Архитектура компьютера

Введение в Ассемблер

Александр Рудалёв

ИМИКТ САФУ

г. Архангельск, 2016 г.

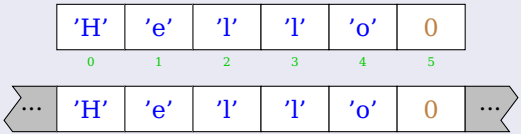
Основные элементы

Основные элементы

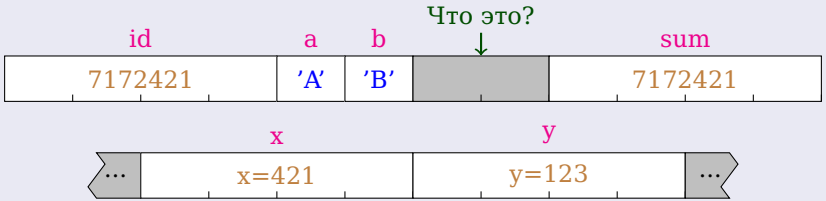
Байт

Int32

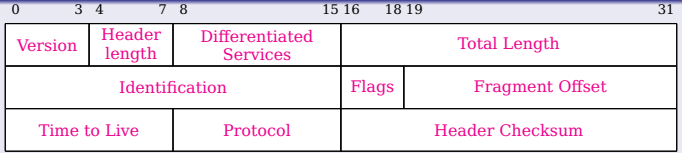
Массив



Структура



Сложная структура



Листинг

```
 1                                  global  _main
 2                                  extern  _ExitProcess@4
 3                                  extern  _printf
 4
 5                                  section .text   use32
 6                                  _main:
 7 00000000 A1[1D000000]                    mov     eax, [A]
 8 00000005 50                              push    eax
 9 00000006 8B1D[21000000]                  mov     ebx, [B]
10 0000000C 53                              push    ebx
11 0000000D 01D8                            add     eax, ebx
12 0000000F 50                              push    eax
13 00000010 68[00000000]                    push    msg
14 00000015 E8(00000000)                    call    _printf
15 0000001A B800000000                      mov     eax, 0
16 0000001F 31C0                            xor     eax, eax
17 00000021 50                              push    eax
18 00000022 E8(00000000)                    call    _ExitProcess@4
19
20                                  section .data
21 00000000 41203D2025640D0A        msg     db "A = %d", 13, 10,
22 00000008 42203D2025640D0A                db "B = %d", 13, 10,
23 00000010 41202B2042203D2025-             db "A + B = %d", 13, 10, 0
24 00000019 640D0A00
25 0000001D D4C3B2A1                A       dd 0xA1B2C3D4
26 00000021 AE080000                B       dd 2222
```

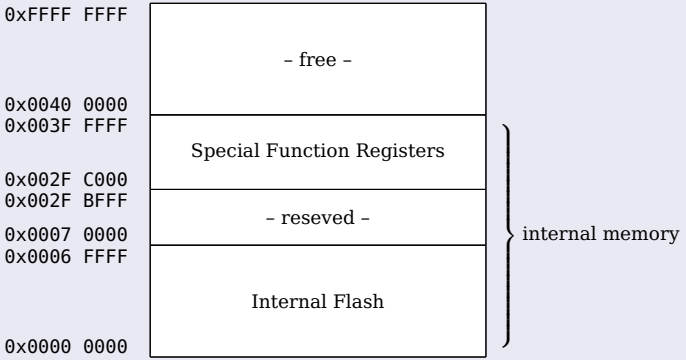
Листинг

```
 1                                  global  _main
 2                                  extern  _ExitProcess@4
 3                                  extern  _printf
 4
 5                                  section .text   use32
 6                                  _main:
 7 00000000 A1[1D000000]                    mov     eax, [A]
 8 00000005 50                              push    eax
 9 00000006 8B1D[21000000]                  mov     ebx, [B]
10 0000000C 53                              push    ebx
11 0000000D 01D8                            add     eax, ebx
12 0000000F 50                              push    eax
13 00000010 68[00000000]                    push    msg
14 00000015 E8(00000000)                    call    _printf
15 0000001A B800000000                      mov     eax, 0
16 0000001F 31C0                            xor     eax, eax
17 00000021 50                              push    eax
18 00000022 E8(00000000)                    call    _ExitProcess@4
19
20                                  section .data
21 00000000 41203D2025640D0A        msg     db "A = %d", 13, 10,
22 00000008 42203D2025640D0A                db "B = %d", 13, 10,
23 00000010 41202B2042203D2025-             db "A + B = %d", 13, 10, 0
24 00000019 640D0A00
25 0000001D D4C3B2A1                A       dd 0xA1B2C3D4
26 00000021 AE080000                B       dd 2222
```

Дамп памяти

```
00000000  4c 01 02 00 62 bc ee 54  e2 00 00 00 0e 00 00 00  |L...b..T........|
00000010  00 00 00 00 2e 74 65 78  74 00 00 00 00 00 00 00  |.....text.......|
00000020  00 00 00 00 27 00 00 00  64 00 00 00 8b 00 00 00  |....'...d.......|
00000030  00 00 00 00 05 00 00 00  20 00 50 60 2e 64 61 74  |........ .P‘.dat|
00000040  61 00 00 00 00 00 00 00  00 00 00 00 25 00 00 00  |a...........%...|
00000050  bd 00 00 00 e2 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  40 00 30 c0 a1 1d 00 00  00 50 8b 1d 21 00 00 00  |@.0......P..!...|
00000070  53 01 d8 50 68 00 00 00  00 e8 00 00 00 00 b8 00  |S..Ph...........|
00000080  00 00 00 31 c0 50 e8 00  00 00 00 01 00 00 00 04  |...1.P..........|
00000090  00 00 00 06 00 08 00 00  00 04 00 00 00 06 00 11  |................|
000000a0  00 00 00 04 00 00 00 06  00 16 00 00 00 08 00 00  |................|
000000b0  00 14 00 23 00 00 00 07  00 00 00 14 00 41 20 3d  |...#.........A =|
000000c0  20 25 64 0d 0a 42 20 3d  20 25 64 0d 0a 41 20 2b  | %d..B = %d..A +|
000000d0  20 42 20 3d 20 25 64 0d  0a 00 d4 c3 b2 a1 ae 08  | B = %d.........|
000000e0  00 00 2e 66 69 6c 65 00  00 00 00 00 00 00 fe ff  |...file.........|
000000f0  00 00 67 01 6c 61 62 30  33 2e 61 73 6d 00 00 00  |..g.lab03.asm...|
00000100  00 00 00 00 00 00 2e 74  65 78 74 00 00 00 00 00  |.......text.....|
00000110  00 00 01 00 00 00 03 01  27 00 00 00 05 00 00 00  |........’.......|
00000120  00 00 00 00 00 00 00 00  00 00 2e 64 61 74 61 00  |...........data.|
00000130  00 00 00 00 00 00 02 00  00 00 03 01 25 00 00 00  |............%...|
00000140  00 00 00 00 00 00 00 00  00 00 00 00 00 00 2e 61  |...............a|
00000150  62 73 6f 6c 75 74 00 00  00 00 ff ff 00 00 03 00  |bsolut..........|
00000160  00 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  02 00 5f 70 72 69 6e 74  66 00 00 00 00 00 00 00  |.._printf.......|
00000180  00 00 02 00 5f 6d 61 69  6e 00 00 00 00 00 00 00  |...._main.......|
00000190  01 00 00 00 02 00 6d 73  67 00 00 00 00 00 00 00  |......msg.......|
000001a0  00 00 02 00 00 00 03 00  41 00 00 00 00 00 00 00  |........A.......|
000001b0  1d 00 00 00 02 00 00 00  03 00 42 00 00 00 00 00  |..........B.....|
000001c0  00 00 21 00 00 00 02 00  00 00 03 00 40 66 65 61  |..!.........@fea|
000001d0  74 2e 30 30 01 00 00 00  ff ff 00 00 03 00 13 00  |t.00............|
000001e0  00 00 5f 45 78 69 74 50  72 6f 63 65 73 73 40 34  |.._ExitProcess@4|
```

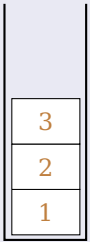
Память (в документации)



Стек

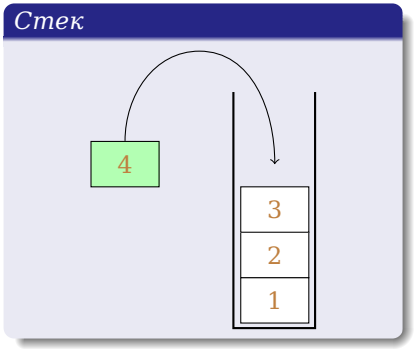
Структура данных «стек»

Стек



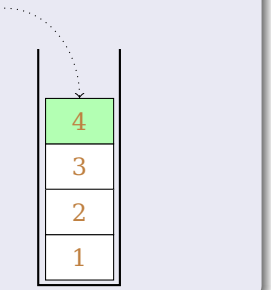
Стек (англ. stack — стопка; читается стэк) — структура данных, представляющая собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Структура данных «стек»



«Положить» элемент в стек можно только на «вершину».

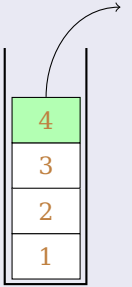
Стек



«Положить» элемент в стек можно только на «вершину».

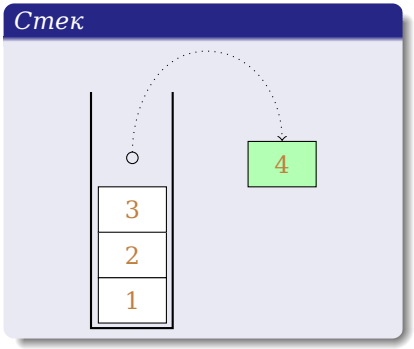
Структура данных «стек»

Стек



«Взять» из стека можно только верхний элемент.

Структура данных «стек»

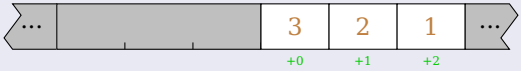


«Взять» из стека можно только верхний элемент.

Стек

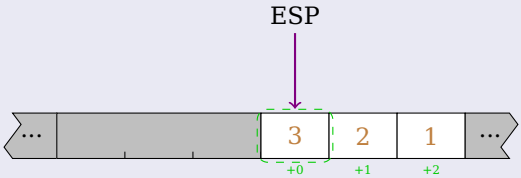
- Теперь рассмотрим как стек храниться в памяти.
- Повернём его на бок.
- Тогда элементы стека будут расположены как в массиве

Стек



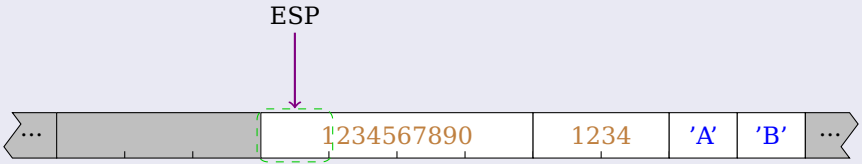
- И представление стека в памяти может быть таким.

Стек



- Добавим ссылку на вершину (позже узнаем почему именно ESP).

Стек



- В стеке могут храниться не только однобайтовые элементы.
- Но вершина стека — это первый хранящейся байт в нём.

Стек и процессор

- Стек — аппаратно реализованная структура в процессоре.
- Он хранится в специальном «сегменте» оперативной памяти.
- На вершину стека смотрит «регистр процессора» ESP.
- Для работы со стеком доступны команды **push** и **pop**.
- Стек используется не только для хранения данных, но и для вызова подпрограмм.

Регистры процессора

Регистры процессора

- Регистр процессора — блок ячеек памяти, образующий сверхбыструю оперативную память внутри процессора
- Размер регистра, как правило зависит от разрядности процессора (или его режима).
- Регистры имеют имена: EAX, AX, ESP, EIP и т.д.
- Регистры разбиваются на функциональные группы, и не все они могут быть доступны программисту.
- В рамках курса мы рассмотрим только несколько видов регистров, доступных при программировании прикладного ПО.

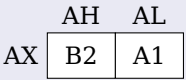
Регистр AX

Регистр

- AX — 16-битный регистр общего назначения.

Регистр AX

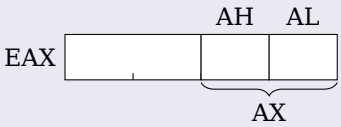
Регистр



- Для доступа к младшему байту регистра AX используется имя AL.
- Для доступа к старшему байту регистра AX используется имя AH.
- В приведённом примере:
  - AX=0xB2A2,
  - AH=0xB2,
  - AL=0xA1
- Внимание!!! Регистры AL, AH, AX используют те же самые два байта в памяти!!!

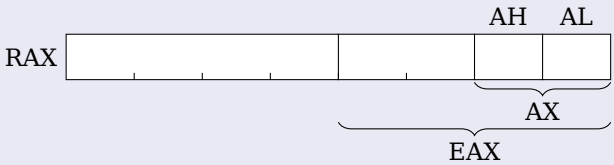
Регистр AX

Регистр



- В 32-битном режиме регистр AX расширяется до регистра EAX.
- Доступ к частям регистра сохраняется.
- Например, для EAX=0xD4C3B2A1 получим, что:
  - AX=0xB2A2,
  - AH=0xB2,
  - AL=0xA1

Регистр



- В 64-битном режиме регистр AX расширяется до регистра RAX.
- Все те же свойства, как и в 32-битном режиме, для регистра сохраняются.

Регистр

EAX | 1234567890 |

- Для сокращения размера изображения, будем обозначать регистр простым блоком, т.к. его размер известен.

Список регистров

Общего назначения

- RAX,EAX,AX,AH,AL (Аккумулятор)
- RBX,EBX,BX,BH,BL
- RCX,ECX,CX,CH,CL (Счётчик)
- RDX,EDX,DX,DH,DL
- R8-R15

Сегментные

- CS (Code Segment)
- DS (Data Segment)
- SS (Stack Segment)
- ES (Extra Segment)
- FS
- GS

Ссылки

- RIP,EIP,IP (Следующая команда)
- RSP,ESP,SP,SPL (Стек)
- RBP,EBP,BP,BPL
- RSI,ESI,SI,SIL (Источник)
- RDI,EDI,DI,DIL (Назначение)

Регистр флагов

- RFLAGS,EFLAGS,FLAGS

Системные регистры

- ...

Введение в Ассемблер

Hello world

```
1  global  _main
2  extern  _ExitProcess@4
3  extern  _printf
4
5  section .text   use32
6  _main:
7          push    msg
8          call    _printf
9
10         xor     eax, eax
11         push    eax
12         call    _ExitProcess@4
13 section .data
14 msg:    db "Hello World!!!", 13, 10, 0
```

Объявление глобальных и используемых имён.

global — *сообщаем какие имена необходимо сделать видимыми в создаваемом объектном файле.*
extern — *какие имена нам понадобятся из других объектных файлов. Т.е., например, функции внешних библиотек.*

Hello world

```
1  global  _main
2  extern  _ExitProcess@4
3  extern  _printf
4
5  section .text   use32
6  _main:
7          push    msg
8          call    _printf
9
10         xor     eax, eax
11         push    eax
12         call    _ExitProcess@4
13 section .data
14 msg:    db "Hello World!!!", 13, 10, 0
```

Объявление секций кода и данных.

Hello world

```
1  global  _main
2  extern  _ExitProcess@4
3  extern  _printf
4
5  section .text   use32
6  _main:
7          push    msg
8          call    _printf
9
10         xor     eax, eax
11         push    eax
12         call    _ExitProcess@4
13 section .data
14 msg:    db "Hello World!!!", 13, 10, 0
```

Код программы.

Как правило, используется выравнивание кода по трём колонкам шириной 8 символов: метки, команды, параметры.

Hello world

```
1  global  _main
2  extern  _ExitProcess@4
3  extern  _printf
4
5  section .text   use32
6  _main:
7          push    msg
8          call    _printf
9
10         xor     eax, eax
11         push    eax
12         call    _ExitProcess@4
13 section .data
14 msg:    db "Hello World!!!", 13, 10, 0
```

Константы

Команды языка Ассемблер: mov

Прототип

mov {приёмник}, {источник}

- Команда **mov** копирует значение из источника в приёмник.
- Приёмником может выступать регистр или участок памяти. Источником — регистр, участок памяти или значение.
- Приёмник и источник должны быть одного размера.
- Один из операндов должен быть регистром.

Примеры

```
; Поместить в регист eax число 123
        mov     eax, 123
; Поместить в регист eax значение переменной A
        mov     eax, [A]
; Поместить в регист eax адрес переменной msg
        mov     eax, msg
```

Неправильные примеры

```
; Пример №1
        mov     eax, bh
; Пример №2
        mov     [B], [A]
```

Команды языка Ассемблер: mov

Прототип

mov {приёмник}, {источник}

- Команда **mov** копирует значение из источника в приёмник.
- Приёмником может выступать регистр или участок памяти. Источником — регистр, участок памяти или значение.
- Приёмник и источник должны быть одного размера.
- Один из операндов должен быть регистром.

Примеры

```
; Поместить в регист eax число 123
        mov     eax, 123
; Поместить в регист eax значение переменной A
        mov     eax, [A]
; Поместить в регист eax адрес переменной msg
        mov     eax, msg
```

Неправильные примеры

```
; Пример №1
        mov     eax, bh
; Пример №2
        mov     [B], [A]
```

Команды языка Ассемблер: mov

Прототип

mov {приёмник}, {источник}

- Команда **mov** копирует значение из источника в приёмник.
- Приёмником может выступать регистр или участок памяти. Источником — регистр, участок памяти или значение.
- Приёмник и источник должны быть одного размера.
- Один из операндов должен быть регистром.

Примеры

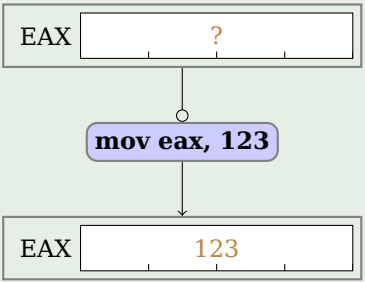
```
; Поместить в регист eax число 123
        mov     eax, 123
; Поместить в регист eax значение переменной A
        mov     eax, [A]
; Поместить в регист eax адрес переменной msg
        mov     eax, msg
```

Неправильные примеры

```
; Пример №1
        mov     eax, bh
; Пример №2
        mov     [B], [A]
```

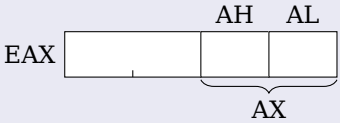
Пример использования команды mov

Пример



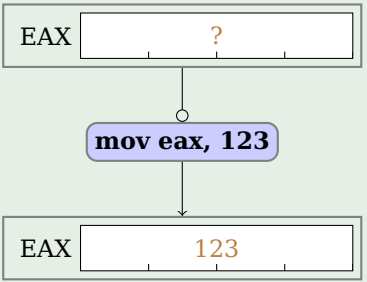
Вопрос

Чему стали равны **AX**, **AH**, **AL**?



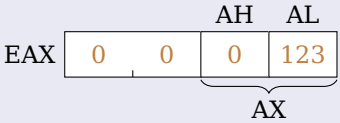
Пример использования команды mov

Пример



Вопрос

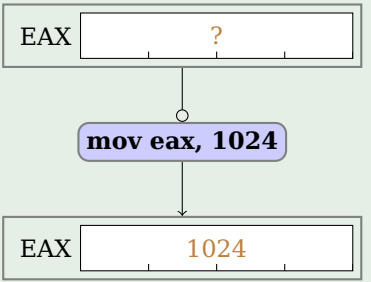
Чему стали равны **AX**, **AH**, **AL**?



$$123 = 0 \cdot 256^3 + 0 \cdot 256^2 + 0 \cdot 256^1 + 123 \cdot 256^0$$

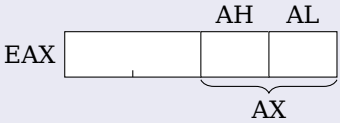
Пример использования команды mov

Пример



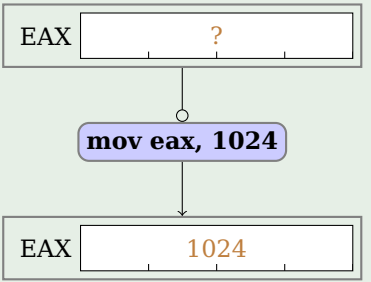
Вопрос

Чему стали равны **AX**, **AH**, **AL**?



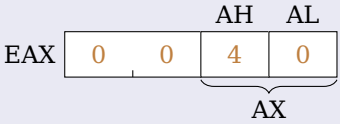
Пример использования команды mov

Пример



Вопрос

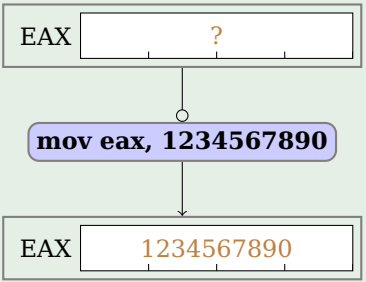
Чему стали равны **AX**, **AH**, **AL**?



$$1024 = 0 \cdot 256^3 + 0 \cdot 256^2 + 4 \cdot 256^1 + 0 \cdot 256^0$$

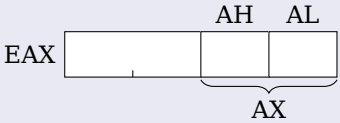
Пример использования команды mov

Пример



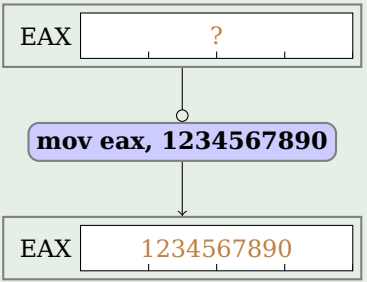
Вопрос

Чему стали равны **AX**, **AH**, **AL**?



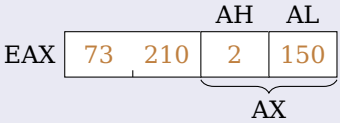
Пример использования команды mov

Пример



Вопрос

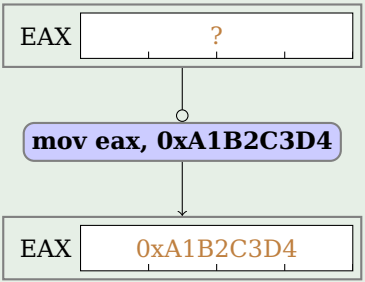
Чему стали равны **AX**, **AH**, **AL**?



$$1234567890 = 73 \cdot 256^3 + 210 \cdot 256^2 + \\ + 2 \cdot 256^1 + 150 \cdot 256^0$$

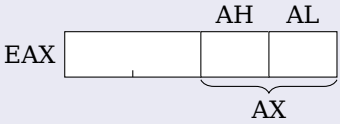
Пример использования команды mov

Пример



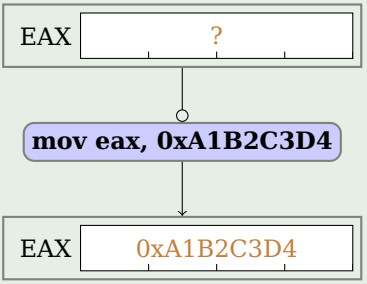
Вопрос

Чему стали равны **AX**, **AH**, **AL**?



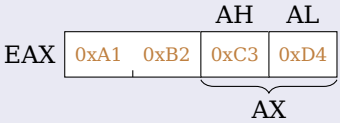
Пример использования команды mov

Пример



Вопрос

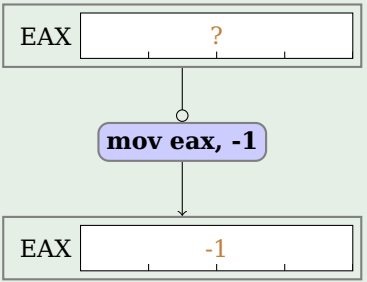
Чему стали равны **AX**, **AH**, **AL**?



$$A1B2C3D4_{16} = A1_{16} \cdot 100^3_{16} + B2_{16} \cdot 100^2_{16} + \\ + C3_{16} \cdot 100^1_{16} + D4_{16} \cdot 100^0_{16}$$

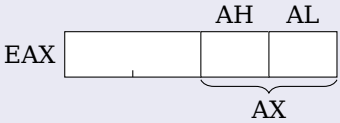
Пример использования команды mov

Пример



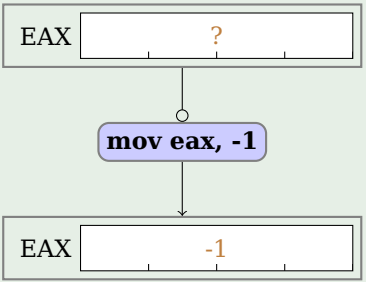
Вопрос

Чему стали равны **AX**, **AH**, **AL**?



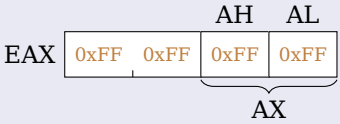
Пример использования команды mov

Пример



Вопрос

Чему стали равны **AX**, **AH**, **AL**?



Команды языка Ассемблер: add

Прототип

add {приёмник}, {источник}

- Команда сложения двух одинаковых целых чисел ({приёмник} и {источнике}).
- Результат сложения помещается в {приёмник}.

Примеры

```
add eax, 123 ; Прибавить к регистру eax число 123
add eax, ebx ; Прибавить к регистру eax регистр ebx
```

Команды языка Ассемблер: add

Прототип

add {приёмник}, {источник}

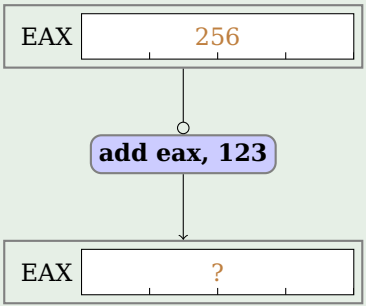
- Команда сложения двух одинаковых целых чисел ({приёмник} и {источнике}).
- Результат сложения помещается в {приёмник}.

Примеры

add eax, 123 *; Прибавить к регистру eax число 123*
add eax, ebx *; Прибавить к регистру eax регистр ebx*

Пример использования команды add

Пример

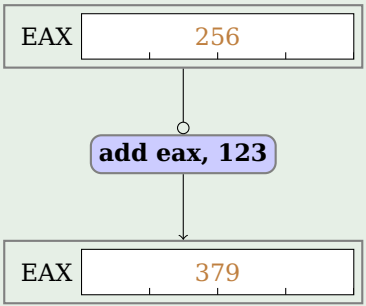


Вопрос

Какой результат?

Пример использования команды add

Пример

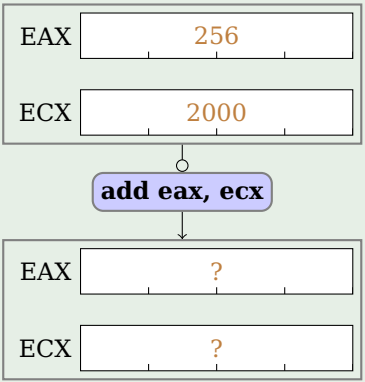


Вопрос

Какой результат?

Пример использования команды add

Пример

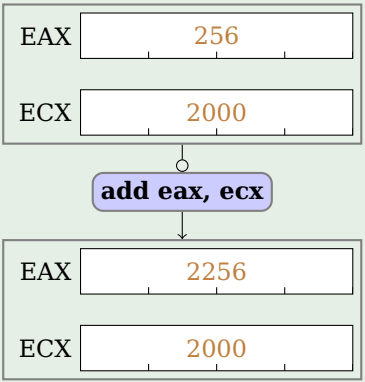


Вопрос

Какой результат?

Пример использования команды add

Пример

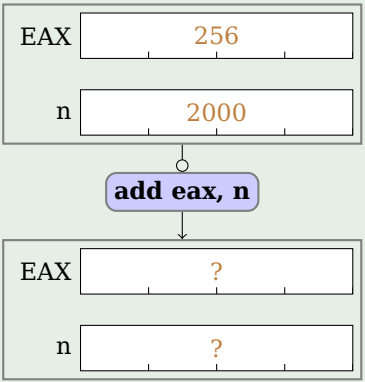


Вопрос

Какой результат?

Пример использования команды add

Пример

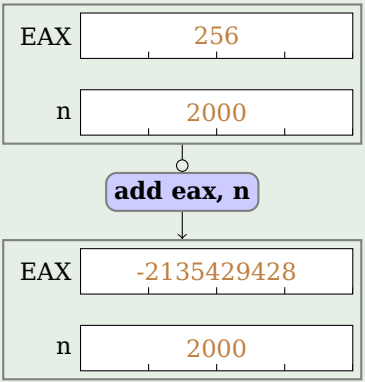


Вопрос

Какой результат?

Пример использования команды add

Пример



Вопрос

Какой результат?

Команды языка Ассемблер: push

Прототип

push {источник}

-

Команды языка Ассемблер: pop

Прототип

pop {приёмник}

-

Подпрограммы

Определение

Соглашение о вызове (англ. *calling convention*) — часть двоичного интерфейса приложений (англ. *application binary interface, ABI*), которая регламентирует технические особенности вызова подпрограммы, передачи параметров, возврата из подпрограммы и передачи результата вычислений в точку вызова. [Wiki]

Как правило, соглашение о вызове определяется из:

- Архитектуры компьютера (в том числе и разрядности режима процессора).
- Операционной системы.
- Языка программирования.

Определение

Соглашение о вызове (англ. *calling convention*) — часть двоичного интерфейса приложений (англ. *application binary interface, ABI*), которая регламентирует технические особенности вызова подпрограммы, передачи параметров, возврата из подпрограммы и передачи результата вычислений в точку вызова. [Wiki]

Как правило, соглашение о вызове определяется из:

- Архитектуры компьютера (в том числе и разрядности режима процессора).
- Операционной системы.
- Языка программирования.

Соглашение о вызове

Что определяет

- Как передать параметры.
  - Через регистры или стек.
  - В каком порядке.
- Как вернуть результат.
  - Через регистры или стек.
  - Кто выделяет память для больших результатов.
- Какие регистры можно изменить в подпрограмме.
- Кто очистит стек, если он использовался для передачи параметров.
- Какие суффиксы и префиксы используются для имён подпрограмм.
- Какое выравнивание использовать для элементов в стеке/памяти.
- Какие типы чисел разрешены.
- ...

Соглашение о вызове

Что определяет

- Как передать параметры.
  - Через регистры или стек.
  - В каком порядке.
- Как вернуть результат.
  - Через регистры или стек.
  - Кто выделяет память для больших результатов.
- Какие регистры можно изменить в подпрограмме.
- Кто очистит стек, если он использовался для передачи параметров.
- Какие суффиксы и префиксы используются для имён подпрограмм.
- Какое выравнивание использовать для элементов в стеке/памяти.
- Какие типы чисел разрешены.
- ...

Соглашение о вызове

Что определяет

- Как передать параметры.
  - Через регистры или стек.
  - В каком порядке.
- Как вернуть результат.
  - Через регистры или стек.
  - Кто выделяет память для больших результатов.
- Какие регистры можно изменить в подпрограмме.
- Кто очистит стек, если он использовался для передачи параметров.
- Какие суффиксы и префиксы используются для имён подпрограмм.
- Какое выравнивание использовать для элементов в стеке/памяти.
- Какие типы чисел разрешены.
- ...

Соглашение о вызове

Что определяет

- Как передать параметры.
  - Через регистры или стек.
  - В каком порядке.
- Как вернуть результат.
  - Через регистры или стек.
  - Кто выделяет память для больших результатов.
- Какие регистры можно изменить в подпрограмме.
- Кто очистит стек, если он использовался для передачи параметров.
- Какие суффиксы и префиксы используются для имён подпрограмм.
- Какое выравнивание использовать для элементов в стеке/памяти.
- Какие типы чисел разрешены.
- ...

Примеры

cdecl — Основной способ вызова в Си для 32-бит архитектур.

pascal — Основной способ вызова для Паскаля, также применялся в Windows 3.x..

fastcall — Передача параметров через регистры. Но, Fastcall не стандартизирован!!! И имеет разные модификации в зависемости от архитектуры/языка программирования/ОС.

stdcall — Применяется при вызове функций WinAPI.

safecall — Обеспечивает более удобный для использования в распространённых языках высокого уровня способ вызова методов интерфейсов при использовании модели COM.

thiscall — Используется в компиляторах C++. Обеспечивает передачу аргументов при вызовах методов класса в объектно ориентированной среде.

cdecl

- Параметры функции передаются через стек в направлении справа-налево.
- Выравнивание параметров: 4 байта.
- Результат функции передаются:
  - в EAX для данных размером $\leq$ 4 байт;
  - в паре EAX:EDX для данных размером $\leq$ 8 байт;
  - в регистре ST0 математического сопроцессора для действительных чисел;
  - иначе вызывающая функция выделяет память для результата и помещает адрес в вершину стека (так называемый «скрытый параметр»), ожидая что в EAX будет реальный адрес, где сохранился результат.
- Вызывающая функция должна очистить стек **самостоятельно**.
- Подпрограмма может изменить состояние регистров EAX, ECX и EDX, значения остальных должны быть восстановлены подпрограммой.
- Стек математического сопроцессора должен быть пустым перед и после вызова.

Команды языка Ассемблер: call

Прототип

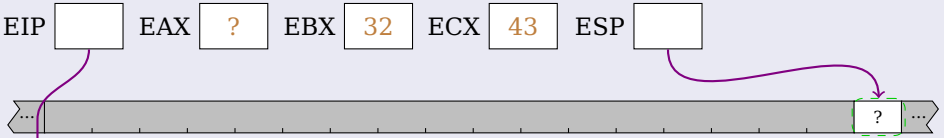
call {адрес}

-

Hello world

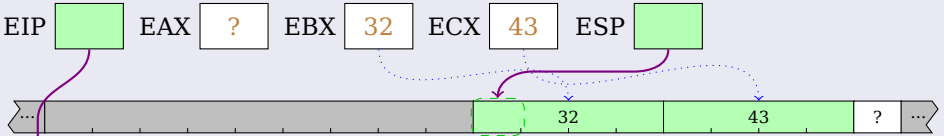
```
1  global  _main
2  extern  _ExitProcess@4
3  extern  _printf
4
5  section .text   use32
6  _main:
7          push    msg
8          call    _printf
9          add     esp, 4
10
11         xor     eax, eax
12         push    eax
13         call    _ExitProcess@4
14 section .data
15 msg:    db "Hello World!!!", 13, 10, 0
```

Вызов подпрограммы



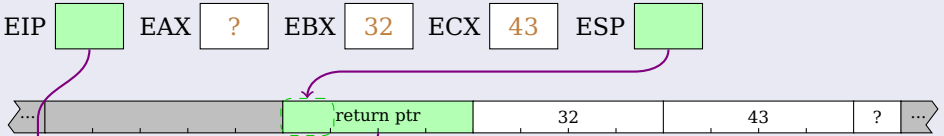

```
1          push    ecx         ; <—— y
2          push    ebx         ; <—— x
3          call    my_sum      ; <—— my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6  ...
7  my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы



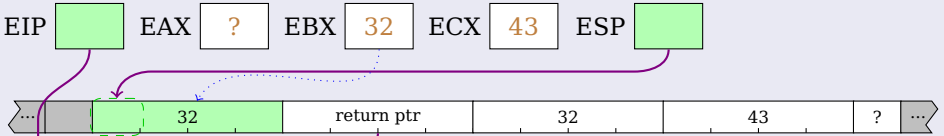

```
1          push    ecx         ; <—— y
2          push    ebx         ; <—— x
3          call    my_sum      ; <—— my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6  ...
7  my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы



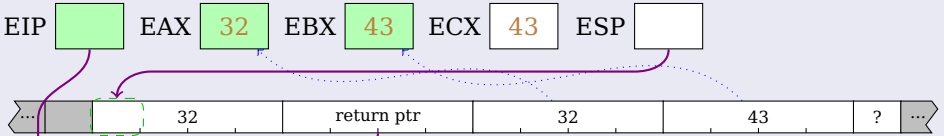

```
1          push    ecx         ; <—— y
2          push    ebx         ; <—— x
3          call    my_sum      ; <—— my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6   ...
7   my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы



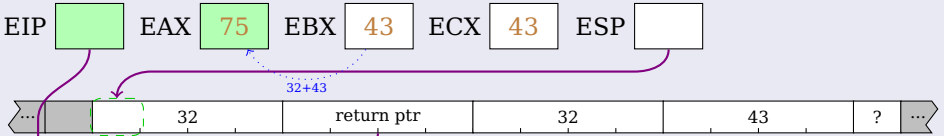

```
1          push    ecx         ; <-- y
2          push    ebx         ; <-- x
3          call    my_sum      ; <-- my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6  ...
7  my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы



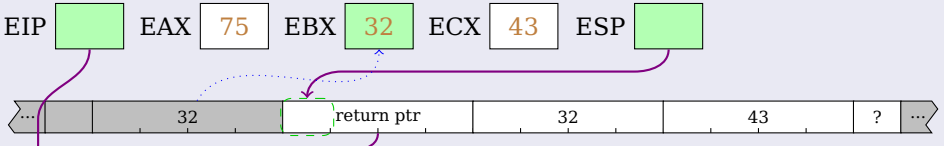

```
1          push    ecx         ; <-- y
2          push    ebx         ; <-- x
3          call    my_sum      ; <-- my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6   ...
7   my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы



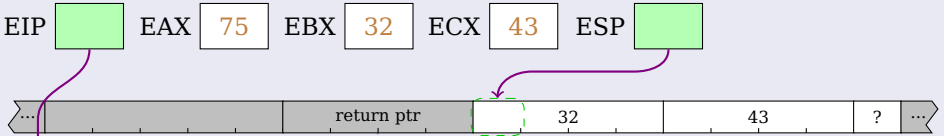

```
1          push    ecx         ; <—— y
2          push    ebx         ; <—— x
3          call    my_sum      ; <—— my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6   ...
7   my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы



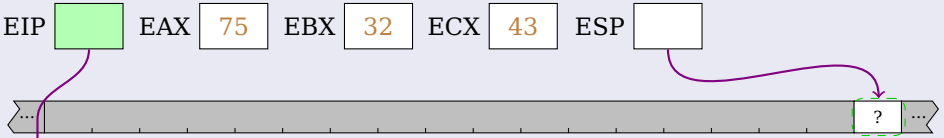

```
1          push    ecx       ; <-- y
2          push    ebx       ; <-- x
3          call    my_sum    ; <-- my_sum(x,y)
4          add     esp, 8    ; очистка стека
5          ; тут некая команда
6  ...
7  my_sum:
8          push    ebx       ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret               ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы




```
1          push    ecx         ; <-- y
2          push    ebx         ; <-- x
3          call    my_sum      ; <-- my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6  ...
7  my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вызов подпрограммы




```
1          push    ecx         ; <-- y
2          push    ebx         ; <-- x
3          call    my_sum      ; <-- my_sum(x,y)
4          add     esp, 8      ; очистка стека
5          ; тут некая команда
6  ...
7  my_sum:
8          push    ebx         ; Используемые регистры помещаем в стек
9          mov     eax, [esp+8]
10         mov     ebx, [esp+12]
11         add     eax, ebx
12         pop     ebx
13         ret                 ; Возвращаемся и удаляем параметры из стека
```

Вопросы?

Сделанов в

$\LaTeX 2_\varepsilon$

Использовано

- пакеты: beamer, tikz
- строк кода: >1100