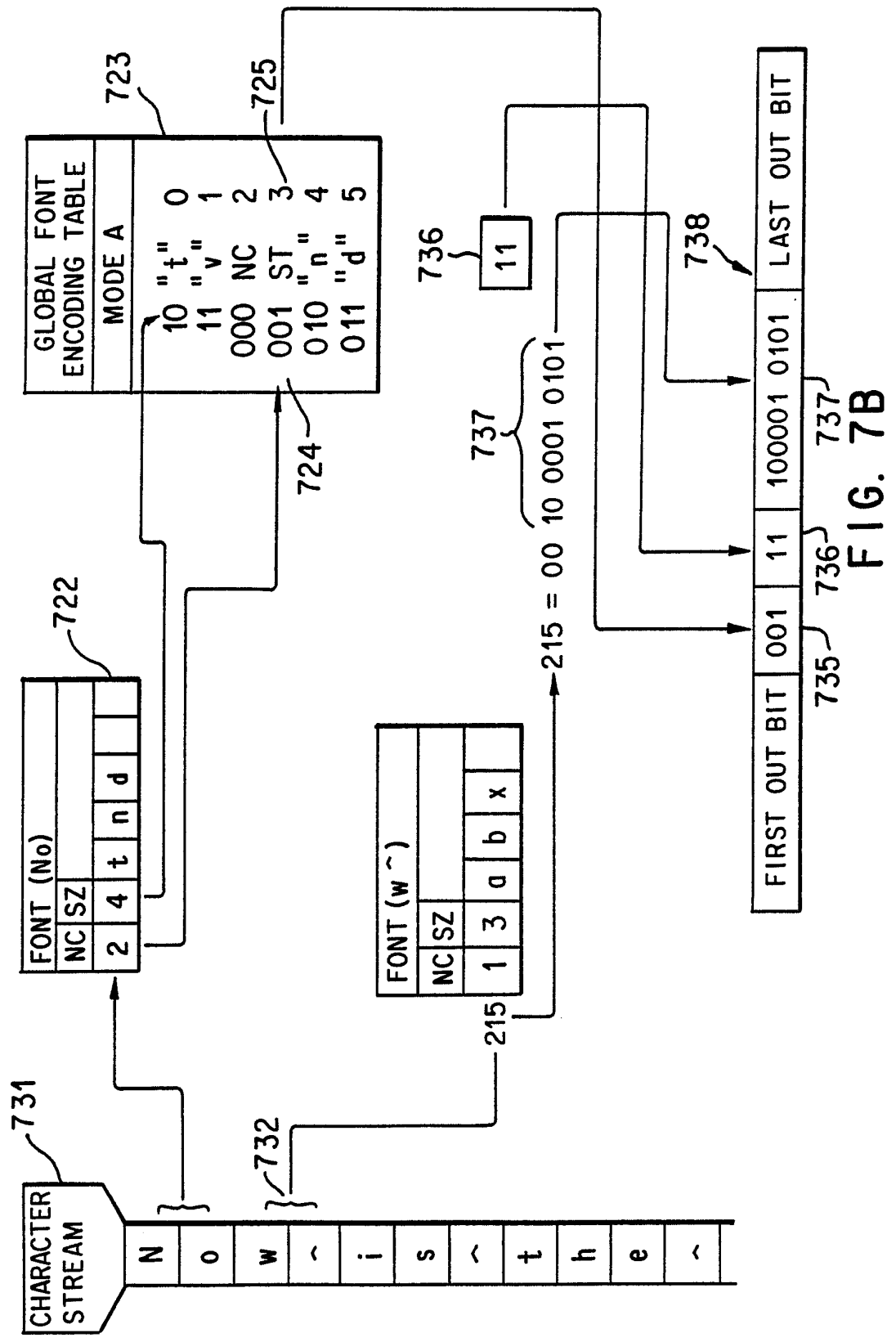
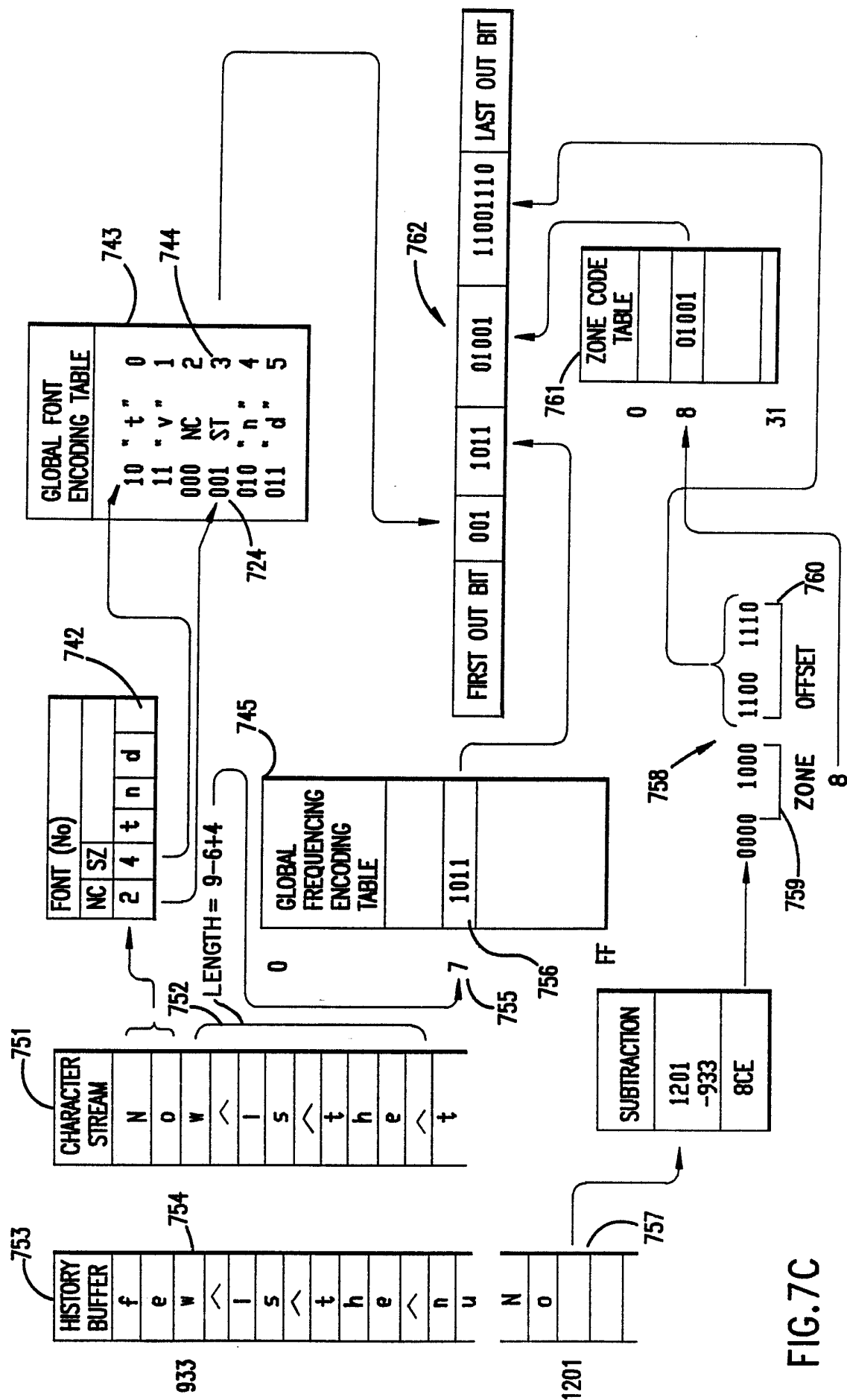


FIG. 7A





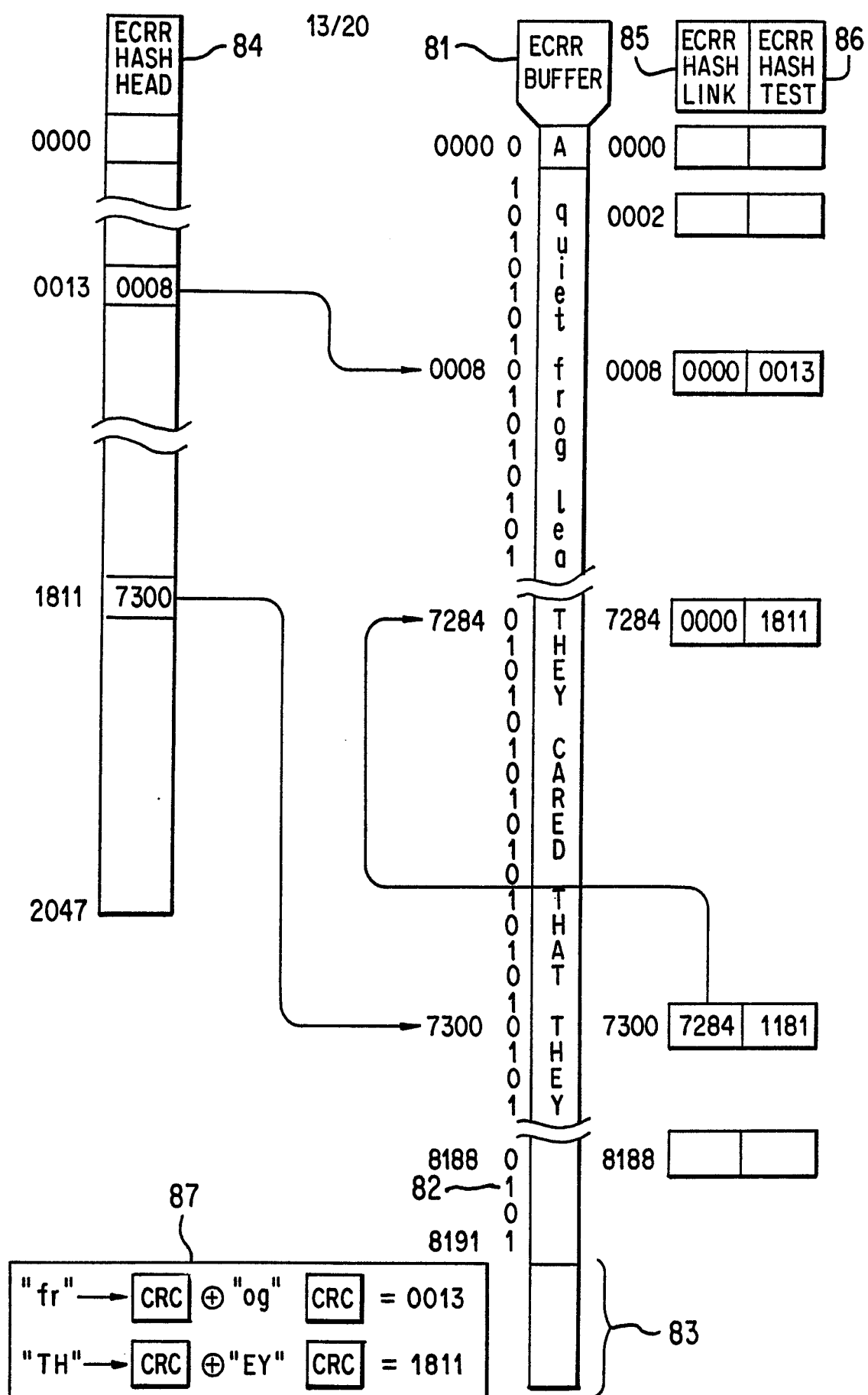


FIG. 8A

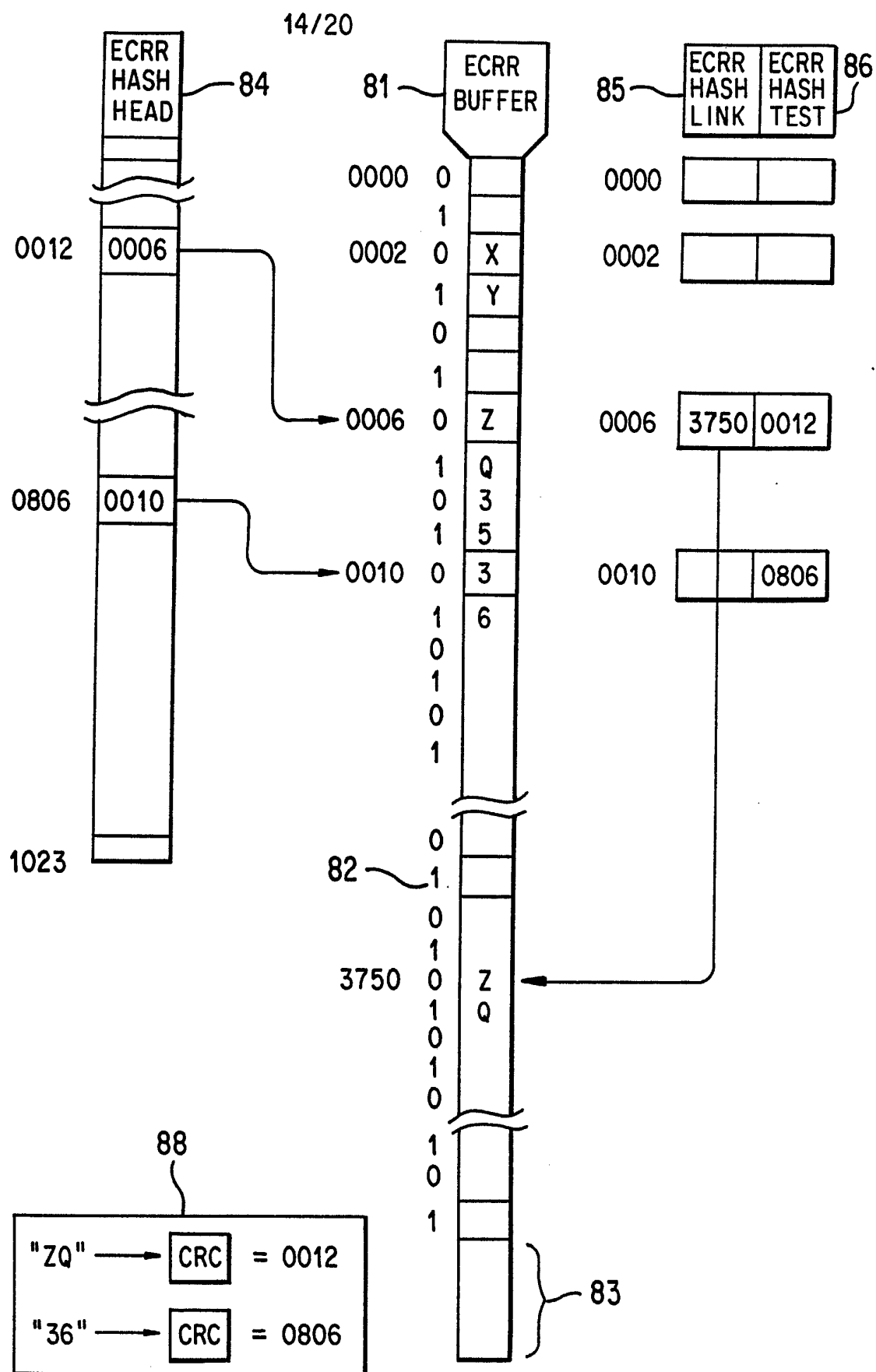


FIG. 8B

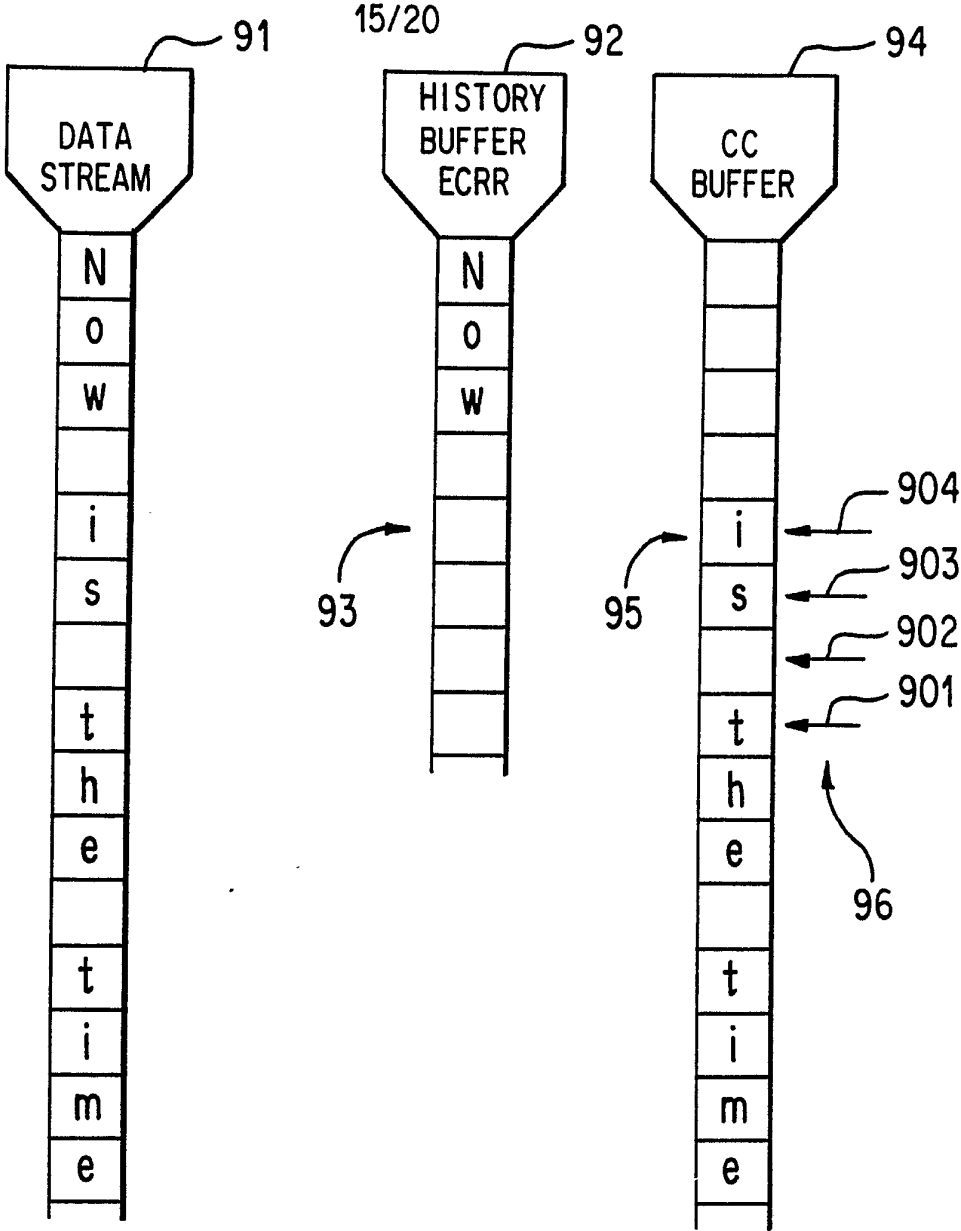


FIG. 9

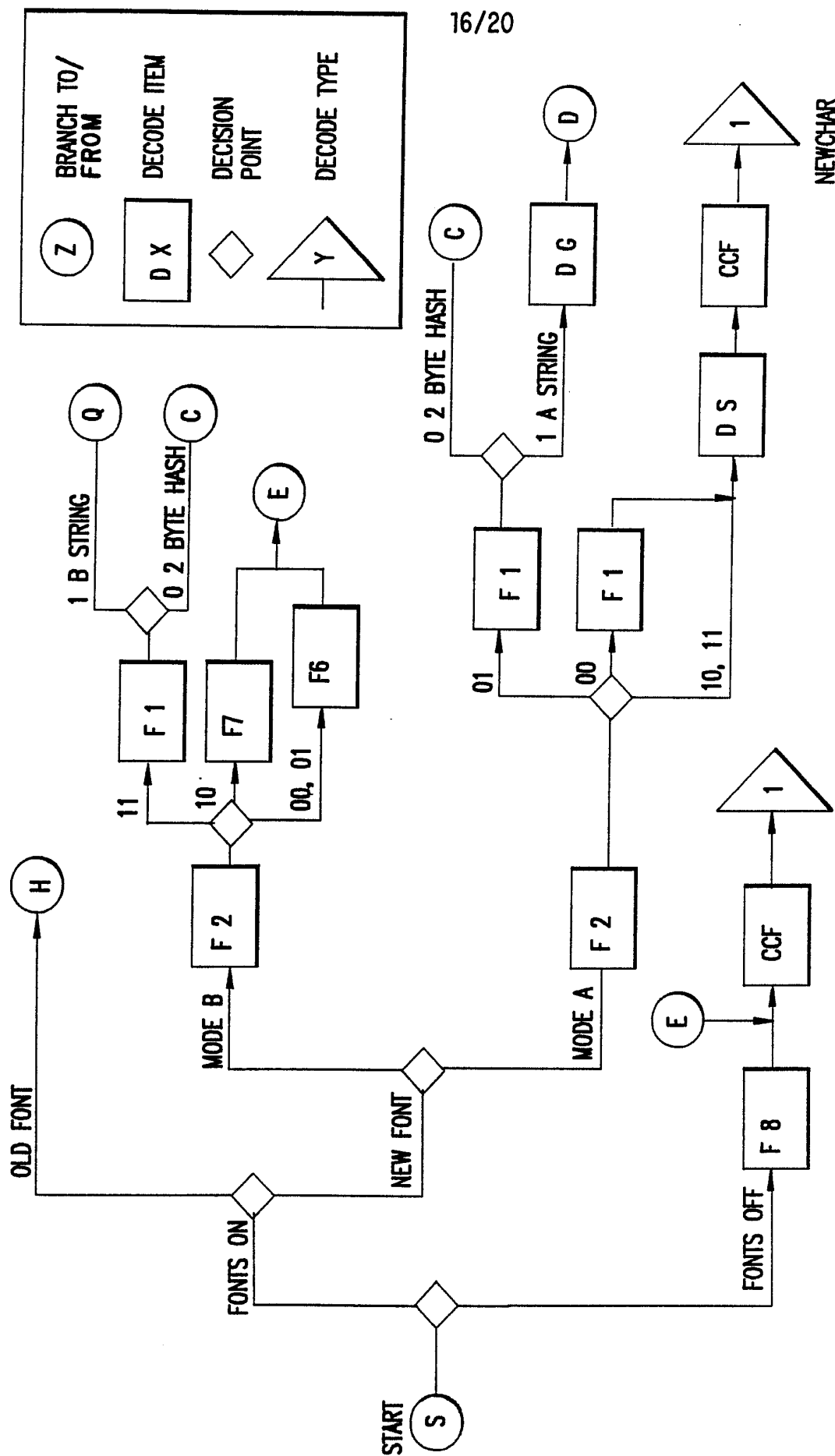
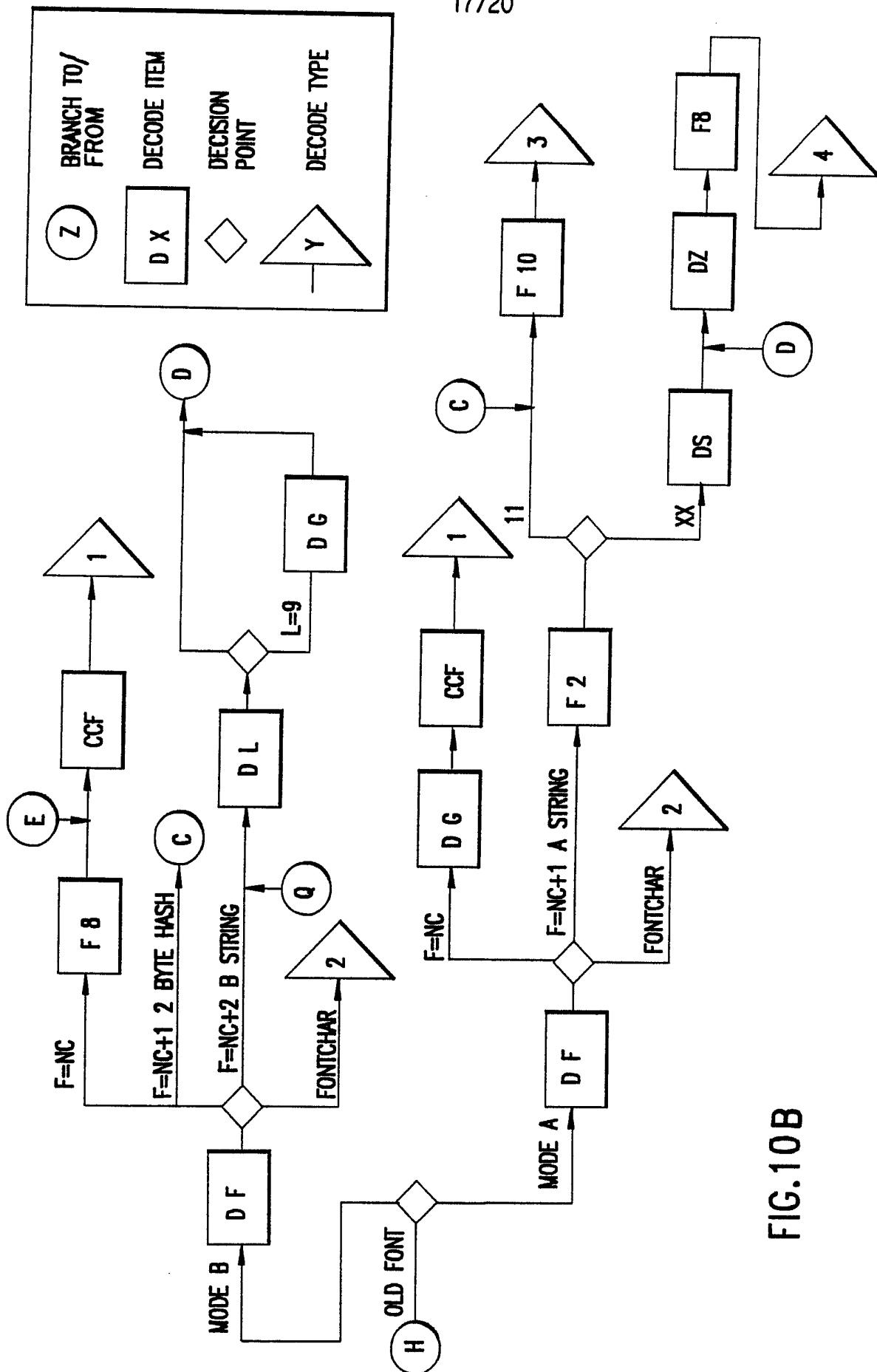


FIG.10A



18/20

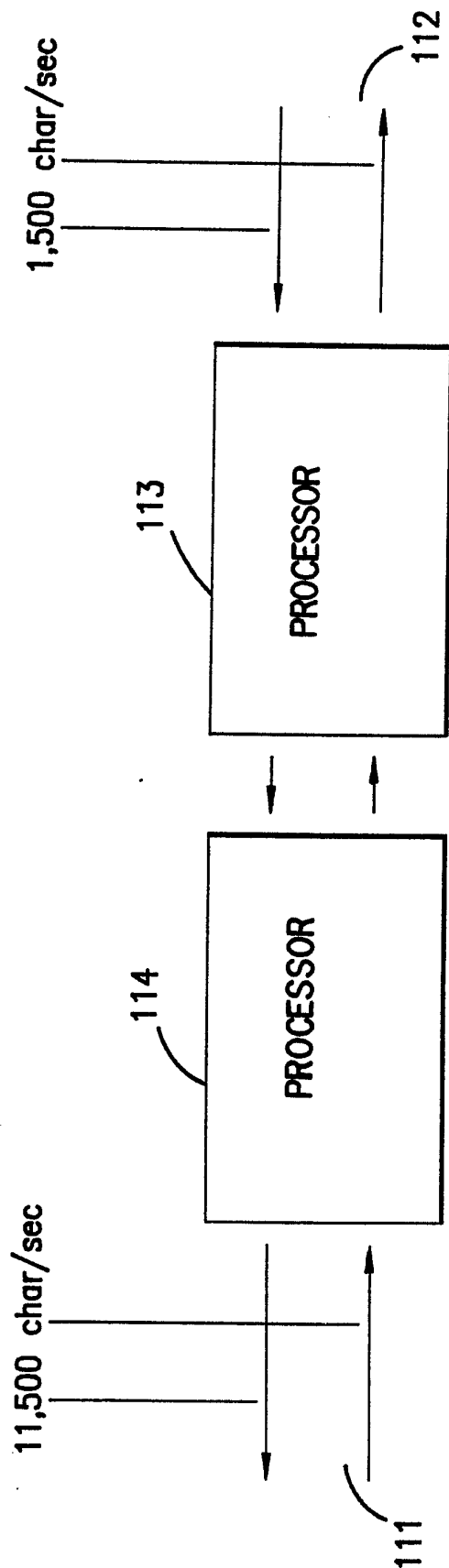


FIG.11

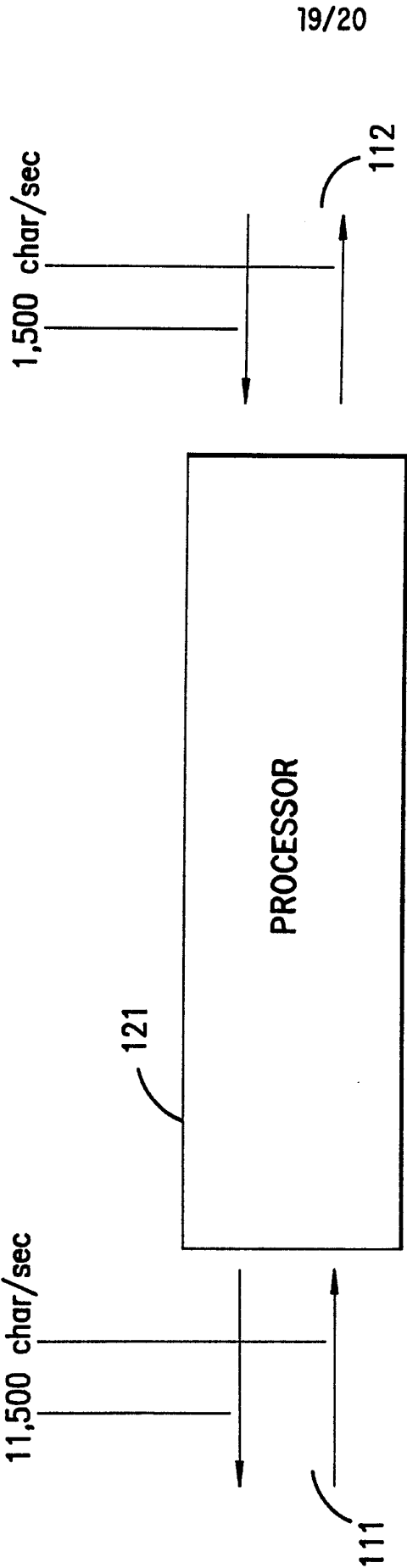


FIG.12 (PRIOR ART)

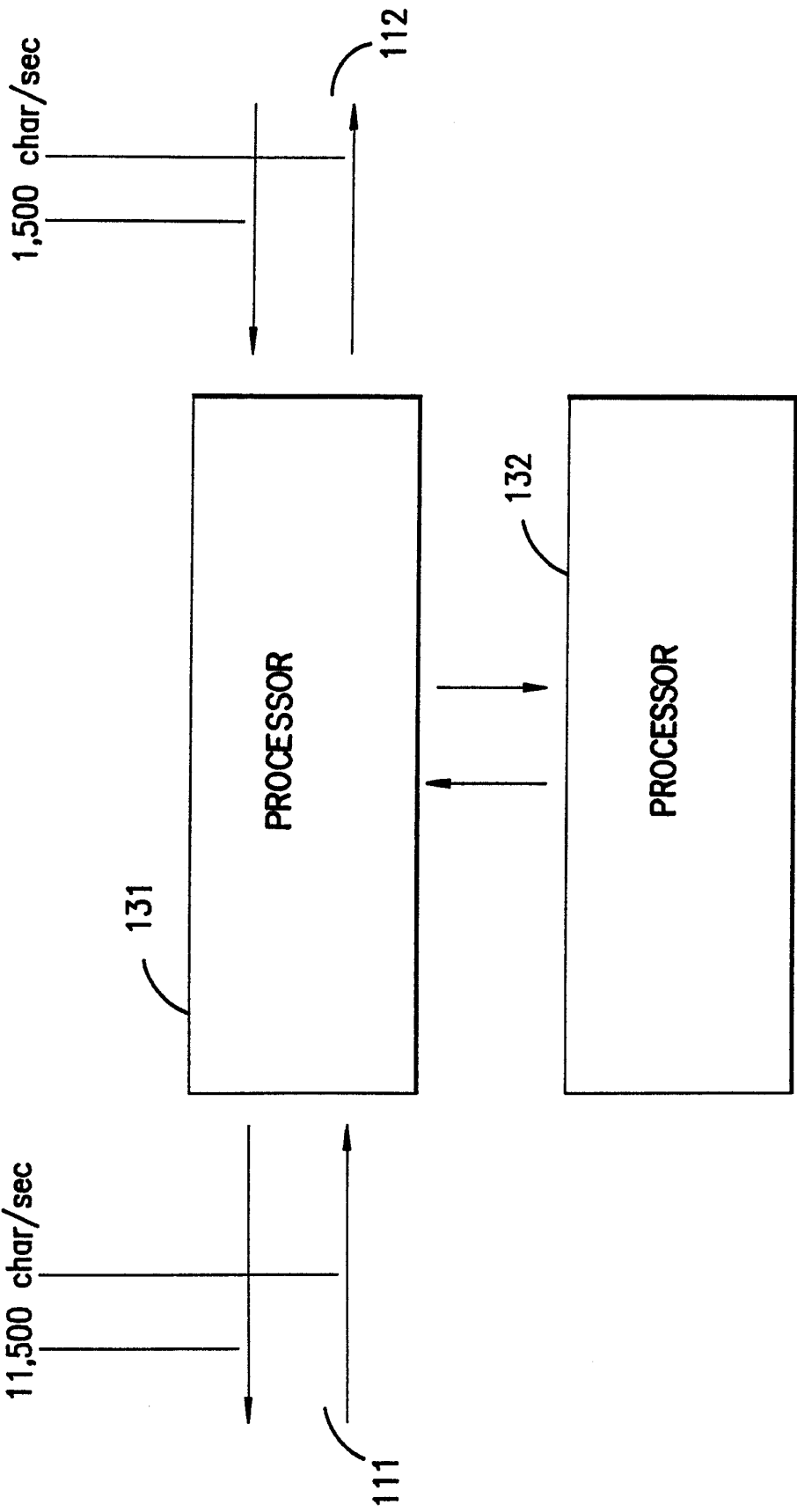



FIG.13 (PRIOR ART)

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 91/05659

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) ⁶		
According to International Patent Classification (IPC) or to both National Classification and IPC		
Int.Cl. 5 H03M7/42; H03M7/30; H03M7/48		
II. FIELDS SEARCHED		
Minimum Documentation Searched ⁷		
Classification System	Classification Symbols	
Int.Cl. 5	H03M	
Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched ⁸		
III. DOCUMENTS CONSIDERED TO BE RELEVANT⁹		
Category ¹⁰	Citation of Document, ¹¹ with indication, where appropriate, of the relevant passages ¹²	Relevant to Claim No. ¹³
A	<p>MINI MICRO SYSTEMS. vol. 21, no. 2, February 1988, BOSTON US pages 77 - 81; BACON: 'How to quadruple dial-up communications efficiency' see page 79, middle column, last paragraph - page 81, right column, last paragraph; figure 2</p> <p>---</p> <p>US,A,4 730 348 (MAC CRISKEN) 8 March 1988</p> <p>see column 6, line 50 - column 14, line 49; figures 3-7</p> <p>---</p>	<p>1,3,4,38</p> <p>1,3,4, 12,16, 17,38,42</p>
<p>¹⁰ Special categories of cited documents:</p> <p>"A" document defining the general state of the art which is not considered to be of particular relevance</p> <p>"E" earlier document but published on or after the international filing date</p> <p>"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)</p> <p>"O" document referring to an oral disclosure, use, exhibition or other means</p> <p>"P" document published prior to the international filing date but later than the priority date claimed</p> <p>"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention</p> <p>"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step</p> <p>"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.</p> <p>"&" document member of the same patent family</p>		
IV. CERTIFICATION		
Date of the Actual Completion of the International Search		Date of Mailing of this International Search Report
04 DECEMBER 1991		27. 12. 91
International Searching Authority		Signature of Authorized Officer
EUROPEAN PATENT OFFICE		FEUER F.S. 

US 9105659
SA 51372

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information. 04/12/91

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US-A-4730348	08-03-88	None	
