Chapter K

The C-Callable Zen Timer Full Listings

[redacted]

[redacted]

Chapter K

As explained toward the end of Chapter 3, the Zen timer was originally implemented as an assembly language tool, but can be modified so as to be callable from C/C++ code. Instructions are given there for modifying the assembly listings for use with C code. To avoid making Chapter 3 (which is already fairly large) completely unwieldy, the full listings for the C-callable Zen timers have been moved here.

There are two versions of the Zen timer shown here. One, PCZTNEAR.ASM, is for use with C code compiled for the Near code model. The other, PCZTFAR.ASM, (which begins on page 427) is for use with C code compiled for the far code model. Note that both of these versions of the Zen timer are the precision Zen timer; modifying the long-period Zen timer for C code is left as an exercise for the reader.

No special assembly options are required to assemble either program shown here. You should read Chapter 3 thoroughly before attempting to assemble and use this code!

Listing K.1 PCZTNEAR.ASM

```
; ****PCZTNEAR.ASM
; The C-near-callable version of the precision Zen timer
;       (PZTIMER.ASM)
;
; Note: use NOSMART with TASM (at least version 2.0) to keep
;       the assembler from turning far calls in the reference
;       timing code into PUSH CS/near call sequences, thereby
;       messing up the reference call times. This problem may
;       arise with other optimizing assemblers as well.
;
; Uses the 8253 timer to time the performance of code that takes
; less than about 54 ms to execute, with a resolution
; of better than 10 ms.
```

```
;
; By Michael Abrash
;
; Externally callable routines:
;
; ZTimerOn: Starts the Zen timer, with interrupts disabled.
;
; ZTimerOff: Stops the Zen timer, saves the timer count,
;       times the overhead code, and restores interrupts to the
;       state they were in when ZTimerOn was called.
;
; ZTimerReport: Prints the net time that passed between starting
;       and stopping the timer.
;
; Note: If longer than about 54 ms passes between ZTimerOn and
;       ZTimerOff calls, the timer turns over and the count is
;       inaccurate. When this happens, an error message is displayed
;       instead of a count. The long-period Zen timer should be used
;       in such cases.
;
; Note: Interrupts *MUST* be left off between calls to ZTimerOn
;       and ZTimerOff for accurate timing and for detection of
;       timer overflow.
;
; Note: These routines can introduce slight inaccuracies into the
;       system clock count for each code section timed even if
;       timer 0 doesn't overflow. If timer 0 does overflow, the
;       system clock can become slow by virtually any amount of
;       time, since the system clock can't advance while the
;       precison timer is timing. Consequently, it's a good idea
;       to reboot at the end of each timing session. (The
;       battery-backed clock, if any, is not affected by the Zen
;       timer.)
;
; All registers, and all flags except the interrupt flag, are
; preserved by all routines. Interrupts are enabled and then disabled
; by ZTimerOn, and are restored by ZTimerOff to the state they were
; in when ZTimerOn was called.
;

_TEXT   segment word public 'CODE'
        assume  cs:_TEXT, ds:nothing
        public  _ZTimerOn, _ZTimerOff, _ZTimerReport

;
; Base address of the 8253 timer chip.
;
BASE_8253               equ     40h
;
; The address of the timer 0 count registers in the 8253.
;
TIMER_0_8253            equ     BASE_8253 + 0
;
; The address of the mode register in the 8253.
;
MODE_8253               equ     BASE_8253 + 3
;
; The address of Operation Command Word 3 in the 8259 Programmable
; Interrupt Controller (PIC) (write only, and writable only when
; bit 4 of the byte written to this address is 0 and bit 3 is 1).
;
```

```
OCW3                    equ     20h
;
; The address of the Interrupt Request register in the 8259 PIC
; (read only, and readable only when bit 1 of OCW3 = 1 and bit 0
; of OCW3 = 0).
;
IRR                     equ     20h
;
; Macro to emulate a POPF instruction in order to fix the bug in some
; 80286 chips which allows interrupts to occur during a POPF even when
; interrupts remain disabled.
;
MPOPF macro
        local   p1, p2
        jmp short p2
p1:     iret                    ;jump to pushed address & pop flags
p2:     push    cs              ;construct far return address to
        call    p1              ; the next instruction
        endm

;
; Macro to delay briefly to ensure that enough time has elapsed
; between successive I/O accesses so that the device being accessed
; can respond to both accesses even on a very fast PC.
;
DELAY   macro
        jmp     $+2
        jmp     $+2
        jmp     $+2
        endm

OriginalFlags           db      ?       ;storage for upper byte of
                                        ; FLAGS register when
                                        ; ZTimerOn called
TimedCount              dw      ?       ;timer 0 count when the timer
                                        ; is stopped
ReferenceCount          dw      ?       ;number of counts required to
                                        ; execute timer overhead code
OverflowFlag            db      ?       ;used to indicate whether the
                                        ; timer overflowed during the
                                        ; timing interval
;
; String printed to report results.
;
OutputStr       label   byte
                db      'Timed count: ', 5 dup (?)
ASCIICountEnd   label   byte
                db      ' microseconds', 0dh, 0ah
                db      '$'
;
; String printed to report timer overflow.
;
OverflowStr     label   byte
        db      0dh, 0ah
        db      '****************************************************'
        db      0dh, 0ah
        db      '* The timer overflowed, so the interval timed was  *'
        db      0dh, 0ah
        db      '* too long for the precision timer to measure.     *'
        db      0dh, 0ah
        db      '* Please perform the timing test again with the    *'
        db      0dh, 0ah
        db      '* long-period timer.                               *'
        db      0dh, 0ah
        db      '****************************************************'
        db      0dh, 0ah
        db      '$'

;********************************************************************
;* Routine called to start timing.                                  *
;********************************************************************

_ZTimerOn       proc    near

;
; Save the context of the program being timed.
;
        push    ax
        pushf
        pop     ax                      ;get flags so we can keep
                                        ; interrupts off when leaving
                                        ; this routine
        mov     cs:[OriginalFlags],ah   ;remember the state of the
                                        ; Interrupt flag
        and     ah,0fdh                 ;set pushed interrupt flag
                                        ; to 0
        push    ax
;
; Turn on interrupts, so the timer interrupt can occur if it's
; pending.
;
        sti
;
; Set timer 0 of the 8253 to mode 2 (divide-by-N), to cause
; linear counting rather than count-by-two counting. Also
; leaves the 8253 waiting for the initial timer 0 count to
; be loaded.
;
        mov     al,00110100b            ;mode 2
        out     MODE_8253,al
;
; Set the timer count to 0, so we know we won't get another
; timer interrupt right away.
; Note: this introduces an inaccuracy of up to 54 ms in the system
; clock count each time it is executed.
;
        DELAY
        sub     al,al
        out     TIMER_0_8253,al         ;lsb
        DELAY
        out     TIMER_0_8253,al         ;msb
;
; Wait before clearing interrupts to allow the interrupt generated
; when switching from mode 3 to mode 2 to be recognized. The delay
; must be at least 210 ns long to allow time for that interrupt to
; occur. Here, ten jumps are used for the delay to ensure that the
; delay time will be more than long enough even on a very fast PC.
;
        rept 10
        jmp     $+2
        endm
```

```
;
; Disable interrupts to get an accurate count.
;
        cli
;
; Set the timer count to 0 again to start the timing interval.
;
        mov     al,00110100b            ;set up to load initial
        out     MODE_8253,al            ;timer count
        DELAY
        sub     al,al
        out     TIMER_0_8253,al         ;load count lsb
        DELAY
        out     TIMER_0_8253,al         ;load count msb
;
; Restore the context and return.
;
        MPOPF                           ;keeps interrupts off
        pop     ax
        ret

_ZTimerOn       endp

;********************************************************************
;* Routine called to stop timing and get count.                     *
;********************************************************************

_ZTimerOff proc near

;
; Save the context of the program being timed.
;
        push    ax
        push    cx
        pushf
;
; Latch the count.
;
        mov     al,00000000b            ;latch timer 0
        out     MODE_8253,al
;
; See if the timer has overflowed by checking the 8259 for a pending
; timer interrupt.
;
        mov     al,00001010b            ;OCW3, set up to read
        out     OCW3,al                 ; Interrupt Request register
        DELAY
        in      al,IRR                  ;read Interrupt Request
                                        ; register
        and     al,1                    ;set AL to 1 if IRQ0 (the
                                        ; timer interrupt) is pending
        mov     cs:[OverflowFlag],al    ;store the timer overflow
                                        ; status
;
; Allow interrupts to happen again.
;
        sti
;
; Read out the count we latched earlier.
;
```

```
        in      al,TIMER_0_8253         ;least significant byte
        DELAY
        mov     ah,al
        in      al,TIMER_0_8253         ;most significant byte
        xchg    ah,al
        neg     ax                      ;convert from countdown
                                        ; remaining to elapsed
                                        ; count
        mov     cs:[TimedCount],ax
; Time a zero-length code fragment to get a reference for how
; much overhead this routine has. Time it 16 times and average it,
; for accuracy, rounding the result.
;
        mov     cs:[ReferenceCount],0
        mov     cx,16
        cli                             ;interrupts off to allow a
                                        ; precise reference count
RefLoop:
        call    ReferenceZTimerOn
        call    ReferenceZTimerOff
        loop    RefLoop
        sti
        add     cs:[ReferenceCount],8   ;total + (0.5 * 16)
        mov     cl,4
        shr     cs:[ReferenceCount],cl  ;(total) / 16 + 0.5
;
; Restore original interrupt state.
;
        pop     ax                      ;retrieve flags when called
        mov     ch,cs:[OriginalFlags]   ;get back the original upper
                                        ; byte of the FLAGS register
        and     ch,not 0fdh             ;only care about original
                                        ; interrupt flag...
        and     ah,0fdh                 ;...keep all other flags in
                                        ; their current condition
        or      ah,ch                   ;make flags word with original
                                        ; interrupt flag
        push    ax                      ;prepare flags to be popped
;
; Restore the context of the program being timed and return to it.
;
        MPOPF                           ;restore the flags with the
                                        ; original interrupt state
        pop     cx
        pop     ax
        ret

_ZTimerOff endp

;
; Called by ZTimerOff to start timer for overhead measurements.
;

ReferenceZTimerOn       proc    near
;
; Save the context of the program being timed.
;
        push    ax
        pushf           ;interrupts are already off
;
```

```
; Set timer 0 of the 8253 to mode 2 (divide-by-N) to cause
; linear counting rather than count-by-two counting.
;
        mov     al,00110100b    ;set up to load
        out     MODE_8253,al    ; initial timer count
        DELAY
;
; Set the timer count to 0.
;
        sub     al,al
        out     TIMER_0_8253,al ;load count lsb
        DELAY
        out     TIMER_0_8253,al ;load count msb
;
; Restore the context of the program being timed and return to it.
;
        MPOPF
        pop     ax
        ret

ReferenceZTimerOn       endp

;
; Called by ZTimerOff to stop timer and add result to ReferenceCount
; for overhead measurements.
;

ReferenceZTimerOff proc near
;
; Save the context of the program being timed.
;
        push    ax
        push    cx
        pushf
;
; Latch the count and read it.
;
        mov     al,00000000b            ;latch timer 0
        out     MODE_8253,al
        DELAY
        in      al,TIMER_0_8253         ;lsb
        DELAY
        mov     ah,al
        in      al,TIMER_0_8253         ;msb
        xchg    ah,al
        neg     ax                      ;convert from countdown
                                        ; remaining to amount
                                        ; counted down
        add     cs:[ReferenceCount],ax
;
; Restore the context of the program being timed and return to it.
;
        MPOPF
        pop     cx
        pop     ax
        ret

ReferenceZTimerOff endp
```

```
;********************************************************************
;* Routine called to report timing results.                         *
;********************************************************************

_ZTimerReport   proc    near

        pushf
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    ds
;
        push    cs      ;DOS functions require that DS point
        pop     ds      ; to text to be displayed on the screen
        assume  ds:_TEXT
;
; Check for timer 0 overflow.
;
        cmp     [OverflowFlag],0
        jz      PrintGoodCount
        mov     dx,offset OverflowStr
        mov     ah,9
        int     21h
        jmp     short EndZTimerReport
;
; Convert net count to decimal ASCII in microseconds.
;
PrintGoodCount:
        mov     ax,[TimedCount]
        sub     ax,[ReferenceCount]
        mov     si,offset ASCIICountEnd - 1
;
; Convert count to microseconds by multiplying by .8381.
;
        mov     dx,8381
        mul     dx
        mov     bx,10000
        div     bx              ;* .8381 = * 8381 / 10000
;
; Convert time in microseconds to five decimal ASCII digits.
;
        mov     bx,10
        mov     cx,5
CTSLoop:
        sub     dx,dx
        div     bx
        add     dl,'0'
        mov     [si],dl
        dec     si
        loop    CTSLoop
;
; Print the results.
;
        mov     ah,9
        mov     dx,offset OutputStr
        int     21h
;
```

```
EndZTimerReport:
        pop     ds
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        MPOPF
        ret

_ZTimerReport   endp

_TEXT   ends
        end
```

Listing K.2 PCZTFAR.ASM

```
; ****PCZTFAR.ASM
; The C-far-callable version of the precision Zen timer
;       (PZTIMER.ASM)
;
; Uses the 8253 timer to time the performance of code that takes
; less than about 54 milliseconds to execute, with a resolution
; of better than ten microseconds.
;
; By Michael Abrash
;
; Externally callable routines:
;
;  ZTimerOn: Starts the Zen timer, with interrupts disabled.
;
;  ZTimerOff: Stops the Zen timer, saves the timer count,
;       times the overhead code, and restores interrupts to the
;       state they were in when ZTimerOn was called.
;
;  ZTimerReport: Prints the net time that passed between starting
;       and stopping the timer.
;
; Note: If longer than about 54 ms passes between ZTimerOn and
;       ZTimerOff calls, the timer turns over and the count is
;       inaccurate. When this happens, an error message is displayed
;       instead of a count. The long-period Zen timer should be used
;       in such cases.
;
; Note: Interrupts *MUST* be left off between calls to ZTimerOn
;       and ZTimerOff for accurate timing and for detection of
;       timer overflow.
;
; Note: These routines can introduce slight inaccuracies into the
;       system clock count for each code section timed even if
;       timer 0 doesn't overflow. If timer 0 does overflow, the
;       system clock can become slow by virtually any amount of
;       time since the system clock can't advance while the
;       precison timer is timing. Consequently, it's a good idea
;       to reboot at the end of each timing session. (The
;       battery-backed clock, if any, is not affected by the Zen
;       timer.)
;
; All registers, and all flags except the interrupt flag, are
; preserved by all routines. Interrupts are enabled and then disabled
; by ZTimerOn, and are restored by ZTimerOff to the state they were
; in when ZTimerOn was called.
;

PZTIMER_TEXT    segment word public 'CODE'
        assume  cs:PZTIMER_TEXT, ds:nothing
        public  _ZTimerOn, _ZTimerOff, _ZTimerReport

;
; Base address of the 8253 timer chip.
;
BASE_8253               equ     40h
;
; The address of the timer 0 count registers in the 8253.
;
TIMER_0_8253            equ     BASE_8253 + 0
;
; The address of the mode register in the 8253.
;
MODE_8253               equ     BASE_8253 + 3
;
; The address of Operation Command Word 3 in the 8259 Programmable
; Interrupt Controller (PIC) (write only, and writable only when
; bit 4 of the byte written to this address is 0 and bit 3 is 1).
;
OCW3                    equ     20h
;
; The address of the Interrupt Request register in the 8259 PIC
; (read only, and readable only when bit 1 of OCW3 = 1 and bit 0
; of OCW3 = 0).
;
IRR                     equ     20h
;
; Macro to emulate a POPF instruction in order to fix the bug in some
; 80286 chips; this allows interrupts to occur during a POPF even when
; interrupts remain disabled.
;
MPOPF macro
        local   p1, p2
        jmp short p2
p1:     iret                    ;jump to pushed address & pop flags
p2:     push    cs              ;construct far return address to
        call    p1              ; the next instruction
        endm

;
; Macro to delay briefly to ensure that enough time has elapsed
; between successive I/O accesses so that the device being accessed
; can respond to both accesses even on a very fast PC.
;
DELAY   macro
        jmp     $+2
        jmp     $+2
        jmp     $+2
        endm

OriginalFlags           db      ?       ;storage for upper byte of
                                        ; FLAGS register when
                                        ; ZTimerOn called
```

```
TimedCount              dw      ?       ;timer 0 count when the timer
                                        ; is stopped
ReferenceCount          dw      ?       ;number of counts required to
                                        ; execute timer overhead code
OverflowFlag            db      ?       ;used to indicate whether the
                                        ; timer overflowed during the
                                        ; timing interval
;
; String printed to report results.
;
OutputStr       label   byte
                db      'Timed count: ', 5 dup (?)
ASCIICountEnd   label   byte
                db      ' microseconds', 0dh, 0ah
                db      '$'
;
; String printed to report timer overflow.
;
OverflowStr     label   byte
        db      0dh, 0ah
        db      '****************************************************'
        db      0dh, 0ah
        db      '* The timer overflowed, so the interval timed was  *'
        db      0dh, 0ah
        db      '* too long for the precision timer to measure.     *'
        db      0dh, 0ah
        db      '* Please perform the timing test again with the    *'
        db      0dh, 0ah
        db      '* long-period timer.                               *'
        db      0dh, 0ah
        db      '****************************************************'
        db      0dh, 0ah
        db      '$'

;********************************************************************
;* Routine called to start timing.                                  *
;********************************************************************

_ZTimerOn       proc    far

;
; Save the context of the program being timed.
;
        push    ax
        pushf
        pop     ax                      ;get flags so we can keep
                                        ; interrupts off when leaving
                                        ; this routine
        mov     cs:[OriginalFlags],ah   ;remember the state of the
                                        ; Interrupt flag
        and     ah,0fdh                 ;set pushed interrupt flag
                                        ; to 0
        push    ax
;
; Turn on interrupts, so the timer interrupt can occur if it's
; pending.
;
        sti
```

```
;
; Set timer 0 of the 8253 to mode 2 (divide-by-N), to cause
; linear counting rather than count-by-two counting. Also
; leaves the 8253 waiting for the initial timer 0 count to
; be loaded.
;
        mov     al,00110100b            ;mode 2
        out     MODE_8253,al
;
; Set the timer count to 0, so we know we won't get another
; timer interrupt right away.
; Note: this introduces an inaccuracy of up to 54 ms in the system
; clock count each time it is executed.
;
        DELAY
        sub     al,al
        out     TIMER_0_8253,al         ;lsb
        DELAY
        out     TIMER_0_8253,al         ;msb
;
; Wait before clearing interrupts to allow the interrupt generated
; when switching from mode 3 to mode 2 to be recognized. The delay
; must be at least 210 ns long to allow time for that interrupt to
; occur. Here, 10 jumps are used for the delay to ensure that the
; delay time will be more than long enough, even on a very fast PC.
;
        rept 10
        jmp     $+2
        endm
;
; Disable interrupts to get an accurate count.
;
        cli
;
; Set the timer count to 0 again to start the timing interval.
;
        mov     al,00110100b            ;set up to load initial
        out     MODE_8253,al            ; timer count
        DELAY
        sub     al,al
        out     TIMER_0_8253,al         ;load count lsb
        DELAY
        out     TIMER_0_8253,al         ;load count msb
;
; Restore the context and return.
;
        MPOPF                           ;keeps interrupts off
        pop     ax
        ret

_ZTimerOn       endp

;********************************************************************
;* Routine called to stop timing and get count.                    *
;********************************************************************

_ZTimerOff proc far

;
; Save the context of the program being timed.
;
```

```
        push    ax
        push    cx
        pushf
;
; Latch the count.
;
        mov     al,00000000b            ;latch timer 0
        out     MODE_8253,al
;
; See if the timer has overflowed by checking the 8259 for a pending
; timer interrupt.
;
        mov     al,00001010b            ;OCW3, set up to read
        out     OCW3,al                 ; Interrupt Request register
        DELAY
        in      al,IRR                  ;read Interrupt Request
                                        ; register
        and     al,1                    ;set AL to 1 if IRQ0 (the
                                        ; timer interrupt) is pending
        mov     cs:[OverflowFlag],al    ;store the timer overflow
                                        ; status
;
; Allow interrupts to happen again.
;
        sti
;
; Read out the count we latched earlier.
;
        in      al,TIMER_0_8253         ;least significant byte
        DELAY
        mov     ah,al
        in      al,TIMER_0_8253         ;most significant byte
        xchg    ah,al
        neg     ax                      ;convert from countdown
                                        ; remaining to elapsed
                                        ; count
        mov     cs:[TimedCount],ax
; Time a zero-length code fragment to get a reference for how
; much overhead this routine has. Time it 16 times and average it
; for accuracy, rounding the result.
;
        mov     cs:[ReferenceCount],0
        mov     cx,16
        cli                             ;interrupts off to allow a
                                        ; precise reference count
RefLoop:
        call    far ptr ReferenceZTimerOn
        call    far ptr ReferenceZTimerOff
        loop    RefLoop
        sti
        add     cs:[ReferenceCount],8   ;total + (0.5 * 16)
        mov     cl,4
        shr     cs:[ReferenceCount],cl  ;(total) / 16 + 0.5
;
; Restore original interrupt state.
;
        pop     ax                      ;retrieve flags when called
        mov     ch,cs:[OriginalFlags]   ;get back the original upper
                                        ; byte of the FLAGS register
        and     ch,not 0fdh             ;only care about original
                                        ; interrupt flag...
        and     ah,0fdh                 ;...keep all other flags in
                                        ; their current condition
        or      ah,ch                   ;make flags word with original
                                        ; interrupt flag
        push    ax                      ;prepare flags to be popped
;
; Restore the context of the program being timed and return to it.
;
        MPOPF                           ;restore the flags with the
                                        ; original interrupt state
        pop     cx
        pop     ax
        ret

_ZTimerOff endp

;
; Called by ZTimerOff to start timer for overhead measurements.
;

ReferenceZTimerOn       proc    far
;
; Save the context of the program being timed.
;
        push    ax
        pushf           ;interrupts are already off
;
; Set timer 0 of the 8253 to mode 2 (divide-by-N), to cause
; linear counting rather than count-by-two counting.
;
        mov     al,00110100b    ;set up to load
        out     MODE_8253,al    ; initial timer count
        DELAY
;
; Set the timer count to 0.
;
        sub     al,al
        out     TIMER_0_8253,al ;load count lsb
        DELAY
        out     TIMER_0_8253,al ;load count msb
;
; Restore the context of the program being timed and return to it.
;
        MPOPF
        pop     ax
        ret

ReferenceZTimerOn       endp

;
; Called by ZTimerOff to stop timer and add result to ReferenceCount
; for overhead measurements.
;

ReferenceZTimerOff proc far
;
; Save the context of the program being timed.
;
        push    ax
        push    cx
        pushf
```

```
;
; Latch the count and read it.
;
        mov     al,00000000b            ;latch timer 0
        out     MODE_8253,al
        DELAY
        in      al,TIMER_0_8253         ;lsb
        DELAY
        mov     ah,al
        in      al,TIMER_0_8253         ;msb
        xchg    ah,al
        neg     ax                      ;convert from countdown
                                        ; remaining to amount
                                        ; counted down
        add     cs:[ReferenceCount],ax
;
; Restore the context of the program being timed and return to it.
;
        MPOPF
        pop     cx
        pop     ax
        ret

ReferenceZTimerOff endp

;********************************************************************
;* Routine called to report timing results.                         *
;********************************************************************

_ZTimerReport   proc    far

        pushf
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    ds
;
        push    cs      ;DOS functions require that DS point
        pop     ds      ; to text to be displayed on the screen
        assume  ds:PZTIMER_TEXT
;
; Check for timer 0 overflow.
;
        cmp     [OverflowFlag],0
        jz      PrintGoodCount
        mov     dx,offset OverflowStr
        mov     ah,9
        int     21h
        jmp     short EndZTimerReport
;
; Convert net count to decimal ASCII in microseconds.
;
PrintGoodCount:
        mov     ax,[TimedCount]
        sub     ax,[ReferenceCount]
        mov     si,offset ASCIICountEnd - 1
;
; Convert count to microseconds by multiplying by .8381.
;
```

```
        mov     dx,8381
        mul     dx
        mov     bx,10000
        div     bx              ;* .8381 = * 8381 / 10000
;
; Convert time in microseconds to five decimal ASCII digits.
;
        mov     bx,10
        mov     cx,5
CTSLoop:
        sub     dx,dx
        div     bx
        add     dl,'0'
        mov     [si],dl
        dec     si
        loop    CTSLoop
;
; Print the results.
;
        mov     ah,9
        mov     dx,offset OutputStr
        int     21h
;
EndZTimerReport:
        pop     ds
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        MPOPF
        ret

_ZTimerReport   endp

PZTIMER_TEXT    ends
        end
```