Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования
«Рязанский государственный университет имени С.А. Есенина»

В.Н. ЛОКТЮХИН

АРХИТЕКТУРА КОМПЬЮТЕРА

в 2 книгах

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА АССЕМБЛЕРЕ IBM PC

Книга 2

Учебное пособие

Рекомендовано УМО
по специальностям педагогического образования
в качестве учебного пособия
для студентов высших учебных заведений,
обучающихся по специальности 050202.65 (030100) –
информатика

Рязань 2008

ББК 32.973
УДК 681.3.06
Л73

Печатается по решению редакционно-издательского совета Государственного образовательного учреждения высшего профессионального образования «Рязанский государственный университет имени С.А. Есенина» в соответствии с планом изданий на 2008 год.

Рецензенты: *А.Б. Путилин*, д-р техн. наук, проф. МГОУ,
А.П. Шибанов, д-р техн. наук, проф. РГРТУ

Локтюхин В.Н.

Л 73 Архитектура компьютера: учебное пособие : в 2 кн. – Кн. 2 : Основы программирования на ассемблере IBM PC / Ряз. гос. ун-т им. С.А. Есенина. – Рязань, 2008. – 100 с.

ISBN 978–5–88006–549–3
ISBN 978–5–88006–551–6

В пособии рассмотрены аспекты архитектуры компьютера во взаимосвязи с программированием на языке ассемблера, даны основы разработки и отладки программ с использованием системного программного обеспечения компьютеров IBM PC.

Предназначено для студентов вузов и пользователей, изучающих архитектуру и программирование персонального компьютера.

Ключевые слова: *персональный компьютер, ассемблер, программирование, команда, режим адресации, отладка программ.*

ББК 32.973

ISBN 978–5–88006–549–3
ISBN 978–5–88006–551–6

ВВЕДЕНИЕ

Цель учебного пособия – изложение аспектов *архитектуры компьютера*, которая представлена *функциями его аппаратных средств*, доступных для их программирования. К ним относятся: регистровая модель компьютера, система машинных операций (команд) и основы их выполнения, форматы команд и данных, режимы их адресации. При этом учитывается, что с позиции программиста *программная архитектура компьютера* – это представление его функций, необходимое для программирования на *ассемблере как машинно-ориентированном языке программирования*. Для данного языка характерно описание программы в виде последовательности операторов, где каждому командному оператору соответствует только одна машинная команда процессора конкретной машины, на которую сориентировано выполнение программы на ассемблере.

С учетом этого рассмотрение *операционных функций* компьютера на самом низшем машинном уровне предлагается в данном пособии осуществлять в тесной взаимосвязи с изучением *основ программирования на ассемблере* как эффективного средства, поддерживающего освоение этой части архитектуры через разработку и отладку программ с иллюстрацией их выполнения на уровне команд (операций) конкретного типа компьютера.

Технология обучения на основе этой концепции заключается в следующем. Поэтапно от разработки простых программ (например, линейных с применением простых команд) к разработке более сложных (например, циклических с командами переходов) происходит постепенное освоение понятий архитектуры, тесно связанных с ассемблером. С учетом этой технологии выстроена последовательность представления материалов и определен порядок выполнения практических и лабораторных занятий, темы которых даны в приложении 7 с указанием разделов книги с информацией для их выполнения. При этом развиваются такие важные *общие компетенции*, как основы умений проектировать программные продукты и профессиональной работы на персональном компьютере.

Для их осуществления в пособии даются элементы языка ассемблера, его системное программное обеспечение, приемы создания программ, а также система машинных операций (команд) процессора на примере i80x86, основы их выполнения, форматы команд и данных, режимы их адресации как компоненты и *ассемблера,* и *программной архитектуры компьютера*.

1. ПРЕДСТАВЛЕНИЕ ДАННЫХ И ВЫПОЛНЕНИЕ АРИФМЕТИКО-ЛОГИЧЕСКИХ ОПЕРАЦИЙ В ЭВМ

1.1. Системы счисления, применяемые в ЭВМ

Система счисления (СС) – это совокупность цифр (символов) и правил для обозначения чисел. При программировании используются позиционные СС. В них значение цифры зависит от ее местоположения в записи числа. Дадим сокращенную и развернутую запись десятичного числа:

а) сокращенная запись десятичного числа X = 232,12:

Разряды i:	2	1	0	-1	-2
X =	2	3	2,	1	2 ;

б) развернутая запись десятичного числа X = 232,12:

$X = 2\cdot10^2 + 3\cdot10^1 + 2\cdot10^0 + 1\cdot10^{-1} + 2\cdot10^{-1}$.

Основанием системы счисления q называется число, показывающее, во сколько раз увеличивается значение цифры в числе при смещении ее на одну позицию влево. С учетом этого определения q-ичная запись числа X_q имеет вид:

$$X_q = \alpha_{m-1}\alpha_{m-2}\ldots\alpha_1\alpha_0,\alpha_{-1}\ldots\alpha_{-n} = \sum_{i=-n}^{m-1}\alpha_i q^i,$$

где $\alpha_i \in \{0, 1, 2,\ldots, q-1\}$ – цифры q-й СС. Число цифр α_i равно q.

1. Двоичная система счисления

Основание q = 2. Цифры $\alpha_i \in \{0,1\}$.

Достоинства, которые привели к широкому применению двоичной системы счисления в ЭВМ:

а) простота выполняемых операций сложения и умножения:

0 + 0 = 0; 1 + 0 = 1; 0 + 1 = 1; 1 + 1 = 10;

$0 \times 0 = 0$; $1 \times 0 = 0$; $0 \times 1 = 0$; $1 \times 1 = 1$;

б) простота технической реализации элементов и устройств, используемых для хранения двоичных слов и выполнения над ними арифметических операций;

в) тесная взаимосвязь с алгеброй логики.

Недостаток: длинная, нудная запись двоичного числа.

2. Шестнадцатеричная система счисления

Основание системы q = 16. Цифры $\alpha_i \in$ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}. Символы A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.

Применяется при программировании для сокращения записи двоичного числа путем эквивалентной замены 4-разрядных двоичных чисел – тетрад соответствующими 16-ричными цифрами:

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.

Например, $10100001_2 = A1_{16}$.

1.2. Перевод из десятичной в q-ичную систему счисления

Далее рассматривается перевод целых чисел.

1. Метод деления на основание новой системы счисления

Правило: исходное число X и получающиеся частные последовательно делят на основание новой системы счисления q до получения частного, равного нулю. q-ичная запись числа X образуется как последовательность остатков от деления, начиная с последнего:

$X = 37 = 25_{16}$

37	16	
32	2	16
5	0	0
	2	

2. Метод взвешенного кодирования

Этот метод применяется для перевода числа из десятичной системы счисления в двоичную.

Пример 1.1. Представить X = 37 в двоичной СС.

Веса:	128	64	32	16	8	4	2	1
Число X = 37 =	0	0	1	0	0	1	0	1.

Таким образом, $X = 37 = 100101_2$.

1.3. Перевод из q-ичной в десятичную систему счисления

1. Метод, основанный на замене цифр одной СС на другую с выполнением необходимых арифметических действий.

$$X = 1001{,}101 = 1\cdot2^3 = 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} = 9\frac{5}{8}.$$

$$X = 2A_{16} = 2\cdot16^1 + 10\cdot16^0 = 32 + 10 = 42.$$

2. Табличный метод перевода.

7	6	5	4	3	2	1	0	Разряды
128	64	32	16	8	4	2	1	Веса 2^i
0	0	0	1	1	1	1	1	Число X

$X = 00011111_2 = 31$.

Перевод числа X из произвольной системы счисления q_1 в систему с основанием q_2 осуществляется через десятичную систему (кроме переводов $2 \rightarrow 8$, $2 \rightarrow 16$ и обратно).

1.4. Двоично-кодированная десятичная система счисления

Используется для представления в ЭВМ десятичных чисел, над которыми выполняются арифметические операции в десятичной СС. Для кодирования цифр $0 \div 9$ чаще всего используется BCD-код с весами 8–4–2–1:

BCD-код	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
Цифра	0	1	2	3	4	5	6	7	8	9

Представим десятичное число X = 2459 в виде двоично-кодиро-ваного десятичного числа:

$X = 2459 = 0010\ 0100\ 0101\ 1001_{2\text{-}10}$.

Обратный перевод: $X_{2\text{-}10} = 0010\ 1001\ 0110 = 296$.

Возможные варианты проведения операций перевода, кодирования и выполнения арифметических операций с заданными числовыми данными приведены в приложении 1.

1.5. Кодирование алфавитно-цифровой информации

Осуществляется с помощью 7-битового кода ASCII:

Цифра, символ	ASCII-код
0	0110000 или 30_{16}
1	0110001 или 31_{16}
2	0110010 или 32_{16}
....	….....
9	0111001 или 39_{16}
A	1000001 или 41_{16}
B	1000010 или 42_{16}
C	1000011 или 43_{16}
....	….....

Для контроля на четность (или нечетность) к 7 битам ASCII-кода часто добавляют старший восьмой бит. Поэтому символы передаются отдельными байтами.

1.6. Представление числовых данных в компьютере

Представление чисел в ЭВМ

Существуют две формы записи и соответствующие им две основные формы представления чисел в ЭВМ:

1) естественная, или *с фиксированной точкой* (запятой); например, X = 243,25;

2) полулогарифмическая, или *с плавающей точкой* (запятой); например, $X = 0{,}24325 \cdot 10^{+3}$.

Форматы (формы представления) числовых данных в МП

Формат (структура) числа – это совокупность элементов и связей между ними для наглядного представления числа внутри машины. В общем случае для его представления используется m-разрядное слово:

m–1	m–2	. . .	2	1	0	– разряды слова
X_{m-1}	X_{m-2}	…	X_2	X_1	X_0	

Число без знака $X = X_{m-1}\, X_{m-2} \ldots X_2\, X_1\, X_0$

Разрядность слова совпадает с разрядностью центрального процессора (или с разрядной сеткой) машины. В персональных компьютерах наиболее распространенной формой представления числовой информации являются целые двоичные числа.

1. Представление в ЭВМ целых двоичных чисел без знака

Для представления числа без знака $X = X_{m-1}\, X_{m-2} \ldots X_2\, X_1\, X_0$, где $X_i = \{0, 1\}$, используются все m разрядов слова.

Для 8-разрядной машины $X_{min} = 00000001_2 = 1$; $X_{max} = 11111111_2 = 2^7 + 2^6 + \ldots + 2^1 + 2^0 = 255 = 2^8 - 1 = 256 - 1 = 255$.

Для m-разрядного слова $X_{max} = 2^m - 1$, а диапазон представления чисел $[0; 2^m - 1]$. Максимальная абсолютная погрешность представления чисел составляет $\Delta_{max} = 1$, а приведенная ошибка – $\delta_{max} = 1/(2^m - 1)$. Таким образом, чем больше разрядность m слова (или МП), тем выше точность представления чисел в ЭВМ.

2. Представление целых двоичных чисел со знаком

Эта форма представления чисел наиболее распространенная, так как позволяет просто выполнять арифметические операции с числами со знаком. Старший (m–1) разряд слова используется для представления знака числа $X = \pm\, 0\ X_{m-2}, X_{m-3}, \ldots, X_1, X_0$, а остальные – значащих цифр. Знаки «+»,«–» – числа кодируются следующим образом: «+» в виде «0», а «–» в виде «1».

Для представления (кодирования) чисел со знаком используются специальные *машинные* коды, из которых наиболее распространен *дополнительный код (ДК)*.

Дополнительный код числа $X = \pm\, 0\ X_{m-2}\ X_{m-3} \ldots X_1\ X_0$ равен:

$$[X]_{ДК}: \begin{cases} X, \text{ если } X \geq 0; \\ |\bar{X}| + 1, \text{если } X < 0, \end{cases}$$

где $|\bar{X}|$ – инверсия модуля числа X, при которой $\bar{0} = 1$, $\bar{1} = 0$.

Например, ДК числа X = + 00001010 равен $[X]_{ДК} = 00001010$.

ДК числа X = – 00001010 равен $[X]_{ДК} = 11110101$

$$+ \underline{\qquad\quad 1}$$

$$11110110$$

Для отрицательных двоичных чисел их перевод в ДК можно осуществить по следующему простому правилу. Смотрим на исходное число X справа налево. Первые встретившиеся нули и встретившуюся после них первую единицу оставляем без изменения. Остальные цифры инвертируем.

Правила образования ДК применяются для любой СС. Например, в *формате слова* для q = 16:

$X = -0018_{16} \rightarrow [X]_{ДК} = |\bar{X}| + 1 = FFE7 + 1 = FFE8_{16}$.

Инверсия (not) 16-ричной цифры α равна: not $\alpha = F - \alpha$.

Инверсии 16-ричных цифр:

$\bar{0} = F,\ \bar{1} = E,\ \bar{2} = D,\ \bar{3} = C,\ \bar{4} = B,\ \bar{5} = A,\ \bar{6} = 9,\ \bar{7} = 8,$

$\bar{8} = 7,\ \bar{9} = 6,\ \bar{A} = 5,\ \bar{B} = 4,\ \bar{C} = 3,\ \bar{D} = 2,\ \bar{E} = 1,\ \bar{F} = 0.$

В общем случае инверсия цифры α_q равна: not $\alpha_q = (q - 1) - \alpha_q$.

Обратный переход $[X]_{ДК} \rightarrow X$ показан ниже:

$$X = \begin{cases} + [X]_{ДК}, \text{ если в знаковом бите 0 или 0÷7 для 16-ричной СС;} \\ - (\overline{[X]_{ДК}} + 1), \text{ если в знаковом бите 1 или 8÷F для 16-ричной СС.} \end{cases}$$

Пример 1.2. Найти эквивалент числа X от его дополнительного кода $[X]_{ДК}$:

$[X]_{ДК} = 11110100 \rightarrow X = - (\overline{[X]_{ДК}} + 1) = - (00001011 + 1) = - 00001100.$

$[X]_{ДК} = 00001100 \rightarrow X = + 00001100.$

3. Представление многобайтовых двоичных чисел

Применение только одного слова для представления числа ограничивает его диапазон и точность, поэтому для задания чисел с повышенной точностью используется несколько байт. Их размещают в памяти в смежных ячейках, причем младший байт многобайтного числа размещают по младшему адресу.

4. Представление чисел с плавающей точкой (ПТ)

Имеет вид: $X = \pm M_x \cdot 2^{\pm П_x}$. Поэтому в ЭВМ должны отдельно представляться мантисса M_x и порядок $П_x$.

1.7. Выполнение арифметико-логических операций в микропроцессорах

1.7.1. Краткие сведения об арифметико-логическом устройстве

На рисунке 1.1 приведена укрупненная структура арифметико-логического устройства (АЛУ).

АЛУ служит в процессоре для выполнения арифметико-логических операций. В его состав входят следующие устройства:

– АЛС (арифметико-логическая схема) – сложная комбинационная схема, которая производит арифметические, логические и другие операции в соответствии с кодом операции (КОП) команды, хранящимся в регистре команд (РК);

– A – регистр (аккумулятор), в котором фиксируется результат, снимаемый с выхода АЛС;

– X (или A) и Y – регистры для хранения исходных операндов.

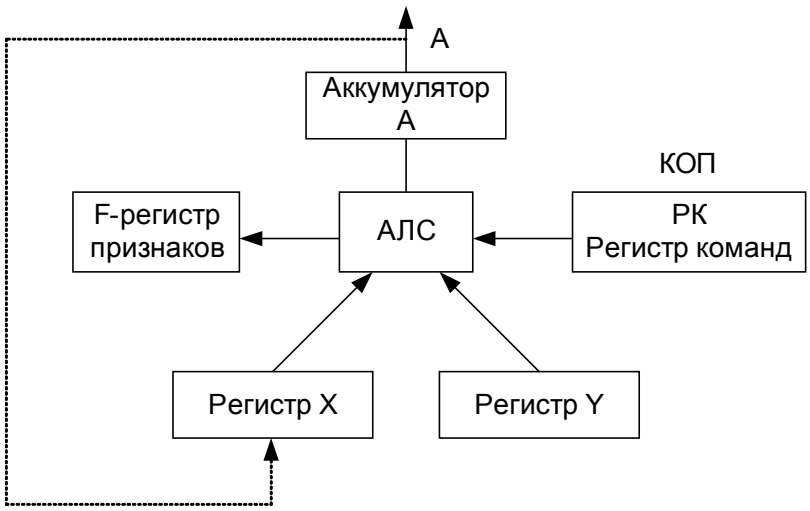


Рис. 1.1. Укрупненная структура АЛУ

АЛУ формирует результат:

$$A \leftarrow X * Y \text{ или } A \leftarrow A * Y,$$

где * – любая операция (сложение (+), вычитание (–), конъюнкция (&), дизъюнкция ($\vee$) и другие).

В регистре F хранятся в виде значений 0 и 1 признаки результата:

Признак нуля	Признак знака «–»	Признак переноса	Признак переполнения	Признак четности
Z	S	C	V	P

Признаки (флажки) результата формируются следующим образом.

$$Z = \begin{cases} 1, \text{ если результат } A = 0; \\ 0, \text{ если } A \neq 0. \end{cases}$$

$$S = \begin{cases} 1, \text{ если } A < 0 \text{ или } A_7 = 1 \text{ (где } A_7 \text{ – старший бит в A)}; \\ 0, \text{ если } A \geq 0 \text{ или } A_7 = 0. \end{cases}$$

$$C = \begin{cases} 1, \text{ если есть перенос (или заем) из } A_7; \\ 0, \text{ если нет переноса (или заема) из } A_7. \end{cases}$$

В триггере признака С фиксируется также значение выдвинутого из регистра А бита при сдвиге слова:

$$V = \begin{cases} 1, \text{ если есть переполнение;} \\ 0, \text{ если нет переполнения при арифметической операции.} \end{cases}$$

$$P = \begin{cases} 1, \text{ если число единиц в результате A четно;} \\ 0, \text{ если нечетно.} \end{cases}$$

1.7.2. Сложение двоичных чисел в ЭВМ

Сложение целых двоичных чисел без знака

При сложении на основе m-разрядного АЛУ чисел X и Y необходимо, чтобы отсутствовало переполнение, т.е. $(X + Y) < 2^m$. Суммирование осуществляется в МП 80x86 по *команде ADD*.

Пример 1.3 (далее примеры даны для 8-битовых чисел).

1101 Переносы, признак C = 0

X = 00000101

$^{+}$Y = $^{+}$<u>00001101</u>

X + Y = 00010010

Поразрядное суммирование для i-го разряда осуществляется с учетом переноса C_i из предыдущего (более младшего) разряда:

$$C_{i+1}\ S_i = X_i + Y_i + C_i,$$

где C_{i+1} – выходной перенос, S_i – младший бит суммы $X_i + Y_i + C_i$.

Сложение целых двоичных чисел со знаком

Производится в ДК. При этом необходимо, чтобы исходные числа были представлены в ДК. Результат сложения выдается в ДК.

Сложение в ДК производится по правилу: сумма дополнительных кодов слагаемых равна ДК суммы

$$[X]_{ДК} + [Y]_{ДК} = [X + Y]_{ДК}$$

при выполнении трех условий:

1) отсутствует переполнение $|X + Y| < 2^{m-1}$;

2) знаковые разряды слагаемых участвуют в операции суммирования по тем же правилам, что и значащие цифры;

3) единица переноса из знакового разряда суммы отбрасывается.

Пример 1.4

$X = +17 = 00010001 \rightarrow {}_{+}[X]_{ДК} = {}_{+}00010001$

$Y = -9 = -00001001 \rightarrow [Y]_{ДК} = \underline{11110111}$

$[X + Y]_{ДК} = 00001000$

Проверка: $X + Y = +00001000 = +2^3 = +8$

Пример 1.5

$X = -17 = -00010001 \rightarrow [X]_{ДК} = {}_{+}11101111$

$Y = +9 = +00001001 \rightarrow [Y]_{ДК} = \underline{00001001}$

$[X + Y]_{ДК} = 11111000$

Для проверки переведем $[X + Y]_{ДК}$ в $X + Y$. Получим: $X + Y = -00001000 = -8$.

При сложении X и Y (пример 2.5) формируются также признаки результата $[X + Y]_{ДК}$, а именно Z = 0, S = 1, C = 0, V = 0, P = 0.

Пример 1.6. Сложить числа, представленные в 16-ричной СС.

$X = -11_{16} = \rightarrow {}_{+}[X]_{ДК} = {}_{+}EF$

$Y = +09_{16} = \rightarrow [Y]_{ДК} = \underline{09}$

$[X + Y]_{ДК} = F8 \rightarrow$ Проверка: $X + Y = -(F8 + 1) = -(07 + 1) = -08$.

Таблица сложения 16-ричных цифр (символов) приведена в приложении 6.

1.7.3. Вычитание двоичных чисел в ЭВМ

Вычитание целых двоичных чисел без знака

Пример 1.7

$^{-}1$ Заемы

$X = 17 = {}_{-}00010001$

$Y = 09 = \underline{00001001}$

$X - Y = 00001000 \rightarrow (8)$

Правило вычитания двоичных цифр:

$0 - 0 = 0, \quad 1 - 1 = 0,$

$1 - 0 = 1, \quad 0 - 1 = 11.$

Заем

Операция вычитания над цифрами X_i и Y_i чисел X и Y производится с учетом входного заема C_i по формуле:

$$C_{i+1}\ r_i = X_i - Y_i - C_i,$$

где C_{i+1} – выходной, r_i – младший бит получаемой разности $X_i - Y_i - C_i$.

Вычитание целых двоичных чисел со знаком

Вычитание производится в дополнительном коде, поэтому исходные операнды заранее представляются в этом коде. Результат также получается в ДК.

В микропроцессорах 80x86 вычитание осуществляется по *команде вычитания SUB* в соответствии с формулой

$$[X]_{ДК} - [Y]_{ДК} = [X - Y]_{ДК}$$

при выполнении трех условий:

1) $|X - Y| < 2^{m-1}$;

2) знаковые разряды операндов участвуют в операции по тем же правилам, что и значащие;

3) единица заема С из результата отбрасывается.

Пример 1.8

$$\begin{array}{rl} & {}^{-}1111 \quad \text{Заемы, С=1} \\ X = 13 = 00001101 \rightarrow [X]_{ДК} = & .00001101 \\ Y = -7 = -00000111 \rightarrow [Y]_{ДК} = & \underline{11111001} \\ [X - Y]_{ДК} = & 00010100 \quad \text{Проверка} \rightarrow X - Y = +20. \end{array}$$

1.7.4. Выполнение в ЭВМ логических операций

Логические операции конъюнкции, дизъюнкции и другие выполняются в ЭВМ над логическими переменными:

$X = X_{m-1}X_{m-2}\ldots X_1X_0$ – логическая переменная X,

$Y = Y_{m-1}Y_{m-2}\ldots Y_1Y_0$ – логическая переменная Y,

где X_i, $Y_i = \{0,1\}$.

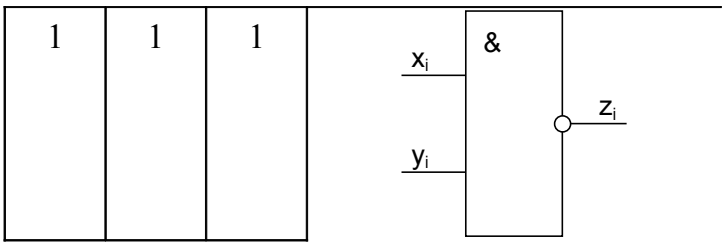
Отметим, что логическая переменная, как машинный элемент информации, представляется в виде двоичного слова.

1. Конъюнкция $Z \leftarrow X\&Y$

Реализуется с помощью *команды AND* путем выполнения поразрядных операций конъюнкции $Z_i = X_i \& Y_i$ над соответствующими битами X_i и Y_i с получением бита результата Z_i в соответствии с таблицей:

X_i	Y_i	Z_i
0	0	0
0	1	0
1	0	0

Логическая схема «И»



Поразрядная операция $Z_i = X_i \& Y_i$ осуществляется в АЛС с помощью логической схемы «И».

Пример 1.9

X = 00001111
Y = $^{\&}$<u>01101011</u>
Z = 00001011

Переносы между разрядами результата отсутствуют. Вследствие этого признак C = 0, остальные признаки формируются. Часто операцию конъюнкции называют *операцией маскирования*. В этом случае одну из переменных называют *маской*. С помощью операции маскирования можно:

1) выделить один бит или группу битов слова,
2) определить модуль числа $|X|$,
3) выявить знак числа.

Пример 1.10

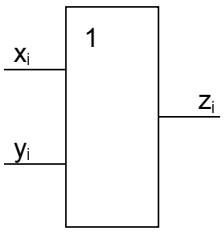
$X=X_7X_6X_5X_4X_3X_2X_1X_0$
Y= <u>0 1 1 1 1 1 1 1</u> – маска для определения $|X|$
$Z = X \& Y = 0\ X_6X_5X_4X_3X_2X_1X_0$

Частным случаем конъюнкции является *операция тестирования X &Y*. При ее выполнении устанавливаются только признаки результата X &Y без формирования его значения. Эта операция выполняется микропроцессором с помощью *команды TEST*.

2. Дизъюнкция Z ← X V Y

X_i	Y_i	Z_i
0	0	0
0	1	1
1	0	1
1	1	1

Логическая схема «ИЛИ»



Операция выполняется по *команде OR* и часто используется для образования из частей слова нового слова.

Пример 1.11

X = 00001001

Y = ^{V}<u>10110000</u>

Z = 10111001

Поразрядная операция $Z_i = X_i \vee Y_i$ осуществляется в АЛС с помощью логической схемы «ИЛИ».

3. Сложение по модулю два (неравнозначное ИЛИ) $Z \leftarrow X \oplus Y$

X_i	Y_i	Z_i
0	0	0
0	1	1
1	0	1
1	1	0

Пример. 1.12

X = 10001110

Y = $^{\oplus}$<u>10001110</u>

Z = 00000000

Операция часто используется для сброса в 0 регистра, определения знака произведения или частного путем сложения по модулю два знаков операндов. Знак Z = Знак X ⊕ Знак Y.

4. Инверсия: $X \leftarrow \overline{X}$

Пример 1.13

X = 00000110, $\overline{X}$ = 11111001.

5. Сравнение X – Y

При сравнении формируются только признаки результата вычитания X – Y, а сама разность не фиксируется. Данная операция осуществляется путем выполнения *команды CMP*.

6. Сдвиги слов

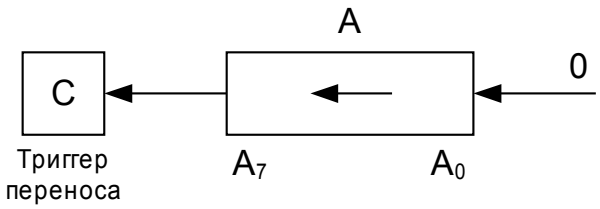
Различают следующие типы сдвигов: логические, арифметические и циклические.

Логические сдвиги

При логическом сдвиге слова влево содержимое всех его бит, например слова $A = A_7A_6 \ldots A_0$ перемещается влево на один разряд.

Освобождающийся разряд A_0 заполняется нулем. Выдвинутый бит A_7 чаще всего размещается в триггере переноса С.

Логический сдвиг влево

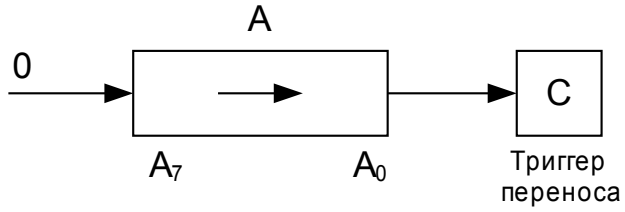


0	0 0 0 0 0 0 1 1	до сдвига A = 3
0	0 0 0 0 0 1 1 0	после сдвига A = 6

Сдвиг числа влево на один разряд соответствует умножению значения А на два: $A \leftarrow 2 \cdot A$.

Сдвиг числа вправо на один бит равносилен операции деления: $A \leftarrow A/2$.

Логический сдвиг вправо



0 0 0 0 1 1 0 0	до сдвига A = 12
0 0 0 0 0 1 1 0	после сдвига A = 6

Арифметические сдвиги

Арифметический сдвиг влево чисел в ДК полностью совпадает с логическим сдвигом влево.

При *арифметическом сдвиге вправо* содержимое старшего разряда не изменяется. Например, для чисел в ДК:

до сдвига $[X]_{ДК} = 11111100$ (–4);
после сдвига $[X]_{ДК} = 11111110$ (–2).

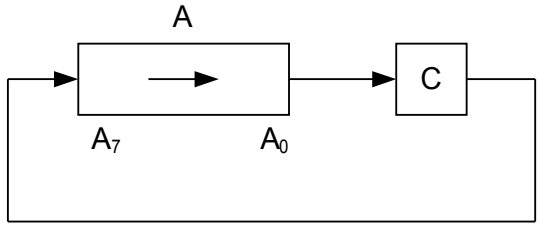
При арифметическом сдвиге выдвинутый бит размещается в триггере переноса С.

Циклические сдвиги через триггер переноса

При данном виде сдвига слова выдвинутый бит размещают в триггере переноса С, а в освобождающийся разряд слова А переносят бит из триггера переноса.

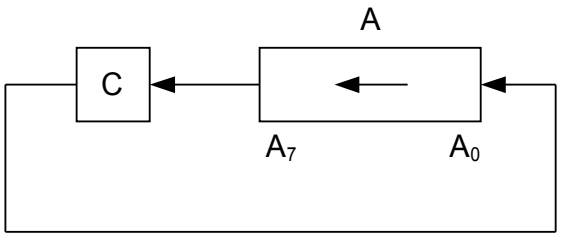
Сдвиг вправо через С

$A_i \leftarrow A_{i+1}$, $C \leftarrow A_0$, $A_7 \leftarrow C$



Сдвиг влево через С

$A_i \leftarrow A_{i-1}$, $A_0 \leftarrow C$, $C \leftarrow A_7$



Циклические сдвиги минуя триггер переноса

При данном виде сдвига слова А выдвинутый бит размещают в триггере переноса С, а в освобождающийся разряд слова А помещают выдвинутый бит из регистра А.

Если в системе команд отсутствует операция логического или арифметического сдвига, то любой сдвиг можно осуществить с помощью циклических сдвигов. Эти сдвиги также используются для организации сдвига слов повышенной длины.

2. ПРИНЦИПЫ ДОСТУПА К ПАМЯТИ. ФОРМАТЫ КОМАНД И РЕЖИМЫ АДРЕСАЦИИ В МП 80X86

2.1. Регистры и модели доступа к памяти. Поддержка многозадачности в защищенном режиме работы МП

2.1.1. Регистровая модель процессора 80x86

При разработке программ на языке ассемблера целесообразно представлять процессор в виде его *регистровой модели* как одной из важных составляющих его программной архитектуры. Эта модель содержит набор *программно доступных регистров процессора* (или МП), содержимое которых пользователь может изменять программно с помощью команд МП. Добавляя к регистрам МП адресуемое множество ячеек (регистров) памяти и портов ввода-вывода (в/в), строится регистровая модель компьютера в целом.

Подобный способ наглядного представления регистров, участвующих в выполняемых МП операциях, является необходимой и важной составляющей процесса разработки прикладных программ на ассемблере. Упрощенно это сводится к тому, что в записях его командных операторов в качестве операндов могут использоваться мнемоники регистров МП и принятые обозначения операндов, размещенных в основной памяти и в адресно-доступных регистрах (портах в/в) внешних устройств. Их подмножества образуют соответственно *карту памяти и карту портов ввода-вывода* ЭВМ, называемую в ПК также интерфейсной картой.

На рисунке 2.1 приведена регистровая (программная) модель МП Intel 8086, регистры которого входят в состав любого 32-битового МП семейства 80x86 при его работе в реальном (R-) режиме.

В приведенной регистровой модели в зависимости от выполняемых функций выделяют следующие группы регистров.

1. Регистры данных. В зависимости от того, чем оперирует команда – словами или байтами – регистры данных можно рассматривать как четыре 16-битовых (AX, CX, BX, DX) или как восемь 8-битовых регистров (AL, AH, CL, CH, BL, BH, DL, DH). Символы L и H означают младшие и старшие байты 16-разрядных регистров.

Каждый из этих регистров МП помимо общих выполняет специализированные функции: AX, AL – аккумулятор; BX – базовый регистр для косвенной адресации памяти; CX – счетчик в операциях с цепочками; DX – указатель при косвенной адресации портов ввода/вывода.

Рис. 2.1. Регистровая модель МП i8086 как базовая для МП 80x86

2. Регистры сегментов CS, DS, SS, ES служат для *поддержки доступа* выполняемой программы к четырем 64-килобайтовым сегментам памяти: кода (команд), данных, стека и дополнительного сегмента данных.

3. Регистры-указатели SP, BP, BX, DI и SI предназначены для хранения внутрисегментных смещений и обеспечивают косвенную адресацию данных в пределах текущего сегмента памяти. Эти регистры также называют *регистрами общего назначения (РОН)*, так как они могут также участвовать в выполнении арифметических и логических операций над двухбайтовыми данными, то есть, используются при этом как регистры данных.

В этой группе указатели стека SP и базы BP предназначены для доступа к данным в текущем сегменте стека (сокращенно SS).

Регистры-указатели DI, SI, а также BX содержат смещение, которое по умолчанию относится к сегменту данных (DS). В операциях с цепочками данных указатель DI по умолчанию относится к дополнительному сегменту данных (ES).

4. Указатель команд IP адресует следующую команду программы (разумеется, с учетом очереди команд) в сегменте кодов, или программы (CS).

5. Регистр признаков F (или *FLAG*) служит для хранения признаков результата выполняемой процессором команды. Формат 16-битового регистра признаков F показан на рисунке 2.2.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Признаки:
OF – переполнение,
DF, IF, TF – признаки управления МП,
SF – знак, ZF – нуль,
AF – вспомогательный перенос, PF – четность,
CF – перенос, X – признак отсутствует.

Рис. 2.2. Формат регистра признаков

Содержимое *пользовательской регистровой модели* для 32-разрядных МП (МП-32) дополняется для целочисленных операций 32-битовыми регистрами. Они являются расширениями регистров AX, BX, CX и т.д. и имеют в своих обозначениях букву E (Extend). Все 32-разрядные пользовательские регистры, кроме указателя команд EIP и регистра признаков EFLAG, являются РОНами, то есть, могут быть также задействованы как указатели памяти:

– EAX, AX, AH, AL; – ESI, SI;
– EBX, BX, BH, BL; – EDI, DI;
– ECX, CX, CH, CL; – EBP, BP;
– EDX, DX, DH, DL; – ESP, SP;
– указатель команд: EIP, IP;
– регистр признаков: *EFLAG*, FLAG;
– сегментные CS, DS, SS, ES, *FS, GS*;
– *системные регистры:*
• *регистры дескрипторных таблиц GDTR, LDTR и IDTR,*
• *регистр дескриптора сегмента состояния задач TSS.*

Приведенный набор пользовательских регистров процессора также содержит дополнительные регистры, применяемые в защищенном (Р-) режиме работы МП (они выделены курсивом).

Так, регистры *FS и GS* позволяют определить *дополнительные сегменты данных*. *Системные регистры GDTR, LDTR, IDTR* и *TSS* поддерживают управление доступом МП к памяти в Р-режиме.

2.1.2. Модели доступа к памяти в реальном и защищенном режимах работы процессора, поддержка многозадачности

В персональных компьютерах на базе МП 80x86 при организации обращения микропроцессора к его основной памяти применяют специальные *модели доступа к памяти*. Их назначение – сформировать физические (линейные) адреса (ФА) размещения команд и данных в памяти. Процедура их получения зависит от режима работы микропроцессора: R – реального, P – защищенного.

***1. Модель доступа к памяти в R-режиме**, называемая сегментацией памяти*, базируется на представлении 1 Мбайт памяти в виде отдельных сегментов. Для данной модели программа может иметь дело одновременно с четырьмя типами сегментов памяти емкостью по $2^{16} = 64$ Кбайт в каждом. Это следующие сегменты:

- сегмент кодов, на который указывает сегментный регистр CS;
- сегмент данных, определяемый сегментным регистром DS;
- сегмент стека, на который указывает сегментный регистр SS;
- дополнительный сегмент данных, определяемый регистром ES.

В частном случае CS = DS = SS = ES, а в общем CS ≠ DS ≠ SS ≠ ES.

На сегмент кодов, в котором размещаются коды команд выполняемой программы, указывает 16-разрядный сегментный регистр CS. При этом указатель команд IP задает в этом сегменте памяти так называемый 16-разрядный *эффективный адрес EA*. Он показывает в сегменте, на сколько ячеек относительно значения CS (как *базового адреса* сегмента) размещена команда, на которую указывает IP. Запись CS:IP называют *логическим (реальным) адресом*.

На основе значений CS и IP формируется 20-битовый *линейный* или *физический адрес команды* $ФА_{ком}$. Он образуется в МП путем сложения с помощью 20-разрядного сумматора 16-битовых значений IP и $CS \cdot 2^4$, то есть, содержимого CS, сдвинутого влево на четыре двоичных разряда (или на один 16-ричный разряд, так как $2^4 = 16^1$):

$$ФА_{ком} = CS \cdot 2^4 + IP.$$

С помощью этого адреса определяется месторасположение команды в основной памяти, реализуемой в виде набора полупроводниковых микросхем.

Для сегмента данных физический адрес $ФА_{оп}$ операнда равен сумме значений $DS \cdot 2^4$ и эффективного адреса EA операнда:

$$ФА_{оп} = DS \cdot 2^4 + EA.$$

Например, при выполнении команды MOV AX,[0502H], для которой величина EA = 0502H указана в команде, значение $ФА_{оп}$ при DS = 3000H равно: $ФА_{оп}$ = 30000H + 0502H = 30502, где H – признак 16-ричной системы счисления.

Для команды MOV AX,[SI+200H], в которой EA = SI + 0200H = = 0502H при SI = 0302H и DS = 3000H, значение адреса $ФА_{оп}$ равно: $ФА_{оп}$ = 30000H + 0302H + 0200H = 30502H.

Для команд обращения к памяти вид сегмента и его адрес по умолчанию обычно задается сегментным регистром DS. Вместе с тем, для такого рода команд можно переопределить вид сегмента, если в команде указать соответствующее ему имя сегментного регистра. Так, команды MOV AX,ES:[0502H] и MOV AX,ES:[SI+200H] будут уже оперировать с операндами, размещенными в дополнительном сегменте данных, на который указывает сегментный регистр ES.

Рассмотренные принципы формирования физического адреса памяти $ФА_{оп}$ справедливы также для МП-32 при его работе в R-режиме, когда вместо EA находится 32-разрядный эффективный адрес EEA, однако действующий в пределах текущего 64 Кбайтового сегмента (принципы формирования EEA рассмотрены в п. 3.7).

Несмотря на простоту организации, сегментация памяти для реального режима неэффективна для реализации мультизадачных систем, так как в данной модели отсутствуют механизмы контроля и защиты доступа к сегментам. Например, программа может обращаться к любому сегменту для производства операций как записи, так и считывания. Также нет никаких препятствий (защиты) для обращения к физически несуществующей памяти и т.д.

Модель доступа к памяти в защищенном режиме используется только для 32-разрядных микропроцессоров. Наличие данного режима с его поддержкой аппаратными средствами МП-32 позволяет в компьютерах IBM PC обеспечить *реализацию многозадачности*, при которой в режиме разделения времени выполняется несколько задач (приложений) с поддержкой функций их *защиты*.

Переход из одного режима в другой производится с помощью установки командой MOV бита PE регистра управления CR0 процессора. При включении компьютера первоначально устанавливается реальный режим работы микропроцессора (бит PE = 0). Он изменится после загрузки операционной системы (ОС), работающей в защищенном режиме, например, Windows XP.

Управление памятью в P-режиме – один из сложных вопросов системного программирования. Это во многом обосновано тем, что в за-

щищенном режиме процедура формирования физического адреса намного сложнее.

Выделяют две основные модели управления памятью: 1 – для ее сегментной и 2 – для ее страничной организации. Далее приводятся принципы построения модели доступа к памяти для ее сегментной организации.

Упрощенно эти принципы базируется на двухступенчатой схеме получения 32-разрядногофизического адреса $ФА^{р}$ операнда в памяти.

На первой ступени, на основе содержимого специального *регистра дескрипторной таблицы*, размещенного в процессоре, определяется начало и размер *дескрипторной таблицы*, заданной в основной памяти. Эту таблицу для сегментной организации памяти в Р-режиме также называют таблицей описания сегментов памяти, так как в ней кроме *32-разрядных базовых адресов сегментов* содержатся их размеры (до 64 Кбайт) и байт с описанием прав доступа к этим сегментам. Содержимое этих дескрипторов предварительно заполняется с помощью выполнения *привилегированных команд* МП.

Далее, на второй ступени, формируется *32-битовый физический адрес для Р-режима* $ФА^{р}$ в виде суммы двух 32-разрядных составляющих: содержимого *базового адреса сегмента* ($базы^{р}$, или селектора) и *эффективного адреса операнда ЕЕА*:

$$ФА^{р} = база^{р} + ЕЕА.$$

Формирование адреса $ФА^{р}$ ведется с контролем заданного размера сегмента и прав доступа. К правам относятся: уровень привилегий сегмента, его целевое использование с указанием для сегментов данных, стека и кодов разрешения на операции чтения, записи и т.п. Попытки нарушить эти правила вызывают особые события, которые через *систему прерываний* фиксируются в вычислительной машине.

Для эффективной *реализации многозадачности в защищенном режиме* работы МП 80х86 применяется три типа дескрипторных таблиц, размещенных в памяти: *таблица глобальных дескрипторов (GDT), таблица дескрипторов обработчиков прерываний (IDT) и таблица локальных дескрипторов (LDT)*.

Таблица GDT содержит информацию о программных сегментах и сегментах данных, используемую ОС, а также информацию о состоянии всех решаемых задач (отсюда название «глобальная»).

В таблицу локальных дескрипторов LDT входят дескрипторы, относящиеся к отдельным (локальным) задачам.

Таблица дескрипторов прерываний (IDT) содержит указатели на программы обработки прерываний и вентили задач, поддерживающие уровень (приоритеты) задач.

Для обращения к этим таблицам процессор содержит следующие *регистры управления памятью в Р-режиме*:

– два 6-байтовых *регистра дескрипторных таблиц GDTR и IDTR*, содержащие 32-разрядные адреса начала (селекторы) таблиц GDT и IDT и их 16-битовые размеры;

– один 10-байтовый *регистр локальной дескрипторной таблицы LDTR*, содержащий 16-битовый селектор для GDT и весь 8-байтовый дескриптор из GDT, описывающий текущую таблицу LDT.

Для организации в Р-режиме контекстной памяти, необходимой для хранения информации о состоянии всех регистров ЦП и некоторых переменных памяти для каждой из выполняемых задач, также применяется *сегмент состояния задачи TSS*.

Для обращения к этому сегменту процессор содержит *регистр сегмента состояния задачи TR*. Это 10-байтовый регистр, содержащий 16-битовый селектор для GDT и весь 8-байтовый дескриптор из GDT, описывающий TSS текущей задачи.

Для сокращения времени обращения к дескрипторной таблице для получения базового адреса сегмента, как составной части 32-разрядного физического адреса, равного $ФА^{Р}$ = база + ЕЕА, значение базы, а также размер сегмента хранятся для каждого сегментного регистра в виде их копий в «теневых регистрах» процессора.

2.2. Классификация команд процессора

Команда ЭВМ – это слово, или упорядоченная последовательность бит, содержащая:

а) *операционную часть*, в которой с помощью кода операции КОП задается операция, выполняемая командой;

б) *адресную часть*, чаще всего состоящую из двух полей A_1 и A_2, с помощью которых прямо или косвенно указываются адреса исходного операнда (источника s) и результата (приемника d):

КОП	A1	A2
Операционная часть	*Адресная часть*	

Все многообразие команд процессоров семейства Intel 80x86, (см. приложение 8 для базовых команд) можно условно разделить по следующим признакам (рис. 2.3).

Рис. 2.3. Классификация команд процессоров семейства Intel 80x86

1. По типу выполняемой операции

1.1. Команды пересылки (передачи) данных. Например, посредством команды MOV AX,BX содержимое регистра BX пересылается в AX (сокращенно AX ← BX).

С помощью одной команды реализуется пересылка типа «регистр – регистр» (R ← R) или «регистр – память» (R ← M, M ← R).

1.2. Команды арифметических и логических операций (АЛО). В соответствии с этими командами МП реализует операции суммирования, вычитания, инкремента, декремента чисел, операции конъюнкции и дизъюнкции, сдвига слов и другие, например операцию сложения AX ← AX + BX, выполняемую командой ADD AX,BX.

1.3. Команды передачи управления JMP, CALL, JNS, JZ и другие. изменяют, например, содержимое указателя команд IP, загружая в него адрес перехода.

1.4. Специальные команды управления МП: HLT (останов), NOP (пустая операция) и другое.

2. По числу адресов операндов

2.1. Безадресные команды (или с неявной адресацией) – не содержат полей для указания адреса операнда, например DAA – команда десятичной коррекции аккумулятора AL.

2.2. Одноадресные команды содержат поле для указания месторасположения только одного операнда. Другой операнд может быть определен неявно, например, аккумулятор AX в команде передачи MOV AX, [0502H].

2.3. Двухадресные команды имеют два адресных поля: одно для операнда-источника, а второе для операнда (результата)-приемника. Например, MOV BX,[0504H].

3. По числу байт команды различают *1-, 2-, 3-байтовые команды и т.д*. Команды МП 80x86 могут содержать 16 байт.

4. По виду режима адресации выделяют две группы команд: 1 – *с прямой,* 2 – *с косвенной адресацией данных*.

2.3. Способы (режимы) адресации данных

Формат команды и способ адресации как элементы программной архитектуры и ассемблера

Независимо от вида применяемой модели доступа к памяти, для формирования физического адреса ФА операнда (см. п. 2.1.2) необходимо также найти его вторую составляющую: *эффективный 16-битовый EA (*или *32-битовый EEA) адрес операнда* в текущем сегменте памяти (далее – памяти).

Правило определения адреса EA операнда в памяти (как развитие модели доступа) называется *способом (*или *режимом) адресации*. Каждый из принятых в МП режимов адресации задается с помощью отдельного короткого поля (или двух полей) в двоичном коде команды, наглядно представляемой в виде ее *формата* или ее *мнемокода*, например, MOV AX,[BX].

В итоге определенному формату команды (приложение 8) соответствует только определенный для нее порядок выполнения операций над ее адресным полем и содержимым одного или нескольких регистров процессора с целью вычисления адреса EA, по которому хранится операнд в памяти (или блоке РОН микропроцессора).

Для вычисления эффективного адреса EA в МП 80x86 используются: прямые и косвенные методы (режимы) адресации. Знание режимов формирования адресов, заложенных в том или ином формате команды МП как *элементе его программной архитектуры*, позволяет создавать эффективные программы, в частности, по минимуму их времени выполнения и объема в байтах. Эти особенности учитываются при программировании на ассемблере, так как его командные операторы – это команды (операции) микропроцессора.

2.3.1. Прямые методы адресации, понятие формата команды

Для этих методов исполнительный адрес $A_{ИСП}$ (или EA) операнда (или операнд-константа) указывается прямо (или непосредственно) в команде. Характерны следующие режимы прямой адресации данных.

1. Регистровая адресация. Для данного режима в адресной части команды имеется обычно 3-битовое поле reg, содержащее адрес (номер) регистра микропроцессора МП, в котором размещен операнд. Пример – команда передачи «регистр – регистр» MOV AX,BX, выполняющая операцию AX ← BX (рис. 2.4).

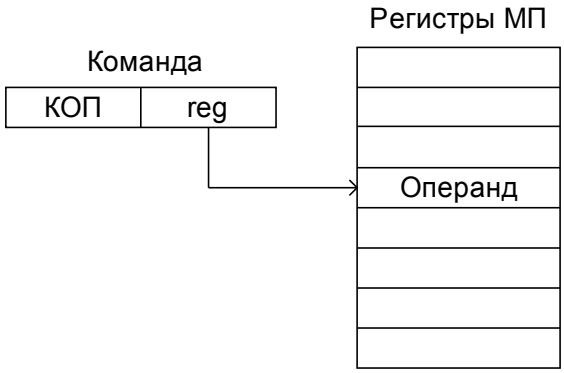


Рис. 2.4. Формат команды с регистровой адресацией

Для описания операционных возможностей команды (режим адресации, выполняемая ей операция, число операндов этой операции и другое) часто применяют ее наглядное представление – *формат команды*, например, в виде, содержащем поля КОП и reg (см. рис. 2.4).

2. Прямая (или абсолютная) адресация. В адресной части команды указывается адрес $A_{ИСП}$ (или EA) операнда, размещенного в памяти. Пример: команда пересылки MOV AX,[0500H] в регистр AX слова с адресом EA = 0500H из сегмента данных оперативной памяти M; сокращенно AX ← [0500H], или AX ← M_{0500} (рис. 2.5).

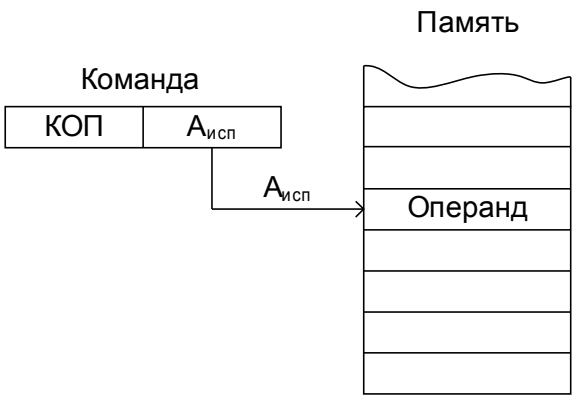


Рис. 2.5. Формат команды с абсолютной (прямой) адресацией

3. Непосредственная адресация. В команде вместо адреса непосредственно указывается операнд-константа. Пример: команда MOV CX,120H загрузки в регистр CX константы 120_{16}; CX← 0120H.

Достоинства команд с прямой адресацией:

– простота механизма их действия;

– команды с регистровой адресацией самые быстрые, так как работают с операндами, размещенными в регистрах МП, которые являются наиболее быстродействующими компонентами компьютера.

Их недостатки:

– команды с абсолютной адресацией характеризуются значительной длиной (по числу байт) и выполняются медленнее по сравнению с командами, использующими косвенную регистровую адресацию данных;

– сложность программирования обработки массивов данных.

2.3.2. Косвенные методы адресации

При применении этих методов для вычисления EA, наряду с адресной частью команды, также используется информация, содержащаяся в регистрах МП, которые называют *указателями памяти*. В МП 80x86 – это регистры BX, SI, DI для обращения к данным, размещенным в сегменте данных (DS), и регистр BP, оперирующий с сегментом стека (SS).

В микропроцессорах 80x86 применяются следующие стандартные режимы косвенной адресации:

1. Косвенная регистровая адресация. В команде (рис. 2.6) содержится короткое 3-битовое поле r/m, которое совместно с дополнительным 2-битовым полем mod = 00 определяет в МП регистр BX, SI или DI, хранящий эффективный адрес EA операнда в памяти, то есть:

$$EA=\{BX, SI, DI\}.$$

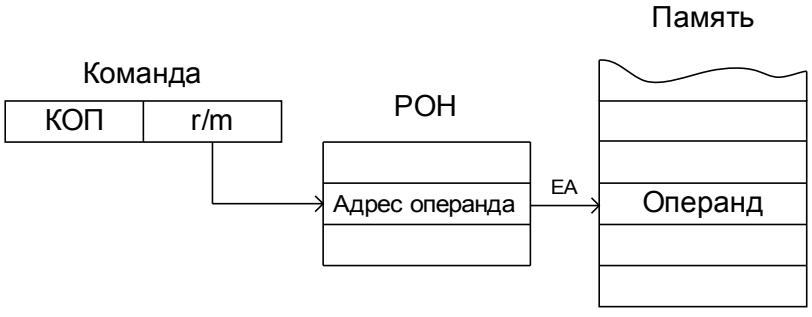


Рис. 2.6. Формат команды с косвенной регистровой адресацией

Пример: команда пересылки MOV AX,[BX] в регистр AX слова из ячейки памяти, на которую указывает регистр BX, сокращенно AX ← [BX], или AX ← M_{BX}. Для указания адреса ячейки, например 0500H, необходимо предварительно его значение загрузить в BX, в частности с помощью команды MOV BX,0500H.

2. Базовая адресация. При данной адресации, наряду с полем r/m, в команде (рис. 2.7) также указывают базу, или смещение (один байт d*isp L* для mod = 01 или два байта *disp HL* для mod = 10). При этом эффективный адрес EA формируется в виде суммы содержимого регистра, на который указывает поле r/m, и значения базы:

$$EA: \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} disp\,L \\ disp\,HL \end{Bmatrix}.$$

Пример: команда пересылки «регистр – память» с базовой адресацией MOV AX,[BX+20]; AX←[BX+20].

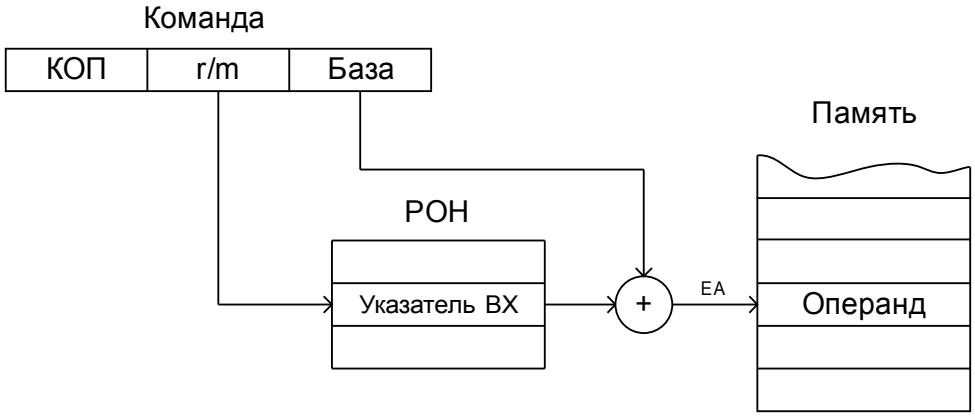


Рис. 2.7. Формат команды с базовой адресацией

3. Индексная адресация. Схема получения адреса операнда схожа со схемой формирования EA для базовой адресации (см. рис. 2.7), только в качестве указателя памяти используется регистр SI или DI, а вместо базы в команде задается значение индекса:

$$EA: \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} disp\,L \\ disp\,HL \end{Bmatrix}.$$

Пример: команда MOV [SI-2],AX ; [SI–2] ← AX.

Отрицательное значение смещения (базы или индекса) задается в команде в дополнительном коде.

4. Базовая индексная адресация. Для данного режима адрес EA формируется в виде суммы, содержащей три составляющие:

$$EA: \begin{vmatrix} BX \\ BP \end{vmatrix} + \begin{vmatrix} SI \\ DI \end{vmatrix} + \begin{vmatrix} disp\,L \\ disp\,HL \end{vmatrix}.$$

Пример: команда MOV AX,[BX+SI+2] ; AX ← [BX + SI + 2].

Пример 2.1

Программа сложения d ← d + s двухбайтовых чисел s и d, размещенных в памяти с адреса 500H, для различных вариантов адресации данных

Вариант 1. Адресация: абсолютная для **s,** *базовая для* **d**.

На рисунке 2.8 показана программная (регистровая) модель выполнения операции сложения **d ← d + s** для данного варианта.

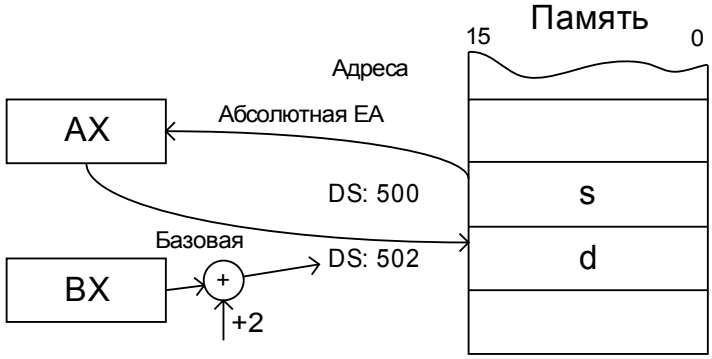


Рис. 2.8. Регистровая модель операции d ← d + s для варианта 1

Для нее характерно использование аккумулятора AX для хранения промежуточных результатов. В соответствии с базовой адресацией в регистре BX хранится константа 500H, которая совместно со значением базы, равным +2, образует адрес EA = 502H операнда **d**, а затем результата в сегменте данных оперативной памяти.

Программа выполнения операции **d ← d + s** содержит следующую последовательность команд (или *фрагмент программы*):

```
MOV  AX,[0500H]    ;AX ← s (абсолютная адресация s)
MOV  BX,0500H      ;BX ← 0500H
ADD  [BX+2],AX     ;d ← d + s (базовая адресация d)
```

Вариант 2. Адресация: индексная для ***s,*** *косвенная регистровая для* ***d***.

Как показано на рисунке 2.9, регистры SI и BX используются для формирования адресов операндов **s** и **d** в памяти для индексной и косвенной регистровой адресации соответственно.

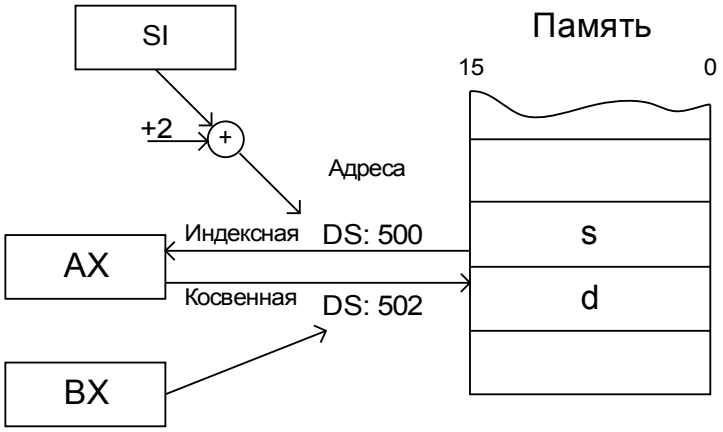


Рис. 2.9. Регистровая модель операции d ← d + s для варианта 2

Программа получения суммы **d ← d + s** для данного варианта содержит следующую последовательность команд:

```
MOV  SI,04FEH    ;SI ← 04FEH
MOV  AX,[SI+2]   ;AX ← s (индексная адресация s)
MOV  BX,0502H    ;BX ← 0502H
ADD  [BX],AX     ;d ← d + s (косв. рег. адресация d)
```

Можно сократить число команд в программе для 2-го варианта, если при формировании адресов **s** и **d** использовать только один регистр SI. Тогда программа будет содержать всего 3 команды:

```
MOV  SI,0502H    ;SI←0502H
MOV  AX,[SI-2]   ;AX← s (индексная адресация s)
ADD  [SI],AX     ;d ← d + s (косв. рег. адресация d)
```

2.3.3. Дополнительные режимы адресации в 32-разрядных МП

Рассмотренные ранее стандартные режимы адресации применяются и в 32-разрядных микропроцессорах (МП-32). Однако в МП-32 все пользовательские 32-битовые регистры являются РОНами и поэтому

каждый из них может выступать в качестве указателя памяти для формирования 32-разрядного адреса операнда EEA.

Наряду с этим, в МП-32 имеются дополнительные режимы адресации, в частности *масштабированная индексная адресация* (см. далее п. 3.6.1).

2.4. Примеры кодирования команд в МП 80x86

Примеры форматов команд с регистровой (1, 5, 6), непосредственной (2, 7, 8) и прямой (3, 4) адресацией приведены на рисунке 2.10.

1. Передача «регистр – регистр» MOV reg1,reg2 ;reg1 ← reg2

1-й байт	2-й байт	3-й байт
7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	
1 0 0 0 1 0 1 w	1 1 reg1 reg2	

2. Передача в регистр константы MOV reg,data ; reg ← data

1 0 1 1 w reg	data L	data H	,если w = 1

3. Передача в аккумулятор из памяти MOV acc,[EA] ; acc ← [EA]

1 0 1 0 0 0 0 w	Мл часть EA	Ст часть EA

4. Передача в память из аккумулятора MOV [EA],acc ; [EA] ← acc

1 0 1 0 0 0 1 w	Мл. часть EA	Ст. часть EA

5. Сложить «регистр – регистр» ADD reg1,reg2 ; reg1← reg1 + reg2

0 0 0 0 0 0 1 w	1 1 reg1 reg2

6. Вычесть «регистр – регистр» SUB reg1,reg2 ; reg1← reg1 – reg2

0 0 1 0 1 0 1 w	1 1 reg1 reg2

7. Сложить с аккумулятором константу ADD acc,data ;acc ← acc+data

0 0 0 0 0 1 0 w	data L	data H	,если w = 1

8. Вычесть из аккумулятора константу SUB acc,data ; acc ← acc – data

0 0 1 0 1 1 0 w	data L	data H	,если w = 1

Рис. 2.10. Примеры форматов команд

В *обобщенном представлении (формате) команд*, например ADD reg1,reg2, после мнемоники команды указывается приемник, а затем через запятую – источник операнда. В байтах data L и data H команды указываются младшая и старшая (если W = 1) части операнда-константы data. Обозначение «acc» в команде означает аккумулятор AX или AL.

Во всех командах содержится однобитовое поле W: если W = 1, то команда оперирует словом, если W = 0 – байтом.

В поле reg (или reg1, reg2, r/m) команд указывается трехбитовый адрес РОНа МП. В таблице 2.1 приведены адреса регистров МП для различных значений W.

Таблица 2.1

Адреса регистров МП

Адрес регистра reg, r/m	Регистр	
	W = 1	W = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Рассмотрим на основе обобщенных форматов команд (см. рис. 2.10) принципы кодирования отдельных команд и связанное с этим получение их *мнемонических обозначений (мнемоник)* на примере разработки программы вычисления заданной далее операции (2.1).

Знание этих принципов является важной базой для изучения основ формирования *мнемоник команд* и их действий с конкретными регистрами на основе форматов команд МП, приведенных в приложение 8, при разработке программы на языке ассемблера.

2.5. Порядок разработки программы с использованием мнемоник команд микропроцессора

***Задание**:* разработать, используя только набор команд, данный на рисунке 2.10, программу для вычисления заданного выражения

$$M = K + N - R + 120_{16}, \qquad (2.1)$$

в котором все операнды и результат – двухбайтовые данные (слова).

Пусть K, R, M размещены в сегменте данных, а N – в регистре DX. Числа со знаком представлены в дополнительном коде (ДК). Например, ДК 16-ричных чисел $K = +60_{16}$, $R = -10_{16}$, $N = +30_{16}$ соответственно равны $[K]_{ДК} = 0060$, $[R]_{ДК} = FFF0$, $[N]_{ДК} = 0030$.

Возможные варианты заданий даны в приложении 2.

Порядок разработки программы

1. *Выражение (2.1) переписывается* в виде, показывающем процесс его вычисления через минимально возможную последовательность операций (вычитания, сложения, передачи и другие), выполняемых командами, приведенными на рисунке 2.10:

$$M = (\underset{\underset{3}{\underset{2}{\underset{1}{\_\_\_\_\_}}}}{(N - R)} + K) + 120_{16}. \qquad (2.2)$$

В ряде случаев целесообразно эту последовательность, например 1-2, начать формировать с операции над операндом, размещенным в регистре, с сохранением получаемого при этом результата в том же регистре.

2. *Формируется регистровая (программная) модель* (рис. 2.11) вычисления выражения 2.2.

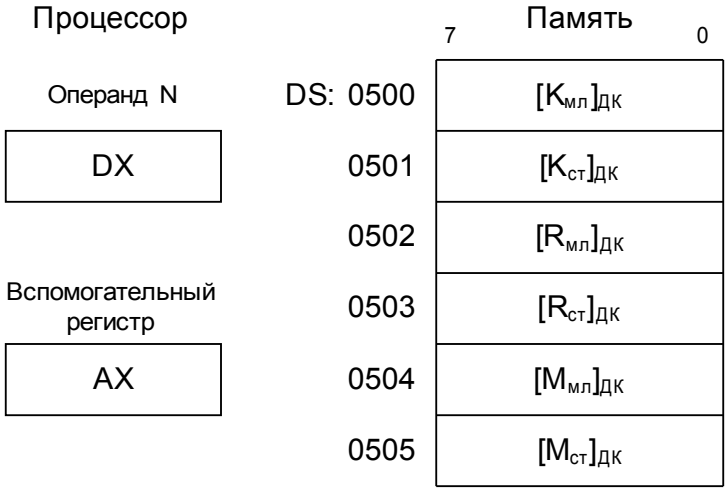


Рис. 2.11. Регистровая модель выполняемой операции (2.2)

В этой модели указываются ячейки основной памяти, в которых размещаются в ДК исходные операнды и результат (например, с логического адреса DS:500H в сегменте данных), а также регистры DX и AX процессора, в которых хранятся операнд N и промежуточные результаты соответственно.

3. Разрабатывается схема алгоритма (СА) вычисления выражения (2.2).

Эта схема (рис. 2.12) представляет собой минимально возможную последовательность операторов – действий, необходимых для вычисления (2.2). В каждом операторе указывается только одна операция, выполняемая определенной командой МП (см. рис. 2.10). Ее мнемоника, наряду с комментарием выполняемой операции, указывается справа от оператора СА. Последовательность этих мнемоник – это основа программы на языке ассемблера (см. пример 3.4).

Поскольку выполнение элементарных операций вычитания и сложения, к которым сводится вычисление выражения (2.2), производится только с помощью команд SUB и ADD типа «регистр–регистр», то предварительно из памяти в регистр-аккумулятор AX передается необходимый операнд.

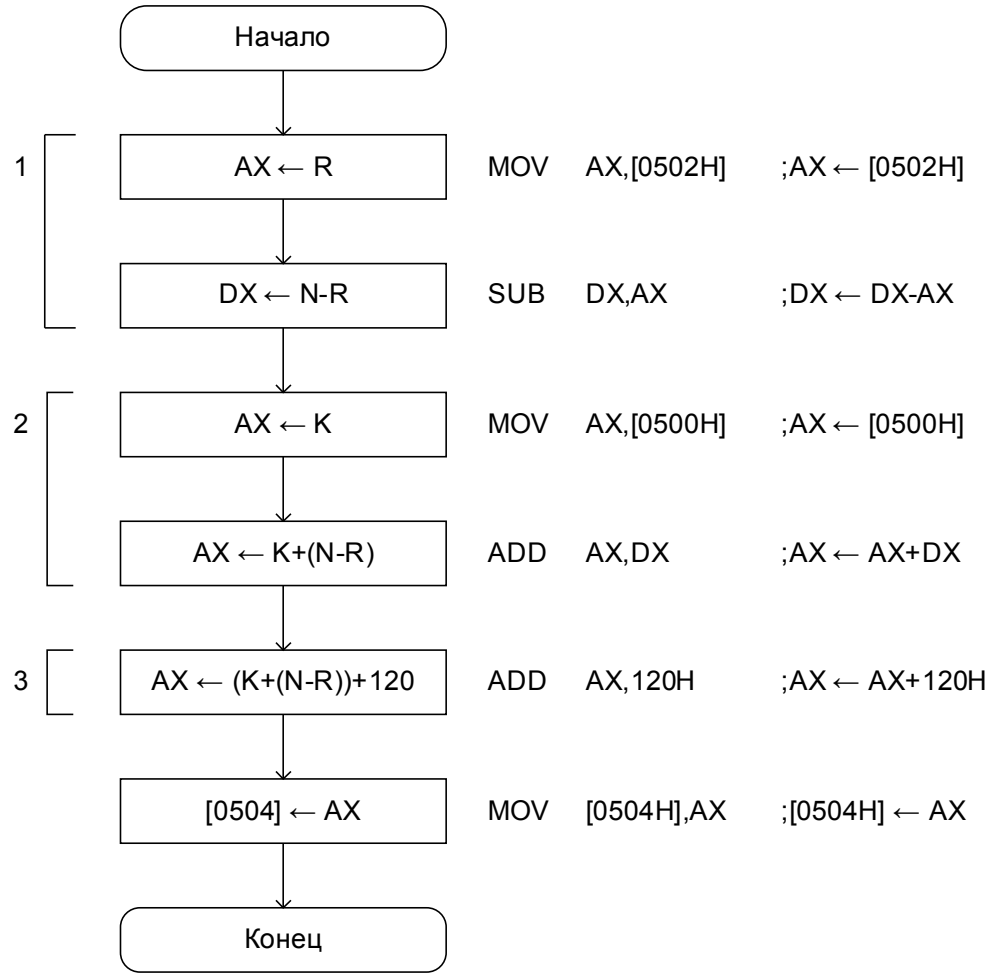


Рис. 2.12. Схема алгоритма вычисления выражения (2.2)

4. Осуществляется кодирование команд с представлением программы в машинных кодах, например в виде таблице 2.2. Кодирование можно осуществить вручную, как это делалось в первых ЭВМ, так и с помощью транслятора – ассемблера, если исходную программу для ре-

шения задачи (2.1) представить на языке ассемблера (см. п. 3.2, пример 3.4). Принципы ее разработки и трансляции (ассемблирования) будут рассмотрены в следующей главе пособия.

Начальный эффективный адрес программы в сегменте кодов, на который указывает сегментный регистр CS, равен 100H. Эффективные адреса данных лежат в диапазоне 0500–0505, а логические – в диапазоне DS:0500 – DS:0505.

Таблица 2.2

Последовательность кодов команд программы
вычисления выражения M = K + N – R + 120

Адрес	Код	Мнемоника команды	Операция
CS:0100	A1	MOV AX,[0502H]	AX ← R
0101	02		
0102	05		
0103	2B	SUB DX,AX	DX ← DX – R
0104	D0		
0105	A1	MOV AX,[0500H]	AX ← K
0106	00		
0107	05		
0108	03	ADD AX,DX	AX ← K + N – R
0109	C2		
010A	05	ADD AX,0120H	AX ← AX + 120
010B	20		
010C	01		
010D	A3	MOV [0504H],AX	[0504] ← AX
010E	04		
010F	05		
0110	90	NOP	Пустая операция
Адреса данных	Данные в ДК	Значения операндов	
DS:0500	60	K = 0060	Операнд K = +60
0501	00		
0502	F0	R = –0010	Операнд R = –10
0503	FF		
0504	XX	M = XXXX	Результат M
0505	XX		
РОН	Данные		
DX	0030	N = 0030	Операнд N = +30

Примеры мнемоник и кодов команд, сформированных в соответствии с их форматами (см. рис. 2.10), приведены ниже:

1. MOV AX,[0502H] ; AX ← [0502H]

W	Мл часть ЕА	Ст часть ЕА	Код
1 0 1 0 0 0 0 1	02_{16}	05_{16}	$\equiv A10205_{16}$

2. SUB DX,AX ; DX ← DX – AX

W	DX AX	Код
0 0 1 0 1 0 1 1	1 1 0 1 0 0 0 0	$\equiv 2BD0_{16}$

Аналогичным образом находятся мнемоники и коды других команд программы (см. табл. 2.2). Выполнение этих примеров закрепляет понимание того: как представляется команда в виде ее формата и мнемокода, как на его основе задаются режимы адресации данных, сколько байт содержит конкретная команда и т.д.

5. Далее осуществляются ввод программы в память ПК и ее отладка с помощью системной программы – отладчика td.exe. В частности эта программа входит в состав таких известных инструментальных средств программирования, как Си и Паскаль.

Отладчик дает возможность управлять процессом исполнения пользовательской программы в режиме отладки. Команды отладчика, вводимые с клавиатуры компьютера, позволяют выводить на экран монитора и изменять содержимое памяти и регистров МП, а также исполнять программу по шагам (командам) и другое.

2.6. Форматы команд МП 80x86 (для базового набора)

2.6.1. Формат двухоперандной команды

Длина команд МП 80x86 базового набора (для 16-разрядных операций) варьируется от одного до шести байт. Форматы команд для 32-разрядных программ будут рассмотрены в п. 3.6.

На рисунке 2.13 приведен наиболее типичный формат двухоперандной команды базового набора (третий и четвертый байты необязательны). В первых (одном или двух) байтах команды находятся код операции КОП и поля для указания режима адресации. Второй байт команды называют еще постбайтом.

7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 0	7 0
КОП D W	m o d r e g r / m	disp L	disp H

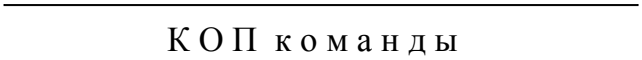


Рис. 2.13. Формат двухоперандной команды

В КОП команды имеются следующие однобайтовые поля.

Бит W определяет операцию с байтом (W = 0) и словом (W = 1).

Бит D показывает, чем является регистр reg: операндом-источником (D = 0) или операндом-получателем (D = 1).

В поле reg указывается 3-битовый адрес (номер) регистра (см. табл. 2.1), в котором размещается один из операндов.

Если на код операции команды отводится два байта (см, рис. 2.13), то во втором байте имеются 2-битовое поле режима адресации mod и 3-битовое поле r/m. В зависимости от значения mod поле r/m определяет операнд в РОНе МП, если mod = 11, или в памяти, если mod ≠ 11.

Для mod = 11 адрес регистра МП, в котором размещен операнд, указывается полем r/m в соответствии с таблицей 2.1.

Если mod = 11, а D = 1, то общий формат двухоперандной команды (рис. 2.13) трансформируется в двухбайтовую команду типа «регистр – регистр». Ранее, на рисунке 2.10 были приведены форматы таких команд: MOV reg1,reg2; ADD reg1,reg2; SUB reg1,reg2, в которых обозначены: reg1 = reg; reg2 = r/m. В качестве регистра-приемника выбран регистр reg, поэтому в них D = 1.

Если mod ≠ 11, то адрес EA операнда в памяти для режимов косвенной адресации, кроме варианта mod = 00 и r/m = 110 для прямой адресации, вычисляется в соответствии с таблицей 2.3.

Таблица 2.3

Способы вычисления адреса EA операнда в памяти

mod r/m	00	01	10
000	(BX) + (SI)	(BX) + (SI) + disp L	(BX) + (SI) + disp H,L
001	(BX) + (DI)	(BX) + (DI) + disp L	(BX) + (DI) + disp H,L
010	(BP) + (SI)	(BP) + (SI) + disp L	(BP) + (SI) + disp H,L
011	(BP) + (DI)	(BP) + (DI) + disp L	(BP) + (DI) + disp H,L
100	(SI)	(SI) + disp L	(SI) + disp H,L
101	(DI)	(DI) + disp L	(DI) + disp H,L
110	disp H,L	(BP) + disp L	(BP) + disp H,L
111	(BX)	(BX) + disp L	(BX) + disp H,L

Отметим, что для mod = 00 и r/m = 110 смещением disp H,L задается 16-битовый эффективный адрес EA операнда в памяти в режиме абсолютной (прямой) адресации.

Если mod = 01, то третий байт команды содержит 8-битовое смещение disp L, которое при вычислении EA автоматически расширяется со знаком до 16 бит. Если mod = 10, то третий и четвертый байты команды содержат 16-битовое смещение disp H,L.

Примеры мнемоник и кодирования двухоперандных команд

1. MOV AX, [SI] ; Команда передачи AX ← M_{SI} с косвенной регистровой адресацией.

КОП	D	W	mod	AX	r/m		Код
1 0 0 0 1 0 1 1			0 0 0 0 0 1 0 0			≡	8B
8	B		0		4		04

2. MOV [DI-10H],CX ; Команда передачи M_{DI-10H} ← CX с индексной адресацией.

Смещение disp L = –10H задается в третьем байте в ДК как $F0_{16}$.

D	W	mod	CX	r/m			Код
1 0 0 0 1 0 0 1		0 1 0 0 1 1 0 1			$F0_{16}$	≡	89
							4D
							F0

3. ADD [BX+200H],AX ; Команда сложения $M_{BX+200H}$ ← $M_{BX+200H}$ + +AX с использованием базовой адресации.

Поля: mod = 10, r/m = 111. Смещение disp H,L = 0200H задается в третьем и четвертом байтах.

D	W	mod	AX	r/m					01
0 0 0 0 0 0 0 1		1 0 0 0 0 1 1 1			00_{16}	02_{16}	≡		87
									00
									02

4. MOV DX, [0502H] ; Команда передачи DX ← M_{0502} с прямой адресацией.

Поля: mod = 00, r/m = 110. В 3-м и 4-м байтах размещаются младшая 02_{16} и старшая 05_{16} части эффективного адреса EA = 0502H.

D	W	mod	DX	r/m				8B
1 0 0 0 1 0 1 1		0 0 0 1 0 1 1 0			02_{16}	05_{16}	≡	16
								02
								05

2.6.2. Формат двухоперандной команды с непосредственным операндом-константой (рис. 2.14)

К О П S W	mod КОП r/m	disp L	disp H	data L	data H

К О П команды	Смещение	Операнд-константа

Рис. 2.14. Формат команды с непосредственным операндом data

Назначение полей mod и r/m и смещения в этой команде сохраняется таким же, как и для команды, приведенной на рисунке 2.13.

Бит S совместно с W (поле S:W) определяет размер непосредственного операнда *data*:

$$SW = \begin{cases} 00 - \textit{один байт data L}; \\ 01 - \textit{два байта data H, L}; \\ 11 - \textit{один байт data L, расширенный до двух байт}. \end{cases}$$

Операнд-константа всегда является источником. Например, в команде MOV [BX+200H],160H загрузки константы 160H в память ее значение размещается в пятом и шестом байтах, а КОП команды и смещение – в байтах 1–2 и 3–4 соответственно. Код этой команды содержит следующую последовательность байт: C7 87 00 02 60 01_{16}.

Микропроцессоры 80x86 имеют в своем составе команды *специального, укороченного формата*, которые позволяют сократить постбайт в часто используемых командах. Это команды пересылки в регистр reg, операций с аккумулятором acc и непосредственным операндом data. Ранее на рисунке 2.10 были приведены примеры команд с укороченным форматом: MOV reg,data ; MOV acc,[EA] ; ADD acc,data ; MOV [EA],acc ; SUB acc,data.

Следует отметить, что при трансляции программы ассемблер всегда выбирает более короткий формат транслируемой команды.

2.6.3. Формат команд с относительной адресацией

Относительная адресация применяется в командах условных и безусловных переходов для вычисления адреса EA команды, к которой осуществляется переход. Формат команд переходов с относительной адресацией содержит два байта. В первом байте размещается КОП, а во втором – смещение disp L:

К О П	Смещение disp L

Эффективный адрес перехода EA вычисляется как сумма 8-битового смещения disp L и текущего значения указателя команд IP, то есть EA = (IP) + disp L.

Рассматриваемый режим адресации обеспечивает переход в диапазоне –128 ÷ +127 байт относительно адреса текущей команды, равного

IP = IP + 2. Отрицательное смещение означает переход назад. Его значение указывается в команде в дополнительном коде ДК.

Пример 2.2

Код команды условного перехода JNZ MM1

	Адрес команды	Код команды	Мнемоника команды	Операция
EA:	0200	05	MM1: ADD AX,304h	
	0201	04		
	0202	03		
IP:	0203	75	JNZ MM1	;IP ← (IP + 2) + FB_{16},
	0204	FB		;если бит FZ = 0
(IP+2)	0205	90	NOP	;Следующая команда

Чтобы определить ДК смещения disp L для команды условного перехода по не нулю (не равно) JNZ MM1 (или JNZ 200H), необходимо вначале найти его значение:

<Смещение disp L> = EA – (IP + 2).

Для данного примера <Смещение disp L> = 200 – (203 + 2) = –5, где EA = 200 – эффективный адрес команды, к которой осуществляется переход (это команда с меткой MM1); (IP) = 203 – это адрес команды перехода JNZ MM1. ДК[–5] = FB_{16}. Значение смещения FB_{16} указывается во втором байте команды JNZ MM1 после байта 75_{15}, задающего КОП этой команды.

2.7. Система команд (операций) МП 80x86

Содержит более ста базовых команд (приложение 8), которые делятся на определенные функциональные группы по виду выполняемых МП операций. В качестве операндов, с которыми оперируют эти команды, могут выступать байты, слова и двойные слова.

1. Команды передачи данных

1.1. Команды пересылки **MOV**

Среди них различают команды пересылки вида:

– регистр/память (r/m) ← константа (data);
– регистр (reg) ← регистр (reg);
– память (m) ← регистр (reg);
– регистр (reg) ← память (m);
– сегментный регистр (sreg) ← регистр (reg).

Примеры команд из этой группы:

```
MOV     DX,100H         ;DX ← 100H
MOV     AX,BX           ;AX ← BX
MOV     [SI],AX         ;[SI] ← AX
```

```
MOV     AX,[BX+2]     ;AX ← [BX + 2]
MOV     DS,AX         ;DS ← AX
```

1.2. Команды обмена **XCHG** «*регистр–регистр» и «регистр–память».* Например:

```
XCHG  AX,BX ; AX <=> BX
```

1.3. Команда преобразования **XLAT**

Реализует операцию AL ← [BX + AL]. Применяется, например, для табличного преобразования десятичных цифр в их ASCII-коды.

1.4. Команда загрузки адреса **LEA**

Загружает в приемник эффективный адрес операнда в памяти. Например, команда LEA BX,X эквивалентна команде MOV BX,offset X загрузки в регистр BX адреса операнда X.

1.5. Команды обращения к стеку **PUSH** *и* **POP**.

Организация стека в МП 80x86

Стек – это определенным образом построенная память, работающая по принципу: «первый пришел – последний ушел». Данный вид памяти организуется в сегменте стека, на который указывает *сегментный регистр SS.*

В стек помещают данные в формате слова или двойного слова (для МП-32). На последний загруженный в стек элемент данных («вершину») указывает регистр, называемый *указателем стека* SP или ESP (для 32-разрядных данных).

Команды обращения к стеку:

– включить в стек содержимое регистра (спецформат):

```
PUSH    reg    ;стек ← регистр, SP ← SP – 2 (или 4)
```

Например, включить в стек регистры AX и ECX:

```
PUSH    AX   ;стек ← AX;  SP ← SP – 2
PUSH    ECX  ;стек ← ECX;  SP ← SP – 4
```

– включить в стек содержимое регистра/памяти:

```
PUSH    r/m   ;стек ← регистр/память, SP ← SP – 2 (или 4);
```

– извлечь из стека содержимое регистра (спецформат):

```
POP  reg   ;регистр ← стек, SP ← SP + 2 (или 4)
```

– извлечь из стека содержимое регистра или памяти

```
POP  r/m   ; регистр/память ← стек, SP ← SP + 2 (или 4)
```

Перед работой со стеком необходимо задать размер и начало («дно») стека. Для этого с помощью директивы .STACK задают его размер, а в указатель SP загружают начальный адрес стека:

```
.STACK   64          ;Размер стека 64 байта
 MOV     SP,100H     ;Начальный адрес стека 100H
```

Команды обращения к стеку применяются в процедурах для того, чтобы сохранить в нем для основной программы содержимое регистров, используемых в процедуре.

Например, если в программе задействованы регистры AX и DX, то их содержимое в начале процедуры включают в стек с помощью команд PUSH, а затем в конце выполнения процедуры восстанавливают посредством последовательности команд POP:

```
;Начало процедуры
PUSH     AX   ;Включить в стек AX
PUSH     DX   ;а затем DX
…             ;Команды процедуры
…
POP  DX   ;Извлечь из стека DX
POP  AX   ;а затем AX
RET       ;Возврат из процедуры
```

Имеются команды, которые включают и извлекают из стека признаки: **PUSHF** и **POPF**, а также, начиная с МП i286, содержимое всех его пользовательских регистров: **PUSHA** и **POPA**.

1.6. Команды ввода **IN** *и вывода* **OUT**

Применяются для обращения к портам ввода-вывода внешних устройств персонального компьютера.

2. Команды арифметических операций

2.1. Команды двоичного сложения и вычитания

В них в качестве источника и приемника операндов выступают регистры и память (константа является только источником):

– регистр/память (r/m) ← регистр/память (r/m) * константа (data),

– регистр/память (r/m) ← регистр/память (r/m) * регистр (reg),

– регистр (reg) ← регистр (reg) * регистр/память (r/m),

где * – выполняемая операция сложения или вычитания.

В зависимости от вида операции, задаваемой мнемокодом команды, различают следующие их виды:

– *команды сложения* **ADD** *и сложения с переносом* **ADC**.

Команда сложения ADC отличается от ADD тем, что к сумме операндов прибавляется значение переноса CF, например:

```
ADD AL,BL          ;AL ← AL + BL
ADC AL,BL          ;AL ← AL + BL + CF
```

– *команды вычитания* **SUB** *и вычитания с заемом* **SBB**, например:

```
SBB  EDX,EAX       ;EDX ← EDX – EAX – CF
```

– *команда сравнения* **CMP**.

При выполнении команды CMP осуществляется вычитание операндов без изменения результата с формированием всех характеризующих его признаков. Например, с помощью выполнения приведенной ниже команды формируется признак нуля ZF:

```
CMP  AL,34H        ;AL – 34H
```

В связи с этим в программах команда сравнения CMP часто предшествует команде условного перехода, например по нулю JZ.

2.2. Команды инкремента **INC** *и декремента* **DEC**

Осуществляют операцию инкремента (+1) и декремента (–1) содержимого регистра МП или памяти, например:

```
INC  BX;           BX ← BX + 1
```

2.3. Команда изменения знака **NEG**

Ее выполнение эквивалентно образованию дополнительного кода отрицательного числа, например:

```
NEG  BX            ;BX ←  BX + 1
```

2.4. Команды десятичной коррекции **DAA** и **DAS** *аккумулятора AL*

Поддерживают операции десятичной арифметики. Так, например, если после команды ADD AL,[SI] сложения двоично-десятичных BCD-чисел по правилам двоичной арифметики выполнить команду DAA, то в аккумуляторе AL будет получен истинный BCD-результат:

```
ADD  AL,[SI]   ;Сложить AL + [SI]
DAA            ;Десятичная коррекция AL
```

В системе команд МП 80x86 также есть команды коррекции аккумулятора AX и AL для операций умножения и сложения (вычитания) неупакованных двоично-десятичных чисел соответственно. В них для представления десятичной цифры необходим один байт.

2.5. Команды умножения целых двоичных чисел со знаком **IMUL** *и без знака* **MUL**

В команде указывается только один операнд. Второй сомножитель в зависимости от размера (байт, слово, двойное слово) находится в аккумуляторе AL, AX или EAX. Получаемое произведение фиксируется в

аккумуляторе и регистре расширения, то есть в AX, DX:AX или EDX:EAX.

Например:

```
IMUL   BL        ;AX ← AL × BL
IMUL   BX        ;DX:AX ← AX × BX
```

В команде IMUL для МП-32 указывают источник и приемник.

2.6. Команды деления целых двоичных чисел со знаком **IDIV** *и без знака* **DIV**

В команде указывается только один операнд – делитель. Делимое в зависимости от размера по умолчанию находится в AX, DX:AX или EDX:EAX. Результат от деления (частное) фиксируется в AL, AX или EAX, а остаток от деления – в AH, DX или EDX.

Например:

```
IDIV  BL         ;AL ← AX /BL
IDIV  BX         ;AX ← DX:AX /BX
```

При делении возможно переполнение.

3. Команды логических операций и сдвигов

3.1. Команда инвертирования **NOT**

Образует инверсию обрабатываемого операнда, например:

```
 NOT  AX;        AX ← AX
```

3.2. Команды сдвиговых операций

Реализуют операции сдвига байта, слова или двойного слова, размещенного в регистре или памяти. Число сдвигов задается предварительно в регистре CL или указывается в команде в виде константы (в МП 8086 она равна единице). Команды МП семейства 80x86 поддерживают все типы сдвигов, рассмотренных в п. 1.7.4:

– *логический сдвиг влево* **SHL** *и вправо* **SHR**, например:

```
SHL   EAX,1      ;Логический сдвиг EAX влево на 1 бит
SHR   AX, CL     ;Логический сдвиг AX вправо на CL бит
```

– *арифметический сдвиг влево* **SAL** *и вправо* **SAR**;

– *циклический сдвиг влево* **RCL** *и вправо* **RCR** *через триггер переноса CF*;

– *циклический сдвиг влево* **ROL** *и вправо* **ROR**, *минуя триггер переноса CF.*

В МП-32 также есть команды *логического сдвига влево* **SHLD** *и вправо* **SHRD** *слов с двойной точностью*, например:

```
SHLD    AX,DX,CL      ;Сдвиг пары DX:AX влево на CL бит
```

3.3. Команды логических операций:

– *конъюнкция* **AND**;

– *дизъюнкция* **OR**;

– *неравнозначное ИЛИ* **XOR**;

– *логическое сравнение (тестирование)* **TEST**;

– *инверсия* **NOT**.

Они поддерживают операции, рассмотренные ранее в п. 1.7.4.

В качестве источника и приемника логических переменных (байта, слова или двойного слова) выступают регистры и память (константа является только источником), например:

```
AND   AX,[SI]    ;Операция конъюнкции AX ← AX & [SI]
TEST  AL,80h     ;Операция  AL & 80h тестирования
                 ;седьмого бита регистра AL
```

4. Команды передачи управления (безусловные)

Рассматриваемые команды при безусловном переходе (БП) к любой команде программы изменяют только содержимое указателей адресов команд CS и IP, устанавливая в них адрес перехода, то есть CS:IP ← <адрес перехода>. Различают следующие типы БП.

4.1. Команда безусловного перехода **JMP** операнд

Различают следующие типы этих команд:

– *с прямой адресацией трехбайтовая*, когда адрес перехода указывается во втором (мл. часть) и третьем байте команды, например:

```
JMP  200h ;IP ← 0200h
```

– *с относительной адресацией двухбайтовая* (см. п. 2.6.3):

```
JMP  <смещение>      ;IP ← (IP + 2) + смещение
```

– *с прямой адресацией пятибайтовая*, когда в третьем и втором байтах команды задается адрес для указателя команд IP, а в пятом и четвертом – адрес для сегментного регистра CS, обеспечивающий переход между сегментами, обозначаемый как FAR (дальний), например:

```
JMP  FAR M1    ;CS:IP ← адрес перехода на метку M1
JMP  500h 200h  ;IP ← 0500h, CS ← 0200h
```

– *с косвенной адресацией*, например:

```
JMP  BX                  ;IP ← BX, адрес перехода находится в BX
JMP  DWORD PTR [BX]     ;CS:IP ← [BX], адрес перехода
```

;типа FAR размещен в памяти

4.2. Команда вызова подпрограммы (ПП) и процедуры **CALL** операнд

Команда CALL <адрес ПП> отличается от JMP тем, что при ее выполнении, наряду с операцией CS:IP ← адрес ПП, предварительно в стеке сохраняется адрес возврата, то есть адрес следующей за **CALL** команды основной программы.

Команда может быть использована для вызова ближней или дальней процедуры, например: CALL <имя процедуры> или CALL FAR <имя процедуры>, например: CALL FAR SUMMA.

Команда CALL имеет те же форматы и режимы адресации вызовов (переходов), что и JMP, за исключением короткого 2-байтового формата с относительной адресацией.

4.3. Команда возврата **RET**

Является завершающей командой ПП или процедуры. При ее выполнении осуществляется возврат к следующей за **CALL** команде основной программы путем выполнения операции: IP ← стек (то есть IP ← адрес возврата) для возврата из ближней процедуры или CS:IP ← стек (то есть CS:IP ← адрес возврата типа FAR).

5. Команды условного перехода (УП) Jcc метка, где cc – мнемоника условного перехода, и цикла LOOP

Являются 2-байтовыми командами с относительной адресацией (см. п. 2.6.3).

5.1. Команды условного перехода **Jcc** метка

В них в качестве условия перехода выступают значение одного признака или совокупность значений 2–3 битов – признаков результата, устанавливаемых командой, которая была выполнена перед командой УП, например:

```
JS   MM1  ;Переход к метке MM1, если признак знака
;SF равен 1 (переход, если знак отрицательный)
```

5.2. Команда цикла **LOOP** метка

Осуществляет декремент счетчика циклов CX, а затем проверяет его значение: если CX ≠ 0, то переход к метке.

6. Команды прерывания INT n и возврата IRET

6.1. Двухбайтовая команда **INT n**, содержащая во втором байте тип (вектор) прерывания n, схожа по механизму действия с командой межсегментного вызова CALL с косвенной адресацией.

Значение кода n в диапазоне [0; 0FFh] задает адреса 4×n ячеек основной памяти, в которых в формате двойного слова заранее загружаются адреса процедур (обработчиков) прерывания, передаваемые далее при выполнении команд INT n в CS:IP.

При выполнении INT n реализуются следующие операции:

1) стек ← содержимое регистра признаков F процессора и адрес возврата в виде текущего значения CS:IP;
2) CS:IP ← адрес процедуры прерывания;
3) IF ← 0 (сброс триггера прерывания).

6.2. Команда возврата из процедуры прерывания **IRET**

Возвращает из стека содержимое регистра признаков, а в регистры CS:IP – значение адреса возврата.

Рассмотренные команды обеспечивают поддержку *системы прерывания ЭВМ* для вызова процедур обслуживания прерываний.

Виды команд прерывания, их применение

1. Команды программного прерывания INT n, включаемые в программу для вызова процедур обработки прерывания, в том числе базовой системы ввода-вывода BIOS, например функций видеосистемы INT 10h и операционной системы DOS, например INT 21h.

2. Команды аппаратного прерывания INT n, формируемые аппаратно с помощью БИС программируемого контроллера прерывания (ПКП), связанного с внешними устройствами, которые работают в режиме ввода-вывода данных по прерыванию.

7. Команды управления микропроцессором

Управляют сбросом и установкой в «1» отдельных бит регистра признаков, а также остановом процессора и другим.

8. Команды операций с цепочками данных

Предназначены для программирования операций по обработке данных, которые представлены в виде цепочек (последовательностей) слов (см. далее п. 3.8).

3. ОСНОВЫ РАЗРАБОТКИ И ОТЛАДКИ ПРОГРАММ НА ЯЗЫКЕ АССЕМБЛЕРА IBM PC

3.1. Элементы языка ассемблера МП 80x86, понятие о макропрограммировании

Программа на языке ассемблера представляет собой последовательность предложений (операторов), описывающих выполняемые операции только конкретного типа МП. В этой программе различают два вида операторов: *командные* и *директивы ассемблера*.

Командные операторы могут содержать до четырех полей (в скобках указаны необязательные поля):

[Метка:] Мнемокод [Операнд] [;Комментарий]

Например:

```
MM1:    MOV    AX,DX    ;Переслать в AX содержимое регистра DX
```

Поле метки не должно начинаться с цифры. Метки могут содержать символы верхнего и нижнего регистров клавиатуры: буквы, цифры, символы подчеркивания (_) и знаки $, @, ?. Максимальная длина метки 31 символ.

Поле мнемокода содержит *мнемонику команды микропроцессора,* например MOV, ADD.

Поле операнда. В нем указываются данные, с которыми работает команда.

Поле комментария всегда начинается с символа «;».

Для описания операндов могут быть использованы метки (например, MM1), константы, регистры МП (AX, BX и другие), переменные памяти (k, r и другие), текстовые строки, выражения (x + 10).

В численной константе основание системы счисления определяется буквой, указанной после константы (B – двоичная, O – восьмеричная, D (или ничего) – десятичная, H – 16-ричная). Константа всегда должна начинаться с цифры. Символьная константа – это символ, заключенный в кавычки. Строковые константы также заключаются в кавычки. Примеры констант: 10001111B, 23O, 45, 0A2H, '4', 'Таблица'.

Программа на ассемблере как *машинно-ориентированном языке программирования* с помощью транслятора автоматически преобразуется в ее объектную программу путем преобразования каждого командного оператора, содержащего мнемокоманду МП, в соответствующий ей машинный код, то есть «один к одному».

Директивы (или псевдооператоры) ассемблера в отличие от командных операторов при трансляции не генерируют кодов команды. Их основное назначение – определение в результате ассемблирования программы областей памяти для размещения данных и команд, запись в эти области их машинных кодов, определение начала и конца программы, задание моделей памяти и другого.

Директивы языка ассемблера имеют следующий формат:

[Метка] Псевдооператор [Операнд] [;Комментарий]

В таблице 3.1 приведены примеры записи в программах директив ассемблера tasm.exe фирмы Borland.

Таблица 3.1

Примеры записи директив ассемблера tasm

Директива	Пример записи			
ORG		ORG	100H	;Программа с адреса 100H
EQU	ALFA	EQU	13H	;Константе с именем ALFA ;присвоить значение13H
DB	K	DB	60H	;Для переменной с именем K ;резервируется ячейка (байт) памяти, ;в которую заносится 60H
DW	X	DW	160H	;Для переменной с именем X ;резервируются 2 ячейки (слово) ;памяти, в которые заносятся 60H ;и 01H
	BUF	DW	8 DUP(?)	;Для переменной BUF резервируется ; 8 слов памяти
	TABL	DB	‘Диск’	;Для переменной TABL ;резервируются ячейки памяти, ;в которые заносится последователь-;ность ASCII-символов «Диск»
PROC	TUR	PROC		;Начало
ENDP	TUR	ENDP		;и конец процедуры с именем TUR
.CODE				;Упрощенные сегментные директивы
.DATA				;сегмента кодов,
.STACK	.STACK	100H		;сегмента данных,
.MODEL	.MODEL	tiny		;сегмента стека размером 100H байт
END		END		;Модель памяти размером 64 Кбайт ;Конец текста программы

Рассмотрим применение основных директив ассемблера.

1. ORG 100H – директива, определяющая начальный адрес программы или данных, например, ORG 500H.

2. Директивы (псевдооператоры) данных – служат для определения констант и данных в памяти. Из них следует выделить:

а) *EQU* и *знак равно* «=» – псевдооператоры определения или присваивания. Например:

```
ALFA    EQU 13H   ;Константе ALFA присвоить значение 13H
```

Применять директиву EQU можно в любом месте программы (обычно там, где определяются данные).

Псевдооператор = отличается от EQU тем, что позволяет переопределять значение константы, например: BETA = 10H.

Переменная, определенная ранее с помощью директивы EQU или =, выступает, например, в команде MOV CX,ALFA в роли константы, непосредственно заданной в коде этой команды.

б) *DB, DW, DD, DQ – псевдооператоры определения данных*.

С их помощью определяют операнд (данные) размером: байт (DB), слово (DW), двойное слово (DD), квадрослово (DQ) и задают его значение с размещением в памяти. Числа со знаком размещаются в памяти в дополнительном коде (ДК). Начальный эффективный адрес их размещения задают с помощью директивы ORG.

Пример 3.1

```
        ;Определение данных в памяти с адреса 500H
         ORG 500H
K       DW  +60H        ;Операнд – слово K = +60
N       DW  -10H        ;Операнд – слово N  =  –10
R       DB  +3          ;Операнд – байт R  = +3
```

После трансляции программы, содержащей приведенный фрагмент, данные со знаком будут представлены и размещены в памяти в ДК, начиная с адреса DS:0500H, следующим образом:

Память		Данные
Адрес	7 0	
DS:0500	60	Операнд K = +60H в формате слова в ДК
0501	00	
0502	F0	Операнд N = –10H в формате слова в ДК
0503	FF	

0504 | 03 | Операнд R = +3H в формате байта в ДК

Подобное определение переменных позволяет в командных операторах записывать их в виде символических имен. Например:

```
MOV     AX,N      ; AX ← N
```

Приведенный оператор эквивалентен по своему действию команде MOV AX,[0500H], содержащей эффективный адрес [0500H] операнда N (см. табл. 2.2). Однако запись команды с указанием адреса [0500H] при составлении программ на ассемблере использовать не рекомендуется.

В случае если при определении данных директива ORG отсутствует, их размещение производится транслятором с нулевого значения эффективного адреса EA памяти.

Пример 3.2

Применение директив определения данных:

– задание таблиц и массива данных:

```
X    DB   1,2,-3,3,4,-5,6,7,8,9,0AH  ;Массив чисел со знаком
```

– резервирование области памяти:

```
Y    DB   0,0,0,0       ;Разместить в памяти четыре нуля
Y    DB   4 DUP(0), 5,-3  ;Разместить 4 нуля, а затем 5 и –3
```

– задание в памяти строковых данных:

```
Text DB   ‘Петров’; Разместить в памяти строку «Петров»
```

При размещении строки в памяти по младшему адресу в виде ASCII-кода размещается ее первый символ, затем второй и т.д.

3. Директивы определения процедуры PROC и ENDP

Отмечают начало и конец процедуры, содержащей блок командных операторов, который завершается командой возврата RET. Процедура вызывается из основной программы с помощью команды вызова CALL или команды прерывания INT. В последнем случае процедура прерывания завершается командой возврата IRET.

Пример 3.3

Процедура SUMMA сложения чисел **d ← d + s**, размещенных в текущем сегменте памяти (см. пример 2.1).

```
SUMMA   PROC
        PUSH    AX
        MOV     AX,[SI-2]  ; AX ← s (индексная адресация s)
        ADD     [SI],AX    ; d ← d + s (косв. рег. адресация d)
        POP     AX
        RET
```

```
SUMMA   ENDP
```

Различают ближнюю (NEAR) и дальнюю (FAR) процедуры. Процедура с атрибутом NEAR (или по умолчанию, как это сделано в примере 3.3) может быть вызвана из текущего сегмента кодов (программы), а с атрибутом FAR – из любого сегмента. Процедуры оформляются в конце основной программы, но до директивы END.

4. Директивы макроопределения MACRO и ENDM

Программы, написанные на языке ассемблера, часто содержат повторяющиеся участки. Такой участок можно оформить в виде *макроопределения (*или *макрокоманды)*, характеризуемого произвольным именем и необязательным списком формальных аргументов.

Начало *макроопределения* (или *макроса*) определяется директивой MACRO с необязательным списком, а ее конец – директивой ENDM. Так, приведенный в примере 3.3 текст программы можно представить в виде макроопределения, например с именем:

```
SUM_SD  MACRO
        MOV     AX,[SI-2]
        ADD     [SI],AX
        ENDM
```

Вызов выполнения такого участка (макровызов) производится путем включения в программу макрокоманды с принятым именем. При каждом появлении в программе операторной строки, содержащей макрокоманду, транслятор генерирует для нее последовательность кодов команд, реализующих повторяющийся участок программы.

Понятие о макропрограммировании

Программирование с применением макрокоманд называют *макропрограммированием*. Его применение обеспечивает более высокий уровень машинно-ориентированных языков программирования. При этом программист оперирует макрокомандами, отражающими действия последовательности нескольких машинных команд («один к нескольким»). Это приводит к более коротким текстам программ на языке ассемблера и сокращению времени на их разработку за счет использования полученных ранее макроопределений.

Введение принципов макропрограммирования при разработке программ позволяет существенно сократить время выполнения программы с применением макрокоманд. Это сокращение обеспечивается

за счет исключения из основной программы достаточно медленных команд обращения CALL к процедуре, выполняющей операцию, идентичную операции, которая реализуется последовательностью команд, генерируемых макроопределением. Основной недостаток макропрограммирования – это увеличение длины объектной программы в байтах, вызванное повторениями одинаковых последовательностей кодов команд МП там, где встречается макрокоманда.

Содержимое макроопределения (также как и процедуры) задается при программировании отдельно от основной программы, из которой оно вызывается. Примеры формирования и вызова процедур и макроопределений даны далее в п. 3.7 при программировании операций вывода текста на экран.

5. Директива управления трансляцией END означает конец программы с определенным именем.

Например, псевдооператор END BEGIN отмечает конец программы с именем BEGIN. Поэтому начало исходной программы должно иметь метку, например, BEGIN. Программу в целом также можно определить как процедуру.

Если директива END не имеет метки, то она используется для завершения вспомогательных ассемблерных модулей, которые могут быть вызваны из основного модуля, обязательно завершаемого как END <имя основного модуля>.

6. Упрощенные директивы определения начала сегментов **.STACK, .CODE** и **.DATA** задают начало сегментов стека (и его объем), программы и данных соответственно.

Прежде чем получить доступ к ячейкам памяти сегмента данных, необходимо загрузить в сегментный регистр ds начальный сегментный адрес с помощью последовательности команд:

```
mov  ax,@data
mov  ds,ax
```

Сегментные регистры cs и ss не надо выставлять на начальные адреса сегментов кода и стека. Это сделает операционная система при загрузке программы.

Для задания сегментов могут быть также использованы директивы **SEGMENT (**начало) и **ENDS (**конец) сегмента. Однако они имеют сложную форму записи, вследствие чего при разработке несложных программ целесообразно применять сокращенные сегментные директивы.

7. Директива определения типа моделей памяти

Директива **.MODEL <тип модели памяти>** задает одну из следующих возможных моделей памяти, например:

– tiny – программа и данные в одном сегменте объемом 64 КБ (или 64 Кбайт); применяется только ближняя адресация команд и данных (меняется только смещение – эффективный адрес EA);

– small – программа в одном сегменте, данные в другом сегменте размером по 64 КБ каждый; используется ближняя адресация (NEAR) команд и данных;

– medium – программа больше 64 КБ, данные в сегменте 64 КБ; адресация команд дальняя (FAR), данных — ближняя (NEAR);

– compact – обратная по своим характеристикам с medium;

– large – программа и данные больше 64 КБ, дальняя адресация команд и данных, массив данных не превышает 64 КБ;

– huge – программа и данные больше 64 КБ, дальняя адресация команд и данных, массив данных больше 64 КБ.

Наиболее целесообразна модель small, так как в ней все переходы ближние и поэтому программа будет выполняться быстрее. При создании com-файлов применяется модель tiny.

Директива .MODEL указывается в начале программы.

По умолчанию ассемблер генерирует коды для 16-разрядного МП i8086. Если он встретит команду 32-разрядного микропроцессора, то транслятор сгенерирует ошибку, что указывает на другой тип используемого МП. Поэтому необходимо в программе указывать тип МП, например **.486** или **.586** для МП Pentium.

8. Операции языка ассемблера

*1. Операция **OFFSET** вычисления (возвращения) эффективного адреса операнда, размещенного в памяти.*

Например, при трансляции команды MOV BX,OFFSET K данная операция позволяет ассемблеру определить адрес операнда K, размещенного в памяти. После выполнении этой команды значение адреса K, например, равное 500H, будет загружено в регистр BX.

*2. Операция присваивания атрибута **PTR*** позволяет совместно с атрибутом, например, BYTE (байт) или WORD (слово), задать в команде длину операнда в случае неоднозначного понимания его размера без этой операции. Например, команды инкремента байта и слова в памяти, адресуемой с помощью указателя BX, с использованием этих атрибутов имеют соответственно вид:

```
INC  BYTE  PTR  [BX]      ;Инкремент байта
INC  WORD PTR [BX]        ;и слова памяти
```

3.2. Разработка программ на языке ассемблера для генерирования исполняемых com-модулей

Далее приведены два примера 3.4 и 3.5, иллюстрирующие разработку программ на языке ассемблера микропроцессора 80x86 для генерирования на их основе исполняемых модулей с расширением .com. Каждая из этих программ вычисляет выражение (2.1).

В приведенных программах директива .MODEL tiny задает модель памяти как один сегмент кодов размером 64 Кбайт. Для его задания используется упрощенная сегментная директива .CODE. Отметим, что при генерировании исполняемого модуля типа .com программа и данные в памяти задаются в одном кодовом сегменте, а типа .exe – в отдельных сегментах: кода и данных.

Пример 3.4

```
;Программа вычисления выражения M = K + N – R + 120H
;для исполняемого модуля типа .com
;Операнды-слова k, r, m размещены в памяти, начиная с адреса 500h,
;n – в регистре DX
        .MODEL   tiny          ;Программа и данные размещены
                               ;в сегменте кодов объемом 64 КБ
        .CODE                  ;Начало кодового сегмента
;Размещение данных с адреса 500h
        org      500h
k       dw       +60h
r       dw       -10h
m       dw        0
;Команды программы с адреса 100h
        org      100h
        begin:                 ;Метка начала программы
        mov      ax,r          ; ax ← r
        sub      dx,ax         ; dx ← dx – r
        mov      ax,k          ; ax ← k
        add      ax,dx         ; ax ← ax + dx
        add      ax,120h       ; ax ← ax +120
        mov      m,ax          ;Передать результат m в память
        mov      ah,4ch        ;Функция DOS выхода
        int      21h           ;из программы
        END      begin         ;Конец программы
```

В программах для указания начального адреса данных 500h используется директива org 500h. Операнды в формате байта или слова задаются после org 500h с помощью директив db или dw соответственно. Директива END begin указывает транслятору на конец программы.

Пример 3.5

```
;Программа вычисления выражения M = K – R + N + 120H
;для исполняемого модуля типа .com для случая базовой адресации
;данных, когда K, R, N, M размещены в памяти с адреса 500h
        .MODEL  tiny            ;Программа и данные размещены в
                                ;сегменте кодов размером 64 Кбайт
        .CODE                   ;Начало кодового сегмента
;Размещение данных с адреса 500h
        org     500h
k       dw      +4h
r       dw      -1h
n       dw      +2h
m       dw       0
;Команды программы с адреса 100h
        org     100h
        begin:                  ;Метка начала программы
        mov     bx,offset k     ;Загрузить в bx адрес операнда k
        mov     ax,[bx]         ;Загрузить в ax операнд k
        sub     ax,[bx+2]       ;Получить в ax разность k – r
        add     ax,[bx+4]       ;ax ← ax + n
        add     ax,120h         ;ax ← ax + 120h
        mov     [bx+6],ax       ;Загрузить результат m в память
        mov     ah,4ch          ;Функция DOS выхода
        int     21h             ;из программы
        END     begin           ;Конец программы
```

Комментарии к программе (пример 3.4)

1. Порядок разработки программы на языке ассемблера рассмотрен в п. 2.5. При выполнении его четвертого этапа составляется программа вычисления заданного выражения на языке ассемблера как исходная форма для автоматического получения машинных кодов команд путем ее трансляции (ассемблирования). Для упрощения разработки программы (пример 3.4) целесообразно воспользоваться последовательностью команд МП, представленной в алгоритме (рис. 2.12) вычисления выражения 2.2. При этом необходимо приведенные в мнемониках ко-

манд адреса операндов заменять их мнемоническими обозначениями. Так, адреса [0500h], [0502h] и [0504h] заменяют на имена переменных k, r и m соответственно.

2. Если задано, что данные представлены в формате байта, то они описываются в программе с помощью директивы db.

Комментарии к программе (пример 3.5)

1. В рассматриваемой программе все переменные размещены в памяти, начиная с адреса 500h. Обращение к ним производится с использованием базовой адресации. При этом выборка необходимого операнда совмещается с операциями вычитания или сложения.

2. В приведенной программе можно применять, например, только косвенную регистровую адресацию данных (см. п. 2.3.2). В этом случае модификация указателя памяти BX на +2 для установки в нем четных адресов 502h, 504h и 506h перед выборкой переменных R, N и M из памяти может быть осуществлена путем последовательного выполнения двух команд инкремента INC BX.

3. При представлении данных в формате байта модификация адреса операнда в указателе BX производится только с помощью одной команды INC BX.

3.3. Системное программное обеспечение ПК для разработки программ на языке ассемблера МП 80x86

3.3.1. Разновидности программного обеспечения для разработки программ на языке ассемблера

В состав программного обеспечения (ПО) микрокомпьютерных, или микропроцессорных систем (МПС) входят:

– программное обеспечение пользователя, содержащее алгоритмы и программы решаемой задачи;

– системное программное обеспечение (СПО).

С помощью СПО осуществляют создание (редактирование) исходных текстов программ, генерирование исполняемых модулей, их запуск и поддержку отладки. Различают следующие виды СПО: резидентное и кросс-СПО (рис. 3.1).

Кросс-СПО включает в себя системные программы (кросс-ассемблер, кросс-отладчик), которые работают на ПК, содержащем микропроцессор, отличающийся своей системой команд от микропроцессора, входящего в состав МПС (рис. 3.1б).

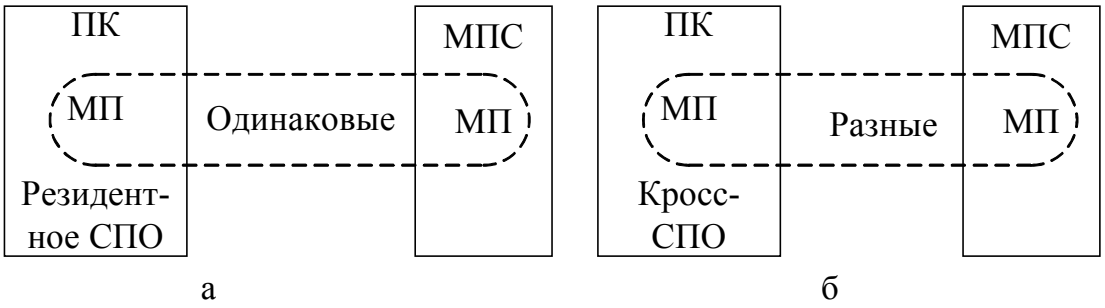


Рис. 3.1. Структура резидентного (а) и кросс-СПО (б)

Кросс-СПО широко используется в ПК в качестве эмуляторов микропроцессоров и микроконтроллеров, применяемых в различных специализированных микрокомпьютерных системах.

3.3.2. Состав резидентного СПО IBM PC для разработки и отладки программ на языке ассемблера

Системные программы, образующие СПО, хранятся на диске компьютера. Возможный набор системных программ для создания исполняемых модулей может содержать: редактор (например, встроенный в оболочку Total Commander, Norton Commander или другие); ассемблер (транслятор) tasm.exe; редактор связей tlink.exe; турбоотладчик td.exe (рис. 3.2). В частности, указанные файлы обычно входят в состав каталога BIN программных пакетов, поддерживающих программирование на языках высокого уровня Си и Паскаль.

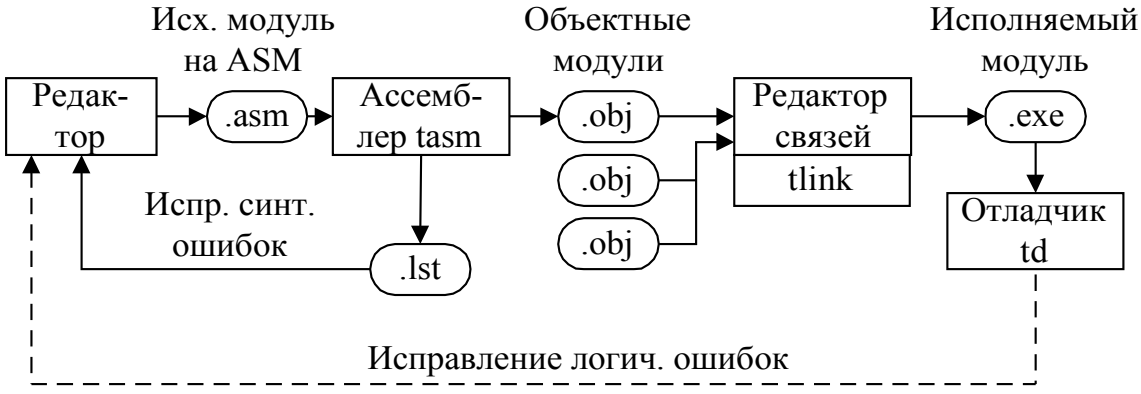


Рис. 3.2. Этапы разработки и отладки программ на языке ассемблера

Рассматриваемое базовое СПО является эффективным средством автоматизации процесса разработки и отладки прикладных программ на языке ассемблера МП семейства 80x86. На рисунке 3.2 показаны этапы разработки и отладки программ на языке ассемблера на основе рассматриваемого состава СПО.

В состав базового СПО для разработки программ на языке ассемблера МП 80х86 входят следующие системные программы.

1. Турбоассемблер tasm.exe

Представляет собой системную программу ПК, осуществляющую преобразование исходной программы на языке ассемблера МП 80х86 в объектную программу.

Исходную программу, еще называют исходным модулем типа .asm, транслятор (ассемблер) преобразует в перемещаемый объектный модуль типа .obj, который наряду с кодами команд содержит служебную информацию, необходимую для формирования далее исполняемого модуля.

Трансляция запускается командой, набираемой в командной строке, например, оболочки Norton Commander:

tasm [опции] <имя файла>.asm, , ↵

Если в команде будет указана одна запятая, то файл листинга программы с расширением .lst не формируется.

Перечень опций может быть вызван командой tasm↵. Отметим наиболее используемые опции:

/zi – включает в объектный модуль информацию для отладки;

/n – подавляет вывод таблицы символов в листинге.

В результате трансляции также выявляются синтаксические ошибки, допущенные при создании программы. Сведения о них сообщаются в листинге программы в виде файла с расширением .lst.

2. Редактор связей (компоновщик) tlink.exe

Объединяет созданные ассемблером объектные модули в единый исполняемый модуль с расширением .exe или .com для размещения его в основной памяти ПК для последующего исполнения. Редактор связей запускается командой

tlink [опции] <объектные файлы> ↵

Отметим наиболее употребляемые опции:

/x – не создавать .MAP-файл с картой исполняемой программы;

/v – включить символьную информацию для отладчика;

/t – создать исполняемый com-файл;

/3 – создать исполняемый модуль для 32-разрядных МП.

3. Отладчик td.exe

Управляет процессом отладки программ и является инструментом для поиска логических ошибок в выполняемой программе.

Далее для примера 3.4 приводится порядок ее создания (редактирования), трансляции и отладки.

3.3.3. Порядок ввода и отладки программы (пример 3.4) с исполняемым com-модулем

Создание исходного файла программы

1. Находясь в системе (оболочке), например Norton Commander, Total Commander или другие, при нажатой клавише Shift нажмите клавишу F4 – система выдаст запрос. Введите в нем имя файла и его расширение .asm (например, petrov.asm) и нажмите Enter (↵) – вы войдете в редактор. Наберите в нем программу (пример 3.4) на языке ассемблера. Специально допустите 2–3 ошибки. Далее при трансляции эти ошибки будут обнаружены, и вы их исправите.

2. Сохраните введенный текст – для этого используйте в оболочке функции его меню или нажмите клавишу F2 (Сохранить), а затем F10 (Выйти).

3. Если требуется скорректировать или дополнить известный файл, то в панели оболочки курсором выделите имя этого файла и нажмите F4 – вы попадете в редактор с вызванным файлом. Чтобы его сохранить после коррекции, используйте в оболочке функции его меню или нажмите F2 (Сохранить), а затем F10 (Выйти).

Ассемблирование

4. С помощью команды

tasm.exe <имя файла>.asm, ,↵

введенной в командной строке оболочки, запустите ассемблер. *Не забудьте ввести две запятые*.

Сформированные в результате ассемблирования объектный модуль программы и ее листинг записываются на диск соответственно в виде файлов с расширением .obj и .lst.

5. Повторите действия 3–4, если эти файлы в результате ассемблирования не получены, это объясняется наличием грубых (фатальных) ошибок в исходном файле. Например, в тексте программы допущены ошибки в наборе обозначений директив.

Если в результате трансляции сформирован только файл с расширением .lst, то это означает, что в программе имеют место несуществен-

ные синтаксические ошибки, например возникшие при наборе мнемоник команд. В этом случае перейдите к пункту 6.

6. Выделите файл с расширением .lst и нажмите клавишу F3 или F4. С помощью клавиш перемещения курсора просмотрите весь листинг и зафиксируйте в нем ошибки и их тип. Листинг содержит последовательность строк, отображающих коды команд и соответствующие им мнемоники. С помощью функций меню оболочки, или, нажав клавишу Esc, выйдите из режима просмотра. Для коррекции текста программы (файл с расширением .asm) и ее последующего ассемблирования повторяйте действия 3–6 до тех пор, пока не будут получен объектный модуль программы с расширением .obj.

При этом для примера 3.4 коды команд в листинге должны полностью соответствовать последовательности кодов, полученных путем их ручного кодирования (см. табл. 2.2) для решения одной и той же задачи (2.1), что говорит о правильности их получения, а также показывает работу ассемблера по принципу «один к одному».

Создание исполняемого модуля

7. Вызовите компоновщик (редактор связей), для чего введите

tlink.exe /t/x <имя файла>.obj ↵

С помощью опции t компоновщику задано сформировать com-файл, а x – не генерировать map-файл исполняемой программы. После выполнения tlink будет создан исполняемый com-модуль, который может быть использован далее для его выполнения (или отладки).

Выполнение и отладка программы

8. С помощью команды

td.exe <имя файла>.com↵

запустите отладчик для работы с созданным com-файлом программы. Отладчик загрузит в память исполняемый модуль с адреса 100h, причем коды программы и данных разместятся в одном сегменте кода емкостью 64 Кбайт. После загрузки отладчик выдаст на экран монитора окно процессора CPU.

9. Нажав ENTER, снимите марку отладчика. На экране изобразится окно отладчика CPU, состоящее из пяти подокон: кодового сегмента, содержащего коды команд программы и их мнемоники; регистров микропроцессора; регистров флажков; сегмента стека; сегмента данных. Переход из одного подокна в другое осуществляется нажатием клавиши Tab или щелчком левой кнопки мышки.

Находясь в подокне, можно, нажав Alt–F10, войти в локальное подменю. С помощью его команд можно изменить содержимое регистров МП или памяти. Нажав F10, можно войти в главное меню отладчика и воспользоваться его командами для управления выполнением программы. Выход из меню производится клавишей Esc.

Ниже рассматриваются основные команды отладчика для отладки программ с помощью окна процессора CPU.

10. *Запись в регистр*. Нажав клавишу Tab, перейдите в подокно регистров. Если вы вводите данные в регистр DX, подведите маркер к регистру DX, введите с клавиатуры код 30h и нажмите ENTER. Тем самым вы ввели в DX операнд N = 0030h. При вводе 16-ричных кодов необходимо, чтобы первый символ начинался с цифры, например 0FFFFh.

11. *Запись в память*. Перейдите в подокно сегмента данных. Нажав Alt-F10, вызовите для данного подокна локальное меню. Выберите в нем команду GOTO и нажмите ENTER. Далее введите с клавиатуры начальный адрес данных ds:0500h↵.

Если исходная программа работает с данными в формате слова, то с помощью клавиш Ctrl–D войдите в подменю и задайте в нем формат WORD (слово). В подокне после адреса ds:0500h вы увидите значения операндов в ДК – они оказались в сегменте данных памяти (DS) в результате работы ассемблера и других системных программ.

Модификация содержимого памяти. Маркером выделите нужную ячейку памяти и введите с клавиатуры необходимый операнд в ДК, например, 0004, по адресу ds:0500. Затем переместите маркер на следующую ячейку и введите в нее значение следующего операнда, например, код 0FFFF, отражающий число –1.

12. Перейдите в подокно кодов (программы). В нем изображаются дизассемблированные команды выполняемой программы, причем текущая команда помечается стрелкой. Для управления выполнением программы используют следующие основные команды, вызываемые активными клавишами:

F7	Выполнение одной команды
F8	Выполнение одной команды с пропуском вызовов
F9	Запуск программы в автоматическом режиме
F4	Выполнение команд до точки останова
Ctrl–F2	Установка программы в исходное состояние
F2	Установка/отмена точки останова

Нажимая последовательно F7, *выполните программу по шагам (командам)*. При этом следите за содержимым регистров МП и памяти. Фиксируя их в виде *трассы программы*, отражающей содержимое изменяющихся регистров МП и памяти, убедитесь в правильности полученного результата. По своей форме трасса схожа с таблицей 2.2, если в ней изобразить колонки, показывающие содержимое регистров AX, DX и ячейки [0504h]. Если в программе выявлены логические ошибки (например, пропущены некоторые команды), из-за наличия которых она не формирует правильного результата, выйдите из отладчика и снова повторите пункты 3–12.

13. Введя Alt–X, выйдите из отладчика.

Рассмотренные приемы отладки программ в окне CPU применяются в основном для отладки несложных программ. Наряду с этим имеется возможность отладки программ в специальном модуле (среде), схожем с ее исходным текстом. Работа в этой среде будет описана для рассматриваемого далее варианта программы, исполняемый модуль которой должен обязательно иметь расширение .exe.

3.4. Разработка программ на языке ассемблера для генерирования исполняемого exe-модуля

Порядок их разработки показан на примере сложения многобайтовых двоичных чисел со знаком, размещенных в памяти:

$$X \leftarrow X + Y \ . \qquad (3.1)$$

Пусть длина m этих чисел составляет 6 байт, то есть m = 6.

Порядок разработки программы

1. Формируются математические зависимости, показывающие процесс вычисления и используемые при этом операции МП, к которым сводится решение задачи (3.1).

С этой целью на конкретном примере выполнения операции 3.1 устанавливается, что сложение X + Y осуществляется микропроцессором последовательно во времени путем сложения байт x_i и y_i исходных многобайтовых чисел X и Y с учетом переноса c_i, начиная с младших байтов слагаемых:

$$x_i \leftarrow x_i + y_i + c_i \ , \qquad (3.2)$$

где i = 1, 2, …, 6; 1 – номер младшего байта слагаемых.

2. Составляется регистровая (программная) модель (рис. 3.3) выполняемой операции 3.2.

В этой модели указываются:

– ячейки памяти, в которых в ДК хранятся числа X и Y в виде последовательности байт x_i и y_i соответственно (i = 1, 2, …, 6);

– регистры МП, участвующие в выполнении операции 3.2:

- SI и DI – указатели памяти для формирования адресов байт слагаемых X и Y с использованием косвенной регистровой адресации (см. п. 2.3.2). Предварительно в SI и DI загружают начальные адреса этих чисел, при этом i = 1;
- CX – счетчик цикла, служащий для хранения числа j = m – i байт чисел X и Y, которые предстоит обработать при последовательном вычислении (3.2). Предварительно в CX устанавливают число байт m = 6 слагаемых, то есть j = m;
- AL – аккумулятор, необходимый для последовательного суммирования байт слагаемых;
- CF – триггер для учета переноса c_i, необходимого при выполнении операции 3.2. В начале CF сбрасывают в «0». При сложении $x_i + y_i + c_i$ в CF фиксируется перенос c_{i+1}.

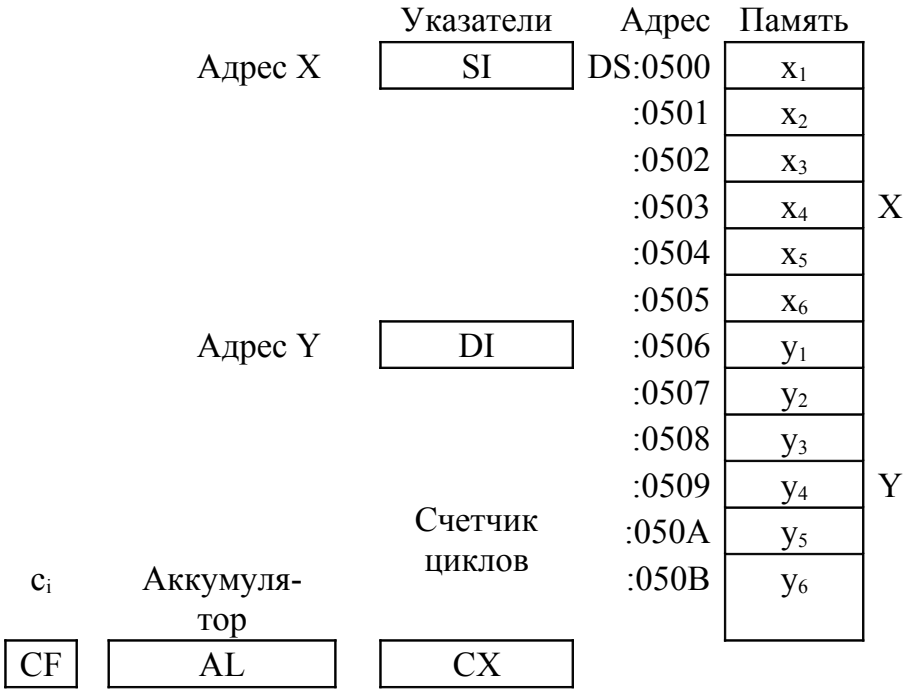


Рис. 3.3. Регистровая модель выполняемой операции (3.2)

3. Разрабатывается схема алгоритма (рис. 3.4) вычисления выражения 3.2.

Операторы этой схемы содержат простые операции, к которым сводится вычисление выражения. Справа от каждого оператора указываются 1–2 команды, посредством которых он реализуется. Следует отметить, что для организации в программе цикла целесообразно применять команду LOOP, действие которой совмещает операции декремента регистра CX (команда DEC CX) и условного перехода по не нулю (команда JNZ). Их применение в программах рассмотрено ранее (см. п. 2.7, *команды условного перехода).*

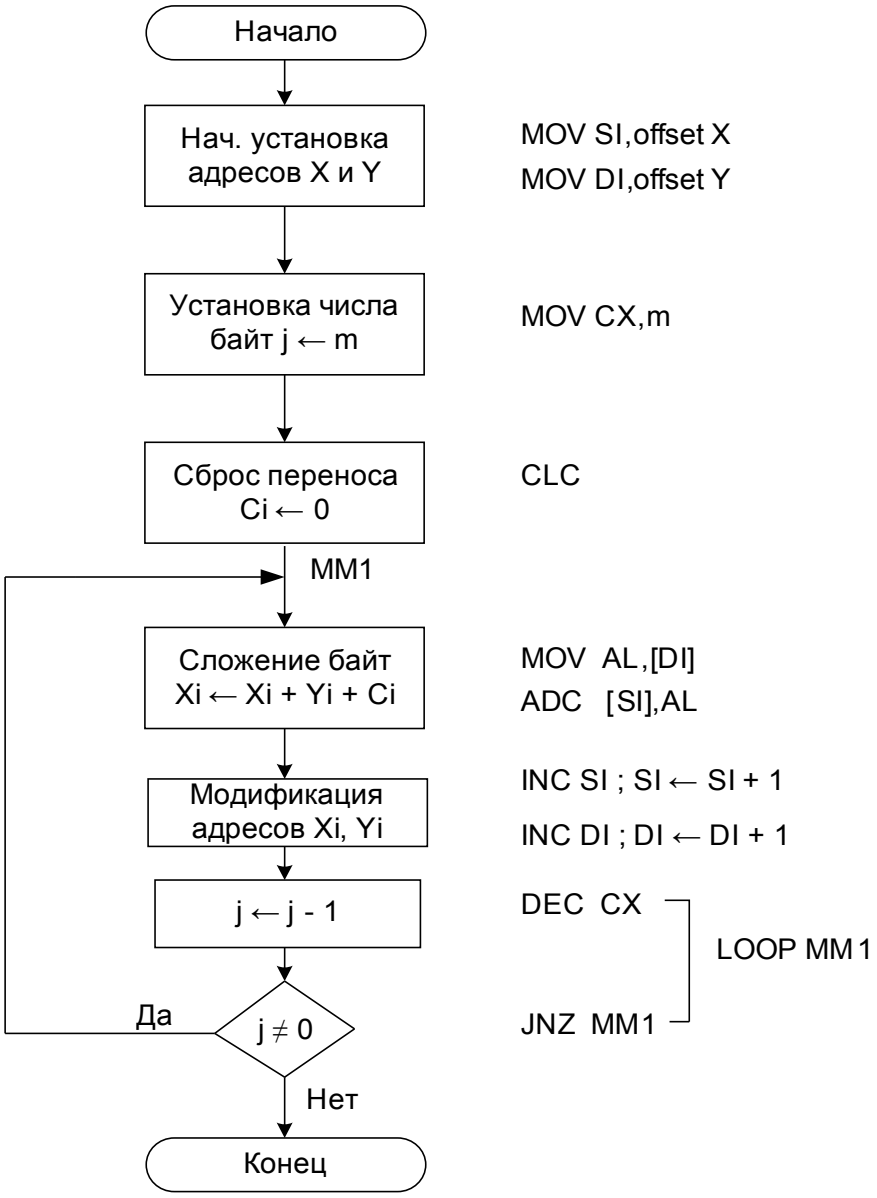


Рис. 3.4. Схема алгоритма вычисления выражения 3.2

4. Разрабатывается программа (пример 3.6) на языке ассемблера для генерирования исполняемого модуля типа .exe.

Исходный модуль представляет собой последовательность операторов (или предложений) языка ассемблера. Программа оформлена в виде процедуры PROC с именем SUM_XY, завершаемой директивой SUM_XY ENDP.

Пример 3.6

```
;Программа сложения X ← X + У
;Адреса X и Y формируются с помощью регистров SI и DI,
;Число байт m = 6 устанавливается в счетчике цикла CX
        .MODEL   small        ;Ближние модели программ и данных
        .STACK   64           ;Размер стека 64 байта
        .DATA                 ;Начало сегмента данных
        org   500h            ;Размещение данных с адреса 500h
X       db    0ffh,60h,90h,0feh,20h,00h    ;X = 0020FE9060FFh
Y       db    06h,05h,04h,03h,02h,01h      ;Y = 010203040506h
m       equ   6               ;Число байт m в X и Y принять равным 6
        .CODE                 ;Начало сегмента кодов
SUM_XY  PROC                  ;Процедура с именем SUM_XY
        mov   ax,@data        ;Указывает регистр DS
        mov   ds,ax           ;на сегмент данных
;Команды программы
;Установка в SI и DI начальных адресов слагаемых X и Y, то есть i←0
        mov   si,offset X
        mov   di,offset Y
; Установка в CX числа байт m (т.е j ← 6) и сброс переноса ci,
        mov   cx,m            ;CX ← m
        clc                   ;CF ← 0
;Сложение байт слагаемых xi ← xi + yi + ci
mm1:    mov   al,[di]         ;AL ← yi
        adc   [si],al         ;Сложение с переносом [SI] ← [SI] + AL + ci
;Модификация адресов байт слагаемых X и Y, то есть i ← i + 1
        inc   si      ;SI ← SI + 1
        inc   di      ;DI ← DI + 1
;Перейти к вычислению следующей суммы байт (j ← j – 1)
        loop  mm1     ; CX ← CX – 1. Если CX ≠ 0, то переход к MM1
        mov   ah,4ch  ;Функция DOS
        int   21h     ;выхода из программы
SUM_XY  ENDP          ;Конец процедуры SUM_XY
        END SUM_XY    ;Конец программы
```

Директива .MODEL small задает в программе модель памяти, при которой программа и данные размещаются в отдельных сегментах. Псевдооператоры (директивы) .STACK, .DATA и .CODE делят программу на сегменты стека, данных и команд.

В сегменте данных с адреса 500h с помощью директив db определены, начиная с младшего байта, значения байтов 6-байтовых переменных X = 0020FE9060FFh и Y = 010203040506h в ДК.

Сегмент команд содержит три группы команд:

1) для установки регистра DS на сегмент данных;
2) для вычисления операции (3.2);
3) для поддержки функций DOS выхода из программы.

5. Производится отладка программы.

Далее, в п. 3.5, приводится порядок ввода и проверки работоспособности программы с формированием в среде отладчика td.exe комментариев в исходной программе.

Рекомендации по разработке программ (для приложения 4)

Общие рекомендации. Разработка программ производиться в соответствии с рассмотренным нами порядком.

Программа, приведенная в примере 3.6, может быть принята за основу разрабатываемой. Основные принципы задания массивов даны в примере 3.2, а размера их элементов – в примере 3.1.

При выполнении операций над элементами x_i (числами со знаком +1, –2, +5 или символами ′1′, ′2′,′5′, BCD-числами и т.д.) одного массива для косвенной регистровой адресации элементов x_i применяется только регистр-указатель SI. Причем при модификации адреса элемента необходимо учитывать его размер: байт или слово, так как от него зависит на +1 или на +2 необходимо изменять адрес указателя памяти. Примеры команд с различными режимами адресации даны в п. 2.3.

Переменную *y*, как результат суммирования или как число искомых элементов для задач поиска, необходимо с помощью директивы определения данных DW или DB задать в сегменте данных после определения массива элементов.

В зависимости от варианта задания, ниже приведены основные рекомендации и фрагменты программ, поддерживающие их разработку с использованием команд с косвенной регистровой адресацией и команд циклов и условных переходов.

1. При программировании передачи массива данных из одной области памяти в другую операцию сложения в программе (пример 3.6) следует заменить на операцию пересылки, естественно, необходимо при этом учитывать размер передаваемых данных (AX – слово, AL – байт).

2. При программировании передачи массива данных из одной области памяти в другую с одновременным выполнением некоторой операции, например X ← X + 1, следует после выборки операнда из памяти в аккумулятор выполнить команду его инкремента.

3. При сложении BCD-чисел необходимо текущий результат двоичного суммирования формировать в аккумуляторе AL, а затем производить его коррекцию с помощью команды DAA. После получения итоговой суммы значение *y* необходимо передать в память.

Пример 3.7

Фрагмент программы сложения BCD-чисел x_i, когда их количество m задано в регистре CX, а адрес массива элементов x_i – в регистре SI. Пусть предварительно в аккумуляторе AL, где хранится текущая сумма s_i, установлено ее нулевое значение.

```
;Сложение  BCD-чисел si ← si + xi
mm2: add   al,[si]      ;AL ← si + xi
     daa                ;Десятичная коррекция аккумулятора
;Модификация адреса слагаемого  xi
     inc   si           ;SI ← SI + 1
;Перейти к вычислению следующей суммы
     loop  mm2          ; CX ← CX – 1. Если CX ≠ 0, то переход к MM2
     mov   y,al         ;Передать сумму y в память
```

4. Для подсчета числа выявляемых элементов, например количества *y* отрицательных чисел x_i в массиве, в состав регистровой модели необходимо включить регистр BX, служащий для хранения текущего количества k выявленных при просмотре массива значений $x_i < 0$.

Пример 3.8

Фрагмент программы определения количества *y* отрицательных двухбайтных чисел x_i, когда размер их массива m задан в регистре CX, а начальный адрес массива x_i – в регистре SI. Пусть предварительно в регистре BX, в котором формируется k (текущее значение *y*), установлен 0.

```
mm3:    mov  ax,[si]     ;AX ← xi
        cmp  ax,0  ;Сравнить xi с нулем, то есть найти (xi – 0)
        jns  m4    ;Переход  к m4 , если xi положительное 0 ≤ xi
        inc  bx    ;Найдено еще одно отрицат. число xi (k ← k – 1)
;Модификация адреса двухбайтового числа  xi
m4:     inc  si    ;SI ← SI + 1
    inc  si    ;SI ← SI + 1
    loop mm3 ; CX ← CX – 1. Если CX ≠ 0, то переход к MM3
```

mov *y*,bx ;Передать полученное значение *y* в память

Определение числа ASCII-символов, например, 35H, можно производить с помощью последовательности команд CMP AL,35H и условного перехода по не нулю JNZ (см. пример 2.2).

3.5. Порядок создания и отладки исполняемого exe-модуля с символьной информацией для отладчика

Создание исходного файла программы

1. С помощью команд редактора оболочки (см. действия 1–3 п. 3.3.3) создайте исходный файл (с расширением .asm) программы на языке ассемблера, приведенной в примере 3.6.

Ассемблирование

2. В командной строке оболочки введите команду

tasm /zi <имя файла>.asm,, ↵

Создание исполняемого модуля

3. При наличии obj-модуля введите команду

tlink /v/x <имя файла>.obj↵

При этом программой tlink будет сгенерирован исполняемый exe-модуль с символьной информацией для отладчика.

Выполнение и отладка программы

4. Для выполнения программы в отладчике вызовите команду

td <имя файла>.exe↵

После недолгой загрузки система перейдет в отладчик TD. На экране изобразятся два окна отладки: окно модуля Module с текстом программы и ниже – окно просмотра Watches. Далее приводится порядок отладки программы в этой среде.

5. С помощью команд главного меню View; Dump↵ и View; Registers↵ выведите на экран окна памяти и регистров. Мышкой сместите их на края модуля отладки, предварительно уменьшив его размеры. Порядок отображения и модификацию данных в окнах отладчика дан в пунктах 10 и 11 п. 3.3.3.

6. Последовательно, нажимая клавишу F7, выполните первые две команды программы: mov ax,@data и mov ds,ax. После чего перейдите в окно памяти и установите в нем начальный адрес данных: Alt–F10, GOTO, ds:500h [ENTER]. *Задание* ds: *обязательно*.

6. Выполните далее программу по шагам, нажимая последовательно F7 или F8. При выполнении, например, программы, данной в примере 3.6, в окне памяти, начиная с адреса ds:500h, будут последовательно отображаться байты суммы, а в окне регистров – значения регистров

AL, CX, SI и DI. При этом содержимое AL представляется в виде двух младших цифр регистра AX.

Клавишами Ctrl–F2 можно перезагрузить программу и выйти на начало ее выполнения, отмечаемое в модуле отладки стрелкой.

7. Нажав Alt–X, выйдите из отладчика.

3.6. Особенности разработки и отладки 32-разрядных прикладных программ на базе МП 80x86

3.6.1. Использование средств 32-разрядных МП 80x86 для разработки программ для реального режима

Все 32-разрядные микропроцессоры (i386, i486, различные модели Pentium) имеют режимы: реального (R-режима) и защищенного (P-режима) адреса. Только в P-режиме эти МП выполняют 32-битовые программы с использованием таких их возможностей, как:

1) адресное пространство памяти 4 Гбайт;

2) использование 32-битовой адресации команд;

3) новые привилегированные команды для P-режима;

4) применение 32-битовых регистров (EAX, EBX, ECX и т.д.);

5) использование масштабированной индексной адресации, например: MOV EAX, [EBX+ESI*2+200h] – базовая индексная с масштабированием и смещением (может содержать до 15 байт);

6) использование режимов 32-битовой адресации данных, включая и новые, например, варианты индексной адресации с масштабированием;

7) четыре сегмента данных вместо двух.

Однако есть возможность использовать преимущества, указанные в п. 4–7, если МП работает и в реальном режиме. Единственный минус – все эти программы для R-режима работают только в сегментах емкостью 64 Кбайт и в памяти 1 Мбайт. Это объясняется тем, что старшие биты A31-A16 эффективного адреса, например, [EBX] или [ESI], не изменяются.

Если используются 32-битовые адреса и операнды, то команда (рис. 3.9) содержит дополнительные байты атрибутов: 66h – размер операнда 32 бита, 67h – размер адреса 32 бита.

Префикс команды REP 0–1 байт	Префикс размера адреса 0–1 байт	Префикс размера операнда 0–1 байт	Префикс замены сегмента 0–1 байт
	67h	66h	

КОП 1 или 2 Байта	Mod reg r/m 0 или 1 байт	Масштаб 0 или 1 Байт	Смещение в команде 0–2, 4 байта	Непосредств. операнд 0–2, 4 байта

Коэфф. 1, 2, 4, 8

Рис. 3.9. Формат команды МП i80x86 для 32-разрядных программ

По умолчанию ассемблер генерирует коды для 16-разрядного МП, вследствие этого необходимо в программе указать тип МП-32, например **.486** или **.586** для МП Pentium. При создании 32-разрядных программ для работы в защищенном режиме, например, под Windows, целесообразно разрешить применение в них привилегированных команд МП 80x86. С этой целью программе указывается, например, директива **. 586P** или **.486P**.

На рисунке 3.10 приведена схема формирования эффективного адреса EEA при 32-битовой адресации для команды MOV EAX, [EBX+ESI*2+200h].

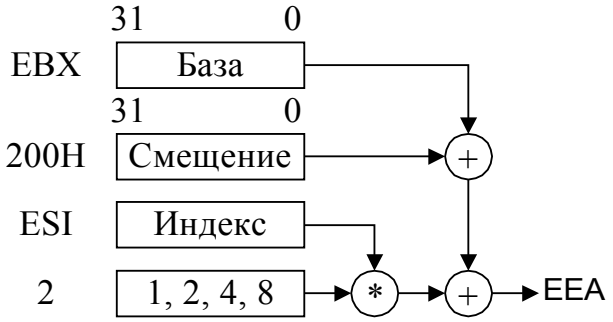


Рис. 3.10. Схема формирования адреса EEA = EBX + ESI*2 + 200h

Далее приведен пример 3.11 программы для R-режима с использованием 32-битовых регистров и режимов адресации данных. Для задания сегментов в ней использованы стандартные директивы **SEGMENT –** начало и **ENDS –** конец сегмента. Параметр use16 при описании сегментов ограничивает их размеры до 64 Кбайт.

В программе применяются индексная с масштабированием адресация данных (см. рис. 3.6) и стандартные директивы ассемблера для определения сегментов.

3.6.2. Порядок создания и отладки exe-модуля 32-разрядной программы с отладочной информацией

Формирование exe-модуля 32-разрядной программы

1. Для созданного в редакторе исходного файла с расширением .asm введите следующую последовательность команд, последовательно запускающих выполнение соответствующих системных программ:

tasm /zi <имя файла>.asm,, ↵
tlink /3/v <имя файла>.obj↵
td <имя файла>.exe↵

Пример 3.11

;Программа вычисления выражения M = K – R + N + 120h
;с формированием exe-модуля, когда данные – 32-разрядные слова
;K, R, N и M размещены в памяти с адреса 500h

```
            .486
stack1      segment     stack use16‘stack’
db          64 dup(0)   ;Стек 64 байта
stack1      ends
dseg  segment     para  use16‘data’
      org   500h        ;Начальный адрес данных
k     dd    +60h        ;Переменные k
r     dd    -10h        ;r
n     dd    +30h        ;n
m     dd    0           ;Результат m
dseg  ends
cseg  segment     para  use16‘code’
      assume      cs:cseg, ds:dseg, ss:stack1
begin proc  far                 ;Дальняя процедура
      push  ds
      sub   eax,eax
      push  ax
      mov   ax,dseg             ;Указание регистру ds
      mov   ds,ax               ;на сегмент данных
;Команды программы
      mov   ebx,offset k              ;ebx ← адрес операнда k
      mov   esi,4
      mov   eax,[ebx]                 ;eax ← адрес операнда k
      sub   eax,[ebx+esi*1]           ;eax ← eax – r
      add   eax,[ebx+esi*2]           ;eax ← eax + n
```

```
        add   eax,120h              ;eax ← eax +120h
        mov   [ebx+esi*4],eax       ;m ← eax
        mov   ah,4ch
        int   21h
begin endp
cseg  ends
        end   begin
```

Порядок работы в отладчике td.exe

2. Выведите на экран окно регистров:

VIEW, REGISTER [ENTER]

3. Находясь в окне регистров, вызовите локальное меню (Alt–F10) и установите длину регистров – 32 бита:

REGISTER 32 bit | YES |

4. Выведите на экран окно памяти:

VIEW, DUMP [ENTER]

Перейдите в окно памяти и становите в нем длину данных – двойное слово:

Ctrl-D, LONG [ENTER]

5. Последовательно, нажимая F8, выполните все команды до mov ds,ax (включительно).

6. В окне данных установите начальный адрес данных:

Alt–F10, GOTO, ds:500h [ENTER]

7. Выполните далее программу по шагам (F8), наблюдая за результатами ее работы.

8. Просмотрите кодировку программы в окне CPU: VIEW, CPU [ENTER]

9. Выйдите из отладчика.

3.7. Особенности программирования на ассемблере с использованием процедур и макроопределений

На примере 3.12 рассмотрена программа выдачи сообщений на экран, которая оформлена в виде процедуры PROC с именем OUR_PROG. В сегменте данных этой программы оператор TEXT1 определяет текст сообщения, выводимого на экран, а SIMV – последовательность звездочек (*), выделяющую этот текст. Порядок отладки программы по выводу заданного текста (приложение 7) дан в п. 3.5.

В процедуре VDISP используется команда прерывания DOS INT 21H, с помощью которой производится обращение к функции для работы с дисплеем. Адрес типа функции задается значением регистра AH = 9. Знак $ в тексте определяет конец изображаемой строки, начальный адрес которой задается в регистре DX командой MOV DX,OFFSET TEXT1. Коды 0Dh и 0Ah обеспечивают перевод маркера в начало следующей строки (0D – возврат каретки, 0A – переход на следующую строку). В основной программе процедура вызывается командой вызова CALL VDISP.

Пример 3.12

Программа вывода текста на экран.

```
;Начало программы
     .MODEL     small ;Ближние модели программы и данных
     .STACK     64    ;Размер стека 64 байта
     .DATA            ;Начало сегмента данных
;Тексты сообщений
text1 db    'Иванова И.А.',0dh,0ah
      db    'Петрова Н.В.',0dh,0ah,'$'
simv  db    '**********', 0dh,0ah,'$'
     .CODE            ;Начало кодового сегмента
our_prog   proc       ;Процедура с именем OUR_PROG
     mov   ax,@data   ;Указывает DS на
     mov   ds,ax      ;сегмент данных
;Команды программы
     mov   dx,offset text1    ;Адрес строки TEXT1 в DX
     call  vdisp              ;Вызов процедуры вывода строки
     mov   dx,offset simv     ;Адрес строки SIMV в DX
     call  vdisp              ;Вызов  процедуры вывода строки
     mov   ah,4ch             ;Функция DOS
     int   21h                ;выхода из программы
our_prog   endp               ;Конец процедуры
;Процедура  вывода строки на экран
vdisp proc
      push  ax
      mov   ah,9              ;Функция DOS
      int   21h               ;вывода строки на экран
      pop   ax
      ret
vdisp endp
END our_prog                  ;Конец программы
```

Операциию вывода текста на экран можно также представить в виде макроопределения VDISP MACRO.

```
vdisp   macro
        push    ax
        mov     ah,9    ;Функция DOS
        int  21h        ;вывода строки на экран
        pop ax
        endm
```

Приведенная последовательность команда вызывается из программы макрокомандой VDISP (вместо CALL VDISP). Исходный тест макроопределения приводится в начале программы.

3.8. Программирование обработки цепочек данных

Цепочечные команды

Под цепочкой понимается последовательность байт или слов, находящихся в смежных ячейках памяти. Такую связанную структуру данных образует, например, последовательность вводимых с терминала символов.

В МП 80x86 имеется пять базовых однобайтовых цепочечных команд-примитивов, предназначенных для обработки одного элемента цепочки. Если элементом является байт, то в мнемонике команд указывается буква B, если слово, то – W, например: MOVSB и MOVSW. В общем случае при обработке строк данных:

– цепочка-приемник находится в дополнительном сегменте данных (ES) и ее элемент адресуется регистром DI,

– цепочка-источник находится в сегменте данных (DS) и ее элемент адресуется регистром SI.

Задание начальных адресов цепочек, например приемника STRING1 и источника STRING2, производится командами LEA DI,ES:STRING1 и LEA SI,STRING2. Однако обе цепочки можно разместить в сегменте данных, если перед их обработкой сделать равными значения сегментных регистров ES и DS.

В зависимости от состояния признака направления DF регистра F после каждой операции над элементом цепочки выполняется инкремент (DF = 0) или декремент (DF = 1) указателей DI и SI (при обработке байт – на единицу, при обработке слов – на два).

Цепочечной команде может предшествовать специальный префикс повторения REP, который заставит МП повторить команду и модифицировать $CX \leftarrow CX + 1$. Операции повторяются, пока $CX \neq 0$. Перед командами сравнения CMPS и SCAS мнемоника префикса имеет

вид REPZ/REPE или REPNZ/REPNE. Для REPZ примитив повторяется, пока CX ≠ 0 или признак ZF = 1, а для REPNZ – пока CX ≠ 0 или ZF = 0.

Программы, приведенные в примерах 3.13 и 3.14, выполняют одно и то же действие – сравнивают строки пока не будут просмотрены 100 элементов или пока не встретятся совпадающие элементы.

Пример 3.13

```
        ;Найти совпадения
mm1: cmpsb      ;Сравнить [SI] – [DI], SI ← SI + 1, DI ← DI + 1
     dec  cx    ; CX ← CX – 1
     jnz  mm1   ;Повторять, пока CX ≠ 0 или FZ = 0
```

Пример 3.14

```
        ;Найти совпадения.
repnz       cmpsb       ;Те же операции, что и в примере 3.13.
```

Используя в примере 3.13 вместо JNZ другие команды условного перехода, можно сравнивать элементы и по другим признакам, например по признаку «больше» или «меньше».

Далее, в примере 3.15, показана процедура копирования 10 элементов из строки STRING2 в строку STRING1.

Пример 3.15

```
        ;Копирование элементов строки
        ;Пусть цепочки находятся в сегменте данных
string2  db  ‘Петрова Н., Иванов А.’ ;Цепочка-источник
string1  db 20 dup(?)                ;Цепочка-приемник
        ;Процедура копирования
copir proc
        push  ds            ;Заставить  указывать es
        pop   es            ;на сегмент данных
        cld                 ;Сброс  для обработки слева направо
        lea   si,string2    ;Адрес строки STRING2 в SI
        lea   di,string1    ;Адрес строки STRING1 в DI
        mov   cx,20         ;Число элементов в CX
rep     movsb               ;Скопировать байты
        ret
copir endp
```

Ввод строковых данных в ПК

В программе значения элементов в строке можно задать с помощью директивы DB или DW.

При вводе строки с клавиатуры целесообразно пользоваться функцией DOS 0AH, вызываемой командой INT 21H. Чтобы воспользоваться ею, необходимо зарезервировать в сегменте данных место для строки. Например, оператор

```
string  db    64,65 dup(?)
```

резервирует место для строки STRING, содержащей 64 элемента.

Чтение строки с клавиатуры в память без изображения на экране производится командами, приведенными в примере 3.16.

Пример 3.16

```
;Чтение строки с клавиатуры
lea    dx,string    ;Сделать  DX указателем буфера
mov  ah,0ah         ;Прочитать строку,
int    21h          ;ввод завершить нажатием [Enter]
```

После их выполнения в первом байте буфера STRING находится длина буфера 64, а во втором – число фактически введенных символов. Элементы строки STRING+2 размещаются с третьего байта.

Порядок программирования и отладки программ обработки цепочек

1. Изучите форматы цепочечных команд и примеры программ обработки строк (пп. 3.8.1, 3.8.2).
2. Разработайте для заданного варианта программу обработки строк (см. приложение 5). Для более наглядного выполнения программы используйте процедуры ввода и изображения строк, приведенные в примере 3.16.
3. С помощью системных программ (см. п. 3.5) сформируйте исходный, объектный и исполняемый модули.
4. Выполните программу в отладчике TD.
5. Выполните программу.

3.9. Встроенное ассемблирование

В общем случае при разработке пользовательских программ используют принципы встроенного ассемблирования и вызовы функций на языке ассемблера из языка высокого уровня (ЯВУ).

Их организация дается на примере взаимодействия ЯВУ «Турбо-Си» с «Турбо-ассемблером».

Встроенное ассемблирование на Турбо-Си

Язык программирования Турбо-Си имеет в своем составе встроенный ассемблер. Это позволяет включать в программу на Си отдельные команды языка ассемблера. При этом включаемая команда должна начинаться с директивы Си – **asm.**

Пример программы на Си со встроенным ассемблированием:

```
int i;
. . .
i=0;                    /*установка i = 0 на СИ*/
asm dec word ptr  i;    /*i = i – 1 на ассемблере*/
i++;                    /*i = i + 1 на Си*/
```

Эти три команды будут выглядеть в отладчике после создания exe-модуля следующим образом:

```
mov  word ptr    [bp-02],0000
dec  word ptr    [bp-02]
inc  word ptr    [bp-02]
```

При трансляции Си-программ со встроенным ассемблером необходимо применять только спецкомпилятор tcc.exe, который поддерживает средства встроенного ассемблера.

Ограничения на встроенное ассемблирование

1. Командами перехода встроенного ассемблера можно ссылаться только на метки Си. Типичная ошибка:

```
asm  JZ  MM1;
asm  DEC  CX;
asm  MM1:  ADD  AX,10;
```

Правильно:

```
asm  JZ  MM1;
       . . .
MM1:  x = x+a;
```

2. Команды встроенного ассемблера, за исключением команд перехода, могут иметь любые операнды, кроме меток Си.

3. Необходимо сохранять регистры BP, SP, CS, DS, SS в командах встроенного ассемблера перед командами, использующими эти регистры.

Недостатком встроенного ассемблирования является то, что программа содержит избыточное число байт. Кроме того, все операнды хранятся в памяти, поэтому операции над памятью производятся значительно медленнее, чем над регистрами МП, особенно для команд с базовой адресацией с использованием регистра BP.

В связи с этим при предъявлении повышенных требований к объему и быстродействию отдельных процедур исполняемой программы их следует разрабатывать на языке ассемблера, предоставляющем разработчику возможность оптимально размещать обрабатываемые данные как в регистрах микропроцессора, так и в памяти.

Вызов функции на языке ассемблера из Си

Для организации вызовов отдельно формируются основной модуль на Си и модуль на ассемблере как вызываемая функция. Исполняемый exe-файл создается программой tcc путем объединения двух исходных файлов:

tcc < имя файла на Си>.c < имя файла на ассемблере>.asm↵

При компоновке необходимо соблюдать соглашения об именах переменных и меток, передаваемых от одного модуля к другому. Это делается с помощью директив **PUBLIC** и **EXTRN** в модуле на ассемблере и директивы **extern** в модуле на Си. Кроме того, должны быть соблюдены одинаковые типы моделей памяти в Си и ассемблере. Это достигается с помощью упрощенных сегментных директив.

Имена общих переменных в модуле на asm должны начинаться со знака подчеркивания: _x, _summa и т.д.

В примере 3.15 приведена программа на Си и ассемблере. Программа на Си вызывает с помощью функции summa() процедуру, которая реализована на ассемблере и производит расчет выражения Z = X + Y, где X, Y, Z – переменные, представленные в виде 16-ричных чисел. Процедура _summa позволяет складывать только двоичные числа в формате слова, поэтому в Си с помощью оператора scanf их вводят в 16-ричной системе счисления.

Пример 3.15

Программа на Си и ассемблере

```
/* Программа на Си (основной модуль) */
#include<stdio.h>
extern int summa(); /* переменная summa определена */
```

```
                    /* в другом модуле */
int X,Y; /* переменные X, Y доступны другим модулям */
extern int z; /* переменная z определена в другом модуле */
main ()
  {
    printf ("\n Введите целое 16-ричное число X = ");
    scanf ("%X, &X");
    printf ("\n Введите целое 16-ричное число Y = ");
    scanf ("%X, &Y");
    summa(); /* вызов процедуры _summa из asm-модуля */
    printf ("\n Сумма Z = X + Y = % X, Z");
  }
;Процедура на ассемблере, реализующая операцию  Z = X + Y
.MODEL    small
.DATA
extrn  _x:word,  _y:word   ;Эти переменные определены
                           ;в модуле на Си
public  _z                 ;Переменная z доступна другим
                           ; модулям
_z       dw   0            ;Переменная z определена в asm-модуле
         .CODE
public  _summa             ;Является глобальной
_summa  proc               ;Процедура _sumtma
          mov      ax,_x
          add      ax,_y
          mov      _z,ax
          ret
_summa  endp
          end
```

Между типами данных в Си и ассемблере существует следующее соответствие:

Типы данных в Си	Типы данных в ассемблере
char int и т.д.	byte (db) word (dw)

Тип переменной z можно определить в модуле на Си директивой int z. Тогда в asm-модуле отсутствовала бы строка:

```
_z     dw    0
```

При вызове ассемблерного модуля необходимо учитывать следующие особенности.

1. Вызываемые ассемблерные модули не должны менять регистры BP, SP, DS, SS. Регистры AX, BX, CX, DX и флаги могут меняться любым образом. Рекомендуется сохранять SI и DI, так как Турбо-Си иногда их использует.

2. 8- и 16-битовые значения результата могут возвращаться из asm-модуля через регистр AX, 32-битовые – через DX и AX.

3. Передача параметров производится через указатель BP.

ПРИЛОЖЕНИЯ

Приложение 1

Варианты заданий к практическим занятиям по представлению данных и выполнению арифметико-логических операций в ЭВМ

1. Переведите заданные в таблице числа X и Y в двоичную и 16-ричную систему счисления (СС). Перевод в двоичную систему СС осуществляйте через 16-ричную СС.

Таблица

Варианты задания чисел X и Y

№ вар.	X	Y	№ вар.	X	Y
1	+21	–66	11	–83	–12
2	–17	+28	12	–29	+21
3	+57	–69	13	–26	–57
4	–84	–13	14	+48	–18
5	+28	–37	15	–6	+33
6	–51	+35	16	+28	–42
7	+25	–56	17	–68	–14
8	–13	+49	18	+44	–25
9	+39	–41	19	–37	+33
10	–27	+21	20	+53	–21

2. Найдите двоично-кодированное десятичное представление модулей чисел X и Y.

3. Представьте полученные в п. 1 16-ричные числа в ДК в формате байта, слова и двойного слова и покажите их размещение в памяти, начиная с адреса 500 + 2N, где N – номер варианта.

4. Сложите числа X и Y в ДК по правилам двоичной и 16-ричной СС в формате байта, найдите признаки результата и выполнить его проверку.

5. Вычьте X – Y в ДК в формате байта, найдите признаки результата и выполнить его проверку.

6. Найдите физический адрес команды $ФА_{ком}$ для заданных значений сегментного регистра CS = 04FFH + 2N и указателя команд IP = 02E0H + 2N, где N – номер варианта в 16-ричной СС. Таблица сложения 16-ричных цифр (символов) приведена в приложении 6.

Приложение 2

Варианты заданий к практическим занятиям и лабораторным работам по разработке линейных программ

№ варианта задания	Размещение операндов				Операнд	Начальный адрес программы
	M	K	R	N		
1. M = K + R + 2N – 3	Память			BH	Байт	100
2. M = K – R – N + 50	Память		DX	Пам	Слово	100
3. M = 2K – N + R – 6	Пам	CL	Память		Байт	100
4. M = K + 3N – R + 20	Память			DX	Слово	100
5. M = K – R – N + 8	Память		CL	Пам	Байт	100
6. M = 2K – R + N + 18	Пам	DX	Память		Слово	100
7. M = 2K – 2R + N –10	Память		DL	Пам	Байт	100
8. M = 120 – K – R + N	Пам	CX	Память		Слово	100
9. M = 20 + K – 2R + N	Память			CH	Байт	100
10. M = 50 + K – 2R + N	Пам	BX	Память		Слово	100
11. M = R – K + 2N + 30	Память			BH	Байт	100
12. M = R – K – N + 20	Память		DX	Пам	Слово	100
13. M = 30 + K + 2R – N	Пам	CL	Память		Байт	100
14. M = 2N – 2R + K – 9	Память			DX	Слово	100
15. M = 3N + R – K + 4	Память		CL	Пам	Байт	100

Примечание. Константы в заданных выражениях представлены в 16-ричной системе счисления.

Приложение 3

Варианты заданий к практическим занятиям и лабораторным работам по разработке программ с применением различных режимов адресации

Разработайте программу, содержащую четыре различных варианта сложения двухбайтных чисел $d \leftarrow d + s$. Операнды источника s и приемника d размещены в памяти по эффективным адресам: источник – $500_{16} + 2N$, приемник – $500_{16} + 2 (N + 1)$, где N – номер варианта в 16-ричной СС. Сформируйте для нее коды команд. Путем выполнения программы проверьте правильность их кодирования.

Варианты программы должны отличаться режимами адресации операндов приемника и источника:

– абсолютная и базовая;

– индексная и косвенная регистровая;

– косвенная регистровая и базовая;

– индексная и индексная.

Варианты отделяйте друг от друга командой NOP.

Приложение 4

Варианты заданий к практическим занятиям и лабораторным работам по разработке циклических программ

1. Перешлите содержимое одной области памяти в другую. Число слов в массиве 10.

2. Перешлите содержимое одной области памяти в другую. Число байт в массиве 12.

3. Сложите 8 двухбайтовых чисел со знаком, размещенных в памяти.

4. Сложите 10 однобайтовых чисел, размещенных в памяти.

5. Сложите 6 однобайтовых BCD-чисел, размещенных в памяти.

6. Определите в массиве из 16 символов, представленных в ASCII-коде, число символов '5' (с кодом 35h).

7. Найдите число положительных двухбайтовых чисел в массиве. Длина массива 8.

8. Найдите в массиве число отрицательных чисел в формате слова**.** Длина массива 8.

9. Произведите инверсию слов, размещенных в массиве из 10 слов.

10. Осуществите преобразование вида $X \leftarrow X + 1$ над словами, размещенными в памяти. Число слов 8.

11. Дана матрица однобайтовых чисел размером 4×5. Составьте программу подсчета числа положительных элементов матрицы.

12. Дана матрица двухбайтовых чисел размером 3×4. Составьте программу подсчета количества отрицательных элементов матрицы.

13. Дано N произвольных чисел. Сформируйте их в массивы отрицательных и положительных чисел.

14. Дана матрица однобайтовых чисел размером 4×5. Подсчитайте число элементов, значения которых < 10.

15. Вычислите Z = X + Y, где X, Y – многобайтовые BCD-числа. Число байт 4 (варианты 11–20 повышенной сложности).

16. Вычислите Z = X + Y, где X, Y – многобайтовые числа в ASCII-формате. Число байт 4.

17. Найдите в массиве двухбайтовых знаковых чисел максимальное число. Длина массива 16 байт.

18. Вычислите Z = X – Y, где X, Y – многобайтовые числа в ASCII-формате. Число байт 4.

19. Найдите Z = X – Y, где X, Y – многобайтовые числа. Число байт 4.

20. Найдите в массиве двухбайтовых знаковых чисел минимальное число. Длина массива 32 байта.

Приложение 5

Варианты заданий по обработке цепочек

1. Скопируйте строку в обратном порядке.
2. Найдите строку (список) с ключевым словом.
3. Найдите в строке символ «точка» и замените его пробелом.
4. Найдите в строке «точку» и после нее все элементы сотрите.
5. Найдите две одинаковые строки и одну из них замените новой.
6. Составьте из трех строк (слов) предложение.
7. Подсчитайте число элементов в строке до символа «ВК».
8. Определите в строке число пробелов.
9. Поменяйте местами две соседние строки.
10. Определите в строке число полей, разделенных пробелами.
11. Распределите слова в предложении в порядке увеличения их размера.
12. Распределите слова в предложении в порядке уменьшения их размера.
13. Определите в строке число символов «а».
14. Найдите в предложении слова, содержащие не менее семи букв.
15. Найдите в предложении слова, содержащие не более восьми букв.

Приложение 6

Таблица сложения 16-ричных цифр (символов)

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Примечание. При сложении двух 16-ричных чисел последовательно складываются соответствующие i-е цифры слагаемых. Получаемая при этом единица переноса прибавляется к сумме цифр слагаемых, получаемой в следующем (i + 1)-м разряде.

Приложение 7

Темы практических занятий и лабораторных работ

1. Представление данных. Машинные коды (ПЗ, приложение 1, п. 1.1–1.5, КР).
2. Выполнение арифметических операций в ЭВМ, формирование физического адреса (ПЗ, приложение 1, п. 1.7, 2.1.2, КР).
3. Изучение состава автоматизированной обучающей системы (АОС) «Микропроцессоры 80x86». Принципы работы с АОС при проведении тестирования (ЛР).
4. Кодирование последовательности команд МП 80x86 для заданного варианта выполняемой операции (ПЗ, затем их проверка в ходе ЛР по теме 6, приложение 2, п. 2.1–2.5, О, Т).
5. Форматы и режимы адресаций МП 80x86. Формирование кодов команд для различных режимов адресации данных (ПЗ, ЛР по аналогии с темой 4, приложение 3, п. 2.6–2.7, О, Т).
6. Элементы программирования на ассемблере IBM PC, разработка фрагментов программ с использованием команд и директив ассемблера (ПЗ, п. 3.1, О, Т).
7. Разработка и отладка линейных программ для создания исполняемых модулей с расширением .COM (ПЗ – разработка программы, ЛР – ее отладка, приложение 3, п. 3.2–3.3, О, Т).

8. Разработка и отладка циклических программ для создания исполняемых модулей с расширением .EXE (ПЗ и ЛР по аналогии с п. 7, приложение 4, варианты 1–10, п. 3.4–3.5, О).
9. Разработка и отладка 32-разрядных программ, их отладка (ЛР, приложение 2, п. 3.6, О).
10. Разработка программы с использованием процедуры, а затем с макроопределением вывода на экран, например, фамилий известных футболистов (ЛР, п. 3.7, О).
11. Разработка программ с использованием цепочечных команд, их отладка (ЛР, приложение 5, п. 3.8, О).

Примечания. 1. Принятые сокращения: ЛР – лабораторная работа, ПЗ – практические занятия, Т – тестирование, КР – контрольная работа, О – отчет, п. – разделы учебного пособия.

2. При выполнении тем 4 и 5 для правильности кодирования команд можно также применять транслятор trans.exe. При этом следует воспользоваться порядком работы (п. 3.3.3), в котором исключить п. 5–7, а в п. 4. вместо команды tasm.exe и расширения .asm задавать trans.exe и .txt – как последовательность кодов команд таблицы 2.2.

Приложение 8

Базовая система команд МП 80X86 (или 8086)

При описании команд используются следующие обозначения:
reg – один из РОН в соответствии со следующей таблицей:

reg	*W = 1*	*W = 0*	*reg*	*W = 1*	*W = 0*	*sreg*	*Сегментный регистр*
000	AX	AL	001	CX	CL	00	ES
010	DX	DL	011	BX	BL	01	CS
100	SP	AH	101	BP	CH	10	SS
110	SI	DH	111	DI	BH	11	DS

D – если D = l, то в reg, если D = 0, то из reg;

W – если W = l, то команда оперирует словом, если W = 0, то – байтом;

S:W = X0 – один байт данных, 01 – два байта данных, 11 – один байт данных, расширенный со знаком до 16 бит;

V – если V = 0, то счетчик равен 1, если V = l, то счетчик в CL;

Z – применяется в цепочечных командах для сравнения с ZF;

data L, data H – младшая и старшая часть (если W = l) непосредственного операнда;

mod – поле режима адресации;

r/m – при mod = 11 интерпретируется как reg, при mod ≠ 11 определяет эффективный адрес (EA) операнда в памяти в соответствии со следующей таблицей, в которой disp L, disp H,L – 8- и 16-битовые смещения в третьем или в третьем-четвертом байтах команды:

mod r/m	00	01	10

000	(BX) + (SI)	(BX) + (SI) + disp L	(BX) + (SI) + disp H,L
001	(BX) + (DI)	(BX) + (DI) + disp L	(BX) + (DI) + disp H,L
010	(BP) + (SI)	(BP) + (SI) + disp L	(BP) + (SI) + disp H,L
011	(BP) + (DI)	(BP) + (DI) + disp L	(BP) + (DI) + disp H,L
100	(SI)	(SI) + disp L	(SI) + disp H,L
101	(DI)	(DI) + disp L	(DI) + disp H,L
110	disp H,L	(BP) + disp L	(BP) + disp H,L
111	(BX)	(BX) + disp L	(BX) + disp H,L

Команды передачи данных

Команды	1-й байт	2-й байт	3-й байт	4-й байт
1	2	3	4	5
MOV – передать				
Регистр или память в/из регистра	100010DW	mod reg r/m		
Непосредственный операнд в регистр или в память	1100011W	mod 000 r/m	data L	data H (w = 1)
Непосредственный операнд в регистр	1011Wreg	data L	data H, (w = 1)	

Продолжение таблицы

1	2	3	4	5
Память в аккумулятор (спецформат)	1010000W	Мл адрес EA	Ст адрес EA	
Аккумулятор в память (спецформат)	1010001W	Мл. адрес EA	Ст адрес EA	
Регистр или память в регистр сегмента	10001110	mod0sreg r/m		
Регистр сегмента в регистр или память	10001100	mod0sreg r/m		
PUSH – вкл. в стек				
Регистр/память	11111111	mod 110 r/m		
Регистр (спецформат)	01010reg			
Регистр сегмента	000sreg110			
POP – исключить				
Регистр/память из стека	10001111	mod 000 r/m		
Регистр (спецформат)	01011reg			
Регистр сегмента	000sreg111			
XCHG – обменять				
Регистр/память с регистром	1000011W	mod reg r/m		
Регистр	10010reg			

с аккумулятором				
IN – ввести из Фиксирован. порта Переменного порта	 1110010W 1110110W	 Порт		
OUT – вывести в Переменный порт Фиксир. порт	 1110111W 1110110W	 Порт		
XLAT – передать байт в AL	11010111			
LEA – загрузить EA в регистр	10001101	mod reg r/m		
LDS – загрузить указатель в DS	11000101	mod reg r/m		
LES – загрузить указатель в ES	11000100	mod reg r/m		
LAHF – загрузить в AH признаки	10011111			
SAHF – запомнить AH в регистре F	10011110			
PUSHF – включить признаки	10011100			
POPF – исключить признаки	10011101			

Команды арифметических операций

Команды	1-й байт	2-й байт	3-й байт	4-й байт
1	2	3	4	5
ADD – сложить Регистр/память с регистром Непосредственный операнд с регистром/памятью Непосредственный операнд с аккум.	 000000DW 100000SW 0000010W	 mod reg r/m mod 000 r/m data L	 data L data H, (w = 1)	 data H (SW = 01)
ADC – сложить с переносом Регистр/память с регистром Непосредственный операнд с регистром/памятью Непосредственный операнд с аккумулятором	 000100DW 100000SW 0001010W	 mod reg r/m mod 010 r/m data L	 data L data H, (w = 1)	 data H (SW = 01)
INC – инкремент Регистра/памяти Регистра	 1111111W 01000reg	 mod 000 r/m		

AAA – ASCII – коррекция сложения	00110111			
DA – десятичная корр. сложения	00100111			
SUB – вычесть Регистр/память с регистром Непосредственный операнд из регистра/памяти Непосредственный операнд из аккумулятора	 001010DW 100000SW 0010110W	 mod reg r/m mod 101 r/m data L	 data L data H, (w = 1)	 data H (SW = 01)
SBB – вычесть с заемом Регистр/память с регистром Непосредственный операнд из регистра/памяти Непосредственный операнд из аккумулятора	 000110DW 100000SW 0010110W	 mod reg r/m mod 011 r/m data L	 data L data H, (w = 1)	 data H (SW = 01)

Продолжение таблицы

1	2	3	4	5
DEC – декремент Регистр/память Регистр	 1111111W 01001reg	 mod 001 r/m		
NEG – изменить знак	1111011W	mod 011 r/m		
CMP – сравнить Регистр/память и регистр Непосредственный операнд с регистром/памятью Непосредственный операнд с аккум.	 001110DW 100000SW 0011110W	 mod reg r/m mod 111 r/m data L	 data L data H, (w = 1)	 data H (SW = 01)
AAS – ASCII коррекция вычитания	00111111			
DAS – десятичная коррекция вычитания	00101111			
IMUL – умножение целых со знаком	1111011W	mod 101 r/m		
MUL – умножение	1111011W	mod 100 r/m		

без знака				
AAM –ASCII коррекция умножения	11010100			
DIV – деление без знака	1111011W	mod 110 r/m		
IDIV – деление со знаком	1111011W	mod 111 r/m		
AAD – ASCII коррекция деления	11010101			
CBW – преобразование байта в слово	10011000			
CWD – преобразование слова в двойное слово	10011001			

Команды логических операций и сдвигов

Команды	1-й байт	2-й байт	3-й байт	4-й байт
1	2	3	4	5
NOT – инверсия	1111011W	mod 010 r/m		
SHL/SAL – сдвиг логич./арифм. влево	110100VW	mod 100 r/m		
SHR – сдвиг логический вправо	110100VW	mod 101 r/m		
SAR – сдвиг арифметический вправо	110100VW	mod 111 r/m		

Продолжение таблицы

1	2	3	4	5
ROL – сдвиг циклический влево	110100VW	mod 000 r/m		
ROR – сдвиг циклический вправо	110100VW	mod 001 r/m		
RCL – сдвиг циклический влево через перенос	110100VW	mod 010 r/m		
RCR – сдвиг цикл. впр. через перенос	110100VW	mod 011r/m		
AND – коньюнкция Регистр/память с регистром Непосредственный операнд с регистром/памятью с аккумулятором	 001000DW 1000000W 0010010W	 mod reg r/m mod 100 r/m data L	 data L data H (w = 1)	 data H (w = 1)

TEST – И без записи результата				
Непосред. операнд с аккумулятором	101010DW	data L	data H	
Регистр/память с регистром	1000010W	mod reg r/m		
Непосредственный операнд с регистром/памятью	1111011W	mod 000r/m	data L	data H (w = 1)
OR – дизьюнкция ИЛИ				
Регистр/память с регистром	000010DW	mod reg r/m		
Непосредственный операнд с регистром/памятью	1000000W	mod 001r/m		
с аккумулятором	0000110W	data L	data H (w = 1)	
XOR – ИЛИ исключающее				
Регистр/память с регистром	001100DW	mod reg r/m		
Непосредственный операнд с регистром/памятью	1000000W	mod 001r/m	data L	data H (w = 1)
с аккумулятором	0011010W	data L	data H (w = 1)	

Команды операций с цепочками данных

Команды	1-й байт
REP – повторить	1111001Z
MOVS – передать байт/слово: [DI] ← [SI]	1010010W
CMPS – сравнить байт/слово: [DI] ← [SI]	1010011W
SCAS – сканировать байт/слово: acc ← [DI]	1010111W
LODS – загрузить байт/слово из [SI] в AL/AX	1010110W
STOS – запомнить байт/слово из AL/AX в [DI]	1010101W

Команды передачи управления (безусловные)

Команды	1-й байт	2-й байт	3-й байт
CALL – ВЫЗОВ			
Прямой в сегменте	11101000	Мл смещение	Ст смещение
Косвенный в сегменте	11111111	mod 010 r/m	
Прямой межсегментный	10011010	Мл смещение	Ст смещение
		сегмент (4-й байт)	сег (5-й байт)
Косвенный межсегментный	11111111	mod 011 r/m	

(mod ≠ 11)			
JMP – БЕЗУСЛ ПЕРЕХОД Прямой в сегменте Прямой в сегменте короткий Косвенный в сегменте Прямой межсегментный Косвенный межсегментный mod ≠ 11	 11101001 11101011 11111111 11101010 11111111	 Мл смещение Смещение mod 100 r/m Мл смещение сегмент (4-й байт) mod 101 r/m	 Ст смещение Ст смещение сегм (5-й байт)
RET – ВОЗВРАТ В сегменте со сложением непосредст операнда с SP В сегменте Межсегментный Межсегментный со сложением непосредственного операнда с SP	 11000010 11000010 11001011 11001010	 data L data L	 data H data H

Команды условных переходов и циклов

Команды	Условие		1-й байт	2-й байт	Условие
JA/JNBE	CF ∪ ZF = 0	>	77	Смещ	Перейти, если выше /не ниже или равно
JNC/JAE/ JNB	CF = 0	>=	73	Смещ	Выше или равно/не ниже (нет переноса)
JC/JB/ JNAE	CF = 1	<	72	Смещ	Ниже/не выше или равно (есть перенос)
JBE/JNA	CF ∪ ZF = 1	<=	76	Смещ	Ниже или равно/не выше
JE/JZ	ZF = 1	=	74	Смещ	Равно/нуль
JG/JNLE	(SF+OF)ZF= =0	>	7F	Смещ	Больше /не меньше и =

Продолжение таблицы

1	2	3	4	5	6
JGE/JNL	SF + OF = 0	>=	7D	Смещ	Больше или =/не меньше
JL/JNGE	SF + OF = 1	<	7C	Смещ	Меньше/не больше или =
JLE/JNG	(SF+OF)ZF= =1	<=	7E	Смещ	Меньше или равно/не равно
JNZ/JNE	ZF = 0		75	Смещ	Не равно/не нуль
JNO	OF = 0		71	Смещ	Нет переполнения
JNP/JPO	PF = 0		7B	Смещ	Нет паритета/паритет нечетный
JNS	SF = 0		79	Смещ	Нет знака (отрицание)
JO	OF = 1		70	Смещ	Есть переполнение
JP/JPE	PF = 1		7A	Смещ	Есть паритет – четный
JS	SF = 1		78	Смещ	Есть знак (отрицательный)
JCNX	(CX) = 0		E3	Смещ	Содержимое регистра CX = 0

LOOP	(CX) ≠ 0		E2	Смещ	Зациклить CX раз
LOOPZ/ LOOPE	(CX) ≠ 0, ZF = 1		E1	Смещ	Зациклить до нуля/равно
LOOPNZ/ LOOPNE	(CX) ≠ 0, ZF = 0		E0	Смещ	Зациклить до нуля/не равно

Примечание. Термины больше, меньше относятся к знаковым числам, представленным в ДК, а выше, ниже – к беззнаковым.

Команды прерывания

Команды	1-й байт	2-й байт
INT – прерывание	11001101	Тип прерывания
INT 3 – прерывание типа 3	11001100	
INTO – прерывание по переполнению	11001110	
IRET – возврат из прерывания	11001111	

Команды управления микропроцессором

Команды	1-й байт	2-й байт	
CLC – сброс переноса	11111000		
CMC – дополнение переноса	11110101		
STC – установка переноса	11111001		
STD – установка направления	11111101		
CLD – сброс направления	11111100		
CLI – сброс прерываний	11111010		
STI – установка прерываний	11111011		
HLT – останов	11110100		
WAIT – ожидание	10011011		
ESC – обращение к сопроцессору	11011xxx	mod yyy m	
LOCK – префикс блокировки	11110000		

Префикс замены сегмента: 001sreg110

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Григорьев, В.Л. Микропроцессор i486. Архитектура и программирование : в 4 кн. – Кн. 1 : Программная архитектура. – М. : ГРАНАЛ, 1993. – 346 с.
2. Григорьев, В.Л. Программирование однокристальных микропроцессоров. – М. : Энергоатомиздат, 1987. – 288 с.
3. Зубков, С.В. Assembler для DOS, Windows и UNIX. – М. : ДМК Пресс, 2000. – 608 с.
4. Локтюхин, В.Н. Основы программирования на ассемблере IBM PC. – М. : Горячая линия–Телеком, 2007. – 88 с.

5. Локтюхин, В.Н. Структура персональных компьютеров : учебное пособие. – Рязань : Сервис, 1999. –140 с.
6. Локтюхин, В.Н. Микропроцессорные системы. Проектирование аппаратных и программных средств : учебное пособие. – Рязань : Сервис, 1998. – 100 с.
7. Локтюхин, В.Н. Микропроцессорные системы. Организация и проектирование интерфейса ввода-вывода : учебное пособие. – Рязань : Сервис, 1998. – 72 с.
8. Локтюхин, В.Н. Микропроцессоры Intel 80x86. Архитектура и программирование : методические указания к лабораторным работам / РГРТА. – Рязань, 2003. – 48 с.
9. Локтюхин, В.Н. Проектирование микропроцессорных систем ввода и преобразования биосигналов / В.Н. Локтюхин, С.И. Мальченко ; РГРТУ. – Рязань, 2007. – 76 с.
10. Майко, Г.В. Ассемблер для IBM PC. – М. : Бизнес–Информ ; Сирин, 1997. – 212 с.
11. Скэнлон, Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера / пер. с англ. – М. : Радио и связь, 1989. – 136 с.
12. Уокерли, Дж. Архитектура и программирование микро-ЭВМ : в 2 кн. / пер. с англ. – М. : Мир, 1984. – Кн. 1. – 486 с.
13. Финогенов, К.Г. Использование языка Ассемблера : учебное пособие для вузов. – М. : Горячая линия–Телеком, 2004. – 438 с.

ОГЛАВЛЕНИЕ

Для заметок

Учебное издание

Локтюхин Виктор Николаевич

АРХИТЕКТУРА КОМПЬЮТЕРА

в 2 книгах

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА АССЕМБЛЕРЕ IBM PC

Книга 2

Учебное пособие

Редактор *О.С. Верещагина*
Технический редактор *В.Н. Локтюхин*

Подписано в печать 28.07.08. Поз. № 74. Бумага офсетная. Формат 60x84 1/16.
Гарнитура Times New Roman. Печать трафаретная.
Усл. печ. л. 7,9. Уч.-изд. л. 10,1. Тираж 500 экз. Заказ №

Государственное образовательное учреждение высшего профессионального образования
«Рязанский государственный университет имени С.А. Есенина»,
390000, г. Рязань, ул. Свободы, 46

Редакционно-издательский центр РГУ имени С.А. Есенина
390023, г. Рязань, ул. Урицкого, 22