

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Е. А. РУЖИЦКАЯ, О. Г. ОСИПОВА, В. А. КОВАЛЁВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

Системы счисления, программная модель микропроцессора, арифметические команды

Практическое пособие

для студентов специальностей

1–40 01 01 «Программное обеспечение

информационных технологий»,

1–40 04 10 «Информатика и технологии программирования»

Гомель
ГГУ им. Ф. Скорины
2016

УДК 004.431.4(076)
ББК 32.973.21я73
Р837

Рецензенты:

кандидат физико-математических наук Т. В. Тихоненко,
кандидат технических наук В. Д. Левчук

Рекомендовано к изданию научно-методическим советом
учреждения образования «Гомельский государственный
университет имени Франциска Скорины»

Ружицкая, Е. А.

Р837 Программирование на языке Assembler : системы счисления, программная модель микропроцессора, арифметические команды : практическое пособие / Е. А. Ружицкая, О. Г. Осипова, В. А. Ковалёва ; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2016. – 47 с.
ISBN 978-985-577-110-5

Практическое пособие предназначено для оказания помощи студентам в обладении машинно-ориентированным языком программирования Assembler. Излагается теоретический материал и дается практическое руководство по таким темам, как системы счисления, программная модель микропроцессора, регистры, арифметические команды ассемблера.

Адресовано студентам 1 курса специальностей 1–40 01 01 «Программное обеспечение информационных технологий», 1–40 04 10 «Информатика и технологии программирования».

**УДК 004.431.4(076)
ББК 32.973.21я73**

ISBN 978-985-577-110-5 © Ружицкая Е. А., Осипова О. Г.,
Ковалёва В. А., 2016

© Учреждение образования
«Гомельский государственный
университет имени Франциска Скорины»,
2016

Оглавление

Предисловие	4
Тема 1. Системы счисления	5
1.1 Десятичная позиционная система счисления с неотрицательной базой	5
1.2 Двоичная система счисления с неотрицательной базой	5
1.3 Восьмеричная система счисления	7
1.4 Шестнадцатеричная система счисления	8
1.5 Связь двоичной, восьмеричной и шестнадцатеричной систем счисления	9
1.6 Перевод чисел из системы счисления с основанием p в систему счисления с основанием q	10
1.7 Представление отрицательных чисел	11
1.8 Практическое задание	13
Тема 2. Программная модель микропроцессора. Регистры	24
2.1 Наборы регистров	24
2.2 Организация памяти	31
Тема 3. Нотация языка Assembler. Программирование линейных вычислительных процессов	33
3.1 Псевдооператоры	34
3.2 Команды пересылки	37
3.3 Арифметические команды	37
3.4 Команды расширения знака	39
3.5 Примеры использования арифметических команд	39
3.6 Порядок работы с программой на ассемблере	41
3.7 Сокращенные директивы сегментации	42
3.8 Практическое задание	45
Литература	47

Предисловие

Необходимыми требованиями к современному специалисту в области информатики и технологии программирования являются: свободное владение программированием как средством использования ЭВМ; умение использовать существующие прикладные программы; применение современных информационных технологий, новейшего программного и технического обеспечения компьютеров.

Актуальной является подготовка специалиста, знающего машинно-ориентированный язык программирования Assembler, умеющего построить эффективные алгоритмы решения поставленной задачи, выбрать наиболее подходящие структуры данных, программные и технические средства его реализации и с учетом операционного окружения разработать программные приложения, отвечающие новейшим компьютерным технологиям и требованиям.

Практическое пособие предназначено для оказания помощи студентам в овладении машинно-ориентированным языком программирования Assembler. Излагается теоретический материал и дается практическое руководство по таким темам, как системы счисления, программная модель микропроцессора, регистры, арифметические команды ассемблера.

Язык программирования Assembler изучается студентами 1 курса специальностей 1–40 01 01 «Программное обеспечение информационных технологий» в рамках дисциплины «Языки программирования» и 1–40 04 10 «Информатика и технологии программирования» в курсе «Программирование».

Тема 1. Системы счисления

Счислением называют совокупность приемов наименования и обозначения чисел. Знаки, применяемые при обозначении чисел, называют цифрами. Системы счисления подразделяются на позиционные и непозиционные.

В непозиционной системе счисления значение цифры (символа) не зависит от его положения в числе. Примером такой системы служит римская система счисления, использующая набор символов: I, V, X, C, D, L и т. д.

В позиционных системах счисления значение цифр изменяется при изменении их положения (позиции) в последовательности.

Пусть p – целое число, большее единицы, которое будем называть основанием системы счисления. Выберем p попарно различных знаков: p -ичных цифр. Кроме того, выберем p последовательных целых чисел, среди которых содержится ноль. Эту последовательность будем называть базой системы счисления. Между p -ичными цифрами и числами базы установим взаимно однозначное соответствие.

1.1 Десятичная позиционная система счисления с неотрицательной базой

Основание системы счисления: $p = 10$.

База системы счисления: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Всякое положительное число a может быть представлено в виде конечной или бесконечной p -ичной дроби

$$a = a_m p^m + a_{m-1} p^{m-1} + a_{m-2} p^{m-2} + \dots + a_{m-n+1} p^{m-n+1} + \dots,$$

где $a_m \neq 0$,

m – целое число (старший p -ичной разряд числа a).

Последовательность p -ичных цифр обозначает число, равное сумме значений её цифр:

$$\begin{aligned} & a_n a_{n-1} \dots a_1 a_0, a_{-1} \dots a_{-m} = \\ & = a_n p^n + a_{n-1} p^{n-1} + \dots + a_1 p^1 + a_0 p^0 + a_{-1} p^{-1} + \dots + a_{-m} p^{-m}. \end{aligned}$$

Пример:

$$856,25_{10} = 8 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}.$$

1.2 Двоичная система счисления с неотрицательной базой

Основание системы счисления: $p = 2$.

База системы счисления: 0, 1.

1.2.1 Перевод из десятичной системы счисления в двоичную систему счисления

Перевод целых чисел состоит в поиске остатков при делении на два.

$$11_{10} = 1011_2$$

$$\begin{array}{r|l} 11 & 2 \\ \hline 10 & 5 \\ \hline 1 & 2 \\ \hline & 1 \\ \hline & 0 \end{array}$$

Перевод десятичных дробей заключается в поиске целых частей при умножении на 2.

$$0,625_{10} = 0,101_2$$

$$\begin{array}{r|l} 0,625 & 2 \\ \hline & 2 \\ \hline 1,250 & 2 \\ \hline & 2 \\ \hline 0,500 & 2 \\ \hline & 2 \\ \hline 1,000 & \end{array}$$

$$41,875_{10} = 101001,111_2$$

$$\begin{array}{r|l} 41 & 2 \\ \hline 40 & 20 \\ \hline 1 & 10 \\ \hline & 5 \\ \hline & 2 \\ \hline & 1 \\ \hline & 0 \end{array}$$

$$\begin{array}{r|l} 0,875 & 2 \\ \hline & 2 \\ \hline 1,750 & 2 \\ \hline & 2 \\ \hline 1,500 & 2 \\ \hline & 2 \\ \hline 1,000 & \end{array}$$

1.2.2 Перевод из двоичной системы счисления в десятичную систему счисления

Перевод из двоичной системы счисления в десятичную систему счисления осуществляется через разряды числа. Двоичное число нужно представить в виде суммы степеней двойки с коэффициентами – цифрами:

$$1011,011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 11,375_{10}.$$

Действия в двоичной системе счисления.

$$\begin{aligned} \text{Сложение: } & 0 + 0 = 0; & 0 + 1 = 1; \\ & 1 + 0 = 1; & 1 + 1 = 10. \end{aligned}$$

$$\begin{array}{r} +1101,110 \\ \underline{101,101} \\ 10011,011 \end{array}$$

$$\begin{array}{r} +11111 \\ \underline{1} \\ 100000 \end{array}$$

Вычитание: $10 - 1 = 1$.

$$\begin{array}{r} 1010 \\ - 1000 \\ \hline 101 \\ 111 \end{array}$$

Умножение: $0 \times 0 = 0$; $0 \times 1 = 0$;
 $1 \times 0 = 0$; $1 \times 1 = 1$.

$$\begin{array}{r} 11011 \\ 101 \\ + 11011 \\ \hline 11011 \\ 11011 \\ \hline 10000111 \end{array}$$

$$\begin{array}{r} 11011 \\ \hline 1001101101 \end{array}$$

Деление – сводится к умножению на 0 или 1 и вычитанию:

$$\begin{array}{r} 1111 \mid 101 \\ 101 \mid 11 \\ \hline 101 \\ 101 \\ \hline 0 \end{array}$$

$$\begin{array}{r} 1101001 \mid 1111 \\ 1111 \mid 1111 \\ \hline 10110 \\ 1111 \\ \hline 1111 \\ 1111 \\ \hline 0 \end{array}$$

1.3 Восьмеричная система счисления

Основание системы счисления: $p = 8$.

База системы счисления: 0, 1, 2, 3, 4, 5, 6, 7.

Связь между десятичной, двоичной и восьмеричной системами счисления представлена в таблице 1.1.

Таблица 1.1 – Связь десятичной, двоичной и восьмеричной систем счисления

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

Правило перевода чисел из двоичной системы счисления в восьмеричную систему счисления:

1) двоичное число разбивается на группы по 3 цифры, называемые *триадами*, слева и справа от запятой; неполные группы дополняются нулями;

2) каждая группа заменяется соответствующей восьмеричной цифрой.

$$111010101,0111_2 = x_8$$

$$111.010.101,011.100_2 = 725,34_8$$

Правило перевода из восьмеричной системы счисления в двоичную систему счисления: каждая восьмеричная цифра заменяется соответствующей триадой.

$$13,3_8 = 001.011,011_2 = 1011,011_2$$

Действия в восьмеричной системе счисления:

$$\begin{array}{r} +325,71203 \\ 67,67375 \\ \hline 415,60600 \end{array}$$

$$\begin{array}{r} -11076,01 \\ 705,62 \\ \hline 10170,17 \end{array}$$

$$\begin{array}{r} \times 32,5 \\ 12 \\ \hline +652 \\ 325 \\ \hline 412,2 \end{array}$$

$$\begin{array}{r} 4604 \mid 34 \\ 34 \mid 127 \\ \hline 120 \\ 70 \\ \hline 304 \\ 304 \\ \hline 0 \end{array}$$

1.4 Шестнадцатеричная система счисления

Основание системы счисления: $p = 16$.

База системы счисления: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Связь между десятичной, двоичной и шестнадцатеричной системами счисления представлена в таблице 1.2.

Правило перевода чисел из двоичной системы счисления в шестнадцатеричную систему счисления:

1) двоичное число разбивается на группы по 4 цифры, называемые *тетрадами*, слева и справа от запятой; неполные группы дополняются нулями.

2) каждая группа заменяется соответствующей шестнадцатеричной цифрой.

$$111010101,0111_2 = x_{16}$$

$$0001.1101.0101,0111_2 = 1D5,7_{16}$$

Правило перевода чисел из шестнадцатеричной системы счисления в двоичную систему счисления: каждая шестнадцатеричная цифра заменяется соответствующей тетрадой.

$$5B,4_{16} = 0101.1011,0100_2 = 1011011,01_2$$

Таблица 1.2 – Связь десятичной, двоичной и шестнадцатеричной систем счисления

10 с/с	2 с/с	16 с/с
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Связь между двоичной, восьмеричной и шестнадцатеричной системами счисления:

$$111010101,0111_2 = 725,34_8 = 1D5,7_{16}$$

Действия в шестнадцатеричной системе счисления:

$$\begin{array}{r} +325,71203 \\ \underline{67,67375} \\ 38C,D8578 \end{array} \quad \begin{array}{r} \underline{11076,01} \\ \underline{705,62} \\ 10970,9F \end{array}$$

$$\begin{array}{r} \times 32,5 \\ \underline{12} \\ +64A \\ \underline{325} \\ 389,A \end{array} \quad \begin{array}{r} \underline{3BEC} \mid \underline{34} \\ \underline{34} \mid 127 \\ \underline{7E} \\ \underline{68} \\ \underline{16C} \\ 0 \end{array}$$

1.5 Связь двоичной, восьмеричной и шестнадцатеричной систем счисления

На рисунке 1.1 представлена схема, по которой переводятся числа между системами счисления, основаниями которых являются степени двойки.

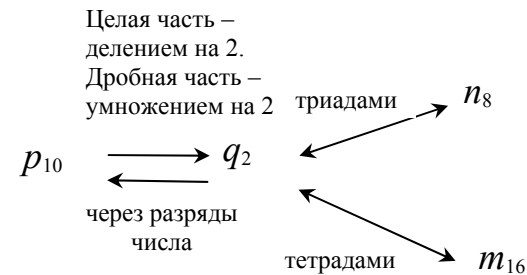


Рисунок 1.1 – Связь двоичной, восьмеричной и шестнадцатеричной систем счисления

1.6 Перевод чисел из системы счисления с основанием p в систему счисления с основанием q

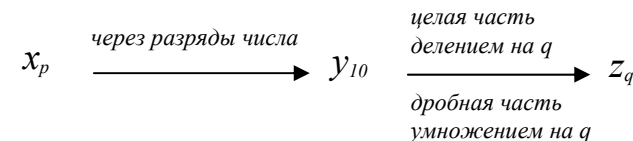


Рисунок 1.2 – Правило перевода чисел из системы счисления с основанием p в систему счисления с основанием q

Пример. Перевести число из системы счисления с основанием 6 в систему счисления с основанием 11: $325,4_6 = x_{11}$.

Переводим число из шестеричной системы счисления в десятичную систему счисления:

$$325,4_6 = 3 \cdot 6^2 + 2 \cdot 6^1 + 5 \cdot 6^0 + 4 \cdot 6^{-1} = 108 + 12 + 5 + 0,66 = 125,66_{10}$$

Из десятичной системы счисления переводим число в одиннадцатеричную систему счисления.

$$\begin{array}{r} 125 \mid 11 \\ \underline{121} \mid 11 \mid 11 \\ 4 \mid 11 \mid 1 \\ 0 \end{array} \quad \begin{array}{r} 0,66 \\ \underline{11} \\ 7,26 \\ \underline{11} \\ 2,86 \\ \underline{11} \\ 9,46 \\ \underline{11} \\ 5,06 \\ \underline{11} \\ 0,66 \end{array}$$

$$325,4_6 = 104,(72950)_{11}$$

1.7 Представление отрицательных чисел

Элементарной единицей хранения информации в компьютере является бит – один двоичный разряд, принимающий значение 0 или 1. Группа из восьми бит называется байтом. Два соседних в памяти байта – слово. Биты в словах и в байтах нумеруются справа налево, начиная с нуля. Наряду с байтами и словами используются такие единицы хранения информации как килобайт (1024 байта) и мегабайт (1024 килобайта).

Будем рассматривать представление целых чисел длиной в слово:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Старший байт								Младший байт							

Существует несколько способов представления отрицательных чисел: «знак+модуль», «дополнение до единицы», «двоичное дополнение».

1.7.1 Знак + модуль

Если длина машинного слова $n = 16$ бит, то модуль числа представляется кодом из $n - 1 = 15$ двоичных разрядов, а крайний левый бит слова используется для кодирования знака числа: 0 – «+», 1 – «-».

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
знак	модуль														

Максимальное положительное число при $n = 16$ равно $2^{15} - 1$.

Недостаток такого способа представления проявляется при сложении или вычитании двух чисел. Если a и b – целые числа, то они складываются следующим образом. Если знаки чисел одинаковы, то модули чисел складываются, а сумме приписывается знак, общий для двух чисел. Если же знаки чисел различны, то необходимо определить большее по модулю число, из модуля большего числа вычесть модуль меньшего числа и поставить знак большего по модулю числа.

Пример сложения чисел способом «знак + модуль»: $-27 + 18 = -9$.

$$\begin{aligned} 27_{10} &= 0000000000011011_2 \\ -27_{10} &= 1000000000011011_2 \\ 18_{10} &= 000000000010010_2 \\ + -27_{10} &= 1000000000011011_2 \\ 18_{10} &= 000000000010010_2 \\ -9_{10} &= 100000000001001_2 \end{aligned}$$

1.7.2 Дополнение до единицы

Для представления отрицательного числа берется его положительное представление и слева дополняется нулями до количества разрядов,

соответствующего одному машинному слову. После этого каждый нуль заменяется на единицу, а каждая единица – на нуль.

Для трехбитовых чисел дополнение до единицы определяется так:

<u>Число:</u>	<u>Дополнение до единицы:</u>
000 – +0	111 – –0
001 – +1	110 – –1
010 – +2	101 – –2
011 – +3	100 – –3

Если a и b – числа, представленные в виде дополнения до единицы, то при их сложении знаки игнорируются, а числа складываются. Будет получена сумма такой же длины, как и исходные числа, но возможен перенос в левый разряд. В этом случае полученный разряд просто прибавляется к сумме. Такая процедура называется «циклическим переносом». При вычитании одного числа из другого берут дополнение до единицы вычитаемого, после чего два числа складывают.

Преимущество «дополнения до единицы» состоит в том, что при сложении и вычитании знаки чисел могут не учитываться.

Недостатки «дополнения до единицы»:

- 1) при введении отрицательных чисел вдвое сокращается диапазон представляемых таким образом абсолютных значений чисел;
- 2) образуется два различных нуля, не эквивалентных в битовой записи;

3) возможен циклический перенос.

Примеры:

+1	+001	+3	+011
<u>-2</u>	<u>101</u>	<u>-2</u>	<u>101</u>
-1	110	1	1 000
			— 1
			001

$$\begin{aligned} 27_{10} &= 0000000000011011_2 \\ -27_{10} &= 1111111111100100_2 \\ 18_{10} &= 000000000010010_2 \\ + -27_{10} &= 1111111111100100_2 \\ 18_{10} &= 000000000010010_2 \\ - ?_{10} &= 111111111110110_2 \\ 9_{10} &= 000000000001001_2 \end{aligned}$$

1.7.3 Двоичное дополнение

Чтобы найти двоичное представление отрицательного числа, необходимо взять его положительную форму, обратить каждый бит, а затем добавить к полученному результату единицу.

Пример:

$$\begin{array}{r} 27_{10} = 0000000000011011_2 \\ + 11111111111100100_2 \\ \hline 1_2 \\ -27_{10} = 11111111111100101_2 \end{array}$$

$$18_{10} = 0000000000010010_2$$

$$\begin{array}{r} + -27_{10} = 11111111111100101_2 \\ 18_{10} = 0000000000010010_2 \\ - ?_{10} = 1111111111110111_2 \\ + 000000000001000_2 \\ \hline 1_2 \\ 9_{10} = 000000000001001_2 \end{array}$$

1.8 Практическое задание

1 Заполнить таблицу, в каждой строке которой одно и то же число должно быть записано в системах счисления с основаниями 2, 8, 10, 16.

Вариант 1	2 с/с	8 с/с	10 с/с	16 с/с
	1001110101,111			
		564,22		
			941,2	
				A21,21

Вариант 2	2 с/с	8 с/с	10 с/с	16 с/с
	1100011101,111			
		461,21		
			541,2	
				B21,1

Вариант 3	2 с/с	8 с/с	10 с/с	16 с/с
	1001100100,101			
		164,22		
			541,1	
				B21,22

Вариант 4	2 с/с	8 с/с	10 с/с	16 с/с
	1100000001,101			
		161,21		
			511,2	
				B2A,1

Вариант 5	2 с/с	8 с/с	10 с/с	16 с/с
	1111110111,111			
		565,25		
			545,2	
				5A1,21

Вариант 6	2 с/с	8 с/с	10 с/с	16 с/с
	1100111101,001			
		661,21		
			546,2	
				B61,1

Вариант 7	2 с/с	8 с/с	10 с/с	16 с/с
	1101111101,111			
		567,27		
			947,2	
				A71,27

Вариант 8	2 с/с	8 с/с	10 с/с	16 с/с
	11000011101,111			
		421,21		
			841,2	
				B28,1

Вариант 9	2 с/с	8 с/с	10 с/с	16 с/с
	1001100101,001			
		261,22		
			149,2	
				D1,21

Вариант 10	2 с/с	8 с/с	10 с/с	16 с/с
	1100000001,111			
		761,21		
			510,2	
				B10,1

Вариант 11	2 с/с	8 с/с	10 с/с	16 с/с
	1001111101,11			
		1564,22		
			41,2	
				A1,21

Вариант 12	2 с/с	8 с/с	10 с/с	16 с/с
	110001111101,111			
		61,21		
			548,2	
			F21,1	

Вариант 13	2 с/с	8 с/с	10 с/с	16 с/с
	1001110101,11			
		164,22		
			841,2	
			A2B,21	

Вариант 14	2 с/с	8 с/с	10 с/с	16 с/с
	110001101,111			
		61,21		
			41,2	
			B2,1	

Вариант 15	2 с/с	8 с/с	10 с/с	16 с/с
	1001110101,1			
		564,21		
			94,2	
			F2,21	

Вариант 16	2 с/с	8 с/с	10 с/с	16 с/с
	110001111101,111			
		161,21		
			141,2	
			1C21,1	

Вариант 17	2 с/с	8 с/с	10 с/с	16 с/с
	100111011101,111			
		764,22		
			441,2	
			A31,21	

Вариант 18	2 с/с	8 с/с	10 с/с	16 с/с
	1100011101,100			
		111,21		
			141,2	
			221,1	

Вариант 19	2 с/с	8 с/с	10 с/с	16 с/с
	1001110000,111			
		664,22		
			331,2	
			A22,21	

Вариант 20	2 с/с	8 с/с	10 с/с	16 с/с
	1100000000,111			
		141,21		
			741,2	
			F21,1	

Вариант 21	2 с/с	8 с/с	10 с/с	16 с/с
	1001000111,111			
		464,25		
			543,2	
			A81,29	

Вариант 22	2 с/с	8 с/с	10 с/с	16 с/с
	1101101101,111			
		765,21		
			943,2	
			B63,1	

Вариант 23	2 с/с	8 с/с	10 с/с	16 с/с
	1001010111,111			
		164,22		
			541,2	
			A27,21	

Вариант 24	2 с/с	8 с/с	10 с/с	16 с/с
	1011110001,111			
		361,21		
			546,2	
			B81,1	

Вариант 25	2 с/с	8 с/с	10 с/с	16 с/с
	1001101101,101			
		364,22		
			641,1	
			B25,22	

Вариант 26	2 с/с	8 с/с	10 с/с	16 с/с
	1011010101,101			
		464,21		
			981,2	A71,21

Вариант 27	2 с/с	8 с/с	10 с/с	16 с/с
	1101011101,111			
		761,21		
			551,2	C21,1

Вариант 28	2 с/с	8 с/с	10 с/с	16 с/с
	1111100100,101			
		564,22		
			545,1	B25,22

Вариант 29	2 с/с	8 с/с	10 с/с	16 с/с
	1100101001,101			
		151,21		
			711,2	B8A,1

Вариант 30	2 с/с	8 с/с	10 с/с	16 с/с
	11001110111,11			
		561,25		
			345,2	5A6,21

2 Произвести действия в двоичной системе счисления.

Вариант 1	100101001,011 + 110111,111 100101001,011 · 111111,111	101101001,011 – 110111,101 100111001,011 / 110111,11
Вариант 2	101101000,011 + 110110,111 100100001,011 · 111011,111	101101001,011 – 101111,111 101101011,011 / 110111,011
Вариант 3	101111001,011 + 110111,111 100101011,011 · 111101,111	101111001,111 – 110110,101 100111001,011 / 110101,11
Вариант 4	111101001,011 + 100110,111 100110101,011 · 101011,111	101001001,001 – 101101,111 101111011,011 / 110111,011
Вариант 5	110101001,011 + 110111,111 100101101,011 · 110111,011	101100001,001 – 110101,101 100101001,011 / 110101,11

Вариант 6	1010101000,011 + 10110,111 10010111,011 · 11101,111	101001011,011 – 101111,111 101101111,011 / 1101,011
Вариант 7	101101011,011 + 110110,111 100111001,011 · 101111,111	101001011,011 – 110101,101 100111101,011 / 111111,11
Вариант 8	101101000,011 + 110110,111 100100001,011 · 111011,111	101111001,011 – 101111,111 101101010,011 / 110110,011
Вариант 9	1111101001,011 + 110111,111 1011101001,011 · 111111,111	101111001,111 – 110110,101 100111011,011 / 110110,11
Вариант 10	101111000,011 + 110110,111 1001010001,011 · 11011,111	10111111001,011 – 101111,111 1011010111,011 / 110111,011
Вариант 11	111101001,011 + 110111,111 10001001,011 · 111111,111	101111111,011 – 110111,101 10011101,011 / 11111,11
Вариант 12	101101000,011 + 110110,111 100100001,011 · 111011,111	101101001,011 – 101111,111 101101011,011 / 110111,011
Вариант 13	10101001,011 + 10111,111 10101001,011 · 11111,111	10101001,011 – 110111,101 100111001,011 / 10111,11
Вариант 14	101101000,011 + 11010,111 100100001,011 · 11011,111	101101001,011 – 101111,111 10101011,011 / 1101,011
Вариант 15	10001001,011 + 110111,111 1001001,011 · 111111,111	101101001,011 – 110111,101 1001001,011 / 1111,11
Вариант 16	111101000,011 + 110110,111 110100001,011 · 111011,111	101111001,011 – 101111,111 101111011,011 / 110111,011
Вариант 17	111111001,011 + 11011,111 100111111,011 · 11111,111	101100001,011 – 110111,101 100101001,011 / 110111,11
Вариант 18	10110100,011 + 110110,111 10010001,011 · 111011,111	101101001,011 – 10111,111 10110011,011 / 11011,011
Вариант 19	111101001,011 + 10111,111 111101001,011 · 11111,111	101111001,011 – 110111,101 100111111,011 / 10111,11
Вариант 20	1101000,011 + 110110,111 100001,011 · 111011,111	1101001,011 – 10111,111 1101011,011 / 1111,011
Вариант 21	111101001,011 + 110111,111 1111101001,011 · 110111,111	111101001,011 – 110101,101 110111001,011 / 110101,11
Вариант 22	101101011,011 + 110100,111 100100101,011 · 111001,111	101101011,011 – 101101,111 101101111,011 / 110101,011
Вариант 23	101001001,011 + 110101,111 100111011,011 · 101011,111	101111001,011 – 100111,101 110111001,011 / 110011,11

Вариант 24	$101110010,011 + 110110,111$ $100111001,011 \cdot 111011,111$	$101101111,011 - 101111,111$ $101001111,011 / 110111,011$
Вариант 25	$110011001,011 + 110111,111$ $101111011,011 \cdot 111101,111$	$110011001,111 - 110110,101$ $111001001,011 / 110101,11$
Вариант 26	$110101101,011 + 110111,111$ $100111001,011 \cdot 111011,111$	$101101101,011 - 110110,101$ $100110001,011 / 11011,11$
Вариант 27	$111101000,011 + 110110,111$ $100101001,011 \cdot 111011,111$	$101101101,011 - 101111,111$ $101101111,011 / 110111,011$
Вариант 28	$111111001,011 + 110111,111$ $110101011,011 \cdot 111101,111$	$111111001,111 - 110110,101$ $110111001,011 / 110101,11$
Вариант 29	$111101011,011 + 100110,111$ $101110101,111 \cdot 101011,111$	$101011001,001 - 101101,111$ $101101011,011 / 110111,011$
Вариант 30	$110111011,011 + 110111,111$ $101101101,011 \cdot 110111,011$	$101100111,001 - 110101,101$ $100101101,011 / 110101,11$

3 Произвести действия в восьмеричной системе счисления.

Вариант 1	$526352,34 + 576,22$ $536352,34 \cdot 516,22$	$526372,34 - 541,22$ $536152,34 / 546,23$
Вариант 2	$516353,31 + 542,22$ $536152,34 \cdot 545,22$	$536752,34 - 536,2$ $436352,34 / 146,22$
Вариант 3	$122352,34 + 376,22$ $236352,34 \cdot 216,22$	$326372,34 - 521,22$ $536132,34 / 146,23$
Вариант 4	$716153,31 + 642,22$ $636132,34 \cdot 565,26$	$536762,34 - 566,2$ $436652,34 / 144,24$
Вариант 5	$125352,34 + 176,25$ $336352,34 \cdot 516,52$	$226375,34 - 541,25$ $336152,34 / 546,23$
Вариант 6	$616653,31 + 542,62$ $536162,34 \cdot 645,22$	$636752,34 - 136,2$ $436652,34 / 546,26$
Вариант 7	$724352,34 + 576,22$ $536752,34 \cdot 576,22$	$576372,34 - 547,22$ $536157,34 / 546,27$
Вариант 8	$526353,31 + 242,22$ $236152,34 \cdot 515,22$	$736752,34 - 536,2$ $336312,34 / 346,22$
Вариант 9	$523352,34 + 576,22$ $536352,34 \cdot 516,22$	$526372,34 - 541,22$ $536152,34 / 546,23$
Вариант 10	$1016353,31 + 52,12$ $537152,34 \cdot 541,22$	$536752,34 - 536,2$ $466357,34 / 176,22$

Вариант 11	$522352,34 + 576,22$ $53352,34 \cdot 516,22$	$52632,34 - 541,22$ $36152,34 / 546,23$
Вариант 12	$516353,31 + 542,22$ $536152,34 \cdot 545,22$	$536752,34 - 536,2$ $436352,34 / 146,22$
Вариант 13	$51352,34 + 57,22$ $36352,34 \cdot 516,22$	$52672,34 - 541,22$ $536152,34 / 56,23$
Вариант 14	$51353,31 + 542,22$ $36152,34 \cdot 545,22$	$53672,34 - 536,2$ $436352,34 / 14,22$
Вариант 15	$126352,34 + 576,22$ $136352,34 \cdot 516,22$	$726372,34 - 541,22$ $636152,34 / 546,23$
Вариант 16	$116353,31 + 542,22$ $136152,34 \cdot 545,22$	$236752,34 - 536,2$ $426352,34 / 146,22$
Вариант 17	$524352,34 + 576,22$ $531352,34 \cdot 516,22$	$526371,34 - 545,22$ $536152,34 / 516,13$
Вариант 18	$51353,31 + 52,22$ $53652,34 \cdot 54,22$	$53672,34 - 56,2$ $43632,34 / 14,22$
Вариант 19	$125352,34 + 576,22$ $516352,34 \cdot 516,22$	$526372,14 - 541,22$ $336152,34 / 516,23$
Вариант 20	$616353,31 + 642,22$ $736152,34 \cdot 515,22$	$136652,34 - 536,2$ $436752,34 / 146,22$
Вариант 21	$716352,34 + 576,22$ $173352,34 \cdot 516,22$	$26372,34 - 541,22$ $36152,34 / 546,23$
Вариант 22	$316353,31 + 512,22$ $336152,34 \cdot 145,22$	$136752,34 - 536,2$ $736352,34 / 146,22$
Вариант 23	$525352,34 + 516,22$ $532352,34 \cdot 566,25$	$126372,34 - 341,22$ $736152,34 / 746,23$
Вариант 24	$716353,31 + 542,22$ $636152,34 \cdot 545,22$	$336752,34 - 536,2$ $136352,34 / 146,22$
Вариант 25	$135352,34 + 376,22$ $237352,34 \cdot 216,22$	$316372,34 - 521,22$ $536432,34 / 146,23$
Вариант 26	$516372,34 + 516,22$ $526332,34 \cdot 515,22$	$516372,34 - 141,22$ $736152,34 / 146,23$
Вариант 27	$716353,31 + 542,22$ $736152,34 \cdot 545,22$	$736752,34 - 536,2$ $736352,34 / 146,22$
Вариант 28	$526352,34 + 376,22$ $536352,34 \cdot 216,22$	$526372,34 - 521,22$ $536532,34 / 146,23$

Вариант 29	$716453,31 + 642,22$ $646132,34 \cdot 565,26$	$536764,34 - 566,2$ $436452,34 / 144,24$
Вариант 30	$125652,34 + 176,25$ $136352,34 \cdot 516,52$	$726375,34 - 541,25$ $536152,34 / 546,23$

4 Произвести действия в шестнадцатеричной системе счисления.

Вариант 1	$A54D,346 + D1,F$ $A34D,346 \cdot D3,F$	$A79D,316 - D2,F$ $A34D,346 / D4,F$
Вариант 2	$A88D,346 + D2,F$ $A44D,346 \cdot D8,F$	$A37D,346 - D9,F$ $A54D,346 / D3,F$
Вариант 3	$B54D,346 + D2,F$ $A34D,316 \cdot D3,F$	$B79D,316 - D1,F$ $B34D,346 / C4,F$
Вариант 4	$A84D,344 + D4,F$ $A44D,344 \cdot D8,F$	$A34D,346 - D9,F$ $A54D,346 / D4,F$
Вариант 5	$A545,346 + D1,F$ $534D,346 \cdot D3,F$	$A79D,516 - D2,5$ $A34D,345 / D4,F$
Вариант 6	$A86D,346 + D2,F$ $664D,346 \cdot D1,F$	$A67D,646 - 69,F$ $A14D,346 / D6,F$
Вариант 7	$754D,746 + C1,F$ $734D,376 \cdot C3,F$	$A797,316 - D7,F$ $734D,346 / 74,F$
Вариант 8	$A18D,346 + D2,F$ $A84D,346 \cdot D8,F$	$A837D,346 - D9,F$ $854D,346 / D1,F$
Вариант 9	$A59D,346 + D9,F$ $934D,346 \cdot 9D,F$	$A9D,316 - D2,F$ $9A3D,346 / 9D,F$
Вариант 10	$C88D,346 + D2,A$ $C44D,346 \cdot D8,A$	$C37D,346 - D9,F$ $C54D,346 / D3,F$
Вариант 11	$F5D,36 + D1,F$ $AFD,346 \cdot D3,F$	$AFD,316 - D2,F$ $A34D,36 / DF,F$
Вариант 12	$A88D,346 + D2,F$ $A44D,346 \cdot D8,F$	$A37D,346 - D9,F$ $A54D,346 / D3,F$
Вариант 13	$A5D,346 + D1,F$ $A34D,346 \cdot D3,F$	$A9D,316 - D2,F$ $A34D,346 / D,F$
Вариант 14	$A8D,346 + D2,F$ $A40D,346 \cdot D8,F$	$A37D,346 - D9,F$ $A4D,346 / D,F$
Вариант 15	$B54D,346 + D1,F$ $AB4D,346 \cdot D3,F$	$A79D,3B6 - D2,F$ $A34D,346 / D1,F$

Вариант 16	$A28D,346 + D2,F$ $A34D,346 \cdot D8,F$	$A47D,346 - D9,F$ $A64D,346 / D3,F$
Вариант 17	$D54D,346 + D1,F$ $34D,346 \cdot D3,F$	$A9D,316 - D2,F$ $A34,346 / D,F$
Вариант 18	$A8D,346 + D2,F$ $A4D,346 \cdot D8,F$	$A37,346 - D9,F$ $A5D,346 / D3,F$
Вариант 19	$C54D,346 + D1,F$ $A4D,346 \cdot 3,F$	$79D,36 - D2,F$ $A3D,36 / D4,F$
Вариант 20	$988D,346 + 92,F$ $944D,346 \cdot D9,F$	$937D,346 - D9,F$ $A94D,346 / D9,F$
Вариант 21	$A14D,346 + D1,F$ $A84D,346 \cdot D3,F$	$A29D,316 - D2,F$ $A64D,346 / D4,F$
Вариант 22	$A18D,346 + D2,F$ $A74D,346 \cdot D8,F$	$A97D,346 - D9,F$ $A24D,346 / D3,F$
Вариант 23	$A1D,346 + D1,F$ $A36D,346 \cdot D3,F$	$A19D,316 - D1,F$ $A84D,346 / D4,F$
Вариант 24	$A78D,346 + D2,F$ $A45D,346 \cdot D8,F$	$A27D,346 - D9,F$ $A84D,346 / D3,F$
Вариант 25	$B56D,346 + D2,F$ $A35D,316 \cdot D3,F$	$B69D,316 - D1,F$ $B54D,346 / C4,F$
Вариант 26	$A84D,346 + D1,F$ $A39D,346 \cdot D3,F$	$A19D,316 - D2,F$ $A74D,346 / D4,F$
Вариант 27	$A89D,346 + D2,F$ $A94D,346 \cdot D8,F$	$A39D,346 - D9,F$ $A94D,346 / D3,F$
Вариант 28	$B55D,356 + D2,F$ $A54D,316 \cdot D3,F$	$B79D,356 - D1,F$ $B34D,356 / C4,F$
Вариант 29	$A84D,347 + D4,F$ $A46D,344 \cdot D8,F$	$A39D,346 - D9,F$ $A55D,346 / D4,F$
Вариант 30	$A548,346 + D1,F$ $539D,346 \cdot D3,F$	$A49D,516 - D2,5$ $A24D,345 / D4,F$

5 Перевести число из системы счисления с основанием p в систему счисления с основанием q .

Вариант 1	$121112,11_3 \rightarrow x_{11}$	Вариант 2	$321312,13_4 \rightarrow x_{12}$
Вариант 3	$12313,11_5 \rightarrow x_{11}$	Вариант 4	$121313,13_4 \rightarrow x_{13}$
Вариант 5	$127912,11_{11} \rightarrow x_4$	Вариант 6	$521511,13_6 \rightarrow x_{12}$

Вариант 7	6543,11 ₇ → x ₁₁	Вариант 8	321312,13 ₄ → x ₈
Вариант 9	823152,11 ₉ → x ₁₃	Вариант 10	A124,13 ₁₄ → x ₄
Вариант 11	1122,12 ₃ → x ₁₁	Вариант 12	11312,13 ₄ → x ₁₂
Вариант 13	5678,11 ₁₂ → x ₉	Вариант 14	321312,43 ₅ → x ₁₂
Вариант 15	121112,11 ₃ → x ₁₇	Вариант 16	331312,13 ₄ → x ₁₅
Вариант 17	ABC,11 ₁₇ → x ₁₁	Вариант 18	425382,13 ₉ → x ₁₂
Вариант 19	120012,11 ₃ → x ₁₅	Вариант 20	14312,13 ₅ → x ₁₂
Вариант 21	122122,11 ₃ → x ₁₁	Вариант 22	121313,13 ₄ → x ₁₂
Вариант 23	122122,11 ₃ → x ₁₁	Вариант 24	323112,13 ₄ → x ₁₂
Вариант 25	14343,11 ₅ → x ₁₁	Вариант 26	121212,11 ₃ → x ₁₁
Вариант 27	321332,13 ₄ → x ₁₂	Вариант 28	12321,11 ₅ → x ₁₁
Вариант 29	321313,13 ₄ → x ₁₂	Вариант 30	139162,11 ₁₁ → x ₄

6 Используя двоичное представление чисел, произвести сложение чисел, используя 3 способа представления целых чисел:

- 1) «знак + модуль»;
- 2) «дополнение до единицы»;
- 3) «двоичное дополнение».

Вариант 1	-45 + 23	Вариант 2	-54 + 21
Вариант 3	-55 + 12	Вариант 4	-49 + 22
Вариант 5	-64 + 11	Вариант 6	-64 + 71
Вариант 7	-47 + 21	Вариант 8	-48 + 18
Вариант 9	-94 + 123	Вариант 10	-54 + 41
Вариант 11	-55 + 123	Вариант 12	-154 + 21
Вариант 13	-45 + 12	Вариант 14	-14 + 21
Вариант 15	-45 + 27	Вариант 16	-74 + 11
Вариант 17	-175 + 23	Вариант 18	-94 + 19
Вариант 19	-15 + 23	Вариант 20	-74 + 16
Вариант 21	-49 + 27	Вариант 22	-54 + 29
Вариант 23	-47 + 23	Вариант 24	-56 + 21
Вариант 25	-51 + 12	Вариант 26	-54 + 23
Вариант 27	-45 + 27	Вариант 28	-57 + 12
Вариант 29	-94 + 22	Вариант 30	-46 + 11

Тема 2. Программная модель микропроцессора. Регистры

Любая выполняемая программа получает в свое распоряжение определенный набор ресурсов микропроцессора. Эти ресурсы необходимы для выполнения и хранения в памяти команд программы, данных и информации о текущем состоянии программы и микропроцессора. Набор этих ресурсов представляет собой программную модель микропроцессора. На рисунке 2.1 представлена программная модель микропроцессора Pentium III.

Программную модель микропроцессора Intel составляют:

- пространство адресуемой памяти (для Pentium III – до $2^{36}-1$ байт);
 - набор регистров для хранения данных общего назначения;
 - набор сегментных регистров;
 - набор регистров состояния и управления;
 - набор регистров устройства вычислений с плавающей точкой (сопроцессора);
 - набор регистров целочисленного MMX-расширения, отображенных на регистры сопроцессора (впервые появились в архитектуре микропроцессора Pentium MMX);
 - набор регистров MMX-расширения с плавающей точкой (впервые появились в архитектуре микропроцессора Pentium III);
 - программный стек. Это специальная информационная структура, работа с которой предусмотрена на уровне машинных команд.
- MMX-расширение микропроцессора Pentium предназначено для поддержки приложений, ориентированных на работу с большими массивами данных целого и вещественного типов, над которыми выполняются одинаковые операции. С данными такого типа обычно работают мультимедийные, графические, коммуникационные системы. Поэтому данное расширение названо MultiMedia eXtension (MMX).
- MMX-расширение целочисленного устройства называется MMX-расширением.
- MMX-расширение вещественного устройства называется XMM-расширением.

2.1 Наборы регистров

Регистры можно разделить на две части:

- 1) пользовательские регистры;
- 2) системные регистры.

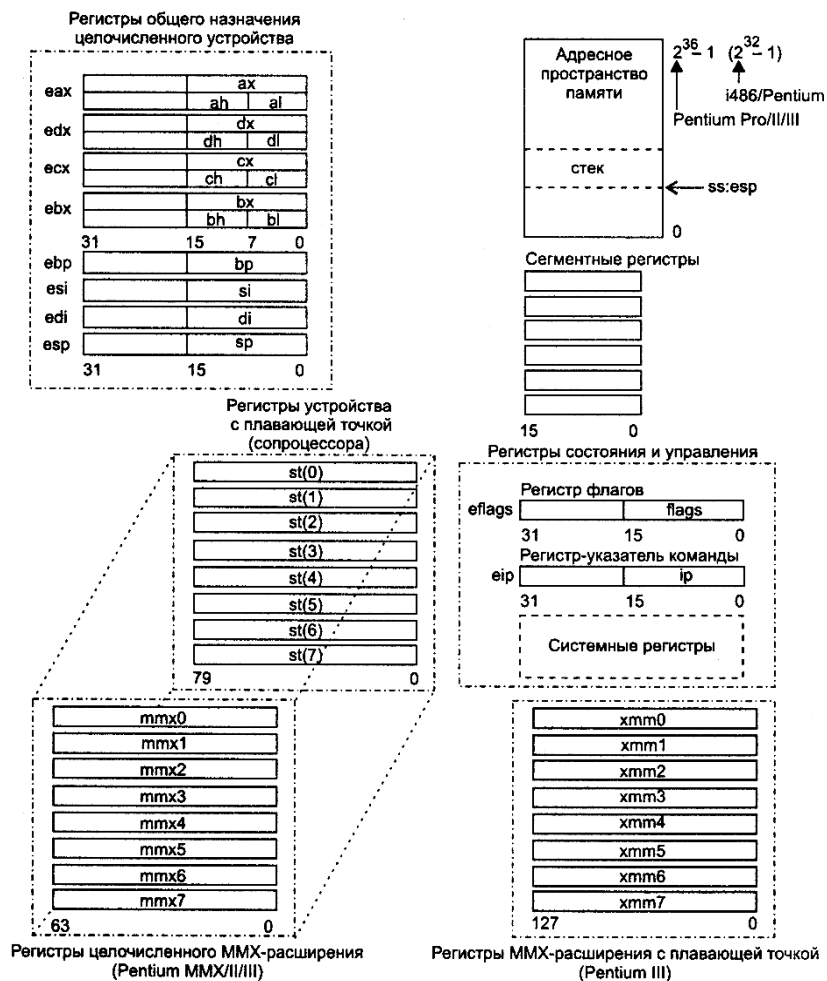


Рисунок 2.1 – Программная модель микропроцессора Intel (Pentium III)

К **пользовательским регистрам** относятся:

- регистры общего назначения;
- индексные регистры;
- регистры для работы со стеком;
- сегментные регистры;
- регистры сопроцессора;
- целочисленные регистры MMX-расширения;
- регистры MMX-расширения с плавающей точкой;
- регистры состояния и управления.

Регистры общего назначения. Регистры этой группы используются для хранения данных и адресов. К ним относятся:

- eax/ax/ah/al (Accumulator register) – аккумулятор. Применяется для хранения промежуточных данных;
- ebx/bx/bh/bl (Base register) – базовый регистр. Применяется для хранения базового адреса некоторого объекта в памяти;
- ecx/cx/ch/cl (Count register) – регистр-счетчик. Применяется в командах, производящих некоторые повторяющиеся действия;
- edx/dx/dh/dl (Data register) – регистр данных. Так же, как и регистр eax/ax/ah/al, он хранит промежуточные данные.

Индексные регистры:

- esi/si (Source Index register) – индекс источника. Этот регистр в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике;
- edi/di (Destination Index register) – индекс приемника (получателя). Этот регистр в цепочечных операциях содержит текущий адрес в цепочке-приемнике.

Регистры для работы со стеком:

- esp/sp (Stack Pointer register) – регистр указателя стека. Содержит указатель вершины стека в текущем сегменте стека;
- ebp/bp (Base Pointer register) – регистр указателя базы кадра стека. Предназначен для организации произвольного доступа к данным внутри стека.

Сегментные регистры. Регистры этой группы используются для хранения адресов сегментов в памяти. В программной модели микропроцессора имеется шесть сегментных регистров: cs, ss, ds, es, gs, fs. Микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых сегментами. Соответственно, такая организация памяти называется сегментной. Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры:

- cs (code segment register) – сегментный регистр кода. Содержит команды программы;
- ds (data segment register) – сегментный регистр данных, который хранит адрес сегмента данных текущей программы;
- ss (stack segment register) – сегментный регистр стека, содержащий адрес сегмента стека;
- дополнительный сегмент данных. Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре ds,

при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в регистрах *es*, *gs*, *fs* (extension data segment registers).

Регистры сопроцессора *st(0)*, *st(1)*, *st(2)*, *st(3)*, *st(4)*, *st(5)*, *st(6)*, *st(7)*. Регистры этой группы предназначены для написания программ, использующих тип данных с плавающей точкой.

Целочисленные регистры MMX-расширения *mmx0*, *mmx1*, *mmx2*, *mmx3*, *mmx4*, *mmx5*, *mmx6*, *mmx7*.

Регистры MMX-расширения с плавающей точкой *xmm0*, *xmm1*, *xmm2*, *xmm3*, *xmm4*, *xmm5*, *xmm6*, *xmm7*.

Регистры состояния и управления – это регистры, которые содержат информацию о состоянии микропроцессора, исполняемой программы и позволяют изменить это состояние:

– *eflags/flags* (flag register) – регистр флагов. Разрядность *eflags/flags* – 32/16 бит. Отдельные биты данного регистра имеют определенное функциональное назначение и называются флагами. Младшая часть этого регистра полностью аналогична регистру *flags* для *i8086*. На рисунке 2.2 показано содержимое регистра *eflags*.

Флаги регистра *eflags/flags* можно разделить на три группы:

1) 8 флагов состояния. Эти флаги могут изменяться после выполнения машинных команд. Флаги состояния регистра *eflags* отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм. В таблице 2.1 приведены флаги состояния и указано их назначение;

2) 1 флаг управления. Обозначается *df* (Directory Flag). Он находится в десятом бите регистра *eflags* и используется цепочечными командами. Значение флага *df* определяет направление поэлементной обработки в этих операциях: от начала строки к концу (*df* = 0) либо, наоборот, от конца строки к ее началу (*df* = 1). Для работы с флагом *df* существуют специальные команды *cld* (снять флаг *df*) и *std* (установить флаг *df*). Применение этих команд позволяет привести флаг *df* в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьшение счетчиков при выполнении операций со строками;

3) 8 системных, флагов, управляющих вводом/выводом, маскируемыми прерываниями, отладкой, переключением между задачами и виртуальным режимом 8086. Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это приведет к прерыванию работы программы.

– *eip/ip* (Instruction Pointer register) – указатель команд. Регистр *eip/ip* имеет разрядность 32/16 бит и содержит смещение следующей

подлежащей выполнению команды относительно содержимого сегментного регистра *cs* в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур. Возникновение прерываний также приводит к модификации регистра *eip/ip*.



Рисунок 2.2 – Содержимое регистра *eflags*

Таблица 2.1 – Флаги состояния

Мнемоника флага	Флаг	Номер бита в eflags	Содержание и назначение
cf	Флаг переноса (Carry Flag)	0	1 – арифметическая операция произвела перенос из старшего бита результата. Старшим является 7-й, 15-й или 31-й бит и зависимости от размерности операнда. 0 – переноса не было
pf	Флаг паритета (Parity Flag)	2	1 – 8 младших разрядов (этот флаг – только для 8 младших разрядов операнда любого размера)

Продолжение таблицы 2.1

Мнемоника флага	Флаг	Номер бита в eflags	Содержание и назначение
			результата содержат четное число единиц; 0 – 8 младших разрядов результата содержат нечетное число единиц
af	Вспомогательный флаг переноса (Auxiliary carry Flag)	4	Только для команд работающих с BCD числами. Фиксирует факт заема из младшей тетрады результата: 1 – в результате операции сложения был произведен перенос из разряда 3 в старший разряд или при вычитании был заем в разряд 3 младшей тетрады из значения в старшей тетраде; 0 – переносов и заемов в (из) 3 разряд(а) младшей тетрады результата не было
zf	Флаг нуля (Zero Flag)	6	1 – результат нулевой; 0 – результат ненулевой
sf	Флаг знака (Sign Flag)	7	Отражает состояние старшего бита результата (биты 7, 15 или 31 для 8-, 16- или 32-разрядных операндов, соответственно): 1 – старший бит результата равен 1; 0 – старший бит результата равен 0
of	Флаг переполнения (Overflow Flag)	11	Флаг of используется для фиксирования факта потери значащего бита при арифметических операциях: 1 – в результате операции происходит перенос (заем) в (из) старшего, знакового бита результата (биты 7, 15 или 31 для 8-, 16- или 32-разрядных операндов, соответственно); 0 – в результате операции не происходит переноса (заёма) в (из) старшего, знакового бита результата

Продолжение таблицы 2.1

Мнемоника флага	Флаг	Номер бита в eflags	Содержание и назначение
iopl	Уровень привилегий ввода-вывода (Input/Output Privilege Level)	12, 13	Используется в защищенном режиме работы микропроцессора для контроля доступа к командам ввода-вывода, в зависимости от привилегированности задачи
nt	Флаг вложенности задачи (Nested Task)	14	Используется в защищенном режиме работы микропроцессора для фиксации факта, что одна задача вложена в другую
tf	Флаг трассировки (Trace Flag)	8	Предназначен для организации пошаговой работы микропроцессора: 1 – микропроцессор генерирует прерывание с номером 1 после выполнения каждой машинной команды. Может использоваться при отладке программ, в частности отладчиками; 0 – обычная работа
if	Флаг прерывания (Interrupt enable Flag)	9	Предназначен для разрешения или запрещения (маскирования) аппаратных прерываний (прерываний по входу INTR): 1 – аппаратные прерывания разрешены; 0 – аппаратные прерывания запрещены
rf	Флаг возобновления (Resume Flag)	16	Используется при обработке прерываний от регистров отладки
vm	Флаг виртуального 8086 (Virtual 8086 Mode)	17	Признак работы микропроцессора в режиме виртуального 8086: 1 – процессор работает в режиме виртуального 8086; 0 – процессор работает в реальном или защищенном режиме
ac	Флаг контроля выравнивания (Alignment Check)	18	Предназначен для разрешения контроля выравнивания при обращениях к памяти.

Окончание таблицы 2.1

Мнемоника флага	Флаг	Номер бита в eflags	Содержание и назначение
vif	Флаг виртуального прерывания (Virtual Interrupt Flag)	19	При определенных условиях (одно из которых – работа микропроцессора в V-режиме) является аналогом флага if. Флаг vif используется совместно с флагом vip.
vip	Флаг отложенного виртуального прерывания (Virtual Interrupt Pending flag)	20	Устанавливается в 1 для индикации отложенного прерывания. Используется при работе в V-режиме совместно с флагом vif.
id	Флаг идентификации (IDentification flag)	21	Используется для того, чтобы показать факт поддержки микропроцессором инструкции cruid.

Системные регистры – это регистры для поддержания различных режимов работы, сервисных функций, а также регистры, специфичные для определенной модели микропроцессора.

На схеме рисунка регистры этой группы не показаны по двум причинам: во-первых, их достаточно много, и, во-вторых, состав их может отличаться для различных моделей микропроцессора.

Многие из этих регистров приведены с наклонной разделительной чертой. Это не разные регистры – это части одного большого 32-рядного регистра. Их можно использовать в программе как отдельные объекты.

2.2 Организация памяти

Бит является наименьшей единицей измерения памяти в компьютере. 8 бит образуют байт. Каждому байту поставлен в соответствие уникальный адрес в диапазоне от 0000 до FFFF. Любые два смежных байта образуют слово. Слово состоит из 16 бит. Байт с большим номером содержит старшие биты слова, а байт с меньшим номером – младшие.

Вся память разбивается на блоки (сегменты), каждый из которых содержит не более 2^{16} (64 КБ). Сегменты начинаются по адресам, кратным 16, то есть имеющим четыре нулевых младших бита. В любой момент времени программе доступны 4 сегмента: сегмент кода, сегмент стека, сегмент данных, дополнительный сегмент. Каждый сегмент

определяется путем размещения старших 16 бит адреса первого байта сегмента в одном из четырех сегментных регистров. Таким образом, сегментный регистр содержит адрес начала сегмента.

Обращение к байтам или словам внутри сегмента осуществляется с помощью 16-битного внутрисегментного смещения. Микропроцессор образует 20-битовый адрес байта (слова), суммируя 16-битное смещение с содержимым 16-битного сегментного регистра, к которому добавляются четыре нуля:

Физический адрес = смещение + 16 · (сегментный регистр)

Умножение на 16 соответствует добавлению к содержимому сегмента 4 младших нулевых битов.

Микропроцессор аппаратно поддерживает две модели использования оперативной памяти:

1) *сегментированную модель*. В этой модели программе выделяются непрерывные области памяти (сегменты), а сама программа может обращаться только к данным, которые находятся в этих сегментах;

2) *страничную модель* можно рассматривать как надстройку над сегментированной моделью. В случае использования этой модели оперативная память рассматривается как совокупность блоков фиксированного размера (4 Кбайт и более). Основное применение этой модели связано с организацией виртуальной памяти, что позволяет операционной системе использовать для работы программ пространство памяти большее, чем объем физической памяти. Для процессоров i486 и Pentium размер возможной виртуальной памяти может достигать 4 Тбайт.

Сегментация – механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния. В основе механизма сегментации лежит понятие *сегмента*, который представляет собой независимый поддерживаемый на аппаратном уровне блок памяти.

Тема 3. Нотация языка Assembler. Программирование линейных вычислительных процессов

Любая команда языка ассемблер состоит из полей метки, мнемокода, операнда и комментария.

Формат команды:

[метка:] код [операнд] [;комментарий]

Обязательным в записи команды ассемблера является только поле мнемокода. Поля команды должны быть разделены хотя бы одним пробелом.

Метка может содержать до 31 символа и должна заканчиваться двоеточием. В имя метки могут входить:

- буквы от A до Z или от a до z;
- цифры от 0 до 9;
- специальные знаки: ? . @ _ \$

Метку можно начать любым символом, кроме цифры. Если используется точка, то она должна быть только первым символом метки.

Примеры меток:

```
Metka
.cikl
Get_cikl
```

Поле операнда может содержать источник и приемник, причем **приемник указывается первым**. Источник от приемника отделяются запятой:

Приемник, Источник

В ассемблере используются следующие константы:

- 1) двоичная – число, записанное в двоичной системе счисления, заканчивающееся буквой B, например: 1001010B;
- 2) десятичная – число, записанное в десятичной системе счисления, заканчивающееся буквой D, например: 1234D или 1234;
- 3) шестнадцатеричная – число, записанное в шестнадцатеричной системе счисления, заканчивающееся буквой H, например: 67ADH;
- 4) литерал – строка букв, цифр и других символов, заключенная в апострофы или кавычки, например: 'Ассемблер'.

В качестве констант можно использовать и отрицательные числа. В случае десятичного числа перед ним достаточно поставить знак минус. Если число двоичное или шестнадцатеричное, то его необходимо записать в дополнительном коде.

Комментарии используются для пояснения текста программы. Для задания многострочных комментариев нужно в начале каждой строки поставить точку с запятой.

3.1 Псевдооператоры

Псевдооператоры управляют работой ассемблера, а не микропроцессора. В отличие от команд ассемблера большинство псевдооператоров не генерируют объектного кода и выполняются на шаге трансляции программы.

Формат задания псевдооператора:

[идентификатор] псевдооператор [операнд]
[; комменарий]

Обязательным является только поле псевдооператора.

3.1.1 Псевдооператоры определения имен

Эти псевдооператоры позволяют программисту определить символические имена для часто используемых выражений. Такие имена создаются с помощью псевдооператоров EQU и =. После присваивания выражению имени можно использовать это имя всюду, где требуется указать выражение. Определенные псевдооператором = имена можно переопределить, а определенные псевдооператором EQU нельзя. Псевдооператор EQU можно использовать как с числовыми, так и с текстовыми выражениями, а псевдооператор = только с числовыми.

Пример:

```
count equ cx
const = 512
const = 2*const+1
```

3.1.2 Псевдооператоры определения переменных

Поименованные элементы данных, содержимое которых может быть изменено, называются переменными. Любая переменная имеет три атрибута: сегмент, смещение и тип.

Сегмент (SEG) определяет сегмент, содержащий переменную.

Смещение (OFFSET) представляет собой расстояние или дистанцию в байтах от начала сегмента.

Тип (TYPE) определяет единицу памяти, выделяемую для хранения переменной.

Псевдооператоры определения переменных:

```
[имя] db выражение [, ...]
[имя] dw выражение [, ...]
[имя] dd выражение [, ...]
```

Псевдооператор db резервирует байты, dw – слова, dd – двойные слова. Выражение может быть числовым или адресным.

Пример:

```
a db 4
b dw 6
c db 3*6-2 ; числовое выражение
```

d db a ; адресное выражение

При определении переменной без присваивания ей начального значения необходимо указать в поле выражения вопросительный знак

c db ?

Если имя переменной не указывается, то просто резервируется память:

db ?

Простые типы данных:

db – резервирование памяти размером в 1 байт;

dw – резервирование памяти размером в 2 байт;

dd – резервирование памяти размером в 4 байт;

df – резервирование памяти размером в 6 байт;

dp – резервирование памяти размером в 6 байт;

dq – резервирование памяти размером в 8 байт;

dt – резервирование памяти размером в 10 байт.

3.1.3 Псевдооператоры определения сегментов

В исходной программе должны быть определены сегменты, из которых состоит программа. Может быть определено до четырех сегментов: сегмент кода, сегмент данных, сегмент стека и дополнительный сегмент.

Формат описания сегмента:

```
Имя segment [тип подгонки] [тип связи] ['класс']
```

...

```
Имя ends
```

Существуют два типа подгонки или выравнивания сегмента:

para – сегмент будет расположен начиная с ячейки памяти, адрес которой кратен 16. Это значит, что первый байт сегмента будет иметь нулевое смещение относительно сегментного регистра;

byte – располагает сегмент начиная с любого адреса.

Тип связи определяет способ, которым сегмент объединяется с другими сегментами:

public – вызывает объединение всех сегментов с одним и тем же именем в виде одного большого сегмента;

at – располагает сегмент по заданному абсолютному адресу.

Класс определяет категорию сегмента:

code – сегмент кода;

data – сегмент данных;

extra – дополнительный сегмент;

stack – сегмент стека.

Псевдооператор segment не сообщает ассемблеру, какого рода сегмент должен быть определен. Для этой цели служит псевдооператор assume.

Формат оператора:

```
assume сегментный_регистр:имя_сегмента[, ...]
```

Оператор assume связывает каждый сегментный регистр с сегментом, который сегментный регистр адресует в данный момент.

Командой assume нельзя загрузить адрес начала сегмента в соответствующий сегментный регистр. Это осуществляется через регистр общего назначения.

Оператор assume помещается сразу же за оператором segment, определяющим сегмент кода:

```
cseg segment para public 'code'
    assume ds:dseg, cs:cseg, ss:sseg
    mov ax,dseg
    mov ds,ax
    ...
cseg ends
```

3.1.4 Псевдооператоры определения процедур

Формат описания процедуры:

```
имя_процедуры proc атрибут_дистанции
```

...

```
ret
```

```
имя_процедуры endp
```

Каждая процедура должна начинаться оператором proc и заканчиваться оператором endp. Процедура должна содержать команду возврата из процедуры ret.

Атрибуты дистанции:

far – дальняя процедура,

near – близкая процедура.

Процедура с атрибутом near может быть вызвана только из того сегмента команд, в котором она определена.

Формат вызова процедуры:

```
call имя_процедуры
```

3.1.5 Псевдооператоры управления трансляцией

Псевдооператор end отмечает конец исходной программы и указывает ассемблеру, где завершить трансляцию. Псевдооператор end должен присутствовать в каждой программе.

Формат оператора:

```
end [метка точки входа]
```

где метка определяет исходную программу. Например, оператор end proc

отмечает конец программы proc.

3.2 Команды пересылки

Формат команды:

```
mov приемник, источник ;И→П
```

Команда позволяет пересылать байт или слово между регистром и ячейкой памяти или между двумя регистрами:

```
mov регистр1, регистр2
mov регистр, память
```

В команде `mov` запрещается пересылка из одной ячейки памяти в другую.

mov память1, память2 ;НЕЛЬЗЯ!

Нельзя загрузить непосредственно адресуемый операнд в регистр сегмента. Это можно сделать только через регистр общего назначения **mov ds, dseg ;НЕЛЬЗЯ!**

```
mov ax, dseg ;НУЖНО!
mov ds, ax
```

Нельзя пересылать значение одного регистра сегмента в другой регистр сегмента. Подобная пересылка делается через регистр общего назначения.

mov ds, es ;НЕЛЬЗЯ!

```
mov ax, es ;НУЖНО!
mov ds, ax
```

Нельзя использовать регистр `CS` в качестве приемника в команде пересылки.

Для работы со стеком используются следующие команды:

```
push источник ;поместить слово в стек
pop приемник ;извлечь слово из стека
```

3.3 Арифметические команды

3.3.1 Команды сложения

Формат команды:

```
add приемник, источник ;П+И→П
```

Команда `add` используется для сложения приемника и источника, результат помещается в приемник.

Оба операнда не могут быть именами переменных. Источником может быть число.

Можно складывать:

```
ПАМЯТЬ+РЕГИСТР
РЕГИСТР+ПАМЯТЬ
РЕГИСТР+РЕГИСТР
```

```
ПАМЯТЬ+ЧИСЛО
РЕГИСТР+ЧИСЛО
```

Нельзя складывать ПАМЯТЬ + ПАМЯТЬ

```
inc приемник ;П+1→П
```

Команда `inc` используется для увеличения содержимого приемника на единицу. Приемник – это регистр или ячейка памяти.

3.3.2 Команды вычитания

Формат команды:

```
sub приемник, источник ;П-И→П
```

Команда `sub` используется для вычитания содержимого источника из содержимого приемника, результат помещается в приемник.

Оба операнда не могут быть именами переменных. Источником может быть число.

Можно вычитать:

```
ПАМЯТЬ – РЕГИСТР
РЕГИСТР – ПАМЯТЬ
РЕГИСТР – РЕГИСТР
ПАМЯТЬ – ЧИСЛО
РЕГИСТР – ЧИСЛО
```

Нельзя вычитать ПАМЯТЬ – ПАМЯТЬ

```
dec приемник ;П-1→П
```

Команда `dec` используется для уменьшения содержимого приемника на единицу. Приемник – это регистр или ячейка памяти.

3.3.3 Команды умножения

Формат команды умножения чисел без знака:

```
mul источник ;al*И→ax при работе с байтами
;ax*И→ax-dx при работе со словами
```

Формат команды умножения чисел со знаком:

```
imul источник ;al*И→ax при работе с байтами
;ax*И→ax-dx при работе со словами
```

Команды `mul` и `imul` умножают содержимое регистра `al` или `ax` (в зависимости от размерности операндов) на содержимое источника, указанного в команде умножения. Источником – это или регистр или ячейка памяти. В результате работы команд умножения с данными длиной байт регистр `ax` расширяется до `dx`, а с данными длиной слово – регистр `ax` расширяется до `dx`.

НЕЛЬЗЯ использовать в команде умножения в качестве источника непосредственное значение!

3.3.4 Команды деления

Формат команды деления чисел без знака:

```
div источник ;al/И→al-ah при работе с байтами
;ax/И→ax-dx при работе со словами
;al - частное, ah - остаток
;ax - частное, dx - остаток
```

Формат команды деления чисел со знаком:

```
idiv источник ;al/И→al-ah при работе с байтами
;ax/И→ax-dx при работе со словами
;al - частное, ah - остаток
;ax - частное, dx - остаток
```

Команды `div` и `idiv` делят содержимое регистра `al` или `ax` (в зависимости от размерности операндов) на содержимое источника, указанного в команде умножения. Источник – это или регистр или ячейка памяти. В результате работы команд умножения с данными длиной байт в регистр `al` помещается частное, в регистр `ah` – остаток, а с данными длиной слово – в регистр `ax` помещается частное, в регистр `dx` – остаток.

НЕЛЬЗЯ использовать в команде деления в качестве источника непосредственное значение!

3.4 Команды расширения знака

Форматы команд:

```
cbw ;расширить байт до слова
cwd ;расширить слово до двойного слова
```

Команда `cbw` воспроизводит 7 бит регистра `AL` во всех битах регистра `AH`.

Команда `cwd` воспроизводит 15 бит регистра `AX` во всех битах регистра `DX`.

3.5 Примеры использования арифметических команд

Вычислить значение выражения $y = \frac{7(a+7b)-3}{4}$.

;Программа работы с байтами

```
dseg segment para public 'data'
a db 3
b db 2
y db ?
mes db 'конец программы$'
dseg ends
```

```
sseg segment para stack 'stack'
db 30 dup (0)
sseg ends
cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov bl,4 ;bl=4
mov al,7 ;al=7
imul b ;al=7*b
add al,a ;al=a+7*b
mov cl,7 ;cl=7
imul cl ;al=7*(a+7*b)
sub al,3 ;al=7*(a+7*b)-3
cbw
idiv bl ;al=(7*(a+7*b)-3)/4 остаток в ah
mov y,al ;y=al
lea dx,mes ;вывод сообщения 'конец программы'
mov ax,0900H
int 21H
mov ax,4C00H ;завершение программы
int 21H
osn endp
cseg ends
end osn
```

;Программа работы со словами

```
dseg segment para public 'data'
a dw 3
b dw 2
y dw ?
mes db 'конец программы$'
dseg ends
sseg segment para stack 'stack'
db 30 dup (0)
sseg ends
cseg segment para public 'code'
osn proc near
assume cs:cseg,ds:dseg,ss:sseg
mov ax,dseg
mov ds,ax
mov bx,4 ;bx=4
mov ax,7 ;ax=7
imul b ;ax=7*b
```

```

add ax, a           ;ax=a+7*b
mov cx, 7           ;cx=7
imul cx             ;ax=7*(a+7*b)
sub ax, 3           ;ax=7*(a+7*b)-3
cwd
idiv bx             ;ax=(7*(a+7*b)-3)/4 остаток в dx
mov y, ax           ;y=ax
lea dx, mes        ;вывод сообщения 'конец программы'
mov ax, 0900H
int 21H
mov ax, 4C00H      ;завершение программы
int 21H
osn endp
cseg ends
end osn

```

3.6 Порядок работы с программой на ассемблере

Все действия с программой на ассемблере могут быть представлены в виде схемы, представленной на рисунке 3.1.

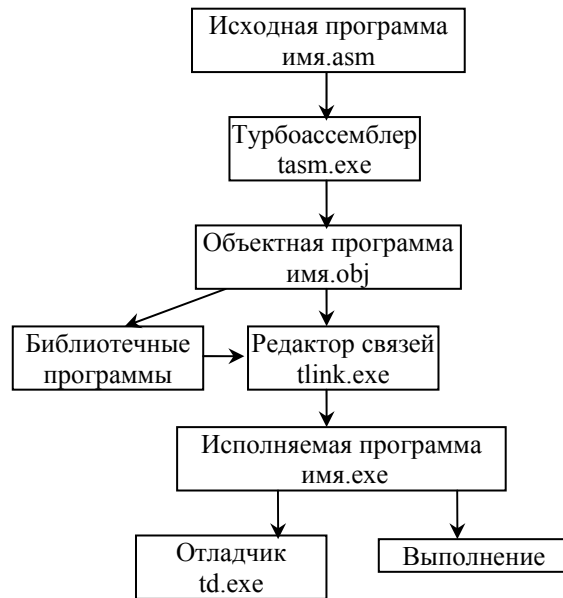


Рисунок 3.1 – Порядок работы с программой на ассемблере

Порядок работы с программой:

1. Исходный текст программы можно набрать, используя любой текстовый редактор.

2. Трансляция программы осуществляется с помощью транслятора ассемблера tasm.exe

tasm.exe имя.asm >>1.txt

Результатом трансляции является файл объектной программы имя.obj.

3. Создание исполняемого файла осуществляется с помощью редактора связей (загрузчика)

tlink.exe имя.obj

Редактор связей – это программа, осуществляющая связь объектных программ, оттранслированных с помощью ассемблера. Редактор связей объединяет все программы в один загрузочный модуль, готовый к выполнению. Результатом загрузки является файл загрузочного модуля имя.exe.

4. Исполнение программы возможно в среде MS DOS или с помощью отладчика:

td.exe имя.exe

Отладчик позволяет проверить работу программы.

3.7 Сокращенные директивы сегментации

Изложенные ранее директивы сегментации называются стандартными директивами сегментации.

Для простых программ, содержащих по одному сегменту кода, данных и стека, можно упростить их написание. Трансляторы masm и tasm ввели возможность использования упрощенных директив сегментации. Чтобы управлять в этом случае размещением сегментов, необходимо указывать директиву model, которая стала управлять размещением сегментов и выполнять функции директивы assume (поэтому при использовании упрощенных директив сегментации директиву assume можно не использовать). Эта директива связывает сегменты, которые, в случае использования упрощенных директив сегментации, имеют predetermined имена с сегментными регистрами.

Структура программы с использованием упрощенных директив сегментации:

```

masm
model small      ;модель памяти
;описание сегмента данных
.data
a db 3

```

```

    b db 2
    y db ?
mes db 'конец программы$'
;описание сегмента стека
.stack
    db 256 dup (0)
;описание сегмента кода
.code
    osn proc near ; начало основной процедуры
    mov ax,@data ; заносим адрес сегмента данных
                    ; в регистр ax
    mov ds,ax      ; ax в ds
;тело программы
    mov bl,4      ;bl=4
    mov al,7      ;al=7
    imul b        ;al=7*b
    add al,a      ;al=a+7*b
    mov cl,7      ;cl=7
    imul cl       ;al=7*(a+7*b)
    sub al,3      ;al=7*(a+7*b)-3
    cbw
    idiv bl       ;al=(7*(a+7*b)-3)/4 остаток в ah
    mov y,al      ;y=al
    lea dx,mes ;вывод сообщения 'конец программы'
    mov ax,0900H
    int 21H
;завершение программы
    mov ax,4C00H
    int 21H
    osn endp
end osn

```

Пакет макроассемблера `masm` позволяет задавать макроопределения (или макросы), представляющие собой именованные группы команд. Они обладают тем свойством, что их можно вставлять в программу в любом месте, указав только имя группы в месте вставки. Режим работы `masm` поддерживает все основные возможности макроассемблера.

Совместно с упрощенными директивами сегментации используется директива указания модели памяти `model`, которая управляет размещением сегментов и выполняет функции директивы `assume`.

Формат `model`:

`model` модификатор Модель памяти

Обязательным параметром директивы `model` является модель памяти. Этот параметр определяет модель сегментации памяти для программного модуля, т. е. набор сегментов программы, размер сегментов данных и кода, способ связывания сегментов и сегментных регистров.

Таблица 3.1 – Модели памяти

Модель	Тип кода	Тип данных	Назначение модели
tiny	near	near	Код и данные объединены в одну группу с именем <code>DGROUP</code> . Используется для создания программ формата <code>.com</code>
small	near	near	Код занимает один сегмент, данные объединены в одну группу с именем <code>DGROUP</code> .
medium	far	near	Код занимает несколько сегментов, по одному на каждый объединяемый программный модуль. Все ссылки на передачу управления типа <code>far</code> . Данные объединены в одной группе, все ссылки на них – типа <code>near</code>
compact	near	far	Код в одном сегменте, ссылки на данные – типа <code>far</code>
large	far	far	Код в нескольких сегментах, по одному на каждый объединяемый программный модуль
flat	near	near	Код и данные в одном 32-битном сегменте (плоская модель памяти)

Упрощенные директивы определения сегментов режима `masm`:

- `.code[размер]` – начало или продолжение сегмента кода;
- `.data` – начало или продолжение сегмента инициализированных данных;
- `.const` – начало или продолжение сегмента постоянных данных (констант);
- `.data?` – начало или продолжение сегмента неинициализированных данных;
- `.stack [размер]` – начало или продолжение сегмента стека;
- `.fardata [имя]` – начало или продолжение сегмента инициализированных данных типа `far`;
- `.fardata? [имя]` – начало или продолжение сегмента неинициализированных данных типа `far`.

Модификаторы:

use16 – сегменты используются как 16-битные;

use32 – сегменты используются как 32-битные;

dos – программа будет работать в MS-DOS.

При использовании директивы model транслятор делает доступными несколько идентификаторов, к которым можно обращаться во время работы программы:

@code – физический адрес сегмента кода;

@data – физический адрес сегмента данных типа near;

@fardata – физический адрес сегмента данных типа far;

@fardata? – физический адрес сегмента неинициализированных данных типа far;

@curseg – физический адрес сегмента неинициализированных данных типа far;

@stack – физический адрес сегмента стека.

3.8 Практическое задание

Составить программу на языке ассемблера для вычисления заданного арифметического выражения. Написать три программы:

1 – в программе использовать описание полных директив сегментации и данных длиной в байт;

2 – в программе использовать описание полных директив сегментации и данных длиной в слово;

3 – в программе использовать описание сокращенных директив сегментации, тип данных – любой.

Вариант 1	$y = \frac{7(a-b)+5}{3}$	Вариант 2	$y = \frac{7(a+b)-3}{2}$
Вариант 3	$y = \frac{2a+b-3}{8+a}$	Вариант 4	$y = \frac{2(a-3)+b}{7+b}$
Вариант 5	$y = \frac{2a+b-5}{b+2}$	Вариант 6	$y = \frac{3(a+b)-2}{6-b}$
Вариант 7	$y = \frac{6a+b-8}{7-a}$	Вариант 8	$y = \frac{2(3a-b)+6}{3b}$
Вариант 9	$y = \frac{(3a-b)+3a}{b-4}$	Вариант 10	$y = \frac{5(a-2b)+7}{3a+1}$
Вариант 11	$y = \frac{a(3a+b)-3}{5b-1}$	Вариант 12	$y = \frac{6(4a-b)+9}{4b-a}$

Вариант 13	$y = \frac{3a-2b+4}{3b-2}$	Вариант 14	$y = \frac{7(a+7b)-3}{4a}$
Вариант 15	$y = \frac{3(a-b)+4}{a+b}$	Вариант 16	$y = \frac{a+1}{5(b+3)}$
Вариант 17	$y = \frac{ab+6}{a-5}$	Вариант 18	$y = \frac{2a-3b}{b-4}$
Вариант 19	$y = \frac{a-5b}{b-3a}$	Вариант 20	$y = \frac{a+4b-2}{a-2b}$
Вариант 21	$y = \frac{3b+2a-4}{5-a}$	Вариант 22	$y = \frac{ab+6}{2-a-b}$
Вариант 23	$y = \frac{a-5b+5}{a+7}$	Вариант 24	$y = \frac{5(a-3b)}{6b-3}$
Вариант 25	$y = \frac{4(a-2b)-5}{b-2a}$	Вариант 26	$y = \frac{7(a-6)+b}{b-5a}$
Вариант 27	$y = \frac{4(a+2b)-6}{b-3a+1}$	Вариант 28	$y = \frac{7(b+a)-1}{4a+6}$
Вариант 29	$y = \frac{2(a+b)-6}{b-1}$	Вариант 30	$y = \frac{2(a+5)-b}{b-2a+1}$

Литература

- 1 Гук, М. Процессоры Pentium III, Athlon и другие / М. Гук, В. Юров. – СПб. : Питер, 2000. – 379 с.
- 2 Зубков, С. В. Ассемблер для DOS, Windows и UNIX / С. В. Зубков. – М. : ДМК Пресс, 2000. – 534 с.
- 3 Программирование на языке ассемблера для персональных ЭВМ : учебное пособие / А. Ф. Каморников [и др.]. – Гомель : ГГУ, 1995. – 95 с.
- 4 Пустоваров, В. И. Ассемблер : программирование и анализ корректности машинных программ / В. И. Пустоваров. – К. : Издательская группа ВНУ, 2000. – 480 с.
- 5 Сван, Т. Освоение Turbo Assembler / Т. Сван. – Киев : Диалектика, 1996. – 540 с.
- 6 Юров, В. Assembler / В. Юров. – СПб. : Питер, 2001. – 624 с.
- 7 Юров, В. Assembler : практикум / В. Юров. – СПб. : Питер, 2002. – 400 с.
- 8 Юров, В. Assembler : специальный справочник / В. Юров. – СПб. : Питер, 2000. – 496 с.

Производственно-практическое издание

Ружицкая Елена Адольфовна,
Осипова Ольга Геннадьевна,
Ковалёва Виктория Александровна

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ASSEMBLER

**Системы счисления,
программная модель микропроцессора,
арифметические команды**

Практическое пособие

Редактор *В. И. Шкредова*
Корректор *В. В. Калугина*

Подписано в печать 19.01.2016. Формат 60x84 1/16.
Бумага офсетная. Ризография. Усл. печ. л. 2,8.
Уч-изд. л. 3,1. Тираж 25 экз. Заказ 30.

Издатель и полиграфическое исполнение:
учреждение образования
«Гомельский государственный университет
имени Франциска Скорины».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/87 от 18.11.2013.
Специальное разрешение (лицензия) № 02330 / 450 от 18.12.2013.
Ул. Советская, 104, 246019, Гомель.