data segment. For example, the option "zt100" causes all data objects larger than 100 bytes in size to be implicitly declared as `far` and grouped in other data segments.

The default data threshold value is 32767. Thus, by default, all objects greater than 32767 bytes in size are implicitly declared as `far` and will be placed in other data segments. If the "zt" option is specified without a size, the data threshold value is 256. The largest value that can be specified is 32767 (a larger value will result in 256 being selected).

If the "zt" option is used to compile any module in a program, then you must compile all the other modules in the program with the same option (and value).

Care must be exercised when declaring the size of objects in different modules. Consider the following declarations in two different C files. Suppose we define an array in one module as follows:

```
extern int Array[100] = { 0 };
```

and, suppose we reference the same array in another module as follows:

```
extern int Array[10];
```

Assuming that these modules were compiled with the option "zt100", we would have a problem. In the first module, the array would be placed in another segment since `Array[100]` is bigger than the data threshold. In the second module, the array would be placed in the default data segment since `Array[10]` is smaller than the data threshold. The extra code required to reference the object in another data segment would not be generated.

Note that this problem can also occur even when the "zt" option is not used (i.e., for objects greater than 32767 bytes in size). There are two solutions to this problem: (1) be consistent when declaring an object's size, or, (2) do not specify the size in data reference declarations.

zv

(C++ only) Enable virtual function removal optimization.

2.3.7 80x86 Floating Point

This group of options deals with control over the type of floating-point instructions that the compiler generates. There are two basic types — floating-point calls (FPC) or floating-point instructions (FPI). They are selectable through the use of one of the compiler options described below. You may wish to use the following list when deciding which option best suits your requirements. Here is a summary of advantages/disadvantages to both.

FPC

1. not IEEE floating-point
2. not tailorable to processor
3. uses coprocessor if present; simulates otherwise
4. 32-bit/64-bit accuracy
5. runs somewhat faster if coprocessor present
6. faster emulation (fewer bits of accuracy)
7. leaner "math" library
8. fatter application code (calls to library rather than in-line instructions)
9. application cannot trap floating-point exceptions
10. ideal for ROM applications

FPI, FPI87

1. IEEE floating-point
2. tailorable to processor (see fp2, fp3, fp5, fp6)
3. uses coprocessor if present; emulates IEEE otherwise
4. up to 80-bit accuracy
5. runs "full-tilt" if coprocessor present
6. slower emulation (more bits of accuracy)
7. fatter "math" library
8. leaner application code (in-line instructions)
9. application can trap floating-point exceptions
10. ideal for general-purpose applications

To see the difference in the type of code generated, consider the following small example.

Example:

```
#include <stdio.h>
#include <time.h>

void main()
{
    clock_t  cstart, cend;
    cstart = clock();
    /*  .
        .
        .
    */
    cend = clock();
    printf( "%4.2f seconds to calculate\n",
            ((float)cend - cstart) / CLOCKS_PER_SEC );
}
```

The following 32-bit code is generated by the Open Watcom C compiler (wcc386) using the "fpc" option.

```
main_   push    ebx
        push    edx
        call    clock_
        mov     edx,eax
        call    clock_
        call    __U4FS  ; unsigned 4 to floating single
        mov     ebx,eax
        mov     eax,edx
        call    __U4FS  ; unsigned 4 to floating single
        mov     edx,eax
        mov     eax,ebx
        call    __FSS   ; floating single subtract
        mov     edx,3c23d70aH
        call    __FSM   ; floating single multiply
        call    __FSFD  ; floating single to floating double
        push    edx
        push    eax
        push    offset L1
        call    printf_
        add     esp,0000000cH
        pop     edx
        pop     ebx
        ret
```

The following 32-bit code is generated by the Open Watcom C compiler (wcc386) using the "fpi" option.

```
main_   push    ebx
        push    edx
        sub     esp,00000010H
        call    clock_
        mov     edx,eax
        call    clock_
        xor     ebx,ebx
        mov     [esp],eax
        mov     +4H[esp],ebx
        mov     +8H[esp],edx
        mov     +0cH[esp],ebx
        fild    qword ptr [esp]     ; integer to double
        fild    qword ptr +8H[esp]  ; integer to double
        fsubp   st(1),st            ; subtract
        fmul    dword ptr L2        ; multiply
        sub     esp,00000008H
        fstp    qword ptr [esp]     ; store into memory
        push    offset L1
        call    printf_
        add     esp,0000000cH
        add     esp,00000010H
        pop     edx
        pop     ebx
        ret
```

fpc

All floating-point arithmetic is done with calls to a floating-point emulation library. If a numeric data processor is present in the system, it will be used by the library; otherwise floating-point operations are simulated in software. This option should be used for any of the following reasons:

1. Speed of floating-point emulation is favoured over code size.
2. An application containing floating-point operations is to be stored in ROM and an 80x87 will not be present in the system.

The macro `__SW_FPC` will be predefined if "fpc" is selected.

> ***Note:*** When any module in an application is compiled with the "fpc" option, then all modules must be compiled with the "fpc" option.
>
> Different math libraries are provided for applications which have been compiled with a particular floating-point option. See the section entitled "Open Watcom C/C++ Math Libraries" on page 105.

See the section entitled "The NO87 Environment Variable" on page 107 for information on testing the floating-point simulation code on personal computers equipped with a coprocessor.

fpi

(16-bit only) The compiler will generate in-line 80x87 numeric data processor instructions into the object code for floating-point operations. Depending on which library the code is linked against, these instructions will be left as is or they will be replaced by special interrupt instructions. In the latter case, floating-point will be emulated if an 80x87 is not present. This is the default floating-point option if none is specified.

(32-bit only) The compiler will generate in-line 387-compatible numeric data processor instructions into the object code for floating-point operations. When any module containing floating-point operations is compiled with the "fpi" option, coprocessor emulation software will be included in the application when it is linked.

For 32-bit Open Watcom Windows-extender applications or 32-bit applications run in Windows 3.1 DOS boxes, you must also include the `WEMU387.386` file in the `[386enh]` section of the `SYSTEM.INI` file.

Example:

```
device=C:\WATCOM\binw\wemu387.386
```

Note that the `WDEBUG.386` file which is installed by the Open Watcom Installation software contains the emulation support found in the `WEMU387.386` file.

Thus, a math coprocessor need not be present at run-time. This is the default floating-point option if none is specified. The macros `__FPI__` and`__SW_FPI` will be predefined if "fpi" is selected.

> ***Note:*** When any module in an application is compiled with a particular "floating-point" option, then all modules must be compiled with the same option.
>
> If you wish to have floating-point emulation software included in the application, you should select the "fpi" option. A math coprocessor need not be present at run-time.
>
> Different math libraries are provided for applications which have been compiled with a particular floating-point option. See the section entitled "Open Watcom C/C++ Math Libraries" on page 105.

See the section entitled "The NO87 Environment Variable" on page 107 for information on testing the math coprocessor emulation code on personal computers equipped with a coprocessor.

fpi87

(16-bit only) The compiler will generate in-line 80x87 numeric data processor instructions into the object code for floating-point operations. An 8087 or compatible math coprocessor must be present at run-time. If the "2" option is used in conjunction with this option, the compiler will generate 287 and upwards compatible instructions; otherwise, the compiler will generate 8087 compatible instructions.

(32-bit only) The compiler will generate in-line 387-compatible numeric data processor instructions into the object code for floating-point operations. When the "fpi87" option is used exclusively, coprocessor emulation software is not included in the application when it is linked. A 387 or compatible math coprocessor must be present at run-time.

The macros `__FPI__` and`__SW_FPI87` will be predefined if "fpi87" is selected. See Note with description of "fpi" option.

fp2

The compiler will generate in-line 80x87 numeric data processor instructions into the object code for floating-point operations. For Open Watcom compilers generating 16-bit code, this option is the default. For 32-bit applications, use this option if you wish to support those few 386 systems that are equipped with a 287 numeric data processor ("fp3" is the default for Open Watcom compilers generating 32-bit code). However, for 32-bit applications, the use of this option will reduce execution performance. Use this option in conjunction with the "fpi" or "fpi87" options. The macro `__SW_FP2` will be predefined if "fp2" is selected.

fp3

The compiler will generate in-line 387-compatible numeric data processor instructions into the object code for floating-point operations. For 16-bit applications, the use of this option will limit the range of systems

on which the application will run but there are execution performance improvements. For Open Watcom compilers generating 32-bit code, this option is the default. Use this option in conjunction with the "fpi" or "fpi87" options. The macro `__SW_FP3` will be predefined if "fp3" is selected.

fp5

The compiler will generate in-line 80x87 numeric data processor instructions into the object code for floating-point operations. The sequence of floating-point instructions will be optimized for greatest possible performance on the Intel Pentium processor. For 16-bit applications, the use of this option will limit the range of systems on which the application will run but there are execution performance improvements. Use this option in conjunction with the "fpi" or "fpi87" options. The macro `__SW_FP5` will be predefined if "fp5" is selected.

fp6

The compiler will generate in-line 80x87 numeric data processor instructions into the object code for floating-point operations. The sequence of floating-point instructions will be optimized for greatest possible performance on the Intel Pentium Pro processor. For 16-bit applications, the use of this option will limit the range of systems on which the application will run but there are execution performance improvements. Use this option in conjunction with the "fpi" or "fpi87" options. The macro `__SW_FP6` will be predefined if "fp6" is selected.

fpd

A subtle problem was detected in the FDIV instruction of Intel's original Pentium CPU. In certain rare cases, the result of a floating-point divide could have less precision than it should. Contact Intel directly for more information on the issue.

As a result, the run-time system startup code has been modified to test for a faulty Pentium. If the FDIV instruction is found to be flawed, the low order bit of the run-time system variable `__chipbug` will be set.

```
extern unsigned __near __chipbug;
```

If the FDIV instruction does not show the problem, the low order bit will be clear. If the Pentium FDIV flaw is a concern for your application, there are two approaches that you could take:

1. You may test the `__chipbug` variable in your code in all floating-point and memory models and take appropriate action (such as display a warning message or discontinue the application).

2. Alternately, you can use the "fpd" option when compiling your code. This option directs the compiler to generate additional code whenever an FDIV instruction is generated which tests the low order bit of `__chipbug` and, if on, calls the software workaround code in the math libraries. If the bit is off, an in-line FDIV instruction will be performed as before.

If you know that your application will never run on a defective Pentium CPU, or your analysis shows that the FDIV problem will not affect your results, you need not use the "fpd" option. The macro `__SW_FPD` will be predefined if "fpd" is selected.

2.3.8 Segments/Modules

This group of options deals with object file data structures that are generated by the compiler.