```
    NAME        EVTHNDLR

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                EVENT HANDLERS                                 ;
;                                  Homework 4                                   ;
;                                   EE/CS 51                                    ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; This program is an event handler that manages interrupt service routines for
; the procedures for displaying strings on the LED display. The included
; functions are general enough to be used by functions other than the display
; functions. The included functions are:
;   InitTimer - initializes timer
;   InitCS  - initializes chip select
;   ClrIRQVectors  - installs IllegalEventHandler for all invalid interrupts
;   IllegalEventHandler - sends EOI to interrupt handler to exit interrupt


;
; external function declarations

    EXTRN   TimerEventHandler:NEAR      ; located in tmrhndlr.asm, this function calls DisplayMux

;
; Include files
$INCLUDE(handlers.inc)                  ; include file for event handlers, interrupts, and
timers

CGROUP  GROUP   CODE

CODE    SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP

;
;
;
; InitTimer
;
; Description:      This function will initialize the timer. The
;                   timer will be initialized to generate interrupts every
;                   MS_PER_SEG milliseconds. The interrupt controller is
;                   also initialized here to allow the timer interrupts.
;                   The timer counts MS_PER_SEG long intervals to generate
;                   the interrupts. This function is based on Glen's code.
;
; Operation:        The appropriate values are written to the timer control
;                   registers in the PCB. The timer count registers are set
;                   to zero. The interrupt controller is set up to accept
;                   timer interrupts and any pending interrupts are cleared
;                   by sending a TimerEOI to the interrupt controller.
;
; Arguments:        None.
; Return Value:     None.
; Local Variables:  None.
; Shared Variables: None.
; Global Variables: None.
; Input:            None.
; Output:           None.
; Error Handling:   None.
; Algorithms:       None.
; Data Structures:  None.
;
InitTimer       PROC    NEAR
```

```
                PUBLIC  InitTimer


                                ;initialize Timer #0 for MS_PER_SEG ms interrupts
        MOV     DX, Tmr0Count   ;initialize the count register to 0
        XOR     AX, AX
        OUT     DX, AL

        MOV     DX, Tmr0MaxCntA ;setup max count for milliseconds per segment
        MOV     AX, MS_PER_SEG  ;   count so can time the segments
        OUT     DX, AL

        MOV     DX, Tmr0Ctrl    ;setup the control register, interrupts on
        MOV     AX, Tmr0CtrlVal
        OUT     DX, AL

                                ;initialize interrupt controller for timers
        MOV     DX, INTCtrlrCtrl;setup the interrupt control register
        MOV     AX, INTCtrlrCVal
        OUT     DX, AL

        MOV     DX, INTCtrlrEOI ;send a timer EOI (to clear out controller)
        MOV     AX, TimerEOI
        OUT     DX, AL


        RET                     ;done so return


InitTimer       ENDP


;
;
;
; InitCS
;
; Description:      This function will initialize the peripheral chip
;                   selects on the 80188. Based on Glen's code.
;
; Operation:        This writes the initial values to the PACS and
;                   MPCS registers.
;
; Arguments:        None.
; Return Value:     None.
; Local Variables:  None.
; Shared Variables: None.
; Global Variables: None.
; Input:            None.
; Output:           None.
; Error Handling:   None.
; Algorithms:       None.
; Data Structures:  None.
;


InitCS  PROC    NEAR
        PUBLIC  InitCS

        MOV     DX, PACSreg     ;setup to write to PACS register
        MOV     AX, PACSval
        OUT     DX, AL          ;write PACSval to PACS (base at 0, 3 wait states)

        MOV     DX, MPCSreg     ;setup to write to MPCS register
        MOV     AX, MPCSval
        OUT     DX, AL          ;write MPCSval to MPCS (I/O space, 3 wait states)
```

```
        RET                     ;done so return

InitCS  ENDP

;
; ClrIRQVectors
;
; Description:      This functions installs the IllegalEventHandler for all
;                   interrupt vectors in the interrupt vector table.  Note
;                   that all 256 vectors are initialized so the code must be
;                   located above 400H.  The initialization skips  (does not
;                   initialize vectors) from vectors FIRST_RESERVED_VEC to
;                   LAST_RESERVED_VEC. This code is modelled after Glen's code.
;
; Arguments:        None.
; Return Value:     None.
;
; Local Variables:  CX - vector counter
;                   ES:SI - pointer to vector table
; Shared Variables: None.
; Global Variables: None.
; Input:            None.
; Output:           None.
; Error Handling:   None.
; Algorithms:       None.
; Data Structures:  None.
;
ClrIRQVectors   PROC    NEAR
                PUBLIC  ClrIRQVectors

InitClrVectorLoop:              ;setup to store the same handler 256 times

        XOR     AX, AX          ;clear ES (interrupt vectors are in segment 0)
        MOV     ES, AX
        MOV     SI, 0           ;initialize SI to the first vector

        MOV     CX, 256         ;up to 256 vectors to initialize


ClrVectorLoop:                  ;loop clearing each vector
                                ;check if should store the vector
        CMP     SI, 4 * FIRST_RESERVED_VEC
        JB      DoStore         ;if before start of reserved field - store it
        CMP     SI, 4 * LAST_RESERVED_VEC
        JBE     DoneStore       ;if in the reserved vectors - don't store it
        ;JA     DoStore         ;otherwise past them - so do the store

DoStore:                        ;store the vector
        MOV     ES: WORD PTR [SI], OFFSET(IllegalEventHandler)
        MOV     ES: WORD PTR [SI + 2], SEG(IllegalEventHandler)

DoneStore:                      ;done storing the vector
        ADD     SI, 4           ;update pointer to next vector

        LOOP    ClrVectorLoop   ;loop until have cleared all vectors
        ;JMP    EndClrIRQVectors;and all done


EndClrIRQVectors:               ;all done, return
        RET


ClrIRQVectors   ENDP
```

```
;
;
;
; IllegalEventHandler
;
; Description:      This function will be modelled after Glen's code. This
;                   function is the event handler for illegal (uninitialized)
;                   interrupts. It is called when an illegal interrupt occurs.
;
; Operation:        When this function is called, nothing happens, except that
;                   it sends a non-specific EOI and returns.
;
; Arguments:        None.
; Return Value:     None.
; Local Variables:  None.
; Shared Variables: None.
; Global Variables: None.
; Input:            None.
; Output:           None.
; Error Handling:   None.
; Algorithms:       None.
; Data Structures:  None.

IllegalEventHandler     PROC    NEAR
                        PUBLIC  IllegalEventHandler

        NOP                             ;do nothing (can set breakpoint here)

        PUSH    AX                      ;save the registers
        PUSH    DX

        MOV     DX, INTCtrlrEOI         ;send a non-sepecific EOI to the
        MOV     AX, NonSpecEOI          ;   interrupt controller to clear out
        OUT     DX, AL                  ;   the interrupt that got us here

EndIllegalEventHandler:
        POP     DX                      ;restore the registers
        POP     AX

        IRET                            ;and return


IllegalEventHandler     ENDP

CODE        ENDS

    END
```