```
    NAME        DISPLAY

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                   DISPLAY                                     ;
;                                  Homework 4                                   ;
;                                   EE/CS 51                                    ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;
; This file contains the functions for displaying strings on the 14-segment
; displays.  The functions included are:
;   Display        - displays a string to the LED display
;   DisplayNum     - displays a number as a decimal to the LED display
;   DisplayHex     - displays a number in hexadecimal to the LED display
;   InitDisplay    - initialize the display and its variables
;   DisplayMux     - multiplex the LED display


; Revision History:
;     10/24/16      Jennifer Du      initial revision
;     10/26/16      Jennifer Du      writing assembly code
;     10/29/16      Jennifer Du      commenting


;external function declarations

    EXTRN   Hex2String:NEAR     ; converts number to hexstring
    EXTRN   Dec2String:NEAR     ; converts number to decstring

    EXTRN   ASCIISegTable:BYTE  ; 14-segment codes for segment buffer



; include files
$INCLUDE(display.inc)           ;
$INCLUDE(common.inc)
$INCLUDE(converts.inc)



CGROUP  GROUP   CODE
DGROUP  GROUP   DATA, STACK

CODE    SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP, DS:DGROUP, SS:STACK


; Display
;
; Description:      This function converts an ASCII string into the
;                   series of 14-segment codes that, when ported to the LED
;                   display, forms a visual representation of that string.
;                   The function is passed a <null> terminated string (str) to
;                   output to the LED display. The string is passed by
;                   reference in ES:SI. The maximum length of the string that can be
;                   displayed at any given moment is 8 characters long. Anything
;                   longer than this will be cut off.
;
; Operation:        This function will loop through the given string, and look
;                   up the 14 segment code for each character in the 14-segment
;                   code table. Then it will write the value of the 14 segment
;                   code to the buffer in the order that the characters appear.
;                   If the string is shorter than the length of the segment
```

```
;                   buffer, the buffer will be padded with blank spaces. If the
;                   string is longer than the length of the segment buffer, it
;                   will be cut off at 8 characters.
;
; Arguments:        SI - address of string to be displayed
; Return Value:     None.
;
; Local Variables:  SI - address of string to be displayed
;                   CX - counter to keep track of current space in segment buffer
;                   AX - stores ASCII value of current character being looked up
;                   BX - temp variable used as index to lookup values in tables
; Shared Variables: segBuffer - place to store the segment code values
; Global Variables: None.
;
; Input:            None.
; Output:           None.
;
; Error Handling:   None.
; Registers Used:   Flags, AX, BX, CX, SI.
;
; Algorithms:       None.
; Data Structures:  The segment buffer is an array of words which holds the
;                   14-segment code values for each character in the string
;


Display         PROC    NEAR
                PUBLIC  Display

StartDisplay:
    PUSHA                           ; save registers

    MOV     CX, 0                   ; initialize counter for the segbuffer

CheckEndOfString:
    XOR     AX,AX                   ; clear AX
    MOV     AL, ES:[SI]             ; get value of the first character in the string
    INC     SI                      ; move to next character in string
    CMP     AL, ASCII_NULL          ; see if the string has ended (aka if the current
                                        ; character is euqal to ASCII_NULL)
    JE      EndOfString             ; if character is null, jump to end of the string
    ;JMP    StoreSegTableValue

StoreSegTableValue:

    SHL     AX, 1                   ; multiply the ascii character value by 2 (since
                                        ; each code is 2 bytes long, we want to look up
                                        ; 2*ASCII_VAL to get to the right character)
    MOV     BX, AX                  ; move the ascii value (index in the table) to BX to access

    MOV     AL, CS:ASCIISegTable[BX]    ; move the code values in byte by byte
    INC     BX                          ; move to the second part of the display code pattern
    MOV     AH, CS:ASCIISegTable[BX]    ; move in higher byte

    MOV     BX, CX                  ; move counter for segbuffer here
    MOV     segBuffer[BX], AX       ; move value into BX

    ADD     CX, WORDSIZE            ; increment segBuffer counter to go to next empty spot
    CMP     CX, numSegsBytes        ; if we reach capacity of the segment buffer, they're equal
    JL      CheckEndOfString        ; if counter is less than length, store more display codes
    JGE     EndDisplay                  ; >= means we end this function, buffer can't fit more

EndOfString:                        ; if we have reached the end of the string

    MOV     BX, CX                  ; move segBuffer counter into BX to access as index
```

```
    MOV     segBuffer[BX], DISPLAY_NULL   ; store null string in each entry in segment buffer
    ADD     CX, WORDSIZE                  ; increment segBuffer counter to go to next empty entry
    ;JMP     CheckEndOfBufferAfterString

CheckEndOfBufferAfterString:              ; string is done and we are checking if buffer
                                          ; capacity has been reached
    CMP     CX, numSegsBytes
    JL      EndOfString                   ; buffer capacity not reached: add more spaces
    ;JGE     EndDisplay                   ; if buffer capacity has been reached, end!

EndDisplay:
    POPA                    ;restore registers
    RET                     ; we are done, return

Display ENDP

;
;
; DisplayNum
;
; Description:      This function turns a given number into its decimal
;                   representation and gets it ready to be displayed on the
;                   LED display. The function is passed a 16-bit signed value
;                   (n) to output in decimal (at most 5 digits plus sign) to
;                   the LED display. The number (n) is passed in AX by value.
;                   The resulting string is written to DS:SI.
;
; Operation:        We will use two previously written functions to
;                   display a number in decimal. First, we will turn the given
;                   number into a string in decimal form using Dec2String, and
;                   then we will call Display on this string to show it
;                   on the LED display. The resulting string will be less than
;                   the length of the LED, and any unused spaces will not display
;                   anything on the LED display.
;
; Arguments:        AX - 16-bit signed value to be turned into a decimal string
; Return Value:     None.
; Local Variables:  AX - number to be displayed
;                   SI - address of string to be displayed
; Shared Variables: segBuffer - place to store the segment code values
;                   stringBuffer - place to store the string from Dec2String function
; Global Variables: None.
; Input:            None.
; Output:           None.
; Error Handling:   None.
; Registers used:   SI, BX, AX.
; Algorithms:       None.
; Data Structures:  stringBuffer - stores the characters of the string after converting
;                       decimal to string.
;

DisplayNum      PROC    NEAR
                PUBLIC  DisplayNum
    PUSHA
    MOV     SI, OFFSET(stringBuffer) ; DS:SI should point to stringBuffer, set this
                                     ; up so Dec2String can write string there.
    MOV     BX, DS                   ; set ES equal to DS for Display function
    MOV     ES, BX

    PUSH    SI                       ; keep Dec2String from changing SI
    CALL    Dec2String               ; turns number to decimal string
    POP     SI

    CALL    Display                  ; calls display on the string
    POPA
    RET
```

```
DisplayNum      ENDP

;
;
; DisplayHex
;
; Description:      This function turns a given number into its hex
;                   representation and gets it ready to be displayed.
;                   The function is passed a 16-bit unsigned value (n) to
;                   output in hexadecimal (at most 4 digits) to the LED
;                   display. The number (n) is passed in AX by value.
;                   The resulting string is written to DS:SI.
;
; Operation:        We will use two previously written functions to
;                   display a number in hex. First, we will turn the given
;                   number into a string in hex form using Hex2String, and
;                   then we will call Display on this string to show it
;                   on the LED display. Any unused digits will show up as
;                   blank on the LED display.
;
; Arguments:        AX - 16-bit unsigned value to be turned into a hex string
;
; Return Value:     None.
;
; Local Variables:  AX - 16-bit unsigned value to be turned into a hex string
;                   SI - address of string to be displayed
; Shared Variables: segBuffer - place to store the segment code values
;                   stringBuffer - place to store the string from Hex2String function
; Global Variables: None.
; Input:            None.
; Output:           None.
; Error Handling:   None.
; Registers used:   SI, BX, AX.
; Algorithms:       None.
; Data Structures:  stringBuffer - string array for storing result of Hex2String
;
DisplayHex      PROC    NEAR
                PUBLIC  DisplayHex

    MOV     SI, OFFSET(stringBuffer)    ; set address of SI up so that Hex2String
                                        ; can write the string here
    MOV     BX, DS                      ; set ES equal to DS
    MOV     ES, BX

    PUSH    SI              ; keep Hex2String from changing SI
    CALL    Hex2String      ; converts number to hex string
    POP     SI

    CALL    Display         ; displays string on LED display
    RET

DisplayHex      ENDP




;
;
;
; InitDisplay
;
; Description:      This function initializes the segment buffer, clears
;                   the display (by clearing the seg buffer), and
;                   initializes display multiplexing variables.
;
; Operation:        This function blanks the digits and initializes the
```

```
;                   display muxing variables.
;
; Arguments:        None.
; Return Value:     None.
; Local Variables:  BX - counter for looping through segment buffer
; Shared Variables: currentSeg - keeps track of next digit for mux
;                   segBuffer  - buffer is filled with DISPLAY_BLANK
; Global Variables: None.
; Input:            None.
; Output:           The LED display is blanked.
; Error Handling:   None.
; Registers used:   BX, DX, AX.
; Algorithms:       None.
; Data Structures:  Segment buffer (segBuffer) - array of 14-segment display codes.
;


InitDisplay     PROC    NEAR
                PUBLIC  InitDisplay

StartInitDisplay:
    PUSHA                           ; save registers

    MOV     BX, 0                   ; start counter at 0 (this counter loops through
                                    ; segment buffer and clears each entry)
    MOV     DX, IO_LED_LOC          ; get I/O location of LED display
    MOV     AL, IO_LED_VAL          ; get I/O value to write to IO_LED_LOC
    OUT     DX, AL                  ; write 0 to I/O location 0FFA4H for display chip select
    logic

ClearDisplay:                       ;start clearing the display
    MOV     segBuffer[BX], LED_BLANK    ; move blank character into each segBuffer entry
    INC     BX                      ; increment counter
    CMP     BX, numSegsBytes        ; see if we have reached the end of segment buffer
    JNE     ClearDisplay            ; if not, then clear next entry in segment buffer
    ;JE      InitMuxVariables

InitMuxVariables:
    MOV     currentSeg, 0           ; Initialize current mux segment

EndInitDisplay:
    POPA                            ; restore registers and
    RET                             ; return

InitDisplay     ENDP




;
;
;
;
; DisplayMux
;
; Description:      Multiplexer for the display. This procedure multiplexes
;                   the LED display under interrupt control. This
;                   function is going to display 1 digit for 1 instance.
;
; Operation:        The multiplexer remembers which digit was called last,
;                   by storing and incrementing the currentSeg variable
;                   (accounting for wraparound). Then it writes the
;                   14-segment code of the next digit to the display at the
;                   current digit. One digit is output each time this function
;                   is called.
;
; Arguments:        None.
```

```
; Return Value:     None.
; Local Variables:  None.
; Shared Variables: currentSeg - number that keeps track of which digit is
;                                being displayed
;                   buffer     - segment buffer holding segment code values
; Global Variables: None.
; Input:            None.
; Output:           The next digit is output to the display.
; Error Handling:   None.
; Registers used:   AX, BX, CX, DX.
; Algorithms:       None.
; Data Structures:  segment buffer - array of bytes holding segment code values
;



DisplayMux          PROC    NEAR
                    PUBLIC  DisplayMux

StartDisplayMux:
    PUSHA                           ; save registers

    MOV     BX, currentSeg          ; BX will be the lookup index
    SHL     BX, 1                   ; multiply by 2 since each word-sized display code
                                        ; starts at even indices (every other one)
    MOV     AX, WORD PTR segBuffer[BX]
                                    ; move display code into AX (word-sized)

    XCHG    AH, AL                  ; move higher byte (AH) into AL to display first
    MOV     DX, HIGH_BYTE_ADDRESS ; higher byte must be ported into 0008H
    OUT     DX, AL                  ; display higher byte code

    XCHG    AH, AL                  ; now we display lower byte (AL)

    MOV     DX, currentSeg          ; display in segBuffer at index currentSeg must be
                                        ; displayed at currentSeg address (index in buffer
                                        ; is equal to index on LED display)
    OUT     DX, AL                  ; display lower byte display code

IncrementMuxCounter:                ; set number to mux next time
    MOV     BX, currentSeg
    INC     BX                      ; increment current segment
    MOV     AX, BX                  ; move current segment to AX to divide
    MOV     CX, numSegs             ; get (currentSeg + 1) mod (number of segments)
    DIV     CX                      ; to account for mux counter wraparound
    MOV     currentSeg, DX

EndDisplayMux:
    POPA                            ; restore registers
    RET                             ; done multiplexing LEDs - return

DisplayMux          ENDP

CODE    ENDS


; the data segment

DATA    SEGMENT PUBLIC  'DATA'

    ; buffer holding currently displayed pattern
segBuffer       DW      numSegsBytes    DUP     (?)

    ; current digit to be muxed next
currentSeg      DW      ?
```

```
    ; character array, stores string before conversion into 14-seg codes
stringBuffer    DB      numSegsBytes    DUP     (?)

DATA    ENDS


;the stack

STACK   SEGMENT STACK  'STACK'

        DB      80 DUP ('Stack ')       ;240 words

        TopOfStack      LABEL   WORD

STACK   ENDS

END
```