Chapter I4
Turbo C and the IBM-PC

Revised: Feb. 23, 2011

I4.1 INTRODUCTION

The previous chapter has provided a detailed discussion of elements of the IBM-PC and Intel 8086 architecture that are relevant to MPX-PC. An operating system needs to be able to directly access hardware elements such as registers and interrupt vectors for various purposes. However, most programming languages are intentionally designed to be hardware independent, and to separate the programmer from the low-level architecture. Even C, which is more "low-level" than most other programming languages, does not provide (in the ANSI standard version) direct access to registers or specific memory locations.

This is one big reason why most early operating systems were written in assembly language. Today's OSs are written primarily in higher level languages, but will generally also contain a good bit of assembly code for the lowest-level routines.

It would seem, then, that to complete an MPX-PC project we must also use some assembly code. This is where Turbo-C comes in. Turbo-C was designed to aid in PC systems programming by providing a number of (non-standard!) extensions to enable the necessary hardware access *without resorting to assembly language at all!* So if you are using Turbo-C, you can use the extensions described in this chapter in addition to normal C code. If you prefer to use another C compiler, you will probably need an assembler as well.

The extensions described below take the form of added library functions, special names, and special keywords.

I4.2 POINTER CLASSES

As discusses in Chapter I3, most memory references in the 8086 architecture are specified by a 16-bit address, relying on the appropriate segment register to build the complete 20-bit address. However, there are times when both data references and instruction references must specify a complete address, including the segment part.

To handle this need in a consistent way, Turbo C distinguishes between *near addresses* and *far addresses*. A near address is a 16-bit value representing an offset only. A far address is a

32-bit value; the high-order 16 bits represent the segment part of a complete address, while the low-order 16 bits represent the offset part.

Because the segmented form of an address is not unique, difficulties may arise in comparing addresses. Two far addresses may be found unequal even though they resolve to the same 20-bit address. For this reason, among others, Turbo C supports a third address format called *huge addresses*. A huge address is a *normalized* far address; it has the highest possible segment value and the lowest possible offset value. We will not discuss huge addresses further, since they will not be needed in the MPX project.

Using this distinction among address representations, Turbo C classifies *pointers* and *functions* as either far or near. A far pointer is a 32-bit value representing a far address; a near pointer is a 16-bit value representing an offset only.

If a function is defined with the far attribute, then every call to that function will be an intersegment call which saves both the CS and the IP on the stack, and the return from the function will restore both these values. Otherwise, intrasegment calls (and returns) will be used which save (and restore) the IP only.

It is possible to explicitly declare the distance attribute of any pointer variable or function using the Turbo C keywords `far` and `near`. For example:

```
int near *np;
char far *fp;
void far func1(char c, int i1);
```

These keywords are recognized only when the "Turbo C keywords" option is selected. Most of the time, however, these keywords are unnecessary. The distance attribute is selected implicitly based on the memory model, as explained below.

Occasionally it is necessary to separate a far pointer into its segment and offset parts, or to construct a far pointer from these parts. For this purpose, Turbo C defines three special "functions" which are actually implemented as in-line macros. These functions are defined in the header file `dos.h`, which must be `#included` by any source file which uses them. The functions `FP_SEG` and `FP_OFF` obtain the segment part and the offset part, respectively, from a far pointer. They have the following prototypes:

```
unsigned int FP_SEG(far pointer);
unsigned int FP_OFF(far pointer);
```

The argument supplied to these functions can be any pointer type with the far attribute. Note that the standard type returned is unsigned int. If the offset part is to be used later as a near address, it must be cast to an appropriate pointer type.

The function `MK_FP` constructs a far pointer from segment and offset parts. Its prototype is:

```
void far *MK_FP(unsigned int seg, unsigned int off);
```

I4.3 MEMORY MODELS

The PC architecture allows the use of compact, consistent machine instructions for data references and branching, as long as it is assumed that the segment registers will not change. This is possible as long as each of the three major program segments, instructions, data, and stack, can be limited to 64K bytes each. For some programs this restriction is perfectly acceptable; for others it would be an intolerable limitation.

When segment registers may change, a more complicated memory reference strategy must be followed by the compiler. There can be a number of different situations, each leading to a different strategy. In the simplest case, all three major segment registers (CS, DS, and SS) may be both fixed and identical. Some programs may require multiple segments for instructions but only one for data, while still others require the opposite.

To meet these varying needs, Turbo C defines six different memory models, each based on different assumptions about the use of the segment registers. These models are called *tiny*, *small*, *medium*, *compact*, *large*, and *huge*. Details of all these models are provided in the Turbo C documentation. For the MPX project we make use of the *large* model, and this is the only one we will discuss here in more detail.

The large model assigns the *far* attribute by default to all pointers and all functions. Thus it supports multiple segments for both code and data. The only restriction enforced by this model, which distinguishes it from the *huge* model, is that all *static* data must be limited to a single 64K segment.

In MPX, it is not likely that either instructions or static data for the project itself will approach 64K in size. However, MPX must dynamically allocate memory, and begining in Module R4 this memory will be used to load and execute other programs to operate as processes. The memory requirements of these other programs may collectively exceed 64K, and the heap used for allocation is outside the initial data segment. Moreover, since programs loaded into this memory will eventually be executed, the CS must be set to a new value for each program. For these reasons, the large model is most appropriate.

The large model must be selected as a Turbo C compiler option, along with the option that tells the compiler not to assume that the DS and SS are equal. This model implies that all pointers are far pointers and all functions are far functions. If you require a near pointer or function you must specify the attribute explicitly. Further, when addresses on the stack are manipulated explicitly in MPX, you must remember that each address has both a segment part and an offset part, requiring a total of four bytes of storage.

I4.4 REGISTER PSEUDOVARIABLES

Because of the close control which an operating system must maintain over the hardware, there will be times in MPX when you need to directly access some of the 8086 registers from Turbo C. Turbo C allows you to access all the data registers, pointer/index registers, and segment registers using special names called *pseudovariables*. You cannot access the flag register

-

directly, but there are ways to determine its value after certain procedure calls. The names assigned to the accessible registers are:

```
_AX  _BX  _CX  _DX  _CS  _SP
_AL  _BL  _CL  _DL  _DS  _BP
_AH  _BH  _CH  _DH  _SS  _DI
                    _ES  _SI
```

These names are made accessible by `#including` the header file `dos.h` and by specifying the compiler option to recognize Turbo C or C++ keywords.

The sixteen bit registers are type `unsigned int` and the eight bit registers ("H" and "L" registers) are type `unsigned char`. The above pseudovariables may be used as any integer or character variable. Suppose you wish to access the value of the BP register. Then the following statement would place the value of the BP register into the integer variable temp:

```
temp = _BP;
```

Obviously these pseudovariables must be used with care, since they access registers that may be used for various purposes by the code generated automatically by Turbo C. You will be provided with various hints and suggestions in the use of the register pseudovariables throughout this manual.

I4.5 INTERRUPT HANDLERS

An important task of an operating system is to provide routines to handle interrupts, as discused in the text. An interrupt handler is a type of subprogram, but it requires a special structure because of the way it is invoked. Turbo C allows programmers to specify that functions are intended as interrupt handlers using the keyword `interrupt`. These functions will automatically be given the special structure that interrupt handlers require. This facility is discussed in detail in Chapter R3.

-