Chapter 15

Determining Processor Types

There are number of utility programs on the market today which can tell you about the configuration of a PC. This information can include the amount of available RAM, the running DOS version and the type of processor the PC has.

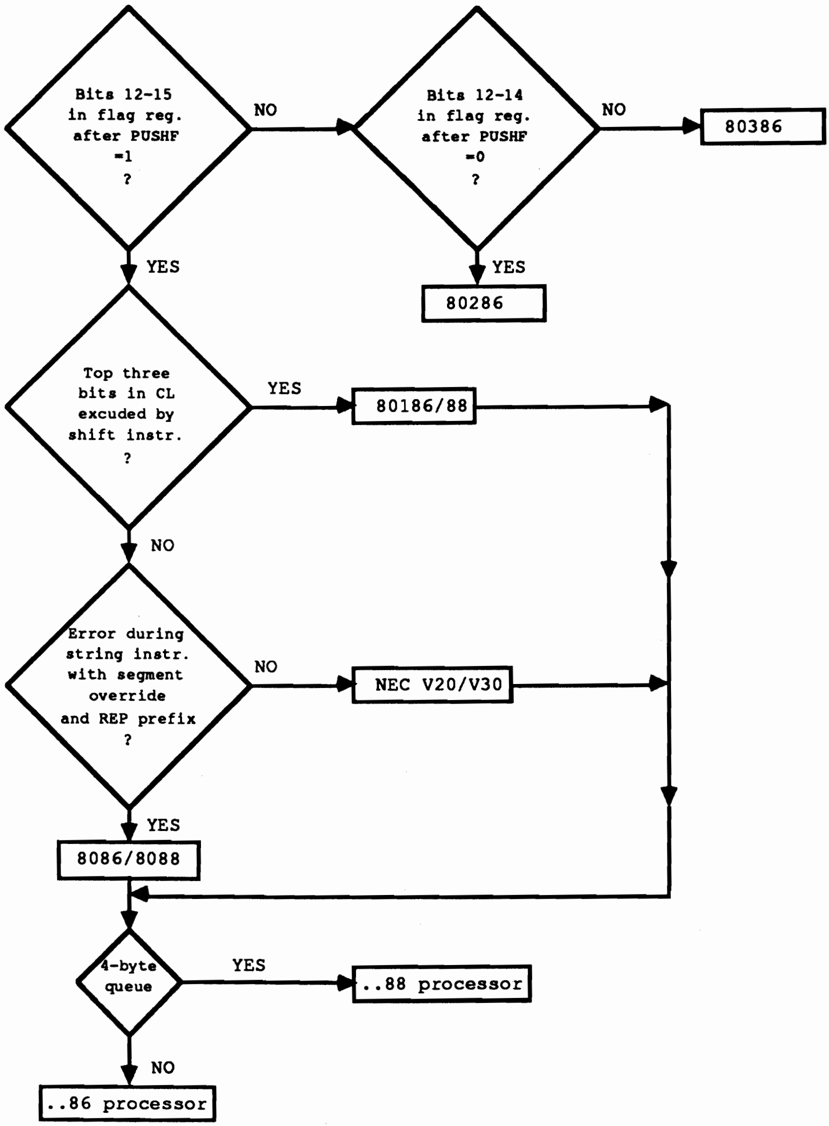
This information can be very useful for developing programs in high level languages, since code generation can be adapted to the particular processor. For example, both Microsoft C and Turbo C allow special code generation for the 8088, the 80286 and the 80386, which makes full use of the capabilities of the particular processor and instruction set. This can dramatically improve performance for programs which work with large groups of data. One way to take advantage of this would be to compile the program once for each of the three processor types. Then a program could be developed to serve as the boot for the actual program. This boot program would determine the type of processor being used and load the main program version most compatible with the processor.

Which processor is which?

This raises the question of how to determine which type of processor is being used, since unlike other configuration information, we cannot find this out by making a BIOS or DOS call. Unfortunately, there is no machine language instruction which instructs the processor to reveal its identity, so we have to use a trick. This trick relies on a condition which, according to a few hardware manufacturers, is totally impossible.

This is a test which involves the different ways the various processors execute certain machine language instructions. Although processors from the 8086 to the 80386 are upwardly software compatible, the development of this processor series brought small changes in the logic of certain instructions. Since these changes are only noticeable in rare situations, a program developed for the 8088 processor will also run correctly on all other processors in the Intel 80x86 series. But if we deliberately put a processor into such a situation, we can determine its identity from its behavior.

These differences are only noticeable at the assembly language level, so our test program must be written in assembly language. We have included listings at the end of this chapter which allow the test routine to be included in Pascal, C and BASIC programs as well.



Determining processor type on a PC

As the flowchart above shows, the routine consists of several tests which can distinguish various processor types from one another. The next test executes only when the current test returns a negative response.

Flag register test

The first test concerns the different layout of the flag register in the different processors. The meaning of bits 0 to 11 is the same in all processors, but bits 12-15 are also defined in processors from 80286 up (through the introduction of the protected mode). This can be noticed in the instructions PUSHF (push the contents of the flag register onto the stack) and POPF (fetch the contents of the flag register from the stack). On processors through the 80188 these instructions always set bits 12-15 of the flag register to 1, but this doesn't occur in the 80286 and 80386 processors. The first test in the routine takes advantage of this fact, in which it places the value 0 on the stack and then loads it into the flag register with the POPF instruction. Since there is no instruction for comparing the contents of bits 12 to 15, the flag register is pushed back onto the stack with a PUSHF instruction. This is so we can get the contents into the AX register with POP AX, where we can test bits 12 to 15.

If all four bits are set, then the processor cannot be an 80286 or an 80386, and the next test is performed. However, if not all four bits are set, then we have reduced the set of possible processors to the 80826 and the 80386. Since POPF also operates differently between these two processors, it is easy to tell them apart. We simply repeat the whole process, this time by placing the value 07000H on the stack instead of 0. When the flag register is loaded with the POPF instruction, bits 12 to 14 of the flag register will be set to 1. If these bits are no longer 1 when the contents of the flag register are fetched from the stack, then the processor must be an 80286, which, in contrast to the 80386, sets these three bits back to 0. The test is then concluded for these two processors.

Narrowing down the field

If the processor did not pass the first test, the following test will show if it is an 80188 or 80186. With the introduction of these two processors, the shift instructions (like SHL and SHR) were changed in the way they use the CL register as a shift counter. While in previous processors the number of shifts could be between 0 and 255, the upper three bits of the CL register are now cleared before the instructions starts, limiting the number of shift operations. This makes sense since a word will contain all zeros anyway after at most 16 shifts (17, if the carry flag is shifted). Additional shifts will cost valuable processor time and will not change the value of the argument at all.

The second test makes use of this behavior by shifting the value 0FFH in the AL register 21H positions to the right with the SHR instruction. If the processor executing the instruction is an 80188 or later type, the upper three bits of the shift counter will first be cleared, and only one shift is performed instead of 21H shifts.

```
  021H (00100001(b))  number of shifts
& 01fH (00011111(b))  mask out the upper three bits
----------------------------------------------------
  001H (00000001(b))  actual number of shifts
```

Unlike its predecessors, which would actually shift the value 0FFH to the right 021H times and return the value 0, the 80188 and 80186 will return the value 07FH. By checking the contents of the AL register after the shift we can easy tell if the processor is an 80188 or 80186 (AL not zero), or not (AL equal to 0). If the processor also fails this test, then we know it is an 8088/8086 or V20/30.

V20 and V30 processors

The V20 and V30 processors are 8088/8086 "clones" which use the same instruction set as their Intel cousins, but which operate considerably faster due to the optimization of internal logic and improved manufacturing. This speed also results in a higher cost, so some PC manufacturers avoid using these processors.

In addition to the faster execution of instructions, these processors also corrected a small error which occurs in the 8088 and 8086 processors. If a hardware interrupt is generated during the execution of a string instruction (such as LODS) in connection with the REP(eat) prefix and a segment override, the execution of this instruction will not resume after the interrupt has been processed. This can easily be determined because the CX register, which functions as the loop counter in this instruction, will not contain a 0 as expected after the instruction.

We make use of this behavior in the test program by loading the CX register with the value 0FFFFH, and then executing a string instruction 65535 times with the REP prefix and segment override. Since even a fast processor needs some time to do this, a hardware interrupt will be generated during one of the 65535 executions of this instruction. In the case of the 8088 or 8086, the instruction will not be resumed after the interrupt, and the remaining "loop passes" will not execute. The test program verifies this from the CX register after the instruction has been executed.

Data bus test

Once we have distinguished between the 8088/8086 and the V20/30, one last test is performed for all processors (except the 80286 and 80386). In this test we determine if the processor is using an 8-bit or a 16-bit data bus. This allows us to tell the difference between the 8088 and 8086, the V20 and V30, or the 80188 and the 80186. We cannot determine the width of the data bus with assembly language commands, but the data bus width is related to the length of the instruction queue within the processor.

Queue

The queue stores the instructions following the instruction currently being executed. Since these instructions are taken from the queue and not from memory, this improves execution speed. This queue is six bytes long on processors with a 16-bit data bus, but only four bytes long on processors with an 8-bit data bus.

The last test is based on this difference in length. The string instruction STOSB (store string byte) used in connection with the REP prefix modifies three bytes in the code segment immediately following the STOSB instruction. These bytes are placed so that they are found within the queue on a processor with a six-byte queue; the processor won't even notice the change. On a processor with a four-byte queue, these instructions are still outside the queue, so the modified versions of the instructions are loaded into the queue. The program makes use of this by modifying the instruction INC DX, which increments the contents of the DX register which contains the processor code in the routine. This instruction is executed only when the processor has a six-byte queue, and the instruction was already in the queue by the time the modification was performed.

On a processor with a four-byte queue, this instruction is replaced by the STI instruction, which doesn't affect the contents of the DX register (or the processor code). STI sets the interrupt bit in the processor flag register. Since this procedure always increments the processor code by one for 16-bit processors, the processor codes in the routine are chosen so that the code for the 16-bit version of a processor always follows the code for the 8-bit version of the same processor.

The following BASIC and Pascal programs use DATA or inline statements instead of assembly language. However, we included the assembly language versions of these statements here so that you can follow the program logic. The C implementation requires direct linking of C and the assembly language routine.

BASIC listing: PROCB.BAS

```
100 '*******************************************************************'
110 '*                          P R O C B                              *'
120 '*-----------------------------------------------------------------*'
130 '*  Task            : Examines the main processor and tells the    *'
140 '*                    user the processor type                      *'
150 '*  Author          : MICHAEL TISCHER                              *'
160 '*  Developed on    : 09/06/1988                                   *'
170 '*  Last update     : 05/23/1989                                   *'
180 '*******************************************************************'
190 '
200 CLS : KEY OFF
210 PRINT"ATTENTION: This program should only be run when GW-BASIC is loaded from"
220 PRINT"the DOS prompt using the command <GWBASIC /m:60000>."
230 PRINT : PRINT"If this isn't the case, press the <s> key to stop."
240 PRINT"Otherwise, press any other key to continue...       ";
250 A$ = INKEY$ : IF A$ = "s" THEN END
260 IF A$ = "" THEN 250
270 CLS                                                    'Clear screen
280 GOSUB 60000                               'Install assembler routine
290 CALL PT(PTYP%)                            'Determine processor type
300 RESTORE 1000                'Read DATA statements starting at line 1000
310 FOR I% = 0 TO PTYP% : READ P$ : NEXT           'Get processor name
320 PRINT "PROCB  -  (c) 1988 by MICHAEL TISCHER"
330 PRINT "Your PC contains a(n) ";P$;" processor."
340 END
350 '
1000 DATA "INTEL 8088", "INTEL 8086", "NEC V20", "NEC V30"
1010 DATA "INTEL 80186", "INTEL 80188", "INTEL 80286", "INTEL 80386"
```

```
1020 '
60000 '**************************************************************'
60010 '* Routine for determining onboard processor type             *'
60020 '*------------------------------------------------------------*'
60030 '* Input  : none                                              *'
60040 '* Output : PT is the starting address of the assembler routine *'
60050 '* Call to the routine:CALL PT(PTYP%)                         *'
60060 '**************************************************************'
60070 '
60080 PT=60000!              'Starting address of BASIC segment routine
60090 DEF SEG                                    'Define BASIC segment
60100 RESTORE 60140
60110 FOR I% = 0 TO 105 : READ X% : POKE PT+I%,X% : NEXT  'POKE routine
60120 RETURN                                         'Return to caller
60130 '
60140 DATA  85,139,236,156,  6, 51,192, 80,157,156, 88, 37,  0,240, 61
60150 DATA   0,240,116, 19,178,  6,184,  0,112, 80,157,156, 88, 37,  0
60160 DATA 112,116, 54,254,194,235, 50,144,178,  4,176,255,177, 33,210
60170 DATA 232,117, 18,178,  2,251,190,  0,  0,185,255,255,243, 38,172
60180 DATA  11,201,116,  2,178,  0, 14,  7,253,176,251,185,  3,  0,232
60190 DATA  23,  0,250,243,170,252,144,144,144, 66,144,251, 50,246,139
60200 DATA 126,  6,137, 21,  7,157, 93,202,  2,  0, 95,131,199,  9,235
60210 DATA 227
```

Assembler listing: PROCBA.ASM

```
;***********************************************************************;
;*                              P R O C B A                             *;
;*---------------------------------------------------------------------*;
;*    Task:          : Determines the type of processor installed in   *;
;*                     a PC                                             *;
;*                     This BASIC version of the program converts       *;
;*                     DATA statements into machine language, and       *;
;*                     executes this code in the BASIC program          *;
;*---------------------------------------------------------------------*;
;*    Author         : MICHAEL TISCHER                                  *;
;*    Developed on   : 09/05/1988                                       *;
;*    Last update    : 05/24/1989                                       *;
;*---------------------------------------------------------------------*;
;*    assembly       : MASM PROCBA;                                     *;
;*                     LINK PROCBA;                                     *;
;*                     EXE2BIN PROCBA PROCBA.BIN                        *;
;*                         convert to DATA statements and add to        *;
;*                         a BASIC program                              *;
;***********************************************************************;

;== Constants ===========================================================

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

;== Code ================================================================

code      segment para 'CODE'     ;Definition of CODE segment

          org  100h

          assume cs:code, ds:code, ss:code, es:code

getproc   proc far                ;GW-BASIC waits for CALL FAR procedure


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

```
          pushf                   ;Save contents of flag registers
          push es                 ;Mark ES

          ;-- test for 80386/80286 - ---------------------------------

          xor  ax,ax              ;Set AX to 0 and
          push ax                 ;push onto stack
          popf                    ;Get as flag register from stack
          pushf                   ;Put on stack again and
          pop  ax                 ;return to AX
          and  ax,0f000h          ;Don't clear the top 4 bits
          cmp  ax,0f000h          ;Are bits 12-15 all equal to 1?
          je   not_a_386          ;YES-> Not an 80386 or 80286

          ;-- Test to see if it should be handled as 80386 or 80286 ----

          mov  dl,p_80286         ;This narrows it down to one of the
          mov  ax,07000h          ;two processors
          push ax                 ;Push value 07000H onto the stack
          popf                    ;Return as flag register
          pushf                   ;and push back onto stack
          pop  ax                 ;Pop off and return to AX register
          and  ax,07000h          ;Do not mask bits 12-14
          je   pende              ;Are bits 12-14 equal to 0?
                                  ;YES-> Treat it as an 80286

          inc  dl                 ;NO-> Treat it as an 80386
          jmp  pend               ;Test ended

          ;-- Test for 80186 or 80188 --------------------------------

not_a_386 label near

          mov  dl,p_80188         ;Load code for 80188
          mov  al,0ffh            ;Set all bits in AL register to
          mov  cl,021h            ;Number of shift operations after CL
          shr  al,cl              ;Shift AL CL times to the right
          jne  t88_86             ;If AL<>0 then it must be handled as
                                  ;80188 or 80186

          ;-- Test for NEC V20 or V30 --- ----------------------------

          mov  dl,p_v20           ;Load code for NEC V20
          sti                     ;Interrupts should be enabled starting
          mov  si,0               ;with the first byte in ES
          mov  cx,0ffffh          ;Read a complete segment
          rep  lods byte ptr es:[si]      ;REP with segment override
                                  ;works only with NEC V20/V30 chips
          or   cx,cx              ;Has the complete segment been read?
          je   t88_86             ;YES--> it's a V20 or V30

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

          ;-- Test for ...88 or ...86 / V20 or V30 -------------------

t88_86    label near

          push cs                 ;Push CS onto the stack
          pop  es                 ;and pop off to ES
          std                     ;Using string inst. count backwards
          mov  al,0fbh            ;Code for "STI"
          mov  cx,3               ;Execute string instruction 3 times
          call get_di             ;Call starting address DI
t86_1:    cli                     ;Suppress interrupts
          rep  stosb
          cld                     ;Using string inst. ocunt backwards
          nop                     ;Fill queue with dummy command
          nop
          nop
```

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

         ;-----------------------------------------------------------

pend     label near               ;End processor test

         xor  dh,dh               ;Set high byte or processor code to 0
         mov  di,[bp+6]           ;Get addr. of processor code variables
         mov  [di],dx             ;Place processor code in this variable
         pop  es                  ;Pop off stack and place in ES
         popf                     ;Pop flag register off of stack and
         pop  bp                  ;Return BP
         ret  2                   ;FAR return takes us back to GW-BASIC
                                  ;Remove parameters from stack

getproc  endp                     ;End of PROG procedure

;-- GET_DI Check with DI for 88/86 Test ------------------------------

get_di   proc near

         pop  di                  ;Pop return address off of stack
         add  di,9                ;Remove starting 9 bytes from it
         jmp  t86_1               ;Return to the test routine

get_di   endp

;== End ==============================================================

code     ends                     ;End of CODE segment
         end  getproc
```

Pascal listing: PROCP.PAS

```
{**********************************************************************}
{*                              P R O C P                             *}
{*--------------------------------------------------------------------*}
{*    Task            : Examines the processor type in the PC and     *}
{*                      tells the user the processor type             *}
{*--------------------------------------------------------------------*}
{*    Author          : MICHAEL TISCHER                               *}
{*    Developed on    : 08/16/1988                                    *}
{*    Last update     : 05/23/1989                                    *}
{**********************************************************************}

program PROCP;

type  ProNames = array[0..7] of string[11]; { Array of processor names }

const ProcName : ProNames = ( 'INTEL 8088',                 { Code 0 }
                              'INTEL 8086',                 { Code 1 }
                              'NEC V20',                    { Code 2 }
                              'NEC V30',                    { Code 3 }
                              'INTEL 80188',                { Code 4 }
                              'INTEL 80186',                { Code 5 }
                              'INTEL 80286',                { Code 6 }
                              'INTEL 80386' );              { Code 7 }

{**********************************************************************}
{* GETPROC: Determines processor type in PC                           *}
{* Input   : none                                                     *}
{* Output  : Processor code (see CONST)                               *}
{* Info    : This function can be used in a program when added as     *}
{*           a UNIT                                                   *}
{**********************************************************************}

function getproc : byte;

begin           { Machine code routine for determining processor type }
```

```
 inline(
        $9C/$51/$52/$57/$56/$06/$33/$C0/$50/$9D/$9C/$58/$25/$00/
        $F0/$3D/$00/$F0/$74/$13/$B2/$06/$B8/$00/$70/$50/$9D/$9C/
        $58/$25/$00/$70/$74/$36/$FE/$C2/$EB/$32/$90/$B2/$04/$B0/
        $FF/$B1/$21/$D2/$E8/$75/$12/$B2/$02/$FB/$BE/$00/$00/$B9/
        $FF/$FF/$F3/$26/$AC/$0B/$C9/$74/$02/$B2/$00/$0E/$07/$FD/
        $B0/$FB/$B9/$03/$00/$E8/$16/$00/$FA/$F3/$AA/$FC/$90/$90/
        $90/$42/$90/$FB/$88/$56/$FF/$07/$5E/$5F/$5A/$59/$9D/$EB/
        $07/$90/$5F/$83/$C7/$09/$EB/$E4
       );
end;

{**************************************************************************}
{**                              MAIN PROGRAM                            **}
{**************************************************************************}

begin
  writeln('PROCP  -  (c) 1988 by MICHAEL TISCHER');
  writeln(#13#10, 'Your PC contains a(n) ', ProcName[getproc],
             ' processor.');
  writeln(#13#10);
end.
```

Assembler listing: PROCPA.ASM

```
;*************************************************************************;
;*                              P R O C P A                              *;
;*-----------------------------------------------------------------------*;
;*    Task           : Determines the type of processor installed in    *;
;*                     a PC.                                             *;
;*                     This version is converted by INLINE statements  *;
;*                     and then used by a Pascal program.                *;
;*-----------------------------------------------------------------------*;
;*    Author         : MICHAEL TISCHER                                   *;
;*    Developed on   : 08/22/1988                                        *;
;*    Last update    : 05/24/1989                                        *;
;*-----------------------------------------------------------------------*;
;*    assembly       : MASM PROCPA;                                      *;
;*                     LINK PROCPA;                                      *;
;*                     EXE2BIN PROCPA PROCPA.BIN                         *;
;*                     ... convert to INLINE statements and add to      *;
;*                         Pascal programs                               *;
;*************************************************************************;

;== Constants ============================================================

p_80386  equu  7                ;Codes for different types of
p_80286  equ   6                ;processors
p_80186  equ   5
p_80188  equ   4
p_v30    equ   3
p_v20    equ   2
p_8086   equ   1
p_8088   equ   0

;== Code =================================================================

code     segment para 'CODE'    ;Definition of CODE segment

         org  100h

         assume cs:code, ds:code, ss:code, es:code

getproc  proc near              ;This program is the essential main
                                ;program

         pushf                  ;Get contents of flag registers
         push cx                ;Get contents of all altered registers
         push dx                ;and push them onto stack
         push di
```

```
          push si
          push es

          ;-- Test for 80386/80286 ------------------------------------

          xor  ax,ax              ;Set AX to 0
          push ax                 ;and push onto stack
          popf                    ;Pop into flag register from stack
          pushf                   ;Return to stack
          pop  ax                 ;And pop back into AX
          and  ax,0f000h          ;Avoid clearing the to 4 bits
          cmp  ax,0f000h          ;Are bits 12-15 all equal to 1?
          je   not_a_386          ;YES->Not an 80386 or an 80286

          ;-- Test whether to handle it as an 80386 or 80286 -----------

          mov  dl,p_80286         ;This narrows it down to one of
          mov  ax,07000h          ;the two processors
          push ax                 ;Push value 7000H onto the stack
          popf                    ;Pop off as flag register
          pushf                   ;and push it back onto the stack
          pop  ax                 ;Pop off and return to AX register
          and  ax,07000h          ;Avoid masking bits 12-14
          je   pende              ;Are bits 12-14 all equal to 0?
                                  ;YES->Handle it as an 80286

          inc  dl                 ;NO->Handle it as an 80386
          jmp  pende              ;End of test

          ;-- Test for 80186 or 80188 ----------------------------------

not_a_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            ;Number of shift operations after CL
          shr  al,cl              ;Shift AL CL times to the right
          jne  t88_86             ;If AL is unequal to 0 it must be
                                  ;handled as an 80188 or 80186

          ;-- Test for NEC V20 or V30 ----------------------------------

          mov  dl,p_v20           ;Load code for NEC V20
          sti                     ;Interrupts should be enabled starting
          mov  si,0               ;with the first byte in ES
          mov  cx,0ffffh          ;Read a complete segment
          rep  lods byte ptr es:[si]      ;REP w/ segment override only
                                  ;works with NEC V20 and V30 processors
          or   cx,cx              ;Has complete segment been read?
          je   t88_86             ;YES-> V20 or V30

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

          ;-- Test for 8088 or 8086/V20 or V30 -------------------------

t88_86    label near

          push cs                 ;Push CS onto stack
          pop  es                 ;Pop off to ES
          std                     ;Using string inst. count backwards
          mov  al,0fbh            ;Instruction code for "STI"
          mov  cx,3               ;Execute string instruction 3 times
          call get_di             ;Get starting address of DI
t86_1:    cli                     ;Suppress interrupts
          rep  stosb
          cld                     ;Using string inst. count backwards
          nop                     ;Fill queue with dummy instruction
          nop
          nop
```

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

        ;------------------------------------------------------------

pende   label near            ;End testing

        mov  [bp-1],dl        ;Place processor code in return var.
        pop  es               ;Pop saved registers from
        pop  si               ;stack
        pop  di
        pop  dx
        pop  cx
        popf                  ;Pop flag register from stack and
        jmp  endit            ;Return to calling program

getproc endp                  ;End of PROG procedure

;-- GET_DI examines DI for 88/86 test ---------------------------------

get_di  proc near

        pop  di               ;Pop return address off of stack
        add  di,9             ;Take first 9 bytes from there
        jmp  t86_1            ;Return to the testing routine

endit   label near

get_di  endp

;== End ================================================================

code    ends                  ;End of CODE segment
        end  getproc
```

C listing: PROCC.C

```
/*********************************************************************/
/*                          P R O C C                                */
/*-------------------------------------------------------------------*/
/*    Task            : Determines the processor type in a PC        */
/*-------------------------------------------------------------------*/
/*    Author          : MICHAEL TISCHER                              */
/*    Developed on    : 08/14/1988                                   */
/*    Last update     : 06/22/1989                                   */
/*-------------------------------------------------------------------*/
/*    (MICROSOFT C)                                                  */
/*    Creation        : CL /AS /c PROCC.C                            */
/*                      LINK PROCC PROCCA                            */
/*    Call            : PROCC                                        */
/*-------------------------------------------------------------------*/
/*    (BORLAND TURBO C)                                              */
/*    Creation        : Create a project file containing these lines: */
/*                      PROCC                                        */
/*                      PROCCA.OBJ                                   */
/*********************************************************************/

extern int getproc()      ;         /* Includes the assembler routine */

/*********************************************************************/
/**                         main  program                          **/
/*********************************************************************/

void main()

{
 static char * procname[] = {    /* Vector w/ pointers to proc. names */
                            "Intel 8088",               /* Code 0 */
                            "Intel 8086",               /* Code 1 */
                            "NEC V20",                  /* Code 2 */
```

```
                                        "NEC V30",              /* Code 3 */
                                        "Intel 80188",          /* Code 4 */
                                        "Intel 80186",          /* Code 5 */
                                        "Intel 80286",          /* Code 6 */
                                        "Intel 80386"           /* Code 7 */
                                       };

 printf("\nPROCC (c) 1988 by Michael Tischer\n\n");
 printf("This PC contains a(n) %s processor\n",
        procname[ getproc() ] );
}
```

Assembler listing: PROCCA.ASM

```
;**********************************************************************;
;*                            P R O C C A                             *;
;*--------------------------------------------------------------------*;
;*    Task            : Make a function available to a C program which *;
;*                      examines the type of processor installed in a  *;
;*                      PC and informs the calling program of this     *;
;*                      information.                                   *;
;*--------------------------------------------------------------------*;
;*    Author          : MICHAEL TISCHER                                *;
;*    Developed on    : 08/15/1988                                     *;
;*    Last update     : 05/24/1989                                     *;
;*--------------------------------------------------------------------*;
;*    assembly        : MASM PROCCA;                                   *;
;*                      ... link to a C program                        *;
;**********************************************************************;

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

CONST  segment word public 'CONST';This segment includes all read-only
CONST  ends                          ;constants

_BSS   segment word public 'BSS'  ;This segment includes al un-initial-
_BSS   ends                          ;ized static variables

_DATA  segment word public 'DATA' ;This segment includes all initialized
                                     ;gobal and static variables

_DATA  ends

;== Constants ==========================================================

p_80386    equ   7                   ;Codes for different processor tpyes
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

;== Program ============================================================

_TEXT  segment byte public 'CODE' ;Program segment

public    _getproc                   ;Function made available for other
                                     ;programs

;-- GETPROC: Determines the type of processor in the current PC --------
;-- Call from C : int getproc( void );
;-- Output      : The processor type's number (see constants above)

_getproc  proc near

          pushf                      ;Secure flag register contents
```

```
        ;-- Test for 80386/80286 ------------------------------------

        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 4 bits
        cmp  ax,0f000h          ;Are bits 12-15 al equal to 1?
        je   not_a_386          ;YES --> Not an 80386 or 80286

        ;-- Test for handling as an 80386 or 80286 ------------------

        mov  dl,p_80286         ;In any case, this routine checks for
        mov  ax,07000h          ;one of the two 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          ;Bits 12-14 not included
        je   pende              ;Are bits 12-14 all equal to 0?
                                ;YES--> Handle it as an 80286

        inc  dl                 ;NO --> Handle it as an 80386
        jmp  pende              ;End test

        ;-- Test for 80186 or 80188 ---------------------------------

not_a_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, handle is as 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  lods byte ptr es:[si]  ;REP with a segment override
                                    ;(works ony 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 -------------------------------

t88_86  label near

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

        inc  dx                 ;Increment processor code
```

```
         nop
q_end:   sti                      ;Re-enable interrupts

         ;------------------------------------------------------------

pende    label near               ;End testing

         popf                     ;Pop flag register off of stack
         xor  dh,dh               ;Set high byte of proc. code to 0
         mov  ax,dx               ;Processor code=return value of funct.
         ret                      ;Back to caller

_getproc endp                     ;End of procedure

;== End ===============================================================

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