

```

;*****;
;*                      T S R C A                      *;
;*-----*
;* Task                : Assembler module for a C program demonstrating*
;*                      TSR access through hotkeys.                  *;
;*-----*
;* Author              : Michael Tischer                          *;
;* Developed on        : 08/10/88                                  *;
;* Last update         : 02/18/92                                  *;
;*-----*
;* Assembly           : MASM -mx TSRCA;   or   TASM /mx TSRCA;      *;
;*                      ... link with C program                    *;
;*****;

```

```

IGROUP group _text           ;Include program segment
DGROUP group _bss, _data     ;Include data segment
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

```

```

_BSS    segment word public 'BSS' ;Include all un-initialized
_BSS    ends                      ;static variables

```

```

_DATA   segment word public 'DATA' ;Include all initialized
                                   ;global and static variables

```

```

extrn   __psp : word            ;Segment address of C program PSP

```

```

_DATA   ends

```

```

;== Constants =====

```

```

TC_STACK    equ 512            ;Turbo C stack
ARG_WORDS   equ 32             ;Stack words copied by TsrCall

```

```

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

;== Program =====

_TEXT segment byte public 'CODE' ;Program segment

;-- Reference to external (C) functions -----

extrn _GetHeapEnd:near ;Returns ending address of heap

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

public _TsrInit ;Enable calls within C program
public _TsrIsInst
public _TsrUnInst
public _TsrCanUnInst
public _TsrCall
public _TsrSetHotkey
public _TsrSetPtr

;-- 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

ret_ax dw 0 ;Store function result
ret_dx dw 0 ;from TsrCall

```

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

c_ss	dw 0	;C stack segment
c_sp	dw 0	;C stack pointer
c_ds	dw 0	;C data segment
c_es	dw 0	;C extra segment
c_dta_ofs	dw 0	;DTA address of C program
c_dta_seg	dw 0	
c_psp	dw 0	;PSP segment address of C program
break_adr	dw 0	;Break address of heap
fct_adr	dw 0	;Address of C TSR function

;-- 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 2FH

;-- 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 replaced by the new interrupt handler  ---
```

```
uprg_sp      dw 0
```

```
-----  
;-- TSRINIT: Ends the C program and activates the new interrupt -  
;-- handler  
;-- Call from C: void TsrInit( bool TC,  
;-- void (fct *)(void),  
;-- unsigned heap_byte );
```

```
_TsrInit proc near
```

```
sframe0      struc                ;Access structure from stack  
bp0           dw ?                ;Gets BP  
ret_adr0      dw ?                ;Return address  
tc0           dw ?                ;Compiler (1 = TURBO C, 0 = MSC )  
fctptr0       dw ?                ;Pointer to C TSR function  
heap0         dw ?                ;Heap bytes needed  
sframe0       ends               ;End structure
```

```
frame        equ [ bp - bp0 ]
```

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

```
;-- Store C segment register -----
```

```
mov c_ss,ss   ;Store registers in their  
mov c_sp,sp   ;corresponding variables  
mov c_es,es  
mov c_ds,ds
```

```
;-- Store specified parameters -----
```

```

mov  ax,frame.fctptr0    ;Get pointer to TSR function
mov  fct_adr,ax          ;and store it

;-- Get DTA address of C program -----

mov  ah,2fh              ;Funct. no.: Get DTA address
int  21h                 ;Call DOS interrupt
mov  c_dta_ofs,bx        ;Store address in
mov  c_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                   ;Pop DS from stack

;-- Compute number of paragraphs -----
;-- that must remain in memory -----

call _GetHeapEnd          ;Call C function in TSR module
add  ax,frame.heap0       ;Add necessary heap memory

;-- Since Turbo C locates the stack after the heap, then --
;-- begins with the end of segment, the heap must be      --
;-- determined first.                                     --

cmp  byte ptr frame.tc0,0 ;Turbo C used?
je   msc2                 ;No --> Microsoft C

add  ax,TC_STACK-1        ;Compute new stack pointer for TC
mov  c_sp,ax              ;and store it
inc  ax                   ;Set break address

;-- Compute the number of paragraphs -----
;-- that must be resident in memory -----

msc2:  mov  dx,ax           ;Move break address to DX
       add  dx,15          ;Avoid loss through integer division
       mov  cl,4           ;Shift 4 times to the right and then

```



```

    shr     dx,c1             ;divide by 16
    mov     ax,ds             ;Move DS to AX
    mov     bx,__psp         ;Get PSP segment address
    mov     c_psp,bx         ;Store in a corresponding variable
    sub     ax,bx             ;Subtract DS from PSP
    add     dx,ax             ;and add to the number of paragraphs
    mov     ax,3100h         ;Funct. no.: End resident program
    int     21h              ;Call DOS interrupt and end program

```

```

_TsrInit     endp

```

```

        assume CS:IGROUP, DS:nothing, ES:nothing, SS:nothing

```

```

;-----
;-- TSRSETHOTKEY: Configures program hotkey
;-- Call from C: void TsrSetHotKey( unsigned KeyMask,
;--                               byte    ScanCode );
;-- 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_adrl     dd ?                 ;Return address
keymask1     dw ?                 ;Mask for hotkey
sc_codel     dw ?                 ;Scan code of 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                          ;Return to caller

```

\_TsrSetHotkey endp

```

;-----
;-- TSRISINST: Determines whether program is already installed ---
;-- Call from C : bool TsrIsInst( byte i2f_fctnr);
;-- 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 ?                ;Funct. no. for INT 2FH
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 ;Funct. no. for INT 2FH
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                   ;Return to caller

_TsrIsInst endp                ;End procedure

;-----
;-- TSRCANUNIST: Determines whether installed copy of TSR can be -----
;--                reinstalled or uninstalled.
;-- Call from C : bool TsrCanUnInst( void );
;-- 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

push di ;Store DI (if reg. variable)  
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  
pop di ;Pop DI from stack  
ret

tcu\_ok: mov ax,-1  
pop di ;Pop DI from stack  
ret

\_TsrCanUninst endp

```

;-----
;-- TSRUNINST: Reinstalls the TSR and releases the allocated memory. -
;-- Call from C : void TsrUnInst( void );
;-- Info      : TSRCANUNINST() 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 8H 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 9H 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 13H 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 28H 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 2FH 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

        sti                     ;Enable interrupts

        ;-- Release memory -----

        mov    es,es:c_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 memory
        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 C: void (*)(void) TsrSetPtr( void far * Fct );

_TsrSetPtr proc    near

```

```

sframe3      struc                      ;Access structure from stack
bp3          dw ?                      ;Gets BP
ret_adr3     dw ?                      ;Return address
fctptr3      dd ?                      ;Pointer to 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,word ptr frame.fctptr3 ;Move offset address to AX
        mov  call_ofs,ax              ;and store it

```

```

        mov  ax,offset _TsrCall;Return near pointer to TsrCall
        pop  bp                      ;Pop BP from stack

```

```

        ret                          ;Return to caller

```

```

_TsrSetPtr endp                      ;End of procedure

```

```

;-----
;-- TSRCALL: Calls a routine in the previously installed copy of -----
;--           the TSR program.
;-- Call from C: void TsrCall( void )
;-- Attention   : - The stack should not be altered in this
;--               routine.

```

```

_TsrCall  proc near

```

```

        ;-- Execute context change to C program and call the -----

```

```

;-- appropriate procedure

push di                ;Push DS,
push si                ;SI and
push ds                ;DI

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:c_psp        ;Get PSP segment address
int  21h               ;Call DOS interrupt

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

;-- Copy arguments to stack of installed TSR program -----

push ss                ;Set DS:SI to arguments on
pop  ds                ;the current stack
mov  si,sp
add  si,8

mov  cx,ARG_WORDS*2
mov  di,es:c_sp         ;Install ES:DI on the stack
sub  di,cx

```



```

mov  es,es:c_ss
rep movsb                ;Copy arguments

;-- Implement and call segment register in installed TSR ---

mov  es,call_seg         ;Move installed TSR segment register
                           ;to ES

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

mov  ss,es:c_ss           ;Activate stack for installed
mov  sp,es:c_sp           ;copy of program
sub  sp,ARG_WORDS*2       ;Subtract arguments
sti                        ;Enable interrupts

mov  ds,es:c_ds           ;Set segment register
mov  es,es:c_es           ;for C program

call [call_ptr]          ;Set function results given by
mov  ret_ax,ax             ;TSRSETPTR
mov  ret_dx,dx

cli                        ;Disable interrupts
mov  ss,uprg_ss           ;Move data from stack
mov  sp,uprg_sp           ;Toggle
sti                        ;Enable interrupts

;-- Context change returning to current program -----

mov  ah,lah               ;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
        pop     ds                ;and 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,ret_ax         ;Return function result
        mov     dx,ret_dx

        pop     si                ;Pop registers
        pop     di
        ret                     ;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
           iredt                    ;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 [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 --> Return 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]     ;Return 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 C program -----

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

        mov ss,c_ss            ;Activate C program's stack
        mov sp,c_sp
        sti                    ;Enable interrupts

        push ax                ;Store processor registers
        push bx                ;for the C 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,u_prg_ss         ;DS:SI indicates end of DOS stack
        mov     si,u_prg_sp

tsrs1:   push    word ptr [si]       ;Push word from DOS stack to C 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,c_psp            ;Get PSP segment addr. of C program
        int     21h                ;Call DOS interrupt

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

        mov     ds,c_ds             ;Set segment register for
        mov     es,c_es             ;C program

        call    [fct_adr]           ;Call start function

```

```

;-- Execute context change to interrupted program -----

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

;-- Restore DOS stack -----

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

pop  es              ;Pop stored registers from
pop  ds              ;C 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

;-----

_text      ends            ;End of code segment
end        end              ;End of program

```