Chapter 7: Character Manipulation

I. Character Strings

A. CUSP: One byte per character, MSB 0, three characters per word.
   1. If length mod 3 != 0 then remaining characters of last word are ignored, and filled with space
      characters, $20

B. CHASM
   1. \$nn The character with hexadecimal value nn
   2. \ Backslash
   3. \CR Carriage return ($0D)
   4. \BEL Bell or beep ($07)
   5. \BS Back space ($08)
   6. \LF Line feed ($0A)
   7. \FF Form feed ($0C)

II. String Instructions

A. LDC = load character from a string. Loads a character into the least significant 8 bits of the accumulator.
   Upper 16 bits of the accumulator are cleared.
   1.Operand contains the address of the string, XR contains the index of the character within the string,
      starting at 0.
   2. All addressing modes allowed, except immediate mode

B. STC = store character into a string. Stores a character (least significant 8 bits) from the accumulator into
   the XR position of the string indicated in the operand.
   1. All addressing modes allowed, except immediate mode.

C. Given :
   1. $000 LDX# 1
   2. $001 LDC S ; 'o'
   3. $002 LDX# 4
   4. $003 STC S
   5. $004 HLT
   6. $005 S: .CHAR 'Howdy' ; $005 is $486F77 and $006 is $647920
   7. At the end of the program the accumulator would have 'o' in it, i.e., $00006F; and S will be 'Howdo'
      with $006 now being $646F20

D. Character constants may be used in any situation in which a value is required, including arithmetic
   expressions.
   1. Arithmetic digit + '0' will be the ASCII value of the corresponding character digit, e.g., 7 + '0' = '7'
   2. 'a' - 'A' is the difference between the two ASCII cases. To convert upper case to lower case just add ('a'
      - 'A'), e.g. 'F' + 'a' - 'A' = 'f'.

III. CUSP Mini-OS String Subroutines (Appendix A.12, pp 373 - 375)

A. PROCEDURE TO_STR(InValue: INTEGER; MaxLen: INTEGER; VAR OutString: CHAR_STRING;
   VAR ActualLen: INTEGER);
   1. JSR $E02
   2. Convert InValue to an ASCII character string representing it as a decimal value.
   3. On return ActualLen will contain the length of the string representation.
   4. MaxLen should contain the length of the buffer OutString.
   5. If the number cannot be represented in the space provided, then OutString will be filled with asterisks,
      and ActualLen will be set to MaxLen.

B. FUNCTION FRM_STR(NumChars: INTEGER; VAR InSTring: CHAR_STRING; VAR OutValue:
   INTEGER) : BOOLEAN;
   1. JSR $E03
   2. Converts the decimal integer stored in the ASCII string InString into its signed two's complement
      integer value that is stored in OutValue.
   3. NumChars is the length of the string representation.
4. If number cannot be represented in 24 bit two's complement form then the accumulator is set to 1 else 0. Note that this is the opposite of what you would expect; False (0) means success.

C. PROCEDURE GET_STR(MaxGetCh: INTEGER; VAR GetStr: CHAR_STRING; VAR ActGetCh: INTEGER);
   1. JSR $E04
   2. Reads a string from the keyboard into a OS buffer until the Enter key is pressed. Then copies up to MaxGetCh characters from the buffer into the GetStr string. Places the number of characters copied into ActGetCh.
   3. Allows the backspacing.
   4. If the user enters more than MaxGetCh characters, then the excess characters are discarded.

D. PROCEDURE PUT_STR(NumChar : INTEGER; VAR PutStr: CHAR_STRING);
   1. JSR $E05
   2. Displays the NumChar characters of the string PutStr on the screen.

E. PROCEDURE PUT_NL;
   1. JSR $E06
   2. Writes a <CR> <LF> to screen

F. PROCEDURE PUT_CHR
   1. JSR $E08
   2. Displays the character in the lower 8 bits of the accumulator on the screen.