

Proposal

Hanzhou Tang Jiutian Yu
hanzhout@smu.edu jiutiany@smu.com

February 2019

Abstract

We want to build a assembly simulator which should supports basic x86 assembly instructions.

1 Why Java

It's tempted to implement a assembly simulator in low level language like C or C++. However, after some consideration, we decide choose Java. There are several reason.

1.1 Java is easy for memory management

With C++11, it has smarter pointer to make life easier [1]. However, it's implemented in library level instead of language supporting. Sometimes, when mistakenly mixed raw pointer and smart pointer, smart pointer may become useless.

1.2 Java is easy to test

There are lots of powerful java testing framework to do unit test. For example, JUnit or Groovy Spock. Meanwhile, because Java support proxy object, it's much easy to mock obejcts and record function invoking.

1.3 Java is easy for package manage

With the help of gradle [3], it's very easy for us to import different libraries and do deployment. We believe it could save us lots of time and effort.

1.4 Java is easy to integrate with REST API

If we could finish our project well, we may want to implement an online version for all users. We could easily provide REST API with the help of Spring [4].

2 Our registers

We do want to implement some basic functionality of x86 platform. So we decide to imitate 10 32 bits registers. Here is a table of our registers. The table originally comes from [2]. As you can see, we ignore all segment registers. That's because according to our design, only one process can run at one time and no context switch. We think in this case segment information is unnecessary. Maybe we're wrong, we may change our decision later.

3 Our instruction set

We want to support a subset of assembly instructions on x86 platform. By doing some research [2], we provide a table which contains all instructions we want support.

References

- [1] JOSUTTIS, N. M. *The C++ standard library: a tutorial and reference*. Addison-Wesley, 2012.
- [2] KUSSWURM, D. *Modern X86 Assembly Language Programming: 32-bit, 64-bit, SSE, and AVX*. Apress, 2014.
- [3] MUSCHKO, B. *Gradle in action*. Manning, 2014.
- [4] WALLS, C., AND BREIDENBACH, R. *Spring in action*. Dreamtech Press, 2005.

Register	Descriptions
EAX	Accumulator. 0 to 7 can be referred as AL. 8 to 15 can be referred as AH. 0 to 15 can be referred as AX.
EBX	Memory pointer, base Register. 0 to 7 can be referred as BL. 8 to 15 can be referred as BH. 0 to 15 can be referred as BX.
ECX	Loop control. 0 to 7 can be referred as CL. 8 to 15 can be referred as CH. 0 to 15 can be referred as CX.
EDX	Integer multiplication, integer division. 0 to 7 can be referred as DL. 8 to 15 can be referred as DH. 0 to 15 can be referred as DX.
ESI	String instruction source pointer.
EDI	String instruction destination pointer.
ESP	Stack Pointer.
EBP	Stack frame base Pointer.
EIP	Instruction pointer register.
EFLAGS	Flag register.

Table 1: The registers we want to imitate

mov	Copy data from one place to another place.
push	Push register, memory location or immediate value onto stack.
pop	Pop the first item from stack.
add	Add two number
sub	Subtraction
cbw	Sign-extends register AL.
cwd	Sign-extends register AX.
bswap	Reverse the bytes of a 32-bit register.
and	Logic and.
or	Logic or.
xor	Logic xor.
not	Logic not.
sal/shl	Left shift.
sar	Arithmetic right shift.
shr	Logic right shift.
cmpsb/cmpsw/cmpps	Compare the values at location.
lodsbl/lodsw/lodsd	Loads the values at location.
stosb/stosw/stosd	Save the values at register to memory.
rep/repne	Repeat a specified instruction by condition
jmp/jcc/jecxz	Unconditonal/conditional jump
call	Push content to stack then do unconditional jump.
ret	Pop stack then do unconditional jump.
enter	Create a stack frame for function.
leave	Remove a stack frame of function.
loop/loope/loopz/loopne/loopnz	Loop.
nop	Advance the instruction pointer.

Table 2: The instructions we want to imitate