Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

А. В. Ковалев, Н. В. Самовендюк, Д. А. Литвинов

ОРГАНИЗАЦИЯ И ФУНКЦИОНИРОВАНИЕ ЭВМ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ
по одноименному курсу для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения

Гомель 2009

УДК 004.2(075.8)
ББК 32.973я73
К56

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 2 от 10.12.2007 г.*)

Рецензент: доц. каф. «Промышленная электроника» ГГТУ им. П. О. Сухого канд. техн. наук
Э. М. Виноградов

Ковалев, А. В.

К56 Организация и функционирование ЭВМ : лаборатор. практикум по одноим. курсу для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. формы обучения / А. В. Ковалев, Н. В. Самовендюк, Д. А. Литвинов. – Гомель : ГГТУ им. П. О. Сухого, 2009. – 45 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

Содержит описание различных систем счисления, способы перевода чисел в различные системы счисления. Рассмотрены основные операторы языка ассемблер, структура программы, директивы определения данных, команды адресации и пересылки данных.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения.

УДК 004.2(075.8)
ББК 32.973я73

Лабораторная работа №1
Системы счисления

Основные понятия и определения

Система счисления - это способ представления чисел с помощью некоторого набора символов, называемых цифрами, а также совокупность приемов и правил, позволяющих однозначно представлять числовую информацию.

Все системы счисления можно разделить на **позиционные** и **непозиционные**.

В непозиционных системах счисления вес цифры (т. е. тот вклад, который она вносит в значение числа) не зависит от ее позиции в записи числа.

Примером непозиционной системы счисления является римская система. В этой системе в качестве цифр используются латинские буквы:

I V X L C D M и т.д. - римские цифры;

1 5 10 50 100 500 1000 - десятичные эквиваленты римским цифрам.

Величина числа определяется суммой или разностью цифр в числе. Если меньшая цифра стоит слева от большей, то она вычитается, если справа — прибавляется. Например:

LXVI = L + X + V + I = 50 + 10 + 5 + 1 = 56
DCVL = D + C + (X – V) =500 + 100 + (50–5) = 645
CDXLIX = (D - C)+(L-X) + (X - I) = (500-100) + (50-10) + (10-1) = 449

К недостаткам таких систем относятся наличие большого количества знаков и сложность выполнения арифметических операций.

В позиционных системах счисления вес каждой цифры изменяется в зависимости от ее положения (позиции) в последовательности цифр, изображающих число. Количественная оценка записанного числа в такой системе счисления определяется как сумма произведений значения цифр, составляющих запись числа, умноженных на вес позиции, в которой располагается цифра.

Примером позиционной системы счисления является широко используемая десятичная система счисления. Например, количественная оценка десятичного числа 777 определяется как 7 · 100 + 7 · 10 + 7, где 100, 10, 1 - соответственно веса третьего, второго и первого разрядов записи оцениваемого числа.

Любая позиционная система счисления характеризуется своим **основанием.**

Основание позиционной системы счисления — это количество различных цифр, используемых для изображения чисел в данной системе счисления.

В десятичной системе используются десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Основанием этой системы является число десять.

За основание системы счисления можно принять любое натуральное число — два, три, четыре и т.д. Следовательно, возможно бесчисленное множество позиционных систем: двоичная, троичная, четверичная и т.д.

Для записи чисел в позиционной системе с основанием q необходимо иметь **алфавит** из n цифр. В качестве цифр используются обозначение соответствующих цифр десятичной системы счисления 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, а в случае, когда десятичных цифр «не хватает» (для систем счисления с основанием q, большим чем 10), для цифр, превышающих 9, вводятся дополнительные обозначения, например, для q =16 добавляют буквы A, B, C, D, E, F, которые соответствуют шестнадцатеричным цифрам, десятичные эквиваленты которых равны соответственно 10, 11, 12, 13, 14, 15. В таблице 1 приведены примеры нескольких систем:

Таблица 1

Основание	Название	Алфавит
n=2	двоичная	0 1
n=3	троичная	0 1 2
n=8	восьмеричная	0 1 2 3 4 5 6 7
n=16	шестнадцатеричная	0 1 2 3 4 5 6 7 8 9 A B C D E F

Чтобы указать основание системы, к которой относится число, необходимо приписать нижний индекс к числу. Например:

10101101_2, 1221_3, 747_8, $3BE5_{16}$

Запись чисел в каждой из систем счисления с основанием q означает сокращенную запись выражения

$$\mathbf{a_{n-1}\, q^{n-1} + a_{n-2}\, q^{n-2} + ... + a_1\, q^1 + a_0\, q^0 + a_{-1}\, q^{-1} + ... + a_{-m}\, q^{-m}}, \qquad \mathbf{(1)}$$

где $\mathbf{a_i}$ — цифры системы счисления; **n** и **m** — число целых и дробных разрядов, соответственно.

Такая форма записи числа называется развернутой или расширенной. Пользуясь этой формулой можно легко перевести число из системы счисления с любым основанием в десятеричную.

Пример:

$\mathbf{101101_2} = 1\cdot2^5 + 0\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 =$
$= 1\cdot32 + 0 + 1\cdot8 + 1\cdot4 + 0 + 1\cdot1 = 45_{10};$
$\mathbf{703_8} = 7\cdot8^2 + 0\cdot8^1 + 3\cdot8^0 = 7\cdot64 + 0 + 3\cdot1 = 451_{10};$
$\mathbf{B2E4_{16}} = 11\cdot16^3 + 2\cdot16^2 + 14\cdot16^1 + 4\cdot16^0 = 11\cdot4096 + 2\cdot256 + 14\cdot16 + 4\cdot1 = 45796_{10}.$

Задание к лабораторной работе

Необходимо написать программу, которая на входе получает две строки из ASCII-символов. В первой строке указывается основание системы счисления, во второй число. На выходе должно выводиться десятичное число, соответствующее введенному. В программе предусмотреть проверку соответствия цифр введенного числа алфавиту заданной системы счисления.

Основание системы счисления выбирается согласно порядковому номеру студента в журнале + 3. Вводимое число соответствует четырем первым цифрам зачетной книжки.

Требование к отчету

В отчете должны быть отображены следующие пункты:

1. Задание.
2. Теория (при необходимости).
3. Таблица соответствия переменных.
4. Схема алгоритма.
5. Листинг программы.
6. Тесты.
7. Результат выполнения программы.

Контрольные вопросы

1. Какое количество обозначает цифра 8 в десятичных числах 4587, 8652, 178, 894?
2. Выпишите алфавиты в 5-ричной, 7-ричной, 12-ричной системах счисления.
3. Запишите в развернутой форме числа: 3412_{10}, 3412_5, 3412_8, 3412_{12}.
4. Запишите в десятичной системе счисления числа: 421_5, 421_7, 421_9, 421_{16}.
5. Какое минимальное основание должна иметь система счисления, если в ней могут быть записаны числа: 101, 2101, 120, 1021.
6. Какое минимальное основание должна иметь система счисления, если в ней могут быть записаны числа: 423, 671, 56, 1721.
7. Запишите десятичный эквивалент числа 101, если считать его написанным во всех системах счисления – от двоичной до семеричной включительно.
8. Представить двоичные числа 1110101, 1001011, 10101011 в десятеричной системе счисления.
9. Представить восьмеричные числа 572, 765, 1274 в десятеричной системе счисления.
10. Представить шестнадцатеричные числа 5A2, FE5, D2E41 в десятеричной системе счисления.

Лабораторная работа №2
Перевод целых десятичных чисел в другие системы счисления

Перевод целых десятичных чисел в другие системы счисления осуществляется на основании следующего алгоритма:

- основание новой системы счисления необходимо выразить в десятичной системе счисления и все последующие действия производить в десятичной системе счисления;
- последовательно выполнять деление заданного числа и получаемых неполных частных на основание новой системы счисления до тех пор, пока не получится неполное частное, меньшее основания новой системы счисления;
- полученные остатки, являющиеся цифрами числа в новой системе счисления, необходимо привести в соответствие с алфавитом новой системы счисления;
- составить число в новой системе счисления, записывая его начиная с последнего остатка.

Пример:

а) перевести число 98_{10} в пятеричную систему

98	5	
95	19	5
	15	3

$a_0=3$ $a_1=4$ $a_2=3$

Таким образом: $98_{10} = 343_5$

б) перевести число 412_{10} в шестнадцатеричную систему

412	16	
400	25	16
	16	1

$a_0=12$ $a_1=9$ $a_2=1$

Таким образом: $412_{10} = 19C_{16}$. Напомним, что $12_{10} = C_{16}$.

Задание к лабораторной работе

Необходимо написать программу, которая на входе получает две строки из ASCII-символов. В первой строке задается десятичное число, во второй указывается основание системы счисления, в которую необходимо перевести заданное число. На выходе должно выводиться число в новой системе счисления в виде строки из ASCII-символов.

Основание системы счисления выбирается согласно порядковому номеру студента в журнале + 3. Вводимое число соответствует четырем первым цифрам зачетной книжки.

Требование к отчету

В отчете должны быть отображены следующие пункты:

1. Задание.
2. Теория (при необходимости).
3. Таблица соответствия переменных.
4. Схема алгоритма.
5. Листинг программы.
6. Тесты.
7. Результат выполнения программы.

Контрольные вопросы

1. Представить десятичные числа 324 и 628 в двоичной системе счисления.
2. Представить десятичные числа 462 и 748 в троичной системе счисления.
3. Представить десятичные числа 194 и 751 в пятеричной системе счисления.
4. Представить десятичные числа 241 и 967 в семеричной системе счисления.
5. Представить десятичные числа 524 и 876 в восьмеричной системе счисления.
6. Представить десятичные числа 1324 и 784 в двенадцатеричной системе счисления.
7. Представить десятичные числа 927 и 1688 в шестнадцатеричной системе счисления.

Лабораторная работа №3
Перевод двоичных чисел в восьмеричную и шестнадцатеричную системы счисления и обратные преобразования

Системы счисления, используемые в ЭВМ (с основанием 2^n)

В аппаратной основе ЭВМ лежат двухпозиционные элементы, которые могут находиться только в двух состояниях; одно из них обозначается 0, а другое - 1. Поэтому основной системой счисления применяемой в ЭВМ является двоичная система, которая обладает рядом преимуществ перед другими системами:

- для ее реализации нужны **технические устройства с двумя устойчивыми состояниями** (есть ток — нет тока, намагничен — не намагничен и т.п.), а не, например, с десятью, — как в десятичной;
- представление информации посредством только двух состояний **надежно** и **помехоустойчиво**;
- возможно **применение аппарата булевой алгебры** для выполнения логических преобразований информации;
- двоичная арифметика намного проще десятичной.

Недостаток двоичной системы — **быстрый рост числа разрядов**, необходимых для записи чисел.

Перевод чисел из десятичной системы в двоичную и наоборот выполняет машина. Однако, чтобы профессионально использовать компьютер, следует научиться понимать слово машины. Для этого и разработаны восьмеричная и шестнадцатеричная системы. Числа в этих системах читаются почти так же легко, как десятичные, требуют соответственно в три (восьмеричная) и в четыре (шестнадцатеричная) раза меньше разрядов, чем в двоичной системе (ведь числа 8 и 16 — соответственно, третья и четвертая степени числа 2).

Для того чтобы целое двоичное число записать в системе с основанием $q=2^n$ (4, 8, 16 и т.д.), необходимо:

- заданное двоичное число разбить справа налево на группы по n цифр в каждой;
- если в последней левой группе окажется меньше n разрядов, то ее надо дополнить слева нулями до нужного числа разрядов;
- рассмотреть каждую группу как n-разрядное двоичное число и записать ее соответствующей цифрой в системе счисления с основанием $q=2^n$.

Для того чтобы произвольное число, записанное в системе счисления с основанием $q=2^n$, перевести в двоичную систему счисления,

нужно каждую цифру этого числа заменить ее n-разрядным эквивалентом в двоичной системе счисления. Для перевода двоичного числа в восьмеричную и шестнадцатеричную системы счисления число разбивается на группы в 3 и 4 разряда соответственно. Для перевода удобно пользоваться двоично-восьмеричной и двоично-шестнадцатеричной таблицами.

Таблица 3 Двоично-восьмеричная

8	**2**
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Таблица 4 Двоично-шестнадцатеричная

16	**2**
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Пример:

1. Перевести число $5A2B_{16}$ двоичную систему.

Для перевода воспользуемся таблицей 4 и заменим каждую цифру в шестнадцатеричном числе на соответствующую ей в таблице четверку двоичных знаков. Получается:

0101 1010 0010 1011.

Лишние нули слева отбрасываются, так как они не влияют на значения целого числа. Таким образом:

$5A2B_{16} = 101101000101011_2$.

2. Перевести двоичное число 11011011101100101_2 в шестнадцатеричную систему.

Разделим заданное число на группы по четыре цифры, начиная справа. Левую группу дополняем нулями.

0001 1011 0111 0110 0101

На основании данных из таблицы 4, заменяем каждую двоичную группу на соответствую шестнадцатеричную цифру.

1 B 7 6 5

Следовательно:

$11011011101100101_2 = 1B765_{16}$

3. Перевести число 723_8 двоичную систему.

Для перевода воспользуемся таблицей 3 и заменим каждую цифру в восьмеричном числе на соответствующую ей в таблице тройку двоичных знаков. Получается:

111 010 011.

Таким образом:

$723_8 = 111010011_2$.

4. Перевести двоичное число 1011101100101_2 в восьмеричную систему.

Разделим заданное число на группы по три цифры, начиная справа. Левую группу дополняем нулями.

001 011 101 100 101

На основании данных из таблицы 3, заменяем каждую двоичную группу на соответствующую восьмеричную цифру.

1 3 5 4 5

Следовательно:

$1011101100101_2 = 13545_8$.

Задание к лабораторной работе

1. Необходимо написать программу, которая на входе получает строку из ASCII-символов, состоящую из 0 и 1, соответствующей записи заданного двоичного числа. На выходе должно выводиться 2 строки ASCII-символов, соответствующие шестнадцатеричному и двоичному представлению введенного числа. В программе предусмотреть проверку ввода некорректных данных.

Вводимое число соответствует четырем первым цифрам зачетной книжки, переведенное в двоичное число с помощью калькулятора.

2. Необходимо написать программу, которая на входе получает строку из ASCII-символов, соответствующей записи числа в заданной системе счисления (для нечетных вариантов – восьмеричная, для четных – шестнадцатеричная). На выходе должна выводиться строка ASCII-символов, состоящую из 0 и 1, соответствующая записи эквивалентного двоичного числа. В программе предусмотреть проверку ввода некорректных данных.

Вводимое число соответствует четырем первым цифрам зачетной книжки, переведенное в заданную систему счисления с помощью калькулятора.

Требование к отчету

В отчете должны быть отображены следующие пункты:

1. Задание.
2. Теория (при необходимости).
3. Таблица соответствия переменных.
4. Схема алгоритма.
5. Листинг программы.
6. Тесты.
7. Результат выполнения программы.

Контрольные вопросы

1. Перевести двоичные числа 10011101110, 1011011101 в восьмеричную систему счисления.
2. Перевести двоичные числа 10110101110, 1011110101 в восьмеричную систему счисления.
3. Перевести двоичные числа 10011101110, 1011011101 в шестнадцатеричную систему счисления.

4. Перевести двоичные числа 10110101110, 1011110101 в шестнадцатеричную систему счисления.
5. Перевести восьмеричные числа 472, 1435 в двоичную систему счисления.
6. Перевести восьмеричные числа 726, 4162 в двоичную систему счисления.
7. Перевести шестнадцатеричные числа 4B7A, C43F в двоичную систему счисления.
8. Перевести шестнадцатеричные числа D719, 3EF6 в двоичную систему счисления.
9. Перевести числа 685, 4372 из восьмеричной системы счисления в шестнадцатеричную систему счисления.
10. Перевести числа D85B, 7EC9 из шестнадцатеричной восьмеричной системы счисления в восьмеричную систему счисления.

Лабораторная работа №4
Арифметические операции в позиционных системах счисления

Любая позиционная система счисления определяется основанием системы, алфавитом и правилами выполнения арифметических операций. Правила выполнения арифметических операций в десятичной системе счисления применимы и к другим позиционным системам счисления. Однако при выполнении операций сложения и умножения необходимо пользоваться таблицами сложения и умножения для конкретной системы счисления.

Например, правила выполнения операций сложения и умножения в пятеричной системе счисления задаются в виде таблицы 5.

Таблица 5 Сложение и умножение в пятеричной системе счисления

Сложение						Вычитание					
+	0	1	2	3	4	*	0	1	2	3	4
0	0	1	2	3	4	0	0	0	0	0	0
1	1	2	3	4	10	1	0	1	2	3	4
2	2	3	4	10	11	2	0	2	4	11	13
3	3	4	10	11	12	3	0	3	11	14	22
4	4	10	11	12	13	4	0	4	13	22	31

Таблицы сложения и умножения легко составить, используя правило счета.

Примеры:

1. Сложить числа 15 и 6 в различных системах счисления

При сложении цифры суммируются по разрядам, и если при этом возникает избыток, то он переносится влево.

Двоичная система: $1111_2 + 110_2$

```
  1111
+  110
-----
 10101
    |  1+0=1
    |  1+1=2=2+0
    |  1+1+1=3=2+1
    |  1+1=2=2+0
```

Пятеричная система: $30_5 + 11_5$

```
  30
+ 11
----
 41
  |  0+1=1
  |  3+1=4
```

Восьмеричная система: $17_8 + 6_8$

$$\begin{array}{r} 17 \\ +\ 6 \\ \hline 25 \end{array}$$

7+6=13=8+5

1+1=2

Шестнадцатеричная система: $F_{16} + 6_{16}$

$$\begin{array}{r} F \\ +\ 6 \\ \hline 15 \end{array}$$

15+6=21=16+5

Ответ: $15+6=21_{10}=10101_2=41_5=25_8=15_{16}$

Проверка. Преобразуем полученные суммы к десятичному виду:
$10101_2=1\cdot2^4+1\cdot2^2+1\cdot2^0=16+4+1=21$,
$41_5=4\cdot5^1+1\cdot5^0=20+1=21$,
$25_8=2\cdot8^1+5\cdot8^0=16+5=21$,
$15_{16}=1\cdot16^1+5\cdot16^0=21$.

2. Вычесть число 6 из 15 в различных системах счисления

Если в процессе операции большее число вычитается из большего, то осуществляется заем из старшего разряда, равный основанию системы счисления.

Двоичная система: $1111_2 - 110_2$

$$\begin{array}{r} 1111 \\ -\ 110 \\ \hline 1001 \end{array}$$

1-0=1

1-1=0

1-1=0

1-0=1

Пятеричная система: $30_5 - 11_5$

$$\begin{array}{r} 30 \\ -\ 11 \\ \hline 14 \end{array}$$

Заем из старшего разряда

5-1=4

2-1=1

Е еричная система: $F_{16} - 6_{16}$

$$\begin{array}{r} 17 \\ +\ 6 \\ \hline 11 \end{array}$$

7-6=1

1-0=1

$$\begin{array}{r} F \\ +\ 6 \\ \hline 9 \end{array}$$

15-6=9

ммы к десятичному виду:

$1001_2=1\cdot2^3+1\cdot2^0=8+1=9,$
$14_5=1\cdot5^1+4\cdot5^0=5+4=9,$
$11_8=1\cdot8^1+1\cdot8^0=8+1=9,$
$9_{16}=9\cdot16^0=9.$

3. Умножить числа 15 и 6 в различных системах счисления

Выполняя умножение многозначных чисел в различных позиционных системах счисления, можно использовать обычный алгоритм перемножения чисел в столбик, но при этом результаты перемножения и сложения однозначных чисел необходимо заимствовать из соответствующих рассматриваемой системе таблиц умножения и сложения.

Двоичная система: $1111_2 \cdot 110_2$

```
  1111
x  110
------
  1111
+ 1111
------
1011010
```

Пятеричная система: $30_5 \cdot 11_5$

```
  11
x 30
----
330
```

Восьмеричная система: $17_8 \cdot 6_8$

```
  17
x  6
----
 132
  |  7x6=42=5x8+2
  |  1x6+5=11=8+3
```

Шестнадцатеричная система: $F_{16} \cdot 6_{16}$

```
  F
x 6
---
 50
  |  15x6=90=5x16+0
```

Ответ: $15x6=90_{10}=1011010_2=330_5=132_8=50_{16}$

Проверка. Преобразуем полученные суммы к десятичному виду:

$1011010_2=1\cdot2^6+1\cdot2^4+1\cdot2^3+1\cdot2^1=64+16+8+2=90,$
$330_5=3\cdot5^2+3\cdot5^1=75+15=90,$
$132_8=1\cdot8^2+3\cdot8^1+2\cdot8^0=64+24+2=90,$
$50_{16}=5\cdot16^1=90.$

4. Разделить число 72 на число 6 в разных системах счисления.

Деление в любой позиционной системе счисления производится по тем же правилам, как и деление углом в десятичной системе.

Двоичная система: $1001000_2:110_2$

1001000	110	
- 110	1100	
110		
- 110		
0		

Пятеричная система: $242_5:11_5$

242	11	
- 22	22	
22		
- 22		
0		

Восьмеричная система: $110_8\cdot6_8$

110	6	
- 6	14	
30		
- 30		
0		

Шестнадцатеричная система: $48_{16}\cdot6_{16}$

48	6	
- 48	12	
0		

Ответ: $72:6=12_{10}=1100_2=22_5=15_8=C_{16}$

Проверка. Преобразуем полученные суммы к десятичному виду:

$1100_2=1\cdot2^3+1\cdot2^2=8+4=12,$
$22_5=2\cdot5^1+2\cdot5^0=10+2=12,$
$14_8=1\cdot8^1+5\cdot8^0=8+4=12,$
$C_{16}=12\cdot16^0=12.$

Задание к лабораторной работе

1. Написать программу, осуществляющую сложение и вычитание чисел в заданной системе счисления. В программе предусмотреть проверку ввода некорректных данных.

Система счисления определяется как N+3, где N – номер варианта .

2. Написать программу, осуществляющую умножение и деление чисел в заданной системе счисления. В программе предусмотреть проверку ввода некорректных данных.

Система счисления определяется как 33-N, где N – номер варианта .

Требование к отчету

В отчете должны быть отображены следующие пункты:

1. Задание.
2. Теория (при необходимости).
3. Таблица соответствия переменных.
4. Схема алгоритма.
5. Листинг программы.
6. Тесты.
7. Результат выполнения программы.

Контрольные вопросы

1. Составьте таблицы сложения и умножения в троичной системе счисления и выполните вычисление 12+22.
2. Составьте таблицы сложения и умножения в троичной системе счисления и выполните вычисление 212-12.
3. Составьте таблицы сложения и умножения в троичной системе счисления и выполните вычисление 21x11.
4. Составьте таблицы сложения и умножения в троичной системе счисления и выполните вычисление 130:2.
5. Вычислить выражение 10111_2 x 110_2.
6. Вычислить выражение 140_5 x 14_5.
7. Вычислить выражение $AFF1_{16} - 19D_{16}$.
8. Вычислить выражение $121211_3 + 221_3$.
9. В какой системе счисления выполнено умножение 213 x 3 = 1144?
10. В саду 100 фруктовых деревьев – 14 яблонь и 42 груши. В какой системе счисления посчитаны деревья?

Лабораторная работа №5
Объявление и инициализация типов данных на языке ассемблера

Все ассемблерные программы состоят из одного или более предложений и комментариев. Предложение и комментарии представляют собой комбинацию знаков, входящих в алфавит языка, а также чисел и идентификаторов, которые тоже формируются из знаков алфавита.

Алфавит языка составляют цифры, строчные и прописные буквы латинского алфавита, а также следующие символы:

? @ _ $: . [ ] ( ) < > { } + / * & % ! ' ~ \| \ = # ^ ; , ` "

Исходный программный модуль на ассемблере – это последовательность предложений. Различают два типа предложений: инструкции процессора и директивы ассемблера.

Инструкции управляют работой процессора, а директивы указывают ассемблеру и редактору связей каким образом следует объединять инструкции для создания модуля, который и станет работающей программой.

Инструкция процессора на языке ассемблера состоит не более чем из четырех поле и имеет следующий формат:

[[метка:]] мнемоника [[операнды]] [[;комментарии]]

Единственное обязательное поле – поле кода операции (мнемоника), определяющее инструкцию, которую должен выполнить микропроцессор. Поле операндов определяется кодом операции и содержит дополнительную информацию о команде. Каждому коду операции соответствует определенное число операндов. Метка служит для обозначения какого-то определенного места в памяти, то есть содержит в символическом виде адрес, по которому храниться инструкция. Преобразование символических имен в действительные адреса осуществляется программой ассемблера. Часть строки исходного текста после символа «;» (если он не является элементом знаковой константы или строки знаков) считается комментарием и ассемблером игнорируется. Комментарии вносятся в программу как поясняющий текст и могут содержать любые знаки до ближайшего символа конца строки. Например:

Метка	Код операции	Операнды	;Комментарий
MET:	*MOVE*	*AX, BX*	*;Пересылка*

Программа на языке ассемблера состоит из программных модулей, содержащихся в различных файлах. Каждый модуль, в свою очередь, состоит из инструкций процессора или директив Ассемблера и заканчивается директивой END. Метка, стоящая после кода псевдооперации END, определяет адрес, с которого должно начаться выполнение программы и называется точкой входа в программу.

Каждый модуль также разбивается на отдельные части директивами сегментации, определяющими начало и конец сегмента. Каждый сегмент начинается директивой начала сегмента – SEGMENT и заканчивается директивой конца сегмента – ENDS. В начале директив ставится имя сегмента.

При программировании на языке ассемблера используются данные следующих типов:

– непосредственные данные, представляющие собой числовые или символьные значения, являющиеся частью команды. Непосредственные данные формируются программистом в процессе написания программы для конкретной команды ассемблера;

– данные простого типа, описываемые с помощью набора директив резервирования памяти, позволяющих выполнить самые элементарные операции по размещению и инициализации числовой и символьной информации. При обработке этих директив ассемблер сохраняет в своей таблице символов информацию о местоположении данных (значения сегментной составляющей адреса и смещения) и типе данных, то есть единицах памяти, выделяемых для размещения данных в соответствии с директивой резервирования и инициализации данных;

– данные сложного типа, которые были введены в язык ассемблера с целью облегчения разработки программ. Сложные типы данных (массивы, структуры, записи, объединения) строятся на основе базовых типов. Введение сложных типов данных позволяет несколько сгладить различия между языками высокого уровня и ассемблером.

Данные простого типа:

– байт - восемь последовательно расположенных битов, пронумерованных от 0 до 7, при этом бит 0 является самым младшим значащим битом;

– слово - последовательность из двух байт, имеющих последовательные адреса. Размер слова - 16 бит, где байт, содержащий нулевой бит, называется младшим, а байт, содержащий 15-й бит - старшим. Микропроцессоры Intel имеют важную особенность - младший байт всегда хранится по меньшему адресу. Адресом слова считается адрес его младшего байта. Адрес старшего байта может быть использован для доступа к старшей половине слова;

– двойное слово - последовательность из четырех байт (32 бита), расположенных по последовательным адресам. Нумерация этих бит производится от 0 до 31. Слово, содержащее нулевой бит, называется младшим словом, а слово, содержащее 31-й бит, - старшим словом. Младшее слово хранится по меньшему адресу. Адресом двойного слова считается адрес его младшего слова. Адрес старшего слова может быть использован для доступа к старшей половине двойного слова;

– учетверенное слово - последовательность из восьми байт (64 бита), расположенных по последовательным адресам. Нумерация бит производится от 0 до 63. Двойное слово, содержащее нулевой бит, называется младшим двойным словом, а слово, содержащее 63-й бит, - старшим двойным словом. Младшее двойное слово хранится по меньшему адресу. Адресом учетверенного слова считается адрес его младшего двойного слова. Адрес старшего двойного слова может быть использован для доступа к старшей половине учетверенного слова.

Очень важно уяснить себе порядок размещения данных в памяти. Он напрямую связан с логикой работы микропроцессора с данными**.**

Ассемблер предоставляет очень широкий набор средств описания и обработки данных, который вполне сравним с аналогичными средствами некоторых языков высокого уровня. Для описания простых типов данных в программе используются специальные директивы резервирования и инициализации данных, которые являются указаниями транслятору на выделение определенного объема памяти. Если проводить аналогию с языками высокого уровня, то директивы резервирования и инициализации данных являются определениями переменных. При работе с этими директивами и с дампами памяти стоит не забывать о принципе <младший байт по младшему адресу>, в соответствии с которым при сохранении данных в памяти младшая часть значения всегда записывается перед старшей частью.

Все директивы позволяют задавать строковые значения, но в памяти они могут выглядеть не так, как вы их описали в директиве. Поэтому следует использовать для определения строк директиву ***db.*** Задаваемые таким образом строки должны заключаться в кавычки. Эти кавычки могут быть одинарными (' ') или двойными (""). Если задать внутри строки два таких ограничителя подряд, то это будет означать, что данная кавычка должна быть частью строки.

Директива INCLUDE

Встречая директиву INCLUDE, Ассемблер весь текст, хранящийся в указанном файле, подставит в программу вместо этой директивы. В общем случае обращение к директиве INCLUDE (включить) имеет следующий вид:

INCLUDE имя_файла

Директиву INCLUDE можно указывать любое число раз и в любых местах программы. В ней можно указать любой файл, причем название файла записывается по правилам операционной системы.

Директивы определения данных

В общем случае все директивы объявления данных имеют следующий синтаксис:

[имя] директива dup_выражение [,dup_выражение]

Синтаксис параметра ***dup_выражение*** может быть следующим:

- ? - неинициализированные данные;
- значение - значение элемента данных;
- количество_повторов DUP.

К директивам объявления и инициализации простых данных относятся:

1. ***DB*** (Define Byte) - определить байт

*Директивой **DB** можно задавать следующие значения:*

- *выражение или константу, принимающую значение для чисел со знаком (-128...+127) и для чисел без знака (0...255);*
- *8-битовое относительное выражение, использующее операции HIGH и LOW;*

– *символьную строку из одного или более символов. Строка заключается в кавычки. В этом случае определяется столько байт, сколько символов в строке;*

2. ***DW*** (Define Word) -- определить слово

*Директивой **DW** можно задавать следующие значения:*

– *выражение или константу, принимающую значение для чисел со знаком (-32768...32767) и для чисел без знака (0...65535);*

– *выражение, занимающее 16 или менее бит, в качестве которого может выступать смещение в 16-битовом сегменте или адрес сегмента памяти;*

– *1- или 2-байтовая строка, заключенная в кавычки;*

3. ***DD*** (Define Double word) - определить двойное слово

*Директивой **DD** можно задавать следующие значения:*

– *выражение или константу, принимающую значение из чисел со знаком (-32768...+32767) и чисел без знака (0...65533) для процессоров 8086 (i8086), а также из чисел со знаком (-2147483648...+2147483647) и из чисел без знака (0...4 294 967 295) для процессоров i80386 (i80386) и выше;*

– *относительное или адресное выражение, состоящее из 16-битового адреса сегмента и 16-битового смещения;*

– *строку длиной до 4 символов, заключенную в кавычки;*

4. ***DQ*** (Define Quarter word) - определить учетверенное слово

*Директивой **DQ** можно задавать следующие значения:*

– *выражение или константу, принимающую значение из чисел со знаком (-32 768...+32 767) и чисел без знака (0...65 535) для i8086, а также из чисел со знаком (-2 147 483 648... +2 147 483 647) и чисел без знака (0...4 294 967 295) для i80386;*

– *относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) и 16 или менее бит (для младших моделей микропроцессоров Intel);*

– *константу со знаком или константу без знака;*

– *строку длиной до 8 байт, заключенную в кавычки;*

5. ***DF*** (Define Far word) - определить указатель дальнего слова
DP (Define Pointer) - определить указатель 48 бит

*Директивами **DF** и **DP** можно задавать следующие значения:*

- *выражение или константу, принимающую значение из чисел со знаком (-32 768...+32 767) и чисел без знака (0...65 535) для i80806, а также из чисел со знаком (-2 147 483 648...+2 147 483 647) и чисел без знака (0...4 294 967 295) для i80386 и выше;*
- *относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) или 16 или менее бит (для младших моделей микропроцессоров Intel);*
- *адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;*
- *константу со знаком и константу без знака;*
- *строку длиной до 6 байт, заключенную в кавычки;*

6. ***DT*** (Define Ten Bytes) - определить 10 байт

Директивой DT можно задавать следующие значения:

- *выражение или константу, принимающую значение из чисел со знаком (-32 768...+32 767) и чисел без знака (0...65 535) для i80806, а также из чисел со знаком (-2 147 483 648...+2 147 483 647) и чисел без знака (0...4 294 967 295) для i80386 и выше;*
- *относительное или адресное выражение, состоящее из 32 или менее бит (для i80386) или 16 или менее бит (для младших моделей микропроцессоров Intel);*
- *адресное выражение, состоящее из 16-битового сегмента и 32-битового смещения;*
- *константу со знаком и константу без знака;*
- *строку длиной до 10 байт, заключенную в кавычки;*
- *упакованную десятичную константу в диапазоне 0...99 999 999 999 999 999 999.*

Примеры директив определения скалярных данных:

```
integer1   DB    16
string1    DB    'abCDf'
empty1     DB    ?
contan2    DW    4*3
string3    DD    'ab'
high4      DQ    18446744073709551615
high5      DT    120892581961462917470617 5d
db6        DB    5 DUP(5 DUP(5 DUP(10)))
```

```
dw6     DW    DUP(1,2,3,4,5)
```

Для резервирования памяти под массивы используется директива DUP:

```
area    DW    128 dup(?)    ;резервируется  память объемом 128 слов
string  DB    50 dup('*')   ;строка заполняется кодом символа '*'
array   DW   256 dup(128) ;массив из 256 слов инициализ. числом 128
```

В ассемблере есть две директивы, которые можно использовать для присвоения значения идентификатору: ***EQU*** и "=". При использовании директивы EQU можно присваивать идентификаторам как строковые, так числовые значения:

имя EQU выражение

Здесь «***выражение***» может быть псевдонимом, числом или строкой, например:

```
psevdequ  ax, @data          ;псевдоним
numb      equ   23*5         ;число
str       equ   'my string'  ;строка
```

Переопределение идентификатора с помощью другой директивы ***EQU*** допускается только в том случае, если раньше ему было присвоено строковое значение.

Директива = применяется только для назначения численного значения, например: ***имя = выражение,*** где идентификатору ***имя*** присваивается результат вычисления ***выражения***.

Выражение может быть или константой, или адресом относительно сегмента и может быть как новым идентификатором, так и идентификатором, который уже был определен директивой = .

```
var1=45
var2=var1+5*34-21
var1=50
```

Следует обратить внимание, что директивы ***EQU*** и "=" не резервируют память.

Примеры:

Листинг 1. Пример применения директив определения данных

```
data    segment
;определение байта
v1db      db  ?                    ;не инициализировано
v2db      db  'Ваша фамилия'       ;символьная строка
v3db      db  56                   ;десятичная константа
v4db      db  04fh             ;шестнадцатиричная константа
v5db      db  0110100b             ;двоичная константа
v6db      db  1,'dat','name',42    ;таблица
v7db      db  8 dup(0)             ;восемь нулей
;определение слова
v1dw      dw  aff3h            ;шестнадцатиричная константа
v2dw      dw  01101111             ;двоичная константа
v3dw      dw  v5db                 ;адресная константа
v4dw      dw  24,5,7               ;3 константы
v5dw      dw  3 dup(*)             ;3 звездочки
;определение двойного слова
v1dd      dd   ?                   ;не определено
v2dd      dd   'FAdf'          ;символьная строка
v3dd      dd   08234               ;десятичная константа
v4dd      dd   v3dw-v4db           ;разность адресов
v5dd      dd   017h,05fh           ;две константы
;определение учетверенного слова
v1dq      dq   ?                   ;не определено
v2dq      dq   a83dh               ;константа
;определение десяти байтов
v1dt      dt   'Family'            ;символьная константа
d1        equ  25
d2        =    v3db
d1        equ  d2
dat       ends
end
```

Любой переменной, объявленной с помощью директив описания простых данных ассемблер присваивает три атрибута:

1. *Сегмент (**seg**) - адрес начала сегмента, содержащего переменную.*
2. *Смещение (**offset**) в байтах от начала сегмента с переменной.*

*3. Тип (**type**) - определяет количество памяти, выделяемой переменной в соответствии с директивой объявления данных.*

Получить и использовать значение этих атрибутов в программе можно с помощью операторов ассемблера ***seg, offset, type.***

```
model small
 .data
pole  dw    5
.code
mov   ax, seg pole
mov   es, ax
mov   sx, offset pole      ;теперь в паре es:dx полный адрес pole
```

Листинг 2. Пример использования директив с упрощенным описанием сегментов памяти для резервирования и инициализации данных.

```
model small       ;код занимает один сегмент
 .data
 message     db     'Запустите эту программу в отладчике','$'
 perem_1     db     0f1h
 perem_2     dw     0b7fh
 perem_3     dd     0f54d567ah
 mas         db     10 dup (' ')
 pole_1      db     5 dup (?)
 adr         dw     perem_3
 adr_full    dd     perem_3
 fin         db     'Конец сегмента данных программы $'

 .code
 start:
             mov    ax,@data
             mov    ds,ax
             mov    ah,09h
             mov    dx,offset message
             int    21h
             mov    ax,4c00h
             int    21h
 end         start
```

Практическое задание к лабораторной работе

1. Определите и запишите в протокол шестнадцатеричный объектный код для следующих директив резервирования памяти:

db 'Ваше имя'
dw Ваш год рождения + номер в группе
db '?' ;вместо вопроса подставьте дату вашего рождения
db день рождения, месяц рождения.

2. Наберите в редакторе листинг 1. Откомпилируйте программу с помощью отладчика. Получите файл листинга. Изучите его и ответьте на следующие вопросы

а) сколько байт занимает в памяти переменная v6db? Запишите ее объектный код

б) запишите значение переменных v3dw и v4dd

в) запишите в протокол объектный код для v1dw

г) запишите объектный код для v2dq

3. Наберите в редакторе листинг 2. Откомпилируйте программу с помощью отладчика. При компиляции и компоновке используйте опции сохранения отладочной информации. Исправьте ошибки и объясните их в протоколе.

Отразите в отчете

–структуру памяти загруженной программы (содержимое сегментных регистров) ;

– адрес сегмента DS после его инициализации значением адреса сегмента данных;

– обьектный код для всех переменных в формате:

Имя переменной = значение переменной-> соответствующий ей объектный код;

–запишите в протокол область данных заданную описанием:

db 2 dup (3 dup(2,6),8)

Контрольные вопросы

1. Что в языке ассемблера называется переменными, метками и именами?
2. Что может выступать в качестве операндов в выражениях языка ассемблера?
3. Основные операторы языка ассемблера.
4. Какие основные типы данных используются в языку ассемблера?
5. Структура программы на языке ассемблера.
6. Структура листинга программы. Какие сведения о программе можно получить в листинге?
7. Директивы определения скалярных данных.
8. Особенности применения директивы *DUP*.
9. Синтаксис и назначение директива DD.
10. Особенности применения директивы *INCLUDE*.
11. Синтаксис и назначение директива DB.
12. Синтаксис и назначение директива DQ, DT.

Лабораторная работа №6
Изучение режимов адресации при пересылке данных в микропроцессорах INTEL 8086

Архитектура процессора 8086

Системный блок персонального компьютера содержит блок питания, системную (материнскую) плату, адаптеры внешних устройств, накопители на жестких магнитных (НЖМД) и гибких (НГМД) дисках, а также ряд других устройств. Для нас наибольший интерес представляет системная плата, на которой размещаются постоянное запоминающее устройство ПЗУ (ROM - read only memory), оперативное запоминающее устройство ОЗУ (RAM - random access memory), процессор и логика управления, связанные между собой шинами. Физически и ОЗУ и ПЗУ выполнены в виде микросхем. Характерным для персонального компьютера является тот факт, что при выключении электропитания содержимое ОЗУ утрачивается (энергозависимая память), а ПЗУ – нет (энергонезависимая память). Одна из основных задач ПЗУ обеспечить процедуру старта персонального компьютера. В ПЗУ хранятся базовая система ввода/вывода BIOS, а также некоторые служебные программы и таблицы, например, начальный загрузчик, программа тестирования POST и т.п.

Оперативная память ОЗУ предназначена для временного хранения программ и данных, которыми они манипулируют. Логически оперативную память можно представить в виде последовательности ячеек, каждая из которых имеет свой номер, называемый адресом.

Центральный процессор (ЦП) в современных персональных компьютерах выполнен в виде одной сверхбольшой интегральной микросхемы (СБИС). ЦП выполняет машинные команды, выбирая их в заданной последовательности из оперативной памяти. Работа всех электронных устройств компьютера координируется сигналами управления, вырабатываемыми ЦП и некоторыми другими СБИС, сигналами тактового генератора, с помощью которых синхронизируются действия всех сигналов.

Возможности компьютера в большей степени зависят от типа установленного процессора и его тактовой частоты. Семейство процессоров 80x86 корпорации Intel включает в себя микросхемы: 8086, 80186, 80286, 80386, 80486, Pentium, Pentium II, Pentium III и т.д. Совместимые с 80x86 микросхемы выпускают также фирмы AMD, IBM,

Cyrix. Особенностью этих процессоров является преемственность на уровне машинных команд: программы, написанные для младших моделей процессоров, без каких-либо изменений могут быть выполнены на более старших моделях. При этом базой является система команд процессора 8086, знание которой является необходимой предпосылкой для изучения остальных процессоров.

Структуру центрального процессора Intel 8086 можно разделить на два логических блока (рис.6.1):

− *блок исполнения* (EU:Execution Unit);

− *блок интерфейса шин* (BIU:Bus Interface Unit).

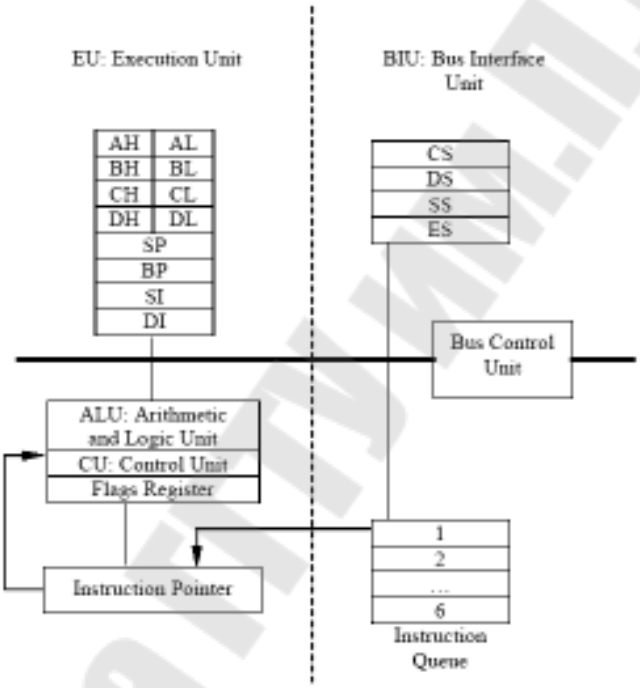

Рисунок 6.1. Структура центрального процессора

Интерфейс (interface) - это совокупность средств, обеспечивающих сопряжение устройств и программных модулей как на физическом, так и на логическом уровнях.

В состав EU входят: арифметическо-логическое устройство ALU, устройство управления CU и десять регистров. Устройства блока EU обеспечивают обработку команд, выполнение арифметических и логических операций.

Три части блока BIU - устройство управления шинами, блок очереди команд и регистры сегментов – предназначены для выполнения следующих функций:

− управление обменом данными с EU, памятью и внешними устройствами ввода/вывода;

− адресация памяти;

− выборка команд (осуществляется с помощью блока очереди команд Queue, который позволяет выбирать команды с упреждением).

С точки зрения программиста, процессор 8086 состоит из восьми регистров общего назначения и адресации, четырех сегментных регистров, регистра адреса команд (счетчика команд) и регистра флагов. Процессор выставляет на 20-раздядную шину адреса адрес выбираемых из памяти команд (или данных), которые по 16-разрадной шине данных поступают в шестибайтный буфер (очередь команд), а затем исполняются.

Микропроцессор оперирует с двоичной системой счисления (двоичной системой представления данных). Символьная информация кодируется в соответствии с кодом ASCII (Американский стандартный код для обмена информацией). Числовые данные кодируются в соответствии с двоичной арифметикой. Отрицательные числа представляются в дополнительном коде.

Минимальная единица информации, соответствующая двоичному разряду, называется *бит* (Bit). Группа из восьми битов называется *байтом* (Byte) и представляет собой наименьшую адресуемую единицу – ячейку памяти. Биты в байте нумеруют справа налево цифрами 0...7.

Двухбайтовое поле образует шестнадцатиразрядное *машинное слово* (Word), биты в котором нумеруются от 0 до 15 справа налево. Байт с меньшим адресом считается младшим.

Четырехбайтовое поле образует *двойное слово* (Double Word), а шестнадцатибайтовое – *параграф* (Paragraph). Таким образом, с помощью 16 разрядной шины данных можно передавать числа от 0 (во всех проводниках сигнал 0) до 65535 (во всех проводниках сигнал 1). Несмотря на то, что двоичная система обладает высокой наглядностью, она имеет существенный недостаток – числа, записанные в двоичной системе, слишком громоздки. С другой стороны, привычная для нас десятичная система слишком сложна для перевода чисел в двоичную систему счисления и обратно. Поэтому наибольшее распространение в практике программирования получила шестнадцатеричная система счисления.

При написании программ принято двоичные числа сопровождать латинской буквой *B* или *b*, (например, 101B), а шестнадцатеричные – буквой *H* или *h* на конце. Если число начинается с буквы, то обязательной является постановка нуля впереди, например, 0BA8H. Кроме

того, Ассемблер позволяет использовать числа представленные в десятичной и восьмеричной системах счисления, которые соответственно сопровождаются латинскими буквами *D* и *O*.

Регистры

Процессор 8086 содержит 14 шестнадцатиразрядных регистров, которые используются для управления исполнением команд, адресации и выполнения арифметических операций. Регистр, содержащий одно слово, адресуется по имени.

Регистры общего назначения

К ним относятся 16-разрядные регистры AX, BX, CX, DX, каждый из которых разделен на две части по восемь разрядов:

AX состоит из AH (старшая часть) и AL (младшая часть);

BX состоит из BH и BL;

CX состоит из CH и CL;

DX состоит из DH и DL;

В общем случае функция, выполняемая тем или иным регистром, определяется командами, в которых он используется. При этом с каждым регистром связано некоторое стандартное его значение. Ниже перечисляются наиболее характерные функции каждого регистра:

− регистр AX служит для временного хранения данных (регистр аккумулятор), часто используется при выполнении операций сложения, вычитания, сравнения и других арифметических и логических операций;

− регистр BX служит для хранения адреса некоторой области памяти (базовый регистр), а также используется как вычислительный регистр;

− регистр CX иногда используется для временного хранения данных, но в основном служит счетчиком, в нем хранится число повторений одной команды или фрагмента программы;

− регистр DX используется главным образом для временного хранения данных, часто служит средством пересылки данных между разными программными системами, а также используется в качестве расширителя аккумулятора для вычислений повышенной точности и при умножении и делении.

Регистры для адресации

В микропроцессоре существуют четыре 16-битовых (2 байта или 1 слово) регистра, которые могут принимать участие в адресации операндов. Один из них одновременно является и регистром общего назначения — это регистр BX, или базовый регистр. Три другие регистра — это указатель базы BP, индекс источника SI и индекс результата DI (два последних еще называются *индексными регистрами*). Отдельные байты этих трех регистров недоступны.

Любой из названных выше 4-х регистров может использоваться для хранения адреса памяти, а команды, работающие с данными из памяти, могут обращаться за ними к этим регистрам. При адресации памяти базовые и индексные регистры могут быть использованы в различных комбинациях. Разнообразные способы сочетания в командах этих регистров и других величин называются способами или *режимами адресации*.

Регистры сегментов

Имеются четыре регистра сегментов, с помощью которых память можно организовать в виде совокупности четырех различных сегментов. Этими регистрами являются:

− CS – регистр программного сегмента (сегмента кода) определяет местоположение части памяти, содержащей программу, то есть выполняемые процессором команды;

− DS – регистр информационного сегмента (сегмента данных) идентифицирует часть памяти, предназначенной для хранения данных;

− SS – регистр стекового сегмента (сегмента стека) определяет часть памяти, используемой как системный стек;

− ES – регистр расширенного сегмента (дополнительного сегмента) указывает дополнительную область памяти, используемую для хранения данных.

Эти четыре различные области памяти могут располагаться практически в любом месте физической машинной памяти. Поскольку местоположение каждого сегмента определяется только содержимым соответствующего регистра сегмента, для реорганизации памяти достаточно всего лишь изменить это содержимое.

Регистр указателя стека

Указатель стека SP – это 16-битовый регистр, который определяет смещение текущей вершины стека. Указатель стека SP вместе с сегментным регистром стека SS используются микропроцессором для формирования физического адреса стека. Стек всегда растет в направлении меньших адресов памяти, то есть когда слово помещается в стек, содержимое SP уменьшается на 2, когда слово извлекается из стека, микропроцессор увеличивает содержимое регистра SP на 2

Регистр указателя команд IP

Регистр указателя команд IP, иначе называемый регистром счетчика команд, имеет размер 16 бит и хранит адрес ячейки памяти, содержащей начало следующей команды. Микропроцессор использует регистр IP совместно с регистром CS для формирования 20-битового физического адреса очередной выполняемой команды, при этом регистр CS задает сегмент выполняемой программы, а IP – смещение от начала сегмента. По мере того, как микропроцессор загружает команду из памяти и выполняет ее, регистр IP увеличивается на число байт в команде. Для непосредственного изменения содержимого регистра IP служат команды перехода.

Регистр флагов

Флаги – это отдельные биты, принимающие значение 0 или 1. Регистр флагов (признаков) содержит девять активных битов (из 16). Каждый бит данного регистра имеет особое значение, некоторые из этих бит содержат код условия, установленный последней выполненной командой. Другие биты показывают текущее состояние микропроцессора.

X X X X *OF* *DF* *IF* *TF* *SF* *ZF* X *AF* X *PF* X *CF*

Биты регистра флагов имеют следующее назначение:

OF (признак переполнения) – равен единице, если возникает арифметическое переполнение, то есть когда объем результата превышает размер ячейки назначения;

DF (признак направления) – устанавливается в единицу для автоматического декремента в командах обработки строк, и в ноль – для инкремента;

IF (признак разрешения прерывания) – прерывания разрешены, если IF=1. Если IF=0, то распознаются лишь немаскированные прерывания;

TF (признаков трассировки) – если TF=1, то процессор переходит в состояние прерывания INT 3 после выполнения каждой команды;

SF (признак знака) – SF=1, когда старший бит результата равен единице. Иными словами, SF=0 для положительных чисел, и SF=1 для отрицательных чисел;

ZF (признак нулевого результата) – ZF=1, если результат равен нулю;

AF (признак дополнительного переноса) – этот флаг устанавливается в единицу во время выполнения команд десятичного сложения и вычитания при возникновении переноса или заема между полубайтами;

PF (признак четности) – этот признак устанавливается в единицу, если результат имеет четное число единиц;

CF (признак переноса) – этот флаг устанавливается в единицу, если имеет место перенос или заем из старшего бита результата, он полезен для произведения операций над числами длиной в несколько слов, которые сопряжены с переносами и заемами из слова в слово;

X – зарезервированные биты.

Легко заметить, что все флаги младшего байта регистра флагов устанавливаются арифметическими или логическими операциями процессора. За исключением флага переполнения, все флаги старшего байта отражают состояние микропроцессора и влияют на характер выполнения программы. Флаги старшего байта устанавливаются и сбрасываются специально предназначенными для этого командами. Флаги младшего байта являются кодами условного перехода для изменения порядка выполнения программы.

Сегменты, принцип сегментации

Числа, устанавливаемые процессором на адресной шине, являются адресами, то есть номерами ячеек оперативной памяти (ОП). Размер ячейки ОП составляет 8 разрядов, то есть 1 байт. Поскольку для адресации памяти процессор использует 16-разрядные адресные регистры, то это обеспечивает ему доступ к 65536 (FFFFh) байт или 64К (1К = 1024 байт = 210 байт) основной памяти.

Блок непосредственно адресуемой памяти размером 64К называется *сегментом*. Такой объем адресуемой памяти явно недостаточен,

и поэтому для адресации большего объема памяти в процессорах семейства x86 используется принцип сегментации памяти. Любой адрес формируется из адреса сегмента (всегда кратен 16, то есть начинается с границы параграфа) и адреса ячейки внутри сегмента (эта часть адреса называется *смещением*). Процедура пересчета логических адресов в физические, называемая вычислением *абсолютного* адреса.

Рассмотрим алгоритм формирования абсолютного адреса на следующем примере. Когда процессор выбирает очередную команду на исполнение, в качестве ее адреса используется содержимое, регистра IP. Этот адрес называется *исполнительным*. Поскольку регистр IP шестнадцатиразрядный, исполнительный адрес тоже содержит 16 двоичных разрядов.

Чтобы получить 20-битовый адрес (адресная шина, соединяющая процессор и память 20-разрядная), дополнительные 4 бита адресной информации

извлекаются из сегментных регистров (в приведенном примере из регистра CS). Сами сегментные регистры имеют размер в 16 разрядов, а содержащиеся в этих регистрах 16-битовые значения называются *базовым адресом сегмента*.

Микропроцессор объединяет 16-битовый исполнительный адрес и 16-битовый базовый адрес следующим образом: он расширяет содержимое сегментного регистра (базовый адрес) четырьмя нулевыми битами в младших разрядах, делая его 20-битовым (полный адрес сегмента) и прибавляет смещение (исполнительный адрес). При этом 20-битовый результат является физическим или абсолютным адресом ячейки памяти (рис. 6.2).

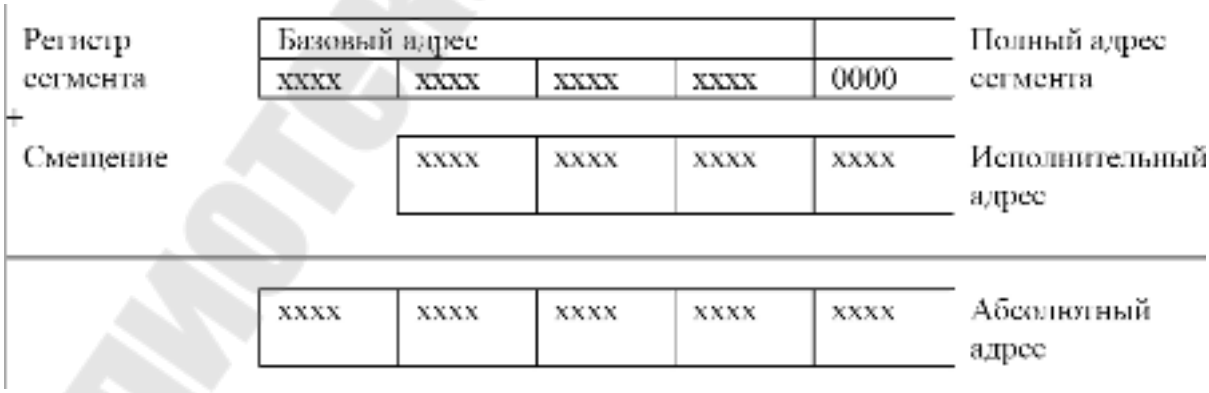

Рисунок 6.2. Принцип получения абсолютного адреса

Существуют три основных типа сегментов:

− сегмент кода – содержит машинные команды, адресуется регистром CS;

− сегмент данных – содержит данные, то есть константы и рабочие области, необходимые программе, адресуется регистром DS;

− сегмент стека – содержит временные данные, адресуется регистром SS.

При записи команд на языке Ассемблера принято указывать адреса с помощью следующей конструкции:

<адрес сегмента>:<смещение>

или

<сегментный регистр>:<адресное выражение>

Стек

Во многих случаях программе требуется временно запомнить некоторую информацию. Эта проблема в персональном компьютере решена посредством реализации стека LIFO («последний пришел - первый ушел»), называемого также стеком включения/извлечения (stack).

Стек – это область памяти для временного хранения данных, в которую по специальным командам можно записывать отдельные слова (но не байты); при этом для запоминания данных достаточно выполнить лишь одну команду и не нужно беспокоиться о выборе соответствующего адреса: процессор автоматически выделяет для них свободное место в области временного хранения.

Наиболее важное использование стека связано с подпрограммами, в этом случае стек содержит адрес возврата из подпрограммы, а иногда и передаваемые в(из) подпрограмму(ы) данные. Стек обычно рассчитан на косвенную адресацию через регистр указатель стека SP. При включении элементов в стек производится автоматический декремент указателя стека, а при извлечении – инкремент, то есть стек всегда «растет» в сторону меньших адресов памяти. Адрес последнего включенного в стек элемента называется *вершиной стека* (TOS), а адрес сегмента стека – *базой стека* (обычно содержится в регистре SS).

Директивы сегментации

Сегментные регистры позволяют одновременно работать:

– с одним сегментом кода;

– с одним сегментом стека;
– с одним сегментом данных;
– с тремя дополнительными сегментами данных.

Физически сегмент представляет собой область памяти, занятую командами и/или данными, адреса которых вычисляются относительно значения в соответствующем сегментном регистре. Директивы определения сегментов:

.code *– начало или продолжение сегмента кода;*

.data *– начало или продолжение сегмента инициализированных данных;*

.stack [размер] *– начало или продолжение сегмента стека модуля. (размер – размер стека)*

Режимы адресации

В зависимости от спецификаций и местоположения источников образования полного (абсолютного) адреса в языке ассемблера различают следующие способы адресации операндов:

− регистровая;
− прямая;
− непосредственная;
− косвенная;
− базовая;
− индексная;
− базово-индексная.

Регистровая адресация подразумевает использование в качестве операнда регистра процессора, например,

```
PUSH   DS
MOV    BP,SP
```

При прямой адресации один операнд представляет собой адрес памяти, второй – регистр:

```
MOV    Data,AX
```

Непосредственная адресация применяется, когда операнд, длинной в байт или слово находится в ассемблерной команде:

```
MOV   AH,4CH
```

При использовании косвенной адресации абсолютный адрес формируется исходя из сегментного адреса в одном из сегментных регистров и смещения в регистрах BX, BP, SI или DI:

```
MOV   AL,[BX]       ;База – в DS, смещение – в BX
MOV   AX,[BP]       ;База – в SS, смещение – в BP
MOV   AX,ES:[SI]    ;База – в ES, смещение – в SI
```

В случае применения базовой адресации исполнительный адрес является суммой значения смещения и содержимого регистра BP или BX, например,

```
MOV   AX,[BP+6]     ;База – SS, смещение – BP+6
MOV   DX,[BX+8]     ;База – DS, смещение – BX+8
```

При индексной адресации исполнительный адрес определяется как сумма значения указанного смещения и содержимого регистра SI или DI так же, как и при базовой адресации, например,

```
MOV   DX,[SI+5]    ;База – DS, смещение – SI+5
```

Базово-индексная адресация подразумевает использование для вычисления исполнительного адреса суммы содержимого базового регистра и индексного регистра, а также смещения, находящегося в операторе, например,

```
MOV   BX,[BP][SI]          ;База – SS, смещение – BP+SI
MOV   ES:[BX+DI],AX        ;База – ES, смещение – BX+DI
MOV   AX,[BP+6+DI]         ;База – SS, смещение - BP+6+DI
```

Пример программы:

%TITLE "Команды MOV и режимы адресации.

```
model small
.data
value      =       528
b_x        DB      1,2,4
w_x        DW      8,16,32,64
```

```
b_var    DB    4
w_var    DW    1234h      ;Число в памяти:34h(мл. б):12h(ст. б)
.code
Start:
     mov   ax,@data       ;Установка в ds адреса
     mov   ds,ax          ;сегмента данных.
;Непосредственная адресация.
    mov    al,255         ;255=0FFh-беззнаковое число
    mov    ah,-1          ;-1=0FFh-отрицательное число
    mov    ax,value/5+20 ;Загрузка в ax константного выражения
    mov    bx, offset w_x; ;Адрес переменной w_x в bx (bx=0003h)
;Регистровая и прямая адресации. Символьная переменная,
;заключённая в квадратные скобки (например [b_x]), – выполняет
;роль адреса этой переменной в памяти
    mov    dl,al
    mov    al,[b_x]
    mov    dx,[w_x]
    mov    si,[w_var]
    mov    al,[b_var]
    mov    ah,[b_var+1]
;Косвенная регистровая.
    mov    cx,[bx]
    mov    [word bx], 0EEEh
;Базовая адресация.
    mov    ax,[bx+2]
    mov    [word bx+2],24
;Индексная адресация.
    mov   si,1
    mov   al,[si+b_x]
;Базово индексная адресация.
    inc    si                    ;si=2
    mov    bx,1                  ;bx=1
    mov    ax,[bx+si+w_x]
    mov    [word bx+si+w_x],128
 END Start                       ;Конец программы/точка входа.
```

Практическое задание к лабораторной работе

1. Набрать и запустить в отладчике пример программы, представленной выше.
2. При возникновении ошибок компиляции программы исправить их.
3. В пошаговом режиме при каждой команде отразить в отчете содержимое регистров, которые участвуют в пересылке данных.
4. Просмотреть, происходит ли изменение битов регистра флагов. В отчете отразить и объяснить по какой причине это происходит.
5. Выполнить номера индивидуальных заданий (таблица 5.1), согласно варианту указанному преподавателем

Разместить в памяти номер зачетной книжки, каждое число представив отдельным байтом. Задать три дополнительные переменные определяющие день (*пример 12*), месяц (*пример 05*) и год (*пример 2008*) рождения.

Таблица 5.1

Индивидуальные задания

Номер	Задание
1	Поменять местами первое и последние числа зачетки
2	Поменять местами второе и предпоследнее числа зачетки
3	Поменять местами день и месяц рождения
4	Заменить первые три числа зачетки последним
5	Заменить последние три числа зачетки первым
6	Заменить первые два числа зачетки на день рождения
7	Заменить первые два числа зачетки на месяц рождения
8	Заменить последние два числа зачетки на день рождения
9	Заменить последние два числа зачетки на месяц рождения
10	Скопировать номер зачетки которые относительно смещены в памяти на [10+день рождения]
11	Скопировать номер зачетки которые относительно смещены в памяти на [10+месяц рождения]
12	Скопировать в регистр год рождения и поменять местами старший и младший байты. Результат поместить в ячейку памяти со смешением [20+месяц рождения]
13	Поменять местами первое число зачетки с ячейкой памяти смещенной относительно первого числа зачетки на величину [10 + последнее число зачетки]
14	Поменять местами последнее число зачетки с ячейкой памяти смещенной относительно начало сегмента данных на величину [10 + первое число зачетки]
15	Зеркально отобразить номер зачетной книжки

В отчете отразить изменение содержимого регистров и ячеек памяти + листинг. Объяснить особенности и область применения изученных видов адресации.

Контрольные вопросы

1. Состав и назначение элементов системного блока персонального компьютера.
2. На какие два логических блока можно разделить структуру центрального процессора Intel 8086? Их основные функции?
3. Регистры микропроцессора Intel 8086 их назначение.
4. Флаги. Структура регистра флагов.
5. В чем состоит принцип сегментации памяти?
6. Организация и назначение стека.
7. Что такое прерывание? Назначение таблицы векторов прерываний?
8. Основные режимы адресации памяти.
9. Особенности применения регистровой адресации.
10. Особенности применения прямой адресации.
11. Особенности применения непосредственной адресации.
12. Особенности применения косвенной адресации.
13. Особенности применения базовой адресации.
14. Особенности применения индексной адресации.
15. Особенности применения базово-индексной адресацию.

Литература

1. Абель П. Язык Ассемблера для IBM PC и программирование: Пер. с анг. Ю.В. Сальникова. – М.: Высш. шк., 1992. – 447 с.
2. Бердышев Е.М. Технология ММХ. Новые возможности процессоров P5 и P6. – М.: Диалог МИФИ, 1998. – 234 с.
3. Борзенко А.Е. IBM PC: устройство, ремонт, модернизация. – М.: ТОО фирма «Компьютер пресс», 1995. – 297 с.
4. Бройдо В.Л., Ильина О.П. Архитектура ЭВМ и систем: Учебник для вузов. – СПб: Изд-во «Питер», 2006. – 718 с.
5. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование. В 4-х кн. – М.: Пранал, 1993.
6. Злобин В.К., Григорьев В.Л. Программирование арифметических операций в микропроцессорах: Учеб. пособие для техн. ВУЗов. – М.: Высш. шк., 1991. – 301 с.
7. Макроассемблер MASM – М: Радио и связь, 1994. – 241 с.
8. Пешков А.Т. Организация и функционирование ЭВМ. Метод. пособие для студ. спец. «Программное обеспечение информационных технологий» дневной формы обуч.: В 3 ч. Ч. 1. Арифметические основы ЭВМ . Мн.: БГУИР, 2004. — 61 с.
9. Сван К. Освоение Turbo Assembler. – Киев: Диалектика, 1996. – 544 с.
10. Таненбаум Э. Архитектура компьютера. – СПб: Изд-во «Питер», 2002. – 704 с.
11. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб: Изд-во «Питер», 2006. – 668 с.
12. Юров Б. Assembler: учебник.2 – е изд. СПб: Изд-во «Питер», 2006. – 637 с.

Содержание

Лабораторная работа №1 Системы счисления..... 3

Лабораторная работа №2 Перевод целых десятичных чисел
в другие системы счисления..... 7

Лабораторная работа №3 Перевод двоичных чисел
в восьмеричную и шестнадцатеричную системы счисления
и обратные преобразования..... 9

Лабораторная работа №4 Арифметические операции
в позиционных системах счисления 14

Лабораторная работа №5 Объявление и инициализация
типов данных на языке ассемблера..... 19

Лабораторная работа №6 Изучение режимов адресации
при пересылке данных в микропроцессорах INTEL 8086 30

Литература 44

Ковалев Алексей Викторович
Самовендюк Николай Владимирович
Литвинов Дмитрий Александрович

ОРГАНИЗАЦИЯ И ФУНКЦИОНИРОВАНИЕ ЭВМ

Лабораторный практикум по одноименному курсу для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения

Подписано в печать 01.06.09.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 2,79. Уч.-изд. л. 2,6.
Изд. № 191.
E-mail: ic@gstu.gomel.by
http://www.gstu.gomel.by

Отпечатано на цифровом дуплекаторе
с макета оригинала авторского для внутреннего использования.
Учреждение образования «Гомельский государственный
технический университет имени П. О. Сухого».
246746, г. Гомель, пр. Октября, 48.