Which compiles to faster code: "n * 3" or "n+(n*2)"?

Asked 16 years, 3 months ago Modified 3 months ago Viewed 2k times

7

Which compiles to faster code: "ans = n * 3" or "ans = n+(n*2)"?

Assuming that n is either an int or a long, and it is is running on a modern Win32 Intel box.

Would this be different if there was some dereferencing involved, that is, which of these would be faster?

```
long    a;
long    *pn;
long     ans;

...
*pn = some_number;
ans = *pn * 3;
```

Or

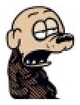
```
ans = *pn+(*pn*2);
```

Or, is it something one need not worry about as optimizing compilers are likely to account for this in any case?

c++ c performance optimization

Share Improve this question Follow

edited Aug 27 at 10:58


Jonas
128k ● 100 ● 326 ● 405

asked Sep 10, 2008 at 10:44

David L Morris
1,501 ● 1 ● 12 ● 20

11 Answers

Sorted by: Highest score (default)

57

IMO such micro-optimization is not necessary unless you work with some exotic compiler. I would put readability on the first place.

Share Improve this answer Follow

answered Sep 10, 2008 at 10:48

aku
124k ● 33 ● 176 ● 203

15

It doesn't matter. Modern processors can execute an integer MUL instruction in one clock cycle or less, unlike older processers which needed to perform a series of shifts and adds internally in order to perform the MUL, thereby using multiple cycles. I would bet that

```
MUL EAX,3
```

executes faster than

```
MOV EBX,EAX
SHL EAX,1
ADD EAX,EBX
```

The last processor where this sort of optimization might have been useful was probably the 486. (yes, this is biased to intel processors, but is probably representative of other architectures as well).

In any event, any reasonable compiler should be able to generate the smallest/fastest code. So always go with readability first.

Share Improve this answer Follow

answered Sep 10, 2008 at 11:30

Ferruccio

101k ● 38 ● 229 ● 303

3 I really doubt the MUL executes faster, once the latency and inflexibility about which registers you can use is taken into account. Moreover on x86, LEA, not the 3-instruction sequence you gave, would be used by any decent compiler

for both 3*n and n+2*n. – R.. GitHub STOP HELPING ICE Jul 1, 2011 at 2:26

1 True, but LEA is only useful when multiplying by a small set of constants (2, 3, 4, 5, 8 & 9 if I recall correctly). Anyway my point was to let the compiler figure out the fastest code. – Ferruccio Aug 1, 2011 at 17:21

10

As it's easy to measure it yourself, why don't do that? (Using `gcc` and `time` from cygwin)

```
/* test1.c */
int main()
{
    int result = 0;
    int times = 1000000000;
    while (--times)
        result = result * 3;
    return result;
}

machine:~$ gcc -O2 test1.c -o test1
machine:~$ time ./test1.exe

real    0m0.673s
user    0m0.608s
sys     0m0.000s
```

Do the test for a couple of times and repeat for the other case.

If you want to peek at the assembly code, `gcc -S -O2 test1.c`

Share Improve this answer answered Sep 10, 2008 at 11:06

Follow

Antti Kissaniemi
19.2k ● 13 ● 56 ● 47

Unfortunately this is a bad example - with i686-apple-darwin8-gcc-4.0.1, it completely removes the "result = result * 3" from the loop since it's always zero. Changing the initial condition to "result = 1" gives a better result.
– Adam Rosenfield Nov 20, 2008 at 1:10

1 or better, create an array of random numbers and process that, so the compiler can't make any assumptions. – DarenW Dec 27, 2009 at 0:25

4

This would depend on the compiler, its configuration and the surrounding code.

You should not try and guess whether things are 'faster' without taking measurements.

In general you should not worry about this kind of nanoscale optimisation stuff nowadays - it's almost always a complete irrelevance, and if you were genuinely working in a domain where it mattered, you would already be using a profiler and looking at the assembly language output of the compiler.

Share Improve this answer

Follow

answered Sep 10, 2008 at 10:51

Will Dean
39.5k ● 11 ● 92 ● 118

4

It's not difficult to find out what the compiler is doing with your code (I'm using DevStudio 2005 here). Write a simple program with the following code:

```
int i = 45, j, k;
j = i * 3;
k = i + (i * 2);
```

Place a breakpoint on the middle line and run the code using the debugger. When the breakpoint is triggered, right click on the source file and select "Go To Disassembly". You will now have a window with the code the CPU is executing. You will notice in this case that the last two lines produce exactly the same instructions, namely, "lea eax,[ebx+ebx*2]" (not bit shifting and adding in this particular case). On a modern IA32 CPU, it's probably more efficient to do a straight MUL rather than bit shifting due to pipelineing nature of the CPU which incurs a penalty when using a modified value too soon.

This demonstrates what aku is talking about, namely, compilers are clever enough to pick the best instructions for your code.

Share Improve this answer Follow

edited Jul 3, 2012 at 14:32
user142162

answered Sep 10, 2008 at 11:45
Skizz
71k ● 10 ● 74 ● 109

I don't the pipeline is an issue. The arithmetic unit can probably hadle ebx+ebx*2 internally in one step. – Artelius Jan 6, 2009 at 21:55

2

It does depend on the compiler you are actually using, but very probably they translate to the same code.

You can check it by yourself by creating a small test program and checking its disassembly.

Share Improve this answer Follow

answered Sep 10, 2008 at 10:48

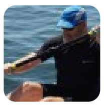
Vinko Vrsalovic
340k ● 55 ● 340 ● 373

1

Most compilers are smart enough to decompose an integer multiplication into a series of bit shifts and adds. I don't know about Windows compilers, but at least with gcc you can get it to spit out the assembler, and if you look at that you can probably see identical assembler for both ways of writing it.

Share Improve this answer Follow

answered Sep 10, 2008 at 10:49



Paul Tomblin
183k ● 59 ● 323 ● 410

1

It doesn't care. I think that there are more important things to optimize. How much time have you invested thinking and writing that question instead of coding and testing by yourself?

:-)

Share Improve this answer Follow

answered Nov 20, 2008 at 1:00

Angel
798 ● 5 ● 7

1

As long as you're using a decent optimising compiler, just *write code that's easy for the compiler to understand*. This makes it easier for the compiler to perform clever optimisations.

You asking this question indicates that an optimising compiler knows more about optimisation than you do. So trust the compiler. Use `n * 3`.

Have a look at this answer as well.

Share Improve this answer Follow

edited May 23, 2017 at 12:06

Community Bot
1 ● 1

answered Nov 20, 2008 at 1:15

Artelius
49k ● 13 ● 92 ● 106

Compilers are good at optimising code such as yours. Any modern compiler would produce the same code for both cases and additionally replace `* 2` by a left shift.

0

Share Improve this answer

Follow

answered Sep 10, 2008 at 10:50

Konrad Rudolph

545k ● 139 ● 956 ● 1.2k

Don't be so sure :) I saw some really weird compilers for embedded software development. – aku Sep 10, 2008 at 10:58

1 In embedded systems, almost all conventional wisdom ends. ;-) – Konrad Rudolph Sep 10, 2008 at 11:12

Trust your compiler to optimize little pieces of code like that. Readability is much more important at the code level. True optimization should come at a higher level.

0

Share Improve this answer

Follow

answered Sep 15, 2008 at 14:09

Mark D

55 ● 1