



64-bit mode does not support 32-bit PUSH and POP instructions [duplicate]

Asked 4 years, 3 months ago Active 6 months ago Viewed 17k times


15


This question already has answers here:

[Push and Pop on AMD64 \[duplicate\]](#) (1 answer)

[Does each PUSH instruction push a multiple of 8 bytes on x64?](#) (2 answers)

Closed 4 years ago.


7


NASM returns an error like: "instruction not supported in 64-bit mode"
(Or with YASM, invalid size for operand 1)

The subject instructions are `pop ecx` and `push ecx` .
What can I use instead of them or is there an other way to fix this issue?

`assembly` `nasm` `x86-64`

Share Improve this question

Follow


edited Jan 22 at 6:47
Peter Cordes
253k 36 438 634

asked Apr 16 '17 at 9:29


Konko
159 1 1 7

show your code, or the question will be invalid. Questions seeking debugging help ("why isn't this code working?") must include the desired behavior, a specific problem or error and the shortest code necessary to reproduce it in the question itself. Questions without a clear problem statement are not useful to other readers – [phuclv](#) Apr 16 '17 at 14:42

1 Answer

Active

Oldest

Votes

第1页 共3页

2021/8/6, 6:30 下午



28



The general idea is that you normally push and pop full registers, i.e. 64-bit registers in 64-bit mode. `push`'s default operand-size is 64-bit, and 32-bit operand-size is not available. [Does each PUSH instruction push a multiple of 8 bytes on x64?](#) (yes, unless you specifically use a 16-bit push, but 32-bit isn't available).

You cannot push a 32 bit register in 64 bit mode; instead, you can push and pop the whole 64 bit register that contains a 32-bit value you want, so that's `push rax` instead of `push eax`. The same holds for memory references - you can `push qword ptr[rax]`, but not `push dword ptr[rax]`.

But: even in 64 bit mode you can still push:

- 8 or 32 bit immediates sign extended to 64; this is generally handled automatically by your assembler as an optimization (if you do `push 1` it will encode it with the most compact encoding, which will be `6A01`, i.e. with an *imm8* operand). It's [always a 64-bit push unless you explicitly specify `push word 1`](#), regardless of what width of immediate the assembler picks.
- the `fs` and `gs` segment registers *but not* the `cs`, `ds`, `es`, `ss` registers (which aren't important in 64-bit mode, and can only be read with `mov`, not `push`, freeing up those push/pop opcode for potential future use).

As an exception, segment registers are either *zero*-extended or pushed on the stack with a 16-bit move (i.e. the other 48 bit on the stack are left unmodified); this isn't really much of a problem, since `pop fs` and `pop gs` just discard these extra bits.

You can emulate a `push imm64` with `push low32 / mov dword [rsp+4], high32`. Or with `mov r64, imm64 / push r64`; `mov` to register (not memory) is the only x86-64 instruction that can take a 64-bit immediate.

With 16-bit operand-size (a `66h` prefix), you can do a 16-bit push which adjusts RSP by 2 instead of 8. But normally don't do this because it will misalign the stack until you do a 16-bit pop or otherwise correct it.

- 16 bit registers (`push ax`) and memory references (`push word ptr[rax]`);
- 8-bit sign-extended or 16 bit immediates. `push word 123`

8-bit registers can't be pushed in any mode (except as part of a wider register), and 32-bit push/pop aren't available in 64-bit mode, [even with a `REX.W=0` prefix](#).

Share Improve this answer

Follow

edited Jan 22 at 6:48



Peter Cordes

253k 36 438 634

answered Apr 16 '17 at 9:35



Matteo Italia

116k 16 183 281

Besides the fact that you can push `r/m16` and `r/m64` but not `r/m32`, the fact that you can push `imm8`, `imm16` and `imm32` but not `imm64` is weird too. And you can't push `CS`, `DS`, `ES` or `SS`, but you can push `FS` and `GS`. Pretty inconsistent, IMO. – [Rudy Velthuis](#) Apr 16 '17 at 13:01

- 1 @RudyVelthuis: it *is* quite inconsistent, although there's a bit of method in this madness. Not being able to push an *imm64* is vaguely coherent with the rest of the instruction set – the only instruction that accepts an *imm64* that I know of is `mov r64, imm64` (actually, it's the only match for *imm64* in the whole instruction set description in Intel manuals). The segment registers besides *fs* and *gs* are no longer used in 64 bit mode, so it's unsurprising that they cannot be pushed. – [Matteo Italia](#) Apr 16 '17 at 14:26
-

I understand the "method behind the madness". But it gets quite confusing, at times. Especially the fact you can push/pop 16 bit but not 32 bit items. – [Rudy Velthuis](#) Apr 16 '17 at 16:46

- 3 @RudyVelthuis, The reason you can't push `eax` in x64 is that the encoding for `push r64` in x64 is the same as the encoding for `push r32` in x86. I.e. `push reg` means `push native_register_size`. Making it otherwise would require extending the instruction-set. and God knows x64 has more than enough instructions already. The 16/8 bit push already existed, so these are retained as legacy. – [Johan](#) Apr 16 '17 at 21:44
-

@Johan: I understood that, when looking at the instruction tables with the opcodes. But it gets more and more confusing, no matter the reason for it. – [Rudy Velthuis](#) Apr 16 '17 at 22:00
