

```

*****
;*                               T S R P A                               *;
;*-----*
;* Task       : Assembler module for a Turbo Pascal program          *;
;*             demonstrating TSR access through hotkeys.              *;
;*-----*
;* Author      : Michael Tischer                                       *;
;* Developed on : 08/12/88                                              *;
;* Last update  : 02/04/92                                             *;
;*-----*
;* Info        : This module must be linked to a program, not        *;
;*              added to a unit.                                       *;
;*-----*
;* Assembly    : MASM TSRPA;      or      TASM TSRPA                  *;
;*              Then link with a Turbo Pascal program                 *;
;*****

DATA    segment word public          ;Turbo data segment

DATA    ends                        ;End of data segments

;== Constants =====

I2F_FCT_0 equ 0AAh                  ;Code for INT 2F, function 0
I2F_FCT_1 equ 0BBh                  ;Code for INT 2F, function 1
TIME_OUT equ 9                      ;Time out for activation in ticks

;== Program =====

CODE    segment byte public          ;Turbo code segment

        assume cs:CODE, ds:DATA

;-- Public declarations of internal functions -----

public  tsrinit                      ;Enable calls within Turbo program
public  tsrisinst
public  tsruninst
public  tsrcanuninst
public  tsrsetptr
public  tsrcall
public  tsrsethotkey

;-- Interrupt handler variables -----
;-- (accessible to code segment only) -----

call_ptr equ this dword
call_ofs  dw 0                      ;Offset address for TSRCall
call_seg  dw 0                      ;Segment address not initialized yet

ds_save   dw 0                      ;Store DS during TSR call
rp_ptr_save equ this dword          ;FAR pointer for return from TSR call
rip_save  dw 0                      ;Store return address
rscs_save dw 0                      ;from TSR call
ret_ax    dw 0                      ;Store function result
ret_dx    dw 0                      ;from TSR call

;-- Variables needed for activating the Turbo program -----

t_ss      dw 0                      ;Turbo stack segment
t_sp      dw 0                      ;Turbo stack pointer
t_ds      dw 0                      ;Turbo data segment
t_es      dw 0                      ;Turbo extra segment

t_dta_ofs dw 0                      ;DTA address of Turbo programs
t_dta_seg dw 0

t_psp     dw 0                      ;PSP segment address of Turbo prg.
prc_adr   dw 0                      ;Address of Turbo TSR procedure

;-- Variables used for testing hotkeys -----

key_mask  dw 3                      ;Hotkey mask for BIOS keyboard flag
;Default: Alt + H
sc_code   db 128                    ;Hotkey scan code
;Default: No key
i2f_code  db 0                      ;Function number for INT 2F

;-- Variables for TSR activation -----

tsrnow    db 0                      ;Is TSR waiting for activation?
tsractive db 0                      ;Is TSR already active?
in_bios   db 0                      ;Display BIOS disk activity

daptr     equ this dword            ;Pointer to the DOS Indos flag
daptr_ofs dw 0                      ;Offset address

```

```

daptr_seg dw 0 ;Segment address
;-- The following variables store the old interrupt handler ---
;-- addresses, which are replace by the new interrupt handler ---

int8_ptr equ this dword ;Old interrupt vector 8H
int8_ofs dw 0 ;Offset address of the old handler
int8_seg dw 0 ;Segment address of the old handler

int9_ptr equ this dword ;Old interrupt vector 9H
int9_ofs dw 0 ;Offset address of the old handler
int9_seg dw 0 ;Segment address of the old handler

int13_ptr equ this dword ;Old interrupt vector 13H
int13_ofs dw 0 ;Offset address of the old handler
int13_seg dw 0 ;Segment address of the old handler

int28_ptr equ this dword ;Old interrupt vector 28H
int28_ofs dw 0 ;Offset address of the old handler
int28_seg dw 0 ;Segment address of the old handler

int2F_ptr equ this dword ;Old interrupt vector 2FH
int2F_ofs dw 0 ;Offset address of the old handler
int2F_seg dw 0 ;Segment address of the old handler

;-- Variables for storing information about the interrupted program ---

u_dta_ofs dw 0 ;DTA address of interrupted program
u_dta_seg dw 0

u_psp dw 0 ;Segment address: Interrupted prg PSP

uprg_ss dw 0 ;SS and SP of interrupted program
uprg_sp dw 0

;-----
;-- TSRINIT: Ends the Turbo program and activates the new interrupt ---
;-- handler
;-- Call from Turbo: procedure TsrInit( PrcPtr : word;
;-- ResPara : word );

tsrinit proc near

sframe0 struc ;Access structure from stack
bp0 dw ? ;Gets BP
ret_adr0 dw ? ;Return address
respara0 dw ? ;Number of paragraphs to be reserved
przp0 dw ? ;Offset of Turbo TSR procedure
sframe0 ends ;End structure

frame equ [ bp - bp0 ]

push bp ;Push BP onto stack
mov bp,sp ;Move SP to BP
push es ;Push ES onto stack

;-- Store Turbo segment register -----

mov t_ss,ss ;Store registers in their
mov t_sp,sp ;corresponding variables
mov t_es,es
mov t_ds,ds

;-- Get PSP for Turbo program -----

mov bx,cs ;Move CS to BX
sub bx,10h ;10H paragraphs = 256 bytes
mov t_psp,bx ;Store segment address

;-- Store specified parameters -----

mov ax,frame.przp0 ;Get pointer to TSR procedure
mov prc_adr,ax ;and store it

;-- Get DTA address of Turbo program -----

mov ah,2fh ;Funct. no.: Get DTA address
int 21h ;Call DOS interrupt
mov t_dta_ofs,bx ;Store address in
mov t_dta_seg,es ;corresponding variables

;-- Get address of INDOS flag -----

mov ah,34h ;Funct. no.: Get INDOS flag address
int 21h ;Call DOS interrupt
mov daptr_ofs,bx ;Store address in

```

```

mov     daptr_seg,es           ;corresponding variables
;-- Get addresses of interrupt handler to be changed -----

mov     ax,3508h               ;Get interrupt vector 8H
int     21h                   ;Call DOS interrupt
mov     int8_ofs,bx            ;Store handler address in
mov     int8_seg,es           ;corresponding variables

mov     ax,3509h               ;Get interrupt vector 9H
int     21h                   ;Call DOS interrupt
mov     int9_ofs,bx            ;Store handler address in
mov     int9_seg,es           ;corresponding variables

mov     ax,3513h               ;Get interrupt vector 13H
int     21h                   ;Call DOS interrupt
mov     int13_ofs,bx           ;Store handler address in
mov     int13_seg,es          ;corresponding variables

mov     ax,3528h               ;Get interrupt vector 28H
int     21h                   ;Call DOS interrupt
mov     int28_ofs,bx           ;Store handler address in
mov     int28_seg,es          ;corresponding variables

mov     ax,352Fh               ;Get interrupt vector 2FH
int     21h                   ;Call DOS interrupt
mov     int2F_ofs,bx           ;Store handler address in
mov     int2F_seg,es          ;corresponding variables

;-- Install new interrupt handler -----

push    ds                    ;Store data segment
mov     ax,cs                 ;Move CS to AX and pass to DS
mov     ds,ax

mov     ax,2508h               ;Funct. no.: Set interrupt 8H
mov     dx,offset int08        ;DS:DX contains the handler address
int     21h                   ;Call DOS interrupt

mov     ax,2509h               ;Funct. no.: Set interrupt 9H
mov     dx,offset int09        ;DS:DX contains the handler address
int     21h                   ;Call DOS interrupt

mov     ax,2513h               ;Funct. no.: Set interrupt 13H
mov     dx,offset int13        ;DS:DX contains the handler address
int     21h                   ;Call DOS interrupt

mov     ax,2528h               ;Funct. no.: Set interrupt 28H
mov     dx,offset int28        ;DS:DX contains the handler address
int     21h                   ;Call DOS interrupt

mov     ax,252Fh               ;Funct. no.: Set interrupt 2FH
mov     dx,offset int2F        ;DS:DX contains the handler address
int     21h                   ;Call DOS interrupt

pop     ds                    ;DS from stack

;-- End resident program -----

mov     ax,3100h               ;Funct. no.: End resident program
mov     dx,frame.respara0      ;Get number of reserved paragraphs
int     21h                   ;Call DOS interrupt and end program

```

tsrinit endp

```

;-----
;-- TSRSETHOTKEY: Configures program hotkey
;-- Call from Turbo: procedure TsrSetHotKey( KeyMask : word;
;--                                     ScanCode : byte );
;-- Info : This procedure is FAR, so that it can be called
;--       from an already installed TSR.
;--

```

tsrsethotkey proc far

```

sframe1  struc                ;Access structure from stack
bp1      dw ?                 ;Gets BP
ret_adr1 dd ?                 ;Return address
sc_code1 dw ?                 ;Scan code of hotkey
keymask1 dw ?                 ;Mask for hotkey
sframe1  ends                ;End structure

```

frame equ [bp - bp1]

```

push    bp                    ;Push BP onto stack
mov     bp,sp                 ;Move SP to BP

```

```

;-- Save passed parameters -----
mov ax,frame.keymask1 ;Get and store
mov key_mask,ax ;hotkey
mov al,byte ptr frame.sc_code1 ;Get and store
mov sc_code,al ;hotkey's scan code

pop bp ;Pop BP from stack
ret 4 ;Return

tsrsethotkey endp

;-----
;-- TSRISINST: Determines whether program is already installed ----
;-- Call from Turbo: function TsrIsInst( i2f_fctno : byte ) : boolean;
;-- Return value: TRUE if the program is already installed,
;-- otherwise FALSE

tsrisinst proc near

sframe2 struc ;Access structure from stack
bp2 dw ? ;Gets BP
ret_adr2 dw ? ;Return address
i2F_code2 dw ? ;Function number for INT 2F
sframe2 ends ;End structure

frame equ [ bp - bp2 ]

push bp ;Push BP onto stack
mov bp,sp ;Move SP to BP

mov ah,byte ptr frame.i2F_code2 ;Func. no. for INT 2F
mov i2F_code,ah ;Store it
mov al,I2F_FCT_0 ;Sub-function
mov bx,ax ;Store both numbers
int 2Fh
xchg bh,bl ;Exchange numbers
cmp ax,bx ;Compare with result
mov ax,0 ;Not from installation
jne isi_end ;Unequal --> Not installed

;-- Get segment address of already installed copy

mov ah,i2f_code ;No --> Segment address of INT 2Fh
mov al,I2F_FCT_1 ;Load sub-function 01H
int 2Fh
mov call_seg,ax ;and store in variables
mov ax,-1 ;if installed

isi_end: pop bp ;Pop BP from stack
ret 2 ;Return to caller

tsrisinst endp ;End procedure

;-----
;-- TSRCANUNIST: Determines whether installed copy of TSR can be ----
;-- reinstalled or uninstalled.
;-- Call from Turbo : function TsrCanUnInst : boolean;
;-- Output : TRUE if reinstallation possible, otherwise FALSE
;-- Info : Program can only be reinstalled if none of the
;-- interrupt vectors for the program have been
;-- redirected to another program.

tsrlist db 08h,09h,13h,28h,2Fh,00h ;List of redirected INTs
;00H indicates end

tsrcanuninst proc near

mov dx,call_seg ;Load segment of installed copy
mov di,offset tsrlist-1 ;Move DI to list

tcu_1: inc di ;Increment DI to next int number
mov al,cs:[di] ;Move next int number to AL
or al,al ;End of list reached?
je tcu_ok ;Yes --> All vectors O.K.

mov ah,35h ;Funct. no.: Get interrupt
int 21h ;Call DOS interrupt
mov cx,es ;Compare ES to CX
cmp dx,cx ;Still in same segment?
je tcu_1 ;Yes --> No reinstallation or
; uninstallation possible
xor ax,ax ;No --> No reinstallation or
; uninstallation possible
ret

```

```

tcsrsetptr: mov ax,-1
            ret

tsrcanuninst endp

;-----
;-- TSRUNINST: Reinstalls the TSR and releases the allocated memory. -
;-- Call from Turbo : procedure TsrUnInst;
;-- Info          : TSRCANUNIST() must be called successfully before
;--                  calling this routine

tsruninst  proc  near

            push ds
            mov  es,call_seg      ;Load segment of installed TSR

            ;-- Reinstall TSR program's interrupt handler -----

            cli                  ;Disable interrupts
            mov  ax,2508h         ;Funct. no.: Set INT 8 handler
            mov  ds,es:int8_seg   ;Segment address of the old handler
            mov  dx,es:int8_ofs   ;Offset address of the old handler
            int  21h              ;Reinstall old handler

            mov  ax,2509h         ;Funct. no.: Set INT 9 handler
            mov  ds,es:int9_seg   ;Segment address of the old handler
            mov  dx,es:int9_ofs   ;Offset address of the old handler
            int  21h              ;Reinstall old handler

            mov  ax,2513h         ;Funct. no.: Set INT 13 handler
            mov  ds,es:int13_seg  ;Segment address of the old handler
            mov  dx,es:int13_ofs  ;Offset address of the old handler
            int  21h              ;Reinstall old handler

            mov  ax,2528h         ;Funct. no.: Set INT 28 handler
            mov  ds,es:int28_seg  ;Segment address of the old handler
            mov  dx,es:int28_ofs  ;Offset address of the old handler
            int  21h              ;Reinstall old handler

            mov  ax,252Fh         ;Funct. no.: Set INT 2F handler
            mov  ds,es:int2F_seg  ;Segment address of the old handler
            mov  dx,es:int2F_ofs  ;Offset address of the old handler
            int  21h              ;Reinstall old handler

            ;-- Release memory -----

            sti                  ;Enable interrupts

            mov  es,es:t_psp      ;Store PSP segment address of TSR
            mov  cx,es            ;program in CX
            mov  es,es:[ 02ch ]   ;Store environment segment address
            mov  ah,49h           ;Funct. no.: Release allocated memory
            int  21h              ;Call DOS interrupt

            mov  es,cx            ;Get ES from CX
            mov  ah,49h           ;Funct. no.: Release allocated mem.
            int  21h              ;Call DOS interrupt

            pop  ds               ;Pop DS from stack
            ret                  ;Return to caller

tsruninst  endp                  ;End of procedure

;-----
;-- TSRSETPTR: Stores the address of the routine from which TSRCALL --
;--                  should be called
;-- Call from Turbo: procedure TsrSetPtr( offset : word );

tsrsetptr  proc  near

sframe3    struc                ;Access structure from stack
bp3         dw ?                ;Gets BP
ret_adr3    dw ?                ;Return address
offset3     dw ?                ;Offset of routine to be called
sframe3     ends                ;End structure

frame      equ [ bp - bp3 ]

            push bp              ;Push BP onto stack
            mov  bp,sp           ;Move SP to BP

            mov  ax,frame.offset3 ;Move offset address to AX
            mov  call_ofs,ax      ;and store it

            pop  bp              ;Pop BP from stack

```

```

ret 2 ;Return to caller
tsrsetptr endp ;End of procedure

;-----
;-- TSRCALL: Calls a routine in the previously installed copy of ----
;-- the TSR program.
;-- Call from Turbo : procedure TsrCall;
;-- Attention : - The stack should not be altered in this
;-- routine.
;-- - this procedure must be declared as FAR,
;-- making it accessible as a procedure pointer.

tsrcall proc far

;-- Execute context change to Turbo program and call the ---
;-- appropriate procedure

pop rip_save ;Pop return address from stack and
pop rcs_save ;store it
mov ds_save,ds ;Store DS

mov ah,2fh ;Funct. no.: Get DTA address
int 21h ;Call DOS interrupt
mov u_dta_ofs,bx ;Store DTA address of
mov u_dta_seg,es ;interrupted program

mov es,call_seg ;Pass segment address
;of installed TSR to ES
mov ah,50h ;Funct. no.: Set PSP address
mov bx,es:t_psp ;Get PSP segment address
int 21h ;Call DOS interrupt

mov ah,1ah ;Funct. no.: Set DTA address
mov dx,es:t_dta_ofs ;Get offset address of new DTA and
mov ds,es:t_dta_seg ;segment address of new DTA
int 21h ;Call DOS interrupt

mov ds,es:t_ds ;Set segment register for
mov es,es:t_es ;Turbo program

call [call_ptr] ;Call Turbo procedure
mov cs:ret_ax,ax ;Store function result
mov cs:ret_dx,dx

;-- Context change returning to Turbo program -----

mov ah,1ah ;Funct. no.: Set DTA address
mov dx,u_dta_ofs ;Get DTA offset and segment address
mov ds,u_dta_seg ;Load interrupted program
int 21h ;Call DOS interrupt

mov es,call_seg ;Return ES and
mov ds,ds_save ;DS

mov ah,50h ;Funct. no.: Set PSP address
mov bx,cs ;Move CS to BX
sub bx,10h ;Calculate segment address of PSP
int 21h ;Call DOS interrupt

mov ax,cs:ret_ax ;Return function result
mov dx,cs:ret_dx
jmp [rptr_save] ;Return to caller

tsrcall endp ;End of procedure

;-----
;-- DOSACTIVE: Determines whether DOS will be interrupted, by checking
;-- the INDOS flag.
;-- Input : None
;-- Output : Zero flag=1 : DOS may be interrupted

dosactive proc near

push ds ;Push DS and BX onto stack
push bx
lds bx,daptr ;DS:BX point to the INDOS flag
cmp byte ptr [bx],0 ;DOS function active?
pop bx ;Pop BX and DS from stack
pop ds

ret ;Return to caller

dosactive endp

;-----

```

```

; New interrupt handler follows -----
;-----
;-- New interrupt 08H handler (Timer) -----

int08      proc far

            cmp     tsrnow,0           ;Should TSR be activated?
            je      i8_end             ;No --> Return to old handler

            dec     tsrnow             ;Yes --> Decrement activation flag

            ;-- TSR should be activated, but is it possible? -----

            cmp     in_bios, 0         ;BIOS disk interrupt already active?
            jne     i8_end             ;Yes --> No activation possible

            call    dosactive          ;Should DOS be interrupted?
            je      i8_tsr             ;Yes --> Call TSR

i8_end:     jmp     [int8_ptr]         ;Jump to old handler

            ;-- Activate TSR -----

i8_tsr:     mov     tsrnow,0           ;TSR no longer waiting for activation
            mov     tsractive,1       ;TSR is active
            pushf                     ;Call old handler, using
            call    [int8_ptr]         ;INT 8H emulation
            call    start_tsr         ;Start TSR program
            iret                     ;Return to interrupted program

int08      endp

;-- New interrupt 09H handler (Keyboard) -----

int09      proc far

            push    ax
            in      al,60h             ;Read keyboard port

            cmp     tsractive,0       ;Is TSR program already active?
            jne     i9_end             ;Yes --> Call old handler then return

            cmp     tsrnow,0          ;TSR waiting for activation?
            jne     i9_end             ;Yes --> Call old handler then return

            ;-- Test for hotkey -----

            cmp     sc_code,128       ;Scan code?
            je      i9_ks             ;No --> Check for toggle keys

            cmp     al,128             ;Yes --> Is it a release code?
            jae     i9_end             ;Yes --> Not a hotkey

            cmp     sc_code,al        ;Make code, compare with key
            jne     i9_end             ;No match --> No activation

i9_ks:      ;-- Check status of toggle keys -----

            push    ds
            mov     ax,040h           ;Move DS to ROM-BIOS
            mov     ds,ax             ;variable segment
            mov     ax,word ptr ds:[17h] ;Get BIOS keyboard flag
            and     ax,key_mask       ;Mask non-hotkey bits
            cmp     ax,key_mask       ;Any hotkey bits remaining?
            pop     ds
            jne     i9_end             ;Hotkey implemented? No --> Return

            cmp     in_bios, 0         ;BIOS disk interrupt active?
            jne     i9_e1             ;Yes --> No activation possible

            call    dosactive          ;Should DOS be interrupted?
            je      i9_tsr             ;Yes --> Start TSR

i9_e1:     mov     tsrnow,TIME_OUT;TSR waits for activation

i9_end:     pop     ax                ;Pop AX from stack
            jmp     [int9_ptr]         ;Jump to old handler

i9_tsr:     mov     tsractive,1       ;TSR now active
            mov     tsrnow,0          ;No delayed start wanted
            pushf                     ;Call old handler
            call    [int9_ptr]         ;Call old handler
            pop     ax                ;Pop AX from stack
            call    start_tsr         ;Start TSR program
            iret                     ;Return to interrupted program

```

```

int09      endp

;-- New interrupt 13H handler (diskette/hard drive) -----

int13      proc far

            inc  in_bios          ;Increment BIOS disk flag
            pushf                 ;Call old interrupt handler
            call [int13_ptr]      ;using INT 13H emulation
            dec  in_bios          ;Reset BIOS disk flag

            sti                   ;Enable interrupts
            ret  2                ;Return to caller without popping
                                   ;flag register from stack

int13      endp

;-- New interrupt 28H handler (DOS idle) -----

int28      proc far

            cmp  tsrnow,0         ;Is TSR waiting for activation?
            je   i28_end         ;No --> Return to caller

            cmp  in_bios, 0       ;Yes --> But is disk int active?
            je   i28_tsr        ;Yes --> No activation

i28_end:    jmp  [int28_ptr]      ;Return to old handler

            ;-- Start TSR -----

i28_tsr:    mov  tsrnow,0         ;TSR no longer waiting for activation
            mov  tsractive,1     ;TSR is (already) active
            pushf                 ;Call old interrupt handler
            call [int28_ptr]      ;using INT 28H emulation
            call start_tsr       ;Start TSR program
            iret                 ;Return to caller

int28      endp

;-- New interrupt 2FH handler (Multiplexer) -----

int2F      proc far

            cmp  ah,i2F_code      ;Call for this TSR?
            jne  i2F_end         ;No --> Revert to old handler

            cmp  al,I2F_FCT_0     ;Yes --> Is this sub-function 00H?
            je   i2F_0          ;Yes --> Execute

            cmp  al,I2F_FCT_1     ;Or is it sub-function 01H?
            je   i2F_1          ;Yes --> Execute

            iret                 ;No --> Ignore call

i2F_end:    ;-- TSR not planned, send call on -----

            jmp  [int2F_ptr]      ;Revert to old handler

i2F_0:      ;-- Sub-function 00: Installation check -----

            xchg ah,al           ;Exchange function and sub-funct. no.
            iret                 ;Return to caller

i2F_1:      ;-- Sub-function 01: Return segment address -----

            mov  ax,cs           ;Move segment address to AX
            iret                 ;Return to caller

int2F      endp

;-- START_TSR: Activate TSR program -----

start_tsr  proc near

            ;-- Execute context change to Turbo program -----

            cli                   ;Disable interrupts
            mov  uprg_ss,ss      ;Store current stack segment and
            mov  uprg_sp,sp      ;stack pointer

            mov  ss,t_ss        ;Activate Turbo program's
            mov  sp,t_sp        ;stack
            sti                   ;Enable interrupts

            push ax              ;Store processor registers

```



```

push bx ;for the Turbo stack
push cx
push dx
push bp
push si
push di
push ds
push es

;-- Store 64 words from the DOS stack -----
mov cx,64 ;Loop counter
mov ds,uprg_ss ;DS:SI indicates end of DOS stack
mov si,uprg_sp

tsrs1: push word ptr [si] ;Push word from DOS stack to TP stack
inc si ;and make SI the next stack
inc si ;word
loop tsrs1 ;Process all 64 words

mov ah,51h ;Funct. no.: Get PSP address
int 21h ;Call DOS interrupt
mov u_psp,bx ;Store segment address of PSP

mov ah,2fh ;Funct. no.: Get DTA address
int 21h ;Call DOS interrupt
mov u_dta_ofs,bx ;Store DTA address of the
mov u_dta_seg,es ;interrupted program

mov ah,50h ;Funct. no.: Set PSP address
mov bx,t_psp ;Get PSP segment addr. of Turbo prog.
int 21h ;Call DOS interrupt

mov ah,1ah ;Funct. no.: Set DTA address
mov dx,t_dta_ofs ;Get offset address and
mov ds,t_dta_seg ;segment address of new DTA
int 21h ;Call DOS interrupt

mov ds,t_ds ;Set segment register for
mov es,t_es ;Turbo program

call [prc_adr] ;Call start function

;-- Execute context change to interrupted program -----
mov ah,1ah ;Funct. no.: Set DTA address
mov dx,u_dta_ofs ;Load DTA offset and segment addrs.
mov ds,u_dta_seg ;for interrupted program
int 21h ;Call DOS interrupt

mov ah,50h ;Funct. no.: Set PSP address
mov bx,u_psp ;PSP segment address: Interrupted prg
int 21h ;Call DOS interrupt

;-- Restore DOS stack -----
mov cx,64 ;Loop counter
mov ds,uprg_ss ;Load DS:SI with end
mov si,uprg_sp ;address of DOS stack
add si,128 ;Set SI to start of DOS stack
dec si ;SI to previous stack word
dec si
pop word ptr [si] ;Pop word from Turbo stack to DOS
loop tsrs2 ;Process all 64 words

pop es ;Pop stored registers from
pop ds ;Turbo stack
pop di
pop si
pop bp
pop dx
pop cx
pop bx
pop ax

cli ;Disable interrupts
mov ss,uprg_ss ;Move stack pointer and stack segment
mov sp,uprg_sp ;of interrupted program

mov tsractive,0 ;TSR no longer active
sti ;Enable interrupts

ret ;Return to caller

start_tsr endp

```

```
,-----  
CODE      ends      ;End of code segment  
          end        ;End of program
```