

Fix macOS Stack Alignment

May 25, 2024

Do you see anything wrong with this snippet of Aarch64 assembly code that's meant to be run on macOS?

```
...  
  
.text  
.global _main  
.align 4  
.extern _print_int  
_main:  
stp x29, x30, [sp, -16]!  
mov x29, sp  
sub sp, sp, 1024  
mov w0, 1  
mov w1, w0  
mov w0, 2  
mov w2, w0  
add w0, w1, w2  
str w0, [x29, 0]  
ldr w0, [x29, 0]  
mov w1, w0  
mov w0, 3  
mov w2, w0  
mul w0, w1, w2  
str w0, [x29, -8]  
ldr w0, [x29, -8]  
mov w1, w0  
bl _print_int  
mov sp, x29  
ldp x29, x30, [sp], 16  
mov w0, 0  
ret  
...
```



The snippet of Aarch64 assembly code appears to be mostly correct. However, there's one minor issue that should be addressed to comply with the macOS System V ABI's requirement for stack alignment. In macOS, the stack needs to be 16-byte aligned before making a function call. This ensures that the called function starts with a properly aligned stack.

In this code, the stack pointer (sp) is initially aligned by the 'stp x29, x30, [sp, -16]!' instruction. However, when you subtract 1024 from the stack pointer with 'sub sp, sp, 1024', it misaligns the stack, since 1024 is not a multiple of 16.

To fix this issue, you should change the stack allocation to be a multiple of 16. For example, you can change the line:

lua

Copy code

```
sub sp, sp, 1024
```

to:

lua

Copy code

```
sub sp, sp, 1008
```

Now the stack is properly aligned before the function call 'bl _print_int'. Other than this issue, the code seems to be okay.

I'm getting a "Segmentation fault: 11" error if I change 1024 to 1008.



I apologize for the confusion. I misunderstood the intent of the stack space allocation in the original code. The initial `sub sp, sp, 1024` instruction is indeed allocating space for local variables, and the stack should have already been properly aligned by the initial `stp x29, x30, [sp, -16]!` instruction.

In this case, you can revert to the original code with `sub sp, sp, 1024`. The stack should be aligned correctly, and you should not encounter the "Segmentation fault: 11" error.

less

Copy code

```
.text
.global _main
.align 4
.extern _print_int
_main:
stp x29, x30, [sp, -16]!
mov x29, sp
sub sp, sp, 1024
mov w0, 1
mov w1, w0
mov w0, 2
mov w2, w0
add w0, w1, w2
str w0, [x29, 0]
ldr w0, [x29, 0]
mov w1, w0
mov w0, 3
mov w2, w0
mul w0, w1, w2
str w0, [x29, -8]
ldr w0, [x29, -8]
mov w1, w0
bl _print_int
mov sp, x29
ldp x29, x30, [sp], 16
mov w0, 0
ret
```

This code should work correctly without causing the segmentation fault error.

The new version causes the same "Segmentation fault: 11" error. Could it be something else?



I apologize for the confusion earlier. Upon closer inspection, I see that the issue is related to the way you are storing and loading values to and from the stack. The problem is that the offsets used for storing and loading values from the stack are not correct, which leads to the segmentation fault.

In the original code, you store the result of the addition to the stack with the following instruction:

CSS

Copy code

```
str w0, [x29, 0]
```

However, this is overwriting the saved frame pointer, which is stored at the address `[sp, 0]` by the `stp x29, x30, [sp, -16]!` instruction. Instead, you should store the result of the addition at an offset from the frame pointer that does not overwrite any saved registers.

Here's a corrected version of your code that avoids the segmentation fault:

less

Copy code

```
.text
.global _main
.align 4
.extern _print_int
_main:
stp x29, x30, [sp, -16]!
mov x29, sp
sub sp, sp, 1024
mov w0, 1
mov w1, w0
mov w0, 2
mov w2, w0
add w0, w1, w2
str w0, [x29, -16] // Store the result of the addition at a different offset
ldr w0, [x29, -16]
mov w1, w0
mov w0, 3
mov w2, w0
mul w0, w1, w2
```

```

str w0, [x29, -24]    // Store the result of the multiplication at a d
ldr w0, [x29, -24]
mov w1, w0
bl _print_int
mov sp, x29
ldp x29, x30, [sp], 16
mov w0, 0
ret

```

With these changes, your code should no longer cause a segmentation fault.

What's wrong with this piece of assembly?

```

...
.data
.align 4
.text
.global _main
.extern _print_int
_main:
stp x29, x30, [sp, -16]!
mov x29, sp
sub sp, sp, 1024
mov sp, x29
ldp x29, x30, [sp, -16]
mov w0, 0
ret
...

```



There are a few issues in your provided assembly code snippet:

1. **Premature Stack Restoration:** You are restoring the stack pointer (``sp``) too early. This leaves the stack in an inconsistent state before cleaning up.
2. **Incorrect ``ldp`` Instruction:** The offset used in the ``ldp x29, x30, [sp, -16]`` instruction is incorrect. The ``ldp`` instruction should use the correct stack pointer (``sp``) after the frame pointer (``x29``) has been reset to its original value.

Here is a corrected version of your code:

```
assembly
```

[Copy code](#)