Введение в архитектуру ЭВМ

Юрий Анатольевич Богоявленский, заведующий кафедрой Информатики и математического обеспечения, к.т.н., доцент: ybgv@cs.petrsu.ru, https://cs.petrsu.ru

Веб страница дисциплины

https://cs.petrsu.ru/~chistyak/architecture/

# Лекция 3. Обзор системы команд. Типы данных. Литералы. Директивы определения данных

## План лекции

# Обзор команд ЦП общего назначения

Эти команды унаследованы от архитектуры `i8086.` Ниже приведены обзор и упрощенные описания этих команд для `ISA IA-32` подготовленные на базе учебного пособия:

https://edu.petrsu.ru/files/upload/2124_1427288068.pdf

Структура обзора соответствует структуре оглавления этого пособия.

Для некоторых классов задач было реализовано много различных расширений системы команд, которые кратко представлены в разделе «Расширения системы команд».

### Передача данных

### Передачи общего назначения

### MOV Пересылка данных

```
MOV Пересылка данных
      Move data
Перен  Напр   Прер   Шаг   Знак   Нуль   Всп   Четн   Перенос
```

O D I T S Z A P C

| Код | Команда | Краткое описание |
|---|---|---|
| 88 /r | mov r/m8,r8 | из регистра-байта в r/m-байт |
| 89 /r | mov r/m16,r16 | из регистра-слова в r/m-слово |
| 89 /r | mov r/m32,r32 | из регистра-двойного слова в r/m-двойное слово |
| 8A /r | mov r8,r/m8 | из r/m-байта в регистр-байт |
| 8B /r | mov r16,r/m16 | из r/m-слова в регистр-слово |
| 8B /r | mov r32,r/m32 | из r/m-двойного слова в регистр-двойное слово |
| 8C /r | mov r/m16,Sreg | из сег.регистра в r/m-слово |
| 8E /r | mov Sreg,r/m16 | из r/m-слова в сег.регистр |
| A0 | mov AL,moffs8 | из байта (сег:пер) в AL |
| A1 | mov AX,moffs16 | из слова (сег:пер) в AX |
| A1 | mov EAX,moffs32 | из двойного слова (сег:пер) в EAX |
| A2 | mov moffs8,AL | из AL в байт (сег:пер) |
| A3 | mov moffs16,AX | из AX в слово (сег:пер) |
| A3 | mov moffs32,EAX | из EAX в двойное слово (сег:пер) |
| B0+rb | mov reg8,imm8 | из байта в команде в регистр |
| B8+rw | mov reg16,imm16 | из слова в команде в регистр |
| B8+rw | mov reg32,imm32 | из двойного слова в команде в регистр |
| C6 | mov r/m8,imm8 | из байта в команде в r/m-байт |
| C7 | mov r/m16,imm16 | из слова в команде в r/m-слово |
| C7 | mov r/m32,imm32 | из двойного слова в команде в r/m-двойное слово |

Первый операнд получает значение второго операнда.
NBNB в синтаксисе AT&T — ВТОРОЙ операнд получает значение первого).

PUSH Запись слова в стек

POP Чтение слова из стека

XCHG Обмен значениями

## Ввод/вывод

IN Ввод из порта внешнего устройства

OUT Вывод в порт внешнего устройства

## Загрузка адресов

LEA Загрузка исполнительного адреса

## Передача флагов

PUSHF Запись в стек содержимого регистра флагов

POPF Чтение из стека в регистр флагов

## Арифметика

## Сложение, вычитание, сравнение

## ADD Сложение

ADD Сложение

Add

| Переп | Напр | Прер | Шаг | Знак | Нуль | Всп | Четн | Перенос |
|---|---|---|---|---|---|---|---|---|
| O | D | I | T | S | Z | A | P | C |
| P | | | | P | P | P | P | P |

| Код | Команда | Краткое описание |
|---|---|---|
| 04 ib | ADD AL,imm8 | йбат в команде + AL |
| 05 iw | ADD AX,imm16 | слово в команде + AX |
| 05 id | ADD EAX,imm32 | двойное слово в команде + EAX |
| 80 /0 ib | ADD r/m8,imm8 | байт в команде + r/m-байт |
| 81 /0 iw | ADD r/m16,imm16 | слово в команде + r/m-слово |
| 81 /0 id | ADD r/m32,imm32 | двойное слово в команде + r/m-слово |
| 82 /0 ib | ADD r/m8,imm8 | байт в команде + r/m-байт |
| 83 /0 ib | ADD r/m16,imm8 | знаково-расширяемый байт в команде + r/m-слово |
| 83 /0 ib | ADD r/m32,imm8 | знаково-расширяемый байт в команде + r/m-двойное слово |
| 00 /r | ADD r/m8,r8 | регистр-байт + r/m-байт |
| 01 /r | ADD r/m16,r16 | регистр-слово + r/m-слово |
| 01 /r | ADD r/m32,r32 | регистр-слово + r/m-двойное слово |
| 02 /r | ADD r8,r/m8 | r/m-байт + регистр-байт |
| 03 /r | ADD r16,r/m16 | r/m-слово + регистр-слово |
| 03 /r | ADD r32,r/m32 | r/m-слово + регистр-двойное слово |

Приемник и источник складываются. Первый операнд (NBNB в синтаксисе AT&T - ВТОРОЙ) получает значение результата, устанавливаются флаги. Если байт в команде складывается с операндом-словом или двойным словом (код команды `83 /0`), то до сложения из байта в ЦП формируется операнд-слово или двойное слово, младший байт которого равен байту в команде, а биты старших байтов совпадают с его старшим битом (знаковое расширение).

ADC Сложение с переносом

SUB Вычитание

SBB Вычитание с заемом

CMP Сравнение операндов

### Умножение и деление

MUL Умножение беззнаковое

IMUL Умножение знаковое

DIV Деление беззнаковое

IDIV Деление знаковое

Уменьшение, увеличение, инверсия знака

DEC Уменьшение на 1

INC Увеличение на 1

NEG Инверсия знака дополнением до 2

### Знаковые расширения

CBW Знаковое расширение байта

CWD Знаковое расширение слова

### Работа с битами

### Логика

AND Логическое умножение (И)

NOT Логическое отрицание (НЕ)

OR Логическое сложение (ИЛИ)

XOR Логическое исключающее ИЛИ

### Сдвиги

SAL, SAR, SHL, SHR Логические и арифметические сдвиги

RCL, RCR, ROL, ROR Циклические сдвиги ..... 114

**Управление последовательностью выполнения**

**Работа с функциями**

CALL Вызов функции

RET Возврат из функции

**Переходы и циклы**

JMP Безусловный переход

Jcc **Условный переход**

Jcc Условный переход
Jump if condition is met

| Переп | Напр | Прер | Шаг | Знак | Нуль | Всп | Четн | Перенос |
|---|---|---|---|---|---|---|---|---|
| O | D | I | T | S | Z | A | P | C |

| Код | Команда | Условие перехода |
|---|---|---|
| По значению одного флага | | |
| 70 cb | JO rel8 | было переполнение (OF=1) |
| 0F 80 cw/cd | JO rel16/rel32 | было переполнение (OF=1) |
| 71 cb | JNO rel8 | не было переполнения (OF=0) |
| 0F 81 cw/cd | JNO rel16/rel32 | не было переполнения (OF=0) |
| 72 cb | JC rel8 | был перенос (CF=1) |
| 0F 82 cw/cd | JC rel16/rel32 | был перенос (CF=1) |
| 73 cb | JNC rel8 | не было переноса (CF=0) |
| 0F 83 cw/cd | JNC rel16/rel32 | не было переноса (CF=0) |

| | | |
|---|---|---|
| 74 cb | JE rel8 | равно (ZF=1) |
| | JZ rel8 | результат равен 0 (ZF=1) |
| 0F 84 cw/cd | JE rel16/rel32 | равно (ZF=1) |
| | JZ | результат равен 0 (ZF=1) |
| 75 cb | JNE rel8 | не равно (ZF=0) |
| | JNZ rel8 | не было нуля (ZF=0) |
| 0F 85 cw/cd | JNE rel16/rel32 | не равно (ZF=0) |
| | JNZ | не было нуля (ZF=0) |
| 78 cb | JS rel8 | был знак минус (SF=1) |
| 0F 88 cw/cd | JS rel16/rel32 | был знак минус (SF=1) |
| 79 cb | JNS rel8 | не было знака минус (SF=0) |
| 0F 89 cw/cd | JNS rel16/rel32 | не было знака минус (SF=0) |
| 7A cb | JP rel8 | была четность (PF=1) |
| | JPE rel8 | количество единиц четно (PF=1) |
| 0F 8A cw/cd | JP rel16/rel32 | была четность (PF=1) |
| | JPE | количество единиц четно (PF=1) |
| 7B cb | JNP rel8 | не было четности (PF=0) |
| | JPO rel8 | количество единиц НЕчетно (PF=0) |
| 0F 8B cw/cd | JNP rel16/rel32 | не было четности (PF=0) |
| | JPO | количество единиц НЕчетно (PF=0) |
| После операций над ЗНАКОВЫМИ операндами | | |

| | | |
|---|---|---|
| 7C cb | JL rel8 | меньше (SF<>OF) |
| | JNGE rel8 | не “больше или равно” (SF<>OF) |
| 0F 8C cw/cd | JL rel16/rel32 | меньше (SF<>OF) |
| | JNGE | не “больше или равно” (SF<>OF) |
| 7D cb | JGE rel8 | больше или равно (SF=OF) |
| | JNL rel8 | не меньше (SF=OF) |
| 0F 8D cw/cd | JGE rel16/rel32 | больше или равно (SF=OF) |
| | JNL | не меньше (SF=OF) |
| 7E cb | JLE rel8 | меньше или равно (ZF=1 и SF<>OF) |
| | JNG rel8 | не больше (ZF=1 и SF<>OF) |
| 0F 8E cw/cd | JLE rel16/rel32 | меньше или равно (ZF=1 и SF<>OF) |
| | JNG | не больше (ZF=1 и SF<>OF) |
| 7F cb | JG rel8 | больше (ZF=0 и SF=OF) |
| | JNLE rel8 | не “меньше или равно” (ZF=0 и SF=OF) |
| 0F 8F cw/cd | JG rel16/rel32 | больше (ZF=0 и SF=OF) |
| | JNLE | не “меньше или равно” (ZF=0 и SF=OF) |
| После операций над БЕЗзнаковыми операндами | | |
| 72 cb | JB rel8 | ниже (CF=1) |
| | JNAE rel8 | не “выше или равно” (CF=1) |
| | JC rel8 | был перенос (CF=1) |
| 0F 82 cw/cd | JB rel16/rel32 | ниже (CF=1) |

| | | |
|---|---|---|
| | JNAE | не “выше или равно” (CF=1) |
| | JC | был перенос (CF=1) |
| 73 cb | JAE rel8 | выше или равно (CF=0) |
| | JNB rel8 | не ниже (CF=0) |
| | JNC rel8 | не было перноса (CF=0) |
| 0F 83 cw/cd | JAE rel16/rel32 | выше или равно (CF=0) |
| | JNB | не ниже (CF=0) |
| | JNC | не было переноса (CF=0) |
| 76 cb | JBE rel8 | ниже или равно (CF=1 или ZF=1) |
| | JNA rel8 | не выше (CF=1 или ZF=1) |
| 0F 86 cw/cd | JBE rel16/rel32 | ниже или равно (CF=1 или ZF=1) |
| | JNA | не выше (CF=1 или ZF=1) |
| 77 cb | JA rel8 | выше (CF=0 и ZF=0) |
| | JNBE rel8 | не “ниже или равно” (CF=0 и ZF=0) |
| 0F 87 cw/cd | JA rel16/rel32 | выше (CF=0 и ZF=0) |
| | JNBE | не “ниже или равно” (CF=0 и ZF=0) |
| Проверка регистра ECX | | |
| E3 cb | JECXZ rel8 | регистр ECX содержит 0 |

`LOOPcc` Управление циклом по регистру (E)CX и флагу ZF

**Обработка прерываний**

`INT, INTO` Программное прерывание

IRET Возврат из программы обработки прерывания

IRETD Возврат из программы обработки прерывания

# Расширения системы команд

Эти расширения реализуются корпорацией Intel для повышения эффективности решения задач некоторых классов путем переноса нужных операций на уровень команд ЦП. К этим классам относятся, например:

- работа с числами с плавающей точкой;
- распараллеливание на основе SIMD;
- криптография;
- управление виртуальными машинами.

## Расширения SIMD для параллельной обработки данных

Классическая архитектура фон Неймана построена по принципу SISD (Single Instruction, Single Data – одна команда – одно данное). Реализация принципа SIMD (Single Instruction, Multiple Data – одна команда – несколько данных) обеспечивает повышение быстродействия за счет параллельного выполнения одной и той же операции над несколькими элементами данных. SIMD реализован расширениями MMX (Multi Media Extensions), SSE (Streaming SIMD Extensions) и другими.

Типичные области применения SIMD это обработка изображений, аудио, трехмерная графика, видео, компьютерные игры, программы компьютерного зрения, цифровая обработка сигналов. Также есть работы по внедрение SIMD в веб обозреватели (Chrome не будет работать без расширения SSE2).

Примеры работы с числами с плавающей точкой и с расширениями SIMD даются на третьем курсе в дисциплине по выбору «Архитектура современных ЭВМ».

Проверить наличие в процессоре команд расширений можно в ОС Linux с помощью утилиты cpuid, в ОС Windows с помощью утилиты CPU-Z.

Информацию о расширениях основной системы команд можно получить из источников:

x86 instruction listings [Электронный ресурс]. - URL: https://en.wikipedia.org/wiki/X86_instruction_listings (27.07.26)

Расширения архитектуры x86 [Электронный ресурс]. — URL:https://ru.ruwiki.ru/wiki/%D0%A0%D0%B0%D1%81%D1%88%D0%B8%D1%80%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D1%8B_x86 (27.07.26)

# Типы данных

Материал этого раздела подготовлен на основе главы 4 руководства «Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 1: Basic Architecture»: https://kappa.cs.petrsu.ru/~chistyak/architecture/docs/64-ia-32-architectures-software-developer-vol-1-manual.pdf

## Основные типы данных

Основные типы данных (см. рисунок ниже) это:

- байт — 8 битов;
- слово — 2 байта, 16 битов;
- двойное слово — 4 байта, 32 бита;
- четверное слово — 8 байтов, 64 бита;
- двойное четверное слово — 16 байтов, 128 битов.

Два последних не используются в командах общего назначения а только в командах расширений и нами не рассматриваются.

| Байт | |
|---|---|
| Биты | 7 0 |
| Адрес в ОП | N |

| Слово | Старший байт | Младший байт |
|---|---|---|
| Биты | 15 8 | 7 0 |
| Адрес в ОП | N+1 | N |

| Двойное слово | Старшее слово | Младшее слово |
|---|---|---|
| Биты | 31 16 | 15 0 |
| Адреа в ОП | N+2 | N |

На рисунке ниже показан порядок `Little endian` (от младшего к старшему) байтов в памяти для основных типов данных, когда их адреса используются для указания операндов команд в памяти.

Младший байт каждого типа данных имеет самый меньший адрес в памяти, и этот адрес также является адресом слова и двойного слова как операнда.

| | Адрес a+0 Четверное слово **E4B5C3FA93EA0D3F** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Адрес a+0 Двойное слово **93EA0D3F** | | | | Адрес a+4 Двойное слово **E4B5C3FA** | | | |
| | Адрес a+o Слово **0D3F** | | Адрес a+2 Слово **93EA** | | Адрес a+4 Слово **C3FA** | | Адрес a+6 Слово **E4B5** | |
| | | Адрес a+1 Слово **EA0D** | | | | | | |
| Адрес ОП | a+0 | a+1 | a+2 | a+3 | a+4 | a+5 | a+6 | a+7 |
| Байты числа | 3F | 0D | EA | 93 | FA | C3 | B5 | E4 |

**Выравнивание адресов оперативной памяти для данных основных типов**

Естественные границы адресов основных типов данных называются адреса ОП делящиеся нацело согласно таблице:

| Тип данных | Делится на |
|---|---|
| слово | 2 |
| двойное слово | 4 |
| четверное слово | 8 |
| двойное четверное слово | 16 |

Адрес, не соответствующий таблице называется невыровненным.

Как правило, не требуется выравнивание данных по естественным границам, однако, для повышения производительности программ, структуры данных (особенно стеки) должны быть выровнены когда это возможно.

Дело в том, что процессору требуется два обращения к ОП (цикла шины) для доступа по невыровненному адресу; для доступа по выровненному адресу требуется только одно обращение к ОП.

Некоторые команды, работающие с двойными четверными словами, требуют, чтобы операнды в памяти были выровнены по естественной границе.

В языке ассемблера `gas` нужное выравнивание можно задать директивами `.align` `.balign` и `.p2align`.

## Числовые типы данных

Интерпретации основных типов данных как беззнаковых и знаковых целых чисел и диапазоны их значений даны в лекции 2 в разделе «Беззнаковый и знаковый типы целых чисел».

## Числа с плавающей точкой

Целые числа и диапазоны их представления в ОП не подходят для многих задач, поэтому был введено представление с «плавающей точкой», соответствующее записи числа вида:

$$Q = m \cdot e^{p},$$

где m — мантисса, e — основание, p — порядок.

Эта представление соответствует т. н. научной или экспоненциальной записи. Пример — фундаментальная физическая константа — постоянная Авогадро

$$N_A = 6.0214076 \cdot 10^{23} \text{ моль}^{-1}.$$

Форматы чисел с плавающей точкой определены стандартами IEEE 754-1985, IEEE 754-2008 и IEEE 754-2019, которые определяют набор базовых двоичных и десятичных форматов. Порядки диапазонов представления чисел с плавающей точкой представлены в таблице ниже.

| Тип данных | Длинав битах | Диапазоны значений | |
|---|---|---|---|
| | | Двоичный | Десятичный |
| Half Precision | 16 | $2^{-14} \cdots 2^{15}$ | $3.1 \times 10^{-5} \cdots 6.50 \times 10^{4}$ |
| Single Precision | 32 | $2^{-126} \cdots 2^{127}$ | $1.18 \times 10^{-38} \cdots 3.40 \times 10^{38}$ |
| Double Precision | 64 | $2^{-1022} \cdots 2^{1023}$ | $2.23 \times 10^{-308} \cdots 1.79 \times 10^{308}$ |
| Double Extended Precision | 80 | $2^{-16382} \cdots 2^{16383}$ | $3.37 \times 10^{-4932} \cdots 1.18 \times 10^{4932}$ |

Числа с плавающей точкой являются моделью (неточной) множества действительных чисел, для работы с ними есть набор команд, реализуемый математическим сопроцессором (FPU - Floating Point Unit).

В работе И.С. Григорьев  Числа с плавающей точкой, М., МГУ им. М. В. Ломоносова, 2024, [Электронный ресурс]

URL: http://mech.math.msu.su/~iliagri/sem2book.htm (08.08.2026) представлены особенности и приемы работы с этими числами.

Автор отмечает: «Вычисления с плавающей точкой и реализация вычислительных алгоритмов представляет собой важный и сложный раздел программирования; идеологически он принципиально отличается от обычного целочисленного программирования. Главная его особенность состоит в том, что вычисления с плавающей точкой производятся не точно, вычислительная ошибка накапливается.»

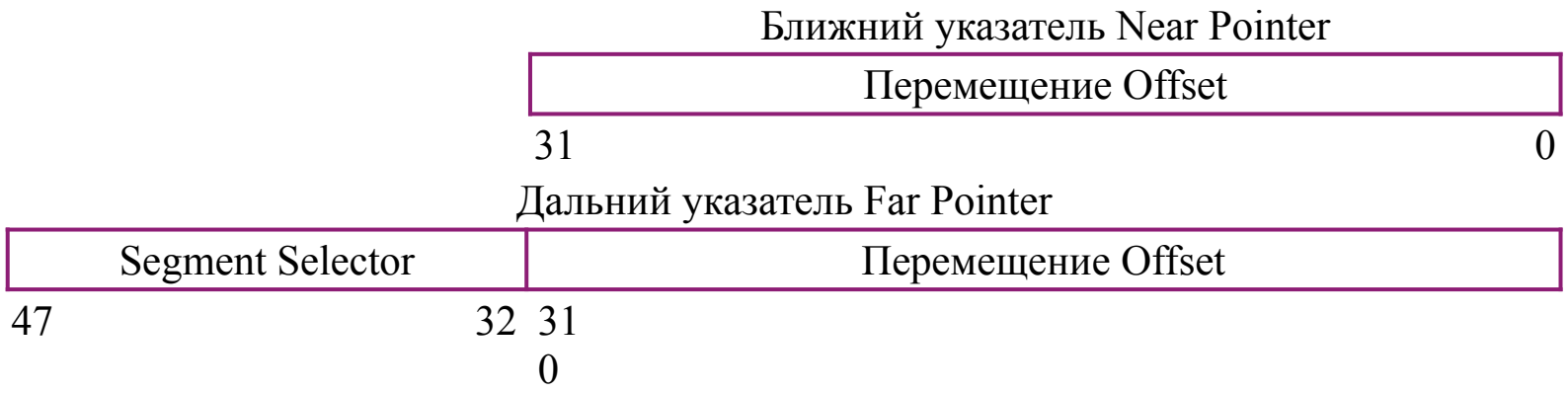
## Тип данных указатель

Указатели - это адреса байтов ОП.

В IA-32 определено два вида указателей:

- near (ближний) - это 32b (или 16b) перемещение (англ. offset), называемое фактическим адресом (effective address).
- far (дальний) — это 48 битовое значение, состоящее из 16b селектора сегмента и 32b (или 16b) перемещения. Он называется логическим адресом.

Ближние указатели задают перемещения в плоской модели ОП, где адрес сегмента задается неявно, или внутри сегмента в сегментной модели ОП, где адрес сегмента в ОП задается явно значением селектора сегмента в сегментном регистре.




## Типы данных битовые поля, строки, упакованные данные SIMD и целые BCD числа

*Битовое поле* это непрерывная последовательность произвольного количества битов, которая начинается с любой разрядной позиции байта в ОП и может содержать до 32 битов.

*Строки* представляют собой непрерывные последовательности битов, байтов, слов или двойных слов. Строка битов может начинаться с любой битовой позиции любого

байта и содержать до $2^{32-1}$ бита. Байтовая строка может содержать байты, слова или двойные слова в диапазоне от нуля до $2^{32-1}$ байтов.(4 Гбайт).

*Упакованные данные SIMD* введены для задания операндов команд SIMD в виде наборов данных которые помещаются в специальные регистры. Например, в 128-битный регистр SSE можно «упаковать» сразу четыре 32-битных числа или два 64-битных числа с плавающей точкой, операции над которыми будут выполняться параллельно, т.е. за время выполнения операции на одним числом.

Работа с SIMD булет рассмотрена на третьем курс в дисциплине по выбору “Архитектура современных ЭВМ“.

*Целые BCD числа* (Binary-coded decimal) обеспечивают представление чисел в ОП непосредственно в десятичной системе, без использования двоичной, что исключает необходимость перевода. При этом десятитонная цифра кодируется в полубайте двоичными значениями от 0000 до 1001. BCD числа применяются также в калькуляторах, устройствах IoT и .т.п. Мы не рассматриваем операции с данными этого типа.

# Литералы

Литерал — это явная запись конкретного значения данного какого-либо типа прямо в исходном коде программы, которое нельзя изменить во время ее работы.

### Символьные литералы

Определены два вида символьных литералов — односимвольные и строковые. Односимвольные занимают один байт, их значения можно использовать в арифметических выражениях. Строковые могут

занимать несколько байтов, их значения нельзя использовать в арифметических выражениях.

## Строковые литералы и escape-последовательности

Строковый литерал это строка из символов текущего набора записанная между символами двойная кавычка, например:

“Две капли сверкнут“

В коде ASCII определены коды однобайтовых непечатаемых управляющих символов и символов, имеющие специальное значение. Они также используются для указания действий, например возврата каретки или табуляции, на терминалах и принтерах

Для помещения в строковые и односимвольные литералы этих кодов в языках программирования реализуется механизм escape-последовательностей — комбинаций символов, начинающихся с экранирующего символа обратной косой черты, за которой следует другой символ или последовательность цифр. Определены следующие escape-последовательности (в конце строки дан шестнадцатеричный код символа).

\a — звуковой сигнал (BEL, 0x07);

\b — возврат на один символ назад (Backspace, 0x08);

\t — горизонтальная табуляция (HT, 0x09);

\n — новая строка (LF, 0x0A);

\v — вертикальная табуляция (VT, 0x0B);

\f — перевод страницы (FF, 0x0C);

\r — возврат каретки (CR, 0x0D);

\' — одинарная кавычка (апостроф, 0x27)

\" — двойная кавычка (строгий апостроф, 0x22).

\\ — обратный слеш (обратная косая черта, 0x5C).

\?  — вопросительный знак, 0x3F

\000  — символ с восьмеричным кодом 000 (от 0 до 377).

\xhh  — символ с шестнадцатеричным кодом hh (от 0 до FF)

Например, в конец строкового литерала “Введите цифру, друг мой\n“ будет вставлен байт 0x0A. После вывода этого литерала на терминал курсор будет переведен в начало следующей строки.

## Односимвольные литералы

Такой литерал задается одинарной кавычкой, за которой следует символ, кодируемый одним байтом, т. е. символ с кодами от 0 до 127 (7F) кода ASCII. В арифметическом выражении значением односимвольнго литерала является однобайтовое целое его кода ASCII. Например литерал 'A имеет значение 0x41, литерал 'a — 0x61.

Escape-последовательности \b, \f, \n, \r, \t, \', \" и \\ также можно использовать для определения этих литералов.

Так, команды:

```
cmpb $'\n', c # это символ перевода строки ?
je kbd_input  # ДА - на ввод следующего символа
```

выполнят вычитание значений байтов:

(код англиского символа c) - \n

и если флаг нуля Z получит значение 1, т.е. значения операндов вычитания равны, то передадут управление на метку kbd_input.

Заметим, что закрывающая кавычка в синтаксисе не требуется, но и не отмечается как ошибка и для улучшения читаемости кода рекомендуется ее использовать, т. е писать 'A', '\n'.

## Числовые литералы

В `gas` различают три типа чисел в зависимости от того, как они хранятся в ОП целевой ЭВМ.

Целые числа соответствуют типу `int` в языке `C`. Большие числа (занимают в ОП более 32 битов) и числа с плавающей точкой мы не рассматриваем.

Числовые целые литералы в системе счисления с основанием:

`10` — не начинаются с цифры 0, не содержат 16-ых цифр;

`8` — начинаются с цифры 0;

`16` — начинаются с символов 0x или 0X;

`2` — начинаются с символов 0b или 0B.

Если цифрам литерала предшествует знак ‘+‘ или ‘-‘, то формируется целое в дополнительном коде, иначе — в беззнаковом представлении.

# Директивы определения данных

В лекции 1 директивы ассемблера определены как один из видов операторов

Напомним формат директивы:

```
<метка> .<имя директивы> пробел/таб <список параметров>
```

Директивы определения данных переводят в двоичную систему числовые литералы, формируют двоичные коды строковых и односимвольных литералов, выделяют ОП для данных и размещают их двоичные коды в объектном файле.

### Определение строковых данных

```
.ascii "строка1", "строка2". . .
```

Параметры — один или несколько строковых литералов, разделённых запятыми. Коды символов текущей локали каждого литерала записывается по

последовательным адресам (без автоматического добавления завершающего нулевого байта)

```
.asciiz "строка1", "строка2". . .
```

Аналогична предыдущей, но в конце каждого литерала добавляется нулевой байт (в стиле языка Си).

### Определение данных основных типов

Согласно разделу «Основные типы данных» это байт, слово, двойное и четверное слова. Директивы определения этих данных имею обобщенную форму:

`<Имя_типа> выражение1, выражение2` ...

Параметры — один или несколько `выражений времени ассеммблирования`, разделённых запятыми (будут рассмотрены в следующих лекциях, простые примеры см. ниже в тексте программы). Для каждого выражения будет получено и размещено в объектном файле двоичное значение выражения соответствующей длины.

`<Имя_типа>` может быть задано как:

- `.byte;`
- `.hword` — `16b` слово (синонимы `.short` и для нашей и некоторых других целевых архитектур — `.word` );
- `.int` — порядок байтов и разрядность зависят от целевой архитектуры (для нас — 32b и `Little endian`), синоним — `.long;`
- `.quad` — `64b` данное, аналогичное `.int,` нами не используется.

### Определение размера данных в секции `.bss`

Образ секции `.bss` не формируется в объектном и исполняемом файлах а создается в момент запуска исполняемого файла загрузчиком ОС и всегда

заполняется только нулевыми байтами. При определении в ней символьных имена можно задать только их размер в байтах директивой:

```
.lcomm <Символьное имя>, <длина>
```

Пример из листинга программы task3.S.

```
bss
.lcomm buf, 100 # 100b буфер для кодов прочитанных символов
.lcomm c, 1     # 1b буфер для чтения байта из файла stdin
. . .
DEFINED SYMBOLS
                task3.S:8      .bss:0000000000000000 buf
                task3.S:8      .bss:0000000000000064 c
```

Т.е. символьное имя `buf` имеет адресное значение `0x00000000` в секции `bss` и занимает в ней `0x64b`, а символьное имя `c` имеет адресное значение `0x00000064` и занимает один байт.

## Пример

```
GAS LISTING d-t-over.S                  page 1


   1                #    Демонстрация директив определения данных.
   2                #    Переполнение для знакового и беззнакового сложения
   3
   4                #    Ассемблирование и редактирование связей - команда m
   5
   6                    .include "my-macro" # подключение файла с
макроопределениями
   1                // Макроопределение завершения работы
   2
   3                .macro Finish
   4                         movl $0, %ebx # first argument: exit code
   5                         movl $1, %eax # sys_exit index
   6                         int $0x80 # kernel interrupt
   7                .endm
   8
   7
   8                .data # секция данных, распределение памяти
   9
  10                #   Строковые данные и escape последовательности
  11
  12 0000 61626320     Str:    .ascii  "abc ABC 123"
  12      41424320
```

```
  12      313233
  13 000b D094D0B2     Str1:   .ascii  "Две капли сверкнут, сверкнут", "на
дне!\n" # escape
  13      D0B520D0
  13      BAD0B0D0
  13      BFD0BBD0
  13      B820D181
  13      D0B2D0B5
  13      D180D0BA
  13      D0BDD183
  13      D1822C20
  13      D181D0B2
  13      D0B5D180
  13      D0BAD0BD
  13      D183D182
  13      D0BDD0B0
  13      20D0B4D0
  13      BDD0B521
  13      0A
  14 004c D0ADD184     Str2:   .asciz  "Эфес о ладонь согреешь!"  # 0x00
байт
  14      D0B5D181
  14      20D0BE20
  14      D0BBD0B0
  14      D0B4D0BE
  14      D0BDD18C
  14      20D181D0
  14      BED0B3D1
  14      80D0B5D0
  14      B5D188D1
  14      8C2100
  15 0077 D0B85CD0     Str3:   .ascii  "и\\или"
# текст и\или
  15      B8D0BBD0
  15      B8
  16
  17              #   Данные основных типов
  18
  19              #       Использование односимвольных литералов

GAS LISTING d-t-over.S                page 2


  20
  21 0080 31000000     OneSymbol:  .int '1', '2', '3'
  21      32000000
  21      33000000
  22 008c 31003200             .hword '1', '2', '3'
  22      3300
  23 0092 31                       .byte '1
  24 0093 32                       .byte '2
  25 0094 63                       .byte '1'+'2'
  26 0095 41                       .byte '\A
  27 0096 0A                       .byte '\n
  28
  29
  30              #       Система счисления целых литералов
```

```
  31
  32 0097 41          Diff_bases: .byte  65   # 10
  33 0098 41                      .byte  0101  # 8
  34 0099 21                      .byte  041   # 8
  35 009a 41                      .byte  0x41  # 16
  36 009b 41                      .byte  0X41  # 16
  37
  38              #                    8      16     10
  39 009c E5040000                .long  02345, 0x4E5, 1253
  39      E5040000
  39      E5040000
  40
  41              #   Параметры - простые выражения
  42
  43              Arifm_Expression:
  44
  45 00a8 4141                     .byte 35+30, 20+45
  46 00aa 4141                     .byte 70-5, 80-15
  47
  48              #   Знаковые и беззнаковые целые
  49              #   и данные для демонстрации переполнения
  50
  51
  52 00ac FF           B1:          .byte   255 # max беззнаковое в байте
  53 00ad F0           B2:          .byte   240 # просто байт
  54
  55 00ae FF           Bm1:         .byte   -1  # знаковый байт
  56 00af 01           Bp1:         .byte   +1  # знаковый байт
  57
  58 00b0 80           B_zn_min:    .byte   -128    # мин в знаковом байте
  59 00b1 7F           B_zn_max:    .byte   +127    # мах в знаковом байте
  60
  61 00b2 FF00         S1:          .hword  255     # просто 16b число
  62 00b4 F000         S2:          .hword  240     # просто 16b число
  63
  64 00b6 FFFF         W_bzn_max:   .hword  65535   # 16b беззнаковый
максимум
  65 00b8 0080         W_zn_min:    .hword  -32768  # 16b знаковый минимум
  66 00ba FF7F         W_zn_max:    .hword  +32767  # 16b знаковый максимум
  67
  68 00bc FFFFFFFF     L1:          .long   4294967295  # 32b беззнаковый
максимум
  69 00c0 FFFFFFFF     Il:          .int    4294967295  # 32b беззнаковый
максимум
  70 00c4 00000080     I1:      .int    -2147483648 # 32b знаковый минимум
  71 00c8 FFFFFF7F     I2:      .int    +2147483647 # 32b знаковый максимум

GAS LISTING d-t-over.S              page 3


  72
  73 00cc 00000080     Q:       .quad   -2147483648 # 64b знаковый минимум
  73      FFFFFFFF
  74
  75              .text # секция команд
  76
  77              .global _start # точка входа
```

```
  78
  79 0000 66A1B200     _start:         movw    S1, %ax # слово 255
  79      0000
  80 0006 660305B4                     addw    S2, %ax # + слово 240 = 495
  80      000000
  81                                     # НЕТ переполнения
  82
  83 000d 29C0                         sub     %eax,%eax
  84 000f A0AC0000                     movb    B1, %al # байт 255
  84      00
  85 0014 0205AD00                     addb    B2, %al # + 240
  85      0000
  86 001a 7202                         jc      UnsignedOverflowb
  87 001c 90                           nop
  88 001d 90                           nop
  89
  90              UnsignedOverflowb:
  91 001e 29C0                         sub     %eax,%eax
  92 0020 66A1B600                     movw    W_bzn_max,%ax    # max
беззнаковое слово
  92      0000
  93 0026 6683C001                     addw    $1,%ax       # +1 к нему
  94 002a 7202                         jc      UnsignedOverflow_w
  95 002c 90                           nop
  96 002d 90                           nop
  97
  98              UnsignedOverflow_w:
  99 002e 29C0                         sub     %eax,%eax
 100 0030 66A1BA00                     movw    W_zn_max, %ax    # max
знаковое слово
 100      0000
 101 0036 6683C001                     addw    $1, %ax      # +1 к нему
 102 003a 7002                         jo      SignedOverflow
 103 003c 90                           nop
 104 003d 90                           nop
 105             SignedOverflow:
 106 003e 29C0                         sub     %eax, %eax
 107 0040 66A1B800                     movw    W_zn_min, %ax    # min
знаковое слово
 107      0000
 108 0046 6683E801                     subw    $1, %ax      # -1 от него
 109 004a 7002                         jo      SignOver
 110 004c 90                           nop
 111 004d 90                           nop
 112             SignOver:
 113
 114                                 Finish # конец работы, возврат в ОС
(макро из файла my-macro)
 114 004e BB000000     >  movl $0,%ebx
 114      00
 114 0053 B8010000     >  movl $1,%eax
 114      00
 114 0058 CD80         >  int $0x80
 115                                 .end   # последняя строка исходного
текста
GAS LISTING d-t-over.S                page 4
```

```
DEFINED SYMBOLS
          d-t-over.S:12     .data:00000000 Str
          d-t-over.S:13     .data:0000000b Str1
          d-t-over.S:14     .data:0000004c Str2
          d-t-over.S:15     .data:00000077 Str3
          d-t-over.S:21     .data:00000080 OneSymbol
          d-t-over.S:32     .data:00000097 Diff_bases
          d-t-over.S:43     .data:000000a8 Arifm_Expression
          d-t-over.S:52     .data:000000ac B1
          d-t-over.S:53     .data:000000ad B2
          d-t-over.S:55     .data:000000ae Bm1
          d-t-over.S:56     .data:000000af Bp1
          d-t-over.S:58     .data:000000b0 B_zn_min
          d-t-over.S:59     .data:000000b1 B_zn_max
          d-t-over.S:61     .data:000000b2 S1
          d-t-over.S:62     .data:000000b4 S2
          d-t-over.S:64     .data:000000b6 W_bzn_max
          d-t-over.S:65     .data:000000b8 W_zn_min
          d-t-over.S:66     .data:000000ba W_zn_max
          d-t-over.S:68     .data:000000bc L1
          d-t-over.S:69     .data:000000c0 Il
          d-t-over.S:70     .data:000000c4 I1
          d-t-over.S:71     .data:000000c8 I2
          d-t-over.S:73     .data:000000cc Q
          d-t-over.S:79     .text:00000000 _start
          d-t-over.S:90     .text:0000001e UnsignedOverflowb
          d-t-over.S:98     .text:0000002e UnsignedOverflow_w
          d-t-over.S:105    .text:0000003e SignedOverflow
          d-t-over.S:112    .text:0000004e SignOver

NO UNDEFINED SYMBOLS
```

# Выводы

- Структура обзора команд ЦП общего назначения соответствует структуре описания этих команд в учебном пособии: https://edu.petrsu.ru/files/upload/2124_1427288068.pdf .
- Для повышения эффективности решения задач некоторых классов в систему команд введены расширения.
- Примеры работы с числами с плавающей точкой и с расширениями SIMD даются на третьем курсе в дисциплине по выбору «Архитектура современных ЭВМ».
- Младший байт каждого основного типа данных согласно порядку Little endian (от младшего к старшему)имеет самый меньший адрес в памяти, и этот адрес также является адресом слова и двойного слова как операнда.

- Выравнивание данных по естественным границам, как правило, не требуется, однако для повышения производительности программ структуры данных (особенно стеки) должны быть выровнены когда это возможно.
- Определены два вида символьных литералов — односимвольные и строковые.
- `escape-последовательности,` начинающиеся с экранирующего символа обратной косой черты введены для помещения в строковые и односимвольные литералы непечатаемых управляющих символов и символов, имеющие специальное значение.
- Значением односимвольнго литерала является однобайтовое целое его кода `ASCII`, его можно использовать в арифметических выражениях.
- Если цифрам числового литерала предшествует знак `'+'` или `'-'`, то формируется целое в дополнительном коде, иначе — в беззнаковом представлении.
- Директивы определения данных переводят в двоичную систему числовые литералы, формируют двоичные коды строковых и односимвольных литералов, выделяют ОП для данных и размещают их двоичные коды в объектном файле.
- Образ секции `.bss` не формируется в объектном и исполняемом файлах а создается в момент запуска исполняемого файла загрузчиком ОС и всегда заполняется только нулевыми байтами. При определении в ней символьных имена можно задать только их размер в байтах: