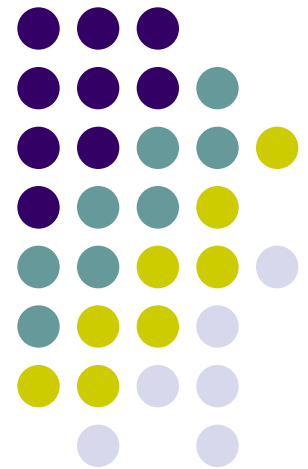


Тема 3

Форматы данных в Ассемблере. Команды Ассемблера





Форматы данных

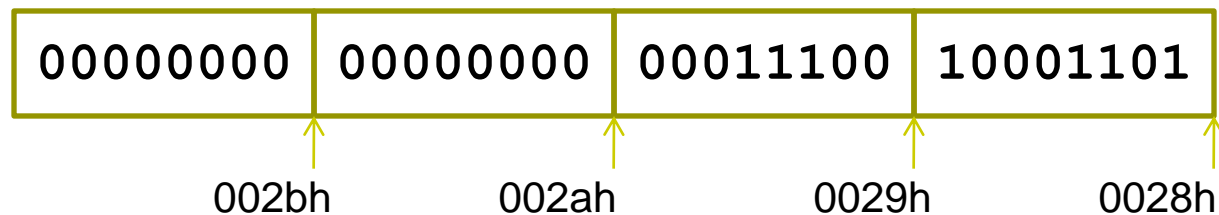
Различают несколько форматов данных, из которых можно выделить:

- целые числа в двоичной системе счисления;
- адреса;
- двоично-десятичные числа;
- числа с плавающей точкой.



Представление целых чисел в двоичной системе счисления

- могут занимать байт, слово или двойное слово (то есть 1, 2, или 4 байта соответственно);
- старшие разряды числа хранятся в старших адресах памяти



Представление числа 7309 ($1C8B_{16}$) в двойном слове, начинающемся со смещения 0028h.

Однако в памяти оно будет в виде $8B1C_{16}$

Представление отрицательных чисел



Отрицательные числа хранятся в дополнительном коде.

Правила преобразования в дополнительный код:

- записываем двоичное представление модуля числа;
- инвертируем каждый бит (т.е. заменяем 0 на 1 и наоборот);
- к результату прибавляем единицу.

Представление числа -7309 в двойном слове



1) Записываем число 7309 $1C8B_{16}$

00000000	00000000	00011100	10001101
----------	----------	----------	----------

2) Инвертируем биты

11111111	11111111	11100011	01110010
----------	----------	----------	----------

3) Прибавляем единицу к младшему разряду

11111111	11111111	11100011	01110011
----------	----------	----------	----------

Одинаковое представление для знаковых и беззнаковых чисел



Число со знаком: -7309

$FFFF\ E373_{16}$

11111111	11111111	11100011	01110011
----------	----------	----------	----------

Беззнаковое число: 4 294 959 987

11111111	11111111	11100011	01110011
----------	----------	----------	----------



Сложение целых чисел

При сложении целых чисел не учитывается то, знаковые они или беззнаковые. Вместо этого выставляются флаги **CF**, **OF**, **ZF**, **SF**:

- **CF=1**, если произошёл перенос из старшего разряда (*переполнение беззнаковых чисел*);
- **OF=1**, если результат сложения чисел одного знака имеет противоположный знак (*переполнение знаковых чисел*);
- **ZF=1**, если все биты результата равны нулю;
- **SF=1**, если старший бит результата равен единице (*результат можно трактовать как отрицательное число*)

Примеры сложения целых чисел



Формула	7+10	200+100	100+100
Первый операнд	00000111	11001000	01100100
Второй операнд	00001010	01100100	01100100
Результат	00010001	01001100	11001000
Установленные флаги	нет	CF	OF, SF

Формула	-100+(-100)	7 + (-10)	7 + (-7)
Первый операнд	10011100	00000111	00000111
Второй операнд	10011100	11110110	11111001
Результат	00111000	11111101	00000000
Установленные флаги	CF, OF	SF	CF, ZF

Адресные данные (указатели)



Различают **ближние** (**near**) и **дальние** (**far**) адреса.

В первом случае в памяти хранится, в зависимости от режима работы процессора, 16- или 32-битное смещение относительно некоторого сегмента. Адрес сегмента не задается, и нет никаких указаний относительно того, к какому сегменту относится это смещение. Программист сам должен определить это!

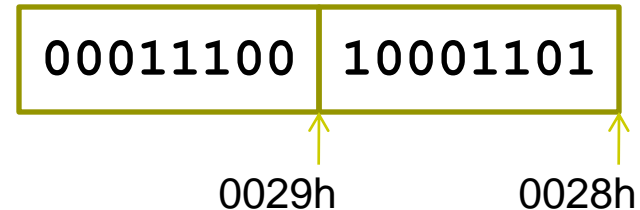
При использовании дальних указателей вместе со смещением (точнее, сразу за ним) записывается 16-битное значение, которое должно быть помещено в один из сегментных регистров. Общая длина дальнего указателя, таким образом, равна 32 или 48 бит, в зависимости от режима работы процессора.

Примеры хранения указателей

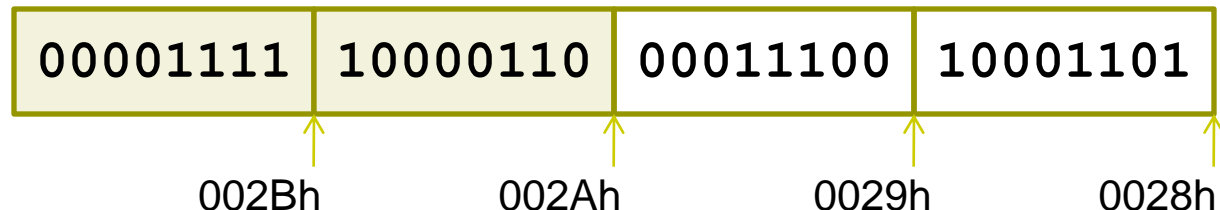


Пусть смещение в некотором сегменте равно **1C8Dh**.
Эту информацию необходимо хранить в указателе ,
смещение которого равно **0028h**.

Ближний указатель



Дальний указатель (адрес начала сегмента – 0F86h)



Двоично-десятичные данные



Двоично-десятичные данные хранят представление числа в десятичной системе счисления.

Различают **неупакованный** и **упакованный** форматы хранения двоично-десятичных чисел.

При хранении в неупакованном формате каждая цифра занимает один байт, в упакованном – в одном байте хранятся две цифры.

Длина числа и положение десятичной точки определяются программистом.

Пример хранения двоично-десятичных данных



Пусть нам надо хранить число 1279. В неупакованном формате для этого потребуются 4 байта, в упакованном достаточно двух.

Неупакованный формат:

00000001	00000010	00000111	00001001
----------	----------	----------	----------

Упакованный формат:

00010010	01111001
----------	----------



Направление возрастания адресов



Числа с плавающей точкой

Числа с плавающей точкой обрабатываются сопроцессором. Данные в этом случае задаются в виде знака **s**, мантиисы **M** и характеристики **q**:



- Бит знака устанавливается в 1, если число отрицательное;
- старший бит мантиисы равен 1, если число не равно 0.

Это **нормальная** запись числа:

$$3,1415926 = 0,31415926 * 10^1$$

$$1000,0001_2 = 0,10000001_2 * 2^4.$$

Числа с плавающей точкой (продолжение)



Различают короткий, длинный и расширенный форматы вещественных чисел (отличия состоят в длине характеристики и мантиссы).

Формат числа	<i>Короткий (float)</i>	<i>Длинный (double)</i>	<i>Расширенный</i>
<i>Длина числа (бит)</i>	32	64	80
<i>Длина мантиссы</i>	24	53	64
<i>Длина характеристики</i>	7	10	15
<i>Min/max числа</i>	3.4E-38 ... 3.4E+38	1.7E-308 ... 1.7E+308	3.37 E-4932 ... 1.18 E+4932

Машинные команды и команды Ассемблера



- **Машинная команда** – минимальная единица работы, с точки зрения программиста, которую способен выполнить процессор.

Машинные команды, хранимые в оперативной памяти согласно модели фон Неймана, представляют собой последовательности из нулей и единиц. Длина команды составляет от 1 до 15 байт.

- **Команда Ассемблера** – минимальная структурная единица программы на Ассемблере.

Зачастую говорят, что эти два понятия равнозначны.

Различия между машинными командами и командами Ассемблера



- операнды машинных команд, как правило, не выделяются явно, а «размазываются» по всему коду команды. Операнды команды Ассемблера задаются явно после имени команды;
- одному имени команды Ассемблера может, в зависимости от набора операндов, несколько машинных команд с различными кодами;
- для одной машинной команды может быть задано несколько мнемонических обозначений, и, соответственно, несколько команд Ассемблера.

Структура машинных команд и команд Ассемблера



Как машинная команда, так и команда Ассемблера содержит:

- **код команды** (что должна делать команда);
- **операнды команды** (описывают, где находятся данные для обработки и куда необходимо помещать результат).

Таким образом, процессор извлекает из команды следующую важную информацию:

- какие действия необходимо совершить при выполнении этой команды;
- где находятся данные, которые подлежат обработке, и куда необходимо поместить результат;
- какой формат имеют эти данные.



Операнды

Операнды содержат:

- информацию о том, где находятся данные, которые подлежат обработке, и куда необходимо поместить результат;
- информацию о том, какой формат имеют эти данные (в частности, длину данных).

Адресация



Адресация – информация о том, где находятся обрабатываемые командой данные.

Возможны следующие местоположения данных:

- в самой команде (непосредственная адресация);
- в регистрах (регистровая адресация);
- в памяти (адресация в памяти).

Как правило, в памяти располагается не более одного операнда!

Получение эффективного адреса при адресации в памяти



Эффективный адрес получается как сумма адресов, хранящихся:

- в самой команде;
- в регистрах **RBX/EBX/BX** или **RBP/EBP/BP**;
- в регистрах **RSI/ESI/SI** или **RDI/EDI/DI**.

(для защищенного режима могут использоваться любые регистры общего назначения).

Адрес сегмента при этом находится в одном из сегментных регистров.

Хотя бы одна часть адреса должна присутствовать!



Шаги выполнения команды

1. *Выборка команды.* Блок управления извлекает команду из памяти, копирует ее во внутреннюю память микропроцессора и увеличивает значение счетчика команд на длину этой команды.
2. *Декодирование команды.* Блок управления определяет тип выполняемой команды, пересылает указанные в ней операнды в арифметическо-логическое устройство (АЛУ).

Шаги выполнения команды



3. *Выборка операндов.* Если в команде используется операнд, расположенный в памяти, блок управления инициирует операцию по его выборке из памяти.
4. *Выполнение команды.* АЛУ выполняет указанную в команде операцию, сохраняет полученный результат в заданном месте и обновляет состояние флагов, по значению которых программа может судить о результате выполнения команды.
5. *Запись результата в память.* Если результат выполнения команды должен быть сохранен в памяти, блок управления инициирует операцию сохранения данных в памяти.

Цикл выполнения команды





Спасибо за
внимание!