

Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

Xeno Kovah – 2014-2015
xeno@legbacore.com

All materials is licensed under a Creative Commons “Share Alike” license.

- <http://creativecommons.org/licenses/by-sa/3.0/>

You are free:



to **Share** — to copy, distribute and transmit the work



to **Remix** — to adapt the work

Under the following conditions:



Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).




Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

Attribution condition: You must indicate that derivative work
"Is derived from Xeno Kovah's 'Intro x86-64' class, available at <http://OpenSecurityTraining.info/IntroX86-64.html>"

Attribution condition: You must indicate that derivative work

"Is derived from Xeno Kovah's 'Intro x86-64' class, available at <http://OpenSecurityTraining.info/IntroX86-64.html>"

MulDivExample.c

<pre>int main(){ unsigned int a = 1; a = a * 6; a = a / 3; return 0x2bad; }</pre>	<pre>main: 00000000140001010 sub rsp,18h 00000000140001014 mov dword ptr [rsp],1 0000000014000101B mov eax,dword ptr [rsp] 0000000014000101E imul eax,eax,6 00000000140001021 mov dword ptr [rsp],eax 00000000140001024 xor edx,edx 00000000140001026 mov eax,dword ptr [rsp] 00000000140001029 mov ecx,3  0000000014000102E div eax,ecx 00000000140001030 mov dword ptr [rsp],eax 00000000140001033 mov eax,2BADh 00000000140001038 add rsp,18h 0000000014000103C ret</pre>
---	---

We already saw that when a C operand is a power of 2, it uses shifts instead of multiplies/divides, but this shows that in other cases, it uses multiply or divide instructions.



DIV - Unsigned Divide

- Three forms
 - Unsigned divide ax by r/m8, al = quotient, ah = remainder
 - Unsigned divide edx:eax by r/m32, eax = quotient, edx = remainder
 - Unsigned divide rdx:rax by r/m64, rax = quotient, rdx = remainder
- If dividend is 32/64bits, edx/rdx will just be set to 0 by the compiler before the instruction (as occurred in the MulDivExample.c code)
- If the divisor is 0, a divide by zero exception is raised.

initial ↓ operation ↓ result	ax r/m8(cx)		edx eax r/mX(ecx)		
	0x8 0x3		0x0 0x8 0x5		
	div ax, cx		div eax, ecx		
	ah al		edx eax r/mX(ecx)		
	0x2 0x2		0x3 0x1 0x5		

Note that there's no form which takes an immediate.



IDIV - Signed Divide


- If you were to then change MulDivExample to signed, you would see the IDIV instruction appear
- Three forms
 - Signed divide ax by r/m8, al = quotient, ah = remainder
 - Signed divide edx:eax by r/mX, eax = quotient, edx = remainder
 - Signed divide rdx:rax by r/m64, rax = quotient, rdx = remainder
- If dividend is 32/64bits, edx/rdx will just be set to 0 by the compiler before the instruction
- If the divisor is 0, a divide by zero exception is raised.

initial ↓ operation ↓ result	ax r/m8(cx)		edx eax r/mX(ecx)		
	0xFE	0x2	0x0	0x8	0x3
	div ax, cx		div eax, ecx		
	ah al		edx eax r/mX(ecx)		
	0x0	0xFF	0x1	0x2	0x3

Note that there's no form which takes an immediate.

MulDivExample.c takeaways

- When a multiply or divide is not by a power of 2, compilers will use normal multiply/divide instructions
- VS compiler prefers IMUL over MUL (unsigned multiply) for simple multiplies, due to its option to use 3 parameters

int main(){	main:		
unsigned int a = 1;	0000000140001010	sub	rsp,18h
a = a * 6;	0000000140001014	mov	dword ptr [rsp],1
a = a / 3;	000000014000101B	mov	eax,dword ptr [rsp]
return 0x2bad;	000000014000101E	imul	eax,ecx,6
}	0000000140001021	mov	dword ptr [rsp],eax
	0000000140001024	xor	edx,edx
	0000000140001026	mov	eax,dword ptr [rsp]
	0000000140001029	mov	ecx,3
	 000000014000102E	div	eax,ecx
	0000000140001030	mov	dword ptr [rsp],eax
	0000000140001033	mov	eax,2BADh
	0000000140001038	add	rsp,18h
	000000014000103C	ret	

We already saw that when a C operand is a power of 2, it uses shifts instead of multiplies/divides, but this shows that in other cases, it uses multiply or divide instructions.

Instructions we now know (28)

- NOP
- PUSH/POP
- CALL/RET
- MOV
- ADD/SUB
- IMUL
- MOVZX/MOVSX
- LEA
- JMP/Jcc (family)
- CMP/TEST
- AND/OR/XOR/NOT
- INC/DEC
- SHR/SHL/SAR/SAL
- DIV/IDIV