# T E C H N O T E :
# Understanding Type 11 & No FPU Installed Errors on the Power Macintosh

By Brian Bechtel
**blob@apple.com**
Apple Developer Technical Support (DTS)

This Technote explains some of the causes of "Type 11" and "No FPU Installed"errors, along with debugging hints to help you find and fix them. It also focuses on what you as a programmer can do to prevent such errors.

Both Type 11 and No FPU Installed errors occur on Power Macintosh computers, depending on various factors. Both error messages are the result of programming errors. The No FPU Installed error usually means that your program is executing data rather than instructions. The Type 11 error is any error (such as a bus error) in native code.

**Note**
In this Technote, I will occasionally mention third-party products by name. If a third-party product is mentioned by name, it's meant to be a possible solution for you to investigate, not a recommendation. DTS doesn't evaluate third-party tools, nor maintain complete lists of possible third-party solutions. You should consult catalogs of development tools such as

- Macintosh Development Tools & Languages

- the Redgate Macintosh Registry
- the ubiquitous Internet
- the APDA catalog and
- reviews in technical magazines to find a product that fits your needs. ◆

There are many Web sites on the Internet — for example, Apple's Third-Party Development Products Database at <http://www.info.apple.com/dev/thirdparty/third_party.html> lists many Macintosh development tools. Perhaps you may find an appropriate tool to meet your development needs in this way.

# About "No FPU Installed" Errors

On a Power Macintosh computer, the error message No FPU Installed usually means your code has jumped to a non-code area and is executing garbage. Somewhere in that data (which is being interpreted as instructions) is an instruction whose op code begins with the hex value F. No FPU Installed is equivalent to a Type 10 Error — i.e., your program has attempted to execute an unknown instruction op code starting with the hex value F.

On a Power Macintosh, No FPU Installed means that some program has jumped to an area of data and has tried to execute any data word starting with the hex value F. The PowerPC chip has floating point support built in, but that floating-point support is different from the Motorola 680x0 family floating-point support. The Motorola 680x0 family uses an external floating-point unit. (There are also external memory management units (MMUs) and other specialized coprocessors.) Motorola 680x0 CPUs use instructions starting with hex value F as instructions for these coprocessors.

## 680x0 Microprocessor Instructions

All Motorola 680x0 microprocessors have instructions consisting of at least one word (the operation word); some instructions can have up to eleven words (see the *Motorola MC68020 32-bit Microprocessor User's Manual*, Third edition, page 3-1.) The operation word determines what kind of instruction is to be executed. For example, the instruction

```
MOVE.B D1, D2
```

translates to the hexadecimal value

```
1401
```

This instruction starts with the hex value 1. The instruction

```
FMOVE.X   FP3,-(A7)
```

translates to

```
F227 6980
```

This instruction starts with the hex value F.

## F-line Instructions

The Motorola 680x0 architecture was originally designed to support a floating-point coprocessor chip. This chip, the Floating Point Unit (FPU), communicates with the CPU via a special set of instructions called F-line instructions. These instructions always start with an operation word beginning with the hex value F.

A program can be compiled to take advantage of the hardware assistance the FPU provides, and thus yield faster floating-point calculations than would be available with SANE (the Standard Apple Numerics Package). Such programs would have instructions in the program which start with the hex value F. A program only using SANE would never have an instruction starting with the hex value F.

**Note**
Some Macintosh models, such as the Macintosh SE/30, the Macintosh IIci, and the Macintosh IIfx, shipped with an FPU coprocessor installed. Other 680x0-based Macintosh computers, such as the Macintosh IIsi and the Macintosh Color Classic, had an optional FPU coprocessor. For these machines, it was possible to purchase an optional card with an FPU coprocessor. ◆

## 68040 & 68040LC Microprocessors

With the introduction of the 68040 and 68040LC chips, things got a bit more complicated. The 68040 microprocessor has most of the 68882 FPU included on the chip. Not everything in the 68881/68882 FPU is in the 68040 chip, however — just the routines that Motorola determined were most frequently used. The rest of the FPU routines are automatically emulated by software. Because the FPU is "built-in," as it were, the 68040 chip handles instructions that start with an operation word beginning with the hex value F by itself. There is no way to add a coprocessor to a 68040 chip; the instructions are never brought out of the chip itself.

The 68040LC is a cost-reduced version of the 68040 chip. Savings came about by removing the FPU portion of the chip. Not only does a 68040LC chip have no FPU, there is no way to add one.

## The Power Macintosh

Power Macintosh computers emulate a Motorola 68040LC, i.e., a machine without FPU support. When you get a bomb with the message No FPU installed, it means some instruction has been executed with an operation word starting with the hex value F, and that your program is running on a machine without a FPU. This machine could be one of the following:

- a Macintosh with a 68020 or 68030 microprocessor and no FPU coprocessor
- a Macintosh with a 68040LC microprocessor (which can never have a FPU)
- a Macintosh with a PowerPC chip running 68K code in emulation (since the 68K emulator emulates a 68040LC microprocessor).

Power Macintosh computers contain very fast floating-point support as part of the CPU. This floating-point support is different from the floating-point support provided by the Motorola 680x0 microprocessors. *Inside Macintosh:PowerPC Numerics*, describes Power Macintosh floating-point support.

# Defining a Type 11 Error

A Type 11 error means an illegal interrupt vector on a 680x0 machine. On a Power Macintosh, a Type 11 error is any exception in native code not handled

by one of the installed exception handlers. On a Power Macintosh, a Type 11 error can be almost any error that occurred in native code. Type 11 errors may include

- an address error
- a bus error
- an illegal instruction error that occurred in native code.

The exception handlers installed for native code don't correctly handle the particular condition which was raised, and the error is returned back to the System Error manager via the Mixed Mode manager. The System Error manager maps all such exceptions to the system error Type 11 Error.

If you install Macsbug 6.5.2 or later, some Type 11 errors may be reported as a PowerPC unmapped memory exception. This is equivalent to a bus error, i.e., an error indicating your program is accessing memory that doesn't exist.

The Modern Memory Manager was designed to be less forgiving than the classic (68K) Memory Manager. Disposing of something twice, disposing of memory that was never allocated, and other memory handling problems will often generate a Type 11 error, while on a 68K machine the problem may go unnoticed.

# Programming Mistakes Causing Type 11 Errors

The following sections give you examples of programming mistakes that may cause Type 11 errors.

## Example #1: Writing Past the End of an Array

Writing past the end of an array can be a subtle and difficult-to-find bug. Example #1 shows you why.

```
void IAmGoingToCrash(void)
{
Str27   badArray;
BlockMoveData("\pThis string is too long for this array", badArray, 39);
}
```

**Note**
I've made the bug very obvious; it may not be quite so obvious, however, in your code. ◆

In this example, I've put a string of 39 characters into an array defined to hold 27 characters. This overwrites the stack, which contains such useful things as your return address. On a 68K Mac, this causes a bus error. On a PowerPC Mac in 68K emulation, this causes a bus error. On a Power Macintosh in native code, this causes a Type 11 error.

### Example #2: Using a Poorly Initialized Pointer or Handle

If your program tries to use an uninitialized or badly initialized pointer, it can generate a Type 11 error. Here is what happens in Example #2:

```
Ptr badPointer = (Ptr)-2;
*badPointer = 0;
```

**Note**
I've made the bug very obvious; it may not be quite so obvious, however, in your code. ◆

In this example, I've created a pointer to non-existent memory (-2 isn't a valid address) and then tried to access the memory to which the pointer refers. On a 68K Mac, this causes a bus error. On a PowerPC Mac in 68K emulation, this also causes a bus error. On a Power Macintosh in native code, this causes a Type 11 error.

## Other Situations Causing Type 11 Errors

The following sections document some of the known bugs in various products that may cause Type 11 errors.

### Color Picker vs Third-Party SCSI Drivers

One reproducible problem stems from a known bug in several third-party SCSI hard disk drivers. The bug causes the Color Picker to crash when trying to resolve a boot volume alias it makes at startup time. This problem exists in any

system when Color Picker 2.0 or 2.0.1 is installed and the boot volume is not a removable drive. The fundamental cause of the problem is that the SCSI driver incorrectly marks the boot drive as a removable drive during the boot process and changes it to be properly marked as a fixed device after startup. To find its code, the Color Picker resolves the boot volume alias it made during startup. Since the alias passed to the Alias Manager is for a removable drive, the Alias Manager can't resolve the alias (since the device is now marked as fixed.) The Color Picker design did not anticipate the boot volume not being found. This results in a Type 11 error. The fix is to upgrade your SCSI driver to a later version.

## Calling a Routine That May Move Memory at Interrupt Time

The Apple Media Kit, release 1.2, had a problem that could cause random errors. During a VBL task, AMK was calling SetCCursor. SetCCursor can possibly move memory. Moving memory during interrupt time is a very bad idea. The Memory Manager may be in an unstable state, such as compacting memory, when an interrupt routine gets called. This can cause a Type 11 error in native code.

A replacement cursor.c file has been provided in the Apple Media Tool/ Programming Environment Runtime folder, and we do have a replacement Runtime Maker:Codes:Program file for Apple Media Tool. If you are currently developing titles, replacing these files and rebuilding your projects will remove the bug.

## Not Enough Heap Space

There are some situations where having small amounts of memory available in the system heap may lead to a Type 11 error. As an example, if the shared library manager can't load a shared library, you may get a Type 11 error. This can happen when QuickTime tries to load a decompressor for certain kinds of images.

## Writing Past the End of an Array

In OpenTransport 1.0.5 or earlier, there was a bug in the TCP/IP control panel code that could corrupt memory if there were more than 256 zones and a MacIP server was found in a zone past the 256th zone. Under these conditions, the control panel code wrote past the end of one of its buffers. This bug would

manifest itself if the user opened the select zone dialog and the TCP/IP control panel was left open long enough for the NBP lookup calls for zones past the 256th zone to complete. This is corrected in OpenTransport 1.0.6 and later, but is given here as an example of a Type 11 error.

## Insufficient Stack Space

Heavy use of local variables or recursion may cause your program to run out of stack space. When this happens, you may have crashes that are difficult to track down. Increasing your program's stack is one solution. Whenever you are using a large number of local variables or recursive routines, you should increase your program's stack (using the procedure listed in *Inside Macintosh: Memory* on page 1-40).

As an extreme example, a faceless background application (documented in Technical Note PS 2) has only a 2K stack by default. A 68K application has 24K of stack on most modern machines (8K if no Color QuickDraw is installed, 32K if A/UX is installed.) You can use the low memory accessor function `LMGetDfltStack` to find your current stack size. Native QuickDraw has several changes in algorithms that have increased the size of some structures. PowerPC alignment issues may cause the size of data structures to increase. Check your compiler documentation for further details.

To help detect collisions between the stack and the heap, a "stack sniffer" VBL task is installed that compares the current ends of the stack and heap and generates a system error 28 in case of a collision. Unfortunately, the Thread Manager is forced to disable the stack sniffer whenever it is installed. (This is because threads can have a stack in places where the stack sniffer doesn't expect them; if the stack sniffer is enabled, it would generate a system error 28.) This means you don't have the same level of protection as you did under older system software versions without the Thread Manager installed.

## Too Much Recursion in QuickDraw

Native QuickDraw before version 2.4.1 used a recursive routine to handle regions. This recursive code could cause your stack to run into the heap when handling certain region shapes. Version 2.4.1 of QuickDraw changed to a non-recursive routine which eliminates this error. (You can detect the version of QuickDraw using the Gestalt selector `'qd '`which returns a version.) In general, recursion on PowerMacs is dangerous unless you have substantial stack space

available; PowerPC stack frames are very large, and putting many of them on the stack may quickly exhaust available stack.

### Interrupts Not Handled by Device Drivers

Some Type 11 errors may legitimately be illegal interrupt vector errors, rather than simply unhandled exceptions in native code. If a device driver for a NuBus or PCI card does not install an interrupt service routine, but the card raises an interrupt, you will receive a Type 11 error. This will happen whether you are running in 68K code or PowerPC code. The only solution for such a problem is to update the driver to correctly handle the interrupt issued by the card.

## Other Programming Errors

Common programming errors may cause Type 11 or No FPU Installed errors. These include the same kinds of errors that usually result in address errors on 680x0-based Macintosh computers. For example:

- Indexing through an array incorrectly so that your program goes outside the memory allocated for that array (as demonstrated above).
- Disposing of memory twice
- Disposing of memory that was never allocated
- Calling DisposeHandle on a resource handle (use DisposeResource instead, to ensure that the Resource Manager doesn't lose handles behind its back.)

## Some Techniques To Help Avoid Type 11 Errors

The following techniques may help resolve situations that might lead to Type 11 errors:

- Test your code carefully.
- Install the Debugging Modern Memory Manager, available on the Tool chest Developer CD. This version of the Modern Memory Manager has additional

checks installed which allow you to track down and eliminate many memory management bugs.

- Use extensions such as EvenBetterBusError (found on the Tool chest Developer CD) in conjunction with Macsbug 6.5.2 or later, to detect use of uninitialized pointers or handles.
- Use third-party testing tools such as QC™ by Onyx or MemoryMine™ by Adianta.

**IMPORTANT**

It goes without saying that good testing before you release a product will prevent expensive customer support calls later on. ◆

## User-Level Recommendations

Here are some user-level suggestions recommended by Apple for unexplained Type 11 or No FPU installed errors:

1. Upgrade your hard disk driver(s) to the latest version available. There are some known problems between SCSI Manager 4.3 and some SCSI disk drivers. Starting with System 7.5, the SCSI Manager 4.3 is installed on all Macintosh Quadras and Power Macintosh computers. There are some known programming problems in some older third-party disk drivers.
2. Upgrade to the latest System Software appropriate for your system. Several significant bug fixes in system updates should reduce the number of Type 11 and No FPU Installed errors you encounter.
3. Do a clean install of your System Software. Use the Extensions Manager control panel to determine if any additional control panels or extensions are conflicting with your system software.
4. Some Type 11 errors may result from corrupted PRAM. Unfortunately, most of extended PRAM is undocumented. (See Inside Macintosh:Operating System Utilites, chapter 7, Parameter RAM Utilities, for what details are documented.) You can restore your default PRAM values by holding down Command-Option-P-R at system startup time.
5. Zap your PRAM by holding down Command-Option-P-R at boot time, or by using a shareware utility such as TechTool.

6. Make sure you are not using composite RAM in a Power Macintosh. Memory specifications are in the developer hardware notes for each computer.
7. Some users claim that installing the shareware extension SoftwareFPU cuts down on No FPU Installed problems. This extension emulates the Motorola FPU (at a considerably slower speed), thus preventing bombs from software which incorrectly makes FPU calls. This may alleviate no FPU Installed errors, but it doesn't address the fundamental problem, namely that some software is executing unexpected data or making illegal calls to a non-existent FPU.
8. Upgrade any software you find that causes repeatable errors.

# Summary

There are no easy solutions for handling Type 11 Error or No FPU Installed errors. Only careful debugging and testing can reduce the number and frequency of these errors. Most problems stem from common programming errors.

## Further Reference


- *Inside Macintosh*, *PowerPC System Software*, Addison-Wesley
- *Inside Macintosh:PowerPC Numerics*, Addison-Wesley
- *Macsbug Reference and Debugging Guide*, Addison-Wesley
- *Macsbug Release Notes,* part of the distribution of Macsbug
- Motorola MC68020 32-bit Microprocessor User's Manual, Third edition (available from Motorola)
- PowerPC™ Microprocessor Family: The Programming Environment (available from IBM or Motorola)
- *Inside Macintosh:Memory,* Addison-Wesley

## Acknowledgements

Thanks to Kate Adams, Mitch Bayersdorfer, Robert Bowers, Deborah Grits, Jim Murphy, and Quinn.