Which is faster: while(1) or while(2)?

Asked 10 years, 5 months ago Modified 7 months ago Viewed 100k times

618

This was an interview question asked by a senior manager.

Which is faster?

```
while(1) {
    // Some code
}
```

or

```
while(2) {
    //Some code
}
```

I said that both have the same execution speed, as the expression inside `while` should finally evaluate to `true` or `false`. In this case, both evaluate to `true` and there are no extra conditional instructions inside the `while` condition. So, both will have the same speed of execution and I prefer while (1).

But the interviewer said confidently: "Check your basics. `while(1)` is faster than `while(2)`." (He was not testing my confidence)

Is this true?

c performance while-loop

Share
Improve this question
Follow

edited May 17 at 10:01
Jan Schultke
38.3k ●8 ●87 ●168

asked Jul 20, 2014 at 7:32
Nikole
4,749 ●3 ●17 ●15

218 A half-decent compiler will optimise both forms to nothing. – user1864610 Jul 20, 2014 at 7:36

72 In optimized build every while(n), n != 0 or for(;;) will be translated to Assembly endless loop with label in the beginning and goto in the end. Exactly the same code, the same performance. – Alex F Jul 20, 2014 at 7:48

66 Not surprising, a stock optimize brings `0x100000f90: jmp 0x100000f90` (address varies, obviously) for *both* snippets. The interviewer probably hedged on a register test vs. a simple

flagged jump. Both the question, and their supposition, is lame. – WhozCraig Jul 20, 2014 at 7:49

54 This question by the interviewer falls under the same auspices as dilbert.com/strips/comic/1995-11-17 - you will meet someone who genuinely believes what they are saying regardless of the quotient of stupidity in their statement. Simply choose from the following: a deep breat, swear, laugh, cry, some combination of the above :) – GMasucci Jul 22, 2014 at 8:13

5 You should always ask why when you encounter this sort of thing. There's a lot of misinformation out there, and interviewers are not immune. Interviewer is dead flat wrong in this case. Compiler writer speaking here. It is also a rather stupid interview question: what difference does it make in practice? Who is really going to write `while (2)`? Just an opportunity for the interviewer to show off his ignorance. – user207421 Sep 27, 2019 at 7:00

23 Answers

Sorted by: Highest score (default)

717

Both loops are infinite, but we can see which one takes more instructions/resources per iteration.

Using gcc, I compiled the two following programs to assembly at varying levels of optimization:

```
int main(void) {
    while(1) {}
    return 0;
}
```

```
int main(void) {
    while(2) {}
    return 0;
}
```

Even with no optimizations (`-O0`), **the generated assembly was identical for both programs**. Therefore, there is no speed difference between the two loops.

For reference, here is the generated assembly (using `gcc main.c -S -masm=intel` with an optimization flag):

With `-O0`:

```
    .file   "main.c"
    .intel_syntax noprefix
    .def    __main; .scl    2;  .type   32; .endef
    .text
    .globl  main
    .def    main;   .scl    2;  .type   32; .endef
    .seh_proc   main
main:
    push    rbp
    .seh_pushreg    rbp
    mov rbp, rsp
    .seh_setframe   rbp, 0
    sub rsp, 32
    .seh_stackalloc 32
    .seh_endprologue
    call    __main
.L2:
    jmp .L2
    .seh_endproc
    .ident  "GCC: (tdm64-2) 4.8.1"
```

With `-O1`:

```
    .file   "main.c"
    .intel_syntax noprefix
    .def    __main; .scl    2;  .type   32; .endef
    .text
    .globl  main
    .def    main;   .scl    2;  .type   32; .endef
    .seh_proc   main
main:
    sub rsp, 40
    .seh_stackalloc 40
    .seh_endprologue
    call    __main
.L2:
    jmp .L2
    .seh_endproc
    .ident  "GCC: (tdm64-2) 4.8.1"
```

With `-O2` and `-O3` (same output):

```
    .file   "main.c"
    .intel_syntax noprefix
    .def    __main; .scl    2;  .type   32; .endef
    .section    .text.startup,"x"
    .p2align 4,,15
    .globl  main
    .def    main;   .scl    2;  .type   32; .endef
    .seh_proc   main
main:
    sub rsp, 40
    .seh_stackalloc 40
    .seh_endprologue
    call    __main
.L2:
    jmp .L2
    .seh_endproc
    .ident  "GCC: (tdm64-2) 4.8.1"
```

In fact, the assembly generated for the loop is identical for every level of optimization:

```
 .L2:
    jmp .L2
    .seh_endproc
    .ident  "GCC: (tdm64-2) 4.8.1"
```

The important bits being:

```
.L2:
    jmp .L2
```

I can't read assembly very well, but this is obviously an unconditional loop. The `jmp` instruction unconditionally resets the program back to the `.L2` label without even comparing a value against true, and of course immediately does so again until the program is somehow ended. This directly corresponds to the C/C++ code:

```
L2:
    goto L2;
```

Edit:

Interestingly enough, even with *no optimizations*, the following loops all produced the exact same output (unconditional `jmp`) in assembly:

```
while(42) {}

while(1==1) {}

while(2==2) {}

while(4<7) {}

while(3==3 && 4==4) {}

while(8-9 < 0) {}

while(4.3 * 3e4 >= 2 << 6) {}

while(-0.1 + 02) {}
```

And even to my amazement:

```
#include<math.h>

while(sqrt(7)) {}

while(hypot(3,4)) {}
```

Things get a little more interesting with user-defined functions:

```
int x(void) {
    return 1;
}

while(x()) {}
```

```
#include<math.h>

double x(void) {
    return sqrt(7);
}

while(x()) {}
```

At `-O0`, these two examples actually call `x` and perform a comparison for each iteration.

First example (returning 1):

```
.L4:
    call    x
    testl   %eax, %eax
    jne .L4
    movl    $0, %eax
    addq    $32, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .ident  "GCC: (tdm64-2) 4.8.1"
```

Second example (returning `sqrt(7)`):

```
.L4:
    call    x
    xorpd   %xmm1, %xmm1
    ucomisd %xmm1, %xmm0
    jp  .L4
    xorpd   %xmm1, %xmm1
    ucomisd %xmm1, %xmm0
    jne .L4
    movl    $0, %eax
    addq    $32, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .ident  "GCC: (tdm64-2) 4.8.1"
```

However, at `-O1` and above, they both produce the same assembly as the previous examples (an unconditional `jmp` back to the preceding label).

TL;DR

Under GCC, the different loops are compiled to identical assembly. The compiler evaluates the constant values and doesn't bother performing any actual comparison.

The moral of the story is:

- There exists a layer of translation between C source code and CPU instructions, and this layer has important implications for performance.
- Therefore, performance cannot be evaluated by only looking at source code.
- The compiler **should be** smart enough to optimize such trivial cases. Programmers **should not** waste their time thinking about them in the vast majority of cases.

Share

Improve this answer

Follow

edited Sep 15, 2022 at 16:14
Mike 'Pomax' Kamermans
53.3k ● 17 ● 125 ● 173

answered Jul 20, 2014 at 8:03
ApproachingDarknessFish
14.3k ● 8 ● 41 ● 81

211 Maybe the interviewer wasn't using gcc – M.M Jul 20, 2014 at 10:39

127 @Matt McNabb That's a good point, but if the interviewer was relying on compiler-specific optimisations, then they need to be very explicit about that in their question, and they need to accept the answer "there is no difference" as being correct for some (most?) compilers. – Jonathan Hartley Jul 21, 2014 at 11:22

121 For the sake of removing any doubt, I tested this in clang 3.4.2, and both loops produce the same assembly at every `-O` level. – Martin Tournoij Jul 21, 2014 at 12:08

20 I don't find this at all surprising since everything you have placed in the conditional section of the loops are compile-time constants. As such, I would suspect a compiler would see they the loops will always be true or false and either respectiively simply `jmp` back to the beginning or remove the loop completely. – sherrellbc Jul 21, 2014 at 12:51

12 @hippietrail I don't see how the content (or lack hereof) of the loop could possibly effect these optimizations (excepting the possibility of any `break` statements), but I just tested it anyway and no, even with code inside the loop the jump is absolute and unconditional for both `while(1)` and `while(2)`. Feel free to test the others yourself if you're actually concerned. – ApproachingDarknessFish Jul 26, 2014 at 4:35

317

Yes, `while(1)` is much faster than `while(2)`, *for a human to read!* If I see `while(1)` in an unfamiliar codebase, I immediately know what the author intended, and my eyeballs can continue to the next line.

If I see `while(2)`, I'll probably halt in my tracks and try to figure out why the author didn't write `while(1)`. Did the author's finger slip on the keyboard? Do the maintainers of this codebase use `while(n)` as an obscure commenting mechanism to make loops look different? Is it a crude workaround for a spurious warning in some broken static analysis tool? Or is this a clue that I'm reading generated code? Is it a bug resulting from an ill-advised find-and-replace-all, or a bad merge, or a cosmic ray? Maybe this line of code is supposed to do something dramatically different. Maybe it was supposed to read `while(w)` or `while(x2)`. I'd better find the author in the file's history and send them a "WTF" email... and now I've broken my mental context. The `while(2)` might consume several minutes of my time, when `while(1)` would have taken a fraction of a second!

I'm exaggerating, but only a little. Code readability is really important. And that's worth mentioning in an interview!

Share Improve this answer Follow

answered Jul 22, 2014 at 5:51
Chris Culter
4,546 ● 2 ● 16 ● 31

8 Absolutely, this is NO exaggeration at all. Definitely the `svn-annotate` or `git blame` (or whatever) will be used here, and in general it takes minutes to load a file blame history. Then just finally to decide "ah I understand, the author wrote this line when fresh out of highschool", just lost 10 minutes... – v.oddou Jul 29, 2014 at 6:08

18 Voted down because I'm sick of conventions. The developer has this puny opportunity to write a number he likes, then someone boring comes and nags about why it isn't `1`. – Utkan Gezer Aug 11, 2014 at 3:34

6 Downvoted due to rhetorics. Reading while(1) is exactly the same speed as while(2). – hdante Nov 29, 2014 at 12:05

1 `while(l)` looks a lot like `while(1)` ... Even innocent looking statements can hide subtile bugs. – chqrlie Dec 12, 2016 at 2:16

7 I went through the exact same mental process as described in this answer a minute ago when I saw `while(2)` in the code (down to considering "Do the maintainers of this codebase use while(n) as an obscure commenting mechanism to make loops look different?"). So no, you're not exaggerating! – Hisham H M Jan 21, 2017 at 18:25

156

The existing answers showing the code generated by a particular compiler for a particular target with a particular set of options do not fully answer the question -- unless the question was asked in that specific context ("Which is faster using gcc 4.7.2 for x86_64 with default options?", for example).

As far as the language definition is concerned, in the *abstract machine* `while (1)` evaluates the integer constant `1`, and `while (2)` evaluates the integer constant `2`; in both cases the result is compared for equality to zero. The language standard says absolutely nothing about the relative performance of the two constructs.

I can imagine that an extremely naive compiler might generate different machine code for the two forms, at least when compiled without requesting optimization.

On the other hand, C compilers absolutely must evaluate *some* constant expressions at compile time, when they appear in contexts that require a constant expression. For example, this:

```
int n = 4;
switch (n) {
    case 2+2: break;
    case 4:   break;
}
```

requires a diagnostic; a lazy compiler does not have the option of deferring the evaluation of `2+2` until execution time. Since a compiler has to have the ability to evaluate constant expressions at compile time, there's no good reason for it not to take advantage of that capability even when it's not required.

The C standard (N1570 6.8.5p4) says that

> An iteration statement causes a statement called the *loop body* to be executed repeatedly until the controlling expression compares equal to 0.

So the relevant constant expressions are `1 == 0` and `2 == 0`, both of which evaluate to the `int` value `0`. (These comparison are implicit in the semantics of the `while` loop; they don't exist as actual C expressions.)

A perversely naive compiler *could* generate different code for the two constructs. For example, for the first it could generate an unconditional infinite loop (treating `1` as a special case), and for the second it could generate an explicit run-time comparison equivalent to `2 != 0`. But I've never encountered a C compiler that would actually behave that way, and I seriously doubt that such a compiler exists.

Most compilers (I'm tempted to say all production-quality compilers) have options to request additional optimizations. Under such an option, it's even less likely that any compiler would generate different code for the two forms.

If your compiler generates different code for the two constructs, first check whether the differing code sequences actually have different performance. If they do, try compiling again with an optimization option (if available). If they still differ, submit a bug report to the compiler vendor. It's not (necessarily) a bug in the sense of a failure to conform to the C standard, but it's almost certainly a problem that should be corrected.

Bottom line: `while (1)` and `while(2)` *almost* certainly have the same performance. They have exactly the same semantics, and there's no good reason for any compiler not to generate identical code.

And though it's perfectly legal for a compiler to generate faster code for `while(1)` than for `while(2)`, it's equally legal for a compiler to generate faster code for `while(1)` than for another occurrence of `while(1)` in the same program.

(There's another question implicit in the one you asked: How do you deal with an interviewer who insists on an incorrect technical point. That would probably be a good question for the Workplace site).

Share

Improve this answer

Follow

edited May 11, 2017 at 1:50

answered Jul 21, 2014 at 0:35

Keith Thompson
263k ● 46 ● 440 ● 653

8 "In this case, the relevant (implicit) constant expressions are 1 != 0 and 2 != 0, both of which evaluate to the int value 1" ... This is overly complicated, and inaccurate. The standard simply says that the controlling expression of `while` must be of scalar type and the loop body is repeated until the expression compares equal to 0. It doesn't say that there is an implicit `expr != 0` that is evaluated ... that would require the result of that -- 0 or 1 -- in turn be compared to 0, ad infinitum. No, the expression is compared to 0, but that comparison doesn't produce a value. P.S. I upvoted. – Jim Balter Jul 21, 2014 at 8:38

3 @JimBalter: I see your point, and I'll update my answer to address it. What I meant, though, was that the standard's wording "... until the controlling expression compares equal to 0" implies evaluating `<expr> == 0`; that's what "compares equal to 0" *means* in C. That comparison is part of the semantics of the `while` loop. There is no implication, either in the standard or in my answer, that the result needs to be compared to `0` again. (I should have written `==` rather than `!=`.) But that part of my answer was unclear, and I'll update it. – Keith Thompson Jul 21, 2014 at 14:56

1 ""compares equal to 0" means in C" -- but that language is in the *standard*, not in C ... the implementation does the comparison, it doesn't generate a C language comparison. You write "both of which evaluate to the int value 1" -- but no such evaluation ever occurs. If you look at the code generated for `while (2)`, `while (pointer)`, `while (9.67)`, by the most naive, unoptimized compiler, you won't see any code generated that yields `0` or `1`. "I should have written == rather than !=" -- no, that wouldn't have made any sense. – Jim Balter Jul 21, 2014 at 19:30

2 @JimBalter: Hmmm. I don't mean to suggest that the "compares equal to 0" implies the existence of an `... == 0` C expression. My point is that both the "compares equal to 0" required by the standard's description of `while` loops and an explicit `x == 0` expression logically imply the same operation. And I think that a *painfully* naive C compiler might generate code that generates an `int` value of `0` or `1` for any `while` loop -- though I don't believe any actual compiler is quite that naive. – Keith Thompson Jul 21, 2014 at 20:24

12 Note: This is already a The Workplace question: workplace.stackexchange.com/questions/4314/... – Joe Jul 22, 2014 at 18:33

150

Wait a minute. The interviewer, did he look like this guy?

CHECK YOUR BASICS. WHILE(1) IS FASTER THAN WHILE(2)

It's bad enough that *the interviewer himself has failed* this interview, what if other programmers at this company have "passed" this test?

No. Evaluating the statements `1 == 0` and `2 == 0` *should be* equally fast. We *could imagine* poor compiler implementations where one might be faster than the other. But there's no *good* reason why one should be faster than the other.

Even if there's some obscure circumstance when the claim would be true, programmers should not be evaluated based on knowledge of obscure (and in this case, creepy) trivia. Don't worry about this interview, the best move here is to walk away.

Disclaimer: *This is NOT an original Dilbert cartoon. This is merely a mashup.*

Share

Improve this answer

Follow

edited Jul 30, 2014 at 8:33

answered Jul 24, 2014 at 18:31

janos
124k ●31 ●238 ●248

no but really, we can all imagine fairly easily that all compilers written by serious companies will produce reasonable code. let's take the "non optimized case" /O0, maybe it will end up like anatolyg has posted. Then its a question of CPU, will the `cmp` operand run in less cycles comparing 1 to 0 than 2 to 0 ? how many cycles does it take to execute cmp in general ? is it variable according to bit patterns ? are they more "complex" bit patterns that slow down `cmp`

? I don't know personally. you could imagine a super idiotic implementation checking bit by bit from rank 0 to n (e.g. n=31). – v.oddou Jul 29, 2014 at 6:16

5 That's my point too: the `cmp` operand *should be* equally fast for 1 and 200. Probably we *could imagine* idiotic implementations where this is not the case. But can we imagine a *non-idiotic* implementation where `while(1)` is faster than `while(200)`? Similarly, if in some pre-historic age, the only available implementation was idiotic like that, should we fuss about it today? I don't think so, this is pointy haired boss talk, and a real gem at that! – janos Jul 29, 2014 at 8:55

@v.ouddou "will the cmp operand run in less cycles comparing 1 to 0 than 2 to 0" -- No. You should learn what a cycle is. "I don't know personally. you could imagine a super idiotic implementation checking bit by bit from rank 0 to n" -- or the other way, still making the interviewer a clueless idiot. And why worry about checking bit by bit? The implementation could be a man in a box who decides to take a lunch break in the middle of evaluating your program. – Jim Balter Sep 18, 2014 at 9:56

Loading 0 or 1 into a register can be faster than loading 2 depending on CPU architecture. But the guy asking this question clearly doesn't realize that the test in the loop compiles away to nothing. – Joshua Sep 25, 2020 at 16:13

82

Your explanation is correct. This seems to be a question that tests your self-confidence in addition to technical knowledge.

By the way, if you answered

> Both pieces of code are equally fast, because both take infinite time to complete

the interviewer would say

> But `while (1)` can do more iterations per second; can you explain why? (this is nonsense; testing your confidence again)

So by answering like you did, you saved some time which you would otherwise waste on discussing this bad question.

Here is an example code generated by the compiler on my system (MS Visual Studio 2012), with optimizations turned off:

```
yyy:
    xor eax, eax
    cmp eax, 1     (or 2, depending on your code)
    je xxx
    jmp yyy
```

```
xxx:
    ...
```

With optimizations turned on:

```
xxx:
    jmp xxx
```

So the generated code is exactly the same, at least with an optimizing compiler.

Share
Improve this answer
Follow

edited Jul 21, 2014 at 9:07
Lightness Races in Orbit
385k ● 77 ● 663 ● 1.1k

answered Jul 20, 2014 at 7:59
anatolyg
28.3k ● 9 ● 64 ● 140

28 This code is really what the compiler outputs on my system. I didn't make it up. – anatolyg Jul 20, 2014 at 10:48

11 icepack "The operand for while is of boolean type" -- utter nonsense. Are you the interviewer? I suggest that you become familiar with the C language and its standard before making such claims. – Jim Balter Jul 21, 2014 at 7:38

30 "I didn't make it up." -- Please don't pay any attention to icepack, who is talking nonsense. C has no boolean type (it does have _Bool in stdbool.h, but that's not the same, and the semantics of `while` long preceded it) and the operand of `while` is not of boolean or _Bool or any other specific type. The operand of `while` can be *any* expression ... the while breaks on 0 and continues on non-0. – Jim Balter Jul 21, 2014 at 7:49

38 "Both pieces of code are equally fast, because both take infinite time to complete" made me think of something interesting. The only ways to terminate an infinite loop would be for a bit to be flipped by a charged particle or the hardware failing: changing the statement from `while (00000001) {}` to `while (00000000) {}`. The more true bits you have the less chance of the value flipping to false. Sadly, 2 also only has one true bit. 3, however, would run for significantly longer. This also only applies to a compiler which doesn't always optimize this away (VC++). – Jonathan Dickinson Jul 21, 2014 at 12:56

8 @mr5 nope. In order for a bit flip to actually result in something like this you are talking about execution times in tens of thousands of years. Merely a **thought experiment.** If you hail from an immortal race you may want to use -1 to prevent bit-flipping from affecting your program. – Jonathan Dickinson Jul 21, 2014 at 13:47

64

The most likely explanation for the question is that the interviewer thinks that the processor checks the individual bits of the numbers, one by one, until it hits a non-zero value:

```
1 = 00000001
2 = 00000010
```

If the "is zero?" algorithm starts from the right side of the number and has to check each bit until it reaches a non-zero bit, the `while(1) { }` loop would have to check twice as many bits per iteration as the `while(2) { }` loop.

This requires a very wrong mental model of how computers work, but it does have its own internal logic. One way to check would be to ask if `while(-1) { }` or `while(3) { }` would be equally fast, or if `while(32) { }` would be *even slower*.

Share Improve this answer Follow

answered Jul 21, 2014 at 14:41
Ryan Cavanaugh
220k ● 59 ● 278 ● 236

42 I assumed the interviewer's misapprehension would be more like "2 is an int that needs to be converted to a boolean in order to be used in a conditional expression, whereas 1 is already boolean." – Russell Borogove Jul 21, 2014 at 15:30

7 And if the comparison algorithm starts from the left, it is the other way around. – Paŭlo Ebermann Jul 22, 2014 at 19:31

1 +1, that's exactly what I thought. you could perfectly imagine somebody believing that fundamentally, the `cmp` algorithm of a CPU is a linear bit check with early loop exit on the first difference. – v.oddou Jul 29, 2014 at 6:19

1 @PeterCordes "I've never heard anyone (other than you) argue that gcc or clang's `-O0` output is not literal enough." I never said such thing and I didn't have gcc or clang in mind at all. You must be misreading me. I'm just not of your opinion that what MSVC produces is hilarious. – blubberdiblub Aug 30, 2019 at 6:00

1 @PeterCordes I was wrong, in MSVC a breakpoint on the `while(1)` does not break before the loop, but on the expression. But it still collapses inside the loop when optimizing the expression away. Take this code with lots of breaks. In unoptimized debugging mode you get this. When optimizing, many breakpoints fall together or even spill over into the next function (the IDE shows the result breakpoints while debugging) - corresponding disassembly. – blubberdiblub Aug 31, 2019 at 7:42

35

Of course I do not know the real intentions of this manager, but I propose a completely different view: When hiring a new member into a team, it is useful to know how he reacts to conflict situations.

They drove you into conflict. If this is true, they are clever and the question was good. For some industries, like banking, posting your problem to Stack Overflow could be a reason for rejection.

But of course I do not know, I just propose one option.

Share
Improve this answer

edited Jul 22, 2014 at 19:23
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jul 22, 2014 at 3:17
Tõnu Samuel
2,896 ● 3 ● 22 ● 31

Follow

2 It is indeed excellent but while(2) vs while(1) is obviously taken from dilbert comics. It CANNOT be invented by somebody in his right mind (how does anybody come up with while(2) as a possible thing to write anyway ?). If your hypothesis was true, definitely you would give a problem so unique that you can google for it. Like "is while(0xf00b442) slower than while(1)", how the bank would find the inerviewee's question otherwise ? do you suppose they are the NSA and have access to keyscore ? – v.oddou Jul 29, 2014 at 6:22

28

I think the clue is to be found in "asked by a senior manager". This person obviously stopped programming when he became a manager and then it took him/her several years to become a senior manager. Never lost interest in programming, but never wrote a line since those days. So his reference is not "any decent compiler out there" as some answers mention, but "the compiler this person worked with 20-30 years ago".

At that time, programmers spent a considerable percentage of their time trying out various methods for making their code faster and more efficient as CPU time of 'the central minicomputer' was so valueable. As did people writing compilers. I'm guessing that the one-and-only compiler his company made available at that time optimized on the basis of 'frequently encountered statements that can be optimized' and took a bit of a shortcut when encountering a while(1) and evaluated everything else, including a while(2). Having had such an experience could explain his position and his confidence in it.

The best approach to get you hired is probably one that enables the senior manager to get carried away and lecture you 2-3 minutes on "the good old days of programming" before YOU **smoothly** lead him towards the next interview subject. (Good timing is important here - too fast and you're interrupting the story - too slow and you are labelled as somebody with insufficient focus). Do tell him at the end of the interview that you'd be highly interested to learn more about this topic.

Share Improve this answer Follow

answered Jul 22, 2014 at 22:30

OldFrank
878 ● 6 ● 7

22

You should have asked him how did he reached to that conclusion. Under any decent compiler out there, the two compile to the same asm instructions. So, he should have told you the compiler as well to start off. And even so, you would have to know the compiler and platform very well to even make a theoretical educated guess. And in the end, it doesn't really matter in practice, since there are other external factors like memory fragmentation or system load that will influence the loop more than this detail.

Share Improve this answer Follow

answered Jul 20, 2014 at 8:13

Rad'Val
9,183 ● 10 ● 66 ● 95

15 @GKFX If you've given your answer and they tell you that you are wrong there is no reason you can't ask them to explain why. If Anatolyg is correct and it is a test of your self confidence then you should be explaining why you answered the way you did and asking them the same. – P.Turpie Jul 22, 2014 at 4:14

I meant as the first thing you say to them. It can't be "Why is x faster?" "I don't know; why is x faster?". Obviously, having answered properly, you can then ask. – GKFX Jul 24, 2014 at 14:54

20

For the sake of this question, I should that add I remember Doug Gwyn from C Committee writing that some early C compilers without the optimizer pass would generate a test in assembly for the `while(1)` (comparing to `for(;;)` which wouldn't have it).

I would answer to the interviewer by giving this historical note and then say that even if I would be very surprised any compiler did this, a compiler could have:

- without optimizer pass the compiler generate a test for both `while(1)` and `while(2)`
- with optimizer pass the compiler is instructed to optimize (with an unconditional jump) all `while(1)` because they are considered as idiomatic. This would leave the `while(2)` with a test and therefore makes a performance difference between the two.

I would of course add to the interviewer that not considering `while(1)` and `while(2)` the same construct is a sign of low-quality optimization as these are equivalent constructs.

Share Improve this answer Follow

answered Jul 23, 2014 at 11:02

ouah
146k ● 15 ● 284 ● 338

11

Another take on such a question would be to see if you got courage to tell your manager that he/she is wrong! And how softly you can communicate it.

My first instinct would have been to generate assembly output to show the manager that any decent compiler should take care of it, and if it's not doing so, you will submit the next patch for it :)

Share

Improve this answer

Follow

edited Sep 7, 2014 at 22:51

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jul 23, 2014 at 1:10

UncleKing
743 ● 4 ● 10

9

To see so many people delve into this problem, shows exactly why this could very well be a test to see how quickly you want to **micro-optimize** things.

My answer would be; it doesn't matter that much, I rather focus on the business problem which we are solving. After all, that's what I'm going to be paid for.

Moreover, I would opt for `while(1) {}` because it is more common, and other teammates would not need to spend time to figure out why someone would go for a higher number than 1.

Now go write some code. ;-)

Share Improve this answer Follow

answered Jul 23, 2014 at 7:37

Bart Verkoeijen
17.6k ● 7 ● 55 ● 58

1 Unless you are being paid to optimize some real-time code for which you need to shave just 1 or 2 milliseconds to fit into its running time demands. Of course that's a job for the optimizer, some would say - that is if you have an optimizer for your architecture. – Neowizard Jul 24, 2014 at 9:38

7

If you're that worried about optimisation, you should use

```
for (;;)
```

because that has no tests. (cynic mode)

Share Improve this answer Follow

answered Mar 31, 2016 at 14:39

Tom Tanner
9,344 ● 3 ● 34 ● 61

6

It seems to me this is one of those behavioral interview questions masked as a technical question. Some companies do this - they will ask a technical question that should be fairly easy for any competent programmer to answer, but when the interviewee gives the correct answer, the interviewer will tell them they are wrong.

The company wants to see how you will react in this situation. Do you sit there quietly and don't push that your answer is correct, due to either self-doubt or fear of upsetting the interviewer? Or are you willing to challenge a person in authority who you know is wrong? They want to see if you are willing to stand up for your convictions, and if you can do it in a tactful and respectful manner.

Share Improve this answer Follow

answered Jul 27, 2014 at 17:32
pacoverflow
3,872 ● 12 ● 43 ● 77

5

Maybe the interviewer posed such dumb question intentionally and wanted you to make 3 points:

1. **Basic reasoning.** Both loops are infinite, it's hard to talk about performance.
2. **Knowledge about optimisation levels.** He wanted to hear from you if you let the compiler do any optimisation for you, it would optimise the condition, especially if the block was not empty.
3. **Knowledge about microprocessor architecture.** Most architectures have a special CPU instruction for comparision with 0 (while not necessarily faster).

Share
Improve this answer
Follow

edited Oct 16, 2014 at 6:44

answered Sep 18, 2014 at 8:54
Rok Kralj
48.7k ● 10 ● 74 ● 80

4

Here's a problem: If you actually write a program and measure its speed, the speed of both loops could be different! For some reasonable comparison:

```
unsigned long i = 0;
while (1) { if (++i == 1000000000) break; }

unsigned long i = 0;
while (2) { if (++i == 1000000000) break; }
```

with some code added that prints the time, some random effect like how the loop is positioned within one or two cache lines could make a difference. One loop might by pure chance be completely within one cache line, or at the start of a cache line, or it might to straddle two cache lines. And as a result, whatever the interviewer claims is fastest might actually be fastest - by coincidence.

Worst case scenario: An optimising compiler doesn't figure out what the loop does, but figures out that the values produced when the second loop is executed are the same ones as produced by the first one. And generate full code for the first loop, but not for the second.

Share Improve this answer Follow

answered Jul 21, 2014 at 0:34

gnasher729
52.5k ● 5 ● 79 ● 102

1 The "cache lines" reasoning will only work on a chip that has an extremely small cache. – sharptooth Jul 21, 2014 at 8:11

10 This is beside the point. Questions about speed assume "all else equal". – Jim Balter Jul 21, 2014 at 8:42

6 @JimBalter: This was not a question of speed, it was a question about an argument with an interviewer. And the possibility that doing an actual test might prove the interviewer "right" - for reasons that have nothing to do with his argument but are pure coincidence. Which would put you into an embarrassing situation if you tried to prove he was wrong. – gnasher729 Jul 21, 2014 at 11:03

1 @gnasher729 All logical arguments aside, the case of coincidence which you talk about would be worth looking at. But still blindly believing such a case exists is not only naive but stupid. As long as you don't explain what, how or why that would happen this answer is useless. – user568109 Jul 21, 2014 at 14:59

4 @gnasher729 "This was not a question of speed" -- I will not debate people who employ dishonest rhetorical techniques. – Jim Balter Jul 21, 2014 at 19:36

4

I used to program C and Assembly code back when this sort of nonsense might have made a difference. When it did make a difference we wrote it in Assembly.

If I were asked that question I would have repeated Donald Knuth's famous 1974 quote about premature optimization and walked if the interviewer didn't laugh and move on.

Share Improve this answer Follow

answered Jul 26, 2014 at 21:55

Peter Wooster
6,089 ● 2 ● 29 ● 39

1 Given he also said "In established engineering disciplines a 12% improvement, easily obtained, is never considered marginal and I believe the same viewpoint should prevail in software engineering", I think you walking is unjustified. – Alice Aug 10, 2014 at 3:20

3

They are both equal - the same.

According to the specifications anything that is not 0 is considered true, so even without any optimization, and a good compiler will not generate any code for while(1) or while(2). The compiler would generate a simple check for `!= 0`.

Share

edited Jul 22, 2014 at 19:36

answered Jul 21, 2014 at 12:49

Improve this answer

Follow

Peter Mortensen
31.6k ● 22 ● 109 ● 133

Nikolay Ivanchev
144 ● 6

@djechlin - because they both take 1 cpu cycle. – UncleKing Jul 23, 2014 at 1:04

The compiler constant folds it, so it's not even evaluated at run time. – PaulHK Aug 7, 2018 at 9:31

3

Judging by the amount of time and effort people have spent testing, proving, and answering this very straight forward question I'd say that both were made very slow by asking the question.

And so to spend even more time on it...

`while (2)` is ridiculous, because,

`while (1)`, and `while (true)` are historically used to make an infinite loop which expects `break` to be called at some stage inside the loop based upon a condition that will certainly occur.

The `1` is simply there to always evaluate to true and therefore, to say `while (2)` is about as silly as saying `while (1 + 1 == 2)` which will also evaluate to true.

And if you want to be completely silly just use: -

```
while (1 + 5 - 2 - (1 * 3) == 0.5 - 4 + ((9 * 2) / 4.0)) {
    if (succeed())
        break;
}
```

I think that the interviewer made a typo which did not effect the running of the code, but if he intentionally used the `2` just to be weird then sack him before he puts weird statements all through your code making it difficult to read and work with.

Share

Improve this answer

Follow

edited Aug 30, 2022 at 15:12

Random
505 ● 3 ● 17

answered Jul 25, 2014 at 4:43

ekerner
5,830 ● 1 ● 39 ● 31

The rollback you made reverts edit by a very high-rep user. These people usually know the policies very well and they're here to teach them to you. I find the last line of you post adding nothing useful to your post. Even if I got the joke I am obviously missing, I would consider it too chatty, not worth the extra paragraph. – Palec Jul 29, 2014 at 10:11

@Palec: Thanks for the comment. I found that the same guy edited many off my posts, mostly only to remove the sign off. So I figured he was just a reputation troll, and hence his reputation. Has online forums depleted the language - and common courtesies - so much that we no longer sign our writings? maybe I'm a little old school but it just seems rude to omit hellos and goodbyes. We are using apps, but we are still humans. – ekerner Jul 29, 2014 at 10:50

1 On Stack Exchange, salutations, taglines, thanks etc. are not welcome. The same is mentioned in help center, specifically under expected behavior. No part of Stack Exchange, not even Stack Overflow, is a forum – they are Q&A sites. Editing is one of the differences. Also comments should serve only to provide feedback on the post, not for discussion (Stack Overflow Chat). And there are many other ways Q&A differs from a forum. More on that in help center and on Meta Stack Overflow. – Palec Jul 29, 2014 at 11:27

Note that when you have over 2000 rep (edit privilege), you gain no more rep from edits. When in doubt why and edit you disagree with has been applied to your post, or you see someone's misbehavior, ask on Meta Stack Overflow. If the editor did the wrong thing, they could be notified by a mod (maybe they did not realize) or even get punished somehow (if the behavior was willfully malicious). Otherwise you get pointers to explanation why the edit/behavior is correct. Unnecessary rollback clutters revision history and can get you into rollback war. I roll back only when really sure the edit is incorrect. – Palec Jul 29, 2014 at 11:36

Finally about signing, greeting, …: It might seem rude to not include these formalities, but it increases signal to noise ratio and no information is lost. You can choose any nickname you want, that is your signature. In most places you have your avatar, too. – Palec Jul 29, 2014 at 11:52

2

That depends on the compiler.

If it optimizes the code, or if it evaluates 1 and 2 to true with the same number of instructions for a particular instruction set, the execution speed will be the same.

In real cases it will always be equally fast, but it would be possible to imagine a particular compiler and a particular system when this would be evaluated differently.

I mean: this is not really a language (C) related question.

Share Improve this answer Follow

answered Jul 23, 2014 at 6:51

user143522
49 ● 1

0

Since people looking to answer this question want the fastest loop, I would have answered that both are equally compiling into the same assembly code, as stated in the other answers. Nevertheless you can suggest to the interviewer using **'loop unrolling'; a do {} while loop** instead of the while loop.

Cautious: **You need to ensure that the loop would at least always run once**.

The loop should have a break condition inside.

Also for that kind of loop I would personally prefer the use of do {} while(42) since any integer, except 0, would do the job.

Share Improve this answer Follow

answered Jan 27, 2018 at 13:58

Antonin GAVREL
11.1k ● 12 ● 60 ● 90

Why would he suggest a do{}while loop? While(1) (or while(2)) does the same thing.
– Catsunami Aug 2, 2018 at 20:18

do while removes one extra jump so better performance. Of course in the case where you know that the loop would be executed at least once – Antonin GAVREL Aug 23, 2018 at 16:49

0

The obvious answer is: as posted, both fragments would run an equally busy infinite loop, which makes the program infinitely *slow*.

Although redefining C keywords as macros would technically have undefined behavior, it is the only way I can think of to make either code fragment fast at all: you can add this line above the 2 fragments:

```
#define while(x) sleep(x);
```

it will indeed make `while(1)` twice as fast (or half as slow) as `while(2)`.

Share
Improve this answer
Follow

edited Mar 15, 2019 at 7:06

answered Feb 21, 2019 at 19:29

chqrlie
144k ● 12 ● 130 ● 207

-4

The only reason I can think of why the `while(2)` would be any slower is:

1. The code optimizes the loop to

   `cmp eax, 2`

2. When the subtract occurs you're essentially subtracting

   a. `00000000 - 00000010 cmp eax, 2`

   instead of

   b. `00000000 - 00000001 cmp eax, 1`

`cmp` only sets flags and does not set a result. So on the least significant bits we know if we need to borrow or not with **b**. Whereas with **a** you have to perform two subtractions before you get a borrow.

Share

Improve this answer

Follow

edited Aug 5, 2014 at 17:12

answered Jul 22, 2014 at 20:46

hdost
873 ● 13 ● 23

15 cmp is going to take 1 cpu cycle regardless in both cases. – UncleKing Jul 23, 2014 at 1:12

8 That code would be incorrect. Correct code would load either 2 or 1 into register eax, then compare eax with 0. – gnasher729 Jul 25, 2014 at 0:02