Введение в архитектуру ЭВМ

Лекция 3. Позиционные системы счисления. Кодировка символов. Машинная арифметика. Дополнительный код. Система команд. Программа `2to10`

План лекции

- **Позиционные системы счисления.**
  - **Алгоритм перевода из системы с основанием `p` в десятичную систему.**
  - **Перевод из десятичной системы в систему с основанием `p`.**
    - **Идея алгоритма**
    - **Алгоритм перевода**
  - **Переводы между системами с основаниями `2`, `8` и `16`.**
    - **Алгоритм перевода целого числа из двоичной системы в систему с основанием `8 (16)`.**
      - **Алгоритм перевода целого числа из системы с основанием `8 (16)` в двоичную систему.**
    - **Домашнее задание.**
- **Кодировка символов**
- **Машинная арифметика**
  - **Дополнительный код**
- **Обзор системы команд**
- **Программа перевода числа из двоичной системы счисления в десятичную, `2to10.S`**

Позиционные системы счисления.

В первой половине IX века персидский математик Мухаммед ибн Муса аль-Хорезми (англ. Muhammad ibn Musa al-Khwarizmi), основываясь на работах индийских математиков, написал работу, где изложил алгоритмы действий в десятичной системе счисления.

Оригинал ее не сохранился, но есть перевод XII века на латынь под названием «De numero Indorum» (Индийский счет), содержащий слова «Dixit Algorizmi» (Сказал Аль-Хорезми), откуда и произошел термин алгоритм.

Вес цифры зависит от ее позиции $525 = 5 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0$

Общая формула позиционной системы счисления.

$$Q = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \ldots + a_1 \cdot p^1 + a_0 \cdot p^0 \quad (1)$$

a_i **— цифры системы,** p **— ее основание.**

Рассмотрим системы с основаниями $2, 8, 16$.

p	цифры															
2	0	1														
8	0	1	2	3	4	5	6	7								
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
											10	11	12	13	14	15

Алгоритм перевода из системы с основанием p **в десятичную систему.**

1. Записать число по формуле (1).

2. Выполнить действия в десятичной системе.

Примеры.

$2 \blacktriangleright 10.\ 101 = 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 4 + 0 + 1 = 5$

$8 \blacktriangleright 10.\ 373 = 3\cdot 8^2 + 7\cdot 8^1 + 3\cdot 8^0 = 192 + 56 + 3 = 251$

$16 \blacktriangleright 10.\ B9C = 11\cdot 16^2 + 9\cdot 16^1 + 12\cdot 16^0 = 2816 + 144 + 12 = 2972$

Перевод из десятичной системы в систему с основанием p.

Идея адгоритма

Нужно найти цифры преставления числа в системе с основанием p**, зная его запись в десятичной системе. Это задача с неизвестным числом неизвестных. Заметим что если правую и левую части** (1) **поделить на** p**, выполняя деление в десятичной системе, то получим частные:**

$$Q/p = a_n\cdot p^{n-1} + a_{n-1}\cdot p^{n-2} + \ .\ .\ .a_1\cdot p^0$$

и a_0 в остатке (2)

Т.е деление дает МЛАДШУЮ цифру представления числа Q **в системе с основанием** p**. Из (2) видно, что если теперь значение** Q/p **поделить на** p **ещё раз, то в остатке получим** a_1 **— следующую цифру представления** Q **в системе с основание** p**. Легко видеть, что цифра** a_n **будет получена в остатке, когда частное будет равно нулю.**

Алгоритм перевода

1. Последовательно делить нацело в десятичной системе на p

подлежащее переводу число, фиксируя получаемые остатки от деления.

2. Прекратить деление когда частное получит значение ноль.

3. Полученные остатки есть последовательные цифры искомог представления, причем младшей из них является первый полученный остаток.

Примеры.

10→2. 30_{10}

30	2				
30	15	2			
0	14	7	2		
a0	1	6	3	2	
	a1	1	2	1	2
		a2	1	0	0
			a3	1	
				a4	

Результат: $30_{10} = 11110_2$

10→16. 2346_{10}

2346	16		
2336	146	16	
10 (A)	144	9	
a0	2	0	
	a1	9	
		a2	

Результат: $2346_{10} = 92A_{16}$

Переводы между системами с основаниями 2, 8 и 16.

Эти переводы не требуют арифметических вычислений, а выполняются символьными заменами из таблицы.

Триада	8-я цифра	16-я цифра	Тетрада
000	0	0	0000
001	1	1	0001
010	2	2	0010
011	3	3	0011
100	4	4	0100
101	5	5	0101
110	6	6	0110
111	7	7	0111
		8	1000
		9	1001
		000	1010
		000	1011
		000	1100
		000	1101
		000	1110
		000	1111

Алгоритм перевода целого числа из двоичной системы в систему с основанием 8 (16).

1. Разбить двоичные цифры, начиная справа на триады (тетрады) и левую неполную из них, если есть, дополнить нулями.

2. Триады (тетрады) заменить соответствующими восьмеричными (шестнадцатеричными) цифрами из таблицы.

Примеры.

$010\ 110\ 001\ 101\ 111\ 010 = 261572_8$

$010\ 110\ 001\ 101\ 111\ 010 = 0001\ 0110\ 0011\ 0111\ 1010 = 1637A_{16}$

Алгоритм перевода целого числа из системы с основанием 8

(16) в двоичную систему.

Заменить восьмеричные (шестнадцатеричные цифры) на соответствующие триады (тетрады) из таблицы.

Примеры.

$4735_8 = 100111011101$

$9AC5_{16} = 1001101011000101$

Домашнее задание.

Выполнить по 20 переводов в направлениях 2 ⇆ 10, 16 ⇆ 10, 8 ⇆ 10, 2 ⇆ 16, 2 ⇆ 8. Через 2-3 недели будет проводиться контрольная работа на владение алгоритмами переводов. Ее успешное выполнение обязательно для получения зачета/экзамена.

Особенности кодировки символов в текущей версии ОС Linux.

Используется версия Unicode — utf-8 (RFC - Request For Comments 3629)**, код символа занимает 8, 16, 24 или 32 бита.**

<u>NB</u>. Символы с кодами от 0 до 127 (7F) кода ASCII **(стр.184 книги) совпадают с однобайтовыми кодами** utf-8**. Т.о. в нашей версии ОС одно байтовыми являются коды служебных символов (**0A — LF**), цифр, прописных и строчных букв английского алфавита.**

Однобайтовые коды цифровых символов — от кода «0**» - 48 (**0x30**) до кода «**9**» - 57 (**0x39**).**

NB. Значение одноцифрового числа = Значение кода — 48 (0x30)

00	00	^@ NUL	16	10	^P DLE	32	20	␣	48	30	0	64	40	@
01	01	^A SOH	17	11	^Q DC1	33	21	!	49	31	1	65	41	A
02	02	^B STX	18	12	^R DC2	34	22	"	50	32	2	66	42	B
03	03	^C ETX	19	13	^S DC3	35	23	#	51	33	3	67	43	C
04	04	^D EOT	20	14	^T DC4	36	24	$	52	34	4	68	44	D
05	05	^E ENQ	21	15	^U NAK	37	25	%	53	35	5	69	45	E
06	06	^F ACK	22	16	^V SYN	38	26	&	54	36	6	70	46	F
07	07	^G BEL	23	17	^W ETB	39	27	'	55	37	7	71	47	G
08	08	^H BS	24	18	^X CAN	40	28	(	56	38	8	72	48	H
09	09	^I HT	25	19	^Y EM	41	29	)	57	39	9	73	49	I
10	0A	^J LF	26	1A	^Z SUB	42	2A	*	58	3A	:	74	4A	J
11	0B	^K VT	27	1B	^[ESC	43	2B	+	59	3B	;	75	4B	K
12	0C	^L FF	28	1C	^\ FS	44	2C	,	60	3C	<	76	4C	L
13	0D	^M CR	29	1D	^] GS	45	2D	-	61	3D	=	77	4D	M
14	0E	^N SO	30	1E	^^ RS	46	2E	.	62	3E	>	78	4E	N
15	0F	^O SI	31	1F	^_ US	47	2F	/	63	3F	?	79	4F	O

NB. Численные значения кодов цифровых символов удовлетворяют неравенству:

0x30 <= Код цифрового символа <= 0x39

Значения кодов `ascii` легко получить командой `man ascii`.

```
ASCII(7)                    Linux Programmer's Manual                    ASCII(7)

NAME
       ascii - ASCII character set encoded in octal, decimal, and
hexadecimal

DESCRIPTION
       ASCII  is the American Standard Code for Information Interchange. It
is a 7-bit code. Many 8-bit codes (e.g., ISO 8859-1) contain ASCII as their
lower half.  The international counterpart of ASCII is known as ISO 646-
IRV.

       The following table contains the 128 ASCII characters.

       C program '\X' escapes are noted.

       Oct   Dec   Hex   Char                        Oct   Dec   Hex   Char

       000   0     00    NUL '\0' (null character)   100   64    40    @
       001   1     01    SOH (start of heading)      101   65    41    A
       002   2     02    STX (start of text)         102   66    42    B
       003   3     03    ETX (end of text)           103   67    43    C
       004   4     04    EOT (end of transmission)   104   68    44    D
       005   5     05    ENQ (enquiry)               105   69    45    E
       006   6     06    ACK (acknowledge)           106   70    46    F
       007   7     07    BEL '\a' (bell)             107   71    47    G
       010   8     08    BS  '\b' (backspace)        110   72    48    H
       011   9     09    HT  '\t' (horizontal tab)   111   73    49    I
       012   10    0A    LF  '\n' (new line)         112   74    4A    J
       013   11    0B    VT  '\v' (vertical tab)     113   75    4B    K
```

014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M

. . .

040	32	20	SPACE	140	96	60	
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

NOTES

History

An ascii manual page appeared in Version 7 of AT&T UNIX.

On older terminals, the underscore code is displayed as a left arrow, called backarrow, the caret is displayed as an up-arrow and the vertical bar has a hole in the middle.

Uppercase and lowercase characters differ by just one bit and the ASCII character 2 differs from the double quote by just one bit, too. That made it much easier to encode characters mechanically or with a non-microcontroller-based electronic keyboard and that pairing was found on old teletypes.

The ASCII standard was published by the United States of America Standards Institute (USASI) in 1968.

SEE ALSO

```
      charsets(7),  iso_8859-1(7),  iso_8859-10(7),  iso_8859-11(7),
iso_8859-13(7), iso_8859-14(7), iso_8859-15(7), iso_8859-16(7),
iso_8859-2(7),  iso_8859-3(7), iso_8859-4(7),  iso_8859-5(7),
iso_8859-6(7),  iso_8859-7(7),  iso_8859-8(7), iso_8859-9(7), utf-8(7)

COLOPHON
      This page is part of release 4.16 of the Linux man-pages project.  A
descrip-ion of the project, information about reporting bugs, and the
latest version of this page, can be found at
https://www.kernel.org/doc/man-pages/.

Linux                              2016-10-08
ASCII(7)
```

Машинная арифметика

В каждой системе счисления есть таблица сложения. Для двоичной системы она имеет вид:

`0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0` **и** `1` **переноса.**

Пример 1.

```
  1 0 0 1 1 1 0 1
+
  0 1 0 1 0 1 0 1
---------------------------
  1 1 1 1 0 0 1 0
```

Пример 2. Получение таблиц для переводов `2 <-> 8, 16.`

```
  0 0 0 0
+
  0 0 0 1
--------------
  0 0 0 1
+
  0 0 0 1
--------------
  0 0 1 0
+
  0 0 0 1
--------------
```

```
  0 0 1 1
+
  0 0 0 1
-------------
  0 1 0 0
```

Продолжите самостоятельно.

Дополнительный код

В архитектуре IA-32 программист может интерпретировать байты, слова и двойные слова как беззнаковые или знаковые целые. При *беззнаковой интерпретации* все биты рассматриваются как цифры двоичного числа, что определяет следующие диапазоны:

Байт: 00 — FF **или** 0 — 255.

Слово: 0000 — FFFF **или** 0 — 65535.

Двойное слово: 00000000 — FFFFFFFF **или** 0 — 4294967295.

При *знаковой интерпретации* старший бит трактуется как индикатор знака, его значение 1 указывает на отрицательное число, 0 — на положительное, что определяет следующие диапазоны:

Байт: 0x80 — 7F **или** -128 — +127.

Слово: 0x8000 — 7FFF **или** -32768 — +32767.

Двойное слово: 80000000 — 7FFFFFFF **или** -2147483648 — +2147483648.

NBNB. Значение отрицательного числа представляется в дополнительном коде, обеспечивавшем одинаковые алгоритмы операций сложения и вычитания для знаковых и беззнаковых целых.

Операции умножения и деления для беззнаковых целых выполняются командами `mul{s}, div{s}`**, для знаковых — командами** `imul{s}, idiv{s}` **соответственно, где** `s` **— суффикс размера операндов** `{b|w|l}`**.**

Алгоритм получение дополнительного кода.

1. Инвертировать биты числа.

2. К результату прибавить единицу.

Пример.

Получим представление в байте отрицательного числа `-5` **из представления числа** `+5`**.**

```
Исходное число:  0 0 0 0 0 1 0 1
        Шаг 1:   1 1 1 1 1 0 1 0
        Шаг 2:   +
                 0 0 0 0 0 0 0 1
                 -----------------------
                 1 1 1 1 1 0 1 1
```

Убедиться самостоятельно, что применение этого алгоритма к полученному двоичному представлению числа `-5` **дает число** `+5`**.**

Команда `neg{b|w|l} операнд` **записывает в** `операнд` **его дополнительный код.**

Обзор системы команд

MOV Пересылка данных
Move data

Переп	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).

ADD Сложение
Add

Перепо Р	Напр D	Прер I	Шаг T	Знак S Р	Нуль Z Р	Всп A Р	Четн Р Р	Перенос С Р

Код	Команда	Краткое описание
04 ib	ADD AL,imm8	байт 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).

Jcc Условный короткий переход
Jump if condition is met

Перепо	Напр D	Прер I	Шаг T	Знак S	Нуль Z	Всп A	Четн P	Перенос 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 (E)CX		
E3 cb	JCXZ rel8	регистр CX содержит 0
	JECXZ rel8	регистр ECX содержит 0

Программа перевода числа из двоичной системы счисления в десятичную, 2to10.S

```
.include "my-macro"  # подключение файла с макроопределениями

/*
Получение 10-х цифр двоичного числа и вывод их на терминал
```

```
 */

.data   # секция данных, распределение памяти

n:  .short 2345     # число для перевода 2->10

digits_of_n:    .ascii "□□□□□" # коды символов 10-х цифр - 5
пробелов

    .align 16   # выравниваем на 16 битовую границу

length: .long 0         # количество 10 цифр в числе

ten:    .short 10       # делитель


.text # секция команд процесора

.global _start  # точка входа - глобальная метка

_start:
    movl $0, %ebx       # счетчик делений
    movw n, %ax         # готовим деление
    movl $4,%esi        # начальное значение индекса - на старший
                        # байт строки digits_of_n

nextdigit:

#        Т.к. переводим 16 бит слово, то работаем с 16 бит рег

        movw  $0, %dx           # готовим деление в цикле
        idivw ten               # делим объединенные регистры
dx:ax на 10
                                # частное в  ax, остаток в dx

#       Превращаем значение остатка в код цифры

      addw $48,%dx

#   !!! пишем код цифры остатка в байт строки 10digits-of-n
#   на который указывает  %esi

    leal digits_of_n(%esi), %ecx # демонстрация исполнительного
```

```
адреса

#    команда leal вычисляет ИА и посылает его в регистр 2 операнда
#    теперь в %ecx - адрес байта куда запишется код цифры
#    команда присутствует только для учебных целей

        movb %dl,digits_of_n(%esi)
```

Лекционный комментарий.

1. Алгоритм перевода получает коды цифр десятичного представления от младших к старшим. Т.е. их надо размещать в байты строки digits_of_n начиная с адреса digits_of_n + 4, т.е

NBNB значение %esi должно меняться в цикле от 4 до 0 (если получаем все пять цифр)!

Адрес байта digits_of_n+	0	1	2	3	4
Значение %esi	0	1	2	3	4
Цифра разряда	a4	a3	a2	a1	a0

```
#    готовим обработку след. цифры

        incl %ebx               # счетчик делений + 1
        decl %esi           # устанавливаем индекс на байт для
                            # след. кода цифры
        cmpl $0, %eax           # частное = 0 ?
        jg   nextdigit          # НЕТ, продолжаем
        movl %ebx, length       # ДА, сохраняем результат

#    Вывод десятичного представления, сис. вызов write

        movl $4, %eax           # номер сист. вызова write
        movl $1, %ebx           # 1 - дескриптор стандартного вывода
        movl $digits_of_n, %ecx # адрес памяти с выводимыми символами
        movl $5, %edx           # количество байтов для вывода
        int $0x80               # выполнить системный вызов

         Finish                 # конец работы, возврат в ОС
                                # (макро из файла my-macro)
```

```
.end    # последняя строка исходного текста
```