Solution to Assignment 2
Assembly Language Programming

Question 1

The best way to write assembly-language programs that are more than a few lines long is to start with a high-level version of the program. It is much easier to write, debug and optimize a high-level description of the code.

The 'C' code for a solution to this problem is as follows:

```
#include <stdio.h>
#include <dos.h>

void printhex1 ( char c )
{
  if ( c < 10 ) {
    putchar ( c + '0' ) ;
  } else {
    putchar ( c - 10 + 'A' ) ;
  }
}

void printhex4 ( short i )
{
  printhex1 ( ( i >> 12 ) & 0xf ) ;
  printhex1 ( ( i >>  8 ) & 0xf ) ;
  printhex1 ( ( i >>  4 ) & 0xf ) ;
  printhex1 ( ( i >>  0 ) & 0xf ) ;
}

main()
{
  short i ;
  for ( i=0 ; i < 64 ; i+=4 ) {
    printhex4 ( peek ( 0, i+2 ) ) ;
    putchar ( ':' ) ;
    printhex4 ( peek ( 0, i+0 ) ) ;
    putchar ( '\r' ) ;
    putchar ( '\n' ) ;
  }
}
```

where `peek()` is a function available in many DOS compilers that returns the value of memory at the given segment and offset.

Many C compilers have options to display the compiled assembly language code. Most compilers also optimize their output. I used this technique and simplified the resulting code to come up with the following solution (the `@`-form labels were generated by the compiler):

```
;
; ELEC 379 Solution for Assignment 2
; Ed Casas, October 8, 1998
;
; print the first 16 interrupt vectors
;

code segment public
        assume  cs:code,ds:code
        org     100h

start:
        jmp     main

;   purpose: print character using int 21H function 2
; arguments: AL - character to print
;   returns: none

putchar:
        push    ax
        push    dx

        mov     dl,al           ; use DOS to
        mov     ah,02h          ; print character
        int     21h

        pop     dx              ; restore ax and dx
        pop     ax
        ret

;   purpose: print a value 0-15 as hex digit
; arguments: AL - value to print
;   returns: none

printhex1:
        push    ax

        cmp     al,10           ; if less than 10
        jge     @2
        add     al,'0'          ; add ASCII '0'
        call    putchar
        jmp     @1
@2:                             ; else subtract 10
        add     al,'A'-10       ; and add ASCII 'A'
        call    putchar
@1:
        pop     ax
        ret
```

```
;   purpose: print a 16-bit value as 4 hex digits
; arguments: AX - value to print
;   returns: none

printhex4:
        push    ax
        push    bx
        push    cx

        mov     bx,ax           ; save value in BX

        mov     cl,12           ; shift and
        shr     ax,cl
        and     al,15           ; mask in MS nybble
        call    printhex1       ; and print it

        mov     ax,bx           ; same with
        mov     cl,8            ; second MS nybble
        shr     ax,cl
        and     al,15
        call    printhex1

        mov     ax,bx           ; same with
        mov     cl,4            ; second LS nybble
        shr     ax,cl
        and     al,15
        call    printhex1

        mov     ax,bx           ; same with LS
        and     al,15           ; nybble
        call    printhex1

        pop     cx
        pop     bx
        pop     ax
        ret

;   purpose: return word at memory location AX:BX
; arguments: AX - segment
;            BX - offset
;   returns: AX - value read from memory

peek:
        push    ds
        mov     ds,ax
        mov     ax,[bx]
        pop     ds
        ret

;   purpose: print first 16 interrupt vectors in
;            hex in segment/offset format SSSS:OOOO
; arguments: none
;   returns: none

; print values of first 16 interrupt vectors

main:
        push    ax
        push    bx
        push    cx

        mov     cx,0            ; initialize pointer into
        ; interrupt table
        jmp     @6
@8:
        mov     ax,0            ; get the segment value
        mov     bx,cx
        add     bx,2
        call    peek
        call    printhex4       ; and print it

        mov     al,':'          ; print separator
        call    putchar

        mov     ax,0            ; get offset value
        mov     bx,cx
        call    peek
        call    printhex4       ; and print it

        mov     al,13           ; print CR/LF
        call    putchar
        mov     al,10
        call    putchar

        add     cx,4            ; point to next interrupt
@6:
        cmp     cx,64           ; loop back if not done
        jl      @8

        pop     cx
        pop     bx
        pop     ax

        int     21h             ; return to DOS

code ends
        end     start
```