CIS40 REAL TIME PROCESSING PROJECT

Project goal, design a function that generates a variable delay up to 60seconds using the PC system timer. The system timer is called by INT 21 with ah = 2ch, the hundredths are returned in dl, and seconds in dh.

The project can be designed for up to only 1 second delay, and 30 with points possible, or for up to 60 seconds incorporating both seconds and hundredths for 50points possible.

The choice is yours based on your time and motivation. The time delay must be incorporated into either the LED scan or the rotating wheel library. If the values are passed to the scan routine from the keyboard interface created in exercises 8-10, and the scan can accept new values without having to restart the program then 10 extra points will be given (for up to 60pts).

The delay routine will accept the hundredths delay parameter from the calling routine in the AL register, and the seconds delay in the AH register.

The delay routine will return to the calling routine after the requested delay period.

A time counting function will be provided to help determine if your delay function is creating the correct delay.

The project can be broken into two parts:

Part 1 Design and psuedo-code a delay algorithm
For the 1 second delay using the hundredths this is worth 10 of the 30 points, for the minute of delay using both seconds and hundredths the algorithm and psuedo-code is worth 20points of the 60.

This portion can be done in groups or individually as you prefer.

Part 2 Code and test the algorithm in assembly

The use of MASM will be necessary for this part, a code listing and demo is required for full credit

This portion must be done by each person individually. (Each person must attempt to write and debug their code, however consulting with others is allowed)

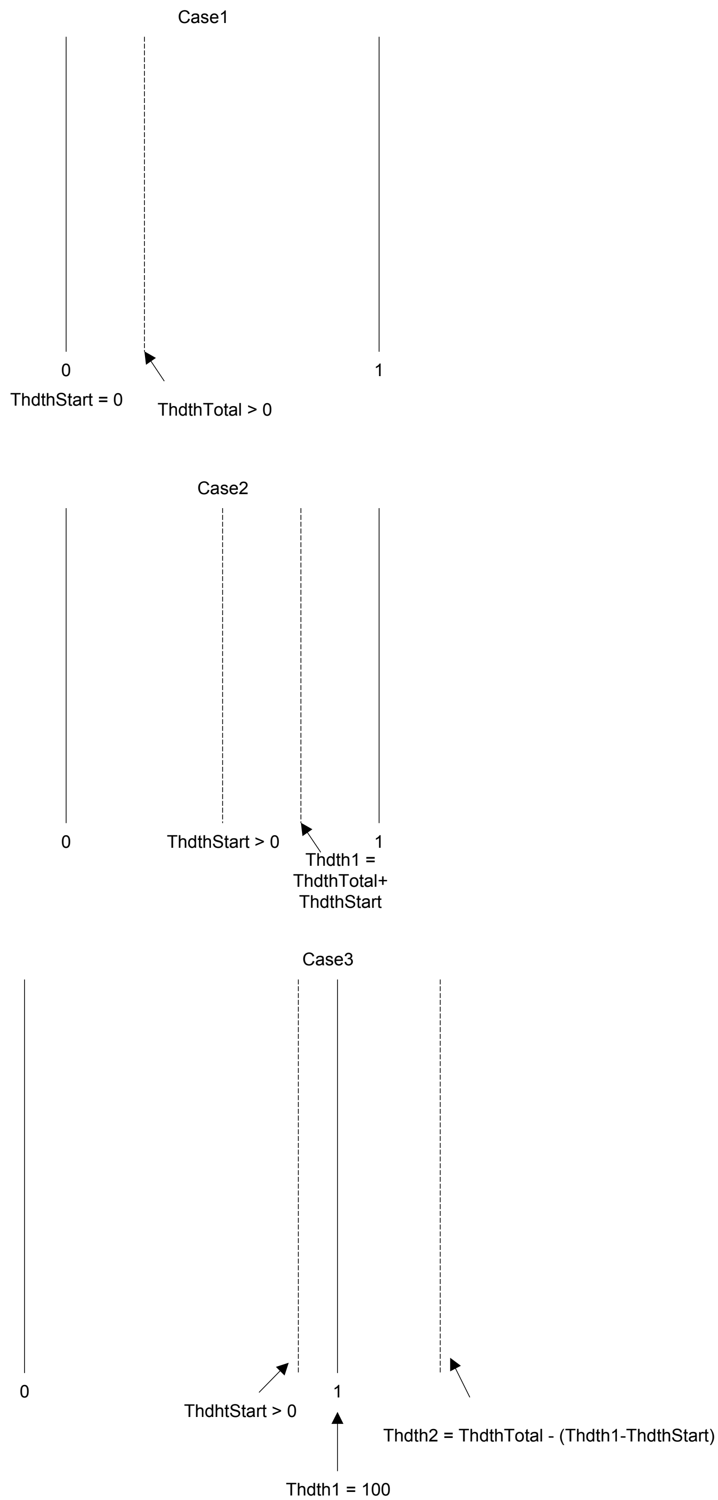
CREATING AND TROUBLESHOOTING THE ALGORITHM
There are many ways to implement the algorithm, the following diagrams indicate some of the issues related to tracking the total time accurately.

A couple of approaches might be to perform the delays in small pieces, or to try to *calculate total delay incorporating hundredths and seconds into one total calculation, which counts entire length of time based on hundredths* ***(this is probably the easist method).***

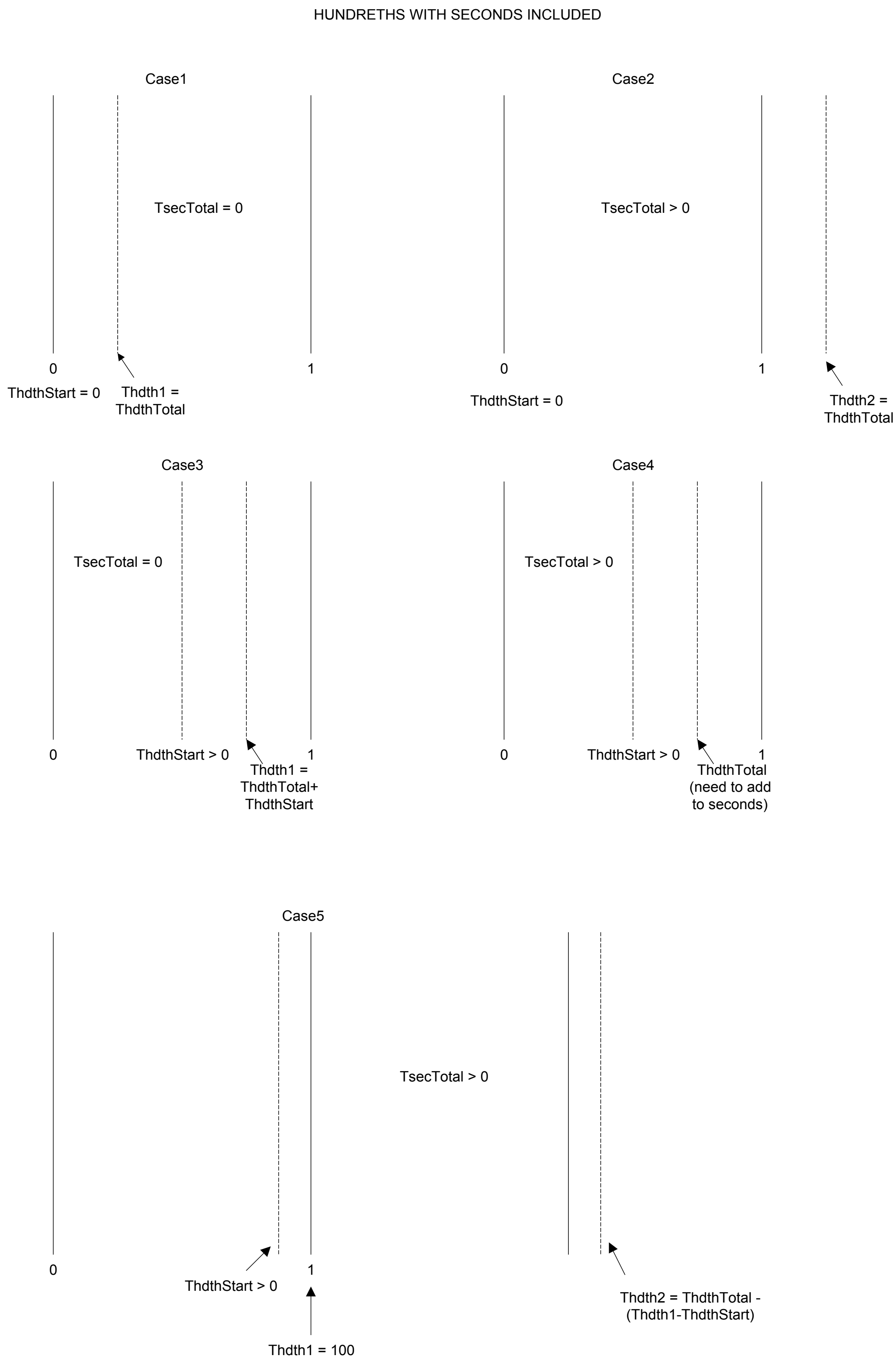
The difficulty in creating the algorithm is that PC system time ticks in increments of either 5 or 6 hundredths of a second (it alternates), therefore it is very easy to miss the hundredths to seconds transitions The other problem occurs when you are debugging your code (trying to step through it) ; the system time keeps changing. Therefore, the name REAL TIME processing applies to this software (this is a problem in most real time software systems since they interface to hardware that is constantly changing). To troubleshoot real time systems, in many cases you have to record a series of real time events (in this case each hundredths and seconds tick of the PC system), therefore a software buffer to save them is required. Two methods of debugging real time programs are shown in subsequent sections.

ISSUES REGARDING HUNDREDTHS AND SECONDS START AND END POINTS
Hundredths only routine, and hundredths start point relative to one second tick point

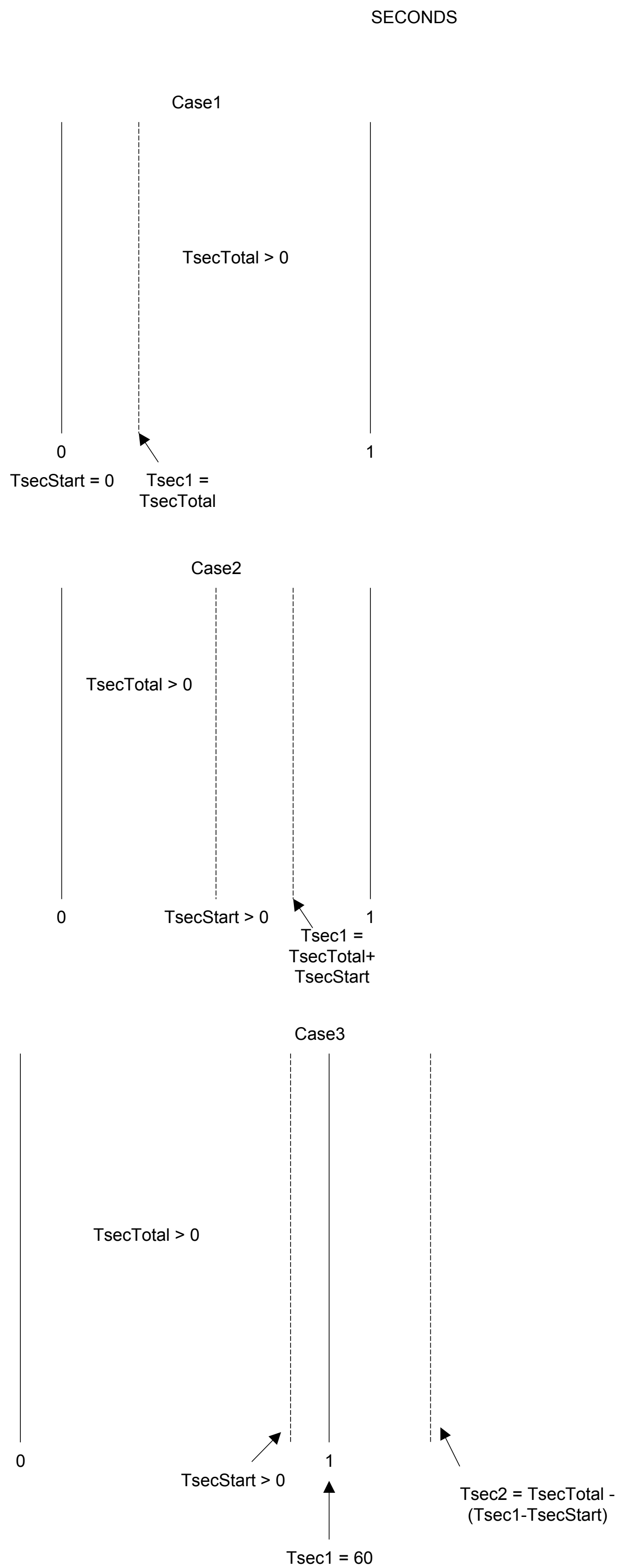
HUNDRETHS



Hundredths with Seconds Routine
hundredths start point relative to one second tick point, however now multiple seconds may be between start and end points



Seconds also wrap back to zero after 59, so total if adding seconds must include wrap effect



Test Code Library Provided for the Project
Assuming you name your delay routine Tdelay, then adding the library cntsechd.lib to your project will help you test it verify it works correctly by recording the length of your time delay.
The cntsechd.lib contains two routines to help you debug your project. Note copy the cntsechd.lib to the c:\masm611\lib folder. Then, to incorporate them into your project add the library as shown above. Then add the following lines of code to call the routines.

```
    .Model Small
    .STACK  200
        .DATA
;Add the following prototype definitions
CountSecHdths              PROTO
ResetCountSecHdths         PROTO

        .CODE
        .STARTUP

Tdelay   PROC              NEAR
;Somewhere at the top of the delay routine invoke ResetCountSecHdths
    invoke  ResetCountSecHdths
        .        .
LOOP_UNTIL_TIMEOUT:
        Mov      ah,2ch
        Int      21h
;add the count routine after the int 21h command to read the system time (note the dl ;register must stay
intact before calling CountSecHdths)
        invoke  CountSecHdths

        ;your own delay testing code follows
        cmp     ax,0
        jne        LOOP_UNTIL_TIMEOUT
```

Reading out the counted delay time

At segment DS = 2000 and offset 1FF0 the results of the CountSecHdths routine will be stored.

Total Accumulated Time
Perform d 2000:1FF0
1FF0 = Total Counted Seconds
1FF2 = Total Counted Hundredths

Individual Time Stamps
Perform d 2000:2000
The individual hundredth time stamps will be stored for each new value of the dl register. Also, every seconds worth of hundredth time stamps, a second value will be placed in the buffer. For 60 seconds worth of delay the buffer will store over 1200 time stamps.

The seconds values placed in the buffer do not align with the system time but only show the total accumulated time.

Looking at the stored data can help to troubleshoot problems by helping to determine if the boundary conditions are being included (that is if counting delay increments crossing 99 back to 0 are included). Also, what beginning and ending values of delay work or not.

DISPLAY.INC tools
The include file display.inc has three routines for helping you monitor if your program is working correctly. There is a rotating display wheel which allows you to view each time delay by changing the wheel position. The code below shows how to incorporate the tools into your project:

```
    .Model Small
    .LIST

    .STACK  200
        .DATA

CountSecHdths           PROTO
ResetCountSecHdths      PROTO
Tdelay                  PROTO           ;this is the prototype of your delay routine


        .CODE

INCLUDE DISPLAY.INC                             ;this includes routine used in main

        .STARTUP
Main PROC
LOCAL   WheelPosition:WORD

        mov     bx,WheelPosition
        xor     ax,ax
        mov     [bx],ax
        invoke  ClearScreen, 25*80

LOOPTOP:

        mov     al,50                   ;pass hundreths of seconds
        mov     ah,0                    ;pass seconds
        invoke  Tdelay                  ;this is the delay routine you must define
        invoke  CenterCursor, 0CH,24H
        invoke  RotateWheel, WheelPosition


        jmp     LOOPTOP
Main    ENDP
        .EXIT


        END
```

Using Flags for Debugging and trapping events

A flag variable can be used to either detect if a certain event happened or a path of the code was taken. This is useful for debugging real time code, since the clock can't be stopped for single stepping through the routines.

```
cstPathOne EQU 1
cstPathTwo EQU 2
cstPathThree EQU 4

;Reset flag at top of code
        mov     bx,flgTestPath
        xor     al,al
        mov     [bx],al

;Place flag setting code at critical points in code
        mov     bx,flgTestPath
        mov     al,cstPathOne
        or      [bx],al

        mov     bx,flgTestPath
        mov     al,cstPathTwo
        or      [bx],al
```

Test the flag variable either at the end of the routine or some critical point for breaking code so that you can inspect the current register values. The individual flags can be tested one at a time, then either jmp statements or breakpoints can be set to stop the code for inspection.

```
        mov     bx, flgTestPath
        mov     al,[bx]
        test    al, cstPathOne
        jz      TEST_PATH_TWO
        jmpTRAP_Path_One
TEST_PATH_TWO:
        test    al, cstPathTwo
        jz      TEST_PATH_TWO
        jmpTRAP_Path_Two
```

You may not want to test the flags at all since this may change register values, in which case you can just inspect the flags memory location at the end of your routine or some breakpoint.

Generating a Data Dump Buffer

Data buffers (memory storage locations), are a useful tool for debugging real time programs. An example is the set of fixed address equate definitions shown below (note a data dump buffer is provided in the test code library see "Individual Time Stamps" under the "Test Code Library Provided for the Project" section above):

```
cstDUMP_OFFSET   EQU     2000h
CountSec         EQU cstDUMP_OFFSET - 10h
CountHdths       EQU cstDUMP_OFFSET - 0Eh
flgHdthsLSB_1    EQU cstDUMP_OFFSET - 0Ch
NextHdthsValue   EQU cstDUMP_OFFSET - 0Ah
LastHdthsValue   EQU cstDUMP_OFFSET - 08h

;initialize the dump register (either si, or di would work well for this) at the top of your routine
      mov     si,cstDUMP_OFFSET

;save information to the buffer
       mov     bx,CountSec   ;increment second count
       mov     al,1
       add     [bx],al

       mov     al,[bx]
       mov     [si],al
       inc     si

;after you break your program you can dump out the parameters at 1FF0h as shown in ;this example and the
data in the buffer at 2000h
```