

# Операционные Системы

## Язык ассемблера

January 5, 2019

# Язык ассемблера для x86

- ▶ Язык ассемблера концептуально прост:
  - ▶ минимум синтаксических правил;
  - ▶ много различных инструкций (зависит от архитектуры).
- ▶ Есть много диалектов:
  - ▶ будем использовать GNU, а. к. а. AT&T.

# Классы инструкций

- ▶ Инструкции копирования:
  - ▶ из памяти в регистр и назад;
  - ▶ из регистра в регистр;
  - ▶ реже из памяти в память.
- ▶ Арифметические инструкции.
- ▶ Инструкции перехода:
  - ▶ условного перехода и безусловного.
- ▶ Прочие инструкции.

# Регистры

- ▶ Регистры - очень быстрые именованные ячейки памяти.
- ▶ Регистры специального назначения
  - ▶ указатель команд;
  - ▶ флаговый регистр;
  - ▶ и много много других.
- ▶ Регистры общего назначения.

# Регистры x86

- ▶ Указатель команд - *RIP*.
- ▶ Флаговый регистр - *RFLAGS*.
- ▶ Регистры общего назначения:
  - ▶ указатель стека - *RSP*;
  - ▶ указатель "базы" - *RBP*;
  - ▶ *RAX, RBX, RCX, RDX, RSI, RDI, R8 - R15*.

# Инструкция копирования MOV

- ▶ `movq <src>, <dst>`
  - ▶ `movq %RAX, %RBX`
  - ▶ `movq (%RAX), %RAX`
  - ▶ `movq $42, %RAX`
  - ▶ `movq 42, %RAX`
  - ▶ `movq $value, %RAX`
  - ▶ `movq value, %RAX`

# Простые арифметические инструкции

- ▶ `addq <src>, <dst>`
  - ▶ `addq %RAX, %RBX`
  - ▶ `addq %RAX, value`
  - ▶ `addq $42, %RAX`
- ▶ `sub <src>, <dst>`
- ▶ `incq <op>`
  - ▶ `incq %RAX`
- ▶ `decq <op>`

# Инструкции беззнакового умножения и деления

- ▶ `mulq <op>`:

- ▶  $RAX = (<op> \times RAX) \bmod 2^{64}$

- ▶  $RDX = (<op> \times RAX) / 2^{64}$

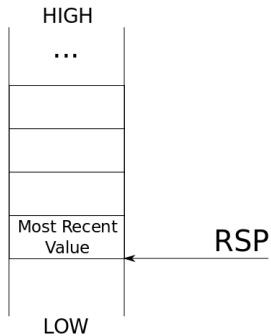
- ▶ `divq <op>`:

- ▶  $RDX = (RDX \times 2^{64} + RAX) \bmod <op>$

- ▶  $RAX = (RDX \times 2^{64} + RAX) / <op>$



# Стек

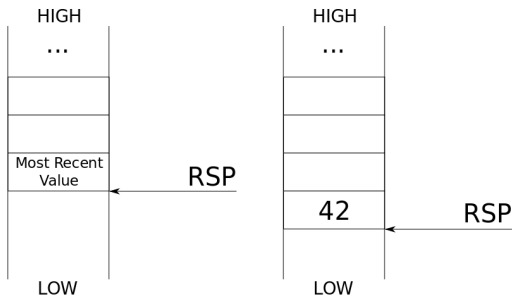


# Инструкции работы со стеком

- ▶ `pushq <src>` - уменьшает *RSP* на 8 и сохраняет по полученному адресу *src*
  - ▶ `pushq $42`
  - ▶ `pushq %RAX`

# Инструкции работы со стеком

PUSHQ \$42

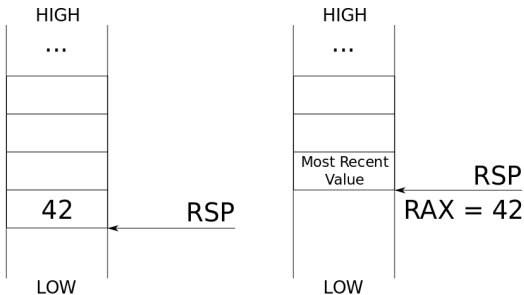


# Инструкции работы со стеком

- ▶ `popq <dst>` - обратное действие к *pushq*
  - ▶ `popq %RAX`
- ▶ `movq (%RSP), %RAX`

# Инструкции работы со стеком

POPQ %RAX



# Метки и переменные

- ▶ Метка - просто имя для некоторого адреса:

```
1      .data
2 value:
3      .quad 42
4
5      .text
6 add42:
7      movq %rdi, %rax
8      addq value, %rax
9      retq
```

# Инструкции безусловного перехода

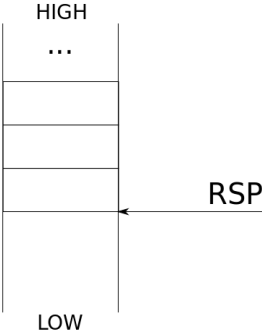
- ▶ Инструкции безусловного перехода изменяют значение регистра *RIP*:
  - ▶ `jmp <label>`
  - ▶ `call <label>` - инструкция вызова функции
  - ▶ `retq` - инструкция возврата из функции

# Функции

- ▶ Функция:
  - ▶ функцию можно вызвать;
  - ▶ функция возвращает управление *вызвавшему коду*.

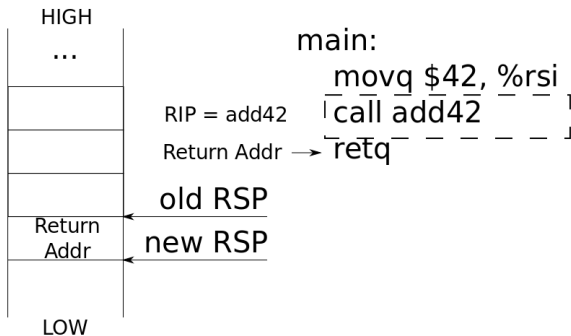


## Вызов функции



```
main:
    movq $42, %rsi
    call add42
    retq
```

# Вызов функции



# Флаговый регистр RFLAGS

- ▶ Флаговый регистр хранит флаги!
- ▶ Флаги регистра RFLAGS:
  - ▶ *ZF* - результат операции 0;
  - ▶ *CF* - произошло беззнаковое переполнение;
  - ▶ *OF* - произошло знаковое переполнение.

# Инструкции условного перехода

- ▶ `jcc <label>` - выполняет переход, если условие `cc` истинно.
  - ▶ `jz, je` - проверяет, что  $ZF = 1$ ;
  - ▶ `jne, jnz` -  $ZF = 0$ ;
  - ▶ `jg` - если "больше" (знаковый вариант);
  - ▶ `jge` - "больше или равно" (знаковый вариант);
  - ▶ `ja` - если "больше" (беззнаковый вариант);
  - ▶ `jae` - "больше или равно" (беззнаковый вариант).

# Инструкции сравнения

- ▶ Арифметические инструкции изменяют *RFLAGS*, но также изменяют свои аргументы!
- ▶ Есть команды, которые выставляют флаги, но не изменяют свои аргументы:
  - ▶ `cmpq <src>, <dst>` - вычисляет разность  $dst - src$  и выставляет флаги;
  - ▶ т. е. `cmpq` работает как `subq`, но не изменяет *dst*.

# Пример ветвления

```
1 max:
2     movq %rdi, %rax
3     cmpq %rsi, %rdi # rdi - rsi
4     ja  rdi_gt      # rdi - rsi > 0
5     movq %rsi, %rax
6 rdi_gt:
7     ret
```

# Соглашения

- ▶ ABI (Application Binary Interface) - набор соглашений
  - ▶ как в функцию передаются параметры;
  - ▶ как функция возвращает значения;
  - ▶ какие регистры функция должна сохранить, а какие может испортить;
  - ▶ и многое другое.

# Различные ABI

- ▶ Разные компиляторы используют различные ABI:
  - ▶ например, Microsoft используют свой собственный ABI;
  - ▶ Unix-like системы, зачастую, используют System V ABI.
- ▶ Мы будем использовать System V ABI
  - ▶ скачайте ABI и найдите, как в функцию передаются параметры.