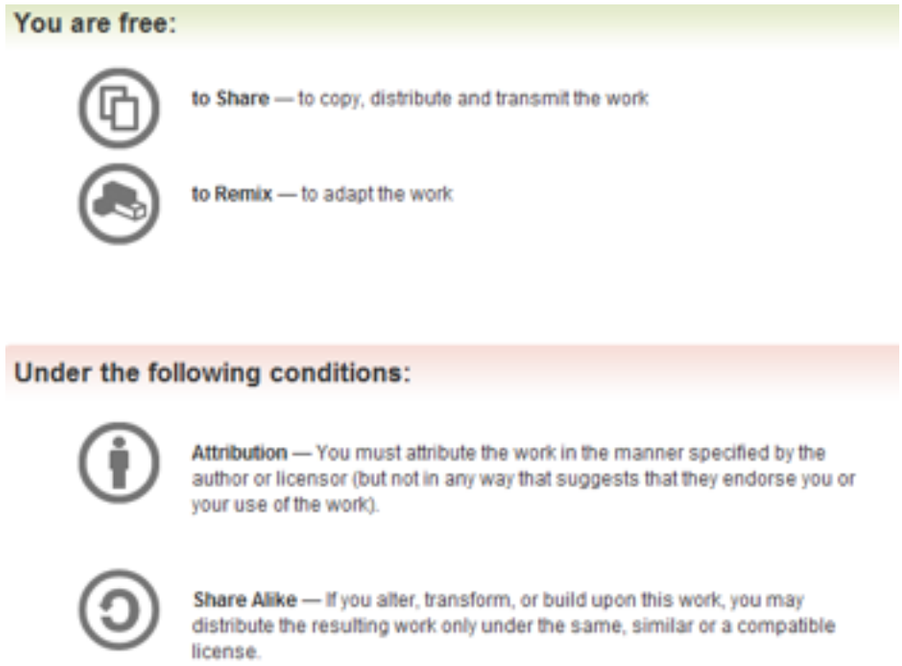
Introduction to Intel x86-64 Assembly, Architecture, Applications, & Alliteration

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"

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"

Refresher - Boolean (“bitwise”) logic

AND “&”

0	0	0
0	1	0
1	0	0
1	1	1

Operands **Result**

OR “|”

0	0	0
0	1	1
1	0	1
1	1	1

XOR “^”

0	0	0
0	1	1
1	0	1
1	1	0

NOT “~”

0	1
1	0

AND - Logical AND

- Destination operand can be r/mX or register
- Source operand can be r/mX or register or immediate (No source *and* destination as r/mXs)

and al, bl

	00110011b (al - 0x33)
AND	01010101b (bl - 0x55)
result	00010001b (al - 0x11)

and al, 0x42

	00110011b (al - 0x33)
AND	01000010b (imm - 0x42)
result	00000010b (al - 0x02)

Book p. 231

OR - Logical Inclusive OR

- Destination operand can be r/mX or register
- Source operand can be r/mX or register or immediate (No source *and* destination as r/mXs)

or al, bl

	00110011b (al - 0x33)
OR	01010101b (bl - 0x55)
result	01110111b (al - 0x77)

or al, 0x42

	00110011b (al - 0x33)
OR	01000010b (imm - 0x42)
result	01110011b (al - 0x73)

Book p. 231

XOR - Logical Exclusive OR

- Destination operand can be r/mX or register
- Source operand can be r/mX or register or immediate (No source *and* destination as r/mXs)

xor al, al

	00110011b (al - 0x33)
XOR	00110011b (al - 0x33)
result	00000000b (al - 0x00)

XOR is commonly used to zero a register, by XORing it with itself, because it's faster than a MOV

xor al, 0x42

	00110011b (al - 0x33)
OR	01000010b (imm - 0x42)
result	01110001b (al - 0x71)

Book p. 231

NOT - One's Complement Negation

- Single source/destination operand can be r/mX

not al

NOT	00110011b (al - 0x33)
result	11001100b (al - 0xCC)

Xeno trying to be clever on a boring example, and failing…

not [al+bl]

al	0x10000000
bl	0x00001234
al+bl	0x10001234
[al+bl]	0 (assumed memory at 0x10001234)
NOT	00000000b
result	11111111b

Book p. 231

ForLoop.c - simple for loop

```
#include <stdio.h>

int main(){
   int i;
   for(i = 0; i < 10; i++){
      printf("i = %d\n", i);
   }
}
```

```
main:
0000000140001010  sub     rsp,38h
0000000140001014  mov     dword ptr [rsp+20h],0
000000014000101C  jmp     0000000140001028
000000014000101E  mov     eax,dword ptr [rsp+20h]
0000000140001022  inc     eax
0000000140001024  mov     dword ptr [rsp+20h],eax
0000000140001028  cmp     dword ptr [rsp+20h],0Ah
000000014000102D  jge     0000000140001042
000000014000102F  mov     edx,dword ptr [rsp+20h]
0000000140001033  lea     rcx,[40006000h]
000000014000103A  call    qword ptr [40008368h]
0000000140001040  jmp     000000014000101E
0000000140001042  xor     eax,eax
0000000140001044  add     rsp,38h
0000000140001048  ret
```

INC/DEC - Increment / decrement

- Single source/destination operand can be r/mX
- Increase or decrease the value by 1
- When optimized, compilers will tend to favor *not* using inc/dec, as directed by the Intel optimization guide. So their presence may be indicative of hand-written, or un-optimized code

rax	0x7e117a1e

xor rax, rax

rax	0x0

inc rax

rax	0x1

rax	0xdec0ded

mov rax, 0

rax	0x0

dec rax

rax	0xFFFFFFFF`FFFFFFFF

Book p. 215-216

ForLoop.c - takeaways

- For loops will be some combination of a JCC (to determine if the exit condition is met yet), and an absolute JMP (to return from the end to the conditional checking code)
- In the absence of an explicit return value VS & GCC default to returning 0

```
#include <stdio.h>

int main(){
   int i;
   for(i = 0; i < 10; i++){
      printf("i = %d\n", i);
   }
}
```

```
main:
0000000140001010  sub       rsp,38h
0000000140001014  mov       dword ptr [rsp+20h],0
000000014000101C  jmp       0000000140001028
000000014000101E  mov       eax,dword ptr [rsp+20h]
0000000140001022  inc       eax
0000000140001024  mov       dword ptr [rsp+20h],eax
0000000140001028  cmp       dword ptr [rsp+20h],0Ah
000000014000102D  jge       0000000140001042
000000014000102F  mov       edx,dword ptr [rsp+20h]
0000000140001033  lea       rcx,[40006000h]
000000014000103A  call      qword ptr [40008368h]
0000000140001040  jmp       000000014000101E
0000000140001042  xor       eax,eax
0000000140001044  add       rsp,38h
0000000140001048  ret
```

Instructions we now know (22)

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