

```

;*****;
;*          P R O C C A . A S M          *;
;*-----*
;* Task      : Makes two functions available for linking to *;
;*            C programs. These functions help determine *;
;*            the processor type and the type of coprocess- *;
;*            or. *;
;*-----*
;* Author    : MICHAEL TISCHER *;
;* Developed on : 08/15/88 *;
;* Last update : 10/17/91 *;
;*-----*
;* Assembly   : MASM PROCCA; or TASM PROCCA *;
;*            ... then link with a C program *;
;*****;

IGROUP group _text          ;Program segment
DGROUP group _bss, _data    ;Data segment
        assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS    segment word public 'BSS' ;This segment handles all uninitial-
_BSS    ends                      ;ized static variables

_DATA   segment word public 'DATA' ;This segment handles all initialized
        ;global and static variables
_DATA   ends
;== Constants =====

p_80486 equ 8                  ;Codes for different processor types
p_80386 equ 7
p_80286 equ 6
p_80186 equ 5
p_80188 equ 4
p_v30   equ 3
p_v20   equ 2
p_8086  equ 1
p_8088  equ 0

co_80387 equ 3                ;Codes for the coprocessors
co_80287 equ 2
co_8087  equ 1
co_none  equ 0

NOP_CODE equ 90h              ;Code for machine language command NOP
DEC_DX_C equ 4Ah              ;Code of DEC DX

;== Global variables =====

_DATA   segment word public 'DATA'

cpz      dw 0                  ;For coprocessor test

_DATA   ends

;== Program =====

_TEXT   segment byte public 'CODE' ;Program segment

public   _getproc              ;Function made available for other
public   _getco                ;programs

;-- GETPROC: Determines the type of processor a PC is equipped with ---
;-- Call from C : int getproc( void );
;-- Output      : The processor type number (see constants above)

_getproc proc near

        pushf                  ;Secure flag register contents
        push di

        ;== Determine whether model came before or after 80286 =====

        xor ax,ax              ;Set AX to 0
        push ax                ;and push onto stack
        popf                   ;Pop flag register off of stack
        pushf                  ;Push back onto stack

```

```

pop     ax                ;and pop off of AX
and     ax,0f000h         ;Do not clear the upper four bits
cmp     ax,0f000h         ;Are bits 12 - 15 all equal to 1?
je      not_286_386       ;YES --> Not 80386 or 80286

;-- Test for determining whether 80486, 80386 or 80286 -----

mov     dl,p_80286        ;In any case, it's one of the
mov     ax,07000h         ;three processors
push    ax                ;Push 07000h onto stack
popf    ;Pop flag register off
pushf   ;and push back onto the stack
pop     ax                ;Pop into AX register
and     ax,07000h         ;Mask everything except bits 12-14
je      pende             ;Are bits 12 - 14 all equal to 0?
                ;YES --> It's an 80286

inc     dl                ;No --> it's either an 80386 or an
                ;80486. First set to 386

;-- The following test to differentiate between 80386 and ---
;-- 80486 is based on an extension of the EFlag register on
;-- the 80486 in bit position 18.
;-- The 80386 doesn't have this flag, which is why you
;-- cannot use software to change its contents.

cli                      ;No interrupts now

db 066h,08Bh,0DCh        ;mov     ebx,esp      store current SP
db 066h,083h,0E4h,0FCh   ;and     esp,0FFFCh   round off DWORD
db 066h,09Ch             ;pushfd             Store flag register
db 066h,058h             ;pop      eax         from stack to AX
db 066h,08Bh,0C8h        ;mov     ecx,eax      and CX
db 066h,035h,000h,0h,4h,0h ;xor     eax,1 shl 18  XOR alignment bit
db 066h,050h             ;push     eax         and put in flag
db 066h,09Dh             ;popfd            register
db 066h,09Ch             ;pushfd            Push flag onto stack
db 066h,058h             ;pop      eax         and pop AX off
db 066h,051h             ;push     ecx        Return old flag
db 066h,09Dh             ;popfd            contents
db 066h,033h,0C1h        ;xor     eax,ecx      Test AL bit
db 066h,0C1h,0E8h,012h   ;shr     eax,18      Shift AL bit to bit 0
db 066h,083h,0E0h,001h   ;and     eax,1h      Mask all others
db 066h,08Bh,0E3h        ;mov     esp,ebx      Restore SP.

sti                      ;Allow interrupts again
add     dl,al             ;AL is 1, if 486
jmp     pende             ;Test is ended

;== Test for 80186 or 80188 =====

not_286_386 label near

mov     dl,p_80188        ;Load code for 80188
mov     al,0ffh           ;Set all bits in AL register to 1
mov     cl,021h           ;Move number of shift operations to CL
shr     al,cl             ;AL CL shift to the right
jne     t88_86            ;If AL <> 0, it must be either an
                ;80188 or 80186

;== Test for NEC V20 or V30 =====

mov     dl,p_v20          ;Load code for NEC V20
sti                      ;Enable interrupts
push    si                ;Mark contents of SI register
mov     si,0              ;Starting with first byte in ES, read
mov     cx,0ffffh         ;a complete segment
rep     lodsb byte ptr es:[si] ;REP with a segment override
                ;(works only with NEC V20, V30)
pop     si                ;Pop SI off of stack
or      cx,cx             ;Has entire segment been read?
je      t88_86            ;YES --> V20 or V30

mov     dl,p_8088         ;NO --> Must be 8088 or 8086

;== Test for 88/86 or V20/V30 =====

```

```

    ;-- Run test with help of queue (as above), however, this is
    ;-- a smaller queue

t88_86    label near

    push cs                ;Push CS onto stack
    pop  es                ;and pop ES off
    std                     ;Increment on string instructions
    mov  di,offset q2_end  ;Set DI at end of queue
    mov  al,0fbh           ;Instruction code for "STI"
    mov  cx,3              ;Execute string instruction 3 times
    cli                     ;Suppress interrupts
    rep  stosb             ;Overwrite INC DX instruction
    cld                     ;Increment on string instructions
    nop                     ;Fill queue with dummy instructions
    nop
    nop

    inc  dx                ;Increment processor code
    nop
q2_end:   sti              ;Re-enable interrupts

    ;-----

pende     label near      ;End testing

    pop  di                ;Pop DI off of stack
    popf                 ;Pop flag register off of stack
    xor  dh,dh            ;Set high byte of proc. code to 0
    mov  ax,dx            ;Proc. code = return value of funct.

    ret                  ;Return to caller

_getproc  endp            ;End of procedure

;-- GETCO: Determines the kind of coprocessor, if available -----
;-- Call from C : int getco( void );
;-- Output      : The number of the coprocessor type (see constants)

_getco    proc near

    mov  dx,co_none        ;First assume there is no CP

    mov  byte ptr cs:wait1,NOP_CODE  ;WAIT-instruction on 8087
    mov  byte ptr cs:wait2,NOP_CODE  ;Replace by NOP

wait1:    finit            ;Initialize Cop
    mov  byte ptr cpz+1,0  ;Move high byte control word to 0
wait2:    fstcw cpz         ;Store control word
    cmp  byte ptr cpz+1,3  ;High byte control word = 3?
    jne  gcende            ;No ---> No coprocessor

    ;-- Coprocessor exists. Test for 8087 -----

    inc  dx
    and  cpz,0FF7Fh        ;Mask interrupt enable mask flag
    fldcw cpz              ;Load in the control word
    fdisi                  ;Set IEM flag
    fstcw cpz              ;Store control word
    test cpz,80h           ;IEM flag set?
    jne  gcende            ;YES ---> 8087, end test

    ;-- Test for 80287/80387 -----

    inc  dx
    finit                  ;Initialize cop
    fldl                     ;Number 1 to cop stack
    fldz                     ;Number 0 to cop stack
    fdiv                     ;Divide 1 by 0, erg to ST
    fld  st                 ;Move ST onto stack
    fchs                    ;Reverse sign in ST
    fcompp                  ;Compare and pop ST and ST(1)
    fstsw cpz               ;Store result from status word
    mov  ah,byte ptr cpz+1 ;in memory and move AX register
    sahf                    ;to flag register

```

```

        je      gcende          ;Zero-Flag = 1 : 80287
        inc     dx              ;Not 80287, must be 80387 or inte-
                                ;grated coprocessor on 80486

gcende:  mov     ax,dx          ;Move function result to AX
        ret                          ;Return to caller

_getco   endp

;== End =====

_text    ends                  ;End of program segment
        end                ;End of assembler source

```