Logic Manual

Manufacturing Information Systems
CODE CONVERT

SYSTEM TEN BY SINGER

THIS MANUAL SUPERSEDES
ALL PRELIMINARY VERSIONS

SINGER
FRIDEN DIVISION

This module has been designed as an interdependent part of Manufacturing Information Systems, but its modularity will allow it to be used in other software systems where code conversion is necessary.

The manual assumes reader familiarity with general System Ten concepts. It is expected that the reader will be responsible for interfacing a specific set of MIS modules with the programming needs of a particular installation.

Determination of one's code conversion needs requires functional analysis of how the code is to be handled within the CPU, as well as recognition of the differences between input code and output code. Such an analysis might include the following points:

1. What type of code is generated by the input terminal? (MIS models 100 and 105 generate System Ten USASCII code; CD-30 terminals generate 8-bit code that is truncated upon entry through the IOC and is read in 6-bit form.)

2. Are there any elements of message processing that will require the System Ten CPU to analyze message content? (Input terminal partition programs may need to interpret data at some fixed point in all messages -- to determine terminal identification number, for example.)

   a. If data will require such interpretation, a translation sequence (and specifically associated table and constants) may need to be constructed. Code resulting from this should be in System Ten USASCII.

   b. If the code is already in System Ten USASCII, translation for this function is not necessary.

3. What type of code is required as output code? A translation sequence (and specifically associated table and constants) must be constructed for each differently coded portion of the message.

Once an installation's code conversion needs have been defined, it is the user's responsibility to write an interfacing program that will accurately call the actual code convert program. This document provides a listing of the code conversion process and a detailed functional explanation of the routine and its associated constants; it also explains in detail how to construct the calling program -- constants and buffers required, and instructions necessary to branch to the conversion program. The reader is especially urged to become thoroughly familiar with the process by which the index pointer is calculated, and with the use of the three index registers.

The Code Conversion Module is a multi-purpose code translation program which is used to translate code of one type into code of another type. Through correct referencing, it can handle code generated by either CD-30 (truncated System Ten USASCII) or MIS (System Ten USASCII) terminals, and translate that code on a character-by-character basis into output code that can be written in single-frame mode (such as USASCII or BCD) or double-frame mode (such as EBCDIC).

Possible input code	Possible output code
CD-30 (truncated System Ten USASCII)	USASCII
	IBM-BCD
	GE-BCD
MIS (normal System Ten USASCII)	EBCDIC

(Note: if the installation has only MIS terminals, and the output code desired is USASCII, this module is not needed. MIS terminals generate data in System Ten code, which is a subset of USASCII; hence any calls to the Code Conversion program are superfluous.)

The user supplies three addresses: the first location of the input area, the first location of the output area, and the starting location of the table used to make the required conversion. He also supplies the conversion table and the length of the character string to be converted during this translation.

The program operates by focusing on an index-referenced character in the input area, computing another index value equal to the binary value of that character (A=100001=33) and adding that value (or double that value, for double frame) to the table address. This method, which can be called relative addressing, results in a correctly translated character being placed in the proper output location.

This process works whether the desired output code is to be written in single-frame mode or double-frame mode. If the output is to be double frame, the relative table address will be to the first of two characters, which are both placed in the proper output location.

HARDWARE REQUIREMENTS

There is no hardware specifically related to this module.

CORE REQUIREMENTS AND ALLOCATION

The needs of the specific MIS application will determine the total amount and allocation of core for this module. Depending on the particular installation, the tables and code conversion routine can be in Common and the calling sequence in the input terminal partition, or the calling sequence can be in any partition where a message being processed will be accessible. Normally, the most efficient use of core will be to store the code conversion table(s), constants, and code conversion routine in Common, and the input/output buffers, terminal constants, and calling sequence(s) in partition.

For applications where only a single-character code conversion is required (CD-30 to System Ten USASCII, or Systen Ten USASCII to BCD, for instance), this module will normally need a minimum of 408 locations in Common (including a single 64-character table) and 350 locations in each input terminal partition.

For applications where only a double-character conversion is required (System Ten USASCII from model 100 and 105 terminals to EBCDIC, for example), this module will normally need 476 locations in Common (including a single 128-character table) and 790 locations in each terminal partition.

Core requirements and allocation for combinations of single- and double-character conversion will depend on the nature of the combination and the interfacing programming.

Calling program

The calling program resides in an input terminal partition. Code coming from the input terminal that needs any analysis by the System Ten CPU (other than character counts) must be in System Ten internal USASCII. If the code is from a CD-30 terminal, this requires a single-character code conversion, in addition to whatever conversion might be necessary for conversion into output code. If the code is from a MIS model 100 or 105 terminal, the only translation necessary will be into output code. (In a CD-30 message, for instance, for the conversion into System Ten USASCII these characters may contain the transaction code and terminal I.D.; the transaction code is used by the input terminal module to check for wrong-length messages, while the terminal I.D. and/or transaction code can be used for transaction and terminal validation as well as message routing.)

The number of characters to be translated are specified by CCSIZE, then the calling sequences are set up according to the need. The process should be repeated for a new string of characters or a different translation.

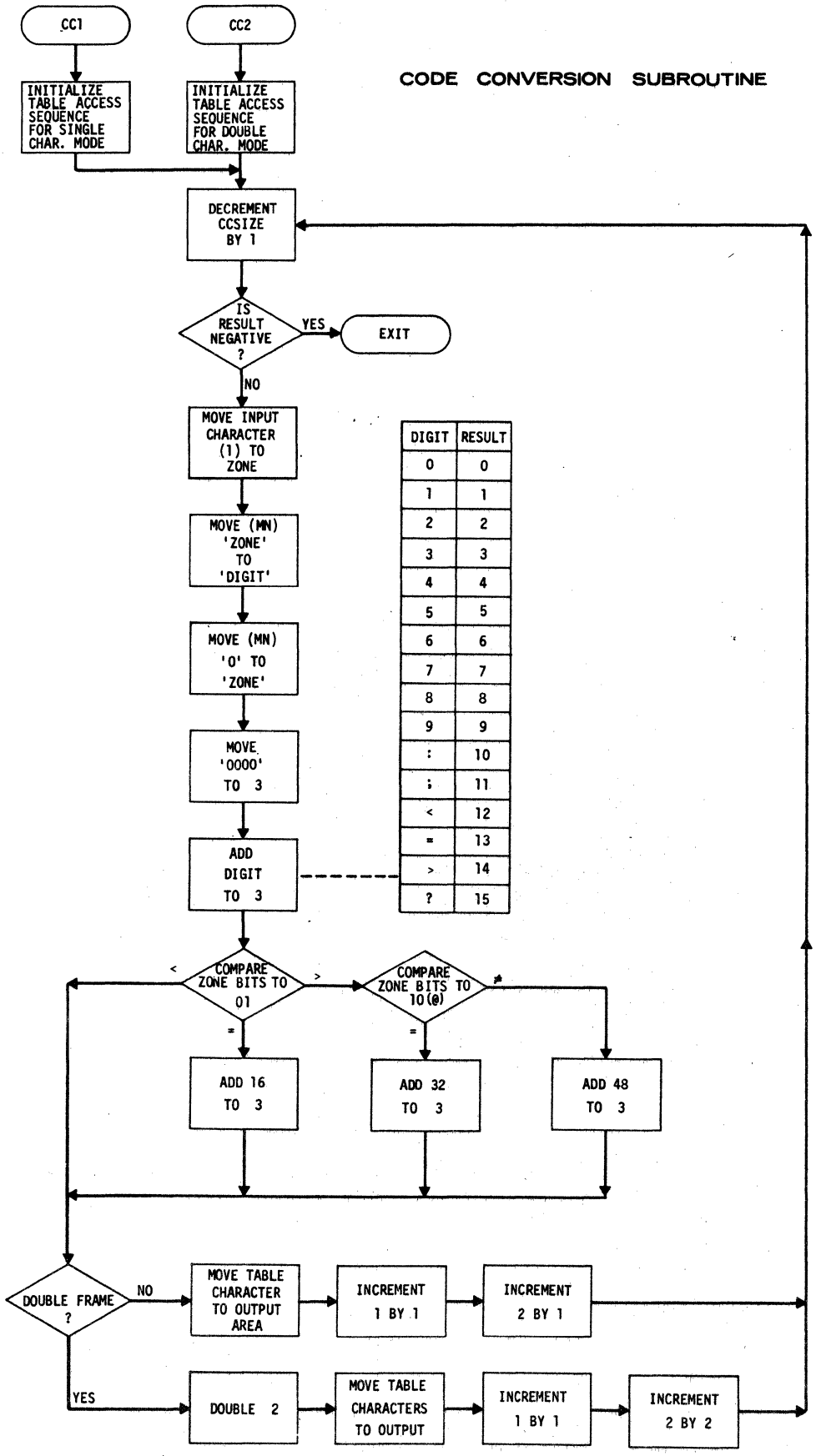
Code Conversion Program

The table access routine facilitates converting the message, one character at a time, until the message string is exhausted; it is also responsible for computing the index pointer which accesses the conversion table at the correct point for each character. To do this, the program examines the bit structure of each character, using bits 7 and 5 as zone bits to reference a line of the table, and bits 4 through 1 as numeric bits to reference a particular position in that line. Then, depending on whether the conversion is to be to single-character or double-character code, the program places the correct one or two characters in the appropriate location in the output area.

```
       TITLE 'CODE CONVERT'
*****************************************************************
*
*      CODE CONVERT TABLES
*
       COMMON
       ORG 320C
*
*****************************************************************
*                                                               *
* CODE CONVERSION SUBROUTINE                                    *
*                                                               *
* FOR SINGLE CHARACTER OUTPUT, ARGUMENTS AND CALLING SEQUENCE ARE:*
*                                                               *
*      TBL1    DM    C64                 64 CHARACTER TABLE     *
*      SIZE1   DM    N3'006'             3 CHARACTER LENGTH (1-999)*
*      IADDR1  DM    A'    '             HIGH ORDER ADDR OF INPUT *
*      OADDR1  DM    A'    '             HIGH ORDER ADDR OF OUTPUT*
*      TADDR1  DM    A'TBL1'             HIGH ORDER ADDR OF TABLE *
*                                                               *
*              MC    SIZE1,CCSIZE        SET LENGTH             *
*              MC    IADDR1,X1           SET REG1 TO INPUT ADDR *
*              MC    OADDR1,X2           SET REG2 TO OUTPUT ADDR*
*              MN    TADDR1,CC8+1        SET A=ADDR TO TABLE    *
*              BC    CC4+1(6),CC1(5)     BRANCH AND LINK        *
*                                                               *
* FOR DOUBLE CHARACTER OUTPUT, ARGUMENTS AND CALLING SEQUENCE ARE:*
*                                                               *
*      TBL2    DM    C128                128 CHARACTER TABLE    *
*      SIZE2   DM    N3'200'             3 CHARACTER LENGTH (1-999)*
*      IADDR2  DM    A'    '             HIGH ORDER ADDR OF INPUT *
*      OADDR2  DM    A'    '             HIGH ORDER ADDR OF OUTPUT*
*      TADDR2  DM    A'TBL2'             HIGH ORDER ADDR OF TABLE *
*                                                               *
*              MC    SIZE2,CCSIZE        SET LENGTH             *
*              MC    IADDR2,X1           SET REG1 TO INPUT ADDR *
*              MC    OADDR2,X2           SET REG2 TO OUTPUT ADDR*
*              MN    TADDR2,CC10+1       SET A=ADDR TO TABLE    *
*              BC    CC4+1(6),CC2(5)     BRANCH AND LINK        *
*                                                               *
*****************************************************************
 EJECT
*
* INITIALIZE TABLE ACCESS ROUTINE
*
CC1    MC CC8(35),CC7         SINGLE CHARACTER ENTRY
       BC CC3(5)
       ORG *-5
CC7ERO DM     C'0000'
CC7ONE DM     C                          ZONE
CC2    MC CC9(45),CC7         DOUBLE CHARACTER ENTRY
*
* EXIT FROM SUBROUTINE AT END OF FIELD
*
CC3    S  CC16(1),CCSIZE      DECREMENT INPUT FIELD SIZE
CC4    BC 0000(1),0000(0)
*
* COMPUTE INDEX FOR NEXT INPUT CHARACTER
*
       MC OP(1,1),CCZONE      ISOLATE ZONE & DIGIT BITS
```

```
        MN  CCZONE,CCDIGT       X
        MN  CCZERO(1),CCZONE    X
        MC  CCZERO(4),X3        COMPUTE INDEX
        A   CCDIGT,X3           X
        C   CCZONE,CCZERO       X
        BC  CC7(1),CC6(2)       X
        C   CCZONE,CCAT         X
        BC  CC5(2)              X
        A   CC48,X3             X
        BC  CC7(5)              X
CC5     A   CC32,X3             X
        BC  CC7(5)              X
        ORG *-5
CCAT    DM      C'@'                        COMMERCIAL AT SIGN
CC16    DM      N'16'                       TABLE INDEX
CC32    DM      N'32'                       X
CC6     A   CC16,X3             X
* MOVE TABLE CHARACTER TO OUTPUT AREA
*
CC7     DM      C45                         THIS AREA IS LOADED WITH
*                                           ONE OF THE FOLLOWING ROUTINES
*                                           UPON ENTRY TO SUBROUTINE
 EJECT
* SINGLE CHARACTER ROUTINE
*
CC8     MC  0000(1,3),000P(,2) MV CHAR FROM TABLE TO OUTPUT
        A   CC16(1),X1          INCREMENT INPUT POINTER
        A   CC16(1),X2          INCREMENT OUTPUT POINTER
        BC  CC3(5)              GO TO END OF FIELD TEST
*
* DOUBLE CHARACTER ROUTINE
*
CC9     A   X3,X3               DOUBLE INDEX
CC10    MC  0000(2,3),000P(,2) MV CHAR FROM TABLE TO OUTPUT
        A   CC16(1),X1          INCREMENT INPUT POINTER
        A   CC32+1(1),X2        INCREMENT OUTPUT POINTER
        BC  CC3(5)              GO TO END OF FIELD TEST
*
* CONSTANTS AND WORK AREAS
*
        ORG     *-5
CCSIZE  DM      N3                          LENGTH OF INPUT FIELD
CCDIGT  DM      C'0'                        DIGIT
CC48    DM      N'48'                       X
        NORMAL
*
*         THESE INDEX REGISTERS ARE TO GO IN THE SAME PARTITION AS THE
          CALLING SEQUENCES IN THE INPUT TERMINAL PROGRAM.
*
        ORG   0011
X1      DM    C4            INDEX REGISTER 1
        ORG   0021
X2      DM    C4            INDEX REGISTER 2
        ORG   0031
X3      DM    C4            INDEX REGISTER 3
        END
```

CODE CONVERSION SUBROUTINE



Loading

Use any System Ten loading program as appropriate for the installation. The tables, table access routine, and associated constants will normally be loaded into Common, unless there is room to duplicate them and place them in each input terminal partition. If the table is in partition, the A address referred to in CC8 (and/or CC10) must be followed by a P to indicate that the address is in partition rather than Common. The user's calling program and its associated constants should be loaded into the appropriate partition. (The tables must either all be in Common or all in partition.)

ORGs

The Common portion of this program may be ORGed at any location that does not conflict with other specified areas. In determining where to place this programming, the reader is reminded to be cognizant of the order and placement of other MIS system constants.

In input terminal partition programming, it is usually convenient for the programmer to plan to use the three index registers to hold the input address, output address, and index pointer, respectively, in passes through this module, and to initialize them appropriately.

Operating Instructions

There is no operator intervention required for this module.

List and Explanation

The following are constants, buffers and storage areas defined by DM statements. They should not necessarily be loaded in the order presented here. An asterisk (*) indicates that the user should define the term to fit his own requirements.

GENERAL PURPOSE CONSTANTS

Label	DM	Explanation
CCZERO	C'0000'	Constant used for comparison.
CC16	N'16'	Used to increment X3 by value necessary to correctly address the table being used; also, first digit ('1') used to decrement character counter (CCSIZE).
CC32	N'32'	Used to increment X3 by value necessary to correctly address the table being used.
CC48	N'48'	Used to increment X3 by value necessary to correctly address the table being used.
CCAT	C'@'	"At" sign used in compare operation to determine which zone bits are on for this character; results in incrementing X3 by the value necessary to correctly address the table being used.

CONVERSION TABLES

Label	DM	Explanation
*TBL1	0C64 (See below)	Table 1 (Single frame).
*TBL2	0C128 (See below	Table 2 (Double frame).

RELATIVE ADDRESSES

Label	DM	Explanation
*SIZE1	C3 (User defined)	Size (three digits, 001-999) of input character string.
*IADDR1	A4 (User defined)	Beginning address of input character string.
*TADDR1	A4 (user defined)	Address of Table 1.
*SIZE2	C3 (User defined)	Size (three digits, 001-999) of input character string, conversion using Table 2.
*IADDR2	A4 (User defined)	Beginning address of input character string, conversion using Table 2.
*OADDR2	A4 (User defined)	Beginning address of output area to be used when referencing Table 2.
*TADDR2	A4 (User defined)	Address of Table 2.

BUFFERS AND STORAGE AREAS

Label	DM	Explanation
*X1	C4	Index register 1; holds input address.
*X2	C4	Index register 2; holds output address.
*X3	C4	Index register 3; holds pointer indicating where in a specified table the character's translated value is; also referred to as index pointer.
CCSIZE	N3	Counter used to keep track of number of characters being converted; successively decremented until it is a negative value, then program exits.
CCZONE	N1	Storage area for input character's zone bits. Numeric bits are set at '0000'.
CCDIGT	C'0'	Storage area for input character's numeric bits. Zone bits of CCDIGT are fixed at '01'.

SPECIAL DEFINITIONS FOR THIS DOCUMENT

Zone bits = bits 7 and 5 for any character.

Numeric bits = bits 4 through 1 for any character.

System Switches

None of the MIS system switches are referenced or modified by the instructions of this module.

Conversion Tables

The two tables referenced in this module are used to translate code into single frame GE-BCD and double frame EBCDIC.

1. Table 1 has been set up so that a single-character translation from System Ten USASCII into GE-BCD code will take place when the index pointer is correctly calculated.

Table 1 (64 characters)

```
'0←↑+K\:O=MLP[J;Q !"#$%&''()-N>]/'

',123456789ABCDEFGHIRSTUVWXY*?<@Z'
```

The single quote mark following the ampersand in the first line must be modified when assembled. (The assembler requires two single quote marks so that it can distinguish between a quote mark used as a delimiter and a quote mark used as a character.)

The table is addressed by using an index pointer computed from the numeric and zone bits of the input character and added to the starting address of the table, thus giving the location of the translated character. The character at this address is placed in the output area (with single-character conversions, the input and output areas can be the same).

2. Table 2 has been set up so that a double-character translation from System Ten USASCII into EBCDIC will take place when the index pointer is correctly calculated.

Table 2 (128 characters)

```
'04J5O7K7K5L605M7M4M5L5N4K606K416'

'0O1O2O3O4O5O6O7O8O9OJ7N5L4N7N6O6'

'L71L2L3L4L5L6L7L8L9L1M2M3M4M5M6M'

'7M8M9M2N3N4N5N6N7N8N9NM4N6M5L4M6'
```

Note: for this document, zero is represented as 0, the letter as O.

This table is addressed by doubling the index pointer computed from the binary representation of the System Ten character, and adding this value to the starting address of the table, thus giving the beginning address of the translated character. Since this conversion is to be from one character into two, this character and the one following it must both be placed in the output area -- which must be separate from the input area so that there will be sufficient room for the output string.

For example, suppose the character we wish to translate has a System Ten binary value of 011001 (the character is 9). The code conversion program determines the index pointer by isolating zone and numeric bits as described in the next section, then it picks up the 50th and 51st characters (90) and places them in the output area.

Computing the Index Pointer

The index pointer for referencing the code conversion table is computed in index register 3 (X3). The computation process involves isolating the zone bits (bits 7 and 5) and the numeric bits (bits 4 through 1) for each character.

Zone bits of 00 mean one of the addends to X3 will be 0; zone bits of 01 mean this addend will be 16; zone bits of 10 mean this addend will be 32; zone bits 11 mean this addend will be 48. The decimal equivalent of the numeric bits for this character form the other addend to X3, such that this index pointer now represents a given number of positions in a table. If this is to be a single-character conversion, X3 is left at this value; if it is to be a double-character conversion, X3 is doubled. The resulting value in X3 is used as an address relative to the beginning location of the conversion table being used.

Since the process by which the value in the index pointer (X3) is calculated is the same for all characters, it then becomes a relatively simple task to set up the conversion tables such that, when the table is accessed , the index pointer will indicate precisely where in the table the correctly translated character or characters will be.

The length modifier on the Move Character instruction that moves the correct character into the output area is specified as 1 for single-character conversions, or 2 for double-character conversions. Therefore, in a double-character conversion, the correct combination of two characters must begin at the location which will be specified by the value in the index pointer (X3). Regardless of the type of output code desired, or the number of conversions to take place within a program, organization of a conversion table depends on the proper calculation of the index pointer, the process being the same for all tables. This requires that the bit structure of any output code must increment in the same order as the bit structure of the input code.

The programmer should keep in mind that the whole purpose of a conversion table is to convert characters at the bit structure level. Thus it becomes his task to use the table as a translation medium, so that an incoming character's bit structure, when used to calculate the index pointer, will correctly reference a System Ten USASCII character (or pair of characters) that will produce the desired output bit structure, regardless of the type of output code desired.

SYSTEM TEN CHARACTER CODE		TO WRITE BCD EQUIVALENT		TO WRITE GE EQUIVALENT	
SP	000000	0	010000	0	010000
!	000001	J	101010	←	111111
"	000010	SP	000000	↑	111110
#	000011	+	001011	+	001011
$	000100	K	101011	K	101011
%	000101	<	011100	\	111100
&	000110	P	110000	:	011010
'	000111	,	001100	Ø	101111
(	001000	<	011100	=	011101
)	001001	\	111100	M	101101
*	001010	L	101100	L	101100
+	001011	P	110000	P	110000
,	001100	;	011011	[	111011
-	001101	@	100000	J	101010
.	001110	[	001110	;	011011
/	001111	1	010001	Q	110001
0	010000	*	001010	SP	000000
1	010001	!	000001	!	000001
2	010010	"	000010	"	000010
3	010011	#	000011	#	000011
4	010100	$	000100	$	000100
5	010101	%	000101	%	000101
6	010110	&	000110	&	000110
7	010111	'	000111	'	000111
8	011000	(	001000	(	001000
9	011001	)	001001	)	001001
:	011010	-	00y101	-	001101
;	011011	N	101110	N	101110
<	011100	↑	111110	>	011110
=	011101	+	001011	]	111101
>	011110	.	001110	.	001110
?	011111	Z	111010	/	001111
@	100000	,	001100	,	001100
A	100001	Q	110001	1	010001
B	100010	R	110010	2	010010
C	100011	S	110011	3	010011
D	100100	T	110100	4	010100
E	100101	U	110101	5	010101
F	100110	V	110110	6	010110
G	100111	W	110111	7	010111
H	101000	X	111000	8	011000
I	101001	Y	111001	9	011001
J	101010	A	100001	A	100001
K	101011	B	100010	B	100010
L	101100	C	100011	C	100011
M	101101	D	100100	D	100100
N	101110	E	100101	E	100101
Ø	101111	F	100110	F	100110
P	110000	G	100111	G	100111
Q	110001	H	101000	H	101000
R	110010	I	101001	I	101001
S	110011	2	010010	R	110010
T	110100	3	010011	S	110011
U	110101	4	010100	T	110100
V	110110	5	010101	U	110101
W	110111	6	010110	V	110110
X	111000	7	010111	W	110111
Y	111001	8	0/1000	X	111000
Z	111010	9	011001	Y	111001
[	111011	]	111101	*	001010
\	111100	>	011110	?	011111
]	111101	M	101101	<	011100
↑	111110	SP	000000	@	100000
←	111111	SP	000000	Z	111010

FIGURE 1

To read this chart, start with the System Ten USASCII character and its bit structure in the first column. The bit structure for this character in either IBM or GE-BCD is indicated at the same level in columns 2 and 3. The character on the left in each of these two columns is the System Ten USASCII character which has this bit structure.

For instance, suppose a character coming from the Job Information Station (which sends messages in System Ten USASCII) is an A, with a bit structure of 100001, and that the desired output code is GE-BCD. The task of the programmer is to design his conversion table such that, when the index pointer is calculated in the normal fashion, the character pointed to has a bit structure 010001, equivalent to a GE-BCD character A. This happens to be a System Ten USASCII digit 1. See Figure 1.

As another example, suppose we have a System Ten USASCII character 4, which we wish to translate into IBM-BCD. The bit structure of the character to be translated is 010100. To write an IBM-BCD 4, we need a character with a bit structure of 000100; in System Ten USASCII this is a $. Thus the index pointer would be calculated to point to a location 20 characters after the beginning address of the table (zone bits 01 mean the table level starting at relative location 16, plus 4 locations for the numeric bits), at which location is the System Ten USASCII character $.

Truncated CD-30 code presents the same sort of problem, since it comes into the System Ten CPU in 6-bit form. A programmer should remember, however, that in order to correctly analyze message content from the CD-30, the System Ten CPU must read those characters in its internal USASCII subset; however, other portions of the message can be directly translated into output code if content analysis is not needed at this point. See Figure 2.

The only difference in procedure in converting to double-frame code (EDCDIC) is that the index pointer will point to the first of two characters, both of which will be written in a combined form to produce the desired 8-bit EBCDIC character. The two characters will have a total of 12 bits, of which EBCDIC will need only 8; hence to get the proper 8 bits, the tape controller will drop the zone bits of both characters and write in double-frame mode, resulting in a single 8-bit character comprised of the numeric bits of the first character (low-order bits of the EBCDIC character) and the second character (high-order bits of the EBCDIC character). The ninth bit will be added to the character frame as either a 0 or 1, in accordance with parity declaration. The parity (ninth) bit is added by the tape or disc controller.

Functional Description

This module consists of two parts: the table access routine and user-supplied tables in Common, and the user-supplied entry from input terminal partition.

TO WRITE CD-30 EQUIVALENT		TO WRITE FROM CD-30 TO BCD		TO WRITE FROM CD-30 TO GE	
SP	000000	0	010000	0	010000
1	010001	!	000001	!	000001
2	010010	"	000010	"	000010
3	010011	#	000011	#	000011
4	010100	$	000100	$	000100
5	010101	%	000101	%	000101
6	010110	&	000110	&	000110
7	010111	'	000111	'	000111
8	011000	(	001000	(	001000
9	011001	)	001001	)	001001
T1	010001	!	000001	!	000001
NA		SP	000000	0	010000
NA		SP	000000	0	010000
T8	011000	(	001000	(	001000
NA		SP	000000	0	010000
NA		SP	000000	0	010000
0	010000	*	001010	SP	000000
/	001111	1	010001	Q	110001
S	110011	2	010010	R	110010
T	110100	3	010011	S	110011
U	110101	4	010100	T	110100
V	110110	5	010101	U	110101
W	110111	6	010110	V	110110
X	111000	7	010111	W	110111
Y	111001	8	011000	X	111000
Z	111010	9	011001	Y	111001
T3	010011	#	000011	#	000011
,	011011	;	011011	[	111011
NA		SP	000000	0	010000
T4	010100	$	000100	$	000100
NA		SP	000000	0	010000
NA		SP	000000	0	010000
-	001101	@	100000	J	101010
J	101010	A	100001	A	100001
K	101011	B	100010	B	100010
L	101100	C	100011	C	100011
M	101101	D	100100	D	100100
N	101110	E	100101	E	100101
Ø	101111	F	100110	F	100110
P	110000	G	100111	G	100111
Q	110001	H	101000	H	101000
R	110010	I	101001	I	101001
T2	010010	"	000010	"	000010
%	000101	¤	011100	\	111100
NA		SP	000000	0	010000
T6	010110	&	000110	&	000110
NA		SP	000000	0	010000
NA		SP	000000	J	101010
NA		SP	000000	0	010000
A	100001	Q	110001	1	010001
B	100010	R	110010	2	010010
C	100011	S	110011	3	010011
D	100100	T	110100	4	010100
E	100101	U	110101	5	010101
F	100110	V	110110	6	010110
G	100111	W	110111	7	010111
H	101000	X	111000	8	011000
I	101001	Y	111001	9	011001
T7	010111	'	000111	'	000111
.	001110	[	111011	;	011011
NA		SP	000000	0	010000
T5	010101	%	000101	%	000101
NA		SP	000000	0	010000
NA		SP	000000	0	010000

FIGURE 2

The first column of this chart shows the character that was sent by the CD-30 Terminals (if the CPU recognizes a character of bit structure 010010 as a 2, it started out in the CD-30 terminal as an S, with bit structure 00110010). Thus, the first column shows CD-30 truncated characters. Columns 2 and 3 show the bit structure for the System Ten character in IBM and GE-BCD. The character at the left of each of these two columns is the System Ten representation of the CD-30 character required to get this bit structure.

Table Access

CC1 and CC2 represent, respectively, the single-character and double-character entry points. In either case, the MC instruction results in a set of processing instructions (CC8 for single character, CC9 for double character) being moved into the processing area at CC7.

The instruction at CC3 uses the leftmost digit of CC16 -- in this case, 1 -- to decrement the counter in CCSIZE which tells how many characters are yet to be translated in this character string.

CC4's A operand holds the return address stored there by a Branch and Link instruction from the user's routine, which is used only if all characters for this stage have been translated; its B operand simply passes control on to the next instructions.

Then the routine isolates zone and numeric bits, by moving one character at a time from the input area into first CCZONE, then CCDIGT, fixing the numeric bits of CCZONE to 0000 and the zone bits of CCDIGT to 01. The numeric value is stored in CCDIGT: the bit configuration in CCZONE will then result in 0, 16, 32, or 48 being added to X3. X3 then holds a number representing a position in a conversion table where a particular character's translated value is to be found.

Next, the routine processes whatever instructions have been placed in the subroutine buffer area in CC7. If this is to be a single-character translation, the instructions starting at CC8 are there; if double character, those starting at CC9 are there. Either set operates the same way, except that, if this is to be double-character conversion, the value in X3 is doubled first. The character at the address indicated by X3 is moved to the output area location specified by the value in X2; X1 is incremented by 1; X2 is incremented by 1 if this is single-character conversion, or by 2 if it is double-character conversion, and the subroutine exits to CC3 and back to the calling program in partition when all characters have been converted.

Calling Program (user's)

This routine is to be supplied by the user. What will be explained here are two general approaches which can be adapted to convert single- or double-character messages. The specific programming must be done by the user.

SINGLE CHARACTER

The calling sequence required to effect a single-character translation is illustrated below, step by step. A brief operating description follows each statement.

```
MC                IADDR1,11
```

This instruction loads a four-position address constant defining the leftmost position of the input string into index register 1.

```
MC                OADDR1,21
```

This instruction loads a four-position address constant defining the leftmost position of the output area into index register 2. If desired, this address constant may be the same as that loaded into index register 1, in which case the translated characters will overlay the input characters.

```
MN                TADDR1,CC8+1
```

This statement sets the A address of a move-character instruction, which selects the translated characters from the translate table, to the leftmost position of the 64-character translate table.

```
MC                INLENG,CCSIZE
```

This instruction sets CCSIZE, a three-character counter, to the length (001-999) of the input character string.

```
BC                CC4+1(6),CC1(5)
```

This instruction has two effects: the address of the instruction following it is loaded into the A operand at CC4, there to be used as an exit point when all characters involved in this stage of translation are processed; and it branches to the single-character entry point at CC1.

A subroutine calling sequence and associated declaratives, which could be used to convert a 32-character buffer (BUFFER) containing CD-30 transmission code to its equivalent in System Ten internal USASCII, is illustrated below.

```
BUFFER DM    C32

TBL    DM    OC64
       DM    C32'Ø123456789ØØØØØØØ/STUVWXYZØ''ØØØ'
       DM    C32'-JKLMNOPQRØ%ØØØØØØABCDEFGHIØ.ØØØØ'
BUFADR DM    A'BUFFER'
TBLADR DM    A'TBL'
SIZE   DM    N'032'

       MC    BUFADR,11
       MC    BUFADR,21
       MN    TBLADR,CC8+1
       MC    SIZE,CCSIZE
       BC    CC4+1(6),CC1(5)
```

The contents of BUFFER before and after translation are given below.

Before: 3XY2ØY2ØF43G43ØVIFDØSFTUØSFE5UI3

After: THIS IS OUTPUT FROM CODE CONVERT

DOUBLE CHARACTER

The calling sequence required to effect a double -character translation is illustrated below, step by step. An explanation follows each step. The reader will note that it differs from the single-character translation process only in that the input and output addresses for this stage are in X1 and X2 respectively, the address of Table 2 is placed in the A operand of the Move Character instruction at CC10, and the branch is to the double-character entry point at CC2.

```
       MC              IADDR2,11
```

This instruction loads a four-position address constant defining the leftmost position of the input string into index register 1.

```
       MC              OADDR2,21
```

This instruction loads a four-position address constant defining the leftmost position of the output area into index register 2. This address must reference a different area than index register 1 does.

```
        MN              TADDR2,CC10+1
```

The above statement sets the A address of a move character instruction, which selects translated characters from the translate table, to the leftmost position of the 128-character translate table. Observe that the B operand of this statement is different from the corresponding statement in the single-character calling sequence.

```
        MC              INLENG,CCSIZE
```

This instruction sets CCSIZE, a three-character counter, to the length (001-999) of the input character string.

```
        BC              CC4+1(6),CC2(5)
```

The above instruction has two effects: the address of the instruction following it is loaded into the A operand at CC4, there to be used as an exit point when all characters involved in this stage of translation are processed; and it branches to the double-character entry point at CC2.

A subroutine calling sequence and associated declaratives, which could be used to produce the two-character (double frame) codes required to write the contents of BUF1 in EBCDIC on the Model 45 tape drive, are illustrated below.

```
BUF1    DM    C29'THIS IS INPUT TO CODE CONVERT'
BUF2    DM    C58
TBL     DM    0C128
        DM    C32'04J507K7K51605M7M4M5L5N4K606K416'
        DM    C32'0O1O2O3O4O5O6O7O8O9OJ7N5L4N7N6O6'
        DM    C32'L71L2L3L4L5L6L7L8L9L1M2M3M4M5M6M'
        DM    C32'7M8M9M2N3N4N5N6N7N8N9NM4N6M5L4M6'
BUFAD1  DM    A'BUF1'
BUFAD2  DM    A'BUF2'
TBLADR  DM    A'TBL'
SIZE    DM    N'029

        MC    BUFAD1,11
        MC    BUFAD2,21
        NM    TBLADR,CC10+1
        MC    SIZE,CCSIZE
        BC    CC4+1(6),CC2(5)
```

The contents of BUF2 following execution of code convert will be:

```
'3N8L9L2N049L2N049L5M7M4N3N043N6M043L6M4L5L043L6M5M5N5L9M3N
```

SYSTEM TEN CHARACTER	CHARACTER CODE	CD-30 CHARACTER TBL 1	FROM SYSTEM TEN TO DOUBLE FRAME TBL 2	FROM CD-30 TO DOUBLE FRAME TBL 3
SP	000000	SP	04	04
!	000001	1	J5	1Ø
"	000010	2	Ø7	2Ø
#	000011	3	K7	3Ø
$	000100	4	K5	4Ø
%	000101	5	L6	5Ø
&	000110	6	05	6Ø
'	000111	7	M7	7Ø
(	001000	8	M4	8Ø
)	001001	9	M5	9Ø
*	001010	1*	L5	1Ø*
+	001011		N4	
,	001100		K6	
-	001101	8*	06	8Ø*
.	001110		K4	
/	001111		16	
0	010000	0	0Ø	0Ø
1	010001	/	1Ø	16
2	010010	S	2Ø	2N
3	010011	T	3Ø	3N
4	010100	U	4Ø	4N
5	010101	V	5Ø	5N
6	010110	W	6Ø	6N
7	010111	X	7Ø	7N
8	011000	Y	8Ø	8N
9	011001	Z	9Ø	9N
:	011010	3*	J7	3Ø*
;	011011	,	N5	K6
<	011100		L4	
=	011101	4*	N7	4Ø*
>	011110		N6	
?	011111		Ø6	
@	100000	-	L7	06
A	100001	J	1L	1M
B	100010	K	2L	2M
C	100011	L	3L	3M
D	100100	M	4L	4M
E	100101	N	5L	5M
F	100110	Ø	6L	6M
G	100111	P	7L	7M
H	101000	Q	8L	8M
I	101001	R	9L	9M
J	101010	2*	1M	2Ø*
K	101011	%	2M	L6
L	101100		3M	
M	101101	6*	4M	6Ø*
N	101110		5M	
O	101111	-	6M	06
P	110000		7M	
Q	110001	A	8M	1L
R	110010	B	9M	2L
S	110011	C	2N	3L
T	110100	D	3N	4L
U	110101	E	4N	5L
V	110110	F	5N	6L
W	110111	G	6N	7L
X	111000	H	7N	8L
Y	111001	I	8N	9L
Z	111010	7*	9N	7Ø*
[	111011	.	M4	K4
\	111100		N6	
]	111101	5*	M5	5Ø*
~	111110		L4	
_	111111		M6	

* Indicates CD-30 Transaction Codes

FIGURE 3

This chart is designed to show conversions from CD-30 code into double frame EBCDIC with an intermediate translation from CD-30 into System Ten USASCII, then into double frame EBCDIC. The character shown in TBL 1 will result in the System Ten USASCII bit configuration of a character that is the same as the CD-30 character originally sent. The column under TBL 2 shows the translation pair of characters necessary to get from System Ten USASCII, and the column under TBL 3 shows the translation pair of characters necessary to translate truncated CD-30 code into double frame EBCDIC.

SYSTEM TEN CHARACTER	CD-30 TERMINAL OUTPUT FRIDEN BCD	CD-30 (TRUNCATED)	SYSTEM TEN USASCII	USASCII	USASCII 7-TRACK TAPE CODE	7-TRACK BCD TAPE CODE	GE-400 SERIES	EBCDIC (BINARY)	SYSTEM TEN CHARACTERS (DOUBLE FRAME)	HEX
	8 7 6 5 4 3 2 1		7 5 4 3 2 1	7 6 5 4 3 2 1	B A 8 4 2 1	B A 8 4 2 1	B A 8 4 2 1	0 1 2 3 4 5 6 7		
0	6	0	5	6 5	A	8 2		1 1 1 1 0 0 0 0	0 0	F 0
1	1	!	5 1	6 5 1	A 1	1	1	0 0 0 1	1 0	F 1
2	2	"	5 2	6 5 2	A 2	2	2	0 0 1 0	2 0	F 2
3	5 2 1	#	5 2 1	6 5 2 1	A 2 1	2 1	2 1	0 0 1 1	3 0	F 3
4	3	$	5 3	6 5 3	A 4	4	4	0 1 0 0	4 0	F 4
5	5 3 1	%	5 3 1	6 5 3 1	A 4 1	4 1	4 1	0 1 0 1	5 0	F 5
6	5 3 2	&	5 3 2	6 5 3 2	A 4 2	4 2	4 2	0 1 1 0	6 0	F 6
7	3 2 1	'	5 3 2 1	6 5 3 2 1	A 4 2 1	4 2 1	4 2 1	0 1 1 1	7 0	F 7
8	4	(	5 4	6 5 4	A 8	8	8	1 0 0 0	8 0	F 8
9	5 4 1	)	5 4 1	6 5 4 1	A 8 1	8 1	8 1	1 0 0 1	9 0	F 9
A	7 6 1	Q	7 1	7 1	B 1	B A 1	A 1	1 1 0 0 0 0 0 1	1 L	C 1
B	7 6 2	R	7 2	7 2	B 2	B A 2	A 2	0 0 1 0	2 L	C 2
C	7 6 5 2 1	S	7 2 1	7 2 1	B 2 1	B A 2 1	A 2 1	0 0 1 1	3 L	C 3
D	7 6 3	T	7 3	7 3	B 4	B A 4	A 4	0 1 0 0	4 L	C 4
E	7 6 5 3 1	U	7 3 1	7 3 1	B 4 1	B A 4 1	A 4 1	0 1 0 1	5 L	C 5
F	7 6 3 2	V	7 3 2	7 3 2	B 4 2	B A 4 2	A 4 2	0 1 1 0	6 L	C 6
G	7 6 3 2 1	W	7 3 2 1	7 3 2 1	B 4 2 1	B A 4 2 1	A 4 2 1	0 1 1 1	7 L	C 7
H	7 6 4	X	7 4	7 4	B 8	B A 8	A 8	1 0 0 0	8 L	C 8
I	7 6 5 4 1	Y	7 4 1	7 4 1	B 8 1	B A 8 1	A 8 1	1 0 0 1	9 L	C 9
J	7 5 1	A	7 4 2	7 4 2	B 8 2	B 1	B 1	1 1 0 1 0 0 0 1	1 M	D 1
K	7 5 2	B	7 4 2 1	7 4 2 1	B 8 2 1	B 2	B 2	0 0 1 0	2 M	D 2
L	7 2 1	C	7 4 3	7 4 3	B 8 4	B 2 1	B 2 1	0 0 1 1	3 M	D 3
M	7 5 3	D	7 4 3 1	7 4 3 1	B 8 4 1	B 4	B 4	0 1 0 0	4 M	D 4
N	7 3 1	E	7 4 3 2	7 4 3 2	B 8 4 2	B 4 1	B 4 1	0 1 0 1	5 M	D 5
O	7 4 3 2	F	7 4 3 2 1	7 4 3 2 1	B 8 4 2 1	B 4 2	B 4 2	0 1 1 0	6 M	D 6
P	7 5 3 2 1	G	7 5	7 5	B A	B 4 2 1	B 4 2 1	0 1 1 1	7 M	D 7
Q	7 5 4	H	7 5 1	7 5 1	B A 1	B 8	B 8	1 0 0 0	8 M	D 8
R	7 4 1	I	7 5 2	7 5 2	B A 2	B 8 1	B 8 1	1 0 0 1	9 M	D 9
S	6 5 2	2	7 5 2 1	7 5 2 1	B A 2 1	A 2	B A 2	1 1 1 0 0 0 1 0	2 N	E 2
T	6 1	3	7 5 3	7 5 3	B A 4	A 2 1	B A 2 1	0 0 1 1	3 N	E 3
U	6 5 3	4	7 5 3 1	7 5 3 1	B A 4 1	A 4	B A 4	0 1 0 0	4 N	E 4
V	6 3 1	5	7 5 3 2	7 5 3 2	B A 4 2	A 4 1	B A 4 1	0 1 0 1	5 N	E 5
W	6 3 2	6	7 5 3 2 1	7 5 3 2 1	B A 4 2 1	A 4 2	B A 4 2	0 1 1 0	6 N	E 6
X	6 5 3 2 1	7	7 5 4	7 5 4	B A 8	A 4 2 1	B A 4 2 1	0 1 1 1	7 N	E 7
Y	6 5 4	8	7 5 4 1	7 5 4 1	B A 8 1	A 8	B A 8	1 0 0 0	8 N	E 8
Z	6 4 1	9	7 5 4 2	7 5 4 2	B A 8 2	A 8 1	B A 8 1	1 0 0 1	9 N	E 9
SP	5	SP		6		A	A	0 1 0 0 0 0 0 0	0 4	4 0
!		"	1	6 1	1	B 8 2	B A 8 4 2 1	0 1 0 1 1 0 1 0	J 5	5 A
"		#	2	6 2	2		B A 8 4 2	0 1 1 1 1 1 1 1	O 7	7 F
#		$	2 1	6 2 1	2 1	8 2 1	8 2 1	0 1 1 1 1 0 1 1	K 7	7 B
$		%	3	6 3	4	B 8 2 1	B 8 2 1	0 1 0 1 1 0 1 1	K 5	5 B
%	7 5 4 2 1	&	3 1	6 3 1	4 1	A 8 4	B A 8 4	0 1 1 0 1 1 0 0	L 6	6 C
&		'	3 2	6 3 2	4 2	B A	A 8 2	0 1 0 1 0 0 0 0	O 5	5 0
'		(	3 2 1	6 3 2 1	4 2 1	8 4	B 8 4 2 1	0 1 1 1 1 1 0 1	M 7	7 D
(		)	4	6 4	8	A 8 4	A 8 4 1	0 1 0 0 1 1 0 1	M 4	4 D
)		*	4 1	6 4 1	8 1	B A 8 4	B 8 4 1	0 1 0 1 1 1 0 1	M 5	5 D
*		+	4 2	6 4 2	8 2	B 8 4	B 8 4	0 1 0 1 1 1 0 0	L 5	5 C
+		,	4 2 1	6 4 2 1	8 2 1	B A	B A	0 1 0 0 1 1 1 0	N 4	4 E
,	6 5 4 2 1	;	4 3	6 4 3	8 4	A 8 2 1	B A 8 2 1	0 1 1 0 1 0 1 1	K 6	6 B
-	7	@	4 3 1	6 4 3 1	8 4 1	B	B 8 2	0 1 1 0 0 0 0 0	O 6	6 0
.	7 6 4 2 1	[	4 3 2	6 4 3 2	8 4 2	B A 8 2 1	A 8 2 1	0 1 0 0 1 0 1 1	K 4	4 B
/	6 5 1	\	4 3 2 1	6 4 3 2 1	8 4 2 1	A 1	B A 1	0 1 1 0 1 0 0 1	1 6	6 1
:		:	5 4 2	6 5 4 2	A 8 2	8 4 1	8 4 1	0 1 1 1 1 0 1 0	J 7	7 A
;		;	5 4 2 1	6 5 4 2 1	A 8 2 1	B 8 4 2	B 8 4 2	0 1 0 1 1 1 1 0	N 5	5 E
=		=	5 4 3 1	6 5 4 3 1	A 8 4 1	8 2 1	B A 8 4 1	0 1 1 1 1 1 1 0	N 7	7 E
>		>	5 4 3 2	6 5 4 3 2	A 8 4 2	8 4 2	8 4 2	0 1 1 0 1 1 1 0	N 6	6 E
?		?	5 4 3 2 1	6 5 4 3 2 1	A 8 4 2 1	B A 8 2	8 4 2 1	0 1 1 0 1 1 1 1	O 6	6 F
@		\	7	7	B	8 4	8 4	0 1 1 1 1 1 0 0	L 7	7 C
[		]	7 5 4 2 1	7 5 4 2 1	B A 8 2 1	B A 8 4 1	8 2	0 1 0 0 1 1 0 1	M 4	4 D
]		<	7 5 4 3	7 5 4 3	B A 8 4	A 8 4 2	A 8 4 2 1	0 1 1 0 1 1 1 0	N 6	6 E
<			7 5 4 3 1	7 5 4 3 1	B A 8 4 1	B 8 4 1	A 8 4	0 1 0 1 1 1 0 1	M 5	5 D
			7 5 4 3 2 1	7 5 4 3 2 1	B A 8 4 2 1		B A 8 2	0 1 1 0 1 1 0 1	M 6	6 D
			5 4 3	6 5 4 3	A 8 4	B A 8 4 2	A 8 4 2	0 1 0 0 1 1 0 0	L 4	4 C
			7 5 4 3 2	7 5 4 3 2	B A 8 4 2		B	0 1 0 0 1 1 0 1	L 4	4 C

FIGURE 4

This chart shows the relative codes for the System Ten characters listed on the left. The second column shows the original 8-bit representation of this character that the CD-30 sent. (For these characters, the eighth bit is off; therefore, no eighth bit is shown here.) The third column shows the character that System Ten reads, because of the standard truncation of CD-30 code (bits 8 and 5 are ignored, bit 6 is accepted as the fifth bit in System Ten USASCII). The remaining columns show the representation of the character on the far left, in the type of code indicated at the top of each column.

SINGER
FRIDEN DIVISION

40-502 PRINTED IN U.S.A.