

Why does the Mac ABI require 16-byte stack alignment for x86-32?

Asked 15 years, 9 months ago Modified 4 years, 6 months ago

Viewed 8k times



34



I can understand this requirement for the old PPC RISC systems and even for x86-64, but for the old tried-and-true x86? In this case, the stack needs to be aligned on 4 byte boundaries only. Yes, some of the MMX/SSE instructions require 16byte alignments, but if that is a requirement of the callee, then it should ensure the alignments are correct. Why burden **every** caller with this extra requirement? This can actually cause some drops in performance because every call-site must manage this requirement. Am I missing something?

Update: After some more investigation into this and some consultation with some internal colleagues, I have some theories about this:

1. Consistency between the PPC, x86, and x64 version of the OS
2. It seems that the GCC codegen now consistently does a `sub esp,xxx` and then "mov"s the data onto the stack rather than simply doing a "push" instruction. This could actually be faster on some hardware.

3. While this does complicate the call sites a little, there is very little extra overhead when using the default "cdecl" convention where the caller cleans up the stack.

The issue I have with the last item, is that for calling conventions that rely on the callee cleaning the stack, the above requirements **really** "uglify" the codegen. For instance, what some compiler decided to implement a faster register-based calling style for its own internal use (ie any code that isn't intended to be called from other languages or sources)? This stack-alignment thing could negate some of the performance gains achieved by passing some parameters in registers.

Update: So far the only real answers have been consistency, but to me that's a bit too easy of an answer. I have well over 20 years experience with the x86 architecture and if consistency, not performance, or something else concrete, is really the reason then I respectfully suggest that is a bit naive for the developers to require it. They're ignoring nearly three decades of tools and support. Especially if they're expecting tools vendors to quickly and easily adapt their tools for their platform (maybe not... it **is** Apple...) without having to jump through several seemingly unnecessary hoops.

I'll give this topic another day or so then close it...

Related

- [It's my stack frame, I don't care about your stack frame!](#)

macos

memory-alignment

callstack

calling-convention

abi

Share

Improve this question

Follow

edited Jun 20, 2020 at 9:12



Community Bot

1 • 1

asked Mar 4, 2009 at 21:12



Allen Bauer

16.8k • 2 • 59 • 75

-
- 1 Not API. ABI (Application **B**inary Interface. – [Allen Bauer](#)
Mar 7, 2009 at 1:18
-

Related: [Why does System V / AMD64 ABI mandate a 16 byte stack alignment?](#) - modern versions of the i386 System V ABI require the same thing. – [Peter Cordes](#) Jun 7, 2020 at 9:05

Hey @AllenBauer, you say you understand this "even for x86-64" – can you (or anyone) maybe explain the rationale there? I personally don't understand it for any platform (as you say, it's *my* stack frame). – [Tom](#) Feb 22, 2022 at 12:12

10 Answers

Sorted by:

Highest score (default)



From "Intel®64 and IA-32 Architectures Optimization Reference Manual", section 4.4.2:

31



"For best performance, the Streaming SIMD Extensions and Streaming SIMD Extensions 2 require their memory operands to be aligned to 16-byte boundaries. Unaligned data can cause significant performance penalties compared to aligned data."

From Appendix D:

"It is important to ensure that the stack frame is aligned to a 16-byte boundary upon function entry to keep local `__m128` data, parameters, and XMM register spill locations aligned throughout a function invocation."

<http://www.intel.com/Assets/PDF/manual/248966.pdf>

Share Improve this answer

answered May 22, 2009 at 16:28

Follow



rob mayoff

385k ● 68 ● 830 ● 878



6



I am not sure as I don't have first hand proof, but I believe the reason is SSE. SSE is much faster if your buffers are already aligned on a 16 bytes boundary (`movps` vs `movups`), and any x86 has at least `sse2` for mac os x. It can be taken care of by the application user, but the cost is pretty significant. If the overall cost for making it mandatory in the ABI is not too significant, it may worth it. SSE is used quite pervasively in mac os X: accelerate framework, etc...

Share Improve this answer

answered May 7, 2009 at 2:26

Follow



David Cournapeau

80.6k ● 9 ● 68 ● 71

-
- 1 That is the best reason I can come up with as well... however the requirement is that the stack is aligned *before* the call. Once the callee is in control, the stack is no longer aligned! (the return address is now the top of the stack).

– [Allen Bauer](#) May 7, 2009 at 16:02

-
- 4 It doesn't matter so much that the stack pointer is not aligned at that point because you want the arguments to be aligned in memory. So with your typical stack frame, you are guaranteed that you are 16-byte aligned at 8(%ebp), which is your arguments begin. – [Lara Dougan](#) Nov 6, 2009 at 2:08
-



I believe it's to keep it inline with the x86-64 ABI.

5

Share Improve this answer

answered Mar 7, 2009 at 1:12

Follow



Andrew Grant

58.8k ● 22 ● 131 ● 144



-
- 1 That makes sense... to a point. What is the value in this, really? Only tool creators really care about this stuff as most developers simply rely on the tool to "do the right thing."

– [Allen Bauer](#) Mar 7, 2009 at 1:20

Maybe due to the (relatively) short life x86-32 is likely to have on the Mac? – [Andrew Grant](#) Mar 7, 2009 at 2:17



First, note that the 16 bytes alignment is an exception introduced by Apple to the System V IA-32 ABI.

3



The stack alignment is only needed when calling system functions, because many system libraries are using SSE or AltiVec extensions which require the 16 bytes alignment. I found an explicit reference in the [libgmalloc MAN page](#).



You can perfectly handle your stack frame the way you want, but if you try to call a system function with a misaligned stack, you will end up with a **`misaligned_stack_error`** message.

Edit: For the record, you can get rid of alignment problems when compiling with GCC by using the [mstack-realign](#) option.

Share Improve this answer

Follow

edited Jul 31, 2010 at 12:39



[Wouter van Nifterick](#)

24.1k ● 7 ● 81 ● 123


answered Jan 15, 2010 at 12:06



[Laurent Etiemble](#)

27.8k ● 5 ● 58 ● 81

1 The problem is that the compiler doesn't really know that a given call is a system function or not. This means that the only "safe" thing to do is to ensure the stack remains aligned throughout the call chain. We already take advantage of this fact when dealing with hand-coded low-level assembler functions that are known to never end up calling system functions. – [Allen Bauer](#) Jan 15, 2010 at 18:20

Oh another thing, it is kinda hard to "recompile with GCC" since we're in the process of modifying our existing Delphi compiler to target the Mac... GCC isn't involved since we've got our own frontend and code generator/backend that is why this is an issue. – [Allen Bauer](#) Jan 15, 2010 at 18:42 



This is an efficiency issue.

2



Making sure the stack is 16-byte aligned in every function that uses the new SSE instructions adds a lot of overhead for using those instructions, effectively reducing performance.



On the other hand, keeping the stack 16-byte aligned at all times ensures that you can use SSE instructions freely with no performance penalty. There is no cost to this (cost measured in instructions at least). It only involves changing a constant in the prologue of the function.

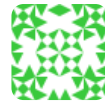
Wasting stack space is cheap, it is probably the hottest part of the cache.

Share Improve this answer

[edited Jan 22, 2010 at 9:35](#)

Follow

answered Jan 8, 2010 at 11:34



[user239558](#)

7,276 ● 1 ● 31 ● 37


-
- 2 I find this to be a very shallow explanation. Why does every function in the call chain have to do this work on the off chance that a SSE instruction *may* be used? If this "overhead" is no big deal, then it is "no big deal" to do it *at the point where the SSE instructions are being used!* I don't require my neighbors to keep *my* house clean.

– [Allen Bauer](#) Jan 8, 2010 at 19:18

-
- 1 Your conclusion is incorrect. Notice the difference between making and keeping. There is no work involved in keeping the stack 16-byte aligned. This simply involves changing a constant in the prologue to ensure that the stack is aligned. I've updated my original answer to underscore this. OTOH, making the stack 16-byte aligned involves work, and has a cost measured in instructions. – [user239558](#) Jan 22, 2010 at 9:36

That is only assuming your compiler's code generator works like GCC's. The world is far more than GCC. If the compiler reserved stack space for all locals and all parameters for all functions the current function calls, that is valid. However, many compilers may not work that way, and in fact trying to *make* them work that way may be too costly. The other thing is that not *all* SSE instructions require alignment, only the MOVxxA instructions do. So even then the subset of potential instructions the system is tuning for is relatively small. An app may *never* use SSE, directly or indirectly. – [Allen Bauer](#) Jan 22, 2010 at 17:40 ✎

-
- 2 The cost analysis is the same whether stack space for all locals is reserved by the prologue or not. Whenever stack space is allocated `sub $xx, %esp` is the way to do it. Keeping the stack 16 byte aligned means `xx` is a multiple of 16. All

the compiler needs to do is to round up. Maybe you could give an example of where this hurts? – [user239558](#) Jan 30, 2010 at 0:59 



2

My guess is that Apple believes everyone just uses XCode (gcc) which aligns the stack for you. So requiring the stack to be aligned so the kernel doesn't have to is just a micro-optimization.



Share Improve this answer

answered Jan 25, 2010 at 7:06



Follow



Mike

1,780 ● 2 ● 18 ● 33



1

While I cannot really answer your question of WHY, you may find the manuals at the following site useful:

<http://www.agner.org/optimize/>



Regarding the ABI, have a look especially at:

http://www.agner.org/optimize/calling_conventions.pdf



Hope that's useful.

Share Improve this answer

answered May 22, 2009 at 17:07

Follow



PhiS

4,640 ● 26 ● 35



Hmm, didn't OS X ABI also do funny RISC like things like passing small structs in registers?

1

So that points to the consistency with other platforms theory.



Come to think of it, the FreeBSD syscall api also aligns 64-bit values. (like e.g. lseek and mmap)



Share Improve this answer

edited May 22, 2009 at 19:34

Follow

answered May 22, 2009 at 16:12



Marco van de Voort

26.3k ● 6 ● 59 ● 93



In order to maintain consistency in kernel. This allows the same kernel to be booted on multiple architectures without modification.

0



Share Improve this answer

edited Nov 27, 2010 at 19:06

Follow



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 7, 2009 at 13:13



PixelSmack

157 ● 10

That's the only thing that seems to be what folks say, however for higher level languages, this is a detail that is (should be) hidden. Any compiled x86-32 ObjC, C, or C++ application would not care since this is an opaque detail.

– Allen Bauer Mar 7, 2009 at 21:02

- 1 A kernel needs to be compatible with the call-stack of user processes because it will need to use that occasionally for working space to handle certain system-calls or interrupts.
– [SingleNegationElimination](#) May 7, 2009 at 2:38

It doesn't seem to hurt the Windows and Linux kernels to not be aligned. What is so special about the MacOS on x86?

- [Allen Bauer](#) May 7, 2009 at 16:04
-



Not sure why no one has considered the possibility of easy portability from legacy PowerPC-based platform?

0

Read this:



http://developer.apple.com/library/mac/#documentation/DeveloperTools/Conceptual/LowLevelABI/100-32-bit_PowerPC_Function_Calling_Conventions/32bitPowerPC.html#//apple_ref/doc/uid/TP40002438-SW20



And then zoomed into "32-bit PowerPC Function Calling Conventions" and finally this:

"These are the embedding alignment modes available in the 32-bit PowerPC environment:

Power alignment mode is derived from the alignment rules used by the IBM XLC compiler for the AIX operating system. It is the default alignment mode for the PowerPC-architecture version of GCC used on AIX and Mac OS X.

Because this mode is most likely to be compatible between PowerPC-architecture

compilers from different vendors, it's typically used with data structures that are shared between different programs."

In view of the legacy PowerPC-based background of OSX, portability is a major consideration - it dictates following the convention all the way back to AIX's XLC compiler. When you think in terms of the need to make sure all the tools and applications will work together with minimal rework, I think it is important to stick to the same legacy ABI as far as possible.

That gives the philosophy, and reading further is the rule explicitly mentioned ("Prolog and Epilog"):

The called function is responsible for allocating its own stack frame, making sure to preserve 16-byte alignment in the stack. This operation is accomplished by a section of code called the prolog, which the compiler places before the body of the subroutine. After the body of the subroutine, the compiler places an epilog to restore the processor to the state it was prior to the subroutine call.

Share Improve this answer

answered Jul 12, 2011 at 2:38

Follow



Peter Teoh

6,665 ● 4 ● 44 ● 59