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 2. Регистры. Позиционные системы счисления. Кодировка символов. Машинная арифметика.

План лекции

Регистры центрального процессора.

Ссылка на видеобзор регистров находится на веб страница дисциплины рядом со ссылкой на текст лекции. Описание регистров — на стр. 150 учебного пособия:

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

Специфика использования регистров совпадает с этой спецификой для архитектуры 8086 и дана на стр. 49 того же пособия.

Позиционные системы счисления.

Определение позиционной 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.

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

Представление двоичного числа с помощью 16-х цифр

— Веб-разработка — цвета в CSS, #FF5733 — оранжевый.

— Криптография — представления хеш-значений.

— Сетевые технологии — IP-адреса, 2001:0db8:85a3 — в протоколе IPv6.

— Системное программирование — ошибки и состояния — 0×0000FFFF указывает на тип сбоя.

— Адреса памяти и команды, например в файле листинга

- `57 0000 29090000 n: .long 2345 # int n = 2345;` число

- `73 0006 A1000000  movl n, %eax  # готовим к команде деления`

— Языки C, C++, Python и т.п. — работа с битами.

Это особенно удобно для восьми битовых байтов (октетов) в которых каждой двоичной тетраде — полубайту `(nibble, nybble)` взаимно однозначно соответствуют шестнадцатеричная цифра. Так, 32 разрядное двоичное число — с весьма длинной и непонятной записью:

`10010011111010100000110100111111`

`(тетрады – 1001 0011 1110 1010 0000 1101 0011 1111)`

будет представлено в четырех байтах (восьми полубайтах) в естественном порядке как `0x93EA0D3F` (`0x` указывает, что число представлено в шестнадцатеричной системе).

Порядок байтов в ОП при представлении целых чисел

В текстах запись числа `0x93EA0D3F` соответствует принятому порядку при котором степени основания разрядов возрастают справа налево, т. е. запись этого числа по формуле позиционной системы счисления с заменой шестнадцатеричных цифр на десятичные имеет вид:

$$Q = 9\cdot16^7+3\cdot16^6+14\cdot16^5+10\cdot16^4+0\cdot16^3+13\cdot16^2+3\cdot16^1+15\cdot16^0$$

Однако при хранении числа в ОП возникает проблема. Если число не может быть представлено одним байтом, нужно зафиксировать в архитектуре системы, в каком порядке байты будут храниться в ОП или передаваться по линиям связи. Принято два таких порядка.

От младшего к старшему - `Little endian` (остроконечный) - байты младших цифр имеет меньший адрес. Число `93EA0D3F` в этом порядке будет представлено в ОП так:

Адрес ОП	a+0	a+1	a+2	a+3
Байты числа	3F	0D	EA	93

Преимущество. Адрес двойного слова, младшего слова и младшего байта совпадают и равны, для этого примера - a+0. Для порядке Big endian это не так.

От старшего к младшему - Big endian (тупоконечный) - байты младших цифр имеет больший адрес. Число 93EA0D3F в этом порядке будет представлено так:

Адрес ОП	a+0	a+1	a+2	a+3
Байты числа	93	EA	0D	3F

Порядок байтов выбирает конструктор архитектуры ЭВМ.

Порядок Little endian принят а архитектурах Intel, Big endian - в сетях TCP/IP (network byte order), архитектурах Motorola, соответствует порядку десятичной системы счисления.

Алгоритмы переводов чисел между представлениями их в позиционных системах счисления с разными основаниями

Алгоритм перевода из системы с основанием p в десятичную систему.

1. Записать число по формуле (1).
2. Выполнить действия в десятичной системе.

Примеры.

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

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

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

Алгоритм перевода из десятичной системы в систему с произвольным основанием p.

Идея алгоритма

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:

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

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 искомого представления, причем младшей из них является первый полученный остаток.

Примеры.

$10 \rightarrow 2.\ 30_{10}$

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
		A	1010
		B	1011
		C	1100

		D	1101
		E	1110
		F	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 $2 \leftrightarrows 10$, $16 \leftrightarrows 10$, $8 \leftrightarrows 10$,

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 стандарта кодирования символов Unicode – utf-8 (RFC - Request For Comments 3629), код символа занимает 8, 16, 24 или 32 бита.

NB. 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.

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/.

Машинная арифметика

Таблица сложения двоичной системы счисления

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.

Беззнаковый и знаковый типы целых чисел

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 – 7FFFFFF`F или `-2147483648 – +2147483648`.

NBNB. Значения знаковых целых чисел представляются 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.

Переполнение при арифметических операциях над целыми

Определение.

Событие 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:

```
  1100 1111 = 0xCF
     +
  1111 1100 = 0xFC
  ----------------
 11100 1011= 0x1CB
```

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	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 jc. Переход по адресу ее операнда выполнится если произошло переполнение. Переход по адресу операнда команды jnc выполнится, если переполнение не произошло.

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

Выводы

- Для хранения в ОП байтов целых чисел в архитектуре IA-32 принят порядок от младшего к старшему — Little endian (остроконечный).
- Профессионалу в сфере IT необходимо владеть алгоритмами переводов чисел между представлениями их в позиционных системах счисления с разными основаниями.

- В текущей версии нашей ОС используется версия стандарта кодирования символов Unicode – utf-8 (RFC - Request For Comments 3629), код символа занимает 8, 16, 24 или 32 бита.
- Символы с кодами от 0 до 127 (7F) кода ASCII (стр.184 учебного пособия) совпадают с однобайтовыми кодами utf-8.
- Значения кодов цифровых символов однобайтовые и находятся в диапазоне от кода «0» - 48 (0x30) до кода «9» - 57 (0x39).
- Содержимое байтов, слов и двойных слов в ОП и регистрах можно интерпретировать как знаковые или беззнаковые целые числа. Они отличаются трактовкой старшего бита, диапазонами возможных значений и применяемыми операциями умножения и деления.
- Значение отрицательного числа представляется в дополнительном коде, обеспечивавшем одинаковые алгоритмы операций сложения и вычитания для знаковых и беззнаковых целых.
- Возникновение переполнения разрядной сетки при арифметических операциях отражается в битах регистра флагов.
- При переполнении результата операции над беззнаковыми целыми получает значение 1 флаг переноса CF (carry flag, бит 0 регистра флагов) — над знаковыми — флаг переполнения OF (overflow flag, бит 11 регистра флагов).
- Наступления переполнения после операций над беззнаковыми целыми проверяется командой jc, над знаковыми — командой jo.