Языки программирования (Asm)

Занятие №1. Введение. Арифметические операции.

МИРЭА - РТУ, БК №252

Москва 2020

Введение

Язык ассемблера можно назвать самым низкоуровневым языком программирования. Помимо него существует возможность программирования в машинных кодах. Правда, вряд ли есть смысл писать шестнадцатиричные коды вместо ассемблерных инструкций. Например, понятная команда

```
1 add eax, ebx ; сложить два регистра EAX + EBX
```

могла быть записана как 01 D8. Данная запись сложна как в написании, так и в прочтении, поэтому и был создан язык ассемблера. Для того, чтобы научиться писать программы на языке ассемблера нужно изучить систему команд той ЭВМ, на которой предполагается запускать код, а также представлять общие принципы работы компьютера.
В данном курсе будет рассмотрено только написание программ для операционных систем семейства **Linux** на языке ассемблера **NASM**.

Отметим здесь то, что зачастую язык ассемблера называют просто ассемблером. Однако, непосредственно ассемблером является та программа, которая транслирует исходный код на языке ассемблера в машинный код. Как уже говорилось выше, мы будем писать программы под ассемблер NASM. Для трансляции исходного кода будут использоваться следующие команды bash.

```
nasm -felf64 hello.asm
ld hello.o -o hello
./hello
```

Для совместимости с 32-разрядной архитектурой можно использовать следующие параметры.

```
nasm -felf32 hello.asm
ld -m elf_i386 hello.o -o hello
./hello
```

При этом, 32-разрядный код можно запускать на 64-разрядных машинах, но не наоборот.

Копирование данных. Команда MOV

Копирование данных, которое в языках программирования высокого уровня выполнялось оператором присвоения (=), в языке ассемблера зачастую будет выполняться командой MOV.
Общий синтаксис команды следующий:

```
1 MOV приемник, источник
```

Команда MOV может осуществлять копирование данных из оперативной памяти в регистры, и наоборот

```
1 mov ebx, [param]
2 mov [summa], eax
```

из одного регистра в другой

```
1 mov ebx, eax
```

а также может заносить непосредственное значение в регистр.

```
1 mov eax, 3
2 mov ebx, 2
```

Арифметические операции

Четыре основные арифметические операции выполняются при помощи следующих команд

```
1 add eax, ebx
2 sub eax, ebx
3 mul ebx
4 div ebx
```

После команд ADD (сложение) и SUB (вычитание) указывается два регистра (приемник, источник). Результат операции будет занесен в приемник. После команд MUL (умножение) и DIV (деление) указывается только источник, приемником всегда является регистр EAX.

Инкремент и декремент в ассемблере задаются командами INC и DEC, соответственно.

```
1 inc eax
2 dec ebx
```

Вывод данных

Чтобы разобраться с тем, как работает вывод информации на экран, рассмотрим код программы hello_world.asm

```
1  section .data:
2    hello db 'Hello world', 0xA
3    len equ $ - hello
4
5  section .text:
6    global _start
7  _start:
8    mov eax, 4
9    mov ebx, 1
10   mov ecx, hello
11   mov edx, len
12   int 0x80
13
14   mov eax, 1
15   mov ebx, 0
16   int 0x80
```

Метка _start нужна для создания точки входа в программу. По аналогии с функцией main языка C, программа написанная на языке ассемблера будет выполняться начиная с метки _start.

```
1 section .text
2   global _start
```

Строка с сообщением, а также длина нашей строки хранятся в отдельной секции кода программы

```
1 section .data:
2   hello db 'Hello world', 0xA
3   len equ $ - hello
```

В этой секции программы хранятся переменные, значение которых задано заранее.

Далее мы должны обратиться к операционной системе для вывода строки на экран. Для этого заполняем 4 регистра необходимыми значениями. Число 4 в EAX говорит о том, что будет вызвана стандартная функция write, число 1 в EBX говорит о том, что нужно использовать стандартный поток вывода. В регистр ECX помещается адрес строки, а в регистр EDX ее размер. После этого должно быть вызвано прерывание процессора.

```
1 mov eax, 4
2 mov ebx, 1
3 mov ecx, hello
4 mov edx, len
5 int 0x80
```

И наконец аналогом return 0 в конце функции main являются следующие команды

```
1 mov eax, 1
2 mov ebx, 0
3 int 0x80
```

Ввод данных

Для ввода данных, сначала выделим свободную память в секции .bss

```
section .bss
  x resb 3 ; выделить 3 байта
```

Для считывания трех символов в переменную x нужно выполнить следующие команды

```
mov eax, 3
mov ebx, 0
mov ecx, x
mov edx, 3
int 0x80
```

Число 3 в EAX говорит о том, что будет вызвана функция чтения, число 0 в EBX указывает на стандартный поток ввода, регистр ECX должен содержать адрес переменной, а в регистре EDX по-прежнему хранится число байт. После этого вызывается прерывание процессора.

Работа с символами

Следует помнить, что в предыдущих примерах все данные выводятся на экран и считываются из терминала как ASCII-символы, а не как целые или любые другие числа. Для того, чтобы производить математические операции над числами, считанными из терминала, нужно преобразовывать их в числа. Также и для вывода чисел на экран, их нужно преобразовывать в строки.
Символу 0 соответствует целое число 48
'0' = 0x30 = 48d
Поэтому, при считывании одной цифры из терминала, нужно отнять от значения этого байта число 48 (оно же 0x30).

```
1 mov eax, [x]
2 sub eax, 0x30
```