IR Translator
y86 (and x86) to VEX-IR

Zach Sisco

CEG 7420

Spring 2017

Problem

- Want to analyze binary code (y86 and x86)
- Need higher-level representation that makes side-effects explicit and information tracing easier across different architectures

Project: Reverse Engineering tool suite

- Target architectures: y86 and x86
- Target IR: VEX-IR

- Already lift x86 to VEX
- How to approach lifting y86 to VEX?

y86 to VEX?

1. Integrate y86 into libvex
2. Binary translation from y86 to x86

Binary translation

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MNEMONIC              Y86-OP   -> X86-OP
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

halt                  00       -> f4
nop                   10       -> 90

rrmovl r, s           20rs     -> 89 bin(11 r s)

cmovle r, s           21rs     -> 0f 4e bin(11 r s)
cmovl  r, s           22rs     -> 0f 4c bin(11 r s)
cmove  r, s           23rs     -> 0f 44 bin(11 r s)
cmovne r, s           24rs     -> 0f 45 bin(11 r s)
cmovge r, s           25rs     -> 0f 4d bin(11 r s)
cmovg  r, s           26rs     -> 0f 4f bin(11 r s)

irmovl ABCD, r        30XrDCBA -> b8+r DCBA

rmmovl r, 0xABCD(s)   40rsDCBA -> 89 bin(10 r s) DCBA

mrmovl 0xABCD(s), r   50rsDCBA -> 8b bin(10 r s) DCBA
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MNEMONIC              Y86-OP   -> X86-OP
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

mrmovl 0xABCD(s), r   50rsDCBA -> 8b bin(10 r s) DCBA

addl r, s             60rs     -> 01 bin(11 r s)
subl r, s             61rs     -> 29 bin(11 r s)
andl r, s             62rs     -> 21 bin(11 r s)
xorl r, s             63rs     -> 31 bin(11 r s)

jmp ABCD              70DCBA   -> FAR  -> ff 25 EFGH
                                  NEAR -> e9 EFGH
jle ABCD              71DCBA   -> 0f 8e EFGH=[signed offset from this_address+6 to ABCD]
jl  ABCD              72DCBA   -> 0f 8c EFGH=[signed offset from this_address+6 to ABCD]
je  ABCD              73DCBA   -> 0f 84 EFGH=[signed offset from this_address+6 to ABCD]
jne ABCD              74DCBA   -> 0f 85 EFGH=[signed offset from this_address+6 to ABCD]
jge ABCD              75DCBA   -> 0f 8d EFGH=[signed offset from this_address+6 to ABCD]
jg  ABCD              76DCBA   -> 0f 8f EFGH=[signed offset from this_address+6 to ABCD]

call ABCD             80DCBA   -> e8 EFGH=[signed 32-bit offset from this_address+5 to ABDC]
ret                   90       -> c3
pushl r               A0rX     -> 50+r
popl  r               B0rX     -> 58+r

int 0x80              CD80     -> CD80
```

Example

```
0x00: jmp 0x06: 70 06 00 00 00          //Jump past halt.                          00: e9 01 00 00 00
0x05: hlt 00                                                                     05: f4

0x06: irmovl eax 0x02: 30 80 02 00 00 00 //Put 2 in EAX                          06: b8 02 00 00 00
0x0C: irmovl ebx 0x03: 30 83 03 00 00 00 //Put 3 in EBX                         0b: bb 03 00 00 00
0x12: subl eax, ebx: 61 03               //EBX = EBX-EAX. EBX = 1.               10: 29 c3
0x14: jge 0x1A: 75 1A 00 00 00           //Jump to 1A since EBX-EAX >= 0         12: 0f 8d 01 00 00 00
0x19: hlt 00                                                                     18: f4

0x1A: subl eax, ebx: 61 03               //EBX = EBX-EAX. EBX = -1.              19: 29 c3
0x1C: jl 0x05: 72 05 00 00 00            //Jump to 0x05 since EBX-EAX = -1.      1b: 0f 8c e4 ff ff ff
0x21:                                                                            21:
```

y2x86translate

```
Command line arguments:
-b <filename>
      y86 binary file to translate/lift [REQUIRED]
-w <filename>
      write translated binary to <filename>
-i
      lift to VEX-IR
```

```
┌─[y2x86translate]$
└─> python y2x86translate.py -b test-binaries/jumptest.yo -i
IRSB {
   t0:Ity_I32 t1:Ity_I32 t2:Ity_I32 t3:Ity_I32 t4:Ity_I64 t5:Ity_I64 t6:Ity_I64 t7:Ity_I32 t8:Ity_I32 t9:Ity_I32 t10:Ity_I32 t11:Ity_I32 t12:Ity_I8 t13:
Ity_I8 t14:Ity_I8 t15:Ity_I32 t16:Ity_I8 t17:Ity_I8 t18:Ity_I8 t19:Ity_I32 t20:Ity_I8 t21:Ity_I8 t22:Ity_I8 t23:Ity_I32 t24:Ity_I32 t25:Ity_I16 t26:Ity_
I32 t27:Ity_I64 t28:Ity_I1 t29:Ity_I32 t30:Ity_I32 t31:Ity_I32 t32:Ity_I32 t33:Ity_I32 t34:Ity_I32 t35:Ity_I32 t36:Ity_I32

   00 | ------ IMark(0x0, 3, 0) ------
   01 | t25 = GET:I16(gs)
   02 | t24 = 16Uto32(t25)
   03 | t4 = GET:I64(ldt)
   04 | t5 = GET:I64(gdt)
   05 | t26 = GET:I32(eax)
   06 | t27 = x86g_use_seg_selector(t4,t5,t24,t26):Ity_I64
   07 | t29 = 64HIto32(t27)
   08 | t28 = CmpNE32(t29,0x00000000)
   09 | if (t28) { PUT(eip) = 0x0; Ijk_MapFail }
   10 | t1 = GET:I32(esi)
   11 | PUT(ip) = 0x00000003
   12 | ------ IMark(0x3, 2, 0) ------
   13 | t10 = LDle:I32(t26)
   14 | t8 = Xor32(t10,t1)
   15 | STle(t26) = t8
   16 | PUT(ip) = 0x00000005
   17 | ------ IMark(0x5, 2, 0) ------
   18 | t14 = LDle:I8(t26)
   19 | t13 = GET:I8(17)
   20 | t12 = Xor8(t14,t13)
   21 | STle(t26) = t12
   22 | PUT(ip) = 0x00000007
   23 | ------ IMark(0x7, 2, 0) ------
   24 | t18 = LDle:I8(t26)
   25 | t16 = Xor8(t18,t13)
   26 | STle(t26) = t16
   27 | PUT(ip) = 0x00000009
   28 | ------ IMark(0x9, 3, 0) ------
   29 | t33 = Add32(t1,0x00000034)
   30 | t22 = LDle:I8(t33)
   31 | t21 = GET:I8(9)
   32 | t20 = Xor8(t22,t21)
   33 | STle(t33) = t20
   34 | PUT(cc_op) = 0x0000000d
   35 | t35 = 8Uto32(t20)
   36 | PUT(cc_dep1) = t35
   37 | PUT(cc_dep2) = 0x00000000
   38 | PUT(cc_ndep) = 0x00000000
   39 | ------ IMark(0xc, 0, 0) ------
   NEXT: PUT(eip) = 0x0000000c; Ijk_NoDecode
}
```

```
┌─[y2x86translate]$
└─> python y2x86translate.py -b test-binaries/jumptest.yo
e901000000f4b802000000bB0300000029C30f8d01000000f429C30f8cE4FFFFFF
┌─[y2x86translate]$
└─>
```

y2x86translate

- Leverages *PyVEX*
  - Lifts and parses VEX for use in Python
  - https://github.com/angr/pyvex

- To Do:
  - Validation