Архитектура ЭВМ

Лекция 4. Чем удобны ЯП высокого уровня?

к.ф.-м.н. Филонов Павел Владимирович
filonovpv@gmail.com

Московский Государственный Технический Университет
Гражданской Авиации

22 сентября 2013 г.

Смотрите в этой серии

Смотрите в этой серии

1. Типы данных и форматы констант

Смотрите в этой серии

1. Типы данных и форматы констант
2. Арифметика на x86

Смотрите в этой серии

1. Типы данных и форматы констант
2. Арифметика на x86
3. Disassemble&Debugging

Смотрите в этой серии

1. Типы данных и форматы констант
2. Арифметика на x86
3. Disassemble&Debugging
4. Jump'ов разных и побольше!

Смотрите в этой серии

1. Типы данных и форматы констант
2. Арифметика на x86
3. Disassemble&Debugging
4. Jump'ов разных и побольше!
5. Побитовые операции

Структура программы

```
; NASM example program
; by Filonov Pavel (filonovpv at gmail.com)
; 23 Jul 2013

%include "stud_io.inc"

global _start

section .data ; секция инициализированных данных
    ; ...

section .bss  ; секция неинициализированных данных
    ; ...

section .text ; секция исполняемого кода
_start:
    ; ...
    FINISH
```

DB и его друзья

```
;<метка> <псевдоинструкция> <список значений>
ten dd 10
msg db "Hello, world"
```

Псевдоинструкции в .data

```
db  0x55                    ; просто байт 0x55
db  0x55, 0x56, 0x57        ; три байта по порядку
db  'a', 0x55               ; можно использовать символы
db  'hello', 13, 10, '$'    ; как и строки
dw  0x1234                  ; 0x34 0x12
dw  'a'                     ; 0x61 0x00 (просто число)
dw  'ab'                    ; 0x61 0x62 (символьная константа)
dw  'abc'                   ; 0x61 0x62 0x63 0x00 (строка)
dd  0x12345678              ; 0x78 0x56 0x34 0x12
dd  1.234567e20             ; число с плавающей точкой
dq  0x123456789abcdef0      ; 8-ми байтное число
dq  1.234567e20             ; число двойной точности
dt  1.234567e20             ; число увеличенной точности
```

RESB и его друзья

```
;<метка> <псевдоинструкция> <размер>
ten dd 10
msg db "Hello, world"
```

Псевдоинструкции в .bss

```
buffer      resb    64   ; резервирует 64 байта
wordvar     resw    1    ; резервирует одно слово (2 байта)
regval      resd    1    ; размер одного регистра (4 байта)
realarray   resq    10   ; массив из 10 вещественных чисел
```

EQU

```
msg db "Hello, world", 0x0a
len equ $-msg
```

INCBIN

```
incbin "file.dat"           ; вставить весь файл
incbin "file.dat",1024,512 ; пропустить первые 1024 байта и
                            ; и вставить не больше 512 байт
```

Числовые константы

```
200           ; десятичная
0200          ; всё ешё десятичная
0200d         ; явно десятичная
0d200         ; также десятичная
0c8h          ; шестнадцатеричная (hex)
$0c8          ; снова hex (0 обязательно)
0xc8          ; опять hex
0hc8          ; всё ещё hex
310q          ; восьмеричная
0o310         ; опять восьмеричная
0q310         ; и снова восьмеричная
11001000b     ; двоичная
1100_1000b    ; та же двоичная константа
1100_1000y    ; и снова двоичная
0b1100_1000   ; она же
0y1100_1000   ; всё ещё двоичная
```

Команда MOV

MOV (Move) — набор команд, предназначенных для пересылки данных из одного места в другое

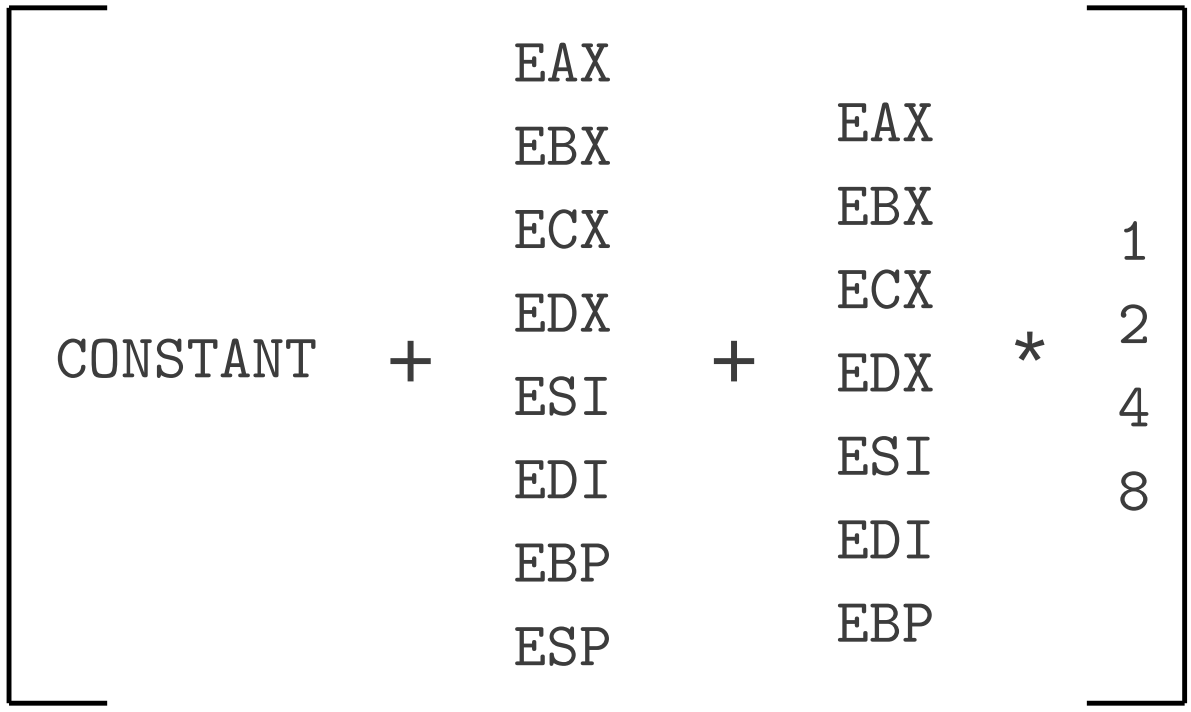
Виды операндов

```
mov eax, ebx        ; из одного регистра в другой
mov eax, 112        ; задать(непосредственным операндом)
                    ; значение регистра
mov eax, [count]    ; из памяти в регистр, здесь
                    ; count - адрес в памяти (метка), а
                    ; [count] - значение по этому адресу
mov [count], eax    ; из регистра в память
mov [x], dword 25   ; задать(непосредственным операндом)
                    ; значение ячейки памяти
```

Сцецификаторы размеров операнда

- byte — байт
- word — слово (2 байта)
- dword — двойное слово (4 байта)

Исполняемый адрес



LEA (Load Effective Address) — загружает в регистр вычисленное значение исполняемого адреса. Обращение к памяти не производится

```
lea edi, [array + eax + 4*ebx]
```

Пример

Команды сложения и вычитания

ADD (Addition) — целочисленное сложение
SUB (Substraction) — целочисленное вычитание

Примеры

```
add eax, ebx          ; eax := eax + ebx
add al,  cl           ; al := al + cl
add al,  10           ; al := al + 10
sub ebx, [a]          ; ebx := ebx - [a]
sub [a], dword 100    ; [a] := [a] - 100
```

Команды ADD и SUB устанавливают флаги (ZF, SF, OF, CF, PF)
ADC (Add with Carry) — сложить с переносом (к сумме прибавляется значение флага CF)
SBB (Substraction with Borrow) — вычитание с заимствованием (из разности вычитается CF)

Команды inc, dec, neg и cmp

INC (Increase) — увеличивает значение операнда на единицу
DEC (Decrease) — уменьшает значение операнда на единицу
Команды INC, DEC устанавливают флаги ZF, OF, SF (но не CF)

NEG (Negative) — изменяет знак числа (вычисляет дополнительный код)
CMP (Compare) — вычитает второй операнд из первого (результат не сохраняется). CMP используется ради установки флагов и обычно за ним следует условный переход

Беззнаковое умножение и деление

MUL (multiplication) — беззнаковое целочисленное умножения (в качестве операнда указывается только один множитель, второй задан неявно разрядностью операции)
DIV (division) — беззнаковое целочисленное деление (в качестве операнда указывается делитель, делимое задаётся неявно разрядностью операции)
При выполнении команды DIV не забудьте обнулить старшую часть делимого!

Разря-дность	Умножение		Деление		
	неявный множитель	результат умножения	делимое	частное	остаток
8	AL	AX	AX	AL	AH
16	AX	DX:AX	DX:AX	AX	DX
32	EAX	EDX:EAX	EDX:EAX	EAX	EDX

Знаковое умножение и деление

iMUL (multiplication) — знаковое целочисленное умножение. Имеет форму с одним, двумя или тремя операндами.
iDIV (division) — знаковое целочисленное деление (в качестве операнда указывается делитель, делимое задаётся неявно разрядностью операции)
CBW, CWD, CWDE, CDQ — увеличение разрядности знакового числа. cbw расширяет AL до AX, cwd AX до DX:AX, cwde AX до EAX, cdq EAX до EDX:EAX

Кол-во операндов	Запись	Описание
1	imul r/m	Аналогично mul
2	imul r, r/m	Первый := Первый*Второй
3	imul r, r/m, imm	Первый := Второй*Третий

Отладчик gdb

Программа gdb предназначение для отладки программ на различных языках программирования.
Использование: gdb программа

основные команды

- run (r) — начать выполнение программы
- break (b) — установить точку остановки
- next instruction (ni) — следующая инструкция
- step instruction (si) — следующая инструкция (с заходом в подпрограммы)
- continue (c) — продолжить выполнение
- info registers (i r) — показать значение регистров
- print (p)— распечатать значение регистра или памяти
- quit (q) — выход

Условные и безусловные переходы

Виды переходов

- **Дальние** (far) — для передачи управления в другой сегмент (не используются в «плоской»> модели памяти)
- **Близкие** (near) — для передачи управление в произвольное место внутри одного сегмента
- **Короткие** (short) — переходы, используемые для оптимизации и позволяющие прыгать не более чем на 127 байт вперёд и не менее чем на 128 байт назад.

Безусловные переходы по умолчанию объявляются близкими.
Условные переходы по умолчанию объявляются короткими.
Если адрес перехода на попадает в необходимый диапазон, то nasm выдаст ошибку следующего вида:
`error: short jump is out of range`

Условные и безусловные переходы

Виды переходов

- **Дальние** (far) — для передачи управления в другой сегмент (не используются в «плоской»> модели памяти)
- **Близкие** (near) — для передачи управление в произвольное место внутри одного сегмента
- **Короткие** (short) — переходы, используемые для оптимизации и позволяющие прыгать не более чем на 127 байт вперёд и не менее чем на 128 байт назад.

Безусловные переходы по умолчанию объявляются близкими.
Условные переходы по умолчанию объявляются короткими.
Если адрес перехода на попадает в необходимый диапазон, то nasm выдаст ошибку следующего вида:
`error: short jump is out of range`

Условные переходы по отдельным флагам

команда	условие	команда	условие
jz	$ZF = 1$	jnz	$ZF = 0$
js	$SF = 1$	jns	$SF = 0$
jc	$CF = 1$	jnc	$CF = 0$
jo	$OF = 1$	jno	$OF = 0$
jp	$PF = 1$	jnp	$PF = 0$

Условные переходы по результатам сравнений

команда	jump if	Выражение	Условие
je jne	equal not equal	$a = b$ $a \neq b$	ZF = 1 ZF = 0
jl jnge	less not greater or equal	$a < b$	SF $\neq$ OF
jle jng	less or equal not greater	$a \leq b$	SF $\neq$ OF или ZF = 1
jg jnle	greater not less or equal	$a > b$	SF=OF и ZF = 0
jge jnl	greater or equal not less	$a \geq b$	SF=OF
jb jnae	below not above or equal	$a < b$	CF = 1
jbe jna	below or equal not above	$a \leq b$	CF=1 или ZF = 1
ja jnbe	above not less or equal	$a > b$	CF=0 и ZF = 0
jae jnb	above or equal not below	$a \geq b$	CF=0

Условные переходы и ECX

Регистр **ECX** традиционно используется как счётчик цикла и в архитектуре x86 присутствует несколько специальных условных переходов, связанных со значением данного регистра.

loop — уменьшает значение **CX** и делает которкий переход, если CX не равен нулю.

jcxz (jump if CX is zero) — производит условный переход, если значение счётчика **CX** равно нулю.

loope, jecxz — аналогичные команды, работающие со значением регистра **ECX**

Побитовые операции

логические операции

- **AND** — логическое "И"
- **OR** — логическое "ИЛИ"
- **NOT** — логическое "НЕ"
- **XOR** — исключающее "ИЛИ"
- **TEST** — выполняет AND, но только устанавливает флаги

операции сдвига

- **SHR** (shift right) — простой сдвиг вправо (деление на 2^n)
- **SHL** (shift left) — простой сдвиг влево (умножение на 2^n)
- **SAR** (shift arithmetic right) — арифметический сдвиг вправо (знаковое деление на 2^n)
- **SAL** (shift arithmetic left) — арифметический сдвиг влево (синоним **SHL**)
- **ROR** (rotate right) — циклический сдвиг вправо
- **ROL** (rotate left) — циклический сдвиг влево