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

Xeno Kovah – 2014
xkovah at gmail
All materials is licensed under a Creative Commons “Share Alike” license.

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

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"
```c
int main(){
    unsigned int a = 1;
    a = a * 6;
    a = a / 3;
    return 0x2bad;
}
```

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 32/64 bits, 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.

<table>
<thead>
<tr>
<th>initial</th>
<th>ax</th>
<th>r/m8(cx)</th>
<th>edx</th>
<th>eax</th>
<th>r/mX(ecx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8</td>
<td>0x3</td>
<td></td>
<td>0x0</td>
<td>0x8</td>
<td>0x3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>operation</th>
<th>div ax, cx</th>
<th>div eax, ecx</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>ah</td>
<td>al</td>
</tr>
<tr>
<td></td>
<td>0x2</td>
<td>0x2</td>
</tr>
</tbody>
</table>

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:
  - Unsigned divide `ax` by `r/m8`, `al = quotient, ah = remainder`
  - Unsigned divide `edx:eax` by `r/mX`, `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.
- If the divisor is 0, a divide by zero exception is raised.

<table>
<thead>
<tr>
<th>initial operation</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>div <code>ax, cx</code></td>
<td>ah al</td>
</tr>
<tr>
<td>div <code>eax, ecx</code></td>
<td>edx eax</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>ax</code></th>
<th><code>r/m8(cx)</code></th>
<th><code>edx</code></th>
<th><code>eax</code></th>
<th><code>r/mX(ecx)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFE</td>
<td>0x2</td>
<td>0x0</td>
<td>0x8</td>
<td>0x3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><code>ah</code></th>
<th><code>al</code></th>
<th><code>edx</code></th>
<th><code>eax</code></th>
<th><code>r/mX(ecx)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0xFF</td>
<td>0x1</td>
<td>0x2</td>
<td>0x3</td>
</tr>
</tbody>
</table>

Note that there's no form which takes an immediate.
int main()
{
    unsigned int a = 1;
    a = a * 6;
    a = a / 3;
    return 0x2bad;
}

main:
0000000140001010  sub         rsp,18h
0000000140001014  mov         dword ptr [rsp],1
000000014000101B  mov         eax,dword ptr [rsp]
000000014000101E  imul        eax,eax,6
0000000140001021  mov         dword ptr [rsp],eax
0000000140001024  xor         edx,edx
0000000140001026  mov         eax,dword ptr [rsp]
000000014000102E  div         eax,ecx
0000000140001030  mov         dword ptr [rsp],eax
0000000140001033  mov         eax,2BADh
0000000140001038  add         rsp,18h
000000014000103C  ret

• 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

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/MOVXSX
- LEA
- JMP/Jcc (family)
- CMP/TEST
- AND/OR/XOR/NOT
- INC/DEC
- SHR/SHL/SAR/SAL
- DIV/IDIV