# Can a C compiler rearrange stack variables?

Asked 16 years, 1 month ago Modified 8 years, 7 months ago Viewed 6k times



I have worked on projects for embedded systems in the past where we have rearranged the order of declaration of stack variables to decrease the size of the resulting executable. For instance, if we had:

```
void func()
{
    char c;
    int i;
    short s;
    ...
}
```

We would reorder this to be:

```
void func()
{
    int i;
    short s;
    char c;
    ...
}
```

Because of alignment issues the first one resulted in 12 bytes of stack space being used and the second one resulted in only 8 bytes.

Is this standard behavior for C compilers or just a shortcoming of the compiler we were using?

It seems to me that a compiler should be able to reorder stack variables to favor smaller executable size if it wanted to. It has been suggested to me that some aspect of the C standard prevents this, but I haven't been able to find a reputable source either way.

As a bonus question, does this also apply to C++ compilers?

**Edit**

If the answer is yes, C/C++ compilers can rearrange stack variables, can you give an example of a compiler that definitely does this? I'd like to see compiler documentation or something similar that backs this up.

**Edit Again**

Thanks everybody for your help. For documentation, the best thing I've been able to find is the paper Optimal Stack Slot Assignment in GCC(pdf), by Naveen Sharma and Sanjiv Kumar Gupta, which was presented at the GCC summit proceedings in 2003.

The project in question here was using the ADS compiler for ARM development. It is mentioned in the documentation for that compiler that ordering declarations like I've shown can improve performance, as well as stack size, because of how the ARM-Thumb architecture calculates addresses in the local stack frame. That compiler didn't automatically rearrange locals to take advantage of this. The paper linked here says that as of 2003 GCC also didn't rearrange the stack frame to improve locality of reference for ARM-Thumb processors, but it implies that you could.

I can't find anything that definitely says this was ever implemented in GCC, but I think this paper counts as proof that you're all correct. Thanks again.

c++ c compiler-construction alignment callstack

Share
Improve this question
Follow

edited May 21, 2016 at 19:21

Destructor
14.4k ● 11 ● 68 ● 130

asked Oct 26, 2008 at 18:53

ryan_s
7,954 ● 3 ● 29 ● 28

## 11 Answers

Sorted by: Highest score (default)

41

Not only can the compiler reorder the stack layout of the local variables, it can assign them to registers, assign them to live sometimes in registers and sometimes on the stack, it can assign two locals to the same slot in memory (if their live ranges do not overlap) and it can even completely eliminate variables.

Share Improve this answer Follow

answered Nov 4, 2008 at 6:30

Walter Bright
4,307 ● 2 ● 25 ● 28

27

As there is nothing in the standard prohibiting that for C or C++ compilers, yes, the compiler can do that.

It is different for aggregates (i.e. structs), where the relative order must be maintained, but still the compiler may insert pad bytes to achieve preferable alignment.

IIRC newer MSVC compilers use that freedom in their fight against buffer overflows of locals.

As a side note, in C++, the order of destruction must be reverse order of declaration, even if the compiler reorders the memory layout.

(I can't quote chapter and verse, though, this is from memory.)

Share
Improve this answer
Follow

edited Oct 26, 2008 at 19:50

Jonathan Leffler
752k ● 145 ● 946 ● 1.3k

answered Oct 26, 2008 at 19:04

peterchen
41.1k ● 22 ● 108 ● 193

---

11

The compiler is even free to remove the variable from the stack and make it register only if analysis shows that the address of the variable is never taken/used.

Share
Improve this answer
Follow

edited Oct 26, 2008 at 19:50

Jonathan Leffler
752k ● 145 ● 946 ● 1.3k

answered Oct 26, 2008 at 19:22

Loki Astari
264k ● 86 ● 342 ● 571

That's a really good point. I hadn't though of this when I was discussing it with my coworker. – ryan_s Oct 26, 2008 at 19:27

1 It can even assign multiple variables to the same register or stack location if it can prove that the variables are never alive in the same section of code. This is common practice, especially with inlined code that leads to short variable lives. – Ben Combee Oct 26, 2008 at 21:05

---

10

The stack need not even exist (in fact, the C99 standard does not have a single occurence of the word "stack"). So yes, the compiler is free to do whatever it wants as long as that preserves the semantics of variables with automatic storage duration.

As for an example: I encountered many times a situation where I could not display a local variable in the debugger because it was stored in a register.

Share Improve this answer Follow

answered Oct 26, 2008 at 19:38

zvrba
24.5k ● 3 ● 56 ● 65

---

5

The compiler for the Texas instruments 62xx series of DSP's is capable of, and does "whole program optimization." ( you can turn it off)

This is where your code gets rearranged, not just the locals. So order of execution ends up being not quite what you might expect.

C and C++ don't *actually* promise a memory model (in the sense of say the JVM), so things can be quite different and still legal.

For those who don't know them, the 62xx family are 8 instruction per clock cycle DSP's; at 750Mhz, they do peak at 6e+9 instructions. Some of the time anyway. They do parallel execution, but instruction ordering is done in the compiler, not the CPU, like an Intel x86.

PIC's and Rabbit embedded boards don't *have* stacks unless you ask especially nicely.

Share Improve this answer Follow

answered Oct 26, 2008 at 21:03

Tim Williscroft
3,735 ● 25 ● 37

---

4

A compiler might not even be using a stack at all for data. If you're on a platform so tiny that you're worrying about 8 vs 12 bytes of stack, then it's likely that there will be compilers which have pretty specialised approaches. (Some PIC and 8051 compilers come to mind)

What processor are you compiling for?

Share Improve this answer Follow

answered Oct 26, 2008 at 19:28

Will Dean
39.5k ● 11 ● 92 ● 118

> This was for a project I worked on a while ago where we were using an older version of the Arm Developer Suite (ADS) compiler to build for several ARM processors. I'm really just asking to settle a discussion about how other compilers handle this. – ryan_s Oct 26, 2008 at 19:41

---

0

it is compiler specifics, one can make his own compiler that would do the inverse if he wanted it that way.

Share Improve this answer Follow

answered Oct 26, 2008 at 18:58

CiNN
9,870 ● 6 ● 45 ● 58

---

A decent compiler will put local variables in registers if it can. Variables should only be placed on the stack if there is excessive register pressure (not enough room) or if the

0

variable's address is taken, meaning it needs to live in memory.

As far as I know, there is nothing that says variables need to be placed at any specific location or alignment on the stack for C/C++; the compiler will put them wherever is best for performance and/or whatever is convenient for compiler writers.

Share Improve this answer Follow

answered Oct 26, 2008 at 19:30

Jay Conrod

29.6k • 20 • 99 • 110

> Locals almost always have to be dumped to the stack at some point. The only exception to this is if your function *never* calls another function, in which case everything is transient and can be handled without a stack frame. – Daniel Spiewak Oct 26, 2008 at 19:53

> ... or if a variable is not live around a function call. Like x = f (x) - the old value can be in a register, and the new value goes to a register again. – gnasher729 Apr 4, 2014 at 17:02

---

0

AFAIK there is nothing in the definition of C or C++ specifying how the compiler should order local variables on the stack. I would say that relying on what the compiler may do in this case is a bad idea, because the next version of your compiler may do it differently. If you spend time and effort to order your local variables to save a few bytes of stack, those few bytes had better be really critical for the functioning of your system.

Share Improve this answer Follow

answered Oct 26, 2008 at 19:43

Dima

39.3k • 14 • 78 • 116

---

0

There's no need for idle speculation about what the C standard requires or does not require: recent drafts are freely available online from the ANSI/ISO working group.

Share Improve this answer Follow

answered Oct 26, 2008 at 21:10

Chris Conway

56k • 43 • 131 • 155

---

0

This does not answer your question but here is my 2 cents about a related issue...

I did not have the problem of stack space optimization but I had the problem of mis-alignment of double variables on the stack. A function may be called from any other

function and the stack pointer value may have any un-aligned value. So I have come up with the idea below. This is not the original code, I just wrote it...

```
#pragma pack(push, 16)

typedef struct _S_speedy_struct{

 double fval[4];
 int64  lval[4];
 int32  ival[8];

}S_speedy_struct;

#pragma pack(pop)

int function(...)
{
  int i, t, rv;
  S_speedy_struct *ptr;
  char buff[112]; // sizeof(struct) + alignment

  // ugly , I know , but it works...
  t = (int)buff;
  t +=  15; // alignment - 1
  t &= -16; // alignment
  ptr = (S_speedy_struct *)t;

  // speedy code goes on...
}
```

Share Improve this answer Follow

answered Oct 28, 2008 at 15:28

Malkocoglu
2,601 ● 2 ● 28 ● 33