

Информатика

Архитектура x86. Ассемблер NASM. Основные команды. Отладка

Гирик Алексей Валерьевич

Университет ИТМО
2022

Материалы курса

- Презентации, материалы к лекциям, литература, задания на лабораторные работы
 - shorturl.at/jqRZ6



Система команд архитектуры x86-64

Типы команд

- все команды можно поделить на несколько групп:
 - основные
 - сопроцессора
 - расширений
 - MMX
 - XMM
 - SSE, SSE2, SSE3, ...
 - ...

Функциональная классификация команд

- основные команды процессора можно поделить на несколько групп:
 - пересылки данных
 - общего назначения
 - работы со стеком
 - арифметические
 - двоичной арифметики
 - десятичной арифметики
 - логические
 - передачи управления
 - цепочечные
 - управления состоянием ЦП

Команды пересылки данных

■ `mov <DST>, <SRC>`

- `mov rax, rbx` ; из регистра в регистр
- `mov rax, 0xFF` ; непосредственный операнд
- `mov rax, [0xFF]` ; из памяти в регистр
- `mov rax, [rax]` ; косвенная адресация
- `mov [rax], [rbx]` ; так не получится

■ `xchg <OP1>, <OP2>`

- `xchg rbx, rax` ; обменять два регистра
- `xchg rax, [rbx]` ; обменять регистр и память
- `xchg [rbx], [rax]` ; это тоже не работает

Операнды-адреса

В общем случае, допустимы следующие варианты задания адреса в командах, где одним из операндов может быть адрес памяти:

<code>[offset]</code>	<code>; смещение задано константой</code>
<code>[reg]</code>	<code>; смещение задано в регистре</code>
<code>[offset + reg]</code>	<code>; сумма смещения и регистра</code>
<code>[offset + reg*scale]</code>	<code>; scale = 1, 2, 4 или 8</code>
<code>[offset + reg + reg*scale]</code>	

Примеры:

```
mov    rax, [0xffee]
mov    rax, [array + rbx]
```

В чем проблема?

```
...  
mov [rax], 100  
...
```



Маловато будет!

```
$ nasm -felf64 test.S  
exp.S:16: error: operation size not specified
```


При работе с памятью необходимо задавать размер операнда

```
...  
mov dword [rax], 100  
...
```

либо можно задать размер непосредственного
операнда:

```
mov [rax], dword 100
```



***Вот это мой
размерчик!***

Арифметические команды

■ add <OP1>, <OP2>

□ add rax, 3 ; сложение

■ sub <OP1>, <OP2>

□ sub rax, rbx ; вычитание

■ inc <OP>

□ inc rax ; инкремент

■ dec <OP>

□ dec qword [rax] ; декремент

■ neg <OP>

□ neg rax ; изменение знака

■ cmp <OP1>, <OP2>

□ cmp rax, rbx ; сравнение (вычитание без сохранения)

Арифметические команды

■ `mul <OP>`

□ `mul rdx` ; умножение беззнаковое

■ `imul <OP>`

□ `imul qword [rdx]` ; умножение знаковое

■ `div <OP>`

□ `div rbx` ; деление беззнаковое

■ `idiv <OP>`

□ `idiv rbx` ; деление знаковое

Логические команды

■ not <OP>

□ not rax

; поразрядное НЕ

■ or <OP1>, <OP2>

□ or rax, rbx

; поразрядное ИЛИ

■ and <OP1>, <OP2>

□ and rax, [rbx]

; поразрядное И

■ xor <OP1>, <OP2>

□ xor rax, rax

; поразрядное исключающее ИЛИ

■ test <OP1>, <OP2>

□ test rax, rbx

; поразрядное И без сохранения

; результата (но с установкой

; флагов)

Операции сдвига

■ `shl <OP>, N`

□ `shl rax, 10` ; сдвиг влево

■ `shr <OP>, N`

□ `shr dword [rax], 10` ; сдвиг вправо

■ `sar <OP>, N`

□ `sar rax, 1` ; арифметический сдвиг вправо

■ `rol <OP>, N`

□ `rol rax, 2` ; циклический сдвиг влево

■ `ror <OP>, N`

□ `ror rax, cl` ; циклический сдвиг вправо

Вместо N можно использовать регистр `cl` (и только этот регистр!)

Команды передачи управления

■ `jmp <ADDR>`

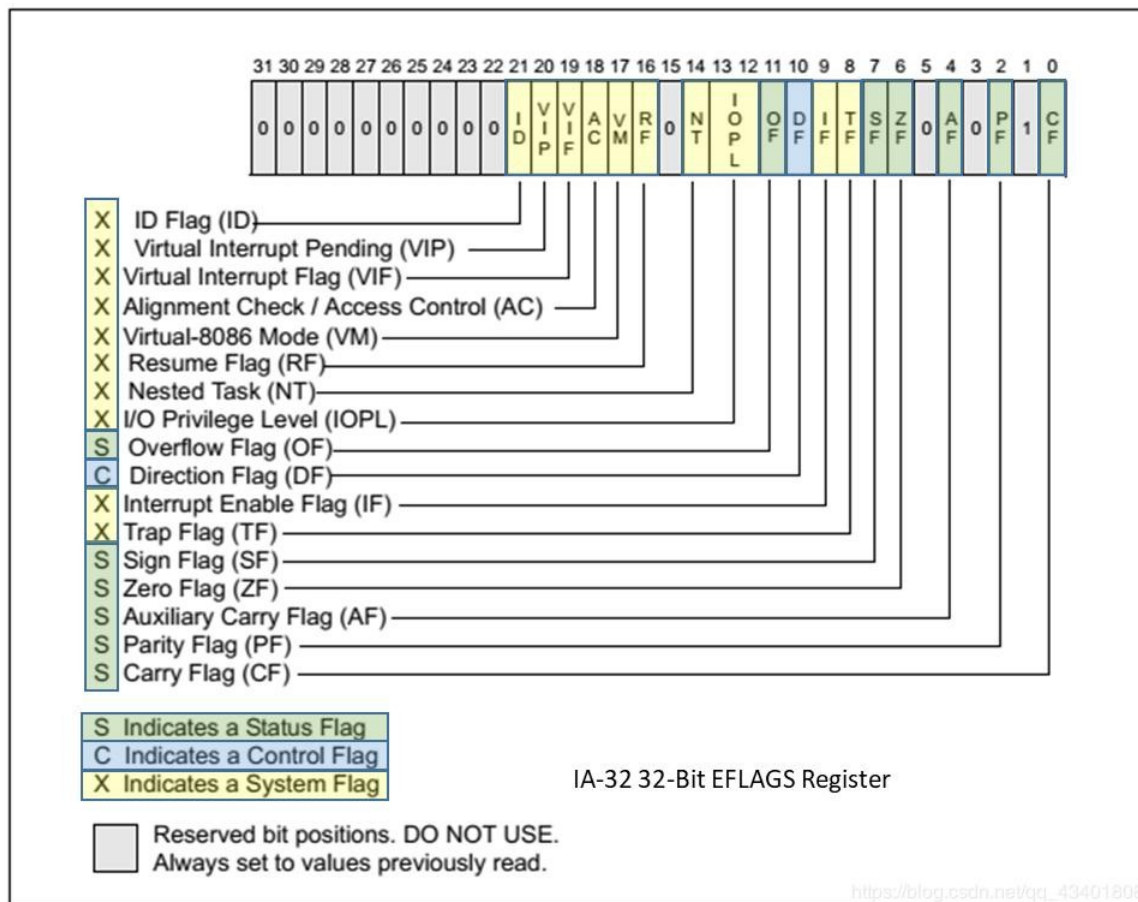
- `jmp loop_beg` ; безусловный переход
- `jmp [rax]` ; переход на адрес, содержащийся
; в регистре `rax`

```
loop_beg:           ; начало цикла
...
...
jmp loop_beg        ; переход в начало цикла
```

Флаги в регистре rflags

Флаги:

- CF
- ZF
- SF
- OF



Команды передачи управления

■ j??? <ADDR>

	; условный переход, если...	
□ jz, js, jc, jo	; установлен флаг	
□ jnz, jns, jnc, jno	; НЕ установлен флаг	
□ je	; операнды равны	
□ jne	; операнды НЕ равны	
□ jl, jnge	; меньше	} для чисел со знаком
□ jle, jng	; меньше или равно	
□ jg, jnle	; больше	
□ jge, jnl	; больше или равно	} для чисел без знака
□ jb, jnae	; меньше	
□ jbe, jna	; меньше или равно	
□ ja, jnbe	; больше	
□ jae, jnb	; больше или равно	

Справка по системе команд x86/x86-64

- официальная документация Intel
 - <https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>
- неофициальная документация
 - <https://www.felixcloutier.com/x86/>
- ...

Пример: вывод звездочек

- Задача: написать программу на ассемблере, которая выводит на консоль фиксированное количество звездочек

```
$ ./stars
```

```
*
```

```
**
```

```
***
```

```
****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```



Как получить случайное число?

- можно использовать пару библиотечных функций `srand()` и `rand()`

```
srand(time(NULL));  
int n = rand() % 16;
```

- на x86-64 можно использовать инструкцию `rdrand`

```
rdrand    rax  
and       rax, 0xf  
mov       [n], rax
```

Цепочечные (строковые) инструкции

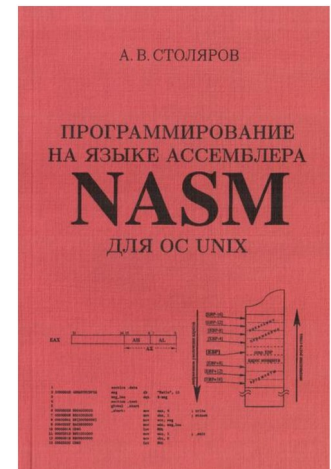
- используют регистры `rsi`, `rdi`, `rcx`, `rax` и флаг `DF`

- `stosb/stosw/stosd/stosq`
- `lodsб/lodsw/lodsd/lodsq`
- `movsb/movsw/movsd/movsq`
- `cmpsб/cmpsw/cmьsd/cmьsq`
- `scasб/scasw/scasd/scasq`

- могут использоваться либо в цикле, либо с префиксами повторения

- направление работы определяется флагом `DF`

- `std` ; установка `DF` = адреса уменьшаются
- `cld` ; сброс `DF` = адреса увеличиваются



Подробнее – §2.7

Применение цепочечных инструкций

- заполнить массив звездочками

```
mov    al, '*'           ; байт для записи
mov    rdi, buf          ; куда записываем, buf – адрес строки
mov    rcx, 8            ; сколько раз повторять
cld                               ; увеличивать адрес
rep stosb                 ; заполнить строку
```

- цепочечные инструкции работают быстро! В этом основной смысл их использования

Отладка программ

Что такое отладчик

- Отладчик (debugger) – программа, которая позволяет выполнять другую программу в пошаговом режиме и наблюдать за состоянием памяти отлаживаемой программы
- Отладчик может быть интегрирован в среду разработки (Visual Studio, XCode) или поставляться отдельно (gdb, lldb)
- Отладку можно выполнять на уровне исходных кодов или машинных инструкций

Что можно узнать с помощью отладчика

- Значения переменных во время выполнения
- Стек вызовов функций
- Значения аргументов функций
- Состояние памяти
- Состояние регистров процессора

- Если происходит исключение, то
 - тип исключения
 - место возникновения

Отладчик gdb

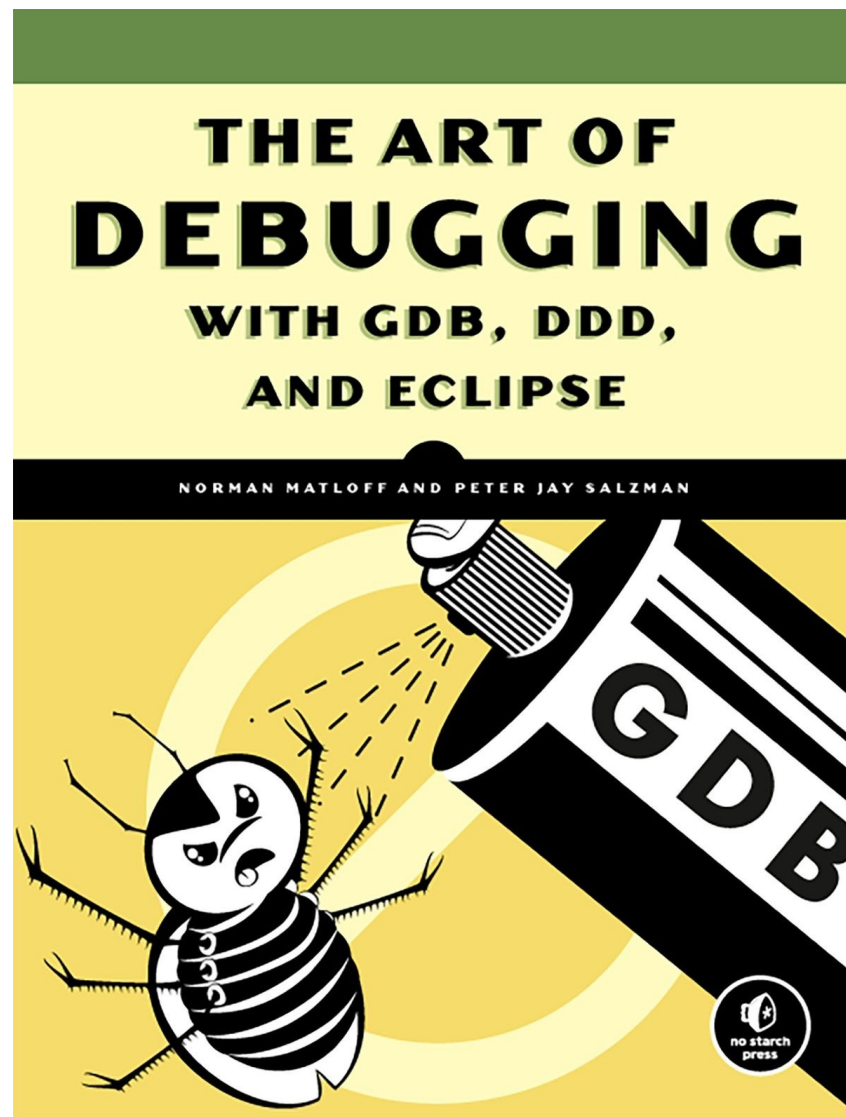
- Основные команды:
 - ❑ `help <cmd>` – показать справку по `cmd`
 - ❑ `break <func>` – точка останова в `func`
 - ❑ `run` – запустить программу
 - ❑ `next`, `step` – выполнить следующую строку исходного кода
 - ❑ `ni`, `si` – выполнить следующую инструкцию
 - ❑ `list` – показать исходный код
 - ❑ `disas /s` – показать код+инструкции
 - ❑ `frame` – показать текущее положение
 - ❑ `kill` – остановить выполнение программы
 - ❑ `quit` – выход

Отладчик gdb

- Информация о текущем состоянии:
 - ❑ `info reg` – состояние регистров
 - ❑ `info local` – локальные переменные
 - ❑ `info arg` – значения аргументов функции
 - ❑ `p <переменная>` – вывести значение переменной
 - ❑ `x/<формат> <объект>` – вывести значение объекта (переменной, адреса,...) в заданном формате
 - ❑ `bt` – показать состояние стека вызовов

Книжка по gdb

- *N. Matloff, P. Salzman*
The Art of Debugging with
gdb, ddd and Eclipse



gdb dashboard

<https://github.com/cyrus-and/gdb-dashboard>

```
GDB dashboard

Output/messages
17 for (i = 0; i < text_length; i++) {

Assembly
0x0000555555551ec 48 8b 45 f8 encrypt+103 mov rax,QWORD PTR [rbp-0x8]
0x0000555555551f0 48 01 d0 encrypt+107 add rax,rdx
0x0000555555551f3 31 ce encrypt+110 xor esi,ecx
0x0000555555551f5 89 f2 encrypt+112 mov edx,esi
0x0000555555551f7 88 10 encrypt+114 mov BYTE PTR [rax],dl
0x0000555555551f9 48 83 45 f8 01 encrypt+116 add QWORD PTR [rbp-0x8],0x1
0x0000555555551fe 48 8b 45 f8 encrypt+121 mov rax,QWORD PTR [rbp-0x8]
0x000055555555202 48 3b 45 e8 encrypt+125 cmp rax,QWORD PTR [rbp-0x18]
0x000055555555206 72 bb encrypt+129 jb 0x555555551c3 <encrypt+62>
0x000055555555208 90 encrypt+131 nop

Breakpoints
[1] break at 0x0000555555552d9 in xor.c:56 for xor.c:56 hit 1 time
[2] break at 0x000055555555199 in xor.c:13 for encrypt hit 1 time
[3] break at 0x000055555555521b in xor.c:27 for dump if i = 5
[4] write watch for output[10] hit 1 time

Expressions
password[1 % password_length] = 101 'e'
text[i] = 32 ' '
output[i] = 69 'E'

History
$$1 = 0x55555559260 "\f\032\v\006\022\004\032\001\037E": 12 '\f'
$$0 = 0x7fffffffef2c "hunter2": 104 'h'

Memory
password
0x00007fffffffef2c 68 75 6e 74 65 72 32 00 64 6f 65 73 6e 74 20 6c hunter2-doesnt-1
text
0x00007fffffffef34 64 6f 65 73 6e 74 20 6c 6f 6f 6b 20 6c 69 6b 65 doesn't look like
0x00007fffffffef44 20 73 74 61 72 73 20 74 6f 20 6d 65 00 48 4f 53 -stars-to-me-HOS

output
0x0000555555559260 0c 1a 0b 07 0b 06 12 04 1a 01 1f 45 00 00 00 .....E....
0x0000555555559270 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

Registers
rax 0x000055555555926b rbx 0x0000000000000000 rcx 0x0000000000000065
rdx 0x0000000000000045 rsi 0x0000000000000045 rdi 0x00007fffffffef40
rbp 0x00007fffffffefc40 rsp 0x00007fffffffefc00 r8 0x0000000000000003
r9 0x000000000000a3330 r10 0x0000555555559010 r11 0x0000000000000030
r12 0x000000555555550a0 r13 0x00007fffffffef60 r14 0x0000000000000000
r15 0x0000000000000000 rip 0x0000555555551f9 eflags [ IF ]
cs 0x000000033 ss 0x0000002b ds 0x00000000
es 0x00000000 fs 0x00000000 gs 0x00000000

Source
12 /* obtain the lengths */
13 password_length = strlen(password);
14 text_length = strlen(text);
15
16 /* perform the encryption */
17 for (i = 0; i < text_length; i++) {
18     output[i] = text[i] ^ password[i % password_length];
19 }
20 }
21

Stack
[0] from 0x0000555555551f9 in encrypt+116 at xor.c:17
[1] from 0x0000555555552f0 in main+139 at xor.c:56

Threads
[1] id 8 name xor from 0x0000555555551f9 in encrypt+116 at xor.c:17

Variables
arg password = 0x7fffffffef2c "hunter2": 104 'h'
arg text = 0x7fffffffef34 "doesn't look like stars to me": 100 'd'
arg output = 0x55555559260 "\f\032\v\006\022\004\032\001\037E": 12 '\f'
loc password_length = 7
loc text_length = 28
loc i = 11

>>> |
```

Задание к следующей лекции

- Столяров. Программирование на языке ассемблера NASM для ОС UNIX
 - гл. 2 §2.6 –§2.9
 - гл. 3

