

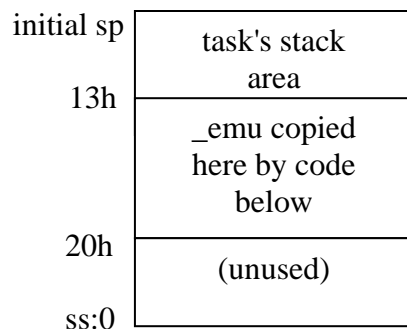
Making Borland and Paradigm C Floating Point Reentrant

x86 Real Mode

Borland and Paradigm C for real mode store floating point emulator data in the stack at ss:0. Since each task has its own stack, each will have its own emulator data area at the end of its stack. This makes the floating point emulation reentrant!

It is necessary to use a large data model and the 80x87 emulation package. Each task using floating point must have a bound far stack. This is accomplished by passing a non-zero value for the stack_size parameter in the call to create_taskf().

Here is what each stack looks like for BC 4.5 and later (for BC 3.1, _emu is at ss:0):



Your code for a sample task would look like this:

```
extern "C" byte far _emu; // in emuvars.asm

void _cdecl fptaskA_main(void)
{
    byte far * task_emu;
    STACK_CHECK(OFF); // necessary for smx v3.6 and higher, and BC 3.1
    task_emu = (byte far *)MK_FP(_SS, 0x20); // (_SS, 0) for BC 3.1
    memcpy(task_emu, &_emu, 0x110);

    unlockx();
    while (TRUE)
    {
        // your floating point and other code goes here
    }
}
```

Notice the 4 lines at the top of the task function, which have to be added to all tasks that use floating point. This copies the emulator data to the stack and disables smx stack checking for this

task. `STACK_CHECK(OFF)` must be used because the `_emu` structure is stored in the stack pad (if padding is enabled). Note that if the pad is small it will extend past the guard band into the stack. The need to turn off stack checking is unfortunate but unavoidable.

You must compile with `-f` (this is the default, so it does not appear in our makefiles). Also link `EMU.LIB` and `MATH_LIB`. **The order of files is important.** Here is what the end of your link line should look like:

```
emu math$(m) c$(m)
```

Note that `$(m)` signifies the memory model (e.g. "l"), so that `c$(m)` means `cl` (i.e. the Borland C run-time library, `cl.lib`).

Note: If a global floating point variable is shared between tasks, you must ensure that wherever it is set, the task is locked, or it must be protected by a semaphore, so that another task cannot preempt mid-move (due to an interrupt) and then reference the variable, which has been only partially set. (Remember that the emulator sets a variable with multiple moves, which can be interrupted.)

Summary of requirements:

1. Large or Compact memory model.
2. Bound far stack (pass non-zero stack size to `create_taskf()`)
3. `STACK_CHECK(OFF)` and other code above, at the top of each fp task.
4. About 300 bytes of extra stack space for each fp task.