

Лекция 5. Типы данных. Плавающая точка. Команды SIMD. Переполнение. Директивы определения данных.

План лекции

Числа с плавающей точкой.

Архитектура SIMD

Аппаратные типы данных

Логические типы данных

Целочисленное переполнение

Директивы ассемблера

Константы

Директивы определения данных

Пример «Директивы определения данных и обнаружение переполнения»

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

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

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

Эта запись берет начало от широко распространенной т. н. научной или экспоненциальной записи. Пример — фундаментальная физическая константа число Авогадро $N_A = 6.0214076 \cdot 10^{23}$ 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 закодированных в 32, 64, 128 и других битовых представлениях.

Базовые двоичные форматы определяются формулой:

$$Q = (-1)^s \cdot 2^{(p-127)} \cdot (1 + m/2^{23})$$

элементы которой в 32 битовом представлении распределены по битам как показано на рисунке.

Номера битов		
31	30 - 23	22 - 0
s	p	m

Где s - бит знака числа — 0 — положительное, 1 — отрицательное.

Примерный диапазон значений — от $1.2 \cdot 10^{-38}$ до $3.4 \cdot 10^{+38}$.

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> (13.07.2024)

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.»

Архитектура SIMD

В современных процессорах реализованы дополнительные наборы команд поддерживающие архитектуру SIMD (Single Instruction, Multiple Data — одиночный поток команд, множественный поток данных). Эта архитектура обеспечивает повышение быстродействия за счет параллельной обработки нескольких элементарных данных (например слов) одной командой. Работа в SIMD осуществляется с помощью следующих расширений системы команд:

- MMX (неофициально - MultiMedia eXtension);
- SSE (Streaming SIMD Extensions), SSE2, SSE3, SSSE3 and SSE4;
- AVX, (Advanced Vector Extensions) AVX2, AVX512, AMX.

Один из авторов расширения SSE, участник разработки Pentium III, д.т.н. Владимир Мстиславович Пентковский (1946 — 2012).

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 браузеры (Chrome не будет работать без SSE2). Отметим, что есть и другие расширения системы команд процессоров, например криптографическое — AES (Advanced Encryption Standard instruction set). Описание системы команд и ее расширений можно найти в источнике:

URL: https://en.wikipedia.org/wiki/X86_instruction_listings (18.01.24)

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

Пример вывода `cruid`.

```

CPU 0:
  vendor_id = "GenuineIntel"
  version information (1/eax):
    processor type = primary processor (0)
    family        = Intel Pentium Pro/II/III/Celeron/Core/Core 2/Atom,
AMD Athlon/Duron, Cyrix M2, VIA C3 (6)
    model         = 0xa (10)
    stepping id   = 0x7 (7)
    extended family = 0x0 (0)
    extended model = 0x2 (2)
    (simple synth) = Intel Core i3-2000 / Core i5-2000 / Core i7-2000 /
Mobile Core i7-2000 (Sandy Bridge D2/J1/Q0) / Pentium G500/G600/G800 /
Pentium B915C (Sandy Bridge Q0) / Celeron G400/G500/700/800/B800 (Sandy
Bridge J1/Q0) / Xeon E1-1100 / E3-1200 (Sandy Bridge D2/J1/Q0), 32nm
  feature information (1/edx):
    x87 FPU on chip                = true
    VME: virtual-8086 mode enhancement = true
. . .
    MMX Technology                  = true
. . .
    SSE extensions                  = true
    SSE2 extensions                 = true
  feature information (1/ecx):
    PNI/SSE3: Prescott New Instructions = true
. . .
    SSSE3 extensions               = true
. . .
    SSE4.1 extensions              = true
    SSE4.2 extensions              = true
. . .
    AES instruction                 = true
. . .
    AVX: advanced vector extensions = true
. . .
    RDRAND instruction              = false
. . .
  Thermal and Power Management Features (6):
    digital thermometer             = true
  extended feature flags (7):
. . .

```

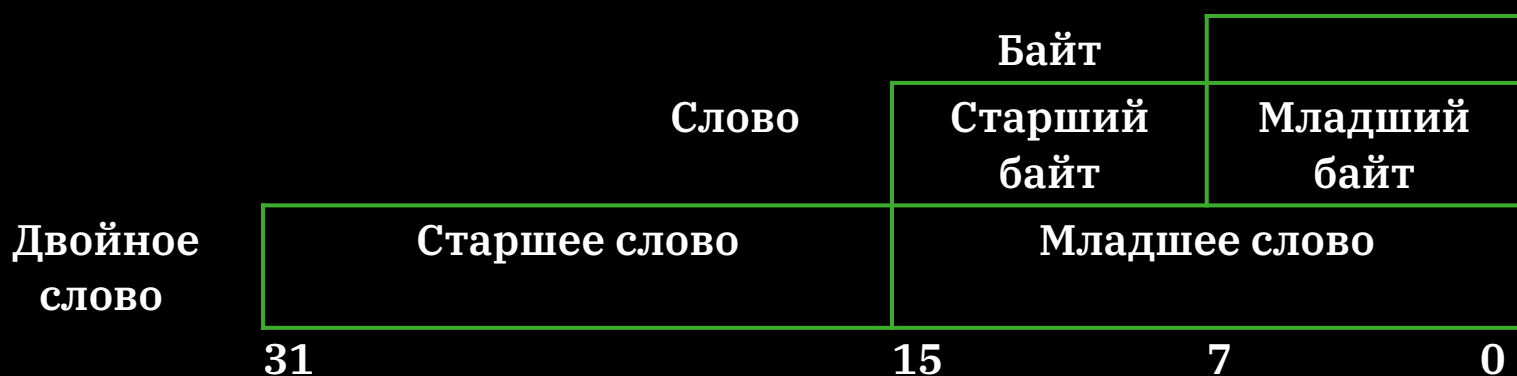
```

AVX2: advanced vector extensions 2      = false
. . .
AVX512PF: prefetch instructions          = false
AVX512ER: exponent & reciprocal instrs = false
AVX512CD: conflict detection instrs     = false
. . .
AVX512BW: byte & word instructions      = false
AVX512VL: vector length                 = false
. . .
AVX512VBMI: vector byte manipulation    = false
. . .
AVX512_VBMI2                           = false
. . .
AVX512_VNNI                             = false
AVX512_BITALG: bit count/shuffle        = false
AVX512: VPOPCNTDQ instruction           = false
. . .
AVX512_4VNNIW: neural network instrs    = false
AVX512_4FMAPS: multiply acc single prec = false
CPU 1:

```

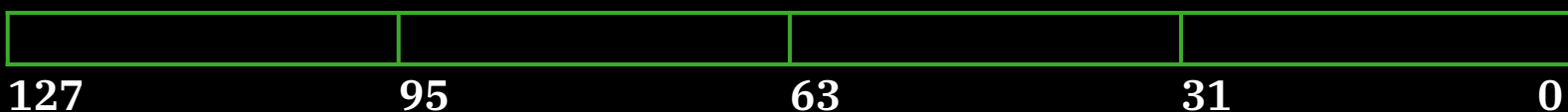
Аппаратные типы данных

Это представленные на рисунке единицы ОП к которым можно получить доступ при выполнении одной команды.



(в архитектуре Intel®64 есть и четверное 64 битное слово).

128-битный упакованный тип данных с плавающей точкой одинарной точности для SSE.



Адреса этих единиц доступа могут быть определены с помощью символических имен (прежде всего меток), которые затем можно

использовать в качестве операндов команд.

Говорят, что по отношению к ОП команда

`mov{b|w|l} операнд1, операнд2`

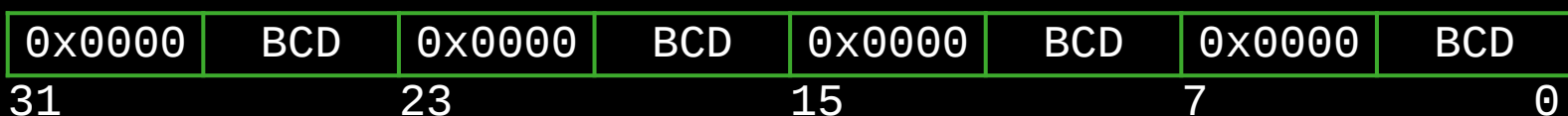
выполняет операцию ЧТЕНИЯ: ОП $\Rightarrow$ ЦП, если операнд1 находится в ОП, а операнд2 в регистре ЦП и ЗАПИСИ: ЦП $\Rightarrow$ ОП если операнд 1 находится в регистре ЦП а операнд2 в ОП. При чтении значение операнд1 не изменяется, при записи значение операнд2 заменяется на значение операнд1.

Упакованный тип данных предназначен для использования командами SIMD, в нем хранятся четыре 32 битовых слова в плавающей точкой.

Логические типы данных

- Целые знаковые и беззнаковые числа в байте, слове, двойном и четверном словах рассмотрены в предыдущих лекциях.
- Байтовые строка — последовательность байтов произвольной длины, используется для хранения кодов символов — есть их примеры в текстах задач task3 и 2to10.
- Битовое поле — специальный тип для битовых операций.
- Упакованные и неупакованные BCD данные.
- Адрес — 32 байта.
- Данные математического сопроцессора.
- Данные MMX, SSE и других расширений системы команд.

Неупакованные BCD числа.



Упакованные BCD числа.



Упакованный и неупакованный двоично-десятичные BCD (Binary Coded Decimal) типы были введены для выполнения арифметических операций непосредственно в десятичной системе счисления без перевода из двоичной. Десятичная цифра кодируется в четырех битах (полубайте, *nibble*), значения полубайта от 0xA до 0xF не используются

BCD представления легко получить из кодов ASCII цифр, где для символов цифр 0 - 9 зафиксированы коды 0x30h - 0x39h и их правый полубайт есть BCD код соответствующей цифры.

Для арифметических BCD-операций используются те же команды, что и для целых двоичных, но с применением до или после них специальных команд коррекции операндов или результатов.

Математические сопроцессоры имеют полный набор команд, позволяющих обрабатывать BCD числа.

Целочисленное переполнение

Определение.

Событие целочисленное переполнение возникает после выполнения арифметической операции если ее результат требует для записи в устройство хранения чисел больше разрядов, чем реализовано в этом устройстве.

Примеры.

1. Для двухразрядного десятичного калькулятора операция $+35 + 78 = 113$ приводит к переполнению через верхнюю границу, а $-45 - 63 = -108$ — через нижнюю.

2. Операция сложения байтов, интерпретируемых как беззнаковые целые:

$$\begin{array}{r} 1100\ 1111 = 0\text{xCF} \\ + \\ 1111\ 1100 = 0\textxFC} \\ \hline 11100\ 1011 = 0\text{x1CB} \end{array}$$

приводит к переполнению через верхнюю границу.

Естественно, что арифметические операции над знаковыми целыми также могут инициировать событие переполнения.

Наступление переполнения процессор фиксирует в регистре флагов.

NBNB. При переполнении результата операции над беззнаковыми целыми получает значение 1 флаг переноса CF (carry flag) — над знаковыми — флаг переполнения OF (overflow flag). Проверка факта переполнения выполняется командами коротких условных переходов jcc.

Флаг CF расположен в бите регистра флагов с номером 0 — флаг

OF — в бите с номером 11, как показано на рисунке.

OF	X	X	X	X	X	X	X	X	X	CF
11			8	7		4	3			0

Здесь представлены три младших полубайта регистра флагов, X — не интересующие нас биты. Легко видеть, что при беззнаковом переполнении значение правого полубайта будет нечетным, а при знаковом — значение левого полубайта будет больше или равно 0x8

Для проверки наступления переполнения после операций над беззнаковыми целыми нужно выполнить команду jс. Переход по адресу ее операнда выполнится если произошло переполнение. Переход по адресу операнда команды jпс выполнится, если переполнение не произошло.

Для проверки наступления переполнения после операций над знаковыми целыми нужно выполнить команду jо. Переход по адресу ее операнда выполнится если произошло переполнение. Переход по адресу операнда команды jпо выполнится, если переполнение не произошло.

Директивы ассемблера

Директивы предназначены для управления ассемблированием и размещения в исходном файле различной информации, например о выделении необходимой памяти, определения и размещения в объектном файле данных и их значений, описания секций, внешних имен и др. См. список директив по адресу:

Формат директивы:

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

NB. Параметры могут задаваться выражениями, в том числе адресными, т. е. адресное значение символьного имени, полученное при ассемблировании можно разместить как данное в секции данных. Имя директивы всегда начинается с символа точка.

Константы

Константы используются в командах, выражениях и как параметры директив.

Числовые целые константы 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 0х или 0Х;**
- 2 — начинаются с символов 0b или 0B.**

Строковые константы это символы, находящиеся между символами "двойная кавычка".

См. детали по адресу:

<https://kappa.cs.petrSU.ru/~chistyak/architecture/docs/gas/gas-3.html#ss3.6>

Пример «Директивы определения данных и обнаружения переполнения»

```

2          #      Переполнения 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
9
10        .data # секция данных, распределение памяти
11
12        #      Система счисления операнда
13
14        Diff_bases: .byte 65 # 10
15                   .byte 0101 # 8
16                   .byte 041 # 8
17                   .byte 0x41 # 16
18                   .byte 0X41 # 16
19
20        #      8      16      10
21        0005 E5040000 .long 02345, 0x4E5, 1253
22        E5040000
23        E5040000
24
25        #      Выражения в директивах
26
27        Arifm_Expression:
28
29        0011 4141 .byte 35+30, 20+45
30        0013 4141 .byte 70-5, 80-15
31
32        0015 61626320 Str: .ascii "abc ABC 123"
33        41424320
34        313233
35        0020 D094D0B2 Str1: .ascii "Две капли сверкнул, сверкнул"
36        D0B520D0
37        BAD0B0D0
38        BFD0BBDD0
39        B820D181
40        D0B2D0B5
41        D180D0BA
42        D0BDD183
43        D1822C20
44        D181D0B2
45        D0B5D180
46        D0BAD0BD
47        D183D182
48        0054 D0BDD0B0 Str2: .asciz "на дне!"
49        20D0B4D0
50        BDD0B521
51        00

```

```

31
32 0061 FF          B1:          .byte    255
33 0062 F0          B2:          .byte    240
34 0063 FF          Bm1:         .byte    -1
35 0064 01          Bp1:         .byte    +1
36 0065 80          B_zn_min:    .byte    -128
37 0066 7F          B_zn_max:    .byte    +127
38 0067 0F101F20     BBB:         .byte    15,16,31,32,63,64
38      3F40
39 006d FF00         S1:          .short   255
40 006f F000         S2:          .short   240
41 0071 FFFF         W_bzn_max:   .short   65535
42 0073 0080         W_zn_min:    .short   -32768
43 0075 FF7F         W_zn_max:    .short   +32767
44 0077 0F001000     SSS:         .short   15,16,31,32,63,64
44      1F002000
44      3F004000
45 0083 FFFFFFFFFF   L1:          .long    4294967295
46 0087 FFFFFFFFFF   I1:          .int     4294967295
47 008b 00000080     I1:          .int     -2147483648
48 008f FFFFFFFF7F   I2:          .int     +2147483647
49 0093 00000080     Q:          .quad    -2147483648
49      FFFFFFFFFF
50
51                  .text # секция команд
52
53                  .global _start # точка входа
54
55 0000 66A16D00      _start:          movw    S1, %ax # слово 255
55      0000
56 0006 6603056F          addw    S2, %ax # + слово 240 = 495
56      000000
57                                  # НЕТ переполнения
58
59 000d 29C0          sub     %eax,%eax
60 000f A0610000      movb    B1, %al # байт 255
60      00
61 0014 02056200      addb    B2, %al # + 240
61      0000
62 001a 7202          jc      UnsignedOverflowb
63 001c 90            nop
64 001d 90            nop
65
66                  UnsignedOverflowb:
67 001e 29C0          sub     %eax,%eax
68 0020 66A17100      movw    W_bzn_max,%ax      # max
68                                  #беззнак слово
68      0000
69 0026 6683C001      addw    $1,%ax      # +1 к нему
70 002a 7202          jc      UnsignedOverflow_w
71 002c 90            nop
72 002d 90            nop
73
74                  UnsignedOverflow_w:

```

```

75 002e 29C0          sub    %eax,%eax
76 0030 66A17500      movw   W_zn_max, %ax      # max
знаковое                                     #слово
76          0000
77 0036 6683C001      addw   $1, %ax      # +1 к нему
GAS LISTING d-t-over.S                      page 3

78 003a 7002          jo      SignedOverflow
79 003c 90            nop
80 003d 90            nop
81          SignedOverflow:
82 003e 29C0          sub    %eax, %eax
83 0040 66A17300      movw   W_zn_min, %ax      # min
знаковое                                     #слово
83          0000
84 0046 6683E801      subw   $1, %ax      # -1 от него
85 004a 7002          jo      SignOver
86 004c 90            nop
87 004d 90            nop
88          SignOver:
89
90          Finish # конец работы, возврат в ОС
(макро из файл?
90 004e BB000000      >  movl  $0,%ebx
90          00
90 0053 B8010000      >  movl  $1,%eax
90          00
90 0058 CD80          >  int   $0x80
91          .end    # последняя строка исходного
текста

```

GAS LISTING d-t-over.S page 4

DEFINED SYMBOLS

```

d-t-over.S:12      .data:0000000000000000 Diff_bases
d-t-over.S:23      .data:0000000000000001 Arifm_Expression
d-t-over.S:28      .data:0000000000000015 Str
d-t-over.S:29      .data:0000000000000020 Str1
d-t-over.S:30      .data:0000000000000054 Str2
d-t-over.S:32      .data:0000000000000061 B1
d-t-over.S:33      .data:0000000000000062 B2
d-t-over.S:34      .data:0000000000000063 Bm1
d-t-over.S:35      .data:0000000000000064 Bp1
d-t-over.S:36      .data:0000000000000065 B_zn_min
d-t-over.S:37      .data:0000000000000066 B_zn_max
d-t-over.S:38      .data:0000000000000067 BBB
d-t-over.S:39      .data:000000000000006d S1
d-t-over.S:40      .data:000000000000006f S2
d-t-over.S:41      .data:0000000000000071 W_bzn_max
d-t-over.S:42      .data:0000000000000073 W_zn_min
d-t-over.S:43      .data:0000000000000075 W_zn_max
d-t-over.S:44      .data:0000000000000077 SSS
d-t-over.S:45      .data:0000000000000083 L1
d-t-over.S:46      .data:0000000000000087 I1
d-t-over.S:47      .data:000000000000008b I1

```

d-t-over.S:48	.data:00000000000000008f	I2
d-t-over.S:49	.data:000000000000000093	Q
d-t-over.S:55	.text:000000000000000000	_start
d-t-over.S:66	.text:00000000000000001e	UnsignedOverflowb
d-t-over.S:74	.text:00000000000000002e	UnsignedOverflow_w
d-t-over.S:81	.text:00000000000000003e	SignedOverflow
d-t-over.S:88	.text:00000000000000004e	SignOver

NO UNDEFINED SYMBOLS