EECE 379 : DESIGN OF DIGITAL AND MICROCOMPUTER SYSTEMS
2000/2001 WINTER SESSION, TERM 1

Solution for Assignment 1 (Revised)

The first version of these solutions gave a solution for the wrong device (a 74LS168) for Question 1.

Question 1

The following solution attempts to reduce the complexity of the design by separating the counting and control functions. The four control inputs are combined into a three-bit operation-select value. This value is then used to select the next counter state.

The use of conditional assignments would have simplified the description because of the prioritized operation of the reset and load controls.

```
-- ELEC 379 Solution to Assignment 1
-- 74LS168 Decade Counter
-- Ed Casas

entity ec162 is
port (
   sr, pe, cet, cep, cp : in bit ;
   p : in bit_vector (3 downto 0) ;
   q : out bit_vector (3 downto 0) ;
   tc : out bit ) ;
end ec162;

architecture rtl of ec162 is
   signal c, nextc, cplus1 : bit_vector (3 downto 0) ;
   signal operation : bit_vector (2 downto 0) ;
   signal countenable : bit ;
begin

   -- both cet and cep must be high to count
   countenable <= cet and cep ;

   -- build operation control word
   operation <= sr & pe & countenable ;

   -- operation selects source of next count
   with operation select nextc <=
        "0000" when "000",
        "0000" when "001",
        "0000" when "010",
        "0000" when "011",
        p      when "100",
        p      when "101",
        c      when "110",
        cplus1 when "111" ;

   -- next-count lookup table
   with c select cplus1 <=
        "0001" when "0000",
        "0010" when "0001",
        "0011" when "0010",
        "0100" when "0011",
        "0101" when "0100",
        "0110" when "0101",
        "0111" when "0110",
        "1000" when "0111",
        "1001" when "1000",
        "0000" when "1001",
        "1011" when "1010",
        "0100" when "1011",
        "1101" when "1100",
        "0100" when "1101",
        "1111" when "1110",
        "0000" when others ;

   -- connect count to output
   q <= c ;

   -- terminal count
   with c select tc <=
        cet when "1001",
        ’0’ when others ;

   -- instantiate the count register
   process(cp)
   begin
        if cp’event and cp = ’1’ then
                c <= nextc ;
        end if ;
   end process ;

end rtl ;
```

Figure 1 shows the simulation results.

Question 2

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’ ) ;
```

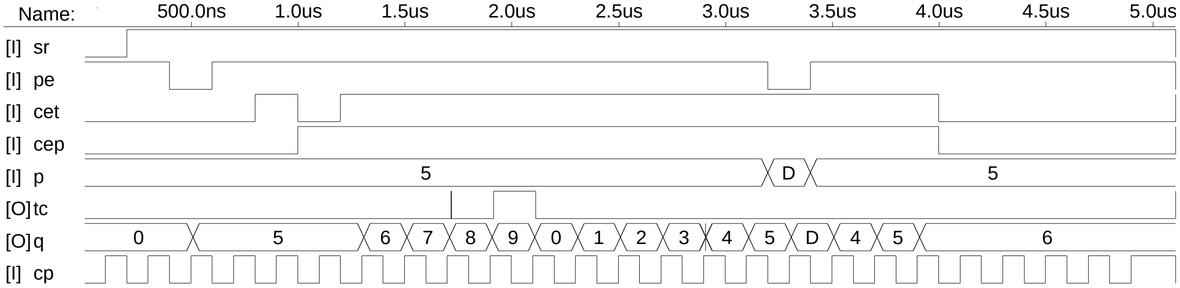

Figure 1: Simulation Results.

```
  }
}

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 1
; Ed Casas
;
; 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
```