```
; Arthur Zatarain 1994
; Ampro PCCLOCk utility - read CMOS clock and set dos time

; Call from pascal READ_SMWATCH to read watch and set dos time
;                  SET_SMWATCH  to write DOS time to Watch
; other values are reserved

TITLE AMZ  Dallas DS1216E Smart Watch Borland Pascal Interface
subttl  APR 4 1994


;        MODEL TPASCAL
CODE    SEGMENT BYTE PUBLIC

        assume cs:code

SET_SMWATCH       PROC NEAR
              PUBLIC SET_SMWATCH
              PUSH DS
              PUSH CS
              POP DS     ; DS now points to CS
              call set_sw_time
              POP DS
              RET
SET_SMWATCH       ENDP


READ_SMWATCH  PROC NEAR
              PUBLIC READ_SMWATCH
              PUSH DS
              PUSH CS
              POP DS     ; DS now points to CS
              call rd_sw_time
              POP DS
READ_SMWATCH ENDP


BIOS_SEG        EQU     0F000H          ; rom address

WRIT_0          EQU     0FFF2H          ; write 0 to SmartWatch
WRIT_1          EQU     0FFF3H          ; write 1 to SmartWatch
READ            EQU     0FFF4H          ; read SmartWatch


        ASSUME CS:CODE,DS:CODE

SMWATCH:              ; main entry point

; get date and time information from DOS and set the SmartWatch

SET_SW_TIME:    MOV     AH,2AH                  ; DOS get DATE function
                INT     21H
                INC     AL                      ; make day of week (1-7)
                MOV     BYTE PTR SW_REG_4,AL    ;
                MOV     AL,DL
                CALL    BIN2BCD
                MOV     BYTE PTR SW_REG_5,AL    ; save day of month (1-31)
                MOV     AL,DH
                CALL    BIN2BCD
                MOV     BYTE PTR SW_REG_6,AL    ; save month (1-12)
                MOV     AX,CX                   ; put year in ax
                SUB     AX,1980                 ; make ax = 0 to 99
                CALL    BIN2BCD
                MOV     BYTE PTR SW_REG_7,AL    ; save year (0 to 99)
```

```
; date information has been installed in smart watch buffer, get
; time from DOS

                MOV     AH,2CH                  ; DOS get time function
                INT     21H
                MOV     AL,DH                   ; convert seconds to bcd
                CALL    BIN2BCD
                MOV     BYTE PTR SW_REG_1,AL
                MOV     AL,CL                   ; convert minutes to bcd
                CALL    BIN2BCD
                MOV     BYTE PTR SW_REG_2,AL
                MOV     AL,CH
                CALL    BIN2BCD                 ; convert hours
                MOV     BYTE PTR SW_REG_3,AL
                JMP     WRT_SW_TIME             ; set the smart watch

; write time information from buffer to dallas smart watch "E"

WRT_SW_TIME:    CLI
                CALL    WAKE_UP                 ; wake up smart watch
                MOV     SI,OFFSET SW_REG_0
                MOV     CX,8                    ; number of bytes in pattern
WRT_NEXT_BYTE:  PUSH    CX
                MOV     CX,8                    ; number of bits in pattern byte
                LODSB                           ; get byte
WRT_BIT:        SHR     AL,1
                JC      PUT_ONE                 ; is one bit
                MOV     AH,ES:[WRIT_0]          ; WRITE zero bit
                JMP     SW_1

PUT_ONE:        MOV     AH,ES:[WRIT_1]          ; WRITE one bit
SW_1:           LOOP    WRT_BIT
                POP     CX
                LOOP    WRT_NEXT_BYTE
                STI
                RET

; read the smart watch time into buffer

RD_SW_TIME:     CLI
                CALL    WAKE_UP                 ; wake up smart watch
                MOV     SI,OFFSET RD_REG_0
                MOV     CX,8
GET_BYTE:       PUSH    CX                      ; save byte count
                MOV     CX,8                    ; number of bits to read
                XOR     AL,AL
GET_BIT:        MOV     AH,ES:[READ]            ; read bit
                SHR     AH,1                    ; rotate bit to carry
                RCR     AL,1                    ; rotate carry bit to al
                LOOP    GET_BIT
                MOV     BYTE PTR [SI],AL
                POP     CX
                INC     SI
                LOOP    GET_BYTE                ; get next byte
                STI

; see if smart watch read was good. register 4 should always be non-zero.
; a 0h or ffh indicates unsucessful read.

                MOV     AL,BYTE PTR RD_REG_4    ; see if reg 4 is non zero
                OR      AL,AL
                JZ      RD_ERR                  ; bit 0 must have been 0
                CMP     AL,0FFH
                JZ      RD_ERR                  ; bit 0 must have been 1
                MOV     AL,0                    ; show good results
                RET

; if register 4 is 0 or 0ffh the clock was not read sucessfully

RD_ERR:       ret


; wake up the smart watch. also initialize ES to point to ROM memory
; location.

WAKE_UP:        MOV     AX,BIOS_SEG             ;
                MOV     ES,AX                   ; es points to rom location
                MOV     AH,ES:[READ]            ; reset the smart clock
                MOV     SI,OFFSET WAKE_UP_PAT
                MOV     CX,8                    ; number of bytes in pattern
                CLD
WAKE_BYTE:      PUSH    CX
                MOV     CX,8                    ; number of bits in pattern byte
                LODSB
WAKE_BIT:       SHR     AL,1
                JC      IS_ONE                  ; is one bit
                MOV     AH,ES:[WRIT_0]          ; WRITE zero bit
                JMP     WAKE_1

IS_ONE:         MOV     AH,ES:[WRIT_1]          ; WRITE one bit
WAKE_1:         LOOP    WAKE_BIT
                POP     CX
                LOOP    WAKE_BYTE               ; another byte
                RET

; convert the hex byte in al to bcd, max value passed = bcd 99
; returns 'al' = bcd value

BIN2BCD:        PUSH    BX
                PUSH    CX
                MOV     BL,-1                   ; convert cx to bcd
DO10:           INC     BL                      ; start cnt
                SUB     AL,10                   ; subtract 10 till negitive
                JNC     DO10
                ADD     AL,10                   ; restore cl positive
                MOV     CL,4
                SHL     BL,CL
                OR      AL,BL                   ; or in 10's count
                POP     CX
                POP     BX
                RET

; convert the bcd byte in 'al' to binary value returned in 'al'

BCD2BIN:        PUSH    BX
                PUSH    CX
                MOV     AH,AL                   ; save value
                AND     AL,0F0H                 ; mask upper nibble
                SHR     AL,1                    ; shift right 1 bit
                MOV     BL,AL                   ; bl=upper nibble * 8
                MOV     CL,2
                SHR     AL,CL                   ; al= upper nibble * 2
                ADD     AL,BL                   ;
                MOV     BL,AL                   ; bl= upper nibble *(8+2)
                MOV     AL,AH                   ; get binary value back
                AND     AL,0FH                  ; mask lower nibble
                ADD     AL,BL                   ; add to binary upper nibble
                POP     CX
                POP     BX
                RET
```

```
; string to wake up smart watch

WAKE_UP_PAT:    DB      0C5H,03AH,0A3H,05CH,0C5H,03AH,0A3H,05CH

; SmartWatch register storage for setting time

SW_REG_0:       DB      0                       ; .1, .01 sec (0 to 99)
SW_REG_1:       DB      0                       ; 10 sec, seconds (00 to 59)
SW_REG_2:       DB      0                       ; 10 min, minutes (00 to 59)
SW_REG_3:       DB      0                       ; hour (00 to 23)
SW_REG_4:       DB      0                       ; day of week (1 to 7)
SW_REG_5:       DB      0                       ; date (01 to 31)
SW_REG_6:       DB      0                       ; month (01 to 12)
SW_REG_7:       DB      0                       ; year (00 to 99)

; SmartWatch register storage for reading time

RD_REG_0:       DB      0                       ; same as above
RD_REG_1:       DB      0
RD_REG_2:       DB      0
RD_REG_3:       DB      0
RD_REG_4:       DB      0
RD_REG_5:       DB      0
RD_REG_6:       DB      0
RD_REG_7:       DB      0

CODE            ENDS
                END
```