`testl` eax against eax?

Asked 16 years, 2 months ago Modified 4 years, 10 months ago

Viewed 96k times

131

I am trying to understand some assembly.

The assembly as follows, I am interested in the `testl` line:

```
000319df  8b4508         movl   0x08(%ebp), %eax
000319e2  8b4004         movl   0x04(%eax), %eax
000319e5  85c0           testl  %eax, %eax
000319e7  7407           je     0x000319f0
```

I am trying to understand that point of `testl` between `%eax` and `%eax`? I think the specifics of what this code isn't important, I am just trying to understand the test with itself - wouldn't the value always be true?

assembly x86 instructions

Share

Improve this question

Follow

edited Apr 17, 2018 at 0:37

Michael Petch

47.5k ● 9 ● 116 ● 212

asked Sep 29, 2008 at 1:22

maxpenguin

5,149 ● 6 ● 30 ● 22

8 Answers

Sorted by: Highest score (default)

104

The meaning of `test` is to AND the arguments together, and check the result for zero. So this code tests if EAX is zero or not. `je` will jump if zero.

BTW, this generates a smaller instruction than `cmp eax, 0` which is the reason that compilers will generally do it this way.

Share Improve this answer Follow

edited Apr 28, 2017 at 3:57
phuclv
41.6k ● 15 ● 177 ● 519

answered Sep 29, 2008 at 1:24
Mike F

97

It tests whether `eax` is 0, or above, or below. In this case, the jump is taken if `eax` is 0.

Share Improve this answer Follow

edited Dec 9, 2016 at 1:08

answered Sep 29, 2008 at 1:23
C. K. Young
223k ● 47 ● 390 ● 443

3 I made an edit to turn this popular answer into a better canonical answer to "what's this TEST thing all about, and how is it different from CMP", which is sort of implied. See my own answer further down for comments about the semantic meaning of the synonymous JE and JZ. Please review my edit since it's pretty major, and it's still your answer. – Peter Cordes Dec 6, 2016 at 23:37

1 @PeterCordes I appreciate the intention, but I'm going to revert your edit. 1. Your "voice" is very different from mine, and right now it reads a lot more like your answer than mine. 2. More problematic is the bold assertion that the flags come out exactly the same way between `test` and `cmp`. Yes, I understand that's your belief based on your comments to Cody. However, putting it in my post is a different matter; it's not an assertion I'm willing to stand by, simply because *I don't know* if it's identical in all cases. – C. K. Young Dec 9, 2016 at 1:02

1 @PeterCordes If I find some spare time, I do want to flesh this answer out to be more canonical. I would write it like I write it, though, and I'm quite particular about how I write things. :-) For example, I'd write `je`, `jz`, `cmp`, and `test`, and not JE, JZ, CMP, or TEST. I'm picky like that. – C. K. Young Dec 9, 2016 at 1:12

1 I wasn't trying to boost my own answer. I actually forgot that I'd answered this question myself when I made that edit, and only noticed afterwards. I just looked at this after someone bumped it, and what started as a small edit snowballed into too much. No offence taken that you wanted to roll it back; it was just a suggestion and it definitely reads like my work, not yours. I'll take some of what I wrote and put it into my own answer. – Peter Cordes Dec 9, 2016 at 2:53

2 Wow, after editing my answer to this question to include what I added to yours, I realized I had almost exactly duplicated most of what I wrote in June. Oops! I updated it with more

reasoning to back up my claim that `test a,a` and `cmp $0,a` set flags identically; thanks for pointing out that that's a non-trivial claim. re: TEST vs. `test`: recently I've started using all-caps like Intel's manuals. But when I'm talking about AT&T mnemonics vs. Intel mnemonics, I use `testb` style for AT&T. IDK if that helps readability. – Peter Cordes Dec 9, 2016 at 3:21

35

The test instruction does a logical AND-operation between the operands but does not write the result back into a register. Only the flags are updated.

In your example the test eax, eax will set the zero flag if eax is zero, the sign-flag if the highest bit set and some other flags as well.

The Jump if Equal (je) instruction jumps if the zero flag is set.

You can translate the code to a more readable code like this:

```
cmp eax, 0
je  somewhere
```

That has the same functionality but requires some bytes more code-space. That's the reason why the compiler emitted a test instead of a compare.

Share Improve this answer Follow

answered Sep 29, 2008 at 1:29

Nils Pipenbrinck
86.2k ● 33 ● 155 ● 223

3 Actually, cmp might not work there. That is, it works for the specific case presented, but cmp affects flags differently than test does, due to it being an internal sub instead of and. Something to keep in mind. – Serafina Brocious Sep 29, 2008 at 1:34

4 for a test against zero it's perfectly valid. – Nils Pipenbrinck Sep 29, 2008 at 1:36

3 But you don't know what else looks at the flags later. The effects on flags are very different, so this can be an issue and very frequently is. – Serafina Brocious Sep 29, 2008 at 1:37

2 No, the only flags that are set by a different /method/ are carry and overflow, both of which are set to 0. The /values/ of the other flags will differ because cmp uses sub and test uses and. – Serafina Brocious Sep 29, 2008 at 2:14

2 @CodyBrocious: `test eax, eax` and `cmp eax, 0` both set all flags, and set them to identical values. Both instructions set all flags "according to the result". Subtracting `0` can never produce carry or overflow. Your argument is correct for any immediate other than 0, but not for 0.
– Peter Cordes Jun 25, 2016 at 20:50

31

`test` is like `and`, except it only writes FLAGS, leaving both its inputs unmodified. With two *different* inputs, it's useful for testing if some bits are all zero, or if at least one is set. (e.g. `test al, 3` sets ZF if EAX is a multiple of 4 (and thus has both of its low 2 bits zeroed).

`test eax,eax` sets all flags exactly the same way that `cmp eax, 0` would:

- CF and OF cleared (AND/TEST always does that; subtracting zero never produces a carry)
- ZF, SF and PF according to the value in EAX. (`a = a&a = a-0`).
  (PF as usual is only set according to the low 8 bits)

Except for the obsolete AF (auxiliary-carry flag, used by ASCII/BCD instructions). TEST leaves it undefined, but CMP sets it "according to the result". Since subtracting zero can't produce a carry from the 4th to 5th bit, CMP should always clear AF.

TEST is smaller (no immediate) and sometimes faster (can macro-fuse into a compare-and-branch uop on more CPUs in more cases than CMP). **That makes `test` the preferred idiom for comparing a register against zero**. It's a peephole optimization for `cmp reg,0` that you can use regardless of the semantic meaning.

The only common reason for using CMP with an immediate 0 is when you want to compare against a memory operand. For example, `cmpb $0, (%esi)` to check for a terminating zero byte at the end of an implicit-length C-style string.

AVX512F adds `kortestw k1, k2` and AVX512DQ/BW (Skylake-X but not KNL) add `ktestb/w/d/q k1, k2`, which operate on AVX512 mask registers (k0..k7) but still set regular FLAGS like `test` does, the same way that integer `OR` or `AND` instructions do. (Sort of like SSE4 `ptest` or SSE `ucomiss`: inputs in the SIMD domain and result in integer FLAGS.)

`kortestw k1,k1` is the idiomatic way to branch / cmovcc / setcc based on an AVX512 compare result, replacing SSE/AVX2 `(v)pmovmskb/ps/pd` + `test` or `cmp`.

Use of `jz` vs. `je` can be confusing.

`jz` and `je` are literally the same instruction, i.e. the same opcode in the machine code. **They do the same thing, but have different semantic meaning for humans**. Disassemblers (and typically asm output from compilers) will only ever use one, so the semantic distinction is lost.

`cmp` and `sub` set ZF when their two inputs are equal (i.e. the subtraction result is 0). `je` (jump if equal) is the semantically relevant synonym.

`test %eax,%eax` / `and %eax,%eax` again sets ZF when the result is zero, but there's no "equality" test. ZF after test doesn't tell you whether the two operands were equal. So `jz` (jump if zero) is the semantically relevant synonym.

Share Improve this answer Follow

edited Aug 1, 2019 at 23:47

answered Jun 25, 2016 at 20:45

Peter Cordes
361k ● 49 ● 699 ● 958

I would consider adding the basic information about `test` being bitwise `and` operation, may be not obvious for people just learning assembly (and being lazy/unaware to check instruction reference guide every 60 seconds ;) :)). – Ped7g Dec 16, 2017 at 9:37

1 @Ped7g: fair enough, I guess it doesn't hurt to put everything in this answer, instead of leaving that part to the other answers. Added AVX512 `kortest*` and `ktest*` while I was at it. – Peter Cordes Dec 16, 2017 at 13:24

BTW, this basically the same as [my answer to another version of the same question](), but I said more stuff about performance there, e.g. possibly avoiding register-read stalls on old P6-family CPUs like Nehalem by rewriting the register with the same value. – Peter Cordes Apr 8, 2018 at 3:38

@PeterCordes This should be the accepted answer : exhaustive and technical. Unlike the accepted post, this quenches one's curiosity and thirst for knowledge. Keep it up Sir. – programmersn Aug 31, 2018 at 10:12

It should be noted that PF is set to the parity of the low 8 bits, which in this case is AL. – ecm Aug 1, 2019 at 21:19

5

This snippet of code is from a subroutine that was given a pointer to something, probably some struct or object. The 2nd line dereferences that pointer, fetching a value from that thing - possibly itself a pointer or maybe just an int, stored as its 2nd member (offset +4). The 3rd and 4th lines test this value for zero (NULL if it's a pointer) and skip the following few operations (not shown) if it is zero.

The test for zero sometimes is coded as a compare to an immediate literal zero value, but the compiler (or human?) who wrote this might have thought a testl op would run faster - taking into consideration all the modern CPU stuff like pipelining and register renaming. It's from the same bag of tricks that holds the idea of clearing a register with XOR EAX,EAX (which i saw on someone's license plate in Colorado!) rather than the obvious but maybe slower MOV EAX, #0 (i use an older notation).

In asm, like perl, TMTOWTDI.

Share Improve this answer Follow

answered Sep 29, 2008 at 3:40

DarenW
16.9k ● 8 ● 69 ● 104

3

If eax is zero it will perform the conditional jump, otherwise it will continue execution at 319e9

Share Improve this answer Follow

answered Sep 29, 2008 at 1:25

Mike Thompson
6,738 ● 3 ● 33 ● 39

0

In some programs they can be used to check for a buffer overflow. At the very top of the allocated space a 0 is placed. After inputting data into the stack, it looks for the

0 at the very beginning of the allocated space to make sure the allocated space is not overflowed.

It was used in the stack0 exercise of exploits-exercises to check if it was overflowed and if there wasnt and there was a zero there, it would display "Try again"

```
0x080483f4 <main+0>:    push   ebp
0x080483f5 <main+1>:    mov    ebp,esp
0x080483f7 <main+3>:    and    esp,0xfffffff0
0x080483fa <main+6>:    sub    esp,0x60
0x080483fd <main+9>:    mov    DWORD PTR
[esp+0x5c],0x0 ;puts a zero on stack
0x08048405 <main+17>:   lea    eax,[esp+0x1c]
0x08048409 <main+21>:   mov    DWORD PTR [esp],eax
0x0804840c <main+24>:   call   0x804830c
<gets@plt>
0x08048411 <main+29>:   mov    eax,DWORD PTR
[esp+0x5c]
0x08048415 <main+33>:   test   eax,eax
; checks if its zero
0x08048417 <main+35>:   je     0x8048427 <main+51>
0x08048419 <main+37>:   mov    DWORD PTR
[esp],0x8048500
0x08048420 <main+44>:   call   0x804832c
<puts@plt>
0x08048425 <main+49>:   jmp    0x8048433 <main+63>
0x08048427 <main+51>:   mov    DWORD PTR
[esp],0x8048529
0x0804842e <main+58>:   call   0x804832c
<puts@plt>
0x08048433 <main+63>:   leave
0x08048434 <main+64>:   ret
```

Share Improve this answer Follow

answered Dec 6, 2016 at 21:19

user7259278

1

I don't see what this specific case of checking a register for non-zero adds to this Q&A. Especially when `cmp DWORD PTR [esp+0x5c], 0` / `jz 0x8048427 <main+51>` would have been more efficient than a separate MOV-load and then TEST. This is hardly a common use-case for checking for a zero. – Peter Cordes Dec 6, 2016 at 23:32

-4

we could see the **jg**，**jle** If `testl  %edx,%edx. jle  .L3` we could easy find **jle**is suit `(SF^OF)|ZF`,if %edx is zero ,ZF=1,but if %edx is not zero and is -1,after the testl ,the OF=0,and the SF =1,so the flag =true,that implement jump .sorry ,my English is poor

Share Improve this answer Follow

answered Mar 14, 2017 at 7:35

cbei_you
1