New Technical Notes

Macintosh

Developer Support

_StripAddress: The Untold Story

Memory M.ME.StripAddress

Revised by: Paul Snively & Andrew Shebanow August 1990
Written by: Andrew Shebanow October 1988

Inside Macintosh, Volume V, The OS Utilities, incorrectly documents the `_StripAddress` trap; this Technical Note correctly documents the trap and gives guidelines for its use.
Changes since April 1990: Added a discussion of why the `_StripAddress` trap should be used under certain circumstances when patching traps.

Don't Believe Everything You Read

The `_StripAddress` trap is described in the OS Utilities chapter of *Inside Macintosh*, Volume V as:

> `FUNCTION StripAddress(theAddress: LONGINT): LONGINT;`
>
> *If the system is running in 24-bit mode, `_StripAddress` is identical in function to the global variable `Lo3Bytes`: it returns the value of the low-order three bytes of the address passed in `theAddress`. If the system is in 32-bit mode, however, `_StripAddress` simply passes back the address unchanged.*

This description is incorrect, and it is located in the wrong chapter of *Inside Macintosh* (it should be in the Memory Manager chapter). The `_StripAddress` trap now takes a `Ptr` as a parameter and returns a `Ptr`:

```
FUNCTION StripAddress(theAddress: Ptr): Ptr;
```

The effect of `_StripAddress` on the passed address depends on the native (bootup) mode the operating system uses, and **not** on the current mode. The following chart shows the results of `_StripAddress` in all of its configurations:

Operating System	24-Bit MMUMode	32-Bit MMUMode
System Software < 7.0	masked with `Lo3Bytes`	masked with `Lo3Bytes`
System 7.0 (24-Bit Bootup)	masked with `Lo3Bytes`	masked with `Lo3Bytes`
System 7.0 (32-Bit Bootup)	*(never in 24-bit mode)*	address unchanged
A/UX 1.1.1	*(never in 24-bit mode)*	address unchanged
A/UX 2.0 (24-Bit Login)	masked with `Lo3Bytes`	masked with `Lo3Bytes`
A/UX 2.0 (32-Bit Login)	*(never in 24-bit mode)*	address unchanged

Should I Call _StripAddress Each Time I Use An Address?

In a word, **no**. Twenty-four-bit addresses are not inherently dangerous—they only cause problems when you access them in 32-bit mode, so unless you are switching modes with the `_SwapMMUMode` trap, you rarely need to call `_StripAddress` inside a "typical" application or driver. Calling `_StripAddress` is **unnecessary** in the following three cases:

Dereferencing A Pointer or A Handle

You don't need to call `_StripAddress` before dereferencing a pointer or a handle unless you are switching the CPU into 32-bit mode with the `_SwapMMUMode` trap. After all, if this is a full-time 32-bit machine, the pointer is always a valid 32-bit address, and if it is a 24-bit machine, your addresses are valid 24-bit addresses unless you switch the machine into 32-bit mode yourself.

Comparing Pointers And Handles For Equality

As long as you don't futz with the high bits of pointers and handles yourself (which you cannot do safely if you want to run in 32-bit mode in any case), you should be able to compare pointers and handles for equality without doing a `_StripAddress`, since the high byte always contains the same "garbage" when you are in 24-bit mode that it did when the pointer or handle was created. There is an exception to this rule, which is discussed in the "Comparing Dereferenced Handles" section later in this Note.

Performing Address Arithmetic

You do not need to call `_StripAddress` before performing address arithmetic, since adding or subtracting two 24-bit addresses always results in the correct 24-bit address, regardless of the high byte values. Random high byte values can cause ordered comparisons on the results of pointer arithmetic to fail, since underflow or overflow could occur, so you should never test the result of address arithmetic against a value (only against `NIL` or 0).

Okay, So When Do I Need To Call _StripAddress?

Ordered Address Comparison

If you need to sort by address or do any other kind of ordered address comparison, you need to call `_StripAddress` on each address before doing any ordered comparisons (>, <, >=, <=). Remember, even though the CPU only uses the lower 24 bits in 24-bit mode, it still uses all 32 bits when performing arithmetic operations.

Comparing Dereferenced Handles (Master Pointers)

If you want to perform any type of comparison on dereferenced handles, you need to call `_StripAddress` on the value first, since the Memory Manager stores its flags in the high byte of the Master Pointer on 24-bit machines, and these flags can change at any time. Of course, this implies that you need to call `_StripAddress` before comparing two pointers for equality if you could be comparing against a dereferenced handle.

On Handles And Pointers Before Dereferencing in 32-Bit Mode

If you are going to switch the machine into 32-bit mode yourself, you need to call `_StripAddress` on all 24-bit pointers and handles that you access while in 32-bit mode. Of course, you had better **not** call `_StripAddress` on a 32-bit address (for example, a 32-bit NuBus™ slot address could generate some very nice fireworks if you strip off its high byte with `_StripAddress` and then try to access the "improved" address). For example, the 32-Bit QuickDraw routine `_GetPixBaseAddr` returns a 32-bit address. Refer to Technical Note #277, 32-Bit QuickDraw: Version 1.2 Features, for more details about 32-Bit QuickDraw and 32-bit addressing.

Program Counter in 32-Bit Mode

This issue is described in depth in Technical Note #228, Use Care When Swapping MMU Mode, so this Note won't go into depth here. Basically, if you are going to switch into 32-bit mode within a code resource, you need to make sure that your Program Counter (`PC`) contains a valid 32-bit address **before** you switch modes.

One not-so-obvious example of this case is a trap patch that lives in a block in the heap. If you pass the address of the block to the `_SetTrapAddress` trap without first calling `_StripAddress` on it and the patched trap is later called in 32-bit mode, bad things happen. Specifically, the high byte of the address contains Memory Manager information, so when the patched trap is called, the `PC` points to never-never land, making results extremely unpredictable. So if you are going to patch a trap using the address of a heap block, call `_StripAddress` on it first.

Filenames Passed To _OpenResFile And _OpenRFPerm

This issue is described in depth in Technical Note #232, Strip With `_OpenResFile` and `_OpenRFPerm`. A patch to these traps calls `_RecoverHandle` on the string passed to these routines, which can cause crashes if `_StripAddress` is not called.

Whaaahh! _StripAddress Is Too Slow! How Can I Make It Faster?

If you follow the guidelines discussed in this Note, speed shouldn't be an issue, since you are calling `_StripAddress` rarely, if at all. Just for the sake of argument, though, let's say that you do call `_StripAddress` inside of a time-critical loop inside an interrupt routine. The solution to this problem is to cache the mask that `_StripAddress` uses to do its work. Here's how:

MPW C:

```
long gStripAddressMask;        /* our cached global mask variable */

/* you can use this macro to do a quick _StripAddress equivalent */
#define QuickStrip(ptr) ((ptr) & gStripAddressMask)
```

```
main()
{
  /* do all of the usual initialization */
  /* cache _StripAddress result */
  gStripAddressMask = 0xffffffff;
  gStripAddressMask = (long) StripAddress((Ptr) gStripAddressMask);
  /* have your program do something useful here… */
}
```

MPW Pascal:

```
PROGRAM StripTease;
VAR
  gStripAddressMask : LONGINT;        { our cached global mask variable }

{ you can use this function to do a quick _StripAddress equivalent }
FUNCTION QuickStrip(thePtr : Ptr) : Ptr;
BEGIN
  QuickStrip := Ptr(BAND(LONGINT(thePtr),gStripAddressMask));
END;

BEGIN
  { do all of the usual initialization }
  { cache _StripAddress result }
  gStripAddressMask := $FFFFFFFF;
  gStripAddressMask := LONGINT(StripAddress(Ptr(gStripAddressMask)));
  { have your program do something useful here… }
END.
```

This technique avoids the overhead of the Trap Dispatcher, works on present and future machines, and it should be fast enough for any application—just call the `QuickStrip` routine (macro to us C weenies) in place of `_StripAddress`.

Further Reference:

- *Inside Macintosh,* Volume V, The OS Utilities
- Technical Note M.OV.32BitClean — The Joy Of Being 32-Bit Clean
- Technical Note M.ME.SwapMMUMode — Use Care When Swapping MMU Mode
- Technical Note M.TB.StripOpenResFile — Strip With `_OpenResFile` And `_OpenRFPerm`
- Technical Note M.IM.32BitQD — 32-Bit QuickDraw: Version 1.2 Features

NuBus is a trademark of Texas Instruments.